OpenShift Container Platform 3.10

クラスターの設定

OpenShift Container Platform 3.10 のインストールおよび設定

Last Updated: 2018-10-26

OpenShift Container Platform 3.10 クラスターの設定

OpenShift Container Platform 3.10 のインストールおよび設定

法律上の通知

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

『OpenShift のインストールと設定』では、ご使用の環境で OpenShift をインストールし、設定するための基本事項を説明します。扱われるトピックを参照して、OpenShift の稼働に必要な一度だけ実行するタスク (one-time task) を実行してください。

目次

第1章 概要 **18**

第2章 レジストリーのセットアップ **19**
2.1. レジストリーの概要 19
2.1.1. レジストリーについて 19
2.1.2. 統合レジストリーまたはスタンドアロンレジストリー 19
2.1.3. Red Hat Quay レジストリー 19
2.2. 既存クラスターへのレジストリーのデプロイ 19
2.2.1. 概要 19
2.2.2. レジストリーのデプロイ 20
2.2.3. レジストリーの DaemonSet としてのデプロイ 20
2.2.4. レジストリーのコンピュートリソース 20
2.2.5. レジストリーのストレージ 20
2.2.5.1. 実稼働環境での使用 21
2.2.5.1.1. Amazon S3 のストレージのバックエンドとしての使用 21
2.2.5.2. 非実稼働環境での使用 22
2.2.6. レジストリーコンソールの有効化 23
2.2.6.1. レジストリーコンソールのデプロイ 23
2.2.6.2. レジストリーコンソールのセキュリティー保護 24
2.2.6.3. レジストリーコンソールのトラブルシューティング 26
2.2.6.3.1. デバッグモード 26
2.2.6.3.2. SSL 証明書パスの表示 26
2.3. レジストリーへのアクセス 26
2.3.1. ログの表示 26
2.3.2. ファイルストレージ 27
2.3.3. レジストリーへの直接アクセス 29
2.3.3.1. ユーザーの前提条件 29
2.3.3.2. レジストリーへのログイン 29
2.3.3.3. イメージのプッシュとプル 30
2.3.4. レジストリーメトリクスへのアクセス 31
2.4. レジストリーのセキュリティー保護および公開 32
2.4.1. 概要 32
2.4.2. レジストリーを手動でセキュリティー保護する 33
2.4.3. セキュアなレジストリーの手動による公開 36
2.4.4. 非セキュアなレジストリーの手動での公開 38
2.5. レジストリー設定の拡張 39
2.5.1. レジストリー IP アドレスの維持 40
2.5.2. Docker レジストリーのホワイトリスト化 40
2.5.3. レジストリーのホスト名の設定 41
2.5.4. レジストリー設定の上書き 41
2.5.5. レジストリー設定の参照 43
2.5.5.1. ログ 44
2.5.5.2. フック 44
2.5.5.3. ストレージ 44
2.5.5.4. 認証 45
2.5.5.5. ミドルウェア 45
2.5.5.5.1. CloudFront ミドルウェア 46
2.5.5.5.2. ミドルウェア設定オプションの上書き 47
2.5.5.5.3. イメージのプルスルー 48
2.5.5.5.4. Manifest Schema v2 サポート 49
2.5.5.6. OpenShift 50

2.5.5.7. レポート (Reporting) 51
2.5.5.8. HTTP 51
2.5.5.9. 通知 51
2.5.5.10. Redis 52
2.5.5.11. 健全性 52
2.5.5.12. プロキシー 52
2.5.5.13. キャッシュ 52
2.6. 既知の問題 52
2.6.1. 概要 53
2.6.2. レジストリーのプルスルーに伴う同時ビルド 53
2.6.3. 共有 NFS ボリュームとスケーリングされたレジストリーの使用時のイメージのプッシュエラー 53
2.6.4. 内部で管理されたイメージのプルに失敗し「見つかりません (not found)」のエラーが表示される 54
2.6.5. S3 ストレージでのイメージのプッシュが失敗し「500 内部サーバーエラー (500 Internal Server Error)」と表示される 54
2.6.6. イメージのプルーニングの失敗 55

第3章 ルーターのセットアップ 56
3.1. ルーターの概要 56
3.1.1. ルーターについて 56
3.1.2. ルーターのサービスアカウント 56
3.1.2.1. ラベルにアクセスするためのパーミッション 56
3.2. デフォルト HAPROXY ルーターの使用 57
3.2.1. 概要 57
3.2.2. ルーターの作成 58
3.2.3. その他の基本ルーターコマンド 58
3.2.4. ルートを特定のルーターに絞り込む 59
3.2.5. HAProxy Strict SNI 60
3.2.6. TLS 暗号化スイート 60
3.2.7. 高可用性ルーター 60
3.2.8. ルーターサービスポートのカスタマイズ 60
3.2.9. 複数ルーターの使用 61
3.2.10. デプロイメント設定へのノードセレクターの追加 61
3.2.11. ルーターシャードの使用 62
3.2.11.1. ルーターシャードの作成 64
3.2.11.2. ルーターシャードの変更 66
3.2.12. ルーターのホスト名の検索 67
3.2.13. デフォルトのルーティングサブドメインのカスタマイズ 68
3.2.14. カスタムルーティングサブドメインへのルートホスト名の強制 69
3.2.15. ワイルドカード証明書の使用 69
3.2.16. 証明書を手動で再デプロイする 70
3.2.17. セキュリティー保護されたルートの使用 71
3.2.18. (サブドメインの) ワイルドカードルートの使用 72
3.2.19. コンテナーネットワークスタックの使用 77
3.2.20. ルーターメトリクスの公開 78
3.2.21. 再起動時の接続失敗の防止 80
3.2.22. 大規模クラスターの ARP キャッシュのチューニング 81
3.2.23. DDoS 攻撃からの保護 82
3.3. カスタマイズされた HAPROXY ルーターのデプロイ 83
3.3.1. 概要 83
3.3.2. ルーター設定テンプレートの取得 83
3.3.3. ルーター設定テンプレートの変更 84
3.3.3.1. 背景 84
3.3.3.2. Go テンプレートアクション 84

3.3.3.3. ルーターが提供する情報 85
3.3.3.4. アノテーション 89
3.3.3.5. 環境変数 90
3.3.3.6. 使用例 90
3.3.4. ConfigMap を使用してルーター設定テンプレートを置き換える 92
3.3.5. Stick Table の使用 93
3.3.6. ルーターの再ビルド 95
3.4. HAPROXY ルーターを設定して PROXY プロトコルを使用する 95
3.4.1. 概要 95
3.4.2. PROXY プロトコルを使用する理由 96
3.4.3. PROXY プロトコルの使用 96
3.5. F5 ルータープラグインの使用 100
3.5.1. 概要 100
3.5.2. 前提条件とサポート容易性 101
3.5.2.1. 仮想サーバーの設定 102
3.5.3. F5 ルーターのデプロイ 103
3.5.4. F5 ルーターのパーティションパス 104
3.5.5. F5 ネイティブ統合のセットアップ 105

第4章 RED HAT CLOUDFORMS のデプロイ 107
4.1. {MGMT-APP} の OPENSHIFT CONTAINER PLATFORM へのデプロイ 107
4.1.1. はじめに 107
4.2. RED HAT CLOUDFORMS を OPENSHIFT CONTAINER PLATFORM で使用するための要件 108
4.3. ロール変数の設定 109
4.3.1. 概要 109
4.3.2. 一般的な変数 109
4.3.3. テンプレートパラメーターのカスタマイズ 110
4.3.4. データベース変数 110
4.3.4.1. コンテナー化された (Pod 化された) データベース 110
4.3.4.2. 外部データベース 110
4.3.5. ストレージクラス変数 111
4.3.5.1. NFS (デフォルト) 112
4.3.5.2. NFS (外部) 112
4.3.5.3. クラウドプロバイダー 113
4.3.5.4. 事前設定 (詳細) 113
4.4. インストーラーの実行 113
4.4.1. OpenShift Container Platform のインストール時またはインストール後の Red Hat CloudForms のデプロイ 113
4.4.2. インベントリーファイルの例 114
4.4.2.1. すべてのデフォルト 114
4.4.2.2. 外部 NFS ストレージ 114
4.4.2.3. PV サイズの上書き 115
4.4.2.4. メモリー要件の上書き 115
4.4.2.5. 外部 PostgreSQL データベース 115
4.5. コンテナープロバイダー統合の有効化 115
4.5.1. 単一コンテナープロバイダーの追加 115
4.5.1.1. 手動の追加 116
4.5.1.2. 自動の追加 116
4.5.2. 複数のコンテナープロバイダー 116
4.5.2.1. スクリプトの作成 116
4.5.2.1.1. 例 117
4.5.2.2. Playbook の実行 118
4.5.3. プロバイダーの更新 118

4.6. RED HAT CLOUDFORMS のアンインストール 118
4.6.1. アンインストール Playbook の実行 118
4.6.2. トラブルシューティング 119

第5章 マスターとノードの設定 120
5.1. インストールの依存関係 120
5.2. マスターとノードの設定 120
5.3. ANSIBLE を使用した設定の変更 120
5.3.1. htpasswd コマンドの使用 122
5.4. 手動による設定変更 123
5.5. マスター設定ファイル 124
5.5.1. 受付制御の設定 124
5.5.2. アセットの設定 125
5.5.3. 認証と承認の設定 127
5.5.4. コントローラーの設定 127
5.5.5. etcd の設定 128
5.5.6. 付与の設定 129
5.5.7. イメージ設定 130
5.5.8. イメージポリシーの設定 130
5.5.9. Kubernetes のマスター設定 131
5.5.10. ネットワーク設定 132
5.5.11. OAuth 認証設定 133
5.5.12. プロジェクトの設定 135
5.5.13. スケジューラーの設定 137
5.5.14. セキュリティーアロケーターの設定 137
5.5.15. サービスアカウントの設定 137
5.5.16. 提供情報の設定 139
5.5.17. ボリュームの設定 140
5.5.18. 基本的な監査 141
5.5.19. 高度な監査 142
5.5.20. etcd の TLS 暗号の指定 145
5.6. ノード設定ファイル 146
5.6.1. Pod とノードの設定 148
5.6.2. Docker の設定 148
5.6.3. ローカルストレージの設定 149
5.6.4. Docker 1.9 以降を使用したイメージの並行プル 149
5.7. パスワードおよびその他の機密データ 149
5.8. 新規設定ファイルの作成 150
5.9. 設定ファイルの使用によるサーバーの起動 151
5.10. ロギングレベルの設定 152
5.11. マスターおよびノードサービスの再起動 156

第6章 OPENSHIFT ANSIBLE BROKER の設定 157
6.1. 概要 157
6.2. OPENSHIFT ANSIBLE BROKER 設定の変更 158
6.3. レジストリー設定 158
6.3.1. 実稼働または開発 160
6.3.2. レジストリー認証情報の保存 160
6.3.3. モックレジストリー 162
6.3.4. Dockerhub レジストリー 162
6.3.5. APB のフィルタリング 162
6.3.6. ローカルの OpenShift Container レジストリー 164
6.3.7. Red Hat Container Catalog レジストリー 164

6.3.8. Red Hat Connect Partner Registry 164
6.3.9. 複数のレジストリー 165
6.4. ブローカー認証 165
6.4.1. Basic 認証 165
6.4.1.1. デプロイメントテンプレートおよびシークレット 166
6.4.1.2. サービスカタログおよびブローカー通信の設定 167
6.4.2. Bearer 認証 168
6.4.2.1. デプロイメントテンプレートおよびシークレット 168
6.4.2.2. サービスカタログおよびブローカー通信の設定 169
6.5. DAO 設定 169
6.6. ログ設定 169
6.7. OPENSHIFT 設定 170
6.8. ブローカー設定 170
6.9. シークレット設定 171
6.10. プロキシー環境での実行 172
6.10.1. レジストリーアダプターのホワイトリスト 172
6.10.2. Ansible を使用したプロキシー環境でのブローカーの設定 172
6.10.3. プロキシー環境でのブローカーの手動設定 173
6.10.4. Pod でのプロキシー環境変数の設定 174

第7章 ホストの既存クラスターへの追加 175
7.1. ホストの追加 175
手順 175
7.2. ETCD ホストの既存クラスターへの追加 177
7.3. 共存する ETCD での既存のマスターの置き換え 178
7.4. ノードの移行 179

第8章 デフォルトのイメージストリームとテンプレートの読み込み 181
8.1. 概要 181
8.2. サブスクリプションタイプ別のサービス 181
8.2.1. OpenShift Container Platform サブスクリプション 181
8.2.2. xPaaS ミドルウェアアドオンサブスクリプション 182
8.3. 作業を開始する前に 182
8.4. 前提条件 182
8.5. OPENSHIFT CONTAINER PLATFORM イメージのイメージストリームの作成 183
8.6. XPAAS ミドルウェアイメージのイメージストリームの作成 183
8.7. データベースサービステンプレートの作成 184
8.8. インスタントアプリケーションおよびクイックスタートテンプレートの作成 184
8.9. 次のステップ 185

第9章 カスタム証明書の設定 186
9.1. 概要 186
9.2. インストール時のカスタム証明書の設定 186
9.3. WEB コンソールまたは CLI 用カスタム証明書の設定 186
9.4. カスタムマスターホスト証明書の設定 188
9.5. デフォルトルーター用のカスタムワイルドカード証明書の設定 189
9.6. イメージレジストリー用のカスタム証明書の設定 190
9.7. ロードバランサー用のカスタム証明書の設定 191
9.8. カスタム証明書の変更およびクラスターへの組み込み 192
9.8.1. カスタムマスター証明書のの変更およびクラスターへの組み込み 192
9.8.2. カスタムルーター証明書の変更およびクラスターへの組み込み 193
9.9. の他のコンポーネントでのカスタム証明書の使用 193

第10章 証明書の再デプロイ 194

10.1. 概要 194
10.2. 証明書の有効期限のチェック 194
10.2.1. ロール変数 194
10.2.2. 証明書の有効期限切れ Playbook の実行 195
他のサンプル Playbook 196
10.2.3. 出力形式 196
HTML レポート 196
JSON レポート 197
10.3. 証明書の再デプロイ 197
10.3.1. 現行の OpenShift Container Platform および etcd CA を使用したすべての証明書の再デプロイ 198
10.3.2. 新規またはカスタムの OpenShift Container Platform CA の再デプロイ 198
10.3.3. 新規 etcd CA の再デプロイ 199
10.3.4. マスター証明書のみの再デプロイ 200
10.3.5. etcd 証明書のみの再デプロイ 200
10.3.6. ノード証明書のみの再デプロイ 201
10.3.7. レジストリー証明書またはルーター証明書のみの再デプロイ 201
10.3.7.1. レジストリー証明書のみの再デプロイ 201
10.3.7.2. ルーター証明書のみの再デプロイ 201
10.3.8. カスタムのレジストリー証明書またはルーター証明書の再デプロイ 201
10.3.8.1. 手動によるレジストリー証明書の再デプロイ 201
10.3.8.2. 手動によるルーター証明書の再デプロイ 203

第11章 認証およびユーザーエージェントの設定 206
11.1. 概要 206
11.2. アイデンティティープロバイダーパラメーター 206
11.3. アイデンティティープロバイダーの設定 208
11.3.1. Ansible を使用したアイデンティティープロバイダーの設定 208
11.3.2. マスター設定ファイルでのアイデンティティープロバイダーの設定 209
11.3.3. アイデンティティープロバイダーまたはメソッドの設定 209
11.3.3.1. lookup マッピング方法を使用する場合のユーザーの手動プロビジョニング 209
11.3.4. Allow All 210
11.3.5. Deny All 211
11.3.6. HTPasswd 211
11.3.7. Keystone 213
11.3.7.1. マスターでの認証の設定 213
11.3.7.2. Keystone 認証を使用するユーザーの作成 215
11.3.7.3. ユーザーの確認 215
11.3.8. LDAP 認証 216
11.3.9. Basic 認証 (リモート) 218
11.3.9.1. マスターでの認証の設定 220
11.3.9.2. トラブルシューティング 222
11.3.10. 要求ヘッダー 222
要求ヘッダーを使用した Apache 認証 225
前提条件のインストール 226
Apache の設定 227
その他の mod_auth_form 要件 229
ユーザーの作成 229
マスターの設定 229
サービスの再起動 229
設定の確認 229
11.3.11. GitHub 230
11.3.12. GitHub でのアプリケーションの登録 230
11.3.13. マスターでの認証の設定 231

11.3.14. GitHub 認証を持つユーザーの作成 233
11.3.15. ユーザーの確認 233
11.3.16. GitLab 233
11.3.17. Google 235
11.3.18. OpenID Connect 236
11.4. トークンオプション 239
11.5. 付与オプション 240
11.6. セッションオプション 240
11.7. ユーザーエージェントによる CLI バージョンの不一致の防止 241

第12章 グループと LDAP の同期 244
12.1. 概要 244
12.2. LDAP 同期の設定 244
12.2.1. LDAP クライアント設定 244
12.2.2. LDAP クエリー定義 245
12.2.3. ユーザー定義の名前マッピング 246
12.3. LDAP 同期の実行 246
12.4. グループのプルーニングジョブの実行 247
12.5. 同期の例 247
12.5.1. RFC 2307 スキーマの使用によるグループの同期 248
12.5.1.1. ユーザー定義の名前マッピングに関する RFC2307 250
12.5.2. ユーザー定義のエラートレランスに関する RFC 2307 の使用によるグループの同期 252
12.5.3. Active Directory の使用によるグループの同期 254
12.5.4. 拡張された Active Directory の使用によるグループの同期 256
12.6. ネスト化されたメンバーシップ同期の例 259
12.7. LDAP 同期設定の仕様 262
12.7.1. v1.LDAPSyncConfig 262
12.7.2. v1.StringSource 264
12.7.3. v1.LDAPQuery 265
12.7.4. v1.RFC2307Config 266
12.7.5. v1.ActiveDirectoryConfig 268
12.7.6. v1.AugmentedActiveDirectoryConfig 268

第13章 LDAP フェイルオーバーの設定 270
13.1. 基本リモート認証設定の前提条件 270
13.2. 証明書の生成およびリモート BASIC 認証サーバーとの共有 270
13.3. SSSD での LDAP フェイルオーバーの設定 271
13.4. APACHE での SSSD の使用の設定 273
13.5. OPENSHIFT CONTAINER PLATFORM が SSSD を基本リモート認証サーバーとして使用する設定 277

第14章 SDN の設定 278
14.1. 概要 278
14.2. 利用可能な SDN プロバイダー 278
VMware NSX-T (™) の OpenShift Container Platform へのインストール 278
14.3. ANSIBLE を使用した POD ネットワークの設定 278
14.4. マスターでの POD ネットワークの設定 279
14.5. ノードでの POD ネットワークの設定 280
14.6. サービスネットワークの拡張 281
14.7. SDN プラグイン間の移行 282
14.7.1. ovs-multitenant から ovs-networkpolicy への移行 282
14.8. クラスターネットワークへの外部アクセス 283
14.9. FLANNEL の使用 284

第15章 NUAGE SDN の設定 287

15.1. NUAGE SDN と OPENSHIFT CONTAINER PLATFORM 287
15.2. 開発者のワークフロー 287
15.3. オペレーションワークフロー 287
15.4. インストール 287

第16章 KURYR SDN の設定 290
16.1. KURYR SDN および OPENSHIFT CONTAINER PLATFORM 290
16.2. インストール 290
16.3. 検証 292

第17章 AMAZON WEB サービス (AWS) の設定 293
17.1. 概要 293
17.2. パーミッション 293
17.3. セキュリティーグループの設定 294
17.3.1. 検出された IP アドレスとホスト名の上書き 294
17.4. AWS 変数の設定 295
17.5. AWS についての OPENSHIFT CONTAINER PLATFORM マスターの設定 296
17.5.1. Ansible を使用した AWS についての OpenShift Container Platform の設定 296
17.5.2. 手動による AWS についての OpenShift Container Platform マスターの設定 297
17.5.3. 手動による AWS についての OpenShift Container Platform ノードの設定 297
17.6. キーと値のアクセスペアの設定 298
17.7. 設定変更の適用 298
17.8. クラスターに対する AWS のラベリング 299
17.8.1. タグを必要とするリソース 299
17.8.2. 既存クラスターへのタグ付け 299

第18章 OPENSTACK の設定 301
18.1. 概要 301
18.2. パーミッション 301
18.3. セキュリティーグループの設定 301
18.4. OPENSTACK 変数の設定 302
18.5. OPENSTACK についての OPENSHIFT CONTAINER PLATFORM マスターの設定 302
18.5.1. 手動による OpenStack についての OpenShift Container Platform マスターの設定 303
18.5.2. 手動による OpenStack についての OpenShift Container Platform ノードの設定 304
18.6. 設定変更の適用 304
18.6.1. 動的に作成した OpenStack PV のゾーンラベルの設定 305

第19章 GOOGLE COMPUTE ENGINE の設定 306
19.1. 作業を開始する前の注意事項 306
19.1.1. Google Cloud Platform での認証設定 306
19.1.2. Google Compute Engine オブジェクト 307
19.2. GCE 向けの OPENSHIFT CONTAINER PLATFORM の設定 310
19.2.1. オプション 1: Ansible を使用した GCP 向けの OpenShift Container Platform の設定 311
19.2.2. オプション 2: GCE 用の OpenShift Container Platform の手動設定 312
19.2.2.1. GCE 向けのマスターホストの手動設定 312
19.2.2.2. GCE 向けのノードホストの手動設定 313
19.2.3. GCP 向けの OpenShift Container Platform レジストリーの設定 314
19.2.3.1. GCP 向けの OpenShift Container Platform レジストリーの手動設定 315
19.2.3.1.1. レジストリーが GCP オブジェクトストレージを使用していることを確認します。 316
19.2.4. OpenShift Container Platform が GCP ストレージを使用するための設定 318
19.3. サービスとしての GCP 外部のロードバランサー使用 319

第20章 AZURE の設定 322
20.1. 概要 322

20.2. パーミッション 322
20.3. 前提条件 322
20.4. AZURE 設定ファイル 322
20.4.1. クラスターのインストール中の Azure の設定 323
20.4.2. Azure 向けのマスターホストの手動設定 324
20.4.3. Azure 向けのノードホストの手動設定 324
20.4.4. 手動による設定変更の適用 325

第21章 VMWARE VSPHERE の設定 326
21.1. 概要 326
21.2. VMWARE VSPHERE クラウドプロバイダーの有効化 326
21.3. VMWARE VSPHERE 設定ファイル 328
21.4. マスターの設定 329
21.5. ノードの設定 329
21.6. 設定変更の適用 330
21.7. 永続ボリュームのバックアップ 330

第22章 ローカルボリュームの設定 332
22.1. 概要 332
22.2. ローカルボリュームのマウント 332
22.3. ローカルプロビジョナーの設定 333
22.4. ローカルプロビジョナーのデプロイ 334
22.5. 新規デバイスの追加 335

第23章 永続ストレージの設定 340
23.1. 概要 340
23.2. NFS を使用した永続ストレージ 340
23.2.1. 概要 340
23.2.2. プロビジョニング 341
23.2.3. ディスククォータの実施 342
23.2.4. NFS ボリュームのセキュリティー 342
23.2.4.1. グループ ID 343
23.2.4.2. ユーザー ID 344
23.2.4.3. SELinux 345
23.2.4.4. エクスポート設定 346
23.2.5. リソースの回収 346
23.2.6. 自動化 347
23.2.7. その他の設定とトラブルシューティング 348
23.3. RED HAT GLUSTER STORAGE を使用する永続ストレージ 348
23.3.1. 概要 348
23.3.1.1. コンバージドモード 348
23.3.1.2. インデペンデントモード 349
23.3.1.3. スタンドアロンの Red Hat Gluster Storage 349
23.3.1.4. GlusterFS ボリューム 350
23.3.1.5. gluster-block ボリューム 350
23.3.1.6. Gluster S3 Storage 351
23.3.2. 考慮事項 351
23.3.2.1. ソフトウェア要件 351
23.3.2.2. ハードウェア要件 352
23.3.2.3. ストレージのサイジング 352
23.3.2.4. ボリューム操作の動作 353
23.3.2.5. ボリュームのセキュリティー 353
23.3.2.5.1. POSIX パーミッション 354
23.3.2.5.2. SELinux 354

23.3.3. サポート要件 355
23.3.4. インストール 355
23.3.4.1. インデペンデントモード: Red Hat Gluster Storage ノードのインストール 355
23.3.4.2. インストーラーの使用 355
23.3.4.2.1. 例: 基本的なコンバージドモードのインストール 357
23.3.4.2.2. 例: 基本的なインデペンデントモードのインストール 358
23.3.4.2.3. 例: 統合 OpenShift Container レジストリーを使用するコンバージドモード 360
23.3.4.2.4. 例: OpenShift ロギングおよびメトリクス用のコンバージドモード 361
23.3.4.2.5. 例: アプリケーション、レジストリー、ロギングおよびメトリクス用のコンバージドモード 363
23.3.4.2.6. 例: アプリケーション、レジストリー、ロギングおよびメトリクス用のインデペンデントモード 365
23.3.5. コンバージドモードのアンイストール 368
23.3.6. プロビジョニング 368
23.3.6.1. 静的プロビジョニング 368
23.3.6.2. 動的プロビジョニング 372
23.4. OPENSTACK CINDER を使用した永続ストレージ 373
23.4.1. 概要 373
23.4.2. Cinder PV のプロビジョニング 373
23.4.2.1. 永続ボリュームの作成 373
23.4.2.2. Cinder の PV 形式 375
23.4.2.3. Cinder ボリュームのセキュリティー 375
23.5. CEPH RADOS ブロックデバイス (RBD) を使用した永続ストレージ 376
23.5.1. 概要 376
23.5.2. プロビジョニング 376
23.5.2.1. Ceph シークレットの作成 377
23.5.2.2. 永続ボリュームの作成 377
23.5.3. Ceph ボリュームのセキュリティー 379
23.6. AWS ELASTIC BLOCK STORE を使用した永続ストレージ 380
23.6.1. 概要 380
23.6.2. プロビジョニング 380
23.6.2.1. 永続ボリュームの作成 381
23.6.2.2. ボリュームのフォーマット 382
23.6.2.3. ノード上の EBS ボリュームの最大数 382
23.7. GCE PERSISTENT DISK を使用した永続ストレージ 382
23.7.1. 概要 382
23.7.2. プロビジョニング 383
23.7.2.1. 永続ボリュームの作成 383
23.7.2.2. ボリュームのフォーマット 384
23.8. ISCSI を使用した永続ストレージ 384
23.8.1. 概要 384
23.8.2. プロビジョニング 385
23.8.2.1. ディスククォータの実施 385
23.8.2.2. iSCSI ボリュームのセキュリティー 385
23.8.2.3. iSCSI のマルチパス化 386
23.8.2.4. iSCSI のカスタムイニシエーター IQN 386
23.9. ファイバーチャネルを使用した永続ストレージ 387
23.9.1. 概要 387
23.9.2. プロビジョニング 387
23.9.2.1. ディスククォータの実施 388
23.9.2.2. ファイバーチャネルボリュームのセキュリティー 388
23.10. AZURE DISK を使用した永続ストレージ 388
23.10.1. 概要 388

23.10.2. 前提条件 388
23.10.3. プロビジョニング 389
23.10.4. Azure Disk でのリージョンクラウドの設定 389
23.10.4.1. 永続ボリュームの作成 389
23.10.4.2. ボリュームのフォーマット 391
23.11. AZURE FILE を使用した永続ストレージ 391
23.11.1. 概要 391
23.11.2. 作業を開始する前の注意事項 391
23.11.3. Azure File でのリージョンクラウドの設定 393
23.11.4. PV の作成 393
23.11.5. Azure Storage Account シークレットの作成 393
23.12. FLEXVOLUME プラグインを使用した永続ストレージ 394
23.12.1. 概要 395
23.12.2. FlexVolume ドライバー 395
23.12.2.1. マスター実行の割り当て/割り当て解除がある FlexVolume ドライバー 396
23.12.2.2. マスター実行の割り当て/割り当て解除がない FlexVolume ドライバー 399
23.12.3. FlexVolume ドライバーのインストール 399
23.12.4. FlexVolume ドライバーを使用したストレージの使用 400
23.13. 永続ストレージ用の VMWARE VSPHERE ボリューム 401
23.13.1. 概要 401
前提条件 401
23.13.2. VMware vSphere ボリュームのプロビジョニング 402
23.13.2.1. 永続ボリュームの作成 402
23.13.2.2. VMware vSphere ボリュームのフォーマット 403
23.14. ローカルボリュームを使用した永続ストレージ 403
23.14.1. 概要 403
23.14.2. プロビジョニング 404
23.14.3. ローカルの Persistent Volume Claim (永続ボリューム要求) の作成 404
23.14.4. 機能のステータス 404
23.15. CONTAINER STORAGE INTERFACE (CSI) を使用した永続ストレージ 405
23.15.1. 概要 405
23.15.2. アーキテクチャー 406
23.15.2.1. 外部の CSI コントローラー 406
23.15.2.2. CSI ドライバーの DaemonSet 407
23.15.3. デプロイメント例 407
23.15.4. 動的プロビジョニング 412
23.15.5. 使用法 412
23.16. OPENSTACK MANILA を使用した永続ストレージ 413
23.16.1. 概要 413
23.16.2. インストールと設定 413
23.16.2.1. 外部プロビジョナーの起動 413
23.16.3. 使用法 416
23.17. 動的プロビジョニングとストレージクラスの作成 417
23.17.1. 概要 417
23.17.2. 動的にプロビジョニングされる使用可能なプラグイン 417
23.17.3. StorageClass の定義 419
23.17.3.1. 基本 StorageClass オブジェクトの定義 419
23.17.3.2. StorageClass のアノテーション 419
23.17.3.3. OpenStack Cinder オブジェクトの定義 420
23.17.3.4. AWS ElasticBlockStore (EBS) オブジェクトの定義 420
23.17.3.5. GCE PersistentDisk (gcePD) オブジェクトの定義 421
23.17.3.6. GlusterFS オブジェクトの定義 422
23.17.3.7. Ceph RBD オブジェクトの定義 423

23.17.3.8. Trident オブジェクトの定義 424
23.17.3.9. VMWare vSphere オブジェクトの定義 424
23.17.3.10. Azure File のオブジェクト定義 424
23.17.3.11. Azure Disk オブジェクト定義 425
23.17.4. デフォルト StorageClass の変更 427
23.17.5. 追加情報とサンプル 427
23.18. ボリュームのセキュリティー 428
23.18.1. 概要 428
23.18.2. SCC、デフォルト値および許可される範囲 428
23.18.3. 補助グループ 432
23.18.4. fsGroup 435
23.18.5. ユーザー ID 437
23.18.6. SELinux オプション 439
23.19. セレクターとラベルによるボリュームのバインディング 441
23.19.1. 概要 441
23.19.2. 必要になる状況 441
23.19.3. デプロイメント 441
23.19.3.1. 前提条件 441
23.19.3.2. 永続ボリュームと要求の定義 442
23.19.3.3. 永続ボリュームと要求のデプロイ 443
23.20. コントローラー管理の割り当ておよび割り当て解除 443
23.20.1. 概要 443
23.20.2. 割り当て/割り当て解除の管理元の判別 443
23.20.3. コントローラー管理の割り当て/割り当て解除を有効にするノードの設定 444
23.21. 永続ボリュームスナップショット 444
23.21.1. 概要 444
23.21.2. 機能 445
23.21.3. インストールと設定 445
23.21.3.1. 外部のコントローラーおよびプロビジョナーの起動 445
23.21.3.2. スナップショットユーザーの管理 448
23.21.4. ボリュームスナップショットとボリュームスナップショットデータのライフサイクル 449
23.21.4.1. Persistent Volume Claim (永続ボリューム要求) と永続ボリューム 449
23.21.4.1.1. スナップショットプロモーター 449
23.21.4.2. スナップショットの作成 449
23.21.4.3. スナップショットの復元 451
23.21.4.4. スナップショットの削除 451

第24章 永続ストレージの例 452
24.1. 概要 452
24.2. 2 つの PERSISTENT VOLUME CLAIM (永続ボリューム要求) 間での NFS マウントの共有 452
24.2.1. 概要 452
24.2.2. 永続ボリュームの作成 452
24.2.3. Persistent Volume Claim (永続ボリューム要求) の作成 453
24.2.4. NFS ボリュームアクセスの確保 454
24.2.5. Pod の作成 455
24.2.6. 同じ PVC を参照する追加 Pod の作成 459
24.3. CEPH RBD を使用した詳細例 461
24.3.1. 概要 461
24.3.2. ceph-common パッケージのインストール 461
24.3.3. Ceph シークレットの作成 462
24.3.4. 永続ボリュームの作成 462
24.3.5. Persistent Volume Claim (永続ボリューム要求) の作成 463
24.3.6. Pod の作成 464

24.3.7. グループ ID と所有者 ID の定義 (オプション) 465
24.3.8. ceph-user-secret をプロジェクトのデフォルトとして設定する方法 466
24.4. 動的プロビジョニングでの CEPH RBD の使用 466
24.4.1. 概要 466
24.4.2. 動的ボリューム用プールの作成 467
24.4.3. 動的な永続ストレージでの既存の Ceph クラスターの使用 467
24.4.4. ceph-user-secret をプロジェクトのデフォルトとして設定する方法 470
24.5. GLUSTERFS を使用する詳細例 471
24.5.1. 概要 471
24.5.2. 前提条件 472
24.5.3. 静的プロビジョニング 472
24.5.4. ストレージの使用 476
24.6. GLUSTERFS を動的プロビジョニングに使用する詳細例 478
24.6.1. 概要 478
24.6.2. 前提条件 478
24.6.3. 動的プロビジョニング 479
24.6.4. ストレージの使用 480
24.7. 特権付き POD へのボリュームのマウント 482
24.7.1. 概要 482
24.7.2. 前提条件 482
24.7.3. 永続ボリュームの作成 482
24.7.4. 通常ユーザーの作成 483
24.7.5. Persistent Volume Claim (永続ボリューム要求) の作成 483
24.7.6. 設定の検証 484
24.7.6.1. Pod の SCC の確認 484
24.7.6.2. マウントの検証 484
24.8. 統合 OPENSHIFT CONTAINER レジストリーから GLUSTERFS への切り替え 485
24.8.1. 概要 485
24.8.2. 前提条件 485
24.8.3. GlusterFS PersistentVolumeClaim の手動プロビジョニング 485
24.8.4. PersistentVolumeClaim のレジストリーへの割り当て 489
24.9. ラベルによる永続ボリュームのバインド 489
24.9.1. 概要 489
24.9.1.1. 前提条件 490
24.9.2. 仕様の定義 490
24.9.2.1. ラベルのある永続ボリューム 490
24.9.2.2. セレクターのある Persistent Volume Claim (永続ボリューム要求) 491
24.9.2.3. ボリュームエンドポイント 491
24.9.2.4. PV、PVC、およびエンドポイントのデプロイ 492
24.10. ストレージクラスを使用した動的プロビジョニング 492
24.10.1. 概要 492
24.10.2. シナリオ 1: 2 種類の StorageClass を持つ基本的な動的プロビジョニング 492
24.10.3. シナリオ 2: クラスターにおけるデフォルトの StorageClass の動作を有効にする方法 495
24.11. 既存のレガシーストレージに対するストレージクラスの使用 499
24.11.1. 概要 499
24.11.1.1. シナリオ 1: レガシーデータを含む既存の永続ボリュームに StorageClass をリンクさせる 500
24.12. AZURE BLOB STORAGE での統合 DOCKER レジストリーの設定 502
24.12.1. 概要 502
24.12.2. 作業を開始する前に 502
24.12.3. レジストリー設定の上書き 502

第25章 一時ストレージの設定 505
25.1. 概要 505

25.2. 一時ストレージの有効化 505

第26章 HTTP プロキシーの使用 507
26.1. 概要 507
26.2. NO_PROXY の設定 507
26.3. ホストでのプロキシーの設定 508
26.4. ANSIBLE を使用したホストでのプロキシー設定 509
26.5. DOCKER PULL のプロキシー設定 509
26.6. プロキシーの背後での MAVEN の使用 510
26.7. S2I ビルドでのプロキシーの設定 510
26.8. デフォルトテンプレートでのプロキシーの設定 511
26.9. POD でのプロキシー環境変数の設定 511
26.10. GIT リポジトリーのアクセス 511

第27章 グローバルビルドのデフォルトと上書きの設定 513
27.1. 概要 513
27.2. グローバルビルドのデフォルトの設定 514
27.2.1. Ansible を使用したグローバルビルドのデフォルトの設定 514
27.2.2. グローバルビルドのデフォルトの手動設定 515
27.3. グローバルビルドの上書きの設定 516
27.3.1. Ansible を使用したグローバルビルドの上書きの設定 516
27.3.2. グローバルビルドの上書きの手動設定 517

第28章 PIPELINE 実行の設定 519
28.1. 概要 519
28.2. OPENSHIFT JENKINS CLIENT プラグイン 520
28.3. OPENSHIFT JENKINS の同期プラグイン 520

第29章 ルートのタイムアウトの設定 522

第30章 ネイティブのコンテナールーティングの設定 523
30.1. ネットワークの概要 523
30.2. ネイティブのコンテナールーティングの設定 523
30.3. コンテナーネットワーク用のノードのセットアップ 524
30.4. コンテナーネットワーク用のルーターのセットアップ 524

第31章 エッジロードバランサーからのルーティング 525
31.1. 概要 525
31.2. ロードバランサーの SDN への追加 525
31.3. RAMP ノードを使用したトンネルの確立 525
31.3.1. 高可用性を備えた Ramp ノードの設定 528

第32章 コンテナーログの集計 530
32.1. 概要 530
32.2. デプロイ前の設定 530
32.3. ロギング ANSIBLE 変数の指定 531
32.4. EFK スタックのデプロイ 540
32.5. デプロイメントの概要と調整 541
32.5.1. Ops クラスター 541
32.5.2. Elasticsearch 541
32.5.3. Fluentd 546
32.5.4. Kibana 550
32.5.5. Curator 551
32.5.5.1. Curator 設定の作成 553
32.6. クリーンアップ 554

32.7. KIBANA のトラブルシューティング 554
32.8. 外部 ELASTICSEARCH インスタンスへのログの送信 556
32.9. 外部 SYSLOG サーバーへのログの送信 557
32.10. ELASTICSEARCH 管理操作の実行 560
32.11. 集計されたロギングのドライバーの変更 560
32.12. ELASTICSEARCH の手動ロールアウト 562
32.12.1. Elasticsearch ローリングクラスター再起動の実行 562
32.12.2. Elasticsearch フルクラスター再起動の実行 563

第33章 集計ロギングのサイジングに関するガイドライン 565
33.1. 概要 565
33.2. インストール 565
33.2.1. 大規模クラスター 567
33.3. SYSTEMD-JOURNALD と RSYSLOG 568
33.4. EFK ロギングのスケールアップ 568
33.5. ストレージに関する考慮事項 569

第34章 クラスターメトリクスの有効化 571
34.1. 概要 571
34.2. 作業を開始する前に 571
34.3. メトリクスプロジェクト 571
34.4. メトリクスデータストレージ 572
34.4.1. 永続ストレージ 572
34.4.2. クラスターメトリクスの容量計画 572
既知の問題と制限 574
34.4.3. 非永続ストレージ 574
34.5. メトリクス ANSIBLE ロール 575
34.5.1. メトリクス Ansible 変数の指定 575
34.5.2. シークレットの使用 578
34.5.2.1. ユーザー固有の証明書の提供 578
34.6. メトリクスコンポーネントのデプロイ 579
34.6.1. メトリクスの診断 580
34.7. メトリクスのパブリック URL の設定 580
34.8. HAWKULAR METRICS への直接アクセス 581
34.8.1. OpenShift Container Platform プロジェクトと Hawkular テナント 581
34.8.2. 承認 581
34.9. OPENSHIFT CONTAINER PLATFORM クラスターメトリクス POD のスケーリング 581
34.10. クリーンアップ 581
34.11. OPENSHIFT CONTAINER PLATFORM 上の PROMETHEUS 582
34.11.1. Prometheus ロール変数の設定 582
34.11.2. Ansible インストーラーを使用した Prometheus のデプロイ 583
34.11.2.1. Prometheus をデプロイする他の方法 584
34.11.2.2. Prometheus Web UI へのアクセス 584
34.11.2.3. Prometheus での OpenShift Container Platform の設定 585
34.11.3. Prometheus 経由の OpenShift Container Platform メトリクス 586
34.11.3.1. 現行のメトリクス 586
34.11.4. Prometheus のアンデプロイ 588

第35章 WEB コンソールのカスタマイズ 589
35.1. 概要 589
35.2. 拡張スクリプトとスタイルシートの読み込み 589
35.2.1. 拡張プロパティーの設定 590
35.3. 外部ロギングソリューションの拡張オプション 591
35.4. ガイド付きツアーのカスタマイズと無効化 591

35.5. ドキュメントリンクのカスタマイズ 591
35.6. ロゴのカスタマイズ 592
35.7. メンバーシップのホワイトリストのカスタマイズ 592
35.8. ドキュメントへのリンクの変更 592
35.9. CLI をダウンロードするリンクの追加または変更 593
35.9.1. About ページのカスタマイズ 593
35.10. ナビゲーションメニューの設定 594
35.10.1. トップナビゲーションドロップダウンメニュー 594
35.10.2. アプリケーションランチャー 595
35.10.3. システムステータスバッジ 595
35.10.4. プロジェクトの左ナビゲーション 596
35.11. 主要アプリケーションの設定 598
35.12. カタログカテゴリーの設定 599
35.13. クォータ通知メッセージの設定 600
35.14. CREATE FROM URL NAMESPACE ホワイトリストの設定 601
35.15. COPY LOGIN コマンドの無効化 601
35.15.1. ワイルドカードルートの有効化 601
35.16. ログインページのカスタマイズ 601
35.16.1. 使用例 602
35.17. OAUTH エラーページのカスタマイズ 602
35.18. ログアウト URL の変更 602
35.19. ANSIBLE による WEB コンソールカスタマイズの設定 603

第36章 外部永続ボリュームプロビジョナーのデプロイ 605
36.1. 概要 605
36.2. 作業を開始する前に 605
36.2.1. 外部プロビジョナーの Ansible ロール 605
36.2.2. 外部プロビジョナーの Ansible 変数 605
36.2.3. AWS EFS プロビジョナーの Ansible 変数 606
36.3. プロビジョナーのデプロイ 607
36.3.1. AWS EFS プロビジョナーのデプロイ 607
36.3.1.1. AWS EFS オブジェクト定義 607
36.4. クリーンアップ 608

第1章 概要

本書では、OpenShift Container Platform クラスターのインストール後に利用できる追加の設定オプションについて説明します。

第2章 レジストリーのセットアップ

2.1. レジストリーの概要

2.1.1. レジストリーについて

OpenShift Container Platform は、ソースコードからコンテナーイメージをビルドし、デプロイし、そのライフサイクルを管理することができます。これを可能にするために、OpenShift Container Platform は、内部の統合 Docker レジストリーを提供します。このレジストリーは、ローカルでイメージを管理するために OpenShift Container Platform 環境にデプロイすることができます。

2.1.2. 統合レジストリーまたはスタンドアロンレジストリー

完全な OpenShift Container Platform クラスターの初回インストール時に、レジストリーはインストールプロセスで自動的にデプロイされている可能性があります。自動的にデプロイされていなかった場合やレジストリー設定のカスタマイズが追加で必要になる場合には、「既存クラスターへのレジストリーのデプロイ」を参照してください。

OpenShift Container Platform のレジストリーは、完全な OpenShift Container Platform クラスターの統合部分として実行されるようにデプロイできる一方で、スタンドアロンのコンテナーイメージレジストリーとして個別にインストールすることも可能です。

スタンドアロンレジストリーをインストールするには、「スタンドアロンレジストリーのインストール」の手順に従ってください。このインストールパスは、レジストリーと専用の Web コンソールを実行するオールインワンのクラスターをデプロイします。

2.1.3. Red Hat Quay レジストリー

エンタープライズ向けの高品質なコンテナーイメージレジストリーを必要とされる場合、Red Hat Quay をホストされたサービスおよびソフトウェアのどちらとしてもご利用いただけます。これは独自のデータセンターやクラウド環境にインストールしてご利用いただけます。Red Hat Quay の高度なレジストリーには、geo レプリケーション、イメージのスキャニング、およびイメージのロールバック機能が含まれます。

Quay.io サイトにアクセスして独自のホストレベルの Quay レジストリーのアカウントをセットアップできます。その後は、Quay チュートリアルをご利用いただき、Quay レジストリーにログインしてからイメージの管理を開始できます。または、独自の Red Hat Quay レジストリーをセットアップする方法についての詳細は、「Getting Started with Red Hat Quay」を参照してください。

現時点では、リモートのコンテナーイメージレジストリーにアクセスするのと同じ方法で Red Hat Quay レジストリーに OpenShift からアクセスできます。Red Hat Quay にアクセスするための認証情報をセキュリティー保護されたレジストリーとしてセットアップする方法については、「Allowing Pods to Reference Images from Other Secured Registries」を参照してください。

2.2. 既存クラスターへのレジストリーのデプロイ

2.2.1. 概要

OpenShift Container Platform クラスターの初回インストール時に統合レジストリーが事前に自動的にデプロイされなかった場合や、正常に実行されず、既存のクラスターに再デプロイする必要がある場合は、以下のセクションで新規レジストリーをデプロイするためのオプションを参照してください。

注記

スタンドアロンレジストリーをインストールしていない場合、このトピックは不要になります。

2.2.2. レジストリーのデプロイ

統合 Docker レジストリーをデプロイするには、クラスター管理者権限を持つユーザーとして **oc adm registry** コマンドを使用します。以下は例になります。

```
$ oc adm registry --config=/etc/origin/master/admin.kubeconfig \ 1
    --service-account=registry \ 2
    --images='registry.access.redhat.com/openshift3/ose-
${component}:${version}' 3
```

1. **--config** は、クラスター管理者用の CLI 設定ファイルへのパスです。
2. **--service-account** は、レジストリーの Pod の実行に使用されるサービスアカウントです。
3. OpenShift Container Platform の適切なイメージをプルするために必要です。

これでサービスとデプロイメント設定が作成されます。これらは共に **docker-registry** と呼ばれます。正常にデプロイされると、Pod が **docker-registry-1-cpty9** などの名前で作成されます。

レジストリーの作成時に指定できるオプションの詳細の一覧を表示するには、以下を実行します。

```
$ oc adm registry --help
```

--fs-group の値は、レジストリーが使用している SCC (通常は制限付き SCC) によって許可されている必要があります。

2.2.3. レジストリーの DaemonSet としてのデプロイ

レジストリーを **--daemonset** オプションを使用して **DaemonSet**としてデプロイするには、 **oc adm registry** コマンドを使用します。

DaemonSet は、ノードの作成時に、指定した Pod のコピーがノードに含まれていることを確認します。ノードが削除されると、Pod はガベージコレクションされます。

DaemonSetに関する詳細は、「Daemonset の使用」を参照してください。

2.2.4. レジストリーのコンピュートリソース

デフォルトで、レジストリーはコンピュートリソースの要求や制限に関する設定がない状態で作成されます。実稼働環境では、レジストリーのデプロイメント設定を更新してレジストリー Pod のリソース要求や制限を設定しておくことが強く推奨されます。設定しない場合、レジストリー Pod は **BestEffort** Pod と判断されます。

要求や制限の設定に関する詳細は、「コンピュートリソース」を参照してください。

2.2.5. レジストリーのストレージ

レジストリーには、コンテナーのイメージとメタデータが格納されています。Pod をレジストリーと共に単純にデプロイする場合、Pod がすでに存在する場合に破棄される一時ボリュームが使用されます。この場合、誰かがビルドしたりレジストリーにプッシュしたりしたイメージは消えてしまいます。

以下のセクションでは、サポートされているレジストリーのストレージドライバーの一覧を示しています。詳細は、Docker レジストリーのドキュメントを参照してください。

以下の一覧には、レジストリーの設定ファイルで設定を行う必要があるストレージドライバーが含まれます。

- ファイルシステム。ファイルシステムはデフォルトですので、設定の必要はありません。
- S3。詳細は、CloudFront の設定のドキュメントを参照してください。
- OpenStack Swift
- Google Cloud Storage (GCS)
- Microsoft Azure
- Aliyun OSS

レジストリーの一般的なストレージ設定オプションはサポートされています。詳細は、Docker レジストリーのドキュメントを参照してください。

以下のストレージオプションは、ファイルシステムドライバーで設定する必要があります。

- GlusterFS ストレージ
- Ceph Rados ブロックデバイス

注記

サポートされている永続ストレージドライバーの詳細は、「永続ストレージの設定」および「永続ストレージの例」を参照してください。

2.2.5.1. 実稼働環境での使用

実稼働環境での使用の場合には、リモートボリュームを割り当てるか、または各自が選択した永続ストレージ方法を定義して使用します。

たとえば、既存の Persistent Volume Claim (永続ボリューム要求) を使用するには、以下を実行します。

```
$ oc volume deploymentconfigs/docker-registry --add --name=registry-
storage -t pvc \
    --claim-name=<pvc_name> --overwrite
```

注記

スケーリングされたレジストリーを共有 NFS ボリュームで使用している場合は、既知の問題を参照してください。

2.2.5.1.1. Amazon S3 のストレージのバックエンドとしての使用

Amazon Simple Storage Service のストレージを内部 Docker レジストリーと共に使用する方法もあります。Amazon Simple Storage Service は AWS マネジメントコンソールで管理できるセキュアなクラウドストレージです。これを使用するには、レジストリーの設定ファイルを手動で編集し、レジストリー Pod にマウントする必要があります。ただし、設定を開始する前にアップストリームの推奨手順を確認してください。

デフォルトの YAML 設定ファイルをベースとして取得し、**storage** セクションの **filesystem** エントリーを、以下のように **s3** エントリーに置き換えます。これにより、ストレージのセクションは以下のようになります。

```
storage:
  cache:
    layerinfo: inmemory
  delete:
    enabled: true
  s3:
    accesskey: awsaccesskey 1
    secretkey: awssecretkey 2
    region: us-west-1
    regionendpoint: http://myobjects.local
    bucket: bucketname
    encrypt: true
    keyid: mykeyid
    secure: true
    v4auth: false
    chunksize: 5242880
    rootdirectory: /s3/object/name/prefix
```

1. Amazon のアクセスキーに置き換えてください。
2. Amazon のシークレットキーに置き換えてください。

s3 のすべての設定オプションは、アップストリームのドライバー参照ドキュメントに記載されています。

レジストリー設定を上書きすると、設定ファイルを Pod にマウントするための追加の手順に進みます。

警告

レジストリーが S3 ストレージのバックエンドで実行されると、問題が報告されます。

2.2.5.2. 非実稼働環境での使用

非実稼働環境の場合、**--mount-host=<path>** オプションを使って、永続ストレージに使用するレジストリーのディレクトリーを指定します。次に、レジストリーのボリュームがホストのマウントとして、指定された **<path>** に作成されます。

重要

--mount-host オプションは、レジストリーのコンテナーが実行されているノードからディレクトリーをマウントします。**docker-registry** デプロイメント設定をスケールアップすると、レジストリー Pod とコンテナーが別々のノードで実行されので、それぞれ独自のローカルストレージを使用したレジストリーコンテナーが 2 つ以上作成される可能性があります。これは予期しない動作を生じさせます。その後に繰り返される同一イメージのプル要求が最終的に到達するコンテナーによっては必ずしも成功しない場合があるためです。

--mount-host オプションを使用する場合には、レジストリーコンテナーを特権モードで実行する必要があります。**--mount-host** を指定すると、自動的に特権モードが有効になります。ただしデフォルトでは、すべての Pod が特権付きコンテナー をデフォルトで実行できる訳ではありません。それでもこのオプションを使用する必要がある場合は、レジストリーを作成してから、レジストリーがインストール時に作成された **registry** サービスアカウントを使用するように指定してください。

```
$ oc adm registry --service-account=registry \
    --config=/etc/origin/master/admin.kubeconfig \
    --images='registry.access.redhat.com/openshift3/ose-
${component}:${version}' \
    --mount-host=<path>
```

重要

Docker レジストリー Pod は、ユーザー **1001** として実行されます。このユーザーは、ホストのディレクトリーへの書き込みができなければなりません。したがって、以下のコマンドでディレクトリーの所有権をユーザー ID **1001** に変更する必要がある場合があります。

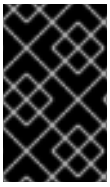
```
$ sudo chown 1001:root <path>
```

2.2.6. レジストリーコンソールの有効化

OpenShift Container Platform は、統合レジストリーへの Web ベースのインターフェースを提供します。このレジストリーコンソールは、イメージの参照と管理を行うためのオプションのコンポーネントであり、Pod として実行されるステートレスサービスとしてデプロイされます。

注記

OpenShift Container Platform をスタンドアロンレジストリーとしてインストールした場合、レジストリーコンソールはインストール時にすでにデプロイされ、そのセキュリティーが自動的に保護されています。



重要

Cockpit がすでに実行されている場合、レジストリーコンソールとのポート競合 (デフォルトでは9090) を避けるために、次に進む前にこれをシャットダウンする必要があります。

2.2.6.1. レジストリーコンソールのデプロイ

重要

最初にレジストリーを公開しておく必要があります。

1. **デフォルト**のプロジェクトにパススルールートを作成します。このルートは、以下の手順でレジストリーコンソールのアプリケーションを作成する際に必要になります。

```
$ oc create route passthrough --service registry-console \
    --port registry-console \
    -n default
```

2. レジストリーコンソールのアプリケーションをデプロイします。**<openshift_oauth_url>** は、OpenShift Container Platform OAuth プロバイダー の URL に置き換えます。通常これはマスターになります。

```
$ oc new-app -n default --template=registry-console \
    -p
OPENSHIFT_OAUTH_PROVIDER_URL="https://<openshift_oauth_url>:8443" \
    -p REGISTRY_HOST=$(oc get route docker-registry -n default --
template='{{ .spec.host }}') \
    -p COCKPIT_KUBE_URL=$(oc get route registry-console -n default -
-template='https://{{ .spec.host }}')
```

注記

レジストリーコンソールへのログインの試行時にリダイレクト URL が間違っていた場合は、**oc get oauthclients** で OAuth クライアントを確認してください。

3. 最後に、Web ブラウザーでこのルート URI を使用しているコンソールを表示します。

2.2.6.2. レジストリーコンソールのセキュリティー保護

デフォルトでは、レジストリーコンソールは、レジストリーコンソールのデプロイの手順ごとに手動でデプロイされる場合に自己署名 TLS 証明書を生成します。詳細は、「レジストリーコンソールのトラブルシューティング」を参照してください。

組織の署名済み証明書をシークレットボリュームとして追加する際には、以下の手順に従ってください。ここでは、証明書が **oc** クライアントホストで利用可能であることを前提としています。

1. 証明書とキーを含む **.cert** ファイルを作成します。以下を使用してファイルをフォーマットします。

   - サーバー証明書と中間証明機関用の **BEGIN CERTIFICATE** ブロック 1 つまたは複数。
   - キーの **BEGIN PRIVATE KEY** または同種のものを含むブロック。キーは暗号化することができません。
     以下は例を示しています。

```
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAnBgNV
BAoMIGI0OGE2NGNkNmMwNTQ1YThhZTgxOTEzZDE5YmJjMmRjMRIwEAYDVQQDDAls
...
-----END CERTIFICATE-----
```

```
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAnBgNV
BAoMIGI0OGE2NGNkNmMwNTQ1YThhZTgxOTEzZDE5YmJjMmRjMRIwEAYDVQQDDAls
...
-----END CERTIFICATE-----
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCyOJ5garOYw0sm
8TBCDSqQ/H1awGMzDYdB11xuHHsxYS2VepPMzMzryHR137I4dGFLhvdTvJUH8lUS
...
-----END PRIVATE KEY-----
```

- セキュリティーの保護されたレジストリーには、以下の SAN (サブジェクトの別名: Subject Alternative Names) 一覧が含まれているはずです。
  - 2 つのサービスのホスト名。
    以下は例を示しています。

    ```
    docker-registry.default.svc.cluster.local
    docker-registry.default.svc
    ```

  - サービス IP アドレス。
    以下は例を示しています。

    ```
    172.30.124.220
    ```

    以下のコマンドを使って Docker レジストリーのサービス IP アドレスを取得します。

    ```
    oc get service docker-registry --
    template='{{.spec.clusterIP}}'
    ```

  - パブリックホスト名。
    以下は例を示しています。

    ```
    docker-registry-default.apps.example.com
    ```

    以下のコマンドを使って Docker レジストリーのパブリックホスト名を取得します。

    ```
    oc get route docker-registry --template '{{.spec.host}}'
    ```

    たとえば、サーバー証明書には以下のような SAN の詳細が記載されるはずです。

    ```
    X509v3 Subject Alternative Name:
                    DNS:docker-registry-public.openshift.com,
    DNS:docker-registry.default.svc, DNS:docker-
    registry.default.svc.cluster.local, DNS:172.30.2.98, IP
    Address:172.30.2.98
    ```

    レジストリーコンソールは、証明書を **/etc/cockpit/ws-certs.d** ディレクトリーから読み込み、拡張子 **.cert** が付いたファイルをアルファベット順で (最後から) 使用します。したがって **.cert** ファイルには、 OpenSSL スタイルでフォーマットされた PEM ブロックが少なくとも 2 つ含まれている必要があります。

証明書がない場合には、 **openssl** コマンドで自己署名証明書を作成し、**0-self-signed.cert** ファイルに保存します。

2. シークレットを作成します。

```
$ oc create secret generic console-secret \
    --from-file=/path/to/console.cert
```

3. このシークレットを **registry-console** デプロイメント設定に追加します。

```
$ oc volume dc/registry-console --add --type=secret \
    --secret-name=console-secret -m /etc/cockpit/ws-certs.d
```

これにより、レジストリーコンソールの新規デプロイメントがトリガーされ、署名済み証明書が組み込まれます。

2.2.6.3. レジストリーコンソールのトラブルシューティング

2.2.6.3.1. デバッグモード

レジストリーコンソールのデバッグモードは環境変数を使用して有効にされます。以下のコマンドは、レジストリーコンソールをデバッグモードで再デプロイします。

```
$ oc set env dc registry-console G_MESSAGES_DEBUG=cockpit-ws,cockpit-
wrapper
```

デバッグモードを有効にすると、より詳細なログがレジストリコンソールの Pod ログに表示されます。

2.2.6.3.2. SSL 証明書パスの表示

レジストリーコンソールが使用している証明書を確認するには、コマンドをコンソール Pod から実行します。

1. **デフォルト** のプロジェクトに Pod を一覧表示して、レジストリーコンソールの Pod 名を検索します。

```
$ oc get pods -n default
NAME                       READY     STATUS    RESTARTS   AGE
registry-console-1-rssrw   1/1       Running   0          1d
```

2. 直前のコマンドで取得した Pod 名を使って、**cockpit-ws** プロセスが使用している証明書パスを取得します。以下は、自動生成された証明書を使用しているコンソールの例です。

```
$ oc exec registry-console-1-rssrw remotectl certificate
certificate: /etc/cockpit/ws-certs.d/0-self-signed.cert
```

2.3. レジストリーへのアクセス

2.3.1. ログの表示

Docker レジストリーのログを表示するには、デプロイメント設定で **oc logs** コマンドを使用します。

```
$ oc logs dc/docker-registry
2015-05-01T19:48:36.300593110Z time="2015-05-01T19:48:36Z" level=info
msg="version=v2.0.0+unknown"
2015-05-01T19:48:36.303294724Z time="2015-05-01T19:48:36Z" level=info
msg="redis not configured" instance.id=9ed6c43d-23ee-453f-9a4b-
031fea646002
2015-05-01T19:48:36.303422845Z time="2015-05-01T19:48:36Z" level=info
msg="using inmemory layerinfo cache" instance.id=9ed6c43d-23ee-453f-9a4b-
031fea646002
2015-05-01T19:48:36.303433991Z time="2015-05-01T19:48:36Z" level=info
msg="Using OpenShift Auth handler"
2015-05-01T19:48:36.303439084Z time="2015-05-01T19:48:36Z" level=info
msg="listening on :5000" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
```

2.3.2. ファイルストレージ

タグとイメージメタデータは OpenShift Container Platform に格納されますが、レジストリーは、レイヤーと署名データを **/registry** にあるレジストリーコンテナーにマウントされているボリュームに格納します。**oc exec** は特権付きコンテナーでは機能しないため、レジストリーの内容を確認するには、レジストリー Pod のコンテナーを格納しているノードに対して SSH を手動で実行し、コンテナー自体で **docker exec** を実行します。

1. 現在の Pod を一覧表示し、Docker レジストリーの Pod 名を検索します。

   ```
   # oc get pods
   ```

   次に、**oc describe** を使用して、コンテナーを実行しているノードのホスト名を検索します。

   ```
   # oc describe pod <pod_name>
   ```

2. 必要なノードにログインします。

   ```
   #ssh node.example.com
   ```

3. ノードホストのデフォルトプロジェクトから実行中のコンテナーを一覧表示し、Docker レジストリーのコンテナー ID を特定します。

   ```
   #docker ps --filter = name = registry_docker-registry。* _ default_
   ```

4. **oc rsh** コマンドを使用してレジストリーの内容を一覧表示します。

   ```
   # oc rsh dc/docker-registry find /registry
   /registry/docker
   /registry/docker/registry
   /registry/docker/registry/v2
   /registry/docker/registry/v2/blobs (1)
   /registry/docker/registry/v2/blobs/sha256
   /registry/docker/registry/v2/blobs/sha256/ed
   /registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3
   d83c648c24f92cece5f89d95ac6c34ce751111810
   /registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3
   d83c648c24f92cece5f89d95ac6c34ce751111810/data (2)
   ```

```
/registry/docker/registry/v2/blobs/sha256/a3
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd
84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd
84406680ae93d633cb16422d00e8a7c22955b46d4/data
/registry/docker/registry/v2/blobs/sha256/f7
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f
259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f
259582bb33502bdb0fcf5011e03c60577c4284845/data
/registry/docker/registry/v2/repositories (3)
/registry/docker/registry/v2/repositories/p1
/registry/docker/registry/v2/repositories/p1/pause (4)
/registry/docker/registry/v2/repositories/p1/pause/_manifests
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures (5)
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24f92cece5f
89d95ac6c34ce751111810
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisi
ons/sha256/e9a2ac6418981897b399d3709f1b4a6d2723cd38a4909215ce2752a5c
068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24f92cece5f
89d95ac6c34ce751111810/link (6)
/registry/docker/registry/v2/repositories/p1/pause/_uploads (7)
/registry/docker/registry/v2/repositories/p1/pause/_layers (8)
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3
ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3
ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4/link
(9)
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f7
2a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f7
2a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845/link
```

1. このディレクトリーには、すべてのレイヤーと署名が Blob として格納されます。
2. このファイルには、Blob の内容が含まれます。
3. このディレクトリーには、すべてのイメージリポジトリーが格納されます。
4. このディレクトリーは単一イメージリポジトリーの **p1**/**pause** 用です。

5 このディレクトリーには、特定のイメージマニフェストリビジョンの署名が含まれます。

6 このファイルには、Blob (署名データを含む) への参照が含まれます。

7 このディレクトリーには、指定されたリポジトリーに対して現在アップロードされステージングされているすべてのレイヤーが含まれます。

8 このディレクトリーには、このリポジトリーが参照するすべてのレイヤーへのリンクが含まれます。

9 このファイルには、イメージを介してこのリポジトリーにリンクされている特定のレイヤーへの参照が含まれます。

2.3.3. レジストリーへの直接アクセス

さらに高度な使用方法として、レジストリーに直接アクセスし、**docker** コマンドを起動することが可能です。これにより、**docker push** や **docker pull** などの操作で直接イメージをプッシュするか、または統合レジストリーからイメージをプルすることができます。これを実行するには、**docker login** コマンドを使ってレジストリーにログインしている必要があります。実行できる操作は、以下のセクションで説明されているようにユーザーが持つ権限によって異なります。

2.3.3.1. ユーザーの前提条件

レジストリーに直接アクセスするには、使用するユーザーが、使用目的に応じて以下の前提条件を満たしている必要があります。

- 直接アクセスするには、ユーザーは選択するアイデンティティープロバイダーの通常ユーザーである必要があります。通常ユーザーは、レジストリーへのログインに必要なアクセストークンを生成できます。**system:admin** などのシステムユーザーはアクセストークンを取得できないため、レジストリーに直接アクセスすることはできません。
  たとえば **HTPASSWD** 認証を使用している場合は、以下のコマンドを使用してこれを作成することができます。

```
# htpasswd /etc/origin/openshift-htpasswd <user_name>
```

- **docker pull** コマンドを使用する場合などにイメージをプルするには、ユーザーに **registry-viewer** ロールがなければなりません。このロールを追加するには、以下を実行します。

```
$ oc policy add-role-to-user registry-viewer <user_name>
```

- イメージの書き出しやプッシュを実行するには (**docker push** コマンドを使用する場合など)、ユーザーに **registry-editor** ロールが必要です。このロールを追加するには、以下を実行します。

```
$ oc policy add-role-to-user registry-editor <user_name>
```

ユーザーパーミッションに関する詳細は、「ロールバインディングの管理」を参照してください。

2.3.3.2. レジストリーへのログイン

注記

ユーザーが、レジストリーに直接アクセスできるための前提条件を満たしていることを確認してください。

レジストリーに直接ログインするには、以下を実行します。

1. OpenShift Container Platform に**通常ユーザー**としてログインしていることを確認します。

```
$ oc login
```

2. アクセストークンを使用して Docker レジストリーにログインします。

```
docker login -u openshift -p $(oc whoami -t) <registry_ip>:<port>
```

注記

ユーザー名には、任意の値を指定でき、トークンには必要な情報が全て含まれます。コロンが含まれるユーザー名を指定すると、ログインに失敗します。

2.3.3.3. イメージのプッシュとプル

レジストリーにログインすると、レジストリーに **docker pull** および **docker push** 操作を実行できます。

重要

任意のイメージをプルできますが、**system:registry** ロールを追加している場合は、各自のプロジェクトにあるレジストリーにのみイメージをプッシュすることができます。

以下の例では、以下を使用します。

コンポーネント	値
<registry_ip>	**172.30.124.220**
<port>	**5000**
<project>	**openshift**
<image>	**busybox**
<tag>	省略 (デフォルトは **latest**)

1. 任意のイメージをプルします。

```
$ docker pull docker.io/busybox
```

2. 新規イメージに **<registry_ip>:<port>/<project>/<image>** 形式でタグ付けします。プロジェクト名は、イメージを正しくレジストリーに配置し、これに後でアクセスできるようにするには OpenShift Container Platform のプル仕様に**必ず**表示されている必要があります。

```
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
```

注記

通常ユーザーには、指定されたプロジェクトの **system:image-builder** ロールが必要です。このロールにより、ユーザーはイメージの書き出しやプッシュを実行できます。このロールが設定されていない場合には以下の手順の **docker push** が失敗します。新規プロジェクトを作成して **busybox** イメージをプッシュしてテストすることができます。

3. 新しくタグ付けされたイメージをレジストリーにプッシュします。

```
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest:
sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403ca
b55
```

2.3.4. レジストリーメトリクスへのアクセス

OpenShift Container レジストリーは、Prometheus メトリクス のエンドポイントを提供します。Prometheus はスタンドアロンのオープンソースシステムのモニタリングおよびアラートツールキットです。

メトリクスはレジストリーのエンドポイントの **/extensions/v2/metrics** パスに公開されます。ただし、このルートは最初に有効にされている必要があります。有効化の方法についてはレジストリー設定の拡張を参照してください。

以下は、メトリクスクエリーの簡単な例です。

```
$ curl -s -u <user>:<secret> \ 1
    http://172.30.30.30:5000/extensions/v2/metrics | grep openshift | head
-n 10

# HELP openshift_build_info A metric with a constant '1' value labeled by
major, minor, git commit & git version from which OpenShift was built.
# TYPE openshift_build_info gauge
openshift_build_info{gitCommit="67275e1",gitVersion="v3.6.0-
alpha.1+67275e1-803",major="3",minor="6+"} 1
# HELP openshift_registry_request_duration_seconds Request latency summary
in microseconds for each operation
# TYPE openshift_registry_request_duration_seconds summary
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.5"} 0
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.9"} 0
openshift_registry_request_duration_seconds{name="test/origin-
pod",operation="blobstore.create",quantile="0.99"} 0
openshift_registry_request_duration_seconds_sum{name="test/origin-
```

```
pod",operation="blobstore.create"} 0
openshift_registry_request_duration_seconds_count{name="test/origin-
pod",operation="blobstore.create"} 5
```

1 **<user>** は任意ですが、**<secret>** はレジストリー設定で指定された値と一致していなければなりません。

メトリクスにアクセスするための別の方法は、クラスターロールの使用です。エンドポイントはここでも有効にする必要がありますが、**<secret>** を指定する必要はありません。設定ファイルのメトリクスに対応する部分は以下のようになります。

```
openshift:
  version: 1.0
  metrics:
    enabled: true
...
```

メトリクスにアクセスするために必要なクラスターロールがない場合、これを作成する必要があります。

```
$ cat <<EOF |
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-scraper
rules:
- apiGroups:
  - image.openshift.io
  resources:
  - registry/metrics
  verbs:
  - get
EOF
oc create -f -
```

このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user prometheus-scraper <username>
```

高度なクエリーと推奨されるビジュアライザーについては、アップストリームの Prometheus ドキュメントを参照してください。

2.4. レジストリーのセキュリティー保護および公開

2.4.1. 概要

デフォルトでは、OpenShift Container Platform レジストリーは、TLS 経由でトラフィックを提供できるようクラスターのインストール時にセキュリティー保護されます。サービスを外部に公開するために、パススルールートもデフォルトで作成されます。

何らかの理由でレジストリーが保護されていないか、または公開されていない場合は、これらを手動で実行する方法について以下のセクションを参照してください。

2.4.2. レジストリーを手動でセキュリティー保護する

レジストリーを手動でセキュリティー保護して TLS 経由でトラフィックを処理するには、以下を実行します。

1. レジストリーをデプロイします。
2. レジストリーのサービス IP とポートを取得します。

```
$ oc get svc/docker-registry
NAME              LABELS
SELECTOR                  IP(S)            PORT(S)
docker-registry   docker-registry=default                    docker-
registry=default   172.30.124.220   5000/TCP
```

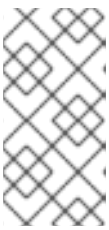
3. 既存のサーバー証明書を使用するか、指定の IP およびホスト名で有効で、指定の CA により署名されたサーバー証明書およびキーを作成します。レジストリーのサービス IP と **docker-registry.default.svc.cluster.local** ホスト名のサーバー証明書を作成するには、Ansible ホストインベントリーファイル (デフォルトでは **/etc/ansible/hosts**) に一覧表示されている最初のマスターから以下のコマンドを実行します。

```
$ oc adm ca create-server-cert \
    --signer-cert=/etc/origin/master/ca.crt \
    --signer-key=/etc/origin/master/ca.key \
    --signer-serial=/etc/origin/master/ca.serial.txt \
    --hostnames='docker-registry.default.svc.cluster.local,docker-
registry.default.svc,172.30.124.220' \
    --cert=/etc/secrets/registry.crt \
    --key=/etc/secrets/registry.key
```

ルーターを外部に公開する場合、公開ルートのホスト名を **--hostnames** フラグに追加します。

```
--hostnames='mydocker-registry.example.com,docker-
registry.default.svc.cluster.local,172.30.124.220 \
```

ルートを外部にアクセス可能にするためにデフォルトの証明書を更新する際のその他詳細については、「レジストリーとルーター証明書の再デプロイ」を参照してください。



注記

oc adm ca create-server-cert コマンドは、2 年間有効な証明書を生成します。この期間は **--expire-days** オプションを使って変更することができますが、セキュリティー上の理由から、値をこれより大きくすることは推奨されません。

4. レジストリー証明書のシークレットを作成します。

```
$ oc create secret generic registry-certificates \
    --from-file=/etc/secrets/registry.crt \
    --from-file=/etc/secrets/registry.key
```

5. シークレットをレジストリー Pod のサービスアカウント (**デフォルト**のサービスアカウントなど) に追加します。

```
$ oc secrets link registry registry-certificates
$ oc secrets link default  registry-certificates
```

注記

デフォルトでは、対象のシークレットを参照するサービスアカウントだけに、そのシークレットを制限する動作は無効になっています。つまり、マスターの設定ファイルで **serviceAccountConfig.limitSecretReferences** が **false** (デフォルトの設定) に設定されている場合、シークレットをサービスにリンクさせる必要はありません。

6. **docker-registry** サービスを一時停止します。

```
$ oc rollout pause dc / docker-registry
```

7. シークレットボリュームをレジストリーのデプロイメント設定に追加します。

```
$ oc volume dc/docker-registry --add --type=secret \
    --secret-name=registry-certificates -m /etc/secrets
```

8. 以下の環境変数をレジストリーのデプロイメント設定に追加して TLS を有効にします。

```
$ oc set env dc/docker-registry \
    REGISTRY_HTTP_TLS_CERTIFICATE=/etc/secrets/registry.crt \
    REGISTRY_HTTP_TLS_KEY=/etc/secrets/registry.key
```

詳細は、Docker ドキュメントのレジストリーの設定のセクションを参照してください。

9. レジストリーの liveness probe に使用されているスキームを HTTP から HTTPS に更新します。

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec":
{"containers":[{
    "name":"registry",
    "livenessProbe":  {"httpGet": {"scheme":"HTTPS"}}
  }]}}}}'
```

10. レジストリーを OpenShift Container Platform 3.2 以降に初めてデプロイした場合は、レジストリーの readiness probe として使用されているスキームを HTTP から HTTPS に更新します。

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec":
{"containers":[{
    "name":"registry",
    "readinessProbe":  {"httpGet": {"scheme":"HTTPS"}}
  }]}}}}'
```

11. **docker-registry** サービスを再開します。

```
$ oc rollout resume dc/docker-registry
```

12. レジストリーが TLS モードで実行されていることを検証します。最後の **docker-registry** のデプロイメントが完了するまで待機し、Docker ログでレジストリーコンテナーを確認します。**`listening on :5000, tls`** のエントリーが見つかるはずです。

```
$ oc logs dc/docker-registry | grep tls
time="2015-05-27T05:05:53Z" level=info msg="listening on :5000, tls"
instance.id=deeba528-c478-41f5-b751-dc48e4935fc2
```

13. CA 証明書を Docker 証明書ディレクトリーにコピーします。これはクラスター内のすべてのノードで実行する必要があります。

```
$ dcertsdir=/etc/docker/certs.d
$ destdir_addr=$dcertsdir/172.30.124.220:5000
$ destdir_name=$dcertsdir/docker-
registry.default.svc.cluster.local:5000

$ sudo mkdir -p $destdir_addr $destdir_name
$ sudo cp ca.crt $destdir_addr     (1)
$ sudo cp ca.crt $destdir_name
```

(1) **ca.crt** ファイルは、マスター上の **/etc/origin/master/ca.crt** のコピーです。

14. 認証を使用している場合、**docker** のバージョンによっては、OS レベルで証明書を信頼するようにクラスターを設定することも必要になります。

    a. 証明書をコピーします。

```
$ cp /etc/origin/master/ca.crt /etc/pki/ca-
trust/source/anchors/myregistrydomain.com.crt
```

    b. 以下を実行します。

```
$ update-ca-trust enable
```

15. **/etc/sysconfig/docker** ファイルのこの特定のレジストリーに対してのみ、**`--insecure-registry`** オプションを削除します。次にデーモンを再度読み込み、 **docker** サービスを再起動して設定の変更を反映させます。

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

16. **docker** クライアント接続を検証します。レジストリーに対する **`docker push`**、またはレジストリーからの **`docker pull`** が正常に実行されるはずです。必ずこのレジストリーにログインしていることを確認してください。

```
$ docker tag|push <registry/image> <internal_registry/project/image>
```

以下は例を示しています。

```
$ docker pull busybox
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
$ docker push 172.30.124.220:5000/openshift/busybox
...
```

```
cf2616975b4a: Image successfully pushed
Digest:
sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403ca
b55
```

2.4.3. セキュアなレジストリーの手動による公開

OpenShift Container Platform クラスター内から、OpenShift Container Platform レジストリーにログインするのではなく、外部からレジストリーにアクセスできるようにするには、先にレジストリーのセキュリティーを確保してから、このレジストリーをルートに公開します。この方法を使うと、ルートアドレスを使ってクラスターの外部からレジストリーにログインし、ルートのホストを使ってイメージにタグ付けしたり、イメージをプッシュしたりできます。

1. 以下の前提条件に関するそれぞれの手順は、標準的なクラスターのインストール時にデフォルトで実行されます。これが実行されていない場合は、手動で実行してください。

   a. レジストリーを手動でデプロイする。

   b. レジストリーを手動でセキュリティー保護する。

   c. ルーターを手動でデプロイする。

2. パススルールートは、クラスターの初回インストール時にこのレジストリーについてデフォルトで作成されている必要があります。

   a. ルートが存在するかどうかを確認します。

```
$ oc get route/docker-registry -o yaml
apiVersion: v1
kind: Route
metadata:
  name: docker-registry
spec:
  host: <host> 1
  to:
    kind: Service
    name: docker-registry 2
  tls:
    termination: passthrough 3
```

1. ルートのホスト。この名前は、外部から DNS 経由でルーターの IP アドレスに解決できる必要があります。
2. レジストリーのサービス名。
3. このルートをパススルールートとして指定します。

注記

Re-encrypt ルートはセキュアなレジストリーを公開するためにもサポートされています。

b. ルートが存在していない場合、**oc create route passthrough** コマンドで、レジストリーをルートのサービスとして指定してルートを作成します。デフォルトでは、作成したルートの名前はサービス名と同じです。

i. **docker-registry** サービスの詳細を取得します。

```
$ oc get svc
NAME              CLUSTER_IP       EXTERNAL_IP   PORT(S)
SELECTOR                  AGE
docker-registry   172.30.69.167    <none>        5000/TCP
docker-registry=default   4h
kubernetes        172.30.0.1       <none>
443/TCP,53/UDP,53/TCP   <none>                    4h
router            172.30.172.132   <none>        80/TCP
router=router             4h
```

ii. ルートを作成します。

```
$ oc create route passthrough    \
    --service=docker-registry    \(1)
    --hostname=<host>
route "docker-registry" created      (2)
```

(1) レジストリーをルートのサービスとして指定します。

(2) ルート名はサービス名と同じです。

3. 次に、ホストシステム上のレジストリーに使用されている証明書を信頼し、ホストによるイメージのプッシュおよびプルを許可する必要があります。参照される証明書は、レジストリーのセキュリティー保護を実行したときに作成されたものです。

```
$ sudo mkdir -p /etc/docker/certs.d/<host>
$ sudo cp <ca_certificate_file> /etc/docker/certs.d/<host>
$ sudo systemctl restart docker
```

4. レジストリーのセキュリティー保護の実行時に得られた情報を使ってレジストリーにログインします。ただし、ここではサービス IP ではなくルートで使用されているホスト名を指定します。セキュリティーが保護され、公開されているレジストリーにログインする際は、必ず **docker login** コマンドのレジストリーを指定してください。

```
# docker login -e user@company.com \
    -u f83j5h6 \
    -p Ju1PeM47R0B92Lk3AZp-bWJSck2F7aGCiZ66aFGZrs2 \
    <host>
```

5. これでルートのホストを使ってイメージのタグ付けとプッシュを実行できます。たとえば、**test** という名前のプロジェクトにある **busybox** イメージをタグ付けし、プッシュするには、以下を実行します。

```
$ oc get imagestreams -n test
NAME      DOCKER REPO   TAGS      UPDATED

$ docker pull busybox
$ docker tag busybox <host>/test/busybox
```

```
$ docker push <host>/test/busybox
The push refers to a repository [<host>/test/busybox] (len: 1)
8c2e06607696: Image already exists
6ce2e90b0bc7: Image successfully pushed
cf2616975b4a: Image successfully pushed
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8c
a31

$ docker pull <host>/test/busybox
latest: Pulling from <host>/test/busybox
cf2616975b4a: Already exists
6ce2e90b0bc7: Already exists
8c2e06607696: Already exists
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8c
a31
Status: Image is up to date for <host>/test/busybox:latest

$ oc get imagestreams -n test
NAME      DOCKER REPO                          TAGS      UPDATED
busybox   172.30.11.215:5000/test/busybox   latest    2 seconds ago
```

注記

イメージストリームにはルート名とポートではなく、レジストリーサービスの IP アドレスとポートが設定されます。詳細は **`oc get imagestreams`** を参照してください。

2.4.4. 非セキュアなレジストリーの手動での公開

レジストリーのセキュリティーを確保してレジストリーを公開する代わりに、OpenShift Container Platform の非稼働環境に、セキュアでないレジストリーを簡単に公開できます。これにより、SSL 証明書を使用せずにレジストリーへの外部ルートを作成することができます。

警告

非実稼働環境以外では、非セキュアなレジストリーを外部アクセスに公開すべきではありません。

非セキュアなレジストリーを公開するには、以下を実行します。

1. レジストリーを公開します。

```
# oc expose service docker-registry --hostname=<hostname> -n default
```

以下の JSON ファイルが作成されます。

```
apiVersion: v1
```

```
kind: Route
metadata:
  creationTimestamp: null
  labels:
    docker-registry: default
  name: docker-registry
spec:
  host: registry.example.com
  port:
    targetPort: "5000"
  to:
    kind: Service
    name: docker-registry
status: {}
```

2. ルートが正常に作成されたことを確認します。

```
# oc get route
NAME              HOST/PORT                     PATH      SERVICE
LABELS                    INSECURE POLICY   TLS TERMINATION
docker-registry   registry.example.com            docker-registry
docker-registry=default
```

3. レジストリーの健全性を確認します。

```
$ curl -v http://registry.example.com/healthz
```

HTTP 200 / OK メッセージが表示されるはずです。

レジストリーを公開したら、ポート番号を **OPTIONS** エントリーに追加して **/etc/sysconfig/docker** ファイルを更新します。以下は例になります。

```
OPTIONS='--selinux-enabled --insecure-registry=172.30.0.0/16 --
insecure-registry registry.example.com:80'
```

重要

上記のオプションは、ログインしようとしているクライアントに追加してください。

また、Docker がクライアント上で実行されていることも確認してください。

公開されている非セキュアなレジストリーにログインする際に、**docker login** コマンドでレジストリーを指定していることを確認します。以下は例になります。

```
# docker login -e user@company.com \
    -u f83j5h6 \
    -p Ju1PeM47R0B92Lk3AZp-bWJSck2F7aGCiZ66aFGZrs2 \
    <host>
```

2.5. レジストリー設定の拡張

2.5.1. レジストリー IP アドレスの維持

OpenShift Container Platform はサービス IP アドレスによって統合レジストリーを参照します。したがって **docker-registry** サービスを削除し、再作成しようとする場合、古い IP アドレスを新しいサービスで再利用するように調整すると、完全に透過的な移行が可能になります。新規 IP アドレスを取得することが避けられない場合は、マスターのみを再起動してクラスターの中断を最小限に抑えられます。

アドレスの再利用

IP アドレスを再利用するには、古い **docker-registry** サービスの IP アドレスを削除する前に保存し、新たに割り当てられた IP アドレスを、新しい **docker-registry** サービスに保存されているアドレスに置き換えるように調整します。

1. サービスの **clusterIP** を書き留めておきます。

```
$ oc get svc/docker-registry -o yaml | grep clusterIP:
```

2. サービスを削除します。

```
$ oc delete svc / docker-registry dc / docker-registry
```

3. レジストリーの定義を **registry.yaml** に作成し、**<options>** を、たとえば「非実稼働環境での使用」のセクションの手順 3 で使用されているものなどに置き換えます。

```
$ oc adm registry <options> -o yaml > registry.yaml
```

4. **registry.yaml** を編集し、そこで **Service** を検索して、**clusterIP** を手順 1 で書き留めたアドレスに変更します。

5. 変更した **registry.yaml** を使ってレジストリーを作成します。

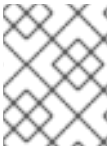
```
$ oc create -f registry.yaml
```

マスターの再起動

IP アドレスを再利用できない場合、古い IP アドレスを含むプル仕様を使った操作は失敗します。クラスターの中断を最小限に抑えるには、マスターを再起動する必要があります。

```
# master-restart api
# master-restart controllers
```

これで、古い IP アドレスを含む古いレジストリー URL がキャッシュからクリアされます。



注記

クラスター全体を再起動すると、Pod に不要なダウンタイムが発生し、実質、キャッシュのクリアが行われないので、これは推奨していません。

2.5.2. Docker レジストリーのホワイトリスト化

Docker レジストリーのホワイトリストを指定すると、OpenShift Container Platform ユーザーがダウンロードして入手できるイメージやテンプレートのセットをキュレートできます。このキュレートされたセットは、1 つまたは複数の Docker レジストリーに保管され、その後ホワイトリストに追加されます。ホワイトリストを使用する場合、OpenShift Container Platform からアクセスできるのは指定されたレジストリーのみで、その他すべてのレジストリーはデフォルトでアクセスが拒否されます。

ホワイトリストを設定するには、以下を実行します。

1. **/etc/sysconfig/docker** ファイルを編集し、すべてのレジストリーをブロックします。

```
BLOCK_REGISTRY='--block-registry=all'
```

BLOCK_REGISTRY 行のコメント解除が必要となる場合があります。

2. 同じファイルで、アクセスを許可するレジストリーを追加します。

```
ADD_REGISTRY='--add-registry=<registry1> --add-registry=<registry2>'
```

レジストリーへのアクセスの許可

```
ADD_REGISTRY = ' -  add-registry = registry.access.redhat.com'
```

上記の例では、Red Hat カスタマーポータルで入手できるイメージへのアクセスを制限しています。

ホワイトリストが設定されると、ホワイトリストにない Docker レジストリーからプルしようとすると、許可されていないレジストリーであることを示すエラーメッセージが表示されます。

2.5.3. レジストリーのホスト名の設定

内部参照と外部参照の両方でレジストリーを認識するために使用するホスト名とポートを設定できます。これを実行すると、イメージストリームはイメージのホスト名ベースのプッシュおよびプル仕様を提供することができ、これによってイメージのコンシューマーがレジストリーのサービス IP の変更の影響を受けないようにし、イメージストリームとその参照をクラスター間で移植できる可能性があります。

クラスター内でレジストリーを参照するために使用されるホスト名を設定するには、マスター設定ファイルの **imagePolicyConfig** セクションで **internalRegistryHostname** を設定します。外部のホスト名は、同じ場所で **externalRegistryHostname** の値を設定することで管理されます。

イメージポリシーの設定

```
imagePolicyConfig:
  internalRegistryHostname: docker-registry.default.svc.cluster.local:5000
  externalRegistryHostname: docker-registry.mycompany.com
```

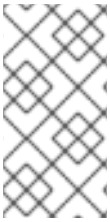
レジストリー自体は、同じ内部ホスト名の値で設定する必要があります。これは、レジストリーのデプロイメント設定で **REGISTRY_OPENSHIFT_SERVER_ADDR** 環境変数を設定するか、またはレジストリー設定の OpenShift セクションで値を設定することで実行できます。

注記

レジストリーで TLS を有効にしている場合、サーバー証明書にはレジストリーの参照に使用されるホスト名が含まれている必要があります。ホスト名をサーバー証明書に追加する方法については、「レジストリーのセキュリティー保護」を参照してください。

2.5.4. レジストリー設定の上書き

統合レジストリーのデフォルトの設定 (デフォルトでは実行中のレジストリーコンテナーの **/config.yml** にあります) は、独自の カスタム設定で上書きできます。



注記

このファイルのアップストリームの設定オプションも環境変数を使って上書きできます。ミドルウェアのセクションは、環境変数を使って上書きできるオプションがごく少数であるため例外とします。特定の設定オプションを上書きする方法についてこちらを参照してください。

レジストリー設定ファイルの直接的な管理を可能にし、**ConfigMap** を使って更新された設定をデプロイするには、以下を実行します。

1. レジストリーをデプロイします。
2. 必要に応じて、レジストリー設定ファイルをローカルで編集します。以下は、レジストリーにデプロイされている初期 YAML ファイルです。サポートされているオプションを確認してください。

レジストリー設定ファイル

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /registry
  delete:
    enabled: true
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
      options:
        acceptschema2: true
        pullthrough: true
        enforcequota: false
        projectcachettl: 1m
        blobrepositorycachettl: 10m
  storage:
    - name: openshift
openshift:
  version: 1.0
  metrics:
    enabled: false
    secret: <secret>
```

3. 各ファイルの内容を保持する **ConfigMap** をこのディレクトリーに作成します。

```
$ oc create configmap registry-config \
    --from-file=</path/to/custom/registry/config.yml>/
```

4. **registry-config** ConfigMap をボリュームとしてレジストリーのデプロイメント設定に追加し、カスタム設定ファイルを **/etc/docker/registry/** にマウントします。

```
$ oc volume dc/docker-registry --add --type=configmap \
    --configmap-name=registry-config -m /etc/docker/registry/
```

5. 以下の環境変数をレジストリーのデプロイメント設定に追加して、直前の手順の設定パスを参照するようにレジストリーを更新します。

```
$ oc set env dc/docker-registry \
    REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yml
```

上記の手順は、反復プロセスとして実行し、希望の設定を実現することができます。たとえば、トラブルシューティング時に、**デバッグ** モードに切り換えるよう設定が一時的に更新される場合があります。

既存の設定を更新するには、以下を実行します。

警告

この手順を実行すると、現在デプロイされているレジストリー設定が上書きされます。

1. ローカルのレジストリー設定ファイル **config.yml** を編集します。
2. **registry-config** configmap を削除します。

```
$ oc delete configmap registry-config
```

3. 更新された設定ファイルを参照するよう configmap を再作成します。

```
$ oc create configmap registry-config\
    --from-file=</path/to/custom/registry/config.yml>/
```

4. 更新された設定を読み取るためにレジストリーを再デプロイします。

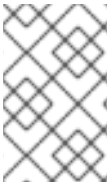
```
$ oc rollout latest docker-registry
```

ヒント

設定ファイルをソース管理リポジトリーに保持します。

2.5.5. レジストリー設定の参照

アップストリームの docker ディストリビューションライブラリーでは、多数の設定オプションを利用できます。ただし、すべての設定オプションがサポートされているか、または有効にされているわけではありません。このセクションは、レジストリー設定を上書きする際の参考としてご利用ください。



注記

このファイルのアップストリームの設定オプションも環境変数を使って上書きできます。ただし、ミドルウェアのセクションは、環境変数を使って上書きすることは**できません**。特定の設定オプションを上書きする方法についてこちらを参照してください。

2.5.5.1. ログ

アップストリームのオプションはサポートされています。

例:

```
log:
  level: debug
  formatter: text
  fields:
    service: registry
    environment: staging
```

2.5.5.2. フック

メールフックはサポートされていません。

2.5.5.3. ストレージ

以下のセクションでは、サポートされているレジストリーのストレージドライバーの一覧を示しています。詳細は、Docker レジストリーのドキュメントを参照してください。

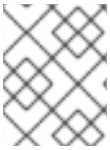
以下の一覧には、レジストリーの設定ファイルで設定を行う必要があるストレージドライバーが含まれます。

- ファイルシステム。ファイルシステムはデフォルトですので、設定の必要はありません。
- S3。詳細は、CloudFront の設定のドキュメントを参照してください。
- OpenStack Swift
- Google Cloud Storage (GCS)
- Microsoft Azure
- Aliyun OSS

レジストリーの一般的なストレージ設定オプションはサポートされています。詳細は、Docker レジストリーのドキュメントを参照してください。

以下のストレージオプションは、ファイルシステムドライバーで設定する必要があります。

- GlusterFS ストレージ
- Ceph Rados ブロックデバイス



注記

サポートされている永続ストレージドライバーの詳細は、「永続ストレージの設定」および「永続ストレージの例」を参照してください。

一般的なストレージ設定オプション

```
storage:
  delete:
    enabled: true (1)
  redirect:
    disable: false
  cache:
    blobdescriptor: inmemory
  maintenance:
    uploadpurging:
      enabled: true
      age: 168h
      interval: 24h
      dryrun: false
    readonly:
      enabled: false
```

(1) このエントリーは、イメージのプルーニングが正常に機能するために**必須**です。

2.5.5.4. 認証

認証オプションは変更することができません。**openshift** の拡張がサポートされている唯一のオプションです。

```
auth:
  openshift:
    realm: openshift
```

2.5.5.5. ミドルウェア

リポジトリーのミドルウェアの拡張を使用して、OpenShift Container Platform やイメージプロキシーとの対話を行う OpenShift Container Platform ミドルウェアの設定を行うことができます。

```
middleware:
  registry:
    - name: openshift (1)
  repository:
    - name: openshift (2)
      options:
        acceptschema2: true (3)
        pullthrough: true (4)
        mirrorpullthrough: true (5)
        enforcequota: false (6)
        projectcachettl: 1m (7)
```

```
        blobrepositorycachettl: 10m (8)
  storage:
    - name: openshift (9)
```

(1)(2)(9) これらの入力は必須です。これらの入力によって必要なコンポーネントが読み込まれていることを確認できます。これらの値は変更しないでください。

(3) レジストリーへのプッシュ時に manifest schema v2 を格納できます。詳細は、こちらを参照してください。

(4) レジストリーがリモート Blob のプロキシーとして機能できるようにします。詳細は、 こちらを参照してください。

(5) レジストリーキャッシュの Blob がリモートレジストリーから提供されるようにし、その後の迅速なアクセスを可能にします。Blob の初回アクセス時にミラーリングが開始されます。このオプションは、プルスルーが無効にされている場合は機能しません。

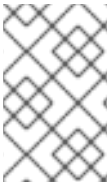
(6) ターゲットに設定されているプロジェクトで定義されているサイズ制限を超える Blob のアップロードを防止します。

(7) レジストリーにキャッシュされている制限の有効期限のタイムアウト。値が小さいほど、制限の変更がレジストリーに伝搬するまでの時間が短くなります。ただし、レジストリーは制限をサーバーからより頻繁に照会するため、結果としてプッシュは遅くなります。

(8) Blob とリポジトリー間の記憶されている関連付けの有効期限のタイムアウト。値が高いほどルックアップが高速になり、レジストリー操作がより効率的になる可能性が高くなります。一方、アクセスできなくなったユーザーにイメージレイヤーを提供するリスクと同様にメモリー使用量も上昇します。

2.5.5.5.1. CloudFront ミドルウェア

CloudFront ミドルウェア拡張は、CloudFront CDN ストレージプロバイダーである AWS をサポートするために追加することができます。CloudFront ミドルウェアは、イメージコンテンツの海外への配信を高速化します。Blob は世界中の複数のエッジロケーションに配信されます。クライアントは常に最小の待機時間でエッジにアクセスできます。



注記

CloudFront ミドルウェア拡張を使用できるストレージは S3 ストレージのみです。また、使用できるのは Blob が提供されている間のみです。したがって、高速化できるのは Blob のダウンロードのみで、アップロードは高速化しません。

以下は、CloudFront ミドルウェアを含む S3 ストレージドライバーの最小限の設定例です。

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
```

```
  s3: (1)
    accesskey: BJKMSZBRESWJQXRWMAEQ
    secretkey: 5ah5I91SNXbeoUXXDasFtadRqOdy62JzlnOW1goS
    region: us-east-1
    bucket: docker.myregistry.com
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
  storage:
    - name: cloudfront (2)
      options:
        baseurl: https://jrpbyn0k5k88bi.cloudfront.net/ (3)
        privatekey: /etc/docker/cloudfront-ABCEDFGHIJKLMNOPQRST.pem (4)
        keypairid: ABCEDFGHIJKLMNOPQRST (5)
    - name: openshift
```

(1) S3 ストレージは、CloudFront ミドルウェアに関係なく同じ方法で設定する必要があります。

(2) CloudFront ストレージミドルウェアは、OpenShift ミドルウェアより前に一覧表示される必要があります。

(3) CloudFront ベースの URL。AWS マネジメントコンソールでは、これは CloudFront ディストリビューションの **Domain Name** として一覧表示されます。

(4) AWS のプライベートキーが格納されているファイルシステムの場所。Amazon EC2 のキーペアと混同しないよう注意してください。信頼される署名者の CloudFront キーペアの作成については、AWS ドキュメントを参照してください。このファイルは、レジストリー Pod にシークレットとしてマウントする必要があります。

(5) Cloudfront キーペアの ID。

2.5.5.5.2. ミドルウェア設定オプションの上書き

middleware セクションは、環境変数を使って上書きできません。ただし例外がいくつかあります。以下は例になります。

```
middleware:
  repository:
    - name: openshift
      options:
        acceptschema2: true (1)
        pullthrough: true (2)
        mirrorpullthrough: true (3)
        enforcequota: false (4)
        projectcachettl: 1m (5)
        blobrepositorycachettl: 10m (6)
```

1. ブール環境変数 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ACCEPTSCHEMA2** で上書きできる設定オプション。manifest schema v2 をマニフェストの Put 要求で 受け入れる機能を有効にします。認識される値は **true** と **false** (以下の他のすべてのブール変数に適用されます) になります。

2. ブール環境変数 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_PULLTHROUGH** で上書きできる設定オプション。リモートレジストリーのプロキシーモードを有効にします。

3. ブール環境変数 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_MIRRORPULLTHROUGH** で上書きできる設定オプション。リモート Blob が提供されている場合、レジストリーに対して Blob をローカルにミラーリングするように指示します。

4. ブール環境変数 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA** で上書きできる設定オプション。クォータ実施をオンまたはオフにする機能を有効にします。デフォルトでは、クォータの実施はオフになっています。

5. 環境変数 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_PROJECTCACHETTL** で上書きできる設定オプション。プロジェクトのクォータオブジェクトのエビクションタイムアウトを指定します。有効な時間文字列 (例: **2m**) を取ります。空白の場合は、デフォルトのタイムアウトが取得されます。ゼロ (**0m**) の場合、キャッシングは無効にされます。

6. 環境変数 **REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_BLOBREPOSITORYCACHETTL** で上書きできる設定オプション。Blob と含まれているレポジトリーの関連付けについてのエビクションタイムアウトを指定します。値のフォーマットは **projectcachettl** のケースと同じです。

2.5.5.5.3. イメージのプルスルー

この機能を有効にすると、Blob がローカルに存在しない限り、レジストリーは要求された Blob をリモートレジストリーから取得しようと試みます。リモートの候補は、クライアントがプルするイメージストリームの状態で保存された **DockerImage** エントリーから算出されます。このエントリーのすべての固有のリモートレジストリーの参照は Blob が見つかるまで繰り返し試行されます。

プルスルーは、プルされているイメージのイメージストリームタグが存在する場合にのみ発生します。たとえば、プルされているイメージが **docker-registry.default.svc:5000/yourproject/yourimage:prod** である場合、レジストリーは、プロジェクト **yourproject** で **yourimage:prod** という名前のイメージストリームタグを検索します。タグが見つかると、レジストリーはそのイメージストリームタグに関連付けられた **dockerImageReference** を使ってイメージのプルを試行します。

プルスルーを実行すると、レジストリーは、参照されているイメージストリームタグに関連付けられたプロジェクトにあるプル認証情報を使用します。また、この機能を使用すると、ユーザーは、イメージを参照しているイメージストリームタグにアクセスできる限り、アクセスに必要な認証情報を持たないレジストリーのイメージをプルすることができます。

使用しているレジストリーに、プルスルーの実行対象である外部レジストリーを信頼するのに必要な証明書があることを確認してください。証明書は Pod の **/etc/pki/tls/certs** ディレクトリーに配置する必要があります。証明書は設定マップまたはシークレットを使ってマウントできます。ここで、**/etc/pki/tls/certs** ディレクトリー全体を置き換える必要があることに注意してください。新しい証明書を組み込み、マウントするシークレットまたは設定マップのシステム証明書を置き換えます。

デフォルトでは、イメージストリームタグは **Source** の参照ポリシータイプを使用することに注意してください。これは、イメージストリームの参照がイメージのプル仕様に対して解決される場合、使用される仕様はイメージのソースを参照することを意味します。外部レジストリーでホストされているイメージであれば、外部レジストリーが参照され、この結果としてリソースは外部レジストリーでイメー

ジを参照し、プルします。この場合、**registry.access.redhat.com/openshift3/jenkins-2-rhel7** とプルスルーは適用されません。イメージストリームを参照しているリソースが内部レジストリーを参照しているプル仕様を使用するようにするには、イメージストリームタグは **Local** の参照ポリシータイプを使用する必要があります。詳細は、「参照ポリシー」を参照してください。

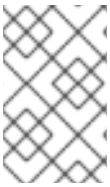
この機能はデフォルトでオンになっています。ただし、設定オプションを使って無効にすることができます。

デフォルトでは、この方法で提供されるすべてのリモート Blob は、**mirrorpullthrough** が無効にされていない限りローカルに格納され、その後のアクセスが高速になります。ミラーリング機能の欠点はストレージの使用量が増えることにあります。

注記

ミラーリングは、クライアントが Blob を 1 バイト以上取得しようとする際に開始されます。実際に必要になる前に、特定イメージを統合レジストリーに事前にフェッチするには、以下のコマンドを実行します。

```
$ oc get imagestreamtag/${IS}:${TAG} -o jsonpath='{
.image.dockerImageLayers[*].name }' | \
  xargs -n1 -I {} curl -H "Range: bytes=0-1" -u user:${TOKEN} \
  http://${REGISTRY_IP}:${PORT}/v2/default/mysql/blobs/{}
```



注記

OpenShift Container Platform のミラーリング機能をアップストリームの レジストリーのプルスルーキャッシュ機能と混同しないようにしてください。これらは似ていますが、全く異なる機能です。

2.5.5.5.4. Manifest Schema v2 サポート

各イメージには、Blob を記述するマニフェスト、それを実行するための命令、および追加のメタデータがあります。マニフェストはバージョン管理されており、バージョンごとに構造やフィールドが異なります。同一イメージが複数のマニフェストバージョンで表現されます。それぞれのバージョンはそれぞれ異なるダイジェストがあります。

現在サポートされているレジストリーは manifest v2 schema 1 (**schema1**) と manifest v2 schema 2 (**schema2**) です。前者は古くなっていますが、今後もしばらくはサポートされます。

各種の Docker クライアントとの互換性の問題に注意する必要があります。

- Docker クライアントのバージョン 1.9 以前は、**schema1** のみをサポートしています。このクライアントがプルまたはプッシュするマニフェストはレガシースキーマになります。
- Docker クライアントのバージョン 1.10 は、**schema1** と **schema2** の両方をサポートしています。デフォルトでは、新しい方のスキーマをサポートする場合はこちらをレジストリーにプッシュします。

イメージを **schema1** で格納するレジストリーは、常にイメージを変更せずにクライアントに返します。**Schema2** は新規の Docker クライアントにのみ変更しない状態で移動します。古いクライアントの場合は、オンザフライで **schema1** に変換されます。

これにより、大きな影響が想定されます。たとえば、新規 Docker クライアントでレジストリーにプッシュされたイメージは、古い Docker のダイジェストでプルすることはできません。格納されたイメージのマニフェストは **schema2** であり、そのダイジェストはこのバージョンのマニフェストをプルする

場合にのみ使用できるためです。

このため、レジストリーはデフォルトでは **schema2** を格納しないように設定されています。これにより、すべての Docker クライアントはクライアントのバージョンに関係なく、プッシュされたイメージをレジストリーからプルできます。

すべてのレジストリークライアントが **schema2** をサポートしていることを確認できたら、そのサポートをレジストリーで有効にすることができます。特定のオプションについては、上記の「ミドルウェア設定の参照」を参照してください。

2.5.5.6. OpenShift

このセクションでは、OpenShift Container Platform に特有の機能のグローバル設定について説明します。今後のリリースでは、 Middleware セクションにある **openshift** 関連の設定は廃止される予定です。

現在、このセクションではレジストリーメトリクスの収集を設定できます。

```
openshift:
  version: 1.0 (1)
  server:
    addr: docker-registry.default.svc (2)
  metrics:
    enabled: false (3)
    secret: <secret> (4)
  requests:
    read:
      maxrunning: 10 (5)
      maxinqueue: 10 (6)
      maxwaitinqueue 2m (7)
    write:
      maxrunning: 10 (8)
      maxinqueue: 10 (9)
      maxwaitinqueue 2m (10)
```

(1) このセクションの設定バージョンを指定する必須エントリー。サポートされている値は **1.0** のみです。

(2) レジストリーのホスト名。マスターで設定されている値と同じ値に設定される必要があります。これは環境変数 **REGISTRY_OPENSHIFT_SERVER_ADDR** で上書きされる可能性があります。

(3) メトリクスの収集を有効にするには **true** に設定します。これはブール環境変数 **REGISTRY_OPENSHIFT_METRICS_ENABLED** で上書きされる可能性があります。

(4) クライアント要求の承認に使用されるシークレット。メトリクスのクライアントは これを **Authorization** ヘッダーでベアラートークンとして使用します。環境変数 **REGISTRY_OPENSHIFT_METRICS_SECRET** で上書きできます。

(5) 同時に行えるプル要求の最大数。環境変数 **REGISTRY_OPENSHIFT_REQUESTS_READ_MAXRUNNING** で上書きできます。ゼロは無制限を意味します。

(6)

キューに入れられるプル要求の最大数。環境変数 **REGISTRY_OPENSHIFT_REQUESTS_READ_MAXINQUEUE** で上書きできます。ゼロは無制限を意

7 拒否されるまでのキューにあるプル要求の最大待機時間。環境変数 **REGISTRY_OPENSHIFT_REQUESTS_READ_MAXWAITINQUEUE** で上書きできます。ゼロは無制限を意味します。

8 同時に行えるプッシュ要求の最大数。環境変数 **REGISTRY_OPENSHIFT_REQUESTS_WRITE_MAXRUNNING**で上書きできます。ゼロは無制限を意味します。

9 キューにあるプッシュ要求の最大数。環境変数 **REGISTRY_OPENSHIFT_REQUESTS_WRITE_MAXINQUEUE** で上書きできます。ゼロは無制限を意味します。

10 拒否されるまでのキューにあるプッシュ要求の最大待機時間。環境変数 **REGISTRY_OPENSHIFT_REQUESTS_WRITE_MAXWAITINQUEUE** で上書きできます。ゼロは無制限を意味します。

使用状況の情報については レジストリーメトリクスへのアクセスを参照してください。

2.5.5.7. レポート (Reporting)

レポート (Reporting) はサポートされていません。

2.5.5.8. HTTP

アップストリームのオプションはサポートされています。環境変数を使って設定を変更する方法についてはこちらを参照してください。変更の必要があるのは **tls** セクションのみです。以下は例になります。

```
http:
  addr: :5000
  tls:
    certificate: /etc/secrets/registry.crt
    key: /etc/secrets/registry.key
```

2.5.5.9. 通知

アップストリームのオプションはサポートされています。REST API リファレンス はより包括的な統合オプションを提供します。

例:

```
notifications:
  endpoints:
    - name: registry
      disabled: false
      url: https://url:port/path
      headers:
        Accept:
          - text/plain
```

```
      timeout: 500
      threshold: 5
      backoff: 1000
```

2.5.5.10. Redis

Redis はサポートされていません。

2.5.5.11. 健全性

アップストリームのオプションはサポートされています。レジストリーのデプロイメント設定は、**/healthz** で統合されたヘルスチェックを提供します。

2.5.5.12. プロキシー

プロキシー設定は有効にできません。この機能は OpenShift Container Platform リポジトリーのミドルウェア拡張、**pullthrough: true** で提供されます。

2.5.5.13. キャッシュ

統合レジストリーは、データをアクティブにキャッシュして、速度の遅い外部リソースに対する呼び出しの回数を減らします。キャッシュには 2 種類あります。

1. Blob のメタデータのキャッシュに使用されるストレージキャッシュ。このキャッシュには有効期限がなく、データは明示的に削除されるまで残り続けます。
2. アプリケーションキャッシュには、Blob とリポジトリーの関連付けが含まれます。このキャッシュ内のデータには有効期限があります。

キャッシュを完全にオフににするには設定を変更する必要があります。

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache: {} 1
openshift:
  version: 1.0
  cache:
    disabled: true 2
    blobrepositoryttl: 10m
```

1 ストレージのバックエンドでアクセスしたメタデータのキャッシュを無効にします。このキャッシュがない場合、レジストリーサーバーはメタデータのバックエンドに絶えずアクセスします。

2 Blob とリポジトリーの関連付けを含むキャッシュを無効にします。このキャッシュがない場合、レジストリーサーバーは継続的にマスター API のデータを照会し、関連付けを再計算します。

2.6. 既知の問題

2.6.1. 概要

以下は、統合レジストリーのデプロイまたは使用時の既知の問題です。

2.6.2. レジストリーのプルスルーに伴う同時ビルド

ローカルの **docker-registry** デプロイメントは追加の負荷を受けます。デフォルトで、ここでは **registry.access.redhat.com** からのコンテンツをキャッシュするようになりました。STI ビルドの **registry.access.redhat.com** のイメージはローカルレジストリーに保存されます。それらをプルしようとすると、ローカル **docker-registry** からのプルが試行され。その結果として、過剰な数の同時ビルドがプルでタイムアウトになり、ビルドが失敗する可能性のある状況になります。この問題を軽減するには、**docker-registry** デプロイメントを複数のレプリカにスケーリングします。Pod のログでタイムアウトの有無をチェックします。

2.6.3. 共有 NFS ボリュームとスケーリングされたレジストリーの使用時のイメージのプッシュエラー

スケーリングされたレジストリーを共有 NFS ボリュームで使用すると、イメージのプッシュ時に以下のいずれかのエラーが発生することがあります。

- **digest invalid: provided digest did not match uploaded content**
- **blob upload unknown**
- **blob upload invalid**

これらのエラーは、Docker のイメージのプッシュの試行時に内部レジストリーサービスによって返されます。その原因は、ノード間のファイル属性の同期に起因します。NFS クライアント側のキャッシング、ネットワーク待機時間およびレイヤーサイズなどはすべて、デフォルトのラウンドロビンロードバランシング設定を使用してイメージをプッシュする際に発生するエラーの要因になる可能性があります。

このような障害の可能性を最小限に抑えるには、以下の手順を実行します。

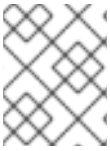
1. **docker-registry** サービスの **sessionAffinity** が**ClientIP** に設定されていることを確認します。

```
$ oc get svc / docker-registry --template = '{{。
spec.sessionAffinity}}'
```

これにより **ClientIP** が返されるはずです。ClientIP は OpenShift Container Platform の最近のバージョンのデフォルトです。これが返されない場合は、以下のように変更してください。

```
$ oc patch svc / docker-registry -p '{' spec ':{' sessionAffinity
':' ClientIP '}}'
```

2. NFS サーバー上のレジストリーボリュームの NFS エクスポート行に **no_wdelay** オプションが一覧表示されていることを確認します。**no_wdelay** オプションは、サーバーによる書き込みの遅延を防ぎ、このレジストリーの要件である、書き込み直後の読み取りの整合性を大幅に改善します。



注記

NFS についてのガイドラインの使用は開始時に推奨されています。実稼働環境に移行するときは、NFS をオフにすることができます。

2.6.4. 内部で管理されたイメージのプルに失敗し「見つかりません (not found)」のエラーが表示される

このエラーは、プルされたイメージがプルしたイメージストリームとは異なるイメージストリームにプッシュされた場合に発生します。これは、ビルドされたイメージを任意のイメージストリームに再タグ付けすることによって発生します。

```
$ oc tag srcimagestream:latest anyproject/pullimagestream:latest
```

その後、以下のようなイメージ参照を使用してプルします。

```
internal.registry.url：5000 / anyproject / pullimagestream：latest
```

Docker を手動でプルするときにも同様のエラーが発生します。

```
Error: image anyproject/pullimagestream:latest not found
```

このエラーを防ぐには、内部で管理されたイメージのタグ付けを完全に避けるか、またはビルドしたイメージを必要な namespace に 手動で再プッシュします。

2.6.5. S3 ストレージでのイメージのプッシュが失敗し「500 内部サーバーエラー (500 Internal Server Error)」と表示される

レジストリーが S3 ストレージのバックエンドで実行されると、問題が報告されます。 Docker レジストリーへのプッシュは、以下のエラーを出して失敗することがあります。

```
Received unexpected HTTP status: 500 Internal Server Error
```

これをデバッグするには、レジストリーのログを表示する必要があります。ログで、プッシュの失敗時に発生した同様のエラーメッセージを探します。

```
time="2016-03-30T15:01:21.22287816-04:00" level=error msg="unknown error
completing upload: driver.Error{DriverName:\"s3\", Enclosed:(*url.Error)
(0xc20901cea0)}" http.request.method=PUT
...
time="2016-03-30T15:01:21.493067808-04:00" level=error msg="response
completed with error" err.code=UNKNOWN err.detail="s3: Put
https://s3.amazonaws.com/oso-tsi-
docker/registry/docker/registry/v2/blobs/sha256/ab/abe5af443833d60cf672e2a
c57589410dddec060ed725d3e676f1865af63d2e2/data: EOF" err.message="unknown
error" http.request.method=PUT
...
time="2016-04-02T07:01:46.056520049-04:00" level=error msg="error putting
into main store: s3: The request signature we calculated does not match
the signature you provided. Check your key and signing method."
http.request.method=PUT
atest
```

このようなエラーを確認した場合には、Amazon S3 サポートにお問い合わせください。リージョンや特定のバケットで問題がある可能性があります。

2.6.6. イメージのプルーニングの失敗

イメージのプルーニング時に以下のエラーが発生した場合:

```
BLOB
sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c
error deleting blob
```

さらに、レジストリーのログに以下の情報が含まれている場合。

```
error deleting blob
\"sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c\
": operation unsupported
```

上記に該当する場合、お使いのカスタム設定ファイルにはストレージセクションに必須のエントリー、つまり **true** に設定された **`storage:delete:enabled`** が含まれていないことを意味します。これらを追加し、レジストリーを再デプロイして、再度イメージプルーニングの操作を行ってください。

第3章 ルーターのセットアップ

3.1. ルーターの概要

3.1.1. ルーターについて

トラフィックをクラスターに送る方法は多数あります。最も一般的な方法として、 OpenShift Container Platform インストールで OpenShift Container Platform ルーターを、サービス向けの外部トラフィックの ingress ポイントとして使用できます。

OpenShift Container Platform は以下のルータープラグインを提供し、サポートしています。

- HAProxy テンプレートルーターはデフォルトのプラグインです。これは、**openshift3/ose-haproxy-router** イメージを使用して、OpenShift Container Platform のコンテナーにあるテンプレートルータープラグインと共に HAProxy インスタンスを実行します。現在は、HTTP(S) トラフィックと SNI を使用した TLS 対応のトラフィックをサポートしています。ルーターのコンテナーは、プライベート IP でのみリッスンする多数のコンテナーとは異なり、ホストのネットワークインターフェースでリッスンします。ルーターは、ルートに関連付けられたサービスで識別される実際の Pod の IP に対するルート名の外部要求をプロキシー処理します。
- F5 ルーターは、ルートを同期するためにお使いの環境で既存の **F5 BIG-IP®** システムに統合されます。F5 iControl REST API を使用するには、**F5 BIG-IP®** バージョン 11.4 以降が必要です。

注記

F5 ルータープラグインは OpenShift Container Platform 3.0.2 以降で利用できます。

- デフォルトの HAProxy ルーターのデプロイ
- カスタム HAProxy ルーターのデプロイ
- F5 BIG-IP® ルーターのデプロイ
- PROXY プロトコルを使用するように HAProxy ルーターを設定する
- ルートのタイムアウトの設定

3.1.2. ルーターのサービスアカウント

OpenShift Container Platform クラスターをデプロイする前に、ルーターのサービスアカウントを用意しておく必要があります。これには、クラスターのインストール時に自動的に作成されます。このサービスアカウントには、ホストのポートの指定に使用できる SCC (security context constraints) へのパーミッションがあります。

3.1.2.1. ラベルにアクセスするためのパーミッション

namespace ラベルが使用されている場合 (ルーターシャードを作成している場合など)、ルーターのサービスアカウントには**cluster-reader** のパーミッションが必要になります。

```
$ oc adm policy add-cluster-role-to-user \
    cluster-reader \
    system:serviceaccount:default:router
```

サービスアカウントを準備したら、デフォルト HAProxy ルーター、カスタマイズ HAProxy ルーター、または F5 ルーターのインストールに進むことができます。

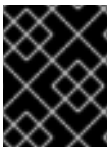
3.2. デフォルト HAPROXY ルーターの使用

3.2.1. 概要

oc adm router コマンドには、新規インストールでのルーターのセットアップタスクを単純化する管理者 CLI が提供されます。**oc adm router** コマンドは、サービスとデプロイメント設定オブジェクトを作成します。 **--service-account** オプションを使用して、ルーターがマスターとの通信で使用するサービスアカウントを指定します。

ルーターサービスアカウントは事前に作成するか、**oc adm router --service-account** コマンドで作成することができます。

OpenShift Container Platform コンポーネント間のあらゆる形式の通信は TLS によって保護され、各種の証明書や認証方法を使用します。**--default-certificate** .pem フォーマットファイルは提供されるか、または **oc adm router** コマンドで作成されます。ルートが作成されると、ユーザーはルートの処理時にルーターが使用するルート証明書を提供できるようになります。



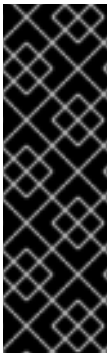
重要

ルーターを削除する際に、デプロイ設定やサービス、およびシークレットも削除されていることを確認します。

ルーターは特定のノードにデプロイされます。これにより、クラスター管理者と外部ネットワークマネージャーはどの IP アドレスがルーターを実行し、ルーターがどのトラフィックを処理するかについて調整しやすくなります。ノードセレクターを使用してルーターを特定のノードにデプロイします。

重要

ルーターはデフォルトでホストネットワークを使用し、ホストのすべてのインターフェースのポート 80 と 443 に直接割り当てられます。ポート 80/443 が利用できるホストにルーターを制限し、他のサービスに使用されないようにします。これはノードセレクターとスケジューラー設定を使用して設定します。たとえば、これはルートなどのサービスの実行用として専用のインフラストラクチャーノードを設定することで実行できます。

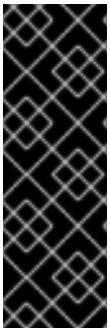


重要

お使いのルーターで個別の **openshift-router** サービスアカウントを使用することをお勧めします。**--service-account** フラグを **oc adm router** コマンドで使用してこれを指定できます。

```
$ oc adm router --dry-run --service-account=router 1
```

1 **--service-account** は、**openshift-router** のサービスアカウントの名前です。



重要

oc adm router を使用して作成されるルーター Pod には、ルーター Pod がデプロイされるためにノードが満たさなければならないデフォルトのリソース要求が設定されます。デフォルトのリソース要求は、インフラストラクチャーコンポーネントの信頼性を向上させる取り組みとして、ルーター Pod の QoS 層をリソース要求を持たない Pod よりも高く設定するために使用されます。デフォルト値は、基本的なルーターがデプロイされるのに必要な最小限のリソースを表しており、ルーターデプロイメント設定で編集できます。また、ルーターの負荷に基づいてそれらの値を引き上げることもできます。

3.2.2. ルーターの作成

ルーターが存在しない場合は、以下を実行してルーターを作成します。

```
$ oc adm router <router_name> --replicas=<number> --service-account=router
```

高可用性の設定が作成されない限り、**--replicas** は通常 **1** になります。

ルーターのホスト IP アドレスを見つけるには、以下を実行します。

```
$ oc get po <router-pod>  --template={{.status.hostIP}}
```

また、ルーターシャードを使用して、ルーターを特定の namespace またはルートに絞り込むか、ルーターの作成後に環境変数を設定できます。この場合には、シャードごとにルーターを作成します。

3.2.3. その他の基本ルーターコマンド

デフォルトルーターの確認

router という名前のデフォルトのルーターサービスアカウントは、クラスターのインストール時に自動的に作成されます。このアカウントがすでに存在することを確認するには、以下を実行します。

```
$ oc adm router --dry-run --service-account=router
```

デフォルトルーターの表示

デフォルトルーターが作成されている場合でそれを確認するには、以下を実行します。

```
$ oc adm router --dry-run -o yaml --service-account=router
```

ラベル付けされたノードへのルーターのデプロイ

指定されたノードラベルに一致するノードにルーターをデプロイするには、以下を実行します。

```
$ oc adm router <router_name> --replicas=<number> --selector=<label> \
    --service-account=router
```

たとえば、**router** という名前のルーターを作成し、それを **node-role.kubernetes.io/infra=true** とラベルが付けられたノードに配置するには、以下を実行します。

```
$ oc adm router router --replicas=1 --selector='node-
role.kubernetes.io/infra=true' \
  --service-account=router
```

クラスターのインストール時に、**openshift_router_selector** および **openshift_registry_selector** の Ansible 設定はデフォルトで **node-role.kubernetes.io/infra=true** に設定されます。デフォルトのルーターおよびレジストリーは、**node-role.kubernetes.io/infra=true** ラベルに一致するノードがある場合にのみ自動的にデプロイされます。

ラベルの更新に関する情報については、「Updating Labels on Nodes」を参照してください。

複数のインスタンスがスケジューラーポリシーに従って複数の異なるホストに作成されます。

複数の異なるルーターイメージの使用

複数の異なるルーターイメージを使用し、使用されるルーター設定を表示するには、以下を実行します。

```
$ oc adm router <router_name> -o <format> --images=<image> \
    --service-account=router
```

以下は例を示しています。

```
$ oc adm router region-west -o yaml --images=myrepo/somerouter:mytag \
    --service-account=router
```

3.2.4. ルートを特定のルーターに絞り込む

ROUTE_LABELS 環境変数を使用してルートを絞り込むことで、特定のルーターによってのみ使用されるようにできます。

たとえば、複数のルーターと 100 のルートがある場合、ラベルをルートに割り当てることで、それらの一部を 1 つのルーターで処理し、残りを別のルーターで処理するようにできます。

1. ルーターの作成後に、**ROUTE_LABELS** 環境変数を使用してルーターにタグ付けします。

```
$ oc env dc/<router=name>  ROUTE_LABELS="key=value"
```

2. ラベルを必要なルートに追加します。

```
oc label route <route=name> key=value
```

3. ラベルがルートに割り当てられていることを確認するには、ルートの設定をチェックします。

```
$ oc describe route/<route_name>
```

同時接続の最大数の設定

ルーターはデフォルトで最大 20000 の接続を処理できるようになっています。この上限は必要に応じて変更できます。接続が少なすぎると、ヘルスチェックが機能せず、不必要な再起動が実行されます。システムをこの接続の最大数に対応するように設定する必要があります。**'sysctl fs.nr_open'** と **'sysctl fs.file-max'** に示される上限は十分に大きな値である必要があります。そうでない場合には HAproxy は起動しません。

ルーターを作成したら、**--max-connections=** オプションで必要な上限を設定します。

```
$ oc adm router --max-connections=10000   ....
```

ルーターのデプロイメント設定で **ROUTER_MAX_CONNECTIONS** 環境変数を編集し、値を変更します。ルーター Pod は新しい値で再起動されます。**ROUTER_MAX_CONNECTIONS** が存在しない場合は、デフォルト値の 20000 が使用されます。

注記

接続にはフロントエンドおよび内部バックエンドが含まれます。これは 2 つの接続としてカウントされます。必ず **ROUTER_MAX_CONNECTIONS** の値を作成しようとしている接続数の 2 倍以上になるように設定してください。

3.2.5. HAProxy Strict SNI

HAProxy **strict-sni** は、ルーターのデプロイメント設定の **ROUTER_STRICT_SNI** 環境変数によって管理できます。また、これは **--strict-sni** コマンドラインオプションを使用してルーターを作成する時にも設定できます。

```
$ oc adm router --strict-sni
```

3.2.6. TLS 暗号化スイート

ルーターの作成時に **--ciphers** オプションを使用して、ルーターの暗号化スイートを設定します。

```
$ oc adm router --ciphers=modern   ....
```

値は **modern**、**intermediate**、または **old** で、デフォルトは **intermediate**です。または、 ":" で区切られた暗号セットを指定することもできます。暗号は以下によって表示されるセットから取られたものである必要があります。

```
$ openssl ciphers
```

また、既存のルーターには **ROUTER_CIPHERS** 環境変数を使用します。

3.2.7. 高可用性ルーター

IP フェイルオーバーを使用して OpenShift Container Platform クラスターで高可用性ルーターをセットアップできます。このセットアップには複数の異なるノードの複数のレプリカが含まれるため、フェイルオーバーソフトウェアは現在のレプリカが失敗したら別のレプリカに切り替えることができます。

3.2.8. ルーターサービスポートのカスタマイズ

環境変数の **ROUTER_SERVICE_HTTP_PORT** と **ROUTER_SERVICE_HTTPS_PORT** を設定することで、テンプレートルーターがバインドするサービスポートをカスタマイズできます。これは、テンプレートルーターを作成し、そのデプロイメント設定を編集することで実行できます。

以下の例では、**0** レプリカのルーターデプロイメントを作成し、ルーターサービス HTTP と HTTPS ポートをカスタマイズし、これを適切に (**1** レプリカに) スケーリングしています。

```
$ oc adm router --replicas=0 --ports='10080:10080,10443:10443' 1
$ oc set env dc/router ROUTER_SERVICE_HTTP_PORT=10080  \
                    ROUTER_SERVICE_HTTPS_PORT=10443
$ oc scale dc/router --replicas=1
```

1 公開されるポートがコンテナーネットワークモード **--host-network=false** を使用するルーターに対して適切に設定されていることを確認します。

重要

テンプレートルーターサービスポートをカスタマイズする場合は、ルーター Pod が実行されるノードのファイアウォールでカスタムポートが開いているようにする必要があります (Ansible または **iptables** のいずれか、または **firewall-cmd** で使用するその他のカスタム方法を使用します)。

以下は、カスタムルーターサービスポートを開くために **iptables** を使用した例です。

```
$ iptables -A INPUT -p tcp --dport 10080 -j ACCEPT
$ iptables -A INPUT -p tcp --dport 10443 -j ACCEPT
```

3.2.9. 複数ルーターの使用

管理者は同じ定義を使用して複数のルーターを作成し、同じルートのセットを提供することができます。各ルーターは複数の異なるノードに置かれ、異なる IP アドレスを持ちます。ネットワーク管理者は、各ノードに必要なトラフィックを送る必要があります。

複数のルーターをグループ化して、クラスター内や複数テナントでのルーティングの負荷を別のルーターまたはシャードに分散することができます。グループの各ルーターまたはシャードは、ルーターのセレクターに基づいてルートを許可します。管理者は **ROUTE_LABELS** を使用してクラスター全体でシャードを作成できます。ユーザーは **NAMESPACE_LABELS** を使用して namespace (プロジェクト) でシャードを作成できます。

3.2.10. デプロイメント設定へのノードセレクターの追加

特定のルーターを特定のノードにデプロイするには、2 つの手順を実行する必要があります。

1. ラベルを必要なノードに追加します。

```
$ oc label node 10.254.254.28 "router=first"
```

2. ノードセレクターをルーターのデプロイメント設定に追加します。

```
$ oc edit dc <deploymentConfigName>
```

template.spec.nodeSelector フィールドに、ラベルに対応するキーと値を追加します。

```
...
  template:
    metadata:
      creationTimestamp: null
      labels:
        router: router1
    spec:
      nodeSelector:        1
        router: "first"
...
```

1 **router=first** ラベルに対応するキーと値はそれぞれ **router** と **first** になります。

3.2.11. ルーターシャードの使用

ルーターのシャード化 により、**NAMESPACE_LABELS** と **ROUTE_LABELS** を使用してルーターの namespace とルートの絞り込みが実行されます。これにより、複数のルーターデプロイメントでルートのサブセットを分散させることができます。重複しないサブセットを使用することにより、ルートセットのパーティション設定を効果的に行うことができます。または、重複するルートのサブセットを構成する複数のシャードを定義することもできます。

デフォルトで、ルーターはすべてのプロジェクト (namespace) からすべてのルートを選択します。シャード化によってラベルがルートまたはルーターの namespace およびラベルセレクターに追加されます。各ルーターシャードは特定のラベルのセットで選択されるルーターを構成するか、または特定のラベルセレクターで選択される namespace に属します。

注記

ルーターサービスアカウントには [**cluster reader**] パーミッションセットを設定し、他の namespace のラベルにアクセスできるようにする必要があります。

ルーターのシャード化および DNS

外部 DNS サーバーは要求を必要なシャードにルートするために必要となるので、管理者はプロジェクトの各ルーターに個別の DNS エントリーを作成する必要があります。ルーターは不明なルートを別のルーターに転送することはありません。

以下は例を示しています。

- Router A はホスト 192.168.0.5 にあり、***.foo.com** のルートを持つ。
- Router B はホスト 192.168.1.9 にあり、***.example.com** のルートを持つ。

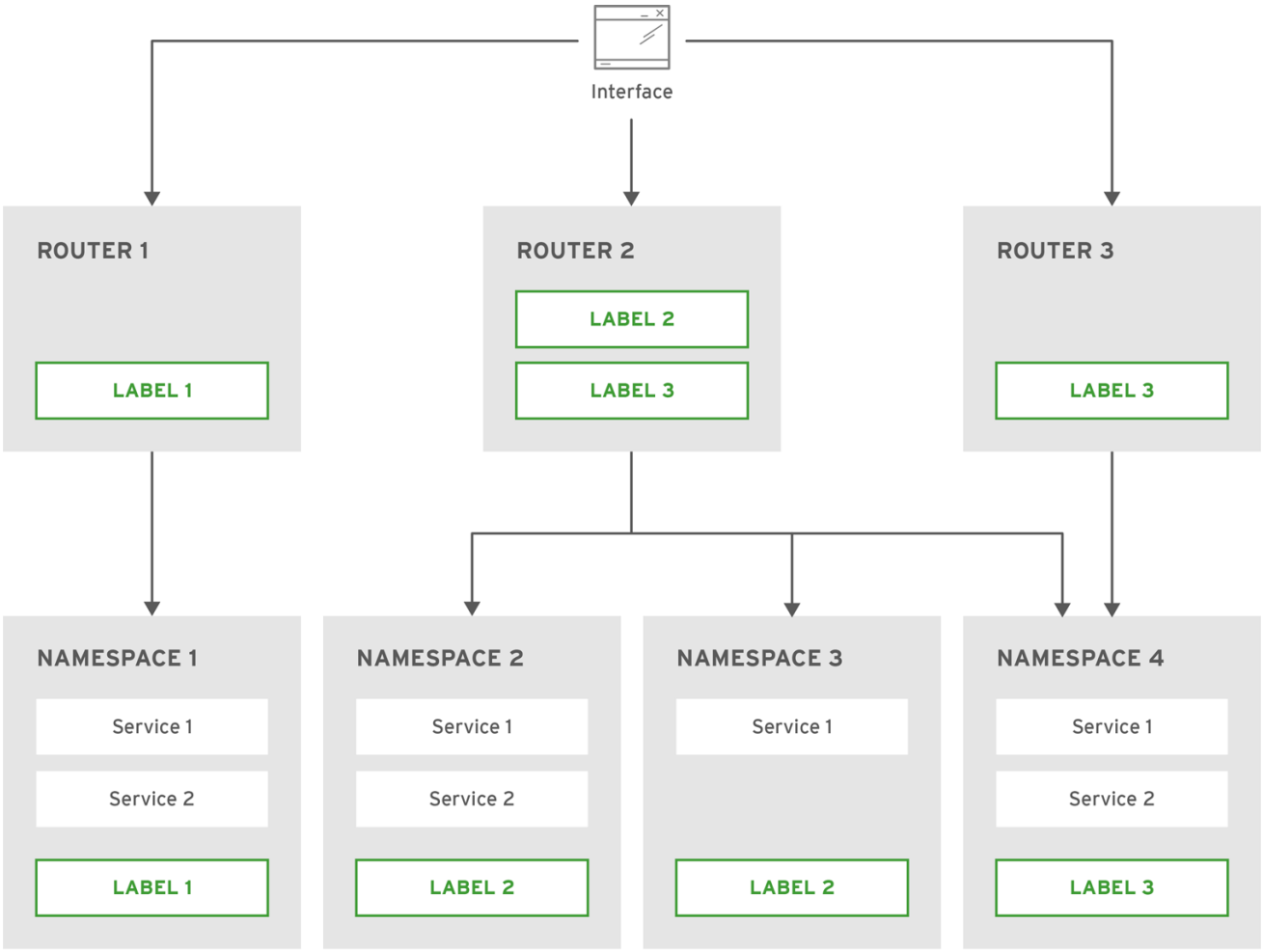
各 DNS エントリーは *.foo.com を Router A をホストするノードに解決し、*.example.com を Router B をホストするノードに解決する必要があります。

- ***.foo.com A IN 192.168.0.5**
- ***.example.com A IN 192.168.1.9**

ルーターのシャード化の例

このセクションでは、namespace およびルートラベルを使用するルーターのシャード化について説明します。

図3.1 namespace ラベルに基づくルーターのシャード化



1. namespace ラベルセレクターでルーターを設定します。

```
$ oc set env dc/router NAMESPACE_LABELS="router=r1"
```

2. ルーターには namespace にセレクターがあるため、ルーターは一致する namespace のルートのみを処理します。このセレクターが namespace に一致するようにするには、namespace に適宜ラベルを付けます。

```
$ oc label namespace default "router=r1"
```

3. ルートをデフォルトの namespace に作成すると、ルートはデフォルトのルーターで利用できるようになります。

```
$ oc create -f route1.yaml
```

4. 新規プロジェクト (namespace) を作成し、**route2** というルートを作成します。

```
$ oc new-project p1
$ oc create -f route2.yaml
```

ルートがルーターで利用できないことを確認します。

5. namespace **p1** に **router=r1** のラベルを付けます。

```
$ oc label namespace p1 "router=r1"
```

このラベルを追加すると、ルートはルーターで利用できるようになります。

例

ルーターのデプロイメント **finops-router** はルートセレクター **NAMESPACE_LABELS="name in (finance, ops)"** を使用して設定され、ルーターのデプロイメント **dev-router** はラベルセレクター **NAMESPACE_LABELS="name=dev"** を使用して設定されます。
すべてのルートが **name=finance**、**name=ops**、および **name=dev** というラベルの付けられた namespace にある場合、この設定により、2 つのルーターのデプロイメント間でルートが効果的に分散されます。

上記のシナリオでは、シャード化は重複するセットを持たないパーティション設定の特別なケースとなります。ルートは複数のルーターシャード間で分割されます。

ルート選択の基準によって、ルートの分散方法が決まります。複数のルーターデプロイメントに重複するルートのサブセットを設定することも可能です。

例

上記の例では **finops-router** と **dev-router** のほかに **devops-router** があり、これはラベルセレクター **NAMESPACE_LABELS="name in (dev, ops)"** を使用して設定されます。
name=dev または **name=ops** というラベルが付けられた namespace のルートは 2 つの異なるルーターデプロイメントによって提供されるようになりました。これは、「namespace ラベルに基づくルーターのシャード化」の手順で説明されているように、ルートの重複するサブセットを定義するケースです。

また、これによりさらに複雑なルーティングルールを作成し、優先度の高いトラフィックを専用の **finops-router** に転送し、優先度の低いトラフィックは **devops-router** に送信できます。

ルートのラベルに基づくルーターのシャード化

NAMESPACE_LABELS によって、提供するプロジェクトをフィルターでき、それらのプロジェクトからすべてのルートを選択できますが、ルートはルート自体に関連する他の基準に基づいてパーティション設定する必要がある場合があります。**ROUTE_LABELS** セレクターを使用すると、ルート自体を細かくフィルターできます。

例

ルーターデプロイメント **prod-router** はルートセレクター **ROUTE_LABELS="mydeployment=prod"** を使用して設定され、ルーターデプロイメント **devtest-router** はラベルセレクター **ROUTE_LABELS="mydeployment in (dev, test)"** を使用して設定されます。
この設定は、namespace の種類を問わず、ルートのラベルに基づいて 2 つのルーターデプロイメント間でルートのパーティション設定を行います。

この例では、提供されるルートすべてがラベル **"mydeployment=<tag>"** でタグ付けされていることを想定しています。

3.2.11.1. ルーターシャードの作成

このセクションでは、ルーターシャードのさらに詳細な例を示します。さまざまなラベルを持つ **a** — **z** という 26 のルートがあることを想定してください。

ルートで使用可能なラベル

```
sla=high        geo=east     hw=modest     dept=finance
sla=medium      geo=west     hw=strong     dept=dev
sla=low                                    dept=ops
```

これらのラベルは、サービスレベルアグリーメント、地理的な場所、ハードウェア要件、部門などの概念を表しています。ルートは各列のラベルを最大 1 つ持つことができます。ルートによっては他のラベルを持つこととも、ラベルをまったく持たないこともあります。

名前	SLA	Geo (地理的な場所)	HW	Dept (部門)	その他のラベル
a	high	east	modest	finance	type=static
b		west	strong		type=dynamic
c, d, e	low		modest		type=static
g — k	medium		strong	dev	
l — s	high		modest	ops	
t — z		west			type=dynamic

これは **oc adm router**、**oc set env**、**oc scale** がどのように連携してルーターシャードを作成するかを表している便利なスクリプト **mkshard** です。

```
#!/bin/bash
# Usage: mkshard ID SELECTION-EXPRESSION
id=$1
sel="$2"
router=router-shard-$id            (1)
oc adm router $router --replicas=0 (2)
dc=dc/router-shard-$id             (3)
oc set env   $dc ROUTE_LABELS="$sel" (4)
oc scale $dc --replicas=3          (5)
```

1. 作成されたルーターは **router-shard-<id>** という名前を持ちます。
2. ここではスケーリングを指定しません。
3. ルーターのデプロイメント設定。
4. **oc set env** を使用して選択式を設定します。選択式は環境変数 **ROUTE_LABELS** の値です。
5. 拡張します。

mkshard を複数回実行して、複数のルーターを作成します。

ルーター	選択式	ルート
router-shard-1	sla=high	a, l — s
router-shard-2	geo=west	b, t — z
router-shard-3	dept=dev	g — k

3.2.11.2. ルーターシャードの変更

ルーターシャードはラベルに基づいた構成なので、(**oc label** を使用して) ラベルまたは (**oc set env** を使用して) 選択式のいずれかを変更できます。

このセクションでは「ルーターシャードの作成」セクションで扱った例をさらに詳細に取り上げ、選択式の変更方法を示します。

これは、新規の選択式を使用できるよう既存のルーターを変更する便利なスクリプト **modshard** です。

```
#!/bin/bash
# Usage: modshard ID SELECTION-EXPRESSION...
id=$1
shift
router=router-shard-$id          1
dc=dc/$router                    2
oc scale $dc --replicas=0        3
oc set env   $dc "$@"                4
oc scale $dc --replicas=3        5
```

1. 変更後のルーターの名前は **router-shard-<id>** になります。
2. 変更が発生するデプロイメント設定。
3. 縮小します。
4. **oc set env** を使用して新しい選択式を設定します。「ルーターシャードの作成」セクションの **mkshard** とは異なり、**modshard** の **ID** 以外の引数として指定される選択式には環境変数名とその値が含まれている必要があります。
5. 拡大して元に戻します。

注記

modshard では、**router-shard-<id>** のデプロイメントストラテジーが**ローリング**の場合、**oc scale** コマンドは不要です。

たとえば **router-shard-3** の部門を拡張して **ops** と **dev** を含めるには、以下を実行します。

```
$ modshard 3 ROUTE_LABELS='dept in (dev, ops)'
```

結果として、**router-shard-3** はルート **g — s** (**g — k** と **l — s** の組み合わせ) を選択します。

この例ではシャードから除外する 1 つの部門を指定します。このシナリオ例では 3 つの部門しかないため、これによって前述の例と同じ結果が得られます。

```
$ modshard 3 ROUTE_LABELS='dept != finance'
```

この例は 3 つのコンマで区切られた属性を指定しており、結果としてルート **b** のみが選択されます。

```
$ modshard 3 ROUTE_LABELS='hw=strong,type=dynamic,geo=west'
```

ルートのラベルを使用する **ROUTE_LABELS** と同様に、**NAMESPACE_LABELS** 環境変数を使用して、ルートはルートの namespace ラベルのラベルに基づいて選択できます。この例では、ラベル **frequency=weekly** を持つルートの namespace を提供するように **router-shard-3** を変更します。

```
$ modshard 3 NAMESPACE_LABELS='frequency=weekly'
```

最後の例は **ROUTE_LABELS** と **NAMESPACE_LABELS** を組み合わせて、ラベル**sla=low** を持ち、ラベル **frequency=weekly** を持つ namespace のルート選択します。

```
$ modshard 3 \
    NAMESPACE_LABELS='frequency=weekly' \
    ROUTE_LABELS='sla=low'
```

3.2.12. ルーターのホスト名の検索

サービスを公開する際に、ユーザーは外部ユーザーがアプリケーションにアクセスするために使用する DNS 名からの同じルートを使用できます。外部ネットワークのネットワーク管理者は、ホスト名がルートを許可したルーター名に解決することを確認する必要があります。ユーザーはこのホスト名を指す CNAME を使用して DNS をセットアップできます。ただし、ユーザーはルーターのホスト名を知らない場合があり、その場合はクラスター管理者がホスト名を知らせることができます。

クラスター管理者は、**--router-canonical-hostname** オプションをルーター作成時のルーターの正規ホスト名で使用できます。以下は例になります。

```
# oc adm router myrouter --router-canonical-hostname="rtr.example.com"
```

これは、ルーターのホスト名を含む **ROUTER_CANONICAL_HOSTNAME** 環境変数をルーターのデプロイメント設定に作成します。

すでに存在しているルーターの場合、クラスター管理者はルーターのデプロイメント設定を編集し、**ROUTER_CANONICAL_HOSTNAME** 環境変数を追加します。

```
spec:
  template:
    spec:
      containers:
        - env:
          - name: ROUTER_CANONICAL_HOSTNAME
            value: rtr.example.com
```

ROUTER_CANONICAL_HOSTNAME 値は、ルートを許可したすべてのルーターのルートステータスに表示されます。ルートステータスはルーターがリロードされるたびに更新されます。

ユーザーがルートを作成すると、すべてのアクティブなルーターはそのルートを評価し、条件を満たしていればそのルートを許可します。**ROUTER_CANONICAL_HOSTNAME** 環境変数を定義するルーターがルートを許可すると、ルーターはルートステータスの **routerCanonicalHostname** フィールドに値を入力します。ユーザーはルートステータスを検証して、どのルーターがルートを許可したかを確認でき、ルーターを一覧から選択し、ネットワーク管理者に渡すルーターのホスト名を見つけることができます (該当する場合)。

```
status:
  ingress:
    conditions:
      lastTransitionTime: 2016-12-07T15:20:57Z
      status: "True"
      type: Admitted
      host: hello.in.mycloud.com
      routerCanonicalHostname: rtr.example.com
      routerName: myrouter
      wildcardPolicy: None
```

oc describe にはホスト名が含まれます (利用可能な場合)。

```
$ oc describe route/hello-route3
...
Requested Host: hello.in.mycloud.com exposed on router myroute (host
rtr.example.com) 12 minutes ago
```

上記の情報を使用して、ユーザーは DNS 管理者に対し、ルートのホスト **hello.in.mycloud.com** から CNAME をルーターの正規ホスト名 **rtr.example.com** に応じてセットアップするよう依頼できます。この結果として、**hello.in.mycloud.com** へのトラフィックがユーザーのアプリケーションに到達するようになります。

3.2.13. デフォルトのルーティングサブドメインのカスタマイズ

マスター設定ファイル (デフォルトでは **/etc/origin/master/master-config.yaml** ファイル) を変更することで、お使いの環境のデフォルトルーティングサブドメインとして使用されるサフィックスをカスタマイズできます。ホスト名を指定しないルートの場合、このデフォルトのルーティングサブドメインを使用してホスト名が生成されます。

以下の例は、設定されたサフィックスを **v3.openshift.test** に設定する方法を示しています。

```
routingConfig:
  subdomain: v3.openshift.test
```

注記

この変更には、マスターを実行している場合は再起動が必要となります。

OpenShift Container Platform マスターが上記の設定を実行している場合、 namespace の **mynamespace** に追加されるホスト名を持たない **no-route-hostname** というルートの例では、生成されるホスト名は以下のようになります。

```
no-route-hostname-mynamespace.v3.openshift.test
```

3.2.14. カスタムルーティングサブドメインへのルートホスト名の強制

管理者がすべてのルートを特定のルーティングサブドメインに限定する場合、**--force-subdomain** オプションを **oc adm router** コマンドに渡すことができます。これはルートで指定されたホスト名を上書きし、**--force-subdomain** オプションに提供されるテンプレートに基づいてホスト名を生成するようルーターに強制します。

以下の例ではルーターを実行し、カスタムサブドメインテンプレート **${name}-${namespace}.apps.example.com** を使用してルートホスト名を上書きしています。

```
$ oc adm router --force-subdomain='${name}-${namespace}.apps.example.com'
```

3.2.15. ワイルドカード証明書の使用

証明書を含まない TLS 対応のルートはルーターのデフォルト証明書を代わりに使用します。ほとんどの場合、この証明書は信頼された認証局から提供されますが、利便性を考慮して OpenShift Container Platform CA を使用して証明書を作成することができます。以下は例になります。

```
$ CA=/etc/origin/master
$ oc adm ca create-server-cert --signer-cert=$CA/ca.crt \
      --signer-key=$CA/ca.key --signer-serial=$CA/ca.serial.txt \
      --hostnames='*.cloudapps.example.com' \
      --cert=cloudapps.crt --key=cloudapps.key
```

注記

oc adm ca create-server-cert コマンドは、2 年間有効な証明書を生成します。この期間は **--expire-days** オプションを使って変更することができますが、セキュリティー上の理由から、値をこれより大きくすることは推奨されません。

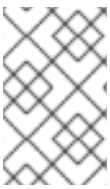
Ansible ホストインベントリーファイル (デフォルトで **/etc/ansible/hosts**) に最初に一覧表示されているマスターから **oc adm** コマンドを実行します。

ルーターは、証明書とキーが単一ファイルに PEM 形式で入力されていると予想します。

```
$ cat cloudapps.crt cloudapps.key $CA/ca.crt > cloudapps.router.pem
```

ここで **--default-cert** フラグを使用できます。

```
$ oc adm router --default-cert=cloudapps.router.pem --service-
account=router
```



注記

ブラウザーは、ワイルドカードを 1 つ深いレベルのサブドメインに有効であると見なします。この例では、証明書は **a.cloudapps.example.com** に対して有効ですが、**a.b.cloudapps.example.com** には有効ではありません。

3.2.16. 証明書を手動で再デプロイする

ルーター証明書を手動で再デプロイするには、以下を実行します。

1. デフォルトのルーター証明書を含むシークレットがルーターに追加されているかどうかを確認します。

```
$ oc volumes dc/router

deploymentconfigs/router
  secret/router-certs as server-certificate
    mounted at /etc/pki/tls/private
```

   証明書が追加されている場合は、以下の手順を省略してシークレットを上書きします。

2. デフォルト証明書ディレクトリーが以下の変数 **DEFAULT_CERTIFICATE_DIR** に設定されていることを確認します。

```
$ oc env dc/router --list

DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
```

   設定されていない場合は、以下のコマンドを使用してディレクトリーを作成します。

```
$ oc env dc/router DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
```

3. 証明書を PEM 形式にエクスポートします。

```
$ cat custom-router.key custom-router.crt custom-ca.crt > custom-
router.crt
```

4. ルーター証明書シークレットを上書きするか、またはこれを作成します。
   証明書シークレットがルーターに追加されている場合は、シークレットを上書きします。追加されていなければ、新規シークレットを作成します。

   シークレットを上書きするには、以下のコマンドを実行します。

```
$ oc create secret generic router-certs --from-file=tls.crt=custom-
router.crt --from-file=tls.key=custom-router.key --
type=kubernetes.io/tls -o json | oc replace -f -
```

   新規シークレットを作成するには、以下のコマンドを実行します。

```
$ oc create secret generic router-certs --from-file=tls.crt=custom-
router.crt --from-file=tls.key=custom-router.key --
type=kubernetes.io/tls

$ oc volume dc/router --add --mount-path=/etc/pki/tls/private --
secret-name='router-certs' --name router-certs
```

5. ルーターをデプロイします。

```
$ oc rollout latest dc/router
```

3.2.17. セキュリティー保護されたルートの使用

現時点で、パスワードで保護されたキーファイルはサポートされていません。HAProxy は開始時にパスワードのプロンプトを出しますが、このプロセスを自動化する方法は提供しません。キーファイルからパスフレーズを削除するには、以下を実行できます。

```
# openssl rsa -in <passwordProtectedKey.key> -out <new.key>
```

以下の例は、トラフィックが宛先にプロキシー処理される前に TLS 終端がルーターで生じる場合にセキュアな edge termination ルートを使用する方法を示しています。セキュアな edge termination ルートは TLS 証明書とキー情報を指定します。TLS 証明書は、ルーターのフロントエンドで提供されます。

最初にルーターインスタンスを起動します。

```
# oc adm router --replicas=1 --service-account=router
```

次に、セキュアな edge ルートのプライベートキー、CSR、証明書を作成します。この手順はお使いの認証局やプロバイダーによって異なります。**www.example.test** というドメインの単純な自己署名証明書の場合は、以下の例を参照してください。

```
# sudo openssl genrsa -out example-test.key 2048
#
# sudo openssl req -new -key example-test.key -out example-test.csr  \
  -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=www.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr  \
      -signkey example-test.key -out example-test.crt
```

上記の証明書とキーを使用してルートを生成します。

```
$ oc create route edge --service=my-service \
    --hostname=www.example.test \
    --key=example-test.key --cert=example-test.crt
route "my-service" created
```

その定義を確認します。

```
$ oc get route/my-service -o yaml
apiVersion: v1
kind: Route
metadata:
  name:  my-service
spec:
  host: www.example.test
  to:
    kind: Service
    name: my-service
  tls:
    termination: edge
    key: |
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |
```

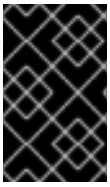
```
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

www.example.test の DNS エントリーがルーターインスタンスを指し、ドメインへのルートが利用できることを確認します。以下の例では、Curl をローカルリゾルバーと共に使用して DNS ルックアップのシミュレーションを行っています。

```
# routerip="4.1.1.1"  #  replace with IP address of one of your router
instances.
# curl -k --resolve www.example.test:443:$routerip
https://www.example.test/
```

3.2.18. (サブドメインの) ワイルドカードルートの使用

HAProxy ルーターはワイルドカードルートをサポートしており、**ROUTER_ALLOW_WILDCARD_ROUTES** 環境変数を **true** に設定することでこれを有効にできます。ルーター許可のチェックをパスする **Subdomain** のワイルドカードポリシーを持つすべてのルートは HAProxy ルーターによって提供されます。次に、HAProxy ルーターはルートのワイルドカードポリシーに基づいて (ルートの) 関連サービスを公開します。



重要

ルートのワイルドカードポリシーを変更するには、ルートを削除してから更新されたワイルドカードポリシーでこれを再作成する必要があります。ルートの **.yaml** ファイルでルートのワイルドカードポリシーのみを編集しても機能しません。

```
$ oc adm router --replicas=0 ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
$ oc scale dc/router --replicas=1
```

Web コンソールでワイルドカードルートを設定する方法についてはこちらを参照してください。

セキュアなワイルドカード edge termination ルートの使用

以下の例では、トラフィックが宛先にプロキシー処理される前にルーターで生じる TLS 終端を反映しています。サブドメイン **example.org** (***.example.org**) のホストに送られるトラフィックは公開されるサービスにプロキシーされます。

セキュアな edge termination ルートは TLS 証明書とキー情報を指定します。TLS 証明書は、サブドメイン (***.example.org**) に一致するすべてのホストのルーターのフロントエンドによって提供されます。

1. ルーターインスタンスを起動します。

   ```
   $ oc adm router --replicas=0 --service-account=router
   $ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
   $ oc scale dc/router --replicas=1
   ```

2. セキュリティー保護された edge ルートについてのプライベートキー、証明書署名要求 (CSR) および証明書を作成します。
   この手順はお使いの認証局やプロバイダーによって異なります。***.example.test** というドメインの単純な自己署名証明書の場合はこの例を参照してください。

```
# sudo openssl genrsa -out example-test.key 2048
#
# sudo openssl req -new -key example-test.key -out example-test.csr
\
  -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=*.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr  \
      -signkey example-test.key -out example-test.crt
```

3. 上記の証明書とキーを使用してワイルドカードのルートを生成します。

```
$ cat > route.yaml  <<REOF
apiVersion: v1
kind: Route
metadata:
  name:  my-service
spec:
  host: www.example.test
  wildcardPolicy: Subdomain
  to:
    kind: Service
    name: my-service
  tls:
    termination: edge
    key: "$(perl -pe 's/\n/\\n/' example-test.key)"
    certificate: "$(perl -pe 's/\n/\\n/' example-test.cert)"
REOF
$ oc create -f route.yaml
```

***.example.test** の DNS エントリーがお使いのルーターインスタンスを指し、ドメインへのルートが利用できることを確認します。

この例では **curl** をローカルリゾルバーと共に使用し、DNS ルックアップのシミュレーションを行います。

```
# routerip="4.1.1.1"  #  replace with IP address of one of your
router instances.
# curl -k --resolve www.example.test:443:$routerip
https://www.example.test/
# curl -k --resolve abc.example.test:443:$routerip
https://abc.example.test/
# curl -k --resolve anyname.example.test:443:$routerip
https://anyname.example.test/
```

ワイルドカードルートを許可しているルーター (**ROUTER_ALLOW_WILDCARD_ROUTES** を **true** に設定する) の場合、ワイルドカードルートに関連付けられたサブドメインの所有権についてのいくつかの注意点があります。

ワイルドカードルートの設定前に、所有権は、最も古いルートを持つ namespace のホスト名についての要求に基づいて設定されました (これはその他の要求を行うルートよりも優先されました)。たとえば、ルート **r1** がルート **r2** より古い場合、**one.example.test** の要求を持つ namespace **ns1** のルート **r1** は同じホスト名 **one.example.test** について namespace **ns2** のルート **ns2** よりも優先されます。

さらに、他の namespace のルートは重複しないホスト名を要求することを許可されていました。たとえば、namespace **ns1** のルート **rone** は **www.example.test** を要求でき、namespace **d2** の別のルート **rtwo** は **c3po.example.test** を要求できました。

これは、同じサブドメイン (上記の例では **example.test**) を要求するワイルドカードルートが**ない**場合には同様になります。

ただし、ワイルドカードルートはサブドメイン内のホスト名 (***.example.test** 形式のホスト名) をすべて要求する必要があります。ワイルドカードルートの要求は、そのサブドメイン (**example.test**) の最も古いルートがワイルドカードルートと同じ namespace 内にあるかどうかによって許可または拒否されます。最も古いルートは通常のルートまたはワイルドカードルートのいずれかになります。

たとえば、ホスト **owner.example.test** を要求する **最も古い** ルートが namespace **ns1** にすでに存在し、後からそのサブドメイン (**example.test**) のルートを要求する新規のワイルドカードルート **wildthing** が追加される場合、そのワイルドカードルートによる要求は、そのルートが所有ルートと同じ namespace (**ns1**) にある場合に**のみ**許可されます。

以下の例では、ワイルドカードルートの要求が成功する場合と失敗する場合のさまざまなシナリオを示しています。

以下の例では、ワイルドカードルートを許可するルーターは、ワイルドカードルートがサブドメインを要求していない限り、サブドメイン **example.test** のホストに対する重複しない要求を許可します。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test
$ oc expose service myservice --hostname=bname.example.test

$ oc project ns2
$ oc expose service anotherservice --hostname=second.example.test
$ oc expose service anotherservice --hostname=cname.example.test

$ oc project otherns
$ oc expose service thirdservice --hostname=emmy.example.test
$ oc expose service thirdservice --hostname=webby.example.test
```

以下の例では、ワイルドカードルートを許可するルーターは、所有している namespace が **ns1** なので、**owner.example.test** または **aname.example.test** の要求を許可しません。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test

$ oc project ns2
$ oc expose service secondservice --hostname=bname.example.test
$ oc expose service secondservice --hostname=cname.example.test

$ # Router will not allow this claim with a different path name `/p1` as
$ # namespace `ns1` has an older route claiming host `aname.example.test`.
```

```
$ oc expose service secondservice --hostname=aname.example.test --
path="/p1"

$ # Router will not allow this claim as namespace `ns1` has an older route
$ # claiming host name `owner.example.test`.
$ oc expose service secondservice --hostname=owner.example.test

$ oc project otherns

$ # Router will not allow this claim as namespace `ns1` has an older route
$ # claiming host name `aname.example.test`.
$ oc expose service thirdservice --hostname=aname.example.test
```

以下の例では、ワイルドカードルートを許可するルーターは、所有している namespace が **ns1** で、そのワイルドカードルートが同じ namespace に属しているので、**`*.example.test** の要求を許可します。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  router will allow this claim.
```

以下の例では、ワイルドカードルートを許可するルーターは、所有している namespace が **ns1** で、ワイルドカードルートが別の namespace **cyclone** に属するため、**`*.example.test** の要求を許可しません。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Switch to a different namespace/project.
$ oc project cyclone

$ # Reusing the route.yaml from a prior example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  router will deny (_NOT_ allow) this claim.
```

同様に、ワイルドカードルートを持つ namespace がサブドメインを要求すると、その namespace 内のルートのみがその同じサブドメインでホストを要求できます。

以下の例では、ワイルドカードルートを持つ namespace **ns1** のルートがサブドメイン **example.test** を要求すると、namespace **ns1** 内のルートのみがその同じサブドメインのホストを要求することを許可されます。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ oc project otherns

$ # namespace `otherns` is allowed to claim for other.example.test
$ oc expose service otherservice --hostname=other.example.test

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  Router will allow this claim.

$ #  In addition, route in namespace otherns will lose its claim to host
$ #  `other.example.test` due to the wildcard route claiming the
subdomain.

$ # namespace `ns1` is allowed to claim for deux.example.test
$ oc expose service mysecondservice --hostname=deux.example.test

$ # namespace `ns1` is allowed to claim for deux.example.test with path
/p1
$ oc expose service mythirdservice --hostname=deux.example.test --
path="/p1"

$ oc project otherns

$ # namespace `otherns` is not allowed to claim for deux.example.test
$ # with a different path '/otherpath'
$ oc expose service otherservice --hostname=deux.example.test --
path="/otherpath"

$ # namespace `otherns` is not allowed to claim for owner.example.test
$ oc expose service yetanotherservice --hostname=owner.example.test

$ # namespace `otherns` is not allowed to claim for unclaimed.example.test
$ oc expose service yetanotherservice --hostname=unclaimed.example.test
```

以下の例では、所有権のあるルートが削除され、所有権が namespace 内または namespace 間で渡されるさまざまなシナリオが示されています。namespace **ns1** のホスト **eldest.example.test** を要求するルートが存在する場合、その namespace 内のワイルドカードルートはサブドメイン **example.test** を要求できます。ホスト **eldest.example.test** のルートが削除されると、次に古

いルート **senior.example.test** が最も古いルートになりますが、これは他のルートに影響を与えません。ホスト **senior.example.test** のルートが削除されると、次に古いルート **junior.example.test** が最も古いルートになり、ワイルドカードルートの要求をブロックします。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=eldest.example.test
$ oc expose service seniorservice --hostname=senior.example.test

$ oc project otherns

$ # namespace `otherns` is allowed to claim for other.example.test
$ oc expose service juniorservice --hostname=junior.example.test

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ #   host: www.example.test
$ #   wildcardPolicy: Subdomain

$ oc create -f route.yaml   #  Router will allow this claim.

$ #  In addition, route in namespace otherns will lose its claim to host
$ #  `junior.example.test` due to the wildcard route claiming the
subdomain.

$ # namespace `ns1` is allowed to claim for dos.example.test
$ oc expose service mysecondservice --hostname=dos.example.test

$ # Delete route for host `eldest.example.test`, the next oldest route is
$ # the one claiming `senior.example.test`, so route claims are
unaffacted.
$ oc delete route myservice

$ # Delete route for host `senior.example.test`, the next oldest route is
$ # the one claiming `junior.example.test` in another namespace, so claims
$ # for a wildcard route would be affected. The route for the host
$ # `dos.example.test` would be unaffected as there are no other wildcard
$ # claimants blocking it.
$ oc delete route seniorservice
```

3.2.19. コンテナーネットワークスタックの使用

OpenShift Container Platform ルーターはコンテナー内で実行され、デフォルトの動作として、ホスト (例: ルーターコンテナーが実行されるノードなど) のネットワークスタックを使用します。このデフォルトの動作には、リモートクライアントからのネットワークトラフィックがターゲットサービスとコンテナーに到達するためにユーザー空間で複数のホップを使用する必要がないので、パフォーマンス上のメリットがあります。

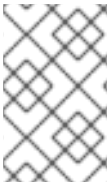
さらに、このデフォルト動作によってルーターはノードの IP アドレスではなくリモート接続の実際のソース IP アドレスを取得できます。これは、発信元の IP に基づいて ingress ルールを定義し、スティッキーセッションをサポートし、他に使用されているものの中でトラフィックを監視するのに役立

ちます。

このホストネットワークの動作は **--host-network** ルーターコマンドラインオプションによって制御でき、デフォルトの動作は **--host-network=true** を使用した場合と等しくなります。コンテナーネットワークスタックを使用してルーターを実行する場合は、ルーターを作成する際に **--host-network=false** オプションを使用します。以下は例になります。

```
$ oc adm router --service-account=router --host-network=false
```

内部的には、これは外部ネットワークがルーターと通信するために、ルーターコンテナーが 80 と 443 ポートを公開する必要があることを意味します。



注記

コンテナーネットワークスタックを使用して実行することで、ルーターは接続のソース IP アドレスを実際のリモート IP アドレスではなくノードの NAT された IP アドレスとして扱うことを意味します。

注記

マルチテナントネットワークの分離を使用する OpenShift Container Platform クラスターでは、**--host-network=false** オプションを指定したデフォルト以外の namespace のルーターはクラスターのすべてのルートを読み込みますが、ネットワークの分離により複数の namespace にあるルートには到達できません。**--host-network=true** オプションを指定すると、ルートはコンテナーネットワークをバイパスし、クラスターの任意の Pod にアクセスできます。この場合、分離が必要な場合は、複数の namespace のルートを追加しないでください。

3.2.20. ルーターメトリクスの公開

HAProxy ルーターメトリクス は、外部メトリクス収集および集約システム (例: Prometheus、statsd) で使用されるようにデフォルトで Prometheus 形式で公開されます。メトリクスは独自の HTML 形式でブラウザーで閲覧したり CSV ダウンロードを実行するために HAProxy ルーターから直接利用することもできます。これらのメトリクスには、HAProxy ネイティブメトリクスおよび一部のコントローラーメトリクスが含まれます。

以下のコマンドを使用してルーターを作成する場合、OpenShift Container Platform は Prometheus 形式のメトリクスをデフォルトが 1936 の統計ポートで利用可能にします。

```
$ oc adm router --service-account=router
```

- Prometheus 形式で未加工統計を抽出するには、以下を実行します。

  ```
  curl <user>:<password>@<router_IP>:<STATS_PORT>
  ```

  以下は例を示しています。

  ```
  $ curl admin:sLzdR6SgDJ@10.254.254.35:1936/metrics
  ```

  メトリクスにアクセスするために必要な情報は、ルーターサービスのアノテーションで確認できます。

  ```
  $ oc edit router service <router-service-name>
  ```

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/port: "1936"
    prometheus.io/scrape: "true"
    prometheus.openshift.io/password: IImoDqON02
    prometheus.openshift.io/username: admin
```

prometheus.io/port はデフォルトが 1936 の統計ポートです。アクセスを許可するようファイウォールを設定する必要がある場合があります。直前のユーザー名およびパスワードを使用してメトリクスにアクセスします。パスは /**metrics** です。

```
$ curl <user>:<password>@<router_IP>:<STATS_PORT>
for example:
$ curl admin:sLzdR6SgDJ@10.254.254.35:1936/metrics
...
# HELP haproxy_backend_connections_total Total number of
connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default"
,route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default"
,route="hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default"
,route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked
and the current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming
bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming
bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="f
e_no_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="f
e_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-
registry-5-nk5fz",route="docker-
registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-
vkjqx",route="hello-route",server="10.130.0.90:8080",service="hello-
svc-1"} 0
...
```

- ブラウザーでメトリクスを取得するには、以下を実行します。

  1. 以下の環境変数をルーターデプロイメント設定ファイルから削除します。

     ```
     $ oc edit service router

     - name: ROUTER_LISTEN_ADDR
       value: 0.0.0.0:1936
     - name: ROUTER_METRICS_TYPE
       value: haproxy
     ```

  2. ブラウザーで以下の URL を使用して統計ウィンドウを起動します。ここでは、**STATS_PORT** 値はデフォルトで **1936** になります。

     ```
     http://admin:<Password>@<router_IP>:<STATS_PORT>
     ```

     ;csv を URL に追加して CVS 形式の統計を取得できます。

     以下は例を示しています。

     ```
     http://admin:<Password>@<router_IP>:1936;csv
     ```

     ルーター IP、admin 名、およびパスワードを取得するには、以下を実行します。

     ```
     oc describe pod <router_pod>
     ```

- メトリクス収集を制限するには、以下を実行します。

  ```
  $ oc adm router --service-account=router --stats-port=0
  ```

3.2.21. 再起動時の接続失敗の防止

プロキシーのリロード時にルーターに接続すると、接続が誤ったネットワークキューに入れられ、ドロップする可能性がわずかながらにあります。現在、この問題への対応が行われています。回避策として、**iptables** ルールをインストールしてリロード期間の接続を禁止することができます。ただしこれを実行すると、ホストで **iptables** を操作できるようルーターが昇格した権限で実行されることが必要になります。

また、リロード時に発生した接続は一時的に無視され、接続開始を再送信する必要があります。これにより接続にかかる時間は長くなりますが、接続エラーを防ぐことができます。

この防止に向けて、サービスアカウントを変更し、ルーターに環境変数を設定してルーターが **iptables** を使用するように設定します。

特権付き SCC の使用

ルーターの作成時に、ルーターが特権付き SCC を使用できるようにします。これにより、ルーターユーザーはノードに root 権限でコンテナーを作成できます。

```
$ oc adm policy add-scc-to-user privileged -z router
```

ルーターデプロイメント設定にパッチを適用し、特権付きコンテナーを作成する

ここで特権付きコンテナーを作成できます。次に、ルーターの必要な iptables ルールの設定を可能にする権限を使用するようルーターデプロイメント設定を行います。このパッチはルーターデプロイメント設定を変更して、作成されるコンテナーが特権付きコンテナーとして実行され (かつ適切な機能を取得し)、root として実行できるようにします。

```
$ oc patch dc router -p '{"spec":{"template":{"spec":{"containers":
[{"name":"router","securityContext":
{"privileged":true}}],"securityContext":{"runAsUser": 0}}}}}'
```

iptables を使用するようルーターを設定する

ルーターデプロイメント設定でオプションを設定します。

```
$ oc set env dc/router -c router DROP_SYN_DURING_RESTART=true
```

ルーターにデフォルト以外の名を使用した場合は、**dc/router** も変更する必要があります。

3.2.22. 大規模クラスターの ARP キャッシュのチューニング

(**net.ipv4.neigh.default.gc_thresh3** の値 (デフォルトで**65536**) を上回る) 多数のルート を持つ OpenShift Container Platform クラスターでは、ARP キャッシュでより多くのエントリーを許可するためにルーター Pod を実行するクラスターの各ノードで sysctl 変数のデフォルト値を増やす必要があります。

問題が発生している場合、以下のようなカーネルメッセージが表示されます。

```
[ 1738.811139] net_ratelimit: 1045 callbacks suppressed
[ 1743.823136] net_ratelimit: 293 callbacks suppressed
```

この問題が発生すると、**oc** コマンドは以下のエラーを出して失敗することがあります。

```
Unable to connect to the server: dial tcp: lookup <hostname> on <ip>:
<port>: write udp <ip>:<port>-><ip>:<port>: write: invalid argument
```

IPv4 の ARP エントリーの実際の量を確認するには、以下を実行します。

```
# ip -4 neigh show nud all | wc -l
```

数字が **net.ipv4.neigh.default.gc_thresh3** しきい値に近づき始めたら、値を増やします。以下を実行して現在値を取得します。

```
# sysctl net.ipv4.neigh.default.gc_thresh1
net.ipv4.neigh.default.gc_thresh1 = 128
# sysctl net.ipv4.neigh.default.gc_thresh2
net.ipv4.neigh.default.gc_thresh2 = 512
# sysctl net.ipv4.neigh.default.gc_thresh3
net.ipv4.neigh.default.gc_thresh3 = 1024
```

以下の sysctl は変数を OpenShift Container Platform の現在のデフォルト値に設定します。

```
# sysctl net.ipv4.neigh.default.gc_thresh1=8192
# sysctl net.ipv4.neigh.default.gc_thresh2=32768
# sysctl net.ipv4.neigh.default.gc_thresh3=65536
```

これらの設定を永続化するには、このドキュメントを参照してください。

3.2.23. DDoS 攻撃からの保護

timeout http-request をデフォルトの HAProxy ルーターイメージに追加して、分散型の denial-of-service (DDoS) 攻撃 (slowloris など) からデプロイメントを保護します。

```
# and the haproxy stats socket is available at /var/run/haproxy.stats
global
  stats socket ./haproxy.stats level admin

defaults
  option http-server-close
  mode http
  timeout http-request 5s
  timeout connect 5s 1
  timeout server 10s
  timeout client 30s
```

1 **timeout http-request** は最大 5 秒に設定されます。HAProxy は HTTP 要求全体の送信のための 5 秒をクライアントに対して指定します。この指定がないと、HAProxy はエラーを出して接続を切断します。

また、環境変数 **ROUTER_SLOWLORIS_TIMEOUT** が設定されている場合、クライアントが HTTP 要求全体を送信するためにかかる合計時間が制限されます。これが設定されていない場合、HAProxy は接続を切断します。

環境変数を設定することで、情報をルーターのデプロイメント設定の一部として取得でき、テンプレートを手動で変更することが不要になります。一方、HAProxy 設定を手動で追加すると、ルーター Pod の再ビルドとルーターテンプレートファイルの保守が必要になります。

アノテーションを使用して、以下を制限する機能を含む基本的な DDoS 保護を HAProxy テンプレートルーターに実装します。

- 同時 TCP 接続の数
- クライアントが TCP 接続を要求できるレート
- HTTP 要求を実行できるレート

アプリケーションによってはトラフィックのパターンが完全に異なる場合があるため、これらはルートごとに有効にされます。

表3.1 HAProxy テンプレートルーター設定

設定	説明
haproxy.router.openshift.io/rate-limit-connections	設定した内容を有効にします (**true** に設定した場合など)。
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	このルートの同じ IP アドレスで接続できる同時 TCP 接続の数。

設定	説明
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	クライアント IP で開くことができる TCP 接続の数。
haproxy.router.openshift.io/rate-limit-connections.rate-http	クライアント IP が 3 秒間で実行できる HTTP 要求の数。

3.3. カスタマイズされた HAPROXY ルーターのデプロイ

3.3.1. 概要

デフォルトの HAProxy ルーターは多数のユーザーのニーズを対応することを目的としています。ただし、このルーターは HAProxy のすべての機能を公開している訳ではありません。そのため、ユーザーはそれぞれのニーズに合わせてルーターを変更する必要があります。

新機能をアプリケーションバックエンド内で実装したり、現在の操作を変更する必要がある場合があります。ルータープラグインはこのカスタマイズを行うために必要なすべての機能を提供します。

ルーター Pod はテンプレートファイルを使用して必要な HAProxy 設定ファイルを作成します。テンプレートファイルは golang テンプレート です。テンプレートを処理する際に、ルーターはルーターのデプロイメント設定、許可されたルートのセット、一部のヘルパー機能などの OpenShift Container Platform 情報にアクセスします。

ルーター Pod が起動し、リロードされるたびに、HAProxy 設定ファイルが作成され、HAProxy が起動します。『HAProxy 設定マニュアル』には HAProxy のすべての機能と有効な設定ファイルの作成方法が記載されています。

configMap を使用して新規テンプレートをルーター Pod に追加することができます。この方法により、ルーターデプロイメント設定を変更して、**configMap** をルーター Pod のボリュームとしてマウントできます。 **TEMPLATE_FILE** 環境変数はルーター Pod のテンプレートファイルのフルパス名に設定されます。

または、カスタムルーターイメージをビルドし、ルーターの一部またはすべてをデプロイする際にこれを使用することができます。すべてのルーターが同じイメージを実行する必要はありません。これを実行するには、 **haproxy-template.config** ファイルを変更し、ルーターイメージを再ビルドします。新しいイメージはクラスターの Docker リポジトリーにプッシュされ、ルーターのデプロイメント設定の **image:** フィールドが新しい名前で更新されます。クラスターが更新されたら、イメージを再ビルドし、プッシュする必要があります。

いずれの場合でも、ルーター Pod はテンプレートファイルを使用して起動します。

3.3.2. ルーター設定テンプレートの取得

HAProxy テンプレートファイルはかなり大きく複雑です。一部を変更するのであれば、すべてを書き換えるよりも既存のテンプレートを変更する方が簡単です。マスターでルーターを実行し、ルーター Pod を参照することで実行中のルーターから **haproxy-config.template**ファイルを取得できます。

```
# oc get po
NAME                       READY     STATUS    RESTARTS   AGE
router-2-40fc3             1/1       Running   0          11d
```

```
# oc rsh router-2-40fc3 cat haproxy-config.template > haproxy-
config.template
# oc rsh router-2-40fc3 cat haproxy.config > haproxy.config
```

または、ルーターを実行しているノードにログオンします。

```
# docker run --rm --interactive=true --tty --entrypoint=cat \
    registry.access.redhat.com/openshift3/ose-haproxy-router:v3.7 haproxy-
config.template
```

イメージ名は **docker イメージ** から取られます。

この内容をカスタマイズされたテンプレートのベースとして使用するためにファイルに保存します。保存された **haproxy.config** は実際に実行されているものを示します。

3.3.3. ルーター設定テンプレートの変更

3.3.3.1. 背景

このテンプレートは golang テンプレートに基づいています。これは、ルーターのデプロイメント設定の環境変数や、以下に示す設定情報およびルーターが提供するヘルパー機能を参照することができます。

テンプレートファイルの構造は作成される HAProxy 設定ファイルを反映します。テンプレートの処理時に、**{{" something "}}** によって囲まれていないものはすべて設定ファイルに直接コピーされます。**{{" something "}}** で囲まれている部分は評価されます。生成されるテキスト (ある場合) は設定ファイルにコピーされます。

3.3.3.2. Go テンプレートアクション

define アクションは、処理されるテンプレートを含むファイルに名前を付けます。

```
{{define "/var/lib/haproxy/conf/haproxy.config"}}pipeline{{end}}
```

表3.2 テンプレートルーター関数

関数	意味
processEndpointsForAlias(alias ServiceAliasConfig, svc ServiceUnit, action string) []Endpoint	有効なエンドポイントの一覧を返します。アクションが「Shuffle」の場合、エンドポイントの順序はランダム化されます。
env(variable, default …string) string	Pod からの名前付き環境変数の取得を試行します。これが定義されていないか、または空の場合、オプションの 2 つ目の引数が返されます。それ以外の場合には空の文字列を返します。
matchPattern(pattern, s string) bool	1 つ目の引数は正規表現を含む文字列で、2 つ目の引数はテストに使用できる変数です。1 つ目の引数として提供される正規表現が 2 つ目の引数として提供される文字列と一致するかどうかを示すブール値を返します。

関数	意味
isInteger(s string) bool	指定された変数が整数かどうかを判別します。
firstMatch(s string, allowedValues …string) bool	所定の文字列を許可された文字列の一覧と比較します。左から右にスキャンし、最初の一致を返します。
matchValues(s string, allowedValues …string) bool	指定された文字列と許可された文字列の一覧を比較します。文字列が許可される値の場合は「true」を返します。それ以外の場合は、「false」を返します。
generateRouteRegexp(hostname, path string, wildcard bool) string	ルートホスト (とパス) に一致する正規表現を生成します。最初の引数はホスト名であり、2 つ目はパス、3 つ目はワイルドカードブール値です。
genCertificateHostName(hostname string, wildcard bool) string	証明書の提供/証明書のマッチングに使用するホスト名を生成します。1 つ目の引数はホスト名で、2 つ目はワイルドカードブール値です。
isTrue(s string) bool	指定された変数に「true」が含まれるかどうかを判別します。

これらの関数は、HAProxy テンプレートルータープラグインによって提供されます。

3.3.3.3. ルーターが提供する情報

このセクションでは、ルーターがテンプレートで利用可能にする OpenShift Container Platform の情報について説明しますす。ルーター設定パラメーターは HAProxy ルータープラグインに与えられるデータセットです。フィールドには **(dot) .Fieldname** を使用してアクセスします。

以下のルーター設定パラメーター表は各種フィールドの定義を詳しく取り上げています。特に **.State** には許可されたルートセットが設定されます。

表3.3 ルーター設定パラメーター

フィールド	種類	説明
WorkingDir	string	ファイルが書き込まれるディレクトリー。デフォルトは **/var/lib/containers/router** に設定されます。
State	**map[string] (ServiceAliasConfig)`**	ルート。
ServiceUnits	**map[string]ServiceUnit**	サービスのルックアップ。
DefaultCertificate	string	pem 形式のデフォルト証明書へのフルパス名。

フィールド	種類	説明
PeerEndpoints	**`[]Endpoint**	ピア。
StatsUser	string	統計の公開に使用するユーザー名 (テンプレートがサポートしている場合)。
StatsPassword	string	統計の公開に使用するパスワード (テンプレートがサポートしている場合)。
StatsPort	int	統計の公開に使用するポート (テンプレートがサポートしている場合)。
BindPorts	bool	ルーターがデフォルトポートをバインドすべきかどうか。

表3.4 ルーター ServiceAliasConfig (Route)

フィールド	種類	説明
Name	string	ルートのユーザー固有の名前。
Namespace	string	ルートの namespace。
Host	string	ホスト名。例: **www.example.com**。
Path	string	オプションのパス。例: **www.example.com/myservice** (ここでは、**myservice** がパスになります)。
TLSTermination	**routeapi.TLSTerminationType**	このバックエンドの終端ポリシー。マッピングファイルとルーター設定を利用します。
Certificates	**map[string]Certificate**	このバックエンドをセキュリティー保護するために使用する証明書。証明書 ID で指定します。
Status	**ServiceAliasConfigStatus**	永続化する必要がある設定のステータスを示します。
PreferPort	string	ユーザーが公開したいポートを示します。空の場合、サービスのポートが選択されます。

フィールド	種類	説明
InsecureEdgeTerminationPolicy	**routeapi.InsecureEdgeTerminationPolicyType**	edge termination ルートへの非セキュアな接続の必要な動作を示します。 **none** (または **disable**)、**allow**、または **redirect**
RoutingKeyName	string	Cookie ID を隠すために使用するルート + namespace 名のハッシュ。
IsWildcard	bool	このサービスユニットが ワイルドカードサポートを必要とすることを示します。
Annotations	**map[string]string**	このルートに割り当てられるアノテーション。
ServiceUnitNames	**map[string]int32**	このルートをサポートするサービスコレクション。マップの他のエントリーに関連してサービス名で指定され、これに割り当てられた重みで評価されます。
ActiveServiceUnits	int	ゼロ以外の重みを持つ **ServiceUnitNames** の数。

ServiceAliasConfig はサービスのルートです。ホスト + パスによって特定されます。デフォルトのテンプレートは **{{range $cfgIdx, $cfg := .State }}** を使用してルートを繰り返し処理します。その **{{range}}** ブロック内で、テンプレートは **$cfg.Field** を使用して現在の **ServiceAliasConfig** のフィールドを参照できます。

表3.5 ルーター ServiceUnit

フィールド	種類	説明
Name	string	名前はサービス名 + namespace に対応します。**ServiceUnit** を特定します。
EndpointTable	**[]Endpoint**	サービスをサポートするエンドポイント。これはルーターの最終のバックエンド実装に変換されます。

ServiceUnit はサービスをカプセル化したものであり、そのサービスをサポートするエンドポイントであり、またサービスを参照するルートです。これは、ルーター設定ファイルの作成を進めるデータです。

表3.6 ルーターエンドポイント

フィールド	種類
ID	string
IP	string
Port	string
TargetName	string
PortName	string
IdHash	string
NoHealthCheck	bool

Endpoint は、Kubernetes エンドポイントの内部表現です。

表3.7 ルーター証明書、ServiceAliasConfigStatus

フィールド	種類	説明
Certificate	string	パブリック/プライベートキーのペアを表します。これは ID で識別され、ファイル名になります。CA 証明書には **PrivateKey** が設定されません。
ServiceAliasConfigStatus	string	この設定に必要なファイルがディスクに永続化されていることを示します。有効な値の例: "saved" または "" (ブランク)。

表3.8 ルーター証明書タイプ

フィールド	種類	説明
ID	string	
Contents	string	証明書。
PrivateKey	string	プライベートキー。

表3.9 ルーター TLSTerminationType

フィールド	種類	説明
TLSTerminationType	string	セキュアな通信が停止する場合を指示します。
InsecureEdgeTerminationPolicyType	string	ルートへの非セキュアな接続の必要な動作を示します。各ルーターは公開するポートを独自に決定することがありますが、通常はポート 80 になります。

TLSTerminationType と **InsecureEdgeTerminationPolicyType** はセキュアな通信が停止する場合を指示します。

表3.10 ルーター TLSTerminationType 値

定数	値	意味
TLSTerminationEdge	**edge**	edge ルーターでの暗号化を終了します。
TLSTerminationPassthrough	**passthrough**	宛先での暗号化を終了し、宛先でトラフィックを復号化します。
TLSTerminationReencrypt	**reencrypt**	edge ルーターで暗号化を終了し、送信先から提供される新規の証明書で再暗号化します。

表3.11 ルーター InsecureEdgeTerminationPolicyType 値

種類	意味
Allow	トラフィックは非セキュアなポートのサーバーに送信されます (デフォルト)。
Disable	トラフィックは非セキュアなポートでは許可されません。
Redirect	クライアントはセキュアなポートにリダイレクトされます。

なし (**""**) は **Disable** と同じです。

3.3.3.4. アノテーション

各ルートにはアノテーションが割り当てることができます。各アノテーションは名前と値のみで構成されます。

```
apiVersion: v1
```

```
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms
[...]
```

名前は既存のアノテーションと競合しないものにできます。値は文字列です。文字列では複数のトークンをスペースで区切ることができます (例: **aa bb cc**)。テンプレートは **{{index}}** を使用してアノテーションの値を抽出します。以下は例になります。

```
{{$balanceAlgo := index $cfg.Annotations
"haproxy.router.openshift.io/balance"}}
```

この例は、これをどのようにクライアントの相互承認に使用できるかを示しています。

```
{{ with $cnList := index $cfg.Annotations "whiteListCertCommonName" }}
  {{   if ne $cnList "" }}
    acl test ssl_c_s_dn(CN) -m str {{ $cnList }}
    http-request deny if !test
  {{   end }}
{{ end }}
```

このコマンドを使用してホワイトリストの CN を処理できます。

```
$ oc annotate route <route-name> --overwrite whiteListCertCommonName="CN1
CN2 CN3"
```

詳細は、「Route-specific Annotations」を参照してください。

3.3.3.5. 環境変数

テンプレートはルーター Pod に存在する任意の環境変数を使用できます。環境変数はデプロイメント設定に設定できます。また、新規の環境変数を追加できます。

これらは **env** 関数で参照されます。

```
{{env "ROUTER_MAX_CONNECTIONS" "20000"}}
```

1 つ目の文字列は変数であり、変数がないか、または **nil** の場合には 2 つ目の文字列がデフォルトになります。**ROUTER_MAX_CONNECTIONS** が設定されていないか、または **nil** の場合、20000 が使用されます。環境変数はマップと言えます。ここでキーは環境変数名で、内容は変数の値になります。

詳細は、「Route-specific Environment variables」を参照してください。

3.3.3.6. 使用例

以下で HAProxy テンプレートファイルに基づく単純なテンプレートを示します。

以下のコメントで開始します。

```
{{/*
  Here is a small example of how to work with templates
  taken from the HAProxy template file.
*/}}
```

テンプレートは任意の数の出力ファイルを作成できます。define コンストラクトを使用して出力ファイルを作成します。ファイル名は定義する引数として指定され、define ブロック内の一致する終了部分までのすべての情報がそのファイルの内容として書き込まれます。

```
{{ define "/var/lib/haproxy/conf/haproxy.config" }}
global
{{ end }}
```

上記は **global** を **/var/lib/haproxy/conf/haproxy.config** ファイルにコピーし、ファイルを閉じます。

ロギングを環境変数に基づいてセットアップします。

```
{{ with (env "ROUTER_SYSLOG_ADDRESS" "") }}
  log {{.}} {{env "ROUTER_LOG_FACILITY" "local1"}} {{env
"ROUTER_LOG_LEVEL" "warning"}}
{{ end }}
```

env 関数は、環境変数の値を抽出します。環境変数が定義されていないか、または **nil** の場合、2 つ目の引数が返されます。

with コンストラクトは with ブロック内の "." (ドット) の値を with に引数として提供される値に設定します。**with** アクションは Dot で **nil** をテストします。**nil** ではない場合、その句は**end** (終了部分) まで処理されます。上記では、**ROUTER_SYSLOG_ADDRESS** に **/var/log/msg** が含まれ、**ROUTER_LOG_FACILITY** が定義されておらず、**ROUTER_LOG_LEVEL** に **info** が含まれていると想定しています。以下が出力ファイルにコピーされます。

```
  log /var/log/msg local1 info
```

許可された各ルートは設定ファイルの行を生成します。**range** を使用して、許可されたルートを確認します。

```
{{ range $cfgIdx, $cfg := .State }}
  backend be_http_{{$cfgIdx}}
{{end}}
```

.State は **ServiceAliasConfig** のマップです。ここで、キーはルート名になります。**range** はマップを通じて、パスするたびに **key**を使用して **$cfgIdx** を設定し、`**$cfg** がルートを記述する **ServiceAliasConfig** をポイントするよう設定します。 **myroute** と **hisroute** という 2 つのルートがある場合、上記により以下が出力ファイルにコピーされます。

```
  backend be_http_myroute
  backend be_http_hisroute
```

ルートアノテーション **$cfg.Annotations** もマップであり、アノテーション名がキーとなり、コンテンツの文字列が値になっています。ルートは必要な数だけアノテーションを持つことができ、テンプレートの作成者が使用方法を定義します。ユーザーはアノテーションをルートにコード化し、テンプレート作成者は HAProxy テンプレートをカスタマイズしてそのアノテーションを処理できるようにします。

通常はアノテーションをインデックス化して値を取得します。

```
{{$balanceAlgo := index $cfg.Annotations
"haproxy.router.openshift.io/balance"}}
```

インデックスは指定されたアノテーションの値を抽出します。そのため、`**$balanceAlgo** はアノテーションまたは **nil** に関連付けられた文字列が含まれます。上記のように、**nil** 以外の文字列についてテストし、**with** コンストラクトを使用して影響を確認することができます。

```
{{ with $balanceAlgo }}
  balance $balanceAlgo
{{ end }}
```

$balanceAlgo が **nil** でない場合、**balance $balanceAlgo** は出力ファイルにコピーされます。

2 つ目の例では、アノテーションのタイムアウト値の設定に基づいてサーバータイムアウトを設定します。

```
$value := index $cfg.Annotations "haproxy.router.openshift.io/timeout"
```

$value を評価して、適切に作成された文字列が含まれていることを確認できます。**matchPattern** 関数は正規表現を受け入れ、引数が表現を満たしていれば **true** を返します。

```
matchPattern "[1-9][0-9]*(us\|ms\|s\|m\|h\|d)?" $value
```

これにより **5000ms** を受け入れますが、**7y** は受け取りません。この結果はテストで使用できます。

```
{{if (matchPattern "[1-9][0-9]*(us\|ms\|s\|m\|h\|d)?" $value) }}
  timeout server  {{$value}}
{{ end }}
```

これを使用してトークンに一致させることもできます。

```
matchPattern "roundrobin|leastconn|source" $balanceAlgo
```

または、**matchValues** を使用してトークンと一致させることもできます。

```
matchValues $balanceAlgo "roundrobin" "leastconn" "source"
```

3.3.4. ConfigMap を使用してルーター設定テンプレートを置き換える

ConfigMap を使用して、ルーターイメージを再ビルドすることなくルーターインスタンスをカスタマイズできます。ルーター環境変数の作成し、変更することができるだけでなく、**haproxy-config.template**、**reload-haproxy** その他のスクリプトを変更することもできます。

1. 上記のように変更する **haproxy-config.template** をコピーして、必要に応じて変更します。
2. ConfigMap を作成します。

```
$ oc create configmap customrouter --from-file=haproxy-
config.template
```

customrouter ConfigMap には変更された **haproxy-config.template** ファイルのコピーが含まれています。

3. ルーターデプロイメント設定を変更し、ConfigMap をファイルとしてマウントし、**TEMPLATE_FILE** 環境変数がこれをポイントするようにします。これは、**oc set env** と **oc volume** コマンドを使用するか、またはルーターデプロイメント設定を編集して実行できます。

oc コマンドの使用

```
$ oc volume dc/router --add --overwrite \
    --name=config-volume \
    --mount-path=/var/lib/haproxy/conf/custom \
    --source='{"configMap": { "name": "customrouter"}}'
$ oc set env dc/router \
    TEMPLATE_FILE=/var/lib/haproxy/conf/custom/haproxy-
config.template
```

ルーターデプロイメント設定の編集

oc edit dc router を使用して、テキストエディターでルーターデプロイメント設定を編集します。

```
...
        - name: STATS_USERNAME
          value: admin
        - name: TEMPLATE_FILE   1
          value: /var/lib/haproxy/conf/custom/haproxy-
config.template
        image: openshift/origin-haproxy-routerp
...
        terminationMessagePath: /dev/termination-log
        volumeMounts: 2
        - mountPath: /var/lib/haproxy/conf/custom
          name: config-volume
      dnsPolicy: ClusterFirst
...
      terminationGracePeriodSeconds: 30
      volumes: 3
      - configMap:
          name: customrouter
        name: config-volume
...
```

1. **spec.container.env** フィールドに **TEMPLATE_FILE** 環境変数を追加して、マウントされた **haproxy-config.template** ファイルをポイントするようにします。
2. **spec.container.volumeMounts** フィールドを追加して、マウントポイントを作成します。
3. 新しい **spec.volumes** フィールドを追加し、ConfigMap を示唆します。

変更を保存し、エディターを終了します。ルーターが再起動します。

3.3.5. Stick Table の使用

以下のカスタマイズの例を高可用なルーティングセットアップで使用することで、ピア間の同期を行う stick-table を使用できます。

ピアセクションの追加

ピア間で stick-table を同期するには、HAProxy 設定でピアセクションを定義する必要があります。このセクションによって、HAProxy がどのようにピアを識別し、接続するかが決まります。プラグインはデータを **.PeerEndpoints** 変数にあるテンプレートに提供するので、ルーターサービスのメンバーを簡単に識別できます。以下を追加することで、ピアセクションをルーターイメージ内の **haproxy-config.template** ファイルに追加することができます。

```
{{ if (len .PeerEndpoints) gt 0 }}
peers openshift_peers
  {{ range $endpointID, $endpoint := .PeerEndpoints }}
    peer {{$endpoint.TargetName}} {{$endpoint.IP}}:1937
  {{ end }}
{{ end }}
```

リロードスクリプトの変更

stick-table を使用する場合、HAProxy にピアセクションでローカルホスト名として見なすものをオプションで指示することができます。エンドポイントの作成時に、プラグインは **TargetName** をエンドポイントの **TargetRef.Name** の値に設定するよう試みます。**TargetRef** が設定されていない場合、**TargetName** は IP アドレスに設定されます。**TargetRef.Name** は Kubernetes ホスト名に対応しているので、**-L** オプションを **reload-haproxy** スクリプトに追加してローカルホストをピアセクションで識別できます。

```
peer_name=$HOSTNAME (1)

if [ -n "$old_pid" ]; then
  /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name -sf
$old_pid
else
  /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name
fi
```

(1) ピアセクションで使用されるエンドポイントターゲット名と一致している必要があります。

バックエンドの変更

最後に、stick-table をバックエンド内で使用するために、HAProxy 設定を変更して stick-table およびピアセットを使用することができます。以下は、stick-table を使用するように TCP 接続の既存のバックエンドを変更している例です。

```
            {{ if eq $cfg.TLSTermination "passthrough" }}
backend be_tcp_{{$cfgIdx}}
  balance leastconn
  timeout check 5000ms
  stick-table type ip size 1m expire 5m{{ if (len $.PeerEndpoints) gt 0 }}
peers openshift_peers {{ end }}
  stick on src
                {{ range $endpointID, $endpoint :=
$serviceUnit.EndpointTable }}
  server {{$endpointID}} {{$endpoint.IP}}:{{$endpoint.Port}} check inter
```

```
5000ms
                {{ end }}
            {{ end }}
```

この変更を行った後に、ルーターを再ビルドできます。

3.3.6. ルーターの再ビルド

ルーターを再ビルドするには、実行中のルーターにある複数のファイルのコピーが必要になります。作業ディレクトリーを作成し、ルーターからファイルをコピーします。

```
# mkdir - myrouter/conf
# cd myrouter
# oc get po
NAME                       READY     STATUS    RESTARTS   AGE
router-2-40fc3             1/1       Running   0          11d
# oc rsh router-2-40fc3 cat haproxy-config.template > conf/haproxy-
config.template
# oc rsh router-2-40fc3 cat error-page-503.http > conf/error-page-503.http
# oc rsh router-2-40fc3 cat default_pub_keys.pem >
conf/default_pub_keys.pem
# oc rsh router-2-40fc3 cat ../Dockerfile > Dockerfile
# oc rsh router-2-40fc3 cat ../reload-haproxy > reload-haproxy
```

これらのファイルのいずれも編集するか、または置き換えることができます。ただし、**conf/haproxy-config.template** と **reload-haproxy** が変更される可能性が高くなります。

ファイルの更新後:

```
# docker build -t openshift/origin-haproxy-router-myversion .
# docker tag openshift/origin-haproxy-router-myversion
172.30.243.98:5000/openshift/haproxy-router-myversion (1)
# docker push 172.30.243.98:5000/openshift/origin-haproxy-router-pc:latest
(2)
```

(1) このバージョンをリポジトリーでタグ付けします。この場合、リポジトリーは **172.30.243.98:5000** になります。

(2) タグ付けバージョンをリポジトリーにプッシュします。まずリポジトリーに **docker ログイン** する必要がある場合があります。

新規ルーターを使用するには、**image:** 文字列を変更するか、または **--images=<repo>/<image>:<tag>** フラグを **oc adm router** コマンドに追加してルーターデプロイ設定を編集します。

変更のデバッグ時に、デプロイメント設定で **imagePullPolicy: Always** を設定して各 Pod の作成時にイメージプルを強制的に実行すると便利です。デバッグが完了したら、これを **imagePullPolicy: IfNotPresent** に戻して各 Pod の起動時のプルを回避します。

3.4. HAPROXY ルーターを設定して PROXY プロトコルを使用する

3.4.1. 概要

デフォルトで HAProxy ルーターは、非セキュアな edge および re-encrypt ルートへの受信接続に HTTP を使用することを想定しています。ただし、PROXY プロトコルを使用することで、ルーターが受信要求を予想するよう設定することができます。このトピックでは、HAProxy ルーターと外部ロードバランサーを PROXY プロトコルを使用するように設定する方法を説明しています。

3.4.2. PROXY プロトコルを使用する理由

プロキシーサーバーやロードバランサーなどの中間サービスが HTTP 要求を転送する際に、これは接続のソースアドレスを要求の「Forwarded」ヘッダーに追加して、この情報を後続の中間サービスと、要求が最終的に転送されるバックエンドサービスに提供できます。ただし、接続が暗号化されている場合、中間サービスは「Forwarded」ヘッダーを変更できません。この場合、要求が転送されても HTTP ヘッダーは元のソースアドレスを正確に通信することができません。

この問題を解決するために、一部のロードバランサーは、単純に HTTP を転送する代替手段として PROXY プロトコルを使用して HTTP 要求をカプセル化します。カプセル化によって、ロードバランサーは転送される要求自体を変更することなく、情報を要求に追加することができます。これにより、ロードバランサーは、暗号化された接続を転送する場合でもソースアドレスを通信できます。

HAProxy ルーターが PROXY プロトコルを受け入れ、HTTP 要求のカプセル化を解除するように設定できます。ルーターは edge および re-encrypt ルートの暗号化を終了するので、ルーターは要求の「Forwarded」HTTP ヘッダー (および関連する HTTP ヘッダー) を更新でき、PROXY プロトコルを使用して通信されるソースアドレスを追加できます。

警告

PROXY プロトコルと HTTP は互換性がなく、組み合わせることはできません。ルーターの前にロードバランサーを使用する場合、これらがどちらも PROXY プロトコルまたは HTTP のいずれかを使用する必要があります。一方を PROXY プロトコルを使用するように設定し、他方を HTTP を使用するように設定すると、ルーティングが失敗します。

3.4.3. PROXY プロトコルの使用

デフォルトで、HAProxy ルーターは PROXY プロトコルを使用しません。**ROUTER_USE_PROXY_PROTOCOL** 環境変数を使用することで、ルーターが受信接続に PROXY プロコトルの使用を予想するように設定できます。

PROXY プロコトルの有効化

```
$ oc env dc/router ROUTER_USE_PROXY_PROTOCOL=true
```

変数を **true** または **TRUE** 以外の値に設定し、PROXY プロコトルを無効にします。

PROXY プロトコルの無効化

```
$ oc env dc/router ROUTER_USE_PROXY_PROTOCOL=false
```

ルーターで PROXY プロトコルを有効にする場合、ルーターの前のロードバランサーが PROXY プロコトルを使用するように設定する必要があります。以下は、Amazon の Elastic Load Balancer (ELB) サー

ビスを PROXY プロトコルを使用するように設定した例です。この例では、ELB がポート 80 (HTTP)、443 (HTTPS)、5000 (イメージレジストリーの場合) を 1 つ以上の EC2 インスタンスで実行されるルーターに転送することを想定しています。

Amazon ELB を設定して PROXY プロコトルを使用する

1. 後続の手順を単純化するために、最初に一部の Shell 変数を設定します。

```
$ lb='infra-lb' (1)
$ instances=( 'i-079b4096c654f563c' ) (2)
$ secgroups=( 'sg-e1760186' ) (3)
$ subnets=( 'subnet-cf57c596' ) (4)
```

(1) ELB の名前。

(2) ルーターが実行されているインスタンス。

(3) この ELB のセキュリティーグループ。

(4) この ELB のサブネット。

2. 次に、適切なリスナーやセキュリティーグループおよびサブネットを使用して ELB を作成します。

注記

すべてのリスナーが HTTP プロトコルではなく TCP プロトコルを使用するよう設定する必要があります。

```
$ aws elb create-load-balancer --load-balancer-name "$lb" \
   --listeners \

'Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=
80' \

'Protocol=TCP,LoadBalancerPort=443,InstanceProtocol=TCP,InstancePort
=443' \

'Protocol=TCP,LoadBalancerPort=5000,InstanceProtocol=TCP,InstancePor
t=5000' \
   --security-groups $secgroups \
   --subnets $subnets
{
    "DNSName": "infra-lb-2006263232.us-east-1.elb.amazonaws.com"
}
```

3. ルーターインスタンスを ELB に登録します。

```
$ aws elb register-instances-with-load-balancer --load-balancer-name
"$lb" \
   --instances $instances
{
    "Instances": [
```

```
        {
            "InstanceId": "i-079b4096c654f563c"
        }
    ]
}
```

4. ELB のヘルスチェックを設定します。

```
$ aws elb configure-health-check --load-balancer-name "$lb" \
   --health-check
'Target=HTTP:1936/healthz,Interval=30,UnhealthyThreshold=2,HealthyTh
reshold=2,Timeout=5'
{
    "HealthCheck": {
        "HealthyThreshold": 2,
        "Interval": 30,
        "Target": "HTTP:1936/healthz",
        "Timeout": 5,
        "UnhealthyThreshold": 2
    }
}
```

5. 最後に、**ProxyProtocol** 属性を有効にしたロードバランサーのポリシーを作成し、ELB の TCP ポート 80 および 443 でポリシーを設定します。

```
$ aws elb create-load-balancer-policy --load-balancer-name "$lb" \
   --policy-name "${lb}-ProxyProtocol-policy" \
   --policy-type-name 'ProxyProtocolPolicyType' \
   --policy-attributes
'AttributeName=ProxyProtocol,AttributeValue=true'
$ for port in 80 443
  do
    aws elb set-load-balancer-policies-for-backend-server \
     --load-balancer-name "$lb" \
     --instance-port "$port" \
     --policy-names "${lb}-ProxyProtocol-policy"
  done
```

設定の確認

ロードバランサーを以下のように検証して、設定が正しいことを確認します。

```
$ aws elb describe-load-balancers --load-balancer-name "$lb" |
    jq '.LoadBalancerDescriptions| [.[]|.ListenerDescriptions]'
[
  [
    {
      "Listener": {
        "InstancePort": 80,
        "LoadBalancerPort": 80,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": ["infra-lb-ProxyProtocol-policy"] (1)
    },
```

```
    {
      "Listener": {
        "InstancePort": 443,
        "LoadBalancerPort": 443,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": ["infra-lb-ProxyProtocol-policy"] 2
    },
    {
      "Listener": {
        "InstancePort": 5000,
        "LoadBalancerPort": 5000,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": [] 3
    }
  ]
]
```

1 TCP ポート 80 のリスナーには PROXY プロトコルを使用するポリシーが設定されている必要があります。

2 TCP ポート 443 のリスナーには同じポリシーが設定されている必要があります。

3 TCP ポート 5000 のリスナーにはこのポリシーを設定**できません**。

または、ELB をすでに設定していても、PROXY プロトコルを使用するよう設定されていない場合は、TCP ポート 80 の既存リスナーを、HTTP ではなく TCP プロコトルを使用するように変更する必要があります (TCP ポート 443 はすでに TCP プロトコルを使用しているはずです)。

```
$ aws elb delete-load-balancer-listeners --load-balancer-name "$lb" \
   --load-balancer-ports 80
$ aws elb create-load-balancer-listeners --load-balancer-name "$lb" \
   --listeners
'Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=80'
```

プロコトル更新の確認

プロコトルが以下のように更新されていることを確認します。

```
$ aws elb describe-load-balancers --load-balancer-name "$lb" |
   jq '[.LoadBalancerDescriptions[]|.ListenerDescriptions]'
[
  [
    {
      "Listener": {
        "InstancePort": 443,
        "LoadBalancerPort": 443,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    },
```

```
    {
      "Listener": {
        "InstancePort": 5000,
        "LoadBalancerPort": 5000,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    },
    {
      "Listener": {
        "InstancePort": 80,
        "LoadBalancerPort": 80,
        "Protocol": "TCP", (1)
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    }
  ]
]
```

1 TCP ポート 80 のリスナーなど、すべてのリスナーが TCP プロコトルを使用している必要があります。

次にロードバランサーポリシーを作成し、上記の手順 5 に説明されているようにこれを ELB に追加します。

3.5. F5 ルータープラグインの使用

3.5.1. 概要

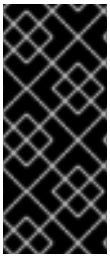
注記

F5 ルータープラグインは OpenShift Container Platform 3.0.2 以降で利用できます。

警告

F5 ルータープラグインは {product_title} バージョン 3.11 で非推奨になります。これは F5 BIG-IP Controller に置き換わります。詳細は、「F5 Container Connector - OpenShift」を参照してください。BIG-IP Controller への移行については、「Replace the F5 Router in OpenShift」を参照してください。

F5 ルーターブラグインはコンテナーイメージとして提供され、デフォルトの HAProxy ルーターのように Pod として実行されます。



重要

F5 と Red Hat 間のサポート体制により、Red Hat は F5 の統合を完全にサポートしています。F5 は **F5 BIG-IP®** 製品のサポートを提供しています。F5 と Red Hat は Red Hat OpenShift との統合を共同でサポートしています。Red Hat はバグフィックスおよび機能拡張を支援する一方で、すべての情報が F5 Networks に通信され、それらが開発サイクルの一部として管理されます。

3.5.2. 前提条件とサポート容易性

F5 ルータープラグインのデプロイ時に、以下の要件を満たしていることを確認してください。

- F5 ホスト IPに以下が設定されている:
  - API アクセスの認証情報
  - プライベートキーによる SSH アクセス
- 高度な Shell アクセスを持つ F5 ユーザー
- HTTP ルートの仮想サーバー:
  - **HTTP プロファイル** は **http** である必要があります。
- HTTP プロファイルルートを持つ仮想サーバー:
  - **HTTP プロファイル** は **http** である必要があります。
  - **SSL プロファイル (クライアント)** は **clientssl** である必要があります。
  - **SSL プロファイル (サーバー)** は **serverssl** である必要があります。
- edge 統合の場合 (非推奨):
  - 稼働中の Ramp ノード
  - Ramp ノードへの稼働中のトンネル
- ネイティブ統合の場合:
  - ポート 4789/UDP のすべてのノードと通信できるホストの内部 IP
  - F5 ホストにインストールされた sdn-services アドオンライセンス

OpenShift Container Platform は以下の **F5 BIG-IP®** バージョンのみをサポートしています。

- 11.x
- 12.x

重要

以下の機能は **F5 BIG-IP®** でサポートされていません。

- ワイルドカードルートと re-encrypt ルートの併用: ルートに証明書とキーを指定する必要があります。証明書、キー、認証局 (CA) を指定しても、CA は使用されません。
- 関連ルートを持たないサービスを含め、すべてのサービスのプールが作成されます。
- アプリケーションのアイドリング
- **redirect** モードの暗号化されていない HTTP トラフィックと TLS edge termination。(**`insecureEdgeTerminationPolicy: Redirect`**)
- シャード化、つまり F5 で複数の **`vservers`** を設定する。
- SSL 暗号 (**`ROUTER_CIPHERS=modern/old`**)
- エンドポイントのヘルスチェックの間隔やチェックタイプのカスタマイズ。
- メトリクスサーバーの使用による F5 メトリクスの提供。
- サービスでされるポートではない各種ターゲットポート (**`PreferPort`**/**`TargetPort`**) の指定。
- ソース IP ホワイトリストのカスタマイズ。つまり特定の IP アドレスのルートのみに対してトラフィックを許可する。
- **`max connect time`** または **`tcp FIN timeout`** などのタイムアウト値のカスタマイズ。
- **F5 BIG-IP®** の HA モード。

3.5.2.1. 仮想サーバーの設定

openshift-F5 統合ルーターを使用する前提条件として、2 つの仮想サーバー (HTTP および HTTPS プロファイルのそれぞれに 1 つの仮想サーバー) を **F5 BIG-IP®** アプライアンスでセットアップする必要があります。

F5 BIG-IP® アプライアンスで仮想サーバーを設定するには、F5 の指示に従います。

仮想サーバーを作成する際に、以下の設定が有効であることを確認します。

- HTTP サーバーの場合は、**`ServicePort`** を **`'http'/80`** に設定します。
- HTTPS サーバーの場合は、**`ServicePort`** を **`'https'/443`** に設定します。
- 基本設定で、両方の仮想サーバーの HTTP プロファイルを **/Common/http** に設定します。
- HTTPS サーバーの場合は、 デフォルトの **client-ssl** プロファイルを作成し、**SSL プロファイル (クライアント)** に対してこれを選択します。
  - デフォルトのクライアント SSL プロファイルを作成するには、F5 の指示、特に「**Configuring the fallback (default) client SSL profile**」セクションに従います。このセクションは、この証明書/キーペアはデフォルトで、ルートまたはサービス名にカスタムの

証明書が指定されていない場合に提供されることについて説明しています。

3.5.3. F5 ルーターのデプロイ

重要

ルート証明書は **scp** コマンドを使用してコピーされるため、F5 ルーターは特権モードで実行される必要があります。

```
$ oc adm policy remove-scc-from-user hostnetwork -z router
$ oc adm policy add-scc-to-user privileged -z router
```

oc adm router コマンドを使用して F5 ルーターをデプロイしますが、**F5 BIG-IP®** ホストの以下のパラメーターを指定する追加のフラグ (または環境変数) を提供します。

フラグ	説明
--type=f5-router	F5 ルーターの起動を指定します (デフォルトの **--type** は **haproxy-router**です)。
--external-host	**F5 BIG-IP®** ホストの管理インターフェースのホスト名または IP アドレスを指定します。
--external-host-username	**F5 BIG-IP®** のユーザー名 (通常は **admin**) を指定します。**F5 BIG-IP** のユーザーアカウントは F5 BIG-IP システムの Advanced Shell (Bash) へのアクセスがなければなりません。
--external-host-password	**F5 BIG-IP®** のパスワードを指定します。
--external-host-http-vserver	HTTP 接続の F5 仮想サーバーの名前を指定します。これは、ルーター Pod の起動前にユーザーによって設定される必要があります。
--external-host-https-vserver	HTTPS 接続の F5 仮想サーバーの名前を指定します。これは、ルーター Pod の起動前にユーザーによって設定される必要があります。
--external-host-private-key	**F5 BIG-IP®** ホストの SSH プライベートキーファイルへのパスを指定します。ルートのキーと証明書ファイルをアップロードし、削除する必要があります。
--external-host-insecure	F5 ルーターが **F5 BIG-IP®** ホストの証明書の厳しい検証を省略することを示す Boolean フラグ。

フラグ	説明
`--external-host-partition-path`	**F5 BIG-IP®** パーティションパス (デフォルトは **/Common**) を指定します。

以下は例を示しています。

```
$ oc adm router \
    --type=f5-router \
    --external-host=10.0.0.2 \
    --external-host-username=admin \
    --external-host-password=mypassword \
    --external-host-http-vserver=ose-vserver \
    --external-host-https-vserver=https-ose-vserver \
    --external-host-private-key=/path/to/key \
    --host-network=false \
    --service-account=router
```

HAProxy ルーターの場合のように、**`oc adm router`** コマンドがサービスとデプロイメント設定オブジェクトを作成するため、F5 ルーター自体が実行されるレプリケーションコントローラーと Pod が作成されます。レプリケーションコントローラーはクラッシュが発生した場合に F5 ルーターを再起動します。F5 ルーターはルート、エンドポイント、ノードを監視し、**F5 BIG-IP®** を適宜設定するため、F5 ルーターを適切に設定された **F5 BIG-IP®** デプロイメントで実行することは高可用性の要件を満たすはずです。

3.5.4. F5 ルーターのパーティションパス

パーティションパスによって、デフォルトの **/Common** パーティションではなく、カスタムの **F5 BIG-IP®** 管理パーティションで OpenShift Container Platform ルーティング設定を保存できます。カスタム管理パーティションを使用して **F5 BIG-IP®** 環境を保護することができます。つまり、**F5 BIG-IP®** システムオブジェクトに保存される OpenShift Container Platform 特有の設定が論理コンテナー内にあり、管理者はその管理パーティションでアクセス制御を定義できます。

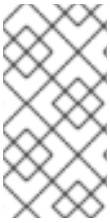
管理パーティションの詳細については、**F5 BIG-IP® ドキュメント**を参照してください。

パーティションパスについて OpenShift Container Platform を設定するには、以下を実行します。

1. オプションで一部のクリーニング手順を実行できます。
    a. F5 が **/Common** および **/Custom** パスに切り替えられるよう設定されていることを確認してください。
    b. **`vxlan5000`** の静的 FDB を削除します。詳細は、F5 BIG-IP® ドキュメントを参照してください。
2. カスタムパーティションの仮想サーバーを設定します。
3. **`--external-host-partition-path`** フラグを使用して F5 ルーターをデプロイし、パーティションパスを指定します。

```
$ oc adm router --external-host-partition-path=/OpenShift/zone1 ...
```

3.5.5. F5 ネイティブ統合のセットアップ



注記

このセクションでは、OpenShift Container Platform への F5 ネイティブ統合のセットアップ方法を確認します。F5 アプライアンスと OpenShift Container Platform 接続の概念および F5 ネイティブ統合のデータフローはルートに関するトピックの「F5 Native Integration」セクションで説明されています。

注記

F5 BIG-IP® アプライアンスのバージョン 12.x 以降のみが、このセクションに記載されているネイティブ統合に対応しています。また、適切に統合を行うために sdn-services アドオンライセンスが必要になります。バージョン 11.x の場合は、**ramp ノード** のセットアップ手順に従ってください。

OpenShift Container Platform バージョン 3.4 時点で、F5 の OpenShift Container Platform とのネイティブな統合により、F5 の Ramp ノードを OpenShift SDN で作成されるオーバーレイネットワークの Pod に到達するように設定する必要がありません。

F5 コントローラー Pod は、Pod に直接正常に接続できるようにするために必要な十分な情報を使って起動する必要があります。

1. OpenShift Container Platform クラスターにゴースト **hostsubnet** を作成します。

```
$ cat > f5-hostsubnet.yaml << EOF
{
    "kind": "HostSubnet",
    "apiVersion": "v1",
    "metadata": {
        "name": "openshift-f5-node",
        "annotations": {
        "pod.network.openshift.io/assign-subnet": "true",
 "pod.network.openshift.io/fixed-vnid-host": "0"   (1)
        }
    },
    "host": "openshift-f5-node",
    "hostIP": "10.3.89.213"   (2)
} EOF
$ oc create -f f5-hostsubnet.yaml
```

(1) F5 をグローバルにします。

(2) F5 アプライアンスの内部 IP です。

2. 作成したゴースト **hostsubnet** に割り当てるサブネットを決定します。

```
$ oc get hostsubnets
NAME                   HOST                   HOST IP
SUBNET
openshift-f5-node      openshift-f5-node      10.3.89.213
10.131.0.0/23
openshift-master-node  openshift-master-node  172.17.0.2
10.129.0.0/23
```

```
openshift-node-1        openshift-node-1        172.17.0.3
10.128.0.0/23
openshift-node-2        openshift-node-2        172.17.0.4
10.130.0.0/23
```

3. 新たに作成した **hostsubnet** の **SUBNET** を確認します。この例では **10.131.0.0/23** です。

4. Pod ネットワーク全体の CIDR を取得します。

```
$ oc get clusternetwork
```

この値は **10.128.0.0/14** のように表示されます。マスク (この例では **14**) に注意してください。

5. ゲートウェイアドレスを作成するには、**hostsubnet** から任意の IP アドレス (例: **10.131.0.5**) を選択します。Pod ネットワークのマスク (**14**) を使用します。ゲートウェイアドレスは **10.131.0.5/14** になります。

6. こちらの指示に従って F5 コントローラー Pod を起動してください。さらに、サービスアカウントに「ノード」のクラスターリソースへのアクセスを許可し、VXLAN のネイティブ統合に 2 つの新しい追加オプションを使用します。

```
$ # Add policy to allow router to access nodes using the sdn-reader
role
$ oc adm policy add-cluster-role-to-user system:sdn-reader
system:serviceaccount:default:router
$ # Launch the router pod with vxlan-gw and F5's internal IP as
extra arguments
$ #--external-host-internal-ip=10.3.89.213
$ #--external-host-vxlan-gw=10.131.0.5/14
$ oc adm router \
    --type=f5-router \
    --external-host=10.3.89.90 \
    --external-host-username=admin \
    --external-host-password=mypassword \
    --external-host-http-vserver=ose-vserver \
    --external-host-https-vserver=https-ose-vserver \
    --external-host-private-key=/path/to/key \
    --service-account=router \
    --host-network=false \
    --external-host-internal-ip=10.3.89.213 \
    --external-host-vxlan-gw=10.131.0.5/14
```

注記

external-host-username は、F5 BIG-IP システムの Advanced Shell (Bash) へのアクセスを持つ **F5 BIG-IP** ユーザーアカウントです。

これで Ramp ノードを設定せずに F5 のセットアップを開始できます。

第4章 RED HAT CLOUDFORMS のデプロイ

4.1. {MGMT-APP} の OPENSHIFT CONTAINER PLATFORM へのデプロイ

4.1.1. はじめに

OpenShift Container Platform インストーラーには、Red Hat CloudForms 4.6 (CloudForms Management Engine 5.9 または CFME) を OpenShift Container Platform にデプロイするための Ansible ロールの **openshift-management** と Playbook が含まれています。

警告

現在の実装には、OpenShift Container Platform 3.6 ドキュメントで説明されているように、Red Hat CloudForms 4.5 のテクノロジープレビューのデプロイメントプロセスとの互換性はありません。

Red Hat CloudForms を OpenShift Container Platform にデプロイする際には、以下の 2 つ点について決定する必要があります。

1. 外部またはコンテナー化された (**Pod 化された** とも言う) PostgreSQL データベースを使用するかどうか。
2. 永続ボリューム (PV) をどのストレージクラスでサポートするか。

最初の点については、Red Hat CloudForms で使用する PostgreSQL データベースが置かれている場所によって Red Hat CloudForms を以下のいずれかの方法でデプロイできます。

デプロイのバリアント	説明
完全コンテナー化	すべてのアプリケーションサービスと PostgreSQL データベースは、OpenShift Container Platform で Pod として実行されます。
外部データベース	アプリケーションは外部でホストされた PostgreSQL データベースサーバーを使用し、その他すべてのサービスは OpenShift Container Platform で Pod として実行されます。

2 つ目の点については、**openshift-management** ロールが、多くのデフォルトのデプロイメントパラメーターの上書き用にカスタマイズオプションを提供します。これには、 PV をサポートするための以下のストレージクラスのオプションが含まれています。

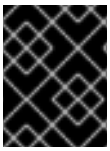
ストレージクラス	説明
NFS (デフォルト)	ローカル、クラスター上

ストレージクラス	説明
NFS (外部)	NFS の他の場所 、ストレージアプライアンスなど
クラウドプロバイダー	クラウドプロバイダー (GCE または AWS) の自動ストレージプロビジョニングを使用
事前設定 (詳細)	ユーザーが事前にすべてを作成済みであることを前提とする

本書では、Red Hat CloudForms を OpenShift Container Platform で実行するための要件、利用可能な設定変数の説明、および OpenShift Container Platform の初回インストール時かクラスターのプロビジョニング後のいずれかでインストーラーを実行する方法などについてのトピックを扱います。

4.2. RED HAT CLOUDFORMS を OPENSHIFT CONTAINER PLATFORM で使用するための要件

デフォルトの要件は以下の表に記載されています。これらはテンプレートパラメーターをカスタマイズして上書きできます。



重要

以下の要件を満たしていないと、アプリケーションのパフォーマンスが低下するか、またはデプロイに失敗する可能性があります。

表4.1 デフォルトの要件

項目	要件	説明	カスタマイズパラメーター
アプリケーションメモリー	≥ 4.0 Gi	アプリケーションのメモリー最小要件	**APPLICATION_MEM_REQ**
アプリケーションストレージ	≥ 5.0 Gi	アプリケーションの PV サイズ最小要件	**APPLICATION_VOLUME_CAPACITY**
PostgreSQL メモリー	≥ 6.0 Gi	データベースのメモリー最小要件	**POSTGRESQL_MEM_REQ**
PostgreSQL ストレージ	≥ 15.0 Gi	データベースの PV サイズ最小要件	**DATABASE_VOLUME_CAPACITY**
クラスターホスト	≥ 3	クラスター内のホスト数	該当なし

以上の要件をまとめると以下のようになります。

- ユーザーにはいくつかのクラスターノードが必要である。

- クラスターノードには多くの利用可能なメモリーが必要である。
- ユーザーは、ローカルまたはクラウドプロバイダーに数 GiB の利用可能なストレージを持っている必要がある。
- PV サイズはテンプレートパラメーターの値を上書きして変更できる。

4.3. ロール変数の設定

4.3.1. 概要

以下のセクションでは、Ansible インベントリーファイル で使用されるロール変数について説明します。ロール変数は、インストーラーを実行する際に Red Hat CloudForms インストールの動作を制御するために使用されます。

4.3.2. 一般的な変数

変数	必須	デフォルト	説明
openshift_management_install_management	No	**false**	ブール値。アプリケーションをインストールするには **true** に設定します。
openshift_management_app_template	必要	**cfme-template**	インストールする Red Hat CloudForms のデプロイメントバリアントです。コンテナー化されたデータベースに **cfme-template** を設定するか、または外部データベースに **cfme-template-ext-db** を設定します。
openshift_management_project	No	**openshift-management**	Red Hat CloudForms インストールの namespace (プロジェクト)。
openshift_management_project_description	No	**CloudForms Management Engine**	namespace (プロジェクト) の説明。
openshift_management_username	No	**admin**	デフォルトの管理ユーザー名。この値を変更してもユーザー名は変更されません。名前をすでに変更しており、統合スクリプト (コンテナープロバイダーを追加するためのスクリプトなど) を実行している場合にのみこの値を変更してください。

変数	必須	デフォルト	説明
openshift_management_password	No	**smartvm**	デフォルトの管理パスワード。この値を変更してもパスワードは変更されません。このパスワードをすでに変更しており、統合スクリプト (コンテナープロバイダーを追加するためのスクリプトなど) を実行している場合にのみこの値を変更してください。

4.3.3. テンプレートパラメーターのカスタマイズ

openshift_management_template_parameters Ansible ロール変数は、アプリケーションまたは PV テンプレートで上書きする必要のあるテンプレートを指定するために使用します。

たとえば、PostgreSQL Pod のメモリー要件を減らしたい場合には、以下を設定します。

```
openshift_management_template_parameters={'POSTGRESQL_MEM_REQ': '1Gi'}
```

Red Hat CloudForms テンプレートが処理される際に、**1Gi** が **POSTGRESQL_MEM_REQ** テンプレートパラメーターの値に使用されます。

すべてのテンプレートパラメーターが (コンテナー化されたデータベースと外部データベースの) **両方** のテンプレートバリアントにある訳ではありません。たとえば、Pod 化されたデータベースのテンプレートには **POSTGRESQL_MEM_REQ** パラメーターがありますが、このパラメーターは外部のデータベーステンプレートにはありません。そこには Pod を必要とするデータベースが存在せず、この情報は必要とされないためです。

したがって、テンプレートパラメーターを上書きする場合には十分注意する必要があります。テンプレートで定義されていないパラメーターを追加するとエラーの原因になります。**管理アプリケーションの作成を確認する**タスクの実行時にエラーを受信した場合は、インストーラーを再度実行する前にアンインストールのスクリプトを実行してください。

4.3.4. データベース変数

4.3.4.1. コンテナー化された (Pod 化された) データベース

cfme-template.yaml ファイルにある **POSTGRES_*** または **DATABASE_*** テンプレートパラメーターは、インベントリーファイルの **openshift_management_template_parameters** ハッシュを使用してカスタマイズすることができます。

4.3.4.2. 外部データベース

cfme-template-ext-db.yaml ファイルにある **POSTGRES_*** または **DATABASE_*** テンプレートパラメーターは、インベントリーファイルの **openshift_management_template_parameters** ハッシュを使用してカスタマイズすることができます。

外部 PostgreSQL データベースは、ユーザーにデータベース接続パラメーターを指定するように要求します。必要な接続キーはインベントリーの **openshift_management_template_parameters** パラメーターに設定する必要があります。必要なキー以下の通りです。

- **DATABASE_USER**
- **DATABASE_PASSWORD**

- **DATABASE_IP**
- **DATABASE_PORT** (ほとんどの PostgreSQL サーバーはポート **5432** で実行されます)
- **DATABASE_NAME**

注記

外部データベースで PostgreSQL 9.5 が実行されていることを確認します。実行されていないと、CloudForms アプリケーションを正常にデプロイできない場合があります。

インベントリーに、以下のような行が含まれているはずです。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template-ext-db (1)
openshift_management_template_parameters={'DATABASE_USER': 'root',
'DATABASE_PASSWORD': 'mypassword', 'DATABASE_IP': '10.10.10.10',
'DATABASE_PORT': '5432', 'DATABASE_NAME': 'cfme'}
```

(1) **openshift_management_app_template** パラメーターを **cfme-template-ext-db** に設定します。

4.3.5. ストレージクラス変数

変数	必須	デフォルト	説明
openshift_management_storage_class	No	**nfs**	使用されるストレージのタイプ。オプションには **nfs**、**nfs_external**、**preconfigured**、**cloudprovider** があります。
openshift_management_storage_nfs_external_hostname	No	**false**	NetApp アプライアンスなどの外部 NFS サーバーを使用している場合、ここにホスト名を設定する必要があります。外部 NFS を使用していない場合は値を **false** のままにしておきます。さらに外部 NFS では、ユーザーがアプリケーション PV とオプションでデータベース PV をサポートする NFS エクスポートを作成する必要があります。

変数	必須	デフォルト	説明
openshift_management_storage_nfs_base_dir	No	**/exports/**	外部 NFS を使用している場合、ベースパスをこのエクスポートの位置に設定できます。ローカルの NFS の場合、ローカルの NFS エクスポートに使用されているデフォルトのパスを変更する必要がある場合にも、この値を変更します。
openshift_management_storage_nfs_local_hostname	No	**false**	**[nfs]** グループがインベントリーにない場合や、ローカルの NFS ホストをクラスターに手動で定義する必要がある場合は、このパラメーターを必要な NFS サーバーのホスト名に設定します。サーバーは OpenShift Container Platform クラスターの一部である必要があります。

4.3.5.1. NFS (デフォルト)

NFS ストレージクラスは、概念実証およびテストデプロイメントに最も適しています。このクラスは、デプロイメント用のデフォルトのステージクラスにもなります。これを選択した場合、追加の設定は不要です。

このストレージクラスは、必要な PV をサポートするように NFS をクラスターホスト (デフォルトではインベントリーファイルの最初のマスター) に設定します。アプリケーションは PV を必要とし、データベース (外部でホストされている可能性がある) は 2 番目の PV が必要となる場合があります。PV サイズの最小要件は、Red Hat CloudForms アプリケーションは 5GiB、PostgreSQL データベースは 15GiB です (NFS 専用で使用する場合は、ボリュームまたはパーティション上の使用可能な最小領域は 20GiB です)。

カスタマイズは、以下のロール変数を使って行われます。

- **openshift_management_storage_nfs_base_dir**
- **openshift_management_storage_nfs_local_hostname**

4.3.5.2. NFS (外部)

外部 NFS は、必要な PV にエクスポートを提供するために事前設定された NFS サーバーを使用します。外部 NFS の場合、ユーザーは **cfme-app** とオプションで **cfme-db** (コンテナー化されたデータベース用) のエクスポートが必要です。

設定は、以下のロール変数を使って提供されます。

- **openshift_management_storage_nfs_external_hostname**
- **openshift_management_storage_nfs_base_dir**

openshift_management_storage_nfs_external_hostnameパラメーターは、外部 NFS サーバーのホスト名または IP に設定する必要があります。

/exports がエクスポートの親ディレクトリーではない場合、ベースディレクトリーを **openshift_management_storage_nfs_base_dir** パラメーターで設定する必要があります。

たとえば、サーバーエクスポートが **/exports/hosted/prod/cfme-app** の場合は、**openshift_management_storage_nfs_base_dir=/exports/hosted/prod** を設定する必要があります。

4.3.5.3. クラウドプロバイダー

ストレージクラスに OpenShift Container Platform のクラウドプロバイダー統合を使用している場合、Red Hat CloudForms は必要な PV をサポートするためにこのクラウドプロバイダーを使用することができます。これを正常に実行するには、**openshift_cloudprovider_kind** 変数 (AWS または GCE 用)と、ユーザーが選択したクラウドプロバイダーに固有のすべての関連パラメーターを設定している必要があります。

アプリケーションがこのストレージクラスを使って作成されている場合、必要な PV は、設定済みのクラウドプロバイダーのストレージ統合を使って自動的にプロビジョニングされます。

このストレージクラスの動作を設定するために使用される追加の変数はありません。

4.3.5.4. 事前設定 (詳細)

preconfigured (事前設定されている) ストレージクラスの場合、ユーザーは各自の操作を正確に把握しており、すべてのストレージ要件が事前に配慮されていることを意味します。この場合、通常は適切なサイズに設定された PV がすでに作成されているので、インストーラーはストレージ設定を変更する必要がありません。

このストレージクラスの動作を設定するために使用される追加の変数はありません。

4.4. インストーラーの実行

4.4.1. OpenShift Container Platform のインストール時またはインストール後の Red Hat CloudForms のデプロイ

Red Hat CloudForms のデプロイを、OpenShift Container Platform の初回インストール時か、またはクラスターのプロビジョニング後のいずれかに実行することを選択できます。

1. **openshift_management_install_management** がインベントリーファイルの **[OSEv3:vars]** セクションの下で **true** に設定されていることを確認します。

```
[OSEv3:vars]
openshift_management_install_management=true
```

2. インベントリーファイルにある Red Hat CloudForms のその他のロール変数を、「ロール変数の設定」で説明されているように設定します。「インベントリーファイルの例」には、これを実行するのに役立つリソースがあります。

3. OpenShift Container Platform がすでにプロビジョニングされているかどうかに応じて、実行する Playbook を選択します。

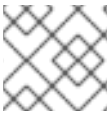
   a. Red Hat CloudForms を、OpenShift Container Platform クラスターのインストールと同時にインストールする必要がある場合には、「インストール Playbook の実行」で説明されているように標準の **config.yml** Playbook を呼び出して、OpenShift Container Platform クラスターと Red Hat CloudForms のインストールを開始します。

   b. Red Hat CloudForms を、すでにプロビジョニングされている OpenShift Container Platform クラスターにインストールする必要がある場合には、Red Hat CloudForms Playbook を直接呼び出してインストールを開始します。

```
# ansible-playbook -v [-i /path/to/inventory] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
management/config.yml
```

4.4.2. インベントリーファイルの例

このセクションでは、インベントリーファイルのスニペットの例について説明し、使い始めに役立つ OpenShift Container Platform での Red Hat CloudForms の各種の設定について説明します。



注記

変数の詳しい説明については、「ロール変数の設定」を参照してください。

4.4.2.1. すべてのデフォルト

以下は、すべてのデフォルト値と選択肢を使用した最も単純な例です。これで、Red Hat CloudForms のインストールが完全にコンテナー化 (Pod 化) されます。すべてのアプリケーションコンポーネントと PostgreSQL データベースが OpenShift Container Platform に Pod として作成されます。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
```

4.4.2.2. 外部 NFS ストレージ

以下は、前述の例と同様に (ただしローカルの NFS サービスをクラスターで使用する代わりに)、既存の外部 NFS サーバー (ストレージアプライアンスなど) を使用しています。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_storage_class=nfs_external 1
openshift_management_storage_nfs_external_hostname=nfs.example.com 2
```

1 **nfs_external** に設定します。

2 NFS サーバーのホスト名に設定します。

外部 NFS ホストが **/exports/hosted/prod** などの異なる親ディレクトリーの下でエクスポートディレクトリーをホストしている場合は、以下の変数を追加します。

```
openshift_management_storage_nfs_base_dir=/exports/hosted/prod
```

4.4.2.3. PV サイズの上書き

以下の例では、永続ボリューム (PV) のサイズを上書きします。PV サイズは **openshift_management_template_parameters** で設定する必要があります。これにより、アプリケーションとデータベースは相互に干渉しあうことなく作成済みの PV で要求を実行できます。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_template_parameters={'APPLICATION_VOLUME_CAPACITY':
'10Gi', 'DATABASE_VOLUME_CAPACITY': '25Gi'}
```

4.4.2.4. メモリー要件の上書き

テストまたは概念実証のインストールでは、アプリケーションとデータベースのメモリー要件を設定している容量内に収めるようにする必要がある場合があります。メモリーの上限を下げると、パフォーマンスが低下するか、またはアプリケーションの初期化に完全に失敗したりする可能性があることに注意してください。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_template_parameters={'APPLICATION_MEM_REQ': '3000Mi',
'POSTGRESQL_MEM_REQ': '1Gi', 'ANSIBLE_MEM_REQ': '512Mi'}
```

この例では、インストーラーに対し、パラメーター **APPLICATION_MEM_REQ** を **3000Mi** に、**POSTGRESQL_MEM_REQ** を **1Gi** に、さらに **ANSIBLE_MEM_REQ** を **512Mi**に設定してアプリケーションテンプレートを処理するように指示します。

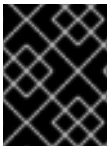
これらのパラメーターは、前述の例 PV サイズの上書きに示されているパラメーターと組み合せることができます。

4.4.2.5. 外部 PostgreSQL データベース

外部データベースを使用するには、**openshift_management_app_template** パラメーターの値を **cfme-template-ext-db** に変更する必要があります。

さらに、データベース接続情報を **openshift_management_template_parameters** 変数を使って入力する必要があります。詳細は、「ロール変数の設定」を参照してください。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template-ext-db
openshift_management_template_parameters={'DATABASE_USER': 'root',
'DATABASE_PASSWORD': 'mypassword', 'DATABASE_IP': '10.10.10.10',
'DATABASE_PORT': '5432', 'DATABASE_NAME': 'cfme'}
```



重要

PostgreSQL 9.5 が実行中であることを確認してください。実行されていないと、アプリケーションを正常にデプロイできない場合があります。

4.5. コンテナープロバイダー統合の有効化

4.5.1. 単一コンテナープロバイダーの追加

「インストーラーの実行」で説明されているように Red Hat CloudForms を OpenShift Container Platform にデプロイした後にコンテナープロバイダーの統合を有効にする方法として 、OpenShift Container Platform をコンテナープロバイダーとして手動で追加する方法と、このロールに含まれている Playbook を試行する方法の 2 つの方法があります。

4.5.1.1. 手動の追加

OpenShift Container Platform クラスターをコンテナープロバイダーとして手動で追加する手順については、以下の Red Hat CloudForms ドキュメントを参照してください。

- Integration with OpenShift Container Platform

4.5.1.2. 自動の追加

コンテナープロバイダーの自動的な統合は、このロールに含まれている Playbook を使用して実行できます。

この Playbook は以下を実行します。

1. 必要な認証シークレットを収集します。
2. Red Hat CloudForms アプリケーションとクラスター API への公開ルートを検索します。
3. REST の呼び出しを実行し、OpenShift Container Platform クラスターをコンテナープロバイダーとして追加します。

コンテナープロバイダーの Playbook を実行するには、以下を実行します。

```
# ansible-playbook -v [-i /path/to/inventory] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
management/add_container_provider.yml
```

4.5.2. 複数のコンテナープロバイダー

このロールには、最新の OpenShift Container Platform クラスターを Red Hat CloudForms デプロイメントに統合するための Playbook に加え、複数のコンテナープラットフォームを任意の Red Hat CloudForms サーバーに コンテナープロバイダーとして追加することを可能にするスクリプトが含まれています。コンテナープラットフォームは、OpenShift Container Platform または OpenShift Origin です。

複数のプロバイダースクリプトを使用する場合、Playbook の実行時に **EXTRA_VARS** パラメーターを CLI に手動で設定することが必要になります。

4.5.2.1. スクリプトの作成

複数のプロバイダースクリプトを作成するには、以下の手動の設定を実行します。

1. **/usr/share/ansible/openshift-ansible/roles/openshift_management/files/examples/container_providers.yml** のサンプルを、**/tmp/cp.yml** など別の場所にコピーします。このファイルは後で変更します。
2. Red Hat CloudForms の名前またはパスワードを変更している場合は、コピーした**container_providers.yml** ファイルの **management_server** キーにある **hostname**、**user**、**password** の各パラメーターを更新します。

3. コンテナープロバイダーとして追加する必要のある各 Container Platform クラスターについて、**container_providers** キーの下にエントリーを入力します。

   a. 以下のパラメーターを設定する必要があります。

      - **auth_key** - **cluster-admin** 権限を持つサービスアカウントのトークンです。
      - **hostname** - クラスター API をポイントしているホスト名です。各コンテナープロバイダーは固有のホスト名を持っている必要があります。
      - **name** - Red Hat CloudForms サーバーのコンテナープロバイダーの概要ページに表示されるクラスター名です。この名前は固有でなければなりません。

   ヒント

   auth_key ベアラートークンをクラスターから取得するには、以下を実行します。

   ```
   $ oc serviceaccounts get-token -n management-infra management-
   admin
   ```

   b. 以下のパラメーターは任意で設定することができます。

      - **port** - Container Platform クラスターが API を **8443** 以外のポートで実行している場合は、このキーを更新します。
      - **endpoint** - SSL 検証を有効にする (**verify_ssl**) か、または検証設定を **ssl-with-validation** に変更することができます。現時点では、カスタムの信頼される CA 証明書のサポートは利用できません。

4.5.2.1.1. 例

例として以下のシナリオを見てみましょう。

- **container_providers.yml** ファイルを **/tmp/cp.yml** にコピーしている。
- 2 つの OpenShift Container Platform クラスターを追加する必要がある。
- Red Hat CloudForms サーバーが **mgmt.example.com** で実行されている。

このシナリオでは、**/tmp/cp.yml** を以下のようにカスタマイズします。

```
container_providers:
  - connection_configurations:
      - authentication: {auth_key: "<token>", authtype: bearer, type:
AuthToken} (1)
        endpoint: {role: default, security_protocol: ssl-without-
validation, verify_ssl: 0}
    hostname: "<provider_hostname1>"
    name: <display_name1>
    port: 8443
    type: "ManageIQ::Providers::Openshift::ContainerManager"
  - connection_configurations:
      - authentication: {auth_key: "<token>", authtype: bearer, type:
AuthToken} (2)
        endpoint: {role: default, security_protocol: ssl-without-
```

```
validation, verify_ssl: 0}
    hostname: "<provider_hostname2>"
    name: <display_name2>
    port: 8443
    type: "ManageIQ::Providers::Openshift::ContainerManager"
management_server:
  hostname: "<hostname>"
  user: <user_name>
  password: <password>
```

1 2 **<token>** をこのクラスターの管理トークンに置き換えます。

4.5.2.2. Playbook の実行

複数プロバイダー統合スクリプトを実行するには、コンテナープロバイダーの設定ファイルへのパスを **EXTRA_VARS** パラメーターとして**ansible-playbook** コマンドに指定する必要があります。**container_providers_config** を設定ファイルパスに設定するには、**-e** (または **--extra-vars**) パラメーターを使用します。

```
# ansible-playbook -v [-i /path/to/inventory] \
    -e container_providers_config=/tmp/cp.yml \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
management/add_many_container_providers.yml
```

Playbook が完成すると、Red Hat CloudForms サービスに 2 つの新しいコンテナープロバイダーが見つかるはずです。**コンピュート** → **コンテナー** → **プロバイダー** の順にページを進んで概要を確認してください。

4.5.3. プロバイダーの更新

単一または複数のコンテナープロバイダーを追加したら、新規プロバイダーを Red Hat CloudForms で更新し、コンテナープロバイダーと管理されているコンテナーに関する最新データをすべて取得する必要があります。そのためには、Red Hat CloudForms Web コンソールの各プロバイダーに進み、それぞれについて更新ボタンをクリックします。

手順については、以下の Red Hat CloudForms ドキュメントを参照してください。

- Managing Providers

4.6. RED HAT CLOUDFORMS のアンインストール

4.6.1. アンインストール Playbook の実行

デプロイした Red Hat CloudForms インストールを OpenShift Container Platform からアンインストールして削除するには、以下の Playbook を実行します。

```
# ansible-playbook -v [-i /path/to/inventory] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
management/uninstall.yml
```

重要

NFS エクスポートの定義とNFS エクスポートに格納されているデータは自動的に削除されません。新規デプロイメントの初期化を試行する前に、古いアプリケーションまたはデータベースデプロイメントのデータを手動で消去することが推奨されます。

4.6.2. トラブルシューティング

古い PostgreSQL データの消去に失敗すると連鎖的なエラーが発生し、**postgresql** Pod が **crashloopbackoff** の状態になる可能性があります。この状態になると **cfme** Pod の起動が阻止されます。**crashloopbackoff** は、以前のデプロイ時に作成されたデータベース NFS エクスポートの不正なファイル権限に起因します。

次に進むには、PostgreSQL エクスポートからすべてのデータを削除し、Pod (デプロイヤー Pod ではない) を削除します。以下の Pod がある場合の例を示します。

```
$ oc get pods
NAME                 READY     STATUS             RESTARTS   AGE
httpd-1-cx7fk        1/1       Running            1          21h
cfme-0               0/1       Running            1          21h
memcached-1-vkc7p    1/1       Running            1          21h
postgresql-1-deploy  1/1       Running            1          21h
postgresql-1-6w2t4   0/1       CrashLoopBackOff   1          21h
```

この場合、以下を実行します。

1. データベース NFS エクスポートのデータを消去します。
2. 以下を実行します。

```
$ oc delete postgresql-1-6w2t4
```

PostgreSQL デプロイヤー Pod は、削除した Pod を置き換えるために新規の **postgresql** Pod の拡張を試みます。 **postgresql** Pod が実行された後に、**cfme** Pod は阻止する操作を停止し、アプリケーションの初期化を開始します。

第5章 マスターとノードの設定

openshift start コマンドおよびそのサブコマンド (マスターサーバーを起動するための **master** および node server を起動するための **node**) が受け取る引数のセット数は限定的ですが、これらは開発または実験環境でサーバーを起動するのには十分です。

ただしこれらの引数は、実稼働環境に必要な設定およびセキュリティーオプションのセットを詳細に記述し、管理するには不十分です。これらのオプションを指定するには、**/etc/origin/master/master-config.yaml** の マスターホストファイル、および ノード設定マップでこれらのオプションを指定する必要があります。

上記のファイルは、デフォルトプラグインの上書き、etcd への接続、サービスアカウントの自動作成、イメージ名の作成、プロジェクト要求のカスタマイズ、ボリュームプラグインの設定などの各種オプションを定義します。

このトピックでは、OpenShift Container Platform のマスターとノードのホストのカスタマイズに使用できるオプションについて説明し、インストール後に設定を変更する方法を紹介します。

これらのファイルはデフォルト値なしで完全に指定されます。したがって、空の値は、ユーザーがそのパラメーターを空の値で起動しようとしていることを意味します。これによりユーザーの設定を正確に推測することを簡単になりますが、指定する必要のあるすべてのオプションを思い出す必要があるという点では容易な作業にはなりません。これをより容易にするには、設定ファイルを **--write-config** オプションを使って作成し、次に **--config** オプションを指定して使用することができます。

5.1. インストールの依存関係

実稼働環境は、標準のクラスターインストールプロセスを使用してインストールする必要があります。実稼働環境では、高可用性 (HA) を維持するために複数のマスターを使用することが適しています。3 つのマスターで構成されるクラスターアーキテクチャーが推奨され、HAproxy の使用が推奨されるソリューションになります。

注意

etcd が **マスターホスト**にインストールされている場合は、etcd は権限を持つマスターを判別できないために、クラスターを 3 つ以上のマスターを使用するように設定する必要があります。2 つのマスターのみを正常に実行できるのは、etcd をマスター以外のホストにインストールしている場合です。

5.2. マスターとノードの設定

マスターとノードの設定ファイルの設定に使用する方法は、OpenShift Container Platform クラスターのインストールに使用した方法と同じでなければなりません。標準のクラスターインストールプロセスに従う場合は、Ansible インベントリーファイルで設定の変更を加えます。

5.3. ANSIBLE を使用した設定の変更

このセクションは、Ansible に精通していることを前提に説明を行います。

Ansible に公開されているのは、利用可能なホスト設定オプションの一部のみです。OpenShift Container Platform のインストール後、Ansible は インベントリーファイルを置き換えられた値で作成します。このインベントリーファイルを変更し、Ansible インストーラー Playbook を再実行することで、OpenShift Container Platform クラスターをカスタマイズできます。

OpenShift Container Platform は Ansible をクラスターインストールで使用することに対応していますが、Ansible Playbook とインベントリーファイルを使うことで、Puppet、Chef、Salt などの他の管理ツールを使用することもできます。

ユースケース: HTPasswd 認証を使用するようにクラスターを設定する

注記

- このユースケースは、Playbook で参照されているすべてのノードに SSH キーがセットアップされていることを前提としています。
- **htpasswd** ユーティリティーは **httpd-tools** パッケージにあります。

```
# yum install httpd-tools
```

Ansible インベントリーを変更し、設定の変更を行うには、以下を実行します。

1. **./hosts** インベントリーファイルを開きます。
2. 新規の変数をファイルの **[OSEv3:vars]** セクションに追加します。

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
openshift_master_htpasswd_users={'<name>': '<hashed-password>',
'<name>': '<hashed-password>'}
# or
#openshift_master_htpasswd_file=<path/to/local/pre-
generated/htpasswdfile>
```

HTPasswd 認証では、指定されたユーザーとパスワードを作成する **openshift_master_htpasswd_users** 変数か、またはすでに作成済みのユーザーとパスワードを持つ事前生成されたフラットファイル (**htpasswd** ファイル) を指定する **openshift_master_htpasswd_file** 変数のいずれかを使用できます。

OpenShift Container Platform は、HTPasswd 認証を設定するためにハッシュ化されたパスワードを必要とします。以下のセクションに示されるように **htpasswd** コマンドを使用してユーザー用のハッシュ化されたパスワードを生成するか、またはユーザーおよび関連付けられたハッシュ化されたパスワードを持つフラットファイルを作成することができます。

以下の例では、認証方法をデフォルトの **deny all** 設定から **htpasswd** に変更し、指定されたファイルを使って **jsmith** および **bloblaw** ユーザーのユーザー ID とパスワードを生成します。

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
openshift_master_htpasswd_users={'jsmith':
'$apr1$wIwXkFLI$bAygtKGmPOqaJftB', 'bloblaw':
'7IRJ$2ODmeLoxf4I6sUEKfiA$2aDJqLJe'}
```

```
# or
#openshift_master_htpasswd_file=<path/to/local/pre-
generated/htpasswdfile>
```

3. 変更を有効にするために、Ansible Playbook を再実行します。

```
$ ansible-playbook -b -i ./hosts ~/src/openshift-
ansible/playbooks/deploy_cluster.yml
```

Playbook が設定を更新し、OpenShift Container Platform マスターサービスを再起動して変更を適用します。

これで、Ansible を使用したマスターとノードの設定ファイルの変更が完了しました。ここまでは単純なユースケースですが、次は、どのマスターとノードの設定オプションが Ansibleに公開されているかを確認し、各自の Ansible インベントリーをカスタマイズします。

5.3.1. `htpasswd` コマンドの使用

OpenShift Container Platform クラスターを HTPasswd 認証を使用するように設定するには、ハッシュ化されたパスワードを持つ 1 名以上のユーザーをインベントリーファイルに追加する必要があります。

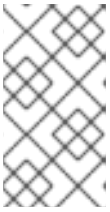
以下を実行することができます。

- ユーザー名とパスワードを生成して **./hosts** インベントリーファイルに直接追加する。
- フラットファイルを作成して認証情報を **./hosts** インベントリーファイルに渡す。

ユーザーおよびハッシュ化されたパスワードを作成するには、以下を実行します。

1. 以下のコマンドを実行して指定されたユーザーを追加します。

```
$ htpasswd -n <user_name>
```



注記

-b オプションを追加すると、パスワードをコマンドラインに指定できます。

```
$ htpasswd -nb <user_name> <password>
```

2. ユーザーのクリアテキストのパスワードを入力し、確定します。
以下は例を示しています。

```
$ htpasswd -n myuser
New password:
Re-type new password:
myuser:$apr1$vdW.cI3j$WSKIOzUPs6Q
```

このコマンドはパスワードのハッシュ化バージョンを生成します。

これで、HTPasswd 認証を設定する際にハッシュ化パスワードを使用できます。ハッシュ化パスワードは、**:** の後に続く文字列です。上記の例では、次を入力します。

```
openshift_master_htpasswd_users={'myuser':
'$apr1$wIwXkFLI$bAygtISk2eKGmqaJftB'}
```

ユーザー名とハッシュ化パスワードを持つフラットファイルを作成するには、以下を実行します。

1. 以下のコマンドを実行します。

```
$ htpasswd -c </path/to/users.htpasswd> <user_name>
```

注記

-b オプションを追加すると、パスワードをコマンドラインに指定できます。

```
$ htpasswd -c -b <user_name> <password>
```

2. ユーザーのクリアテキストのパスワードを入力し、確定します。
以下は例を示しています。

```
htpasswd -c users.htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```

このコマンドは、ユーザー名とユーザーパスワードのハッシュ化されたバージョンを含むファイルを生成します。

これで、HTPasswd 認証を設定する際にこのパスワードファイルを使用できます。

注記

htpasswd コマンドについての詳細は、「HTPasswd Identity Provider」を参照してください。

5.4. 手動による設定変更

ユースケース: HTPasswd 認証を使用するようにクラスターを設定する

設定ファイルを手動で変更するには、以下を実行します。

1. 変更する必要のある設定ファイルを開きます。ここでは **/etc/origin/master/master-config.yaml** ファイルです。

2. 以下の新規変数をファイルの **identityProviders** スタンザに追加します。

```
oauthConfig:
  ...
  identityProviders:
  - name: my_htpasswd_provider
    challenge: true
    login: true
    mappingMethod: claim
    provider:
```

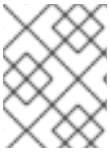
```
      apiVersion: v1
      kind: HTPasswdPasswordIdentityProvider
      file: /path/to/users.htpasswd
```

3. 変更を保存してファイルを閉じます。

4. 変更を有効にするために、マスターを再起動します。

```
$ master-restart api
$ master-restart controllers
```

これでマスターとノードの設定ファイルが手動で変更されました。ここまでは単純なユースケースです。次は、すべてのマスターとノードの設定オプションを確認し、変更を加えることでクラスターをさらにカスタマイズします。



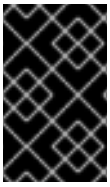
注記

クラスターのノードを変更するには、ノード設定マップを必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

5.5. マスター設定ファイル

このセクションでは、**master-config.yaml** ファイルに記述されているパラメーターについて説明します。

新規のマスター設定ファイルを作成して、OpenShift Container Platform のインストール済みバージョンに有効なオプションを確認できます。



重要

master-config.yaml ファイルを変更する際には常にマスターを再起動して変更を有効にしてください。「OpenShift Container Platform サービスの再起動」を参照してください。

5.5.1. 受付制御の設定

表5.1 受付制御設定パラメーター

パラメーター名	説明
AdmissionConfig	受付制御プラグイン設定が含まれています。OpenShift Container Platform には、API オブジェクトが作成または変更されるたびにトリガーされる、受付制御プラグインの設定可能な一覧が含まれます。このオプションを使用して、一部のプラグインの無効化、他の設定の追加、順序の変更や設定の指定など、デフォルトのプラグイン一覧を上書きできます。プラグインの一覧とその設定はどちらも Ansible で制御することができます。

パラメーター名	説明

APIServerArguments	API サーバーのコマンドライン引数に一致する Kube API サーバーに直接渡されるキーと値のペアです。これらは移行されませんが、存在しない値が参照されてもサーバーは起動しません。これらの値は、**KubernetesMasterConfig** の他の設定を上書きする場合があり、これにより無効な設定が生じる可能性があります。**APIServerArguments** を **event-ttl** という値で使用し、イベントを etcd に保存します。デフォルトは **2h** ですが、メモリーの増加を防ぐためにより小さい値に設定することができます。
	
	`apiServerArguments:`
	`  event-ttl:`
	`  - "15m"`

ControllerArguments	Kube コントローラーマネージャーのコマンドライン引数に一致する、コントローラーマネージャーに直接渡されるキーと値のペアです。これらは移行されませんが、存在しない値が参照されてもサーバーは起動しません。これらの値は、**KubernetesMasterConfig** の他の設定を上書きする場合があり、これにより無効な設定が生じる可能性があります。
DefaultAdmissionConfig	各種の受付プラグインの有効化または無効化に使用されます。このタイプが **pluginConfig** の下に **configuration** オブジェクトとして存在し、受付プラグインがこれをサポートしている場合、**デフォルトでオフ**にされている受付プラグインが有効になります。
PluginConfig	設定ファイルを受付制御プラグインごとに指定することができます。
PluginOrderOverride	マスターにインストールされる受付制御プラグイン名の一覧です。順序には意味があります。空の場合は、プラグインのデフォルトの一覧が使用されます。
SchedulerArguments	スケジューラーのコマンドライン引数に一致する、Kube スケジューラーに直接渡されるキーと値のペアです。これらは移行されませんが、存在しない値が参照されてもサーバーは起動しません。これらの値は、**KubernetesMasterConfig** の他の設定を上書きする場合があり、これにより無効な設定が生じる可能性があります。

5.5.2. アセットの設定

表5.2 アセットの設定パラメーター

パラメーター名	説明

パラメーター名	説明

AssetConfig	これが存在する場合には、アセットサーバーは事前に定義されたパラメーターに基づいて起動します。以下は例になります。
	
	`assetConfig:`
	`  logoutURL: ""`
	`  masterPublicURL:`
	`https://master.ose32.example.com:8443`
	`  publicURL:`
	`https://master.ose32.example.com:8443/console/`
	`  servingInfo:`
	`  bindAddress: 0.0.0.0:8443`
	`  bindNetwork: tcp4`
	`  certFile: master.server.crt`
	`  clientCA: ""`
	`  keyFile: master.server.key`
	`  maxRequestsInFlight: 0`
	`  requestTimeoutSeconds: 0`

corsAllowedOrigins	異なるホスト名を使用して Web アプリケーションから API サーバーにアクセスするには、設定フィールドに **corsAllowedOrigins** を指定するか、または **--cors-allowed-origins** オプションを **openshift start** に指定してそのホスト名をホワイトリスト化する必要があります。その値の固定 (pinning) やエスケープは実行されません。使用例については、「Web Console」を参照してください。
DisabledFeatures	起動することのできない機能の一覧です。**null** に設定してください。この機能を手動で無効にする必要性はほとんどなく、この操作を実行することも推奨されません。
Extensions	サブコンテクストの下位のアセットサーバーファイルシステムから提供されるファイルです。
ExtensionDevelopment	**true** に設定されている場合、起動時だけでなく要求が出されるたびに拡張機能スクリプトとスタイルシートをリロードするようアセットサーバーに指示します。変更のたびにサーバーを再起動しなくても、拡張機能を展開することができます。
ExtensionProperties	コンソールのグローバル変数 **OPENSHIFT_EXTENSION_PROPERTIES** の下に挿入されるキー - (文字列) 値 - (文字列) のペアです。
ExtensionScripts	Web コンソールが読み込む際にスクリプトとして読み込まれるアセットサーバーファイル上のファイルパスです。
ExtensionStylesheets	Web コンソールが読み込む際にスタイルシートとして読み込まれるアセットサーバーファイル上のファイルパスです。
LoggingPublicURL	ロギング用のパブリックエンドポイント (オプション) です。

パラメーター名	説明
LogoutURL	Web コンソールからログアウトした後に Web ブラウザーをリダイレクトするオプションの絶対 URL です。指定されていない場合は、ビルトインされたログアウトページが表示されます。
MasterPublicURL	Web コンソールが OpenShift Container Platform サーバーにアクセスする方法について示します。
MetricsPublicURL	メトリクス用のパブリックエンドポイント (オプション) です。
PublicURL	アセットサーバーの URL です。

5.5.3. 認証と承認の設定

表5.3 認証および承認パラメーター

パラメーター名	説明
authConfig	認証および承認設定オプションを保持します。
AuthenticationCacheSize	キャッシュされる認証結果の数を示します。0 の場合は、デフォルトのキャッシュサイズが使用されます。
AuthorizationCacheTTL	承認結果がキャッシュされる期間を示します。有効な時間文字列 (「5m」など) を取り、空の場合はデフォルトのタイムアウトが取得されます。ゼロ (「0 m」など) の場合、キャッシシングは無効になります。

5.5.4. コントローラーの設定

表5.4 コントローラー設定パラメーター

パラメーター名	説明
Controllers	起動するコントローラーの一覧です。**none** に設定されている場合、コントローラーは自動的に起動されません。デフォルト値は * であり、これによりすべてのコントローラーが起動します。* を使用している場合は、コントローラーの名前の先頭に **-** を追加することでそのコントローラーを除外できます。この時点で他の値は認識されません。
ControllerLeaseTTL	コントローラーの選択を有効にし、マスターに対してコントローラーが起動する前にリースを取得するように指示して、この値で定義された秒数内にリースを更新します。負の値以外の値を設定することで **pauseControllers=true** が強制的に実行されます。デフォルトの値はオフ (0 または省略) であり、コントローラーの選択は -1 で無効にできます。

パラメーター名	説明
PauseControllers	マスターに対してコントローラーを自動的に開始せず、起動前にサーバーへの通知が受信するまで待機するように指示します。

5.5.5. etcd の設定

表5.5 etcd 設定パラメーター

パラメーター名	説明
Address	etcd へのクライアント接続用に公開される host:port です。

etcdClientInfo	etcd に接続する方法についての情報が含まれます。etcd を組み込みまたは組み込み以外の方法で実行するかどうかを指定し、ホストを指定します。残りの設定は Ansible インベントリーで処理されます。以下は例になります。
	
	`etcdClientInfo:`
	`  ca: ca.crt`
	`  certFile: master.etcd-client.crt`
	`  keyFile: master.etcd-client.key`
	`  urls:`
	`  - https://m1.aos.example.com:4001`

etcdConfig	これがある場合、etcd は定義されたパラメーターに基づいて起動します。以下は例になります。
	
	`etcdConfig:`
	`  address: master.ose32.example.com:4001`
	`  peerAddress: master.ose32.example.com:7001`
	`  peerServingInfo:`
	`  bindAddress: 0.0.0.0:7001`
	`  certFile: etcd.server.crt`
	`  clientCA: ca.crt`
	`  keyFile: etcd.server.key`
	`  servingInfo:`
	`  bindAddress: 0.0.0.0:4001`
	`  certFile: etcd.server.crt`
	`  clientCA: ca.crt`
	`  keyFile: etcd.server.key`
	`  storageDirectory:`
	`/var/lib/origin/openshift.local.etcd`

etcdStorageConfig	API リソースを etcd に格納する方法についての情報が含まれます。これらの値は、etcd がクラスターのバッキングストアである場合にのみ関連する値になります。

パラメーター名	説明
KubernetesStoragePrefix	Kubernetes のリソースがその下位に置かれる etcd 内のパスです。この値が変更されると **etcd** 内の既存のオブジェクトは検索できなくなります。デフォルトの値は **kubernetes.io** です。
KubernetesStorageVersion	**etcd** 内の Kubernetes リソースがシリアライズされる API バージョン。etcd から読み取りを行うクラスター内のすべてのクライアントが新規バージョンの読み取りを可能にするコードを取得するまでこの値を変更**することができません**。
OpenShiftStoragePrefix	OpenShift Container Platform リソースがその下位に置かれる etcd 内のパスです。この値が変更されると、etcd 内の既存のオブジェクトは検索できなくなります。デフォルトの値は **openshift.io** です。
OpenShiftStorageVersion	**etcd** 内の OS リソースがシリアライズされる API バージョンです。**etcd** から読み取りを行うクラスター内のすべてのクライアントが新規バージョンの読み取りを可能にするコードを取得するまで、この値を変更**することができません**。
PeerAddress	**etcd** へのピア接続用に公開される host:port です。
PeerServingInfo	**etcd** ピアの提供を開始する方法を記述します。

ServingInfo	提供を開始する方法を記述します。以下は例になります。
	`servingInfo:`
	`  bindAddress: 0.0.0.0:8443`
	`  bindNetwork: tcp4`
	`  certFile: master.server.crt`
	`  clientCA: ca.crt`
	`  keyFile: master.server.key`
	`  maxRequestsInFlight: 500`
	`  requestTimeoutSeconds: 3600`

StorageDir	**etcd** ストレージディレクトリーへのパスです。

5.5.6. 付与の設定

表5.6 付与の設定パラメーター

パラメーター名	説明
GrantConfig	付与を処理する方法を記述します。
GrantHandlerAuto	クライアントの承認付与の要求を自動承認します。
GrantHandlerDeny	クライアントの承認付与の要求を自動的に拒否します。

パラメーター名	説明
GrantHandlerPrompt	ユーザーに対し、クライアントの新規の承認付与要求を承認することを求めるプロンプトを出します。
Method	OAuth クライアントが付与を要求したときに使用するデフォルトのストラテジーを決定します。この方法は特定の OAuth クライアントが独自のストラテジーを提供しない場合にのみ使用します。付与を処理するための有効な方法は以下の通りです。 • auto: 付与要求を常に承認します。信頼されるクライアントの場合に役立ちます。 • prompt: エンドユーザーに対し、付与要求の承認を求めるプロンプトを出します。サードパーティーのクライアントの場合に役立ちます。 • deny: 付与要求を常に拒否します。ブラックリスト化されたクライアントの場合に役立ちます。

5.5.7. イメージ設定

表5.7 イメージ設定パラメーター

パラメーター名	説明
Format	システムコンポーネント用に作成される名前のフォーマットです。
Latest	最新のタグをレジストリーからプルするかどうかを決定します。

5.5.8. イメージポリシーの設定

表5.8 イメージポリシー設定パラメーター

パラメーター名	説明
DisableScheduledImport	スケジュールされたイメージのバックグラウンドインポートの無効化を許可します。
MaxImagesBulkImportedPerRepository	ユーザーが Docker リポジトリーの一括インポートを行う際に、インポートされるイメージの数を制御します。デフォルトの値は 5 に設定され、ユーザーが誤って大量のイメージをインポートすることを防ぎます。**-1** に設定すると無制限になります
MaxScheduledImageImportsPerMinute	スケジュールされたイメージストリームがバックグラウンドでインポートされる 1 分あたりの最大数です。デフォルト値は 60 です。

パラメーター名	説明
ScheduledImageImportMinimumIntervalSeconds	バックグラウンドのインポートがスケジュールされているイメージストリームが、アップストリームのリポジトリーと照合される際の最小間隔 (秒単位) です。デフォルト値は 15 秒です。
AllowedRegistriesForImport	標準ユーザーがイメージのインポートに使用する Docker レジストリーを制限します。この一覧を、有効な Docker イメージを含むものとユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。Images または ImageStreamMappings を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのは管理者またはシステム統合管理者のみです。
InternalRegistryHostname	デフォルトの内部イメージレジストリーのホスト名を設定します。値は **hostname[:port]** 形式である必要があります。後方互換性を考慮して、ユーザーは引き続き **OPENSHIFT_DEFAULT_REGISTRY** 環境変数を使用できますが、この設定はこの環境変数を上書きします。このパラメーターが設定されると、内部レジストリーにはそのホスト名も設定される必要があります。詳細は、「レジストリーのホスト名の設定」を参照してください。
ExternalRegistryHostname	ExternalRegistryHostname は、デフォルトの外部イメージレジストリーのホスト名を設定します。この外部ホスト名は、イメージレジストリーが外部に公開される場合にのみ設定されます。値は ImageStreams の **publicDockerImageRepository** フィールドで使用されます。値は **hostname[:port]** 形式でなければなりません。

5.5.9. Kubernetes のマスター設定

表5.9 Kubernetes のマスター設定パラメーター

パラメーター名	説明
APILevels	起動時に有効にする必要のある API レベルの一覧です (v1 など)。
DisabledAPIGroupVersions	無効にする必要のあるバージョン (または *****) のグループのマップです。
KubeletClientInfo	Kubelets に接続する方法についての情報が含まれます。
KubernetesMasterConfig	kubelet の KubernetesMasterConfig への接続方法についての情報が含まれます。これがある場合、Kubernetes のマスターをこのプロセスで起動します。
MasterCount	実行されていることが予想されるマスターの数です。デフォルトの値は 1 であり、正の整数に設定できますが、-1 に設定されている場合はそれがクラスターの一部であることを示します。

パラメーター名	説明
MasterIP	Kubernetes リソースのパブリック IP アドレスです。空の場合、**net.InterfaceAddrs** の最初の結果が使用されます。
MasterKubeConfig	このノードをマスターに接続する方法を記述した **.kubeconfig** ファイルのファイル名です。
ServicesNodePortRange	サービスのパブリックポートをホストに割り当てる際に使用される範囲です。デフォルトは 30000-32767 です。
ServicesSubnet	サービス IP の割り当てに使用されるサブネットです。
StaticNodeNames	静的に認識されるノードの一覧です。

5.5.10. ネットワーク設定

IPv4 アドレス領域はノード上のすべてのユーザーが共有するため、CIDR を以下のパラメーターで慎重に選択してください。OpenShift Container Platform はそれ自体に使用する CIDR を IPv4 アドレス領域から予約し、外部ユーザーとクラスターが共有するアドレス用の CIDR を IPv4 アドレス領域から予約します。

表5.10 ネットワーク設定パラメーター

パラメーター名	説明
ClusterNetworkCIDR	グローバルなオーバーレイネットワークの L3 領域を指定する CIDR 文字列です。クラスターネットワークの内部使用のために予約されています。
externalIPNetworkCIDRs	サービスの外部 IP フィールドで許可される値を制御します。空の場合、**externalIP** は設定できません。これには CIDR の一覧を含めることができ、この一覧はアクセスについてチェックされます。CIDR にプレフィックス **!** が付いている場合、その CIDR の IP は拒否されます。最初に拒否が適用され、その後に IP が許可された CIDR のいずれかに対してチェックされます。セキュリティー上の理由から、この範囲はユーザーのノード、Pod 、またはサービス CIDR と重複させることはできません。
HostSubnetLength	各ホストのサブネットに割り当てられるビット数です。たとえば 8 の場合は、ホスト上の /24 ネットワークを意味します。
ingressIPNetworkCIDR	ベアメタル上のタイプ **LoadBalancer** のサービス用に ingress IP を割り当てる範囲を制御します。そこから割り当てられる単一の CIDR を含めることができます。デフォルトは **172.46.0.0/16** に設定されています。セキュリティー上の理由から、この範囲は外部 IP 、ノード、Pod、またはサービス用に予約されている CIDR と重複しないようにする必要があります。

パラメーター名	説明
HostSubnetLength	各ホストのサブネットに割り当てられるビット数です。たとえば 8 の場合は、ホスト上の /24 ネットワークを意味します。

NetworkConfig	compiled-in-network プラグインに渡されます。ここでのオプションの多くは Ansible インベントリーで制御されます。
	• **NetworkPluginName** (文字列)
	• **ClusterNetworkCIDR** (文字列)
	• **HostSubnetLength** (署名なし整数)
	• **ServiceNetworkCIDR** (文字列)
	• **ExternalIPNetworkCIDRs** (文字列の配列): サービスの外部 IP フィールドで許可される値を制御します。空の場合、外部 IP を設定できません。CIDR の一覧を含むことができ、そのアクセスがチェックされます。CIDR にプレフィックス **!** が付いている場合、その CIDR のIP は拒否されます。最初に拒否が適用され、次に IP が許可された CIDR のいずれかに対してチェックされます。セキュリティー上の理由から、この範囲はユーザーのノード、Pod、またはサービス CIDR と重複しないようにする必要があります。
	以下は例になります。
	`networkConfig:`
	`  clusterNetworks`
	`  - cidr: 10.3.0.0/16`
	`    hostSubnetLength: 8`
	`  networkPluginName: example/openshift-ovs-subnet`
	`# serviceNetworkCIDR must match kubernetesMasterConfig.servicesSubnet`
	`  serviceNetworkCIDR: 179.29.0.0/16`

NetworkPluginName	使用されるネットワークプラグインの名前です。
ServiceNetwork	サービスネットワークを指定する CIDR 文字列です。

5.5.11. OAuth 認証設定

表5.11 OAuth 設定パラメーター

パラメーター名	説明
AlwaysShowProviderSelection	単一プロバイダーしかない場合でも、プロバイダーの選択ページを強制的にレンダリングします。
AssetPublicURL	外部アクセス用の有効なクライアントのリダイレクト URL の作成に使用されます。

パラメーター名	説明
Error	認証または付与フローでエラーページのレンダリングに使用される Go テンプレートを含むファイルへのパスです。指定しない場合、デフォルトのエラーページが使用されます。
IdentityProviders	ユーザーが自身を確認する方法の順序付きの一覧です。
Login	ログインページのレンダリングに使用される Go テンプレートを含むファイルへのパスです。指定しない場合、デフォルトのログインページが使用されます。
MasterCA	TLS 接続が **MasterURL** に戻っていることを確認するための CA です。
MasterPublicURL	外部アクセス用の有効なクライアントのリダイレクト URL の作成に使用されます。
MasterURL	アクセストークンの承認コードを交換するためのサーバー間の呼び出しに使用されます。

OAuthConfig	これがある場合、/oauth エンドポイントは定義されたパラメーターに基づいて開始します。以下は例になります。
	
	`oauthConfig:`
	`  assetPublicURL:`
	`https://master.ose32.example.com:8443/console/`
	`  grantConfig:`
	`  method: auto`
	`  identityProviders:`
	`  - challenge: true`
	`  login: true`
	`  mappingMethod: claim`
	`  name: htpasswd_all`
	`  provider:`
	`  apiVersion: v1`
	`  kind: HTPasswdPasswordIdentityProvider`
	`  file: /etc/origin/openshift-passwd`
	`  masterCA: ca.crt`
	`  masterPublicURL:`
	`https://master.ose32.example.com:8443`
	`  masterURL:`
	`https://master.ose32.example.com:8443`
	`  sessionConfig:`
	`  sessionMaxAgeSeconds: 3600`
	`  sessionName: ssn`
	`  sessionSecretsFile:`
	`/etc/origin/master/session-secrets.yaml`
	`  tokenConfig:`
	`  accessTokenMaxAgeSeconds: 86400`
	`  authorizeTokenMaxAgeSeconds: 500`

パラメーター名	説明
OAuthTemplates	ログインページなどページのカスタマイズを許可します。
ProviderSelection	プロバイダーの選択ページのレンダリングに使用される Go テンプレートを含むファイルへのパスです。指定されていない場合、デフォルトのプロバイダー選択ページが使用されます。
SessionConfig	セッションの設定に関する情報を保持します。
Templates	ログインページなどのページのカスタマイズを許可します。
TokenConfig	承認とアクセストークンのオプションが含まれます。

5.5.12. プロジェクトの設定

表5.12 プロジェクト設定パラメーター

パラメーター名	説明
DefaultNodeSelector	デフォルトのプロジェクトノードのラベルセレクターを保持します。

パラメーター名	説明
ProjectConfig	プロジェクト作成とデフォルトに関する情報を保持します。 • **DefaultNodeSelector** (文字列): デフォルトのプロジェクトノードのラベルセレクターを保持します。 • **ProjectRequestMessage** (文字列): この文字列は、ユーザーが projectrequest API エンドポイントからプロジェクトを要求できない場合に提示されます。 • **ProjectRequestTemplate** (文字列): projectrequest への応答としてプロジェクトを作成する際に使用されるテンプレートです。フォーマット **<namespace>/<template>** が使用されます。これはオプションであり、指定されていない場合はデフォルトのテンプレートが使用されます。 • **SecurityAllocator**: UID と MCS ラベルのプロジェクトに対する自動割り当てを制御します。空の場合、割り当ては無効にされます。 ◦ **mcsAllocatorRange** (文字列): namespace に割り当てられる MCS カテゴリーの範囲を定義します。フォーマットは **<prefix>/<numberOfLabels>[, <maxCategory>]** です。デフォルトは **s0/2** であり、c0 から c1023 に割り当てられます。つまり、これは合計 535000 のラベルが利用可能であることを意味します。この値を起動後に変更すると、新規プロジェクトは、すでに他のプロジェクトに割り当てられているラベルを受信することがあります。プレフィックスには SELinux の有効な用語のセット (ユーザー、ロール、タイプなど) を使用できます。ただし、プレフィックスをデフォルトとして残しておくと、サーバーはそれらを自動的に設定できます。たとえば、**s0:/2** はラベルを s0:c0,c0 から s0:c511,c511 に割り当て、**s0:/2,512** はラベルを s0:c0,c0,c0 から s0:c511,c511,511 に割り当てます。 ◦ **mcsLabelsPerProject** (整数): プロジェクトごとに予約するラベルの数を定義します。デフォルトは、デフォルトの UID と MCS の範囲に一致する **5** です。 ◦ **uidAllocatorRange** (文字列): プロジェクトに自動的に割り当てられる Unix ユーザー ID (UID) の合計セット数と、各 namespace が取得するブロックのサイズを定義します。たとえば、 **1000-1999/10** は namespace ごとに 10 の UID を割り当て、空間を使い果たす前に最大 100 のブロックを割り当てることが可能です。デフォルトでは、1 万のブロックに 10 億から 20 億を割り当てます。これは、ユーザーの namespace の起動時にコンテナーイメージについて予想される範囲のサイズです。
ProjectRequestMessage	この文字列は、プロジェクトの要求 API エンドポイントからプロジェクトを要求できない場合にユーザーに提示されます。

パラメーター名	説明
ProjectRequestTemplate	**projectrequest** への応答としてプロジェクトを作成する際に使用されるテンプレートです。フォーマットは namespace/template です。これはオプションであり、指定されていない場合はデフォルトのテンプレートが使用されます。

5.5.13. スケジューラーの設定

表5.13 スケジューラー設定パラメーター

パラメーター名	説明
SchedulerConfigFile	スケジューラーのセットアップ方法を記述しているファイルをポイントします。空の場合、デフォルトのスケジューリングルールが取得されます。

5.5.14. セキュリティーアロケーターの設定

表5.14 セキュリティーアロケーターパラメーター

パラメーター名	説明
MCSAllocatorRange	namespace に割り当てられる MCS カテゴリーの範囲を定義します。フォーマットは **<prefix>/<numberOfLabels>[, <maxCategory>]** です。デフォルトは **s0/2** であり、c0 から c1023 に割り当てられます。つまり、合計 535000 のラベルが利用可能であることを意味します (1024 は 2 ~ 535000 を選択します)。この値を起動後に変更すると、新規プロジェクトは、すでに別のプロジェクトに割り当てられているラベルを受信することがあります。プレフィックスには、SELinux の有効な用語のセット (ユーザー、ロール、タイプなど) にすることができます。ただしこれらをデフォルトとして残しておくと、サーバーはこれらを自動的に設定できます。
SecurityAllocator	UID と MCS ラベルのプロジェクトへの自動割り当てを制御します。空の場合、割り当ては無効にされます。
UIDAllocatorRange	プロジェクトに自動的に割り当てられる Unix ユーザー ID (UID) の合計セット数と、各 namespace が取得するブロックのサイズを定義します。たとえば、1000-1999/10 は namespace ごとに 10 の UID を割り当て、空間を使い果たす前に最大 100 のブロックを割り当てることが可能です。デフォルトでは、1 万のブロックに 10 億から 20 億 (ユーザー namespace の起動時にコンテナーイメージが使用する範囲の予想されるサイズ) を割り当てます。

5.5.15. サービスアカウントの設定

表5.15 サービスアカウント設定パラメーター

パラメーター名	説明
LimitSecretReferences	サービスアカウントに、明示的な参照なしに namespace のシークレットの参照を許可するかどうかを制御します。
ManagedNames	すべての namespace に自動作成されるサービスアカウント名の一覧です。名前が指定されていない場合、**ServiceAccountsController** は起動しません。
MasterCA	TLS 接続がマスターに戻っていることを確認する CA です。サービスアカウントのコントローラーは、マスターへの接続を検証できるようにこのファイルの内容を Pod に自動的に挿入します。
PrivateKeyFile	PEM でエンコードされた RSA プライベートキーを含むファイルです。これはサービスアカウントのトークンの署名に使用されます。プライベートキーが指定されていない場合、サービスアカウント **TokensController** は起動しません。
PublicKeyFiles	PEM でエンコードされた RSA パブリックキーを含むファイルの一覧です。いずれかのファイルにプライベートキーが含まれている場合、そのキーのパブリックの部分が使用されます。パブリックキーの一覧は、表示されているサービスアカウントのトークンの確認に使用されます。それぞれのキーは、一覧がすべて使用されるまで、または確認が正常に実行されるまで順番に試行されます。キーが指定されていない場合、サービスアカウントの認証は使用できません。

パラメーター名	説明
ServiceAccountConfig	サービスアカウントに関連するオプションを保持します。 • **LimitSecretReferences** (ブール値): サービスアカウントに、明示的な参照なしに namespace のシークレットの参照を許可するかどうかを制御します。 • **ManagedNames** (文字列): それぞれの namespace に自動作成されるサービスアカウント名の一覧です。名前が指定されていない場合、**ServiceAccountsController** は起動しません。 • **MasterCA** (文字列): TLS 接続がマスターに戻っていることを確認する認証局です。サービスアカウントコントローラーは、マスターへの接続を検証できるようにこのファイルの内容を Pod に自動的に挿入します。 • **PrivateKeyFile** (文字列): PEM でエンコードされた RSA プライベートキーが含まれ、サービスアカウントのトークンの署名に使用されます。プライベートキーが指定されていない場合、サービスアカウント **TokensController** は起動しません。 • **PublicKeyFiles** (文字列): PEM でエンコードされた RSA パブリックキーを含むファイルの一覧です。いずれかのファイルにプライベートキーが含まれている場合、OpenShift Container Platform はキーのパブリックの部分を使用します。パブリックキーの一覧は、サービスアカウントのトークンの確認に使用されます。それぞれのキーは、一覧がすべて使用されるまで、または確認が正常に実行されるまで順番に試行されます。キーが指定されていない場合、サービスアカウントの認証は使用できません。

5.5.16. 提供情報の設定

表5.16 提供情報設定パラメーター

パラメーター名	説明
AllowRecursiveQueries	マスター上の DNS サーバーがクエリーに再帰的に応答することを許可します。オープンリゾルバーは DNS アンプ攻撃に使用される可能であり、マスター DNS は公開ネットワークでアクセスできないようにする必要があることに注意してください。
BindAddress	提供に使用される **ip:port** です。
BindNetwork	イメージをインポートするための制限と動作を制御します。
CertFile	PEM でエンコードされた証明書を含むファイルです。
CertInfo	セキュアなトラフィックを提供するための TLS 証明書情報です。

パラメーター名	説明
ClientCA	クライアント証明書が受信される際にユーザーが認識するすべての署名者の証明書バンドルです。

dnsConfig	これがある場合、DNS サーバーが定義されたパラメーターに基づいて起動します。以下は例になります。
	`dnsConfig:`
	`  bindAddress: 0.0.0.0:8053`
	`  bindNetwork: tcp4`

DNSDomain	ドメインサフィックスを保持します。
DNSIP	IP を保持します。
KeyFile	**CertFile** が指定した証明書の PEM でエンコードされたプライベートキーを含むファイルです。
MasterClientConnectionOverrides	マスターへの接続に使用されたクライアント接続を上書きします。
MaxRequestsInFlight	サーバーに許可されている同時要求数です。ゼロの場合は無制限です。
NamedCertificates	特定のホスト名への要求を保護するのに使用される証明書の一覧です。
RequestTimeoutSecond	要求がタイムアウトするまでの秒数です。デフォルトは 60 分です。-1 の場合、要求について無制限となります。
ServingInfo	アセット用の HTTP 提供情報です。

5.5.17. ボリュームの設定

表5.17 ボリューム設定パラメーター

パラメーター名	説明
DynamicProvisioningEnabled	動的なプロビジョニングを有効化または無効化するブール値です。デフォルトは **true** です。
FSGroup	各ノードでそれぞれの FSGroup についてローカルストレージクォータを有効にします。現時点では、これは emptyDir ボリュームについて、基礎となる **volumeDirectory** が XFS ファイルシステムにある場合にのみ実装されます。
MasterVolumeConfig	マスターノードのボリュームプラグインを設定するオプションが含まれます。

パラメーター名	説明
NodeVolumeConfig	ノードのボリュームを設定するオプションが含まれます。
VolumeConfig	ノードのボリュームプラグインを設定するオプションが含まれます。 • **DynamicProvisioningEnabled** (ブール値): デフォルト値は **true** です。**false** の場合は動的プロビジョニングはオフに切り替わります。
VolumeDirectory	ボリュームがその下に保存されるディレクトリーです。

5.5.18. 基本的な監査

監査は、システムに影響を与えた一連のアクティビティーを個別のユーザー、管理者その他システムのコンポーネント別に記述したセキュリティー関連の時系列のレコードを提供します。

監査は API サーバーレベルで実行され、サーバーに送られるすべての要求をログに記録します。それぞれの監査ログには以下の 2 つのエントリーが含まれます。

1. 以下を含む要求行。
    a. 応答行 (以下の 2 を参照してください) と一致する固有 ID
    b. 要求のソース IP
    c. 呼び出されている HTTP メソッド
    d. 操作を呼び出している元のユーザー
    e. 操作を実行するための偽装ユーザー (**self** はユーザー自身を指します)
    f. 操作を実行するための偽装グループ (**lookup** はユーザーのグループを指します)
    g. 要求または <none> の namespace
    h. 要求される URI
2. 以下を含む応答行。
    a. 上記 1 の固有の ID
    b. 応答コード

Pod の一覧を要求するユーザー **admin** の出力例。

```
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" ip="127.0.0.1"
method="GET" user="admin" as="<self>" asgroups="<lookup>"
namespace="default" uri="/api/v1/namespaces/default/pods"
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" response="200"
```

openshift_master_audit_config 変数は、API サービス監査を有効にします。これは以下のオプションの配列を取ります。

表5.18 監査設定パラメーター

パラメーター名	説明
enabled	監査ログを有効または無効にするブール値です。デフォルト値は **false** です。
auditFilePath	要求をログに記録するファイルパスです。設定されていない場合、ログはマスターログに出力されます。
maximumFileRetentionDays	ファイル名にエンコードされるタイムスタンプに基づいて古い監査ログファイルを保持する最大日数を指定します。
maximumRetainedFiles	古い監査ログファイルを保持する最大日数を指定します。
maximumFileSizeMegabytes	ログファイルがローテーションされる前に、ファイルの最大サイズをメガバイトで指定します。デフォルトは 100 MB です。

監査の設定例

```
auditConfig:
  auditFilePath: "/var/log/audit-ocp.log"
  enabled: true
  maximumFileRetentionDays: 10
  maximumFileSizeMegabytes: 10
  maximumRetainedFiles: 10
```

監査ログの高度なセットアップ

監査ログの高度なセットアップを使用する必要がある場合には、以下を使用できます。

```
openshift_master_audit_config={"enabled": true}
```

auditFilePath のディレクトリーが、存在していない場合に作成されます。

```
openshift_master_audit_config={"enabled": true, "auditFilePath":
"/var/log/openpaas-oscp-audit/openpaas-oscp-audit.log",
"maximumFileRetentionDays": 14, "maximumFileSizeMegabytes": 500,
"maximumRetainedFiles": 5}
```

5.5.19. 高度な監査

高度な監査機能は、詳細なイベントのフィルタリングや複数の出力バックエンドなど、基本的な監査機能に対するいくつかの改良機能を提供します。以下の表に使用可能な追加オプションを示します。

表5.19 高度な監査設定パラメーター

パラメーター名	説明
policyFile	監査ポリシー設定を定義するファイルへのパスです。

パラメーター名	説明
policyConfiguration	組み込まれる監査ポリシー設定です。
logFormat	保存される監査ログのフォーマットを指定します。許可されている値は **legacy** (基本的な監査に使用されるフォーマット) と **json** です。
webHookKubeConfig	監査の Webhook 設定を定義する **.kubeconfig** でフォーマットされたファイルへのパスです。ここにイベントが送信されます。
webHookMode	監査イベントを送信するためのストラテジーを指定します。許可される値は **block** (直前のイベント処理が完了するまで別のイベントの処理をブロックする) と **batch** (イベントをバッファー処理し、バッチで提供する) です。

高度な監査機能を有効にするには、監査ポリシールールを記述する **policyFile** または **policyConfiguration** を指定する必要があります。

監査ポリシーの設定例

```
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:

  # Do not log watch requests by the "system:kube-proxy" on endpoints or
services
  - level: None 1
    users: ["system:kube-proxy"] 2
    verbs: ["watch"] 3
    resources: 4
    - group: ""
      resources: ["endpoints", "services"]

  # Do not log authenticated requests to certain non-resource URL paths.
  - level: None
    userGroups: ["system:authenticated"] 5
    nonResourceURLs: 6
    - "/api*" # Wildcard matching.
    - "/version"

  # Log the request body of configmap changes in kube-system.
  - level: Request
    resources:
    - group: "" # core API group
      resources: ["configmaps"]
    # This rule only applies to resources in the "kube-system" namespace.
    # The empty string "" can be used to select non-namespaced resources.
    namespaces: ["kube-system"] 7

  # Log configmap and secret changes in all other namespaces at the
metadata level.
```

```
  - level: Metadata
    resources:
    - group: "" # core API group
      resources: ["secrets", "configmaps"]

  # Log all other resources in core and extensions at the request level.
  - level: Request
    resources:
    - group: "" # core API group
    - group: "extensions" # Version of group should NOT be included.

  # A catch-all rule to log all other requests at the Metadata level.
  - level: Metadata 8

  # Log login failures from the web console or CLI. Review the logs and
refine your policies.
  - level: Metadata
    nonResourceURLs:
    - /login* 9
    - /oauth* 10
```

1 8 すべてのイベントは以下の 4 つのレベルでログに記録できます。

- **None** - このルールに一致するイベントは記録されません。
- **Metadata** - 要求のメタデータ (要求しているユーザー、タイムスタンプ、リソース、verb など) をログに記録します。要求または応答本体はログに記録しません。基本的な監査で使用されるレベルと同じレベルになります。
- **Request** - イベントのメタデータと要求本体をログに記録します。応答本体はログに記録しません。
- **RequestResponse** - イベントのメタデータ、要求、および応答本体をログに記録します。

2 このルールが適用されるユーザーの一覧です。一覧が空の場合はすべてのユーザーに適用されます。

3 このルールが適用される verb の一覧です。一覧が空の場合はすべての verb に適用されます。これは API 要求に関連付けられる Kubernetes の verb です (**get**、**list**、**watch**、**create**、**update**、**patch**、**delete**、**deletecollection**、**proxy** など)。

4 このルールが適用されるリソースの一覧です。一覧が空の場合はすべてのリソースに適用されます。各リソースは、それが割り当てられるグループ (例: 空の場合は Kubernetes core API、バッチ、build.openshift.io などを指します) 、およびそのグループのリソース一覧として指定されます。

5 このルールが適用されるグループの一覧です。一覧が空の場合はすべてのグループに適用されます。

6 このルールが適用されるリソース以外の URL の一覧です。

7 このルールが適用される namespace の一覧です。一覧が空の場合はすべての namespace に適用されます。

9 Web コンソールが使用するエンドポイントです。

10 CLI が使用するエンドポイントです。

高度な監査についての詳細は、Kubernetes のドキュメントを参照してください。

5.5.20. etcd の TLS 暗号の指定

マスターと etcd サーバー間の通信で使用するサポートされている TLS 暗号を指定できます。

1. 各 etcd ノードで、etcd をアップグレードします。

```
# yum update etcd iptables-services
```

2. お使いのバージョンが 3.2.22 以降であることを確認します。

```
# etcd --version
etcd Version: 3.2.22
```

3. 各マスターホストで、**/etc/origin/master/master-config.yaml** ファイルで有効にする暗号を指定します。

```
servingInfo:
  ...
  minTLSVersion: VersionTLS12
  cipherSuites:
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_RSA_WITH_AES_256_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
...
```

4. 各マスターホストで、マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

5. 暗号が適用されていることを確認します。たとえば、TLSv1.2 暗号 **ECDHE-RSA-AES128-GCM-SHA256** については、以下のコマンドを実行します。

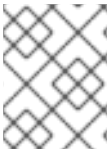
```
# openssl s_client -connect etcd1.example.com:2379 1
CONNECTED(00000003)
depth=0 CN = etcd1.example.com
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = etcd1.example.com
verify error:num=21:unable to verify the first certificate
verify return:1
139905367488400:error:14094412:SSL routines:ssl3_read_bytes:sslv3
alert bad certificate:s3_pkt.c:1493:SSL alert number 42
139905367488400:error:140790E5:SSL routines:ssl23_write:ssl
handshake failure:s23_lib.c:177:
---
Certificate chain
```

```
 0 s:/CN=etcd1.example.com
   i:/CN=etcd-signer@1529635004
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEkjCCAnqgAwIBAgIBATANBgkqhkiG9w0BAQsFADAhMR8wHQYDVQQDDBZldGNk
........
....
eif87qttt0Sl1vS8DG1KQO1oOBlNkg==
-----END CERTIFICATE-----
subject=/CN=etcd1.example.com
issuer=/CN=etcd-signer@1529635004
---
Acceptable client certificate CA names
/CN=etcd-signer@1529635004
Client Certificate Types: RSA sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA256:ECDSA+SHA256:RSA+SHA384:ECDSA+SHA384:RSA+SHA1:ECDSA+SHA1
Shared Requested Signature Algorithms:
RSA+SHA256:ECDSA+SHA256:RSA+SHA384:ECDSA+SHA384:RSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA384
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 1666 bytes and written 138 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID:
    Session-ID-ctx:
    Master-Key:
1EFA00A91EE5FC5EDDCFC67C8ECD060D44FD3EB23D834EDED929E4B74536F273C0F9
299935E5504B562CD56E76ED208D
    Key-Arg   : None
    Krb5 Principal: None
    PSK identity: None
    PSK identity hint: None
    Start Time: 1529651744
    Timeout   : 300 (sec)
    Verify return code: 21 (unable to verify the first certificate)
```

1 **etcd1.example.com** は etcd ホストの名前です。

5.6. ノード設定ファイル

以下の **node-config.yaml** ファイルは、現時点でデフォルト値で生成されるノード設定ファイルのサンプルです。



注記

クラスターのノードを変更するには、ノード設定マップを必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

例5.1 ノード設定ファイルのサンプル

```
allowDisabledDocker: false
apiVersion: v1
authConfig:
  authenticationCacheSize: 1000
  authenticationCacheTTL: 5m
  authorizationCacheSize: 1000
  authorizationCacheTTL: 5m
dnsDomain: cluster.local
dnsIP: 0.0.0.0 (1)
dockerConfig:
  execHandlerName: native
imageConfig:
  format: openshift/origin-${component}:${version}
  latest: false
iptablesSyncPeriod: 5s
kind: NodeConfig
masterKubeConfig: node.kubeconfig
networkConfig:
  mtu: 1450
  networkPluginName: ""
nodeIP: ""
nodeName: node1.example.com
podManifestConfig: (2)
  path: "/path/to/pod-manifest-file" (3)
  fileCheckIntervalSeconds: 30 (4)
proxyArguments:
  proxy-mode:
  - iptables (5)
servingInfo:
  bindAddress: 0.0.0.0:10250
  bindNetwork: tcp4
  certFile: server.crt
  clientCA: node-client-ca.crt
  keyFile: server.key
  namedCertificates: null
volumeDirectory: /root/openshift.local.volumes
```

(1) ここにアドレスを追加することで、Pod の **/etc/resolv.conf** の先頭に追加される IP アドレスを設定します。

(2) 特定のノードセット上、またはすべてのノード上に、スケジューラーを介することなく Pod を直接配置することを許可します。ユーザーは、Pod を使って同じ管理タスクを実行し、各ノードで同じサービスをサポートすることができます。

(3) Pod のマニフェストファイルまたはディレクトリーへのパスを指定します。ディレクトリーの場合、1 つ以上のマニフェストファイルが含まれることが予想されます。これは、Pod をノード上に作成するために Kubelet によって使用されます。

4 新規データのマニフェストファイルをチェックする間隔 (秒単位) です。この値は正の値で入力する必要があります。

5 使用するサービスプロキシーの実装です。

ノード設定ファイルはノードのリソースを決定します。詳細は、「Allocating node resources section in the Cluster Administrator guide」を参照してください。

5.6.1. Pod とノードの設定

表5.20 Pod とノードの設定パラメーター

パラメーター名	説明
NodeConfig	OpenShift Container Platform ノードを起動する完全に指定された設定です。
NodeIP	ノードは複数の IP を持つことがあるため、Pod のトラフィックルーティングに使用する IP を指定します。指定されていない場合、**nodeName** でネットワーク解析/ルックアップが実行され、最初の非ループバックアドレスが使用されます。
NodeName	クラスターの特定ノードを識別するために使用される値です。可能な場合、この値はユーザーの完全修飾ホスト名にできます。ユーザーが静的ノードのセットをマスターに記述している場合、この値は一覧にある値のいずれかに一致している必要があります。
PodEvictionTimeout	失敗したノード上の Pod を削除するための猶予期間を制御します。有効な時間文字列を取ります。空の場合、Pod のデフォルトのエビクションタイムアウトを取ります。
ProxyClientInfo	Pod へのプロキシー処理時に使用するクライアント証明書/キーを指定します。

5.6.2. Docker の設定

表5.21 Docker 設定パラメーター

パラメーター名	説明
AllowDisabledDocker	true の場合、Kubelet は Docker のエラーを無視します。これは、Docker を起動させていないマシンでノードを起動できることを意味します。
DockerConfig	Docker 関連の設定オプションを保持します。
ExecHandlerName	Docker コンテナーでのコマンドの実行に使用するハンドラーです。

5.6.3. ローカルストレージの設定

XFS クォータサブシステムを使用して **emptyDir** ボリューム、および各ノードのシークレットおよび設定マップなどの **emptyDir** ボリュームに基づくボリュームのサイズを制限できます。

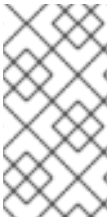
XFS ファイルシステムで **emptyDir** ボリュームのサイズを制限するには、**openshift-node** プロジェクトで **node-config-compute** 設定マップを使用し、それぞれ一意の FSGroup についてローカルボリュームクォータを設定します。

```
apiVersion: kubelet.config.openshift.io/v1
kind: VolumeConfig
  localQuota: 1
    perFSGroup: 1Gi 2
```

1 ノードのローカルボリュームクォータを制御するオプションが含まれます。

2 この値を、**1Gi**、**512Mi** など [FSGroup] 別、ノード別に必要なクォータを示すリソース量に設定します **volumeDirectory** は **grpquota** オプションを指定してマウントされた XFS ファイルシステムになければなりません。一致する SCC (security context constraint) の **fsGroup** タイプは **MustRunAs** に設定する必要があります。

FSGroup が指定されていない場合、つまり要求が **RunAsAny** の SCC に一致することを示す場合、クォータの適用は省略されます。

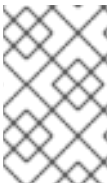


注記

/etc/origin/node/volume-config.yaml ファイルは直接編集しないでください。このファイルは **node-config-compute** 設定マップを基に作成されます。**node-config-compute** 設定マップを使用して **volume-config.yaml** ファイルでパラメーターの作成または編集を実行します。

5.6.4. Docker 1.9 以降を使用したイメージの並行プル

Docker 1.9 以降を使用している場合は、イメージの並行プルを有効にしておくことができます。デフォルトでは、イメージは一度に 1 つずつプルされます。



注記

Docker 1.9 よりも前のバージョンでは、データの破損による問題が発生する可能性があります。1.9 以降では破損の問題は解消し、並行プルに安全に切り替えることができます。

```
kubeletArguments:
  serialize-image-pulls:
  - "false" 1
```

1 並行プルを無効にするには true に変更します (デフォルトの設定)。

5.7. パスワードおよびその他の機密データ

認証設定によっては、LDAP **bindPassword** または OAuth **clientSecret** の値が必須になる場合があります。これらの値は、マスター設定ファイルに直接指定する代わりに、環境変数、外部ファイルまたは暗号化ファイルとして指定することができます。

環境変数の例

```
...
bindPassword:
  env: BIND_PASSWORD_ENV_VAR_NAME
```

外部ファイルの例

```
...
bindPassword:
  file: bindPassword.txt
```

暗号化された外部ファイルの例

```
...
bindPassword:
  file: bindPassword.encrypted
  keyFile: bindPassword.key
```

上記の例の暗号化ファイルとキーファイルを作成するには、以下を入力します。

```
$ oc adm ca encrypt --genkey=bindPassword.key --out=bindPassword.encrypted
> Data to encrypt: B1ndPass0rd!
```

Ansible ホストインベントリーファイル (デフォルトで **/etc/ansible/hosts**) に最初に一覧表示されているマスターから **oc adm** コマンドを実行します。

警告

暗号化データのセキュリティーレベルは復号化キーと同程度です。ファイルシステムのパーミッションとキーファイルへのアクセスを制限する際には十分な注意が必要です。

5.8. 新規設定ファイルの作成

OpenShift Container Platform 設定をゼロから定義するときは、新規設定ファイルを作成することから開始します。

マスターホストの設定ファイルでは、**openshift start** コマンドを **--write-config** オプションと共に使用して設定ファイルを作成します。ノードホストの場合は、**oc adm create-node-config** コマンドを使用して設定ファイルを作成します。

以下のコマンドは、関連する起動設定ファイル、証明書ファイルその他ファイルを指定された **--write-config** または **--node-dir** ディレクトリーに書き込みます。

生成される証明書ファイルは 2 年間有効です。一方、認証局 (CA) の証明書は 5 年間有効です。この値は **--expire-days** と **--signer-expire-days** のオプションを使用して変更できますが、セキュリティー上の理由によりこの値をこれ以上大きい値に変更しないことが推奨されます。

オールインワンサーバー (マスターとノードが同一ホスト上にある) の設定ファイルを指定されたディレクトリーに作成するには、以下を入力します。

```
$ openshift start --write-config=/openshift.local.config
```

マスター設定ファイルとその他の必須ファイルを指定されたディレクトリーに作成するには、以下を実行します。

```
$ openshift start master --write-config=/openshift.local.config/master
```

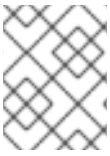
ノード設定ファイルとその他の関連ファイルを指定されたディレクトリーに作成するには、以下を実行します。

```
$ oc adm create-node-config \
    --node-dir=/openshift.local.config/node-<node_hostname> \
    --node=<node_hostname> \
    --hostnames=<node_hostname>,<ip_address> \
    --certificate-authority="/path/to/ca.crt" \
    --signer-cert="/path/to/ca.crt" \
    --signer-key="/path/to/ca.key"
    --signer-serial="/path/to/ca.serial.txt"
    --node-client-certificate-authority="/path/to/ca.crt"
```

ノード設定ファイルを作成する際に、**--hostnames** オプションは、サーバー証明書を有効にする必要のあるすべてのホスト名または IP アドレスのコンマ区切りの一覧を受け入れます。

5.9. 設定ファイルの使用によるサーバーの起動

マスターまたはノード設定ファイルをユーザー仕様に変更すると、これを引数として指定してサーバーを起動すると、使用できるようになります。設定ファイルを指定する場合は、ユーザーが渡す他のコマンドラインオプションはいずれも認識されないことに注意してください。



注記

クラスターのノードを変更するには、ノード設定マップを必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

マスター設定およびノード設定ファイルを使用してオールインワンサーバーを起動するには、以下を実行します。

```
$ openshift start --master-config=/openshift.local.config/master/master-
config.yaml --node-config=/openshift.local.config/node-
<node_hostname>/node-config.yaml
```

マスター設定ファイルを使用してマスターサーバーを起動するには、以下を実行します。

```
$ openshift start master --config=/openshift.local.config/master/master-
config.yaml
```

ノード設定ファイルを使ってノードサーバーを起動するには、以下を実行します。

```
$ openshift start node --config=/openshift.local.config/node-
<node_hostname>/node-config.yaml
```

5.10. ロギングレベルの設定

OpenShift Container Platform は、5 段階のログメッセージの重要度を用いて、**systemd-journald.service** を使ってデバッグ用にログメッセージを収集します。ロギングレベルは、以下のような Kubernetes のロギング規則に基づいています。

表5.22 ログレベルのオプション

オプション	説明
0	エラーと警告のみ
2	通常の情報
4	デバッグレベルの情報
6	API レベルのデバッグ情報 (要求 / 応答)
8	本体レベルの API のデバッグ情報

ユーザーは、**/etc/sysconfig/atomic-openshift-node**、**/etc/sysconfig/atomic-openshift-master-api** ファイル、および **/etc/sysconfig/atomic-openshift-master-controllers** ファイルにログレベルのオプションを設定することにより、ログに記録する INFO メッセージを制御できます。すべてのメッセージを収集するようにログを設定すると、解釈が困難な膨大なログが生成され、多くの領域が占領されます。すべてのメッセージの収集は、デバッグで使用する場合にとどめる必要があります。

注記

FATAL、ERROR、WARNING、および一部の INFO の重要度を伴うメッセージは、ログの設定に関係なくログに表示されます。

マスターまたはノードシステムのログは、以下のコマンドで表示できます。

```
# journalctl -r -u <journal_name>
```

最新のエントリーから表示するには、**-r** オプションを使用します。

以下は例を示しています。

```
# journalctl -r -u atomic-openshift-master-controllers
# journalctl -r -u atomic-openshift-master-api
# journalctl -r -u atomic-openshift-node.service
```

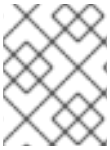
ロギングレベルを変更するには、以下を実行します。

1. マスターの **/etc/sysconfig/atomic-openshift-master** ファイル、またはノードの **/etc/sysconfig/atomic-openshift-node** ファイルを編集します。
2. 上記の **Log Level Options** 表の値を **OPTIONS=--loglevel=** フィールドに入力します。以下は例を示しています。

```
OPTIONS=--loglevel=4
```

3. マスターまたはノードを再起動します。「OpenShift Container Platform サービスの再起動」を参照してください。

再起動後は、すべての新しいログメッセージは新しい設定に従ったメッセージに従います。古いメッセージは変更されません。



注記

デフォルトのログレベルは標準のクラスターインストールプロセスを使用して設定できます。詳細は、「クラスター変数」を参照してください。

以下の例は、各種のログレベルのマスター **journald** ログの抜粋です。タイムスタンプとシステム情報はこれらの例では削除されています。

loglevel = 0 の journalctl -u atomic-openshift-master-controllers.service 出力の抜粋

```
4897 plugins.go:77] Registered admission plugin "NamespaceLifecycle"
4897 start_master.go:290] Warning: assetConfig.loggingPublicURL: Invalid
value: "": required to view aggregated container logs in the console,
master start will continue.
4897 start_master.go:290] Warning: assetConfig.metricsPublicURL: Invalid
value: "": required to view cluster metrics in the console, master start
will continue.
4897 start_master.go:290] Warning: aggregatorConfig.proxyClientInfo:
Invalid value: "": if no client certificate is specified, the aggregator
will be unable to proxy to remote servers,
4897 start_master.go:412] Starting controllers on 0.0.0.0:8444 (v3.7.14)
4897 start_master.go:416] Using images from "openshift3/ose-
<component>:v3.7.14"
4897 standalone_apiserver.go:106] Started health checks at 0.0.0.0:8444
4897 plugins.go:77] Registered admission plugin "NamespaceLifecycle"
4897 configgetter.go:53] Initializing cache sizes based on 0MB limit
4897 leaderelection.go:105] Attempting to acquire openshift-master-
controllers lease as master-bkr-hv03-guest44.dsal.lab.eng.bos.redhat.com-
10.19.41.74-xtz6lbqb, renewing every 3s, hold
4897 leaderelection.go:179] attempting to acquire leader lease...
systemd[1]: Started Atomic OpenShift Master Controllers.
4897 leaderelection.go:189] successfully acquired lease kube-
system/openshift-master-controllers
4897 event.go:218] Event(v1.ObjectReference{Kind:"ConfigMap",
Namespace:"kube-system", Name:"openshift-master-controllers",
UID:"aca86731-ffbe-11e7-8d33-525400c845a8", APIVersion:"v1",
4897 start_master.go:627] Started serviceaccount-token controller
4897 factory.go:351] Creating scheduler from configuration: {{ }
[{NoVolumeZoneConflict <nil>} {MaxEBSVolumeCount <nil>}
{MaxGCEPDVolumeCount <nil>} {MaxAzureDiskVolumeCount <nil>} {Mat
4897 factory.go:360] Registering predicate: NoVolumeZoneConflict
```

```
4897 plugins.go:145] Predicate type NoVolumeZoneConflict already
registered, reusing.
4897 factory.go:360] Registering predicate: MaxEBSVolumeCount
4897 plugins.go:145] Predicate type MaxEBSVolumeCount already registered,
reusing.
4897 factory.go:360] Registering predicate: MaxGCEPDVolumeCount
```

loglevel = 2 の journalctl -u atomic-openshift-master-controllers.service 出力の抜粋

```
4897 master.go:47] Initializing SDN master of type "redhat/openshift-ovs-
subnet"
4897 master.go:107] Created ClusterNetwork default (network:
"10.128.0.0/14", hostSubnetBits: 9, serviceNetwork: "172.30.0.0/16",
pluginName: "redhat/openshift-ovs-subnet")
4897 start_master.go:690] Started "openshift.io/sdn"
4897 start_master.go:680] Starting "openshift.io/service-serving-cert"
4897 controllermanager.go:466] Started "namespace"
4897 controllermanager.go:456] Starting "daemonset"
4897 controller_utils.go:1025] Waiting for caches to sync for namespace
controller
4897 shared_informer.go:298] resyncPeriod 120000000000 is smaller than
resyncCheckPeriod 600000000000 and the informer has already started.
Changing it to 600000000000
4897 start_master.go:690] Started "openshift.io/service-serving-cert"
4897 start_master.go:680] Starting "openshift.io/image-signature-import"
4897 start_master.go:690] Started "openshift.io/image-signature-import"
4897 start_master.go:680] Starting "openshift.io/templateinstance"
4897 controllermanager.go:466] Started "daemonset"
4897 controllermanager.go:456] Starting "statefulset"
4897 daemoncontroller.go:222] Starting daemon sets controller
4897 controller_utils.go:1025] Waiting for caches to sync for daemon sets
controller
4897 controllermanager.go:466] Started "statefulset"
4897 controllermanager.go:456] Starting "cronjob"
4897 stateful_set.go:147] Starting stateful set controller
4897 controller_utils.go:1025] Waiting for caches to sync for stateful set
controller
4897 start_master.go:690] Started "openshift.io/templateinstance"
4897 start_master.go:680] Starting "openshift.io/horizontalpodautoscaling
```

loglevel = 4 の journalctl -u atomic-openshift-master-controllers.service 出力の抜粋

```
4897 factory.go:366] Registering priority: Zone
4897 factory.go:401] Creating scheduler with fit predicates
'map[GeneralPredicates:{} CheckNodeMemoryPressure:{}
CheckNodeDiskPressure:{} Region:{} NoVolumeZoneC
4897 controller_utils.go:1025] Waiting for caches to sync for tokens
controller
4897 controllermanager.go:108] Version: v1.7.6+a08f5eeb62
4897 leaderelection.go:179] attempting to acquire leader lease...
4897 leaderelection.go:189] successfully acquired lease kube-system/kube-
controller-manager
4897 event.go:218] Event(v1.ObjectReference{Kind:"ConfigMap",
Namespace:"kube-system", Name:"kube-controller-manager", UID:"acb3e9c6-
ffbe-11e7-8d33-525400c845a8", APIVersion:"v1", Resou
```

```
4897 controller_utils.go:1032] Caches are synced for tokens controller
4897 plugins.go:101] No cloud provider specified.
4897 controllermanager.go:481] "serviceaccount-token" is disabled
4897 controllermanager.go:450] "bootstrapsigner" is disabled
4897 controllermanager.go:450] "tokencleaner" is disabled
4897 controllermanager.go:456] Starting "garbagecollector"
4897 start_master.go:680] Starting "openshift.io/build"
4897 controllermanager.go:466] Started "garbagecollector"
4897 controllermanager.go:456] Starting "deployment"
4897 garbagecollector.go:126] Starting garbage collector controller
4897 controller_utils.go:1025] Waiting for caches to sync for garbage
collector controller
4897 controllermanager.go:466] Started "deployment"
4897 controllermanager.go:450] "horizontalpodautoscaling" is disabled
4897 controllermanager.go:456] Starting "disruption"
4897 deployment_controller.go:152] Starting deployment controller
```

loglevel = 8 の journalctl -u atomic-openshift-master-controllers.service 出力の抜粋

```
4897 plugins.go:77] Registered admission plugin "NamespaceLifecycle"
4897 start_master.go:290] Warning: assetConfig.loggingPublicURL: Invalid
value: "": required to view aggregated container logs in the console,
master start will continue.
4897 start_master.go:290] Warning: assetConfig.metricsPublicURL: Invalid
value: "": required to view cluster metrics in the console, master start
will continue.
4897 start_master.go:290] Warning: aggregatorConfig.proxyClientInfo:
Invalid value: "": if no client certificate is specified, the aggregator
will be unable to proxy to remote serv
4897 start_master.go:412] Starting controllers on 0.0.0.0:8444 (v3.7.14)
4897 start_master.go:416] Using images from "openshift3/ose-
<component>:v3.7.14"
4897 standalone_apiserver.go:106] Started health checks at 0.0.0.0:8444
4897 plugins.go:77] Registered admission plugin "NamespaceLifecycle"
4897 configgetter.go:53] Initializing cache sizes based on 0MB limit
4897 leaderelection.go:105] Attempting to acquire openshift-master-
controllers lease as master-bkr-hv03-guest44.dsal.lab.eng.bos.redhat.com-
10.19.41.74-xtz6lbqb, renewing every 3s,
4897 leaderelection.go:179] attempting to acquire leader lease...
systemd[1]: Started Atomic OpenShift Master Controllers.
4897 leaderelection.go:189] successfully acquired lease kube-
system/openshift-master-controllers
4897 event.go:218] Event(v1.ObjectReference{Kind:"ConfigMap",
Namespace:"kube-system", Name:"openshift-master-controllers",
UID:"aca86731-ffbe-11e7-8d33-525400c845a8", APIVersion:"
4897 start_master.go:627] Started serviceaccount-token controller
```

loglevel = 2 の journalctl -u atomic-openshift-master-api.service 出力の抜粋

```
4613 plugins.go:77] Registered admission plugin "NamespaceLifecycle"
4613 master.go:320] Starting Web Console https://bkr-hv03-
guest44.dsal.lab.eng.bos.redhat.com:8443/console/
4613 master.go:329] Starting OAuth2 API at /oauth
4613 master.go:320] Starting Web Console https://bkr-hv03-
guest44.dsal.lab.eng.bos.redhat.com:8443/console/
```

```
4613 master.go:329] Starting OAuth2 API at /oauth
4613 master.go:320] Starting Web Console https://bkr-hv03-
guest44.dsal.lab.eng.bos.redhat.com:8443/console/
4613 master.go:329] Starting OAuth2 API at /oauth
4613 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
4613 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/01/22 16:53:14 log.go:33: [restful/swagger] listing is
available at https://bkr-hv03-
guest44.dsal.lab.eng.bos.redhat.com:8443/swaggerapi
[restful] 2018/01/22 16:53:14 log.go:33: [restful/swagger] https://bkr-
hv03-guest44.dsal.lab.eng.bos.redhat.com:8443/swaggerui/ is mapped to
folder /swagger-ui/
4613 master.go:320] Starting Web Console https://bkr-hv03-
guest44.dsal.lab.eng.bos.redhat.com:8443/console/
4613 master.go:329] Starting OAuth2 API at /oauth
4613 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
4613 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/01/22 16:53:14 log.go:33: [restful/swagger] listing is
available at https://bkr-hv03-
guest44.dsal.lab.eng.bos.redhat.com:8443/swaggerapi
[restful] 2018/01/22 16:53:14 log.go:33: [restful/swagger] https://bkr-
hv03-guest44.dsal.lab.eng.bos.redhat.com:8443/swaggerui/ is mapped to
folder /swagger-ui/
Starting Web Console https://bkr-hv03-
guest44.dsal.lab.eng.bos.redhat.com:8443/console/
Starting OAuth2 API at /oauth
No API exists for predefined swagger description /oapi/v1
No API exists for predefined swagger description /api/v1
```

5.11. マスターおよびノードサービスの再起動

マスターまたはノード設定の変更を適用するには、それぞれのサービスを再起動する必要がります。

マスター設定の変更を再度読み込むには、**master-restart** コマンドを使用してコントロールプレーンの静的 Pod で実行されているマスターサービスを再起動します。

```
# master-restart api
```

ノード設定の変更を再度読み込むには、ノードホストでノードサービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

第6章 OPENSHIFT ANSIBLE BROKER の設定

6.1. 概要

OpenShift Ansible Broker (OAB) をクラスターにデプロイする際に、その動作の大半は、起動時に読み込まれるブローカーの設定ファイルによって決定されます。ブローカーの設定は、ブローカーの namespace (デフォルトでは (**openshift-ansible-service-broker**) に ConfigMap オブジェクトとして格納されます。

OpenShift Ansible Broker 設定ファイルの例

```
registry: 1
  - type: dockerhub
    name: docker
    url: https://registry.hub.docker.com
    org: <dockerhub_org>
    fail_on_error: false
  - type: rhcc
    name: rhcc
    url: https://registry.access.redhat.com
    fail_on_error: true
    white_list:
      - "^foo.*-apb$"
      - ".*-apb$"
    black_list:
      - "bar.*-apb$"
      - "^my-apb$"
  - type: local_openshift
    name: lo
    namespaces:
      - openshift
    white_list:
      - ".*-apb$"
dao: 2
  etcd_host: localhost
  etcd_port: 2379
log: 3
  logfile: /var/log/ansible-service-broker/asb.log
  stdout: true
  level: debug
  color: true
openshift: 4
  host: ""
  ca_file: ""
  bearer_token_file: ""
  image_pull_policy: IfNotPresent
  sandbox_role: "edit"
  keep_namespace: false
  keep_namespace_on_error: true
broker: 5
  bootstrap_on_startup: true
  dev_broker: true
  launch_apb_on_bind: false
  recovery: true
```

```
    output_request: true
    ssl_cert_key: /path/to/key
    ssl_cert: /path/to/cert
    refresh_interval: "600s"
    auth:
      - type: basic
        enabled: true
  secrets: 6
    - title: Database credentials
      secret: db_creds
      apb_name: dh-rhscl-postgresql-apb
```

1. 詳細は「レジストリー設定」を参照してください。
2. 詳細は「DAO 設定」を参照してください。
3. 詳細は「ログ設定」を参照してください。
4. 詳細は「OpenShift 設定」を参照してください。
5. 詳細は「ブローカー設定」を参照してください。
6. 詳細は「シークレット設定」を参照してください。

6.2. OPENSHIFT ANSIBLE BROKER 設定の変更

OAB のデフォルト設定をデプロイした後に変更するには、以下を実行します。

1. OAB の namespace の **broker-config** ConfigMap オブジェクトを、**cluster-admin** 権限を持つユーザーとして編集します。

```
$ oc edit configmap broker-config -n openshift-ansible-service-
broker
```

2. 更新内容を保存した後、変更を有効にするために OAB のデプロイメント設定を再デプロイします。

```
$ oc rollout latest dc/asb -n openshift-ansible-service-broker
```

6.3. レジストリー設定

registry セクションでは、ブローカーが APB 用に参照する必要があるレジストリーを定義できます。

表6.1 registry セクションの設定オプション

フィールド	説明	必須
name	レジストリーの名前です。このレジストリーから APB を識別するためにブローカーによって使用されます。	Y

フィールド	説明	必須
user	レジストリーに対して認証するためのユーザー名です。**auth_type** が **secret** または **file** に設定されている場合は使用されません。	N
pass	レジストリーに対して認証するためのパスワードです。**auth_type** が **secret** または **file** に設定されている場合は使用されません。	N
auth_type	レジストリー認証情報が **user** と **pass** でブローカー設定に定義されていない場合にブローカーがレジストリー認証情報を読み取る方法です。**secret** (シークレットをブローカー namespace で使用する) または **file** (マウントされたファイルを使用する) を指定できます。	N
auth_name	読み取る必要があるレジストリー認証情報を格納しているシークレットまたはファイルの名前です。**auth_type** が **secret** に設定されている場合に使用されます。	N (**auth_type** が **secret** または **file** に設定されているに場合にのみ必要)
org	イメージが含まれている namespace または組織です。	N
type	レジストリーのタイプです。使用可能なアダプターは **mock**、**rhcc**、**openshift**、**dockerhub**、および **local_openshift** です。	Y
namespaces	**local_openshift** レジストリータイプの設定に使用する namespace の一覧です。デフォルトでは、ユーザーは **openshift** を使用する必要があります。	N
url	イメージ情報を取得するために使用される URL です。これは、RHCC の場合に広範囲に使用されます。**dockerhub** タイプでは、ハードコードされた URL が使用されます。	N
fail_on_error	このレジストリーが失敗した場合にブートストラップ要求を失敗させるかどうかを指定します。失敗させる場合、その他のレジストリーの読み込みの実行を停止します。	N
white_list	許可されるイメージ名を定義するための正規表現の一覧です。カタログへの APB の追加を許可するホワイトリストを作成する必要があります。レジストリー内のすべての APB を取得する必要がある場合は、最も許容度の高い正規表現である **.*-apb$** を使用できます。詳細については、「APB のフィルタリング」を参照してください	N

フィールド	説明	必須
black_list	許可できないイメージ名を定義するために使用される正規表現の一覧です。詳細については、「APB のフィルタリング」を参照してください。	N
images	OpenShift Container レジストリーで使用されるイメージの一覧です。	N

6.3.1. 実稼働または開発

実稼働ブローカー設定は、Red Hat Container Catalog (RHCC) などの信頼できるコンテナーディストリビューションレジストリーを参照するように設計されています。

```
registry:
  - name: rhcc
    type: rhcc
    url: https://registry.access.redhat.com
    tag: v3.10
    white_list:
      - ".*-apb$"
  - type: local_openshift
    name: localregistry
    namespaces:
      - openshift
    white_list: []
```

開発ブローカー設定は、主にブローカーの開発作業に取り組む開発者によって使用されます。開発者設定を有効にするには、レジストリー名を **dev** に設定し、**broker** セクションの **dev_broker**フィールドを **true** に設定します。

```
registry:
  name: dev
```

```
broker:
  dev_broker: true
```

6.3.2. レジストリー認証情報の保存

ブローカー設定は、ブローカーによるレジストリー認証情報の読み取り方法を決定します。レジストリー認証情報は、**registry** セクションの **user** 値と **pass** 値から読み取ることができます。以下は例になります。

```
registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    user: <user>
    pass: <password>
```

これらの認証情報にパブリックにアクセスできないようにするには、**registry** セクションの

auth_type フィールドを **secret** または **file** タイプに設定します。**secret** タイプは、ブローカーの namespace からシークレットを使用するようにレジストリーを設定します。一方、**file** タイプは、ボリュームとしてマウントされているシークレットを使用するようにレジストリーを設定します。

secret または **file** タイプを使用するには、以下を実行します。

1. 関連するシークレットには、**username** と **password** の値が定義されている必要があります。シークレットを使用する場合は、**openshift-ansible-service-broker** namespace が存在していることを確認する必要があります。シークレットはこの namespace から読み取られるためです。
   たとえば、**reg-creds.yaml** ファイルを作成します。

```
$ cat reg-creds.yaml
---
username: <user_name>
password: <password>
```

2. このファイルから **openshift-ansible-service-broker** namespace にシークレットを作成します。

```
$ oc create secret generic \
    registry-credentials-secret \
    --from-file reg-creds.yaml \
    -n openshift-ansible-service-broker
```

3. **secret** または **file** のどちらのタイプを使用するか選択します。

   - **secret** タイプを使用するには、以下を実行します。

     a. ブローカー設定で、**auth_type** を **secret** に、**auth_name** をシークレットの名前に設定します。

```
registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    auth_type: secret
    auth_name: registry-credentials-secret
```

     b. シークレットが置かれている namespace を設定します。

```
openshift:
  namespace: openshift-ansible-service-broker
```

   - **file** タイプを使用するには、以下を実行します。

     a. **asb** デプロイメント設定を編集し、ファイルを **/tmp/registry-credentials/reg-creds.yaml** にマウントします。

```
$ oc edit dc/asb -n openshift-ansible-service-broker
```

     containers.volumeMounts セクションに、以下を追加します。

```
volumeMounts:
  - mountPath: /tmp/registry-credentials
    name: reg-auth
```

volumes セクションに、以下を追加します。

```
    volumes:
      - name: reg-auth
        secret:
          defaultMode: 420
          secretName: registry-credentials-secret
```

b. ブローカー設定で、**auth_type** を **file** に、**auth_type** をファイルの場所に設定します。

```
registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    auth_type: file
    auth_name: /tmp/registry-credentials/reg-creds.yaml
```

6.3.3. モックレジストリー

モックレジストリーは、ローカルの APB 仕様を読み取る場合に便利です。イメージ仕様を検索するために外部のレジストリーにアクセスする代わりに、ローカル仕様の一覧を使用します。 モックレジストリーを使用するには、レジストリーの名前を **mock** に設定します。

```
registry:
  - name: mock
    type: mock
```

6.3.4. Dockerhub レジストリー

dockerhub タイプを使用すると、DockerHub の特定の組織から APB を読み込むことができます (例: ansibleplaybookbundle)。

```
registry:
  - name: dockerhub
    type: dockerhub
    org: ansibleplaybookbundle
    user: <user>
    pass: <password>
    white_list:
      - ".*-apb$"
```

6.3.5. APB のフィルタリング

APB は、ブローカー設定内のレジストリーベースに設定された **white_list** または **black_list** パラメーターの組み合わせを使用して、イメージ名でフィルタリングできます。

これらはどちらもオプションの正規表現の一覧であり、特定のレジストリーで一致するものを判別できるように検出されたすべての APB に対して実行されます。

表6.2 APB フィルターの動作

存在するパラメーター	許可	ブロック
ホワイトリストのみ	一覧の正規表現に一致。	一致しないすべての APB。
ブラックリストのみ	一致しないすべての APB。	一覧の正規表現に一致する APB。
両方とも存在する	ホワイトリストの正規表現に一致し、ブラックリストの正規表現に一致しない。	ブラックリストの正規表現に一致する APB。
なし	レジストリーのどの APB も許可されない。	レジストリーのすべての APB。

以下は例を示しています。

ホワイトリストのみ

```
white_list:
  - "foo.*-apb$"
  - "^my-apb$"
```

この場合は、**foo.*-apb$** と **my-apb** に一致する APB が許可されます。それ以外の APB はすべて拒否されます。

ブラックリストのみ

```
black_list:
  - "bar.*-apb$"
  - "^foobar-apb$"
```

この場合は、**bar.*-apb$** と **foobar-apb** に一致する APB がブロックされます。それ以外の APB はすべて許可されます。

ホワイトリストとブラックリスト

```
white_list:
  - "foo.*-apb$"
  - "^my-apb$"
black_list:
  - "^foo-rootkit-apb$"
```

ここでは、**foo-rootkit-apb** はホワイトリストに一致するにもかかわらず、ブラックリストによって明確にブロックされます。これは、ホワイトリストの一致が上書きされるためです。

そうでない場合は、**foo.*-apb$** と **my-apb** に一致する APB のみが許可されます。

ブローカー設定の `registry` セクションのサンプル:

```
registry:
  - type: dockerhub
    name: dockerhub
    url: https://registry.hub.docker.com
    user: <user>
    pass: <password>
    org: <org>
    white_list:
      - "foo.*-apb$"
      - "^my-apb$"
    black_list:
      - "bar.*-apb$"
      - "^foobar-apb$"
```

6.3.6. ローカルの OpenShift Container レジストリー

`local_openshift` タイプを使用すると、OpenShift Container Platform クラスター内の OpenShift Container レジストリーから APB を読み込むことができます。公開された APB を検索する namespace を設定できます。

```
registry:
  - type: local_openshift
    name: lo
    namespaces:
      - openshift
    white_list:
      - ".*-apb$"
```

6.3.7. Red Hat Container Catalog レジストリー

`rhcc` タイプを使用すると、Red Hat Container Catalog (RHCC) レジストリーに公開された APB を読み込むことができます。

```
registry:
  - name: rhcc
    type: rhcc
    url: https://registry.access.redhat.com
    white_list:
      - ".*-apb$"
```

6.3.8. Red Hat Connect Partner Registry

Red Hat Container Catalog のサードパーティーのイメージは Red Hat Connect Partner レジストリー (https://registry.connect.redhat.com) から提供されます。**`partner_rhcc`** タイプは、APB の一覧を取得し、それらの仕様を読み込むためにブローカーが Partner レジストリーからブートストラップすることを可能にします。Partner レジストリーには、有効な Red Hat カスタマーポータルのユーザー名およびパスワードを使った、イメージをプルするための認証が必要です。

```
registry:
  - name: partner_reg
    type: partner_rhcc
```

```
    url:  https://registry.connect.redhat.com
    user: <registry_user>
    pass: <registry_password>
    white_list:
      - ".*-apb$"
```

Partner レジストリーには認証が必要なため、ブローカーを Partner レジストリー URL を使用するように設定するには以下の手動の手順も必要になります。

1. OpenShift Container Platform クラスターのすべてのノードで以下のコマンドを実行します。

```
# docker --config=/var/lib/origin/.docker \
    login -u <registry_user> -p <registry_password> \
    registry.connect.redhat.com
```

6.3.9. 複数のレジストリー

複数のレジストリーを使用して APB を論理的な組織に分割し、それらを同じブローカーから管理できます。レジスターには一意の空でない名前が必要です。一意の名前がない場合、サービスブローカーは起動に失敗し、問題について警告するエラーメッセージを表示します。

```
registry:
  - name: dockerhub
    type: dockerhub
    org: ansibleplaybookbundle
    user: <user>
    pass: <password>
    white_list:
      - ".*-apb$"
  - name: rhcc
    type: rhcc
    url: <rhcc_url>
    white_list:
      - ".*-apb$"
```

6.4. ブローカー認証

ブローカーは認証をサポートします。つまり、ブローカーに接続する際に、呼び出し側は各要求に対して Basic 認証または Bearer 認証の認証情報を指定する必要があります。**curl** を使用し、以下のように簡単に実行できます。

```
-u <user_name>:<password>
```

または、以下が表示されます。

```
-h "Authorization: bearer <token>
```

上記をコマンドに指定します。サービスカタログをユーザー名とパスワードの組み合わせ、またはベアラートークンが含まれるシークレットで設定する必要があります。

6.4.1. Basic 認証

Basic 認証の使用を有効にするには、ブローカー設定で以下を設定します。

```
broker:
   ...
   auth:
     - type: basic (1)
       enabled: true (2)
```

(1) **type** フィールドは使用する認証タイプを指定します。

(2) **enabled** フィールドでは、特定の認証タイプを無効にすることができます。これにより、これを無効にするために **auth** のセクション全体を削除する必要がなくなります。

6.4.1.1. デプロイメントテンプレートおよびシークレット

通常、ブローカーはデプロイメントテンプレートで ConfigMap を使用して設定されます。ファイル設定と同様の方法で認証設定を指定できます。

以下は、デプロイメントテンプレートのサンプルになります。

```
auth:
  - type: basic
    enabled: ${ENABLE_BASIC_AUTH}
```

Basic 認証の別の部分には、ブローカーに対して認証するために使用されるユーザー名とパスワードが含まれます。Basic 認証の実装は別のバックエンドサービスでサポートされる可能性があるものの、現時点でサポートされている実装は **シークレット** に対応します。シークレットは **/var/run/asb_auth** の場所にあるボリュームマウントで Pod に挿入される必要があります。これは、ブローカーがユーザー名とパスワードを読み取る場所です。

デプロイメントテンプレート では、シークレットが指定される必要があります。以下は例になります。

```
- apiVersion: v1
  kind: Secret
  metadata:
    name: asb-auth-secret
    namespace: openshift-ansible-service-broker
  data:
    username: ${BROKER_USER}
    password: ${BROKER_PASS}
```

シークレットにはユーザー名とパスワードが含まれる必要があります。値は base64 エンコードである必要があります。それらのエントリーの値を生成する最も簡単な方法として、**echo** および **base64** コマンドを使用できます。

```
$ echo -n admin | base64 (1)
YWRtaW4=
```

(1) **-n** オプションは非常に重要になります。

このシークレットはボリュームマウントで Pod に挿入される必要があります。これはデプロイメントテンプレートでも設定されます。

```
spec:
```

```
  serviceAccount: asb
  containers:
  - image: ${BROKER_IMAGE}
    name: asb
    imagePullPolicy: IfNotPresent
    volumeMounts:
      ...
      - name: asb-auth-volume
        mountPath: /var/run/asb-auth
```

次に **volumes** セクションで、シークレットをマウントします。

```
volumes:
  ...
  - name: asb-auth-volume
    secret:
      secretName: asb-auth-secret
```

上記により、**/var/run/asb-auth** にボリュームマウントが作成されます。このボリュームには、**asb-auth-secret** シークレットで作成されるユーザー名およびパスワードの 2 つのファイルがあります。

6.4.1.2. サービスカタログおよびブローカー通信の設定

ブローカーが Basic 認証を使用するように設定されているため、サービスカタログに対してブローカーとの通信方法について指示する必要があります。これは、ブローカーリソースの **authInfo** セクションで実行できます。

以下は、サービスカタログで **broker** リソースを作成する例になります。**spec** はサービスカタログに対し、ブローカーがリッスンしている URL を示唆します。**authInfo** は認証情報を取得するために読み取る必要のあるシークレットを示唆します。

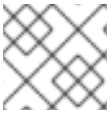
```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: Broker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb-1338-openshift-ansible-service-broker.172.17.0.1.nip.io
  authInfo:
    basicAuthSecret:
      namespace: openshift-ansible-service-broker
      name: asb-auth-secret
```

サービスカタログの v0.0.17 以降、ブローカーのリソース設定は変更されています。

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: ServiceBroker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb-1338-openshift-ansible-service-broker.172.17.0.1.nip.io
  authInfo:
    basic:
      secretRef:
        namespace: openshift-ansible-service-broker
        name: asb-auth-secret
```

6.4.2. Bearer 認証

デフォルトで、認証が指定されていない場合、ブローカーはベアラートークン認証 (Bearer Auth) を使用します。Bearer 認証は Kubernetes **apiserver** ライブラリーから委任された認証を使用します。



注記

Bearer 認証は OpenShift Container Platform 3.7 以降でのみ利用できます。

この設定は、Kubernetes RBAC ロールおよびロールバインディングにより、URL プレフィックスへのアクセスを付与します。ブローカーは設定オプション **cluster_url** を追加して **url_prefix** を指定します。この値はデフォルトで **openshift-ansible-service-broker** になります。

クラスターロールの例

```
- apiVersion: authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: access-asb-role
  rules:
  - nonResourceURLs: ["/ansible-service-broker", "/ansible-service-
broker/*"]
    verbs: ["get", "post", "put", "patch", "delete"]
```

6.4.2.1. デプロイメントテンプレートおよびシークレット

以下は、サービスカタログが使用できるシークレットの作成例です。この例では、ロールの **access-asb-role** がすでに作成されていることを前提としています。また、デプロイメントテンプレート が使用されています。

```
- apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: ansibleservicebroker-client
    namespace: openshift-ansible-service-broker

- apiVersion: authorization.openshift.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: ansibleservicebroker-client
  subjects:
  - kind: ServiceAccount
    name: ansibleservicebroker-client
    namespace: openshift-ansible-service-broker
  roleRef:
    kind: ClusterRole
    name: access-asb-role

- apiVersion: v1
  kind: Secret
  metadata:
    name: ansibleservicebroker-client
```

```
    annotations:
      kubernetes.io/service-account.name: ansibleservicebroker-client
  type: kubernetes.io/service-account-token
```

上記の例ではサービスアカウントを作成し、アクセスを **access-asb-role** に付与し、サービスアカウントトークンのシークレットを作成します。

6.4.2.2. サービスカタログおよびブローカー通信の設定

ブローカーが Bearer 認証トークンを使用するように設定されているため、サービスカタログに対してブローカーとの通信方法について指示する必要があります。これは、**broker** リソースの **authInfo** セクションで実行できます。

以下は、サービスカタログで **broker** リソースを作成する例になります。**spec** はサービスカタログに対し、ブローカーがリッスンしている URL を示唆します。**authInfo** は認証情報を取得するために読み取る必要のあるシークレットを示唆します。

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: ServiceBroker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb.openshift-ansible-service-
broker.svc:1338${BROKER_URL_PREFIX}/
  authInfo:
    bearer:
      secretRef:
        kind: Secret
        namespace: openshift-ansible-service-broker
        name: ansibleservicebroker-client
```

6.5. DAO 設定

フィールド	説明	必須
etcd_host	etcd ホストの URL です。	Y
etcd_port	**etcd_host** との通信時に使用するポートです。	Y

6.6. ログ設定

フィールド	説明	必須
logfile	ブローカーのログを書き込む場所です。	Y
stdout	ログを標準出力に書き込みます。	Y
level	ログ出力のレベルです。	Y

フィールド	説明	必須
color	ログに色付けします。	Y

6.7. OPENSHIFT 設定

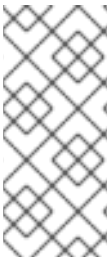
フィールド	説明	必須
host	OpenShift Container Platform ホストです。	N
ca_file	認証局ファイルの場所です。	N
bearer_token_file	使用するベアラートークンの場所です。	N
image_pull_policy	イメージをプルするタイミングです。	Y
namespace	ブローカーがデプロイされている namespace です。シークレットを介して渡されるパラメーター値などに重要です。	Y
sandbox_role	APB サンドボックス環境に対して指定するロールです。	Y
keep_namespace	APB の実行後に namespace を常に保持します。	N
keep_namespace_on_error	APB の実行でエラーが発生した後に namespace を保持します。	N

6.8. ブローカー設定

broker セクションでは、有効/無効にする機能をブローカーに指示します。また、完全な機能を有効にするファイルがディスク上のどこにあるかをブローカーに指示します。

フィールド	説明	デフォルト値	必須
dev_broker	開発ルートにアクセスできるようにします。	**false**	N
launch_apb_on_bind	バインドが no-op (無処理) になることを許可します。	**false**	N
bootstrap_on_startup	ブローカーが起動時に自らをブートストラップできるようにします。APB を設定済みのレジストリーから取得します。	**false**	N

フィールド	説明	デフォルト値	必須
recovery	etcd にある保留中のジョブを処理することによって、ブローカーが自らをリカバリーできるようにします。	**false**	N
output_request	デバッグを容易に行えるように、ブローカーが要求の受信時にそれをログファイルに出力できるようにします。	**false**	N
ssl_cert_key	TLS キーファイルがどこにあるかをブローカーに指示します。これが設定されない場合、API サーバーはキーファイルの作成を試みます。	**""**	N
ssl_cert	TLS **.crt** ファイルがどこにあるかをブローカーに指示します。これが設定されない場合、API サーバーはファイルの作成を試みます。	**""**	N
refresh_interval	レジストリーで新規イメージ仕様をクエリーする間隔です。	**"600s"**	N
auto_escalate	ブローカーが APB の実行中にユーザーのパーミッションをエスカレーションできるようにします。	**false**	N
cluster_url	ブローカーが予期する URL のプレフィックスを設定します。	**openshift-ansible-service-broker**	N



注記

非同期のバインドまたはバインド解除は実験的な機能であり、サポートされていないか、デフォルトでは有効になっていません。非同期バインドがない場合に **launch_apb_on_bind** を **true** に設定すると、バインドアクションがタイムアウトになり、再試行が実行されます。これはパラメーターの異なる同じバインド要求であるため、ブローカーは「409 Conflicts」で処理します。

6.9. シークレット設定

secrets セクションでは、ブローカーの namespace のシークレットとブローカーが実行する APB 間の関連付けを作成します。ブローカーは、これらのルールに従って実行中の APB にシークレットをマウントします。これにより、ユーザーはシークレットを使用して、パラメーターをカタログやユーザーに公開せずに渡すことができます。

このセクションは一覧の形式であり、各エントリーは以下の構造を持ちます。

フィールド	説明	必須
title	ルールのタイトルです。表示と出力の目的でのみ使用されます。	Y
apb_name	指定されたシークレットに関連付けられる APB の名前です。これは完全修飾名 (**<registry_name>-<image_name>**) です。	Y
secret	パラメーターをプルするシークレットの名前です。	Y

create_broker_secret.py ファイルをダウンロードし、これを使用して、この設定セクションの作成とフォーマットを行うことができます。

```
secrets:
- title: Database credentials
  secret: db_creds
  apb_name: dh-rhscl-postgresql-apb
```

6.10. プロキシー環境での実行

プロキシー化された OpenShift Container Platform クラスター内で OAB を実行する場合は、その中心的な概念を理解し、外部ネットワークアクセスに使用するプロキシーのコンテキストで検討することが重要です。

概要としては、ブローカー自体はクラスター内で Pod として実行され、そのレジスターの設定方法に応じて外部ネットワークにアクセスする必要があります。

6.10.1. レジストリーアダプターのホワイトリスト

ブローカーの設定済みレジストリーアダプターは、正常にブートストラップしてリモートの APB マニフェストを読み込むために、外部レジスターと通信できなければなりません。これらの要求はプロキシー経由で実行できますが、プロキシーでは必要なリモートホストにアクセスできるようにする必要があります。

必要なホワイトリスト化されたホストの例:

レジストリーアダプターのタイプ	ホワイトリスト化されたホスト
rhcc	**registry.access.redhat.com**、**access.redhat.com**
dockerhub	**docker.io**

6.10.2. Ansible を使用したプロキシー環境でのブローカーの設定

初期インストール時に OpenShift Container Platform クラスターがプロキシーの環境下で実行されるように設定した場合 (「グローバルプロキシーオプションの設定」を参照)、OAB はデプロイ時に以下を実行します。

- クラスター全体のプロキシー設定を自動的に継承する
- **cidr** フィールドおよび **serviceNetworkCIDR** を含む必要な **NO_PROXY** 一覧を生成する

それ以外の設定は必要ありません。

6.10.3. プロキシー環境でのブローカーの手動設定

クラスターのグローバルプロキシーオプションが初期インストール時またはブローカーのデプロイ前に設定されていない場合や、グローバルプロキシー設定を変更した場合、ブローカーの外部アクセスについてプロキシー経由で手動で設定する必要があります。

1. OAB をプロキシー環境で実行する前に「HTTP プロキシーの使用」を確認し、クラスターがプロキシー環境で実行されるように適切に設定されていることを確認してください。
   特に、クラスターは内部クラスター要求をプロキシー処理**しない**ように設定されている必要があります。通常、これは以下の **NO_PROXY** 設定を使用して設定されます。

   ```
   .cluster.local,.svc,<serviceNetworkCIDR_value>,<master_IP>,
   <master_domain>,.default
   ```

   その他の必要な **NO_PROXY** 設定も追加する必要があります。詳細については、「NO_PROXY の設定」を参照してください。

   注記

   バージョンなしまたは v1 の APB をデプロイするブローカーは、**172.30.0.1** も **NO_PROXY** の一覧に追加する**必要があります**。v2 より前の APB は、シークレットの交換ではなく、**exec** HTTP 要求を使用して実行中の APB Pod から認証情報を抽出しました。実験的なプロキシーサポートがあるブローカーを OpenShift Container Platform 3.9 より前のクラスターで実行していない限り、この点を心配する必要はおそらくありません。

2. ブローカーの **DeploymentConfig** を **cluster-admin** 権限を持つユーザーとして編集します。

   ```
   $ oc edit dc/asb -n openshift-ansible-service-broker
   ```

3. 以下の環境変数を設定します。
   - **HTTP_PROXY**
   - **HTTPS_PROXY**
   - **NO_PROXY**

   注記

   詳細については、「Pod でのプロキシー環境変数の設定」を参照してください。

4. 更新内容を保存した後、変更を有効にするために OAB のデプロイメント設定を再デプロイします。

   ```
   $ oc rollout latest dc/asb -n openshift-ansible-service-broker
   ```

6.10.4. Pod でのプロキシー環境変数の設定

一般に、APB Pod 自体もプロキシー経由の外部アクセスを必要とします。ブローカーは、自らにプロキシー設定があることを認識すると、生成する APB Pod にこれらの環境変数を透過的に適用します。APB 内で使用されるモジュールが環境変数経由でプロキシー設定に従う限り、APB もこれらの設定に基づいて動作します。

最後に、APB によって生成されたサービスもプロキシー経由の外部ネットワークアクセスを必要とする場合があります。APB は、それ自体の実行環境でこのようなサービスを検出した場合にこれらの環境変数を明示的に設定するように作成されている**必要があります**。そうでない場合には、クラスターオペレーターが必要なサービスを変更してそれらを環境に組み込む必要があります。

第7章 ホストの既存クラスターへの追加

7.1. ホストの追加

scaleup.yml Playbook を実行して新規ホストをクラスターに追加できます。この Playbook は、マスターをクエリーし、新規ホストの新規証明書を生成して配布し、これらの新規ホストでのみ、設定 Playbook を実行します。**scaleup.yml** Playbook を実行する前に、前提条件となる「ホストの準備」手順をすべて完了してください。

重要

scaleup.yml の Playbook は新規ホストの設定のみを実行します。マスターサービスの **NO_PROXY** の更新やマスターサービスの再起動は行いません。

scaleup.yml Playbook を実行するには、現在のクラスター設定を表す既存のインベントリーファイル (**/etc/ansible/hosts** など) が必要です。以前に **`atomic-openshift-installer`** コマンドを使用してインストールを実行した場合は、**~/.config/openshift/hosts** を調べて、インストーラーによって生成された最新のインベントリーファイルを見つけ、それをそのまま使用するか、必要に応じて、インベントリーファイルを変更することができます。その場合には、**`ansible-playbook`** を呼び出すときに、**`-i`** オプションを使用してそのファイルを指定する必要があります。

重要

推奨される最大ノード数については、「クラスターの制限」のセクションを参照してください。

手順

1. **atomic-openshift-utils** パッケージを更新して最新の Playbook を取得します。

```
# yum update atomic-openshift-utils
```

2. **/etc/ansible/hosts** ファイルを編集し、 **new_<host_type>** を **[OSEv3:children]** セクションに追加します。
たとえば、新規ノードホストを追加するには、**new_nodes** を追加します。

```
[OSEv3:children]
masters
nodes
new_nodes
```

新規マスターホストを追加するには、**new_masters** を追加します。

3. **[new_<host_type>]** セクションを作成して、新規ホストのホスト情報を指定します。このセクションは、以下の新規ノードの追加例で示されているように、既存のセクションと同じような形式で作成します。

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-
infra'
```

```
infra-node2.example.com openshift_node_group_name='node-config-
infra'

[new_nodes]
node3.example.com openshift_node_group_name='node-config-infra'
```

その他のオプションについては、「ホスト変数の設定」を参照してください。

新規マスターを追加する場合は、**[new_masters]** セクションと **[new_nodes]** セクションの両方に新規ホストを追加して、新規マスターホストが OpenShift SDN の一部となるようにします。

```
[masters]
master[1:2].example.com

[new_masters]
master3.example.com

[nodes]
master[1:2].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-
infra'
infra-node2.example.com openshift_node_group_name='node-config-
infra'

[new_nodes]
master3.example.com
```

重要

マスターホストに **node-role.kubernetes.io/infra=true** ラベルを付け、それ以外に専用インフラストラクチャーノードがない場合は、エントリーに **openshift_schedulable=true** を追加してホストにスケジュール可能であることを示すマークを明示的に付ける必要もあります。そうしないと、レジストリー Pod とルーター Pod をどこにも配置できなくなります。

4. **scaleup.yml** Playbook を実行します。インベントリーファイルがデフォルトの **/etc/ansible/hosts** 以外の場所にある場合は、**-i** オプションで場所を指定します。

   - ノードを追加する場合は、以下を指定します。

     ```
     # ansible-playbook [-i /path/to/file] \
         /usr/share/ansible/openshift-ansible/playbooks/openshift-
     node/scaleup.yml
     ```

   - マスターを追加する場合は、以下を実行します。

     ```
     # ansible-playbook [-i /path/to/file] \
         /usr/share/ansible/openshift-ansible/playbooks/openshift-
     master/scaleup.yml
     ```

5. Playbook の実行後に、インストールの検証を行います。

6. **[new_<host_type>]** セクションで定義したホストを適切なセクションに移動します。このようにホストを移動することで、このインベントリーファイルを使用するその後の Playbook の実行で、正しくノードが処理されるようになります。**[new_<host_type>]** セクションは空のままで結構です。たとえば、新規ノードを追加する場合は以下のように指定します。

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
node3.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-
infra'
infra-node2.example.com openshift_node_group_name='node-config-
infra'

[new_nodes]
```

7.2. ETCD ホストの既存クラスターへの追加

etcd scaleup Playbook を実行して新規 etcd ホストを追加することができます。この Playbook は、マスターをクエリーし、新規ホストの新規証明書を生成してこれを配布し、設定 Playbook を新規ホストにのみ実行します。etcd **scaleup.yml** Playbook を実行する前に、前提条件となる「ホストの準備」の手順をすべて完了してください。

etcd ホストを既存クラスターに追加するには、以下を実行します。

1. **atomic-openshift-utils** パッケージを更新して最新の Playbook を取得します。

```
$ yum update atomic-openshift-utils
```

2. **/etc/ansible/hosts** ファイルを編集し、**new_<host_type>** を **[OSEv3:children]** グループに、ホストを **new_<host_type>** グループに追加します。
たとえば、新規 etcd を追加するには、**new_etcd** を追加します。

```
[OSEv3:children]
masters
nodes
etcd
new_etcd

[etcd]
etcd1.example.com
etcd2.example.com

[new_etcd]
etcd3.example.com
```

3. etcd **scaleup.yml** Playbook を実行します。インベントリーファイルがデフォルトの **/etc/ansible/hosts** 以外の場所にある場合は、**-i** オプションで場所を指定します。

```
$ ansible-playbook [-i /path/to/file] \
  /usr/share/ansible/openshift-ansible/playbooks/openshift-
etcd/scaleup.yml
```

4. Playbook が正常に完了したら、インストールの検証を行います。

7.3. 共存する ETCD での既存のマスターの置き換え

マシンを別のデータセンターに移行し、割り当てられているネットワークと IP が変更される場合には、以下の手順を実行します。

1. プライマリー「etcd」と「マスター」ノードをバックアップします。

重要

「etcd バックアップ」の説明にあるように、**/etc/etcd/** ディレクトリーがバックアップされていることを確認します。

2. 置き換えるマスターの数だけ、新規マシンをプロビジョニングします。

3. クラスターを追加または拡張します。たとえば、etcd が共存するマスターを 3 つ追加する場合には、マスターノード 3 つまたは etcd ノード 3 つに拡張します。

   a. マスターを追加します。このプロセスのステップ 3 で、**[new_masters]** と **[new_nodes]** に新規データセンターのホストを追加して、マスターの **scaleup.yml** Playbook を実行します。

   b. 同じホストを「etcd」セクションに配置して、etcd **scaleup.yml** Playbook を実行します。

   c. ホストが追加されたことを確認します。

```
# oc get nodes
```

   d. マスターホストの IP が追加されたことを確認します。

```
# oc get ep kubernetes
```

   e. etcd が追加されたことを確認します。**ETCDCTL_API** の値は、使用するバージョンにより異なります。

```
# source /etc/etcd/etcd.conf
# ETCDCTL_API=2 etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-
file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URLS
member list
```

   f. **/etc/origin/master** ディレクトリーから、インベントリーファイルの最初に記載されている新規マスターホストに、**/etc/origin/master/ca.serial.txt** をコピーします。デフォルトでは、これは **/etc/ansible/hosts** です。

4. etcd ホストを削除します。

   a. **/etc/etcd/ca** ディレクトリーを、インベントリーファイルの最初に記載されている新規 etcd ホストにコピーします。デフォルトでは、これは **/etc/ansible/hosts** です。

   b. **master-config.yaml** ファイルから以前の etcd クライアントを削除します。

```
# grep etcdClientInfo -A 11 /etc/origin/master/master-config.yaml
```

c. マスターを再起動します。

```
# master-restart api
# master-restart controllers
```

d. クラスターから以前の etcd メンバーを削除します。**ETCDCTL_API** の値は、使用するバージョンにより異なります。

```
# source /etc/etcd/etcd.conf
# ETCDCTL_API=2 etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-
file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URLS
member list
```

e. 上記のコマンドの出力から ID を取得して、この ID で以前のメンバーを削除します。

```
# etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-
file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URL
member remove 1609b5a3a078c227
```

f. 以前の etcd ホストで etcd サービスを停止して、無効化します。

```
# systemctl stop etcd
# systemctl disable etcd
```

5. 以前のマスター API とコントローラーサービスをシャットダウンします。

```
# systemctl stop atomic-openshift-master-api
```

6. ネイティブのインストールプロセス時に、デフォルトでロードバランサーとしてインストールされていたマスターノードを、HA プロキシー設定から削除します。

7. マシンの使用を停止します。

a. 削除するマスター上の **atomic-openshift-node** サービスを停止します。

```
# systemctl stop atomic-openshift-node
```

b. ノードリソースを削除します。

```
# oc delete node
```

7.4. ノードの移行

ノード上のサービスの実行方法やスケーリング方法、慣れた方法により、ノードを個別での移行や、グループ (2、5、10 ずつなど) 単位での移行が可能です。

1. 移行するノードについては、新規データセンターのノードで使用するために新規の仮想マシンをプロビジョニングします。

2. 新規ノードを追加するには、インフラストラクチャーを拡張します。新規ノードのラベルが適切に設定されており、新規 API サーバーがロードバランサーに追加され、トラフィックを適切に送信していることを確認します。

3. 評価および縮小を実行します。

    a. 現在のノード (古いデータセンター内にある) に「スケジュール対象外」のマークを付けます。

    b. ノードの退避を実行し、ノード上の Pod が他のノードにスケジュールされるようにします。

    c. 退避したサービスが新規ノードで実行されていることを確認します。

4. ノードを削除します。

    a. ノードが空であり、実行中のプロセスがないことを確認します。

    b. サービスを停止するか、またはノードを削除します。

第8章 デフォルトのイメージストリームとテンプレートの読み込み

8.1. 概要

OpenShift Container Platform インストールには、Red Hat が提供するイメージストリームおよびテンプレートの便利なセットが含まれています。これにより、開発者は新規アプリケーションを簡単に作成できます。デフォルトで、クラスターインストールプロセスは、すべてのユーザーが表示アクセスを持つデフォルトのグローバルプロジェクトである **openshift** プロジェクトにこれらのセットを自動的に作成します。

8.2. サブスクリプションタイプ別のサービス

お使いの Red Hat アカウントのアクティブなサブスクリプションに応じて、以下のイメージストリームとテンプレートのセットが Red Hat によって提供され、サポートされます。サブスクリプションの詳細については、Red Hat の営業担当者にお問い合わせください。

8.2.1. OpenShift Container Platform サブスクリプション

アクティブな **OpenShift Container Platform サブスクリプション** により、イメージストリームとテンプレートのコアのセットが提供され、サポートされます。これには以下のテクノロジーが含まれます。

種類	テクノロジー
言語とフレームワーク	• .NET Core • Node.js • Perl • PHP • Python • Ruby
データベース	• MariaDB • MongoDB • MySQL • PostgreSQL
ミドルウェアサービス	• Red Hat JBoss Web Server (Tomcat) • Red Hat Single Sign-on

種類	テクノロジー
他のサービス	• Jenkins • Jenkins Slaves

8.2.2. xPaaS ミドルウェアアドオンサブスクリプション

xPaaS ミドルウェアイメージのサポートは、**xPaaS ミドルウエアアドオンサブスクリプション** で提供されます。これらのサブスクリプションは、xPaaS 製品ごとに独立したサブスクリプションとなっています。お使いのアカウントで該当するサブスクリプションがアクティブになっている場合は、以下のテクノロジーのイメージストリームとテンプレートが提供され、サポートされます。

種類	テクノロジー
ミドルウェアサービス	• Red Hat JBoss A-MQ • Red Hat JBoss BPM Suite Intelligent Process Server • Red Hat JBoss BRMS Decision Server • Red Hat JBoss Data Grid • Red Hat JBoss EAP • Red Hat JBoss Fuse Integration Services • Red Hat JBoss Data Virtualization

8.3. 作業を開始する前に

このトピックのタスクの実行を検討する前に、以下のいずれかを実行してこれらのイメージストリームとテンプレートが OpenShift Container Platform クラスターにすでに登録されているかどうかを確認してください。

- Web コンソールにログインして **Add to Project** をクリックします。
- CLI を使用して **openshift** プロジェクト用のイメージストリームとテンプレートの一覧を表示します。

```
$ oc get is -n openshift
$ oc get templates -n openshift
```

デフォルトのイメージストリームとテンプレートが削除または変更されている場合は、このトピックに従ってデフォルトのオブジェクトを各自で作成できます。そうしない場合は、以下の指示に従う必要はありません。

8.4. 前提条件

デフォルトのイメージストリームとテンプレートを作成する前に、以下を確認してください。

- 統合 Docker レジストリーサービスが OpenShift Container Platform インストールにデプロイされている必要があります。
- **oc create** コマンドを **cluster-admin** 権限で実行できる必要があります。デフォルトの **openshift** プロジェクト で動作できるようにするためです。
- **atomic-openshift-utils** RPM パッケージがインストールされている必要があります。手順については、「ソフトウェアの前提条件」を参照してください。
- イメージストリームとテンプレートが含まれているディレクトリーのシェル変数を定義します。これにより、以降のセクションで使用するコマンドが大幅に短くなります。これを実行するには、以下のように入力します。

```
$ IMAGESTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/image-
streams"; \
    XPAASSTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/xpaas-
streams"; \
    XPAASTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/xpaas-
templates"; \
    DBTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/db-templates";
\
    QSTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v3.10/quickstart-
templates"
```

8.5. OPENSHIFT CONTAINER PLATFORM イメージのイメージストリームの作成

Red Hat サブスクリプションマネージャーで、ノードのホストをサブスクライブしており、Red Hat Enterprise Linux (RHEL) 7 ベースのイメージを使用したイメージストリームのコアセットを使用する場合には、以下を実行します。

```
$ oc create -f $IMAGESTREAMDIR/image-streams-rhel7.json -n openshift
```

または、CentOS 7 ベースのイメージを使用するイメージストリームのコアセットを作成するには、以下を実行します。

```
$ oc create -f $IMAGESTREAMDIR/image-streams-centos7.json -n openshift
```

CentOS と RHEL の両方のイメージストリームセットは同じ名前なので、両方を作成することはできません。両方のイメージストリームセットをユーザーが使用できるようにするには、一方のセットを別のプロジェクトに作成するか、いずれかのファイルを編集し、イメージストリームの名前を一意の名前に変更します。

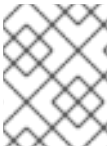
8.6. XPAAS ミドルウェアイメージのイメージストリームの作成

xPaaS ミドルウェアイメージストリームは、**JBoss EAP**、**JBoss JWS**、**JBoss A-MQ**、**JBoss Fuse Integration Services**、**Decision Server**、**JBoss Data Virtualization**、および **JBoss Data Grid** のイメージを提供します。それらのイメージは、提供されるテンプレートを使用してこれらのプラット

フォームのアプリケーションを作成するために使用できます。

xPaaS ミドルウェアイメージストリームセットを作成するには、以下を実行します。

```
$ oc create -f $XPAASSTREAMDIR/jboss-image-streams.json -n openshift
```



注記

これらのイメージストリームによって参照されるイメージにアクセスするには、該当する xPaaS ミドルウェアサブスクリプションが必要です。

8.7. データベースサービステンプレートの作成

データベースサービステンプレートを使用すると、他のコンポーネントで利用できるデータベースイメージを簡単に実行できます。データベース (**MongoDB**、**MySQL**、および **PostgreSQL**) ごとに、2 つのテンプレートが定義されています。

1 つのテンプレートはコンテナー内の一時ストレージを使用します。つまり、保存データは Pod の移動などによってコンテナーが再起動されると失われます。このテンプレートは、デモ目的にのみ使用してください。

もう 1 つのテンプレートは永続ボリュームをストレージに使用しますが、OpenShift Container Platform インストールに永続ボリュームが設定されている必要があります。

データベーステンプレートのコアセットを作成するには、以下を実行します。

```
$ oc create -f $DBTEMPLATES -n openshift
```

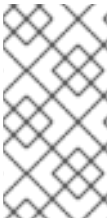
テンプレートを作成したら、ユーザーは各種のテンプレートを簡単にインスタンス化し、データベースデプロイメントにすばやくアクセスできるようになります。

8.8. インスタントアプリケーションおよびクイックスタートテンプレートの作成

インスタントアプリケーションおよびクイックスタートテンプレートでは、実行中のアプリケーションのオブジェクトの完全なセットを定義します。これには以下が含まれます。

- GitHub パブリックリポジトリーにあるソースからアプリケーションをビルドするためのビルド設定
- ビルド後にアプリケーションイメージをデプロイするためのデプロイメント設定
- アプリケーション Pod に負荷分散を提供するサービス
- アプリケーションに外部アクセスを提供するルート

いくつかのテンプレートでは、アプリケーションがデータベース操作を実行できるように、データベースデプロイメントとサービスも定義します。



注記

データベースを定義するテンプレートでは、一時ストレージを使用してデータベースコンテンツを格納します。何らかの理由でデータベース Pod が再起動すると、すべてのデータベースデータが失われるため、これらのテンプレートはデモ目的にのみ使用してください。

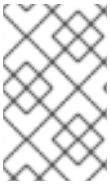
これらのテンプレートを使用すると、ユーザーは、OpenShift Container Platform で提供される各種の言語イメージを使用する完全なアプリケーションを簡単にインスタンス化できます。また、インストール時にテンプレートのパラメーターをカスタマイズし、サンプルリポジトリーではなく独自のリポジトリーからソースがビルドされるようにできます。つまり、これは新規アプリケーションのビルドの単純な開始点となります。

コアのインスタントアプリケーションおよびクイックスタートテンプレートを作成するには、以下を実行します。

```
$ oc create -f $QSTEMPLATES -n openshift
```

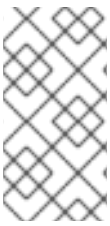
各種の xPaaS ミドルウェア製品 (**JBoss EAP**、**JBoss JWS**、**JBoss A-MQ**、**JBoss Fuse Integration Services**、**Decision Server**、および **JBoss Data Grid**) を使用するアプリケーションを作成するためのテンプレートのセットも用意されています。このテンプレートセットを登録するには、以下を実行します。

```
$ oc create -f $XPAASTEMPLATES -n openshift
```



注記

xPaaS ミドルウェアテンプレートには、xPaaS ミドルウェアイメージストリームが必要です。さらに、xPaaS ミドルウェアイメージストリームには、該当する xPaaS ミドルウェアサブスクリプションが必要です。



注記

データベースを定義するテンプレートでは、一時ストレージを使用してデータベースコンテンツを格納します。何らかの理由でデータベース Pod が再起動すると、すべてのデータベースデータが失われるため、これらのテンプレートはデモ目的にのみ使用してください。

8.9. 次のステップ

これらのアーティファクトを作成したら、開発者は「Web コンソール」にログインし、「テンプレートからの作成」フローを実行できるようになります。任意のデータベースまたはアプリケーションテンプレートを選択し、現在のプロジェクトで実行するデータベースサービスまたはアプリケーションを作成できます。一部のアプリケーションテンプレートでは独自のデータベースサービスも定義することに注意してください。

サンプルアプリケーションはすべて、**SOURCE_REPOSITORY_URL** パラメーター値が示すように、テンプレートのデフォルトの参照先である GitHub リポジトリーからビルドされます。これらのリポジトリーはフォークすることができ、テンプレートから作成する際にフォークを **SOURCE_REPOSITORY_URL** パラメーター値として指定できます。これにより、開発者は独自のアプリケーションの作成を試行することができます。

開発者は、開発者ガイドの「インスタントアプリおよびクイックスタートテンプレートの使用」セクションでこれらの手順を確認できます。

第9章 カスタム証明書の設定

9.1. 概要

管理者は、OpenShift Container Platform API のパブリックホスト名および Web コンソール用のカスタム提供証明書を設定できます。この設定は、クラスターインストールの実行時か、またはインストール後に行うことができます。

9.2. インストール時のカスタム証明書の設定

クラスターインストールの実行時に、カスタム証明書はインベントリーファイルで設定可能な **openshift_master_named_certificates** パラメーターと **openshift_master_overwrite_named_certificates** パラメーターを使用して設定できます。詳細については、「Ansible を使用したカスタム証明書の設定」を参照してください。

カスタム証明書の設定パラメーター

```
openshift_master_overwrite_named_certificates=true (1)
openshift_master_named_certificates=[{"certfile": "/path/on/host/to/crt-
file", "keyfile": "/path/on/host/to/key-file", "names": ["public-master-
host.com"], "cafile": "/path/on/host/to/ca-file"}] (2)
openshift_hosted_router_certificate={"certfile": "/path/on/host/to/app-
crt-file", "keyfile": "/path/on/host/to/app-key-file", "cafile":
"/path/on/host/to/app-ca-file"} (3)
```

(1) **openshift_master_named_certificates** パラメーターの値を指定した場合は、このパラメーターを **true** に設定します。

(2) マスター API 証明書をプロビジョニングします。

(3) ワイルドカード API 証明書をプロビジョニングします。

以下は、マスター API 証明書のパラメーターの例です。

```
openshift_master_overwrite_named_certificates=true
openshift_master_named_certificates=[{"names":
["master.148.251.233.173.nip.io"], "certfile": "/home/cloud-user/master-
bundle.cert.pem", "keyfile": "/home/cloud-
user/master.148.251.233.173.nip.io.key.pem" ]
```

以下は、ワイルドカード API 証明書のパラメーターの例です。

```
openshift_hosted_router_certificate={"certfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/ca-
chain.cert.pem"}
```

9.3. WEB コンソールまたは CLI 用カスタム証明書の設定

Web コンソールおよび CLI 用のカスタム証明書は、「マスター設定ファイル」の **servingInfo** セクションで指定できます。

- **servingInfo.namedCertificates** セクションでは、Web コンソール用のカスタム証明書を指定します。
- **servingInfo** セクションでは、CLI およびその他の API 呼び出し用のカスタム証明書を指定します。

この方法で複数の証明書を設定し、それぞれの証明書を複数のホスト名、複数のルーター、または OpenShift Container Platform イメージレジストリーに関連付けることができます。

デフォルトの証明書は、**namedCertificates** のほかにも **servingInfo.certFile** および **servingInfo.keyFile** 設定セクションに設定する必要があります。

注記

namedCertificates セクションは、**/etc/origin/master/master-config.yaml** ファイルの **masterPublicURL** および **oauthConfig.assetPublicURL** 設定に関連付けられたホスト名についてのみ設定する必要があります。**masterURL** に関連付けられたホスト名にカスタム提供証明書を使用すると、インフラストラクチャーコンポーネントが内部の **masterURL** ホストを使用してマスター API と通信しようとするため、TLS エラーが発生します。

カスタム証明書の設定

```
servingInfo:
  logoutURL: ""
  masterPublicURL: https://openshift.example.com:8443
  publicURL: https://openshift.example.com:8443/console/
  bindAddress: 0.0.0.0:8443
  bindNetwork: tcp4
  certFile: master.server.crt (1)
  clientCA: ""
  keyFile: master.server.key (2)
  maxRequestsInFlight: 0
  requestTimeoutSeconds: 0
  namedCertificates:
    - certFile: wildcard.example.com.crt (3)
      keyFile: wildcard.example.com.key (4)
      names:
        - "openshift.example.com"
  metricsPublicURL: "https://metrics.os.example.com/hawkular/metrics"
```

(1) (2) CLI およびその他の API 呼び出し用の証明書とキーファイルへのパス。

(3) (4) Web コンソール用の証明書とキーファイルへのパス。

Ansible インベントリーファイル (デフォルトでは **/etc/ansible/hosts**) の **openshift_master_cluster_public_hostname** パラメーターと **openshift_master_cluster_hostname** パラメーターは異なっていなければなりません。これらが同じであると、名前付き証明書が失敗し、証明書の再インストールが必要になります。

```
# Native HA with External LB VIPs
openshift_master_cluster_hostname=internal.paas.example.com
openshift_master_cluster_public_hostname=external.paas.example.com
```

OpenShift Container Platform で DNS を使用する方法の詳細については、「DNS のインストールの前提条件」を参照してください。

この方法では、OpenShift Container Platformによって生成される自己署名証明書を利用して、必要に応じて信頼できるカスタム証明書を個々のコンポーネントに追加できます。

内部インフラストラクチャーの証明書は自己署名のままであることに注意してください。これは一部のセキュリティーチームや PKI チームから不適切な使用法と見なされる場合があります。ただし、これらの証明書を信頼するクライアントはクラスター内のその他のコンポーネントだけであるため、これに伴うリスクは最小限です。外部のユーザーとシステムはすべて、信頼できるカスタム証明書を使用します。

相対パスは、マスター設定ファイルの場所に基づいて解決されます。設定の変更が反映されるようにサーバーを再起動します。

9.4. カスタムマスターホスト証明書の設定

OpenShift Container Platform の外部ユーザーとの信頼できる接続を容易にするために、Ansible インベントリーファイル (デフォルトでは **/etc/ansible/hosts**) の **openshift_master_cluster_public_hostname** パラメーターで指定されたドメイン名に一致する名前付き証明書をプロビジョニングできます。

この証明書は Ansible がアクセスできるディレクトリーに置き、以下のように Ansible インベントリーファイルにパスを追加する必要があります。

```
openshift_master_named_certificates=[{"certfile": "/path/to/console.ocp-
c1.myorg.com.crt", "keyfile": "/path/to/console.ocp-c1.myorg.com.key",
"names": ["console.ocp-c1.myorg.com"]}]
```

パラメーター値は以下のようになります。

- **certfile** は、OpenShift Container Platform ルーター証明書を含むファイルへのパスです。
- **keyfile** は、OpenShift Container Platform ワイルドカードキーを含むファイルへのパスです。
- **names** は、クラスターのパブリックホスト名です。

ファイルパスは、Ansible が実行されるシステムにとってローカルでなければなりません。証明書はマスターホストにコピーされ、**/etc/origin/master** ディレクトリー内にデプロイされます。

レジストリーのセキュリティーを保護する場合、サービスのホスト名と IP アドレスをレジストリーのサーバー証明書に追加します。SAN (Subject Alternative Name) には以下の情報が含まれている必要があります。

- 2 つのサービスホスト名。

  ```
  docker-registry.default.svc.cluster.local
  docker-registry.default.svc
  ```

- サービス IP アドレス。
  以下は例を示しています。

  ```
  172.30.252.46
  ```

以下のコマンドを使って Docker レジストリーのサービス IP アドレスを取得します。

```
oc get service docker-registry --template='{{.spec.clusterIP}}'
```

- パブリックホスト名。

```
docker-registry-default.apps.example.com
```

以下のコマンドを使って Docker レジストリーのパブリックホスト名を取得します。

```
oc get route docker-registry --template '{{.spec.host}}'
```

たとえば、サーバー証明書には以下のような SAN の詳細が記載されるはずです。

```
X509v3 Subject Alternative Name:
                DNS:docker-registry-public.openshift.com, DNS:docker-
registry.default.svc, DNS:docker-registry.default.svc.cluster.local,
DNS:172.30.2.98, IP Address:172.30.2.98
```

9.5. デフォルトルーター用のカスタムワイルドカード証明書の設定

OpenShift Container Platform のデフォルトルーターをデフォルトのワイルドカード証明書を使って設定できます。デフォルトのワイルドカード証明書を使用すると、OpenShift Container Platform にデプロイされているアプリケーションでカスタム証明書を使用せずにデフォルトの暗号化を簡単に使用することができます。

注記

デフォルトのワイルドカード証明書の使用は、非実稼働環境でのみ推奨されます。

デフォルトのワイルドカード証明書を設定するには、**.<app_domain>** で有効な証明書をプロビジョニングします。ここで、**<app_domain>** は、Ansible インベントリーファイル(デフォルトでは **/etc/ansible/hosts**) の **openshift_master_default_subdomain** の値です。プロビジョニングが完了したら、証明書、キー、CA 証明書ファイルを Ansible ホストに置き、以下の行を Ansible インベントリーファイルに追加します。

```
openshift_hosted_router_certificate={"certfile": "/path/to/apps.c1-
ocp.myorg.com.crt", "keyfile": "/path/to/apps.c1-ocp.myorg.com.key",
"cafile": "/path/to/apps.c1-ocp.myorg.com.ca.crt"}
```

以下は例を示しています。

```
openshift_hosted_router_certificate={"certfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/ca-
chain.cert.pem"}
```

パラメーター値は以下のようになります。

- **certfile** は、OpenShift Container Platform ルーター証明書を含むファイルへのパスです。
- **keyfile** は、OpenShift Container Platform ワイルドカードキーを含むファイルへのパスです。

- **cafile** は、このキーと証明書のルート CA を含むファイルへのパスです。中間 CA を使用している場合は、中間 CA とルート CA の両方がこのファイルに含まれている必要があります。

これらの証明書ファイルを OpenShift Container Platform クラスターで初めて使用する場合は、Ansible **deploy_router.yml** Playbook を実行し、これらのファイルを OpenShift Container Platform 設定ファイルに追加します。この Playbook は、証明書ファイルを **/etc/origin/master/** ディレクトリーに追加します。

```
# ansible-playbook [-i /path/to/inventory] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
hosted/deploy_router.yml
```

これらの証明書を使用するのが初めてでない場合、たとえば、既存の証明書を変更するか、期限切れの証明書を置き換える場合は、以下の Playbook を実行します。

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/redeploy-
certificates.yml
```

注記

この Playbook を実行する場合は、証明書名を変更しないでください。証明書名を変更した場合は、新規証明書の場合と同様に Ansible **deploy_cluster.yml** Playbook を再実行してください。

9.6. イメージレジストリー用のカスタム証明書の設定

OpenShift Container Platform イメージレジストリーは、ビルドとデプロイメントを容易にする内部サービスです。レジストリーとの通信の大部分は、OpenShift Container Platform の内部コンポーネントによって処理されます。そのため、レジストリーサービス自体が使用する証明書を置き換える必要はありません。

ただし、デフォルトでは、レジストリーは、外部のシステムとユーザーがイメージのプルとプッシュを実行できるルートを使用します。内部の自己署名証明書を使用する代わりに、外部ユーザーに提供されるカスタム証明書を使用して**re-encrypt ルート**を使用することができます。

これを設定するには、コードの以下の行を、Ansible インベントリーファイル (デフォルトは **/etc/ansible/hosts** ファイルの **[OSEv3:vars]** セクションに追加します。レジストリールートで使用する証明書を指定します。

```
openshift_hosted_registry_routehost=registry.apps.c1-ocp.myorg.com 1
openshift_hosted_registry_routecertificates={"certfile":
"/path/to/registry.apps.c1-ocp.myorg.com.crt", "keyfile":
"/path/to/registry.apps.c1-ocp.myorg.com.key", "cafile":
"/path/to/registry.apps.c1-ocp.myorg.com-ca.crt"} 2
openshift_hosted_registry_routetermination=reencrypt 3
```

1. レジストリーのホスト名です。
2. **cacert**、**root**、および **cafile** ファイルの場所です。
   - **certfile** は、OpenShift Container Platform ルーター証明書を含むファイルへのパスです。
   - **keyfile** は、OpenShift Container Platform ワイルドカードキーを含むファイルへのパスです。

- **cafile** は、このキーと証明書のルート CA を含むファイルへのパスです。中間 CA を使用している場合は、中間 CA とルート CA の両方がこのファイルに含まれている必要があります。

3 暗号化を実行する場所を指定します。

- edge ルーターで暗号化を終了し、送信先から提供される新規の証明書で再暗号化するには、 **reencrypt** に設定し、**re-encrypt ルート**を指定します。
- 送信先で暗号化を終了するには、**passthrough** に設定します。トラフィックは送信先で復号化されます。

9.7. ロードバランサー用のカスタム証明書の設定

OpenShift Container Platform クラスターでデフォルトのロードバランサーか、またはエンタープライズレベルのロードバランサーを使用している場合、カスタム証明書の使用により、 パブリックに署名されたカスタム証明書を使って Web コンソールと API を外部で利用できるようにし、既存の内部証明書を内部のエンドポイント用に残しておくことができます。

カスタム証明書をこの方法で使用するように OpenShift Container Platform を設定するには、以下を実行します。

1. マスター設定ファイルの **servingInfo** セクションを編集します。

```
servingInfo:
  logoutURL: ""
  masterPublicURL: https://openshift.example.com:8443
  publicURL: https://openshift.example.com:8443/console/
  bindAddress: 0.0.0.0:8443
  bindNetwork: tcp4
  certFile: master.server.crt
  clientCA: ""
  keyFile: master.server.key
  maxRequestsInFlight: 0
  requestTimeoutSeconds: 0
  namedCertificates:
    - certFile: wildcard.example.com.crt 1
      keyFile: wildcard.example.com.key 2
      names:
        - "openshift.example.com"
  metricsPublicURL:
"https://metrics.os.example.com/hawkular/metrics"
```

1 Web コンソールの証明書ファイルへのパスです。

2 Web コンソールのキーファイルへのパスです。

注記

masterPublicURL および **oauthConfig.assetPublicURL** 設定に関連付けられたホスト名についてのみ **namedCertificates** セクションを設定します。**masterURL** に関連付けられたホスト名にカスタム提供証明書を使用すると、インフラストラクチャーコンポーネントが内部の masterURL ホストを使用してマスター API と通信しようとするため、TLS エラーが発生します。

2. Ansible インベントリーファイル (デフォルトでは **/etc/ansible/hosts**) で **openshift_master_cluster_public_hostname** パラメーターと **openshift_master_cluster_hostname** パラメーターを指定します。これらの値は異なっていなければなりません。同じ値を指定した場合、名前付き証明書が失敗します。

```
# Native HA with External LB VIPs
openshift_master_cluster_hostname=paas.example.com 1
openshift_master_cluster_public_hostname=public.paas.example.com 2
```

1 SSL パススルー用に設定された内部ロードバランサーの FQDN です。

2 カスタム (パブリック) 証明書を使用する外部ロードバランサーの FQDN です。

お使いのロードバランサー環境に固有の情報については、お使いのプロバイダーについての OpenShift Container Platform リファレンスアーキテクチャーと「Custom Certificate SSL Termination (Production)」を参照してください。

9.8. カスタム証明書の変更およびクラスターへの組み込み

カスタムマスター証明書とカスタムルーター証明書を変更して、既存の OpenShift Container Platform クラスターに組み込むことができます。これを行うには、Ansible インベントリーファイル (デフォルトでは **/etc/ansible/hosts**) を編集し、Playbook を実行します。

9.8.1. カスタムマスター証明書のの変更およびクラスターへの組み込み

カスタム証明書を変更するには、以下を実行します。

1. Ansible インベントリーファイルを編集して **openshift_master_overwrite_named_certificates=true** を設定します。

2. **openshift_master_named_certificates** パラメーターを使用して証明書へのパスを指定します。

```
openshift_master_overwrite_named_certificates=true
openshift_master_named_certificates=[{"certfile":
"/path/on/host/to/crt-file", "keyfile": "/path/on/host/to/key-file",
"names": ["public-master-host.com"], "cafile": "/path/on/host/to/ca-
file"}] 1
```

1 マスター API 証明書へのパスです。

3. 以下の Playbook を実行します。

```
ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/redeploy-certificates.yml
```

9.8.2. カスタムルーター証明書の変更およびクラスターへの組み込み

カスタムルーター証明書を変更し、これを組み込むには、以下を実行します。

1. Ansible インベントリーファイルを編集して **openshift_master_overwrite_named_certificates=true** を設定します。
2. **openshift_hosted_router_certificate** パラメーターを使用して証明書へのパスを指定します。

```
openshift_master_overwrite_named_certificates=true
openshift_hosted_router_certificate={"certfile":
"/path/on/host/to/app-crt-file", "keyfile": "/path/on/host/to/app-
key-file", "cafile": "/path/on/host/to/app-ca-file"} (1)
```

(1) ワイルドカード API 証明書へのパスです。

3. 以下の Playbook を実行します。

```
ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/openshift-hosted/redeploy-router-certificates.yml
```

9.9. の他のコンポーネントでのカスタム証明書の使用

ロギングやメトリクスなどのその他のコンポーネントでカスタム証明書を使用する方法については、「Certificate Management」を参照してください。

第10章 証明書の再デプロイ

10.1. 概要

OpenShift Container Platform は、以下のコンポーネントで証明書を使用してセキュアな接続を提供します。

- マスター (API サーバーとコントローラー)
- etcd
- ノード
- レジストリー
- ルーター

インストーラーによって提供される Ansible Playbook を使用すると、クラスター証明書の有効期限を自動的にチェックできます。これらの証明書を自動的にバックアップしたり、再デプロイしたりするための Playbook も提供されます。これらの Playbook を使用すると、一般的な証明書エラーを修正できます。

証明書の再デプロイのユースケースとして以下が考えられます。

- インストーラーによって誤ったホスト名が検出され、そのことがすぐに判明しなかった。
- 証明書の有効期限が切れており、証明書を更新する必要がある。
- 新しい CA があり、その CA を代わりに使用する証明書を作成する。

10.2. 証明書の有効期限のチェック

インストーラーを使用して、設定可能な日数内に有効期限が切れる証明書に関する警告やすでに有効期限が切れた証明書に関する通知を受け取ることができます。証明書の有効期限切れ Playbook では、Ansible の **openshift_certificate_expiry** ロールが使用されます。

ロールによって検査される証明書には、以下が含まれます。

- マスターおよびノードサービス証明書
- etcd シークレットのルーターおよびレジストリーサービス証明書
- マスター、ノード、ルーター、レジストリー、および **cluster-admin** ユーザーの **kubeconfig** ファイル
- etcd 証明書 (組み込み証明書を含む)

10.2.1. ロール変数

openshift_certificate_expiry ロールは、以下の変数を使用します。

表10.1 コア変数

変数名	デフォルト値	説明
openshift_certificate_expiry_config_base	**/etc/origin**	OpenShift Container Platform 基本設定ディレクトリーです。
openshift_certificate_expiry_warning_days	**30**	現在を起点として指定された日数内に有効期限が切れる証明書にフラグを付けます。
openshift_certificate_expiry_show_all	**no**	正常な (期限切れや警告のない) 証明書を結果に組み込みます。

表10.2 オプションの変数

変数名	デフォルト値	説明
openshift_certificate_expiry_generate_html_report	**no**	有効期限切れのチェック結果に関する HTML レポートを生成します。
openshift_certificate_expiry_html_report_path	**/tmp/cert-expiry-report.html**	HTML レポートを保存するための完全パスです。
openshift_certificate_expiry_save_json_results	**no**	有効期限のチェック結果を JSON ファイルとして保存します。
openshift_certificate_expiry_json_results_path	**/tmp/cert-expiry-report.json**	JSON レポートを保存するための完全パスです。

10.2.2. 証明書の有効期限切れ Playbook の実行

OpenShift Container Platform インストーラーは、**openshift_certificate_expiry** ロールのさまざまな設定セットを使用して証明書の有効期限切れ Playbook のサンプル一式を提供します。

これらの Playbook は、クラスターを表す インベントリーファイルと一緒に使用する必要があります。最適な結果を得るには、**ansible-playbook** に **-v** オプションを指定して実行します。

easy-mode.yaml のサンプル Playbook を使用すると、ロールアウトを仕様に合わせて調整する前に試すことができます。この Playbook は以下を実行します。

- JSON レポートと定型化された HTML レポートを **/tmp/** に生成します。
- ほぼ常に結果が得られるように警告期間を長い期間に設定します。
- すべての証明書 (正常であるかどうかを問わず) を結果に組み込みます。

easy-mode.yaml Playbook

```
- name: Check cert expirys
```

```
  hosts: nodes:masters:etcd
  become: yes
  gather_facts: no
  vars:
    openshift_certificate_expiry_warning_days: 1500
    openshift_certificate_expiry_save_json_results: yes
    openshift_certificate_expiry_generate_html_report: yes
    openshift_certificate_expiry_show_all: yes
  roles:
    - role: openshift_certificate_expiry
```

easy-mode.yaml Playbook を実行するには、以下を実行します。

```
$ ansible-playbook -v -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
checks/certificate_expiry/easy-mode.yaml
```

他のサンプル Playbook
その他のサンプル Playbook も **/usr/share/ansible/openshift-ansible/playbooks/certificate_expiry/** ディレクトリーから直接実行できます。

表10.3 他のサンプル Playbook

ファイル名	使用法
default.yaml	**openshift_certificate_expiry** ロールのデフォルトの動作を生成します。
html_and_json_default_paths.yaml	HTML および JSON アーティファクトをデフォルトのパスに生成します。
longer_warning_period.yaml	有効期限切れの警告期間を 1500 日に変更します。
longer-warning-period-json-results.yaml	有効期限切れの警告期間を 1500 日に変更し、結果を JSON ファイルとして保存します。

これらのサンプル Playbook を実行するには、以下を実行します。

```
$ ansible-playbook -v -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
checks/certificate_expiry/<playbook>
```

10.2.3. 出力形式

上述のように、チェックレポートは 2 つの形式で出力できます。機械の解析に適した JSON 形式と簡単にスキミングできる定型化された HTML ページを使用できます。

HTML レポート
インストーラーには、HTML レポートのサンプルが付属しています。以下のファイルをブラウザーで開いて表示できます。

/usr/share/ansible/openshift-ansible/roles/openshift_certificate_expiry/examples/cert-expiry-report.html

JSON レポート
保存された JSON の結果には、**data** と **summary** の 2 つの最上位のキーがあります。

data キーは、検証された各ホストの名前がキーとして、該当するホストごとに特定された証明書の確認結果が値として含まれるハッシュです。

summary キーは、以下の条件に当てはまる証明書の合計数をまとめたハッシュです。

- クラスター全体で検査済み
- 問題なし
- 設定された警告期間内に有効期限が切れる
- すでに有効期限が切れている

完全な JSON レポートのサンプルについては、**/usr/share/ansible/openshift-ansible/roles/openshift_certificate_expiry/examples/cert-expiry-report.json** を参照してください。

JSON データのサマリーでは、さまざまなコマンドラインツールを使用して警告や有効期限を簡単にチェックできます。たとえば、**grep** を使用して **summary** という語を検索し、一致した箇所の後ろの 2 行を出力できます (**-A2**)。

```
$ grep -A2 summary /tmp/cert-expiry-report.json
    "summary": {
        "warning": 16,
        "expired": 0
```

また、**jq** ツールが使用可能な場合は、このツールを使用して特定の値を選択できます。以下の最初の 2 つの例は、**warning** または **expired** のいずれかの値を選択する方法を示しています。3 つ目の例は、両方の値を一度に選択する方法を示しています。

```
$ jq '.summary.warning' /tmp/cert-expiry-report.json
16

$ jq '.summary.expired' /tmp/cert-expiry-report.json
0

$ jq '.summary.warning,.summary.expired' /tmp/cert-expiry-report.json
16
0
```

10.3. 証明書の再デプロイ

該当するすべてのホストにマスター、etcd、ノード、レジストリーまたはルーター証明書を再デプロイするには、以下の Playbook を使用します。現在の CA を使用してこれらすべての証明書を一度に再デプロイすることも、特定のコンポーネント用の証明書のみを再デプロイすることも、新しく生成された CA またはカスタム CA 自体を再デプロイすることもできます。

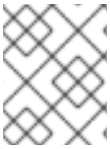
証明書の有効期限切れ Playbook と同様に、これらの Playbook はクラスターを表すインベントリーファイルを使用して実行する必要があります。

特にインベントリーでは、すべてのホスト名と IP アドレスが現在のクラスター設定に一致するように、以下の変数でそれらを指定するか、または上書きする必要があります。

- **openshift_hostname**
- **openshift_public_hostname**
- **openshift_public_ip**
- **openshift_master_cluster_hostname**
- **openshift_master_cluster_public_hostname**

以下のコマンドを実行すると、必要な Playbook が利用できるようになります。

```
# yum install atomic-openshift-utils
```



注記

再デプロイ中に自動生成された証明書の有効性 (有効期限が切れるまでの日数) も Ansible で設定できます。「証明書の有効性の設定」を参照してください。

注記

OpenShift Container Platform CA および etcd 証明書の有効期限は 5 年です。署名付きの OpenShift Container Platform 証明書の有効期限は 2 年です。

10.3.1. 現行の OpenShift Container Platform および etcd CA を使用したすべての証明書の再デプロイ

redeploy-certificates.yml Playbook は、OpenShift Container Platform CA 証明書を再生成**しません**。新しいマスター 、etcd、ノード、レジストリー、およびルーター証明書は、新規の証明書に署名するために現在の CA 証明書を使用して作成されます。

これには、以下の順次の再起動も伴います。

- etcd
- マスターサービス
- ノードサービス

現行の OpenShift Container Platform CA を使用してマスター、etcd、およびノード証明書を再デプロイするには、この Playbook を実行し、インベントリーファイルを指定します。

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/redeploy-
certificates.yml
```

10.3.2. 新規またはカスタムの OpenShift Container Platform CA の再デプロイ

openshift-master/redeploy-openshift-ca.yml Playbook は、新規の CA 証明書を生成し、クライアントの **kubeconfig** ファイルとノードの信頼できる CA のデータベース (CA-trust) を含むすべてのコンポーネントに更新されたバンドルを配布することによって OpenShift Container Platform CA 証明書を再

デプロイします。

これには、以下の順次の再起動も伴います。

- マスターサービス
- ノードサービス
- Docker

さらに、証明書を再デプロイする際には、OpenShift Container Platform によって生成される CA を使用する代わりに、カスタム CA 証明書を指定することもできます。

マスターサービスが再起動すると、レジストリーとルーターは、再デプロイされなくてもマスターと引き続き通信できます。これは、マスターの提供証明書が同一であり、レジストリーとルーターの CA が依然として有効であるためです。

新たに生成される CA またはカスタム CA を再デプロイするには、以下を実行します。

1. カスタム CA を使用する場合は、インベントリーファイルに以下の変数を設定します。現在の CA を使用する場合は、この手順を省略します。

```
# Configure custom ca certificate
# NOTE: CA certificate will not be replaced with existing clusters.
# This option may only be specified when creating a new cluster or
# when redeploying cluster certificates with the redeploy-
certificates
# playbook.
openshift_master_ca_certificate={'certfile': '</path/to/ca.crt>',
'keyfile': '</path/to/ca.key>'}
```

   CA 証明書が中間 CA によって発行された場合、CA が子の証明書を検証できるように、バンドルされた証明書に完全なチェーン (中間証明書とルート証明書) が含まれている必要があります。

   以下は例を示しています。

```
$ cat intermediate/certs/intermediate.cert.pem \
      certs/ca.cert.pem >> intermediate/certs/ca-chain.cert.pem
```

2. **openshift-master/redeploy-openshift-ca.yml** Playbook を実行し、インベントリーファイルを指定します。

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
master/redeploy-openshift-ca.yml
```

新規の OpenShift Container Platform CA が導入されている場合、新規の CA によって署名された証明書をすべてのコンポーネントに再デプロイする必要がある場合にはいつでも **openshift-master/redeploy-certificates.yml** Playbook を使用できます。

10.3.3. 新規 etcd CA の再デプロイ

openshift-etcd/redeploy-ca.yml Playbook は、新規 CA 証明書を生成し、すべての etcd ピアとマスタークライアントに更新したバンドルを配布することによって etcd CA 証明書を再デプロイします。

これには、以下の順次の再起動も伴います。

- etcd
- マスターサービス

新たに生成された etcd CA を再デプロイするには、以下を実行します。

1. **openshift-etcd/redeploy-ca.yml** Playbook を実行し、インベントリーファイルを指定します。

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
etcd/redeploy-ca.yml
```

新規 etcd CA が導入されている場合、新規 CA によって署名された証明書を etcd ピアとマスタークライアントに再デプロイする必要がある場合にはいつでも **openshift-etcd/redeploy-certificates.yml** Playbook を使用できます。または、**redeploy-certificates.yml** Playbook を使用して、etcd ピアとマスタークライアントに加えて、OpenShift Container Platform コンポーネントの証明書も再デプロイできます。

10.3.4. マスター証明書のみの再デプロイ

openshift-master/redeploy-certificates.yml Playbook は、マスター証明書のみを再デプロイします。これには、マスターサービスの順次の再起動も伴います。

マスター証明書を再デプロイするには、この Playbook を実行し、インベントリーファイルを指定します。

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
master/redeploy-certificates.yml
```

重要

この Playbook を実行した後に、サービス提供証明書を含む既存のシークレットを削除するか、またはアノテーションを削除し、適切なサービスに再度追加して、サービス提供証明書またはキーペアを再生成する必要があります。

10.3.5. etcd 証明書のみの再デプロイ

openshift-etcd/redeploy-certificates.yml Playbook は、マスタークライアント証明書を含む etcd 証明書のみを再デプロイします。

これには、以下の順次の再起動も伴います。

- etcd
- マスターサービス

etcd 証明書を再デプロイするには、この Playbook を実行し、インベントリーファイルを指定します。

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
etcd/redeploy-certificates.yml
```

10.3.6. ノード証明書のみの再デプロイ

openshift-node/redeploy-certificates.yml Playbook は、ノード証明書のみを再デプロイします。これには、ノードサービスの順次の再起動も伴います。

ノード証明書を再デプロイするには、この Playbook を実行し、インベントリーファイルを指定します

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
node/redeploy-certificates.yml
```

10.3.7. レジストリー証明書またはルーター証明書のみの再デプロイ

openshift-hosted/redeploy-registry-certificates.yml Playbook と **openshift-hosted/redeploy-router-certificates.yml** Playbook は、インストーラーによって作成されたレジストリー証明書とルーター証明書を置き換えます。レジストリーとルーターでカスタム証明書が使用されている場合にそれらを手動で置き換えるには、「 カスタムのレジストリー証明書またはルーター証明書の再デプロイ」を参照してください。

10.3.7.1. レジストリー証明書のみの再デプロイ

レジストリー証明書を再デプロイするには、以下の Playbook を実行し、インベントリーファイルを指定します。

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
hosted/redeploy-registry-certificates.yml
```

10.3.7.2. ルーター証明書のみの再デプロイ

ルーター証明書を再デプロイするには、以下の Playbook を実行し、インベントリーファイルを指定します。

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
hosted/redeploy-router-certificates.yml
```

10.3.8. カスタムのレジストリー証明書またはルーター証明書の再デプロイ

再デプロイされた CA が原因でノードが退避させられると、レジストリー Pod とルーター Pod が再起動されます。レジストリー証明書とルーター証明書を新規 CA と共に再デプロイしなかった場合は、それらが古い証明書を使用してマスターにアクセスできなくなるため、停止状態が生じることがあります。

証明書を再デプロイする Playbook は、カスタムのレジストリー証明書またはルーター証明書を再デプロイできません。この問題を解決するため、レジストリー証明書とルーター証明書を手動で再デプロイする必要があります。

10.3.8.1. 手動によるレジストリー証明書の再デプロイ

レジストリー証明書を手動で再デプロイするには、新規レジストリー証明書を **registry-certificates** という名前のシークレットに追加してから、レジストリーを再デプロイする必要があります。

1. これ以降の手順では **default** プロジェクトに切り替えます。

```
$ oc project default
```

2. 最初にレジストリーを OpenShift Container Platform 3.1 以前で作成した場合は、環境変数が証明書を保存するために使用されている場合があります (この方法は現在は推奨されていません。代わりにシークレットをご使用ください)。

   a. 以下のコマンドを実行し、**OPENSHIFT_CA_DATA**、**OPENSHIFT_CERT_DATA**、および **OPENSHIFT_KEY_DATA** 環境変数を探します。

```
$ oc env dc/docker-registry --list
```

   b. これらの環境変数が存在しない場合は、この手順を省略します。存在する場合は、以下の **ClusterRoleBinding** を作成します。

```
$ cat <<EOF |
apiVersion: v1
groupNames: null
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: registry-registry-role
roleRef:
  kind: ClusterRole
  name: system:registry
subjects:
- kind: ServiceAccount
  name: registry
  namespace: default
userNames:
- system:serviceaccount:default:registry
EOF
oc create -f -
```

   次に、以下のコマンドを実行して環境変数を削除します。

```
$ oc env dc/docker-registry OPENSHIFT_CA_DATA-
OPENSHIFT_CERT_DATA- OPENSHIFT_KEY_DATA- OPENSHIFT_MASTER-
```

3. 以下の環境変数をローカルに設定し、後で使用するコマンドを単純化します。

```
$ REGISTRY_IP=`oc get service docker-registry -o
jsonpath='{.spec.clusterIP}'`
$ REGISTRY_HOSTNAME=`oc get route/docker-registry -o
jsonpath='{.spec.host}'`
```

4. 新規レジストリー証明書を作成します。

```
$ oc adm ca create-server-cert \
    --signer-cert=/etc/origin/master/ca.crt \
    --signer-key=/etc/origin/master/ca.key \
    --hostnames=$REGISTRY_IP,docker-registry.default.svc,docker-
registry.default.svc.cluster.local,$REGISTRY_HOSTNAME \
```

```
    --cert=/etc/origin/master/registry.crt \
    --key=/etc/origin/master/registry.key \
    --signer-serial=/etc/origin/master/ca.serial.txt
```

Ansible ホストインベントリーファイル (デフォルトで **/etc/ansible/hosts**) に最初に一覧表示されているマスターから **oc adm** コマンドを実行します。

5. **registry-certificates** シークレットを新規レジストリー証明書で更新します。

```
$ oc create secret generic registry-certificates \
    --from-
file=/etc/origin/master/registry.crt,/etc/origin/master/registry.key
\
    -o json --dry-run | oc replace -f -
```

6. レジストリーを再デプロイします。

```
$ oc rollout latest dc/docker-registry
```

10.3.8.2. 手動によるルーター証明書の再デプロイ

ルーターの初回デプロイ時に、アノテーションがサービス提供証明書シークレットを自動的に作成するルーターのサービスに追加されます。

ルーター証明書を手動で再デプロイするため、そのサービス提供証明書の再作成をトリガーできます。これは、シークレットを削除し、アノテーションを削除してから **router** サービスに再度追加し、ルーターを再デプロイして実行できます。

1. これ以降の手順では **default** プロジェクトに切り替えます。

```
$ oc project default
```

2. 最初にレジストリーを OpenShift Container Platform 3.1 以前で作成した場合は、環境変数が証明書を保存するために使用されている場合があります (この方法は現在は推奨されていません。代わりにサービス提供証明書シークレットをご使用ください)。

   a. 以下のコマンドを実行し、**OPENSHIFT_CA_DATA**、**OPENSHIFT_CERT_DATA**、および **OPENSHIFT_KEY_DATA** 環境変数を探します。

```
$ oc env dc/router --list
```

   b. それらの変数が存在する場合は、以下の **ClusterRoleBinding** を作成します。

```
$ cat <<EOF |
apiVersion: v1
groupNames: null
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: router-router-role
roleRef:
  kind: ClusterRole
  name: system:router
subjects:
```

```
  - kind: ServiceAccount
    name: router
    namespace: default
userNames:
- system:serviceaccount:default:router
EOF
oc create -f -
```

c. それらの変数が存在する場合は、以下のコマンドを実行してそれらを削除します。

```
$ oc env dc/router OPENSHIFT_CA_DATA- OPENSHIFT_CERT_DATA-
OPENSHIFT_KEY_DATA- OPENSHIFT_MASTER-
```

3. 証明書を取得します。

- 外部の認証局 (CA) を使用して証明書に署名する場合、以下の内部プロセスに従って、新規の証明書を作成し、これを OpenShift Container Platform に指定します。
- 内部の OpenShift Container Platform CA を使用して証明書に署名する場合は、以下のコマンドを実行します。

重要

以下のコマンドは、内部で署名される証明書を生成します。これは、OpenShift Container Platform CA を信頼するクライアントによってのみ信頼されます。

```
$ cd /root
$ mkdir cert ; cd cert
$ oc adm ca create-server-cert \
    --signer-cert=/etc/origin/master/ca.crt \
    --signer-key=/etc/origin/master/ca.key \
    --signer-serial=/etc/origin/master/ca.serial.txt \
    --hostnames='*.hostnames.for.the.certificate' \
    --cert=router.crt \
    --key=router.key \
```

これらのコマンドは以下のファイルを生成します。

- **router.crt** という名前の新規の証明書
- 署名する CA 証明書チェーン **/etc/origin/master/ca.crt** のコピーです。このチェーンには中間の CA を使用する場合に複数の証明書が含まれる場合があります。
- 名前が **router.key** の、対応するプライベートキーで、

4. 生成された証明書を連結する新規ファイルを作成します。

```
$ cat router.crt /etc/origin/master/ca.crt router.key > router.pem
```

5. 新規シークレットを生成する前に、現在のシークレットをバックアップします。

```
$ oc export secret router-certs > ~/old-router-certs-secret.yaml
```

6. 新規の証明書およびキーを保持する新規シークレットを作成し、既存のシークレットの内容を置き換えます。

```
$ oc create secret tls router-certs --cert=router.pem \ 1
    --key=router.key -o json --dry-run | \
    oc replace -f -
```

1 **router.pem** は、生成した証明書の連結を含むファイルです。

7. 以下のアノテーションを **router** サービスから削除します。

```
$ oc annotate service router \
    service.alpha.openshift.io/serving-cert-secret-name- \
    service.alpha.openshift.io/serving-cert-signed-by-
```

8. アノテーションを再度追加します。

```
$ oc annotate service router \
    service.alpha.openshift.io/serving-cert-secret-name=router-certs
```

9. ルーターを再デプロイします。

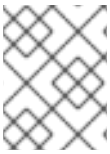
```
$ oc rollout latest dc/router
```

第11章 認証およびユーザーエージェントの設定

11.1. 概要

OpenShift Container Platform のマスターには、OAuth サーバーがビルトインされています。開発者および管理者は OAuth アクセストークンを取得して API に対する認証を実行します。

管理者はマスター設定ファイルを使用して OAuth を アイデンティティープロバイダーを指定するように設定できます。アイデンティティープロバイダーは、クラスターインストールの実行中に設定するのが最適ですが、インストール後に設定することもできます。



注記

/、:、および % を含む OpenShift Container Platform ユーザー名はサポートされません。

Deny All アイデンティティープロバイダーがデフォルトで使用されます。このアイデンティティープロバイダーは、すべてのユーザー名とパスワードについてアクセスを拒否します。アクセスを許可するには、別のアイデンティティープロバイダーを選択し、マスター設定ファイル (デフォルトでは、**/etc/origin/master/master-config.yaml** にあります) を適宜設定する必要があります。

設定ファイルなしでマスターを実行する場合は、Allow All アイデンティティープロバイダーがデフォルトで使用されます。このアイデンティティープロバイダーは、空でないユーザー名とパスワードによるログインをすべて許可します。これはテストを行う場合に便利です。その他のアイデンティティープロバイダーを使用するか、token、grant、または session オプション を変更する場合は、設定ファイルからマスターを実行する必要があります。

注記

外部ユーザーのセットアップを管理するためのロールが割り当てられている必要があります。

アイデンティティープロバイダーに変更を加えたら、変更を有効にするためにマスターサービスを再起動する必要があります。

```
# master-restart api
# master-restart controllers
```

11.2. アイデンティティープロバイダーパラメーター

すべてのアイデンティティープロバイダーには共通する 4 つのパラメーターがあります。

パラメーター	説明
name	プロバイダー名は、プロバイダーのユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。

パラメーター	説明
`challenge`	**true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、**`WWW-Authenticate`** challenge ヘッダー付きで送信されます。 これはすべてのアイデンティティープロバイダーでサポートされる訳ではありません。 ブラウザークライアントに対するクロスサイトリクエストフォージェリー (CSRF) 攻撃を防止するため、基本的な認証 challenge は **`X-CSRF-Token`** ヘッダーが要求に存在する場合にのみ送信されます。基本的な **`WWW-Authenticate`** challenge を受信する必要があるクライアントでは、このヘッダーを空でない値に設定する必要があります。
`login`	**true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。これはすべてのアイデンティティープロバイダーによってサポートされている訳ではありません。 ユーザーがアイデンティティープロバイダーのログインにリダイレクトされる前にブランドページに移動するようにする場合、マスター設定ファイルで **`oauthConfig → alwaysShowProviderSelection: true`** を設定します。このプロバイダー選択ページは カスタマイズできます。
`mappingMethod`	新規アイデンティティーがログイン時にユーザーにマップされる方法を定義します。以下の値のいずれかを入力します。 **claim** デフォルトの値です。アイデンティティーの推奨ユーザー名が指定されたユーザーをプロビジョニングします。そのユーザー名が指定されたユーザーがすでに別のアイデンティティーにマッピングされている場合は失敗します。 **lookup** 既存のアイデンティティー、ユーザーアイデンティティーマッピング、およびユーザーを検索しますが、ユーザーまたはアイデンティティーの自動プロビジョニングは行いません。これにより、クラスター管理者は手動で、または外部のプロセスを使用してアイデンティティーとユーザーを設定できます。この方法を使用する場合は、ユーザーを手動でプロビジョニングする必要があります。「lookup マッピング方法を使用する場合のユーザーの手動プロビジョニング」を参照してください。 **generate** アイデンティティーの推奨ユーザー名が指定されたユーザーをプロビジョニングします。この推奨ユーザー名が指定されたユーザーがすでに既存のアイデンティティーにマッピングされている場合は、**myuser2** などの一意のユーザー名が生成されます。この方法は、OpenShift Container Platform のユーザー名とアイデンティティープロバイダーのユーザー名とを完全一致させる必要のある外部プロセス (LDAP グループ同期など) と組み合わせて使用することはできません。 **add** アイデンティティーの推奨ユーザー名が指定されたユーザーをプロビジョニングします。推奨ユーザー名が指定されたユーザーがすでに存在する場合、アイデンティティーは既存のユーザーにマッピングされ、そのユーザーの既存のアイデンティティーマッピングに追加されます。これは、同じユーザーセットを識別して同じユーザー名にマッピングするアイデンティティープロバイダーが複数設定されている場合に必要です。

注記

mappingMethod パラメーターを **add** に設定すると、アイデンティティープロバイダーの追加または変更時に新規プロバイダーのアイデンティティーを既存ユーザーにマッピングできます。

11.3. アイデンティティープロバイダーの設定

OpenShift Container Platform は単一のアイデンティティープロバイダーの設定のみをサポートします。ただし、LDAP フェイルオーバーなどのより複雑な設定の Basic 認証を拡張することが可能です。

これらのパラメーターを使用して、インストール時またはインストール後にアイデンティティープロバイダーを定義できます。

11.3.1. Ansible を使用したアイデンティティープロバイダーの設定

初回のクラスターインストールでは、Deny All アイデンティティープロバイダーがデフォルトで設定されます。ただし、これは、インベントリーファイルで設定可能な **openshift_master_identity_providers** パラメーターを使用してインストール時に上書きすることができます。OAuth 設定のセッションオプションもインベントリーファイルで設定できます。

Ansible を使用したアイデンティティープロバイダー設定の例

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
#openshift_master_htpasswd_users={'user1': '<pre-hashed password>',
'user2': '<pre-hashed password>'
# or
#openshift_master_htpasswd_file=<path to local pre-generated htpasswd
file>

# Allow all auth
#openshift_master_identity_providers=[{'name': 'allow_all', 'login':
'true', 'challenge': 'true', 'kind': 'AllowAllPasswordIdentityProvider'}]

# LDAP auth
#openshift_master_identity_providers=[{'name': 'my_ldap_provider',
'challenge': 'true', 'login': 'true', 'kind':
'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email':
['mail'], 'name': ['cn'], 'preferredUsername': ['uid']}, 'bindDN': '',
'bindPassword': '', 'ca': '', 'insecure': 'false', 'url':
'ldap://ldap.example.com:389/ou=users,dc=example,dc=com?uid'}]
# Configuring the ldap ca certificate (1)
#openshift_master_ldap_ca=<ca text>
# or
#openshift_master_ldap_ca_file=<path to local ca file to use>

# Available variables for configuring certificates for other identity
providers:
#openshift_master_openid_ca
#openshift_master_openid_ca_file
#openshift_master_request_header_ca
#openshift_master_request_header_ca_file
```

1 **openshift_master_identity_providers** パラメーターに CA 証明書の場所を指定する場合、証明書の値を **openshift_master_ldap_ca** パラメーターに指定したり、パスを **openshift_master_ldap_ca_file** パラメーターに使用したりしないようにしてください。

11.3.2. マスター設定ファイルでのアイデンティティープロバイダーの設定

マスター設定ファイルを変更することで、マスターホストで必要なアイデンティティープロバイダーを使用した認証を設定できます。

例11.1 マスター設定ファイルでのアイデンティティープロバイダーの設定例

```
...
oauthConfig:
  identityProviders:
  - name: htpasswd_auth
    challenge: true
    login: true
    mappingMethod: "claim"
...
```

デフォルトの **claim** 値に設定されている場合、アイデンティティーを以前に存在していたユーザー名にマッピングすると OAuth が失敗します。

11.3.3. アイデンティティープロバイダーまたはメソッドの設定

11.3.3.1. lookup マッピング方法を使用する場合のユーザーの手動プロビジョニング

lookup マッピング方法を使用する場合、ユーザープロビジョニングは外部システムによって API 経由で行われます。通常、アイデンティティーは、ログイン時にユーザーに自動的にマッピングされます。「 lookup」マッピング方法は、この自動マッピングを自動的に無効にします。そのため、ユーザーを手動でプロビジョニングする必要があります。

identity オブジェクトの詳細については、「Identity」ユーザー API オブジェクトを参照してください。

lookup マッピング方法を使用する場合は、アイデンティティープロバイダーを設定した後にユーザーごとに以下の手順を使用してください。

1. OpenShift Container Platform ユーザーを作成します (まだ作成していない場合)。

   ```
   $ oc create user <username>
   ```

   たとえば、以下のコマンドを実行して OpenShift Container Platform ユーザー **bob** を作成します。

   ```
   $ oc create user bob
   ```

2. OpenShift Container Platform アイデンティティーを作成します (まだ作成していない場合)。アイデンティティープロバイダーの名前と、アイデンティティープロバイダーの範囲でこのアイデンティティーを一意に表す名前を使用します。

```
$ oc create identity <identity-provider>:<user-id-from-identity-
provider>
```

<identity-provider> は、マスター設定のアイデンティティープロバイダーの名前であり、以下の該当するアイデンティティープロバイダーセクションに表示されています。

たとえば、以下のコマンドを実行すると、アイデンティティープロバイダーが **ldap_provider**、アイデンティティープロバイダーのユーザー名が **bob_s** のアイデンティティーが作成されます。

```
$ oc create identity ldap_provider:bob_s
```

3. 作成したユーザーとアイデンティティーのユーザー/アイデンティティーマッピングを作成します。

```
$ oc create useridentitymapping <identity-provider>:<user-id-from-
identity-provider> <username>
```

たとえば、以下のコマンドを実行すると、アイデンティティーがユーザーにマッピングされます。

```
$ oc create useridentitymapping ldap_provider:bob_s bob
```

11.3.4. Allow All

空でないユーザー名とパスワードによるログインを許可するには、 **identityProviders** スタンザに **AllowAllPasswordIdentityProvider** を設定します。

例11.2 AllowAllPasswordIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: my_allow_provider 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: AllowAllPasswordIdentityProvider
```

1. このプロバイダー名は、プロバイダーのユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。
2. **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。
3. **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。
4. このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。

11.3.5. Deny All

すべてのユーザー名とパスワードについてアクセスを拒否するには、 **identityProviders** スタンザに **DenyAllPasswordIdentityProvider** を設定します。

例11.3 DenyAllPasswordIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: my_deny_provider (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: DenyAllPasswordIdentityProvider
```

(1) このプロバイダー名は、プロバイダーのユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。

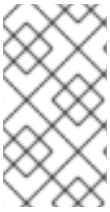
(2) **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。

(3) **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。

(4) このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。

11.3.6. HTPasswd

ユーザー名およびパスワードを **htpasswd** を使用して生成されたフラットファイルに対して検証するには、**identityProviders** スタンザに **HTPasswdPasswordIdentityProvider** を設定します。



注記

htpasswd ユーティリティーは **httpd-tools** パッケージにあります。

```
# yum install httpd-tools
```

OpenShift Container Platform では、 Bcrypt、SHA-1、および MD5 暗号化ハッシュ関数がサポートされ、MD5 は **htpasswd** のデフォルトです。プレーンテキスト、暗号化テキスト、およびその他のハッシュ関数は、現時点ではサポートされていません。

フラットファイルは、その変更時間が変わると再度読み取られます。サーバーの再起動は必要ありません。

htpasswd コマンドを使用するには、以下の手順を実行します。

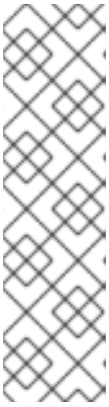
- ユーザー名とハッシュされたパスワードを含むフラットファイルを作成するには、以下を実行します。

```
$ htpasswd -c </path/to/users.htpasswd> <user_name>
```

次に、ユーザーのクリアテキストのパスワードを入力し、確認します。コマンドにより、ハッシュされたバージョンのパスワードが生成されます。

以下は例を示しています。

```
htpasswd -c users.htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```



注記

-b オプションを追加すると、パスワードをコマンドラインに指定できます。

```
$ htpasswd -c -b <user_name> <password>
```

以下は例を示しています。

```
$ htpasswd -c -b file user1 MyPassword!
Adding password for user user1
```

- ファイルにログインを追加するか、ログインを更新するには、以下のコマンドを実行します。

```
$ htpasswd </path/to/users.htpasswd> <user_name>
```

- ファイルからログインを削除するには、以下のコマンドを実行します。

```
$ htpasswd -D </path/to/users.htpasswd> <user_name>
```

例11.4 HTPasswdPasswordIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: my_htpasswd_provider (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: HTPasswdPasswordIdentityProvider
      file: /path/to/users.htpasswd (5)
```

(1) このプロバイダー名は、プロバイダーのユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。

2 **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **`WWW-Authenticate`** challenge ヘッダーと共に送信されます。

3 **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。

4 このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。

5 **`htpasswd`** を使用して生成されたファイルです。

11.3.7. Keystone

Keystone は、アイデンティティー、トークン、カタログ、およびポリシーサービスを提供する OpenStack プロジェクトです。OpenShift Container Platform クラスターと Keystone を統合すると、内部データベースにユーザーを格納するように設定された OpenStack Keystone v3 サーバーによる共有認証を有効にできます。この設定を完了すると、ユーザーは Keystone 認証情報を使用して OpenShift Container Platform にログインできるようになります。

11.3.7.1. マスターでの認証の設定

1. 状況に応じて以下のいずれかの手順を実行します。

   - Openshift のインストールがすでに完了している場合は、**/etc/origin/master/master-config.yaml** ファイルを新規ディレクトリーにコピーします。以下は例になります。

     ```
     $ cd /etc/origin/master
     $ mkdir keystoneconfig; cp master-config.yaml keystoneconfig
     ```

   - OpenShift Container Platform をまだインストールしていない場合は、OpenShift Container Platform API サーバーを起動し、(将来の) OpenShift Container Platform マスターのホスト名と、起動コマンドによって作成された設定ファイルを格納するディレクトリーを指定します。

     ```
     $ openshift start master --public-master=<apiserver> --write-
     config=<directory>
     ```

     以下は例を示しています。

     ```
     $ openshift start master --public-
     master=https://myapiserver.com:8443 --write-config=keystoneconfig
     ```

注記

Ansible を使用してインストールする場合は、**`identityProvider`** 設定を Ansible Playbook に追加する必要があります。Ansible を使用してインストールした後、以下の手順に従って設定を手動で変更した場合、インストールツールまたはアップグレードを再実行するたびに変更内容がすべて失われます。

2. 新規の **keystoneconfig/master-config.yaml** ファイルの **identityProviders** スタンザを編集し、**KeystonePasswordIdentityProvider** の設定例をコピーして貼り付け、既存のスタンザを置き換えます。

```
oauthConfig:
  ...
  identityProviders:
  - name: my_keystone_provider (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: KeystonePasswordIdentityProvider
      domainName: default (5)
      url: http://keystone.example.com:5000 (6)
      ca: ca.pem (7)
      certFile: keystone.pem (8)
      keyFile: keystonekey.pem (9)
```

(1) このプロバイダー名は、プロバイダーのユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。

(2) **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。

(3) **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。

(4) このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。

(5) Keystone のドメイン名です。Keystone では、ユーザー名はドメイン固有です。単一ドメインのみがサポートされます。

(6) Keystone サーバーへの接続に使用する URL (必須) です。

(7) オプション: 設定された URL のサーバー証明書を検証するために使用する証明書バンドルです。

(8) オプション: 設定された URL に対して要求を実行する際に提示するクライアント証明書です。

(9) クライアント証明書のキーです。**certFile** が指定されている場合は必須です。

3. 以下の変更を **identityProviders** スタンザに加えます。

   a. プロバイダーの **name** (「my_keystone_provider」) を、使用する Keystone サーバーに合わせて変更します。この名前は、プロバイダーのユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。

   b. 必要な場合、**mappingMethod** を変更して、プロバイダーのアイデンティティーとユーザーオブジェクト間でマッピングを確立する方法を制御します。

c. **domainName** を OpenStack Keystone サーバーのドメイン名に変更します。Keystone では、ユーザー名はドメイン固有です。単一ドメインのみがサポートされます。

d. OpenStack Keystone への接続に使用する **url** を指定します。

e. オプションで、設定された URL のサーバー証明書を検証できるように **ca** を使用する証明書バンドルに変更します。

f. オプションで、**certFile** を、設定された URL に対する要求の実行時に提示するクライアント証明書に変更します。

g. **certFile** が指定されている場合は、**keyFile** をクライアント証明書のキーに変更する必要があります。

4. 変更を保存してファイルを閉じます。

5. OpenShift Container Platform API サーバーを起動し、変更したばかりの設定ファイルを指定します。

```
$ openshift start master --config=<path/to/modified/config>/master-
config.yaml
```

設定が完了すると、OpenShift Container Platform Web コンソールにログインするすべてのユーザーに Keystone 認証情報を使用してログインすることを求めるプロンプトが出されます。

11.3.7.2. Keystone 認証を使用するユーザーの作成

外部認証プロバイダー (この場合は Keystone) と統合する場合、OpenShift Container Platform にはユーザーを作成しません。Keystone は「system of record」であり、ユーザーは Keystone データベースで定義され、設定された認証サーバーに対する有効な Keystone ユーザー名を持つ任意のユーザーがログインできます。

ユーザーを OpenShift Container Platform に追加するには、そのユーザーが Keystone データベースに存在している必要があります。また、必要な場合はそのユーザーの新しい Keystone アカウントを作成する必要があります。

11.3.7.3. ユーザーの確認

1 名以上のユーザーがログインしたら、**oc get users** を実行してユーザーの一覧を表示し、ユーザーが正しく作成されていることを確認できます。

例11.5 oc get users コマンドの出力

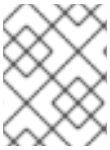
```
$ oc get users
NAME         UID                                    FULL NAME
IDENTITIES
bobsmith     a0c1d95c-1cb5-11e6-a04a-002186a28631   Bob Smith
keystone:bobsmith (1)
```

(1) OpenShift Container Platform のアイデンティティーは、Keystone ユーザー名とそれにプレフィックスとして付加されるアイデンティティープロバイダー名で構成されます。

ここからは、ユーザーロールの管理方法を学習することをお勧めします。

11.3.8. LDAP 認証

単純なバインド認証を使用してユーザー名とパスワードを LDAPv3 サーバーに対して検証するには、**identityProviders** スタンザに **LDAPPasswordIdentityProvider** を設定します。



注記

これらの手順に従うのではなく、LDAP サーバーのフェイルオーバーを要求する場合には、SSSD で LDAP フェイルオーバーを設定して Basic 認証の方法を拡張します。

認証時に、指定されたユーザー名に一致するエントリーが LDAP ディレクトリーで検索されます。完全一致が 1 つ検出された場合には、エントリーの識別名 (DN) と指定されたパスワードを使用した単純なバインドが試みられます。

以下の手順が実行されます。

1. 設定された **url** の属性およびフィルターとユーザーが指定したユーザー名を組み合わせて検索フィルターを生成します。
2. 生成されたフィルターを使用してディレクトリーを検索します。検索によって 1 つもエントリーが返されない場合は、アクセスを拒否します。
3. 検索で取得したエントリーの DN とユーザー指定のパスワードを使用して LDAP サーバーへのバインドを試みます。
4. バインドが失敗した場合は、アクセスを拒否します。
5. バインドが成功した場合は、アイデンティティー、電子メールアドレス、表示名、および推奨ユーザー名として設定された属性を使用してアイデンティティーを作成します。

設定される **url** は、LDAP ホストと使用する検索パラメーターを指定する RFC 2255 URL です。URL の構文は以下のようになります。

```
ldap://host:port/basedn?attribute?scope?filter
```

上記の例は、以下のコンポーネントで構成されています。

URL コンポーネント	説明
ldap	通常の LDAP の場合は、文字列 **ldap** を使用します。セキュアな LDAP (LDAPS) の場合は、代わりに **ldaps** を使用します。
host:port	LDAP サーバーの名前とポートで。デフォルトは、ldap の場合は **localhost:389**、LDAPS の場合は **localhost:636** です。
basedn	すべての検索が開始されるディレクトリーのブランチの DN です。これは少なくともディレクトリーツリーの最上位になければなりませんが、ディレクトリーのサブツリーを指定することもできます。

URL コンポーネント	説明
attribute	検索対象の属性です。RFC 2255 はコンマ区切りの属性の一覧を許可しますが、属性をどれだけ指定しても最初の属性のみが使用されます。属性を指定しない場合は、デフォルトで **uid** が使用されます。使用しているサブツリーのすべてのエントリー間で一意の属性を選択することを推奨します。
scope	検索の範囲です。**one** または **sub** のいずれかを指定できます。範囲を指定しない場合は、デフォルトの範囲として **sub** が使用されます。
filter	有効な LDAP 検索フィルターです。指定しない場合、デフォルトは **(objectClass=*)** です。

検索の実行時に属性、フィルター、指定したユーザー名が組み合わされて以下のような検索フィルターが作成されます。

```
(&(<filter>)(<attribute>=<username>))
```

たとえば、以下の URL について見てみましょう。

```
ldap://ldap.example.com/o=Acme?cn?sub?(enabled=true)
```

クライアントが **bob** というユーザー名を使用して接続を試みる場合、生成される検索フィルターは **(&(enabled=true)(cn=bob))** になります。

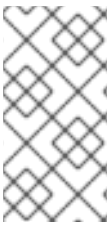
LDAP ディレクトリーの検索に認証が必要な場合は、エントリー検索の実行に使用する **bindDN** と **bindPassword** を指定します。

LDAPPasswordIdentityProvider を使用したマスターの設定

```
oauthConfig:
  ...
  identityProviders:
  - name: "my_ldap_provider" 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: LDAPPasswordIdentityProvider
      attributes:
        id: 5
        - dn
        email: 6
        - mail
        name: 7
        - cn
        preferredUsername: 8
```

```
      - uid
    bindDN: "" 9
    bindPassword: "" 10
    ca: my-ldap-ca-bundle.crt 11
    insecure: false 12
    url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid" 13
```

1. このプロバイダー名は返されるユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。
2. **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **`WWW-Authenticate`** challenge ヘッダーと共に送信されます。
3. **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。
4. このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。
5. アイデンティティーとして使用する属性の一覧です。最初の空でない属性が使用されます。少なくとも 1 つの属性が必要です。一覧表示される属性のいずれにも値がない場合、認証は失敗します。
6. メールアドレスとして使用する属性の一覧です。最初の空でない属性が使用されます。
7. 表示名として使用する属性の一覧です。最初の空でない属性が使用されます。
8. このアイデンティティーのユーザーをプロビジョニングする際に推奨ユーザー名として使用する属性の一覧です。最初の空でない属性が使用されます。
9. 検索フェーズでバインドするために使用するオプションの DN です。
10. 検索フェーズでバインドするために使用するオプションのパスワードです。この値は、環境変数、外部ファイル、または暗号化ファイルで指定することもできます。
11. 設定された URL のサーバー証明書を検証するために使用する証明書バンドルです。空の場合、システムで信頼されたルートが使用されます。これは **insecure: false** の場合にのみ適用されます。
12. **true** の場合、サーバーへの TLS 接続は確立されません。**false** の場合、**`ldaps://`** URL は TLS を使用して接続し、**`ldap://`** URL は TLS にアップグレードされます。
13. LDAP ホストと使用する検索パラメーターを指定する RFC 2255 URL です (上記を参照してください)。



注記

LDAP 統合のためのユーザーのホワイトリストを作成するには、**lookup** マッピング方法を使用します。LDAP からのログインが許可される前に、クラスター管理者は各 LDAP ユーザーのアイデンティティーとユーザーオブジェクトを作成する必要があります。

11.3.9. Basic 認証 (リモート)

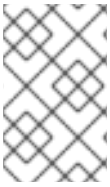
Basic 認証は、ユーザーがリモートのアイデンティティープロバイダーに対して検証した認証情報を使用して OpenShift Container Platform にログインすることを可能にする汎用バックエンド統合メカニズムです。

Basic 認証は汎用性があるため、このアイデンティティープロバイダー使用して詳細な認証設定を実行できます。詳細なリモート Basic 認証設定の開始点として、LDAP フェイルオーバーを設定したり、コンテナー化された Basic 認証リポジトリーを使用できます。

注意

Basic 認証では、ユーザー ID とパスワードのスヌーピングを防ぎ、中間者攻撃を回避するためにリモートサーバーへの HTTPS 接続を使用する必要があります。

BasicAuthPasswordIdentityProvider を設定していると、ユーザーはユーザー名とパスワードを OpenShift Container Platform に送信し、サーバー間の要求を行い、認証情報を Basic 認証ヘッダーとして渡すことで、これらの認証情報をリモートサーバーに対して検証することができます。このため、ユーザーはログイン時に認証情報を OpenShift Container Platform に送信する必要があります。



注記

これはユーザー名/パスワードログインの仕組みにのみ有効で、OpenShift Container Platform はリモート認証サーバーに対するネットワーク要求を実行できる必要があります。

identityProviders スタンザで **BasicAuthPasswordIdentityProvider** を設定し、サーバー間の Basic 認証要求を使用してユーザー名とパスワードをリモートサーバーに対して検証できるようにします。ユーザー名とパスワードは Basic 認証で保護されるリモート URL に対して検証され、JSON を返します。

401 応答は認証の失敗を示しています。

200 以外のステータスまたは空でない「エラー」キーはエラーを示しています。

```
{"error":"Error message"}
```

sub (サブジェクト) キーを持つ **200** ステータスは、成功を示しています。

```
{"sub":"userid"} 1
```

1 このサブジェクトは認証ユーザーに固有である必要があり、変更することができません。

正常な応答により、以下のような追加データがオプションで提供されることがあります。

- **name** キーを使用した表示名。以下は例になります。

  ```
  {"sub":"userid", "name": "User Name", ...}
  ```

- **email** キーを使用したメールアドレス。以下は例になります。

  ```
  {"sub":"userid", "email":"user@example.com", ...}
  ```

- **preferred_username** キーを使用した推奨ユーザー名。これは、固有の変更できないサブ

ジェクトがデータベースキーまたは UID であり、判読可能な名前が存在する場合に便利です。これは、認証された ID に対して OpenShift Container Platform ユーザーをプロビジョニングする場合にヒントとして使われます。以下は例を示しています。

```
{"sub":"014fbff9a07c", "preferred_username":"bob", ...}
```

11.3.9.1. マスターでの認証の設定

1. 状況に応じて以下のいずれかの手順を実行します。

   - Openshift のインストールがすでに完了している場合は、**/etc/origin/master/master-config.yaml** ファイルを新規ディレクトリーにコピーします。以下は例になります。

     ```
     $ mkdir basicauthconfig; cp master-config.yaml basicauthconfig
     ```

   - OpenShift Container Platform をまだインストールしていない場合は、OpenShift Container Platform API サーバーを起動し、(将来の) OpenShift Container Platform マスターのホスト名と、起動コマンドによって作成された設定ファイルを格納するディレクトリーを指定します。

     ```
     $ openshift start master --public-master=<apiserver> --write-
     config=<directory>
     ```

     以下は例を示しています。

     ```
     $ openshift start master --public-
     master=https://myapiserver.com:8443 --write-
     config=basicauthconfig
     ```

     注記

     Ansible を使用してインストールする場合は、**identityProvider** 設定を Ansible Playbook に追加する必要があります。Ansible を使用してインストールした後、以下の手順に従って設定を手動で変更した場合、インストールツールまたはアップグレードを再実行するたびに変更内容がすべて失われます。

2. 新規の **master-config.yaml** ファイルの **identityProviders** スタンザを編集し、**BasicAuthPasswordIdentityProvider** 設定のサンプルをコピーして貼り付け、既存のスタンザを置き換えます。

```
oauthConfig:
  ...
  identityProviders:
  - name: my_remote_basic_auth_provider (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: BasicAuthPasswordIdentityProvider
      url: https://www.example.com/remote-idp (5)
```

```
      ca: /path/to/ca.file 6
      certFile: /path/to/client.crt 7
      keyFile: /path/to/client.key 8
```

1. このプロバイダー名は返されるユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。
2. **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。
3. **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。
4. このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。
5. Basic 認証ヘッダーで認証情報を受け入れる URL。
6. オプション: 設定された URL のサーバー証明書を検証するために使用する証明書バンドルです。
7. オプション: 設定された URL に対して要求を実行する際に提示するクライアント証明書です。
8. クライアント証明書のキーです。**certFile** が指定されている場合は必須です。

以下の変更を **identityProviders** スタンザに加えます。

a. プロバイダーの **name** をデプロイメントに対して固有で関連するものに設定します。この名前は返されるユーザー ID にプレフィックスとして付加され、アイデンティティー名が作成されます。

b. 必要な場合、**mappingMethod** を設定して、プロバイダーのアイデンティティーとユーザーオブジェクト間でマッピングを確立する方法を制御します。

c. Basic 認証ヘッダーで認証情報を受け入れるサーバーへの接続に使用する HTTPS **url** を指定します。

d. オプションで、設定された URL のサーバー証明書を検証するために **ca** を使用する証明書バンドルに設定するか、またはこれを空にしてシステムで信頼されるルートを使用します。

e. オプションで、**certFile** を削除するか、またはこれを設定された URL へ要求を行う時に提示するクライアント証明書に設定します。

f. **certFile** を指定する場合、**keyFile** をクライアント証明書のキーに設定する必要があります。

3. 変更を保存してファイルを閉じます。

4. OpenShift Container Platform API サーバーを起動し、変更したばかりの設定ファイルを指定します。

```
$ openshift start master --config=<path/to/modified/config>/master-
config.yaml
```

これが設定されると、OpenShift Container Platform Web コンソールにログインするユーザーには、Basic 認証の認証情報を使用してログインすることを求めるプロンプトが表示されます。

11.3.9.2. トラブルシューティング

最もよく起こる問題は、バックエンドサーバーへのネットワーク接続に関連しています。簡単なデバッグの場合は、マスターで **curl** コマンドを実行します。正常なログインをテストするには、以下のコマンド例の **<user>** と **<password>** を有効な認証情報に置き換えます。無効なログインをテストするには、それらを正しくない認証情報に置き換えます。

```
curl --cacert /path/to/ca.crt --cert /path/to/client.crt --key
/path/to/client.key -u <user>:<password> -v
https://www.example.com/remote-idp
```

正常な応答

sub (サブジェクト) キーを持つ **200** ステータスは、成功を示しています。

```
{"sub":"userid"}
```

サブジェクトは認証ユーザーに固有である必要があり、変更することはできません。

正常な応答により、以下のような追加データがオプションで提供されることがあります。

- **name** キーを使用した表示名:

  ```
  {"sub":"userid", "name": "User Name", ...}
  ```

- **email** キーを使用したメールアドレス:

  ```
  {"sub":"userid", "email":"user@example.com", ...}
  ```

- **preferred_username** キーを使用した推奨ユーザー名:

  ```
  {"sub":"014fbff9a07c", "preferred_username":"bob", ...}
  ```

  preferred_username キーは、固有の変更できないサブジェクトがデータベースキーまたは UID であり、判読可能な名前が存在する場合に便利です。これは、認証 ID に対して OpenShift Container Platform ユーザーをプロビジョニングする場合にヒントとして使われます。

失敗の応答

- **401** 応答は認証の失敗を示しています。
- **200** 以外のステータスまたは空でない「エラー」キーはエラーを示しています: **{"error":"Error message"}**

11.3.10. 要求ヘッダー

identityProviders スタンザで **RequestHeaderIdentityProvider** を設定して、**X-Remote-User** などの要求ヘッダー値からユーザーを識別します。これは通常、プロキシー認証と組み合わせて使用され、要求ヘッダー値を設定します。これは OpenShift Enterprise 2 のリモートユーザープラグインに

よって管理者が Kerberos、LDAP、その他の数多くの形式のエンタープライズ認証を指定する方法と似ています。

さらに、SAML 認証などの詳細な設定に要求ヘッダーのアイデンティティープロバイダーを使用できます。

ユーザーがこのアイデンティティープロバイダーを使用して認証を行うには、認証プロキシー経由で **`https://<master>/oauth/authorize`** (およびサブパス) にアクセスする必要があります。これを実行するには、OAuth トークンに対する非認証の要求を **`https://<master>/oauth/authorize`** にプロキシー処理するプロキシーエンドポイントにリダイレクトするよう OAuth サーバーを設定します。

ブラウザーベースのログインフローが想定されるクライアントからの非認証要求をリダイレクトするには、以下を実行します。

1. **`login`** パラメーターを **true** に設定します。
2. **`provider.loginURL`** パラメーターをインタラクティブなクライアントを認証する認証プロキシー URL に設定してから、要求を **`https://<master>/oauth/authorize`** にプロキシーします。

`WWW-Authenticate` challenge が想定されるクライアントからの非認証要求をリダイレクトするには、以下を実行します。

1. **`challenge`** パラメーターを **true** に設定します。
2. **`provider.challengeURL`** パラメーターを **`WWW-Authenticate`** challenge が想定されるクライアントを認証する認証プロキシー URL に設定し、要求を **`https://<master>/oauth/authorize`** にプロキシーします。

`provider.challengeURL` および **`provider.loginURL`** パラメーターには、URL のクエリー部分に以下のトークンを含めることができます。

- **`${url}`** は現在の URL と置き換えられ、エスケープされてクエリーパラメーターで保護されます。
  例: **`https://www.example.com/sso-login?then=${url}`**
- **`${query}`** は最新のクエリー文字列と置き換えられ、エスケープされません。
  例: **`https://www.example.com/auth-proxy/oauth/authorize?${query}`**

警告

非認証要求が OAuth サーバーに到達することを想定する場合は、要求ヘッダーのユーザー名がチェックされる前に受信要求の有効なクライアント証明書をチェックするように、**`clientCA`** パラメーターをこのアイデンティティープロバイダーに対して設定する必要があります。これを設定しない場合、OAuth サーバーへの直接的な要求は、要求ヘッダーを設定するだけでこのプロバイダーのアイデンティティーになりすます可能性があります。

RequestHeaderIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: my_request_header_provider (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://www.example.com/challenging-
proxy/oauth/authorize?${query}" (5)
      loginURL: "https://www.example.com/login-proxy/oauth/authorize?
${query}" (6)
      clientCA: /path/to/client-ca.file (7)
      clientCommonNames: (8)
      - my-auth-proxy
      headers: (9)
      - X-Remote-User
      - SSO-User
      emailHeaders: (10)
      - X-Remote-User-Email
      nameHeaders: (11)
      - X-Remote-User-Display-Name
      preferredUsernameHeaders: (12)
      - X-Remote-User-Login
```

(1) このプロバイダー名は要求ヘッダーのユーザー名にプレフィックスとして付加され、アイデンティティー名が作成されます。

(2) **RequestHeaderIdentityProvider** は、設定された **challengeURL** にリダイレクトすることで、**WWW-Authenticate** challenge を要求するクライアントにのみ応答します。設定される URL は **WWW-Authenticate** challenge を使用して応答します。

(3) **RequestHeaderIdentityProvider** は、設定された **loginURL** にリダイレクトすることで、ログインフローを要求するクライアントにのみ応答できます。設定される URL はログインフローを使用して応答します。

(4) このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。

(5) オプション: 非認証の **/oauth/authorize** 要求のリダイレクト先となる URLです。これは、ブラウザーベースのクライアントを認証し、その要求を **https://<master>/oauth/authorize** にプロキシーします。**https://<master>/oauth/authorize** にプロキシーする URL は **/authorize** で終了し (末尾のスラッシュなし)、OAuth 承認フローが適切に機能するようサブパスもプロキシーします。**${url}** は現在の URL に置き換えられ、エスケープされてクエリーパラメーターで保護されます。**${query}** は現在のクエリー文字列に置き換えられます。

(6) オプション: 非認証の **/oauth/authorize** 要求のリダイレクト先となる URL です。**WWW-Authenticate** challenge が想定されるクライアントを認証し、それらを **https://<master>/oauth/authorize** にプロキシーします。**${url}** は現在の URL に置き換えられ、エスケープされてクエリーパラメーターで保護されます。**${query}** は現在のクエリー文字列に置き換えられます。

7 オプション: PEM でエンコードされた証明書バンドルです。これが設定されている場合、要求ヘッダーのユーザー名をチェックする前に、有効なクライアント証明書が提示され、指定ファイルで認証局に対して検証される必要があります。

8 オプション: 共通名 (**cn**) の一覧です。これが設定されている場合は、要求ヘッダーのユーザー名をチェックする前に指定される一覧の Common Name (**cn**) を持つ有効なクライアント証明書が提示される必要があります。空の場合、すべての Common Name が許可されます。これは **clientCA** との組み合わせる場合にのみ使用できます。

9 ユーザーアイデンティティーを順番にチェックする際に使用するヘッダー名です。値を含む最初のヘッダーはアイデンティティーとして使用されます。これは必須であり、大文字小文字を区別します。

10 メールアドレスを順番にチェックする際に使用するヘッダー名です。値を含む最初のヘッダーはメールアドレスとして使用されます。これは任意であり、大文字小文字を区別します。

11 表示名を順番にチェックする際に使用するヘッダー名です。値を含む最初のヘッダーは表示名として使用されます。これは任意であり、大文字小文字を区別します。

12 推奨ユーザー名を順番にチェックする際に使用するヘッダー名です (**headers** で指定されるヘッダーで決定される変更できないアイデンティティーと異なる場合)。値を含む最初のヘッダーは、プロビジョニング時に推奨ユーザー名として使用されます。これは任意であり、大文字小文字を区別します。

要求ヘッダーを使用した Apache 認証

この例は、マスターと同じホストに認証プロキシーを設定しています。同じホストにプロキシーとマスターがあると便利ですが、ご使用中の環境に適さない場合があります。たとえば、すでにマスターでルーターを実行している場合には、ポート 443 が利用できなくなります。

この参照設定は Apache の **mod_auth_form** を使用していますが、これは決して必須ではなく、以下の要件を満たしていれば他のプロキシーを簡単に使用することができます。

1. クライアント要求の **X-Remote-User** ヘッダーをブロックして、スプーフィングを防ぎます。
2. **RequestHeaderIdentityProvider** 設定でクライアント証明書認証を実施します。
3. challenge フローを使用してすべての認証要求についての **X-Csrf-Token** ヘッダーを設定する必要があります。
4. **/oauth/authorize** エンドポイントとそのサブパスのみがプロキシーされる必要があります。バックエンドサーバーがクライアントを正しい場所へ送信できるようリダイレクトは書き換えないでください。
5. **https://<master>/oauth/authorize** へプロキシーする URL は **/authorize** で終了 (末尾のスラッシュなし) する必要があります。以下は例になります。
   - **https://proxy.example.com/login-proxy/authorize?…** → **https://<master>/oauth/authorize?…**
6. **https://<master>/oauth/authorize** にプロキシーされる URL のサブパスは、**https://<master>/oauth/authorize** のサブパスにプロキシーする必要があります。以下は例になります。
   - **https://proxy.example.com/login-proxy/authorize/approve?…** → **https://<master>/oauth/authorize/approve?…**

前提条件のインストール

1. **mod_auth_form** モジュールは Optional チャンネル にある **mod_session** パッケージの一部として同梱されます。以下のパッケージをインストールします。

```
# yum install -y httpd mod_ssl mod_session apr-util-openssl
```

2. 信頼されたヘッダーを送信する要求を検証するために CA を生成します。この CA は マスターのアイデンティティープロバイダーの設定 の **clientCA** のファイル名として使用されます。

```
# oc adm ca create-signer-cert \
  --cert='/etc/origin/master/proxyca.crt' \
  --key='/etc/origin/master/proxyca.key' \
  --name='openshift-proxy-signer@1432232228' \
  --serial='/etc/origin/master/proxyca.serial.txt'
```

注記

oc adm ca create-signer-cert コマンドは、5 年間有効な証明書を生成します。この期間は **--expire-days** オプションを使って変更することができますが、セキュリティー上の理由から、値をこれ以上大きくすることは推奨されません。

Ansible ホストインベントリーファイル (デフォルトで **/etc/ansible/hosts**) に最初に一覧表示されているマスターから **oc adm** コマンドを実行します。

3. このプロキシー用のクライアント証明書を生成します。x509 証明書ツーリングを使用して実行することができます。**oc adm** CLI を使用すると便利です。

```
# oc adm create-api-client-config \
  --certificate-authority='/etc/origin/master/proxyca.crt' \
  --client-dir='/etc/origin/master/proxy' \
  --signer-cert='/etc/origin/master/proxyca.crt' \
  --signer-key='/etc/origin/master/proxyca.key' \
  --signer-serial='/etc/origin/master/proxyca.serial.txt' \
  --user='system:proxy' (1)

# pushd /etc/origin/master
# cp master.server.crt /etc/pki/tls/certs/localhost.crt (2)
# cp master.server.key /etc/pki/tls/private/localhost.key
# cp ca.crt /etc/pki/CA/certs/ca.crt
# cat proxy/system\:proxy.crt \
  proxy/system\:proxy.key > \
  /etc/pki/tls/certs/authproxy.pem
# popd
```

(1) ユーザー名は任意に指定できますが、ログにそのまま表示されるのでわかりやすい名前をつけると便利です。

(2) マスターと異なるホスト名で認証プロキシーを実行する場合、上記のようにデフォルトのマスター証明書を使用するのではなく、ホスト名と一致する証明書を生成することが重要です。**/etc/origin/master/master-config.yaml** ファイルの **masterPublicURL** の値は、**SSLCertificateFile** に対して指定される証明書の **X509v3 Subject Alternative Name** に含まれる必要があります。新規の証明書を作成する必要がある場

合は、**oc adm ca create-server-cert** コマンドを使用できます。

注記

oc adm create-api-client-config コマンドは、2 年間有効な証明書を生成します。この期間は **--expire-days** オプションを使って変更することができますが、セキュリティー上の理由から、値をこれ以上大きくすることは推奨されません。Ansible ホストインベントリーファイル (デフォルトは **/etc/ansible/hosts**) に一覧表示される 1 つ目のマスターからのみ **oc adm** コマンドを実行します。

Apache の設定

このプロキシーはマスターと同じホスト上に配置する必要はありません。プロキシーは、クライアント証明書を使用してマスターに接続し、**X-Remote-User** ヘッダーを信頼するように設定されます。

1. Apache 設定の証明書を作成します。**SSLProxyMachineCertificateFile** パラメーターの値として指定する証明書は、プロキシーをサーバーに対して認証するために使用されるプロキシーのクライアント証明書です。これは、拡張されたキーのタイプとして **TLS Web Client Authentication** を使用する必要があります。

2. 以下のそれぞれに対して Apache を設定します。

```
LoadModule auth_form_module modules/mod_auth_form.so
LoadModule session_module modules/mod_session.so
LoadModule request_module modules/mod_request.so

# Nothing needs to be served over HTTP.  This virtual host simply
redirects to
# HTTPS.
<VirtualHost *:80>
  DocumentRoot /var/www/html
  RewriteEngine              On
  RewriteRule     ^(.*)$     https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
  # This needs to match the certificates you generated.  See the CN and
X509v3
  # Subject Alternative Name in the output of:
  # openssl x509 -text -in /etc/pki/tls/certs/localhost.crt
  ServerName www.example.com

  DocumentRoot /var/www/html
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/localhost.crt
  SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
  SSLCACertificateFile /etc/pki/CA/certs/ca.crt

  SSLProxyEngine on
  SSLProxyCACertificateFile /etc/pki/CA/certs/ca.crt
  # It's critical to enforce client certificates on the Master.  Otherwise
  # requests could spoof the X-Remote-User header by accessing the
Master's
  # /oauth/authorize endpoint directly.
```

```
  SSLProxyMachineCertificateFile /etc/pki/tls/certs/authproxy.pem

  # Send all requests to the console
  RewriteEngine              On
  RewriteRule     ^/console(.*)$     https://%{HTTP_HOST}:8443/console$1
[R,L]

  # In order to using the challenging-proxy an X-Csrf-Token must be
present.
  RewriteCond %{REQUEST_URI} ^/challenging-proxy
  RewriteCond %{HTTP:X-Csrf-Token} ^$ [NC]
  RewriteRule ^.* - [F,L]

  <Location /challenging-proxy/oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://[MASTER]:8443/oauth/authorize
    AuthType basic
  </Location>

  <Location /login-proxy/oauth/authorize>
    # Insert your backend server name/ip here.
    ProxyPass https://[MASTER]:8443/oauth/authorize

    # mod_auth_form providers are implemented by mod_authn_dbm,
mod_authn_file,
    # mod_authn_dbd, mod_authnz_ldap and mod_authn_socache.
    AuthFormProvider file
    AuthType form
    AuthName openshift
    ErrorDocument 401 /login.html
  </Location>

  <ProxyMatch /oauth/authorize>
    AuthUserFile /etc/origin/master/htpasswd
    AuthName openshift
    Require valid-user
    RequestHeader set X-Remote-User %{REMOTE_USER}s env=REMOTE_USER

    # For ldap:
    # AuthBasicProvider ldap
    # AuthLDAPURL "ldap://ldap.example.com:389/ou=People,dc=my-
domain,dc=com?uid?sub?(objectClass=*)"

    # It's possible to remove the mod_auth_form usage and replace it with
    # something like mod_auth_kerb, mod_auth_gssapi or even
mod_auth_mellon.
    # The former would be able to support both the login and challenge
flows
    # from the Master.  Mellon would likely only support the login flow.

    # For Kerberos
    # yum install mod_auth_gssapi
    # AuthType GSSAPI
    # GssapiCredStore keytab:/etc/httpd.keytab
  </ProxyMatch>
```

```
</VirtualHost>

RequestHeader unset X-Remote-User
```

その他の mod_auth_form 要件
サンプルログインページは、openshift_extras リポジトリーから利用できます。このファイルは **DocumentRoot** (デフォルトでは /**var**/**www**/**html**) に配置する必要があります。

ユーザーの作成
ここで、Apache がアカウント情報を保存するために使用しているシステムでユーザーを作成することができます。たとえば、ファイルベース (file-backed) 認証が使用されます。

```
# yum -y install httpd-tools
# touch /etc/origin/master/htpasswd
# htpasswd /etc/origin/master/htpasswd <user_name>
```

マスターの設定
/etc/origin/master/master-config.yaml ファイルの **identityProviders** スタンザも更新する必要があります。

```
  identityProviders:
  - name: requestheader
    challenge: true
    login: true
    provider:
      apiVersion: v1
      kind: RequestHeaderIdentityProvider
      challengeURL: "https://[MASTER]/challenging-proxy/oauth/authorize?
${query}"
      loginURL: "https://[MASTER]/login-proxy/oauth/authorize?${query}"
      clientCA: /etc/origin/master/proxyca.crt
      headers:
      - X-Remote-User
```

サービスの再起動
最後に、以下のサービスを再起動します。

```
# systemctl restart httpd
# master-restart api
# master-restart controllers
```

設定の確認

1. プロキシーをバイパスしてテストします。正しいクライアント証明書とヘッダーを指定すれば、トークンを要求できます。

```
# curl -L -k -H "X-Remote-User: joe" \
   --cert /etc/pki/tls/certs/authproxy.pem \
   https://[MASTER]:8443/oauth/token/request
```

2. クライアント証明書を指定しない場合、要求は拒否されます。

```
# curl -L -k -H "X-Remote-User: joe" \
   https://[MASTER]:8443/oauth/token/request
```

3. これは、設定された **challengeURL** (追加のクエリーパラメーターを含む) へのリダイレクトを示します。

```
# curl -k -v -H 'X-Csrf-Token: 1' \
    '<masterPublicURL>/oauth/authorize?client_id=openshift-
challenging-client&response_type=token'
```

4. これは、**WWW-Authenticate** 基本 challenge の 401 応答を示します。

```
#  curl -k -v -H 'X-Csrf-Token: 1' \
     '<redirected challengeURL from step 3 +query>'
```

5. これはアクセストークンを使用するリダイレクトを示します。

```
#  curl -k -v -u <your_user>:<your_password> \
     -H 'X-Csrf-Token: 1' '<redirected_challengeURL_from_step_3
+query>'
```

11.3.11. GitHub

GitHub は OAuth を使用します。OpenShift Container Platform クラスターを統合して OAuth 認証を使用できます。OAuth は基本的にトークンの交換フローを促進します。

GitHub 認証を設定することによって、ユーザーは GitHub 認証情報を使用して OpenShift Container Platform にログインできます。GitHub ユーザー ID を持つすべてのユーザーが OpenShift Container Platform クラスターにログインできないようにするために、アクセスを特定の GitHub 組織のユーザーに制限することができます。

11.3.12. GitHub でのアプリケーションの登録

1. GitHub で Settings → OAuth applications → Developer applications → Register an application を順番にクリックし、new OAuth application のページに移動します。

2. アプリケーション名を入力します。例: **My OpenShift Install**

3. ホームページの URL を入力します。例: **https://myapiserver.com:8443**

4. オプションで、アプリケーションの説明を入力します。

5. 承認コールバック URL を入力します。ここで、URL の末尾にはアイデンティティープロバイダーの **名前** (マスター設定ファイル の **identityProviders** スタンザで定義されます。このトピックの以下のセクションで設定を行います) が含まれます。

```
<apiserver>/oauth2callback/<identityProviderName>
```

以下は例を示しています。

```
https://myapiserver.com:8443/oauth2callback/github/
```

6. **Register application** をクリックします。GitHub はクライアント ID とクライアントシークレットを提供します。このウィンドウを開いたままにして、それらの値をコピーし、マスター設定ファイルに貼り付けます。

11.3.13. マスターでの認証の設定

1. 状況に応じて以下のいずれかの手順を実行します。

   - Openshift のインストールがすでに完了している場合は、**/etc/origin/master/master-config.yaml** ファイルを新規ディレクトリーにコピーします。以下は例になります。

     ```
     $ cd /etc/origin/master
     $ mkdir githubconfig; cp master-config.yaml githubconfig
     ```

   - OpenShift Container Platform をまだインストールしていない場合は、OpenShift Container Platform API サーバーを起動し、(将来の) OpenShift Container Platform マスターのホスト名と、起動コマンドによって作成された設定ファイルを格納するディレクトリーを指定します。

     ```
     $ openshift start master --public-master=<apiserver> --write-
     config=<directory>
     ```

     以下は例を示しています。

     ```
     $ openshift start master --public-
     master=https://myapiserver.com:8443 --write-config=githubconfig
     ```

     注記

     Ansible を使用してインストールする場合は、**`identityProvider`** 設定を Ansible Playbook に追加する必要があります。Ansible を使用してインストールした後、以下の手順に従って設定を手動で変更した場合、インストールツールまたはアップグレードを再実行するたびに変更内容がすべて失われます。

     注記

     `openshift start master` を単独で使用するとホスト名が自動的に検出されますが、GitHub はアプリケーション登録時に指定した正確なホスト名にリダイレクトできなければなりません。このような理由から、間違ったアドレスにリダイレクトされる可能性があるため、ID の自動検出は使用できません。代わりに、Web ブラウザーが OpenShift Container Platform クラスターとの対話に使用するホスト名を指定する必要があります。

2. 新規 **master-config.yaml** ファイルの **`identityProviders`** スタンザを編集し、**`GitHubIdentityProvider`** 設定例をコピーして貼り付け、既存のスタンザを置き換えます。

   ```
   oauthConfig:
     ...
     identityProviders:
     - name: github 1
       challenge: false 2
       login: true 3
       mappingMethod: claim 4
       provider:
         apiVersion: v1
   ```

```
      kind: GitHubIdentityProvider
      clientID: ... 5
      clientSecret: ... 6
      organizations: 7
      - myorganization1
      - myorganization2
      teams: 8
      - myorganization1/team-a
      - myorganization2/team-b
```

1. このプロバイダー名は GitHub の数字ユーザー ID にプレフィックスとして付加され、アイデンティティー名が作成されます。これはコールバック URL を作成するためにも使用されます。

2. **GitHubIdentityProvider** を使用して**`WWW-Authenticate`** challenge を送信することはできません。

3. **true** の場合、(Web コンソールの場合のように) Web クライアントからの非認証トークン要求はログインする GitHub にリダイレクトされます。

4. このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。

5. 登録済みの GitHub OAuth アプリケーションのクライアント ID です。アプリケーションは、**`<master>/oauth2callback/<identityProviderName>`** のコールバック URL を使用して設定する必要があります。

6. GitHub で発行されるクライアントシークレットです。この値は環境変数、外部ファイル、または暗号化されたファイルでも指定できます。

7. 組織のオプションの一覧です。これが指定されている場合、少なくとも一覧のいずれかの組織のメンバーである GitHub ユーザーのみがログインできます。その組織が **clientID** で設定された GitHub OAuth アプリケーションを所有していない場合、組織の所有者はこのオプションを使用するためにサードパーティーのアクセスを付与する必要があります。これは組織の管理者が初回の GitHub ログイン時に、または GitHub の組織設定で実行できます。これは **`teams`** フィールドと組み合わせて使用することはできません。

8. チームのオプションの一覧です。これが指定されている場合、少なくとも一覧のいずれかのチームのメンバーである GitHub ユーザーのみがログインできます。そのチームの組織が **clientID** で設定された GitHub OAuth アプリケーションを所有していない場合、組織の所有者はこのオプションを使用するためにサードパーティーのアクセスを付与する必要があります。これは組織の管理者が初回の GitHub ログイン時に、または GitHub の組織設定から実行できます。これは **`organizations`** フィールドと組み合わせて使用することはできません。

3. 以下の変更を **`identityProviders`** スタンザに加えます。

   a. プロバイダーの **`name`** を変更して、GitHub で設定したコールバック URL に一致させます。たとえば、コールバック URL を **`https://myapiserver.com:8443/oauth2callback/github/`** として定義した場合、**`name`** は **`github`** にする必要があります。

   b. **`clientID`** を以前に登録した GitHub のクライアント ID に変更します。

   c. **`clientSecret`** を以前に登録した GitHub のクライアントシークレットに変更します。

d. **organizations** または **teams** を変更して、ユーザーが認証を行うためにメンバーシップを設定している必要がある 1 つ以上の GitHub 組織またはチームの一覧を組み込むようにします。これが指定されている場合、少なくとも一覧のいずれかの組織またはチームのメンバーである GitHub ユーザーのみがログインできます。これが指定されていない場合、有効な GitHub アカウントを持つすべてのユーザーがログインできます。

4. 変更を保存してファイルを閉じます。

5. OpenShift Container Platform API サーバーを起動し、変更したばかりの設定ファイルを指定します。

```
$ openshift start master --config=<path/to/modified/config>/master-
config.yaml
```

これが設定されると、OpenShift Container Platform の Web コンソールにログインするユーザーには GitHub の認証情報を使用してログインすることを求めるプロンプトが出されます。初回ログイン時に、ユーザーは **authorize application** をクリックして GitHub が OpenShift Container Platform でのユーザー名、パスワードおよび組織のメンバーシップを使用することを許可する必要があります。その後、ユーザーは Web コンソールにリダイレクトされます。

11.3.14. GitHub 認証を持つユーザーの作成

GitHub などの外部認証プロバイダーを統合する場合は、ユーザーを OpenShift Container Platform では作成しません。GitHub は「system of record」であり、ユーザーは GitHub で定義され、指定される組織に属するすべてのユーザーがログインできることになります。

ユーザーを OpenShift Container Platform に追加するには、そのユーザーを GitHub で承認された組織に追加する必要があり、必要な場合は、そのユーザーの新しい GitHub アカウントを作成します。

11.3.15. ユーザーの確認

1 名以上のユーザーがログインしたら、**oc get users** を実行してユーザーの一覧を表示し、ユーザーが正しく作成されていることを確認できます。

例11.6 oc get users コマンドの出力

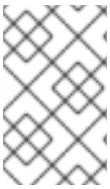
```
$ oc get users
NAME         UID                                    FULL NAME
IDENTITIES
bobsmith     433b5641-066f-11e6-a6d8-acfc32c1ca87   Bob Smith
github:873654 (1)
```

(1) OpenShift Container Platform のアイデンティティーは、アイデンティティープロバイダー名と GitHub の内部の数字のユーザー ID で構成されます。そのため、ユーザーが GitHub のユーザー名またはメールアドレスを変更した場合でも、GitHub アカウントに割り当てられる認証情報に依存せず、OpenShift Container Platform にログインできます。これにより安定したログインが作成されます。

ここからは、ユーザーロールを制御する方法を学習することをお勧めします。

11.3.16. GitLab

OAuth 統合 を使用して、**identityProviders** スタンザで **GitLabIdentityProvider** を設定し、GitLab.com またはその他の GitLab インスタンスをアイデンティティープロバイダーとして使用するようにします。OAuth プロバイダー機能には GitLab バージョン 7.7.0 以降が必要です。



注記

GitLab をアイデンティティープロバイダーとして使用するには、**<master>/oauth/token/request** を使用してトークンを取得し、コマンドラインツールを使用する必要があります。

例11.7 GitLabIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: gitlab 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: GitLabIdentityProvider
      url: ... 5
      clientID: ... 6
      clientSecret: ... 7
      ca: ... 8
```

1. このプロバイダー名は GitLab 数字ユーザー ID にプレフィックスとして付加され、アイデンティティー名が作成されます。これはコールバック URL を作成するためにも使用されます。
2. **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。これは Resource Owner Password Credentials 付与フローを使用して GitLab からアクセストークンを取得します。
3. **true** の場合、Web クライアント (Web コンソールなど) からの非認証トークン要求はログインする GitLab にリダイレクトされます。
4. このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。
5. GitLab OAuth プロバイダーのホスト URL です。これは **https://gitlab.com/** か、または他の GitLab のセルフホストインスタンスのいずれかになります。
6. 登録済みの GitLab OAuth アプリケーションのクライアント ID です。アプリケーションは**<master>/oauth2callback/<identityProviderName>** のコールバック URL で設定する必要があります。
7. GitLab で発行されるクライアントシークレットです。この値は環境変数、外部ファイル、または暗号化されたファイルでも指定できます。
8. CA は、GitLab インスタンスへの要求を行う際に使用する任意の信頼される認証局バンドルです。空の場合、デフォルトのシステムルートが使用されます。

11.3.17. Google

Google の OpenID Connect 統合を使用して、**identityProviders** スタンザで **GoogleIdentityProvider** を設定し、アイデンティティープロバイダーとして Google を使用するようにします。

注記

Google をアイデンティティープロバイダーとして使用するには、**<master>/oauth/token/request** を使用してトークンを取得し、コマンドラインツールで使用する必要があります。

警告

Google をアイデンティティープロバイダーとして使用することで、Google ユーザーはサーバーに対して認証されます。以下のように **hostedDomain** 設定属性を使用して、特定のホストドメインのメンバーに認証を限定することができます。

例11.8 GoogleIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: google (1)
    challenge: false (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: GoogleIdentityProvider
      clientID: ... (5)
      clientSecret: ... (6)
      hostedDomain: "" (7)
```

1. このプロバイダー名は Google の数字のユーザー ID にプレフィックスとして付加され、アイデンティティー名が作成されます。これはリダイレクト URL を作成するためにも使用されます。
2. **GoogleIdentityProvider** を使用して **WWW-Authenticate** challenge を送信することはできません。
3. **true** の場合、Web クライアント (Web コンソールなど) からの非認証トークン要求はログインする Google へリダイレクトされます。
4. このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。

5 登録済みの Google プロジェクト のクライアント ID です。プロジェクトは、**<master>/oauth2callback/<identityProviderName>** のリダイレクト URI で設定する必要があります。

6 Google で発行されるクライアントシークレットです。この値は環境変数、外部ファイル、または暗号化されたファイルでも指定できます。

7 サインインアカウントを制限するオプションの ホストレベルのドメイン です。空の場合、すべての Google アカウントの認証が許可されます。

11.3.18. OpenID Connect

identityProviders スタンザで **OpenIDIdentityProvider** を設定し、Authorization Code Flow を使用して OpenID Connect アイデンティティープロバイダーと統合します。

OpenShift Container Platform の OpenID Connect アイデンティティープロバイダーとして Red Hat シングルサインオンを設定できます。

注記

ID Token および **UserInfo** の復号化はサポートされていません。

デフォルトで、**openid** の範囲が要求されます。必要な場合は、**extraScopes** フィールドで追加の範囲を指定できます。

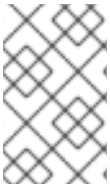
要求は、OpenID アイデンティティープロバイダーから返される JWT **id_token** から読み取られ、指定される場合は **UserInfo** URL によって返される JSON から読み取られます。

1 つ以上の要求をユーザーのアイデンティティーとして使用するように設定される必要があります。標準のアイデンティティー要求は **sub** になります。

また、どの要求をユーザーの推奨ユーザー名、表示名およびメールアドレスとして使用するか指定することもできます。複数の要求が指定されている場合、値が入力されている最初の要求が使用されます。標準の要求は以下の通りです。

sub	「subject identifier」の省略形です。発行側のユーザーのリモートアイデンティティーです。
preferred_username	ユーザーのプロビジョニング時に優先されるユーザー名です。**janedoe** などのユーザーを参照する際に使用する省略形の名前です。通常は、ユーザー名またはメールなどの、認証システムのユーザーのログインまたはユーザー名に対応する値です。
email	メールアドレス。
name	表示名。

詳細は、OpenID claim のドキュメントを参照してください。



注記

OpenID Connect アイデンティティープロバイダーを使用するには、**<master>/oauth/token/request** を使用してトークンを取得し、コマンドラインツールで使用する必要があります。

OpenIDIdentityProvider を使用する標準マスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect (1)
    challenge: true (2)
    login: true (3)
    mappingMethod: claim (4)
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ... (5)
      clientSecret: ... (6)
      claims:
        id: (7)
        - sub
        preferredUsername:
        - preferred_username
        name:
        - name
        email:
        - email
      urls:
        authorize: https://myidp.example.com/oauth2/authorize (8)
        token: https://myidp.example.com/oauth2/token (9)
```

(1) このプロバイダー名はアイデンティティー要求の値にプレフィックスとして付加され、アイデンティティー名が作成されます。これは、リダイレクト URL を作成するためにも使用されます。

(2) **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。この場合、OpenID プロバイダーが Resource Owner Password Credentials 付与フローをサポートしている必要があります。

(3) **true** の場合、Web クライアント (Web コンソールなど) からの非認証トークン要求は、ログインする認証 URL にリダイレクトされます。

(4) このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。

(5) OpenID プロバイダーに登録されているクライアントのクライアント ID です。このクライアントは **<master>/oauth2callback/<identityProviderName>** にリダイレクトすることを許可されている必要があります。

(6) クライアントシークレットです。この値は環境変数、外部ファイル、または暗号化されたファイルでも指定できます。

7 アイデンティティーとして使用する要求の一覧です。最初に空でない要求が使用されます。1 つ以上の要求が必要になります。一覧表示される要求のいずれにも値がないと、認証は失敗します。たとえば、ユーザーのアイデンティティーとして返される **id_token** の **sub** 要求の値が使用されます。

8 OpenID 仕様に記述される承認エンドポイントです。**https** を使用する必要があります。

9 OpenID 仕様に記述されるトークンエンドポイントです。**https**を使用する必要があります。

カスタム証明書バンドル、追加の範囲、追加の承認要求パラメーター、および **userInfo** URL も指定できます。

例11.9 OpenIDIdentityProvider を使用する完全なマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect
    challenge: false
    login: true
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ...
      clientSecret: ...
      ca: my-openid-ca-bundle.crt (1)
      extraScopes: (2)
      - email
      - profile
      extraAuthorizeParameters: (3)
        include_granted_scopes: "true"
      claims:
        id: (4)
        - custom_id_claim
        - sub
        preferredUsername: (5)
        - preferred_username
        - email
        name: (6)
        - nickname
        - given_name
        - name
        email: (7)
        - custom_email_claim
        - email
      urls:
        authorize: https://myidp.example.com/oauth2/authorize
        token: https://myidp.example.com/oauth2/token
        userInfo: https://myidp.example.com/oauth2/userinfo (8)
```

1 設定される URL のサーバー証明書を検証するために使用する証明書バンドルです。空の場合、システムで信頼されるルートを使用します。

2 承認トークン要求時に **openid** の範囲のほかに要求する範囲のオプションの一覧です。

3 承認トークン要求に追加する追加パラメーターのオプションのマップです。

4 アイデンティティーとして使用する要求の一覧です。空でない最初の要求が使用されます。1 つ以上の要求が必要になります。一覧表示される要求のいずれにも値がないと、認証は失敗します。

5 このアイデンティティーのユーザーをプロビジョニングする際に推奨ユーザー名として使用される要求の一覧です。空でない最初の要求が使用されます。

6 表示名として使用する要求の一覧です。空でない最初の要求が使用されます。

7 メールアドレスとして使用する要求の一覧です。空でない最初の要求が使用されます。

8 OpenID 仕様に記述される UserInfo エンドポイントです。**https** を使用する必要があります。

11.4. トークンオプション

OAuth サーバーは以下の 2 種類のトークンを生成します。

アクセストークン	API へのアクセスを付与する永続的なトークン。
認証コード	アクセストークンの交換にのみ使われる一時的なトークン。

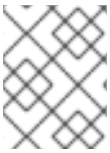
tokenConfig スタンザを使用してトークンオプションを設定します。

例11.10 マスター設定のトークンオプション

```
oauthConfig:
  ...
  tokenConfig:
    accessTokenMaxAgeSeconds: 86400 1
    authorizeTokenMaxAgeSeconds: 300 2
```

1 **accessTokenMaxAgeSeconds** を設定して、アクセストークンの有効期間を制御します。デフォルトの期間は 24 時間です。

2 **authorizeTokenMaxAgeSeconds** を設定して、認証コードの有効期間を制御します。デフォルトの期間は 5 分です。



注記

OAuthClient オブジェクト定義 により **accessTokenMaxAgeSeconds** 値を上書きできます。

11.5. 付与オプション

OAuth サーバーが、ユーザーが以前にパーミッションを付与していないクライアントに対するトークン要求を受信する場合、OAuth サーバーが実行するアクションは OAuth クライアントの付与ストラテジーによって変わります。

トークンを要求する OAuth クライアントが独自の付与ストラテジーを提供しない場合、サーバー全体でのデフォルトストラテジーが使用されます。デフォルトストラテジーを設定するには、**grantConfig** スタンザで **method** 値を設定します。**method** の有効な値は以下の通りです。

auto	付与を自動承認し、要求を再試行します。
prompt	ユーザーに対して付与の承認または拒否を求めるプロンプトを出します。
deny	付与を自動的に拒否し、失敗エラーをクライアントに返します。

例11.11 マスター設定の付与オプション

```
oauthConfig:
  ...
  grantConfig:
    method: auto
```

11.6. セッションオプション

OAuth サーバーは、ログインおよびリダイレクトフローで署名および暗号化される Cookie ベースセッションを使用します。

sessionConfig スタンザを使用してセッションオプションを設定します。

例11.12 マスター設定のセッションオプション

```
oauthConfig:
  ...
  sessionConfig:
    sessionMaxAgeSeconds: 300 (1)
    sessionName: ssn (2)
    sessionSecretsFile: "..." (3)
```

1. セッションの最大期間を制御します。トークン要求が完了すると、セッションは自動的に期限切れとなります。auto-grant が有効にされていない場合、ユーザーがクライアント承認要求を承認または拒否するためにかかると想定される時間の間、セッションは継続する必要があります。
2. セッションを保存するために使用される Cookie の名前です。
3. シリアライズされた **SessionSecrets** オブジェクトを含むファイル名です。空の場合、サーバーが起動されるたびにランダムの署名および暗号化シークレットが生成されます。

sessionSecretsFile が指定されていない場合、マスターサーバーが起動されるたびにランダムの署名および暗号化シークレットが生成されます。つまりマスターが再起動されると、進行中のログインではセッションが無効になります。また、これは他のマスターのいずれかによって生成されるセッションを復号化することはできないことも意味します。

使用する署名および暗号化シークレットを指定するには、**sessionSecretsFile** を指定します。これにより、シークレット値と設定ファイルを分離でき、たとえば、設定ファイルをデバッグなどの目的に合わせて配布可能な状態とすることができます。

複数のシークレットを **sessionSecretsFile** に指定してローテーションを有効にできます。一覧の最初のシークレットを使用して、新しいセッションに署名し、これを暗号化します。既存のセッションは、成功するまで各シークレットによって復号化され、認証されます。

例11.13 セッションシークレット設定:

```
apiVersion: v1
kind: SessionSecrets
secrets: (1)
- authentication: "..." (2)
  encryption: "..." (3)
- authentication: "..."
  encryption: "..."
...
```

(1) Cookie セッションの認証と暗号化に使用するシークレットの一覧です。シークレットを 1 つ以上指定する必要があります。各シークレットでは認証および暗号化シークレットを設定する必要があります。

(2) 署名シークレットです。HMAC を使用してセッションを認証するために使用されます。32 または 64 バイトでシークレットを使用することを推奨しています。

(3) 暗号化シークレットです。セッションを暗号化するために使用されます。16、24、または 32 文字で、AES-128、AES-192、または AES-256 を選択するために使用されます。

11.7. ユーザーエージェントによる CLI バージョンの不一致の防止

OpenShift Container Platform は、アプリケーション開発者の CLI が OpenShift Container Platform API にアクセスすることを防止するために使用されるユーザーエージェントを実装しています。

OpenShift Container Platform CLI のユーザーエージェントは、OpenShift Container Platform 内の値のセットで構成されています。

```
<command>/<version> (<platform>/<architecture>) <client>/<git_commit>
```

たとえば、以下の場合を考慮しましょう。

- <command> = **oc**
- <version> = クライアントのバージョン。たとえば、**v3.3.0**。**/api** で Kubernetes API に対して行われる要求が Kubernetes のバージョンを受信し、**/oapi** で OpenShift Container Platform API に対して行われる要求が OpenShift Container Platform のバージョン (**oc version** によって指定される) を受信します。

- <platform> = **linux**
- <architecture> = **amd64**
- <client> = **openshift** または **kubernetes**。要求が **/api** で Kubernetes API に対して行われるか、**/oapi** で OpenShift Container Platform API に対して行われるかによって決まります。
- <git_commit> = クライアントバージョンの Git コミット (例: **f034127**)

上記の場合、ユーザーエージェントは以下のようになります。

```
oc/v3.3.0 (linux/amd64) openshift/f034127
```

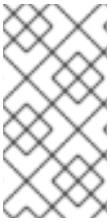
OpenShift Container Platform 管理者として、マスター設定の **userAgentMatching** 設定を使用してクライアントが API にアクセスできないようにすることができます。そのため、クライアントが特定のライブラリーまたはバイナリーを使用している場合、クライアントは API にアクセスできなくなります。

以下のユーザーエージェントの例は、Kubernetes 1.2 クライアントバイナリー、OpenShift Origin 1.1.3 バイナリー、POST および PUT **httpVerbs** を拒否します。

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old.  Go to
https://example.org to update it."
    deniedClients:
    - regex: '\w+/v(?:(?:1\.1\.1)|(?:1\.0\.1)) \(.+/.+\) openshift/\w{7}'
    - regex: '\w+/v(?:1\.1\.3) \(.+/.+\) openshift/\w{7}'
      httpVerbs:
      - POST
      - PUT
    - regex: '\w+/v1\.2\.0 \(.+/.+\) kubernetes/\w{7}'
      httpVerbs:
      - POST
      - PUT
    requiredClients: null
```

管理者は、予想されるクライアントに正確に一致しないクライアントを拒否することもできます。

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old.  Go to
https://example.org to update it."
    deniedClients: []
    requiredClients:
    - regex: '\w+/v1\.1\.3 \(.+/.+\) openshift/\w{7}'
    - regex: '\w+/v1\.2\.0 \(.+/.+\) kubernetes/\w{7}'
      httpVerbs:
      - POST
      - PUT
```



注記

クライアントのユーザーエージェントが設定と一致しない場合にエラーが発生します。変更する要求が一致するように、ホワイトリストを実施します。ルールは特定の verb にマップされるので、変化する要求を禁止し、変化しない要求を許可することができます。

第12章 グループと LDAP の同期

12.1. 概要

OpenShift Container Platform 管理者として、グループを使用してユーザーを管理し、権限を変更し、連携を強化できます。組織ではユーザーグループをすでに作成し、それらを LDAP サーバーに保存している場合があります。OpenShift Container Platform はそれらの LDAP レコードを内部 OpenShift Container Platform レコードと同期できるので、グループを 1 つの場所で管理できます。現時点で OpenShift Container Platform はグループメンバーシップを定義するための 3 つの共通スキーマ (RFC 2307、Active Directory、拡張された Active Directory) を使用してグループと LDAP サーバーの同期をサポートしています。

注記

グループを同期するには **cluster-admin** 権限を持っている必要があります。

12.2. LDAP 同期の設定

LDAP 同期を実行するには、同期設定ファイルが必要です。このファイルには LDAP クライアント設定の詳細が含まれます。

- LDAP サーバーへの接続の設定。
- LDAP サーバーで使用されるスキーマに依存する同期設定オプション。

同期設定ファイルには、OpenShift Container Platform Group 名を LDAP サーバーのグループにマップする管理者が定義した名前マッピングの一覧も含まれます。

12.2.1. LDAP クライアント設定

LDAP クライアント設定

```
url: ldap://10.0.0.0:389 (1)
bindDN: cn=admin,dc=example,dc=com (2)
bindPassword: password (3)
insecure: false (4)
ca: my-ldap-ca-bundle.crt (5)
```

(1) データベースをホストする LDAP サーバーの接続プロトコル、 IP アドレス、および **scheme://host:port** としてフォーマットされる接続先のポートです。

(2) バインド DN として使用する任意の識別名 (DN) です。同期操作のエントリーを取得するために昇格した権限が必要となる場合、OpenShift Container Platform はこれを使用します。

(3) バインドに使用する任意のパスワードです。同期操作のエントリーを取得するために昇格した権限が必要となる場合、OpenShift Container Platform はこれを使用します。この値は環境変数、外部ファイル、または暗号化ファイルでも指定できます。

(4) **true** の場合、サーバーへの TLS 接続は行われません。**false** の場合、セキュアな LDAP (**ldaps://**) URL は TLS を使用して接続し、非セキュアな LDAP (**ldap://**) URL は TLS にアップグレードされます。

5 設定された URL のサーバー証明書を検証するために使用する証明書バンドルです。空の場合、OpenShift Container Platform はシステムで信頼されるルートを使用します。**insecure** が **false** に設定されている場合にのみ、これが適用されます。

12.2.2. LDAP クエリー定義

同期設定は、同期に必要となるエントリーの LDAP クエリー定義で構成されています。LDAP クエリーの特定の定義は、LDAP サーバーにメンバーシップ情報を保存するために使用されるスキーマに依存します。

LDAP クエリー定義

```
baseDN: ou=users,dc=example,dc=com 1
scope: sub 2
derefAliases: never 3
timeout: 0 4
filter: (objectClass=inetOrgPerson) 5
pageSize: 0 6
```

1 すべての検索が開始されるディレクトリーのブランチの識別名 (DN) です。ディレクトリーツリーの上部を指定する必要がありますが、ディレクトリーのサブツリーを指定することもできます。

2 検索の範囲です。有効な値は **base**、**one**、または **sub** です。これを定義しない場合、**sub** の範囲が使用されます。範囲オプションについては、以下の表で説明されています。

3 LDAP ツリーのエイリアスに関連する検索の動作です。有効な値は **never**、**search**、**base**、または **always** です。これを定義しない場合、デフォルトは **always** となり、エイリアスを逆参照します。逆参照の動作については以下の表で説明されています。

4 クライアントによって検索に許可される時間制限です。秒単位で表示されます。0 の値はクライアント側の制限がないことを意味します。

5 有効な LDAP 検索フィルターです。これを定義しない場合、デフォルトは **(objectClass=*)** になります。

6 サーバーからの応答ページの任意の最大サイズです。これは、LDAP エントリーで測定されます。0 に設定すると、応答ページのサイズ制限はなくなります。クエリーで、クライアントまたはサーバーがデフォルトで許可しているエントリー数より多いエントリー数が返される場合には、ページングサイズの設定が必要となります。

表12.1 LDAP 検索範囲オプション

LDAP 検索範囲	説明
base	クエリーに対して指定されるベース DN で指定するオブジェクトのみを考慮します。
one	クエリーについてベース DN とツリー内の同じレベルにあるすべてのオブジェクト考慮します。
sub	クエリーに指定されるベース DN のサブツリー全体を考慮します。

表12.2 LDAP 逆参照動作

逆参照動作	説明
never	LDAP ツリーにあるエイリアスを逆参照しません。
search	検索中に見つかったエイリアスのみを逆参照します。
base	ベースオブジェクトを検索中にエイリアスのみを逆参照ます。
always	LDAP ツリーにあるすべてのエイリアスを常に逆参照します。

12.2.3. ユーザー定義の名前マッピング

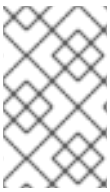
ユーザー定義の名前マッピングは、OpenShift Container Platform Groups の名前を LDAP サーバーでグループを検出する固有の識別子に明示的にマップします。マッピングは通常の YAML 構文を使用します。ユーザー定義のマッピングには LDAP サーバーのすべてのグループのエントリーを含めることも、それらのグループのサブセットのみを含めることもできます。ユーザー定義の名前マッピングを持たないグループが LDAP サーバーにある場合、同期時のデフォルト動作では OpenShift Container Platform Group の名前として指定される属性が使用されます。

ユーザー定義の名前マッピング

```
groupUIDNameMapping:
  "cn=group1,ou=groups,dc=example,dc=com": firstgroup
  "cn=group2,ou=groups,dc=example,dc=com": secondgroup
  "cn=group3,ou=groups,dc=example,dc=com": thirdgroup
```

12.3. LDAP 同期の実行

同期設定ファイルを作成すると、同期を開始できます。OpenShift Container Platform では、管理者は同じサーバーを使用して多数の異なる同期タイプを実行できます。



注記

デフォルトでは、すべてのグループ同期またはプルーニング操作がドライランされるので、OpenShift Container Platform Group レコードを変更するために **sync-groups** コマンドで **--confirm** フラグを設定する必要があります。

OpenShift Container Platform を使用して LDAP サーバーからすべてのグループを同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=config.yaml --confirm
```

設定ファイルで指定された LDAP サーバーのグループに対応する OpenShift Container Platform の Group をすべて同期するには、以下を実行します。

```
$ oc adm groups sync --type=openshift --sync-config=config.yaml --confirm
```

LDAP グループのサブセットと OpenShift Container Platform を同期するには、ホワイトリストファイル、ブラックリストファイル、またはその両方を使用します。

注記

ブラックリストファイル、ホワイトリストファイル、またはホワイトリストのリテラルの組み合わせを使用できます。ホワイトリストおよびブラックリストのファイルには 1 行ごとに 1 つの固有のグループ識別子を含める必要があり、ホワイトリストのリテラルはコマンド自体に直接含めることができます。これらのガイドラインは LDAP サーバーにあるグループと OpenShift Container Platform にすでにあるグループに適用されます。

```
$ oc adm groups sync --whitelist=<whitelist_file> \
                     --sync-config=config.yaml    \
                     --confirm
$ oc adm groups sync --blacklist=<blacklist_file> \
                     --sync-config=config.yaml    \
                     --confirm
$ oc adm groups sync <group_unique_identifier>    \
                     --sync-config=config.yaml    \
                     --confirm
$ oc adm groups sync <group_unique_identifier>    \
                     --whitelist=<whitelist_file> \
                     --blacklist=<blacklist_file> \
                     --sync-config=config.yaml    \
                     --confirm
$ oc adm groups sync --type=openshift             \
                     --whitelist=<whitelist_file> \
                     --sync-config=config.yaml    \
                     --confirm
```

12.4. グループのプルーニングジョブの実行

グループを作成した LDAP サーバーのレコードが存在しなくなった場合、管理者は OpenShift Container Platform レコードからグループを削除することを選択できます。プルーニングジョブは、同期ジョブに使用されるものと同じ同期設定ファイルとホワイトまたはブラックリストを受け入れます。詳細は、「グループのプルーニング」セクションを参照してください。

12.5. 同期の例

このセクションでは、RFC 2307、Active Directory と 拡張された Active Directory スキーマの例を紹介しています。以下のすべての例では 2 名のメンバー (**Jane** と **Jim**) が所属する **admins** というグループを同期します。それぞれの例では以下について説明しています。

- グループとユーザーが LDAP サーバーに追加される方法。
- LDAP 同期設定ファイルの概観。
- 同期後に生成される OpenShift Container Platform の Group レコード。

注記

これらの例では、すべてのユーザーがそれぞれのグループの直接的なメンバーであることを想定しています。特に、グループには他のグループがメンバーとして含まれません。ネスト化されたグループを同期する方法の詳細については、「ネスト化されたメンバーシップ同期の例」を参照してください。

12.5.1. RFC 2307 スキーマの使用によるグループの同期

RFC 2307 スキーマでは、ユーザー (Jane と Jim) とグループの両方がファーストクラスエントリーとして LDAP サーバーに存在し、グループメンバーシップはグループの属性に保存されます。以下の `ldif` のスニペットでは、このスキーマのユーザーとグループを定義しています。

RFC 2307 スキーマを使用する LDAP エントリー: rfc2307.ldif

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com (1)
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com (2)
member: cn=Jim,ou=users,dc=example,dc=com
```

(1) このグループは LDAP サーバーのファーストクラスエントリーです。

(2) グループのメンバーは、グループの属性としての識別参照と共に一覧表示されます。

このグループを同期するには、まず設定ファイルを作成する必要があります。RFC 2307 スキーマでは、ユーザーとグループエントリー両方の LDAP クエリー定義と内部 OpenShift Container Platform レコードでそれらを表すのに使用する属性を指定する必要があります。

明確にするために、OpenShift Container Platform で作成するグループは (可能な場合) ユーザーまたは管理者に表示されるフィールドに識別名以外の属性を使用する必要があります。たとえば、メールによって OpenShift Container Platform グループのユーザーを識別し、一般名としてグループの名前を使用します。以下の設定ファイルではこのような関係を作成しています。

注記

ユーザー定義の名前マッピングを使用する場合は、設定ファイルが異なります。

RFC 2307 スキーマを使用する LDAP 同期設定: rfc2307_config.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389 (1)
insecure: false (2)
rfc2307:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn (3)
    groupNameAttributes: [ cn ] (4)
    groupMembershipAttributes: [ member ] (5)
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    userUIDAttribute: dn (6)
    userNameAttributes: [ mail ] (7)
    tolerateMemberNotFoundErrors: false
    tolerateMemberOutOfScopeErrors: false
```

(1) このグループのレコードが保存される LDAP サーバーの IP アドレスとホストです。

(2) **true** の場合、サーバーへの TLS 接続は行われません。**false** の場合、セキュアな LDAP (**ldaps://**) URL は TLS を使用して接続し、非セキュアな LDAP (**ldap://**) URL は TLS にアップグレードされます。

(3) LDAP サーバーのグループを一意に識別する属性です。groupUIDAttribute に DN を使用している場合、**groupsQuery** フィルターを指定できません。詳細なフィルターを実行するには、ホワイトリスト/ブラックリスト方法を使用します。

(4) Group の名前として使用する属性です。

(5) メンバーシップ情報を保存するグループの属性です。

(6)

LDAP サーバーでユーザーを一意に識別する属性です。userUIDAttribute に DN を使用している場合は、**usersQuery** フィルターを指定できません。詳細のフィルターを実行するには、ホワイト

7 OpenShift Container Platform Group レコードでユーザー名として使用される属性です。

rfc2307_config.yaml ファイルと同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=rfc2307_config.yaml --confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

rfc2307_config.yaml ファイルを使用して作成される OpenShift Container Platform Group

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 1
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com 2
    openshift.io/ldap.url: LDAP_SERVER_IP:389 3
  creationTimestamp:
  name: admins 4
users: 5
- jane.smith@example.com
- jim.adams@example.com
```

1 この OpenShift Container Platform グループと LDAP サーバーが最後に同期された時間です。ISO 6801 形式を使用します。

2 LDAP サーバーのグループの固有識別子です。

3 このグループのレコードが保存される LDAP サーバーの IP アドレスとホストです。

4 同期ファイルが指定するグループ名です。

5 グループのメンバーのユーザーです。同期ファイルで指定される名前が使用されます。

12.5.1.1. ユーザー定義の名前マッピングに関する RFC2307

グループとユーザー定義の名前マッピングを同期する場合、設定ファイルは、以下に示すこれらのマッピングが含まれるように変更されます。

ユーザー定義の名前マッピングに関する RFC 2307 スキーマを使用する LDAP 同期設定: rfc2307_config_user_defined.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
groupUIDNameMapping:
  "cn=admins,ou=groups,dc=example,dc=com": Administrators 1
rfc2307:
    groupsQuery:
```

```
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn (2)
    groupNameAttributes: [ cn ] (3)
    groupMembershipAttributes: [ member ]
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    userUIDAttribute: dn (4)
    userNameAttributes: [ mail ]
    tolerateMemberNotFoundErrors: false
    tolerateMemberOutOfScopeErrors: false
```

(1) ユーザー定義の名前マッピングです。

(2) ユーザー定義の名前マッピングでキーに使用される固有の識別属性です。groupUIDAttribute に DN を使用している場合は **groupsQuery** フィルターを指定できません。詳細なフィルターを実行するには、ホワイトリスト/ブラックリスト方法を使用します。

(3) 固有の識別子がユーザー定義の名前マッピングに存在しない場合に OpenShift Container Platform Group に名前を付けるための属性です。

(4) LDAP サーバーでユーザーを一意に識別する属性です。userUIDAttribute に DN を使用している場合は、**usersQuery** フィルターを指定できません。詳細のフィルターを実行するには、ホワイトリスト/ブラックリスト方法を使用します。

rfc2307_config_user_defined.yaml ファイルと同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=rfc2307_config_user_defined.yaml --
confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

rfc2307_config_user_defined.yaml ファイルを使用して作成される OpenShift Container Platform Group

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: Administrators (1)
users:
- jane.smith@example.com
- jim.adams@example.com
```

1 ユーザー定義の名前マッピングが指定するグループ名です。

12.5.2. ユーザー定義のエラートレランスに関する RFC 2307 の使用によるグループの同期

デフォルトでは、同期されるグループにメンバークエリーで定義された範囲外にあるエントリーを持つメンバーが含まれる場合、グループ同期は以下のエラーを出して失敗します。

```
Error determining LDAP group membership for "<group>": membership lookup
for user "<user>" in group "<group>" failed because of "search for entry
with dn="<user-dn>" would search outside of the base dn specified (dn="
<base-dn>")".
```

これは **usersQuery** フィールドの **baseDN** が間違って設定されていることを示していることがよくあります。ただし、**baseDN** にグループの一部のメンバーが意図的に含まれていない場合、**tolerateMemberOutOfScopeErrors: true** を設定することでグループ同期が継続されます。範囲外のメンバーは無視されます。

同様に、グループ同期プロセスでグループのメンバーの検出に失敗した場合、同期はエラーを出して失敗します。

```
Error determining LDAP group membership for "<group>": membership lookup
for user "<user>" in group "<group>" failed because of "search for entry
with base dn="<user-dn>" refers to a non-existent entry".

Error determining LDAP group membership for "<group>": membership lookup
for user "<user>" in group "<group>" failed because of "search for entry
with base dn="<user-dn>" and filter "<filter>" did not return any results".
```

これは、**usersQuery** フィールドが間違って設定されていることを示していることがよくあります。ただし、グループに欠落していると認識されているメンバーエントリーが含まれる場合、**tolerateMemberNotFoundErrors: true** を設定することでグループ同期が継続されます。問題のあるメンバーは無視されます。

警告

LDAP グループ同期のエラートレランスを有効にすると、同期プロセスは問題のあるメンバーエントリーを無視します。LDAP グループ同期が正しく設定されていない場合、同期された OpenShift Container Platform グループにメンバーが欠落する可能性があります。

問題のあるグループメンバーシップに関する RFC 2307 スキーマを使用する LDAP エントリー: rfc2307_problematic_users.ldif

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users
```

```
dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
member: cn=INVALID,ou=users,dc=example,dc=com (1)
member: cn=Jim,ou=OUTOFSCOPE,dc=example,dc=com (2)
```

1. LDAP サーバーに存在しないメンバーです。
2. 存在する可能性はあるが、同期ジョブのユーザークエリーでは **baseDN** に存在しないメンバーです。

上記の例でエラーを許容するには、以下を同期設定ファイルに追加する必要があります。

エラーを許容する RFC 2307 スキーマを使用した LDAP 同期設定: rfc2307_config_tolerating.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
rfc2307:
    groupsQuery:
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
    groupUIDAttribute: dn
    groupNameAttributes: [ cn ]
    groupMembershipAttributes: [ member ]
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
```

```
        scope: sub
        derefAliases: never
    userUIDAttribute: dn 1
    userNameAttributes: [ mail ]
    tolerateMemberNotFoundErrors: true 2
    tolerateMemberOutOfScopeErrors: true 3
```

2 **true** の場合、同期ジョブは一部のメンバーが見つからなかったグループを許容し、LDAP エントリーが見つからなかったメンバーは無視されます。グループのメンバーが見つからない場合、同期ジョブのデフォルト動作は失敗します。

3 **true** の場合、同期ジョブは、一部のメンバーが **usersQuery** ベース DN で指定されるユーザー範囲外にいるグループを許容し、メンバークエリー範囲外のメンバーは無視されます。グループのメンバーが範囲外の場合、同期ジョブのデフォルト動作は失敗します。

1 LDAP サーバーでユーザーを一意に識別する属性です。userUIDAttribute に DN を使用している場合は、**usersQuery** フィルターを指定できません。詳細のフィルターを実行するには、ホワイトリスト/ブラックリスト方法を使用します。

rfc2307_config_tolerating.yaml ファイルを使用して同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=rfc2307_config_tolerating.yaml --
confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

rfc2307_config.yaml ファイルを使用して作成される OpenShift Container Platform Group

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: admins
users: 1
- jane.smith@example.com
- jim.adams@example.com
```

1 同期ファイルで指定されるグループのメンバーのユーザーです。検索中に許容されるエラーがないメンバーです。

12.5.3. Active Directory の使用によるグループの同期

Active Directory スキーマでは、両方のユーザー (Jane と Jim) がファーストクラスエントリーとして LDAP サーバーに存在し、グループメンバーシップはユーザーの属性に保存されます。以下の **ldif** のスニペットでは、このスキーマのユーザーとグループを定義しています。

Active Directory スキーマを使用する LDAP エントリー: active_directory.ldif

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: admins (1)

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: admins
```

(1) ユーザーのグループメンバーシップはユーザーの属性として一覧表示され、グループはサーバー上にエントリーとして存在しません。**memberOf** 属性はユーザーのリテラル属性である必要はありません。一部の LDAP サーバーでは、これは検索中に作成され、クライアントに返されますが、データベースにコミットされません。

このグループを同期するには、まず設定ファイルを作成する必要があります。Active Directory スキーマでは、ユーザーエントリーの LDAP クエリー定義と内部 OpenShift Container Platform Group レコードでそれらを表すのに使用する属性を指定する必要があります。

明確にするために、OpenShift Container Platform で作成したグループは (可能な場合) ユーザーまたは管理者に表示されるフィールドに識別名以外の属性を使用する必要があります。たとえば、メールによって OpenShift Container Platform グループのユーザーを識別しますが、LDAP サーバーのグループ名でグループの名前を定義します。以下の設定ファイルでは、このような関係を作成しています。

Active Directory スキーマを使用する LDAP 同期設定: active_directory_config.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
activeDirectory:
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        filter: (objectclass=inetOrgPerson)
        pageSize: 0
    userNameAttributes: [ mail ] (1)
    groupMembershipAttributes: [ memberOf ] (2)
```

1. OpenShift Container Platform Group レコードでユーザー名として使用される属性です。
2. メンバーシップ情報を保存するユーザーの属性です。

active_directory_config.yaml ファイルを使用して同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=active_directory_config.yaml --confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

active_directory_config.yaml ファイルを使用して作成される OpenShift Container Platform グループ

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 (1)
    openshift.io/ldap.uid: admins (2)
    openshift.io/ldap.url: LDAP_SERVER_IP:389 (3)
  creationTimestamp:
  name: admins (4)
users: (5)
- jane.smith@example.com
- jim.adams@example.com
```

1. この OpenShift Container Platform グループと LDAP サーバーが最後に同期された時間です。ISO 6801 形式を使用します。
2. LDAP サーバーのグループの固有識別子です。
3. このグループのレコードが保存される LDAP サーバーの IP アドレスとホストです。
4. LDAP サーバーに一覧表示されるグループ名です。
5. グループのメンバーのユーザーです。同期ファイルで指定される名前が使用されます。

12.5.4. 拡張された Active Directory の使用によるグループの同期

拡張された Active Directory スキーマでは、両方のユーザー (Jane と Jim) とグループがファーストクラスエントリーとして LDAP サーバーに存在し、グループメンバーシップはユーザーの属性に保存されます。以下の **ldif** のスニペットではこのスキーマのユーザーとグループを定義しています。

拡張された Active Directory スキーマを使用する LDAP エントリー: augmented_active_directory.ldif

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
```

```
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com (1)

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com (2)
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
```

(1) ユーザーのグループメンバーシップはユーザーの属性として一覧表示されます。

(2) このグループは LDAP サーバーのファーストクラスエントリーです。

このグループを同期するには、まず設定ファイルを作成する必要があります。拡張された Active Directory スキーマでは、ユーザーエントリーとグループエントリーの両方の LDAP クエリー定義と内部 OpenShift Container Platform の Group レコードでそれらを表すのに使用する属性を指定する必要があります。

明確にするために、OpenShift Container Platform で作成するグループは (可能な場合) ユーザーまたは管理者に表示されるフィールドに識別名以外の属性を使用する必要があります。たとえば、メールによって OpenShift Container Platform グループのユーザーを識別し、一般名としてグループの名前を使用します。以下の設定ファイルではこのような関係を作成しています。

拡張された Active Directory スキーマを使用する LDAP 同期設定: augmented_active_directory_config.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
    groupsQuery:
```

```
        baseDN: "ou=groups,dc=example,dc=com"
        scope: sub
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn (1)
    groupNameAttributes: [ cn ] (2)
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        filter: (objectclass=inetOrgPerson)
        pageSize: 0
    userNameAttributes: [ mail ] (3)
    groupMembershipAttributes: [ memberOf ] (4)
```

(1) LDAP サーバーのグループを一意に識別する属性です。groupUIDAttribute に DN を使用している場合、**groupsQuery** フィルターを指定できません。詳細なフィルターを実行するには、ホワイトリスト/ブラックリスト方法を使用します。

(2) Group の名前として使用する属性です。

(3) OpenShift Container Platform Group レコードでユーザー名として使用される属性です。

(4) メンバーシップ情報を保存するユーザーの属性です。

augmented_active_directory_config.yaml ファイルを使用して同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=augmented_active_directory_config.yaml
--confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

augmented_active_directory_config.yaml ファイルを使用して作成される OpenShift Group

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 (1)
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com (2)
    openshift.io/ldap.url: LDAP_SERVER_IP:389 (3)
  creationTimestamp:
  name: admins (4)
users: (5)
- jane.smith@example.com
- jim.adams@example.com
```

(1) この OpenShift Container Platform グループと LDAP サーバーが最後に同期された時間です。ISO 6801 形式を使用します。

(2) LDAP サーバーのグループの固有識別子です。

3 このグループのレコードが保存される LDAP サーバーの IP アドレスとホストです。

4 同期ファイルが指定するグループ名です。

5 グループのメンバーのユーザーです。同期ファイルで指定される名前が使用されます。

12.6. ネスト化されたメンバーシップ同期の例

OpenShift Container Platform の Group はネスト化しません。LDAP サーバーはデータが使用される前にグループメンバーシップを平坦化する必要があります。Microsoft の Active Directory Server は、**LDAP_MATCHING_RULE_IN_CHAIN** ルールによりこの機能をサポートしており、これには OID **1.2.840.113556.1.4.1941** が設定されています。さらに、このマッチングルールを使用すると、明示的にホワイトリスト化されたグループのみをを同期できます。

このセクションでは、拡張された Active Directory スキーマの例を取り上げ、1 名のユーザー **Jane** と 1 つのグループ **otheradmins** をメンバーとして持つ **admins** というグループを同期します。**otheradmins** グループには 1 名のユーザーメンバー **Jim** が含まれます。この例では以下のことを説明しています。

- グループとユーザーが LDAP サーバーに追加される方法。
- LDAP 同期設定ファイルの概観。
- 同期後に生成される OpenShift Container Platform の Group レコード。

拡張された Active Directory スキーマでは、ユーザー (**Jane** と **Jim**) とグループの両方がファーストクラスエントリーとして LDAP サーバーに存在し、グループメンバーシップはユーザーまたはグループの属性に保存されます。以下の **ldif** のスニペットはこのスキーマのユーザーとグループを定義します。

ネスト化されたメンバーを持つ拡張された Active Directory スキーマを使用する LDAP エントリー: augmented_active_directory_nested.ldif

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com (1)

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
```

```
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=otheradmins,ou=groups,dc=example,dc=com (2)

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com (3)
objectClass: group
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=otheradmins,ou=groups,dc=example,dc=com

dn: cn=otheradmins,ou=groups,dc=example,dc=com (4)
objectClass: group
cn: otheradmins
owner: cn=admin,dc=example,dc=com
description: Other System Administrators
memberOf: cn=admins,ou=groups,dc=example,dc=com (5) (6)
member: cn=Jim,ou=users,dc=example,dc=com
```

(1) (2) (5) ユーザーとグループのメンバーシップはオブジェクトの属性として一覧表示されます。

(3) (4) このグループは LDAP サーバーのファーストクラスエントリーです。

(6) **otheradmins** グループは **admins** グループのメンバーです。

Active Directory を使用してネスト化されたグループを同期するには、ユーザーエントリーとグループエントリーの両方の LDAP クエリー定義と内部 OpenShift Container Platform Group レコードでそれらを表すのに使用する属性を指定する必要があります。さらに、この設定では特定の変更が必要となります。

- **oc adm groups sync** コマンドはグループを明示的にホワイトリスト化する必要があります。
- ユーザーの **groupMembershipAttributes** には **"memberOf:1.2.840.113556.1.4.1941:"** を含め、**LDAP_MATCHING_RULE_IN_CHAIN** ルールに従う必要があります。
- **groupUIDAttribute** は **dn** に設定される必要があります。
- **groupsQuery**:
  - **filter** を設定しないでください。
  - 有効な **derefAliases** を設定する必要があります。
  - **baseDN** を設定しないでください。この値は無視されます。
  - **scope** を設定しないでください。この値は無視されます。

明確にするために、OpenShift Container Platform で作成するグループは (可能な場合) ユーザーまたは

管理者に表示されるフィールドに識別名以外の属性を使用する必要があります。たとえば、メールによって OpenShift Container Platform グループのユーザーを識別し、一般名としてグループの名前を使用します。以下の設定ファイルではこのような関係を作成しています。

ネスト化されたメンバーを持つ拡張された Active Directory スキーマを使用する LDAP 同期設定: augmented_active_directory_config_nested.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
    groupsQuery: (1)
        derefAliases: never
        pageSize: 0
    groupUIDAttribute: dn (2)
    groupNameAttributes: [ cn ] (3)
    usersQuery:
        baseDN: "ou=users,dc=example,dc=com"
        scope: sub
        derefAliases: never
        filter: (objectclass=inetOrgPerson)
        pageSize: 0
    userNameAttributes: [ mail ] (4)
    groupMembershipAttributes: [ "memberOf:1.2.840.113556.1.4.1941:" ] (5)
```

(1) **groupsQuery** フィルターは指定できません。**groupsQuery** ベース DN とスコープ値は無視されます。**groupsQuery** は有効な **derefAliases** を設定する必要があります。

(2) LDAP サーバーのグループを一意に識別する属性です。**dn** に設定される必要があります。

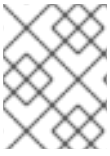
(3) Group の名前として使用する属性です。

(4) OpenShift Container Platform の Group レコードでユーザー名として使用される属性です。**mail** または **sAMAccountName** がほとんどのインストールで推奨されます。

(5) メンバーシップ情報を保存するユーザーの属性です。**LDAP_MATCHING_RULE_IN_CHAIN** を使用することに注意してください。

augmented_active_directory_config_nested.yaml ファイルを使用して同期するには、以下を実行します。

```
$ oc adm groups sync \
    'cn=admins,ou=groups,dc=example,dc=com' \
    --sync-config=augmented_active_directory_config_nested.yaml \
    --confirm
```



注記

cn=admins,ou=groups,dc=example,dc=com グループを明示的にホワイトリスト化する必要があります。

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

augmented_active_directory_config_nested.yaml ファイルを使用して作成される OpenShift Group

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 (1)
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com (2)
    openshift.io/ldap.url: LDAP_SERVER_IP:389 (3)
  creationTimestamp:
  name: admins (4)
users: (5)
- jane.smith@example.com
- jim.adams@example.com
```

(1) この OpenShift Container Platform グループと LDAP サーバーが最後に同期された時間です。ISO 6801 形式を使用します。

(2) LDAP サーバーのグループの固有識別子です。

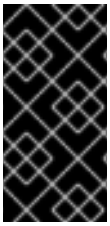
(3) このグループのレコードが保存される LDAP サーバーの IP アドレスとホストです。

(4) 同期ファイルが指定するグループ名です。

(5) 同期ファイルで指定されるように名前が付けられる Group のメンバーのユーザーです。グループメンバーシップは Microsoft Active Directory Server によって平坦化されているため、ネスト化されたグループのメンバーが含まれることに注意してください。

12.7. LDAP 同期設定の仕様

設定ファイルのオブジェクト仕様は以下で説明されています。スキーマオブジェクトにはそれぞれのフィールドがあることに注意してください。たとえば、v1.ActiveDirectoryConfig は **groupsQuery** フィールドを持ちませんが、v1.RFC2307Config と v1.AugmentedActiveDirectoryConfig の両方にこのフィールドがあります。



重要

バイナリー属性はサポートされていません。LDAP サーバーの全属性データは、UTF-8 エンコード文字列の形式である必要があります。たとえば、ID 属性として、バイナリー属性を使用することはできません (例: **objectGUID**)。代わりに **sAMAccountName** または **userPrincipalName** などの文字列属性を使用する必要があります。

12.7.1. v1.LDAPSyncConfig

LDAPSyncConfig は、LDAP グループ同期を定義するために必要な設定オプションを保持します。

名前	説明	スキーマ
kind	このオブジェクトが表す REST リソースを表す文字列の値です。サーバーはクライアントが要求を送信するエンドポイントからこれを推測できることがあります。これを更新することはできません。CamelCase。詳細については、https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#types-kinds を参照してください。	string
apiVersion	オブジェクトのこの表現のバージョンスキーマを定義します。サーバーは認識されたスキーマを最新の内部値に変換し、認識されない値は拒否することがあります。詳細については、https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#resources を参照してください。	string
url	ホストは接続先の LDAP サーバーのスキーム、ホストおよびポートになります。**scheme://host:port**	string
bindDN	LDAP サーバーをバインドする任意の DN です。	string
bindPassword	検索フェーズでバインドする任意のパスワードです。	v1.StringSource
insecure	**true** の場合、接続に TLS を使用しないでください。**ldaps://** の URL スキームで true に設定することはできません。**false** の場合、**ldaps://** URL は TLS を使用して接続し、**ldap://** URL は、https://tools.ietf.org/html/rfc2830 で指定されるように StartTLS を使用して TLS 接続にアップグレードされます。	ブール値

名前	説明	スキーマ
ca	サーバーへ要求を行う際に使用する任意の信頼された認証局バンドルです。空の場合は、デフォルトのシステムルートが使用されます。	string
groupUIDNameMapping	LDAP グループ UID の OpenShift Container Platform Group 名への任意の直接マッピングです。	object
rfc2307	RFC2307 と同じ方法でセットアップされた LDAP サーバーからデータを抽出するための設定を保持します。ファーストクラスグループとユーザーエントリーを抽出し、グループメンバーシップはメンバーを一覧表示するグループエントリーの複数値の属性によって決定されます。	v1.RFC2307Config
activeDirectory	Active Directory に使用されるのと同じ方法でセットアップされた LDAP サーバーからデータを抽出するための設定を保持します。ファーストクラスユーザーエントリーを抽出し、グループメンバーシップはメンバーが属するグループを一覧表示するメンバーの複数値の属性によって決定されます。	v1.ActiveDirectoryConfig
augmentedActiveDirectory	上記の Active Directory で使用されるのと同じ方法でセットアップされた LDAP サーバーからデータを抽出するための設定を保持します。1 つの追加として、ファーストクラスグループエントリーが存在し、それらはメタデータを保持するために使用されますが、グループメンバーシップは設定されません。	v1.AugmentedActiveDirectoryConfig

12.7.2. v1.StringSource

StringSource によって文字列インラインを指定できます。または環境変数またはファイルを使用して外部から指定することもできます。文字列の値のみを含む場合、単純な JSON 文字列にマーシャルします。

名前	説明	スキーマ
value	クリアテキスト値、または **keyFile** が指定されている場合は暗号化された値を指定します。	string
env	クリアテキスト値、または **keyFile** が指定されている場合は暗号化された値を含む環境変数を指定します。	string
file	クリアテキスト値、または **keyFile** が指定されている場合は暗号化された値を含むファイルを参照します。	string
keyFile	値を復号化するために使用するキーを含むファイルを参照します。	string

12.7.3. v1.LDAPQuery

LDAPQuery は LDAP クエリーの作成に必要なオプションを保持します。

名前	説明	スキーマ
baseDN	すべての検索が開始されるディレクトリーのブランチの DN です。	string
scope	検索の (任意の) 範囲です。**base** (ベースオブジェクトのみ)、**one** (ベースレベルのすべてのオブジェクト)、**sub** (サブツリー全体) のいずれかになります。設定されていない場合は、デフォルトで **sub** になります。	string
derefAliases	エイリアスに関する検索の (任意の) 動作です。**never** (エイリアスを逆参照しない)、**search** (検索中の逆参照のみ)、**base** (ベースオブジェクト検索時の逆参照のみ)、**always** (常に逆参照を行う) のいずれかになります。設定されていない場合、デフォルトで **always** になります。	string

名前	説明	スキーマ
timeout	応答の待機を中止するまでにサーバーへの要求を未処理のままにする時間制限 (秒単位) を保持します。これが **0** の場合、クライアント側の制限が設定されないことになります。	整数
filter	ベース DN を持つ LDAP サーバーから関連するすべてのエントリーを取得する有効な LDAP 検索フィルターです。	string
pageSize	LDAP エントリーで測定される、推奨される最大ページサイズです。ページサイズ **0** はページングが実行されないことを意味します。	整数

12.7.4. v1.RFC2307Config

RFC2307Config は、RFC2307 スキーマを使用してどのように LDAP グループ同期が LDAP サーバーに相互作用するかを定義するために必要な設定オプションを保持します。

名前	説明	スキーマ
groupsQuery	グループエントリーを返す LDAP クエリーのテンプレートを保持します。	v1.LDAPQuery
groupUIDAttribute	LDAP グループエントリーのどの属性が固有の識別子として解釈されるかを定義します。(**ldapGroupUID**)	string
groupNameAttributes	LDAP グループエントリーのどの属性が OpenShift Container Platform グループに使用する名前として解釈されるかを定義します。	文字列の配列
groupMembershipAttributes	LDAP グループエントリーのどの属性がメンバーとして解釈されるかを定義します。それらの属性に含まれる値は **UserUIDAttribute** でクエリーできる必要があります。	文字列の配列

名前	説明	スキーマ
usersQuery	ユーザーエントリーを返す LDAP クエリーのテンプレートを保持します。	v1.LDAPQuery
userUIDAttribute	LDAP ユーザーエントリーのどの属性が固有の識別子として解釈されるかを定義します。**GroupMembershipAttributes** で検出される値に対応している必要があります。	string
userNameAttributes	LDAP ユーザーエントリーのどの属性が順番に OpenShift Container Platform ユーザー名として使われるかを定義します。空でない値を持つ最初の属性が使用されます。これは **LDAPPasswordIdentityProvider** の **PreferredUsername** 設定と一致している必要があります。OpenShift Container Platform Group レコードでユーザーの名前として使用する属性です。ほとんどのインストールで、**mail** または **sAMAccountName** を選択することが推奨されます。	文字列の配列
tolerateMemberNotFoundErrors	ユーザーエントリーがない場合の LDAP 同期ジョブの動作を決定します。**true** の場合、何も検出しない ユーザーの LDAP クエリーは許容され、エラーのみがログに記録されます。**false** の場合、ユーザーのクエリーが何も検出しないと、LDAP 同期ジョブは失敗します。デフォルトの値は「false」です。「true」に設定されたこのフラグを持つ LDAP 同期ジョブの設定が間違っていると、グループメンバーシップが削除されることがあるため、注意してこのフラグを使用してください。	ブール値

名前	説明	スキーマ
tolerateMemberOutOfScopeErrors	範囲外のユーザーエントリーが検出される場合の LDAP 同期ジョブの動作を決定します。**true** の場合、すべてのユーザークエリーに指定されるベース DN 外のユーザーの LDAP クエリーは許容され、エラーのみがログに記録されます。**false** の場合、ユーザークエリーですべてのユーザークエリーで指定されるベース DN 外を検索すると LDAP 同期ジョブは失敗します。このフラグを **true** に設定した LDAP 同期ジョブの設定が間違っていると、ユーザーのいないグループが発生することがあるため、注意してこのフラグを使用してください。	ブール値

12.7.5. v1.ActiveDirectoryConfig

ActiveDirectoryConfig は必要な設定オプションを保持し、どのように LDAP グループ同期が Active Directory スキーマを使用して LDAP サーバーと相互作用するかを定義します。

名前	説明	スキーマ
usersQuery	ユーザーエントリーを返す LDAP クエリーのテンプレートを保持します。	v1.LDAPQuery
userNameAttributes	LDAP ユーザーエントリーのどの属性が OpenShift Container Platform ユーザー名として解釈されるかを定義します。OpenShift Container Platform Group レコードでユーザーの名前として使用する属性です。ほとんどのインストールで、**mail** または **sAMAccountName** を選択することが推奨されています。	文字列の配列
groupMembershipAttributes	LDAP ユーザーのどの属性がメンバーの属するグループとして解釈されるかを定義します。	文字列の配列

12.7.6. v1.AugmentedActiveDirectoryConfig

AugmentedActiveDirectoryConfig は必要な設定オプションを保持し、どのように LDAP グループ同期が拡張された Active Directory スキーマを使用して LDAP サーバーに相互作用するかを定義します。

名前	説明	スキーマ
usersQuery	ユーザーエントリーを返す LDAP クエリーのテンプレートを保持します。	v1.LDAPQuery
userNameAttributes	LDAP ユーザーエントリーのどの属性が OpenShift Container Platform ユーザー名として解釈されるかを定義します。OpenShift Container Platform Group レコードでユーザーの名前として使用する属性です。ほとんどのインストールで、**mail** または **sAMAccountName** を選択することが推奨されています。	文字列の配列
groupMembershipAttributes	LDAP ユーザーのどの属性がメンバーの属するグループとして解釈されるかを定義します。	文字列の配列
groupsQuery	グループエントリーを返す LDAP クエリーのテンプレートを保持します。	v1.LDAPQuery
groupUIDAttribute	LDAP グループエントリーのどの属性が固有の識別子として解釈されるかを定義します。(**ldapGroupUID**)	string
groupNameAttributes	LDAP グループエントリーのどの属性が OpenShift Container Platform グループに使用する名前として解釈されるかを定義します。	文字列の配列

第13章 LDAP フェイルオーバーの設定

OpenShift Container Platform は Lightweight Directory Access Protocol (LDAP) セットアップで使用するための 認証プロバイダーを提供しますが、接続できるのは単一の LDAP サーバーのみです。OpenShift Container Platform インストール時に、LDAP フェイルオーバーについて System Security Services Daemon (SSSD) を設定し、ある LDAP サーバーが失敗した場合にクラスターにアクセスできるようにします。

この設定には、詳細な設定および通信先となる OpenShift Container Platform の認証サーバー (**リモート Basic 認証サーバー** とも呼ばれます) が別途必要となります。メールアドレスなどの追加の属性を OpenShift Container Platform に渡すようにこのサーバーを設定し、それらの属性を Web コンソールで表示できるようにします。

このトピックでは、専用の物理または仮想マシン (VM) のセットアップを実行する方法や、コンテナーでの SSSD の設定方法についても説明します。

重要

このトピックのすべてのセクションを完了する必要があります。

13.1. 基本リモート認証設定の前提条件

- セットアップを始める前に、LDAP サーバーの以下の情報について知っておく必要があります。
  - ディレクトリーサーバーが FreeIPA、Active Directory、または別の LDAP ソリューションでサポートされているかどうか。
  - LDAP サーバーの Uniform Resource Identifier (URI) (例: **ldap.example.com**)。
  - LDAP サーバーの CA 証明書の場所。
  - LDAP サーバーがユーザーグループの RFC 2307 または RFC2307bis に対応しているかどうか。
- サーバーを準備します。
  - **remote-basic.example.com**: リモート Basic 認証サーバーとして使用する VM。
    - Red Hat Enterprise Linux 7.0 以降などの、このサーバーの SSSD バージョン 1.12.0 を含むオペレーティングシステムを選択します。
  - **openshift.example.com**: OpenShift Container Platform の新規インストール。
    - このクラスターに認証方法を設定することはできません。
    - このクラスターで OpenShift Container Platform を起動することはできません。

13.2. 証明書の生成およびリモート BASIC 認証サーバーとの共有

Ansible ホストインベントリーファイル (デフォルトは **/etc/ansible/hosts**) に一覧表示された 1 つ目のマスターホストで以下の手順を実行します。

1. リモート Basic 認証サーバーと OpenShift Container Platform 間の通信に新異性をもたせるために、このセットアップの他のフェーズで使用する Transport Layer Security (TLS) 証明書のセットを作成します。以下のコマンドを実行します。

```
# openshift start \
    --public-master=https://openshift.example.com:8443 \
    --write-config=/etc/origin/
```

出力には、**/etc/origin/master/ca.crt** および **/etc/origin/master/ca.key** の署名用証明書が含まれます。

2. 署名用証明書を使用してリモート Basic 認証サーバーで使用するキーを生成します。

```
# mkdir -p /etc/origin/remote-basic/
# oc adm ca create-server-cert \
    --cert='/etc/origin/remote-basic/remote-basic.example.com.crt' \
    --key='/etc/origin/remote-basic/remote-basic.example.com.key' \
    --hostnames=remote-basic.example.com \ 1
    --signer-cert='/etc/origin/master/ca.crt' \
    --signer-key='/etc/origin/master/ca.key' \
    --signer-serial='/etc/origin/master/ca.serial.txt'
```

1 リモート Basic 認証サーバーにアクセスする必要のある、すべてのホスト名およびインターネット IP アドレスのコンマ区切りの一覧です。

注記

生成する証明書ファイルは 2 年間有効です。この期間は、**--expire-days** および **--signer-expire-days** の値を変更して変更することができますが、セキュリティー上の理由により、730 より大きな値を設定しないでください。

重要

リモート Basic 認証サーバーにアクセスする必要のあるすべてのホスト名およびインターフェース IP アドレスを一覧表示しない場合、HTTPS 接続は失敗します。

3. 必要な証明書およびキーをリモート Basic 認証サーバーにコピーします。

```
# scp /etc/origin/master/ca.crt \
      root@remote-basic.example.com:/etc/pki/CA/certs/

# scp /etc/origin/remote-basic/remote-basic.example.com.crt \
      root@remote-basic.example.com:/etc/pki/tls/certs/

# scp /etc/origin/remote-basic/remote-basic.example.com.key \
      root@remote-basic.example.com:/etc/pki/tls/private/
```

13.3. SSSD での LDAP フェイルオーバーの設定

リモート Basic 認証サーバーで以下の手順を実行します。

メールアドレスおよび表示名などの属性を取得し、それらを OpenShift Container Platform に渡して Web インターフェースに表示することができるように SSSD を設定します。以下の手順では、メールアドレスを OpenShift Container Platform に指定するように SSSD を設定します。

1. 必要な SSSD および Web サーバーコンポーネントをインストールします。

```
# yum install -y sssd \
                 sssd-dbus \
                 realmd \
                 httpd \
                 mod_session \
                 mod_ssl \
                 mod_lookup_identity \
                 mod_authnz_pam \
                 php \
                 mod_php
```

2. LDAP サーバーに対してこの VM を認証するように SSSD を設定します。LDAP サーバーが FreeIPA または Active Directory 環境の場合、**realmd** を使用してこのマシンをドメインに参加させることができます。

```
# realm join ldap.example.com
```

より高度なケースの場合については、『システムレベル認証ガイド』を参照してください。

3. SSSD を使用して LDAP のフェイルオーバーの状態を使用するには、**ldap_uri** 行の **/etc/sssd/sssd.conf** ファイルにその他のエントリーを追加します。FreeIPA に登録されたシステムは DNS SRV レコードを使用してフェイルオーバーを自動的に処理します。

4. **/etc/sssd/sssd.conf** ファイルの **[domain/DOMAINNAME]** セクションを変更し、この属性を追加します。

```
[domain/example.com]
...
ldap_user_extra_attrs = mail 1
```

1 LDAP ソリューションについてのメールアドレスの取得に必要な適切な属性を指定します。IPA の場合は、**mail** を指定します。他の LDAP ソリューションは **email** などの別の属性を使用する可能性があります。

5. **/etc/sssd/sssd.conf** ファイルの **domain** パラメーターには **[domain/DOMAINNAME]** セクションに一覧表示されているドメイン名のみが含まれていることを確認します。

```
domains = example.com
```

6. メール属性を取得するために Apache パーミッションを付与します。以下の行を **/etc/sssd/sssd.conf** ファイルの **[ifp]** セクションに追加します。

```
[ifp]
user_attributes = +mail
allowed_uids = apache, root
```

7. すべての変更が適切に適用されていることを確認するには、SSSD を再起動します。

```
$ systemctl restart sssd.service
```

8. ユーザー情報が適切に取得できるかテストします。

```
$ getent passwd <username>
username:*:12345:12345:Example User:/home/username:/usr/bin/bash
```

9. 指定したメール属性がドメインからメールアドレスを返すことを確認します。

```
# dbus-send --print-reply --system --
dest=org.freedesktop.sssd.infopipe \
    /org/freedesktop/sssd/infopipe
org.freedesktop.sssd.infopipe.GetUserAttr \
    string:username \ 1
    array:string:mail 2

method return time=1528091855.672691 sender=:1.2787 ->
destination=:1.2795 serial=13 reply_serial=2
   array [
      dict entry(
         string "mail"
         variant             array [
               string "username@example.com"
            ]
      )
   ]
```

1 LDAP ソリューションでユーザー名を指定します。

2 設定した属性を指定します。

10. LDAP ユーザーとして VM へのログインを試行し、LDAP 認証情報を使用してログインできることを確認します。ログインにはローカルコンソールまたは SSH などのリモートサービスを使用できます。

重要

デフォルトで、すべてのユーザーは LDAP 認証情報を使用してリモート Basic 認証サーバーにログインできます。この動作は変更することができます。

- IPA に参加するシステムを使用する場合、ホストベースのアクセス制御を設定します。
- Active Directory に参加するシステムを使用する場合には、グループポリシーオブジェクトを使用します。
- その他のケースについては、SSSD 設定 についてのドキュメントを参照してください。

13.4. APACHE での SSSD の使用の設定

1. 以下の内容を含む **/etc/pam.d/openshift** ファイルを作成します。

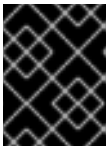
```
auth required pam_sss.so
account required pam_sss.so
```

この設定により、認証要求が **openshift** スタックに対して発行された時に PAM (プラグ可能な認証モジュール) は **pam_sss.so** を使用して認証とアクセス制御を決定できるようになります。

2. **/etc/httpd/conf.modules.d/55-authnz_pam.conf** ファイルを編集して、以下の行のコメントを解除します。

```
LoadModule authnz_pam_module modules/mod_authnz_pam.so
```

3. Apache **httpd.conf** ファイルをリモート Basic 認証用に設定するには、**openshift-remote-basic-auth.conf** ファイルを **/etc/httpd/conf.d** ディレクトリーに作成します。以下のテンプレートを使用して必要な設定および値を指定します。



重要

テンプレートを十分に確認し、その内容を環境に合うようにカスタマイズします。

```
LoadModule request_module modules/mod_request.so
LoadModule php7_module modules/libphp7.so

# Nothing needs to be served over HTTP.  This virtual host simply
redirects to
# HTTPS.
<VirtualHost *:80>
  DocumentRoot /var/www/html
  RewriteEngine              On
  RewriteRule     ^(.*)$     https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
  # This needs to match the certificates you generated.  See the CN
and X509v3
  # Subject Alternative Name in the output of:
  # openssl x509 -text -in /etc/pki/tls/certs/remote-
basic.example.com.crt
  ServerName remote-basic.example.com

  DocumentRoot /var/www/html

  # Secure all connections with TLS
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/remote-basic.example.com.crt
  SSLCertificateKeyFile /etc/pki/tls/private/remote-
basic.example.com.key
  SSLCACertificateFile /etc/pki/CA/certs/ca.crt

  # Require that TLS clients provide a valid certificate
  SSLVerifyClient require
  SSLVerifyDepth 10

  # Other SSL options that may be useful
  # SSLCertificateChainFile ...
  # SSLCARevocationFile ...
```

```
  # Send logs to a specific location to make them easier to find
  ErrorLog logs/remote_basic_error_log
  TransferLog logs/remote_basic_access_log
  LogLevel warn

  # PHP script that turns the Apache REMOTE_USER env var
  # into a JSON formatted response that OpenShift understands
  <Location /check_user.php>
    # all requests not using SSL are denied
    SSLRequireSSL
    # denies access when SSLRequireSSL is applied
    SSLOptions +StrictRequire
    # Require both a valid basic auth user (so REMOTE_USER is always
set)
    # and that the CN of the TLS client matches that of the
OpenShift master
    <RequireAll>
      Require valid-user
      Require expr %{SSL_CLIENT_S_DN_CN} == 'system:openshift-
master'
    </RequireAll>
    # Use basic auth since OpenShift will call this endpoint with a
basic challenge
    AuthType Basic
    AuthName openshift
    AuthBasicProvider PAM
    AuthPAMService openshift

    # Store attributes in environment variables. Specify the email
attribute that
    # you confirmed.
    LookupOutput Env
    LookupUserAttr mail REMOTE_USER_MAIL
    LookupUserGECOS REMOTE_USER_DISPLAY_NAME

    # Other options that might be useful

    # While REMOTE_USER is used as the sub field and serves as the
immutable ID,
    # REMOTE_USER_PREFERRED_USERNAME could be used to have a
different username
    # LookupUserAttr <attr_name> REMOTE_USER_PREFERRED_USERNAME

    # Group support may be added in a future release
    # LookupUserGroupsIter REMOTE_USER_GROUP
  </Location>

  # Deny everything else
  <Location ~ "^((?!\/check_user\.php).)*$">
      Deny from all
  </Location>
</VirtualHost>
```

4. **check_user.php** スクリプトを **/var/www/html** ディレクトリーに作成します。以下のコードを組み込みます。

```
<?php
// Get the user based on the Apache var, this should always be
// set because we 'Require valid-user' in the configuration
$user = apache_getenv('REMOTE_USER');

// However, we assume it may not be set and
// build an error response by default
$data = array(
    'error' => 'remote PAM authentication failed'
);

// Build a success response if we have a user
if (!empty($user)) {
    $data = array(
        'sub' => $user
    );
    // Map of optional environment variables to optional JSON fields
    $env_map = array(
        'REMOTE_USER_MAIL' => 'email',
        'REMOTE_USER_DISPLAY_NAME' => 'name',
        'REMOTE_USER_PREFERRED_USERNAME' => 'preferred_username'
    );

    // Add all non-empty environment variables to JSON data
    foreach ($env_map as $env_name => $json_name) {
        $env_data = apache_getenv($env_name);
        if (!empty($env_data)) {
            $data[$json_name] = $env_data;
        }
    }
}

// We always output JSON from this script
header('Content-Type: application/json', true);

// Write the response as JSON
echo json_encode($data);
?>
```

5. Apache がモジュールを読み込めるようにします。**/etc/httpd/conf.modules.d/55-lookup_identity.conf** ファイルを変更し、以下の行のコメントを解除します。

```
LoadModule lookup_identity_module modules/mod_lookup_identity.so
```

6. SELinux ブール値を設定し、SElinux が Apache が D-BUS を介して SSSD に接続することを許可するようにします。

```
# setsebool -P httpd_dbus_sssd on
```

7. SELinux に Apache による PAM サブシステムへの問い合わせを受け入れることを指示するブール値を設定します。

```
# setsebool -P allow_httpd_mod_auth_pam on
```

8. Apache を起動します。

```
# systemctl start httpd.service
```

13.5. OPENSHIFT CONTAINER PLATFORM が SSSD を基本リモート認証サーバーとして使用する設定

作成した新規のアイデンティティープロバイダーを使用するようクラスターのデフォルト設定を変更します。Ansible ホストインベンリーファイルに最初に一覧表示されるマスターホストで以下の手順を実行します。

1. **/etc/origin/master/master-config.yaml** ファイルを開きます。

2. **identityProviders** セクションの場所を見つけ、これを以下のコードに置き換えます。

```
  identityProviders:
  - name: sssd
    challenge: true
    login: true
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: BasicAuthPasswordIdentityProvider
      url: https://remote-basic.example.com/check_user.php
      ca: /etc/origin/master/ca.crt
      certFile: /etc/origin/master/openshift-master.crt
      keyFile: /etc/origin/master/openshift-master.key
```

3. 更新された設定を使って OpenShift Container Platform を起動します。

```
# openshift start \
    --public-master=https://openshift.example.com:8443 \
    --master-config=/etc/origin/master/master-config.yaml \
    --node-config=/etc/origin/node-node1.example.com/node-
config.yaml
```

4. **oc** CLI を使用してログインをテストします。

```
oc login https://openshift.example.com:8443
```

有効な LDAP 認証情報のみを使用してログインすることができます。

5. アイデンティティーを一覧表示し、各ユーザー名のメールアドレスが表示されていることを確認します。以下のコマンドを実行します。

```
$ oc get identity -o yaml
```

第14章 SDN の設定

14.1. 概要

OpenShift SDNは、OpenShift Container Platform クラスターでの Pod 間の通信を有効にして **Pod ネットワーク**を構築します。現在利用可能な SDN プラグインは 3 種類 (**ovs-subnet**、**ovs-multitenant**、**ovs-networkpolicy**) あり、これらは Pod ネットワークを設定するためのそれぞれ異なる方法を提供します。

14.2. 利用可能な SDN プロバイダー

アップストリームの Kubernetes プロジェクトはデフォルトのネットワークソリューションを備えていません。その代わりに、Kubernetes では Container Network Interface (CNI) を開発し、ネットワークプロバイダーが独自の SDN ソリューションを統合することを可能にしています。

Red Hat は、サードパーティーのプラグインのほかにも追加設定なしで使用できるいくつかの OpenShift SDN プラグインを提供しています。

Red Hat は数多くの SDN プロバイダーと協力し、Kubernetes CNI インターフェースを使用した OpenShift Container Platform 上での SDN ネットワークソリューション (これには、製品のエンタイトルメントプロセスでの SDN プラグインのサポートプロセスが含まれます) の認定を行っています。Red Hat は、ユーザーが OpenShift でサポートケースを作成される場合、両社が共にユーザーのニーズに対応できるよう交換プロセスを促進し、これを容易にできます。

以下は、サードパーティーのベンダーが OpenShift Container Platform で直接検証を行い、サポートしている SDN ソリューションです。

- Cisco Contiv (™)
- Juniper Contrail (™)
- Nokia Nuage (™)
- Tigera Calico (™)
- VMware NSX-T (™)

VMware NSX-T (™) の OpenShift Container Platform へのインストール
VMware NSX-T (™) は、クラウドネイティブなアプリケーション環境を構築するための SDN およびセキュリティー基盤を提供しています。これらの環境には、vSphere Hypervisors (ESX) のほかに KVM とネイティブなパブリッククラウドが含まれます。

現在の統合には、NSX-T と OpenShift Container Platform の両方の**新規**インストールが必要です。現時点で、NSX-T バージョン 2.1 がサポートされ、これは ESX と KVM のハイパーバイザーの使用のみに対応しています。

詳細は「NSX-T Container Plug-in for OpenShift - Installation and Administration Guide」を参照してください。

14.3. ANSIBLE を使用した POD ネットワークの設定

初回のクラスターインストールでは、**ovs-subnet** プラグインはデフォルトでインストールされ、設定されますが、このプラグインは、**`os_sdn_network_plugin_name`** パラメーターを使ってインストール時に上書きされる可能性があります。これは Ansible インベントリーファイルで設定できます。

たとえば、標準の **ovs-subnet** プラグインを上書きし、代わりに **ovs-multitenant** プラグインを使用するには、以下を実行します。

```
# Configure the multi-tenant SDN plugin (default is 'redhat/openshift-ovs-
subnet')
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
```

インベントリーファイルに設定できるネットワーキング関連の Ansible 変数の詳細については、「クラスター変数の設定」を参照してください。

14.4. マスターでの POD ネットワークの設定

クラスター管理者は、マスター設定ファイル (デフォルトの場所は **/etc/origin/master/master-config.yaml**) の **networkConfig** セクションにあるパラメーターを変更することで、マスターホスト上で Pod ネットワーク設定を管理できます。

単一 CIDR の Pod ネットワーク設定

```
networkConfig:
  clusterNetworks:
  - cidr: 10.128.0.0/14 (1)
    hostSubnetLength: 9 (2)
  networkPluginName: "redhat/openshift-ovs-subnet" (3)
  serviceNetworkCIDR: 172.30.0.0/16 (4)
```

(1) ノード IP の割り当て用のクラスターネットワーク

(2) ノード内での Pod IP の割り当て用のビットの数

(3) **ovs-subnet** プラグインに **redhat/openshift-ovs-subnet**、**ovs-multitenant** プラグインに **redhat/openshift-ovs-multitenant**、**ovs-networkpolicy** プラグインに **redhat/openshift-ovs-networkpolicy** をそれぞれ設定します。

(4) クラスターに割り当てられるサービス IP

また、複数の CIDR 範囲を持つ Pod ネットワークを作成することもできます。これは、個別の範囲をその範囲と **hostSubnetLength** を指定した **clusterNetworks** フィールドに追加して実行できます。

複数の範囲は同時に使用することができ、この範囲は拡張または縮小することが可能です。ノードは、ノードの退避、削除および再作成によってある範囲から別の範囲に移動できます。詳細は、「ノードの管理」セクションを参照してください。ノードの割り当ては一覧の順序で行われ、範囲が一杯になると以下の一覧に移行します。

複数 CIDR の Pod ネットワークの設定

```
networkConfig:
  clusterNetworks:
  - cidr: 10.128.0.0/14 (1)
    hostSubnetLength: 9 (2)
  - cidr: 10.132.0.0/14
    hostSubnetLength: 9
  externalIPNetworkCIDRs: null
```

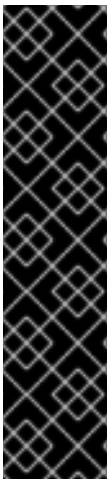
```
hostSubnetLength: 9
ingressIPNetworkCIDR: 172.29.0.0/16
networkPluginName: redhat/openshift-ovs-multitenant 3
serviceNetworkCIDR: 172.30.0.0/16
```

1 ノード IP の割り当て用クラスターネットワーク。

2 ノード内での Pod IP の割り当て用のビットの数

3 **ovs-subnet** プラグインは **redhat/openshift-ovs-subnet**、**ovs-multitenant** プラグインは **redhat/openshift-ovs-multitenant**、**ovs-networkpolicy** プラグインは **redhat/openshift-ovs-networkpolicy** をそれぞれ設定します。

clusterNetworks 値に要素を追加するか、その CIDR 範囲を使用しているノードがなければ要素を削除することができます。ただし変更を有効にするために、必ず **atomic-openshift-master-api** および **atomic-openshift-master-controllers** のサービスを再起動してください。



重要

hostSubnetLength の値は、クラスターの初回作成後に変更することができません。**cidr** は、ノードが範囲内に割り当てられている場合に最初のネットワークが含まれるより大きいネットワークに変更することのみ可能です。また、**serviceNetworkCIDR** は拡張可能です。たとえば、デフォルト値が **10.128.0.0/14** の場合、**cidr** を **10.128.0.0/9** (上半分の net 10 を参照) に変更することは可能ですが、**10.64.0.0/16** には元の値と重複しないために変更することができません。

serviceNetworkCIDR は **172.30.0.0/16** から **172.30.0.0/15** に変更できますが、**172.28.0.0/14** に変更できません。元の範囲が新規の範囲内にある場合でも、その範囲が CIDR の開始部分になければならないためです。詳細は、「サービスネットワークの拡張」を参照してください。

14.5. ノードでの POD ネットワークの設定

クラスター管理者は、ノード設定ファイルの **networkConfig** セクションにあるパラメーターを変更することで、ノード上の Pod ネットワーク設定を制御できます。

```
networkConfig:
  mtu: 1450 1
  networkPluginName: "redhat/openshift-ovs-subnet" 2
```

1 Pod オーバーレイネットワーク用の最大転送単位 (MTU)

2 **ovs-subnet** プラグインに **redhat/openshift-ovs-subnet**、**ovs-multitenant** プラグインに **redhat/openshift-ovs-multitenant**、**ovs-networkpolicy** プラグインに **redhat/openshift-ovs-networkpolicy** をそれぞれ設定します。

注記

OpenShift Container Platform SDN を構成するすべてのマスターおよび ノードで MTU サイズを変更する必要があります。また、tun0 インターフェースの MTU サイズはクラスターを構成するすべてのノードで同一である必要があります。

14.6. サービスネットワークの拡張

サービスネットワークのアドレスの低い方で実行している場合、現在の範囲が新規の範囲の開始部分にある限り、範囲を拡張することができます。

1. すべてのマスターの設定ファイルで **serviceNetworkCIDR** パラメーターを変更します (デフォルトは **/etc/origin/master/master-config.yaml**)。**/** の後の数のみをより小さい数に変更します。

2. **clusterNetwork** オブジェクトを削除します。

```
$ oc delete clusternetwork default
```

3. マスターホストサービスを再起動します。

```
# master-restart controllers
```

4. Ansible インベントリーファイルの **openshift_portal_net** 変数の値を新規の CIDR に更新します。

```
# Configure SDN cluster network and kubernetes service CIDR blocks.
These
# network blocks should be private and should not conflict with
network blocks
# in your infrastructure that pods may require access to. Can not be
changed
# after deployment.
#osm_cluster_network_cidr=10.1.0.0/16
#openshift_portal_net=172.30.0.0/<new_CIDR_range>
```

5. **config.yml** Ansible Playbook を実行してクラスターを再デプロイします。

```
# ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-master/config.yml
```

6. マスターインスタンスからか、またはクラスター管理者として、ノードからの Pod の退避を許可し、そのノードでの他の Pod のスケジューリングを無効にします。

```
$ NODE=ose-app-node01.example.com
$ oc adm manage-node ${NODE} --schedulable=false
NAME                           STATUS                     AGE
VERSION
ose-app-node01.example.com   Ready,SchedulingDisabled   20m
v1.6.1+5115d708d7

$ oc adm drain ${NODE} --ignore-daemonsets
node "ose-app-node01.example.com" already cordoned
pod "perl-1-build" evicted
pod "perl-1-3lnsh" evicted
pod "perl-1-9jzd8" evicted
node "ose-app-node01.example.com" drained
```

注記

移行されないローカルボリュームと共にコンテナーが実行されている場合には、以下のコマンドを実行します。**oc adm drain ${NODE} --ignore-daemonsets --delete-local-data**.

7. ノード上の Pod を一覧表示し、それらが削除されていることを確認します。

```
$ oc adm manage-node ${NODE} --list-pods

Listing matched pods on node: ose-app-node01.example.com

NAME      READY     STATUS    RESTARTS   AGE
```

8. ノードごとに、これまでの 2 つの手順を繰り返します。

Pod またはノードの退避またはドレイン (解放) についての詳細は、「ノードの保守」を参照してください。

14.7. SDN プラグイン間の移行

SDN プラグインをすでに 1 つ使用していて、別のプラグインへ切り替える場合には、以下を実行します。

1. 設定ファイル内のすべてのマスターとノードで**networkPluginName** パラメーターを変更します。
2. すべてのマスターで **atomic-openshift-master-api** および **atomic-openshift-master-controllers** サービスを再起動します。
3. すべてのマスターおよびノードで **atomic-origin-node** サービスを停止します。
4. OpenShift SDN プラグインを切り換える場合、すべてのマスターおよびノードで **openvswitch** サービスを再起動します。
5. すべてのマスターおよびノードで **atomic-openshift-node** サービスを停止します。
6. OpenShift SDN プラグインからサードパーティーのプラグインへ切り替える場合は、OpenShift SDN 固有のアーティファクトを消去します。

```
$ oc delete clusternetwork --all
$ oc delete hostsubnets --all
$ oc delete netnamespaces --all
```

ovs-subnet から **ovs-multitenant** OpenShift SDN プラグインに切り替えると、クラスター内の既存のすべてのプロジェクトが完全に分離します (つまり、これらに固有の VNID が割り当てられます)。クラスター管理者は、管理者 CLI を使用してプロジェクトネットワークの変更を選択できます。

以下を実行して VNID をチェックします。

```
$ oc get netnamespace
```

14.7.1. ovs-multitenant から ovs-networkpolicy への移行

「SDN プラグイン間の移行」セクションでの上記の一般的なプラグインの移行に加え、**ovs-multitenant** プラグインから **ovs-networkpolicy** プラグインへの移行にはもう 1 つの追加の手順があります。つまり、すべての namespace に **NetID**があることを確認する必要があります。これは、以前にプロジェクトを結合している場合や、プロジェクトをグローバルにしている場合に、**ovs-networkpolicy** プラグインに切り換える前にこのやり直しを実行する必要があります。そうしない場合には、NetworkPolicy オブジェクトが適切に機能しなくなる可能性があります。

ヘルパースクリプトを使うと、**NetID's** の修正、以前に分離した namespace を分離するための NetworkPolicy オブジェクトの作成、および以前に結合した namespace 間の接続の有効化を実行できます。

ovs-multitenant プラグインを実行した状態でヘルパースクリプトを使用して **ovs-networkpolicy** プラグインを移行するには、以下の手順に従ってください。

1. スクリプトをダウンロードし、実行ファイルパーミッションを追加します。

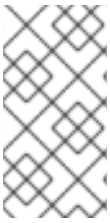
```
$ curl -O
https://raw.githubusercontent.com/openshift/origin/master/contrib/migration/migrate-network-policy.sh
$ chmod a+x migrate-network-policy.sh
```

2. スクリプトを実行します (クラスター管理者ロールが必要です)。

```
$ ./migrate-network-policy.sh
```

このスクリプトを実行すると、すべての namespace が他のすべての namespace から完全に分離されるので、**ovs-networkpolicy** プラグインへの移行が完了するまでは、異なる namespace にある Pod 間で接続を試行してもこれに失敗します。

新たに作成した namespace に同じポリシーをデフォルトで適用したい場合には、移行スクリプトで作成した **default-deny** および **allow-from-global-namespaces** ポリシーと一致するデフォルトの NetworkPolicy オブジェクトが作成されるように設定します。



注記

スクリプトが失敗するか他のエラーが発生した場合、または **ovs-multitenant** プラグインに戻す場合は、移行取り消し (un-migration) スクリプトを使用します。このスクリプトは移行スクリプトが実行した変更を取り消し、以前に結合した namespace を再度結合します。

14.8. クラスターネットワークへの外部アクセス

OpenShift Container Platform の外部にあるホストがクラスターネットワークへのアクセスを要求した場合、ユーザーには 2 つの選択肢があります。

1. ホストを OpenShift Container Platform ノードとして設定する。ただし、これには「スケジュール対象外 (unschedulable)」のマークを付け、マスターがこれにコンテナーをスケジュールできないようにします。

2. ユーザーのホストとクラスターネットワーク上のホストの間にトンネルを作成する。

上記のオプションはどちらもOpenShift SDN 内でのエッジロードバランサーからコンテナーへのルーティングを設定するための実践的なユースケースの一部として本ドキュメントで紹介されています。

14.9. FLANNEL の使用

デフォルトの SDN の代わりとして、OpenShift Container Platform は、**flannel** ベースのネットワークをインストールするための Ansible Playbook を提供しています。これは、SDN に依存しているクラウドプロバイダーのプラットフォーム (Red Hat OpenStack Platform など) で OpenShift Container Platform を実行していて、両方のプラットフォームでパケットのカプセル化を 2 回実行することを避ける場合に役立ちます。

Flannel は、すべてのコンテナーに単一の IP ネットワーク空間を使用し、隣接する空間のサブセットを各インスタンスに割り当てることを可能にします。これにより、コンテナーは何にも妨げられずに同じネットワーク空間内の任意の IP へ接続を試みることができます。これはネットワークを使ってアプリケーション内にあるコンテナーを別のアプリケーションから分離することができないために、マルチテナンシーを妨げます。

マルチテナンシーの分離とパフォーマンスのどちらを優先するかに応じて、内部ネットワークに OpenShift SDN (マルチテナンシー) と Flannel (パフォーマンス) のいずれか適切な方を選択する必要があります。

重要

Flannel は、Red Hat OpenStack Platform の OpenShift Container Platform のみでサポートされています。

重要

Neutron の現行バージョンはデフォルトで、各ポートに対するポートセキュリティーが有効になっているので、ポート自体のアドレスと異なる MAC アドレスを使ったパケットの送受信を実行できません。Flannel は、仮想の MAC アドレスと IP アドレスを作成し、パケットをポート上で送受信する必要があります。したがって、Flannel のトラフィックを実行するポートでは、ポートセキュリティーを無効にしておく必要があります。

OpenShift Container Platform クラスターで Flannel を無効にするには、以下を実行します。

1. Neutron のポートセキュリティーコントロールは、Flannel と互換性を持つように設定される必要があります。Red Hat OpenStack Platform のデフォルト設定では、**`port_security`** のユーザーコントロールは無効にされています。Neutron を、ユーザーが個々のポートで **`port_security`** 設定を制御できるように設定します。

   a. Neutron サーバーで、以下を **/etc/neutron/plugins/ml2/ml2_conf.ini** ファイルに追加します。

   ```
   [ml2]
   ...
   extension_drivers = port_security
   ```

   b. 次に、Neutron のサービスを再起動します。

   ```
   service neutron-dhcp-agent restart
   service neutron-ovs-cleanup restart
   service neutron-metadata-agentrestart
   service neutron-l3-agent restart
   service neutron-plugin-openvswitch-agent restart
   service neutron-vpn-agent restart
   service neutron-server  restart
   ```

2. Red Hat OpenStack Platform で OpenShift Container Platform インスタンスが作成されている間に、ポートのポートセキュリティーとセキュリティーグループの両方を無効にします。ここで、コンテナーネットワークの Flannel インターフェースは以下のようになります。

```
neutron port-update $port --no-security-groups --port-security-
enabled=False
```

注記

Flannel は、ノードのサブネットを設定し、割り当てるために etcd からの情報を収集します。したがって、etcd ホストに割り当てられるセキュリティーグループは、ノードからポート 2379/tcp へのアクセスを許可し、ノードのセキュリティーグループは etcd ホストでのそのポートへの egress 通信を許可する必要があります。

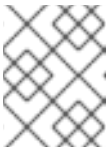
a. インストールを実行する前に以下の変数を Ansible インベントリーファイルに設定します。

```
openshift_use_openshift_sdn=false (1)
openshift_use_flannel=true (2)
flannel_interface=eth0
```

(1) **openshift_use_openshift_sdn** を **false** に設定してデフォルトの SDN を無効にします。

(2) **openshift_use_flannel** を **true** に設定して設定されている **flannel** を有効にします。

b. オプションで、**flannel_interface** 変数を使用してホスト間の通信に使用するインターフェースを指定することができます。この変数がない場合は、OpenShift Container Platform インストールではデフォルトのインターフェースが使用されます。



注記

Flannel を使用した Pod とサービスのカスタムのネットワーク CIDR は、今後のリリースでサポートされる予定です。**BZ#1473858**

3. OpenShift Container Platform をインストールした後、iptables の一連のルールを各 OpenShift Container Platform ノードに追加します。

```
iptables -A DOCKER -p all -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

これらの変更を **/etc/sysconfig/iptables** で永続化するには、すべてのノードで以下のコマンドを使用します。

```
cp /etc/sysconfig/iptables{,.orig}
sh -c "tac /etc/sysconfig/iptables.orig | sed -e '0,/:DOCKER -/
s/:DOCKER -/:DOCKER ACCEPT/' | awk '"\!"p && /POSTROUTING/{print \"-
A POSTROUTING -o eth1 -j MASQUERADE\"; p=1} 1' | tac >
/etc/sysconfig/iptables"
```

注記

iptables-save コマンドは、現在の **インメモリー** の iptables ルールをすべて保存します。ただし、Docker、Kubernetes、OpenShift Container Platform では永続化することが意図されていない iptables ルール (サービスなど) が多数作成されるために、これらのルールを保存すると問題が発生する可能性があります。

コンテナーのトラフィックを OpenShift Container Platform の残りのトラフィックから分離するには、分離されたテナントネットワークを作成し、すべてのノードをこれに割り当てることを推奨します。異なるネットワークインターフェース (eth1) を使用している場合は、インターフェースをブート時に **/etc/sysconfig/network-scripts/ifcfg-eth1** ファイルを使用して起動するように設定してください。

```
DEVICE=eth1
TYPE=Ethernet
BOOTPROTO=dhcp
ONBOOT=yes
DEFTROUTE=no
PEERDNS=no
```

第15章 NUAGE SDN の設定

15.1. NUAGE SDN と OPENSHIFT CONTAINER PLATFORM

Nuage Networks Virtualized Services Platform (VSP) は、仮想ネットワークとソフトウェアによるネットワーク制御 (SDN) インフラストラクチャーを Docker コンテナー環境に提供し、IT 運用を単純化して OpenShift Container Platform のネイティブなネットワーク機能を拡張します。

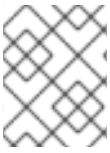
Nuage Networks VSP は、OpenShift Container Platform で実行される Docker ベースのアプリケーションに対応し、Pod と従来のワークロード間の仮想ネットワークのプロビジョニングを加速化し、セキュリティーポリシーをクラウドインフラストラクチャー全体で有効にします。また、セキュリティーアプライアンスによる、コンテナーアプリケーション用の詳細なセキュリティーとマイクロセグメンテーションポリシーの組み込みを自動化します。

VSP を OpenShift Container Platform のアプリケーションワークフローに統合することにより、DevOps チームが直面するネットワークのラグを取り除き、ビジネスアプリケーションのより迅速な調整と更新を可能にします。また、VSP は OpenShift Container Platform のさまざまなワークフローに対応し、ユーザーがポリシーベースの自動化によって使いやすさと完全な制御のいずれかを選択できる各種シナリオに対応します。

VSP を OpenShift Container Platform に統合する方法についての詳細は、「Networking」を参照してください。

15.2. 開発者のワークフロー

ワークフローは開発者環境で使用され、ネットワークのセットアップ時に開発者のインプットはほとんど必要ありません。ここでは **nuage-openshift-monitor** は、OpenShift Container Platform プロジェクトで作成される Pod の適切なポリシーとネットワークを提供するために必要な、VSP 構成 (ゾーン、サブネットなど) を作成します。プロジェクトが作成されると、**nuage-openshift-monitor** がそのプロジェクトのデフォルトのゾーンとデフォルトのサブネットを作成します。指定のプロジェクトに作成されたデフォルトのサブネットが使い果たされると、**nuage-openshift-monitor** が追加のサブネットを動的に作成します。



注記

プロジェクト間の分離を確保するため、各 OpenShift Container Platform プロジェクトに個別の VSP ゾーンが作成されます。

15.3. オペレーションワークフロー

このワークフローは、アプリケーションをロールアウトするオペレーションチームが使用します。このワークフローでは、まずネットワークとセキュリティーポリシーが、アプリケーションをデプロイするために組織が規定するルールに従って VSD に設定されます。管理ユーザーは、複数のゾーンとサブネットを作成し、ラベルを使ってそれらを同じプロジェクトにマップすることがあります。Pod をスピンアップする一方で、ユーザーは、Pod が接続すべきネットワークと、ポリシーが適用されるべきネットワークを Nuage Label を使って指定します。これにより、デプロイメントでプロジェクト間およびプロジェクト内のトラフィックを詳細に制御できます。たとえば、プロジェクト間の通信はプロジェクトベースで有効にされますが、この方法は、プロジェクトを共有プロジェクトにデプロイされている共通サービスに接続するために使用できます。

15.4. インストール

VSP と OpenShift Container Platform の統合は、仮想マシン (VM) とベアメタルの OpenShift Container Platform インストールの両方で機能します。

高可用性 (HA) を備えた環境は、複数マスターと複数ノードを使って設定することができます。

マルチマスターモードによる Nuage VSP 統合は、このセクションで説明するネイティブの HA 設定方法のみに対応しています。この統合は、負荷分散ソリューション (デフォルトは HAProxy) と組み合わせることもできます。インベントリーファイルには、3 つのマスターホスト、ノード、etcd サーバー、およびすべてのマスターホスト上でマスター API の負荷を分散するための HAProxy として機能するホストが含まれます。HAProxy ホストはインベントリーファイルの [lb] セクションに定義され、Ansible が HAProxy を負荷分散ソリューションとして自動的にインストールし、設定することを可能にします。

Ansible ノードファイルでは、Nuage VSP をネットワークプラグインとしてセットアップするために、以下のパラーメーターを指定する必要があります。

```
 # Create and OSEv3 group that contains masters, nodes, load-balancers,
and etcd hosts
 masters
 nodes
 etcd
 lb

 # Nuage specific parameters
 openshift_use_openshift_sdn=False
 openshift_use_nuage=True
 os_sdn_network_plugin_name='nuage/vsp-openshift'
 openshift_node_proxy_mode='userspace'

 # VSP related parameters
 vsd_api_url=https://192.168.103.200:8443
 vsp_version=v4_0
 enterprise=nuage
 domain=openshift
 vsc_active_ip=192.168.103.201
 vsc_standby_ip=192.168.103.202
 uplink_interface=eth0

 # rpm locations
 nuage_openshift_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/n
uage-openshift-monitor-4.0.X.1830.el7.centos.x86_64.rpm
 vrs_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-
openvswitch-4.0.X.225.el7.x86_64.rpm
 plugin_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/vsp-
openshift-4.0.X1830.el7.centos.x86_64.rpm

 # Required for Nuage Monitor REST server and HA
 openshift_master_cluster_method=native
 openshift_master_cluster_hostname=lb.nuageopenshift.com
 openshift_master_cluster_public_hostname=lb.nuageopenshift.com
 nuage_openshift_monitor_rest_server_port=9443

 # Optional parameters
 nuage_interface_mtu=1460
 nuage_master_adminusername='admin's user-name'
 nuage_master_adminuserpasswd='admin's password'
```

```
 nuage_master_cspadminpasswd='csp admin password'
 nuage_openshift_monitor_log_dir=/var/log/nuage-openshift-monitor

 # Required for brownfield install (where a {product-title} cluster exists
without Nuage as the networking plugin)
 nuage_dockker_bridge=lbr0

 # Specify master hosts
 [masters]
 fqdn_of_master_1
 fqdn_of_master_2
 fqdn_of_master_3

 # Specify load balancer host
 [lb]
 fqdn_of_load_balancer
```

第16章 KURYR SDN の設定

16.1. KURYR SDN および OPENSHIFT CONTAINER PLATFORM

重要

Kuryr SDN の設定機能はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

Kuryr (具体的には Kuryr-Kubernetes) は、CNI と OpenStack Neutron を使用して構築した SDN ソリューションです。Kuryr の利点として、幅広い Neutron SDN バックエンドを使用でき、Kubernetes Pod と OpenStack 仮想マシン (VM) の相互接続性が確保できる点が挙げられます。

Kuryr-Kubernetes と OpenShift Container Platform の統合は主に、OpenStack の仮想マシンで実行される OpenShift Container Platform クラスター用に設計されました。Kuryr-Kubernetes コンポーネントは OpenShift Container Platform の **`openshift-infra`** namespace に Pod としてインストールされます。

- kuryr-controller: 任意のノードにインストールされる単一のサービスインスタンスです。**`Deployment`** として OpenShift Container Platoform でモデリングされます。
- kuryr-cni: 各 OpenShift Container Platform ノードで Kuryr を CNI ドライバーとしてインストールし、設定するコンテナーです。**`DaemonSet`** として OpenShift Container Platoform でモデリングされます。

16.2. インストール

openshift-ansible を実行するシステムは OSP および OCP リポジトリーにサブスクライブしている必要があります。OpenStack 統合ではいくつかの追加パッケージが必要になります。依存関係をインストールするには、以下を実行します。

```
$ sudo yum install -y ansible openshift-ansible python2-shade python-dns \
    python2-heatclient python2-octaviaclient python-openstackclient bind-
utils
```

Kuryr-Kubernetes をネットワークプラグインとしてセットアップするために、Ansible ノードファイルで以下のパラメーターを指定します。

```
# Enable Kuryr.
openshift_use_openshift_sdn=False
openshift_use_kuryr=True
os_sdn_network_plugin_name=cni

# Set userspace so that there are no iptables remains.
openshift_node_proxy_mode='userspace'

# Disable management of the OpenShift Registry and OpenShift Router. If
```

```
you
 # need those services, install them manually after OpenShift deployment
using
 # `oadm`. This way services will get correct IPs provided by Kuryr.
 openshift_hosted_manage_registry=false
 openshift_hosted_manage_router=false

 # Keystone URL.
 kuryr_openstack_auth_url=http://127.0.0.1/identity

 # OpenStack domain name of user owning Kuryr resources.
 kuryr_openstack_user_domain_name=default

 # OpenStack project name of user owning Kuryr resources.
 kuryr_openstack_user_project_name=admin

 # OpenStack project id for Kuryr resources.
 kuryr_openstack_project_id=ec0b31802fd043c08bc15b74d2f9a3d3

 # OpenStack username that will own kuryr resources.
 kuryr_openstack_username=admin

 # Password for that user.
 kuryr_openstack_password=password

 # Default Neutron security groups' IDs for Kubernetes pods
 kuryr_openstack_pod_sg_id=f74c83a8-a520-421a-930e-21b6cd098c6a,01f85594-
9950-4ded-a92c-5ad546a41188

 # Default Neutron subnet ID for Kubernetes pods.
 kuryr_openstack_pod_subnet_id=c85cdee6-0ed1-4d8f-ae61-7afa4674b311

 # Default OpenStack project ID for Kubernetes resources.
 kuryr_openstack_pod_project_id=ec0b31802fd043c08bc15b74d2f9a3d3

 # Neutron subnet ID for Kubernetes worker node VMs.
 kuryr_openstack_worker_nodes_subnet_id=477cfa49-e641-4d31-a7b5-
5bc834743f61

 # Default Neutron subnet ID for Kubernetes services.
 kuryr_openstack_service_subnet_id=3b31a106-4084-4db9-bc0c-00b97afe186e
```

インストール前に、OpenShift Container Platform ノードが内部名の解決に使用する DNS サーバーを指定する必要もあります。OpenStack はノード名の解決を追加設定なしで提供しません。以下の例では、**10.20.30.40** は DNS サーバーの IP アドレスになります。

```
openshift_openstack_dns_nameservers=[10.20.30.40]
```

DNS サーバーが **nsupdate** (RFC 2136) でリモート更新をサポートする場合、以下の設定を追加すると、Playbook はこれを自動的に設定できます。

```
openshift_openstack_external_nsupdate_keys={private: {"key_secret": "
<nsupdate key>", "key_algorithm": "<nsupdate key algorithm>", "key_name": "
<nsupdate key name>", "server": 10.20.30.40}}
```

最後に、**provision_install.yml** Playbook を実行して OpenShift Container Platform をインストールします。動的インベントリーファイル (**inventory.py**) を指定し、作成した Ansible ノードのファイルのパスを指定する必要があります。

```
$ ansible-playbook --user openshift -i /usr/share/ansible/openshift-
ansible/playbooks/openstack/inventory.py -i ansible-nodes.txt
/usr/share/ansible/openshift-ansible/playbooks/openstack/openshift-
cluster/provision_install.yml
```

作成したノードでカスタム設定してから OpenShift Container Platform をインストールする場合には、**provision.yml** および **install.yml** Playbook を別に実行してください。**provision.yml** は OpenStack リソース (ノードやネットワークなど) を作成し、**install.yml** は OpenShift Container Platform をインストールします。

16.3. 検証

OpenShift Container Platform のインストールが完了したら、Kuryr Pod が正常にデプロイされているかどうかを確認できます。

```
$ oc -n openshift-infra get pods -o wide
NAME                                READY     STATUS    RESTARTS   AGE
IP              NODE
bootstrap-autoapprover-0            1/1       Running   0          3d
10.11.0.7       master-0.openshift.example.com
kuryr-cni-ds-66kt2                  2/2       Running   0          3d
192.168.99.14   infra-node-0.openshift.example.com
kuryr-cni-ds-ggcpz                  2/2       Running   0          3d
192.168.99.16   master-0.openshift.example.com
kuryr-cni-ds-mhzjt                  2/2       Running   0          3d
192.168.99.6    app-node-1.openshift.example.com
kuryr-cni-ds-njctb                  2/2       Running   0          3d
192.168.99.12   app-node-0.openshift.example.com
kuryr-cni-ds-v8hp8                  2/2       Running   0          3d
192.168.99.5    infra-node-1.openshift.example.com
kuryr-controller-59fc7f478b-qwk4k   1/1       Running   0          3d
192.168.99.5    infra-node-1.openshift.example.com
```

kuryr-cni Pod は、すべての OpenShift Container Platform ノードで実行する必要があります。単一の kuryr-controller インスタンスはこれらのノードのいずれかで実行する必要があります。

第17章 AMAZON WEB サービス (AWS) の設定

17.1. 概要

OpenShift Container Platform は、AWS ボリュームをアプリケーションデータの永続ストレージとして使用するなど、AWS EC2 インフラストラクチャーにアクセスできるように設定できます。これを実行するには、AWS を設定した後に OpenShift Container Platform ホストで追加の設定を行う必要があります。

17.2. パーミッション

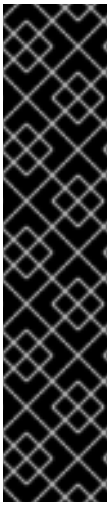
OpenShift Container Platform で AWS を設定するには以下のパーミッションが必要です。

表17.1 マスターのパーミッション

Elastic Compute Cloud(EC2)	**`ec2:DescribeVolume`、`ec2:CreateVolume`、`ec2:CreateTags`、`ec2:DescribeInstance`、`ec2:AttachVolume`、`ec2:DetachVolume`、`ec2:DeleteVolume`、`ec2:DescribeSubnets`、`ec2:CreateSecurityGroup`、`ec2:DescribeSecurityGroups`、`ec2:DescribeRouteTables`、`ec2:AuthorizeSecurityGroupIngress`、`ec2:RevokeSecurityGroupIngress`、`ec2:DeleteSecurityGroup`**
Elastic Load Balancing	**`elasticloadbalancing:DescribeTags`、`elasticloadbalancing:CreateLoadBalancerListeners`、`elasticloadbalancing:ConfigureHealthCheck`、`elasticloadbalancing:DeleteLoadBalancerListeners`、`elasticloadbalancing:RegisterInstancesWithLoadBalancer`、`elasticloadbalancing:DescribeLoadBalancers`、`elasticloadbalancing:CreateLoadBalancer`、`elasticloadbalancing:DeleteLoadBalancer`、`elasticloadbalancing:ModifyLoadBalancerAttributes`、`elasticloadbalancing:DescribeLoadBalancerAttributes`**

表17.2 ノードのパーミッション

Elastic Compute Cloud(EC2)	**`ec2:DescribeInstance*`**



重要

- マスターホスト、ノードホスト、サブネットにはすべて、**`kubernetes.io/cluster/<clusterid>,Value=(owned|shared)`** のタグが必要です。
- 1 つのセキュリティーグループ (ノードにリンクされていることが望ましい) に **`kubernetes.io/cluster/<clusterid>,Value=(owned|shared)`** タグが必要です。
  - すべてのセキュリティーグループに **`kubernetes.io/cluster/<clusterid>,Value=(owned|shared)`** のタグを付けないでください。その場合、Elastic Load Balancing (ELB) がロードバランサーを作成できなくなります。

17.3. セキュリティーグループの設定

AWS に OpenShift Container Platform をインストールする際は、適切なセキュリティーグループがセットアップされていることを確認してください。

以下は、セキュリティーグループに設定しておく必要のあるポートです。これらがないとインストールは失敗します。また、インストールするクラスターの設定によっては、追加のポートが必要になる場合があります。セキュリティーグループの詳細、およびその適切な調整方法については、「必要なポート」を参照してください。

すべての OpenShift Container Platform ホスト	• インストーラー/Ansible を実行しているホストの tcp/22
etcd セキュリティーグループ	• マスターの tcp/2379 • etcd ホストの tcp/2380
マスターのセキュリティーグループ	• tcp/8443 (0.0.0.0/0) • 3.2 よりも前にインストールされた環境、または 3.2 にアップグレードされた環境向けのすべての OpenShift Container Platform ホストの tcp/53 • 3.2 よりも前にインストールされた環境、または 3.2 にアップグレードされた環境向けのすべての OpenShift Container Platform ホストの udp/53 • 3.2 を使ってインストールされた新しい環境向けのすべての OpenShift Container Platform ホストの tcp/8053 • 3.2 を使ってインストールされた新しい環境向けのすべての OpenShift Container Platform ホストの udp/8053
ノードのセキュリティーグループ	• マスターの tcp/10250 • ノードの udp/4789
インフラストラクチャーノード (OpenShift Container Platform ルーターをホストできるノード)	• tcp/443 (0.0.0.0/0) • tcp/80 (0.0.0.0/0)
CRI-O	CRI-O を使用している場合は、tcp/10010 を開き、**`oc exec`** および **`oc rsh`** 操作を実行できるようにします。

マスターまたはルートの負荷分散のために外部のロードバランサー (ELB) を設定する場合は、ELB の Ingress および Egress のセキュリティーグループを設定することも必要になります。

17.3.1. 検出された IP アドレスとホスト名の上書き

AWS では、以下のような場合に変数の上書きが必要になります。

変数	使用法
hostname	ユーザーが **DNS hostnames** と **DNS resolution** の両方に対して設定されていない VPC にインストールしている。
ip	複数のネットワークインターフェースを設定しており、デフォルト以外のインターフェースを使用することを検討している。
public_hostname	• VPC サブネットが **Auto-assign Public IP** 向けに設定されていないマスターインスタンス。このマスターへの外部アクセスについては、ユーザーは ELB か、または必要な外部アクセスを提供する他のロードバランサーを使用する必要があります。または、VPN 接続を介してホストの内部名に接続する必要があります。 • メタデータが無効にされているマスターインスタンス。 • この値は、実際にはノードによって使用されません。
public_ip	• VPC サブネットが **Auto-assign Public IP** について設定されていないマスターインスタンス。 • メタデータが無効にされているマスターインスタンス。 • この値は、実際にはノードによって使用されません。

警告

openshift_hostname がメタデータで提供される **private-dns-name** 値以外の値に設定される場合、それらのプロバイダーのネイティブクラウド統合は機能しなくなります。

EC2 ホストの場合には特に、**DNS host names** と **DNS resolution** の両方が有効にされている VPC にデプロイされる必要があります。**openshift_hostname** は上書きできません。

17.4. AWS 変数の設定

必要な AWS 変数を設定するには、OpenShift Container Platform のマスターとノード両方のすべてのホストに、**/etc/origin/cloudprovider/aws.conf**ファイルを以下の内容で作成します。

```
[Global]
Zone = us-east-1c 1
```

1 これは AWS インスタンスのアベイラビリティーゾーンであり、EBS ボリュームがある場所です。この情報は AWS マネジメントコンソールから取得されます。

17.5. AWS についての OPENSHIFT CONTAINER PLATFORM マスターの設定

AWS は OpenShift Container Platform のマスターホストに 2 通りの方法で設定できます。

- Ansible の使用
- **master-config.yaml** ファイルの手動変更

17.5.1. Ansible を使用した AWS についての OpenShift Container Platform の設定

クラスターのインストール時に、**openshift_cloudprovider_aws_access_key**、**openshift_cloudprovider_aws_secret_key**、**openshift_cloudprovider_kind**、**openshift_clusterid** のパラメーターを使用して、AWS を設定できます。これは、インベントリーファイルで設定可能です。

Ansible を使用した AWS の設定例

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
#openshift_cloudprovider_aws_access_key="{{
lookup('env','AWS_ACCESS_KEY_ID') }}"
#openshift_cloudprovider_aws_secret_key="{{
lookup('env','AWS_SECRET_ACCESS_KEY') }}"
#
#openshift_clusterid=unique_identifier_per_availablility_zone
#
# AWS (Using API Credentials)
#openshift_cloudprovider_kind=aws
#openshift_cloudprovider_aws_access_key=aws_access_key_id
#openshift_cloudprovider_aws_secret_key=aws_secret_access_key
#
# AWS (Using IAM Profiles)
#openshift_cloudprovider_kind=aws
# Note: IAM roles must exist before launching the instances.
```

注記

Ansible が AWS を設定する際に、以下のファイルがユーザー向けに作成されます。

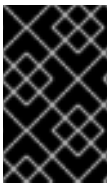
- **/etc/origin/cloudprovider/aws.conf**
- **/etc/origin/master/master-config.yaml**
- **/etc/origin/node/node-config.yaml**
- **/etc/sysconfig/atomic-openshift-master-api**
- **/etc/sysconfig/atomic-openshift-master-controllers**
- **/etc/sysconfig/atomic-openshift-node**

17.5.2. 手動による AWS についての OpenShift Container Platform マスターの設定

すべてのマスターで、マスター設定ファイル (デフォルトは **/etc/origin/master/master-config.yaml**) を編集するか、または作成し、**apiServerArguments** と **controllerArguments** の各セクションの内容を更新します。

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/origin/cloudprovider/aws.conf"
  controllerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/origin/cloudprovider/aws.conf"
```

現時点では、クラウドプロバイダーの統合を正常に機能させるため、**nodeName** は AWS のインスタンス名と**一致している必要があります**。また、この名前は RFC1123 に準拠している必要もあります。



重要

コンテナー化インストールをトリガーする場合、**/etc/origin** と **/var/lib/origin** のディレクトリーのみがマスターとノードのコンテナーにマウントされます。したがって、**aws.conf** は **/etc/** ではなく **/etc/origin/** になければなりません。

17.5.3. 手動による AWS についての OpenShift Container Platform ノードの設定

適切なノード設定マップを編集して、**kubeletArguments** セクションの内容を更新します。

```
kubeletArguments:
  cloud-provider:
    - "aws"
  cloud-config:
    - "/etc/origin/cloudprovider/aws.conf"
```

重要

コンテナー化インストールをトリガーする場合、**/etc/origin** と **/var/lib/origin** のディレクトリーのみがマスターとノードのコンテナーにマウントされます。したがって、**aws.conf** は **/etc/** ではなく **/etc/origin/** になければなりません。

17.6. キーと値のアクセスペアの設定

以下の環境変数が、マスターの **/etc/sysconfig/atomic-openshift-master-api** ファイルと **/etc/sysconfig/atomic-openshift-master-controllers** ファイル、およびノードの **/etc/sysconfig/atomic-openshift-node** ファイルに設定されていることを確認します。

```
AWS_ACCESS_KEY_ID=<key_ID>
AWS_SECRET_ACCESS_KEY=<secret_key>
```

注記

アクセスキーは、AWS IAM ユーザーを設定する際に取得されます。

17.7. 設定変更の適用

マスターおよびノードのすべてのホストで OpenShift Container Platform サービスを起動または再起動し、設定の変更を適用します。「OpenShift Container Platform サービスの再起動」を参照してください。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

クラウドプロバイダーを使用しない状態から、使用するように切り替えると、エラーメッセージが表示されます。クラウドプロバイダーを追加すると、ノードが **hostname** を **`externalID`** として使用する (クラウドプロバイダーが使用されていなかった場合のケース) 代わりに、クラウドプロバイダーの **`instance-id`** (クラウドプロバイダーによって指定される) の使用に切り替えるため、ノードの削除が試みられます。この問題を解決するには、以下を実行します。

1. CLI にクラスター管理者としてログインします。
2. 既存のノードラベルをチェックし、これらをバックアップします。

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?
=Taints)'
```

3. ノードを削除します。

```
$ oc delete node <node_name>
```

4. 各ノードホストで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

5. 以前に使用していた各ノードのラベルを再度追加します。

17.8. クラスターに対する AWS のラベリング

OpenShift Container Platform バージョン 3.7 以降の **atomic-openshift-installer** では、AWS プロバイダー認証情報を設定している場合に、すべてのホストにラベルが付けられていることも確認する必要があります。

クラスターに関連付けられているリソースを正確に特定するには、キー **kubernetes.io/cluster/<clusterid>** でリソースにタグを付けます。

- **<clusterid>** はクラスターに固有の名前です。

該当の値を、ノードがこのクラスターだけに所属する場合には **owned** に、また、他のシステムとリソースが共有されている場合には **shared** に設定します。

すべてのリソースに **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** タグを付けることで、複数ゾーンまたは複数クラスターに関連する潜在的な問題を回避できます。

注記

OpenShift Container Platform 3.6 よりも前のバージョンでは、これは **Key=KubernetesCluster,Value=clusterid** でした。

OpenShift Container Platform へのラベル付けとタグ付けに関する詳細は、「Pod およびサービス」を参照してください。

17.8.1. タグを必要とするリソース

タグ付けが必要なリソースは以下の 4 種類です。

- インスタンス
- セキュリティーグループ
- ロードバランサー
- EBS ボリューム

17.8.2. 既存クラスターへのタグ付け

クラスターは **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** タグの値を使用して、AWS クラスターに所属するリソースを判断します。したがって、関連のリソースはすべて、そのキーの同じ値を使用して **kubernetes.io/cluster/<clusterid>,Value=(owned|shared)** タグでラベルを付ける必要があります。これらのリソースには以下が含まれます。

- 全ホスト
- AWS インスタンスで使用する関連のロードバランサーすべて
- EBS ボリュームすべて。タグ付けが必要な EBS ボリュームは以下を使用して見つけることができます。

```
$ oc get pv -o json|jq '.items[].spec.awsElasticBlockStore.volumeID'
```

- AWS インスタンスで使用する関連のセキュリティーグループすべて

注記

すべての既存セキュリティーグループに **kubernetes.io/cluster/<name>,Value=<clusterid>** のタグを付けないでください。その場合、Elastic Load Balancing (ELB) がロードバランサーを作成できなくなります。

リソースにタグを付けた後に、マスターサービスをマスター上で、ノードサービスをすべてのノード上で再起動します。「設定変更の適用」を参照してください。

第18章 OPENSTACK の設定

18.1. 概要

OpenShift Container Platform は、OpenStack にデプロイする場合に、アプリケーションデータ用にOpenStack Cinder ボリュームを永続ストレージとして使用するなど、OpenStack インフラストラクチャーにアクセスするように設定できます。

18.2. パーミッション

OpenShift Container Platform に OpenStack を設定するには、以下のロールが必要です。

メンバー

インスタンス、ネットワーキングポート、Floating IP、ボリュームなどのアセットを作成するために、テナントの member ロールが必要になります。

18.3. セキュリティーグループの設定

OpenStack に OpenShift Container Platform を インストールする際は、適切なセキュリティーグループがセットアップされていることを確認します。

以下は、セキュリティーグループに設定しておく必要のあるポートです。これらがないとインストールは失敗します。また、インストールするクラスターの設定によっては、追加のポートが必要になる場合があります。セキュリティーグループの詳細、およびその適切な調整方法については、「必要なポート」を参照してください。

すべての OpenShift Container Platform ホスト	• インストーラー/Ansible を実行しているホストの tcp/22
etcd セキュリティーグループ	• マスターの tcp/2379 • etcd ホストの tcp/2380
マスターのセキュリティーグループ	• tcp/8443 (0.0.0.0/0) • 3.2 よりも前にインストールされた環境、または 3.2 にアップグレードされた環境向けのすべての OpenShift Container Platform ホストの tcp/53 • 3.2 よりも前にインストールされた環境、または 3.2 にアップグレードされた環境向けのすべての OpenShift Container Platform ホストの udp/53 • 3.2 を使ってインストールされた新しい環境向けのすべての OpenShift Container Platform ホストの tcp/8053 • 3.2 を使ってインストールされた新しい環境向けのすべての OpenShift Container Platform ホストの udp/8053

ノードのセキュリティーグループ	• マスターの tcp/10250 • ノードの udp/4789
インフラストラクチャーノード (OpenShift Container Platform ルーターをホストできるノード)	• tcp/443 (0.0.0.0/0) • tcp/80 (0.0.0.0/0)
CRI-O	CRI-O を使用している場合は、tcp/10010 を開き、**`oc exec`** および **`oc rsh`** 操作を実行できるようにします。

マスターまたはルートの負荷分散のために外部のロードバランサー (ELB) を設定する場合は、ELB の Ingress および Egress のセキュリティーグループを設定することも必要になります。

18.4. OPENSTACK 変数の設定

必要な OpenStack 変数を設定するには、OpenShift Container Platform のマスターとノード両方のすべてのホストに、**/etc/cloud.conf** ファイルを以下の内容で作成します。

```
[Global]
auth-url = <OS_AUTH_URL>
username = <OS_USERNAME>
password = <password>
domain-id = <OS_USER_DOMAIN_ID>
tenant-id = <OS_TENANT_ID>
region = <OS_REGION_NAME>

[LoadBalancer]
subnet-id = <UUID of the load balancer subnet>
```

OS_ 変数の値については OpenStack の管理者にお問い合わせください。この値は通常 OpenStack の設定で使用されます。

18.5. OPENSTACK についての OPENSHIFT CONTAINER PLATFORM マスターの設定

OpenStack は、OpenShift Container Platform のマスターとノードの各ホストに以下の 2 通りの方法で設定できます。

- Ansible の使用
- **master-config.yaml** ファイルと適切なノード設定マップを手動で変更します。
  === Ansible を使用した OpenStack についての OpenShift Container Platform の設定

クラスターのインストール時に、OpenStack は以下のパラメーターを使用して設定できます。これらのパラメーターは、インベントリーファイルで設定可能です。

- **`openshift_cloudprovider_kind`**

- **openshift_cloudprovider_openstack_auth_url**
- **openshift_cloudprovider_openstack_username**
- **openshift_cloudprovider_openstack_password**
- **openshift_cloudprovider_openstack_domain_id**
- **openshift_cloudprovider_openstack_domain_name**
- **openshift_cloudprovider_openstack_tenant_id**
- **openshift_cloudprovider_openstack_tenant_name**
- **openshift_cloudprovider_openstack_region**
- **openshift_cloudprovider_openstack_lb_subnet_id**

Ansible を使用した OpenStack の設定例

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
#openshift_cloudprovider_openstack_username="{{ lookup('env','USERNAME')
}}"
#openshift_cloudprovider_openstack_password="{{ lookup('env','PASSWORD')
}}"
#
# Openstack
#openshift_cloudprovider_kind=openstack
#openshift_cloudprovider_openstack_auth_url=http://openstack.example.com:3
5357/v2.0/
#openshift_cloudprovider_openstack_username=username
#openshift_cloudprovider_openstack_password=password
#openshift_cloudprovider_openstack_domain_id=domain_id
#openshift_cloudprovider_openstack_domain_name=domain_name
#openshift_cloudprovider_openstack_tenant_id=tenant_id
#openshift_cloudprovider_openstack_tenant_name=tenant_name
#openshift_cloudprovider_openstack_region=region
#openshift_cloudprovider_openstack_lb_subnet_id=subnet_id
```

18.5.1. 手動による OpenStack についての OpenShift Container Platform マスターの設定

すべてのマスターで、マスター設定ファイル (デフォルトは **/etc/origin/master/master-config.yaml**) を編集するか、または作成し、**apiServerArguments** と **controllerArguments** の各セクションの内容を更新します。

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "openstack"
```

```
    cloud-config:
      - "/etc/cloud.conf"
  controllerArguments:
    cloud-provider:
      - "openstack"
    cloud-config:
      - "/etc/cloud.conf"
```

重要

コンテナー化インストールをトリガーすると、**/etc/origin** と **/var/lib/origin** のディレクトリーのみがマスターとノードのコンテナーにマウントされます。したがって、**cloud.conf** は **/etc/** ではなく **/etc/origin/** になければなりません。

18.5.2. 手動による OpenStack についての OpenShift Container Platform ノードの設定

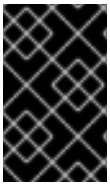
適切なノード設定マップを編集して、**kubeletArguments** と **nodeName** セクションの内容を更新します。

```
nodeName:
  <instance_name> 1

kubeletArguments:
  cloud-provider:
    - "openstack"
  cloud-config:
    - "/etc/cloud.conf"
```

1 ノードホストの OpenStack インスタンス名 (RFC1123 に準拠)

注記

nodeName は OpenStack インスタンス名と一致しない場合に、クラウドプロバイダーの統合は機能しません。



重要

コンテナー化インストールをトリガーすると、**/etc/origin** と **/var/lib/origin** のディレクトリーのみがマスターとノードのコンテナーにマウントされます。したがって、**cloud.conf** は **/etc/** ではなく **/etc/origin/** になければなりません。

18.6. 設定変更の適用

マスターおよびノードのすべてのホストで OpenShift Container Platform サービスを起動または再起動し、設定の変更を適用します。「OpenShift Container Platform サービスの再起動」を参照してください。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

クラウドプロバイダーを使用しない状態から、使用するように切り替えると、エラーメッセージが表示されます。クラウドプロバイダーを追加すると、ノードが **hostname** を **externalID** として使用する (クラウドプロバイダーが使用されていなかった場合のケース) 代わりに、クラウドプロバイダーの **instance-id** (クラウドプロバイダーによって指定される) の使用に切り替えるため、ノードの削除が試みられます。この問題を解決するには、以下を実行します。

1. CLI にクラスター管理者としてログインします。
2. 既存のノードラベルをチェックし、これらをバックアップします。

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?
=Taints)'
```

3. ノードを削除します。

```
$ oc delete node <node_name>
```

4. 各ノードホストで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

5. 以前に使用していた各ノードのラベルを再度追加します。

18.6.1. 動的に作成した OpenStack PV のゾーンラベルの設定

管理者は、動的に作成された OpenStack PV のゾーンラベルを設定できます。このオプションは、OpenStack Cinder ゾーン名がコンピュートゾーン名などに一致しない場合、Cinder ゾーンが 1 つだけで、コンピュートゾーンが多数ある場合に有用です。管理者は、動的に Cinder ボリュームを作成してから、ラベルをチェックできます。

PV のゾーンラベルを表示します。

```
# oc get pv --show-labels
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS    CLAIM                 STORAGECLASS   REASON
AGE       LABELS
pvc-1faa6f93-64ac-11e8-930c-fa163e3c373c   1Gi        RWO
Delete           Bound     openshift-node/pvc1   standard
12s       failure-domain.beta.kubernetes.io/zone=nova
```

デフォルトの設定が有効になっています。**oc get pv --show-labels** コマンドは、**failure-domain.beta.kubernetes.io/zone=nova** ラベルを返します。

ゾーンラベルを無効にするには、cloud.conf ファイルに以下を追加して更新します。

```
[BlockStorage]
ignore-volume-az = yes
```

マスターサービスの再起動後に作成された PV にはゾーンラベルがありません。

第19章 GOOGLE COMPUTE ENGINE の設定

アプリケーションデータ用に 永続ストレージとして GCE ボリュームを使用するなど OpenShift Container Platform が既存の Google Compute Engine (GCE) インフラストラクチャー にアクセスするように設定します。

19.1. 作業を開始する前の注意事項

19.1.1. Google Cloud Platform での認証設定

ロール

OpenShift Container Platform に GCP を設定するには、以下の GCP ロールが必要です。

roles/owner	サービスアカウント、クラウドストレージ、インスタンス、イメージ、テンプレート、Cloud DNS エントリーの作成や、ロードバランサーとヘルスチェックのデプロイに必要です。

ユーザーがテストフェーズ中に環境の再デプロイを想定している場合には、**delete** パーミッションが必要な場合もあります。

サービスアカウントを使用することで、GCP オブジェクトのデプロイ時に個人ユーザーの使用を回避することもできます。

ロールの設定方法に関する手順など、詳細は、GCP ドキュメントのロールの理解のセクションを参照してください。

スコープおよびサービスアカウント

GCP はスコープを使用して、承認済みのアイデンティティーがリソース内での操作を許可されているかどうかを判断します。たとえば、読み取り専用のスコープのアクセストークンを持つアプリケーション A は、読み取りだけできますが、読み取り/書き込みスコープのアクセストークンを持つアプリケーション B はデータの読み取りと変更が可能です。

スコープは、GCP API レベルで **https://www.googleapis.com/auth/compute.readonly** として定義されます。

インスタンスの作成時に **--scopes=[SCOPE,…]** オプションを使用してスコープを指定するか、インスタンスが GCP API にアクセスしないようにする場合には、**--no-scopes** オプションを使用してスコープなしでインスタンスを作成できます。

詳細情報は、GCP ドキュメントのスコープのセクション を参照してください。

GCP の全プロジェクトには、プロジェクトエディターパーミッションの付いた **[PROJECT_NUMBER]-compute@developer.gserviceaccount.com** サービスアカウントが含まれています。

デフォルトでは、新規作成されたインスタンスは自動的に有効化されて以下のアクセススコープが割り当てられたデフォルトのサービスアカウントとして実行されます。

- https://www.googleapis.com/auth/devstorage.read_only
- https://www.googleapis.com/auth/logging.write

- https://www.googleapis.com/auth/monitoring.write
- https://www.googleapis.com/auth/pubsub
- https://www.googleapis.com/auth/service.management.readonly
- https://www.googleapis.com/auth/servicecontrol
- https://www.googleapis.com/auth/trace.append
- https://www.googleapis.com/auth/bigquery
- https://www.googleapis.com/auth/cloud-platform
- https://www.googleapis.com/auth/compute.readonly
- https://www.googleapis.com/auth/compute
- https://www.googleapis.com/auth/datastore
- https://www.googleapis.com/auth/logging.write
- https://www.googleapis.com/auth/monitoring
- https://www.googleapis.com/auth/monitoring.write
- https://www.googleapis.com/auth/servicecontrol
- https://www.googleapis.com/auth/service.management.readonly
- https://www.googleapis.com/auth/sqlservice.admin
- https://www.googleapis.com/auth/devstorage.full_control
- https://www.googleapis.com/auth/devstorage.read_only
- https://www.googleapis.com/auth/devstorage.read_write
- https://www.googleapis.com/auth/taskqueue
- https://www.googleapis.com/auth/userinfo.email

インスタンスの作成時に、**`--service-account=SERVICE_ACCOUNT`** オプションで別のサービスアカウントを指定するか、**`gcloud`** CLI で **`--no-service-account`** オプションを使用してインスタンスのサービスアカウントを明示的に無効化します。

詳細情報は、GCP ドキュメントの新規サービスアカウントの作成セクション を参照してください。

19.1.2. Google Compute Engine オブジェクト

OpenShift Container Platform と Google Compute Engine (GCE) を統合するには、以下のコンポーネントまたはサービスが必要です。

GCP プロジェクト

GCP プロジェクトは、全 GCP サービスの作成、有効化、使用の基盤を形成するベースレベルの組織エンティティーです。これには、API の管理、課金の有効化、コラボレーターの追加/削除、パーミッションの管理が含まれます。

詳細情報は、GCP ドキュメントのプロジェクトリソースセクション を参照してください。

重要

プロジェクト ID は一意識別子で、Google Cloud Engine すべてで一意でなければなりません。つまり、`myproject` という名前のプロジェクトがすでに作成されている場合には、このプロジェクト ID を使用できません。

課金

アカウントに課金がアタッチされていない限り、新規リソースを作成できません。新規プロジェクトは、既存のプロジェクトにリンクすることも、新規情報を入力することもできます。

詳細情報は、GCP ドキュメントの請求アカウントの作成、変更、終了 を参照してください。

クラウドのアイデンティティーおよびアクセス管理

OpenShift Container Platform のデプロイには、適切なパーミッションが必要です。ユーザーは、サービスアカウント、クラウドストレージ、インスタンス、イメージ、テンプレート、Cloud DNS エントリーの作成、ロードバランサーやヘルスチェックのデプロイができる必要があります。テスト中に環境を再デプロイできるようにするには、削除のパーミッションが役立ちます。

特定のパーミッションを割り当ててサービスアカウントを作成し、このアカウントを使用して、通常のユーザーではなくインフラストラクチャーコンポーネントをデプロイします。また、異なるユーザーまたはサービスアカウントへのアクセスを制限するためのロールを作成することも可能です。

GCP インスタンスは、サービスアカウントを使用して、アプリケーションが GCP API を呼び出せるようにします。たとえば、OpenShift Container Platform ノードホストは、GCP ディスク API を呼び出して、アプリケーションに永続ボリュームを提供することができます。

IAM サービスを使用することで、さまざまなインフラストラクチャーやサービスリソースへのアクセス制御、粒度の細かいロールを利用できます。詳細情報は、GCP ドキュメントのクラウドの概要へのアクセスのセクションを参照してください。

SSH キー

GCP は、作成したインスタンスで SSH を使用してログインできるように、SSH 公開キーを認証キーとして注入します。SSH キーは、インスタンス別またはプロジェクト別に設定できます。

また、既存の SSH キーを使用できます。GCP メタデータは、SSH アクセスができるように、起動時にインスタンスに注入される SSH キーの保存に役立ちます。

詳細情報は、GCP ドキュメントのメタデータセクションを参照してください。

GCP リージョンおよびゾーン

GCP には、リージョンとアベイラビリティーゾーンに対応するグローバルインフラストラクチャーがあります。GCP にある OpenShift Container Platform を異なるゾーンにデプロイすると、単一障害点を回避できますが、ストレージに関して注意点があります。

GCP ディスクは、1 つのゾーン内に作成されるので、OpenShift Container Platform ノードのホストがゾーン A でダウンし、Pod がゾーン B に移動した場合に、ディスクが異なるゾーンに配置されているので、永続ストレージはこれらの Pod にアタッチできません。

OpenShift Container Platform をインストールする前に、複数のゾーンからなる OpenShift Container Platform 環境のゾーン 1 つをデプロイするかどうか判断するのは重要です。マルチゾーン環境をデプロイする場合には、単一のリージョンに 3 つの異なるゾーンを使用する設定が推奨されます。

詳細情報は、GCP ドキュメントのリージョンとゾーン および Kubernetes ドキュメントの複数ゾーン を参照してください。

外部 IP アドレス

GCP インスタンスがインターネットと通信できるように、インスタンスに外部 IP アドレスをアタッチする必要があります。また、外部 IP アドレスは、Virtual Private Cloud (VPC) ネットワーク外から、GCP にデプロイされたインスタンスと通信するのに必要です。

警告

インターネットアクセスに **外部 IP アドレス** が必要になるので、プロバイダーには制限となります。受信トラフィックが必要ない場合には、ファイアウォールルールを設定して、インスタンスで受信する外部トラフィックをブロックすることができます。

詳細情報は、GCP ドキュメントの外部 IP アドレスを参照してください。

クラウド DNS

GCP クラウド DNS は、GCP DNS サーバーを使用してドメイン名をグローバル DNS に公開するために使用する DNS サービスです。

パブリッククラウドの DNS ゾーンには、Google の「ドメイン」サービスまたはサードパーティーのプロバイダーを使用して購入したドメイン名を使用する必要があります。ゾーンを作成する時に、Google が提供するネームサーバーをレジストラーに追加 する必要があります。

詳細情報は、GCP ドキュメントの Cloud DNS セクションを参照してください。

注記

GCP VPC ネットワークには、内部ホスト名を自動的に解決する内部の DNS サービスがあります。

インスタンスに対する内部の完全修飾ドメイン名 (FQDN) は、**`[HOST_NAME].c.[PROJECT_ID].internal`** 形式に従います。

詳細情報は、GCP ドキュメントの内部 DNS を参照してください。

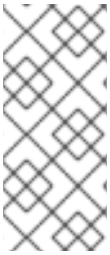
負荷分散

GCP 負荷分散サービスにより、GCP クラウド内の複数のインスタンスに、トラフィックを分散することができます。

負荷分散には 5 つのタイプがあります。

- 内部
- ネットワーク負荷分散
- HTTP(S) 負荷分散
- SSL Proxy 負荷分散

- TCP Proxy 負荷分散



注記

HTTPS および TCP Proxy 負荷分散は、マスターノードに HTTPS ヘルスチェックを使用する唯一の方法で、**/healthz** のステータスを確認します。

HTTPS 負荷分散には、カスタムの証明書が必要なので、この実装は、TCP Proxy 負荷分散を使用して、このプロセスを簡素化します。

詳細情報は、GCP ドキュメントの負荷分散 を参照してください。

インスタンスサイズ

正常な OpenShift Container Platform の環境には、最低でも以下のハードウェア要件を満たす必要があります。

表19.1 インスタンスサイズ

ロール	サイズ
マスター	**n1-standard-8**
ノード	**n1-standard-4**

GCP では、カスタムのインスタンスサイズを作成して、異なる要件に適合できるようにします。詳細は、「カスタムのマシンタイプでのインスタンス作成」を参照するか、インスタンスサイズに関する詳細は、「マシンタイプ」および「OpenShift Container Platform のハードウェア最小要件」を参照してください。

ストレージオプション

デフォルトでは、GCP インスタンスごとに、オペレーティングシステムを含む小規模な Root 永続ディスクが含まれます。インスタンスで実行するアプリケーションで、より多くのストレージ容量が必要な場合に、このインスタンスにさらにストレージオプションを追加できます

- 標準の永続ディスク
- SSD 永続ディスク
- ローカル SSD
- クラウドストレージバケット

詳細情報は、GCP ドキュメントのストレージオプション を参照してください。

19.2. GCE 向けの OPENSHIFT CONTAINER PLATFORM の設定

OpenShift Container Platform は、GCE 用に 2 種類の方法で設定できます。

- Ansible の使用
- 手動での **master-config.yaml** ファイルの変更

19.2.1. オプション 1: Ansible を使用した GCP 向けの OpenShift Container Platform の設定

Google Compute Platform (GCP) 向けの OpenShift Container Platform は、インストール時またはインストール後に、Ansible インベントリーファイルを変更することで設定できます。

手順

1. 最低でも、**openshift_cloudprovider_kind**, **openshift_gcp_project** と **openshift_gcp_prefix** のパラメーター、マルチゾーンのデプロイメントにはオプションで **openshift_gcp_multizone**、デフォルトのネットワーク名を使用しない場合は **openshift_gcp_network_name** を定義する必要があります。
インストール時に Ansible インベントリーファイルに以下のセクションを追加して、GCP 用に OpenShift Container Platform 環境を設定します。

```
[OSEv3:vars]
openshift_cloudprovider_kind=gce
openshift_gcp_project=<projectid> 1
openshift_gcp_prefix=<uid> 2
openshift_gcp_multizone=False 3
openshift_gcp_network_name=<network name> 4
```

1. 既存のインスタンスを実行している GCP プロジェクト ID を指定します。この ID は、Google Cloud Platform Console でプロジェクトを作成すると生成されます。
2. 一意の文字列を指定して、各 OpenShift Container Platform クラスターを特定します。これは、GCP 全体で一意でなければなりません。
3. オプションで **True** の設定して、GCP でのマルチゾーンのデプロイメントをトリガーします。デフォルトでは **False** に設定されています。
4. オプションで、**default** ネットワークを使用しない場合に、ネットワーク名を指定します。

Ansible でインストールすると、GCP 環境に適合されるように、以下のファイルが作成されて設定されます。

- **/etc/origin/cloudprovider/gce.conf**
- **/etc/origin/master/master-config.yaml**
- **/etc/origin/node/node-config.yaml**

2. GCP を使用してロードバランサーサービスを実行 する場合は、Compute Engine VM ノードインスタンスには **ocp** のサフィックスが必要です。たとえば、**openshift_gcp_prefix** パラメーターの値を **mycluster** に設定する場合は、このノードに **myclusterocp** のタグを付ける必要があります。Compute Engine VM インスタンスにネットワークタグを追加する方法に関する詳細は、「ネットワークタグの追加と削除」を参照してください。

3. オプションで、マルチゾーンサポートを設定できます。
クラスターのインストールプロセスでは、単一ゾーンのサポートがデフォルトで設定されますが、単一障害点を避けるためにマルチゾーンを設定することができます。

GCP ディスクがゾーン内に作成されるので、異なるゾーンで GCP に OpenShift Container

Platform をデプロイすると、ストレージで問題が発生する可能性があります。OpenShift Container Platform ノードのホストがゾーン A でダウンし、Pod がゾーン B に移動した場合に、ディスクが異なるゾーンに配置されているので、永続ストレージはこれらの Pod にアタッチできません。詳細情報は、Kubernetes ドキュメントの マルチゾーンの制限 を参照してください。

Ansible インベントリーを使用してマルチゾーンサポートを有効にするには、以下のパラメーターを追加します。

```
[OSEv3:vars]
openshift_gcp_multizone=true
```

シングルゾーンサポートに戻すには、**openshift_gcp_multizone** の値を **false** に設定して、Ansible インベントリーファイルに戻ります。

19.2.2. オプション 2: GCE 用の OpenShift Container Platform の手動設定

19.2.2.1. GCE 向けのマスターホストの手動設定

全マスターホストで以下の手順を実行します。

手順

1. デフォルトでは、**/etc/origin/master/master-config.yaml** のマスター設定ファイルの **apiServerArguments** と **controllerArguments** に GCE パラメーターを追加します。

```
apiServerArguments:
  cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"*
controllerArguments:
  cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"*
```

2. Ansible を使用して GCP 用に OpenShift Container Platform を設定する場合には、**/etc/origin/cloudprovider/gce.conf** ファイルは自動的に作成されます。GCP 用の OpenShift Container Platform を手動で設定するので、このファイルを作成して、以下を入力する必要があります。

```
[Global]
project-id = <project-id> 1
network-name = <network-name> 2
node-tags = <node-tags> 3
node-instance-prefix = <instance-prefix> 4
multizone = true 5
```

1 既存のインスタンスを実行している GCP プロジェクト ID を指定します。

2 デフォルトを使用しない場合には、ネットワーク名を指定します。

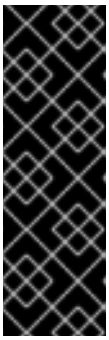
3 GCP ノードにタグを付けます。サフィックスに **ocp** を含める必要があります。たとえば、**node-instance-prefix** パラメーターの値を **mycluster** に設定する場合は、ノードは **myclusterocp** のタグを付ける必要があります。

4 一意の文字列を指定して、OpenShift Container Platform クラスターを特定します。

5 **True** の設定して、GCP でのマルチゾーンのデプロイメントをトリガーします。デフォルトでは **False** に設定されています。

クラスターインストールでは、シングルゾーンのサポートがデフォルトで設定されます。

異なるゾーンで GCP に OpenShift Container Platform をデプロイすると、単一障害点を回避しやすくなりますが、ストレージで問題が発生する可能性があります。これは、GCP ディスクがゾーン内に作成されるためです。OpenShift Container Platform ノードのホストがゾーン A でダウンし、Pod がゾーン B に移動した場合に、ディスクが異なるゾーンに配置されているので、永続ストレージはこれらの Pod にアタッチできません。詳細情報は、Kubernetes ドキュメントの マルチゾーンの制限 を参照してください。



重要

GCP を使用してロードバランサーサービスを実行 する場合は、Compute Engine VM ノードインスタンスには **ocp** のサフィックス: **<openshift_gcp_prefix>ocp** が必要です。たとえば、**openshift_gcp_prefix** パラメーターの値を **mycluster** に設定する場合は、このノードに **myclusterocp** のタグを付ける必要があります。Compute Engine VM インスタンスにネットワークタグを追加する方法に関する詳細は、「ネットワークタグの追加と削除」を参照してください。

3. OpenShift Container Platform サービスを再起動します。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

シングルゾーンサポートに戻すには、**multizone** の値を **false** に設定して、マスターとノードホストサービスを再起動します。

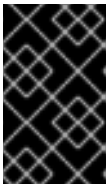
19.2.2.2. GCE 向けのノードホストの手動設定

全ノードホスト上で以下を実行します。

手順

1. 適切なノード設定マップを編集して、**kubeletArguments** セクションの内容を更新します。

```
kubeletArguments:
  *cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"*
```



重要

クラウドプロバイダーの統合を正常に機能させるため、**nodeName** は GCP のインスタンス名と一致していなければなりません。また、この名前は RFC1123 に準拠している必要もあります。

2. すべてのノードで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

19.2.3. GCP 向けの OpenShift Container Platform レジストリーの設定

Google Cloud Platform (GCP) は、OpenShift Container Platform が OpenShift Container Platform コンテナーレジストリーを使用してコンテナーイメージを保存するために、使用可能なオブジェクトクラウドストレージを提供します。

詳細情報は、GCP ドキュメントのクラウドストレージ を参照してください。

前提条件

インストールする前に、バケットを作成して、レジストリーイメージをホストする必要があります。以下のコマンドでは、設定したサービスアカウントを使用してリージョンバケットを作成します。

```
gsutil mb -c regional -l <region> gs://ocp-registry-bucket
cat <<EOF > labels.json
{
  "ocp-cluster": "mycluster"
}
EOF
gsutil label set labels.json gs://ocp-registry-bucket
rm -f labels.json
```

注記

デフォルトでは、バケットのデータは、Google が管理するキーを使用して自動的に暗号化されます。データの暗号化に別のキーを指定する場合には、GCP で利用可能な データ暗号化オプション を参照してください。

詳細情報は、ストレージバケットの作成ドキュメント を参照してください。

手順

レジストリーが Google Cloud Storage (GCS) バケットを使用できるように、Ansible インベントリーファイル を設定します。

```
[OSEv3:vars]
# GCP Provider Configuration
openshift_hosted_registry_storage_provider=gcs
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_replicas=1 (1)
openshift_hosted_registry_storage_gcs_bucket=<bucket_name> (2)
openshift_hosted_registry_storage_gcs_keyfile=<bucket_keyfile> (3)
openshift_hosted_registry_storage_gcs_rootdirectory=<registry_directory>
(4)
```

-

1. 設定するレプリカ数
2. レジストリーストレージのバケット名
3. データの暗号化にカスタムのキーファイルを使用する場合にバケットのキーファイルが配置されているインストーラーホストのパス
4. データの保存に使用するディレクトリー。デフォルトは **/registry**

詳細情報は、GCP ドキュメントのクラウドストレージ を参照してください。

19.2.3.1. GCP 向けの OpenShift Container Platform レジストリーの手動設定

GCP オブジェクトストレージを使用するには、レジストリーの設定ファイルを編集してレジストリー Pod にマウントします。

ストレージドライバーの設定ファイルに関する詳細情報は、「Google Cloud ストレージドライバーのドキュメント」を参照してください。

手順

1. 現在の/**etc/registry/config.yml** ファイルをエクスポートします。

```
$ oc get secret registry-config \
    -o jsonpath='{.data.config\.yml}' -n default | base64 -d \
  >> config.yml.old
```

2. 以前の /**etc/registry/config.yml** ファイルから新規の設定ファイルを作成します。

```
$ cp config.yml.old config.yml
```

3. このファイルを編集して GCP パラメーターを追加します。レジストリーの設定ファイルの **storage** セクションに、バケットと keyfile を指定します。

```
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  gcs:
    bucket: ocp-registry 1
    keyfile: mykeyfile 2
```

1. GCP バケット名に置き換えます。
2. JSON 形式のサービスアカウントの秘密鍵ファイル。Google アプリケーションのデフォルトの認証情報を使用する場合は、**keyfile** パラメーターは指定しないでください。

4. **registry-config** シークレットを削除します。

```
$ oc delete secret registry-config -n default
```

5. シークレットを再作成して、更新された構成ファイルを参照します。

```
$ oc create secret generic registry-config \
    --from-file=config.yml -n default
```

6. 更新された設定を読み取るためにレジストリーを再デプロイします。

```
$ oc rollout latest docker-registry -n default
```

19.2.3.1.1. レジストリーが GCP オブジェクトストレージを使用していることを確認します。

レジストリーが GCP バケットストレージを使用しているかどうかを確認します。

手順

1. GCP ストレージを使用して正常にレジストリーをデプロイした後に、GCP バケットストレージではなく **emptydir** が使用される場合には、**deploymentconfig** のレジストリーは、何も情報を表示しません。

```
$ oc describe dc docker-registry -n default
...
Mounts:
  ...
  /registry from registry-storage (rw)
Volumes:
registry-storage:
Type:       EmptyDir (1)
...
```

(1) Pod の寿命を共有する一時ディレクトリー

2. **/registry** のマウントポイントが空かどうかを確認します。これは、GCP ストレージが使用するボリュームです。

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- ls -l
/registry
total 0
```

3. 空の場合は、GCP バケット設定が **registry-config** シークレットで実行されているためです。

```
$ oc describe secret registry-config
Name:         registry-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque
```

```
Data
====
config.yml:  398 bytes
```

4. インストーラーは、「インストールドキュメントのストレージ」セクションで記載されているように、拡張されたレジストリー機能を使用して、希望の設定で **config.yml** ファイルを作成します。以下のコマンドで、ストレージバケット設定が保存されている **storage** セクションを含めて設定ファイルを表示します。

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
      -o=jsonpath='{.items[0].metadata.name}') \
  cat /etc/registry/config.yml

version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  gcs:
    bucket: ocp-registry
auth:
  openshift:
    realm: openshift
middleware:
  registry:
  - name: openshift
  repository:
  - name: openshift
    options:
      pullthrough: True
      acceptschema2: True
      enforcequota: False
  storage:
  - name: openshift
```

または、以下でシークレットを表示できます。

```
$ oc get secret registry-config -o jsonpath='{.data.config\.yml}' |
base64 -d
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
```

```
    gcs:
      bucket: ocp-registry
  auth:
    openshift:
      realm: openshift
  middleware:
    registry:
    - name: openshift
    repository:
    - name: openshift
      options:
        pullthrough: True
        acceptschema2: True
        enforcequota: False
    storage:
    - name: openshift
```

GCP コンソールで **Storage** を表示して **Browser** をクリックし、バケットを選択するか、**gsutil** コマンドを実行して、イメージのプッシュが正常に行われたことを確認します。

```
$ gsutil ls gs://ocp-registry/
gs://ocp-registry/docker/

$ gsutil du gs://ocp-registry/
7660385     gs://ocp-
registry/docker/registry/v2/blobs/sha256/03/033565e6892e5cc6dd03187d
00a4575720a928db111274e0fbf31b410a093c10/data
7660385     gs://ocp-
registry/docker/registry/v2/blobs/sha256/03/033565e6892e5cc6dd03187d
00a4575720a928db111274e0fbf31b410a093c10/
7660385     gs://ocp-registry/docker/registry/v2/blobs/sha256/03/
...
```

emptyDir ボリュームを使用する場合には、**/registry** マウントポイントは以下のようになります。

```
$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc         30G  226M   30G   1% /registry


$ oc exec \
    $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}')  -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 1000000000 1000000000 22 Jun 19 12:24 docker
```

19.2.4. OpenShift Container Platform が GCP ストレージを使用するための設定

OpenShift Container Platform は、永続ボリュームメカニズムを活用して GCP ストレージを使用できます。OpenShift Container Platform は、GCP にディスクを作成して、正しいインスタンスにこのディスクをアタッチします。

GCP ディスクは **ReadWriteOnce** アクセスモードで、1 つのノードで読み取り/書き込み可能な状態でボリュームをマウントできます。詳細情報は、『アーキテクチャーガイド』の「アクセスモード」のセクション を参照してください。

手順

1. OpenShift Container Platform は、**gce-pd** プロビジョナーを使用しており、Ansible インベントリーで **openshift_cloudprovider_kind=gce** および **openshift_gcp_*** 変数を使用する場合に、以下の **storageclass** を作成します。それ以外の場合、Ansible を使用せずに OpenShift Container Platform を設定しており、**storageclass** はインストール時に作成されていない場合には、以下のように、手動で作成できます。

```
$ oc get --export storageclass standard -o yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 annotations:
   storageclass.beta.kubernetes.io/is-default-class: "true"
 creationTimestamp: null
 name: standard
 selfLink: /apis/storage.k8s.io/v1/storageclasses/standard
parameters:
 type: pd-standard
provisioner: kubernetes.io/gce-pd
reclaimPolicy: Delete
```

PV を要求して、以前の手順で示した storageclass を使用すると、OpenShift Container Platform は GCP インフラストラクチャーにディスクを作成します。ディスクが作成されたことを確認するには以下を実行します。

```
$ gcloud compute disks list | grep kubernetes
kubernetes-dynamic-pvc-10ded514-7625-11e8-8c52-42010af00003  us-
west1-b  10       pd-standard  READY
```

19.3. サービスとしての GCP 外部のロードバランサー使用

LoadBalancer サービスを使用してサービスを外部に公開することで、OpenShift Container Platform が GCP ロードバランサーを使用するように設定できます。OpenShift Container Platform は GCP にロードバランサーを作成し、必要なファイアウォールルールを作成します。

手順

1. 新規アプリケーションを作成します。

```
$ oc new-app openshift/hello-openshift
```

2. ロードバランサーサービスを公開します。

```
$ oc expose dc hello-openshift --name='hello-openshift-external' --
type='LoadBalancer'
```

このコマンドは、以下の例とよく似た **LoadBalancer** サービスを作成します。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-openshift
  name: hello-openshift-external
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: port-1
    nodePort: 30714
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: port-2
    nodePort: 30122
    port: 8888
    protocol: TCP
    targetPort: 8888
  selector:
    app: hello-openshift
    deploymentconfig: hello-openshift
  sessionAffinity: None
  type: LoadBalancer
```

3. サービスが作成されたことを確認します。

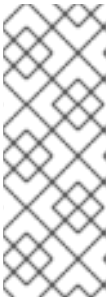
```
$ oc get svc
NAME                        TYPE           CLUSTER-IP
EXTERNAL-IP     PORT(S)                         AGE
hello-openshift             ClusterIP      172.30.62.10     <none>
8080/TCP,8888/TCP               20m
hello-openshift-external    LoadBalancer   172.30.147.214
35.230.97.224   8080:31521/TCP,8888:30843/TCP   19m
```

LoadBalancer タイプと外部 IP の値は、サービスが GCP ロードバランサーを使用してアプリケーションを公開することを示します。

OpenShift Container Platform は、GCP インフラストラクチャーに、必要なオブジェクトを作成します。以下に例を挙げます。

- ファイアウォール:

```
$ gcloud compute firewall-rules list | grep k8s
k8s-4612931a3a47c204-node-http-hc        my-net  INGRESS    1000
tcp:10256
k8s-fw-a1a8afaa7762811e88c5242010af0000  my-net  INGRESS    1000
tcp:8080,tcp:8888
```



注記

これらのファイアウォールは、**<openshift_gcp_prefix>ocp** のタグがついたインスタンスに適用されます。たとえば、**openshift_gcp_prefix** パラメーターが **mycluster** に設定されている場合には、ノードに **myclusterocp** のタグを付ける必要があります。Compute Engine VM インスタンスにネットワークタグを追加する方法については、「ネットワークタグの追加と削除」を参照してください。

- ヘルスチェック:

```
$ gcloud compute http-health-checks list | grep k8s
k8s-4612931a3a47c204-node        10256  /healthz
```

- ロードバランサー:

```
$ gcloud compute target-pools list | grep k8s
a1a8afaa7762811e88c5242010af0000  us-west1  NONE
k8s-4612931a3a47c204-node
$ gcloud compute forwarding-rules list | grep
a1a8afaa7762811e88c5242010af0000
a1a8afaa7762811e88c5242010af0000  us-west1  35.230.97.224  TCP
us-west1/targetPools/a1a8afaa7762811e88c5242010af0000
```

ロードバランサーが正しく設定されたことを確認するには、外部ホストから以下のコマンドを実行します。

```
$ curl 35.230.97.224:8080
Hello OpenShift!
```

第20章 AZURE の設定

20.1. 概要

OpenShift Container Platform は、Microsoft Azure ディスクをアプリケーションデータの永続ストレージとして使用するなど、Azure インフラストラクチャーにアクセスするように設定できます。これを実行するには、Microsoft Azure を適切に設定した後に OpenShift Container Platform ホストで追加の設定を行う必要があります。

20.2. パーミッション

OpenShift Container Platform 向けに Microsoft Azure を設定するには、以下のロールが必要です。

Contributor	すべての種類の Microsoft Azure リソースを作成し、管理します。

管理者ロールの追加に関する詳細情報は、「Azure サブスクリプション管理者の追加または変更」を参照してください。

20.3. 前提条件

- OpenShift Container Platform バージョン 3.5 以降で、Microsoft Azure Disk を永続ボリュームとして使用する場合には、Azure Cloud Provider を有効化する必要があります。
- Microsoft Azure で実行している OpenShift Container Platform ノードの仮想マシン (VM) はすべて、単一のリソースグループに所属する必要があります。
- Microsoft Azure 仮想マシンは、OpenShift Container Platform ノードと同じにする必要があり、これには大文字を含めることはできません。
- Azure Managed Disks (マネージドディスク) を使用する予定の場合:
  - OpenShift Container Platform バージョン 3.7 以降が必要です。
  - Azure マネージドディスクで、仮想マシンを作成する必要があります。
- アンマネージドディスクを使用する予定の場合:
  - アンマネージドディスクで、仮想マシンを作成する必要があります。
- OpenShift Container Platform クラスターに、カスタムの DNS 設定を使用する場合やクラスターノードが別の Microsoft Azure Virtual Networks (VNet) に含まれる場合には、クラスター内の各ノードが他のノードの IP アドレスを解決できるように、DNS を設定する必要があります。

20.4. AZURE 設定ファイル

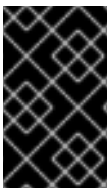
Azure ついて OpenShift Container Platform を設定するには、各ノードホストに **/etc/azure/azure.conf** ファイルが必要です。

ファイルが存在しない場合は、ファイルを作成し、以下を追加します。

```
tenantId: <> 1
```

```
subscriptionId: <> 2
aadClientId: <> 3
aadClientSecret: <> 4
aadTenantId: <> 5
resourceGroup: <> 6
cloud: <> 7
location: <> 8
vnetName: <> 9
securityGroupName: <> 10
primaryAvailabilitySetName: <> 11
```

1. クラスターがデプロイされているサブスクリプションの AAD テナント ID。
2. クラスターがデプロイされている Azure サブスクリプション ID。
3. Azure RM API と対話するための RBAC アクセス権を持つ AAD アプリケーションのクライアント ID。
4. Azure RM API と対話するための RBAC アクセス権を持つ AAD アプリケーションのクライアントシークレット。
5. これがテナント ID と同一であることを確認します (オプション)。
6. Azure VM が属する Azure のリソースグループ名。
7. 特定のクラウドリージョン。**AzurePublicCloud** など。
8. コンパクトな形式の Azure リージョン。**southeastasia** など (オプション)。
9. インスタンスを含む仮想ネットワーク。ロードバランサー作成時に使用します。
10. インスタンスとロードバランサーに関連付けられているセキュリティーグループ名。
11. ロードバランサーなどのリソースの作成時に使用するように設定されている可用性 (オプション)。



重要

インスタンスへのアクセスに使用される NIC には **internal-dns-name** が設定されている必要があります。これがないと、ノードはクラスターに再結合できず、コンソールにビルドログを表示できず、**oc rsh** が正常に機能しなくなる可能性があります。

20.4.1. クラスターのインストール中の Azure の設定

クラスターのインストール時に、Azure は以下のパラメーターを使用して設定できます。これらのパラメーターは、インベントリーファイルで設定可能です。

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
openshift_cloudprovider_kind=azure
openshift_cloudprovider_azure_client_id="{{
```

```
lookup('env','AZURE_CLIENT_ID') }}"
openshift_cloudprovider_azure_client_secret="{{
lookup('env','AZURE_CLIENT_SECRET') }}"
openshift_cloudprovider_azure_tenant_id="{{
lookup('env','AZURE_TENANT_ID') }}"
openshift_cloudprovider_azure_subscription_id="{{
lookup('env','AZURE_SUBSCRIPTION_ID') }}"
openshift_cloudprovider_azure_resource_group=openshift
openshift_cloudprovider_azure_location=australiasoutheast
```

注記

Ansible が Azure を設定する際に、以下のファイルがユーザー用に追加で作成されます。

- **/etc/origin/cloudprovider/azure.conf**

20.4.2. Azure 向けのマスターホストの手動設定

すべてのマスターで、マスター設定ファイル (デフォルトは **/etc/origin/master/master-config.yaml**) を編集するか、または作成し、**apiServerArguments** と **controllerArguments** の各セクションの内容を更新します。

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "azure"
    cloud-config:
      - "/etc/azure/azure.conf"
  controllerArguments:
    cloud-provider:
      - "azure"
    cloud-config:
      - "/etc/azure/azure.conf"
```

重要

コンテナー化インストールをトリガーすると、**/etc/origin** と **/var/lib/origin** のディレクトリーのみがマスターとノードのコンテナーにマウントされます。したがって、**master-config.yaml** は**/etc/** ではなく **/etc/origin/master** になければなりません。

20.4.3. Azure 向けのノードホストの手動設定

1. 適切なノード設定マップを編集して、**kubeletArguments** セクションの内容を更新します。

```
kubeletArguments:
  cloud-provider:
    - "azure"
  cloud-config:
    - "/etc/azure/azure.conf"
```

重要

コンテナー化インストールをトリガーすると、**/etc/origin** と **/var/lib/origin** のディレクトリーのみがマスターとノードのコンテナーにマウントされます。したがって、**node-config.yaml** は**/etc/** ではなく **/etc/origin/node** になければなりません。

20.4.4. 手動による設定変更の適用

マスターおよびノードのすべてのホストで OpenShift Container Platform サービスを起動または再起動し、設定の変更を適用します。「OpenShift Container Platform サービスの再起動」を参照してください。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

クラウドプロバイダーを使用しない状態から、使用するように切り替えると、エラーメッセージが表示されます。クラウドプロバイダーを追加すると、ノードが **hostname** を **`externalID`** として使用する (クラウドプロバイダーが使用されていなかった場合のケース) 代わりに、クラウドプロバイダーの **`instance-id`** (クラウドプロバイダーによって指定される) の使用に切り替えるため、ノードの削除が試みられます。この問題を解決するには、以下を実行します。

1. CLI にクラスター管理者としてログインします。
2. 既存のノードラベルをチェックし、これらをバックアップします。

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?
=Taints)'
```

3. ノードを削除します。

```
$ oc delete node <node_name>
```

4. 各ノードホストで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

5. 以前に使用していた各ノードのラベルを再度追加します。

第21章 VMWARE VSPHERE の設定

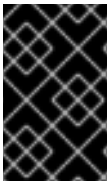
21.1. 概要

OpenShift Container Platform は、VMWare vSphere VMDK ボリュームをアプリケーションデータの永続ストレージとして使用するなど、VMWare vSphere VMDK ボリュームにアクセスするように設定できます。

vSphere クラウドプロバイダーは、OpenShift Container Platform 内での vSphere の管理対象ストレージの使用を許可します。vSphere クラウドプロバイダーは以下に対応しています。

- ボリューム
- 永続ボリューム
- ストレージクラスとボリュームのプロビジョニング

21.2. VMWARE VSPHERE クラウドプロバイダーの有効化

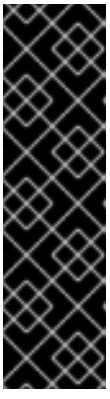


重要

VMware vSphere を有効化すると、各ノードの仮想マシンに VMware ツールをインストールする必要があります。詳しい情報は、「VMware ツールのインストール」を参照してください。

OpenShift Container Platform で VMWare vSphere クラウドプロバイダーを有効にするには、以下を実行します。

1. VM フォルダーを作成し、OpenShift Container Platform ノード VM をこのフォルダーに移動します。
2. ノードの VM 名が正規表現 **`[a-z](()?[0-9a-z])?(\.[a-z0-9](([-0-9a-z])?[0-9a-z])?)*`** に従っていることを確認します。



重要

VM 名では以下が禁止されています。

- 数字から始めることはできません。
- 大文字を含めることはできません。
- **-** 以外の特殊文字を含めることはできません。
- 文字数は 3 文字未満、64 文字以上に指定できません。

3. 各ノード VM で **`disk.EnableUUID`** パラメーターを **`TRUE`** に設定します。これにより VMDK は常に一貫した UUID を VM に提供し、ディスクが適切にマウントされるようになります。クラスターに参加するすべての仮想マシンノードについては、以下の GOVC ツールを使用する手順に従ってください。

   a. GOVC 環境をセットアップします。

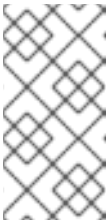
```
export GOVC_URL='vCenter IP OR FQDN'
export GOVC_USERNAME='vCenter User'
export GOVC_PASSWORD='vCenter Password'
export GOVC_INSECURE=1
```

b. ノード VM パスを見つけます。

```
govc ls /datacenter/vm/<vm-folder-name>
```

c. すべての VM で disk.EnableUUID を true に設定します。

```
govc vm.change -e="disk.enableUUID=1" -vm='VM Path'
```



注記

OpenShift Container Platform ノード VM がテンプレート VM で作成されている場合、**`disk.EnableUUID=1`** をテンプレート VM に設定することができます。このテンプレートからクローン作成される VM はこのプロパティーを継承します。

4. ロールを作成して、ロールを vSphere クラウドプロバイダーユーザーと vSphere エンティティーに割り当てます。vSphere クラウドプロバイダーでは、vCenter と対話するために以下の権限が必要になります。カスタムロール、ユーザーおよびロールの割り当てを作成する手順については、vSphere Documentation Center を参照してください。

ロール	権限	エンティティー	子への伝播
manage-k8s-node-vms	Resource.AssignVMToPool System.Anonymous System.Read System.View VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.AddRemoveDevice VirtualMachine.Config.RemoveDisk VirtualMachine.Inventory.Create VirtualMachine.Inventory.Delete	クラスター、ホスト、VM フォルダー	必要
manage-k8s-volumes	Datastore.AllocateSpace Datastore.FileManagement System.Anonymous System.Read System.View	データストア	No

ロール	権限	エンティティー	子への伝播
k8s-system-read-and-spbm-profile-view	StorageProfile.View System.Anonymous System.Read System.View	vCenter	No
ReadOnly	System.Anonymous System.Read System.View	データセンター、データストアクラスター、データストアストレージフォルダー	No

注記

vSphere クラウドプロバイダーを有効にすると、ノード名が vCenter インベントリーの VM 名に設定されます。

21.3. VMWARE VSPHERE 設定ファイル

VMWare vSphere について OpenShift Container Platform を設定するには、各ノードホストに **/etc/origin/cloudprovider/vsphere.conf** ファイルが必要です。

ファイルが存在しない場合は、ファイルを作成し、以下を追加します。

```
[Global]
        user = "username" 1
        password = "password" 2
        server = "10.10.0.2" 3
        port = "443" 4
        insecure-flag = "1" 5
        datacenter = "datacenter-name" 6
        datastore = "datastore-name" 7
        working-dir = "vm-folder-path" 8
        vm-uuid = "vm-uuid" 9
[Disk]
    scsicontrollertype = pvscsi
```

1. vSphere クラウドプロバイダーの vCenter ユーザー名。
2. 指定されたユーザーの vCenter パスワード。
3. vCenter サーバーの IP アドレスまたは FQDN。
4. (オプション) vCenter サーバーのポート番号。デフォルトはポート **443** になります。
5. vCenter が自己署名証明書を使用している場合は **1** に設定します。
6. ノード VM がデプロイされているデータセンターの名前。
7.

ストレージクラスまたは動的プロビジョニングを使ったボリュームのプロビジョニングに使用されるデータストアの名前。データストアがストレージフォルダー内にある場合やデータストアがデー

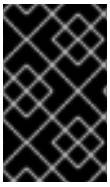
8 (オプション) ノード VM が置かれている vCenter VM フォルダーパス。ノード VM がルートの VM フォルダーにある場合、空のパス (**`working-dir = ""`**) に設定することができます。

9 (オプション) ノード VM の VM インスタンス UUID。空 (**`vm-uuid = ""`**) に設定することができます。空に設定する場合、これは仮想マシン上の **/sys/class/dmi/id/product_serial** ファイルから取得されます (root アクセス権が必要)。

21.4. マスターの設定

すべてのマスターでマスター設定ファイル (デフォルトは **/etc/origin/master/master-config.yaml**) を編集するか、または 作成し、**`apiServerArguments`** と **`controllerArguments`** の各セクションの内容を以下で更新します。

```
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      {}
  apiServerArguments:
    cloud-provider:
    - "vsphere"
    cloud-config:
    - "/etc/origin/cloudprovider/vsphere.conf"
  controllerArguments:
    cloud-provider:
    - "vsphere"
    cloud-config:
    - "/etc/origin/cloudprovider/vsphere.conf"
```



重要

コンテナー化インストールをトリガーすると、**/etc/origin** と **/var/lib/origin** のディレクトリーのみがマスターとノードのコンテナーにマウントされます。したがって、**master-config.yaml** は**/etc/** ではなく **/etc/origin/master** になければなりません。

21.5. ノードの設定

1. 適切な「ノード設定マップ」を編集して、**`kubeletArguments`** セクションの内容を更新します。

```
kubeletArguments:
  cloud-provider:
    - "vsphere"
  cloud-config:
    - "/etc/origin/cloudprovider/vsphere.conf"
```

重要

コンテナー化インストールをトリガーする際は、**/etc/origin** と **/var/lib/origin** のディレクトリーのみがマスターとノードのコンテナーにマウントされます。したがって、**node-config.yaml** は**/etc/** ではなく **/etc/origin/node** になければなりません。

21.6. 設定変更の適用

マスターおよびノードのすべてのホストで OpenShift Container Platform サービスを起動または再起動し、設定の変更を適用します。「OpenShift Container Platform サービスの再起動」を参照してください。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

クラウドプロバイダーを使用しない状態から、使用するように切り替えると、エラーメッセージが表示されます。クラウドプロバイダーを追加すると、ノードが **hostname** を **`externalID`** として使用する (クラウドプロバイダーが使用されていなかった場合のケース) 代わりに、クラウドプロバイダーの **`instance-id`** (クラウドプロバイダーによって指定される) の使用に切り替えるため、ノードの削除が試みられます。この問題を解決するには、以下を実行します。

1. CLI にクラスター管理者としてログインします。
2. 既存のノードラベルをチェックし、これらをバックアップします。

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?
=Taints)'
```

3. ノードを削除します。

```
$ oc delete node <node_name>
```

4. 各ノードホストで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

5. 以前に使用していた各ノードのラベルを再度追加します。

21.7. 永続ボリュームのバックアップ

OpenShift Container Platform は、自由にクラスター内のノードにあるボリュームをアタッチしたり、アタッチ解除できるように、**個別の永続ディスク** として新規ボリュームをプロビジョニングします。そのため、スナップショットを使用するボリュームはバックアップできません。

以下の方法で、PV のバックアップを作成します。

1. PV を使用してアプリケーションを停止します。
2. 永続ディスクのクローンを作成します。
3. アプリケーションを再起動します。

4. クローンしたディスクのバックアップを作成します。

5. クローンしたディスクを削除します。

第22章 ローカルボリュームの設定

22.1. 概要

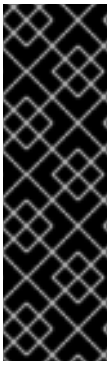
OpenShift Container Platform は、アプリケーションデータのローカルボリュームにアクセスするように設定できます。

ローカルのボリュームとは、ローカルにマウントされたファイルシステムを表す永続ボリューム (PV) です。OpenShift Container Platform 3.10 の時点では、これには raw ブロックデバイスも含まれます。raw デバイスは、物理デバイスに、さらにダイレクトなルートを提供し、アプリケーションがその物理デバイスに対する I/O 操作のタイミングをより詳細に制御できるようにします。このように、raw デバイスは、通常独自のキャッシングを行うデータベース管理システムなどの複雑なアプリケーションに適しています。ローカルボリュームにはユニークな機能がいくつか含まれています。ローカルボリューム PV を使用する Pod は、ローカルボリュームがマウントされるノードでスケジューリングされます。

また、ローカルボリュームには、ローカルにマウントされたデバイスに PV を自動作成するプロビジョナーが含まれます。現時点で、このプロビジョナーは事前設定されたディレクトリーのみをスキャンします。このプロビジョナーは、ボリュームを動的にプロビジョニングする機能はありませんが、今後のリリースで実装される可能性はあります。

ローカルボリュームのプロビジョナーを使用すると、ローカルストレージを OpenShift Container Platform 内で使用することができます。ローカルボリュームのプロビジョナーは以下に対応しています。

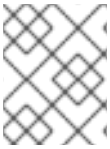
- ボリューム
- PV



重要

ローカルボリュームは、テクノロジープレビュー機能のみとなっています。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) ではサポートされていません。これらは、機能的に完全でない可能性があり、Red Hat では実稼働環境での使用を推奨しません。テクノロジープレビュー機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様に機能性をテストしていただき、開発プロセス中にフィードバックをお寄せいただくことを目的としています。Red Hat テクノロジープレビュー機能のサポート対象範囲に関する詳しい情報は、「テクノロジプレビュー機能のサポート範囲」を参照してください。

22.2. ローカルボリュームのマウント



注記

すべてのローカルボリュームは、OpenShift Container Platform によって PV として使用される前に手動でマウントする必要があります。

ローカルボリュームをマウントします。

1. すべてのボリュームは、**/mnt/local-storage/<storage-class-name>/<volume>** パスにマウントしてください。管理者は、必要に応じてディスクパーティションまたは LVM などの方法でローカルデバイスを作成し、これらのデバイスに適切なファイルシステムを作成して、スクリプトまたは **/etc/fstab** エントリーを使用してそれらをマウントします。

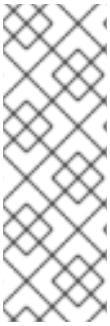
```
# device name   # mount point                  # FS    # options #
extra
/dev/sdb1       /mnt/local-storage/ssd/disk1 ext4     defaults 1 2
/dev/sdb2       /mnt/local-storage/ssd/disk2 ext4     defaults 1 2
/dev/sdb3       /mnt/local-storage/ssd/disk3 ext4     defaults 1 2
/dev/sdc1       /mnt/local-storage/hdd/disk1 ext4     defaults 1 2
/dev/sdc2       /mnt/local-storage/hdd/disk2 ext4     defaults 1 2
```

2. Docker コンテナー内で実行中のプロセスに全ボリュームがアクセスできるようにします。これを可能にするには、以下のように、マウントされたファイルシステムのラベルを変更してください。

```
---
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-
storage/
---
```

22.3. ローカルプロビジョナーの設定

OpenShift Container Platform は、ローカルデバイス用に PV を作成する場合や、再利用の有効化に使用されなくなり PV を消去する場合に、外部のプロビジョナーを使用します。



注記

- ローカルボリュームのプロビジョナーは大半のプロビジョナーとは異なり、動的なプロビジョニングに対応していません。
- ローカルボリュームのプロビジョナーを使用する場合には、管理者は、各ノードでローカルボリュームを事前設定して、discovery ディレクトリーの配下にマウントする必要があります。その後にプロビジョナーは各ボリュームについて PV の作成とクリーンアップを実行してボリュームを管理します。

ローカルプロビジョナーを設定します。

1. ConfigMap を使用して外部プロビジョナーが、ストレージクラスとディレクトリーを関連付けられるように設定します。この設定は、プロビジョナーがデプロイされる前に作成する必要があります。以下に例を示します。

```
kind: ConfigMap
metadata:
  name: local-volume-config
data:
    storageClassMap: |
        local-ssd: (1)
            hostDir:  /mnt/local-storage/ssd (2)
            mountDir: /mnt/local-storage/ssd (3)
        local-hdd:
            hostDir: /mnt/local-storage/hdd
            mountDir: /mnt/local-storage/hdd
```

(1) ストレージクラス名

(2)

ホスト上のディレクトリーへのパス。**/mnt/local-storage** のサブディレクトリーでなければなりません。

3 プロビジョナー Pod のディレクトリーへのパス。ホストで使用されているディレクトリー構造と同じ構造を使用することを推奨します。今回の例では、**`mountDir`** は省略可能です。

2. **(オプション)** ローカルボリュームのプロビジョナーおよびその設定用にスタンドアロンの namespace を作成します。例: **`oc new-project local-storage`**

上記の設定により、プロビジョナーは以下を作成します。

- **/mnt/local-storage/ssd** ディレクトリーにマウントされる全サブディレクトリーに対して、**`local-ssd`** のストレージクラスを指定した PV 1 つ
- **/mnt/local-storage/hdd** ディレクトリーにマウントされる全サブディレクトリーに対して、**`local-hdd`** のストレージクラスを指定した PV 1 つ

警告

ConfigMap の構文は、OpenShift Container Platform 3.9 と 3.10 では変更されています。この機能はテクノロジープレビューであるため、ConfigMap は更新時に自動的に変換されません。

22.4. ローカルプロビジョナーのデプロイ

注記

プロビジョナーを起動する前に、すべてのローカルデバイスをマウントし、ストレージクラスとそれらのディレクトリーと共に ConfigMap を作成します。

ローカルプロビジョナーをデプロイします。

1. **local-storage-provisioner-template.yaml** ファイルからローカルプロビジョナーをインストールします。
2. サービスアカウントを作成して、root ユーザーでの Pod 実行、hostPath ボリュームの使用をはじめ、SELinux コンテキストでローカルボリュームの監視、管理、消去ができるようにします。

```
$ oc create serviceaccount local-storage-admin
$ oc adm policy add-scc-to-user privileged -z local-storage-admin
```

プロビジョナー Pod で任意の Pod が作成したローカルボリュームのコンテンツを削除できるようにするには、root 権限と任意の SELinux コンテキストが必要です。ホスト上の **/mnt/local-storage** パスにアクセスするには hostPath が必要です。

3. テンプレートをインストールします。

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/release-
3.10/examples/storage-examples/local-examples/local-storage-
provisioner-template.yaml
```

4. **CONFIGMAP**、**SERVICE_ACCOUNT**, **NAMESPACE** および **PROVISIONER_IMAGE** パラメーターに値を指定して、テンプレートをインスタンス化します。

```
$ oc new-app -p CONFIGMAP=local-volume-config \
  -p SERVICE_ACCOUNT=local-storage-admin \
  -p NAMESPACE=local-storage \
  -p PROVISIONER_IMAGE=registry.access.redhat.com/openshift3/local-
storage-provisioner:v3.9 \ 1
  local-storage-provisioner
```

1 **v3.10** など、お使いの OpenShift Container Platform バージョン番号を指定します。

5. 必要なストレージクラスを追加します。

```
$ oc create -f ./storage-class-ssd.yaml
$ oc create -f ./storage-class-hdd.yaml
```

以下は例を示しています。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: local-ssd
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: local-hdd
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

その他の設定オプションについては、テンプレート を参照してください。このテンプレートは、全ノードで Pod を実行する DeamonSet を作成します。Pod は ConfigMap に指定されるディレクトリーを監視し、それらの PV を自動的に作成します。

このプロビジョナーは、PV が開放されると、変更されたディレクトリーからすべてのデータが削除されるので、Root パーミッションを使用して実行します。

22.5. 新規デバイスの追加

新規デバイスの追加は半自動で実行されます。プロビジョナーは設定されたディレクトリーで新規マウントについて定期的にチェックします。管理者は、以下のように、SELinux ラベルを適用して、新しいサブディレクトリーの作成、デバイスのマウント、Pod によるデバイスの使用許可を行う必要があります。

```
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-
storage/
```

重要

上記のいずれかの操作を省くと、適切な PV が作成されなくなることがあります。

== Raw ブロックデバイスの設定: ローカルのボリュームプロビジョナーを使用すると、Raw ブロックデバイスを静的にプロビジョニングできます。この機能は、デフォルトでは無効になっており、追加の設定が必要です。

Raw ブロックデバイスを設定するには以下を行います。

1. 全マスターで、**BlockVolume** 機能ゲートを有効化します。全マスターでマスター設定ファイルを編集または作成して (デフォルトは **/etc/origin/master/master-config.yaml**)、**apiServerArguments** および **controllerArguments** セクションに、**BlockVolume=true** を追加します。

```
apiServerArguments:
   feature-gates:
   - BlockVolume=true
...

 controllerArguments:
   feature-gates:
   - BlockVolume=true
...
```

2. ノード設定 ConfigMap を編集して、全ノードで機能ゲートを有効化します。

```
$ oc edit configmap node-config-compute --namespace
openshift-node
$ oc edit configmap node-config-master --namespace
openshift-node
$ oc edit configmap node-config-infra --namespace
openshift-node
```

3. 以下のように、すべての ConfigMaps の、**kubeletArguments** の機能ゲートアレイに **BlockVolume=true** が含まれていることを確認します。

ノードの configmap feature-gates 設定

```
kubeletArguments:
   feature-gates:
   -
RotateKubeletClientCertificate=true,RotateKubeletServerCe
rtificate=true,BlockVolume=true
```

4. マスターを再起動します。ノードは、設定の変更後に自動的に再起動されます。これは数分かかる可能性があります。

=== 新規 Raw ブロックデバイスの準備: プロビジョナーを起動する前に、Pod が使用できる全 Raw ブロックデバイスを **/mnt/local-storage/<storage class>** ディレクトリー構造にリンクします。たとえば、**/dev/dm-36** のディレクトリーを利用できるようにしま

す。

1. **/mnt/local-storage** に、デバイスのストレージクラスのディレクトリーを作成します。

```
$ mkdir -p /mnt/local-storage/block-devices
```

2. このデバイスを参照するシンボリックリンクを作成します。

```
$ ln -s /dev/dm-36 dm-uuid-LVM-1234
```

名前の競合が発生するのを回避するには、**/dev/disk/by-uuid** または **/dev/disk/by-id** ディレクトリーからのリンクと、シンボリックリンクに同じ名前を使用します。

1. プロビジョナー設定用の ConfigMap を作成するか、更新します。

```
kind: ConfigMap
metadata:
  name: local-volume-config
data:
    storageClassMap: |
        block-devices: (1)
            hostDir:  /mnt/local-storage/block-devices (2)
            mountDir: /mnt/local-storage/block-devices (3)
```

(1) ストレージクラス名

(2) ホスト上のディレクトリーへのパス。**/mnt/local-storage** のサブディレクトリーでなければなりません。

(3) プロビジョナー Pod のディレクトリーへのパス。ホストが使用するディレクトリー構造を使用する場合 (推奨) には、**`mountDir`** パラメーターを省略します。

2. デバイスと、**/mnt/local-storage/** の **`SELinux`** ラベルを変更します。

```
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-
storage/
$ chcon unconfined_u:object_r:svirt_sandbox_file_t:s0 /dev/dm-36
```

3. Raw ブロックデバイスのストレージクラスを作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: block-devices
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

プロビジョナーは、このブロックデバイス **/dev/dm-36** を使用する準備ができ、PV としてプロビジョニングします。

=== Raw ブロックデバイスプロビジョナーのデプロイ

Raw ブロックデバイスへのプロビジョナーのデプロイは、ローカルボリュームへのプロビジョナーのデプロイに似ていますが、2 点相違点があります。

1. プロビジョナーは特権付きのコンテナーで実行する必要があります。
2. プロビジョナーは、ホストから **/dev** のファイルシステムにアクセスできる必要があります。

Raw ブロックデバイスにプロビジョナーをデプロイします。

1. **local-storage-provisioner-template.yaml** ファイルからテンプレートをダウンロードします。
2. テンプレートを編集します。
   a. コンテナー仕様の **securityContext** の **privileged** 属性を **true** に設定します。

```
...
  containers:
...
    name: provisioner
...
      securityContext:
        privileged: true
...
```

   b. **hostPath** を使用して、コンテナーにホストの **/dev/** ファイルシステムをマウントします。

```
...
  containers:
...
    name: provisioner
...
    volumeMounts:
    - mountPath: /dev
      name: dev
...
  volumes:
    - hostPath:
        path: /dev
        name: dev
...
```

3. 変更済みの YAML ファイルからテンプレートを作成します。

```
$ oc create -f local-storage-provisioner-template.yaml
```

4. プロビジョナーを起動します。

```
$ oc new-app -p CONFIGMAP=local-volume-config \
  -p SERVICE_ACCOUNT=local-storage-admin \
  -p NAMESPACE=local-storage \
  -p
  PROVISIONER_IMAGE=registry.access.redhat.com/openshift3/local-
storage-provisioner:v3.10 \
  local-storage-provisioner
```

=== Raw ブロックデバイスの永続ボリューム

以下のように、Pod で raw ブロックデバイスを使用するには、**volumeMode:** を **Block** に、**storageClassName** を **block-devices** に設定して、永続ボリューム要求 (PVC) を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  storageClassName: block-devices
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 1Gi
```

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-test
  labels:
    name: busybox-test
spec:
  restartPolicy: Never
  containers:
    - resources:
        limits :
          cpu: 0.5
      image: gcr.io/google_containers/busybox
      command:
        - "/bin/sh"
        - "-c"
        - "while true; do date; sleep 1; done"
      name: busybox
      volumeDevices:
        - name: vol
          devicePath: /dev/xvda
  volumes:
      - name: vol
        persistentVolumeClaim:
          claimName: block-pvc
```

ボリュームは、Pod にマウントされていませんが、**/dev/xvda** Raw ブロックデバイスとして公開されます。

第23章 永続ストレージの設定

23.1. 概要

Kubernetes の永続ボリュームフレームワークにより、お使いの環境で利用可能なネットワークストレージを使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これは、アプリケーションのニーズに応じて初回 OpenShift Container Platform インストールの完了後に行うことができ、ユーザーは基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようになります。

このトピックでは、以下のサポートされるボリュームプラグインを使って永続ボリュームを OpenShift Container Platform で設定する方法を説明します。

- NFS
- GlusterFS
- OpenStack Cinder
- Ceph RBD
- AWS Elastic Block Store (EBS)
- GCE Persistent Disk
- iSCSI
- ファイバーチャネル
- Azure ディスク
- Azure File
- FlexVolume
- VMWare vSphere
- Container Storage Interface (CSI)
- 動的プロビジョニングとストレージクラスの作成
- ボリュームのセキュリティー
- セレクターとラベルによるボリュームのバインディング

23.2. NFS を使用した永続ストレージ

23.2.1. 概要

OpenShift Container Platform クラスターは、NFS を使用している永続ストレージを使ってプロビジョニングすることが可能です。永続ボリューム (PV) および Persistent Volume Claim (永続ボリューム要求、PVC) は、プロジェクト全体でボリュームを共有するための便利な方法を提供します。PV 定義に含まれる NFS に固有の情報は、Pod 定義で直接定義することも可能ですが、この方法の場合にはボリュームが一意のクラスターリソースとして作成されされないため、ボリュームが競合の影響を受けやすくなります。

このトピックでは、NFS 永続ストレージタイプの具体的な使用方法について説明します。OpenShift Container Platform と NFS についてある程度理解していることを前提とします。OpenShift Container Platform 永続ボリューム (PV) の一般的なフレームワークの詳細は永続ストレージの概念に関するトピックを参照してください。

23.2.2. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントするには、ストレージがあらかじめ基礎となるインフラストラクチャーに存在している必要があります。NFS ボリュームをプロビジョニングするには、NFS サーバーの一覧とエクスポートパスのみが必要です。

最初に、PV のオブジェクト定義を作成します。

例23.1 NFS を使用した PV オブジェクト定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 5Gi 2
  accessModes:
  - ReadWriteOnce 3
  nfs: 4
    path: /tmp 5
    server: 172.17.0.2 6
  persistentVolumeReclaimPolicy: Recycle 7
```

1. ボリュームの名前。これは、各種の **oc <command> pod** コマンドの PV アイデンティティーです。
2. このボリュームに割り当てられるストレージの量。
3. これはボリュームへのアクセスの制御に関連するように見えますが、実際はラベルの場合と同様に、PVC を PV に一致させるために使用されます。現時点では、**accessModes** に基づくアクセスルールは適用されていません。
4. 使用されているボリュームタイプ。この場合は **nfs** プラグインです。
5. NFS サーバーがエクスポートしているパス。
6. NFS サーバーのホスト名または IP アドレス
7. PV の回収ポリシー。ボリュームが要求から解放されるタイミングでボリュームで実行されることを定義します。有効な選択肢は **Retain** (デフォルト) と **Recycle** です。「リソースの回収」を参照してください。

注記

各 NFS ボリュームは、クラスター内のスケジュール可能なすべてのノードによってマウント可能でなければなりません。

定義をファイル (**nfs-pv.yaml** など) に保存し、PV を作成します。

```
$ oc create -f nfs-pv.yaml
persistentvolume "pv0001" created
```

PV が作成されたことを確認します。

```
# oc get pv
NAME                      LABELS    CAPACITY     ACCESSMODES   STATUS
CLAIM     REASON    AGE
pv0001                    <none>    5368709120   RWO           Available
31s
```

以下の手順で PVC が作成されます。これは新規 PV にバインドされます。

例23.2 PVC オブジェクト定義

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce 1
  resources:
    requests:
      storage: 1Gi 2
```

1 PV について前述されているように、**accessModes** はセキュリティーを実施するのではなく、PV を PVC と一致させるラベルとして機能します。

2 この要求は **1Gi** 以上の容量を提供する PV を検索します。

定義をファイル (**nfs-claim.yaml** など) に保存し、PVC を作成します。

```
# oc create -f nfs-claim.yaml
```

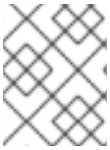
23.2.3. ディスククォータの実施

ディスクパーティションを使用して、ディスククォータとサイズ制限を実施することができます。それぞれのパーティションを独自のエクスポートとすることができ、それぞれのエクスポートは 1 つの PV になります。OpenShift Container Platform は PV に固有の名前を適用しますが、NFS ボリュームのサーバーとパスの一意性については管理者に委ねられています。

この方法でクォータを有効にすると、開発者は永続ストレージを具体的な量 (10Gi など) を指定して要求し、この容量以上が設定されている対応のボリュームに適合できます。

23.2.4. NFS ボリュームのセキュリティー

このセクションでは、一致するパーミッションや SELinux の考慮点を含む、NFS ボリュームのセキュリティーについて説明します。ユーザーは、POSIX パーミッションやプロセス UID、補助グループおよび SELinux の基礎的な点を理解している必要があります。



注記

NFS ボリュームを実装する前に 「ボリュームのセキュリティー」のトピックをすべてお読みください。

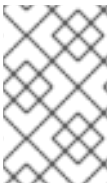
開発者は、Pod 定義の **volumes** セクションで、PVC を名前で参照するか、または NFS ボリュームのプラグインを直接参照して NFS ストレージを要求します。

NFS サーバーの **/etc/exports** ファイルにはアクセス可能な NFS ディレクトリーが含まれています。ターゲットの NFS ディレクトリーには、POSIX の所有者とグループ ID があります。OpenShift Container Platform NFS プラグインは、同じ POSIX の所有者とエクスポートされる NFS ディレクトリーにあるパーミッションを使って、コンテナーの NFS ディレクトリーをマウントします。ただし、コンテナーは NFS マウントの所有者と同等の実効 UID では実行されません。これは期待される動作です。

ターゲットの NFS ディレクトリーが NFS サーバーに表示される場合を例に取って見てみましょう。

```
# ls -lZ /opt/nfs -d
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs

# id nfsnobody
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

コンテナーは SELinux ラベルと一致している必要があり、ディレクトリーにアクセスするために UID **65534** (**nfsnobody** 所有者) か、または補助グループの **5555** のいずれかを使って実行する必要があります。



注記

ここで、所有者 ID 65534 は一例として使用されています。NFS の **root_squash** が **root** (0) を **nfsnobody** (65534) にマップしても、NFS エクスポートは任意の所有者 ID を持つことができます。所有者 65534 は NFS エクスポートには必要ありません。

23.2.4.1. グループ ID

NFS アクセスに対応する際の推奨される方法として、補助グループを使用することができます (NFS エクスポートのパーミッションを変更するオプションがないことを前提としています)。OpenShift Container Platform の補助グループは共有ストレージに使用されます (例: NFS)。これとは対照的に、Ceph RBD や iSCSI などのブロックストレージは、Pod の **securityContext** で **fsGroup** SCC ストラテジーと **fsGroup** の値を使用します。

注記

一般的に、永続ストレージへのアクセスを取得する場合、ユーザー ID ではなく補助グループ ID を使用することが推奨されます。補助グループについては「ボリュームのセキュリティー」トピックで詳しく説明されています。

上記のターゲット NFS ディレクトリーの例で使用したグループ ID は 5555 なので、Pod は、**supplementalGroups** を使用してグループ ID を Pod レベルの **securityContext** 定義の下で定義することができます。以下は例になります。

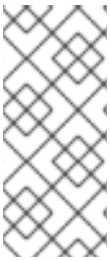
```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    supplementalGroups: [5555] 2
```

1 **securityContext** は特定のコンテナーの下位ではなく、この Pod レベルで定義します。

2 Pod 向けに定義される GID の配列。ここでは、配列に 1 つの要素があり、追加の GID はコンマで区切られます。

Pod の要件を満たすカスタム SCC が存在しない場合、Pod は **制限付きの** SCC に一致する可能性があります。この SCC では、**supplementalGroups** ストラテジーが **RunAsAny** に設定されています。これは、指定されるグループ ID は範囲のチェックなしに受け入れられることを意味します。

その結果、上記の Pod は受付をパスして起動します。しかし、グループ ID の範囲をチェックすることが望ましい場合は、「Pod のセキュリティーとカスタム SCC」で説明されているようにカスタム SCC の使用が推奨されます。カスタム SCC は、最小および最大のグループ ID が定義され、グループ ID の範囲チェックが実施され、グループ ID 5555 が許可されるように作成できます。



注記

カスタム SCC を使用するには、最初にそこに適切なサービスアカウントを追加しておく必要があります。たとえば、所定のプロジェクトでは、Pod 仕様において別のサービスアカウントが指定されていない限り、**default** のサービスアカウントを使用してください。詳細は、「SCC のユーザー、グループまたはプロジェクトへの追加」を参照してください。

23.2.4.2. ユーザー ID

ユーザー ID は、コンテナーイメージまたは Pod 定義で定義できます。ユーザー ID に基づいてストレージアクセスを制御する方法については、「ボリュームのセキュリティー」のトピックで説明されています。NFS 永続ストレージをセットアップする前に、必ず読んでおいてください。

注記

永続ストレージへのアクセスを取得する場合、通常はユーザー ID ではなく 補助グループ ID を使用することが推奨されます。

上記の ターゲット NFS ディレクトリーの例 では、コンテナーは UID を 65534 (ここではグループ ID を省略します) に設定する必要があります。したがって以下を Pod 定義に追加することができます。

```
spec:
  containers: 1
  - name:
  ...
    securityContext:
      runAsUser: 65534 2
```

■

1. Pod には、各コンテナーに固有の **securityContext** (ここに表示されている) と、その Pod で定義されたすべてのコンテナーに適用される Pod レベルの **securityContext** が含まれます。

2. 65534 は **nfsnobody** ユーザーです。

デフォルトのプロジェクトと**制限付き** SCC を前提とする場合は、Pod が要求するユーザー ID 65534 は許可されず、Pod は失敗します。Pod が失敗する理由は以下の通りです。

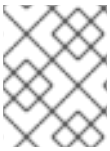
- 65534 をユーザー ID として要求している。
- ユーザー ID 65534 を許可する SCC を確認するために Pod で利用できるすべての SCC が検査される (実際は SCC のすべてのポリシーがチェックされますが、ここでのフォーカスはユーザー ID になります)。
- 利用可能なすべての SCC は **runAsUser** ストラテジーに **MustRunAsRange** を使用するため、UID の範囲チェックが必要である。
- 65534 は SCC またはプロジェクトのユーザー ID 範囲に含まれていない。

一般に、事前定義された SCC は変更しないことが勧められています。ただし、この状況を改善するには、「ボリュームのセキュリティー」のトピックで説明されているようにカスタム SCC を作成することが推奨されます。カスタム SCC は、最小および最大のユーザー ID が定義され、UID 範囲のチェックの実施が設定されており、UID 65534 が許可されるように作成できます。

注記

カスタム SCC を使用するには、最初にそこに適切なサービスアカウントを追加しておく必要があります。たとえば、所定のプロジェクトでは、Pod 仕様において別のサービスアカウントが指定されていない限り、**default** のサービスアカウントを使用してください。詳細は、「SCC のユーザー、グループまたはプロジェクトへの追加」を参照してください。

23.2.4.3. SELinux



注記

SELinux を使用してストレージアクセスを制御する方法についての詳細は、「ボリュームのセキュリティー」を参照してください。

デフォルトでは、SELinux は Pod からリモートの NFS サーバーへの書き込みを許可していません。NFS ボリュームは正常にマウントされますが、読み取り専用です。

SELinux を各ノードで有効にした状態で NFS ボリュームへの書き込みを有効にするには、以下を実行します。

```
# setsebool -P virt_use_nfs 1
```

上記の **-P** オプションは、ブール値をリブート間で継続させます。

virt_use_nfs ブール値は **docker-selinux** パッケージで定義されます。このブール値が定義されていないことを示すエラーが表示された場合は、このパッケージがインストールされていることを確認してください。

23.2.4.4. エクスポート設定

任意のコンテナーユーザーにボリュームの読み取りと書き出しを許可するには、NFS サーバーにエクスポートされる各ボリュームは以下の条件を満たしている必要があります。

- 各エクスポートを以下のように指定します。

```
/<example_fs> *(rw,root_squash)
```

- ファイアウォールは、マウントポイントへのトラフィックを許可するように設定する必要があります。
  - NFSv4 の場合、デフォルトのポート **2049** (**nfs**) とポート **111** (**portmapper**) を設定します。

    NFSv4

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

  - NFSv3 の場合、以下の 3 つのポートを設定します。**2049** (**nfs**)、**20048** (**mountd**)、**111** (**portmapper**)。

    NFSv3

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

- NFS エクスポートとディレクトリーは、ターゲット Pod からアクセスできるようにセットアップされる必要があります。この場合、エクスポートをコンテナーのプライマリー UID で所有されるように設定するか、または上記の「グループ ID」で説明したように、**supplementalGroups** を使用して Pod にグループアクセスを付与します。Pod のセキュリティーに関する追加情報は、「ボリュームのセキュリティー」のトピックを参照してください。

23.2.5. リソースの回収

NFS は OpenShift Container Platform の **再利用可能な** プラグインインターフェースを実装します。回収タスクは、それぞれの永続ボリュームに設定されるポリシーに基づいて自動プロセスによって処理されます。

デフォルトでは、PV は **Retain** に設定されます。**Recycle** に設定される NFS ボリュームは、要求からの解放後 (このボリュームにバインドされるユーザーの **PersistentVolumeClaim** の削除後) に除去されます (**rm -rf** がこのボリュームで実行されます)。リサイクルされた NFS ボリュームは新規の要求にバインドさせることができます。

PV への要求が解除される (PVC が削除される) と、PV オブジェクトは再利用できません。代わりに、新規の PV が元のボリュームと同じ基本ボリューム情報を使って作成されます。

たとえば、管理者は **nfs1** という名前の PV を作成するとします。

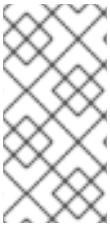
```
apiVersion: v1
kind: PersistentVolume
```

```
metadata:
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

ユーザーは、**nfs1** にバインドされる **PVC1** を作成します。次にユーザーは **PVC1** を削除し、**nfs1** への要求を解除します。これにより、**nfs1** は **Released** になります。管理者が同じ NFS 共有を利用可能にする必要がある場合には、同じ NFS サーバー情報を使って新規 PV を作成する必要があります。この場合、PV の名前は元の名前とは異なる名前にします。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

元の PV を削除して、PV を同じ名前で再作成することは推奨されません。PV のステータスを **Released** から **Available** に手動で変更しようとすると、エラーが発生してデータが失われる可能性があります。



注記

Recycle の保持ポリシーを持つ PV はデータを除去 (**rm -rf** を実行) し、要求について **Available** というマークを付けます。**Recycle** の保持ポリシーは、OpenShift Container Platform 3.6 以降で非推奨となっています。Recycler を利用しているユーザーは、代わりに動的プロビジョニングとボリュームの削除機能を使用してください。

23.2.6. 自動化

クラスターは、NFS を使用している永続ストレージを使って以下の方法でプロビジョニングすることができます。

- ディスクパーティションを使ってストレージクォータを実施する。
- 要求を持つプロジェクトにボリュームを制限してセキュリティーを実施する。
- 破棄されたリソースの回収を各 PV に設定する。

スクリプトを使って上記タスクを自動化する方法は多数あります。まずは Ansible Playbook のサンプルを使用して開始することができます。

23.2.7. その他の設定とトラブルシューティング

適切なエクスポートとセキュリティーマッピングを行うため、使用している NFS のバージョンおよびその設定方法に応じて追加の設定が必要になることがあります。以下は例になります。

NFSv4 のマウントにすべてのファイルの所有者が **nobody:nobody** と誤って表示される。	• NFS の ID マッピング設定 (/etc/idmapd.conf) に原因がある可能性が高い。 • Red Hat ソリューションを参照してください。
NFSv4 の ID マッピングが無効になっている	• NFS クライアントとサーバーの両方で以下を実行してください。 `# echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable_idmapping`

23.3. RED HAT GLUSTER STORAGE を使用する永続ストレージ

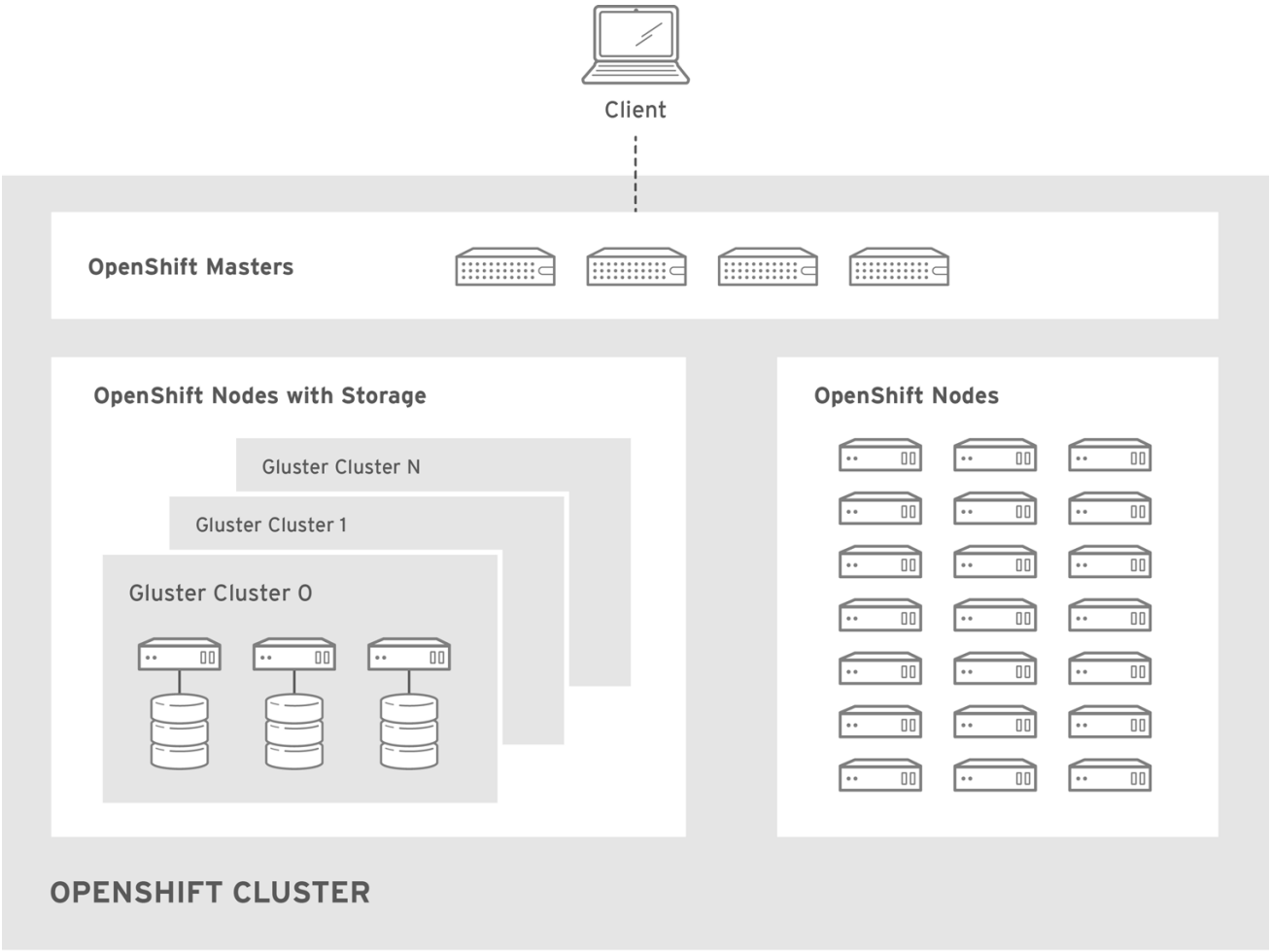
23.3.1. 概要

Red Hat Gluster Storage は、OpenShift Container Platform の永続ストレージと動的プロビジョニングを提供するように設定でき、OpenShift Container Platform 内のコンテナー化ストレージ (**コンバージドモード**) としても、独自のノード上の非コンテナー化ストレージ (**インデペンデントモード**) としても使用できます。

23.3.1.1. コンバージドモード

コンバージドモードの場合には、Red Hat Gluster Storage は、コンテナー化されたディレクトリーを OpenShift Container Platform ノードで直接実行します。それにより、コンピュートおよびストレージインスタンスをスケジュールでき、同じハードウェアのセットから実行することができます。

図23.1 アーキテクチャー - コンバージドモード



コンバージドモードは Red Hat Gluster Storage 3.1 update 3 より利用できます。詳細については、「converged mode for OpenShift Container Platform」を参照してください。

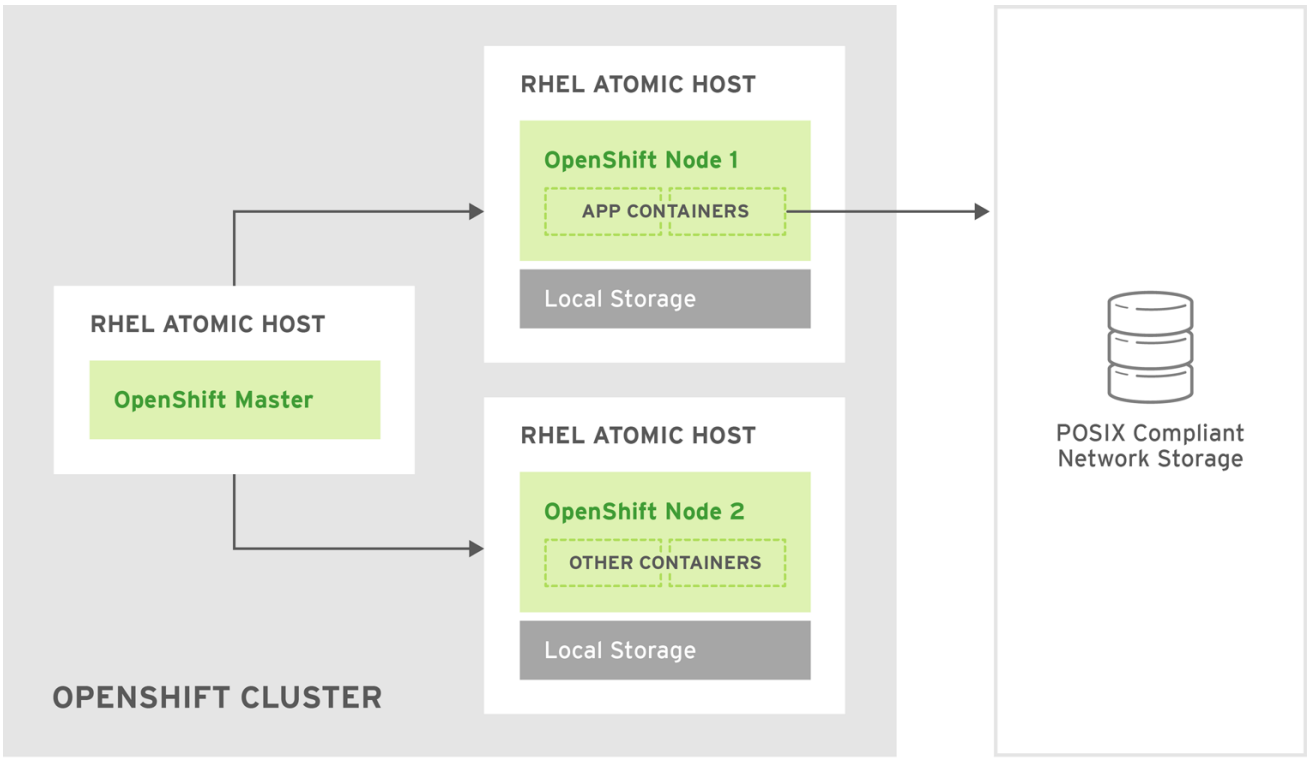
23.3.1.2. インデペンデントモード

インデペンデントモードの場合、Container-Ready Storage を使用することで、Red Hat Gluster Storage は独自の専用ノードで実行され、GlusterFS のボリューム管理 REST サービスの heketi のインスタンスによって管理されます。heketi サービスは、スタンドアロンまたはコンテナー化のいずれかで実行できます。コンテナー化の場合、簡単なメカニズムで高可用性をサービスに提供できます。ここでは、heketi をコンテナー化した場合の設定に焦点を当てます。

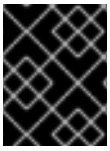
23.3.1.3. スタンドアロンの Red Hat Gluster Storage

スタンドアロンの Red Hat Gluster Storage クラスターが環境で使用できる場合、OpenShift Container Platform の GlusterFS ボリュームプラグインを使用してそのクラスター上でボリュームを使用することができます。この方法は、アプリケーションが専用のコンピュートノード、OpenShift Container Platform クラスターで実行され、ストレージはその専用ノードから提供される従来のデプロイメントです。

図23.2 アーキテクチャー - OpenShift Container Platform の GlusterFS ボリュームプラグインを使用したスタンドアロンの Red Hat Gluster Storage クラスター



Red Hat Gluster Storage の詳細については、『Red Hat Gluster Storage Installation Guide』および『Red Hat Gluster Storage Administration Guide』を参照してください。



重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.3.1.4. GlusterFS ボリューム

GlusterFS ボリュームは、POSIX に準拠したファイルシステムを提供し、クラスター上の 1 つ以上のノードにまたがる 1 つ以上の「ブリック」から構成されます。このブリックは所定のストレージノード上のディレクトリーであり、一般的にブロックストレージデバイスのマウントポイントになります。GlusterFS はボリュームの設定に応じて、所定のボリュームのブリック間でファイルの分散および複製を処理します。

heketi は、ボリューム管理において、作成、削除、サイズ変更といった一般的な操作に使用することが推奨されます。OpenShift Container Platform は、GlusterFS プロビジョナーを使用する際に heketi が存在していることを前提としています。heketi はデフォルトで、レプリカ数が 3 のボリュームを作成します。このボリュームの各ファイルには 3 つの異なるノードをまたがる 3 つのコピーがあります。したがって、heketi が使用する Red Hat Gluster Storage クラスターでは 3 つ以上のノードを利用可能にすることが推奨されます。

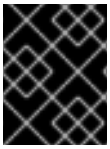
GlusterFS ボリュームに使用可能な機能は多数ありますが、これらについては本書では扱いません。

23.3.1.5. gluster-block ボリューム

gluster-block ボリュームは、iSCSI 上にマウントすることが可能なボリュームです。既存の GlusterFS ボリュームにファイルを作成し、そのファイルをブロックデバイスとして iSCSI ターゲットを介して提

供することでマウントできます。このような GlusterFS ボリュームは、ブロックホスティングボリュームと呼ばれます。

gluster-block ボリュームにはトレードオフもあります。iSCSI ターゲットとして使用される場合、複数のノード/クライアントでマウントできる GlusterFS ボリュームとは対照的に、gluster-block ボリュームをマウントできるのは 1 回に 1 つのノード/クライアントのみです。ただし、バックエンドのファイルであるため、GlusterFS ボリュームでは一般にコストのかかる操作 (メタデータの参照など) を、GlusterFS ボリュームでの一般的により高速な操作 (読み取り、書き込みなど) に変換することが可能です。それにより、特定の負荷に対するパフォーマンスを大幅に改善できる可能性があります。



重要

現時点では、gluster-block ボリュームは OpenShift ロギングと OpenShift メトリクスストレージにのみを使用することが推奨されます。

23.3.1.6. Gluster S3 Storage

Gluster S3 サービスは、ユーザーアプリケーションが S3 インターフェースを介して GlusterFS ストレージへアクセスすることを可能にします。このサービスは GlusterFS の、オブジェクトデータ用とオブジェクトメタデータ用の 2 つのボリュームにバインドされ、受信する S3 REST 要求をボリューム上のファイルシステム操作に変換します。このサービスは OpenShift Container Platform 内の Pod として実行することが推奨されます。

重要

現時点では、Gluster S3 サービスの使用およびインストールはテクノロジープレビューの段階にあります。

23.3.2. 考慮事項

このセクションでは、Red Hat Gluster Storage を OpenShift Container Platform で使用する際に考慮すべきトピックについて取り上げます。

23.3.2.1. ソフトウェア要件

GlusterFS ボリュームにアクセスするには、すべてのスケジュール可能なノードで **`mount.glusterfs`** コマンドを利用できる必要があります。RPM ベースのシステムの場合は、**glusterfs-fuse** パッケージがインストールされている必要があります。

```
# yum install glusterfs-fuse
```

このパッケージはすべての RHEL システムにインストールされています。ただし、Red Hat Gluster Storage の最新バージョンにアップデートすることを推奨します。そのためには、以下の RPM リポジトリーを有効にする必要があります。

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-
server-rpms
```

glusterfs-fuse がノードにすでにインストールされている場合、最新バージョンがインストールされていることを確認します。

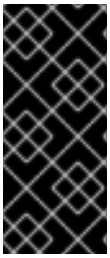
```
# yum update glusterfs-fuse
```

23.3.2.2. ハードウェア要件

コンバージドモード (converged mode) またはインデペンデントモード (independent mode) のクラスターで使用されるノードはストレージノードとみなされます。単一ノードは複数のグループに分割できませんが、ストレージノードはそれぞれ別個のクラスターグループに分類できます。ストレージノードの各グループについては、以下が当てはまります。

- 1 グループあたり 3 つ以上のストレージノードが必要です。
- 各ストレージノードには 8 GB 以上の RAM が必要です。これにより、Red Hat Gluster Storage Pod、その他のアプリケーションおよび基礎となる OS を実行できます。
  - 各 GlusterFS ボリュームはストレージクラスターにあるすべてのストレージノードのメモリー (約 30 MB) も消費します。RAM の合計量は、コンカレントボリュームがいくつ求められているか、またはいくつ予想されるかによって決める必要があります。
- 各ストレージノードには、現在のデータまたはメタデータを含まない 1 つ以上の raw ブロックデバイスが必要です。それらのブロックデバイス全体は GlusterFS ストレージで使用されます。以下が存在しないことを確認してください。
  - パーティションテーブル (GPT または MSDOS)
  - ファイルシステムまたは未処理のファイルシステムの署名
  - 以前のボリュームグループの LVM2 署名および論理ボリューム
  - LVM2 物理ボリュームの LVM2 メタデータ

  不確かな場合には、**`wipefs -a <device>`** で上記のすべてを消去する必要があります。



重要

2 つのクラスター、つまりインフラストラクチャーアプリケーション (OpenShift Container レジストリーなど) のストレージ専用のクラスターと一般的なアプリケーションのストレージ専用のクラスターについて計画することをお勧めします。これには、合計で 6 つのストレージノードが必要となりますが、この設定は I/O およびボリューム作成のパフォーマンスへの潜在的な影響を回避するために推奨されます。

23.3.2.3. ストレージのサイジング

GlusterFS クラスターはすべて、そのストレージを利用するアプリケーションで想定される要件に基づいてサイズを決定する必要があります。たとえば、「OpenShift ロギング」と「OpenShift メトリクス」の両方に対して、サイジングガイドが提供されています。

その他考慮すべき事項:

- コンバージドモードまたはインデペンデントモードのクラスターでは、デフォルトの動作により 3 通りのレプリケーションを持つ GlusterFS ボリュームが作成されます。そのため、合計のストレージの計画を立てる際には、必要な容量の 3 倍にする必要があります。
  - 例として、各 heketi インスタンスは 2 GB の **`heketidbstorage`** ボリュームを作成する場合、ストレージクラスター内の 3 つのノードに合計で 6 GB の raw ストレージが必要になります。この容量は常に必要となり、これをサイジングの際に考慮する必要があります。
  - 統合 OpenShift Container レジストリーなどのアプリケーションでは、GlusterFS の単一ボリュームがアプリケーションの複数インスタンスで共有されます。

- gluster-block ボリュームを使用する場合は、所定のブロックのボリューム容量をフルサイズで保持するための十分な容量を備えた GlusterFS ブロックホスティングボリュームがなければなりません。
  - デフォルトでは、このようなブロックホスティングボリュームが存在しない場合、これが設定されたサイズで自動的に 1 つ作成されます。デフォルトのサイズは 100 GB です。クラスター内に新しいブロックホスティングボリュームを作成するための十分なスペースがない場合、ブロックボリュームの作成は失敗します。自動作成の動作、および自動作成されるボリュームのサイズはどちらも設定することが可能です。
  - OpenShift ロギングや OpenShift メトリクスなどの gluster-block ボリュームを使用する複数のインスタンスを持つアプリケーションは、各インスタンスにつき 1 つのボリュームを使用します。
- Gluster S3 サービスは、2 つの GlusterFS ボリュームにバインドされます。デフォルトのクラスターインストールでは、ボリュームはそれぞれ 1 GB となり、合計 6 GB の raw ストレージを使用します。

23.3.2.4. ボリューム操作の動作

作成や削除などのボリューム操作は、さまざまな環境条件の影響を受けることもあれば、アプリケーションに影響を与えることもあります。

- アプリケーション Pod が動的にプロビジョニングされた GlusterFS の Persistent Volume Claim (永続ボリューム要求、PVC) を要求する場合は、ボリュームの作成と対応する PVC へのバインドにかかる追加の時間を考慮する必要があります。これはアプリケーション Pod の起動時間に影響します。

注記

GlusterFS ボリュームの作成時間は、ボリュームの数に応じて直線的に増加します。たとえば、クラスター内に推奨されるハードウェア仕様を使用する 100 のボリュームがある場合、各ボリュームの作成、割り当て、Pod へのバインドに約 6 秒の時間がかかりました。

- PVC が削除されると、基礎となる GlusterFS ボリュームの削除がトリガーされます。PVC は **`oc get pvc`** の出力からすぐに消えますが、ボリュームが完全に削除される訳ではありません。GlusterFS ボリュームは、**`heketi-cli volume list`** および **`gluster volume list`** のコマンドライン出力に表示されなくなったときにのみ削除されていると見なされます。

注記

GlusterFS ボリュームの削除とそのストレージのリサイクルの時間は、アクティブな GlusterFS ボリュームの数に応じて直線的に増加します。保留中のボリューム削除は実行中のアプリケーションに影響しませんが、ストレージ管理者は、特にリソース消費を大きな規模で調整している場合には、ボリュームの削除にかかる時間を認識し、それを見積もることができるようにしておく必要があります。

23.3.2.5. ボリュームのセキュリティー

このセクションでは、Portable Operating System Interface [Unix 向け] (POSIX) パーミッションや SELinux に関する考慮事項を含む、Red Hat Gluster Storage ボリュームのセキュリティーについて説明します。ボリュームのセキュリティー、POSIX パーミッションおよび SELinux に関する基本を理解していることを前提とします。

23.3.2.5.1. POSIX パーミッション

Red Hat Gluster Storage ボリュームは POSIX 準拠のファイルシステムを表します。そのため、**chmod** や **chown** などの標準コマンドラインツールを使用してアクセスパーミッションを管理できます。

コンバージドモードとインデペンデントモードでは、ボリュームの作成時に、ボリュームの Root を所有するグループ ID を指定することも可能です。静的なプロビジョニングでは、これは、**heketi-cli** ボリュームの作成コマンドの一部として指定します。

```
$ heketi-cli volume create --size=100 --gid=10001000
```

警告

このボリュームに関連付けられる PersistentVolume には、PersistentVolume を使用する Pod がファイルシステムにアクセスできるように、グループ ID をアノテーションとして付加する必要があります。このアノテーションは以下の形式で指定します。

pv.beta.kubernetes.io/gid: "<GID>" ---

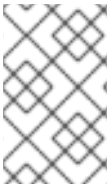
動的プロビジョニングの場合は、プロビジョナーがグループ ID を自動的に生成し、これを適用します。**gidMin** および **gidMax** StorageClass パラメーターを使用してこのグループ ID の選択範囲を制御できます (「動的プロビジョニング」を参照してください)。プロビジョナーは、生成される PersistentVolume にグループ ID をアノテーションとして付ける処理も行います。

23.3.2.5.2. SELinux

デフォルトでは、SELinux は Pod からリモート Red Hat Gluster Storage サーバーへの書き込みを許可しません。SELinux が有効な状態で Red Hat Gluster Storage ボリュームへの書き込みを有効にするには、GlusterFS を実行する各ノードで以下のコマンドを実行します。

```
$ sudo setsebool -P virt_sandbox_use_fusefs on 1
$ sudo setsebool -P virt_use_fusefs on
```

1 **-P** オプションを使用すると、再起動した後もブール値が永続化されます。



注記

virt_sandbox_use_fusefs ブール値は、**docker-selinux** パッケージによって定義されます。このブール値が定義されていないというエラーが表示される場合は、このパッケージがインストールされていることを確認してください。

注記

Atomic Host を使用している場合に、Atomic Host をアップグレードすると、SELinux のブール値が消去されるので、Atomic Host をアップグレードする場合には、これらのブール値を設定し直す必要があります。

23.3.3. サポート要件

Red Hat Gluster Storage と OpenShift Container Platform のサポートされる統合を作成するには、以下の要件が満たされている必要があります。

インデペンデントモードまたは スタンドアロンの Red Hat Gluster Storage の場合:

- 最小バージョン: Red Hat Gluster Storage 3.1.3
- すべての Red Hat Gluster Storage ノードに Red Hat Network チャンネルとサブスクリプションマネージャーリポジトリーへの有効なサブスクリプションが必要です。
- Red Hat Gluster Storage ノードは、「Planning Red Hat Gluster Storage Installation」に記載されている要件に準拠している必要があります。
- Red Hat Gluster Storage ノードは、最新のパッチとアップグレードが適用された完全に最新の状態でなければなりません。最新バージョンへのアップグレードについては、『Red Hat Gluster Storage Installation Guide』を参照してください。
- 各 Red Hat Gluster Storage ノードには、完全修飾ドメイン名 (FQDN) が設定されている必要があります。適切な DNS レコードが存在すること、および FQDN が正引きと逆引きの両方の DNS ルックアップで解決できることを確認してください。

23.3.4. インストール

スタンドアロンの Red Hat Gluster Storage の場合、OpenShift Container Platform で使用するためにインストールする必要があるコンポーネントはありません。OpenShift Container Platform には組み込み GlusterFS ボリュームドライバーが付属しており、これを使用して既存のボリュームを既存のクラスターで活用できます。既存のボリュームの使用方法については、「プロビジョニング」を参照してください。

コンバージドモードおよびインデペンデントモードでは、「クラスターインストールプロセスを使用して、必要なコンポーネントをインストールすることを推奨します。

23.3.4.1. インデペンデントモード: Red Hat Gluster Storage ノードのインストール

インデペンデントモードの場合は、Red Hat Gluster Storage ノードに適切なシステム設定 (ファイアウォールポートやカーネルモジュールなど) が設定されており、Red Hat Gluster Storage サービスが実行されている必要があります。このサービスは追加で設定できず、Trusted Storage Pool を作成することはできません。

Red Hat Gluster Storage ノードのインストールは本書の対象外です。詳細については、「インデペンデントモードの設定」を参照してください。

23.3.4.2. インストーラーの使用

クラスターインストールプロセスを使用すると、2 つの GlusterFS ノードグループの 1 つまたは両方をインストールできます。

- **`glusterfs`**: ユーザーアプリケーション用の汎用ストレージクラスター。
- **`glusterfs-registry`**: 統合 OpenShift Container レジストリーなどのインフラストラクチャーアプリケーション用の専用ストレージクラスター。

I/O およびボリューム作成のパフォーマンスに対する潜在的影響を回避するため、両方のグループをデプロイすることを推奨します。これらはどちらもインベントリーホストファイルで定義されます。

クラスターを定義するには、**[OSEv3:children]** グループに該当する名前を追加し、類似した名前付きグループを作成して、グループにノード情報を設定します。その後、**[OSEv3:vars]** グループのさまざまな変数を使用してクラスターを定義できます。**glusterfs** 変数は **openshift_storage_glusterfs_** から、**glusterfs-registry** 変数は **openshift_storage_glusterfs_registry_** からそれぞれ始まります。**openshift_hosted_registry_storage_kind** などのその他のいくつかの変数は GlusterFS クラスターと対話します。

Red Hat Gluster Storage Pod が、サービス停止後に、異なるバージョンの Red Hat Gluster Storage が含まれるクラスターにアップグレードされないようにするには、すべてのコンテナー化コンポーネントに、イメージ名とバージョンタグを指定することを推奨します。関連のある変数は以下のとおりです。

- **openshift_storage_glusterfs_image**
- **openshift_storage_glusterfs_block_image**
- **openshift_storage_glusterfs_s3_image**
- **openshift_storage_glusterfs_heketi_image**
- **openshift_storage_glusterfs_registry_image**
- **openshift_storage_glusterfs_registry_block_image**
- **openshift_storage_glusterfs_registry_s3_image**
- **openshift_storage_glusterfs_registry_heketi_image**

注記

gluster-block および **gluster-s3** のイメージ変数は、適切なデプロイメント変数 (末尾が **_block_deploy** および **_s3_deploy** の変数) が True の場合のみ必要です。

以下は、このバージョンの OpenShift Container Platform で推奨される値です。

- **openshift_storage_glusterfs_image=registry.access.redhat.com/rhgs3/rhgs-server-rhel7:v3.10**
- **openshift_storage_glusterfs_block_image=registry.access.redhat.com/rhgs3/rhgs-gluster-block-prov-rhel7:v3.10**
- **openshift_storage_glusterfs_s3_image=registry.access.redhat.com/rhgs3/rhgs-s3-server-rhel7:v3.10**
- **openshift_storage_glusterfs_heketi_image=registry.access.redhat.com/rhgs3/rhgs-volmanager-rhel7:v3.10**

注記

これらの値は、対応する **registry** 変数と同じです。

変数の詳細な一覧については、GitHub の GlusterFS ロールに関する README を参照してください。

変数を設定したら、インストールの環境に応じていくつかの Playbook が利用可能になります。

- クラスターインストールのメイン Playbook を使用すると、OpenShift Container Platform の初期インストールと並行して GlusterFS クラスターをデプロイできます。
  - これには、GlusterFS ストレージを使用する統合 OpenShift Container レジストリーのデプロイが含まれます。
  - OpenShift ロギングや OpenShift メトリクスは含まれません。現時点では、これは別の手順で扱われるためです。詳細については、「OpenShift ロギングおよびメトリクス用のコンバージドモード」を参照してください。
- **playbooks/openshift-glusterfs/config.yml** を使用して、クラスターを既存の OpenShift Container Platform インストールにデプロイできます。
- **playbooks/openshift-glusterfs/registry.yml** を使用して、クラスターを既存の OpenShift Container Platform インストールにデプロイできます。さらに、GlusterFS ストレージを使用する統合 OpenShift Container レジストリーもデプロイされます。

重要

OpenShift Container Platform クラスターに既存のレジストリーがない状態でなければなりません。

- **playbooks/openshift-glusterfs/uninstall.yml** を使用して、インベントリーホストファイルの設定に一致する既存のクラスターを削除できます。これは、設定エラーによってデプロイメントが失敗した場合に OpenShift Container Platform 環境をクリーンアップするのに便利です。

注記

GlusterFS Playbook では、必ず冪等性を確保できるわけではありません。

注記

GlusterFS インストール全体 (ディスクデータを含む) を削除してインストールし直すことなく、特定のインストールに対して Playbook を複数回実行することは、現在はサポートされていません。

23.3.4.2.1. 例: 基本的なコンバージドモードのインストール

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
```

2. **[OSEv3:children]** セクションに **glusterfs** を追加して、 **[glusterfs]** グループを有効にします。

```
[OSEv3:children]
masters
nodes
glusterfs
```

3. GlusterFS ストレージをホストする各ストレージノードのエントリーを含む **[glusterfs]** セクションを追加します。ノードごとに、 **glusterfs_devices** を GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも 1 つのデバイスを一覧に含める必要があります。各デバイスは、パーティションや LVM PV がないベアでなければなりません。変数は以下の形式で指定します。

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2>", ... ]'
```

以下は例を示しています。

```
[glusterfs]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. **[glusterfs]** の下に一覧表示されているホストを **[nodes]** グループに追加します。

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-compute"
node12.example.com openshift_node_group_name="node-config-compute"
node13.example.com openshift_node_group_name="node-config-compute"
```

5. インストーラーを実行します。

- OpenShift Container Platform の初回インストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

23.3.4.2.2. 例: 基本的なインデペンデントモードのインストール

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_storage_glusterfs_namespace=app-storage
```

```
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_is_native=false
openshift_storage_glusterfs_heketi_is_native=true
openshift_storage_glusterfs_heketi_executor=ssh
openshift_storage_glusterfs_heketi_ssh_port=22
openshift_storage_glusterfs_heketi_ssh_user=root
openshift_storage_glusterfs_heketi_ssh_sudo=false
openshift_storage_glusterfs_heketi_ssh_keyfile="/root/.ssh/id_rsa"
```

2. **[OSEv3:children]** セクションに **glusterfs** を追加して、 **[glusterfs]** グループを有効にします。

```
[OSEv3:children]
masters
nodes
glusterfs
```

3. GlusterFS ストレージをホストする各ストレージノードのエントリーを含む **[glusterfs]** セクションを追加します。ノードごとに、 **glusterfs_devices** を GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくと も 1 つのデバイスを一覧に含める必要があります。各デバイスはパーティションや LVM PV がないベアでなければなりません。また、**glusterfs_ip** をノードの IP アドレスに設定します。変数は以下の形式で指定します。

```
<hostname_or_ip> glusterfs_ip=<ip_address> glusterfs_devices='[ "
</path/to/device1/>", "</path/to/device2>", ... ]'
```

以下は例を示しています。

```
[glusterfs]
gluster1.example.com glusterfs_ip=192.168.10.11 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster2.example.com glusterfs_ip=192.168.10.12 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster3.example.com glusterfs_ip=192.168.10.13 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
```

4. インストーラーを実行します。

- OpenShift Container Platform の初回インストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

23.3.4.2.3. 例: 統合 OpenShift Container レジストリーを使用するコンバージドモード

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs (1)
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-
role.kubernetes.io/infra=true'
```

(1) 統合 OpenShift Container レジストリーをインフラストラクチャーノードで実行することが推奨されます。インフラストラクチャーノードは、OpenShift Container Platform クラスターのサービスを提供するために管理者がデプロイするアプリケーションを実行する専用ノードです。

2. **[OSEv3:children]** セクションに **glusterfs_registry** を追加して、**[glusterfs_registry]** グループを有効にします。

```
[OSEv3:children]
masters
nodes
glusterfs_registry
```

3. GlusterFS ストレージをホストする各ストレージノードのエントリーを含む **[glusterfs_registry]** セクションを追加します。ノードごとに、 **glusterfs_devices** を GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも 1 つのデバイスを一覧に含める必要があります。各デバイスはパーティションや LVM PV がないベアでなければなりません。変数は次の形式で指定します。

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2>", ... ]'
```

以下は例を示しています。

```
[glusterfs_registry]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. **[glusterfs_registry]** の下に一覧表示されているホストを **[nodes]** グループに追加します。

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-infra"
```

```
node12.example.com openshift_node_group_name="node-config-infra"
node13.example.com openshift_node_group_name="node-config-infra"
```

5. インストーラーを実行します。

- OpenShift Container Platform の初回インストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

23.3.4.2.4. 例: OpenShift ロギングおよびメトリクス用のコンバージドモード

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-
role.kubernetes.io/infra": "true"} 1
openshift_metrics_cassandra_nodeselector={"node-
role.kubernetes.io/infra": "true"} 2
openshift_metrics_heapster_nodeselector={"node-
role.kubernetes.io/infra": "true"} 3
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-
registry-block" 4

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-
role.kubernetes.io/infra": "true"} 5
openshift_logging_curator_nodeselector={"node-
role.kubernetes.io/infra": "true"} 6
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":
"true"} 7
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi 8
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-
block" 9

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
```

```
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=fals
e
```

1 2 3 5 6 7 統合 OpenShift Container レジストリー、ロギングおよびメトリクスは、管理者が OpenShift Container Platform クラスターにサービスを提供するためにデプロイしたアプリケーションである「インフラストラクチャー」アプリケーション専用のノードで実行することを推奨します。

4 9 ロギングとメトリクスに使用する StorageClass を指定します。この名前は、ターゲットの GlusterFS クラスター (例: **glusterfs-<name>-block**) 名から生成されます。この例では **registry** にデフォルト設定されています。

8 OpenShift ロギングでは、PVC のサイズを指定する必要があります。ここで指定される値は単なる例であり、推奨される値ではありません。

注記

これらの変数とその他の変数の詳細については、GlusterFS ロールに関する README を参照してください。

2. **[OSEv3:children]** セクションに **glusterfs_registry** を追加して、**[glusterfs_registry]** グループを有効にします。

```
[OSEv3:children]
masters
nodes
glusterfs_registry
```

3. GlusterFS ストレージをホストする各ストレージノードのエントリーを含む **[glusterfs_registry]** セクションを追加します。ノードごとに、**glusterfs_devices** を GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも 1 つのデバイスを一覧に含める必要があります。各デバイスはパーティションや LVM PV がないベアでなければなりません。変数は次の形式で指定します。

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2>", ... ]'
```

以下は例を示しています。

```
[glusterfs_registry]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. **[glusterfs_registry]** の下に一覧表示されているホストを **[nodes]** グループに追加します。

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-infra"
node12.example.com openshift_node_group_name="node-config-infra"
node13.example.com openshift_node_group_name="node-config-infra"
```

5. インストーラーを実行します。

- OpenShift Container Platform の初回インストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

23.3.4.2.5. 例: アプリケーション、レジストリー、ロギングおよびメトリクス用のコンバージドモード

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs 1
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-
role.kubernetes.io/infra=true'

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-
role.kubernetes.io/infra": "true"} 2
openshift_metrics_cassandra_nodeselector={"node-
role.kubernetes.io/infra": "true"} 3
openshift_metrics_heapster_nodeselector={"node-
role.kubernetes.io/infra": "true"} 4
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-
registry-block" 5

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-
role.kubernetes.io/infra": "true"} 6
openshift_logging_curator_nodeselector={"node-
role.kubernetes.io/infra": "true"} 7
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":
"true"} 8
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi 9
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-
block" 10

openshift_storage_glusterfs_namespace=app-storage
```

```
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=fals
e
```

1 2 3 4 6 7 8 統合 OpenShift Container レジストリー、ロギングおよびメトリクスをインフラストラクチャーノードで実行することが推奨されます。インフラストラクチャーノードは、OpenShift Container Platform クラスターのサービスを提供するために管理者がデプロイするアプリケーションを実行する専用ノードです。

5 10 ロギングとメトリクスに使用する StorageClass を指定します。この名前は、ターゲットの GlusterFS クラスター (例: **glusterfs-<name>-block**) 名から生成されます。この例では **<name>** は **registry** にデフォルト設定されています。

9 OpenShift ロギングでは、PVC サイズを指定する必要があります。ここで指定される値は単なる例であり、推奨される値ではありません。

2. **[OSEv3:children]** セクションに **glusterfs** と **glusterfs_registry** を追加し、**[glusterfs]** グループと **[glusterfs_registry]** グループを有効にします。

```
[OSEv3:children]
...
glusterfs
glusterfs_registry
```

3. **[glusterfs]** セクションと **[glusterfs_registry]** セクションを追加し、両セクションに GlusterFS ストレージをホストするストレージノードを入力します。ノードごとに **glusterfs_devices** を、GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも 1 つのデバイスを一覧に含める必要があります。各デバイスは、パーティションや LVM PV がないベアでなければなりません。変数は以下の形式で指定します。

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "
</path/to/device2>", ... ]'
```

以下は例を示しています。

```
[glusterfs]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'

[glusterfs_registry]
```

```
node14.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node15.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node16.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. **[glusterfs]** と **[glusterfs_registry]** に一覧表示されているホストを **[nodes]** グループに追加します。

```
[nodes]
...
node11.example.com openshift_node_group_name='node-config-compute'
1
node12.example.com openshift_node_group_name='node-config-compute'
2
node13.example.com openshift_node_group_name='node-config-compute'
3
node14.example.com openshift_node_group_name='node-config-infra'" 4
node15.example.com openshift_node_group_name='node-config-infra'" 5
node16.example.com openshift_node_group_name='node-config-infra'" 6
```

1 2 3 4 5 6 各ノードには、一般的なアプリケーションまたはインフラストラクチャーアプリケーションのスケジューリングをそれらのノードで許可するかどうかを示すマークが付けられます。アプリケーションの制限方法は管理者が設定します。

5. インストーラーを実行します。

- OpenShift Container Platform の初回インストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

23.3.4.2.6. 例: アプリケーション、レジストリー、ロギングおよびメトリクス用のインデペンデントモード

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs 1
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-
role.kubernetes.io/infra=true'

openshift_metrics_install_metrics=true
```

```
openshift_metrics_hawkular_nodeselector={"node-
role.kubernetes.io/infra": "true"} 2
openshift_metrics_cassandra_nodeselector={"node-
role.kubernetes.io/infra": "true"} 3
openshift_metrics_heapster_nodeselector={"node-
role.kubernetes.io/infra": "true"} 4
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-
registry-block" 5

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-
role.kubernetes.io/infra": "true"} 6
openshift_logging_curator_nodeselector={"node-
role.kubernetes.io/infra": "true"} 7
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":
"true"} 8
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi                              9
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-
block"       10

openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_is_native=false
openshift_storage_glusterfs_heketi_is_native=true
openshift_storage_glusterfs_heketi_executor=ssh
openshift_storage_glusterfs_heketi_ssh_port=22
openshift_storage_glusterfs_heketi_ssh_user=root
openshift_storage_glusterfs_heketi_ssh_sudo=false
openshift_storage_glusterfs_heketi_ssh_keyfile="/root/.ssh/id_rsa"

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=fals
e
openshift_storage_glusterfs_registry_is_native=false
openshift_storage_glusterfs_registry_heketi_is_native=true
openshift_storage_glusterfs_registry_heketi_executor=ssh
openshift_storage_glusterfs_registry_heketi_ssh_port=22
openshift_storage_glusterfs_registry_heketi_ssh_user=root
openshift_storage_glusterfs_registry_heketi_ssh_sudo=false
openshift_storage_glusterfs_registry_heketi_ssh_keyfile="/root/.ssh/
id_rsa"
```

1 2 3 4 6 7 8 統合 OpenShift Container レジストリーは、管理者が OpenShift Container Platform クラスターにサービスを提供するためにデプロイしたアプリケーションである「インフラストラクチャー」アプリケーション専用のノードで実行することが推奨されます。インフラストラクチャーアプリケーション用のノードは、管理者が選択し、それにラベルを付けます。

5 10 ロギングとメトリクスに使用する StorageClass を指定します。この名前は、ターゲットの GlusterFS クラスター (例: **glusterfs-<name>-block**) 名から生成されます。この例では **registry** にデフォルト設定されています。

9 OpenShift ロギングでは、PVC のサイズを指定する必要があります。ここで指定される値は単なる例であり、推奨される値ではありません。

2. **[OSEv3:children]** セクションに **glusterfs** と **glusterfs_registry** を追加し、**[glusterfs]** グループと **[glusterfs_registry]** グループを有効にします。

```
[OSEv3:children]
...
glusterfs
glusterfs_registry
```

3. **[glusterfs]** セクションと **[glusterfs_registry]** セクションを追加し、両セクションに GlusterFS ストレージをホストするストレージノードを入力します。ノードごとに、**glusterfs_devices** を、GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくと も 1 つのデバイスを一覧に含める必要があります。各デバイスは、パーティションや LVM PV がないベアでなければなりません。また、**glusterfs_ip** をノードの IP アドレスに設定します。変数は以下の形式で指定します。

```
<hostname_or_ip> glusterfs_ip=<ip_address> glusterfs_devices='[ "
</path/to/device1/>", "</path/to/device2>", ... ]'
```

以下は例を示しています。

```
[glusterfs]
gluster1.example.com glusterfs_ip=192.168.10.11 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster2.example.com glusterfs_ip=192.168.10.12 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster3.example.com glusterfs_ip=192.168.10.13 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'

[glusterfs_registry]
gluster4.example.com glusterfs_ip=192.168.10.14 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster5.example.com glusterfs_ip=192.168.10.15 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
gluster6.example.com glusterfs_ip=192.168.10.16 glusterfs_devices='[
"/dev/xvdc", "/dev/xvdd" ]'
```

4. インストーラーを実行します。

- OpenShift Container Platform の初回インストール:

```
ansible-playbook -i <path_to_inventory_file>
```

```
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/config.yml
```

23.3.5. コンバージドモードのアンインストール

コンバージドモードの場合、OpenShift Container Platform のインストールには、クラスターから全リソースおよびアーティファクトをアンインストールするための Playbook が同梱されています。この Playbook を使用するには、コンバージドモードのターゲットインスタンスをインストールするのに使用した元のインベントリーファイルを指定して、以下の Playbook を実行します。

```
# ansible-playbook -i <path_to_inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/uninstall.yml
```

さらに、Playbook は、**openshift_storage_glusterfs_wipe** と呼ばれる変数の使用をサポートします。これは、有効化されている場合には、Red Hat Gluster Storage バックエンドストレージに使用されていたブロックデバイス上のデータを破棄します。**openshift_storage_glusterfs_wipe** 変数を使用するには、以下を実行します。

```
# ansible-playbook -i <path_to_inventory_file> -e
"openshift_storage_glusterfs_wipe=true"
/usr/share/ansible/openshift-ansible/playbooks/openshift-
glusterfs/uninstall.yml
```

警告

この手順ではデータが破棄されるので、注意して進めてください。

23.3.6. プロビジョニング

GlusterFS ボリュームは、静的または動的にプロビジョニングできます。静的プロビジョニングは、すべての設定で使用できます。動的プロビジョニングは、コンバージドモードおよびインデペンデントモードでサポートされます。

23.3.6.1. 静的プロビジョニング

1. 静的プロビジョニングを有効にするには、最初に GlusterFS ボリュームを作成します。**gluster** コマンドラインインターフェースを使用してこれを行う方法については、『Red Hat Gluster Storage Administration Guide』を参照してください。また、**heketi-cli** を使用し

てこれを行う方法については、heketi プロジェクトサイトを参照してください。この例では、ボリュームに **myVol1** という名前を付けます。

2. **gluster-endpoints.yaml** で以下のサービスとエンドポイントを定義します。

```
---
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster (1)
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster (2)
subsets:
  - addresses:
      - ip: 192.168.122.221 (3)
    ports:
      - port: 1 (4)
  - addresses:
      - ip: 192.168.122.222 (5)
    ports:
      - port: 1 (6)
  - addresses:
      - ip: 192.168.122.223 (7)
    ports:
      - port: 1 (8)
```

(1)(2) これらの名前は一致している必要があります。

(3)(5)(7) **ip** の値には、Red Hat Gluster Storage サーバーのホスト名ではなく、実際の IP アドレスを指定する必要があります。

(4)(6)(8) ポート番号は無視されます。

3. OpenShift Container Platform マスターホストからサービスとエンドポイントを作成します。

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. サービスとエンドポイントが作成されたことを確認します。

```
$ oc get services
NAME                      CLUSTER_IP       EXTERNAL_IP   PORT(S)
SELECTOR        AGE
glusterfs-cluster         172.30.205.34    <none>        1/TCP
<none>          44s
```

```
$ oc get endpoints
NAME                ENDPOINTS
AGE
docker-registry     10.1.0.3:5000
4h
glusterfs-cluster
192.168.122.221:1,192.168.122.222:1,192.168.122.223:1   11s
kubernetes          172.16.35.3:8443
4d
```

注記

エンドポイントはプロジェクトごとに一意です。GlusterFS にアクセスする各プロジェクトには独自のエンドポイントが必要です。

5. ボリュームにアクセスするには、ボリューム上のファイルシステムにアクセスできるユーザー ID (UID) またはグループ ID (GID) でコンテナーを実行する必要があります。この情報は以下の方法で取得できます。

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0    myVol1 (1) (2)
```

(1) UID は 592 です。

(2) GID は 590 です。

6. **`gluster-pv.yaml`** で以下の PersistentVolume (PV) を定義します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume (1)
  annotations:
    pv.beta.kubernetes.io/gid: "590" (2)
spec:
  capacity:
    storage: 2Gi (3)
  accessModes: (4)
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster (5)
    path: myVol1 (6)
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

(1) ボリュームの名前です。

(2) GlusterFS ボリュームのルートの GID です。

3. このボリュームに割り当てられるストレージの量。
4. **accessModes** は、PV と PVC を一致させるためのラベルとして使用します。現時点で、それらは現時点ではいずれの形式のアクセス制御も定義しません。
5. 以前に作成されたエンドポイントリソースです。
6. アクセス対象の GlusterFS ボリュームです。

7. OpenShift Container Platform マスターホストから PV を作成します。

```
$ oc create -f gluster-pv.yaml
```

8. PV が作成されたことを確認します。

```
$ oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES
STATUS      CLAIM     REASON    AGE
gluster-default-volume   <none>    2147483648   RWX
Available                          2s
```

9. **gluster-claim.yaml** で、新規 PV にバインドする PersistentVolumeClaim (PVC) を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim   1
spec:
  accessModes:
  - ReadWriteMany       2
  resources:
     requests:
       storage: 1Gi     3
```

1. この要求名は、**volumes** セクションで Pod によって参照されます。
2. PV の **accessModes** に一致する必要があります。
3. この要求は、**1Gi** 以上の容量がある PV を検索します。

10. OpenShift Container Platform マスターホストから PVC を作成します。

```
$ oc create -f gluster-claim.yaml
```

11. PV と PVC がバインドされていることを確認します。

```
$ oc get pv
NAME          LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
gluster-pv    <none>    1Gi        RWX           Available
gluster-claim             37s
```

```
$ oc get pvc
NAME             LABELS    STATUS    VOLUME       CAPACITY
ACCESSMODES   AGE
gluster-claim    <none>    Bound     gluster-pv   1Gi        RWX
24s
```

注記

PVC はプロジェクトごとに一意です。GlusterFS ボリュームにアクセスする各プロジェクトには独自の PVC が必要です。PV は単一のプロジェクトにバインドされないため、複数のプロジェクトにまたがる PVC が同じ PV を参照する場合があります。

23.3.6.2. 動的プロビジョニング

1. 動的プロビジョニングを有効にするには、最初に **StorageClass** オブジェクト定義を作成します。以下の定義は、OpenShift Container Platform でこの例を使用するために必要な最小要件に基づいています。その他のパラメーターと仕様定義については、「動的プロビジョニングとストレージクラスの作成」を参照してください。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: glusterfs
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://10.42.0.0:8080" 1
  restauthenabled: "false" 2
```

1. heketi サーバーの URL です。
2. この例では認証が有効ではないため、**false** に設定します。

2. OpenShift Container Platform マスターホストから StorageClass を作成します。

```
# oc create -f gluster-storage-class.yaml
storageclass "glusterfs" created
```

3. 新たに作成される StorageClass を使用して PVC を作成します。以下は例になります。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: gluster1
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
        storage: 30Gi
 storageClassName: glusterfs
```

4. OpenShift Container Platform マスターホストから PVC を作成します。

```
# oc create -f glusterfs-dyn-pvc.yaml
persistentvolumeclaim "gluster1" created
```

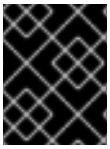
5. PVC を表示し、ボリュームが動的に作成され、PVC にバインドされていることを確認します。

```
# oc get pvc
NAME           STATUS VOLUME
CAPACITY    ACCESSMODES    STORAGECLASS    AGE
gluster1        Bound pvc-78852230-d8e2-11e6-a3fa-0800279cf26f
30Gi     RWX         gluster-dyn 42s
```

23.4. OPENSTACK CINDER を使用した永続ストレージ

23.4.1. 概要

OpenStack Cinder を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と OpenStack についてある程度の理解があることが前提となります。



重要

Cinder を使用して永続ボリューム (PV) を作成する前に、OpenStack 向けに OpenShft Container Platform を設定してください。

Kubernetes の 永続ボリュームフレームワークは、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。OpenStack Cinder ボリュームを動的にプロビジョニングできます。

永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスター全体で共有できます。ただし、Persistent Volume Claim (永続ボリューム要求) は、プロジェクトまたは namespace に固有で、ユーザーが要求できます。

重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.4.2. Cinder PV のプロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントするには、ストレージが基礎となるインフラストラクチャーに存在している必要があります。OpenShift Container Platform が OpenStack 用に設定されていることを確認した後、Cinder に必要なのは、Cinder ボリューム ID と **`PersistentVolume`** API のみになります。

23.4.2.1. 永続ボリュームの作成

注記

Cinder では、「リサイクル」回収ポリシーはサポートされません。

OpenShift Container Platform に PV を作成する前に、PV をオブジェクト定義に定義する必要があります。

1. オブジェクト定義を **cinder-pv.yaml** などのファイルに保存します。

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" (1)
spec:
  capacity:
    storage: "5Gi" (2)
  accessModes:
    - "ReadWriteOnce"
  cinder: (3)
    fsType: "ext3" (4)
    volumeID: "f37a03aa-6212-4c62-a805-9ce139fab180" (5)
```

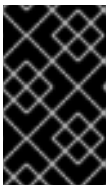
(1) 永続ボリューム要求または Pod で使用するボリューム名。

(2) このボリュームに割り当てられるストレージの量。

(3) ボリュームタイプ。今回の場合は **cinder** です。

(4) マウントするファイルシステムタイプです。

(5) 使用する Cinder ボリューム



重要

ボリュームをフォーマットしてプロビジョニングした後には、**fstype** パラメーターの値は変更しないでください。この値を変更すると、データの損失や、Pod の障害につながる可能性があります。

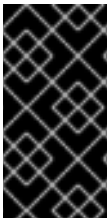
2. 永続ボリュームを作成します。

```
# oc create -f cinder-pv.yaml

persistentvolume "pv0001" created
```

3. 永続ボリュームの存在を確認します。

```
# oc get pv

NAME      LABELS    CAPACITY    ACCESSMODES    STATUS      CLAIM
REASON    AGE
pv0001    <none>    5Gi         RWO            Available
2s
```

次に、ユーザーは永続ボリューム要求 (Persistent Volume Claim) を使用してストレージを要求できるので、この新規の永続ボリュームを活用できます。



重要

Persistent Volume Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から Persistent Volume Claim (永続ボリューム要求) にアクセスしようとすると、Pod にエラーが発生します。

23.4.2.2. Cinder の PV 形式

OpenShift Container Platform は、ボリュームをマウントしてコンテナーに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない Cinder ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

23.4.2.3. Cinder ボリュームのセキュリティー

お使いのアプリケーションで Cinder PV を使用する場合に、そのデプロイメント設定にセキュリティーを追加します。

注記

Cinder ボリュームを実装する前に、ボリュームのセキュリティーを確認します。

1. 適切な **fsGroup** ストラテジーを使用するSCC を作成します。
2. サービスアカウントを作成して、そのアカウントを SCC に追加します。

```
[source,bash]
$ oc create serviceaccount <service_account>
$ oc adm policy add-scc-to-user <new_scc> -z <service_account> -n
<project>
```

3. アプリケーションのデプロイ設定で、サービスアカウント名と **securityContext** を指定します。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1  1
  selector:    2
    name: frontend
  template:    3
    metadata:
      labels:  4
        name: frontend  5
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
```

```
      ports:
      - containerPort: 8080
        protocol: TCP
    restartPolicy: Always
    serviceAccountName: <service_account> 6
    securityContext:
      fsGroup: 7777 7
```

1. 実行する Pod のコピー数です。
2. 実行する Pod のラベルセレクターです。
3. コントローラーが作成する Pod のテンプレートです。
4. Pod のラベルには、ラベルセレクターからのラベルが含まれている必要があります。
5. パラメーターの拡張後の名前の最大長さは 63 文字です。
6. 作成したサービスアカウントを指定します。
7. Pod の `fsGrou` を指定します。

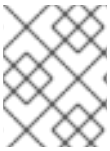
23.5. CEPH RADOS ブロックデバイス (RBD) を使用した永続ストレージ

23.5.1. 概要

OpenShift Container Platform クラスターでは、Ceph RBD を使用して永続ストレージをプロビジョニングできます。

永続ボリューム (PV) と Persistent Volume Claim (永続ボリューム要求、PVC)は、単一プロジェクトでボリュームを共有できます。PV 定義に含まれている Ceph RBD 固有の情報は Pod 定義で直接定義することも可能ですが、この方法だとボリュームが一意のクラスターリソースとして作成されず、競合の影響を受けやすくなります。

このトピックでは、OpenShift Container Platform と Ceph RBD についてある程度理解していることを前提とします。OpenShift Container Platform 永続ボリューム (PV) の一般的なフレームワークについての詳細は、永続ストレージの概念に関するトピックを参照してください。



注記

本書では、**プロジェクト**と **namespace** は区別せずに使用されています。両者の関係については、「プロジェクトおよびユーザー」を参照してください。

重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.5.2. プロビジョニング

Ceph ボリュームをプロビジョニングするには、以下が必要になります。

- 基礎となるインフラストラクチャーの既存ストレージデバイス。

- OpenShift Container Platform シークレットオブジェクトで使用される Ceph キー。
- Ceph イメージ名。
- ブロックストレージ上部のファイルシステムタイプ (ext4 など)。
- クラスター内のスケジュール可能な各 OpenShift Container Platform ノードにインストールされた **ceph-common**。

```
# yum install ceph-common
```

23.5.2.1. Ceph シークレットの作成

承認キーをシークレット設定に定義します。これは後に OpenShift Container Platform で使用できるように base64 に変換されます。

注記

Ceph ストレージを使用して永続ボリュームをサポートするには、シークレットを PVC や Pod と同じプロジェクトに作成する必要があります。シークレットは、単にデフォルトプロジェクトに置くことはできません。

1. Ceph MON ノードで **ceph auth get-key** を実行し、**client.admin** ユーザーのキー値を表示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFBOFF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ==
```

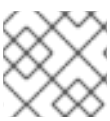
2. シークレット定義を **ceph-secret.yaml** などのファイルに保存し、シークレットを作成します。

```
$ oc create -f ceph-secret.yaml
```

3. シークレットが作成されたことを確認します。

```
# oc get secret ceph-secret
NAME          TYPE      DATA      AGE
ceph-secret   Opaque    1         23d
```

23.5.2.2. 永続ボリュームの作成



注記

Ceph RBD では、「リサイクル」回収ポリシーはサポートされません。

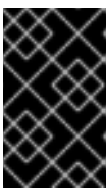
開発者は、PVC を参照するか、Pod 仕様の **volumes** セクションにある Gluster ボリュームプラグインを直接参照することによって Ceph RBD ストレージを要求します。PVC は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から PV にアクセスしようとすると、Pod エラーが発生します。

1. OpenShift Container Platform に PV を作成する前に、PV をオブジェクト定義に定義します。

例23.3 Ceph RBD を使用した永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv 1
spec:
  capacity:
    storage: 2Gi 2
  accessModes:
    - ReadWriteOnce 3
  rbd: 4
    monitors: 5
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret 6
    fsType: ext4 7
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1. Pod 定義で参照されるか、または各種の **oc** ボリュームコマンドで表示される PV の名前。
2. このボリュームに割り当てられるストレージの量。
3. **accessModes** は、PV と PVC を一致させるためのラベルとして使用されます。現時点で、これらはいずれの形式のアクセス制御も定義しません。すべてのブロックストレージは単一ユーザーに対して定義されます (非共有ストレージ)。
4. 使用されるボリュームタイプ。この場合は、**rbd** プラグインです。
5. Ceph モニターの IP アドレスとポートの配列。
6. OpenShift Container Platform から Ceph サーバーへのセキュアな接続を作成するために使用される Ceph シークレット。
7. Ceph RBD ブロックデバイスにマウントされるファイルシステムタイプ。



重要

ボリュームをフォーマットしてプロビジョニングした後に **fstype** パラメーターの値を変更すると、データ損失や Pod エラーが発生する可能性があります。

2. 定義を **ceph-pv.yaml** などのファイルに保存し、PV を作成します。

```
# oc create -f ceph-pv.yaml
```

3. 永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES
STATUS      CLAIM     REASON    AGE
ceph-pv                  <none>    2147483648   RWO
Available                          2s
```

4. 新規 PV にバインドされる PVC を作成します。

例23.4 PVC オブジェクト定義

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes: 1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi 2
```

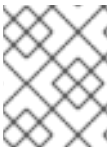
1 **accessModes** はアクセス権を実施しません。代わりに PV を PVC に一致させるラベルとして機能します。

2 この要求は、**2Gi** 以上の容量がある PV を検索します。

5. 定義をファイル (**ceph-claim.yaml** など) に保存し、PVC を作成します。

```
# oc create -f ceph-claim.yaml
```

23.5.3. Ceph ボリュームのセキュリティー



注記

Ceph RBD ボリュームを実装する前に、「ボリュームのセキュリティー」トピックの詳細を参照してください。

共有ボリューム (NFS および GlusterFS) とブロックボリューム (Ceph RBD、iSCSI、およびほとんどのクラウドストレージ) の大きな違いは、Pod 定義またはコンテナーイメージで定義されたユーザー ID とグループ ID がターゲットの物理ストレージに適用されることです。これはブロックデバイスの所有権の管理と呼ばれます。たとえば、Ceph RBD マウントで所有者が **123** に、グループ ID が **567** に設定されていて、Pod で **runAsUser** が **222** に、**fsGroup** が **7777** に定義されている場合、Ceph RBD 物理マウントの所有権は **222:7777** に変更されます。

注記

ユーザー ID とグループ ID が Pod 仕様で定義されていない場合でも、生成される Pod では、これらの ID のデフォルト値が一致する SCC またはプロジェクトに基づいて定義されることがあります。詳細については、「ボリュームのセキュリティー」トピックの詳細を参照してください。このトピックでは、SCC のストレージの側面とデフォルト値について説明しています。

Pod の **securityContext** 定義で **fsGroup** スタンザを使用して、Pod で Ceph RBD ボリュームのグループ所有権を定義します。

```
spec:
  containers:
    - name:
    ...
  securityContext: 1
    fsGroup: 7777 2
```

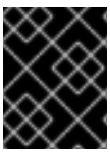
1 **securityContext** は特定のコンテナーの配下ではなく、この Pod レベルで定義します。

2 Pod 内のすべてのコンテナーは同じ fsGroup ID を持ちます。

23.6. AWS ELASTIC BLOCK STORE を使用した永続ストレージ

23.6.1. 概要

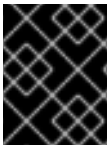
OpenShift Container Platform では、AWS Elastic Block Store ボリューム (EBS) がサポートされます。AWS EC2 を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と AWS についてある程度の理解があることが前提となります。



重要

AWS を使用して永続ボリュームを作成する前に、まず OpenShift Container Platform を AWS ElasticBlockStore 用に適切に設定する必要があります。

Kubernetes 永続ボリュームフレームワークは、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようになります。AWS Elastic Block Store ボリュームは、動的にプロビジョニングできます。永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスター全体で共有できます。ただし、Persistent Volume Claim (永続ボリューム要求)は、プロジェクトまたは namespace に固有で、ユーザーが要求できます。



重要

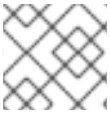
インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.6.2. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントするには、基礎となるインフラストラクチャーにストレージが存在している必要があります。OpenShift Container Platform が AWS

Elastic Block Store 用に設定されていることを確認した後、OpenShift と AWS に必要になるのは、AWS EBS ボリューム ID と **PersistentVolume** API のみです。

23.6.2.1. 永続ボリュームの作成



注記

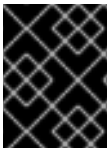
AWS では、「リサイクル」回収ポリシーはサポートされません。

OpenShift Container Platform に永続ボリュームを作成する前に、永続ボリュームをオブジェクト定義で定義する必要があります。

例23.5 AWS を使用した永続ボリュームオブジェクトの定義

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  awsElasticBlockStore: 3
    fsType: "ext4" 4
    volumeID: "vol-f37a03aa" 5
```

1. ボリュームの名前です。これは Persistent Volume Claim (永続ボリューム要求)を使用して、または Pod からボリュームを識別するために使用されます。
2. このボリュームに割り当てられるストレージの量。
3. これは使用されるボリュームタイプを定義します。この場合は、**awsElasticBlockStore** プラグインです。
4. マウントするファイルシステムタイプです。
5. これは使用される AWS ボリュームです。



重要

ボリュームをフォーマットしてプロビジョニングした後に **fstype** パラメーターの値を変更すると、データ損失や Pod エラーが発生する可能性があります。

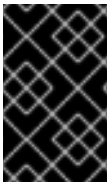
定義を **aws-pv.yaml** などのファイルに保存し、永続ボリュームを作成します。

```
# oc create -f aws-pv.yaml
persistentvolume "pv0001" created
```

永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM     REASON
AGE
pv0001    <none>    5Gi        RWO           Available
2s
```

次に、ユーザーは永続ボリューム要求 (Persistent Volume Claim) を使用してストレージを要求できるので、この新規の永続ボリュームを活用できます。



重要

Persistent Volume Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

23.6.2.2. ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナーに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない AWS ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

23.6.2.3. ノード上の EBS ボリュームの最大数

OpenShift Container Platform では、デフォルトで 1 つのノードに最大 39 の EBS ボリュームを割り当てることができます。この制限は、AWS ボリュームの制限に合致します。

OpenShift Container Platform では、環境変数 **KUBE_MAX_PD_VOLS** を設定することで、より大きな制限値を設定できます。ただし、AWS では、割り当てられるデバイスに特定の命名スキーム (AWS デバイスの命名) を使用する必要があります。この命名スキームでは、最大で 52 のボリュームしかサポートされません。これにより、OpenShift Container Platform 経由でノードに割り当てることができるボリュームの数が 52 に制限されます。

23.7. GCE PERSISTENT DISK を使用した永続ストレージ

23.7.1. 概要

OpenShift Container Platform では、GCE Persistent Disk ボリューム (gcePD) がサポートされます。GCE を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と GCE についてある程度の理解があることが前提となります。

重要

GCE を使用して永続ボリュームを作成する前に、まず OpenShift Container Platform を GCE Persistent Disk 用に適切に設定する必要があります。

Kubernetes の 永続ボリュームフレームワークは、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。また、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。GCE Persistent Disk ボリュームは、動的にプロビジョニ

ングできます。永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスター全体で共有できます。ただし、Persistent Volume Claim (永続ボリューム要求) は、プロジェクトまたは namespace に固有で、ユーザーが要求できます。

重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.7.2. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントするには、基礎となるインフラストラクチャーにストレージが存在している必要があります。OpenShift Container Platform が GCE Persistent Disk 用に設定されていることを確認した後、OpenShift Container Platform と GCE に必要となるのは、GCE Persistent Disk ボリューム ID と **PersistentVolume** API のみです。

23.7.2.1. 永続ボリュームの作成

注記

GCE では、「リサイクル」回収ポリシーはサポートされません。

OpenShift Container Platform に永続ボリュームを作成する前に、永続ボリュームをオブジェクト定義で定義する必要があります。

例23.6 GCE を使用した永続ボリュームオブジェクトの定義

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" (1)
spec:
  capacity:
    storage: "5Gi" (2)
  accessModes:
    - "ReadWriteOnce"
  gcePersistentDisk: (3)
    fsType: "ext4" (4)
    pdName: "pd-disk-1" (5)
```

(1) ボリュームの名前です。これは Persistent Volume Claim (永続ボリューム要求)を使用して、または Pod からボリュームを識別するために使用されます。

(2) このボリュームに割り当てられるストレージの量。

(3) これは使用されるボリュームタイプを定義します。この場合は、**gcePersistentDisk** プラグインです。

(4) マウントするファイルシステムタイプです。

(5) これは使用される GCE Persistent Disk ボリュームです。

重要

ボリュームをフォーマットしてプロビジョニングした後に **fstype** パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

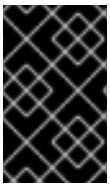
定義を **gce-pv.yaml** などのファイルに保存し、永続ボリュームを作成します。

```
# oc create -f gce-pv.yaml
persistentvolume "pv0001" created
```

永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM     REASON
AGE
pv0001    <none>    5Gi        RWO           Available
2s
```

次に、ユーザーは永続ボリューム要求 (Persistent Volume Claim) を使用してストレージを要求できるので、この新規の永続ボリュームを活用できます。



重要

Persistent Volume Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

23.7.2.2. ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナーに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない GCE ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

23.8. ISCSI を使用した永続ストレージ

23.8.1. 概要

iSCSI を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と iSCSI についてある程度の理解があることが前提となります。

Kubernetes の 永続ボリュームフレームワークは、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。

重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.8.2. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントする前に、基礎となるインフラストラクチャーにストレージが存在することを確認します。iSCSI に必要になるのは、iSCSI ターゲットポータル、有効な iSCSI 修飾名 (IQN)、有効な LUN 番号、ファイルシステムタイプ、および **PersistentVolume** API のみです。

オプションで、マルチマスポータルと challenge ハンドシェイク認証プロトコル (CHAP) 設定を提供できます。

注記

iSCSI では、「リサイクル」回収ポリシーはサポートされません。

例23.7 永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
     targetPortal: 10.16.154.81:3260
     portals: ['10.16.154.82:3260', '10.16.154.83:3260']
     iqn: iqn.2014-12.example.server:storage.target00
     lun: 0
     fsType: 'ext4'
     readOnly: false
     chapAuthDiscovery: true
     chapAuthSession: true
     secretRef:
       name: chap-secret
```

23.8.2.1. ディスククォータの実施

LUN パーティションを使用してディスククォータとサイズ制限を実施します。それぞれの LUN には 1 つの永続ボリュームです。Kubernetes では、永続ボリュームに一意の名前を使用する必要があります。

上記の方法でクォータを実施すると、エンドユーザーは永続ストレージを具体的な量 (10Gi など) で要求することができ、これを同等またはそれ以上の容量の対応するボリュームに一致させることができます。

23.8.2.2. iSCSI ボリュームのセキュリティー

ユーザーは **PersistentVolumeClaim** でストレージを要求します。この要求はユーザーの namespace にのみ存在し、同じ namespace 内の Pod からのみ参照できます。namespace をまたいで永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

それぞれの iSCSI LUN は、クラスター内のすべてのノードからアクセスできる必要があります。

23.8.2.3. iSCSI のマルチパス化

iSCSI ベースのストレージの場合は、複数のターゲットポータルの IP アドレスに同じ IQN を使用することでマルチパスを設定できます。マルチパス化により、パス内の 1 つ以上のコンポーネントで障害が発生した場合でも、永続ボリュームにアクセスできることができます。

Pod 仕様でマルチパスを指定するには、**portals** フィールドを使用します。以下は例になります。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi_pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260'] 1
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    readOnly: false
```

1 **portals** フィールドを使用してターゲットポータルを追加します。

23.8.2.4. iSCSI のカスタムイニシエーター IQN

iSCSI ターゲットが特定に IQN に制限されている場合に、カスタムイニシエーターの iSCSI Qualified Name (IQN) を設定します。ただし、iSCSI PV が割り当てられているノードが必ず、これらの IQN を使用する保証はありません。

カスタムのイニシエーター IQN を指定するには、**initiatorName** フィールドを使用します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi_pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260']
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    initiatorName: iqn.2016-04.test.com:custom.iqn 1
    fsType: ext4
    readOnly: false
```

-

1 カスタムのイニシエーター IQN を追加するには、**initiatorName** フィールドを使用します。

23.9. ファイバーチャネルを使用した永続ストレージ

23.9.1. 概要

ファイバーチャネルを使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と Fibre Channel についてある程度の理解があることが前提となります。

Kubernetes の 永続ボリュームフレームワークは、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。

重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.9.2. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントするには、基礎となるインフラストラクチャーにストレージが存在している必要があります。ファイバーチャネルの永続ストレージに必要になるのは、**targetWWNs** (ファイバーチャネルターゲットのワールドワイド名の配列)、有効な **LUN** 番号、ファイルシステムタイプ、および **PersistentVolume** API のみです。 永続ボリュームと LUN は 1 対 1 でマッピングされます。

注記

ファイバーチャネルでは、「リサイクル」回収ポリシーはサポートされません。

永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  fc:
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5'] 1
    lun: 2
    fsType: ext4
```

1 ファイバーチャネルI WWN は、**/dev/disk/by-path/pci-<IDENTIFIER>-fc-0x<WWN>-lun-<LUN#>** として識別されます。ただし、**WWN** までのパス (**0x** を含む) と WWN の後の文字 (**-** (ハイフン) を含む) を入力する必要はありません。

重要

ボリュームをフォーマットしてプロビジョニングした後に **fstype** パラメーターの値を変更すると、データ損失や Pod エラーが発生する可能性があります。

23.9.2.1. ディスククォータの実施

LUN パーティションを使用してディスククォータとサイズ制限を実施します。それぞれの LUN には 1 つの永続ボリュームです。Kubernetes では、永続ボリュームに一意の名前を使用する必要があります。

上記の方法でクォータを実施すると、エンドユーザーは永続ストレージを具体的な量 (10Gi など) で要求することができ、これを同等またはそれ以上の容量の対応するボリュームに一致させることができます。

23.9.2.2. ファイバーチャネルボリュームのセキュリティー

ユーザーは **PersistentVolumeClaim** でストレージを要求します。この要求はユーザーの namespace にのみ存在し、同じ namespace 内の Pod からのみ参照できます。namespace をまたいで永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

それぞれのファイバーチャネル LUN は、クラスター内のすべてのノードからアクセスできる必要があります。

23.10. AZURE DISK を使用した永続ストレージ

23.10.1. 概要

OpenShift Container Platform では、Microsoft Azure Disk ボリュームがサポートされます。Azure を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と Azure についてある程度の理解があることが前提となります。

Kubernetes の 永続ボリュームフレームワークは、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。

Azure Disk ボリュームは、動的にプロビジョニングできます。永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスター全体で共有できます。ただし、Persistent Volume Claim (永続ボリューム要求) は、プロジェクトまたは namespace に固有で、ユーザーが要求できます。

重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.10.2. 前提条件

Azure を使用して永続ボリュームを作成する前に、OpenShift Container Platform クラスターが以下の要件を満たしていることを確認してください。

- まず、OpenShift Container Platform を Azure Disk 用に設定する必要があります。

- インフラストラクチャー内の各ノードは、Azure 仮想マシン名に一致している必要があります。
- それぞれのノードホストは、同じリソースグループに属している必要があります。

23.10.3. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントするには、基礎となるインフラストラクチャーにストレージが存在している必要があります。OpenShift Container Platform が Azure Disk 用に設定されていることを確認した後、OpenShift Container Platform と Azure に必要になるのは、Azure Disk Name および Disk URI と **PersistentVolume** API のみです。

23.10.4. Azure Disk でのリージョンクラウドの設定

Azure には、インスタンスをデプロイする複数のリージョンがあります。必要なリージョンを指定するには、以下を **azure.conf** ファイルに追加します。

```
cloud: <region>
```

リージョンは以下のいずれかになります。

- ドイツクラウド: **AZUREGERMANCLOUD**
- 中国クラウド: **AZURECHINACLOUD**
- パブリッククラウド: **AZUREPUBLICCLOUD**
- 米国クラウド: **AZUREUSGOVERNMENTCLOUD**

23.10.4.1. 永続ボリュームの作成

注記

Azure では、**リサイクル**回収ポリシーはサポートされません。

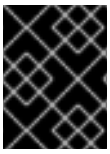
OpenShift Container Platform に永続ボリュームを作成する前に、永続ボリュームをオブジェクト定義で定義する必要があります。

例23.8 Azure を使用した永続ボリュームオブジェクトの定義

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" 1
spec:
  capacity:
    storage: "5Gi" 2
  accessModes:
    - "ReadWriteOnce"
  azureDisk: 3
    diskName: test2.vhd 4
    diskURI: https://someacount.blob.core.windows.net/vhds/test2.vhd 5
```

```
    cachingMode: ReadWrite  6
    fsType: ext4  7
    readOnly: false  8
```

1. ボリュームの名前です。これは Persistent Volume Claim (永続ボリューム要求)を使用して、または Pod からボリュームを識別するために使用されます。
2. このボリュームに割り当てられるストレージの量。
3. これは使用されるボリュームタイプを定義します (この例では、**azureDisk** プラグイン)。
4. Blob ストレージのデータディスクの名前。
5. Blob ストレージのデータディスクの URI
6. ホストのキャッシングモード: None、ReadOnly、または ReadWrite。
7. マウントするファイルシステムタイプ (**ext4**、**xfs** など)。
8. デフォルトは **false** (読み取り/書き込み) です。ここで **ReadOnly** を指定すると、**VolumeMounts** で **ReadOnly** 設定が強制的に実行されます。



重要

ボリュームをフォーマットしてプロビジョニングした後に **fsType** パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

1. 定義を **azure-pv.yaml** などのファイルに保存し、永続ボリュームを作成します。

```
# oc create -f azure-pv.yaml
persistentvolume "pv0001" created
```

2. 永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
pv0001    <none>    5Gi        RWO           Available
2s
```

これで、Persistent Volume Claim (永続ボリューム要求) を使用してストレージを要求し、新規の永続ボリュームを活用できるようになります。

重要

Azure Disk PVC を介してボリュームがマウントされている Pod の場合、新規ノードへの Pod のスケジューリングに数分の時間がかかります。**ディスクの割り当て解除**操作が完了するまで 2 ～ 3 分待ってから、新規デプロイメントを開始してください。**ディスクの割り当て解除**操作が完了する前に新規の Pod の作成要求が開始されると、Pod の作成によって開始された**ディスクの割り当て**操作が失敗し、結果として Pod の作成が失敗します。

重要

Persistent Volume Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

23.10.4.2. ボリュームのフォーマット

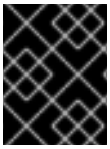
OpenShift Container Platform は、ボリュームをマウントしてコンテナーに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない Azure ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

23.11. AZURE FILE を使用した永続ストレージ

23.11.1. 概要

OpenShift Container Platform では、Microsoft Azure File ボリュームがサポートされます。Azure を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と Azure についてのある程度の理解があることが前提となります。



重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

23.11.2. 作業を開始する前の注意事項

1. すべてのノードに **samba-client**、**samba-common**、および **cifs-utils** をインストールします。

```
$ sudo yum install samba-client samba-common cifs-utils
```

2. すべてのノードで SELinux ブール値を有効にします。

```
$ /usr/sbin/setsebool -P virt_use_samba on
$ /usr/sbin/setsebool -P virt_sandbox_use_samba on
```

3. **mount** コマンドを実行して **dir_mode** および **file_mode** パーミッションなどを確認します。

```
$ mount
```

dir_mode および **file_mode** のパーミッションが **0755** に設定されている場合には、デフォルト値 **0755** を **0777** または **0775** に変更します。OpenShift Container Platform 3.9 では、デフォルトの **dir_mode** および **file_mode** パーミッションが **0777** から **0755** に変更されるので、この手動の手順が必要です。以下の例では、変更された値が含まれる設定ファイルを紹介しています。

Azure File での MySQL および PostgresSQL を使用する場合の留意事項

- Azure File がマウントされるディレクトリーの所有者 UID は、コンテナーの UID とは異なります。
- MySQL コンテナーは、マウントされたディレクトリーのファイルの所有者パーミッションを変更します。所有者 UID とコンテナープロセスの UID が一致しないので、この操作に失敗してしまいます。そのため、Azure File で MySQL を実行するには、以下を行うようにしてください。
  - PV 設定ファイルの **runAsUser** 変数に、Azure File のマウントされたディレクトリー UID を指定します。

```
spec:
  containers:
    ...
  securityContext:
    runAsUser: <mounted_dir_uid>
```

  - PV 設定ファイルの **mountOptions** でコンテナーのプロセス UID を指定します。

```
mountOptions:
  - dir_mode=0700
  - file_mode=0600
  - uid=<conatiner_process_uid>
  - gid=0
```

- PostgreSQL は Azure File との併用はできません。これは、PostgreSQL に Azure File ディレクトリーのハードリンクが必要であるにもかかわらず、Azure File では ハードリンク をサポートしていないため、Pod の起動に失敗します。

PV 設定ファイル例

```
# azf-pv.yml
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "azpv"
spec:
  capacity:
    storage: "1Gi"
  accessModes:
    - "ReadWriteMany"
  azureFile:
    secretName: azure-secret
    shareName: azftest
    readOnly: false
  mountOptions:
    - dir_mode=0777
    - file_mode=0777
```

ストレージクラスの設定ファイル例

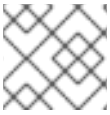
```
$ azure-file-sc.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```
  name: azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
parameters:
  storageAccount: ocp39str
  location: centralus
```

23.11.3. Azure File でのリージョンクラウドの設定

Azure Disk は複数のリージョンクラウドに対応していますが、Azure File は、エンドポイントがハードコードされているために Azure パブリッククラウドのみをサポートしています。

23.11.4. PV の作成



注記

Azure File では、**リサイクル**回収ポリシーはサポートされません。

23.11.5. Azure Storage Account シークレットの作成

Azure Storage Account の名前とキーをシークレット設定に定義します。これは後に OpenShift Container Platform で使用できるように base64 に変換されます。

1. Azure Storage Account の名前とキーを取得し、base64 にエンコードします。

```
apiVersion: v1
kind: Secret
metadata:
  name: azure-secret
type: Opaque
data:
  azurestorageaccountname: azhzdGVzdA==
  azurestorageaccountkey:
eElGMXpKYm5ub2pGTE1Ta0JwNTBteDAyckhzTUsyc2pVN21GdDRMMTNob0I3ZHJBYUo4
akQ2K0E0NDNqSm9nVjd5MkZVT2hRQ1dQbU02WWFOSHk3cWc9PQ==
```

2. シークレット定義を **azure-secret.yaml** などのファイルに保存し、シークレットを作成します。

```
$ oc create -f azure-secret.yaml
```

3. シークレットが作成されたことを確認します。

```
$ oc get secret azure-secret
NAME           TYPE      DATA      AGE
azure-secret   Opaque    1         23d
```

4. OpenShift Container Platform に PV を作成する前に、PV をオブジェクト定義に定義します。

Azure File を使用した PV オブジェクト定義の例

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" (1)
spec:
  capacity:
    storage: "5Gi" (2)
  accessModes:
    - "ReadWriteMany"
  azureFile: (3)
    secretName: azure-secret (4)
    shareName: example (5)
    readOnly: false (6)
```

(1) ボリュームの名前です。これは PVC (永続ボリューム要求) を使用して、または Pod からボリュームを識別するために使用されます。

(2) このボリュームに割り当てられるストレージの量。

(3) これは使用されるボリュームタイプを定義します (**azureFile** プラグイン)。

(4) 使用されるシークレットの名前。

(5) ファイル共有の名前。

(6) デフォルトは **false** (読み取り/書き込み) です。ここで **ReadOnly** を指定すると、**VolumeMounts** で **ReadOnly** 設定が強制的に実行されます。

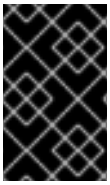
5. 定義を **azure-file-pv.yaml** などのファイルに保存し、PV を作成します。

```
$ oc create -f azure-file-pv.yaml
persistentvolume "pv0001" created
```

6. PV が作成されたことを確認します。

```
$ oc get pv
NAME      LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
pv0001    <none>    5Gi        RWM           Available
2s
```

これで、PVC (永続ボリューム要求) を使用してストレージを要求し、新規の永続ボリュームを活用できるようになります。



重要

PVC (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

23.12. FLEXVOLUME プラグインを使用した永続ストレージ

23.12.1. 概要

OpenShift Container Platform には、各種のストレージテクノロジーを使用するためにボリュームプラグインが組み込まれています。組み込みプラグインがないバックエンドのストレージを使用する場合は、FlexVolume ドライバーを使用して OpenShift Container Platform を拡張し、アプリケーションに永続ストレージを提供できます。

23.12.2. FlexVolume ドライバー

FlexVolume ドライバーは、クラスター内のすべてのマシン (マスターとノードの両方) の明確に定義されたディレクトリーに格納されている実行可能ファイルです。OpenShift Container Platform は、**flexVolume** をソースとする **PersistentVolume** によって表されるボリュームの割り当て、割り当て解除、マウント、またはアンマウントが必要になるたびに FlexVolume ドライバーを呼び出します。

ドライバーの最初のコマンドライン引数は常に操作名です。その他のパラメーターは操作ごとに異なります。ほとんどの操作は、JSON (JavaScript Object Notation) 文字列をパラメーターとして取ります。このパラメーターは完全な JSON 文字列であり、JSON データを含むファイルの名前ではありません。

FlexVolume ドライバーには以下が含まれます。

- すべての **flexVolume.options**。
- **kubernetes.io/** というプレフィックスが付いた **flexVolume** のいくつかのオプション。たとえば、**fsType** や **readwrite** などです。
- **kubernetes.io/secret/** というプレフィックスが付いた参照先シークレット (指定されている場合) の内容。

FlexVolume ドライバーの JSON 入力例

```
{
 "fooServer": "192.168.0.1:1234", (1)
        "fooVolumeName": "bar",
 "kubernetes.io/fsType": "ext4", (2)
 "kubernetes.io/readwrite": "ro", (3)
 "kubernetes.io/secret/<key name>": "<key value>", (4)
 "kubernetes.io/secret/<another key name>": "<another key value>",
}
```

1. **flexVolume.options** のすべてのオプション。
2. **flexVolume.fsType** の値。
3. **flexVolume.readOnly** に基づく **ro**/**rw**。
4. **flexVolume.secretRef** によって参照されるシークレットのすべてのキーと値。

OpenShift Container Platform は、ドライバーの標準出力に JSON データが含まれていると想定します。指定されていない場合、出力には操作の結果が示されます。

FlexVolume ドライバーのデフォルトの出力

```
{
```

```
 "status": "<Success/Failure/Not supported>",
 "message": "<Reason for success/failure>"
}
```

ドライバーの終了コードは、成功の場合は **0**、エラーの場合は **1** です。

操作はべき等です。つまり、すでに割り当てられているボリュームの割り当て操作や、すでにマウントされているボリュームのマウント操作は成功します。

FlexVolume ドライバーは以下の 2 つのモードで動作します。

- マスター実行の割り当て/割り当て解除操作あり
- マスター実行の割り当て/割り当て解除操作なし

attach/detach 操作は、OpenShift Container Platform マスターにより、ノードにボリュームを割り当てるため、およびノードからボリュームの割り当てを解除するために使用されます。これは何らかの理由でノードが応答不能になった場合に役立ちます。その後、マスターはノード上のすべての Pod を強制終了し、ノードからすべてのボリュームの割り当てを解除して、ボリュームを他のノードに割り当てることで、元のノードがまだ到達不能な状態であってもアプリケーションを再開できます。

重要

マスター実行の、別のマシンからのボリュームの割り当て解除は、すべてのストレージバックエンドでサポートされる訳ではありません。

23.12.2.1. マスター実行の割り当て/割り当て解除がある FlexVolume ドライバー

マスター制御の割り当て/割り当て解除をサポートする FlexVolume ドライバーは、以下の操作を実装する必要があります。

init

ドライバーを初期化します。マスターとノードの初期化中に呼び出されます。

- 引数: なし
- 実行場所: マスター、ノード
- 予期される出力: デフォルトの JSON

getvolumename

ボリュームの一意の名前を返します。この名前は、後続の **detach** 呼び出しで **<volume-name>** として使用されるため、すべてのマスターとノード間で一致している必要があります。**<volume-name>** の **/** 文字は自動的に **~** に置き換えられます。

- 引数: **<json>**
- 実行場所: マスター、ノード
- 予期される出力: デフォルトの JSON + **volumeName**:

```
{
 "status": "Success",
 "message": "",
```

```
 "volumeName": "foo-volume-bar" 1
}
```

1 ストレージバックエンド **foo** のボリュームの一意の名前。

attach

指定されたノードに、JSON で表現したボリュームを割り当てます。この操作は、ノード上のデバイスが既知の場合 (つまり、そのデバイスが実行前にストレージバックエンドによって割り当て済みの場合)、そのデバイスの名前を返します。デバイスが既知でない場合は、後続の **waitforattach** 操作によってノード上のデバイスが検出される必要があります。

- 引数: **<json> <node-name>**
- 実行場所: マスター
- 予期される出力: デフォルトの JSON + **device** (既知の場合)。

```
{
 "status": "Success",
 "message": "",
 "device": "/dev/xvda" 1
}
```

1 ノード上のデバイスの名前 (既知の場合)。

waitforattach

ボリュームがノードに完全に割り当てられ、デバイスが出現するまで待機します。前の **attach** 操作から **<device-name>** が返された場合は、それが入力パラメーターとして渡されます。そうでない場合、**<device-name>** は空であり、この操作によってノード上のデバイスを検出する必要があります。

- 引数: **<device-name> <json>**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON + **device**

```
{
 "status": "Success",
 "message": "",
 "device": "/dev/xvda" 1
}
```

1 ノード上のデバイスの名前。

detach

指定されたボリュームをノードから割り当て解除します。**<volume-name>** は、**getvolumename** 操作から返されるデバイスの名前です。**<volume-name>** の **/** 文字は、**~** に自動的に置き換えられます。

- 引数: **<volume-name> <node-name>**
- 実行場所: マスター
- 予期される出力: デフォルトの JSON

isattached

ボリュームがノードに割り当てられていることを確認します。

- 引数: **<json> <node-name>**
- 実行場所: マスター
- 予期される出力: デフォルトの JSON + **attached**

```
{
 "status": "Success",
 "message": "",
 "attached": true (1)
}
```

(1) ノードへのボリュームの割り当てのステータス。

mountdevice

ボリュームのデバイスをディレクトリーにマウントします。**<device-name>** は、前の **waitforattach** 操作から返されるデバイスの名前です。

- 引数: **<mount-dir> <device-name> <json>**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

unmountdevice

ボリュームのデバイスをディレクトリーからアンマウントします。

- 引数: **<mount-dir>**
- 実行場所: ノード

その他のすべての操作は、**{"status": "Not supported"}** と終了コード **1** を出して JSON を返します。

注記

OpenShift Container Platform 3.6 では、マスター実行の割り当て/割り当て解除操作はデフォルトで有効にされています。これらの操作は古いバージョンでも使用できる場合がありますが、その場合には明示的に有効にする必要があります。「コントローラー管理の割り当ておよび割り当て解除」を参照してください。有効にされていない場合、割り当て/割り当て解除操作は、ボリュームの割り当て/割り当て解除が実行される必要のあるノードで開始されます。FlexVolume ドライバー呼び出しの構文とすべてのパラメーターはどちらの場合も同じになります。

23.12.2.2. マスター実行の割り当て/割り当て解除がない FlexVolume ドライバー

マスター制御の割り当て/割り当て解除をサポートしない FlexVolume ドライバーは、ノードでのみ実行され、以下の操作を実装する必要があります。

`init`

ドライバーを初期化します。すべてのノードの初期化中に呼び出されます。

- 引数: なし
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

`mount`

ボリュームをディレクトリーにマウントします。これには、ノードへのボリュームの割り当て、ノードのデバイスの検出、その後のデバイスのマウントを含む、ボリュームのマウントに必要なあらゆる操作が含まれます。

- 引数: **`<mount-dir> <json>`**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

`unmount`

ボリュームをディレクトリーからアンマウントします。これには、ノードからのボリュームの割り当て解除など、アンマウント後のボリュームのクリーンアップに必要なあらゆる操作が含まれます。

- 引数: **`<mount-dir>`**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

その他のすべての操作は、**`{"status": "Not supported"}`** と終了コード **`1`** を出して JSON を返します。

23.12.3. FlexVolume ドライバーのインストール

FlexVolume ドライバーをインストールします。

1. この実行可能ファイルがクラスター内のすべてのマスターとノードに存在することを確認します。
2. この実行可能ファイルをボリュームプラグインのパス (**/usr/libexec/kubernetes/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>**) に配置します。

たとえば、ストレージ **`foo`** の FlexVolume ドライバーをインストールするには、実行可能ファイルを **/usr/libexec/kubernetes/kubelet-plugins/volume/exec/openshift.com~foo/foo** に配置します。

OpenShift Container Platform 3.10 では、**controller-manager** が静的 Pod として実行されるので、割り当ておよび割り当て解除の操作を行う FlexVolume バイナリーファイルは、外部の依存関係がなく、独立した実行可能ファイルでなければなりません。

Atomic ホストでは、FlexVolume プラグインディレクトリーのデフォルトの場所は **/etc/origin/kubelet-plugins/** です。FlexVolume の実行可能ファイルを、クラスター内の全マスターとノードの **/etc/origin/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>** ディレクトリーに配置する必要があります。

23.12.4. FlexVolume ドライバーを使用したストレージの使用

インストールされているストレージを参照するには、**PersistentVolume** オブジェクトを使用します。OpenShift Container Platform の各 **PersistentVolume** オブジェクトは、ストレージバックエンドの 1 つのストレージアセット (通常はボリューム) を表します。

FlexVolume ドライバーを使用した永続ボリュームオブジェクト定義例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 1Gi 2
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: openshift.com/foo 3
    fsType: "ext4" 4
    secretRef: foo-secret 5
    readOnly: true 6
    options: 7
      fooServer: 192.168.0.1:1234
      fooVolumeName: bar
```

1. ボリュームの名前。これはPersistent Volume Claim (永続ボリューム要求) を使用するか、または Pod からボリュームを識別するために使用されます。この名前は、バックエンドストレージのボリューム名とは異なるものにすることができます。
2. このボリュームに割り当てられるストレージの量。
3. ドライバーの名前。このフィールドは必須です。
4. ボリュームに存在するオプションのファイルシステム。このフィールドはオプションです。
5. シークレットへの参照。このシークレットのキーと値は、起動時に FlexVolume ドライバーに渡されます。このフィールドはオプションです。
6. 読み取り専用のフラグ。このフィールドはオプションです。
7. FlexVolume ドライバーの追加オプション。**options** フィールドでユーザーが指定するフラグに加え、以下のフラグも実行可能ファイルに渡されます。

```
"fsType":"<FS type>",
```

```
"readwrite":"<rw>",
"secret/key1":"<secret1>"
...
"secret/keyN":"<secretN>"
```

注記

シークレットは、呼び出しのマウント/マウント解除のためにだけ渡されます。

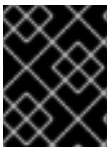
23.13. 永続ストレージ用の VMWARE VSPHERE ボリューム

23.13.1. 概要

OpenShift Container Platform では、VMWare vSphere の仮想マシンディスク (VMDK: Virtual Machine Disk) ボリュームがサポートされます。VMWare vSphere を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と VMWare vSphere についてのある程度の理解があることが前提となります。

OpenShift Container Platform の永続ボリューム (PV) フレームワークを使用すると、管理者がクラスターに永続ストレージをプロビジョニングできるようになるだけでなく、ユーザーが、基盤となるインフラストラクチャーに精通していなくてもこれらのリソースを要求できるようになります。VMWare vSphere VMDK ボリュームは、動的にプロビジョニングできます。

永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスター全体で共有できます。ただし、PVC (永続ボリューム要求) は、プロジェクトまたは namespace に固有で、ユーザーによる要求が可能です。



重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

前提条件

vSphere を使用して永続ボリュームを作成する前に、OpenShift Container Platform クラスターが以下の要件を満たしていることを確認してください。

- 最初に OpenShift Container Platform を vSphere 用に設定する必要があります。
- インフラストラクチャー内の各ノードホストは、vSphere 仮想マシン名に一致する必要があります。
- それぞれのノードホストは、同じリソースグループに属している必要があります。

重要

VMDK を使用する前に、以下のいずれかの方法を使用して VMDK を作成します。

- **vmkfstools** を使用して作成する。
  セキュアシェル (SSH) を使用して ESX にアクセスし、以下のコマンドを使用して vmdk ボリュームを作成します。

```
vmkfstools -c 2G
/vmfs/volumes/DatastoreName/volumes/myDisk.vmdk
```

- **vmware-vdiskmanager** を使用して作成する。

```
shell vmware-vdiskmanager -c -t 0 -s 40GB -a lsilogic
myDisk.vmdk
```

23.13.2. VMware vSphere ボリュームのプロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントするには、ストレージが基礎となるインフラストラクチャーに存在している必要があります。OpenShift Container Platform が vSpehere 用に設定されていることを確認した後、OpenShift Container Platform と vSphere に必要になるのは、VM フォルダーパス、ファイルシステムタイプ、および **PersistentVolume** API のみです。

23.13.2.1. 永続ボリュームの作成

OpenShift Container Platform に PV を作成する前に、PV をオブジェクト定義に定義する必要があります。

VMware vSphere を使用した PV オブジェクト定義の例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 2Gi 2
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: 3
    volumePath: "[datastore1] volumes/myDisk" 4
    fsType: ext4 5
```

1. ボリュームの名前です。これを使用して、PVC (永続ボリューム要求) で、または Pod から、ボリュームを識別する必要があります。
2. このボリュームに割り当てられるストレージの量。
3. これは使用されるボリュームタイプを定義します (この例では、**vsphereVolume** プラグイン)。**vsphereVolume** ラベルは、vSphere VMDK ボリュームを Pod にマウントするために使用されます。ボリュームがアンマウントされてもボリュームのコンテンツは保持されます。このボリュームタイプは、VMFS データストアと VSAN データストアの両方をサポートします。

4. VMDK ボリュームが存在し、ボリューム定義内に括弧 ([]) を含める必要があります。

5. マウントするファイルシステムタイプ (**ext4**、**xfs**、その他のファイルシステムなど)。

重要

ボリュームをフォーマットしてプロビジョニングした後に **fsType** パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

永続ボリュームを作成します。

1. 定義を **vsphere-pv.yaml** などのファイルに保存し、PV を作成します。

```
$ oc create -f vsphere-pv.yaml
  persistentvolume "pv0001" created
```

2. PV が作成されたことを確認します。

```
$ oc get pv
NAME    LABELS   CAPACITY   ACCESSMODES   STATUS      CLAIM    REASON
AGE
pv0001  <none>   2Gi        RWO           Available
2s
```

これで、PVC (永続ボリューム要求) を使用してストレージを要求し、永続ボリュームを活用できるようになります。

重要

PVC (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

23.13.2.2. VMware vSphere ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナーに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない vSphere ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用できるようになります。

23.14. ローカルボリュームを使用した永続ストレージ

23.14.1. 概要

OpenShift Container Platform クラスターでは、ローカルボリュームを使用して永続ストレージをプロビジョニングできます。ローカル永続ボリュームを使用すると、標準の PVC インターフェースからディスク、パーティション、ディレクトリーなどのローカルストレージデバイスにアクセスできます。

ローカルボリュームは、Pod をノードに手動でスケジュールせずに使用できます。ボリュームのノード制約がシステムによって認識されるためです。ただし、ローカルボリュームは、依然として基礎となるノードの可用性に依存しており、すべてのアプリケーションに適しているわけではありません。

注記

ローカルボリュームはアルファ機能であり、OpenShift Container Platform の今後のリリースで変更される場合があります。既知の問題と回避策については、「Feature Status(Local Volume)」セクションを参照してください。

警告

ローカルボリュームは、静的に作成された永続ボリュームとしてのみ使用できます。

23.14.2. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントするには、ストレージが基礎となるインフラストラクチャーに存在している必要があります。**PersistentVolume** API を使用する前に、OpenShift Container Platform がローカルボリューム用に設定されていることを確認してください。

23.14.3. ローカルの Persistent Volume Claim (永続ボリューム要求) の作成

Persistent Volume Claim (永続ボリューム要求) をオブジェクト定義に定義します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: example-local-claim
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi 1
  storageClassName: local-storage 2
```

1. ストレージボリュームの必要なサイズ。
2. ローカル PV で使用されるストレージクラスの名前。

23.14.4. 機能のステータス

機能すること:

- ノードアフィニティーがあるディレクトリーを指定して PV を作成する。

- 前述の PV にバインドされている PVC を使用する Pod は常に該当ノードにスケジューリングされる。
- ローカルディレクトリーを検出し、PV の作成、クリーンアップ、および削除を行う外部の静的プロビジョナーデーモンセット。

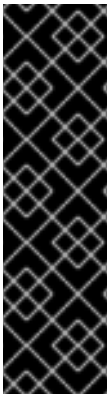
機能しないこと:

- 単一 Pod に複数のローカル PVC を持たせる。
- PVC バインディングでは Pod のスケジューリング要件を考慮しないため、最適でないか、または適切でない決定がなされる可能性がある。
  - 回避策
    - ローカルボリュームを必要とする Pod を最初に実行します。
    - それらの Pod に高い優先順位を設定します。
    - 動作が中断している Pod への PVC のバインドを解除する回避策となるコントローラーを実行します。
- 外部プロビジョナーの起動後にマウントを追加した場合、外部プロビジョナーはマウントの正確な容量を検出できなくなる。
  - 回避策
    - 新規のマウントポイントを追加する前に、デーモンセットを停止し、新規マウントポイントを追加してから deamonset を起動します。
- 同じ PV を使用する複数の Pod で別々の **`fsgroup`** を指定すると、**`fsgroup`** の競合が発生する。

23.15. CONTAINER STORAGE INTERFACE (CSI) を使用した永続ストレージ

23.15.1. 概要

Container Storage Interface (CSI) により、OpenShift Container Platform は CSI インターフェースを永続ストレージとして実装するストレージバックエンドからストレージを使用できます。



重要

現時点で CLI ボリュームはテクノロジープレビュー機能で、実稼働のワークロードを対象としていません。CSI ボリュームは、今後の OpenShift Container Platform のリリースで変更される可能性があります。また、テクノロジープレビュー機能は、Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能のリリースに先駆けてご提供することができ、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

詳細は、「テクノロジープレビュー機能のサポート範囲」を参照してください。

注記

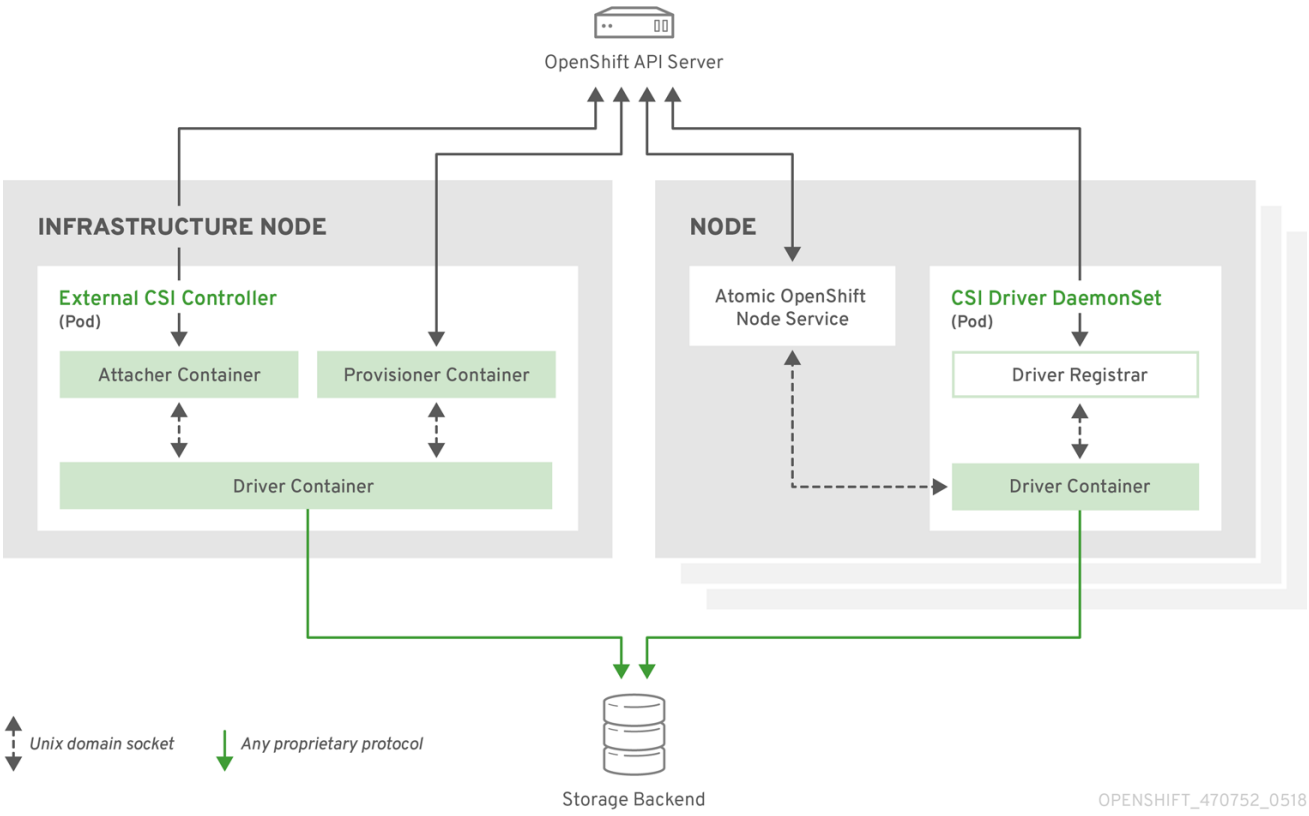
OpenShift Container Platform には、CSI ドライバーが含まれていません。コミュニティーまたはストレージベンダー が提供する CSI ドライバーを使用することを推奨します。

OpenShift Container Platform 3.10 は、CSI 仕様 のバージョン 0.2.0 をサポートします。

23.15.2. アーキテクチャー

CSI ドライバーは通常、コンテナーイメージとして提供されます。これらのコンテナーは、実行先の OpenShift Container Platform を認識しません。OpenShift Container Platform でサポートされる CSI 互換のストレージを使用するには、クラスター管理者は、OpenShift Container Platform とストレージドライバーの橋渡しとして機能する、コンポーネントを複数デプロイする必要があります。

以下の図では、OpenShift Container Platform クラスターの Pod で実行されるコンポーネントの俯瞰図を示しています。



異なるストレージバックエンドに対して複数の CSI ドライバーを実行できます。各ドライバーには、独自の外部コントローラーのデプロイメントおよびドライバーと CSI レジストラーを含む DaemonSet が必要です。

23.15.2.1. 外部の CSI コントローラー

外部の CSI コントローラーは、2 つのコンテナーを含む 1 つまたは複数の Pod を配置するデプロイメントです。

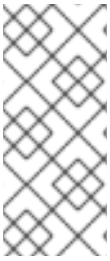
- OpenShift Container Platform からの **`attach`** および **`detach`** の呼び出しを適切な CSI ドライバーへの **`ControllerPublish`** および **`ControllerUnpublish`** 呼び出しに変換する外部の CSI アタッチャーコンテナー。

- OpenShift Container Platform からの **provision** および **delete** 呼び出しを適切な CSI ドライバーへの **CreateVolume** および **DeleteVolume** 呼び出しに変換する外部の CSI プロビジョナーコンテナー。
- CSI ドライバーコンテナー

CSI アタッチャーおよび CSI プロビジョナーコンテナーは、Unix Domain Socket を使用して、CSI ドライバーコンテナーと対話し、CSI のコミュニケーションが Pod 外に出ないようにします。CSI ドライバーは、Pod 外からはアクセスできません。

注記

attach、**detach**、**provision** および **delete** 操作では通常、CSI ドライバーがストレージバックエンドに対する認証情報を使用する必要があります。CSI コントローラー Pod をインフラストラクチャーノードで実行し、コンピュートノードで致命的なセキュリティー違反が発生した場合でさえも、認証情報がユーザープロセスに漏洩されないようにします。



注記

外部のアタッチャーは、サードパーティーの割り当て/割り当て解除操作をサポートしない CSI ドライバーに対しても実行する必要があります。外部アタッチャーは、CSI ドライバーに対して **ControllerPublish** または **ControllerUnpublish** の操作を実行しませんが、依然として、必要とされる OpenShift Container Platform 割り当て API を実装するために実行する必要があります。

23.15.2.2. CSI ドライバーの DaemonSet

最後に CSI ドライバーの DaemonSet は、OpenShift Container Platform が CSI ドライバーで提供されるストレージをノードにマウントして、永続ボリューム (PV) としてユーザーワークロード (Pod) で使用できるように、全ノードで Pod を実行します。CSI ドライバーがインストールされた Pod には、以下のコンテナーが含まれます。

- ノード上で実行中の **openshift-node** サービスに、CSI ドライバーを登録する CSI ドライバーレジストラー。このノードで実行中の **openshift-node** プロセスは、ノードで利用可能な Unix Domain Socket を使用して CSI ドライバーと直接、接続します。
- CSI ドライバー

ノードにデプロイされた CSI ドライバーには、ストレージバックエンドへの認証情報をできる限り少なく指定する必要があります。OpenShift Container Platform は、**NodePublish**/**NodeUnpublish** および **NodeStage**/**NodeUnstage** (実装されている場合) などの CSI 呼び出しのノードプラグインセットのみを使用します。

23.15.3. デプロイメント例

OpenShift Container Platform は CSI ドライバーがインストールされた状態で提供されないので、この例では、OpenShift Container Platform に OpenStack Cinder 向けのコミュニティードライバーをデプロイする方法が示されています。

1. CSI コンポーネントの実行先のプロジェクトと、このコンポーネントを実行するサービスアカウントを新たに作成します。明示的なノードセレクターを使用して、マスターノード上でも CSI ドライバーが設定された Daemonset を実行します。

```
# oc adm new-project csi --node-selector=""
```

```
Now using project "csi" on server "https://example.com:8443".

# oc create serviceaccount cinder-csi
serviceaccount "cinder-csi" created

# oc adm policy add-scc-to-user privileged
system:serviceaccount:csi:cinder-csi
scc "privileged" added to: ["system:serviceaccount:csi:cinder-csi"]
```

2. この YAML ファイルを適用して、外部の CSI アタッチャーとプロビジョナーを含むデプロイメントおよび CSI ドライバーを含む DaemonSet を作成します。

```
# This YAML file contains all API objects that are necessary to run
Cinder CSI
# driver.
#
# In production, this needs to be in separate files, e.g. service
account and
# role and role binding needs to be created once.
#
# It server as an example how to use external attacher and external
provisioner
# images shipped with {product-title} with a community CSI driver.

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cinder-csi-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["create", "delete", "get", "list", "watch", "update",
"patch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["volumeattachments"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]

---

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
```

```
metadata:
  name: cinder-csi-role
subjects:
  - kind: ServiceAccount
    name: cinder-csi
    namespace: csi
roleRef:
  kind: ClusterRole
  name: cinder-csi-role
  apiGroup: rbac.authorization.k8s.io

---
apiVersion: v1
data:
  cloud.conf:
W0dsb2JhbF0KYXV0aC11cmwgPSBodHRwczovL2V4YW1wbGUuY29tOjEzMDAwL3YyLjAv
CnVzZXJuYW1lID0gYWxhZGRpbgpwYXNzd29yZCA9IG9wZW5zZXNhbWUKdGVuYW50LWlk
ID0gZTBmYTg1YjZhMDY0NDM5NTlkMmQzYjQ5NzE3NGJlZDYKcmVnaW9uID0gcmVnaW9u
T25lCg== 1
kind: Secret
metadata:
  creationTimestamp: null
  name: cloudconfig
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: cinder-csi-controller
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cinder-csi-controllers
  template:
    metadata:
      labels:
        app: cinder-csi-controllers
    spec:
      serviceAccount: cinder-csi
      containers:
        - name: csi-attacher
          image: registry.access.redhat.com/openshift3/csi-
attacher:v3.10
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
            - "--leader-election"
            - "--leader-election-namespace=$(MY_NAMESPACE)"
            - "--leader-election-identity=$(MY_NAME)"
          env:
            - name: MY_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: MY_NAMESPACE
              valueFrom:
```

```
                fieldRef:
                  fieldPath: metadata.namespace
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: csi-provisioner
          image: registry.access.redhat.com/openshift3/csi-
provisioner:v3.10
          args:
            - "--v=5"
            - "--provisioner=csi-cinderplugin"
            - "--csi-address=$(ADDRESS)"
          env:
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: cinder-driver
          image: quay.io/jsafrane/cinder-csi-plugin
          command: [ "/bin/cinder-csi-plugin" ]
          args:
            - "--nodeid=$(NODEID)"
            - "--endpoint=unix://$(ADDRESS)"
            - "--cloud-config=/etc/cloudconfig/cloud.conf"
          env:
            - name: NODEID
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
            - name: cloudconfig
              mountPath: /etc/cloudconfig
      volumes:
        - name: socket-dir
          emptyDir:
        - name: cloudconfig
          secret:
            secretName: cloudconfig

---

kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: cinder-csi-ds
spec:
  selector:
    matchLabels:
      app: cinder-csi-driver
```

```
  template:
    metadata:
      labels:
        app: cinder-csi-driver
    spec:
      nodeSelector:
          role: node
      serviceAccount: cinder-csi
      containers:
        - name: csi-driver-registrar
          image: registry.access.redhat.com/openshift3/csi-driver-
registrar:v3.10
          securityContext:
            privileged: true
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
          env:
            - name: ADDRESS
              value: /csi/csi.sock
            - name: KUBE_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: cinder-driver
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
            allowPrivilegeEscalation: true
          image: quay.io/jsafrane/cinder-csi-plugin
          command: [ "/bin/cinder-csi-plugin" ]
          args:
            - "--nodeid=$(NODEID)"
            - "--endpoint=unix://$(ADDRESS)"
            - "--cloud-config=/etc/cloudconfig/cloud.conf"
          env:
            - name: NODEID
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
            - name: cloudconfig
              mountPath: /etc/cloudconfig
            - name: mountpoint-dir
              mountPath:
/var/lib/origin/openshift.local.volumes/pods/
              mountPropagation: "Bidirectional"
            - name: cloud-metadata
```

```
            mountPath: /var/lib/cloud/data/
          - name: dev
            mountPath: /dev
      volumes:
        - name: cloud-metadata
          hostPath:
            path: /var/lib/cloud/data/
        - name: socket-dir
          hostPath:
            path: /var/lib/kubelet/plugins/csi-cinderplugin
            type: DirectoryOrCreate
        - name: mountpoint-dir
          hostPath:
            path: /var/lib/origin/openshift.local.volumes/pods/
            type: Directory
        - name: cloudconfig
          secret:
            secretName: cloudconfig
        - name: dev
          hostPath:
            path: /dev
```

1 「OpenStack 設定」に記載されているように、OpenStack デプロイメントの **cloud.conf** に置き換えます。たとえば、シークレットは、**oc create secret generic cloudconfig --from-file cloud.conf --dry-run -o yaml** を使用して生成できます。

23.15.4. 動的プロビジョニング

永続ストレージの動的プロビジョニングは、CSI ドライバーおよび基盤のストレージバックエンドの機能により異なります。CSI ドライバーのプロバイダーは、OpenShift Container Platform で StorageClass の作成方法および、設定で利用可能なパラメーターについて文書を作成する必要があります。

OpenStack Cinder の例にあるように、この StorageClass をデプロイして、動的プロビジョニングを有効化できます。以下の例では、新規のデフォルトストレージクラスを作成して、特別なストレージクラスを必要としない PVC がすべて、インストールした CSI ドライバーによりプロビジョニングされるようにします。

```
# oc create -f - << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cinder
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-cinderplugin
parameters:
EOF
```

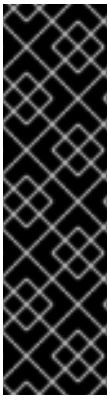
23.15.5. 使用法

CSI ドライバーがデプロイされ、動的プロビジョニング用に StorageClass が作成されたら、OpenShift Container Platform は CSI を使用する準備が整います。以下の例では、変更を加えずに、デフォルトの MySQL テンプレートをインストールします。

```
# oc new-app oc new-app mysql-persistent
--> Deploying template "openshift/mysql-persistent" to project default
...

# oc get pvc
[root@host-172-16-120-9 ~]# oc get pvc
NAME              STATUS    VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mysql             Bound     kubernetes-dynamic-pv-3271ffcb4e1811e8   1Gi
RWO            cinder         3s
```

23.16. OPENSTACK MANILA を使用した永続ストレージ

23.16.1. 概要



重要

OpenStack Manila を使用した永続ボリューム (PV) のプロビジョニングはテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

OpenShift Container Platform は、OpenStack Manila 共有ファイルシステムサービスを使用して PV をプロビジョニングできます。

OpenStack Manila サービスが正しく設定され、OpenShift Container Platform クラスターからアクセスできることが前提です。プロビジョニングが可能なのは、NFS 共有タイプのみです。

PV、Persistent Volume Claim (永続ボリューム要求、PVC)、動的プロビジョニング および RBAC 認証について理解しておくことを推奨します。

23.16.2. インストールと設定

この機能は、外部プロビジョナーにより提供され、OpenShift Container Platform クラスターに外部プロビジョナーをインストールし、設定する必要があります。

23.16.2.1. 外部プロビジョナーの起動

外部プロビジョナーサービスは、コンテナーイメージとして配布され、OpenShift Container Platform クラスターで通常通り実行できます。

API オブジェクトを管理しているコンテナーを許可するには、以下のようにして、必要なロールベースアクセス制御 (RBAC) ルールを管理者が設定する必要があります。

1. **ServiceAccount** を作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: manila-provisioner-runner
```

2. **ClusterRole** を作成します。

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: manila-provisioner-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
```

3. **ClusterRoleBinding** を使用して、以下のようにルールをバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: manila-provisioner
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: manila-provisioner-role
subjects:
- kind: ServiceAccount
  name: manila-provisioner-runner
  namespace: default
```

4. 新しい **StorageClass** を作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "manila-share"
provisioner: "externalstorage.k8s.io/manila"
parameters:
  type: "default" (1)
  zones: "nova" (2)
```

(1) プロビジョナーがボリューム用に作成する Manila 共有タイプ

2 ボリュームが作成される可能性のある Manila アベイラビリティーゾーンセット

環境変数を使用してプロビジョナーによる Manila サービスへの接続、認証、承認ができるように設定します。以下のリストから、お使いのインストールに合った適切な環境変数の組み合わせを選択します。

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_NAME
OS_TENANT_NAME
```

```
OS_USERID
OS_PASSWORD
OS_AUTH_URL
OS_TENANT_ID
```

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_ID
OS_TENANT_NAME
```

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_ID
OS_TENANT_ID
```

プロビジョナーに変数を渡すには、**Secret** を使用します。以下の例は、1 つ目の変数の組み合わせ向けに設定した **Secret** です。

```
apiVersion: v1
kind: Secret
metadata:
  name: manila-provisioner-env
type: Opaque
data:
  os_username: <base64 encoded Manila username>
  os_password: <base64 encoded password>
  os_auth_url: <base64 encoded OpenStack Keystone URL>
  os_domain_name: <base64 encoded Manila service Domain>
  os_tenant_name: <base64 encoded Manila service Tenant/Project name>
```

注記

より新しいバージョンの OpenStack では、「テナント」ではなく「プロジェクト」を使用しますが、プロビジョナーが使用する環境変数は、名前に **TENANT** を使用する必要があります。

最後の手順では、デプロイメントなどを使用して、プロビジョナー自体を起動します。

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: manila-provisioner
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: manila-provisioner
    spec:
      serviceAccountName: manila-provisioner-runner
      containers:
        - image: "registry.access.redhat.com:8888/openshift3/manila-
provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          name: manila-provisioner
          env:
            - name: "OS_USERNAME"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_username
            - name: "OS_PASSWORD"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_password
            - name: "OS_AUTH_URL"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_auth_url
            - name: "OS_DOMAIN_NAME"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_domain_name
            - name: "OS_TENANT_NAME"
              valueFrom:
                secretKeyRef:
                  name: manila-provisioner-env
                  key: os_tenant_name
```

23.16.3. 使用法

プロビジョナーの実行後に、PVC と適切な StorageClass を使用して PV をプロビジョニングすることができます。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: manila-nfs-pvc
```

```
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2G
  storageClassName: manila-share
```

次に、**PersistentVolumeClaim** は、新たにプロビジョニングされた Manila 共有でサポートされる **PersistentVolume** にバインドされます。**PersistentVolumeClaim** の後に **PersistentVolume** を削除したら、プロビジョナーは Manila 共有を削除して、エクスポートを中止します。

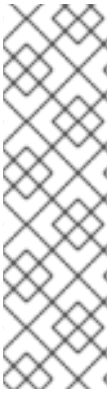
23.17. 動的プロビジョニングとストレージクラスの作成

23.17.1. 概要

StorageClass リソースオブジェクトは、要求可能なストレージを記述し、分類するほか、**動的にプロビジョニングされるストレージ**のパラメーターを要求に応じて渡すための手段を提供します。**StorageClass** オブジェクトは、さまざまなレベルのストレージとストレージへのアクセスを制御するための管理メカニズムとしても機能します。クラスター管理者 (**cluster-admin**) またはストレージ管理者 (**storage-admin**) は、ユーザーが基礎となるストレージボリュームソースに関する詳しい知識がなくても要求できる **StorageClass** オブジェクトを定義し、作成します。

OpenShift Container Platform の永続ボリュームフレームワークは、この機能を有効にし、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。このフレームワークにより、ユーザーは基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようになります。

OpenShift Container Platform では、数多くのストレージタイプを永続ボリュームとして使用できます。これらはすべて管理者によって静的にプロビジョニングされますが、一部のストレージタイプは組み込みプロバイダーとプラグイン API を使用して動的に作成できます。



注記

動的プロビジョニングを有効にするには、**openshift_master_dynamic_provisioning_enabled** 変数を Ansible インベントリーファイルの **[OSEv3:vars]** セクションに追加し、その値を **True** に設定します。

```
[OSEv3:vars]

openshift_master_dynamic_provisioning_enabled=True
```

23.17.2. 動的にプロビジョニングされる使用可能なプラグイン

OpenShift Container Platform は、以下の**プロビジョナープラグイン**を提供します。これらには、クラスターの設定済みプロバイダーの API を使用して新規ストレージリソースを作成する動的プロビジョニング用の一般的な実装が含まれます。

ストレージタイプ	プロビジョナープラグインの名前	必要な設定	備考
OpenStack Cinder	**kubernetes.io/cinder**	OpenStack の設定	
AWS Elastic Block Store (EBS)	**kubernetes.io/aws-ebs**	AWS の設定	複数クラスターを別々のゾーンで使用する際の動的プロビジョニングの場合、各ノードに **Key=kubernetes.io/cluster/xxxx,Value=clusterid** のタグを付けます。ここで、**xxxx** と **clusterid** はクラスターごとに固有になります。3.6 より前のバージョンでは、これには **Key=KubernetesCluster,Value=clusterid** が使用されていました。
GCE Persistent Disk (gcePD)	**kubernetes.io/gce-pd**	GCE の設定	マルチゾーン設定では、PV が現行クラスターのノードが存在しないゾーンで作成されないようにするため、Openshift クラスターを GCE プロジェクトごとに実行することが推奨されます。
GlusterFS	**kubernetes.io/glusterfs**	GlusterFS の設定	
Ceph RBD	**kubernetes.io/rbd**	Ceph RBD の設定	
NetApp の Trident	**netapp.io/trident**	Trident の設定	NetApp ONTAP、SolidFire、および E-Series ストレージ向けのストレージオーケストレーター。
VMWare vSphere	**kubernetes.io/vsphere-volume**	vSphere と Kubernetes の使用開始	
Azure Disk	**kubernetes.io/azure-disk**	Azure の設定	

重要

選択したプロビジョナープラグインでは、関連するクラウド、ホスト、またはサードパーティープロバイダーを、関連するドキュメントに従って設定する必要もあります。

23.17.3. StorageClass の定義

現時点で、**StorageClass** オブジェクトはグローバルなスコープ付きオブジェクトであり、**cluster-admin** または **storage-admin** ユーザーによって作成される必要があります。

注記

GCE と AWS の場合、デフォルトの StorageClass が OpenShift Container Platform のインストール時に作成されます。デフォルトの StorageClass を変更または削除することができます。

現時点で 6 つのプラグインがサポートされています。以下のセクションでは、**StorageClass** の基本オブジェクトの定義とサポートされている各プラグインタイプの具体的な例について説明します。

23.17.3.1. 基本 StorageClass オブジェクトの定義

StorageClass 基本オブジェクトの定義

```
kind: StorageClass (1)
apiVersion: storage.k8s.io/v1 (2)
metadata:
  name: foo (3)
  annotations: (4)
    ...
provisioner: kubernetes.io/plug-in-type (5)
parameters: (6)
  param1: value
  ...
  paramN: value
```

(1) (必須) API オブジェクトタイプ。

(2) (必須) 現在の apiVersion。

(3) (必須) StorageClass の名前。

(4) (オプション) StorageClass のアノテーション

(5) (必須) このストレージクラスに関連付けられているプロビジョナーのタイプ。

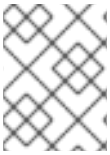
(6) (オプション) 特定のプロビジョナーに必要なパラメーター。これはプラグインによって異なります。

23.17.3.2. StorageClass のアノテーション

StorageClass をクラスター全体のデフォルトとして設定するには、以下を実行します。

```
    storageclass.kubernetes.io/is-default-class: "true"
```

これにより、特定のボリュームを指定しない Persistent Volume Claim (永続ボリューム要求、PVC) が**デフォルト** StorageClass によって自動的にプロビジョニングされるようになります。



注記

ベータアノテーションの **`storageclass.beta.kubernetes.io/is-default-class`** は以前として使用可能ですが、今後のリリースで削除される予定です。

StorageClass の記述を設定するには、以下のように指定します。

```
    kubernetes.io/description: My StorageClass Description
```

23.17.3.3. OpenStack Cinder オブジェクトの定義

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast  (1)
  availability: nova  (2)
  fsType: ext4  (3)
```

(1) Cinder で作成されるボリュームタイプ。デフォルトは空です。

(2) アベイラビリティーゾーン。指定しない場合、ボリュームは通常 OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。

(3) 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **`fsType`** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **`ext4`** です。

23.17.3.4. AWS ElasticBlockStore (EBS) オブジェクトの定義

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1  (1)
  zone: us-east-1d  (2)
  iopsPerGB: "10"  (3)
```

```
    encrypted: "true" 4
    kmsKeyId: keyvalue 5
    fsType: ext4 6
```

1. **io1**、**gp2**、**sc1**、**st1** から選択します。デフォルトは **gp2** です。有効な Amazon Resource Name (ARN) 値については、AWS のドキュメントを参照してください。
2. AWS ゾーン。ゾーンを指定しない場合、ボリュームは通常、OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。zone パラメーターと zones パラメーターを同時に使用することはできません。
3. **io1** ボリュームのみ。1 GiB あたり 1 秒あたりの I/O 処理数。AWS ボリュームプラグインは、この値と要求されたボリュームのサイズを乗算してボリュームの IOPS を算出します。値の上限は、AWS でサポートされる最大値である 20,000 IOPS です。詳細については、AWS のドキュメントを参照してください。
4. EBS ボリュームを暗号化するかどうかを示します。有効な値は **true** または **false** です。
5. オプション。ボリュームを暗号化する際に使用するキーの完全な ARN。値を指定しない場合でも **encypted** が **true** に設定されている場合は、AWS によってキーが生成されます。有効な ARN 値については、AWS のドキュメントを参照してください。
6. 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

23.17.3.5. GCE PersistentDisk (gcePD) オブジェクトの定義

gce-pd-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 1
  zone: us-central1-a 2
  zones: us-central1-a, us-central1-b, us-east1-b 3
  fsType: ext4 4
```

1. **pd-standard** または **pd-ssd** のいずれかを選択します。デフォルトは **pd-ssd** です。
2. GCE ゾーン。ゾーンを指定しない場合、ボリュームは通常、OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。zone パラメーターと zones パラメーターを同時に使用することはできません。
3. GCE ゾーンのコンマ区切りの一覧。ゾーンを指定しない場合、ボリュームは通常、OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。zone パラメーターと zones パラメーターを同時に使用することはできません。
4.

動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウ

23.17.3.6. GlusterFS オブジェクトの定義

glusterfs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://127.0.0.1:8081" 1
  restuser: "admin" 2
  secretName: "heketi-secret" 3
  secretNamespace: "default" 4
  gidMin: "40000" 5
  gidMax: "50000" 6
```

1. GlusterFS ボリュームをオンデマンドでプロビジョニングする heketi (Gluster 用のボリューム管理 REST サービス) URL。一般的な形式は **{http/https}://{IPaddress}:{Port}** です。GlusterFS 動的プロビジョナーの場合、これは必須パラメーターです。heketi サービスが OpenShift Container Platform で ルーティング可能なサービスとして公開されている場合には、解決可能な完全修飾ドメイン名 (FQDN) と heketi サービス URL が割り当てられます。
2. ボリュームを作成するためのアクセスを持つ heketi ユーザー。通常は「admin」です。
3. heketi との通信に使用するユーザーパスワードを含むシークレットの ID。オプションです。**secretNamespace** と **secretName** の両方を省略した場合、空のパスワードが使用されます。指定するシークレットは**"kubernetes.io/glusterfs"** タイプである必要があります。
4. 前述の **secretName** の namespace。オプションです。**secretNamespace** と **secretName** の両方を省略した場合、空のパスワードが使用されます。指定するシークレットは**"kubernetes.io/glusterfs"** タイプである必要があります。
5. オプション。この StorageClass のボリュームの GID 範囲の最小値です。
6. オプション。この StorageClass のボリュームの GID 範囲の最大値です。

注記

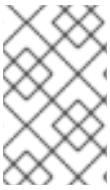
gidMin 値と **gidMax** 値を指定しない場合、デフォルトはそれぞれ 2000 と 2147483647 になります。動的にプロビジョニングされる各ボリュームには、この範囲 (**gidMin-gidMax**) の GID が割り当てられます。GID は、対応するボリュームが削除されるとプールから解放されます。GID プールは StorageClass ごとに設定されます。複数のストレージクラス間で GID 範囲が重複している場合、プロビジョナーによって、重複する GID が割り当てられる可能性があります。

heketi 認証を使用する場合は、管理キーを含むシークレットも存在している必要があります。

heketi-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  key: bXlwYXNzd29yZA== (1)
type: kubernetes.io/glusterfs
```

(1) base64 でエンコードされたパスワード。例: **echo -n "mypassword" | base64**



注記

PV が動的にプロビジョニングされると、GlusterFS プラグインによってエンドポイントと **gluster-dynamic-<claimname>** という名前のヘッドレスサービスが自動的に作成されます。PVC が削除されると、これらの動的リソースは自動的に削除されます。

23.17.3.7. Ceph RBD オブジェクトの定義

ceph-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 10.16.153.105:6789 (1)
  adminId: admin (2)
  adminSecretName: ceph-secret (3)
  adminSecretNamespace: kube-system (4)
  pool: kube (5)
  userId: kube (6)
  userSecretName: ceph-secret-user (7)
  fsType: ext4 (8)
```

(1) Ceph モニター (コンマ区切り)。必須です。

(2) Ceph クライアント ID。これにより、プール内にイメージを作成できます。デフォルトは「admin」です。

(3) **adminId** のシークレット名。必須です。設定するシークレットのタイプは「kubernetes.io/rbd」である必要があります。

(4) **adminSecret** の namespace。デフォルトは「default」です。

(5) Ceph RBD プール。デフォルトは「rbd」です。

(6) Ceph RBD イメージのマッピングに使用される Ceph クライアント ID。デフォルトは **adminId** と同じです。

7 Ceph RBD イメージをマッピングするために使用する **userId** 用の Ceph シークレットの名前。PVC と同じ namespace に存在する必要があります。必須です。

8 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

23.17.3.8. Trident オブジェクトの定義

trident.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gold
provisioner: netapp.io/trident 1
parameters: 2
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
```

Trident は、これらのパラメーターを Trident に登録されているさまざまなストレージプールの選択条件として使用します。Trident 自体は個別に設定されます。

1 Trident を OpenShift Container Platform にインストールする方法の詳細については、Trident のドキュメントを参照してください。

2 サポートされているパラメーターの詳細については、Trident のドキュメントのストレージ属性に関するセクションを参照してください。

23.17.3.9. VMWare vSphere オブジェクトの定義

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: kubernetes.io/vsphere-volume 1
parameters:
  diskformat: thin 2
```

1 VMWare vSphere を OpenShift Container Platform で使用する方法の詳細については、VMWare vSphere のドキュメントを参照してください。

2 **diskformat**: **thin**、**zeroedthick**、および **eagerzeroedthick**。詳細については、vSphere のドキュメントを参照してください。デフォルト: **thin**

23.17.3.10. Azure File のオブジェクト定義

Azure file の動的プロビジョニングを設定するには、以下を実行します。

1. ユーザーのプロジェクトにロールを作成します。

```
$ cat azf-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: system:controller:persistent-volume-binder
  namespace: <user's project name>
rules:
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "get", "delete"]
```

2. **persistent-volume-binder** サービスにバインドされるロールを **kube-system** プロジェクトに作成します。

```
$ cat azf-rolebind.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: system:controller:persistent-volume-binder
  namespace: <user's project>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: system:controller:persistent-volume-binder
subjects:
- kind: ServiceAccount
  name: persistent-volume-binder
namespace: kube-system
```

3. **admin** として、ユーザーのプロジェクトにサービスアカウントを追加します。

```
$ oc policy add-role-to-user admin system:serviceaccount:kube-
system:persistent-volume-binder -n <user's project>
```

4. Azure File 向けにストレージクラスを作成します。

```
$ cat azfsc.yaml | oc create -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azfsc
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
```

ユーザーは、このストレージクラスを使用する PVC を作成できるようになりました。

23.17.3.11. Azure Disk オブジェクト定義

azure-advanced-disk-storageclass.yaml

azure-advanced-disk-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/azure-disk
parameters:
  storageAccount: azure_storage_account_name  (1)
  storageaccounttype: Standard_LRS  (2)
  kind: Dedicated  (3)
```

(1) Azure ストレージアカウントの名前。これはクラスターと同じリソースグループに存在している必要があります。ストレージアカウントを指定した場合、**location** は無視されます。ストレージアカウントを指定しない場合、新しいストレージアカウントがクラスターと同じリソースグループに作成されます。**storageAccount** を指定する場合は、**kind** の値は **Dedicated** でなければなりません。

(2) Azure ストレージアカウントの SKU の層。デフォルトは空です。**注:** プレミアム VM は **Standard_LRS** ディスクと **Premium_LRS** ディスクの両方を割り当て、標準 VM は **Standard_LRS** ディスクのみを、マネージド VM はマネージドディスクのみを、アンマネージド VM はアンマネージドディスクのみを割り当てることができます。

(3) 許容値は、**Shared** (デフォルト)、**Dedicated** および **Managed** です。

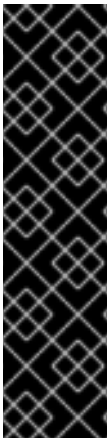
a. **kind** が **Shared** に設定されている場合は、Azure は、クラスターと同じリソースグループにあるいくつかの共有ストレージアカウントに、アンマネージドディスクをすべて作成します。

b. **kind** が **Managed** に設定されている場合は、Azure は新しいマネージドディスクを作成します。

c. **kind** が **Dedicated** に設定されており、**storageAccount** が指定されている場合には、Azure は、クラスターと同じリソースグループ内にある新規のアンマネージドディスク用に、指定のストレージアカウントを使用します。これを機能させるには、以下が前提となっています。

- 指定のストレージアカウントが、同じリージョン内にあること。
- Azure Cloud Provider にストレージアカウントへの書き込み権限があること。

d. **kind** が **Dedicated** に設定されており、**storageAccount** が指定されていない場合には、Azure はクラスターと同じリソースグループ内の新規のアンマネージドディスク用に、新しい専用のストレージアカウントを作成します。



重要

Azure StorageClass は OpenShift Container Platform バージョン 3.7 で改訂され、以前のバージョンからアップグレードした場合には、以下のいずれかを行ってください。

- **kind: dedicated** のプロパティーを指定して、アップグレード前に作成した Azure StorageClass を使用し続ける。または、
- **azure.conf** ファイルにロケーションのパラメーター (例: **"location": "southcentralus",**) を追加して、デフォルトのプロパティー **kind: shared** を使用します。こうすることで、新しいストレージアカウントを作成し、今後使用できるようになります。

23.17.4. デフォルト StorageClass の変更

GCE と AWS を使用している場合にデフォルトの StorageClass を変更するには、以下のプロセスを使用します。

1. StorageClass の一覧を表示します。

```
$ oc get storageclass

NAME                 TYPE
gp2 (default)        kubernetes.io/aws-ebs 1
standard             kubernetes.io/gce-pd
```

1 **(default)** はデフォルトの StorageClass を示します。

2. デフォルトの StorageClass のアノテーション **storageclass.kubernetes.io/is-default-class** の値を **false** に変更します。

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": \
    {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. アノテーション **storageclass.kubernetes.io/is-default-class=true** を追加するか、このアノテーションに変更して別の StorageClass をデフォルトにします。

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": \
    {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. 変更内容を確認します。

```
$ oc get storageclass

NAME                 TYPE
gp2                  kubernetes.io/aws-ebs
standard (default)   kubernetes.io/gce-pd
```

23.17.5. 追加情報とサンプル

- 動的プロビジョニング用の StorageClass の例と使用法
- 動的プロビジョニングのない StorageClass の例と使用法

23.18. ボリュームのセキュリティー

23.18.1. 概要

このトピックでは、ボリュームのセキュリティーに関連する Pod のセキュリティーについての一般的なガイドを提供します。Pod レベルのセキュリティーに関する全般的な情報については、SCC (Security Context Constraints) の管理と SCC (Security Context Constraints) の概念に関するトピックを参照してください。OpenShift Container Platform の永続ボリューム (PV) フレームワークに関する全般的な情報については、永続ストレージの概念に関するトピックを参照してください。

永続ストレージにアクセスするには、クラスター管理者/ストレージ管理者とエンド開発者間の調整が必要です。クラスター管理者は、基礎となる物理ストレージを抽象化する PV を作成します。開発者は、容量などの一致条件に基づいて Pod と (必要に応じて) PV にバインドされる PVC を作成します。

同じプロジェクト内の複数の Persistent Volume Claim (永続ボリューム要求、PVC) を同じ PV にバインドできます。ただし、PVC を PV にバインドすると、最初の要求のプロジェクトの外部にある要求をその PV にバインドできなくなります。基礎となるストレージに複数のプロジェクトからアクセスする必要がある場合は、プロジェクトごとに独自の PV が必要です。これらの PV は同じ物理ストレージを参照できます。これに関連して、バインドされる PV はプロジェクトに結び付けられます。PV と PVC の詳細な例については、「WordPress and MySQL using NFS」を参照してください。

クラスター管理者が PV への Pod アクセスを許可するには、以下を行う必要があります。

- 実際のストレージに割り当てられるグループ ID またはユーザー ID を把握しておく
- SELinux に関する考慮事項を理解しておく
- これらの ID が Pod の要件に一致するプロジェクトまたは SCC に対して定義される正式な ID の範囲で許可されることを確認する

グループ ID、ユーザー ID および SELinux の値は、Pod 定義の **`SecurityContext`** セクションで定義します。グループ ID は、Pod に対してグローバルであり、Pod で定義されるすべてのコンテナーに適用されます。ユーザー ID は同様にグローバルにすることも、コンテナーごとに固有にすることもできます。ボリュームへのアクセスは以下の 4 つのセクションで制御します。

- **`supplementalGroups`**
- **`fsGroup`**
- **`runAsUser`**
- **`seLinuxOptions`**

23.18.2. SCC、デフォルト値および許可される範囲

SCC により、Pod にデフォルトのユーザー ID、**`fsGroup`** ID、補助グループ ID、および SELinux ラベルが割り当てられるかが左右されます。また、SCC で、許容可能な ID の範囲に対して、Pod 定義 (またはイメージ) で指定される ID を検証するかも決まります。検証が必要な場合で検証が失敗すると、Pod も失敗します。

SCC は、**`runAsUser`**、**`supplementalGroups`**、**`fsGroup`** などのストラテジーを定義します。これらのストラテジーは Pod に権限があるかどうかを判断するのに役立ちます。**RunAsAny** に設定されるストラテジーの値は、Pod が対象のストラテジーの内容を何でも実行できると実質的に宣言します。このストラテジーに対する承認は省略され、このストラテジーをもとに生成される OpenShift Container

Platform のデフォルト値はありません。したがって、生成されるコンテナーの ID と SELinux ラベルは、OpenShift Container Platform ポリシーではなく、コンテナーのデフォルトに基づいて割り当てられます。

以下に **RunAsAny** の簡単な概要を示します。

- Pod 定義 (またはイメージ) に定義されるすべての ID が許可されます。
- Pod 定義 (およびイメージ) に ID がない場合は、コンテナーによって ID が割り当てられます。Docker の場合、この ID は **root** (0) です。
- SELinux ラベルは定義されないため、Docker によって一意のラベルが割り当てられます。

このような理由により、ID 関連のストラテジーについて **RunAsAny** が設定された SCC は、通常の開発者がアクセスできないように保護する必要があります。一方、**MustRunAs** または **MustRunAsRange** に設定された SCC ストラテジーは、(ID 関連のストラテジーについての) ID 検証をトリガーします。その結果、Pod 定義またはイメージに値が直接指定されていない場合は、OpenShift Container Platform によってデフォルト値がコンテナーに割り当てられます。

注意

RunAsAny **`FSGroup`** ストラテジーが設定された SCC へのアクセスを許可すると、ユーザーがブロックデバイスにアクセスするのを防止することもできます。Pod では、ユーザーのブロックデバイスを引き継ぐために **`fsGroup`** を指定する必要があります。通常、これを行うには、SCC **`FSGroup`** ストラテジーを **MustRunAs** に設定します。ユーザーの Pod に **RunAsAny** **`FSGroup`** ストラテジーが設定された SCC が割り当てられている場合、ユーザーが **`fsGroup`** ストラテジーを各自で指定する必要があることに気づくまで、**permission denied** エラーが出される可能性があります。

SCC では、許可される ID (ユーザーまたはグループ) の範囲を定義できます。範囲チェックが必要な場合 (**MustRunAs** を使用する場合など) で、許容可能な範囲が SCC で定義されていない場合は、プロジェクトによって ID 範囲が決定されます。したがって、プロジェクトでは、許容可能な ID の範囲がサポートされます。ただし、SCC とは異なり、プロジェクトは **`runAsUser`** などのストラテジーを定義しません。

許容可能な範囲を設定すると、コンテナー ID の境界を定義するだけでなく、範囲の最小値を対象の ID のデフォルト値にできるので役に立ちます。たとえば、SCC ID ストラテジーの値が **MustRunAs** で、ID 範囲の最小値が **100**で、ID が Pod 定義に存在しない場合、100 がこの ID のデフォルト値になります。

Pod の受付プロセスの一環として、Pod で使用可能な SCC が (ほとんどの場合は優先順位の高い SCC から最も制限の厳しい SCC の順序で) 検査され、Pod の要求に最も一致する SCC が選択されます。SCC のストラテジータイプを **RunAsAny** に設定すると、制限が緩和されます。一方、**MustRunAs** タイプに設定すると、制限がより厳しくなります。これらすべてのストラテジーが評価されます。Pod に割り当てられた SCC を確認するには、**`oc get pod`** コマンドを使用します。

```
# oc get pod <pod_name> -o yaml
...
metadata:
  annotations:
    openshift.io/scc: nfs-scc 1
  name: nfs-pod1 2
  namespace: default 3
...
```

1. Pod が使用した SCC (この場合は、カスタム SCC) の名前。
2. Pod の名前。
3. プロジェクトの名前。OpenShift Container Platform では、「namespace」と「プロジェクト」は置き換え可能な用語として使用されています。詳細については、「Projects and Users」を参照してください。

Pod で一致した SCC をすぐに確認できない場合があります。そのため、上記のコマンドは、ライブコンテナーの UID、補助グループ、および SELinux のラベル変更を理解するのに非常に役立ちます。

ストラテジーが **RunAsAny** に設定された SCC では、そのストラテジーの特定の値を Pod 定義 (またはイメージ) に定義できます。これがユーザー ID (**runAsUser**) に適用される場合、コンテナーが root として実行されないように SCC へのアクセスを制限することが推奨されます。

Pod が **制限付き** SCC に一致することが多くあるため、これに伴うセキュリティーについて理解しておくことが重要です。**制限付き** SCC には以下の特性があります。

- **runAsUser** ストラテジーが **MustRunAsRange** に設定されているため、ユーザー ID が制限されます。これにより、ユーザー ID の検証が強制的に実行されます。
- 許容可能なユーザー ID の範囲が SCC で定義されないため (詳細については、**oc export scc restricted**を参照してください)、プロジェクトの **openshift.io/sa.scc.uid-range** 範囲が範囲チェックとデフォルト ID に使用されます (必要な場合)。
- デフォルトのユーザー ID は、ユーザー ID が Pod 定義で指定されておらず、一致する SCC の **runAsUser** が **MustRunAsRange** に設定されている場合に生成されます。
- SELinux ラベルが必要です (**seLinuxContext** が **MustRunAs** に設定されているため)。プロジェクトのデフォルトの MCS ラベルが使用されます。
- **FSGroup** ストラテジーが **MustRunAs** に設定され、指定される最初の範囲の最小値を値に使用するように指示されているため、**fsGroup** ID が単一の値に制限されます。
- 許容可能な **fsGroup** ID の範囲が SCC で定義されないため、プロジェクトの **openshift.io/sa.scc.supplemental-groups** の範囲 (またはユーザー ID に使用されるものと同じ範囲) の最小値が検証とデフォルト ID に使用されます (必要な場合)。
- デフォルトの **fsGroup** ID は、**fsGroup** ID が Pod で指定されておらず、一致する SCC の **FSGroup** が **MustRunAs** に設定されている場合に生成されます。
- 範囲チェックが必要でないため、任意の補助グループ ID が許可されます。これは **supplementalGroups** ストラテジーが **RunAsAny** に設定されているためです。
- 実行中の Pod に対してデフォルトの補助グループは生成されません。上記の 2 つのストラテジーについて **RunAsAny** が設定されているためです。したがって、グループが Pod 定義 (またはイメージ) に定義されていない場合は、コンテナーには補助グループが事前に定義されません。

以下に、SCC とプロジェクトの相互関係をまとめた **default** プロジェクトとカスタム SCC (**my-custom-scc**) の例を示します。

```
$ oc get project default -o yaml (1)
...
metadata:
```

```
  annotations: 2
    openshift.io/sa.scc.mcs: s0:c1,c0 3
    openshift.io/sa.scc.supplemental-groups: 1000000000/10000 4
    openshift.io/sa.scc.uid-range: 1000000000/10000 5

$ oc get scc my-custom-scc -o yaml
...
fsGroup:
  type: MustRunAs 6
  ranges:
  - min: 5000
    max: 6000
runAsUser:
  type: MustRunAsRange 7
  uidRangeMin: 1000100000
  uidRangeMax: 1000100999
seLinuxContext: 8
  type: MustRunAs
  SELinuxOptions: 9
    user: <selinux-user-name>
    role: ...
    type: ...
    level: ...
supplementalGroups:
  type: MustRunAs 10
  ranges:
  - min: 5000
    max: 6000
```

1. **default** はプロジェクトの名前です。
2. デフォルト値は、対応する SCC 戦略が **RunAsAny** でない場合にのみ生成されます。
3. Pod 定義または SCC で定義されていない場合の SELinux のデフォルト値です。
4. 許容可能なグループ ID の範囲です。ID の検証は、SCC ストラテジーが **RunAsAny** の場合にのみ実行されます。複数の範囲をコンマで区切って指定できます。サポートされている形式についてはこちらを参照してください。
5. **<4>** と同じですが、ユーザー ID に使用されます。また、ユーザー ID の単一の範囲のみがサポートされます。
6. 10 **MustRunAs** は、グループ ID の範囲チェックを実施して、コンテナーのグループのデフォルトを指定します。この SCC の定義に基づく場合、デフォルトは 5000 になります (最小の ID 値)。範囲が SCC から省略される場合、デフォルトは 1000000000 になります (プロジェクトから派生します)。サポートされている別のタイプ **RunAsAny** では、範囲チェックを実行しないため、いずれのグループ ID も許可され、デフォルトグループは生成されません。
7. **MustRunAsRange** は、ユーザー ID の範囲チェックを実施して、UID のデフォルトを指定します。この SCC の定義に基づく場合、デフォルトの UID は 1000100000 になります (最小値)。最小範囲と最大範囲が SCC から省略される場合、デフォルトのユーザー ID は 1000000000 になります (プロジェクトから派生します)。これ以外には **MustRunAsNonRoot** と **RunAsAny** のタイプがサポートされます。許可される ID の範囲は、ターゲットのストレージに必要ないずれのユーザー ID も含めるように定義することができます。

8 **MustRunAs** に設定した場合は、コンテナーは SCC の SELinux オプション、またはプロジェクトに定義される MCS のデフォルトを使用して作成されます。**RunAsAny** というタイプは、SELinux コンテキストが不要であることや、Pod に定義されていない場合はコンテナーに設定されないことを示します。

9 SELinux のユーザー名、ロール名、タイプ、およびラベルは、ここで定義できます。

2 つの形式が許可される範囲にサポートされています。

1. **M/N**。**M** は開始 ID で **N** はカウントです。したがって、範囲は **M** から **M+N-1** (これ自体を含む) までになります。

2. **M-N**。**M** は同じく開始 ID で **N** は終了 ID です。デフォルトのグループ ID が最初の範囲の開始 ID になります (このプロジェクトでは **1000000000** desu)。SCC で最小のグループ ID が定義されていない場合は、プロジェクトのデフォルトの ID が適用されます。

23.18.3. 補助グループ

注記

補助グループの操作にあたっては、事前に「SCC、デフォルト、許可される範囲」の説明をお読みください。

ヒント

永続ストレージへのアクセスを取得する場合、通常はグループ ID (補助グループまたは fsGroup) を使用する方がユーザー ID を使用するよりも適切です。

補助グループは Linux の正規グループです。Linux で実行されるプロセスには、UID、GID、および 1 つ以上の補助グループがあります。これらの属性はコンテナーのメインプロセスで設定されます。**supplementalGroups** ID は、通常は NFS や GlusterFS などの共有ストレージへのアクセスを制御する場合に使用されます。一方、fsGroup は Ceph RBD や iSCSI などのブロックストレージへのアクセスを制御する場合に使用されます。

OpenShift Container Platform の共有ストレージプラグインは、マウントの POSIX パーミッションとターゲットストレージのパーミッションが一致するようにボリュームをマウントします。たとえば、ターゲットストレージの所有者 ID が **1234** で、そのグループ ID が **5678** の場合、ホストノードのマウントとコンテナーのマウントはそれらの同じ ID を持ちます。したがって、ボリュームにアクセスするためにはコンテナーのメインプロセスがこれらの ID の一方または両方と一致する必要があります。

たとえば、以下の NFS エクスポートについて見てみましょう。

OpenShift Container Platform ノード側:

注記

showmount では、NFS サーバーの **rpcbind** および **rpc.mount** が使用するポートへのアクセスが必要です。

```
# showmount -e <nfs-server-ip-or-hostname>
Export list for f21-nfs.vm:
/opt/nfs  *
```

NFS サーバー側:

```
# cat /etc/exports
/opt/nfs *(rw,sync,root_squash)
...

# ls -lZ /opt/nfs -d
drwx------. 1000100001 5555 unconfined_u:object_r:usr_t:s0   /opt/nfs
```

/opt/nfs/ エクスポートには UID **1000100001** とグループ **5555** でアクセスすることができます。通常、コンテナーは root として実行しないようにする必要があります。そのため、この NFS の例では、UID **1000100001** として実行されないコンテナーやグループ **5555** のメンバーでないコンテナーは、NFS エクスポートにアクセスできません。

多くの場合、Pod と一致する SCC では特定のユーザー ID の指定は許可されません。そのため、Pod へのストレージアクセスを許可する場合には、補助グループを使用する方法はより柔軟な方法として使用できます。たとえば、前述のエクスポートへの NFS アクセスを許可する場合は、グループ **5555** を Pod 定義に定義できます。

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: ...
    volumeMounts:
    - name: nfs 1
      mountPath: /usr/share/... 2
  securityContext: 3
    supplementalGroups: [5555] 4
  volumes:
  - name: nfs 5
    nfs:
      server: <nfs_server_ip_or_host>
      path: /opt/nfs 6
```

1. ボリュームマウントの名前。**volumes** セクションの名前と一致する必要があります。
2. コンテナーで表示される NFS エクスポートのパス。
3. Pod のグローバルセキュリティーコンテキスト。Pod 内部のすべてのコンテナーに適用されます。コンテナーではそれぞれ独自の **securityContext** を定義することもできますが、グループ ID は Pod に対してグローバルであり、個々のコンテナーに対して定義することはできません。
4. 補助グループ (ID の配列) は 5555 に設定しています。これで、エクスポートへのグループアクセスが許可されます。
5. ボリュームの名前。**volumeMounts** セクションの名前と一致する必要があります。
6. NFS サーバー上の実際の NFS エクスポートのパス。

前述の Pod にあるすべてのコンテナー (一致する SCC またはプロジェクトでグループ **5555** を許可することを前提とします) は、グループ **5555** のメンバーとなり、コンテナーのユーザー ID に関係なくボリュームにアクセスできます。ただし、ここでの前提条件に留意してください。場合によっては、SCC

が許可されるグループ ID の範囲を定義されておらず、代わりにグループ ID の検証が必要になることがあります (**supplementalGroups** を **MustRunAs** に設定した結果など)。ただし、**制限付き** SCC の場合はこれと異なります。**プロジェクトがこの NFS エクスポートにアクセスするようにカスタマイズされていない限り、通常プロジェクトが 5555 というグループ ID を許可することはありません。したがって、このシナリオでは、前述の Pod の 5555 というグループ ID は SCC またはプロジェクトの許可されたグループ ID の範囲内にないために Pod は失敗します。**

補助グループとカスタム SCC

前述の例にあるような状況に対応するため、以下のようカスタム SCC を作成することができます。

- **最小と最大のグループ ID を定義する。**
- **ID の範囲チェックを実施する。**
- **グループ ID 5555 を許可する。**

多くの場合、定義済みの SCC を修正したり、定義済みプロジェクトで許可される ID の範囲を変更したりするよりも、新規の SCC を作成する方が適切です。

新規 SCC を作成するには、既存の SCC をエクスポートし、新規の SCC の要件を満たすように YAML ファイルをカスタマイズするのが最も簡単な方法です。たとえば、以下を実行します。

1. **制限付き SCC を新規 SCC のテンプレートとして使用します。**

```
$ oc export scc restricted > new-scc.yaml
```

2. **必要な仕様に合うように new-scc.yaml ファイルを編集します。**
3. **新規 SCC を作成します。**

```
$ oc create -f new-scc.yaml
```

注記

oc edit scc コマンドを使用して、インスタンス化された SCC を修正することができます。

以下の例は nfs-scc という名前の新規 SCC の一部です。

```
$ oc export scc nfs-scc

allowHostDirVolumePlugin: false (1)
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc (2)
priority: 9 (3)
...
supplementalGroups:
  type: MustRunAs (4)
  ranges:
```

```
  - min: 5000 5
    max: 6000
...
```

1. allow ブール値は制限付き SCC の場合と同じです。
2. 新規 SCC の名前。
3. 数値が大きいほど優先順位が高くなります。Nil の場合や省略した場合は優先順位が最も低くなります。優先順位が高い SCC は優先順位が低い SCC より前に並べ替えられるため、新規 Pod と一致する確率が高くなります。
4. supplementalGroups はストラテジーであり、MustRunAs に設定されています。つまり、グループ ID のチェックが必要になります。
5. 複数の範囲を使用することができます。ここで許可されるグループ ID の範囲は 5000 〜 5999 で、デフォルトの補助グループは 5000 です。

前述の Pod をこの新規 SCC に対して実行すると (当然ですが Pod が新規 SCC に一致することを前提とします)、Pod が開始されます。これは、Pod 定義で指定したグループ 5555 がカスタム SCC によって許可されるようになったためです。

23.18.4. fsGroup

注記

補助グループの操作にあたっては、事前に「SCC、デフォルト、許可される範囲」の説明をお読みください。

ヒント

永続ストレージへのアクセスを取得する場合、通常はグループ ID (補助グループまたは fsGroup) を使用する方がユーザー ID を使用するよりも適切です。

fsGroup は Pod の「ファイルシステムグループ」の ID を定義するもので、コンテナーの補助グループに追加されます。supplementalGroups の ID は共有ストレージに適用されますが、fsGroup の ID はブロックストレージに使用されます。

ブロックストレージ (Ceph RBD、iSCSI、各種クラウドストレージなど) は通常、ブロックストレージボリュームを直接に、または PVC を使用して要求した単一 Pod に専用のものとして設定されます。共有ストレージとは異なり、ブロックストレージは Pod によって引き継がれます。つまり、Pod 定義 (またはイメージ) で指定されたユーザー ID とグループ ID が実際の物理ブロックデバイスに適用されます。通常、ブロックストレージは共有されません。

以下の fsGroup の定義は Pod 定義の一部分です。

```
kind: Pod
...
spec:
  containers:
  - name: ...
```

```
  securityContext: 1
    fsGroup: 5555 2
  ...
```

1 supplementalGroups と同じように、fsGroup はコンテナーごとに定義するのではなく Pod に対してグローバルに定義する必要があります。

2 5555 はボリュームのグループパーミッションのグループ ID になり、ボリュームで作成されるすべての新規ファイルのグループ ID になります。

supplementalGroups と同様に、前述の Pod にあるすべてのコンテナー (一致する SCC またはプロジェクトでグループ 5555 を許可することを前提とします) は、グループ 5555 のメンバーとなり、コンテナーのユーザー ID に関係なくブロックボリュームにアクセスできます。Pod が制限付き SCC に一致し、その fsGroup ストラテジーが MustRunAs である場合、Pod の実行は失敗します。しかし、SCC の fsGroup ストラテジーを RunAsAny に設定した場合は、任意の fsGroup ID (5555 を含む) が許可されます。SCC の fsGroup ストラテジーを RunAsAny に設定して、fsGroup ID を指定しない場合は、ブロックストレージの「引き継ぎ」は行われず、Pod へのパーミッションが拒否される可能性があります。

fsGroups とカスタム SCC

前述の例にあるような状況に対応するため、以下のようにカスタム SCC を作成することができます。

- 最小と最大のグループ ID を定義する。
- ID の範囲チェックを実施する。
- グループ ID 5555 を許可する。

定義済みの SCC を修正したり、定義済みプロジェクトで許可される ID の範囲を変更したりするよりも、新規の SCC を作成する方が適切です。

以下の新規 SCC の定義の一部について見てみましょう。

```
# oc export scc new-scc
...
kind: SecurityContextConstraints
...
fsGroup:
  type: MustRunAs 1
  ranges: 2
  - max: 6000
    min: 5000 3
...
```

1 MustRunAs ではグループ ID の範囲チェックをトリガーします。一方、RunAsAny では範囲チェックは必要ありません。

2 許可されるグループ ID の範囲は 5000 ～ 5999 (これら自体を含む) です。複数の範囲がサポートされていますが、1 つしか使用していません。ここで許可されるグループ ID の範囲は 5000 ～ 5999 で、デフォルトの fsGroup は 5000 です。

3

最小値 (または範囲全体) を SCC から省略することができます。その場合、範囲のチェックとデフォルト値の生成はプロジェクトの `openshift.io/sa.scc.supplemental-groups` の範囲に

前述の Pod をこの新規 SCC に対して実行すると (当然ですが Pod が新しい SCC に一致することを前提とします)、Pod が開始されます。これは、Pod 定義で指定したグループ 5555 がカスタム SCC によって許可されているためです。また、Pod でブロックデバイスの「引き継ぎ」が行われます。そのため、ブロックストレージを Pod の外部のプロセスから表示する場合、そのグループ ID は実際には 5555 になります。

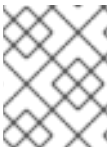
以下は、ブロックの所有権をサポートしているボリュームの例です。

- AWS Elastic Block Store
- OpenStack Cinder
- Ceph RBD
- GCE Persistent Disk
- iSCSI
- emptyDir
- gitRepo

注記

この一覧にはすべてが網羅されている訳ではありません。

23.18.5. ユーザー ID



注記

補助グループの操作にあたっては、事前に「SCC、デフォルト、許可される範囲」の説明をお読みください。

ヒント

永続ストレージへのアクセスを取得する場合、通常はグループ ID (補助グループまたは fsGroup) を使用する方がユーザー ID を使用するよりも適切です。

ユーザー ID はコンテナーイメージまたは Pod 定義で定義できます。Pod 定義では、1 つのユーザー ID をすべてのコンテナーに対してグローバルに定義するか、個々のコンテナーに固有のものとして定義するか、またはその両方として定義できます。以下の Pod 定義の一部ではユーザー ID を指定しています。

```
spec:
  containers:
  - name: ...
    securityContext:
      runAsUser: 1000100001
```

1000100001 はコンテナー固有の ID であり、エクスポートの所有者 ID と一致します。NFS エクスポー

トの所有者 ID が 54321 である場合は、その番号が Pod 定義で使用されます。コンテナー定義の外部で `securityContext` を指定すると、ID は Pod 内のすべてのコンテナーに対してグローバルになります。

グループ ID と同じように、SCC やプロジェクトで設定されているポリシーに従ってユーザー ID を検証することもできます。SCC の `runAsUser` ストラテジーを RunAsAny に設定した場合は、Pod 定義またはイメージで定義されているすべてのユーザー ID が許可されます。

警告

つまり、0 (root) の UID さえも許可されることになります。

代わりに `runAsUser` ストラテジーを MustRunAsRange に設定した場合は、指定したユーザー ID について、許可される ID の範囲にあるかどうかが検証されます。Pod でユーザー ID を指定しない場合は、デフォルト ID が許可されるユーザー ID の範囲の最小値に設定されます。

先の「NFS の例」に戻って、コンテナーでその UID を 1000100001 に設定する必要があります (上記の Pod の例を参照してください)。デフォルトプロジェクトと制限付き SCC を想定した場合、Pod で要求した 1000100001 というユーザー ID は許可されず、したがって Pod は失敗します。Pod が失敗するのは以下の理由によります。

- Pod が独自のユーザー ID として 1000100001 を要求している。
- 使用可能なすべての SCC が独自の `runAsUser` ストラテジーとして MustRunAsRange を使用しており、そのため UID の範囲チェックが要求される。
- 1000100001 が SCC にもプロジェクトのユーザー ID 範囲にも含まれていない。

この状況に対応するには、適切なユーザー ID 範囲を指定して新規 SCC を作成します。また、新規プロジェクトを適切なユーザー ID 範囲を定義して作成することもできます。さらに、以下のような推奨されない他のオプションがあります。

- 最小および最大のユーザー ID 範囲に 1000100001 を組み込むように 制限付き SCC を変更できます。ただし、これは定義済みの SCC の変更をできる限り避ける必要があるため推奨されません。
- RunAsAny を `runAsUser` の値に使用できるように 制限付き SCC を変更できます。これにより、ID 範囲のチェックを省略できます。この方法ではコンテナーが root として実行される可能性があるため、この方法を使用しないことを「強く」推奨します。
- *ユーザー ID 1000100001 を許可するように デフォルトプロジェクトの UID 範囲を変更できます。通常、この方法も推奨できません。ユーザー ID に単一範囲しか指定できず、範囲が変更された場合に他の Pod が実行されなくなる可能性があるためです。*

ユーザー ID とカスタム SCC

定義済みの SCC を変更することは可能な限り避ける必要があります。組織のセキュリティー上のニーズに合ったカスタム SCC を作成するか、または必要なユーザー ID をサポートする新規プロジェクトを作成することを推奨します。

前述の例にあるような状況に対応するため、以下のようにカスタム SCC を作成することができます。

- 最小と最大のユーザー ID を定義する。
- UID の範囲チェックを引き続き実施する。
- 1000100001 という UID を許可する。

以下は例を示しています。

```
$ oc export scc nfs-scc

allowHostDirVolumePlugin: false (1)
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc (2)
priority: 9 (3)
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsRange (4)
  uidRangeMax: 1000100001 (5)
  uidRangeMin: 1000100001
...
```

(1) allowXX のブール値は制限付き SCC の場合と同じです。

(2) この新規 SCC の名前は nfs-scc です。

(3) 数値が大きいほど優先順位が高くなります。Nil の場合や省略した場合は優先順位が最も低くなります。優先順位が高い SCC は優先順位が低い SCC より前に並べ替えられるため、新規 Pod と一致する確率が高くなります。

(4) runAsUser ストラテジーは MustRunAsRange に設定されています。つまり、UID の範囲チェックが実施されます。

(5) UID の範囲は 1000100001 ～ 1000100001 です (1 つの値の範囲)。

これで、先の例の Pod 定義に `runAsUser: 1000100001` が表示され、Pod が新規の nfs-scc と一致し、UID 1000100001 で実行できるようになります。

23.18.6. SELinux オプション

特権付き SCC を除くすべての定義済み SCC では、`seLinuxContext` を MustRunAs に設定します。そのため、Pod の要件と一致する可能性が高い SCC の場合、Pod での SELinux ポリシーの使用を強制的に実行します。Pod で使用される SELinux ポリシーは、その Pod 自体やイメージ、SCC、またはプロジェクト (デフォルトを指定する) で定義できます。

SELinux のラベルは Pod の `securityContext.seLinuxOptions` セクションで定義でき、`user`、`role`、`type`、および `level` を使用できます。

注記

このトピックでは、レベルと MCS ラベルを置き換え可能な用語として使用します。

```
...
 securityContext: 1
    seLinuxOptions:
      level: "s0:c123,c456" 2
...
```

1 *level は、Pod 全体に対してグローバルに定義することも、コンテナーごとに個別に定義することもできます。*

2 *SELinux の level ラベル。*

以下の例は SCC とデフォルトプロジェクトからの抜粋です。

```
$ oc export scc scc-name
...
seLinuxContext:
  type: MustRunAs 1

# oc export project default
...
metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c1,c0 2
...
```

1 *MustRunAs によりボリュームのラベルが再設定されます。*

2 *ラベルを Pod や SCC で指定しない場合は、プロジェクトのデフォルトが適用されます。*

特権付き SCC を除くすべての定義済み SCC では、`seLinuxContext` を MustRunAs に設定します。これにより、Pod での MCS ラベルの使用が強制的に実行されます。MCS ラベルは、Pod 定義やイメージで定義するか、またはデフォルトとして指定されます。

SCC によって、SELinux ラベルを必要とするかどうかが決まります。また、SCC でデフォルトラベルを指定できます。`seLinuxContext` ストラテジーを MustRunAs に設定していて、Pod (またはイメージ) がラベルを定義していない場合は、OpenShift Container Platform は SCC 自体またはプロジェクトから選択されるラベルにデフォルト設定されます。

`seLinuxContext` を RunAsAnyに設定した場合は、デフォルトラベルは提供されず、コンテナーによって最終的なラベルが決められます。Docker の場合、コンテナーでは一意の MCS ラベルが使用されますが、これが既存のストレージマウントのラベル付けに一致する可能性はほとんどありません。SELinux 管理をサポートしているボリュームについては、指定されるラベルからアクセスできるようにラベルの再設定がなされ、ラベルの排他性によってはそのラベルのみがアクセスできるようになります。

この場合、特権なしコンテナーについては以下の 2 つが関係します。

- *ボリュームには、特権なしコンテナーからアクセス可能な type が付与されます。この type は通常は `container_file_t` で、ボリュームがコンテナーのコンテンツとして扱われます。従来、指定されていたラベルは `svirt_sandbox_file_t` でしたが、このラベルはセキュリティー上の問題のために使用されなくなりました。*

- `level` を指定した場合は、指定される MCS ラベルを使用してボリュームのラベルが設定されます。

Pod からボリュームにアクセスできるようにするためには、Pod で両方のボリュームカテゴリーを持つ必要があります。たとえば、s0:c1,c2 を持つ Pod は、s0:c1,c2 のボリュームにアクセスでき、s0 のボリュームにはすべての Pod からアクセスできるようにします。

Pod で承認が失敗する場合、またはパーミッションエラーが原因でストレージのマウントが失敗している場合は、SELinux の実施による干渉が生じている可能性があります。これについては、たとえば以下を実行してチェックできます。

```
# ausearch -m avc --start recent
```

これは、ログファイルに AVC (Access Vector Cache) のエラーがないかどうかを検査します。

23.19. セレクターとラベルによるボリュームのバインディング

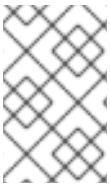
23.19.1. 概要

ここでは、selector 属性と label 属性を使用して Persistent Volume Claim (永続ボリューム要求、PVC) を永続ボリューム (PV) にバインドするための必要な手順について説明します。セレクターとラベルを実装することで、通常のユーザーは、クラスター管理者が定義する識別子を使用して、プロビジョニングされたストレージをターゲットに設定することができます。

23.19.2. 必要になる状況

静的にプロビジョニングされるストレージの場合、永続ストレージを必要とする開発者は、PVC をデプロイしてバインドするためにいくつかの PV の識別属性を把握しておく必要がありますが、その際、問題となる状況がいくつか生じます。通常のユーザーは、PVC のデプロイでも PV の値の指定においても、クラスター管理者に問い合わせをする必要が生じる場合があります。PV 属性だけでは、ストレージボリュームの用途も、ボリュームをグループ化する方法も確認できないためです。

selector 属性と label 属性を使用すると、ユーザーが意識せずに済むように PV の詳細を抽象化できると同時に、分かりやすくカスタマイズ可能なタグを使用してボリュームを識別する手段をクラスター管理者に提供できます。セレクターとラベルによるバインディングの方法を使用することで、ユーザーは管理者によって定義されているラベルのみを確認することが必要になります。



注記

セレクターとラベルの機能は、現時点では静的にプロビジョニングされるストレージの場合にのみ使用できます。現時点では、動的にプロビジョニングされるストレージ用には実装されていません。

23.19.3. デプロイメント

このセクションでは、PVC の定義方法とデプロイ方法を説明します。

23.19.3.1. 前提条件

1. 実行中の OpenShift Container Platform 3.3 以降のクラスター
2. サポート対象のストレージプロバイダーによって提供されているボリューム

3. *cluster-admin ロールのバインディングを持つユーザー*

23.19.3.2. 永続ボリュームと要求の定義

1. *cluser-admin ユーザーとして、PV を定義します。この例では GlusterFS ボリュームを使用します。プロバイダーの設定については、該当する「ストレージプロバイダー」を参照してください。*

例23.9 ラベルのある永続ボリューム

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
  labels: (1)
    volume-type: ssd
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle
```

(1) *セレクターがすべての PV のラベルと一致する PVC がバインドされます (PV が使用可能であることを前提とします)。*

2. *PVC を定義します。*

例23.10 セレクターのある Persistent Volume Claim (永続ボリューム要求)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
  selector: (1)
    matchLabels: (2)
      volume-type: ssd
      aws-availability-zone: us-east-1
```

(1) *selectors セクションの始まりです。*

(2) *ユーザーがストレージを要求する際に使用するラベルすべてを一覧表示します。ターゲットとなる PV の「すべての」 ラベルと一致する必要があります。*

23.19.3.3. 永続ボリュームと要求のデプロイ

cluster-admin ユーザーとして、永続ボリュームを作成します。

例23.11 永続ボリュームの作成

```
# oc create -f gluster-pv.yaml
persistentVolume "gluster-volume" created

# oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES   STATUS
CLAIM     REASON    AGE
gluster-volume           map[]     2147483648   RWX
Available                        2s
```

PV が作成されると、セレクターがその「すべての」ラベルと一致するユーザーであれば PVC を作成できます。

例23.12 Persistent Volume Claim (永続ボリューム要求) の作成

```
# oc create -f gluster-pvc.yaml
persistentVolumeClaim "gluster-claim" created
# oc get pvc
NAME          LABELS    STATUS    VOLUME
gluster-claim           Bound     gluster-volume
```

23.20. コントローラー管理の割り当ておよび割り当て解除

23.20.1. 概要

OpenShift Container Platform 3.4 の時点では、ノードセット自体による各自のボリュームの割り当て/割り当て解除操作の管理のままにするのではなく、管理者がクラスターのマスターで実行されるコントローラーを有効にして、ノードセットに代わってボリュームの割り当て/割り当て解除操作を管理することができます。

コントローラー管理の割り当ておよび割り当て解除を有効にすることには、以下のメリットがあります。

- ノードが失われた場合でも、そのノードに割り当てられていたボリュームの割り当て解除をコントローラーによって実行でき、これを別の場所で再び割り当てることができます。
- 割り当て/割り当て解除に必要な認証情報をすべてのノードで用意する必要がないため、セキュリティーが強化されます。

OpenShift Container Platform 3.6 の時点では、コントローラーで管理される割り当て/割り当て解除はデフォルトの設定になっています。

23.20.2. 割り当て/割り当て解除の管理元の判別

ノード自体にアノテーション `volumes.kubernetes.io/controller-managed-attach-detach` が設定されている場合、そのノードの割り当て/割り当て解除操作はコントローラーによって管理されます。コントローラーは、すべてのノードでこのアノテーションの有無を自動的に検査し、その有無に応じて動作します。したがって、ユーザーがノードでこのアノテーションの有無を調べることで、コントローラーが管理する割り当て/割り当て解除がそのノードで有効にされているかどうかを判別することができます。

さらに、ノードでコントローラー管理の割り当て/割り当て解除が選択されていることを確認するには、ノードのログで以下の行を検索します。

```
Setting node annotation to enable volume controller attach/detach
```

この行が見つからない場合は、以下の行が代わりにログに含まれているはずです。

```
Controller attach/detach is disabled for this node; Kubelet will attach
and detach volumes
```

コントローラーの端末から、コントローラーが特定ノードの割り当て/割り当て解除操作を管理しているかどうかを確認するには、まずロギングレベルを少なくとも 4 に設定する必要があります。次に、以下の行を見つけます。

```
processVolumesInUse for node <node_hostname>
```

ログの表示方法とロギングレベルの設定方法については、「ロギングレベルの設定」を参照してください。

23.20.3. コントローラー管理の割り当て/割り当て解除を有効にするノードの設定

コントローラー管理の割り当て/割り当て解除を有効にするには、個々のノードで独自の割り当て/割り当て解除をオプトインし、無効にするように設定します。編集対象のノード設定ファイルについての詳細は、「ノード設定ファイル」を参照してください。このファイルには、以下の行を追加します。

```
kubeletArguments:
  enable-controller-attach-detach:
  - "true"
```

ノードを設定したら、設定を有効にするためにノードを再起動する必要があります。

23.21. 永続ボリュームスナップショット

23.21.1. 概要

重要

永続ボリュームスナップショットはテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

ストレージシステムの多くは、データを損失から保護するために永続ボリューム (PV) の「スナップショット」を作成する機能を備えています。外部のスナップショットコントローラーおよびプロビジョナーは、この機能を OpenShift Container Platform クラスターで使用して OpenShift Container Platform API を使用してボリュームスナップショットを扱う方法を提供しています。

本書では、OpenShift Container Platform におけるボリュームスナップショットのサポートの現状について説明しています。PV、Persistent Volume Claim (永続ボリューム要求、PVC)、および動的プロビジョニングについて理解しておくことを推奨されます。

23.21.2. 機能

- *`PersistentVolumeClaim` にバインドされる `PersistentVolume` のスナップショットの作成*
- *既存の `VolumeSnapshots` の一覧表示*
- *既存の `VolumeSnapshot` の削除*
- *既存の `VolumeSnapshot` からの `PersistentVolume` の新規作成*
- *サポートされている `PersistentVolume` のタイプ:*
  - *AWS Elastic Block Store (EBS)*
  - *Google Compute Engine (GCE) Persistent Disk (PD)*

23.21.3. インストールと設定

外部のスナップショットコントローラーおよびプロビジョナーは、ボリュームスナップショットの機能を提供する外部コンポーネントです。これらの外部コンポーネントはクラスターで実行されます。コントローラーは、ボリュームスナップショットの作成、削除、および関連イベントのレポートを行います。プロビジョナーは、ボリュームスナップショットから新規の `PersistentVolumes` を作成します。詳細は、「スナップショットの作成」および「スナップショットの復元」を参照してください。

23.21.3.1. 外部のコントローラーおよびプロビジョナーの起動

外部のコントローラーおよびプロビジョナーサービスはコンテナーイメージとして配布され、OpenShift Container Platform クラスターで通常どおり実行できます。また、コントローラーおよびプロビジョナーの RPM バージョンもあります。

API オブジェクトを管理しているコンテナーを許可するには、以下のようにして、必要なロールベースアクセス制御 (RBAC) ルールを管理者が設定する必要があります。

1. *ServiceAccount と ClusterRole を以下のように作成します。*

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: snapshot-controller-runner
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: snapshot-controller-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
  - apiGroups: ["apiextensions.k8s.io"]
    resources: ["customresourcedefinitions"]
    verbs: ["create", "list", "watch", "delete"]
  - apiGroups: ["volumesnapshot.external-storage.k8s.io"]
    resources: ["volumesnapshots"]
    verbs: ["get", "list", "watch", "create", "update", "patch",
"delete"]
  - apiGroups: ["volumesnapshot.external-storage.k8s.io"]
    resources: ["volumesnapshotdatas"]
    verbs: ["get", "list", "watch", "create", "update", "patch",
"delete"]
```

2. *ClusterRoleBinding を使用して、以下のようにルールをバインドします。*

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: snapshot-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: snapshot-controller-role
subjects:
- kind: ServiceAccount
  name: snapshot-controller-runner
  namespace: default
```

外部のコントローラーおよびプロビジョナーを Amazon Web Services (AWS) にデプロイしている場合、それらはアクセスキーを使用して認証できる必要があります。認証情報を Pod に提供するために、管理者が以下のように新規のシークレットを作成します。

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: awskeys
type: Opaque
data:
  access-key-id: <base64 encoded AWS_ACCESS_KEY_ID>
  secret-access-key: <base64 encoded AWS_SECRET_ACCESS_KEY>
```

AWS における外部のコントローラーおよびプロビジョナーコンテナーのデプロイメント (どちらの Pod コンテナーもシークレットを使用して AWS のクラウドプロバイダー API にアクセスします):

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
    spec:
      serviceAccountName: snapshot-controller-runner
      containers:
        - name: snapshot-controller
          image: "registry.access.redhat.com/openshift3/snapshot-
controller:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "aws"]
          env:
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: access-key-id
            - name: AWS_SECRET_ACCESS_KEY
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: secret-access-key
        - name: snapshot-provisioner
          image: "registry.access.redhat.com/openshift3/snapshot-
provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "aws"]
          env:
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: access-key-id
            - name: AWS_SECRET_ACCESS_KEY
              valueFrom:
```

```
                secretKeyRef:
                  name: awskeys
                  key: secret-access-key
```

GCE の場合、GCE のクラウドプロバイダー API にアクセスするためにシークレットを使用する必要はありません。管理者は以下のようにデプロイメントに進むことができます。

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
    spec:
      serviceAccountName: snapshot-controller-runner
      containers:
        - name: snapshot-controller
          image: "registry.access.redhat.com/openshift3/snapshot-
controller:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "gce"]
        - name: snapshot-provisioner
          image: "registry.access.redhat.com/openshift3/snapshot-
provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "gce"]
```

23.21.3.2. スナップショットユーザーの管理

クラスターの設定によっては、管理者以外のユーザーが API サーバーで VolumeSnapshot オブジェクトを操作できるようにする必要があります。これは、特定のユーザーまたはグループにバインドされる ClusterRole を作成することで実行できます。

たとえば、ユーザー「alice」がクラスター内のスナップショットを操作する必要があるとします。クラスター管理者は以下の手順を実行します。

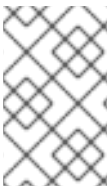
1. 新規の ClusterRole を定義します。

```
apiVersion: v1
kind: ClusterRole
metadata:
  name: volumesnapshot-admin
rules:
- apiGroups:
  - "volumesnapshot.external-storage.k8s.io"
  attributeRestrictions: null
  resources:
  - volumesnapshots
  verbs:
```

```
    - create
    - delete
    - deletecollection
    - get
    - list
    - patch
    - update
    - watch
```

2. ClusterRole バインドオブジェクトを作成してクラスターロールをユーザー「alice」にバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: volumesnapshot-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: volumesnapshot-admin
subjects:
- kind: User
  name: alice
```



注記

これは API アクセス設定の一例にすぎません。VolumeSnapshot オブジェクトの動作は他の OpenShift Container Platform API オブジェクトと同様です。API RBAC の管理についての詳細は、API アクセス制御のドキュメントを参照してください。

23.21.4. ボリュームスナップショットとボリュームスナップショットデータのライフサイクル

23.21.4.1. Persistent Volume Claim (永続ボリューム要求) と永続ボリューム

PersistentVolumeClaim は PersistentVolume にバインドされます。PersistentVolume のタイプは、スナップショットがサポートするいずれかの永続ボリュームタイプである必要があります。

23.21.4.1.1. スナップショットプロモーター

StorageClass を作成するには、以下を実行します。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: snapshot-promoter
provisioner: volumesnapshot.external-storage.k8s.io/snapshot-promoter
```

この StorageClass は、先に作成した VolumeSnapshot から PersistentVolume を復元する場合に必要です。

23.21.4.2. スナップショットの作成

PV のスナップショットを作成するには、新しい VolumeSnapshot オブジェクトを以下のように作成します。

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: snapshot-demo
spec:
  persistentVolumeClaimName: ebs-pvc
```

persistentVolumeClaimName は、PersistentVolume にバインドされる PersistentVolumeClaim の名前です。この特定 PV のスナップショットが作成されます。

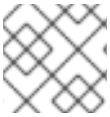
次に、VolumeSnapshot に基づく VolumeSnapshotData オブジェクトが自動的に作成されます。VolumeSnapshot と VolumeSnapshotData の関係は PersistentVolumeClaim と PersistentVolume の関係に似ています。

PV のタイプによっては、反映される VolumeSnapshot の状態に応じ、操作が複数の段階にわたる場合があります。

1. 新規 VolumeSnapshot オブジェクトが作成されます。
2. コントローラーによってスナップショット操作が開始されます。スナップショット対象の PersistentVolume をフリーズし、アプリケーションを一時停止する必要が生じる場合があります。
3. ストレージシステムによるスナップショットの作成が完了し (スナップショットが「切り取られ」)、スナップショット対象の PersistentVolume が通常の操作に戻ります。スナップショット自体の準備はまだできていません。ここでの状態は Pending タイプで状態の値は True です。実際のスナップショットを表す VolumeSnapshotData オブジェクトが新たに作成されます。
4. 新規スナップショットの作成が完成し、使用できる状態になります。ここでの状態は Ready タイプで、状態の値は True です。

重要

データの整合性の確保 (Pod/アプリケーションの停止、キャッシュのフラッシュ、ファイルシステムのフリーズなど) はユーザーの責任で行う必要があります。



注記

エラーの場合は、VolumeSnapshot の状態が Error 状態に追加されます。

VolumeSnapshot の状態を表示するには、以下を実行します。

```
$ oc get volumesnapshot -o yaml
```

状態が以下のように表示されます。

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
  kind: VolumeSnapshot
  metadata:
```

```
    clusterName: ""
    creationTimestamp: 2017-09-19T13:58:28Z
    generation: 0
    labels:
      Timestamp: "1505829508178510973"
    name: snapshot-demo
    namespace: default
    resourceVersion: "780"
    selfLink: /apis/volumesnapshot.external-
storage.k8s.io/v1/namespaces/default/volumesnapshots/snapshot-demo
    uid: 9cc5da57-9d42-11e7-9b25-90b11c132b3f
  spec:
    persistentVolumeClaimName: ebs-pvc
    snapshotDataName: k8s-volume-snapshot-9cc8813e-9d42-11e7-8bed-
90b11c132b3f
  status:
    conditions:
    - lastTransitionTime: null
      message: Snapshot created successfully
      reason: ""
      status: "True"
      type: Ready
    creationTimestamp: null
```

23.21.4.3. スナップショットの復元

VolumeSnapshot から PV を復元するには、以下のように PVC を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: snapshot-pv-provisioning-demo
  annotations:
    snapshot.alpha.kubernetes.io/snapshot: snapshot-demo
spec:
  storageClassName: snapshot-promoter
```

annotations:snapshot.alpha.kubernetes.io/snapshot は、復元する VolumeSnapshot の名前です。 storageClassName:StorageClass はVolumeSnapshot を復元するために管理者によって作成されます。

新規の PersistentVolume が作成されて PersistentVolumeClaim にバインドされます。PV のタイプによっては処理に数分の時間がかかることがあります。

23.21.4.4. スナップショットの削除

snapshot-demo を削除するには、以下を実行します。

```
$ oc delete volumesnapshot/snapshot-demo
```

VolumeSnapshot にバインドされている VolumeSnapshotData が自動的に削除されます。

第24章 永続ストレージの例

24.1. 概要

以下のセクションでは、一般的なストレージのユースケースを定義するための総合的な手順について詳しく説明します。以下の例では、永続ボリュームとそのセキュリティーの管理だけでなく、システムのユーザーがボリュームに対する要求を行う方法についても取り上げます。

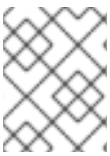
- 2 つの Pod 間での NFS PV の共有
- Ceph-RBD ブロックストレージボリューム
- GlusterFS ボリュームを使用した共有ストレージ
- GlusterFS を使用した動的プロビジョニングストレージ
- 特権付き Pod への PV のマウント
- GlusterFS ストレージによる Docker レジストリーのサポート
- ラベルによる永続ボリュームのバインド
- 動的プロビジョニングでの StorageClass の使用
- 既存レガシーストレージでの StorageClass の使用
- Azure Blob Storage での統合 Docker レジストリーの設定

24.2. 2 つの PERSISTENT VOLUME CLAIM (永続ボリューム要求) 間での NFS マウントの共有

24.2.1. 概要

以下のユースケースでは、クラスター管理者が 2 つの別々のコンテナーで共有ストレージを利用しようとしている場合に、その解決策を設定する方法について説明します。ここでは NFS の使用例を取り上げていますが、GlusterFS など他の共有ストレージタイプにも簡単に応用できます。また、この例では共有ストレージに関連した Pod のセキュリティーの設定についても説明します。

NFS を使用した永続ストレージでは、永続ボリューム (PV)、Persistent Volume Claim (永続ボリューム要求、PVC)、永続ストレージとしての NFS の使用について説明しています。このトピックでは既存の NFS クラスターと OpenShift Container Platform 永続ストアの使用について詳しく説明しますが、既存の NFS サーバーおよびエクスポートが OpenShift Container Platform インフラストラクチャーに存在することを前提とします。



注記

oc コマンドはすべて OpenShift Container Platform のマスターホストで実行されます。

24.2.2. 永続ボリュームの作成

PV オブジェクトを OpenShift Container Platform で作成する前に、永続ボリューム (PV) ファイルを以下のように定義します。

例24.1 NFS を使用した永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv (1)
spec:
  capacity:
    storage: 1Gi (2)
  accessModes:
    - ReadWriteMany (3)
  persistentVolumeReclaimPolicy: Retain (4)
  nfs: (5)
    path: /opt/nfs (6)
    server: nfs.f22 (7)
    readOnly: false
```

(1) PV の名前。Pod 定義で参照されたり、各種の oc ボリュームコマンドで表示されたりします。

(2) このボリュームに割り当てられるストレージの量。

(3) accessModes は、PV と PVC を一致させるためのラベルとして使用されます。現時点で、これらはいずれの形態のアクセス制御も定義しません。

(4) ボリューム回収ポリシー Retain は、ボリュームにアクセスする Pod が終了した後にもボリュームが維持されることを示します。

(5) 使用するボリュームタイプを定義します。この例では NFS プラグインを定義しています。

(6) NFS マウントパスです。

(7) NFS サーバーです。IP アドレスで指定することもできます。

PV の定義を nfs-pv.yaml などのファイルに保存し、以下のように永続ボリュームを作成します。

```
# oc create -f nfs-pv.yaml
persistentvolume "nfs-pv" created
```

永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME          LABELS    CAPACITY    ACCESSMODES    STATUS       CLAIM
REASON    AGE
nfs-pv        <none>    1Gi         RWX            Available
37s
```

24.2.3. Persistent Volume Claim (永続ボリューム要求) の作成

Persistent Volume Claim (永続ボリューム要求、PVC) では、必要なアクセスモードとストレージ容量を指定します。現時点では、これら 2 つの属性のみに基づいて PVC が 1 つの PV にバインドされます。PV が PVC にバインドされると、その PV は基本的に当該 PVC のプロジェクトに結び付けられ、

別の PVC にバインドすることはできません。PV と PVC には 1 対 1 のマッピングが存在します。ただし、同じプロジェクト内の複数の Pod が同じ PVC を使用することは可能です。以下の例ではそのユースケースを取り上げています。

例24.2 PVC オブジェクト定義

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc  1
spec:
  accessModes:
  - ReadWriteMany  2
  resources:
     requests:
       storage: 1Gi  3
```

1 この要求名は、volumes セクションで Pod によって参照されます。

2 PV についての先の説明にあるように、accessModes はアクセスを実施するものではなく、PV と PVC を一致させるためのラベルとして機能します。

3 この要求は、1Gi 以上の容量がある PV を検索します。

PVC の定義を nfs-pvc.yaml などのファイルに保存し、以下のように PVC を作成します。

```
# oc create -f nfs-pvc.yaml
persistentvolumeclaim "nfs-pvc" created
```

PVC が作成されていて、予想される PV にバインドされていることを確認します。

```
# oc get pvc
NAME            LABELS    STATUS    VOLUME       CAPACITY   ACCESSMODES
AGE
nfs-pvc         <none>    Bound     nfs-pv       1Gi        RWX
24s
                                      1
```

1 要求の nfs-pvc が nfs-pv PV にバインドされています。

24.2.4. NFS ボリュームアクセスの確保

NFS サーバー内のノードへのアクセスが必要です。このノードで、以下のように NFS エクスポートのマウントを確認します。

```
[root@nfs nfs]# ls -lZ /opt/nfs/
total 8
-rw-r--r--. 1 root 100003  system_u:object_r:usr_t:s0     10 Oct 12 23:27
```

```
test2b
                1
                        2
```

1 所有者の ID は 0 です。

2 グループの ID は 100003 です。

NFS マウントにアクセスするためには、コンテナーが SELinux ラベルを一致する必要があり、UID を 0 にして実行するか、または補助グループ範囲内の 100003 に指定して実行します。NFS マウントのグループ (後の Pod 定義で定義される) に一致させることでボリュームにアクセスできるようにします。

デフォルトでは、SELinux では Pod からリモートの NFS サーバーへの書き込みは許可されません。SELinux を各ノードで有効にした状態で NFS ボリュームへの書き込みを有効にするには、以下のコマンドを実行します。

```
# setsebool -P virt_use_nfs on
```

24.2.5. Pod の作成

Pod 定義ファイルまたはテンプレートファイルを使用して Pod を定義することができます。以下は、1 つのコンテナーを作成して NFS ボリュームを読み書きアクセス用にマウントする Pod 仕様です。

例24.3 Pod オブジェクトの定義

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-nfs-pod 1
  labels:
    name: hello-openshift-nfs-pod
spec:
  containers:
    - name: hello-openshift-nfs-pod
      image: openshift/hello-openshift 2
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
        - name: nfsvol 3
          mountPath: /usr/share/nginx/html 4
  securityContext:
      supplementalGroups: [100003] 5
      privileged: false
  volumes:
    - name: nfsvol
      persistentVolumeClaim:
        claimName: nfs-pvc 6
```

1 oc get pod によって表示されるこの Pod の名前。

2 この Pod が実行するイメージ。

3. ボリュームの名前。この名前は containers セクションと volumes セクションの両方で同じにする必要があります。
4. コンテナーで表示されるマウントパス。
5. コンテナーに割り当てるグループ ID。
6. 先の手順で作成した PVC。

Pod 定義を nfs.yaml などのファイルに保存し、以下のように Pod を作成します。

```
# oc create -f nfs.yaml
pod "hello-openshift-nfs-pod" created
```

Pod が作成されていることを確認します。

```
# oc get pods
NAME                         READY     STATUS    RESTARTS   AGE
hello-openshift-nfs-pod      1/1       Running   0          4s
```

詳細は oc describe pod コマンドで以下のように表示されます。

```
[root@ose70 nfs]# oc describe pod hello-openshift-nfs-pod
Name:    hello-openshift-nfs-pod
Namespace:   default (1)
Image(s):   fedora/S3
Node:    ose70.rh7/192.168.234.148 (2)
Start Time:   Mon, 21 Mar 2016 09:59:47 -0400
Labels:    name=hello-openshift-nfs-pod
Status:    Running
Reason:
Message:
IP:    10.1.0.4
Replication Controllers: <none>
Containers:
  hello-openshift-nfs-pod:
    Container ID:
docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc6f
    Image:  fedora/S3
    Image ID:
docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a00a
    QoS Tier:
      cpu:  BestEffort
      memory:  BestEffort
    State:  Running
      Started:  Mon, 21 Mar 2016 09:59:49 -0400
    Ready:  True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type  Status
  Ready  True
Volumes:
```

```
  nfsvol:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
the same namespace)
    ClaimName: nfs-pvc (3)
    ReadOnly: false
  default-token-a06zb:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-a06zb
Events: (4)
  FirstSeen LastSeen Count From   SubobjectPath
Reason  Message
  ————————— ———————— ————— ————   ———————————————
———————  ———————
  4m  4m  1 {scheduler }
Scheduled Successfully assigned hello-openshift-nfs-pod to ose70.rh7
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Pulled  Container image "openshift3/ose-pod:v3.1.0.4" already present on
machine
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Created  Created with docker id 866a37108041
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD
Started  Started with docker id 866a37108041
  4m  4m  1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}
Pulled  Container image "fedora/S3" already present on machine
  4m  4m  1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}
Created  Created with docker id a3292104d6c2
  4m  4m  1 {kubelet ose70.rh7} spec.containers{hello-openshift-nfs-pod}
Started  Started with docker id a3292104d6c2
```

(1) *プロジェクト (namespace) 名。*

(2) *Pod を実行している OpenShift Container Platform ノードの IP アドレス。*

(3) *Pod で使用される PVC 名。*

(4) *Pod の起動と NFS ボリュームのマウントをもたらすイベントの一覧。ボリュームをマウントできない場合、コンテナーは正常に起動しません。*

`oc get pod <name> -o yaml` コマンドでは、Pod の承認に使用される SCC や Pod のユーザー ID とグループ ID、SELinux ラベルなどの内部情報がさらに表示されます。

```
[root@ose70 nfs]# oc get pod hello-openshift-nfs-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    openshift.io/scc: restricted (1)
  creationTimestamp: 2016-03-21T13:59:47Z
  labels:
    name: hello-openshift-nfs-pod
  name: hello-openshift-nfs-pod
  namespace: default (2)
  resourceVersion: "2814411"
  selflink: /api/v1/namespaces/default/pods/hello-openshift-nfs-pod
  uid: 2c22d2ea-ef6d-11e5-adc7-000c2900f1e3
```

```
spec:
  containers:
  - image: fedora/S3
    imagePullPolicy: IfNotPresent
    name: hello-openshift-nfs-pod
    ports:
    - containerPort: 80
      name: web
      protocol: TCP
    resources: {}
    securityContext:
      privileged: false
    terminationMessagePath: /dev/termination-log
    volumeMounts:
    - mountPath: /usr/share/S3/html
      name: nfsvol
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-a06zb
      readOnly: true
  dnsPolicy: ClusterFirst
  host: ose70.rh7
  imagePullSecrets:
  - name: default-dockercfg-xvdew
  nodeName: ose70.rh7
  restartPolicy: Always
  securityContext:
    supplementalGroups:
    - 100003 (3)
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  volumes:
  - name: nfsvol
    persistentVolumeClaim:
      claimName: nfs-pvc (4)
  - name: default-token-a06zb
    secret:
      secretName: default-token-a06zb
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2016-03-21T13:59:49Z
    status: "True"
    type: Ready
  containerStatuses:
  - containerID:
docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc6f
    image: fedora/S3
    imageID:
docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a00a
    lastState: {}
    name: hello-openshift-nfs-pod
    ready: true
    restartCount: 0
    state:
      running:
```

```
      startedAt: 2016-03-21T13:59:49Z
hostIP: 192.168.234.148
phase: Running
podIP: 10.1.0.4
startTime: 2016-03-21T13:59:47Z
```

1. Pod が使用する SCC。
2. プロジェクト (namespace) 名。
3. Pod の補助グループ ID (すべてのコンテナー)。
4. Pod で使用される PVC 名。

24.2.6. 同じ PVC を参照する追加 Pod の作成

以下の Pod 定義 (同じ namespace で作成されている) では別のコンテナーを使用しています。ただし、以下の volumes セクションで要求名を指定することで、同じバッキングストレージを使用できます。

例24.4 Pod オブジェクトの定義

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-nfs-pod 1
  labels:
    name: busybox-nfs-pod
spec:
  containers:
  - name: busybox-nfs-pod
    image: busybox 2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: nfsvol-2 3
      mountPath: /usr/share/busybox 4
      readOnly: false
  securityContext:
    supplementalGroups: [100003] 5
    privileged: false
  volumes:
  - name: nfsvol-2
    persistentVolumeClaim:
      claimName: nfs-pvc 6
```

1. oc get pod によって表示されるこの Pod の名前。
2. この Pod が実行するイメージ。
3. ボリュームの名前。この名前は containers セクションと volumes セクションの両方で同じにする必要があります。
4. コンテナーで表示されるマウントパス。

5 コンテナーに割り当てるグループ ID。

6 先に作成されており、別のコンテナーでも使用されている PVC。

Pod 定義を nfs-2.yaml などのファイルに保存し、以下のように Pod を作成します。

```
# oc create -f nfs-2.yaml
pod "busybox-nfs-pod" created
```

Pod が作成されていることを確認します。

```
# oc get pods
NAME                READY     STATUS    RESTARTS   AGE
busybox-nfs-pod     1/1       Running   0          3s
```

詳細は oc describe pod コマンドで以下のように表示されます。

```
[root@ose70 nfs]# oc describe pod busybox-nfs-pod
Name:    busybox-nfs-pod
Namespace:   default
Image(s):   busybox
Node:    ose70.rh7/192.168.234.148
Start Time:   Mon, 21 Mar 2016 10:19:46 -0400
Labels:    name=busybox-nfs-pod
Status:    Running
Reason:
Message:
IP:    10.1.0.5
Replication Controllers: <none>
Containers:
  busybox-nfs-pod:
    Container ID:
docker://346d432e5a4824ebf5a47fceb4247e0568ecc64eadcc160e9bab481aecfb0594
    Image:  busybox
    Image ID:
docker://17583c7dd0dae6244203b8029733bdb7d17fccbb2b5d93e2b24cf48b8bfd06e2
    QoS Tier:
      cpu:  BestEffort
      memory:  BestEffort
    State:  Running
      Started:  Mon, 21 Mar 2016 10:19:48 -0400
    Ready:  True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type  Status
  Ready  True
Volumes:
  nfsvol-2:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
the same namespace)
    ClaimName: nfs-pvc
    ReadOnly: false
```

```
  default-token-32d2z:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-32d2z
Events:
  FirstSeen LastSeen Count From   SubobjectPath    Reason  Message
  ————————— ———————— ————— ————   ———————————    ——————  ———————

  4m  4m  1 {scheduler }       Scheduled Successfully assigned busybox-
nfs-pod to ose70.rh7
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD Pulled
Container image "openshift3/ose-pod:v3.1.0.4" already present on machine
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD Created
Created with docker id 249b7d7519b1
  4m  4m  1 {kubelet ose70.rh7} implicitly required container POD Started
Started with docker id 249b7d7519b1
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Pulled
Container image "busybox" already present on machine
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Created
Created with docker id 346d432e5a48
  4m  4m  1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Started
Started with docker id 346d432e5a48
```

ここから分かるように、いずれのコンテナーでも、バックエンドの同じ NFS マウントに割り当てられた同じストレージ要求を使用しています。

24.3. CEPH RBD を使用した詳細例

24.3.1. 概要

このトピックでは、既存の Ceph クラスターを OpenShift Container Platform の永続ストアとして使用する詳細な例を紹介します。ここでは、作業用の Ceph クラスターがすでに設定されていることを前提とします。まだ設定されていない場合は、Red Hat Ceph Storage の概要について参照してください。

Ceph RADOS ブロックデバイスを使用した永続ストレージでは、永続ボリューム (PV)、Persistent Volume Claim (永続ボリューム要求、PVC)、永続ストレージとしての Ceph RBD の使用について説明しています。

注記

oc … コマンドはすべて OpenShift Container Platform のマスターホストで実行されます。

24.3.2. ceph-common パッケージのインストール

ceph-common ライブラリーは、すべてのスケジュール可能な OpenShift Container Platform ノードにインストールする必要があります。

注記

OpenShift Container Platform のオールインワンホストは、Pod のワークロードを実行するために使用されることは多くありません。したがって、これはスケジュール可能なノードに含まれません。

```
# yum install -y ceph-common
```

24.3.3. Ceph シークレットの作成

ceph auth get-key コマンドを Ceph の MON ノードで実行すると、client.admin ユーザーのキー値が以下のように表示されます。

例24.5 Ceph のシークレットの定義

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFBOFF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== 1
```

1 この base64 キーは、Ceph の MON ノードの 1 つで ceph auth get-key client.admin | base64 コマンドを使用して生成されたものであり、出力をコピーし、これをシークレットキーの値として貼り付けています。

シークレット定義を ceph-secret.yaml などのファイルに保存し、シークレットを作成します。

```
$ oc create -f ceph-secret.yaml
secret "ceph-secret" created
```

シークレットが作成されたことを確認します。

```
# oc get secret ceph-secret
NAME          TYPE      DATA      AGE
ceph-secret   Opaque    1         23d
```

24.3.4. 永続ボリュームの作成

次に、OpenShift Container Platform で PV オブジェクトを作成する前に、永続ボリュームファイルを定義します。

例24.6 Ceph RBD を使用した永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv          1
spec:
  capacity:
    storage: 2Gi         2
  accessModes:
    - ReadWriteOnce      3
  rbd:                   4
    monitors:            5
```

```
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret 6
    fsType: ext4 7
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle
```

1. PV の名前。Pod 定義で参照されたり、各種の oc ボリュームコマンドで表示されたりします。
2. このボリュームに割り当てられるストレージの量。
3. accessModes は、PV と PVC を一致させるためのラベルとして使用します。現時点で、これらはいずれの形態のアクセス制御も定義しません。ブロックストレージはすべて、単一ユーザーに対して定義されます (非共有ストレージ)。
4. 使用するボリュームタイプを定義します。この例では rbd プラグインを定義しています。
5. Ceph モニターの IP アドレスとポートの配列です。
6. 先に定義した Ceph のシークレットです。OpenShift Container Platform から Ceph サーバーへのセキュアな接続を作成するのに使用します。
7. Ceph RBD ブロックデバイスにマウントされるファイルシステムタイプです。

PV の定義を ceph-pv.yaml などのファイルに保存し、永続ボリュームを作成します。

```
# oc create -f ceph-pv.yaml
persistentvolume "ceph-pv" created
```

永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES   STATUS
CLAIM     REASON    AGE
ceph-pv                  <none>    2147483648   RWO           Available
2s
```

24.3.5. Persistent Volume Claim (永続ボリューム要求) の作成

Persistent Volume Claim (永続ボリューム要求、PVC) では、必要なアクセスモードとストレージ容量を指定します。現時点では、これら 2 つの属性のみに基づいて PVC が 1 つの PV にバインドされます。PV が PVC にバインドされると、その PV は基本的に当該 PVC のプロジェクトに結び付けられ、別の PVC にバインドすることはできません。PV と PVC には 1 対 1 のマッピングが存在します。ただし、同じプロジェクト内の複数の Pod が同じ PVC を使用することは可能です。

例24.7 PVC オブジェクト定義

```
kind: PersistentVolumeClaim
apiVersion: v1
```

```
metadata:
  name: ceph-claim
spec:
  accessModes:      1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi  2
```

1 PV についての先の説明にあるように、accessModes はアクセスを実施するものではなく、PV と PVC を一致させるためのラベルとして機能します。

2 この要求は 2Gi 以上の容量を提供する PV を探します。

PVC の定義を ceph-claim.yaml などのファイルに保存し、以下のように PVC を作成します。

```
# oc create -f ceph-claim.yaml
persistentvolumeclaim "ceph-claim" created

#and verify the PVC was created and bound to the expected PV:
# oc get pvc
NAME         LABELS    STATUS    VOLUME    CAPACITY   ACCESSMODES   AGE
ceph-claim   <none>    Bound     ceph-pv   1Gi        RWX           21s
                                           1
```

1 要求が ceph-pv PV にバインドされています。

24.3.6. Pod の作成

Pod 定義ファイルまたはテンプレートファイルを使用して Pod を定義できます。以下は、1 つのコンテナーを作成し、Ceph RBD ボリュームを読み書きアクセス用にマウントする Pod 仕様です。

例24.8 Pod オブジェクトの定義

```
apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1           1
spec:
  containers:
  - name: ceph-busybox
    image: busybox          2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1       3
      mountPath: /usr/share/busybox 4
      readOnly: false
  volumes:
  - name: ceph-vol1         5
    persistentVolumeClaim:
      claimName: ceph-claim 6
```

1. oc get pod によって表示されるこの Pod の名前。
2. この Pod が実行するイメージ。この例では、busybox に sleep の実行を指示しています。
3. 5. ボリュームの名前。この名前は containers セクションと volumes セクションの両方で同じにする必要があります。
4. コンテナーで表示されるマウントパス。
6. Ceph RBD クラスターにバインドされる PVC。

Pod 定義を ceph-pod1.yaml などのファイルに保存し、以下のように Pod を作成します。

```
# oc create -f ceph-pod1.yaml
pod "ceph-pod1" created

#verify pod was created
# oc get pod
NAME        READY     STATUS    RESTARTS   AGE
ceph-pod1   1/1       Running   0          2m
                      1
```

1. しばらくすると、Pod が Running 状態になります。

24.3.7. グループ ID と所有者 ID の定義 (オプション)

Ceph RBD などのブロックストレージを使用する場合、物理ブロックストレージは Pod の管理対象になります。Pod で定義されたグループ ID は、コンテナー内の Ceph RBD マウントと実際のストレージ自体の両方のグループ ID になります。そのため、通常はグループ ID を Pod 仕様に定義する必要はありません。ただし、グループ ID が必要な場合は、以下の Pod 定義の例に示すように fsGroup を使用して定義することができます。

例24.9 グループ ID の Pod 定義

```
...
spec:
  containers:
    - name:
    ...
  securityContext: 1
    fsGroup: 7777  2
...
```

1. securityContext は特定のコンテナーの下位ではなく、この Pod レベルで定義します。
2. Pod 内のコンテナーはすべて同じ fsGroup ID になります。

24.3.8. ceph-user-secret をプロジェクトのデフォルトとして設定する方法

すべてのプロジェクトで永続ストレージを使用できるようにする場合は、デフォルトのプロジェクトテンプレートを修正する必要があります。デフォルトプロジェクトテンプレートの修正についての詳細は、「デフォルトプロジェクトテンプレートの修正」を参照してください。これをデフォルトプロジェクトテンプレートに追加することで、プロジェクトの作成アクセス権を持つすべてのユーザーが Ceph クラスターにアクセスできるようになります。

例24.10 デフォルトプロジェクトの例

```
...
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: v1
  kind: Secret
  metadata:
    name: ceph-user-secret
  data:
    key: yoursupersecretbase64keygoeshere (1)
  type:
    kubernetes.io/rbd
...
```

1. 各自のスーパーシークレット Ceph ユーザーキーをここに base64 形式で記述します。「Ceph のシークレットの作成」を参照してください。

24.4. 動的プロビジョニングでの CEPH RBD の使用

24.4.1. 概要

このトピックでは、既存の Ceph クラスターを OpenShift Container Platform の永続ストレージとして使用する詳細な例を紹介します。ここでは、作業用の Ceph クラスターがすでに設定されていることを前提とします。まだ設定されていない場合は、Red Hat Ceph Storage の概要について参照してください。

「Ceph RADOS ブロックデバイスを使用した永続ストレージ」では、永続ボリューム (PV)、Persistent Volume Claim (永続ボリューム要求、PVC)、永続ストレージとしての Ceph RADOS ブロックデバイス (RBD) の使用方法について説明しています。

注記

- OpenShift Container Platform のマスターホストで、全 oc コマンドを実行します。
- OpenShift Container Platform のオールインワンホストは、Pod のワークロードを実行するために使用されることは多くありません。したがって、これはスケジュール可能なノードに含まれません。

24.4.2. 動的ボリューム用プールの作成

1. 最新の ceph-common パッケージをインストールします。

```
yum install -y ceph-common
```

注記

ceph-common ライブラリーは、all schedulable OpenShift Container Platform ノードにインストールする必要があります。

2. 管理者または MON ノードによって、以下のように動的ボリューム用の新規プールが作成されます。

```
$ ceph osd pool create kube 1024
$ ceph auth get-or-create client.kube mon 'allow r, allow command
"osd blacklist"' osd 'allow class-read object_prefix rbd_children,
allow rwx pool=kube' -o ceph.client.kube.keyring
```

注記

RBD のデフォルトプールを使用することも可能ですが、このオプションは推奨されません。

24.4.3. 動的な永続ストレージでの既存の Ceph クラスターの使用

動的な永続ストレージに既存の Ceph クラスターを使用するには、以下を実行します。

1. client.admin 向けに base64 でエンコードされたキーを作成します。

```
$ ceph auth get client.admin
```

Ceph シークレット定義例

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
  namespace: kube-system
```

```
data:
  key: QVFBOFF2SlZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== (1)
type: kubernetes.io/rbd (2)
```

(1) この base64 キーは、Ceph の MON ノードの 1 つで ceph auth get-key client.admin | base64 コマンドを使用して生成されたものであり、出力をコピーし、これをシークレットキーの値として貼り付けています。

(2) この値は、Ceph RBD を動的プロビジョニングで機能させるために必要です。

2. client.admin 用に Ceph シークレットを作成します。

```
$ oc create -f ceph-secret.yaml
secret "ceph-secret" created
```

3. シークレットが作成されたことを確認します。

```
$ oc get secret ceph-secret
NAME          TYPE                DATA      AGE
ceph-secret   kubernetes.io/rbd   1         5d
```

4. ストレージクラスを作成します。

```
$ oc create -f ceph-storageclass.yaml
storageclass "dynamic" created
```

Ceph ストレージクラスの例

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: dynamic
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/rbd
parameters:
  monitors: 192.168.1.11:6789,192.168.1.12:6789,192.168.1.13:6789 (1)
  adminId: admin (2)
  adminSecretName: ceph-secret (3)
  adminSecretNamespace: kube-system (4)
  pool: kube (5)
  userId: kube (6)
  userSecretName: ceph-user-secret (7)
```

(1) Ceph が監視する IP アドレスのコンマ区切りの一覧。この値は必須です。

(2) Ceph クライアント ID。プール内にイメージを作成する権限があります。デフォルトは admin です。

(3) adminId のシークレット名。この値は必須です。設定するシークレットには kubernetes.io/rbd が含まれる必要があります。

4. *adminSecret の namespace。デフォルトは default です。*

5. *Ceph RBD プール。デフォルトは rbd ですが、この値は推奨されません。*

6. *Ceph RBD イメージのマッピングに使用される Ceph クライアント ID。デフォルトは adminId のシークレット名と同じです。*

7. *Ceph RBD イメージをマッピングするための userId の Ceph シークレット名。PVC と同じ namespace に存在する必要があります。Ceph シークレットが新規プロジェクトのデフォルトとして設定されていない限り、このパラメーターの値を指定する必要があります。*

5. *ストレージクラスが作成されたことを確認します。*

```
$ oc get storageclasses
NAME                 TYPE
dynamic (default)   kubernetes.io/rbd
```

6. *PVC オブジェクト定義を作成します。*

PVC オブジェクト定義例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim-dynamic
spec:
  accessModes:  1
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi  2
```

1. *accessModes はアクセス権としての効果はなく、代わりに PV と PVC を照合するラベルとして機能します。*

2. *この要求は 2Gi 以上の容量を提供する PV を探します。*

7. *PVC を作成します。*

```
$ oc create -f ceph-pvc.yaml
persistentvolumeclaim "ceph-claim-dynamic" created
```

8. *PVC が作成されていて、予想される PV にバインドされていることを確認します。*

```
$ oc get pvc
NAME         STATUS   VOLUME
CAPACITY ACCESSMODES   AGE
ceph-claim   Bound    pvc-f548d663-3cac-11e7-9937-0024e8650c7a 2Gi
RWO           1m
```

9. *Pod オブジェクト定義を以下のように作成します。*

Pod オブジェクトの定義例

```
apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1 (1)
spec:
  containers:
  - name: ceph-busybox
    image: busybox (2)
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1 (3)
      mountPath: /usr/share/busybox (4)
      readOnly: false
  volumes:
  - name: ceph-vol1
    persistentVolumeClaim:
      claimName: ceph-claim (5)
```

(1) *oc get pod によって表示されるこの Pod の名前。*

(2) *この Pod が実行するイメージ。この例では、busybox は sleep に設定されています。*

(3) *ボリュームの名前。この名前は containers セクションと volumes セクションの両方で同じにする必要があります。*

(4) *コンテナーでのマウントパス。*

(5) *Ceph RBD クラスターにバインドされる PVC。*

10. *Pod を作成します。*

```
$ oc create -f ceph-pod1.yaml
pod "ceph-pod1" created
```

11. *Pod が作成されていることを確認します。*

```
$ oc get pod
NAME        READY     STATUS    RESTARTS   AGE
ceph-pod1   1/1       Running   0          2m
```

しばらくすると、Pod のステータスが Running に変わります。

24.4.4. ceph-user-secret をプロジェクトのデフォルトとして設定する方法

全プロジェクトで永続ストレージを使用できるようにするには、デフォルトのプロジェクトテンプレートを変更する必要があります。これをデフォルトのプロジェクトテンプレートに追加すると、プロジェクト作成の権限があるユーザーはすべて Ceph クラスターにアクセスできるようになります。詳細情報は、「デフォルトのプロジェクトテンプレート変更」を参照してください。

デフォルトプロジェクトの例

```
...
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: v1
  kind: Secret
  metadata:
    name: ceph-user-secret
  data:
    key: QVFCbEV4OVpmaGJtQ0JBQW55d2Z0NHZtcS96cE42SW1JVUQvekE9PQ== (1)
  type:
    kubernetes.io/rbd
...
```

(1) base64 形式で Ceph のユーザーキーをここに配置します。

24.5. GLUSTERFS を使用する詳細例

24.5.1. 概要

このトピックでは、既存のコンバージドモード、インデペンデントモードまたはスタンドアロンの Red Hat Gluster Storage クラスターを OpenShift Container Platform の永続ストレージとして使用する詳細例を紹介します。ここでは作業用の Red Hat Gluster Storage クラスターがすでに設定されていることを前提とします。コンバージドモードまたはインデペンデントモードのインストールについてのヘルプは、「Red Hat Gluster Storage を使用する永続ストレージ」を参照してください。スタンドアロンの Red Hat Gluster Storage の場合については、『Red Hat Gluster Storage Administration Guide』を参照してください。

GlusterFS ボリュームを動的にプロビジョニングする方法の詳細例については、「GlusterFS を動的プロビジョニングに使用する詳細例」を参照してください。

注記

oc コマンドはすべて OpenShift Container Platform のマスターホストで実行されます。

24.5.2. 前提条件

GlusterFS ボリュームにアクセスするには、すべてのスケジュール可能なノードで `mount.glusterfs` コマンドを利用できる必要があります。RPM ベースのシステムの場合は、glusterfs-fuse パッケージがインストールされている必要があります。

```
# yum install glusterfs-fuse
```

このパッケージはすべての RHEL システムにインストールされています。ただし、Red Hat Gluster Storage の最新バージョンにアップデートすることを推奨します。そのためには、以下の RPM リポジトリーを有効にする必要があります。

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-
server-rpms
```

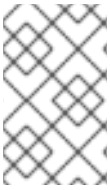
glusterfs-fuse がノードにすでにインストールされている場合、最新バージョンがインストールされていることを確認します。

```
# yum update glusterfs-fuse
```

デフォルトでは、SELinux は Pod からリモート Red Hat Gluster Storage サーバーへの書き込みを許可しません。SELinux が有効な状態で Red Hat Gluster Storage ボリュームへの書き込みを有効にするには、GlusterFS を実行する各ノードで以下のコマンドを実行します。

```
$ sudo setsebool -P virt_sandbox_use_fusefs on (1)
$ sudo setsebool -P virt_use_fusefs on
```

(1) `-P` オプションを使用すると、再起動した後もブール値が永続化されます。



注記

`virt_sandbox_use_fusefs` ブール値は、docker-selinux パッケージによって定義されます。このブール値が定義されていないというエラーが表示される場合は、このパッケージがインストールされていることを確認してください。

注記

Atomic Host を使用している場合に、Atomic Host をアップグレードすると、SELinux のブール値が消去されるので、Atomic Host をアップグレードする場合には、これらのブール値を設定し直す必要があります。

24.5.3. 静的プロビジョニング

1. 静的プロビジョニングを有効にするには、最初に GlusterFS ボリュームを作成します。`gluster` コマンドラインインターフェースを使用してこれを行う方法については、『Red Hat Gluster Storage Administration Guide』を参照してください。また、`heketi-cli` を使用してこれを行う方法については、heketi プロジェクトサイトを参照してください。この例では、ボリュームに `myVol1` という名前を付けます。

2. `gluster-endpoints.yaml` で以下のサービスとエンドポイントを定義します。

```
---
```

```
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster (1)
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster (2)
subsets:
  - addresses:
      - ip: 192.168.122.221 (3)
    ports:
      - port: 1 (4)
  - addresses:
      - ip: 192.168.122.222 (5)
    ports:
      - port: 1 (6)
  - addresses:
      - ip: 192.168.122.223 (7)
    ports:
      - port: 1 (8)
```

(1) (2) これらの名前は一致している必要があります。

(3) (5) (7) ip の値には、Red Hat Gluster Storage サーバーのホスト名ではなく、実際の IP アドレスを指定する必要があります。

(4) (6) (8) ポート番号は無視されます。

3. OpenShift Container Platform マスターホストからサービスとエンドポイントを作成します。

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. サービスとエンドポイントが作成されたことを確認します。

```
$ oc get services
NAME                      CLUSTER_IP       EXTERNAL_IP   PORT(S)
SELECTOR        AGE
glusterfs-cluster         172.30.205.34    <none>        1/TCP
<none>          44s

$ oc get endpoints
NAME                ENDPOINTS
AGE
docker-registry     10.1.0.3:5000
4h
glusterfs-cluster
```

```
192.168.122.221:1,192.168.122.222:1,192.168.122.223:1    11s
kubernetes           172.16.35.3:8443
4d
```

注記

エンドポイントはプロジェクトごとに一意です。GlusterFS にアクセスする各プロジェクトには独自のエンドポイントが必要です。

5. *ボリュームにアクセスするには、ボリューム上のファイルシステムにアクセスできるユーザー ID (UID) またはグループ ID (GID) でコンテナーを実行する必要があります。この情報は以下の方法で取得できます。*

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0    myVol1 (1) (2)
```

(1) *UID は 592 です。*

(2) *GID は 590 です。*

6. *gluster-pv.yaml で以下の PersistentVolume (PV) を定義します。*

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume (1)
  annotations:
    pv.beta.kubernetes.io/gid: "590" (2)
spec:
  capacity:
    storage: 2Gi (3)
  accessModes: (4)
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster (5)
    path: myVol1 (6)
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

(1) *ボリュームの名前です。*

(2) *GlusterFS ボリュームのルートの GID です。*

(3) *このボリュームに割り当てられるストレージの量。*

(4) *accessModes は、PV と PVC を一致させるためのラベルとして使用します。現時点で、それらは現時点ではいずれの形式のアクセス制御も定義しません。*

(5) *以前に作成されたエンドポイントリソースです。*

6 アクセス対象の GlusterFS ボリュームです。

7. OpenShift Container Platform マスターホストから PV を作成します。

```
$ oc create -f gluster-pv.yaml
```

8. PV が作成されたことを確認します。

```
$ oc get pv
NAME                     LABELS    CAPACITY     ACCESSMODES
STATUS      CLAIM     REASON    AGE
gluster-default-volume   <none>    2147483648   RWX
Available                          2s
```

9. gluster-claim.yaml で、新規 PV にバインドする PersistentVolumeClaim (PVC) を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim    1
spec:
  accessModes:
  - ReadWriteMany        2
  resources:
     requests:
       storage: 1Gi      3
```

1 この要求名は、volumes セクションで Pod によって参照されます。

2 PV の accessModes に一致する必要があります。

3 この要求は、1Gi 以上の容量がある PV を検索します。

10. OpenShift Container Platform マスターホストから PVC を作成します。

```
$ oc create -f gluster-claim.yaml
```

11. PV と PVC がバインドされていることを確認します。

```
$ oc get pv
NAME          LABELS    CAPACITY   ACCESSMODES   STATUS      CLAIM
REASON    AGE
gluster-pv    <none>    1Gi        RWX           Available
gluster-claim             37s

$ oc get pvc
NAME            LABELS    STATUS    VOLUME       CAPACITY
ACCESSMODES   AGE
gluster-claim   <none>    Bound     gluster-pv   1Gi        RWX
24s
```

注記

PVC はプロジェクトごとに一意です。GlusterFS ボリュームにアクセスする各プロジェクトには独自の PVC が必要です。PV は単一のプロジェクトにバインドされないため、複数のプロジェクトにまたがる PVC が同じ PV を参照する場合があります。

24.5.4. ストレージの使用

ここまでの時点で、PVC にバインドされる GlusterFS ボリュームを動的に作成しましたので、この PVC を Pod で利用できるようになりました。

1. *Pod オブジェクト定義を以下のように作成します。*

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-pod
  labels:
    name: hello-openshift-pod
spec:
  containers:
  - name: hello-openshift-pod
    image: openshift/hello-openshift
    ports:
    - name: web
      containerPort: 80
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
      readOnly: false
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1 1
```

1 *先の手順で作成した PVC の名前。*

2. *OpenShift Container Platform マスターホストから、以下のように Pod を作成します。*

```
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

3. *Pod を表示します。イメージがまだ存在していない場合はダウンロードする必要があるので、数分かかります。*

```
# oc get pods -o wide
NAME                                READY     STATUS    RESTARTS
AGE       IP                NODE
hello-openshift-pod                 1/1       Running   0
9m        10.38.0.0         node1
```

4. *コンテナーへの* `oc exec` *を実行し、*index.html *ファイルを Pod の* `mountPath` *定義内に作成します。*

```
$ oc exec -ti hello-openshift-pod /bin/sh
$ cd /usr/share/nginx/html
$ echo 'Hello OpenShift!!!' > index.html
$ ls
index.html
$ exit
```

5. ***ここで、Pod の URL に対して curl を実行します。***

```
# curl http://10.38.0.0
Hello OpenShift!!!
```

6. ***Pod を削除してから再作成し、これが出現するまで待機します。***

```
# oc delete pod hello-openshift-pod
pod "hello-openshift-pod" deleted
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
# oc get pods -o wide
NAME                                READY     STATUS    RESTARTS
AGE       IP                NODE
hello-openshift-pod                 1/1       Running   0
9m        10.37.0.0         node1
```

7. ***もう一度 Pod に対して curl を実行します。データは前と同じになりますが、Pod の IP アドレスは変更されている可能性があることに注意してください。***

```
# curl http://10.37.0.0
Hello OpenShift!!!
```

8. ***以下の操作をいずれかのノードで実行して、index.html ファイルが GlusterFS ストレージに書き込まれていることを確認します。***

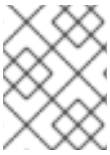
```
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_1e7
30a5462c352835055018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,sw
idth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c
06e606ff4cc29ccb9d018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,sw
idth=512,noquota)

$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c
06e606ff4cc29ccb9d018c73ee292/brick
$ ls
index.html
$ cat index.html
```

```
Hello OpenShift!!!
```

24.6. GLUSTERFS を動的プロビジョニングに使用する詳細例

24.6.1. 概要

このトピックでは、既存のコンバージドモード、インデペンデントモードまたはスタンドアロンの Red Hat Gluster Storage クラスターを OpenShift Container Platform の動的永続ストレージとして使用する詳細例を紹介します。ここでは作業用の Red Hat Gluster Storage クラスターがすでに設定されていることを前提とします。コンバージドモードまたはインデペンデントモードのインストールについてのヘルプは、「Red Hat Gluster Storage を使用する永続ストレージ」を参照してください。スタンドアロンの Red Hat Gluster Storage の場合については、『Red Hat Gluster Storage Administration Guide』を参照してください。



注記

oc コマンドはすべて OpenShift Container Platform のマスターホストで実行されます。

24.6.2. 前提条件

GlusterFS ボリュームにアクセスするには、すべてのスケジュール可能なノードで `mount.glusterfs` コマンドを利用できる必要があります。RPM ベースのシステムの場合は、glusterfs-fuse パッケージがインストールされている必要があります。

```
# yum install glusterfs-fuse
```

このパッケージはすべての RHEL システムにインストールされています。ただし、Red Hat Gluster Storage の最新バージョンにアップデートすることを推奨します。そのためには、以下の RPM リポジトリーを有効にする必要があります。

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-
server-rpms
```

glusterfs-fuse がノードにすでにインストールされている場合、最新バージョンがインストールされていることを確認します。

```
# yum update glusterfs-fuse
```

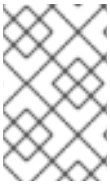
デフォルトでは、SELinux は Pod からリモート Red Hat Gluster Storage サーバーへの書き込みを許可しません。SELinux が有効な状態で Red Hat Gluster Storage ボリュームへの書き込みを有効にするには、GlusterFS を実行する各ノードで以下のコマンドを実行します。

```
$ sudo setsebool -P virt_sandbox_use_fusefs on (1)
$ sudo setsebool -P virt_use_fusefs on
```

(1) -P オプションを使用すると、再起動した後もブール値が永続化されます。

注記

virt_sandbox_use_fusefs ブール値は、*docker-selinux* パッケージによって定義されます。このブール値が定義されていないというエラーが表示される場合は、このパッケージがインストールされていることを確認してください。



注記

Atomic Host を使用している場合に、*Atomic Host* をアップグレードすると、*SELinux* のブール値が消去されるので、*Atomic Host* をアップグレードする場合には、これらのブール値を設定し直す必要があります。

24.6.3. 動的プロビジョニング

1. 動的プロビジョニングを有効にするには、最初に `StorageClass` オブジェクト定義を作成します。以下の定義は、*OpenShift Container Platform* でこの例を使用するために必要な最小要件に基づいています。その他のパラメーターと仕様定義については、「動的プロビジョニングとストレージクラスの作成」を参照してください。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: glusterfs
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://10.42.0.0:8080" (1)
  restauthenabled: "false" (2)
```

(1) *heketi* サーバーの *URL* です。

(2) この例では認証が有効ではないため、`false` に設定します。

2. *OpenShift Container Platform* マスターホストから *StorageClass* を作成します。

```
# oc create -f gluster-storage-class.yaml
storageclass "glusterfs" created
```

3. 新たに作成される *StorageClass* を使用して *PVC* を作成します。以下は例になります。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: gluster1
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
        storage: 30Gi
 storageClassName: glusterfs
```

4. *OpenShift Container Platform* マスターホストから *PVC* を作成します。

```
# oc create -f glusterfs-dyn-pvc.yaml
persistentvolumeclaim "gluster1" created
```

5. PVC を表示し、ボリュームが動的に作成され、PVC にバインドされていることを確認します。

```
# oc get pvc
NAME           STATUS VOLUME
CAPACITY    ACCESSMODES    STORAGECLASS    AGE
gluster1        Bound pvc-78852230-d8e2-11e6-a3fa-0800279cf26f
30Gi     RWX         gluster-dyn 42s
```

24.6.4. ストレージの使用

ここまでの時点で、PVC にバインドされる GlusterFS ボリュームを動的に作成しましたので、この PVC を Pod で利用できるようになりました。

1. Pod オブジェクト定義を以下のように作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-pod
  labels:
    name: hello-openshift-pod
spec:
  containers:
  - name: hello-openshift-pod
    image: openshift/hello-openshift
    ports:
    - name: web
      containerPort: 80
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
      readOnly: false
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1 (1)
```

(1) 先の手順で作成した PVC の名前。

2. OpenShift Container Platform マスターホストから、以下のように Pod を作成します。

```
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

3. Pod を表示します。イメージがまだ存在していない場合はダウンロードする必要があるので、数分かかります。

```
# oc get pods -o wide
NAME                                READY     STATUS     RESTARTS
```

```
AGE       IP              NODE
hello-openshift-pod                       1/1       Running   0
9m        10.38.0.0       node1
```

4. コンテナーへの **oc exec** を実行し、**index.html** ファイルを **Pod** の **mountPath** 定義内に作成します。

```
$ oc exec -ti hello-openshift-pod /bin/sh
$ cd /usr/share/nginx/html
$ echo 'Hello OpenShift!!!' > index.html
$ ls
index.html
$ exit
```

5. ここで、**Pod** の **URL** に対して **curl** を実行します。

```
# curl http://10.38.0.0
Hello OpenShift!!!
```

6. **Pod** を削除してから再作成し、これが出現するまで待機します。

```
# oc delete pod hello-openshift-pod
pod "hello-openshift-pod" deleted
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
# oc get pods -o wide
NAME                                READY     STATUS    RESTARTS
AGE       IP              NODE
hello-openshift-pod                       1/1       Running   0
9m        10.37.0.0       node1
```

7. もう一度 **Pod** に対して **curl** を実行します。データは前と同じになりますが、**Pod** の **IP** アドレスは変更されている可能性があることに注意してください。

```
# curl http://10.37.0.0
Hello OpenShift!!!
```

8. 以下の操作をいずれかのノードで実行して、**index.html** ファイルが **GlusterFS** ストレージに書き込まれていることを確認します。

```
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_1e7
30a5462c352835055018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,sw
idth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c
06e606ff4cc29ccb9d018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,sw
```

```
idth=512,noquota)

$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c
06e606ff4cc29ccb9d018c73ee292/brick
$ ls
index.html
$ cat index.html
Hello OpenShift!!!
```

24.7. 特権付き POD へのボリュームのマウント

24.7.1. 概要

永続ボリュームは、特権付き SCC (Security Context Constraint) が割り当てられた Pod にマウントすることができます。

注記

このトピックでは、特権付き Pod へのボリュームのマウントのユースケースの例として GlusterFS を使用していますが、これはいずれの サポート対象ストレージプラグイン を使用するケースにも適用できます。

24.7.2. 前提条件

- 既存の Gluster ボリューム。
- すべてのホストにインストールされている glusterfs-fuse。
- GlusterFS の定義:
  - エンドポイントとサービス: gluster-endpoints-service.yaml および gluster-endpoints.yaml
  - 永続ボリューム: gluster-pv.yaml
  - Persistent Volume Claim (永続ボリューム要求): gluster-pvc.yaml
  - 特権付き Pod: gluster-S3-pod.yaml
- cluster-admin ロールのバインディングを持つユーザー。このガイドでは、このユーザーを `admin` と呼んでいます。

24.7.3. 永続ボリュームの作成

PersistentVolume を作成すると、プロジェクトに関係なく、ユーザーがストレージにアクセスできるようになります。

1. admin として、サービス、エンドポイントオブジェクトおよび永続ボリュームを作成します。

```
$ oc create -f gluster-endpoints-service.yaml
$ oc create -f gluster-endpoints.yaml
$ oc create -f gluster-pv.yaml
```

2. オブジェクトが作成されたことを以下のように確認します。

```
$ oc get svc
NAME              CLUSTER_IP      EXTERNAL_IP   PORT(S)    SELECTOR
AGE
gluster-cluster   172.30.151.58   <none>        1/TCP      <none>
24s
```

```
$ oc get ep
NAME              ENDPOINTS                                  AGE
gluster-cluster   192.168.59.102:1,192.168.59.103:1   2m
```

```
$ oc get pv
NAME                     LABELS    CAPACITY   ACCESSMODES   STATUS
CLAIM     REASON    AGE
gluster-default-volume   <none>    2Gi        RWX
Available                        2d
```

24.7.4. 通常ユーザーの作成

通常ユーザーを以下のように 特権付き SCC (または SCC へのアクセスのあるグループ) に追加すると、当該ユーザーは特権付き Pod を実行できるようになります。

1. admin として、ユーザーを SCC に追加します。

```
$ oc adm policy add-scc-to-user privileged <ユーザー名>
```

2. 通常ユーザーとしてログインします。

```
$ oc login -u <username> -p <password>
```

3. 次に、新規プロジェクトを作成します。

```
$ oc new-project <project_name>
```

24.7.5. Persistent Volume Claim (永続ボリューム要求) の作成

1. 通常ユーザーとして、ボリュームにアクセスするための PersistentVolumeClaim を作成します。

```
$ oc create -f gluster-pvc.yaml -n <project_name>
```

2. 要求にアクセスするための Pod を定義します。

例24.11 Pod 定義

```
apiVersion: v1
id: gluster-S3-pvc
kind: Pod
metadata:
  name: gluster-nginx-priv
spec:
  containers:
```

```
    - name: gluster-nginx-priv
      image: fedora/nginx
      volumeMounts:
        - mountPath: /mnt/gluster (1)
          name: gluster-volume-claim
      securityContext:
        privileged: true
  volumes:
    - name: gluster-volume-claim
      persistentVolumeClaim:
        claimName: gluster-claim (2)
```

(1) Pod 内のボリュームマウント。

(2) gluster-claim には PersistentVolume の名前を反映させる必要があります。

3. Pod を作成するとマウントディレクトリーが作成され、ボリュームがそのマウントポイントに割り当てられます。
   通常ユーザーとして、以下のように定義から Pod を作成します。

```
$ oc create -f gluster-S3-pod.yaml
```

4. Pod が正常に作成されたことを確認します。

```
$ oc get pods
NAME                READY     STATUS    RESTARTS   AGE
gluster-S3-pod   1/1       Running   0          36m
```

   Pod が作成されるまでに数分の時間がかかることがあります。

24.7.6. 設定の検証

24.7.6.1. Pod の SCC の確認

1. Pod 設定をエクスポートします。

```
$ oc export pod <Pod 名>
```

2. 出力を確認します。openshift.io/scc の値が privileged であることを確認します。

   例24.12 スニペットのエクスポート

```
metadata:
  annotations:
    openshift.io/scc: privileged
```

24.7.6.2. マウントの検証

1. Pod にアクセスし、ボリュームがマウントされていることを確認します。

```
$ oc rsh <pod_name>
[root@gluster-S3-pvc /]# mount
```

2. Gluster ボリュームの出力結果を確認します。

例24.13 ボリュームマウント

```
192.168.59.102:gv0 on /mnt/gluster type fuse.gluster
(rw,relatime,user_id=0,group_id=0,default_permissions,allow_other,
max_read=131072)
```

24.8. 統合 OPENSHIFT CONTAINER レジストリーから GLUSTERFS への切り替え

24.8.1. 概要

このトピックでは、GlusterFS ボリュームを統合 OpenShift Container レジストリーに割り当てる方法を説明します。この操作は、コンバージドモード、インデペンデントモード、またはスタンドアロンの Red Hat Gluster Storage のいずれかで実行できます。ここでは、レジストリーがすでに起動されていて、ボリュームが作成済みであることを前提とします。

24.8.2. 前提条件

- ストレージを設定せずにデプロイされている既存の レジストリー。
- 既存の GlusterFS ボリューム。
- すべてのスケジュール可能なノードにインストールされている glusterfs-fuse。
- cluster-admin ロールのバインディングを持つユーザー。
  - このガイドでは、このユーザーを admin と呼んでいます。

注記

oc コマンドはすべてマスターノードで admin ユーザーとして実行されます。

24.8.3. GlusterFS PersistentVolumeClaim の手動プロビジョニング

1. 静的プロビジョニングを有効にするには、最初に GlusterFS ボリュームを作成します。gluster コマンドラインインターフェースを使用してこれを行う方法については、『Red Hat Gluster Storage Administration Guide』を参照してください。また、heketi-cli を使用してこれを行う方法については、heketi プロジェクトサイトを参照してください。この例では、ボリュームに myVol1 という名前を付けます。

2. gluster-endpoints.yaml で以下のサービスとエンドポイントを定義します。

```
---
apiVersion: v1
kind: Service
metadata:
```

```
  name: glusterfs-cluster (1)
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster (2)
subsets:
  - addresses:
      - ip: 192.168.122.221 (3)
    ports:
      - port: 1 (4)
  - addresses:
      - ip: 192.168.122.222 (5)
    ports:
      - port: 1 (6)
  - addresses:
      - ip: 192.168.122.223 (7)
    ports:
      - port: 1 (8)
```

(1) (2) これらの名前は一致している必要があります。

(3) (5) (7) ip の値には、Red Hat Gluster Storage サーバーのホスト名ではなく、実際の IP アドレスを指定する必要があります。

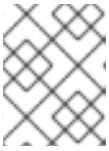
(4) (6) (8) ポート番号は無視されます。

3. OpenShift Container Platform マスターホストからサービスとエンドポイントを作成します。

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. サービスとエンドポイントが作成されたことを確認します。

```
$ oc get services
NAME                       CLUSTER_IP       EXTERNAL_IP   PORT(S)
SELECTOR        AGE
glusterfs-cluster          172.30.205.34    <none>        1/TCP
<none>          44s

$ oc get endpoints
NAME                ENDPOINTS
AGE
docker-registry     10.1.0.3:5000
4h
glusterfs-cluster
192.168.122.221:1,192.168.122.222:1,192.168.122.223:1   11s
kubernetes          172.16.35.3:8443
4d
```



注記

エンドポイントはプロジェクトごとに一意です。GlusterFS にアクセスする各プロジェクトには独自のエンドポイントが必要です。

5. *ボリュームにアクセスするには、ボリューム上のファイルシステムにアクセスできるユーザー ID (UID) またはグループ ID (GID) でコンテナーを実行する必要があります。この情報は以下の方法で取得できます。*

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0    myVol1 (1) (2)
```

(1) *UID は 592 です。*

(2) *GID は 590 です。*

6. *gluster-pv.yaml で以下の PersistentVolume (PV) を定義します。*

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume (1)
  annotations:
    pv.beta.kubernetes.io/gid: "590" (2)
spec:
  capacity:
    storage: 2Gi (3)
  accessModes: (4)
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster (5)
    path: myVol1 (6)
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

(1) *ボリュームの名前です。*

(2) *GlusterFS ボリュームのルートの GID です。*

(3) *このボリュームに割り当てられるストレージの量。*

(4) *accessModes は、PV と PVC を一致させるためのラベルとして使用します。現時点で、それらは現時点ではいずれの形式のアクセス制御も定義しません。*

(5) *以前に作成されたエンドポイントリソースです。*

(6) *アクセス対象の GlusterFS ボリュームです。*

7. *OpenShift Container Platform マスターホストから PV を作成します。*

```
$ oc create -f gluster-pv.yaml
```

8. *PV が作成されたことを確認します。*

```
$ oc get pv
NAME                     LABELS     CAPACITY      ACCESSMODES
STATUS      CLAIM     REASON     AGE
gluster-default-volume   <none>     2147483648    RWX
Available                          2s
```

9. *gluster-claim.yaml で、新規 PV にバインドする PersistentVolumeClaim (PVC) を作成します。*

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim    (1)
spec:
  accessModes:
  - ReadWriteMany        (2)
  resources:
     requests:
       storage: 1Gi      (3)
```

(1) *この要求名は、volumes セクションで Pod によって参照されます。*

(2) *PV の accessModes に一致する必要があります。*

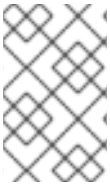
(3) *この要求は、1Gi 以上の容量がある PV を検索します。*

10. *OpenShift Container Platform マスターホストから PVC を作成します。*

```
$ oc create -f gluster-claim.yaml
```

11. *PV と PVC がバインドされていることを確認します。*

```
$ oc get pv
NAME          LABELS     CAPACITY    ACCESSMODES    STATUS       CLAIM
REASON     AGE
gluster-pv    <none>     1Gi         RWX            Available
gluster-claim             37s

$ oc get pvc
NAME             LABELS     STATUS     VOLUME         CAPACITY
ACCESSMODES    AGE
gluster-claim    <none>     Bound      gluster-pv     1Gi         RWX
24s
```



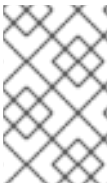
注記

PVC はプロジェクトごとに一意です。GlusterFS ボリュームにアクセスする各プロジェクトには独自の PVC が必要です。PV は単一のプロジェクトにバインドされないため、複数のプロジェクトにまたがる PVC が同じ PV を参照する場合があります。

24.8.4. PersistentVolumeClaim のレジストリーへの割り当て

次に進む前に、docker-registry サービスが実行中であることを確認します。

```
$ oc get svc
NAME              CLUSTER_IP       EXTERNAL_IP   PORT(S)
SELECTOR                  AGE
docker-registry   172.30.167.194   <none>        5000/TCP
docker-registry=default   18m
```



注記

docker-registry サービス、またはそれに関連付けられている Pod のいずれかが実行されていない場合は、レジストリーの設定手順を再度参照してトラブルシューティングを行ってから次に進んでください。

次に、PVC を割り当てます。

```
$ oc volume deploymentconfigs/docker-registry --add --name=registry-
storage -t pvc \
      --claim-name=gluster-claim --overwrite
```

OpenShift Container レジストリーの使用についての詳細は、「レジストリーのセットアップ」を参照してください。

24.9. ラベルによる永続ボリュームのバインド

24.9.1. 概要

このトピックでは、PV でラベルを定義して PVC 内でセレクターを一致させることで Persistent Volume Claim (永続ボリューム要求、PVC) を 永続ボリューム (PV) にバインドする詳細例を紹介します。この機能はすべてのストレージオプションで使用できます。ここでは、OpenShift Container Platform クラスターに永続ストレージリソースが含まれていて、それらのリソースを PVC によるバインディングに使用できることを前提としています。

ラベルとセレクターに関する注記

ラベルは OpenShift Container Platform の機能であり、ユーザー定義のタグ (キーと値のペア) をオブジェクトの仕様の一部としてサポートします。その主な目的は、オブジェクト間で同一ラベルを定義してオブジェクトを任意にグループ化できるようにすることです。定義したラベルをセレクターでターゲットとして指定すると、指定のラベル値を持つすべてのオブジェクトが一致します。この機能により、PVC を PV にバインドすることができます。ラベルについての詳細は、「Pods and Services」を参照してください。

注記

この例では、変更された GlusterFS の PV および PVC 仕様を使用しています。ただし、実装したセレクターとラベルはすべてのストレージオプションで汎用的に使用できます。使用しているボリュームプロバイダーの独自の設定については、「関連するストレージオプション」を参照してください。

24.9.1.1. 前提条件

以下があることを前提とします。

- *少なくとも 1 つのマスターと 1 つのノードがある既存の OpenShift Container Platform クラスター*
- *少なくとも 1 つのサポート対象ストレージボリューム*
- *cluster-admin 権限を持つユーザー*

24.9.2. 仕様の定義

注記

ここでの仕様は GlusterFS に合わせてカスタマイズされています。使用しているボリュームプロバイダーの独自の設定については、「関連するストレージオプション」を参照してください。

24.9.2.1. ラベルのある永続ボリューム

例24.14 glusterfs-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
  labels: 1
    storage-tier: gold
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster 2
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

1 *ラベルを使用して、ボリューム間で共有している共通の属性や特性を識別します。この例では、Gluster ボリュームを定義し、storage-tier という名前のカスタム属性 (キー) を持たせて、gold という値を割り当てています。要求で `storage-tier=gold` を使用して PV を選択すると、その PV に一致します。*

2

エンドポイントは、Gluster で信頼されるプールを定義します。これについては以下で説明します。

24.9.2.2. セレクターのある Persistent Volume Claim (永続ボリューム要求)

selector スタンザのある要求 (以下の例を参照してください) は、事前にバインドされていない既存の非要求の PV との一致を試みます。PVC セレクターがあるため、PV の容量は無視されます。ただし、accessModes は一致条件において考慮されます。

要求はその selector スタンザに含まれるすべてのキーと値のペアに一致する必要がある、ということに注意してください。要求に一致する PV がない場合、PVC はバインドされない (保留中の) ままになります。PV はその後に作成され、要求によってラベルの一致の有無が自動的にチェックされます。

例24.15 glusterfs-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
  selector: 1
    matchLabels:
      storage-tier: gold
      aws-availability-zone: us-east-1
```

1 selector スタンザでは、この要求と一致させるために PV で必要なすべてのラベルを定義します。

24.9.2.3. ボリュームエンドポイント

PV を Gluster ボリュームに割り当てるには、オブジェクトを作成する前にエンドポイントを設定する必要があります。

例24.16 glusterfs-ep.yaml

```
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster
subsets:
  - addresses:
      - ip: 192.168.122.221
    ports:
      - port: 1
  - addresses:
```

```
      - ip: 192.168.122.222
    ports:
      - port: 1
```

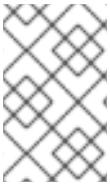
24.9.2.4. PV、PVC、およびエンドポイントのデプロイ

この例では、oc コマンドを cluster-admin 権限のあるユーザーとして実行します。実稼働環境では、クラスタークライアントが PVC の定義と作成を行うことなどが予想されます。

```
# oc create -f glusterfs-ep.yaml
endpoints "glusterfs-cluster" created
# oc create -f glusterfs-pv.yaml
persistentvolume "gluster-volume" created
# oc create -f glusterfs-pvc.yaml
persistentvolumeclaim "gluster-claim" created
```

最後に、PV と PVC が正常にバインドされていることを確認します。

```
# oc get pv,pvc
NAME             CAPACITY   ACCESSMODES      STATUS     CLAIM
REASON    AGE
gluster-volume   2Gi        RWX              Bound      gfs-
trial/gluster-claim            7s
NAME             STATUS     VOLUME           CAPACITY   ACCESSMODES
AGE
gluster-claim    Bound      gluster-volume   2Gi        RWX
7s
```



注記

PVC はプロジェクトに対してローカルですが、PV はクラスター全体にわたるグローバルリソースです。開発者および管理者以外のユーザーは、使用可能な PV のすべて (またはいずれか) にアクセスできない場合があります。

24.10. ストレージクラスを使用した動的プロビジョニング

24.10.1. 概要

この例では、StorageClass のさまざまな設定と Google Cloud Platform Compute Engine (GCE) を使用した動的プロビジョニングについて、いくつかのシナリオを紹介します。これらの例では、Kubernetes、GCE、および永続ディスクについて理解していること、また OpenShift Container Platform がインストールされていて GCE を使用できるよう適切に設定されていることを前提とします。

- 基本的な動的プロビジョニング
- クラスターの動的プロビジョニング動作のデフォルト設定

24.10.2. シナリオ 1: 2 種類の StorageClass を持つ基本的な動的プロビジョニング

StorageClass を使用すると、ストレージのレベルや使用状況を区別し、記述することができます。この例では、cluster-admin または storage-admin が GCE で 2 つの異なるストレージのクラスを設定します。

- slow: 低コストで効率的なシーケンシャルデータの操作に最適化されている (低速読み取り/書き込み)
- fast: 高速なランダム IOPS と持続的スループットに最適化されている (高速読み取り/書き込み)

これらの StorageClass を作成することで、cluster-admin または storage-admin はユーザーに対して、StorageClass の特定のレベルまたはサービスについての要求の作成を許可することができます。

例24.17 StorageClass 低速オブジェクトの定義

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow (1)
provisioner: kubernetes.io/gce-pd (2)
parameters:
  type: pd-standard (3)
  zone: us-east1-d (4)
```

1. StorageClass の名前。
2. 使用するプロビジョナープラグイン。StorageClass の必須フィールドです。
3. PD のタイプ。この例では pd-standard を使用しています。このタイプは、持続的 IOPS とスループットが高い pd-ssd と比べていくらかコストを下げられる一方、持続的 IOPS の速度やスループットもわずかに低下します。
4. ゾーンは必須です。

例24.18 StorageClass 高速オブジェクトの定義

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zone: us-east1-d
```

cluster-admin または storage-admin として、両方の定義を YAML ファイル (slow-gce.yaml と fast-gce.yaml など) に保存します。次に StorageClass を作成します。

```
# oc create -f slow-gce.yaml
storageclass "slow" created
```

```
# oc create -f fast-gce.yaml
storageclass "fast" created

# oc get storageclass
NAME        TYPE
fast        kubernetes.io/gce-pd
slow        kubernetes.io/gce-pd
```

重要

cluster-admin ユーザーまたは storage-admin ユーザーは、適切な StorageClass 名を適切なユーザー、グループ、およびプロジェクトに送る必要があります。

通常ユーザーとして、以下のように新規プロジェクトを作成します。

```
# oc new-project rh-eng
```

要求の YAML 定義を作成し、これをファイル (pvc-fast.yaml) に保存します。

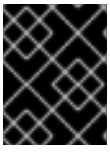
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 10Gi
 storageClassName: fast
```

oc create コマンドを使用して要求を追加します。

```
# oc create -f pvc-fast.yaml
persistentvolumeclaim "pvc-engineering" created
```

要求がバインドされているかどうかをチェックします。

```
# oc get pvc
NAME              STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
pvc-engineering   Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi       RWX           2m
```



重要

この要求は rh-eng プロジェクトで作成され、バインドされているため、同じプロジェクトのいずれのユーザーにも共有できます。

cluster-admin ユーザーまたは storage-admin ユーザーとして、最近動的にプロビジョニングした永続ボリューム (PV) を表示します。

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS    CLAIM                          REASON    AGE
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX
Delete          Bound     rh-eng/pvc-engineering                   5m
```

重要

動的にプロビジョニングされたすべてのボリュームについて、RECLAIMPOLICY がデフォルトで Delete になっていることに注意してください。これは、ボリュームが要求がシステムに存在している間存続することを意味します。要求を削除するとボリュームも削除され、ボリュームのすべてのデータが失われます。

最後に GCE コンソールをチェックします。新規のディスクが作成され、使用できる状態になります。

```
kubernetes-dynamic-pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  SSD
persistent disk  10 GB  us-east1-d
```

これで、Pod で Persistent Volume Claim (永続ボリューム要求) を参照し、ボリュームの使用を開始することができます。

24.10.3. シナリオ 2: クラスターにおけるデフォルトの StorageClass の動作を有効にする方法

この例では、`cluster-admin` または `storage-admin` により、StorageClass を要求に暗黙的に指定していない他のすべてのユーザーおよびプロジェクトについてデフォルトのストレージクラスが有効になります。この方法は、特化した StorageClass をクラスター全体に設定し、伝達することなしに `cluster-admin` または `storage-admin` がストレージボリュームを容易に管理できるようにする場合に役に立ちます。

以下の例は 「シナリオ 1: 2 種類の StorageClass を持つ基本的な動的プロビジョニング」 を基づいて作成されています。`cluster-admin` または `storage-admin` は、デフォルトの StorageClass として指定するための別の StorageClass を作成します。

例24.19 デフォルトの StorageClass オブジェクトの定義

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: generic 1
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" 2
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  zone: us-east1-d
```

1 StorageClass の名前。クラスター内で一意にする必要があります。

2 この StorageClass にデフォルトクラスのマークを付けるアノテーション。このバージョンの API では引用符付きの `"true"` を使用する必要があります。このアノテーションがない場合、OpenShift Container Platform はこれをデフォルトの StorageClass ではないと見なします。

```
```

cluster-admin または storage-admin として、この定義を YAML ファイル (generic-gce.yaml) に保存し、StorageClass を作成します。

```
# oc create -f generic-gce.yaml
storageclass "generic" created

# oc get storageclass
NAME        TYPE
generic     kubernetes.io/gce-pd
fast        kubernetes.io/gce-pd
slow        kubernetes.io/gce-pd
```

通常ユーザーとして、StorageClass の要件なしに新規の要求定義を作成し、これをファイル (generic-pvc.yaml) に保存します。

例24.20 デフォルトのストレージ要求オブジェクトの定義

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering2
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 5Gi
```

以下のように実行し、要求がバインドされていることをチェックします。

```
# oc create -f generic-pvc.yaml
persistentvolumeclaim "pvc-engineering2" created
                                                                    3s
# oc get pvc
NAME                STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
pvc-engineering     Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi       RWX           41m
pvc-engineering2    Bound     pvc-a9f70544-8bfd-11e6-9962-42010af00004
5Gi        RWX           7s   1
```

1 **pvc-engineering2 は、デフォルトで、動的にプロビジョニングされたボリュームにバインドされます。**

cluster-admin または storage-admin として、ここまでに定義した永続ボリュームを表示します。

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS    CLAIM                          REASON    AGE
```

```
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX
Delete          Bound     rh-eng/pvc-engineering2              5m  (1)
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO
Delete          Bound     mytest/gce-dyn-claim1                21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX
Delete          Bound     rh-eng/pvc-engineering               46m  (2)
```

(1) *この PV は、デフォルトの StorageClass からデフォルトの動的ボリュームにバインドされています。*

(2) *この PV は 高速 StorageClass を使用して「シナリオ 1: 2 種類の StorageClass を持つ基本的な動的プロビジョニング」の 1 番目の PVC にバインドされています。*

GCE を使用して (動的にプロビジョニングされるのではなく) 手動でプロビジョニングされるディスクを作成します。次に、新規の GCE ディスク (pv-manual-gce.yaml) に接続する永続ボリュームを作成します。

例24.21 手動の PV オブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-manual-gce
spec:
 capacity:
   storage: 35Gi
 accessModes:
   - ReadWriteMany
 gcePersistentDisk:
   readOnly: false
   pdName: the-newly-created-gce-PD
   fsType: ext4
```

オブジェクト定義ファイルを実行します。

```
# oc create -f pv-manual-gce.yaml
```

ここでもう一度 PV を表示します。pv-manual-gce ボリュームが Available になっていることに留意してください。

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS      CLAIM                      REASON     AGE
pv-manual-gce                              35Gi       RWX
Retain          Available                                         4s
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX
Delete          Bound       rh-eng/pvc-engineering2              12m
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO
Delete          Bound       mytest/gce-dyn-claim1                21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX
Delete          Bound       rh-eng/pvc-engineering               53m
```

今度は generic-pvc.yaml PVC 定義と同一の別の要求を作成しますが、名前を変更し、ストレージクラス名は設定しません。

例24.22 要求オブジェクトの定義

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-engineering3
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 15Gi
```

このインスタンスではデフォルトの StorageClass が有効になっているため、手動で作成された PV は要求のリクエストに対応せず、ユーザーは新たに動的にプロビジョニングされた永続ボリュームを受け取ることになります。

```
# oc get pvc
NAME                STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
pvc-engineering     Bound     pvc-e9b4fef7-8bf7-11e6-9962-42010af00004
10Gi       RWX           1h
pvc-engineering2    Bound     pvc-a9f70544-8bfd-11e6-9962-42010af00004
5Gi        RWX           19m
pvc-engineering3    Bound     pvc-6fa8e73b-8c00-11e6-9962-42010af00004
15Gi       RWX           6s
```

重要

デフォルトの StorageClass がこのシステムで有効になっているため、手動で作成された永続ボリュームを前述の要求によってバインドし、動的にプロビジョニングされた新規ボリュームがバインドされないようにするには、PV をデフォルトの StorageClass で作成しておく必要があります。

デフォルトの StorageClass がこのシステムで有効になっているため、手動で作成された永続ボリュームを前述の要求によってバインドし、動的にプロビジョニングされた新規ボリュームをバインドしないようにするためには、PV をデフォルトの StorageClass で作成しておく必要があります。

これを解決するには、cluster-admin ユーザーまたは storage-admin ユーザーは、別の GCE ディスクを作成するか、または必要に応じて最初の手動の PV を削除し、StorageClass 名を割り当てる PV オブジェクト定義 (pv-manual-gce2.yaml) を使用することのみが必要になります。

例24.23 デフォルトの StorageClass 名を持つ手動 PV の仕様

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-manual-gce2
spec:
```

```
  capacity:
    storage: 35Gi
  accessModes:
    - ReadWriteMany
  gcePersistentDisk:
    readOnly: false
    pdName: the-newly-created-gce-PD
    fsType: ext4
  storageClassName: generic (1)
```

(1) 先に作成した汎用 の StorageClass の名前。

オブジェクト定義ファイルを実行します。

```
# oc create -f pv-manual-gce2.yaml
```

PV を一覧表示します。

```
# oc get pv
NAME                                       CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS      CLAIM                                REASON     AGE
pv-manual-gce                              35Gi       RWX
Retain          Available                                                   4s (1)
pv-manual-gce2                             35Gi       RWX
Retain          Bound       rh-eng/pvc-engineering3                         4s (2)
pvc-a9f70544-8bfd-11e6-9962-42010af00004   5Gi        RWX
Delete          Bound       rh-eng/pvc-engineering2                         12m
pvc-ba4612ce-8b4d-11e6-9962-42010af00004   5Gi        RWO
Delete          Bound       mytest/gce-dyn-claim1                           21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004   10Gi       RWX
Delete          Bound       rh-eng/pvc-engineering                          53m
```

(1) 元の手動 PV はまだバインドされておらず、Available です。これは、default StorageClass で作成されていないためです。

(2) 2 番目の PVC (名前以外) は手動で作成された Available の PV である pv-manual-gce2 にバインドされています。

重要

動的にプロビジョニングされたすべてのボリュームの RECLAIMPOLICY がデフォルトで Delete である点に注目してください。PV に動的にバインドされた PVC が削除されると、GCE ボリュームが削除され、すべてのデータが消失します。ただし、手動で作成された PV の RECLAIMPOLICY はデフォルトで Retain です。

24.11. 既存のレガシーストレージに対するストレージクラスの使用

24.11.1. 概要

この例では、レガシーデータボリュームが存在し、cluster-admin または storage-admin がその

ボリュームを特定のプロジェクトで使用できるようにする必要があります。StorageClass を使用すると、他のユーザーおよびプロジェクトがこのボリュームへのアクセスを要求から取得する可能性が低くなります。これは、要求には完全一致する StorageClass 名の値が必要になるためです。また、ここでは動的なプロビジョニングも無効にしています。この例では以下の要件を満たしていることを前提としています。

- OpenShift Container Platform、GCE、および永続ディスクについてある程度理解している。
- OpenShift Container Platform が GCE を使用するように適切に設定されている。

24.11.1.1. シナリオ 1: レガシーデータを含む既存の永続ボリュームに StorageClass をリンクさせる

`cluster-admin` または `storage-admin` として、過去の財務データ用の StorageClass を定義し、作成します。

例24.24 StorageClass の finance-history オブジェクトの定義

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: finance-history (1)
provisioner: no-provisioning (2)
parameters: (3)
```

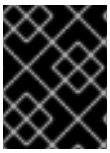
(1) StorageClass の名前。

(2) これは必須フィールドです。しかし、動的なプロビジョニングは行われないため、実際のプロビジョナーのプラグインタイプでない限り、このフィールドに値を指定する必要があります。

(3) パラメーターは動的プロビジョナーで使用されるだけなので、空白のままにしておくことができます。

この定義を YAML ファイル (`finance-history-storageclass.yaml`) に保存して、StorageClass を作成します。

```
# oc create -f finance-history-storageclass.yaml
storageclass "finance-history" created


# oc get storageclass
NAME                TYPE
finance-history     no-provisioning
```



重要

`cluster-admin` ユーザーまたは `storage-admin` ユーザーは、適切な StorageClass 名を適切なユーザー、グループ、およびプロジェクトに送る必要があります。

StorageClass が存在します。`cluster-admin` または `storage-admin` は StorageClass で使用するための永続ボリューム (PV) を作成することができます。(動的にプロビジョニングされない) GCE およ

び新しい GCE ディスク (gce-pv.yaml) に接続される永続ボリュームを使用して、手動でプロビジョニングされたディスクを作成します。

例24.25 財務履歴の PV オブジェクト

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-finance-history
spec:
 capacity:
   storage: 35Gi
 accessModes:
   - ReadWriteMany
 gcePersistentDisk:
   readOnly: false
   pdName: the-existing-PD-volume-name-that-contains-the-valuable-data
(1)
   fsType: ext4
 storageClassName: finance-history (2)
```

(2) StorageClass 名。完全一致している必要があります。

(1) すでに存在し、レガシーデータが含まれている GCE ディスクの名前。

cluster-admin または storage-admin として、PV を作成し、これを表示します。

```
# oc create -f gce-pv.yaml
persistentvolume "pv-finance-history" created

# oc get pv
NAME                CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM                       REASON    AGE
pv-finance-history   35Gi       RWX           Retain          Available
2d
```

pv-finance-history が Available で、いつでも利用可能であることに留意してください。

ユーザーとして、Persistent Volume Claim (永続ボリューム要求、PVC) を YAML ファイルとして作成し、以下のように適切な StorageClass 名を指定します。

例24.26 finance-history オブジェクト定義の要求

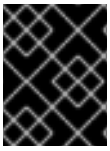
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-finance-history
spec:
 accessModes:
  - ReadWriteMany
 resources:
```

```
    requests:
      storage: 20Gi
  storageClassName: finance-history (1)
```

(1) StorageClass 名。完全一致している必要があります。そうでない場合には、削除されるか、または名前が一致する別の StorageClass が作成されるまで要求が非バインドの状態になります。

PVC と PV を作成および表示して、バインドされているか確認します。

```
# oc create -f pvc-finance-history.yaml
persistentvolumeclaim "pvc-finance-history" created

# oc get pvc
NAME                   STATUS    VOLUME                CAPACITY
ACCESSMODES   AGE
pvc-finance-history    Bound     pv-finance-history    35Gi       RWX
9m


# oc get pv  (cluster/storage-admin)
NAME                   CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM                           REASON    AGE
pv-finance-history     35Gi       RWX           Retain          Bound
default/pvc-finance-history                  5m
```



重要

同じクラスター内の StorageClass を、レガシーデータ (動的プロビジョニングなし) および動的プロビジョニングの両方に対して使用することができます。

24.12. AZURE BLOB STORAGE での統合 DOCKER レジストリーの設定

24.12.1. 概要

このトピックでは、Microsoft Azure Blob Storage で OpenShift 統合 Docker レジストリー を設定する方法を説明します。

24.12.2. 作業を開始する前に

- Microsoft Azure Portal、Microsoft Azure CLI、または Microsoft Azure Storage Explorer を使用してストレージコンテナーを作成します。ストレージアカウント名、ストレージアカウントキー、およびコンテナー名をメモしてください。
- デプロイされていない場合は、統合 Docker レジストリーをデプロイします。

24.12.3. レジストリー設定の上書き

新規レジストリー Pod を作成し、古い Pod と自動的に置き換えるには、以下の手順を実行します。

1. registryconfig.yaml という名前の新規レジストリー設定ファイルを作成して、以下の情報を追加します。

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
  azure: (1)
    accountname: azureblobacc
    accountkey:  azureblobacckey
    container: azureblobname
    realm: core.windows.net (2)
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
      options:
        acceptschema2: false
        pullthrough: true
        enforcequota: false
        projectcachettl: 1m
        blobrepositorycachettl: 10m
  storage:
    - name: openshift
```

(1) accountname、acountkey、および container の値をそれぞれストレージアカウント名、ストレージアカウントキー、およびストレージコンテナー名で置き換えます。

(2) Azure のリージョンクラウドを使用している場合は、必要なレルムに設定します。たとえば、ドイツのリージョンクラウドの場合は core.cloudapi.de に設定します。

2. 新規レジストリー設定を作成します。

```
$ oc create secret generic registry-config --from-
file=config.yaml=registryconfig.yaml
```

3. シークレットを追加します。

```
$ oc volume dc/docker-registry --add --type=secret \
    --secret-name=registry-config -m /etc/docker/registry/
```

4. REGISTRY_CONFIGURATION_PATH 環境変数を設定します。

```
$ oc set env dc/docker-registry \
    REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yaml
```

5. *レジストリー設定をすでに作成している場合は、以下の手順を実行します。*

   a. *シークレットを削除します。*

   ```
   $ oc delete secret registry-config
   ```

   b. *新規レジストリー設定を作成します。*

   ```
   $ oc create secret generic registry-config --from-
   file=config.yaml=registryconfig.yaml
   ```

   c. *新規ロールアウトを開始して設定を更新します。*

   ```
   $ oc rollout latest docker-registry
   ```

第25章 一時ストレージの設定

25.1. 概要

OpenShift Container Platform は、Pod およびコンテナーの作業データの一時ストレージを管理できるように、設定可能です。コンテナーは、記述可能な階層、ログディレクトリー、EmptyDir ボリュームを活用できる反面、このストレージには複数の制限があります。この点については、こちらを参照してください。

一時ストレージ管理により、管理者は個別の Pod やコンテナーが消費するリソースを制限できるようになるだけでなく、Pod およびコンテナーが対象の一時ストレージの使用を制限したり、要求したりできます。これは、OpenShift Container Platform 3.10 のテクノロジープレビュー機能で、デフォルトでは無効になっています。

注記

このテクノロジープレビュー機能は、OpenShift Container Platform でローカルストレージを公開するメカニズムを変更するわけではなく、既存のメカニズム、root ディレクトリーや runtime ディレクトリーはそのまま使用されます。このテクノロジープレビュー機能は、当リソースの使用の管理メカニズムを提供するだけです。

25.2. 一時ストレージの有効化

一時ストレージを有効化するには、以下を実行します。

1. すべてのマスターで、マスター設定ファイル (デフォルトは /etc/origin/master/master-config.yaml) を編集するか、または作成して `LocalStorageCapacityIsolation=true` を `apiServerArguments` と `controllerArguments` の各セクションに追加します。

```
apiServerArguments:
   feature-gates:
   - LocalStorageCapacityIsolation=true
...

 controllerArguments:
   feature-gates:
   - LocalStorageCapacityIsolation=true
...
```

2. コマンドラインで、全ノードの ConfigMap を編集して、LocalStorageCapacityIsolation を有効化します。編集の必要のある ConfigMaps は以下のように特定します。

```
$ oc get cm -n openshift-node
NAME                   DATA      AGE
node-config-compute    1         52m
node-config-infra      1         52m
node-config-master     1         52m
```

`node-config-compute`、`node-config-infra` および `node-config-master` のマップごとに、機能ゲートを追加する必要があります。

```
oc edit cm node-config-master -n openshift-node
```

すでに `feature-gates:` の宣言がある場合には、以下のテキストを機能ゲートの一覧に追加します。

```
,LocalStorageCapacityIsolation=true
```

`feature-gates:` の宣言がない場合には、以下のセクションを追加します。

```
  feature-gates:
  - LocalStorageCapacityIsolation=true
```

3. `node-config-compute`、`node-config-infra` をはじめ、他の ConfigMaps についても同様に繰り返します。
4. OpenShift Container Platform を再起動して、apiserver を実行するコンテナーを削除します。

注記

これらの手順を省略すると、一時ストレージの管理が有効化されなくなる可能性があります。

第26章 HTTP プロキシーの使用

26.1. 概要

実稼働環境では、インターネットへの直接アクセスを拒否し、代わりに HTTP または HTTPS プロキシーを使用することができます。これらのプロキシーを使用するように OpenShift Container Platform を設定することは、設定ファイルまたは JSON ファイルで標準的な環境変数を設定するのと同じくらい簡単に実行できます。この設定は、クラスターインストール 時に実行するか、またはインストール後に行うことができます。

プロキシー設定はクラスター内の各ホストで同じである必要があります。したがって、プロキシーの設定やその変更を行う場合は、各 OpenShift Container Platform ホスト上のファイルを更新して同じ値にする必要があります。その後はクラスター内の各ホストで OpenShift Container Platform サービスを再起動する必要があります。

`NO_PROXY`、`HTTP_PROXY`、および `HTTPS_PROXY` 環境変数は各ホストの /etc/sysconfig/atomic-openshift-master-api または /etc/sysconfig/atomic-openshift-master-controllers ファイルと /etc/sysconfig/atomic-openshift-node ファイルにあります。

26.2. NO_PROXY の設定

`NO_PROXY` 環境変数は、OpenShift Container Platform のすべてのコンポーネントと、OpenShift Container Platform によって管理されるすべての IP アドレスを一覧表示します。

CIDR を許可する OpenShift サービスでは、以下のように `NO_PROXY` にホスト、IP アドレス、または IP 範囲 (CIDR 形式) のコンマ区切りの一覧を指定できます。

マスターホストの場合

- ノードホスト名
- マスター IP またはホスト名
- etcd ホストの IP アドレス

ノードホストの場合

- マスター IP またはホスト名

Docker サービスの場合

- レジストリーのサービス IP とホスト名
- レジストリーサービス URL `docker-registry.default.svc.cluster.local`
- レジストリールートホスト名 (作成されている場合)

注記

Docker を使用している場合は、Docker はホスト、ドメイン拡張子、または IP アドレスのコンマ区切りの一覧を受け入れますが、CIDR 形式の IP 範囲は受け入れません。CIDR 形式の IP 範囲を受け入れるのは OpenShift サービスのみです。`no_proxy' 変数には、プロキシーを使用すべきでない ドメイン拡張子のコンマ区切りの一覧を含める必要があります。

たとえば、no_proxy が .school.edu に設定されている場合は、特定の学校からドキュメントを取得するためにプロキシーが使用されることはありません。

*NO_PROXY には、*master-config.yaml *ファイルにあるように、SDN ネットワークとサービス IP アドレスも含まれています。*

/etc/origin/master/master-config.yaml

```
networkConfig:
  clusterNetworks:
  - cidr: 10.1.0.0/16
    hostSubnetLength: 9
  serviceNetworkCIDR: 172.30.0.0/16
```

*OpenShift Container Platform では、ドメインサフィックスに付加するワイルドカードとして * を使用できません。たとえば、以下は受け入れられます。*

```
NO_PROXY=.example.com
```

しかし、以下の場合は受け入れられません。

```
NO_PROXY=*.example.com
```

*NO_PROXY で許可されるワイルドカードは * 1 文字だけです。このワイルドカードはすべてのホストに一致し、プロキシーを効果的に無効にします。*

この一覧にある名前はそれぞれ、ホスト名をサフィックスとして含むドメインまたはホスト名自体のいずれかと一致します。

注記

ノードを拡張するときは、ホスト名の一覧ではなくドメイン名を使用してください。

たとえば、example.com は example.com、example.com:80、および www.example.com と一致します。

26.3. ホストでのプロキシーの設定

1. *OpenShift Container Platform 制御ファイルのプロキシー環境変数を編集します。クラスター内のすべてのファイルが正しいことを確認してください。*

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,10.1.0.0/16,172.30.0.0/16 (1)
```

1 ホスト名と CIDR をサポートします。SDN ネットワークとサービスの IP 範囲 `10.1.0.0/16,172.30.0.0/16` がデフォルトで含まれている必要があります。

2. マスターまたはノードホストを再起動します。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

26.4. ANSIBLE を使用したホストでのプロキシー設定

クラスターのインストール中に、`openshift_no_proxy`、`openshift_http_proxy` および `openshift_https_proxy` パラメーターを使用して、`NO_PROXY`、`HTTP_PROXY` および `HTTPS_PROXY` の環境変数を設定することができます。これらは、インベントリーファイルで設定可能です。

Ansible を使用したプロキシー設定例

```
# Global Proxy Configuration
# These options configure HTTP_PROXY, HTTPS_PROXY, and NOPROXY environment
# variables for docker and master services.
openshift_http_proxy=http://<user>:<password>@<ip_addr>:<port>
openshift_https_proxy=https://<user>:<password>@<ip_addr>:<port>
openshift_no_proxy='.hosts.example.com,some-host.com'
#
# Most environments do not require a proxy between OpenShift masters,
nodes, and
# etcd hosts. So automatically add those host names to the
openshift_no_proxy list.
# If all of your hosts share a common domain you may wish to disable this
and
# specify that domain above.
# openshift_generate_no_proxy_hosts=True
```

注記

Ansible パラメーターを使用してビルド用に設定できるその他のプロキシー設定があります。たとえば、以下の通りです。

`openshift_builddefaults_git_http_proxy` パラメーターと `openshift_builddefaults_git_https_proxy` パラメーターを使用すると、Git クローン作成のためにプロキシーを使用することができます。

`openshift_builddefaults_http_proxy` パラメーターと `openshift_builddefaults_https_proxy` パラメーターは、Docker ビルドストラテジーとカスタムビルドストラテジーのプロセスで環境変数を利用可能にできます。

26.5. DOCKER PULL のプロキシー設定

OpenShift Container Platform のノードホストは Docker レジストリーに対してプッシュ操作とプル操作を実行する必要があります。ノードがアクセスするためにプロキシーを必要としないレジストリーの場合は、以下の項目を指定した `NO_PROXY` パラメーターを含めてください。

- レジストリーのホスト名
- レジストリーサービスの IP アドレス
- サービス名

これにより、外部の HTTP プロキシーをオプションのままにして、そのレジストリーをブラックリストに入れることができます。

1. 以下を実行して、レジストリーサービスの IP アドレス `docker_registy_ip` を取得します。

```
$ oc describe svc/docker-registry -n default

Name:   docker-registry
Namespace:  default
Labels:   docker-registry=default
Selector:  docker-registry=default
Type:   ClusterIP
IP:   172.30.163.183 (1)
Port:   5000-tcp 5000/TCP
Endpoints:  10.1.0.40:5000
Session Affinity: ClientIP
No events.
```

(1) レジストリーサービスの IP です。

2. /etc/sysconfig/docker ファイルを編集し、シェル形式の `NO_PROXY` 変数を追加して、`<docker_registry_ip>` を直前の手順で取得した IP アドレスに置き換えます。

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,<docker_registry_ip>,docker-
registry.default.svc.cluster.local
```

3. Docker サービスを再起動します。

```
# systemctl restart docker
```

26.6. プロキシーの背後での MAVEN の使用

プロキシーで Maven を使用するには、`HTTP_PROXY_NONPROXYHOSTS` 変数を使用する必要があります。

Maven をプロキシーの背後に設定する手順などを含めた Red Hat JBoss Enterprise Application Platform 向けの OpenShift Container Platform 環境の設定に関する詳細については、「Red Hat JBoss Enterprise Application Platform for OpenShift」を参照してください。

26.7. S2I ビルドでのプロキシーの設定

S2I ビルドはさまざまな場所から依存関係を取得します。.s2i/environment ファイルを使用して単純なシェル変数を指定することができます。それに応じて OpenShift Container Platform はビルドイメージの参照時に応答します。

以下は、サンプル値を指定したサポート対象のプロキシー環境変数です。

```
HTTP_PROXY=http://USERNAME:PASSWORD@10.0.1.1:8080/
HTTPS_PROXY=https://USERNAME:PASSWORD@10.0.0.1:8080/
NO_PROXY=master.hostname.example.com
```

26.8. デフォルトテンプレートでのプロキシーの設定

OpenShift Container Platform でデフォルトで利用可能なテンプレートサンプルには、HTTP プロキシーの設定が含まれていません。これらのテンプレートに基づく既存のアプリケーションについては、アプリケーションのビルド設定の source セクションを変更して、以下のプロキシー設定を追加します。

```
...
source:
  type: Git
  git:
    uri: https://github.com/openshift/ruby-hello-world
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
...
```

これは、Git クローン作成用にプロキシーを使用するプロセスと似ています。

26.9. POD でのプロキシー環境変数の設定

デプロイメント設定の templates.spec.containers スタンザで NO_PROXY、HTTP_PROXY、および HTTPS_PROXY 環境変数を設定して、プロキシー接続情報を渡すことができます。実行時の Pod のプロキシー設定についても同じことを実行できます。

```
...
containers:
- env:
  - name: "HTTP_PROXY"
    value: "http://<user>:<password>@<ip_addr>:<port>"
...
```

oc set env コマンドを使用して、既存のデプロイメント設定を新規の環境変数で更新することもできます。

```
$ oc set env dc/frontend HTTP_PROXY=http://<user>:<password>@<ip_addr>:
<port>
```

OpenShift Container Platform インスタンスで ConfigChange トリガーを設定している場合は、変更が自動的に行われます。そうでない場合は、変更が反映されるようにアプリケーションを手動で再デプロイしてください。

26.10. GIT リポジトリーのアクセス

プロキシーの使用によってのみ Git リポジトリーにアクセスできる場合は、使用するプロキシーを BuildConfig の source セクションで定義できます。HTTP プロキシーと HTTPS プロキシーの両方

を使用するように設定できます。どちらのフィールドもオプションです。NoProxy フィールドを使用して、プロキシーを実行しないドメインを指定することもできます。

注記

これを機能させるために、ソース URI では HTTP プロトコルまたは HTTPS プロトコルを使用する必要があります。

```
source:
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
```

第27章 グローバルビルドのデフォルトと上書きの設定

27.1. 概要

開発者はプロジェクト内の特定のビルド設定において、Git クローン作成のためのプロキシー設定などの設定を定義することができます。開発者に特定の設定をそれぞれのビルド設定で定義するように要求するのではなく、管理者が受付制御プラグインを使用して、すべてのビルドでこの設定を自動的に使用するグローバルビルドのデフォルトと上書きを設定することができます。

これらのプラグインから取得した設定は、ビルドプロセス時に使用されるだけで、ビルド設定やビルド自体で設定されません。プラグインでの設定を使用することで、管理者はグローバル設定をいつでも変更することができ、既存のビルド設定またはビルドから実行されたビルドに、新規設定が割り当てられます。

- `BuildDefaults` 受付制御プラグインを使用すると、管理者は Git HTTP や HTTPS プロキシーなどの設定についてのグローバルなデフォルトと、デフォルトの環境変数を設定することができます。これらのデフォルトによって、特定のビルド用に設定された値が上書きされることはありません。ただし、それらの値がビルド定義に存在しない場合は、デフォルト値に設定されます。

- `BuildOverrides` 受付制御プラグインを使用すると、管理者はビルドに保存されている値に関係なく、ビルドの設定を上書きできます。現時点では、このプラグインには、ビルド時にレジストリーからのローカルイメージが強制的に更新されるように、ビルドストラテジーの `forcePull` フラグを上書きするサポートがあります。更新することで、ユーザーはプル可能なイメージでしかビルドできないようにします。このプラグインは、イメージラベルセットが全ビルドイメージに適用されるように設定することも可能です。
  `BuildOverrides` の受付制御プラグインと上書き可能な値の設定に関する情報は、「グローバルビルドの上書きの手動設定」を参照してください。

デフォルトのノードセレクターおよび、`BuildDefaults` または `BuildOverride` 受付プラグインは以下のように連携されます。

- マスター設定ファイルの `projectConfig.defaultNodeSelector` フィールドに定義されているデフォルトのプロジェクトノードセレクターは、指定の `nodeSelector` 値なしに、全プロジェクトに作成済みの Pod に適用されます。これらの設定は、`BuildDefaults` または `BuildOverride` ノードセレクターが設定されていないクラスターで、`nodeSelector="null"` が指定されているビルドに適用されます。

- `nodeSelector="null"` パラメーターがビルド設定に設定されている場合のみ、クラスター全体のデフォルトのビルドノードセレクター `admissionConfig.pluginConfig.BuildDefaults.configuration.nodeSelector` が適用されます。`nodeSelector=null` はデフォルト設定です。

- デフォルトのプロジェクトまたはクラスター全体のノードセレクターの場合には、デフォルト設定がビルドのノードセレクターに AND として追加されます。このビルドのノードセレクターは、`BuildDefaults` または `BuildOverride` 受付プラグインで設定されます。これらの設定は、`BuildOverrides` ノードセレクターとプロジェクトのデフォルトノードセレクターの条件を満たすノードに対してのみ、ビルドがスケジューリングされるという意味です。

注記

RunOnceDuration プラグインを使用することで、ビルド Pod の実行時間のハード制限を定義できます。

27.2. グローバルビルドのデフォルトの設定

グローバルビルドのデフォルトは以下の 2 通りの方法で設定できます。

- Ansible の使用
- master-config.yaml ファイルを変更して手動で設定する

27.2.1. Ansible を使用したグローバルビルドのデフォルトの設定

クラスターのインストール時に、以下のパラメーターを使用して、BuildDefaults プラグインを設定できます。これらのパラメーターは、インベントリーファイルで設定可能です。

- openshift_builddefaults_http_proxy
- openshift_builddefaults_https_proxy
- openshift_builddefaults_no_proxy
- openshift_builddefaults_git_http_proxy
- openshift_builddefaults_git_https_proxy
- openshift_builddefaults_git_no_proxy
- openshift_builddefaults_image_labels
- openshift_builddefaults_nodeselectors
- openshift_builddefaults_annotations
- openshift_builddefaults_resources_requests_cpu
- openshift_builddefaults_resources_requests_memory
- openshift_builddefaults_resources_limits_cpu
- openshift_builddefaults_resources_limits_memory

例27.1 Ansible を使用したビルドのデフォルトの設定例

```
# These options configure the BuildDefaults admission controller which
injects
# configuration into Builds. Proxy related values will default to the
global proxy
# config values. You only need to set these if they differ from the
global proxy settings.
openshift_builddefaults_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_no_proxy=mycorp.com
openshift_builddefaults_git_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_no_proxy=mycorp.com
openshift_builddefaults_image_labels=
[{'name':'imagelabelname1','value':'imagelabelvalue1'}]
```

```
openshift_builddefaults_nodeselectors={'nodelabel1':'nodelabelvalue1'}
openshift_builddefaults_annotations=
{'annotationkey1':'annotationvalue1'}
openshift_builddefaults_resources_requests_cpu=100m
openshift_builddefaults_resources_requests_memory=256Mi
openshift_builddefaults_resources_limits_cpu=1000m
openshift_builddefaults_resources_limits_memory=512Mi

# Or you may optionally define your own build defaults configuration
serialized as json
#openshift_builddefaults_json='{"BuildDefaults":{"configuration":
{"apiVersion":"v1","env":
[{"name":"HTTP_PROXY","value":"http://proxy.example.com.redhat.com:3128"
},{"name":"NO_PROXY","value":"ose3-
master.example.com"}],"gitHTTPProxy":"http://proxy.example.com:3128","gi
tNoProxy":"ose3-master.example.com","kind":"BuildDefaultsConfig"}}}'
```

27.2.2. グローバルビルドのデフォルトの手動設定

BuildDefaults プラグインを設定するには、以下の手順を実行します。

1. マスターノードの **/etc/origin/master/master-config.yaml** ファイルにプラグインの設定を追加します。

```
admissionConfig:
  pluginConfig:
    BuildDefaults:
      configuration:
        apiVersion: v1
        kind: BuildDefaultsConfig
        gitHTTPProxy: http://my.proxy:8080 (1)
        gitHTTPSProxy: https://my.proxy:8443 (2)
        gitNoProxy: somedomain.com, otherdomain.com (3)
        env:
        - name: HTTP_PROXY (4)
          value: http://my.proxy:8080
        - name: HTTPS_PROXY (5)
          value: https://my.proxy:8443
        - name: BUILD_LOGLEVEL (6)
          value: 4
        - name: CUSTOM_VAR (7)
          value: custom_value
        imageLabels:
        - name: url (8)
          value: https://containers.example.org
        - name: vendor
          value: ExampleCorp Ltd.
        nodeSelector: (9)
          key1: value1
          key2: value2
        annotations: (10)
          key1: value1
          key2: value2
```

```
      resources: 11
        requests:
          cpu: "100m"
          memory: "256Mi"
        limits:
          cpu: "100m"
          memory: "256Mi"
```

1. Git リポジトリーからソースコードのクローンを作成する場合に使用する HTTP プロキシーを設定します。
2. Git リポジトリーからソースコードのクローンを作成する場合に使用する HTTPS プロキシーを設定します。
3. プロキシーを実行しないドメインの一覧を設定します。
4. ビルド時に使用する HTTP プロキシーを設定するデフォルトの環境変数。この環境変数は、アセンブルおよびビルドの段階で依存関係をダウンロードするために使用できます。
5. ビルド時に使用する HTTPS プロキシーを設定するデフォルトの環境変数。この環境変数は、アセンブルおよびビルドの段階で依存関係をダウンロードするために使用できます。
6. ビルド時にビルドのログレベルを設定するデフォルトの環境変数。
7. すべてのビルドに追加されるその他のデフォルトの環境変数。
8. すべてのイメージビルドに適用されるラベル。このラベルは `BuildConfig` で上書きできます。
9. ビルド Pod は `key1=value2` および `key2=value2` のラベルが付いたノード上でのみ実行されます。ユーザーは、これらの値が無視される場合に、ビルドに対して異なる `nodeSelectors` セットを定義することができます。
10. ビルド Pod にはこれらのアノテーションが追加されます。
11. `BuildConfig` に関連リソースが定義されていない場合は、デフォルトのリソースをビルド Pod に設定します。

2. 変更を有効にするために、マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

27.3. グローバルビルドの上書きの設定

グローバルビルドの上書きは以下の 2 通りの方法で設定できます。

- *Ansible の使用*
- master-config.yaml *ファイルを変更して手動で設定する*

27.3.1. Ansible を使用したグローバルビルドの上書きの設定

クラスターのインストール時に、以下のパラメーターを使用して、**BuildOverrides** プラグインを設定できます。これらのパラメーターは、インベントリーファイルで設定可能です。

- **openshift_buildoverrides_force_pull**
- **openshift_buildoverrides_image_labels**
- **openshift_buildoverrides_nodeselectors**
- **openshift_buildoverrides_annotations**
- **openshift_buildoverrides_tolerations**

例27.2 Ansible を使用したビルドの上書きの設定例

```
# These options configure the BuildOverrides admission controller which
injects
# configuration into Builds.
openshift_buildoverrides_force_pull=true
openshift_buildoverrides_image_labels=
[{'name':'imagelabelname1','value':'imagelabelvalue1'}]
openshift_buildoverrides_nodeselectors={'nodelabel1':'nodelabelvalue1'}
openshift_buildoverrides_annotations=
{'annotationkey1':'annotationvalue1'}
openshift_buildoverrides_tolerations=
[{'key':'mykey1','value':'myvalue1','effect':'NoSchedule','operator':'Eq
ual'}]

# Or you may optionally define your own build overrides configuration
serialized as json
#openshift_buildoverrides_json='{"BuildOverrides":{"configuration":
{"apiVersion":"v1","kind":"BuildOverridesConfig","forcePull":"true","tol
erations":
[{'key':'mykey1','value':'myvalue1','effect':'NoSchedule','operator':'Eq
ual'}]}}}'
```

27.3.2. グローバルビルドの上書きの手動設定

BuildOverrides プラグインを設定するには、以下の手順を実行します。

1. マスターの **/etc/origin/master/master-config.yaml** ファイルにプラグインの設定を追加します。

```
admissionConfig:
  pluginConfig:
    BuildOverrides:
      configuration:
        apiVersion: v1
        kind: BuildOverridesConfig
        forcePull: true (1)
        imageLabels:
        - name: distribution-scope (2)
          value: private
```

```
    nodeSelector: (3)
      key1: value1
      key2: value2
    annotations: (4)
      key1: value1
      key2: value2
    tolerations: (5)
    - key: mykey1
      value: myvalue1
      effect: NoSchedule
      operator: Equal
    - key: mykey2
      value: myvalue2
      effect: NoExecute
      operator: Equal
```

(1) ビルドの開始前に、すべてのビルドがビルダーイメージとソースイメージをプルするよう強制的に実行します。

(2) すべてのイメージビルドに適用される追加のラベル。ここで定義されたラベルは `BuildConfig` で定義されたラベルよりも優先されます。

(3) ビルド Pod は `key1=value2` および `key2=value2` のラベルが付けられたノード上でのみ実行されます。ユーザーはキー/値のラベルを追加定義して、ビルドが実行されるノードのセットをさらに制限することができます。ただし、ノードには少なくともこれらのラベルを付ける必要があります。

(4) ビルド Pod にはこれらのアノテーションが追加されます。

(5) ビルド Pod にある既存の容認はここに一覧されている値で上書きされます。

2. 変更を有効にするために、マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

第28章 PIPELINE 実行の設定

28.1. 概要

ユーザーがPipeline ビルドストラテジーを使用して初めてビルド設定を作成する際に、OpenShift Container Platform は `jenkins-ephemeral` という名前のテンプレートを `openshift` namespace で検索し、ユーザーのプロジェクト内でそれをインスタンス化します。OpenShift Container Platform に同梱される `jenkins-ephemeral` テンプレートは、インスタンス化の実行時に以下を作成します。

- OpenShift Container Platform の公式の Jenkins イメージを使用した Jenkins のデプロイメント設定
- Jenkins デプロイメントにアクセスするためのサービスとルート
- 新規の Jenkins サービスアカウント
- サービスアカウントにプロジェクトへの編集アクセスを付与する RoleBinding

クラスター管理者は、組み込みテンプレートのコンテンツの変更、またはクラスターを異なるテンプレート場所にポイントするためのクラスター設定の編集によって作成されるものを制御できます。

デフォルトテンプレートのコンテンツを変更するには、以下を実行します。

```
$ oc edit template jenkins-ephemeral -n openshift
```

Jenkins 用に永続ストレージを使用する `jenkins-persistent` テンプレートなどの異なるテンプレートを使用するには、以下をマスター設定ファイルに追加します。

```
jenkinsPipelineConfig:
  autoProvisionEnabled: true 1
  templateNamespace: openshift 2
  templateName: jenkins-persistent 3
  serviceName: jenkins-persistent-svc 4
  parameters: 5
    key1: value1
    key2: value2
```

1. これが指定されていない場合はデフォルトで `true` に設定されます。`false` が指定されている場合は、いずれのテンプレートもインスタンス化されません。
2. インスタンス化されるテンプレートが含まれる namespace。
3. インスタンス化されるテンプレートの名前。
4. インスタンス化の実行時にテンプレートによって作成されるサービスの名前。
5. インスタンス化の実行中にテンプレートに渡すオプションの値。

Pipeline ビルド設定の作成時に、OpenShift Container Platform は `serviceName` に一致するサービスを検索します。つまり、`serviceName` はプロジェクト内で一意であるように選択される必要があります。サービスが見つからない場合、OpenShift Container Platform は `jenkinsPipelineConfig`

テンプレートをインスタンス化します。この方法が適さない場合 (たとえば、OpenShift Container Platform の外部にある Jenkins サーバーを使用する場合)、ユーザーのロールに応じていくつかのことを実行できます。

- クラスター管理者の場合は、単に `autoProvisionEnabled` を `false` に設定します。これにより、クラスター全体で自動プロビジョニングが無効にされます。
- 非特権ユーザーである場合は、OpenShift Container Platform で使用するサービスを作成する必要があります。サービス名は `jenkinsPipelineConfig` の `serviceName` のクラスター設定値と一致している必要があります。デフォルト値は `jenkins` です。プロジェクトの外部で Jenkins サーバーを実行しているために自動プロビジョニングを無効にしている場合は、この新規サービスを既存の Jenkins サーバーにポイントすることが推奨されます。「外部サービスの統合」を参照してください。

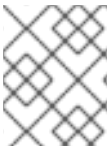
後者のオプションは、選択したプロジェクト内でのみ自動プロビジョニングを無効にするためにも使用できます。

28.2. OPENSHIFT JENKINS CLIENT プラグイン

OpenShift Jenkins Client プラグインは、OpenShift API Server との高度な対話を実現するために、読み取り可能かつ簡潔で、包括的で Fluent (流れるような) な Jenkins パイプライン構文を提供することを目的とした Jenkins プラグインです。このプラグインは、スクリプトを実行しているノードで使用できる必要がある OpenShift コマンドラインツール (oc) を活用します。

プラグインのインストールと設定の詳細については、以下のリンクから公式ドキュメントを参照し、確認してください。

- インストール
- OpenShift クラスターの設定
- 認証情報の設定
- Jenkins ノードの設定



注記

このプラグインの使用に関する情報を必要としている開発者の場合は、「OpenShift Pipeline の概要」を参照してください。

28.3. OPENSHIFT JENKINS の同期プラグイン

この Jenkins プラグインは、OpenShift BuildConfig および Build オブジェクトと Jenkins ジョブおよびビルドとの同期を維持します。

OpenShift Jenkins 同期プラグインは以下を実行します。

- Jenkins での動的なジョブ/実行の作成。
- ImageStreams、ImageStreamTag、または ConfigMap からのスレーブ Pod テンプレートの動的作成。
- 環境変数の挿入。
- OpenShift Web コンソールでの Pipeline の可視化。

- *Jenkins Git プラグインとの統合。これにより、OpenShift ビルドから Jenkins Git プラグインにコミット情報が渡されます。*

このプラグインの詳細については、以下を参照してください。

- *OpenShift Jenkins 同期プラグイン*
- *OpenShift Container Platform 同期プラグイン*

第29章 ルートのタイムアウトの設定

OpenShift Container Platform のインストールとルーターのデプロイ後、Service Level Availability (SLA) の要件に合わせてタイムアウトの数値を低く設定したり、バックエンドの処理速度が遅いためにタイムアウトの数値を高く設定したりする必要のあるサービスがある場合には、既存のルートのデフォルトのタイムアウトを設定することができます。

`oc annotate` コマンドを使用して、ルートにタイムアウトを追加します。

```
# oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit>
```

たとえば、`myroute` という名前のルートに 2 秒のタイムアウトを設定するには以下のようにします。

```
# oc annotate route myroute --overwrite
haproxy.router.openshift.io/timeout=2s
```

サポートされる時間単位は、マイクロ秒 (us)、ミリ秒 (ms)、秒 (s)、分 (m)、時間 (h)、または日 (d) です。

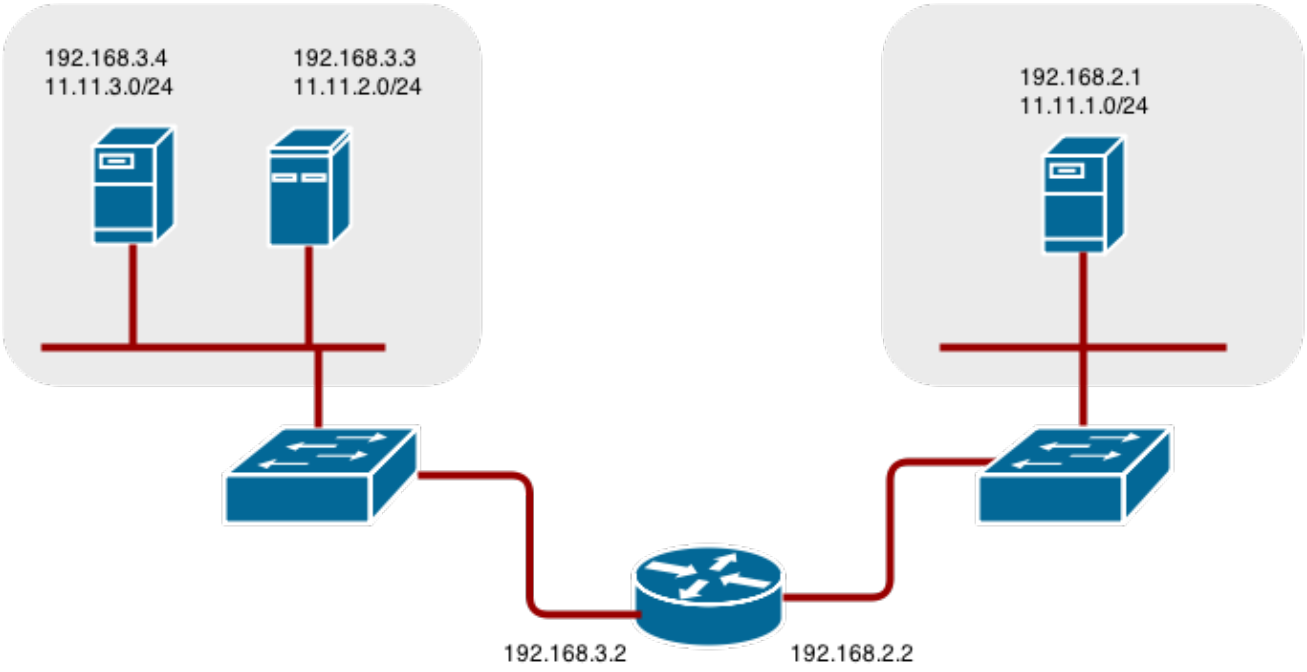
第30章 ネイティブのコンテナールーティングの設定

30.1. ネットワークの概要

一般的なネットワーク設定を以下に示します。

- 11.11.0.0/16 はコンテナーネットワークです。
- 11.11.x.0/24 サブネットは各ノード用に予約済みで、Docker Linux ブリッジに割り当てられています。
- 各ノードには 11.11.0.0/16 の範囲内にある (ローカルサブネットを除く) あらゆるものに到達するために使用するルーターへのルートがあります。
- ルーターには各ノードのルートがあるため、適切なノードにポイントできます。
- ネットワークトポロジーが変更されない限り、新規ノードが追加されても既存のノードを変更する必要はありません。
- IP 転送が各ノードで有効になっています。

以下の図はこのトピックで説明されているコンテナーのネットワーク設定を示しています。2 つのネットワークインターフェースカードを搭載したルーターとして機能する 1 つの Linux ノード、2 つのスイッチ、およびこれらのスイッチに接続された 3 つのノードを使用しています。



30.2. ネイティブのコンテナールーティングの設定

既存のスイッチとルーター、Linux のカーネルネットワークスタックを使用してコンテナーネットワークを設定できます。

ネットワーク管理者は、新規ノードがクラスターに追加される際に、ルーターを変更するか、ルーターを変更するスクリプトを作成する必要があります。

このプロセスを変化させて、あらゆるタイプのルーターで使用することができます。

30.3. コンテナーネットワーク用のノードのセットアップ

1. 未使用の 11.11.x.0/24 サブネット IP アドレスをノードの Linux ブリッジに割り当てます。

```
# brctl addbr lbr0
# ip addr add 11.11.1.1/24 dev lbr0
# ip link set dev lbr0 up
```

2. 新規ブリッジを使用するように Docker 起動スクリプトを変更します。デフォルトでは、起動スクリプトは `/etc/sysconfig/docker` ファイルです。

```
# docker -d -b lbr0 --other-options
```

3. 11.11.0.0/16 ネットワークのルーターへのルートを追加します。

```
# ip route add 11.11.0.0/16 via 192.168.2.2 dev p3p1
```

4. ノードで IP 転送を有効にします。

```
# sysctl -w net.ipv4.ip_forward=1
```

30.4. コンテナーネットワーク用のルーターのセットアップ

以下の手順は、複数の NIC を搭載した Linux ボックスがルーターとして使用されていることを前提としています。必要に応じて手順を変更して、特定のルーターに対応する構文を使用してください。

1. ルーターで IP 転送を有効にします。

```
# sysctl -w net.ipv4.ip_forward=1
```

2. クラスターに追加された各ノードのルートを追加します。

```
# ip route add <node_subnet> via <node_ip_address> dev <interface
through which node is L2 accessible>
# ip route add 11.11.1.0/24 via 192.168.2.1 dev p3p1
# ip route add 11.11.2.0/24 via 192.168.3.3 dev p3p2
# ip route add 11.11.3.0/24 via 192.168.3.4 dev p3p2
```

第31章 エッジロードバランサーからのルーティング

31.1. 概要

OpenShift Container Platform クラスター内の Pod は、クラスターネットワーク上の IP アドレスを介してのみ到達可能です。エッジロードバランサーを使用することで、ネットワーク外部からのトラフィックを受け取り、OpenShift Container Platform クラスター内の Pod にトラフィックをプロキシーすることができます。ロードバランサーがクラスターネットワークに含まれていない場合は、内部クラスターネットワークがエッジロードバランサーからアクセスできないため、ルーティングが困難になります。

OpenShift Container Platform クラスターがクラスターネットワークソリューションとして OpenShift Container Platform SDN を使用している場合にこの問題を解決するには、Pod へのネットワークアクセスを以下の 2 つの方法で確保することができます。

31.2. ロードバランサーの SDN への追加

可能な場合は、ネットワークプラグインとして OpenShift SDN を使用するロードバランサー自体で OpenShift Container Platform ノードインスタンスを実行します。これにより、エッジマシンは独自の Open vSwitch ブリッジを取得し、このブリッジは、クラスターにある Pod とノードへのアクセスを提供するために SDN によって自動的に設定されます。ルーティングテーブル は、Pod が作成および削除される際に SDN によって動的に設定されるため、ルーティングソフトウェアは Pod に到達することができます。

Pod がロードバランサー自体に設定されないように、ロードバランサーマシンに スケジュール対象外ノードというマークを付けます。

```
$ oc adm manage-node <load_balancer_hostname> --schedulable=false
```

ロードバランサーがコンテナーとしてパッケージ化されている場合は、OpenShift Container Platform との統合が簡単になります。ロードバランサーをホストポートが公開されている状態の Pod として実行します。OpenShift Container Platform で事前にパッケージ化された HAProxy ルーターはこの方法で実行されます。

31.3. RAMP ノードを使用したトンネルの確立

前述のソリューションを使用できない場合もあります。たとえば、F5® は互換性のないカスタム Linux カーネルとディストリビューションを使用するため、F5 BIG-IP® ホストは OpenShift Container Platform ノードインスタンスまたは OpenShift Container Platform SDN を実行できません。

代わりに、F5 BIG-IP® を有効にして Pod に到達できるようにするために、クラスターネットワーク内の既存のノードを Ramp ノード として選択し、F5 BIG-IP® ホストと指定された Ramp ノード間でトンネルを確立することができます。Ramp ノードは通常の OpenShift Container Platform ノードであるため、Ramp ノードには、トラフィックをクラスターネットワーク内のノードにある Pod にルーティングするための設定が必要になります。そのため、Ramp ノードは F5 BIG-IP® ホストがクラスターネットワーク全体にアクセスする際に使用するゲートウェイのロールを引き継ぎます。

以下は、F5 BIG-IP® ホストと指定された Ramp ノード間で ipip トンネルを確立する例です。

F5 BIG-IP® ホスト側:

1. 以下の変数を設定します。

```
# F5_IP=10.3.89.66 (1)
```

```
# RAMP_IP=10.3.89.89 2
# TUNNEL_IP1=10.3.91.216 3
# CLUSTER_NETWORK=10.128.0.0/14 4
```

1 2 *F5_IP 変数と RAMP_IP 変数はそれぞれ、共有内部ネットワーク上にある F5 BIG-IP® ホストと Ramp ノードの IP アドレスを指しています。*

3 *ipip トンネルの F5® ホストの終端となる競合しない任意の IP アドレス。*

4 *OpenShift SDN が Pod にアドレスを割り当てるために使用するオーバーレイネットワークの CIDR 範囲。*

2. *古い route、self、tunnel および SNAT pool を削除します。*

```
# tmsh delete net route $CLUSTER_NETWORK || true
# tmsh delete net self SDN || true
# tmsh delete net tunnels tunnel SDN || true
# tmsh delete ltm snatpool SDN_snatpool || true
```

3. *新規の tunnel、self、route および SNAT pool を作成し、この SNAT pool を仮想サーバーで使用します。*

```
# tmsh create net tunnels tunnel SDN \
    \{ description "OpenShift SDN" local-address \
    $F5_IP profile ipip remote-address $RAMP_IP \}
# tmsh create net self SDN \{ address \
    ${TUNNEL_IP1}/24 allow-service all vlan SDN \}
# tmsh create net route $CLUSTER_NETWORK interface SDN
# tmsh create ltm snatpool SDN_snatpool members add { $TUNNEL_IP1 }
# tmsh modify ltm virtual  ose-vserver source-address-translation {
type snat pool SDN_snatpool }
# tmsh modify ltm virtual  https-ose-vserver source-address-
translation { type snat pool SDN_snatpool }
```

Ramp ノード:

注記

以下で、永続性のない設定が作成されます。つまり、ramp ノードまたは openvswitch サービスを再起動すると、この設定はなくなります。

1. *以下の変数を設定します。*

```
# F5_IP=10.3.89.66
# TUNNEL_IP1=10.3.91.216
# TUNNEL_IP2=10.3.91.217 1
# CLUSTER_NETWORK=10.128.0.0/14 2
```

1 *ipip トンネルの Ramp ノードの終端となる 2 番目の任意の IP アドレス。*

2 *OpenShift SDN が Pod にアドレスを割り当てるために使用するオーバーレイネットワークの CIDR 範囲。*

2. *古いトンネルを削除します。*

```
# ip tunnel del tun1 || true
```

3. *適切な L2 接続インターフェース (たとえば、eth0) を使用して、Ramp ノードで ipip トンネルを作成します。*

```
# ip tunnel add tun1 mode ipip \
    remote $F5_IP dev eth0
# ip addr add $TUNNEL_IP2 dev tun1
# ip link set tun1 up
# ip route add $TUNNEL_IP1 dev tun1
# ping -c 5 $TUNNEL_IP1
```

4. *SDN サブネットの未使用の IP を使用してトンネル IP を SNAT します。*

```
# source /run/openshift-sdn/config.env
# tap1=$(ip -o -4 addr list tun0 | awk '{print $4}' | cut -d/ -f1 |
head -n 1)
# subaddr=$(echo ${OPENSHIFT_SDN_TAP1_ADDR:-"$tap1"} | cut -d "." -f
1,2,3)
# export RAMP_SDN_IP=${subaddr}.254
```

5. *この `RAMP_SDN_IP` を追加のアドレスとして tun0 (ローカル SDN のゲートウェイ) に割り当てます。*

```
# ip addr add ${RAMP_SDN_IP} dev tun0
```

6. *SNAT の OVS ルールを変更します。*

```
# ipflowopts="cookie=0x999,ip"
# arpflowopts="cookie=0x999, table=0, arp"
#
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"${ipflowopts},nw_src=${TUNNEL_IP1},actions=mod_nw_src:${RAMP_SDN_IP
},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \

"${ipflowopts},nw_dst=${RAMP_SDN_IP},actions=mod_nw_dst:${TUNNEL_IP1
},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "${arpflowopts}, arp_tpa=${RAMP_SDN_IP}, actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "${arpflowopts}, priority=200, in_port=2,
arp_spa=${RAMP_SDN_IP}, arp_tpa=${CLUSTER_NETWORK},
actions=goto_table:30"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "arp, table=5, priority=300, arp_tpa=${RAMP_SDN_IP},
actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
    "ip,table=5,priority=300,nw_dst=${RAMP_SDN_IP},actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0
"${ipflowopts},nw_dst=${TUNNEL_IP1},actions=output:2"
```

7. 高可用性を実現するように Ramp ノードを設定する予定がない場合は、必要に応じて、Ramp ノードをスケジュール対象外としてマークします。以下のセクションに従う予定や、高可用性を備えた Ramp ノードを作成する予定がある場合は、この手順を省略してください。

```
$ oc adm manage-node <ramp_node_hostname> --schedulable=false
```

注記

F5 ルータープラグインは F5 BIG-IP® と統合します。

31.3.1. 高可用性を備えた Ramp ノードの設定

keepalived を内部で使用する OpenShift Container Platform の ipfailover 機能を使用することで、F5 BIG-IP® の観点から Ramp ノードの高可用性を確保することができます。これを実行するには、まず同じ L2 サブネット上の 2 つのノード (たとえば、ramp-node-1 および ramp-node-2) を起動します。

次に、仮想 IP (VIP) を使用するために未割り当ての IP アドレスを同じサブネット内から選択します。この IP アドレスは、F5 BIG-IP® でトンネルを設定するときに使用する `RAMP_IP` 変数として設定されます。

たとえば、Ramp ノードに対して使用するサブネットは 10.20.30.0/24 とし、10.20.30.2 を ramp-node-1 に、10.20.30.3 を ramp-node-2 に割り当てているとします。VIP については、同じ 10.20.30.0/24 サブネットから未割り当てのアドレスを選択します (例: 10.20.30.4)。次に、ipfailover を設定するために、両方のノードに f5rampnode などのラベルでマークを付けします。

```
$ oc label node ramp-node-1 f5rampnode=true
$ oc label node ramp-node-2 f5rampnode=true
```

ipfailover ドキュメントで説明されているのと同様に、ここでサービスアカウントを作成し、そのアカウントを特権付き SCC に追加する必要があります。最初に、f5ipfailover サービスアカウントを作成します。

```
$ oc create serviceaccount f5ipfailover -n default
```

次に、f5ipfailover サービスを特権付き SCC に追加できます。default namespace の f5ipfailover を特権付き SCC に追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-user privileged
system:serviceaccount:default:f5ipfailover
```

最後に、選択した VIP (`RAMP_IP` 変数) と f5ipfailover サービスアカウントを使用して、先に設定した f5rampnode ラベルを使用して VIP を 2 つのノードに割り当て、ipfailover を設定します。

```
# RAMP_IP=10.20.30.4
# IFNAME=eth0 (1)
# oc adm ipfailover <name-tag> \
    --virtual-ips=$RAMP_IP \
    --interface=$IFNAME \
    --watch-port=0 \
    --replicas=2 \
    --service-account=f5ipfailover  \
    --selector='f5rampnode=true'
```

1 *RAMP_IP を設定する必要があるインターフェース。*

上記の設定を行うと、現在 VIP が割り当てられている Ramp ノードホストで障害が発生した場合に VIP (RAMP_IP 変数) が自動的に再割り当てされます。

第32章 コンテナーログの集計

32.1. 概要

OpenShift Container Platform のクラスター管理者として、EFK スタックをデプロイして各種の OpenShift Container Platform サービスのログを集計できます。アプリケーション開発者は、表示アクセス権を持つプロジェクトのログを表示することができます。EFK スタックは、複数のコンテナーまたは削除された Pod からのものであっても、ホストとアプリケーションからログを集計します。

EFK スタックは ELK スタックの変更バージョンであり、以下で構成されています。

- Elasticsearch (ES): すべてのログが格納されるオブジェクトストア。
- Fluentd: ノードからログを収集し、そのログを Elasticsearch に送ります。
- Kibana: Elasticsearch の Web UI。

クラスターにデプロイされると、スタックはすべてのノードおよびプロジェクトのログを Elasticsearch に集計し、ログを表示するために Kibana UI を提供します。クラスター管理者はすべてのログを表示できますが、アプリケーション開発者は表示権限を持つプロジェクトのログのみを表示できます。スタックのコンポーネントは安全な通信を実行します。

注記

「Docker コンテナーログの管理」では、コンテナーログを管理し、ノードディスクが満杯になることを阻止するための `json-file` ロギングドライバーオプションの使用について説明しています。

32.2. デプロイ前の設定

1. Ansible Playbook は集計されたロギングをデプロイおよびアップグレードするために利用できます。『クラスターのインストール ガイド』を十分に理解しておくようにしてください。このセクションでは、Ansible を使用する準備に必要な情報と設定に関する情報を提供します。EFK スタックの各種の領域を設定するために、パラメーターが Ansible インベントリーファイルに追加されています。

2. 「サイジングのガイドライン」を確認して、デプロイメントの最適な設定方法を判断してください。

3. クラスターのルーターをデプロイしていることを確認します。

4. Elasticsearch に必要なストレージがあることを確認します。各 Elasticsearch レプリカに独自のストレージボリュームが必要であることに注意してください。詳細については、「Elasticsearch」を参照してください。

5. プロジェクトを選択します。デプロイされたら、EFK スタックは OpenShift Container Platform クラスター内のすべてのプロジェクトのログを収集します。このセクションの例では、デフォルトのプロジェクトロギング を使用しています。プロジェクトがまだ存在していない場合は、Ansible Playbook によってプロジェクトが作成されます。プロジェクトに対してノードセレクターを指定する場合にのみ、プロジェクトを作成する必要があります。そうしない場合は、`openshift-logging` ロールによってプロジェクトが作成されます。

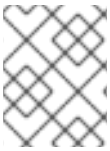
```
$ oc adm new-project logging --node-selector=""
$ oc project logging
```

注記

Fluentd はクラスター全体にデプロイする必要があり、セレクターによって Fluentd がデプロイされる場所が制限されるため、プロジェクトには空のノードセレクターを指定することをお勧めします。コンポーネントの配置を制御するには、コンポーネントごとにノードセレクターを指定して、デプロイメント設定に適用する必要があります。

32.3. ロギング ANSIBLE 変数の指定

EFK デプロイメントのパラメーターをインベントリーホストファイルに指定して、デフォルトを上書きすることができます。パラメーターを選択する前に、「Elasticsearch」と「Fluentd」のセクションを参照してください。



注記

デフォルトでは、Elasticsearch サービスはクラスターのノード間での TCP 通信にポート 9300 を使用します。

パラメーター	説明
openshift_logging_use_ops	**true** に設定されている場合、操作ログに対して 2 番目の Elasticsearch クラスターと Kibana を設定します。Fluentd はメインクラスターと操作ログ用に予約されているクラスター (**default**、**openshift**、**openshift-infra** の各プロジェクトのログ、Docker、OpenShift、およびジャーナルのシステムログで構成される) の間でログを分割します。これにより、2 番目の Elasticsearch クラスターと Kibana がデプロイされます。デプロイメントは名前に含まれる **-ops** サフィックスで見分けることができ、以下に一覧表示されている並列デプロイメントオプションがあります。並列デプロイメントオプションについては、「Curator 設定の作成」で説明されています。
openshift_logging_master_url	Kubernetes マスターの URL。これは公開されている必要はありませんが、クラスター内からアクセスできる必要があります (例: https://<PRIVATE-MASTER-URL>:8443)。
openshift_logging_master_public_url	Kubernetes マスターの公開された URL。これは、Kibana プロキシーによる認証のリダイレクトに使用されます (例: https://<CONSOLE-PUBLIC-URL-MASTER>:8443)。
openshift_logging	集計されたロギングがデプロイされる namespace。
openshift_logging_install_logging	ロギングをインストールする場合は **true** に設定します。ロギングをアンインストールする場合は **false** に設定します。
openshift_logging_purge_logging	通常のアンインストールでは、再インストール中の予期せぬデータ損失を防ぐために PVC が維持されます。Ansible Playbook が完全かつ不可逆的にすべてのロギング永続データ (PVC を含む) を確実に削除するようにするには、**openshift_logging_install_logging** を 'false' に設定してアンインストールをトリガーし、**openshift_logging_purge_logging** を 'true' に設定します。デフォルトでは 'false' に設定されています。

パラメーター	説明
openshift_logging_install_eventrouter	**openshift_logging_install_logging** と併用します。いずれも 'true' に設定されている場合は、**eventrouter** がインストールされ、'false' に設定されている場合は、**eventrouter** がアンインストールされます。
openshift_logging_eventrouter_image_prefix	**eventrouter** のロギングイメージのプレフィックス
openshift_logging_eventrouter_image_version	ロギング **eventrouter** のイメージバージョン
openshift_logging_eventrouter_sink	**eventrouter** のシンク (受信側) を選択します。**stdout** と **glog** がサポートされています。デフォルトでは **stdout** に設定されています。
openshift_logging_eventrouter_nodeselector	Pod が到達するノードを選択するためのラベルのマップ (**"node":"infra"**、**"region":"west"** など)。
openshift_logging_eventrouter_replicas	デフォルトでは '1' に設定されます。
openshift_logging_eventrouter_cpu_limit	**eventrouter** に割り当てる最小 CPU 量。デフォルトでは '100m' に設定されます。
openshift_logging_eventrouter_memory_limit	**eventrouter** Pod のメモリー制限。デフォルトでは '128Mi' に設定されます。
openshift_logging_eventrouter_namespace	**eventrouter** がデプロイされる namespace。デフォルトでは 'default' に設定されます。
openshift_logging_image_pull_secret	認証済みレジストリーからコンポーネントイメージをプルするために使用される既存のプルシークレットの名前を指定します。
openshift_logging_curator_default_days	ログレコードを削除するために Curator が使用するデフォルトの最小期間 (日単位)。
openshift_logging_curator_run_hour	Curator が実行される時刻 (時間)。
openshift_logging_curator_run_minute	Curator が実行される時刻 (分)。
openshift_logging_curator_run_timezone	実行時間を割り出すために Curator が使用するタイムゾーン。タイムゾーンは、**America/New_York** または **UTC** など、tzselect(8) または timedatectl(1) の "リージョン/ローカリティー" 形式の文字列で指定します。

パラメーター	説明
openshift_logging_curator_script_log_level	Curator のスクリプトログレベル。
openshift_logging_curator_log_level	Curator プロセスのログレベル。
openshift_logging_curator_cpu_limit	Curator に割り当てる CPU 量。
openshift_logging_curator_memory_limit	Curator に割り当てるメモリー量。
openshift_logging_curator_nodeselector	Curator インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。
openshift_logging_curator_ops_cpu_limit	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_curator_cpu_limit** と同等です。
openshift_logging_curator_ops_memory_limit	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_curator_memory_limit** と同等です。
openshift_logging_kibana_hostname	Kibana に到達する Web クライアントの外部ホスト名。
openshift_logging_kibana_cpu_limit	Kibana に割り当てる CPU 量。
openshift_logging_kibana_memory_limit	Kibana に割り当てるメモリー量。
openshift_logging_kibana_proxy_debug	**true** の場合、Kibana プロキシーのログレベルを DEBUG に設定します。
openshift_logging_kibana_proxy_cpu_limit	Kibana プロキシーに割り当てる CPU 量。
openshift_logging_kibana_proxy_memory_limit	Kibana プロキシーに割り当てるメモリー量。
openshift_logging_kibana_replica_count	Kibana を拡張して達成するレプリカ数。
openshift_logging_kibana_nodeselector	Kibana インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。

パラメーター	説明
openshift_logging_kibana_env_vars	{"ELASTICSEARCH_REQUESTTIMEOUT":"30000"} など、Kibana デプロイメント設定に追加する環境変数のマッピング。
openshift_logging_kibana_key	Kibana ルートの作成時に使用する公開されているキー。
openshift_logging_kibana_cert	Kibana ルートの作成時にキーに一致する証明書。
openshift_logging_kibana_ca	オプション。キーに添付する CA と、Kibana ルートの作成時に使用される証明書。
openshift_logging_kibana_ops_hostname	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_kibana_hostname** と同等です。
openshift_logging_kibana_ops_cpu_limit	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_kibana_cpu_limit** と同等です。
openshift_logging_kibana_ops_memory_limit	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_kibana_memory_limit** と同等です。
openshift_logging_kibana_ops_proxy_debug	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_kibana_proxy_debug** と同等です。
openshift_logging_kibana_ops_proxy_cpu_limit	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_kibana_proxy_cpu_limit** と同等です。
openshift_logging_kibana_ops_proxy_memory_limit	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_kibana_proxy_memory_limit** と同等です。
openshift_logging_kibana_ops_replica_count	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_kibana_replica_count** と同等です。
openshift_logging_es_allow_external	**true** に設定すると、Elasticsearch を re-encrypt ルートとして公開します。デフォルトでは **false** に設定されます。

パラメーター	説明
openshift_logging_es_hostname	ルートと TLS サーバーの証明書に使用する外部向けホスト名。デフォルトでは **es** に設定されます。 たとえば、**openshift_master_default_subdomain** が **=example.test** に設定されている場合、**openshift_logging_es_hostname** のデフォルト値は **es.example.test** になります。
openshift_logging_es_cert	Elasticsearch が外部 TLS サーバー証明書に使用する証明書の場所。デフォルトは、生成される証明書になります。
openshift_logging_es_key	Elasticsearch が外部 TLS サーバー証明書に使用するキーの場所。デフォルトは、生成されるキーになります。
openshift_logging_es_ca_ext	Elasticsearch が外部 TLS サーバーに使用する CA 証明書の場所。デフォルトは内部 CA です。
openshift_logging_es_ops_allow_external	**true** に設定すると、Elasticsearch が re-encrypt ルートとして公開されます。デフォルトでは **false** に設定されます。
openshift_logging_es_ops_hostname	ルートと TLS サーバー証明書に使用する外部向けホスト名。デフォルトでは **es-ops** に設定されます。 たとえば、**openshift_master_default_subdomain** が **=example.test** に設定されている場合、**openshift_logging_es_ops_hostname** のデフォルト値は **es-ops.example.test** になります。
openshift_logging_es_ops_cert	Elasticsearch が外部 TLS サーバー証明書に使用する証明書の場所。デフォルトは、生成される証明書になります。
openshift_logging_es_ops_key	Elasticsearch が外部 TLS サーバー証明書に使用するキーの場所。デフォルトは、生成されるキーになります。
openshift_logging_es_ops_ca_ext	Elasticsearch が外部 TLS サーバーに使用する CA 証明書の場所。デフォルトは内部 CA です。
openshift_logging_fluentd_nodeselector	Fluentd インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。Fluentd を実行する必要があるノード (通常はすべて) には、Fluentd が実行およびログ収集できるようになる前に、このラベルを付ける必要があります。 インストール後に Aggregated Logging クラスターを拡張する際に、**openshift_logging** ロールがこのノードセレクターを使用して **openshift_logging_fluentd_hosts** によって提供されるノードにラベルを付けます。 インストールの一環として、Fluentd ノードセレクターのラベルを永続化されたノードラベルの一覧に追加することをお勧めします。

パラメーター	説明
openshift_logging_fluentd_cpu_limit	Fluentd Pod の CPU 上限。
openshift_logging_fluentd_memory_limit	Fluentd Pod のメモリー上限。
openshift_logging_fluentd_journal_read_from_head	最初の起動時に Fluentd がジャーナルの先頭から読み取る必要がある場合は、**true** に設定します。これを使用することで、現在のログレコードを受信する ES で遅延が発生する可能性があります。
openshift_logging_fluentd_hosts	Fluentd をデプロイするためにラベルを付ける必要があるノードの一覧。デフォルトでは、全ノードに ['--all'] のラベルを付けます。null 値は **openshift_logging_fluentd_hosts={}** です。Fluentd Pod を起動するには、DaemonSet の **nodeSelector** を、['host1.example.com', 'host2.example.com'] などの有効なラベルに更新します。
openshift_logging_fluentd_audit_container_engine	**openshift_logging_fluentd_audit_container_engine** が **true** に設定されている場合には、コンテナーエンジンの監査ログが収集され、ES に保存されます。この変数を有効にすると、EFK が指定の監査ログファイルか、デフォルトの **/var/log/audit.log** ファイルを監視することができ、プラットフォームのコンテナーエンジンに関する監査情報を収集して、Kibana に配置します。
openshift_logging_fluentd_audit_file	監査ログファイルの場所。デフォルトは **/var/log/audit/audit.log** です。この変数を有効にすると、EFK が指定の監査ログファイルか、デフォルトの **/var/log/audit.log** ファイルを監視することができ、プラットフォームのコンテナーエンジンに関する監査情報を収集して、Kibana に配置します。
openshift_logging_fluentd_audit_pos_file	監査ログファイルの Fluentd **in_tail** の位置ファイルが配置される場所。デフォルトは **/var/log/audit/audit.log** です。この変数を有効にすると、EFK が指定の監査ログファイルか、デフォルトの **/var/log/audit.log** ファイルを監視することができ、プラットフォームのコンテナーエンジンに関する監査情報を収集して、Kibana に配置します。
openshift_logging_es_host	Fluentd がログを送信する必要がある ES サービスの名前。
openshift_logging_es_port	Fluentd がログを送信する必要がある ES サービスのポート。
openshift_logging_es_ca	Fluentd が **openshift_logging_es_host** との通信に使用する CA の場所。
openshift_logging_es_client_cert	Fluentd が **openshift_logging_es_host** に使用するクライアント証明書の場所。

パラメーター	説明
openshift_logging_es_client_key	Fluentd が **openshift_logging_es_host** に使用するクライアントキーの場所。
openshift_logging_es_cluster_size	デプロイする Elasticsearch レプリカ。冗長性を実現するには 3 つ以上のレプリカが必要です。
openshift_logging_es_cpu_limit	ES クラスターの CPU 量の上限。
openshift_logging_es_memory_limit	Elasticsearch インスタンスごとに予約する RAM 容量。512 M 以上である必要があります。使用可能なサフィックスは G、g、M、m です。
openshift_logging_es_number_of_replicas	すべての新規インデックスのプライマリーシャードごとのレプリカシャード数。デフォルトは '0' に設定されています。実稼働環境のクラスターに対しては、**1** 以上を設定することが推奨されます。
openshift_logging_es_number_of_shards	ES で作成される新規インデックスごとのプライマリシャード数。デフォルトは '1' に設定されます。
openshift_logging_es_pv_selector	特定の PV を選択するために PVC に追加されるキー/値のマップ。
openshift_logging_es_pvc_dynamic	バッキングストレージを動的にプロビジョニングするには、パラメーターの値を **true** に設定します。**true** に設定すると、storageClass の仕様が PVC から省略されます。**openshift_logging_es_pvc_storage_class_name** のパラメーターの値を設定すると、この値は、**openshift_logging_es_pvc_dynamic** パラメーターの値を上書きします。
openshift_logging_es_pvc_storage_class_name	デフォルト以外のストレージクラスを使用するには、**glusterprovisioner** または **cephrbdprovisioner** などのストレージクラス名を指定します。ストレージクラス名を指定した後には、openshift_logging_es_pvc_dynamic の値が何であっても、動的ボリュームプロビジョニングがアクティブになります。
openshift_logging_es_pvc_size	Elasticsearch インスタンスごとに作成する Persistent Volume Claim (永続ボリューム要求) のサイズ (例: 100 G)。省略する場合は、PVC が作成されず、代わりに一時ボリュームが使用されます。

パラメーター	説明
openshift_logging_es_pvc_prefix	Elasticsearch インスタンスのストレージとして使用される Persistent Volume Claim (永続ボリューム要求) の名前のプレフィックス。**logging-es-1** のように、インスタンスごとに数字が付加されます。まだ存在していない場合は、サイズ **es-pvc-size** を使用して作成されます。 **openshift_logging_es_pvc_prefix** が設定されている場合: • **openshift_logging_es_pvc_dynamic=true** であると、**openshift_logging_es_pvc_size** の値はオプションになります。 • **openshift_logging_es_pvc_dynamic=false** であると、**openshift_logging_es_pvc_size** の値を設定する必要があります。
openshift_logging_es_recover_after_time	リカバリーを試みる前に ES が待機する時間。
openshift_logging_es_storage_group	Elasticsearch ストレージボリュームにアクセスするための補助グループ ID の数。バッキングボリュームはこのグループ ID によるアクセスを許可する必要があります。
openshift_logging_es_nodeselector	Elasticsearch インスタンスをデプロイできるターゲットのノードを判断するマップとして指定されているノードセレクター。このノードセレクターは、Elasticsearch インスタンスの実行用に予約または最適化されているノードにこれらのインスタンスを配置するために使用することができます。たとえば、セレクターは **{"node-type":"infrastructure"}** と指定できます。Elasticsearch をデプロイする前に、少なくとも 1 つのアクティブノードにこのラベルを付ける必要があります。
openshift_logging_es_ops_allow_cluster_reader	cluster-reader ロールに操作ログの読み取りが許可されている場合は **true** に設定します。
openshift_logging_es_ops_host	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_host** と同等です。
openshift_logging_es_ops_port	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_port** と同等です。
openshift_logging_es_ops_ca	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_ca** と同等です。
openshift_logging_es_ops_client_cert	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_client_cert** と同等です。

パラメーター	説明
openshift_logging_es_ops_client_key	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_client_key** と同等です。
openshift_logging_es_ops_cluster_size	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_cluster_size** と同等です。
openshift_logging_es_ops_cpu_limit	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_cpu_limit** と同等です。
openshift_logging_es_ops_memory_limit	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_memory_limit** と同等です。
openshift_logging_es_ops_pv_selector	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_pv_selector** と同等です。
openshift_logging_es_ops_pvc_dynamic	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_pvc_dynamic** と同等です。
openshift_logging_es_ops_pvc_size	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_pvc_size** と同等です。
openshift_logging_es_ops_pvc_prefix	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_pvc_prefix** と同等です。
openshift_logging_es_ops_recover_after_time	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_recovery_after_time** と同等です。
openshift_logging_es_ops_storage_group	**openshift_logging_use_ops** が **true** に設定されている場合は、Ops クラスターの **openshift_logging_es_storage_group** と同等です。
openshift_logging_es_ops_nodeselector	Elasticsearch インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。このノードセレクターは、Elasticsearch インスタンスの実行用に予約または最適化されているノード上にこれらのインスタンスを配置するために使用できます。たとえば、セレクターは **node-type=infrastructure** と指定できます。Elasticsearch をデプロイする前に、少なくとも 1 つのアクティブノードにこのラベルを付ける必要があります。

パラメーター	説明
openshift_logging_elasticsearch_kibana_index_mode	デフォルト値 **unique** では、各ユーザーが独自の Kibana インデックスを持つことができます。このモードでは、保存されたクエリー、ビジュアライゼーション、およびダッシュボードは共有されません。 値 **shared_ops** を設定することもできます。このモードでは、すべての操作ユーザーが Kibana インデックスを共有します。これにより、各操作ユーザーが同じクエリー、ビジュアライゼーション、およびダッシュボードを参照することができます。
openshift_logging_kibana_ops_nodeselector	Kibana インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。
openshift_logging_curator_ops_nodeselector	Curator インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。

Custom Certificates

デプロイメントプロセスで生成されるものを利用する代わりに、以下のインベントリー変数を使用してカスタム証明書を指定できます。カスタム証明書は、ユーザーのブラウザーと Kibana 間の通信を暗号化し、保護するために使用されます。これらが指定されない場合は、セキュリティー関連のファイルが生成されます。

ファイル名	説明
openshift_logging_kibana_cert	Kibana サーバーのブラウザー向けの証明書。
openshift_logging_kibana_key	ブラウザー向けの Kibana 証明書で使用されるキー。
openshift_logging_kibana_ca	ブラウザー向けの Kibana 証明書用に使用される CA ファイルへの制御ノード上の絶対パス。
openshift_logging_kibana_ops_cert	Ops Kibana サーバーのブラウザー向け証明書。
openshift_logging_kibana_ops_key	ブラウザー向けの Ops Kibana 証明書で使用されるキー。
openshift_logging_kibana_ops_ca	ブラウザー向けの Ops Kibana 証明書用に使用される CA ファイルへの制御ノード上の絶対パス。

32.4. EFK スタックのデプロイ

EFK スタックは Ansible Playbook を使用して EFK コンポーネントにデプロイされます。デフォルトのインベントリーファイルを使用して、デフォルトの OpenShift Ansible の場所から Playbook を実行します。

```
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
logging/config.yml
```

Playbook を実行すると、スタックをサポートするために必要なすべてのリソース (シークレット、ServiceAccount、DeploymentConfig など) がデプロイされます。Playbook はスタックが実行されるまでコンポーネント Pod のデプロイを待機します。待機手順が失敗しても、デプロイメントは成功する可能性があります。つまり、レジストリーからコンポーネントイメージを取得できる場合があります。これには数分の時間がかかる可能性があります。以下を使用してプロセスを確認できます。

```
$ oc get pods -w
```

最終的に Pod のステータスは Running になります。関連イベントの取得によるデプロイメント時の Pod のステータスを確認するには、以下を実行します。

```
$ oc describe pods/<pod_name>
```

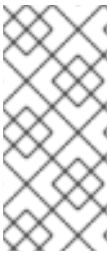
Pod の実行に失敗したら、ログを確認してください。

```
$ oc logs -f <pod_name>
```

32.5. デプロイメントの概要と調整

このセクションでは、デプロイされたコンポーネントに対して行うことができる調整について説明します。

32.5.1. Ops クラスター



注記

default、openshift、および openshift-infra プロジェクトのログが自動的に集計され、Kibana インターフェースで .operations 項目にグループ化されます。

EFK スタックをデプロイしたプロジェクト (ここでは logging) は .operations に集計されず、その ID の下に表示されます。

インベントリーファイルで `openshift_logging_use_ops` を true に設定した場合、Fluentd はメインの Elasticsearch クラスターと操作ログ用に予約された別のクラスター間でログを分割するように設定されます。この操作ログはノードシステムログおよびプロジェクト default、openshift、openshift-infra として定義されています。したがって、操作ログのインデックス化、アクセス、管理を行うために個別の Elasticsearch クラスター、個別の Kibana、および個別の Curator がデプロイされます。これらのデプロイメントは、`-ops` を含む名前によって区別されます。このオプションを有効にする場合は、これらの個別のデプロイメントに留意してください。`-ops` が含まれるという名前の変更はありますが、以下の説明の大部分が操作クラスターにも適用されます (存在する場合)。

32.5.2. Elasticsearch

高可用性環境には、それぞれが別のホスト上にある Elasticsearch のレプリカが少なくとも 3 つ必要で

す。Elasticsearch レプリカには各自のストレージが必要ですが、OpenShift Container Platform のデプロイメント設定ではすべての Pod 間でストレージボリュームを共有します。そのため、拡張時には、EFK デプロイヤーによって Elasticsearch の各レプリカに各自のデプロイメント設定が行われます。

インベントリーファイルで `openshift_logging_es_cluster_size` を変更し、ロギング Playbook を再実行することで、作成後にクラスターを拡張することができます。追加のクラスタリングパラメーターは変更可能で、「ロギング Ansible 変数の指定」で説明されています。

以下に示すようにストレージおよびネットワークの場所を選択する際の留意事項については、Elastic のドキュメントを参照してください。

すべての Elasticsearch デプロイメントの表示

現在の Elasticsearch デプロイメントをすべて表示するには、以下を実行します。

```
$ oc get dc --selector logging-infra=elasticsearch
```

ノードセレクター

Elasticsearch は大量のリソースを使用する可能性があるため、クラスターのすべてのメンバーでは、メンバー同士およびリモートストレージとのネットワーク接続の待機時間を短く設定する必要があります。待機時間を短く設定するには、ノードセレクターを使用してインスタンスを専用ノード、つまりクラスター内の専用リージョンに指定します。

ノードセレクターを設定するには、インベントリーファイルで `openshift_logging_es_nodeselector` 設定オプションを指定します。これはすべての Elasticsearch デプロイメントに適用されます。そのため、ノードセレクターを個別化する必要がある場合は、デプロイメント後に各デプロイメント設定を手動で編集する必要があります。ノードセレクターは Python と互換性のある dict (辞書) として指定されます。たとえば、`{"node-type":"infra", "region":"east"}` のようになります。

永続的な Elasticsearch ストレージ

デフォルトでは、`openshift_logging` Ansible ロールは Pod のすべてのデータが再起動時に消失する一時デプロイメントを作成します。実稼働環境での使用では、Elasticsearch デプロイメント設定ごとに永続ストレージボリュームを指定します。デプロイ前に必要な Persistent Volume Claim (永続ボリューム要求) を作成するか、またはそれらを独自に作成されるようにできます。PVC には `openshift_logging_es_pvc_prefix` 設定 (デフォルトでは `logging-es-`) と一致する名前を付ける必要があります。各 PVC 名には連番が追加され、`logging-es-1`、`logging-es-2` などのようになります。デプロイメントに必要な PVC がすでに存在する場合は、その PVC が使用されます。PVC が存在せず、`openshift_logging_es_pvc_size` が指定されている場合は、そのサイズ要求を使用して PVC が作成されます。

警告

Lucene は NFS でサポートされていないファイルシステムの動作に依存するため、NFS ストレージをボリュームまたは永続ボリュームとして (または Gluster などの NAS を介して) 使用することは Elasticsearch ストレージではサポートされません。その場合、データ破損やその他の問題が発生する可能性があります。NFS ストレージが要件である場合は、大規模なファイルをボリュームに配置してストレージデバイスとして使用し、1 つのホスト上でそれをローカルにマウントすることができます。たとえば、NFS ストレージボリュームが /nfs/storage にマウントされている場合は、以下のようになります。

```
$ truncate -s 1T /nfs/storage/elasticsearch-1
$ mkfs.xfs /nfs/storage/elasticsearch-1
$ mount -o loop /nfs/storage/elasticsearch-1 /usr/local/es-
storage
$ chown 1000:1000 /usr/local/es-storage
```

次に、以下で説明するように /usr/local/es-storage をホストマウントとして使用します。異なるバッキングファイルをそれぞれの Elasticsearch レプリカのストレージとして使用します。

このループバックはノード上で、OpenShift Container Platform の外部から手動で保守する必要があります。コンテナー内から保守することはできません。

各ノードホスト上のローカルディスクボリューム (利用可能な場合) を Elasticsearch レプリカのストレージとして使用することができます。これを実行するには、以下のような準備が必要です。

1. *関連するサービスアカウントに、ローカルボリュームをマウントし、編集する権限を指定する必要があります。*

```
$ oc adm policy add-scc-to-user privileged  \
       system:serviceaccount:logging:aggregated-logging-
elasticsearch 1
```

1 *ロギング Playbook を実行する際に、以前に作成したプロジェクトを使用します (例: logging)。*

2. *それぞれの Elasticsearch レプリカの定義にパッチを適用し、その権限を要求する必要があります。以下に例を示します (Ops クラスターの場合は `--selector component=es-ops` に変更)。*

```
$ for dc in $(oc get deploymentconfig --selector component=es -o
name); do
    oc scale $dc --replicas=0
    oc patch $dc \
       -p '{"spec":{"template":{"spec":{"containers":
[{"name":"elasticsearch","securityContext":{"privileged":
true}}]}}}}'
  done
```

3. *Elasticsearch レプリカは、ローカルストレージを使用するために適切なノード上に配置する必要があります。また、適切なノードが一定期間ダウンしている場合であっても、移動させてはなりません。この場合は、管理者がストレージを割り当てたノードに対して一意のノードセレクターを各 Elasticsearch レプリカに提供する必要があります。ノードセレクターを設定するには、各 Elasticsearch デプロイメント設定を編集し、nodeSelector セクションを追加または編集して、必要な各ノードに対して適用した一意のラベルを指定します。*

```
apiVersion: v1
kind: DeploymentConfig
spec:
  template:
    spec:
      nodeSelector:
        logging-es-node: "1" 1
```

1 *このラベル (この場合は `logging-es-node=1`) が付与されている単一のノードを持つレプリカを、ラベルによって一意に識別する必要があります。`oc label` コマンドを使用して、必要に応じてノードにラベルを適用してください。*

代わりに `oc patch` コマンドを使用して、ノードセレクターの適用を自動化することもできます。

```
$ oc patch dc/logging-es-<suffix> \
   -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-es-
node":"1"}}}}}'
```

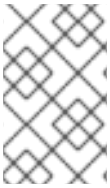
4. *これらの手順を実行すると、この例 (ストレージが各ノードの同じパスにマウントされていることを前提とします) のように、ローカルホストのマウントを各レプリカに適用できます (Ops クラスターの場合は `--selector component=es-ops` に変更します)。*

```
$ for dc in $(oc get deploymentconfig --selector component=es -o
name); do
    oc set volume $dc \
          --add --overwrite --name=elasticsearch-storage \
          --type=hostPath --path=/usr/local/es-storage
    oc rollout latest $dc
    oc scale $dc --replicas=1
  done
```

Elasticsearch のスケールの変更

クラスターで使用される Elasticsearch インスタンスの数を増やす必要がある場合、このタスクは永続ボリュームの特性や、データの保存やクラスターのリカバリーに関する Elasticsearch の設定方法により、Elasticsearch デプロイメント設定の拡張ほど簡単に実行できません。また、拡張時には各 Elasticsearch クラスターノードのデプロイメント設定を作成する必要があります。

Elasticsearch のスケールを変更する最も簡単な方法は、前述のようにインベントリーホストファイルを変更し、ロギング Playbook を再実行することです。デプロイメント用に永続ストレージが提供されていると仮定すると、この方法によって混乱が生じることはないはずです。



注記

新しい `openshift_logging_es_cluster_size` の値が、クラスターにある現在の Elasticsearch のノード数 (スケールアップした後) より大きい場合のみ、ロギング Playbook を使用して Elasticsearch クラスターのサイズを調節できます。

カスタマイズを維持する必要があるなどの理由で再インストールを希望しない場合は、デプロイヤーによって提供されるテンプレートを使用して、新規の Elasticsearch デプロイメント設定をクラスターに追加することができます。ただし、これにはより複雑な手順が必要になります。

ルートとしての Elasticsearch の公開

デフォルトでは、OpenShift の集計ロギングでデプロイされた Elasticsearch はロギングクラスターの外部からアクセスできません。データにアクセスする必要があるツールに対しては、Elasticsearch への外部アクセスのルートを有効にすることができます。

OpenShift トークンを使用して Elasticsearch にアクセスできます。また、サーバー証明書を作成する際に (Kibana と同様に)、外部の Elasticsearch および Elasticsearch Ops ホスト名を指定することができます。

1. *re-encrypt ルートとして Elasticsearch にアクセスするには、以下の変数を定義します。*

```
openshift_logging_es_allow_external=True
openshift_logging_es_hostname=elasticsearch.example.com
```

2. *以下の Ansible Playbook を実行します。*

```
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
logging/config.yml
```

3. *Elasticsearch にリモートでログインするには、要求に 3 つの HTTP ヘッダーが含まれている必要があります。*

```
Authorization: Bearer $token
X-Proxy-Remote-User: $username
X-Forwarded-For: $ip_address
```

4. *ログにアクセスできるようになるには、プロジェクトへのアクセスが必要です。以下は例になります。*

```
$ oc login <user1>
$ oc new-project <user1project>
$ oc new-app <httpd-example>
```

5. *要求内で使用するこの ServiceAccount のトークンを取得する必要があります。*

```
$ token=$(oc whoami -t)
```

6. *以前に設定したトークンを使用して、公開ルート経由で Elasticsearch にアクセスできる必要があります。*

```
$ curl -k -H "Authorization: Bearer $token" -H "X-Proxy-Remote-User:
$(oc whoami)" -H "X-Forwarded-For: 127.0.0.1"
```

```
https://es.example.test/project.my-project.*/_search?q=level:err |
python -mjson.tool
```

32.5.3. Fluentd

Fluentd はノードラベルセレクターに従ってレプリカをデプロイする DeamonSet としてデプロイされます。ノードラベルセレクターはインベントリーパラメーター `openshift_logging_fluentd_nodeselector` を使用して指定することができ、デフォルトは `logging-infra-fluentd` です。OpenShift クラスターのインストールの一環として、Fluentd ノードセレクターを永続化されたノードラベルの一覧に追加することが推奨されます。

Fluentd は `journald` をシステムログソースとして使用します。これらは、オペレーティングシステム、コンテナーランタイム、および OpenShift からのログメッセージです。

利用可能なコンテナーのランタイムでは、ログメッセージのソースを特定するための最小限の情報が提供されます。ログ収集およびログの正規化は、Pod が削除されてから行われるので、ラベルまたはアノテーションなどの追加のメタデータが API サーバーから取得できません。

ログコレクターがログの処理を完了する前に、特定の名前および namespace が指定された Pod が削除された場合には、よく似た名前の Pod および namespace とログメッセージを区別する方法がなくなる可能性があります。これにより、ログがインデックス化され、Pod をデプロイしたユーザーが所有していないインデックスにアノテーションが付けられる可能性があります。

重要

利用可能なコンテナーのランタイムでは、ログメッセージのソースを特定するための最小限の情報が提供され、個別のログメッセージが一意となる確証はなく、これらのメッセージにより、その出処を突き止められるわけではありません。

OpenShift Container Platform 3.9 のクリーンインストールではデフォルトのログドライバーとして `json-file` を使用します。ただし、OpenShift Container Platform 3.7 からアップグレードされた環境では既存の `journald` ログドライバーの設定が維持されます。`json-file` ログドライバーを使用することが推奨されます。既存のログドライバー設定を `json-file` に変更する手順については、「集計されたロギングドライバーの変更」を参照してください。

外部ログアグリゲーターにログを送信するための Fluentd の設定

`secure-forward` プラグインを使用して、デフォルトの Elasticsearch ではなく外部のログアグリゲーターにログのコピーを送信するように Fluentd を設定することができます。ここから、ローカルでホストされている Fluentd によるログレコードの処理後に、さらなるログの処理が可能です。

ロギングデプロイメントでは、外部アグリゲーターを設定するための `secure-forward.conf` セクションが Fluentd configmap に用意されています。

```
<store>
@type secure_forward
self_hostname pod-${HOSTNAME}
shared_key thisisasharedkey
secure yes
enable_strict_verification yes
ca_cert_path /etc/fluent/keys/your_ca_cert
ca_private_key_path /etc/fluent/keys/your_private_key
ca_private_key_passphrase passphrase
<server>
```

```
  host ose1.example.com
  port 24284
</server>
<server>
  host ose2.example.com
  port 24284
  standby
</server>
<server>
  host ose3.example.com
  port 24284
  standby
</server>
</store>
```

oc edit コマンドを使用して更新できます。

```
$ oc edit configmap/logging-fluentd
```

secure-forward.conf で使用される証明書を Fluentd Pod にマウントされる既存のシークレットに追加することができます。your_ca_cert と your_private_key の値は configmap/logging-fluentd の secure-forward.conf で指定されている値と一致している必要があります。

```
$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_ca_cert','value':'$(base64
/path/to/your_ca_cert.pem)'}]"
$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_private_key','value':'$(base64
/path/to/your_private_key.pem)'}]"
```

注記

your_private_key は、汎用名に置き換えます。これは、JSON パスへのリンクで、お使いのホストシステムのパスではありません。

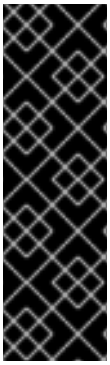
外部アグリゲーターを設定する際は、Fluentd からのメッセージを安全に受信できる必要があります。

外部アグリゲーターが別の Fluentd サーバーである場合、fluent-plugin-secure-forward プラグインがインストールされていて、それによって提供される入力プラグインを使用する必要があります。

```
<source>
  @type secure_forward

  self_hostname ${HOSTNAME}
  bind 0.0.0.0
  port 24284

  shared_key thisisasharedkey

  secure yes
  cert_path        /path/for/certificate/cert.pem
  private_key_path /path/for/certificate/key.pem
  private_key_passphrase secret_foo_bar_baz
</source>
```

- *fluent-plugin-secure-forward リポジトリーに fluent-plugin-secure-forward プラグインの設定方法に関する詳細の説明があります。*

Fluentd から API サーバーへの接続数の削減



重要

mux はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

mux は Secure Forward のリスナーサービスです。

パラメーター	説明
openshift_logging_use_mux	デフォルトは **False** に設定されています。**True** に設定されている場合は、**mux** というサービスがデプロイされます。このサービスは、クラスターで実行されるノードエージェント Fluentd deamonset の Fluentd **secure_forward** アグリゲーターとして機能します。**openshift_logging_use_mux** を使用して OpenShift API サーバーへの接続数を減らし、生ログを **mux** に送信して Kubernetes メタデータプラグインをオフにするように Fluentd の各ノードを設定します。これには、**openshift_logging_mux_client_mode** を使用する必要があります。
openshift_logging_mux_client_mode	**openshift_logging_mux_client_mode** の値は **minimal** と **maximal** です。デフォルトはありません。**openshift_logging_mux_client_mode** により、Fluentd ノードエージェントは、ログを Elasticsearch に直接送信するのではなく mux に送信します。値 **maximal** は、Fluentd がレコードを **mux** に送信する前にできるだけ多くの処理をノードで実行することを意味しています。**maximal** 値は **mux** を使用する場合に推奨されます。値 **minimal** は、Fluentd がまったく処理を行わないことを意味しており、処理用に生ログを **mux** に送信します。**minimal** 値の使用は推奨されません。
openshift_logging_mux_allow_external	デフォルトは **False** に設定されます。**True** に設定されている場合は、**mux** サービスがデプロイされ、クラスターの外部で実行されている Fluentd クライアントが **secure_forward** を使用してログを送信できるように設定されます。これにより、OpenShift ロギングを OpenShift 以外のクライアント、または他の OpenShift クラスターの中央ロギングサービスとして使用することができます。
openshift_logging_mux_hostname	デフォルトは **mux** に **openshift_master_default_subdomain** を追加した値です。これは、**external_clients** が **mux** に接続するために使用するホスト名であり、TLS サーバーの証明書サブジェクトで使用されます。

パラメーター	説明
openshift_logging_mux_port	24284
openshift_logging_mux_cpu_limit	500M
openshift_logging_mux_memory_limit	1Gi
openshift_logging_mux_default_namespaces	デフォルトは **mux-undefined** です。一覧の最初の値は未定義のプロジェクトに使用する namespace であり、これにデフォルトで作成する追加の namespace が続きます。通常、この値を設定する必要はありません。
openshift_logging_mux_namespaces	デフォルト値は空であり、外部の **mux** クライアント向けに追加の namespace を作成し、ログと関連付けることができます。この値を設定する必要があります。

Fluentd でのログのスロットリング

特に冗長なプロジェクトについては、管理者は処理される前の Fluentd によるログの読み取り速度を減速することができます。

警告

スロットリングは設定されたプロジェクトのログ集計が遅れる一因になる可能性があります。Fluentd が追い付く前に Pod が削除された場合、ログエントリーが消失する可能性があります。

注記

Systemd ジャーナルをログソースとして使用している場合、スロットリングは機能しません。スロットリングの実装は、各プロジェクトの個々のログファイルの読み取りを調整できる機能によって決まります。ジャーナルからの読み取り時に、単一のログソースしか存在せず、ログファイルは存在しないと、ファイルベースのスロットリングは利用できません。Fluentd プロセスに読み込まれるログエントリーを制限する方法はありません。

Fluentd に対して制限する必要があるプロジェクトを指示するには、デプロイメント後に ConfigMap のスロットル設定を編集します。

```
$ oc edit configmap/logging-fluentd
```

throttle-config.yaml キーの形式は、プロジェクト名と、各ノードでのログの読み取りに必要な速度が含まれる YAML ファイルです。デフォルトはノードごとに一度に 1,000 行です。以下に例を示します。

```
logging:
  read_lines_limit: 500

test-project:
  read_lines_limit: 10

.operations:
  read_lines_limit: 100
```

EFK スタックのいずれかの部分を変更する場合は (具体的には Elasticsearch または Fluentd)、まず Elasicsearch をゼロにスケールダウンして、他のノードと一致しないように Fluentd を調整する必要があります。次に、変更を行って、Elasicsearch と Fluentd のスケールを元に戻します。

Elasicsearch をゼロに調整するには、以下を実行します。

```
$ oc scale --replicas=0 dc/<ELASTICSEARCH_DC>
```

デーモンセット設定の nodeSelector がゼロに一致するように変更します。

Fluentd ノードセレクターの取得:

```
$ oc get ds logging-fluentd -o yaml |grep -A 1 Selector
     nodeSelector:
       logging-infra-fluentd: "true"
```

oc patch コマンドでの deamonset の nodeSelector 変更:

```
$ oc patch ds logging-fluentd -p '{"spec":{"template":{"spec":
{"nodeSelector":{"nonexistlabel":"true"}}}}}'
```

Fluentd ノードセレクターの取得:

```
$ oc get ds logging-fluentd -o yaml |grep -A 1 Selector
     nodeSelector:
       "nonexistlabel: "true"
```

Elastcsearch のサイズをゼロから元に戻します。

```
$ oc scale --replicas=# dc/<ELASTICSEARCH_DC>
```

deamonset 設定の nodeSelector を変更して logging-infra-fluentd: "true" に戻します。

oc patch コマンドでの deamonset の nodeSelector 変更:

```
oc patch ds logging-fluentd -p '{"spec":{"template":{"spec":
{"nodeSelector":{"logging-infra-fluentd":"true"}}}}}'
```

32.5.4. Kibana

OpenShift Container Platform の Web コンソールから Kibana コンソールにアクセスするには、マスター webconsole-config configmap ファイルに `loggingPublicURL` パラメーターを追加し、Kibana コンソールの URL (`kibana-hostname` パラメーター) を指定します。値は HTTPS URL である必要があります。

```
...
clusterInfo:
  ...
  loggingPublicURL: "https://kibana.example.com"
...
```

`loggingPublicURL` パラメーターを設定すると、OpenShift Container Platform Web コンソールの Browse → Pods → <pod_name> → Logs タブに View Archive ボタンが作成されます。このボタンは Kibana コンソールにリンクします。

通常通り Kiabana デプロイメントを拡張して冗長性を実現できます。

```
$ oc scale dc/logging-kibana --replicas=2
```

注記

ロギング Playbook の複数の実行にわたってスケールを維持するには、インベントリーファイルの `openshift_logging_kibana_replica_count` を必ず更新してください。

`openshift_logging_kibana_hostname` 変数によって指定されたサイトにアクセスすることで、ユーザーインターフェースを確認できます。

Kibana に関する詳細については、Kibana のドキュメントを参照してください。

Kibana Visualize

Kibana Visualize を使用すると、視覚化機能やダッシュボードを作成してコンテナーを監視できます。また Pod ログにより、管理者ユーザー (`cluster-admin` または `cluster-reader`) はデプロイメント、namespace、Pod、およびコンテナーごとにログを表示することができます。

Kibana Visualize は Elasticsearch および ES-OPS Pod 内に存在し、それらの Pod 内で実行する必要があります。ダッシュボードとその他の Kibana UI オブジェクトを読み込むには、まずはダッシュボードに追加するユーザーとして Kibana にログインしてから、ログアウトする必要があります。これにより、以下の手順で使用するユーザーごとの必要な設定が作成されます。次に、以下を実行します。

```
$ oc exec <$espod> -- es_load_kibana_ui_objects <user-name>
```

ここで、$espod はいずれかの Elasticsearch Pod の名前です。

32.5.5. Curator

Curator を利用することで、管理者はスケジュールされた Elasticsearch のメンテナンス操作を設定し、プロジェクトごとに自動的に実行することができます。これは、設定に基づいてアクションを毎日実行するようにスケジュール設定されています。Elasticsearch クラスターごとに 1 つの Curator Pod のみを使用することが推奨されます。Curator は以下の構造を持つ YAML 設定ファイルで設定されます。

```
$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE

$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE
 ...
```

利用可能なパラメーターを以下に示します。

変数名	説明
PROJECT_NAME	プロジェクトの実際の名前 (**myapp-devel** など)。OpenShift Container Platform の**操作**ログについては、名前 **.operations** をプロジェクト名として使用します。
ACTION	実行するアクション。現在許可されているのは **delete** のみです。
UNIT	**days**、**weeks**、または **months** のいずれか。
VALUE	単位数を示す整数。
.defaults	**.defaults** を **$PROJECT_NAME** として使用して、指定されていないプロジェクトのデフォルトを設定します。
runhour	(数値) Curator ジョブを実行する時刻 (時、24 時間形式)。**.defaults** で使用します。
runminute	(数値) Curator ジョブを実行する時刻 (分)。**.defaults** で使用します。
timezone	(文字列) tzselect(8) または timedatectl(1) 形式での文字列。デフォルトのタイムゾーンは UTC です。
.regex	プロジェクト名に一致する正規表現の一覧。
pattern	適切にエスケープされた有効な正規表現パターン。一重引用符で囲まれています。

たとえば、以下のように Curator を設定します。

- *1 day を経過した myapp-dev プロジェクトのインデックスを削除する*
- *1 week を経過した myapp-qe プロジェクトのインデックスを削除する*
- *8 weeks を経過したoperations ログを削除する*
- *31 days を経過したその他すべてのプロジェクトのインデックスを削除する*
- *毎日午前 0 時に Curator ジョブを実行する*

以下を使用します。

```
config.yaml: |
  myapp-dev:
    delete:
      days: 1

  myapp-qe:
    delete:
      weeks: 1

  .operations:
    delete:
      weeks: 8

  .defaults:
    delete:
      days: 31
    runhour: 0
    runminute: 0
    timezone: America/New_York

  .regex:
    - pattern: '^project\..+\-dev\..*$'
      delete:
        days: 15
    - pattern: '^project\..+\-test\..*$'
      delete:
        days: 30
```

重要

`month` を操作の `$UNIT` として使用する場合、Curator は今月の当日ではなく、今月の最初の日からカウントを開始します。たとえば、今日が 4 月 15 日であり、現時点で 2 カ月を経過したインデックスを削除する場合 (`delete: months: 2`)、Curator は 2 月 15 日より古い日付のインデックスを削除するのではなく、2 月 1 日より古いインデックスを削除します。つまり、今月の最初の日付まで遡り、次にその日付から丸 2 カ月遡ります。Curator で厳密な指定を行うための最適な方法として、日数で指定することができます (例: `delete: days: 30`)。

32.5.5.1. Curator 設定の作成

`openshift_logging` Ansible ロールは、Curator が設定の読み取りに使用する ConfigMap を提供します。この ConfigMap を編集するか、または置き換えることで、Curator を再設定することができます。現時点で、Ops および非 Ops 両方の Curator インスタンスを設定するために `logging-curator` ConfigMap が使用されています。`.operations` 設定はすべてアプリケーションログ設定と同じ場所にあります。

1. 提供された ConfigMap を編集して Curator インスタンスを設定するには、以下を実行します。

```
$ oc edit configmap/logging-curator
```

2. 提供された ConfigMap を置換するには、以下を実行します。

```
$ create /path/to/mycuratorconfig.yaml
$ oc create configmap logging-curator -o yaml \
  --from-file=config.yaml=/path/to/mycuratorconfig.yaml | \
  oc replace -f -
```

3. *変更を行った後に、Curator を再デプロイします。*

```
$ oc rollout latest dc/logging-curator
$ oc rollout latest dc/logging-curator-ops
```

32.6. クリーンアップ

デプロイメント中に生成されたものをすべて削除します。

```
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
logging/config.yml \
    -e openshift_logging_install_logging=False
```

32.7. KIBANA のトラブルシューティング

OpenShift Container Platform で Kibana コンソールを使用することで発生する可能性がある問題は簡単に解決できますが、この場合役に立つエラーメッセージは表示されません。OpenShift Container Platform に Kibana をデプロイする際に問題が発生した場合は、以下のトラブルシューティングセクションを確認してください。

ログインループ

Kibana コンソールの OAuth2 プロキシーはマスターホストの OAuth2 サーバーとシークレットを共有する必要があります。シークレットが両方のサーバーで同一でない場合はログインループが発生する可能性があり、Kibana のログインページに継続的にリダイレクトされます。

この問題を修正するには、現在の OAuthClient を削除し、`openshift-ansible` を使用して `openshift_logging` ロールを再実行します。

```
$ oc delete oauthclient/kibana-proxy
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
logging/config.yml
```

コンソール表示時の不明なエラー

Kibana コンソールにアクセスしようとする際に、以下のブラウザーエラーが表示される場合があります。

```
{"error":"invalid_request","error_description":"The request is missing a
required parameter,
 includes an invalid parameter value, includes a parameter more than once,
or is otherwise malformed."}
```

このエラーは、OAuth2 クライアントとサーバー間の不一致が原因で発生します。ログイン後にサーバーが安全にリダイレクトできるように、クライアントのリターンアドレスがホワイトリストで指定されている必要があります。

この問題を修正するには、OAuthClient エントリーを置き換えます。

```
$ oc delete oauthclient/kibana-proxy
$ ansible-playbook [-i </path/to/inventory>] \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
logging/config.yml
```

問題が解決しない場合は、OAuth クライアントに一覧表示されている URL の Kibana にアクセスしていることを確認してください。この問題は、転送先ポート (標準の 443 HTTPS ポートではなく 1443 など) の URL にアクセスすることで発生する可能性があります。OAuth クライアントを編集することで、サーバーのホワイトリストを調整できます。

```
$ oc edit oauthclient/kibana-proxy
```

コンソール表示時の 503 エラー

Kibana コンソールを表示する時にプロキシーエラーが発生する場合は、2 つの問題のうちのいずれかが原因である可能性があります。

1 つ目の問題は、Kibana が Pod を認識していない可能性があります。Elasticsearch の起動が遅い場合、Kibana はそれに到達しようとしてタイムアウトする場合があります。関連サービスにエンドポイントがあるかどうか確認してください。

```
$ oc describe service logging-kibana
Name:                   logging-kibana
[...]
Endpoints:              <none>
```

Kibana Pod が実行中である場合、エンドポイントが一覧表示されます。実行中でない場合は、Kibana Pod とデプロイメントの状態を確認してください。デプロイメントをスケールダウンして、再び元に戻さなければならない場合があります。

考えられる 2 つ目の問題として、Kibana サービスにアクセスするためのルートがマスクされている場合に発生する可能性があります。これは、あるプロジェクトでテストデプロイメントを実行し、次に最初のデプロイメントを完全に削除することなく別のプロジェクトでデプロイした場合に発生する可能性があります。複数のルートが同じ宛先に送信される場合、デフォルトルーターは最初に作成されたルートにのみルーティングします。問題が発生するルートをチェックして、そのルートが複数の場所で定義されているかどうかを確認してください。

```
$ oc get route  --all-namespaces --selector logging-infra=support
```

F-5 ロードバランサーと有効にされた X-Forwarded-For

`X-Forwarded-For` が有効にされた Kibana の前に F-5 ロードバランサーの使用を試みる場合、Elasticsearch `Searchguard` プラグインが Kibana からの接続を適切に受け取ることができないという問題が発生する可能性があります。

Kibana のエラーメッセージの例

```
Kibana: Unknown error while connecting to Elasticsearch
```

```
Error: Unknown error while connecting to Elasticsearch
Error: UnknownHostException[No trusted proxies]
```

余分なヘッダーを無視するように Searchguard を設定するには、以下の手順を実行します。

1. すべての Fluentd Pod をスケールダウンします。
2. Fluentd Pod の終了後に Elasticsearch をスケールダウンします。
3. `searchguard.http.xforwardedfor.header: DUMMY` を Elasticsearch の設定セクションに追加します。

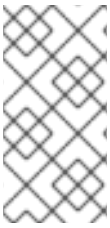
```
$ oc edit configmap/logging-elasticsearch 1
```

1 この方法では、Elasticsearch の設定が ConfigMap に含まれている必要があります。

4. Elasticsearch を拡張して元に戻します。
5. すべての Fluentd Pod を拡張します。

32.8. 外部 ELASTICSEARCH インスタンスへのログの送信

Fluentd は、Elasticsearch デプロイメント設定の `ES_HOST`、`ES_PORT`、`OPS_HOST`、および `OPS_PORT` 環境変数の値にログを送信します。アプリケーションログは `ES_HOST` の宛先に、操作ログは `OPS_HOST` の宛先に送信されます。



注記

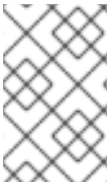
AWS Elasticsearch インスタンスへのログの直接送信はサポートされていません。Fluentd Secure Forward を使用して、`fluent-plugin-aws-elasticsearch-service` プラグインで設定した制御対象の Fluentd のインスタンスにログを送信してください。

ログを特定の Elasticsearch インスタンスに送信するには、デプロイメント設定を編集して、上記の変数の値を必要なインスタンスに置き換えます。

```
$ oc edit dc/<deployment_configuration>
```

外部 Elasticsearch インスタンスにアプリケーションログと操作ログの両方を含めるには、`ES_HOST` と `OPS_HOST` を同じ宛先に設定して、`ES_PORT` と `OPS_PORT` にも同一の値があるようにします。

外部でホストされている Elasticsearch インスタンスが TLS を使用していない場合、`_CLIENT_CERT`、`_CLIENT_KEY`、`_CA` 変数を空になるように更新します。TLS を使用していてもそれが Mutual TLS ではない場合は、`_CLIENT_CERT` と `_CLIENT_KEY` 変数を空になるように更新し、logging-fluentd シークレットについて、Elasticsearch インスタンスと通信するための適切な `_CA` 値でパッチを適用するか再作成します。指定された Elasticsearch インスタンスと同様に Mutual TLS を使用している場合、logging-fluentd シークレットについて、クライアントキー、クライアント証明書、CA を使ってパッチを適用するか再作成します。



注記

指定された Kibana と Elasticsearch イメージを使用していない場合、同じマルチテナント機能は利用できず、データは特定のプロジェクトへのユーザーアクセスによる制限を受けません。

32.9. 外部 SYSLOG サーバーへのログの送信

`fluent-plugin-remote-syslog` プラグインをホストで使用して、ログを外部 syslog サーバーに送信します。

環境変数を `logging-fluentd` または `logging-mux` デプロイメント設定で設定します。

```
- name: REMOTE_SYSLOG_HOST (1)
  value: host1
- name: REMOTE_SYSLOG_HOST_BACKUP
  value: host2
- name: REMOTE_SYSLOG_PORT_BACKUP
  value: 5555
```

(1) *必要なリモート syslog ホスト。各ホストで必須です。*

これによって 2 つの宛先が作成されます。`host1` の syslog サーバーはデフォルトポート `514` でメッセージを受信し、`host2` は同じメッセージをポート `5555` で受信します。

または、独自のカスタム fluent.conf *を `logging-fluentd` または `logging-mux` ConfigMap に設定できます。*

Fluentd 環境変数

パラメーター	説明
USE_REMOTE_SYSLOG	デフォルトは **false** です。**fluent-plugin-remote-syslog** gem を使用できるようにするには、**true** に設定します。
REMOTE_SYSLOG_HOST	(必須) リモート syslog サーバーのホスト名または IP アドレス。
REMOTE_SYSLOG_PORT	接続先のポート番号。デフォルトは **514** です。
REMOTE_SYSLOG_SEVERITY	syslog の重大度を設定します。デフォルトは **debug** です。
REMOTE_SYSLOG_FACILITY	syslog ファシリティーを設定します。デフォルトは **local0** です。
REMOTE_SYSLOG_USE_RECORD	デフォルトは **false** です。レコードの重大度フィールドおよびファシリティーフィールドを使用して syslog メッセージに設定するには、**true** に設定します。
REMOTE_SYSLOG_REMOVE_TAG_PREFIX	タグからプレフィックスを削除します。デフォルトは **''** (空) です。

パラメーター	説明
REMOTE_SYSLOG_TAG_KEY	これが指定されている場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにタグを設定します。
REMOTE_SYSLOG_PAYLOAD_KEY	これが指定されている場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにペイロードを設定します。

警告

この実装は安全ではないため、接続にスヌーピングがないことを保証できる環境でのみ使用してください。

Fluentd ロギング Ansible 変数

パラメーター	説明
openshift_logging_fluentd_remote_syslog	デフォルトは **false** に設定されます。fluent-plugin-remote-syslog gem を使用できるようにするには、**true** に設定します。
openshift_logging_fluentd_remote_syslog_host	リモート syslog サーバーのホスト名または IP アドレス。必須です。
openshift_logging_fluentd_remote_syslog_port	接続先のポート番号。デフォルトは **514** です。
openshift_logging_fluentd_remote_syslog_severity	syslog の重大度を設定します。デフォルトは **debug** です。
openshift_logging_fluentd_remote_syslog_facility	syslog ファシリティーを設定します。デフォルトは **local0** です。
openshift_logging_fluentd_remote_syslog_use_record	デフォルトは **false** に設定されます。レコードの重大度フィールドおよびファシリティーフィールドを使用して syslog メッセージに設定するには、**true** に設定します。

パラメーター	説明
openshift_logging_fluentd_remote_syslog_remove_tag_prefix	タグからプレフィックスを削除します。デフォルトは **''** (空) です。
openshift_logging_fluentd_remote_syslog_tag_key	文字列が指定された場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにタグを設定します。
openshift_logging_fluentd_remote_syslog_payload_key	文字列が指定された場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにペイロードを設定します。

Mux ロギング Ansible 変数

パラメーター	説明
openshift_logging_mux_remote_syslog	デフォルトは **false** に設定されます。fluent-plugin-remote-syslog gem を使用できるようにするには、**true** に設定します。
openshift_logging_mux_remote_syslog_host	リモート syslog サーバーのホスト名または IP アドレス。必須です。
openshift_logging_mux_remote_syslog_port	接続先のポート番号。デフォルトは **514** です。
openshift_logging_mux_remote_syslog_severity	syslog の重大度を設定します。デフォルトは **debug** です。
openshift_logging_mux_remote_syslog_facility	syslog ファシリティーを設定します。デフォルトは **local0** です。
openshift_logging_mux_remote_syslog_use_record	デフォルトは **false** に設定されます。レコードの重大度フィールドおよびファシリティーフィールドを使用して syslog メッセージに設定するには、**true** に設定します。
openshift_logging_mux_remote_syslog_remove_tag_prefix	タグからプレフィックスを削除します。デフォルトは **''** (空) です。
openshift_logging_mux_remote_syslog_tag_key	文字列が指定された場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにタグを設定します。

パラメーター	説明
`openshift_logging_mux_remote_syslog_payload_key`	文字列が指定された場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにペイロードを設定します。

32.10. ELASTICSEARCH 管理操作の実行

Elasticsearch と通信して管理操作を実行するのに使用する管理者の証明書、キー、CA は、logging-elasticsearch シークレット内にあります。

注記

これらが EFK インストールにあるかどうかを確認するには、以下のコマンドを実行します。

```
$ oc describe secret logging-elasticsearch
```

1. *メンテナンスを実行しようとしているクラスター内にある Elasticsearch Pod に接続します。*
2. *Pod をクラスター内で検索するには、以下のいずれかのコマンドを使用します。*

```
$ oc get pods -l component=es -o name | head -1
$ oc get pods -l component=es-ops -o name | head -1
```

3. *Pod に接続します。*

```
$ oc rsh <your_Elasticsearch_pod>
```

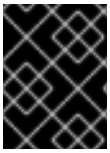
4. *Elasticsearch コンテナーに接続すると、インデックス API のマニュアルに従い、シークレットからマウントされた証明書を使用して Elasticsearch と通信することができます。*
   Fluentd では、インデックス形式 project.{project_name}.{project_uuid}.YYYY.MM.DD を使用してログを Elasticsearch に送信します。ここで、YYYY.MM.DD はログレコードの日付です。

   たとえば uuid が 3b3594fa-2ccd-11e6-acb7-0eb6b35eaee3 の logging プロジェクトから 2016 年 6 月 15 日以降のすべてのログを削除するには、以下のコマンドを実行します。

```
$ curl --key /etc/elasticsearch/secret/admin-key \
  --cert /etc/elasticsearch/secret/admin-cert \
  --cacert /etc/elasticsearch/secret/admin-ca -XDELETE \
  "https://localhost:9200/project.logging.3b3594fa-2ccd-11e6-acb7-
0eb6b35eaee3.2016.06.15"
```

32.11. 集計されたロギングのドライバーの変更

集計されたロギングについては、`json-file` ログドライバーの使用を推奨します。



重要

json-file ドライバーを使用する場合は、Docker バージョン docker-1.12.6-55.gitc4618fb.el7_4 now 以降を使用していることを確認してください。

Fluentd では、/etc/docker/daemon.json ファイルおよび /etc/sysconfig/docker ファイルをチェックして、Docker が使用しているドライバーを判別します。

docker info コマンドを使用すると、Docker が使用しているドライバーを確認できます。

```
# docker info | grep Logging

Logging Driver: journald
```

json-file に変更するには、以下の手順に従います。

1. /etc/sysconfig/docker ファイルまたは /etc/docker/daemon.json ファイルのいずれかを変更します。
以下は例を示しています。

```
# cat /etc/sysconfig/docker
OPTIONS=' --selinux-enabled --log-driver=json-file --log-opt max-size=1M --log-opt max-file=3 --signature-verification=False'

cat /etc/docker/daemon.json
{
"log-driver": "json-file",
"log-opts": {
"max-size": "1M",
"max-file": "1"
}
}
```

2. Docker サービスを再起動します。

```
systemctl restart docker
```

3. Fluentd を再起動します。

警告

13 以上のノードで一度に Fluentd を再起動すると、Kubernetes スケジューラーに大きな負荷が生成されます。以下の手順に従って Fluentd を再起動する場合は、細心の注意を払ってください。

Fluentd を再起動する方法は 2 つあります。1 つのノードまたはノードセット上で Fluentd を再起動するか、またはすべてのノードで Fluentd を再起動できます。

a. *以下の手順は、1 つのノードまたはノードセット上で Fluentd を再起動する方法を示しています。*

i. *Fluentd が実行しているノードの一覧を表示します。*

```
$ oc get nodes -l logging-infra-fluentd=true
```

ii. *各ノードについて、ラベルを削除して Fluentd をオフにします。*

```
$ oc label node $node logging-infra-fluentd-
```

iii. *Fluentd がオフになっていることを確認します。*

```
$ oc get pods -l component=fluentd
```

iv. *各ノードについて Fluentd を再起動します。*

```
$ oc label node $node logging-infra-fluentd=true
```

b. *以下の手順は、すべてのノード上で Fluentd を再起動する方法を示しています。*

i. *すべてのノードで Fluentd をオフにします。*

```
$ oc label node -l logging-infra-fluentd=true --overwrite
logging-infra-fluentd=false
```

ii. *Fluentd がオフになっていることを確認します。*

```
$ oc get pods -l component=fluentd
```

iii. *すべてのノードで Fluentd を再起動します。*

```
$ oc label node -l logging-infra-fluentd=false --overwrite
logging-infra-fluentd=true
```

iv. *Fluentd がオンになっていることを確認します。*

```
$ oc get pods -l component=fluentd
```

32.12. ELASTICSEARCH の手動ロールアウト

OpenShift Container Platform 3.7 では Aggregated Logging スタックで Elasticsearch Deployment Config オブジェクトが更新され、Config Change Trigger が除外されました。このため、dc を変更しても自動ロールアウトは実行されなくなります。これは、ES クラスターで意図しない再起動を回避するものでしたが、クラスターメンバーの再起動時にシャードのリバランスが過剰に発生しまう可能性があります。

このセクションでは、ローリング再起動とフル再起動の 2 つの再起動手順を説明します。ローリング再起動はダウンタイムを発生させずに Elasticsearch クラスターに適切な変更を適用し (3 つのマスターが設定されている場合)、フル再起動は既存データを損なわずに大規模な変更を安全に適用します。

32.12.1. Elasticsearch ローリングクラスター再起動の実行

以下のいずれかの変更を行う場合は、ローリング再起動を推奨します。

- Elasticsearch Pod の実行で再起動が必要なノード
- logging-elasticsearch configmap
- logging-es-* デプロイメント設定
- 新規イメージのデプロイメントまたはアップグレード

これは今後推奨される再起動ポリシーになります。

注記

openshift_logging_use_ops が True に設定される場合、ES クラスターへのアクションを ops クラスターに対して繰り返す必要があります。

1. ノードを意図的に停止する際のシャードのバランシングを防止します。

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> --
          curl -s
          --cacert /etc/elasticsearch/secret/admin-ca \
          --cert /etc/elasticsearch/secret/admin-cert \
          --key /etc/elasticsearch/secret/admin-key \
          -XPUT 'https://localhost:9200/_cluster/settings' \
          -d '{ "transient": { "cluster.routing.allocation.enable" :
"none" } }'
```

2. 完了したら、ES クラスターの各 dc について oc rollout latest を実行して、最新バージョンの dc オブジェクトをデプロイします。

```
$ oc rollout latest <dc_name>
```

新しい Pod がデプロイされます。Pod に 2 つのコンテナーが準備できたら、以下の dc に進むことができます。

3. クラスターのすべての「dc」がロールアウトされたら、シャードバランシングを再度有効にします。

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> --
          curl -s
          --cacert /etc/elasticsearch/secret/admin-ca \
          --cert /etc/elasticsearch/secret/admin-cert \
          --key /etc/elasticsearch/secret/admin-key \
          -XPUT 'https://localhost:9200/_cluster/settings' \
          -d '{ "transient": { "cluster.routing.allocation.enable" :
"all" } }'
```

32.12.2. Elasticsearch フルクラスター再起動の実行

Elasticsearch のメジャーバージョンの変更など、変更プロセス中にデータ整合性が損なわれる危険性がある変更の場合は、フル再起動を推奨します。

注記

openshift_logging_use_ops が True に設定される場合、ES クラスターへのアクションを ops クラスターに対して繰り返す必要があります。

注記

logging-es-ops サービスに変更を行う場合は、パッチとして代わりにコンポーネント "es-ops-blocked" および "es-ops" を使用します。

1. *ES クラスターが停止しているときに、ES クラスターへのすべての外部通信を無効にします。以下のコマンドを実行して、非クラスターロギングサービス (logging-es や logging-es-ops など) を編集して、ES Pod に一致しないようにします。*

```
$  oc patch svc/logging-es -p '{"spec":{"selector":{"component":"es-
blocked","provider":"openshift"}}}'
```

2. *シャードの同期フラッシュを実行して、シャットダウン前のディスクへの書き込みを待機している保留中の操作がないようにします。*

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> --
          curl -s
          --cacert /etc/elasticsearch/secret/admin-ca \
          --cert /etc/elasticsearch/secret/admin-cert \
          --key /etc/elasticsearch/secret/admin-key \
          -XPOST 'https://localhost:9200/_flush/synced'
```

3. *ノードを意図的に停止する際のシャードのバランシングを防止します。*

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> --
          curl -s
          --cacert /etc/elasticsearch/secret/admin-ca \
          --cert /etc/elasticsearch/secret/admin-cert \
          --key /etc/elasticsearch/secret/admin-key \
          -XPUT 'https://localhost:9200/_cluster/settings' \
          -d '{ "transient": { "cluster.routing.allocation.enable" :
"none" } }'
```

4. *完了したら、ES クラスターの各 dc について oc rollout latest を実行して、最新バージョンの dc オブジェクトをデプロイします。*

```
$ oc rollout latest <dc_name>
```

新しい Pod がデプロイされます。Pod に 2 つのコンテナーが準備できたら、以下の dc に進むことができます。

5. *再起動が完了したら、ES クラスターへのすべての外部通信を有効にします。以下のコマンドを再度実行して、非クラスターロギングサービス (logging-es や logging-es-ops など) を編集して、ES Pod に一致するようにします。*

```
$ oc patch svc/logging-es -p '{"spec":{"selector":
{"component":"es","provider":"openshift"}}}'
```

第33章 集計ロギングのサイジングに関するガイドライン

33.1. 概要

Elasticsearch、Fluentd、Kibana (EFK) スタックは、OpenShift Container Platform インストール内部で実行されるノードとアプリケーションからログを集計します。デプロイされると、Fluentd を使用して、すべてのノード、プロジェクト、Pod からのイベントログを Elasticsearch (ES) に集計します。また、Kibana Web UI が一元化されており、ユーザーと管理者は集計されたデータを使用して、多彩な視覚化機能およびダッシュボードを作成できます。

Fluentd 一括アップロードにより JSON 形式でインデックスにログが記録されてから、Elasticsearch により検索要求が該当のシャードに転送されます。

33.2. インストール

集計ロギングスタックを OpenShift Container Platform にインストールする一般的な手順は、「コンテナーログの集計」に記載されています。インストールガイドを参照する際にいくつかの重要な事項に留意してください。

ロギング Pod をクラスター上に均等に分散させるため、空のノードセレクターを使用する必要があります。

```
$ oc adm new-project logging --node-selector=""
```

これは、後で実行されるノードのラベリングと共に、ロギングプロジェクトへの Pod 配置を制御します。これでロギングプロジェクトを作成できます。

```
$ oc project logging
```

Elasticsearch (ES) は、ノード障害に対する回復性を確保するために、少なくとも 3 つのクラスターサイズでデプロイする必要があります。これは `openshift_logging_es_cluster_size` パラメーターをインベントリーホストファイルに設定することで指定されます。

パラメーターの詳細の一覧については、「Ansible 変数」を参照してください。

既存の Kibana インストールがない場合は、kibana.example.com を `openshift_logging_kibana_hostname` への値として使用します。

イメージがレジストリーからすでに取得されているかどうかや、クラスターのサイズによっては、インストールに時間がかかる場合があります。

logging namespace 内部で、`oc get all` を実行してデプロイメントを確認できます。

```
$ oc get all

NAME                        REVISION                REPLICAS
TRIGGERED BY
logging-curator             1                       1
logging-es-6cvk237t         1                       1
logging-es-e5x4t4ai         1                       1
logging-es-xmwvnorv         1                       1
logging-kibana              1                       1
NAME                        DESIRED                 CURRENT
AGE
```

```
logging-curator-1             1                         1             3d
logging-es-6cvk237t-1         1                         1             3d
logging-es-e5x4t4ai-1         1                         1             3d
logging-es-xmwvnorv-1         1                         1             3d
logging-kibana-1              1                         1             3d
NAME                          HOST/PORT                 PATH
SERVICE               TERMINATION   LABELS
logging-kibana                kibana.example.com
logging-kibana        reencrypt     component=support,logging-
infra=support,provider=openshift
logging-kibana-ops            kibana-ops.example.com
logging-kibana-ops    reencrypt     component=support,logging-
infra=support,provider=openshift
NAME                          CLUSTER-IP                EXTERNAL-IP
PORT(S)               AGE
logging-es                    172.24.155.177            <none>
9200/TCP              3d
logging-es-cluster            None                      <none>
9300/TCP              3d
logging-es-ops                172.27.197.57             <none>
9200/TCP              3d
logging-es-ops-cluster        None                      <none>
9300/TCP              3d
logging-kibana                172.27.224.55             <none>
443/TCP               3d
logging-kibana-ops            172.25.117.77             <none>
443/TCP               3d
NAME                          READY                     STATUS
RESTARTS              AGE
logging-curator-1-6s7wy       1/1                       Running       0
3d
logging-deployer-un6ut        0/1                       Completed     0
3d
logging-es-6cvk237t-1-cnpw3   1/1                       Running       0
3d
logging-es-e5x4t4ai-1-v933h   1/1                       Running       0
3d
logging-es-xmwvnorv-1-adr5x   1/1                       Running       0
3d
logging-fluentd-156xn         1/1                       Running       0
3d
logging-fluentd-40biz         1/1                       Running       0
3d
logging-fluentd-8k847         1/1                       Running       0
3d
```

最終的には以下のようなセットアップになります。

```
$ oc get pods -o wide

NAME                          READY     STATUS      RESTARTS   AGE
NODE
logging-curator-1-6s7wy       1/1       Running     0          3d
ip-172-31-24-239.us-west-2.compute.internal
logging-deployer-un6ut        0/1       Completed   0          3d
ip-172-31-6-152.us-west-2.compute.internal
```

```
logging-es-6cvk237t-1-cnpw3   1/1       Running     0          3d
ip-172-31-24-238.us-west-2.compute.internal
logging-es-e5x4t4ai-1-v933h   1/1       Running     0          3d
ip-172-31-24-235.us-west-2.compute.internal
logging-es-xmwvnorv-1-adr5x   1/1       Running     0          3d
ip-172-31-24-233.us-west-2.compute.internal
logging-fluentd-156xn         1/1       Running     0          3d
ip-172-31-24-241.us-west-2.compute.internal
logging-fluentd-40biz         1/1       Running     0          3d
ip-172-31-24-236.us-west-2.compute.internal
logging-fluentd-8k847         1/1       Running     0          3d
ip-172-31-24-237.us-west-2.compute.internal
logging-fluentd-9a3qx         1/1       Running     0          3d
ip-172-31-24-231.us-west-2.compute.internal
logging-fluentd-abvgj         1/1       Running     0          3d
ip-172-31-24-228.us-west-2.compute.internal
logging-fluentd-bh74n         1/1       Running     0          3d
ip-172-31-24-238.us-west-2.compute.internal
...
...
```

各 ES インスタンスに割り当てられる RAM の容量はデフォルトで 8 GB です。`openshift_logging_es_memory_limit` は openshift-ansible ホストインベントリーファイルで使用されるパラメーターです。この値の半分が個々の elasticsearch Pod java プロセスのヒープサイズに渡されることに注意してください。

EFK のインストールの詳細はこちらを参照してください。

33.2.1. 大規模クラスター

ノードが 100 以上ある場合、最初に `docker pull registry.access.redhat.com/openshift3/logging-fluentd:v3.10` からロギングイメージを事前にプルすることを推奨します。ロギングインフラストラクチャー Pod (Elasticsearch、Kibana、Curator) をデプロイしたら、以下の例のようにノードのラベリングを一度に 20 ノード単位で実行します。

単純なループを使用します。

```
$ while read node; do oc label nodes $node logging-infra-fluentd=true;
done < 20_fluentd.lst
```

以下の方法も有効です。

```
$ oc label nodes 10.10.0.{100..119} logging-infra-fluentd=true
```

ノードをグループ化してラベリングすると、OpenShift ロギングで DaemonSet が一定のペースで使用されるので、イメージレジストリーなどの共有リソース上の競合を回避できます。

注記

「CrashLoopBackOff | ImagePullFailed | Error」の問題が発生していないかを確認します。診断用のコマンドとしては、`oc logs <pod>`、`oc describe pod <pod>`、`oc get event` を利用できます。

33.3. SYSTEMD-JOURNALD と RSYSLOG

レート制限

Red Hat Enterprise Linux (RHEL) 7 では、systemd-journald.socket ユニットはブートプロセスで/dev/log を作成し、入力を systemd-journald.service に渡します。すべての syslog() 呼び出しはジャーナルに移ります。

Rsyslog は imjournal モジュールをジャーナルファイルのデフォルトの入力モードとして使用します。このトピックの詳細については、「Interaction of rsyslog and journal」を参照してください。

簡易テストハーネスを開発して、logger をクラスターノード上で使用し、システムログ内にさまざまなレートでさまざまなサイズのエントリーを記録しました。`systemd-219-19.el7.x86_64` を搭載したデフォルトの Red Hat Enterprise Linux (RHEL) 7 インストールにおける所定のロギングレート (毎秒約 40 行) によるテストシミュレーションで、デフォルトのレート制限 `rsyslogd` に達しました。これらの制限を調整した後、ローカルジャーナルファイルが破損したため、エントリーの journald への書き込みが停止しました。この問題は以降のバージョンの systemd で解決されています。

スケールアップ

プロジェクトをスケールアップすると、デフォルトのロギング環境で調整が必要になる場合があります。systemd-219-22.el7.x86_64 に更新した後、以下を実行します。

```
$IMUXSockRateLimitInterval 0
$IMJournalRatelimitInterval 0
```

上記の行を /etc/rsyslog.conf に追加します。

```
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=30s
```

また、上記を /etc/systemd/journald.conf に追加します。

ここで、サービスを再起動します。

```
$ systemctl restart systemd-journald.service
$ systemctl restart rsyslog.service
```

これらの設定は、一括アップロードのバースト性を担う設定です。

レート制限を削除した後、システムロギングデーモンの CPU 使用率が高くなることがあります。以前はスロットルされていた可能性のあるメッセージが処理されるためです。

Rsyslog (ratelimit.interval、ratelimit.burst を参照してください) はジャーナルからのエントリー読み出しを 300 秒に 10,000 メッセージに制限するよう設定されます。経験則として、rsyslog レート制限を systemd-journald レート制限に調節しておくことを推奨します。

33.4. EFK ロギングのスケールアップ

必要なスケールを最初のデプロイメントで指定しなかった場合、影響を最小限に抑えてクラスターを調整するには、更新したパラメーター値 `openshift_logging_es_cluster_size` でインベントリー

ファイルを更新してから、Ansible ロギング Playbook を再度実行します。詳細については、「Elasticsearch 管理操作の実行」のセクションを参照してください。

33.5. ストレージに関する考慮事項

Elasticsearch インデックスはシャードとそれに対応するレプリカシャードのコレクションです。これにより ES は高可用性を内部に実装しているので、ハードウェアベースのミラーリング RAID のバリアントを使用する必要はほとんどありません。RAID 0 を使用して全体的なディスクパフォーマンスを向上することは可能です。

それぞれの検索要求は、インデックス内のそれぞれのシャードのコピーに一致する必要があります。各 ES インスタンスには独自のストレージが必要ですが、OpenShift Container Platform デプロイメントで提供できるのはすべての Pod で共有されるボリュームだけです。したがって、Elasticsearch は単一ノードで実装しないでください。

永続ボリュームを各 Elasticsearch デプロイメント設定に追加して、レプリカシャードごとに 1 つのボリュームを割り当てます。これは、OpenShift Container Platform では多くの場合、Persistent Volume Claim (永続ボリューム要求) によって実行されます。

- シャードあたり 1 ボリューム
- レプリカシャードあたり 1 ボリューム

PVC の名前は openshift_logging_es_pvc_prefix 設定に基づいて指定する必要があります。詳細は「永続的な Elasticsearch ストレージ」を参照してください。

以下に示すのは、OpenShift Container Platform 集計ロギングの容量計画のガイドラインです (サンプルシナリオ)。

前提条件:

1. アプリケーション: Apache
2. 1 行あたりのバイト数: 256
3. アプリケーションでの 1 秒あたりのロード行数: 1
4. 生テキストデータ → JSON

ベースライン (1 分あたり 256 文字 → 15 KB/分)

ロギングインフラ Pod	ストレージスループット
3 es 1 kibana 1 curator 1 fluentd	6 Pod の合計: 90000 x 86400 = 7.7 GB/日
3 es 1 kibana 1 curator 11 fluentd	16 Pod の合計: 225000 x 86400 = 24.0 GB/日
3 es 1 kibana 1 curator 20 fluentd	25 Pod の合計: 225000 x 86400 = 32.4 GB/日

ロギング環境に必要なロギングスループットおよびディスク容量の合計を計算するには、お使いのアプリケーションに関する知識が必要です。たとえば、アプリケーションが平均して 1 秒あたり 10 行、1 行あたり 256 バイトをロギングする場合、アプリケーションごとのスループットとディスク容量は以下のように計算されます。

```
(bytes-per-line * (lines-per-second) = 2560 bytes per app per second
(2560) * (number-of-pods-per-node,100) = 256,000 bytes per second per node
256k * (number-of-nodes) = total logging throughput per cluster
```

Fluentd では systemd journal および /var/lib/docker/containers/ からのログを Elasticsearch に送信します。詳細はこちらを参照してください。

最適なパフォーマンスを得るには、ローカルの SSD ドライブの使用を推奨します。Red Hat Enterprise Linux (RHEL) 7 では、SATA ディスク以外のすべてのブロックデバイスについては deadline IO スケジューラーがデフォルトです。SATA ディスクについては、デフォルトの IO スケジューラーは cfq になります。

ES 用のストレージのサイジングは、インデックスの最適化方法よって大きく変わります。このため、必要なデータ量とアプリケーションログデータの集計方法を事前に検討しておく必要があります。一部の Elasticsearch ユーザーは、絶対的なストレージ使用率をおよそ 50% に維持し、常に 70% 未満にする必要があることを確認しています。これは、大規模なマージ操作時に Elasticsearch が応答しなくなるのを回避するのに役立ちます。

第34章 クラスターメトリクスの有効化

34.1. 概要

kubelet はメトリクスを公開しますが、これは Heapster によって収集され、バックエンドに保存されます。

OpenShift Container Platform 管理者は、すべてのコンテナーとコンポーネントから収集したクラスターのメトリクスを 1 つのユーザーインターフェースで表示できます。これらのメトリクスは、Horizontal Pod Autoscaler によるスケーリングのタイミングと方法の決定にも使用されます。

このトピックでは、Hawkular Metric をメトリクスエンジンとして使用した例について説明します。このエンジンはデータを Cassandra データベースに永続的に保存します。これが設定されると、CPU、メモリー、ネットワークベースのメトリクスを OpenShift Container Platform Web コンソールから表示できるようになり、Horizontal Pod Autoscaler で使用できるようになります。

Heapster はマスターサーバーからすべてのノードの一覧を取得して、`/stats` エンドポイントから各ノードへ個別に通信します。ここから、Heapster は CPU、メモリー、ネットワーク使用状況のメトリクスを収集して、Hawkular Metrics にエクスポートします。

kubelet で利用できるストレージボリュームメトリクスは、`/stats` エンドポイントからは利用できません。ただし、`/metrics` エンドポイントからは利用できます。詳細は、Prometheus 経由の OpenShift Container Platform メトリクスについて参照してください。

Web コンソールで個々の Pod を参照すると、メモリーと CPU に個別のスパークラインチャートが表示されます。表示される時間範囲は選択可能で、これらのチャートは 30 秒ごとに自動更新されます。Pod に複数のコンテナーがある場合は、メトリクスを表示する特定のコンテナーを選択します。

リソース制限がプロジェクトに定義されている場合、各 Pod のドーナツチャートも表示できます。ドーナツチャートはリソース制限に対する使用量を示します。たとえば、`145 Available of 200 MiB` は、ドーナツチャートでは `55 MiB Used` と表示されます。

34.2. 作業を開始する前に

Ansible Playbook はクラスターメトリクスのデプロイとアップグレードに使用できます。『クラスターのインストールガイド』に精通しておくようにしてください。Ansible を使用するための予備知識や、設定に関する情報が記載されています。クラスターメトリクスのさまざまな領域を設定するためのパラメーターが Ansible インベントリーファイルに追加されています。

以下に示すセクションでは、デフォルト値を変更するために Ansible インベントリーファイルに追加できる各種の領域とパラメーターを説明します。

- メトリクスプロジェクト
- メトリクスデータストレージ

34.3. メトリクスプロジェクト

自動スケールを機能させるには、クラスターメトリクスのコンポーネントを openshift-infra プロジェクトにデプロイする必要があります。特に Horizontal Pod Autoscaler はこのプロジェクトを使用して Heapster サービスを検出し、それをメトリクスの取得に使用します。メトリクスプロジェクトは、`openshift_metrics_project` をインベントリーファイルに追加することで変更できます。

34.4. メトリクスデータストレージ

メトリクスデータは、永続ストレージまたは一時的な Pod ボリュームのいずれかに保存できます。

34.4.1. 永続ストレージ

OpenShift Container Platform クラスターメトリクスを永続ストレージと共に実行すると、メトリクスは永続ボリュームに保存され、再起動または再作成される Pod を維持できます。これは、メトリクスデータをデータ損失から保護する必要がある場合に適しています。実稼働環境では、メトリクス Pod に永続ストレージを設定することを強く推奨します。

Cassandra ストレージのサイズ要件は Pod 数に依存します。セットアップで十分なサイズ要件を確保し、ディスクが一杯にならないように使用状況を監視するのは、管理者の責任です。永続ボリューム要求のサイズは `openshift_metrics_cassandra_pvc_size` ansible variable で指定され、デフォルトは 10 GB に設定されています。

動的にプロビジョニングされた永続ボリュームを使用する場合は、インベントリーファイルで `openshift_metrics_cassandra_storage_type` 変数を `dynamic` に設定します。

34.4.2. クラスターメトリクスの容量計画

`openshift_metrics` Ansible ロールを実行した後、`oc get pods` の出力は以下のようになります。

```
 # oc get pods -n openshift-infra
 NAME                                READY               STATUS
RESTARTS           AGE
 hawkular-cassandra-1-l5y4g          1/1                 Running      0
17h
 hawkular-metrics-1t9so              1/1                 Running      0
17h
 heapster-febru                      1/1                 Running      0
17h
```

OpenShift Container Platform メトリクスは Cassandra データベースを使用して保存されます。このデータベースは`openshift_metrics_cassandra_limits_memory: 2G` の設定でデプロイされます。この値は Cassandra の開始スクリプトで決定される空きメモリー容量に応じてさらに調整できます。この値はほとんどの OpenShift Container Platform メトリクスインストールに対応しますが、クラスターメトリクスをデプロイする前に、環境変数を使用して `MAX_HEAP_SIZE` と ヒープの新しい生成サイズ `HEAP_NEWSIZE` を Cassandra Dockerfile で変更できます。

デフォルトで、メトリクスデータは 7 日間保存されます。7 日が経過すると、Cassandra は最も古いメトリクスデータのパージを開始します。削除された Pod とプロジェクトのメトリクスデータは自動的にはパージされません。7 日を超えたデータのみが削除されます。

> 例34.1 10 のノードと 1000 の Pod による累積データ
>
> 10 のノードと 1000 の Pod を含むテストシナリオでは、24 時間で 2.5 GB のメトリクスデータが累積されました。このため、このシナリオでのメトリクスデータの容量計画の計算式は以下のようになります。
>
> $(((2.5 \times 10^9) \div 1000) \div 24) \div 10^6 = \sim 0.125$ MB/hour per pod.

> 例34.2 120 のノードと 10000 の Pod による累積データ

> *120 のノードと 10000 の Pod を含むテストシナリオでは、24 時間で 25 GB のメトリクスデータが累積されました。このため、このシナリオにおけるメトリクスデータの容量計画の計算式は以下のようになります。*
>
> *$(((11.410 \times 10^9) \div 1000) \div 24) \div 10^6 = 0.475$ MB/hour*

	1000 の Pod	10000 の Pod
24 時間で累積される Cassandra ストレージデータ (デフォルトのメトリクスパラメーター)	2.5 GB	11.4 GB

`openshift_metrics_duration` の 7 日間および `openshift_metrics_resolution` の 30 秒というデフォルト値を維持する場合、Cassandra Pod の週次のストレージ要件は以下のようになります。

	1000 の Pod	10000 の Pod
7 日間で累積される Cassandra ストレージデータ (デフォルトのメトリクスパラメーター)	20 GB	90 GB

先の表では、予期せずモニタリングされた Pod 使用量のバッファーとして、予期されたストレージ容量に対して予備の 10% が追加されています。

警告

Cassandra の永続化ボリュームに十分な空き容量がなくなると、データ損失が発生します。

クラスターメトリクスを永続ストレージと共に使用するには、永続ボリュームに ReadWriteOnce アクセスモードが設定されていることを確認します。このモードがアクティブではない場合、Persistent Volume Claim (永続ボリューム要求) は永続ボリュームを特定できず、Cassandra の開始に失敗します。

永続ストレージをメトリクスコンポーネントと併用するには、十分なサイズの永続ボリュームがあることを確認します。Persistent Volume Claim (永続ボリューム要求) の作成は OpenShift Ansible `openshift_metrics` ロールによって処理されます。

OpenShift Container Platform メトリクスは、動的にプロビジョニングされた永続ボリュームもサポートします。この機能を OpenShift Container Platform メトリクスで使用するには、`openshift_metrics_cassandra_storage_type` の値を `dynamic` に設定する必要があります。EBS、GCE、Cinder ストレージバックエンドを使用すると永続ボリュームを動的にプロビジョニングできます。

パフォーマンスの設定とクラスターメトリクス Pod のスケーリングについては、「Scaling Cluster Metrics」のトピックを参照してください。

表34.1 クラスター内のノード/Pod の数に基づく Cassandra データベースのストレージ要件

ノード数	Pod 数	Cassandra ストレージの増加率	1 日あたりの Cassandra ストレージの増加量	1 週間あたりの Cassandra ストレージの増加量
210	10500	1 時間あたり 500 MB	15 GB	75 GB
990	11000	1 時間あたり 1 GB	30 GB	210 GB

上記の計算では、ストレージ要件が計算値を超過しないようにするためのオーバーヘッドとして、予期されたサイズのおよそ 20% が追加されています。

`METRICS_DURATION` と `METRICS_RESOLUTION` の値がデフォルト (それぞれ 7 日間と 15 秒間) に維持されている場合、1 週間の Cassandra ストレージサイズ要件を上記の値に設定することを問題なく計画できるでしょう。

警告

OpenShift Container Platform メトリクスは、メトリクスデータのデータストアとして Cassandra データベースを使用するので、メトリクス設定のプロセスで `USE_PERSISTANT_STORAGE=true` に設定した場合には、NFS でネットワークストレージの上層に PV がデフォルトで配置されます。ただし、Cassandra ドキュメントにあるように、ネットワークストレージと、Cassandra を組み合わせて使用することは推奨していません。

既知の問題と制限
テストの結果として、`heapster` メトリクスコンポーネントは最大 25,000 の Pod を処理できることが確認されています。Pod 数がこの数を超過すると、Heapster のメトリクス処理が遅くなり始め、メトリクスグラフが表示されなくなる可能性があります。Heapster がメトリクスを収集できる Pod 数を増加する作業と、メトリクスを収集する代替ソリューションのアップストリーム開発が進められています。

34.4.3. 非永続ストレージ

OpenShift Container Platform クラスターメトリクスを非永続ストレージと共に実行すると、Pod の削除時に、保存されていたすべてのメトリクスが削除されます。クラスターメトリクスは非永続データで実行する方がはるかに容易ですが、非永続データで実行すると永続的なデータが損失するリスクが伴います。ただし、メトリクスはコンテナーが再起動されても存続します。

非永続ストレージを使用するためには、インベントリーファイルで `openshift_metrics_cassandra_storage_type` 変数を `emptyDir` に設定する必要があります。

注記

非永続ストレージを使用している場合、メトリクスデータは、Cassandra Pod が実行されるノード上の /var/lib/origin/openshift.local.volumes/pods に書き込まれます。/var にメトリクスを保存するために十分な空き容量があることを確認してください。

34.5. メトリクス ANSIBLE ロール

OpenShift Container Platform の Ansible `openshift_metrics` ロールは、Configuring Ansible インベントリーファイルにある変数を使用して、すべてのメトリクスコンポーネントを設定し、デプロイします。

34.5.1. メトリクス Ansible 変数の指定

OpenShift Ansible に含まれる `openshift_metrics` ロールはクラスターメトリクスをデプロイするタスクを定義します。以下は、上書きが必要な場合にインベントリーファイルに追加できるロール変数の一覧です。

表34.2 Ansible 変数

変数	説明
`openshift_metrics_install_metrics`	`true` の場合にメトリクスをデプロイします。そうでない場合はアンデプロイします。
`openshift_metrics_start_cluster`	コンポーネントをデプロイした後にメトリクスクラスターを起動します。
`openshift_metrics_startup_timeout`	再起動を試行する前に Hawkular Metrics および Heapster が起動するまでの時間 (秒単位)。
`openshift_metrics_duration`	メトリクスがパージされるまで保管される日数。
`openshift_metrics_resolution`	メトリクスが収集される頻度。数値と時間を示す識別子である秒 (s)、分 (m)、時間 (h) で定義されます。
`openshift_metrics_cassandra_pvc_prefix`	Cassandra に作成される Persistent Volume Claim (永続ボリューム要求) のプレフィックス。プレフィックスには 1 から始まる連番が付加されます。
`openshift_metrics_cassandra_pvc_size`	各 Cassandra ノードの Persistent Volume Claim (永続ボリューム要求) のサイズ。
`openshift_metrics_cassandra_pvc_storage_class_name`	明示的にストレージクラスを設定する場合には、`openshift_metrics_cassandra_storage_type` を `emptydir` または `dynamic` に設定しないでください。

変数	説明
openshift_metrics_cassandra_storage_type	**emptyDir** を一時ストレージとして使用します (テスト用)。**pv** を永続ボリュームとして使用します。これはインストール前に作成する必要があります。または **dynamic** を動的永続ボリュームとして使用します。
openshift_metrics_cassandra_replicas	メトリクススタックの Cassandra ノードの数。この値は Cassandra レプリケーションコントローラーの数を指定します。
openshift_metrics_cassandra_limits_memory	Cassandra Pod のメモリー制限。たとえば、値 **2Gi** にすると Cassandra のメモリーは 2 GB に制限されます。この値は開始スクリプトによって、スケジュールされているノードの空きメモリー容量に基づいてさらに調整できます。
openshift_metrics_cassandra_limits_cpu	Cassandra Pod の CPU 制限。値 **4000m** (4000 ミリコア) の場合、Cassandra は 4 基の CPU に制限されます。
openshift_metrics_cassandra_requests_memory	Cassandra Pod について要求するメモリー量。値 **2Gi** の場合、2 GB のメモリーが要求されます。
openshift_metrics_cassandra_requests_cpu	Cassandra Pod の CPU 要求。値 **4000m** (4000 ミリコア) の場合、4 基の CPU が要求されます。
openshift_metrics_cassandra_storage_group	Cassandra に使用される補助ストレージグループ。
openshift_metrics_cassandra_nodeselector	必要な既存のノードセレクターを設定して、Pod が特定のラベルを持つノードに配置されるようにします。たとえば、**{"node-role.kubernetes.io/infra":"true"}** とします。
openshift_metrics_hawkular_ca	Hawkular 証明書に署名するための認証局 (CA) ファイル (オプション)。
openshift_metrics_hawkular_cert	Hawkular メトリクスへのルートを再暗号化するために使用される証明書ファイル。証明書にはルートに使用されるホスト名を含める必要があります。これが指定されない場合、デフォルトのルーター証明書が使用されます。
openshift_metrics_hawkular_key	Hawkular 証明書で使用されるキーファイル。

変数	説明
openshift_metrics_hawkular_limits_memory	Hawkular Pod を制限するためのメモリー量。値 **2Gi** の場合、Hawkular Pod は 2 GB のメモリーに制限されます。この値は開始スクリプトで、スケジュールされているノードの空きメモリー容量に基づいてさらに調整できます。
openshift_metrics_hawkular_limits_cpu	Hawkular Pod の CPU 制限。値 **4000m** (4000 ミリコア) の場合、Hawkular Pod は 4 基の CPU に制限されます。
openshift_metrics_hawkular_replicas	Hawkular メトリクスのレプリカの数。
openshift_metrics_hawkular_requests_memory	Hawkular Pod について要求するメモリー量。値 **2Gi** の場合、2 GB のメモリーが要求されます。
openshift_metrics_hawkular_requests_cpu	Hawkular Pod の CPU 要求。値 **4000m** (4000 ミリコア) の場合、4 基の CPU が要求されます。
openshift_metrics_hawkular_nodeselector	必要な既存のノードセレクターを設定して、Pod が特定のラベルを持つノードに配置されるようにします。たとえば、**{"node-role.kubernetes.io/infra":"true"}** とします。
openshift_metrics_heapster_allowed_users	許可する CN のコンマ区切りの一覧。デフォルトで、OpenShift サービスプロキシーが接続できるように設定されます。Horizontal Pod Autoscaling を適切に機能させるには、上書きする際に **system:master-proxy** を一覧に追加します。
openshift_metrics_heapster_limits_memory	Heapster Pod を制限するメモリー量。値 **2Gi** の場合、Heapster Pod は 2 GB のメモリーに制限されます。
openshift_metrics_heapster_limits_cpu	Heapster Pod の CPU 制限。値 **4000m** (4000 ミリコア) の場合、Heapster Pod は 4 基の CPU に制限されます。
openshift_metrics_heapster_requests_memory	Heapster Pod について要求するメモリー量。値 **2Gi** の場合、2 GB のメモリーが要求されます。
openshift_metrics_heapster_requests_cpu	Heapster Pod の CPU 要求。値 **4000m** (4000 ミリコア) の場合、4 基の CPU が要求されます。
openshift_metrics_heapster_standalone	Heapster のみをデプロイします。Hawkular Metrics や Cassandra コンポーネントはデプロイされません。

変数	説明
`openshift_metrics_heapster_nodeselector`	必要な既存のノードセレクターを設定して、Pod が特定のラベルを持つノードに配置されるようにします。たとえば、`{"node-role.kubernetes.io/infra":"true"}` とします。
`openshift_metrics_hawkular_hostname`	Hawkular Metrics ルートのホスト名を使用するので、`openshift_metrics` Ansible ロールを実行する場合に設定します。この値は、完全修飾ドメイン名と対応している必要があります。

要求と制限を指定する方法の詳細については、「コンピュートリソース」を参照してください。

Cassandra で永続ストレージを使用している場合、`openshift_metrics_cassandra_pvc_size` 変数を使用してクラスターに十分なディスクサイズを設定することは管理者の責任です。また、管理者はディスクが一杯にならないようにディスク使用量を監視する必要もあります。

警告

Cassandra の永続化ボリュームの領域が不足するとデータが損失します。

その他のすべての変数はオプションで、詳細なカスタマイズが可能です。たとえば、Kubernetes マスターを `https://kubernetes.default.svc:443` で使用できないカスタムインストールでは、代わりに `openshift_metrics_master_url` パラメーターを使用して値を指定できます。

34.5.2. シークレットの使用

OpenShift Container Platform Ansible `openshift_metrics` ロールは、コンポーネント間で使用する自己署名証明書を自動生成し、re-encrypt ルートを生成して Hawkular Metrics サービスを公開します。このルートによって Web コンソールで Hawkular Metrics サービスにアクセスできます。

Web コンソールを実行するブラウザーがこのルート経由の接続を信頼するには、ルートの証明書を信頼する必要があります。これは、信頼された認証局によって署名されたユーザー固有の証明書を提供することで実行できます。`openshift_metrics` ロールによって、ユーザー固有の証明書を指定でき、この証明書がルートの作成時に使用されます。

ユーザー固有の証明書を提供しない場合、ルーターのデフォルト証明書が使用されます。

34.5.2.1. ユーザー固有の証明書の提供

ユーザー固有の証明書を提供し、re-encrypt ルートで使用するには、`openshift_metrics_hawkular_cert`、`openshift_metrics_hawkular_key`、`openshift_metrics_hawkular_ca` 変数をインベントリーファイルで設定します。

`hawkular-metrics.pem` 値には .pem 形式の証明書を含める必要があります。この pem ファイルに `hawkular-metrics-ca.cert` シークレット経由で署名した認証局の証明書を提供することも必要で

す。

詳細については、re-encrypt ルートに関するマニュアルを参照してください。

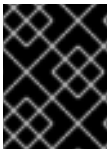
34.6. メトリクスコンポーネントのデプロイ

すべてのメトリクスコンポーネントのデプロイと設定は OpenShift Container Platform Ansible で処理されるので、すべてを 1 つのステップでデプロイできます。

以下の例では、デフォルトパラメーターを使用して、メトリクスのデプロイ時に永続ストレージを使用する場合の方法と使用しない場合の方法を示しています。

重要

Ansible Playbook を実行するホストには、ホストあたり 75MiB 以上の空きメモリーがインベントリーで必要になります。



重要

アップストリームの Kubernetes ルールに応じて、`eth0` のデフォルトインターフェースでのみメトリクスを収集できます。

例34.3 永続ストレージを使用するデプロイ

以下のコマンドでは、Hawkular Metrics ルートを hawkular-metrics.example.com を使用し、永続ストレージを使用してデプロイされるように設定します。

十分な容量を備えた永続ボリュームを用意してください。

```
$ ansible-playbook [-i </path/to/inventory>]
<OPENSHIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
    -e openshift_metrics_install_metrics=True \
    -e openshift_metrics_hawkular_hostname=hawkular-metrics.example.com \
    -e openshift_metrics_cassandra_storage_type=pv
```

例34.4 永続ストレージを使用しないデプロイ

以下のコマンドでは、Hawkular Metrics ルートを hawkular-metrics.example.com を使用し、永続ストレージを使用しないでデプロイするように設定します。

```
$ ansible-playbook [-i </path/to/inventory>]
<OPENSHIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
    -e openshift_metrics_install_metrics=True \
    -e openshift_metrics_hawkular_hostname=hawkular-metrics.example.com
```

警告

これは永続ストレージを使用しないでデプロイされるため、メトリクスデータが損失する可能性があります。

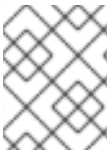
34.6.1. メトリクスの診断

メトリクススタックの状態の評価に使用できるメトリクス用の診断があります。メトリクスの診断を実行するには、以下のコマンドを実行します。

```
$ oc adm diagnostics MetricsApiProxy
```

34.7. メトリクスのパブリック URL の設定

OpenShift Container Platform Web コンソールは、Hawkular Metrics サービスから受信したデータを使用してグラフを表示します。Hawkular Metrics サービスにアクセスする URL は、マスター webconsole-config configmap ファイルで`metricsPublicURL` オプションを使用して設定する必要があります。この URL は メトリクスコンポーネントのデプロイメント時に使用される `openshift_metrics_hawkular_hostname` インベントリー変数で作成されるルートに対応します。



注記

`openshift_metrics_hawkular_hostname` はコンソールにアクセスするブラウザーで解決できる必要があります。

たとえば、`openshift_metrics_hawkular_hostname` が `hawkular-metrics.example.com` に対応する場合、webconsole-config configmap ファイルに以下の変更を行う必要があります。

```
clusterInfo:
  ...
  metricsPublicURL: "https://hawkular-
metrics.example.com/hawkular/metrics"
```

webconsole-config configmap ファイルを更新し、保存した後に、OpenShift Container Platform インスタンスを再起動する必要があります。

OpenShift Container Platform サーバーが再び稼働すると、メトリクスが Pod 概要のページに表示されます。

注意

自己署名証明書を使用している場合、Hawkular Metrics サービスはコンソールとは別のホスト名でホストされ、別の証明書を使用することに注意してください。場合によっては、ブラウザータブを明示的に開いて `metricsPublicURL` に指定される値を直接参照して、この証明書を受け入れる必要があります。

これを回避するには、お使いのブラウザーで許可されるように設定された証明書を使用します。

34.8. HAWKULAR METRICS への直接アクセス

メトリクスに直接アクセスし、これを管理するには、Hawkular Metrics API を使用します。

注記

API から Hawkular Metrics にアクセスした場合は、読み取り操作しか実行できません。メトリクスの書き込みはデフォルトで無効にされています。個々のユーザーがメトリクスの書き込みも実行できるようにするには、`openshift_metrics_hawkular_user_write_access` 変数を true に設定する必要があります。

ただし、デフォルトの設定を使用して Heapster からのみメトリクスを入力することを推奨します。書き込みアクセスが有効になると、どのユーザーもメトリクスをシステムに書き込めるようになり、これがパフォーマンスに影響を及ぼし、Cassandra のディスク使用量が予期せずに増加する可能性があります。

Hawkular Metrics のマニュアルでは、API の使用方法を説明していますが、OpenShift Container Platform で使用するように設定されたバージョンの Hawkular Metrics とは処理方法に多少の違いがあります。

34.8.1. OpenShift Container Platform プロジェクトと Hawkular テナント

Hawkular Metrics はマルチテナントアプリケーションで、OpenShift Container Platform 内のプロジェクトが Hawkular Metrics 内のテナントに対応するように設定されます。

このため、MyProject という名前のプロジェクトのメトリクスにアクセスする場合は、Hawkular-Tenant ヘッダーを MyProject に設定する必要があります。

また、_system という名前の特殊なテナントもあり、これにはシステムレベルのメトリクスが含まれています。これにアクセスするには、cluster-reader または cluster-admin レベルの権限が必要です。

34.8.2. 承認

Hawkular Metrics サービスは、ユーザーを OpenShift Container Platform に対して認証し、ユーザーがアクセスしようとしているプロジェクトに対するアクセスを持つかどうかを判別します。

Hawkular Metrics はクライアントからベアラートークンを受け取り、SubjectAccessReview を使用して、OpenShift Container Platform サーバーでトークンを検証します。ユーザーがプロジェクトに対する適切な読み取り権限を持つ場合、ユーザーはこのプロジェクトのメトリクスを読み取ることができます。_system テナントについては、このテナントからの読み取りを要求するユーザーには、cluster-reader パーミッションがなければなりません。

Hawkular Metrics API にアクセスする場合、ベアラートークンは Authorization ヘッダーに渡す必要があります。

34.9. OPENSHIFT CONTAINER PLATFORM クラスターメトリクス POD のスケーリング

クラスターメトリクス機能のスケーリングに関する情報は、『Scaling and Performance Guide』に記載されています。

34.10. クリーンアップ

OpenShift Container Platform Ansible `openshift_metrics` ロールによってデプロイされたものはすべて、以下の手順を実行して削除できます。

```
$ ansible-playbook [-i </path/to/inventory>]
<OPENSHIFT_ANSIBLE_DIR>/playbooks/openshift-metrics/config.yml \
    -e openshift_metrics_install_metrics=False
```

34.11. OPENSHIFT CONTAINER PLATFORM 上の PROMETHEUS

Prometheus はスタンドアロンでオープンソースシステムの、モニタリングおよびアラートツールキットです。Prometheus を使用すると、OpenShift Container Platform システムリソースのメトリクスとアラートを視覚化できます。

重要

OpenShift Container Platform 上での Prometheus はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

34.11.1. Prometheus ロール変数の設定

Prometheus ロールは以下を作成します。

- *`openshift-metrics` namespace*
- *Prometheus `clusterrolebinding` およびサービスアカウント*
- *OAuth プロキシーの背後の Prometheus、Alertmanager および Alert Buffer をステートフルセットとして備えた Prometheus Pod*
- *Prometheus と `prometheus-alerts` ConfigMaps*
- *Prometheus と Prometheus Alerts サービスおよび直接ルート*

Prometheus デプロイメントはデフォルトで無効です。`openshift_hosted_prometheus_deploy` を `true` に設定してこれを有効にします。以下は例になります。

```
# openshift_hosted_prometheus_deploy=true
```

以下のロール変数を設定して、Prometheus をインストールし、設定します。

表34.3 Prometheus 変数

変数	説明

変数	説明
openshift_prometheus_namespace	コンポーネントがデプロイされるプロジェクト namespace。デフォルトは **openshift-metrics** に設定されます。例: **openshift_prometheus_namespace=${USER_PROJECT}**
openshift_prometheus_node_selector	Prometheus がデプロイされるノードのセレクター。
openshift_prometheus_storage_kind	Prometheus の PV を作成するために設定します。例: **openshift_prometheus_storage_kind=nfs**
openshift_prometheus_alertmanager_storage_kind	Alertmanager の PV を作成するために設定します。例: **openshift_prometheus_alertmanager_storage_kind=nfs**
openshift_prometheus_alertbuffer_storage_kind	Alert Buffer の PV を作成するために設定します。例: **openshift_prometheus_alertbuffer_storage_kind=nfs**
openshift_prometheus_storage_type	Prometheus の PVC を作成するために設定します。例: **openshift_prometheus_storage_type=pvc**
openshift_prometheus_alertmanager_storage_type	Alertmanager の PVC を作成するために設定します。例: **openshift_prometheus_alertmanager_storage_type=pvc**
openshift_prometheus_alertbuffer_storage_type	Alert Buffer の PVC を作成するために設定します。例: **openshift_prometheus_alertbuffer_storage_type=pvc**
openshift_prometheus_additional_rules_file	追加の Prometheus ルールファイル。デフォルトで**null** に設定されます。

34.11.2. Ansible インストーラーを使用した Prometheus のデプロイ

重要

Ansible Playbook を実行するホストには、ホストあたり 75MiB 以上の空きメモリーがインベントリーで必要になります。

Ansible インストーラーは、Prometheus をデプロイするデフォルトの方法です。

ラベルをノードに追加します。

```
# Inventory file
openshift_prometheus_namespace=openshift-metrics

openshift_prometheus_node_selector={"node-
role.kubernetes.io/infra":"true"}
```

playbook を実行します。

```
$ ansible-playbook -vvv -i ${INVENTORY_FILE} playbooks/openshift-
prometheus/config.yml
```

34.11.2.1. Prometheus をデプロイする他の方法

ノードセレクターを使用したデプロイ

Prometheus をデプロイするノードにラベルを付けます。

```
# oc adm label node/$NODE ${KEY}=${VALUE}
```

Ansible とコンテナーリソースを使って Prometheus をデプロイします。

```
# Inventory file
openshift_prometheus_namespace=openshift-metrics

# Set node selector for prometheus
openshift_prometheus_node_selector={"${KEY}":"${VALUE}"}
```

playbook を実行します。

```
$ ansible-playbook -vvv -i ${INVENTORY_FILE} playbooks/openshift-
prometheus/config.yml
```

デフォルト以外の namespace を使用したデプロイ

namespace を特定します。

```
# Inventory file
openshift_prometheus_node_selector={"node-
role.kubernetes.io/infra":"true"}

# Set non-default openshift_prometheus_namespace
openshift_prometheus_namespace=${USER_PROJECT}
```

playbook を実行します。

```
$ ansible-playbook -vvv -i ${INVENTORY_FILE} playbooks/openshift-
prometheus/config.yml
```

34.11.2.2. Prometheus Web UI へのアクセス

Prometheus サーバーは `localhost:9090` で Web UI を自動的に公開します。Prometheus Web UI には`view` ロールでアクセスできます。

34.11.2.3. Prometheus での OpenShift Container Platform の設定

Prometheus のストレージ関連の変数

Prometheus の各コンポーネント (Prometheus、Alertmanager、Alert Buffer、OAuth プロキシーを含む) を使用して、対応するロール変数を設定して PVC を設定できます。以下は例になります。

```
openshift_prometheus_storage_type: pvc
openshift_prometheus_alertmanager_pvc_name: alertmanager
openshift_prometheus_alertbuffer_pvc_size: 10G
openshift_prometheus_pvc_access_modes: [ReadWriteOnce]
```

Prometheus アラートルールファイル変数

追加のルール変数へのパスを設定すると、アラートルールを含む外部ファイルを追加できます。

```
openshift_prometheus_additional_rules_file: <PATH>
```

ファイルは、Prometheus アラートルールの形式に準拠する必要があります。以下の例では、クラスターノードの 1 つがダウンしている時にアラートを送信するルールを設定しています。

```
groups:
- name: example-rules
  interval: 30s # defaults to global interval
  rules:
  - alert: Node Down
    expr: up{job="kubernetes-nodes"} == 0
    for: 10m (1)
    annotations:
      miqTarget: "ContainerNode"
      severity: "HIGH"
      message: "{{ '{{' }}{{ '$labels.instance' }}{{ '}}' }} is down"
```

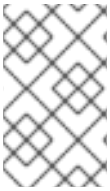
(1) オプションの `for` の値は、この要素に関するアラートを送信する前に Prometheus が待機する時間を指定します。たとえば、`10m` に設定すると、Prometheus は、この問題が発生してからアラートを送信するまで 10 分待ちます。

リソース制限を制御する Prometheus 変数

Prometheus の各コンポーネント (Prometheus、Alertmanager、Alert Buffer、OAuth プロキシーを含む) を使用して、対応するロール変数を設定することで、CPU、メモリー制限および要求を指定できます。以下は例になります。

```
openshift_prometheus_alertmanager_limits_memory: 1Gi
openshift_prometheus_oauth_proxy_cpu_requests: 100m
```

詳細については、「OpenShift Prometheus」を参照してください。



注記

`openshift_metrics_project: openshift-infra` がインストールされると、メトリクスは `http://${POD_IP}:7575/metrics` エンドポイントから収集できるようになります。

34.11.3. Prometheus 経由の OpenShift Container Platform メトリクス

システムの状態は、システムが出力するメトリクスによって測定できます。このセクションでは、ストレージサブシステムおよびクラスターの健全性を識別する現行のメトリクスと提案されているメトリクスについて説明します。

34.11.3.1. 現行のメトリクス

このセクションでは、現時点の Kubernetes のストレージサブシステムから出力されるメトリクスについて説明します。

クラウドプロバイダー API 呼び出しメトリクス

このメトリクスは、すべての cloudprovider API 呼び出しの成功と失敗の時刻と回数を報告します。これらのメトリクスには `aws_attach_time` と `aws_detach_time` が含まれています。出力されるメトリクスのタイプはヒストグラムであるため、Prometheus はこれらのメトリクスについての合計、カウント、バケットメトリクスも生成します。

GCE からの cloudprovider メトリクスの概要のサンプル:

```
cloudprovider_gce_api_request_duration_seconds { request =
"instance_list"}
cloudprovider_gce_api_request_duration_seconds { request = "disk_insert"}
cloudprovider_gce_api_request_duration_seconds { request = "disk_delete"}
cloudprovider_gce_api_request_duration_seconds { request = "attach_disk"}
cloudprovider_gce_api_request_duration_seconds { request = "detach_disk"}
cloudprovider_gce_api_request_duration_seconds { request = "list_disk"}
```

AWS からの cloudprovider メトリクスの概要のサンプル:

```
cloudprovider_aws_api_request_duration_seconds { request =
"attach_volume"}
cloudprovider_aws_api_request_duration_seconds { request =
"detach_volume"}
cloudprovider_aws_api_request_duration_seconds { request = "create_tags"}
cloudprovider_aws_api_request_duration_seconds { request =
"create_volume"}
cloudprovider_aws_api_request_duration_seconds { request =
"delete_volume"}
cloudprovider_aws_api_request_duration_seconds { request =
"describe_instance"}
cloudprovider_aws_api_request_duration_seconds { request =
"describe_volume"}
```

詳細は、「Cloud Provider (specifically GCE and AWS) metrics for Storage API calls」を参照してください。

ボリューム操作のメトリクス

これらのメトリクスは、ストレージ操作が開始されてからの所要時間を報告します。これらのメトリクスはプラグインレベルで操作時間を追跡しますが、`goroutine` の実行時間や内部キューからピックアップされる操作にかかった時間は除外されます。これらのメトリクスのタイプはヒストグラムです。

ボリューム操作メトリクスの概要のサンプル

```
storage_operation_duration_seconds { volume_plugin = "aws-ebs",
operation_name = "volume_attach" }
storage_operation_duration_seconds { volume_plugin = "aws-ebs",
operation_name = "volume_detach" }
storage_operation_duration_seconds { volume_plugin = "glusterfs",
operation_name = "volume_provision" }
storage_operation_duration_seconds { volume_plugin = "gce-pd",
operation_name = "volume_delete" }
storage_operation_duration_seconds { volume_plugin = "vsphere",
operation_name = "volume_mount" }
storage_operation_duration_seconds { volume_plugin = "iscsi" ,
operation_name = "volume_unmount" }
storage_operation_duration_seconds { volume_plugin = "aws-ebs",
operation_name = "unmount_device" }
storage_operation_duration_seconds { volume_plugin = "cinder" ,
operation_name = "verify_volumes_are_attached" }
storage_operation_duration_seconds { volume_plugin = "<n/a>" ,
operation_name = "verify_volumes_are_attached_per_node" }
```

詳細は、「Volume operation metrics」を参照してください。

ボリューム統計メトリクス

これらのメトリクスは通常、PVC の使用状況の統計を報告します (空き領域に対する使用中の領域など)。出力されるメトリクスのタイプはゲージです。

表34.4 ボリューム統計メトリクス

メトリクス	種類	ラベル/タグ
volume_stats_capacityBytes	ゲージ	namespace,persistentvolumeclaim,persistentvolume=
volume_stats_usedBytes	ゲージ	namespace=<persistentvolumeclaim-namespace> persistentvolumeclaim=<persistentvolumeclaim-name> persistentvolume=<persistentvolume-name>
volume_stats_availableBytes	ゲージ	namespace=<persistentvolumeclaim-namespace> persistentvolumeclaim=<persistentvolumeclaim-name> persistentvolume=

メトリクス	種類	ラベル/タグ
volume_stats_InodesFree	ゲージ	namespace=<persistentvolumeclaim-namespace> persistentvolumeclaim=<persistentvolumeclaim-name> persistentvolume=<persistentvolume-name>
volume_stats_Inodes	ゲージ	namespace=<persistentvolumeclaim-namespace> persistentvolumeclaim=<persistentvolumeclaim-name> persistentvolume=<persistentvolume-name>
volume_stats_InodesUsed	ゲージ	namespace=<persistentvolumeclaim-namespace> persistentvolumeclaim=<persistentvolumeclaim-name> persistentvolume=<persistentvolume-name>

34.11.4. Prometheus のアンデプロイ

***Prometheus** をアンデプロイするには、以下のコマンドを実行します。*

```
$ ansible-playbook -vvv -i ${INVENTORY_FILE} playbooks/openshift-
prometheus/config.yml -e openshift_prometheus_state=absent
```

第35章 WEB コンソールのカスタマイズ

35.1. 概要

管理者は拡張機能を使用して Web コンソールをカスタマイズできます。拡張機能でスクリプトを実行すると、Web コンソールを読み込む際にカスタムスタイルシートを読み込むことができます。拡張スクリプトによって、Web コンソールのデフォルト動作を上書きし、必要に応じてカスタマイズできます。

たとえば、拡張スクリプトを使用して、自社独自のブランディングを追加したり、自社固有の機能を追加したりすることができます。この使用例として一般的なのは、さまざまな環境のリブランディングやホワイトラベリングです。同じ拡張コードを使用しながら、Web コンソールを変更する設定を指定できます。

注意

Web コンソールのスタイルや動作に対して、以下に記載されていないような広範な変更を実行するには注意が必要です。スクリプトやスタイルシートを追加することはできますが、大幅にカスタマイズすると、今後のバージョンで Web コンソールのマークアップや動作が変更された場合に、アップグレード時に再作業が必要になる可能性があります。

35.2. 拡張スクリプトとスタイルシートの読み込み

OpenShift Container Platform 3.9 では、拡張スクリプトとスタイルシートは、URL がブラウザーからアクセス可能であれば、任意の `https://` URL でホストできます。ファイルは、パブリックにアクセスできるルートを使用してプラットフォーム上の Pod からや OpenShift Container Platform 外部の別のサーバー上の Pod からホストできます。

スクリプトとスタイルシートを追加するには、`openshift-web-console` namespace で `webconsole-config`ConfigMap を編集します。Web コンソール設定は ConfigMap の `webconsole-config.yaml` キーにあります。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

スクリプトを追加するには、`extensions.scriptURLs` プロパティーを更新します。値は URL の配列です。

スタイルシートを追加するには、`extensions.stylesheetURLs` プロパティーを更新します。値は URL の配列です。

`extensions.stylesheetURLs` 設定の例

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    extensions:
      scriptURLs:
        - https://example.com/scripts/menu-customization.js
        - https://example.com/scripts/nav-customization.js
      stylesheetURLs:
```

```
        - https://example.com/styles/logo.css
        - https://example.com/styles/custom-styles.css
    [...]
```

ConfigMap を保存した後、Web コンソールコンテナーは数分以内に新規の拡張ファイルについて自動的に更新されます。

注記

スクリプトとスタイルシートは正しいコンテンツタイプで提供する必要があります。そうしないと、それらはブラウザーで実行されません。スクリプトは `Content-Type: application/javascript`、スタイルシートは `Content-Type: text/css` で提供する必要があります。

最良の実践例として、拡張スクリプトを Immediately Invoked Function Expression (IIFE) でラップすることができます。これにより、Web コンソールや他の拡張で使用される名前と競合するグローバル変数を作成することを防ぐことができます。以下は例になります。

```
(function() {
  // Put your extension code here...
}());
```

以降のセクションの例では、Web コンソールをカスタマイズする一般的な方法を示します。

注記

拡張の他の例については、GitHub の OpenShift Origin リポジトリーを参照してください。

35.2.1. 拡張プロパティーの設定

特定の拡張について環境ごとに異なるテキストを使用する場合、Web コンソール設定で環境を定義し、同じ拡張スクリプトを環境全体で使用できます。

拡張プロパティーを追加するには、`openshift-web-console` namespace で `webconsole-config` ConfigMap を 編集します。Web コンソール設定は ConfigMap の `webconsole-config.yaml` キーにあります。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

`extensions.properties` の値を更新します。これはキーと値のペアのマップです。

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    extensions:
      [...]
      properties:
        doc_url: https://docs.openshift.com
```

```
        key1: value1
        key2: value2
    [...]
```

これによって、以下のコードが実行された場合と同様に、拡張からアクセスできるグローバル変数が生成されます。

```
window.OPENSHIFT_EXTENSION_PROPERTIES = {
  doc_url: "https://docs.openshift.com",
  key1: "value1",
  key2: "value2"
}
```

35.3. 外部ロギングソリューションの拡張オプション

OpenShift Container Platform 3.6 には、OpenShift Container Platform の EFK ロギングスタックを使用しなくても外部ロギングソリューションにリンクできる拡張オプションがあります。

```
'use strict';
angular.module("mylinkextensions", ['openshiftConsole'])
       .run(function(extensionRegistry) {
          extensionRegistry.add('log-links', _.spread(function(resource,
options) {
            return {
              type: 'dom',
              node: '<span><a href="https://extension-
point.example.com">' + resource.metadata.name + '</a><span class="action-
divider">|</span></span>'
            };
          }));
       });
hawtioPluginLoader.addModule("mylinkextensions");
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.4. ガイド付きツアーのカスタマイズと無効化

ユーザーの特定ブラウザーへの初回ログイン時に、ガイド付きツアーが表示されます。新規ユーザーに対して auto_launch を有効にできます。

```
window.OPENSHIFT_CONSTANTS.GUIDED_TOURS.landing_page_tour.auto_launch =
true;
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.5. ドキュメントリンクのカスタマイズ

ランディングページのドキュメントリンクはカスタマイズできます。window.OPENSHIFT_CONSTANTS.CATALOG_HELP_RESOURCES はタイトルと href を含むオブジェクトの配列で、これらはリンクに変換されます。この配列を完全に上書きしたり、追加のリンクをプッシュまたはポップしたり、既存のリンクの属性を変更したりすることができます。以下は例になります。

```
window.OPENSHIFT_CONSTANTS.CATALOG_HELP_RESOURCES.push({
  title: 'Blog',
  href: 'https://blog.openshift.com'
});
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.6. ロゴのカスタマイズ

以下のスタイルでは、Web コンソールヘッダーのロゴを変更します。

```
#header-logo {
  background-image: url("https://www.example.com/images/logo.png");
  width: 190px;
  height: 20px;
}
```

example.com URL を実際のイメージへの URL に置換して、幅と高さを調整します。理想的な高さは 20px です。

「拡張スクリプトとスタイルシートの読み込み」で説明されているスタイルシートを追加します。

35.7. メンバーシップのホワイトリストのカスタマイズ

メンバーシップページのデフォルトのホワイトリストには、admin、basic-user、edit などのクラスターロールのサブセットだけでなく、プロジェクト内に定義されているカスタムロールが表示されます。

たとえば、ホワイトリストに、独自のカスタムのクラスターロールセットを追加します。

```
window.OPENSHIFT_CONSTANTS.MEMBERSHIP_WHITELIST = [
  "admin",
  "basic-user",
  "edit",
  "system:deployer",
  "system:image-builder",
  "system:image-puller",
  "system:image-pusher",
  "view",
  "custom-role-1",
  "custom-role-2"
];
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.8. ドキュメントへのリンクの変更

Web コンソールのさまざまなセクションに、外部ドキュメントへのリンクが表示されます。以下の例では、ドキュメントへの指定された 2 つのリンクの URL を変更します。

```
window.OPENSHIFT_CONSTANTS.HELP['get_started_cli']      =
"https://example.com/doc1.html";
window.OPENSHIFT_CONSTANTS.HELP['basic_cli_operations'] =
```

```
"https://example.com/doc2.html";
```

または、すべてのドキュメントリンクのベース URL を変更できます。

この例では、デフォルトのヘルプ URL `https://example.com/docs/welcome/index.html` が作成されます。

```
window.OPENSHIFT_CONSTANTS.HELP_BASE_URL = "https://example.com/docs/"; 1
```

1 パスの末尾は / にする必要があります。

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.9. CLI をダウンロードするリンクの追加または変更

Web コンソールの About ページには、コマンドラインインターフェース (CLI) ツールのダウンロードリンクがあります。これらのリンクはリンクテキストと URL の両方を提供して設定でき、それらをファイルパッケージに直接ポイントさせるか、または実際のパッケージを指す外部ページにポイントさせることを選択できます。

ダウンロードできるパッケージを直接ポイントするには、以下を実行します。ここでリンクテキストはパッケージプラットフォームになります。

```
window.OPENSHIFT_CONSTANTS.CLI = {
  "Linux (32 bits)": "https://<cdn>/openshift-client-tools-linux-
32bit.tar.gz",
  "Linux (64 bits)": "https://<cdn>/openshift-client-tools-linux-
64bit.tar.gz",
  "Windows":         "https://<cdn>/openshift-client-tools-windows.zip",
  "Mac OS X":        "https://<cdn>/openshift-client-tools-mac.zip"
};
```

または、実際のダウンロードパッケージにリンクするページをポイントするには、以下を実行します。ここでは、Latest Release をリンクテキストに指定しています。

```
window.OPENSHIFT_CONSTANTS.CLI = {
  "Latest Release": "https://<cdn>/openshift-client-tools/latest.html"
};
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.9.1. About ページのカスタマイズ

Web コンソールの About ページをカスタマイズするには、以下の手順に従います。

1. 以下のような拡張を作成します。

```
angular
  .module('aboutPageExtension', ['openshiftConsole'])
  .config(function($routeProvider) {
    $routeProvider
      .when('/about', {
        templateUrl:
```

```
'https://example.com/extensions/about/about.html',
        controller: 'AboutController'
      });
    }
  );

hawtioPluginLoader.addModule('aboutPageExtension');
```

2. カスタムテンプレートを作成します。
使用している OpenShift Container Platform リリースの about.html のバージョンから開始します。テンプレート内には 2 つの angular スコープ変数、`version.master.openshift` および `version.master.kubernetes` があります。

3. Web コンソールの適切な Cross-Origin Resource Sharing (CORS) 応答ヘッダーを持つ URL でテンプレートをホストします。

   a. Web コンソールドメインからの要求を許可するように `Access-Control-Allow-Origin` 応答を設定します。

   b. `GET` を含めるように `Access-Control-Allow-Methods` を設定します。

   c. `Content-Type` を含めるように `Access-Control-Allow-Headers` を設定します。

または、AngularJS $templateCache を使用すると、テンプレートを JavaScript に直接含めることができます。

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.10. ナビゲーションメニューの設定

35.10.1. トップナビゲーションドロップダウンメニュー

Web コンソールのトップナビゲーションバーには、ヘルプアイコンとユーザードロップダウンメニューがあります。これらには angular-extension-registry を使用してメニュー項目を追加できます。

以下の拡張ポイントを利用できます。

- `nav-help-dropdown` - ヘルプアイコンのドロップダウンメニュー、デスクトップ画面の幅で表示される
- `nav-user-dropdown` - ユーザードロップダウンメニュー、デスクトップ画面の幅で表示される
- `nav-dropdown-mobile` - トップナビゲーション項目の単一メニュー、モバイル画面の幅で表示される

以下の例では、`nav-help-dropdown` メニューを `<myExtensionModule>` の名前で拡張しています。

注記

`<myExtensionModule>` はプレースホルダー名です。各ドロップダウンメニュー拡張は、今後作成される angular モジュールと競合しないように一意にする必要があります。

```
angular
  .module('<myExtensionModule>', ['openshiftConsole'])
  .run([
    'extensionRegistry',
    function(extensionRegistry) {
      extensionRegistry
        .add('nav-help-dropdown', function() {
          return [
            {
              type: 'dom',
              node: '<li><a href="http://www.example.com/report"
target="_blank">Report a Bug</a></li>'
            }, {
              type: 'dom',
              node: '<li class="divider"></li>'  // If you want a
horizontal divider to appear in the menu
            }, {
              type: 'dom',
              node: '<li><a href="http://www.example.com/status"
target="_blank">System Status</a></li>'
            }
          ];
        });
    }
  ]);

hawtioPluginLoader.addModule('<myExtensionModule>');
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.10.2. アプリケーションランチャー

トップナビゲーションバーには、他の Web アプリケーションにリンクするオプションのアプリケーションランチャーもあります。このドロップダウンメニューはデフォルトでは空ですが、リンクを追加すると、マストヘッドのヘルプメニューの左側に表示されます。

```
// Add items to the application launcher dropdown menu.
window.OPENSHIFT_CONSTANTS.APP_LAUNCHER_NAVIGATION = [{
  title: "Dashboard",                    // The text label
  iconClass: "fa fa-dashboard",          // The icon you want to appear
  href: "http://example.com/dashboard",  // Where to go when this item is
clicked
  tooltip: 'View dashboard'              // Optional tooltip to display
on hover
}, {
  title: "Manage Account",
  iconClass: "pficon pficon-user",
  href: "http://example.com/account",
  tooltip: "Update email address or password."
}];
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.10.3. システムステータスバッジ

トップナビゲーションバーにはオプションのシステムステータスバッジも追加できます。これによって、メンテナンス時期など、システム全体のイベントをユーザーに通知できます。黄色の警告アイコンを使う既存のスタイルをバッジに使用するには、以下の例に従います。

```
'use strict';

angular
  .module('mysystemstatusbadgeextension', ['openshiftConsole'])
  .run([
    'extensionRegistry',
    function(extensionRegistry) {
      // Replace http://status.example.com/ with your domain
      var system_status_elem = $('<a href="http://status.example.com/"' +
      'target="_blank" class="nav-item-iconic system-status"><span
title="' +
      'System Status" class="fa status-icon pficon-warning-triangle-o">' +
      '</span></a>');

      // Add the extension point to the registry so the badge appears
      // To disable the badge, comment this block out
      extensionRegistry
        .add('nav-system-status', function() {
          return [{
            type: 'dom',
            node: system_status_elem
          }];
        });
    }
  ]);

hawtioPluginLoader.addModule('mysystemstatusbadgeextension');
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.10.4. プロジェクトの左ナビゲーション

プロジェクト内でナビゲートするとき、プライマリーおよびセカンダリーナビゲーションのメニューが左側に表示されます。このメニューの構造は定数として定義され、上書きや修正が可能です。

注記

プロジェクトナビゲーションに大幅なカスタマイズを実行すると、ユーザーエクスペリエンスに影響することがあるので、十分に注意して行ってください。既存のナビゲーション項目を変更した場合、今後のアップグレードでこのカスタマイズを更新することが必要になる可能性があります。

```
// Append a new primary nav item.  This is a simple direct navigation item
// with no secondary menu.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.push({
  label: "Dashboard",           // The text label
  iconClass: "fa fa-dashboard", // The icon you want to appear
  href: "/dashboard"            // Where to go when this nav item is
clicked.
                                // Relative URLs are pre-pended with the
```

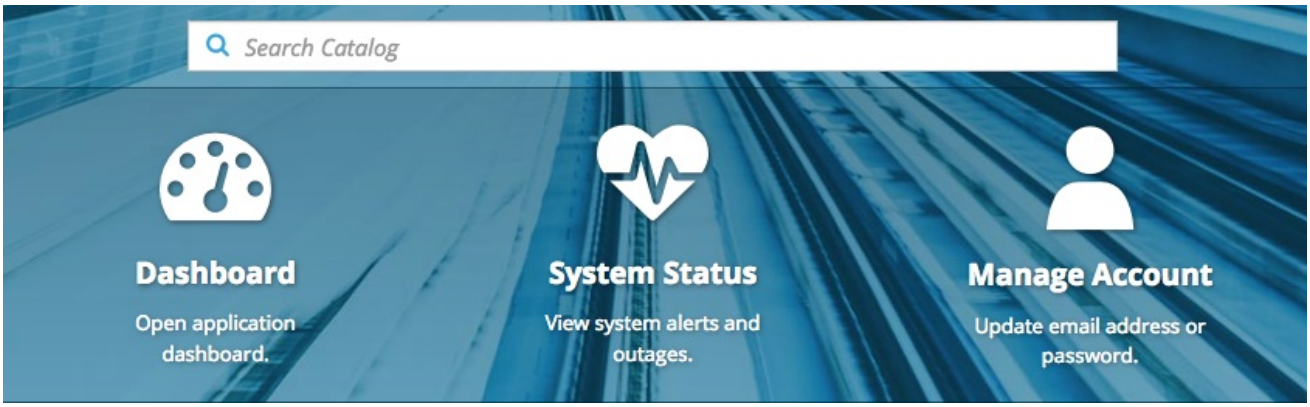
```
path
                                // '/project/<project-name>'
});

// Splice a primary nav item to a specific spot in the list.  This primary
item has
// a secondary menu.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.splice(2, 0, { // Insert at
the third spot
  label: "Git",
  iconClass: "fa fa-code",
  secondaryNavSections: [       // Instead of an href, a sub-menu can be
defined
    {
      items: [
        {
          label: "Branches",
          href: "/git/branches",
          prefixes: [
            "/git/branches/"     // Defines prefix URL patterns that will
cause
                                 // this nav item to show the active
state, so
                                 // tertiary or lower pages show the
right context
          ]
        }
      ]
    },
    {
      header: "Collaboration",   // Sections within a sub-menu can have an
optional header
      items: [
        {
          label: "Pull Requests",
          href: "/git/pull-requests",
          prefixes: [
            "/git/pull-requests/"
          ]
        }
      ]
    }
  ]
});

// Add a primary item to the top of the list.  This primary item is shown
conditionally.
window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION.unshift({
  label: "Getting Started",
  iconClass: "pficon pficon-screen",
  href: "/getting-started",
  prefixes: [                   // Primary nav items can also specify
prefixes to trigger
    "/getting-started/"         // active state
  ],
  isValid: function() {         // Primary or secondary items can define
```

```
an isValid
    return isNewUser;           // function. If present it will be called
to test whether
                                // the item should be shown, it should
return a boolean
  }
});

// Modify an existing menu item
var applicationsMenu =
_.find(window.OPENSHIFT_CONSTANTS.PROJECT_NAVIGATION, { label:
'Applications' });
applicationsMenu.secondaryNavSections.push({ // Add a new secondary nav
section to the Applications menu
  // my secondary nav section
});
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.11. 主要アプリケーションの設定

Web コンソールのランディングページカタログには、主要アプリケーションへのリンクの (オプションの) 一覧があります。これらはページ上部近くに表示され、アイコン、タイトル、簡単な説明およびリンクを指定できます。



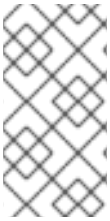

```
// Add featured applications to the top of the catalog.
window.OPENSHIFT_CONSTANTS.SAAS_OFFERINGS = [{
  title: "Dashboard",                       // The text label
  icon: "fa fa-dashboard",                  // The icon you want to
appear
  url: "http://example.com/dashboard",      // Where to go when this
item is clicked
  description: "Open application dashboard."  // Short description
}, {
  title: "System Status",
  icon: "fa fa-heartbeat",
  url: "http://example.com/status",
  description: "View system alerts and outages."
}, {
  title: "Manage Account",
  icon: "pficon pficon-user",
```

```
  url: "http://example.com/account",
  description: "Update email address or password."
}];
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.12. カタログカテゴリーの設定

カタログカテゴリーは、Web コンソールカタログのランディングページ内の項目の表示を整理します。各カテゴリーには 1 つ以上のサブカテゴリーがあります。一致するサブカテゴリータグに含まれるタグがある場合、ビルダーイメージ、テンプレート、またはサービスがサブカテゴリーにグループ化されます。また、1 つの項目を複数のサブカテゴリーに表示することができます。カテゴリーとサブカテゴリーは、1 つ以上の項目がある場合にのみ表示されます。



注記

カタログカテゴリーに大幅なカスタマイズを実行すると、ユーザーエクスペリエンスに影響することがあるので、十分に注意して行ってください。既存のカテゴリー項目を変更した場合、今後のアップグレードでこのカスタマイズを更新することが必要になる可能性があります。

```
// Find the Languages category.
var category =
_.find(window.OPENSHIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES,
                       { id: 'languages' });
// Add Go as a new subcategory under Languages.
category.subCategories.splice(2,0,{ // Insert at the third spot.
  // Required. Must be unique.
  id: "go",
  // Required.
  label: "Go",
  // Optional. If specified, defines a unique icon for this item.
  icon: "icon-go-gopher",
  // Required. Items matching any tag will appear in this subcategory.
  tags: [
    "go",
    "golang"
  ]
});

// Add a Featured category as the first category tab.
window.OPENSHIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES.unshift({
  // Required. Must be unique.
  id: "featured",
  // Required
  label: "Featured",
  subCategories: [
    {
      // Required. Must be unique.
      id: "go",
      // Required.
      label: "Go",
      // Optional. If specified, defines a unique icon for this item.
      icon: "icon-go-gopher",
      // Required. Items matching any tag will appear in this subcategory.
```

```
      tags: [
        "go",
        "golang"
      ]
    },
    {
      // Required. Must be unique.
      id: "jenkins",
      // Required.
      label: "Jenkins",
      // Optional. If specified, defines a unique icon for this item.
      icon: "icon-jenkins",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "jenkins"
      ]
    }
  ]
});
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.13. クォータ通知メッセージの設定

ユーザーがクォータに達すると、クォータ通知が通知ドロワーに追加されます。カスタムクォータ通知メッセージ (クォータリソースタイプ別) を通知に追加できます。以下は例になります。

```
Your project is over quota. It is using 200% of 2 cores CPU (Limit).
Upgrade
to <a href='https://www.openshift.com'>OpenShift Online Pro</a> if you
need
additional resources.
```

通知の「Upgrade to...」の部分はカスタムメッセージで、追加リソースへのリンクなどの HTML を指定できます。

注記

クォータメッセージは HTML マークアップなので、特殊文字はすべて HTML 向けに正しくエスケープする必要があります。

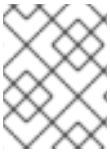
`window.OPENSHIFT_CONSTANTS.QUOTA_NOTIFICATION_MESSAGE` プロパティーを拡張スクリプトに設定して、各リソースについてメッセージをカスタマイズします。

```
// Set custom notification messages per quota type/key
window.OPENSHIFT_CONSTANTS.QUOTA_NOTIFICATION_MESSAGE = {
  'pods': 'Upgrade to <a href="https://www.openshift.com">OpenShift Online
Pro</a> if you need additional resources.',
  'limits.memory': 'Upgrade to <a
href="https://www.openshift.com">OpenShift Online Pro</a> if you need
additional resources.'
};
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.14. CREATE FROM URL NAMESPACE ホワイトリストの設定

Create from URL は、明示的に OPENSHIFT_CONSTANTS.CREATE_FROM_URL_WHITELIST に指定される namespace からのイメージストリームまたはテンプレートとのみ機能します。namespace をホワイトリストに追加するには、以下の手順に従います。



注記

openshift はデフォルトでホワイトリストに含まれています。これは削除しないでください。

```
// Add a namespace containing the image streams and/or templates
window.OPENSHIFT_CONSTANTS.CREATE_FROM_URL_WHITELIST.push(
  'shared-stuff'
);
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.15. COPY LOGIN コマンドの無効化

Web コンソールではユーザーは、現在のアクセストークンなどのログインコマンドをユーザーメニューおよび Command Line Tools ページからクリップボードにコピーできます。この機能は、ユーザーのアクセストークンがコピーされたコマンドに含まれないように変更することができます。

```
// Do not copy the user's access token in the copy login command.
window.OPENSHIFT_CONSTANTS.DISABLE_COPY_LOGIN_COMMAND = true;
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

35.15.1. ワイルドカードルートの有効化

ワイルドカードルートをルーターで有効にした場合、Web コンソールでもワイルドカードルートを有効にできます。これによりユーザーはルートを作成するときに、*.example.com などのアスタリスクで始まるホスト名を入力できます。ワイルドカードルートを有効にするには、以下を実行します。

```
window.OPENSHIFT_CONSTANTS.DISABLE_WILDCARD_ROUTES = false;
```

「拡張スクリプトとスタイルシートの読み込み」で説明されているようにスクリプトを追加します。

HAProxy をワイルドカードルートを許可するように設定する方法についてはこちらを参照してください。

35.16. ログインページのカスタマイズ

Web コンソールのログインページおよびログインプロバイダー選択ページも変更できます。以下のコマンドを実行して、変更可能なテンプレートを作成します。

```
$ oc adm create-login-template > login-template.html
$ oc adm create-provider-selection-template > provider-selection-
template.html
```

ファイルを編集して、スタイルを変更するか、または内容を追加します。ただし、波括弧内の必須パラメーターを削除しないよう注意してください。

カスタムログインページまたはプロバイダー選択ページを使用するには、以下のオプションをマスター設定ファイルに設定します。

```
oauthConfig:
  ...
  templates:
    login: /path/to/login-template.html
    providerSelection: /path/to/provider-selection-template.html
```

相対パスは、マスター設定ファイルを基準にして解決されます。この設定を変更したらサーバーを再起動する必要があります。

複数のログインプロバイダーが設定されている場合、または master-config.yaml ファイルの `alwaysShowProviderSelection` オプションが true に設定されている場合、OpenShift Container Platform へのユーザーのトークンが期限切れになるたびに、このカスタムページが他のタスクに進む前にユーザーに表示されます。

35.16.1. 使用例

サービス利用規約情報はカスタムログインページを使用して作成できます。このようなページは、GitHub や Google などのサードパーティーログインプロバイダーを使用している場合にも、ユーザーが信頼し、予想できるブランドのページを提示して、その後にユーザーを認証プロバイダーにリダイレクトする際に役立ちます。

35.17. OAUTH エラーページのカスタマイズ

認証中にエラーが発生した場合に表示されるページを変更できます。

1. 以下のコマンドを実行して、変更可能なテンプレートを作成します。

```
$ oc adm create-error-template > error-template.html
```

2. ファイルを編集して、スタイルを変更するか、または内容を追加します。
テンプレートで `Error` 変数および `ErrorCode` 変数を使用できます。カスタムエラーページを使用するには、以下のオプションをマスター設定ファイルに設定します。

```
oauthConfig:
  ...
  templates:
    error: /path/to/error-template.html
```

相対パスは、マスター設定ファイルを基準にして解決されます。

3. この設定を変更したらサーバーを再起動する必要があります。

35.18. ログアウト URL の変更

`webconsole-config` ConfigMap の `clusterInfo.logoutPublicURL` パラメーターを変更すると、コンソールをログアウトしたときにコンソールユーザーに表示される場所を変更できます。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

以下の例では、ログアウト URL を https://www.example.com/logout に変更しています。

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    clusterInfo:
      [...]
      logoutPublicURL: "https://www.example.com/logout"
    [...]
```

これは、要求ヘッダーおよび OAuth または OpenID アイデンティティープロバイダーで認証する場合に便利です。この場合、外部 URL にアクセスしてシングルサインオンセッションを破棄する必要があるからです。

35.19. ANSIBLE による WEB コンソールカスタマイズの設定

クラスターのインストール時に、以下のパラメーターを使用して、Web コンソールへの変更を多数設定できます。これらのパラメーターは、インベントリーファイルで設定可能です。

- openshift_master_logout_url
- openshift_web_console_extension_script_urls
- openshift_web_console_extension_stylesheet_urls
- openshift_master_oauth_templates
- openshift_master_metrics_public_url
- openshift_master_logging_public_url

Ansible による Web コンソールカスタマイズの例

```
# Configure `clusterInfo.logoutPublicURL` in the web console configuration
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_cu
stomization.html#changing-the-logout-url
#openshift_master_logout_url=https://example.com/logout

# Configure extension scripts for web console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_cu
stomization.html#loading-custom-scripts-and-stylesheets
#openshift_web_console_extension_script_urls=
['https://example.com/scripts/menu-
customization.js','https://example.com/scripts/nav-customization.js']

# Configure extension stylesheets for web console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_cu
```

```
stomization.html#loading-custom-scripts-and-stylesheets
#openshift_web_console_extension_stylesheet_urls=
['https://example.com/styles/logo.css','https://example.com/styles/custom-
styles.css']

# Configure a custom login template in the master config
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_cu
stomization.html#customizing-the-login-page
#openshift_master_oauth_templates={'login': '/path/to/login-
template.html'}

# Configure `clusterInfo.metricsPublicURL` in the web console
configuration for cluster metrics. Ansible is also able to configure
metrics for you.
# See:
https://docs.openshift.com/enterprise/latest/install_config/cluster_metric
s.html
#openshift_master_metrics_public_url=https://hawkular-
metrics.example.com/hawkular/metrics

# Configure `clusterInfo.loggingPublicURL` in the web console
configuration for aggregate logging. Ansible is also able to install
logging for you.
# See:
https://docs.openshift.com/enterprise/latest/install_config/aggregate_logg
ing.html
#openshift_master_logging_public_url=https://kibana.example.com
```

第36章 外部永続ボリュームプロビジョナーのデプロイ

36.1. 概要

外部プロビジョナーは、特定のストレージプロバイダーの動的プロビジョニングを可能にするアプリケーションです。外部プロビジョナーは、OpenShift Container Platform が提供するプロビジョナープラグインと共に実行でき、StorageClass オブジェクトの設定と同じ方法で設定されます。この方法は、「動的プロビジョニングとストレージクラスの作成」のセクションで説明しています。これらのプロビジョナーは外部プロビジョナーであるため、OpenShift Container Platform とは独立してデプロイし、更新できます。

36.2. 作業を開始する前に

外部プロビジョナーのデプロイとアップグレードには、Ansible Playbook も使用できます。

注記

「クラスターメトリクスの設定」と「クラスターロギングの設定」のセクションを十分に理解してから、先に進むようにしてください。

36.2.1. 外部プロビジョナーの Ansible ロール

OpenShift Ansible `openshift_provisioners` ロールは Ansible インベントリーファイルからの変数を使用して、外部プロビジョナーを設定し、デプロイします。それぞれの `install` 変数を `true` に上書きして、インストールするプロビジョナーを指定する必要があります。

36.2.2. 外部プロビジョナーの Ansible 変数

以下は、`install` 変数が `true` で、すべてのプロビジョナーに適用されるロール変数の一覧です。

表36.1 Ansible 変数

変数	説明
`openshift_provisioners_install_provisioners`	`true` の場合、それぞれの `install` 変数が `true` に設定されているすべてのプロビジョナーをデプロイします。それ以外の場合は削除します。
`openshift_provisioners_image_prefix`	コンポーネントイメージのプレフィックス。たとえば `openshift3/efs-provisioner:v3.6` の場合、プレフィックスを `openshift3/` に設定します。
`openshift_provisioners_image_version`	コンポーネントイメージのバージョン。たとえば `openshift3/efs-provisioner:v3.6` の場合、バージョンを `v3.6` に設定します。
`openshift_provisioners_project`	プロビジョナーのデプロイ先のプロジェクト。デフォルトは `openshift-infra` です。

36.2.3. AWS EFS プロビジョナーの Ansible 変数

AWS EFS プロビジョナーは、所定の EFS ファイルシステムのディレクトリーに動的に作成されたディレクトリーを基盤とする NFS PV を動的にプロビジョニングします。AWS EFS プロビジョナー Ansible 変数を設定するには、以下の要件を満たす必要があります。

- AmazonElasticFileSystemReadOnlyAccess *ポリシー (またはこれ以上) を割り当てられた IAM ユーザー。*
- *クラスターリージョン内の EFS ファイルシステム。*
- *任意のノード (クラスターのリージョン内の任意のゾーン) がファイルシステムの DNS 名で EFS ファイルシステムをマウントできる、マウントターゲットとセキュリティーグループ。*

表36.2 必須の EFS Ansible 変数

変数	説明
`openshift_provisioners_efs_fsid`	EFS ファイルシステムのファイルシステム ID。例: `fs-47a2c22e`
`openshift_provisioners_efs_region`	EFS ファイルシステムの Amazon EC2 リージョン
`openshift_provisioners_efs_aws_access_key_id`	IAM ユーザーの AWS アクセスキー (指定された EFS ファイルシステムが存在するかチェックする)
`openshift_provisioners_efs_aws_secret_access_key`	IAM ユーザーの AWS シークレットアクセスキー (指定された EFS ファイルシステムが存在するかチェックする)

表36.3 オプションの EFS Ansible 変数

変数	説明
`openshift_provisioners_efs`	`true` の場合、AWS EFS プロビジョナーは `openshift_provisioners_install_provisioners` が `true` か `false` によって、インストールまたはアンインストールされます。デフォルトは `false` です。
`openshift_provisioners_efs_path`	EFS ファイルシステム内のディレクトリーのパスで、ここで EFS プロビジョナーがディレクトリーを作成して、作成する各 PV をサポートします。これは存在している必要があり、EFS プロビジョナーでマウントできる必要があります。デフォルトは `/persistentvolumes` です。
`openshift_provisioners_efs_name`	StorageClass で指定する `provisioner` 名。デフォルトは `openshift.org/aws-efs` です。

変数	説明
openshift_provisioners_efs_nodeselector	Pod が配置されるノードを選択するラベルのマップ。例: **{"node":"infra","region":"west"}**
openshift_provisioners_efs_supplementalgroup	EFS ファイルシステムへの書き込みパーミッションのために必要な場合の、Pod に指定する補助グループ。デフォルトは **65534** です。

36.3. プロビジョナーのデプロイ

OpenShift Ansible 変数で指定される設定に従って、すべてのプロビジョナーを同時にデプロイすることも、プロビジョナーを 1 つずつデプロイすることもできます。以下の例では、指定されたプロビジョナーをデプロイして、対応する StorageClass を作成し、設定する方法を示します。

36.3.1. AWS EFS プロビジョナーのデプロイ

以下のコマンドは、EFS ボリューム内のディレクトリーを /data/persistentvolumes に設定します。このディレクトリーはファイルシステム内に存在している必要があり、プロビジョナー Pod によってマウントや書き込みができる必要があります。

```
$ ansible-playbook -v -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
provisioners/config.yml \
   -e openshift_provisioners_install_provisioners=True \
   -e openshift_provisioners_efs=True \
   -e openshift_provisioners_efs_fsid=fs-47a2c22e \
   -e openshift_provisioners_efs_region=us-west-2 \
   -e openshift_provisioners_efs_aws_access_key_id=AKIAIOSFODNN7EXAMPLE \
   -e
openshift_provisioners_efs_aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPx
RfiCYEXAMPLEKEY \
   -e openshift_provisioners_efs_path=/data/persistentvolumes
```

36.3.1.1. AWS EFS オブジェクト定義

aws-efs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: openshift.org/aws-efs (1)
parameters:
  gidMin: "40000" (2)
  gidMax: "50000" (3)
```

(1) *この値を openshift_provisioners_efs_name 変数と同じ値に設定します。デフォルトは openshift.org/aws-efs です。*

2. StorageClass の GID 範囲の最小値 (オプション)。

3. StorageClass の GID 範囲の最大値 (オプション)。

動的にプロビジョニングされた各ボリュームに対応する NFS ディレクトリーには、`gidMin` から `gidMax` の範囲に一意の GID 所有者が割り当てられます。指定されない場合、デフォルトで `gidMin` は`2000`、`gidMax` は `2147483647` になります。要求によってプロビジョニングされるボリュームを使用するすべての Pod は、必要な GID を補助グループとして自動的に実行され、ボリュームの読み取り/書き込みができます。補助グループを持たない (かつルートとして実行されていない) 他のマウンターは、ボリュームの読み取り/書き込みはできません。補助グループを使用して NFS アクセスを管理する方法については、「グループ ID」セクションの NFS ボリュームセキュリティーのトピックを参照してください。

36.4. クリーンアップ

以下のコマンドを実行すると、OpenShift Ansible `openshift_provisioners` ロールによってデプロイされたものをすべて削除できます。

```
$ ansible-playbook -v -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
provisioners/config.yml \
   -e openshift_provisioners_install_provisioners=False
```