



OpenShift Container Platform 3.9

クラスター管理

OpenShift Container Platform 3.9 クラスター管理

OpenShift Container Platform 3.9 クラスター管理

OpenShift Container Platform 3.9 クラスター管理

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 クラスター管理』では、OpenShift クラスターを管理するための通常のタスクや他の詳細設定についてのトピックを扱います。

目次

第1章 概要	11
第2章 ノードの管理	12
2.1. 概要	12
2.2. ノードの一覧表示	12
2.3. ノードの追加	13
2.4. ノードの削除	13
2.5. ノードのラベルの更新	14
2.6. ノード上の POD の一覧表示	14
2.7. ノードをスケジュール対象外 (UNSCHEDULABLE) またはスケジュール対象 (SCHEDULABLE) としてマークする	14
2.8. POD のノードからの退避	15
2.9. ノードの再起動	16
2.9.1. インフラストラクチャーノード	16
2.9.2. Pod の非アフィニティーの使用	16
2.9.3. ルーターを実行するノードの処理	17
2.10. ノードリソースの設定	17
2.10.1. ノードあたりの最大 Pod 数の設定	18
2.11. DOCKER ストレージの再設定	19
2.12. ノードトラフィックインターフェースの変更	21
第3章 ユーザーの管理	22
3.1. 概要	22
3.2. ユーザーの作成	22
3.3. ユーザーおよび ID リストの表示	22
3.4. グループの作成	23
3.5. ユーザーおよびグループラベルの管理	24
3.6. ユーザーの削除	24
第4章 プロジェクトの管理	26
4.1. 概要	26
4.2. プロジェクトのセルフプロビジョニング	26
4.2.1. 新規プロジェクトのテンプレートの変更	26
4.2.2. セルフプロビジョニングの無効化	27
4.3. ノードセクターの使用	27
4.3.1. クラスター全体でのデフォルトノードセクターの設定	27
4.3.2. プロジェクト全体でのノードセクターの設定	28
4.3.3. 開発者が指定するノードセクター	29
4.4. ユーザーあたりのセルフプロビジョニングされたプロジェクト数の制限	29
第5章 POD の管理	31
5.1. 概要	31
5.2. 1 回実行 (RUN-ONCE) POD 期間の制限	31
5.2.1. RunOnceDuration プラグインの設定	31
5.2.2. プロジェクト別のカスタム期間の指定	31
5.2.2.1. Egress ルーター Pod のデプロイ	32
5.2.2.2. Egress ルーターサービスのデプロイ	33
5.2.3. Egress ファイアウォールでの Pod アクセスの制限	33
5.2.3.1. Pod アクセス制限の設定	34
5.3. POD で利用可能な帯域幅の制限	35
5.4. POD の DISRUPTION BUDGET (停止状態の予算) の設定	36
5.5. POD の PRESET (プリセット) を使用した情報の POD への挿入	37

第6章 ネットワークの管理	39
6.1. 概要	39
6.2. POD ネットワークの管理	39
6.2.1. プロジェクトネットワークへの参加	39
6.3. プロジェクトネットワークの分離	39
6.3.1. プロジェクトネットワークのグローバル化	39
6.4. ルートおよび INGRESS オブジェクトにおけるホスト名の競合防止の無効化	40
6.5. EGRESS トラフィックの制御	41
6.5.1. 外部リソースへのアクセスを制限するための Egress ファイアウォールの使用	41
6.5.2. 外部リソースから Pod トラフィックを認識可能にするための Egress ルーターの使用	44
6.5.2.1. リダイレクトモードでの Egress ルーター Pod のデプロイ	45
6.5.2.2. 複数の宛先へのリダイレクト	47
6.5.2.3. ConfigMap の使用による EGRESS_DESTINATION の指定	48
6.5.2.4. Egress ルーター HTTP プロキシ Pod のデプロイ	49
6.5.2.5. Egress ルーター Pod のフェイルオーバーの有効化	51
6.5.3. 外部リソースへのアクセスを制限するための iptables ルールの使用	52
6.6. 外部プロジェクトトラフィックの静的 IP の有効化	53
6.7. マルチキャストの有効化	54
6.8. NETWORKPOLICY の有効化	54
6.8.1. NetworkPolicy およびルーター	56
6.8.2. 新規プロジェクトのデフォルト NetworkPolicy の設定	57
6.9. HTTP STRICT TRANSPORT SECURITY の有効化	57
6.10. スループットの問題のトラブルシューティング	58
第7章 サービスアカウントの設定	60
7.1. 概要	60
7.2. ユーザー名およびグループ	60
7.3. サービスアカウントの管理	61
7.4. サービスアカウント認証の有効化	62
7.5. 管理サービスアカウント	62
7.6. インフラストラクチャーサービスアカウント	63
7.7. サービスアカウントおよびシークレット	63
第8章 ロールベースアクセス制御 (RBAC) の管理	64
8.1. 概要	64
8.2. ロールとバインディングの表示	64
8.2.1. クラスターロールの表示	64
8.2.2. ローカルのロールバインディングの表示	73
8.3. ロールバインディングの管理	74
8.4. ローカルロールの作成	76
8.5. クラスターおよびローカルのロールバインディング	77
第9章 イメージポリシー	79
9.1. 概要	79
9.2. インポート用に許可されるレジストリーの設定	79
9.3. IMAGEPOLICY 受付プラグインの設定	80
9.4. IMAGEPOLICY 受付プラグインのテスト	82
第10章 イメージの署名	84
10.1. 概要	84
10.2. ATOMIC CLI を使用したイメージの署名	84
10.3. OPENSIFT CLI を使用したイメージ署名の検証	85
10.4. レジストリー API の使用によるイメージ署名へのアクセス	86
10.4.1. API 経由でのイメージ署名の書き込み	86

10.4.2. API 経由でのイメージ署名の読み取り	87
10.4.3. 署名ストアからのイメージ署名の自動インポート	87
第11章 スコープ付きトークン	89
11.1. 概要	89
11.2. 評価	89
11.3. ユーザースコープ	89
11.4. ロールスコープ	89
第12章 イメージのモニタリング	91
12.1. 概要	91
12.2. イメージ統計の表示	91
12.3. IMAGESTREAMS 統計の表示	91
12.4. イメージのプルーニング	92
第13章 SCC (SECURITY CONTEXT CONSTRAINTS) の管理	93
13.1. 概要	93
13.2. SCC (SECURITY CONTEXT CONSTRAINTS) の一覧表示	93
13.3. SCC (SECURITY CONTEXT CONSTRAINTS) オブジェクトの検査	93
13.4. 新規 SCC (SECURITY CONTEXT CONSTRAINTS) の作成	94
13.5. SCC (SECURITY CONTEXT CONSTRAINTS) の削除	95
13.6. SCC (SECURITY CONTEXT CONSTRAINTS) の更新	96
13.6.1. SCC (Security Context Constraints) 設定のサンプル	96
13.7. デフォルト SCC (SECURITY CONTEXT CONSTRAINTS) の更新	97
13.8. 使用方法	97
13.8.1. 特権付き SCC のアクセス付与	97
13.8.2. 特権付き SCC のサービスアカウントアクセスの付与	98
13.8.3. Dockerfile の USER によるイメージ実行の有効化	98
13.8.4. ルートを要求するコンテナイメージの有効化	99
13.8.5. レジストリーでの --mount-host の使用	99
13.8.6. 追加機能の提供	99
13.8.7. クラスターのデフォルト動作の変更	100
13.8.8. hostPath ボリュームプラグインの使用	100
13.8.9. 受付を使用した特定 SCC の初回使用	100
13.8.10. SCC のユーザー、グループまたはプロジェクトへの追加	101
第14章 スケジューリング	102
14.1. 概要	102
14.1.1. 概要	102
14.1.2. デフォルトスケジューリング	102
14.1.3. 詳細スケジューリング	102
14.1.4. カスタムスケジューリング	102
14.2. デフォルトスケジューリング	102
14.2.1. 概要	102
14.2.2. 汎用スケジューラー	102
14.2.3. ノードのフィルター	103
14.2.3.1. フィルターされたノード一覧の優先順位付け	103
14.2.3.2. 最適ノードの選択	103
14.2.4. スケジューラーポリシー	103
14.2.4.1. スケジューラーポリシーの変更	105
14.2.5. 利用可能な述語	106
14.2.5.1. 静的な述語	106
14.2.5.1.1. デフォルトの述語	106
14.2.5.1.2. 他の静的な述語	107

14.2.5.2. 汎用的な述語	108
汎用的な非クリティカル述語	108
汎用的なクリティカル述語	108
14.2.5.3. 設定可能な述語	109
14.2.6. 利用可能な優先度	111
14.2.6.1. 静的優先度	111
14.2.6.1.1. デフォルトの優先度	111
14.2.6.1.2. 他の静的優先度	112
14.2.6.2. 設定可能な優先度	112
14.2.7. 使用例	114
14.2.7.1. インフラストラクチャーのトポロジレベル	114
14.2.7.2. アフィニティー	114
14.2.7.3. 非アフィニティー	114
14.2.8. ポリシー設定のサンプル	115
14.3. カスタムスケジューリング	117
14.3.1. 概要	117
14.3.2. スケジューラーのデプロイ	117
14.4. POD 配置の制御	119
14.4.1. 概要	119
14.4.2. ノード名の使用による Pod 配置の制約	119
14.4.3. ノードセクターの使用による Pod 配置の制約	120
14.4.4. プロジェクト対する Pod 配置の制御	122
14.5. 詳細スケジューリング	124
14.5.1. 概要	124
14.5.2. 詳細スケジューリングの使用	125
14.6. 詳細スケジューリングおよびノードのアフィニティー	125
14.6.1. 概要	125
14.6.2. ノードのアフィニティーの設定	126
14.6.2.1. ノードアフィニティーの required (必須) ルールの設定	128
14.6.2.2. ノードアフィニティーの preferred (優先) ルールの設定	129
14.6.3. 各種の例	129
14.6.3.1. 一致するラベルを持つノードのアフィニティー	129
14.6.3.2. 一致するラベルのないノードのアフィニティー	130
14.7. 詳細スケジューリングおよび POD のアフィニティーと非アフィニティー	131
14.7.1. 概要	131
14.7.2. Pod のアフィニティーおよび非アフィニティーの設定	131
14.7.2.1. アフィニティールールの設定	133
14.7.2.2. 非アフィニティールールの設定	134
14.7.3. 各種の例	135
14.7.3.1. Pod のアフィニティー	135
14.7.3.2. Pod の非アフィニティー	136
14.7.3.3. 一致するラベルのない Pod のアフィニティー	137
14.8. 詳細スケジューリングおよびノードセクター	138
14.8.1. 概要	138
14.8.2. ノードセクターの設定	138
14.9. 詳細スケジューリングおよび容認	139
14.9.1. 概要	139
14.9.2. テイントおよび容認 (Toleration)	139
14.9.2.1. 複数テイントの使用	141
14.9.3. テイントの既存ノードへの追加	142
14.9.4. 容認の Pod への追加	142
14.9.4.1. Pod のエビクションを遅延させる容認期間 (秒数) の使用	142
14.9.4.1.1. 容認の秒数のデフォルト値の設定	143

14.9.5. ノードの問題の発生時における Pod エビクションの禁止	144
14.9.6. Daemonset および容認	145
14.9.7. 各種の例	145
14.9.7.1. ノードをユーザー専用にする	145
14.9.7.2. ユーザーのノードへのバインド	145
14.9.7.3. 特殊ハードウェアを持つノード	146
第15章 クォータの設定	147
15.1. 概要	147
15.2. クォータで管理されるリソース	147
15.3. クォータのスコープ	148
15.4. クォータの実施	149
15.5. 要求 VS 制限	149
15.6. リソースクォータ定義のサンプル	150
15.7. クォータの作成	153
15.8. クォータの表示	153
15.9. クォータの同期期間の設定	154
15.10. デプロイメント設定におけるクォータアカウンティング	154
15.11. リソース消費における明示的なクォータの要求	154
第16章 複数プロジェクトのクォータ設定	156
16.1. 概要	156
16.2. プロジェクトの選択	156
16.3. 適用可能な CLUSTERRESOURCEQUOTAS の表示	157
16.4. 選択における粒度	158
第17章 制限範囲の設定	159
17.1. 概要	159
17.1.1. コンテナの制限	160
17.1.2. Pod の制限	161
17.1.3. イメージの制限	162
17.1.4. イメージストリームの制限	163
17.1.4.1. イメージ参照の数	163
17.1.5. PersistentVolumeClaim の制限	164
17.2. 制限範囲の作成	164
17.3. 制限の表示	165
17.4. 制限の削除	165
第18章 オブジェクトのプルーニング	166
18.1. 概要	166
18.2. プルーニングの基本操作	166
18.3. デプロイメントのプルーニング	166
18.4. ビルドのプルーニング	167
18.5. イメージのプルーニング	168
18.5.1. イメージのプルーニングの各種条件	170
18.5.2. セキュアまたは非セキュアな接続の使用	171
18.5.3. イメージのプルーニングに関する問題	172
イメージがプルーニングされない	172
非セキュアなレジストリーに対するセキュアな接続の使用	173
18.5.3.1. セキュリティが保護されたレジストリーに対する非セキュアな接続の使用	173
正しくない認証局の使用	173
18.6. レジストリーのハードプルーニング	174
18.7. CRON ジョブのプルーニング	177

第19章 カスタムリソースによる KUBERNETES API の拡張	178
19.1. カスタムリソース定義の作成	178
19.2. カスタムオブジェクトの作成	179
19.3. カスタムオブジェクトの管理	180
19.4. ファイナライザー	181
第20章 ガベージコレクション	182
20.1. 概要	182
20.2. コンテナのガベージコレクション	182
20.2.1. 削除するコンテナの検出	183
20.3. イメージのガベージコレクション	183
20.3.1. 削除するイメージの検出	184
第21章 ノードリソースの割り当て	185
21.1. 概要	185
21.2. 割り当てられるリソースについてのノードの設定	185
21.3. 割り当てられるリソースの計算	185
21.4. ノードの割り当て可能なリソースおよび容量の表示	186
21.5. ノードによって報告されるシステムリソース	186
21.6. ノードの実施	187
21.7. エビクションしきい値	188
21.8. スケジューラー	189
第22章 不透明な整数リソース	190
22.1. 概要	190
22.2. 不透明な整数リソースの作成	190
第23章 オーバーコミット	193
23.1. 概要	193
23.2. 要求および制限	193
23.2.1. Buffer Chunk Limit の調整	193
23.3. コンピュートリソース	194
23.3.1. CPU	194
23.3.2. メモリー	194
23.4. QOS (QUALITY OF SERVICE) クラス	194
23.5. マスターでのオーバーコミットの設定	195
23.6. ノードでのオーバーコミットの設定	196
23.6.1. Quality of Service (QoS) 層でのメモリー予約	196
23.6.2. CPU 制限の実施	197
23.6.3. システムリソースのリソース予約	197
23.6.4. カーネルの調整可能なフラグ	199
23.6.5. swap メモリーの無効化	199
第24章 INGRESS トラフィックの固有の外部 IP の割り当て	200
24.1. 概要	200
24.2. 制限	200
24.3. 固有の外部 IP を使用するようクラスターを設定する	201
24.3.1. サービスの Ingress IP の設定	201
24.4. 開発またはテスト目的での INGRESS CIDR のルーティング	202
24.4.1. サービス externalIP	202
第25章 OUT OF RESOURCE (リソース不足) エラーの処理	204
25.1. 概要	204
25.2. エビクションポリシーの設定	204
25.2.1. ノード設定を使用したポリシーの作成	205

25.2.2. エビクションシグナルについて	206
25.2.3. エビクションのしきい値について	208
25.2.3.1. ハードエビクションのしきい値について	209
25.2.3.1.1. デフォルトのハードエビクションしきい値	209
25.2.3.2. ソフトエビクションのしきい値について	209
25.3. スケジューリング用のリソース量の設定	210
25.4. ノードの状態変動の制御	211
25.5. ノードレベルのリソースの回収	211
Imagefs が設定されている場合	211
Imagefs が設定されていない場合	212
25.6. POD エビクションについて	212
25.6.1. QoS および Out of Memory Killer について	213
25.7. POD スケジューラーおよび OOR 状態について	213
25.8. シナリオ例	214
25.9. 推奨される対策	215
25.9.1. DaemonSet および Out of Resource (リソース不足) の処理	215
第26章 ルーターのモニタリングおよびデバッグ	216
26.1. 概要	216
26.2. 統計の表示	216
26.3. 統計ビューの無効化	216
26.4. ログの表示	216
26.5. ルーター内部の表示	217
第27章 高可用性	219
27.1. 概要	219
27.2. IP フェイルオーバーの設定	220
27.2.1. 仮想 IP アドレス	221
27.2.2. チェックおよび通知スクリプト	221
27.2.3. VRRP プリエンプション	223
27.2.4. Keepalived マルチキャスト	224
27.2.5. コマンドラインオプションおよび環境変数	225
27.2.6. VRRP ID オフセット	226
27.2.7. 高可用サービスの設定	226
27.2.7.1. IP フェイルオーバー Pod のデプロイ	228
27.2.8. 高可用サービスの仮想 IP の動的更新	229
27.3. サービスの EXTERNALIP および NODEPORT の設定	230
27.4. INGRESSIP の高可用性	230
第28章 IPTABLES	231
28.1. 概要	231
28.2. IPTABLES	231
28.3. IPTABLES.SERVICE	231
第29章 ストラテジーによるビルドのセキュリティー保護	233
29.1. 概要	233
29.2. ビルドストラテジーのグローバルな無効化	233
29.3. ユーザーへのビルドストラテジーのグローバルな制限	234
29.4. プロジェクト内でのユーザーへのビルドストラテジーの制限	235
第30章 SECCOMP を使用したアプリケーション機能の制限	236
30.1. 概要	236
30.2. SECCOMP の有効化	236
30.3. OPENSIFT CONTAINER PLATFORM での SECCOMP の設定	236

30.4. OPENSIFT CONTAINER PLATFORM でのカスタム SECCOMP プロファイルの設定	237
第31章 SYSCTL	238
31.1. 概要	238
31.2. SYSCTL について	238
31.3. NAMESPACE を使用した SYSCTL VS ノードレベルの SYSCTL	238
31.4. 安全 VS 安全でない SYSCTL	239
31.5. 安全でない SYSCTL の有効化	239
31.6. POD 用の SYSCTL の設定	240
第32章 データストア層でのデータの暗号化	241
32.1. 概要	241
32.2. 設定および暗号がすでに有効にされているかどうかの判別	241
32.3. 暗号化設定について	241
32.3.1. 利用可能なプロバイダー	242
32.4. データの暗号化	243
32.5. データが暗号化されていることの確認	244
32.6. すべてのシークレットが暗号化されていることの確認	244
32.7. 復号化キーのローテーション	245
32.8. データの復号化	245
第33章 IPSEC を使用したホストの暗号化	247
33.1. 概要	247
33.2. ホストの暗号化	247
33.2.1. 前提条件	247
33.2.2. 証明書での IPsec の設定	247
33.2.3. libreswan IPsec ポリシー	248
33.2.3.1. 便宜的なグループ (opportunistic group) 設定	248
33.2.3.2. 明示的な通信設定	249
33.3. IPSEC ファイアウォール設定	250
33.4. IPSEC の開始および終了	251
33.5. IPSEC の最適化	251
33.6. トラブルシューティング	251
第34章 依存関係ツリーのビルド	252
34.1. 概要	252
34.2. 使用法	252
第35章 バックアップおよび復元	253
35.1. 概要	253
35.2. 前提条件	253
35.3. クラスターのバックアップ	254
35.3.1. マスターのバックアップ	254
35.3.2. etcd のバックアップ	254
35.3.3. レジストリー証明書のバックアップ	255
35.4. 単一メンバーの ETCD クラスター用のクラスターの復元	255
35.5. 複数メンバーで構成される ETCD クラスター用のクラスターの復元	256
35.5.1. コンテナ化された etcd デプロイメント	257
35.5.2. コンテナ化されていない etcd デプロイメント	258
35.5.3. etcd メンバーの追加	258
35.6. 新規 ETCD ホストの追加	260
35.7. OPENSIFT CONTAINER PLATFORM サービスの再オンライン化	264
35.8. プロジェクトのバックアップ	265
35.8.1. ロールバックインディング	265

35.8.2. サービスアカウント	265
35.8.3. シークレット	265
35.8.4. Persistent Volume Claim (永続ボリューム要求、PVC)	266
35.9. プロジェクトの復元	266
35.10. アプリケーションデータのバックアップ	266
35.11. アプリケーションデータの復元	267
第36章 OPENSIFT SDN のトラブルシューティング	269
36.1. 概要	269
36.2. 用語	269
36.3. HTTP サービスへの外部アクセスのデバッグ	270
36.4. ルーターのデバッグ	271
36.5. サービスのデバッグ	272
36.6. ノード間通信のデバッグ	273
36.7. ローカルネットワークのデバッグ	275
36.7.1. ノードのインターフェース	275
36.7.2. ノード内の SDN フロー	275
36.7.3. デバッグ手順	275
36.7.3.1. IP 転送は有効にされているか?	275
36.7.3.2. ルートは正しく設定されているか?	276
36.7.4. Open vSwitch は正しく設定されているか?	276
36.7.4.1. iptables 設定に誤りがないか?	277
36.7.4.2. 外部ネットワークは正しく設定されているか?	277
36.8. 仮想ネットワークのデバッグ	277
36.8.1. 仮想ネットワークのビルドに障害が発生している	277
36.9. POD の EGRESS のデバッグ	278
36.10. ログの読み取り	278
36.11. KUBERNETES のデバッグ	279
36.12. 診断ツールを使用したネットワークの問題の検出	279
36.13. その他の注意点	279
36.13.1. ingress についての追加情報	279
36.13.2. TLS ハンドシェイクのタイムアウト	280
36.13.3. デバッグについての他の注意点	280
第37章 診断ツール	281
37.1. 概要	281
37.2. 診断ツールの使用	281
37.3. サーバー環境における診断の実行	284
37.4. クライアント環境での診断の実行	284
37.5. ANSIBLE ベースの正常性チェック	284
37.5.1. ansible-playbook による正常性チェックの実行	287
37.5.2. Docker CLI での正常性チェックの実行	288
第38章 アプリケーションのアイドルリング	289
38.1. 概要	289
38.2. アプリケーションのアイドルリング	289
38.2.1. 単一サービスのアイドルリング	289
38.2.2. 複数サービスのアイドルリング	289
38.3. アプリケーションのアイドルリング解除	290
第39章 クラスター容量の分析	291
39.1. 概要	291
39.2. コマンドラインでのクラスター容量分析の実行	291
39.3. POD 内のジョブとしてのクラスター容量分析の実行	292

第1章 概要

『OpenShift クラスター管理』では、OpenShift Container Platform クラスターを管理するための日常的なタスクや他の詳細な設定についてのトピックを扱います。

第2章 ノードの管理

2.1. 概要

インスタンスにある **ノード** は、**CLI** を使用して管理することができます。

ノードの管理操作を実行すると、CLI は実際のノードホストの表現である **ノードオブジェクト** と対話します。**マスター** はノードオブジェクトの情報を使用し、**ヘルスチェック** でノードの検証を実行します。

2.2. ノードの一覧表示

マスターに認識されるすべてのノードを一覧表示するには、以下を実行します。

```
$ oc get nodes
NAME                                STATUS    ROLES    AGE    VERSION
master.example.com                 Ready    master    7h    v1.9.1+a0ce1bc657
node1.example.com                  Ready    compute    7h    v1.9.1+a0ce1bc657
node2.example.com                  Ready    compute    7h    v1.9.1+a0ce1bc657
```

単一ノードについての情報のみを一覧表示するには、**<node>** を完全なノード名に置き換えます。

```
$ oc get node <node>
```

これらのコマンドの出力にある **STATUS** 列には、ノードの以下の状態が表示されます。

表2.1 ノードの状態

状態	説明
Ready	ノードは StatusOK を返し、マスターから実行されるヘルスチェックをパスしています。
NotReady	ノードはマスターから実行されるヘルスチェックをパスしていません。
SchedulingDisabled	ノードへの Pod の 配置をスケジュール できません。



注記

STATUS 列には、CLI でノードの状態を検索できない場合にノードについて **Unknown** が表示されます。

現在の状態の理由を含む特定ノードについての詳細情報を取得するには、以下を実行します。

```
$ oc describe node <node>
```

以下に例を示します。

```
$ oc describe node node1.example.com
Name:      node1.example.com
Labels:    kubernetes.io/hostname=node1.example.com
```



```

CreationTimestamp: Wed, 10 Jun 2015 17:22:34 +0000
Conditions:
  Type      Status LastHeartbeatTime   LastTransitionTime  Reason           Message
  Ready     True   Wed, 10 Jun 2015 19:56:16 +0000   Wed, 10 Jun 2015 17:22:34
+0000      kubelet is posting ready status
Addresses: 127.0.0.1
Capacity:
  memory: 1017552Ki
  pods:   100
  cpu:    2
Version:
  Kernel Version:  3.17.4-301.fc21.x86_64
  OS Image:        Fedora 21 (Twenty One)
  Container Runtime Version: docker://1.6.0
  Kubelet Version: v0.17.1-804-g496be63
  Kube-Proxy Version: v0.17.1-804-g496be63
ExternalID: node1.example.com
Pods:      (2 in total)
  docker-registry-1-9yyw5
  router-1-maytv
No events.

```

2.3. ノードの追加

ノードを既存の OpenShift Container Platform クラスターに追加するには、ノードコンポーネントのインストール、必要な証明書の生成およびその他の重要な手順を処理する Ansible Playbook を実行できます。Playbook を直接実行する方法については、[通常インストール \(Advanced installation\)](#) 方式を参照してください。

またはクイックインストール方式を使用する場合は、[インストーラーを再実行してノードを追加](#)すると、同じ手順を実行できます。

2.4. ノードの削除

CLI を使用してノードを削除する場合、ノードオブジェクトは Kubernetes で削除されますが、ノード自体にある Pod は削除されません。レプリケーションコントローラーでサポートされないベア Pod は OpenShift Container Platform からアクセスできなくなり、レプリケーションコントローラーでサポートされる Pod は他の利用可能なノードにスケジュール変更されます。[ローカルのマニフェスト Pod](#) は手動で削除する必要があります。

OpenShift Container Platform クラスターからノードを削除するには、以下を実行します。

1. 削除しようとしているノードから [Pod を退避](#)します。
2. ノードオブジェクトを削除します。

```
$ oc delete node <node>
```

3. ノードがノード一覧から削除されていることを確認します。

```
$ oc get nodes
```

Pod は、**Ready** 状態にある残りのノードに対してのみスケジュールされます。

- すべての Pod およびコンテナを含む、OpenShift Container Platform のすべてのコンテンツをノードホストからアンインストールする必要がある場合は、[ノードのアンインストール](#)を継続し、**uninstall.yml** Playbook を使用する手順に従います。この手順では、Ansible を使用した[通常インストール \(Advanced installation\) 方式](#)についての全般的な理解があることが前提となります。

2.5. ノードのラベルの更新

ノードで[ラベル](#)を追加し、更新するには、以下を実行します。

```
$ oc label node <node> <key_1>=<value_1> ... <key_n>=<value_n>
```

詳細な使用法を表示するには、以下を実行します。

```
$ oc label -h
```

2.6. ノード上の POD の一覧表示

1 つ以上のノードにすべてまたは選択した Pod を一覧表示するには、以下を実行します。

```
$ oc adm manage-node <node1> <node2> \
  --list-pods [--pod-selector=<pod_selector>] [-o json|yaml]
```

選択したノードのすべてまたは選択した Pod を一覧表示するには、以下を実行します。

```
$ oc adm manage-node --selector=<node_selector> \
  --list-pods [--pod-selector=<pod_selector>] [-o json|yaml]
```

2.7. ノードをスケジュール対象外 (UNSCHEDULABLE) またはスケジュール対象 (SCHEDULABLE) としてマークする

デフォルトで、**Ready ステータス**の正常なノードはスケジュール対象としてマークされます。つまり、新規 Pod をこのノードに配置することができます。手動でノードをスケジュール対象外としてマークすると、新規 Pod のノードでのスケジューリングがブロックされます。ノード上の既存 Pod には影響がありません。

1 つまたは複数のノードをスケジュール対象外としてマークするには、以下を実行します。

```
$ oc adm manage-node <node1> <node2> --schedulable=false
```

以下に例を示します。

```
$ oc adm manage-node node1.example.com --schedulable=false
NAME                                LABELS
STATUS
node1.example.com    kubernetes.io/hostname=node1.example.com
Ready,SchedulingDisabled
```

現時点でスケジュール対象外のノードをスケジュール対象としてマークするには、以下を実行します。

```
$ oc adm manage-node <node1> <node2> --schedulable
```

または、特定のノード名 (例: **<node1> <node2>**) を指定する代わりに、**--selector=<node_selector>** オプションを使用して選択したノードをスケジュール対象またはスケジュール対象外としてマークすることができます。

2.8. POD のノードからの退避

Pod を退避することで、すべての Pod または選択した Pod を 1 つまたは複数の指定ノードから移行できます。Pod の退避を実行するには、まずノードを**スケジュール対象外としてマーク**する必要があります。

レプリケーションコントローラーでサポートされる Pod のみを退避できます。レプリケーションコントローラーは他のノードで新規 Pod を作成し、既存の Pod を指定したノードから削除します。デフォルトで、ベア Pod (レプリケーションコントローラーでサポートされていない Pod) はこの影響を受けません。

1 つ以上のノードですべてまたは選択した Pod を退避するには、以下を実行します。

```
$ oc adm drain <node1> <node2> [--pod-selector=<pod_selector>]
```

--force オプションを使用すると、ベア Pod の削除を強制的に実行できます。**true** に設定されると、Pod がレプリケーションコントローラー、ReplicaSet、ジョブ、daemonset、または StatefulSet で管理されていない場合でも削除が続行されます。

```
$ oc adm drain <node1> <node2> --force=true
```

--grace-period を使用して、各 Pod を正常に終了するための期間 (秒単位) を設定できます。負の値の場合には、Pod に指定されるデフォルト値が使用されます。

```
$ oc adm drain <node1> <node2> --grace-period=-1
```

--ignore-daemonsets を使用し、これを **true** に設定すると、Daemonset で管理された Pod を無視できます。

```
$ oc adm drain <node1> <node2> --ignore-daemonset=true
```

--timeout を使用すると、中止する前の待機期間を設定できます。値 **0** は無限の時間を設定します。

```
$ oc adm drain <node1> <node2> --timeout=5s
```

--delete-local-data を使用し、これを **true** に設定すると、Pod が emptyDir (ノードがドレイン (解放)) される場合に削除されるローカルデータ) を使用する場合でも削除が続行されます。

```
$ oc adm drain <node1> <node2> --delete-local-data=true
```

退避を実行せずに移行するオブジェクトを一覧表示するには、**--dry-run** オプションを使用し、これを **true** に設定します。

```
$ oc adm drain <node1> <node2> --dry-run=true
```

特定のノード名 (例: **<node1> <node2>**) を指定する代わりに、**--selector=<node_selector>** オプションを使用し、選択したノードで Pod を退避することができます。

2.9. ノードの再起動

プラットフォームで実行されるアプリケーションを停止せずにノードを再起動するには、まず **Pod の退避** を実行する必要があります。ルーティング階層によって可用性が高くされている Pod については、何も実行する必要はありません。ストレージ (通常はデータベース) を必要とするその他の Pod については、それらが 1 つの Pod が一時的にオフラインになっても作動したままになることを確認する必要があります。ステートフルな Pod の回復性はアプリケーションごとに異なりますが、いずれの場合でも、**ノードの非アフィニティ (node anti-affinity)** を使用して Pod が使用可能なノード間に適切に分散するようにスケジューラーを設定することが重要になります。

別の課題として、ルーターやレジストリーのような重要なインフラストラクチャーを実行しているノードを処理する方法を検討する必要があります。同じノードの退避プロセスが適用されますが、一部のエッジケースについて理解しておくことが重要です。

2.9.1. インフラストラクチャーノード

インフラストラクチャーノードは、OpenShift Container Platform 環境の一部を実行するためにラベルが付けられたノードです。現在、ノードの再起動を管理する最も簡単な方法として、インフラストラクチャーを実行するために利用できる 3 つ以上のノードを確保することができます。以下のシナリオでは、2 つのノードのみが利用可能な場合に OpenShift Container Platform で実行されるアプリケーションのサービスを中断しかねないよくある問題を示しています。

- ノード A がスケジューリング対象外としてマークされており、すべての Pod の退避が行われている。
- このノードで実行されているレジストリー Pod がノード B に再デプロイされる。これは、ノード B が両方のレジストリー Pod を実行していることを意味します。
- ノード B はスケジューリング対象外としてマークされ、退避が行われる。
- ノード B の 2 つの Pod エンドポイントを公開するサービスは、それらがノード A に再デプロイされるまでの短い期間すべてのエンドポイントを失う。

3 つのインフラストラクチャーノードを使用する同じプロセスではサービスの中断が生じません。しかし、Pod のスケジューリングにより、退避してローテーションに戻された最後のノードはゼロ (0) レジストリーを実行していることになり、他の 2 つのノードは 2 つのレジストリーと 1 つのレジストリーをそれぞれ実行します。最善の解決法として、Pod の非アフィニティを使用できます。これは現在テスト目的で利用できる Kubernetes のアルファ機能ですが、実稼働ワークロードに対する使用はサポートされていません。

2.9.2. Pod の非アフィニティの使用

Pod の非アフィニティ は、**ノードの非アフィニティ** とは若干異なります。ノードの非アフィニティの場合、Pod のデプロイ先となる適切な場所がほかにない場合には違反が生じます。Pod の非アフィニティの場合は **required** (必須) または **preferred** (優先) のいずれかに設定できます。

docker-registry Pod をサンプルとして使用すると、この機能を有効にするための最初の手順として、Pod に **scheduler.alpha.kubernetes.io/affinity** を設定します。この Pod はデプロイメント設定を使用するため、アノテーションを追加するのに適している場所は Pod テンプレートのメタデータになります。

```
$ oc edit dc/docker-registry -o yaml
```

```
...
  template:
    metadata:
```

```

annotations:
  scheduler.alpha.kubernetes.io/affinity: |
    {
      "podAntiAffinity": {
        "requiredDuringSchedulingIgnoredDuringExecution": [{
          "labelSelector": {
            "matchExpressions": [{
              "key": "docker-registry",
              "operator": "In",
              "values": ["default"]
            }]
          },
          "topologyKey": "kubernetes.io/hostname"
        }]
      }
    }
  }

```

重要

scheduler.alpha.kubernetes.io/affinity は、コンテンツが JSON の場合でも文字列として内部に保存されます。上記の例では、この文字列を YAML デプロイメント設定にアノテーションとして追加する方法を示しています。

この例では、Docker レジストリー Pod に **docker-registry=default** のラベルがあることを想定しています。Pod の非アフィニティーでは任意の Kubernetes の一致式を使用できます。

最後に必要となる手順として、**/etc/origin/master/scheduler.json** で **MatchInterPodAffinity** スケジューラーの述語を有効にします。これが有効にされると、2つのインフラストラクチャーノードのみが利用可能で、1つのノードが再起動される場合に、Docker レジストリー Pod が他のノードで実行されることが禁じられます。**oc get pods** は、適切なノードが利用可能になるまでこの Pod を **unready** (準備ができていない) として報告します。ノードが利用可能になると、すべての Pod は **Ready** (準備ができています) 状態に戻り、次のノードを再起動することができます。

2.9.3. ルーターを実行するノードの処理

ほとんどの場合、OpenShift Container Platform ルーターを実行する Pod はホストのポートを公開します。**PodFitsPorts** スケジューラーの述語により、同じポートを使用するルーター Pod が同じノードで実行されないようにし、Pod の非アフィニティーが適用されます。ルーターの高可用性を維持するために **IP フェイルオーバー** を利用している場合には、他に実行することはありません。高可用性を確保するために AWS Elastic Load Balancing などの外部サービスを使用するルーター Pod の場合は、そのような外部サービスがルーター Pod の再起動に対して対応します。

ルーター Pod でホストのポートが設定されていないということも稀にあります。この場合は、インフラストラクチャーノードについての[推奨される再起動プロセス](#)に従う必要があります。

2.10. ノードリソースの設定

ノードのリソースは、kubelet 引数をノード設定ファイル (**/etc/origin/node/node-config.yaml**) に追加して設定することができます。**kubeletArguments** セクションを追加し、必要なオプションを組み込みます。

```

kubeletArguments:
  max-pods: 1

```

```

- "40"
resolv-conf: ❷
- "/etc/resolv.conf"
image-gc-high-threshold: ❸
- "90"
image-gc-low-threshold: ❹
- "80"

```

- ❶ この kubelet で実行できる Pod の最大数。
- ❷ コンテナ DNS 解決設定のベースとして使用されるリゾルバーの設定ファイル。
- ❸ イメージのガベージコレクションが常に実行される場合のディスク使用量のパーセント。デフォルト: 90%
- ❹ イメージのガベージコレクションが一度も実行されない場合のディスク使用量のパーセント。デフォルト: 80%

利用可能なすべての kubelet オプションを表示するには、以下を実行します。

```
$ kubelet -h
```

この設定は、[通常インストール \(Advanced installation\)](#) の実行時に `openshift_node_kubelet_args` 変数を使用して実行できます。以下は例になります。

```
openshift_node_kubelet_args={'max-pods': ['40'], 'resolv-conf':
['/etc/resolv.conf'], 'image-gc-high-threshold': ['90'], 'image-gc-low-
threshold': ['80']}
```

2.10.1. ノードあたりの最大 Pod 数の設定



注記

OpenShift Container Platform の各バージョンでサポートされている最大の制限については、「[Cluster Limits](#)」のページを参照してください。

`/etc/origin/node/node-config.yaml` ファイルでは、`pods-per-core` および `max-pods` の 2 つのパラメーターがノードにスケジュールできる Pod の最大数を制御します。いずれのオプションも使用されている場合、2 つの内の小さい方の値でノードの Pod 数が制限されます。これらの値を超えると、以下の状況が発生します。

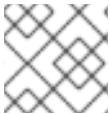
- OpenShift Container Platform と Docker の両方で CPU 使用率が増加する。
- Pod のスケジューリングの速度が遅くなる。
- メモリー不足のシナリオが生じる可能性がある (ノードのメモリー量によって異なる)。
- IP アドレスのプールを消費する。
- リソースのオーバーコミットが起こり、アプリケーションのパフォーマンスが低下する。

**注記**

Kubernetes では、単一コンテナを保持する Pod は実際には 2 つのコンテナを使用します。2 つ目のコンテナは実際のコンテナの起動前にネットワークを設定するために使用されます。そのため、10 の Pod を使用するシステムでは、実際には 20 のコンテナが実行されていることになります。

pods-per-core は、ノードのプロセッサコア数に基づいてノードが実行できる Pod 数を設定します。たとえば、4 プロセッサコアを搭載したノードで **pods-per-core** が **10** に設定される場合、このノードで許可される Pod の最大数は 40 になります。

```
kubeletArguments:
  pods-per-core:
    - "10"
```

**注記**

pods-per-core を 0 に設定すると、この制限が無効になります。

max-pods は、ノードのプロパティにかかわらず、ノードが実行できる Pod 数を固定値に設定します。「[Cluster Limits](#)」では **max-pods** のサポートされる最大の値について説明しています。

```
kubeletArguments:
  max-pods:
    - "250"
```

上記の例では、 **pods-per-core** のデフォルト値は **10** であり、 **max-pods** のデフォルト値は **250** です。これは、ノードにあるコア数が 25 以上でない限り、デフォルトでは **pods-per-core** が制限を設定することになります。

2.11. DOCKER ストレージの再設定

Docker イメージをダウンロードし、コンテナを実行して削除する際、Docker は常にマップされたディスク領域を解放する訳ではありません。結果として、一定の時間が経過するとノード上で領域不足が生じる可能性があり、これにより OpenShift Container Platform で新規 Pod を作成できなくなるか、または Pod の作成に時間がかかる可能性があります。

たとえば、以下は 6 分が経過しても **ContainerCreating** 状態にある Pod を示しており、イベントログは [FailedSync イベント](#) について示しています。

```
$ oc get pod
NAME                                READY    STATUS
RESTARTS   AGE
cakephp-mysql-persistent-1-build    0/1      ContainerCreating    0
6m
mysql-1-9767d                        0/1      ContainerCreating    0
2m
mysql-1-deploy                       0/1      ContainerCreating    0
6m

$ oc get events
LASTSEEN   FIRSTSEEN   COUNT   NAME                                KIND
SUBOBJECT                                     TYPE      REASON
```

SOURCE			MESSAGE	
6m	6m	1	cakephp-mysql-persistent-1-build	Pod
Normal	Scheduled		default-scheduler	
Successfully assigned cakephp-mysql-persistent-1-build to ip-172-31-71-195.us-east-2.compute.internal				
2m	5m	4	cakephp-mysql-persistent-1-build	Pod
Warning	FailedSync		kubelet, ip-172-31-71-195.us-east-2.compute.internal	Error syncing pod
2m	4m	4	cakephp-mysql-persistent-1-build	Pod
Normal	SandboxChanged		kubelet, ip-172-31-71-195.us-east-2.compute.internal	Pod sandbox changed, it will be killed and re-created.

この問題に対する 1 つの解決法として、Docker ストレージを再設定し、Docker で不要なアーティファクトを削除することができます。

Docker ストレージを再起動するノードで、以下を実行します。

1. 以下のコマンドを実行して、ノードをスケジューリング対象外としてマークします。

```
$ oc adm manage-node <node> --schedulable=false
```

2. 以下のコマンドを実行して Docker および **atomic-openshift-node** サービスをシャットダウンします。

```
$ systemctl stop docker atomic-openshift-node
```

3. 以下のコマンドを実行してローカルのボリュームディレクトリを削除します。

```
$ rm -rf /var/lib/origin/openshift.local.volumes
```

このコマンドは、ローカルイメージのキャッシュをクリアします。その結果、**ose-*** イメージを含むイメージが再度プルする必要があります。これにより、イメージストアは回復しますが、Pod の起動時間が遅くなる可能性があります。

4. **/var/lib/docker** ディレクトリを削除します。

```
$ rm -rf /var/lib/docker
```

5. 以下のコマンドを実行して Docker ストレージを再設定します。

```
$ docker-storage-setup --reset
```

6. 以下のコマンドを実行して Docker ストレージを再作成します。

```
$ docker-storage-setup
```

7. **/var/lib/docker** ディレクトリを再作成します。

```
$ mkdir /var/lib/docker
```

8. 以下のコマンドを実行して Docker および **atomic-openshift-node** サービスを再起動します。


```
$ systemctl start docker atomic-openshift-node
```

9. 以下のコマンドを実行してノードをスケジュール対象としてマークします。

```
$ oc adm manage-node <node> --schedulable=true
```

2.12. ノードトラフィックインターフェースの変更

デフォルトで DNS はすべてのノードトラフィックをルーティングします。ノードの登録時に、マスターはノード IP アドレスを DNS 設定から受信するため、DNS 経由でノードにアクセスすることが、ほとんどのデプロイメントにおいて最も柔軟な解決策となります。

お使いのデプロイメントでクラウドプロバイダーを使用している場合、ノードはクラウドプロバイダーから IP 情報を取得します。ただし **openshift-sdn** は、(設定されている場合は) nodeName や (nodeName が設定されていない場合は) システムホスト名の DNS ルックアップを含む、各種の方式を使用して IP の判別を試行します。

ノードトラフィックインターフェースには変更が必要になる場合があります。たとえば、以下の場合はこれに該当します。

- OpenShift Container Platform が、内部ホスト名がすべてのホストで設定されていない/解決可能でないクラウドプロバイダーにインストールされている。
- マスターから見たノードの IP とノード自体から見たノードの IP が同じではない。

openshift_set_node_ip Ansible 変数を設定すると、デフォルトのネットワークインターフェース以外のインターフェースでのノードトラフィックが強制的に実行されます。

ノードトラフィックインターフェースを変更するには、以下を実行します。

1. **openshift_set_node_ip** Ansible 変数を **true** に設定します。
2. **openshift_ip** を、設定するノードの IP アドレスに設定します。



注記

openshift_set_node_ip は本セクションで記載したケースの回避策として有効ですが、通常は実稼働環境には適していません。ノードが新規 IP アドレスを受信すると適切に機能しなくなるためです。

第3章 ユーザーの管理

3.1. 概要

ユーザーとは、OpenShift Container Platform API と対話するエンティティです。ユーザーは、アプリケーションを開発する開発者の場合もあれば、クラスターを管理する管理者の場合もあります。ユーザーは、グループのすべてのメンバーに適用されるパーミッションを設定するグループに割り当てることができます。たとえば、API アクセスをグループに付与して、そのグループのすべてのメンバーに API アクセスを付与することができます。

このトピックでは、[ユーザー](#)アカウントの管理について説明します。これには、OpenShift Container Platform での新規ユーザーアカウントの作成方法とそれらの削除方法が含まれます。

3.2. ユーザーの作成

ユーザーの作成プロセスは、設定される [アイデンティティプロバイダー](#)によって異なります。デフォルトで、OpenShift Container Platform は、すべてのユーザー名およびパスワードのアクセスを拒否する **DenyAll** アイデンティティプロバイダーを使用します。

以下のプロセスでは、新規ユーザーを作成してからロールをそのユーザーに追加します。

1. アイデンティティプロバイダーに応じたユーザーアカウントを作成します。これは、アイデンティティプロバイダー設定の一部として使用される **mappingmethod** によって異なります。詳細は、「[Mapping Identities to Users \(アイデンティティのユーザーへのマッピング\)](#)」セクションを参照してください。
2. 新規ユーザーに必要なロールを付与します。

```
# oc create clusterrolebinding <clusterrolebinding_name> /
--clusterrole=<role> --user=<user>
```

ここで、**--clusterrole** オプションは必要なクラスターロールになります。たとえば、新規ユーザーに対して、クラスター内のすべてに対するアクセスを付与する **cluster-admin** 権限を付与するには、以下を実行します。

```
# oc create clusterrolebinding registry-controller /
--clusterrole=cluster-admin --user=admin
```

ロールの説明および一覧については、[Architecture Guide](#) の [クラスターロールおよびローカルロール](#) についてのセクションを参照してください。

クラスター管理者は、[各ユーザーのアクセスレベルの管理](#)も実行できます。



注記

アイデンティティプロバイダーおよび定義されたグループ構造によっては、一部のロールがユーザーに自動的に付与される場合があります。詳細は、「[グループの LDAP との同期](#)」についてのセクションを参照してください。

3.3. ユーザーおよび ID リストの表示

OpenShift Container Platform のユーザー設定は、OpenShift Container Platform 内の複数の場所に保存されます。アイデンティティプロバイダーの種類を問わず、OpenShift Container Platform はロール

ベースのアクセス制御 (RBAC) 情報およびグループメンバーシップなどの詳細情報を内部に保存します。ユーザー情報を完全に削除するには、ユーザーアカウントに加えてこのデータも削除する必要があります。

OpenShift Container Platform では、2 つのオブジェクトタイプ (**user** および **identity**) に、アイデンティティプロバイダー外のユーザーデータが含まれます。

ユーザーの現在のリストを取得するには、以下を実行します。

```
$ oc get user
NAME          UID                                FULL NAME    IDENTITIES
demo          75e4b80c-dbf1-11e5-8dc6-0e81e52cc949
htpasswd_auth:demo
```

ID の現在のリストを取得するには、以下を実行します。

```
$ oc get identity
NAME          IDP NAME      IDP USER NAME  USER NAME  USER
UID
htpasswd_auth:demo  htpasswd_auth  demo           demo
75e4b80c-dbf1-11e5-8dc6-0e81e52cc949
```

2 つのオブジェクトタイプ間で一致する UID があることに注意してください。OpenShift Container Platform の使用を開始した後に認証プロバイダーの変更を試行する場合で重複するユーザー名がある場合、そのユーザー名は、ID リストに古い認証方式を参照するエントリーがあるために機能しなくなります。

3.4. グループの作成

ユーザーは OpenShift Container Platform に要求するエンティティーである一方で、ユーザーのセットで構成される 1 つの以上のグループに編成することもできます。グループは、許可ポリシーなどの場合のように数多くのユーザーを 1 度に管理する際や、パーミッションを複数のユーザーに 1 度に付与する場合などに役立ちます。

組織が LDAP を使用している場合、LDAP レコードの OpenShift Container Platform に対する同期を実行し、複数のグループを 1 つの場所で設定できるようにすることができます。この場合、ユーザーについての情報が LDAP サーバーにあることを仮定します。詳細は、「[Syncing groups with LDAP \(グループの LDAP との同期\)](#)」セクションを参照してください。LDAP を使用していない場合は、以下の手順を使用してグループを手動で作成できます。

新規グループを作成するには、以下を実行します。

```
# oc adm groups new <group_name> <user1> <user2>
```

たとえば、**west** グループを作成し、そのグループ内に **john** および **betty** ユーザーを置くには、以下を実行します。

```
# oc adm groups new west john betty
```

グループが作成されたことを確認し、グループに関連付けられたユーザーを一覧表示するには、以下を実行します。

```
# oc get groups
NAME      USERS
```

```
west      john, betty
```

次の手順は、* [ロールバインディングの管理](#)です。

3.5. ユーザーおよびグループラベルの管理

ラベルをユーザーまたはグループに追加するには、以下を実行します。

```
$ oc label user/<user_name> <label_name>
```

たとえばユーザー名が **theuser** で、ラベルが **level=gold** の場合には、以下のようになります。

```
$ oc label user/theuser level=gold
```

ラベルを削除するには、以下を実行します。

```
$ oc label user/<user_name> <label_name>-
```

ユーザーまたはグループのラベルを表示するには、以下を実行します。

```
$ oc describe user/<user_name>
```

3.6. ユーザーの削除

ユーザーを削除するには、以下を実行します。

1. ユーザーレコードを削除します。

```
$ oc delete user demo
user "demo" deleted
```

2. ユーザー ID を削除します。

ユーザーの ID は使用するアイデンティティプロバイダーに関連付けられます。**oc get user** でユーザーレコードからプロバイダー名を取得します。

この例では、アイデンティティプロバイダー名は **htpasswd_auth** です。コマンドは、以下のようになります。

```
# oc delete identity htpasswd_auth:demo
identity "htpasswd_auth:demo" deleted
```

この手順を省略すると、ユーザーは再度ログインできなくなります。

上記の手順の完了後は、ユーザーが再びログインすると、新規のアカウントが OpenShift Container Platform に作成されます。

ユーザーの再ログインを防ごうとする場合 (たとえば、ある社員が会社を退職し、そのアカウントを永久に削除する必要がある場合)、そのユーザーを、設定されたアイデンティティプロバイダーの認証バックエンド (**htpasswd**、**kerberos** その他) から削除することもできます。

たとえば **htpasswd** を使用している場合、該当のユーザー名とパスワードで OpenShift Container Platform に設定された **htpasswd** ファイルのエントリを削除します。

Lightweight Directory Access Protocol (LDAP) または Red Hat Identity Management (IdM) などの外部 ID 管理については、ユーザー管理ツールを使用してユーザーエントリーを削除します。

第4章 プロジェクトの管理

4.1. 概要

OpenShift Container Platform では、プロジェクトは関連オブジェクトを分類し、分離するために使用されます。管理者は、開発者に特定プロジェクトへのアクセスを付与し、開発者の独自プロジェクトの作成を許可したり、個別プロジェクト内の管理者権限を付与したりできます。

4.2. プロジェクトのセルフプロビジョニング

開発者の独自プロジェクトの作成を許可することができます。テンプレートに基づいてプロジェクトをプロビジョニングするエンドポイントがあります。web コンソールおよび `oc new-project` コマンドは、開発者による新規プロジェクトの作成時にこのエンドポイントを使用します。

4.2.1. 新規プロジェクトのテンプレートの変更

API サーバーは、`master-config.yaml` ファイルの `projectRequestTemplate` パラメーターで識別されるテンプレートに基づいてプロジェクトを自動的にプロビジョニングします。パラメーターが定義されない場合、API サーバーは要求される名前でプロジェクトを作成するデフォルトテンプレートを作成し、要求するユーザーをプロジェクトの「admin (管理者)」ロールに割り当てます。

独自のカスタムプロジェクトテンプレートを作成するには、以下を実行します。

1. 現在のデフォルトプロジェクトテンプレートを使って開始します。

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

2. オブジェクトを追加するか、または既存オブジェクトを変更することにより、テキストエディターで `template.yaml` ファイルを変更します。

3. テンプレートを読み込みます。

```
$ oc create -f template.yaml -n default
```

4. 読み込まれたテンプレートを参照するよう `master-config.yaml` ファイルを変更します。

```
...
projectConfig:
  projectRequestTemplate: "default/project-request"
...
```

プロジェクト要求が送信されると、API はテンプレートで以下のパラメーターを置き換えます。

パラメーター	説明
<code>PROJECT_NAME</code>	プロジェクトの名前。必須。
<code>PROJECT_DISPLAYNAME</code>	プロジェクトの表示名。空にできます。
<code>PROJECT_DESCRIPTION</code>	プロジェクトの説明。空にできます。

パラメーター	説明
PROJECT_ADMIN_USER	管理ユーザーのユーザー名。
PROJECT_REQUESTING_USER	要求するユーザーのユーザー名。

API へのアクセスは、**self-provisioner** ロールと **self-provisioners** のクラスターのロールバインディングで開発者に付与されます。デフォルトで、このロールはすべての認証された開発者が利用できます。

4.2.2. セルフプロビジョニングの無効化

self-provisioners クラスターロールを認証されたユーザーグループから削除すると、新規プロジェクトのセルフプロビジョニングのパーミッションが拒否されます。

```
$ oc adm policy remove-cluster-role-from-group self-provisioner
system:authenticated system:authenticated:oauth
```

セルフプロビジョニングを無効にする場合、**projectRequestMessage** パラメーターを **master-config.yaml** ファイルに設定し、開発者に対して新規プロジェクトの要求方法を指示します。このパラメーターは、開発者のプロジェクトのセルフプロビジョニング試行時に web コンソールやコマンドラインに表示される文字列です。以下は例になります。

```
Contact your system administrator at projectname@example.com to request a
project.
```

または、以下のようになります。

```
To request a new project, fill out the project request form located at
https://internal.example.com/openshift-project-request.
```

サンプル YAML ファイル

```
...
projectConfig:
  ProjectRequestMessage: "message"
...
```

4.3. ノードセクターの使用

ノードセクターは、Pod の配置を制御するためにラベルが付けられたノードと併用されます。



注記

ラベルは、**通常インストール (Advanced installation)** の実行時に割り当てるか、または**インストール後にノードに追加**することができます。

4.3.1. クラスター全体でのデフォルトノードセクターの設定

クラスター管理者は、クラスター全体でのノードセクターを使用して Pod の配置を特定ノードに制限することができます。

`/etc/origin/master/master-config.yaml` でマスター設定ファイルを編集し、デフォルトノードセクターの値を追加します。これは、指定された **nodeSelector** 値なしにすべてのプロジェクトで作成された Pod に適用されます。

```
...
projectConfig:
  defaultNodeSelector: "type=user-node,region=east"
...
```

変更を有効にするために OpenShift サービスを再起動します。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

4.3.2. プロジェクト全体でのノードセクターの設定

ノードセクターを使って個々のプロジェクトを作成するには、プロジェクトの作成時に **--node-selector** オプションを使用します。たとえば、複数のリージョンを含む OpenShift Container Platform トポロジーがある場合、ノードセクターを使用して、特定リージョンのノードにのみ Pod をデプロイするよう特定の OpenShift Container Platform プロジェクトを制限することができます。

以下では、**myproject** という名前の新規プロジェクトを作成し、Pod を **user-node** および **east** のラベルが付けられたノードにデプロイするように指定します。

```
$ oc adm new-project myproject \
  --node-selector='type=user-node,region=east'
```

いったんこのコマンドが実行されると、これが指定プロジェクト内にあるすべての Pod に対して管理者が設定するノードセクターになります。



注記

new-project サブコマンドはクラスター管理者および開発者コマンドの **oc adm** と **oc** の両方で利用できますが、**oc adm** コマンドのみがノードセクターを使った新規プロジェクトの作成に利用できます。**new-project** サブコマンドは、プロジェクトのセルフプロビジョニング時にプロジェクト開発者が利用することはできません。

oc adm new-project コマンドを使用すると、**annotation** セクションがプロジェクトに追加されます。プロジェクトを編集し、デフォルトを上書きするように **openshift.io/node-selector** 値を編集できます。

```
...
metadata:
  annotations:
    openshift.io/node-selector: type=user-node,region=east
...
```

また、以下のコマンドを使用して既存プロジェクトの namespace のデフォルト値を上書きできます。


```
# oc patch namespace myproject -p \
  '{"metadata":{"annotations":{"openshift.io/node-selector":"region=infra"}}}'
```

openshift.io/node-selector が空の文字列 (**oc adm new-project --node-selector=""**) に設定される場合、プロジェクトには、クラスター全体のデフォルトが設定されている場合でも管理者設定のノードセクターはありません。これは、クラスター管理者はデフォルトを設定して開発者のプロジェクトをノードのサブセットに制限したり、インフラストラクチャーまたは他のプロジェクトでクラスター全体をスケジュールしたりできることを意味します。

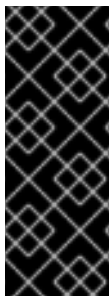
4.3.3. 開発者が指定するノードセクター

OpenShift Container Platform 開発者は、追加でノードを制限する必要がある場合に **Pod 設定でのノードセクターの設定**を行うことができます。これはプロジェクトノードセクターに追加されるものであり、ノードセクターの値を持つすべてのプロジェクトについて依然としてノードセクターの値を指定できることになります。

たとえば、プロジェクトが上記のアノテーションで作成 (**openshift.io/node-selector: type=user-node, region=east**) されており、開発者が別のノードセクターをそのプロジェクトの Pod に設定する場合 (例: **clearance=classified**)、Pod はこれらの 3 つのラベル (**type=user-node**、**region=east**、および **clearance=classified**) を持つノードにのみスケジュールされます。**region=west** が Pod に設定されている場合、Pod はラベル **region=east** および **region=west** を持つノードを要求しても成功しません。ラベルは 1 つの値にのみ設定できるため、Pod はスケジュールされません。

4.4. ユーザーあたりのセルフプロビジョニングされたプロジェクト数の制限

指定されるユーザーが要求するセルフプロビジョニングされたプロジェクトの数は、**ProjectRequestLimit** [受付制御プラグイン](#)で制限できます。



重要

プロジェクトの要求テンプレートが、「[新規プロジェクトのテンプレートの変更](#)」で説明されるプロセスを使用して OpenShift Container Platform 3.1 (またはそれ以前のバージョン) で作成される場合、生成されるテンプレートには、**ProjectRequestLimitConfig** に使用されるアノテーション **openshift.io/requester: \${PROJECT_REQUESTING_USER}** が含まれません。アノテーションは追加する必要があります。

ユーザーの制限を指定するには、設定をマスター設定ファイル (**/etc/origin/master/master-config.yaml**) 内のプラグインに指定する必要があります。プラグインの設定は、ユーザーラベルのセクターの一覧および関連付けられるプロジェクト要求の最大数を取ります。

セクターは順番に評価されます。現在のユーザーに一致する最初のセクターは、プロジェクトの最大数を判別するために使用されます。セクターが指定されていない場合、制限はすべてのユーザーに適用されます。プロジェクトの最大数が指定されていない場合、無制限のプロジェクトが特定のセクターに対して許可されます。

以下の設定は、ユーザーあたりのグローバル制限を 2 プロジェクトに設定し、ラベル **level=advanced** を持つユーザーに対して 10 プロジェクト、ラベル **level=admin** を持つユーザーに対して無制限のプロジェクトを許可します。

```
admissionConfig:
```

```

pluginConfig:
  ProjectRequestLimit:
    configuration:
      apiVersion: v1
      kind: ProjectRequestLimitConfig
      limits:
        - selector:
            level: admin ❶
        - selector:
            level: advanced ❷
            maxProjects: 10
        - maxProjects: 2 ❸

```

- ❶ セレクター **level=admin** の場合、**maxProjects** は指定されません。これは、このラベルを持つユーザーにはプロジェクト要求の最大数が設定されないことを意味します。
- ❷ セレクター **level=advanced** の場合、最大数の 10 プロジェクトが許可されます。
- ❸ 3 目のエントリーにはセレクターが指定されていません。これは、セレクターが直前の 2 つのルールを満たさないユーザーに適用されることを意味します。ルールは順番に評価されるため、このルールは最後に指定する必要があります。



注記

「[ユーザーおよびグループラベルの管理](#)」では、ユーザーおよびグループのラベルを追加し、削除し、表示する方法について詳述しています。

変更を加えた後にそれらの変更を有効にするには、OpenShift Container Platform を再起動します。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

第5章 POD の管理

5.1. 概要

このトピックでは、Pod を 1 回実行する場合の期間や使用可能な帯域幅を含む Pod の管理について説明します。

5.2. 1 回実行 (RUN-ONCE) POD 期間の制限

OpenShift Container Platform は 1 回実行 (run-once) Pod を使用して Pod のデプロイやビルドの実行などのタスクを実行します。1 回実行 (run-once) Pod は、**RestartPolicy** が **Never** または **OnFailure** の Pod です。

クラスター管理者は **RunOnceDuration** の受付制御プラグインを使用し、1 回実行 (run-once) Pod の有効期間の制限を強制的に実行できます。期限が切れると、クラスターはそれらの Pod をアクティブに終了しようとします。このような制限を設ける主な理由は、ビルドなどのタスクが長い時間にわたって実行されることを防ぐことにあります。

5.2.1. RunOnceDuration プラグインの設定

このプラグインの設定には、1 回実行 (run-once) Pod のデフォルト有効期限を含める必要があります。この期限はグローバルに実施されますが、プロジェクト別の期限によって置き換えられることがあります。

```
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      RunOnceDuration:
        configuration:
          apiVersion: v1
          kind: RunOnceDurationConfig
          activeDeadlineSecondsOverride: 3600 ❶
```

❶ 1 回実行 (run-once) Pod のグローバルのデフォルト値 (秒単位) を指定します。

5.2.2. プロジェクト別のカスタム期間の指定

1 回実行 (run-once) Pod のグローバルな最長期間を設定することに加え、管理者はアノテーション (**openshift.io/active-deadline-seconds-override**) を特定プロジェクトに追加し、グローバルのデフォルト値を上書きすることができます。

```
apiVersion: v1
kind: Project
metadata:
  annotations:
    openshift.io/active-deadline-seconds-override: "1000" ❶
```

❶ 1 回実行 (run-once) Pod のデフォルト有効期限 (秒単位) を 1000 秒に上書きします。上書きに使用する値は、文字列形式で指定される必要があります。

5.2.2.1. Egress ルーター Pod のデプロイ

例5.1 Egress ルーターの Pod 定義のサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  containers:
  - name: egress-router
    image: openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
      - name: EGRESS_SOURCE ❶
        value: 192.168.12.99
      - name: EGRESS_GATEWAY ❷
        value: 192.168.12.1
      - name: EGRESS_DESTINATION ❸
        value: 203.0.113.25
  nodeSelector:
    site: springfield-1 ❹
```

- ❶ この Pod で使用するためにクラスター管理者が予約するノードサブセットの IP アドレス。
- ❷ ノード自体で使用するデフォルトゲートウェイと同じ値。
- ❸ Pod の接続は 203.0.113.25 にリダイレクトされます。ソース IP アドレスは 192.168.12.99 です。
- ❹ Pod はラベルサイトが **springfield-1** のノードにのみデプロイされます。

pod.network.openshift.io/assign-macvlan annotation はプライマリーネットワークインターフェースに Macvlan ネットワークインターフェースを作成してから、それを Pod のネットワーク namespace に移行し、**egress-router** コンテナを起動します。



注記

"true" の周りの引用符をそのまま残します。これらを省略するとエラーが生じます。

Pod には **openshift3/ose-egress-router** イメージを使用する単一コンテナが含まれ、そのコンテナは特権モードで実行されるので、Macvlan インターフェースを設定したり、**iptables** ルールをセットアップしたりできます。

環境変数は **egress-router** イメージに対し、使用するアドレスを指示します。これは、**EGRESS_SOURCE** を IP アドレスとして、また **EGRESS_GATEWAY** をゲートウェイとして使用するよう Macvlan を設定します。

NAT ルールが設定され、Pod のクラスター IP アドレスの TCP または UDP ポートへの接続が **EGRESS_DESTINATION** の同じポートにリダイレクトされるようにします。

クラスター内の一部のノードのみが指定されたソース IP アドレスを要求でき、指定されたゲートウェイを使用できる場合、受け入れ可能なノードを示す **nodeName** または **nodeSelector** を指定することができます。

5.2.2.2. Egress ルーターサービスのデプロイ

通常、egress ルーターを参照するサービスを作成する必要がある場合があります (ただし、これは必ずしも必須ではありません)。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: http
      port: 80
    - name: https
      port: 443
  type: ClusterIP
  selector:
    name: egress-1
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにリダイレクトされます。

5.2.3. Egress ファイアウォールでの Pod アクセスの制限

OpenShift Container Platform クラスター管理者は egress ポリシーを使用して、一部またはすべての Pod がクラスターからアクセスできる外部アドレスを制限できます。これにより、以下が可能になります。

- Pod の対話を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
または
- Pod の対話をパブリックインターネットに制限し、(クラスター外の) 内部ホストへの接続を開始できないようにする。
または
- Pod が接続する理由のない指定された内部サブネット/ホストに到達できないようにする。

プロジェクトは複数の異なる egress ポリシーで設定でき、たとえば指定された IP 範囲への **<project A>** のアクセスを許可する一方で、同じアクセスを **<project B>** に対して拒否することができます。

注意

egress ポリシーで Pod のアクセスを制限するには、**ovs-multitenant プラグイン** を有効にする必要があります。

プロジェクト管理者は、**EgressNetworkPolicy** オブジェクトを作成することも、プロジェクトで作成するオブジェクトを編集することもできません。また、**EgressNetworkPolicy** の作成に関連して他のいくつかの制限があります。

1. デフォルトプロジェクト (および **oc adm pod-network make-projects-global** でグローバルにされたその他のプロジェクト) には egress ポリシーを設定することができません。
2. (**oc adm pod-network join-projects** を使用して) 2 つのプロジェクトをマージする場合、マージしたプロジェクトのいずれでも egress ポリシーを使用することはできません。
3. いずれのプロジェクトも複数の egress ポリシーオブジェクトを持つことができません。

上記の制限のいずれかに違反すると、プロジェクトの egress ポリシーに障害が発生し、すべての外部ネットワークトラフィックがドロップされる可能性があります。

5.2.3.1. Pod アクセス制限の設定

Pod アクセス制限を設定するには、**oc** コマンドまたは REST API を使用する必要があります。**oc [create|replace|delete]** を使用すると、**EgressNetworkPolicy** オブジェクトを操作できます。**api/swagger-spec/oapi-v1.json** ファイルには、オブジェクトの機能方法についての API レベルの詳細情報が含まれます。

Pod のアクセス制限を設定するには、以下を実行します。

1. 対象とするプロジェクトに移動します。
2. Pod の制限ポリシーについての JSON ファイルを作成します。

```
# oc create -f <policy>.json
```

3. ポリシーの詳細情報を使って JSON ファイルを設定します。以下は例になります。

```
{
  "kind": "EgressNetworkPolicy",
  "apiVersion": "v1",
  "metadata": {
    "name": "default"
  },
  "spec": {
    "egress": [
      {
        "type": "Allow",
        "to": {
          "cidrSelector": "1.2.3.0/24"
        }
      },
      {
        "type": "Allow",
        "to": {
          "dnsName": "www.foo.com"
        }
      },
      {
        "type": "Deny",
        "to": {
          "cidrSelector": "0.0.0.0/0"
        }
      }
    ]
  }
}
```

```

    }
  }
}

```

上記のサンプルがプロジェクトに追加されると、IP 範囲 **1.2.3.0/24** およびドメイン名 **www.foo.com** へのトラフィックは許可されますが、その他すべての外部 IP アドレスへのアクセスは拒否されます (ポリシーが **外部** トラフィックにのみ適用されるので他の Pod へのトラフィックは影響を受けません)。

EgressNetworkPolicy のルールは順番にチェックされ、一致する最初のルールが実施されます。上記の例の 3 つの例が維持される場合、**0.0.0.0/0** ルールが最初にチェックされ、すべてのトラフィックに一致し、それらすべてを拒否するため、**1.2.3.0/24** および **www.foo.com** へのトラフィックは許可されません。

ドメイン名の更新は 30 秒以内に反映されます。上記の例で **www.foo.com** は **10.11.12.13** に解決されますが、**20.21.22.23** に変更されたとします。OpenShift Container Platform では最長 30 秒後にこれらの DNS 更新に対応します。

5.3. POD で利用可能な帯域幅の制限

QoS (Quality-of-Service) トラフィックシェーピングを Pod に適用し、その利用可能な帯域幅を効果的に制限することができます。(Pod からの) Egress トラフィックは、設定したレートを超えるパケットを単純にドロップするポリシングによって処理されます。(Pod への) Ingress トラフィックは、データを効果的に処理できるようにシェーピングでパケットをキューに入れて処理されます。Pod に設定する制限は、他の Pod の帯域幅には影響を与えません。

Pod の帯域幅を制限するには、以下を実行します。

1. オブジェクト定義 JSON ファイルを作成し、**kubernetes.io/ingress-bandwidth** および **kubernetes.io/egress-bandwidth** アノテーションを使用してデータトラフィックの速度を指定します。たとえば、Pod の egress および ingress の両方の帯域幅を 10M/s に制限するには、以下を実行します。

例5.2 制限が設定された Pod オブジェクト定義

```

{
  "kind": "Pod",
  "spec": {
    "containers": [
      {
        "image": "nginx",
        "name": "nginx"
      }
    ]
  },
  "apiVersion": "v1",
  "metadata": {
    "name": "iperf-slow",
    "annotations": {
      "kubernetes.io/ingress-bandwidth": "10M",
      "kubernetes.io/egress-bandwidth": "10M"
    }
  }
}

```

2. オブジェクト定義を使用して Pod を作成します。

```
oc create -f <file_or_dir_path>
```

5.4. POD の DISRUPTION BUDGET (停止状態の予算) の設定

pod disruption budget は [Kubernetes API](#) の一部であり、他の [オブジェクトタイプ](#) のように **oc** コマンドで管理できます。この設定により、メンテナンスのためのノードのドレインなど、操作時に Pod への安全面の各種の制約を指定できます。



注記

OpenShift Container Platform 3.6 より、Pod の disruption budget は完全にサポートされています。

PodDisruptionBudget は、同時に起動している必要のあるレプリカの最小数またはパーセンテージを指定する API オブジェクトです。これらをプロジェクトに設定することは、ノードのメンテナンス (クラスターのスケールダウンまたはクラスターのアップグレードなどの実行) 時に役立ち、この設定は (ノードの障害時ではなく) 自発的なエビクションの場合にのみ許可されます。

PodDisruptionBudget オブジェクトの設定は、以下の主要な部分で構成されています。

- 一連の Pod に対するラベルのクエリ機能であるラベルセクター。
- 同期に利用可能にする必要のある Pod の最小数を指定する可用性レベル。

以下は、**PodDisruptionBudget** リソースのサンプルです。

```
apiVersion: policy/v1beta1 ❶
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  selector: ❷
    matchLabels:
      foo: bar
  minAvailable: 2 ❸
```

- ❶ **PodDisruptionBudget** は **policy/v1beta1** API グループの一部です。
- ❷ 一連のリソースに対するラベルのクエリ。**matchLabels** と **matchExpressions** の結果は論理的に結合されます。
- ❸ 同時に利用可能である必要のある Pod の最小数。これには、整数またはパーセンテージ (例: **20%**) を指定する文字列を使用できます。

上記のオブジェクト定義で YAML ファイルを作成した場合、これを以下のようにプロジェクトに追加することができます。

```
$ oc create -f </path/to/file> -n <project_name>
```


以下を実行して、Pod の disruption budget をすべてのプロジェクトで確認することができます。

```
$ oc get poddisruptionbudget --all-namespaces
```

NAMESPACE	NAME	MIN-AVAILABLE	SELECTOR
another-project	another-pdb	4	bar=foo
test-project	my-pdb	2	foo=bar

PodDisruptionBudget は、最低でも **minAvailable** の Pod がシステムで実行されている場合は正常であるとみなされます。この制限を超えるすべての Pod は**エビクション**の対象となります。

5.5. POD の PRESET (プリセット) を使用した情報の POD への挿入

Pod の Preset は、ユーザーが指定する情報を Pod の作成時に Pod に挿入するオブジェクトです。

重要

Pod の Preset 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。

挿入可能な Pod の Preset オブジェクトを使用します。

- シークレットオブジェクト
- ConfigMap オブジェクト
- ストレージボリューム
- コンテナボリュームのマウント
- 環境変数

すべての情報を Pod に追加したい場合、開発者は Pod ラベルが PodPreset のラベルセクターに一致することのみを確認する必要があります。Pod のラベルは、一致するラベルセクターを持つ 1 つ以上の Pod Preset オブジェクトに Pod を関連付けます。

Pod の Preset を使用すると、開発者は Pod が使用するサービスについての詳細を知らなくても Pod をプロビジョニングできます。管理者は、開発者による Pod のデプロイを阻むことなくサービスの設定項目を開発者に対して非表示にできます。たとえば、管理者はシークレットでデータベースの名前、ユーザー名、およびパスワードを提供し、環境変数でデータベースポートを提供する Pod の Preset を作成できます。Pod の開発者はすべての情報を Pod に組み込むために使用するラベルのみを把握している必要があります。さらに開発者も Pod の Preset を作成して同じタスクを実行することができます。たとえば開発者は、環境変数を複数 Pod に自動的に挿入する Preset を作成できます。



注記

Pod の Preset 機能は、[サービスカタログ](#)がインストールされている場合にのみ利用できます。

Pod 仕様で `podpreset.admission.kubernetes.io/exclude: "true"` パラメーターを使用し、特定の Pod が挿入されないようにすることができます。[Pod 仕様のサンプル](#)を参照してください。

詳細は、「[Pod の Preset \(プリセット\) を使用した情報の Pod への挿入](#)」を参照してください。

第6章 ネットワークの管理

6.1. 概要

このトピックでは、プロジェクトの分離および発信トラフィックの制御を含む、全般的な[クラスターネットワーク](#)の管理について説明します。

Pod ごとの帯域幅の制限などの Pod レベルのネットワーク機能については、[Pod の管理](#)で説明されています。

6.2. POD ネットワークの管理

クラスターが [ovs-multitenant SDN プラグイン](#) を使用するように設定されている場合、管理者 CLI を使用してプロジェクトの別個の Pod オーバーレイネットワークを管理することができます。必要な場合は、プラグイン設定手順について「[SDN の設定](#)」セクションを参照してください。

6.2.1. プロジェクトネットワークへの参加

プロジェクトを既存のプロジェクトネットワークに参加させるには、以下を実行します。

```
$ oc adm pod-network join-projects --to=<project1> <project2> <project3>
```

上記の例で、**<project2>** および **<project3>** のすべての Pod およびサービスから、**<project1>** のすべての Pod およびサービスへのアクセスが可能となり、その逆の場合も可能になります。サービスは、IP または完全修飾 DNS 名 (**<service>.<pod_namespace>.svc.cluster.local**) のいずれかでアクセスできます。たとえば、プロジェクト **myproject** の **db** という名前のサービスにアクセスするには、**db.myproject.svc.cluster.local** を使用します。

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用することもできます。

6.3. プロジェクトネットワークの分離

プロジェクトネットワークをクラスターから分離したり、その逆を実行するには、以下を実行します。

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

上記の例では、**<project1>** および **<project2>** のすべての Pod およびサービスは、クラスター内のグローバル以外のプロジェクトの Pod およびサービスにアクセスできず、その逆も実行できません。

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用することもできます。

6.3.1. プロジェクトネットワークのグローバル化

プロジェクトからクラスター内のすべての Pod およびサービスにアクセスできるようにするか、その逆を可能にするには、以下を実行します。

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

上記の例では、**<project1>** および **<project2>** のすべての Pod およびサービスはクラスター内のすべての Pod およびサービスにアクセスでき、その逆の場合も可能になります。

または、特定のプロジェクト名を指定する代わりに `--selector=<project_selector>` オプションを使用することもできます。

6.4. ルートおよび INGRESS オブジェクトにおけるホスト名の競合防止の無効化

OpenShift Container Platform では、ルートおよび ingress オブジェクトのホスト名の競合防止はデフォルトで有効にされています。これは、**cluster-admin** ロールのないユーザーは、作成時にのみルーターまたは ingress オブジェクトのホスト名を設定でき、その後は変更できなくなることを意味しています。ただし、ルートおよび ingress オブジェクトのこの制限は、一部またはすべてのユーザーに対して緩和することができます。



警告

OpenShift Container Platform はオブジェクト作成のタイムスタンプを使用して特定のホスト名の最も古いルートや ingress オブジェクトを判別するため、ルートまたは ingress オブジェクトは、古いルートがそのホスト名を変更したり、ingress オブジェクトが導入される場合に新規ルートのホスト名をハイジャックする可能性があります。

OpenShift Container Platform クラスター管理者は、作成後もルートのホスト名を編集できます。また、特定のユーザーがこれを実行できるようにロールを作成することもできます。

```
$ oc create clusterrole route-editor --verb=update --
resource=routes.route.openshift.io/custom-host
```

次に、新規ロールをユーザーにバインドできます。

```
$ oc adm policy add-cluster-role-to-user route-editor user
```

ingress オブジェクトのホスト名の競合防止を無効にすることもできます。これを実行することで、**cluster-admin** ロールを持たないユーザーが作成後も ingress オブジェクトのホスト名を編集できるようになります。これは、ingress オブジェクトのホスト名の編集を許可する場合などに Kubernetes の動作に依存する OpenShift Container Platform のインストールで役に立ちます。

1. 以下を **master.yaml** ファイルに追加します。

```
admissionConfig:
  pluginConfig:
    openshift.io/IngressAdmission:
      configuration:
        apiVersion: v1
        allowHostnameChanges: true
      kind: IngressAdmissionConfig
    location: ""
```

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。

```
$ systemctl restart atomic-openshift-master-api atomic-openshift-
```

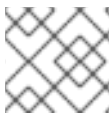
6.5. EGRESS トラフィックの制御

クラスター管理者は、ホストレベルで数多くの静的 IP アドレスを特定ノードに割り当てることができます。アプリケーション開発者がそれぞれのアプリケーションサービスに専用 IP アドレスを必要とする場合、ファイアウォールアクセスを要求するプロセスでこのアドレスを要求することができます。その後、開発者はデプロイメント設定の **nodeSelector** を使用して、開発者のプロジェクトから egress ルーターをデプロイし、静的 IP アドレスが事前に割り当てられたホストに Pod が到達することを確認できます。

egress Pod のデプロイメントでは、宛先に到達するために必要なソース IP のいずれか、保護されるサービスの宛先 IP、およびゲートウェイ IP を宣言します。Pod のデプロイ後は、[サービスを作成](#)して、egress ルーター Pod にアクセスし、そのソース IP を企業ファイアウォールに追加できます。その後、開発者はプロジェクトで作成された egress ルーターサービスへのアクセス情報を取得します (例: **service.project.cluster.domainname.com**)。

開発者が外部の firewalled サービスにアクセスする必要がある場合、実際の保護されたサービス URL ではなくアプリケーション (例: JDBC 接続情報) で、egress ルーター Pod のサービス (**service.project.cluster.domainname.com**) に対して呼び出し実行することができます。

さらに、静的 IP アドレスをプロジェクトに割り当て、指定されたプロジェクトからの発信外部接続すべてに認識可能な起点を設定できます。これは、トラフィックと特定の宛先に送信するために使用されるデフォルトの egress ルーターとは異なります。詳細は、「[外部プロジェクトトラフィックの固定 IP の有効化](#)」セクションを参照してください。



注記

egress ルーターは OpenShift Dedicated では利用できません。

OpenShift Container Platform クラスター管理者は、以下を使用して egress トラフィックを制御できます。

ファイアウォール

egress ファイアウォールを使用すると、受け入れ可能な発信トラフィックポリシーを実施し、特定のエンドポイントまたは IP 範囲 (サブネット) のみを動的エンドポイント (OpenShift Container Platform 内の Pod) が通信できる受け入れ可能なターゲットとすることができます。

ルーター

egress ルーターを使用することで、識別可能なサービスを作成し、トラフィックを特定の宛先に送信できます。これにより、それらの外部の宛先はトラフィックを既知のソースから送られるものとして処理します。これにより namespace の特定の Pod のみがトラフィックをデータベースにプロキシ送信するサービス (egress ルーター) と通信できるよう外部データベースが保護されるため、セキュリティ対策として役立ちます。

iptables

上記の OpenShift Container Platform 内のソリューションのほかにも、発信トラフィックに適用される iptables ルールを作成することができます。これらのルールは、egress ファイアウォールよりも多くのオプションを許可しますが、特定のプロジェクトに制限することはできません。

6.5.1. 外部リソースへのアクセスを制限するための Egress ファイアウォールの使用

OpenShift Container Platform クラスター管理者は egress ファイアウォールを使用して、一部またはすべての Pod がクラスター内からアクセスできる外部アドレスを制限できます。これにより、以下が可能になります。

- Pod の対話を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
または
- Pod の対話をパブリックインターネットに制限し、(クラスター外の) 内部ホストへの接続を開始できないようにする。
または
- Pod が接続する理由のない指定された内部サブネット/ホストに到達できないようにする。

プロジェクトを複数の異なる egress ポリシーを持つように設定できます。たとえば、**<project A>** の指定した IP 範囲へのアクセスを許可する一方で、同じアクセスを **<project B>** に対して拒否することができます。または、アプリケーション開発者の (Python) pip mirror からの更新を制限したり、更新を必要なソースからの更新のみに強制的に制限したりすることができます。

注意

Pod アクセスを egress ポリシーで制限するには、**ovs-multitenant** または **ovs-networkpolicy** プラグインを有効にする必要があります。

プロジェクト管理者は、**EgressNetworkPolicy** オブジェクトを作成することも、プロジェクトで作成するオブジェクトを編集することもできません。また、**EgressNetworkPolicy** の作成に関連して他のいくつかの制限があります。

- デフォルトプロジェクト (および **oc adm pod-network make-projects-global** でグローバルにされたその他のプロジェクト) には egress ポリシーを設定することができません。
- (**oc adm pod-network join-projects** を使用して) 2 つのプロジェクトをマージする場合、マージしたプロジェクトのいずれでも egress ポリシーを使用することはできません。
- いずれのプロジェクトも複数の egress ポリシーオブジェクトを持つことができません。

上記の制限のいずれかに違反すると、プロジェクトの egress ポリシーに障害が発生し、すべての外部ネットワークトラフィックがドロップされる可能性があります。

oc コマンドまたは REST API を使用して egress ポリシーを設定します。**oc [create|replace|delete]** を使用して **EgressNetworkPolicy** オブジェクトを操作できます。**api/swagger-spec/oapi-v1.json** ファイルには、オブジェクトを実際に機能させる方法についての API レベルの詳細情報が含まれます。

egress ポリシーを設定するには、以下を実行します。

1. 対象とするプロジェクトに移動します。
2. 必要なポリシー詳細情報で JSON ファイルを作成します。以下は例になります。

```
{
  "kind": "EgressNetworkPolicy",
  "apiVersion": "v1",
  "metadata": {
    "name": "default"
  },
}
```



```

"spec": {
  "egress": [
    {
      "type": "Allow",
      "to": {
        "cidrSelector": "1.2.3.0/24"
      }
    },
    {
      "type": "Allow",
      "to": {
        "dnsName": "www.foo.com"
      }
    },
    {
      "type": "Deny",
      "to": {
        "cidrSelector": "0.0.0.0/0"
      }
    }
  ]
}

```

上記のサンプルがプロジェクトに追加されると、IP 範囲 **1.2.3.0/24** およびドメイン名 **www.foo.com** へのトラフィックが許可されますが、その他のすべての外部 IP アドレスへのアクセスは拒否されます。このポリシーは**外部**トラフィックにのみ適用されるため、その他すべての Pod へのトラフィックは影響を受けません。

EgressNetworkPolicy のルールは順番にチェックされ、一致する最初のルールが実施されます。上記の例の 3 つの例が維持される場合、**0.0.0.0/0** ルールが最初にチェックされ、すべてのトラフィックに一致し、それらすべてを拒否するため、**1.2.3.0/24** および **www.foo.com** へのトラフィックは許可されません。

ドメイン名の更新は、ローカルの非権威サーバーのドメインの TTL (time to live) 値か、または TTL をフェッチできない場合は 30 分の値に基づいてポーリングされます。Pod は必要な場合には、同じローカルの非権威サーバーのドメインを解決します。そうでない場合には、egress ネットワークポリシーコントローラーと Pod で認識されるドメインの IP アドレスが異なり、egress ネットワークが予想通りに実施されない場合があります。上記の例で **www.foo.com** が **10.11.12.13** に解決され、DNS TTL が 1 分に設定されていますが、後に **20.21.22.23** に変更されたとします。OpenShift Container Platform は最長 1 分後にこれらの変更に対応します。

注記

egress ファイアウォールは、DNS 解決用に Pod が置かれるノードの外部インターフェースに Pod が常にアクセスできるようにします。DNS 解決がローカルノード上のいずれかによって処理されない場合は、Pod でドメイン名を使用している場合には DNS サーバーの IP アドレスへのアクセスを許可する egress ファイアウォールを追加する必要があります。**デフォルトインストーラー**はローカル dnsmasq をセットアップするため、このセットアップを使用する場合は、ルールを追加する必要はありません。

1. JSON ファイルを使用して EgressNetworkPolicy オブジェクトを作成します。

```
$ oc create -f <policy>.json
```

注意

[ルート](#)を作成してサービスを公開すると、**EgressNetworkPolicy** は無視されます。Egress ネットワークポリシーサービスのエンドポイントのフィルターは、ノード **kubeproxy** で実行されます。ルーターが使用される場合は、**kubeproxy** はバイパスされ、egress ネットワークポリシーの実施は適用されません。管理者は、ルートを作成するためのアクセスを制限してこのバイパスを防ぐことができます。

6.5.2. 外部リソースから Pod トラフィックを認識可能にするための Egress ルーターの使用

OpenShift Container Platform egress ルーターは、他の用途で使用されていないプライベートソース IP アドレスを使用して、指定されたリモートサーバーにトラフィックをリダイレクトするサービスを実行します。このサービスにより、Pod はホワイトリスト IP アドレスからのアクセスのみを許可するように設定されたサーバーと通信できるようになります。



重要

egress ルーターはすべての発信接続のために使用されることが意図されていません。多数の egress ルーターを作成することで、ネットワークハードウェアの制限を引き上げる可能性があります。たとえば、すべてのプロジェクトまたはアプリケーションに egress ルーターを作成すると、ソフトウェアの MAC アドレスのフィルターにフォールバックする前にネットワークインターフェースが処理できるローカル MAC アドレス数の上限を超えてしまう可能性があります。



重要

現時点で、egress ルーターには Amazon AWS との互換性がありません。AWS に macvlan トラフィックとの互換性がないためです。

デプロイメントに関する考慮事項

Egressルーターは 2 つ目の IP アドレスおよび MAC アドレスをノードのプライマリーネットワークインターフェースに追加します。OpenShift Container Platform をベアメタルで実行していない場合は、ハイパーバイザーまたはクラウドプロバイダーが追加のアドレスを許可するように設定する必要があります。

Red Hat OpenStack Platform

OpenShift Container Platform を Red Hat OpenStack Platform を使ってデプロイしている場合、OpenStack 環境で IP および MAC アドレスのホワイトリストを作成する必要があります。これを行わないと、[通信は失敗します](#)。

```
neutron port-update $neutron_port_uuid \
  --allowed_address_pairs list=true \
  type=dict mac_address=<mac_address>,ip_address=<ip_address>
```

Red Hat Enterprise Virtualization

[Red Hat Enterprise Virtualization](#) を使用している場合は、**EnableMACAntiSpoofingFilterRules** を **false** に設定する必要があります。

VMware vSphere

VMware vSphere を使用している場合は、[vSphere 標準スイッチのセキュリティー保護についての VMWare ドキュメント](#)を参照してください。vSphere Web クライアントからホストの仮想スイッチを選択して、VMWare vSphere デフォルト設定を表示し、変更します。

とくに、以下が有効にされていることを確認します。

- [MAC アドレスの変更](#)
- [偽装転送 \(Forged Transit\)](#)
- [無作為別モード \(Promiscuous Mode\) 操作](#)

Egress ルーターモード

egress ルーターは、[リダイレクトモード](#)と[HTTP プロキシモード](#)の2つの異なるモードで実行できます。リダイレクトモードは、HTTP および HTTPS 以外のすべてのサービスで機能します。HTTP および HTTPS サービスの場合は、HTTP プロキシモードを使用します。

6.5.2.1. リダイレクトモードでの Egress ルーター Pod のデプロイ

リダイレクトモードでは、egress ルーターは、トラフィックを独自の IP アドレスから 1 つ以上の宛先 IP アドレスにリダイレクトするために iptables ルールをセットアップします。予約されたソース IP アドレスを使用する必要があるクライアント Pod は、宛先 IP に直接接続するのではなく、egress ルーターに接続するように変更される必要があります。

1. 上記を使用して Pod 設定を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ❶
spec:
  initContainers:
  - name: egress-router
    image: registry.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
      - name: EGRESS_SOURCE ❷
        value: 192.168.12.99
      - name: EGRESS_GATEWAY ❸
        value: 192.168.12.1
      - name: EGRESS_DESTINATION ❹
        value: 203.0.113.25
      - name: EGRESS_ROUTER_MODE ❺
        value: init
  containers:
  - name: egress-router-wait
    image: registry.access.redhat.com/openshift3/ose-pod
  nodeSelector:
    site: springfield-1 ❻
```

- 1 プライマリーネットワークインターフェースで Macvlan ネットワークインターフェースを作成し、これを Pod のネットワークプロジェクトに移行してから **egress-router** コンテナを起動します。"**true**" の周りの引用符はそのまま残します。これらを省略すると、エラーが発生します。プライマリーネットワークインターフェース以外のネットワークインターフェースで Macvlan インターフェースを作成するには、アノテーションの値を該当インターフェースの名前に設定します。たとえば、**eth1** を使用します。
- 2 ノードが置かれている物理ネットワークの IP アドレスで、この Pod で使用するようにクラスター管理者が予約します。
- 3 ノードで使用されるデフォルトゲートウェイと同じ値です。
- 4 トラフィックの送信先となる外部サーバー。この例では、Pod の接続は 203.0.113.25 にリダイレクトされます。ソース IP アドレスは 192.168.12.99 です。
- 5 これは egress ルーターイメージに対して、これが「init コンテナ」としてデプロイされていることを示しています。以前のバージョンの OpenShift Container Platform (および egress ルーターイメージ) はこのモードをサポートしておらず、通常のコンテナとして実行される必要がありました。
- 6 Pod はラベル **site=springfield-1** の設定されたノードにのみデプロイされます。

2. 上記の定義を使用して Pod を作成します。

```
$ oc create -f <pod_name>.json
```

Pod が作成されているかどうかを確認するには、以下を実行します。

```
$ oc get pod <pod_name>
```

3. egress ルーターを参照するサービスを作成し、他の Pod が Pod の IP アドレスを見つけられるようにします。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: http
      port: 80
    - name: https
      port: 443
  type: ClusterIP
  selector:
    name: egress-1
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにリダイレクトされます。

egress ルーターのセットアップは、**openshift3/ose-egress-router** イメージで作成される「init コンテナ」で実行され、このコンテナは Macvlan インターフェースを設定し、**iptables** ルールをセットアップできるように特権モード実行されます。**iptables** ルールのセットアップ終了後に、これは終

了し、**openshift3/ose-pod** コンテナが Pod が強制終了されるまで (特定のタスクを実行しない) 実行状態になります。

環境変数は **egress-router** イメージに対し、使用するアドレスを指示します。これは、**EGRESS_SOURCE** を IP アドレスとして、また **EGRESS_GATEWAY** をゲートウェイとして使用するよう Macvlan を設定します。

NAT ルールが設定され、Pod のクラスター IP アドレスの TCP または UDP ポートへの接続が **EGRESS_DESTINATION** の同じポートにリダイレクトされるようにします。

クラスター内の一部のノードのみが指定されたソース IP アドレスを要求でき、指定されたゲートウェイを使用できる場合、受け入れ可能なノードを示す **nodeName** または **nodeSelector** を指定することができます。

6.5.2.2. 複数の宛先へのリダイレクト

前の例では、任意のポートでの egress Pod (またはその対応するサービス) への接続は単一の宛先 IP にリダイレクトされます。ポートによっては複数の異なる宛先 IP を設定することもできます。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
  env:
  - name: EGRESS_SOURCE
    value: 192.168.12.99
  - name: EGRESS_GATEWAY
    value: 192.168.12.1
  - name: EGRESS_DESTINATION
    value: | ①
      80    tcp 203.0.113.25
      8080  tcp 203.0.113.26 80
      8443  tcp 203.0.113.26 443
      203.0.113.27
  - name: EGRESS_ROUTER_MODE
    value: init
  containers:
  - name: egress-router-wait
    image: registry.access.redhat.com/openshift3/ose-pod
```

① ここでは、複数行の文字列に YAML 構文を使用しています。詳細は以下を参照してください。

EGRESS_DESTINATION の各行は以下の 3 つのタイプのいずれかになります。

- **<port> <protocol> <IP address>**: これは、指定される **<port>** への着信接続が指定さ

れる **<IP address>** の同じポートにリダイレクトされることを示します。**<protocol>** は **tcp** または **udp** のいずれかになります。上記の例では、最初の行がローカルポート 80 から 203.0.113.25 のポート 80 にトラフィックをリダイレクトしています。

- **<port> <protocol> <IP address> <remote port>**: 接続が **<IP address>** の別の **<remote port>** にリダイレクトされるのを除き、上記と同じになります。この例では、2 番目と 3 番目の行ではローカルポート 8080 および 8443 を 203.0.113.26 のリモートポート 80 および 443 にリダイレクトしています。
- **<fallback IP address>**: **EGRESS_DESTINATION** の最後の行が単一 IP アドレスである場合、それ以外のポートの接続はその IP アドレス (上記の例では 203.0.113.27) の対応するポートにリダイレクトされます。フォールバック IP アドレスがない場合、他のポートでの接続は単純に拒否されます。)

6.5.2.3. ConfigMap の使用による EGRESS_DESTINATION の指定

宛先マッピングのセットのサイズが大きいか、またはこれが頻繁に変更される場合、ConfigMap を使用して一覧を外部で維持し、egress ルーター Pod がそこから一覧を読み取れるようにすることができます。これには、プロジェクト管理者が ConfigMap を編集できるという利点がありますが、これには特権付きコンテナが含まれるため、管理者は Pod 定義を直接編集することはできません。

1. **EGRESS_DESTINATION** データを含むファイルを作成します。

```
$ cat my-egress-destination.txt
# Egress routes for Project "Test", version 3

80    tcp 203.0.113.25

8080  tcp 203.0.113.26 80
8443  tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

空の行とコメントをこのファイルに追加できることに注意してください。

2. このファイルから ConfigMap オブジェクトを作成します。

```
$ oc delete configmap egress-routes --ignore-not-found
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

ここで、**egress-routes** は作成される ConfigMap オブジェクトの名前で、**my-egress-destination.txt** はデータの読み取り元のファイルの名前です。

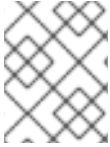
3. 前述のように egress ルーター Pod 定義を作成しますが、ConfigMap を環境セクションの **EGRESS_DESTINATION** に指定します。

```
...
env:
- name: EGRESS_SOURCE
  value: 192.168.12.99
- name: EGRESS_GATEWAY
  value: 192.168.12.1
- name: EGRESS_DESTINATION
```

```

    valueFrom:
      configMapKeyRef:
        name: egress-routes
        key: destination
  - name: EGRESS_ROUTER_MODE
    value: init
  ...

```



注記

egress ルーターは、ConfigMap が変更されても自動的に更新されません。更新を取得するには Pod を再起動します。

6.5.2.4. Egress ルーター HTTP プロキシ Pod のデプロイ

HTTP プロキシモードでは、egress ルーターはポート **8080** で HTTP プロキシとして実行されます。これは、HTTP または HTTPS ベースのサービスと通信するクライアントの場合にのみ機能しますが、通常それらを機能させるのにクライアント Pod への多くの変更は不要です。環境変数を設定することで、プログラムは HTTP プロキシを使用するように指示されます。

1. 例として以下を使用して Pod を作成します。

```

apiVersion: v1
kind: Pod
metadata:
  name: egress-http-proxy
  labels:
    name: egress-http-proxy
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ❶
spec:
  initContainers:
  - name: egress-router-setup
    image: registry.access.redhat.com/openshift3/ose-egress-router
    securityContext:
      privileged: true
  env:
  - name: EGRESS_SOURCE ❷
    value: 192.168.12.99
  - name: EGRESS_GATEWAY ❸
    value: 192.168.12.1
  - name: EGRESS_ROUTER_MODE ❹
    value: http-proxy
  containers:
  - name: egress-router-proxy
    image: registry.access.redhat.com/openshift3/ose-egress-http-proxy
    env:
  - name: EGRESS_HTTP_PROXY_DESTINATION ❺
    value: |
      !*.example.com
      !192.168.1.0/24
      *

```

- ❶ プライマリーネットワークインターフェースで Macvlan ネットワークインターフェースを作成してから、これを Pod のネットワークプロジェクトに移行し、**egress-router** コンテナを起動します。"**true**" の周りの引用符をそのまま残します。これらを省略すると、エラーが発生します。
- ❷ ノード自体が置かれている物理ネットワークの IP アドレスで、この Pod で使用するためにクラスター管理者が予約します。
- ❸ ノード自体で使用するデフォルトゲートウェイと同じ値。
- ❹ これは egress ルーターイメージに対し、これが HTTP プロキシの一部としてデプロイされているため、iptables のリダイレクトルールを設定できないことを示します。
- ❺ プロキシの設定方法を指定する文字列または YAML の複数行文字列です。これは、init コンテナの他の環境変数ではなく、HTTP プロキシコンテナの環境変数として指定されることに注意してください。

EGRESS_HTTP_PROXY_DESTINATION 値に以下のいずれかを指定できます。また、* を使用することができます。これは「すべてのリモート宛先への接続を許可」することを意味します。設定の各行には、許可または拒否する接続の 1 つのグループを指定します。

- IP アドレス (例: **192.168.1.1**) は該当する IP アドレスへの接続を許可します。
 - CIDR 範囲 (例: **192.168.1.0/24**) は CIDR 範囲への接続を許可します。
 - ホスト名 (例: **www.example.com**) は該当ホストへのプロキシを許可します。
 - *. が先に付けられるドメイン名 (例: ***.example.com**) は該当ドメインおよびそのサブドメインのすべてへのプロキシを許可します。
 - 上記のいずれかに ! を付けると、接続は許可されるのではなく、拒否されます。
 - 最後の行が * の場合、拒否されていないすべてのものが許可されます。または、許可されていないすべてのものが拒否されます。
2. egress ルーターを参照するサービスを作成し、他の Pod が Pod の IP アドレスを見つけられるようにします。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: http-proxy
      port: 8080 ❶
  type: ClusterIP
  selector:
    name: egress-1
```

- ❶ http ポートが常に **8080** に設定されていることを確認します。

3. **http_proxy** または **https_proxy** 変数を設定して、クライアント Pod (egress プロキシ Pod ではない) を HTTP プロキシを使用するように設定します。

```
...
env:
- name: http_proxy
  value: http://egress-1:8080/ ❶
- name: https_proxy
  value: http://egress-1:8080/
...
```

❶ 手順 2 で作成されたサービス。



注記

すべてのセットアップに **http_proxy** および **https_proxy** 環境変数が必要になる訳ではありません。上記を実行しても作業用セットアップが作成されない場合は、Pod で実行しているツールまたはソフトウェアについてのドキュメントを参照してください。

リダイレクトする **egress ルーター** の上記の例と同様に、ConfigMap を使用して **EGRESS_HTTP_PROXY_DESTINATION** を指定することもできます。

6.5.2.5. Egress ルーター Pod のフェイルオーバーの有効化

レプリケーションコントローラーを使用し、ダウンタイムを防ぐために egress ルーター Pod の 1 つのコピーを常に確保できるようにします。

1. 以下を使用してレプリケーションコントローラーの設定ファイルを作成します。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 ❶
  selector:
    name: egress-demo
  template:
    metadata:
      name: egress-demo
      labels:
        name: egress-demo
      annotations:
        pod.network.openshift.io/assign-macvlan: "true"
    spec:
      initContainers:
        - name: egress-demo-init
          image: registry.access.redhat.com/openshift3/ose-egress-
router
      env:
        - name: EGRESS_SOURCE
          value: 192.168.12.99
        - name: EGRESS_GATEWAY
          value: 192.168.12.1
        - name: EGRESS_DESTINATION
```



```

        value: 203.0.113.25
      - name: EGRESS_ROUTER_MODE
        value: init
      securityContext:
        privileged: true
    containers:
      - name: egress-demo-wait
        image: registry.access.redhat.com/openshift3/ose-pod
    nodeSelector:
      site: springfield-1

```

- 1 **replicas** が 1 に設定されていることを確認します。1 つの Pod のみが指定される **EGRESS_SOURCE** 値を随時使用できるためです。これは、ルーターの単一コピーのみがラベル **site=springfield-1** が設定されたノードで実行されることを意味します。

2. この定義を使用して Pod を作成します。

```
$ oc create -f <replication_controller>.json
```

3. 検証するには、レプリケーションコントローラー Pod が作成されているかどうかを確認します。

```
$ oc describe rc <replication_controller>
```

6.5.3. 外部リソースへのアクセスを制限するための iptables ルールの使用

クラスター管理者の中には、**EgressNetworkPolicy** のモデルや egress ルーターの対象外の発信トラフィックに対してアクションを実行する必要がある管理者がいる場合があります。この場合には、iptables ルールを直接作成してこれを実行することができます。

たとえば、特定の宛先へのトラフィックをログに記録するルールを作成したり、1 秒ごとに設定される特定数を超える発信接続を許可しないようにしたりできます。

OpenShift Container Platform はカスタム iptables ルールを自動的に追加する方法を提供していませんが、管理者がこのようなルールを手動で追加できる場所を提供します。各ノードは起動時に、**filter** テーブルに **OPENSHIFT-ADMIN-OUTPUT-RULES** という空のチェーンを作成します (チェーンがすでに存在していないと仮定します)。管理者がこのチェーンに追加するすべてのルールは、Pod からクラスター外にある宛先へのすべてのトラフィックに適用されます (それ以外のトラフィックには適用されません)。

この機能を使用する際には、注意すべきいくつかの点があります。

1. 各ノードにルールが作成されていることを確認するのは管理者のタスクになります。OpenShift Container Platform はこれを自動的に確認する方法は提供しません。
2. ルールは egress ルーターによってクラスターを退出するトラフィックには適用されず、ルールは **EgressNetworkPolicy** ルールが適用された後に実行されます (そのため、**EgressNetworkPolicy** で拒否されるトラフィックは表示されません)。
3. ノードには「外部」IP アドレスと「内部」SDN IP アドレスの両方があるため、Pod からノードまたはノードからマスターへの接続の処理は複雑になります。そのため、一部の Pod とノード間/Pod とマスター間のトラフィックはこのチェーンを通過しますが、他の Pod とノード間/Pod とマスター間のトラフィックはこれをバイパスする場合があります。

6.6. 外部プロジェクトトラフィックの静的 IP の有効化

クラスター管理者は特定の静的 IP アドレスをプロジェクトに割り当て、トラフィックが外部から容易に識別できるようにできます。これは、トラフィックを特定の宛先に送信するために使用されるデフォルトの egress ルーターの場合とは異なります。

識別可能な IP トラフィックは起点を可視化することで、クラスターのセキュリティーを強化します。これが有効にされると、指定されたプロジェクトからのすべての発信外部接続は同じ固定ソース IP を共有します。つまり、すべての外部リソースがこのトラフィックを認識できるようになります。

egress ルーターの場合とは異なり、これは **EgressNetworkPolicy** ファイアウォールルールに基づいて実行されます。

静的 IP を有効にするには、以下を実行します。

1. 必要な IP で **NetNamespace** を更新します。

```
$ oc patch netnamespace <project_name> -p '{"egressIPs": [  
  <IP_address>"]}'
```

たとえば、**MyProject** プロジェクトを IP アドレス 192.168.1.100 に割り当てるには、以下を実行します。

```
$ oc patch netnamespace MyProject -p '{"egressIPs":  
  ["192.168.1.100"]}'
```

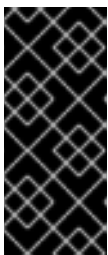
egressIPs フィールドは配列ですが、単一 IP アドレスに設定される必要があります。複数の IP を設定すると、他の IP は無視されます。

2. egress IP を必要なノードホストに手動で割り当てます。ノードホストの **HostSubnet** オブジェクトの **egressIPs** フィールドを設定します。そのノードホストに割り当てる必要のある任意の数の IP を含めることができます。

```
$ oc patch hostsubnet <node_name> -p \  
  '{"egressIPs": ["<IP_address_1>", "<IP_address_2>"]}'
```

たとえば **node1** に egress IPs 192.168.1.100、192.168.1.101 および 192.168.1.102 が必要である場合、以下ようになります。

```
$ oc patch hostsubnet node1 -p \  
  '{"egressIPs": ["192.168.1.100", "192.168.1.101",  
  "192.168.1.102"]}'
```



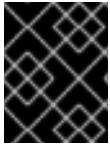
重要

Egress IP はプライマリーネットワークインターフェースの追加の IP として実装され、ノードのプライマリー IP と同じサブネットに置かれる必要があります。一部のクラウドまたは仮想マシンソリューションを使用する場合に、プライマリーネットワークインターフェースで追加の IP アドレスを許可するには追加の設定が必要になる場合があります。

プロジェクトに対して上記が有効にされる場合、そのプロジェクトからのすべての egress トラフィックはその egress IP をホストするノードにルーティングされ、(NAT を使用して) その IP アドレスに接

続されます。**egressIPs** が **NetNamespace** で設定されているものの、その egress IP をホストするノードがない場合、namespace からの egress トラフィックはドロップされます。

6.7. マルチキャストの有効化



重要

現時点で、マルチキャストは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のソリューションとしては適していません。

OpenShift Container Platform の Pod 間のマルチキャストトラフィックはデフォルトで無効にされています。プロジェクトの対応する **netnamespace** オブジェクトにアノテーションを設定することで、プロジェクトごとにマルチキャストを有効にすることができます。

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled=true
```

アノテーションを削除してマルチキャストを無効にします。

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled-
```

[ネットワークを結合](#)している場合、すべてのプロジェクトで有効にされるようにマルチキャストを各プロジェクトの **netnamespace** で有効にする必要があります。マルチキャストをデフォルトプロジェクトで有効にするには、これを **kube-service-catalog** プロジェクトおよび [グローバル](#) にされたその他すべてのプロジェクトでも有効にする必要があります。



注記

マルチキャストのグローバルプロジェクトは、ユニキャストの場合のクラスター内のすべてのプロジェクトと通信するという文脈で「グローバル」ではなく、マルチキャストによって他のグローバルプロジェクトのみと通信します。

6.8. NETWORKPOLICY の有効化

ovs-subnet および **ovs-multitenant** プラグインにはネットワークの分離についての独自のレガシーモデルがありますが、Kubernetes **NetworkPolicy** はサポートしません。ただし、**NetworkPolicy** サポートは、**ovs-networkpolicy** プラグインを使用すると利用できます。

ovs-networkpolicy プラグインを使用するよう設定されるクラスターでは、ネットワークの分離は完全に **NetworkPolicy** オブジェクトによって制御されます。デフォルトで、プロジェクトのすべての Pod は他の Pod およびネットワークのエンドポイントからアクセスできます。プロジェクト内の 1 つ以上の Pod を分離するには、そのプロジェクトで **NetworkPolicy** オブジェクトを作成し、許可する着信接続を指定します。プロジェクト管理者は独自のプロジェクト内で **NetworkPolicy** オブジェクトの作成および削除を実行できます。

Pod を参照する **NetworkPolicy** オブジェクトを持たない Pod は完全にアクセスできますが、Pod を参照する 1 つ以上の **NetworkPolicy** オブジェクトを持つ Pod は分離されます。これらの分離された Pod は 1 つ以上の **NetworkPolicy** オブジェクトで許可される接続のみを受け入れます。

複数の異なるシナリオに対応するいくつかの **NetworkPolicy** オブジェクト定義を見てみましょう。

- すべてのトラフィックを拒否

プロジェクトに「deny by default (デフォルトで拒否)」を実行させるには、すべての Pod に一致するが、トラフィックを一切許可しない **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
  ingress: []
```

- プロジェクト内の Pod からの接続のみを許可

Pod が同じプロジェクト内の他の Pod からの接続を受け入れるが、他のプロジェクトの Pod からの接続を拒否するように設定するには、以下を実行します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
    - from:
      - podSelector: {}
```

- Pod ラベルに基づいて HTTP および HTTPS トラフィックのみを許可

特定のラベル (以下の例の **role=frontend**) の付いた Pod への HTTP および HTTPS アクセスのみを有効にするには、以下と同様の **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
    - ports:
      - protocol: TCP
        port: 80
      - protocol: TCP
        port: 443
```

NetworkPolicy オブジェクトは加算されるものです。つまり、複数の **NetworkPolicy** オブジェクトを組み合わせることで複雑なネットワーク要件を満たすことができます。

たとえば、先の例で定義された **NetworkPolicy** オブジェクトの場合、同じプロジェクト内に **allow-same-namespace** と **allow-http-and-https** ポリシーの両方を定義することができます。これにより、ラベル **role=frontend** の付いた Pod は各ポリシーで許可されるすべての接続、つまり、同じ namespace の Pod からのすべての接続、および **すべての namespace の Pod からのポート 80 443** での接続を受け入れます。

6.8.1. NetworkPolicy およびルーター

ovs-multitenant プラグインを使用する場合、ルーターからすべての namespace へのトラフィックは自動的に許可されます。これは、ルーターは通常 **デフォルトの namespace** にあり、すべての namespace がその namespace の Pod からの接続を許可するためです。ただし **ovs-networkpolicy** プラグインを使用すると、これは自動的に実行されません。そのため、デフォルトで namespace を分離するポリシーがある場合は、ルーターがこれにアクセスできるように追加の手順を実行する必要があります。

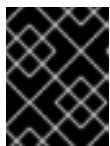
1 つのオプションとして、すべてのソースからのアクセスを許可する各サービスのポリシーを作成できます。以下は例になります。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-to-database-service
spec:
  podSelector:
    matchLabels:
      role: database
  ingress:
  - ports:
    - protocol: TCP
      port: 5432
```

これにより、ルーターはサービスにアクセスできますが、同時に他のユーザーの namespace にある Pod もこれにアクセスできます。これらの Pod は通常はパブリックルーターを使用してサービスにアクセスできるため、これによって問題が発生することはないはずです。

または、**ovs-multitenant** プラグインの場合のように、デフォルト namespace からの完全アクセスを許可するポリシーを作成することもできます。

1. ラベルをデフォルト namespace に追加します。



重要

これはクラスター全体に対して 1 回のみ実行する必要があります。クラスター管理ロールには、ラベルを namespace に追加することが求められます。

```
$ oc label namespace default name=default
```

2. その namespace からの接続を許可するポリシーを作成します。



注記

接続を許可するそれぞれの namespace についてこの手順を実行します。プロジェクト管理者ロールを持つユーザーがポリシーを作成できます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-from-default-namespace
spec:
  podSelector:
```

```

ingress:
- from:
  - namespaceSelector:
      matchLabels:
        name: default

```

6.8.2. 新規プロジェクトのデフォルト **NetworkPolicy** の設定

クラスター管理者は、新規プロジェクトの作成時に、デフォルトのプロジェクトテンプレートを変更してデフォルトの **NetworkPolicy** オブジェクト (1 つ以上) の自動作成を有効にできます。これを実行するには、以下を行います。

1. 「[新規プロジェクトのテンプレートの変更](#)」で説明されているように、カスタムプロジェクトテンプレートを作成し、マスターがこれを使用するように設定します。
2. 必要な **NetworkPolicy** オブジェクトを含むようにテンプレートを編集します。

```
$ oc edit template project-request -n default
```



注記

NetworkPolicy オブジェクトを既存テンプレートに含めるには、**oc edit** コマンドを使用します。現時点では、**oc patch** を使用してオブジェクトを **Template** リソースに追加することはできません。

- a. それぞれのデフォルトポリシーを **objects** 配列の要素として追加します。

```

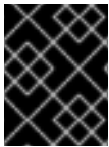
objects:
...
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector:
      ingress:
        - from:
            - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-default-namespace
  spec:
    podSelector:
      ingress:
        - from:
            - namespaceSelector:
                matchLabels:
                  name: default
...

```

6.9. HTTP STRICT TRANSPORT SECURITY の有効化

HTTP Strict Transport Security (HSTS) ポリシーは、ホストで HTTPS トラフィックのみを許可するセキュリティの拡張機能です。デフォルトで、すべての HTTP 要求はドロップされます。これは、web サイトとの対話の安全性を確保したり、ユーザーのためにセキュアなアプリケーションを提供するのに役立ちます。

HSTS が有効にされると、HSTS はサイトから Strict Transport Security ヘッダーを HTTPS 応答に追加します。リダイレクトするルートで **insecureEdgeTerminationPolicy** 値を使用し、HTTP を HTTPS に送信するようにします。ただし、HSTS が有効にされている場合は、要求の送信前にクライアントがすべての要求を HTTP URL から HTTPS に変更するためにリダイレクトの必要がなくなります。これはクライアントでサポートされる必要はなく、**max-age=0** を設定することで無効にできます。

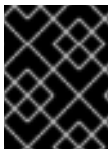


重要

HSTS はセキュアなルート (edge termination または re-encrypt) でのみ機能します。この設定は、HTTP またはパススルールートには適していません。

ルートに対して HSTS を有効にするには、**haproxy.router.openshift.io/hsts_header** 値を edge termination または re-encrypt ルートに追加します。

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
```



重要

haproxy.router.openshift.io/hsts_header 値にパラメーターのスペースやその他の値が入っていないことを確認します。**max-age** のみが必要になります。

必須の **max-age** パラメーターは、HSTS ポリシーの有効期間 (秒単位) を示します。クライアントは、ホストから HSTS ヘッダーのある応答を受信する際には常に **max-age** を更新します。**max-age** がタイムアウトになると、クライアントはポリシーを破棄します。

オプションの **includeSubDomains** パラメーターは、クライアントに対し、ホストのすべてのサブドメインがホストと同様に処理されるように指示します。

max-age が 0 より大きい場合、オプションの **preload** パラメーターは外部サービスがこのサイトをそれぞれの HSTS プリロードのリストに含めることを許可します。たとえば、Google などのサイトは **preload** が設定されているサイトの一覧を作成します。ブラウザはこれらのリストを使用し、サイトと対話する前でも HTTPS 経由でのみ通信するサイトを判別できます。**preload** 設定がない場合、ブラウザはヘッダーを取得するために HTTPS 経由でサイトと通信している必要があります。

6.10. スループットの問題のトラブルシューティング

OpenShift Container Platform でデプロイされるアプリケーションでは、特定のサービス間で非常に長い待ち時間が発生するなど、ネットワークのスループットの問題が生じることがあります。

Pod のログが問題の原因を指摘しない場合は、以下の方法を使用してパフォーマンスの問題を分析します。

- ping または `tcpdump` などのパケットアナライザーを使用して Pod とそのノード間のトラフィックを分析します。
たとえば、問題を生じさせる動作を再現している間に各ノードで `tcpdump` ツールを実行します。両サイトでキャプチャーしたデータを確認し、送信および受信タイムスタンプを比較して Pod への/からのトラフィックの待ち時間を分析します。待ち時間は、ノードのインターフェースが他の Pod やストレージデバイス、またはデータプレーンからのトラフィックでオーバーロードする場合に OpenShift Container Platform で発生する可能性があります。

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host  
<podip 2> 1
```

- 1 **podip** は Pod の IP アドレスです。以下のコマンドを実行して Pod の IP アドレスを取得します。

```
# oc get pod <podname> -o wide
```

`tcpdump` は 2 つの Pod 間のすべてのトラフィックが含まれる `/tmp/dump.pcap` のファイルを生成します。理想的には、ファイルサイズを最小限に抑えるために問題を再現するすぐ前と問題を再現したすぐ後にアナライザーを実行することが良いでしょう。以下のようにノード間でパケットアナライザーを実行することもできます (式から SDN を排除する)。

```
# tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- ストリーミングのスループットおよび UDP スループットを測定するために `iperf` などの帯域幅測定ツールを使用します。ボトルネックの特定を試行するには、最初に Pod から、次にノードからツールを実行します。`iperf3` ツールは RHEL 7 の一部として組み込まれています。

`iperf3` のインストールおよび使用についての詳細は、こちらの [Red Hat ソリューション](#) を参照してください。

第7章 サービスアカウントの設定

7.1. 概要

ユーザーが OpenShift Container Platform CLI または web コンソールを使用する場合、API トークンはユーザーを OpenShift Container Platform API に対して認証します。ただし、一般ユーザーの認証情報を利用できない場合、以下のようにコンポーネントが API 呼び出しを行うのが通例になります。

- レプリケーションコントローラーが Pod を作成するか、または削除するために API 呼び出しを実行する。
- コンテナ内のアプリケーションが検出目的で API 呼び出しを実行する。
- 外部アプリケーションがモニターまたは統合目的で API 呼び出しを実行する。

サービスアカウントは、一般ユーザーの認証情報を共有せずに API アクセスをより柔軟に制御する方法を提供します。

7.2. ユーザー名およびグループ

すべてのサービスアカウントには、一般ユーザーのように、ロールを付与できるユーザー名が関連付けられています。ユーザー名はそのプロジェクトおよび名前から派生します。

```
system:serviceaccount:<project>:<name>
```

たとえば、**view (表示)** ロールを **top-secret** プロジェクトの **robot** サービスアカウントに追加するには、以下を実行します。

```
$ oc policy add-role-to-user view system:serviceaccount:top-secret:robot
```

重要

プロジェクトの特定のサービスアカウントにアクセスを付与する必要がある場合は、**-z** フラグを使用できます。サービスアカウントが属するプロジェクトから、**-z** フラグを使用し、**<serviceaccount_name>** を指定します。これにより誤字を防ぎ、アクセスを指定したサービスアカウントのみに付与できるため、この方法を使用することを強くお勧めします。以下は例になります。

```
$ oc policy add-role-to-user <role_name> -z
<serviceaccount_name>
```

プロジェクトにいない場合は、以下の例に示すように **-n** オプションを使用してこれを適用するプロジェクトの namespace を示します。

すべてのサービスアカウントは以下の 2 つのグループのメンバーです。

system:serviceaccount

システムのすべてのサービスアカウントが含まれます。

system:serviceaccount:<project>

指定されたプロジェクトのすべてのサービスアカウントが含まれます。

たとえば、すべてのプロジェクトのすべてのサービスアカウントが **top-secret** プロジェクトのリソースを表示できるようにするには、以下を実行します。

```
$ oc policy add-role-to-group view system:serviceaccount -n top-secret
```

managers プロジェクトのすべてのサービスアカウントが **top-secret** プロジェクトのリソースを編集できるようにするには、以下を実行します。

```
$ oc policy add-role-to-group edit system:serviceaccount:managers -n top-secret
```

7.3. サービスアカウントの管理

サービスアカウントは、各プロジェクトに存在する API オブジェクトです。サービスアカウントを管理するには、**sa** または **serviceaccount** オブジェクトタイプと共に **oc** コマンドを使用するか、または web コンソールを使用することができます。

現在のプロジェクトで既存のサービスアカウントの一覧を取得するには、以下を実行します。

```
$ oc get sa
NAME          SECRETS    AGE
builder       2          2d
default       2          2d
deployer      2          2d
```

新規のサービスアカウントを作成するには、以下を実行します。

```
$ oc create sa robot
serviceaccount "robot" created
```

サービスアカウントの作成後すぐに、以下の 2 つのシークレットがこれに自動的に追加されます。

- API トークン
- OpenShift Container レジストリーの認証情報

これらは、サービスアカウントの記述で表示できます。

```
$ oc describe sa robot
Name: robot
Namespace: project1
Labels: <none>
Annotations: <none>

Image pull secrets: robot-dockercfg-qzbhb

Mountable secrets:  robot-token-f4khf
                    robot-dockercfg-qzbhb

Tokens:
    robot-token-f4khf
    robot-token-z8h44
```

システムはサービスアカウントに API トークンとレジストリーの認証情報が常にあることを確認します。

生成される API トークンとレジストリーの認証情報は期限切れになることはありませんが、シークレットを削除することで取り消すことができます。シークレットが削除されると、新規のシークレットが自動生成され、これに置き換わります。

7.4. サービスアカウント認証の有効化

サービスアカウントは、プライベート RSA キーで署名されるトークンを使用して API に対して認証されます。認証層では一致するパブリック RSA キーを使用して署名を検証します。

サービスアカウントトークンの生成を有効にするには、マスターで `/etc/origin/master/master-config.yml` ファイルの **serviceAccountConfig** スタンザを更新し、(署名用に) **privateKeyFile** と **publicKeyFiles** 一覧の一致するパブリックキーファイルを指定します。

```
serviceAccountConfig:
  ...
  masterCA: ca.crt ❶
  privateKeyFile: serviceaccount.private.key ❷
  publicKeyFiles:
  - serviceaccount.public.key ❸
  - ...
```

- ❶ API サーバーの提供する証明書を検証するために使用される CA ファイル。
- ❷ プライベート RSA キーファイル (トークンの署名用)。
- ❸ パブリック RSA キーファイル (トークンの検証用)。プライベートキーファイルが提供されている場合、パブリックキーコンポーネントが使用されます。複数のパブリックキーファイルを使用でき、トークンはパブリックキーのいずれかで検証できる場合に受け入れられます。これにより、署名するキーのローテーションが可能となり、以前の署名者が生成したトークンは依然として受け入れられます。

7.5. 管理サービスアカウント

サービスアカウントは、ビルド、デプロイメントおよびその他の Pod を実行するために各プロジェクトで必要になります。マスターの `/etc/origin/master/master-config.yml` ファイルの **managedNames** 設定は、すべてのプロジェクトに自動作成されるサービスアカウントを制御します。

```
serviceAccountConfig:
  ...
  managedNames: ❶
  - builder ❷
  - deployer ❸
  - default ❹
  - ...
```

- ❶ すべてのプロジェクトで自動作成するサービスアカウントの一覧。

❷

各プロジェクトの **builder** サービスは、ビルド Pod で必要になり、内部コンテナレジストリーを使用してプロジェクトのイメージストリームにイメージをプッシュする **system:image-builder**

- 3 各プロジェクトの **deployer** サービスアカウントはデプロイメント Pod で必要になり、レプリケーションコントローラーおよびプロジェクトの Pod の表示および変更を可能にする **system:deployer** ロールが付与されます。
- 4 デフォルトのサービスアカウントは、別のサービスアカウントが指定されない限り、他のすべての Pod で使用されます。

プロジェクトのすべてのサービスアカウントには、内部コンテナレジストリーを使用してイメージストリームからイメージのプルを可能にする **system:image-puller** ロールが付与されます。

7.6. インフラストラクチャーサービスアカウント

一部のインフラストラクチャーコントローラーは、サービスアカウント認証情報を使用して実行されます。以下のサービスアカウントは、サーバーの起動時に OpenShift Container Platform インフラストラクチャープロジェクト (**openshift-infra**) に作成され、クラスター全体で以下のロールが付与されます。

サービスアカウント	説明
replication-controller	system:replication-controller ロールの割り当て
deployment-controller	system:deployment-controller ロールの割り当て
build-controller	system:build-controller ロールの割り当て。さらに、 build-controller サービスアカウントは、特権付きのビルド Pod を作成するために特権付きセキュリティコンテキストに組み込まれます。

これらのサービスアカウントが作成されるプロジェクトを設定するには、マスターで **/etc/origin/master/master-config.yml** ファイルの **openshiftInfrastructureNamespace** フィールドを設定します。

```
policyConfig:
  ...
  openshiftInfrastructureNamespace: openshift-infra
```

7.7. サービスアカウントおよびシークレット

マスターで **/etc/origin/master/master-config.yml** ファイルの **limitSecretReferences** フィールドを **true** に設定し、Pod のシークレット参照をサービスアカウントでホワイトリストに入れることが必要になるようにします。この値を **false** に設定すると、Pod がプロジェクトのすべてのシークレットを参照できるようになります。

```
serviceAccountConfig:
  ...
  limitSecretReferences: false
```

第8章 ロールベースアクセス制御 (RBAC) の管理

8.1. 概要

CLI を使用して **RBAC リソース** を表示し、管理者 CLI を使用して **ロールとバインディング** を管理することができます。

8.2. ロールとバインディングの表示

ロールは、**クラスター全体**および**プロジェクトのスコープ**の両方で各種のアクセスレベルを付与するために使用できます。**ユーザーおよびグループ**は、1 度に複数のロールに関連付けるか、または**バインド**することができます。**oc describe** コマンドを使用して、ロールおよびそれらのバインディングの詳細を確認できます。

クラスター全体でバインドされた **cluster-admin** の**デフォルトクラスターロール**を持つユーザーはすべてのリソースに対してすべてのアクションを実行できます。ローカルにバインドされた **admin** の**デフォルトクラスターロール**を持つユーザーはそのプロジェクト内のロールとバインディングをローカルに管理できます。



注記

「[Evaluating Authorization](#)」セクションで動詞の詳細リストを確認してください。

8.2.1. クラスターロールの表示

クラスターロールおよびそれらの関連付けられたルールセットを表示するには、以下を実行します。

```
$ oc describe clusterrole.rbac
```

クラスターロールの表示

```
$ oc describe clusterrole.rbac
Name: admin
Labels: <none>
Annotations: openshift.io/description=A user that has edit rights within
the project and can change the project's membership.
rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources          Non-Resource URLs Resource Names Verbs
  -----
  appliedclusterresourcequotas      []      [] [get list watch]
  appliedclusterresourcequotas.quota.openshift.io []      [] [get list
watch]
  bindings          []      [] [get list watch]
  buildconfigs      []      [] [create delete deletecollection get list
patch update watch]
  buildconfigs.build.openshift.io    []      [] [create delete
deletecollection get list patch update watch]
  buildconfigs/instantiate      []      [] [create]
  buildconfigs.build.openshift.io/instantiate []      [] [create]
  buildconfigs/instantiatebinary []      [] [create]
  buildconfigs.build.openshift.io/instantiatebinary []      [] [create]
  buildconfigs/webhooks          []      [] [create delete deletecollection get
```

```

list patch update watch]
  buildconfigs.build.openshift.io/webhooks [] [] [create delete
deletecollection get list patch update watch]
  buildlogs [] [] [create delete deletecollection get list patch
update watch]
  buildlogs.build.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  builds [] [] [create delete deletecollection get list patch
update watch]
  builds.build.openshift.io [] [] [create delete deletecollection
get list patch update watch]
  builds/clone [] [] [create]
  builds.build.openshift.io/clone [] [] [create]
  builds/details [] [] [update]
  builds.build.openshift.io/details [] [] [update]
  builds/log [] [] [get list watch]
  builds.build.openshift.io/log [] [] [get list watch]
  configmaps [] [] [create delete deletecollection get list patch
update watch]
  cronjobs.batch [] [] [create delete deletecollection get list
patch update watch]
  daemonsets.extensions [] [] [get list watch]
  deploymentconfigrollbacks [] [] [create]
  deploymentconfigrollbacks.apps.openshift.io [] [] [create]
  deploymentconfigs [] [] [create delete deletecollection get list
patch update watch]
  deploymentconfigs.apps.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  deploymentconfigs/instantiate [] [] [create]
  deploymentconfigs.apps.openshift.io/instantiate [] [] [create]
  deploymentconfigs/log [] [] [get list watch]
  deploymentconfigs.apps.openshift.io/log [] [] [get list watch]
  deploymentconfigs/rollback [] [] [create]
  deploymentconfigs.apps.openshift.io/rollback [] [] [create]
  deploymentconfigs/scale [] [] [create delete deletecollection get
list patch update watch]
  deploymentconfigs.apps.openshift.io/scale [] [] [create delete
deletecollection get list patch update watch]
  deploymentconfigs/status [] [] [get list watch]
  deploymentconfigs.apps.openshift.io/status [] [] [get list watch]
  deployments.apps [] [] [create delete deletecollection get list
patch update watch]
  deployments.extensions [] [] [create delete deletecollection get
list patch update watch]
  deployments.extensions/rollback [] [] [create delete
deletecollection get list patch update watch]
  deployments.apps/scale [] [] [create delete deletecollection get
list patch update watch]
  deployments.extensions/scale [] [] [create delete
deletecollection get list patch update watch]
  deployments.apps/status [] [] [create delete deletecollection get
list patch update watch]
  endpoints [] [] [create delete deletecollection get list patch
update watch]
  events [] [] [get list watch]
  horizontalpodautoscalers.autoscaling [] [] [create delete

```

```

deletecollection get list patch update watch]
  horizontalpodautoscalers.extensions [] [] [create delete
deletecollection get list patch update watch]
  imagestreamimages [] [] [create delete deletecollection get list
patch update watch]
  imagestreamimages.image.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  imagestreamimports [] [] [create]
  imagestreamimports.image.openshift.io [] [] [create]
  imagestreammappings [] [] [create delete deletecollection get
list patch update watch]
  imagestreammappings.image.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  imagestreams [] [] [create delete deletecollection get list
patch update watch]
  imagestreams.image.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  imagestreams/layers [] [] [get update]
  imagestreams.image.openshift.io/layers [] [] [get update]
  imagestreams/secrets [] [] [create delete deletecollection get
list patch update watch]
  imagestreams.image.openshift.io/secrets [] [] [create delete
deletecollection get list patch update watch]
  imagestreams/status [] [] [get list watch]
  imagestreams.image.openshift.io/status [] [] [get list watch]
  imagestreamtags [] [] [create delete deletecollection get list
patch update watch]
  imagestreamtags.image.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  jenkins.build.openshift.io [] [] [admin edit view]
  jobs.batch [] [] [create delete deletecollection get list patch
update watch]
  limitranges [] [] [get list watch]
  localresourceaccessreviews [] [] [create]
  localresourceaccessreviews.authorization.openshift.io [] [] [create]
  localsubjectaccessreviews [] [] [create]
  localsubjectaccessreviews.authorization.k8s.io [] [] [create]
  localsubjectaccessreviews.authorization.openshift.io [] [] [create]
  namespaces [] [] [get list watch]
  namespaces/status [] [] [get list watch]
  networkpolicies.extensions [] [] [create delete deletecollection
get list patch update watch]
  persistentvolumeclaims [] [] [create delete deletecollection get
list patch update watch]
  pods [] [] [create delete deletecollection get list patch
update watch]
  pods/attach [] [] [create delete deletecollection get list
patch update watch]
  pods/exec [] [] [create delete deletecollection get list patch
update watch]
  pods/log [] [] [get list watch]
  pods/portforward [] [] [create delete deletecollection get list
patch update watch]
  pods/proxy [] [] [create delete deletecollection get list patch
update watch]
  pods/status [] [] [get list watch]

```

```

podsecuritypolicyreviews      [] [] [create]
podsecuritypolicyreviews.security.openshift.io [] [] [create]
podsecuritypolicyselfsubjectreviews [] [] [create]
podsecuritypolicyselfsubjectreviews.security.openshift.io [] []
[create]
podsecuritypolicysubjectreviews [] [] [create]
podsecuritypolicysubjectreviews.security.openshift.io [] [] [create]
processedtemplates [] [] [create delete deletecollection get
list patch update watch]
processedtemplates.template.openshift.io [] [] [create delete
deletecollection get list patch update watch]
projects [] [] [delete get patch update]
projects.project.openshift.io [] [] [delete get patch update]
replicasets.extensions [] [] [create delete deletecollection get
list patch update watch]
replicasets.extensions/scale [] [] [create delete
deletecollection get list patch update watch]
replicationcontrollers [] [] [create delete deletecollection get
list patch update watch]
replicationcontrollers/scale [] [] [create delete
deletecollection get list patch update watch]
replicationcontrollers.extensions/scale [] [] [create delete
deletecollection get list patch update watch]
replicationcontrollers/status [] [] [get list watch]
resourceaccessreviews [] [] [create]
resourceaccessreviews.authorization.openshift.io [] [] [create]
resourcequotas [] [] [get list watch]
resourcequotas/status [] [] [get list watch]
resourcequotausage [] [] [get list watch]
rolebindingrestrictions [] [] [get list watch]
rolebindingrestrictions.authorization.openshift.io [] [] [get list
watch]
rolebindings [] [] [create delete deletecollection get list
patch update watch]
rolebindings.authorization.openshift.io [] [] [create delete
deletecollection get list patch update watch]
rolebindings.rbac.authorization.k8s.io [] [] [create delete
deletecollection get list patch update watch]
roles [] [] [create delete deletecollection get list patch
update watch]
roles.authorization.openshift.io [] [] [create delete
deletecollection get list patch update watch]
roles.rbac.authorization.k8s.io [] [] [create delete
deletecollection get list patch update watch]
routes [] [] [create delete deletecollection get list patch
update watch]
routes.route.openshift.io [] [] [create delete deletecollection
get list patch update watch]
routes/custom-host [] [] [create]
routes.route.openshift.io/custom-host [] [] [create]
routes/status [] [] [get list watch update]
routes.route.openshift.io/status [] [] [get list watch update]
scheduledjobs.batch [] [] [create delete deletecollection get
list patch update watch]
secrets [] [] [create delete deletecollection get list patch
update watch]

```

```

  serviceaccounts      [] [] [create delete deletecollection get list
patch update watch impersonate]
  services             [] [] [create delete deletecollection get list patch
update watch]
  services/proxy       [] [] [create delete deletecollection get list
patch update watch]
  statefulsets.apps    [] [] [create delete deletecollection get list
patch update watch]
  subjectaccessreviews [] [] [create]
  subjectaccessreviews.authorization.openshift.io [] [] [create]
  subjectrulesreviews [] [] [create]
  subjectrulesreviews.authorization.openshift.io [] [] [create]
  templateconfigs      [] [] [create delete deletecollection get list
patch update watch]
  templateconfigs.template.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  templateinstances    [] [] [create delete deletecollection get list
patch update watch]
  templateinstances.template.openshift.io [] [] [create delete
deletecollection get list patch update watch]
  templates            [] [] [create delete deletecollection get list patch
update watch]
  templates.template.openshift.io [] [] [create delete
deletecollection get list patch update watch]

```

Name: basic-user

Labels: <none>

Annotations: openshift.io/description=A user that can get basic information about projects.

rbac.authorization.kubernetes.io/autoupdate=true

PolicyRule:

Resources	Non-Resource URLs	Resource Names	Verbs
clusterroles			[get list]
clusterroles.authorization.openshift.io			[get list]
clusterroles.rbac.authorization.k8s.io			[get list watch]
projectrequests			[list]
projectrequests.project.openshift.io			[list]
projects			[list watch]
projects.project.openshift.io			[list watch]
selfsubjectaccessreviews.authorization.k8s.io			[create]
selfsubjectrulesreviews			[create]
selfsubjectrulesreviews.authorization.openshift.io			[create]
storageclasses.storage.k8s.io			[get list]
users		[~]	[get]
users.user.openshift.io		[~]	[get]

Name: cluster-admin

Labels: <none>

Annotations: authorization.openshift.io/system-only=true

openshift.io/description=A super-user that can perform any action in the cluster. When granted to a user within a project, they have full control over quota and membership and can perform every action...

rbac.authorization.kubernetes.io/autoupdate=true


```

PolicyRule:
  Resources Non-Resource URLs Resource Names Verbs
  -----
  [*]      []      [*]
  *.*      []      [*]

Name: cluster-debugger
Labels: <none>
Annotations: authorization.openshift.io/system-only=true
             rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources Non-Resource URLs Resource Names Verbs
  -----
  [/debug/pprof] [] [get]
  [/debug/pprof/*] [] [get]
  [/metrics] [] [get]

Name: cluster-reader
Labels: <none>
Annotations: authorization.openshift.io/system-only=true
             rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources Non-Resource URLs Resource Names Verbs
  -----
  [*]      [] [get]
  apiservices.apiregistration.k8s.io [] [] [get list watch]
  apiservices.apiregistration.k8s.io/status [] [] [get list watch]
  appliedclusterresourcequotas [] [] [get list watch]
...

```

各種のロールにバインドされたユーザーおよびグループを示す、クラスターのロールバインディングの現在のセットを表示するには、以下を実行します。

```
$ oc describe clusterrolebinding.rbac
```

クラスターのロールバインディングの表示

```

$ oc describe clusterrolebinding.rbac
Name: admin
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind  Name  Namespace
  ----  ---  -
  ServiceAccount template-instance-controller openshift-infra

Name: basic-users
Labels: <none>

```

```
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
```

```
Role:
```

```
  Kind: ClusterRole
```

```
  Name: basic-user
```

```
Subjects:
```

```
  Kind Name      Namespace
```

```
  ----
```

```
  Group system:authenticated
```

```
Name: cluster-admin
```

```
Labels: kubernetes.io/bootstrapping=rbac-defaults
```

```
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
```

```
Role:
```

```
  Kind: ClusterRole
```

```
  Name: cluster-admin
```

```
Subjects:
```

```
  Kind Name      Namespace
```

```
  ----
```

```
  ServiceAccount pvinstaller default
```

```
  Group system:masters
```

```
Name: cluster-admins
```

```
Labels: <none>
```

```
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
```

```
Role:
```

```
  Kind: ClusterRole
```

```
  Name: cluster-admin
```

```
Subjects:
```

```
  Kind Name      Namespace
```

```
  ----
```

```
  Group system:cluster-admins
```

```
  User system:admin
```

```
Name: cluster-readers
```

```
Labels: <none>
```

```
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
```

```
Role:
```

```
  Kind: ClusterRole
```

```
  Name: cluster-reader
```

```
Subjects:
```

```
  Kind Name      Namespace
```

```
  ----
```

```
  Group system:cluster-readers
```

```
Name: cluster-status-binding
```

```
Labels: <none>
```

```
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
```

```
Role:
```

```
  Kind: ClusterRole
```

```
  Name: cluster-status
```

```
Subjects:
```

```
  Kind Name      Namespace
```

```

-----
Group system:authenticated
Group system:unauthenticated

```

```

Name: registry-registry-role
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:registry
Subjects:
  Kind  Name  Namespace
  ----  ----  -
  ServiceAccount registry default

```

```

Name: router-router-role
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:router
Subjects:
  Kind  Name  Namespace
  ----  ----  -
  ServiceAccount router default

```

```

Name: self-access-reviewers
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-access-reviewer
Subjects:
  Kind Name  Namespace
  ---- ----  -
  Group system:authenticated
  Group system:unauthenticated

```

```

Name: self-provisioners
Labels: <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: self-provisioner
Subjects:
  Kind Name  Namespace
  ---- ----  -
  Group system:authenticated:oauth

```

```

Name: system:basic-user
Labels: kubernetes.io/bootstrapping=rbac-defaults

```

Annotations: rbac.authorization.kubernetes.io/autoupdate=true

Role:

Kind: ClusterRole

Name: system:basic-user

Subjects:

Kind	Name	Namespace
-----	-----	-----
Group	system:authenticated	
Group	system:unauthenticated	

Name: system:build-strategy-docker-binding

Labels: <none>

Annotations: rbac.authorization.kubernetes.io/autoupdate=true

Role:

Kind: ClusterRole

Name: system:build-strategy-docker

Subjects:

Kind	Name	Namespace
-----	-----	-----
Group	system:authenticated	

Name: system:build-strategy-jenkinspipeline-binding

Labels: <none>

Annotations: rbac.authorization.kubernetes.io/autoupdate=true

Role:

Kind: ClusterRole

Name: system:build-strategy-jenkinspipeline

Subjects:

Kind	Name	Namespace
-----	-----	-----
Group	system:authenticated	

Name: system:build-strategy-source-binding

Labels: <none>

Annotations: rbac.authorization.kubernetes.io/autoupdate=true

Role:

Kind: ClusterRole

Name: system:build-strategy-source

Subjects:

Kind	Name	Namespace
-----	-----	-----
Group	system:authenticated	

Name: system:controller:attachdetach-controller

Labels: kubernetes.io/bootstrapping=rbac-defaults

Annotations: rbac.authorization.kubernetes.io/autoupdate=true

Role:

Kind: ClusterRole

Name: system:controller:attachdetach-controller

Subjects:

Kind	Name	Namespace
-----	-----	-----

```
ServiceAccount attachdetach-controller kube-system
```

```
Name: system:controller:certificate-controller
Labels: kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true
Role:
  Kind: ClusterRole
  Name: system:controller:certificate-controller
Subjects:
  Kind  Name  Namespace
  ----  -
  ServiceAccount certificate-controller kube-system

Name: system:controller:cronjob-controller
Labels: kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate=true

...
```

8.2.2. ローカルのロールバインディングの表示

すべての[デフォルトクラスターロール](#)は、ユーザーまたはグループにローカルにバインドできます。

[カスタムローカルロール](#)を作成できます。

ローカルのロールバインディングも表示することができます。

各種のロールにバインドされたユーザーおよびグループを示す、ローカルのロールバインディングの現在のセットを表示するには、以下を実行します。

```
$ oc describe rolebinding.rbac
```

デフォルトでは、ローカルのロールバインディングを表示する際に現在のプロジェクトが使用されます。または、プロジェクトは **-n** フラグで指定できます。これは、ユーザーに **admin** の[デフォルトクラスターロール](#)がすでにある場合、別のプロジェクトのローカルのロールバインディングを表示するのに役立ちます。

ローカルのロールバインディングの表示

```
$ oc describe rolebinding.rbac -n joe-project
Name: admin
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind Name Namespace
  ---- -
  User joe

Name: system:deployers
```

```

Labels:  <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:deployer
Subjects:
  Kind    Name    Namespace
  ----    -
  ServiceAccount deployer joe-project

Name:  system:image-builders
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-builder
Subjects:
  Kind    Name    Namespace
  ----    -
  ServiceAccount builder joe-project

Name:  system:image-pullers
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind Name    Namespace
  ---- -
  Group system:serviceaccounts:joe-project

```

8.3. ロールバインディングの管理

[ロール](#)を[ユーザー](#)または[グループ](#)に追加、または[バインド](#)することにより、そのユーザーまたはグループにそのロールによって付与される関連アクセスが提供されます。**oc adm policy** コマンドを使用して、ユーザーおよびグループに対するロールの追加および削除を実行できます。

以下の操作を使用し、ローカルのロールバインディングでのユーザーまたはグループの関連付けられたロールを管理する際に、プロジェクトは **-n** フラグで指定できます。これが指定されていない場合には、現在のプロジェクトが使用されます。

表8.1 ローカルのロールバインディング操作

コマンド	説明
\$ oc adm policy who-can <verb> <resource>	リソースに対してアクションを実行できるユーザーを示します。
\$ oc adm policy add-role-to-user <role> <username>	指定されたロールを現在のプロジェクトの指定ユーザーにバインドします。

コマンド	説明
<code>\$ oc adm policy remove-role-from-user <role> <username></code>	現在のプロジェクトの指定ユーザーから指定されたロールを削除します。
<code>\$ oc adm policy remove-user <username></code>	現在のプロジェクトの指定ユーザーとそれらのロールのすべてを削除します。
<code>\$ oc adm policy add-role-to-group <role> <groupname></code>	指定されたロールを現在のプロジェクトの指定グループにバインドします。
<code>\$ oc adm policy remove-role-from-group <role> <groupname></code>	現在のプロジェクトの指定グループから指定されたロールを削除します。
<code>\$ oc adm policy remove-group <groupname></code>	現在のプロジェクトの指定グループとそれらのロールのすべてを削除します。

以下の操作を使用して、クラスターのロールバインディングも管理できます。クラスターのロールバインディングは namespace を使用していないリソースを使用するため、**-n** フラグはこれらの操作に使用されません。

表8.2 クラスターのロールバインディング操作

コマンド	説明
<code>\$ oc adm policy add-cluster-role-to-user <role> <username></code>	指定されたロールをクラスターのすべてのプロジェクトの指定ユーザーにバインドします。
<code>\$ oc adm policy remove-cluster-role-from-user <role> <username></code>	指定されたロールをクラスターのすべてのプロジェクトの指定ユーザーから削除します。
<code>\$ oc adm policy add-cluster-role-to-group <role> <groupname></code>	指定されたロールをクラスターのすべてのプロジェクトの指定グループにバインドします。
<code>\$ oc adm policy remove-cluster-role-from-group <role> <groupname></code>	指定されたロールをクラスターのすべてのプロジェクトの指定グループから削除します。

たとえば、以下を実行して **admin** ロールを **joe-project** の **alice** ユーザーに追加できます。

```
$ oc adm policy add-role-to-user admin alice -n joe-project
```

次に、ローカルのロールバインディングを表示し、出力に追加されていることを確認します。

```
$ oc describe rolebinding.rbac -n joe-project
Name: admin
Labels: <none>
Annotations: <none>
Role:
Kind: ClusterRole
```

```

Name: admin
Subjects:
  Kind Name Namespace
  ---- ----
  User joe
  User alice ❶

Name: system:deployers
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:deployer
Subjects:
  Kind Name Namespace
  ---- ----
  ServiceAccount deployer joe-project

Name: system:image-builders
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-builder
Subjects:
  Kind Name Namespace
  ---- ----
  ServiceAccount builder joe-project

Name: system:image-pullers
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind Name Namespace
  ---- ----
  Group system:serviceaccounts:joe-project

```

❶ **alice** ユーザーが **admins RoleBinding** に追加されています。

8.4. ローカルロールの作成

プロジェクトのローカルロールを作成するには、以下のコマンドを実行します。

```
$ oc create role ...
```

このコマンドの以下の **help** の抜粋は、このコマンドの使用方法を説明しています。

```
Create a role with single rule.
```


Usage:

```
oc create role NAME --verb=verb --resource=resource.group/subresource [-
-resource-name=resourcename] [--dry-run] [options]
```

Examples:

```
# Create a Role named "pod-reader" that allows user to perform "get",
"watch" and "list" on pods
```

```
oc create role pod-reader --verb=get --verb=list --verb=watch --
resource=pods
```

```
# Create a Role named "pod-reader" with ResourceName specified
oc create role pod-reader --verb=get,list,watch --resource=pods --
resource-name=readablepod --resource-name=anotherpod
```

```
# Create a Role named "foo" with API Group specified
oc create role foo --verb=get,list,watch --resource=rs.extensions
```

```
# Create a Role named "foo" with SubResource specified
oc create role foo --verb=get,list,watch --resource=pods,pods/status
```

Options:

```
--dry-run=false: If true, only print the object that would be sent,
without sending it.
```

```
--resource=[]: resource that the rule applies to
```

```
--resource-name=[]: resource in the white list that the rule applies
to, repeat this flag for multiple items
```

```
--verb=[]: verb that applies to the resources contained in the rule
```

```
...
```

たとえば、ユーザーが Pod を表示できるようにするロールを作成するには、以下を実行します。

```
$ oc create role podview --verb=get --resource=pod -n bob-project
```

オプションで、これに説明のアノテーションを付けます。

新規ロールをユーザーにバインドするには、以下を実行します。

```
$ oc adm policy add-role-to-user podview user2 --role-namespace=bob-
project -n bob-project
```

8.5. クラスターおよびローカルのロールバインディング

クラスターのロールバインディングは、クラスターレベルで存在するバインディングですが、ロールバインディングはプロジェクトレベルで存在します。クラスターの **view (表示)** ロールは、ユーザーがプロジェクトを表示できるようローカルのロールバインディングを使用してユーザーにバインドする必要があります。ローカルロールは、クラスターのロールが特定の状況に必要なパーミッションのセットを提供しない場合にのみ作成する必要があります。

一部のクラスターのロール名は最初は判別しにくい場合があります。クラスターロール **clusteradmin** は、ローカルのロールバインディングを使用してユーザーにバインドでき、このユーザーがクラスター管理者の特権を持っているように見えますが、実際にはそうではありません。一方、特定プロジェクトにバインドされる **clusteradmin** クラスターロールはそのプロジェクトのスーパー

管理者のような機能があり、クラスターロール **admin** のパーミッションを付与するほか、レート制限を編集する機能などのいくつかの追加パーミッションを付与します。これは、(実際のクラスター管理者にバインドされる) クラスターのロールバインディングを一覧表示しない Web コンソール UI を使う場合にとくに分かりにくくなりますが、この Web コンソール UI はローカルのロールバインディング (**clusteradmin** をローカルにバインドするために使用される) を一覧表示します。

第9章 イメージポリシー

9.1. 概要

インポートするイメージや、タグ付けしたり、クラスターで実行したりするイメージを制御することができます。この目的のために使用できる 2 つの機能があります。

インポート用に許可されるレジストリーは、イメージの起点 (origin) を特定の外部レジストリーのセットに制限できるイメージポリシー設定です。このルールセットはイメージストリームにインポートされるか、またはタグ付けされるすべてのイメージに適用されます。したがって、ルールセットと一致しないレジストリーを参照するイメージは拒否されます。

ImagePolicy 受付プラグイン を使用すると、クラスターでの実行を許可するイメージを指定できます。これは現時点ではベータ機能と見なされています。この機能により、以下を制御することができます。

- **イメージソース**: イメージのプルに使用できるレジストリーについての指定。
- **イメージの解決**: イメージが再タグ付けによって変更されないよう Pod のイミュータブルなダイジェストでの実行を強制する。
- **コンテナイメージラベルの制限**: イメージのラベルを制限するか、または要求する。
- **イメージアノテーションの制限**: 統合コンテナレジストリーでのイメージのアノテーションを制限するか、または要求する。

9.2. インポート用に許可されるレジストリーの設定

以下の例に示されるように、**imagePolicyConfig:allowedRegistriesForImport** セクション以下にある **master-config.yaml** にインポート用に許可されるレジスターを設定できます。この設定がない場合は、すべてのイメージが許可されます。

例9.1 インポート用に許可されるレジストリーの設定例

```
imagePolicyConfig:
  allowedRegistriesForImport:
    -
      domainName: registry.access.redhat.com ❶
    -
      domainName: *.mydomain.com
      insecure: true ❷
    -
      domainName: local.registry.corp:5000 ❸
```

- ❶ 指定されたセキュアなレジストリーからのイメージを許可します。
- ❷ **mydomain.com** の任意のサブドメインでホストされる非セキュアなレジストリーからのイメージを許可します。**mydomain.com** はホワイトリストに追加されません。
- ❸ ポートが指定された指定レジストリーからのイメージを許可します。

各ルールは以下の属性で構成されています。

- **domainName**: ホスト名であり、オプションでその最後は **:<port>** サフィックスになり、ここで特殊なワイルドカード文字 (**?**, *****) が認識されます。ワイルドカード文字は **:** 区切り文字の前後の両方に置くことができます。ワイルドカードは区切り文字の前または後の部分に適用されます。
- **insecure**: **:<port>** の部分が **domainName** にない場合、一致するポートを判別するために使用されるブール値です。true の場合、**domainName** はインポート時に非セキュアなフラグが使用されている限り、サフィックスが **:80** のポートが設定されているか、またはポートが未指定のレジストリーに一致します。false の場合、サフィックスが **:443** のポートか、またはポートが未指定のレジストリーが一致します。

ルールが同じドメインのセキュアなポートと非セキュアなポートの両方に一致する場合、ルールは 2 回一覧表示されるはずですが (1 回は **insecure=true** が設定され、もう 1 回は **insecure=false** が設定されます)。

修飾されていないイメージ参照は、ルールの評価前に **docker.io** に対して修飾されます。これらをホワイトリストに追加するには、**domainName: docker.io** を使用します。

domainName: * ルールは任意のレジストリーのホスト名に一致しますが、ポートは依然として **443** に制限された状態になります。任意のポートで機能する任意のレジストリーに一致させるには、**domainName: *:*** を使用します。

[インポート用に許可されるレジストリーの設定例](#)で設定されるルールに基づいて、以下が実行されます。

- **oc tag --insecure reg.mydomain.com/app:v1 app:v1** は、**mydomain.com** ルールの処理によってホワイトリストに追加されます。
- **oc import-image --from reg1.mydomain.com:80/foo foo:latest** もホワイトリストに追加されます。
- **oc tag local.registry.corp/bar bar:latest** は、ポートが 3 番目のルールの **5000** に一致しないために拒否されます。

拒否されたイメージのインポートにより、以下のテキストのようなエラーメッセージが生成されます。

```
The ImageStream "bar" is invalid:
* spec.tags[latest].from.name: Forbidden: registry "local.registry.corp"
not allowed by whitelist: "local.registry.corp:5000", "*.mydomain.com:80",
"registry.access.redhat.com:443"
* status.tags[latest].items[0].dockerImageReference: Forbidden: registry
"local.registry.corp" not allowed by whitelist:
"local.registry.corp:5000", "*.mydomain.com:80",
"registry.access.redhat.com:443"
```

9.3. IMAGEPOLICY 受付プラグインの設定

この機能を有効にするには、**master-config.yaml** でプラグインを設定します。

例9.2 アノテーション付きのサンプルファイル

```
admissionConfig:
  pluginConfig:
    openshift.io/ImagePolicy:
```

```

configuration:
  kind: ImagePolicyConfig
  apiVersion: v1
  resolveImages: AttemptRewrite ❶
  executionRules: ❷
    - name: execution-denied
      # Reject all images that have the annotation
      images.openshift.io/deny-execution set to true.
      # This annotation may be set by infrastructure that wishes to
      flag particular images as dangerous
      onResources: ❸
        - resource: pods
        - resource: builds
      reject: true ❹
      matchImageAnnotations: ❺
        - key: images.openshift.io/deny-execution
          value: "true"
      skipOnResolutionFailure: true ❻
    - name: allow-images-from-internal-registry
      # allows images from the internal registry and tries to
      resolve them
      onResources:
        - resource: pods
        - resource: builds
      matchIntegratedRegistry: true
    - name: allow-images-from-dockerhub
      onResources:
        - resource: pods
        - resource: builds
      matchRegistries:
        - docker.io
      resolutionRules: ❷
        - targetResource:
            resource: pods
            localNames: true
        - targetResource: ❸
            group: batch
            resource: jobs
            localNames: true ❹

```

- ❶ イミュータブルなイメージダイジェストを使用してイメージを解決し、Pod でイメージのプル仕様を更新します。
- ❷ 着信リソースに対して評価する規則の配列です。reject==true ルールのみがある場合、デフォルトは **allow all** になります。accept ルールがある場合のデフォルトは **deny all** になります。
- ❸ ルールを実施するリソースを示します。何も指定されていない場合、デフォルトは **pods** になります。
- ❹ この規則が一致する場合、Pod は拒否されることを示します。
- ❺ イメージオブジェクトのメタデータで一致するアノテーションの一覧。
- ❻ イメージを解決できない場合に Pod を失敗させません。

- 7 Kubernetes リソースでのイメージストリームの使用を許可するルールの配列。デフォルト設定は、pods、replicationcontrollers、replicasets、statefulsets、daemonsets、deployments および jobs がイメージフィールドで同じプロジェクトイメージストリームのタグ参照を使用することを許可します。
- 8 このルールが適用されるグループおよびリソースを特定します。リソースが * の場合、このルールはそのグループのすべてのリソースに適用されます。
- 9 **LocalNames** は、単一のセグメント名 (例: **ruby:2.4**) が、リソースまたはターゲットイメージストリームで **local name resolution** が有効にされている場合にのみ namespace のローカルイメージストリームとして解釈されるようにします。

注記

デフォルトのレジストリープレフィックス (**docker.io** または **registry.access.redhat.com** など) を使用してプルされるインフラストラクチャーイメージを使用する場合、レジストリープレフィックスがないイメージは **matchRegistries** 値には一致しません。インフラストラクチャーイメージにイメージポリシーに一致するレジストリープレフィックスを持たせるには、**master-config.yaml** ファイルに **imageConfig.format** 値を設定します。

9.4. IMAGEPOLICY 受付プラグインのテスト

1. **openshift/image-policy-check** を使用して設定をテストします。
たとえば、上記の情報を使用して、以下のようにテストします。

```
oc import-image openshift/image-policy-check:latest --confirm
```

2. この YAML を使用して Pod を作成します。Pod が作成されるはずですが。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: docker.io/openshift/image-policy-check:latest
    name: first
```

3. 別のレジストリーを参照する別の Pod を作成します。Pod は拒否されます。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: different-registry/openshift/image-policy-check:latest
    name: first
```

4. インポートされたイメージを使用して内部レジストリーを参照する Pod を作成します。Pod は作成され、イメージ仕様を確認すると、タグの位置にダイジェストが表示されます。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-
    check:latest
    name: first
```

5. インポートされたイメージを使用して内部レジストリーを参照する Pod を作成します。Pod は作成され、イメージ仕様を確認すると、タグが変更されていないことを確認できます。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-
    check:v1
    name: first
```

6. **oc get istag/image-policy-check:latest** からダイジェストを取得し、これを **oc annotate images/<digest> images.openshift.io/deny-execution=true** に使用します。以下は例になります。

```
$ oc annotate
images/sha256:09ce3d8b5b63595ffca6636c7daefb1a615a7c0e3f8ea68e5db044
a9340d6ba8 images.openshift.io/deny-execution=true
```

7. この Pod を再作成します。Pod は拒否されます。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: test-pod
spec:
  containers:
  - image: <internal registry IP>:5000/<namespace>/image-policy-
    check:latest
    name: first
```

第10章 イメージの署名

10.1. 概要

Red Hat Enterprise Linux (RHEL) システムでのコンテナイメージの署名により、以下を実行できます。

- コンテナイメージの起点の検証
- イメージが改ざんされていないことの確認
- ホストにプルできる検証済みイメージを判別するポリシーの設定

RHEL システムでのコンテナイメージの署名についてのアーキテクチャーの詳細は、「[Container Image Signing Integration Guide](#)」を参照してください。

OpenShift Container レジストリーは、REST API 経由で署名を保存する機能を提供します。**oc** CLI を使用して、検証済みのイメージを web コンソールまたは CLI に表示し、イメージの署名を検証することができます。



注記

イメージ署名の保存についての初期サポートは OpenShift Container Platform 3.3 で追加されています。イメージ署名の検証についての初期サポートは OpenShift Container Platform 3.6 で追加されています。

10.2. ATOMIC CLI を使用したイメージの署名

OpenShift Container Platform はイメージの署名を自動化しません。署名には、通常はワークステーションに安全に保存される開発者のプライベート GPG キーが必要になります。本書では、このワークフローについて説明します。

atomic コマンドラインインターフェース (CLI)(バージョン 1.12.5 以降) は、OpenShift Container レジストリーにプッシュできるコンテナイメージに署名するためのコマンドを提供します。**atomic** CLI は、Red Hat ベースのディストリビューション (RHEL、Centos、および Fedora) で利用できます。**atomic** CLI は RHEL Atomic Host システムには事前にインストールされます。**atomic** パッケージの RHEL ホストへのインストールについての詳細は、「[イメージ署名サポートの有効化](#)」を参照してください。



重要

atomic CLI は、**oc login** で認証された証明書を使用します。**atomic** および **oc** コマンドの両方で同じホストの同じユーザーを使用するようにしてください。たとえば、**atomic** CLI を **sudo** として使用する場合、OpenShift Container Platform に **sudo oc login** を使用してログインします。

署名をイメージに割り当てるには、ユーザーに **image-signer** クラスターロールがなければなりません。クラスター管理者は以下を使用してこれを追加できます。

```
$ oc adm policy add-cluster-role-to-user system:image-signer <user_name>
```

イメージにはプッシュ時に署名できます。

■


```
$ atomic push [--sign-by <gpg_key_id>] --type atomic <image>
```

署名は、**atomic** トランスポートタイプの引数が指定される際に OpenShift Container Platform に保存されます。詳細は、「[Signature Transports](#)」を参照してください。

atomic CLI を使用してイメージをセットアップし、実行する方法についての詳細は、「[RHEL Atomic Host Managing Containers: Signing Container Images](#)」ドキュメントか、または **atomic push --help** 出力で引数の詳細を参照してください。

atomic CLI および OpenShift Container レジストリーの使用についてのワークフローの特定の例については、「[Container Image Signing Integration Guide](#)」で説明されています。

10.3. OPENSIFT CLI を使用したイメージ署名の検証

oc adm verify-image-signature コマンドを使用して、OpenShift Container レジストリーにインポートされたイメージの署名を検証できます。このコマンドは、イメージ署名に含まれるイメージ ID が信頼できるかどうかを検証します。ここでは、パブリック GPG キーを使用して署名自体を検証し、提供される予想 ID と指定イメージの ID (プル仕様) のマッチングが行われます。

デフォルトで、このコマンドは通常 **\$GNUPGHOME/pubring.gpg** にあるパブリック GPG キーリングをパス **~/.gnupg** で使用します。デフォルトで、このコマンドは検証結果をイメージオブジェクトに保存し直すことはありません。これを実行するには、以下に示すように **--save** フラグを指定する必要があります。



注記

イメージの署名を検証するには、ユーザーに **image-auditor** クラスターロールがなければなりません。クラスター管理者は、以下を使用してこれを追加できます。

```
$ oc adm policy add-cluster-role-to-user system:image-auditor
<user_name>
```

検証済みのイメージで無効な GPG キーまたは無効な予想 ID と共に **--save** フラグを使用すると、保存された検証ステータスが削除され、イメージは未検証の状態になります。

イメージ署名を検証するには、以下の形式を使用します。

```
$ oc adm verify-image-signature <image> --expected-identity=<pull_spec> [-save] [options]
```

<pull_spec> はイメージストリームを記述して確認でき、**<image>** はイメージストリームタグを記述して確認することができます。以下のコマンド出力例を参照してください。

イメージ署名の検証例

```
$ oc describe is nodejs -n openshift
Name:                nodejs
Namespace:           openshift
Created:             2 weeks ago
Labels:              <none>
Annotations:         openshift.io/display-name=Node.js
                    openshift.io/image.dockerRepositoryCheck=2017-07-05T18:24:01Z
```

```
Docker Pull Spec: 172.30.1.1:5000/openshift/nodejs
...

$ oc describe istag nodejs:latest -n openshift
Image Name:
sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288
...

$ oc adm verify-image-signature \

sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288 \
--expected-identity 172.30.1.1:5000/openshift/nodejs:latest \
--public-key /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release \
--save
```

10.4. レジストリー API の使用によるイメージ署名へのアクセス

OpenShift Container レジストリーは、イメージ署名の書き込みおよび読み取りを実行できる **extensions** エンドポイントを提供します。イメージ署名は、Docker レジストリー API 経由で OpenShift Container Platform の KVS (key-value store) に保存されます。



注記

このエンドポイントは実験段階にあり、アップストリームの Docker レジストリープロジェクトではサポートされていません。Docker レジストリー API の一般的な情報については、[アップストリーム API のドキュメント](#)を参照してください。

10.4.1. API 経由でのイメージ署名の書き込み

新規署名をイメージに追加するには、HTTP **PUT** メソッドを使用して JSON ペイロードを **extensions** エンドポイントに送信できます。

```
PUT /extensions/v2/<namespace>/<name>/signatures/<digest>
```

```
$ curl -X PUT --data @signature.json http://<user>:
<token>@<registry_endpoint>:5000/extensions/v2/<namespace>/<name>/signatures/sha256:<digest>
```

署名コンテンツを含む JSON ペイロードの構造は以下のようになります。

```
{
  "version": 2,
  "type": "atomic",
  "name":
    "sha256:4028782c08eae4a8c9a28bf661c0a8d1c2fc8e19dbaae2b018b21011197e1484@c
    ddeb7006d914716e2728000746a0b23",
  "content": "<cryptographic_signature>"
}
```

name フィールドには、**<digest>@<name>** 形式の一意の値であるイメージ署名の名前が含まれます。**<digest>** はイメージ名を表し、**<name>** は署名の名前になります。署名の名前には 32 文字の長さが必要です。**<cryptographic_signature>** は、[コンテナ/イメージライブラリー](#)で説明されている仕様に従っている必要があります。

10.4.2. API 経由でのイメージ署名の読み取り

署名済みのイメージが OpenShift Container レジストリーにすでにプッシュされていることを仮定した場合、以下のコマンドを使って署名を読み取ることができます。

```
GET /extensions/v2/<namespace>/<name>/signatures/<digest>
```

```
$ curl http://<user>:
<token>@<registry_endpoint>:5000/extensions/v2/<namespace>/<name>/signatures/sha256:<digest>
```

<namespace> は OpenShift Container Platform プロジェクト名またはレジストリーのリポジトリ名を表し、**<name>** はイメージリポジトリの名前を指します。**digest** はイメージの SHA-256 チェックサムを表します。

指定されたイメージに署名データが含まれる場合、上記のコマンド出力により、以下の JSON 応答が生成されます。

```
{
  "signatures": [
    {
      "version": 2,
      "type": "atomic",
      "name":
"sha256:4028782c08eae4a8c9a28bf661c0a8d1c2fc8e19dbaae2b018b21011197e1484@cddeb7006d914716e2728000746a0b23",
      "content": "<cryptographic_signature>"
    }
  ]
}
```

name フィールドには、**<digest>@<name>** 形式の一意の値であるイメージ署名の名前が含まれます。**<digest>** はイメージ名を表し、**<name>** は署名の名前になります。署名の名前には 32 文字の長さが必要です。**<cryptographic_signature>** は、[コンテナ/イメージライブラリー](#)で説明されている仕様に従っている必要があります。

10.4.3. 署名ストアからのイメージ署名の自動インポート

OpenShift Container Platform は、署名ストアがすべての OpenShift Container Platform マスターノードに設定されている場合に、レジストリー設定ディレクトリーを使用してイメージ署名を自動インポートします。

レジストリー設定ディレクトリーには、各種レジストリー (リモートコンテナイメージを保存するサーバー) およびそれらに保存されるコンテンツの設定が含まれます。この単一ディレクトリーを使用すると、設定がコンテナ/イメージのすべてのユーザー間で共有されるように、各コマンドのコマンドラインオプションでその設定を指定する必要がありません。

デフォルトのレジストリー設定ディレクトリーは、**/etc/containers/registries.d/default.yaml** ファイルにあります。

すべての Red Hat イメージについてイメージ署名の自動インポートを実行する設定例:

```
docker:
  registry.access.redhat.com:
```

sigstore: https://access.redhat.com/webassets/docker/content/sigstore

1

- 1 署名ストアの URL を定義します。この URL は、既存署名の読み取りに使用されます。



注記

OpenShift Container Platform によって自動的にインポートされる署名は、デフォルトで**未検証**の状態になり、イメージ管理者による検証が必要になります。

レジストリー設定ディレクトリーについての詳細は、「[Registries Configuration Directory](#)」を参照してください。

第11章 スコープ付きトークン

11.1. 概要

ユーザーは、他のエンティティーに対し、自らと同様に機能する権限を制限された方法で付与する必要があるかもしれません。たとえば、プロジェクト管理者は Pod の作成権限を委任する必要があるかもしれません。これを実行する方法の 1 つとして、スコープ付きトークンを作成することができます。

スコープ付きトークンは、指定されるユーザーを識別するが、そのスコープによって特定のアクションに制限するトークンです。現時点で、**cluster-admin** のみがスコープ付きトークンを作成できます。

11.2. 評価

スコープは、トークンの一連のスコープを **PolicyRules** のセットに変換して評価されます。次に、要求がそれらのルールに対してマッチングされます。要求属性は、追加の許可検査のために「標準」承認者に渡せるよう、スコープルールのいずれかに一致している必要があります。

11.3. ユーザースコープ

ユーザースコープでは、指定されたユーザーについての情報を取得することにフォーカスが置かれます。それらはインテントベースであるため、ルールは自動的に作成されます。

- **user:full:** ユーザーのすべてのパーミッションによる API の完全な読み取り/書き込みアクセスを許可します。
- **user:info:** ユーザー (名前、グループなど) についての情報の読み取り専用アクセスを許可します。
- **user:check-access: self-localsubjectaccessreviews** および **self-subjectaccessreviews** へのアクセスを許可します。それらは、要求オブジェクトの空のユーザーおよびグループを渡す変数です。
- **user:list-projects:** ユーザーがアクセスできるプロジェクトを一覧表示するための読み取り専用アクセスを許可します。

11.4. ロールスコープ

ロールスコープにより、namespace でフィルターされる指定ロールと同じレベルのアクセスを持たせることができます。

- **role:<cluster-role name>:<namespace or * for all>:** 指定された namespace のみにあるクラスターロール (cluster-role) で指定されるルールにスコープを制限します。



注記

注意: これは、アクセスのエスカレートを防ぎます。ロールはシークレット、ロールバインディング、およびロールなどのリソースへのアクセスを許可しますが、このスコープはそれらのリソースへのアクセスを制限するのに役立ちます。これにより、予期しないエスカレーションを防ぐことができます。**edit (編集)** などのロールはエスカレートされるロールと見なされることが多いですが、シークレットのアクセスを持つロールの場合はロールのエスカレーションが生じます。

- **role:<cluster-role name>:<namespace or * for all>:!:** bang (!) を含めることでこのスコープでアクセスのエスカレートを許可されますが、それ以外には上記の例と同様になります。

第12章 イメージのモニタリング

12.1. 概要

CLI を使用し、インスタンスでイメージをモニタリングできます。

12.2. イメージ統計の表示

OpenShift Container Platform は、管理対象のすべてのイメージの使用状況についてのいくつかの統計を表示できます。つまり、すべてのイメージは直接またはビルドによって内部レジストリーにプッシュされます。

使用状況の統計を表示するには、以下を実行します。

```
$ oc adm top images
NAME                                IMAGESTREAMTAG          PARENTS
USAGE                               METADATA               STORAGE
sha256:80c985739a78b openshift/python (3.5)
yes                                303.12MiB
sha256:64461b5111fc7 openshift/ruby (2.2)
yes                                234.33MiB
sha256:0e19a0290ddc1 test/ruby-ex (latest)      sha256:64461b5111fc71ec
Deployment: ruby-ex-1/test         yes                    150.65MiB
sha256:a968c61adad58 test/django-ex (latest)  sha256:80c985739a78b760
Deployment: django-ex-1/test       yes                    186.07MiB
```

コマンドにより、以下の情報が表示されます。

- イメージ ID
- プロジェクト、名前、および付随する **ImageStreamTag** のタグ
- イメージ ID を使用するイメージの潜在的な親
- イメージが使用されている場所についての情報
- イメージに適切な Docker メタデータ情報が含まれるかどうかを示すフラグ
- イメージのサイズ

12.3. IMAGESTREAMS 統計の表示

OpenShift Container Platform は、すべての **ImageStreams** の使用状況についてのいくつかの統計を表示できます。

使用状況の統計を表示するには、以下を実行します。

```
$ oc adm top imagestreams
NAME                                STORAGE               IMAGES  LAYERS
openshift/python                   1.21GiB              4       36
openshift/ruby                     717.76MiB            3       27
test/ruby-ex                       150.65MiB            1       10
test/django-ex                     186.07MiB            1       10
```

コマンドにより、以下の情報が表示されます。

- プロジェクトおよび **ImageStream** の名前
- 内部 [Red Hat Container レジストリー](#) に保存される **ImageStream** 全体のサイズ
- この特定の **ImageStream** が参照しているイメージの数
- **ImageStream** を構成する層の数

12.4. イメージのプルーニング

上記のコマンドで返される情報は、イメージのプルーニングを実行する際に役立ちます。

第13章 SCC (SECURITY CONTEXT CONSTRAINTS) の管理

13.1. 概要

SCC (Security Context Constraints) により、管理者は Pod のパーミッションを制御できます。この API タイプについての詳細は、[SCC \(security context constraints\)](#) アーキテクチャーのドキュメントを参照してください。SCC は、[CLI](#) を使用し、インスタンスで通常の API [オブジェクト](#) として管理できます。



注記

SCC を管理するには、[cluster-admin](#) 権限が必要です。



重要

デフォルトの SCC を変更しないでください。デフォルトの SCC をカスタマイズすると、アップグレード時に問題が生じる可能性があります。代わりに [新規 SCC を作成](#) してください。

13.2. SCC (SECURITY CONTEXT CONSTRAINTS) の一覧表示

SCC の現在の一覧を取得するには、以下を実行します。

```
$ oc get scc
```

NAME	PRIV	CAPS	SELINUX	RUNASUSER
FSGROUP	SUPGROUP	PRIORITY	READONLYROOTFS	VOLUMES
anyuid	false	[]	MustRunAs	RunAsAny
RunAsAny	RunAsAny	10	false	[configMap
downwardAPI	emptyDir	persistentVolumeClaim	secret]	
hostaccess	false	[]	MustRunAs	MustRunAsRange
MustRunAs	RunAsAny	<none>	false	[configMap
downwardAPI	emptyDir	hostPath	persistentVolumeClaim	secret]
hostmount-anyuid	false	[]	MustRunAs	RunAsAny
RunAsAny	RunAsAny	<none>	false	[configMap
downwardAPI	emptyDir	hostPath	nfs	persistentVolumeClaim
hostnetwork	false	[]	MustRunAs	MustRunAsRange
MustRunAs	MustRunAs	<none>	false	[configMap
downwardAPI	emptyDir	persistentVolumeClaim	secret]	
nonroot	false	[]	MustRunAs	MustRunAsNonRoot
RunAsAny	RunAsAny	<none>	false	[configMap
downwardAPI	emptyDir	persistentVolumeClaim	secret]	
privileged	true	[*]	RunAsAny	RunAsAny
RunAsAny	RunAsAny	<none>	false	[*]
restricted	false	[]	MustRunAs	MustRunAsRange
MustRunAs	RunAsAny	<none>	false	[configMap
downwardAPI	emptyDir	persistentVolumeClaim	secret]	

13.3. SCC (SECURITY CONTEXT CONSTRAINTS) オブジェクトの検査

特定の SCC を検査するには、`oc get`、`oc describe`、`oc export`、または `oc edit` を使します。たとえば、`restricted` SCC を検査するには、以下を実行します。

-

```
$ oc describe scc restricted
Name:      restricted
Priority:   <none>
Access:
  Users:    <none>
  Groups:    system:authenticated
Settings:
  Allow Privileged:  false
  Default Add Capabilities:  <none>
  Required Drop Capabilities:  KILL,MKNOD,SYS_CHROOT,SETUID,SETGID
  Allowed Capabilities:  <none>
  Allowed Seccomp Profiles:  <none>
  Allowed Volume Types:
configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,secret
  Allow Host Network:  false
  Allow Host Ports:    false
  Allow Host PID:      false
  Allow Host IPC:      false
  Read Only Root Filesystem:  false
  Run As User Strategy: MustRunAsRange
    UID:      <none>
    UID Range Min:  <none>
    UID Range Max:  <none>
  SELinux Context Strategy: MustRunAs
    User:      <none>
    Role:      <none>
    Type:      <none>
    Level:     <none>
  FSGroup Strategy: MustRunAs
    Ranges:    <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:    <none>
```



注記

アップグレード時にカスタマイズされた SCC を保持するには、優先順位、ユーザー、グループ、ラベル、およびアノテーション以外にはデフォルトの SCC の設定を編集しないでください。

13.4. 新規 SCC (SECURITY CONTEXT CONSTRAINTS) の作成

新規 SCC を作成するには、以下を実行します。

1. JSON または YAML ファイルで SCC を定義します。

SCC (Security Context Constraints) オブジェクトの定義

```
kind: SecurityContextConstraints
apiVersion: v1
metadata:
  name: scc-admin
allowPrivilegedContainer: true
runAsUser:
  type: RunAsAny
seLinuxContext:
```

```

    type: RunAsAny
  fsGroup:
    type: RunAsAny
  supplementalGroups:
    type: RunAsAny
  users:
  - my-admin-user
  groups:
  - my-admin-group

```

オプションとして、**requiredDropCapabilities** フィールドに必要な値を設定してドロップ機能を SCC に追加することができます。指定された機能はコンテナからドロップされることになります。たとえば、SCC を **KILL**、**MKNOD**、および **SYS_CHROOT** の必要なドロップ機能を使って作成するには、以下を SCC オブジェクトに追加します。

```

requiredDropCapabilities:
- KILL
- MKNOD
- SYS_CHROOT

```

使用できる値の一覧は、[Docker ドキュメント](#)で確認できます。

ヒント

機能は Docker に渡されるため、特殊な **ALL** 値を使用してすべての機能をドロップすることができます。

- 次に、作成するファイルを渡して **oc create** を実行します。

```

$ oc create -f scc_admin.yaml
securitycontextconstraints "scc-admin" created

```

- SCC が作成されていることを確認します。

```

$ oc get scc scc-admin
NAME          PRIV          CAPS          SELINUX        RUNASUSER      FSGROUP
SUPGROUP     PRIORITY     READONLYROOTFS  VOLUMES
scc-admin    true         []             RunAsAny       RunAsAny       RunAsAny
RunAsAny     <none>       false          [awsElasticBlockStore
azureDisk azureFile cephFS cinder configMap downwardAPI emptyDir fc
flexVolume flocker gcePersistentDisk gitRepo glusterfs iscsi nfs
persistentVolumeClaim photonPersistentDisk quobyte rbd secret
vsphere]

```

13.5. SCC (SECURITY CONTEXT CONSTRAINTS) の削除

SCC を削除するには、以下を実行します。

```

$ oc delete scc <scc_name>

```

**注記**

デフォルトの SCC を削除する場合、これは再起動時に再生成されます。

13.6. SCC (SECURITY CONTEXT CONSTRAINTS) の更新

既存 SCC を更新するには、以下を実行します。

```
$ oc edit scc <scc_name>
```

**注記**

アップグレード時にカスタマイズされた SCC を保持するには、優先順位、ユーザー、グループ以外にデフォルトの SCC の設定を編集しないでください。

13.6.1. SCC (Security Context Constraints) 設定のサンプル

明示的な runAsUser 設定がない場合

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext: ❶
  containers:
  - name: sec-ctx-demo
    image: gcr.io/google-samples/node-hello:1.0
```

- ❶ コンテナまたは Pod が実行時に使用するユーザー ID を要求しない場合、有効な UID はこの Pod を作成する SCC によって異なります。制限付き SCC はデフォルトですべての認証ユーザーに付与されるため、ほとんどの場合はすべてのユーザーおよびサービスアカウントで利用でき、使用されます。この制限付き SCC は、**securityContext.runAsUser** フィールドの使用できる値を制限し、これをデフォルトに設定するために **MustRunAsRange** ストラテジーを使用します。受付プラグインではこの範囲を指定しないため、現行プロジェクトで **openshift.io/sa.scc.uid-range** アノテーションを検索して範囲フィールドにデータを設定します。最終的にコンテナの **runAsUser** は予測が困難な範囲の最初の値と等しい値になります。予測が困難であるのはすべてのプロジェクトにはそれぞれ異なる範囲が設定されるためです。詳細は、「[Understanding Pre-allocated Values and Security Context Constraints \(事前に割り当てられた値および SCC \(Security Context Constraint\) について\)](#)」を参照してください。

明示的な runAsUser 設定がない場合

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000 ❶
```

```
containers:
  - name: sec-ctx-demo
    image: gcr.io/google-samples/node-hello:1.0
```

- 1 特定のユーザー ID を要求するコンテナまたは Pod が OpenShift Container Platform によって受け入れられるのは、サービスアカウントまたはユーザーにそのユーザー ID を許可する SCC へのアクセスが付与されている場合のみです。SCC は、任意の ID や特定の範囲内にある ID、または要求に固有のユーザー ID を許可します。

これは SELinux、fsGroup、および Supplemental Groups で機能します。詳細は、「[Volume Security \(ボリュームセキュリティ\)](#)」を参照してください。

13.7. デフォルト SCC (SECURITY CONTEXT CONSTRAINTS) の更新

デフォルト SCC は、それらが見つからない場合にはマスターの起動時に作成されます。SCC をデフォルトにリセットするか、またはアップグレード後に既存の SCC を新規のデフォルト定義に更新するには、以下を実行します。

1. リセットする SCC を削除し、マスターを再起動してその再作成を実行します。
2. `oc adm policy reconcile-sccs` コマンドを使用します。

`oc adm policy reconcile-sccs` コマンドは、すべての SCC ポリシーをデフォルト値に設定しますが、すでに設定した可能性のある追加ユーザー、グループ、ラベル、アノテーションおよび優先順位を保持します。変更される SCC を表示するには、オプションなしでコマンドを実行するか、または `-o <format>` オプションで優先する出力を指定してコマンドを実行します。

確認後は、既存 SCC のバックアップを取ってから `--confirm` オプションを使用してデータを永続化します。



注記

優先順位や許可をリセットする場合は、`--additive-only=false` オプションを使用します。



注記

SCC に優先順位、ユーザー、グループ、ラベル、またはアノテーション以外のカスタマイズ設定がある場合、これらの設定は調整時に失われます。

13.8. 使用方法

以下では、SCC を使用する一般的なシナリオおよび手順について説明します。

13.8.1. 特権付き SCC のアクセス付与

管理者が管理者グループ外のユーザーまたはグループに対して **特権付き Pod** を追加作成するためのアクセスを付与することが必要になることがあります。これを実行するには、以下を行います。

1. SCC へのアクセスを付与するユーザーまたはグループを決定します。



警告

ユーザーへのアクセス付与は、ユーザーが Pod を直接作成する場合にのみ可能です。**ほとんどの場合**、システム自体がユーザーの代わりに作成する Pod については、関連するコントローラーの作動に使用される **サービスアカウントにアクセスを付与する必要があります**。ユーザーの代わりに Pod を作成するリソースの例として、Deployments、StatefulSets、DaemonSets などが含まれます。

2. 以下を実行します。

```
$ oc adm policy add-scc-to-user <scc_name> <user_name>
$ oc adm policy add-scc-to-group <scc_name> <group_name>
```

たとえば、**e2e-user** の **特権付き SCC** へのアクセスを許可するには、以下を実行します。

```
$ oc adm policy add-scc-to-user privileged e2e-user
```

3. 特権モードを要求するようにコンテナの **SecurityContext** を変更します。

13.8.2. 特権付き SCC のサービスアカウントアクセスの付与

最初に、**サービスアカウント**を作成します。たとえば、サービスアカウント **mysvcacct** をプロジェクト **myproject** で作成するには、以下を実行します。

```
$ oc create serviceaccount mysvcacct -n myproject
```

次に、サービスアカウントを**特権付き SCC** に追加します。

```
$ oc adm policy add-scc-to-user privileged
system:serviceaccount:myproject:mysvcacct
```

その後は、リソースがサービスアカウントの代わりに作成されていることを確認します。これを実行するには、**spec.serviceAccountName** フィールドをサービスアカウント名に設定します。サービスアカウント名を空のままにすると、**デフォルト**のサービスアカウントが使用されます。

次に、少なくとも 1 つの Pod のコンテナがセキュリティコンテキストで特権モードを要求していることを確認します。

13.8.3. Dockerfile の USER によるイメージ実行の有効化

特権付き SCC へのアクセスをすべての人に与えることなく、イメージが事前割り当て UID で強制的に実行されないようにクラスターのセキュリティを緩和するには、以下を実行します。

1. すべての認証されたユーザーに **anyuid** SCC へのアクセスを付与します。

```
$ oc adm policy add-scc-to-group anyuid system:authenticated
```

**警告**

これにより、**USER** が **Dockerfile** に指定されていない場合は、イメージをルート ID で実行することができます。

13.8.4. ルートを要求するコンテナイメージの有効化

一部のコンテナイメージ (例: **postgres** および **redis**) には root アクセスが必要であり、ボリュームの保有方法についてのいくつかの予測が設定されています。これらのイメージについては、サービスアカウントを **anyuid** SCC に追加します。

```
$ oc adm policy add-scc-to-user anyuid
system:serviceaccount:myproject:mysvcacct
```

13.8.5. レジストリーでの --mount-host の使用

PersistentVolume および **PersistentVolumeClaim** オブジェクトを使用する **永続ストレージ** オブジェクトを **レジストリーのデプロイメント** に使用することが推奨されます。テストを実行中で、**oc adm registry** コマンドを **--mount-host** オプションと共に使用する必要がある場合には、まずレジストリーの新規 **サービスアカウント** を作成し、これを **特権付き SCC** に追加する必要があります。詳細の説明については、『**Administrator Guide**』を参照してください。

13.8.6. 追加機能の提供

場合によっては、Docker が追加設定なしの機能として提供していない機能がイメージで必要になることがあります。この場合、Pod 仕様で追加機能を要求することができ、これは SCC に対して検証されます。

**重要**

これによりイメージを昇格された機能を使って実行できますが、これは必要な場合にのみ実行する必要があります。追加機能を有効にするためにデフォルトの **restricted SCC** を編集することはできません。

非 root ユーザーによって使用される場合、**setcap** コマンドを使用して、追加機能を要求するファイルに該当する機能が付与されていることを確認する必要があります。たとえば、イメージの **Dockerfile** では、以下ようになります。

```
setcap cap_net_raw,cap_net_admin+p /usr/bin/ping
```

さらに機能が Docker のデフォルトとして提供されている場合には、これを要求するために Pod 仕様を変更する必要はありません。たとえば、**NET_RAW** がデフォルトで指定されており、機能がすでに **ping** で設定されている場合、**ping** を実行するのに特別な手順は必要ありません。

追加機能を提供するには、以下を実行します。

1. 新規 SCC を作成します。

2. **allowedCapabilities** フィールドを使用して許可された機能を追加します。
3. Pod の作成時に、**securityContext.capabilities.add** フィールドで機能を要求します。

13.8.7. クラスターのデフォルト動作の変更

UID の事前割り当てを実行せず、任意ユーザーによるコンテナの実行を許可し、特権付きコンテナを禁止するようクラスターを変更するには、以下を実行します。



注記

アップグレード時にカスタマイズされた SCC を保持するには、優先順位、ユーザー、グループ、ラベル、およびアノテーション以外にはデフォルトの SCC の設定を編集しないでください。

1. **restricted** SCC を編集します。

```
$ oc edit scc restricted
```
2. **runAsUser.Type** を **RunAsAny** に変更します。
3. **allowPrivilegedContainer** が **false** に設定されていることを確認します。
4. 変更を保存します。

UID を事前に割り当てないようにし、コンテナが root で実行されないようにクラスターを変更するには、以下を実行します。

1. **restricted** SCC を編集します。

```
$ oc edit scc restricted
```
2. **runAsUser.Type** を **MustRunAsNonRoot** に変更します。
3. 変更を保存します。

13.8.8. hostPath ボリュームプラグインの使用

すべての人に **特権付き** SCC へのアクセスを付与することなく、Pod で **hostPath** ボリュームプラグインを使用できるようにクラスターのセキュリティーを緩和するには、以下を実行します。

1. **restricted** SCC を編集します。

```
$ oc edit scc restricted
```
2. **allowHostDirVolumePlugin: true** を追加します。
3. 変更を保存します。

13.8.9. 受付を使用した特定 SCC の初回使用

SCC の **Priority** フィールドを設定し、受付コントローラーで SCC の並び替え順序を制御することができます。並び替えについての詳細は、「[SCC Prioritization](#)」セクションを参照してください。

13.8.10. SCC のユーザー、グループまたはプロジェクトへの追加

SCC をユーザーまたはグループに追加する前に、まず **scc-review** オプションを使用してユーザーまたはグループが Pod を作成できるかどうかをチェックできます。詳細は、「[Authorization](#)」のトピックを参照してください。

SCC はプロジェクトに直接付与されません。代わりに、サービスアカウントを SCC に追加し、Pod にサービスアカウント名を指定するか、または指定されない場合は **default** サービスアカウントを使用して実行します。

SCC をユーザーに追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-user <scc_name> <user_name>
```

SCC をサービスアカウントに追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-user <scc_name> \
  system:serviceaccount:<serviceaccount_namespace>:<serviceaccount_name>
```

現在の場所がサービスアカウントが属するプロジェクトの場合、**-z** フラグを使用し、**<serviceaccount_name>** のみを指定することができます。

```
$ oc adm policy add-scc-to-user <scc_name> -z <serviceaccount_name>
```



重要

上記の **-z** フラグについては、誤字を防ぎ、アクセスが指定されたサービスアカウントのみに付与されるため、その使用を強く推奨します。プロジェクトにいない場合は、**-n** オプションを使用して、それが適用されるプロジェクトの namespace を指定します。

SCC をグループに追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-group <scc_name> <group_name>
```

SCC を namespace のすべてのサービスアカウントに追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-group <scc_name> \
  system:serviceaccounts:<serviceaccount_namespace>
```

第14章 スケジューリング

14.1. 概要

14.1.1. 概要

Pod のスケジューリングは、クラスター内のノードへの新規 Pod の配置を決定する内部プロセスです。

スケジューラーコードは、新規 Pod の作成時にそれらを確認し、それらをホストするのに最も適したノードを識別します。次に、マスター API を使用して Pod のバインディング (Pod とノードのバインディング) を作成します。

14.1.2. デフォルトスケジューリング

OpenShift Container Platform には、ほとんどのユーザーのニーズに対応するデフォルトスケジューラーが同梱されます。デフォルトスケジューラーは、Pod に最適なノードを判別するための固有のツールおよびカスタマイズ可能なツールの両方を使用します。

デフォルトスケジューラーが Pod の配置と利用できるカスタマイズ可能なパラメーターを判別する方法についての詳細は、「[デフォルトスケジューリング](#)」を参照してください。

14.1.3. 詳細スケジューリング

新規 Pod の配置場所に対する制御を強化する必要がある場合、OpenShift Container Platform の詳細スケジューリング機能を使用すると、Pod が特定ノード上か、または特定の Pod と共に実行されることを要求する (または実行されることが優先される) よう Pod を設定することができます。また詳細設定により、Pod をノードに配置することや他の Pod と共に実行することを防ぐこともできます。

詳細スケジューリングについての詳細は、「[詳細スケジューリング](#)」を参照してください。

14.1.4. カスタムスケジューリング

OpenShift Container Platform では、Pod 仕様を編集してユーザー独自のスケジューラーまたはサードパーティーのスケジューラーを使用することもできます。

詳細は、「[カスタムスケジューラー](#)」を参照してください。

14.2. デフォルトスケジューリング

14.2.1. 概要

OpenShift Container Platform のデフォルトの Pod スケジューラーは、クラスター内のノードにおける新規 Pod の配置場所を判別します。スケジューラーは Pod からのデータを読み取り、設定されるポリシーに基づいて適切なノードを見つけようとします。これは完全に独立した機能であり、スタンドアロン/プラグ可能ソリューションです。Pod を変更することはなく、Pod を特定ノードに関連付ける Pod のバインディングのみを作成します。

14.2.2. 汎用スケジューラー

既存の汎用スケジューラーはプラットフォームで提供されるデフォルトのスケジューラー エンジン であり、Pod をホストするノードを 3 つの手順で選択します。

1. スケジューラーは [述語](#)を使用して不適切なノードをフィルターに掛けて除外します。
2. スケジューラーは [ノードのフィルター](#)された一覧の優先順位付けを行います。
3. スケジューラーは、Pod の最も優先順位の高い Pod を選択します。

14.2.3. ノードのフィルター

利用可能なノードは、指定される制約や要件に基づいてフィルターされます。フィルターは、各ノードで [述語](#) というフィルター関数の一覧を使用して実行されます。

14.2.3.1. フィルターされたノード一覧の優先順位付け

優先順位付けは、各ノードに一連の[優先度関数](#)を実行することによって行われます。この関数は 0 -10 までのスコアをノードに割り当て、0 は不適切であることを示し、10 は Pod のホストに適していることを示します。スケジューラー設定は、それぞれの優先度関数について単純な **重み** (正の数値) を取ることができます。各優先度関数で指定されるノードのスコアは重み (ほとんどの優先度のデフォルトの重みは 1) で乗算され、すべての優先度で指定されるそれぞれのノードのスコアを追加して組み合わせられます。この重み属性は、一部の優先度により重きを置くようにするなどのために管理者によって使用されます。

14.2.3.2. 最適ノードの選択

ノードの並び替えはそれらのスコアに基づいて行われ、最高のスコアを持つノードが Post をホストするように選択されます。複数のノードに同じ高スコアが付けられている場合、それらのいずれかがランダムに選択されます。

14.2.4. スケジューラーポリシー

[述語](#)と[優先度](#)の選択によって、スケジューラーのポリシーが定義されます。

スケジューラー設定ファイルは、スケジューラーが反映する述語と優先度を指定する JSON ファイルです。

スケジューラーポリシーファイルがない場合、デフォルトの設定ファイル `/etc/origin/master/scheduler.json` が適用されます。



重要

スケジューラー設定ファイルで定義される述語および優先度は、デフォルトのスケジューラーポリシーを完全に上書きします。デフォルトの述語および優先度のいずれが必要な場合、スケジューラー設定ファイルにその関数を明示的に指定する必要があります。

デフォルトのスケジューラー設定ファイル

```
{
  "apiVersion": "v1",
  "kind": "Policy",
  "predicates": [
    {
      "name": "NoVolumeZoneConflict"
    },
    {
```

```

        "name": "MaxEBSVolumeCount"
      },
      {
        "name": "MaxGCEPDVolumeCount"
      },
      {
        "name": "MaxAzureDiskVolumeCount"
      },
      {
        "name": "MatchInterPodAffinity"
      },
      {
        "name": "NoDiskConflict"
      },
      {
        "name": "GeneralPredicates"
      },
      {
        "name": "PodToleratesNodeTaints"
      },
      {
        "name": "CheckNodeMemoryPressure"
      },
      {
        "name": "CheckNodeDiskPressure"
      },
      {
        "argument": {
          "serviceAffinity": {
            "labels": [
              "region"
            ]
          }
        },
        "name": "Region"
      }
    ],
    "priorities": [
      {
        "name": "SelectorSpreadPriority",
        "weight": 1
      },
      {
        "name": "InterPodAffinityPriority",
        "weight": 1
      },
      {
        "name": "LeastRequestedPriority",
        "weight": 1
      },
      {
        "name": "BalancedResourceAllocation",
        "weight": 1
      }
    ]
  }

```

```

        "name": "NodePreferAvoidPodsPriority",
        "weight": 10000
    },
    {
        "name": "NodeAffinityPriority",
        "weight": 1
    },
    {
        "name": "TaintTolerationPriority",
        "weight": 1
    },
    {
        "argument": {
            "serviceAntiAffinity": {
                "label": "zone"
            }
        },
        "name": "Zone",
        "weight": 2
    }
]
}

```

14.2.4.1. スケジューラーポリシーの変更

デフォルトで、スケジューラーポリシーは[マスター設定ファイル](#)の **kubernetesMasterConfig.schedulerConfigFile** フィールドで上書きされない限り、`/etc/origin/master/scheduler.json` というマスターのファイルに定義されます。

変更されたスケジューラー設定ファイルのサンプル

```

kind: "Policy"
version: "v1"
"predicates": [
    {
        "name": "PodFitsResources"
    },
    {
        "name": "NoDiskConflict"
    },
    {
        "name": "MatchNodeSelector"
    },
    {
        "name": "HostName"
    },
    {
        "argument": {
            "serviceAffinity": {
                "labels": [
                    "region"
                ]
            }
        },
        "name": "Region"
    }
]

```

```

    }
  ],
  "priorities": [
    {
      "name": "LeastRequestedPriority",
      "weight": 1
    },
    {
      "name": "BalancedResourceAllocation",
      "weight": 1
    },
    {
      "name": "ServiceSpreadingPriority",
      "weight": 1
    },
    {
      "argument": {
        "serviceAntiAffinity": {
          "label": "zone"
        }
      },
      "name": "Zone",
      "weight": 2
    }
  ]
}
]

```

スケジューラーポリシーを変更するには、以下を実行します。

1. 必要な[デフォルトの述語および優先度](#)を設定するためにスケジューラー設定ファイルを編集します。カスタム設定を作成したり、[サンプルのポリシー設定](#)のいずれかを使用または変更したりすることができます。
2. 必要な[設定可能な述語](#)と[設定可能な優先度](#)を追加します。
3. 変更を有効にするために OpenShift Container Platform を再起動します。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

14.2.5. 利用可能な述語

述語は、不適切なノードをフィルターに掛けるルールです。

OpenShift Container Platform には、デフォルトでいくつかの述語が提供されています。これらの述語の一部は、特定のパラメーターを指定してカスタマイズできます。複数の述語を組み合わせることでノードの追加フィルターを指定できます。

14.2.5.1. 静的な述語

これらの述語はユーザーから設定パラメーターまたは入力を取りません。これらはそれぞれの正確な名前を使用してスケジューラー設定に指定されます。

14.2.5.1.1. デフォルトの述語

デフォルトのスケジューラーポリシーには以下の述語が含まれます。

NoVolumeZoneConflict は Pod が要求するボリュームがゾーンで利用可能であることを確認します。

```
{"name" : "NoVolumeZoneConflict"}
```

MaxEBSVolumeCount は、AWS インスタンスに割り当てることのできるボリュームの最大数を確認します。

```
{"name" : "MaxEBSVolumeCount"}
```

MaxGCEPDVolumeCount は、Google Compute Engine (GCE) 永続ディスク (PD) の最大数を確認します。

```
{"name" : "MaxGCEPDVolumeCount"}
```

MatchInterPodAffinity は、Pod のアフィニティー/非アフィニティールールが Pod を許可するかどうかを確認します。

```
{"name" : "MatchInterPodAffinity"}
```

NoDiskConflict は Pod が要求するボリュームが利用可能であるかどうかを確認します。

```
{"name" : "NoDiskConflict"}
```

PodToleratesNodeTaints は Pod がノードのテイントを許容できるかどうかを確認します。

```
{"name" : "PodToleratesNodeTaints"}
```

CheckNodeMemoryPressure は、Pod がメモリー不足 (memory pressure) 状況にあるノードでスケジュールできるかどうかを確認します。

```
{"name" : "CheckNodeMemoryPressure"}
```

14.2.5.1.2. 他の静的な述語

OpenShift Container Platform は以下の述語もサポートしています。

CheckNodeDiskPressure は、Pod がディスク不足 (disk pressure) の状況にあるノードでスケジュールできるかどうかを確認します。

```
{"name" : "CheckNodeDiskPressure"}
```

CheckVolumeBinding は、バインドされている PVC とバインドされていない PVC の両方の場合に Pod が要求するボリュームに基づいて適しているかどうかを評価します* バインドされている PVC については、述語は対応する PV のノードアフィニティーが指定ノードによって満たされていることを確認します。* バインドされていない PVC については、述語は PVC 要件を満たす PV を検索し、PV のノードアフィニティーが指定ノードによって満たされていることを確認します。

述語は、すべてのバインドされる PVC にノードと互換性のある PV がある場合や、すべてのバインドされていない PVC が利用可能なノードと互換性のある PV に一致する場合に true を返します。

```
{"name" : "CheckVolumeBinding"}
```

CheckVolumeBinding 述語は、デフォルト以外のスケジューラーで有効にする必要があります。

CheckNodeCondition は Pod をノードでスケジュールできるかどうかを確認し、**out of disk** (ディスク不足)、**network unavailable** (ネットワークが使用不可)、または **not ready** (準備できていない) 状態を報告します。

```
{"name" : "CheckNodeCondition"}
```

PodToleratesNodeNoExecuteTaints は、Pod がノードの **NoExecute** テイントを容認できるかどうかを確認します。

```
{"name" : "PodToleratesNodeNoExecuteTaints"}
```

CheckNodeLabelPresence は、すべての指定されたラベルがノードに存在するかどうかを確認します (その値が何であるかを問わない)。

```
{"name" : "CheckNodeLabelPresence"}
```

checkServiceAffinity は、ServiceAffinity ラベルがノードでスケジュールされる Pod について同種のものであることを確認します。

```
{"name" : "checkServiceAffinity"}
```

MaxAzureDiskVolumeCount は Azure ディスクボリュームの最大数を確認します。

```
{"name" : "MaxAzureDiskVolumeCount"}
```

14.2.5.2. 汎用的な述語

以下の汎用的な述語は、非クリティカル述語とクリティカル述語が渡されるかどうかを確認します。非クリティカル述語は、非クリティカル Pod のみが渡す必要のある述語であり、クリティカル述語はすべての Pod が渡す必要のある述語です。

デフォルトのスケジューラーポリシーにはこの汎用的な述語が含まれます。

汎用的な非クリティカル述語

PodFitsResources は、リソースの可用性 (CPU、メモリー、GPU など) に基づいて適切な候補を判別します。ノードはそれらのリソース容量を宣言し、Pod は要求するリソースを指定できます。使用されるリソースではなく、要求されるリソースに基づいて適切な候補が判別されます。

```
{"name" : "PodFitsResources"}
```

汎用的なクリティカル述語

PodFitsHostPorts は、ノードに要求される Pod ポートの空きポートがある (ポートの競合がない) かどうかを判別します。

```
{"name" : "PodFitsHostPorts"}
```

HostName は、ホストパラメーターの有無と文字列のホスト名との一致に基づいて適切なノードを判別します。

```
{"name" : "HostName"}
```


MatchNodeSelector は、Pod で定義される **ノードセレクトア (nodeSelector)** のクエリーに基づいて適したノードを判別します。

```
{"name" : "MatchNodeSelector"}
```

14.2.5.3. 設定可能な述語

これらの述語はスケジューラ設定 **/etc/origin/master/scheduler.json** (デフォルト) に設定し、述語の機能に影響を与えるラベルを追加することができます。

これらは設定可能であるため、ユーザー定義の名前が異なる限り、同じタイプ (ただし設定パラメータは異なる) の複数の述語を組み合わせることができます。

これらの優先度の使用方法についての情報は、「[スケジューラポリシーの変更](#)」を参照してください。

ServiceAffinity は、Pod で実行されるサービスに基づいて Pod をノードに配置します。同じノードまたは併置されているノードに同じサービスの複数の Pod を配置すると、効率が向上する可能性があります。

この述語は **ノードセレクトア** の特定ラベルを持つ Pod を同じラベルを持つノードに配置しようとしています。

Pod がノードセレクトアでラベルを指定していない場合、最初の Pod は可用性に基づいて任意のノードに配置され、該当サービスの後続のすべての Pod はそのノードと同じラベルの値を持つノードにスケジュールされます。

```
"predicates":[
  {
    "name": "<name>", ❶
    "weight" : "1" ❷
    "argument":{
      "serviceAffinity":{
        "labels":[
          "<label>" ❸
        ]
      }
    }
  }
],
```

- ❶ 述語の名前を指定します。
- ❷ 1 (不適切) から 10 (最適) までの重みを指定します。
- ❸ マッチングに使用するラベルを指定します。以下は例になります。

```
"name": "ZoneAffinity",
"weight" : "1"
"argument":{
  "serviceAffinity":{
    "labels":[
      "rack"
```

たとえば、ノードセクター **rack** を持つサービスの最初の Pod がラベル **region=rack** を持つノードにスケジュールされている場合、同じサービスに属するその他すべての後続の Pod は同じ **region=rack** ラベルを持つノードにスケジュールされます。詳細は、「[Pod 配置の制御](#)」を参照してください。

複数レベルのラベルもサポートされています。ユーザーは同じリージョン内および (リージョン下の) 同じゾーン内のノードでスケジュールされるようサービスのすべての Pod を指定することもできます。

LabelsPresence は、値の種類を問わず、特定のノードに特定のラベルが定義されているかどうかを確認します。ラベルによるマッチングは、ノードに物理的な場所やステータスがラベルで定義されている場合などに役立ちます。

```
"predicates":[
  {
    "name": "<name>", ①
    "weight" : "1" ②
    "argument":{
      "labelsPresence":{
        "labels":[
          "<label>" ③
          presence: true/false
        ]
      }
    }
  }
],
```

- ① 述語の名前を指定します。
- ② 1 (不適切) から 10 (最適) までの重みを指定します。
- ③ マッチングに使用するラベルを指定します。

ラベルが必須かどうかを指定します。

- **presence:false** の場合、要求されるラベルのいずれかがノードラベルにある場合、Pod をスケジュールすることはできません。ラベルが存在しない場合は Pod をスケジュールできます。
- **presence:true** の場合、要求されるラベルのすべてがノードラベルにある場合、Pod をスケジュールできます。ラベルのすべてが存在しない場合、Pod はスケジュールされません。

以下に例を示します。

```
"name": "RackPreferred",
"weight" : "1"
"argument":{
  "labelsPresence":{
    "labels":[
      "rack"
    ]
  }
  "labelsPresence":{
    "labels":[
      - "region"
    ]
    presence: true
  }
}
```

14.2.6. 利用可能な優先度

優先度は、設定に応じて残りのノードにランクを付けるルールです。

優先度のカスタムセットは、スケジューラーを設定するために指定できます。OpenShift Container Platform ではデフォルトでいくつかの優先度があります。他の優先度は、特定のパラメーターを指定してカスタマイズできます。優先順位に影響を与えるために、複数の優先度を組み合わせ、異なる重みをそれぞれのノードに指定することができます。

14.2.6.1. 静的優先度

静的優先度は、重みを除き、ユーザーからいずれの設定パラメーターも取りません。重みは指定する必要があり、0 または負の値にすることはできません。

これらはスケジューラー設定 `/etc/origin/master/scheduler.json` (デフォルト) に指定されます。

14.2.6.1.1. デフォルトの優先度

デフォルトのスケジューラーポリシーには、以下の優先度が含まれています。それぞれの優先度関数は、重み **10000** を持つ **NodePreferAvoidPodsPriority** 以外は重み **1** を持ちます。

SelectorSpreadPriority は、Pod に一致するサービス、レプリケーションコントローラー (RC)、レプリケーションセット (RS)、およびステータスフルなセットを検索し、次にそれらのセクターに一致する既存の Pod を検索します。スケジューラーは、一致する既存 Pod が少ないノードを優先し、Pod のスケジュール時にそれらのセクターに一致する Pod 数の最も少ないノードで Pod をスケジュールします。

```
{"name" : "SelectorSpreadPriority", "weight" : 1}
```

InterPodAffinityPriority は、ノードの対応する PodAffinityTerm が満たされている場合に **weightedPodAffinityTerm** 要素を使った繰り返し処理や **重み** の合計への追加によって合計を計算します。合計値の最も高いノードが最も優先されます。

```
{"name" : "InterPodAffinityPriority", "weight" : 1}
```

LeastRequestedPriority は要求されたリソースの少ないノードを優先します。これは、ノードでスケジュールされる Pod によって要求されるメモリーおよび CPU のパーセンテージを計算し、利用可能な/残りの容量の値の最も高いノードを優先します。

```
{"name" : "LeastRequestedPriority", "weight" : 1}
```

BalancedResourceAllocation は、均衡が図られたリソース使用率に基づいてノードを優先します。これは、容量の一部として消費済み CPU とメモリー間の差異を計算し、2つのメトリクスがどの程度相互に近似しているかに基づいてノードの優先度を決定します。これは常に **LeastRequestedPriority** と併用する必要があります。

```
{"name" : "BalancedResourceAllocation", "weight" : 1}
```

NodePreferAvoidPodsPriority は、レプリケーションコントローラー以外のコントローラーによって所有される Pod を無視します。

```
{"name" : "NodePreferAvoidPodsPriority", "weight" : 10000}
```

NodeAffinityPriority は、ノードアフィニティーのスケジューリング設定に応じてノードの優先順位を決定します。

```
{ "name" : "NodeAffinityPriority", "weight" : 1 }
```

TaintTolerationPriority は、Pod についての **容認不可能な** テイント数の少ないノードを優先します。容認不可能なテイントとはキー **PreferNoSchedule** のあるテイントのことです。

```
{ "name" : "TaintTolerationPriority", "weight" : 1 }
```

14.2.6.1.2. 他の静的優先度

OpenShift Container Platform は以下の優先度もサポートしています。

EqualPriority は、優先度の設定が指定されていない場合に、すべてのノードに等しい重み **1** を指定します。この優先順位はテスト環境にのみ使用することを推奨します。

```
{ "name" : "EqualPriority", "weight" : 1 }
```

MostRequestedPriority は、要求されたリソースの最も多いノードを優先します。これは、ノードスケジューリングされる Pod で要求されるメモリーおよび CPU のパーセンテージを計算し、容量に対して要求される部分の平均の最大値に基づいて優先度を決定します。

```
{ "name" : "MostRequestedPriority", "weight" : 1 }
```

ImageLocalityPriority は、Pod コンテナのイメージをすでに要求しているノードを優先します。

```
{ "name" : "ImageLocalityPriority", "weight" : 1 }
```

ServiceSpreadingPriority は、同じマシンに置かれる同じサービスに属する Pod 数を最小限にすることにより Pod を分散します。

```
{ "name" : "ServiceSpreadingPriority", "weight" : 1 }
```

14.2.6.2. 設定可能な優先度

これらの優先度は、デフォルトでスケジューラー設定 `/etc/origin/master/scheduler.json` で設定し、これらの優先度に影響を与えるラベルを追加できます。

優先度関数のタイプは、それらが取る引数によって識別されます。これらは設定可能なため、ユーザー定義の名前が異なる場合に、同じタイプの (ただし設定パラメーターは異なる) 設定可能な複数の優先度を組み合わせることができます。

これらの優先度の使用方法についての情報は、「[スケジューラーポリシーの変更](#)」を参照してください。

ServiceAntiAffinity はラベルを取り、ラベルの値に基づいてノードのグループ全体に同じサービスに属する Pod を適正に分散します。これは、指定されたラベルの同じ値を持つすべてのノードに同じスコアを付与します。また Pod が最も集中していないグループ内のノードにより高いスコアを付与します。

```
"priorities":[
  {
```

```

    "name": "<name>", ❶
    "weight" : "1" ❷
    "argument":{
      "serviceAntiAffinity":{
        "labels":[
          "<label>" ❸
        ]
      }
    }
  }
}
]

```

- ❶ 優先度の名前を指定します。
- ❷ 1 (不適切) から 10 (最適) までの重みを指定します。
- ❸ マッチングに使用するラベルを指定します。

以下に例を示します。

```

    "name": "RackSpread",
    "weight" : "1"
    "argument":{
      "serviceAffinity":{
        "labels":[
          "rack"
        ]
      }
    }
  }
}
]

```

LabelPreference は、その値が何であるかを問わず、特定のラベルが定義されているノードを優先します。

```

"predicates":[
  {
    "name": "<name>", ❶
    "weight" : "1" ❷
    "argument":{
      "labelsPresence":{
        "labels":[
          "<label>" ❸
        ]
        presence: true/false
      }
    }
  }
]

```

- ❶ 優先度の名前を指定します。
- ❷ 1 (不適切) から 10 (最適) までの重みを指定します。
- ❸ マッチングに使用するラベルを指定します。

ラベルが必須かどうかを指定します。

- **presence:false** の場合、要求されるラベルのいずれかがノードラベルにある場合、Pod をスケジューリングすることはできません。ラベルが存在しない場合は Pod をスケジューリングできます。
- **presence:true** の場合、要求されるラベルのすべてがノードラベルにある場合、Pod をスケジューリングできます。ラベルのすべてが存在しない場合、Pod はスケジューリングされません。

以下に例を示します。

```
"name": "RackPreferred",
"weight" : "1"
"argument": {
  "labelsPresence": {
    "labels": [
      "rack"
```

14.2.7. 使用例

OpenShift Container Platform 内でのスケジューリングの重要な使用例として、柔軟なアフィニティーと非アフィニティーポリシーのサポートを挙げることができます。

14.2.7.1. インフラストラクチャーのトポロジレベル

管理者は、[ノードのラベル](#) (例: **region=r1**、**zone=z1**、**rack=s1**) を指定してインフラストラクチャーの複数のトポロジレベルを定義することができます。

これらのラベル名には特別な意味はなく、管理者はそれらのインフラストラクチャーラベルに任意の名前 (例: 都市/建物/部屋) を付けることができます。さらに、管理者はインフラストラクチャートポロジに任意の数のレベルを定義できます。通常は、(**regions** → **zones** → **racks** などの) 3 つのレベルが適切なサイズです。管理者はこれらのレベルのそれぞれにアフィニティーと非アフィニティールールを任意の組み合わせで指定することができます。

14.2.7.2. アフィニティー

管理者は、任意のトポロジレベルまたは複数のレベルでもアフィニティーを指定できるようにスケジューラーを設定することができます。特定レベルのアフィニティーは、同じサービスに属するすべての Pod が同じレベルに属するノードにスケジューリングされることを示します。これは、管理者がピア Pod が地理的に離れ過ぎないようにすることでアプリケーションの待機時間の要件に対応します。同じアフィニティーグループ内で Pod をホストするために利用できるノードがない場合、Pod はスケジューリングされません。

Pod がスケジューリングされる場所に対する制御を強化する必要がある場合は、「[ノードアフィニティーの使用](#)」および「[Pod のアフィニティーおよび非アフィニティーの使用](#)」を参照してください。これらの詳細スケジューリング機能により、管理者は Pod をスケジューリングできるノードを指定し、他の Pod に関連してスケジューリングを強制的に実行したり、拒否したりできます。

14.2.7.3. 非アフィニティー

管理者は、任意のトポロジレベルまたは複数のレベルでも非アフィニティーを設定できるようにスケジューラーを設定することができます。特定レベルの非アフィニティー (または「分散」) は、同じサービスに属するすべての Pod が該当レベルに属するノード全体に分散されることを示します。これにより、アプリケーションが高可用性の目的で適正に分散されます。スケジューラーは、可能な限り均等になるようにすべての適用可能なノード全体にサービス Pod を配置しようとします。

Pod がスケジューリングされる場所に対する制御を強化する必要がある場合は、「[ノードアフィニティーの使用](#)」および「[Pod のアフィニティーおよび非アフィニティーの使用](#)」を参照してください。これらの詳細スケジューリング機能により、管理者は Pod をスケジューリングできるノードを指定し、他の Pod に関連してスケジューリングを強制的に実行したり、拒否したりできます。

14.2.8. ポリシー設定のサンプル

以下の設定は、スケジューラーポリシーファイルを使って指定される場合のデフォルトのスケジューラー設定を示しています。

```
kind: "Policy"
version: "v1"
predicates:
...
- name: "RegionZoneAffinity" ❶
  argument:
    serviceAffinity: ❷
      labels: ❸
        - "region"
        - "zone"
priorities:
...
- name: "RackSpread" ❹
  weight: 1
  argument:
    serviceAntiAffinity: ❺
      label: "rack" ❻
```

❶ 述語の名前です。

❷ 述語のタイプです。

❸ 述語のラベルです。

❹ 優先度の名前です。

❺ 優先度のタイプです。

❻ 優先度のラベルです。

以下の設定例のいずれの場合も、述語と優先度関数の一覧は、指定された使用例に関連するもののみを含むように切り捨てられます。実際には、完全な/分かりやすいスケジューラーポリシーには、上記のデフォルトの述語および優先度のほとんど（すべてではなくても）が含まれるはずです。

以下の例は、region (affinity) → zone (affinity) → rack (anti-affinity) の3つのトポロジーレベルを定義します。

```
kind: "Policy"
version: "v1"
predicates:
...
- name: "RegionZoneAffinity"
  argument:
    serviceAffinity:
```

```

      labels:
        - "region"
        - "zone"
    priorities:
    ...
    - name: "RackSpread"
      weight: 1
      argument:
        serviceAntiAffinity:
          label: "rack"

```

以下の例は、city (affinity) → building (anti-affinity) → room (anti-affinity) の3つのトポロジーレベルを定義します。

```

kind: "Policy"
version: "v1"
predicates:
...
- name: "CityAffinity"
  argument:
    serviceAffinity:
      labels:
        - "city"
priorities:
...
- name: "BuildingSpread"
  weight: 1
  argument:
    serviceAntiAffinity:
      label: "building"
- name: "RoomSpread"
  weight: 1
  argument:
    serviceAntiAffinity:
      label: "room"

```

以下の例では、「region」ラベルが定義されたノードのみを使用し、「zone」ラベルが定義されたノードを優先するポリシーを定義します。

```

kind: "Policy"
version: "v1"
predicates:
...
- name: "RequireRegion"
  argument:
    labelsPresence:
      labels:
        - "region"
      presence: true
priorities:
...
- name: "ZonePreferred"
  weight: 1
  argument:
    labelPreference:
      label: "zone"

```



```
presence: true
```

以下の例では、静的および設定可能な述語および優先度を組み合わせています。

```
kind: "Policy"
version: "v1"
predicates:
...
- name: "RegionAffinity"
  argument:
    serviceAffinity:
      labels:
        - "region"
- name: "RequireRegion"
  argument:
    labelsPresence:
      labels:
        - "region"
      presence: true
- name: "BuildingNodesAvoid"
  argument:
    labelsPresence:
      labels:
        - "building"
      presence: false
- name: "PodFitsPorts"
- name: "MatchNodeSelector"
priorities:
...
- name: "ZoneSpread"
  weight: 2
  argument:
    serviceAntiAffinity:
      label: "zone"
- name: "ZonePreferred"
  weight: 1
  argument:
    labelPreference:
      label: "zone"
      presence: true
- name: "ServiceSpreadingPriority"
  weight: 1
```

14.3. カスタムスケジューリング

14.3.1. 概要

デフォルトのスケジューラーと共に複数のカスタムスケジューラーを実行し、各 Pod に使用できるスケジューラーを設定できます。

特定のスケジューラーを使用して指定された Pod をスケジューリングするには、[Pod 仕様にスケジューラーの名前を指定します](#)。

14.3.2. スケジューラーのデプロイ

以下の手順は、スケジューラーをクラスターにデプロイするための一般的なプロセスです。



注記

スケジューラーの作成/デプロイ方法については、本書では扱いません。これらについては、[Kubernetes ソースディレクトリーの plugin/pkg/scheduler](#)などを参照してください。

1. Pod 設定を作成するか、または編集し、**schedulerName** パラメーターでスケジューラーの名前を指定します。名前は一意である必要があります。

スケジューラーを含む Pod 仕様のサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: custom-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulerName: custom-scheduler ❶
  containers:
  - name: pod-with-second-annotation-container
    image: docker.io/ocpqe/hello-pod
```

- ❶ 使用するスケジューラーの名前です。スケジューラー名が指定されていない場合、Pod はデフォルトのスケジューラーを使用して自動的にスケジュールされます。

2. 以下のコマンドを実行して Pod を作成します。

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

3. 以下のコマンドを実行し、Pod がカスタムスケジューラーで作成されていることを確認します。

```
$ oc get pod custom-scheduler -o yaml
```

4. 以下のコマンドを実行して Pod のステータスを確認します。

```
$ oc get pod
```

Pod は実行されていないはずです。

NAME	READY	STATUS	RESTARTS	AGE
custom-scheduler	0/1	Pending	0	2m

5. カスタムスケジューラーをデプロイします。
6. 以下のコマンドを実行して Pod のステータスを確認します。

```
$ oc get pod
```

Pod は実行されているはずです。

NAME	READY	STATUS	RESTARTS	AGE
custom-scheduler	1/1	Running	0	4m

7. 以下のコマンドを実行し、スケジューラーが使用されていることを確認します。

```
$ oc describe pod custom-scheduler
```

以下の切り捨てられた出力に示されるように、スケジューラーの名前が一覧表示されます。

```
[...]
Events:
  FirstSeen    LastSeen    Count   From              SubobjectPath  Type
Reason Message
-----
1m           1m           1      my-scheduler      Normal
Scheduled    Successfully assigned custom-scheduler to <$node1>
[...]
```

14.4. POD 配置の制御

14.4.1. 概要

クラスター管理者は、特定のロールを持つアプリケーション開発者が Pod のスケジュール時に特定ノードをターゲットとすることを防ぐポリシーを設定できます。

Pod ノード制約の受付コントローラーは、Pod がラベルを使用して指定されたノードホストのみにデプロイされるようにし、特定のロールを持たないユーザーが **nodeSelector** フィールドを使用して Pod をスケジュールできないようにします。

14.4.2. ノード名の使用による Pod 配置の制約

Pod ノード制約の受付コントローラーを使用し、Pod にラベルを割り当て、これを Pod 設定の **nodeName** 設定に指定することで、Pod が指定されたノードホストにのみデプロイされるようにします。

1. 必要なラベル (詳細は、「[ノードでのラベルの更新](#)」を参照) および [ノードセレクター](#) が環境にセットアップされていることを確認します。
たとえば、Pod 設定が必要なラベルを示す **nodeName** 値を持つことを確認します。

```
apiVersion: v1
kind: Pod
spec:
  nodeName: <value>
```

2. マスター設定ファイル (`/etc/origin/master/master-config.yaml`) を 2 箇所を変更します。
 - a. **PodNodeConstraints** を **admissionConfig** セクションに追加します。

```
...
admissionConfig:
```

```

    pluginConfig:
      PodNodeConstraints:
        configuration:
          apiVersion: v1
          kind: PodNodeConstraintsConfig
    ...

```

b. 次に、同じ設定を **kubernetesMasterConfig** セクションに追加します。

```

    ...
    kubernetesMasterConfig:
      admissionConfig:
        pluginConfig:
          PodNodeConstraints:
            configuration:
              apiVersion: v1
              kind: PodNodeConstraintsConfig
    ...

```

3. 変更を有効にするために OpenShift Container Platform を再起動します。

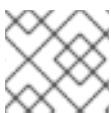
```
#systemctl restart atomic-openshift-master
```

14.4.3. ノードセクターの使用による Pod 配置の制約

ノードセクターを使用して、Pod が特定のラベルを持つノードにのみ配置されるようにすることができます。クラスター管理者は、Pod ノード制約の受付コントローラーを使用して、**pods/binding** パーミッションのないユーザーがノードセクターを使用して Pod をスケジュールできないようにするポリシーを設定できます。

マスター設定ファイルの **nodeSelectorLabelBlacklist** フィールドを使用して、一部のロールが Pod 設定の **nodeSelector** フィールドで指定できるラベルを制御できます。**pods/binding** パーミッション**ロール**を持つユーザー、サービスアカウントおよびグループは任意のノードセクターを指定できます。**pods/binding** パーミッションがない場合は、**nodeSelectorLabelBlacklist** に表示されるすべてのラベルに **nodeSelector** を設定することは禁止されます。

たとえば、OpenShift Container Platform クラスターは、2つの地域にまたがる5つのデータセンターで構成される場合があります。米国の "us-east"、"us-central"、および "us-west"、およびアジア太平洋地域 (APAC) の "apac-east" および "apac-west" です。それぞれの地理的地域の各ノードには、**region: us-east** などのラベルが付けられます。



注記

ラベルの割り当ての詳細は、「[ノードでのラベルの更新](#)」を参照してください。

クラスター管理者は、アプリケーション開発者が地理的に最も近い場所にあるノードにのみ Pod をデプロイできるインフラストラクチャーを作成できます。ノードセクターを作成し、米国のデータセンターを **superregion: us** に、APAC のデータセンターを **superregion: apac** に分類できます。

データセンターごとのリソースの均等なロードを維持するには、必要な **region** をマスター設定の **nodeSelectorLabelBlacklist** セクションに追加できます。その後は、米国の開発者が Pod を作成するたびに、Pod は **superregion: us** ラベルの付いた地域のいずれかにあるノードにデプロイされます。開発者が Pod に特定の region (地域) をターゲットに設定しようとする (例: **region: us-**

east)、エラーが出されます。これを Pod にノードセクターを設定せずに試行すると、ターゲットとした region (地域) にデプロイすることができます。それは **superregion: us** がプロジェクトレベルのノードセクターとして設定されており、**region: us-east** というラベルが付けられたノードには **superregion: us** というラベルも付けられているためです。

1. 必要なラベル (詳細は、「[ノードでのラベルの更新](#)」を参照) および **ノードセクター** が環境にセットアップされていることを確認します。
たとえば、Pod 設定が必要なラベルを示す **nodeSelector** 値を持つことを確認します。

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    <key>: <value>
  ...
```

2. マスター設定ファイル (**/etc/origin/master/master-config.yaml**) を 2 箇所を変更します。

- a. **nodeSelectorLabelBlacklist** を、Pod の配置を拒否する必要があるノードホストに割り当てられるラベルと共に **admissionConfig** セクションに追加します。

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiVersion: v1
        kind: PodNodeConstraintsConfig
        nodeSelectorLabelBlacklist:
          - kubernetes.io/hostname
          - <label>
  ...
```

- b. 次に、同じ設定を **kubernetesMasterConfig** セクションに追加し、Pod の直接の作成を制限します。

```
...
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      PodNodeConstraints:
        configuration:
          apiVersion: v1
          kind: PodNodeConstraintsConfig
          nodeSelectorLabelBlacklist:
            - kubernetes.io/hostname
            - <label_1>
  ...
```

3. 変更を有効にするために OpenShift Container Platform を再起動します。

```
#systemctl restart atomic-openshift-master
```

14.4.4. プロジェクト対する Pod 配置の制御

Pod ノードセクターの受付コントローラーを使用して、Pod を特定のプロジェクトに関連付けられたノードに対して強制的に適用したり、Pod がそれらのノードでスケジュールされないようにしたりできます。

Pod ノードセクター の受付コントローラーは、[プロジェクトのラベル](#)と Pod で指定されるノードセクターを使用して Pod を配置する場所を決定します。新規 Pod は、Pod のノードセクターがプロジェクトのラベルに一致する場合にのみプロジェクトに関連付けられたノードに配置されます。

Pod の作成後に、ノードセクターは Pod にマージされ、Pod 仕様に元々含まれていたラベルとノードセクターの新規ラベルが含まれるようにします。以下の例は、マージの結果について示しています。

Pod ノードセクター の受付コントローラーにより、特定のプロジェクトで許可されるラベルの一覧を作成することもできます。この一覧は開発者がプロジェクトで利用できるラベルを認識するための **ホワイトリスト** として機能し、管理者がクラスターでのラベル設定の制御を強化するのに役立ちます。

Pod ノードセクター の受付コントローラーをアクティブにするには、以下を実行します。

1. 以下の方法のいずれかを使用して **Pod ノードセクター** の受付コントローラーとホワイトリストを設定します。

- 以下をマスター設定ファイル (`/etc/origin/master/master-config.yaml`) に追加します。

```
admissionConfig:
  pluginConfig:
    PodNodeSelector:
      configuration:
        podNodeSelectorPluginConfig: ❶
          clusterDefaultNodeSelector: "k3=v3" ❷
          ns1: region=west,env=test,infra=fedora,os=fedora ❸
```

- ❶ **Pod ノードセクター** の受付コントローラープラグインを追加します。

- ❷ ❸ すべてのノードのデフォルトラベルを作成します。

指定されたプロジェクトで許可されるラベルのホワイトリストを作成します。ここで、プロジェクトは **ns1** で、ラベルはそれに続く **key=value** ペアになります。

- 受付コントローラーの情報を含むファイルを作成します。

```
podNodeSelectorPluginConfig:
  clusterDefaultNodeSelector: "k3=v3"
  ns1: region=west,env=test,infra=fedora,os=fedora
```

次に、マスター設定でファイルを参照します。

```
admissionConfig:
  pluginConfig:
    PodNodeSelector:
      location: <path-to-file>
```



注記

プロジェクトにノードセクターが指定されていない場合、そのプロジェクトに関連付けられた Pod はデフォルトのノードセクター (**clusterDefaultNodeSelector**) を使用してマージされます。

2. 変更を有効にするために OpenShift Container Platform を再起動します。

```
#systemctl restart atomic-openshift-master
```

3. **scheduler.alpha.kubernetes.io/node-selector** アノテーションおよびラベルを含むプロジェクトオブジェクトを作成します。

```
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "ns1",
    "annotations": {
      "scheduler.alpha.kubernetes.io/node-selector":
"env=test,infra=fedora" ❶
    }
  },
  "spec": {},
  "status": {}
}
```

- ❶ プロジェクトのラベルセクターに一致するラベルを作成するためのアノテーションです。ここで、キー/値のラベルは **env=test** および **infra=fedora** になります。

4. ノードセクターにラベルを含む Pod 仕様を作成します。以下は例になります。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: hello-pod
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod:latest"
      imagePullPolicy: IfNotPresent
      name: hello-pod
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      securityContext:
        capabilities: {}
        privileged: false
        terminationMessagePath: /dev/termination-log
      dnsPolicy: ClusterFirst
```

```
restartPolicy: Always
nodeSelector: ❶
  env: test
  os: fedora
serviceAccount: ""
status: {}
```

❶ プロジェクトラベルに一致するノードセレクターです。

5. プロジェクトに Pod を作成します。

```
oc create -f pod.yaml --namespace=ns1
```

6. ノードセレクターのラベルが Pod 設定に追加されていることを確認します。

```
get pod pod1 --namespace=ns1 -o json

nodeSelector": {
  "env": "test",
  "infra": "fedora",
  "os": "fedora"
}
```

ノードセレクターは Pod にマージされ、Pod は適切なプロジェクトでスケジュールされます。

プロジェクト仕様で指定されていないラベルを使って Pod を作成する場合、Pod はノードでスケジュールされません。

たとえば、ここでラベル **env: production** はいずれのプロジェクト仕様にもありません。

```
nodeSelector:
  "env: production"
  "infra": "fedora",
  "os": "fedora"
```

ノードセレクターのアノテーションのないノードがある場合は、Pod はそこにスケジュールされます。

14.5. 詳細スケジューリング

14.5.1. 概要

詳細スケジューリングには、Pod が特定ノードで実行されることを要求したり、Pod が特定ノードで実行されることが優先されるように Pod を設定することが関係します。

通常、詳細スケジューリングは必要になりません。OpenShift Container Platform が Pod を合理的な方法で自動的に配置するためです。たとえば、デフォルトスケジューラーは Pod をノード間で均等に分散し、ノードの利用可能なリソースを考慮します。ただし、Pod を配置する場所についてはさらに制御を強化する必要がある場合があります。

Pod をより高速なディスクが搭載されたマシンに配置する必要がある場合 (またはそのマシンに配置するのを防ぐ場合)、または 2 つの異なるサービスの Pod が相互に通信できるように配置する必要がある場合、詳細スケジューリングを使用してそれを可能にすることができます。

適切な新規 Pod を特定のノードグループにスケジュールし、その他の新規 Pod がそれらのノードでスケジュールされるのを防ぐには、必要に応じてこれらの方法を組み合わせることができます。

14.5.2. 詳細スケジューリングの使用

クラスターで詳細スケジューリングを起動する方法はいくつかあります。

Pod のアフィニティーおよび非アフィニティー

Pod のアフィニティーにより、**Pod** がその配置に使用できるアフィニティー (または非アフィニティー) を、(セキュリティ上の理由によるアプリケーションの待機時間の要件などのために) **Pod** のグループに対して指定できるようにします。ノード自体は配置に対する制御を行いません。

Pod のアフィニティーはノードのラベルと Pod のラベルセクターを使用して Pod 配置のルールを作成します。ルールは mandatory (必須) または best-effort (優先) のいずれかにすることができます。

「[Pod のアフィニティーおよび非アフィニティーの使用](#)」を参照してください。

ノードのアフィニティー

ノードのアフィニティーにより、**Pod** がその配置に使用できるアフィニティー (または非アフィニティー) を、(高可用性のための特殊なハードウェア、場所、要件などにより) ノードのグループに対して指定できるようにします。ノード自体は配置に対する制御を行いません。

ノードのアフィニティーはノードのラベルと Pod のラベルセクターを使用して Pod 配置のルールを作成します。ルールは mandatory (必須) または best-effort (優先) のいずれかにすることができます。

「[ノードアフィニティーの使用](#)」を参照してください。

ノードセクター

ノードセクターは詳細スケジューリングの最も単純な形態です。ノードのアフィニティーのように、ノードセクターはノードのラベルと Pod のラベルセクターを使用し、**Pod** がその配置に使用するノードを制御できるようにします。ただし、ノードセクターにはノードのアフィニティーが持つ required (必須) ルールまたは preferred (優先) ルールはありません。

「[ノードセクターの使用](#)」を参照してください。

テイントおよび容認 (Toleration)

テイント/容認により、ノードはノード上でスケジュールする必要のある (またはスケジュールすべきでない) **Pod** を制御できます。テイントはノードのラベルであり、容認は Pod のラベルです。スケジュールを可能にするには、Pod のラベルはノードのラベル (テイント) に一致する (またはこれを許容する) 必要があります。

テイント/容認にはアフィニティーと比較して 1 つ利点があります。たとえばアフィニティーの場合は、異なるラベルを持つノードの新規グループをクラスターに追加する場合、ノードにアクセスさせたい Pod とノードを使用させたくない Pod のそれぞれに対してアフィニティーを更新する必要がありますが、テイント/容認の場合には、新規ノードに到達させる必要のある Pod のみを更新すれば、他の Pod は拒否されることになります。

「[テイントおよび容認の使用](#)」を参照してください。

14.6. 詳細スケジューリングおよびノードのアフィニティー

14.6.1. 概要

ノードのアフィニティー は、Pod の配置場所を判別するためにスケジューラーによって使用されるルールセットです。ルールはカスタムの **ノードのラベル** と Pod で指定されるラベルセクターを使って定義されます。ノードのアフィニティーにより、**Pod ノード** のグループに対してはアフィニティー (または非アフィニティー) を指定できます。ノード自体は配置に対して制御を行いません。

たとえば、Pod を特定の CPU を搭載したノードまたは特定のアベイラビリティゾーンにあるノードでのみ実行されるよう設定することができます。

ノードのアフィニティールールには、**required (必須)** および **preferred (優先)** の 2 つのタイプがあります。

required (必須) ルールは、Pod をノードにスケジュールする前に **満たされている必要があります**。一方、preferred (優先) ルールは、ルールが満たされる場合にスケジューラーがルールの実施を試行しますが、その実施が必ずしも保証される訳ではありません。



注記

ランタイム時にノードのラベルに変更が生じ、その変更により Pod でのノードのアフィニティールールを満たさなくなる状態が生じるでも、Pod はノードで引き続き実行されます。

14.6.2. ノードのアフィニティーの設定

ノードのアフィニティーは、Pod 仕様で設定することができます。**required (必須) ルール**、**preferred (優先) ルール** のいずれかまたはその両方を指定することができます。両方を指定する場合、ノードは最初に required (必須) ルールを満たす必要があり、その後に preferred (優先) ルールを満たそうとします。

以下の例は、Pod をキーが **e2e-az-NorthSouth** で、その値が **e2e-az-North** または **e2e-az-South** のいずれかであるラベルの付いたノードに Pod を配置することを求めるルールが設定された Pod 仕様です。

ノードのアフィニティーの required (必須) ルールが設定された Pod 設定ファイルのサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: ❶
      requiredDuringSchedulingIgnoredDuringExecution: ❷
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-NorthSouth ❸
                operator: In ❹
                values:
                  - e2e-az-North ❺
                  - e2e-az-South ❻
  containers:
    - name: with-node-affinity
      image: docker.io/ocpqe/hello-pod
```

❶ ノードのアフィニティーを設定するためのスタンザです。

- ② required (必須) ルールを定義します。
- ③⑤⑥ ルールを適用するために一致している必要のあるキー/値のペア (ラベル)。
- ④ 演算子は、ノードのラベルと Pod 仕様の **matchExpression** パラメーターの値のセットの間の関係を表します。この値は、**In**、**NotIn**、**Exists**、または **DoesNotExist**、**Lt**、または **Gt** にすることができます。

以下の例は、キーが **e2e-az-EastWest** で、その値が **e2e-az-East** または **e2e-az-West** のラベルが付いたノードに Pod を配置すること優先する preferred (優先) ルールが設定されたノード仕様です。

ノードのアフィニティーの preferred (優先) ルールが設定された Pod 設定ファイルのサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: ①
      preferredDuringSchedulingIgnoredDuringExecution: ②
        - weight: 1 ③
          preference:
            matchExpressions:
              - key: e2e-az-EastWest ④
                operator: In ⑤
                values:
                  - e2e-az-East ⑥
                  - e2e-az-West ⑦
  containers:
    - name: with-node-affinity
      image: docker.io/ocpqe/hello-pod
```

- ① ノードのアフィニティーを設定するためのスタンザです。
- ② preferred (優先) ルールを定義します。
- ③ preferred (優先) ルールの重みを指定します。最も高い重みを持つノードが優先されます。
- ④⑥⑦ ルールを適用するために一致している必要のあるキー/値のペア (ラベル)。
- ⑤ 演算子は、ノードのラベルと Pod 仕様の **matchExpression** パラメーターの値のセットの間の関係を表します。この値は、**In**、**NotIn**、**Exists**、または **DoesNotExist**、**Lt**、または **Gt** にすることができます。

ノードの非アフィニティー についての明示的な概念はありませんが、**NotIn** または **DoesNotExist** 演算子を使用すると、動作が複製されます。

注記

同じ Pod 設定でノードのアフィニティーと [ノードセレクター](#) を使用している場合、以下に注意してください。

- **nodeSelector** と **nodeAffinity** の両方を設定する場合、Pod が候補ノードでスケジュールされるにはどちらの条件も満たしている必要があります。
- **nodeAffinity** タイプに関連付けられた複数の **nodeSelectorTerms** を指定する場合、**nodeSelectorTerms** のいずれかが満たされている場合に Pod をノードにスケジュールすることができます。
- **nodeSelectorTerms** に関連付けられた複数の **matchExpressions** を指定する場合、すべての **matchExpressions** が満たされている場合にのみ Pod をノードにスケジュールすることができます。

14.6.2.1. ノードアフィニティーの required (必須) ルールの設定

Pod がノードにスケジュールされる前に、required (必須) ルールを **満たしている必要があります**。

以下の手順は、ノードとスケジューラーがノードに配置する必要のある Pod を作成する単純な設定を示しています。

1. ノード設定を編集するか、または **oc label node** コマンドを使用してラベルをノードに追加します。

```
$ oc label node node1 e2e-az-name=e2e-az1
```

2. Pod 仕様では、**nodeAffinity** スタンザを使用して **requiredDuringSchedulingIgnoredDuringExecution** パラメーターを設定します。
 - a. 満たす必要のあるキーおよび値を指定します。新規 Pod を編集したノードにスケジュールする必要がある場合、ノードのラベルと同じ **key** および **value** パラメーターを使用します。
 - b. **operator** を指定します。演算子は **In**、**NotIn**、**Exists**、**DoesNotExist**、**Lt**、または **Gt** にすることができます。たとえば、演算子 **In** を使用してラベルがノードで必要になるようにします。

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-name
                operator: In
                values:
                  - e2e-az1
                  - e2e-az2
```

3. Pod の作成します。

```
$ oc create -f e2e-az2.yaml
```

14.6.2.2. ノードアフィニティーの **preferred** (優先) ルールの設定

preferred (優先) ルールは、ルールを満たす場合に、スケジューラーはルールの実施を試行しますが、その実施が必ずしも保証される訳ではありません。

以下の手順は、ノードとスケジューラーがノードに配置しようとする Pod を作成する単純な設定を示しています。

1. ノード設定を編集するか、または **oc label node** コマンドを実行してラベルをノードに追加します。

```
$ oc label node node1 e2e-az-name=e2e-az3
```

2. Pod 仕様では、**nodeAffinity** スタンザを使用して **preferredDuringSchedulingIgnoredDuringExecution** パラメーターを設定します。
 - a. ノードの重みを数字の 1-100 で指定します。最も高い重みを持つノードが優先されます。
 - b. 満たす必要のあるキーおよび値を指定します。新規 Pod を編集したノードにスケジュールする必要がある場合、ノードのラベルと同じ **key** および **value** パラメーターを使用します。

```
preferredDuringSchedulingIgnoredDuringExecution:
- weight: 1
  preference:
    matchExpressions:
    - key: e2e-az-name
      operator: In
      values:
      - e2e-az3
```

3. **operator** を指定します。演算子は **In**、**NotIn**、**Exists**、**DoesNotExist**、**Lt**、または **Gt** にすることができます。たとえば、演算子 **In** を使用してラベルをノードで必要になるようにします。
4. Pod を作成します。

```
$ oc create -f e2e-az3.yaml
```

14.6.3. 各種の例

以下の例は、ノードのアフィニティーを示しています。

14.6.3.1. 一致するラベルを持つノードのアフィニティー

以下の例は、一致するラベルを持つノードと Pod のノードのアフィニティーを示しています。

- **Node1** ノードにはラベル **zone:us** があります。

```
$ oc label node node1 zone=us
```

- Pod **pod-s1** にはノードアフィニティーの **required** (必須) ルールの下に **zone** と **us** のキー/値のペアがあります。

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "zone"
            operator: In
            values:
            - us
```

- 標準コマンドを使用して Pod を作成します。

```
$ oc create -f pod-s1.yaml
pod "pod-s1" created
```

- Pod **pod-s1** を **Node1** にスケジュールできます。

```
oc get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
pod-s1	1/1	Running	0	4m	IP1	node1

14.6.3.2. 一致するラベルのないノードのアフィニティー

以下の例は、一致するラベルを持たないノードと Pod のノードのアフィニティーを示しています。

- **Node1** ノードにはラベル **zone:emea** があります。

```
$ oc label node node1 zone=emea
```

- Pod **pod-s1** にはノードアフィニティーの required (必須) ルールの下に **zone** と **us** のキー/値のペアがあります。

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
```

```
- matchExpressions:
  - key: "zone"
    operator: In
    values:
      - us
```

- Pod **pod-s1** は **Node1** にスケジューリングすることができません。

```
oc describe pod pod-s1
<---snip-->
Events:
  FirstSeen LastSeen Count From                                SubObjectPath  Type
Reason
-----
1m          33s      8    default-scheduler Warning
FailedScheduling    No nodes are available that match all of the
following predicates: MatchNodeSelector (1).
```

14.7. 詳細スケジューリングおよび POD のアフィニティーと非アフィニティー

14.7.1. 概要

Pod のアフィニティー および **Pod の非アフィニティー** により、他の Pod との関連で Pod を配置する方法についてのルールを指定できます。ルールは、カスタムの [ノードのラベル](#) および Pod で指定されるラベセレクターを使用して定義されます。Pod のアフィニティー/非アフィニティーにより、**Pod** はアフィニティー (または非アフィニティー) を、その配置に使用できる **Pod** のグループに対して指定できます。ノード自体は配置に対する制御を行いません。

たとえば、アフィニティールールを使用することで、サービス内で、または他のサービスの Pod との関連で Pod を分散したり、パックしたりすることができます。非アフィニティールールにより、特定のサービスの Pod がそのサービスの Pod のパフォーマンスに干渉すると見なされる別のサービスの Pod と同じノードでスケジューリングされることを防ぐことができます。または、関連する障害を減らすために複数のノードまたはアベイラビリティゾーン間でサービスの Pod を分散することもできます。

Pod のアフィニティー/非アフィニティーにより、他の Pod のラベルに基づいて Pod のスケジューリング対象とするノードを制限することができます。[ラベル](#) はキー/値のペアです。

- Pod のアフィニティーはスケジューラーに対し、新規 Pod のラベセレクターが現在の Pod のラベルに一致する場合に他の Pod と同じノードで新規 Pod を見つけるように指示します。
- Pod の非アフィニティーは、新規 Pod のラベセレクターが現在の Pod のラベルに一致する場合に、同じラベルを持つ Pod と同じノードで新規 Pod を見つけることを禁止します。

Pod のアフィニティーには、**required (必須)** および **preferred (優先)** の 2 つのタイプがあります。

required (必須) ルールは、Pod をノードにスケジューリングする前に **満たされている必要があります**。一方、**preferred (優先)** ルールは、ルールが満たされる場合にスケジューラーがルールの実施を試行しますが、その実施が必ずしも保証される訳ではありません。

14.7.2. Pod のアフィニティーおよび非アフィニティーの設定

Pod のアフィニティー/非アフィニティーは Pod 仕様ファイルで設定します。[required \(必須\)](#) ルー

ル、**preferred (優先) ルール**のいずれかまたはその両方を指定することができます。両方を指定する場合、ノードは最初に **required (必須) ルール**を満たす必要があり、その後に **preferred (優先) ルール**を満たそうとします。

以下の例は、Pod のアフィニティーおよび非アフィニティーに設定される Pod 仕様を示しています。

この例では、Pod のアフィニティールールは ノードにキー **security** と値 **S1** を持つラベルの付いた 1 つ以上の Pod がすでに実行されている場合にのみ Pod をノードにスケジュールできることを示しています。Pod の非アフィニティールールは、ノードがキー **security** と値 **S2** を持つラベルが付いた Pod がすでに実行されている場合は Pod をノードにスケジュールしないように設定することを示しています。

Pod のアフィニティーが設定された Pod 設定のサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity: ❶
      requiredDuringSchedulingIgnoredDuringExecution: ❷
        - labelSelector:
            matchExpressions:
              - key: security ❸
                operator: In ❹
                values:
                  - S1 ❺
          topologyKey: failure-domain.beta.kubernetes.io/zone
  containers:
    - name: with-pod-affinity
      image: docker.io/ocpqe/hello-pod
```

❶ Pod のアフィニティーを設定するためのスタンザです。

❷ **required (必須) ルール**を定義します。

❸ ❺ ルールを適用するために一致している必要のあるキーと値 (ラベル) です。

❹ 演算子は、既存 Pod のラベルと新規 Pod の仕様の **matchExpression** パラメーターの値のセットの間の関係を表します。これには **In**、**NotIn**、**Exists**、または **DoesNotExist** のいずれかを使用できます。

Pod の非アフィニティーが設定された Pod 設定のサンプル

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: ❶
      preferredDuringSchedulingIgnoredDuringExecution: ❷
```



```

- weight: 100 ③
  podAffinityTerm:
    labelSelector:
      matchExpressions:
        - key: security ④
          operator: In ⑤
          values:
            - S2 ⑥
    topologyKey: kubernetes.io/hostname
containers:
- name: with-pod-affinity
  image: docker.io/ocpqe/hello-pod

```

- ① Pod の非アフィニティーを設定するためのスタンザです。
- ② preferred (優先) ルールを定義します。
- ③ preferred (優先) ルールの重みを指定します。最も高い重みを持つノードが優先されます。
- ④ ⑥ ルールを適用するために一致している必要のあるキーと値 (ラベル) です。
- ⑤ 演算子は、既存 Pod のラベルと新規 Pod の仕様の **matchExpression** パラメーターの値のセットの間の関係を表します。これには **In**、**NotIn**、**Exists**、または **DoesNotExist** のいずれかを使用できます。



注記

ノードのラベルに、Pod のノードのアフィニティールールを満たさなくなるような結果になる変更がランタイム時に生じる場合も、Pod はノードで引き続き実行されます。

14.7.2.1. アフィニティールールの設定

以下の手順は、ラベルの付いた Pod と Pod のスケジュールを可能にするアフィニティーを使用する Pod を作成する 2 つの Pod の単純な設定を示しています。

1. Pod 仕様の特定のラベルの付いた Pod を作成します。

```

$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod

```

2. 他の Pod の作成時に、以下のように Pod 仕様を編集します。

- a. **podAffinity** スタンザを使用し、**requiredDuringSchedulingIgnoredDuringExecution** パラメーターまたは **preferredDuringSchedulingIgnoredDuringExecution** パラメーターを設定しま

す。

- b. 満たしている必要のあるキーおよび値を指定します。新規 Pod を他の Pod と共にスケジュールする必要がある場合、最初の Pod のラベルと同じ **key** および **value** パラメーターを使用します。

```
podAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  - labelSelector:
      matchExpressions:
      - key: security
        operator: In
        values:
        - S1
    topologyKey: failure-domain.beta.kubernetes.io/zone
```

- c. **operator** を指定します。演算子は **In**、**NotIn**、**Exists**、または **DoesNotExist** にすることができます。たとえば、演算子 **In** を使用してラベルをノードで必要になるようにします。
- d. **topologyKey** を指定します。これは、システムがトポロジードメインを表すために使用する事前にデータが設定された [Kubernetes ラベル](#) です。

3. Pod を作成します。

```
$ oc create -f <pod-spec>.yaml
```

14.7.2.2. 非アフィニティールールの設定

以下の手順は、ラベルの付いた Pod と Pod のスケジュールの禁止を試行する非アフィニティの **preferred** (優先) ルールを使用する Pod を作成する 2 つの Pod の単純な設定を示しています。

1. Pod 仕様の特定のラベルの付いた Pod を作成します。

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s2
  labels:
    security: S2
spec:
  containers:
  - name: security-s2
    image: docker.io/ocpqe/hello-pod
```

2. 他の Pod の作成時に、Pod 仕様を編集して以下のパラメーターを設定します。
3. **podAffinity** スタンザを使用して、**requiredDuringSchedulingIgnoredDuringExecution** パラメーターまたは **preferredDuringSchedulingIgnoredDuringExecution** パラメーターを設定します。
 - a. ノードの重みを 1-100 で指定します。最も高い重みを持つノードが優先されます。

- b. 満たしている必要のあるキーおよび値を指定します。新規 Pod を他の Pod と共にスケジュールされないようにする必要がある場合、最初の Pod のラベルと同じ **key** および **value** パラメーターを使用します。

```
podAntiAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    podAffinityTerm:
      labelSelector:
        matchExpressions:
        - key: security
          operator: In
          values:
          - S2
      topologyKey: kubernetes.io/hostname
```

- c. preferred (優先) ルールの場合、重みを 1-100 で指定します。
- d. **operator** を指定します。演算子は **In**、**NotIn**、**Exists**、または **DoesNotExist** にすることができます。たとえば、演算子 **In** を使用してラベルをノードで必要になるようにします。
4. **topologyKey** を指定します。これは、システムがトポロジドメインを表すために使用する事前にデータが設定された [Kubernetes ラベル](#)です。
5. Pod を作成します。

```
$ oc create -f <pod-spec>.yaml
```

14.7.3. 各種の例

以下の例は、Pod のアフィニティーおよび非アフィニティーについて示しています。

14.7.3.1. Pod のアフィニティー

以下の例は、一致するラベルとラベルセレクターを持つ Pod についての Pod のアフィニティーを示しています。

- Pod **team4** にはラベル **team:4** が付けられています。

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- Pod **team4a** には、**podAffinity** の下にラベルセレクター **team:4** が付けられています。

```
$ cat pod-team4a.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4a
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: team
            operator: In
            values:
            - "4"
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
```

- **team4a** Pod は **team4** Pod と同じノードにスケジュールされます。

14.7.3.2. Pod の非アフィニティー

以下の例は、一致するラベルとラベルセクターを持つ Pod についての Pod の非アフィニティーを示しています。

- Pod **pod-s1** にはラベル **security:s1** が付けられています。

```
cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- Pod **pod-s2** には、**podAntiAffinity** の下にラベルセクター **security:s1** が付けられています。

```
cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
```

```

      - key: security
        operator: In
        values:
          - s1
    topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-antiaffinity
    image: docker.io/ocpqe/hello-pod

```

- Pod **pod-s2** は、**security:s2** ラベルの付いた Pod を持つノードがない場合はスケジュールされません。そのラベルの付いた他の Pod がない場合、新規 Pod は保留状態のままになります。

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
pod-s2	0/1	Pending	0	32s	<none>	

14.7.3.3. 一致するラベルのない Pod のアフィニティー

以下の例は、一致するラベルとラベルセクターのない Pod についての Pod のアフィニティーを示しています。

- Pod **pod-s1** にはラベル **security:s1** が付けられています。

```

$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod

```

- Pod **pod-s2** にはラベルセクター **security:s2** があります。

```

$ cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
            - key: security
              operator: In
              values:
                - s2
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity

```

```
image: docker.io/ocpqe/hello-pod
```

- Pod **pod-s2** は **pod-s1** と同じノードにスケジュールできません。

14.8. 詳細スケジューリングおよびノードセクター

14.8.1. 概要

ノードセクター はキーと値のペアのマップを指定します。ルールは、カスタムの **ノードのラベル** および Pod で指定されるセクターを使用して定義されます。

Pod がノードで実行する要件を満たすには、Pod はノードのラベルとして示されるキーと値のペアを持っている必要があります。

同じ Pod 設定でノードのアフィニティーと **ノードセクター** を使用している場合は、以下の「[重要な考慮事項](#)」を参照してください。

14.8.2. ノードセクターの設定

Pod 設定で **nodeSelector** を使用することで、Pod を特定のラベルの付いたノードのみに配置することができます。

- 必要なラベル (詳細は、「[ノードでのラベルの更新](#)」を参照) および **ノードセクター** が環境にセットアップされていることを確認します。
たとえば、Pod 設定が必要なラベルを示す **nodeSelector** 値を持つことを確認します。

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:
    <key>: <value>
  ...
```

- マスター設定ファイル (**/etc/origin/master/master-config.yaml**) を 2 箇所を変更します。

- nodeSelectorLabelBlacklist** を、Pod の配置を拒否する必要のあるノードホストに割り当てられるラベルと共に **admissionConfig** セクションに追加します。

```
...
admissionConfig:
  pluginConfig:
    PodNodeConstraints:
      configuration:
        apiversion: v1
        kind: PodNodeConstraintsConfig
        nodeSelectorLabelBlacklist:
          - kubernetes.io/hostname
          - <label>
  ...
```

- 次に、同じ設定を **kubernetesMasterConfig** セクションに追加し、Pod の直接の作成を制限します。

```

...
kubernetesMasterConfig:
  admissionConfig:
    pluginConfig:
      PodNodeConstraints:
        configuration:
          apiVersion: v1
          kind: PodNodeConstraintsConfig
          nodeSelectorLabelBlacklist:
            - kubernetes.io/hostname
            - <label_1>
...

```

3. 変更を有効にするために OpenShift Container Platform を再起動します。

```
#systemctl restart atomic-openshift-master
```

注記

同じ Pod 設定でノードセクターと [ノードのアフィニティ](#) を使用している場合は、以下に注意してください。

- **nodeSelector** と **nodeAffinity** の両方を設定する場合、Pod が候補ノードでスケジュールされるにはどちらの条件も満たしている必要があります。
- **nodeAffinity** タイプに関連付けられた複数の **nodeSelectorTerms** を指定する場合、**nodeSelectorTerms** のいずれかが満たされている場合に Pod をノードにスケジュールすることができます。
- **nodeSelectorTerms** に関連付けられた複数の **matchExpressions** を指定する場合、すべての **matchExpressions** が満たされている場合にのみ Pod をノードにスケジュールすることができます。

14.9. 詳細スケジューリングおよび容認

14.9.1. 概要

テイントおよび容認により、ノードはノード上でスケジュールする必要のある (またはスケジュールすべきでない) Pod を制御できます。

14.9.2. テイントおよび容認 (Toleration)

テイントにより、ノードは Pod に一致する容認がない場合に Pod のスケジュールを拒否することができます。

テイントはノード仕様 (**NodeSpec**) でノードに適用され、容認は Pod 仕様 (**PodSpec**) で Pod に適用されます。ノードのテイントはノードに対し、テイントを容認しないすべての Pod を拒否するよう指示します。

テイントおよび容認は、key、value、および effect で構成されています。演算子により、これらの3つのパラメーターのいずれかを空のままにすることができます。

表14.1 テイントおよび容認コンポーネント

パラメーター	説明						
key	key には、253 文字までの文字列を使用できます。キーは文字または数字で開始する必要があり、文字、数字、ハイフン、ドットおよびアンダースコアを含めることができます。						
value	value には、63 文字までの文字列を使用できます。値は文字または数字で開始する必要があり、文字、数字、ハイフン、ドットおよびアンダースコアを含めることができます。						
effect	<p>effect は以下のいずれかにすることができます。</p> <table> <tr> <td>NoSchedule</td><td> <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされません。 • ノードの既存 Pod はそのままになります。 </td></tr> <tr> <td>PreferNoSchedule</td><td> <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります、スケジューラーはスケジュールしないようにします。 • ノードの既存 Pod はそのままになります。 </td></tr> <tr> <td>NoExecute</td><td> <ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールできません。 • 一致する容認を持たないノードの既存 Pod は削除されます。 </td></tr> </table>	NoSchedule	<ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされません。 • ノードの既存 Pod はそのままになります。 	PreferNoSchedule	<ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります、スケジューラーはスケジュールしないようにします。 • ノードの既存 Pod はそのままになります。 	NoExecute	<ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールできません。 • 一致する容認を持たないノードの既存 Pod は削除されます。
NoSchedule	<ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされません。 • ノードの既存 Pod はそのままになります。 						
PreferNoSchedule	<ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります、スケジューラーはスケジュールしないようにします。 • ノードの既存 Pod はそのままになります。 						
NoExecute	<ul style="list-style-type: none"> • テイントに一致しない新規 Pod はノードにスケジュールできません。 • 一致する容認を持たないノードの既存 Pod は削除されます。 						
operator	<table> <tr> <td>Equal</td><td>key/value/effect パラメーターは一致する必要があります。これはデフォルトになります。</td></tr> <tr> <td>Exists</td><td>key/effect パラメーターは一致する必要があります。いずれかに一致する value パラメーターを空のままにする必要があります。</td></tr> </table>	Equal	key/value/effect パラメーターは一致する必要があります。これはデフォルトになります。	Exists	key/effect パラメーターは一致する必要があります。いずれかに一致する value パラメーターを空のままにする必要があります。		
Equal	key/value/effect パラメーターは一致する必要があります。これはデフォルトになります。						
Exists	key/effect パラメーターは一致する必要があります。いずれかに一致する value パラメーターを空のままにする必要があります。						

容認はテイントと一致します。

- **operator** パラメーターが **Equal** に設定されている場合:
 - **key** パラメーターは同じになります。
 - **value** パラメーターは同じになります。

- **effect** パラメーターは同じになります。
- **operator** パラメーターが **Exists** に設定されている場合:
 - **key** パラメーターは同じになります。
 - **effect** パラメーターは同じになります。

14.9.2.1. 複数テイントの使用

複数のテイントを同じノードに、複数の容認を同じ Pod に配置することができます。OpenShift Container Platform は複数のテイントと容認を以下のように処理します。

1. Pod に一致する容認のあるテイントを処理します。
2. 残りの一致しないテイントは Pod について以下の effect を持ちます。
 - effect が **NoSchedule** の一致しないテイントが 1 つ以上ある場合、OpenShift Container Platform は Pod をノードにスケジュールできません。
 - effect が **NoSchedule** の一致しないテイントがなく、effect が **PreferNoSchedule** の一致しない テイントが 1 つ以上ある場合、OpenShift Container Platform は Pod のノードへのスケジュールを試行しません。
 - effect が **NoExecute** のテイントが 1 つ以上ある場合、OpenShift Container Platform は Pod をノードからエビクトするか (ノードですでに実行中の場合)、または Pod のそのノードへのスケジュールが実行されません (ノードでまだ実行されていない場合)。
 - テイントを容認しない Pod はすぐにエビクトされます。
 - 容認の仕様に **tolerationSeconds** を指定せずにテイントを容認する Pod は永久にバインドされたままになります。
 - 指定された **tolerationSeconds** を持つテイントを容認する Pod は指定された期間バインドされます。

以下に例を示します。

- ノードには以下のテイントがあります。

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
$ oc adm taint nodes node1 key1=value1:NoExecute
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- Pod には以下の容認があります。

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
```

この場合、3 つ目のテイントに一致する容認がないため、Pod はノードにスケジュールできません。Pod はこのテイントの追加時にノードですでに実行されている場合は実行が継続されます。3 つ目のテイントは 3 つのテイントの中で Pod で容認されない唯一のテイントであるためです。

14.9.3. テイントの既存ノードへの追加

[テイントおよび容認コンポーネント](#)の表で説明されているパラメーターと共に **oc adm taint** コマンドを使用してテイントをノードに追加します。

```
$ oc adm taint nodes <node-name> <key>=<value>:<effect>
```

以下に例を示します。

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

この例では、テイントを、キー **key1**、値 **value1**、およびテイント effect **NoSchedule** を持つ **node1** にテイントを配置します。

14.9.4. 容認の Pod への追加

容認を Pod に追加するには、Pod 仕様を **tolerations** セクションを含めるように編集します。

Equal 演算子を含む Pod 設定ファイルのサンプル

```
tolerations:
- key: "key1" ①
  operator: "Equal" ②
  value: "value1" ③
  effect: "NoExecute" ④
  tolerationSeconds: 3600 ⑤
```

① ② ③ ④ [テイントおよび容認コンポーネント](#) の表で説明されている toleration パラメーターです。

⑤ **tolerationSeconds** パラメーターは、Pod がエビクトされる前にノードにバインドされる期間を指定します。以下の「[Pod エビクションを遅延させる容認期間 \(秒数\) の使用](#)」を参照してください。

Exists 演算子を含む Pod 設定ファイルのサンプル

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

これらの容認のいずれも [上記の oc adm taint コマンド](#)で作成されるテイントに一致します。いずれかの容認のある Pod は **node1** にスケジュールできます。

14.9.4.1. Pod のエビクションを遅延させる容認期間 (秒数) の使用

Pod 仕様に **tolerationSeconds** パラメーターを指定して、Pod がエビクトされる前にノードにバイ

ンドされる期間を指定できます。effect **NoExecute** のあるテイントがノードに追加される場合、テイントを容認しない Pod は即時にエビクトされます (テイントを容認する Pod はエビクトされません)。ただし、エビクトされる Pod に **tolerationSeconds** パラメーターがある場合、Pod は期間切れになるまでエビクトされません。

以下に例を示します。

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

ここで、この Pod が実行中であるものの、一致するテイントがない場合、Pod は 3,600 秒間バインドされたままとなり、その後にエビクトされます。テイントが期限前に削除される場合、Pod はエビクトされません。

14.9.4.1.1. 容認の秒数のデフォルト値の設定

このプラグインは、**node.alpha.kubernetes.io/notReady:NoExecute** および **node.alpha.kubernetes.io/notReady:NoExecute** テイントを 5 分間容認するための Pod のデフォルトの容認を設定します。

ユーザーが提供する Pod 設定にいずれかの容認がある場合、デフォルトは追加されません。

デフォルトの容認の秒数を有効にするには、以下を実行します。

1. マスター設定ファイル (**/etc/origin/master/master-config.yaml**) を変更して **DefaultTolerationSeconds** を admissionConfig セクションに追加します。

```
admissionConfig:
  pluginConfig:
    DefaultTolerationSeconds:
      configuration:
        kind: DefaultAdmissionConfig
        apiVersion: v1
        disable: false
```

2. 変更を有効にするために、OpenShift を再起動します。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-
master-controllers
```

3. デフォルトが追加されていることを確認します。

- a. Pod を作成します。

```
$ oc create -f </path/to/file>
```

以下に例を示します。

```
$ oc create -f hello-pod.yaml
pod "hello-pod" created
```

b. Pod の容認を確認します。

```
$ oc describe pod <pod-name> |grep -i toleration
```

以下に例を示します。

```
$ oc describe pod hello-pod |grep -i toleration
Tolerations:
node.alpha.kubernetes.io/notReady=:Exists:NoExecute for 300s
```

14.9.5. ノードの問題の発生時における Pod エビクションの禁止

OpenShift Container Platform は、**node unreachable** および **node not ready** 状態をテイントとして表示するよう設定できます。これにより、デフォルトの 5 分を使用するのではなく、unreachable (到達不能) または not ready (準備ができていない) 状態になるノードにバインドされたままになる期間を Pod 仕様ごとに指定することができます。

テイントベースのエビクション機能が有効にされた状態で、テイントはノードコントローラーによって自動的に追加され、Pod を **Ready** ノードからエビクトするための通常のロジックは無効にされます。

- ノードが not ready (準備ができていない) 状態になると、**node.alpha.kubernetes.io/notReady:NoExecute** テイントは追加され、Pod はノードでスケジュールできなくなります。既存 Pod は容認期間 (秒数) 中はそのまま残ります。
- ノードが not reachable (到達不能) の状態になると、**node.alpha.kubernetes.io/unreachable:NoExecute** テイントは追加され、Pod はノードでスケジュールできません。既存の Pod は容認期間 (秒数) 中はそのまま残ります。

テイントベースのエビクションを有効にするには、以下を実行します。

1. マスター設定ファイル (**/etc/origin/master/master-config.yaml**) を変更して以下を **kubernetesMasterConfig** セクションに追加します。

```
kubernetesMasterConfig:
  controllerArguments:
    feature-gates:
      - "TaintBasedEvictions=true"
```

2. テイントがノードに追加されていることを確認します。

```
oc describe node $node | grep -i taint

Taints: node.alpha.kubernetes.io/notReady:NoExecute
```

3. 変更を有効にするために、OpenShift を再起動します。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

4. 容認を Pod に追加します。

```
tolerations:
- key: "node.alpha.kubernetes.io/unreachable"
  operator: "Exists"
```

```
effect: "NoExecute"
tolerationSeconds: 6000
```

または

```
tolerations:
- key: "node.alpha.kubernetes.io/notReady"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```



注記

ノードの問題の発生時に Pod エビクションの既存のレート制限の動作を維持するために、システムはテイントをレートが制限された方法で追加します。これにより、マスターがノードからパーティション化される場合などのシナリオで発生する大規模な Pod エビクションを防ぐことができます。

14.9.6. Daemonset および容認

Daemonset Pod は、Default Toleration Seconds (デフォルトの容認期間の秒数) が無効にされている場合でも、**tolerationSeconds** のない **node.alpha.kubernetes.io/unreachable** および **node.alpha.kubernetes.io/notReady** の **NoExecute** 容認の設定と共に作成されます。

14.9.7. 各種の例

テイントおよび容認は、Pod をノードから切り離し、ノードで実行されるべきでない Pod をエビクトする柔軟性のある方法として使用できます。以下は典型的なシナリオのいくつかになります。

- ノードをユーザー専用にする
- ユーザーをノードにバインドする
- 特殊ハードウェアを持つノードを専用ノードにする

14.9.7.1. ノードをユーザー専用にする

ノードのセットを特定のユーザーセットが排他的に使用するよう指定できます。

専用ノードを指定するには、以下を実行します。

1. テイントをそれらのノードに追加します。
以下に例を示します。

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. カスタムの受付コントローラーを作成して、対応する容認を Pod に追加します。
容認のある Pod のみが専用ノードを使用することを許可されます。

14.9.7.2. ユーザーのノードへのバインド

特定ユーザーが専用ノードのみを使用できるようにノードを設定することができます。

ノードをユーザーの使用可能な唯一のノードとして設定するには、以下を実行します。

1. テイントをそれらのノードに追加します。
以下に例を示します。

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. カスタムの[受付コントローラー](#)を作成して、対応する容認を Pod に追加します。
受付コントローラーは、Pod が **key:value** ラベル (**dedicated=groupName**) が付けられたノードのみにスケジュールされるようにノードのアフィニティーを追加します。
3. テイントと同様のラベル (**key:value** ラベルなど) を専用ノードに追加します。

14.9.7.3. 特殊ハードウェアを持つノード

ノードの小規模なサブセットが特殊ハードウェア(GPU など) を持つクラスターでは、テイントおよび容認を使用して、特殊ハードウェアを必要としない Pod をそれらのノードから切り離し、特殊ハードウェアを必要とする Pod をそのままにすることができます。また、特殊ハードウェアを必要とする Pod に対して特定のノードを使用することを要求することもできます。

Pod が特殊ハードウェアからブロックされるようにするには、以下を実行します。

1. 以下のコマンドのいずれかを使用して、特殊ハードウェアを持つノードにテイントを設定します。

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

2. [受付コントローラー](#)を使用して特殊ハードウェアを使用する Pod に対応する容認を追加します。

たとえば受付コントローラーは容認を追加することで、Pod の一部の特徴を使用し、Pod が特殊ノードを使用できるかどうかを判別できます。

Pod が特殊ハードウェアのみを使用できるようにするには、追加のメカニズムが必要です。たとえば、特殊ハードウェアを持つノードにラベルを付け、ハードウェアを必要とする Pod でノードのアフィニティーを使用できます。

第15章 クォータの設定

15.1. 概要

ResourceQuota オブジェクトで定義されるリソースクォータは、プロジェクトごとにリソース消費量の総計を制限する制約を指定します。これは、タイプ別にプロジェクトで作成できるオブジェクトの数を制限すると共に、そのプロジェクトのリソースが消費できるコンピュートリソースおよびストレージの合計量を制限することができます。



注記

コンピュートリソースについての詳細は、『[Developer Guide](#)』を参照してください。

15.2. クォータで管理されるリソース

以下では、クォータで管理できる一連のコンピュートリソースとオブジェクトタイプについて説明します。



注記

status.phase in (Failed, Succeeded) が true の場合、Pod は終了状態にあります。

表15.1 クォータで管理されるコンピュートリソース

リソース名	説明
cpu	非終了状態のすべての Pod での CPU 要求の合計はこの値を超えることができません。 cpu および requests.cpu は同じ値で、交換可能なものとして使用できます。
memory	非終了状態のすべての Pod でのメモリー要求の合計はこの値を超えることができません memory および requests.memory は同じ値で、交換可能なものとして使用できます。
requests.cpu	非終了状態のすべての Pod での CPU 要求の合計はこの値を超えることができません。 cpu および requests.cpu は同じ値で、交換可能なものとして使用できます。
requests.memory	非終了状態のすべての Pod でのメモリー要求の合計はこの値を超えることができません memory および requests.memory は同じ値で、交換可能なものとして使用できます。
limits.cpu	非終了状態のすべての Pod での CPU 制限の合計はこの値を超えることができません。
limits.memory	非終了状態のすべての Pod でのメモリー制限の合計はこの値を超えることができません。

表15.2 クォータで管理されるストレージリソース

リソース名	説明
requests.storage	任意の状態のすべての Persistent Volume Claim (永続ボリューム要求、PVC) でのストレージ要求の合計はこの値を超えることができません。
persistentvolumeclaims	プロジェクトに存在できる Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。
<storage-class-name>.storageclass.storage.k8s.io/requests.storage	一致するストレージクラスを持つ、任意の状態のすべての Persistent Volume Claim (永続ボリューム要求、PVC) でのストレージ要求の合計はこの値を超えることができません。
<storage-class-name>.storageclass.storage.k8s.io/persistentvolumeclaims	プロジェクトに存在できる、一致するストレージクラスを持つ Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。

表15.3 クォータで管理されるオブジェクト数

リソース名	説明
pods	プロジェクトに存在できる非終了状態の Pod の合計数です。
replicationcontrollers	プロジェクトに存在できるレプリケーションコントローラーの合計数です。
resourcequotas	プロジェクトに存在できるリソースクォータの合計数です。
services	プロジェクトに存在できるサービスの合計数です。
secrets	プロジェクトに存在できるシークレットの合計数です。
configmaps	プロジェクトに存在できる ConfigMap オブジェクトの合計数です。
persistentvolumeclaims	プロジェクトに存在できる Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。
openshift.io/imagestreams	プロジェクトの存在できるイメージストリームの合計数です。

15.3. クォータのスコープ

各クォータには **スコープ** のセットが関連付けられます。クォータは、列挙されたスコープの交差部分に一致する場合にのみリソースの使用状況を測定します。

スコープをクォータに追加すると、クォータが適用されるリソースのセットを制限できます。許可されるセット以外のリソースを設定すると、検証エラーが発生します。

スコープ	説明
Terminating	<code>spec.activeDeadlineSeconds >= 0</code> の Pod に一致します。
NotTerminating	<code>spec.activeDeadlineSeconds</code> が <code>nil</code> の Pod に一致します。
BestEffort	<code>cpu</code> または <code>memory</code> のいずれかの QoS (Quality of Service) が Best Effort の Pod に一致します。コンピュータリソースのコミットについての詳細は、「 QoS (Quality of Service) クラス 」を参照してください。
NotBestEffort	<code>cpu</code> および <code>memory</code> の QoS (Quality of Service) が Best Effort でない Pod に一致します。

BestEffort スコープは、以下のリソースを制限することにクォータを制限します。

- `pods`

Terminating、**NotTerminating**、および **NotBestEffort** スコープは、以下のリソースを追跡することにクォータを制限します。

- `pods`
- `memory`
- `requests.memory`
- `limits.memory`
- `cpu`
- `requests.cpu`
- `limits.cpu`

15.4. クォータの実施

プロジェクトのリソースクォータが最初に作成されると、プロジェクトは、更新された使用状況の統計が計算されるまでクォータの制約に違反する可能性のある新規リソースの作成機能を制限します。

クォータが作成され、使用状況の統計が更新されると、プロジェクトは新規コンテンツの作成を許可します。リソースを作成または変更する場合、クォータの使用量はリソースの作成または変更要求があるときに増分します。

リソースを削除する場合、クォータの使用量は、プロジェクトのクォータ統計の次の完全な再計算時に減分されます。設定可能な時間を指定して、クォータ使用量の統計値を現在確認されるシステム値まで下げるのに必要な時間を決定します。

プロジェクト変更がクォータ使用制限を超える場合、サーバーはそのアクションを拒否し、クォータ制約を違反していること、およびシステムで現在確認される使用量の統計値を示す適切なエラーメッセージがユーザーに返されます。

15.5. 要求 VS 制限

コンピュータリソースの割り当て時に、各コンテナでは CPU とメモリーのそれぞれに要求値と制限値を指定できます。クォータはこれらの値を制限できます。

クォータに **requests.cpu** または **requests.memory** の値が指定されている場合、すべての着信コンテナがそれらのリソースを明示的に要求することが求められます。クォータに **limits.cpu** または **limits.memory** の値が指定されている場合、すべての着信コンテナがそれらのリソースの明示的な制限を指定することが求められます。

15.6. リソースクォータ定義のサンプル

core-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: core-object-counts
spec:
  hard:
    configmaps: "10" ❶
    persistentvolumeclaims: "4" ❷
    replicationcontrollers: "20" ❸
    secrets: "10" ❹
    services: "10" ❺
```

- ❶ プロジェクトに存在できる **ConfigMap** オブジェクトの合計数です。
- ❷ プロジェクトに存在できる Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。
- ❸ プロジェクトに存在できるレプリケーションコントローラーの合計数です。
- ❹ プロジェクトに存在できるシークレットの合計数です。
- ❺ プロジェクトに存在できるサービスの合計数です。

openshift-object-counts.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: openshift-object-counts
spec:
  hard:
    openshift.io/imagestreams: "10" ❶
```

- ❶ プロジェクトの存在できるイメージストリームの合計数です。

compute-resources.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
```

```

name: compute-resources
spec:
  hard:
    pods: "4" ❶
    requests.cpu: "1" ❷
    requests.memory: 1Gi ❸
    limits.cpu: "2" ❹
    limits.memory: 2Gi ❺

```

- ❶ プロジェクトに存在できる非終了状態の Pod の合計数です。
- ❷ 非終了状態のすべての Pod において、CPU 要求の合計は 1 コアを超えることができません。
- ❸ 非終了状態のすべての Pod において、メモリー要求の合計は 1 Gi をを超えることができません。
- ❹ 非終了状態のすべての Pod において、CPU 制限の合計は 2 コアを超えることができません。
- ❺ 非終了状態のすべての Pod において、メモリー制限の合計は 2 Gi をを超えることができません。

besteffort.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: besteffort
spec:
  hard:
    pods: "1" ❶
  scopes:
    - BestEffort ❷

```

- ❶ プロジェクトに存在できる QoS (Quality of Service) が **BestEffort** の非終了状態の Pod の合計数です。
- ❷ クォータを、メモリーまたは CPU のいずれかの QoS (Quality of Service) が **BestEffort** の一致する Pod のみに制限します。

compute-resources-long-running.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-long-running
spec:
  hard:
    pods: "4" ❶
    limits.cpu: "4" ❷
    limits.memory: "2Gi" ❸
  scopes:
    - NotTerminating ❹

```

- ❶ 非終了状態の Pod の合計数です。

- ❶ 非終了状態のすべての Pod において、CPU 制限の合計はこの値を超えることができません。
- ❷ 非終了状態のすべての Pod において、メモリー制限の合計はこの値を超えることができません。
- ❸ クォータを `spec.activeDeadlineSeconds` が `nil` に設定されている一致する Pod のみに制限します。ビルド Pod は、`RestartNever` ポリシーが適用されない場合に `NotTerminating` になります。

compute-resources-time-bound.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-time-bound
spec:
  hard:
    pods: "2" ❶
    limits.cpu: "1" ❷
    limits.memory: "1Gi" ❸
  scopes:
    - Terminating ❹
```

- ❶ 非終了状態の Pod の合計数です。
- ❷ 非終了状態のすべての Pod において、CPU 制限の合計はこの値を超えることができません。
- ❸ 非終了状態のすべての Pod において、メモリー制限の合計はこの値を超えることができません。
- ❹ クォータを `spec.activeDeadlineSeconds >=0` に設定されている一致する Pod のみに制限します。たとえば、このクォータはビルド Pod またはデプロイヤー Pod に影響を与えますが、web サーバーまたはデータベースなどの長時間実行されない Pod には影響を与えません。

storage-consumption.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage-consumption
spec:
  hard:
    persistentvolumeclaims: "10" ❶
    requests.storage: "50Gi" ❷
    gold.storageclass.storage.k8s.io/requests.storage: "10Gi" ❸
    silver.storageclass.storage.k8s.io/requests.storage: "20Gi" ❹
    silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" ❺
    bronze.storageclass.storage.k8s.io/requests.storage: "0" ❻
    bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" ❼
```

- ❶ プロジェクト内の Persistent Volume Claim (永続ボリューム要求、PVC) の合計数です。
- ❷ プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、要求されるストレージの合計はこの値を超えることができません。

- 3 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、gold ストレージクラスで要求されるストレージの合計はこの値を超えることができません。
- 4 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、silver ストレージクラスで要求されるストレージの合計はこの値を超えることができません。
- 5 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、silver ストレージクラスの要求の合計数はこの値を超えることができません。
- 6 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、bronze ストレージクラスで要求されるストレージの合計はこの値を超えることができません。これが 0 に設定される場合、bronze ストレージクラスはストレージを要求できないことを意味します。
- 7 プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、bronze ストレージクラスで要求されるストレージの合計はこの値を超えることができません。これが 0 に設定される場合は、bronze ストレージクラスでは要求を作成できないことを意味します。

15.7. クォータの作成

クォータを作成するには、「[リソースクォータ定義のサンプル](#)」に示されるように、まずクォータをファイルの仕様に基づいて定義します。次に、そのファイルを使用してこれをプロジェクトに適用します。

```
$ oc create -f <resource_quota_definition> [-n <project_name>]
```

以下に例を示します。

```
$ oc create -f resource-quota.json -n demoproject
```

15.8. クォータの表示

web コンソールでプロジェクトの **Quota** ページに移動し、プロジェクトのクォータで定義されるハード制限に関連する使用状況の統計を表示できます。

CLI を使用してクォータの詳細を表示することもできます。

1. 最初に、プロジェクトで定義されたクォータの一覧を取得します。たとえば、**demoproject** というプロジェクトの場合、以下を実行します。

```
$ oc get quota -n demoproject
NAME                AGE
besteffort          11m
compute-resources   2m
core-object-counts  29m
```

2. 次に、関連するクォータについての説明を表示します。たとえば、**core-object-counts** クォータの場合は、以下ようになります。

```
$ oc describe quota core-object-counts -n demoproject
Name:      core-object-counts
Namespace: demoproject
Resource  Used Hard
-----  -

```

```

configmaps 3 10
persistentvolumeclaims 0 4
replicationcontrollers 3 20
secrets 9 10
services 2 10

```

15.9. クォータの同期期間の設定

リソースのセットが削除される際に、リソースの同期期間が `/etc/origin/master/master-config.yaml` ファイルの **resource-quota-sync-period** 設定によって決定されます。

クォータの使用状況が復元される前に、ユーザーがリソースの再使用を試行すると問題が発生する場合があります。**resource-quota-sync-period** 設定を変更して、リソースセットの再生成が所定の期間 (秒単位) に実行され、リソースを再度利用可能にすることができます。

```

kubernetesMasterConfig:
  apiLevels:
    - v1beta3
    - v1
  apiServerArguments: null
  controllerArguments:
    resource-quota-sync-period:
      - "10s"

```

変更後に、マスターサービスを再起動してそれらの変更を適用します。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

再生成時間の調整は、リソースの作成および自動化が使用される場合のリソース使用状況の判別に役立ちます。



注記

resource-quota-sync-period 設定は、システムパフォーマンスのバランスを取るよう設計されています。同期期間を短縮すると、マスターに大きな負荷がかかる可能性があります。

15.10. デプロイメント設定におけるクォータアカウンティング

クォータがプロジェクトに定義されている場合、デプロイメント設定の考慮事項については、「[デプロイメントリソース](#)」を参照してください。

15.11. リソース消費における明示的なクォータの要求

リソースがクォータで管理されていない場合、ユーザーには消費できるリソース量の制限がありません。たとえば、gold ストレージクラスに関連するストレージのクォータがない場合、プロジェクトが作成できる gold ストレージの容量はバインドされません。

高コストのコンピュートまたはストレージリソースの場合、管理者はリソースを消費するための明示的なクォータの付与が必要となるようにする場合があります。たとえば、プロジェクトに gold ストレージクラスに関連するストレージのクォータが明示的に付与されていない場合、そのプロジェクトのユーザーはこのタイプのストレージを作成することができません。

特定リソースの消費における明示的なクォータが必要となるようにするには、以下のスタンザを master-config.yaml に追加する必要があります。

```
admissionConfig:
  pluginConfig:
    ResourceQuota:
      configuration:
        apiVersion: resourcequota.admission.k8s.io/v1alpha1
        kind: Configuration
        limitedResources:
          - resource: persistentvolumeclaims ❶
            matchContains:
          - gold.storageclass.storage.k8s.io/requests.storage ❷
```

- ❶ デフォルトで消費が制限されるグループ/リソースです。
- ❷ デフォルトで制限対象となる、グループ/リソースに関連付けられたクォータで追跡されるリソースの名前です。

上記の例では、クォータシステムは **PersistentVolumeClaim** を作成するか、または更新するすべての操作をインターセプトします。これは、クォータで認識されるリソースが消費されることを確認し、プロジェクトのそれらのリソースのクォータがない場合に要求は拒否されます。この例ではユーザーが gold ストレージクラスに関連付けられたストレージを使用する **PersistentVolumeClaim** を作成しており、プロジェクトに一致するクォータがない場合には要求が拒否されます。

第16章 複数プロジェクトのクォータ設定

16.1. 概要

ClusterResourceQuota オブジェクトで定義される複数プロジェクトのクォータは、[クォータ](#)を複数プロジェクト間で共有できるようにします。それぞれの選択されたプロジェクトで使用されるリソースは集計され、その集計は選択したすべてのプロジェクトでリソースを制限するために使用されます。

16.2. プロジェクトの選択

プロジェクトは、アノテーションの選択またはラベルの選択のいずれか、またはその両方に基づいて選択できます。たとえば、アノテーションに基づいてプロジェクトを選択するには、以下のコマンドを実行します。

```
$ oc create clusterquota for-user \
  --project-annotation-selector openshift.io/requester=<user-name> \
  --hard pods=10 \
  --hard secrets=20
```

これは以下の **ClusterResourceQuota** オブジェクトを作成します。

```
apiVersion: v1
kind: ClusterResourceQuota
metadata:
  name: for-user
spec:
  quota: ①
  hard:
    pods: "10"
    secrets: "20"
  selector:
    annotations: ②
    openshift.io/requester: <user-name>
    labels: null ③
status:
  namespaces: ④
  - namespace: ns-one
    status:
      hard:
        pods: "10"
        secrets: "20"
      used:
        pods: "1"
        secrets: "9"
  total: ⑤
  hard:
    pods: "10"
    secrets: "20"
  used:
    pods: "1"
    secrets: "9"
```

① 選択したプロジェクトに対して実施される **ResourceQuotaSpec** オブジェクトです。

- 2 アノテーションの単純なキー/値のセクターです。
- 3 プロジェクトを選択するために使用できるラベルセクターです。
- 4 選択された各プロジェクトの現在のクォータの使用状況を記述する namespace ごとのマップです。
- 5 選択されたすべてのプロジェクトにおける使用量の総計です。

この複数プロジェクトのクォータの記述は、デフォルトのプロジェクト要求エンドポイントを使用して **<user-name>** によって要求されるすべてのプロジェクトを制御します。ここでは、10 Pod および 20 シークレットに制限されます。

同様にラベルに基づいてプロジェクトを選択するには、以下のコマンドを実行します。

```
$ oc create clusterresourcequota for-name \ 1
    --project-label-selector=name=frontend \ 2
    --hard=pods=10 --hard=secrets=20
```

- 1 **clusterresourcequota** および **clusterquota** は同じコマンドのエイリアスです。for-name は **clusterresourcequota** オブジェクトの名前です。
- 2 ラベル別にプロジェクトを選択するには、**--project-label-selector=key=value** 形式を使用してキーと値のペアを指定します。

これは以下の **ClusterResourceQuota** オブジェクト定義を作成します。

```
apiVersion: v1
kind: ClusterResourceQuota
metadata:
  creationTimestamp: null
  name: for-name
spec:
  quota:
    hard:
      pods: "10"
      secrets: "20"
  selector:
    annotations: null
    labels:
      matchLabels:
        name: frontend
```

16.3. 適用可能な CLUSTERRESOURCEQUOTAS の表示

プロジェクト管理者は、各自のプロジェクトを制限する複数プロジェクトのクォータを作成したり、変更したりすることはできませんが、それぞれのプロジェクトに適用される複数プロジェクトのクォータを表示することはできます。プロジェクト管理者は、**AppliedClusterResourceQuota** リソースを使ってこれを実行できます。

```
$ oc describe AppliedClusterResourceQuota
```

以下が生成されます。

```
Name:      for-user
Namespace:  <none>
Created:    19 hours ago
Labels:     <none>
Annotations: <none>
Label Selector: <null>
AnnotationSelector: map[openshift.io/requester:<user-name>]
Resource   Used   Hard
-----
pods        1    10
secrets     9    20
```

16.4. 選択における粒度

クォータの割り当てを要求する際にロックに関して考慮する必要があるため、複数プロジェクトのクォータで選択されるアクティブなプロジェクトの数は重要な考慮点になります。単一の複数プロジェクトクォータで 100 を超えるプロジェクトを選択すると、それらのプロジェクトの API サーバーの応答に負の影響が及びます。

第17章 制限範囲の設定

17.1. 概要

LimitRange オブジェクトで定義される制限範囲は、Pod、コンテナ、イメージ、イメージストリーム、および Persistent Volume Claim (永続ボリューム要求、PVC) のレベルでプロジェクトのコンピュートリソース制約を列挙し、Pod、コンテナ、イメージ、イメージストリームまたは Persistent Volume Claim (永続ボリューム要求、PVC) で消費できるリソースの量を指定します。

すべてのリソース作成および変更要求は、プロジェクトの各 **LimitRange** オブジェクトに対して評価されます。リソースが列挙された制約のいずれかに違反する場合、リソースは拒否されます。リソースが明示的な値を指定しない場合で、制約がデフォルト値をサポートする場合は、デフォルト値がリソースに適用されます。

コア Limit Range オブジェクトの定義

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "core-resource-limits" ❶
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" ❷
        memory: "1Gi" ❸
      min:
        cpu: "200m" ❹
        memory: "6Mi" ❺
    - type: "Container"
      max:
        cpu: "2" ❻
        memory: "1Gi" ❼
      min:
        cpu: "100m" ❽
        memory: "4Mi" ❾
      default:
        cpu: "300m" ❿
        memory: "200Mi" ㉑
      defaultRequest:
        cpu: "200m" ㉒
        memory: "100Mi" ㉓
      maxLimitRequestRatio:
        cpu: "10" ㉔
```

- ❶ 制限範囲オブジェクトの名前です。
- ❷ すべてのコンテナにおいて Pod がノードで要求できる CPU の最大量です。
- ❸ すべてのコンテナにおいて Pod がノードで要求できるメモリの最大量です。
- ❹ すべてのコンテナにおいて Pod がノードで要求できる CPU の最小量です。

- 5 すべてのコンテナにおいて Pod がノードで要求できるメモリの最小量です。
- 6 Pod の単一コンテナが要求できる CPU の最大量です。
- 7 Pod の単一コンテナが要求できるメモリの最大量です。
- 8 Pod の単一コンテナが要求できる CPU の最小量です。
- 9 Pod の単一コンテナが要求できるメモリの最小量です。
- 10 指定されない場合にコンテナによる使用制限となる CPU のデフォルト量です。
- 11 指定されない場合にコンテナによる使用制限となるメモリのデフォルト量です。
- 12 指定されない場合にコンテナによる要求制限となる CPU のデフォルト量です。
- 13 指定されない場合にコンテナの要求制限となるメモリのデフォルト量です。
- 14 制限の要求に対する比率でコンテナで実行できる CPU バーストの最大量です。

CPU およびメモリの測定方法についての詳細は、「[コンピュータリソース](#)」を参照してください。

OpenShift Container Platform の Limit Range オブジェクトの定義

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "openshift-resource-limits"
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi 1
    - type: openshift.io/ImageStream
      max:
        openshift.io/image-tags: 20 2
        openshift.io/images: 30 3

```

- 1 内部レジストリーにプッシュできるイメージの最大サイズです。
- 2 イメージストリームの仕様に基づく一意のイメージタグの最大数です。
- 3 イメージストリームのステータスに基づく一意のイメージ参照の最大数です。

コアおよび OpenShift Container Platform リソースのどちらも単一の制限範囲オブジェクトで指定できます。ここでは明確にする目的でこれらを 2 つのサンプルに分けています。

17.1.1. コンテナの制限

サポートされるリソース:

- CPU
- メモリー

サポートされる制約:

コンテナごとに設定されます。指定される場合、以下が一致している必要があります。

表17.1 コンテナ

制約	動作
Min	<p>Min[resource]: container.resources.requests[resource] (必須) または container/resources.limits[resource] (オプション) より小さいか等しい</p> <p>設定で min CPU を定義している場合、要求値はその CPU 値よりも大きくなければなりません。制限値を指定する必要はありません。</p>
Max	<p>container.resources.limits[resource] (必須): Max[resource] より小さいか等しい</p> <p>設定で max CPU を定義している場合、要求値を定義する必要はありませんが、CPU 制約の最大値の条件を満たす制限値を設定する必要があります。</p>
MaxLimitRequestRatio	<p>MaxLimitRequestRatio[resource]: (container.resources.limits[resource] / container.resources.requests[resource]) より小さいか等しい</p> <p>設定で maxLimitRequestRatio 値を定義している場合、新規コンテナには要求値および制限値の両方が必要になります。さらに OpenShift Container Platform は、制限を要求で除算して制限の要求に対する比率を算出します。</p> <p>たとえば、コンテナの limit 値が cpu: 500 で、request 値が cpu: 100 である場合、cpu の要求に対する制限の比は 5 になります。この比率は maxLimitRequestRatio より小さいか等しくなければなりません。</p>

サポートされるデフォルト:**Default[resource]**

指定がない場合は **container.resources.limit[resource]** を所定の値にデフォルト設定します。

Default Requests[resource]

指定がない場合は、**container.resources.requests[resource]** を所定の値にデフォルト設定します。

17.1.2. Pod の制限**サポートされるリソース:**

- CPU
- メモリー

サポートされる制約:

Pod のすべてのコンテナにおいて、以下が一致している必要があります。

表17.2 Pod

制約	実施される動作
Min	<code>Min[resource]: container.resources.requests[resource]</code> (必須) または <code>container.resources.limits[resource]</code> (オプション) より小さいか等しい
Max	<code>container.resources.limits[resource]</code> (必須): <code>Max[resource]</code> より小さいか等しい
MaxLimitRequestRatio	<code>MaxLimitRequestRatio[resource]: (container.resources.limits[resource] / container.resources.requests[resource])</code> より小さいか等しい

17.1.3. イメージの制限

サポートされるリソース:

- ストレージ

リソースタイプ名:

- `openshift.io/Image`

イメージごとに設定されます。指定される場合、以下が一致している必要があります。

表17.3 イメージ

制約	動作
Max	<code>image.dockerimagemetadata.size: Max[resource]</code> より小さいか等しい



注記

制限を超える Blob がレジストリーにアップロードされないようにするために、クォータを実施するようレジストリーを設定する必要があります。環境変数

REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA は **true** に設定される必要があります。この設定は、新規デプロイメントについてデフォルトで実行されます。古いデプロイメント設定を更新するには、「[Enforcing quota in the Registry](#)」を参照してください。



警告

イメージのサイズは、アップロードされるイメージのマニフェストで常に表示される訳ではありません。これは、とりわけ Docker 1.10 以上で作成され、v2 レジストリーにプッシュされたイメージの場合に該当します。このようなイメージが古い Docker デーモンでプルされると、イメージマニフェストはレジストリーによってスキーマ v1 に変換されますが、この場合サイズ情報が欠落します。イメージに設定されるストレージの制限がこのアップロードを防ぐことはありません。

現在、[この問題](#)への対応が行われています。

17.1.4. イメージストリームの制限

サポートされるリソース:

- `openshift.io/image-tags`
- `openshift.io/images`

リソースタイプ名:

- `openshift.io/ImageStream`

イメージストリームごとに設定されます。指定される場合、以下が一致している必要があります。

表17.4 ImageStream

制約	動作
<code>Max[openshift.io/image-tags]</code>	<code>length(uniqueimagetags(imagestream.spec.tags))</code> : <code>Max[openshift.io/image-tags]</code> より小さいか等しい <code>uniqueimagetags</code> は、指定された仕様タグのイメージへの一意の参照を返します。
<code>Max[openshift.io/images]</code>	<code>length(uniqueimages(imagestream.status.tags))</code> : <code>Max[openshift.io/images]</code> より小さいか等しい <code>uniqueimages</code> はステータスタグにある一意のイメージ名を返します。名前はイメージのダイジェストに等しくなります。

17.1.4.1. イメージ参照の数

リソース `openshift.io/image-tags` は、一意の[イメージ参照](#)を表します。使用できる参照は、`ImageStreamTag`、`ImageStreamImage` および `DockerImage` になります。それらは `oc tag` および `oc import-image` コマンドを使用するか、または [タグのトラッキング](#)を使用して作成されます。内部参照か外部参照であるかの区別はありませんが、イメージストリームの仕様でタグ付けされる一意の参照はそれぞれ 1 回のみカウントされます。この参照は内部コンテナレジストリーへのプッシュを制限することはありませんが、タグの制限に役立ちます。

リソース `openshift.io/images` は、イメージストリームのステータスに記録される一意のイメージ名を表します。これにより、内部レジストリーにプッシュできるイメージ数を制限できます。内部参照か外部参照であるかの区別はありません。

17.1.5. PersistentVolumeClaim の制限

サポートされるリソース:

- ストレージ

サポートされる制約:

プロジェクトのすべての Persistent Volume Claim (永続ボリューム要求、PVC) において、以下が一致している必要があります。

表17.5 Pod

制約	実施される動作
Min	$\text{Min}[\text{resource}] \leftarrow \text{claim.spec.resources.requests}[\text{resource}]$ (必須)
Max	$\text{claim.spec.resources.requests}[\text{resource}]$ (必須) $\leftarrow \text{Max}[\text{resource}]$

Limit Range オブジェクトの定義

```
{
  "apiVersion": "v1",
  "kind": "LimitRange",
  "metadata": {
    "name": "pvcs" ❶
  },
  "spec": {
    "limits": [{
      "type": "PersistentVolumeClaim",
      "min": {
        "storage": "2Gi" ❷
      },
      "max": {
        "storage": "50Gi" ❸
      }
    }]
  }
}
```

❶ 制限範囲オブジェクトの名前です。

❷ Persistent Volume Claim (永続ボリューム要求、PVC) で要求できるストレージの最小量です。

❸ Persistent Volume Claim (永続ボリューム要求、PVC) で要求できるストレージの最大量です。

17.2. 制限範囲の作成

制限範囲をプロジェクトに適用するには、必要な仕様に基づいてファイルシステムで制限範囲オブジェクトの定義を作成します。以下を実行します。

```
$ oc create -f <limit_range_file> -n <project>
```

17.3. 制限の表示

web コンソールでプロジェクトの **Quota** ページに移動し、プロジェクトで定義される制限範囲を表示できます。

CLI を仕様して制限範囲の詳細を表示することもできます。

1. まず、プロジェクトで定義される制限範囲の一覧を取得します。たとえば、**demoproject** というプロジェクトの場合は以下ようになります。

```
$ oc get limits -n demoproject
NAME              AGE
resource-limits   6d
```

2. 次に、関連のある制限範囲の説明を表示します。たとえば、**resource-limits** 制限範囲の場合は以下ようになります。

```
$ oc describe limits resource-limits -n demoproject
Name:                resource-limits
Namespace:           demoproject
Type                 Resource
Max      Default Request Default Limit  Max Limit/Request Ratio
----      -
Pod
-          -
Pod
1Gi      -          -
Container
200m          300m
Container
1Gi      100Mi      200Mi
openshift.io/Image
1Gi      -          -
openshift.io/ImageStream
-          -
openshift.io/ImageStream
-          -
```

17.4. 制限の削除

プロジェクトの制限を実施しないように有効な制限範囲を削除します。

```
$ oc delete limits <limit_name>
```

第18章 オブジェクトのプルーニング

18.1. 概要

OpenShift Container Platform で作成される **API オブジェクト** は、一定期間が経過すると、アプリケーションのビルドやデプロイなどの通常のユーザー操作によって **etcd データストア** に蓄積されます。

管理者は、不要になった古いバージョンのオブジェクトを OpenShift Container Platform インスタンスから定期的にプルーニングできます。たとえば、イメージのプルーニングにより、使用されなくなったものの、ディスク領域を使用している古いイメージや層を削除できます。

18.2. プルーニングの基本操作

CLI は、共通の親コマンドでプルーニング操作を分類します。

```
$ oc adm prune <object_type> <options>
```

これにより、以下が指定されます。

- **builds**、**deployments**、または **images** などのアクションを実行するための **<object_type>** です。
- オブジェクトタイプのプルーニングの実行においてサポートされる **<options>** です。

18.3. デプロイメントのプルーニング

使用年数やステータスによりシステムで不要となったデプロイメントをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune deployments [<options>]
```

表18.1 デプロイメントのプルーニング用の CLI の設定オプション

オプション	説明
--confirm	ドライランの実行ではなく、プルーニングが実行されることを示します。
--orphans	デプロイメント設定が存在せず、ステータスが complete (完了) または failed (失敗) で、レプリカ数がゼロであるすべてのデプロイメントをプルーニングします。
--keep-complete=<N>	デプロイメント設定に基づき、ステータスが complete (完了) で、レプリカ数がゼロである最後の N デプロイメントを保持します (デフォルト: 5)。
--keep-failed=<N>	デプロイメント設定に基づき、ステータスが failed (失敗) で、レプリカ数がゼロである最後の N デプロイメントを保持します (デフォルト: 1)。

オプション	説明
--keep-younger-than=<duration>	現在の時間との対比で <duration> 未満の新しいオブジェクトはプルーニングしません (デフォルト: 60m)。有効な測定単位には、ナノ秒 (ns)、マイクロ秒 (us)、ミリ秒 (ms)、秒 (s)、分 (m)、および時間 (h) が含まれます。

プルーニング操作によって削除されるものを確認するには、以下を実行します。

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
  --keep-younger-than=60m
```

プルーニング操作を実際に実行するには、以下を実行します。

```
$ oc adm prune deployments --orphans --keep-complete=5 --keep-failed=1 \
  --keep-younger-than=60m --confirm
```

18.4. ビルドのプルーニング

使用年数やステータスによりシステムで不要となったビルドをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune builds [<options>]
```

表18.2 ビルドのプルーニング用の CLI の設定オプション

オプション	説明
--confirm	ドライランの実行ではなく、プルーニングが実行されることを示します。
--orphans	ビルド設定が存在せず、ステータスが complete (完了)、failed (失敗)、error (エラー)、または canceled (中止) のすべてのビルドをプルーニングします。
--keep-complete=<N>	ビルド設定に基づき、ステータスが complete (完了) の最後の N ビルドを保持します (デフォルト: 5)。
--keep-failed=<N>	ビルド設定に基づき、ステータスが failed (失敗)、error (エラー)、または canceled (中止) の最後の N ビルドを保持します (デフォルト: 1)。
--keep-younger-than=<duration>	現在の時間との対比で <duration> 未満の新しいオブジェクトはプルーニングしません (デフォルト: 60m)。

プルーニング操作によって削除されるものを確認するには、以下を実行します。

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
  --keep-younger-than=60m
```

プルーニング操作を実際に実行するには、以下を実行します。

```
$ oc adm prune builds --orphans --keep-complete=5 --keep-failed=1 \
  --keep-younger-than=60m --confirm
```



注記

開発者は、ビルド設定を変更することにより、[自動ビルドプルーニング](#)を有効にできます。

18.5. イメージのプルーニング

使用年数やステータスまたは制限の超過によりシステムで不要となったイメージをプルーニングするために、管理者は以下のコマンドを実行できます。

```
$ oc adm prune images [<options>]
```



注記

現時点でイメージをプルーニングするには、まず [アクセストークン](#) を使ってユーザーとして [CLI にログイン](#) する必要があります。また、ユーザーには [クラスターロール](#) (`system:image-pruner`) 以上のロールがなければなりません (例: `cluster-admin`)。



注記

`--prune-registry=false` が使用されていない限り、イメージのプルーニングにより、統合レジストリーのデータが削除されます。この操作が適切に機能するには、`storage:delete:enabled` が `true` に設定された状態での [レジストリーの設定](#) を確認します。



注記

`--namespace` フラグの付いたイメージをプルーニングしてもイメージは削除されず、イメージストリームのみが削除されます。イメージは namespace を使用しないリソースです。そのため、プルーニングを特定の namespace に制限すると、イメージの現在の使用量を算出できなくなります。

デフォルトで、統合レジストリーは Blob メタデータをキャッシュしてストレージに対する要求数を減らし、要求の処理速度を高めます。プルーニングによって統合レジストリーのキャッシュが更新されることはありません。プルーニング後にプッシュされる、プルーニングされた層を含むイメージは破損します。キャッシュにメタデータを持つプルーニングされた層はプッシュされないためです。したがって、プルーニング後はキャッシュをクリアする必要があります。これは、レジストリーの再デプロイによって実行できます。

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

統合レジストリーが [Redis キャッシュ](#) を使用する場合は、データベースを手動でクリーンアップする必要があります。

プルニング後にレジストリーを再デプロイするのオプションを実行できない場合は、[キャッシュを永久に無効にする](#)必要があります。

表18.3 イメージのプルニング用の CLI の設定オプション

オプション	説明
--all	レジストリーにプッシュされていないものの、プルスルー (pullthrough) でミラーリングされたイメージを組み込みます。これはデフォルトでオンに設定されます。プルニングを統合レジストリーにプッシュされたイメージに制限するには、 --all=false を渡します。
--certificate-authority	OpenShift Container Platform で管理されるレジストリーと通信する際に使用する認証局ファイルへのパスです。デフォルトは現行ユーザーの設定ファイルの認証局データに設定されます。これが指定されている場合、セキュアな通信が実行されます。
--confirm	ドライランを実行する代わりにプルニングが実行されることを示します。これには、統合 Docker レジストリーへの有効なルートが必要になります。このコマンドがクラスターネットワーク外で実行される場合、ルートは --registry-url を使用して指定される必要があります。
--force-insecure	このオプションは注意して使用してください。HTTP 経由でホストされているか、または無効な HTTPS 証明書を持つ Docker レジストリーへの非セキュアな接続を許可します。詳細は、 「セキュアまたは非セキュアな接続の使用」 を参照してください。
--keep-tag-revisions=<N>	それぞれのイメージストリームについては、タグごとに最大 N のイメージリビジョンを保持します (デフォルト: 3)。
--keep-younger-than=<duration>	現在の時間との対比で <duration> 未満の新しいイメージはプルニングしません。現在の時間との対比で <duration> 未満の他のオブジェクトで参照されるイメージはプルニングしません (デフォルト: 60m)。
--prune-over-size-limit	同じプロジェクトに定義される最小の 制限 を超える各イメージをプルニングします。このフラグは --keep-tag-revisions または --keep-younger-than と共に使用することはできません。
--registry-url	レジストリーと通信する際に使用するアドレスです。このコマンドは、管理されるイメージおよびイメージストリームから判別されるクラスター内の URL の使用を試行します。これに失敗する (レジストリーを解決できないか、これにアクセスできない) 場合、このフラグを使用して他の機能するルートを指定する必要があります。レジストリーのホスト名の前には、特定の接続プロトコルを実施する https:// または http:// を付けることができます。

オプション	説明
--prune-registry	他のオプションで規定される条件と共に、このオプションは、OpenShift Container Platform イメージ API オブジェクトに対応するレジストリーのデータがブルーニングされるかどうかを制御します。デフォルトで、イメージのブルーニングは、イメージ API オブジェクトとレジストリーの対応するデータの両方を処理します。このオプションは、イメージオブジェクトの数を減らすなどの目的で etcd の内容のみを削除することを検討していて、レジストリーのストレージのクリーンアップは検討していない場合や、レジストリーの適切なメンテナンス期間中などに レジストリーのハードブルーニング によってこれを別途実行しようとする場合に役立ちます。

18.5.1. イメージのブルーニングの各種条件

- **--keep-younger-than** 分前よりも後に作成され、現時点で以下によって参照されていない「OpenShift Container Platform で管理される」イメージ (アノテーション **openshift.io/image.managed** を持つイメージ) を削除します。
 - **--keep-younger-than** 分前よりも後に作成された Pod。
 - **--keep-younger-than** 分前よりも後に作成されたイメージストリーム。
 - 実行中の Pod。
 - 保留中の Pod。
 - レプリケーションコントローラー。
 - デプロイメント設定。
 - ビルド設定。
 - ビルド。
 - **stream.status.tags[].items** の **--keep-tag-revisions** の最新のアイテム。
- 同じプロジェクトで定義される最小の[制限](#)を超えており、現時点で以下によって参照されていない「OpenShift Container Platform で管理される」イメージ (アノテーション **openshift.io/image.managed** を持つイメージ) を削除します。
 - 実行中の Pod。
 - 保留中の Pod。
 - レプリケーションコントローラー。
 - デプロイメント設定。
 - ビルド設定。
 - ビルド。
- 外部レジストリーからのブルーニングはサポートされていません。

- イメージがプルーニングされる際、イメージのすべての参照は **status.tags** にイメージの参照を持つすべてのイメージストリームから削除されます。
- イメージによって参照されなくなったイメージ層も削除されます。

注記

--prune-over-size-limit は **--keep-tag-revisions** または **--keep-younger-than** フラグと共に使用することができません。これを実行すると、この操作が許可されないことを示す情報が返されます。

注記

--prune-registry=false とその後に [レジストリーのハードプルーニング](#) を実行することで、OpenShift Container Platform イメージ API オブジェクトの削除とイメージデータのレジストリーからの削除を分離することができます。これによりタイミングウィンドウが制限され、1つのコマンドで両方をプルーニングする場合よりも安全に実行できるようになります。ただし、タイミングウィンドウを完全に取り除くことはできません。

たとえばプルーニングの実行時にプルーニング対象のイメージを特定する場合も、そのイメージを参照する Pod を引き続き作成することができます。また、プルーニングの操作時にイメージを参照している可能性のある API オブジェクトを追跡することもできます。これにより、削除されたコンテンツの参照に関連して発生する可能性のある問題を軽減することができます。

また、**--prune-registry** オプションを指定しないか、または **--prune-registry=true** を指定してプルーニングを再実行しても、**--prune-registry=false** を指定して以前にプルーニングされたイメージの、イメージレジストリー内で関連付けられたストレージがプルーニングされる訳ではないことに注意してください。**--prune-registry=false** を指定してプルーニングされたすべてのイメージは、[レジストリーのハードプルーニング](#) によってのみ削除できます。

プルーニング操作によって削除されるものを確認するには、以下を実行します。

1. 最高3つのタグリビジョンを保持し、6分前よりも後に作成されたリソース (イメージ、イメージストリームおよび Pod) を保持します。

```
$ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m
```

2. 定義された制限を超えるすべてのイメージをプルーニングします。

```
$ oc adm prune images --prune-over-size-limit
```

前述のオプションでプルーニング操作を実際に行うには、以下を実行します。

```
$ oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m --confirm
```

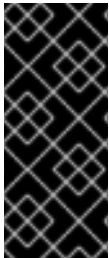
```
$ oc adm prune images --prune-over-size-limit --confirm
```

18.5.2. セキュアまたは非セキュアな接続の使用

セキュアな通信の使用は優先され、推奨される方法です。これは、必須の証明書検証と共に HTTPS 経由で実行されます。**prune** コマンドは、可能な場合は常にセキュアな通信の使用を試行します。これを使用できない場合には、非セキュアな通信にフォールバックすることがあり、これには危険が伴います。この場合、証明書検証は省略されるか、または単純な HTTP プロトコルが使用されます。

非セキュアな通信へのフォールバックは、**--certificate-authority** が指定されていない場合、以下のケースで可能になります。

1. **prune** コマンドが **--force-insecure** オプションと共に実行される。
2. 指定される **registry-url** の前に **http://** スキームが付けられる。
3. 指定される **registry-url** がローカルリンクアドレスまたは **localhost** である。
4. 現行ユーザーの設定が非セキュアな接続を許可する。これは、ユーザーが **--insecure-skip-tls-verify** を使用してログインするか、またはプロンプトが出される際に非セキュアな接続を選択することによって生じる可能性があります。



重要

レジストリーのセキュリティーが、OpenShift Container Platform で使用されるものとは異なる認証局で保護される場合、これを **--certificate-authority** フラグを使用して指定する必要があります。そうしないと、**prune** コマンドは、「[正しくない認証局の使用](#)」または「[セキュリティーが保護されたレジストリーに対する非セキュアな接続の使用](#)」で一覧表示されているエラーと同様のエラーを出して失敗します。

18.5.3. イメージのプルーニングに関する問題

イメージがプルーニングされない

イメージが蓄積し続け、**prune** コマンドが予想よりも小規模な削除を実行する場合、プルーニング候補のイメージについて満たすべき条件があることを確認します。

とくに削除する必要のあるイメージが、それぞれの[タグ履歴](#)において選択したタグリビジョンのしきい値よりも高い位置にあることを確認します。たとえば、**sha:abz** という名前の古く陳腐化したイメージがあるとします。イメージがタグ付けされている namespace **N** で以下のコマンドを実行すると、イメージが **myapp** という単一イメージストリームで 3 回タグ付けされていることに気づかれるでしょう。

```
$ image_name="sha:abz"
$ oc get is -n N -o go-template='{{range $isi, $is := .items}}{{range $ti, $tag := $is.status.tags}}\n    '{{range $ii, $item := $tag.items}}{{if eq $item.image\n    ""$image_name}}\n    '$'}}{{{$is.metadata.name}}:{{$tag.tag}} at position {{$ii}} out of {{len $tag.items}}\n'\n    '{{end}}{{end}}{{end}}{{end}}'\nmyapp:v2 at position 4 out of 5\nmyapp:v2.1 at position 2 out of 2\nmyapp:v2.1-may-2016 at position 0 out of 1
```

デフォルトオプションが使用される場合、イメージは **myapp:v2.1-may-2016** タグの履歴の **0** の位置にあるためプルーニングされません。イメージがプルーニングの対象と見なされるようにするには、管理者は以下を実行する必要があります。

1. **oc adm prune images** コマンドで **--keep-tag-revisions=0** を指定します。

注意

このアクションを実行すると、イメージが指定されたしきい値よりも新しいか、またはこれよりも新しいオブジェクトによって参照されていない限り、すべてのタグが基礎となるイメージと共にすべての namespace から削除されます。

2. リビジョンのしきい値より下にあるすべての **istag**、つまり **myapp:v2.1** および **myapp:v2.1-may-2016** を削除します。
3. 同じ **istag** にプッシュする新規ビルドを実行するか、または他のイメージをタグ付けしてイメージを履歴内でさらに移動させます。ただし、これは古いリリースタグの場合には常に適切な操作となる訳ではありません。

特定のイメージのビルド日時が名前の一部になっているタグは、その使用を避ける必要があります (イメージが未定義の期間保持される必要がある場合を除きます)。このようなタグは履歴内で1つのイメージのみに関連付けられる可能性があり、その場合にこれらをプルーニングできなくなります。詳細は、**istag** の命名規則を参照してください。

非セキュアなレジストリーに対するセキュアな接続の使用

oc adm prune images の出力で以下のようなメッセージが表示される場合、レジストリーのセキュリティは保護されておらず、**oc adm prune images** クライアントがセキュアな接続の使用を試行することを示しています。

```
error: error communicating with registry: Get
https://172.30.30.30:5000/healthz: http: server gave HTTP response to
HTTPS client
```

1. 推奨される解決法として、**レジストリーのセキュリティを保護**することができます。これが必要でない場合には、**--force-insecure** をコマンドに追加して、クライアントが非セキュアな接続の使用を強制することができます (**推奨される方法ではありません**)。

18.5.3.1. セキュリティーが保護されたレジストリーに対する非セキュアな接続の使用

oc adm prune images コマンドの出力に以下のエラーのいずれかが表示される場合、レジストリーのセキュリティ保護に使用されている認証局で署名された証明書が、接続の検証用に **oc adm prune images** クライアントで使用されるものとは異なることを意味します。

```
error: error communicating with registry: Get
http://172.30.30.30:5000/healthz: malformed HTTP response
"\x15\x03\x01\x00\x02\x02"
error: error communicating with registry: [Get
https://172.30.30.30:5000/healthz: x509: certificate signed by unknown
authority, Get http://172.30.30.30:5000/healthz: malformed HTTP response
"\x15\x03\x01\x00\x02\x02"]
```

デフォルトでは、ユーザーの接続ファイルに保存されている認証局データが使用されます。これはマスター API との通信の場合も同様です。

--certificate-authority オプションを使用して Docker レジストリーサーバーに適切な認証局を指定します。

正しくない認証局の使用

以下のエラーは、セキュリティが保護された Docker レジストリーの証明書の署名に使用される認証局がクライアントで使用される認証局とは異なることを示しています。

```
error: error communicating with registry: Get https://172.30.30.30:5000/:
x509: certificate signed by unknown authority
```

フラグ **--certificate-authority** を使用して適切な認証局を指定します。

回避策として、**--force-insecure** フラグを代わりに追加することもできます (推奨される方法ではありません)。

18.6. レジストリーのハードプルーニング

OpenShift Container レジストリーは、OpenShift Container Platform クラスターの etcd で参照されない Blob を蓄積します。基本的なイメージプルーニングの手順はこれらに対応しません。これらの Blob は孤立した **Blob** と呼ばれています。

孤立した Blob は以下のシナリオで発生する可能性があります。

- **oc delete image <sha256:image-id>** コマンドを使ってイメージを手動で削除すると、etcd のイメージのみが削除され、レジストリーのストレージからは削除されない。
- **docker** デーモンの障害によって生じるレジストリーへのプッシュにより、一部の Blob はアップロードされるものの、(最後のコンポーネントとしてアップロードされる) イメージマニフェストはアップロードされない。固有のイメージ Blob すべては孤立する。
- OpenShift Container Platform がクォータの制限によりイメージを拒否する。
- 標準のイメージプルーナーがイメージマニフェストを削除するが、関連する Blob を削除する前に中断される。
- 対象の Blob を削除できないというレジストリープルーナーのバグにより、それらを参照するイメージオブジェクトは削除されるが、Blob は孤立する。

基本的なイメージプルーニングとは異なるレジストリーの **ハードプルーニング** により、孤立した Blob を削除することができます。OpenShift Container レジストリーのストレージ領域が不足している場合や、孤立した Blob があると思われる場合にはハードプルーニングを実行する必要があります。

これは何度も行う操作ではなく、多数の孤立した Blob が新たに作成されているという証拠がある場合にのみ実行する必要があります。または、(作成されるイメージの数によって異なりますが) 1 日 1 回などの定期的な間隔で標準のイメージプルーニングを実行することもできます。

孤立した Blob をレジストリーからハードプルーニングするには、以下を実行します。

1. **ログイン:** CLI を使用し、[アクセストークン](#)を持つユーザーとしてログインします。
2. **基本的なイメージプルーニングの実行:** 基本的なイメージプルーニングにより、不要になった追加のイメージが削除されます。ハードプルーニングによってイメージが削除される訳ではなく、レジストリーストレージに保存された Blob のみが削除されます。したがって、ハードプルーニングの実行前にこれを実行する必要があります。
手順については、「[イメージのプルーニング](#)」を参照してください。
3. **レジストリーの読み取り専用モードへの切り替え:** レジストリーが読み取り専用モードで実行されていない場合、プルーニングと同時に実行されているプッシュの結果は以下のいずれかになります。
 - 失敗する。さらに孤立した Blob が新たに発生する。
 - 成功する。ただし、(参照される Blob の一部が削除されたため) イメージをプルできない。

プッシュは、レジストリーが読み取り書き込みモードに戻されるまで成功しません。したがって、ハードブルーニングは注意してスケジューリングする必要があります。

レジストリーを読み取り専用モードに切り換えるには、以下を実行します。

- a. 以下の環境変数を設定します。

```
$ oc env -n default \
    dc/docker-registry \
    'REGISTRY_STORAGE_MAINTENANCE_READONLY={"enabled":true}'
```

- b. デフォルトで、レジストリーは直前の手順が完了すると自動的に再デプロイするはずで、再デプロイが完了するのを待機してから次に進んでください。ただし、これらのトリガーを無効にしている場合は、レジストリーを手動で再デプロイし、新規の環境変数が選択されるようにする必要があります。

```
$ oc rollout -n default \
    latest dc/docker-registry
```

4. **system:image-pruner ロールの追加:** 一部のリソースを一覧表示するには、レジストリーインスタンスの実行に使用するサービスアカウントに追加のパーミッションが必要になります。

- a. サービスアカウント名を取得します。

```
$ service_account=$(oc get -n default \
    -o jsonpath='{$system:serviceaccount:{.metadata.namespace}:
    {.spec.template.spec.serviceAccountName}}\n' \
    dc/docker-registry)
```

- b. **system:image-pruner** クラスターロールをサービスアカウントに追加します。

```
$ oc adm policy add-cluster-role-to-user \
    system:image-pruner \
    ${service_account}
```

5. (オプション) プルーナーのドライランモードでの実行: 削除される Blob の数を確認するには、ドライランモードでハードプルーナーを実行します。これにより変更が加えられることはありません。

```
$ oc -n default \
    exec -i -t "$(oc -n default get pods -l deploymentconfig=docker-
    registry \
    -o jsonpath='{$.items[0].metadata.name}}\n')" \
    -- /usr/bin/dockerregistry -prune=check
```

または、ブルーニング候補の実際のパスを取得するには、ロギングレベルを上げます。

```
$ oc -n default \
    exec "$(oc -n default get pods -l deploymentconfig=docker-
    registry \
    -o jsonpath='{$.items[0].metadata.name}}\n')" \
    -- /bin/sh \
    -c 'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -
    prune=check'
```

出力サンプル (切り捨て済み)

```
$ oc exec docker-registry-3-vhndw \
  -- /bin/sh -c 'REGISTRY_LOG_LEVEL=info /usr/bin/dockerregistry -
prune=check'

time="2017-06-22T11:50:25.066156047Z" level=info msg="start prune
(dry-run mode)" distribution_version="v2.4.1+unknown"
kubernetes_version=v1.6.1+$Format:%h$ openshift_version=unknown
time="2017-06-22T11:50:25.092257421Z" level=info msg="Would delete
blob:
sha256:00043a2a5e384f6b59ab17e2c3d3a3d0a7de01b2cabeb606243e468acc663
fa5" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:25.092395621Z" level=info msg="Would delete
blob:
sha256:0022d49612807cb348cab562c072ef34d756adfe0100a61952cbcb87ee65
78a" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:25.092492183Z" level=info msg="Would delete
blob:
sha256:0029dd4228961086707e53b881e25eba0564fa80033fbbb2e27847a28d16a
37c" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:26.673946639Z" level=info msg="Would delete
blob:
sha256:ff7664dfc213d6cc60fd5c5f5bb00a7bf4a687e18e1df12d349a1d07b2cf7
663" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:26.674024531Z" level=info msg="Would delete
blob:
sha256:ff7a933178ccd931f4b5f40f9f19a65be5eeec207e4fad2a5bafd28afbef
57e" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
time="2017-06-22T11:50:26.674675469Z" level=info msg="Would delete
blob:
sha256:ff9b8956794b426cc80bb49a604a0b24a1553aae96b930c6919a6675db3d5
e06" go.version=go1.7.5 instance.id=b097121c-a864-4e0c-ad6c-
cc25f8fdf5a6
...
Would delete 13374 blobs
Would free up 2.835 GiB of disk space
Use -prune=delete to actually delete the data
```

6. **ハードプルーニングの実行:** ハードプルーニングを実行するには、**docker-registry** Pod の実行中インスタンスで以下のコマンドを実行します。

```
$ oc -n default \
  exec -i -t "$(oc -n default get pods -l deploymentconfig=docker-
registry -o jsonpath='{$.items[0].metadata.name}\n')" \
  -- /usr/bin/dockerregistry -prune=delete
```

出力サンプル

```
$ oc exec docker-registry-3-vhndw \
```

```
-- /usr/bin/dockerregistry -prune=delete
```

```
Deleted 13374 blobs
Freed up 2.835 GiB of disk space
```

7. レジストリーを読み取り書き込みモードに戻す: プルーニングの終了後は、以下を実行してレジストリーを読み取り書き込みモードに戻すことができます。

```
$ oc env -n default dc/docker-registry
REGISTRY_STORAGE_MAINTENANCE_READONLY-
```

18.7. CRON ジョブのプルーニング

重要

cron ジョブについては、現時点で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。

cron ジョブは正常なジョブのプルーニングを実行できますが、失敗したジョブを適切に処理しない可能性があります。そのため、クラスター管理者は定期的な**ジョブのクリーンアップ**を手動で実行する必要があります。また、信頼できるユーザーの小規模なグループに cron ジョブへの**アクセスを制限**し、ジョブや Pod が作成され過ぎないように適切な**クォータ**を設定することも推奨されます。

第19章 カスタムリソースによる KUBERNETES API の拡張

Kubernetes API では、リソースは特定の種類の API オブジェクトのコレクションを保管するエンドポイントです。たとえば、ビルトインされた Pod リソースには Pod オブジェクトのコレクションが含まれます。

カスタムリソースは、Kubernetes API を拡張するか、またはプロジェクトまたはクラスターに独自の API を導入することを可能にするオブジェクトです。

カスタムリソース定義 (CRD) ファイルは、独自のオブジェクトの種類を定義し、API サーバーがライフサイクル全体を処理できるようにします。CRD をクラスターにデプロイすると、Kubernetes API サーバーは指定されたカスタムリソースを提供し始めます。

新規のカスタムリソース定義 (CRD) の作成時に、Kubernetes API サーバーは、クラスター全体または単一プロジェクト (namespace) でアクセスできる新規 RESTful リソースパスを作成することによって応答します。既存のビルトインオブジェクトの場合のように、プロジェクトを削除すると、そのプロジェクトのすべてのカスタムオブジェクトが削除されます。

19.1. カスタムリソース定義の作成

CRD を作成するには、YAML ファイルを開き、以下の例のようにフィールドに入力します。

カスタムリソース定義の YAML ファイルサンプル

```
apiVersion: apiextensions.k8s.io/v1beta1 ❶
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com ❷
spec:
  group: stable.example.com ❸
  version: v1 ❹
  scope: Namespaced ❺
  names:
    plural: crontabs ❻
    singular: crontab ❼
    kind: CronTab ❽
    shortNames:
      - ct ❾
```

- ❶ **apiextensions.k8s.io/v1beta1** API を使用します。
- ❷ 定義の名前を指定します。これは **group** および **plural** フィールドの値を使用する `<plural-name><group>` 形式である必要があります。
- ❸ API のグループ名を指定します。API グループは、論理的に関連付けられるオブジェクトのコレクションです。たとえば、**Job** または **ScheduledJob** などのすべてのバッチオブジェクトはバッチ API グループ (`batch.api.example.com` など) である可能性があります。組織の完全修飾ドメイン名を使用することが奨励されます。
- ❹ URL で使用されるバージョン名を指定します。それぞれの API グループは複数バージョンで存在させることができます。たとえば、**v1alpha**、**v1beta**、**v1** などが使用されます。
- ❺ カスタムオブジェクトがクラスター (**Cluster**) の 1 つのプロジェクト (**Namespaced**) またはすべてのプロジェクトで利用可能であるかどうかを指定します。

- 6 URL で使用される複数形の名前を指定します。**plural** フィールドは API URL のリソースと同じになります。
- 7 CLI および表示用にエイリアスとして使用される単数形の名前を指定します。
- 8 作成できるオブジェクトの種類を指定します。タイプは CamelCase にすることができます。
- 9 CLI でリソースに一致する短い文字列を指定します。



注記

デフォルトで、カスタムリソース定義のスコープはクラスターに設定され、すべてのプロジェクトで利用可能です。

定義ファイルの設定後にオブジェクトを定義します。

```
oc create -f <file-name>.yaml
```

新規の RESTful API エンドポイントは以下のように作成されます。

```
/apis/<spec:group>/<spec:version>/<scope>/*/<names-plural>/...
```

たとえば、サンプルファイルを使用すると、以下のエンドポイントが作成されます。

```
/apis/stable.example.com/v1/namespaces/*/crontabs/...
```

次にこのエンドポイント URL はカスタムオブジェクトを作成し、管理するために使用できます。オブジェクトの種類は、作成したカスタムリソース定義オブジェクトの **spec.kind** フィールドに基づくものです。

19.2. カスタムオブジェクトの作成

カスタムリソース定義オブジェクトの作成後に、カスタムオブジェクトを作成することができます。

カスタムオブジェクトにはカスタムフィールドを含めることができます。これらのフィールドには任意の JSON を含めることができます。

以下の例では、**cronSpec** および **image** カスタムフィールドが **CronTab** という種類のカスタムオブジェクトに設定されます。**CronTab** という種類は、上記で作成したカスタムリソース定義オブジェクトの **spec.kind** フィールドから取られています。

カスタムオブジェクトの YAML ファイルサンプル

```
apiVersion: "stable.example.com/v1" 1
kind: CronTab 2
metadata:
  name: my-new-cron-object 3
spec: 4
  cronSpec: "* * * * /5"
  image: my-awesome-cron-image
```


- ❶ カスタムリソース定義からグループ名および API バージョン (名前/バージョン) を指定します。
- ❷ カスタムリソース定義のタイプを指定します。
- ❸ オブジェクトの名前を指定します。
- ❹ オブジェクトのタイプに固有の条件を指定します。

オブジェクトファイルの設定後に、オブジェクトを作成します。

```
oc create -f <file-name>.yaml
```

19.3. カスタムオブジェクトの管理

次に、カスタムリソースを管理することができます。

特定の種類のカスタムリソースについての情報を取得するには、以下を入力します。

```
oc get <kind>
```

以下に例を示します。

```
oc get crontab

NAME                                KIND
my-new-cron-object    CronTab.v1.stable.example.com
```

リソース名では大文字と小文字が区別されず、CRD で定義される単数形または複数形のいずれか、および任意の短縮名を指定できることに注意してください。以下は例になります。

```
oc get crontabs
oc get crontab
oc get ct
```

未加工 JSON データを表示することもできます。

```
oc get <kind> -o yaml
```

これを作成するために使用した YAML のカスタム <1> **cronSpec** および <2> **image** フィールドが含まれることを確認できるはずです。

```
oc get ct -o yaml

apiVersion: v1
items:
- apiVersion: stable.example.com/v1
  kind: CronTab
  metadata:
    clusterName: ""
    creationTimestamp: 2017-05-31T12:56:35Z
    deletionGracePeriodSeconds: null
    deletionTimestamp: null
    name: my-new-cron-object
```



```

namespace: default
resourceVersion: "285"
selfLink: /apis/stable.example.com/v1/namespaces/default/crontabs/my-
new-cron-object
uid: 9423255b-4600-11e7-af6a-28d2447dc82b
spec:
  cronSpec: '* * * * /5' ❶
  image: my-awesome-cron-image ❷

```

19.4. ファイナライザー

カスタムオブジェクトは **ファイナライザー** をサポートします。これにより、コントローラーはオブジェクトの削除前に満たしている必要のある条件を実装できます。

以下のようにファイナライザーをカスタムオブジェクトに追加できます。

```

apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  finalizers:
    - finalizer.stable.example.com

```

ファイナライザーが設定されたオブジェクトでの最初の削除要求により、オブジェクトの削除ではなく **metadata.deletionTimestamp** フィールドの値が設定されます。これによりコントローラーがトリガーされ、コントローラーはオブジェクトによる対象ファイナライザーの実行を監視します。

次に、各コントローラーは一覧からファイナライザーを削除し、削除要求を再び発行します。この要求により、ファイナライザーの一覧が空である (つまり、すべてのファイナライザーが実行済みである) 場合にのみオブジェクトが削除されます。

第20章 ガベージコレクション

20.1. 概要

OpenShift Container Platform ノードは、2 種類のガベージコレクションを実行します。

- **コンテナのガベージコレクション**: 終了したコンテナを削除します。
- **イメージのガベージコレクション**: 実行中の Pod によって参照されていないイメージを削除します。

20.2. コンテナのガベージコレクション

コンテナのガベージコレクションはデフォルトで有効にされ、エビクションのしきい値に達すると自動的に実行されます。ノードは Pod のコンテナを API からアクセス可能な状態にしようとします。Pod が削除された場合、コンテナも削除されます。コンテナは Pod が削除されておらず、エビクションのしきい値に達していない限り保持されます。ノードがディスク不足 (disk pressure) の状態にある場合、コンテナが削除され、それらのログは **oc logs** でアクセスできなくなります。

コンテナのガベージコレクションのポリシーは 3 つのノード設定に基づいています。

設定	説明
minimum-container-ttl-duration	コンテナがガベージコレクションの対象となるのに必要な最小の年数です。デフォルトは 0 です。制限なしにするには 0 を使用します。この設定の値は、時間の h 、分の m 、秒の s などの単位のサフィックスを使用して指定することができます。
maximum-dead-containers-per-container	Pod コンテナごとに保持するインスタンス数です。デフォルトは 1 です。
maximum-dead-containers	ノードにある実行されないコンテナの合計の最大数です。デフォルトは、無制限を意味する -1 です。

競合が生じる場合、**maximum-dead-containers** 設定は **maximum-dead-containers-per-container** 設定よりも優先されます。たとえば、**maximum-dead-containers-per-container** の数を保持することでコンテナの合計数が **maximum-dead-containers** より大きくなる場合、最も古いコンテナが削除され、**maximum-dead-containers** の制限が満たされるようにします。

ノードが実行されていないコンテナを削除すると、それらのコンテナの内部にあるすべてのファイルも削除されます。そのノードで作成されたコンテナに対してのみガベージコレクションが実行されます。

ノードホストにある **/etc/origin/node/node-config.yaml** ファイルの **kubeletArguments** セクションでこれらの設定の値を指定します。このセクションがすでに存在しない場合には、これを追加します。

コンテナのガベージコレクション設定

```
kubeletArguments:
  minimum-container-ttl-duration:
    - "10s"
```

```
maximum-dead-containers-per-container:
  - "2"
maximum-dead-containers:
  - "240"
```

20.2.1. 削除するコンテナの検出

ガベージコレクターの各グループでは、以下の手順が実行されます。

1. 利用可能なコンテナの一覧を取得します。
2. 実行中であるか、または **minimum-container-ttl-duration** パラメーターよりも長く存続していないすべてのコンテナをフィルターに掛けます。
3. 残りのすべてのコンテナを、Pod およびイメージ名のメンバーシップに基づいて同等のクラスに分類します。
4. 特定されないコンテナ (kubelet で管理されているコンテナであるが、それらの名前の形式が正しくないコンテナ) をすべて削除します。
5. **maximum-dead-containers-per-container** パラメーターよりも多くのコンテナが含まれるそれぞれのクラスについて、そのクラスのコンテナを作成時間で並び替えます。
6. **maximum-dead-containers-per-container** パラメーターの条件が満たされるまで、コンテナを最も古いものから順に削除し始めます。
7. 依然として **maximum-dead-containers** パラメーターよりも多くのコンテナが一覧にある場合、コレクターは各クラスのコンテナの削除を開始し、それぞれのクラスにあるコンテナ数がクラスあたりのコンテナの平均数、または **<all_remaining_containers>/<number_of_classes>** よりも大きくなならないようにします。
8. これでも十分でない場合は、一覧にあるすべてのコンテナを並び替えて、**maximum-dead-containers** の条件を満たすまで、コンテナを古いものから順番に削除し始めます。

重要

各種のニーズに合わせてデフォルト設定を更新してください。

ガベージコレクションは、関連付けられている Pod のないコンテナのみを削除します。

20.3. イメージのガベージコレクション

イメージのガベージコレクションでは、ノードの **cAdvisor** によって報告されるディスク使用量に基づいて、ノードから削除するイメージを決定します。この場合、以下の設定が考慮に入れます。

設定	説明
image-gc-high-threshold	イメージのガベージコレクションをトリガーするディスク使用量のパーセント (整数で表される) です。デフォルトは 85 です。

設定	説明
image-gc-low-threshold	イメージのガベージコレクションが解放しようとするディスク使用量のパーセント (整数で表される) です。デフォルトは 80 です。

ノードホストにある `/etc/origin/node/node-config.yaml` ファイルの `kubeletArguments` セクションでこれらの設定の値を指定します。このセクションがすでに存在しない場合には、これを追加します。

イメージのガベージコレクション設定

```
kubeletArguments:
  image-gc-high-threshold:
    - "85"
  image-gc-low-threshold:
    - "80"
```

20.3.1. 削除するイメージの検出

以下の 2 つのイメージ一覧がそれぞれのガベージコレクターの実行で取得されます。

- 1 つ以上の Pod で現在実行されているイメージの一覧
- ホストで利用可能なイメージの一覧

新規コンテナの実行時に新規のイメージが表示されます。すべてのイメージにはタイムスタンプのマークが付けられます。イメージが実行中 (上記の最初の一覧) か、または新規に検出されている (上記の 2 番目の一覧) 場合、これには現在の時間のマークが付けられます。残りのイメージには以前のタイムスタンプのマークがすでに付けられています。すべてのイメージはタイムスタンプで並び替えられます。

コレクションが開始されると、停止条件を満たすまでイメージが最も古いものから順番に削除されます。

第21章 ノードリソースの割り当て

21.1. 概要

より信頼性の高いスケジューリングを提供し、ノードにおけるリソースのオーバーコミットを最小限にするために、各ノードは、ホスト上のすべての基礎となる **ノードコンポーネント** (例: kubelet、kube-proxy、Docker) および残りのシステムコンポーネント (例: **sshd**、**NetworkManager**) に使用される分をリソースの一部として保持できます。このリソースが指定されると、スケジューラーは、ノードが Pod に割り当てたリソース (例: メモリー、CPU) についての詳細な情報を取得できます。

21.2. 割り当てられるリソースについてのノードの設定

ノードコンポーネント用に予約されたリソースは 2 つのノード設定に基づきます。

設定	説明
kube-reserved	ノードコンポーネント用に予約されたリソースです。デフォルトは none です。
system-reserved	残りのシステムコンポーネント用に予約されたリソースです。デフォルトは none です。

これらは、`<resource_type>=<resource_quantity>` ペアのセット (例: **cpu=200m,memory=512Mi**) を使用して **ノード設定ファイル** (デフォルトでは `/etc/origin/node/node-config.yaml` ファイル) の **kubeletArguments** セクションに設定することができます。このセクションは、すでに存在しない場合は追加します。

例21.1 ノードの割り当て可能なリソースの設定

```
kubeletArguments:
  kube-reserved:
    - "cpu=200m,memory=512Mi"
  system-reserved:
    - "cpu=200m,memory=512Mi"
```

現時点で、**cpu** および **memory** のリソースタイプがサポートされています。**cpu** の場合、リソースの量はコアの単位で指定されます (例: 200m、0.5、1)。**memory** の場合、これはバイト単位で指定されます (例: 200Ki、50Mi、5Gi)。

詳細は、「**コンピュートリソース**」を参照してください。

フラグが設定されていない場合、これはデフォルトで **0** になります。フラグのいずれも設定されていない場合、割り当てられるリソースは、割り当て可能なリソースの導入前であるためにノードの容量に設定されます。

21.3. 割り当てられるリソースの計算

リソースの割り当てられる量は以下の数式に基づいて計算されます。

```
[Allocatable] = [Node Capacity] - [kube-reserved] - [system-reserved] -
[Hard-Eviction-Thresholds]
```



注記

Hard-Eviction-Thresholds を割り当て可能なリソースから差し引く点はシステムの信頼性を強化するための動作の変更点です。この値に基づいて割り当て可能なリソースをノードレベルでエンドユーザー Pod に対して適用できます。**experimental-allocatable-ignore-eviction** 設定は、レガシー動作を保持するために利用できますが、今後のリリースでは非推奨となります。

[Allocatable] が負の値の場合、これは **0** に設定されます。

21.4. ノードの割り当て可能なリソースおよび容量の表示

ノードの現在の容量および割り当て可能なリソースを表示するには、以下を実行できます。

```
$ oc get node/<node_name> -o yaml
...
status:
...
  allocatable:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
  capacity:
    cpu: "4"
    memory: 8010948Ki
    pods: "110"
...
```

21.5. ノードによって報告されるシステムリソース

OpenShift Container Platform 3.3 より、各ノードはコンテナランタイムおよび kubelet によって利用されるシステムリソースについて報告します。--**system-reserved** および --**kube-reserved** の設定をより容易に行えるようにするには、ノード要約 API を使用して対応するノードのリソース使用状況をインストロスペクトできます。この API は **<master>/api/v1/nodes/<node>/proxy/stats/summary** からアクセスできます。

たとえば、**cluster.node22** ノードからリソースにアクセスするには、以下を実行できます。

```
$ curl <certificate details>
https://<master>/api/v1/nodes/cluster.node22/proxy/stats/summary
{
  "node": {
    "nodeName": "cluster.node22",
    "systemContainers": [
      {
        "cpu": {
          "usageCoreNanoSeconds": 929684480915,
          "usageNanoCores": 190998084
        },

```

```

        "memory": {
            "rssBytes": 176726016,
            "usageBytes": 1397895168,
            "workingSetBytes": 1050509312
        },
        "name": "kubelet"
    },
    {
        "cpu": {
            "usageCoreNanoSeconds": 128521955903,
            "usageNanoCores": 5928600
        },
        "memory": {
            "rssBytes": 35958784,
            "usageBytes": 129671168,
            "workingSetBytes": 102416384
        },
        "name": "runtime"
    }
]
}

```

証明書の詳細については、「[REST API Overview](#)」を参照してください。

21.6. ノードの実施

ノードは、Pod が設定された割り当て可能な値に基づいて消費できるリソースの合計量を制限できます。この機能は、Pod がリソースのシステムサービス (例: コンテナランタイム、ノードエージェントなど) のリソース不足の状態を防ぎ、ノードの信頼性を大幅に強化します。管理者におかれましては、ノードの信頼性を強化するために必要なノードの使用状況のターゲットに基づいてリソースを予約することを強く奨励します。

ノードは、QoS (Quality of Service) を実施する新規の cgroup 階層を使用してリソースの制約を実施します。すべての Pod は、システムデーモンと切り離された専用の cgroup 階層で起動します。

この機能を設定するために、以下の kubelet 引数を指定します。

例21.2 ノードの cgroup 設定

```

kubeletArguments:
  cgroups-per-qos:
    - "true" ❶
  cgroup-driver:
    - "systemd" ❷
  enforce-node-allocatable:
    - "pods" ❸

```

❶ ❶ ノードによって管理される新規の cgroup 階層を有効化または無効化します。この設定の変更にはノードの完全なドレイン (解放) が必要になります。このフラグは、ノードがノードの割り当て可能な分を実施できるようにするには true である必要があります。ユーザーにはこの値を変更しないようにすることを推奨します。

❷ ❷

cgroup 階層を管理する際にノードで使用する cgroup ドライバーです。この値はコンテナランタイムに関連付けられたドライバーに一致する必要があります。有効な値は **systemd** お

3

ノードがノードのリソース制約を実施するスコープのカンマ区切りの一覧です。有効な値は **pods**、**system-reserved**、および **kube-reserved** です。デフォルトは **pods** です。ユーザーにはこの値を変更しないようにすることを推奨します。

オプションとして、`enforce-node-allocatable` フラグでトークンを指定することで、ノードが kube で予約される制限およびシステムで予約される制限を実施するようにできます。これらが指定される場合、対応する **--kube-reserved-cgroup** または **--system-reserved-cgroup** を指定する必要があります。今後のリリースでは、ノードおよびコンテナランタイムは **system.slice** と切り離された共通の cgroup でパッケージ化されます。パッケージ化されるまでは、ユーザーは `enforce-node-allocatable` フラグのデフォルト値を変更しないようにすることを推奨します。

管理者は Guaranteed Pod と同様にシステムデーモンを処理する必要があります。システムデーモンは、境界となる制御グループ内で予想外の動作をする可能性があり、この動作はクラスターデプロイメントの一部として管理される必要があります。システム予約の制限を実施することにより、ノード上の重要なシステムサービスで CPU が不足したり、OOM による強制終了が生じる可能性があります。そのため、オペレーターが正確な推定値を判別するためにノードの徹底的なプロファイリングを実行した場合や、そのグループのプロセスで OOM による強制終了が発生した場合に確実に復元できる場合にのみシステム予約の制限を実施することが推奨されます。

したがって、ユーザーはデフォルトで **pods** に対してのみノードの割り当て可能分を実施し、ノード全体の信頼性を維持するためにシステムデーモンの適切な予約を確保しておくことを強くお勧めします。

21.7. エビクションしきい値

ノードがメモリー不足の状態にある場合、ノード全体に、またノードで実行されているすべての Pod に影響が及ぶ可能性があります。システムデーモンによるメモリーの使用がメモリーの予約されている量を超える場合、OOM イベントが発生し、ノード全体やノードで実行されているすべての Pod に影響が及び可能性があります。システムの OOM を防止する (またはその発生可能性を下げる) ために、ノードは [Out Of Resource Handling \(リソース不足の処理\)](#) を行います。

--eviction-hard フラグで一部のメモリーを確保することで、ノードは、ノードのメモリー可用性が絶対値またはパーセンテージを下回る場合は常に Pod のエビクトを試行します。システムデーモンがノードに存在しない場合、Pod はメモリーの **capacity - eviction-hard** に制限されます。このため、メモリー不足の状態になる前にエビクションのバッファとして確保されているリソースは Pod で利用することはできません。

以下の例は、ノードの割り当て可能分のメモリーに対する影響について説明しています。

- ノード容量: **32Gi**
- **--kube reserved: 2Gi**
- **--system-reserved: 1Gi**
- **--eviction-hard** は **<100Mi** に設定される。

このノードについては、有効なノードの割り当て可能な値は **28.9Gi** です。ノードおよびシステムコンポーネントが予約分をすべて使い切る場合、Pod に利用可能なメモリーは **28.9Gi** となり、この使用量を超える場合に kubelet は Pod をエビクトします。

トップレベルの cgroup でノードの割り当て可能分 (**28.9Gi**) を実施する場合、Pod は **28.9Gi** を超えることはできません。エビクションは、システムデーモンが **3.1Gi** よりも多くのメモリーを消費しない限り実行されません。

上記の例ではシステムデーモンが予約分すべてを使い切らない場合も、ノードのエビクションが開始される前に、Pod では境界となる cgroup からの memcg OOM による強制終了が発生します。この状況で QoS をより効果的に実行するには、ノードですべての Pod のトップレベルの cgroup に対し、ハードエビクションしきい値が **Node Allocatable + Eviction Hard Thresholds** になるよう適用できます。

システムデーモンがすべての予約分を使い切らない場合で、Pod が **28.9Gi** を超えるメモリーを消費する場合、ノードは Pod を常にエビクトします。エビクションが時間内に生じない場合には、Pod が **29Gi** のメモリーを消費すると OOM による強制終了が生じます。

21.8. スケジューラー

スケジューラーは、**node.Status.Capacity** ではなく **node.Status.Allocatable** の値を使用して、ノードが Pod スケジューリングの候補になるかどうかを判別します。

デフォルトで、ノードはそのマシン容量をクラスターで完全にスケジュール可能であるとして報告します。

第22章 不透明な整数リソース

22.1. 概要

不透明な整数リソースは、クラスターのオペレーターがシステムで認識されない新規のノードレベルのリソースを提供することを可能にします。ユーザーは CPU やメモリーと同様に Pod 仕様にあるこれらのリソースを消費できます。スケジューラーは、利用可能な量を上回るリソースが複数の Pod に同時に割り当てられないようにリソースアカウンティングを実行します。



注記

不透明な整数リソースは現時点でアルファ機能であり、リソースアカウンティングのみが実装されています。これらのリソースについてのリソースクォータや制限範囲のサポートはなく、これらが QoS に影響を与えることはありません。

不透明な整数リソースが **不透明 (opaque)** と言われるのは、OpenShift Container Platform がリソースが何を認識しない状態でも、そのリソースが十分にある場合に Pod をノードにスケジュールするためです。それらが **整数リソース** と言われるのは、それらが整数で表される量で利用可能であるか、または **公開** されるためです。API サーバーはこれらのリソースの量を整数に制限します。**有効な** 量の例は、**3**、**3000m**、および **3Ki** などです。

不透明な整数リソースは以下を割り当てるために使用できます。

- ラストレベルキャッシュ (LLC)
- Graphics processing unit (GPU) デバイス
- Field-programmable gate array (FPGA) デバイス
- 帯域幅を並列ファイルシステムと共有するためのスロット

たとえば、ノードに特殊な種類のディスクストレージ 800 GiB がある場合、その特殊ストレージに **opaque-int-resource-special-storage** などの名前を作成することができます。これを 100 GiB などの特定サイズのチャンクの単位で公開することができます。この場合、ノードではタイプ **opaque-int-resource-special-storage** の 8 つのリソースがあることを公開します。

不透明な整数リソースの名前はプレフィックス **pod.alpha.kubernetes.io/opaque-int-resource-** で開始する必要があります。

22.2. 不透明な整数リソースの作成

以下は、不透明な整数リソースを使用するために必要な 2 つの手順です。まず、クラスターのオペレーターは 1 つ以上のノードでノード別の不透明なリソースに名前を付け、これを公開する必要があります。次に、アプリケーション開発者は Pod で不透明なリソースを要求する必要があります。

不透明な整数リソースを利用可能にするには、以下を実行します。

1. リソースを割り当て、**pod.alpha.kubernetes.io/opaque-int-resource-** で開始される名前を割り当てます。
2. クラスターのノードについて **status.capacity** で利用可能な数量を指定する PATCH HTTP 要求を API サーバーに送信することにより、新規の不透明な整数リソースを公開します。
たとえば、以下の HTTP 要求は **openshift-node-1** ノードで 5 つの **foo** リソースを公開します。

```
PATCH /api/v1/nodes/openshift-node-1/status HTTP/1.1
Accept: application/json
Content-Type: application/json-patch+json
Host: openshift-master:8080

[
  {
    "op": "add",
    "path": "/status/capacity/pod.alpha.kubernetes.io~1opaque-int-
resource-foo",
    "value": "5"
  }
]
```



注記

path にある `~1` は `/` 文字のエンコーディングです。JSON-Patch の操作パスの値は JSON-Pointer として解釈されます。詳細は、[IETF RFC 6901](#)、[セクション 3](#) を参照してください。

この操作の後に、ノード **status.capacity** には新規リソースが含まれます。**status.allocatable** フィールドは、新規リソースで自動的に、また非同期的に更新されます。



注記

スケジューラーは、Pod が適切であるかどうかを評価する際にノードの **status.allocatable** 値を使用するため、ノードの容量を新規リソースで修正してから、そのリソースを要求する最初の Pod がノードでスケジュールされるまでの間に少しの遅延が生じる可能性があります。

アプリケーション開発者は、不透明なリソースの名前を **spec.containers[].resources.requests** フィールドのキーとして組み込むよう Pod 設定を編集することにより、不透明なリソースを消費できます。

例: 以下の Pod は 2 つの CPU および 1 つの **foo** (不透明なリソース) を要求しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: myimage
    resources:
      requests:
        cpu: 2
        pod.alpha.kubernetes.io/opaque-int-resource-foo: 1
```

Pod は、(CPU、メモリー、およびすべての不透明なリソースを含む) リソース要求のすべてが満たされる場合にのみスケジュールされます。Pod は、リソース要求がいずれのノードでも満たされない場合には **PENDING** 状態のままになります。

```
Conditions:
  Type      Status
  PodScheduled  False
...
Events:
  FirstSeen    LastSeen  Count  From              SubObjectPath Type      Reason
Message
-----
-----
  14s         0s      6  default-scheduler  Warning    FailedScheduling  No nodes
are available that match all of the following predicates:: Insufficient
pod.alpha.kubernetes.io/opaque-int-resource-foo (1).
```

この情報は『Developer Guide』の「[Quotas and Limit Ranges](#)」でも参照できます。

第23章 オーバーコミット

23.1. 概要

コンテナは、[コンピュートリソース要求および制限](#)を指定することができます。要求はコンテナのスケジューリングに使用され、最小限のサービス保証を提供します。一方、制限はノードで消費できるコンピュートリソースの量を制限します。

[scheduler](#) は、クラスター内のすべてのノードにおけるコンピュートリソース使用の最適化を試行します。これは Pod のコンピュートリソース要求とノードの利用可能な容量を考慮に入れて Pod を特定のノードに配置します。

要求および制限により、管理者はノードでのリソースのオーバーコミットを許可し、管理できます。これは、保証されるパフォーマンスとキャパシティのトレードオフが許容される開発環境において役立ちます。

23.2. 要求および制限

各コンピュートリソースについて、コンテナはリソース要求および制限を指定できます。スケジューリングの決定は要求に基づいて行われ、ノードに要求される値を満たす十分な容量があることが確認されます。コンテナが制限を指定するものの、要求を省略する場合、要求はデフォルトで制限値に設定されます。コンテナは、ノードの指定される制限を超えることはできません。

制限の実施方法は、コンピュートリソースのタイプによって異なります。コンテナが要求または制限を指定しない場合、コンテナはリソース保証のない状態でノードにスケジュールされます。実際に、コンテナはローカルのもっとも低い優先順位で利用できる指定リソースを消費できます。リソースが不足する状態では、リソース要求を指定しないコンテナに最低レベルの QoS (Quality of Service) が設定されます。

23.2.1. Buffer Chunk Limit の調整

Fluentd ロガーが多数のログを処理できない場合、メモリーの使用量を減らし、データ損失を防ぐためにファイルバッファリングに切り換える必要があります。

Fluentd `buffer_chunk_limit` は、デフォルト値が `8m` の環境変数 `BUFFER_SIZE_LIMIT` によって決定されます。出力ごとのファイルのバッファサイズは、デフォルト値が `256Mi` の環境変数 `FILE_BUFFER_LIMIT` によって決定されます。永続的なボリュームサイズは、`FILE_BUFFER_LIMIT` に出力を乗算した結果よりも大きくなければなりません。

Fluentd および Mux Pod では、永続ボリューム `/var/lib/fluentd` は PVC または `hostmount` などによって作成される必要があります。その領域はファイルバッファに使用されます。

`buffer_type` および `buffer_path` は、以下のように Fluentd 設定ファイルで設定されます。

```
$ egrep "buffer_type|buffer_path" *.conf
output-es-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-config`
output-es-ops-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-ops-config`
filter-pre-mux-client.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-mux-client`
```

Fluentd `buffer_queue_limit` は 32 です。

23.3. コンピュートリソース

コンピュートリソースについてのノードで実施される動作は、リソースタイプによって異なります。

23.3.1. CPU

コンテナには要求する CPU の量が保証され、さらにコンテナで指定される任意の制限までノードで利用可能な CPU を消費できます。複数のコンテナが追加の CPU の使用を試行する場合、CPU 時間が各コンテナで要求される CPU の量に基づいて分配されます。

たとえば、あるコンテナが 500m の CPU 時間を要求し、別のコンテナが 250m の CPU 時間を要求した場合、ノードで利用可能な追加の CPU 時間は 2:1 の比率でコンテナ間で分配されます。コンテナが制限を指定している場合、指定した制限を超えて CPU を使用しないようにスロットリングされます。

CPU 要求は、Linux カーネルの CFS 共有サポートを使用して実施されます。デフォルトで、CPU 制限は、Linux カーネルの CFS クォータサポートを使用して 100ms の測定間隔で実施されます。ただし、[これは無効にすることができます](#)。

23.3.2. メモリー

コンテナには要求するメモリー量が保証されます。コンテナは要求したよりも多くのメモリーを使用できますが、いったん要求した量を超えた場合には、ノードのメモリーが不足している状態では強制終了される可能性があります。

コンテナが要求した量よりも少ないメモリーを使用する場合、システムタスクやデーモンがノードのリソース予約で確保されている分よりも多くのメモリーを必要としない限りそれが強制終了されることはありません。コンテナがメモリーの制限を指定する場合、その制限量を超えると即時に強制終了されます。

23.4. QOS (QUALITY OF SERVICE) クラス

ノードは、要求を指定しない Pod がスケジューリングされている場合やノードのすべての Pod での制限の合計が利用可能なマシンの容量を超える場合に **オーバーコミット** されます。

オーバーコミットされる環境では、ノード上の Pod がいずれかの時点で利用可能なコンピュートリソースよりも多くの量の使用を試行することができます。これが生じると、ノードはそれぞれの Pod に優先順位を指定する必要があります。この決定を行うために使用される機能は、QoS (Quality of Service) クラスと呼ばれます。

各コンピュートリソースについて、コンテナは 3 つの QoS クラスに分類されます (優先順位は降順)。

表23.1 QoS (Quality of Service) クラス

優先順位	クラス名	説明
1 (最高)	Guaranteed	制限およびオプションの要求がすべてのリソースについて設定されている場合 (0 と等しくない) でそれらの値が等しい場合、コンテナは Guaranteed として分類されます。

優先順位	クラス名	説明
2	Burstable	制限およびオプションの要求がすべてのリソースについて設定されている場合 (0 と等しくない) でそれらの値が等しくない場合、コンテナは Burstable として分類されます。
3 (最低)	BestEffort	要求および制限がリソースのいずれについても設定されない場合、コンテナは BestEffort として分類されます。

メモリーは圧縮できないリソースであるため、メモリー不足の状態では、最も優先順位の低いコンテナが最初に強制終了されます。

- **Guaranteed** コンテナは優先順位が最も高いコンテナとして見なされ、保証されます。強制終了されるのは、これらのコンテナで制限を超えるか、またはシステムがメモリー不足の状態にあるものの、エビクトできる優先順位の低いコンテナが他にない場合のみです。
- システム不足の状態にある **Burstable** コンテナは、制限を超過し、**BestEffort** コンテナが他に存在しない場合に強制終了される可能性があります。
- **BestEffort** コンテナは優先順位の最も低いコンテナとして処理されます。これらのコンテナのプロセスは、システムがメモリー不足になると最初に強制終了されます。

23.5. マスターでのオーバーコミットの設定

スケジューリングは要求されるリソースに基づいて行われる一方で、クォータおよびハード制限はリソース制限のことを指しており、これは要求されるリソースよりも高い値に設定できます。要求と制限の間の差異は、オーバーコミットのレベルを定めるものとなります。たとえば、コンテナに 1Gi のメモリー要求と 2Gi のメモリー制限が指定される場合、コンテナのスケジューリングはノードで 1Gi を利用可能とする要求に基づいて行われますが、2Gi まで使用することができます。そのため、この場合のオーバーコミットは 200% になります。

OpenShift Container Platform 管理者がオーバーコミットのレベルを制御し、ノードのコンテナ密度を管理する必要がある場合、開発者コンテナで設定された要求と制限の比率を上書きするようマスターを設定することができます。この設定を制限とデフォルトを指定する [プロジェクトごとの LimitRange](#) と共に使用することで、オーバーコミットを必要なレベルに設定できるようコンテナの制限と要求を調整することができます。

これを実行するには、以下の例にあるように **master-config.yaml** で **ClusterResourceOverride** 受付コントローラーを設定することが必要です (既存の設定ツリーが存在する場合はこれを再利用するか、または必要に応じて存在しない要素を導入します)。

```

admissionConfig:
  pluginConfig:
    ClusterResourceOverride: ❶
      configuration:
        apiVersion: v1
        kind: ClusterResourceOverrideConfig
        memoryRequestToLimitPercent: 25 ❷
        cpuRequestToLimitPercent: 25 ❸
        limitCPUToMemoryPercent: 200 ❹

```


- 1 これはプラグイン名です。大文字/小文字の区別が必要であり、プラグインの完全に一致する名前以外はすべて無視されます。
- 2 (オプション、1-100) コンテナのメモリー制限が指定されているか、デフォルトに設定されている場合、メモリー要求は制限のこのパーセンテージに対応して上書きされます。
- 3 (オプション、1-100) コンテナの CPU 制限が指定されているか、またはデフォルトに設定されている場合、CPU 要求は制限のこのパーセンテージに対応して上書きされます。
- 4 (オプション、正の整数) コンテナのメモリー制限が指定されているか、デフォルトに設定されている場合、CPU 制限はメモリー制限のパーセンテージに対応して上書きされます。この場合、1Gi の RAM が 1 CPU コアと等しくなる場合に 100 パーセントになります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。

マスター設定の変更後は、マスターの再起動が必要になります。

制限がコンテナに設定されていない場合にはこれらの上書きは影響を与えないことに注意してください。(個別プロジェクトごとに、または [プロジェクトテンプレート](#) を使用して) デフォルトの制限で [LimitRange オブジェクトを作成](#) し、上書きが適用されるようにします。

また、上書き後も、コンテナの制限および要求がプロジェクトのいずれかの LimitRange オブジェクトで依然として検証される必要があることにも注意してください。たとえば、開発者が最小限度に近い制限を指定し、要求を最小限度よりも低い値に上書きすることで、Pod が禁止される可能性があります。この最適でないユーザーエクスペリエンスについては、今後の作業で対応する必要がありますが、現時点ではこの機能および LimitRanges を注意して設定してください。

上書きが設定されている場合に、プロジェクトを編集し、以下のアノテーションを追加することで、上書きをプロジェクトごとに無効にすることができます (たとえば、インフラストラクチャーコンポーネントの設定を上書きと切り離して実行できます)。

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

23.6. ノードでのオーバーコミットの設定

オーバーコミット環境では、最適なシステム動作を提供できるようにノードを適切に設定する必要があります。

23.6.1. Quality of Service (QoS) 層でのメモリー予約

experimental-qos-reserved パラメーターを使用して、特定の QoS レベルの Pod で予約されるメモリーのパーセンテージを指定することができます。この機能は、最も低い QoS クラスの Pod が高い QoS クラスの Pod で要求されるリソースを使用できないようにするために要求されたリソースの予約を試行します。

高い QoS レベル用にリソースを予約することで、リソース制限を持たない Pod が高い QoS レベルの Pod で要求されるリソースを侵害しないようにできます。

experimental-qos-reserved を設定するには、ノードの `/etc/origin/node/node-config.yaml` ファイルを編集します。

```
kubeletArguments:
  cgroups-per-qos:
    - true
  cgroup-driver:
```



```
- 'systemd'
cgroup-root:
- '/'
experimental-qos-reserved: ❶
- 'memory=50%'
```

- ❶ Pod のリソース要求が QoS レベルでどのように予約されるかを指定します。

OpenShift Container Platform は、以下のように **experimental-qos-reserved** パラメーターを使用します。

- **experimental-qos-reserved=memory=100%** の値は、**Burstable** および **BestEffort** QoS クラスが、これらより高い QoS クラスで要求されたメモリーを消費するのを防ぎます。これにより、**Guaranteed** および **Burstable** ワークロードのメモリーリソースの保証レベルを上げることが優先され、**BestEffort** および **Burstable** ワークロードでの OOM が発生するリスクが高まります。
- **experimental-qos-reserved=memory=50%** の値は、**Burstable** および **BestEffort** QoS クラスがこれらより高い QoS クラスによって要求されるメモリーの半分を消費することを許可します。
- **experimental-qos-reserved=memory=0%** の値は、**Burstable** および **BestEffort** QoS クラスがノードの割り当て可能分を完全に消費することを許可しますが (利用可能な場合)、これにより、**Guaranteed** ワークロードが要求したメモリーにアクセスできなくなるリスクが高まります。この状況により、この機能は無効にされています。

23.6.2. CPU 制限の実施

デフォルトで、ノードは Linux カーネルの CPU CFS クォータのサポートを使用して指定された CPU 制限を実施します。ノードで CPU 制限を実施する必要がない場合は、以下を含むように [ノード設定ファイル](#) (**node-config.yaml** ファイル) を変更してその実施を無効にすることができます。

```
kubeletArguments:
  cpu-cfs-quota:
  - "false"
```

CPU 制限の実施が無効にされる場合、それがノードに与える影響を理解しておくことが重要になります。

- コンテナが CPU の要求をする場合、これは Linux カーネルの CFS 共有によって引き続き実施されます。
- コンテナが CPU の要求を明示的に指定しないものの、制限を指定する場合には、要求は指定された制限にデフォルトで設定され、Linux カーネルの CFS 共有で実施されます。
- コンテナが CPU の要求と制限の両方を指定する場合、要求は Linux カーネルの CFS 共有で実施され、制限はノードに影響を与えません。

23.6.3. システムリソースのリソース予約

[スケジューラー](#) は、Pod 要求に基づいてノード上のすべての Pod に十分なリソースがあることを確認します。これは、ノード上のコンテナの要求の合計がノード容量を上回らないことを確認します。これには、ノードで起動されたすべてのコンテナが含まれますが、クラスターの範囲外で起動されたコンテナやプロセスは含まれません。

ノード容量の一部を予約して、クラスターが機能できるようノードで実行する必要のあるシステムデーモン用に確保することが推奨されます (**sshd**、**docker** など)。とくに、メモリーなどの圧縮できないリソースのリソース予約を行うことが推奨されます。

Pod 以外のプロセスのリソースを明示的に予約する必要がある場合、以下の 2 つの方法でこれを実行できます。

- 優先される方法として、スケジューリングに利用できるリソースを指定してノードリソースを割り当てることができます。詳細は、「[ノードリソースの割り当て](#)」を参照してください。
- 2 つ目の方法として **resource-reserver** Pod を作成できます。この Pod は、クラスターによるスケジューリングの対象外となるようノードで容量を確保します。以下は例になります。

例23.1 resource-reserver Pod の定義

```
apiVersion: v1
kind: Pod
metadata:
  name: resource-reserver
spec:
  containers:
  - name: sleep-forever
    image: gcr.io/google_containers/pause:0.8.0
    resources:
      limits:
        cpu: 100m ①
        memory: 150Mi ②
```

- ① クラスターに認識されないホストレベルのデーモン用にノード上で確保する CPU の量です。
- ② クラスターに認識されないホストレベルのデーモン用にノード上で確保するメモリーの量です。

定義は **resource-reserver.yaml** のようなファイルに保存し、ファイルを **/etc/origin/node/** または別の指定がある場合は **--config=<dir>** などのノード設定ディレクトリーに置くことができます。

さらに、ノードサーバーをノード設定ディレクトリーから定義を読み取れるように設定する必要があります。これは、[ノード設定ファイル](#) の **kubeletArguments.config** フィールドでディレクトリーの名前(通常の名前は **node-config.yaml**) を指定することにより実行できます。

```
kubeletArguments:
  config:
  - "/etc/origin/node" ①
```

- ① **--config=<dir>** が指定されている場合、ここでは **<dir>** を使用します。

resource-reserver.yaml ファイルが有効な状態でノードサーバーを起動すると、**sleep-forever** コンテナも起動します。スケジューラーはノードの残りの容量も考慮し、クラスター Pod を配置する場所を適宜調整します。

resource-reserver Pod を削除するには、ノード設定ディレクトリーから **resource-reserver.yaml** ファイルを削除するか、またはこれを移動することができます。

23.6.4. カーネルの調整可能なフラグ

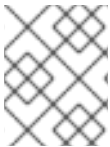
ノードが起動すると、メモリー管理用のカーネルの調整可能なフラグが適切に設定されます。カーネルは、物理メモリーが不足しない限り、メモリーの割り当てに失敗することはありません。

この動作を確認するために、ノードはカーネルに対し、常にメモリーのオーバーコミットを実行するように指示します。

```
$ sysctl -w vm.overcommit_memory=1
```

また、ノードはカーネルに対し、メモリーが不足する状況でもパニックにならないように指示します。その代わりに、カーネルの OOM killer は優先順位に基づいてプロセスを強制終了します。

```
$ sysctl -w vm.panic_on_oom=0
```



注記

上記のフラグはノード上にすでに設定されているはずであるため、追加のアクションは不要です。

23.6.5. swap メモリーの無効化

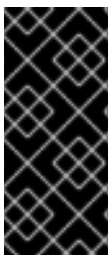
QoS 保証を維持するため、swap はノード上でデフォルトで無効にすることができます。そうしない場合、ノードの物理リソースがオーバーサブスクライブし、Pod の配置時の Kubernetes スケジューラーによるリソース保証が影響を受ける可能性があります。

たとえば、2 つの Guaranteed pod がメモリー制限に達した場合、それぞれのコンテナが swap メモリーを使用し始める可能性があります。十分な swap 領域がない場合には、pod のプロセスはシステムのオーバーサブスクライブのために終了する可能性があります。

swap を無効にするには、以下を実行します。

```
$ swapoff -a
```

swap を無効にしないと、ノードが **MemoryPressure** にあることを認識しなくなり、Pod がスケジューリング要求に対応するメモリーを受け取れなくなります。結果として、追加の Pod がノードに配置され、メモリー不足の状態が加速し、最終的にはシステムの Out Of Memory (OOM) イベントが発生するリスクが高まります。



重要

swap が有効にされている場合、利用可能なメモリーについての [リソース不足の処理 \(out of resource handling\)](#) のエビクションしきい値は予想通りに機能しなくなります。メモリー不足の状態の場合に Pod をノードからエビクトし、Pod を不足状態にない別のノードで再スケジューリングできるようにリソース不足の処理 (out of resource handling) を利用できるようにします。

第24章 INGRESS トラフィックの固有の外部 IP の割り当て

24.1. 概要

外部トラフィックをクラスターにつなぐ方法の 1 つとして、ExternalIP または IngressIP アドレスを使用することができます。



重要

この機能は、クラウド以外のデプロイメントでのみサポートされます。クラウド (GCE、AWS、および OpenStack) デプロイメントの場合、ロードバランサーサービスを使用し、クラウドの自動デプロイメントでサービスのエンドポイントをターゲットに設定します。

OpenShift Container Platform は 2 つの IP アドレスのプールをサポートします。

- IngressIP は、サービスの外部 IP アドレスを選択する場合に Loadbalancer で使用されます。
- ExternalIP は、ユーザーが設定されたプールから特定 IP を選択する場合に使用されます。



注記

これらはいずれも、ネットワークインターフェースコントローラー (NIC) または仮想イーサネット、または外部ルーティングのいずれであっても、使用される OpenShift Container Platform ホストのデバイスに設定される必要があります。この場合、Ipfailover はホストを設定し、NIC を設定するため、これを使用することが推奨されます。

IngressIP および ExternalIP はいずれも外部トラフィックのクラスターへのアクセスを可能にし、適切にルーティングされている場合に、外部トラフィックはサービスが公開する TCP/UDP ポート経由でサービスのエンドポイントに到達できます。これは、外部 IP をサービスに手動で割り当てる際に、制限された数の共有 IP アドレスのポート領域を管理しなくてはならない場合よりも単純になります。またこれらのアドレスは、[高可用性](#)を設定する場合に仮想 IP (VIP) としても使用できます。

OpenShift Container Platform は IP アドレスの自動および手動割り当ての両方をサポートしており、それぞれのアドレスは 1 つのサービスの最大数に割り当てられることが保証されます。これにより、各サービスは、ポートが他のサービスで公開されているかによらず、自らの選択したポートを公開できます。

24.2. 制限

ExternalIP を使用するには、以下を実行できます。

- [externalIPNetworkCIDRs](#) 範囲から IP アドレスを選択します。
- マスター設定ファイルで、IP アドレスを [ingressIPNetworkCIDR](#) プールから割り当てます。この場合、OpenShift Container Platform はローカルバランサーサービスタイプのクラウド以外のバージョンを実装し、IP アドレスをサービスに割り当てます。

注意

割り当てた IP アドレスがクラスター内の 1 つ以上のノードで終了することを確認する必要があります。既存の `oc adm ipfailover` を使用して外部 IP の可用性が高いことを確認します。

手動で設定された外部 IP の場合、起こり得るポートのクラッシュについては「first-come, first-served (先着順)」で処理されます。ポートを要求する場合、その IP アドレスに割り当てられていない場合にのみ利用可能となります。以下は例になります。

手動で設定された外部 IP のポートのクラッシュ例

2 つのサービスが同じ外部 IP アドレス 172.7.7.7 で手動で設定されている。

MongoDB service A がポート 27017 を要求し、次に **MongoDB service B** が同じポートを要求する。最初の要求がこのポートを取得します。

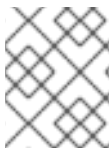
ただし、Ingress コントローラーが外部 IP を割り当てる場合、ポートのクラッシュは問題とはなりません。コントローラーが各サービスに固有のアドレスを割り当てるためです。

24.3. 固有の外部 IP を使用するようにクラスターを設定する

クラウド以外のクラスターで、`ingressIPNetworkCIDR` はデフォルトで `172.29.0.0/16` に設定されています。クラスター環境がこのプライベート範囲をまだ使用していない場合は、このデフォルトを使用できます。ただし、異なる範囲を使用する必要がある場合は、ingress IP を割り当てる前に、`/etc/origin/master/master-config.yaml` ファイルで `ingressIPNetworkCIDR` を設定する必要があります。次に、マスターサービスを再起動します。

注意

LoadBalancer タイプのサービスに割り当てられる外部 IP は常に `ingressIPNetworkCIDR` の範囲にあります。`ingressIPNetworkCIDR` が割り当てられた外部 IP がこの範囲内からなくなるように変更される場合、影響を受けるサービスには、新規の範囲と互換性のある新規の外部 IP が割り当てられます。



注記

高可用性を使用している場合、この範囲は 255 IP アドレスより少なくなければなりません。

`/etc/origin/master/master-config.yaml` のサンプル

```
networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16
```

24.3.1. サービスの Ingress IP の設定

Ingress IP を割り当てるには、以下を実行します。

1. **loadBalancerIP** 設定で特定の IP を要求する LoadBalancer サービスの YAML ファイルを作成します。

LoadBalancer 設定サンプル

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
```

```
ports:
- name: db
  port: 3306
loadBalancerIP: 172.29.0.1
type: LoadBalancer
selector:
  name: my-db-selector
```

- Pod に LoadBalancer サービスを作成します。

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

- 外部 IP のサービスを確認します。たとえば、**myservice** という名前のサービスを確認します。

```
$ oc get svc myservice
```

LoadBalancer タイプのサービスに外部 IP が割り当てられている場合、出力には IP が表示されます。

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
myservice	172.30.74.106	172.29.0.1	3306/TCP	30s

24.4. 開発またはテスト目的での INGRESS CIDR のルーティング

ingress CIDR のトラフィックをクラスターのノードに送信する静的ルートを追加します。以下は例になります。

```
# route add -net 172.29.0.0/16 gw 10.66.140.17 eth0
```

上記の例では、**172.29.0.0/16** は **ingressIPNetworkCIDR**、**10.66.140.17** はノード IP です。

24.4.1. サービス externalIP

クラスターの内部 IP アドレスに加えて、アプリケーション開発者はクラスターの外部にある IP アドレスを設定することができます。OpenShift Container Platform 管理者は、トラフィックがこの IP を持つノードに到達することを確認する必要があります。

externalIP は、**master-config.yaml** ファイルで設定される **externalIPNetworkCIDRs** 範囲から管理者によって選択される必要があります。**master-config.yaml** が変更される際に、マスターサービスは再起動される必要があります。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-master-controllers
```

externalIPNetworkCIDR /etc/origin/master/master-config.yaml のサンプル

```
networkConfig:
  externalIPNetworkCIDR: 172.47.0.0/24
```

サービス externalIP 定義 (JSON)

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "my-service"
  },
  "spec": {
    "selector": {
      "app": "MyApp"
    },
    "ports": [
      {
        "name": "http",
        "protocol": "TCP",
        "port": 80,
        "targetPort": 9376
      }
    ],
    "externalIPs" : [
      "80.11.12.10"
    ]
  }
}
```

- ① ポート が公開される外部 IP アドレスの一覧です (これは内部 IP アドレス一覧に追加される一覧です)。

第25章 OUT OF RESOURCE (リソース不足) エラーの処理

25.1. 概要

このトピックでは、OpenShift Container Platform がメモリー不足 (OOM) やディスク領域不足の状況を防ぐためのベストエフォートの取り組みについて説明します。

ノードは、利用可能なコンピュータリソースが少ない場合に安定性を維持する必要があります。これは、メモリーやディスクなどの圧縮不可能なリソースを扱う場合にとくに重要になります。どちらかのリソースが消費されると、ノードは不安定になります。

管理者は、[エビクションポリシー](#)を使用してノードをプロアクティブにモニターし、ノードでコンピュータリソースおよびメモリーリソースが不足する状況を防ぐことができます。

このトピックでは、OpenShift Container Platform がリソース不足の状況に対処する方法についての情報を提供し、[シナリオ例](#)や[推奨される対策](#)について説明します。

- [リソースの回収](#)
- [Pod のエビクション](#)
- [Pod のスケジューリング](#)
- [リソース不足および Out of Memory Killer](#)



警告

swap メモリーがノードに対して有効にされる場合、ノードは **MemoryPressure** 状態にあるかどうかを検知できません。

メモリーベースのエビクションを利用するには、オペレーターは [swap](#) を無効にする必要があります。

25.2. エビクションポリシーの設定

エビクションポリシーにより、ノードが利用可能なリソースが少ない状況で実行されている場合に 1 つ以上の Pod が失敗することを許可します。Pod の失敗により、ノードは必要なリソースを回収できます。

エビクションポリシーは、[エビクショントリガーシグナル](#)と、[ノード設定ファイル](#)または[コマンドライン](#)で設定される特定の[エビクションしきい値](#)の組み合わせです。エビクションは、ノードがしきい値を超える Pod に対して即時のアクションを実行するハードエビクションか、またはノードがアクションを実行する前の猶予期間を許可するソフトエビクションのどちらかになります。[ハードおよびソフトエビクション](#)の違いという重要な情報については、以下のセクションを参照してください。

適切に設定されたエビクションポリシーを使用することで、ノードは、プロアクティブにモニターし、コンピュータリソースを完全に使い切る事態を防ぐことができます。



注記

ノードによる Pod の失敗が生じる場合、Pod 内のすべてのコンテナが停止し、**PodPhase** は **Failed** に切り替わります。

25.2.1. ノード設定を使用したポリシーの作成

エビクションポリシーを設定するには、ノード設定ファイル (`/etc/origin/node/node-config.yaml` ファイル) を編集して **eviction-hard** または **eviction-soft** パラメーターでエビクションしきい値を指定します。

以下に例を示します。

例25.1 ハードエビクションについてのノード設定ファイルのサンプル

```
kubeletArguments:
  eviction-hard: ❶
    - memory.available<500Mi ❷
    - nodefs.available<500Mi
    - nodefs.inodesFree<100Mi
    - imagefs.available<100Mi
    - imagefs.inodesFree<100Mi
```

❶ エビクションのタイプ: **ハードエビクション**にこのパラメーターを使用します。

❷ 特定のエビクショントリガーシグナルに基づくエビクションのしきい値です。

例25.2 ソフトエビクションについてのノード設定ファイルのサンプル

```
kubeletArguments:
  eviction-soft: ❶
    - memory.available<500Mi ❷
    - nodefs.available<500Mi
    - nodefs.inodesFree<100Mi
    - imagefs.available<100Mi
    - imagefs.inodesFree<100Mi
  eviction-soft-grace-period: ❸
    - memory.available=1m30s
    - nodefs.available=1m30s
    - nodefs.inodesFree=1m30s
    - imagefs.available=1m30s
    - imagefs.inodesFree=1m30s
```

❶ エビクションのタイプ: **ソフトエビクション**にこのパラメーターを使用します。

❷ 特定のエビクショントリガーシグナルに基づくエビクションのしきい値です。

❸ ソフトエビクションの猶予期間です。パフォーマンスを最適化するためにデフォルト値のままにします。

1. 変更を有効するために OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

25.2.2. エビクシヨシグナルについて

以下の表にあるシグナルのいずれかに基づいてエビクシヨシの意思決定をトリガーするようノードを設定することができます。エビクシヨシグナルは、しきい値と共に[エビクシヨシのしきい値](#)に追加できます。

各シグナルの値は、ノード要約 API に基づいて **Description** 列で説明されています。

シグナルを表示するには、以下を実行します。

```
curl <certificate details> \
  https://<master>/api/v1/nodes/<node>/proxy/stats/summary
```

表25.1 サポートされるエビクシヨシグナル

ノードの 状態	エビクシヨ シグナル	値	説明
MemoryP ressure	memory. availab le	memory. availab le = node.st atus.ca pacity[memory] - node.st ats.mem ory.wor kingSet	ノードの利用可能なメモリーがエビクシヨシしきい値を超えている。
DiskPre ssure	nodefs. availab le	nodefs. availab le = node.st ats.fs. availab le	ノードの root ファイルシステムまたはイメージファイルシステムのいずれかで利用可能なディスク領域がエビクシヨシしきい値を超えている。
	nodefs. inodesF ree	nodefs. inodesF ree = node.st ats.fs. inodesF ree	

ノードの 状態	エビクシヨ ンシグナル	値	説明
	imagefs .available	imagefs .available = node.st ats.run time.im agefs.a vailabl e	
	imagefs .inodes Free	imagefs .inodes Free = node.st ats.run time.im agefs.i nodesFr ee	

上記のシグナルのそれぞれは、リテラル値またはパーセンテージベースの値のいずれかをサポートします。パーセンテージベースの値は、各シグナルに関連付けられる合計容量との関連で計算されます。

スクリプトは kubelet が実行する一連の手順を使用し、cgroup から **memory.available** 値を派生させます。スクリプトは計算から非アクティブなファイルメモリー (つまり、非アクティブな LRU リストのファイルベースのメモリーのバイト数) を計算から除外します。非アクティブなファイルメモリーはリソースの不足時に回収可能になることが想定されます。



注記

free -m はコンテナで機能しないため、**free -m** のようなツールは使用しないでください。

ノードは、以下のようにディスクの不足状態を検出する際に **nodefs** および **imagefs** ファイルシステムのパーティションをサポートします。

- **nodefs** ファイルシステムは、ノードがローカルディスク、デーモンログなど (/ を指定するファイルシステムなど) に使用するファイルシステムです。
- **imagefs** ファイルシステムは、コンテナランタイムがイメージおよび個別のコンテナの書き込み可能な層を保存するために使用するファイルシステムです。

OpenShift Container Platform はこれらのファイルシステムを 10 秒ごとにモニターします。

ボリュームおよびログを専用ファイルシステムに保存する場合、ノードはそのファイルシステムをモニターしません。



注記

OpenShift Container Platform 3.4, の時点で、ノードはディスク不足に基づくエビクションの意思決定をトリガーする機能をサポートします。ディスク不足のために Pod をエビクトする前に、ノードは[コンテナおよびイメージのガベージコレクション](#)も実行します。今後のリリースでは、ガベージコレクションは、純粋なディスクのエビクションベースの設定が優先されるために非推奨になります。

25.2.3. エビクションのしきい値について

しきい値を[ノード設定ファイル](#)に追加することで、ノードのリソース回収をトリガーするエビクションしきい値を指定するようノードを設定することができます。

関連付けられた猶予期間とは別にエビクションのしきい値に達する場合、ノードは、ノードがメモリー不足またはディスク不足であることを示す状態を報告します。これにより、スケジューラーがリソース回収の試行期間中に Pod をノードにスケジュールすることを防ぐことができます。

ノードは、**node-status-update-frequency** 引数で指定される頻度で、ノードのステータス更新を継続的に報告します。これはデフォルトで **10s** (10 秒) に設定されています。

エビクションのしきい値は、しきい値に達する際にノードが即時にアクションを実行する場合に[ハード](#)となり、リソース回収前の猶予期間を許可する場合は[ソフト](#)になります。



注記

ソフトエビクションは、特定レベルの使用状況をターゲットとしているものの、一時的な値の上昇を許容できる場合により一般的に使用されます。ソフトエビクションは、ハードエビクションのしきい値よりも低く設定することが推奨されますが、期間はオペレーターが固有に設定できます。システム予約もソフトエビクションのしきい値以上に設定する必要があります。

ソフトエビクションのしきい値は高度な機能になります。ソフトエビクションのしきい値の使用を試行する前にハードエビクションのしきい値を設定してください。

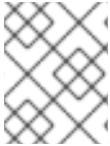
しきい値は以下の形式で設定されます。

```
<eviction_signal><operator><quantity>
```

- **eviction-signal** 値は任意の[サポートされるエビクションシグナル](#)にすることができます。
- **operator** 値は < になります。
- **quantity** 値は、Kubernetes で使用される[数量表現](#)と一致している必要があり、% トークンで終了する場合はパーセンテージで表現されます。

たとえば、オペレーターが 10Gi メモリーのあるノードを持つ場合で、オペレーターは利用可能なメモリーが 1Gi を下回る場合にエビクションを導入する必要がある場合、メモリーのエビクションしきい値は以下のいずれかで指定することができます。

```
memory.available<1Gi
memory.available<10%
```



注記

ノードはエビクションしきい値の評価とモニターを 10 秒ごとに実行し、値を変更することはできません。これはハウスキープ処理の間隔になります。

25.2.3.1. ハードエビクションのしきい値について

ハードエビクションのしきい値には猶予期間がなく、しきい値を超えることが確認される場合には、ノードは関連付けられた不足状態にあるリソースを回収するために即時のアクションを実行します。ハードエビクションのしきい値に達する場合、ノードは正常な終了なしに Pod を即時に強制終了します。

ハードエビクションのしきい値を設定するには、「[ポリシー作成のためのノード設定の使用](#)」に示されるように、エビクションしきい値を **eviction-hard** の下にある [ノード設定ファイル](#) に追加します。

ハードエビクションのしきい値が設定されたノード設定ファイルのサンプル

```
kubeletArguments:
  eviction-hard:
    - memory.available<500Mi
    - nodefs.available<500Mi
    - nodefs.inodesFree<100Mi
    - imagefs.available<100Mi
    - imagefs.inodesFree<100Mi
```

この例は一般的なガイドラインを示すためのもので、推奨される設定ではありません。

25.2.3.1.1. デフォルトのハードエビクションしきい値

OpenShift Container Platform は、**eviction-hard** の以下のデフォルト設定を使用します。

```
...
kubeletArguments:
  eviction-hard:
    - memory.available<100Mi
    - nodefs.available<10%
    - nodefs.inodesFree<5%
    - imagefs.available<15%
...
```

25.2.3.2. ソフトエビクションのしきい値について

ソフトエビクションのしきい値は、エビクションしきい値と管理者が指定する必要な猶予期間のペアを設定します。ノードは、猶予期間が経過するまではエビクションシグナルに関連付けられたリソースを回収しません。猶予期間がノード設定ファイルに指定されない場合、ノードは起動時にエラーを出して失敗します。

さらにソフトエビクションのしきい値に達する場合、オペレーターは Pod をノードからエビクトする際に使用する Pod 終了の猶予期間として許可される最長期間を指定できます。**eviction-max-pod-grace-period** が指定されると、ノードは **pod.Spec.TerminationGracePeriodSeconds** と **maximum-allowed-grace-period** の値の小さい方を使用します。これが指定されていない場合は、ノードは正常な停止なしに、Pod を即時に強制終了します。

ソフトエビクションのしきい値については、以下のフラグがサポートされています。

- **eviction-soft**: エビクションしきい値のセットです (例: **memory.available<1.5Gi**)。対応する猶予期間を経過してからこの値に達すると、Pod のエビクションをトリガーします。
- **eviction-soft-grace-period**: エビクションの猶予期間のセットです (例: **memory.available=1m30s**)。これは、Pod のエビクションをトリガーするまでソフトエビクションのしきい値が有効である期間に対応します。
- **eviction-max-pod-grace-period**: ソフトエビクションのしきい値に達する際の Pod の終了時に使用される最長で許可される猶予期間 (秒単位) です。

ソフトエビクションのしきい値を設定するには、「[ポリシー作成のためのノード設定の使用](#)」に示されるように、エビクションのしきい値を **eviction-soft** の下にある [ノード設定ファイル](#) に追加します。

ソフトエビクションのしきい値が設定されたノード設定ファイルのサンプル

```
kubeletArguments:
  eviction-soft:
    - memory.available<500Mi
    - nodefs.available<500Mi
    - nodefs.inodesFree<100Mi
    - imagefs.available<100Mi
    - imagefs.inodesFree<100Mi
  eviction-soft-grace-period:
    - memory.available=1m30s
    - nodefs.available=1m30s
    - nodefs.inodesFree=1m30s
    - imagefs.available=1m30s
    - imagefs.inodesFree=1m30s
```

この例は一般的なガイドラインを示すためのもので、推奨される設定ではありません。

25.3. スケジューリング用のリソース量の設定

スケジューラーがノードを完全に割り当て、エビクションを防止できるようにするために、スケジューリングで利用できるノードリソースの数量を制御できます。

system-reserved を、Pod のデプロイおよび system-daemon 用にスケジューラーで利用可能にするリソース量と等しくなるようにします。エビクションは、Pod が割り当て可能なリソースの要求量よりも多くのリソースを使用する場合に生じます。

ノードは 2 つの値を報告します。

- **Capacity**: マシンにあるリソースの量です。
- **Allocatable**: スケジューリング用に利用可能にされるリソースの量です。

割り当て可能なリソースの量を設定するには、ノード設定ファイル (**/etc/origin/node/node-config.yaml** ファイル) を編集し、**eviction-hard** または **eviction-soft** の **system-reserved** パラメーターを追加するか、または変更します。

+

```
kubeletArguments:
  eviction-hard: 1
    - "memory.available<500Mi"
```

```
system-reserved:
  - "1.5Gi"
```

- 1 このしきい値は、**eviction-hard** または **eviction-soft** のいずれかにできます。

1. 変更を有効するために OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

25.4. ノードの状態変動の制御

ノードがソフトエビクトしきい値の上下で変動しているものの、関連付けられた猶予期間を超えていない場合、対応するノード状態は **true** と **false** の間で変動します。これにより、スケジュールの問題が生じる場合があります。

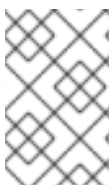
この変動を防ぐには、**eviction-pressure-transition-period** パラメーターを設定して、ノードが不足状態からの切り換え前に待機する期間を制御します。

1. **<resource_type>=<resource_quantity>** のペアを使用してこのパラメーターを編集するか、またはこれをノード設定ファイル (**/etc/origin/node/node-config.yaml**) の **kubeletArguments** セクションに追加します。

```
kubeletArguments:
  eviction-pressure-transition-period="5m"
```

+ ノードは、指定期間の指定された不足状態についてエビクションしきい値に達していないことを確認する場合に状態を **false** に戻します。

+



注記

調整を行う前にデフォルト値 (5 分) を使用します。このデフォルト値のオプションには、システムを安定させ、安定した状態になる前にスケジューラーが新規 Pod をノードに割り当てないようにすることが意図されています。

1. 変更を有効するために OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

25.5. ノードレベルのリソースの回収

エビクション条件が満たされる場合、ノードはシグナルが定義されたしきい値を下回るまで、不足状態にあるリソースを回収するプロセスを実行します。このプロセスでは、ノードはいずれの新規 Pod のスケジューリングもサポートしません。

ノードは、ホストシステムにコンテナランタイム用の専用 **imagefs** が設定されているかどうかに基づいて、エンドユーザー Pod のエビクト前にノードレベルのリソース回収を試行します。

Imagefs が設定されている場合

ホストシステムに **imagefs** が設定されている場合:

- **nodefs** ファイルシステムがエビクションしきい値を満たす場合、ノードは以下の順番でディスク領域を解放します。
 - 実行されない Pod/コンテナの削除
- **imagefs** ファイルシステムがエビクションのしきい値を満たす場合、ノードは以下の順番でディスク領域を解放します。
 - すべての未使用イメージの削除

Imagefs が設定されていない場合

ホストシステムに **imagefs** が設定されていない場合:

- **nodefs** ファイルシステムがエビクションしきい値を満たす場合、ノードは以下の順番でディスク領域を解放します。
 - 実行されない Pod/コンテナの削除
 - すべての未使用イメージの削除

25.6. POD エビクションについて

エビクションしきい値に達し、猶予期間を経過している場合、ノードはシグナルが定義されたしきい値を下回るまで Pod のエビクトのプロセスを実行します。

ノードは、エビクトする Pod を **QoS (Quality of Service)** に基づいてランク付けし、同じ QoS レベルにある Pod については Pod のスケジューリング要求との関連で不足状態のリソースの消費量に基づいてランク付けします。

それぞれの QoS レベルには OOM スコープが設定されており、Linux out-of-memory ツール (OOM killer) はこのスコープを使用して強制終了する Pod を判別します。以下の「[QoS および Out of Memory Killer について](#)」を参照してください。

以下の表には、各 QoS レベルと関連付けられた OOM スコアが一覧表示されています。

表25.2 Quality of Service (QoS) レベル

QoS (Quality of Service)	説明
Guaranteed	要求との関連で不足状態にあるリソースの最大量を消費する Pod が最初に失敗します。いずれの Pod もその要求を超えていない場合、ストラテジーは不足状態にあるリソースを最も多く消費する Pod をターゲットにします。
Burstable	リソースの要求との関連で、不足状態にあるリソースの最大量を消費する Pod が最初に失敗します。いずれの Pod もその要求を超えていない場合、ストラテジーは不足状態にあるリソースを最も多く消費する Pod をターゲットにします。
BestEffort	不足状態にあるリソースの最大量を消費する Pod が最初に失敗します。

Guaranteed Pod は、(ノード、**docker**、**journald** などの) システムデーモンが **システム予約**か、または **kube 予約** の割り当てを使用して予約されている量よりも多くのリソースを消費しているか、またはノードに **Guaranteed Pod** のみが残っているのではない限り、別の Pod のリソース消費が原因でエビクトされることはありません。

ノードに **Guaranteed** Pod のみが残っている場合、ノードはノードの安定性に最も影響の少ない **Guaranteed** Pod をエビクトし、他の **Guaranteed** Pod に対する予想外の消費による影響を制限します。

ローカルディスクは **BestEffort** リソースです。必要な場合は、ノードは **DiskPressure** の発生時にディスクを回収するため、Pod を 1 度に 1 つずつエビクトします。ノードは QoS に基づいて Pod をランク付けします。ノードが inode の不足状態に対応している場合、ノードは QoS の最も低いレベルにある Pod を最初にエビクトして inode を回収します。ノードが利用可能なディスクの不足状態に対応している場合は、ノードはローカルディスクを最も多く消費する QoS 内の Pod をランク付けし、それらの Pod を最初にエビクトします。

25.6.1. QoS および Out of Memory Killer について

ノードによるメモリーの回収が可能になる前にシステムの OOM (Out of Memory) イベントが発生する場合、ノードの動作はこれに対応する OOM killer によって異なります。

ノードは、Pod の QoS に基づいて各コンテナの **oom_score_adj** 値を設定します。

表25.3 Quality of Service (QoS) レベル

QoS (Quality of Service)	oom_score_adj 値
Guaranteed	-998
Burstable	$\min(\max(2, 1000 - (1000 * \text{memoryRequestBytes}) / \text{machineMemoryCapacityBytes}), 999)$
BestEffort	1000

ノードがシステムの OOM イベントが発生する前にメモリーを回収できない場合、**oom_killer** は **oom_score** を計算します。

```
% of node memory a container is using + `oom_score_adj` = `oom_score`
```

次にノードは、スコアの最も高いコンテナを強制終了します。

スケジューリング要求に関連してメモリーを最も多く消費する、QoS の最も低いレベルにあるコンテナが最初に失敗します。

Pod のエビクションとは異なり、Pod コンテナが OOM が原因で失敗する場合、ノードはノードの再起動ポリシーに基づいてこのコンテナを再起動できます。

25.7. POD スケジューラーおよび OOR 状態について

スケジューラーは、追加の Pod をノードに配置する際にノードの状態を表示します。たとえば、ノードに以下のようなエビクションのしきい値がある場合は、以下のようになります。

```
eviction-hard is "memory.available<500Mi"
```

さらに利用可能なメモリーが 500Mi を下回る場合、ノードは **Node.Status.Conditions** の **MemoryPressure** の値を true として報告します。

表25.4 ノードの状態およびスケジューラーの動作

ノードの状態	スケジューラーの動作
MemoryPressure	ノードがこの状態を報告する場合、スケジューラーは BestEffort Pod をノードに配置しません。
DiskPressure	ノードがこの状態を報告する場合、スケジューラーは追加の Pod をノードに配置しません。

25.8. シナリオ例

以下のシナリオについて考えてみましょう。

オペレーター:

- メモリー容量が **10Gi** のノードがある。
- システムデーモン (カーネル、ノードなど) のメモリー容量の 10% を予約する必要がある。
- システムの OOM の悪化または発生の可能性を軽減するため、メモリー使用率が 95% の時点で Pod をエビクトする必要がある。

この設定から、**system-reserved** にはエビクションのしきい値でカバーされるメモリー量が含まれていることを読み取ることができます。

この容量に達するのは、一部の Pod による使用がその要求を超えるか、またはシステムによる使用が **1Gi** を超える場合のいずれかになります。

ノードに 10 Gi の容量があり、システムデーモン (**system-reserved**) 用に 10% の容量を予約する必要がある場合、以下の計算を実行します。

```
capacity = 10 Gi
system-reserved = 10 Gi * .1 = 1 Gi
```

割り当て可能なリソースの量は以下ようになります。

```
allocatable = capacity - system-reserved = 9 Gi
```

これは、デフォルトでスケジューラーはノードに対し、9 Gi のメモリーを要求する Pod をスケジューリングすることを意味します。

エビクションをオンに設定して、エビクションのトリガーが利用可能なメモリーが 30 秒間で 10% の容量を下回ることをノードで確認される場合や、5% の容量を下回る場合には即時にトリガーされるようにする必要があります場合、スケジューラーで 8Gi が割り当て可能であることを確認できる必要があります。そのため、システム予約ではエビクションしきい値の大きい方の値がカバーされている必要があります。

```
capacity = 10 Gi
eviction-threshold = 10 Gi * .1 = 1 Gi
system-reserved = (10Gi * .1) + eviction-threshold = 2 Gi
allocatable = capacity - system-reserved = 8 Gi
```

node-config.yaml に以下を入力します。

```
kubeletArguments:
  system-reserved:
    - "2Gi"
  eviction-hard:
    - memory.available<.5Gi
  eviction-soft:
    - memory.available<1Gi
  eviction-soft-grace-period:
    - memory.available=30s
```

この設定により、スケジューラーは Pod が設定される要求を下回る量のメモリーを使用することを前提とし、メモリー不足を即時に発生させ、エビクションをトリガーする Pod をノードに配置しないようにします。

25.9. 推奨される対策

25.9.1. DaemonSet および Out of Resource (リソース不足) の処理

ノードが DaemonSet によって作成された Pod をエビクトすると、Pod は即時に再作成され、同じノードに再スケジュールされます。これが生じるのは、ノードが DaemonSet から作成された Pod とそれ以外のオブジェクトを区別することができないためです。

通常 DaemonSet は作成する Pod がエビクションの候補として特定されることを防ぐため、**BestEffort** Pod を作成しません。DaemonSet が起動する適切な Pod は **Guaranteed** Pod になります。

第26章 ルーターのモニタリングおよびデバッグ

26.1. 概要

基礎となる実装によっては、実行中の**ルーター**を複数の方法でモニターすることができます。このトピックでは、HAProxy テンプレートルーターおよびその正常性を確認するためのコンポーネントについて説明します。

26.2. 統計の表示

HAProxy ルーターは、HAProxy 統計の web リスナーを公開します。ルーターのパブリック IP アドレスと適切に設定されたポート (デフォルトは **1936**) を入力して統計ページを表示し、プロンプトが出されたら管理者パスワードを入力します。このパスワードおよびポートはルーターのインストール時に設定されますが、それらはコンテナの **haproxy.config** ファイルを表示して確認することができます。

26.3. 統計ビューの無効化

デフォルトで、HAProxy 表示はポート **1936** で公開されます (パスワードで保護されたアカウントを使用する)。HAProxy 統計の公開を無効にするには、統計ポート番号として **0** を指定します。

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

注: HAProxy は依然として統計を収集し、保存しますが、web リスナー経由での統計の**公開**が行われなくなります。要求を HAProxy ルーターコンテナ内の HAProxy AF_UNIX ソケットに送信すれば、依然として統計にアクセスできます。

```
$ cmd="echo 'show stat' | socat - UNIX-CONNECT:/var/lib/haproxy/run/haproxy.sock"
$ routerPod=$(oc get pods --selector="router=router" \
  --template="{{with index .items 0}}{{.metadata.name}}{{end}}")
$ oc exec $routerPod -- bash -c "$cmd"
```



重要

セキュリティ保護の理由により **oc exec** コマンドは、特権付きコンテナにアクセスする場合には機能しません。その代わりに、ノードホストに対して SSH を実行して必要なコンテナで **docker exec** コマンドを使用することができます。

26.4. ログの表示

ルーターのログを表示するには、Pod で **oc logs** コマンドを実行します。ルーターは基礎となる実装を管理するプラグインプロセスとして実行されているため、このログは実際の HAProxy ログではなく、プラグインのログになります。

HAProxy で生成されるログを表示するには、以下の環境変数を使用して syslog サーバーを起動し、その位置情報をルーター Pod に渡します。

表26.1 ルーター Syslog 変数

環境変数	説明
ROUTER_SYSLOG_ADDRESS	syslog サーバーの IP アドレスです。ポートが指定されていない場合、ポート 514 がデフォルトになります。
ROUTER_LOG_LEVEL	これはオプションであり、HAProxy ログレベルを変更する際に設定します。設定されていない場合は、デフォルトのログレベルは warning になります。これは HAProxy がサポートするログレベルに変更することができます。
ROUTER_SYSLOG_FORMAT	これはオプションであり、カスタマイズされた HAProxy ログ形式を定義する際に設定されます。これを HAProxy が受け入れるログ形式の文字列に変更できます。

メッセージを syslog サーバーに送信できるように実行中のルーター Pod を設定するには、以下を実行します。

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=<dest_ip:dest_port>
ROUTER_LOG_LEVEL=<level>
```

たとえば、以下はデフォルトポート **514** で 127.0.0.1 にログを送信するよう HAProxy を設定し、ログレベルを **debug** に変更します。

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=127.0.0.1
ROUTER_LOG_LEVEL=debug
```

26.5. ルーター内部の表示

routes.json

ルートは HAProxy ルーターで処理され、メモリー、ディスクおよび HAProxy 設定ファイルに保存されます。HAProxy 設定ファイルを生成するためにテンプレートに渡される内部ルート表示は **/var/lib/haproxy/router/routes.json** ファイルで確認できます。ルーティングの問題のトラブルシューティング時には、このファイルを表示して設定を有効にするために使用されているデータを確認できます。

HAProxy 設定

HAProxy 設定および特定ルート用に作成されたバックエンドは **/var/lib/haproxy/conf/haproxy.config** ファイルで確認することができます。マッピングファイルは同じディレクトリーにあります。ヘルパーのフロントエンドとバックエンドは、着信要求のバックエンドへのマッピング時にマッピングファイルを使用します。

証明書

証明書は 2 つの場所に保存されます。

- edge termination および re-encrypt 終端ルートの証明書は **/var/lib/haproxy/router/certs** ディレクトリーに保存されます。
- re-encrypt 終端ルートのバックエンドへの接続に使用される証明書は **/var/lib/haproxy/router/cacerts** ディレクトリーに保存されます。

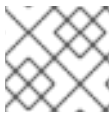
ファイルはルートの namespace および名前指定されます。キー、証明書および CA 証明書は単一ファイルに連結されます。[OpenSSL](#) を使用してこれらのファイルの内容を表示できます。

第27章 高可用性

27.1. 概要

このトピックでは、OpenShift Container Platform クラスターの Pod およびサービスの高可用性の設定について説明します。

IP フェイルオーバーは、ノードセットの仮想 IP (VIP) アドレスのプールを管理します。セットのすべての VIP はセットから選択されるノードによって提供されます。VIP は単一ノードが利用可能である限り提供されます。ノード上で VIP を明示的に配布する方法がないため、VIP のないノードがある可能性も、多数の VIP を持つノードがある可能性もあります。ノードが 1 つのみ存在する場合は、すべての VIP がそのノードに配置されます。



注記

VIP はクラスター外からルーティングできる必要があります。

IP フェイルオーバーは各 VIP のポートをモニターし、ポートがノードで到達可能かどうかを判別します。ポートが到達不能な場合、VIP はノードに割り当てられません。ポートが 0 に設定されている場合、このチェックは抑制されます。[check スクリプト](#)は必要なテストを実行します。

IP フェイルオーバーは **Keepalived** を使用して一連のホストでの外部からアクセスできる VIP アドレスのセットをホストします。各 VIP は 1 度に 1 つのホストによって提供されます。**Keepalived** は VRRP プロトコルを使用して (一連のホストの) どのホストがどの VIP を提供するかを判別します。ホストが利用不可の場合や **Keepalived** が監視しているサービスが応答しない場合は、VIP は一連のホストの内の別のホストに切り換えられます。したがって、VIP はホストが利用可能である限り常に提供されます。

Keepalived を実行するホストが **check** スクリプトを渡す場合、ホストは [プリエンプションストラテジー](#) に応じて、その優先順位および現在の **MASTER** の優先順位に基づいて **MASTER** 状態になります。

管理者は、状態が変更されるたびに呼び出されるスクリプトを **--notify-script=** オプションを使用して提供できます。**Keepalived** は VIP を提供する場合は **MASTER** 状態に、別のノードが VIP を提供する場合は **BACKUP** 状態に、または **check** スクリプトが失敗する場合は **FAULT** 状態になります。[notify スクリプト](#)は、状態が変更されるたびに新規の状態で呼び出されます。

OpenShift Container Platform は、**oc adm ipfailover** コマンドの実行による IP フェイルオーバーのデプロイメント設定の作成をサポートします。IP フェイルオーバーのデプロイメント設定は VIP アドレスのセットを指定し、それらの提供先となるノードのセットを指定します。クラスターには複数の IP フェイルオーバーのデプロイメント設定を持たせることができ、それぞれが固有な VIP アドレスの独自のセットを管理します。IP フェイルオーバー設定の各ノードは IP フェイルオーバー Pod として実行され、この Pod は **Keepalived** を実行します。

VIP を使用してホストネットワーク (例: ルーター) を持つ Pod にアクセスする場合、アプリケーション Pod は ipfailover Pod を実行しているすべてのノードで実行されている必要があります。これにより、いずれの ipfailover ノードもマスターになり、必要時に VIP を提供することができます。アプリケーション Pod が ipfailover のすべてのノードで実行されていない場合、一部の ipfailover ノードが VIP を提供できないか、または一部のアプリケーション Pod がトラフィックを受信できなくなります。この不一致を防ぐために、ipfailover とアプリケーション Pod の両方に同じセクターとレプリケーション数を使用します。

VIP を使用してサービスにアクセスする場合には、いずれのノードもノードの ipfailover セットに入れることができます。それは、(アプリケーション Pod が実行されている場所を問わず) サービスはすべてのノードで到達可能であるためです。ipfailover ノードのいずれもいつでもマスターにすることができます。

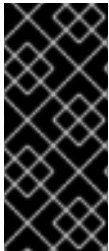
ます。サービスは外部 IP およびサービスポートを使用するか、または nodePort を使用することができます。

サービス定義で外部 IP を使用する場合、VIP は外部 IP に設定され、ipfailover のモニターポートはサービスポートに設定されます。nodePort はクラスターのすべてのノードで開かれ、サービスは VIP をサポートしているいずれのノードからのトラフィックについても負荷分散を行います。この場合、ipfailover のモニターノードはサービス定義で nodePort に設定されます。



重要

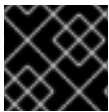
nodePort のセットアップは特権付きの操作で実行されます。



重要

サービス VIP の可用性が高い場合でも、パフォーマンスは依然として影響を受けます。**keepalived** はそれぞれの VIP が設定内のノードによって提供されるようにし、他のノードに VIP がない場合でも複数の VIP が同じノードに配置されることがあります。ipfailover が複数の VIP を同じノードに配置する場合、外部から一連の VIP 間で負荷分散を行う方法は失敗する可能性があります。

ingressIP を使用する場合は、ipfailover を ingressIP 範囲と同じ VIP 範囲を持つように設定できます。また、モニターポートを無効にすることもできます。この場合、すべての VIP がクラスター内の同じノードに表示されます。すべてのユーザーが ingressIP でサービスをセットアップし、これを高い可用性のあるサービスにすることができます。

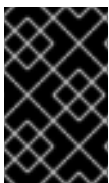


重要

クラスター内の VIP の最大数は 255 です。

27.2. IP フェイルオーバーの設定

oc adm ipfailover コマンドを適切な[オプション](#)と共に使用し、ipfailover デプロイメント設定を作成します。



重要

現時点で、ipfailover はクラウドインフラストラクチャーと互換性がありません。AWS の場合、[AWS コンソールの使用](#)により Elastic Load Balancer (ELB) で OpenShift Container Platform の高可用性を維持することができます。

管理者は、クラスター全体に ipfailover を設定することも、ラベルセクターの定義に基づいてノードのサブセットに ipfailover を設定することもできます。また、複数の IP フェイルオーバーのデプロイメント設定をクラスター内に設定することもでき、それぞれの設定をクラスター内で相互に切り離すことができます。**oc adm ipfailover** コマンドは ipfailover デプロイメント設定を作成し、これによりフェイルオーバー Pod が使用される制約またはラベルに一致する各ノードで実行されるようにします。この Pod は、すべての **Keepalived** デーモン間で VRRP (Virtual Router Redundancy Protocol) を使用する **Keepalived** を実行し、監視されるポートでサービスが利用可能であることを確認し、利用可能でない場合は **Keepalived** が VIP を自動的に浮動させます。

実稼働環境で使用する場合には、2 つ以上のノードで **--selector=<label>** を使用してノードを選択するようにします。また、指定のラベルが付けられたセクターのノード数に一致する **--replicas=<n>** 値を設定します。

oc adm ipfailover コマンドには、**Keepalived** を制御する環境変数を設定するコマンドラインオプションが含まれます。[環境変数](#)は **OPENSIFT_HA_*** で開始され、必要に応じて変更できます。

たとえば、以下のコマンドは **router=us-west-ha** のラベルが付けられたノードのセレクションに対して IP フェイルオーバー設定を作成します (7 仮想 IP を持つ 4 ノードで、ルータープロセスなどのポート 80 でリッスンするサービスをモニタリング)。

```
$ oc adm ipfailover --selector="router=us-west-ha" \
  --virtual-ips="1.2.3.4,10.1.1.100-104,5.6.7.8" \
  --watch-port=80 --replicas=4 --create
```

27.2.1. 仮想 IP アドレス

Keepalived は一連の仮想 IP アドレスを管理します。管理者はこれらすべてのアドレスについて以下の点を確認する必要があります。

- 仮想 IP アドレスは設定されたホストでクラスター外からアクセスできる。
- 仮想 IP アドレスはクラスター内でこれ以外の目的で使用されていない。

各ノードの **Keepalived** は、必要とされるサービスが実行中であるかどうかを判別します。実行中の場合、VIP がサポートされ、**Keepalived** はネゴシエーションに参加してそのノードが VIP を提供するかを決定します。これに参加するノードについては、このサービスが VIP の監視 ポートでリッスンしている、またはチェックが無効にされている必要があります。



注記

セット内の各 VIP は最終的に別のノードによって提供される可能性があります。

27.2.2. チェックおよび通知スクリプト

Keepalived は、オプションのユーザー指定のチェックスクリプトを定期的に行ってアプリケーションの正常性をモニターします。たとえば、このスクリプトは要求を発行し、応答を検証することで web サーバーをテストします。

スクリプトは **oc adm ipfailover** コマンドに **--check-script=<script>** オプションを指定して実行されます。このスクリプトは **PASS** の場合は **0** で終了するか、または **FAIL** の場合は **1** で終了する必要があります。

デフォルトでチェックは 2 秒ごとに実行されますが、**--check-interval=<seconds>** オプションを使用して頻度を変更することができます。

チェックスクリプトが指定されない場合、[TCP 接続](#)をテストする単純なデフォルトスクリプトが実行されます。このデフォルトテストは、モニターポートが **0** の場合は抑制されます。

それぞれの VIP について、**keepalived** はノードの状態を保持します。ノードの VIP は **MASTER**、**BACKUP**、または **FAULT** の状態になります。**FAULT** 状態にないノードのすべての VIP はネゴシエーションに参加し、VIP の **MASTER** を決定します。選ばれなかったすべてのノードは **BACKUP** 状態になります。**MASTER** での **check** スクリプトが失敗すると、VIP は **FAULT** 状態になり、再ネゴシエーションがトリガーされます。**BACKUP** が失敗すると、VIP は **FAULT** 状態になります。**check** スクリプトが **FAULT** 状態の VIP に再度渡されると、その VIP は **FAULT** 状態を終了して **MASTER** のネゴシエーションを行います。結果としてその VIP の状態は **MASTER** または **BACKUP** のいずれかになります。

管理者はオプションの **notify** スクリプトを提供できます。このスクリプトは状態が変更されるたびに呼び出されます。**Keepalived** は以下の 3 つのパラメーターをこのスクリプトに渡します。

- **\$1** - "GROUP"|"INSTANCE"
- **\$2**: グループまたはインスタンスの名前です。
- **\$3**: 新規の状態 ("MASTER"|"BACKUP"|"FAULT") です。

これらのスクリプトは IP フェイルオーバー Pod で実行され、ホストのファイルシステムではなく Pod のファイルシステムを使用します。オプションにはスクリプトへの完全パスが必要です。管理者は Pod でスクリプトを利用可能にし、**notify** スクリプトを実行して結果を抽出できるようにする必要があります。スクリプトを提供する方法として、**ConfigMap** の使用が推奨されます。

check および **notify** スクリプトの完全パス名は、**keepalived** 設定ファイル、**/etc/keepalived/keepalived.conf** に追加されます。これは **keepalived** が起動するたびに読み込まれます。スクリプトは以下のように **ConfigMap** を使って Pod に追加できます。

1. 必要なスクリプトを作成し、これを保持する **ConfigMap** を作成します。スクリプトには入力引数は指定されず、**OK** の場合は **0** を、**FAIL** の場合は **1** を返します。

check スクリプト **mycheckscript.sh**:

```
#!/bin/bash
# Whatever tests are needed
# E.g., send request and verify response
exit 0
```

2. **ConfigMap** を作成します。

```
$ oc create configmap mycustomcheck --from-file=mycheckscript.sh
```

3. スクリプトを Pod に追加する方法として、**oc** コマンドの使用またはデプロイメント設定の編集の 2 つの方法があります。どちらの場合も、マウントされた **configMap** ファイルの **defaultMode** は実行を許可する必要があります。通常は、値 **0755 (493、10 進数)** が使用されます。

- a. **oc** コマンドの使用:

```
$ oc env dc/ipf-ha-router \
    OPENSIFT_HA_CHECK_SCRIPT=/etc/keepalive/mycheckscript.sh
$ oc volume dc/ipf-ha-router --add --overwrite \
    --name=config-volume \
    --mount-path=/etc/keepalive \
    --source='{ "configMap": { "name": "mycustomcheck",
    "defaultMode": 493}}'
```

- b. **ipf-ha-router** デプロイメント設定の編集:

- i. **oc edit dc ipf-ha-router** を使用し、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。

```
...
spec:
  containers:
    - env:
```

```

- name: OPENSIFT_HA_CHECK_SCRIPT ❶
  value: /etc/keepalive/mycheckscript.sh
...
volumeMounts: ❷
- mountPath: /etc/keepalive
  name: config-volume
dnsPolicy: ClusterFirst
...
volumes: ❸
- configMap:
  defaultMode: 0755 ❹
  name: customrouter
  name: config-volume
...

```

- ❶ `spec.container.env` フィールドで、マウントされたスクリプトファイルを参照する `OPENSIFT_HA_CHECK_SCRIPT` 環境変数を追加します。
- ❷ `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。
- ❸ 新規の `spec.volumes` フィールドを追加して ConfigMap に言及します。
- ❹ これはファイルの実行パーミッションを設定します。読み取られる場合は 10 進数 (493) で表示されます。

ii. 変更を保存してエディターを終了します。これにより `ipf-ha-router` が再起動します。

27.2.3. VRRP プリエンプション

ホストが check スクリプトを渡すことで **FAULT** 状態を終了する場合、その新規ホストが現在の **MASTER** 状態にあるホストよりも優先度が低い場合は **BACKUP** になります。ただしそのホストの優先度が高い場合は、プリエンプションストラテジーがクラスター内でのそのルールを決定します。

`nopreempt` ストラテジーは **MASTER** を低優先度のホストから高優先度のホストに移行しません。デフォルトの `preempt 300` の場合、`keepalived` は指定された 300 秒の間待機し、**MASTER** を優先度の高いホストに移行します。

プリエンプションを指定するには、以下を実行します。

- a. `preemption-strategy` を使用して `ipfailover` を作成します。

```

$ oc adm ipfailover --preempt-strategy=nopreempt \
...

```

- b. `oc set env` コマンドを使用して変数を設定します。

```

$ oc set env dc/ipf-ha-router \
--overwrite=true \
OPENSIFT_HA_PREEMPTION=nopreempt

```

- c. `oc edit dc ipf-ha-router` を使用してルーターデプロイメント設定を編集します。

```
...
spec:
  containers:
  - env:
    - name: OPENSIFT_HA_PREEMPTION
      value: nopreempt
...
```

1

27.2.4. Keepalived マルチキャスト

OpenShift Container Platform の IP フェイルオーバーは **keepalived** を内部で使用します。



重要

前述のラベルが付いたノードで **multicast** が有効にされており、それらが 224.0.0.18 (VRRP マルチキャスト IP アドレス) のネットワークトラフィックを許可することを確認します。

keepalived デーモンを起動する前に、起動スクリプトは、マルチキャストトラフィックのフローを許可する **iptables** ルールを検証します。このルールがない場合、起動スクリプトは新規ルールを作成し、これを IP テーブル接続に追加します。この新規ルールが IP テーブルに追加される場所は **--iptables-chain=** オプションによって異なります。**--iptables-chain=** オプションが指定される場合、ルールはオプションで指定されるチェーンに追加されます。そうでない場合は、ルールは **INPUT** チェーンに追加されます。



重要

iptables ルールは、1 つ以上の **keepalived** デーモンがノードで実行されている場合に存在している必要があります。

iptables ルールは、最後の **keepalived** デーモンの終了後に削除できます。このルールは自動的に削除されません。

各ノードで **iptables** ルールを手動で管理できます。(ipfailover が **--iptables-chain=""** オプションで作成されていない限り) 何も存在しない場合にこのルールが作成されます。



重要

手動で追加されたルールがシステム起動後も保持されることを確認する必要があります。

すべての **keepalived** デーモンはマルチキャスト 224.0.0.18 で VRRP を使用してそのピアとネゴシエーションするので注意が必要です。[それぞれの VIP](#) に異なる VRRP-id (0..255 の範囲) が設定されます。

```
$ for node in openshift-node-{5,6,7,8,9}; do    ssh $node <<EOF

export interface=${interface:-"eth0"}
echo "Check multicast enabled ... ";
ip addr show $interface | grep -i MULTICAST

echo "Check multicast groups ... "
```

```
ip maddr show $interface | grep 224.0.0
```

```
EOF
done;
```

27.2.5. コマンドラインオプションおよび環境変数

表27.1 コマンドラインオプションおよび環境変数

オプション	変数名	デフォルト	注記
--watch-port	OPENSIFT_HA_MONITOR_PORT	80	ipfailover Pod は、各 VIP のこのポートに対して TCP 接続を開こうとします。接続が設定されると、サービスは実行中であると見なされます。このポートが 0 に設定される場合、テストは常にパスします。
--interface	OPENSIFT_HA_NETWORK_INTERFACE		使用する ipfailover のインターフェース名で、VRRP トラフィックを送信するために使用されます。デフォルトで eth0 が使用されます。
--replicas	OPENSIFT_HA_REPLICA_COUNT	2	作成するレプリカの数です。これは、ipfailover デプロイメント設定の spec.replicas 値に一致している必要があります。
--virtual-ips	OPENSIFT_HA_VIRTUAL_IPS		複製する IP アドレス範囲の一覧です。これは指定する必要があります (例: 1.2.3.4-6,1.2.3.9)。詳細については、 こちら を参照してください。
--vrrp-id-offset	OPENSIFT_HA_VRRP_ID_OFFSET	0	詳細は、 VRRP ID オフセット を参照してください。
--iptables-chain	OPENSIFT_HA_IPTABLES_CHAIN	INPUT	iptables チェーンの名前であり、 iptables ルールを自動的に追加し、VRRP トラフィックをオンにすることを許可するために使用されます。この値が設定されていない場合、 iptables ルールは追加されません。チェーンが存在しない場合は作成されません。
--check-script	OPENSIFT_HA_CHECK_SCRIPT		Pod のファイルシステム内の、アプリケーションの動作を確認するために定期的に行われるスクリプトの完全パス名です。詳細は、 こちら を参照してください。

オプション	変数名	デフォルト	注記
<code>--check-interval</code>	<code>OPENSHIFT_HA_CHECK_INTERVAL</code>	2	check スクリプトが実行される期間 (秒単位) です。
<code>--notify-script</code>	<code>OPENSHIFT_HA_NOTIFY_SCRIPT</code>		Pod ファイルシステム内の、状態が変更されるたびに実行されるスクリプトの完全パス名です。詳細は、 こちら を参照してください。
<code>--preemption-strategy</code>	<code>OPENSHIFT_HA_PREEMPTION</code>	preempt 300	新たな優先度の高いホストを処理するための戦略です。詳細は、「 VRRP ブリエンプション 」のセクションを参照してください。

27.2.6. VRRP ID オフセット

ipfailover デプロイメント設定で管理される各 ipfailover Pod (ノード/レプリカあたり 1 Pod) は **keepalived** デーモンを実行します。作成される ipfailover デプロイメント設定が多くなると、作成される Pod も多くなり、共通の VRRP ネゴシエーションに参加するデーモンも多くなります。このネゴシエーションはすべての **keepalived** デーモンによって実行され、これはどのノードがどの VIP を提供するかを決定します。

keepalived は内部で固有の vrrp-id を各 VIP に割り当てます。ネゴシエーションはこの vrrp-id セットを使用し、決定後には優先される vrrp-id に対応する VIP が優先されるノードで提供されます。

したがって、ipfailover デプロイメント設定で定義されるすべての VIP について、ipfailover Pod は対応する vrrp-id を割り当てます。これは、`--vrrp-id-offset` から開始し、順序に従って vrrp-id を VIP の一覧に割り当てることによって実行されます。vrrp-id には範囲 1..255 の値を設定できます。

複数の ipfailover デプロイメント設定がある場合、デプロイメント設定の VIP 数を増やす余地があることや vrrp-id 範囲のいずれも重複しないことを確認できるよう `--vrrp-id-offset` を注意して指定する必要があります。

27.2.7. 高可用サービスの設定

以下の例では、ノードのセットに IP フェイルオーバーを指定して可用性の高い **router** および **geo-cache** ネットワークサービスをセットアップする方法について説明します。

1. サービスに使用されるノードにラベルを付けます。の手順は、OpenShift Container Platform クラスターのすべてのノードでサービスを実行し、クラスターのすべてのノード内で固定されない VIP を使用する場合はオプションになります。
以下の例では、地理的区分 US west でトラフィックを提供するノードのラベルを定義します (`ha-svc-nodes=geo-us-west`)。


```
$ oc label nodes openshift-node-{5,6,7,8,9} "ha-svc-nodes=geo-us-west"
```

2. サービスアカウントを作成します。ipfailover を使用したり、(環境ポリシーによって異なる) ルーターを使用する場合は事前に作成された **router** サービスアカウントか、または新規の ipfailover サービスアカウントのいずれかを再利用できます。
以下の例は、**デフォルト** namespace で ipfailover という名前の新規サービスアカウントを作成します。

```
$ oc create serviceaccount ipfailover -n default
```

3. **デフォルト** namespace の ipfailover サービスアカウントを **privileged** SCC に追加します。

```
$ oc adm policy add-scc-to-user privileged  
system:serviceaccount:default:ipfailover
```

4. **router** および **geo-cache** サービスを起動します。



重要

ipfailover は手順 1 のすべてのノードで実行されるため、手順 1 のすべてのノードでルーター/サービスを実行することも推奨されます。

- a. 最初の手順で使用されるラベルに一致するノードでルーターを起動します。以下の例では、ipfailover サービスアカウントを使用して 5 つのインスタンスを実行します。

```
$ oc adm router ha-router-us-west --replicas=5 \  
  --selector="ha-svc-nodes=geo-us-west" \  
  --labels="ha-svc-nodes=geo-us-west" \  
  --service-account=ipfailover
```

- b. 各ノードでレプリカと共に **geo-cache** サービスを実行します。**geo-cache** サービスの実行については、「[設定サンプル](#)」を参照してください。

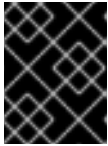


重要

ファイルで参照される **myimages/geo-cache** Docker イメージが使用するイメージに置き換えられていることを確認します。レプリカの数 **geo-cache** ラベルのノード数に変更します。ラベルが最初の手順で使用したものと一致していることを確認します。

```
$ oc create -n <namespace> -f ./examples/geo-cache.json
```

5. **router** および **geo-cache** サービスの ipfailover を設定します。それぞれに独自の VIP があり、いずれも最初の手順の **ha-svc-nodes=geo-us-west** のラベルが付けられた同じノードを使用します。レプリカの数が最初の手順のラベル設定に一覧表示されているノード数と一致していることを確認してください。



重要

router、**geo-cache** および **ipfailover** はすべてデプロイメント設定を作成します。それらの名前はすべて異なっている必要があります。

6. **ipfailover** が必要なインスタンスでモニターする必要がある VIP およびポート番号を指定します。

router の **ipfailover** コマンド:

```
$ oc adm ipfailover ipf-ha-router-us-west \
  --replicas=5 --watch-port=80 \
  --selector="ha-svc-nodes=geo-us-west" \
  --virtual-ips="10.245.2.101-105" \
  --iptables-chain="INPUT" \
  --service-account=ipfailover --create
```

以下は、ポート 9736 でリッスンする **geo-cache** サービスの **oc adm ipfailover** コマンドです。2つの **ipfailover** デプロイメント設定があるため、それぞれの VIP が独自のオフセットを取得できるように **--vrrp-id-offset** を設定する必要があります。この場合 **10** の値は、**ipf-ha-geo-cache** が 10 から開始するために **ipf-ha-router-us-west** には最大 10 の VIP (0-9) を持たせることができることを意味します。

```
$ oc adm ipfailover ipf-ha-geo-cache \
  --replicas=5 --watch-port=9736 \
  --selector="ha-svc-nodes=geo-us-west" \
  --virtual-ips=10.245.3.101-105 \
  --vrrp-id-offset=10 \
  --service-account=ipfailover --create
```

上記のコマンドでは、各ノードに **ipfailover**、**router**、および **geo-cache** Pod があります。各 **ipfailover** 設定の VIP のセットは重複してならず、外部またはクラウド環境の別の場所で使用することはできません。それぞれの例の 5 つの VIP **10.245.{2,3}.101-105** は、2 つの **ipfailover** デプロイメント設定で提供されます。IP フェイルオーバーはどのアドレスがどのノードで提供されるかを動的に選択します。

管理者は、すべての **router** VIP が同じ **router** を参照し、すべての **geo-cache** VIP が同じ **geo-cache** サービスを参照することを前提とした上で VIP アドレスを参照する外部 DNS をセットアップします。1 つのノードが実行中である限り、すべての VIP アドレスが提供されます。

27.2.7.1. IP フェイルオーバー Pod のデプロイ

postgresql-ingress サービスの定義に基づいてノードポート 32439 および外部 IP アドレスでリッスンする **postgresql** をモニターするために **ipfailover** ルーターをデプロイします。

```
$ oc adm ipfailover ipf-ha-postgresql \
  --replicas=1 <1> --selector="app-type=postgresql" <2> \
  --virtual-ips=10.9.54.100 <3> --watch-port=32439 <4> \
  --service-account=ipfailover --create
```

- 1 デプロイするインスタンスの数を指定します。

ipfailover がデプロイされる場所を制限します。

モニターする仮想 IP アドレスです。

各ノード上の ipfailover がモニターするポートです。

27.2.8. 高可用サービスの仮想 IP の動的更新

IP フェイルオーバーのデフォルトのデプロイメント方法として、デプロイメントを再作成します。高可用のルーティングサービスの動的更新を最小限のダウンタイムまたはダウンタイムなしで実行するには、以下を実行する必要があります。

- ローリングアップデート (Rolling Update) ストラテジーを使用するように IP フェイルオーバーサービスデプロイメント設定を更新する。
- 仮想 IP アドレスの更新された一覧またはセットを使用して **OPENSIFT_HA_VIRTUAL_IPS** 環境変数を更新します。

以下の例は、デプロイメントストラテジーおよび仮想 IP アドレスを動的に更新する方法について示しています。

1. 以下を使用して作成された IP フェイルオーバー設定を見てみましょう。

```
$ oc adm ipfailover ipf-ha-router-us-west \
  --replicas=5 --watch-port=80 \
  --selector="ha-svc-nodes=geo-us-west" \
  --virtual-ips="10.245.2.101-105" \
  --service-account=ipfailover --create
```

2. デプロイメント設定を編集します。

```
$ oc edit dc/ipf-ha-router-us-west
```

3. **spec.strategy.type** フィールドを **Recreate** から **Rolling** に更新します。

```
spec:
  replicas: 5
  selector:
    ha-svc-nodes: geo-us-west
  strategy:
    recreateParams:
      timeoutSeconds: 600
    resources: {}
    type: Rolling ❶
```

- ❶ **Rolling** に設定します。

4. 追加の仮想 IP アドレスを含めるように **OPENSIFT_HA_VIRTUAL_IPS** 環境変数を更新します。

```
- name: OPENSIFT_HA_VIRTUAL_IPS
  value: 10.245.2.101-105,10.245.2.110,10.245.2.201-205 ❶
```

- ❶ **10.245.2.110,10.245.2.201-205** が一覧に追加されます。

5. VIP のセットに一致するよう外部 DNS を更新します。

27.3. サービスの **EXTERNALIP** および **NODEPORT** の設定

ユーザーは VIP をサービスの **ExternalIP** として割り当てることができます。**Keepalived** は、各 VIP が ipfailover 設定のノードで提供されることを確認します。要求がそのノードに到達すると、クラスター内のすべてのノードで実行されているサービスがサービスのエンドポイント間で要求の負荷の分散を行います。

NodePorts を ipfailover 監視ポートに設定して、**keepalived** がアプリケーションが実行中であることを確認できるようにします。NodePort はクラスター内のすべてのノードで公開されるため、すべての ipfailover ノードの **keepalived** で利用可能になります。

27.4. INGRESSIP の高可用性

クラウド以外のクラスターでは、ipfailover およびサービスへの **ingressIP** を組み合わせることができます。結果として、ingressIP を使用してサービスを作成するユーザーに高可用サービス性が提供されます。

この方法では、まず **ingressIPNetworkCIDR** 範囲を指定し、次に ipfailover 設定を作成する際に同じ範囲を使用します。

ipfailover はクラスター全体に対して最大 255 の VIP をサポートするため、**ingressIPNetworkCIDR** は **/24** 以下に設定する必要があります。

第28章 IPTABLES

28.1. 概要

システムコンポーネントには、OpenShift Container Platform、コンテナ、および適切なネットワーク操作のためにカーネルの iptables 設定に依存するファイアウォールポリシーを管理するソフトウェアなど、数多くのコンポーネントがあります。さらに、クラスター内のすべてのノードの iptables 設定はネットワークが機能するように正しくなければなりません。

すべてのコンポーネントは、他のコンポーネントが iptables をどのように使用するかを認識せずに独立して iptables を使用します。そのため、あるコンポーネントを別のコンポーネントの設定から分離することが容易になります。さらに、OpenShift Container Platform および Docker サービスは、iptables がそれらがセットアップした時と全く同じ設定であると仮定します。それらは他のコンポーネントによって導入される変更を検出しない場合がありますが、これらを検出する場合は修正の実装により一部の遅れが生じる可能性があります。OpenShift Container Platform は問題をモニターし、解決しますが、Docker サービスはこれを実行しません。



重要

ノード上の iptables 設定に対して加えるいかなる変更も OpenShift Container Platform および Docker サービスの操作に影響を与えないものであることを確認してください。また多くの場合、変更はクラスター内のすべてのノードに対して実行される必要があります。iptables は複数の同時ユーザーを持つように設計されておらず、OpenShift Container Platform および Docker ネットワークに障害が発生する可能性があるため、これを変更する際には注意が必要です。

OpenShift Container Platform は複数のチェーンを提供しますが、それらの 1 つは、管理者が独自の目的で使用する意図されている **OPENSIFT-ADMIN-OUTPUT-RULES** です。

詳細は、「[外部リソースへのアクセスを制限するための iptables ルールの使用](#)」を参照してください。

OpenShift Container Platform および Docker ネットワークが適性に機能するために、カーネル iptables のチェーン、チェーンの順序、およびルールがクラスター内の各ノードに適切に設定される必要があります。システム内には、カーネル iptables と対話し、OpenShift Container Platform および Docker サービスに意図せずに影響を与える可能性のあるツールやサービスがシステムいくつかあります。

28.2. IPTABLES

iptables ツールは、Linux カーネルの IPv4 パケットフィルターのテーブルを設定し、維持し、検査するために使用できます。

ファイアウォールなどの他の使用とは別に、OpenShift Container Platform および Docker サービスはチェーンを一部のテーブルで管理します。チェーンは特定の順序で挿入され、ルールはそれぞれのニーズに応じて固有のものになります。

注意

iptables --flush [chain] は、キーが必要な設定を削除できます。このコマンドを実行しないでください。

28.3. IPTABLES.SERVICE

iptables サービスはローカルのネットワークファイアウォールをサポートします。これは、iptables 設定を完全に制御することを想定します。これが起動すると、詳細な iptables 設定をフラッシュし、それを復元します。ルールはその設定ファイル **/etc/sysconfig/iptables** から復元されます。設定ファイルは操作時に最新の状態に保たれないため、動的に追加されたルールは毎回の再起動時に失われます。



警告

iptables.service を停止し、起動することにより、OpenShift Container Platform および Docker で必要な設定が破棄されます。OpenShift Container Platform および Docker にはこの変更は通知されません。

```
# systemctl disable iptables.service
# systemctl mask iptables.service
```

iptables.service を実行する必要がある場合、制限された設定を設定ファイルに維持し、OpenShift Container Platform および Docker を使用してそれらが必要とするルールをインストールするようにします。

iptables.service 設定は以下から読み取られます。

```
/etc/sysconfig/iptables
```

ルールの永続的な変更を実行するには、このファイルで変更を編集します。Docker または OpenShift Container Platform ルールは含めないようにしてください。

iptables.service がノードで起動または再起動した後は、Docker サービスおよび **atomic-openshift-node.service** を再起動して、必要な iptables 設定を再構築する必要があります。



重要

Docker サービスの再起動により、ノードで実行されているすべてのコンテナが停止し、再起動されます。

```
# systemctl restart iptables.service
# systemctl restart docker
# systemctl restart atomic-openshift-node.service
```

第29章 ストラテジーによるビルドのセキュリティ保護

29.1. 概要

OpenShift Container Platform のビルドは、Docker デモンソケットにアクセスできる特権付きコンテナで実行されます。セキュリティ対策として、ビルドおよびそれらのビルドに使用されるストラテジーを実行するユーザーを制限することを推奨します。カスタムビルドは、ソースビルドよりも安全性が低いと言えます。それらはノードの Docker ソケットへの完全なアクセスを持つ可能性があり、それらのアクセスでビルド内で任意のコードを実行する可能性があるためです。そのため、これはデフォルトで無効にされます。Docker ビルドのパーミッションを付与する場合についても、Docker ビルドロジックの脆弱性により権限がホストノードで付与される可能性があるために注意が必要です。

デフォルトで、ビルドを作成できるすべてのユーザーには Docker および Source-to-Image (S2I) ビルドストラテジーを使用するためのパーミッションが付与されます。cluster-admin 権限を持つユーザーは、このページの「ユーザーへのビルドストラテジーのグローバルな制限」セクションで言及されているようにカスタムビルドストラテジーを有効にすることができます。

許可ポリシーを使用して、どのユーザーがどのビルドストラテジーを使用してビルドできるかについて制限することができます。各ビルドには、対応するビルドサブリソースがあります。ストラテジーを使用してビルド作成するには、ユーザーにビルドを作成するパーミッション および ビルドストラテジーのサブリソースで作成するパーミッションがなければなりません。ビルドストラテジーのサブリソースでの create パーミッションを付与するデフォルトロールが提供されます。

表29.1 ビルドストラテジーのサブリソースおよびロール

ストラテジー	サブリソース	ロール
Docker	ビルド/docker	system:build-strategy-docker
Source-to-Image (S2I)	ビルド/ソース	system:build-strategy-source
カスタム	ビルド/カスタム	system:build-strategy-custom
JenkinsPipeline	ビルド/jenkinspipeline	system:build-strategy-jenkinspipeline

29.2. ビルドストラテジーのグローバルな無効化

特定のビルドストラテジーへのアクセスをグローバルに禁止するには、cluster-admin 権限を持つユーザーとしてログインし、system:authenticated グループから対応するロールを削除します。

```
$ oc adm policy remove-cluster-role-from-group system:build-strategy-custom system:authenticated
$ oc adm policy remove-cluster-role-from-group system:build-strategy-docker system:authenticated
$ oc adm policy remove-cluster-role-from-group system:build-strategy-source system:authenticated
$ oc adm policy remove-cluster-role-from-group system:build-strategy-jenkinspipeline system:authenticated
```

3.2 よりも前のバージョンでは、ビルドストラテジーのサブリソースは **admin** および **edit** ロールに組み込まれていました。ビルドストラテジーのサブリソースがこれらのロールから削除されていることも確認してください。

```
$ oc edit clusterrole admin
$ oc edit clusterrole edit
```

それぞれのロールについて、無効にするストラテジーのリソースに対応する行を削除します。

例29.1 admin の Docker ビルドストラテジーの無効化

```
kind: ClusterRole
metadata:
  name: admin
...
rules:
- resources:
  - builds/custom
  - builds/docker ①
  - builds/source
...
...
```

- ① **admin** ロールを持つユーザーに対して Docker ビルドをグローバルに無効にするためにこの行を削除します。

29.3. ユーザーへのビルドストラテジーのグローバルな制限

一連の特定ユーザーのみが特定のストラテジーでビルドを作成できるようにするには、以下を実行します。

1. [ビルドストラテジーへのグローバルアクセスを無効にします。](#)
2. ビルドストラテジーに対応するロールを特定ユーザーに割り当てます。たとえば、**system:build-strategy-docker** クラスターロールをユーザー **devuser** に追加するには、以下を実行します。

```
$ oc adm policy add-cluster-role-to-user system:build-strategy-docker devuser
```



警告

ユーザーに対して **builds/docker** サブリソースへのクラスターレベルでのアクセスを付与することは、そのユーザーがビルドを作成できるすべてのプロジェクトにおいて、Docker ストラテジーを使ってビルドを作成できることを意味します。

29.4. プロジェクト内でのユーザーへのビルドストラテジーの制限

ユーザーにビルドストラテジーをグローバルに付与するのと同様に、プロジェクト内の特定ユーザーのセットのみが特定ストラテジーでビルドを作成できるようにするには、以下を実行します。

1. [ビルドストラテジーへのグローバルアクセスを無効にします](#)。
2. ビルドストラテジーに対応するロールをプロジェクト内の特定ユーザーに付与します。たとえば、プロジェクト **devproject** 内の **system:build-strategy-docker** ロールをユーザー **devuser** に追加するには、以下を実行します。

```
$ oc adm policy add-role-to-user system:build-strategy-docker  
devuser -n devproject
```

第30章 SECCOMP を使用したアプリケーション機能の制限

30.1. 概要

seccomp (セキュアコンピューティングモード) は、アプリケーションが行うシステム呼び出しのセットを制限し、クラスター管理者が OpenShift Container Platform で実行されるワークロードのセキュリティを強化するために使用されます。

seccomp サポートは Pod 設定の 2 つのアノテーションを使用して有効になります。

- **seccomp.security.alpha.kubernetes.io/pod**: Pod のすべてのコンテナに適用されるプロファイルです (上書きなし)。
- **container.seccomp.security.alpha.kubernetes.io/<container_name>**: コンテナ固有のプロファイルです (上書きあり)。



重要

デフォルトで、コンテナは **unconfined** seccomp 設定で実行されます。

詳細な設計情報については、[seccomp 設計についてのドキュメント](#)を参照してください。

30.2. SECCOMP の有効化

seccomp は Linux カーネルの 1 つの機能です。seccomp がシステムで有効にされていることを確認するには、以下を実行します。

```
$ cat /boot/config-`uname -r` | grep CONFIG_SECCOMP=
CONFIG_SECCOMP=y
```

30.3. OPENSIFT CONTAINER PLATFORM での SECCOMP の設定

seccomp プロファイルは json ファイルであり、システムコールを提供し、システムコールの呼び出し時に取るべき適切なアクションを実行します。

1. seccomp プロファイルを作成します。
多くの場合は[デフォルトのプロファイル](#)だけで十分ですが、クラスター管理者は個別システムのセキュリティ制約を定義する必要があります。

独自のカスタムプロファイルを作成するには、**seccomp-profile-root** ディレクトリーですべてのノードのファイルを作成します。

デフォルトの **docker/default** プロファイルを使用している場合は、これを作成する必要はありません。

2. ノードをプロファイルが保存される **seccomp-profile-root** を使用するように設定します。**node-config.yaml** で **kubeletArguments** を使用するには、以下を実行します。

```
kubeletArguments:
  seccomp-profile-root:
    - "/your/path"
```


3. 変更を適用するためにノードサービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

4. 使用できるプロファイルを制御し、デフォルトプロファイルを設定するために、**seccompProfiles** フィールドで **SCC を設定** します。最初のプロファイルがデフォルトとして使用されます。

seccompProfiles フィールドで使用できる形式には以下が含まれます。

- **docker/default**: コンテナランタイムのデフォルトプロファイルです (いずれのプロファイルも不要です)。
- **unconfined**: 拘束のないプロファイルで、seccomp を無効にします。
- **localhost/<profile-name>**: ノードのローカル seccomp プロファイルの root にインストールされるプロファイルです。
たとえば、デフォルトの **docker/default** プロファイルを使用している場合、以下で **restricted (制限付き) SCC** を設定します。

```
seccompProfiles:
- docker/default
```

30.4. OPENSIFT CONTAINER PLATFORM でのカスタム SECCOMP プロファイルの設定

クラスターの Pod が **restricted (制限付き) SCC** でカスタムプロファイルで実行されるようにするには、以下を実行します。

1. **seccomp-profile-root** に seccomp プロファイルを作成します。
2. **seccomp-profile-root** を設定します。

```
kubeletArguments:
  seccomp-profile-root:
  - "/your/path"
```

3. 変更を適用するためにノードサービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

4. **restricted (制限付き) SCC** を設定します。

```
seccompProfiles:
- localhost/<profile-name>
```

第31章 SYSCTL

31.1. 概要

sysctl 設定は Kubernetes 経由で公開され、ユーザーがコンテナ内の namespace の特定のカーネルパラメーターをランタイム時に変更できるようにします。namespace を使用する sysctl のみを Pod 上で独立して設定できます。sysctl に namespace が使用されていない場合 (この状態は **ノードレベル** と呼ばれる)、OpenShift Container Platform 内で設定することはできません。さらに **安全** とみなされる sysctl のみがデフォルトでホワイトリストに入れられます。他の **安全でない** sysctl はノードで手動で有効にし、ユーザーが使用できるようにできます。

31.2. SYSCTL について

Linux では、管理者は sysctl インターフェースを使ってランタイム時にカーネルパラメーターを変更することができます。パラメーターは **/proc/sys/** 仮想プロセスファイルシステムで利用できます。これらのパラメーターは以下を含む各種のサブシステムを対象とします。

- カーネル (共通のプレフィックス: **kernel.**)
- ネットワーク (共通のプレフィックス: **net.**)
- 仮想メモリー (共通のプレフィックス: **vm.**)
- MDADM (共通のプレフィックス: **dev.**)

追加のサブシステムについては、[カーネルのドキュメント](#)で説明されています。すべてのパラメーターの一覧を取得するには、以下を実行できます。

```
$ sudo sysctl -a
```

31.3. NAMESPACE を使用した SYSCTL VS ノードレベルの SYSCTL

現時点の Linux カーネルでは、数多くの sysctl に **namespace** が使用されています。これは、それらをノードの各 Pod に対して個別に設定できることを意味します。namespace の使用は、sysctl を Kubernetes 内の Pod 環境でアクセス可能にするための要件になります。

以下の sysctl は namespace を使用するものとして知られている sysctl です。

- **kernel.shm***
- **kernel.msg***
- **kernel.sem**
- **fs.mqueue.***
- **net.***

namespace が使用されていない sysctl は **ノードレベル** と呼ばれており、クラスター管理者がノードの基礎となる Linux ディストリビューションを使用 (例: **/etc/sysctls.conf** を使用) するか、または特権付きコンテナで DaemonSet を使用することによって手動で設定する必要があります。



注記

特殊な sysctl が設定されたノードにテイントのマークを付けることを検討してください。それらの sysctl 設定を必要とする Pod のみをそれらのノードにスケジューリングします。[Kubernetes の テイントおよび容認 \(Toleration\) 機能](#)を使用してこれを実施します。

31.4. 安全 VS 安全でない SYSCTL

sysctl は **安全な** および **安全でない** sysctl に分類されます。適切な namespace の設定に加えて、安全な sysctl は同じノード上の Pod 間で適切に分離する必要があります。つまり、Pod ごとに安全な sysctl を設定することについて以下を留意する必要があります。

- この設定はノード上の他の Pod に影響を与えないものである。
- この設定はノードの正常性に与えることを許可しないものである。
- この設定は Pod のリソース制限を超える CPU またはメモリーリソースの取得を許可しないものである。

namespace を使用した sysctl は必ずしも常に安全であると見なされる訳ではありません。

OpenShift Container Platform 3.3.1 の場合、以下の sysctl が安全なセットでサポートされています (ホワイトリスト化されています)。

- **kernel.shm_rmid_forced**
- **net.ipv4.ip_local_port_range**

この一覧は、kubelet が分離メカニズムのサポートを強化する今後のバージョンでさらに拡張されます。

すべての安全な sysctl はデフォルトで有効にされます。すべての安全でない sysctl はデフォルトで無効にされ、ノードごとにクラスター管理者によって手動で許可される必要があります。無効にされた安全でない sysctl が設定された Pod はスケジューリングされますが、起動に失敗します。



警告

安全でないという性質上、安全でない sysctl は各自の責任で使用されます。場合によっては、コンテナの正しくない動作やリソース不足、またはノードの完全な破損などの深刻な問題が生じる可能性があります。

31.5. 安全でない SYSCTL の有効化

上記の警告を念頭に置いた上で、クラスター管理者は、高パフォーマンスまたはリアルタイムのアプリケーション調整などの非常に特殊な状況で特定の安全でない sysctl を許可することができます。

安全でない sysctl を使用する必要がある場合、クラスター管理者はノードでそれらを個別に有効にする必要があります。namespace を使用した sysctl のみをこの方法で有効にできます。

1. 「[ノードリソースの設定](#)」で説明されているように、必要な安全でない sysctl を設定するには `/etc/origin/node/node-config.yaml` ファイルの `kubeletArguments` フィールドを使用します。

```
kubeletArguments:
  experimental-allowed-unsafe-sysctls:
    - "kernel.msg*,net.ipv4.route.min_pmtu"
```

2. 変更を適用するためにノードサービスを再起動します。

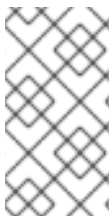
```
# systemctl restart atomic-openshift-node
```

31.6. POD 用の SYSCTL の設定

sysctl はアノテーションを使用して Pod に設定されます。それらは同じ Pod のすべてのコンテナに適用されます。

以下は、安全な sysctl および安全でない sysctl の各種のアノテーションを使用した例です。

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
  annotations:
    security.alpha.kubernetes.io/sysctls: kernel.shm_rmid_forced=1
    security.alpha.kubernetes.io/unsafe-sysctls:
      net.ipv4.route.min_pmtu=1000,kernel.msgmax=1 2 3
spec:
  ...
```



注記

安全でない sysctl が指定された Pod は、それらの 2 つの安全でない sysctl を明示的に有効にしていないノードでの起動に失敗します。ノードレベルの sysctl の場合のようにそれらを Pod を正しいノードにスケジュールするには、[テイント](#)および[容認機能](#)または[ノードのラベル](#)を使用します。

第32章 データストア層でのデータの暗号化

32.1. 概要

このトピックでは、データストア層でシークレットデータの暗号化を有効にし、これを設定する方法について説明します。サンプルでは **secrets** リソースを使用していますが、**configmaps** などのすべてのリソースを暗号化することができます。



重要

この機能を使用するには、etcd v3 以降が必要になります。

32.2. 設定および暗号がすでに有効にされているかどうかの判別

データ暗号化をアクティブにするには、`--experimental-encryption-provider-config` 引数を Kubernetes API サーバーに渡します。

master-config.yaml の抜粋

```
kubernetesMasterConfig:
  apiServerArguments:
    experimental-encryption-provider-config:
      - /path/to/encryption-config.yaml
```

master-config.yaml およびその形式についての詳細は、「[マスター設定ファイル](#)」のトピックを参照してください。

32.3. 暗号化設定について

すべての利用可能なプロバイダーを含む暗号化設定ファイル

```
kind: EncryptionConfig
apiVersion: v1
resources: ❶
  - resources: ❷
    - secrets
  providers: ❸
    - aescbc: ❹
      keys:
        - name: key1 ❺
          secret: c2VjcmV0IGlzIHNLy3VyZQ== ❻
        - name: key2
          secret: dGhpcyBpcyBwYXNzd29yZA==
    - secretbox:
      keys:
        - name: key1
          secret: YWJjZGVmZ2hpamtsbW5vcHFyc3R1dnd4eXoxMjM0NTY=
    - aesgcm:
      keys:
        - name: key1
          secret: c2VjcmV0IGlzIHNLy3VyZQ==
        - name: key2
```

```
secret: dGhpcyBpcyBwYXNzd29yZA==
- identity: {}
```

- 1 **resources** のそれぞれの配列項目は分離した設定であり、詳細な設定が含まれます。
- 2 **resources.resources** フィールドは暗号化が必要な Kubernetes リソース名 (**resource** または **resource.group**) の配列です。
- 3 **providers** 配列は、順序付けられた[使用可能な暗号化プロバイダーの一覧](#)です。エントリーごとに 1 つのプロバイダータイプ (**identity** または **aescbc**) のみを指定できますが、同じ項目に両方を指定することはできません。
- 4 一覧の最初のプロバイダーがストレージに移動するリソースを暗号化するために使用されます。
- 5 シークレットの任意の名前です。
- 6 Base64 のエンコーディングされたランダムキーです。異なるプロバイダーが異なるキーの長さを指定します。詳細は、「[キーの生成方法](#)」についての説明を参照してください。

ストレージからのリソースの読み取り時に、保存されたデータに一致する各プロバイダーはデータの復号化を順番に試行します。情報またはシークレットキーの不一致により保存データを読み取れるプロバイダーがない場合にエラーが返され、クライアントがそのリソースにアクセスできなくなります。



重要

リソースが暗号化設定で読み取れない場合 (キーの変更により)、キーを基礎となる etcd ディレクトリーから削除することのみが必要になります。そのリソースの読み取りを試行する呼び出しは、キーが削除されるか、または有効な復号化キーが提供されない限り失敗します。

32.3.1. 利用可能なプロバイダー

名前	暗号化	強度	速度	キーの長さ	他の考慮事項
identity	なし	該当なし	該当なし	該当なし	暗号化なしのそのままの状態で作成されたリソースです。最初のプロバイダーとして設定される場合、リソースは新規の値が書き込まれるときに復号化されます。
aescbc	PKCS#7 パディングが設定された AES-CBC	最も強い	高速	32 バイト	暗号化に推奨されるオプションですが、 secretbox よりも若干遅くなる可能性があります。
secretbox	XSalsa20 および Poly1305	強い	より高速	32 バイト	より新しい標準であり、高レベルのレビューが必要な環境では受け入れ可能と見なされない可能性があります。

名前	暗号化	強度	速度	キーの長さ	他の考慮事項
aesgcm	AES-GCM およびランダム初期化ベクター (IV)	200,000 回の書き込みごとにローテーションが必要です。	最速	16、24、または 32 バイト	自動化されたキーの回転スキームが実行される場合以外には、 使用することが推奨されません 。

各プロバイダーは複数のキーをサポートします。キーは復号化の順序で試行されます。プロバイダーが最初のプロバイダーの場合、最初のキーが暗号化に使用されます。



注記

Kubernetes には適切な nonce ジェネレーターがないため、AES-GCM の nonce としてランダム IV を使用します。AES-GCM では適切な nonce がセキュアな状態であることが求められるため、AES-GCM は推奨されません。200,000 回の書き込み制限は nonce の致命的な誤用の可能性を比較的強く抑えます。

32.4. データの暗号化

新規の暗号化設定ファイルを作成します。

```
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
    - secrets
  providers:
    - aescbc:
      keys:
        - name: key1
          secret: <BASE 64 ENCODED SECRET>
    - identity: {}
```

新規シークレットを作成するには、以下を実行します。

- 1. 32 バイトのランダムキーを生成し、これを base64 でエンコーディングします。たとえば、Linux および macOS では、以下を使用します。

```
$ head -c 32 /dev/urandom | base64
```



重要

暗号化キーは、`/dev/urandom` などの暗号で保護された乱数ジェネレーターで生成する必要があります。Golang の `math/random` や Python の `random.random()` などは適していません。

- 2. この値を `secret` フィールドに配置します。

3. API サーバーを再起動します。

```
# systemctl restart atomic-openshift-master-api
```



重要

暗号化プロバイダー設定ファイルには、etcd の内容を復号化できるキーが含まれるため、マスター API サーバーを実行するユーザーのみがこれを読み取れるようにマスターでパーミッションを適切に制限する必要があります。

32.5. データが暗号化されていることの確認

データは etcd に書き込まれる際に暗号化されます。API サーバーの再起動後、新たに作成されたか、または更新されたシークレットは保存時に暗号化されます。これを確認するには、**etcdctl** コマンドラインプログラムを使用してシークレットの内容を検索できます。

1. **default** の namespace に、**secret1** という新規シークレットを作成します。

```
$ oc create secret generic secret1 -n default --from-literal=mykey=mydata
```

2. **etcdctl** コマンドラインを使用し、etcd からシークレットを読み取ります。

```
$ ETCDCTL_API=3 etcdctl get /kubernetes.io/secrets/default/secret1 -w fields [...] | grep Value
```

[...] には、etcd サーバーに接続するために追加の引数を指定する必要があります。

最終的なコマンドは以下と同様になります。

```
$ ETCDCTL_API=3 etcdctl get /kubernetes.io/secrets/default/secret1 -w fields \
--cacert=/var/lib/origin/openshift.local.config/master/ca.crt \
--key=/var/lib/origin/openshift.local.config/master/master.etcd-client.key \
--cert=/var/lib/origin/openshift.local.config/master/master.etcd-client.crt \
--endpoints 'https://127.0.0.1:4001' | grep Value
```

3. 上記のコマンド出力には、**aescbc** プロバイダーが結果として生成されるデータを暗号化したことを示す **k8s:enc:aescbc:v1:** のプレフィックスが付けられます。
4. シークレットが API 経由で取得される場合は、正しく復号化されていることを確認します。

```
$ oc get secret secret1 -n default -o yaml | grep mykey
```

これは **mykey: bXlkYXRh** と一致するはずです。

32.6. すべてのシークレットが暗号化されていることの確認

シークレットは書き込み時に暗号化されるため、シークレットの更新を実行するとその内容は暗号化されることになります。


```
$ oc adm migrate storage --include=secrets --confirm
```

このコマンドはすべてのシークレットを読み取り、次にサーバー側の暗号化を適用するようにそれらを更新します。書き込みの競合のためにエラーが発生する場合は、コマンドを再試行してください。

大規模クラスターの場合、シークレットを namespace 別に細分化するか、または更新をスクリプト化することができます。

32.7. 復号化キーのローテーション

複数の API サーバーが実行されている高可用デプロイメントがある場合などに、ダウンタイムを発生させずにシークレットを変更するには、複数の手順からなる操作が必要になります。

1. 新規キーを生成し、これをすべてのサーバーで同時プロバイダーの 2 つ目のキーエントリーとして追加します。
2. すべての API サーバーを再起動し、各サーバーが新規キーを使用して復号化できるようにします。



注記

単一 API サーバーを使用している場合は、この手順を省略できます。

```
# systemctl restart atomic-openshift-master-api
```

3. 新規キーを **keys** 配列の最初のエントリーにし、これが設定で暗号化に使用されるようにします。
4. すべての API サーバーを再起動し、各サーバーが新規キーを使用して暗号化できるようにします。

```
# systemctl restart atomic-openshift-master-api
```

5. 以下を実行し、新規キーですべての既存シークレットを暗号化します。

```
$ oc adm migrate storage --include=secrets --confirm
```

6. 新規キーを使用して etcd をバックアップし、すべてのシークレットを更新した後に、古い復号化キーを設定から削除します。

32.8. データの復号化

データストア層で暗号化を無効にするには、以下を実行します。

1. **identity** プロバイダーを、設定の最初のエントリーとして配置します。

```
kind: EncryptionConfig
apiVersion: v1
resources:
- resources:
- secrets
providers:
- identity: {}
```

```
- aescbc:
  keys:
    - name: key1
      secret: <BASE 64 ENCODED SECRET>
```

1. すべての API サーバーを再起動します。

```
# systemctl restart atomic-openshift-master-api
```

2. 以下を実行し、すべてのシークレットの復号化を強制的に実行します。

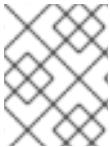
```
$ oc adm migrate storage --include=secrets --confirm
```

第33章 IPSEC を使用したホストの暗号化

33.1. 概要

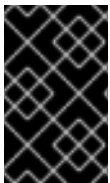
IPsec は、インターネットプロトコル (IP) を使用して通信するすべてのマスターとノードホスト間の通信を暗号化することによって OpenShift Container Platform クラスターのトラフィックを保護します。

このトピックでは、すべてのクラスター管理および Pod データトラフィックを含め、OpenShift Container Platform ホストが IP アドレスを受信する IP サブセット全体の通信のセキュリティーを保護する方法について説明します。



注記

OpenShift Container Platform 管理トラフィックは HTTPS を使用するため、IPsec の有効化により 2 度目の管理トラフィックの暗号化が実行されることになります。



重要

この手順はクラスターの各マスターホストで繰り返し、その後にノードホストで実行される必要があります。IPsec が有効にされていないホストは、これが有効にされているホストと通信することができません。

33.2. ホストの暗号化

33.2.1. 前提条件

- **libreswan** 3.15 以降がクラスターホストにインストールされていることを確認します。[便宜的なグループ \(opportunistic group\) 機能](#)が必要な場合は、**libreswan** バージョン 3.19 以降が必要になります。
- MTU を IPsec ヘッダーのスペースを確保するように設定する方法についての詳細は、「[ノードでの Pod ネットワークの設定](#)」セクションを参照してください。このトピックでは、62 バイトを必要とする IPsec 設定について説明します。クラスターが MTU が 1500 のイーサネットワークで動作している場合、IPsec および SDN のカプセル化のオーバーヘッドに対応するために SDN MTU は 1388 にする必要があります。

OpenShift Container Platform 設定で MTU を変更した後に、SDN インターフェースを削除し、OpenShift Container Platform ノードプロセスを再起動して SDN に変更を認識させる必要があります。

```
# systemctl stop atomic-openshift-node
# ovs-vsctl del-br br0
# systemctl start atomic-openshift-node
```

33.2.2. 証明書での IPsec の設定

デフォルトで、OpenShift Container Platform は手動で相互に認証された HTTPS 通信でクラスター管理通信のセキュリティーを保護します。つまり、クライアント (例: OpenShift Container Platform ノード) およびサーバー (例: OpenShift Container Platform API サーバー) は相互にそれぞれの証明書を送信し、それは既知の認証局 (CA) に対してチェックされます。これらの証明書はクラスターの設定時に生成され、通常は各ホストに存在します。また、これらの証明書を使用して IPsec での Pod 通信のセキュリティーを保護することもできます。

この手順は、各ホストに以下があることを前提とします。

- クラスター CA ファイル
- ホストのクライアント証明書ファイル
- ホストのプライベートキーファイル

1. **libreswan** 証明書データベースへのインポート後に証明書のニックネームが何であるかを判別します。ニックネームは証明書のサブジェクトの共通名 (CN) から直接取られます。

```
# openssl x509 \
-in /path/to/client-certificate -subject -noout | \
sed -n 's/.*CN=\.*/\1/p'
```

2. **openssl** を使用してクライアント証明書、CA 証明書、およびプライベートキーファイルを **PKCS#12** ファイルに追加します。これは、複数の証明書およびキーの共通ファイル形式です。

```
# openssl pkcs12 -export \
-in /path/to/client-certificate \
-inkey /path/to/private-key \
-certfile /path/to/certificate-authority \
-passout pass: \
-out certs.p12
```

3. **PKCS#12** ファイルを **libreswan** 証明書データベースにインポートします。-w オプションは、パスワードが一時的で **PKCS#12** ファイルに割り当てられないため、空のままになります。

```
# ipsec initnss
# pk12util -i certs.p12 -d sql:/etc/ipsec.d -w ""
# rm certs.p12
```

33.2.3. libreswan IPsec ポリシー

必要な証明書が **libreswan** 証明書データベースにインポートされた後に、それらを使用してクラスター内のホスト間の通信をセキュリティー保護するポリシーを作成します。

libreswan 3.19 以降を使用している場合、[便宜的なグループ \(opportunistic group\) 設定](#)が推奨されます。これ以外の場合は、明示的な接続が必要になります。

33.2.3.1. 便宜的なグループ (opportunistic group) 設定

以下の設定は 2 つの **libreswan** 接続を作成します。最初の設定は OpenShift Container Platform 証明書を使用してトラフィックを暗号化し、2 つ目の設定はクラスターの外部トラフィック用に暗号化に対する例外を作成します。

1. 以下を **/etc/ipsec.d/openshift-cluster.conf** ファイルに配置します。

```
conn private
left=%defaulttroute
leftid=%fromcert
# our certificate
```

```

leftcert="NSS Certificate DB:<cert_nickname>" ❶
right=%opportunisticgroup
rightid=%fromcert
# their certificate transmitted via IKE
rightca=%same
ikev2=insist
authby=rsasig
failureshunt=drop
negotiationshunt=hold
auto=ondemand

conn clear
left=%defaultroute
right=%group
authby=never
type=passthrough
auto=route
priority=100

```

❶ <cert_nickname> を、手順 1 の証明書ニックネームに置き換えます。

2. **libreswan** に対して、**/etc/ipsec.d/policies/** のポリシーファイルを使用して各ポリシーを適用する IP サブネットおよびホストを示します。このファイルでは、それぞれの設定された接続に対応するポリシーファイルが設定されます。そのため、上記の例では、**private** および **clear** の 2 つの接続のそれぞれに **/etc/ipsec.d/policies/** のファイルが設定されます。

/etc/ipsec.d/policies/private には、ホストの IP アドレスの受信元であるクラスターの IP サブネットが含まれる必要があります。これにより、デフォルトでは、リモートホストのクライアント証明書がローカルホストの認証局の証明書に対して認証される場合、クラスターサブネットのホスト間のすべての通信が暗号化されることになります。リモートホストの証明書が認証されない場合、2 つのホスト間のすべてのトラフィックがブロックされます。

たとえば、すべてのホストが **172.16.0.0/16** アドレス空間のアドレスを使用するように設定される場合、**private** ポリシーファイルには **172.16.0.0/16** が含まれることになります。暗号化する追加サブセットの任意の数がこのファイルに追加され、それらのサブネットへのすべてのトラフィックでも IPsec が使用されることになります。

3. トラフィックがクラスターに出入りすることを確認するためにすべてのホストとサブネットゲートウェイ間の通信の暗号化を解除します。ゲートウェイを **/etc/ipsec.d/policies/clear** ファイルに追加します。

```
172.16.0.1/32
```

追加のホストおよびサブネットをこのファイルに追加できます。これにより、これらのホストおよびサブネットへのすべてのトラフィックの暗号が解除されます。

33.2.3.2. 明示的な通信設定

この設定では、各 IPsec ノード設定がクラスター内の他のすべてのノードの設定を明示的に一覧表示する必要があります。各ノードで Ansible などの設定管理ツールを使用してこのファイルを生成することが推奨されます。

1. また、この設定では各ノードの完全な証明書サブジェクトをその他のノードの設定に配置する必要があります。このサブジェクトをノードの証明書から読み取るには、**openssl** を使用します。

-

```
# openssl x509 \
-in /path/to/client-certificate -text | \
grep "Subject:" | \
sed 's/[[:blank:]]*Subject: //'
```

- 以下の行を、クラスター内のその他のノード用に各ノードの `/etc/ipsec.d/openshift-cluster.conf` ファイルに配置します。

```
conn <other_node_hostname>
    left=<this_node_ip> ①
    leftid="CN=<this_node_cert_nickname>" ②
    lefttrsasigkey=%cert
    leftcert=<this_node_cert_nickname> ③
    right=<other_node_ip> ④
    rightid="<other_node_cert_full_subject>" ⑤
    righttrsasigkey=%cert
    auto=start
    keyingtries=%forever
```

- ① `<this_node_ip>` をこのノードのクラスター IP アドレスに置き換えます。
- ② ③ `<this_node_cert_nickname>` を手順 1 のノードの証明書ニックネームに置き換えます。
- ④ `<other_node_ip>` を他のノードのクラスター IP アドレスに置き換えます。
- ⑤ `<other_node_cert_full_subject>` を上記の他のノードの証明書に置き換えます。たとえば、`"O=system:nodes,CN=openshift-node-45.example.com"` のようになります。

- 以下を各ノードの `/etc/ipsec.d/openshift-cluster.secrets` ファイルに配置します。

```
: RSA "<this_node_cert_nickname>" ①
```

- ① `<this_node_cert_nickname>` を手順 1 のノードの証明書ニックネームに置き換えます。

33.3. IPSEC ファイアウォール設定

クラスター内のすべてのノードは、IPSec 関連のネットワークトラフィックを許可する必要があります。これには、UDP ポート 500 だけでなく IP プロトコル番号の 50 と 51 が含まれます。

たとえば、クラスターノードがインターフェース `eth0` で通信する場合、以下のようになります。

```
-A OS_FIREWALL_ALLOW -i eth0 -p 50 -j ACCEPT
-A OS_FIREWALL_ALLOW -i eth0 -p 51 -j ACCEPT
-A OS_FIREWALL_ALLOW -i eth0 -p udp --dport 500 -j ACCEPT
```



注記

また IPSec は、NAT traversal に UDP ポート 4500 を使用します。ただし、これは通常のクラスターデプロイメントに適用することはできません。

33.4. IPSEC の開始および終了

1. **ipsec** サービスを開始し、新規の設定およびポリシーを読み込み、暗号化を開始します。

```
# systemctl start ipsec
```

2. **ipsec** サービスを有効にして起動時に開始します。

```
# systemctl enable ipsec
```

33.5. IPSEC の最適化

IPSec を使用した暗号化の場合のパフォーマンス関連の提案についての詳細は、『[Scaling and Performance Guide](#)』を参照してください。

33.6. トラブルシューティング

2 つのノード間で認証を完了できない場合、すべてのトラフィックが拒否されるため、それらの間で ping を行うことはできません。**clear** ポリシーが適切に設定されていない場合も、クラスター内の別のホストから SSH をホストに対して実行することはできません。

ipsec status コマンドを使用して **clear** および **private** ポリシーが読み込まれていることを確認できます。

第34章 依存関係ツリーのビルド

34.1. 概要

OpenShift Container Platform は、[イメージストリームタグ](#)の更新時を検知するために **BuildConfig** の **イメージ変更トリガー**を使用します。**oc adm build-chain** コマンドを使用し、指定された **イメージストリーム**でイメージを更新することによって影響を受ける **イメージ**を特定する依存関係ツリーをビルドできます。

build-chain ツールは、トリガーする **ビルド**を判別します。このツールはそれらのビルドの出力を解析し、それらが別の **イメージストリームタグ**を更新するかどうかを判別します。更新する場合、このツールは依存関係ツリーを引き続きフォローします。最後に、トップレベルのタグの更新によって影響を受けるイメージストリームタグを指定するグラフを出力します。このツールのデフォルトの出力構文は人間が判読できる形式に設定されます。DOT 形式もサポートされます。

34.2. 使用法

以下の表は、よく使用される **build-chain** の使用方法と一般的な構文について説明しています。

表34.1 よく使用される build-chain 操作

説明	構文
<image-stream> の 最新 タグの依存関係ツリーをビルドします。	<pre>\$ oc adm build-chain <image-stream></pre>
DOT 形式で v2 タグの依存関係ツリーをビルドし、DOT ユーティリティーを使用してこれを可視化します。	<pre>\$ oc adm build-chain <image-stream>:v2 \ -o dot \ dot -T svg -o deps.svg</pre>
test プロジェクトにある指定されたイメージストリームタグについての全プロジェクト間の依存関係ツリーをビルドします。	<pre>\$ oc adm build-chain <image-stream>:v1 \ -n test --all</pre>



注記

graphviz パッケージをインストールして **dot** コマンドを使用する必要がある場合があります。

第35章 バックアップおよび復元

35.1. 概要

OpenShift Container Platform では、クラスターレベルでの **バックアップ** (状態の別個のストレージへの保存) と **復元** (別個のストレージからの状態の再作成) が可能です。さらに、[プロジェクトごとのバックアップ](#)の事前サポートがあります。クラスターインストールの完全な状態には、以下が含まれます。

- 各マスターの etcd データ
- API オブジェクト
- レジストリーストレージ
- ボリュームストレージ

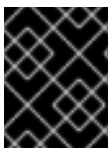
このトピックでは、[永続ストレージ](#)のバックアップおよび復元については扱いません。それらは基礎となるストレージプロバイダーに関連するトピックであるためです。ただし、[アプリケーションデータ](#)の汎用的なバックアップの実行方法についての例を示します。



重要

このトピックでは、アプリケーションおよび OpenShift Container Platform クラスターの一般的なバックアップ方法のみを説明します。ここではカスタム要件は考慮されません。そのため、完全なバックアップおよび復元手順を作成する必要があります。データ損失を防ぐには、必要な予防策を取る必要があります。

etcd バックアップにはストレージボリュームへのすべての参照が含まれることに注意してください。etcd の復元時に、OpenShift Container Platform はノードでの従前の Pod の起動と同じストレージの再割り当てを開始します。これは、クラスターからノードを削除し、新規ノードを追加する場合のプロセスと変わりありません。そのノードに割り当てられているすべてのものは、Pod がどのノードに再スケジュールされる場合でも Pod に再度割り当てられます。



重要

バックアップと復元は保証されません。独自のデータは各自でバックアップする必要があります。

35.2. 前提条件

1. 復元手順には完全な再インストールが必要になるため、初期インストールで使用されるすべてのファイルを保存します。これには、以下が含まれる場合があります。
 - `~/config/openshift/installer.cfg.yml` ([クイックインストール](#)方式を使用する場合)
 - Ansible Playbook およびインベントリーファイル ([通常インストール \(Advanced installation\)](#)方式を使用する場合)
 - `/etc/yum.repos.d/ose.repo` ([非接続インストール](#)方式を使用する場合)
2. インストール後の手順についての手順をバックアップします。一部のインストールには、インストーラーに含まれない手順が含まれる場合があります。これには、OpenShift Container Platform が制御できる範囲を超えているサービスへの変更やモニターエージェントなどの追加

サービスのインストールが含まれる場合があります。高度なインストーラーでサポートされていない追加の設定も、複数の認証プロバイダーの使用時などに影響を受ける可能性があります。

3. 各種のユーティリティーコマンドを提供するパッケージをインストールします。

```
# yum install etcd
```

4. コンテナベースのインストールを使用する場合、代わりに etcd イメージをプルします。

```
# docker pull rhel7/etcd
```

etcd データディレクトリーの場所 (または以下のセクションの **\$ETCD_DATA_DIR**) をメモします。この場所は **etcd** がどのようにデプロイされているかによって異なります。

デプロイメントタイプ	データディレクトリー
オールインワン (all-in-one) クラスター	/var/lib/origin/openshift.local.etcd
外部 etcd (マスターまたは別のホストのいずれかに配置される)	/var/lib/etcd



警告

組み込みの etcd は OpenShift Container Platform 3.7 以降サポートされなくなりました。

35.3. クラスターのバックアップ

35.3.1. マスターのバックアップ

1. 各マスターのすべての証明書およびキーを保存します。

```
# cd /etc/origin/master
# tar cf /tmp/certs-and-keys-$(hostname).tar *.key *.cert
```

35.3.2. etcd のバックアップ

1. **etcd** が複数ホストで実行されている場合、各ホストでこれを停止します。

```
# sudo systemctl stop etcd
```

この手順は必ずしも必要となる訳ではありませんが、これを実行することにより、**etcd** データが完全に同期されます。

2. **etcd** バックアップを作成します。

```
# etcdctl backup \
  --data-dir $ETCD_DATA_DIR \
  --backup-dir $ETCD_DATA_DIR.bak
```



注記

etcd が複数ホストで実行されている場合、各種のインスタンスがそれらのデータを定期的に同期するため、それら内のいずれかのバックアップを作成するだけで十分になります。



注記

コンテナベースのインストールの場合、**docker exec** を使用してコンテナ内で **etcdctl** を実行する必要があります。

3. **db** ファイルを作成したバックアップにコピーします。

```
# cp "$ETCD_DATA_DIR"/member/snap/db
  "$ETCD_DATA_DIR.bak"/member/snap/db
```

35.3.3. レジストリー証明書のバックアップ

1. すべてのマスターおよびノードホストのすべてのレジストリー証明書を保存します。

```
# cd /etc/docker/certs.d/
# tar cf /tmp/docker-registry-certs-$(hostname).tar *
```



注記

1 つ以上の[外部のセキュリティー保護されたレジストリー](#)を使用している場合、イメージのプルまたはプッシュが必要なホストは、Pod を実行するためにレジストリー証明書を信頼する必要があります。

35.4. 単一メンバーの **ETCD** クラスター用のクラスターの復元

クラスターを復元するには、以下を実行します。

1. OpenShift Container Platform を再インストールします。
これは、OpenShift Container Platform が以前にインストールされたのと [同じ方法](#) で実行される必要があります。
2. 必要なすべてのインストール後の手順を実行します。
3. 各マスターで、証明書およびキーを復元します。

```
# cd /etc/origin/master
# tar xvf /tmp/certs-and-keys-$(hostname).tar
```

4. **etcd** バックアップから復元を実行します。

```
# mv $ETCD_DATA_DIR $ETCD_DATA_DIR.orig
# cp -Rp $ETCD_DATA_DIR.bak $ETCD_DATA_DIR
# chcon -R --reference $ETCD_DATA_DIR.orig $ETCD_DATA_DIR
# chown -R etcd:etcd $ETCD_DATA_DIR
```

5. **etcd** の **--force-new-cluster** オプションを使用して単一ノードクラスターを新規に作成します。これは、**/etc/etcd/etcd.conf** の値を使用して実行することも、**systemd** ユニットファイルを一時的に変更し、サービスを通常の方法で起動することによって実行することもできます。

これを実行するには、**/usr/lib/systemd/system/etcd.service** ファイルを編集し、**--force-new-cluster** を追加します。

```
# sed -i '/ExecStart/s/"$/ --force-new-cluster"/'
/usr/lib/systemd/system/etcd.service
# systemctl show etcd.service --property ExecStart --no-pager

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd --force-new-cluster"
```

次に、**etcd** サービスを再起動します。

```
# systemctl daemon-reload
# systemctl start etcd
```

6. **etcd** サービスが適切に起動したことを確認してから、**/usr/lib/systemd/system/etcd.service** ファイルを再編集し、**--force-new-cluster** オプションを削除します。

```
# sed -i '/ExecStart/s/ --force-new-cluster//'
/usr/lib/systemd/system/etcd.service
# systemctl show etcd.service --property ExecStart --no-pager

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd"
```

7. **etcd** サービスを再起動してから **etcd** クラスターが適切に実行されていることを確認し、OpenShift Container Platform の設定を表示します。

```
# systemctl daemon-reload
# systemctl restart etcd
```

35.5. 複数メンバーで構成される **ETCD** クラスター用のクラスターの復元

マスターホストに **etcd** をデプロイする必要がある場合、新規ホストを作成する必要はありません。マスターホストから専用 **etcd** をデプロイする必要がある場合は、新規ホストを作成する必要があります。

初期の **etcd** メンバーとするシステムを選択してから、その **etcd** バックアップおよび設定を復元します。

1. **etcd** ホストで以下を実行します。

```
# ETCD_DIR=/var/lib/etcd/
# mv $ETCD_DIR /var/lib/etcd.orig
# cp -Rp var/lib/etcd.orig/openshift-backup-pre-upgrade/ $ETCD_DIR
```

```
# chcon -R --reference /var/lib/etcd.orig/ $ETCD_DIR
# chown -R etcd:etcd $ETCD_DIR
```

2. バックアップまたは **.rpmsave** から **/etc/etcd/etcd.conf** ファイルを復元します。
3. 使用される環境に応じて、[コンテナ化された etcd デプロイメント](#) または [コンテナ化されていない etcd デプロイメント](#) についての説明に従ってください。

35.5.1. コンテナ化された **etcd** デプロイメント

1. **etcd** の **--force-new-cluster** オプションを使用して単一ノードクラスターを新規に作成します。これは、**/etc/etcd/etcd.conf** の値を使用して長く複雑なコマンドを使用して実行することも、**systemd** ユニットファイルを一時的に変更し、通常の方法でサービスを起動して実行することもできます。

これを実行するには、**/etc/systemd/system/etcd_container.service** ファイルを編集し、**--force-new-cluster** を追加します。

```
# sed -i '/ExecStart=/s$/ --force-new-cluster/'
/etc/systemd/system/etcd_container.service

ExecStart=/usr/bin/docker run --name etcd --rm -v \
/var/lib/etcd:/var/lib/etcd:z -v /etc/etcd:/etc/etcd:ro --env-
file=/etc/etcd/etcd.conf \
--net=host --entrypoint=/usr/bin/etcd rhel7/etcd:3.1.9 --force-new-
cluster
```

次に、**etcd** サービスを再起動します。

```
# systemctl daemon-reload
# systemctl start etcd_container
```

2. **etcd** サービスが適切に起動したことを確認してから、**/etc/systemd/system/etcd_container.service** ファイルを再編集し、**--force-new-cluster** オプションを削除します。

```
# sed -i '/ExecStart=/s/ --force-new-cluster//'
/etc/systemd/system/etcd_container.service

ExecStart=/usr/bin/docker run --name etcd --rm -v
/var/lib/etcd:/var/lib/etcd:z -v \
/etc/etcd:/etc/etcd:ro --env-file=/etc/etcd/etcd.conf --net=host \
--entrypoint=/usr/bin/etcd rhel7/etcd:3.1.9
```

3. **etcd** サービスを再起動してから **etcd** クラスターが適切に実行されていることを確認し、OpenShift Container Platform の設定を表示します。

```
# systemctl daemon-reload
# systemctl restart etcd_container
# etcdctl --cert-file=/etc/etcd/peer.crt \
--key-file=/etc/etcd/peer.key \
--ca-file=/etc/etcd/ca.crt \
--peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
ls /
```

4. クラスターに追加する必要がある他の etcd メンバーがある場合、[etcd メンバーの追加](#)に進みます。または、単一ノードの外部 etcd が必要な場合、[OpenShift Container Platform サービスの再オンライン化](#)に進みます。

35.5.2. コンテナ化されていない etcd デプロイメント

1. etcd の **--force-new-cluster** オプションを使用して単一ノードクラスターを新規に作成します。これは、`/etc/etcd/etcd.conf` の値を使用して長く複雑なコマンドを使用して実行することも、**systemd** ユニットファイルを一時的に変更し、通常の方法でサービスを起動して実行することもできます。
これを実行するには、`/usr/lib/systemd/system/etcd.service` ファイルを編集し、**--force-new-cluster** を追加します。

```
# sed -i '/ExecStart/s/"$/ --force-new-cluster"/'
/usr/lib/systemd/system/etcd.service
# systemctl show etcd.service --property ExecStart --no-pager

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd --force-new-cluster"
```

次に **etcd** サービスを再起動します。

```
# systemctl daemon-reload
# systemctl start etcd
```

2. **etcd** サービスが適切に起動したことを確認してから、`/usr/lib/systemd/system/etcd.service` ファイルを再編集し、**--force-new-cluster** オプションを削除します。

```
# sed -i '/ExecStart/s/ --force-new-cluster//'
/usr/lib/systemd/system/etcd.service
# systemctl show etcd.service --property ExecStart --no-pager

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd"
```

3. **etcd** サービスを再起動してから etcd クラスターが適切に実行されていることを確認し、OpenShift Container Platform の設定を表示します。

```
# systemctl daemon-reload
# systemctl restart etcd
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
  ls /
```

4. クラスターに追加する必要がある他の etcd メンバーがある場合、[etcd メンバーの追加](#)に進みます。または、単一ノードの外部 etcd が必要な場合、[OpenShift Container Platform サービスの再オンライン化](#)に進みます。

35.5.3. etcd メンバーの追加

etcd メンバーをクラスターに追加するには、まず、最初のメンバーの **peerURLs** 値でデフォルトの **localhost** ピアを調整する必要があります。

1. **member list** コマンドを使用して最初のメンバーのメンバー ID を取得します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://17
2.18.0.75:2379" \
  member list
```

2. 直前の手順で取得されたメンバー ID を渡して、**etcdctl member update** コマンドを使用して **peerURLs** の値を更新します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
  --
peers="https://172.18.1.18:2379,https://172.18.9.202:2379,https://17
2.18.0.75:2379" \
  member update 511b7fb6cc0001 https://172.18.1.18:2380
```

または、**curl** を使用できます。

```
# curl --cacert /etc/etcd/ca.crt \
  --cert /etc/etcd/peer.crt \
  --key /etc/etcd/peer.key \
  https://172.18.1.18:2379/v2/members/511b7fb6cc0001 \
  -XPUT -H "Content-Type: application/json" \
  -d '{"peerURLs":["https://172.18.1.18:2380"]}'
```

3. **member list** コマンドを再実行し、ピア URL に **localhost** が含まれなくなるようにします。
4. ここで、メンバーは 1 度に 1 つずつクラスターに追加します。



警告

各メンバーは 1 度に 1 つずつ完全に追加され、オンライン化される必要があります。各メンバーをクラスターに追加する際、**peerURLs** 一覧はその時点で正しくなければならず、各メンバーが追加されるたびに拡張できるようにします。**etcdctl member add** コマンドは、以下の説明にあるように各メンバーの追加時に **etcd.conf** ファイルに設定される必要のある値を出力します。

- a. それぞれのメンバーについて、そのシステムの **etcd.conf** ファイルにある値を使用してクラスターに追加します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key \
  --ca-file=/etc/etcd/ca.crt \
```

```
--peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
member add 10.3.9.222 https://172.16.4.27:2380
```

Added member named 10.3.9.222 with ID 4e1db163a21d7651 to cluster

```
ETCD_NAME="10.3.9.222"
```

```
ETCD_INITIAL_CLUSTER="10.3.9.221=https://172.16.4.18:2380,10.3.9.222=https://172.16.4.27:2380"
```

```
ETCD_INITIAL_CLUSTER_STATE="existing"
```

- b. 上記の **etcdctl member add** コマンドで提供される環境変数を使用し、メンバーシステム自体で **/etc/etcd/etcd.conf** ファイルを編集し、これらの設定が一致することを確認します。

- c. ここで、新規メンバーで etcd を起動します。

```
# rm -rf /var/lib/etcd/member
# systemctl enable etcd
# systemctl start etcd
```

- d. サービスが正常に起動し、etcd クラスターが正常な状態にあることを確認します。

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
--key-file=/etc/etcd/peer.key \
--ca-file=/etc/etcd/ca.crt \
--peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
member list
```

```
51251b34b80001: name=10.3.9.221 peerURLs=https://172.16.4.18:2380
clientURLs=https://172.16.4.18:2379
d266df286a41a8a4: name=10.3.9.222
peerURLs=https://172.16.4.27:2380
clientURLs=https://172.16.4.27:2379
```

```
# etcdctl --cert-file=/etc/etcd/peer.crt \
--key-file=/etc/etcd/peer.key \
--ca-file=/etc/etcd/ca.crt \
--peers="https://172.16.4.18:2379,https://172.16.4.27:2379" \
cluster-health
```

```
cluster is healthy
member 51251b34b80001 is healthy
member d266df286a41a8a4 is healthy
```

- e. クラスターに追加する次のメンバーについてもこのプロセスを繰り返します。

5. すべての etcd メンバーの追加後に、[OpenShift Container Platform サービスの再オンライン化](#)に進みます。

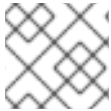
35.6. 新規 ETCD ホストの追加

etcd メンバーが失敗する際にクォーラム(定足数)の etcd クラスターメンバーが実行されている場合、存続しているメンバーをそのまま使用してダウンタイムを発生させずに etcd メンバーを追加することができます。

提案されるクラスターサイズ

奇数の etcd ホストを含むクラスターの場合、フォールトトレランスに対応できます。奇数の etcd ホストがあることで、クォーラム(定足数)に必要な数が変わることはありませんが、障害発生時の耐性が高まります。たとえば、クラスターサイズが3メンバーである場合、クォーラム(定足数)は2で、1メンバーは障害耐性用になります。これにより、クラスターはメンバーの2つが正常である限り、機能し続けます。

3つの etcd ホストで構成される実稼働クラスターの使用が推奨されます。



注記

以下は、etcd ホストの **/etc/etcd** 設定のバックアップがあることを前提としています。

1. 新規 etcd メンバーが OpenShift Container Platform ノードでもある場合は、「[必要なホスト数のクラスターへの追加](#)」を参照してください。この手順の残りの部分では、1つのホストのみを追加していることを前提としていますが、複数のホストを追加する場合には、それぞれのホストですべての手順を実行してください。

2. 存続するノードで etcd および iptables をアップグレードします。

```
# yum update etcd iptables-services
```

バージョン **etcd-2.3.7-4.el7.x86_64** 以上がインストールされており、同じバージョンが各ホストにインストールされていることを確認します。

3. 新規ホストで etcd および iptables をインストールします。

```
# yum install etcd iptables-services
```

バージョン **etcd-2.3.7-4.el7.x86_64** 以上がインストールされており、同じバージョンが新規ホストにインストールされていることを確認します。

4. クラスター設定の変更を行う前に、存続するホストで [etcd データストアのバックアップ](#)を実行します。
5. 失敗した etcd メンバーを置き換える場合、新規メンバーを追加する **前** に、失敗したメンバーを削除します。

```
# etcdctl -C https://<surviving host IP>:2379 \
  --ca-file=/etc/etcd/ca.crt \
  --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key cluster-health

# etcdctl -C https://<surviving host IP>:2379 \
  --ca-file=/etc/etcd/ca.crt \
  --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key member remove <failed member
identifier>
```

失敗した etcd メンバーで etcd サービスを停止します。

```
# systemctl stop etcd
```

6. 新規ホストで、適切な iptables ルールを追加します。

```
# systemctl enable iptables.service --now
# iptables -N OS_FIREWALL_ALLOW
# iptables -t filter -I INPUT -j OS_FIREWALL_ALLOW
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state \
  --state NEW -m tcp --dport 2379 -j ACCEPT
# iptables -A OS_FIREWALL_ALLOW -p tcp -m state \
  --state NEW -m tcp --dport 2380 -j ACCEPT
# iptables-save > /etc/sysconfig/iptables
```

7. 新規ホストの必要な証明書を生成します。存続する etcd ホストで以下を実行します。

- a. `/etc/etcd/ca/` ディレクトリーのバックアップを作成します。

- b. 証明書の変数および作業ディレクトリーを設定し、作成されていない場合には **PREFIX** ディレクトリーを作成します。

```
# cd /etc/etcd
# export NEW_ETCD="<NEW_HOST_NAME>"

# export CN=$NEW_ETCD
# export SAN="IP:<NEW_HOST_IP>"
# export PREFIX="./generated_certs/etcd-$CN/"
```

- c. `$PREFIX` ディレクトリーを作成します。

```
$ mkdir -p $PREFIX
```

- d. **server.csr** および **server.crt** 証明書を作成します。

```
# openssl req -new -keyout ${PREFIX}server.key \
  -config ca/openssl.cnf \
  -out ${PREFIX}server.csr \
  -reqexts etcd_v3_req -batch -nodes \
  -subj /CN=$CN

# openssl ca -name etcd_ca -config ca/openssl.cnf \
  -out ${PREFIX}server.crt \
  -in ${PREFIX}server.csr \
  -extensions etcd_v3_ca_server -batch
```

- e. **peer.csr** および **peer.crt** 証明書を作成します。

```
# openssl req -new -keyout ${PREFIX}peer.key \
  -config ca/openssl.cnf \
  -out ${PREFIX}peer.csr \
  -reqexts etcd_v3_req -batch -nodes \
  -subj /CN=$CN

# openssl ca -name etcd_ca -config ca/openssl.cnf \
  -out ${PREFIX}peer.crt \
  -in ${PREFIX}peer.csr \
  -extensions etcd_v3_ca_peer -batch
```

- f. **etcd.conf** および **ca.crt** ファイルをコピーし、ディレクトリーの内容をアーカイブします。

```
# cp etcd.conf ${PREFIX}
# cp ca.crt ${PREFIX}
# tar -czvf ${PREFIX}${CN}.tgz -C ${PREFIX} .
```

- g. ファイルを新規の etcd ホストに転送します。

```
# scp ${PREFIX}${CN}.tgz $CN:/etc/etcd/
```

8. 存続している etcd ホスト上で、新規ホストをクラスターに追加します。

- a. 新規ホストをクラスターに追加します。

```
# export ETCD_CA_HOST="<SURVIVING_ETCD_HOSTNAME>"
# export NEW_ETCD="<NEW_ETCD_HOSTNAME>"
# export NEW_ETCD_IP="<NEW_HOST_IP>"

# etcdctl -C https://${ETCD_CA_HOST}:2379 \
  --ca-file=/etc/etcd/ca.crt \
  --cert-file=/etc/etcd/peer.crt \
  --key-file=/etc/etcd/peer.key member add ${NEW_ETCD}
https://${NEW_ETCD_IP}:2380

ETCD_NAME="<NEW_ETCD_HOSTNAME>"
ETCD_INITIAL_CLUSTER="
<NEW_ETCD_HOSTNAME>=https://<NEW_HOST_IP>:2380,
<SURVIVING_ETCD_HOST>=https://<SURVIVING_HOST_IP>:2380
ETCD_INITIAL_CLUSTER_STATE="existing"
```

etcdctl member add 出力の 3 つの環境変数をコピーします。これらは後で使用されます。

- b. 新規ホストで、コピーされた設定データを抽出し、パーミッションを設定します。

```
# tar -xf /etc/etcd/<NEW_ETCD_HOSTNAME>.tgz -C /etc/etcd/ --
  overwrite
# chown -R etcd:etcd /etc/etcd/*
```

- c. 新規ホストで、etcd データを削除します。

```
# rm -rf /var/lib/etcd/member
# chown -R etcd:etcd /var/lib/etcd
```

9. 新規 etcd ホストの **etcd.conf** ファイルで、以下を実行します。

- a. 以下を直前の手順で生成された値に置き換えます。

- ETCD_NAME
- ETCD_INITIAL_CLUSTER
- ETCD_INITIAL_CLUSTER_STATE
IP アドレスを以下の「NEW_ETCD」値に置き換えます。

- ETCD_LISTEN_PEER_URLS
- ETCD_LISTEN_CLIENT_URLS
- ETCD_INITIAL_ADVERTISE_PEER_URLS
- ETCD_ADVERTISE_CLIENT_URLS

失敗したメンバーを置き換えるには、失敗したホストを etcd 設定から削除する必要があります。

10. 新規ホストで etcd を起動します。

```
# systemctl enable etcd --now
```

11. 新規メンバーが正常に追加されたことを確認するには、以下を実行します。

```
etcdctl -C https://${ETCD_CA_HOST}:2379 --ca-file=/etc/etcd/ca.crt \
--cert-file=/etc/etcd/peer.crt \
--key-file=/etc/etcd/peer.key cluster-health
```

12. すべてのマスターで、新規 etcd ホストを参照するようマスター設定を更新します。

- クラスタ内のすべてのマスターで、**/etc/origin/master/master-config.yaml** を編集します。
- etcdClientInfo** セクションを見つけます。
- 新規 etcd ホストを **urls** 一覧に追加します。
- 失敗した etcd ホストが置き換えられている場合、これを一覧から削除します。
- マスター API サービスを再起動します。
各マスターで以下を実行します。

```
# systemctl restart atomic-openshift-master-api atomic-openshift-
master-controllers
```

etcd メンバーを追加する手順が完了します。

35.7. OPENSIFT CONTAINER PLATFORM サービスの再オンライン化

それぞれの OpenShift Container Platform マスターで、バックアップからマスターおよびノード設定を復元し、すべての関連するサービスを有効にしてから再起動します。

```
# cp /etc/sysconfig/atomic-openshift-master-api.rpmsave
/etc/sysconfig/atomic-openshift-master-api
# cp /etc/sysconfig/atomic-openshift-master-controllers.rpmsave
/etc/sysconfig/atomic-openshift-master-controllers
# cp /etc/origin/master/master-config.yaml.<timestamp>
/etc/origin/master/master-config.yaml
# cp /etc/origin/node/node-config.yaml.<timestamp> /etc/origin/node/node-
config.yaml
# cp /etc/origin/master/scheduler.json.<timestamp>
/etc/origin/master/scheduler.json
# systemctl enable atomic-openshift-master-api
```

```
# systemctl enable atomic-openshift-master-controllers
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-master-api
# systemctl start atomic-openshift-master-controllers
# systemctl start atomic-openshift-node
```

それぞれの OpenShift Container Platform ノードで、バックアップから **node-config.yaml** ファイルを復元し、**atomic-openshift-node** サービスを有効にしてから再起動します。

```
# cp /etc/origin/node/node-config.yaml.<timestamp> /etc/origin/node/node-
config.yaml
# systemctl enable atomic-openshift-node
# systemctl start atomic-openshift-node
```

OpenShift Container Platform クラスターが再びオンライン化されます。

35.8. プロジェクトのバックアップ

OpenShift Container Platform の今後のリリースでは、その特長の 1 つであるプロジェクトごとのバックアップと復元のサポートが提供されます。

現時点では、プロジェクトレベルで API オブジェクトをバックアップするには、保存する各オブジェクトについて **oc export** を使用します。たとえば、YAML 形式でデプロイメント設定 **frontend** を保存するには、以下を実行します。

```
$ oc export dc frontend -o yaml > dc-frontend.yaml
```

プロジェクトのすべて (namespace およびプロジェクトなどのクラスターオブジェクトの例外を除く) をバックアップするには、以下を実行します。

```
$ oc export all -o yaml > project.yaml
```

35.8.1. ロールバインディング

カスタムポリシーの [ロールバインディング](#) はプロジェクトで使用されることがよくあります。たとえば、プロジェクトの管理者は別のユーザーに、プロジェクトの特定のロールを付与し、そのユーザーにプロジェクトアクセスを付与することができます。

これらのロールバインディングはエクスポートできます。

```
$ oc get rolebindings -o yaml --export=true > rolebindings.yaml
```

35.8.2. サービスアカウント

カスタムサービスアカウントがプロジェクトに作成されている場合、これらをエクスポートする必要があります。

```
$ oc get serviceaccount -o yaml --export=true > serviceaccount.yaml
```

35.8.3. シークレット

ソースコントロール管理シークレット (SSH パブリックキー、ユーザー名/パスワード) などのカスタムシークレットは、これらが使用される場合はエクスポートする必要があります。

```
$ oc get secret -o yaml --export=true > secret.yaml
```

35.8.4. Persistent Volume Claim (永続ボリューム要求、PVC)

プロジェクト内のアプリケーションが Persistent Volume Claim (永続ボリューム要求、PVC) によって永続ボリュームを使用する場合、これらをバックアップする必要があります。

```
$ oc get pvc -o yaml --export=true > pvc.yaml
```

35.9. プロジェクトの復元

プロジェクトを復元するには、プロジェクトを再作成し、バックアップ時にエクスポートされたすべてのオブジェクトを再作成します。

```
$ oc new-project myproject
$ oc create -f project.yaml
$ oc create -f secret.yaml
$ oc create -f serviceaccount.yaml
$ oc create -f pvc.yaml
$ oc create -f rolebindings.yaml
```



注記

一部のリソースは作成できない可能性があります (例: Pod およびデフォルトのサービスアカウント)。

35.10. アプリケーションデータのバックアップ

多くの場合、**rsync** がコンテナイメージ内にインストールされていることを前提とすると、アプリケーションデータは **oc rsync** コマンドを使用してバックアップできます。Red Hat **rhel7** ベースイメージには **rsync** が含まれます。したがって、**rhel7** をベースとするすべてのイメージにはこれが含まれることになります。「[Troubleshooting and Debugging CLI Operations - rsync](#)」を参照してください。



警告

これはアプリケーションデータの **汎用的な** バックアップであり、データベースシステムの特殊なエクスポート/インポート手順などのアプリケーション固有のバックアップ手順は考慮に入れられません。

永続ボリュームのタイプによっては、他のバックアップ手段もある場合があります (Cinder、NFS、Gluster など)。

バックアップのパスも **アプリケーションに固有** のものです。**deploymentconfig** でボリュームの **mountPath** を参照してバックアップするパスを判別することができます。

Jenkins デプロイメントのアプリケーションデータのバックアップ例

1. アプリケーションデータ **mountPath** を **deploymentconfig** から取得します。

```
$ oc get dc/jenkins -o jsonpath='{ .spec.template.spec.containers[?
(@.name=="jenkins")].volumeMounts[?(@.name=="jenkins-
data")].mountPath }'
/var/lib/jenkins
```

2. 現在実行中の Pod の名前を取得します。

```
$ oc get pod --selector=deploymentconfig=jenkins -o jsonpath='{
.metadata.name }'
jenkins-1-37nux
```

3. **oc rsync** コマンドを使用してアプリケーションデータをコピーします。

```
$ oc rsync jenkins-1-37nux:/var/lib/jenkins /tmp/
```



注記

このタイプのアプリケーションデータバックアップは、アプリケーション Pod が現時点で実行中の場合にのみ実行できます。

35.11. アプリケーションデータの復元

アプリケーションデータの復元プロセスは **oc rsync** ツールを使用した [アプリケーションバックアップ手順](#) に似ています。同じ制限が適用され、アプリケーションデータの復元プロセスには永続ボリュームが必要です。

Jenkins デプロイメントのアプリケーションデータの復元例

1. バックアップを確認します。

```
$ ls -la /tmp/jenkins-backup/
total 8
drwxrwxr-x.  3 user      user    20 Sep  6 11:14 .
drwxrwxrwt. 17 root      root    4096 Sep  6 11:16 ..
drwxrwsrwx. 12 user      user    4096 Sep  6 11:14 jenkins
```

2. **oc rsync** ツールを使用してデータを実行中の Pod にコピーします。

```
$ oc rsync /tmp/jenkins-backup/jenkins jenkins-1-37nux:/var/lib
```



注記

アプリケーションによっては、アプリケーションを再起動する必要があります。

3. 新規データでアプリケーションを再起動します (オプション)。

```
$ oc delete pod jenkins-1-37nux
```

または、デプロイメントを 0 にスケールダウンしてから再びスケールアップします。

```
$ oc scale --replicas=0 dc/jenkins  
$ oc scale --replicas=1 dc/jenkins
```


第36章 OPENSIFT SDN のトラブルシューティング

36.1. 概要

[SDN のドキュメント](#)で説明されているように、あるコンテナから別のコンテナへのトラフィックを適切に渡すために作成されるインターフェースには複数のレイヤーがあります。接続の問題をデバッグするには、スタックの複数のレイヤーをテストして問題の原因を判別する必要があります。本書は複数のレイヤーを調べて問題を特定し、解決するのに役立ちます。

問題の原因の一部は OpenShift Container Platform が複数の方法で設定でき、ネットワークが複数の異なる場所で正しく設定されない可能性がある点にあります。本書では、いくつかのシナリオを使用しますが、これらのシナリオは大半のケースに対応していることが予想されます。実際に生じている問題がこれらのシナリオで扱われていない場合には、導入されている各種のツールおよび概念を使用してデバッグ作業を行うことができます。

36.2. 用語

クラスター

クラスター内の一連のマシンです。**例:** マスターおよびノード。

マスター

OpenShift Container Platform クラスターのコントローラーです。マスターはクラスター内のノードではない場合があります、そのため Pod への IP 接続がない場合があることに注意してください。

ノード

Pod をホストできる OpenShift Container Platform を実行するクラスター内のホストです。

Pod

OpenShift Container Platform によって管理される、ノード上で実行されるコンテナのグループです。

サービス

1 つ以上の Pod でサポートされる、統一ネットワークインターフェースを表す抽象化です。

ルーター

複数の URL とパスを OpenShift Container Platform サービスにマップし、外部トラフィックがクラスターに到達できるようにする web プロキシです。

ノードアドレス

ノードの IP アドレスです。これはノードが割り当てられるネットワークの所有者によって割り当てられ、管理されます。クラスター内の任意のノード (マスターおよびクライアント) からアクセスできる必要があります。

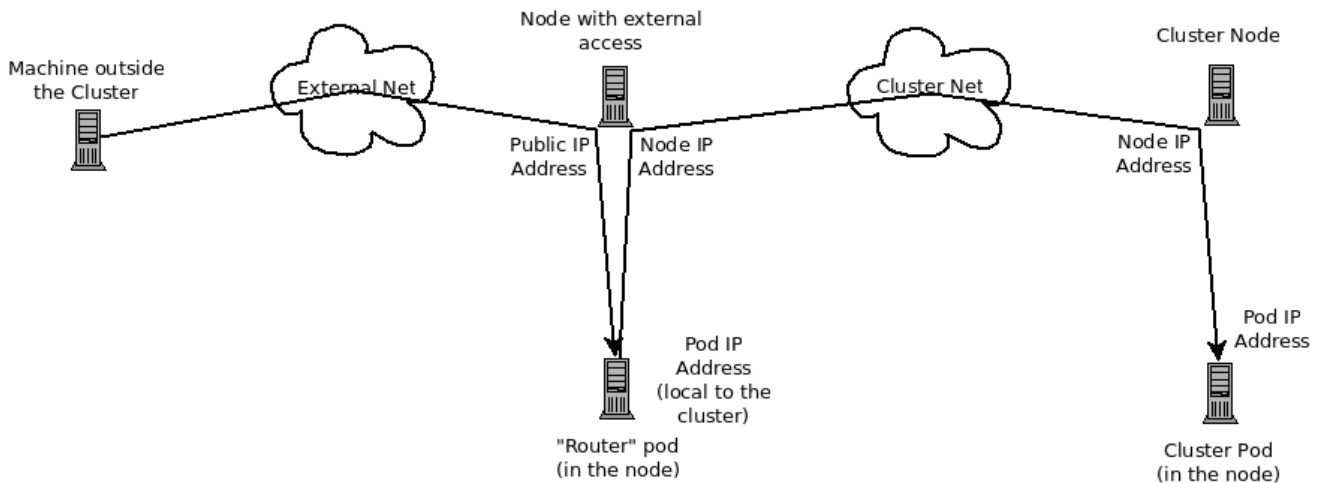
Pod アドレス

Pod の IP アドレスです。これらは OpenShift Container Platform によって割り当てられ、管理されます。デフォルトで、これらは 10.128.0.0/14 ネットワーク (または古いバージョンでは 10.1.0.0/16) から割り当てられます。クライアントノードからのみアクセスできます。

サービスアドレス

サービスを表す IP アドレスで、内部で Pod アドレスにマップされます。これらは OpenShift Container Platform によって割り当てられ、管理されます。デフォルトで、これらは 172.30.0.0/16 ネットワークから割り当てられます。クライアントノードからのみアクセスできます。

以下の図は、外部アクセスに関係するすべての構成部分を示しています。



36.3. HTTP サービスへの外部アクセスのデバッグ

クラスター外のマシンを使用している場合で、クラスターで提供されるリソースにアクセスしようとしている場合、パブリック IP アドレスでリッスンし、クラスター内のトラフィックを「ルーティング」する Pod でプロセスが実行されている必要があります。この場合、[OpenShift Container Platform ルーター](#)は、HTTP、HTTPS (SNI を使用)、WebSockets または TLS (SNI を使用) について使用できます。

クラスター外より HTTP サービスにアクセスできないことを想定し、障害が発生しているマシンのコマンドラインを使って問題を再現します。以下を実行します。

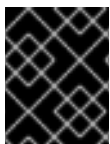
```
curl -kv http://foo.example.com:8000/bar      # But replace the argument
with your URL
```

成功する場合は、正しい場所からバグを再現しているかどうかを確認します。サービスに機能する Pod と機能しない Pod が含まれる可能性もあります。したがって、「[ルーターのデバッグ](#)」セクションを参照してください。

失敗した場合は、IP アドレスに対して DNS 名を解決します (ないことを想定します)。

```
dig +short foo.example.com                  # But replace the hostname
with yours
```

IP アドレスが返されない場合は、DNS をトラブルシューティングする必要がありますが、これについては本書では扱いません。



重要

返される IP アドレスがルーターを実行するルーターであることを確認します。そうでない場合は、DNS を修正します。

次に **ping -c address** および **tracpath address** を使用して、ルーターホストに到達できることを確認します。それらが ICMP パケットに応答しない場合もあり、この場合はそれらのテストは失敗しますが、ルーターマシンにはアクセスできる場合があります。この場合、コマンドを使ってルーターのポートに直接アクセスしてみます。

```
telnet 1.2.3.4 8000
```

以下が表示される場合があります。

■

```
Trying 1.2.3.4...
Connected to 1.2.3.4.
Escape character is '^['.
```

この場合、IP アドレスのポートでリッスンしているものがあることを示しています。**ctrl-]** を押してから **enter** キーを押し、**close** を入力して telnet を終了します。「[ルーターのデバッグ](#)」セクションに移行してルーターの他のものを確認します。

または、以下が表示される可能性があります。

```
Trying 1.2.3.4...
telnet: connect to address 1.2.3.4: Connection refused
```

これは、ルーターがそのポートでリッスンしていないことを示します。ルーターの設定方法における追加のポイントについては、「[ルーターのデバッグ](#)」セクションを参照してください。

または、以下が表示される場合があります。

```
Trying 1.2.3.4...
telnet: connect to address 1.2.3.4: Connection timed out
```

これは、IP アドレス上のいずれとも通信できないことを示します。ルーティング、ファイアウォールを確認し、IP アドレスでリッスンしているルーターがあることを確認します。ルーターをデバッグするには、「[ルーターのデバッグ](#)」セクションを参照してください。IP ルーティングおよびファイアウォールの問題については、本書では扱いません。

36.4. ルーターのデバッグ

IP アドレスを使用し、そのマシンに対して **ssh** を実行してルーターソフトウェアがそのマシン上で実行されており、正しく設定されていることを確認する必要があります。ここで **ssh** を実行し、管理者の OpenShift Container Platform 認証情報を取得します。

注記

管理者認証情報を利用できるが [デフォルトシステムユーザー](#) **system:admin** としてログインしていない場合、認証情報が [CLI 設定ファイル](#)にある限りはこのユーザーとしていつでもログインし直すことができます。以下のコマンドはログインを実行し、[デフォルトプロジェクト](#)への切り替えを実行します。

```
$ oc login -u system:admin -n default
```

ルーターが実行されていることを確認します。

```
# oc get endpoints --namespace=default --selector=router
NAMESPACE   NAME           ENDPOINTS
default     router         10.128.0.4:80
```

このコマンドが失敗する場合、OpenShift Container Platform 設定は破損しています。この設定の修正については、本書では扱われません。

1 つ以上のルーターエンドポイントが一覧表示されますが、エンドポイント IP アドレスはクラスター内の Pod アドレスの 1 つであるため、それらが指定の外部 IP アドレスでマシン上で実行されているかどうかを識別することはできません。ルーターホスト IP アドレスの一覧を取得するには、以下を実行し

ます。

```
# oc get pods --all-namespaces --selector=router --template='{{range
.items}}HostIP: {{.status.hostIP}} PodIP: {{.status.podIP}}{{end}}
{{"\n"}}'
```

HostIP: 192.168.122.202 PodIP: 10.128.0.4

外部アドレスに対応するホスト IP が表示されるはずです。表示されない場合は、[ルーターのドキュメント](#)を参照して、適切なノードで実行されるようにルーター Podを設定するか (アフィニティを適切に設定する)、またはルーターが実行されている IP アドレスに一致するよう DNS を更新します。

(本書の) この時点では、ノードでルーター Pod を実行しても HTTP 要求を機能させることはできません。まず、ルーターが外部 URL を正しいサービスにマップしていること、またそれが機能している場合は、そのサービスの詳細を調べてすべてのエンドポイントがアクセス可能であることを確認する必要があります。

OpenShift Container Platform が認識するすべてのルートを一覧表示します。

```
# oc get route --all-namespaces
```

NAME	HOST/PORT	PATH	SERVICE	LABELS
TLS TERMINATION				
route-unsecured	www.example.com	/test	service-name	

URL のホスト名およびパスが返されるルートの一覧のいずれにも一致しない場合はルートを追加する必要があります。[ルーターのドキュメント](#)を参照してください。

ルートが存在する場合、エンドポイントへのアクセスをデバッグする必要があります。これはサービスに関する問題をデバッグしている場合と同様のプロセスです。そのため、次の「[サービスのデバッグ](#)」セクションに進んでください。

36.5. サービスのデバッグ

クラスター内からサービスと通信できない場合 (サービスが直接通信できないか、またはルーターを使用してクラスターに入るまですべてが正常に機能している場合)、サービスに関連付けられているエンドポイントを判別し、それらをデバッグする必要があります。

最初にサービスを取得します。

```
# oc get services --all-namespaces
```

NAMESPACE	NAME	LABELS	SELECTOR	IP(S)	PORT(S)
default	docker-registry	docker-registry=default			
	docker-registry=default			172.30.243.225	5000/TCP
default	kubernetes	component=apiserver,provider=kubernetes			
	<none>			172.30.0.1	443/TCP
default	router	router=router			
	router=router			172.30.213.8	80/TCP

サービスが一覧に表示されます。表示されない場合は、[サービス](#)を定義する必要があります。

サービス出力に一覧表示される IP アドレスは Kubernetes サービス IP アドレスであり、これは Kubernetes がサービスをサポートする Pod のいずれかにマップするものです。この IP アドレスと通信できるはずですが、通信できたとしても、すべての Pod にアクセスできる訳ではありません。また、通信できない場合もすべての Pod がアクセスできない訳ではありません。これは kubeproxy が接続している 1 つの IP アドレスのステータスのみを示しています。

サービスをテストします。ノードのいずれかより以下を実行します。

```
curl -kv http://172.30.243.225:5000/bar # Replace the
argument with your service IP address and port
```

次にサービスをサポートしている Pod を見つけます (**docker-registry** を破損したサービスの名前に置き換えます)。

```
# oc get endpoints --selector=docker-registry
NAME                      ENDPOINTS
docker-registry          10.128.2.2:5000
```

ここではエンドポイントは 1 つだけであることを確認できます。そのため、サービステストが成功し、ルーターテストに成功した場合には、極めて稀なことが生じている可能性があります。ただし、複数のエンドポイントがあるか、またはサービステストが失敗した場合には、**それぞれの** エンドポイントについて以下を試行します。機能していないエンドポイントを特定できたら、次のセクションに進みます。

最初に、それぞれのエンドポイントをテストします (適切なエンドポイント IP、ポートおよびパスを持つように URL を変更します)。

```
curl -kv http://10.128.2.2:5000/bar
```

これが機能する場合は、次のエンドポイントをテストします。失敗した場合はその情報をメモしておきます。次のセクションでその原因を判別します。

すべてが失敗した場合は、ローカルノードが機能していない可能性があります。その場合は、「[ローカルネットワークのデバッグ](#)」セクションに移行してください。

すべてが機能する場合は、「[Kubernetes のデバッグ](#)」セクションに移行してサービス IP アドレスが機能しない理由を判別します。

36.6. ノード間通信のデバッグ

機能していないエンドポイントの一覧を使用して、ノードに対する接続をテストする必要があります。

1. すべてのノードに予想される IP アドレスがあることを確認します。

```
# oc get hostsubnet
NAME                      HOST                      HOST IP
SUBNET
rh71-os1.example.com     rh71-os1.example.com     192.168.122.46
10.1.1.0/24
rh71-os2.example.com     rh71-os2.example.com     192.168.122.18
10.1.2.0/24
rh71-os3.example.com     rh71-os3.example.com     192.168.122.202
10.1.0.0/24
```

DHCP を使用している場合はそれらが変更されている可能性があります。ホスト名、IP アドレス、およびサブネットが予想される内容に一致していることを確認します。ノードの詳細が変更されている場合は、**oc edit hostsubnet** を使用してエントリーを訂正します。

2. ノードアドレスおよびホスト名が正しいことを確認した後に、エンドポイント IP およびノード IP を一覧表示します。

```
# oc get pods --selector=docker-registry \
  --template='{range .items}}HostIP: {{.status.hostIP}}    PodIP:
{{.status.podIP}}{{end}}{{"\n"}}'
```

HostIP: 192.168.122.202 PodIP: 10.128.0.4

3. 事前にメモしたエンドポイント IP アドレスを見つけ、これを **PodIP** エントリーを検索し、対応する **HostIP** アドレスを見つけます。次に、**HostIP** からのアドレスを使用してノードホストレベルで接続をテストします。

- **ping -c 3 <IP_address>**: 応答がないことは、中間ルーターが ICMP トラフィックを消費している可能性があることを意味しています。
- **tracpath <IP_address>**: ICMP パケットがすべてのホップによって返される場合、ターゲットにつながる IP ルートを表示します。
tracpath と **ping** の両方が失敗する場合、ローカルまたは仮想ネットワークの接続の問題を探します。

4. ローカルネットワークの場合は、以下を確認します。

- 追加設定なしの状態のパケットのターゲットアドレスへのルートを確認します。

```
# ip route get 192.168.122.202
192.168.122.202 dev ens3 src 192.168.122.46
cache
```

上記の例では、ソースアドレスが **192.168.122.46** の **ens3** という名前のインターフェースからターゲットに直接つながります。これが予想される結果である場合は **ip a show dev ens3** を使用してインターフェースの詳細を取得し、このインターフェースが予想されるインターフェースであることを確認します。

または、結果が以下になる可能性もあります。

```
# ip route get 192.168.122.202
1.2.3.4 via 192.168.122.1 dev ens3 src 192.168.122.46
```

これは、正しくルーティングされるために **via** 値をパススルーします。トラフィックが正しくルーティングされていることを確認します。ルートトラフィックのデバッグについては、本書では扱われません。

ノード間ネットワークの他のデバッグオプションについては、以下を確認して解決できます。

- どちらの側にもイーサネットリンクがあるか? **ethtool <network_interface>** で **Link detected: yes** を検索します。
- デュプレックス設定とイーサネット速度はどちらの側でも適切に設定されているか? **ethtool <network_interface>** 情報の残りの部分を確認します。
- ケーブルは適切にプラグインされているか? 正しいポートに接続されているか?
- スイッチは適切に設定されているか?

ノード間設定が適切であることを確認した後は、両サイドで SDN 設定を確認する必要があります。

36.7. ローカルネットワークのデバッグ

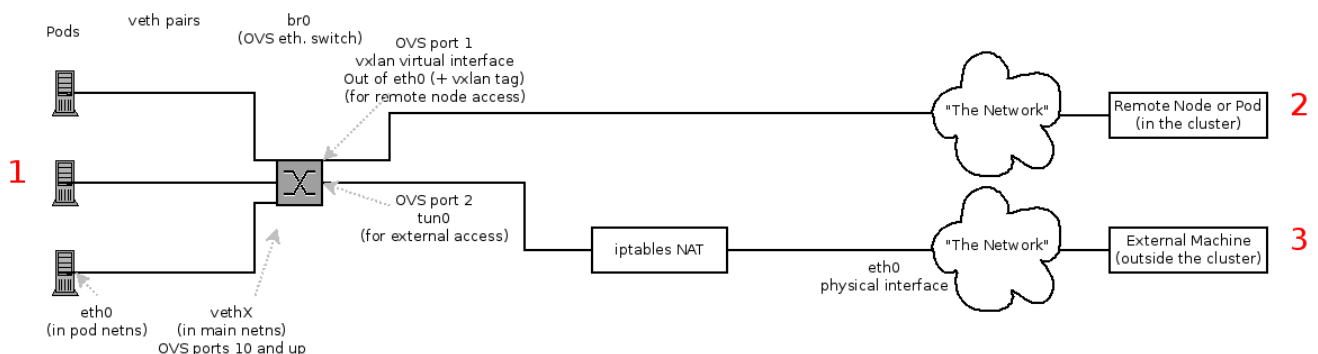
ここで通信できないものの、ノード間通信が設定された 1 つ以上のエンドポイントの一覧が表示されます。それぞれのエンドポイントについて問題点を特定する必要がありますが、まずは SDN が複数の異なる Pod についてノードでネットワークをどのように設定しているかについて理解する必要があります。

36.7.1. ノードのインターフェース

以下は OpenShift SDN が作成するインターフェースです。

- **br0**: コンテナが割り当てられる OVS ブリッジデバイスです。OpenShift SDN はこのブリッジにサブネットに固有ではないフロールールも設定します。
- **tun0**: OVS 内部ポート (**br0** のポート 2) です。これにはクラスターサブネットゲートウェイアドレスが割り当てられ、外部ネットワークアクセスに使用されます。OpenShift SDN はクラスターサブネットから NAT 経由で外部ネットワークにアクセスできるように **netfilter** およびルートルールを設定します。
- **vxlan_sys_4789**: OVS VXLAN デバイス (**br0** のポート 1) です。これはリモートノードのコンテナへのアクセスを提供します。OVS ルールでは **vxlan0** として参照されます。
- **vethX** (メイン netns 内): Docker netns における **eth0** の Linux 仮想イーサネットのピアです。これは他のポートのいずれかの OVS ブリッジに割り当てられます。

36.7.2. ノード内の SDN フロー



アクセスしようとしているもの (またはアクセスされるもの) によってパスは異なります。SDN が (ノード内に) で接続する場所は 4 か所あります。それらには上記の図で赤のラベルが付けられています。

- **Pod**: トラフィックは同じマシンのある Pod から別の Pod に移動します (1 から他の 1 へ)。
- **リモートノード (または Pod)**: トラフィックは同じクラスター内のローカル Pod からリモートノードまたは Pod に移動します (1 から 2 へ)。
- **外部マシン**: トラフィックはローカル Pod からクラスター外に移動します (1 から 3 へ)。

当然のこととして、トラフィックはこれらと反対方向でも移動します。

36.7.3. デバッグ手順

36.7.3.1. IP 転送は有効にされているか?

`sysctl net.ipv4.ip_forward` が 1 に設定されていること (およびホストが仮想マシンであるかどうか) を確認します。

36.7.3.2. ルートは正しく設定されているか?

`ip route` でルートテーブルを確認します。

```
# ip route
default via 192.168.122.1 dev ens3
10.128.0.0/14 dev tun0 proto kernel scope link      #
This sends all pod traffic into OVS
10.128.2.0/23 dev tun0 proto kernel scope link src 10.128.2.1 #
This is traffic going to local pods, overriding the above
169.254.0.0/16 dev ens3 scope link metric 1002      #
This is for Zeroconf (may not be present)
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.42.1 #
Docker's private IPs... used only by things directly configured by docker;
not OpenShift
192.168.122.0/24 dev ens3 proto kernel scope link src 192.168.122.46 #
The physical interface on the local subnet
```

10.128.x.x 行が表示されるはずですが (Pod ネットワークが設定内でデフォルト範囲に設定されていることを前提とします)。これが表示されない場合は、OpenShift Container Platform ログを確認します ([「ログの読み取り」](#) セクションを参照してください)。

36.7.4. Open vSwitch は正しく設定されているか?

両サイドで Open vSwitch ブリッジを確認します。

```
# ovs-vsctl list-br
br0
```

これは **br0** である必要があります。

ovs が認識するすべてのポートを一覧表示できます。

```
# ovs-ofctl -O OpenFlow13 dump-ports-desc br0
OFPST_PORT_DESC reply (OF1.3) (xid=0x2):
  1(vxlan0): addr:9e:f1:7d:4d:19:4f
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
  2(tun0): addr:6a:ef:90:24:a3:11
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
  8(vethe19c6ea): addr:1e:79:f3:a0:e8:8c
    config:      0
    state:       0
    current:     10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
  LOCAL(br0): addr:0a:7f:b4:33:c2:43
```



```
config:      PORT_DOWN
state:       LINK_DOWN
speed: 0 Mbps now, 0 Mbps max
```

とくにアクティブなすべての Pod の **vethX** デバイスがポートとして表示されるはずです。

次に、そのブリッジに設定されているフローを一覧表示します。

```
# ovs-ofctl -O OpenFlow13 dump-flows br0
```

ovs-subnet または **ovs-multitenant** プラグインのどちらを使用しているかに応じて結果は若干異なりますが、以下のような一般的な設定を確認することができます。

1. すべてのリモートノードには **tun_src=<node_IP_address>** に一致するフロー (ノードからの着信 VXLAN トラフィック) およびアクション **set_field:<node_IP_address>->tun_dst** を含む別のフロー (ノードへの発信 VXLAN トラフィック) が設定されている必要があります。
2. すべてのローカル Pod には **arp_spa=<pod_IP_address>** および **arp_tpa=<pod_IP_address>** に一致するフロー (Pod の着信および発信 ARP トラフィック) と、**nw_src=<pod_IP_address>** および **nw_dst=<pod_IP_address>** に一致するフロー (Pod の着信および発信 IP トラフィック) が設定されている必要があります。

フローがない場合は、「[ログの読み取り](#)」セクションを参照してください。

36.7.4.1. iptables 設定に誤りがないか?

iptables-save の出力をチェックし、トラフィックにフィルターを掛けていないことを確認します。OpenShift Container Platform は通常の操作時に iptables ルールを設定するため、ここにエントリーが表示されていても不思議なことではありません。

36.7.4.2. 外部ネットワークは正しく設定されているか?

外部ファイアウォール (ある場合) を確認し、ターゲットアドレスへのトラフィックを許可するかどうかを確認します (これはサイトごとに異なるため、本書では扱われません)。

36.8. 仮想ネットワークのデバッグ

36.8.1. 仮想ネットワークのビルドに障害が発生している

仮想ネットワーク (例: OpeStack) を使用して OpenShift Container Platform をインストールしている場合で、ビルドに障害が発生している場合、ターゲットノードホストの最大伝送単位 (MTU: maximum transmission unit) はプライマリーネットワークインターフェース (例: **eth0**) の MTU との互換性がない可能性があります。

ビルドが正常に完了するには、データをノードホスト間で渡すために SDN の MTU が eth0 ネットワークの MTU よりも小さくしなければなりません。

1. **ip addr** コマンドを実行してネットワークの MTU を確認します。

```
# ip addr
---
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
```

```
link/ether fa:16:3e:56:4c:11 brd ff:ff:ff:ff:ff:ff
inet 172.16.0.0/24 brd 172.16.0.0 scope global dynamic eth0
    valid_lft 168sec preferred_lft 168sec
inet6 fe80::f816:3eff:fe56:4c11/64 scope link
    valid_lft forever preferred_lft forever
---
```

上記のネットワークの MTU は 1500 です。

2. ノード設定の MTU はネットワーク値よりも小さくありません。ターゲットに設定されたノードホストの **mtu** を確認します。

```
# cat /etc/origin/node/node-config.yaml
...
networkConfig:
  mtu: 1450
  networkPluginName: company/openshift-ovs-subnet
...
```

上記のノード設定ファイルでは、**mtu** 値はネットワーク MTU よりも低くなるため、設定は不要になります。**mtu** 値がこれより高くなる場合はファイルを編集して、値をプライマリーネットワークインターフェースの MTU よりも少なくとも 50 単位分下げてノードサービスを再起動します。これにより、より大きなパケットのデータをノード間で渡すことが可能になります。

36.9. POD の EGRESS のデバッグ

Pod から外部サービスへのアクセスを試行する場合、以下の例ようになります。

```
curl -kv github.com
```

DNS が適切に解決されていることを確認します。

```
dig +search +noall +answer github.com
```

これにより、github サーバーの IP アドレスが返されるはずですが、正しいアドレスが返されていることを確認します。アドレスが返されない場合やお使いのマシンのいずれかのアドレスが返される場合、ローカル DNS サーバーのワイルドカードエントリーに一致している可能性があります。

これを修正するには、ワイルドカードエントリーを持つ DNS サーバーが **/etc/resolv.conf** の **nameserver** として一覧表示されていないことを確認するか、またはワイルドカードドメインが **search** 一覧に一覧表示されていないことを確認する必要があります。

正しい IP アドレスが返される場合、「[ローカルネットワークのデバッグ](#)」の前述のデバッグに関するアドバイスに従ってください。通常、トラフィックはポート 2 の Open vSwitch から **iptables** ルールおよびルートテーブルを通過するはずですが。

36.10. ログの読み取り

次を実行します: **journalctl -u atomic-openshift-node.service --boot | less**

Output of setup script: 行を検索します。'+' で始まるすべての行については、その下にスク립ト手順が記述されます。この部分で明らかなエラーがあるかどうかを調べます。

スクリプトを追ってみると、**Output of adding table=0** という行を見つけることができるはずです。これは OVS ルールであり、エラーは存在しないはずです。

36.11. KUBERNETES のデバッグ

iptables -t nat -L を確認して、サービスがローカルマシンで **kubeproxy** の適切なポートに NAT されていることを確認します。



警告

これについては、まもなく全面的に変更されます... Kubeproxy は除去され、**iptables** のみのソリューションに置き換わります。

36.12. 診断ツールを使用したネットワークの問題の検出

クラスター管理者として診断ツールを実行し、共通するネットワークの問題を診断します。

```
# oc adm diagnostics NetworkCheck
```

診断ツールは、指定したコンポーネントのエラー状態をチェックする一連のチェックを実行します。詳細は、「[診断ツール](#)」のセクションを参照してください。



注記

現時点で、診断ツールでは IP フェイルオーバーの問題を診断できません。回避策として、スクリプトをマスターの <https://raw.githubusercontent.com/openshift/openshift-sdn/master/hack/ipf-debug.sh> で (またはマスターへのアクセスのある別のマシンから) 実行して役に立つデバッグ情報を生成できます。ただし、このスクリプトはサポート対象外です。

デフォルトで、**oc adm diagnostics NetworkCheck** はエラーのログを **/tmp/openshift/** に記録します。これは **--network-logdir** オプションで設定できます。

```
# oc adm diagnostics NetworkCheck --network-logdir=<path/to/directory>
```

36.13. その他の注意点

36.13.1. ingress についての追加情報

- Kube: サービスを NodePort として宣言し、クラスター内のすべてのマシンでそのポートを要求し、kube-proxy およびサポートする Pod にルーティングします。 <https://kubernetes.io/docs/concepts/services-networking/service/#type-nodeport> を参照してください (一部のノードは外部からアクセスできる必要があります)。
- Kube: LoadBalancer として宣言し、**独自に** 判別したオブジェクトが残りを実行します。
- OS/AE: いずれもルーターを使用します。

36.13.2. TLS ハンドシェイクのタイムアウト

Pod がデプロイに失敗する場合、docker ログで TLS ハンドシェイクのタイムアウトを確認します。

```
$ docker log <container_id>
...
[...] couldn't get deployment [...] TLS handshake timeout
...
```

この状態は通常はセキュアな接続を確立する際のエラーであり、このエラーは tun0 とプライマリーインターフェース (例: eth0) 間の MTU 値の大きな違い (例: tun0 MTU が 1500 に対し eth0 MTU が 9000 (ジャンボフレーム) である場合) によって引き起こされる可能性があります。

36.13.3. デバッグについての他の注意点

- (Linux 仮想イーサネットペア) のピアインターフェースは **ethtool -S ifname** で判別できません。
- ドライバertype: **ethtool -i ifname**

第37章 診断ツール

37.1. 概要

oc adm diagnostics コマンドは一連のチェックを実行し、ホストまたはクラスターのエラーの状態についてチェックします。とくに以下を実行します。

- デフォルトのレジストリーおよびルーターが実行中であり、正しく設定されていることを確認します。
- **ClusterRoleBindings** および **ClusterRoles** で、ベースポリシーとの整合性を確認します。
- すべてのクライアント設定コンテキストが有効で接続可能であることを確認します。
- SkyDNS が適切に機能しており、Pod に SDN 接続があることを確認します。
- ホストのマスターおよびノード設定を検証します。
- ノードが実行中で、利用可能であることを確認します。
- 既知のエラーについてホストログを分析します。
- systemd ユニットがホストに対して予想通りに設定されていることを確認します。

37.2. 診断ツールの使用

OpenShift Container Platform は数多くの方法でデプロイできます (ソースからのビルド、VM イメージ、コンテナイメージの使用、またはエンタープライズ RPM の使用など)。それぞれの方法には異なる設定および環境が想定されます。環境について想定される相違を最小限にするために診断は **openshift** バイナリーに追加されており、OpenShift Container Platform サーバーまたはクライアントがどこに置かれていても、診断を全く同じ環境で実行できるようになっています。

診断ツールを使用するには (マスターホスト上で使用するのが望ましい)、クラスター管理者として以下を実行します。

```
$ oc adm diagnostics
```

これは利用可能なすべての診断を実行し、適用されないものは省略します。

名前別に 1 つまたは複数の特定の診断を実行するか、または問題に対応する際に使用する特定の診断を実行することができます。以下は例になります。

```
$ oc adm diagnostics <name1> <name2>
```

ほとんどの場合、このオプションには機能する設定ファイルが必要になります。たとえば **NodeConfigCheck** は、ノード設定が利用可能でない場合は実行されません。

診断では標準的な場所にある設定ファイルを検索します。

- クライアント:
 - **\$KUBECONFIG** 環境変数で示されます。

- ~/.kube/config file
- マスター:
 - /etc/origin/master/master-config.yaml
- ノード:
 - /etc/origin/node/node-config.yaml

標準以外の場所もフラグ (**--config**、**--master-config** および **--node-config**) で指定できます。設定ファイルが見つからないか、または指定されない場合、関連する診断は省略されます。

利用可能な診断には以下が含まれます。

診断名	目的
AggregatedLogging	集約されたログ統合を使用して適切な設定および操作を確認します。
AnalyzeLogs	systemd サービスログで問題の有無を確認します。チェックの実行に設定ファイルは不要です。
ClusterRegistry	クラスターにビルドおよびイメージストリーム用の有効な Docker レジストリーがあることを確認します。
ClusterRoleBindings	デフォルトのクラスターロールバインディングが存在し、ベースポリシーに応じて予想されるサブジェクトが含まれることを確認します。
ClusterRoles	クラスターロールが存在し、ベースポリシーに応じて予想されるパーミッションが含まれることを確認します。
ClusterRouter	クラスター内に有効なデフォルトルーターがあることを確認します。
ConfigContexts	クライアント設定の各コンテキストが完成したものであり、その API サーバーへの接続があることを確認します。
DiagnosticPod	アプリケーションの観点で診断を実行する Pod を作成します。これは Pod 内の DNS が予想通りに機能しており、デフォルトサービスアカウントの認証情報がマスター API に対して正しく認証されることを確認します。

診断名	目的
EtcWriteVolume	一定期間における etcd に対する書き込みのボリュームを確認し、操作およびキー別にそれらを分類します。この診断は他の診断と同じ速度で実行されず、かつ etcd への負荷を増えることから、とくに要求される場合にのみ実行されます。
MasterConfigCheck	このホストのマスター設定ファイルで問題の有無を確認します。
MasterNode	このホストで実行されているマスターがノードも実行していることを確認し、それがクラスター SDN のメンバーであることを確認します。
MetricsApiProxy	統合 Heapster メトリクスがクラスター API プロキシ経由でアクセス可能であることを確認します。
NetworkCheck	<p>複数ノードで診断 Pod を作成し、アプリケーションの観点で共通するネットワーク問題を診断します。たとえば、これは Pod がサービスや他の Pod、および外部ネットワークに接続できることを確認します。</p> <p>エラーがある場合は、この診断は詳細な分析用としてローカルディレクトリー (デフォルトで <code>/tmp/openshift/</code>) に結果および取得されたファイルを保存します。ディレクトリーは <code>--network-logdir</code> フラグで指定することができます。</p>
NodeConfigCheck	このホストのノード設定ファイルで問題の有無を確認します。
NodeDefinitions	マスター API で定義されたノードが利用可能な状態にあり、Pod をスケジュールできることを確認します。
RouteCertificateValidation	すべてのルート証明書で、拡張される検証で拒否される可能性のあるものがあるかどうかを確認します。
ServiceExternalIPs	マスター設定に基づいて無効にされている外部 IP を指定する既存サービスの有無を確認します。
UnitStatus	OpenShift Container Platform に関連して、このホストでユニットについての systemd ステータスを確認します。チェックの実行に設定ファイルは不要です。

37.3. サーバー環境における診断の実行

マスターおよびノードの診断は、Ansible でデプロイされるクラスターで最も役立ちます。この場合、いくつかの診断上のメリットがあります。

- マスターおよびノード設定は標準的な場所にある設定ファイルに基づく。
- systemd ユニットがサーバーを管理するように設定される。
- すべてのコンポーネントが journald に対してログを記録する。

Ansibleが指定する場所に設定ファイルが置かれるため、それらを検索する場所を指定する必要がなくなります。フラグなしで **oc adm diagnostics** を実行すると、標準的な場所にあるマスターおよびノード設定ファイルを検索でき、それらが見つかり次第使用することができます。これにより、Ansible を使用してインストールする場合のされるユースケースをできる限り単純にすることができます。さらに、予想される場所にない設定ファイルを指定することも容易になります。

```
$ oc adm diagnostics --master-config=<file_path> --node-config=<file_path>
```

systemd ユニットおよび journald のログエントリは現在のログ診断ロジックに必要です。他のデプロイメントタイプの場合、ログはファイルに記録されるか、標準出力に出力されるか、またはノードとマスターを組み合わせる可能性もあります。このような場合にログ診断は適切に機能せず、省略されます。

37.4. クライアント環境での診断の実行

通常のユーザーまたは **cluster-admin** ユーザーとしてのアクセスを持つ場合、OpenShift Container Platform マスターまたはノードが動作するホストでの実行が可能です。診断はユーザーが利用できるアクセスをできるだけ多く使用することを試みます。

通常のアクセスを持つクライアントはマスターへの接続を診断し、診断 Pod を実行することができます。複数のユーザーまたはマスターが設定されている場合、接続のテストはそれらすべてを対象に実行されますが、診断 Pod は現行ユーザー、サーバー、またはプロジェクトに対してのみ実行されます。

(すべてのユーザーが利用できるが、現行マスターのみで) 利用可能な **cluster-admin** アクセスを持つクライアントは、ノード、レジストリー、ルーターなどのインフラストラクチャーのステータスを診断できます。いずれの場合も、**oc adm diagnostics** を実行すると、標準的な場所にあるクライアント設定を検索でき、利用可能な場合はこれを使用できます。

37.5. ANSIBLE ベースの正常性チェック

追加の正常性診断チェックは、OpenShift Container Platform クラスターのインストールおよび管理に使用される [Ansible ベースのツール](#) で利用できます。それらは現行 OpenShift Container Platform インストールによくあるデプロイメントの問題を報告します。

これらのチェックは、**ansible-playbook** コマンドの使用 ([通常インストール \(Advanced installation\)](#) で使用されるのと同じ方式) によるか、または **openshift-ansible** の [コンテナ化されたバージョン](#) として実行できます。**ansible-playbook** 方式については、チェックは **atomic-openshift-utils** RPM パッケージを使って行われます。コンテナ化方式の場合は、**openshift3/ose-ansible** コンテナイメージが [Red Hat Container Registry](#) 経由で配布されます。各方式の使用例については、後続のセクションで説明されます。 |

以下の正常性チェックは、デプロイされた OpenShift Container Platform クラスターを対象に、指定された **health.yml** playbook を使用して Ansible インベントリーファイルに対して実行されることが意図されている診断タスクのセットのこと指します。



警告

正常性チェック Playbook はホストに変更を加える可能性があります。そのため、これらの Playbook は Ansible を使ってデプロイされたクラスターで、デプロイ時に使用したものと同一インベントリーファイルを使う場合にのみ使用できます。ほとんどの場合、発生する変更はチェックでの必須情報の収集に必要な依存関係のインストールに関係するものですが、特定のシステムコンポーネント (**docker** またはネットワークなど) も、現在の状態がインベントリーファイルの設定と異なる場合に変更される可能性があります。これらの正常性チェックは、使用するインベントリーファイルが現在のクラスター設定に変更を加えないことが予想される場合にのみ実行してください。

表37.1 正常性診断チェック

チェック名	目的
etcd_imagedata_size	<p>このチェックは、etcd クラスターの OpenShift Container Platform イメージデータの合計サイズを測定します。このチェックは計算されたサイズがユーザー定義の制限を超える場合に失敗します。制限が指定されない場合、このチェックはイメージデータのサイズが etcd クラスターで現在使用されている領域の 50 % 以上になる場合に失敗します。</p> <p>このチェックに失敗することは、etcd の多くの領域が OpenShift Container Platform イメージデータによって使用されていることを示し、これにより、最終的には etcd クラスターがクラッシュする可能性があります。</p> <p>ユーザー定義の制限は etcd_max_image_data_size_bytes 変数を渡すことで設定できます。たとえば、etcd_max_image_data_size_bytes=40000000000 を設定する場合、etcd に保存されるイメージデータの合計サイズが 40 GB を超えるとチェックが失敗します。</p>
etcd_traffic	<p>このチェックは、etcd ホストの通常よりも高いレベルのトラフィックを検知します。etcd 期間の警告と共に journalctl ログエントリーが見つかる場合に失敗します。</p> <p>etcd パフォーマンスを強化する方法についての詳細は、「Recommended Practices for OpenShift Container Platform etcd Hosts」および「Red Hat Knowledgebase」を参照してください。</p>

チェック名	目的
etcd_volume	<p>このチェックにより、etcd クラスターのボリューム使用がユーザー指定の最大しきい値を超えないようにできます。最大しきい値が指定されていない場合、デフォルトは合計ボリュームサイズの 90% に設定されます。</p> <p>ユーザー定義の制限は、etcd_device_usage_threshold_percent 変数を渡すことで設定できます。</p>
docker_storage	<p>docker デーモン (ノードおよびコンテナ化されたインストール) に依存するホストでのみ実行されます。docker の合計使用量がユーザー定義制限を超えないことを確認します。ユーザー定義の制限が設定されていない場合、docker 使用量の最大しきい値のデフォルトは利用可能な合計サイズの 90% になります。</p> <p>合計パーセントの使用量についてのしきい値の制限は、max_thinpool_data_usage_percent=90 などのようにインベントリーファイルの変数で設定できます。</p> <p>また、このチェックは docker のストレージが サポートされる設定 を使用していることを確認します。</p>
curator、elasticsearch、fluentd、kibana	<p>この一連のチェックは、Curator、Kibana、Elasticsearch、および Fluentd Pod がデプロイされており、これらが running 状態であることを検証し、接続を制御ホストと公開される Kibana URL 間で確立できることを検証します。これらのチェックは、openshift_logging_install_logging インベントリー変数が true に設定されている場合にのみ実行され、それらが クラスターロギング が有効にされているデプロイメントで実行されるようにします。</p>

チェック名	目的
logging_index_time	<p>このチェックは、ロギングスタックデプロイメントにおけるログ作成から Elasticsearch によるログ集計までの通常の時間差よりも値が高くなるケースを検知します。新規のログエントリーがタイムアウト内 (デフォルトでは 30 秒内) に Elasticsearch によってクエリーされない場合に失敗します。このチェックはロギングが有効にされている場合にのみ実行されます。</p> <p>ユーザー定義のタイムアウトは、openshift_check_logging_index_timeout_seconds 変数を渡すことで設定できます。たとえば、openshift_check_logging_index_timeout_seconds=45 を設定すると、新規作成されるログエントリーが 45 秒を経過しても Elasticsearch でクエリーされない場合に失敗します。</p>



注記

インストールプロセスの一部として実行されることが意図されている同様のチェックセットについては、「[Configuring Cluster Pre-install Checks](#)」を参照してください。証明書の有効期限をチェックするための別のチェックのセットについては、「[Redeploying Certificates](#)」を参照してください。

37.5.1. ansible-playbook による正常性チェックの実行

ansible-playbook コマンドを使用して **openshift-ansible** 正常性チェックを実行するには、クラスターのインベントリーファイルを指定し、**health.yml** playbook を実行します。

```
# ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    checks/health.yml
```

コマンドラインに変数を設定するには、**key=value** 形式の必要な変数に **-e** フラグを組み込みます。以下は例になります。

```
# ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
    checks/health.yml
    -e openshift_check_logging_index_timeout_seconds=45
    -e etcd_max_image_data_size_bytes=40000000000
```

特定のチェックを無効にするには、Playbook を実行する前にインベントリーファイルのカンマ区切りのチェック名の一覧と共に変数 **openshift_disable_check** を組み込みます。以下は例になります。

```
openshift_disable_check=etcd_traffic,etcd_volume
```

または、**ansible-playbook** コマンドの実行時に **-e openshift_disable_check=<check1>, <check2>** で変数として無効にするチェックを設定します。

37.5.2. Docker CLI での正常性チェックの実行

Docker コンテナで **openshift-ansible** playbook を実行し、Docker CLI で **ose-ansible** イメージを実行できるホストでの Ansible のインストールおよび設定の手間を省くことができます。

これを実行するには、以下の **docker run** コマンドをコンテナの実行権限を持つ非 root ユーザーとして実行する際にクラスタのインベントリーファイルと **health.yml** playbook を指定します。

```
# docker run -u `id -u` \ ❶
    -v $HOME/.ssh/id_rsa:/opt/app-root/src/.ssh/id_rsa:Z,ro \ ❷
    -v /etc/ansible/hosts:/tmp/inventory:ro \ ❸
    -e INVENTORY_FILE=/tmp/inventory \
    -e PLAYBOOK_FILE=playbooks/openshift-checks/health.yml \ ❹
    -e OPTS="-v -e openshift_check_logging_index_timeout_seconds=45 -e
etcd_max_image_data_size_bytes=40000000000" \ ❺
    openshift3/ose-ansible
```

- ❶ これらのオプションにより、コンテナは現行ユーザーと同じ UID で実行されます。これは SSH キーをコンテナ内で読み取られるようにするようパーミッションで必要になります (SSH プライベートキーはその所有者によってのみ読み取り可能であることが予想されます)。
- ❷ SSH キーは、コンテナを非 root ユーザーとして実行するなどの通常の使用では **/opt/app-root/src/.ssh** の下にマウントします。
- ❸ **/etc/ansible/hosts** は、異なる場合はクラスタのインベントリーファイルの場所に切り替えます。このファイルは、コンテナの **INVENTORY_FILE** 環境変数に基づいて使用される **/tmp/inventory** にバインドおよびマウントされます。
- ❹ **PLAYBOOK_FILE** 環境変数は、コンテナ内の **/usr/share/ansible/openshift-ansible** に関連して **health.yml** playbook の場所に設定されます。
- ❺ **-e key=value** 形式で単一の実行に必要な変数を設定します。

上記のコマンドでは、SSH キーは **:Z** フラグを使ってマウントされ、コンテナが SSH キーを制限付き SELinux コンテキストから読み取れるようにします。これは、元の SSH キーファイルのラベルが **system_u:object_r:container_file_t:s0:c113,c247** などに変更されることを意味しています。**:Z** についての詳細は、**docker-run(1)** man ページを参照してください。

これらのボリュームマウント仕様に関連して予期しない影響が生じる可能性があることに注意してください。たとえば、**\$HOME/.ssh** ディレクトリーをマウント (したがってラベルを変更) する場合、**sshd** はパブリックキーにアクセスしてリモートログインを許可できなくなります。元のファイルラベルの変更を防ぐには、SSH キー (またはディレクトリー) のコピーをマウントすることが推奨されています。

.ssh ディレクトリー全体をマウントする理由はいくつかあります。たとえば、これにより SSH 設定を使用してキーをホストに一致させたり、他の接続パラメーターを変更したりできます。また **known_hosts** ファイルを指定したり、SSH にホストキーを検証させたりすることができます。このホストキーの検証はデフォルトでは無効にされていますが、**-e ANSIBLE_HOST_KEY_CHECKING=True** を **docker** コマンドラインに追加することで環境変数を使用して再度有効にすることができます。

第38章 アプリケーションのアイドルリング

38.1. 概要

OpenShift Container Platform 管理者は、アプリケーションをアイドルリング状態にしてリソース消費を減らすことができます。これは、コストがリソース消費と関連付けられるパブリッククラウドにデプロイされている場合に役立ちます。

スケーラブルなリソースが使用されていない場合、OpenShift Container Platform はリソースを検出した後にそれらを 0 レプリカに設定してアイドルリングします。ネットワークトラフィックがリソースに送信される場合、レプリカをスケールアップしてアイドルリング解除を実行し、操作を続行します。

アプリケーションは複数のサービスやデプロイメント設定などの他のスケーラブルなリソースで構成されています。アプリケーションのアイドルリングには、関連するすべてのリソースのアイドルリングを実行することが関係します。

38.2. アプリケーションのアイドルリング

アプリケーションのアイドルリングには、サービスに関連付けられたスケーラブルなリソース (デプロイメント設定、レプリケーションコントローラーなど) を検索する必要があります。アプリケーションのアイドルリングには、サービスを検索してこれをアイドルリング状態としてマークし、リソースを zero レプリカにスケールダウンすることが関係します。

oc idle コマンドを実行して[単一サービスのアイドルリング](#)を実行するか、または **--resource-names-file** オプションを使用して[複数サービスのアイドルリング](#)を実行できます。

38.2.1. 単一サービスのアイドルリング

以下のコマンドを使用して単一サービスをアイドルリングします。

```
$ oc idle <service>
```

38.2.2. 複数サービスのアイドルリング

必要なサービスの一覧を作成し、**--resource-names-file** オプションを **oc idle** コマンドで使用することで複数サービスをアイドルリングします。

これは、アプリケーションがプロジェクト内の一連のサービスにまたがる場合や、同じプロジェクト内で複数のアプリケーションを一括してアイドルリングするため、複数サービスをスクリプトを併用してアイドルリングする場合に役立ちます。

1. 複数サービスの一覧を含むテキストファイルを作成します (それぞれを各行に指定)。
2. **--resource-names-file** オプションを使用してサービスをアイドルリングします。

```
$ oc idle --resource-names-file <filename>
```



注記

idle コマンドは単一プロジェクトに制限されます。クラスター全体でアプリケーションをアイドルリングするには、各プロジェクトに対して idle コマンドをそれぞれ実行します。

38.3. アプリケーションのアイドルリング解除

アプリケーションサービスは、ネットワークトラフィックを受信し、直前の状態に再びスケールアップすると再びアクティブになります。これには、サービスへのトラフィックとルートを通るトラフィックの両方が含まれます。

アプリケーションのアイドルリング解除はリソースをスケールアップすることで手動で実行できます。たとえば、`deploymentconfig` をスケールアップするには、以下のコマンドを実行します。

```
$ oc scale --replicas=1 dc <deploymentconfig>
```



注記

現時点で、ルーターによる自動アイドルリング解除はデフォルトの HAProxy ルーターのみでサポートされています。

第39章 クラスター容量の分析

39.1. 概要

クラスター管理者は、クラスター容量ツールを使用して、現在のリソースが使い切られる前にそれらを増やすべくスケジュール可能な Pod 数を表示し、スケジュール可能な Pod 数を表示したり、Pod を今後スケジュールできるようにすることができます。この容量は、クラスター内の個別ノードからのものを集めたものであり、これには CPU、メモリー、ディスク領域などが含まれます。

クラスター容量ツールはより正確な見積もりを出すべく、スケジュールの一連の意思決定をシミュレーションし、リソースが使い切られる前にクラスターでスケジュールできる入力 Pod のインスタンス数を判別します。



注記

ノード間に分散しているすべてのリソースがカウントされないため、残りの割り当て可能な容量は概算となります。残りのリソースのみが分析対象となり、クラスターでのスケジュール可能な所定要件を持つ Pod のインスタンス数という点から消費可能な容量を見積もります。

Pod のスケジューリングはその選択およびアフィニティー条件に基づいて特定のノードセットについてのみサポートされる可能性があります。そのため、クラスターでスケジュール可能な残りの Pod 数を見積もることが困難になる場合があります。

クラスター容量分析ツールは、コマンドラインからスタンドアロンのユーティリティとして実行することも、OpenShift Container Platform クラスター内の Pod で **ジョブとして** 実行することもできます。これを Pod 内のジョブとして実行すると、介入なしに複数回実行することができます。

39.2. コマンドラインでのクラスター容量分析の実行

コマンドラインでツールを実行するには、以下を実行します。

```
$ cluster-capacity --kubeconfig <path-to-kubeconfig> \
  --podspec <path-to-pod-spec>
```

--kubeconfig オプションは Kubernetes 設定ファイルを示し、**--podspec** オプションはツールがリソース使用状況を見積もるために使用するサンプル Pod 仕様ファイルを示します。**podspec** はそのリソース要件を **limits** または **requests** として指定します。クラスター容量ツールは、Pod のリソース要件をその見積もりの分析に反映します。

Pod 仕様入力の例は以下の通りです。

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
```



```

imagePullPolicy: Always
resources:
  limits:
    cpu: 150m
    memory: 100Mi
  requests:
    cpu: 150m
    memory: 100Mi

```

--verbose オプションを追加して、クラスター内の各ノードにスケジュールできる Pod 数についての詳細説明を出力できます。

```

$ cluster-capacity --kubeconfig <path-to-kubeconfig> \
  --podspect <path-to-pod-spec> --verbose

```

出力は以下のようになります。

```

small-pod pod requirements:
- CPU: 150m
- Memory: 100Mi

```

The cluster can schedule 52 instance(s) of the pod small-pod.

Termination reason: Unschedulable: No nodes are available that match all of the following predicates:: Insufficient cpu (2).

Pod distribution among nodes:

```

small-pod
- 192.168.124.214: 26 instance(s)
- 192.168.124.120: 26 instance(s)

```

上記の例では、クラスターにスケジュールできる Pod の見積り数は 52 です。

39.3. POD 内のジョブとしてのクラスター容量分析の実行

クラスター容量ツールを Pod 内のジョブとして実行すると、ユーザーの介入なしに複数回実行できるという利点があります。クラスター容量ツールをジョブとして実行するには、**ConfigMap** を使用する必要があります。

1. クラスターロールを作成します。

```

$ cat << EOF | oc create -f -
kind: ClusterRole
apiVersion: v1
metadata:
  name: cluster-capacity-role
rules:
- apiGroups: [""]
  resources: ["pods", "nodes", "persistentvolumeclaims",
    "persistentvolumes", "services"]
  verbs: ["get", "watch", "list"]
EOF

```


2. サービスアカウントを作成します。

```
$ oc create sa cluster-capacity-sa
```

3. ロールをサービスアカウントに追加します。

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

4. Pod 仕様を定義し、作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
      requests:
        cpu: 150m
        memory: 100Mi
```

5. クラスター容量分析は、**cluster-capacity-configmap** という名前の **ConfigMap** を使用してボリュームにマウントされ、入力 Pod 仕様ファイル **pod.yaml** はパス **/test-pod** のボリューム **test-volume** にマウントされます。

ConfigMap を作成していない場合は、ジョブの作成前にこれを作成します。

```
$ oc create configmap cluster-capacity-configmap \
  --from-file=pod.yaml=pod.yaml
```

6. ジョブ仕様ファイルの以下のサンプルを使用してジョブを作成します。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: cluster-capacity-job
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: cluster-capacity-pod
    spec:
      containers:
      - name: cluster-capacity
```

```

        image: openshift/origin-cluster-capacity
        imagePullPolicy: "Always"
        volumeMounts:
        - mountPath: /test-pod
          name: test-volume
        env:
        - name: CC_INCLUSTER ❶
          value: "true"
        command:
        - "/bin/sh"
        - "-ec"
        - |
          /bin/cluster-capacity --podspec=/test-pod/pod.yaml --
verbose
        restartPolicy: "Never"
        serviceAccountName: cluster-capacity-sa
        volumes:
        - name: test-volume
          configMap:
            name: cluster-capacity-configmap

```

- ❶ クラスター容量ツールにクラスター内で Pod として実行されていることを認識させる環境変数です。

ConfigMap の **pod.yaml** キーは Pod 仕様ファイル名と同じですが、これは必須ではありません。これを実行することで、入力 Pod 仕様ファイルは **/test-pod/pod.yaml** として Pod 内でアクセスできます。

7. クラスター容量イメージを Pod のジョブとして実行します。

```
$ oc create -f cluster-capacity-job.yaml
```

8. ジョブログを確認し、クラスター内でスケジュールできる Pod の数を確認します。

```
$ oc logs jobs/cluster-capacity-job
small-pod pod requirements:
  - CPU: 150m
  - Memory: 100Mi
```

The cluster can schedule 52 instance(s) of the pod small-pod.

Termination reason: Unschedulable: No nodes are available that match all of the following predicates:: Insufficient cpu (2).

Pod distribution among nodes:

```
small-pod
  - 192.168.124.214: 26 instance(s)
  - 192.168.124.120: 26 instance(s)
```