



OpenShift Container Platform 3.9

アーキテクチャー

OpenShift Container Platform 3.9 アーキテクチャー情報

OpenShift Container Platform 3.9 アーキテクチャー

OpenShift Container Platform 3.9 アーキテクチャー情報

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

インフラストラクチャーおよびコアコンポーネントを含む OpenShift Container Platform 3.9 のアーキテクチャーについて説明します。これらのトピックでは、認証、ネットワークおよびソースコード管理についても扱います。

目次

第1章 概要	7
1.1. 各種の層について	7
1.2. OPENSIFT CONTAINER PLATFORM アーキテクチャーについて	8
1.3. OPENSIFT CONTAINER PLATFORM のセキュリティーを保護する方法	9
1.3.1. TLS サポート	9
第2章 インフラストラクチャーコンポーネント	12
2.1. KUBERNETES インフラストラクチャー	12
2.1.1. 概要	12
2.1.2. マスター	12
2.1.2.1. 高可用性マスター	13
2.1.3. ノード	14
2.1.3.1. Kubelet	14
2.1.3.2. サービスプロキシ	14
2.1.3.3. ノードオブジェクト定義	14
2.2. CONTAINER レジストリー	15
2.2.1. 概要	15
2.2.2. 統合 OpenShift Container レジストリー	15
2.2.3. サードパーティーレジストリー	15
2.2.3.1. 認証	16
2.3. WEB コンソール	16
2.3.1. 概要	16
2.3.2. CLI ダウンロード	17
2.3.3. ブラウザーの要件	18
2.3.4. プロジェクトの概要	18
2.3.5. JVM コンソール	20
2.3.6. StatefulSet	21
第3章 コアとなる概念	23
3.1. 概要	23
3.2. コンテナおよびイメージ	23
3.2.1. コンテナ	23
3.2.1.1. Init コンテナ	23
3.2.2. イメージ	24
イメージバージョンタグポリシー	24
3.2.3. コンテナレジストリー	25
3.3. POD およびサービス	25
3.3.1. Pod	25
3.3.1.1. Pod 再起動ポリシー	28
3.3.1.2. Pod の Preset (プリセット) を使用した情報の Pod への挿入	29
3.3.2. Init コンテナ	29
3.3.3. サービス	31
3.3.3.1. サービス externalIP	31
3.3.3.2. サービス ingressIP	32
3.3.3.3. サービス NodePort	33
3.3.3.4. サービスプロキシモード	33
3.3.3.5. ヘッドレスサービス	34
3.3.3.5.1. ヘッドレスサービスの作成	34
3.3.3.5.2. ヘッドレスサービスを使用したエンドポイントの検出	35
3.3.4. ラベル	35
3.3.5. エンドポイント	36
3.4. プロジェクトおよびユーザー	36

3.4.1. ユーザー	36
3.4.2. Namespace	37
3.4.3. プロジェクト	37
3.4.3.1. インストール時に提供されるプロジェクト	38
3.5. ビルドおよびイメージストリーム	38
3.5.1. ビルド	38
3.5.1.1. Docker ビルド	39
3.5.1.2. Source-to-Image (S2I) ビルド	39
3.5.1.3. カスタムビルド	40
3.5.1.4. Pipeline ビルド	40
3.5.2. イメージストリーム	41
3.5.2.1. 重要な用語	43
3.5.2.2. イメージストリームの設定	44
3.5.2.3. イメージストリームイメージ	45
3.5.2.4. イメージストリームタグ	45
3.5.2.5. イメージストリーム変更トリガー	47
3.5.2.6. イメージストリームのマッピング	48
3.5.2.7. イメージストリームの使用	51
3.5.2.7.1. イメージストリームについての情報の取得	51
3.5.2.7.2. 追加タグのイメージストリームへの追加	52
3.5.2.7.3. 外部イメージのタグの追加	53
3.5.2.7.4. イメージストリームタグの更新	53
3.5.2.7.5. イメージストリームタグのイメージストリームからの削除	53
3.5.2.7.6. タグの定期的なインポートの設定	53
3.6. デプロイメント	54
3.6.1. レプリケーションコントローラー	54
3.6.2. ジョブ	55
3.6.3. デプロイメントおよびデプロイメント設定	55
3.7. テンプレート	57
3.7.1. 概要	57
第4章 追加の概念	58
4.1. 認証	58
4.1.1. 概要	58
4.1.2. ユーザーおよびグループ	58
4.1.3. API 認証	58
4.1.3.1. 権限の借用	59
4.1.4. OAuth	59
4.1.4.1. OAuth クライアント	60
4.1.4.2. OAuth クライアントとしてのサービスアカウント	61
4.1.4.3. OAuth クライアントとしてのサービスアカウントのリダイレクト URI	61
4.1.4.3.1. OAuth の API イベント	63
4.1.4.3.1.1. 設定ミスによって生じる API イベントのサンプル	65
4.1.4.4. 統合	67
4.1.4.5. OAuth サーバーメタデータ	68
4.1.4.6. OAuth トークンの取得	69
4.1.4.7. Prometheus の認証メトリクス	70
4.2. 承認	70
4.2.1. 概要	70
4.2.2. 承認の評価	76
4.2.3. クラスターおよびローカル RBAC	77
4.2.4. クラスターロールおよびローカルロール	77
4.2.4.1. クラスターロールの更新	78

4.2.4.2. カスタムロールおよびパーミッションの適用	79
4.2.4.3. クラスターロールの集計	79
4.2.5. SCC (Security Context Constraints)	79
4.2.5.1. SCC ストラテジー	83
4.2.5.1.1. RunAsUser	83
4.2.5.1.2. SELinuxContext	83
4.2.5.1.3. SupplementalGroups	83
4.2.5.1.4. FSGroup	83
4.2.5.2. ボリュームの制御	83
4.2.5.3. FlexVolume へのアクセスの制限	85
4.2.5.4. Seccomp	85
4.2.5.5. 受付 (Admission)	85
4.2.5.5.1. SCC の優先度設定	86
4.2.5.5.2. 事前に割り当てられた値および SCC (Security Context Constraints) について	87
4.2.6. 認証済みユーザーが実行できる内容の判別	88
4.3. 永続ストレージ	88
4.3.1. 概要	88
4.3.2. ボリュームおよび要求のライフサイクル	89
4.3.2.1. プロビジョニング	89
4.3.2.2. バインディング	89
4.3.2.3. 使用	89
4.3.2.4. Persistent Volume Claim (永続ボリューム要求) の保護	90
4.3.2.5. リリース	90
4.3.2.6. 回収	90
4.3.2.6.1. リサイクル	91
4.3.3. 永続ボリューム	92
4.3.3.1. 永続ボリュームのタイプ	92
4.3.3.2. 容量	93
4.3.3.3. アクセスモード	93
4.3.3.4. 回収ポリシー	95
4.3.3.5. フェーズ	95
4.3.3.6. マウントオプション	96
4.3.4. Persistent Volume Claim (永続ボリューム要求、PVC)	97
4.3.4.1. ストレージクラス	97
4.3.4.2. アクセスモード	97
4.3.4.3. リソース	97
4.3.4.4. ボリュームとしての要求	97
4.3.5. ブロックボリュームのサポート	98
4.4. ソースコントロール管理	101
4.5. 受付コントローラー	101
4.5.1. 概要	101
4.5.2. 一般的な受付ルール	102
4.5.3. カスタマイズ可能な受付プラグイン	103
4.5.4. コンテナを使用した受付コントローラー	103
4.6. カスタム受付コントローラー	103
4.6.1. 概要	103
4.6.2. 受付 Webhook	103
4.6.2.1. 受付 Webhook のタイプ	105
4.6.2.2. 受付 Webhook の作成	108
4.6.2.3. 受付 Webhook オブジェクトのサンプル	109
4.7. 他の API オブジェクト	110
4.7.1. LimitRange	110
4.7.2. ResourceQuota	110

4.7.3. リソース	110
4.7.4. シークレット	110
4.7.5. PersistentVolume	110
4.7.6. PersistentVolumeClaim	110
4.7.6.1. カスタムリソース	111
4.7.7. OAuth オブジェクト	111
4.7.7.1. OAuthClient	111
4.7.7.2. OAuthClientAuthorization	112
4.7.7.3. OAuthAuthorizeToken	112
4.7.7.4. OAuthAccessToken	113
4.7.8. ユーザーオブジェクト	114
4.7.8.1. アイデンティティ	114
4.7.8.2. ユーザー	115
4.7.8.3. UserIdentityMapping	116
4.7.8.4. グループ	116
第5章 ネットワーク	117
5.1. ネットワーク	117
5.1.1. 概要	117
5.1.2. OpenShift Container Platform DNS	117
5.2. OPENSIFT SDN	118
5.2.1. 概要	118
5.2.2. マスター上の設計	119
5.2.3. ノード上の設計	119
5.2.4. パケットフロー	120
5.2.5. ネットワーク分離	120
5.3. 利用可能な SDN プラグイン	121
5.3.1. OpenShift SDN	121
5.3.2. サードパーティの SDN プラグイン	121
5.3.2.1. Flannel SDN	121
5.3.2.2. Nuage SDN	122
5.4. 利用可能なルータープラグイン	126
5.4.1. デフォルトの HAProxy ルーター	126
5.4.2. HAProxy テンプレートルーター	126
5.4.3. F5 BIG-IP® ルータープラグイン	130
5.4.3.1. SDN 経由での Pod に対するトラフィックのルーティング	130
5.4.3.2. F5 統合の詳細	131
5.4.3.3. F5 ネイティブ統合	131
接続	131
データフロー: パケットから Pod へ	132
F5 ホストからのデータフロー	133
データフロー: トラフィックを F5 ホストに返す	133
5.5. ポート転送	134
5.5.1. 概要	134
5.5.2. サーバー操作	134
5.6. リモートコマンド	134
5.6.1. 概要	134
5.6.2. サーバー操作	134
5.7. ルート	135
5.7.1. 概要	135
5.7.2. ルーター	135
5.7.2.1. テンプレートルーター	136
5.7.3. 利用可能なルータープラグイン	137

5.7.4. スティックセッション	137
5.7.5. ルーターの環境変数	138
5.7.6. 負荷分散ストラテジー	143
5.7.7. HAProxy Strict SNI	144
5.7.8. ルーターの暗号スイート	144
5.7.9. ルートホスト名	145
5.7.10. ルートタイプ	146
5.7.10.1. パスベースのルート	146
5.7.10.2. セキュリティー保護されたルート	147
5.7.11. ルーターのシャード化	151
5.7.12. 他のバックエンドおよび重み	152
5.7.13. ルート固有のアノテーション	153
5.7.14. ルート固有の IP ホワイトリスト	155
5.7.15. ワイルドカードサブドメインポリシーを指定するルートの作成	156
5.7.16. ルートステータス	156
5.7.17. ルート内の特定ドメインの拒否または許可	156
5.7.18. Namespace 所有権チェックの無効化	158
第6章 サービスカタログコンポーネント	160
6.1. サービスカタログ	160
6.1.1. 概要	160
6.1.2. 設計	160
6.1.2.1. リソースの定義	161
6.1.3. 概念および用語	161
6.1.4. 提供されるクラスターサービスブローカー	164
6.2. テンプレートサービスブローカー	164
6.3. OPENSIFT ANSIBLE BROKER	164
6.3.1. 概要	164
6.3.2. Ansible Playbook Bundle	165

第1章 概要

OpenShift v3 は、基礎となる Docker 形式のコンテナイメージおよび Kubernetes 概念を可能な限り正確に表示することを目的としてレイヤー化されたシステムであり、開発者がアプリケーションを簡単に構成できるようにすることに重点が置かれています。たとえば、Ruby のインストール、コードのプッシュ、および MySQL の追加などを簡単に実行できます。

OpenShift v2 とは異なり、作成後の設定ではモデルのすべての側面においてより大きな柔軟性が確保されます。アプリケーションを別個のオブジェクトとみなす概念は、「サービス」を構成するという柔軟な概念に置き換えられ、2 つの Web コンテナでデータベースを再利用したり、データベースをネットワークエッジに直接公開したりできるようになりました。

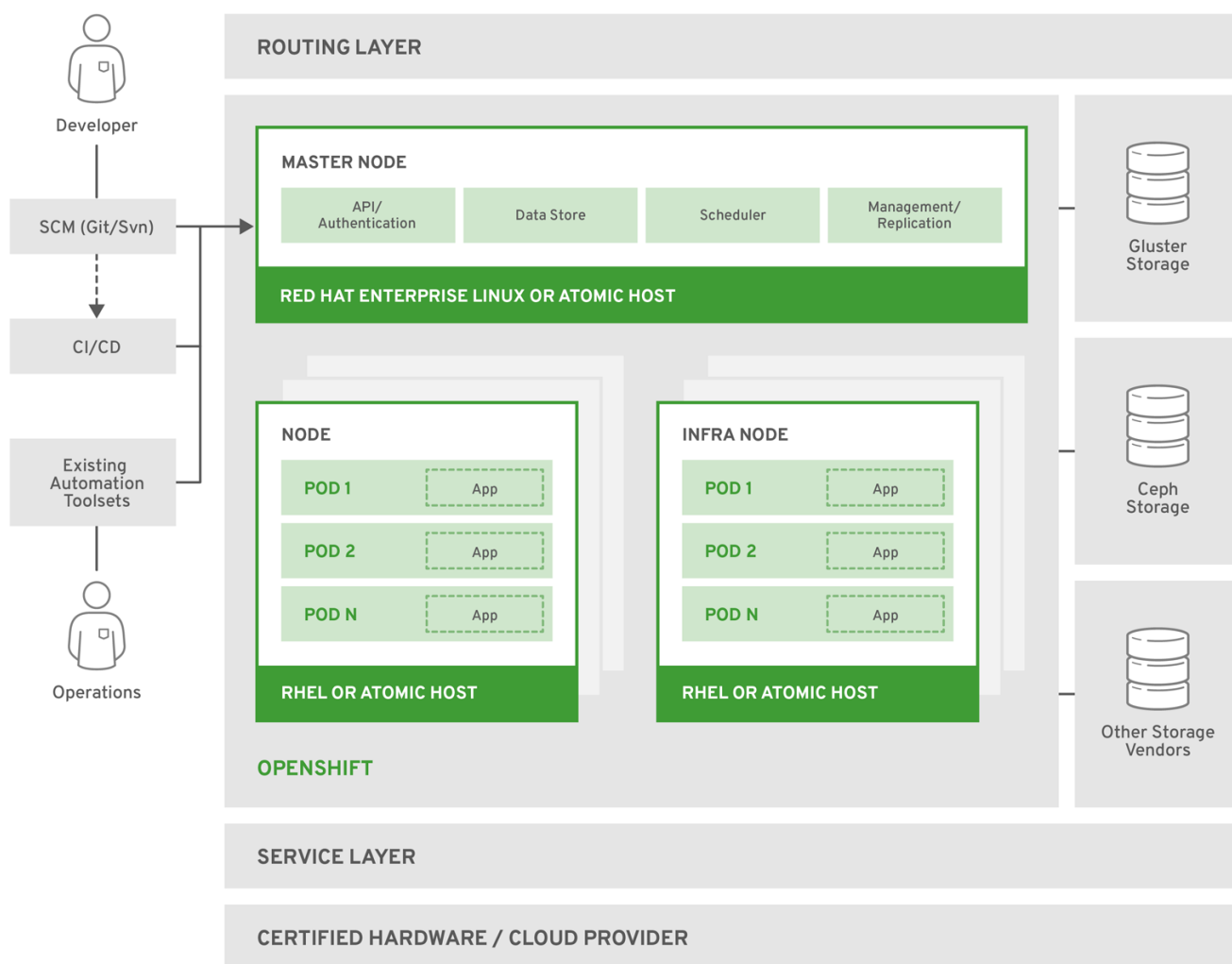
1.1. 各種の層について

Docker サービスは、Linux ベースの軽量なコンテナイメージのパッケージ化および作成のために抽象化を提供します。Kubernetes はクラスター管理を行い、複数のホストでコンテナをオーケストレーションします。

OpenShift Container Platform は以下の機能を追加します。

- ソースコードの管理、ビルド、およびデプロイメント (開発者向け)。
- システム全体で移行するイメージの大規模な管理およびプロモート
- 大規模なアプリケーション管理
- 大規模な開発者組織を編成するためのチームおよびユーザーの追跡
- クラスターをサポートするネットワークインフラストラクチャー

図1.1 OpenShift Container Platform アーキテクチャーの概要



OPENSIFT_415489_0218

1.2. OPENSIFT CONTAINER PLATFORM アーキテクチャーについて

OpenShift Container Platform のアーキテクチャーはマイクロサービスをベースとしており、分割された小規模なユニットが相互に連携します。これは [Kubernetes クラスター](#) の上部で実行され、オブジェクトについてのデータは信頼できるクラスター化されたキー値ストアの [etcd](#) に保存されます。これらのサービスは機能別に分類されます。

- **REST API:** コアオブジェクトをそれぞれ公開します。
- **コントローラー:** API を読み取り、変更を他のオブジェクトに適用してステータスを報告するか、またはオブジェクトに書き戻します。

ユーザーは REST API を呼び出して、システムの状態を変更します。コントローラーは REST API を使用してユーザーの必要な状態を読み取ってから、システムの他の部分の同期を試行します。たとえば、ユーザーが **ビルド** を要求する場合、「ビルド」オブジェクトを作成します。ビルドコントローラーは新規ビルドが作成されていることを確認し、そのビルドを実行するためにクラスターでプロセスを実行します。ビルドが完了すると、コントローラーは REST API でビルドオブジェクトを更新し、ユーザーはそれらのビルドが完了したことを確認できます。

コントローラーのパターンは、OpenShift Container Platform の多くの機能に拡張性があることを意味します。ビルドを実行し、起動する方法については、イメージの管理方法や [デプロイメント](#) が実行される方法とは切り離してカスタマイズできます。コントローラーはシステムの「ビジネスロジック」を

実行し、ユーザーのアクションを実行します。これらのコントローラーをカスタマイズするか、または独自のロジックに置き換えることにより、複数の異なる動作を実装できます。システム管理の視点では、これは API を使用し、繰り返されるスケジュールにおける共通の管理アクションをスクリプト化することも意味しています。これらのスクリプトは変更の有無を監視し、アクションを実行するコントローラーとしても機能します。OpenShift Container Platform では、このようにクラスターをカスタマイズする機能をファーストクラスの動作として使用できます。

この機能を有効にするため、コントローラーはシステムに対する変更の安定したストリームを利用し、システムのビューをユーザーの実行内容と同期します。このイベントストリームは、変更が発生するとすぐにそれらの変更を etcd から REST API にプッシュし、次にコントローラーにプッシュして、変更がシステム全体で非常に迅速かつ効率的に適用されるようにします。ただし、障害は常に発生する可能性があるため、コントローラーは起動時にシステムの最新の状態を取得でき、すべてが適切な状態にあることを確認できなければなりません。この再同期は、障害の発生時にオペレーターが影響を受けるコンポーネントを再起動でき、システムでダブルチェックしてから次に進むことができるので重要になります。コントローラーは常にシステムを同期するため、システムは最終的にはユーザーの意図に収束していきます。

1.3. OPENSIFT CONTAINER PLATFORM のセキュリティを保護する方法

OpenShift Container Platform および Kubernetes API は、認証情報を提示するユーザーの[認証](#)を行ってから、それらのロールに基づいてユーザーの[承認](#)を行います。開発者および管理者はどちらも数多くの方法で認証されますが、主に [OAuth トークン](#) および X.509 クライアント証明書が使用されます。OAuth トークンは JSON Web Algorithm **RS256** を使用して署名されます。これは SHA-256 を使用した RSA 署名アルゴリズムです。

開発者 (システムのクライアント) は通常、[クライアントプログラム](#) (oc など) を使用するか、またはブラウザを使用して [Web コンソール](#) に対して REST API 呼び出しを実行し、ほとんどの通信に OAuth ベアラートークンを使用します。インフラストラクチャーコンポーネント (ノードなど) は、システムで生成されるアイデンティティーが含まれるクライアント証明書を使用します。コンテナで実行されるインフラストラクチャーコンポーネントはそれらの[サービスアカウント](#)に関連付けられるトークンを使用して API に接続します。

承認は、「Pod の作成」または「サービスの一覧表示」などのアクションを定義し、それらをポリシードキュメントの各種ロールに分類する OpenShift Container Platform ポリシーエンジンで処理されます。ロールは、ユーザーまたはグループ ID によってユーザーまたはグループにバインドされます。ユーザーまたはサービスアカウントがアクションを試行すると、ポリシーエンジンはユーザーに割り当てられた 1 つ以上のロール (例: クラスター管理者または現行プロジェクトの管理者) をチェックし、その継続を許可します。

クラスターで実行されるすべてのコンテナはサービスアカウントに関連付けられるため、[シークレット](#)をそれらのサービスアカウントに関連付け、コンテナに自動的に配信することもできます。これにより、インフラストラクチャーでイメージ、ビルドおよびデプロイメントコンポーネントのプルおよびプッシュを行うためのシークレットを管理でき、アプリケーションコードでそれらのシークレットを簡単に利用することも可能になります。

1.3.1. TLS サポート

REST API とのすべての通信チャネル、および etcd および API サーバーなどの[マスターコンポーネント](#)間の通信のセキュリティは TLS で保護されます。TLS は、X.509 サーバー証明書およびパブリックキーインフラストラクチャーを使用して強力な暗号化、データの整合性、およびサーバーの認証を提供します。デフォルトで、新規の内部 PKI は OpenShift Container Platform のそれぞれのデプロイメントについて作成されます。内部 PKI は 2048 ビット RSA キーおよび SHA-256 署名を使用します。パブリックホストの[カスタム証明書](#)もサポートされます。

OpenShift Container Platform は [crypto/tls](#) の Golang 標準ライブラリーの実装を使用し、外部の暗号および TLS ライブラリーには依存しません。さらに、クライアントは GSSAPI 認証および OpenPGP 署名の外部ライブラリーに依存します。GSSAPI は通常 OpenSSL の libcrypto を使用する MIT Kerberos または Heimdal Kerberos のいずれかによって提供されます。OpenPGP 署名の検証は libpgpme および GnuPG によって処理されます。

非セキュアなバージョンである SSL 2.0 および SSL 3.0 はサポート対象外であり、これを利用することはできません。OpenShift Container Platform サーバーおよび **oc** クライアントはデフォルトで TLS 1.2 のみを提供します。TLS 1.0 および TLS 1.1 はサーバー設定で有効にできます。サーバーおよびクライアントはどちらも、認証される暗号化アルゴリズムと Perfect Forward Secrecy を使用する最新の暗号スイートを優先的に使用します。RC4、3DES、および MD5 などの非推奨かつ非セキュアなアルゴリズムを使用する暗号スイートは無効にされます。一部の内部クライアント (LDAP 認証など) には TLS 1.0 から 1.2 への設定についての制限が少なく、より多くの暗号スイートが有効にされます。

表1.1 サポートされる TLS バージョン

TLS バージョン	OpenShift Container Platform サーバー	oc クライアント	他のクライアント
SSL 2.0	非対応	非対応	非対応
SSL 3.0	非対応	非対応	非対応
TLS 1.0	No [a]	No [a]	Maybe (可能性あり) [b]
TLS 1.1	No [a]	No [a]	Maybe (可能性あり) [b]
TLS 1.2	Yes	Yes	Yes
TLS 1.3	N/A [c]	N/A [c]	N/A [c]
<p>[a] デフォルトで無効にされますが、サーバー設定で有効にできます。</p> <p>[b] LDAP クライアントなどの一部の内部クライアントです。</p> <p>[c] TLS 1.3 は開発中です。</p>			

以下は OpenShift Container Platform のサーバーの有効にされた暗号スイートの一覧であり、**oc** クライアントは優先される順序で並べ替えられます。

- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA

第2章 インフラストラクチャーコンポーネント

2.1. KUBERNETES インフラストラクチャー

2.1.1. 概要

OpenShift Container Platform 内で、Kubernetes はコンテナまたはホストのセット全体でコンテナ化されたアプリケーションを管理し、デプロイメント、メンテナンス、およびアプリケーションのスケリングのメカニズムを提供します。Docker サービスはコンテナ化されたアプリケーションをパッケージ化し、インスタンス化し、これを実行します。Kubernetes クラスタは 1 つ以上のマスターおよびノードセットで構成されます。

また、オプションとして[高可用性](#) (HA) のマスターを設定し、クラスタから単一障害点がなくなるようにすることもできます。



注記

OpenShift Container Platform は Kubernetes 1.9 および Docker 1.13 を使用します。

2.1.2. マスター

マスターは、API サーバー、コントローラーマネージャーサーバー、および etcd などのマスターコンポーネントが含まれるホストです。マスターはその Kubernetes クラスタで [ノード](#) を管理し、[Pod](#) がノードで実行されるようスケジュールします。

表2.1 マスターコンポーネント

コンポーネント	説明
API サーバー	Kubernetes API サーバーは Pod、サービスおよびレプリケーションコントローラーのデータを検証し、設定します。さらに Pod をノードに割り当て、Pod の情報をサービス設定に同期します。これはスタンドアロンプロセスとして実行できます。
etcd	etcd は永続的なマスターの状態を保存し、他のコンポーネントは etcd で特定の状態にするために必要な変更の有無について確認します。etcd については、通常は 2n+1 のピアサービスでデプロイされるなど、オプションで高可用性の設定を行うことができます。
コントローラーマネージャーサーバー	コントローラーマネージャーサーバーは etcd でレプリケーションコントローラーオブジェクトの変更を確認し、API を使用して必要な状態を実行します。これはスタンドアロンプロセスとして実行できます。この複数のプロセスでは、一度に 1 つのアクティブなリーダーを使用してクラスタを作成します。
HAProxy	オプションです。 高可用マスター を native メソッドで設定し、API マスターエンドポイント間の負荷分散を行う際に使用されます。 通常インストール (advanced installation) 方式 では、 native メソッドで HAProxy を設定できます。または、 native メソッドを使う場合でも、選択する独自のロードバランサーを事前に設定することができます。

2.1.2.1. 高可用性マスター

単一マスター設定の使用時に、マスターまたはそのサービスのいずれかが失敗しても、実行中のアプリケーションの可用性はそのまま維持されます。ただし、マスターサービスが失敗すると、システムのアプリケーションの失敗に対応する機能、または新規アプリケーションの作成に対応する機能に制限が生じます。オプションで高可用性 (HA) のマスターを、クラスターに単一障害点がなくなるように設定できます。

マスターの可用性についての懸念を軽減するために、2 つのアクティビティを実行することが推奨されます。

1. **runbook** エントリは、マスターの再作成のために作成される必要があります。runbook エントリはすべての高可用サービスに必要なバックストップです。追加ソリューションは runbook が参照される頻度を制御するのみになります。たとえば、マスターホストのコールドスタンバイは、新規アプリケーションの作成または失敗したアプリケーションコンポーネントの回復によるダウンタイムが数分未満とすることを要求する SLA を十分に満たすことができます。
2. 高可用性ソリューションを使用して、クラスターから単一障害点がなくなるようにマスターを設定します。**通常インストール (advanced installation) 方式**は **native** HA メソッドを使用し、HAProxy を設定して特定のサンプルを提供します。また、HAProxy の代わりに **native** メソッドを使用して同様の概念を採用し、それらの概念を既存の HA ソリューションに適用することもできます。



注記

インストール後の単一マスタークラスターから複数マスターへの移行はサポートされていません。

native HA メソッドを HAProxy で使用する際に、マスターコンポーネントには以下の可用性が設定されます。

表2.2 HAProxy による可用性マトリックス

ロール	スタイル	備考
etcd	Active-active	負荷分散機能と完全な冗長性のあるデプロイメントです。別個のホストにインストールすることも、マスターホストに併置することもできます。
API サーバー	Active-active	HAProxy で管理されます。
コントローラーマネージャーサーバー	Active-passive	一度に 1 つのインスタンスがクラスターリーダーとして選択されます。
HAProxy	Active-passive	API マスターエンドポイント間で負荷を分散します。

クラスター化された etcd ではクォーラム(定足数)を維持するためにホストの数を奇数にする必要がありますが、マスターサービスにはクォーラム(定足数) やホストの数を奇数にしなければならないという要件はありません。ただし、2 つ以上のマスターサービスが HA 用に必要になるため、マスターサービスと etcd を同じ場所に配置する場合には、一律に奇数のホストを維持することが通例になります。

2.1.3. ノード

ノードはコンテナのランタイム環境を提供します。Kubernetes クラスターの各ノードには、[マスター](#)で管理される必要なサービスがあります。また、ノードには Docker サービス、[kubelet](#) および [サービスプロキシ](#) を含む Pod を実行するための必要なサービスも含まれます。

OpenShift Container Platform は、ノードをクラウドプロバイダー、物理システムまたは仮想システムから作成します。Kubernetes は、それらのノードの表現である [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からの情報を使用し、ヘルスチェックでノードを検証します。ノードはヘルスチェックをパスするまで無視され、マスターはノードが有効になるまでチェックを続けます。[Kubernetes ドキュメント](#)にはノード管理についての詳細が記載されています。

管理者は CLI を使用して OpenShift Container Platform インスタンスで [ノードを管理](#) できます。ノードサーバーの起動時に完全な設定およびセキュリティーオプションを定義するには、[専用のノード設定ファイル](#)を使用します。



重要

推奨されるノードの最大数については、「[クラスターの制限](#)」セクションを参照してください。

2.1.3.1. Kubelet

各ノードには、Pod を記述する YAML ファイルのコンテナマニフェストで指定されるようにノードを更新する kubelet があります。kubelet はマニフェストのセットを使用して、そのコンテナが起動していること、および実行を継続することを確認します。

コンテナマニフェストは以下によって kubelet に提供されます。

- 20 秒ごとにチェックされるコマンドラインのファイルパス。
- 20 秒ごとにチェックされるコマンドラインで渡される HTTP エンドポイント。
- `/registry/hosts/$(hostname -f)` などの etcd サーバーを監視し、変更を実行する kubelet。
- HTTP をリッスンし、単純な API に応答して新規マニフェストを送信する kubelet。

2.1.3.2. サービスプロキシ

各ノードは、ノードの API で定義されるサービスを反映した単純なネットワークプロキシも実行します。これにより、ノードは一連のバックエンドで単純な TCP および UDP ストリーム転送を実行できます。

2.1.3.3. ノードオブジェクト定義

以下は、Kubernetes のノードオブジェクト定義の例になります。

```
apiVersion: v1 ①
kind: Node ②
metadata:
  creationTimestamp: null
  labels: ③
    kubernetes.io/hostname: node1.example.com
  name: node1.example.com ④
```

```
spec:
  externalID: node1.example.com 5
status:
  nodeInfo:
    bootID: ""
    containerRuntimeVersion: ""
    kernelVersion: ""
    kubeProxyVersion: ""
    kubeletVersion: ""
    machineID: ""
    osImage: ""
    systemUUID: ""
```

- 1 **apiVersion** は使用する API バージョンを定義します。
- 2 **Node** に設定された **kind** はこれをノードオブジェクトの定義として特定します。
- 3 **metadata.labels** は、ノードに追加されているラベルを一覧表示します。
- 4 **metadata.name** はノードオブジェクトの名前を定義する必須の値です。この値は、**oc get nodes** コマンドの実行時に **NAME** 列に表示されます。
- 5 **spec.externalID** は、ノードに到達できる完全修飾ドメイン名です。空の場合、デフォルトは **metadata.name** 値に設定されます。

2.2. CONTAINER レジストリー

2.2.1. 概要

OpenShift Container Platform は、Docker Hub、サードパーティーによって実行されるプライベートレジストリーおよび統合 OpenShift Container Platform レジストリーを含む、イメージのソースとして Docker レジストリー API を実装するすべてのサーバーを利用できます。

2.2.2. 統合 OpenShift Container レジストリー

OpenShift Container Platform は **OpenShift Container レジストリー (OCR)** という統合コンテナレジストリーを提供します。これは、新規イメージリポジトリをオンデマンドで自動的にプロビジョニングする機能を追加します。これにより、作成されるイメージをプッシュするためのアプリケーションビルドのビルトインロケーションがユーザーに提供されます。

新規イメージが OCR にプッシュされるたびに、レジストリーは OpenShift Container Platform に新規イメージについて通知し、namespace、名前、およびイメージメタデータなどの関連するすべての情報を伝えます。OpenShift Container Platform の異なる部分が新規イメージに対応し、新規ビルドおよびデプロイメントを作成します。

また OCR はビルドおよびデプロイメントの統合なしに、単独でコンテナレジストリーとして機能するスタンドアロンコンポーネントとしてデプロイできます。詳細については、「[OpenShift Container レジストリーのスタンドアロンデプロイメントのインストール](#)」を参照してください。

2.2.3. サードパーティーレジストリー

OpenShift Container Platform はサードパーティーのイメージを使用してコンテナを作成できますが、これらのレジストリーは OpenShift Container Platform レジストリーと同じイメージ通知サポートを提供する訳ではありません。取得されたタグの更新は、**oc import-image <stream>** を実行す

ると同様に単純です。新規イメージが検出されると、事前に記述されたビルドおよびデプロイメントの応答が発生します。

2.2.3.1. 認証

OpenShift Container Platform は、ユーザーが指定する認証情報を使ってプライベートリポジトリにアクセスするためにレジストリーと通信します。これにより、OpenShift はプライベートリポジトリから/へのイメージのプッシュ/プルを行うことができます。「[認証](#)」のトピックには詳細が記載されています。

2.3. WEB コンソール

2.3.1. 概要

OpenShift Container Platform Web コンソールは、Web ブラウザーからアクセスできるユーザーインターフェースです。開発者は Web コンソールを使用して [プロジェクト](#) のコンテンツの可視化、ブラウズ、および管理を実行できます。



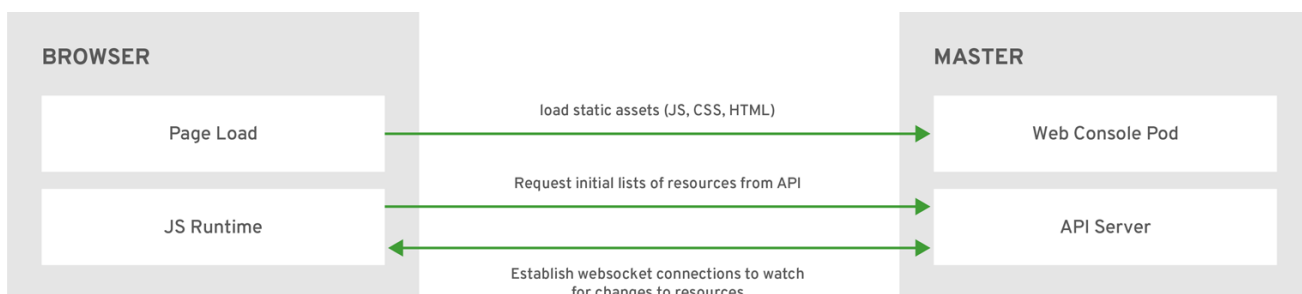
注記

Web コンソールを使用するには JavaScript が有効にされている必要があります。[WebSocket](#) をサポートする Web ブラウザーを使用することが最も推奨されます。

Web コンソールは [マスター](#) 上の Pod として実行されます。Web コンソールを実行するために必要な静的なアセットは Pod によって提供されます。また、管理者は拡張を使用して [Web コンソールのカスタマイズ](#) を実行できます。これにより、Web コンソールによる読み込みの実行時にスクリプトを実行でき、カスタムスタイルシートを読み込むことができます。

ブラウザーから Web コンソールにアクセスする際に、まず必要な静的アセットをすべて読み込みます。次に、**openshift start** オプションの **--public-master** で定義される値、**openshift-web-console** namespace で定義される **webconsole-config** Config Map の関連パラメーター **masterPublicURL** で定義される値を使用して、OpenShift Container Platform API に要求を行います。Web コンソールは WebSocket を使用して API サーバーとの継続的な接続を維持し、更新情報を利用可能になる時点で受信します。

図2.1 Web コンソール要求アーキテクチャー



OPENSIFT_415489_0618

Web コンソールの設定されたホスト名および IP アドレスは、ブラウザーが要求を [クロスオリジン](#) の要求とみなす場合でも API サーバーに安全にアクセスできるようにホワイトリスト化されます。異なるホスト名を使用して Web アプリケーションから API サーバーにアクセスするには、**openshift start** で **--cors-allowed-origins** オプションを指定するか、または関連する [マスター設定ファイルパラメーター](#) の **corsAllowedOrigins** で指定してホスト名をホワイトリスト化する必要があります。

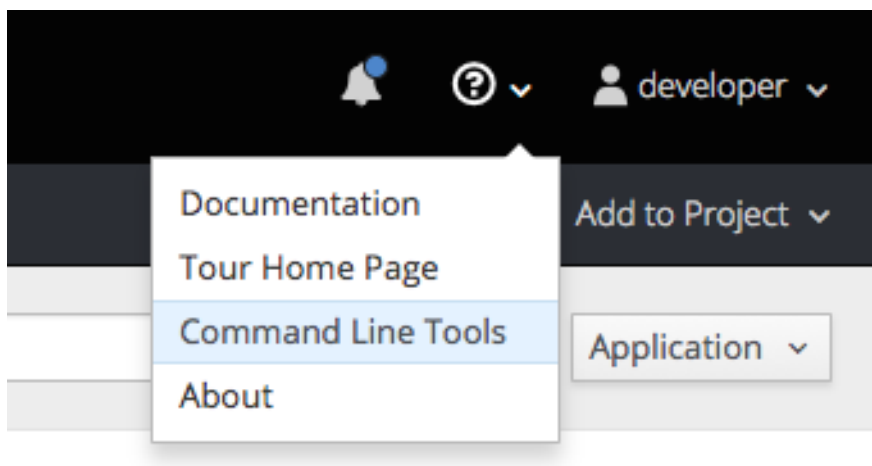
corsAllowedOrigins パラメーターは設定フィールドで制御されます。値に対してピニングやエスケープは実行されません。以下は、ホスト名を固定し、ドットをエスケープする方法の例を示しています。

```
corsAllowedOrigins:  
- (?i)//my\.subdomain\.domain\.com(:|\z)
```

- **(?i)** は大文字/小文字を区別します。
- **//** はドメインの開始部分に固定されます (また、**http:** または **https:** の後のダブルスラッシュに一致します)。
- **\.** はドメイン名のドットをエスケープします。
- **(:|\z)** はドメイン名 **(\z)** またはポートセパレーター **(:)** の終了部分に一致します。

2.3.2. CLI ダウンロード

Web コンソールのヘルプアイコンから CLI ダウンロードにアクセスできます。



クラスター管理者は、[これらのリンクの追加のカスタマイズ](#)を実行できます。

Command Line Tools

With the OpenShift command line interface (CLI), you can create applications and manage OpenShift projects from a terminal. You can download the `oc` client tool using the links below. For more information about downloading and installing it, please refer to the [Get Started with the CLI](#) documentation.

Download `oc` :

[Latest Release](#) 

After downloading and installing it, you can start by logging in. You are currently logged into this console as **developer**. If you want to log into the CLI using the same session token:

```
oc login https://127.0.0.1:8443 --token=<hidden>
```



A token is a form of a password. Do not share your API token. To reveal your token, press the copy to clipboard button and then paste the clipboard contents.

After you login to your account you will get a list of projects that you can switch between:

```
oc project <project-name>
```

If you do not have any existing projects, you can create one:

```
oc new-project <project-name>
```

To show a high level overview of the current project:

```
oc status
```

For other information about the command line tools, check the [CLI Reference](#) and [Basic CLI Operations](#).

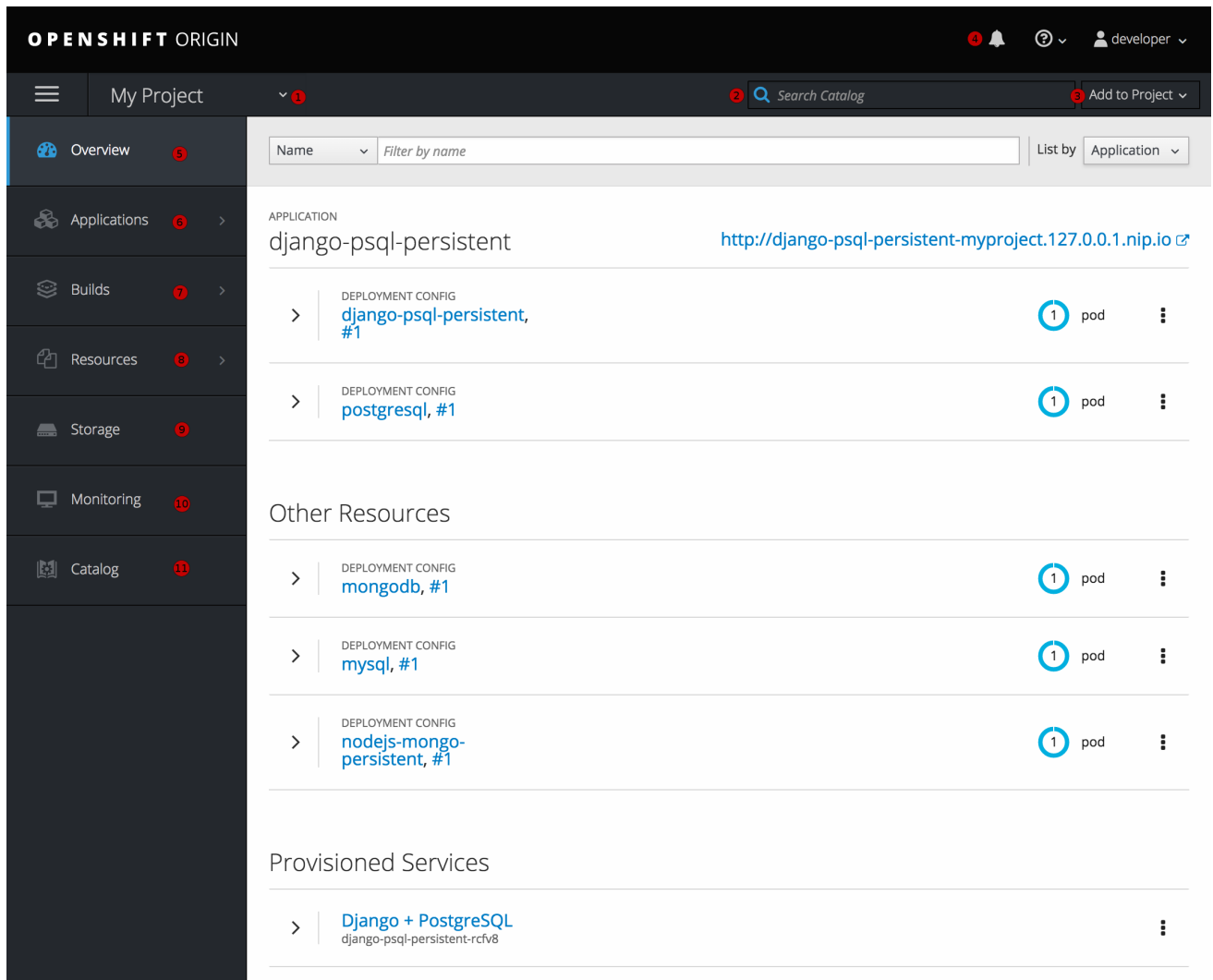
2.3.3. ブラウザーの要件

OpenShift Container Platform の[テスト済みの統合](#)を確認します。

2.3.4. プロジェクトの概要

[ログイン](#)後、Web コンソールは開発者に現在選択されている[プロジェクト](#)の概要を提供します。

図2.2 Web コンソールのプロジェクト概要



プロジェクトセクターを使うと、アクセスできる **プロジェクト間の切り替え** を実行できます。

プロジェクトビュー内でサービスをすぐに見つけられるように検索条件を入力します。

ソースリポジトリ を使用するか、またはサービスカタログのサービスを使用して新規アプリケーションを作成します。

プロジェクトに関連する通知です。

Overview タブ (現在選択されている): 各コンポーネントのハイレベルビューと共にプロジェクトのコンテンツを可視化します。

Applications タブ: デプロイメント、Pod、サービスおよびルート参照し、これらに対してアクションを実行できます。

Builds タブ: ビルドおよびイメージストリーム参照し、これらに対してアクションを実行できます。

Resources タブ: 現在のクォータの消費およびその他のリソースを表示します。

Storage タブ: Persistent Volume Claim (永続ボリューム要求) を表示し、アプリケーションのストレージを要求します。

Monitoring タブ: ビルド、Pod、デプロイメントのログ、およびプロジェクトのすべてのオブジェクトについてのイベント通知を表示します。

Catalog タブ: プロジェクト内でカタログにすぐに移動できます。



注記

Cockpit は OpenShift Container Platform 3.1 以降に自動的にインストールされ、有効にされています。これは後に開発環境をモニターするのに役立ちます。『[Red Hat Enterprise Linux Atomic Host: Getting Started with Cockpit](#)』は Cockpit の使用についての詳細を記載しています。






2.3.5. JVM コンソール

Java イメージをベースとする Pod の場合、Web コンソールは関連する統合コンポーネントを表示し、管理するための [hawt.io](#) ベースの JVM コンソールへのアクセスを公開します。コンテナに **jolokia** という名前のポートがある場合、**Connect** リンクが **Browse → Pods** ページの Pod の詳細に表示されます。

図2.3 JVM コンソールへのリンクを持つ Pod

Template

CONTAINER: STI-BUILD

-  **Image:** openshift/origin-sti-builder:latest
-  **Mount:** docker-socket → /var/run/docker.sock
-  **Mount:** builder-dockercfg-p7gmj-push → /var/run/secrets/openshift.io/push
-  **Mount:** builder-token-t6b9i → /var/run/secrets/kubernetes.io/serviceaccount
-  [Open Java Console](#)

Volumes

docker-socket

Type: host path (bare host directory volume)
Path: /var/run/docker.sock

JVM コンソールへの接続後に、接続されている Pod の関連コンポーネントに応じて異なるページが表示されます。

図2.4 JVM コンソール

Connected to quickstart-java-camel-spring-container

Back

JMX

Threads

Camel

Total: 9 Runnable: 3 Timed waiting: 2 Waiting: 4

Filter...

ID	State	Name	Waited Time	Blocked Time	Native	Suspended
15		Thread-5	1 hour			
14		Camel (camel-1) thread #0 - file://src/data	1 hour			
9		Jolokia Agent Cleanup Thread				
8		Thread-3		279 ms	(in native)	
6		server-timer	1 hour			
4		Signal Dispatcher				
3		Finalizer	1 hour			
2		Reference Handler	1 hour	10 ms		
1		main				

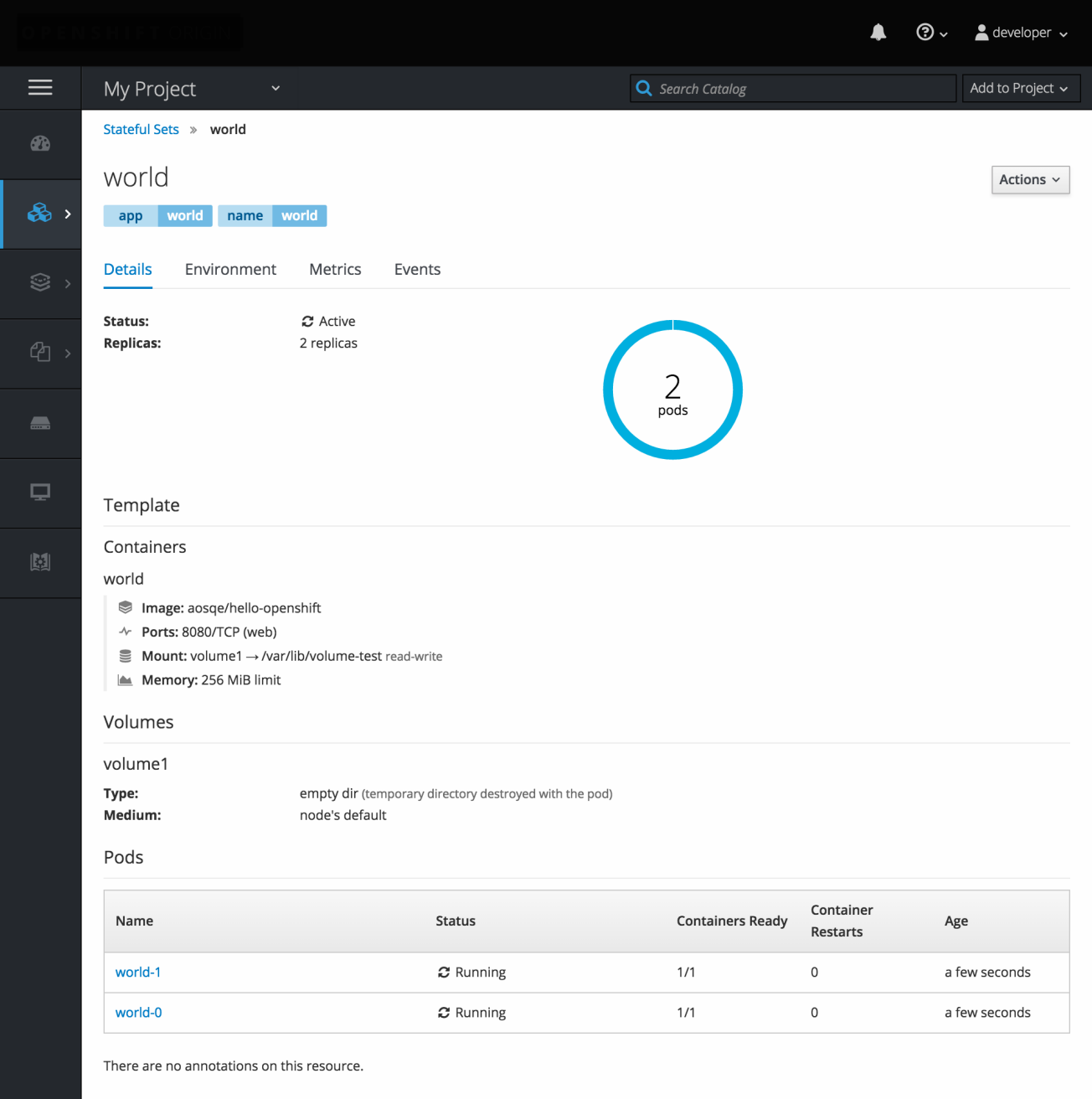
以下のページが利用可能になります。

ページ	説明
JMX	JMX ドメインおよび MBean を表示し、管理します。
スレッド	スレッドの状態を表示し、モニターします。
ActiveMQ	Apache ActiveMQ ブローカーを表示し、管理します。
Camel	Apache Camel のルートおよび依存関係を表示し、管理します。
OSGi	JBoss Fuse OSGi 環境を表示し、管理します。

2.3.6. StatefulSet

StatefulSet コントローラーは Pod の一意のアイデンティティを提供し、デプロイメントおよびスケーリングの順序を決定します。**StatefulSet** の使用は一意のネットワーク ID、永続ストレージ、正常なデプロイメントおよびスケーリング、および正常な削除および終了に役立ちます。

図2.5 OpenShift Container Platform の StatefulSet



第3章 コアとなる概念

3.1. 概要

以下のトピックでは、OpenShift Container Platform の使用時に使われるコアとなる概念およびオブジェクトについてのハイレベルのアーキテクチャ情報を提供します。これらのオブジェクトの多くは Kubernetes をベースとしており、さらに機能が充実した開発ライフサイクルプラットフォームを提供するために OpenShift Container Platform によって拡張されています。

- **コンテナ**および**イメージ**は、アプリケーションをデプロイするための構成要素です。
- **Pod** および**サービス**は、コンテナの相互通信およびプロキシ接続を可能にします。
- **プロジェクト**および**ユーザー**は、コミュニティがコンテンツを共同で編成し、管理するためのスペースと手段を提供します。
- **ビルド**および**イメージストリーム**は、有効なイメージのビルドおよび新規イメージへの対応を可能にします。
- **デプロイメント**は、ソフトウェア開発およびデプロイメントライフサイクルの拡張したサポートを追加します。
- **ルート**はサービスを一般に公開します。
- **テンプレート**は、カスタマイズされたパラメーターに基づく数多くのオブジェクトの同時作成を可能にします。

3.2. コンテナおよびイメージ

3.2.1. コンテナ

OpenShift Container Platform アプリケーションの基本的な単位は **コンテナ** と呼ばれています。**Linux コンテナテクノロジー**は、プロセスを指定されたリソースとの対話に制限するために実行中のプロセスを分離する軽量なメカニズムです。

数多くのアプリケーションインスタンスは、お互いのプロセス、ファイル、ネットワークなどが表示されない状態で単一ホストのコンテナで実行される場合があります。コンテナは任意のワークロードに使用されますが、通常、それぞれのコンテナは Web サーバーまたはデータベースなどの (「マイクロサービス」と呼ばれることの多い) 単一サービスを提供します。

Linux カーネルは数年にわたりコンテナテクノロジーの各種機能を統合してきました。最近では、Docker プロジェクトはホスト上の Linux コンテナの便利な管理インターフェースを開発しました。OpenShift Container Platform および Kubernetes は、複数ホストのインストールで Docker 形式のコンテナをオーケストレーションする機能を追加しています。

OpenShift Container Platform の使用時に Docker CLI またはサービスとの直接的な対話は発生しないものの、それらの機能および用語を理解しておくことは、OpenShift Container Platform におけるそれらの役割やアプリケーションのコンテナ内での機能を理解する上で重要です。**docker** RPM は RHEL 7、CentOS および Fedora の一部として利用できるため、これを OpenShift Container Platform と切り離して実験的に使用することができます。ガイドとなる概要情報については、『[Get Started with Docker Formatted Container Images on Red Hat Systems](#)』という記事を参照してください。

3.2.1.1. Init コンテナ

Pod にはアプリケーションコンテナのほかに init コンテナを含めることができます。Init コンテナにより、セットアップスクリプトやバインディングコードを再編成できます。init コンテナは、完了するまで常に行われる点で通常のコンテナとは異なります。各 init コンテナは次のコンテナが起動する前に正常に完了している必要があります。

詳細については、「[Pod およびサービス](#)」を参照してください。

3.2.2. イメージ

OpenShift Container Platform のコンテナは Docker 形式のコンテナ イメージ をベースにしています。イメージは、単一コンテナを実行するためのすべての要件、およびそのニーズおよび機能を記述するメタデータを含むバイナリーです。

イメージについては、パッケージ化テクノロジーのコンテキストで考えることができます。コンテナには、作成時にコンテナに追加のアクセスを付与しない限り、イメージで定義されるリソースにのみアクセスできます。同じイメージを複数のホスト間の複数コンテナにデプロイし、それらの間で負荷を分散することにより、OpenShift Container Platform はイメージにパッケージ化されたサービスの冗長性および水平方向のスケーリングを提供できます。

Docker CLI を直接使用してイメージをビルドすることができますが、OpenShift Container Platform はコードおよび設定を既存イメージに追加して新規イメージの作成を支援するビルダーイメージも提供しています。

アプリケーションは一定の期間にわたって開発されるため、単一のイメージ名が実際には「同一」イメージの数多くの異なるバージョンを参照する場合があります。それぞれの異なるイメージは、通常は 12 文字 (例: **fd44297e2ddb**) に省略されるハッシュ (**fd44297e2ddb050ec4f...** などの長い 16 進数) で一意に参照されます。

イメージバージョンタグポリシー

Docker サービスは、必要なイメージを指定するために、イメージ名に加えて、バージョン番号ではなくタグ (**v1**、**v2.1**、**GA**、またはデフォルト **latest**) の適用を可能にします。そのため、同じイメージが **centos** (これは **latest** タグを示します)、**centos:centos7**、または **fd44297e2ddb** などとして参照されます。



警告

公式の OpenShift Container Platform イメージには **latest** タグを使用しないでください。これらは **openshift3/** 以降のイメージであり、**latest** は **3.4**、または **3.5** などの数多くのバージョンを参照する可能性があります。

イメージへのタグの付け方によって更新ポリシーが決定されます。より具体的なタグを使用すると、イメージが更新される頻度は低くなります。以下を使用して選択されている OpenShift Container Platform イメージポリシーを判別してください。

vX.Y

vX.Y タグは X.Y.Z-<number> を参照します。たとえば、**registry-console** イメージが v3.4 に更新されると、これは最新の 3.4.Z-<number> タグを参照します (例: 3.4.1-8)。

X.Y.Z

上記の vX.Y サンプルと同様です。X.Y.Z タグは最新の X.Y.Z-<number> を参照します。たとえば、3.4.1 は 3.4.1-8 を参照します。

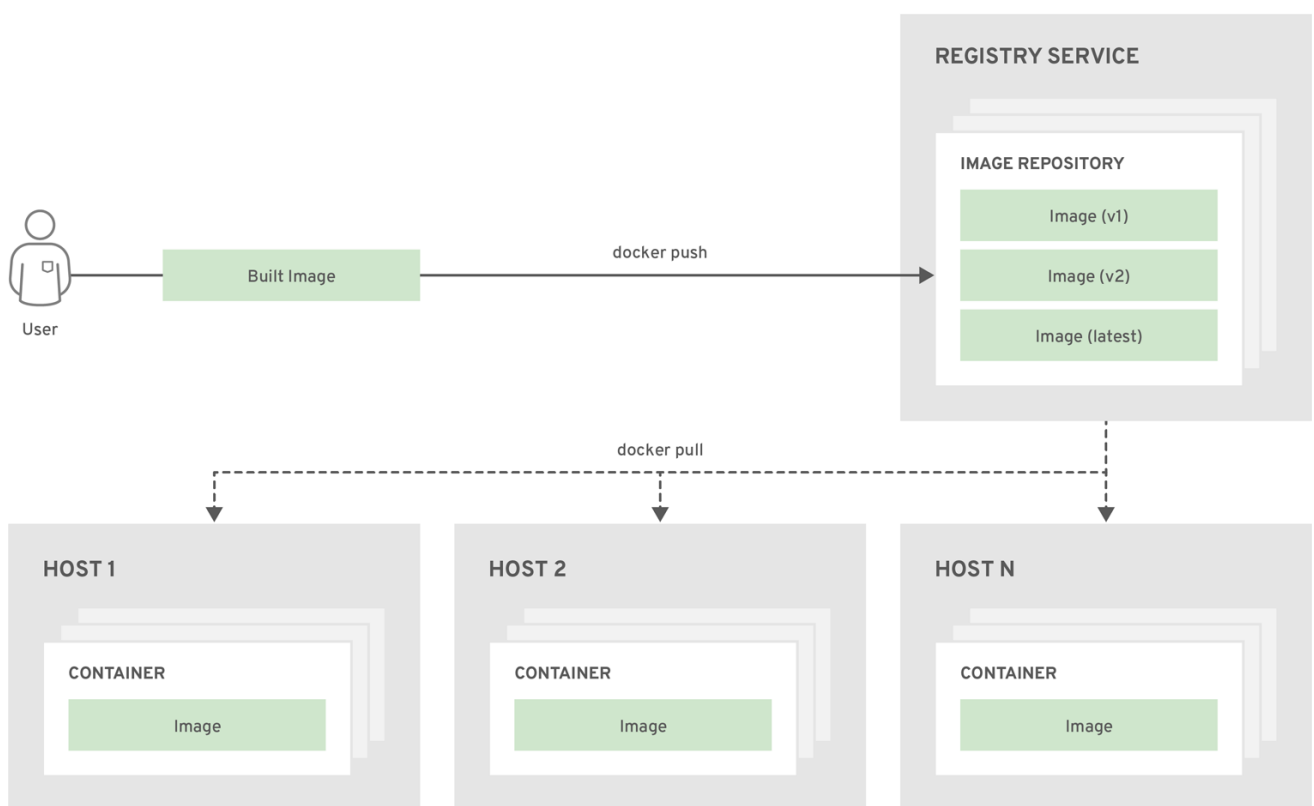
X.Y.Z-<number>

タグは一意であり、変更されません。このタグを使用する際、イメージが更新される際にイメージはタグを更新しません。たとえば、イメージが更新される場合でも、3.4.1-8 は 3.4.1-8 を常に参照します。

3.2.3. コンテナレジストリー

コンテナレジストリーは Docker 形式のコンテナイメージの保存および取得を行うサービスです。レジストリーには、1 つ以上のイメージリポジトリのコレクションが含まれます。各イメージリポジトリには 1 つ以上のタグ付けされたイメージが含まれます。Docker は独自のレジストリーである [Docker Hub](#) を提供しますが、プライベートまたはサードパーティーのレジストリーを使用することもできます。Red Hat はサブスクリプションをお持ちのお客様に対し、[registry.access.redhat.com](#) にてレジストリーを提供しています。OpenShift Container Platform はカスタムコンテナイメージを管理するための独自の内部レジストリーも提供します。

以下の図には、コンテナ、イメージ、およびレジストリー間の関係が示されています。



OPENSIFT_415489_0218

3.3. POD およびサービス

3.3.1. Pod

OpenShift Container Platform は **Pod** という Kubernetes の概念を使用しています。これはホスト上に同時にデプロイされる 1 つ以上の **コンテナ** であり、定義され、デプロイされ、管理される最小のコンピュータ単位です。

Pod はコンテナに対するマシンインスタンス (物理または仮想) とほぼ同等です。各 Pod には独自の内部 IP アドレスが割り当てられるため、各 Pod はそのポート領域全体を所有し、Pod 内のコンテナはそれらのローカルストレージおよびネットワークを共有できます。

Pod にはライフサイクルがあります。Pod が定義されると、ノードで実行されるように割り当てられ、

コンテナが終了するまで実行されるか、またはその他の理由でコンテナが削除されるまで実行されます。ポリシーおよび終了コードによっては、Pod は終了後に削除されるか、またはコンテナのログへのアクセスを有効にするために保持される可能性があります。

ほとんどの場合、OpenShift Container Platform は Pod を変更不可能なものとして処理します。Pod が実行中の場合、Pod に変更を加えることはできません。OpenShift Container Platform が変更を実装する場合、まず既存の Pod を終了してから、これを変更された設定またはベースイメージのいずれかまたはその両方で再作成します。Pod は拡張可能なものとして処理されますが、再作成時に状態を維持しません。そのため、通常 Pod はユーザーから直接管理されるのではなく、ハイレベルの [コントローラー](#) で管理される必要があります。



注記

OpenShift Container Platform ノードホストごとの Pod の最大数については、「[クラスターの制限](#)」を参照してください。



警告

[レプリケーションコントローラー](#)によって管理されないベア Pod はノードの中断時に再スケジュールされません。

以下は Pod のサンプル定義です。これは、統合コンテナレジストリーという OpenShift Container Platform インフラストラクチャーの一部で、長期間実行されるサービスを提供します。これは数多くの Pod の機能を示していますが、それらのほとんどは他のトピックで説明されるため、ここではこれらについて簡単に説明します。

例3.1 Pod オブジェクト定義 (YAML)

```
apiVersion: v1
kind: Pod
metadata:
  annotations: { ... }
  labels:
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
  generateName: docker-registry-1-
spec:
  containers:
    - env:
        - name: OPENSIFT_CA_DATA
          value: ...
        - name: OPENSIFT_CERT_DATA
          value: ...
        - name: OPENSIFT_INSECURE
          value: "false"
        - name: OPENSIFT_KEY_DATA
          value: ...
        - name: OPENSIFT_MASTER
          value: https://master.example.com:8443
```

```

image: openshift/origin-docker-registry:v0.6.2 ⑤
imagePullPolicy: IfNotPresent
name: registry
ports: ⑥
- containerPort: 5000
  protocol: TCP
resources: {}
securityContext: { ... } ⑦
volumeMounts: ⑧
- mountPath: /registry
  name: registry-storage
- mountPath: /var/run/secrets/kubernetes.io/serviceaccount
  name: default-token-br6yz
  readOnly: true
dnsPolicy: ClusterFirst
imagePullSecrets:
- name: default-dockercfg-at06w
restartPolicy: Always ⑨
serviceAccount: default ⑩
volumes: ⑪
- emptyDir: {}
  name: registry-storage
- name: default-token-br6yz
secret:
  secretName: default-token-br6yz

```

- ① Pod は 1 つ以上のラベルで「タグ付け」されます。これは単一操作で Pod のグループを選択したり、管理したりするために使用できます。これらのラベルはメタデータ ハッシュにキー/値形式で保存されます。この例で使用されている 1 つのラベルは **docker-registry=default** です。
- ② Pod はそれらの namespace 内で一意の名前を持つ必要があります。Pod 定義は **generateName** 属性で名前のベースを指定できますが、一意の名前を生成するためにランダムな文字が自動的に追加されます。
- ③ **containers** はコンテナ定義の配列を指定します。この場合 (ほとんどの場合)、1 つのみが指定されます。
- ④ 必要な値を各コンテナに渡すために、環境変数を指定することができます。
- ⑤ Pod の各コンテナは独自の Docker 形式のコンテナイメージからインスタンス化されます。
- ⑥ コンテナは、Pod の IP で利用可能にされるポートにバインドできます。
- ⑦ OpenShift Container Platform は、コンテナが特権付きコンテナとして実行されるか、選択したユーザーが実行できるようにするかなどを指定するコンテナのセキュリティコンテキストを定義します。デフォルトのコンテキストには多くの制限がありますが、管理者は必要に応じてこれを変更できます。
- ⑧ コンテナは外部ストレージボリュームがマウントされるコンテナ内の場所を指定します。この場合、レジストリーのデータを保存するためのボリュームと、OpenShift Container Platform API に対する要求の実行時にレジストリーが必要とする認証情報にアクセスするためのボリュームがあります。

Pod 再起動ポリシーと使用可能な値の **Always**、**OnFailure**、および **Never** です。デフォルト値は **Always** です。

- 10 OpenShift Container Platform API に対して要求を実行する Pod には共通するパターンが見られ、**serviceAccount** フィールドには、要求を実行する際に Pod が認証する必要のある**サービスアカウント**ユーザーを指定します。これにより、カスタムインフラストラクチャーコンポーネントの詳細なアクセス制御が可能になります。
- 11 Pod はコンテナで使用するストレージボリュームを定義します。この場合、レジストリーストレージの一時ボリュームおよびサービスアカウントの認証情報が含まれる **シークレット** ボリュームを提供します。



注記

この Pod 定義には、Pod が作成され、ライフサイクルが開始された後に OpenShift Container Platform によって自動的に設定される属性は含まれません。[Kubernetes Pod ドキュメント](#)には、Pod の機能および目的についてのさらに詳細な情報が記載されています。

3.3.1.1. Pod 再起動ポリシー

Pod 再起動ポリシーは、Pod のコンテナの終了時に OpenShift Container Platform がどのように反応するかを決定します。このポリシーは Pod のすべてのコンテナに適用されます。

以下の値を使用できます。

- **Always:** Pod が再起動するまで、Pod で正常に終了したコンテナの継続的な再起動を、指数関数的バックオフ遅延値 (10 秒、20 秒、40 秒) に基づいて試行します。デフォルトは **Always** です。
- **OnFailure:** Pod で失敗したコンテナの継続的な再起動を、5 分を上限として指数関数的バックオフ遅延値 (10 秒、20 秒、40 秒) に基づいて試行します。
- **Never:** Pod で終了したコンテナまたは失敗したコンテナの再起動を試行しません。Pod は即時に失敗し、終了します。

ノードにバインドされた Pod は別のノードにバインドされなくなります。これは、Pod をノードの失敗後も存続させるにはコントローラーが必要であることを示しています。

条件	コントローラーのタイプ	再起動ポリシー
(バッチ計算などで) 終了することが予想される Pod	ジョブ	OnFailure または Never
(Web サービスなど) 終了しないことが予想される Pod	レプリケーションコントローラー	Always
マシンごとに実行される必要のある Pod	Daemonset	任意

Pod のコンテナが失敗し、再起動ポリシーが **OnFailure** に設定されている場合、Pod はノード上に留まり、コンテナは再起動します。コンテナの再起動を望まない場合には、再起動ポリシーの **Never** を使用します。

Pod 全体が失敗すると、OpenShift Container Platform は新規 Pod を起動します。開発者はアプリケーションが新規 Pod で再起動される可能性に対応する必要があります。とくに、アプリケーションは一時的なファイル、ロック、以前の実行で生じた未完成の出力などを処理する必要があります。

OpenShift Container Platform が失敗したコンテナに関連して再起動ポリシーを使用する方法についての詳細は、Kubernetes ドキュメントの「[Example States](#)」を参照してください。

3.3.1.2. 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。

挿入できる Pod の Preset オブジェクトの使用

- 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

開発者は、すべての情報を Pod に追加するために Pod ラベルが PodPreset のラベルセクターに一致することを確認する必要があります。Pod のラベルは Pod を、一致するラベルセクターを持つ 1 つ以上の Pod の Preset オブジェクトに関連付けます。

Pod の Preset を使用する開発者は、Pod が消費するサービスの詳細を把握していなくても Pod のプロビジョニングを実行できます。管理者はサービスの設定項目を開発者に非表示にできます。これによって、開発者の Pod のデプロイが妨げられることはありません。



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 仕様の **Podpreset.admission.kubernetes.io/exclude: "true"** パラメーターを使用して、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。

3.3.2. Init コンテナ

init コンテナは、Pod のアプリケーションコンテナの起動前に起動している Pod のコンテナです。Init コンテナは、残りのコンテナが起動する前にボリュームを共有し、ネットワーク操作を実行し、計算を実行しています。Init コンテナは一部の前提条件が満たされるまでアプリケーションの起動をブロックしたり、遅延させたりすることもできます。

Pod が起動し、ネットワークおよびボリュームが初期化されると、init コンテナが順番に起動します。各 init コンテナは、次のコンテナが起動する前に正常に終了している必要があります。init コンテナが (ランタイムのために) 起動に失敗するか、または失敗して終了する場合、Pod の [再起動ポリシー](#)に基づいてリタイアします。

Pod はすべての init コンテナが正常に実行されるまで準備状態になりません。

一部の [init コンテナの使用例](#)については、Kubernetes ドキュメントを参照してください。

以下の例は、2 つの init コンテナを持つ単純な Pod の概要を示しています。最初の init コンテナは **myservice** を待機し、2 つ目は **mydb** を待機します。両方のコンテナが正常に実行されると、Pod が起動します。

例3.2 Init コンテナ Pod オブジェクト定義のサンプル (YAML)

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-Pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice ❶
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']
  - name: init-mydb ❷
    image: busybox
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']
```

❶ **myservice** コンテナを指定します。

❷ **mydb** コンテナを指定します。

各 init コンテナには、**readinessProbe** を除くすべての [アプリコンテナのフィールド](#)が含まれます。Pod の起動を継続するには、Init コンテナは終了している必要があります、完了 (completion) 以外の readiness を定義することはできません。

Init コンテナには Pod の **activeDeadlineSeconds** およびコンテナの **livenessProbe** を含めることができ、init コンテナの永久的な失敗を防ぐことができます。有効な期限には init コンテナで使用される時間が含まれます。

3.3.3. サービス

Kubernetes [サービス](#) は内部ロードバランサーとして機能します。これは、受信する接続をプロキシして送信するために一連の複製された [Pod](#) を特定します。サポートする Pod は、サービスが一貫して利用可能な状態である場合に任意でサービスに追加したり削除したりでき、サービスに依存するすべてのものが一貫したアドレスでサービスを参照できます。デフォルトのサービス clusterIP アドレスは OpenShift Container Platform 内部ネットワークからのもので、Pod が相互にアクセスできるようにするために使用されます。

サービスへの外部アクセスを許可するには、クラスターの [外部](#) にある追加の **externalIP** および **ingressIP** アドレスをサービスに割り当てることができます。これらの **externalIP** アドレスを、サービスへの [高可用性のある](#) アクセスを提供する仮想 IP アドレスにすることもできます。

サービスには、アクセス時に該当のサポートする Pod にプロキシ化される IP アドレスとポートのペアが割り当てられます。サービスはラベルセレクターを使用して、特定のポートで特定のネットワークサービスを提供する実行中のすべてのコンテナを見つけます。

Pod と同様にサービスは REST オブジェクトです。以下の例は、上記の定義された Pod のサービス定義を示しています。

例3.3 サービスオブジェクト定義 (YAML)

```
apiVersion: v1
kind: Service
metadata:
  name: docker-registry ❶
spec:
  selector: ❷
    docker-registry: default
  clusterIP: 172.30.136.123 ❸
  ports:
    - nodePort: 0
      port: 5000 ❹
      protocol: TCP
      targetPort: 5000 ❺
```

- ❶ サービス名 **docker-registry** は、同じ namespace の別の Pod に挿入されるサービス IP と共に環境変数を作成するためにも使用されます。名前の最大長は 63 文字です。
- ❷ ラベルセレクターは、**docker-registry=default** ラベルが割り当てられたすべての Pod をサポートする Pod として特定します。
- ❸ 作成時に内部 IP のプールから自動的に割り当てられるサービスの仮想 IP です。
- ❹ サービスがリッスンするポートです。
- ❺ サービスによる接続の転送先となるサポートする Pod のポートです。

[Kubernetes ドキュメント](#) には、サービスについての詳細が記載されています。

3.3.3.1. サービス externalIP

クラスターの内部 IP アドレスに加えて、ユーザー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。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** ファイルに設定される **ExternalIPNetworkCIDRs** 範囲からクラスター管理者によって選択される必要があります。**master-config.yaml** が変更される場合、マスターサービスは再起動する必要があります。

例3.4 externalIPNetworkCIDR /etc/origin/master/master-config.yaml のサンプル

```
networkConfig:
  ExternalIPNetworkCIDR: 192.0.1.0/24
```

例3.5 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" : [
      "192.0.1.1"
    ]
  }
}
```

- 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。

3.3.3.2. サービス ingressIP

クラウド以外のクラスターでは、externalIP アドレスをアドレスのプールから自動的に割り当てることができます。これにより、管理者がそれらを手動で割り当てる必要がなくなります。

このプールは **/etc/origin/master/master-config.yaml** ファイルに設定されます。このファイルを変更した後はマスターサービスを再起動します。

ingressIPNetworkCIDR はデフォルトで **172.29.0.0/16** に設定されます。クラスター環境でこのプライベート範囲を使用していない場合は、デフォルトの範囲を使用するか、またはカスタム範囲を使用します。



注記

[高可用性](#)を設定している場合、この範囲は 256 アドレス未満にする必要があります。

例3.6 ingressIPNetworkCIDR /etc/origin/master/master-config.yaml のサンプル

```
networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16
```

3.3.3.3. サービス NodePort

サービス **type=NodePort** を設定して、フラグで設定された範囲 (デフォルト: 30000-32767) からポートを割り当てます。各ノードはそのポート (すべてのノードでポート番号が同じ) をサービスにプロキシ送信します。

選択されたポートは、サービス設定の **spec.ports[*].nodePort** の下に報告されます。

カスタムポートを指定するには、単純にポート番号を **nodePort** フィールドに設定します。カスタムポート番号は **nodePorts** の設定された範囲内になければなりません。'**master-config.yaml**' が変更される場合、マスターサービスの再起動が必要になります。

例3.7 servicesNodePortRange /etc/origin/master/master-config.yaml のサンプル

```
kubernetesMasterConfig:
  servicesNodePortRange: ""
```

サービスは **<NodeIP>:spec.ports[].nodePort** および **spec.clusterIp:spec.ports[].port** として表示されます。



注記

nodePort の設定は、特権付きの操作で実行されます。

3.3.3.4. サービスプロキシモード

OpenShift Container Platform にはサービスルーティングインフラストラクチャーの 2 つの異なる実装があります。デフォルトの実装は完全に **iptables** をベースとしており、エンドポイント Pod 間の着信サービス接続を分散するために確率的な **iptables** 再作成ルールを使用します。古い方の実装はユーザー空間プロセスを使用して着信接続を受け入れた後に、クライアントとエンドポイント Pod 間のトラフィックをプロキシ送信します。

iptables ベースの実装はより効率的ですが、この場合すべてのエンドポイントで接続が常に受け入れ可能であることが条件になります。ユーザー空間の実装は速度が遅くなりますが、機能するエンドポイントが見つかるまで複数のエンドポイントを試行できます。適切な [Readiness チェック](#) (または通常信頼できるノードおよび Pod) がある場合は、**iptables** ベースのサービスプロキシが最適なオプションになります。または、クラスターのインストール時またはデプロイ後に、ノード設定ファイルを編集してユーザー空間ベースのプロキシを有効にできます。

3.3.3.5. ヘッドレスサービス

アプリケーションが負荷分散や単一サービス IP アドレスを必要しない場合にヘッドレスサービスを作成できます。ヘッドレスサービスを作成する場合、負荷分散やプロキシは実行されず、クラスター IP はこのサービスに割り当てられません。このサービスの場合、サービスにセクターが定義されているかどうかに応じて DNS が自動的に設定されます。

セクターのあるサービス: セクターを定義するヘッドレスサービスの場合、エンドポイントコントローラーは API に **Endpoints** レコードを作成し、DNS 設定を変更して、サービスをサポートする Pod を直接参照する **A** レコード (アドレス) を返します。

セクターなしのサービス: セクターを定義しないヘッドレスサービスの場合、エンドポイントコントローラーは **Endpoints** レコードを作成しません。ただし、DNS システムは以下のレコードを検索し、設定します。

- **ExternalName** タイプサービスの場合は、**CNAME** レコード。
- それ以外のすべてのサービスタイプの場合は、名前をサービスと共有するエンドポイントの **A** レコード。

3.3.3.5.1. ヘッドレスサービスの作成

ヘッドレスサービスの作成は標準的なサービスの作成と同様ですが、**ClusterIP** アドレスを宣言しません。ヘッドレスサービスを作成するには、**clusterIP: None** パラメーター値をサービス YAML 定義に追加します。

たとえば、以下は Pod のグループを同じクラスターまたはサービスの一部として組み込む場合です。

Pod の一覧

```
$ oc get Pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
frontend-1-287hw	1/1	Running	0	7m	172.17.0.3	node_1
frontend-1-68km5	1/1	Running	0	7m	172.17.0.6	node_1

ヘッドレスサービスは以下のように定義できます。

ヘッドレスサービス定義

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: ruby-helloworld-sample
    template: application-template-stibuild
  name: frontend-headless ❶
spec:
  clusterIP: None ❷
  ports:
    - name: web
      port: 5432
      protocol: TCP
      targetPort: 8080
```

```
selector:
  name: frontend ❸
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

- ❶ ヘッドレスサービスの名前です。
- ❷ **clusterIP** 変数を **None** に設定すると、ヘッドレスサービスが宣言されます。
- ❸ **frontend** ラベルが付いたすべての Pod を選択します。

ヘッドレスサービスには独自の IP アドレスがありません。

```
$ oc get svc
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
frontend                            ClusterIP           172.30.232.77   <none>           5432/TCP
12m
frontend-headless                    ClusterIP           None            <none>           5432/TCP
10m
```

3.3.3.5.2. ヘッドレスサービスを使用したエンドポイントの検出

ヘッドレスサービスを使用する利点として、Pod の IP アドレスを直接検出できることがあります。標準サービスはロードバランサーまたはプロキシとして機能するか、またはサービス名を使用してワークロードオブジェクトへのアクセスを付与します。ヘッドレスサービスの場合、サービス名はサービスごとに分類される Pod の IP アドレスセットに解決します。

標準サービスの DNS **A** レコードを検索すると、サービスの負荷分散された IP を取得できます。

```
$ dig frontend.test A +search +short
172.30.232.77
```

ヘッドレスサービスの場合には、個別 Pod の IP の一覧を取得できます。

```
$ dig frontend-headless.test A +search +short
172.17.0.3
172.17.0.6
```

注記

ヘッドレスサービスを StatefulSet と共に、また初期化および終了時に Pod の DNS を解決する必要のある関連のユースケースで使用する場合、**publishNotReadyAddresses** を **true** に設定します (デフォルト値は **false**)。 **publishNotReadyAddresses** が **true** に設定されている場合、これは DNS 実装で、サービスに関連付けられたエンドポイントのサブセットの **notReadyAddresses** を公開する必要があることを示します。

3.3.4. ラベル

ラベルは、API オブジェクトを編成し、分類し、選択するために使用されます。たとえば、**Pod** にはラ

ベルによる「タグ付け」が実行されてから、[サービス](#)がラベルセクターを使用してそれらがプロキシする Pod を識別します。これにより、サービスが Pod のグループを参照すること、また異なるコンテナを含む可能性のある Pod を関連エンティティとして処理することを可能にします。

ほとんどのオブジェクトでは、ラベルをそのメタデータに組み込むことができます。そのため、ラベルは任意に関連付けられたオブジェクトをグループに分類するために使用できます。たとえば、特定アプリケーションのすべての [Pod](#)、[サービス](#)、[レプリケーションコントローラー](#)、および[デプロイメント設定](#)をグループ化することができます。

ラベルは、以下の例のような単純なキー/値のペアになります。

```
labels:
  key1: value1
  key2: value2
```

以下の例を見てみましょう。

- **nginx** コンテナで構成される Pod に **role=webserver** ラベルがある。
- **Apache httpd** コンテナで構成される Pod に同じラベル **role=webserver** がある。

この場合、**role=webserver** ラベルを持つ Pod を使用するように定義されたサービスまたはレプリケーションコントローラーは上記 Pod の両方を同じグループとして処理します。

[Kubernetes ドキュメント](#)には、ラベルについてのさらに詳細な情報が記載されています。

3.3.5. エンドポイント

サービスをサポートするサーバーはエンドポイントと呼ばれ、サービスと同じ名前を持つタイプ **Endpoints** のオブジェクトで指定されます。サービスが Pod でサポートされる場合、それらの Pod は通常はサービス仕様のラベルセクターで指定され、OpenShift Container Platform は、それらの Pod を参照するエンドポイントオブジェクトを自動的に作成します。

サービスを作成する場合でも、OpenShift Container Platform クラスターの Pod ではなく、外部ホストでサポートされるようにする必要がある場合があります。この場合、サービスの **selector** フィールドを省略し、[エンドポイントオブジェクトを手動で作成](#)できます。

OpenShift Container Platform は、大半のユーザーが Pod およびサービス用に予約されたネットワークブロックの IP アドレスを参照するエンドポイントオブジェクトの手動による作成を許可しないことに注意してください。**endpoints/restricted**のリソースの [createパーミッション](#)を持つクラスター管理者その他ユーザーのみがこれらのエンドポイントオブジェクトを作成できます。

3.4. プロジェクトおよびユーザー

3.4.1. ユーザー

OpenShift Container Platform との対話はユーザーに関連付けられます。OpenShift Container Platform ユーザーオブジェクトはシステム内のパーミッションを付与されるアクターを表します。パーミッションは [ロールをそれらまたはそれらのグループに追加](#)して付与されます。

ユーザーにはいくつかのタイプが存在します。

一般ユーザー	ほとんどの対話型の OpenShift Container Platform ユーザーは一般ユーザーとして表示されます。一般ユーザーは、初回ログイン時にシステムに自動的に作成されるか、または API で作成できます。一般ユーザーは User オブジェクトで表示されず。例: joe alice
システムユーザー	これらのユーザーの多くは、API との安全な対話を主な目的として、インフラストラクチャが定義される際に自動的に作成されます。これらには、クラスター管理者 (すべてのアクセスを持つ)、ノードごとのユーザー、ルーターおよびレジストリーで使用できるユーザーなどが含まれます。また、非認証要求に対してデフォルトで使用される anonymous (匿名) システムユーザーも含まれます。例: system:admin system:openshift-registry system:node:node1.example.com
サービスアカウント	これらはプロジェクトに関連付けられる特殊なシステムユーザーです。これらの中にはプロジェクトの初回作成時に自動作成されるものがありますが、プロジェクト管理者は各 プロジェクト のコンテンツへのアクセスを定義する目的で追加のサービスアカウントを作成できます。サービスアカウントは ServiceAccount オブジェクトで表示されます。例: system:serviceaccount:default:deployer system:serviceaccount:foo:builder

すべてのユーザーには、OpenShift Container Platform にアクセスするために何らかの**認証**が必要です。認証がないか、または認証が無効の API 要求は、**anonymous** (匿名) システムユーザーによる要求として認証されます。認証が実行されると、ユーザーの実行が**許可される**内容がポリシーによって決定されます。

3.4.2. Namespace

Kubernetes の namespace は、クラスター内でリソースのスコープを設定するメカニズムを提供します。OpenShift Container Platform において、**プロジェクト**は追加のアノテーションを含む Kubernetes の namespace を指します。

Namespace は以下の一意のスコープを提供します。

- 基本的な名前の衝突を避けるための名前付きリソース。
- 信頼できるユーザーに委任された管理権限。
- コミュニティーのリソース消費を制限する機能。

システムのほとんどのオブジェクトのスコープは namespace 別に設定されますが、一部にはノードやユーザーを含め、予想されるもので namespace がないものがあります。

[Kubernetes ドキュメント](#)には namespace についてのさらに詳細な情報が記載されています。

3.4.3. プロジェクト

プロジェクトは追加のアノテーションを含む Kubernetes の namespace であり、一般ユーザーのリソースへのアクセスを管理する中心的な手段です。プロジェクトはユーザーのコミュニティが他のコミュニティと切り離してコンテンツを編成し、管理することを可能にします。ユーザーには、管理者によってプロジェクトへのアクセスが付与される必要があり、ユーザーがプロジェクトの作成を許可されている場合には、ユーザー独自のプロジェクトへのアクセスが自動的に付与されます。

プロジェクトには、別個の**name**、**displayName**、および**description**を含めることができます。

- 必須の **name** はプロジェクトの一意的 ID であり、CLI ツールまたは API を使用する場合に最も明確に表示されます。名前の最大長は 63 文字です。
- オプションの **displayName** はプロジェクトが Web コンソールで表示される方法を示します (デフォルトは **name** に設定されます)。
- オプションの **description** にはプロジェクトのさらに詳細な記述を使用でき、これも Web コンソールで表示できます。

各プロジェクトは、以下の独自のセットのスコープを設定します。

オブジェクト	Pod、サービス、レプリケーションコントローラーなど。
ポリシー	ユーザーがオブジェクトに対してアクションを実行できるか/できないかについてのルール。
制約	制限を設定できる各種オブジェクトのクォータ。
サービスアカウント	サービスアカウントは、プロジェクトのオブジェクトへの指定されたアクセスで自動的に機能します。

クラスター管理者は [プロジェクトの作成](#) を実行でき、プロジェクトの [管理者権限の委任](#) をユーザーコミュニティの任意のメンバーに対して実行できます。また、クラスター管理者は開発者が [独自のプロジェクト](#) を作成することも許可します。

開発者および管理者は、[CLI](#) または [Web コンソール](#) を使用して [プロジェクトとの対話](#) を実行できます。

3.4.3.1. インストール時に提供されるプロジェクト

OpenShift Container Platform には追加設定なしで使用できる数多くのプロジェクトが含まれますが、**openshift** はユーザーにとって最も重要なプロジェクトになります。

openshift: ユーザーに表示されるプロジェクトで、主に日常的なタスクのオブジェクトを格納するために使用されます。これらには、テンプレートやイメージなどの複数プロジェクトでアクセスされるアプリケーションオブジェクトが含まれます。これらのオブジェクトは、Pod 間の通信が不要なものである必要があります。

3.5. ビルドおよびイメージストリーム

3.5.1. ビルド

ビルド は、入力パラメーターを、作成されるオブジェクトに変換するプロセスです。ほとんどの場合、このプロセスは入力パラメーターまたはソースコードを実行可能なイメージに変換するために使用されます。[BuildConfig](#) オブジェクトはビルドプロセス全体の定義です。

OpenShift Container Platform は、Docker 形式のコンテナをビルドイメージから作成し、それらを [コンテナレジストリー](#) にプッシュして Kubernetes を利用します。

ビルドオブジェクトは共通する特性を共有します。これらには、ビルドの入力、ビルドプロセスを完了する必要性、ビルドプロセスのロギング、正常なビルドからのリソースの公開、およびビルドの最終ステータスの公開などが含まれます。ビルドはリソース制限を利用し、CPU 使用、メモリー使用およびビルドまたは Pod の実行時間などのリソースの制限を指定します。

OpenShift Container Platform ビルドシステムは、ビルド API で指定される選択可能なタイプに基づく、**ビルドストラテジー** の拡張可能なサポートを提供します。以下は利用可能な 3 つの主なビルドストラテジーです。

- [Docker ビルド](#)
- [Source-to-Image \(S2I\) ビルド](#)
- [カスタムビルド](#)

デフォルトでは、Docker ビルドおよび S2I ビルドがサポートされます。

ビルドの作成されるオブジェクトは、この作成に使用されるビルダーによって異なります。Docker および S2I ビルドの場合、作成されるオブジェクトは実行可能なイメージです。カスタムビルドの場合、作成されるオブジェクトはビルダーイメージの作成者が指定するものになります。

さらに [Pipeline ビルド](#) ストラテジーを使用して、高度なワークフローを実装することができます。

- [継続的インテグレーション](#)
- [継続的デプロイ](#)

ビルドコマンドの一覧については、『[開発者ガイド](#)』を参照してください。

OpenShift Container Platform の Docker を使用したビルドについての詳細は、[アップストリームドキュメント](#)を参照してください。

3.5.1.1. Docker ビルド

Docker ビルドストラテジーは `docker build` コマンドを起動するため、**Dockerfile** とそれに含まれるすべての必要なアーティファクトのあるリポジトリが実行可能なイメージを生成することを予想します。

3.5.1.2. Source-to-Image (S2I) ビルド

[Source-to-Image \(S2I\)](#) は再現可能な Docker 形式のコンテナイメージをビルドするためのツールです。これはアプリケーションソースをコンテナイメージに挿入し、新規イメージをアセンブルして実行可能なイメージを生成します。新規イメージはベースイメージ (ビルダー) とビルドされたソースを組み込み、**docker run** コマンドで使用することができます。S2I は増分ビルドをサポートします。これは以前にダウンロードされた依存関係、以前にビルドされたアーティファクトなどを再利用します。

S2I の利点には以下が含まれます。

イメージの柔軟性	S2I スクリプトを作成して、アプリケーションコードをほとんどすべての既存の Docker 形式コンテナに挿入し、既存のエコシステムを活用することができます。現時点で S2I は tar を使用してアプリケーションソースを挿入するため、イメージは tar が実行されたコンテンツを処理できる必要があることに注意してください。
スピード	S2I の場合、アセンブルプロセスでは、各手順で新規の層を作成せずに多数の複雑な操作を実行できるため、プロセスのスピードが速くなります。さらに、S2I スクリプトを作成してアプリケーションイメージの以前のバージョンに保存されたアーティファクトを再利用できるため、ビルドの実行時に毎回ダウンロードまたはビルドを実行する必要がありません。

パッチ適用容易性 (Patchability)	S2I では、基礎となるイメージがセキュリティ上の問題でパッチを必要とする場合にアプリケーションを一貫した方法で再ビルドできます。
運用効率	Dockerfile が許可するアクションを実行する代わりにビルド操作を制限することで、PaaS オペレーターはビルドシステムの意図しない、または意図的な誤用を避けることができます。
運用上のセキュリティ	任意の Dockerfile をビルドすると、root 権限の昇格のためにホストシステムを公開します。これは Docker ビルドプロセス全体が Docker 権限を持つユーザーとして実行されるため、悪意あるユーザーによって悪用される可能性があります。S2I は root ユーザーとして実行される操作を制限し、スクリプトを root 以外のユーザーとして実行できます。
ユーザー効率	S2I は開発者が、アプリケーションのビルド時の開発の反復スピードを低下させる可能性がある yum install タイプの操作を実行することを防ぎます。
エコシステム	S2I は、アプリケーションのベストプラクティスを利用できるイメージの共有されたエコシステムを促進します。
再現性	生成されるイメージには、特定バージョンのビルドツールおよび依存関係などのすべての入力を含めることができます。これにより、イメージを正確に再現することができます。

3.5.1.3. カスタムビルド

カスタムビルドストラテジーにより、開発者はビルドプロセス全体を対象とする特定のビルダーイメージを定義できます。独自のビルダーイメージを使用してビルドプロセスをカスタマイズできます。

[カスタムビルダーイメージ](#)は、RPM またはベースイメージのビルドなどの、ビルドプロセスのロジックで組み込まれた単純な Docker 形式のコンテナイメージです。**openshift/origin-custom-docker-builder** イメージは、カスタムビルダーイメージの実装例として [Docker Hub](#) レジストリーで利用できます。

3.5.1.4. Pipeline ビルド

Pipeline ストラテジーは、開発者が Jenkins パイプラインプラグインで実行される **Jenkins** パイプラインを定義することを可能にします。ビルドは他のビルドタイプの場合と同様に OpenShift Container Platform で起動し、モニターし、管理できます。

Pipeline ワークフローは Jenkinsfile で定義され、ビルド設定に直接組み込まれるか、または Git リポジトリで指定され、ビルド設定で参照されます。

プロジェクトの Pipeline ストラテジーを使用した初回のビルド設定の定義時に、OpenShift Container Platform は Jenkins サーバーをインスタンス化して Pipeline を実行します。プロジェクトの後続の Pipeline ビルド設定はこの Jenkins サーバーを共有します。

Jenkins サーバーのデプロイ方法や自動プロビジョニングの設定または無効化の方法についての詳細は、「[Pipeline 実行の設定](#)」を参照してください。



注記

Jenkins サーバーは、すべての Pipeline ビルド設定が削除される場合でも自動的に削除されません。これはユーザーによって手動で削除される必要があります。

Jenkins パイプラインについての詳細は、[Jenkins ドキュメント](#)を参照してください。

3.5.2. イメージストリーム

イメージストリームおよびその関連付けられたタグは、OpenShift Container Platform 内で [Docker イメージ](#)を参照するための抽象化を提供します。イメージストリームとそのタグを使用して、利用可能なイメージを確認し、リポジトリのイメージが変更される場合でも必要な特定のイメージを使用していることを確認できます。

イメージストリームには実際のイメージデータは含まれませんが、イメージリポジトリと同様に、関連するイメージの単一の仮想ビューが表示されます。

ビルドおよびデプロイメントをそれぞれ実行し、[ビルド](#)および[デプロイメント](#)を、新規イメージが追加される際やこれに対応する際の通知をイメージストリームで確認できるように設定できます。

たとえば、デプロイメントが特定のイメージを使用していて、そのイメージの新規バージョンが作成される場合、イメージの新規バージョンを選択するようにデプロイメントが自動的に実行されます。

ただし、デプロイメントまたはビルドで使用されるイメージストリームタグが更新されない場合、Docker レジストリーの Docker イメージが更新されている場合でもビルドまたはデプロイメントは以前の (既知の適切であると予想される) イメージの使用を継続します。

ソースイメージは以下のいずれかに保存できます。

- OpenShift Container Platform の[統合レジストリー](#)
- [registry.access.redhat.com](#) または [hub.docker.com](#) などの外部レジストリー
- OpenShift Container Platform クラスターの他のイメージストリーム

(ビルド設定またはデプロイメント設定などの) イメージストリームタグを参照するオブジェクトを定義する際は、Docker リポジトリではなくイメージストリームタグを参照します。アプリケーションのビルドまたはデプロイの実行時に、OpenShift Container Platform はイメージストリームタグを使用して Docker リポジトリをクエリーし、イメージの関連付けられた ID を検索し、そのイメージを使用します。

イメージストリームメタデータは他のクラスター情報と共に etcd インスタンスに保存されます。

以下のイメージストリームには、Python v3.4 イメージを参照する **34** と、Python v3.5 イメージを参照する **35** の 2 つのタグが含まれます。

```
oc describe is python
Name:      python
Namespace: imagestream
Created:   25 hours ago
Labels:    app=python
Annotations: openshift.io/generated-by=OpenShiftWebConsole
             openshift.io/image.dockerRepositoryCheck=2017-10-03T19:48:00Z
Docker Pull Spec: docker-registry.default.svc:5000/imagestream/python
Image Lookup: local=false
Unique Images: 2
```


Tags: 2

34

tagged from centos/python-34-centos7

* centos/python-34-centos7@sha256:28178e2352d31f240de1af1370be855db33ae9782de737bb005247d8791a54d0

14 seconds ago

35

tagged from centos/python-35-centos7

* centos/python-35-centos7@sha256:2efb79ca3ac9c9145a63675fb0c09220ab3b8d4005d35e0644417ee552548b10

7 seconds ago

イメージストリームの使用には、いくつかの大きな利点があります。

- コマンドラインを使用して再プッシュすることなく、タグ付けや、タグのロールバック、およびイメージの迅速な処理を実行できます。
- 新規イメージがレジストリーにプッシュされると、ビルドおよびデプロイメントをトリガーできます。また、OpenShift Container Platform には他のリソースの汎用トリガーがあります (Kubernetes オブジェクトなど)。
- [定期的な再インポート用のタグにマークを付ける](#)こともできます。ソースイメージが変更されると、その変更は選択され、イメージストリームに反映されます。これにより、ビルドまたはデプロイメント設定に応じてビルドおよび/またはデプロイメントフローがトリガーされます。
- 詳細なアクセス制御を使用してイメージを共有し、イメージをチーム間で迅速に分配できます。
- ソースイメージが変更されても、イメージストリームタグはイメージの既知の適切なバージョンを参照したままになり、アプリケーションが予期せずに中断しないようにします。
- イメージストリームオブジェクトのパーミッションを使用して、イメージを閲覧し、使用できるユーザーについてセキュリティ設定を行うことができます。
- クラスターレベルでイメージを読み込んだり、一覧表示するパーミッションのないユーザーは、イメージストリームを使用してプロジェクトでタグ付けされたイメージを取得できます。

選別されたイメージストリームのセットについては、[OpenShift Image Streams and Templates library](#) を参照してください。

イメージストリームの使用時に、イメージストリームタグの参照先およびタグおよびイメージへの変更による影響について把握しておくことは重要です。以下は例になります。

- イメージストリームタグが Docker イメージタグを参照する場合、Docker イメージタグの更新方法を理解しておく必要があります。たとえば、Docker イメージタグ **docker.io/ruby:2.4** はおそらく v2.4 ruby イメージを常に参照します。一方、Docker イメージタグ **docker.io/ruby:latest** はおそらくメジャーバージョンで変更されます。そのため、イメージストリームタグが参照する Docker イメージタグは、イメージストリームタグの安定度を示すものとなります (Docker イメージタグを参照するように設定している場合)。
- イメージストリームタグが別のイメージストリームタグをフォローする場合 (Docker イメージ

タグを直接参照しない場合)、イメージストリームタグが別のイメージストリームタグをフォローするように更新される可能性があります。この場合、互換性のないバージョンの変更が選択されてしまう可能性があります。

3.5.2.1. 重要な用語

Docker リポジトリ

関連する Docker イメージおよびそれらを識別するタグのコレクションです。たとえば、OpenShift Jenkins イメージは Docker リポジトリにあります。

```
docker.io/openshift/jenkins-2-centos7
```

Docker レジストリー

Docker リポジトリからイメージを保存し、提供できるコンテンツサーバーです。以下は例になります。

```
registry.access.redhat.com
```

Docker イメージ

コンテナとして実行できる特定のコンテナセットです。通常は Docker リポジトリ内の特定のタグに関連付けられます。

Docker イメージタグ

特定のイメージを区別する、リポジトリ内の Docker イメージに適用されるラベルです。たとえば、ここでは **3.6.0** がタグとして使用されています。

```
docker.io/openshift/jenkins-2-centos7:3.6.0
```



注記

新規の Docker イメージコンテンツを参照するようにいつでも更新できる Docker イメージタグです。

Docker イメージ ID

イメージをプルするために使用できる SHA (セキュアハッシュアルゴリズム) コードです。以下は例になります。

```
docker.io/openshift/jenkins-2-centos7@sha256:ab312bda324
```



注記

SHA イメージ ID は変更できません。特定の SHA ID は同一の Docker イメージコンテンツを常に参照します。

イメージストリーム

タグで識別される任意の数の Docker 形式のコンテナイメージへのポインターが含まれる OpenShift Container Platform オブジェクトです。イメージストリームは Docker リポジトリと同等のものとみなすことができます。

イメージストリームタグ

イメージストリーム内のイメージへの名前付きポインターです。イメージストリームタグは Docker イメージタグに似ています。以下の「[イメージストリームタグ](#)」を参照してください。

イメージストリームイメージ

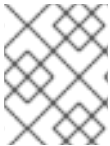
イメージがタグ付けされている特定のイメージストリームから特定の Docker イメージを取得できるようにするイメージです。イメージストリームイメージは、特定のイメージの SHA ID についてのメタデータをプルする API リソースオブジェクトです。以下の「[イメージストリームイメージ](#)」を参照してください。

イメージストリームトリガー

イメージストリームタグの変更時に特定のアクションを生じさせるトリガーです。たとえば、インポートによりタグの値が変更される可能性があり、これによってデプロイメント、ビルド、またはそれらについてリッスンする他のリソースがある場合はトリガーが発生します。以下の「[イメージストリームトリガー](#)」を参照してください。

3.5.2.2. イメージストリームの設定

イメージストリームのオブジェクトファイルには以下の要素が含まれます。



注記

イメージおよびイメージストリームの管理についての詳細は、『[開発者ガイド](#)』を参照してください。

イメージストリームオブジェクト定義

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: 2017-09-29T13:33:49Z
  generation: 1
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample ❶
  namespace: test
  resourceVersion: "633"
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample ❷
  tags:
    - items:
        - created: 2017-09-02T10:15:09Z
          dockerImageReference: 172.30.56.218:5000/test/origin-ruby-sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d ❸
          generation: 2
          image:
            sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5 ❹
            - created: 2017-09-29T13:40:11Z
              dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
```



```
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
  generation: 1
  image:
sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
  tag: latest 5
```

- ❶ イメージストリームの名前です。
- ❷ 新規イメージをこのイメージストリームで追加/更新するためにプッシュできる Docker リポジトリパスです。
- ❸ このイメージストリームタグが現在参照している SHA ID です。このイメージストリームタグを参照するリソースはこの ID を使用します。
- ❹ このイメージストリームタグが以前に参照した SHA ID です。古いイメージにロールバックするために使用できます。
- ❺ イメージストリームタグ名です。

イメージストリームを参照するビルド設定のサンプルについては、設定の **Strategy** スタンザで「[BuildConfig の概要](#)」を参照してください。

イメージストリームを参照するデプロイメント設定のサンプルについては、設定の **Strategy** スタンザで「[デプロイメント設定の作成](#)」の部分参照してください。

3.5.2.3. イメージストリームイメージ

イメージストリームイメージ は、イメージストリームから特定のイメージ ID を参照します。

イメージストリームイメージにより、イメージがタグ付けされている特定のイメージストリームからイメージについてのメタデータを取得できます。

イメージストリームイメージのオブジェクトは、イメージをインポートしたり、イメージストリームにタグ付けしたりする場合に OpenShift Container Platform に常に自動的に作成されます。イメージストリームイメージのオブジェクトは、イメージストリームを作成するために使用するイメージストリーム定義で明示的に定義する必要はありません。

イメージストリームイメージはリポジトリからのイメージストリーム名およびイメージ ID で構成されており、@ 記号で区切られています。

```
<image-stream-name>@<image-id>
```

上記のイメージストリームオブジェクトサンプルのイメージを参照する際に、イメージストリームイメージは以下ようになります。

```
origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

3.5.2.4. イメージストリームタグ

イメージストリームタグは、イメージストリームのイメージに対する名前付きポインターです。これは **istag** として省略されることが多くあります。イメージストリームタグは、指定のイメージストリームおよびタグについてイメージを参照するか、または取得するために使用されます。

イメージストリームタグは、ローカルイメージまたは外部で管理されるイメージを参照できます。これには、タグで参照したすべてのイメージのスタックとして参照されるイメージの履歴が含まれます。新規または既存のイメージが特定のイメージストリームタグでタグ付けされる場合は常に、それは履歴スタックの最初の位置に置かれます。これまで先頭の位置を占めていたイメージは 2 番目の位置などに置かれます。これにより、タグが過去のイメージを再び参照できるよう簡単にロールバックできます。

以下のイメージストリームタグは、[上記のイメージストリームオブジェクトのサンプル](#)から取られています。

2 つのイメージが履歴に含まれるイメージストリームタグ

```
tags:
- items:
  - created: 2017-09-02T10:15:09Z
    dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7d
d13d
    generation: 2
    image:
sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
  - created: 2017-09-29T13:40:11Z
    dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab
3fc5
    generation: 1
    image:
sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest
```

イメージストリームタグは **永続** タグまたは **トラッキング** タグにすることができます。

- **永続タグ** は、Python 3.5 などの特定バージョンのイメージを参照するバージョン固有のタグです。
- **トラッキングタグ** は別のイメージストリームタグをフォローする参照タグで、シンボリックリンクなどのようにフォローするイメージを変更するために更新される可能性があります。これらの新たなレベルは後方互換性を持つことが保証されないことに注意してください。たとえば、OpenShift Container Platform に同梱される **latest** イメージストリームタグはトラッキングタグです。これは、**latest** イメージストリームタグのコンシューマーが、新規レベルが利用可能になるとイメージで提供されるフレームワークの最新レベルに更新されることを意味します。**v3.6** への **latest** イメージストリームタグは **v3.7** に変更される可能性が常にあります。これらの **latest** イメージストリームタグは Docker の **latest** タグとは異なる動作をすることに注意してください。この場合、**latest** イメージストリームタグは Docker リポジトリの最新イメージを参照せず、イメージの最新バージョンでない可能性のある別のイメージストリームタグを参照します。たとえば、**latest** イメージストリームタグがイメージの **v3.2** を参照する場合、**3.3** バージョンがリリースされても **latest** タグは **v3.3** に自動的に更新されず、これが **v3.3** イメージストリームタグをポイントするように手動で更新されるまで **v3.2** を参照したままになります。



注記

トラッキングタグは単一のイメージストリームに制限され、他のイメージストリームを参照することはできません。

各自のニーズに合わせて独自のイメージストリームタグを作成できます。「[推奨されるタグ付け規則](#)」を参照してください。

イメージストリームタグは、コロンで区切られた、イメージストリームの名前とタグで構成されています。

```
<image stream name>:<tag>
```

たとえば、[上記のイメージストリームオブジェクトのサンプル](#)で

sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d イメージを参照するためのイメージストリームタグは以下のようになります。

```
origin-ruby-sample:latest
```

3.5.2.5. イメージストリーム変更トリガー

イメージストリームトリガーにより、ビルドおよびデプロイメントは、アップストリームイメージの新規バージョンが利用可能になると自動的に起動します。

たとえば、ビルドおよびデプロイメントはイメージストリームタグの変更時に自動的に起動します。これは、特定のイメージストリームタグをモニターし、変更の検出時にビルドまたはデプロイメントに通知することで実行されます。

ImageChange トリガーにより、[イメージストリームタグ](#)の内容の変更時 (新規バージョンのイメージのプッシュ時) に新規レプリケーションコントローラーが常に生成されます。

例3.8 ImageChange トリガー

```
triggers:
- type: "ImageChange"
  imageChangeParams:
    automatic: true ①
    from:
      kind: "ImageStreamTag"
      name: "origin-ruby-sample:latest"
      namespace: "myproject"
    containerNames:
      - "helloworld"
```

① **imageChangeParams.automatic** フィールドが **false** に設定されると、トリガーが無効になります。

上記の例では、**origin-ruby-sample** イメージストリームの **latest** タグの値が変更され、新規イメージの値がデプロイメント設定の **helloworld** コンテナに指定される現在のイメージと異なる場合に、**helloworld** コンテナの新規イメージを使用して新規のレプリケーションコントローラーが作成されます。

**注記**

ImageChange トリガーがデプロイメント設定 (**ConfigChange** トリガーと **automatic=false** が設定されているか、または **automatic=true** が設定されている) で定義されていて、**ImageChange** トリガーで参照されている **ImageStreamTag** がまだ存在していない場合、初回のデプロイメントプロセスは、ビルドによってイメージがインポートされるか、**ImageStreamTag** にプッシュされるとすぐに自動的に開始されます。

3.5.2.6. イメージストリームのマッピング

[統合レジストリー](#) が新規イメージを受信すると、これは OpenShift Container Platform にマップするイメージストリームを作成し、これを送信してイメージのプロジェクト、名前、タグおよびイメージメタデータを提供します。

**注記**

イメージストリームのマッピングの設定は拡張機能です。

この情報は、新規イメージを作成する際 (すでに存在しない場合) やイメージをイメージストリームにタグ付けする際に使用されます。OpenShift Container Platform は、コマンド、エントリーポイント、および環境変数などの各イメージについての完全なメタデータを保存します。OpenShift Container Platform のイメージは不変であり、名前の最大長は 63 文字です。

**注記**

イメージの手動のタグ付けの詳細については、『[開発者ガイド](#)』を参照してください。

以下のイメージストリームマッピングのサンプルにより、**test/origin-ruby-sample:latest** のようなイメージのタグ付けが行われます。

イメージストリームマッピングオブジェクト定義

```
apiVersion: v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
    - name:
sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
  size: 0
    - name:
sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
  size: 196634330
    - name:
sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
  size: 0
    - name:
sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
  size: 0
```

```

- name:
sha256: ca062656bfff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
  size: 177723024
- name:
sha256: 63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
  size: 55679776
- name:
sha256: 92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
  size: 11939149
dockerImageMetadata:
  Architecture: amd64
  Config:
    Cmd:
      - /usr/libexec/s2i/run
    Entrypoint:
      - container-entrpoint
    Env:
      - RACK_ENV=production
      - OPENSIFT_BUILD_NAMESPACE=test
      - OPENSIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-
world.git
      - EXAMPLE=sample-app
      - OPENSIFT_BUILD_NAME=ruby-sample-build-1
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
    ExposedPorts:
      8080/tcp: {}
    Labels:
      build-date: 2015-12-23
      io.k8s.description: Platform for building and running Ruby 2.2
applications
      io.k8s.display-name: 172.30.56.218:5000/test/origin-ruby-
sample:latest
      io.openshift.build.commit.author: Ben Parees
<bparees@users.noreply.github.com>
      io.openshift.build.commit.date: Wed Jan 20 10:14:27 2016 -0500
      io.openshift.build.commit.id:
00cad392d39d5ef9117cbc8a31db0889eedd442
      io.openshift.build.commit.message: 'Merge pull request #51 from
php-coder/fix_url_and_sti'
      io.openshift.build.commit.ref: master
      io.openshift.build.image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e898
6b28e
      io.openshift.build.source-location:
https://github.com/openshift/ruby-hello-world.git
      io.openshift.builder-base-version: 8d95148
      io.openshift.builder-version:
8847438ba06307f86ac877465eadc835201241df

```

```

    io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
    io.openshift.tags: builder,ruby,ruby22
    io.s2i.scripts-url: image:///usr/libexec/s2i
    license: GPLv2
    name: CentOS Base Image
    vendor: CentOS
    User: "1001"
    WorkingDir: /opt/app-root/src
  Container:
86e9a4a3c760271671ab913616c51c9f3cea846ca524bf07c04a6f6c9e103a76
  ContainerConfig:
    AttachStdout: true
    Cmd:
      - /bin/sh
      - -c
      - tar -C /tmp -xf - && /usr/libexec/s2i/assemble
    Entrypoint:
      - container-entrpoint
    Env:
      - RACK_ENV=production
      - OPENSIFT_BUILD_NAME=ruby-sample-build-1
      - OPENSIFT_BUILD_NAMESPACE=test
      - OPENSIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-
world.git
      - EXAMPLE=sample-app
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
    ExposedPorts:
      8080/tcp: {}
    Hostname: ruby-sample-build-1-build
    Image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e898
6b28e
    OpenStdin: true
    StdinOnce: true
    User: "1001"
    WorkingDir: /opt/app-root/src
    Created: 2016-01-29T13:40:00Z
    DockerVersion: 1.8.2.fc21
    Id: 9d7fd5e2d15495802028c569d544329f4286dcd1c9c085ff5699218dbaa69b43
    Parent:
57b08d979c86f4500dc8cad639c9518744c8dd39447c055a3517dc9c18d6fccc
    Size: 441976279
    apiVersion: "1.0"
    kind: DockerImage
    dockerImageMetadataVersion: "1.0"
    dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7d
d13d

```

3.5.2.7. イメージストリームの使用

以下のセクションでは、イメージストリームおよびイメージストリームタグを使用する方法について説明します。イメージストリームの使用方法についての詳細は、「[イメージの管理](#)」を参照してください。

3.5.2.7.1. イメージストリームについての情報の取得

イメージストリームについての一般的な情報およびこれが参照するすべてのタグについての詳細情報を取得するには、以下のコマンドを使用します。

```
oc describe is/<image-name>
```

以下は例になります。

```
oc describe is/python

Name:      python
Namespace: default
Created:   About a minute ago
Labels:    <none>
Annotations: openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup: local=false
Unique Images: 1
Tags:      1

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
    About a minute ago
```

特定のイメージストリームタグについて利用可能なすべての情報を取得するには、以下を実行します。

```
oc describe istag/<image-stream>:<tag-name>
```

以下は例になります。

```
oc describe istag/python:latest

Image Name: sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Docker Image: centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Name:      sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Created:   2 minutes ago
Image Size: 251.2 MB (first layer 2.898 MB, last binary layer 72.26 MB)
Image Created: 2 weeks ago
Author:    <none>
Arch:      amd64
Entrypoint: container-entrypoint
Command:   /bin/sh -c $STI_SCRIPTS_PATH/usage
Working Dir: /opt/app-root/src
```



```
User: 1001
Exposes Ports: 8080/tcp
Docker Labels: build-date=20170801
```

**注記**

上記の表示されている以上の情報が出力されます。

3.5.2.7.2. 追加タグのイメージストリームへの追加

既存タグのいずれかを参照するタグを追加するには、**oc tag** コマンドを使用できます。

```
oc tag <image-name:tag> <image-name:tag>
```

以下は例になります。

```
oc tag python:3.5 python:latest
```

```
Tag python:latest set to
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25.
```

oc describe コマンドを使用して、イメージストリームに、外部 Docker イメージを参照するタグ (3.5) と、この最初のタグに基づいて作成されているために同じイメージを参照する別のタグ (**latest**) の 2 つのタグが含まれることを確認します。

```
oc describe is/python
```

```
Name: python
Namespace: default
Created: 5 minutes ago
Labels: <none>
Annotations: openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup: local=false
Unique Images: 1
Tags: 2

latest
  tagged from
  python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25

  * centos/python-35-
  centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
    About a minute ago

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
  centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
    5 minutes ago
```


3.5.2.7.3. 外部イメージのタグの追加

内部または外部イメージを参照する追加タグなど、タグ関連のすべての操作に **oc tag** コマンドを使用します。

```
oc tag <repository/image> <image-name:tag>
```

たとえば、このコマンドは **docker.io/python:3.6.0** イメージを **python** イメージストリームの **3.6** タグにマップします。

```
oc tag docker.io/python:3.6.0 python:3.6
Tag python:3.6 set to docker.io/python:3.6.0.
```

外部イメージのセキュリティが保護されている場合、そのレジストリーにアクセスするために認証情報を使ってシークレットを作成する必要があります。詳細については、「[プライベートレジストリーからのイメージのインポート](#)」を参照してください。

3.5.2.7.4. イメージストリームタグの更新

別のタグをイメージストリームに反映するようにタグを更新するには、以下を実行します。

```
oc tag <image-name:tag> <image-name:latest>
```

たとえば、以下では **3.6** タグをイメージタグに反映させるために **latest** タグを更新します。

```
oc tag python:3.6 python:latest
Tag python:latest set to
python@sha256:438208801c4806548460b27bd1fbc7bb188273d13871ab43f.
```

3.5.2.7.5. イメージストリームタグのイメージストリームからの削除

古いタグをイメージストリームから削除するには、以下を実行します。

```
oc tag -d <image-name:tag>
```

以下は例になります。

```
oc tag -d python:3.5

Deleted tag default/python:3.5.
```

3.5.2.7.6. タグの定期的なインポートの設定

外部 Docker レジストリーを使用している場合、(最新のセキュリティ更新を取得する場合などに) イメージを定期的に再インポートするには、**--scheduled** フラグを使用します。

```
oc tag <repository/image> <image-name:tag> --scheduled
```

以下は例になります。

```
oc tag docker.io/python:3.6.0 python:3.6 --scheduled
```

Tag python:3.6 set to import docker.io/python:3.6.0 periodically.

このコマンドにより、OpenShift Container Platform はこの特定のイメージストリームタグを定期的に更新します。この期間のデフォルト値はクラスター全体で 15 分に設定されます。

定期的なチェックを削除するには、上記のコマンド再実行しますが、**--scheduled** フラグを省略します。これにより、その動作がデフォルトに再設定されます。

```
oc tag <repository/image> <image-name:tag>
```

3.6. デプロイメント

3.6.1. レプリケーションコントローラー

レプリケーションコントローラーは、特定の数の Pod のレプリカが常に実行されるようにします。Pod が終了するか、または削除される場合、レプリケーションコントローラーは定義された数まで追加のインスタンス化を実行するように機能します。同様に、必要な数以上に実行されている場合、必要な数量に一致する数になるように追加分を削除します。

レプリケーションコントローラー設定は以下で構成されます。

1. 必要なレプリカ数 (ランタイム時に調整可能)。
2. 複製された Pod の作成時に使用する Pod 定義。
3. 管理された Pod を識別するためのセレクター。

セレクターは、レプリケーションコントローラーで管理される Pod に割り当てられるラベルのセットです。これらのラベルはレプリケーションコントローラーがインスタンス化する Pod 定義に組み込まれます。レプリケーションコントローラーはセレクターを使用して、必要に応じて調整を行うためにすでに実行されている Pod のインスタンス数を判別します。

レプリケーションコントローラーは負荷またはトラフィックに基づいて自動スケーリングを実行せず、追跡も行いません。レプリカ数は外部の自動スケーラーで調整される必要があります。

レプリケーションコントローラーは、**ReplicationController** というコアの Kubernetes オブジェクトです。

以下は、**ReplicationController** 定義のサンプルです。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 ①
  selector: ②
    name: frontend
  template: ③
    metadata:
      labels: ④
        name: frontend ⑤
```

```
spec:
  containers:
  - image: openshift/hello-openshift
    name: helloworld
    ports:
    - containerPort: 8080
      protocol: TCP
    restartPolicy: Always
```

- ❶ 実行する Pod のコピー数です。
- ❷ 実行する Pod のラベルセクターです。
- ❸ コントローラーが作成する Pod のテンプレートです。
- ❹ Pod のラベルにはラベルセクターの内容が含まれている必要があります。
- ❺ パラメーターの拡張後の名前の最大長は 63 文字です。

3.6.2. ジョブ

ジョブは、特定の理由のために Pod を作成することを目的としている点でレプリケーションコントローラーに似ています。相違点としては、レプリケーションコントローラーは継続的に実行されている Pod を対象としていますが、ジョブは 1 回だけ実行する (one-time) Pod を対象としています。ジョブは正常に完了している状態を追跡し、指定された完了数に達するとジョブ自体が完了します。

以下の例は、 π (Pi) を 2000 桁計算し、これを出力してから完了します。

```
apiVersion: extensions/v1
kind: Job
metadata:
  name: pi
spec:
  selector:
    matchLabels:
      app: pi
  template:
    metadata:
      name: pi
      labels:
        app: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
        restartPolicy: Never
```

ジョブの使用方法についての詳細は、「[ジョブ](#)」のトピックを参照してください。

3.6.3. デプロイメントおよびデプロイメント設定

レプリケーションコントローラーでビルドする OpenShift Container Platform は、デプロイメントの概念を使用したソフトウェアの開発およびデプロイメントライフサイクルの拡張サポートを追加しま

す。最も単純なケースでは、デプロイメントにより新規アプリケーションコントローラーのみが作成され、それが Pod を起動します。ただし、OpenShift Container Platform デプロイメントは、イメージの既存デプロイメントから新規デプロイメントに移行する機能も提供し、レプリケーションコントローラーの作成前後に実行するフックも定義します。

OpenShift Container Platform の **DeploymentConfig** オブジェクトは、デプロイメントについての以下の詳細を定義します。

1. **ReplicationController** 定義の要素。
2. 新規デプロイメントの自動作成のトリガー。
3. デプロイメント間の移行ストラテジー。
4. ライフサイクルフック。

デプロイメントがトリガーされる際には常に、手動または自動であるかを問わず、デプロイヤー Pod が (古いレプリケーションコントローラーの縮小、新規レプリケーションコントローラーの拡大およびフックの実行などの) デプロイメントを管理します。デプロイメント Pod は、デプロイメントのログを維持するためにデプロイメントの完了後は無期限で保持されます。デプロイメントが別のものに置き換えられる場合、以前のレプリケーションコントローラーは必要に応じて簡単なロールバックを有効にできるように保持されます。

デプロイメントの作成およびその対話方法についての詳細は、「[デプロイメント](#)」を参照してください。

以下は、いくつかの省略およびコールアウトを含む **DeploymentConfig** 定義のサンプルです。

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  replicas: 5
  selector:
    name: frontend
  template: { ... }
  triggers:
    - type: ConfigChange ❶
    - imageChangeParams:
        automatic: true
        containerNames:
          - helloworld
        from:
          kind: ImageStreamTag
          name: hello-openshift:latest
      type: ImageChange ❷
  strategy:
    type: Rolling ❸
```

- ❶ **ConfigChange** トリガーにより、新規デプロイメントがレプリケーションコントローラーテンプレートの変更時に常に作成されます。
- ❷ **ImageChange** トリガーにより、新規デプロイメントが、バッキングイメージの新規バージョンが名前付きイメージストリームで利用可能になると常に作成されます。

- 3 デフォルトのローリングストラテジーにより、デプロイメント間のダウンタイムなしの移行が実行されます。

3.7. テンプレート

3.7.1. 概要

テンプレートは、OpenShift Container Platform で作成されるオブジェクトの一覧を生成するためにパラメーター化され、処理される[オブジェクト](#)のセットを記述します。作成するオブジェクトには、ユーザーがプロジェクト内で作成するパーミッションを持つすべてのものが含まれます。たとえば、[サービス](#)、[ビルド設定](#)、および[デプロイメント設定](#)などが含まれます。テンプレートは、テンプレートで定義されるすべてのオブジェクトに適用される[ラベル](#)のセットも定義します。

テンプレートの作成および使用についての詳細は、[テンプレートについてのガイド](#)を参照してください。

第4章 追加の概念

4.1. 認証

4.1.1. 概要

認証層は、OpenShift Container Platform API への要求に関連付けられたユーザーを識別します。次に、認証層は要求が許可されるかどうかを判断するために要求側のユーザーについての情報を使用します。

管理者として、[マスター設定ファイル](#)を使用して、[認証の設定](#)を実行できます。

4.1.2. ユーザーおよびグループ

OpenShift Container Platform の **ユーザー** は、OpenShift Container Platform API に対して要求を実行できるエンティティーです。通常、これは OpenShift Container Platform と対話する開発者または管理者のアカウントを表します。

ユーザーは 1 つ以上の **グループ** に割り当てることができます。それぞれのグループはユーザーの特定のセットを表します。グループは、[許可ポリシーを管理](#)して、一度に複数ユーザーにパーミッションを付与する場合などに役立ちます。たとえば、個々のユーザーにパーミッションを付与するのではなく、[プロジェクト内のオブジェクト](#)へのアクセスを許可します。

明示的に定義されるグループのほかにも、システムグループまたは **仮想グループ** が OpenShift で自動的にプロビジョニングされます。これらは、[クラスターのバインディングを表示](#)する際に確認できます。

仮想グループのデフォルトセットでは、とくに以下の点に留意してください。

仮想グループ	説明
system:authenticated	認証されたすべてのユーザーに自動的に関連付けられます。
system:authenticated:oauth	OAuth アクセストークンで認証されたすべてのユーザーに自動的に関連付けられます。
system:unauthenticated	認証されていないすべてのユーザーに自動的に関連付けられます。

4.1.3. API 認証

OpenShift Container Platform API への要求は以下の方法を使用して認証されます。

OAuth アクセストークン

- **<master>/oauth/authorize** および **<master>/oauth/token** エンドポイントを使用して OpenShift Container Platform OAuth サーバーから取得されます。
- **Authorization: Bearer...** ヘッダーとして送信されます。
- OpenShift Container Platform サーバーのバージョン 3.6 よりも前のバージョンでは、websocket 要求の **access_token=...** クエリーパラメーターとして送信されます。

- OpenShift Container Platform サーバーのバージョン 3.6 以降では、websocket 要求の **base64url.bearer.authorization.k8s.io.<base64url-encoded-token>** 形式の websocket サブプロトコルヘッダーとして送信されます。

X.509 クライアント証明書

- API サーバーへの HTTPS 接続を要求します。
- 信頼される認証局バンドルに対して API サーバーによって検証されます。
- API サーバーは証明書を作成し、これをコントローラーに配布してそれらを認証できるようにします。

無効なアクセストークンまたは無効な証明書での要求は認証層によって拒否され、401 エラーが出されます。

アクセストークンまたは証明書が提示されない場合、認証層は **system:anonymous** 仮想ユーザーおよび **system:unauthenticated** 仮想グループを要求に割り当てます。これにより、認証層は匿名ユーザーが実行できる要求 (ある場合) を判別できます。

4.1.3.1. 権限の借用

OpenShift Container Platform API への要求には **Impersonate-User** (ユーザーの代理) ヘッダーを含めることができます。これは、要求を指定ユーザーからの要求であるかのように処理されることを要求側のユーザーが指定していることを示します。このユーザー権限の借用は、**--as=<user>** フラグを要求に追加して実行できます。

ユーザー A がユーザー B の権限を借用する前に、ユーザー A は認証されます。次に承認チェックが実行され、ユーザー A がユーザー B というユーザーの権限を借用することが許可されていることが確認されます。ユーザー A がサービスアカウント **system:serviceaccount:namespace:name** の権限を借用することを要求している場合、OpenShift Container Platform はユーザー A が **namespace** の **name** という **serviceaccount** の権限を借用できることを確認します。チェックに失敗すると、要求は 403 (Forbidden) エラーコードを出して失敗します。

デフォルトで、プロジェクト管理者およびエディターはそれらの namespace のサービスアカウントの権限を借用できます。**sudoer** ロールにより、ユーザーは **system:admin** の権限を借用できます。これにはクラスター管理者のパーミッションがあります。**system:admin** の権限を借用できる機能は、タイプミスの発生を防ぐ保護を提供しますが、クラスター管理者のセキュリティの保護を提供するものではありません。たとえば、**oc delete nodes --all** の実行が失敗しても、**oc delete nodes --all --as=system:admin** の実行は成功します。以下のコマンドを実行すると、ユーザーにこのパーミッションを付与できます。

```
$ oc create clusterrolebinding <any_valid_name> --clusterrole=sudoer --user=<username>
```

ユーザーの代わりにプロジェクト要求を作成する必要がある場合、**--as=<user> --as-group=<group1> --as-group=<group2>** フラグをコマンドに組み込みます。**system:authenticated:oauth** はプロジェクト要求を作成できる唯一のブートストラップグループであるため、以下の例に示されるようにそのグループの権限を借用する必要があります。

```
$ oc new-project <project> --as=<user> \
--as-group=system:authenticated --as-group=system:authenticated:oauth
```

4.1.4. OAuth

OpenShift Container Platform マスターには、ビルトイン OAuth サーバーが含まれます。ユーザーは OAuth アクセストークンを取得して、API に対して認証を実行します。

新規の OAuth トークンが要求されると、OAuth サーバーは設定済みの[アイデンティティプロバイダー](#)を使用して要求したユーザーのアイデンティティを判別します。

次に、そのアイデンティティがマップするユーザーを判別し、そのユーザーのアクセストークンを作成し、そのトークンを使用できるように返します。

4.1.4.1. OAuth クライアント

OAuth トークンのすべての要求は、トークンを受信し、使用する OAuth クライアントを指定する必要があります。以下の OAuth クライアントは、OpenShift Container Platform API の起動時に自動的に作成されます。

OAuth クライアント	使用法
openshift-web-console	Web コンソールのトークンを要求します。
openshift-browser-client	対話式ログインを処理できるユーザーエージェントを使用して <master>/oauth/token/request でトークンを要求します。
openshift-challenging-client	WWW-Authenticate チャレンジを処理できるユーザーエージェントでトークンを要求します。

追加のクライアントを登録するには、以下を実行します。

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
  name: demo ①
secret: "... " ②
redirectURIs:
  - "http://www.example.com/" ③
grantMethod: prompt ④
')
```

- ① **<master>/oauth/authorize** および **<master>/oauth/token** への要求を実行する際に、OAuth クライアントの **name** が **client_id** パラメーターとして使用されます。
- ② **<master>/oauth/token** への要求の実行時に、**secret** は **client_secret** パラメーターとして使用されます。
- ③ **<master>/oauth/authorize** および **<master>/oauth/token** への要求で指定される **redirect_uri** パラメーターは、**redirectURIs** のいずれかに等しくなければなりません (またはそのいずれかで開始されていなければなりません)。
- ④ **grantMethod** は、このクライアントがトークンを要求する際に、ユーザーによるアクセスの付与が行われていない場合に実行するアクションを決定します。「Grant Options」に表示されるものと同じ値を使用します。

4.1.4.2. OAuth クライアントとしてのサービスアカウント

サービスアカウントは、OAuth クライアントの制約のある形態で使用できます。サービスアカウントは一部の基本ユーザー情報へのアクセスを許可するスコープのサブセットと、サービスアカウント自体の namespace 内のロールベースの権限のみを要求できます。

- **user:info**
- **user:check-access**
- **role:<any_role>:<serviceaccount_namespace>**
- **role:<any_role>:<serviceaccount_namespace>:!**

サービスアカウントを OAuth クライアントとして使用する場合:

- **client_id** は **system:serviceaccount:<serviceaccount_namespace>:<serviceaccount_name>** になります。
- **client_secret** には、サービスアカウントの API トークンのいずれかを指定できます。以下は例になります。

```
$ oc sa get-token <serviceaccount_name>
```

- **WWW-Authenticate** チャレンジを取得するには、サービスアカウントの **serviceaccounts.openshift.io/oauth-want-challenges** アノテーションを **true** に設定します。
- **redirect_uri** はサービスアカウントのアノテーションに一致する必要があります。詳細は、「[OAuth クライアントとしてのサービスアカウントのリダイレクト URI](#)」を参照してください。

4.1.4.3. OAuth クライアントとしてのサービスアカウントのリダイレクト URI

アノテーションキーには、以下のようにプレフィックスの **serviceaccounts.openshift.io/oauth-redirecturi**. または **serviceaccounts.openshift.io/oauth-redirectreference**. が含まれる必要があります。

```
serviceaccounts.openshift.io/oauth-redirecturi.<name>
```

最も単純な形態として、有効なりダイレクト URI を直接指定するためにアノテーションを使用できません。以下は例になります。

```
"serviceaccounts.openshift.io/oauth-redirecturi.first":
"https://example.com"
"serviceaccounts.openshift.io/oauth-redirecturi.second":
"https://other.com"
```

上記の例の **first** および **second** ポストフィックスは 2 つの有効なりダイレクト URI を分離するために使用されます。

複雑な設定の場合、静的なりダイレクト URI のみでは不十分な場合があります。たとえば、ルートのすべての ingress が有効とみなされるようにする必要があります。この場合、**serviceaccounts.openshift.io/oauth-redirectreference**. プレフィックスを使用した動的なりダイレクト URI を使用できます。

以下は例になります。

```
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

このアノテーションの値にはシリアル化された JSON データが含まれるため、拡張フォーマットを使用するとより容易に表示できます。

```
{
  "kind": "OAuthRedirectReference",
  "apiVersion": "v1",
  "reference": {
    "kind": "Route",
    "name": "jenkins"
  }
}
```

ここでは、**OAuthRedirectReference** により **jenkins** という名前のルートを参照できます。そのため、そのルートのすべての ingress は有効とみなされます。**OAuthRedirectReference** の詳細な仕様は以下のようにになります。

```
{
  "kind": "OAuthRedirectReference",
  "apiVersion": "v1",
  "reference": {
    "kind": ..., ①
    "name": ..., ②
    "group": ... ③
  }
}
```

- ① **kind** は参照されているオブジェクトのタイプを参照します。現時点では、**route** のみがサポートされています。
- ② **name** はオブジェクトの名前を参照します。このオブジェクトはサービスアカウントと同じ namespace にある必要があります。
- ③ **group** はオブジェクトのグループを参照します。ルートのグループは空の文字列であるため、これを空白のままにします。

両方のアノテーションのプレフィックスも組み合わせて、参照オブジェクトで提供されるデータを上書きできます。以下は例になります。

```
"serviceaccounts.openshift.io/oauth-redirecturi.first": "custompath"
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

first ポストフィックスはアノテーションを関連付けるために使用されます。**jenkins** ルートに **https://example.com** の ingress がある場合、**https://example.com/custompath** が有効とみなされますが、**https://example.com** は有効とみなされません。上書きデータを部分的に指定するためのフォーマットは以下のようにになります。

タイプ	構文
スキーム	"https://"
ホスト名	"//website.com"
ポート	"//:8000"
パス	"examplepath"



注記

ホスト名の上書きを指定すると、参照されるオブジェクトのホスト名データが置き換わりますが、これは望ましい動作ではありません。

上記の構文のいずれも、以下のフォーマットで組み合わせることができます。

<scheme>://<hostname>:<port>/<path>

柔軟性を持たせるように、同じオブジェクトを複数回参照することができます。

```
"serviceaccounts.openshift.io/oauth-redirecturi.first": "custompath"
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{"kind\":\"Route\",\"name\":\"jenkins\"}}"
"serviceaccounts.openshift.io/oauth-redirecturi.second": "//:8000"
"serviceaccounts.openshift.io/oauth-redirectreference.second": "
{"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{"kind\":\"Route\",\"name\":\"jenkins\"}}"
```

jenkins という名前のルートに **https://example.com** の ingress がある場合、**https://example.com:8000** と **https://example.com/custompath** の両方が有効とみなされます。

必要な動作を得るために、静的で動的なアノテーションを同時に使用できます。

```
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{"kind\":\"OAuthRedirectReference\",\"apiVersion\":\"v1\",\"reference\":
{"kind\":\"Route\",\"name\":\"jenkins\"}}"
"serviceaccounts.openshift.io/oauth-redirecturi.second":
"https://other.com"
```

4.1.4.3.1. OAuth の API イベント

API サーバーは、API マスターログへの直接的なアクセスがないとデバッグが困難な **unexpected condition** のエラーメッセージを返すことがあります。このエラーの根本的な理由は意図的に非表示にされます。認証されていないユーザーにサーバーの状態についての情報を提供することを避けるためです。

これらのエラーのサブセットは、サービスアカウントの OAuth 設定の問題に関連するものです。これらの問題は、管理者以外のユーザーが確認できるイベントでキャプチャーされます。**unexpected condition** というサーバーエラーが OAuth の実行時に発生する場合、**oc get events** を実行し、こ

これらのイベントについて **ServiceAccount** で確認します。

以下の例では、適切な OAuth リダイレクト URI がないサービスアカウントについて警告しています。

```
$ oc get events | grep ServiceAccount
1m          1m          1          proxy          ServiceAccount
Warning     NoSAOAuthRedirectURIs  service-account-oauth-client-getter
system:serviceaccount:myproject:proxy has no redirectURIs; set
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or
create a dynamic URI using serviceaccounts.openshift.io/oauth-
redirectreference.<some-value>=<reference>
```

oc describe sa/<service-account-name> を実行すると、指定のサービスアカウント名に関連付けられた OAuth イベントが報告されます。

```
$ oc describe sa/proxy | grep -A5 Events
Events:
  FirstSeen      LastSeen        Count   From              SubObjectPath      Type            Message
  -----
  3m             3m             1       service-account-oauth-client-getter
Warning         NoSAOAuthRedirectURIs
system:serviceaccount:myproject:proxy has no redirectURIs; set
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or
create a dynamic URI using serviceaccounts.openshift.io/oauth-
redirectreference.<some-value>=<reference>
```

以下は発生する可能性のあるイベントエラーの一覧です。

リダイレクト URI アノテーションが指定されていないか、無効な URI が指定されている

Reason	Message
NoSAOAuthRedirectURIs	system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>

無効なルートが指定されている

Reason	Message
NoSAOAuthRedirectURIs	[routes.route.openshift.io "<name>" not found, system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>]

無効な参照タイプが指定されている

Reason	Message
NoSAOAuthRedirectURIs	[no kind "<name>" is registered for version "v1", system:serviceaccount:myproject:proxy has no redirectURIs; set

```
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or
create a dynamic URI using serviceaccounts.openshift.io/oauth-
redirectreference.<some-value>=<reference>]
```

SA トークンがない

Reason	Message
NoSAOAuthTokens	system:serviceaccount:myproject:proxy has no tokens

4.1.4.3.1.1. 設定ミスによって生じる API イベントのサンプル

以下の手順は、障害が発生する場合に問題を解決する方法を示しています。

1. サービスアカウントを OAuth クライアントとして使用してプロジェクトを作成します。
 - a. プロキシサービスアカウントオブジェクトの YAML を作成し、これがルートの **proxy** を使用することを確認します。

```
vi serviceaccount.yaml
```

以下のサンプルコードを追加します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: proxy
  annotations:
    serviceaccounts.openshift.io/oauth-redirectreference.primary:
      '{"kind": "OAuthRedirectReference", "apiVersion": "v1", "reference":
        {"kind": "Route", "name": "proxy"}}'
```

- b. プロキシへのセキュアな接続を作成するために、ルートオブジェクトの YAML を作成します。

```
vi route.yaml
```

以下のサンプルコードを追加します。

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: proxy
spec:
  to:
    name: proxy
  tls:
    termination: Reencrypt
apiVersion: v1
kind: Service
metadata:
  name: proxy
  annotations:
    service.alpha.openshift.io/serving-cert-secret-name: proxy-
```

```

tls
spec:
  ports:
    - name: proxy
      port: 443
      targetPort: 8443
  selector:
    app: proxy

```

- c. プロキシをサイドカーとして起動するために、デプロイメント設定の YAML を作成します。

```
vi proxysidecar.yaml
```

以下のサンプルコードを追加します。

```

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: proxy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: proxy
  template:
    metadata:
      labels:
        app: proxy
    spec:
      serviceAccountName: proxy
      containers:
        - name: oauth-proxy
          image: openshift/oauth-proxy:v1.0.0
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8443
              name: public
          args:
            - --https-address=:8443
            - --provider=openshift
            - --openshift-service-account=proxy
            - --upstream=http://localhost:8080
            - --tls-cert=/etc/tls/private/tls.crt
            - --tls-key=/etc/tls/private/tls.key
            - --cookie-secret=SECRET
          volumeMounts:
            - mountPath: /etc/tls/private
              name: proxy-tls

        - name: app
          image: openshift/hello-openshift:latest
      volumes:

```

```
- name: proxy-tls
  secret:
    secretName: proxy-tls
```

d. オブジェクトを作成します。

```
oc create -f serviceaccount.yaml
oc create -f route.yaml
oc create -f proxysidecar.yaml
```

2. **oc edit sa/proxy** を実行してサービスアカウントを編集し、**serviceaccounts.openshift.io/oauth-redirectreference** アノテーションを、存在しないルートを参照するように変更します。

```
apiVersion: v1
imagePullSecrets:
- name: proxy-dockercfg-08d5n
kind: ServiceAccount
metadata:
  annotations:
    serviceaccounts.openshift.io/oauth-redirectreference.primary:
      '{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
        {"kind":"Route","name":"notexist"}}'
    ...
```

3. OAuth ログでサービスを確認し、サーバーエラーを見つけます。

```
The authorization server encountered an unexpected condition that
prevented it from fulfilling the request.
```

4. **oc get events** を実行して **ServiceAccount** イベントを表示します。

```
oc get events | grep ServiceAccount

23m          23m          1          proxy
ServiceAccount                                     Warning
NoSAOAuthRedirectURIs  service-account-oauth-client-getter
[routes.route.openshift.io "notexist" not found,
system:serviceaccount:myproject:proxy has no redirectURIs; set
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=
<redirect> or create a dynamic URI using
serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=
<reference>]
```

4.1.4.4. 統合

OAuth トークンのすべての要求には **<master>/oauth/authorize** への要求が必要になります。ほとんどの認証統合では、認証プロキシをこのエンドポイントの前に配置するか、または OpenShift Container Platform を、サポートする [アイデンティティプロバイダー](#) に対して認証情報を検証するように設定します。**<master>/oauth/authorize** の要求は、CLI などの対話式ログインページを表示できないユーザーエージェントから送られる場合があります。そのため、OpenShift Container Platform は、対話式ログインフローのほかにも **WWW-Authenticate** チャレンジを使用した認証をサポートします。

認証プロキシが **<master>/oauth/authorize** エンドポイントの前に配置される場合、対話式ログインページを表示したり、対話式ログインフローにリダイレクトする代わりに、認証されていないブラウザ以外のユーザーエージェントの **WWW-Authenticate** チャレンジを送信します。

注記

ブラウザクライアントに対するクロスサイトリクエストフォージェリー (CSRF) 攻撃を防止するため、基本的な認証チャレンジは **X-CSRF-Token** ヘッダーが要求に存在する場合にのみ送信されます。基本的な **WWW-Authenticate** チャレンジを受信する必要があるクライアントでは、このヘッダーを空でない値に設定する必要があります。

認証プロキシが **WWW-Authenticate** チャレンジをサポートしないか、または OpenShift Container Platform が WWW-Authenticate チャレンジをサポートしないアイデンティティプロバイダーを使用するように設定されている場合、ユーザーはブラウザで **<master>/oauth/token/request** にアクセスし、アクセストークンを手動で取得できます。

4.1.4.5. OAuth サーバーメタデータ

OpenShift Container Platform で実行されているアプリケーションは、ビルトイン OAuth サーバーについての情報を検出する必要がある場合があります。たとえば、それらは **<master>** サーバーのアドレスを手動の設定なしで検出する必要があります。これを支援するために、OpenShift Container Platform は IETF の [OAuth 2.0 Authorization Server Metadata](#) のドラフト仕様を実装しています。

そのため、クラスター内で実行されているすべてのアプリケーションは、**https://openshift.default.svc/.well-known/oauth-authorization-server** に対して **GET** 要求を実行し、以下の情報を取得できます。

```
{
  "issuer": "https://<master>", ①
  "authorization_endpoint": "https://<master>/oauth/authorize", ②
  "token_endpoint": "https://<master>/oauth/token", ③
  "scopes_supported": [ ④
    "user:full",
    "user:info",
    "user:check-access",
    "user:list-scoped-projects",
    "user:list-projects"
  ],
  "response_types_supported": [ ⑤
    "code",
    "token"
  ],
  "grant_types_supported": [ ⑥
    "authorization_code",
    "implicit"
  ],
  "code_challenge_methods_supported": [ ⑦
    "plain",
    "S256"
  ]
}
```


- 1 **https** スキームを使用し、クエリーまたはフラグメントコンポーネントがない認可サーバーの発行者 ID です。これは、認可サーバーについての情報が含まれる **.well-known RFC 5785** リソースが公開される場所です。
- 2 認可サーバーの認可エンドポイントの URL です。RFC 6749 を参照してください。
- 3 認可サーバーのトークンエンドポイントの URL です。RFC 6749 を参照してください。
- 4 この認可サーバーがサポートする OAuth 2.0 RFC 6749 スコープの値の一覧を含む JSON 配列です。サポートされるスコープの値すべてが公開される訳ではないことに注意してください。
- 5 この認可サーバーがサポートする OAuth 2.0 **response_type** 値の一覧を含む JSON 配列です。使用される配列の値は、RFC 7591 の OAuth 2.0 Dynamic Client Registration Protocol で定義される **response_types** パラメーターで使用されるものと同じです。
- 6 この認可サーバーがサポートする OAuth 2.0 grant type の値の一覧が含まれる JSON 配列です。使用される配列の値は、RFC 7591 の **OAuth 2.0 Dynamic Client Registration Protocol** で定義される **grant_types** パラメーターで使用されるものと同じです。
- 7 この認可サーバーでサポートされる PKCE RFC 7636 コードのチャレンジメソッドの一覧が含まれる JSON 配列です。コードのチャレンジメソッドの値は、RFC 7636 のセクション 4.3 で定義される **code_challenge_method** パラメーターで使用されます。有効なコードのチャレンジメソッドの値は、IANA **PKCE Code Challenge Method** レジストリーで登録される値です。「IANA OAuth パラメーター」を参照してください。

4.1.4.6. OAuth トークンの取得

OAuth サーバーは、標準的な **Authorization Code Grant (認可コードによるグラント)** および **Implicit Grant (暗黙的グラント)** の OAuth 認証フローをサポートします。

OAuth トークンを、(**openshift-challenging-client** などの) **WWW-Authenticate challenge** を要求するように設定された **client_id** で **Implicit Grant (暗黙的グラント) フロー (response_type=token)** を使用して要求する場合、以下が **/oauth/authorize** から送られる可能性のあるサーバー応答とそれらの処理方法になります。

ステータス	内容	クライアントの応答
302	Location ヘッダーに URL フラグメントの access_token パラメーターが含まれます (RFC 4.2.2)。	access_token 値を OAuth トークンとして使用します。
302	Location ヘッダーに error クエリーパラメーターが含まれます (RFC 4.1.2.1)。	失敗します。オプションで error (およびオプションの error_description) クエリー値をユーザーに表示します。
302	他の Location ヘッダー	リダイレクトに従い、これらのルールを使用して結果を処理します。
401	WWW-Authenticate ヘッダーが存在します。	タイプ (Basic 、 Negotiate など) が認識される場合にチャレンジに応答し、要求を再送信し、これらのルールを使用して結果を処理します。

ステータス	内容	クライアントの応答
401	WWW-Authenticate ヘッダーがありません。	チャレンジの認証ができません。失敗し、応答本体を表示します (これにはリンク、または OAuth トークンを取得する別の方法についての詳細が含まれる可能性があります)。
その他	その他	失敗し、オプションで応答の本体をユーザーに表示します。

4.1.4.7. Prometheus の認証メトリクス

OpenShift Container Platform は認証の試行中に以下の Prometheus システムメトリクスをキャプチャーします。

- **openshift_auth_basic_password_count** は **oc login** ユーザー名およびパスワードの試行回数をカウントします。
- **openshift_auth_basic_password_count_result** は **oc login** ユーザー名およびパスワードの試行回数を結果 (成功またはエラー) 別にカウントします。
- **openshift_auth_form_password_count** は Web コンソールのログイン試行回数をカウントします。
- **openshift_auth_form_password_count_result** は Web コンソールのログイン試行回数を結果 (成功またはエラー) 別にカウントします。
- **openshift_auth_password_total** は **oc login** および Web コンソールのログイン試行の合計数をカウントします。

4.2. 承認

4.2.1. 概要

Role-based Access Control (RBAC: ロールベースアクセス制御) オブジェクトは、ユーザーがプロジェクト内の所定の [アクション](#) を実行することが許可されるかどうかを決定します。

これにより、プラットフォーム管理者は [クラスターロール](#) および [バインディング](#) を使用して、OpenShift Container Platform プラットフォーム自体およびすべてのプロジェクトへの各種のアクセスレベルを持つユーザーを制御できます。

さらに開発者は [ローカルロール](#) および [バインディング](#) を使用し、[プロジェクト](#) へのアクセスを持つユーザーを制御できます。承認は [認証](#) とは異なる手順であることに注意してください。認証については、アクションを実行するユーザーのアイデンティティを判別することと関係があります。

承認は以下を使用して管理されます。

ルール	オブジェクト のセットで許可された 動詞 を設定します。たとえば、これには Pod の create を実行できるかどうかなどが含まれます。
-----	---

ロール	ルールのコレクションです。 ユーザーおよびグループ を一度に複数の ロール に関連付けたり、 バインド したりできます。
バインディング	ユーザーおよび/またはグループと ロール の関連付けです。

クラスター管理者は、[CLI の使用](#)によりルール、ロールおよびバインディングを可視化できます。

たとえば、**admin** および **basic-user** の [デフォルトクラスターロール](#)のルールセットを示す以下の抜粋を見てみましょう。

```
$ oc describe clusterrole.rbac admin basic-user
```

```
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]
  replicaset.extensions [] [] [create delete deletecollection get
list patch update watch]
  replicaset.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]
  resourcequotausages [] [] [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]

ローカルロールバインディングを表示して得られる以下の抜粋は、上記のロールが各種のユーザーおよびグループにバインドされていることを示しています。

```
oc describe rolebinding.rbac admin basic-user -n alice-project
```

Name: admin

Labels: <none>

Annotations: <none>

Role:

Kind: ClusterRole

Name: admin

Subjects:

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

User system:admin

User alice

Name: basic-user

Labels: <none>

Annotations: <none>

Role:

Kind: ClusterRole

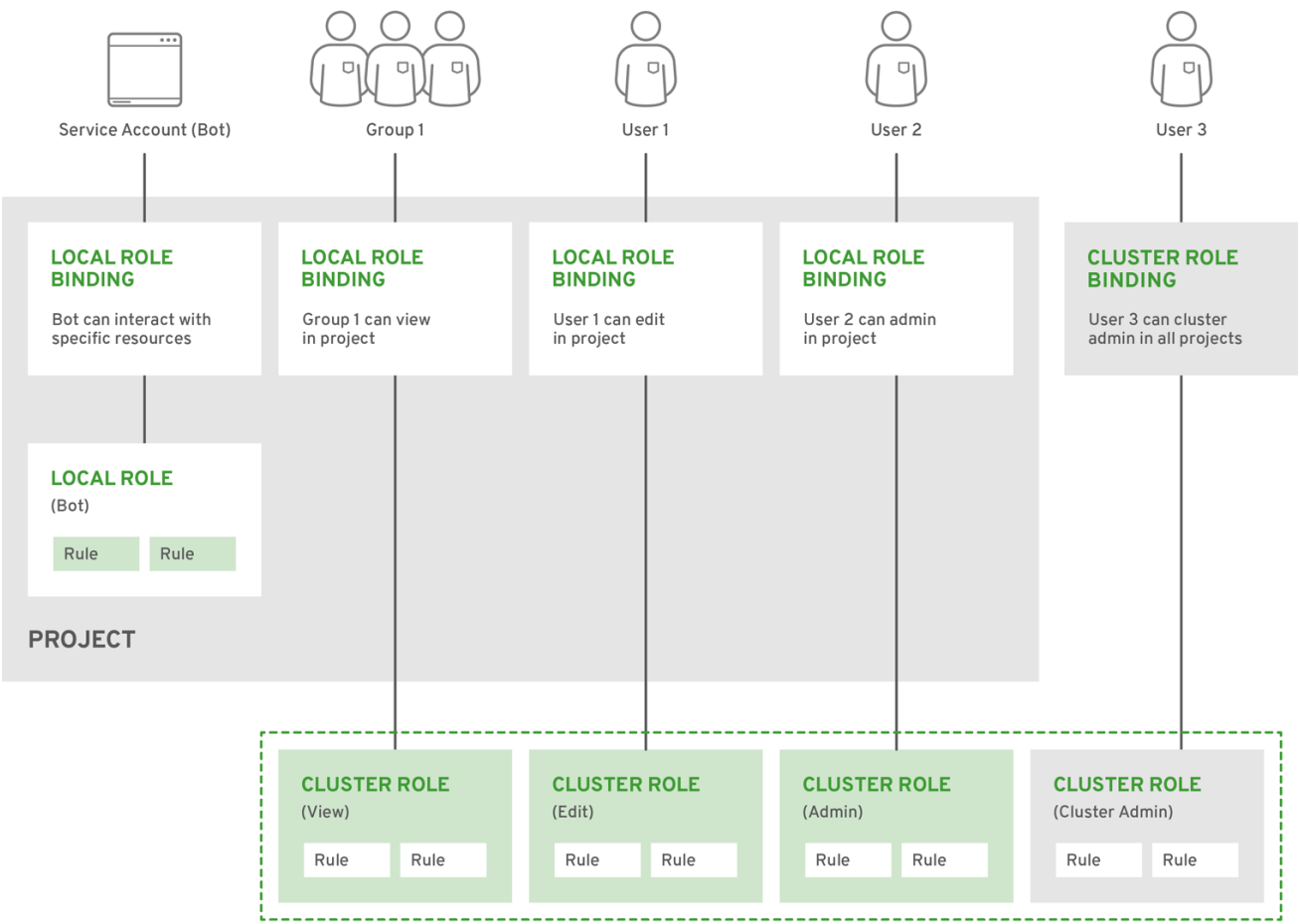
Name: basic-user

Subjects:

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

User joe
Group devel

クラスターロール、クラスターロールのバインディング、ローカルロールのバインディング、ユーザー、グループおよびサービスアカウントの関係は以下に説明されています。



OPENSIFT_415489_0218

4.2.2. 承認の評価

OpenShift Container Platform が承認を評価する際、いくつかのファクターを組み合わせで意思決定が行われます。

アイデンティティ	承認のコンテキストでは、ユーザー名およびユーザーが属するグループの一覧になります。		
アクション	実行されるアクションです。ほとんどの場合、これは以下で構成されます。		
	プロジェクト	アクセスされる プロジェクト です。	
	動詞	get、list、create、update、delete、deletecollection または watch を使用できます。	
	リソース名	アクセスされる API エンドポイントです。	

バインディング	バインディングの詳細一覧です。
---------	-----------------

OpenShift Container Platform は以下の手順を使って承認を評価します。

1. アイデンティティおよびプロジェクトでスコープ設定されたアクションは、ユーザーまたはそれらのグループに適用されるすべてのバインディングを検索するために使用されます。
2. バインディングは、適用されるすべてのロールを見つけるために使用されます。
3. ロールは、適用されるすべてのルールを見つけるために使用されます。
4. 一致を見つけるために、アクションが各ルールに対してチェックされます。
5. 一致するルールが見つからない場合、アクションはデフォルトで拒否されます。

4.2.3. クラスターおよびローカル RBAC

承認を制御する 2 つのレベルの RBAC ロールおよびバインディングがあります。

クラスター RBAC	すべてのプロジェクトで適用可能なルールおよびバインディングです。クラスター全体で存在するロールはクラスターロールとみなされます。クラスターロールのバインディングはクラスターロールのみを参照できます。
ローカル RBAC	所定のプロジェクトにスコープ設定されているルールおよびバインディングです。プロジェクトにのみ存在するロールはローカルロールとみなされます。ローカルロールのバインディングはクラスターロールおよびローカルロールの両方を参照できます。

この 2 つのレベルからなる階層により、ローカルロールによる個別プロジェクト内のカスタマイズが可能になる一方で、クラスターロールによる複数プロジェクト間での再利用が可能になります。

評価時に、クラスターロールのバインディングおよびローカルロールのバインディングの両方が使用されます。以下は例になります。

1. クラスター全体の「allow」ルールがチェックされます。
2. ローカルにバインドされた「allow」ルールがチェックされます。
3. デフォルトで拒否します。

4.2.4. クラスターロールおよびローカルロール

ロールはポリシールールのコレクションであり、一連のリソースで実行可能な一連の許可された動詞です。OpenShift Container Platform には、クラスター全体またはローカルでユーザーおよびグループにバインドできるデフォルトのクラスターロールのセットが含まれます。

デフォルトのクラスターロール	説明
----------------	----

デフォルトのクラスターロール	説明
admin	プロジェクトマネージャーです。 ローカルバインディング で使用されている場合、 admin ユーザーにはプロジェクトのリソースを閲覧し、クォータを除くプロジェクトのすべてのリソースを変更する権限があります。
basic-user	プロジェクトおよびユーザーについての基本的な情報を取得できるユーザーです。
cluster-admin	プロジェクトですべてのアクションを実行できるスーパーユーザーです。 ローカルバインディング でユーザーにバインドされる場合、クォータおよびプロジェクトのすべてのリソースに対するすべてのアクションに対する フルコントロール があります。
cluster-status	基本的なクラスターのステータス情報を取得できるユーザーです。
edit	プロジェクトのほとんどのオブジェクトを変更できるが、ロールまたはバインディングを表示したり、変更したりする機能を持たないユーザーです。
self-provisioner	独自のプロジェクトを作成できるユーザーです。
view	変更はできないが、プロジェクトでほとんどのオブジェクトを確認できるユーザーです。これらのユーザーはロールまたはバインディングを表示したり、変更したりできません。

ヒント

[ユーザーおよびグループ](#)は複数のロールに同時に関連付けたり、[バインド](#) できることに注意してください。

プロジェクト管理者は、CLI を使用して関連付けられる動詞およびリソースのマトリックスを含むロールを可視化し、[ローカルバインディングを表示](#)できます。



重要

プロジェクト管理者にバインドされるクラスターロールは、[ローカルバインディング](#)によってプロジェクトに制限されます。これは、**cluster-admin** または **system:admin** に付与されるクラスターロールの場合のように[クラスター全体](#)でバインドされる訳ではありません。

クラスターロールは、クラスターレベルで定義される[ロール](#)ですが、クラスターレベルでバインドすることも、プロジェクトレベルでバインドすることもできます。

[プロジェクトのローカルロールの作成方法についてはこちら](#)を参照してください。

4.2.4.1. クラスターロールの更新

OpenShift Container Platform クラスターの[アップグレード](#)後に、デフォルトのロールは更新され、サーバーの起動時に自動的に調整されます。以下を使用した他の推奨方法についての詳細は、「[ポリシー定義の更新](#)」を参照してください。

```
$ oc adm policy reconcile-cluster-roles
```

4.2.4.2. カスタムロールおよびパーミッションの適用

カスタムロールおよびパーミッションを追加するか、または更新するには、以下のコマンドを使用することを強く推奨します。

```
# oc auth reconcile -f FILE
```

このコマンドは、他のクライアントを切断しない方法で新規パーミッションが適切に適用されるようにします。

4.2.4.3. クラスターロールの集計

デフォルトのクラスターの管理、編集および表示ロールは、[クラスターロールの集計](#)をサポートします。ここでは、各ロールのクラスターロールは新規ロールの作成時に動的に更新されます。この機能は、[カスタムリソースを作成](#)して Kubernetes API を拡張する場合にのみ関連します。

[クラスターロールの集計の使用方法についてはこちら](#)を参照してください。

4.2.5. SCC (Security Context Constraints)

ユーザーの実行できる内容を制御する [RBAC リソース](#)のほかに、OpenShift Container Platform は [Pod](#) が実行できる内容および Pod がアクセスできる内容を制御する **SCC (security context constraints)** を提供します。管理者は CLI を使用して [SCC を管理](#)することができます。

SCC は、永続ストレージへのアクセスを管理する場合にも非常に便利です。

SCC は、システムで許可されるために Pod の実行時に必要となる一連の条件を定義するオブジェクトです。管理者は以下を制御できます。

1. [特権付きコンテナー](#)の実行
2. コンテナーが追加を要求できる機能
3. ホストディレクトリーのボリュームとしての使用
4. コンテナーの SELinux コンテキスト
5. ユーザー ID
6. ホストの namespace およびネットワークの使用
7. Pod のボリュームを所有する **FSGroup** の割り当て
8. 許可される補助グループの設定
9. 読み取り専用のルートファイルシステムの要求
10. ボリュームタイプの使用の制御
11. 許可される seccomp プロファイルの設定

デフォルトでは、7 つの SCC がクラスターに追加され、クラスター管理者は CLI を使用してそれらを表示できます。

```
$ 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 secret]
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]
```



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、OpenShift Container Platform のアップグレード時に問題が発生する可能性があります。その代わりに、[新規 SCC を作成](#)します。

各 SCC の定義についても、クラスター管理者は CLI を使用して表示できます。たとえば、特権付き SCC の場合は、以下ようになります。

```
# oc export scc/privileged
allowHostDirVolumePlugin: true
allowHostIPC: true
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
allowPrivilegedContainer: true
allowedCapabilities: ❶
- '*'
apiVersion: v1
defaultAddCapabilities: [] ❷
fsGroup: ❸
  type: RunAsAny
groups: ❹
- system:cluster-admins
- system:nodes
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: 'privileged allows access to all privileged
and host
```

```

    features and the ability to run as any user, any group, any fsGroup,
and with
    any SELinux context. WARNING: this is the most relaxed SCC and
should be used
    only for cluster administration. Grant with caution.'
    creationTimestamp: null
    name: privileged
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: [] ⑤
runAsUser: ⑥
  type: RunAsAny
seLinuxContext: ⑦
  type: RunAsAny
seccompProfiles:
- '*'
supplementalGroups: ⑧
  type: RunAsAny
users: ⑨
- system:serviceaccount:default:registry
- system:serviceaccount:default:router
- system:serviceaccount:openshift-infra:build-controller
volumes:
- '*'

```

- ① Pod で要求できる要求の一覧です。特殊な記号 * は任意の機能を許可しますが、一覧が空の場合は、いずれの機能も要求できないことを意味します。
- ② Pod に追加される追加機能の一覧です。
- ③ セキュリティーコンテキストの許可される値を定める **FSGroup** ストラテジータイプです。
- ④ この SCC へのアクセスを持つグループです。
- ⑤ Pod からドロップされる機能の一覧です。
- ⑥ セキュリティーコンテキストの許可される値を定める run as user ストラテジータイプです。
- ⑦ セキュリティーコンテキストの許可される値を定める SELinux コンテキストストラテジータイプです。
- ⑧ セキュリティーコンテキストの許可される補助グループを定める補助グループストラテジーです。
- ⑨ この SCC へのアクセスを持つユーザーです。

SCC の **users** および **groups** フィールドは使用できる SCC を制御します。デフォルトで、クラスター管理者、ノードおよびビルドコントローラーには特権付き SCC へのアクセスが付与されます。認証されるすべてのユーザーには制限付き SCC へのアクセスが付与されます。

Docker には、Pod の各コンテナについて許可される[デフォルトの機能一覧](#)があります。コンテナはこれらの機能をデフォルト一覧から使用しますが、Pod マニフェストの作成者は追加機能を要求したり、デフォルトから一部をドロップしてこの一覧を変更できます。**allowedCapabilities**、**defaultAddCapabilities**、および**requiredDropCapabilities** フィールドは、Pod からのこのような要求を制御し、要求できる機能を決定し、各コンテナに追加するものや禁止する必要のあるものを決定するために使用されます。

特権付き SCC:

- 特権付き Pod を許可します。
- ホストディレクトリーのボリュームとしてのマウントを許可します。
- Pod の任意ユーザーとしての実行を許可します。
- Pod の MCS ラベルを使った実行を許可します。
- Pod がホストの IPC namespace を使用することを許可します。
- Pod がホストの PID namespace を使用することを許可します。
- Pod が FSGroup を使用することを許可します。
- Pod が補助グループを使用することを許可します。
- Pod が seccomp プロファイルを使用することを許可します。
- Pod が任意の機能を要求することを許可します。

制限付き SCC:

- Pod が特権付きとして実行できないようにします。
- Pod がホストディレクトリーボリュームを使用できないようにします。
- Pod が事前に割り当てられた UID の範囲のユーザーとして実行されることを要求します。
- Pod が事前に割り当てられた MCS ラベルで実行されることを要求します。
- Pod が FSGroup を使用することを許可します。
- Pod が補助グループを使用することを許可します。



注記

各 SCC の詳細は、SCC で利用可能な kubernetes.io/description アノテーションを参照してください。

SCC は Pod がアクセスできるセキュリティー機能を制限する各種の設定およびストラテジーで構成されています。これらの設定は以下のカテゴリーに分類されます。

Controlled by a boolean (ブール値による制御)	このタイプのフィールドはデフォルトで最も制限のある値に設定されます。たとえば、 AllowPrivilegedContainer は指定されていない場合は、常に false に設定されます。
Controlled by an allowable set (許可されるセットによる制御)	このタイプのフィールドは該当セットに対してチェックされ、それらの値が許可されることを確認します。

Controlled by a strategy (ストラテジーによる制御)	<p>値を生成するストラテジーを持つ項目は以下を提供します。</p> <ul style="list-style-type: none"> • 値を生成するメカニズム • 指定された値が許可される値のセット内の値であることを確認するメカニズム
---	---

4.2.5.1. SCC ストラテジー

4.2.5.1.1. RunAsUser

1. **MustRunAs - runAsUser** が設定されることを要求します。デフォルトで設定済みの **runAsUser** を使用します。設定済みの **runAsUser** に対して検証します。
2. **MustRunAsRange** - 事前に割り当てられた値を使用していない場合に、最小および最大値が定義されることを要求します。デフォルトでは最小値を使用します。許可される範囲全体に対して検証します。
3. **MustRunAsNonRoot** - Pod がゼロ以外の **runAsUser** で送信されること、または **USER** ディレクティブをイメージに定義することを要求します。デフォルトは指定されません。
4. **RunAsAny** - デフォルトは指定されません。**runAsUser** の指定を許可します。

4.2.5.1.2. SELinuxContext

1. **MustRunAs** - 事前に割り当てられた値を使用していない場合に **seLinuxOptions** が設定されることを要求します。デフォルトとして **seLinuxOptions** を使用します。**seLinuxOptions** に対して検証します。
2. **RunAsAny** - デフォルトは指定されません。**seLinuxOptions** の指定を許可します。

4.2.5.1.3. SupplementalGroups

1. **MustRunAs** - 事前に割り当てられた値を使用していない場合に、少なくとも 1 つの範囲が指定されることを要求します。デフォルトとして最初の範囲の最小値を使用します。すべての範囲に対して検証します。
2. **RunAsAny** - デフォルトは指定されません。**supplementalGroups** の指定を許可します。

4.2.5.1.4. FSGroup

1. **MustRunAs** - 事前に割り当てられた値を使用していない場合に、少なくとも 1 つの範囲が指定されることを要求します。デフォルトとして最初の範囲の最小値を使用します。最初の範囲の最初の ID に対して検証します。
2. **RunAsAny** - デフォルトは指定されません。**fsGroup** ID の指定を許可します。

4.2.5.2. ボリュームの制御

特定のボリュームタイプの使用は、SCC の **volumes** フィールドを設定して制御できます。このフィールドの許可される値は、ボリュームの作成時に定義されるボリュームソースに対応します。

- [azureFile](#)

- [azureDisk](#)
- [flocker](#)
- [flexVolume](#)
- [hostPath](#)
- [emptyDir](#)
- [gcePersistentDisk](#)
- [awsElasticBlockStore](#)
- [gitRepo](#)
- [secret](#)
- [nfs](#)
- [iscsi](#)
- [glusterfs](#)
- [persistentVolumeClaim](#)
- [rbd](#)
- [cinder](#)
- [cephFS](#)
- [downwardAPI](#)
- [fc](#)
- [configMap](#)
- [vsphereVolume](#)
- [quobyte](#)
- [photonPersistentDisk](#)
- [projected](#)
- [portworxVolume](#)
- [scaleIO](#)
- [storageos](#)
- * (すべてのボリュームタイプの使用を許可する特殊な値)
- **none** (すべてのボリュームタイプの使用を不許可にする特殊な値。後方互換性が必要な場合にのみ使用されます)

新規 SCC の許可されるボリュームの推奨される最小セットは **configMap**、**downwardAPI**、**emptyDir**、**persistentVolumeClaim**、**secret**、および **projected** です。



注記

許可されるボリュームタイプの一覧は、新規タイプが OpenShift Container Platform の各リリースと共に追加されるため、すべてを網羅した一覧である必要はありません。



注記

後方互換性のために、**allowHostDirVolumePlugin** の内容は **volumes** フィールドの設定を上書きします。たとえば、**allowHostDirVolumePlugin** が **false** に設定されるが **volumes** フィールドで許可される場合、**hostPath** 値は **volumes** から削除されます。

4.2.5.3. FlexVolume へのアクセスの制限

OpenShift Container Platform は、それらのドライバーに基づいて FlexVolume についての制御を追加で提供します。SCC が FlexVolume の使用を許可する場合、Pod は任意の FlexVolume を要求できます。ただし、クラスター管理者が **AllowedFlexVolumes** フィールドでドライバー名を指定する場合、Pod はこれらのドライバーでのみ FlexVolumes を使用する必要があります。

アクセスを 2 つの FlexVolume のみに制限する例

```
volumes:
- flexVolume
allowedFlexVolumes:
- driver: example/lvm
- driver: example/cifs
```

4.2.5.4. Seccomp

SeccompProfiles は、Pod またはコンテナの seccomp アノテーションに設定できる許可されたプロファイルを一覧表示します。未使用 (nil) または空の値は、プロファイルが Pod またはコンテナで指定されないことを意味します。ワイルドカード * を使用してすべてのプロファイルを許可します。Pod の値を生成するために使用される場合、最初のワイルドカード以外のプロファイルがデフォルトとして使用されます。

カスタムプロファイルの設定および使用についての詳細は、[seccomp についての説明](#)を参照してください。

4.2.5.5. 受付 (Admission)

SCC の設定された **Admission Control** (受付制御) により、ユーザーに付与された機能に基づくりソース作成に対する制御が可能になります。

SCC の観点では、これは受付コントローラーが、SCC の適切なセットを取得するためにコンテキストで利用可能なユーザー情報を検査できることを意味します。これにより、Pod にはその運用環境についての要求を行ったり、Pod に適用する一連の制約を生成したりする権限が与えられます

受付が Pod を許可するために使用する SCC のセットはユーザーアイデンティティおよびユーザーが属するグループによって決定されます。さらに、Pod がサービスアカウントを指定する場合、許可される SCC のセットにはサービスアカウントでアクセスできる制約が含まれます。

受付は以下の方法を使用して、Pod の最終的なセキュリティーコンテキストを作成します。

1. 使用できるすべての SCC を取得します。
2. 要求に指定されていないセキュリティーコンテキストの設定のフィールド値を生成します。
3. 利用可能な制約に対する最終的な設定を検証します。

制約の一致するセットが検出される場合、Pod が受け入れられます。要求が SCC に一致しない場合、Pod は拒否されます。

Pod はすべてのフィールドを SCC に対して検証する必要があります。以下は、検証する必要がある 2 つのフィールドのみについての例になります。



注記

これらの例は、事前に割り当てられる値を使用するストラテジーに関連するものです。

MustRunAs の FSGroup SCC ストラテジー

Pod が **fsGroup** ID を定義する場合、その ID はデフォルトの **fsGroup** ID に等しくなければなりません。そうでない場合、Pod はその SCC によって検証されず、次の SCC が評価されます。

SecurityContextConstraints.fsGroup フィールドに値 **RunAsAny** があり、Pod 仕様が **Pod.spec.securityContext.fsGroup** を省略する場合、このフィールドは有効とみなされます。検証時に、他の SCC 設定が他の Pod フィールドを拒否し、そのため Pod を失敗させる可能性があることに注意してください。

MustRunAs の SupplementalGroups SCC ストラテジー

Pod 仕様が 1 つ以上の **supplementalGroups** ID を定義する場合、Pod の ID は namespace の **openshift.io/sa.scc.supplemental-groups** アノテーションの ID のいずれかに等しくなければなりません。そうでない場合は、Pod は SCC で検証されず、次の SCC が評価されます。

SecurityContextConstraints.supplementalGroups フィールドに値 **RunAsAny** があり、Pod 仕様が **Pod.spec.securityContext.supplementalGroups** を省略する場合、このフィールドは有効とみなされます。検証時に、他の SCC 設定が他の Pod フィールドを拒否し、そのため Pod を失敗させる可能性があることに注意してください。

4.2.5.5.1. SCC の優先度設定

SCC には、受付コントローラーによる要求の検証を試行する際の順序に影響を与える優先度フィールドがあります。優先度の高い SCC は並び替える際にセットの先頭に移動します。利用可能な SCC の完全なセットが決定されると、それらは以下に戻づいて順序付けられます。

1. 優先度が高い順。nil は 0 優先度とみなされます。
2. 優先度が等しい場合、SCC は最も制限の多いものから少ないものの順に並べ替えられます。
3. 優先度と制限のどちらも等しい場合、SCC は名前順に並べ替えられます。

デフォルトで、クラスター管理者に付与される **anyuid** SCC には SCC セットの優先度が指定されます。これにより、クラスター管理者は Pod の **SecurityContext** で **RunAsUser** を指定しなくても Pod を任意のユーザーとして実行できます。管理者は、希望する場合は依然として **RunAsUser** を指定できます。

4.2.5.5.2. 事前に割り当てられた値および SCC (Security Context Constraints) について

受付コントローラーは、これが namespace の事前に設定された値を検索し、Pod の処理前に SCC (Security Context Constraints) を設定するようにトリガーする SCC (Security Context Constraint) の特定の条件を認識します。各 SCC ストラテジーは他のストラテジーとは別個に評価されます。この際、(許可される場合に) Pod 仕様の値と共に集計された各ポリシーの事前に割り当てられた値が使用され、実行中の Pod で定義される各種 ID の最終の値が設定されます。

以下の SCC により、受付コントローラーは、範囲が Pod 仕様で定義されていない場合に事前に定義された値を検索できます。

1. 最小または最大値が設定されていない **MustRunAsRange** の **RunAsUser** ストラテジー。受付は **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を設定します。
2. レベルが設定されていない **MustRunAs** の **SELinuxContext** ストラテジー。受付は **openshift.io/sa.scc.mcs** アノテーションを検索してレベルを設定します。
3. **MustRunAs** の **FSGroup** ストラテジー。受付は **openshift.io/sa.scc.supplemental-groups** アノテーションを検索します。
4. **MustRunAs** の **SupplementalGroups** ストラテジー。受付は **openshift.io/sa.scc.supplemental-groups** アノテーションを検索します。

生成フェーズでは、セキュリティーコンテキストのプロバイダーが Pod に設定されていない値をデフォルト設定します。デフォルト設定は使用されるストラテジーに基づいて行われます。

1. **RunAsAny** および **MustRunAsNonRoot** ストラテジーはデフォルトの値を提供しません。そのため、Pod が定義されるフィールドを必要とする場合 (グループ ID など)、このフィールドは Pod 仕様内に定義する必要があります。
2. **MustRunAs** (単一の値) ストラテジーは、常に使用されるデフォルト値を提供します。たとえば、グループ ID の場合、Pod 仕様が独自の ID 値を定義する場合でも、namespace のデフォルトフィールドが Pod のグループに表示されます。
3. **MustRunAsRange** および **MustRunAs** (範囲ベース) ストラテジーは、範囲の最小値を提供します。単一の値の **MustRunAs** ストラテジーの場合のように、namespace のデフォルト値は実行中の Pod に表示されます。範囲ベースのストラテジーが複数の範囲で設定可能な場合、これは最初に設定された範囲の最小値を指定します。

注記

FSGroup および SupplementalGroups ストラテジー

は、**openshift.io/sa.scc.supplemental-groups** アノテーションが namespace に存在しない場合に **openshift.io/sa.scc.uid-range** アノテーションにフォールバックします。いずれも存在しない場合、SCC は作成に失敗します。

注記

デフォルトで、アノテーションベースの **FSGroup** ストラテジーはアノテーションの最小値に基づく単一の範囲で設定されます。たとえば、アノテーションが **1/3** を読み取る場合、**FSGroup** ストラテジーは **1** の最小値および最大値で自らを設定します。追加のグループを **FSGroup** フィールドで許可する必要がある場合、アノテーションを使用しないカスタム SCC を設定することができます。



注記

openshift.io/sa.scc.supplemental-groups アノテーション

は、`<start>/<length>` または `<start>--<end>` 形式のカンマ区切りのブロックの一覧を受け入れます。**openshift.io/sa.scc.uid-range** アノテーションは単一ブロックのみを受け入れます。

4.2.6. 認証済みユーザーが実行できる内容の判別

OpenShift Container Platform プロジェクト内で、namespace でスコープ設定されたすべてのリソース (サードパーティのリソースを含む) に対して実行できる動詞を判別することができます。以下を実行します。

```
$ oc policy can-i --list --loglevel=8
```

この出力は、情報収集のために実行する必要がある API 要求を判断するのに役立ちます。

ユーザーの判読可能な形式で情報を受信するには、以下を実行します。

```
$ oc policy can-i --list
```

この出力により、詳細な一覧が表示されます。

特定の動詞が実行可能かどうかを判断するには、以下を実行します。

```
$ oc policy can-i <verb> <resource>
```

ユーザースコープ は、指定のスコープに関する詳細情報を提供します。以下に例を示します。

```
$ oc policy can-i <verb> <resource> --scopes=user:info
```

4.3. 永続ストレージ

4.3.1. 概要

ストレージの管理は、コンピュータリソースの管理とは異なります。OpenShift Container Platform は Kubernetes 永続ボリューム (PV) フレームワークを使用してクラスター管理者がクラスターの永続ストレージのプロビジョニングを実行できるようにします。Persistent Volume Claims (PVC、永続ボリューム要求) を使用することで、開発者は基礎となるストレージインフラストラクチャーについての特定の知識がなくても PV リソースを要求できます。

PVC はプロジェクトに固有のものであり、開発者によって PV を使用する手段として作成され、使用されます。それら自体の PV リソースはいずれの単一プロジェクトにもスコープ設定されず、それらは OpenShift Container Platform クラスターで共有でき、任意のプロジェクトから要求できます。PV が PVC に **バインド** された後は、その PV は追加の PVC にバインドできなくなります。これにはバインドされた PV を単一の namespace (バインディングプロジェクトの namespace) にスコープ設定する機能があります。

PV は **PersistentVolume** API オブジェクトで定義されます。このオブジェクトは、クラスター管理者によってプロビジョニングされた、クラスター内のネットワーク設定された既存ストレージの一部を表します。これは、ノードがクラスターリソースであるのと同様にクラスター内のリソースになります。

す。PV は **Volumes** のようなボリュームプラグインですが、PV を使用する個々の **Pod** とは別のライフサイクルを持ちます。PV オブジェクトは、NFS、iSCSI、またはクラウドプロバイダー固有のストレージシステムのいずれの場合についてもストレージの実装の詳細をキャプチャーします。



重要

インフラストラクチャーにおけるストレージの高可用性については、基礎となるストレージのプロバイダーによって異なります。

PVC は、開発者によるストレージの要求を表す **PersistentVolumeClaim** API オブジェクトによって定義されます。これは Pod がノードリソースを消費する点で Pod に似ており、PVC は PV リソースを消費します。たとえば、Pod は特定のレベルのリソース (CPU およびメモリーなど) を要求し、PVC は特定の**ストレージ容量**および**アクセスモード**を要求できます (たとえば、それらは読み取り/書き込みで 1 回、読み取り専用で複数回マウントできます)。

4.3.2. ボリュームおよび要求のライフサイクル

PV はクラスターのリソースです。PVC はそれらのリソースの要求であり、リソースに対する要求チェックとして機能します。PV と PVC 間の相互作用には以下のライフサイクルが設定されます。

4.3.2.1. プロビジョニング

PVC で定義される開発者からの要求に対応し、クラスター管理者はストレージおよび一致する PV をプロビジョニングする 1 つ以上の動的プロビジョナーを設定します。

またクラスター管理者は、使用可能な実際のストレージの詳細を保持する多数の PV を前もって作成できます。PV は API に存在し、利用可能な状態になります。

4.3.2.2. バインディング

PVC の作成時に、ストレージの特定の容量を要求し、必要なアクセスモードを指定し、ストレージを記述し、分類するためのストレージクラスを作成できます。マスターのコントロールループは新規 PVC の有無を監視し、新規 PVC を適切な PV にバインドします。適切な PV がない場合、ストレージクラスのプロビジョナーが PV を作成します。

PV ボリュームが要求したボリュームを上回る可能性があります。これは、手動でプロビジョニングされた PV の場合にとくにそう言えます。超過を最小限にするために、OpenShift Container Platform は他のすべての条件に一致する最小の PV にバインドされます。

一致するボリュームが存在しないか、ストレージクラスを提供するいずれの利用可能なプロビジョナーで作成されない場合に、要求は無期限にバインドされないままになります。要求は、一致するボリュームが利用可能になるとバインドされます。たとえば、多数の手動でプロビジョニングされた 50Gi ボリュームを持つクラスターは 100Gi を要求する PVC に一致しません。PVC は 100Gi PV がクラスターに追加されるとバインドされます。

4.3.2.3. 使用

Pod は要求をボリュームとして使用します。クラスターは要求を検査して、バインドされたボリュームを検索し、Pod にそのボリュームをマウントします。複数のアクセスモードをサポートするボリュームの場合、要求を Pod のボリュームとして使用する際に適用するモードを指定する必要があります。

ユーザーがバインドされている要求を持つ場合、バインドされた PV は必要な限りユーザーに所属することになります。Pod のスケジュールおよび要求された PV のアクセスは、**persistentVolumeClaim** を Pod のボリュームブロックに組み込んで実行できます。構文の詳細については、[以下](#)を参照してください。

4.3.2.4. Persistent Volume Claim (永続ボリューム要求) の保護



注記

PVC の保護はアルファ機能であり、OpenShift Container Platform の今後のリリースで変更される場合があります。

PVC 保護の目的は、Pod でアクティブに使用される PVC がシステムから削除されることによってデータ損失が生じないようにすることにあります。



注記

PVC は、Pod ステータスが **Pending** で Pod がノードに割り当てられている場合や、Pod ステータスが **Running** の場合に Pod によってアクティブに使用された状態になります。

PVC の保護機能が有効にされている場合、Pod でアクティブに使用されている PVC を削除しても PVC はすぐに除去されません。PVC の除去は PVC が Pod でアクティブに使用されなくなるまで延期されます。

PVC のステータスが **Terminating** の場合、PVC が保護され、**Finalizers** 一覧に **kubernetes.io/pvc-protection** が含まれていることを確認できます。

```
oc describe pvc hostpath
Name:          hostpath
Namespace:     default
StorageClass:  example-hostpath
Status:        Terminating
Volume:
Labels:        <none>
Annotations:   volume.beta.kubernetes.io/storage-class=example-hostpath
               volume.beta.kubernetes.io/storage-
provisioner=example.com/hostpath
Finalizers:    [kubernetes.io/pvc-protection]
...
```

PVC の保護を有効にするには、「[Persistent Volume Claim \(永続ボリューム要求\) 保護の設定](#)」を参照してください。

4.3.2.5. リリース

ボリュームの処理が終了したら、API から PVC オブジェクトを削除できます。これにより、リソースの回収が可能になります。ボリュームは要求の削除時に「リリース」されたものとみなされますが、別の要求で利用できる状態にはありません。以前の要求者のデータはボリューム上に残り、ポリシーに基づいて処理される必要があります。

4.3.2.6. 回収

PersistentVolume の回収ポリシーは、クラスターに対してリリース後のボリュームの処理方法について指示します。ボリュームの回収ポリシーは、**Retained**、**Recycled**、または **Deleted** のいずれかにすることができます。

Retained 回収ポリシーにより、これをサポートするボリュームプラグインについてのリソースを手動で回収することを可能にします。**Deleted** 回収ポリシーは、OpenShift Container Platform の

PersistentVolume オブジェクトと、および AWS EBS、GCE PD、または Cinder ボリュームなどの外部インフラストラクチャーの関連ストレージセットの両方を削除します。



注記

動的にプロビジョニングされているボリュームは常に削除されます。

4.3.2.6.1. リサイクル

適切なボリュームプラグインでサポートされる場合、リサイクルはボリュームの基本的なスクラブを実行 (`rm -rf /thevolume/*`) し、これを新規の要求で再度利用可能にします。



警告

recycle 回収ポリシーは動的プロビジョニングが優先されるために非推奨となり、今後のリリースでは削除されます。

「[ControllerArguments](#)」のセクションで説明されているように、コントローラーマネージャーのコマンドライン引数を使用してカスタム Recycler Pod テンプレートを設定できます。カスタム Recycler Pod テンプレートには、以下の例に示されているように **volumes** 仕様が含まれます。

```
apiVersion: v1
kind: Pod
metadata:
  name: pv-recycler-
  namespace: openshift-infra ❶
spec:
  restartPolicy: Never
  serviceAccount: pv-recycler-controller ❷
  volumes:
  - name: nfsvol
    nfs:
      server: any-server-it-will-be-replaced ❸
      path: any-path-it-will-be-replaced ❹
  containers:
  - name: pv-recycler
    image: "gcr.io/google_containers/busybox"
    command: ["/bin/sh", "-c", "test -e /scrub && rm -rf /scrub/.*?*/scrub/.[!]* /scrub/* && test -z \"$(ls -A /scrub)\" || exit 1"]
    volumeMounts:
    - name: nfsvol
      mountPath: /scrub
```

❶ リサイクラー Pod が実行される namespace です。**openshift-infra** には、ボリュームをリサイクルできる **pv-recycler-controller** サービスアカウントがすでに含まれているため、これは推奨される namespace になります。

❷ NFS ボリュームのマウントが許可されるサービスアカウントの名前です。これは指定された namespace に存在する必要があります。**pv-recycler-controller** アカウントは、**openshift-infra** namespace に自動作成され、必要なすべてのパーミッションを含むた

め、推奨されるアカウントになります。

3 **4** カスタムリサイクラー Pod テンプレートの **volumes** 部分に指定される特定の **server** および **path** 値は、リサイクルされる PV の特定の対応する値に置き換えられます。

4.3.3. 永続ボリューム

各 PV には、ボリュームの仕様およびステータスである **spec** および **status** が含まれます。

永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /tmp
    server: 172.17.0.2
```

4.3.3.1. 永続ボリュームのタイプ

OpenShift Container Platform は以下の **PersistentVolume** プラグインをサポートします。

- [NFS](#)
- [HostPath](#)
- [GlusterFS](#)
- [Ceph RBD](#)
- [OpenStack Cinder](#)
- [AWS Elastic Block Store \(EBS\)](#)
- [GCE Persistent Disk](#)
- [iSCSI](#)
- [ファイバーチャネル](#)
- [Azure Disk](#)
- [Azure File](#)
- [VMWare vSphere](#)
- [ローカル](#)

4.3.3.2. 容量

通常 PV は特定のストレージ容量を持ちます。これは PV の **capacity** 属性を使用して設定されます。

現時点でストレージ容量は、設定または要求できる唯一のリソースです。今後の属性には IOPS、スループットなどが含まれる可能性があります。

4.3.3.3. アクセスモード

PersistentVolume はリソースプロバイダーでサポートされるすべての方法でホストにマウントできます。プロバイダーには各種の機能があり、それぞれの PV のアクセスモードは特定のボリュームでサポートされる特定のモードに設定されます。たとえば、NFS は複数の読み取り/書き込みクライアントをサポートしますが、特定の NFS PV は読み取り専用としてサーバー上でエクスポートされる可能性があります。それぞれの PV は、その特定の PV の機能について記述するアクセスモードの独自のセットを取得します。

要求は、同様のアクセスモードのボリュームに一致します。一致する条件はアクセスモードとサイズの2つの条件のみです。要求のアクセスモードは要求 (request) を表します。そのため、より多くのアクセスを付与することはできますが、アクセスを少なくすることはできません。たとえば、RWO が要求されるものの、利用できる唯一のボリュームが NFS PV (RWO+ROX+RWX) の場合、要求は RWO をサポートするために NFS に一致します。

直接的なマッチングが常に最初に試行されます。ボリュームのモードは要求するモードに一致するか、またはモードには一致する以上のモードが含まれる必要があります。サイズは予想されるものより多いか、またはこれと同等である必要があります。2つのタイプのボリューム (NFS および iSCSI など) のどちらにも同じセットのアクセスモードがある場合、それらのいずれかがそれらのモードを持つ要求に一致する可能性があります。ボリュームのタイプ間で順序付けすることはできず、タイプを選択することはできません。

同じモードのすべてのボリュームは分類され、サイズ別 (小さいものから大きいもの順) に分類されます。バインダーは一致するモードのグループを取得し、1つのサイズが一致するまで (サイズの順序で) 処理を繰り返します。

アクセスモードは以下ようになります。

アクセスモード	CLI の省略形	説明
ReadWriteOnce	RWO	ボリュームは単一ノードで読み取り/書き込みとしてマウントできます。
ReadOnlyMany	ROX	ボリュームは数多くのノードで読み取り専用としてマウントできます。
ReadWriteMany	RWX	ボリュームは数多くのノードで読み取り/書き込みとしてマウントできます。

重要

ボリュームの **AccessModes** はボリュームの機能の記述子です。それらは施行されている制約ではありません。ストレージプロバイダーはリソースの無効な使用から生じるランタイムエラーに対応します。

たとえば、Ceph は **ReadWriteOnce** アクセスモードを提供します。ボリュームの ROX 機能を使用する必要がある場合は、要求に **read-only** のマークを付ける必要があります。プロバイダーのエラーは、マウントエラーとしてランタイム時に表示されます。

iSCSI およびファイバーチャネルボリュームにはフェンシングメカニズムがありません。ボリュームが一度に 1 つのノードでのみ使用されるようにする必要があります。ノードのドレイン (解放) などの特定の状況では、ボリュームは 2 つのノードで同時に使用される可能性があります。ノードをドレイン (解放) する前に、これらのボリュームを使用する Pod が削除されていることを確認してください。

以下の表では、異なる永続ボリュームによってサポートされるアクセスモードを一覧表示しています。

表4.1 永続ボリュームのサポートされるアクセスモード

ボリュームプラグイン	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWS EBS	□	-	-
Azure File	□	□	□
Azure Disk	□	-	-
Ceph RBD	□	□	-
ファイバーチャネル	□	□	-
GCE Persistent Disk	□	-	-
GlusterFS	□	□	□
HostPath	□	-	-
iSCSI	□	□	-
NFS	□	□	□

ボリュームプラグイン	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
Openstack Cinder	□	-	-
VMWare vSphere	□	-	-
ローカル	□	-	-



注記

- Pod が AWS EBS、GCE Persistent Disk、または Openstack Cinder PV を使用する場合は、[再作成デプロイメントストラテジー](#)を使用します。

4.3.3.4. 回収ポリシー

現時点で、回収ポリシーは以下のようになります。

回収ポリシー	説明
Retain (保持)	手動による回収
Recycle (リサイクル)	基本的なスクラブ (例: <code>rm -rf /<volume>/*</code>)



注記

現時点では、NFS および HostPath のみが「リサイクル」回収ポリシーをサポートしています。



警告

recycle 回収ポリシーは動的プロビジョニングが優先されるために非推奨となり、今後のリリースでは削除されます。

4.3.3.5. フェーズ

ボリュームは以下のフェーズのいずれかに置かれます。

フェーズ	説明
Available	要求にバインドされていない空きリソースです。

フェーズ	説明
Bound	ボリュームが要求にバインドされています。
Released	要求が検出されていますが、リソースはクラスターによって回収されていません。
Failed	ボリュームがその自動回収に失敗しています。

CLI には PV にバインドされている PVC の名前が表示されます。

4.3.3.6. マウントオプション

アノテーション **volume.beta.kubernetes.io/mount-options** を使用して永続ボリュームのマウント中にマウントオプションを指定できます。

以下は例になります。

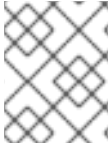
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
  annotations:
    volume.beta.kubernetes.io/mount-options: rw,nfsvers=4,noexec ❶
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Recycle
  claimRef:
    name: claim1
    namespace: default
```

❶ 永続ボリュームのディスクへのマウント中に指定されたマウントオプションが使用されます。

以下の永続ボリュームタイプがマウントオプションをサポートします。

- NFS
- GlusterFS
- Ceph RBD
- OpenStack Cinder
- AWS Elastic Block Store (EBS)
- GCE Persistent Disk

- iSCSI
- Azure Disk
- Azure File
- VMWare vSphere



注記

ファイバーチャネルおよび HostPath 永続ボリュームはマウントオプションをサポートしません。

4.3.4. Persistent Volume Claim (永続ボリューム要求、PVC)

各 PVC には、要求の仕様およびステータスである **spec** および **status** が含まれます。

Persistent Volume Claim (永続ボリューム要求、PVC) オブジェクト定義

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: gold
```

4.3.4.1. ストレージクラス

要求により、ストレージクラスの名前を **storageClassName** 属性に指定し、特定のストレージクラスをオプションで要求できます。要求されたクラスの PV、つまり PVC と同じ **storageClassName** を持つ PV のみが PVC にバインドされます。クラスター管理者は 1 つ以上のストレージクラスを提供するように動的プロビジョナーを設定できます。クラスター管理者は、PVC の仕様に一致する PV をオンデマンドで作成できます。

クラスター管理者は、すべての PVC にデフォルトストレージクラスを設定することもできます。デフォルトのストレージクラスが設定されると、PVC では **StorageClass** または **storageClassName** アノテーションが "" に設定され、ストレージクラスなしで PV にバインドされるように明示的に要求する必要があります。

4.3.4.2. アクセスモード

要求では、特定のアクセスモードのストレージを要求する際にボリュームと同じ規則を使用します。

4.3.4.3. リソース

要求では、Pod の場合のようにリソースの特定の数量を要求できます。この場合の要求はストレージについてのものです。同じリソースモデルがボリュームと要求の両方に適用されます。

4.3.4.4. ボリュームとしての要求

Pod は要求をボリュームとして使用してストレージにアクセスします。要求は要求を使用する Pod と同じ namespace に存在する必要があります。クラスターは Pod の namespace で要求を見つけ、要求をサポートする **PersistentVolume** を取得するためにこれを使用します。次に、ボリュームはホストにマウントされ、Pod に組み込まれます。

```
kind: Pod
apiVersion: v1
metadata:
  name: myPod
spec:
  containers:
  - name: myfrontend
    image: dockerfile/nginx
    volumeMounts:
    - mountPath: "/var/www/html"
      name: mypd
  volumes:
  - name: mypd
    persistentVolumeClaim:
      claimName: myclaim
```

4.3.5. ブロックボリュームのサポート

重要

ブロックボリュームのサポートはテクノロジープレビュー機能であり、手動でプロビジョニングされた永続ボリュームでのみ利用可能です。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。

一部の新規 API フィールドを PV および PVC 仕様に組み込み、raw ブロックボリュームを静的にプロビジョニングできます。

ブロックボリュームを使用するには、まず **BlockVolume** 機能ゲートを有効にする必要があります。マスターの機能ゲートを有効にするには、**feature-gates** を **apiServerArguments** および **controllerArguments** に追加します。ノードの機能ゲートを有効にするには、**feature-gates** を **kubeletArguments** に追加します。以下は例になります。

```
kubeletArguments:
  feature-gates:
  - BlockVolume=true
```

詳細情報は、「[ローカルボリュームの設定](#)」を参照してください。

永続ボリュームのサンプル

```
apiVersion: v1
kind: PersistentVolume
```

```

metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block ❶
  persistentVolumeReclaimPolicy: Retain
  fc:
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false

```

❶ **volumeMode** フィールドは、この PV が raw ブロックボリュームであることを示します。

Persistent Volume Claim (永続ボリューム要求) のサンプル

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block ❶
  resources:
    requests:
      storage: 10Gi

```

❶ **volumeMode** フィールドは、raw ブロック永続ボリュームが要求されていることを示します。

Pod 仕様のサンプル

```

apiVersion: v1
kind: Pod
metadata:
  name: Pod-with-block-volume
spec:
  containers:
    - name: fc-container
      image: fedora:26
      command: ["/bin/sh", "-c"]
      args: [ "tail -f /dev/null" ]
      volumeDevices: ❶
        - name: data
          devicePath: /dev/xvda ❷
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: block-pvc ❸

```

- 1 (volumeMounts と同様に) **volumeDevices** は、ブロックデバイスに使用され、**PersistentVolumeClaim** ソースでのみ使用できます。
- 2 (mountPath と同様に) **devicePath** は、物理デバイスへのパスを表します。
- 3 ボリュームソースのタイプは **persistentVolumeClaim** であり、予想される通りに PVC の名前に一致する必要があります。

表4.2 VolumeMode の許可される値

値	デフォルト
Filesystem	Yes
Block	No

表4.3 ブロックボリュームのバインドシナリオ

PV VolumeMode	PVC VolumeMode	バインディングの結果
Filesystem	Filesystem	バインド
Unspecified	Unspecified	バインド
Filesystem	Unspecified	バインド
Unspecified	Filesystem	バインド
Block	Block	バインド
Unspecified	Block	バインドなし
Block	Unspecified	バインドなし
Filesystem	Block	バインドなし
Block	Filesystem	バインドなし

**重要**

値を指定しないと、**Filesystem** のデフォルト値が指定されます。

表4.4 ブロックボリュームをサポートし、今後サポートするプラグインのステータス

プラグイン	ブロックボリュームのサポート
ファイバーチャネル	(マージ済み) OpenShift Container Platform 3.9
Ceph RBD	(マージ済み) OpenShift Container Platform 3.10
iSCSI	(進行中) OpenShift Container Platform 3.10
AWS EBS	(進行中) OpenShift Container Platform 3.10
GCE PD	(進行中) OpenShift Container Platform 3.10
GlusterFS	(進行中) OpenShift Container Platform 3.10

4.4. ソースコントロール管理

OpenShift Container Platform は、内部 (インハウス Git サーバーなど) または外部 ([GitHub](#)、[Bitbucket](#) など) でホストされている既存のソースコントロール管理 (SCM) システムを利用します。現時点で、OpenShift Container Platform は [Git](#) ソリューションのみをサポートします。

SCM 統合は[ビルド](#)に密接に関連し、以下の 2 つの点を実行します。

- リポジトリを使用して **BuildConfig** を作成します。これにより OpenShift Container Platform 内でのアプリケーションのビルドが可能になります。**BuildConfig** を[手動で作成](#)することも、リポジトリを検査して OpenShift Container Platform で[自動的に作成](#)することもできます。
- リポジトリの変更時の[ビルドをトリガー](#)します。

4.5. 受付コントローラー

4.5.1. 概要

受付制御プラグインはリソースの永続化の前にマスター API への要求をインターセプトしますが、要求の認証および承認後にこれを実行します。

それぞれの受付制御プラグインは、要求がクラスターに受け入れられる前にシーケンスで実行されます。シーケンスのプラグインが要求を拒否する場合、要求全体がただちに拒否され、エラーがエンドユーザーに返されます。

受付制御プラグインは、システムで設定されたデフォルトを適用するために受信オブジェクトを変更する場合があります。さらに、受付制御プラグインはクォータの使用の拡張などを実行する要求処理の一環として関連リソースを変更する場合があります。



警告

OpenShift Container Platform マスターには、各タイプのリソース (Kubernetes および OpenShift Container Platform) についてデフォルトで有効にされているプラグインのデフォルトの一覧が含まれます。それらはマスターが適切に機能するために必要です。これらの一覧を変更することは、実際の変更内容を把握している場合でない限りは推奨されません。本製品の今後のバージョンでは異なるセットのプラグインを使用し、それらの順序を変更する可能性があります。マスター設定ファイルでプラグインのデフォルトの一覧を上書きする場合、新規バージョンの OpenShift Container Platform マスターの要件を反映できるように一覧を更新する必要があります。

4.5.2. 一般的な受付ルール

3.3 以降で、OpenShift Container Platform は Kubernetes および OpenShift Container Platform リソースの単一の受付チェーンを使用します。これは、別個の受付チェーンが使用されていた 3.2 以前とは異なります。これは、トップレベルの **admissionConfig.pluginConfig** 要素に **kubernetesMasterConfig.admissionConfig.pluginConfig** に含まれていた受付プラグイン設定が含まれることを意味しています。

kubernetesMasterConfig.admissionConfig.pluginConfig は **admissionConfig.pluginConfig** に移動し、マージされる必要があります。

また 3.3 より、サポートされるすべての受付プラグインは単一チェーン内で順序付けられます。**admissionConfig.pluginOrderOverride** または **kubernetesMasterConfig.admissionConfig.pluginOrderOverride** を設定する必要はなくなります。代わりに、プラグイン固有の設定を追加するか、または以下のような **DefaultAdmissionConfig** スタンザを追加してデフォルトでオフになっているプラグインを有効にする必要があります。

```
admissionConfig:
  pluginConfig:
    AlwaysPullImages: ❶
    configuration:
      kind: DefaultAdmissionConfig
      apiVersion: v1
      disable: false ❷
```

- ❶ 受付プラグイン名です。
- ❷ プラグインが有効にされる必要があることを示しています。これはオプションであり、参照用にのみ表示されます。

disable を **true** にすると、on にデフォルト設定される受付プラグインが無効になります。



警告

受付プラグインは、API サーバーのセキュリティを実施するために一般的に使用されています。これらを無効にする場合には注意して行ってください。



注記

単一の受付チェーンに安全に組み込むことのできない **admissionConfig** 要素を使用していた場合は、API サーバーログで警告を受信し、API サーバーはレガシーの互換性のために 2 つの異なる受付チェーンで開始されることになります。警告を解決するには、**admissionConfig** を更新します。

4.5.3. カスタマイズ可能な受付プラグイン

クラスター管理者は、一部の受付コントロールプラグインを、以下のような特定の動作を制御するように設定できます。

- [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](#)
- [グローバルビルドのデフォルトと上書きの設定](#)
- [Pod 配置の制御](#)
- [ロールバインディングの管理](#)

4.5.4. コンテナを使用した受付コントローラー

コンテナを使用する受付コントローラーも [init コンテナ](#)をサポートします。

4.6. カスタム受付コントローラー

4.6.1. 概要

デフォルトの[受付コントローラー](#)のほかに、**受付 Webhook** を受付チェーンの一部として使用できます。

受付 Webhook は Webhook サーバーを呼び出して、ラベルの挿入など、作成時に Pod を変更するか、または受付プロセス時に Pod 設定の特定の部分を検証します。

受付 Webhook はリソースの永続化の前にマスター API への要求をインターセプトしますが、要求の認証および承認後にこれを実行します。

4.6.2. 受付 Webhook

OpenShift Container Platform では、API 受付チェーンで Webhook サーバーを呼び出す受付 Webhook オブジェクトを使用できます。

設定可能な 2 種類の受付 Webhook オブジェクトがあります。

- **変更用の受付 Webhook**は、変更用の Webhook を使用した、永続化する前のリソースコンテンツの変更を可能にします。
- **検証用の受付 Webhook** は、検証用の Webhook を使用したカスタム受付ポリシーの実施を可能にします。

Webhook および外部 Webhook サーバーの設定については本書では扱いません。ただし、Webhook サーバーは、OpenShift Container Platform で適切に機能するために、**インターフェース**に準拠する必要があります。



重要

受付 Webhook 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。

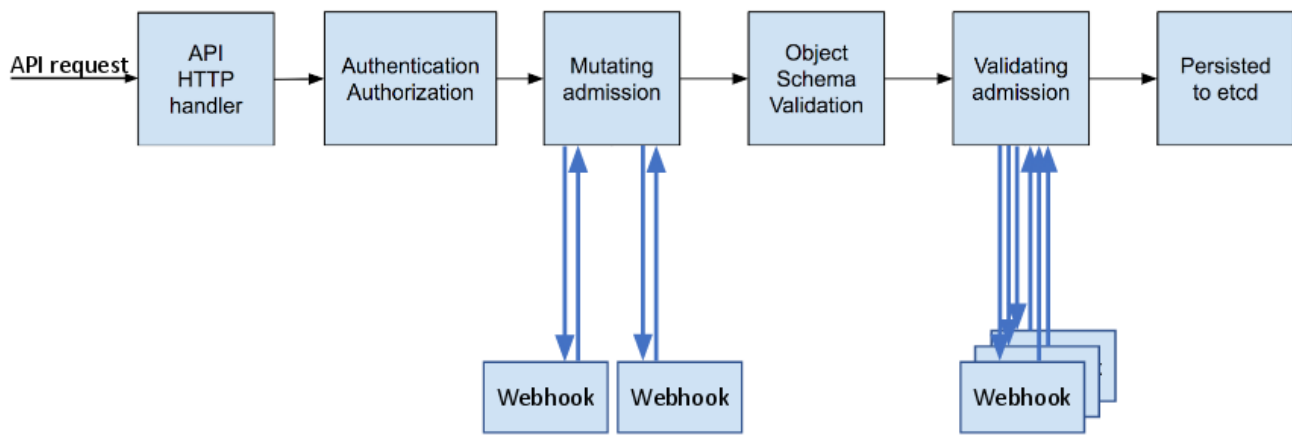
オブジェクトがインスタンス化されると、OpenShift Container Platform は API 呼び出しを実行してオブジェクトを許可します。受付プロセスでは、**変更用の受付コントローラー** は Webhook を呼び出して、アフィニティーラベルの挿入などのタスクを実行します。受付プロセスの終了時に、**検証用の受付コントローラー** は Webhook を呼び出し、アフィニティーラベルの検証などにより、オブジェクトが適切に設定されていることを確認します。検証にパスすると、OpenShift Container Platform はオブジェクトを設定済みとしてスケジュールします。

API 要求が送信されると、変更用または検証用の受付コントローラーは設定内の外部 Webhook の一覧を使用し、それらを並行して呼び出します。

- Webhook の **すべて** が要求を承認する場合、受付チェーンは継続します。
- Webhook の **いずれか** が要求を拒否する場合、受付要求は拒否されます。これは **初回の** webhook の拒否理由に基づいて実行されます。
複数の Webhook が受付要求を拒否する場合、最初のものだけがユーザーに返されます。
- Webhook の呼び出し時にエラーが生じる場合、その要求は無視され、受付要求を承認/拒否するために使用されます。

受付コントローラーと Webhook サーバー間の通信のセキュリティは TLS を使用して保護する必要があります。CA 証明書を生成し、その証明書を使用して Webhook サーバーで使用されるサーバー証明書に署名します。PEM 形式の CA 証明書は、**サービス提供証明書のシークレット**などのメカニズムを使用して受付コントローラーに提供されます。

以下の図は、複数の Webhook を呼び出す 2 つの受付 Webhook を含むプロセスを示しています。



受付 Webhook の単純な使用例として、リソースの構文の検証に使用されるケースがあります。たとえば、使用するインフラストラクチャーではすべての Pod が共通のラベルセットを持つことを条件とする場合、それらのラベルを持たない Pod を永続化しないようにする必要があります。この場合、これらのラベルを挿入する Webhook やそれらのラベルの有無を検証する Webhook を作成することができます。その後 OpenShift Container Platform は、ラベルを持ち、検証をパスした Pod をスケジューリングし、ラベルがないためにパスしない Pod を拒否します。

共通のユースケースには以下が含まれます。

- サイドカーコンテナを Pod に挿入するためのリソースの変更
- 一部のリソースをプロジェクトからブロックするためのプロジェクトの制限
- 依存するフィールドで複雑な検証を実行するためのカスタムリソース検証

4.6.2.1. 受付 Webhook のタイプ

クラスター管理者は、API サーバーの受付チェーンに **変更用の受付 Webhook** または **検証用の受付 Webhook** を含めることができます。

変更用の受付 Webhook は、受付プロセスの変更フェーズで起動します。これにより、リソースコンテンツが永続化される前に変更することができます。受付 Webhook の一例として、[Pod ノードセレクター](#)機能があります。この機能は namespace でアノテーションを使用してラベルセレクターを検索し、これを Pod 仕様に追加します。

受付 Webhook 設定の変更例:

```

apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration ❶
metadata:
  name: <controller_name> ❷
webhooks:
- name: <webhook_name> ❸
  clientConfig: ❹
    service:
      namespace: ❺
      name: ❻
      path: <webhook_url> ❼
    caBundle: <cert> ❽
  
```

```

rules: ⑨
- operations: ⑩
  - <operation>
  apiGroups:
  - ""
  apiVersions:
  - "*"
  resources:
  - <resource>
failurePolicy: <policy> ⑪

```

- ① 変更用の受付 Webhook 設定を指定します。
- ② 受付 Webhook オブジェクトの名前です。
- ③ 呼び出す Webhook の名前です。
- ④ Webhook サーバーに接続し、これを信頼し、データをこれに送信する方法についての情報です。
- ⑤ フロントエンドサービスが作成されるプロジェクトです。
- ⑥ フロントエンドサービスの名前です。
- ⑦ 受付要求に使用される Webhook URL です。
- ⑧ Webhook サーバーで使用されるサーバー証明書に署名する PEM でエンコーディングされた CA 証明書です。
- ⑨ API サーバーがこのコントローラーを使用するタイミングを定義するルールです。
- ⑩ このコントローラーを呼び出すために API サーバーをトリガーする操作です。
 - create
 - update
 - delete
 - connect
- ⑪ Webhook 受付サーバーが利用できない場合にポリシーを実行する方法を指定します。 **Ignore** (allow/fail open) または **Fail** (block/fail closed) になります。

検証用の受付 **Webhook** は受付プロセスの検証フェーズで起動します。このフェーズでは、特定 API リソースの不変条件を実施し、リソースが再び変更されないようにすることができます。Pod ノードセレクターも、すべての **nodeSelector** フィールドがプロジェクトのノードセレクターの制限で制約されていることを確認する検証用の受付の例となります。

検証用の受付 **Webhook** 設定の例:

```

apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration ①
metadata:
  name: <controller_name> ②
webhooks:

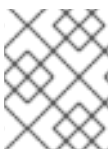
```

```

- name: <webhook_name> ③
  clientConfig: ④
    service:
      namespace: default ⑤
      name: kubernetes ⑥
      path: <webhook_url> ⑦
      caBundle: <cert> ⑧
  rules: ⑨
  - operations: ⑩
    - <operation>
    apiGroups:
      - ""
    apiVersions:
      - "*"
    resources:
      - <resource>
  failurePolicy: <policy> ⑪

```

- ① 検証用の受付 Webhook 設定を指定します。
- ② Webhook 受付オブジェクトの名前です。
- ③ 呼び出す Webhook の名前です。
- ④ Webhook サーバーに接続し、これを信頼し、データをこれに送信する方法についての情報です。
- ⑤ フロントエンドサービスが作成されるプロジェクトです。
- ⑥ フロントエンドサービスの名前です。
- ⑦ 受付要求に使用される Webhook URL です。
- ⑧ Webhook サーバーで使用されるサーバー証明書に署名する PEM でエンコーディングされた CA 証明書です。
- ⑨ API サーバーがこのコントローラーを使用するタイミングを定義するルールです。
- ⑩ このコントローラーを呼び出すために API サーバーをトリガーする操作です。
 - create
 - update
 - delete
 - connect
- ⑪ Webhook 受付サーバーが利用できない場合にポリシーを実行する方法を指定します。 **Ignore** (allow/fail open) または **Fail** (block/fail closed) になります。



注記

Fail open の場合に、すべてのクライアントの予測できない動作が生じる可能性があります。

4.6.2.2. 受付 Webhook の作成

最初に外部 Webhook サーバーをデプロイし、これが適切に機能することを確認します。これを実行しない場合、Webhook が **fail open** または **fail closed** として設定されているかに応じて、操作は無条件に許可または拒否されます。

1. YAML ファイルを使用して[変更用](#)、または[検証用](#)受付 Webhook オブジェクトを設定します。
2. 以下のコマンドを実行してオブジェクトを作成します。

```
oc create -f <file-name>.yaml
```

受付 Webhook オブジェクトの作成後、OpenShift Container Platform が新規設定を反映するまでに数秒の時間がかかります。

3. 受付 Webhook のフロントエンドサービスを作成します。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    role: webhook ❶
  name: <name>
spec:
  selector:
    role: webhook ❷
```

❶ ❷ Webhook をトリガーするための自由形式のラベルです。

4. 以下のコマンドを実行してオブジェクトを作成します。

```
oc create -f <file-name>.yaml
```

5. Webhook で制御する必要のある Pod に受付 Webhook 名を追加します。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    role: webhook ❶
  name: <name>
spec:
  containers:
    - name: <name>
      image: myrepo/myimage:latest
      imagePullPolicy: <policy>
      ports:
        - containerPort: 8000
```

❶ Webhook をトリガーするためのラベルです。



注記

独自のセキュアでポータブルな Webhook 受付サーバーをビルドする方法についてのエンドツーエンドの例については、[kubernetes-namespace-reservation プロジェクト](#)を参照し、ライブラリーについては [generic-admission-apiserver](#) を参照してください。

4.6.2.3. 受付 Webhook オブジェクトのサンプル

以下は、[namespace](#) が予約される場合に [namespace](#) の作成 を許可しない受付 Webhook のサンプルです。

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: namespace-reservations.admission.online.openshift.io
webhooks:
- name: namespace-reservations.admission.online.openshift.io
  clientConfig:
    service:
      namespace: default
      name: webhooks
    path:
/apis/admission.online.openshift.io/v1beta1/namespace-reservations
  caBundle: KUBE_CA_HERE
  rules:
  - operations:
    - CREATE
    apiGroups:
    - ""
    apiVersions:
    - "b1"
    resources:
    - namespaces
  failurePolicy: Ignore
```

以下は、**webhook** という名前の受付 Webhook によって評価される Pod のサンプルです。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    role: webhook
  name: webhook
spec:
  containers:
  - name: webhook
    image: myrepo/myimage:latest
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 8000
```

以下は Webhook のフロントエンドサービスです。

```
apiVersion: v1
```

```
kind: Service
metadata:
  labels:
    role: webhook
  name: webhook
spec:
  ports:
    - port: 443
      targetPort: 8000
  selector:
    role: webhook
```

4.7. 他の API オブジェクト

4.7.1. LimitRange

制限範囲は、Kubernetes [namespace](#) のリソースに設定される最小/最大の制限を実施するメカニズムを提供します。

制限範囲を namespace に追加することで、個別の Pod またはコンテナによって消費される CPU およびメモリの最小および最大量を施行できます。

4.7.2. ResourceQuota

Kubernetes は、[namespace](#) で作成されるオブジェクト数と、namespace 内のオブジェクト間で要求されるリソース合計量の両方を制限できます。これにより、namespace 内の複数のチームで単一の Kubernetes クラスターを共有でき、あるチームによって別のチームがクラスターリソース不足になることを防ぐことができます。

ResourceQuota についての詳細は、『[クラスター管理](#)』を参照してください。

4.7.3. リソース

Kubernetes の **Resource** は、Pod またはコンテナによって要求され、割り当てられ、消費されるものです。例として、メモリー (RAM)、CPU、ディスク時間、およびネットワーク帯域幅があります。

詳細は、『[開発者ガイド](#)』を参照してください。

4.7.4. シークレット

[シークレット](#) は、キー、パスワード、および証明書などの機密情報のストレージです。これらは所定の Pod でアクセスできますが、定義とは別に保持されます。

4.7.5. PersistentVolume

[永続ボリューム](#) は、クラスター管理者によってプロビジョニングされるインフラストラクチャーのオブジェクト (**PersistentVolume**) です。永続ボリュームは、ステートフルなアプリケーションの耐久性のあるストレージを提供します。

4.7.6. PersistentVolumeClaim

PersistentVolumeClaim オブジェクトは、[Pod 作成者によるストレージの要求](#)です。Kubernetes

は、要求を利用可能なボリュームのプールに対して一致させ、それらをバインドします。この要求は、Pod によってボリュームとして使用されます。Kubernetes はボリュームがこれを要求する Pod と同じノードで利用可能であることを確認します。

4.7.6.1. カスタムリソース

カスタムリソース は、API を拡張するか、独自の API をプロジェクトまたはクラスターに導入できるようにする Kubernetes API の拡張です。

リンク https://access.redhat.com/documentation/ja-jp/openshift_container_platform/3.9/html-single/cluster_administration/#admin-guide-custom-resources [カスタムリソースによる Kubernetes API の拡張] を参照してください。

4.7.7. OAuth オブジェクト

4.7.7.1. OAuthClient

OAuthClient は、[RFC 6749, section 2](#) に説明されているように OAuth クライアントを表します。

以下の **OAuthClient** オブジェクトは自動的に作成されます。

openshift-web-console	Web コンソールのトークンを要求するために使用されるクライアント
openshift-browser-client	対話式ログインを処理できるユーザーエージェントで /oauth/token/request でトークンを要求するために使用されるクライアント
openshift-challenging-client	WWW-Authenticate チャレンジを処理できるユーザーエージェントでトークンを要求するために使用されるクライアント

OAuthClient オブジェクト定義

```
kind: "OAuthClient"
accessTokenMaxAgeSeconds: null ❶
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "openshift-web-console" ❷
  selflink: "/oapi/v1/oauthClients/openshift-web-console"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01Z"
respondWithChallenges: false ❸
secret: "45e27750-a8aa-11e4-b2ea-3c970e4b7ffe" ❹
redirectURIs:
  - "https://localhost:8443" ❺
```

❶ アクセストークンの有効期間 (秒単位)([以下の説明](#)を参照してください)。

- 2 **name** は OAuth 要求の **client_id** パラメーターとして使用されます。
- 3 **respondWithChallenges** が **true** に設定される場合、**/oauth/authorize** への認証されていない要求は、設定される認証方法でサポートされている場合には **WWW-Authenticate** チャレンジを生じさせます。
- 4 **secret** パラメーターの値は、承認コードフローの **client_secret** パラメーターとして使用されます。
- 5 1 つ以上の絶対 URI を **redirectURIs** セクションに配置できます。承認要求と共に送信される **redirect_uri** パラメーターの前には、指定された **redirectURIs** のいずれかが付けられる必要があります。

accessTokenMaxAgeSeconds 値は、個々の OAuth クライアントについてのマスター設定ファイルのデフォルトの **accessTokenMaxAgeSeconds** 値を上書きします。クライアントにこの値を設定することにより、他のクライアントの有効期間に影響を与えることなく、クライアントのアクセストークンの有効期間を長く設定できます。

- **null** の場合、マスター設定ファイルのデフォルト値が使用されます。
- **0** に設定される場合、トークンは有効期限切れになりません。
- **0** よりも大きな値に設定される場合、クライアント用に発行されるトークンには指定された有効期限が設定されます。たとえば、**accessTokenMaxAgeSeconds: 172800** により、トークンは発行後 48 時間後に有効期限切れになります。

4.7.7.2. OAuthClientAuthorization

OAuthClientAuthorization は、特定の **OAuthClient** に特定のスコープが設定された **OAuthAccessToken** が付与されることについての **User** による承認を表します。

OAuthClientAuthorization オブジェクトの作成は、**OAuth** サーバーへの承認要求時に実行されます。

OAuthClientAuthorization オブジェクト定義

```
kind: "OAuthClientAuthorization"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "bob:openshift-web-console"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
  clientName: "openshift-web-console"
  userName: "bob"
  userID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
  scopes: []
```

4.7.7.3. OAuthAuthorizeToken

OAuthAuthorizeToken は、[RFC 6749, section 1.3.1](#) に説明されているように **OAuth** 承認コードを表します。

OAuthAuthorizeToken は、[RFC 6749, section 4.1.1](#) で説明されているように **/oauth/authorize** エンドポイントへの要求によって作成されます。

OAuthAuthorizeToken は次に、[RFC 6749, section 4.1.3](#) に説明されているように、**/oauth/token** エンドポイントへの要求で **OAuthAccessToken** を取得するために使用できます。

OAuthAuthorizeToken オブジェクト定義

```
kind: "OAuthAuthorizeToken"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "MDAwYjM5YjMtMzM1MC00NDY4LTkxODItOTA2OTE2YzE0M2Fj" ❶
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
clientName: "openshift-web-console" ❷
expiresIn: 300 ❸
scopes: []
redirectURI: "https://localhost:8443/console/oauth" ❹
userName: "bob" ❺
userID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" ❻
```

- ❶ **name** は、**OAuthAccessToken** を交換するために承認コードとして使用されるトークン名を表します。
- ❷ **clientName** 値は、このトークンを要求した **OAuthClient** です。
- ❸ **expiresIn** 値は **creationTimestamp** の有効期限 (秒単位) です。
- ❹ **redirectURI** 値は、このトークンが作成された承認フローでユーザーがリダイレクトされた場所です。
- ❺ **userName** は、このトークンが **OAuthAccessToken** の取得を許可するユーザーの名前を表します。
- ❻ **userID** は、このトークンが **OAuthAccessToken** の取得を許可するユーザーの **UID** を表します。

4.7.7.4. OAuthAccessToken

OAuthAccessToken は、[RFC 6749, section 1.4](#) に説明されているように **OAuth** アクセストークンを表します。

OAuthAccessToken は、[RFC 6749, section 4.1.3](#) に説明されているように **/oauth/token** エンドポイントへの要求によって作成されます。

アクセストークンは、API に対して認証を行うためにベアータークンとして使用されます。

OAuthAccessToken オブジェクト定義

```
kind: "OAuthAccessToken"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "ODli0GE5ZmMtYzcyYi00Nzk1LTg4MGEtNzQyZmUxZmUwY2Vh" ❶
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:02-00:00"
clientName: "openshift-web-console" ❷
```

```

expiresIn: 86400 ③
scopes: []
redirectURI: "https://localhost:8443/console/oauth" ④
userName: "bob" ⑤
userID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" ⑥
authorizeToken: "MDAwYjM5YjMtMzM1MC00NDY4LTkxODItOTA2OTEyZyE0M2Fj" ⑦

```

- ① **name** は、API に対して認証を行うためにベアラートークンとして使用されるトークン名です。
- ② **clientName** 値は、このトークンを要求した OAuthClient です。
- ③ **expiresIn** 値は **creationTimestamp** の有効期限 (秒単位) です。
- ④ **redirectURI** は、このトークンが作成された承認フローでユーザーがリダイレクトされた場所です。
- ⑤ **userName** は、このトークンが認証を許可するユーザーを表します。
- ⑥ **userID** は、このトークンが認証を許可するユーザーの **UID** を表します。
- ⑦ **authorizeToken** は、このトークンを取得するために使用される **OAuthAuthorizationToken** の名前です (ある場合)。

4.7.8. ユーザーオブジェクト

4.7.8.1. アイデンティティー

ユーザーが OpenShift Container Platform にログインする際に、設定された [アイデンティティープロバイダー](#) を使用して実行されます。これにより、ユーザーのアイデンティティーが決定され、その情報が OpenShift Container Platform に提供されます。

次に OpenShift Container Platform は **UserIdentityMapping** でその **Identity** を検索します。



注記

アイデンティティープロバイダーが **lookup** マッピング方法などで設定されている場合で、外部の LDAP システムを使用している場合には、この自動マッピングは実行されません。この場合、マッピングは手動で作成する必要があります。詳細は、[「Lookup マッピング方法」](#) を参照してください。

- **Identity** がすでに存在する場合でも、これが **User** にマップされていないと、ログインは失敗します。
- **Identity** がすでに存在し、これが **User** にマップされている場合、ユーザーにはマップされた **User** の **OAuthAccessToken** が付与されます。
- **Identity** が存在しない場合、**Identity**、**User**、および **UserIdentityMapping** が作成され、ユーザーにはマップされた **User** の **OAuthAccessToken** が付与されます。

Identity オブジェクト定義

```

kind: "Identity"
apiVersion: "user.openshift.io/v1"

```

```

metadata:
  name: "anypassword:bob" ❶
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
providerName: "anypassword" ❷
providerUserName: "bob" ❸
user:
  name: "bob" ❹
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" ❺

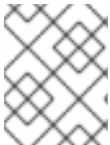
```

- ❶ アイデンティティー名は `providerName:providerUserName` の形式である必要があります。
- ❷ **providerName** はアイデンティティープロバイダーの名前です。
- ❸ **providerUserName** は、アイデンティティープロバイダーのスコップでこのアイデンティティーを一意に表す名前です。
- ❹ **user** パラメーターの **name** は、このアイデンティティーがマップされるユーザーの名前です。
- ❺ **uid** は、このアイデンティティーがマップされるユーザーの UID を表します。

4.7.8.2. ユーザー

User はシステムのアクターを表します。ユーザーには、[ロールをユーザーまたはグループに追加](#)してパーミッションが付与されます。

ユーザーオブジェクトは初回ログイン時に自動的に作成されるか、API で作成できます。



注記

/, :, および % を含む OpenShift Container Platform ユーザー名はサポートされません。

User オブジェクト定義

```

kind: "User"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "bob" ❶
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
identities:
  - "anypassword:bob" ❷
fullName: "Bob User" ❸

```

- ❶ **name** は、ロールをユーザーに追加する際に使用されるユーザー名です。
- ❷ **identities** の値は、このユーザーにマップされるアイデンティティーオブジェクトです。これはログインできないユーザーについて **null** または空にすることができます。

- ③ **fullName** 値は、ユーザーのオプションの表示名です。

4.7.8.3. UserIdentityMapping

UserIdentityMapping は **Identity** を **User** にマップします。

UserIdentityMapping を作成し、更新し、または削除することにより、**Identity** および **User** オブジェクトの対応するフィールドが変更されます。

Identity は単一の **User** にのみマップされるため、特定のアイデンティティとしてログインすると、**User** が明確に判別されます。

User には複数のアイデンティティをマップできます。これにより、複数のログイン方法で同じ **User** を識別できます。

UserIdentityMapping オブジェクト定義

```
kind: "UserIdentityMapping"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "anypassword:bob" ①
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
identity:
  name: "anypassword:bob"
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
user:
  name: "bob"
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
```

- ① **UserIdentityMapping** 名は、マップされた **Identity** 名に一致します。

4.7.8.4. グループ

Groupは、システム内のユーザーの一覧を表します。グループには、[ロールをユーザーまたはグループに追加](#)してパーミッションが付与されます。

Group オブジェクト定義

```
kind: "Group"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "developers" ①
  creationTimestamp: "2015-01-01T01:01:01-00:00"
users:
  - "bob" ②
```

- ① **name** は、ロールをグループに追加する際のグループ名です。

- ② **users** の値は、このグループのメンバーであるユーザーオブジェクトの名前です。

第5章 ネットワーク

5.1. ネットワーク

5.1.1. 概要

Kubernetes は、確実に Pod 間がネットワークで接続されるようにし、内部ネットワークから IP アドレスを各 Pod に割り当てます。こうすることで、Pod 内の全コンテナが同じホスト上にあるかのように動作します。各 Pod に IP アドレスを割り当てると、ポートの割り当て、ネットワーク、名前の指定、サービス検出、負荷分散、アプリケーション設定、移行などの点で、Pod を物理ホストや仮想マシンのように扱うことができます。

Pod 間のリンクを作成する必要はないので、この IP アドレスを使用して Pod 間で直接相互に通信することは推奨されません。代わりに、「サービス」を作成して、そのサービスと対話することを推奨します。

5.1.2. OpenShift Container Platform DNS

フロントエンドサービスやバックエンドサービスなど、複数の「サービス」を実行して複数の Pod で使用している場合には、フロントエンド Pod がバックエンドサービスと通信できるように、ユーザー名、サービス IP などの環境変数を作成します。サービスが削除され、再作成された場合には、新規の IP アドレスがサービスに割り当てられるので、サービス IP の環境変数の更新値を取得するには、フロントエンド Pod を再作成する必要があります。さらに、バックエンドサービスは、フロントエンド Pod を作成する前に作成し、サービス IP が正しく生成され、フロントエンド Pod に環境変数として提供できるようにする必要があります。

このような理由から、サービスの DNS やサービスの IP/ポートでサービスに到達できるように、OpenShift Container Platform には DNS が組み込まれています。OpenShift Container Platform は、サービスの DNS クエリーに応答するマスターで「SkyDNS」を実行することで、スプリット DNS をサポートします。マスターは、デフォルトで、ポート 53 をリスンします。

ノードが起動すると、以下のメッセージで、Kubelet が正しくマスターに解決されていることが分かります。

```
0308 19:51:03.118430    4484 node.go:197] Started Kubelet for node
openshiftdev.local, server at 0.0.0.0:10250
I0308 19:51:03.118459    4484 node.go:199] Kubelet is setting 10.0.2.15
as a
DNS nameserver for domain "local"
```

2 番目のメッセージが表示されない場合は、Kubernetes サービスが利用できない可能性があります。

ノードホストで、各コンテナのネームサーバーのフロントにマスター名が追加され、コンテナの検索ドメインはデフォルトでは、**<Pod_namespace>.cluster.local** になります。コンテナは、ノード上の他のネームサーバーよりも先にネームサーバーのクエリーをマスターに転送します。これは、Docker 形式のコンテナではデフォルトの動作です。マスターは、以下の形式の **.cluster.local** ドメインでクエリーに対応します

表5.1 DNS 名の例

オブジェクトタイプ	例
デフォルト	<Pod_namespace>.cluster.local

オブジェクトタイプ	例
サービス	<service>.<Pod_namespace>.svc.cluster.local
エンドポイント	<name>.<namespace>.endpoints.cluster.local

これにより、新しいサービスを取得するためにフロントエンドの Pod を再起動し、サービスに対して新しい IP を作成せずに済みます。また、Pod がサービスの DNS を使用するので、環境変数を使用する必要がなくなります。さらに、DNS は変更しないので、設定ファイルで **db.local** としてデータベースサービスを参照できます。また、検索はサービス IP に対して解決するため、ワイルドカードの検索もサポートされます。さらにサービス名 (つまり DNS) が事前に確立しているので、フロントエンド Pod の前にバックエンドサービスを作成する必要がなくなります。

この DNS 構造では、ポータル IP はサービスに割り当てられず、kube-proxy は負荷を分散しないか、またはエンドポイントのルーティングを提供するヘッドレスサービスに対応しています。サービス DNS は依然として使用でき、サービスの Pod 毎に 1 つずつある複数のレコードに対応し、クライアントによる Pod 間のラウンドロビンが可能にします。

5.2. OPENSIFT SDN

5.2.1. 概要

OpenShift Container Platform は、Software Defined Networking (SDN) アプローチを使用して、クラスターのネットワークを統合し、OpenShift Container Platform クラスターの Pod 間の通信を可能にします。OpenShift SDN により、このような Pod ネットワークが確立され、メンテナンスされます。OpenShift SDN は Open vSwitch (OVS) を使用してオーバーレイネットワークを設定します。

OpenShift SDN では以下のように、Pod ネットワークを構成するための SDN プラグインを 3 つ提供します。

- **ovs-subnet** プラグインはオリジナルのプラグインで、Pod が他の Pod やサービスすべてと通信できる「フラットな」Pod ネットワークを提供します。
- **ovs-multitenant** プラグインは、Pod とサービスをプロジェクトごとに分離します。プロジェクト毎に、一意の Virtual Network ID (VNID) を受け取り、プロジェクトに割り当てられた Pod からのトラフィックを特定します。別のプロジェクトからの Pod は、別のプロジェクトの Pod やサービスに対するパケットの送信や受信ができません。
ただし、VNID 0 を受け取るプロジェクトは、他の Pod すべてとの間で通信できるという面で、追加の特権があります。OpenShift Container Platform クラスターでは、**default** プロジェクトに VNID 0 が割り当てられています。これにより、ロードバランサーなど、特定のサービスがクラスター内の他の全 Pod との間でスムーズに通信できるようにします。
- **ovs-networkpolicy** プラグインでは、プロジェクト管理者が NetworkPolicy オブジェクトを使用して分離ポリシーを設定できます。



注記

マスターおよびノードでの SDN の設定に関する情報は、「[SDN の設定](#)」を参照してください。

5.2.2. マスター上の設計

OpenShift Container Platform マスターでは、OpenShift SDN が、**etcd** に保存されている、ノードのレジストリーを管理します。システム管理者がノードを登録すると、OpenShift SDN がクラスターネットワークから未使用のサブネットを割り当てて、レジストリーのこのサブネットを保存します。ノードが削除されると、OpenShift SDN はレジストリーからサブネットを削除し、このサブネットを割り当て可能とみなします。

デフォルトの設定では、クラスターネットワークは **10.128.0.0/14** ネットワーク (つまり **10.128.0.0 - 10.131.255.255**) で、ノードには **/23** サブネット (つまり **10.128.0.0/23**、**10.128.2.0/23**、**10.128.4.0/23** など) が割り当てられます。つまり、このクラスターネットワークには、512 個のサブネットをノードに割り当てることができ、特定のノードには 510 個のアドレスが割り当てられ、このノードで実行中のコンテナに割り当てることができます。クラスターネットワークのサイズやアドレス範囲、さらにホストのサブネットサイズは、設定可能です。



注記

サブネットが次に大きいオクテットに拡張される場合には、共有の octet でサブネットのビットが 0 のものが先に割り当てられます。たとえば、ネットワークが 10.1.0.0/16 で **hostsubnetlength=6** が指定されている場合には、10.1.0.0/26 および 10.1.1.0/26 から 10.1.255.0/26 が 10.1.0.64/26、10.1.1.64/26 が埋まる前に、割り当てられます。こうすることで、サブネットを把握しやすくなります。

マスター上の OpenShift SDN では、ローカル (マスター) ホストが、クラスターネットワークにアクセスできるように設定されないで、マスターホストは、ノードとして実行されない限り、クラスターネットワーク経由で Pod にアクセスできません。

ovs-multitenant プラグインを使用する場合には、OpenShift SDN マスターはプロジェクトの作成や削除を監視し、VXLAN VNID をプロジェクトに割り当てます。この VNID は後で、ノードが正しくトラフィックを分離するために使用します。

5.2.3. ノード上の設計

ノードでは OpenShift SDN は先に、前述のレジストリーに、SDN マスターを持つローカルホストを登録し、マスターがノードにサブネットを割り当てられるようにします。

次に OpenShift SDN は、3 つのネットワークデバイスを作成して設定します。

- **br0**: Pod 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以外のフロールールも設定します。
- **tun0**: OVS の内部ポート (**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、外部の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外部ネットワークに NAT 経由でアクセスできるように、**netfilter** およびルーティング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。

Pod がホストで起動されるたびに、OpenShift SDN は以下を行います。

1. 対象の Pod に、ノードのクラスターサブネットから、空いている IP アドレスを割り当てます。
2. ホスト側の Pod の veth インターフェースペアを OVS ブリッジ **br0** に割り当てます。

3. OpenFlow ルールを OVS データベースに追加して、新規の Pod にアドレス指定されたトラフィックを正しい OVS ポートにルーティングします。
4. **ovs-multitenant** プラグインの場合は、Pod からのトラフィックには、その Pod の VNID をタグ付けし、トラフィックの VNID が Pod の VNID (または特権のある VNID 0) と一致する場合にはその Pod にトラフィックを許可するという OpenFlow ルールを追加します。一致しないトラフィックは、一般的なルールで除外されます。

OpenShift SDN ノードは、SDN マスターからのサブネットの更新も監視します。新しいサブネットが追加された場合には、リモートサブネットの宛先 IP アドレスを持つパケットが **vxlان0 (br0 のポート 1)** に移動してネットワーク送信されるように、**br0** に OpenFlow ルールを追加します。**ovs-subnet** プラグインは VNID 0 が指定された VXLAN に全パケットを送信しますが、**ovs-multitenant** プラグインは、ソースコンテナに対して適切な VNID を使用します。

5.2.4. パケットフロー

A と B の 2 つのコンテナがあり、コンテナ A の **eth0** をベースにするピア仮想 Ethernet デバイスの名前が **vethA**、コンテナ B の **eth0** のピア名が **vethB** とします。



注記

Docker サービスによるピアの仮想 Ethernet デバイスの使用方法を理解するには、[Docker の高度なネットワークに関するドキュメント](#) を参照してください。

まず、コンテナ A がローカルホストにあり、コンテナ B もローカルホストにあると仮定します。コンテナ A からコンテナ B のパケットフローは以下のようになります。

eth0 (A の netns) → vethA → br0 → vethB → eth0 (B の netns)

次に、コンテナ A がローカルホストに、コンテナ B がクラスターネットワーク上のリモートホストにあると想定します。その場合には、コンテナ A からコンテナ B のパケットフローは以下のようになります。

eth0 (A の netns) → vethA → br0 → vxlan0 → ネットワーク [1] → vxlan0 → br0 → vethB → eth0 (B の netns)

最後に、コンテナ A が外部ホストに接続すると、トラフィックは以下のようになります。

eth0 (A の netns) → vethA → br0 → tun0 → (NAT) → eth0 (物理デバイス) → インターネット

パケット配信の意思決定はほぼ、OVS ブリッジ **br0** の OpenFlow ルールをもとに行われ、プラグインのネットワークアーキテクチャーを簡素化し、ルーティングを柔軟化します。**ovs-multitenant** プラグインの場合は、OpenFlow ルールをもとにした意思決定により、強制的な「[ネットワーク分離](#)」が可能になります。

5.2.5. ネットワーク分離

ovs-multitenant プラグインを使用して、ネットワーク分離を実現できます。デフォルト以外のプロジェクトに割り当てられた Pod からパケットが送信される場合は、OVS ブリッジ **br0** により、このパケットに、プロジェクトが割り当てた VNID のタグを付けます。パケットが、ノードのクラスターサブネットに含まれる別の IP アドレスに転送される場合には、OVS ブリッジは、VNID が一致する場合のみ、宛先の Pod に対するこのパケットの配信を許可します。

パケットが別のノードから VXLAN トンネル経由で受信された場合には、トンネル ID を VNID として使用し、OVS ブリッジは、トンネル ID が宛先の Pod の VNID に一致する場合にのみ、ローカル Pod へのパケットの配信を許可します。

他のクラスターサブネットが宛先のパケットは、その VNID でタグ付けされ、クラスターサブネットを所有するノードのトンネルの宛先アドレスが指定された VXLAN トンネルに配信されます。

前述の通り、VNID 0 は、任意の VNID が指定されたトラフィックは VNID 0 が割り当てられた Pod に、VNID 0 が指定されたトラフィックは任意の Pod に送信できる点で特権があります。デフォルトの OpenShift Container Platform プロジェクトには VNID 0 が割り当てられています。他のプロジェクトにはすべて、一意の分離可能な VNID が割り当てられています。クラスター管理者はオプションで、管理者 CLI を使用してプロジェクトの「Pod ネットワークを管理」できます。

5.3. 利用可能な SDN プラグイン

OpenShift Container Platform は、OpenShift Container Platform と Kubernetes の間のインターフェースとして、Kubernetes [Container Network Interface \(CNI\)](#) をサポートします。Software Defined Network (SDN) プラグインを使用することで、ネットワーク機能がユーザーのネットワークのニーズに対応します。必要に応じて、CNI インターフェースをサポートするプラグインをさらに追加できます。

5.3.1. OpenShift SDN

OpenShift SDN は、デフォルトで Ansible ベースのインストール手順の一部としてインストールされ、設定されます。詳細情報は、「[OpenShift SDN](#)」のセクションを参照してください。

5.3.2. サードパーティーの SDN プラグイン

5.3.2.1. Flannel SDN

flannel は、コンテナ専用設計された仮想ネットワーク層です。OpenShift Container Platform は、デフォルトの Software-Defined Networking (SDN) コンポーネントの代わりに、ネットワークコンテナとして flannel を使用できます。これは、OpenStack など、SDN にも依存するクラウドプロバイダープラットフォーム内で OpenShift Container Platform を実行している場合や、両プラットフォームを通してパケットを 2 回カプセル化することを防ぐ場合に便利です。

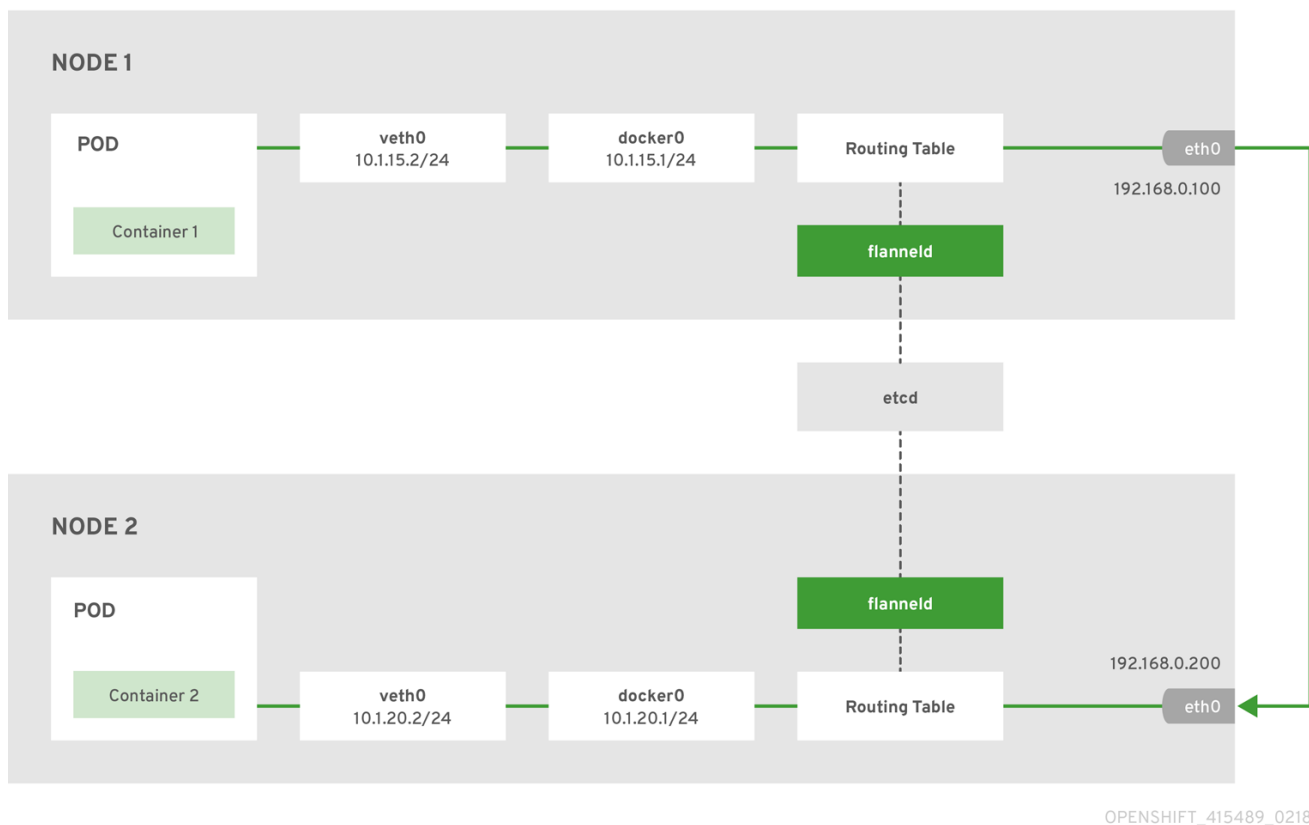
アーキテクチャー

OpenShift Container Platform は、**flannel** を **host-gw** モードで実行し、コンテナ間のルートをマッピングします。ネットワーク内の各ホストは、**flanneld** と呼ばれるエージェントを実行します。このエージェントは以下を行います。

- ホストごとに一意のサブネットを管理する
- ホスト上の各コンテナに IP アドレスを割り当てる
- 別のホスト上であっても、コンテナ間のルートをマッピングする

各 **flanneld** エージェントは、この情報を中央の **etcd** ストアに提供し、ホスト上の他のエージェントがパケットを、**flannel** ネットワーク内の他のコンテナにルーティングできるようにします。

以下の図は、**flannel** ネットワークを使用したコンテナ間のアーキテクチャーおよびデータフローを示します。



OPENSIFT_415489_0218

ノード 1 には以下のルートが含まれます。

```
default via 192.168.0.100 dev eth0 proto static metric 100
10.1.15.0/24 dev docker0 proto kernel scope link src 10.1.15.1
10.1.20.0/24 via 192.168.0.200 dev eth0
```

ノード 2 には以下のルートが含まれます。

```
default via 192.168.0.200 dev eth0 proto static metric 100
10.1.20.0/24 dev docker0 proto kernel scope link src 10.1.20.1
10.1.15.0/24 via 192.168.0.100 dev eth0
```

5.3.2.2. Nuage SDN

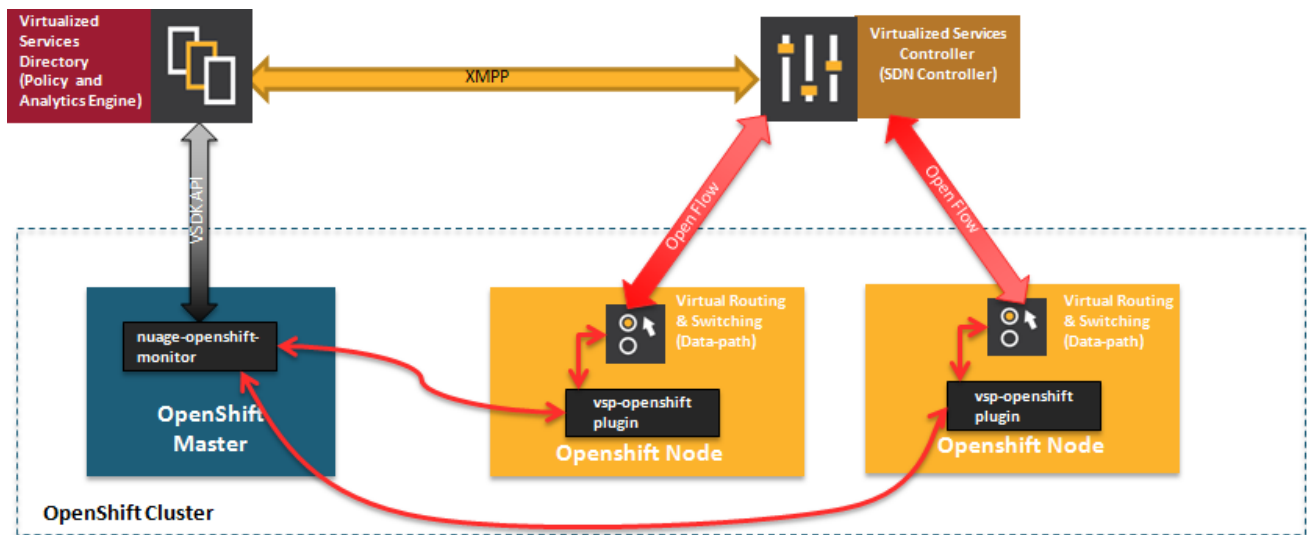
「[Nuage Networks](#)」のSDNソリューションは、OpenShift Container Platform クラスターの Pod に対して、非常にスケーラブルで、ポリシーベースのオーバーレイネットワークを提供します。Nuage SDN は、Ansible ベースのインストール手順の一部としてインストールして設定することができます。Nuage SDN での OpenShift Container Platform のインストールおよびデプロイ方法に関する情報は、「[標準インストール](#)」のセクションを参照してください。

[Nuage Networks](#) は、Virtualized Services Platform (VSP) と呼ばれる、非常にスケーラブルなポリシーベースの SDN プラットフォームを提供します。Nuage VSP は、データプレーン用にオープンソースの Open vSwitch とともに、SDN Controller を使用します。

Nuage は、オーバーレイを使用して、OpenShift Container Platform と VM およびベアメタルサーバーからなる他の環境の間をポリシーベースで接続できるようにします。プラットフォームのリアルタイムアナリティクスエンジンでは、OpenShift Container Platform アプリケーションの可視化およびセキュリティ監視を実現します。

Nuage VSP は OpenShift Container Platform と統合し、DevOps チームが直面するネットワークのラグを取り除くことで、ビジネスアプリケーションがすばやく起動、更新できるようにします。

図5.1 Nuage VSP と OpenShift Container Platform との統合



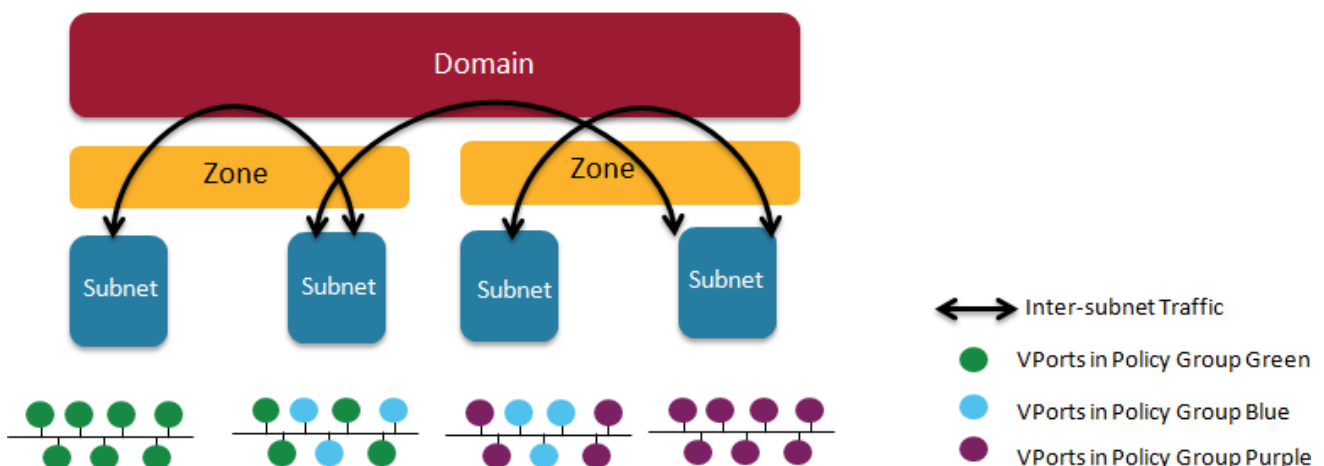
統合を行う固有のコンポーネントが 2 つあります。

1. **nuage-openshift-monitor** サービス。OpenShift Container Platform マスターノードで個別のサービスとして実行されます。
2. **vsp-openshift** プラグイン。クラスターの各ノードで OpenShift Container Platform ランタイムにより呼び出されます。

Nuage Virtual Routing and Switching ソフトウェア (VRS) は、オープンソースの Open vSwitch をベースにしており、データパス転送を行います。VRS は各ノードで実行され、コントローラーからポリシー設定を取得します。

Nuage VSP の用語

図5.2 Nuage VSP のビルディングブロック



1. **ドメイン**: 組織には 1 つまたは複数のドメインが含まれます。ドメインは単一の「レイヤー 3」の領域を指します。標準のネットワーク用語では、ドメインは、VRF インスタンスと同じ位置づけです。
2. **ゾーン**: ゾーンは、ドメインの配下に定義されます。ゾーンは、直接ネットワーク上のなにかにマッピングされるわけではなく、ゾーンの全エンドポイントが同じポリシーセットに準拠するなど、ポリシーが関連付けられているオブジェクトとして機能します。
3. **サブネット**: サブネットはゾーンの配下に定義されます。サブネットは、ドメインインスタンス内の固有のレイヤー 2 サブネットを指します。サブネットは、ドメイン内で一意で他とは異なる

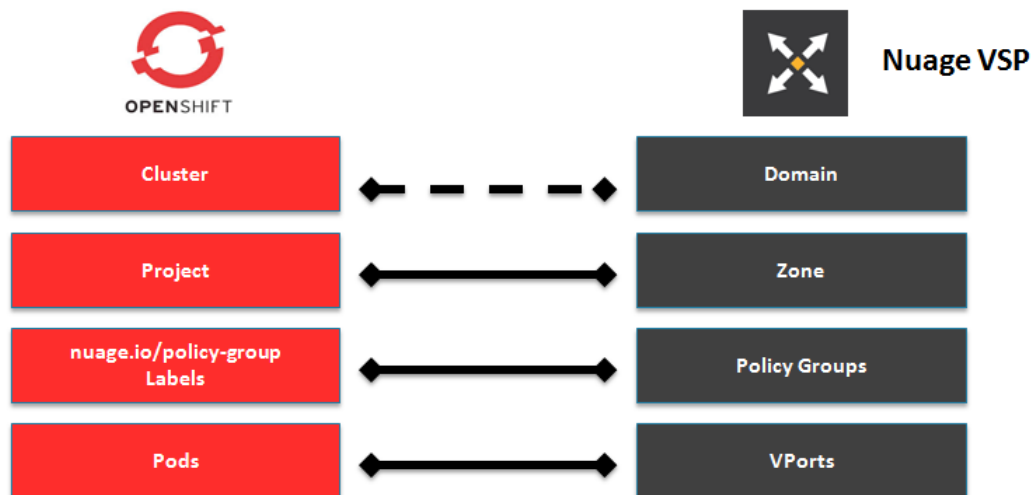
ります。つまり、ドメイン内のサブネットは、重複したり、標準の IP サブネット定義に従って他のサブネットを含めたりすることもできません。

4. VPort: VPort は、ドメイン階層の新しいレベルで、より粒度の高い設定を可能にするために設計されました。コンテナや VM に加え、ホストやブリッジインターフェースにアタッチには VPort も使用し、ベアメタルサーバー、アプリケーション、レガシー VLAN に接続できるようにします。
5. ポリシーグループ: ポリシーグループは VPort のコレクションです。

コンストラクトのマッピング

OpenShift Container Platform のコンセプトの多くは、Nuage VSP のコンストラクトに直接マッピングできます。

図5.3 Nuage VSP および OpenShift Container Platform のマッピング



Nuage サブネットは、OpenShift Container Platform ノードにマッピングされませんが、特定のプロジェクトのサブネットは、OpenShift Container Platform 内の複数のノードに対応できます。

OpenShift Container Platform で起動する Pod は VSP で作成された仮想ポートに変換されます。**vsp-openshift** プラグインは、VRS と対話し、VSC 経由で VSD からその仮想ポートのポリシーを取得します。ポリシーグループは、複数の Pod のグループ化がサポートされていますが、同じポリシーが適用されている必要があります。現在、Pod は、「[オペレーションワークフロー](#)」を使用してポリシーグループに割り当てることができます。このワークフローでは、ポリシーグループは VSD の管理者ユーザーが作成します。ポリシーグループに含まれる Pod は、Pod の仕様で **nuage.io/policy-group** ラベルとして指定されます。

統合コンポーネント

Nuage VSP は、2 つの主要コンポーネントを使用して OpenShift Container Platform と統合します。

1. **nuage-openshift-monitor**
2. **vsp-openshift plugin**

nuage-openshift-monitor

nuage-openshift-monitor は、プロジェクト、サービス、ユーザー、ユーザーグループなどが作成されていないか、OpenShift Container Platform API サーバーを監視するサービスです。



注記

複数のマスターがある高可用性の (HA) OpenShift Container Platform クラスターの場合には、**nuage-openshift-monitor** プロセスは、機能性に変更を加えずに、全マスター上で個別に実行されます。

開発者のワークフローでは、**nuage-openshift-monitor** も、VSD REST API を実行して OpenShift Container Platform コンストラクトを VSP コンストラクトにマッピングすることで、VSD オブジェクトを自動作成します。各クラスターインスタンスは、Nuage VSP の単一ドメインにマッピングします。これにより、Nuage でエンタープライズのドメインインスタンスごとに 1 つ設定するなど、特定のエンタープライズで複数のクラスターをインストールできます。各 OpenShift Container Platform プロジェクトは、Nuage VSP のクラスターのドメインに含まれるゾーンにマッピングされます。**nuage-openshift-monitor** で、プロジェクトの追加、削除が検出された場合に、対象のプロジェクトに対応する VSDK API を使用してゾーンをインスタンス化し、そのゾーンにサブネットのブロックを割り当てます。さらに、**nuage-openshift-monitor** は、このプロジェクトのネットワークマクログループも作成します。同様に、**nuage-openshift-monitor** でサービスの追加や削除が検出された場合には、サービス IP に対応するネットワークマクロを作成して、そのネットワークマクロを該当のプロジェクトのネットワークマクログループに割り当てて (ラベルを使用したユーザー提供のネットワークマクログループもサポートされています)、対象のサービスへの通信を有効化します。

開発者のワークフローでは、ゾーン内で作成された Pod はすべて、そのサブネットプールからの IP を取得します。**nuage-openshift-monitor** が、master-config ファイルのプラグイン固有のパラメーター 2 つをもとにして、サブネットプールを割り当て、管理します。ただし、実際の IP アドレスの解決および vport ポリシーの解決は、プロジェクトの作成時にインスタンス化されたドメイン/ゾーンをもとに、VSD が行います。最初のサブネットプールが足りなくなった場合には、**nuage-openshift-monitor** はクラスターの CIDR から追加のサブネットを検出し、特定のプロジェクトに割り当てます。

オペレーションのワークフローでは、アプリケーションまたは Pod 仕様に Nuage が認識するラベルを指定して、Pod と固有のユーザー定義ゾーンやサブネットを解決します。ただし、これは、**nuage-openshift-monitor** を使用して開発者ワークフローで作成したゾーンやサブネットの Pod を解決するために使用できません。



注記

オペレーションワークフローでは、管理者は VSD コンストラクトを事前に作成し、Pod を特定のゾーン/サブネットにマッピングして、OpenShift エンティティー (ACL ルール、ポリシーグループ、ネットワークマクロ、ネットワークマクログループ) 間の通信を可能にします。Nuage ラベルの使用方法に関する説明は『[Nuage VSP Openshift Integration Guide](#)』に記載されています。

vsp-openshift Plug-in

vsp-openshift ネットワークプラグインは、OpenShift Container Platform ランタイムが各 OpenShift Container Platform ノードで呼び出します。このプラグインは、ネットワークプラグイン init および Pod の設定、破棄、ステータスフックを実装します。vsp-openshift プラグインは、Pod に IP アドレスも割り当てます。特に、VRS (転送エンジン) と通信して、IP 情報を Pod に設定します。

5.4. 利用可能なルータープラグイン

ルーターは、ノードに割り当てて OpenShift クラスターのトラフィックを制御することができます。OpenShift はデフォルトのルーターとして HAProxy を使用しますが、オプションも提供されています。

5.4.1. デフォルトの HAProxy ルーター

5.4.2. HAProxy テンプレートルーター

HAProxy テンプレートのルーター実装は、テンプレートルータープラグインの参照実装です。これは、**openshift3/ose-haproxy-router** リポジトリを使用して、テンプレートルータープラグインとともに、HAProxy インスタンスを実行します。

テンプレートルーターには、以下の 2 つのコンポーネントがあります。

- エンドポイントとルートを監視して変更をもとに HAProxy の再読み込みをトリガーするラッパー
- ルートとエンドポイントをベースに HAProxy 設定ファイルをビルドするコントローラー



注記

[HAProxy ルーター](#) はバージョン 1.8.1 を使用します。

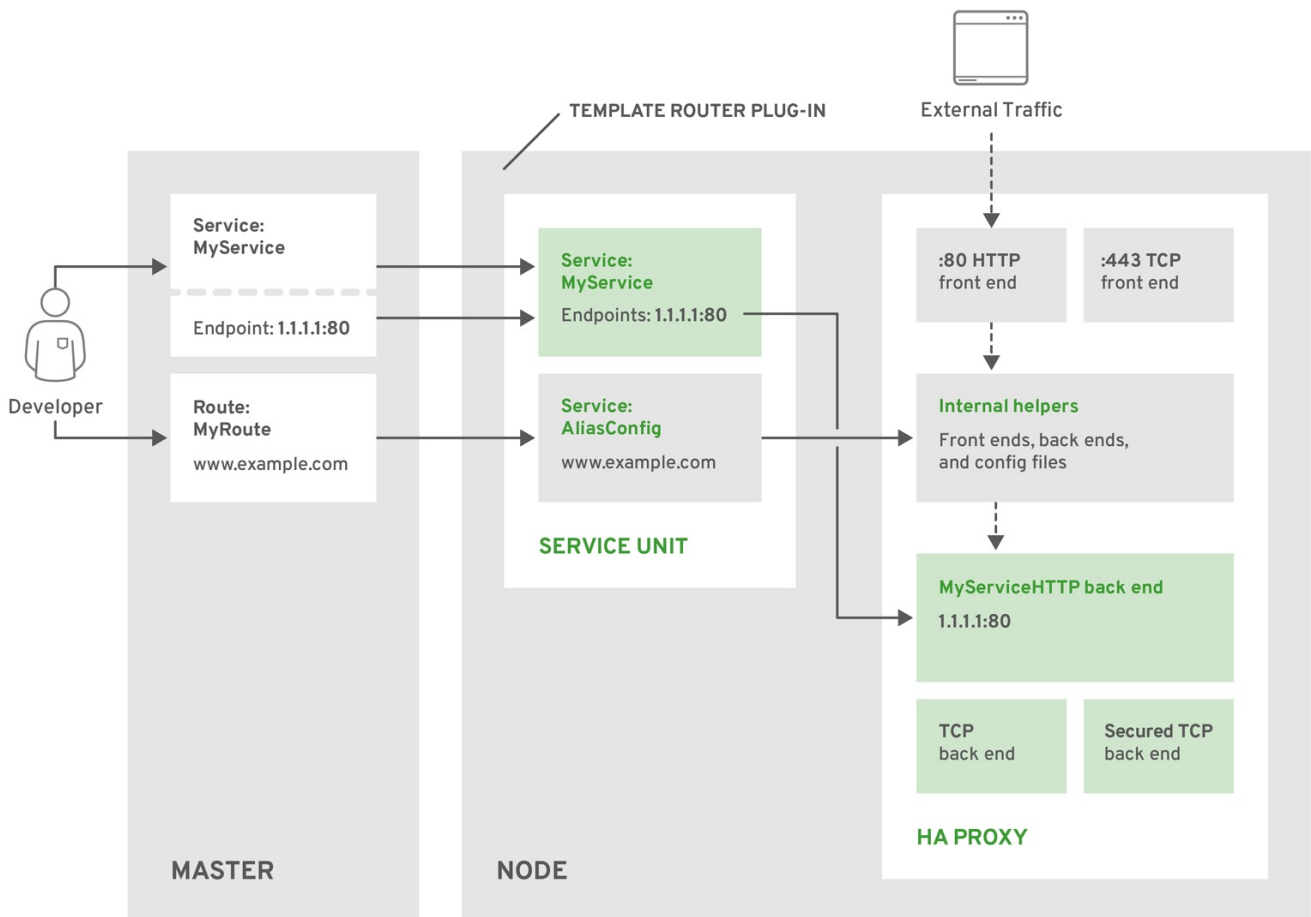
コントローラーおよび HAProxy は、Pod 内に常駐しており、デプロイメント設定で管理されます。ルーターの設定プロセスは、**oc adm router** コマンドで自動化されています。

コントローラーは、HAProxy プロキシが正常であるかどうか、またルートとエンドポイントに変更がないかを監視します。変更が検出されたら、新しい haproxy-config ファイルを作成して、HAProxy を再起動します。haproxy-config ファイルは、ルーターのテンプレートファイルと OpenShift Container Platform からの情報をベースに構築されます。

HAProxy テンプレートファイルは、必要に応じてカスタマイズして、OpenShift Container Platform で現在サポートされていない機能をサポートすることができます。[HAProxy マニュアル](#) では、HAProxy がサポートする全機能を説明しています。

以下の図では、データがプラグインを使用してマスターから最終的に HAProxy 設定にどのように移動するかが記載されています。

図5.4 HAProxy ルーターデータフロー



OPENSIFT_415489_0218

HAProxy テンプレートルーターメトリクス

HAProxy ルーターは、外部のメトリクスコレクションや集計システム (例 Prometheus、statsd) で消費されるように、[Prometheus 形式](#) のメトリクスを提供して公開します。ルーターは、[HAProxy CSV 形式](#) のメトリクスを提供するように設定したり、まったくルーターメトリクスを提供しないように設定したりできます。

メトリクスは、ルーターコントローラーおよび HAProxy の両方から 5 秒ごとに取得されます。ルーターメトリクスカウンターは、ルーターのデプロイ時に 0 から開始され、経時的に増加します。HAProxy メトリクスカウンターは、HAProxy が再読み込みされるたびに 0 にリセットされます。ルーターはフロントエンド、バックエンド、サーバー毎に HAProxy 統計を収集します。バックエンドでは複数のサーバーを使用できるので、サーバーが 500 台を超える場合には、これらのサーバーではなく、バックエンドに関するレポートを作成することで、リソースの使用量を減らします。

この統計は、利用可能な HAProxy [統計](#) のサブセットです。

以下の HAProxy メトリクスは定期的に収集され、Prometheus 形式に変換されます。フロントエンドについてはすべて、"F" カウンターが収集されます。バックエンドごとにカウンターが収集される場合には、サーバーごとの "S" サーバーカウンターが収集されます。それ以外の場合は、バックエンドごとに "B" カウンターが収集され、サーバーカウンターは収集されません。

詳細は、[ルーター環境変数](#)を参照してください。

以下の表を参照してください。

列 1 - HAProxy CSV 統計のインデックス

列 2

F	フロントエンドメトリクス
b	サーバーのしきい値が原因でサーバーメトリクスを表示しない場合のバックエンドメトリクス
B	サーバーメトリクスを表示する場合のバックエンドメトリクス
S	サーバーメトリクス

列 3 - カウンター

列 4 - カウンターの説明

インデックス	使用法	カウンター	説明
2	bBS	current_queue	現在キューにある要求で、サーバーに割り当てられていない要求の数。
4	FbS	current_sessions	現在アクティブなセッション数。
5	FbS	max_sessions	アクティブなセッションの最大実数。
7	FbBS	connections_total	接続の合計数。
8	FbS	bytes_in_total	受信バイトの現在の合計。
9	FbS	bytes_out_total	送信バイトの現在の合計。
13	bS	connection_errors_total	接続エラーの合計。
14	bS	response_errors_total	応答エラーの合計。
17	bBS	up	現在のバックエンドのヘルスステータス (1 = UP、0 = DOWN)。
21	S	check_failures_total	失敗したヘルスチェックの合計数。

24	S	downtime_seconds_total	合計ダウンタイム (秒)。
33	FbS	current_session_rate	直近の 1 秒間で、1 秒あたりの現在のセッション数。
35	FbS	max_session_rate	1 秒あたりの最大セッション実数。
40	FbS	http_responses_total	HTTP 応答合計数、コード 2xx。
43	FbS	http_responses_total	HTTP 合計応答数、コード 5xx。
60	bS	http_average_response_latency_milliseconds	直近の要求 1024 件のうちの HTTP 応答 (ミリ秒単位)。

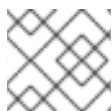
ルーターコントローラーは、以下のアイテムを収集します。これらは、Prometheus 形式のメトリクスでのみ提供されます。

名前	説明
template_router_reload_seconds	ルーターの再読み込みにかかる時間を秒単位で測定します。
template_router_write_config_seconds	ルーター設定のディスクへの書き込みにかかる時間を秒単位で測定します。
haproxy_exporter_up	最後に成功した haproxy の収集です。
haproxy_exporter_csv_parse_failures	CSV の解析時のエラー数です。
haproxy_exporter_scrape_interval	次の収集が許可されるまでの秒単位の時間です (データのサイズに比例します)。
haproxy_exporter_server_threshold	追跡したサーバー数と現在のしきい値です。
haproxy_exporter_total_scrapes	現在の合計 HAProxy 収集数です。
http_request_duration_microseconds	マイクロ秒単位の HTTP 要求のレイテンシーです。
http_request_size_bytes	バイト単位の HTTP 要求サイズです。

http_response_size_bytes	バイト単位の HTTP 応答サイズです。
openshift_build_info	OpenShift のビルドに使用された major、minor、git commit、git version でラベル付けされた定数値 '1' のメトリクスです。
ssh_tunnel_open_count	SSH トンネルを開放しようとして試行した合計数です。
ssh_tunnel_open_fail_count	SSH トンネルを開放しようとして失敗した合計数です。

5.4.3. F5 BIG-IP® ルータープラグイン

ルーターは、クラスターにトラフィックを送信する手段の 1 つです。F5 BIG-IP® ルータープラグインは、利用可能な「[ルータープラグイン](#)」の 1 つです。



注記

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

F5 ルータープラグインは、お使いの環境で既存の **F5 BIG-IP®** システムと統合します。F5 iControl REST API を使用するには、**F5 BIG-IP®** バージョン 11.4 以降が必要です。F5 ルーターは、HTTP vhost および要求パスで一致する [unsecured](#)、[edge termination](#)、[re-encryption termination](#) および [passthrough termination](#) ルートをサポートします。

F5 ルーターは、[HAProxy テンプレートルーター](#) と同等機能を備えており、OpenShift Enterprise 2 での **F5 BIG-IP®** に対応した追加機能も含まれます。以前のバージョンで使用されていた **routing-daemon** と比較すると、F5 ルーターは以下の機能を追加でサポートします。

- パスベースのルーティング (ポリシールールを使用)、
- Re-encryption (クライアントおよびサーバーの SSL プロファイルを使用した実装)
- 暗号化接続のパススルー (SNI プロトコルを解析し、サーバー名ルックアップ用に F5 ルーターが管理するデータグループを使用する iRule で実装)



注記

パススルールートは特別なケースです。**F5 BIG-IP®** 自体が HTTP 要求を認識できず、パスを検証できないので、パスベースのルーティングは技術的に、パススルールートと併用できません。同じ制限が、テンプレートルーターにも適用されます。これは、パススルー暗号化の技術的制限で、OpenShift Container Platform の技術的制限ではありません。

5.4.3.1. SDN 経由での Pod に対するトラフィックのルーティング

F5 BIG-IP® は [OpenShift SDN](#) 外にあるので、クラスター管理者は **F5 BIG-IP®** と SDN 上にあるホスト (通常は OpenShift Container Platform ノードホスト) 間でピアツーピアトンネルを作成する必要があります。この **ramp ノード** は、Pod に対して [スケジュール不可](#) と設定して、**F5 BIG-IP®** ホストのゲートウェイ以外として機能しないようにすることができます。また、このようなホストを複

数設定して、冗長化のために OpenShift Container Platform **ipfailover** 機能を使用できます。**F5 BIG-IP®** ホストは、トンネルのリモートエンドポイントに、**ipfailover** VIP を使用するように設定する必要があります。

5.4.3.2. F5 統合の詳細

F5 ルーターの操作は、以前のバージョンで使用していた OpenShift Container Platform **routing-daemon** とよく似ています。いずれも REST API 呼び出しを使用して以下を行います。

- プールを作成、削除する
- これらのプールにエンドポイントを追加して、このプールからエンドポイントを削除する
- ポリシールールが vhost をベースにしてプールにルーティングするように設定する

いずれも **scp** と **ssh** コマンドを使用してカスタムの TLS/SSL 証明書を **F5 BIG-IP®** にアップロードします。

F5 ルーターは、仮想サーバー上のプールとポリシールールを以下のように設定します。

- ユーザーが OpenShift Container Platform でルートを作成するか、削除すると、ルーターは **F5 BIG-IP®** にルート用のプールを作成して (すでにプールが存在しない場合)、TLS 以外のルートの HTTP vserver またはエッジまたは再暗号化ルートの HTTPS vserver など、適切な vserver のポリシーに対してルールを追加/削除します。edge および re-encrypt ルートの場合には、ルーターも TLS 証明書と鍵をアップロードして設定します。ルーターは、ホストおよびパスペースのルートをサポートします。



注記

パススルーは特別なケースです。これをサポートするには、SNI ClientHello ハンドシェイクレコードを解析して、F5 データグループでサーバー名をルックアップするための iRule を記述する必要があります。ルーターはこの iRule を作成して、この iRule と vserver を関連付け、パススルールートの作成/削除に伴い、F5 データグループを更新します。この実装の詳細以外は、パススルールートは、他のルートと同じように機能します。

- OpenShift Container Platform でサービスを作成すると、ルートは **F5 BIG-IP®** にプールを追加します (プールが存在しない場合)、このサービスにエンドポイントが作成/削除されると、ルーターは適切なプールメンバーを追加/削除します。
- ユーザーがルートと、特定のプールに関連付けられた全エンドポイントを削除すると、ルーターはそのプールを削除します。

5.4.3.3. F5 ネイティブ統合

F5 と OpenShift Container Platform をネイティブで統合するには、OpenShift SDN で作成されるので、オーバーレイネットワーク上の Pod に到達できるように、ramp ノードを設定する必要はありません。

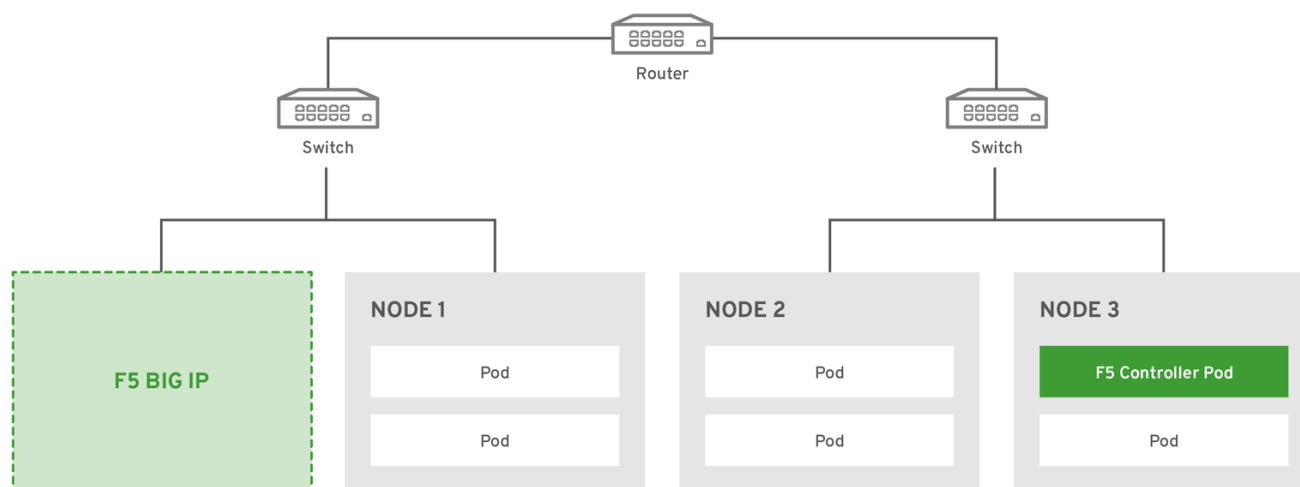
また、**F5 BIG-IP®** アプライアンスのバージョン 12.x 以降のみが、このセクションに記載されているネイティブ統合に対応しています。また、適切に統合を行うために **sdn-services** アドオンライセンスが必要になります。バージョン 11.x の場合は、**ramp ノードを設定してください**。

接続

F5 アプライアンスは、L3 接続で OpenShift Container Platform クラスターに接続できます。OpenShift Container Platform のノード間では、L2 スイッチの接続性は必要ありません。F5 アプ

イアンスでは、複数のインターフェースを使用して統合を管理できます。

- 管理インターフェース: F5 アプライアンスの Web コンソールに到達する
- 外部インターフェース: 受信 Web トラフィック用に仮想サーバーを設定する
- 内部インターフェース: アプライアンスをプログラムして、Pod に到達する



OPENSIFT_415489_0218

F5 コントローラー Pod は、アプライアンスへの **admin** 権限があります。F5 イメージは、OpenShift Container Platform クラスタ (ノード上にスケジュール) で起動して、iControl REST API を使用してポリシーで仮想サーバーをプログラムし、VxLAN デバイスを設定します。

データフロー: パケットから Pod へ



注記

このセクションでは、パケットが Pod に到達する方法および Pod がパケットに到達する方法について説明します。これらのアクションは、ユーザーではなく、F5 コントローラー Pod と F5 アプライアンスにより実行されます。

ネイティブで統合されると、F5 アプライアンスは VxLAN のカプセル化を使用して直接 Pod に到達します。この統合は、OpenShift Container Platform が **openshift-sdn** をネットワークプラグインとして使用している場合のみ機能します。**openshift-sdn** プラグインは、このプラグインが作成するオーバーレイネットワークに対して、VxLAN のカプセル化を採用します。

Pod と F5 アプライアンスの間でデータパスを正常に確立するには以下を行います。

1. F5 は、Pod 向けの VxLAN パケットをカプセル化する必要があります。これには、**sdn-services** ライセンスアドオンが必要です。VxLAN デバイスを作成する必要があります。Pod オーバーレイネットワークはこのデバイス経由でルーティングする必要があります。
2. F5 は Pod の VTEP IP アドレスを認識する必要があります。これは、Pod が配置されているノードの IP アドレスです。
3. F5 は、Pod 向けのパケットをカプセル化する時に、オーバーレイネットワークに使用する **source-ip** を知っておく必要があります。これは **ゲートウェイアドレス** として知られていません。
4. OpenShift Container Platform ノードは、F5 ゲートウェイアドレスがなにか (戻りトラフィックの VTEP アドレス) を知っておく必要があります。このアドレスは、内部インター

フェースのアドレスでなければなりません。クラスターの全ノードは、自動的にこれを学習する必要があります。

5. オーバーレイネットワークはマルチテナントに対応しているので、F5 は **admin** を代表する VxLAN ID を使用して F5 が全テナントに到達できるようにする必要があります。手動作成した **hostsubnet (Pod.network.openshift.io/fixed-vnid-host: 0)** でアノテーションを指定することで、F5 により **vnid** が **0** (OpenShift Container Platform の **admin** namespace でデフォルトの **vnid**) の全パケットがカプセル化されるようにします。

ghost hostsubnet は、設定の一部として手動で作成します。これは、3 番目と 4 番目の要件を満たします。F5 コントローラー Pod の起動時に、F5 アプライアンスが適切にプログラムされるように、この新しい **ghost hostsubnet** が提供されます。



注記

サブネットがクラスターのノードに割り当てられるので、**ghost hostsubnet** という用語が使用されます。ただし、実際には、クラスターの実際のノードではなく、外部のアプライアンスによるハイジャックが行われます。

1 番目の要件は、F5 コントローラー Pod の起動時に、F5 コントローラー Pod が満たします。2 番目の要件は、F5 コントローラー Pod が対応しますが、継続プロセスです。クラスターに追加される新規ノードごとに、コントローラー Pod は VxLAN デバイスの VTEP FDB のエントリーを作成します。コントローラー Pod は、クラスターの **nodes** リソースへのアクセス権が必要ですが、これは、サービスアカウントに適切な権限を割り当てることで対応できます。以下のコマンドを使用してください。

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

F5 ホストからのデータフロー



注記

以下のアクションは、ユーザーではなく F5 コントローラー Pod によって実行されます。

1. 宛先 Pod はパケットの F5 仮想サーバーで識別されます。
2. VxLAN 動的 FDB は Pod の IP アドレスで検索されます。MAC アドレスが見つかる場合はステップ 5 に進みます。
3. Pod の MAC アドレスを求める ARP 要求で、VTEP FDB の全エントリーをいっぱいにします。Pod の MAC アドレスと VTEP を値として使用して、VxLAN ダイナミック FDB にエントリーが追加されます。
4. VxLAN ヘッダーで IP パケットをカプセル化します。Pod の MAC およびノードの VTEP が、VxLAN 動的 FDB からの値として指定されます。
5. ARP を送信し、ホストの周辺にあるキャッシュを確認して、VTEP の MAC アドレスを計算します。
6. F5 ホストの内部アドレスでパケットを送信します。

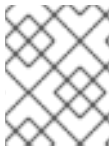
データフロー: トラフィックを F5 ホストに返す



注記

以下のアクションは、ユーザーではなく F5 コントローラー Pod によって実行されます。

1. Pod は、宛先を F5 ホストの VxLAN ゲートウェイアドレスとしてパケットを戻します。
2. ノードの **openvswitch** は、このパケットの VTEP が F5 ホストの内部インターフェースアドレスであるかどうか判断します。これは、ghost **hostsubnet** の作成時に分かります。
3. VxLAN パケットは、F5 ホストの内部インターフェースに送信されます。



注記

マルチテナントを回避するように、VNID はあらかじめ、データフロー全体で **0** に固定されます。

5.5. ポート転送

5.5.1. 概要

OpenShift Container Platform は、Kubernetes に組み込まれている機能を活用して、[Pod へのポート転送をサポートします](#)。これは、HTTP と **SPDY** または **HTTP/2** などの多重化ストリーミングプロトコルを使用して実装されます。

開発者は「[CLI](#)」を使用して Pod にポート転送できます。CLI は、ユーザーが指定した各ローカルポートをリッスンし、「[記載のプロトコル](#)」を使用して転送します。

5.5.2. サーバー操作

Kubelet は、クライアントからのポート転送要求を処理します。要求を受信すると、応答をアップグレードし、クライアントがポート転送ストリームを作成するまで待機します。新規ストリームを受信したら、ストリームと Pod ポート間のデータをコピーします。

アーキテクチャ的には、Pod のポートに転送するオプションがあります。OpenShift Container Platform でサポートされる現在の実装はノードホストで直接 **nsenter** を呼び出し、Pod ネットワークの namespace に入り、**socat** を呼び出してストリームと Pod のポート間のデータをコピーします。ただし、カスタムの実装には、**nsenter** と **socat** のバイナリーをホストにインストールしなくともいいように、これらのバイナリーを実行する「ヘルパー」Pod の実行が含まれています。

5.6. リモートコマンド

5.6.1. 概要

OpenShift Container Platform は Kubernetes に組み込まれている機能を活用し、コンテナでのコマンド実行をサポートします。これは、HTTP と **SPDY** または **HTTP/2** などの多重化ストリーミングプロトコルを使用して実装されます。

開発者は「[CLI を使用](#)」して、コンテナでリモートコマンドを実行します。

5.6.2. サーバー操作

Kubelet は、クライアントからのリモート実行要求を処理します。要求を受信すると応答をアップグレードして、要求ヘッダーを評価してどのストリーム (**stdin**、**stdout** および/または **stderr**) を受信するか判断し、クライアントがストリームを作成するまで待機します。

Kubelet が全ストリームを受信したら、コンテナでコマンドを実行して、ストリームとコマンドの **stdin**、**stdout** および **stderr** を適切にコピーします。コマンドが中断されたら、Kubelet はアップグレードされた接続と基盤の接続を終了します。

アーキテクチャ的に、コンテナでコマンドを実行するオプションがあります。OpenShift Container Platform で現在サポートされている実装では、ノードホストで **nsenter** を直接呼び出して、コマンド実行前に、ノードホストがコンテナの namespace に入ることができるようにします。ただし、カスタム実装には **docker exec** の使用や、ホストでインストールする必要がある **nsenter** バイナリーが必須とならないように **nsenter** を実行する「ヘルパー」コンテナの実行が含まれる場合があります。

5.7. ルート

5.7.1. 概要

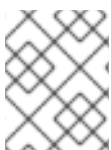
OpenShift Container Platform ルートは、外部クライアントが名前ですべてのサービスに到達できるように、**www.example.com** などのホスト名で [サービス](#) を公開します。

ホスト名の DNS 解決はルーティングとは別に処理されます。管理者が、OpenShift Container Platform ルーターを実行する OpenShift Container Platform ノードを解決するように [DNS ワイルドカードエントリ](#) を設定している場合があります。異なるホスト名を使用している場合には、DNS レコードを個別に変更して、ルーターを実行するノードを解決する必要があります。

各ルーターは名前 (63 文字に制限)、サービスセクター、およびオプションのセキュリティー設定で構成されています。

5.7.2. ルーター

OpenShift Container Platform 管理者は、OpenShift Container Platform のノードに **ルーター** をデプロイできるので、[開発者が作成した](#) ルートを外部クライアントが利用できるようになります。OpenShift Container Platform のルーティング層はプラグ可能で、複数の [ルータープラグイン](#) がデフォルトで提供、サポートされています。



注記

ルーターのデプロイに関する情報は、[『インストールと設定ガイド』](#) を参照してください。

ルーターは、サービスセクターを使用して、「[サービス](#)」と、サービスをバックアップするエンドポイントを検索します。ルーターとサービス両方が負荷分散を提供する場合には、OpenShift Container Platform はルーターの負荷分散を使用します。ルーターはサービスの IP アドレスに関連の変更がないかを検出して、その設定に合わせて変化します。これは、カスタムルーターが API オブジェクトの変更を外部のルーティングソリューションに通信できるので、便利です。

要求のパスは、1 つまたは複数のルーターに対して、ホスト名の DNS を解決することから始まります。推奨の方法は、複数のルーターインスタンスでバックされる 1 つまたは複数の仮想 IP (VIP) アドレスを参照するワイルドカード DNS エントリでクラウドドメインを定義する方法です。クラウドドメイン外の名前とアドレスを使用するルートには、個別の DNS エントリ設定が必要です。

VIP アドレスがルーターよりも少ない場合には、アドレス分のルーターが **active** で、残りは

passive になります。passive ルーターはホットスタンバイ ルーターとして知られています。たとえば、VIP アドレスが 2 つ、ルーターが 3 つの場合には、「active-active-passive」構成になります。ルーターの VIP 設定に関する詳細情報は、「[高可用性](#)」を参照してください。

ルートは、ルーターセット間で [シャード化](#) されます。管理者は、クラスター全体でシャード化を設定し、ユーザーはプロジェクトに含まれる namespace にシャード化を設定できます。オペレーターは、シャード化すると、複数のルーターグループを定義できるようになります。このグループ内の各ルーターはトラフィックのサブネット 1 つにのみ対応できます。

OpenShift Container Platform ルーターは、外部のホスト名マッピングと、ルーターに区別情報を直接渡すプロトコルを使用して [サービス](#) エンドポイントの負荷分散を行います。ルーターは、送信先を判断できるように、プロトコルにホスト名が存在している必要があります。

ルータープラグインはデフォルトで、ホストポート 80 (HTTP) および 443 (HTTPS) にバインドできます。つまり、配置されていないと、これらのポートは使用されないの、ルーターはノードに配置されている必要があります。または、ルーターは、**ROUTER_SERVICE_HTTP_PORT** および **ROUTER_SERVICE_HTTPS_PORT** 環境変数を設定して、他のポートをリスンするように設定してください。

ルーターは、ホストノードのポートにバインドされるので、ルーターがホストネットワークを使用している場合には (デフォルト)、各ノードに 1 つしかこれらのポートをリスンするルーターを配置できません。クラスターネットワークは、全ルーターがクラスター内のすべての Pod にアクセスできるように設定します。

ルーターは以下のプロトコルをサポートします。

- HTTP
- HTTPS (SNI あり)
- WebSockets
- SNI 付きの TLS



注記

WebSocket トラフィックは、同じルート規則を使用し、他のトラフィックと同じ TLS 終端タイプをサポートします。

セキュアな接続を確立するには、クライアントとサーバーに共通する [暗号化](#) を取り決める必要があります。時間が経つにつれ、よりセキュリティの高く、新しい暗号化が利用でき、クライアントソフトウェアに統合されます。以前のクライアントは陳腐化し、セキュリティの低い、以前の暗号化は使用が停止されます。デフォルトでは、ルーターは、一般的に入手できる、幅広いクライアントに対応します。ルーターは、任意のクライアントをサポートする暗号化の中で選択したものを使用し、セキュリティが低い暗号化を使用しないように設定できます。

5.7.2.1. テンプレートルーター

テンプレートルーター は、ルーターの一種で、特定のインフラストラクチャー情報を基盤のルーター実装に提供します。以下に例を示します。

- エンドポイントおよびルートを監視するラッパーです。
- 消費可能なフォームに保存されるエンドポイントとルートデータです。
- 内部の状態を設定可能なテンプレートに渡し、テンプレートを実行します。

- 再読み込みスクリプトを呼び出します。

5.7.3. 利用可能なルータープラグイン

検証された利用可能なルータープラグインについては、「[利用可能なプラグイン](#)」のセクションを参照してください。

これらのルーターのデプロイ方法については、「[ルーターのデプロイ](#)」を参照してください。

5.7.4. スティックセッション

スティッキーセッションの実装は、基盤のルーター設定により異なります。デフォルトの HAProxy テンプレートは、**balance source** ディレクティブを使用してスティッキーセッションを実装し、ソース IP をもとに分散されます。さらに、このテンプレートルータープラグインは、サービス名と namespace を基盤の実装に渡します。これは、ピア間で同期させるスティックテーブルの実装など、より高度な設定に使用できます。

スティッキーセッションは、ユーザー体験の向上のため、ユーザーのセッションからの全トラフィックが確実に同じ Pod に移動されるようにします。ユーザーの要求を満たしながら、Pod は後続の要求で使用できるように、データをキャッシュします。たとえば、バックエンド Pod が 5 つと負荷分散ルーターが 2 つあるクラスターでは、要求を処理するルーターがどれであっても、同じ Pod で、同じ Web ブラウザーからの web トラフィックを受信できるように確保できます。

ルーティングトラフィックを同じ Pod に返すことが望まれる場合でも、保証はできませんが、HTTP ヘッダーを使用して、cookie を設定し、最後の接続で使用した Pod を判断できます。ユーザーがアプリケーに別の要求を送信した場合には、ブラウザーが cookie を再送することで、ルーターでトラフィックの送信先が分かります。

クラスター管理者は、スティッキネスをオフにして、他の接続とパススルールートを分割することも、完全にスティッキネスをオフにすることもできます。

デフォルトでは、パススルールートのスティッキーセッションは、**source** 「[負荷分散ストラテジー](#)」を使用して実装します。すべてのパスルート用には、**ROUTER_TCP_BALANCE_SCHEME** 「[環境変数](#)」で、個別ルート用には、**haproxy.router.openshift.io/balance** 「[ルート固有のアノテーション](#)」で、デフォルトを変更することができます。

他の種類のルートはデフォルトで **leastconn** 「[負荷分散ストラテジー](#)」を使用しますが、これは **ROUTER_LOAD_BALANCE_ALGORITHM** 「[環境変数](#)」を使用して変更できます。また、個別ルートには **haproxy.router.openshift.io/balance** 「[ルート固有のアノテーション](#)」を使用して変更することができます。

注記

cookie は、HTTP トラフィックを表示できないので、パススルールートで設定できません。代わりに、ソース IP アドレスをベースに数が計算され、バックエンドを判断します。

バックエンドが変わった場合には、トラフィックは誤ったサーバーに送られ、スティッキネスが低くなります。ロードバランサーを使用する場合は (ソース IP が表示されない)、同じ番号が全接続に設定され、トラフィックが同じ Pod に送信されます。

さらに、このテンプレートルータープラグインは、サービス名と namespace を基盤の実装に渡します。これは、ピア間で同期させるスティックテーブルの実装など、より高度な設定に使用できます。

このルーター実装固有の設定は、ルーターコンテナの `/var/lib/haproxy/conf` ディレクトリーにある **haproxy-config.template** ファイルに保存されます。ファイルは「[カスタマイズ可能です](#)」。



注記

source の [負荷分散ストラテジー](#) は、NAT の設定が原因で、外部のクライアント IP アドレスを区別しないので、送信元の IP アドレス (HAProxy リモート) は同じです。HAProxy ルーターが **hostNetwork: true** で実行されない限り、すべての外部クライアントは単一の Pod にルーティングされます。

5.7.5. ルーターの環境変数

このセクションで説明されているアイテムはすべて、ルーターの **デプロイメント設定** に環境変数を指定して設定を変更するか、**oc set env** コマンドを使用します。

```
$ oc set env <object_type>/<object_name> KEY1=VALUE1 KEY2=VALUE2
```

以下は例になります。

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=127.0.0.1
ROUTER_LOG_LEVEL=debug
```

表5.2 ルーターの環境変数

変数	デフォルト	説明
DEFAULT_CERTIFICATE		TLS サーバー証明書を公開しないルートに使用するデフォルトの証明書の内容。PEM 形式。
DEFAULT_CERTIFICATE_DIR		tls.crt というファイルを含むディレクトリーへのパス。 tls.crt が PEM ファイルでなく、秘密鍵も含む場合には、同じディレクトリー内の tls.key というファイルと先に統合されます。PEM 形式のコンテンツは、デフォルトの証明書として使用されます。これは、 DEFAULT_CERTIFICATE または DEFAULT_CERTIFICATE_PATH が指定されていない場合のみ使用されます。
DEFAULT_CERTIFICATE_PATH		TLS サーバー証明書を公開しないルートに使用するデフォルト証明書へのパス。PEM 形式。 DEFAULT_CERTIFICATE が指定されていない場合のみ使用されます。
DROP_SYN_DURING_RESTART	false	true に設定されている場合は、iptables ルールを有効にして、再起動中に特定の packets をドロップして、シームレスに再起動できるようにします。詳細は、 「インストールガイド」 を参照してください。
EXTENDED_VALIDATION	true	true の場合は、ルーターにより、証明書が構造的に正しいことを確認します。CA に対して証明書は検証されません。テストをオフにするには、 false に設定します。
NAMESPACE_LABELS		監視する namespace に適用するラベルセレクターです。空はすべてを意味します。

変数	デフォルト	説明
PROJECT_LABELS		監視するプロジェクトに適用するラベルセクターです。空はすべてを意味します。
RELOAD_SCRIPT		ルーターの再読み込みに使用する再読み込みスクリプトのパス。
ROUTER_ALLOWED_DOMAINS		ルートのホスト名のみを含めることができるドメインのコンマ区切りの一覧。ドメインに含まれるサブドメインを使用できます。 ROUTER_DENIED_DOMAINS オプションは、このオプションに指定されている値を上書きします。これが設定されている場合は、許可されているドメイン以外はすべて拒否されます。
ROUTER_BACKEND_PROCESS_ENDPOINTS		テンプレート関数 processEndpointsForAlias の使用時にエンドポイントをどのように処理すべきかを指定する文字列。有効な値は ["shuffle", ""] です。"shuffle" は、全呼び出しごとに要素を無作為に決定します。デフォルトの動作は、事前に決定した順番に、値が返されます。
ROUTER_BIND_PORTS_AFTER_SYNC	false	true または TRUE に設定されている場合には、ルーターは、完全な同期状態になるまでポートにバインドされません。'true' または 'TRUE' に設定されていない場合は、ルーターはポートにバインドされ、即座に要求処理を開始しますが、ルートで読み込まれないものもあります。
ROUTER_COOKIE_NAME		cookie 名を指定して、内部で生成したデフォルト名を上書きします。名前は、大文字、小文字、数字、"_" または "-" を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。
ROUTER_COMPRESSION_MIME	"text/html text/plain text/css"	圧縮するスペースで区切られた mime タイプの一覧です。
ROUTER_DENIED_DOMAINS		ルートのホスト名に含めることができないドメインのコンマ区切りの一覧。ドメインに含まれるサブドメインを使用できません。 ROUTER_ALLOWED_DOMAINS オプションを上書きします。
ROUTER_ENABLE_COMPRESSION		true または TRUE の場合、可能な場合には応答を圧縮します。
ROUTER_LISTEN_ADDR	0.0.0.0:1936	ルーターメトリクスのリッスンアドレスを設定します。
ROUTER_LOG_LEVEL	警告	syslog サーバーに送信するログレベルです。

変数	デフォルト	説明
ROUTER_MAX_CONNECTIONS	20000	同時接続の最大数です。
ROUTER_METRICS_HAPROXY_SERVER_THRESHOLD	500	
ROUTER_METRICS_HAPROXY_EXPORTED		CSV 形式で収集されるメトリクスです。例: defaultSelectedMetrics = []int{2, 4, 5, 7, 8, 9, 13, 14, 17, 21, 24, 33, 35, 40, 43, 60}
ROUTER_METRICS_HAPROXY_BASE_SCRAPE_INTERVAL	5s	
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	
ROUTER_METRICS_TYPE	haproxy	HAProxy ルーターのメトリクスを生成します (haproxy のみがサポートされている値です)。
ROUTER_OVERRIDE_HOSTNAME		true に設定されている場合、 ROUTER_SUBDOMAIN のテンプレートのあるルートの spec.host 値を上書きします。
ROUTER_SERVICE_HTTPS_PORT	443	HTTPS 要求をリッスンするポートです。
ROUTER_SERVICE_HTTP_PORT	80	HTTP 要求をリッスンするポートです。
ROUTER_SERVICE_NAME	パブリック	ルーターがルートステータスで自らを識別する名前です。
ROUTER_CANONICAL_HOSTNAME		ルーターステータスに表示されるルーターの (オプションの) ホスト名です。
ROUTER_SERVICE_NAMESPACE		ルーターがルーターステータスで自らを識別する namespace です。 ROUTER_SERVICE_NAME が使用されている場合は必須です。
ROUTER_SERVICE_NO_SNI_PORT	10443	一部のフロントエンドからバックエンドへの通信に使用される内部ポートです (以下を参照してください)。

変数	デフォルト	説明
ROUTER_SERVICE_SNI_PORT	10444	一部のフロントエンドからバックエンドへの通信に使用される内部ポートです (以下を参照してください)。
ROUTER_SUBDOMAIN		spec.host なしでルートのホスト名を生成するために使用されるテンプレートです (例: <code>\${name}-\${namespace}.myapps.mycompany.com</code>)。
ROUTER_SYSLOG_ADDRESS		ログメッセージを送信するアドレスです。空の場合は無効になります。
ROUTER_SYSLOG_FORMAT		設定されている場合は、基盤のルーター実装で使用するデフォルトのログ形式が上書きされます。この値は、基盤のルーター実装の仕様に従う必要があります。
ROUTER_TCP_BALANCE_SCHEME	ソース	パススルールートを行うための複数のエンドポイント用「 負荷分散ストラテジー 」。利用可能なオプションは source 、 roundrobin または leastconn です。
ROUTER_LOAD_BALANCE_ALGORITHM	leastconn	複数のエンドポイントを持つルート用の「 負荷分散エンドポイント 」。利用可能なオプションは、 source 、 roundrobin および leastconn です。
ROUTE_LABELS		監視するルートに適用するラベルセクターです。何も指定しない場合はすべてを意味します。
STATS_PASSWORD		ルーターの統計にアクセスするのに必要なパスワード (ルーターの実装がサポートする場合)
STATS_PORT		統計を公開するポート (ルーターの実装がサポートする場合)。設定されていない場合は統計は公開されません。
STATS_USERNAME		ルーターの統計にアクセスするために必要なユーザー名 (ルーターの実装がこれをサポートしている場合)。
TEMPLATE_FILE	<code>/var/lib/haproxy/conf/custom/haproxy-config-custom.template</code>	HAProxy テンプレートへのパス (コンテナイメージ内)。
ROUTER_USE_PROXY_PROTOCOL		true または TRUE に設定されている場合には、HAProxy は受信接続がポート 80 と 443 上で PROXY プロトコルを使用していることを想定します。ソース IP アドレスは、ロードバランサーが Amazon ELB などのプロトコルをサポートする場合には、ロードバランサーをパススルーすることができます。

変数	デフォルト	説明
ROUTER_ALLOW_WILDCARD_ROUTES		true または TRUE が設定されている場合には、ルーターの受付チェックを合格するという Subdomain のワイルドカードポリシーが指定されたルートは、HAProxy ルーターがサービスを提供します。
ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK		namespace 所有権ポリシーを緩和するために true に設定されます。
ROUTER_STRICT_SNI		strict-sni
ROUTER_CIPHERS	intermediate	バインドでサポートされる「 暗号 」のセットを指定します。



注記

同じマシンで複数のルーターを実行する場合には、ルーターがリッスンするポート (**ROUTER_SERVICE_SNI_PORT** および **ROUTER_SERVICE_NO_SNI_PORT**) を変更する必要があります。これらのポートは、マシン上で一意であれば、何でも指定できます。また、これらのポートは外部には公開されません。

ルータータイムアウト変数

TimeUnits は数字、その後に単位を指定して表現します。**us** *(マイクロ秒)、**ms** (ミリ秒、デフォルト)、**s** (秒)、**m** (分)、**h** *(時間)、**d** (日)

正規表現: `[1-9][0-9]*(us|ms|s|m|h|d)`

ROUTER_BACKEND_CHECK_INTERVAL	5000ms	バックエンドでの後続の liveness チェックの時間の長さ。
ROUTER_CLIENT_FIN_TIMEOUT	1s	クライアントがルートに接続する場合の TCP FIN タイムアウトの期間を制御します。接続切断のために送信された FIN が指定の時間内に応答されない場合には、HAProxy が接続を切断します。小さい値を設定し、ルーターでリソースをあまり使用していない場合には、リスクはありません。
ROUTER_DEFAULT_CLIENT_TIMEOUT	30s	クライアントがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_CONNECT_TIMEOUT	5s	最大接続時間。

ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	1s	ルーターからルートをバックグする Pod の TCP FIN タイムアウトを制御します。
ROUTER_DEFAULT_SERVER_TIMEOUT	30s	サーバーがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_TUNNEL_TIMEOUT	1h	TCP または WebSocket 接続が開放された状態で保つ時間数。websockets/tcp 接続がある場合 (および HAProxy が再読み込みされる度) に、以前の HAProxy プロセスが指定された時間分、開放された状態に保たれます。
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	300s	新しい HTTP 要求が表示されるまで待機する最大時間を設定します。この値が低すぎる場合には、ブラウザおよびアプリケーションの keepalive 値が低くなりすぎて、問題が発生する可能性があります。タイムアウトの値はインタラクティブで、組み合わせることができます。詳しい情報は以下の注記セクションを参照してください。
ROUTER_SLOWLORIS_TIMEOUT	10s	HTTP 要求の伝送にかかる時間。
RELOAD_INTERVAL	5s	ルーターが新しい変更を受け入れるために再読み込みを許可する最小頻度。
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	HAProxy メトリクスの収集タイムアウト。

注記

有効なタイムアウト値には、想定した個別のタイムアウトではなく、特定の変数を合計した値に指定することができます。

例: **ROUTER_SLOWLORIS_HTTP_KEEPALIVE** は **timeout http-keep-alive** を調節し、デフォルトで **300s** に設定されていますが、haproxy は **tcp-request inspect-delay** も待機し、値は **5s** に設定されているので、今回の場合には、合計のタイムアウトは **300s** に **5s** を加えた値になります。

5.7.6. 負荷分散ストラテジー

ルートに複数のエンドポイントがある場合には、HAProxy は選択した負荷分散ストラテジーをもとに、エンドポイント間に要求を分散します。これは、セッションの最初の要求など、永続情報が利用できない場合に適用されます。

ストラテジーには以下のいずれか使用できます。

- **roundrobin**: 各エンドポイントは、加重に従い、順番に使用されます。これは、サーバーの処理が均等に分散される場合に最もスムーズで公平なアルゴリズムです。

- **leastconn**: 接続数が最も少ないエンドポイントが要求を受信します。接続数が最も少ないエンドポイントが複数ある場合には、ラウンドロビンが実行されます。LDAP、SQL、TSE など、セッションが非常に長い場合にこのアルゴリズムを使用してください。HTTP など、短いセッションを通常使用するプロトコルでの使用を目的とはしていません。
- **source**: ソース IP アドレスは、ハッシュ化され、実行中サーバーの合計加重で分割されて、どのサーバーが要求を受信するか指定します。これにより、サーバーが終了/起動しない限り、同じクライアント IP アドレスは常に同じサーバーに到達します。実行中のサーバー数が変化したことで、ハッシュの結果が変化した場合には、多くのクライアントが異なるサーバーに転送されます。このアルゴリズムは一般的にパススルールートで使用されます。

ROUTER_TCP_BALANCE_SCHEME 環境変数 はパススルールートのデフォルトストラテジーを設定します。**ROUTER_LOAD_BALANCE_ALGORITHM 環境変数** は、残りのルートに対してルーターのデフォルトストラテジーを設定します。[ルート固有のアノテーション](#)、[haproxy.router.openshift.io/balance](#) を使用して、個別のルートを制御できます。

5.7.7. HAProxy Strict SNI

デフォルトでは、ホストは HTTPS または TLS SNI 要求のルートを解決しないので、デフォルト証明書が **503** 応答の一部として、呼び出し元に返されます。これにより、デフォルト証明書が公開され、不正な証明書がサイトに提供されるので、セキュリティの問題を引き起こす可能性があります。バインド用の HAProxy **strict-sni** オプションを使用するとデフォルト証明書の使用が抑制されます。

ROUTER_STRICT_SNI 環境変数はバインド処理を制御します。**true** または **TRUE** に設定されている場合には、**strict-sni** が HAProxy バインドに追加されます。デフォルト設定は **false** です。

このオプションは、ルーターの作成時または後の追加時に設定できます。

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

これは、**ROUTER_STRICT_SNI=true** を設定できます。

5.7.8. ルーターの暗号スイート

各クライアント (例: Chrome 30 または Java8) には、ルーターにセキュアに接続するために使用する暗号スイートが含まれます。ルーターには、接続を完了させるには、暗号化が最低でも 1 つ必要です。

表5.3 ルーター暗号プロファイル

プロファイル	互換性のある最も古いクライアント
modern	Firefox 27、Chrome 30、IE 11 on Windows 7、Edge、Opera 17、Safari 9、Android 5.0、Java 8
intermediate	Firefox 1、Chrome 1、IE 7、Opera 5、Safari 1、Windows XP IE8、Android 2.3、Java 7
old	Windows XP IE6、Java 6

詳細は、[Security/Server Side TLS](#) リファレンスガイドを参照してください。

ルーターは **intermediate** プロファイルにデフォルト設定されています。ルートを作成する時または、既存のルーターの **ROUTER_CIPHERS** を **modern**、**intermediate** または **old** に変更する時

に、**--ciphers** オプションを使用して別のプロジェクトを選択します。または、":" 区切りで暗号化を指定することも可能です。暗号化は、以下のコマンドで表示されたセットの中から選択する必要があります。

```
openssl ciphers
```

5.7.9. ルートホスト名

OpenShift Container Platform ルートを使用してサービスと外部に到達可能なホスト名を関連付けることで、サービスを外部に公開することができます。このエッジホスト名は次に、サービスにトラフィックをルーティングするのに使用します。

異なる namespace から複数のルートが同じホストを要求する場合に、一番古いルートが優先され、その namespace にホストを獲得します。同じ namespace 内に、追加のルートが異なるパスフィールドで定義されている場合には、これらのパスが追加されます。複数のルートに同じパスが使用されている場合には、一番古いものが優先されます。

あるユーザーがホスト名にルート 2 つ (1 つが新しく、1 が古い) を指定しようとしていると仮定します。このユーザーがホスト名に他の 2 つのルートを指定する前に、別のユーザーが同じホスト名にルートを指定したうえに、元のユーザーにより作成済みのルートが削除された場合に、このホスト名への要求は効果がなくなります。他の namespace がこのホスト名を要求し、最初のユーザーの要求はなくなります。

例5.1 指定されたホストを持つルート:

```
apiVersion: v1
kind: Route
metadata:
  name: host-route
spec:
  host: www.example.com ①
  to:
    kind: Service
    name: service-name
```

- ① サービスを公開するために使用される外部から到達可能なホスト名を指定します。

例5.2 ホスト内のルート:

```
apiVersion: v1
kind: Route
metadata:
  name: no-route-hostname
spec:
  to:
    kind: Service
    name: service-name
```

ホスト名がルート定義の一部として指定されていない場合には、OpenShift Container Platform が自動的に生成します。生成されたホスト名は以下のような形式をとります。

```
<route-name>[-<namespace>].<suffix>
```

以下の例では、namespace **mynamespace** にホストを追加せずに、上記のルート設定に対して OpenShift Container Platform が生成したホスト名を示します。

例5.3 生成されるホスト名

```
no-route-hostname-mynamespace.router.default.svc.cluster.local 1
```

- 1** 生成されたホスト名のサフィックスは、デフォルトのルーティングサブドメイン **router.default.svc.cluster.local** です。

クラスター管理者は、環境に合わせて「[デフォルトのルーティングサブドメインとして使用するサフィックスをカスタマイズ](#)」することもできます。

5.7.10. ルートタイプ

ルートにセキュリティーを設定してもしなくても構いません。セキュアなルートは、複数の TLS 終端タイプを使用してクライアントに証明書を提供できます。ルーターは、[edge](#)、[passthrough](#) および [re-encryption](#) 終端をサポートします。

例5.4 セキュリティー保護されていないルートオブジェクト YAML 定義

```
apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name
```

セキュリティー保護されていないルートは、鍵や証明書が必要でないので、設定が最も単純ですが、セキュリティー保護されているルートは、接続を非公開のままにできるのでセキュリティーを確保できます。

Secured ルートは、ルートの TLS 終端が指定されたルートです。利用可能な終端タイプは、[以下で説明されています](#)。

5.7.10.1. パススペースのルート

パススペースのルートは、同じホスト名で、それぞれ異なるパスを使用して複数のルートにサービスを提供できるように、URL と比較可能なパスコンポーネントを指定します (ルートのトラフィックが HTTP ベースでなければならない)。ルーターは、最も限定的なものから順に、ルートを照合する必要がありますが、これはルーターの実装により左右されます。ホスト名とパスは、正常に要求に応答できるように、バックエンドサーバーにパススルーされます。たとえば、<http://example.com/foo/> への要求がルーターに移動すると、Pod に <http://example.com/foo/> への要求が表示されます。

以下の表は、ルートのサンプルおよびそれらのアクセシビリティを示しています。

表5.4 ルートの可用性

ルート	比較対象	アクセス可能
www.example.com/test	www.example.com/test	Yes
	www.example.com	No
www.example.com/test and www.example.com	www.example.com/test	Yes
	www.example.com	Yes
www.example.com	www.example.com/test	はい (ルートではなく、ホストにマッチ)
	www.example.com	Yes

例5.5 パスが 1 つでセキュリティー保護されていないルート

```

apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" ①
  to:
    kind: Service
    name: service-name

```

① パスは、パスベースのルートに唯一追加される属性です。

注記

ルーターは TLS を終了させず、要求のコンテンツを読み込みことができないので、パスベースのルーティングは、パススルー TLS を使用する場合には利用できません。

5.7.10.2. セキュリティー保護されたルート

セキュリティー保護されたルートは、ルートの TLS 終端を指定し、オプションで鍵と証明書を提供します。

注記

OpenShift Container Platform の TLS 終端は、カスタム証明書を提供する [SNI](#) に依存します。ポート 443 で受信する SNI 以外のトラフィックは、TLS 終端およびデフォルトの証明書で処理されます (要求のホスト名と一致せず、バリデーションエラーが発生する可能性があります)。

セキュリティー保護されたルートは、以下の 3 種類のセキュアな TLS 終端を使用できます。

Edge Termination

edge termination では、TLS 終端は、宛先にトラフィックをプロキシ化する前に、ルーターで発生します。TLS 証明書は、ルーターのフロントエンドで提供されるので、ルートに設定する必要があります。設定されていない場合には、「[ルーターのデフォルトの証明書](#)」が TLS 終端に使用されます。

例5.6 Edge Termination を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: edge ❸
    key: |- ❹
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |- ❺
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |- ❻
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

❶ ❷ オブジェクトの名前で、63 文字に制限されます。

❸ **termination** フィールドは edge termination の **edge** です。

❹ **key** フィールドは PEM 形式のキーファイルのコンテンツです。

❺ **certificate** フィールドは PEM 形式の証明書ファイルのコンテンツです。

❻ オプションの CA 証明書は、検証用に証明書チェーンを確立するために必要になる場合があります。

TLS がルーターで終端されるので、内部ネットワークを使用したルーターからエンドポイントへの接続は暗号化されません。

Edge termination ルートは **insecureEdgeTerminationPolicy** を指定して、セキュアでないスキーム (HTTP) 上にあるトラフィックを無効化、許可、リダイレクトすることができます。**insecureEdgeTerminationPolicy** で使用できる値は **None** または空 (無効化する場合)、**Allow** または **Redirect** です。デフォルトの **insecureEdgeTerminationPolicy** は、セキュ

アでないスキーム上のトラフィックを無効にします。一般的なユースケースは、セキュアなスキームを使用してコンテンツを、セキュアでないスキームを使用してアセット (例のイメージ、スタイルシート、javascript) を提供できるようにします。

例5.7 Edge Termination を使用したセキュリティー保護されたルートでの HTTP トラフィックの許可

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured-allow-insecure ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: edge ❸
    insecureEdgeTerminationPolicy: Allow ❹
    [ ... ]
```

❶ ❷ オブジェクトの名前で、63 文字に制限されます。

❸ **termination** フィールドは edge termination の **edge** です。

❹ セキュアでないスキーム **HTTP** で送信される要求を許可するセキュアでないポリシー

例5.8 Edge Termination を使用したセキュリティー保護されたルートでの HTTP トラフィックのリダイレクト

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured-redirect-insecure ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: edge ❸
    insecureEdgeTerminationPolicy: Redirect ❹
    [ ... ]
```

❶ ❷ オブジェクトの名前で、63 文字に制限されます。

❸ **termination** フィールドは edge termination の **edge** です。

❹ セキュアでないスキーム **HTTP** で送信される要求をセキュアなスキーム **HTTPS** にリダイレクトするセキュアでないポリシー

Passthrough Termination

passthrough termination では、暗号化されたトラフィックが TLS 終端を提供するルーターなしに宛先に直接送信されます。そのため、鍵や証明書は必要ありません。

例5.9 Passthrough Termination を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-passthrough-secured ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: passthrough ❸
```

❶❷ オブジェクトの名前で、63 文字に制限されます。

❸ **termination** フィールド **passthrough** に設定します。他の暗号化フィールドは必要ありません。

宛先 Pod は、エンドポイントでトラフィックに証明書を提供します。これは、必須となるクライアント証明書をサポートするための唯一の方法です (相互認証とも呼ばれる)。



注記

Passthrough ルートには **insecureEdgeTerminationPolicy** を指定できます。唯一有効な値は **None** (無効化する場合は空) または **Redirect** です。

Re-encryption Termination

Re-encryption は、edge termination の一種で、ルーターが証明書を使用して TLS を終端し、異なる証明書が設定されている可能性のあるエンドポイントへの接続を再暗号化します。そのため、内部ネットワークなどを含め、接続の全パスが暗号化されています。ルーターは、ヘルスチェックを使用して、ホストの信頼性を判断します。

例5.10 Re-Encrypt Termination を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-pt-secured ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: reencrypt ❸
```

```
key: [as in edge termination]
certificate: [as in edge termination]
caCertificate: [as in edge termination]
destinationCACertificate: |- ❹
  -----BEGIN CERTIFICATE-----
  [...]
  -----END CERTIFICATE-----
```

❶❷ オブジェクトの名前で、63 文字に制限されます。

❸ **termination** フィールドは **reencrypt** に設定されます。他のフィールドは **edge termination** の場合と同じです。

❹ re-encryption には必須です。**destinationCACertificate** は、エンドポイント証明書を検証する CA 証明書を指定して、ルーターから宛先 Pod への接続のセキュリティを確保します。サービスがサービス署名証明書を使用する場合または、管理者がデフォルトの CA 証明書をルーターに指定し、サービスにその CA により署名された証明書がある場合には、このフィールドは省略可能です。

destinationCACertificate フィールドが空の場合は、ルーターは自動的に証明書を提供するサービス用に生成される証明書を自動的に活用し、すべての Pod に **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt** として注入します。これにより、ルートの証明書を生成する必要なしに、新しいルートがエンドツーエンドの暗号化を活用できるようになります。これは、管理者が許可しない限り、**destinationCACertificate** が使用できない、カスタムのルーターまたは F5 ルーターの場合に有用です。



注記

Re-encrypt ルートでは **insecureEdgeTerminationPolicy** に、edge termination ルートと同じ値にすべて指定することができます。

5.7.11. ルーターのシャード化

OpenShift Container Platform では、各ルートは **metadata** フィールドにいくつでも「ラベル」を指定できます。ルーターは **セクター (選択式)** (とも呼ばれる) を使用して、サービスを提供するルート of 全プールからルートのサブセットを選択します。選択式でも、ルートの namespace でラベルを使用できます。選択したルートは、**ルーターのシャード** を形成します。ルートと分けて、ルーターシャードだけを「作成」、「変更」できます。

この設計では、**従来の** シャード化も、**重複** シャード化をサポートします。従来のシャード化では、選択した内容が重複セットにならず、ルートはシャード 1 つのみに所属します。重複シャード化では、選択した内容は重複セットに含まれ、ルートは多数の異なるシャードに所属できます。たとえば、あるルートは **SLA=high** シャード (**SLA=medium** または **SLA=low** シャードではない) や **geo=west** シャード (**geo=east** シャードではない) に所属することができます。

重複シャード化の他の例には、ルートの namespace をベースに選択するルーターセットなどがあります。

ルーター	選択	Namespace
router-1	A* — J*	A*, B*, C*, D*, E*, F*, G*, H*, I*, J*

ルーター	選択	Namespace
router-2	K* — T*	K*, L*, M*, N*, O*, P*, Q*, R*, S*, T*
router-3	Q* — Z*	Q*, R*, S*, T*, U*, V*, W*, X*, Y*, Z*

router-2 および **router-3** は、namespaces **Q***、**R***、**S***、**T*** のルートにサービスを提供します。この例を重複から従来のシャード化に変更するには、**router-2** の選択肢を **K* — P*** に変更して、重複をなくすことができます。

ルートがシャード化されている場合には、指定のルートはこのグループのルーター 0 個以上にバインドされます。ルートをバインドすることで、シャード全体でルートを一意に保つことができます。一意に保つことで、単一のシャード内に、同じルートでもセキュアなバージョンと、セキュアでないバージョンを存在させることができます。つまり、ルートは、作成、バインド、アクティブ化のライフサイクルが可視化されたことになります。

シャード化の環境では、シャードに到達する最初のルートが再起動の有無に拘わらず、期限なしに存在できる権利を持ちます。

green/blue デプロイメント時には、ルートは複数のルーターに選択される場合があります。OpenShift Container Platform のアプリケーション管理者は、別のバージョンのアプリケーションにトラフィックをフラッシュして、以前のバージョンをオフに設定する場合があります。

シャードリングは、管理者によりクラスターレベルで、ユーザーにより、プロジェクト/namespace レベルで実行できます。namespace ラベルを使用する場合には、ルーターのサービスアカウントには、**cluster-reader** パーミッションを設定して、ルーターが namespace 内のラベルにアクセスできるようにします。



注記

同じホスト名を要求するルートが 2 つ以上ある場合には、解決する順番は、ルートの存在期間をもとにし、一番古いルートがホストの要求を優先的に取得します。シャード化されたルーターの場合には、ルートは、ルーターの選択基準にあったラベルをベースに選択されます。ラベルがルートに追加されるタイミングを判断する方法に一貫性はありません。そのため、既存のホスト名を要求する以前のルートが「再度ラベル化されて」、ルーターの選択基準と照合させる場合には、上述の解決順に基づき既存のルートを置き換えます (最も古いルートが優先される)。

5.7.12. 他のバックエンドおよび重み

ルートは通常、**kind: Service** の **to:** トークンを使用したサービスと関連付けられます。ルートへの全要求は、「[負荷分散ストラテジー](#)」をベースに、サービス内のエンドポイントにより処理されます。

サービスは最大 4 つまでルートをサポートすることができます。各サービスが処理する要求の大きさは、サービスの **weight** により統制されます。

最初のサービスは、以前と同様に **to:** トークンを使用して入り、サービスは 3 つまで **alternateBackend:** トークンを使用して入ることができます。各サービスは、デフォルトの **kind: Service** が指定されている必要があります。

各サービスには、**weight** が関連付けられています。サービスが処理する要求の大きさは、**weight / sum_of_all_weights** で算出されます。サービスにエンドポイントが複数ある場合には、サービスの

加重が 1 以上、各エンドポイントに割り当てられるように、エンドポイント全体に分散されます。サービスの **weight** が 0 の場合は、サービスの各エンドポイントには 0 が割り当てられます。

weight は、0-256 の範囲内で指定してください。デフォルトは 1 です。**weight** が 0 の場合は、サービスに要求は渡されません。全サービスの **weight** が 0 の場合は、要求に対して 503 エラーが返されます。サービスにエンドポイントがない場合には、加重は実際には 0 となります。

alternateBackends を使用すると、**roundrobin** 負荷分散ストラテジーを使用して、要求が想定どおりに **weight** をもとにサービスに分散されるようになります。ルートに **roundrobin** を設定する場合は、「[ルートアノテーション](#)」を使用するか、一般的なルーターには環境変数を使用します。

以下は、「[A/B デプロイメント](#)」向けに別のバックエンドを使用したルート設定例です。

alternateBackends および加重が指定されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-alternate-service
  annotations:
    haproxy.router.openshift.io/balance: roundrobin ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
    weight: 20 ❸
  alternateBackends:
  - kind: Service
    name: service-name2 ❹
    weight: 10 ❺
    kind: Service
    name: service-name3 ❻
    weight: 10 ❼
```

❶ このルートは**roundrobin**「[負荷分散ストラテジー](#)」を使用します。

❷ 最初のサービス名は **service-name** であり、0 以上の Pod が含まれる可能性があります。

❹❻ **alternateBackend** サービスにも 0 以上の Pod が含まれる場合があります。

❸❺❷ **weight** の合計は 40 で、**service-name** は要求の 20/40 または 1/2 を、**service-name2** と **service-name3** はそれぞれ、要求の 1/4 を取得し、サービスごとに 1 または複数のエンドポイントが含まれると想定します。

5.7.13. ルート固有のアノテーション

環境変数を使用して、ルーターは、公開する全ルートにデフォルトオプションを設定できます。個別のルートは、アノテーションに個別の設定を指定して、デフォルトの一部を上書きできます。

ルートアノテーション

このセクションで説明されているすべての項目に対して、ルートがその設定を変更できるように **ルート定義** にアノテーションを設定できます。

表5.5 ルートアノテーション

変数	説明	デフォルトで使用する環境変数
<code>haproxy.router.openshift.io/balance</code>	負荷分散アルゴリズムを設定します。使用できるオプションは source 、 roundrobin 、および leastconn です。	<code>passthrough</code> ルートの ROUTER_TCP_BALANCE_SCHEME です。それ以外の場合は ROUTER_LOAD_BALANCE_ALGORITHM を使用します。
<code>haproxy.router.openshift.io/disable_cookies</code>	関連の接続を追跡する cookie の使用を無効にします。 true または TRUE に設定する場合は、分散アルゴリズムを使用して、受信する HTTP 要求ごとに、どのバックエンドが接続を提供するかを選択します。	
<code>haproxy.router.openshift.io/cookie_name</code>	このルートに使用するオプションの cookie を指定します。名前は、大文字、小文字、数字、" <code>_</code> " または " <code>-</code> " を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。	
<code>haproxy.router.openshift.io/rate-limit-connections</code>	レート制限機能を有効にするために true または TRUE を設定します。	
<code>haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp</code>	IP アドレスで共有される同時 TCP 接続の数を制限します。	
<code>haproxy.router.openshift.io/rate-limit-connections.rate-http</code>	IP アドレスが HTTP 要求を実行できるレートを制限します。	
<code>haproxy.router.openshift.io/rate-limit-connections.rate-tcp</code>	IP アドレスが TCP 接続を行うレートを制限します。	
<code>haproxy.router.openshift.io/timeout</code>	ルートのサーバー側のタイムアウトを設定します。(TimeUnits)	ROUTER_DEFAULT_SERVER_TIMEOUT
<code>router.openshift.io/haproxy.health.check.interval</code>	バックエンドのヘルスチェックの間隔を設定します。(TimeUnits)	ROUTER_BACKEND_CHECK_INTERVAL
<code>haproxy.router.openshift.io/ip_whitelist</code>	ルートの「 ホワイトリスト 」を設定します。	

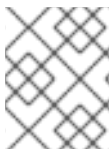
変数	説明	デフォルトで使用する環境変数
haproxy.router.openshift.io/hsts_header	edge terminated または re-encrypt ルートの Strick-Transport-Security ヘッダーを設定します。	

例5.11 ルート設定のカスタムタイムアウト

```

apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms ❶
[...]
```

- ❶ HAProxy 対応の単位 (us、ms、s、m、h、d) で新規のタイムアウトを指定します。単位が指定されていない場合は、ms がデフォルトになります。



注記

passthrough ルートのサーバー側のタイムアウトを低く設定し過ぎると、WebSocket 接続がそのルートで頻繁にタイムアウトする可能性があります。

5.7.14. ルート固有の IP ホワイトリスト

選択した IP アドレスだけにルートへのアクセスを制限するには、ルートに **haproxy.router.openshift.io/ip_whitelist** アノテーションを追加します。ホワイトリストは、承認したソースアドレスの IP アドレスまたは/および CIDR をスペース区切りにします。ホワイトリストに含まれていない IP アドレスからの要求は破棄されます。

例:

ルートの編集時に、以下のアノテーションを追加して必要なソース IP を定義します。または、**oc annotate route <name>** を使用します。

唯一の特定の IP アドレスのみを許可します。

```

metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

複数の IP アドレスを許可します。

```

metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11
    192.168.1.12
```

IP CIDR ネットワークを許可します。


```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

混在した IP アドレスおよび IP CIDR ネットワークを許可します。

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24
    10.0.0.0/8
```

5.7.15. ワイルドカードサブドメインポリシーを指定するルートの作成

ワイルドカードポリシーでは、(許可できるようにルーターを設定する場合) ドメイン内の全ホストに対応するルートを定義できます。ルートは、**wildcardPolicy** フィールドを使用して、設定の一部としてワイルドカードポリシーを指定できます。ワイルドカードルートを許可するポリシーが指定されたルーターは、ワイルドカードポリシーをもとに適切にルートを公開します。

「ワイルドカードルートを許可するように HAProxy ルートを設定する方法についてはこちら」を参照してください。

例5.12 サブドメインワイルドカードポリシーを指定するルート

```
apiVersion: v1
kind: Route
spec:
  host: wildcard.example.com ①
  wildcardPolicy: Subdomain ②
  to:
    kind: Service
    name: service-name
```

- ① サービスを公開するために使用される外部から到達可能なホスト名を指定します。
- ② 外部から到達可能なホスト名は、サブドメイン **example.com** 内の全ホストを許可するように指定します。***.example.com** は、公開されたサービスに到達するためのホスト名 **wildcard.example.com** のサブドメインです。

5.7.16. ルートステータス

route status フィールドは、ルーターでのみ設定されます。ルーターが特定のルーターへのサービス提供を停止するように、ルートに変更が加えられた場合には、ステータスが古くなってしまいますが、ルーターは、**route status** フィールドは消去しません。ルートステータスの以前のエントリーを削除するには、[clear-route-status script](#) を使用してください。

5.7.17. ルート内の特定ドメインの拒否または許可

ルーターは、**ROUTER_DENIED_DOMAINS** および **ROUTER_ALLOWED_DOMAINS** 環境変数を使用して、ルート内のホスト名からのドメインサブセットを限定して拒否または許可するように設定できます。

ROUTER_DENIED_DOMAINS	一覧表示されるドメインは指定のルートで許可されません。
ROUTER_ALLOWED_DOMAINS	一覧表示されるドメインのみが指定のルートで許可されます。

拒否ドメインの一覧に含まれるドメインは、許可ドメイン一覧よりも優先されます。つまり、OpenShift Container Platform は先に、拒否リスト (該当する場合) をチェックして、ホスト名が拒否ドメイン一覧に含まれていない場合に、許可ドメインをチェックします。ただし、許可ドメインの一覧はより制限されており、ルーターは、その一覧に所属するホストが含まれるルートのみを許可します。

たとえば、**myrouter** ルートの **[*.]open.header.test**、**[*.]openshift.org** および **[*.]block.it** ルートを拒否するには、以下を実行します。

```
$ oc adm router myrouter ...
$ oc set env dc/myrouter ROUTER_DENIED_DOMAINS="open.header.test,
openshift.org, block.it"
```

これは、**myrouter** がルートの名前に基づいて以下を許可することを意味します。

```
$ oc expose service/<name> --hostname="foo.header.test"
$ oc expose service/<name> --hostname="www.allow.it"
$ oc expose service/<name> --hostname="www.openshift.test"
```

ただし、**myrouter** は以下を拒否します。

```
$ oc expose service/<name> --hostname="open.header.test"
$ oc expose service/<name> --hostname="www.open.header.test"
$ oc expose service/<name> --hostname="block.it"
$ oc expose service/<name> --hostname="franco.baresi.block.it"
$ oc expose service/<name> --hostname="openshift.org"
$ oc expose service/<name> --hostname="api.openshift.org"
```

または、ホスト名が **[*.]stickshift.org** または **[*.]kates.net** に設定されていない ルートをブロックするには、以下を実行します。

```
$ oc adm router myrouter ...
$ oc set env dc/myrouter ROUTER_ALLOWED_DOMAINS="stickshift.org,
kates.net"
```

これは、**myrouter** ルートが以下を許可することを意味します。

```
$ oc expose service/<name> --hostname="stickshift.org"
$ oc expose service/<name> --hostname="www.stickshift.org"
$ oc expose service/<name> --hostname="kates.net"
$ oc expose service/<name> --hostname="api.kates.net"
$ oc expose service/<name> --hostname="erno.r.kube.kates.net"
```

ただし、**myrouter** は以下を拒否します。

```
$ oc expose service/<name> --hostname="www.open.header.test"
```

```
$ oc expose service/<name> --hostname="drive.ottomatic.org"
$ oc expose service/<name> --hostname="www.wayless.com"
$ oc expose service/<name> --hostname="www.deny.it"
```

両方のシナリオを実装するには、以下を実行します。

```
$ oc adm router adrouter ...
$ oc env dc/adrouter ROUTER_ALLOWED_DOMAINS="openshift.org, kates.net" \
  ROUTER_DENIED_DOMAINS="ops.openshift.org, metrics.kates.net"
```

これにより、ホスト名が **[*.]openshift.org** または **[*.]kates.net** に設定されているルートを許可し、ホスト名が **[*.]ops.openshift.org** または **[*.]metrics.kates.net** に設定されているルートは拒否します。

そのため、以下は拒否されます。

```
$ oc expose service/<name> --hostname="www.open.header.test"
$ oc expose service/<name> --hostname="ops.openshift.org"
$ oc expose service/<name> --hostname="log.ops.openshift.org"
$ oc expose service/<name> --hostname="www.block.it"
$ oc expose service/<name> --hostname="metrics.kates.net"
$ oc expose service/<name> --hostname="int.metrics.kates.net"
```

しかし、以下は許可されます。

```
$ oc expose service/<name> --hostname="openshift.org"
$ oc expose service/<name> --hostname="api.openshift.org"
$ oc expose service/<name> --hostname="m.api.openshift.org"
$ oc expose service/<name> --hostname="kates.net"
$ oc expose service/<name> --hostname="api.kates.net"
```

5.7.18. Namespace 所有権チェックの無効化

ホストとサブドメインは、最初に要求を作成したルートの namespace が所有します。その namespace で作成された他のルートは、サブドメイン上で要求を作成できます。他の namespace はすべて、請求済みのホストおよびサブドメインに要求を作成することはできません。ホストを所有する namespace は、**www.abc.xyz/path1** など、そのホストに関連付けられている全パスも所有します。

たとえば、ホスト **www.abc.xyz** がどのルートからも要求されていない場合に、**ns1** namespace にホストが **www.abc.xyz** のルート **r1** を作成すると、**ns1** の namespace が、ワイルドカードルート **www.abc.xyz** とサブドメイン **abc.xyz** を所有します。別の namespace **ns2** が異なるパス **www.abc.xyz/path1/path2** で、ルートを作成しようとする、別の namespace (今回は **ns1**) のルートがこのホストを所有するので失敗してしまいます。

「ワイルドカードルート」では、サブドメインを所有する namespace はそのサブドメインに含まれるホストすべてを所有します。上記の例のように、namespace が **abc.xyz** というサブドメインを所有する場合には、別の namespace は **z.abc.xyz** を要求できません。

namespace の所有権ルールを無効にするには、これらの制限を無効にして、namespace すべてでホスト (およびサブドメイン) を要求できるようにします。



警告

ルーターで namespace の所有権チェックを無効にする場合には、エンドユーザーが namespace に含まれるホストの所有権を要求できるようになる点に注意してください。この設定の変更は、特定の開発環境で価値がありますが、実稼働環境では慎重にこの機能を使用し、クラスターポリシーで、信頼されないエンドユーザーがルートを作成できないようにロックしてください。

たとえば、**ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true** では、namespace **ns1** が最も古い **r1 www.abc.xyz** を作成する場合に、このホスト名 (+ パス) のみを所有します。別の namespace は、このサブドメイン (**abc.xyz**) に最も古いルートがなくても、ワイルドカードルートを作成することができ、別の namespace が (**foo.abc.xyz**、**bar.abc.xyz**、**baz.abc.xyz** など) ワイルドカード以外の重複ホストを要求することも可能で、この要求は認められます。

別の namespace (例: **ns2**) はルート **r2 www.abc.xyz/p1/p2** を作成でき、許可されます。同様に、別の namespace (**ns3**) は、サブドメインのワイルドカードポリシーでルート **wildthing.abc.xyz** も作成でき、このワイルドカードを所有できます。

この例にあるように、ポリシー **ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true** は制約がゆるく、namespace 全体の要求を許可します。ルーターが namespace の所有を無効にしているルートを許可できるのは、ホストとパスがすでに請求済みである場合だけです。

たとえば、新規ルート **rx** が **www.abc.xyz/p1/p2** を要求しようとする場合に、ルート **r2** はホストとパスの組み合わせを所有しているので拒否されます。まったく同じホストとパスがすでに請求済みなので、これは、ルート **rx** が同じ namespace にある場合も、別の namespace にある場合でも同じです。

この機能は、ルーターの作成時か、またはルーターのデプロイメント設定に環境変数を設定して設定できます。

```
$ oc adm router ... --disable-namespace-ownership-check=true
```

```
$ oc env dc/router ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true
```

[1] これ以降は、デバイス名は、コンテナ B のホストのデバイスを参照します。

第6章 サービスカタログコンポーネント

6.1. サービスカタログ

6.1.1. 概要

マイクロサービスベースのアプリケーションを開発して、クラウドネイティブのプラットフォームで実行する場合には、サービスプロバイダーやプラットフォームに合わせて、異なるリソースをプロビジョニングして、その従属、認証情報、設定を共有する方法は多数あります。

開発者がよりシームレスに作業できるように、OpenShift Container Platform には Kubernetes 向けの [Open Service Broker API \(OSB API\)](#) の実装である **サービスカタログ** が含まれています。これにより、OpenShift Container Platform にデプロイされているアプリケーションを、さまざまな種類のサービスブローカーに接続できるようになります。

サービスカタログでは、クラスター管理者が 1 つの API 仕様を使用して、複数のプラットフォームを統合できます。OpenShift Container Platform web コンソールは、サービスカタログで提供されるサービスブローカーが提供するクラスターサービスカタログを表示するので、これらのサービスを検出、インスタンス化して、アプリケーションで使用できるようにします。

このように、サービスユーザーは異なるプロバイダーが提供する異なるタイプのサービスを簡単かつ一貫性を保ちながら使用できるという利点が得られます。また、サービスプロバイダーは、複数のプラットフォームにアクセスできる統合ポイントから利点を得られます。

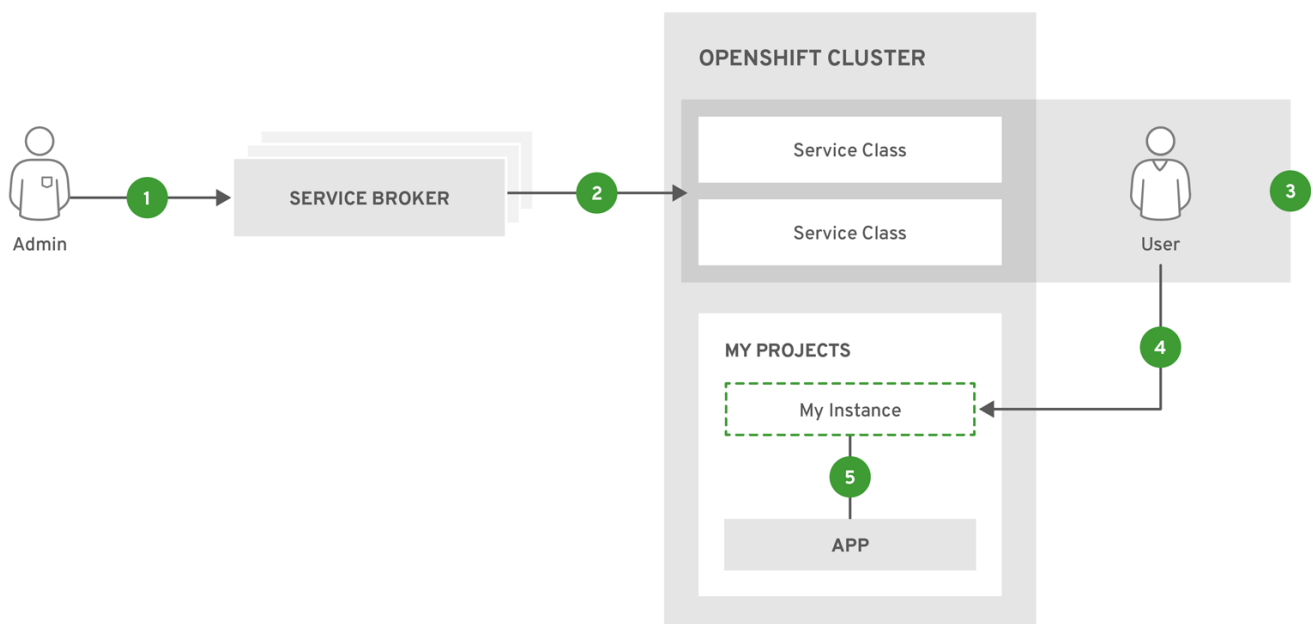
6.1.2. 設計

サービスカタログの設計は基本的なワークフローに基づいています。



注記

以下の新規の用語は「[概念および用語](#)」でさらに定義されています。



OPENSHIFT_415489_0218

クラスター管理者は、1 つまたは複数の **クラスターサービスブローカー** を OpenShift Container Platform クラスターに登録します。これは、デフォルトで提供されているサービスブローカーでインストール時に自動的に行うことも、手動で行うことも可能です。

各サービスブローカーは、ユーザーに提供すべき **クラスターサービスクラス セット**と、これらのサービスのバリエーション (**サービスプラン**) を、OpenShift Container Platform に指定します。

OpenShift Container Platform web コンソールまたは CLI を使用して、ユーザーは利用可能なサービスを検出します。たとえば、クラスターサービスクラスは、BestDataBase と呼ばれる database-as-a-service などの、サービスクラスが利用できる可能性があります。

ユーザーは、クラスターサービスクラスを選択して、自身の新しい **インスタンス** を要求します。たとえば、サービスは **my_db** という名前の BestDataBase インスタンスなどです。

ユーザーは、サービスを Pod セット (アプリケーション) にリンクまたは **バインド** します。たとえば、**my_db** サービスインスタンスは、**my_app** と呼ばれるユーザーアプリケーションにバインドできます。

ユーザーがリソースのプロビジョニングおよびプロビジョニング解除を要求すると、この要求はサービスカタログに送信され、次に適切なクラスターサービスブローカーに要求が送信されます。サービスによっては、**provision**、**deprovision** および **update** などの操作に時間がかかる可能性があります。クラスターサービスブローカーが利用できない場合には、サービスカタログはこの操作の再試行を続けます。

このインフラストラクチャーでは、OpenShift Container Platform で実行中のアプリケーションと、それが使用するサービスの間で疎結合することができます。こうすることで、これらのサービスを使用するアプリケーションが独自のビジネスロジックにフォーカスし、サービスの管理をプロバイダーに任せることができます。

6.1.2.1. リソースの定義

ユーザーがサービスを使用し終えた場合 (または、請求しなくてもいい場合) には、このサービスインスタンスは削除できます。サービスインスタンスを削除するには、サービスのバインドを先に削除する必要があります。サービスのバインドの削除は、**バインド解除** と呼ばれます。削除のプロセスで、削除予定のサービスバインディングを参照するシークレットも削除します。

サービスバインディングが削除されると、サービスインスタンスを削除できあす。サービスインスタンスの削除は **プロビジョニング解除** として知られています。

サービスバインディングやサービスインスタンスが含まれるプロジェクトや namespace を削除する場合は、サービスカタログは先にクラスターサービスブローカーに、関連のインスタンスとバインディングを削除するように要求する必要があります。これにより、サービスカタログは、クラスターのサービスブローカーと通信して、プロビジョニング解除を行うまで待つ必要があるため、実際のプロジェクトや namespace の削除が遅延してしまうことが予想されます。通常の場合では、サービスにより異なりますが、数分以上かかる場合があります。



注記

デプロイメントで使用するサービスバインディングを削除する場合、デプロイメントからバインディングの参照を削除する必要があります。そうしないと、次のロールアウトは失敗します。

6.1.3. 概念および用語

クラスターサービスブローカー

クラスターサービスブローカー は、OSB API 仕様に準拠し、1 つ以上のサービスのセットを管理するサーバーです。このソフトウェアは、独自の OpenShift Container Platform クラスター内またはその他の場所でホストされる可能性があります。

クラスター管理者は、クラスターサービスブローカーを表す **ClusterServiceBroker** API リソースを作成して、OpenShift Container Platform クラスターに登録できます。これにより、クラスター管理者はクラスター内で利用可能なクラスターサービスブローカーを使用して新しい種類の管理済みサービスを作成できるようになります。

ClusterServiceBroker リソースは、クラスターサービスブローカーの接続詳細と、サービスセット（およびこれらのサービスのバリエーション）を OpenShift Container Platform に指定すると、ユーザーに提供できるはずです。**authInfo** セクションには、クラスターサービスブローカーの認証に使用するデータが含まれている点に特に注意してください。

ClusterServiceBroker リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ClusterServiceBroker
metadata:
  name: BestCompanySaaS
spec:
  url: http://bestdatabase.example.com
  authInfo:
    basic:
      secretRef:
        namespace: test-ns
        name: secret-name
```

クラスターサービスクラス

また、サービスカタログのコンテキストで「サービス」と類義の **クラスターサービスクラス** は、特定のクラスターサービスブローカーが提供する管理サービスの 1 種です。新しいクラスターサービスブローカーのリソースがクラスターに追加されるたびに、サービスカタログコントローラーは、適切なクラスターサービスブローカーに接続し、サービスオファリングの一覧を取得します。新しい **ClusterServiceClass** リソースは自動的に、クラスターサービスブローカーごとに作成されます。



注記

さらに、OpenShift Container Platform には、「サービス」と呼ばれるコアとなるコンセプトがあります。サービスとは、内部の負荷分散に関連する別個の Kubernetes リソースです。これらのリソースは、サービスカタログや OSB API のコンテキストで使用する用語と混同しないようにしてください。

ClusterServiceClass リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ClusterServiceClass
metadata:
  name: smallDB
  brokerName: BestDataBase
plans: [...]
```

クラスターサービス計画

クラスターサービスプラン は、クラスターサービスクラスの階層を指します。たとえば、クラスターサービスクラスは、さまざまなレベルの quality-of-service (QoS) を提供するプランを公開し、それぞれに異なるコストが関連付けられています。

サービスインスタンス

サービスインスタンス は、プロビジョニングされたクラスターサービスクラスのインスタンスです。サービスクラスが提供する機能を使用する場合には、新しいサービスインスタンスを作成してください。

新規の **ServiceInstance** リソースを作成した場合には、サービスカタログコントローラーは、適切なクラスターサービスブローカーと接続して、サービスインスタンスをプロビジョニングするように指示を出します。

ServiceInstance リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceInstance
metadata:
  name: my_db
  namespace: test-ns
spec:
  externalClusterServiceClassName: smallDB
  externalClusterServicePlanName: default
```

アプリケーション

アプリケーション という用語は、**サービスインスタンス** を使用する ユーザーのプロジェクトで実行中の Pod など、OpenShift Container Platform デプロイメントのアーティファクトを指します。

認証情報

認証情報 とは、サービスインスタンスと通信するアプリケーションに必要な情報です。

サービスバインディング

サービスバインディング は、サービスインスタンスとアプリケーションの間のリンクを指します。サービスバインディングは、アプリケーションからサービスインスタンスを参照して使用できるように希望するクラスターユーザーが作成します。

サービスバインディングの作成時に、サービスカタログコントローラーはサービスインスタンスの接続情報と認証情報を含む Kubernetes のシークレットを作成します。これらのシークレットは通常どおり Pod にマウントできます。また、**PodPresets** とも統合され、これでシークレットの使用方法や使用する Pod などを表現できます。

ServiceBinding リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: myBinding
  namespace: test-ns
spec:
  instanceRef:
    name: my_db
  parameters:
    securityLevel: confidential
    secretName: mySecret
```

パラメーター

パラメーターは、サービスバインディングまたはサービスインスタンスの使用時に、追加のデータをクラスターサービスブローカーに渡すために提供されている特別なフィールドです。唯一のフォーマット要件は、パラメーターを有効な YAML (または JSON) で指定することです。上記の例では、セキュリティーレベルのパラメーターをサービスバインディング要求でクラスターサービスブローカーに渡します。より高度なセキュリティーが必要なパラメーターの場合は、パラメーターをシークレットに配置し、**parametersFrom** を使用して参照します。

シークレットを参照するサービスバインディングリソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: myBinding
  namespace: test-ns
spec:
  instanceRef:
    name: my_db
  parametersFrom:
    - secretKeyRef:
        name: securityLevel
        key: myKey
  secretName: mySecret
```

6.1.4. 提供されるクラスターサービスブローカー

OpenShift Container Platform は、サービスカタログで使用する以下のクラスターサービスブローカーを提供します。

- [Template Service Broker](#)
- [OpenShift Ansible Broker](#)

6.2. テンプレートサービスブローカー

テンプレートサービスブローカー (TSB) により、サービスカタログで OpenShift Container Platform の最初のリリースより同梱されている「[デフォルトのインスタントアプリおよびクイックスタートテンプレート](#)」を表示できるようになります。TSB は、Red Hat、クラスター管理者、ユーザー、サードパーティベンダーの誰が提供したものでも、OpenShift Container Platform「[テンプレート](#)」に書き込まれたものはサービスとして提供できます。

デフォルトでは、TSB は **openshift** プロジェクトからグローバルで入手可能なオブジェクトを表示します。また、クラスター管理者が選択する他のプロジェクトを監視するように設定することも可能です。

6.3. OPENSIFT ANSIBLE BROKER

6.3.1. 概要

OpenShift Ansible Broker (OAB) は、[Ansible playbook bundle \(APB\)](#) で定義されるアプリケーションを管理する Open Service Broker (OSB) API の実装です。APB は、Ansible ランタイムと、コンテナイメージに同梱されている Ansible playbook のバンドルで構成されており、OpenShift

Container Platform のコンテナアプリケーションを定義、配信する新しい方法を提供します。APB は Ansible を活用して、複雑なデプロイするを自動化する標準メカニズムを構築します。

OAB の設計はこの基本的なワークフローに従います。

1. ユーザーは、OpenShift Container Platform Web コンソールを使用してサービスカタログから利用可能なアプリケーションの一覧を要求します。
2. サービスカタログは利用可能なアプリケーションについて OAB に要求します。
3. OAB は定義されたコンテナレジストリーと通信し、利用可能な APB を把握します。
4. ユーザーは特定の APB をプロビジョニングする要求を実行します。
5. プロビジョニング要求は OAB に移動し、APB でプロビジョニングメソッドを呼び出して、ユーザー要求に対応します。

6.3.2. Ansible Playbook Bundle

Ansible Playbook Bundle (APB) は、Ansible ロールおよび Playbook の既存の投資を利用できる軽量のアプリケーション定義です。

APB は、名前の付いた playbook が含まれる単純なディレクトリーを使用し、プロビジョニングやバインドなどの OSB API アクションを実行します。**apb.yml** の仕様ファイルで定義するメタデータには、デプロイメント中に使用する必須/任意のパラメーターの一覧が含まれています。

全体の設計および APB の作成方法についての詳細は、『[APB Development Guide](#)』を参照してください。