Red Hat
OpenShift

Red Hat OpenShift 入門

Red Hat

■本冊子について

本冊子は、『OpenShift徹底入門』（翔泳社）の内容の一部（３部構成の第１部）を著者、出版社の了解のもと抜粋したものです。

書籍『OpenShift徹底入門』

・刊行：翔泳社
・著者：株式会社リアルグローブ・オートメーティッド、青島 秀治、レッドハット株式会社、森 真也、清水 護、織 学、宇都宮 卓也、斎藤 和史、野間 亮志、荒木 俊博
・監修：須江 信洋

https://www.amazon.co.jp/dp/4798172553/

発売日：2022年01月19日
定価：4,180円（本体3,800円＋税10%）
仕様：B5変・648ページ
ISBN 978-4-7981-7255-2

OpenShift（オープンシフト）徹底入門

株式会社リアルグローブ・オートメーティッド 青島 秀治、
レッドハット株式会社 森 真也、清水 護、織 学、宇都宮 卓也、
斎藤 和史、野間 亮志、荒木 俊博 ／著
レッドハット株式会社 須江 信洋 ／監修

SE
SHOEISHA

はじめに

本書は、エンタープライズ向けKubernetesディストリビューションの1つであるRed Hat OpenShift Container Platform（以下、OpenShift）に関する技術解説書です。OpenShiftはCNCF（Cloud Native Computing Foundation）のKubernetes互換認定を取得しており、エンタープライズシステム向けのKubernetesとして利用できます。また、エンタープライズ環境で扱いやすいように機能を追加するなど、様々な工夫を凝らしています。それらの諸機能を理解してうまく使いこなしていくため、本書では実際のユースケースを考慮し、製品ドキュメントとは違った切り口でまとまった情報を提供しています。限られた紙面の中でではありますが、実際の操作画面やコマンドラインの例、YAMLファイルのサンプル、ログの出力例など、現場で役立つ情報を多数掲載しています。

本書は紙幅が限られているため、OpenShiftのコアであるKubernetes自体についての解説は必要最小限にとどめています。Kubernetesの解説書は和書・洋書ともに多数の良書が出版されています。KubernetesおよびLinuxコンテナに関する詳細情報についてはそれらをあわせてご参照ください。

本書の構成は以下のようになっています。はじめてOpenShiftに触れる方はもちろん、ある程度OpenShiftに慣れてきて実践的な知識やノウハウを求める方にも役立つ内容がまとまっています。

第1部　入門編

OpenShiftの概要を理解し、AWS上での実際のインストールの手順、およびインストール後の基本的な使い方を一通り体験することができます。これにより、独力でOpenShiftを導入して環境を準備することが可能になります。これまでOpenShiftを使ったことがない方は、ここから始めることをおすすめします。

第2部　インフラ実践編

OpenShiftのアーキテクチャを理解し、複雑なインフラ要件に合わせてOpenShiftのインフラ設計および導入を行うために検討しておくべきポイントを把握できます。特にネットワークやストレージ、セキュリティに関して詳しく解説しています。また、導入後の運用やトラブルシューティングにおいて役立つ情報もまとめられています。

第3部　アプリ実践編

OpenShiftにおけるアプリケーションのビルドやデプロイの手順や利用可能なCI/CDツール、および設計指針について解説しています。これにより、OpenShift上で稼働するアプリケーションの開発やビルド・デプロイおよびCI/CDが実践できるようになります。既存のアプリケーションを単にコンテナ化するだけでなく、OpenShiftを活用してクラウドネイティブなアプリケーションを開発・運用するためのノウハウについてもまとめられています。

本書では特に断りがない限り、執筆時点での最新バージョンであるOpenShift 4.9を対象としています。実際にOpenShiftを利用する場合は、バージョンアップに伴う変更等によって本書の記載内容と異なる可能性があることにご注意ください。ご使用になる際は、都度最新バージョンのドキュメントを確認されることをおすすめします。次のサイトを参照するようにしてください[1]。

- Product Documentation for OpenShift Container Platform 4.9
  `https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9`

最後に、本書はコンサルタントやサポート、プリセールスなど、複数の立場からOpenShiftに携わっているメンバーから有志を募って執筆しました。そのため、記述内容は概説的なものにとどまらず、日々の業務で培われた実践的かつ実用性の高いものが詰まっています。OpenShiftは、その基盤となるKubernetesだけでなく、関連する多数のOSS（オープンソースソフトウェア）を含んだプロダクトであり、高頻度でアップデートされる内容をキャッチアップして整理するのはそれなりに難度の高い作業でした。執筆陣の努力により、本書は比較的普遍性の高い内容に仕上がっていると思います。本書で学んだ内容がすぐに古くなってしまうということはありませんのでご安心ください。

本書の執筆にあたっては、内容の検討やレビューで以下の方々にご協力いただきました。伊藤拓也、暮林達也、林智史、木村貴由、福地大輔、今井大亮の皆さん（敬称略）。この場を借りて感謝の意を表します。

執筆者にかえて
須江 信洋

【1】 英語ですが、https://docs.openshift.com/container-platform/4.9/にもドキュメントがあります。こちらのサイトでは、すべてのブックを串刺し検索できるので便利です。

本書の読み方

■概要と対象読者

本書は、エンタープライズ向けKubernetesディストリビューションの1つであるRed Hat OpenShift Container Platform（以下、OpenShift）に関する技術解説書です。エキスパートの経験に基づき、OpenShiftを実際に使うにあたって有用な情報をインフラとアプリケーションの両面から集大成しています。また、環境構築の章では自分でOpenShiftをインストールして利用するための手順も解説しており、これからOpenShiftに触れる方にとっても役立つ内容になっています。

本書の想定する対象読者は以下のとおりです。

- これからOpenShiftを使う方
- OpenShiftの基盤設計や運用に関わる方
- OpenShift上で稼働するアプリケーションの設計・開発・運用に関わる方

なお、紙面の制約上、本書ではOpenShiftのコアであるKubernetes自体についての解説は必要最小限にとどめています。KubernetesやLinuxコンテナに関する詳細情報については本書以外の解説書をあわせてご参照ください。

■サンプルについて

本書に掲載しているコードや設定ファイルのサンプルはGitHubから入手できます。

```
https://github.com/team-ohc-jp-place/openshift-tettei-nyumon
```

■動作確認環境

本書に掲載しているコードや手順は、可能な限り執筆時点（2021年11月）の最新バージョンであるOpenShift 4.9で動作を確認していますが、完全な動作を保証するものではないことにご注意ください。また、各種制約により、異なるバージョンでの動作を前提とする箇所はその旨を明記しています。

- OpenShiftクラスタへのアクセスには、Windows/macOS/Linuxが利用可能です。Webコンソールを利用するには、Webブラウザ（ChromeやFirefoxなど）が必要です。
- OpenShiftのインストールを実行するにはmacOSかLinuxが必要ですが、本書内ではクラウド環境のLinux仮想マシンを利用する手順も紹介しています。

■本書の構成

第1部　入門編

OpenShiftの概要が理解でき、実際にインストールや基本的な操作ができるようになります。これまでOpenShiftを使ったことがない方は、ここから始めることをおすすめします。

- **第1章　OpenShiftの概要**：クラウドネイティブ技術の市場動向を踏まえ、OpenShift の特徴やコンセプト、利用形態などを解説します。
- **第2章　環境構築**：AWS上でOpenShiftをインストールする手順を説明します。
- **第3章　基本的な操作方法**：OpenShiftのWebコンソールとコマンドラインインターフェース（CLI）の利用方法を説明します。
- **第4章　ユーザー管理**：ユーザーと権限の管理方法について説明します。
- **第5章　アプリケーションの実行**：サンプルアプリケーションのデプロイ方法を説明します。

第2部　インフラ実践編

OpenShiftのアーキテクチャが理解でき、複雑なインフラ要件に合わせてOpenShiftの設計や導入を行う際の検討ポイントを把握できます。また、導入後の運用やトラブルシューティングにおいて役立つ情報もまとめられています。

- **第6章　OCPのインストール方法**：設計とインストールにおける考慮点の説明と、最も汎用性の高いAny Platformインストールの手順を紹介します。
- **第7章　Operator**：アプリケーションやOpenShiftクラスタの運用を自律化するOperatorの概要およびOperator Frameworkなどについて説明します。
- **第8章　ネットワーク**：OpenShiftにおけるネットワーク関連の話題、特にRouterやEgressといったOpenShift特有の機能や、NetworkPolicyやMultusといった発展的な内容について解説します。
- **第9章　ストレージ**：OpenShiftにおけるストレージの存在意義と概念、取り扱いについて説明します。
- **第10章　セキュリティ**：コンテナアプリケーションにおけるセキュリティの考え方やOpenShift関連製品が提供するセキュリティ機能について説明します。
- **第11章　Day 2オペレーション**：インストール後の運用やトラブルシューティングについて解説します。
- **第12章　OpenShiftとマルチクラウド**：OpenShift によって実現する世界観としてのOpen Hybrid Cloudと、マルチクラウドに関連するOpenShiftの機能や関連する製品ポートフォリオを紹介します。

第3部　アプリ実践編

OpenShift上で稼働するアプリケーションの開発やビルド・デプロイおよびCI/CDが実践できるようになります。既存のアプリケーションを単にコンテナ化するだけでなく、OpenShiftを活用してクラウドネイティブなアプリケーションを開発・運用するためのノウハウについてもまとめられています。

- **第13章　アプリケーションのコンテナ化とモダナイズ**：アプリケーションをコンテナ化する際の前提知識となる、コンテナの特徴、コンテナデザインパターン、アプリケーションのモダナイズのアプローチなどについて説明します。
- **第14章　コンテナイメージのビルド**：OpenShift でのコンテナイメージのビルドに必要な前提知識について説明します。
- **第15章　アプリケーションのデプロイ**：OpenShiftでのアプリケーションデプロイ方法について説明します。
- **第16章　アプリケーション開発におけるライフサイクル**：アプリケーションのライフサイクルに関してメトリクス・ロギング・トレーシングの3つの視点からの紹介と、OpenShift上での実現方法について解説します。
- **第17章　クラウドネイティブ開発**：クラウドネイティブとは何かの説明から、サービスメッシュやサーバーレスなどのクラウドネイティブ開発に関する技術とOpenShiftとの関係について説明します。
- **第18章　DevOps**：OpenShiftで実践するDevOpsについて解説します。OpenShiftで利用できる、DevOpsツールの特徴や使い方に加え、効果的なDevOpsを実践するうえでのポイントについて紹介します。

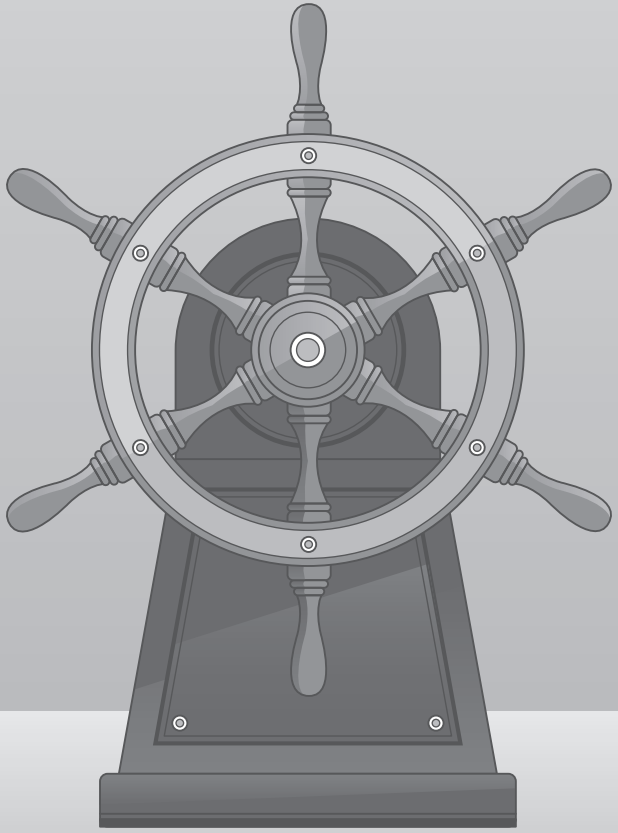
目 次

コラム目次

Chapter

1

OpenShiftの概要



Kubernetes[1]をパッケージングし、商用サポートを提供するKubernetesディストリビューション[2]は、2021年12月時点で61個あります。Red Hat OpenShift Container Platform（以下、OpenShift）は、世界で一番利用されているKubernetesの商用ディストリビューションの1つです。

「OpenShiftは多数のOperatorの集合体」と言うことができます。OperatorとはKubernetesを拡張する手法の1つであり、運用を自律化するためのカスタムコントローラーとCustomResource Definition（CRD）を指します（詳細は7.1節で解説します）。またOpenShiftは、クラスタアップグレードをはじめ、モニタリングやロギング、その他クラスタを構成するOperator群、高機能GUIのOperatorHub[3]から導入可能なサービスメッシュやCI/CD、ストレージクラスタなど、もともとのKubernetesにはない機能が多数追加され、ありとあらゆる要素がOperatorによってインストールでき、自律的に運用できるというコンセプトを持っています。

本章では、クラウドネイティブ技術の市場動向を踏まえ、OpenShiftの特徴やコンセプト、利用形態などについて解説します。

1.1 クラウドネイティブ技術の市場動向

Cloud Native Computing Foundation（以下、CNCF）がKubernetesのホスティングを開始したのは2014年です。それから約7年の歳月が経過し、Kubernetes（2021年11月現在のバージョンはv1.22）はコンテナオーケストレーションツールのデファクトスタンダードになり、世界中にインパクトを与えたテクノロジーの1つと言えるほどに熱い視線を浴び続けています。CNCF Case Studies[4]では、金融や通信、Eコマースなど、多種多様な業界・業種におけるクラウドネイティブ技術の活用事例が2021年11月現在91件紹介されています。また、2000以上の組織がビジネス上の課題を解決するためにOpenShiftを採用しています。

画期的な技術としてKubernetesが世界中から耳目を集めていることは事実ですが、Kubernetesはあくまでコンテナオーケストレーションツールです。コンテナオーケストレーションツールだけではプロダクション環境を安定稼働させることは困難であり、Kubernetes単品で使用されることはほとんどありません。

CNCF TOC（Technical Oversight Committee）では、クラウドネイティブ技術を次のように説明しています[5]。

【1】 Kubernetesはしばしば「K8s」と略されることがあります。
【2】 Certified Kubernetes Distribution
https://landscape.cncf.io/card-mode?category=certified-kubernetes-distribution&grouping=category
【3】 https://operatorhub.io/
【4】 https://www.cncf.io/case-studies/
【5】 https://github.com/cncf/toc/blob/main/DEFINITION.md

クラウドネイティブ技術は、パブリッククラウド、プライベートクラウド、ハイブリッドクラウドなどの近代的でダイナミックな環境において、スケーラブルなアプリケーションを構築および実行するための能力を組織にもたらします。このアプローチの代表例に、コンテナ、サービスメッシュ、マイクロサービス、イミュータブルインフラストラクチャ、および宣言型APIがあります。

これらの手法により、回復性、管理力、および可観測性［オブザーバビリティ］のある疎結合システムが実現します。これらを堅牢な自動化と組み合わせることで、エンジニアはインパクトのある変更を最小限の労力で頻繁かつ予測どおりに行うことができます。（※［ ］は引用者追記）

この説明にあるように、Kubernetesはクラウドネイティブ技術の1つであると解釈できます。ユーザーによって、使用するテクノロジーや重視する要素、アプローチは異なりますが、共通して言えるのは、Kubernetesだけで完結する話ではなく、あらゆるクラウドネイティブ技術を駆使して取り組むことが必然となるということです。

1.1.1 CNCF Projects

CNCFは、Kubernetesをはじめとする多数のプロジェクト[6]をホスティングし、プロジェクトの活性化を支援しています。

ホスティングしているプロジェクトは、基本的には成熟度別（成熟度が高い順）に「Graduated」「Incubating」「Sandbox」の3段階に分類されます（**図1.1**）。2021年11月時点では、KubernetesやPrometheusをはじめとするGraduatedプロジェクトは16個あります。Incubatingは25個、Sandboxは

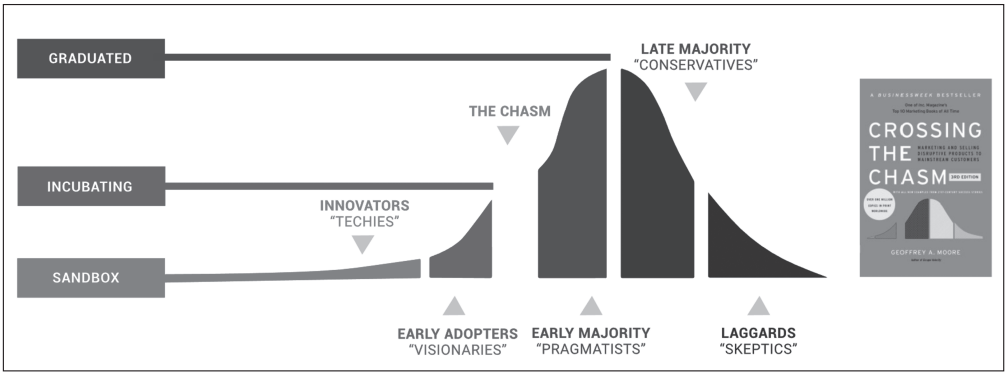


図1.1　成熟度レベル（https://www.cncf.io/projects/）

【6】 CNCF Projects
https://www.cncf.io/projects/

65個あります。また、「Archived」というカテゴリもあり、現時点ではコンテナランタイムの「rkt」がここに分類されています。各成熟度に分類される際の指標など詳細については、CNCF公式ドキュメント[7]を参照してください。

「CNCF Cloud Native Landscape」（**図1.2**）は、未知の領域であるクラウドネイティブ技術を歩んでいく開発者や企業などのユーザーを支援するために提供されているリソースマップのような存在です。各社が提供するCertified Kubernetesディストリビューションや、CI/CDツール、セキュリティ、コンテナレジストリ、データベース、クラウドネイティブストレージなど、カテゴリ別に多種多様なプロジェクトが掲げられています。オープンソースソフトウェア（Open Source Software：OSS）はもちろんのこと、ベンダーがサポートを提供する商用プロジェクトも記載されています。

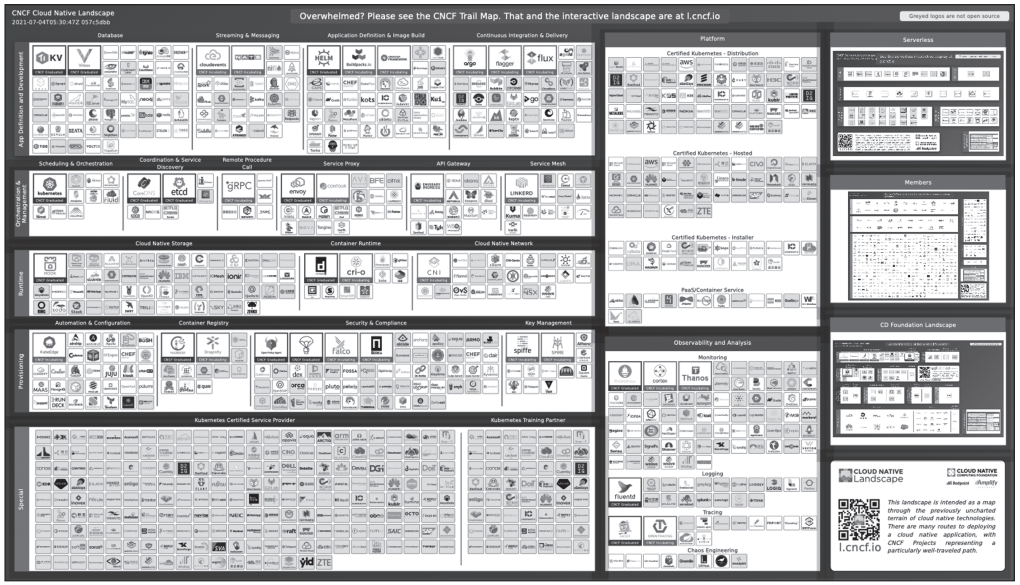


図1.2　CNCF Cloud Native Landscape（https://landscape.cncf.io/）

Kubernetesに触れたことがない方は、「なぜKubernetes周辺ツールが100個近くあるんだ」と思うかもしれません。たしかに特段の意識をせずともKubernetesを使うことで、コンテナのスケーリングや宣言的な運用など一般的なコントロールはできます。

一方で、コンテナの実行速度や起動速度などのパフォーマンス面やセキュリティ面を考慮して内部的に動作するコンテナランタイムを選択したり、Kubernetesクラスタを構成する環境のネットワーク特性を考慮して適切なネットワークを選択したり、既存および今後使いたいストレージに対応させたり、監視・オブザーバビリティ用のツールや、CI/CDツール選定など、目的や用途に合わせた自由

【7】 https://github.com/cncf/toc/blob/main/process/project_proposals.adoc

選択ができるように多数のプロジェクトがラインナップされています。

また、Kubernetesの重要なコンセプトとして、多種多様なインターフェースを通じて疎結合できるように開発されていることが挙げられます。ネットワークであればCNI（Container Network Interface）、ストレージであればCSI（Container Storage Interface）のように、インターフェースを通じて疎結合できるように開発されています。このため、任意のネットワークやストレージなどを選択して利用するのもKubernetesであれば容易です。

コンテナオーケストレーション以外に必要になる代表的な要素を、クラウドネイティブを実現するロードマップ「Cloud Native Trail Map」（**図1.3**）にならって2つ取り上げます。

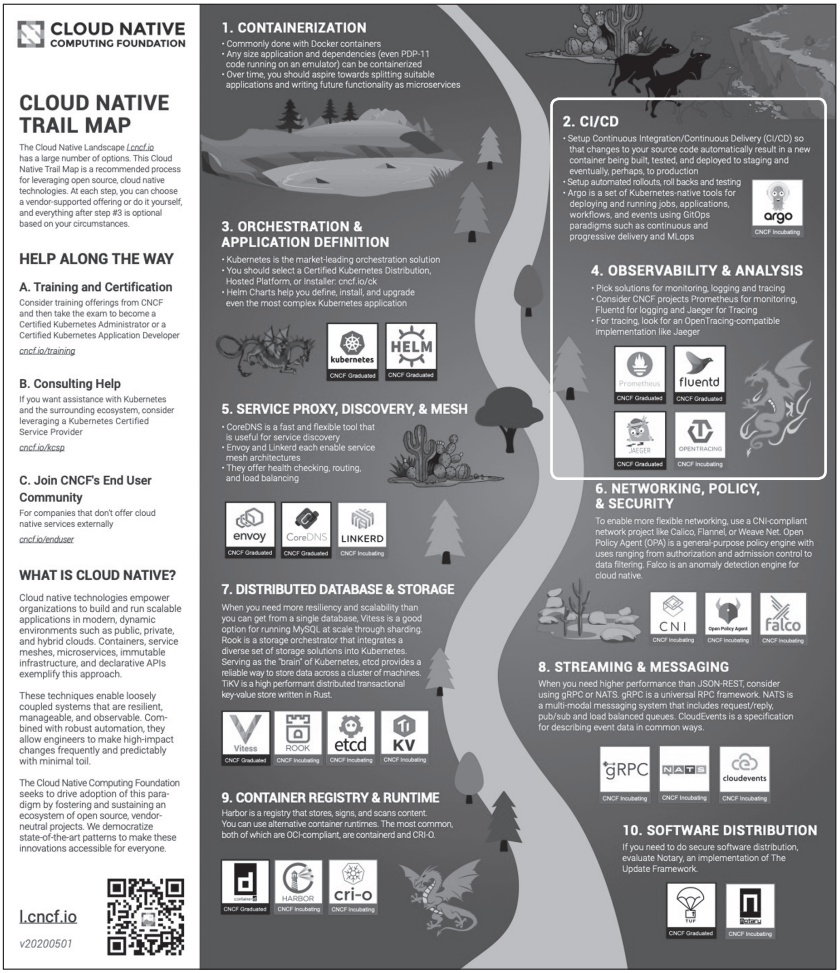


図1.3　Cloud Native Trail Map（https://github.com/cncf/landscape/blob/master/README.md#trail-map）

- Cloud Native Trail Map：2. CI/CD
- Cloud Native Trail Map：4. Observability and Analysis

Cloud Native Trail Mapは、OSSやクラウドネイティブ技術活用のための道しるべとして、技術カテゴリ別にステップに分けて解説およびプロジェクトがマッピングされています。

CI/CDやオブザーバビリティは、従来からコンテナに限らず既存のアプリケーションでも実施してきたように、アプリケーションを各環境にデプロイしたり、継続的に状態を監視するなど、当たり前に実施されてきたことでもあります。

1.1.2 CI/CD——継続的インテグレーションと継続的デリバリー

CI/CDは日本語では、継続的インテグレーション／継続的デリバリーと訳されます。CIはContinuous Integrationの略語で、CDはContinuous Delivery/Deploymentの略語です。CI/CDは、ソフトウェアの変更を常にテストし、ステージング環境や本番環境に自動的に正しくリリースできる状態を整えておくソフトウェア開発手法の1つです。

コンテナ／Kubernetesの文脈では、次のような使い方が想定されています。

> アプリケーションコードの変更をトリガーにして、新しいコンテナイメージをビルドし、コンテナレジストリに格納する。ユニットテストや統合テスト、脆弱性検査など必要なテストをパスした後に任意のKubernetesクラスタにコンテナをデプロイする。

CI/CDを使えば、バグやセキュリティ的な問題をいち早く発見して解決することが可能になります。さらに、品質を高め、変更に応じて自動的にリリースすることでアジリティのある開発ができるようになります。

また、アプリケーションコードによらず、各種のKubernetesマニフェストをGitHubなどのGitリポジトリをSingle Source of Truth（唯一信頼できる情報源）として管理し、リポジトリ上のコード、構成ファイルの変更をトリガーに、Kubernetesクラスタ上のコンテナの構成やクラスタ設定を変更し、システム全体をあるべき状態（Desired State）に近づける「GitOps」[8]という手法もCI/CDのプラクティスの1つとして注目を浴びています。

1つのコンテナのみを扱い、動作環境も限定的である場合は、CI/CDは必須ではないかもしれません。一方で、コンテナオーケストレーションツールであるKubernetesを用い、多数のコンテナを扱う場合はどうでしょうか。Kubernetesクラスタ上で膨大な数のコンテナが動作し、盛んに新機能開発が行われるクラウドネイティブな環境を想像してみてください。都度、人間の手によってコンテナイ

【8】 https://www.weave.works/technologies/gitops/

メージを作成し、テストを行い、クラスタに合わせた設定を動的に行った後にリリースするのは、時間的コストだけでなく、オペミス発生の温床にもなりかねないことは容易に想像できるでしょう。

CI/CDは、従来のソフトウェア開発においても実践できるに越したことはない開発手法でしたが、Kubernetesをはじめとするクラウドネイティブ技術の活用においては「実践することが必須条件」とも言えます。

クラウドネイティブ以前から古くから利用者の多いJenkinsや、KubernetesネイティブなCIツールとしてのTekton、CDツールとしてのArgo CDなどが注目を浴びています。他にもソフトウェア、SaaSなど多数のプロジェクトがあるので、興味のある方はCNCF Cloud Native Landscapeの技術カテゴリ「Continuous Integration & Delivery」を眺めてみてください。

1.1.3 Observability and Analysis ——オブザーバビリティとアナリシス

オブザーバビリティ（Observability：可観測性）は、「システムがいかに観測されやすい状態であるか」を意味します。

分散システムでは、すべてのマシンが完全な状態で動作していることは期待できず、常にどこかが壊れている可能性があります。これは、コンテナ／Kubernetesを活用したクラウドネイティブな環境でも同じことが言えます。従来の監視では、ピンポイントにアプリケーションやネットワーク、データベース、ストレージなどを指定して問題が起きていないかを把握し、アラートを発行することで、「壊れてしまっている対象物」を認識することは可能です。

一方で常にどこかが壊れている可能性のある分散システムでは、従来の監視だけでなく「なぜ壊れてしまっているのか」「何がどう劣化しているのか」を理解できることが求められます。サービス利用者であるエンドユーザーから見て、「対象サービスが利用できない状態」はサービスダウンを意味しますが、対象サービスを構成する分散システムの一部が壊れていることは直接関係がなく、サービスが不満なくいつもどおり利用できることが重要です。そのため、「何が原因で部分的に壊れているのかを観測できるシステムのオブザーバビリティ」が非常に大切になります[9]。

Cloud Native Trail Mapの「4. Observability and Analysis」では、オブザーバビリティを高めるため、モニタリング用途に「Prometheus」を、ロギング用途に「Fluentd」を、トレーシング用途に「Jaeger」を使うことを検討するよう記載されています。これら3つのプロジェクトは、CNCF Cloud Native Landscapeの技術カテゴリ「Observability and Analysis」に分類されており、成熟度レベル（Maturity Level）はGraduatedに位置し、開発コミュニティが活発で、非常に多くのユーザーに使用されています。興味のある方は、各プロジェクトのWebサイトやコードリポジトリなど確認してみてく

【9】 オブザーバビリティと監視の違いや、Kubernetesをメトリクス中心のアプローチによってオブザーバビリティを高めるヒントについては、『Kubernetesで実践するクラウドネイティブDevOps』（John Arundel、Justin Domingus著、オライリー・ジャパン、2020年）などの解説書を参考にしてください。

ださい。

Kubernetesをはじめとするクラウドネイティブ技術は、DevOpsと可観測性に非常に相性が良いとされています。通常は企業組織にDevOpsを推進するなど大きな変革には膨大な時間を要しますが、このような組織変革でも、現在のKubernetesの盛り上がりのムーブメントに乗って、クラウドネイティブ技術を積極的に取り入れ、CI/CDという開発手法やオブザーバビリティを導入していけば、テクノロジードリブンなアプローチで道を切り拓いていくことも期待できます。

1.2 KubernetesとOpenShiftの違い

OpenShiftは、Kubernetesの商用ディストリビューションです。ここでは、アップストリーム版のKubernetes[10]に対して、OpenShiftが提供する付加価値について説明します。

OpenShiftは、CNCFの適合性プログラムに準拠[11]しています。これによって、アップストリーム版のKubernetesとAPIレベルでの互換性があり、ユーザーから見た相互運用性が保証されています。そのため、OpenShiftを「エンタープライズ版Kubernetes」として利用する企業も多く存在します。たとえば、Kubernetesの機能の範囲内であれば、ocコマンド（OpenShift固有のCLI）ではなく、kubectlコマンドを利用して他のKubernetes運用との互換性を担保できます。

OpenShiftは、各クラウドプロバイダー上で提供されるマネージドOpenShiftサービスだけでなく、自身で物理サーバーや仮想マシン、プライベートクラウド、パブリッククラウド、Edgeなど、あらゆるプラットフォーム上に構築し利用することができます。エンタープライズ向けの製品としては当然ですが、サブスクリプションを契約することで、製品サポートを受けられます[12]。

OpenShiftは、エンタープライズ利用を想定したKubernetesとして多種多様な機能群を備え、Kubernetesを利用しやすく、運用しやすくするための配慮が行われています。OpenShiftがカバーする機能の全体イメージを**図1.4**に示します。図中の「Kubernetes（orchestration）」の部分が、アップストリーム版のKubernetesがカバーしている範囲です。それに対し、OpenShiftは、Kubernetesクラスタを構成するノード（物理、仮想環境などにOSが入ったノード）のレイヤーを含む、Kubernetesクラスタ上で動作する多種多様なコンポーネントも含む製品となっています。

【10】CNCFが配布するKubernetesを「アップストリーム（上流）版のKubernetes」と呼びます。

【11】CNCF Certified Kubernetes - Distribution (OpenShift)
https://landscape.cncf.io/card-mode?category=certified-kubernetes-distribution&grouping=category&selected=red-hat-open-shift

【12】提供形態やサブスクリプションについては、本章の「1.3 OpenShift利用形態とサブスクリプション」を参照してください。

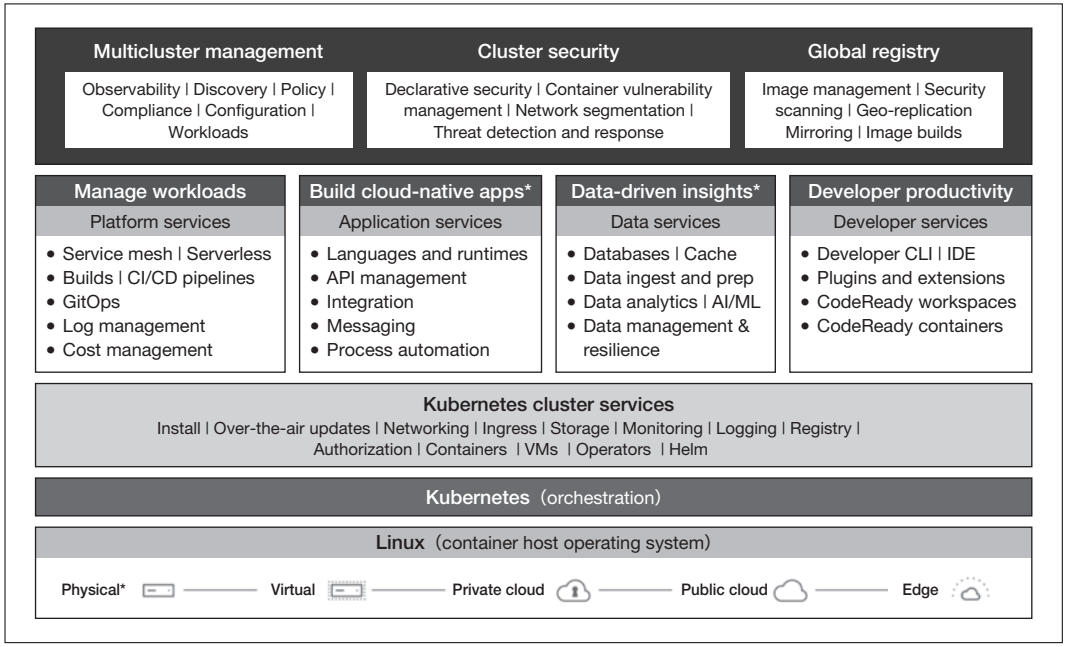


図1.4 OpenShiftがカバーする機能の全体イメージ

たとえば、クラスタモニタリングやロギング、クラスタアップグレードなど、アプリケーションやKubernetesクラスタの運用を下支えする機能群や、CI/CD、ServiceMesh、ServerlessなどDevOpsをはじめとする開発生産性の向上や、マイクロサービスのオブザーバビリティを向上させる仕組みなどを必要に応じて容易に追加できるように構成されています。

他にも、内部コンテナレジストリや、コンテナアプリケーションへのトラフィックを管理するRouter、充実したGUI（Graphical User Interface）が標準機能として用意されているため、運用者が別のツールを選択したり、導入に時間を割いたりする必要はありません。

OpenShiftはアップストリーム版のKubernetesを包含しており、Masterノード（Control Plane）で稼働する「etcd」の可用性（Quorum）を担保するために3台、Workerノードはノードレベルの障害の影響を回避するために2台以上のクラスタ構成が推奨されています【13】。

また、Operatorというテクノロジーを活用することによって、OpenShiftクラスタのインストール、構成変更、アップグレード、機能追加などを容易にしています。OpenShiftの根幹を成す部分と言っても過言ではないのが、Operatorによる自律運用です。Kubernetes APIを拡張することによって、Kubernetesが管理できないリソースに対しても、Kubernetesによって宣言的な管理を可能にします。Operatorによって、クラスタがあるべき状態を維持するように管理され、クラスタ上への機能追加お

【13】基本的には、Masterノード（Control Plane）はK8sクラスタおよび各リソースの管理を行い、Workerノードでは実働アプリケーションが動作します。詳細については、第6章の「6.2.2 ノード」を参照してください。

よび追加された機能のライフサイクル管理もサポートされます[14]。

さらに、1台のクラスタのみならず企業で利用される複数のクラスタの管理や、マルチクラウド上に存在する多数のクラスタなどを統合管理するための機能や、高度なセキュリティ機能、大規模利用を想定したレジストリなど、用途に合わせたオプションも用意されています。

これらの特徴は一例にすぎませんが、企業がKubernetesを利用するにあたって、Kubernetesだけでは不足している運用機能や開発生産性を高められる機能、組織など複数人での使い勝手を便利にする権限設定などの機能をアップストリーム版のKubernetesに追加し、OpenShiftとして提供しています。

OpenShiftのサポート期間

OpenShift は、最新リリースを含む直近３マイナーバージョンまでがサポート対象です。たとえば「4.7」が最新リリースである場合は「4.5」までがサポートされ、「4.4」はEOL（End Of Life：サポート終了）になります。

1.2.1 クラスタ構成

OpenShiftでは、アップストリーム版のKubernetesと同様に、Masterノードと Workerノードによってクラスタを構成します。さらに、OpenShiftではInfraノードを追加し、モニタリングサービスやコンテナイメージレジストリやルーティングをサポートする機能などをMasterノードやWorkerノードではなく、Infraノードに委譲させることもできます。

OpenShiftクラスタの設計や構成コンポーネント、システム要件などについては、第6章の「6.2 クラスタの設計」を参照してください。

1.2.2 クラスタの構築

ユーザーはクラウドプロバイダーが提供するマネージドOpenShiftサービスを利用できますが、自身でオンプレミス環境やクラウドリソースを使用してOpenShiftを構築することもできます。構築方法には、IPIとUPIの2つの方法があります。

■IPI（Installer-Provisioned Infrastructure）

IPIは、インストーラーを用いて動的に仮想マシンやロードバランサー、ストレージなどを準備し、OpenShiftクラスタをインストールする方法です。IPIの詳細およびIPIによるインストール手順につ

【14】Operatorの詳細については、第7章「Operator」を参照してください。

いては、第2章の「2.1 IPIインストール」を参照してください。

■UPI（User-Provisioned Infrastructure）

UPIは、事前に仮想マシンなどのリソースを準備し、OpenShiftクラスタをインストールする方法です。オンプレ環境やインターネット接続に制限のある環境や、既存のIT資産を活用する場合に使われます。UPIの詳細およびUPIによるインストール手順については、第6章の「6.3 UPIインストール」を参照してください。

それぞれのインストール方法が対応するクラウドプロバイダーやプラットフォームは、以下のとおりです（最新の対応環境は公式サイト[15]で確認してください）。

IPI

- Amazon Web Services（AWS）
- Google Cloud Platform（GCP）
- Microsoft Azure（Azure）
- Red Hat OpenStack Platform（RHOSP）version 13 and 16
- Red Hat Virtualization（RHV）
- VMware vSphere
- VMware Cloud（VMC）on AWS
- ベアメタル

UPI

- AWS
- GCP
- Azure
- RHOSP
- RHV
- VMware vSphere
- VMware Cloud（VMC）on AWS
- IBM Z or LinuxONE
- IBM Power Systems
- ベアメタル

【15】OpenShift 4.9 Supported platforms for OpenShift clusters
https://docs.openshift.com/container-platform/4.9/architecture/architecture-installation.html

1.2.3 拡張コンポーネント

OpenShiftのMasterノードには、Kubernetesのクラスタ制御サービスが起動しています。Masterノードで稼働するKubernetesのサービスとして以下のものがあります。

- Kubernetes API（kube-apiserver）
- etcd（etcd-member）
- Kubernetes Controller Manager（kube-controller-manager）
- Kubernetes Scheduler（kube-scheduler）

さらにOpenShiftでは、Kubernetesコンポーネントを補う拡張コンポーネントが提供されています。たとえば、Kubernetesの中核となるKubernetes API（kube-apiserver）に対して、機能を拡張したOpenShift API（openshift-apiserver）がOperator経由で展開されています。OpenShift APIはあくまでKubernetes APIを拡張しているだけにすぎず、アップストリーム版のKubernetesと同様に「kubectl」コマンドを利用できるように構成されています。しかし、OpenShift専用の管理コマンドの「oc」を利用することで、拡張機能が容易に利用できるようにしています。たとえば、「oc」コマンドを利用してコンテナアプリケーションのビルドやクラスタ管理操作を直観的に行うことができます。

OpenShiftのWorkerノードでは、Kubernetesのコアサービス以外にもノード監視を行うためのPrometheusのエージェント（Node Exporter）やクラスタ間のネットワークを経由するCNIプラグインなどが存在します。これらは、OpenShiftを構成するうえで欠かせない機能であるため、Masterノード上でも同様のコンテナが稼働しています。他にもOpenShiftでは、「MachineSet」というリソースオブジェクトがノードを制御しています。これは、KubernetesにおけるPodとReplicaSetの関係と似た機能です。つまり、アプリケーションのワークロードに応じてMachineSetオブジェクトのreplicasフィールドを変更することで、動的なノードの追加や縮退が制御できます。ユーザーがWorkerノード上のサービスの制御を積極的に行うのではなく、OpenShiftがアプリケーションワークロードに応じてWorkerノードの管理を自動的に行う仕組みに委ねることで、インフラの管理コストの削減に役立ちます。

1.2.4 モニタリング

OpenShift上のモニタリングは、クラスタモニタリングとアプリケーションモニタリングの2つに分かれており、Prometheusを使用しています。クラスタの稼働メトリクス情報は「openshift-monitoring」というプロジェクト内のOperatorが管理する、Prometheus（Prometheus Operator）によって取得されています。また、取得したメトリクス情報はGrafanaで可視化されます。可視化に関

しては、Grafana以外にもOpenShiftのコンソール（GUI）上から確認することも可能です。

このコンテナのメトリクス取得には、「kube-state-metrics」が利用されています。kube-state-metricsは、Kubernetes APIを監視し、OpenShiftが提供するPodやDeployment、DaemonSetなどのオブジェクトに対して、メトリクス情報を取得します。また、Prometheus上のCPUやメモリ使用率などのリソースをメトリクスに変換し公開することで、HPA（Horizontal Pod Autoscaling）にも活用されます。特別な設定を行わなくてもOpenShift上で監視できます。ただし、固有アプリケーションのメトリクスを監視する場合は「kube-state-metrics」では取得ができないため、別途設定が必要です。

Prometheusの運用そのものはPrometheus Operatorが行いますが、取得するメトリクスの指定はユーザー自身が決めて対応します。また、Prometheus Operatorでは、Alertmanagerも管理できます。Alertmanagerを使うことによって、Prometheusが取得したメトリクスをトリガーに、Webhookを呼び出してアラートを上げることも可能です。

クラスタモニタリングの機能紹介については第11章の「11.2 クラスタモニタリング」を、アプリケーション（ユーザーワークロード）のモニタリングについては第16章の「16.2 アプリケーションモニタリング」を参照してください。

1.2.5 ロギング

OpenShiftでは、ログ監視のため、EFK（Elasticsearch、Fluentd、Kibana）スタックを活用したCluster Logging（Red Hat OpenShift Logging）の仕組みがあります。

ログコレクターとしてFluentdが利用され、各Workerノード上にDaemonSetとして展開されます。これによってすべてのノードのログを収集し、Elasticsearchに蓄積して分析を行います。また、開発者やクラスタ管理者が集計されたデータを分析するため、Kibanaによって可視化しています。ただし、Elasticsearchは比較的多くのリソースを必要とするため、クラスタインストールの初期状態では、Cluster Loggingは稼働しないようになっています。必要に応じて管理者が、Cluster Logging OperatorとElasticsearch Operatorの双方を準備し、EFKスタックを構築します。

Cluster Logging Operatorは、ログの収集（Collection）、保存（LogStore）、可視化（Visualization）といった、EFKスタックの設定を行うインターフェースです。Cluster Logging Operatorを使えば、ElasticsearchとKibanaはOpenShiftとの認証連携とアクセス制御ができます。「一般の開発者は所属するプロジェクトのログのみ参照できる」といった制限も設定できます。

Cluster Loggingは比較的処理に負荷のかかるコンポーネントであるため、アプリケーションワークロードが動作するWorkerノードと切り離してスケジューリングしたほうが安定稼働が期待できます。この場合、Workerノードの負荷を軽減するためにEFKスタックをInfraノードに導入することもできます。本書では扱いませんが、OpenShiftのサブスクリプション対象がWorkerノードであることから、Infraノードにインフラ機能として構築する選択も考えられます。

Cluster Loggingの概要やインストール方法については、第11章の「11.2 クラスタモニタリング」を参照してください。アプリケーションのロギング方法や、Kibanaダッシュボードを用いたログ検索方法などについては、第16章の「16.2 アプリケーションモニタリング」で解説しています。

1.2.6 ネットワーク

Kubernetesにおけるコンテナ間のネットワークは、Container Network Interface（CNI）を実装するネットワークプラグインに依存します。CNIは、コンテナのネットワークインターフェースを設定するための仕様です。

ネットワークプラグインがオーバーレイネットワークを作ることで、各Workerノードに配置されるコンテナ同士が疎通できます。OpenShiftが同梱するCNIプラグインは、OpenShift SDN、OVN-Kubernetes、Kuryrの3つです。この他にも様々なネットワークプラグインをサポートしています。OpenShift 4.9のデフォルトでは「OpenShift SDN」というOpen vSwitch（OVS）ベースのCNIプラグインを使用し、オーバーレイネットワークを構築します。

こうした、ネットワークプラグインもCluster Network Operator（CNO）によって管理されています。CNOは、OpenShift SDN等のCNIプラグインを管理したり、ユーザーによる設定変更の内容が悪影響を及ぼさないか事前に検査したりします。

OpenShiftにおけるネットワークの詳細については、第8章「ネットワーク」を参照してください。

1.2.7 ストレージ

Kubernetes上でデータベースのようなステートフルなアプリケーションを扱うためには、データを保存するための永続化を実現する方法とストレージが必要です。ビルド済みコンテナイメージをレジストリに格納しておくことで、常に同一のコンテナを動作させることができますが、コンテナは揮発性のあるプロセスであり、起動・再起動、再作成を短いサイクルで繰り返します。つまり、コンテナ内に永続データを保存することはできません。アプリケーションに関連するデータだけでなく、前述のクラスタモニタリングやロギング、内部レジストリなど、様々な用途で必要となる永続データが存在します。

コンテナのストレージインターフェースを設定するための仕様として、Container Storage Interface（CSI）があります。プラグインのCSI driverを用いれば、Kubernetesとバックエンドストレージが通信できるようになります。

また、Red Hat OpenShift Data Foundation（ODF）と呼ばれるOpenShiftクラスタのための専用SDS（Software-defined Storage）が用意されています。ODFは、Rookをコアテクノロジーとして活

用しており、クラスタ内部にストレージを構成するため基盤を問わず、Operatorによってインストールからストレージクラスタの構成や変更などの作業を制御できます。さらに、OpenShiftに組み込まれたPrometheusによって利用状況やメトリクスの取得も可能です。

OpenShiftにおけるストレージの詳細については、第9章「ストレージ」を参照してください。

1.2.8 クラスタアップグレード

Kubernetesは、アップデートサイクルが短く、約4か月ごとに新しいバージョンがリリースされています。ユーザーはこのアップデートサイクルに追従する必要があります。

OpenShiftは、OpenShiftクラスタのアップグレードを支援する仕組みであるOver-The-Air（OTA）アップグレード機能を用意しています。マネージドKubernetesを利用しているかのようにコンソール上あるいはCLIからクラスタバージョンをアップグレードできます。ここでも内部的には、Operatorが活躍しています。Cluster Version Operator（CVO）というOperatorが、現行クラスタで動作する多数のOperator群を管理しており、クラスタのアップグレードパス（アップグレード可能なバージョンの確認、複数バージョンをまたぐ場合の経路など）を確認し、アップグレードを実行する役割を担っています。

OpenShiftのアップグレードの仕組みや手順については、第11章の「11.5 クラスタのアップグレード」を参照してください。

1.2.9 CI/CD

OpenShiftは、Kubernetesの利用において必要不可欠であるCI/CDを実現するための機能として「OpenShift Pipelines」と「OpenShift GitOps」を提供しています。これらはOperatorを通じてOpenShiftに組み込まれます。

OpenShift Pipelinesは、内部的には「Tekton」を使用しています。Tektonは、CIパイプラインの構成・設定をKubernetesリソース（マニフェスト）として管理ができ、KubernetesネイティブなCIツールと呼ばれるOSSです。OpenShift Pipelinesでは、CIパイプラインの開発・利用をサポートする機能群をコミュニティ版Tektonに加えて統合しています。またCI/CDのシーンでよく使われてきたJenkinsについては、Red HatがビルドしたJenkinsイメージを引き続き提供しています。

OpenShift GitOpsは、内部的には「Argo CD」を使用しています。Argo CDは、CD（継続的デリバリー）に特化したCDパイプラインツールであり、KubernetesネイティブなCDツールと呼ばれるOSSです。GitOpsは、Gitリポジトリを Single Source of Truthと捉えることで、リポジトリ上のコードや設定変更をトリガーにしてクラスタに対して変更を反映させるアプローチを採用しています。これ

はKubernetesのControl Loop／リコンシリエーションループ（Reconciliation Loop）と同様に、宣言的な構成管理と自動化によってあるべき状態（Desired State）の維持を実現します。OpenShift GitOpsもOperatorを通じてArgo CDのインストール、構成管理が行われます。

OpenShift PipelinesやGitOpsの概要や使い方、そしてJenkinsとの比較については、第18章「DevOps」を参照してください。

1.2.10 その他

これまで見てきたもの以外にも、エンタープライズ用途でKubernetesを最大限活用できるよう、多数の機能がOpenShiftには組み込まれています。たとえば、セキュリティ、コンテナイメージのビルド、アプリケーションのデプロイ、クラウドネイティブ開発、マルチクラウド対応などです。以降の章では、それぞれ詳しく解説していきます。

COLUMN

Red Hat と OpenShift に期待されていること

皆さんはKubernetesとうまく付き合えていますか？

ご存じのとおり、Kubernetesを利用することで、コンテナのスケーリングやセルフヒーリング、宣言的記述（マニフェスト）による制御を行うことができますよね。一方で、コンテナを動作させるまでの準備や、コンテナ動作後の運用、クラスタ自体の運用など、Kubernetesだけではカバーできない部分があります。一例を挙げると、コンテナイメージのビルド、コンテナイメージの格納・管理、Kubernetesクラスタへのデプロイ、起動したコンテナのモニタリング、コンテナ間（連携するアプリケーションプロセス間）のトレーシング、さらにKubernetesクラスタ自体の管理や、セキュアな状態の確保などです。必ず、Kubernetes周辺の技術要素について検討し、何かしらの手だてを整えなければなりません。

そこで、筆者が把握している範囲で、Red HatおよびOpenShiftに対するユーザーからの期待の声の代表例を以下に挙げてみます。

- KubernetesのRed Hatによる商用サポート
- OpenShiftに組み込まれるOSSテクノロジー選定における先進性と安定性の両立
- Kubernetesへのアプリケーションデプロイの容易性
- Kubernetes上でのアプリケーション開発の柔軟性
- クラスタインストールの容易性、俊敏性
- クラスタアップグレード、他運用面の負担軽減
- ハイブリッド、マルチクラウド対応
- セキュリティ

ユーザーによって求める部分は異なりますが、共通して言えるのは「Kubernetesの運用をいかに手間なく行えるか、そしてKubernetesの能力をいかに即座に利用できるか」に対するRed HatおよびOpenShiftの動向への期待と思われます。テクノロジーの進化とそれを使いこなすことは終わりなき戦いとも言えるかもしれませんが、いちKubernetes好きとしては、技術を負債の要因とせず、最大限活用することを楽しんでいただきたいと思います。

1.3 OpenShift利用形態とサブスクリプション

ユーザーは、2種類の利用形態を選ぶことができます。

- マネージドOpenShiftサービス（Hosted OpenShift）：クラウドプロバイダーが提供したものを利用する（**図1.5**）
- セルフマネージドOpenShift（Self-Hosted OpenShift）：オンプレミスなどに自身で構築して利用する

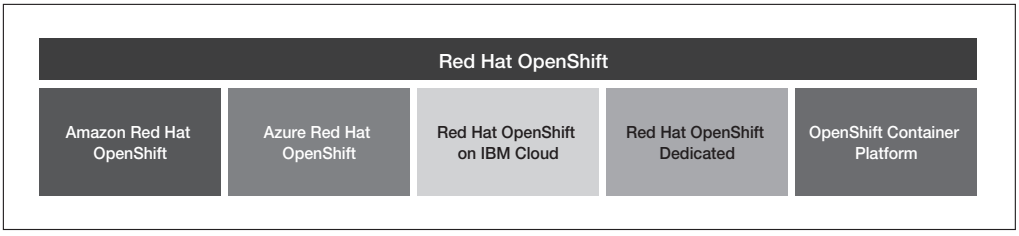


図1.5　マネージドOpenShiftサービス

1.3.1 マネージドOpenShiftサービス

様々なクラウドプロバイダーがマネージドOpenShiftを提供し、運用を行っています。具体的には、クラスタの管理やノードのセキュリティ確保、パッチの適用管理、クラスタアップグレード、メトリクスログの収集など、クラスタ維持に伴う運用作業を行います。ユーザーは、各クラウドプロバイダーのマネージドOpenShiftサービスのプランを契約することで使用できます。以下のようなプランがあります。

- Red Hat OpenShift Dedicated（OSD）
- Red Hat OpenShift Service on AWS（ROSA）
- Azure Red Hat OpenShift（ARO）
- Red Hat OpenShift on IBM Cloud（ROKS）

各種マネージドサービスにおいて、Red Hatのサポートチームも各社と連携したサポートを提供します（**表1.1**）。

表1.1　マネージドOpenShiftサービス提供主体

提供サービス	Red Hat OpenShift Dedicated（OSD on AWS/GCP）	Red Hat OpenShift Service on AWS（ROSA）	Azure Red Hat OpenShift（ARO）	Red Hat OpenShift on IBM Cloud
Infrastructure	AWS or Google Cloud	AWS	Azure	IBM Cloud
Billed by	Red Hat / AWS or Google Cloud	AWS	Microsoft	IBM
Managed by	Red Hat	AWS & Red Hat	Microsoft & Red Hat	IBM
Supported by	Red Hat	AWS & Red Hat	Microsoft & Red Hat	IBM & Red Hat

※執筆時点（2021年11月）の情報です。最新情報は以下を確認してください。
https://cloud.redhat.com/products

1.3.2 セルフマネージドOpenShiftとサブスクリプション

ユーザーは、オンプレミスやパブリッククラウドリソースを使用してOpenShiftを構築できます。この利用形態を「セルフマネージド」と呼び、すべての運用をユーザー自身で行います。

Red Hat社からサブスクリプションを購入することで、OpenShiftを使用し、サポートを受けられます。以下の3種類のサブスクリプションが用意されています（**表1.2**）。

- Red Hat OpenShift Platform Plus [16]
- Red Hat OpenShift Container Platform
- Red Hat OpenShift Kubernetes Engine

【16】https://www.openshift.com/products/platform-plus

表1.2　セルフマネージドOpenShiftの機能の違い

同梱製品	サブスクリプション			備考
	Red Hat OpenShift Platform Plus	Red Hat OpenShift Container Platform	Red Hat OpenShift Kubernetes Engine	
Red Hat OpenShift Container Platform	○	○	○ （機能制限有）	Kubernetes運用および開発を支援する拡張機能を提供
Red Hat Advanced Cluster Management for Kubernetes	○	×	×	オンプレミス、パブリッククラウドを問わず、複数のKubernetesクラスタの統合管理機能を提供
Red Hat Advanced Cluster Security for Kubernetes	○	×	×	ビルド、デプロイ、実行中などあらゆるタイミングで適用可能なセキュリティポリシーを多数提供
Red Hat Quay	○	×	×	セキュアかつグローバルに利用可能な大規模レジストリ機能を提供

COLUMN

サブスクリプションの考え方

OpenShiftクラスタは、Masterノード、Workerノード、Infraノード（オプション）を独立して用意することで構成されます。OpenShiftのサブスクリプションは、基本的にはノードに搭載されたCPUコア数に対して課金されます。CPUコア数をカウントする対象ノードは、Workerノードだけです。Masterノードや Infraノードは考慮する必要がありません。**図1.6**に例を挙げます。

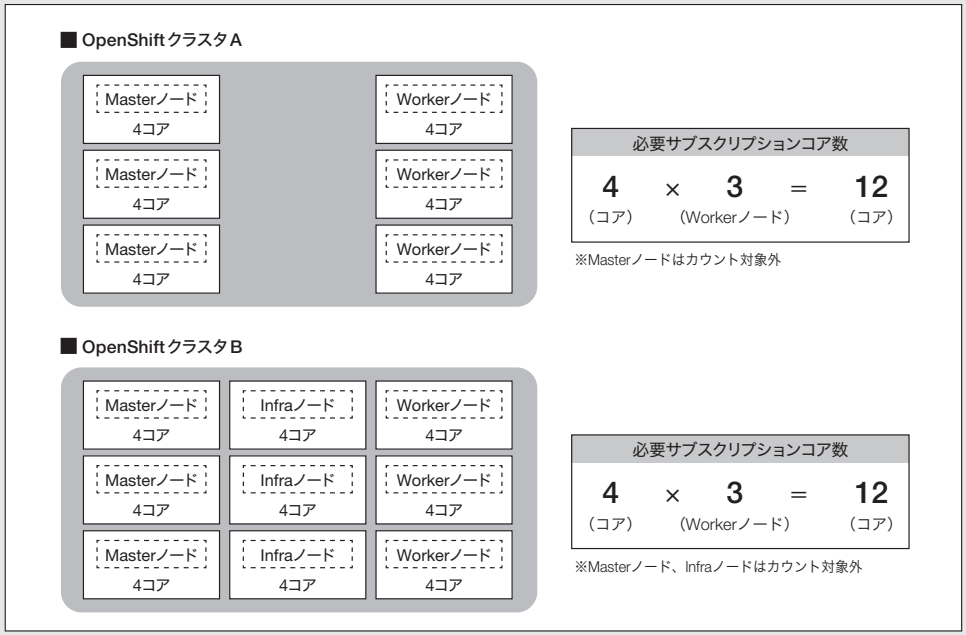


図1.6　必要サブスクリプションコア数の例

例外的にMasterの役割とInfraの役割をまとめて1つのノードとして構成することがあります。さらにEdgeなど特殊なケースでは、Master/Workerなどすべての機能を1台のノードに集約する構成例もあります。基本的な考え方は通常構成時と同様に、Workerノードを構成するノードのCPUコア数に相当するサブスクリプション数が必要となります。

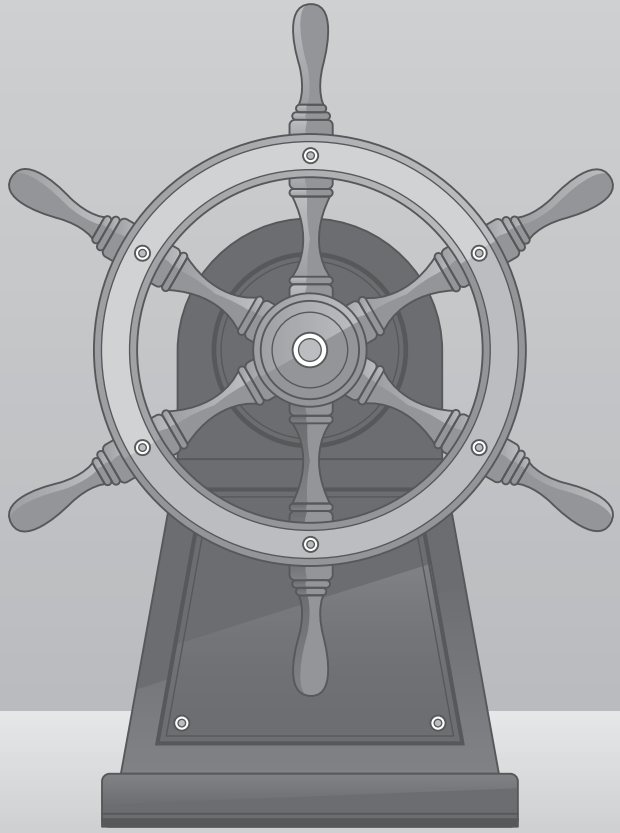
また、物理サーバー上にクラスタを構成する場合は、CPUソケット数に相当するサブスクリプションの選択も可能です。サブスクリプションやサイジングの最新情報および詳細については、以下のガイドを参照してください。

- OpenShift sizing and subscription guide for enterprise Kubernetes
  `https://www.redhat.com/en/resources/openshift-subscription-sizing-guide`

Chapter

2

環境構築



オンプレミス環境やクラウド環境にOpenShiftをインストールする方法は、IPIとUPIの2種類があります。本章ではより簡単にOCPクラスタを作成できるIPIインストールについて解説します。IPIインストールとUPIインストールの違いについては、第6章の「6.1 プラットフォームとインストール方法の選択」を参照してください。

2.1 IPIインストール

本節では、OpenShift 4のインストール方法の一例として、AWS環境にIPIでOpenShift 4.9をインストールする手順を説明します。ここで紹介する構成は、実際に本番環境でも使えるよう、Masterノード・Workerノードともに冗長性があり、複数管理者による管理を意識した構成になっています。

本構成の場合、クラウド費用は1日あたり5,000円程度かかります。そのため、個人で手軽にOpenShiftを試してみるには少々オーバースペックであると感じるかもしれません。より手軽にOpenShiftを試してみたい方は、本章の「2.3 Red Hat CodeReady Containers」を参照してください。

インストールは、次の手順で進めていきます。

1. アカウント作成
   a. Red Hat Customer Portalアカウントの作成
   b. AWSアカウントの作成
2. AWSアカウントの設定
   a. IAMユーザーの作成と権限付与
   b. Amazon Route 53でパブリックホストゾーンを作成
   c. サービスクォータの確認
3. クラスタ作成前の準備
   a. Amazon EC2で踏み台サーバーを作成
   b. AWS CLIのインストール
   c. openshift-installのインストール
   d. Pull-Secretの取得
   e. SSH鍵ペアの作成
   f. クラスタ構成ファイル（install-config.yaml）の作成
4. クラスタ作成
   a. openshift-installでクラスタを作成

5. 補足 クラスタ削除
   a. openshift-installでクラスタを削除

2.1.1 アカウント作成

OpenShift 4のインストールを行うには、Red Hat Customer PortalのアカウントとAWSアカウントが必要です。アカウントを持っていない方は、以下を参考にアカウントを作成してください。

■Red Hat Customer Portalのアカウント作成

Red Hat Customer Portalのアカウントは、OpenShiftのインストーラーやOpenShift CLI、Pull-Secret（クレデンシャル情報）などを取得するときに使用します。次のURLからアカウントを作成してください。

- Red Hat Customer Portal
  `https://access.redhat.com/`

■AWSのアカウント作成

今回はAWS上にOpenShiftをインストールするため、AWSリソースを作成するためのAWSアカウントが必要です。次のサイトを参考にアカウントを作成してください。

- AWSアカウント作成の流れ
  `https://aws.amazon.com/jp/register-flow/`

2.1.2 AWSアカウントの設定

OpenShiftをインストールする前には、AWSアカウントで次の準備が必要です。

■IAMユーザーの作成と権限付与

インストール作業用に`AdministratorAccess`ポリシーがアタッチされたIAMユーザーを作成します。AWSアカウント作成時に`root`ユーザーが作成されるため、この`root`ユーザーを使ってインストール作業を行うこともできます。しかし、`root`ユーザーには強い権限が与えられているため、`root`ユーザーをそのまま使い続けるのはセキュリティの観点から好ましくありません。必要最小限の権限を付与したインストール作業専用のIAMユーザーを作成して、そのIAMユーザーでインストール作業を行うことが推奨されています。

IAMユーザーの作成手順は、次のとおりです。

1. AWSマネジメントコンソールの「IAMダッシュボード」にログインします（**図2.1**）。左側のメニューで［アクセス管理］→［ユーザー］と遷移してから［ユーザーを追加］ボタンをクリックします。

図2.1　IAMダッシュボード：ユーザーを追加

2. ユーザー情報を入力していきます（**図2.2**）。
   - **ユーザー名**：任意（ここでは「ocp-installer」と入力）
   - **アクセスの種類**：［プログラムによるアクセス］をチェック

   ［次のステップ：アクセス権限］ボタンをクリックします。

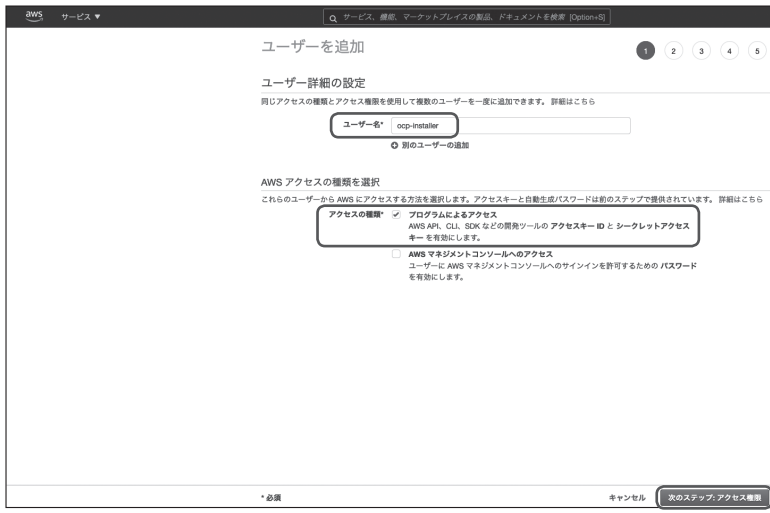


図2.2　IAMダッシュボード：「ユーザー名」と「アクセスの種類」を設定

3. ［アクセス許可の設定］では［既存のポリシーを直接アタッチ］を選択します（**図2.3**）。その下に表示される既存のポリシー一覧の中から［AdministratorAccess］をチェックします。［次のステップ：タグ］ボタンをクリックします。

図2.3　IAMダッシュボード：アクセス許可の設定

4. ［タグの追加（オプション）］では任意のタグを設定できます。この例では何も設定せずに、［次のステップ：確認］ボタンをクリックします（**図2.4**）。

図2.4　IAMダッシュボード：IAMタグの設定

5. 確認を求める画面が表示されたら内容を確認し、[ユーザーの作成] ボタンをクリックします（**図2.5**）。

図2.5　IAMダッシュボード：確認画面

6. ユーザーの作成が完了すると、アクセスキーIDとシークレットアクセスキーが表示されます（**図2.6**）。これ以降、シークレットアクセスキーを確認することはできないため、必ずメモをとるかCSVファイルをダウンロードしてください。

図2.6　IAMダッシュボード：ユーザー追加の成功画面

■Amazon Route 53でパブリックホストゾーンを作成

Amazon Route 53（以下、Route 53）サービスでパブリックホストゾーンを作成します。ここで、クラスタのベースドメインをRoute 53に登録します。

- Creating a public hosted zone
  `https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/CreatingHostedZone.html`

使用するドメインを取得する方法はいくつかあります。今回は、会社や個人で所有しているドメインからOCP用のサブドメインを作成し、そのサブドメインの管理を先ほど作成したIAMユーザーが操作できるRoute 53に委任する方法を紹介します。親ドメインは別のアカウントのRoute 53で管理されていても、別のレジストラ（例：Google Domains）で管理されていても問題はありません。

ここでは例として、親ドメインaoshima.meからOCP用にocpdemo.aoshima.meというサブドメインを作成し、これをパブリックホストゾーンに登録します。

まずは、Route 53のダッシュボードから［ホストゾーン］へ遷移し、［ホストゾーンの作成］をクリックします（**図2.7**）。

- Route 53ダッシュボード
  `https://console.aws.amazon.com/route53/v2/home#Dashboard`

図2.7　Route 53のダッシュボードで［ホストゾーンの作成］を選択

「ホストゾーンの作成」画面に遷移したら、必要な情報を入力していきます（**図2.8**）。ここでは、次のように入力しました。

- **ドメイン名**：使用するドメイン名（ここでは「ocpdemo.aoshima.me」と入力）
- **説明**：ドメインの説明を入力（未入力も可）
- **タイプ**：［パブリックホストゾーン］を選択
- **タグ**：任意

図2.8 「ホストゾーンの作成」画面

必要な情報を入力してから［ホストゾーンの作成］ボタンをクリックします。ホストゾーンの作成が完了すると次の画面に遷移し、ここでNSレコードが確認できます（**図2.9**）。このNSレコードを親ドメイン側で設定します。

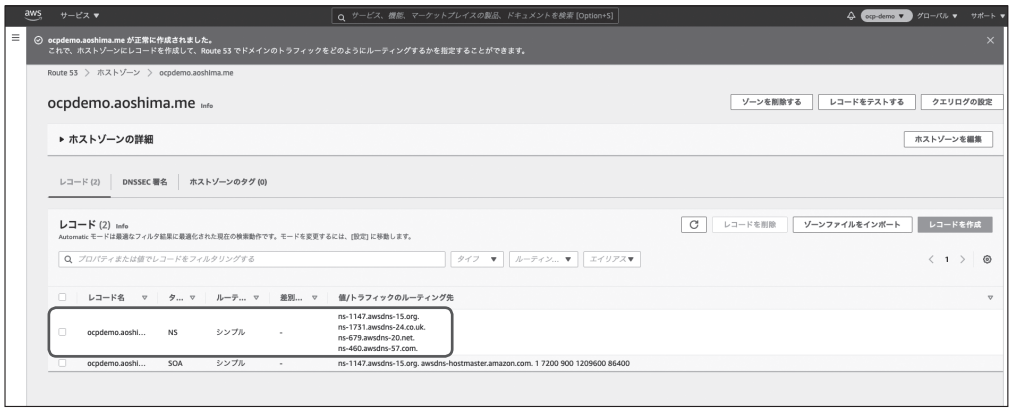


図2.9 パブリックホストゾーンの作成が正常完了した画面

これで、パブリックホストゾーンの作成は完了です。

■サービスクォータの確認

AWSアカウントは、作成可能なコンポーネント数の上限がサービスごとに設定されています。OpenShift Container Platform（以下、OCP）は多数のAWSコンポーネントを使用するため、クラスタの数やサイズによっては制限に引っかかってインストールに失敗することがあります。そのため、クラスタの作成前には現在のサービスクォータの設定とクラスタが必要とするコンポーネント数を確認しておく必要があります。

必要なコンポーネント数がサービスクォータの設定値を上回っていた場合は、AWSコンソールからクォータの引き上げリクエストを送ります。また、サービスクォータはリージョンごとに存在するので、OCPをインストールするリージョンそれぞれについて設定を確認するようにしておきます。

OCPクラスタのインストールおよび実行機能に影響を与える可能性のあるAWSコンポーネントの制限は、以下のオンラインドキュメントにまとめられています。

- AWSアカウントの制限
  `https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/installing/installation-aws-limits_installing-aws-account`

サービスクォータの設定状況は、AWSコンソールの「Service Quotasのダッシュボード」で確認できます（**図2.10**）。

- Service Quotasのダッシュボード（東京リージョン）
  `https://ap-northeast-1.console.aws.amazon.com/servicequotas/home`

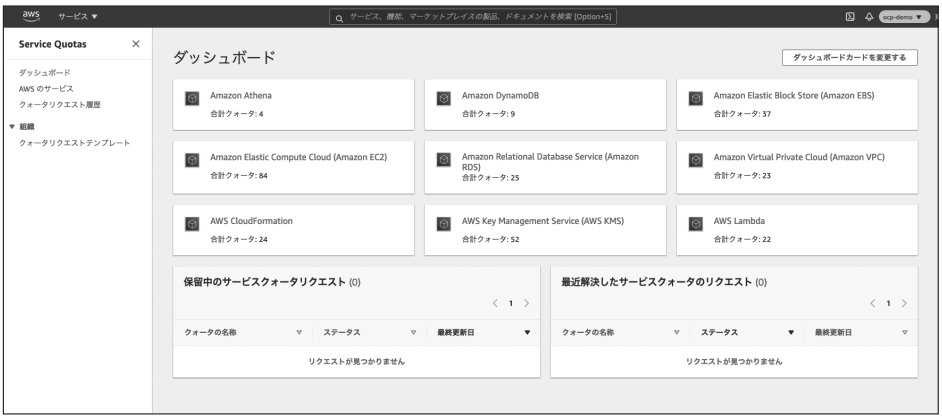


図2.10　Service Quotasのダッシュボード

なお、本書執筆時点（2021年11月）では、デフォルトのサービスクォータ設定でも、デフォルトサイズのOCPクラスタ1台（Masterノード3台+Workerノード3台）をインストールできます。

2.1.3 クラスタ作成前の準備

■Amazon EC2で踏み台サーバーを作成

本章では、Amazon Elastic Compute Cloud（以下、Amazon EC2）で踏み台サーバーを作成し、踏み台サーバーからopenshift-install（OpenShiftインストーラー）を実行する方法を紹介します（**図2.11**）。

openshift-installは、LinuxとmacOSで実行可能なため、踏み台サーバーを使わずに手元のPCにopenshift-installをダウンロードして実行することも可能です（**図2.12**）。ただし、複数人でクラスタを管理する場合は、踏み台サーバーを1つ用意しておくと、次のような便利な点があります。

1. openshift-installの実行環境を管理者がそれぞれ用意する必要がない。
2. インストール設定ファイルやtfstateファイル[1]、インストールログファイルなどを開発者間で共有する仕組みを別途用意しなくてもいい。

本章では踏み台サーバーがある前提で説明をしますが、踏み台サーバーではなく自分の手元のPCから操作する場合は、今後踏み台サーバーで行うセットアップ作業を手元のPC上で実施してください。

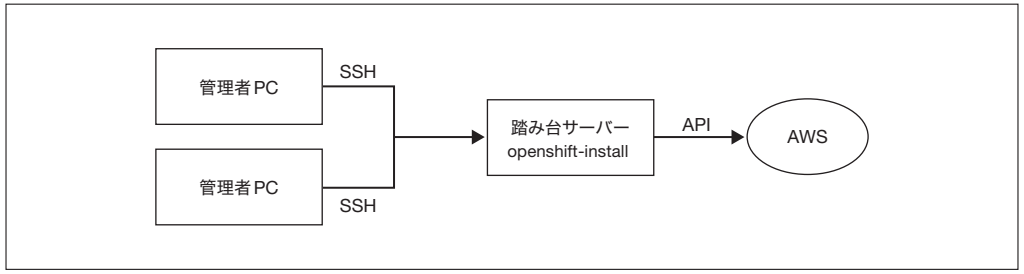


図2.11　本章で紹介する構成（特に管理者が複数人いる場合は、こちらがおすすめ）

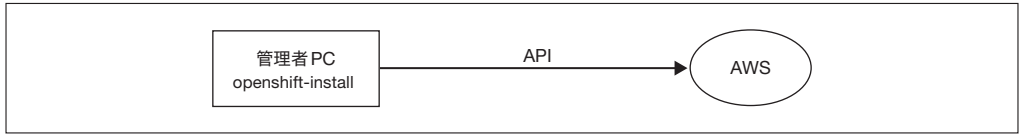


図2.12　踏み台サーバーを使わない構成（個人で管理する場合はこの構成も可能）

【1】 インストール時に生成されるterraform.tfstateファイル（2.1.5項）です。

本書の検証では、AMIは「Red Hat Enterprise Linux 8 (HVM), SSD Volume Type - ami-0bccc42bba4dedac1 (64ビットx86)」を使用し、インスタンスは「t2.micro」、そして、SSH接続できるようなセキュリティグループを設定しました。

踏み台サーバーは、必ずしもRed Hat Enterprise Linuxである必要はありません。openshift-installが実行可能であればいいので、他のLinuxやmacOSでも代替可能です。また、インターネットに疎通できてAWSのAPIを利用できるのであれば、必ずしもAWS上に作成する必要もありません。

■AWS CLIのインストール

踏み台サーバーにAWS Command Line Interface（CLI）をインストールします。踏み台サーバーがRHEL以外の場合は次のサイトを参照してください。

- AWS CLIバージョン2のインストール、更新、アンインストール
  https://docs.aws.amazon.com/ja_jp/cli/latest/userguide/install-cliv2.html

unzipコマンドが必要なため、インストールされていない場合は事前にインストールしてください（sudo yum install unzipを実行します）。

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
$ unzip awscliv2.zip
$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
```

AWS CLIがインストールできたことをaws --versionコマンドで確認します。以下のようにAWS CLIのバージョンが表示されれば成功です。

```
$ aws --version
aws-cli/2.3.4 Python/3.8.8 Linux/4.18.0-305.el8.x86_64 exe/x86_64.rhel.8 prompt/off
```

次に、先ほど作成したIAMユーザー（ocp-installer）の情報をaws configureコマンドで登録します。

▶ 入力例

```
$ aws configure
AWS Access Key ID [None]: AKIASN7WUQMYASAKDH4W
AWS Secret Access Key [None]: ◇XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX◇
Default region name [None]: ap-northeast-1
Default output format [None]:
```

認証情報ファイルが正常に認識されているかどうかは、次のコマンドで確認することができます。

```
$ aws sts get-caller-identity
{
    "UserId": "AIDASN7WUQMYMXTYPHVM3",
    "Account": "167484097328",
    "Arn": "arn:aws:iam::167484097328:user/ocp-installer"
}
```

■openshift-installのインストール

openshift-install（OpenShiftインストーラー）は、下記のRed Hat OpenShift Cluster Managerサイトからダウンロードしてください（**図2.13**）。openshift-installのバージョンによって、インストールされるOpenShiftクラスタのバージョンが決まります。

Red Hat OpenShift Cluster Managerサイトで配布されているopenshift-installは最新バージョンなので、旧バージョンのOpenShiftクラスタをインストールしたい場合は、対応するopenshift-installを下記URLからダウンロードしてください。

- 最新安定バージョンのOpenShift 4をインストールしたい場合：Red Hat OpenShift Cluster Managerサイト
  https://cloud.redhat.com/openshift/install/aws/installer-provisioned
- 特定のバージョンのOpenShift 4をインストールしたい場合：openshift-installバイナリ一覧
  https://mirror.openshift.com/pub/openshift-v4/clients/ocp/

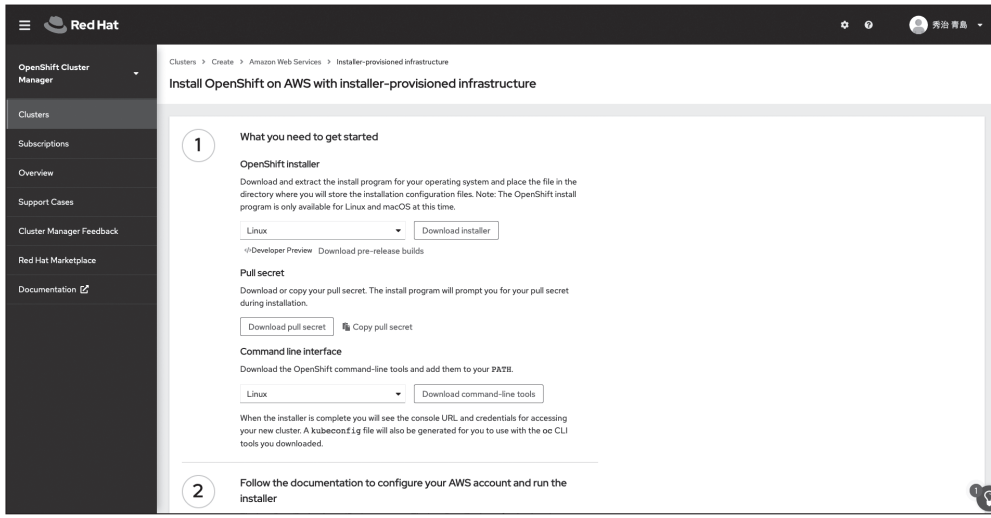


図2.13　Red Hat OpenShift Cluster Managerサイト

今回は例としてhttps://mirror.openshift.com/pub/openshift-v4/clients/ocp/から執筆時（2021年11月）の最新安定バージョンの「4.9.4」をインストールします。

まずは、openshift-installをダウンロードします。

```
$ curl -O https://mirror.openshift.com/pub/openshift-v4/clients/ocp/stable/openshift-install-linux.tar.gz
```

圧縮ファイルを展開します。

```
$ tar xvf openshift-install-linux.tar.gz
```

パスが通っているディレクトリ、たとえば「/usr/local/bin」などにopenshift-installを移動します。

```
$ sudo mv ./openshift-install /usr/local/bin
```

最後に、パスが通っているかどうかを確認します。次のような出力が返ってくれば、正常にパスが通っています。

```
$ openshift-install version
openshift-install 4.9.5
built from commit 8223216bdcef5de56b52240ab7160ca909a9e56c
release image quay.io/openshift-release-dev/ocp-release@sha256:386f4e08c48d01e0c73d294a88bb64fac3284d1d16a5b8938deb3b8699825a88
release architecture amd64
```

■Pull-Secretの取得

Pull-Secret（クレデンシャル情報）は、openshift-installでクラスタをインストールするときに入力を要求されます。事前に取得し、安全な場所で保管しておいてください。Pull-Secretは、openshift-installと同じく、Red Hat OpenShift Cluster Managerサイトで取得できます。

- Red Hat OpenShift Cluster Managerサイト
  https://cloud.redhat.com/openshift/install/aws/installer-provisioned

■SSH鍵ペア作成

MasterノードにSSH接続するためのSSH鍵ペアを作成します。クラスタ作成時（次ページの「構成ファイルの作成」）に、ここで作成したSSH鍵の公開鍵を登録します。ここで登録したSSH鍵は、

インストール失敗時にBootstrapノード[2]にSSH接続して原因調査をしたり、クラスタインストール後も障害が発生したときにMasterノードにSSH接続して対応する場合などに使用することがあるため、クラスタインストール後も大切に保管してください。

```
$ ssh-keygen -t ed25519 -N '<パスフレーズ>' -f <パス>/<ファイル名>
```

▶登録例

```
$ ssh-keygen -t ed25519 -N 'passphrase' -f ~/.ssh/id_ed25519
```

次に、パスフレーズ入力をスキップできるように、秘密鍵とパスフレーズをssh-agentに登録します。パスフレーズを空にした場合は、この作業は必要ありません。

まずは、ssh-agentプロセスをバックグラウンドタスクとして起動します。

```
$ eval "$(ssh-agent -s)"
```

そして、ssh-agentに公開鍵を登録します。

```
$ ssh-add <パス>/<ファイル名>
```

▶登録例

```
$ ssh-add ~/.ssh/id_ed25519
Enter passphrase for /home/ec2-user/.ssh/id_ed25519:
Identity added: /home/ec2-user/.ssh/id_ed25519 (ec2-user@ip-172-31-2-15.ap-northeast-1.compute.➥
internal)
```

鍵が登録されているかどうかを確認します。

```
$ ssh-add -L
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBSGpbxzp7b/jRyJCMzQDB7kZyH19LBziyV1R9pKYfj/ ec2-user@ip-172➥
-31-2-15.ap-northeast-1.compute.internal
```

■構成ファイルの作成

構成ファイル（install-config.yaml）を作成します。ここで、どのような構成でクラスタをインストールするか決めます。

最初に、作業用ディレクトリを作成し、移動します（作業ディレクトリ名は任意でかまいません）。

【2】 Bootstrapノードとは、クラスタインストール時に一時的に作成されるVM（Virtual Machine：仮想マシン）のことです。

```
$ mkdir openshift
$ cd openshift
```

まずは、openshift-install create install-configを実行してinstall-config.yamlファイルを作成します。次の情報を対話形式で入力するので、事前に用意しておきましょう。

- SSH Public Key：<「2.1.3 クラスタ作成前の準備」で作成したSSH公開鍵のパス>
- Platform：AWS
- Region：<クラスタを作成するリージョン>（対応リージョンは以下）
  https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/installing/installation-aws-regions_installing-aws-account
- Base Domain：<「2.1.2 AWSアカウントの設定」でRoute53に登録したドメイン>
- Cluster Name：<クラスタ名>
- Pull Secret：<「2.1.3 クラスタ作成前の準備」で取得したPull-Secret>

install-config.yamlファイルを作成

```
$ openshift-install create install-config
? SSH Public Key /home/ec2-user/.ssh/id_ed25519.pub
? Platform aws
INFO Credentials loaded from the "default" profile in file "/home/ec2-user/.aws/credentials"
? Region ap-northeast-1
? Base Domain ocpdemo.aoshima.me
? Cluster Name democluster
? Pull Secret [? for help]
```

次のようなinstall-config.yamlファイルが作成されます。

```
$ cat ./install-config.yaml
apiVersion: v1
baseDomain: ocpdemo.aoshima.me
compute:
- architecture: amd64
hyperthreading: Enabled
name: worker
platform: {}
replicas: 3
controlPlane:
architecture: amd64
hyperthreading: Enabled
name: master
platform: {}
replicas: 3
metadata:
creationTimestamp: null
```

```
name: democluster
networking:
clusterNetwork:
- cidr: 10.128.0.0/14
hostPrefix: 23
machineNetwork:
- cidr: 10.0.0.0/16
networkType: OpenShiftSDN
serviceNetwork:
- 172.30.0.0/16
platform:
aws:
region: ap-northeast-1
publish: External
pullSecret: '{"auths":{"cloud.openshift.com":{"auth":"XXXXXXXXXX","email":"XXXXXXXXXX"}}}'
sshKey: |
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINZhOovlnefrj5jo4NdPzqHu5hg0kAHNotcN04PwE27B ec2-user@ip-172-
31-32-111.ap-northeast-1.compute.internal
```

このinstall-config.yamlをそのまま使ってクラスタを作成することもできますが、今回は少し構成を変更してみます。

デフォルトではm4.xlargeのMasterノード3台とm4.largeのWorkerノード3台の構成となっていますが、WorkerノードのEC2インスタンスタイプをもう少し大きなm5.xlargeに変更します。次のように、install-config.yamlの「compute」部分（Workerノードの構成部分）を書き換えます。

```
...略...
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  platform:
    aws:
      type: m5.xlarge
  replicas: 3
...略...
```

openshift-install create cluster（OpenShiftクラスタを構築するコマンド）を実行すると、作成したinstall-config.yamlは自動的に削除されます。そのため、インストール前にバックアップをとっておきます。

```
$ cp install-config.yaml install-config.yaml.bak
```

2.1.4 クラスタ作成

■openshift-installでクラスタを作成

クラスタを構築するコマンドを実行します。完了まで30～40分ほどかかります。

```
$ openshift-install create cluster
INFO Credentials loaded from the "default" profile in file "/home/ec2-user/.aws/credentials"
INFO Consuming Install Config from target directory
INFO Creating infrastructure resources...
INFO Waiting up to 20m0s for the Kubernetes API at https://api.democluster.ocpdemo.aoshima.me:644
3...
INFO API v1.22.0-rc.0+a44d0f0 up
INFO Waiting up to 30m0s for bootstrapping to complete...
INFO Destroying the bootstrap resources...
INFO Waiting up to 40m0s for the cluster at https://api.democluster.ocpdemo.aoshima.me:6443 to in
itialize...
INFO Waiting up to 10m0s for the openshift-console route to be created...
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/home
/ec2-user/openshift/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.democluster.oc
pdemo.aoshima.me
INFO Login to the console with user: "kubeadmin", and password: "IoAzZ-5QrAY-LJtqr-3KqGe"
INFO Time elapsed: 40m47s
```

以上で、インストール作業は完了です。AWS上には**図2.14**のようなリソースが構成されます[3]。

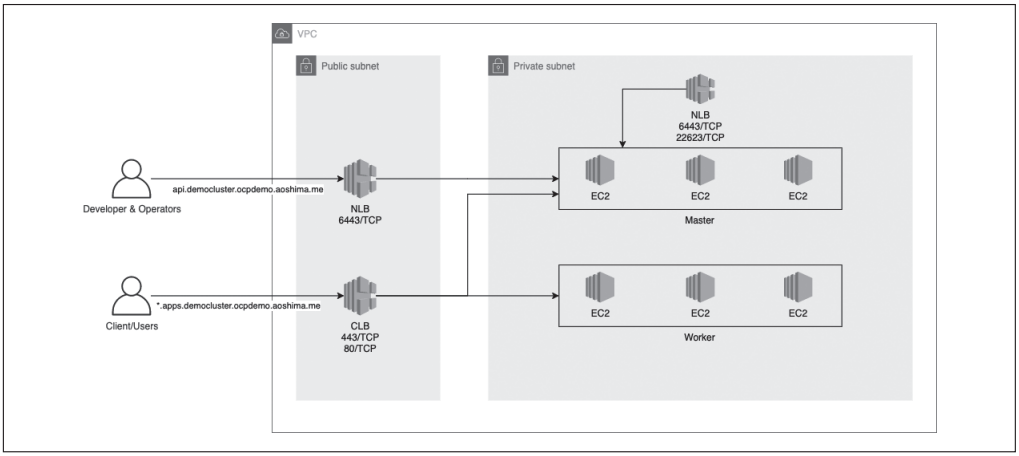


図2.14 インストール直後のAWS構成概要（Amazon EC2とELBのみ）

【3】 EC2とELB（Elastic Load Balancing）以外にも多くのリソースが作成されますが、**図2.14**では省略しています。

意図したバージョンのクラスタが正常にインストールができているかどうかは、カスタムリソースのclusterversionを見ることでも確認することができます。詳細は、次章の「3.2.3 ocコマンドの使い方」を参照してください。

また、詳細なインストールログは、コマンド実行ディレクトリの.openshift_install.logファイルに保存されます。インストールに失敗したときは、このファイルを調査してください。

インストール直後はkubeadminというクラスタ管理者権限を持つユーザーが作成されています。このkubeadminユーザーのパスワードやWebコンソールのURLは、標準出力に記載されているのでメモしておきましょう[4]。ログイン方法の詳細については次章で紹介します。

2.1.5 補足 クラスタ削除

■openshift-installでクラスタを削除

クラスタの削除には、インストール時に生成されたterraform.tfstateファイルが必要です。インストール作業をしたディレクトリにterraform.tfstateが作成されるため、同じディレクトリで次のコマンドを実行してください。

```
$ cd $HOME/openshift
$ openshift-install destroy cluster
```

2.2 サーバー証明書の設置

OpenShiftには、コンテナアプリケーションをクラスタ外部に公開する「Router」という機能が備わっています。ユーザーがOpenShiftにデプロイしたコンテナアプリケーションだけではなく、OpenShiftのWebコンソールなどもRouterを介して外部公開しています。

このRouterに対してサーバー証明書を設置できますが、インストール直後は自己署名証明書を利用しているため、OpenShift上のアプリケーションにアクセスするとブラウザに警告が表示されます(図2.15)。自己署名証明書でも暗号化通信はできますが、実在性証明ができません。必ず、信頼できる認証局が発行した証明書を設置するようにしてください。

【4】 標準出力の結果をメモし忘れた場合でも、.openshift_install.logにログイン情報が記載されています。

図2.15　自己署名証明書を利用している場合の警告

本節では、誰でも無料で使える「ZeroSSL」でSSLサーバー証明書を発行し、OpenShiftに設置するまでの手順を紹介します。他の認証局を使用する場合でも、証明書発行以外の手順は同じです。

まずは、証明書の発行手順から説明していきます。

2.2.1　acme.shでサーバー証明書を発行

acme.shはACME[5]プロトコルのクライアントツールです。SSL証明書を簡単に発行することができます。acme.shは複数の認証局に対応していますが、本例ではデフォルト認証局であるZeroSSLを利用します。また、以下の手順で作成するZeroSSLのSSL証明書は無料です。クレジットカードの登録なども必要ありません[6]。

- acme.sh
  `https://github.com/acmesh-official/acme.sh`

本項の作業は、作業端末（踏み台サーバー）からOpenShiftクラスタに`cluster-admin`権限を持つユーザーでCLIログインした状態で実行してください。インストール時に自動作成された`kubeadmin`ユーザーは、`cluster-admin`権限を持っています。

CLIでのログイン方法については、第3章の「3.2　CLI（ocコマンド）」を参照してください。

【5】　ACMEとは Automatic Certificate Management Environment の略で、証明書発行プロトコルのことです。

【6】　`https://zerossl.com/features/acme/`

まずはacme.shをインストールします。コマンドのemailの部分は自分のものに書き換えてください。後でZeroSSLの登録に使用します。

```
$ curl https://get.acme.sh | sh -s email=my@example.com
```

インストールログに「OK, Close and reopen your terminal to start using acme.sh」と表示されるので一度ターミナルにログインし直します。

続いて次の4つの環境変数をセットします。

- AWS_ACCESS_KEY_ID：AWSのアクセスキーID（OCPクラスタのインストールに使用したIAMユーザーopenshift-installerのID[7]）
- AWS_SECRET_ACCESS_KEY：AWSのシークレットキー
- LE_API：完全修飾ドメイン
- LE_WILDCARD：ワイルドカードドメイン

```
$ export AWS_ACCESS_KEY_ID="..."

$ export AWS_SECRET_ACCESS_KEY="..."

$ export LE_API=$(oc whoami --show-server | cut -f 2 -d ':' | cut -f 3 -d '/' | sed 's/-api././')
$ echo $LE_API
api.democluster.ocpdemo.aoshima.me

$ export LE_WILDCARD=$(oc get ingresscontroller default -n openshift-ingress-operator -o jsonpath
='{.status.domain}')
$ echo $LE_WILDCARD
apps.democluster.ocpdemo.aoshima.me
```

必要な環境変数をセットできたら証明書を発行します。

```
$ acme.sh --issue --dns dns_aws -d ${LE_API} -d *.${LE_WILDCARD}
```

もしここで「AWS Route53 rate exceeded」エラーが発生した場合は、環境変数AWS_DNS_SLOWRATEでAPIのリクエスト間隔を調整することができます[8]。

証明書の発行に成功するとacme.shのデフォルトパス（~/.acme.sh）以下に証明書ファイルが生成されます。管理しやすくするために次のコマンドで別ディレクトリにも出力します。

【7】 OCPクラスタのインストールに使用したIAMユーザーでなくても、次のリンクの操作権限を持つIAMユーザーであれば可能です。
https://github.com/acmesh-official/acme.sh/wiki/How-to-use-Amazon-Route53-API

【8】 https://github.com/acmesh-official/acme.sh/wiki/dnsapi#10-use-amazon-route53-domain-api

```
$ export CERTDIR=$HOME/certificates
$ mkdir -p ${CERTDIR}

$ acme.sh --install-cert -d ${LE_API} -d *.${LE_WILDCARD} --cert-file ${CERTDIR}/cert.pem --key-f
ile ${CERTDIR}/key.pem --fullchain-file ${CERTDIR}/fullchain.pem --ca-file ${CERTDIR}/ca.cer
```


2.2.2 Routerに証明書を設置

前項で取得したサーバー証明書（または別途用意したサーバー証明書）をRouterに設置していきます。

本項の作業は、作業端末（踏み台サーバー）からOpenShiftクラスタにcluster-admin権限を持つユーザーでCLIログインした状態で実行してください。インストール時に自動作成されたkubeadminユーザーは、cluster-admin権限を持っています。CLIでのログイン方法については、「3.2 CLI（ocコマンド）」を参照してください。

まずは証明書からSecretを作成します。openshift-ingressプロジェクトにrouter-certsという名前でSecretを作成します。

```
$ oc create secret tls router-certs --cert=${CERTDIR}/fullchain.pem --key=${CERTDIR}/key.pem -n
openshift-ingress
```

次に、router-certsをデフォルトの証明書として使用するようにRouterの設定を書き換えます。Routerの設定は、openshift-ingress-operatorプロジェクトにあるdefaultという名前のIngress Controllerオブジェクトで行います。

```
$ oc patch ingresscontroller default -n openshift-ingress-operator --type=merge --patch='{"spec":
 { "defaultCertificate": { "name": "router-certs" }}}'
```

以上で、設定作業は終了です。

IngressControllerオブジェクトを更新すると、Routerは自動で再デプロイされて新しい証明書の設定が反映されます。この設定により、Webコンソールなどに適切な証明書を使って、外部からアクセスできるようになります。

Routerの再デプロイが完了するまで数分程度かかるので、少し待ってからWebコンソールをブラウザで開いて証明書が適切に設置されているか確認してみてください。

- [参考] Requesting and installing Let's Encrypt Certificates for OpenShift 4
  https://github.com/redhat-cop/openshift-4-alpha-enablement/blob/master/Lets_Encrypt_Certificates_for_OCP4.adoc#installing

2.3 Red Hat CodeReady Containers環境

Red Hat CodeReady Containers（以下、CRC）は、ローカルマシンの仮想マシン（VM）上で動くミニマルなOpenShift 4環境です。

- Red Hat CodeReady Containers
  https://developers.redhat.com/products/codeready-containers/overview

通常のOpenShiftと比較すると、次のような違いがあります。

- シングルノードで動作する。つまりMasterノードもWorkerノードも同じ1つのVM上で動く
- アップグレードはできない。最新のバージョンを使いたい場合は既存のCRCを削除して、新しいCRCを再インストールする（後述の「2.3.3 アップグレード手順」を参照）
- デフォルトではMonitoringオペレーターが無効になっている。このため、Webコンソールの「Monitoring」機能も使えない
- 一部の設定が変更できない
  - クラスタのベースドメイン。crc.testingで固定
  - クラスタの内部通信に利用するアドレス範囲

CRCはあくまでもテストや開発向けの環境です。本番環境として利用する場合や複数人で利用する場合は通常のOpenShiftを用意してください。

2.3.1 システム要件

システム要件はバージョンによって異なるため、インストールしたいCRCのドキュメントを確認してください。以下はバージョン1.34の例です。

- Red Hat CodeReady Containers 1.34のシステム要件
  https://access.redhat.com/documentation/en-us/red_hat_codeready_containers/1.34/html/getting_started_guide/installation_gsg#minimum-system-requirements_gsg

■ハードウェア要件

最低でも以下のリソースが必要です（バージョン1.34の場合）。CRC上で動かすワークロードによってはさらに多くのリソースが必要になります。

- CPU：4 physical CPU cores
- メモリ：9 GB
- ストレージ：35 GB

■OS要件

次のいずれかのOSが必要です。また、CRCのバージョンごとにOSの対応バージョンも異なります。詳細は対応するCRCのバージョンのシステム要件を参照してください。

- Windows（対応OSバージョンはCRCの各バージョンごとに要確認）
- macOS（対応OSバージョンはCRCの各バージョンごとに要確認）
- Linux
  - Red Hat Enterprise Linux、CentOS、Fedoraは公式対応（対応OSバージョンはCRCの各バージョンごとに要確認）
  - UbuntuとDebianは公式対応はしていないため一部手動でセットアップが必要なこともある

2.3.2 インストール手順

まずはCRCのバイナリまたはインストーラーをダウンロードしてください。

- 最新バージョンのCRCをインストールする場合：Create an OpenShift cluster (local)
  `https://cloud.redhat.com/openshift/create/local`
- 特定のバージョンのCRCをインストールする場合：Index of /pub/openshift-v4/clients/crc
  `https://mirror.openshift.com/pub/openshift-v4/clients/crc/`

特に特定のバージョンのCRCをインストールする場合は、バイナリやインストーラーと一緒に配布されている`release-info.json`ファイルを確認して、対象のCRCがどのバージョンのOpenShiftに対応しているのか確認しておくとよいでしょう（**図2.16**）。

図2.16　release-info.jsonの例（CRCと対応しているOpenShiftのバージョンが確認できる）

ちなみにCRCのインストール後は、crc versionコマンドでCRC自身のバージョンと、対応しているOpenShiftのバージョンを確認することができます。

本書ではmacOSに最新のCRCをインストールする方法を紹介します。

まずは、https://console.redhat.com/openshift/create/localからmacOS向けのCRCインストーラーをダウンロードします（**図2.17**）。

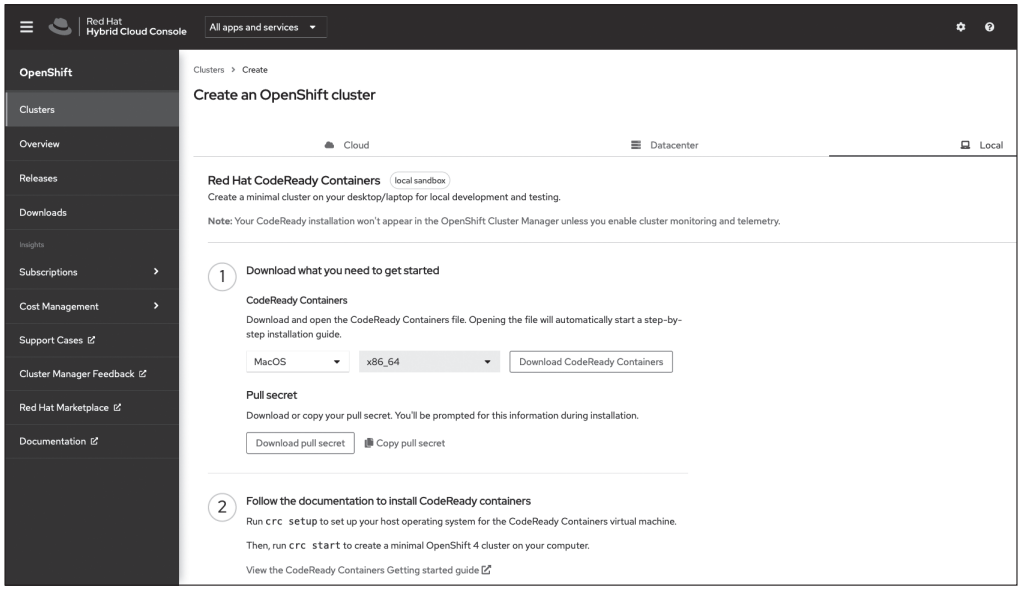


図2.17　インストーラーとPull secretの配布ページ

インストーラー（pkgファイル）のダウンロードが完了したら、ファイルをダブルクリックしてインストーラーを起動します（**図2.18**）。

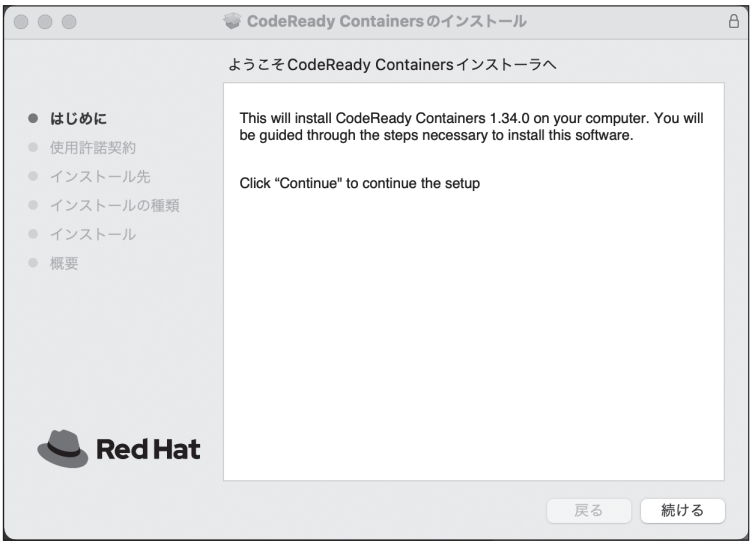


図2.18 CodeReady Containersインストーラーの起動画面

あとはインストーラーの指示に従ってインストール作業を進めます。
インストールが完了したら、インストールしたCRCのバージョンを確認します。

```
$ crc version
CodeReady Containers version: 1.34.0+34c31851
OpenShift version: 4.9.0 (bundle installed at /Applications/CodeReady Containers.app/Contents/Res➥
ources/crc_hyperkit_4.9.0.crcbundle)
```

次に初期設定を行います。

```
$ crc setup
CodeReady Containers is constantly improving and we would like to know more about usage (more det➥
ails at https://developers.redhat.com/article/tool-data-collection)
Your preference can be changed manually if desired using 'crc config set consent-telemetry <yes/n➥
o>'
Would you like to contribute anonymous usage statistics? [y/N]: y
Thanks for helping us! You can disable telemetry with the command 'crc config set consent-telemet➥
ry no'.
INFO Checking if running as non-root
INFO Checking if crc-admin-helper executable is cached
INFO Checking for obsolete admin-helper executable
INFO Checking if running on a supported CPU architecture
INFO Checking minimum RAM requirements
INFO Checking if running emulated on a M1 CPU
INFO Checking if HyperKit is installed
INFO Checking if qcow-tool is installed
```

```
INFO Checking if crc-driver-hyperkit is installed
INFO Checking if CodeReady Containers daemon is running
INFO Checking if launchd configuration for tray exists
INFO Check if CodeReady Containers tray is running
INFO Starting CodeReady Containers tray
INFO Checking if CRC bundle is extracted in '$HOME/.crc'
INFO Checking if /Applications/CodeReady Containers.app/Contents/Resources/crc_hyperkit_4.9.0.crc
bundle exists
INFO Extracting bundle from the CRC executable
INFO Ensuring directory /Applications/CodeReady Containers.app/Contents/Resources exists
INFO Uncompressing crc_hyperkit_4.9.0.crcbundle
crc.qcow2: 11.50 GiB / 11.50 GiB [-----------------------------------------------------------------
-----------------------------------------------------------------] 100.00%
Your system is correctly setup for using CodeReady Containers, you can now run 'crc start' to sta
rt the OpenShift cluster
```

以上で、CRCのインストールと初期セットアップは完了です。

2.3.3 アップグレード手順

CRCのアップグレードは以下の手順で手動で実施します。まずはCRCが稼働している仮想マシンを削除します。この操作で古いVMに保存されていたデータは削除されます。

```
$ crc delete
```

次に前項と同じ手順で新しいCRCをインストールしてください。この操作で古いCRCが新しいCRCに置き換えられます。`crc setup`コマンドまで忘れずに実施してください。

2.3.4 使い方

CRCのクラスタを稼働させるVMの起動・停止・削除方法およびクラスタへの接続方法を説明します。

■VMの起動方法

次のコマンドでVMを起動します。このとき、Pull-Secret（クレデンシャル情報）を尋ねられるので、事前に用意しておきましょう。クレデンシャル情報は、CRCのバイナリをダウンロードしたときと同じ下記URLで取得できます。

- Create an OpenShift cluster (local)
  https://cloud.redhat.com/openshift/create/local

```
$ crc start
...
Started the OpenShift cluster.

The server is accessible via web console at:
https://console-openshift-console.apps-crc.testing

Log in as administrator:
Username: kubeadmin
Password: vtAoY-DcVx7-PEjMY-bCXZI

Log in as user:
Username: developer
Password: developer

Use the 'oc' command line interface:
$ eval $(crc oc-env)
$ oc login -u developer https://api.crc.testing:6443
```


VMの起動が完了すると、ログイン情報が標準出力に表示されます。上記の出力例を見てわかるように、kubeadminとdeveloperの2つのアカウントが作成されています。この内容に従ってクラスタにログインしてください。

■クラスタへの接続方法

ログイン方法は、通常のOpenShiftの場合と同じです。第3章「基本的な操作方法」の内容を参考に、クラスタ起動時に出力されたログイン情報を使ってログインしてください。

CRCには、ocバイナリが同梱されています。ocがインストールされていない環境や、CRCのクラスタバージョンに対してインストール済みのocのバージョンが古くて使えない場合などは、次のようにCodeReady Containers付属のocにパスを通して使用してください。

```
$ eval $(crc oc-env)
```

CRC付属のocにパスが通っているかどうかは、次のようにして確認することができます。

```
$ which oc
/Users/aoshima/.crc/bin/oc/oc
```

また、Webコンソールからログインする場合は、crc consoleコマンドを実行してください。自身のPCのデフォルトブラウザでWebコンソールが開きます。kubeadminかdeveloperアカウントを使ってログインしてください[9]（図2.19）。

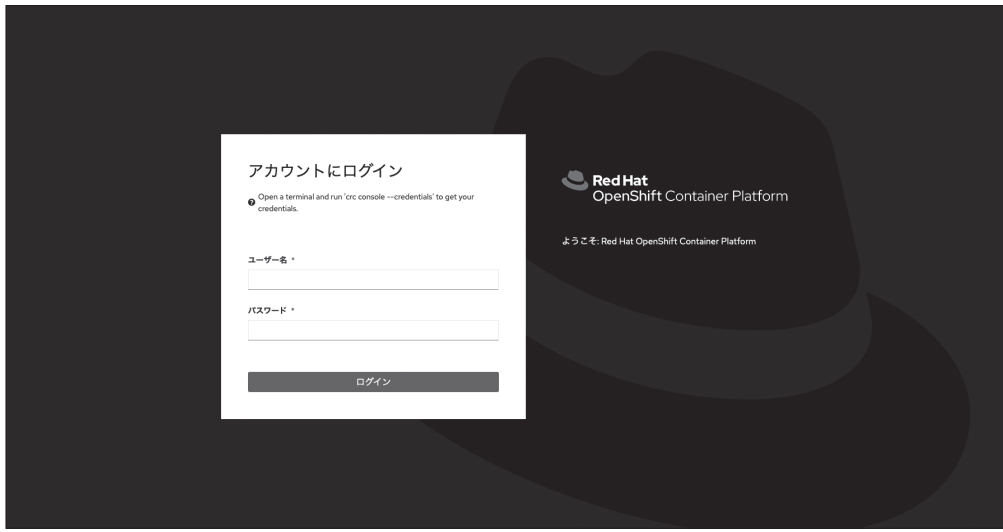


図2.19　CodeReady ContainersのWebコンソール（ログイン画面）

■VMの停止方法

CRCのクラスタを停止させるには、次のコマンドで該当のVMを停止させます。停止しても、VM上のデータは削除されません。

```
$ crc stop
```

■VMの削除方法

次のコマンドで、CRCのVMを削除することができます。この操作により、VM上のデータはすべて削除されます。ただし、~/.crc以下に残っているキャッシュデータなどは消えないため、キャッシュデータも含めて完全に削除する場合は、このディレクトリも削除します。

```
$ crc delete
```

[9]　このログイン場所の違いは、アイデンティティプロバイダーの違いによるものです。アイデンティティプロバイダーについては、第4章「ユーザー管理」で説明します。

OpenShift の学習コンテンツ

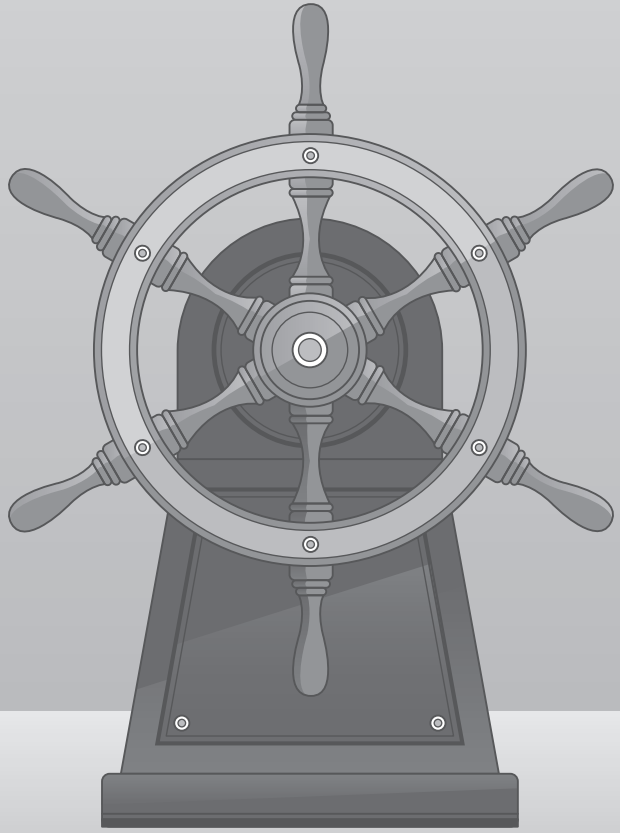
Red Hat Developer（https://developers.redhat.com/）では、ハンズオン形式でOpenShiftを学ぶことができる学習コンテンツや、OpenShiftのサンドボックス環境などを無料で提供しています。OpenShift環境を自前で用意しなくても、手軽にOpenShiftやその関連プロダクトに触れることができるところがうれしいです。

- OpenShiftの学習コンテンツ
  https://developers.redhat.com/learn
- OpenShiftのサンドボックス環境
  https://developers.redhat.com/developer-sandbox

Chapter

3

基本的な操作方法



本章では、OpenShift Container Platform（以下、OCP）の基本的な使い方を紹介します。OCPを利用するには、Webコンソールかコマンドラインインターフェース（Command Line Interface：CLI）を使います。

3.1 Webコンソール

Webコンソールは、Webブラウザからアクセスできるユーザーインターフェースです。コンテンツを視覚的に把握・管理することができます。

WebコンソールのURLは、インストールログの最後にユーザー名、パスワードとともに次のように表示されています。それぞれ太字で表記しています。

```
$ openshift-install create cluster
…
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/home
/ec2-user/openshift/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.democluster.oc
pdemo.aoshima.me
INFO Login to the console with user: "kubeadmin", and password: "SPFiM-bINKC-8HRSd-VAApM"
INFO Time elapsed: 34m43s
```

既存のクラスタを利用するなど、インストールログが確認できない場合は、ログインに必要な情報を管理者等から入手してください。

3.1.1 ログイン

ブラウザでWebコンソールのURLを開くと、**図3.1**のようなログイン画面が表示されます。ユーザー名とパスワードを入力し、［ログイン］ボタンをクリックしてログインします。

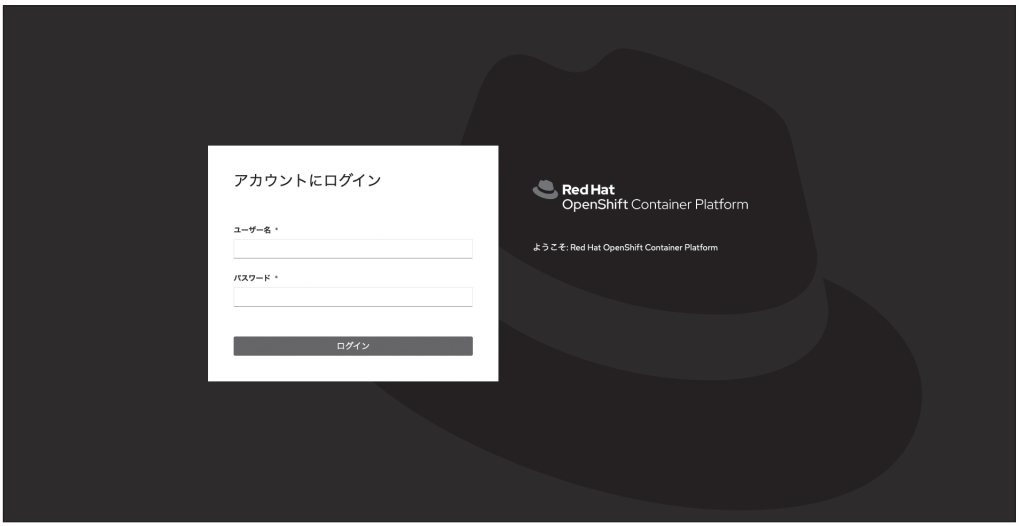


図3.1　ログイン画面

ログインに成功すると、**図3.2**のような画面が表示されます。

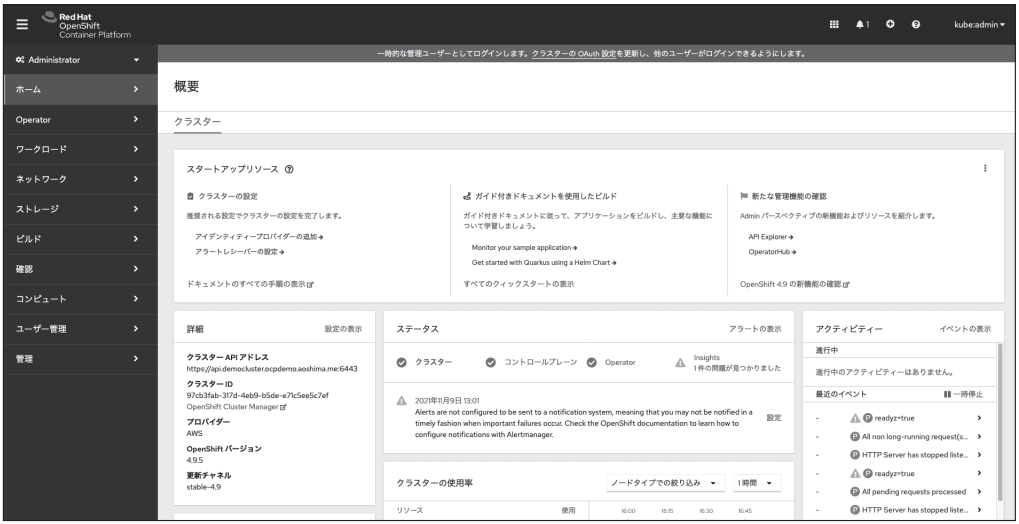


図3.2　Webコンソール（Administrator画面）

3.1.2 使い方

左側のサイドバー（図3.3）から確認・操作したい項目を選ぶと、その画面に移ります。

図3.3　Administrator画面のサイドバー

Webコンソールは、Administrator用（前掲図3.2）とDeveloper用（図3.4）の2つの視点（パースペクティブ）を提供しています。サイドバー上部の選択欄で「Administrator」または「Developer」を選ぶことで、パースペクティブを切り替えられます（図3.5）。

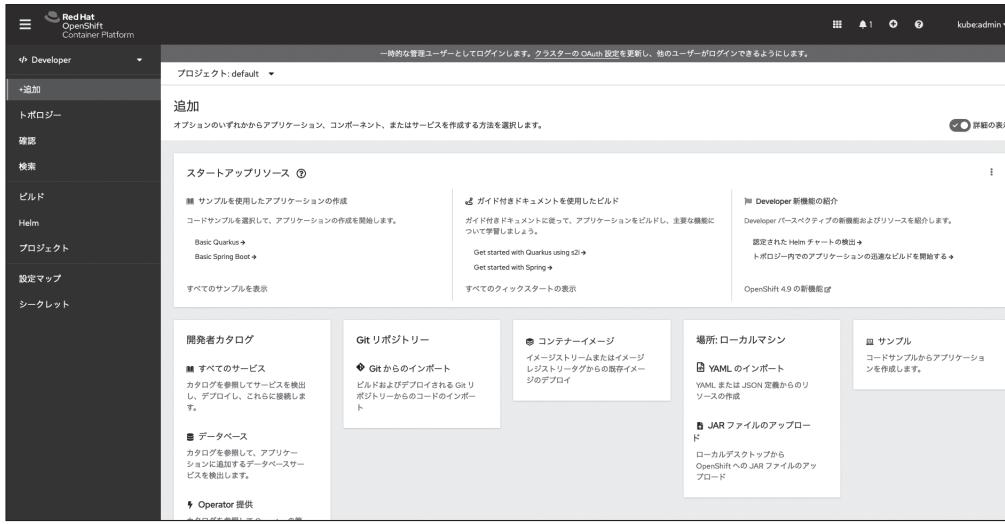


図3.4　Webコンソール（Developer画面）

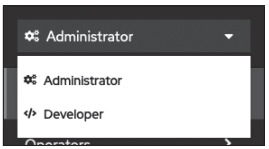


図3.5 AdministratorとDeveloperの切り替え

3.2 CLI（ocコマンド）

OCPのCLIは「oc」というコマンドです。ocを使ってOCPの管理、プロジェクトやアプリケーションの作成・管理を行うことができます。OCPのインストーラーはLinuxとmacOSにしかインストールできませんでしたが、OCPのCLIはLinux、macOS、Windowsにインストールすることができます。

3.2.1 ocコマンドのインストール

ocコマンドは、次のいずれかの方法でインストールできます。

- バイナリをダウンロードしてインストール（❶〜❸のいずれか）
  - ❶ 稼働中のOCPのWebコンソールからバイナリをダウンロード
  - ❷ Red Hat OpenShift Cluster Managerサイトからバイナリをダウンロード
    `https://cloud.redhat.com/openshift/install/aws/installer-provisioned`
  - ❸ `https://mirror.openshift.com`からバイナリをダウンロード
    過去のバージョンのバイナリを取得可
    `https://mirror.openshift.com/pub/openshift-v4/clients/ocp/`
- RPMでインストール
  `https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/cli_tools/_openshift-cli-oc#cli-installing-cli-rpm_cli-developer-commands`

ocコマンドは後方互換性があるので、操作対象のOpenShiftクラスタのバージョンと同じか、それより新しいものを使用すれば正常に動作します。バージョン間の互換性の詳細はOCPのオンラインドキュメントを参照してください。

- OpenShift Container Platformのバージョン管理ポリシー
  `https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/release_notes/ocp-versioning-policy`

本書では、「❷Red Hat OpenShift Cluster Managerサイトからバイナリをダウンロード」してインストールする方法を説明します。バイナリファイルの入手方法が違う場合も、バイナリ取得後の作業手順は同じです。他のインストール方法については、以下のオンラインドキュメントを参考にしてください。

- OpenShift CLIの使用を開始する
  `https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/cli_tools/_openshift-cli-oc#cli-getting-started`

■バイナリをダウンロードしてインストール

それではまず、Red Hat OpenShift Cluster Managerサイト（**図3.6**）にアクセスします。「Command line interface」でインストール先のOSを選択し、［Download command-line tools］ボタンを押してダウンロードします。

- Red Hat OpenShift Cluster Managerサイト
  `https://cloud.redhat.com/openshift/install/aws/installer-provisioned`

なお、踏み台サーバーにSSHでログインしてこの作業を行うときなど、CLI環境で操作していてブラウザを開けない場合は、この［Download command-line tools］ボタンのリンクをコピーして、次のようにcurlなどを使ってダウンロードしてください。

```
$ curl -O https://mirror.openshift.com/pub/openshift-v4/clients/ocp/stable/openshift-client-linux➔
.tar.gz
```

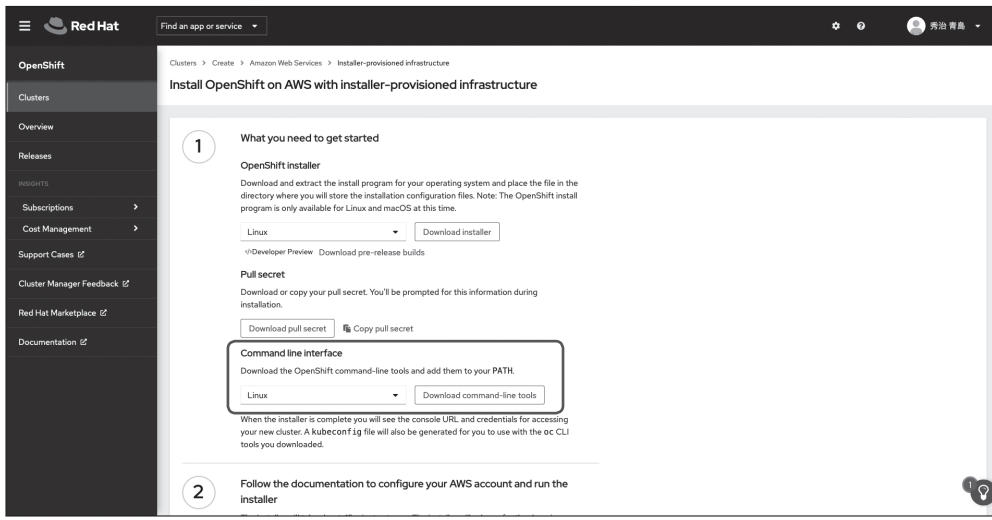


図3.6　Red Hat OpenShift Cluster Managerサイト

ダウンロードしたアーカイブを展開します。Linuxでは次のように展開します。

```
$ tar xvf openshift-client-linux.tar.gz
```

展開されたocファイルを実行可能なパスに移動します。Linuxであれば、/usr/local/bin/の中が実行可能であることが多いです。

```
$ sudo mv oc /usr/local/bin
```

これで、ocコマンドを利用できるようになります。
試しに次のコマンドでインストールが成功しているか確認してみてください。

```
$ oc version
Client Version: 4.9.5
```

上記のversionのように、ocの次に入力した単語を「サブコマンド」と言います。

3.2.2 ocコマンドでログイン

CLIでも、OCPの操作を始める前にログインする必要があります。ログインは、loginサブコマンドで行います。

▶ **ログインの入力例**

```
$ oc login https://api.democluster.ocpdemo.aoshima.me:6443 -u kubeadmin
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be intercepted by
 others.
Use insecure connections? (y/n): y

Authentication required for https://api.democluster.ocpdemo.aoshima.me:6443 (openshift)
Username: kubeadmin
Password:
Login successful.

You have access to 65 projects, the list has been suppressed. You can list all projects with 'oc
projects'

Using project "default".
Welcome! See 'oc help' to get started.
```

3.2.3 ocコマンドの使い方

ocをサブコマンドなしで実行すると、使い方の簡単な説明が表示されます。

```
$ oc
OpenShift Client

This client helps you develop, build, deploy, and run your applications on any
OpenShift or Kubernetes cluster. It also includes the administrative
commands for managing a cluster under the 'adm' subcommand.

To familiarize yourself with OpenShift, login to your cluster and try creating a sample application:

    oc login mycluster.mycompany.com
    oc new-project my-example
    oc new-app django-psql-example
    oc logs -f bc/django-psql-exampl

...
```

使えるコマンドは、helpサブコマンドで表示できます。

サブコマンドの使い方を調べたいときは、「oc help login」のようにhelpサブコマンドに続けて調べたいサブコマンドを入力するか、もしくは「oc login -h」のようにサブコマンドに続けて-hを入力します。

▶ **helpでloginサブコマンドの使い方を調べる**

```
$ oc help login
Log in to your server and save login for subsequent use

 First-time users of the client should run this command to connect to a server, establish an auth
enticated session, and
save connection to the configuration file. The default configuration will be saved to your home
directory under
".kube/config".
...
```

▶ **-hでサブコマンドloginの使い方を調べる**

```
$ oc login -h
Log in to your server and save login for subsequent use

 First-time users of the client should run this command to connect to a server, establish an auth
enticated session, and
save connection to the configuration file. The default configuration will be saved to your home
directory under
```

```
".kube/config".
...
```

それでは、実際にいくつかコマンドを使ってみましょう。

たとえば、意図したバージョンのクラスタが正常にインストールされていることを確認したい場合は、oc get clusterversionコマンドを使います。

```
$ oc get clusterversion
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.9.5     True        False         62m     Cluster version is 4.9.5
```

上記の出力例は、バージョン4.9.5のクラスタが正常にインストールされていることを示しています。

また、ノードのステータスを確認したい場合は、oc get nodeコマンドを使ってください。

```
$ oc get node
NAME                                         STATUS   ROLES    AGE   VERSION
ip-10-0-132-54.ap-northeast-1.compute.internal    Ready    master   90m   v1.22.0-rc.0+a44d0f0
ip-10-0-156-248.ap-northeast-1.compute.internal   Ready    worker   80m   v1.22.0-rc.0+a44d0f0
ip-10-0-164-88.ap-northeast-1.compute.internal    Ready    master   91m   v1.22.0-rc.0+a44d0f0
ip-10-0-182-235.ap-northeast-1.compute.internal   Ready    worker   77m   v1.22.0-rc.0+a44d0f0
ip-10-0-196-225.ap-northeast-1.compute.internal   Ready    worker   81m   v1.22.0-rc.0+a44d0f0
ip-10-0-199-89.ap-northeast-1.compute.internal    Ready    master   91m   v1.22.0-rc.0+a44d0f0
```

上記の出力例は、全ノードのステータスが「Ready」であることを示しています。

■ocコマンドチートシート

運用管理でよく使うコマンドの例を**表3.1**に挙げます。チートシートとしてご利用ください。

表3.1　運用管理でよく使うocコマンド

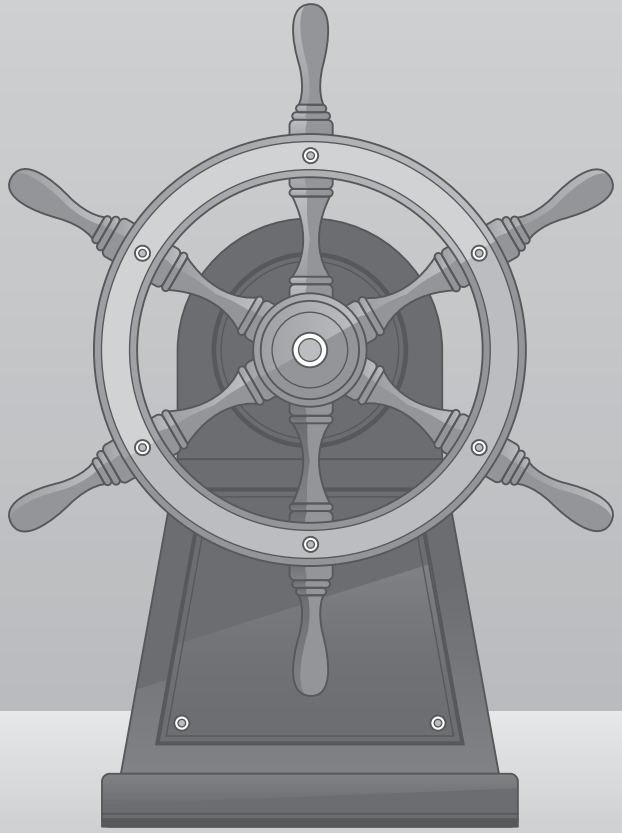
コマンド	説明
oc login -u *USER* -n *PROJ*	カレントプロジェクトを***PROJ***として***USER***でログイン
oc cluster-info	クラスタ情報を出力
oc status	カレントプロジェクトの状態を出力
oc whoami	ログイン中のユーザーを表示
oc projects	プロジェクトの一覧を出力
oc project *PROJ*	カレントプロジェクトを***PROJ***に変更
oc get all	Pod、Service、Deployment、ReplicaSetなどの一覧を出力
oc get pod --show-labels	Pod一覧にラベル情報も付加して出力
oc get pod -l *KEY=VALUE*	条件に合致するPod一覧を出力
oc get events -w	プロジェクトで起こるイベントを継続的に出力

コマンド	説明
`oc get pvc -n PROJ`	*PROJ* の PersistentVolumeClaim の一覧を出力
`oc get deploy --all-namespaces`	すべてのプロジェクトの Deployment の一覧を出力
`oc get node -o wide`	Node の一覧をより詳しく表形式で出力
`oc get svc SVC -o yaml|json`	*SVC* の情報を yaml	json 形式で出力
`oc get csv`	ClusterServiceVersion 情報の出力
`oc describe rc RC`	*RC* の詳細情報を出力
`oc new-project PROJ`	新規プロジェクトとして *PROJ* を作成
`oc create -f MANIFEST`	*MANIFEST* ファイル（yaml	json）で定義されたリソースを新規作成
`oc apply -f MANIFEST`	*MANIFEST* ファイル（yaml	json）で定義されたリソースが存在しない場合は新規作成、存在する場合は更新を適用
`oc process TEMPLATE | oc apply -f`	*TEMPLATE* ファイル（yaml	json）をリソースリストに変換して apply コマンドにパイプで渡す
`oc edit dc DC`	*DC* の設定を変更（vi 形式）
`oc patch dc DC -p PATCH`	*DC* に *PATCH* 文字列（yaml	json）の内容に基づくパッチをあててオブジェクトを更新する
`oc set env dc/DC KEY=VALUE`	*DC* に環境変数を追加
`oc delete pod POD`	*POD* を削除
`oc exec POD bash -it`	*POD* のファーストコンテナで対話的な bash を実行
`oc exec POD date -c CONTAINER`	*POD* の *CONTAINER* で date コマンドを実行（リモートコマンド）
`oc rsh POD`	*POD* のファーストコンテナでシェルセッションを開く
`oc debug dc/DC`	*DC* のデバッグ用 Pod の立ち上げ
`oc port-forward svc/SVC 8888:http`	ローカルの 8888 ポートへのアクセスを *SVC* の http ポートにポートフォワードする
`oc explain RESOURCE`	*RESOURCE*（たとえば pods、nodes）についての説明とフィールドの表示
`oc logs POD -f`	*POD* のログを継続的に出力
`oc logs POD -c CONTAINER`	*POD* の CONTAINER のログを出力
`oc rollout undo dc/DC`	*DC* を 1 つ前のリビジョンにロールバック
`oc config view`	CLI の設定情報を出力
`oc adm policy add-cluster-role-to-user ROLE USER`	*USER* に *ROLE* のクラスタロールを付与
`oc adm top node`	Node のメトリクスを出力
`oc adm drain NODE`	*NODE* からすべての Pod を退避
`oc adm cordon NODE`	*NODE* を Pod のスケジュール不可にする設定
`oc adm uncordon NODE`	*NODE* を Pod のスケジュール可能にする設定

Chapter

4

ユーザー管理



本章では、ユーザーの登録と削除、権限設定などユーザーの管理方法について具体例を通して説明します。

4.1 認証と認可

4.1.1 認証（Authentication）

ユーザーがOpenShiftを操作するためには、まずクラスタに対して「認証（Authentication）」を行う必要があります。認証とは、リクエストを送ってきたのが誰なのか確認することです。

OpenShiftにはOAuthサーバーが含まれており、ユーザーがOpenShiftを操作したいときは、最初にこのOAuthサーバーにアクセストークンを要求します。すると、OAuthサーバーは連携しているIdentity Providerにそのユーザーが登録されているか確認を行い、確認ができたらそのユーザーにアクセストークンを発行します。ユーザーはこのアクセストークンを使ってAPIにリクエストを送ります。以上がユーザーを識別する認証のプロセスになります。

■Identity Provider一覧

Identity Provider（IdP）は、ユーザー認証をサービスとし提供します。OpenShift 4で使用できるIdentity Providerは、以下のとおりです。

- HTPasswd：.htpasswdファイルを利用
- Keystone：OpenStack Keystone v3サーバーを利用
- LDAP：LDAPv3サーバーを利用
- Basic認証：HTTPのBasic認証を利用
- Request Header：X-Remote-Userなどのリクエストヘッダー値を利用
- GitHub/GitHub Enterprise：GitHub/GitHub EnterpriseのOAuthを利用
- GitLab：GitLabのOAuthまたはOpenID Connectを利用
- Google：GoogleのOpenID Connectを利用
- OpenID Connect：OpenID Connectの認可コードフローを利用

これらIdentity Providerについては、以下のオンラインドキュメントも参照してください。

- サポートされるアイデンティティープロバイダー
  https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/authentication_and_authorization/supported-identity-providers

4.1.2 認可（Authorization）

「認可」では、ユーザーが行おうとしている操作に対して、そのユーザーに実行権限があるのかどうかを確認します。OpenShiftでは、RBAC（Role Based Access Control）によって、この実行権限を管理しています。

どのような操作が許可されるか定義されたRoleをUserに対してRoleBindingで紐づけます。認可で使用されるオブジェクトを以下に整理します（**図4.1**）。

- **Role**：プロジェクトスコープのリソース（たとえばPod、Serviceなど）に対して許可される操作の一覧。Role自身は特定のプロジェクトに紐づく。
- **ClusterRole**：プロジェクトスコープのリソース＋Clusterレベルのリソース（たとえばNode、PersistentVolume、プロジェクトなど）に対して許可される操作の一覧。ClusterRoleは、特定のプロジェクトに紐づかない。
- **RoleBinding**：RoleとUserもしくはClusterRoleとUserを紐づける。RoleBindingは特定のプロジェクトに紐づく。
- **ClusterRoleBinding**：ClusterRoleとUserを紐づける。ClusterRoleBindingは、特定のプロジェクトに紐づかない。

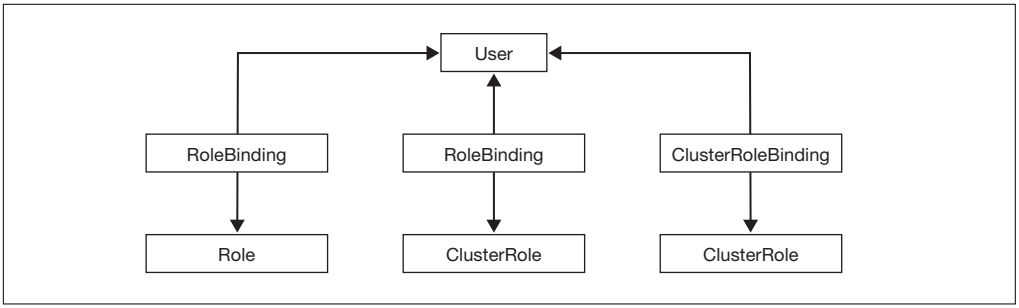


図4.1 権限設定に関するリソース間の関係

ClusterRoleはClusterRoleBindingでUserに紐づける場合と、RoleBindingでUserに紐づける場合があります。

ClusterRoleBindingで紐づけた場合は全プロジェクトに対してClusterRoleで定義された権限が付与されますが、RoleBindingで紐づけた場合はRoleBindingを作成したプロジェクト内でのみ、その権限が付与されます。

プロジェクトスコープのリソースに関する権限設定がしたい場合でも、複数プロジェクトで使用するような権限設定であれば、Roleではなく ClusterRoleで管理しておくことで全プロジェクトから参照できるようになるため、同じ内容のRoleをプロジェクトごとに管理する手間が省けて便利です。

4.2 ユーザー作成

Identity Providerを追加して新規ユーザーを登録する方法を説明します。今回は、一番手軽なHTPasswdを利用することにします。HTPasswdは、ユーザー名とパスワードを格納する .htpasswdファイルを使用してユーザー認証を行うIdentity Providerです。

次の手順で作業します。

1. .htpasswdファイルを作成
2. .htpasswdファイルのSecretを作成
3. OAuthカスタムリソースを編集してIdentity Providerを登録
4. 新規ユーザーに権限を付与

1.と**2.**でIdentity Providerの設定を行い、**3.**でOpenShiftにIdentity Providerを登録します。ここまでが「認証」部分の設定になります。最後に、**4.**でユーザーに権限を与える「認可」部分の設定を行います。

本節の以降の作業では、cluster-admin権限を持つユーザーで実施してください（kubeadminユーザーはcluster-admin権限を持っているので、クラスタインストール直後はkubeadminを使用してください）。

4.2.1 .htpasswdファイルを作成

まずは、踏み台サーバー上で、ユーザー名とパスワードを含む .htpasswdファイルを新規作成してください。今回はadmin01、user01、user02という3人のユーザーを登録することにします[1]。

以下のコマンドを実行すると、プロンプトでパスワード入力を求められます。

```
$ mkdir ~/htpasswd
$ cd ~/htpasswd
$ htpasswd -cB ./users.htpasswd admin01
```

【1】 htpasswdコマンドがインストールされていない場合は、インストールしてください。RHEL 8（Red Hat Enterprise Linux 8）の場合は、sudo yum install httpd-toolsでインストールできます。

```
$ htpasswd -B ./users.htpasswd user01
$ htpasswd -B ./users.htpasswd user02
```

※ -cは新規ファイル作成、-Bはbcryptでパスワード作成のオプション。

4.2.2 .htpasswdファイルのSecretを作成

.htpasswdファイルの内容を、OpenShiftのopenshift-configプロジェクト内にSecretとして登録してください。ここでは、users.htpasswdファイルを、htpass-secretという名前のSecretで登録することにします。

```
$ oc create secret generic htpass-secret --from-file=htpasswd=./users.htpasswd -n openshift-config
```


4.2.3 Identity Providerを登録

次にOAuthサーバーが今作成したSecretを利用するように設定します。OAuthサーバーの設定は、OAuthというカスタムリソースに記載されています。オブジェクトの名前はclusterです。

CLIから設定する場合は、まず次のようにしてOAuthカスタムリソースにパッチをあてます。

```
$ oc patch oauth cluster --type=merge  --patch='{"spec":{"identityProviders":[{"htpasswd":{"file➥
Data": {"name":"htpass-secret"}},"mappingMethod":"claim","name":"my_htpasswd_provider","type":"H➥
TPasswd"}]}}'
```

もちろん、認証サーバーを複数登録することも可能です。その際は、spec.identityProvidersに複数の認証サーバー設定を記載してください。

Webコンソールから設定する場合は、Administrator画面で左側のメニューから［管理］→［クラスター設定］をクリックします。そのあと、右側のパネルでクラスター設定の［設定］タブの設定リソース一覧の中から［OAuth］を選択すると、OAuthの設定画面（**図4.2**）に移動します。この画面でIdentity Providerの設定をすることができます。

図4.2　Identity ProviderをWebコンソールで設定

図4.3は、Identity Providerを追加した後、Webコンソールでログインしたときの画面です。my_htpasswd_providerというIdentity Providerが新規登録されています。

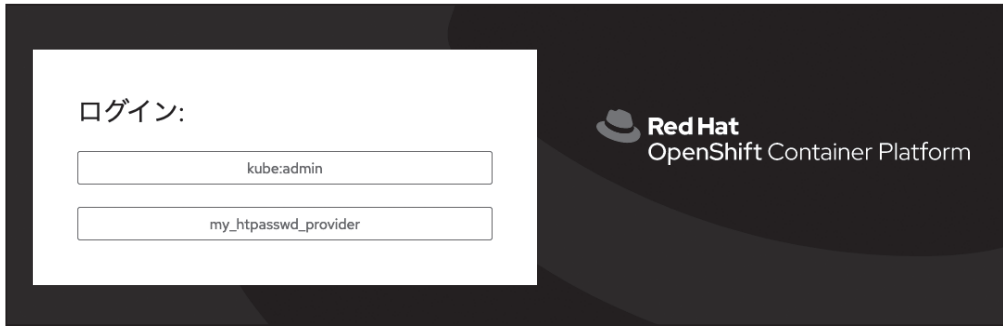


図4.3　Webコンソールのログイン画面（Identity Provider追加後）

4.2.4 新規作成したユーザーでログイン

認証サーバーに登録したユーザーで一度ログインを行うと、OpenShiftにUserとIdentityリソースが作成されます。

```
$ oc login https://api.democluster.ocpdemo.aoshima.me:6443 -u admin01
$ oc login https://api.democluster.ocpdemo.aoshima.me:6443 -u user01
$ oc login https://api.democluster.ocpdemo.aoshima.me:6443 -u user02
```

Webコンソールからログインした場合もUserとIdentityが作成されるので、ocコマンドで再度ログインする必要はありません。

oc getコマンドでUserとIdentityの一覧を表示してみます。

```
$ oc get user
NAME     UID                                    FULL NAME   IDENTITIES
admin01  234e5e5d-7a9e-42f9-a6a3-dfdf3747b34c              my_htpasswd_provider:admin01
user01   2f565a39-6929-4f63-a6b5-87e97fd33d76              my_htpasswd_provider:user01
user02   cd3754e1-38a5-4ab6-be38-cd30a0b1ef8a              my_htpasswd_provider:user02
```

```
$ oc get identity
NAME                           IDP NAME              IDP USER NAME   USER NAME   USER UID
my_htpasswd_provider:admin01   my_htpasswd_provider   admin01         admin01     234e5e5d-7a9e-42
f9-a6a3-dfdf3747b34c
my_htpasswd_provider:user01    my_htpasswd_provider   user01          user01      2f565a39-6929-4f
63-a6b5-87e97fd33d76
my_htpasswd_provider:user02    my_htpasswd_provider   user02          user02      cd3754e1-38a5-4a
b6-be38-cd30a0b1ef8a
```

各ユーザーが、どのIdentity Providerと紐づけられているのかを確認することができます[2]。

4.2.5 ユーザーに権限を付与

admin01ユーザーをクラスタ管理者として使用するには、cluster-adminというClusterRoleをadmin01ユーザーにバインドします。

```
$ oc adm policy add-cluster-role-to-user cluster-admin admin01
```

【2】 確認はkubeadminユーザーで行ってください。ここで新規作成したユーザーはまだ権限を付与されていないため、UserやIdentityの一覧情報を取得することができません。

なお、cluster-adminはデフォルトでクラスタに登録されているClusterRoleなので、自分で作成する必要はありません。このように、よく使用されそうなClusterRoleは、デフォルトで登録されています。

- admin：プロジェクトマネージャ。RoleBindingで使用される場合、adminには、プロジェクト内のすべてのリソースの表示と、プロジェクト内のQuota以外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：すべての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。RoleBindingでユーザーにバインドされる場合、Quotaに対する完全な制御およびプロジェクト内のすべてのリソースに対するすべてのアクションを実行できます。
- cluster-status：基本的なクラスタ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：プロジェクト内でほとんどのオブジェクトを確認できるユーザーです。オブジェクトを変更することはできません。また、ロールとバインディングは表示も変更もできません。

これらデフォルトで登録されているClusterRoleについては、以下のオンラインドキュメントも参照してください。

- デフォルトのクラスタロール
  https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/authentication_and_authorization/using-rbac#default-roles_using-rbac

4.3 ユーザー削除

4.3.1 kubeadminの削除

OpenShiftは、インストール時にクラスタ管理者権限（cluster-admin）を持つkubeadminユーザーが作成されます。このkubeadminユーザーは非常に強い権限を持っていますが、そのパスワードはインストールログに平文で保存されているため、このユーザーをそのまま使い続けるべきではありません。クラスタのセキュリティを強化するために、新たにcluster-admin権限を持つユーザーを作成し、kubeadminは削除してください。

しかし、cluster-admin ロールを付与されたユーザーが作成される前に kubeadmin ユーザーを削除してしまうと、クラスタ管理者権限を持つユーザーが 1 人もいなくなってしまいます。そうなると、クラスタの再インストールが必要になるので十分注意してください [3]。

それでは、kubeadmin を削除しましょう。まずは、kubeadmin 以外の cluster-admin ユーザーでクラスタにログインしておきます。つまり今回は、admin01 ユーザーでログインします。

admin01 ユーザーでログインしたら、kubeadmin Secret を削除します。

```
$ oc delete secrets kubeadmin -n kube-system
```

以上で、kubeadmin ユーザーの削除は完了です。

4.3.2 kubeadmin 以外のユーザーの削除

次に kubeadmin 以外のユーザーを削除してみましょう。ここでは例として user02 を削除してみます。

まず、Identity Provider 上のユーザーエントリを削除します。Identity Provider に HTPasswd を使用している場合は、すでに登録している .htpasswd の Secret リソース（htpass--secret）の内容を書き換えます。

上記の Secret リソースから .htpasswd ファイルの中身を抜き出し、users.htpasswd という名前で作業端末に保存します。

```
$ oc get secret htpass-secret -o jsonpath={.data.htpasswd} -n openshift-config | base64 -d > use➥
rs.htpasswd
```

次に、この users.htpasswd ファイルから user02 のユーザーエントリを削除します。

```
$ htpasswd -D users.htpasswd user02
```

書き換えた users.htpasswd ファイルをもとに Secret オブジェクトを更新します。

```
$ oc create secret generic htpass-secret --from-file=htpasswd=users.htpasswd --dry-run=client -o ➥
yaml -n openshift-config | oc replace -f -
```

以上で、Identity Provider の編集が終わりました。

【3】 上記の手順どおりに admin01 ユーザーを作成し、admin01 ユーザーに cluster-admin 権限を付与できていれば問題ありません。

続いて、OpenShift上のUserオブジェクトとIdentityオブジェクトを削除します。これらのオブジェクトを削除する権限を持ったユーザー、つまり今回でいえばadmin01ユーザーでログインして、次の作業を実施してください。

まずは、Userリソースを削除します。

```
$ oc delete user user02
```

次に、Identityリソースを削除します。

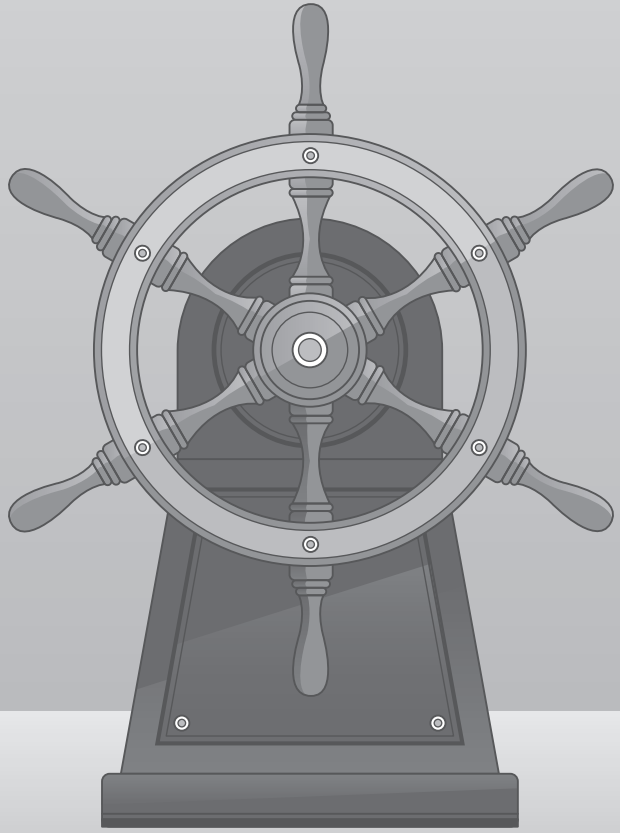
```
$ oc delete identity my_htpasswd_provider:user02
```

これでuser02の削除が完了しました。

Chapter

5

アプリケーションの実行



本章では、具体例を通してサンプルアプリケーションのデプロイ方法を説明します。

5.1 プロジェクト

まずは、アプリケーションをデプロイするためのプロジェクトを用意します。OpenShiftでは、プロジェクトという単位でリソースを分離しています。Kubernetesでいうnamespaceのように使います[1]。プロジェクト単位でのリソース制限やユーザーの権限制御、プロジェクトをまたいだコンテナ同士の通信制御などマルチテナント構成を想定した機能も豊富です。

リソースには、プロジェクトに属するもの（例：Pod、Serviceなど）とプロジェクトに属さないもの（例：Node、PersistentVolumeなど）があるため、慣れないうちは注意が必要です。リソースがプロジェクトに属すものなのかそうでないのかは、次のコマンドで確認することができます。

▶ プロジェクトに属するリソース一覧の表示

```
$ oc api-resources --namespaced=true
NAME                        SHORTNAMES   APIVERSION             NAMESPACED   KIND
bindings                                 v1                     true         Binding
configmaps                  cm           v1                     true         ConfigMap
endpoints                   ep           v1                     true         Endpoints
events                      ev           v1                     true         Event
...
```

▶ プロジェクトに属さないリソース一覧の表示

```
$ oc api-resources --namespaced=false
NAME                        SHORTNAMES   APIVERSION             NAMESPACED   KIND
componentstatuses           cs           v1                     false        ComponentStatus
namespaces                  ns           v1                     false        Namespace
nodes                       no           v1                     false        Node
persistentvolumes           pv           v1                     false        PersistentVolume
...
```

5.1.1 プロジェクトの作成

Webコンソールからプロジェクトを作成します。以下の操作は、プロジェクト作成権限を持つユーザーで実施してください。本書では、前章で作成したcluster-adminのadmin01ユーザーで実施します。

【1】 実態としては、Kubernetesのnamespaceにいくつかのannotationが追加されたものです。

■Webコンソール（Administrator画面）から作成

Webコンソールの Administrator 画面で、左側のメニューから［ホーム］→［プロジェクト］を選択してから、画面右上の［プロジェクトの作成］ボタンをクリックします。**図5.1**のようなダイアログボックスが表示されるので、必要な情報を入力してから［作成］ボタンをクリックしてください。

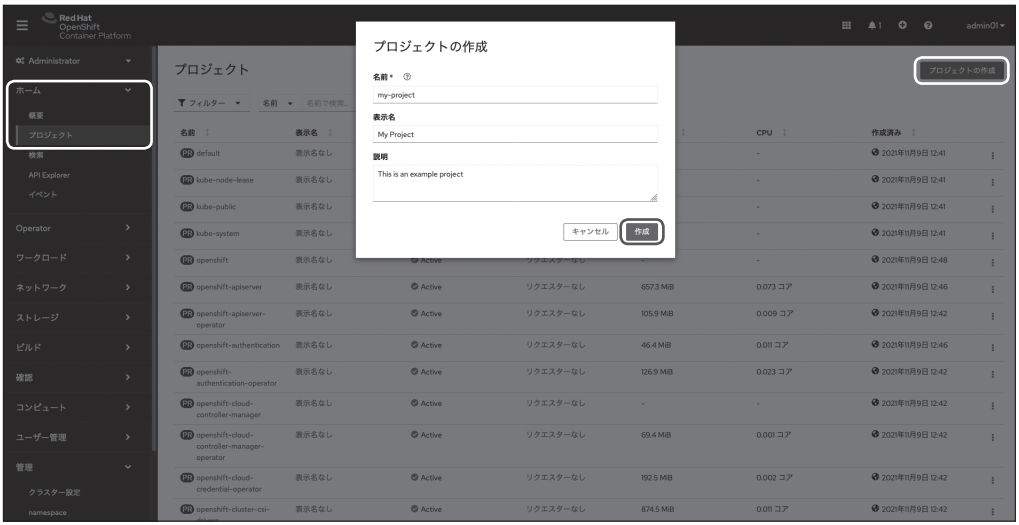


図5.1　Administrator画面でプロジェクトを作成

なお、プロジェクトの作成は、プロジェクトの作成権限を持っているユーザーであれば、Webコンソールの Developer 画面から作成することも、CLIから作成することもできます。詳しくは、以下のオンラインドキュメントを参照してください。

- プロジェクトの使用
  `https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/building_applications/_projects#working-with-projects`

5.2　プロジェクトの操作権限をユーザーに付与

開発者ユーザーにも、`my-project` プロジェクトへのアクセス権限を与えます。Developer 画面で、Developer 画面の［プロジェクト］で操作したいプロジェクト（my-project）を選択して［プロジェクトアクセス］タブに移動します。そこで**図5.2**のように、

- 名前：「`user01`」と入力

- ロール：［Admin］を選択

最後に［保存］ボタンをクリックします。これにより、user01もmy-project内では管理者として振る舞うことができるようになります。

図5.2　プロジェクトのアクセス権限をユーザーに与える

上記の操作により、内部的にはadminというプリセットのクラスタロールをuser01に紐づけるロールバインディングがmy-project内に作成されています。

プリセットの権限設定ではなく、より細かい権限設定をしたい場合は、自分でロールまたはクラスタロールを作成してユーザーに紐づけることもできます。詳しくは、以下のオンラインドキュメントを参照してください。

- RBACの使用によるパーミッションの定義および適用
  https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.9/html/authentication_and_authorization/using-rbac

5.3 アプリケーションのライフサイクル管理

アプリケーションをデプロイする方法は複数ありますが、ここでは一例としてWebコンソールから既存のコンテナイメージを利用してサンプルアプリケーション（workshop-terminal）をデプロイする方法を紹介します。

- workshop-terminal
  `https://github.com/openshift-homeroom/workshop-terminal`

5.3.1 アプリケーションのデプロイ方法

まずは、user01でWebコンソールにログインして、Developer画面を開きます。

次に、左側のメニューから［+追加］を選択し、プロジェクトは「my-project」を選択します（**図5.3**）。

図5.3 ［+追加］を選択

ここで［コンテナーイメージ］を選択すると、設定値の入力画面に遷移します（**図5.4・図5.5**）。

図5.4　コンテナイメージデプロイの入力値（「イメージのデプロイ」画面・前半）

図5.5　コンテナイメージデプロイの入力値（「イメージのデプロイ」画面・後半）

さらに、「イメージのデプロイ」画面で**表5.1**のように必要な情報を入力してから［作成］ボタンをクリックします。

表5.1 「イメージのデプロイ」画面の設定

設定項目		設定値
イメージ		[外部レジストリーからのイメージ名] を選択し、「quay.io/openshifthomeroom/workshop-terminal:3.4.3」と入力する
	ランタイムアイコン	[openshift] を選択する
一般	アプリケーション名	workshop-terminal-app
	名前	workshop-terminal
リソース		[デプロイメント] を選択する
詳細オプション		[アプリケーションへのルートの作成] にチェックを入れる
		「名前をクリックして、詳細オプションにアクセスします。 ルーティング, ヘルスチェック, デプロイメント, スケーリング, リソース制限 and ラベル.」の中の「ルーティング」という文字列をクリックすると、**図5.5**のようにルーティング関連の設定欄が表示される
ルーティング	ホスト名	workshop-terminal-my-project.apps.democluster.ocpdemo.aoshima.me
	パス	/
	ターゲットポート	10080（公開したいServiceを選択する。クリックするとPodに紐づいたServiceが表示されるので、それを選択する）
	セキュリティー	[セキュアなルート] にチェックを入れる
	TLS終端	[Edge] を選択する
	安全でないトラフィック	[Redirect] を選択する
	証明書	空欄のまま

入力内容について説明します。

- **イメージ**：外部レジストリからサンプルアプリとして動くコンテナイメージを取得するように指定しています。「ランタイムアイコン」は、トポロジー画面（**図5.6**）で表示されるアイコンなので何を指定してもかまいませんが、アプリケーションの中身がわかりやすいものにしておいたほうがよいでしょう。
- **一般**：作成されるリソース（Deployment、Service、Routeなど）のラベルと名前を指定しています。［アプリケーション名］がリソースのラベルで、［名前］がリソース名です。
- **リソース**：アプリケーションをDeploymentとDeploymentConfigのどちらでデプロイするのか指定しています。なお、DeploymentConfigはOpenShift固有のリソースなので、Kubernetesには存在しません。DeploymentとDeploymentConfigはとても似た機能なので、今回のサンプルアプリケーションではどちらを選択してもかまいません[2]。また、普段アプリケーションをデプロイするときは、Kubernetesとの移植性を考慮して、DeploymentConfig固有の機能がどうしても必要なとき以外はDeploymentを選択したほうがよいでしょう。

【2】 機能の違いについては、以下のオンラインドキュメントを参照してください。
Comparing Deployment and DeploymentConfig objects
https://docs.openshift.com/container-platform/4.9/applications/deployments/what-deployments-are.html#deployments-comparing-deploymentconfigs_what-deployments-are

- **詳細オプション、ルーティング**：ルートの設定をしています。ルートはクラスタ外部にコンテナアプリケーションを公開するために必要です。通信を暗号化する場合は、TLS終端をどこにするかや使用するサーバー証明書などを設定します。今回は［TLS終端］に［Edge］を選択しているので、暗号化はRouterまでで終了し、Routerからコンテナへの通信は暗号化されていないHTTPになります。また、公開するサービスに独自の証明書が必要な場合は、［証明書］で設置する証明書の設定をしてください。［証明書］で何も設定しなかった場合は、第2章の「2.2 サーバー証明書の設置」で設定したデフォルト証明書が使用されます。今回はデフォルト証明書を使うので何も設定していません。

5.3.2　トポロジー画面

Developer画面の左側のメニューからトポロジー画面に遷移することができます。トポロジー画面（**図5.6**）では、リソースの状態や設定内容また、リソース同士の関係などを視覚的にわかりやすく確認することができます。また、リソースの設定値の変更なども行うことができます。

トポロジー画面で各リソース名をクリックすると、画面右側にリソースの情報が表示されます。たとえば、**図5.6**のように［D workshop-terminal］（Dは Deploymentの略）→［リソース］と進むと、Deploymentの workshop-terminalに紐づいたリソースが表示されます。

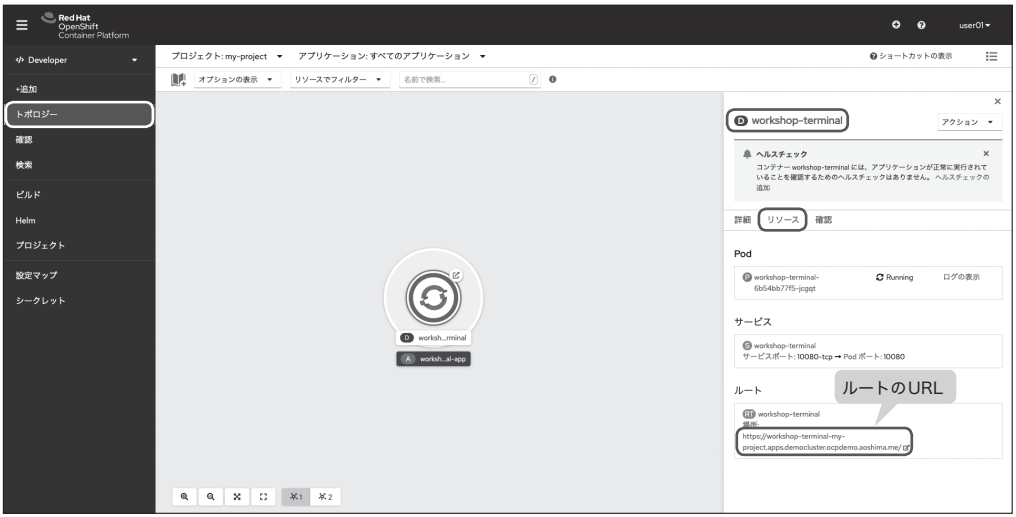


図5.6　トポロジー画面

図5.6のリソース一覧（[リソース]）の［ルート］からルートのURLをクリックし、デプロイしたターミナルアプリケーションが外部公開されていることを確認します。**図5.7**のように表示されれば成功です。

図5.7　ターミナルアプリケーション

■CLIでリソースを確認

続いて、作成されたリソースをCLIで確認してみましょう。

まずは、対象のプロジェクト（my-porject）のリソースの閲覧権限を持つユーザーでクラスタにログインします。

```
$ oc login https://api.democluster.ocpdemo.aoshima.me:6443 -u user01
```

現在使用中のプロジェクトは、oc projectコマンドで確認できます。

```
$ oc project
Using project "my-project" on server "https://api.democluster.ocpdemo.aoshima.me:6443".
```

複数プロジェクトの操作権限を持つユーザーを使っている場合は、my-project以外のプロジェクトがアクティブになっているかもしれません。その場合はoc projectコマンドでmy-projectをアクティブにしてください。

```
$ oc project my-project
```

これで、my-projectプロジェクト内のリソースが確認できるようになりました。oc getコマンドで、Deployment、Replicaset、Pod、Service、Routeを確認してみましょう。

```
$ oc get deployment
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
workshop-terminal       1/1     1            1           28m

$ oc get replicaset
NAME                              DESIRED   CURRENT   READY   AGE
workshop-terminal-6b54bb77f5      1         1         1       28m
```

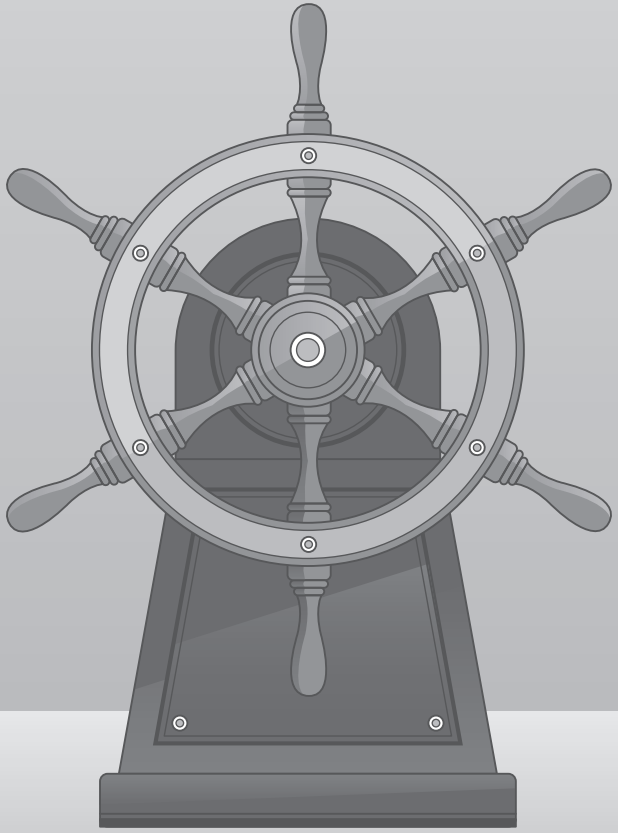
```
$ oc get pod
NAME                                READY   STATUS    RESTARTS   AGE
workshop-terminal-6b54bb77f5-g4cjq   1/1     Running   0          29m

$ oc get service
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
workshop-terminal   ClusterIP   172.30.57.108   <none>        10080/TCP   29m

$ oc get route
NAME                 HOST/PORT                                                    ➔
   PATH   SERVICES         PORT         TERMINATION     WILDCARD
workshop-terminal   workshop-terminal-my-project.apps.democluster.ocpdemo.aoshima.me ➔
   workshop-terminal      10080-tcp    edge/Redirect   None
```

CLIからもリソース情報を確認することができました。

Chapter

6

OCPのインストール方法



第2章「環境構築」では、IPIでOpenShift Container Platform（以下、OCP）のクラスタをAWS環境にインストールする手順を紹介しました。OpenShiftは他にも様々なプラットフォームに導入可能で、複数のインストール方法をサポートしています。

OCPでOpenShiftクラスタをインストールする手順は、おおまかに以下の3つのステップに分けられます。

1. プラットフォームとインストール方法の選択
2. クラスタの設計
3. インストールの実施：UPI（User Provisioned Infrastructure）インストール

本章では、各ステップで考慮して決定すべきことを説明し、最後に最も汎用性の高いインストール方法（Any Platformインストール）でクラスタを構築する手順を紹介します。

6.1 プラットフォームとインストール方法の選択

はじめに、クラスタを構築するプラットフォームを選択します。このとき、インストール方法も一緒に選択します。OCPがサポートするプラットフォームは、パブリッククラウドからオンプレミスまで幅広いのですが、インストール方法によって利用できるプラットフォームが異なります。また、既存のインフラ環境を利用する場合は、環境の制約によってインストール方法が限定されてしまうこともあります。このように、プラットフォームとインストール方法は互いに影響を与えるため、同時に選択することになります。

OCPのインストール方法は、IPIとUPIの2つに大別されます。ここではそれぞれのインストール方法の特徴を説明します。

6.1.1 IPI（Installer-Provisioned Infrastructure）

IPIは、クラウドプロバイダーが提供する基盤構築用のAPIを使用してインストールする方法です。ノードの作成、ロードバランサーの設定、DNSの設定など、OpenShiftクラスタに必要となるコンポーネントを自動で作成し、構築します。

IPIを選択した場合の事前準備として、基盤構築用のAPIを使用するためクラウドプロバイダーのアカウントが必要になります。さらに、OpenShiftで使用するドメインを取得し、そのドメインをプラットフォームのDNSに登録しておく必要があります。

IPIでインストーラーを実行すると、自動で指定した数のノードをデプロイし、DNSサーバーも構成します。このため、ほぼ全自動で簡単にクラスタを構築できます。

その一方で、IPIでは構成は固定的で、細かなカスタマイズはできません。また、IPIが利用できないプラットフォームもあります。この場合は、次に説明するUPIによるOpenShiftのインストールを行うことになります。

6.1.2 UPI (User-Provisioned Infrastructure)

UPIは、ユーザー自身がクラスタの構築に必要となるインフラを準備し、その上にソフトウェアとしてOCPをインストールする方法です。UPIでは、ユーザーが物理サーバーもしくは仮想サーバーを用意してOSをインストールし、さらにDNSの設定やロードバランサーなどの環境を設定します。

UPIでもIPIと同じく、プラットフォームによってはクラウドプロバイダーのユーザーアカウントが求められますが、これはインストール完了後のクラスタでアカウントの認証情報を利用することでクラスタの管理をしやすくするためで、クラウドプロバイダーのユーザーアカウントが必須というわけではありません

UPIには、クラスタを構築する環境に即してカスタマイズできるメリットがあります。たとえば、既存のインフラ環境でクラスタを構築したいものの、ネットワークに制約があってIPIではインストールできない場合でも、UPIでインストールすることができます。ただし、ユーザーが事前に環境を整備する必要があるため、IPIと比べて手間はかかります。

クラウドプロバイダーとは？

「クラウドプロバイダー」と聞くと、AWSやMicrosoftなどのパブリッククラウドサービスの事業者を思い浮かべる方が多いかもしれません。しかしOpenShiftでは、「クラスタの基盤を構築するのためのAPIを提供するIaaSプラットフォーム」と捉えるのが適切です。したがってオンプレミスの環境であっても、OpenStackはインフラ構築用APIを提供するクラウドプロバイダーであり、VMwareもvCenter Serverがインフラ構築用APIを提供するためクラウドプロバイダーとして捉えられます。

また、インストール時に使用するクラウドプロバイダーのアカウントの認証情報は、構築後のクラスタで稼働する「Cloud Credential Operator」によって管理され、プラットフォームに関係するリソースを扱う際に活用されます。

6.1.3 サポートするプラットフォームとインストール方法

執筆時点の最新版であるOCP 4.9では、**表6.1**に挙げている各プラットフォームにインストールすることができます。プラットフォームのバージョンによって条件があることもあるため、必ずOpenShift Container Platformのオンラインドキュメントで、選択するプラットフォームでのインストールの前提条件を確認するようにしてください。

- OpenShift Container Platform 4.9 Documentation
  `https://docs.openshift.com/container-platform/4.9/welcome/index.html`

表6.1 OpenShift Container Platform 4.9がサポートするプラットフォーム

プラットフォーム	IPI	UPI
Amazon Web Services (AWS)	○	○
Google Cloud Platform (GCP)	○	○
Microsoft Azure	○	○
Microsoft Azure Stack Hub	—	○
Red Hat OpenStack Platform (RHOSP)	○	○
Red Hat Virtualization (RHV)	○	○
VMware vSphere	○	○
VMware Cloud (VMC) on AWS	○	○
IBM Z or LinuxONE	—	○
IBM Power Systems	—	○
Bare metal	○	○

6.1.4 サポートするOS

OpenShiftクラスタのすべてのノードでOSが稼働します。サポートしているOSは次の2つです[1]。

- Red Hat Enterprise Linux CoreOS (RHCOS)
- Red Hat Enterprise Linux (RHEL)[2]

【1】 クラスタ構築後、Windows Containerを利用する場合は、Windows Serverが稼働するノードを使います。Windows Serverはクラスタインストール時には使用できないため、ここでは省略しています。

【2】 RHEL 8のWorkerノードがサポートされます。RHEL 7のWorkerノードは非推奨です。
参考: OpenShift Container Platform 4.9 release notes; Red Hat Enterprise Linux (RHEL) 8 now supported for compute machines
`https://docs.openshift.com/container-platform/4.9/release_notes/ocp-4-9-release-notes.html#ocp-4-9-machine-api-rhel8`
OpenShift Container Platform 4.9 release notes; Deprecated and removed features
`https://docs.openshift.com/container-platform/4.9/release_notes/ocp-4-9-release-notes.html#ocp-4-9-deprecated-removed-features`

Masterノードと BootstrapノードのOSはRHCOSであることが必須で、RHELはサポートしていません。

Workerノードと Infraノードは、サーバーに接続される外部機器が独自のドライバーを必要とするなど、RHCOSでは対応できない場合はRHELを使うことができます。

このようにOSの第1候補はRHCOSであり、特別な事情がない限りすべてのノードでRHCOSを使うことを推奨します。

6.1.5 RHCOSのコンポーネント

RHCOSはOpenShiftクラスタでのみ使える、OpenShift環境専用のOSです。RHCOSはRHELをベースに作られているため、RHCOSのコンポーネントはRHELとほぼ同じもので構成され、RHELと同様に管理されます。たとえば、RHCOS内部で稼働するソフトウェアはRPMパッケージで配布されているものと同じで、これらのソフトウェアはRHELカーネルと systemdで管理される一連のサービスとともに稼働します[3]。また、サポートする品質やセキュリティや管理の基準はRHELと同じであるため、同等の信頼性を持つと言えます。セキュリティについては、第10章「セキュリティ」も参照してください。

6.1.6 Ignitionによる自動構成

RHCOSには「Ignition」というユーティリティがあり、RHCOSのインストールや起動後の構成作業を自動で実行できます。構成作業の内容は「Ignition Config」と呼ばれるファイルに記述して、これをRHCOSの起動時に読み込ませることによって自動構成を実現します。

UPIではOpenShiftインストーラーを使ってMasterノード、Workerノード、Bootstrapノードのそれぞれ向けのIgnition Configを作成し、各ノードでRHCOSを起動する際に投入します。

■Machine Config Operatorによる管理

RHCOSはOpenShiftクラスタのMachine Config Operatorによってリモートで管理されます。たとえばRHCOSのアップグレードは、OpenShiftクラスタのアップデートに合わせてMachine Config Operatorが行います。管理者がRHCOSを単独でアップグレードすることはありません。

運用中のRHCOSはMachine Config Operatorで管理されるため、管理者はOpenShiftクラスタのみ管理するだけで済みます。OpenShiftクラスタとRHCOSの両者を管理する必要はありません。

【3】 ただし、RHCOSはコンテナの実行に特化しているため、任意のRPM形式のソフトウェアをインストールすることはできないので注意しましょう。

6.1.7 Any Platformインストール

サポートするプラットフォームを示した**表6.1**の最下行に、「Bare metal」があります。Bare metalは物理サーバーの意味ですが、このBare metalインストールの方法は、物理サーバー環境だけでなく他のあらゆるプラットフォームでも適用できる、最も汎用性の高い“Any Platform”なインストール方法です。

OCPのオンラインドキュメントにも「Installing on any platform」というインストール方法[4]が記載されていますが、これはBare metal UPIインストールの手順とほぼ同じです。ここでは便宜上、この方法を「Any Platformインストール」と呼ぶことにします。

なお、本章第3節では、このAny Platformインストールの方法でOpenShiftクラスタを構築する手順を紹介します。

■Any Platformインストールの特徴

Bare metalを除く他のプラットフォームのインストール方法では、クラウドプロバイダーが必要となります。しかしBare metalおよびAny Platformインストールでは必要ありません。つまり、物理サーバー環境のように、クラウドプロバイダーが存在しない、もしくはクラウドプロバイダーのAPIが利用できない環境であっても、OpenShiftクラスタを構築できる方法が、Any Platformインストールです。

たとえば、VMwareの仮想基盤でクラスタを構築する場合、vCenter Serverにアクセスするユーザーアカウントがないとvsphere IPI/UPIでインストールはできません。また、vCenter Serverのユーザーアカウントがあったとしても、vSphere IPI/UPIが必要とするユーザー権限が与えられていない場合でも同様です。このような場合は、Any Platformインストールでクラスタを構築できます。

■Any Platformインストールの制約

前述のように、Any Platformインストールではクラウドプロバイダーの APIを使用しません。したがって、インストール完了後のクラスタの運用管理でもクラウドプロバイダーのAPIを使うことはできません。そのため、運用保守を行う場合でも、プラットフォームに関係する便利な機能を使うことができなくなります。

たとえば、MachineSetでノードを簡単に増減させたり、ノード障害時に自動復旧するようにしたりはできないため、手作業でのノード管理が必要となります。また、永続ストレージについても、プラットフォームが持つストレージサービスを利用することができないため、管理者が手作業でバックエンドストレージを設定しなくてはなりません。

---

[4] https://docs.openshift.com/container-platform/4.9/installing/installing_platform_agnostic/installing-platform-agnostic.html

このように、Any Platformインストールはクラウドプロバイダーを利用しない方法と比べて、運用フェーズに入ってからの手作業が多く残ってしまうことがあります。

6.2 クラスタの設計

6.2.1 設計のポイント

プラットフォームとインストール方法が決まったら、次は構築するOpenShiftクラスタを設計します。OpenShiftクラスタは様々な要素が含まれています。代表的なコンポーネントとして、「ノード」「ネットワーク」「ストレージ」の3つがあります。

本章ではこの3つと、他に必要となるサービスの設計について説明します。

6.2.2 ノード

ノードはOpenShiftクラスタを構成する主要コンポーネントで、OCPのソフトウェアが稼働する物理サーバーや仮想マシン、クラウドインスタンスを指します。

ノードは「Masterノード」と「Workerノード」の2つに大別されます。それぞれOpenShiftクラスタの中で担う役割や必要となる数が異なります。

■Masterノード

MasterノードはOpenShiftクラスタに必須のノードで、クラスタ全体にわたってノードやPodをはじめとするリソースを管理する役割を持ちます。Masterノードの主な機能として以下のものがあります。

- **API Server**
  OpenShiftクラスタの操作は、すべてAPIによってコントロールされます。そのAPIを公開しているのがAPI Serverです。APIはクラスタ内だけでなく、クラスタ外部にも公開されていて、OpenShift Webコンソールや CLI（ocコマンド）などによるクラスタ外部からの操作も可能としています。

- **OAuth Server/OAuth API Server**
  クラスタ外部からのAPIリクエストを受ける際には、リクエストを送るユーザーが何者かを確認

する必要があり、さらにそのユーザーが適切な権限を持つかを確認する必要があります。OAuth ServerとOAuth API Serverはこの認証と認可の機能を提供しています。

- **etcd**
  OpenShiftクラスタでは、クラスタの構成や設定、各種リソース、アカウント／ロールなどのほぼすべての情報を、「etcd」という分散型のKVS（Key-Value Store）に保管します。etcdは複数のMasterノードで稼働し、ノード間で常時データを複製して保管しています。etcdでは、分散したetcdインスタンスの間で合意をとってクラスタ情報の変更を管理しています。この合意形成のためにetcdは奇数個のインスタンスが必要とされ、これが後述するMasterノードの台数の要件に影響を与えます。なお、偶数個のインスタンスの場合は、インスタンス間で通信できなくなった際に「スプリットブレイン」という何が正しいかを判断できない状態に陥る可能性があるため、避けるべきです。

- **Scheduler**
  Schedulerは、Podを稼働させるノードを決定する役割を持ちます。クラスタ内で新しく作成されるPodを監視し、Podが稼働するために最適なノードを選択します。どのように最適なノードを決めるかは、第11章「Day 2オペレーション」の「11.4 メンテナンス」で詳述しています。

- **Controller Manager**
  OpenShiftには様々なリソースがありますが、各リソースには自らの状態を監視する`Controller`というプロセスがあります。`Controller`は各リソースに問題が発生するとそれを検知し、継続して稼働するにおいて好ましい状態になるよう対応を行います。この各`Controller`をまとめて単一プロセスとして実行しているのがController Managerです。

Masterノードのシステム要件

Masterノードとなるサーバーには、**表6.2**のような最小システム要件があります。Masterノードはetcdを構成するため3台必要になります。Masterノードの台数は、3台より多くても少なくてもサポートされません。ちょうど3台がサポートされる台数であるため、注意してください。

表6.2 Masterノードの最小システム要件

台数※1	OS	CPU（コア）※2	メモリ（GB）	ストレージ（GB）
3	RHCOS	4	16	120

※1 技術的には1台でも稼働しますが、サポート要件上複数台が求められます。
※2 SMT（同時マルチスレッディング）が利用できるCPUの場合は、1スレッド＝1コアと数えてかまいません。

■Workerノード

WorkerノードはOpenShiftクラスタに必須のノードで、アプリケーションのPodを稼働させるというOpenShiftの中核をなす役割を持ちます。Workerノードは、アプリケーションPodにCPUやメモ

リといったコンピュートリソースを提供するため、Workerノードの数とリソースの量はクラスタ全体の規模を反映します。それゆえWorkerノードはクラスタ設計において重要な項目です。

また、WorkerノードではOpenShiftが提供するコンポーネント（OpenShift Logging、OpenShift Monitoringなど）のPodも稼働します。

Workerノードのシステム要件

Workerノードとなるサーバーには、**表6.3**のような最小システム要件があります。Workerノードは、アプリケーションの単一障害点とならないために2台以上が必要です。

表6.3　Workerノードの最小システム要件

台数※1	OS	CPU（コア）※2	メモリ（GB）	ストレージ（GB）
2以上	RHCOS RHEL	2	8	120

※1　技術的には1台でも稼働しますが、サポート要件上複数台が求められます。
※2　SMT（同時マルチスレッディング）が利用できるCPUの場合は、1スレッド＝1コアと数えてかまいません。

Masterノードでのユーザーアプリケーションの稼働

一般的に、MasterノードではユーザーアプリケーションのPodをスケジュールしないようにします。しかし、エッジコンピューティングなどサーバーの台数や空間に制約があるユースケースでは、ノードの数を抑える目的でMasterノードにユーザーアプリケーションのPodをスケジュールするように設定できます。その場合、ノードはMasterとWorkerの両方の機能を果たすことになるため、コンピュートリソースを豊富に搭載するようにしてください。

■特殊なノード

Infraノード

Infraノードは Infrastructure ノードの略で、Workerノードの一種です。WorkerノードではOpenShiftが提供するコンポーネントも稼働しますが、これらのコンポーネントを専用で稼働させるノードがInfraノードです。具体的には、次の4つをInfraノードで稼働させることができます。

- OpenShift Logging
- OpenShift Monitoring
- OpenShift Registry
- Ingress Router

Infraノードを用意しておくと、Workerノードでこれらのコンポーネントにコンピュートリソースを取られて、本来稼働させたいユーザーアプリケーションに割り当てるリソースが圧迫されるのを回避

できます。

可能であればInfraノードを用意することをおすすめしますが、必須ではありません。Infraノードのためのサーバーを用意することが難しいケースも考えられるので、ケースバイケースで検討してください。

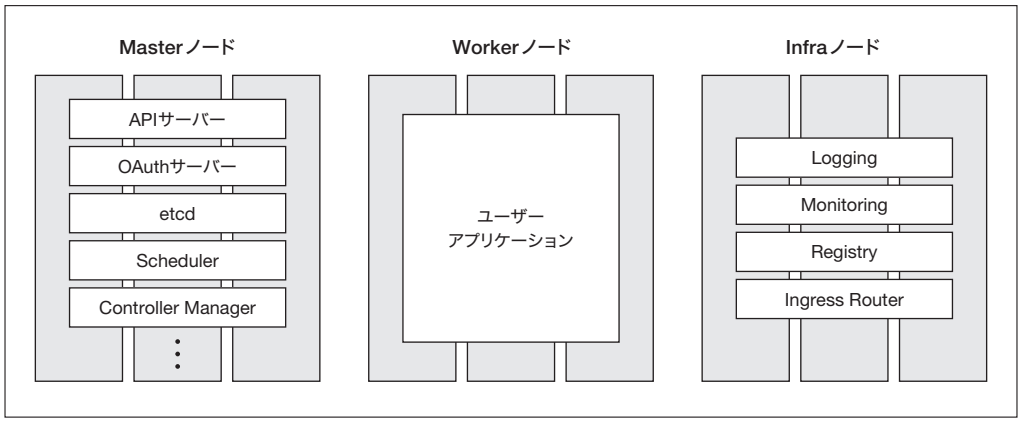


図6.1 各ノードの役割分担

Bootstrapノード

Bootstrapノードは、OpenShiftクラスタをインストールする際に必須となるノードです。クラスタインストールは、Masterノードを展開して最小限のクラスタを作成し、次にWorkerノードを展開してクラスタに追加する手順を踏みます。Bootstrapノードは、Masterノードを展開するためのクラスタの設定情報を提供したり、Masterノードにコントロールプレーン[5]を作成する役割を持ち、Masterノードよりも先に展開されます。Bootstrapノードの詳細な動作についてはGitHubの「OpenShift Installer Overview[6]」に記載されています。

Bootstrapノードはクラスタの一部ではなく、あくまでMasterノードが展開されてクラスタが作成されるまでの一時的なコンポーネントです。そのため、一度Masterノードで最小限のクラスタが作成されるとBootstrapノードは必要なくなるので、削除することができます。

【5】 OpenShift Container Platform 4.9 Documentation; The OpenShift Container Platform control plane
https://docs.openshift.com/container-platform/4.9/architecture/control-plane.html

【6】 OpenShift Installer Overview
https://github.com/openshift/installer/blob/release-4.9/docs/user/overview.md

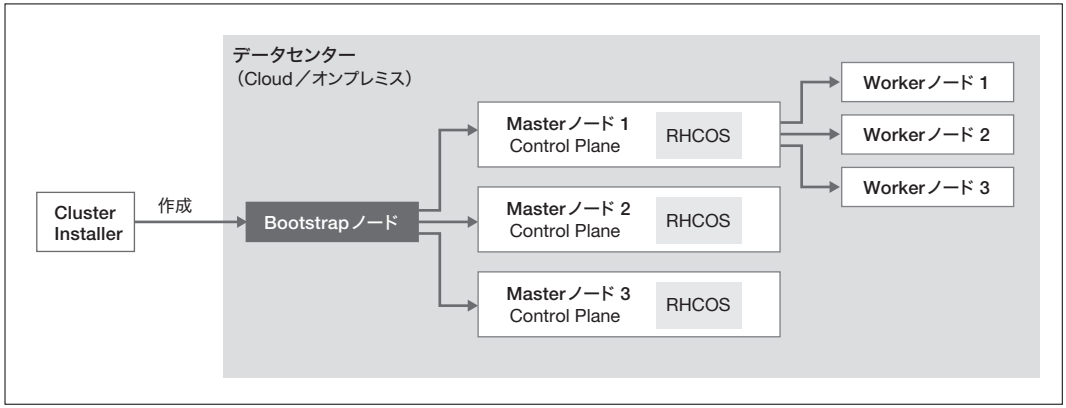


図6.2 OpenShiftクラスタのインストールの流れ

Bootstrapノードとなるサーバーには、**表6.4**のような最小システム要件があります。

表6.4 Bootstrapノードの最小システム要件

台数	OS	CPU（コア）※	メモリ（GB）	ストレージ（GB）
1	RHCOS	4	16	120

※ SMT（同時マルチスレッディング）が利用できるCPUの場合は、1スレッド＝1コアと数えてかまいません。

6.2.3 ノードのサイジング

■Masterノード

Masterノードのハードウェアリソースは、管理対象のWorkerノードの数によって変わります。Masterノードはスケールアウトしたり、CPUやメモリのサイズを変更することができないため、あらかじめクラスタに配備するWorkerノードの最大数を想定してスペックを決めるようにしてください。

参考になる記述がOCPのオンラインドキュメントにあります[7]。「Control plane node sizing」のセクションには、**表6.5**のような表が記載されています（オリジナルは英語表記です）。これはWorkerノードとクラスタのNamespaceの数に対する、Masterノードに搭載するCPUとメモリの対応表です。これを参考にサイジングしてもよいでしょう。なお、それぞれのNamespaceで稼働するワークロード（OpenShiftリソース）の前提についてはオンラインドキュメントを参照してください。

【7】 OpenShift Container Platform 4.9 Documentation：「Recommended host practices」内の「Control plane node sizing」
https://docs.openshift.com/container-platform/4.9/scalability_and_performance/recommended-host-practices.html#master-node-sizing_

表6.5 Masterノードのリソースサイジング

クラスタ内のWorkerノード数	クラスタ内のNamespace数	→	MasterノードのCPUコア※	Masterノードのメモリ（GB）
25ノード	500	→	4	16
100ノード	1000	→	8	32
250ノード	4000	→	16	96

※ SMT（同時マルチスレッディング）が利用できるCPUの場合は、1スレッド＝1コアと数えてかまいません。

■Workerノード

Workerノードのハードウェアリソースは、稼働するPodが求めるリソースに依存します。あらかじめ稼働するアプリケーションがすべてわかっている場合は、アプリケーションが必要とするリソースからクラスタ全体で必要となるリソースをサイジングができます。しかしOpenShiftでこういったケースは稀で、稼働するアプリケーションをすべて事前に特定することが難しい場合が多いでしょう。

この場合は厳密なサイジングは困難なため、様々な仮定を置くことになります。これが確実という方法論はありませんが、次のようなアプローチで臨むことで、ある程度精度の高いサイジングができるでしょう。

1．クラスタ全体で稼働するPodの数を仮定する

まず稼働するPodの数を仮定します。たとえば、クラスタ全体で1200 Pod稼働するとします。

OpenShiftでは1台のWorkerノードで稼働できるPodは最大500です。そのため、「1200 ÷ 500 = 2.4」から切り上げて少なくとも3台のWorkerノードが必要とわかります。

実際には、Workerノードに障害が発生する際に別のWorkerノードにPodを再スケジュールできるようにもっと余裕を持たせます。たとえば、1 Workerノードで稼働させるPodを最大250として、「1200 ÷ 250 = 4.8」、これを切り上げて「5台」などと見積もります。

2．Podが必要とするリソースを仮定する

Podに必要となるリソースはアプリケーションごとに異なりますが、稼働するアプリケーションがわからない場合はこれも仮定します。

たとえば、1 Podあたり平均して0.25コアのCPUと、0.5 GBのメモリを使うと仮定します。クラスタ全体で1200 Pod稼働するとすれば、すべてのWorkerノードで必要となるリソースは次のように計算できます。

- CPU： 0.25 × 1200 = 300コア
- メモリ： 0.5 × 1200 = 600 GB

3. Workerノードの数とリソースを計算して調整する

先に計算したCPUとメモリを5台のWorkerノードで満たすためには、1 Workerノードに求められるリソースは次のように計算できます。

- CPU： 300 ÷ 5 = 60コア
- メモリ： 600 ÷ 5 = 120 GB

計算の結果が現実的でない数値の場合、それはWorkerノードの数が少ないことを意味します。現実的なWorkerノードのリソースになるようにWorkerノードの数を調整します。

たとえばWorkerノードの数を20台とすれば、1 Workerノードに求められるリソースは次のようになり、先ほどよりは用意しやすいスペックとなります。

- CPU： 300 ÷ 20 = 15コア
- メモリ： 600 ÷ 20 = 30 GB

以上のようなPodの数とリソースを仮定してサイジングする方法は、単純ですが概算としては十分なレベルの結果を得られるでしょう。

なお、OpenShiftではオーバーコミットが許容されています。上の例ではオーバーコミットを一切考慮していないので、安全ですがかなり余裕を持たせたサイジングと言えます。この場合、リスクとのバランスを考慮したうえで適切な割合のオーバーコミットを織り込むことで、より現実的な値のリソースで構成することが可能です。

ただし、メモリのオーバーコミットは極力避けるべきです。Workerノードでのメモリ不足はOOM Killer（Out of Memory Killer）によるPodの停止につながり、多くの場合致命的な事態になります。CPUはオーバーコミットしても、メモリはオーバーコミットしないようにすることをおすすめします。

Workerノードのサイジングのヒントになる情報がOCPのオンラインドキュメントに記載されています。参考にしてください。

- OpenShift Container Platform 4.9 Documentation：Planning your environment according to object maximums
  https://docs.openshift.com/container-platform/4.9/scalability_and_performance/planning-your-environment-according-to-object-maximums.html

■Infraノード

Infraノードのハードウェアリソースは Masterノードと同じく、Workerノードの数によります。またこちらもWorkerノードの数に対してInfraノードに搭載するCPUとメモリの参考数値がOCPのオン

ラインドキュメント[8]に記載されています（**表6.6**、オリジナルは英語表記です）。

表6.6 Infraノードのリソースサイジング

クラスタ内のWorkerノード数	→	Infraノードの CPUコア※	Infraノードの メモリ（GB）	Infraノードの CPUコア with Logging	メモリ（GB） with Logging
25ノード	→	4	16	4	64
100ノード	→	8	32	8	128
250ノード	→	16	128	16	128
500ノード	→	32	128	32	192

※ SMT（同時マルチスレッディング）が利用できるCPUの場合は、1スレッド＝1コアと数えてかまいません。

なおこの表では、OpenShift Loggingだけは別にリソースの確保が求められている点に注意してください。

左から2列目と3列目のInfraノードの「CPUコア」と「メモリ（GB）」は、Monitoring、Registry、Routerの3つ分だけのリソースです。これらのコンポーネントはクラスタインストールと同時に自動でインストールされていますが、OpenShift Loggingはクラスタ構築後に管理者が任意でインストールするものです。そのため、Loggingだけは別枠で分かれています。

たとえば、Workerノードが25台以下のクラスタでMonitoring、Registry、Router、Loggingのすべてを Infraノードで稼働させる場合は、CPUは4 + 4 = 8コアを、メモリは16 + 64 = 80 GBをそれぞれInfraノードで確保しておくとよいでしょう。

6.2.4 ネットワーク

次に、OpenShiftクラスタを構築するときに必要となるネットワークについて説明します。ここでは、OpenShiftのネットワークで使われる技術要素の概要のみ取り上げます。ネットワークの詳細については第8章「ネットワーク」を参照してください。

■仮想ネットワーク

クラスタを構成するすべてのノードは、1つのNICから単一のネットワーク（Host Network）に接続することが一般的です。各ノード上ではSDN（Software Defined Networking）のPodが稼働して、全ノードを横断する仮想ネットワークが作られます。OpenShiftクラスタ内のノード間の通信は、おおむねこの仮想ネットワークを使って行われます。

【8】 OpenShift Container Platform 4.9 Documentation：「Recommended host practices」内の「Infrastructure node sizing」
https://docs.openshift.com/container-platform/4.9/scalability_and_performance/recommended-host-practices.html#infrastructure-node-sizing_

クラスタのインストール時に定義する仮想ネットワークとして、「Cluster Network」と「Service Network」の2つがあります。これらのネットワークのサブネットは、他のネットワークと衝突していない限りは任意です。しかしPodやServiceはクラスタ全体で大量に作られることもあるので、広いレンジにするように注意してください。以下にそれぞれのネットワークの役割と、デフォルトのCIDR（Classless Inter-Domain Routing）を示します。

Cluster Network

Cluster Networkは、クラスタで稼働するPodにインターフェースを提供し、クラスタ内部のPod間通信を実現するためのネットワークです。PodにはCluster Networkのレンジにあ るIPアドレスがダイナミックに割り当てられます[9]。

Cluster NetworkのデフォルトCIDRは「10.128.0.0/14」です。

Service Network

Service Networkは、Serviceリソースにインターフェースを提供し、Pod内で稼働するアプリケーションをクラスタ内部／外部に公開する際に使用するネットワークです。ServiceにはService Networkのレンジにあ るIPアドレスがダイナミックに割り当てられます。

Service NetworkのデフォルトCIDRは「172.30.0.0/16」です。

■DNS

OpenShiftでは、クラスタ内外に公開されるアプリケーションやAPIサーバーのURLのドメイン名を解決するためのDNSが必要です。そのため、クラスタインストール時にはDNSサーバーが必要となります。クラスタを構築する環境で使用できる既存のDNSサーバーがあれば、新規で用意しなくても問題ありません。

DNSサーバーでは、**表6.7**に挙げている6つのドメイン名を解決します。`cluster_name`と`base_domain`、およびMasterノードとWorkerノードのホスト名については任意です。

表6.7　DNSサーバーが解決する6つのドメイン名

ドメイン名	レコード	コンポーネント
api.<cluster_name>.<base_domain>.	A/AAAA または CNAME　PTR	Kubernetes API
api-int.<cluster_name>.<base_domain>.	A/AAAA または CNAME　PTR	Kubernetes API
*.app.<cluster_name>.<base_domain>.	A/AAAA または CNAME	Route
bootstrap.<cluster_name>.<base_domain>.	A/AAAA または CNAME　PTR	Bootstrap
<master><n>.<cluster_name>.<base_domain>.	A/AAAA または CNAME　PTR	Master
<worker><n>.<cluster_name>.<base_domain>.	A/AAAA または CNAME　PTR	Worker

【9】 PodにHost NetworkのIPアドレスを割り当てて直接Host Networkで通信することも可能ですが、これは特殊なケースです。アプリケーション上どうしても必要となる場合を除いて、PodはCluster Networkを使うことが一般的です。

■ロードバランサー

OpenShiftでは、クラスタ内外のアプリケーションやAPI通信のためのロードバランサーが必要です。そのためクラスタインストール時にはロードバランサーが必要となります。ロードバランサーは、クラスタを構築する環境で使用できる既存のロードバランサーがあれば、新規で用意する必要はありません。

ロードバランサーは、クラスタ内部のノード間通信の負荷分散用と、クラスタ外部から来るアクセスの負荷分散の2種類を必要とします。別々で用意しても、1つにまとめてもどちらでも問題ありません。**表6.8**のようにポートと対象ノードのエントリをロードバランサーで設定してください。

表6.8 ロードバランサーの設定

ポート	対象ノード	クラスタ内部向け	クラスタ外部向け	負荷分散の対象
6443	Bootstrap Master	○	○	API Server
22623	Bootstrap Master	○	—	Machine Config Server
443	Worker or Infra（Ingress Routerが稼働するノード）	○	○	Ingress RouterのHTTPSアクセス
80	Worker or Infra（Ingress Routerが稼働するノード）	○	○	Ingress RouterのHTTPアクセス

■DHCP

OpenShiftクラスタのノードはすべてHost Networkに接続されるため、それぞれにHost NetworkのIPアドレスを付与する必要があります。このときにDHCPでIPアドレスが割り振られます。クラウドプロバイダーによっては、プラットフォーム側にDHCPのサービスが存在するものもあるので、具体的な設定方法については、OCPのオンラインドキュメントの各クラウドプロバイダーごとのインストール方法【10】を参照してください。

次節では「Any Platformインストールの手順」を記載していますが、ここではノードのセットアップでiPXEを使用するためにDHCPを用意しています。Any Platformインストールでは厳密にはDHCPは必須ではなく、IPXEを使わずにISOイメージからノードをセットアップして静的にIPアドレスを設定する方法もあります。しかし、非常に手間がかかり間違いも起きやすくなるため、DHCPを使うことを推奨します。

【10】OpenShift Container Platform 4.9 Documentation: Cluster installer activities
https://docs.openshift.com/container-platform/4.9/welcome/index.html#cluster-installer-activities

6.2.5 ストレージ

OpenShiftでは、ステートフルなユーザーアプリケーションには永続ストレージが必要となりますが、OpenShiftの基盤コンポーネントである、OpenShift LoggingやOpenShift Registryの構成にも永続ストレージを必要とします。OpenShift Monitoringは永続ストレージがなくても使うことはできますが、実際の運用ではメトリクスデータを永続的に保管するために永続ストレージが必要となることがよくあります。

OpenShiftのアーキテクチャにおいて、ストレージは疎結合なコンポーネントであり、OpenShiftクラスタをインストールする段階では必要ではありません。クラスタインストール完了後に、ストレージをセットアップするのが一般的です。

ストレージの設計は使用するストレージシステムに依存するところが大きいため、本書では説明は省略します。OpenShiftの永続ストレージで使われる技術要素やストレージの選択肢については、第9章「ストレージ」を参照してください。

6.3 UPIインストール

第2章「環境構築」では、AWS環境にIPIでOpenShiftクラスタをインストールする方法を紹介しましたが、本節ではUPIインストールの方法を紹介します。

すでに何度か述べたように、UPIではノードやDNSなどのインストールに必要なインフラをユーザー自身が用意する必要があります。UPIは様々なプラットフォームに対応していますが、インフラを準備するところが異なるだけで、本質的な手順はいずれも変わりません。

そこで本節では、本章第1節で説明したAny Platformインストールを用いて、OpenShiftクラスタを構築する方法を紹介します。この方法は汎用性が高く、インストールに必要なコンポーネントや手順を理解するのに適しています。

6.3.1 Any Platformインストールの流れ

Any Platformインストールでは、次のような流れでクラスタインストールを行います。

1. インストール環境の整備
   - 作業用マシンの作成
   - iPXE/HTTPサーバーの作成
   - DNS／ロードバランサー／DHCPの作成

▸ ノードとなるマシンの用意

2. OpenShiftインストーラー、コマンドラインツール、RHCOSの入手
3. Ignition Configファイルの作成
4. Bootstrapノード／Masterノードの作成
5. Workerノード／Infraノードの作成
6. クラスタ作成の確認

以降は、Any Platformインストールの構成および手順を追いながら、おおまかにその説明を行います。省略している部分については、GitHub（OpenShift Architecture Design Patterns）で同様の内容が公開されているので、そちらを参照してください。

- OpenShift Architecture Design Pattern
  `https://github.com/team-ohc-jp-place/OpenShift-ADP/tree/4.9/BaremetalUPI`

本書では**図6.3**に示している環境を使って、3台のMasterノード、3台のWorkerノード、3台のInfraノードで構成されるクラスタを構築します。

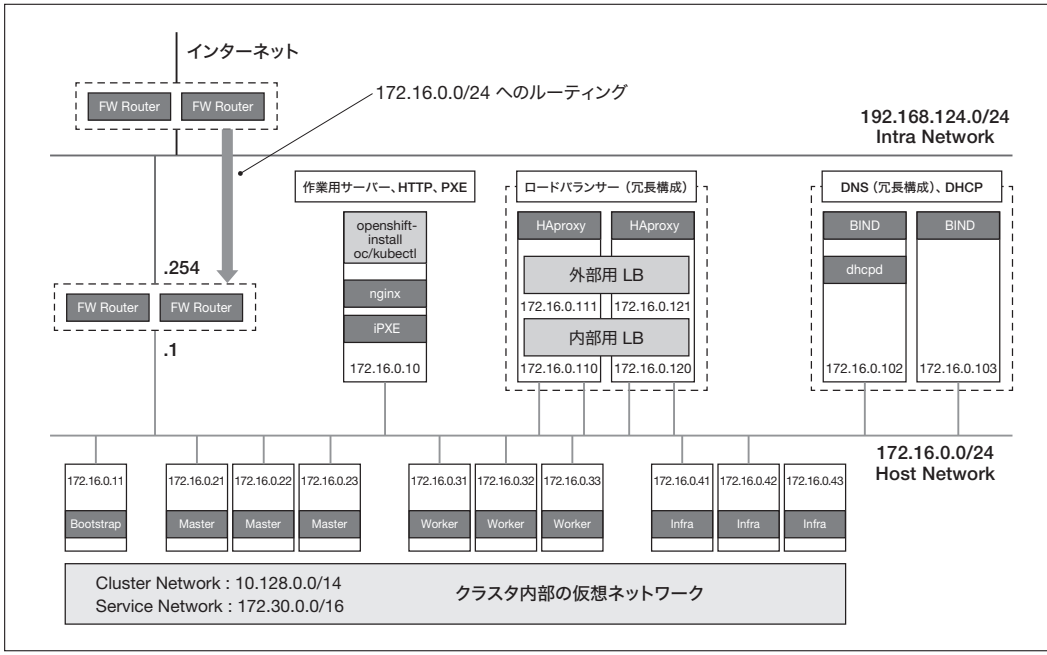


図6.3 Any Platformインストールで構築するクラスタ環境のシステム構成図

6.3.2 インストール環境の整備

■作業用マシンの作成

まずは、インストール作業を行うマシンを作成します。作業用マシンでは、OpenShiftインストーラーを実行したり、コマンドラインツールでクラスタに対してocコマンドを実行したりします。

作業用マシンのスペックに特に指定はありませんが、OpenShiftインストーラーはLinux、macOSのみ、コマンドラインツールはLinux、macOS、Windowsのみ使用できます。また、作業用マシンはクラスタインストール後にノードの追加などの運用管理に利用することもできます。可能であれば、インストール先の環境内にLinuxの作業用マシンを作成しておくことをおすすめします。

■iPXE/HTTPサーバーの作成

今回の環境では、作業用サーバーにiPXEサーバーとHTTPサーバーをセットアップしています。

クラウドプロバイダーが存在するプラットフォームでのクラスタインストールでは、RHCOSの仮想マシンイメージを利用してノードでRHCOSを起動できます。しかし、Bare metalへのインストールおよびAny Platformインストールでは仮想マシンイメージが利用できないため、PXEブートで起動します。RHCOSはISOイメージからでもインストールできますが、クラスタインストールに失敗した際にはノードの再インストールが必要となるため、PXEブートできる環境を整えておくことをおすすめします。

また、HTTPサーバーには、RHCOSのインストールイメージと各ノードのIgnition Configファイルを配置します。各ノードがiPXEで起動する際にこれらのファイルを読み込めるように公開しています。

iPXEサーバーとHTTPサーバーのセットアップ手順は省略します。

■DNS／ロードバランサー／DHCPの作成

DNSとロードバランサーはOpenShiftのクラスタが内外で通信するのに不可欠なシステムです。新規で作成する場合は、冗長構成で作成するようにしてください。すでに環境内に冗長化されたDNSとロードバランサーのシステムがあるなら、それらを活用しても問題ありません。

なお、パブリッククラウドの環境でUPIインストールする場合は、クラウドサービスのDNSとロードバランサーを利用できるので、作成する必要はありません。

DNSサーバー

今回の環境では既存のDNSサーバーが使えないため、新規に作成したRHEL8サーバーでBINDを使って作成しています。ここではBINDのセットアップおよび冗長化の手順は省略します。セットアップ後には、前節の**表6.7**に記載したDNSのエントリを登録しておきます。

参考までに、前述のシステム構成図に準ずるレコードのエントリは以下となります。

▶ **正引き用設定ファイルの例：**/var/named/ocp4.9.example.localdomain.db

```
$ORIGIN example.localdomain.
$TTL 3600
@       IN      SOA     ns1.example.localdomain. root.example.localdomain.(
        2000091802      ; Serial
        3600            ; Refresh
        900             ; Retry
        3600000         ; Expire
        3600 )          ; Minimum
;
        IN      NS      ns1.example.localdomain.
;
api-int.ocp49.example.localdomain.   IN  A  172.16.0.110
api.ocp49.example.localdomain.       IN  A  172.16.0.111
*.apps.ocp49.example.localdomain.    IN  A  172.16.0.111
;
bs.ocp49.example.localdomain.        IN  A  172.16.0.11
m1.ocp49.example.localdomain.        IN  A  172.16.0.21
m2.ocp49.example.localdomain.        IN  A  172.16.0.22
m3.ocp49.example.localdomain.        IN  A  172.16.0.23
w1.ocp49.example.localdomain.        IN  A  172.16.0.31
w2.ocp49.example.localdomain.        IN  A  172.16.0.32
w3.ocp49.example.localdomain.        IN  A  172.16.0.33
i1.ocp49.example.localdomain.        IN  A  172.16.0.41
i2.ocp49.example.localdomain.        IN  A  172.16.0.42
i3.ocp49.example.localdomain.        IN  A  172.16.0.43
```

▶ **逆引き用設定ファイルの例：**/var/named/0.16.172.in-addr.arpa.db

```
$TTL 3600
@       IN      SOA     ns1.example.localdomain. root.example.localdomain.(
        2000091802      ; Serial
        3600            ; Refresh
        900             ; Retry
        3600000         ; Expire
        3600 )          ; Minimum
;
        IN      NS      ns1.example.localdomain.
;
11    IN  PTR bs.ocp49.example.localdomain.
21    IN  PTR m1.ocp49.example.localdomain.
22    IN  PTR m2.ocp49.example.localdomain.
23    IN  PTR m3.ocp49.example.localdomain.
31    IN  PTR w1.ocp49.example.localdomain.
32    IN  PTR w2.ocp49.example.localdomain.
33    IN  PTR w3.ocp49.example.localdomain.
41    IN  PTR i1.ocp49.example.localdomain.
42    IN  PTR i2.ocp49.example.localdomain.
43    IN  PTR i3.ocp49.example.localdomain.
```

ロードバランサー

ロードバランサーは、クラスタ内部のノード間通信の負荷分散用と、クラスタ外部からのアクセスの負荷分散の2種類が必要になります。これらが別々のシステムに分かれている必要はありません。**図6.3**のシステム構成図のように、内部用と外部用を1つの冗長化されたシステム上で別々のIPをアサインする形も可能です。

今回の環境では既存のロードバランサーが使えないため、新規に作成したRHEL8サーバーでHAProxyを使って作成しています。ここではHAProxyのセットアップおよび冗長化の手順は省略します。セットアップ後には、前節の**表6.8**に記載したロードバランサーのエントリを登録しておきます。

参考までに、前述のシステム構成図に準ずるエントリは以下となります。

▶ **HA proxy構成ファイルの例**：/etc/haproxy/haproxy.cfg

```
#---------------------------------------------------------------------
# main frontend which proxys to the backends
#---------------------------------------------------------------------

frontend kubeapi
    default_backend kube_api
    mode tcp
    bind *:6443

frontend machineconfig
    default_backend machine_config
    mode tcp
    bind *:22623

 frontend workerhttp
    default_backend worker_http
    mode tcp
    bind *:80

 frontend workerhttps
    default_backend worker_https
    mode tcp
    bind *:443

#---------------------------------------------------------------------
# round robin balancing between the various backends
#---------------------------------------------------------------------

backend kube_api
    mode tcp
    balance     roundrobin
    option      ssl-hello-chk
    server bs bs.ocp49.example.localdomain:6443 check
    server m1 m1.ocp49.example.localdomain:6443 check
    server m2 m2.ocp49.example.localdomain:6443 check
    server m3 m3.ocp49.example.localdomain:6443 check
```

```
backend machine_config
    mode tcp
    balance     roundrobin
    server bs bs.ocp49.example.localdomain:22623 check
    server m1 m1.ocp49.example.localdomain:22623 check
    server m2 m2.ocp49.example.localdomain:22623 check
    server m3 m3.ocp49.example.localdomain:22623 check

backend worker_http
    mode tcp
    balance     source
    server w1 w1.ocp49.example.localdomain:80  check
    server w2 w2.ocp49.example.localdomain:80  check
    server w3 w3.ocp49.example.localdomain:80  check
    server i1 i1.ocp49.example.localdomain:80  check
    server i2 i2.ocp49.example.localdomain:80  check
    server i3 i3.ocp49.example.localdomain:80  check

backend worker_https
    mode tcp
    balance     source
    server w1 w1.ocp49.example.localdomain:443  check
    server w2 w2.ocp49.example.localdomain:443  check
    server w3 w3.ocp49.example.localdomain:443  check
    server i1 i1.ocp49.example.localdomain:443  check
    server i2 i2.ocp49.example.localdomain:443  check
    server i3 i3.ocp49.example.localdomain:443  check
```

DHCPサーバー

今回の環境では1台のDNSサーバー上にdhcpdを使って作成しています。DHCPサーバーは冗長化しても問題ありませんが、DHCPサーバーはクラスタインストール時に必要なだけで、その後はほぼ参照される機会がないため、本書ではシングル構成としています。

ここではdhcpdのセットアップおよび冗長化の手順は省略します。

■ノードとなるマシンの用意

Bootstrapノード、Masterノード、Workerノードのそれぞれに対応するマシンを作成します。OSがインストールされていないドライブを持つ物理マシンもしくは仮想マシンを必要な数だけ準備してください。前述のシステム構成図では、Bootstrapノードが1台、Masterノードが3台、Workerノードが3台、Infraノードが3台で、合わせて10台となります。

Masterノードで最小限のクラスタが作成された後に、Bootstrapノードとして使っていたマシンをWorkerノードに回すことで、台数を節約することも可能です。

6.3.3 OpenShiftインストーラー、コマンドラインツール、RHCOSの入手

作業用マシンで、OpenShiftインストーラー、コマンドラインツール、RHCOSのファイルをそれぞれダウンロードします。

Red Hat OpenShift Cluster Managerの「Create an OpenShift cluster」のページで、「Datacenter」のタブの下部にある「Platform agnostic (x86_64)」を選びます。

- Red Hat OpenShift Cluster Manager：Create an OpenShift cluster
  `https://console.redhat.com/openshift/create`

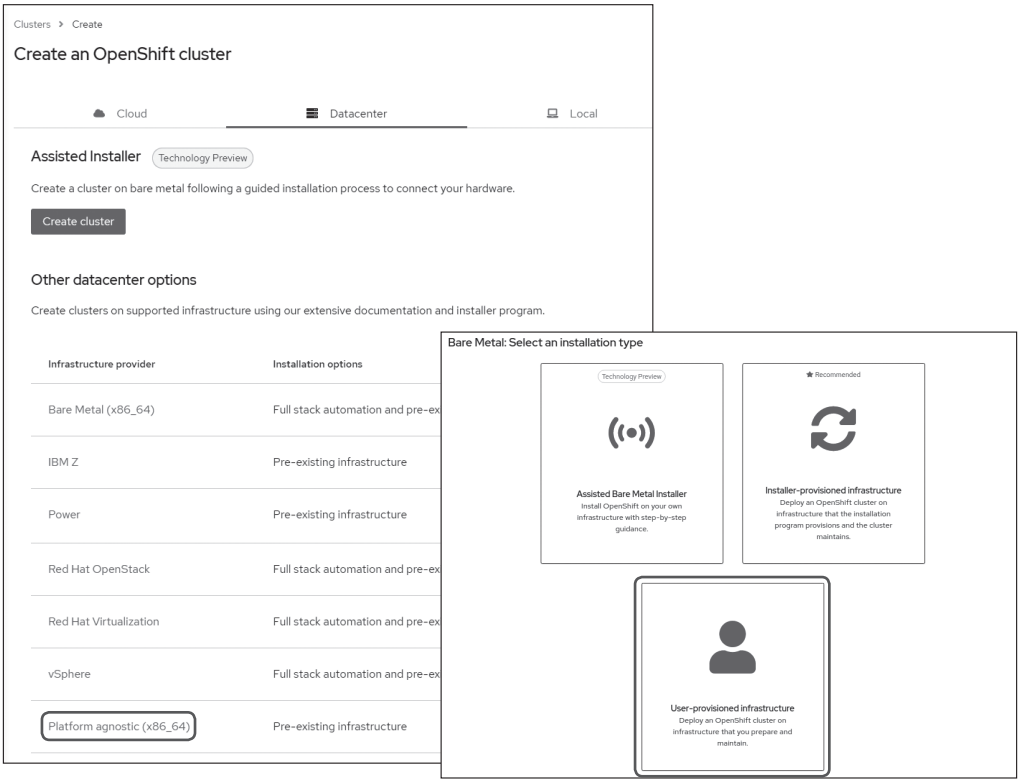


図6.4 OpenShiftインストーラーとコマンドラインツールのダウンロードページへのアクセス

表示されたページの、「① What you need to get started」にリストされているもののうち、以下の6つを作業用マシンにダウンロードします。

- OpenShift installer（作業用マシンのOSに合わせる）

- Pull Secret
- Command line interface（作業用マシンのOSに合わせる）
- RHCOS kernel
- RHCOS initramfs
- RHCOS rootfs

なお、これらの6つのファイルはダウンロードする時点で最新のバージョンのものがダウンロードされます。これにより、その時点で最新のOpenShiftクラスタがインストールされます。

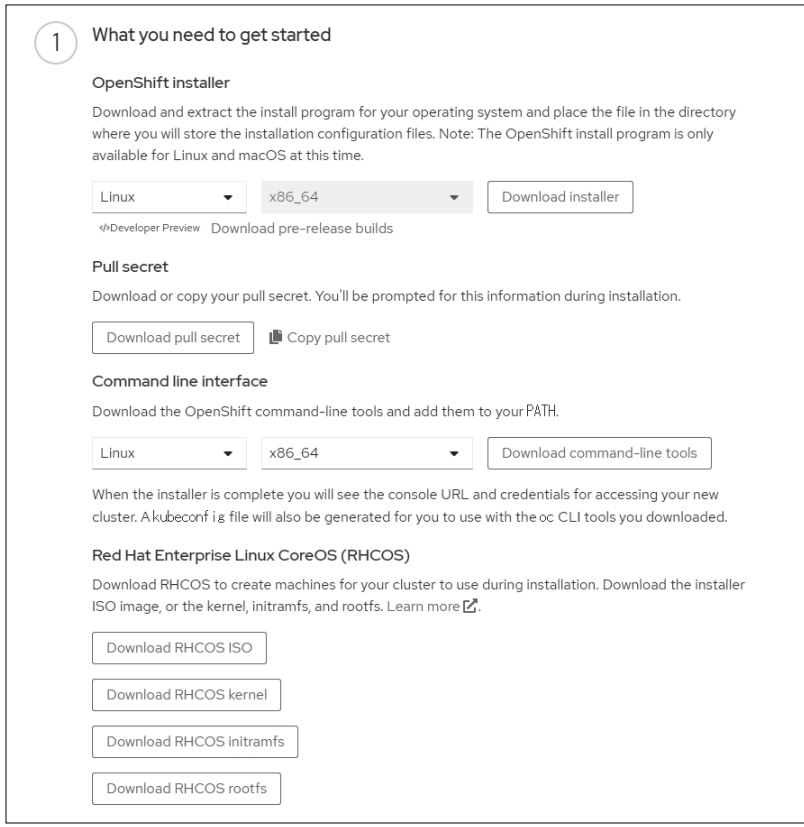


図6.5　OpenShiftインストーラー、コマンドラインツール、RHCOSのダウンロード

ダウンロードした「OpenShift installer」のファイルを解凍すると、`openshift-install`という実行ファイルが入手できます。「Command line interface」のファイルを解凍すると、`oc`と`kubectl`という2つの実行ファイルが入手できます。これらのファイルは以後何回も実行するので、パス指定なしで

実行できるように解凍したディレクトリにパスを通すか、パスが通っているディレクトリに移動しておくと便利です。

RHCOSのkernel、initramfs、rootfsの3つのファイルについては、HTTPサーバーで公開するディレクトリに配置しておきます。

COLUMN

過去のバージョンのOpenShiftクラスタを構築するには

事情によっては最新ではないバージョンのOpenShiftクラスタを構築したい場合があります。その場合は目的のバージョンに対応するインストーラー、コマンドラインツール、RHCOSを入手する必要があります。

過去のバージョンのファイルについては、下記のサイトにリストされているバージョン番号のリンク先からダウンロードできます。

- OpenShiftインストーラーとコマンドラインツール
  `https://mirror.openshift.com/pub/openshift-v4/x86_64/clients/ocp/`
  OpenShiftインストーラー：`openshift-install-<os>.tar.gz`
  コマンドラインツール：`openshift-client-<os>.<tar.gzまたはzip>`
- RHCOS
  `https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/`
  RHCOS kernel：`rhcos-live-kernel-<architecture>`
  RHCOS initramfs：`rhcos-live-initramfs-<architecture>`
  RHCOS rootfs：`rhcos-live-rootfs-<architecture>`

過去のバージョンのOpenShiftクラスタを構築する際は、サポートされるバージョンに注意してください。

6.3.4 Ignition Configファイルの作成

Bootstrapノード、Masterノード、Workerノード用のそれぞれのIgnition Configファイルを作成します。Ignition Configファイルは、`install-config.yaml`というYAMLから`openshift-install`コマンドを使って作成します。

■install-config.yamlの作成

作業用マシンでインストール用ディレクトリを作成し、その中に以下のような`install-config.yaml`ファイルを作成します。

▶ **install-config.yaml**

```
# mkdir -p ~/openshift && cd ~/openshift
# cat << EOF > ~/openshift/install-config.yaml
apiVersion: v1
baseDomain: example.localdomain ──────────────────────────────── ❶
compute: ──────────────────────────────────────────────────────── ❷
- hyperthreading: Enabled ─────────────────────────────────────── ❸
  name: worker
  replicas: 0 ──────────────────────────────────────────────────── ❹
controlPlane: ─────────────────────────────────────────────────── ❷
  hyperthreading: Enabled ─────────────────────────────────────── ❸
  name: master
  replicas: 3 ──────────────────────────────────────────────────── ❺
metadata:
  name: ocp49 ──────────────────────────────────────────────────── ❻
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14 ──────────────────────────────────────── ❼
    hostPrefix: 23 ──────────────────────────────────────────── ❽
  networkType: OpenShiftSDN ──────────────────────────────────── ❾
  serviceNetwork:
  - 172.30.0.0/16 ──────────────────────────────────────────── ❿
platform:
  none: {} ──────────────────────────────────────────────────── ⓫
fips: false ──────────────────────────────────────────────────── ⓬
pullSecret: '{"auths": ...}' ──────────────────────────────────── ⓭
sshKey: 'ssh-ed25519 AAAA...' ──────────────────────────────────── ⓮

EOF
```

❶ ベースとなるドメイン名を指定する。

❷ compute セクションの頭には -（ハイフン）が必要だが、controlPlane セクションには -（ハイフン）は必要ない。

❸ SMT を使用する場合は Enable を指定する。

❹ Worker ノードの数は、この時点では 0 を指定しておき、あとから Worker ノードを手動で展開する。

❺ Master ノードの数は 3 を指定する。

❻ 作成されるクラスタの名前を指定する。❶のベースドメインのサブドメイン名として使用される。

❼ Cluster Network の CIDR を指定する。

❽ 各ノードにアサインされるサブネットの Prefix を指定する。

❾ クラスタで使用する CNI（Container Network Interface）を指定する。デフォルトでは OpenShiftSDN が指定される。

❿ Service Network の CIDR を指定する。

⓫ Bare metal インストールおよび Any Platform インストールでは none を指定する。

⓬ FIPS モードの有効化を指定する。デフォルトは false が指定される。

⓭「...」の部分に、OpenShift インストーラーと一緒にダウンロードした Pull Secret の中身を転記する。

⓮ 各ノードの RHCOS と SSH で通信するための公開鍵を転記する。

■マニフェストファイルの作成

作成したinstall-config.yamlから、マニフェストファイルを作成します。

任意の名前のインストール用のディレクトリを作って、先ほど作成したinstall-config.yamlを作ったディレクトリにコピーします。以下の例では、installdirという名前のディレクトリを作成しています。

インストールディレクトリにコピーしたinstall-config.yamlファイルは、以降の作業で消えてしまうので、オリジナルのinstall-config.yamlは再利用できるように保管しておきましょう。

```
# mkdir -p ~/openshift/installdir
# cp ~/openshift/install-config.yaml ~/openshift/installdir/
```

次のコマンドでマニフェストファイルが作成されます。

```
# cd ~/openshift
# openshift-install create manifests --dir=installdir
INFO Consuming Install Config from target directory
INFO Manifests created in: installdir/manifests and installdir/openshift
```

インストールディレクトリの下に新しいディレクトリが作成されているのを確認します。このディレクトリの中にインストール用のマニフェストファイル（YAMLファイル）が生成されているはずです。確認してみてください。

```
# ls -ltr installdir
total 8
drwxr-x---. 2 root root 4096 Nov 09 15:28 manifests
drwxr-x---. 2 root root 4096 Nov 09 15:28 openshift
# ls -ltr installdir/manifests
total 64
-rw-r-----. 1 root root  169 Nov 09 15:28 openshift-kubevirt-infra-namespace.yaml
-rw-r-----. 1 root root 3833 Nov 09 15:28 openshift-config-secret-pull-secret.yaml
...
# ls -ltr installdir/openshift
total 24
-rw-r-----. 1 root root 1185 Nov 09 15:28 99_openshift-machineconfig_99-master-ssh.yaml
-rw-r-----. 1 root root 2466 Nov 09 15:28 99_openshift-cluster-api_master-user-data-secret.yaml
...
```

これらのマニフェストファイルを編集することで、OpenShiftクラスタの構成を変更できます。たとえば、デフォルトではMasterノードにはユーザーアプリケーションのPodをスケジュールできませんが、installdir/manifests/cluster-scheduler-02-config.ymlを編集することでMasterノードにもユーザーアプリケーションのPodをスケジュールできます。

ここでは特に編集は行わず、デフォルトのまま次に進みます。

■Ignition Configファイルの作成

作成したマニフェストファイルから、Ignition Configファイルを作成します。

次のコマンドでIgnition Configファイルが作成されます。

```
# cd ~/openshift
# openshift-install create ignition-configs --dir=installdir
INFO Consuming Openshift Manifests from target directory
INFO Consuming Worker Machines from target directory
INFO Consuming Master Machines from target directory
INFO Consuming Common Manifests from target directory
INFO Consuming OpenShift Install (Manifests) from target directory
INFO Ignition-Configs created in: installdir and installdir/auth
```

インストールディレクトリの中に、Bootstrapノード、Masterノード、Workerノードのそれぞれの Ignition Configファイル(.ign)が作成されていることがわかります。作成されたIgnition Configファイルは、HTTPサーバーで公開するディレクトリに配置しておきましょう。

```
# cd ~/openshift
# tree installdir
installdir
├── auth
│   ├── kubeadmin-password
│   └── kubeconfig
├── bootstrap.ign
├── master.ign
├── metadata.json
└── worker.ign

1 directory, 6 files
```

Ignition Configファイルと同時に、作成するOpenShiftクラスタのkubeadminユーザーパスワードやコンテキストなど、クラスタの認証に関わるファイルとディレクトリ（auth）も作成されます。

kubeconfigファイルはクラスタが作成されているかocコマンドで確認する際に使われます。このタイミングでexportコマンドでKUBECONFIGという環境変数を宣言しておきます。

```
# export KUBECONFIG=~openshift/installdir/auth/kubeconfig
```

Ignition Configファイルの期限切れ

作成されたIgnition Configファイルには、X509形式の証明書が含まれています。この証明書の期限は24時間になっていて、24時間を超えると期限切れとなり再利用することができま

せん。正確に言えば、証明書が期限切れのIgnition Configファイルを使ってノードを起動することはできますが、認証が通らないためOpenShiftクラスタを構成できません。

Ignition Configファイルが作成後24時間を超えてしまった場合は、再作成してHTTPサーバーにアップロードすることを忘れないようにしましょう。

6.3.5 Bootstrapノード／Masterノードの作成

それではノードを起動してみましょう。

まずはBootstrapノードとMasterノードをすべて起動します。起動の順序やタイミングに制限はないので、Bootstrapノードと3台のMasterノードのサーバーを順に起動します。iPXEとIgnition Configファイルが適切に用意されていれば、サーバーをパワーオンするだけで何もしなくても自動的に起動してきます。

BootstrapノードとMasterノードで画面を確認して、ログインコンソールが図6.6のように表示されていれば起動はできています。エラーメッセージが表示されることがありますが、ここでは無視して問題ありません。

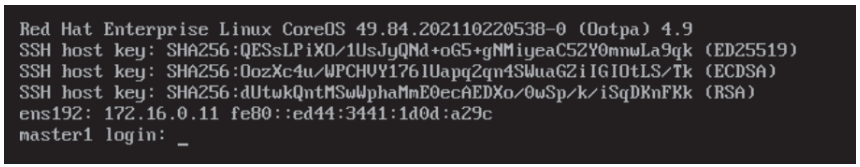


図6.6 Bootstrapノード/Masterノードのログインコンソール

Bootstrapノードと3台のMasterノードの起動が確認できたら、作業用マシンで次のコマンドを実行してBootstrapのプロセスをモニターします。このコマンドが終了すればBootstrapのプロセスは完了し、Bootsrtapノードの作成とそれをもとにしたMasterノードの作成までができています。

```
# openshift-install wait-for bootstrap-complete --dir=installdir  --log-level=debug
DEBUG OpenShift Installer 4.9.5
DEBUG Built from commit 8223216bdcef5de56b52240ab7160ca909a9e56c
INFO Waiting up to 20m0s for the Kubernetes API at https://api.ocp49.example.localdomain:6443...
INFO API v1.22.0-rc.0+a44d0f0 up
INFO Waiting up to 30m0s for bootstrapping to complete...
DEBUG Bootstrap status: complete
INFO It is now safe to remove the bootstrap resources
DEBUG Time elapsed per stage:
DEBUG Bootstrap Complete: 13m12s
DEBUG                API: 1m58s
INFO Time elapsed: 13m12s
```

Bootstrapのプロセスが完了すればBootstrapノードの役目は終了です。コマンド出力に「INFO It is now safe to remove the bootstrap resources」というメッセージが出ていれば、Bootstrapノードを削除してもかまいません。

6.3.6 Workerノード／Infraノードの作成

次にWorkerノードを起動します。InfraノードはWorkerノードの一種なので、一緒に起動することにします。こちらも先ほどと同様に、Workerノードのサーバーを順にパワーオンすれば自動的に起動していきます。BootstrapノードやMasterノードと同じように、Workerノードの画面でログインコンソールが表示されることを確認しましょう。エラーメッセージが表示されることがありますが、ここでは無視して問題ありません。

すべてのWorkerノードが起動したら、WorkerノードのCSR（Certificate Signing Request）に署名をしていきます。CSRは証明書のもとになる情報への署名要求で、承認することで署名されて証明書になります。これらのCSRはWorkerノードに紐づくもので、WorkerノードがOpenShiftクラスタに参加する際に必要となるものです。

作業用マシンでoc get csrコマンドを実行して、CONDITIONがPendingになっているものを探します。

```
# oc get csr
NAME                                                  AGE   SIGNERNAME                                    REQUESTOR
                                                                           REQUESTEDDURATION   CONDITION         NAME
csr-5pc2p                                             64m   kubernetes.io/kubelet-serving                 system:node:m1
.ocp49.example.localdomain                                                 <none>              Approved,Issued
csr-b4qrw                                             64m   kubernetes.io/kubelet-serving                 system:node:m2
.ocp49.example.localdomain                                                 <none>              Approved,Issued
csr-c897d                                             20m   kubernetes.io/kube-apiserver-client-kubelet   system:service
account:openshift-machine-config-operator:node-bootstrapper                <none>              Pending
csr-f8qtn                                             64m   kubernetes.io/kubelet-serving                 system:node:m3
.ocp49.example.localdomain                                                 <none>              Approved,Issued
csr-gcb44                                             64m   kubernetes.io/kube-apiserver-client-kubelet   system:service
account:openshift-machine-config-operator:node-bootstrapper                <none>              Approved,Issued
csr-hgj62                                             21m   kubernetes.io/kube-apiserver-client-kubelet   system:service
account:openshift-machine-config-operator:node-bootstrapper                <none>              Pending
csr-hx6q4                                             21m   kubernetes.io/kube-apiserver-client-kubelet   system:service
account:openshift-machine-config-operator:node-bootstrapper                <none>              Pending
csr-kw65q                                             21m   kubernetes.io/kube-apiserver-client-kubelet   system:service
account:openshift-machine-config-operator:node-bootstrapper                <none>              Pending
csr-m26ks                                             64m   kubernetes.io/kube-apiserver-client-kubelet   system:service
account:openshift-machine-config-operator:node-bootstrapper                <none>              Approved,Issued
csr-mtt6q                                             20m   kubernetes.io/kube-apiserver-client-kubelet   system:service
account:openshift-machine-config-operator:node-bootstrapper                <none>              Pending
csr-nffzk                                             20m   kubernetes.io/kube-apiserver-client-kubelet   system:service
```

```
account:openshift-machine-config-operator:node-bootstrapper         <none>              Pending
csr-pvcwl                                          64m   kubernetes.io/kube-apiserver-client-kubelet   system:service
account:openshift-machine-config-operator:node-bootstrapper         <none>              Approved,Issued
system:openshift:openshift-authenticator-vcv8f     62m   kubernetes.io/kube-apiserver-client           system:service
account:openshift-authentication-operator:authentication-operator   <none>              Approved,Issued
system:openshift:openshift-monitoring-phsxc        62m   kubernetes.io/kube-apiserver-client           system:service
account:openshift-monitoring:cluster-monitoring-operator            <none>              Approved,Issued
```

ここでは6つのCSRがPendingと表示されていますが、3台のWorkerノードと3台のInfraノードの合計6ノードに対応しています。Workerノードの準備が整うと必ず表示されるものなので、もしPendingが表示されていない場合はしばらく待ってから再度コマンドを実行してみてください。

Pending状態のCSRを承認（approve）するには、oc adm certificate approveコマンドを実行します。1つずつ実行してもよいのですが、次のようにxargsコマンドを使って一気に承認することもできます。

```
# oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}{{end}}
{{end}}' | xargs oc adm certificate approve
certificatesigningrequest.certificates.k8s.io/csr-c897d approved
certificatesigningrequest.certificates.k8s.io/csr-hgj62 approved
certificatesigningrequest.certificates.k8s.io/csr-hx6q4 approved
certificatesigningrequest.certificates.k8s.io/csr-kw65q approved
certificatesigningrequest.certificates.k8s.io/csr-mtt6q approved
certificatesigningrequest.certificates.k8s.io/csr-nffzk approved
```

承認が完了すると、今度はすぐに別のPendingのCSRが出てきます。1つのWorkerノードに対して、クライアント・リクエストCSRとサーバー・リクエストCSRの、2つのCSRの承認を要求されます。Pending表示がなくなるまで、oc adm certificate approveコマンドを繰り返し実行し、承認していってください。

CSRをすべて承認したら、oc get nodesコマンドでWorkerノードの状態（STATUS）がReadyになっているのを確認できます。

```
# oc get nodes
NAME                           STATUS   ROLES    AGE   VERSION
i1.ocp49.example.localdomain   Ready    worker   10m   v1.22.0-rc.0+a44d0f0
i2.ocp49.example.localdomain   Ready    worker   10m   v1.22.0-rc.0+a44d0f0
i3.ocp49.example.localdomain   Ready    worker   10m   v1.22.0-rc.0+a44d0f0
m1.ocp49.example.localdomain   Ready    master   75m   v1.22.0-rc.0+a44d0f0
m2.ocp49.example.localdomain   Ready    master   75m   v1.22.0-rc.0+a44d0f0
m3.ocp49.example.localdomain   Ready    master   75m   v1.22.0-rc.0+a44d0f0
w1.ocp49.example.localdomain   Ready    worker   10m   v1.22.0-rc.0+a44d0f0
w2.ocp49.example.localdomain   Ready    worker   10m   v1.22.0-rc.0+a44d0f0
w3.ocp49.example.localdomain   Ready    worker   10m   v1.22.0-rc.0+a44d0f0
```

6.3.7 クラスタ作成の確認

あとはOpenShiftクラスタが作成されるのを待つだけです。作業用マシンで次のコマンドを実行してクラスタインストールのプロセスをモニターします。このコマンドが終了すればクラスタ完成です。

```
# openshift-install wait-for install-complete --dir=installdir
INFO Waiting up to 40m0s for the cluster at https://api.ocp49.example.localdomain:6443 to initial
ize...
INFO Waiting up to 10m0s for the openshift-console route to be created...
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/root
/openshift/installdir/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.ocp49.example.
localdomain
INFO Login to the console with user: "kubeadmin", and password: "NqrXg-pdjy9-2c2er-MYTzr"
INFO Time elapsed: 30m24s
```

クラスタインストールのプロセスではCluster Operatorのセットアップが行われており、これらのOperatorのAVAILABLEがすべてTrueになるまで待つ必要があります。Cluster Operatorは、oc get clusteroperatorsコマンドを使って状態を確認することができます。

```
# oc get clusteroperators
NAME                                VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
authentication                      4.9.5     True        False         False      5m8s
baremetal                           4.9.5     True        False         False      93m
cloud-controller-manager            4.9.5     True        False         False      97m
cloud-credential                    4.9.5     True        False         False      98m
cluster-autoscaler                  4.9.5     True        False         False      95m
config-operator                     4.9.5     True        False         False      96m
console                             4.9.5     True        False         False      9m32s
csi-snapshot-controller             4.9.5     True        False         False      95m
dns                                 4.9.5     True        False         False      93m
etcd                                4.9.5     True        False         False      94m
image-registry                      4.9.5     True        False         False      87m
ingress                             4.9.5     True        False         False      11m
insights                            4.9.5     True        False         False      88m
kube-apiserver                      4.9.5     True        False         False      90m
kube-controller-manager             4.9.5     True        False         False      90m
kube-scheduler                      4.9.5     True        False         False      94m
kube-storage-version-migrator       4.9.5     True        False         False      95m
machine-api                         4.9.5     True        False         False      95m
machine-approver                    4.9.5     True        False         False      94m
machine-config                      4.9.5     True        False         False      94m
marketplace                         4.9.5     True        False         False      95m
monitoring                          4.9.5     True        False         False      10m
network                             4.9.5     True        False         False      95m
```

```
node-tuning                                4.9.5     True        False         False      94m
openshift-apiserver                        4.9.5     True        False         False      87m
openshift-controller-manager               4.9.5     True        False         False      87m
openshift-samples                          4.9.5     True        False         False      87m
operator-lifecycle-manager                 4.9.5     True        False         False      94m
operator-lifecycle-manager-catalog         4.9.5     True        False         False      94m
operator-lifecycle-manager-packageserver   4.9.5     True        False         False      87m
service-ca                                 4.9.5     True        False         False      95m
storage                                    4.9.5     True        False         False      96m
```

通常は何もしなくてもすべてのOperatorでAVAILABLEがTrueと表示されるのですが、いつまで経ってもAVAILABLEがTrueにならないことがあります。この場合、該当するOperatorが管理するPodが稼働していないなどの問題が発生している可能性があります。対処としては、次の実行例のようにoc get pod --all-namespacesコマンドを使って、RunningまたはCompletedのステータス以外のPodを見つけ出し、oc logsコマンドやoc get eventsコマンドなどで原因を調べて対処するようにしてください。

```
# oc get pod --all-namespaces | grep -v Running | grep -v Completed
NAMESPACE                                          NAME                                                ➥
             READY   STATUS        RESTARTS   AGE
openshift-console                                  console-86bfd57fc9-k8sbm                            ➥
             1/1     Terminating   0          3m52s
openshift-kube-apiserver                           kube-apiserver-m1.ocp49.example.local               ➥
             0/4     Init:0/1      0          54s
```

クラスタのインストールが完了したら、クラスタのWebコンソールにアクセスしてみましょう。kubeadminユーザーでクラスタにログインできれば成功です。

WebコンソールのURLとkubeadminユーザーのパスワードは、openshift-install wait-for install-completeコマンドの出力に表示されています。これらがわからなくなってしまったら、再度openshift-install wait-for install-completeコマンドを実行するか、作業用マシンのインストールディレクトリに保存されるインストールログファイルを確認するとわかります。次の実行例では、tailコマンドでインストールログファイルを表示しています。出力で太字にしている部分がユーザー名とパスワードです。

```
# tail installdir/.openshift_install.log
time="2021-11-10T02:27:10+09:00" level=info msg="Waiting up to 40m0s for the cluster at https://a➥
pi.ocp49.example.localdomain:6443 to initialize..."
time="2021-11-10T02:27:10+09:00" level=debug msg="Cluster is initialized"
time="2021-11-10T02:27:10+09:00" level=info msg="Waiting up to 10m0s for the openshift-console ro➥
ute to be created..."
time="2021-11-10T02:27:10+09:00" level=debug msg="Route found in openshift-console namespace: con➥
sole"
```

```
time="2021-11-10T02:27:10+09:00" level=debug msg="OpenShift console route is admitted"
time="2021-11-10T02:27:10+09:00" level=info msg="Install complete!"
time="2021-11-10T02:27:10+09:00" level=info msg="To access the cluster as the system:admin user w
hen using 'oc', run 'export KUBECONFIG=/root/install-bm/auth/kubeconfig'"
time="2021-11-10T02:27:10+09:00" level=info msg="Access the OpenShift web-console here: https://c
onsole-openshift-console.apps.ocp49.example.localdomain"
time="2021-11-10T02:27:10+09:00" level=info msg="Login to the console with user: \"kubeadmin\", a
nd password: \"NqrXg-pdjy9-2c2er-MYTzr\""
time="2021-11-10T02:27:10+09:00" level=info msg="Time elapsed: 30m24s"
```

6.3.8 クラスタ完成後の作業

以上でOpenShiftクラスタのUPIインストールは完成です。この時点でアプリケーションPodを稼働させることも可能です。しかし現実的にコンテナプラットフォームとして使えるようになるには、永続ストレージを用意したり、クラスタアプリケーションをInfraノードに移動させたり、さらに作業を行うことになります。

ここではこれらの作業の詳細については触れません。本節の冒頭で紹介したGitHubのサイト（図6.3の上部）に手順が記載されているので、そちらを参考にして実施してください。

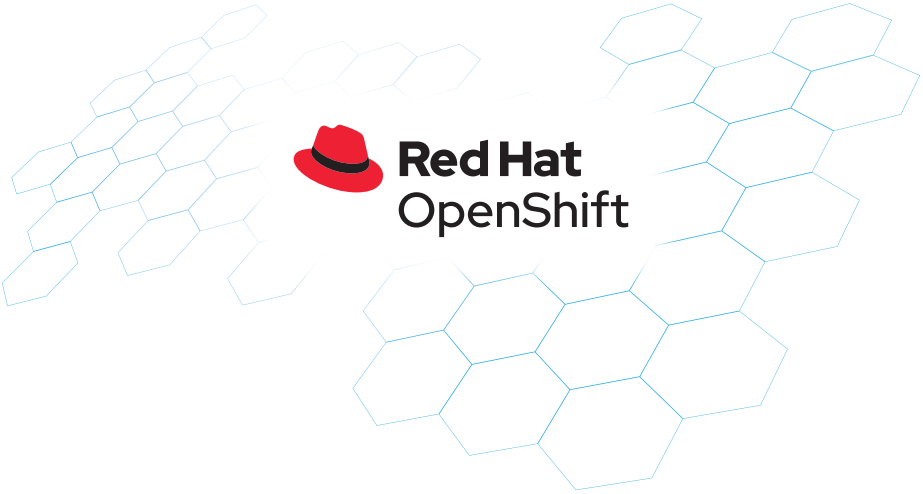
エアギャップ環境への対応

OpenShiftクラスタのインストールは、インターネットに接続できる環境で行われることが想定されています。これは、作業用マシンで各種ファイルをダウンロードするだけでなく、クラスタ内でCluster Operatorが稼働させるアプリケーションのコンテナイメージを、registry.redhat.ioやquay.ioといったインターネット上のコンテナレジストリからPullするために必要だからです。

しかし、場合によってはインターネットに接続できない、いわゆるエアギャップ環境（隔離環境）でOpenShiftクラスタをインストールすることもあるでしょう。そのような場合は、環境内にローカルレジストリを作成してクラスタインストールすることができます。この場合、インターネット上のコンテナレジストリからローカルレジストリにミラーするため、ローカルレジストリが稼働するサーバーだけはインターネットに接続できる必要があるので注意してください。一度ミラーしてしまえばインターネット接続は切断してもかまいません。

具体的な手順については、OCPのオンラインドキュメントの「Mirroring images for a disconnected installation」のページを参照してください。

- OpenShift Container Platform 4.9 Documentation; Mirroring images for a disconnected installation
  https://docs.openshift.com/container-platform/4.9/installing/installing-mirroring-installation-images.html



企 画　レッドハット株式会社
著 者　株式会社リアルグローブ・オートメーティッド、青島 秀治、レッドハット株式会社、
森 真也、清水 護、織 学、宇都宮 卓也、斎藤 和史、野間 亮志、荒木 俊博
監 修　須江 信洋
URL　https://www.redhat.com/ja/global/japan
問い合わせ　https://www.redhat.com/ja/contact

※本冊子は下記よりPCやタブレットで閲覧、ダウンロードできます。
https://tracks.redhat.com/openshiftmook2022