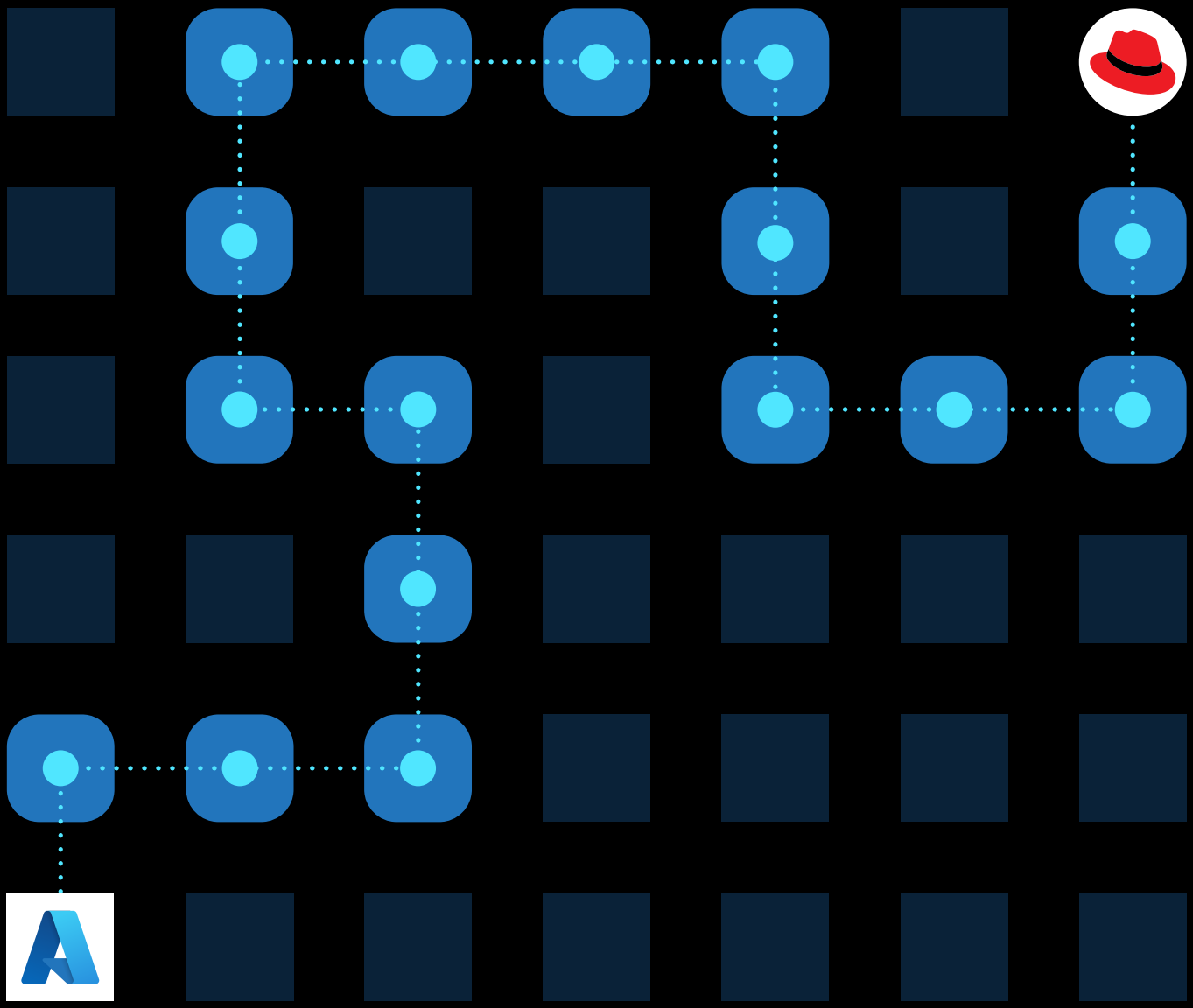
e ブックシリーズ

Microsoft Azure

Azure Red Hat OpenShift の使用を開始する

PoC からプロダクションまで



Azure Red Hat OpenShift の使用を開始する

第 1 章

マイクロソフトと Red Hat へのお問い合わせ

Azure Red Hat OpenShift をご検討中のお客様は、ぜひご相談ください。このガイドでは Azure Red Hat OpenShift の使用法について役立つ情報を紹介していますが、Red Hat とマイクロソフトは、それ以上のエクスペリエンスをもたらす多くのリソースを提供できます。Azure Red Hat OpenShift がお客様のアプリケーションにおけるイノベーションのニーズを満たすアプリケーション・プラットフォームになるよう、両社は連携しています。

Azure Red Hat OpenShift はただ 1 つの単純な理由、つまりお客様からの要望から生まれました。両社の共同顧客は、これまでになく Red Hat のポートフォリオを Microsoft Azure にデプロイしています。この傾向は企業規模の大小に関係なく見られます。自己管理型のオファリングである OpenShift on Azure は数年前から完全にサポートされていますが、クラスタの管理におけるセットアップ、デプロイ、および Day 2 オペレーションには、Kubernetes の専門知識と管理するための時間が必要です。そしてそれにより、ビジネス目標の達成に使うべき貴重な時間が別のことに消費されてしまいます。

マネージド型のオファリングである Azure Red Hat OpenShift を使用して思い通りにデプロイを進めるお客様は増加しており、そのようなお客様においては、セットアップと Day 2 オペレーションに費やす時間が大幅に短縮され、アプリケーションに集中する時間が増えています。

このガイドは実践的なエクスペリエンスに基づいて作成されており、Azure Red Hat OpenShift でのアプリケーション構築におけるベストプラクティスについて説明します。これはセルフヘルプガイドです。前から順番に (「入門」から「総まとめ」まで) 読むこともできますし、具体的に知りたい情報があればそれを探して読むこともできます。ご自由にご利用ください。

このガイドの対象者

このガイドは、Azure と Red Hat OpenShift を活用して、フルマネージド Red Hat OpenShift クラスタのフルサービスをデプロイすることにより、アプリケーションの構築およびデプロイメント機能を強化したいと考えている技術者 (開発者、運用者、プラットフォーム・アーキテクト) を対象としています。

このガイドの内容

このガイドでは、組織内での PoC からプロダクションへのデプロイまで、Azure Red Hat OpenShift の理解と導入に必要な、重要なトピックについて説明します。

第 2 章： Red Hat OpenShift の概要では、まず、Red Hat OpenShift の概要、Red Hat OpenShift がアプリケーション・プラットフォームとして非常に多くの開発者、運用者、プラットフォーム・アーキテクトに選ばれる理由、そして、彼らが Red Hat OpenShift からどのようにして多くの有用性を得ているのかについて説明します。その後、Red Hat OpenShift が推奨されるプラットフォームである理由を説明します。

第 3 章： Azure Red Hat OpenShift では、マネージドサービスと、それがお客様にどのように提供されるのかについて説明します。

第 4 章： プロビジョニングの準備 – エンタープライズ・アーキテクチャに関する考慮事項では、Azure Red Hat OpenShift のデプロイ前に考慮する必要がある重要な事柄について説明します。実際に検討を行った多くのお客様との対話から見えてきた多くのベストプラクティスについてのガイダンスも提供します。

第 5 章： Azure Red Hat OpenShift クラスタのプロビジョニングでは、公式ドキュメントにある、Azure Red Hat OpenShift クラスタのデプロイに必要な主要なリソースを紹介します。

第 6 章： プロビジョニングの後処理 - Day 2 では、しばしば「Day 2」と呼ばれるプロビジョニング後のタスクについて説明します。このガイドでは、マネージドサービスである Azure Red Hat OpenShift を使用することが可能な場合はそのことに言及し、お客様ご自身で対応するよりもどれだけ容易になるかを説明します。

第 7 章： Red Hat OpenShift へのアプリケーションのデプロイは、プラットフォームでのアプリケーションのデプロイに関する簡単なガイドです。ただし、デプロイ方法については他の環境で稼働している Red Hat OpenShift の場合と変わりません。

第 8 章： アプリケーション・プラットフォームの機能紹介では、アプリケーション・プラットフォームのいくつかの主要な機能 (コンテナレジストリ、パイプライン、サーバーレスなど) の概要を説明します。

第 9 章： 他のサービスとの統合では、Azure Service Operator、Azure DevOps、および同様のサービスを使用したプラットフォーム統合について説明します。

第 10 章： ワークロードとチームのオンボーディングでは、アプリケーションチームを立ち上げ、組織内でのサービスの導入を促進するためのベストプラクティスと推奨事項について説明します。

第 11 章： 総まとめは総括です。

第 12 章： 用語集では、各用語について解説します。

第 2 章

Red Hat OpenShift 入門

本章では、Red Hat OpenShift とは何か、その使用方法、そして、このサービスを使用する Red Hat のお客様が通常得られるメリットについて簡単に説明します。これは、初めて OpenShift に関する説明を受ける方にとっては便利な入門書として、このプラットフォームについてもう少し詳しく知りたいと思っている方にとっては短時間で学びなおすための資料として役立ちます。

Red Hat OpenShift の概要

Red Hat OpenShift は、エンタープライズ対応のアプリケーション・プラットフォームとして、ハイブリッドクラウドやマルチクラウドのデプロイメントを管理するフルスタックの自動運用機能を提供します。また、開発者の生産性を向上させ、イノベーションを促進できるように最適化されています。Red Hat OpenShift は、自動運用機能と効率化されたライフサイクル管理によって、開発チームによる新規アプリケーションの構築とデプロイを可能にし、運用チームによる Kubernetes プラットフォームのプロビジョニング、管理、スケーリングを支援します。

開発チームは、数百ものパートナーが提供するイメージとソリューションを利用できます。これらは検証済みで、提供プロセス全体を通じてセキュリティスキャンと暗号化された署名が行われます。オンデマンドイメージへのアクセスも、さまざまなサードパーティのクラウドサービスへのネイティブアクセスも、単一のプラットフォームから行うことが可能です。

ロギング機能とモニタリング機能が組み込まれているため、運用チームはどこにいても、チーム全体でデプロイメントを可視化できます。Red Hat Kubernetes Operator は、サービスを機能させる独自のアプリケーションロジックを搭載しており、単に構成するだけではなく、パフォーマンスに応じて調整し、OS からワンタッチで更新やパッチを適用することができます。つまり、Red Hat OpenShift は、IT チームと開発チームの力を最大限に発揮できるようにするワンストップショップです。

OpenShift クラウドサービス – コスト削減とビジネス上のメリット

Red Hat OpenShift に信頼を寄せる組織は数千におよびます。そのようなお客様は 、アプリケーションの提供方法を変更し、顧客との関係を改善し、競争上の優位性を獲得して業界のリーダーになっています。IDC の調査 (Red Hat OpenShift のビジネス価値 (2012 年 3 月) で概説) によると、調査対象の組織では、Red Hat OpenShift をプラットフォームとして使用することで、組織のビジネスに対して品質の高いアプリケーションや機能をより迅速に提供すると同時に、開発および IT 関連のコストとスタッフの時間的要件を最適化し、強力な価値を得ています。

主な指標は次のとおりです。

- 5 年間の投資対効果 (ROI) が 636%
- 投資回収期間が 10 カ月
- 5 年間の運用コストを 54% 削減
- 年間に提供される新機能の数が 3 倍に増加
- アプリケーション開発者の生産性が 20% 向上
- 予定外のダウンタイムが 71% 短縮
- IT インフラストラクチャチームの効率が 21% 向上

出典：IDC ホワイトペーパー (Red Hat 後援)、Red Hat OpenShift のビジネス価値、Doc # US47539121、2021 年 3 月

この Red Hat OpenShift の価値に加えて、Azure Red Hat OpenShift を含む Red Hat OpenShift クラウドサービスは、さらにビジネス上のメリットを提供します。Forrester による調査、「Red Hat OpenShift クラウドサービスについての Total Economic Impact™ - コスト削減とビジネス上のメリット」では、いくつかの重要な経済的メリットが紹介されています。

OpenShift クラウドサービスの主な経済的メリットは次のとおりです。

- 投資対効果 (ROI) が 468%
- 正味現在価値 (NPV) が 408 万ドル
- 投資回収期間が 6 カ月

これらの経済的メリットに加え、Forrester の調査で判明した主な定量的メリットは次のとおりです。

- **開発速度の向上**：Red Hat OpenShift クラウドサービスを使用すると、組織は開発サイクルを最大 70% 短縮できます。
- **インフラストラクチャの保守作業が不要なため、開発者の作業時間の 20% を回復**：調査対象の組織は、Red Hat OpenShift クラウドサービスにより、開発者がアプリケーション開発インフラストラクチャを保守する必要がなくなり、製品やソリューションの構築に完全に集中できるようになったと述べました。このように回復された開発者の時間には、3 年間で 230 万ドル以上の価値があります。
- **運用効率が 50% 向上**：調査対象の組織は、Red Hat OpenShift クラウドサービスはマネージドサービスであり、このソリューションを使用することで、以前はインフラストラクチャの管理を担当していた DevOps スタッフの 50% を、より生産性の高い他の作業に配置転換できると述べました。この運用効率の向上には、3 年間で 130 万ドル以上の価値があります。*

このレポートでは、数字には表れない次のようなメリットも明らかになりました。

- **開発者の満足度と定着**：調査対象の組織は、開発者が Red Hat OpenShift クラウドサービスのメリットを享受することで、更新を細分化できるようになり、非常に限られたスケジュールで広範なテストを行わなければならないというプレッシャーや、プロダクションで無駄な作業に対応する必要性が減ることを強調しています。
- **セキュリティとリスク軽減**：調査対象の組織は、Red Hat OpenShift クラウドサービスが一部の機能とセキュリティ更新を自動化するため、手作業による保守が不要になり、環境のセキュリティも確保できると述べました。
- **信頼性**：調査対象の組織は、Red Hat OpenShift クラウドサービスを使用すると、環境が拡大している場合でも停止やシステム障害が少なくなるため、長期的にはアプリケーション・プラットフォームの信頼性が向上することを指摘しました。
- **可搬性とビジネス継続性**：調査対象の組織は、Red Hat OpenShift クラウドサービスは可搬性、スケーラビリティ、柔軟性を備えているため、ビジネス継続性を確保し、障害復旧戦略を支援することにも言及しました。

出典：Forrester、Red Hat OpenShift クラウドサービスの Total Economic Impact、2021 年 12 月

***参考資料：**クラウドサービスでアジリティを加速する企業

「Red Hat OpenShift か、素の Kubernetes か」– 独自に Kubernetes アプリケーション・プラットフォームを構築する際のコスト

Red Hat OpenShift は、しばしば「エンタープライズ Kubernetes」と呼ばれますが、それが実際に何を意味するのかをすぐに理解するのは難しいかもしれません。お客様から「OpenShift と素の Kubernetes、どちらにするべきでしょうか」と質問されることがよくあります。しかし、**OpenShift はすでに Kubernetes を使用している**ことを理解することが重要です。Kubernetes は、OpenShift アーキテクチャ全体で、OpenShift プラットフォームが構築される基盤と、OpenShift を実行するための多くのツールを提供します。

Kubernetes は極めて重要なオープンソース・プロジェクトです。Cloud Native Computing Foundation の主要プロジェクトの 1 つであり、コンテナの実行に不可欠なテクノロジーです。

しかし、OpenShift の利用を検討する人たちが抱く本当の疑問は、おそらく「**Kubernetes だけでアプリケーションを実行できるのか**」ということでしょう。多くの組織は、Kubernetes をデプロイすると、コンテナ、さらにはエンタープライズ・アプリケーションを実行するところまではわずか数日で到達できることに気づきます。しかし、Day 2 オペレーションが開始され、セキュリティ要件が発生し、より多くのアプリケーションをデプロイするようになると、一部の組織は、Kubernetes テクノロジーを使用して独自の **PaaS (Platform-as-a-Service)** を構築しようという考えに陥ります。彼らはオープンソースの Ingress コントローラーを追加し、**継続的インテグレーション/継続的デプロイメント (CI/CD)** パイプラインに接続するためのスクリプトをいくつか作成してから、より複雑なアプリケーションをデプロイしようとするのですが、それが問題の始まりです。Kubernetes のデプロイが氷山の一角であるとすれば、Day 2 の管理の複雑さは、水中に隠されたその氷山の本体であり、船を沈めるほどに巨大な塊です。

これらの課題や問題の解決を開始することは可能ですが、多くの場合、この「Kubernetes をベースとするカスタム PaaS」の構築と保守を行うには、数人のメンバーからなる運用チームが数週間から数カ月間にわたって取り組むことが必要になります。これにより組織の効率は落ちるうえ、この取り組みでは極めて複雑になるセキュリティと認証をサポートしなくてはならず、また、開発者チームのオンボーディング時にはあらゆるものをゼロから開発する必要が生じます。このカスタム Kubernetes プラットフォーム (つまり、Kubernetes) を構築し、維持するためのすべてのタスクと、コンテナを正常に実行するために必要なすべての追加要素を挙げると、次のように分類されます。

- **クラスタ管理**：OS のインストール、OS のパッチ適用、Kubernetes のインストール、CNI ネットワークの構成、認証統合、Ingress と Egress のセットアップ、永続ストレージのセットアップ、ノードの強化、セキュリティパッチの適用、基盤となるクラウド/マルチクラウドの構成など。
- **アプリケーションサービス**：ログ集約、ヘルスチェック、パフォーマンス監視、セキュリティパッチの適用、コンテナレジストリ、アプリケーション・ステージングのプロセスの設定など。
- **開発者統合**：CI/CD 統合、開発者用ツール/IDE 統合、フレームワーク統合、ミドルウェア互換性、アプリケーション・パフォーマンス・ダッシュボードの提供、RBAC など。

作業やテクノロジーは他にも多数ありますが、上述したアクティビティのほとんどは、組織がコンテナを本格的に活用するために不可欠です。また、これらすべての設定にかかる時間や労力とその作業の複雑さは十分に大きなものですが、それでも個々の要素を継続的に保守することに比べれば微々たるものです。統合はそれぞれ十分にテストする必要があります。また、各要素およびアクティビティのリリースサイクル、セキュリティポリシー、およびパッチは異なるものになります。

素の Kubernetes ではなく Red Hat OpenShift で得られるもの

組織が素の Kubernetes をベースとするプロダクションでコンテナを実行するようになると、前のセクションで説明した要素は通常、独自設計のアプリケーション・プラットフォームを作成するためにインストールおよび統合されます。

Azure Red Hat OpenShift は、前のセクションで説明した要素すべてを 1 つのプラットフォームに統合するため、IT チームにとっては運用が容易になり、アプリケーションチームにはタスクの実行に必要なものが提供されます。これらのトピックについては後ほど詳しく説明しますが、このことを念頭に置いた上で、OpenShift と Kubernetes の主な違いをいくつか見ていきましょう。

- **容易なデプロイ**：Kubernetes では、アプリケーションのデプロイに時間がかかる場合があります。GitHub コードをマシンにプルし、コンテナを起動し、Docker Hub などのレジストリでホストし、さらに CI/CD パイプラインを理解する必要があるため、非常に複雑な作業になる可能性があります。一方、OpenShift では手間のかかる作業とバックエンドの作業が自動化されており、ユーザーの作業は、プロジェクトを作成してコードをアップロードするだけです。
- **セキュリティ**：現在、ほとんどの Kubernetes プロジェクトは、複数の開発者と運用者のチームで取り組むものになっています。Kubernetes は RBAC や IAM のような制御をサポートするようになりましたが、それでも手動でのセットアップと構成が必要であり、時間がかかります。Red Hat と OpenShift は、長年の経験を経て、セキュリティのベストプラクティスを特定するという素晴らしい成果に到達しました。お客様側での設定は必要なく、すぐに使うことができます。新規ユーザーを追加するだけで、OpenShift は名前空間やさまざまなセキュリティポリシーの作成などを処理します。
- **柔軟性**：Azure Red Hat OpenShift を使用すると、デプロイ、管理、更新に関するよく知られたベストプラクティスを利用できます。バックエンド内の手間のかかる作業は、何かのボタンを押す必要さえもなく、すべて処理されるため、アプリケーションをすばやく提供することができます。さらに、プラットフォームが Kubernetes であるため、CI/CD DevOps パイプラインを手動でカスタマイズでき、プロセス開発における柔軟性と創造性の余地が広がります。これは、物事を完了する方法を教わるのが好きなチームや、効率化されたアプローチの恩恵を受けるチームにとっては利点です。

- **日常の運用**：クラスタは複数の VM のグループで構成されており、必然的に運用チームはクラスタに追加する必要のある新規の VM を起動する必要があります。Kubernetes を介した構成プロセスは時間がかかって複雑になる場合があり、自己登録やクラウド自動化などを設定するためのスクリプトを開発する必要があります。Azure Red Hat OpenShift を使用すると、クラスタのプロビジョニング、スケーリング、アップグレードの運用はプラットフォームによって自動化され、管理されます。
- **管理**：すべてのディストリビューションに付属する Kubernetes の標準ツールとダッシュボードを利用できますが、ほとんどの開発者はより完全で堅牢なプラットフォームを必要としています。Azure Red Hat OpenShift は、Kubernetes API に基づく優れた Web コンソールと、運用チームがワークロードを管理するための機能を提供します。

第 8 章：アプリケーション・プラットフォームの機能紹介では、Azure Red Hat OpenShift の重要な付加価値機能の多くについて説明しています。

まとめ

Red Hat とマイクロソフトの共同顧客の多くが、コンテナ化されたアプリケーションを導入するためのアプリケーション・プラットフォームとして Azure Red Hat OpenShift を採用しています。

次の章では、クラウドサービスとしての Red Hat OpenShift について説明し、そのアーキテクチャ、統合、管理について詳述します。

第 3 章

Azure Red Hat OpenShift

Azure Red Hat OpenShiftを使用すると、実行するインフラストラクチャの構築と管理について心配することなく、フルマネージドの Red Hat OpenShift クラスタをデプロイできます。

Azure Red Hat OpenShift は、クラウドネイティブでオンデマンドのアプリケーション・プラットフォームであり、その開発、運用、サポートは Red Hat とマイクロソフトが共同で行っています。専門の**サイト信頼性エンジニアリング** (**SRE**) チームが OpenShift クラスタの自動化、スケーリング、セキュリティ保護を行い、連携して統合されたサポートエクスペリエンスを提供します。Azure Red Hat OpenShift を使用すれば、仮想マシンの運用は必要なく、パッチ適用も不要です。コントロールプレーン・ノードとワーカーノードへのパッチ適用、更新、監視は、ユーザーに代わって Red Hat とマイクロソフトが行います。Azure Red Hat OpenShift クラスタは Azure サブスクリプションにデプロイされ、Azure の請求書に含まれます。

Azure Red Hat OpenShift により、アプリケーション提供を迅速化できます。

- 開発者が組み込みの CI/CD パイプラインによって革新することが可能になるため、MySQL、PostgreSQL、Redis、Azure Cosmos DB といった数百もの Azure サービスにアプリケーションを簡単に接続できます。
- プロビジョニング、構成、運用が自動化されているため、複雑さが解消され、生産性の向上に対する障壁がなくなります。
- 条件に応じてスケーリングできます。3 つのワーカーノードを備えた高可用性クラスタを数分で開始し、アプリケーションの需要の変化に応じて拡張できます。さらに、標準、ハイメモリー、または高 CPU のワーカーノードを選択できます。
- エンタープライズグレードの運用、セキュリティ、統合されたサポートエクスペリエンスの享受。

次に、Red Hat OpenShift の構築と運用を支える技術的詳細について掘り下げます。

アーキテクチャ

Azure Red Hat OpenShift は、Azure インフラストラクチャ・サービス (仮想マシン、ネットワーク・セキュリティ・グループ、ストレージアカウントなどの Azure サービス) を Red Hat OpenShift インストールの基盤として使用します。Azure アーキテクチャは Azure サブスクリプションにも完全にデプロイされているため、アカウント内で既に稼働している他のサービスと簡単に統合できます。

OpenShift 自体は、Red Hat Enterprise Linux CoreOS の上に構築されています。これは、連携して動作する小さな分離ユニットから成るマイクロサービスベースのアーキテクチャをホストします。各仮想マシンで、Kubernetes クラスタの基盤である `kubelet` サービスが実行されます。コントロールプレーン上で稼働しているこのアーキテクチャの背後にあるデータベースは、信頼性が高く、クラスタ化された key-value ストアである **etcd** です。

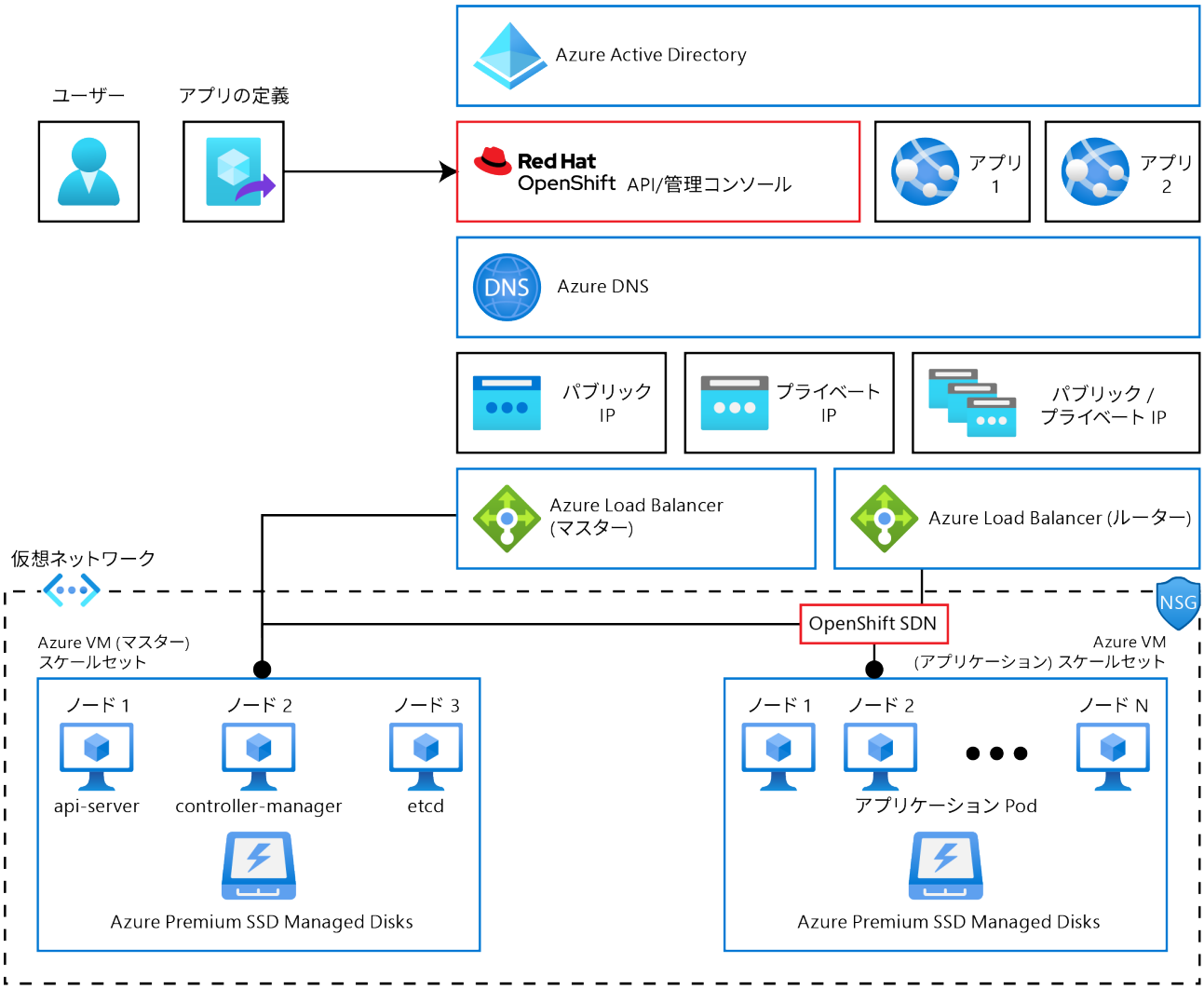


図 3.1：Azure Red Hat OpenShift アーキテクチャ

次の 2 つのセクション、コンピューティング – コントロールノードとワーカーノード、および ネットワークでは、この図の内容について詳しく説明します。

コンピューティング – コントロールノードとワーカーノード

Red Hat OpenShift のアーキテクチャは、3 つの特定の役割 (コントロール、インフラストラクチャ、アプリケーション) のいずれかを持つ仮想マシン (ノード) で実行されます。ただし、執筆時点で、Azure Red Hat OpenShift バージョン 4 はインフラストラクチャ・ノードをサポートしていないため、この e ブックではコントロールノードとワーカーノードについてのみ説明します。

コントロールノード (従来の**マスター**ノード) は、Kubernetes クラスタに不可欠なコンポーネントを含む仮想マシンです。これには、API サーバー、コントローラー・マネージャー・サーバー、スケジューラー、etcd が含まれます。コントロールノードは Kubernetes クラスタを管理し、Pod がワーカーノードで実行されるようスケジュールします。

- **Kubernetes API サーバー**は Pod、サービスおよびレプリケーション・コントローラーのデータを検証し、設定します。また、クラスタの共有状態を確認できる中心的な部分として機能します。
- **Kubernetes コントローラー・マネージャー**は etcd でレプリケーション、namespace、サービス・アカウント・コントローラーのオブジェクトなどのオブジェクトへの変更の有無を監視し、API を使用して指定された状態を実行します。このような複数のプロセスは、一度に 1 つのアクティブなリーダーを設定してクラスタを作成します。
- **Kubernetes スケジューラー**は、割り当て済みのノードなしで新規に作成された Pod の有無を監視し、Pod をホストする最適なノードを選択します。
- **etcd** は、状態のマスターデータを永続的に保管します。他のコンポーネントは etcd を参照し、変更を検出したらその新しい状態へと遷移します。

アプリケーションノードは、アプリケーションが実際に実行される場所です。

Kubernetes クラスタの各ノードは、`kubelet` というサービスを実行します。このサービスは、**コンテナ・ランタイム・インタフェース** (**cri-o**)、サービスプロキシ、および各ノードで実行されるその他の重要なサービスを維持します。すべてのノードは、OpenShift のソフトウェア・デファインド・ネットワーク・テクノロジーに接続されています。Azure Red Hat OpenShift では、これは **Open Virtual Network (OVN)** であり、Azure 仮想ネットワーク上で実行されます。

Azure Red Hat OpenShift は、ストレージ用の Azure ディスクに接続している Azure 仮想マシンで実行されるノードを作成します。ディスクは 1 TB という極めて大きなサイズです。これは、Azure では、ストレージパフォーマンスの IOPS 保証がディスクのサイズに関連しているためです。サイズが 1 TB の場合、etcd データベースによって使用される基盤となるディスクに十分な帯域幅が保証されます。

ネットワーク

Azure Red Hat OpenShift には、コントロールプレーン・ノード用とワーカーノード用の 2 つのサブネットが設定された、単一の Azure 仮想ネットワークが必要です。両方のネットワークの最小サイズは /27 (32 アドレス) です。ただし、後でクラスタをスケーリングする場合は、サイズを小さくしすぎないように注意してください。

2 つの Azure Load Balancer (図では**マスター**、**ルーター**と表記) は、トラフィックを次のように転送します。

- マスター/コントロールプレーンのロードバランサー：OpenShift API のユーザーに Ingress トラフィックを送信します。また、コントロールプレーン・ノードにアウトバウンド接続を提供します
- ルーター/アプリケーションのロードバランサー：OpenShift「ルーター」または Ingress コントローラーを介して、OpenShift で実行されているアプリケーションに Ingress トラフィックを送信します。また、ワーカーノードにアウトバウンド接続を提供します

ネットワークのドキュメントに、ネットワークの構成、要件、および制限に関する優れた記事が掲載されています。

他の Azure サービスとの統合

Azure のネイティブサービスである Azure Red Hat OpenShift は、Azure での使用に適した多くのサービスと一緒にデプロイし、それらと統合することができます。一般的な統合が存在する Azure インフラストラクチャ・サービスのうち、ほんの一部を概要として下図にまとめました。

図 3.2 は、OpenShift on Azure での一般的な Azure サービスの統合ポイントの多くを示しています。

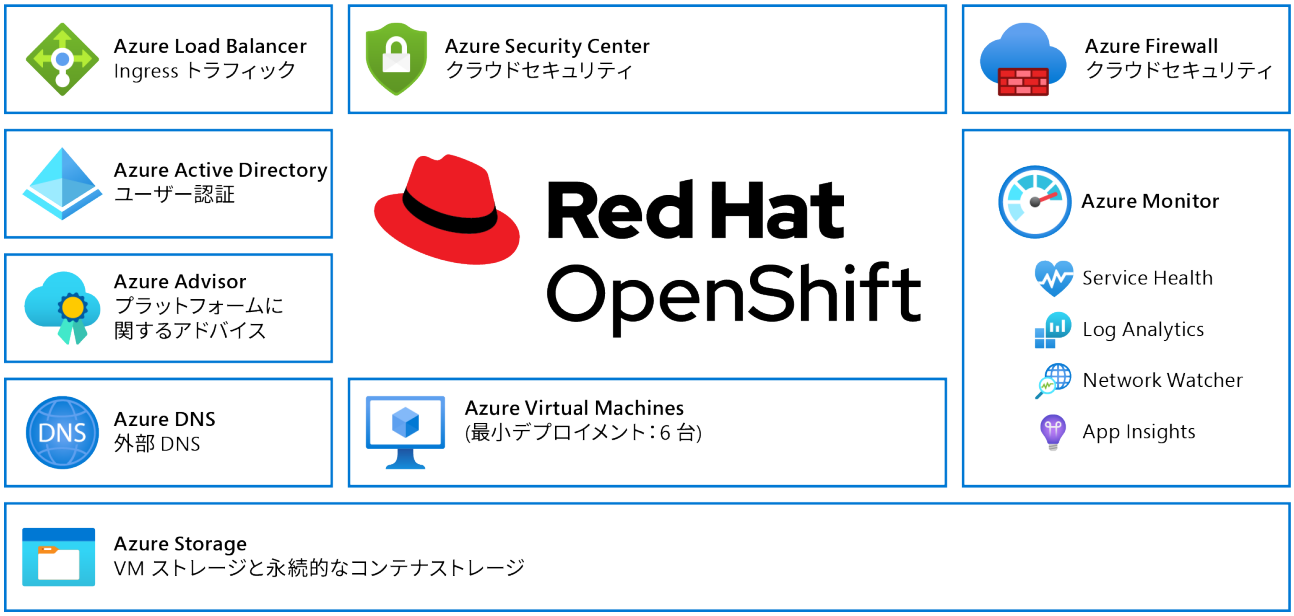


図 3.2：一般的な Azure サービスの統合ポイント

さらに、OpenShift で実行されるアプリケーションは、Azure Service Operator を使用することで Azure サービスと非常に緊密に統合できます。これについては、後ほど第 9 章：他のサービスとの統合で詳しく説明します。

管理

Azure Red Hat OpenShift サービスの一部として管理されるものについて簡単に説明するとすれば、データセンターからクラスタ Operator まですべてが「管理されている」と言えます。クラスタ Operator は、コントロールプレーン・ノードで実行され、クラスタの監視、更新、および正常性を管理するサービスです。つまり、マイクロソフトと Red Hat は、クラスタをオンラインにしておくために、これらのコンポーネントを共同で監視、保守、および管理します。クラスタ Operator より上の部分はすべてお客様が管理することに変わりありません。

Azure Red Hat OpenShift の利用者であるお客様には、クラスタへの完全な **cluster-admin** アクセスが許可されます。つまり、クラスタの構成要素を破損させないという責任も共有します。お客様がクラスタでできることとできないことを理解するには、サポートポリシーを理解することが重要です。

クラウドサービスの一環としてマイクロソフトと Red Hat が提供するサービスの詳細は、Azure Red Hat OpenShift 責任マトリックスに記載されています。

認証と認可

通常、Azure Red Hat OpenShift クラスタに認証を与えるために使用されるのは、Azure Active Directory です。ただし、これは必須ではなく、代わりに GitHub を使用したログインや単純な「パスワードファイル」などの認証メカニズムを使用することができます。

Azure Active Directory が使用されている場合、Azure Red Hat OpenShift と Kubernetes API は認証要求を転送します。ユーザーは自分の資格情報を提示し、自分のロールに基づいて認可されます。

認証レイヤーは、Azure Red Hat OpenShift API への要求に関連付けられているユーザーを識別します。次に、認可レイヤーは、要求しているユーザーに関する情報を使用して、要求を許可すべきかどうかを判別します。

Azure Active Directory の構成手順については、認証 - Azure Active Directory のセクションで説明しています。Azure Active Directory の代わりに別の認証プロバイダーが使用されている場合も、このプロセスは機能的に非常に似ています。

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

クラスタロール、ローカルロール、クラスタロール・バインディング、ローカルロール・バインディング、ユーザー、グループ、およびサービスアカウント間の関係を、図 3.3 に示します。

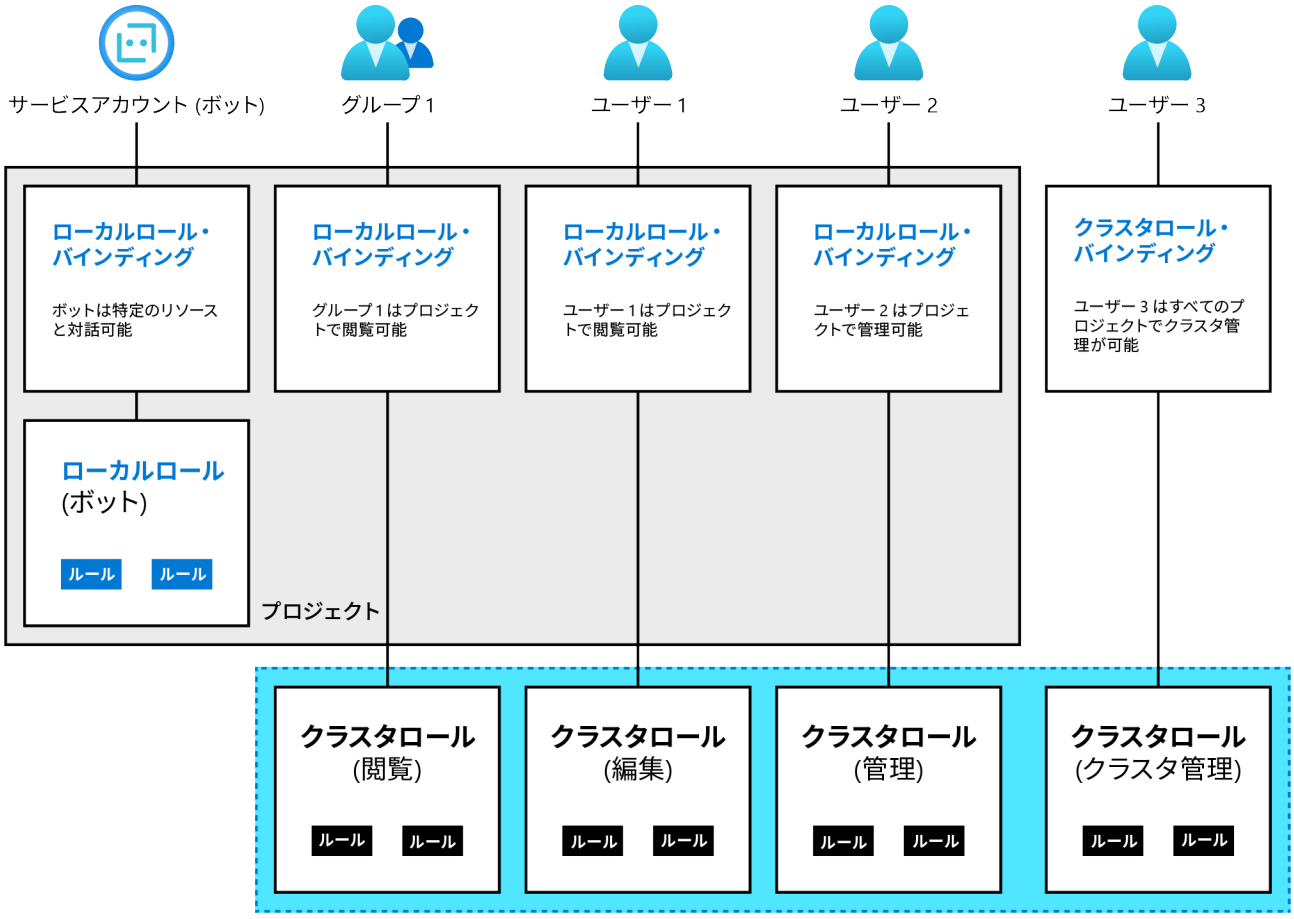


図 3.3：ロール間の関係

Red Hat OpenShift のドキュメントに認証プロバイダーの全リストが記載されています。

サポート

Azure Red Hat OpenShift は、独自のやり方でサポートを管理しています。マイクロソフトと Red Hat のサポートチームは、全世界のサイト信頼性エンジニアリング (SRE) チームと連携して、サービスの運用を支援します。

お客様は Azure ポータルでサポートを要求します。その要求が Azure プラットフォームに関するものか OpenShift に関するものかに関係なく、マイクロソフトと Red Hat のエンジニアが選別して対処し、迅速に対応します。

統合されたサポートプロセスの概要を以下に示します。

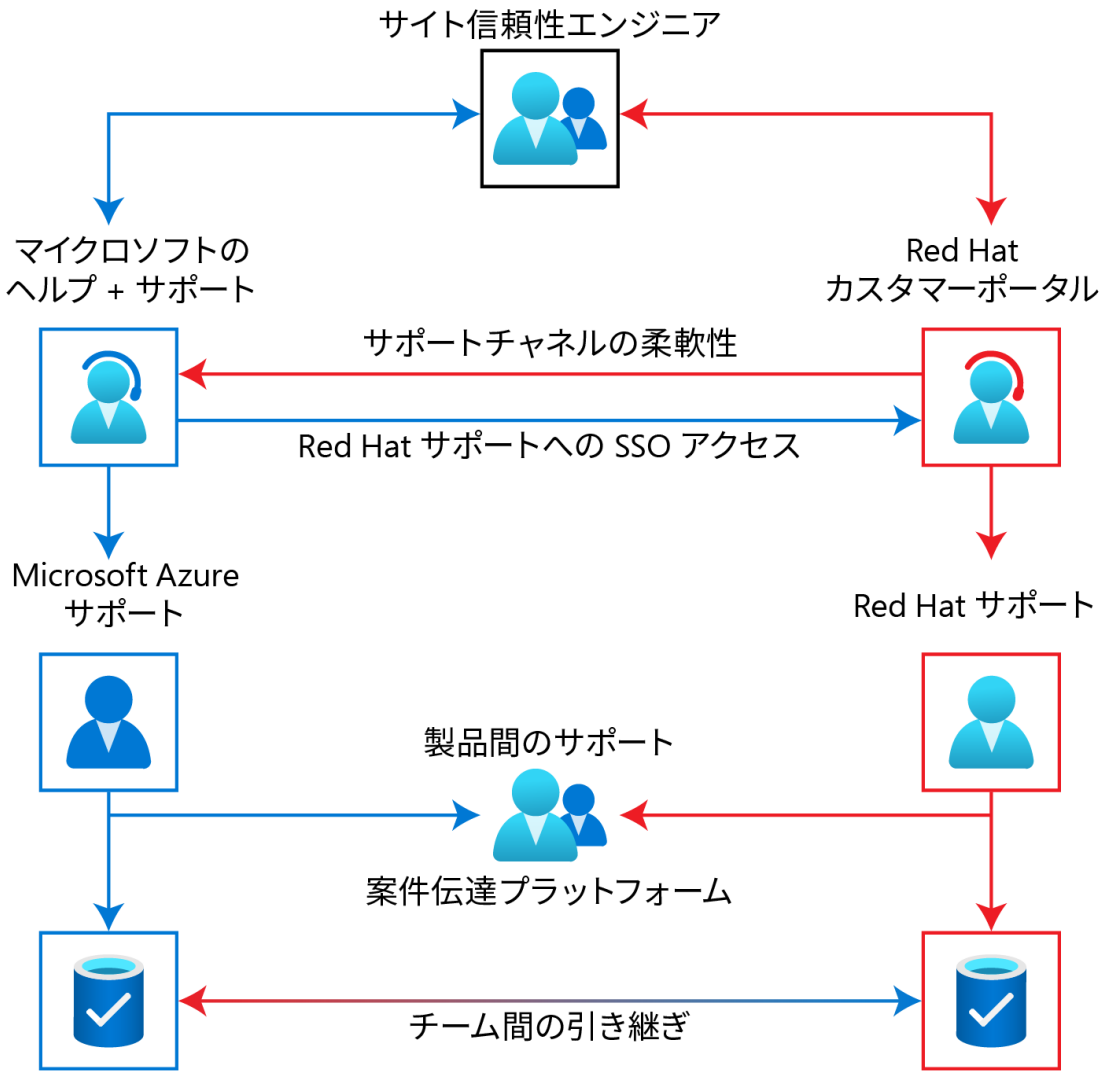


図 3.4 : 統合されたサポートプロセス

図 3.4 は、お客様がマイクロソフトサポートポータルと Red Hat サポートポータルのいずれでもサポートチケットを発行できることを示しています。Red Hat サポートポータルにアクセスするには、クラスタを OpenShift Cluster Manager に登録する必要があります。

マイクロソフトのサポートエンジニアは、お客様の同意を得て、案件伝達プラットフォームを介していつでも Red Hat とシームレスに連携することができます。Red Hat のサポートエンジニアがマイクロソフトとの連携をリクエストする場合も同様です。両社のサポートエンジニアは SRE チームにアクセス可能で、必要に応じてクラスタを修復するための是正措置を講じることができます。

料金とサブスクリプション

Do-it-yourself (DIY) インストールに対する Azure Red Hat OpenShift の主なメリットの 1 つは、コンピューティング、ネットワーク、ストレージ、Azure インフラストラクチャ、および OpenShift サブスクリプションへの請求が個別ではなく、Azure サブスクリプションを介した一括請求であることです。ソリューションのコストを確認するには、Azure Red Hat OpenShift を [Azure 料金計算ツール](#)の見積もり内容に追加するだけです。

料金計算ツールの使い方を以下に示します。

1. 検索ボックスに openshift と入力して、新しい見積もりに追加します。

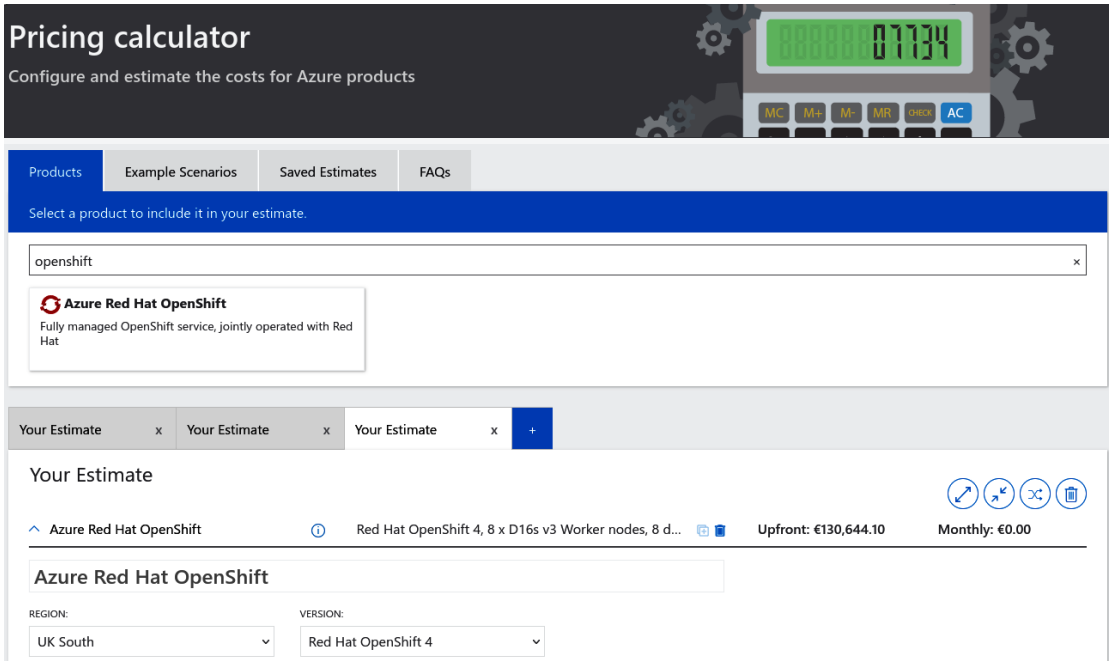


図 3.5：料金計算ツールのパネル

2. ページの最下部に、料金の単位を現地通貨に変更するためのドロップダウンリストがあります。この例では GBP (£) が使用されています。
3. Azure Red Hat OpenShift をデプロイする Azure リージョンを設定します。Azure リージョン間でのコンピューティングコストの違いが考慮されており、料金はリージョン間でわずかに変動します。

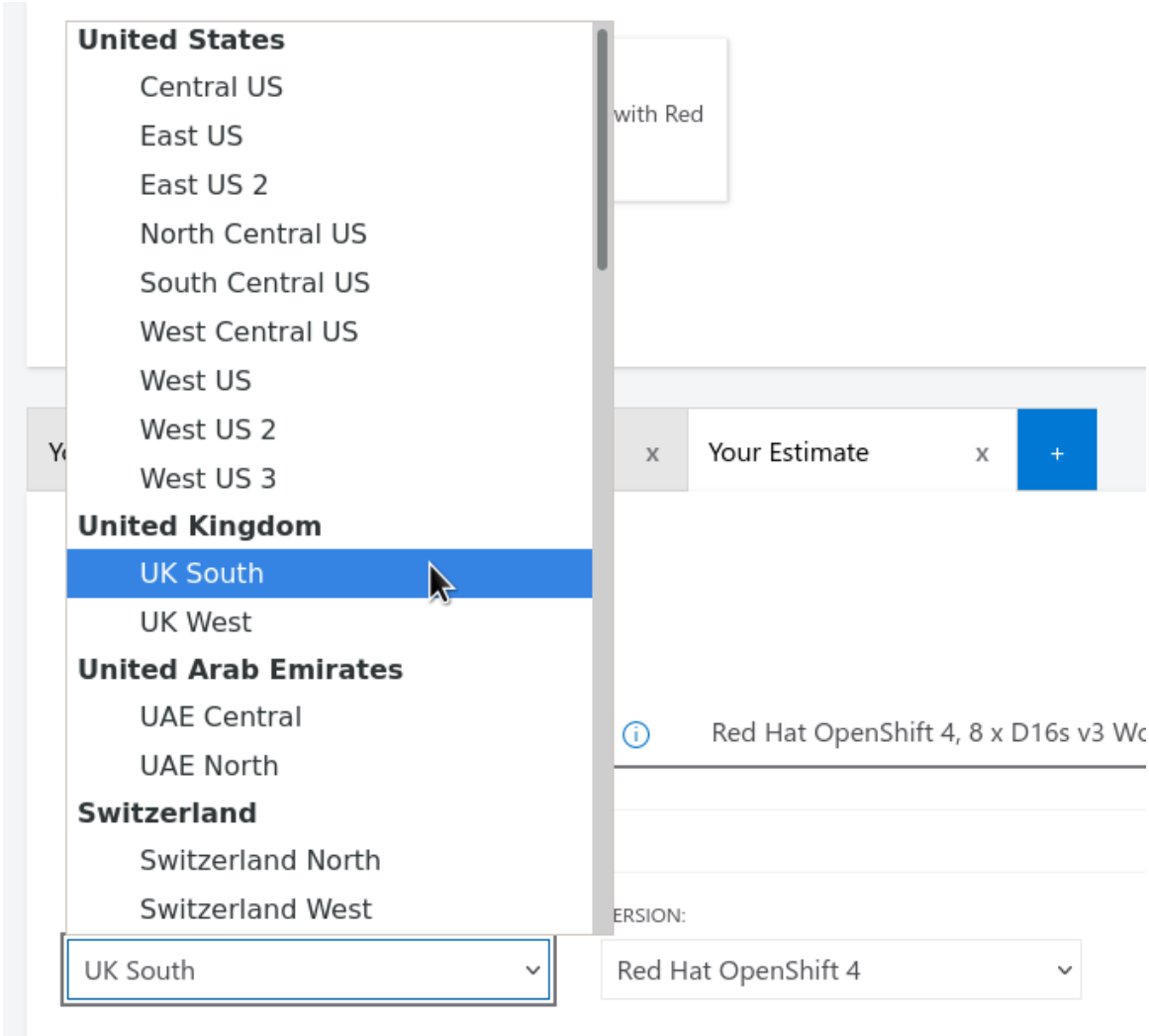


図 3.6：Azure リージョンの選択

4. **ワーカーノード**は、実際のアプリケーションが実行される場所です。**割引のオプション**には 2 つのセクションがあります。**ライセンス**には OpenShift サブスクリプションのコスト、**Virtual Machine** セクションにはクラスタの実行に必要なコンピューティングサービスのコストが表示されます。クラスタでサポートされるワーカーノードの最大数は 100 です。

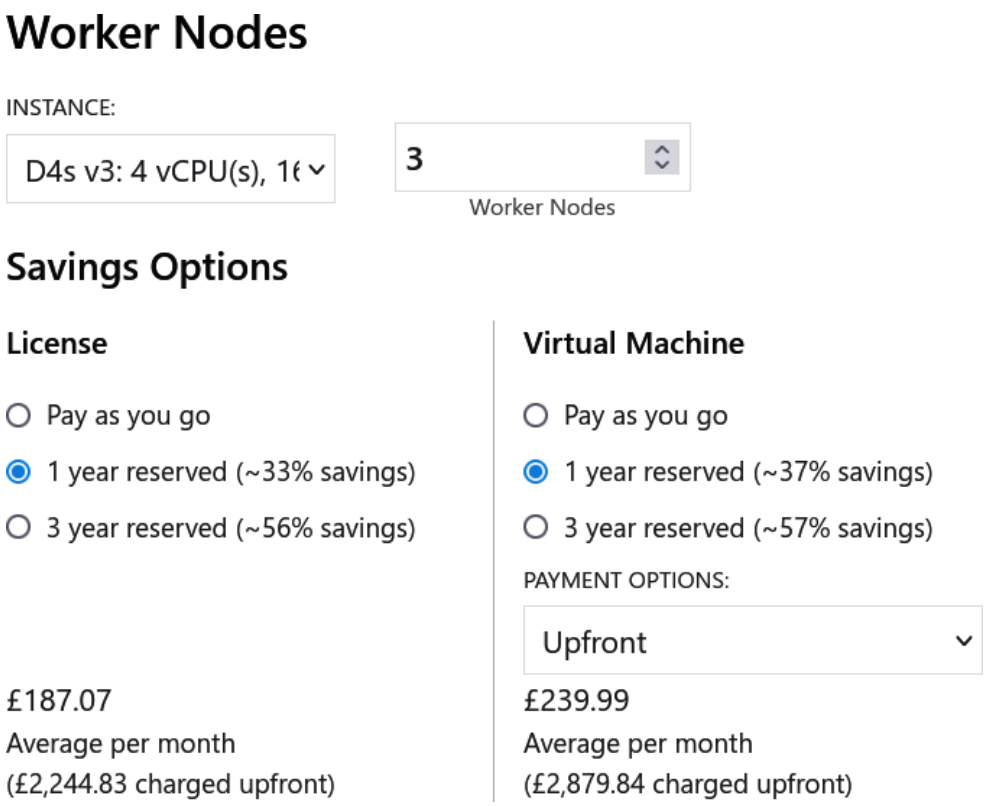


図 3.7：ワーカーノードの詳細

5. **管理 OS ディスク**では、Red Hat CoreOS がオペレーティングシステムのパーティションに使用するディスクのサイズを指定します。これは、別途プロビジョニングされる、アプリケーション永続ボリュームが使用するストレージとは異なります。

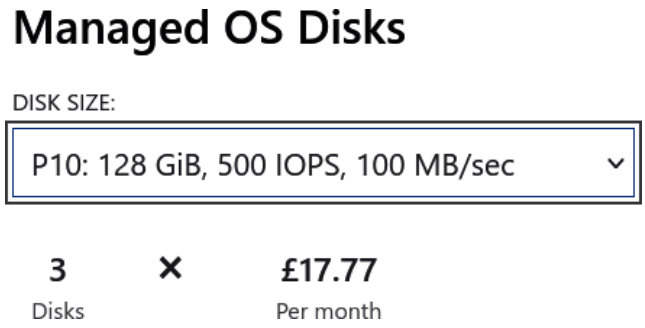


図 3.8：管理 OS ディスク

6. **マスターノード** (最近は通常「コントロールノード」と呼ばれる) にも同様のセクションがあります。コントロールノードには、ワーカーノードよりもはるかに大きなディスクが関連付けられています。これは、Azure のディスクサイズが大きくなると、より高い IOPS とスループットが必要になるからです。クラスタの安定性のために、コントロールプレーン・ノードの数は常に 3 つである必要があります。

料金計算ツールは目安となる料金を示すことを意図したものであり、また、実際の料金は使用状況に応じて変動することに注意してください。

Azure Reserved VM Instances

予約インスタンスとは、そのリソースを 1 年間または 3 年間使用するというコミットメントのことです。Azure Red Hat OpenShift 仮想マシンには、予約インスタンスのオプションがあります。

通常、クラスタを長期にわたって稼働させることがわかっている場合は、IaaS 料金の大幅な割引を受けるために予約インスタンスを選択します。たとえば、本番環境は多くの場合、予約インスタンスを使用して実行されます。予約インスタンスを使用しても、クラスタのサービスレベルやアーキテクチャは変更されません。

Azure Reserved VM Instances の詳細

まとめ

本章では、Azure Red Hat OpenShift マネージド・クラウドサービスの詳細について説明しました。アーキテクチャの概要を示し、他の Azure サービスとの統合 (第 9 章：他のサービスとの統合で詳述) と、管理、認証、サポート、および料金に関する考慮事項について簡単に説明しました。

次の章では、Azure Red Hat OpenShift のデプロイにおけるプロビジョニングの前段階で、組織が答えを出しておく必要がある事柄と意思決定に焦点を当てます。

第 4 章

プロビジョニングの準備 – エンタープライズ・アーキテクチャに関する考慮事項

多くの組織は、Azure ポータルで Azure Red Hat OpenShift を見つけます。そしてドキュメントを一度読めば、追加の作業もほとんど必要なく、Red Hat OpenShift でクラスタを正常にデプロイできるでしょう。しかし、デプロイの計画にもう少し時間をかけて事前にいくつかのポイントを明確にしておけば、将来、クラスタの削除や再プロビジョニングにかかる時間を大幅に節約できます。

本章は、多くのお客様との連携を通じて実際に体験したエクスペリエンスに基づいています。ここでは、プロビジョニングの準備としてまず考慮しておく必要がある一般的な事柄を取り上げます。本章では、以下について説明します。

- ステージング、プロダクションなど、必要なクラスタの数
- パブリックおよびプライベートネットワークの可視性
- オンプレミス・ソリューションへの接続などのハイブリッド接続

まず、必要なクラスタ数の算出方法から始めます。

必要なクラスタの数

OpenShift のデプロイメントパターンは多数ありますが、よくある疑問は「自分の組織ではいくつのクラスタが必要なのか?」ということです。もちろん、それは組織によって異なりますが、その数を算出するのに役立つガイダンスを以下にまとめました。

ライフサイクルステージ：開発、テスト、プロダクション

どのような規模の組織であっても、ほとんどが何らかのライフサイクルステージでエンタープライズ IT システムをデプロイします。このアプローチは、ステージングパターンと呼ばれることもあります。最もよくある 3 つのステージが、開発、テスト、プロダクションです。複数のステージがあれば、プロダクション環境に到達する前に、アプリケーションへの変更やアプリケーションのデプロイを安全な環境でテストすることができます。最も一般的な、推奨されるステージングパターンは、少なくとも 3 つの Azure Red Hat OpenShift クラスタを使用することです。

- **開発**：このステージでは、開発者と運用者があらゆることをテストできます。1 つの大きな「サンドボックス」クラスタを使用する場合もありますが、通常は小規模なクラスタを短時間だけ作成してテストの作成と破棄を頻繁に行い、こまめに廃棄します。多くの場合、その方が安全です。
- **テスト**：このステージでは、プロダクションへのリリース前に、パッチや構成の変更など、今後クラスタに適用する変更をテストし、検証します。一部の組織では「プリプロダクション」とも呼ばれますが、プリプロダクションは完全に別の環境である場合もあります。
- **プロダクション**：実際のアプリケーションが実行されるステージです。

これらに加えて、統合テスト環境などの付加的な環境を用意する組織もあります。しかし、自組織に最適なステージング環境の数を判断できるのはお客様ご自身だけです。よくわからない場合は、他の同様のエンタープライズ・アプリケーションの一般的なデプロイメントパターンを見てみると参考になります。

Azure Red Hat OpenShift をビジネスクリティカルではないアプリケーションに使用している場合は、組織内のクラスタを 2 つだけにする (開発とテストを 1 つにまとめる) こともできますし、開発、テスト、プロダクションを組み入れた単一のクラスタを使用することも可能です。これには、クラウドのコストを抑え、管理が必要なクラスタの総数を減らせるという利点があります。単一のクラスタだけで運用する場合、管理者は、開発、テスト、およびプロダクション用に別々の namespace を使用することも可能です。しかし、単一のクラスタのみを実行することにはデメリットがあります。

- クラスタ全体に影響を与える変更 (ソフトウェアパッチなど) を行う際に、テスト環境を別クラスタで運用していればそちらで問題を検出して防止できますが、単一クラスタ運用の場合はその問題がプロダクションで表出する可能性があります。
- テスト環境や開発環境のアプリケーションが、大量のコンテナを生成したりディスク容量を使い果たしたりするなどの予期しない動作をした場合、プロダクション環境に問題が発生します。

本書では、これを読んでいるお客様の状況に応じて完璧に機能するクラスタの正確な数を示すことはできません。先ほどお勧めしたように、組織で使用している同様のエンタープライズ・アプリケーションのライフサイクルステージの数を参考にするとよいでしょう。

ビジネス継続性、障害復旧、フェイルオーバー

多くの組織では、標準的なステージング環境に加えて、フェイルオーバー・プロダクション環境を少なくとも 1 つ作成することも検討するでしょう。フェイルオーバー・プロダクション環境は、クラスタ全体または Azure リージョンを停止させるような壊滅的な障害が発生した場合に使用されます。これは、しばしば**障害復旧 (DR)** と呼ばれます。通常、これはプロダクションクラスタとは異なる Azure リージョンにデプロイします。

このマネージド・クラウドサービスには、99.95% の**サービスレベル契約 (SLA)** が付帯しています。これは、多くのビジネスクリティカルなアプリケーションにとって十分な数値です。しかし、アプリケーションによっては、より高い SLA が求められる場合があります。ここで重要なのは、単一のクラスタでこの SLA を超えるのは不可能であると理解することです。組織のアプリケーションに 99.95% を超えるサービスレベルが必要か、あるいは 99.95% で十分なのかを確認してください。

十分でなければ、複数のクラスタを並列実行して複合 SLA を算出すれば、より高いレベルのサービス可用性 (99.999% など) が得られます。クラスタはすべて同じリージョン内 (例：`westeurope`) に作成することもできますし、複数のリージョン (例：`westeurope` と `northeurope`) に作成すればさらに高いレベルの可用性を実現できます。下記の Azure ドキュメントでは、複数クラスタの実行時に複合 SLA を算出する方法を説明しています。

[複合 SLA に関する Azure ドキュメント]

Azure Red Hat OpenShift のマルチクラスタおよびマルチリージョンのデプロイについては、本書では取り上げません。それは、このような複雑なアーキテクチャを構築する場合、クラスタへのトラフィックの取り込みに関連する課題がいくつかあり、どのデータをどのようにしてクラスタ間で共有するかを決定するにはかなりの検討が必要となるからです。

リージョンとアベイラビリティゾーン

Azure Red Hat OpenShift は、デプロイ先の各リージョンで 3 つのアベイラビリティゾーンを利用するよう設計されています。Azure では、[アベイラビリティゾーン]はリージョン内の自律型データセンターであり、独自の電力、冷却機能、ネットワーク接続を備えています。Azure Red Hat OpenShift のデプロイメントを確認すると、各アベイラビリティゾーンに単一のコントロールノード (仮想マシン) とアプリケーションノードがあることがわかります。

図 4.1 は、コントロールノードとアプリケーション (ワーカー) ノードが単一の Azure リージョン内の 3 つのアベイラビリティゾーンにどのように分散されているかを示しています。

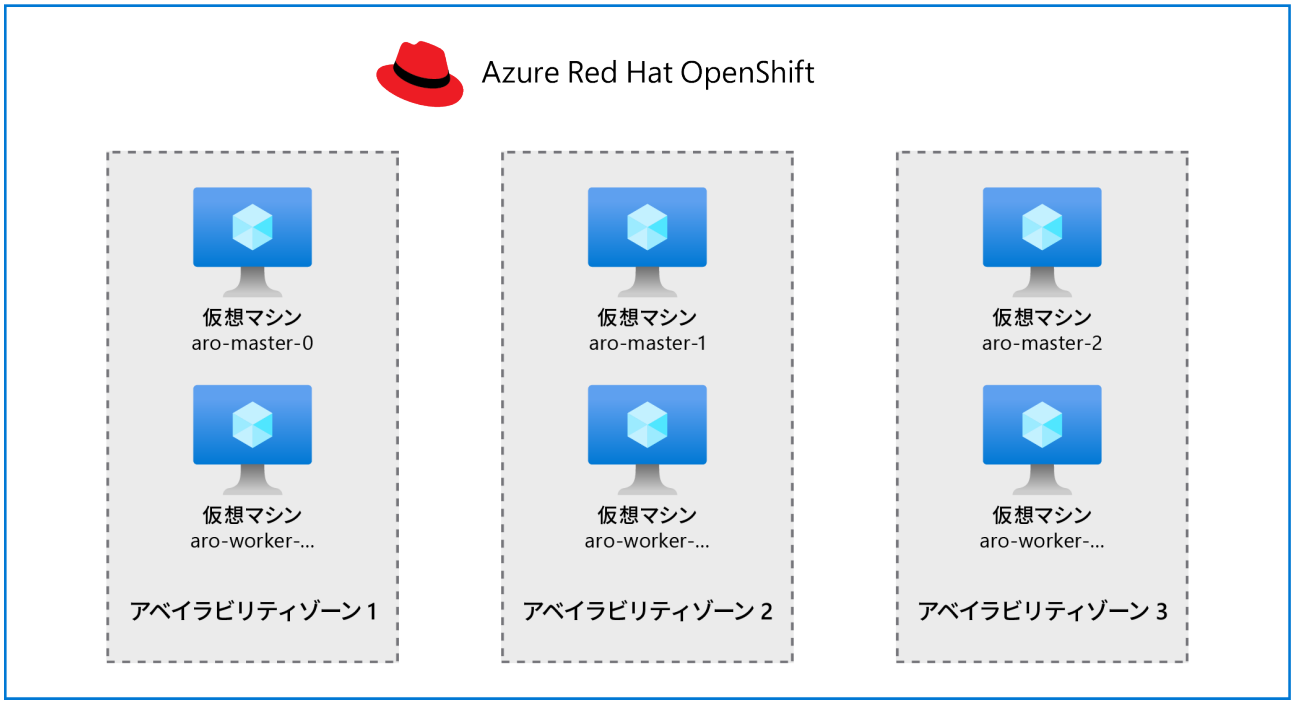


図 4.1：単一の Azure リージョン内の 3 つのアベイラビリティゾーンに分散するコントロールノードとアプリケーションノード

Azure Red Hat OpenShift は、可用性の高いフルマネージドサービスとして提供されるように作られています。したがって、デプロイするコントロールノードとワーカーノードを 3 つ未満にすることはできません。

ネットワークの概念

「Azure Red Hat OpenShift 入門」で、ネットワークの概念についてわかりやすく説明しているマイクロソフトのドキュメントに触れましたが、そのページにはこちらからアクセスできます。

- Azure Red Hat OpenShift のネットワークの概念

このページでは、Azure Red Hat OpenShift のあらゆるネットワーク・コンポーネント (ロードバランサー、パブリック IP アドレスとプライベート IP アドレス、ネットワーク・セキュリティ・グループなど) について詳しく説明しています。

理解しておくべき重要なポイントは次のとおりです。

- Azure Red Hat OpenShift は、1 つの仮想ネットワーク (既存または新規) にデプロイされます。サポートされる仮想ネットワークも 1 つだけですが、OpenShift の上には OVS という独自の**ソフトウェア・デファインド・ネットワーク (SDN)** のレイヤーがあるため、ネットワークの数を増やしてもメリットはありません。
- マスターノードとアプリケーションノードのサブネットの最小サイズは /27 です。
- デフォルトの Pod CIDR は `10.128.0.0/14` です。
- デフォルトのサービス CIDR は `172.30.0.0/16` です。
- 各ノードでは Pod 用に /23 サブネット (512 個の IP アドレス) が割り当てられます。この値は変更できません。
- Egress IP は現在サポートされていません。
- Egress トラフィックルーティングを制御することができます。具体的には、Azure Firewall を介して送信します。執筆時点では、この機能は公開プレビューであり、こちら (エグレストラフィックを制御する) に文書化されています。

パブリックまたはプライベートネットワークの可視性

Azure Red Hat OpenShift はパブリック環境にもプライベート環境にもデプロイできるという話を、よく耳にするかもしれません。これは事実ですが、コントロールプレーンをパブリック/プライベートにすることと、クラスタ上のアプリケーションをパブリック/プライベートにすることの違いを理解することは有用です。

API サーバーの可視性については、プロビジョニング時に決定する必要があります。クラスタのプロビジョニング後にパブリック/プライベートの可視性を調整することはできません。

本章で後ほど Azure Red Hat OpenShift クラスタを作成する際には `az aro create` コマンドを実行しますが、このコマンドにはクラスタの可視性に関する引数があります。次の例は、可視性を調整する方法を示しています。

どちらもプライベート

```
az aro create .... --apiserver-visibility Private --ingress-visibility Private
```

API サーバーがプライベート、ワーカーの Ingress がパブリック

```
az aro create .... --apiserver-visibility Private --ingress-visibility Public
```

apiserver とアプリケーションの Ingress の可視性は、図 4.2 の Azure アーキテクチャの図に示されています。この図は、Azure Red Hat OpenShift が内部およびパブリックの Azure ロードバランサーをどのように使用するかを示しています。ここでは、選択した可視性に応じて API とルーターがデプロイされます。

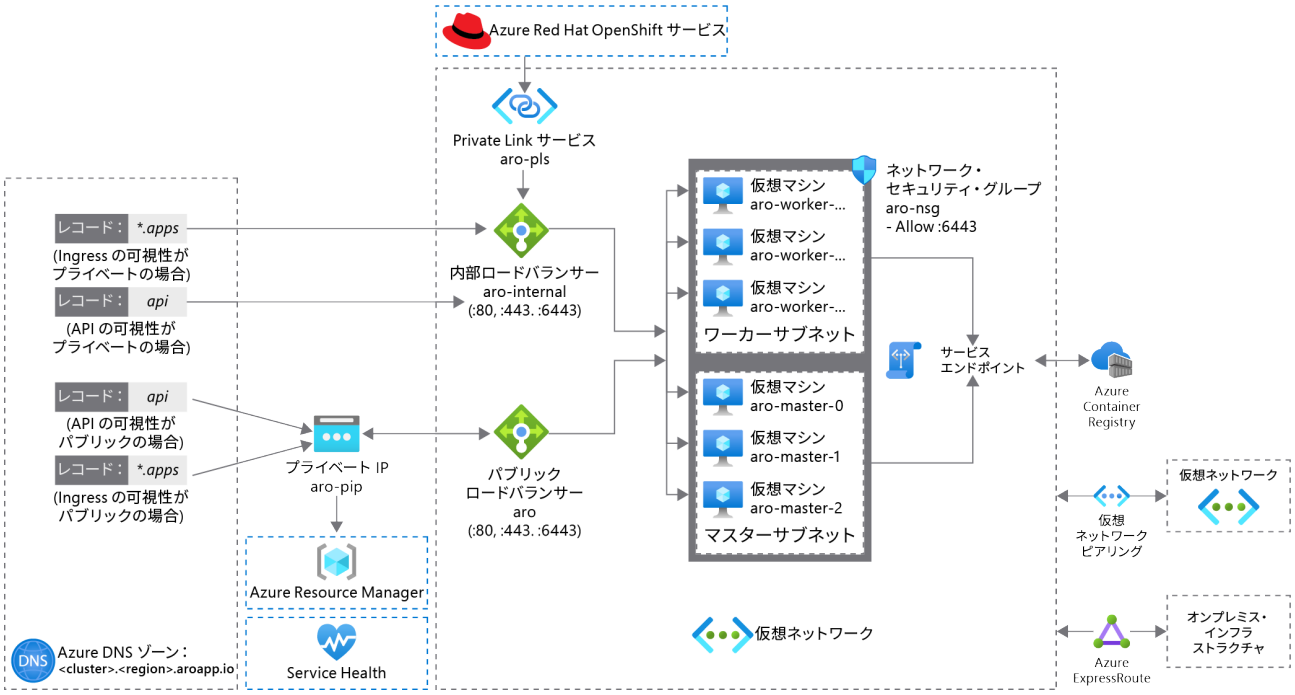


図 4.2：内部およびパブリックの Azure ロードバランサーを使用する Azure Red Hat OpenShift

API サーバーの可視性と Ingress の可視性の詳細は、以下を参照してください。

API サーバーの可視性 (コントロールプレーン)

`--apiserver-visibility` には **Public** または **Private** を指定できます。

- **Private** の場合、Kubernetes API が実行される Red Hat OpenShift コントロールプレーン (従来の「マスターノード」) に、パブリックインターネットからアクセスすることはできません。このコントロールプレーンは、開発者と運用者にクラスタを制御させることを目的としており、クラスタ自体だけでなくアプリケーションのデプロイ、削除、またはスケーリングにも使用できます。Azure への高速ルート接続を使用している企業、または**仮想プライベートネットワーク (VPN)** を使用してコンピューティング・リソースにアクセスしている企業は、ほとんどの場合、プライベートを選択する必要があります。
- **Public** の場合は、クラスタのコントロールプレーンにパブリックインターネットからアクセスできます。これにより、クラスタへのアクセスは必ず認証を通るとはいえ、クラスタがインターネット上で何者かの攻撃を受けるリスクにさらされることになります。しかし、Public に設定するのは、ユーザーのソースネットワークを制御できないラボ環境やテスト環境では有用です。実際のデータが保存されている環境やあらゆる種類のプロダクション環境では、API サーバーの可視性を Private に設定することが強く推奨されます。

API サーバー接続が必要な例の一部を以下に示します。これらは、パブリックかプライベートかを決定する際に考慮に入れる必要があります。

- 開発者が IDE のスクリプトとツール (kubectl rollout など) を使用する
- 運用者がクラスタの状態を確認する (kubectl get nodes など)
- CI/CD サーバーのデプロイメントの状態 (Azure DevOps など) を確認または調整する必要がある
- クラスタセキュリティツールでクラスタに接続して状態を調べる
- 監視ツールが Kubernetes API を利用する

ほとんどの場合、ネットワークは**プライベート**接続を経由するように構成できますが、上記のリストにはない、各組織に固有の事例がある場合があり、**パブリック** API サーバーの可視性に関する予想外の要件が生じる可能性もあります。

Ingress の可視性 (アプリケーション)

`--ingress-visibility` には Public または Private を指定できます。

- **Private** の場合、公開されているサービス (クラスタ上で実行されているアプリケーションに関連) にパブリックインターネットから直接接続することはできません。しかし、ユーザーがアプリケーションに接続する前に、別の Azure ネットワークで任意で実行されている Azure Firewall または Web アプリケーション・ファイアウォールを最初に通過するようにネットワークルーティングを構成することは可能です。**Private** は、クラスタ上のアプリケーションが組織内でのみ使用されることを意図している場合にも適しています。たとえば、給与処理、データ分析、その他の内部 Web アプリケーションなどです。
- **Public** の場合は、クラスタ上のアプリケーションにパブリックインターネットからアクセスできます。ただし、これらのアプリケーションにアクセスできるように、Ingress またはルートリソースを構成する必要があります。これは、Red Hat OpenShift でパブリックの e コマースショッピング Web サイトやその他のパブリック・アプリケーションをホストする場合などに当てはまります。

Ingress は、OpenShift「ルーター」と呼ばれることもあります。

プロダクションに関する推奨事項

以下のことを強くお勧めします。

- クラスタへのアクセスにはプライベート接続を使用し、パブリックインターネットは使わないようにしましょう。Azure ExpressRoute は、クラスタがオンプレミスのネットワークや企業のオフィスからの永続的な接続を必要とする場合に最適なオプションです。それ以外の場合は、Azure への VPN でプライベート接続を提供することもできます。詳細は、**ハイブリッド接続**のセクションをご覧ください。
- API サーバーの可視性 (コントロールプレーン) を Private に設定しましょう。また、ファイアウォールによるアクセス制限を強化するのも有効です。
- 組織内で実行されているアプリケーションの Ingress 可視性 (アプリケーション) を Private に設定しましょう。Azure Red Hat OpenShift のアプリケーションのユースケースをパブリックインターネットでホストする必要がある場合は、別の Azure Red Hat OpenShift クラスタをセットアップします。少なくとも 1 つが内部アプリケーション用、もう 1 つが外部アプリケーション用で、Ingress の可視性はパブリックに設定します。ただし、同じクラスタ内で Public と Private の両方の Ingress を混在させることは可能です。

通常、ほとんどの組織では Azure ネットワークチームやセキュリティチームに相談して、追加の制御が必要かどうかを判断します。たとえば、Web アプリケーションの前面に Web アプリケーション・ファイアウォールを配置することが必要な場合があります。これは、Azure Red Hat OpenShift にアプリケーションをデプロイする際の一般的なデプロイメントパターンでもあります。

ハイブリッド接続

Azure Red Hat OpenShift をデプロイするほとんどの組織が、オンプレミス環境で稼働しているサポートサービスに接続し直す必要があるアプリケーションを実行します。Azure からオンプレミスに接続し直す場合、これは一般にハイブリッド接続またはハイブリッドクラウド・アーキテクチャと呼ばれます。オンプレミス環境に接続を戻すには、いくつかの方法があります。最も注目すべき方法は 2 つです。

- **VPN** は、Azure への複雑度の低い接続に適しています。通常、VPN は Azure VPN Gateway を介して接続されます。詳細は Azure VPN Gateway とはを参照してください。
- **Azure ExpressRoute 回線**は、Azure への堅牢で永続的な専用接続に適しています。詳細は Azure ExpressRoute とはを参照してください。

いずれの接続方法を使用しても、オンプレミスのアプリケーションは Azure Red Hat OpenShift で稼働しているアプリケーションに接続でき、その逆も可能です。

オンプレミス環境から Azure Red Hat OpenShift に接続する場合、通常、ハブ仮想ネットワークを介して接続します。図 4.3 は、Azure ExpressRoute との接続の仕組みを説明した図です。

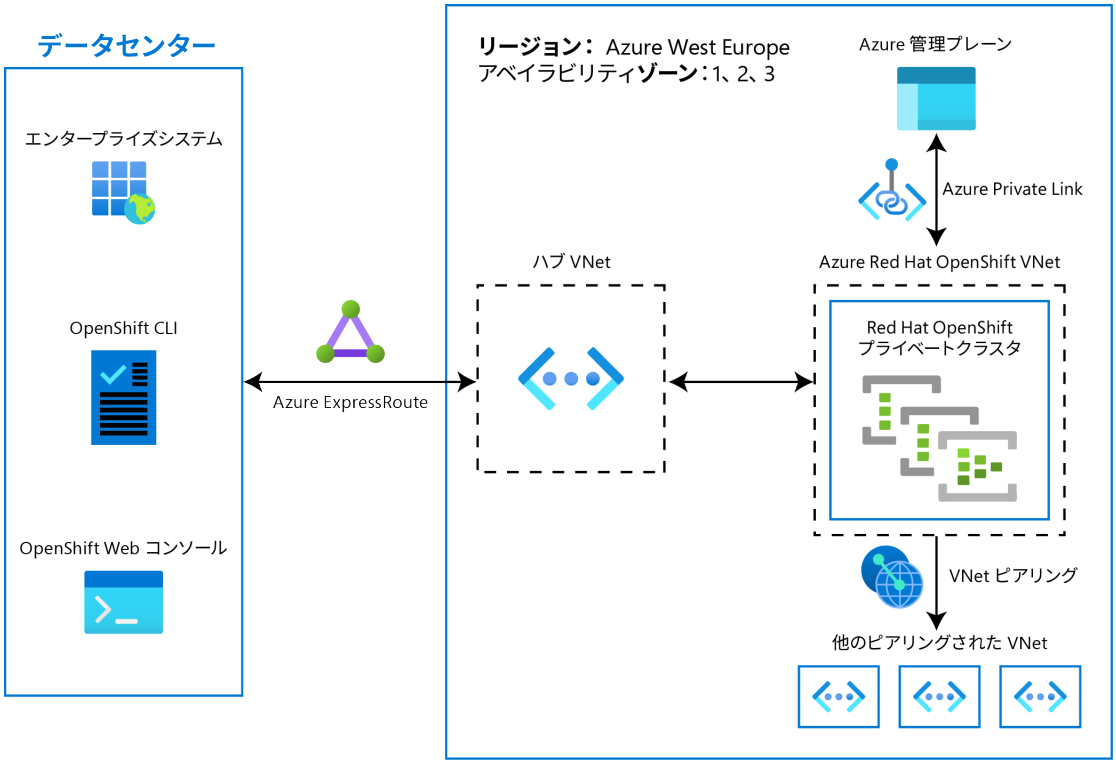


図 4.3：Azure ExpressRoute を使用した接続

前の図は、Azure Red Hat OpenShift に接続する Azure ExpressRoute のアーキテクチャの概要を示しています。この図は Azure ExpressRoute について説明したものですが、VPN 経由の接続も概念的には極めてよく似ています。

ハイブリッド・アーキテクチャで実行されるアプリケーションに関する考慮事項

オンプレミス環境に再接続する Azure Red Hat OpenShift でアプリケーションを実行する場合、アプリケーションの所有者が次のようなことを考慮する必要が生じる可能性があります。

- **接続の待機時間**：Azure ExpressRoute の場合、オンプレミス環境への再接続は比較的低遅延ですが、パブリックインターネットを経由する VPN ネットワークでは、かなり長い遅延が生じる可能性があります。Web サーバーなど、一部のアプリケーションでは、接続を介して HTTP トラフィックを送信しているだけなので、これによって問題が発生することはほとんどありません。しかし、その Web サーバーで稼働しているサーバー側のアプリケーションが、オンプレミスで稼働しているデータベースに接続する必要がある場合は、パフォーマンスにかなりの影響を与える可能性があります。
- **Ingress/Egress トラフィックのコスト**：一部の接続タイプでは、Azure 経由または接続プロバイダーからの Ingress および Egress トラフィックに対して料金が発生します。予防策として、まずテスト環境でコストを算出し、次にピーク負荷時のコストを予測して、予想外の事態が生じないようにしましょう。
- **障害シナリオ**：Azure ExpressRoute は永続的で堅牢な接続を提供するように設計されていますが、VPN 接続は頻繁に中断したり、接続と切断を繰り返したりすることがよくあります。この接続はパブリックインターネットを通るため、VPN 接続の遅延とサービス品質は 1 日を通してかなり頻繁に変動する可能性があります。接続が最適なパフォーマンスで実行されている場合だけでなく、ハイブリッドリンクの状態が悪い場合のアプリケーションのパフォーマンスをテストすることが重要です。また、接続が数時間にわたって切断された場合、Azure Red Hat OpenShift で実行されているアプリケーションはサービス品質が低下しても実行を継続できるのか、その可用性は完全にハイブリッド接続に依存しているのかを確認しておくことも必要です。

この他にも、組織の既存の内部チェックリストからいくつかのポイントを確認する必要があるかもしれませんが、ハイブリッド・アーキテクチャがどのように機能するべきかを考察する手始めとしては、上記のポイントの確認で十分でしょう。

まとめ

本章では、Azure Red Hat OpenShift をデプロイする前に確認と検討が必要な、プロビジョニングの準備として考慮しておくべき一般的な事項の多くについて説明しました。次の章では、実際にクラスタをデプロイする際に役立つ資料をいくつか紹介します。

第 5 章

Azure Red Hat OpenShift クラスタのプロビジョニング

ここまでの内容をざっと振り返りましょう。本書では、ここまでで次のことを取り上げました。

- 第 2 章：Red Hat OpenShift 入門では、Red Hat OpenShift の概要を簡単に説明し、素の Kubernetes ではなく OpenShift を選択するメリットについて説明しました。
- 第 3 章：Azure Red Hat OpenShift では、Azure Red Hat OpenShift マネージド・クラウドサービスの詳細を説明し、アーキテクチャ、管理、認証、サポート、および料金に関する考慮事項など、主要な概念を取り上げました。
- 第 4 章：プロビジョニングの準備 – エンタープライズ・アーキテクチャに関する考慮事項では、Azure Red Hat OpenShift のデプロイ前に考慮する必要がある一般的な事柄について説明しました。

これでクラスタのプロビジョニングとデプロイを行う準備ができました。それでは、ドキュメントサイトで公式のデプロイメント手順を確認しましょう (プロビジョニングはデプロイメントの一環です。デプロイメントのプロビジョニングとは、デプロイメントをサポートするために必要な IT インフラストラクチャの準備ができていることを確認することです)。

チュートリアル – Azure Red Hat OpenShift 4 クラスタを作成する

長期的な観点からコマンドは変更される可能性があるため、本章では、クラスタの作成で入力する個々のコマンドについては説明しません。これはドキュメントで詳しく説明しています。

本章では、プロセス全体に関するガイダンスと、クラスタのデプロイ時に考慮すべきことについて説明します。

手動のデプロイ – 所要時間の見込み

組織によっては、クラスタのプロビジョニングにかかる時間を理解する必要があります。詳しく確認していきましょう。

デプロイメントの前提条件となる大まかなプロセスは次のとおりです。

- az コマンドラインをシステム管理者のワークステーションにデプロイ
- Azure サブスクリプションで使用するためにリソースプロバイダーを登録
- Red Hat プルシークレット (オプション)
- クラスタのドメイン (オプション)
- 2 つの空のサブネットを持つ仮想ネットワーク (1 つはコントロールプレーン・ノード用、もう 1 つはアプリケーションノード用)

これらの前提条件を設定するための所要時間に関しては、Azure および Red Hat OpenShift の熟練した管理者であれば、これらのタスクに初めて取り組む場合は 30 分、手作業でこれらの手順を繰り返し実行する場合は 10 分程度で完了できるでしょう。

その後の自動化されたプロビジョニングプロセスには、対象 Azure リージョンのアクティビティにもよりますが、通常 25 分から 40 分かかります。

デプロイの自動化

Azure Red Hat OpenShift のプロビジョニングプロセスが自動化されていることを踏まえて、組織は通常、ツールを使用して前提条件のステップ (ネットワーク、サブネット、サービス原則などの作成) も自動化したいと考えます。

これらの手順を自動化できるツールは数多くあります。ここで、お勧めのツールをいくつか紹介します。

- az コマンドラインツール：自動化されている場合、このツールは通常、CI/CD プロセスの一部としてコンテナなどにインストールされます。ここで使用される一般的なツールには、Jenkins、Azure DevOps などがあり、場合によっては Ansible も含まれます。az コマンドラインツールのデプロイが必要なのは一度だけですが、追加のクラスタでは、組織のさまざまな部門に対応した Azure サブスクリプション ID の設定が必要な場合があります。
- Azure サブスクリプションで使用するために登録されたリソースプロバイダー：すでに述べたとおり、これは az コマンドラインツールのセットアップに含まれます。
- Red Hat プルシークレット (オプション)：Red Hat は、プルシークレットの取得をサポートする REST API を文書化しています。詳細はこちらの記事を参照してください。
- クラスタのドメイン (オプション)：これは DNS レコードの作成方法によって異なりますが、Azure DNS を使用する場合は、Terraform や Ansible などの一般的な Azure 自動化ツールで作成できます。
- 2 つの空のサブネットを持つ仮想ネットワーク (1 つはコントロールプレーン (マスター) ノード用、もう 1 つはアプリケーション (ワーカー) ノード用)：これは通常、一般的な Azure 自動化ツールによって自動化されます。Terraform や Ansible などのツールを使用して、このネットワークとサブネットを作成できます。

前提条件のステップを手作業で行う場合は 30 分あるいは 10 分といった時間がかかりますが、自動化すればその時間をわずか 1 - 2 分に短縮できます。クラスタのデプロイを高速化することはできませんが (常に 25 分から 40 分を要する)、現実的に、これはオンプレミスに比べて非常に迅速なエンドツーエンドのデプロイメントプロセスになり得ます。

デプロイメントの自動化には、プロセスの高速化以外にも多くの利点があります。デプロイメントの自動化を使用する場合、ログ記録と監査が容易な、堅牢で反復可能なプロセスを構築できるという利点があります。また、お客様がセルフサービスカタログのアイテムを独自のポータルに組み込むのも非常に一般的であり、チームはクラウド・プラットフォーム・チームとやり取りしなくても、Azure Red Hat OpenShift クラスタのプロビジョニングとデプロビジョニングを容易に行うことができます。

クラスタへのアクセス

このセクションでは、プロビジョニング後に Azure Red Hat OpenShift クラスタにアクセスする方法について簡単に説明します。

Web UI 経由

CLI をインストールしている場合は Bash シェルから、または Azure ポータルの Azure Cloud Shell (Bash) セッションから、次のコマンドを実行してクラスタのサインイン URL を取得します。

```
az aro show -n $CLUSTER_NAME -g $RG_NAME --query "consoleProfile.url" -o tsv
```

`openshift.xxxxxxxxxxxxxxxxxxxx.eastus.aroapp.io` のような文字列が返されます。クラスタのサインイン URL は、`https://` の後に `consoleProfile.url` 値が続きます。たとえば、`https: //openshift.xxxxxxxxxxxxxxxxxxxx.eastus.aroapp.io` のようになります。

この URL をブラウザーで開きます。kubeadmin ユーザーでログインするように求められますので、インストールプロセス中にインストーラーから提供されたユーザー名とパスワードを使用してください。

ログインすると、Azure Red Hat OpenShift Web コンソールが表示されます。

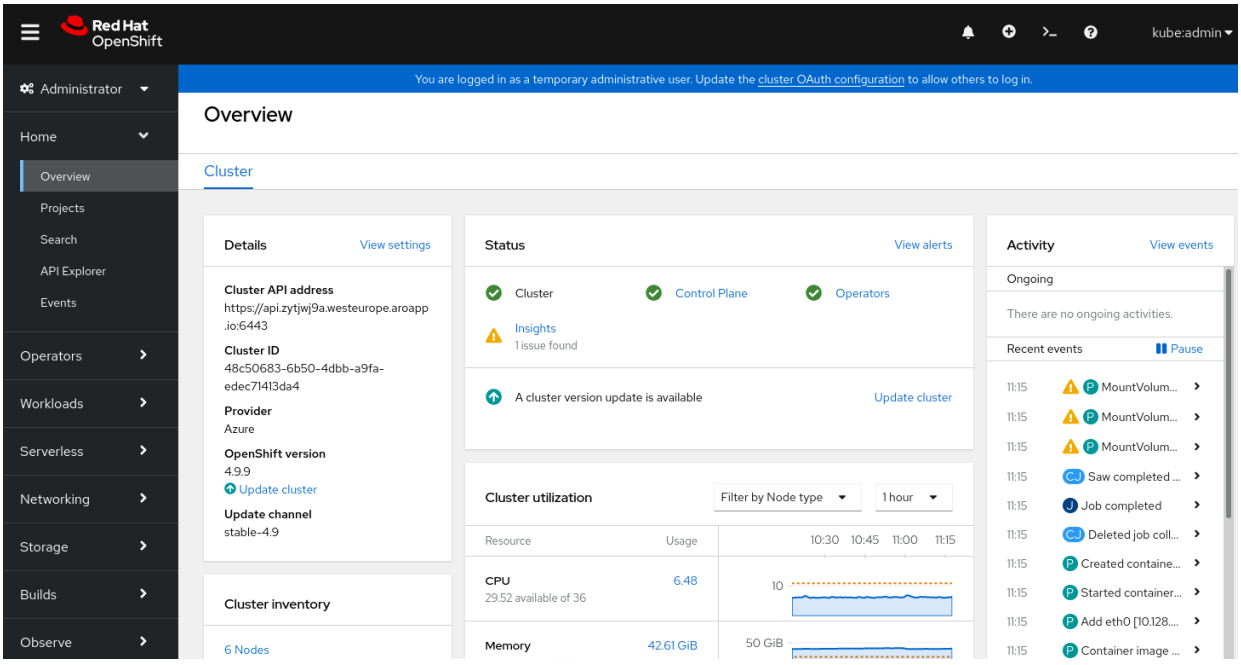


図 5.1：Azure Red Hat OpenShift Web コンソール

少し時間を取って、Web コンソールの内容を確認してみましょう。OpenShift の最新バージョンが実行されているはずです。また、すべてのコンポーネントは正常な状態になっているか、インストール後すぐに正常な状態に移行していきます。

OpenShift CLI (oc) 経由

最新の OpenShift CLI (oc) をダウンロードする必要があります。そのためには、ログインして https://console.redhat.com/openshift/downloads にアクセスし、ページを表示します。

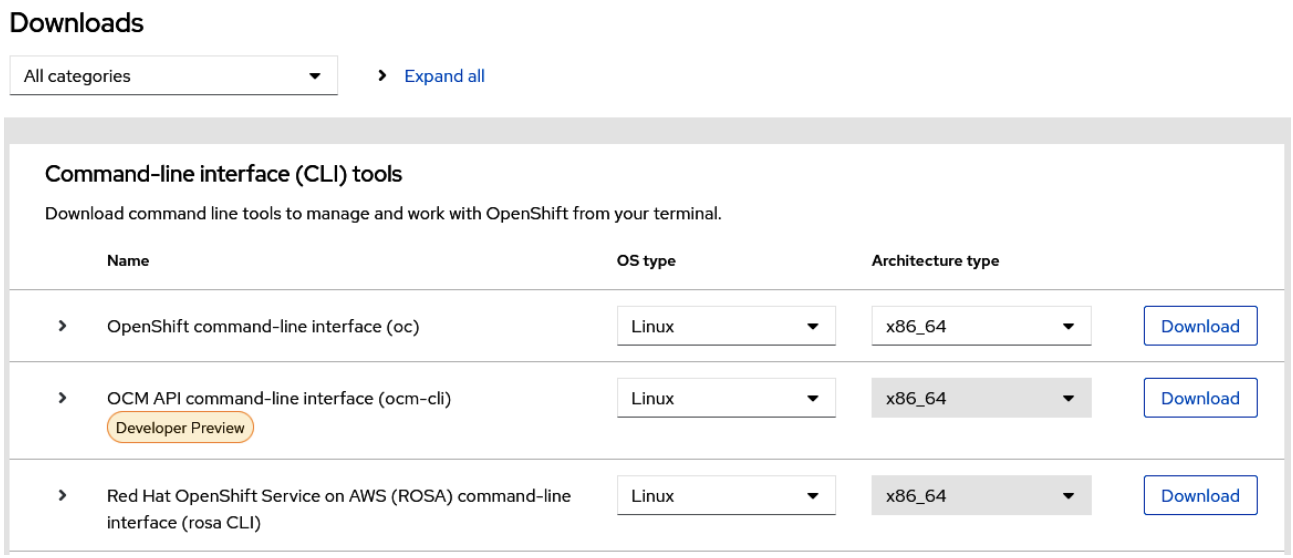


図 5.2：OpenShift コマンドライン・インタフェースのダウンロード

アーカイブ (.tar.gz または .zip) をシステム内の任意の場所に抽出してから、パスに oc コマンドを配置します。Linux では、oc コマンドを /usr/local/sbin/ に配置するのが一般的です。

OpenShift CLI の実行とクラスタへのログイン

コマンドラインからクラスタに対して認証するには、Web コンソールからログインコマンドとトークンを取得する必要があります。ブラウザーで Web コンソールにログインし、右上のユーザー名をクリックして **Copy login command** をクリックします。

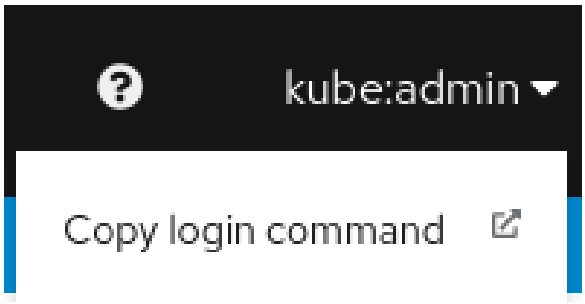


図 5.3：Copy login command

次のような新規のページが開きます。

図 5.4：API トークンを使用したログイン

次に、このコマンドをコピーしてターミナルに貼り付け、Azure Red Hat OpenShift クラスタにログインします。

たとえば、Azure ポータルで Azure Cloud Shell (Bash) セッションを使用している場合、そのログインコマンドを貼り付けると、次のような oc status コマンドが表示されます。

```
user@Azure: oc status
In project default on server https://api.cyki1k6g.westeurope.aroapp.io:6443

svc/openshift - kubernetes.default.svc.cluster.local
svc/kubernetes - 172.30.0.1:443 -> 6443

View details with 'oc describe <resource>/<name>' or list everything with 'oc get all'.
```

ここで、oc コマンドラインを詳しく確認し、新しい Azure Red Hat OpenShift 環境を理解してもよいでしょう。経験豊富な OpenShift 4 ユーザーにとっては、Azure Red Hat OpenShift のこのクラウドサービスは非常に馴染みのあるもので、その動作は、過去に使用した他の OpenShift 4 環境とまったく同じのはずです。

まとめ

公式のプロビジョニング手順が適切に維持されているため、本章は簡潔にまとめました。公式の手順は、Azure サブスクリプションで Azure Red Hat OpenShift を実行するための最も信頼できるガイドだと覚えておいてください。このプロビジョニング手順は、自己管理型の OpenShift 環境のプロビジョニングに必要な手順よりもかなり単純であることがおわかりいただけると思います。これは、相当な量のエンジニアリングが Azure Red Hat OpenShift サービスに組み込まれ、Azure との緊密な統合を提供するとともに、十分にテストされ、十分にサポートされている規範的な「万能」アーキテクチャをデプロイしているためです。概して、Azure Red Hat OpenShift のプロビジョニングに費やす時間が少なくなり、アプリケーションのデプロイにより多くの時間をかけることができるため、これは組織にとってメリットになります。

第 6 章

プロビジョニングの後処理 – Day 2

Azure Red Hat OpenShift をデプロイした後、クラスタをプロダクションで使用できるようにするために通常必要なアクティビティがいくつかあります。本章では、プロビジョニング後に行う一般的なアクティビティの多くについて取り上げます。

- 認証 – Azure Active Directory – ユーザーグループとコミュニティオペレーターを同期する方法など
- Operator – OperatorHub など
- ロギングについて – ログ転送など
- モニタリングについて – OpenShift コンテナのモニタリングなど
- アップグレードとパッチ – サポートされているバージョンのライフサイクルなど
- クラスタのスケーリング – クラスタの手動および自動スケーリングなど
- アプリケーションのスケーリング – アプリケーションのスケーリングに関する簡単な注意事項
- プルシークレットのセットアップ – OpenShift Cluster Manager への登録
- 制限範囲 – 制限範囲を適用できるケースなど
- 永続ストレージ – 利用可能でサポートされているストレージクラスなど
- セキュリティとコンプライアンス – セキュリティ管理に関する注意事項

各セクションでは、プロビジョニング後のさまざまなタスクについて詳述しており、新たにデプロイされたクラスタを取得してプロダクションのアプリケーションに対応できるようにするために必要なことを理解するのに役立ちます。

認証 – Azure Active Directory

Azure Red Hat OpenShift は、OpenShift ドキュメントにリストされている認証プロバイダーをすべてサポートします。認証プロバイダーの全リストを参照してください。しかし、お客様の大多数は、Azure ですでに利用可能な Azure Active Directory を使用して、Azure Red Hat OpenShift への認証とシングルサインオンを提供するものと思われます。

Azure Active Directory 統合の構成は、意図的に「Day 2」オペレーションとしています。それは、組織が別の構成プロバイダーを使用したい場合があるからです。Azure Active Directory のセットアップとインストールのプロセスは、設定が初めての場合には 15 分程度かかりますが、プロセスに慣れてくれば 5 分程度で完了できるようになるでしょう。多くの組織では、ARM テンプレートや Ansible Playbook などのツールを使用して、このプロビジョニングの一部を自動化しています。

- Azure Red Hat OpenShift 用に Azure Active Directory を構成する (グラフィカルポータル)
- Azure Red Hat OpenShift 用に Azure Active Directory を構成する (コマンドライン・インタフェース)

Active Directory ユーザーグループの使用

Azure Active Directory 認証の構成では、ユーザーは既存の資格情報でのみログインできます。ユーザーの既存のユーザーグループを Red Hat OpenShift に移行することはありません。一般的には、Active Directory からユーザーグループをインポートし、ユーザーグループを使用して OpenShift 内でユーザー権限を構成できるようにします。これを行うためには、コミュニティがサポートする Group Sync Operator を利用できます。

- GitHub の Group Sync Operator

注意していただきたいのが、Group Sync Operator はコミュニティによるサポートで提供されているということです。つまり、執筆時点で、Red Hat またはマイクロソフトが正式にサポートしているものではありません。本書では、プロジェクトの README ファイルに付属している Operator 向けの詳細な構成とセットアップの手順に従うことをお勧めします。

Operator

OpenShift 4 の Operator はプラットフォームの基本的な構成要素の 1 つであり、Red Hat OpenShift の利用者に多大な価値をもたらします。Operator はコードから構築され、クラスタ内のコンテナでサービスを実行します。一部の Operator は、クラスタネットワーク、マシン構成、クラスタのアップグレードなどの保守を行います。これらはクラスタ Operator と呼ばれることが多く、OpenShift コンソールの **Administration** セクションでリストを確認できます。

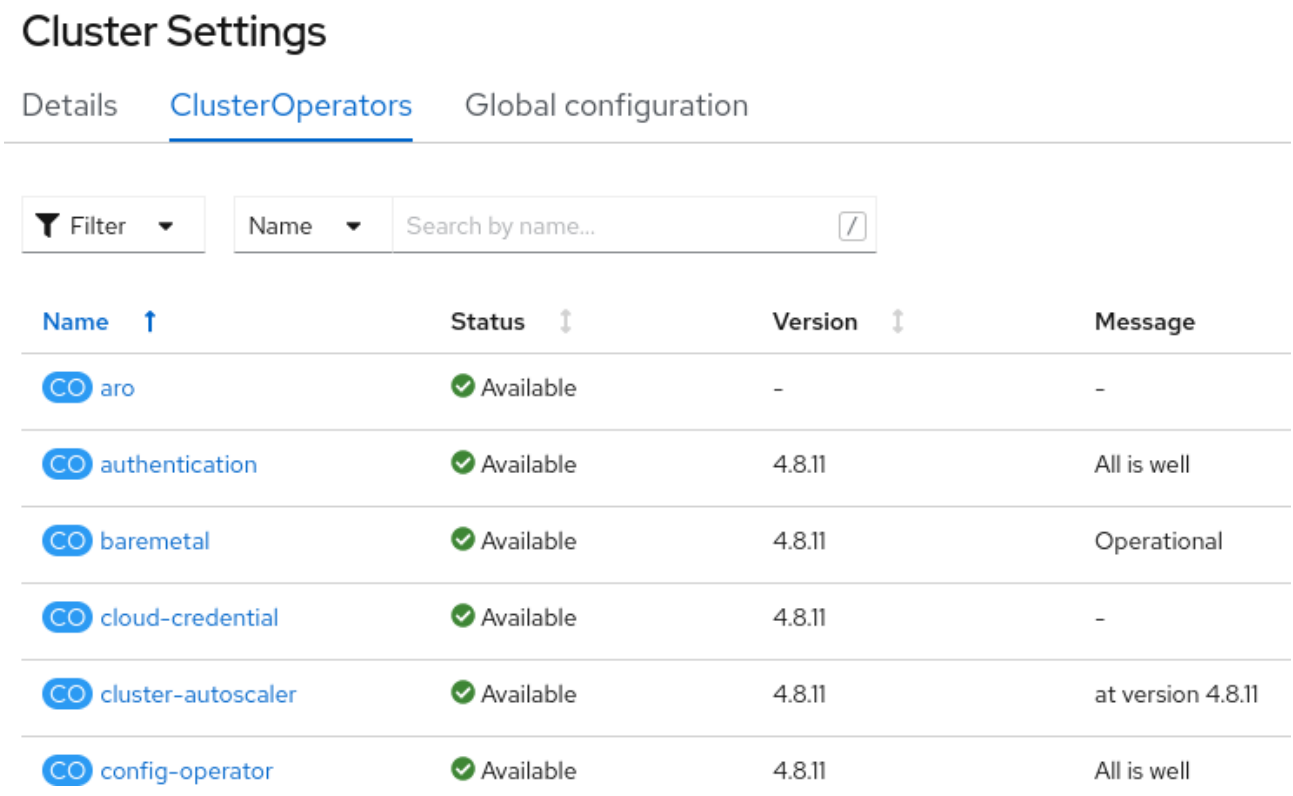


図 6.1：Cluster Settings

上記のスクリーンショットから、Azure Red Hat OpenShift 専用の Operator が存在することがわかります。この Operator はサービスの一部を担っており、クラスタがサポートされている構成状態を維持できるようにします。

Operator は、サービスの正常性、更新、パッチ、スケーリングといった多くの機能を実行できます。

OperatorHub

すべてのクラスタのバックグラウンドで実行される標準のクラスタ Operator 以外にも、管理者は OperatorHub を介してさらに多くの Operator にアクセスできます。管理者が人気のコミュニティやエンタープライズの Operator を Red Hat OpenShift インストールで利用できるようにしたいと考えた場合、OperatorHub を使用してそれらを簡単に見つけることができます。OperatorHub は、OpenShift クラスタのサイドバーにあります。

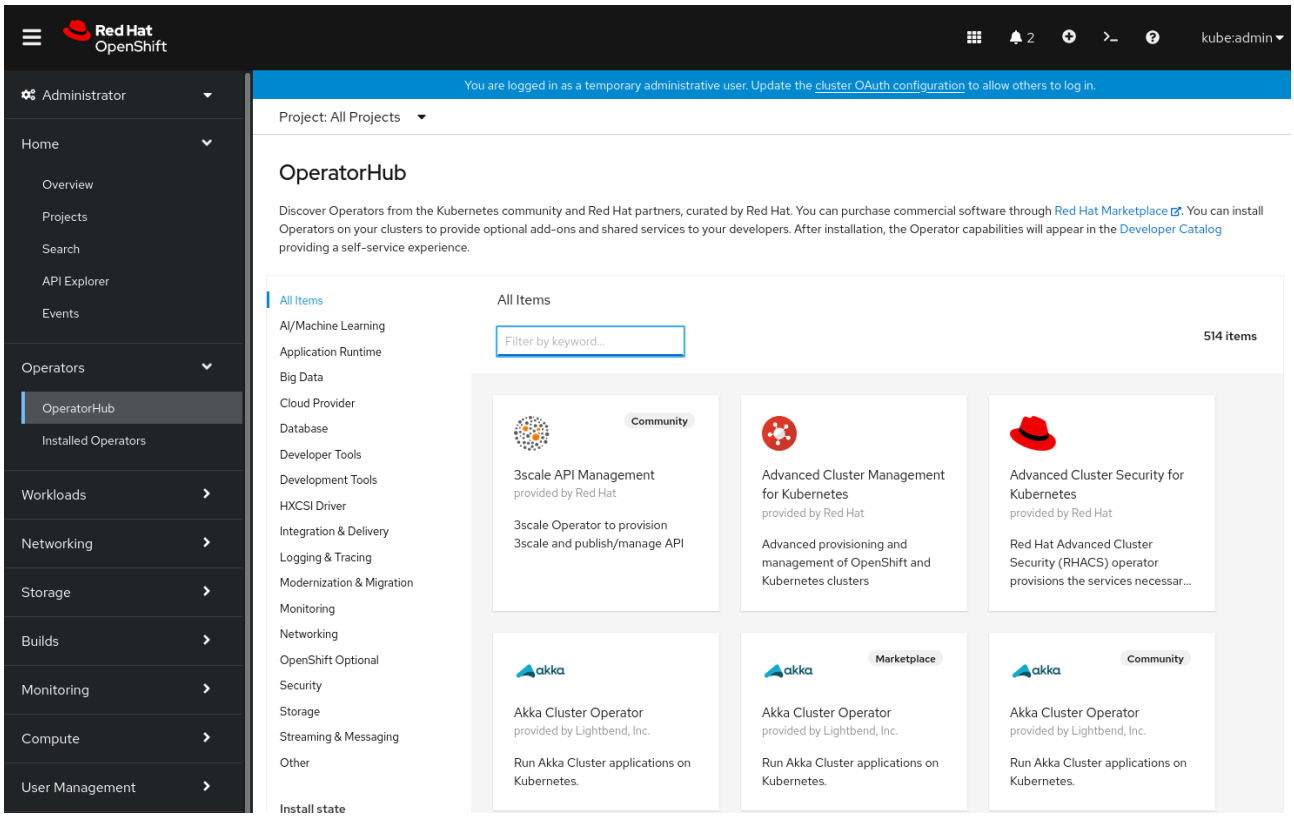


図 6.2：OperatorHub

管理者は、Operator を使用することで、Kubernetes の構成やコードについて心配することなく、人気のソフトウェアを迅速かつ容易にデプロイして維持することができます。Advanced Cluster Management、Red Hat OpenShift Service Mesh、Red Hat OpenShift Pipelines など、多くの Red Hat 製品が Operator としてパッケージされています。また、MariaDB データベース、Couchbase、IBM ブロック・ストレージ・ドライバーなど、Red Hat 以外のソフトウェア用の Operator の膨大なライブラリもあります。

Operator の詳細は、以下を参照してください。

- Operatorhub.io
- OpenShift の Operator

ロギングについて

Azure Red Hat OpenShift は、Red Hat OpenShift と同じロギングおよびモニタリング・アーキテクチャを使用します。どちらのオファリングも、ロギングに同じ Operator を使用しています。

ログは 3 つのカテゴリに分けられます。

- **アプリケーション**：クラスタで実行されているユーザーアプリケーションによって生成されるコンテナログ (インフラストラクチャ・コンテナ・アプリケーションは除く)。
- **インフラストラクチャ**：クラスタノードや OpenShift Container Platform ノードで実行されているインフラストラクチャ・コンポーネントによって生成されるログ (ジャーナルログなど)。インフラストラクチャ・コンポーネントは、openshift*、kube*、またはデフォルトのプロジェクトで実行される Pod です。
- **監査**：/var/log/audit/audit.log ファイルに保存される auditd (ノード監査システム) によって生成されるログ、および Kubernetes API サーバーと OpenShift API サーバーからの監査ログ。

OpenShift ロギングの詳細は、製品ドキュメントの Red Hat OpenShift のロギングについてを参照してください。

Azure Monitor へのクラスタログ転送の使用

一般的な統合は、Azure Red Hat OpenShift のログを Azure Monitor の Container insights サービスに送信することです。これは、Azure Log Analytics とも呼ばれます。これにより、Azure での永続的かつ低コストのログ保持が可能になります。

OpenShift のロギング・アーキテクチャは各ノードで Fluent Bit を実行します。Operator による最初のアプローチは、ログを直接送信するようにこの Fluent Bit サービスの構成を調整することです。しかし、Azure Red Hat OpenShift でのロギング構成の調整は、サポートポリシーの範囲外です。OpenShift には、`ClusterLogForwarder` という、サポートされた状態でログを転送するメカニズムが組み込まれています。

`ClusterLogForwarder` を使用すると、Elasticsearch、Flutentd、syslog にログを転送できます。ただし、Azure Monitor はこれらのプロトコルのいずれも直接サポートしていません。次善の策は、OpenShift からログを受信して Azure Monitor に転送する Fluent Bit インスタンスを中間に追加することです。このアプローチについては現在、マイクロソフトのドキュメントとコミュニティのドキュメントに説明があります。

モニタリングについて

Azure Red Hat OpenShift はマネージドサービスであるため、複雑なカスタムモニタリングを実装する必要はありません。これは、有料サービスの一部として提供されます。

しかし、Azure Container insights を使用して OpenShift コンテナのモニタリングを行うのは極めて一般的です。これは現在公開プレビュー中の機能であり、こちらのドキュメントで説明されています。

アップグレードとパッチ

Azure Red Hat OpenShift クラスタをプロビジョニングする場合、更新チャネルは選択されません。つまり、デフォルトでは、クラスタは更新の受信を開始しません。

更新チャネルを表示するには、ナビゲーションサイドバーから **Administration** → **Cluster Settings** に移動します。Azure Red Hat OpenShift クラスタがプロビジョニングされた直後のクラスタ設定のビューは、次のとおりです。

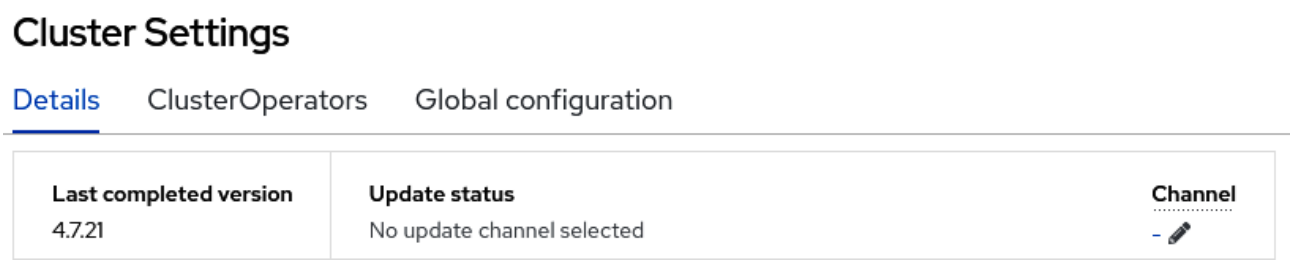


図 6.3：クラスタがプロビジョニングされた直後の Cluster Settings

更新チャネルを選択するには、「-」リンクを選択して、利用可能なオプションを確認します。

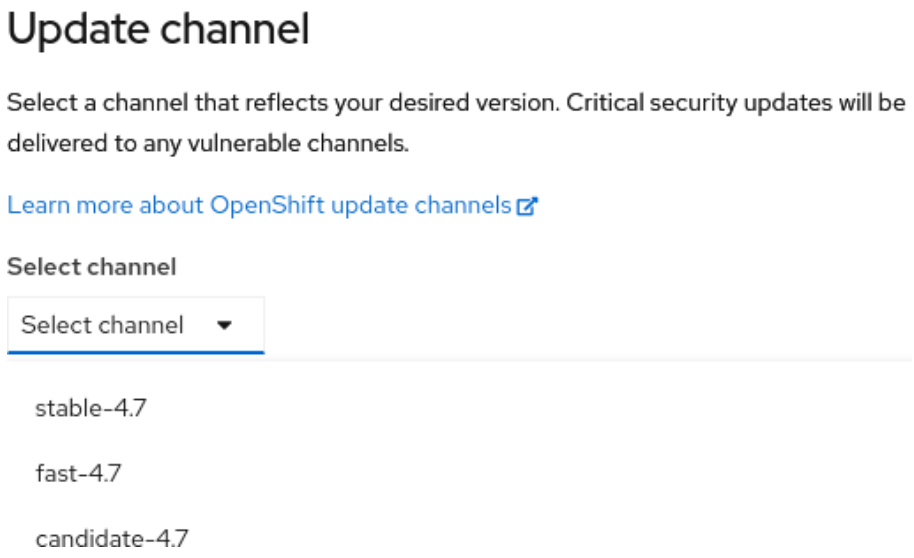


図 6.4：更新チャネルの選択

`stable`、`fast`、`candidate` の違いについては、チャネルとリリースのアップグレードに関するドキュメントページを参照してください。

プロダクションのクラスタでは常に **stable** チャネルを実行することを強くお勧めします。

サポートされているバージョンのライフサイクル

Azure Red Hat OpenShift のどのバージョンがサポートされているかを知ることは、プロダクションへのデプロイを計画する上で重要です。ライフサイクルに関する公式のページには、サポートされているバージョンとサポートされていないバージョンを理解するために役立つ情報があります。

Azure Red Hat OpenShift のサポートライフサイクルのページ

そのページに記載されている情報から抜粋して、以下に簡単にまとめました。

Azure Red Hat OpenShift は、Red Hat OpenShift Container Platform の 2 つの **GA (一般提供)** マイナーバージョンをサポートします。

- Azure Red Hat OpenShift でリリースされた最新の GA マイナーバージョン (ここでは N と呼びます)
- 1 つ前のマイナーバージョン (N-1)

Red Hat OpenShift Container Platform はセマンティック・バージョニングを使用します。セマンティック・バージョニングでは、さまざまなレベルのバージョン番号を使用してバージョンを明示します。次の表は、セマンティックバージョン番号の表示例を示しています。この例では、バージョン番号 4.9.3 を使用しています。

メジャーバージョン (x)	マイナーバージョン (y)	パッチ (z)
4	9	3

バージョンの各番号は、以前のバージョンとの全般的な互換性を示しています。

- **メジャーバージョン**：現時点では、メジャーバージョンのリリースは予定されていません。互換性のない API の変更や、下位互換性が失われる可能性がある場合、メジャーバージョンが変更されます。
- **マイナーバージョン**：約 3 カ月ごとにリリースされます。マイナーバージョンのアップグレードには、機能の追加、機能拡張、廃止、削除、バグ修正、セキュリティ機能の強化などが含まれる場合があります。
- **パッチ**：通常、毎週、または必要に応じてリリースされます。パッチバージョンのアップグレードには、バグ修正とセキュリティ強化が含まれる場合があります。

サポートされているバージョンについて詳しくは、Azure Red Hat OpenShift サポートライフサイクルのページを参照してください。

クラスタのスケーリング

Red Hat OpenShift Container Platform、ひいては Azure Red Hat OpenShift は、スケーラブルなアーキテクチャを念頭に置いて構築されています。OpenShift の文脈でスケーリングを計画する場合、通常、管理者とアプリケーションの所有者は、クラスタのスケーリングとアプリケーションのスケーリングを別個のものとして考慮する必要があります。

このセクションではクラスタのスケーリングについて説明し、アプリケーションのスケーリングについては別のセクションで簡単に説明します。

サポートされている最大数

クラスタのスケーリングとは、クラスタにワーカーノードを追加することです。これにより、クラスタで実行されているアプリケーションのコンピューティング能力が向上します。当然、コントロールプレーンをスケーリングするタイミングについても検討が必要です。Azure Red Hat OpenShift はすでに 3 つのコントロールプレーン・ノードをデプロイしています。これらは原則として、Azure Red Hat OpenShift クラスタでサポートされるワーカーノードの最大数 (現在は 60) まで拡張するのに十分な容量を備えています。

執筆時点では、サポートされているコントロールプレーン・ノードのインスタンスサイズは、`Standard_D8s_v3`、`Standard_D16s_v3`、`Standard_D32s_v3` の 3 つです。

ワーカーノードでサポートされる Azure インスタンスの種類は他に、コンピューティング最適化 (F シリーズ)、メモリー最適化 (E シリーズ)、汎用 (D シリーズ) があります。

Azure Red Hat OpenShift で現在サポートされている Azure インスタンスの種類のリストは、サポートポリシーのページで確認してください。

最小のデプロイメントとゼロへのスケーリング

時折、テストや開発、または可用性の低下が許容されるユースケース向けに、小規模のクラスタをデプロイしたい場合があります。現在、クラスタの最小サイズは 3 つのコントロールプレーン・ノードと 3 つのアプリケーションノードであり、それよりも小さいスケーリングはサポートされていません。

手動によるクラスタのスケーリング

運用者は Azure ポータルを見て、Azure 仮想マシンを直接作成すれば Azure Red Hat OpenShift クラスタにワーカーノードを追加できると考えるかもしれません。しかし、Azure Red Hat OpenShift を含むリソースグループは Azure 管理者のパースペクティブから「ロック」されているため、これは不可能です。これは権限とは関係ありません。**所有者**の権限を持つユーザーであっても、Azure Red Hat OpenShift リソースグループのコンテンツを変更することはできません。したがって、Azure Red Hat OpenShift リソースグループ内で仮想マシンを手動で作成しようとすると、権限拒否エラーとなります。

Azure Red Hat OpenShift のスケーリングで意図されているメカニズムは、OpenShift MachineSets 機能を使用することです。これにより、OpenShift は指示されたときに新しいアプリケーションノードをデプロイします。cluster-admin 権限を持つユーザーの場合、**Compute** に **MachineSets** が表示されます。

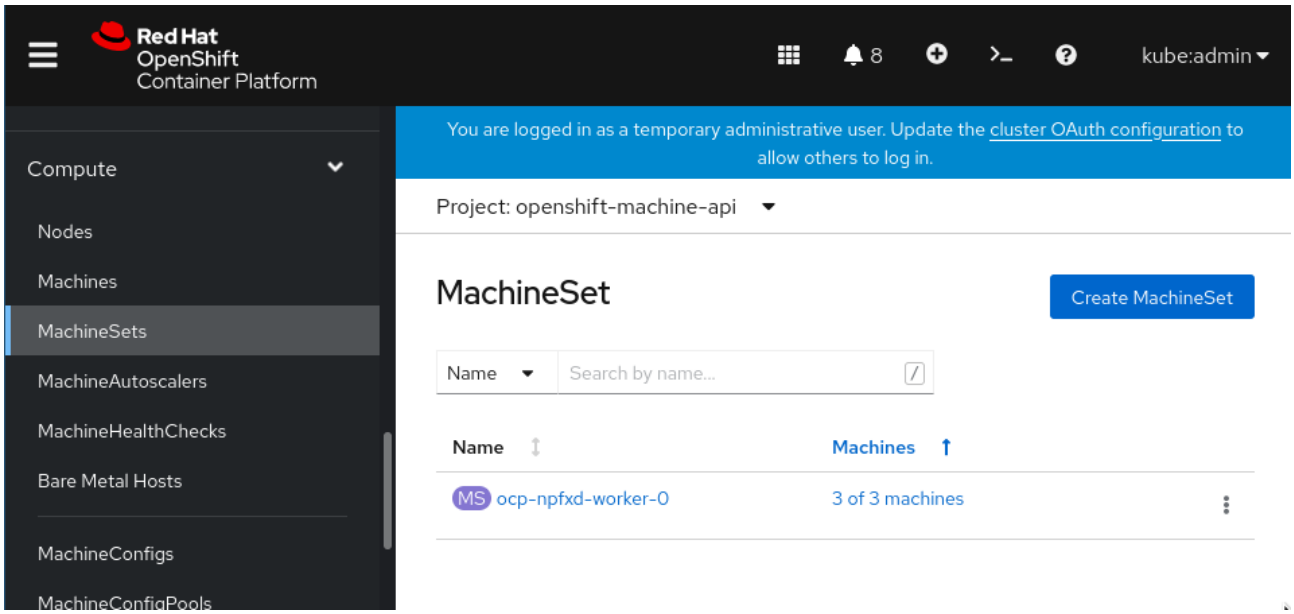


図 6.5：MachineSets への管理者アクセス

MachineSet アプリケーションノードの「編集」メニューをクリックすると、**Edit Machine Count** を選択することで、新しいアプリケーションノードの仮想マシンを簡単にプロビジョニングできることがわかります。

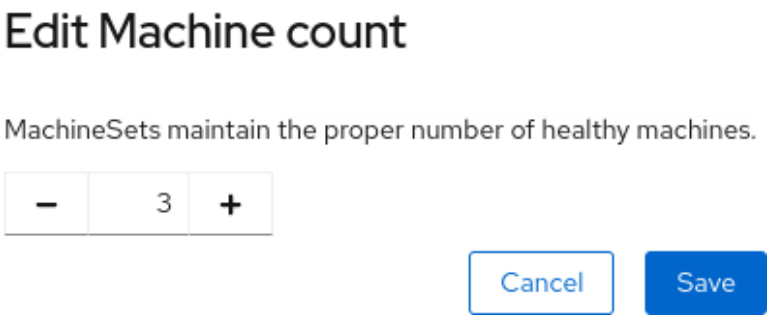


図 6.6：マシン数の編集

数が更新されると、管理者は Azure ポータルまたは **Compute** → **Machines** ビューに移動して、プロビジョニングされている新しいアプリケーションノードの仮想マシンを確認できます。

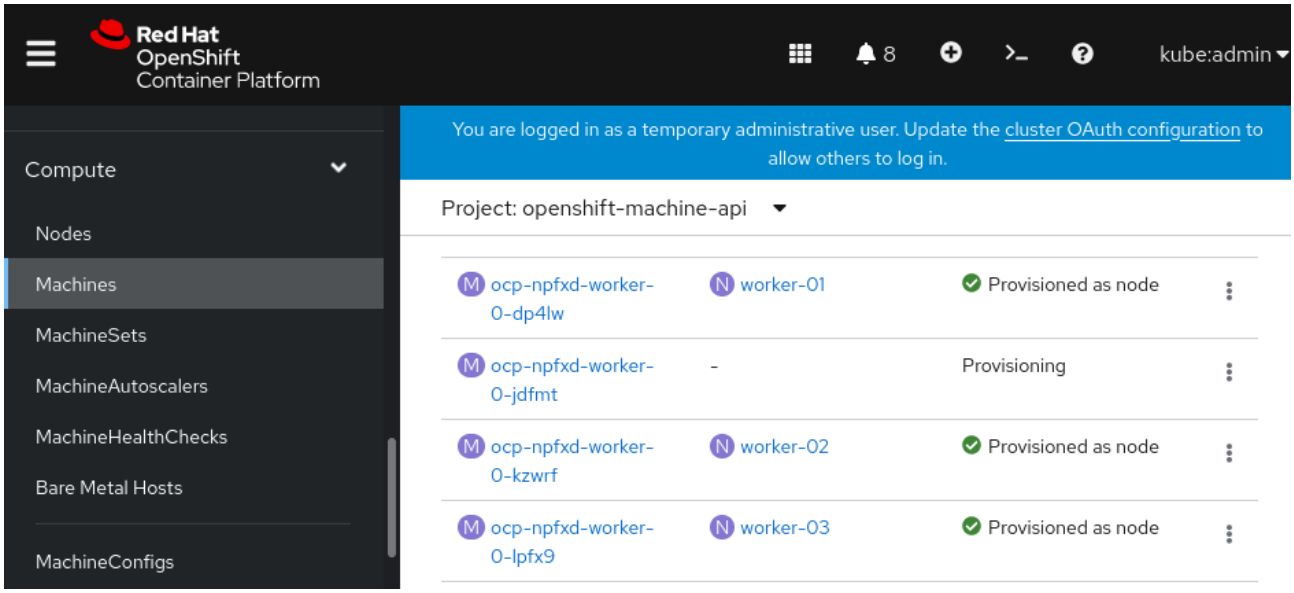


図 6.7：Machines のビュー

ほとんどの Azure リージョンで、追加された仮想マシンをプロビジョニングするためにかかる時間は、通常、最大で約 5 分です。

この追加の容量をオンラインにするために、開発者がプロビジョニング後に行う作業はありません。OpenShift が自動的にそれをクラスタで利用できるようにし、アプリケーションが必要に応じて利用します。

クラスタの自動スケーリング

デフォルトでは、Azure Red Hat OpenShift のデプロイメントは、自動スケーリング機能が稼働するように構成されていませんが、この機能は簡単に有効にできます。管理者が `MachineAutoscaler` リソースを作成するだけです。これは、Azure Red Hat OpenShift の **Compute** サイドバーメニューにあります。

MachineAutoscaler リソースは MachineSet で動作し、MachineSet は必要に応じてアプリケーションノードの仮想マシンの容量を作成または削除します。次の例は、MachineSet で最小数または最大数のマシンを維持できることを示しています。

```
apiVersion: autoscaling.openshift.io/v1beta1
kind: MachineAutoscaler
metadata:
    name: worker-us-east-1a
    namespace: openshift-machine-api
spec:
    minReplicas: 1
    maxReplicas: 12
    scaleTargetRef:
        apiVersion: machine.openshift.io/v1beta1
        kind: MachineSet
        name: worker
```

OpenShift には、クラスタオートスケーラーの概念もあります。これにより、一定量の RAM、CPU、または同様のものを利用可能な状態に保つことでスケーリングが可能になります。MachineAutoscaler と ClusterAutoscaler のどちらを選択するかは、クラスタの容量をどのように追加および削除するかによって異なります。

MachineAutoscaler および ClusterAutoscaler に関する Red Hat OpenShift のドキュメント

アプリケーションのスケーリング

マイクロサービス・アプリケーションのスケーリングは複雑であり、この e ブックでは取り上げません。その代わり、このトピックについて詳しく知るために役立つ 2 つの資料を紹介します。

- HorizontalPodAutoscaler：実行する Pod の最小数と最大数のほか、Pod がターゲットとする CPU 使用率またはメモリー使用率を指定する
- サーバーレスと、Knative によるゼロへのスケーリング

クラスタは、実行中のコンテナとクラスタの残りの容量を監視します。Knative または HorizontalPodAutoscaler が、クラスタ内で現在使われていないリソースよりも多くのリソースを要求した場合、ClusterAutoscaler または MachineSet は、要求されたリソースをクラスタに追加する必要があります。

このアプローチを使用すると、アプリケーションとクラスタ自体が連携して、需要に応じてスケールアップしたり、スケールダウンしたりすることができます。

プルシークレットのセットアップ (OpenShift Cluster Manager への登録)

Azure Red Hat OpenShift の新規デプロイメントには、cloud.redhat.com 用に構成された「プルシークレット」がありません。つまり、デフォルトでは、クラスタは Red Hat Hybrid Cloud Console (http://console.redhat.com) に表示されません。このコンソールは OpenShift Cluster Manager と呼ばれます。

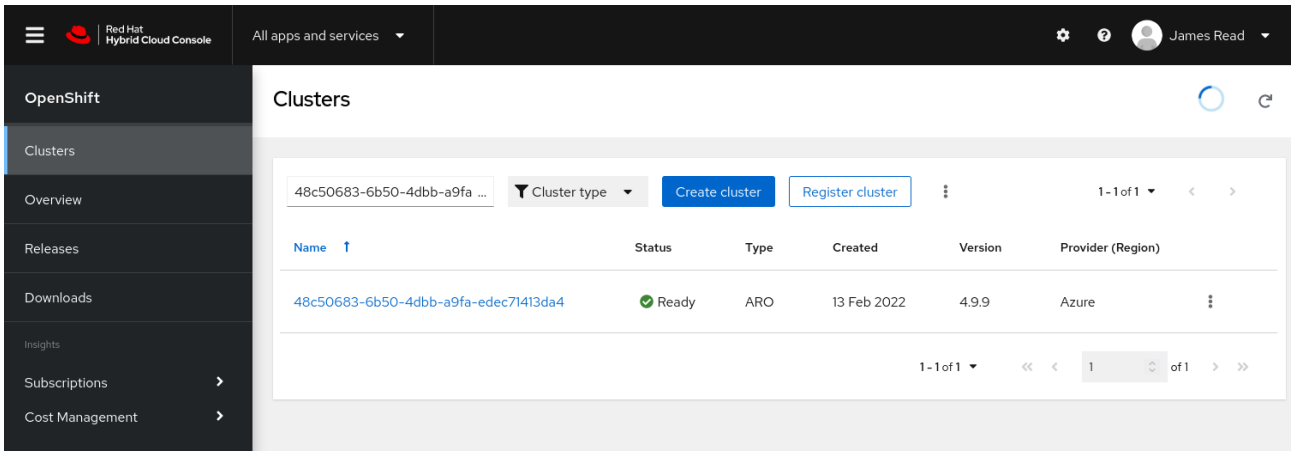


図 6.8：OpenShift Cluster Manager に表示された Azure Red Hat OpenShift クラスタ

cloud.redhat.com のプルシークレットの構成は極めて簡単で、OpenShift Cluster Manager ポータル (http://console.redhat.com) にも表示されます。また、標準のサポートチケットシステムを介して、Red Hat を使用して直接サポートチケットを発行することもできます。

プルシークレットの設定方法については、以下の資料を参照してください。

- プルシークレットを追加または更新する方法

プルシークレットを設定することで得られるさらなる利点は、お客様が Red Hat で直接サポートチケットを発行できることです。プルシークレットは、サポートスタッフがクラスタを確認できるようにする権利をお客様の Red Hat アカウントに付与します。Azure Red Hat OpenShift のサポートチケットの開き方は、他の Red Hat 製品と同じです。

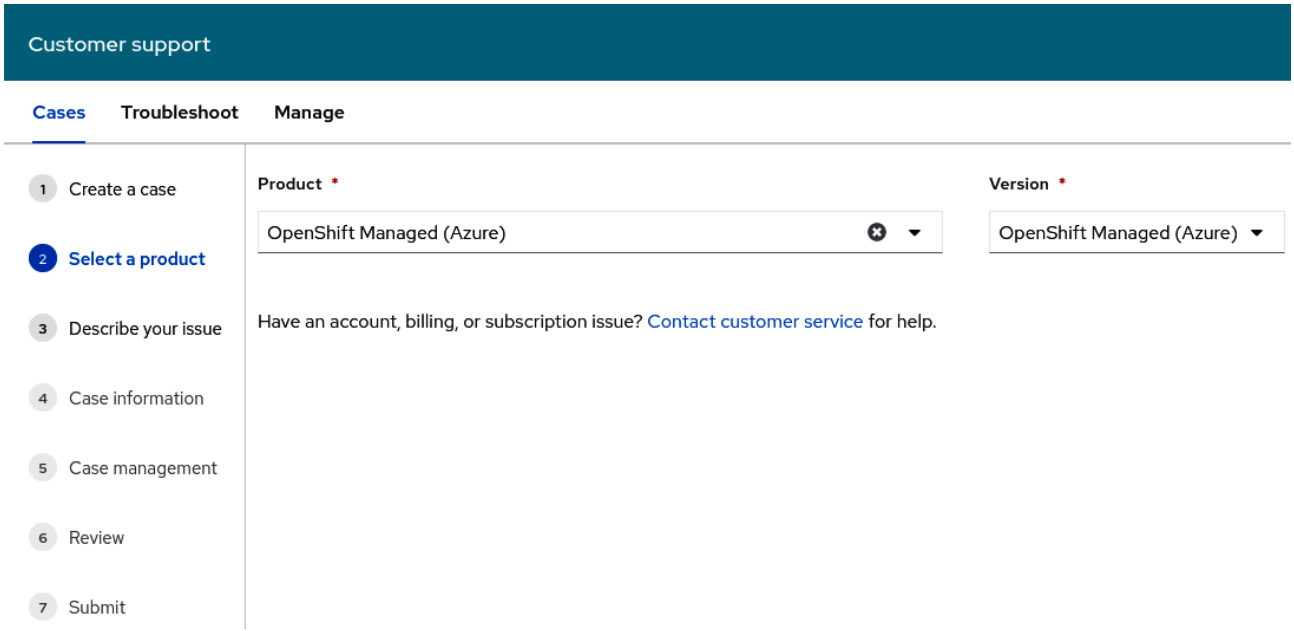


図 6.9：サポートケースの作成

制限範囲

開発チームやアプリケーションチームがコンテナ化されたアプリケーションのデプロイを開始すると、ある程度の時間が経過した後、アプリケーションが誤動作し、クラスタ上で不要なリソースを消費し始める場合があります。たとえば、アプリケーションの障害があってメモリーリークが生じるケースや、アプリケーションが誤って 100% ではなく 10% の CPU 消費でスケーリングするように設定されているケースなどです。誤動作するアプリケーションを制御できなくなってリソースを大量に消費してしまう事態を防ぐために、`LimitRange` を使用することをお勧めします。

`LimitRange` を使用すると、プロジェクト内の特定のオブジェクトのリソース消費を制限できます。`LimitRange` は、次の用途に適用できます。

- Pod とコンテナ：Pod とそのコンテナの CPU およびメモリーの最小要件と最大要件を設定できます。
- イメージストリーム：ImageStream オブジェクト内のイメージとタグの数に制限を設定できます。
- イメージ：内部レジストリにプッシュできるイメージのサイズを制限できます。
- **永続ボリュームクレーム (PVC)**：要求できる PVC のサイズを制限できます。

コンテナに適用される制限範囲の例を次に示します。ここでは、許可される最小、最大、およびデフォルトの CPU およびメモリー要求を制限します。

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio:
        cpu: "10"
```

この `LimitRange` コードスニペットは、YAML を Red Hat OpenShift コンソールのエディターに直接コピーアンドペーストするか、`oc apply -f limitrange.yaml` を使用してファイルから適用することができます。

制限範囲のドキュメント

永続ストレージ

Azure Red Hat OpenShift によってデプロイされた仮想マシンには、Red Hat CoreOS をインストールし、Azure Red Hat OpenShift サービスを実行するために、Azure ディスクが付属しています。これらのディスクは OpenShift クラスタ用で、アプリケーションに使用することはできません。

永続ストレージを必要とするアプリケーションは、Kubernetes の `PersistentVolume` 機能を使用する必要があります。また、OpenShift ドキュメントの永続ストレージについてというページでこの概念を詳細に説明していますので、ぜひ参考にしてください。Azure Red Hat OpenShift は、すべての `PersistentVolume` プロバイダーを自己管理型の OpenShift インストールとして技術的にサポートします。Azure で最も一般的に使用される永続ストレージは次のとおりです。

名前	種類	アクセスモード	ストレージクラス
Azure Files	ファイルシステム、非 POSIX 準拠	ReadWriteOnce	Azure File
Azure Disk	ブロック	ReadWriteOnce	Azure Disk
Red Hat OpenShift Data Foundation	ファイルシステム、ブロック、オブジェクト	(複数)	OCS/ODF ドキュメント

セキュリティとコンプライアンス

Red Hat OpenShift のドキュメントには、多くのセキュリティコントロールを実装してコンプライアンスを維持する方法を理解する上で参考になるコンテンツが多く含まれています。

OpenShift のセキュリティとコンプライアンスに関するドキュメント

ドキュメントの上記のセクションには次の内容が含まれます。

- OpenShift でのコンテナセキュリティの仕組みに関する詳細な説明。OpenShift は、他の Kubernetes ベースのサービスにはない、コンテナ用のすぐに使える多くのセキュリティ制御を提供することを理解することが重要です。これらの制御機能は組織とアプリケーションの安全を維持するために役立ちます。
- Pod の脆弱性のスキャン
- 監査ログへのアクセス
- 証明書の構成

これには、次のようなセキュリティ関連の便利な Operator も含まれます。

- **コンプライアンス・オペレーター**：スキャンを実行し、一般的なセキュリティ問題の修正に関する推奨事項を提供します。
- **ファイルの整合性チェック**：ファイル、特に、機密性の高いセキュリティ構成ファイルが変更されていないことを継続的に確認します。

まとめ

本章では、新規にデプロイされたクラスタをプロダクションのアプリケーションに適したものにするために、組織が通常検討するタスクとトピックの大部分を取り上げました。以後デプロイのたびにこれらのトピックをすべて再確認する必要はありませんが、クラスタを初めてデプロイする場合は、本章で説明した永続ストレージ、制限、認証などのさまざまなトピックを考慮する必要があります。

通常、組織はプロビジョニング後のタスクを可能な限り自動化しようとします。たとえば、Azure Active Directory のセットアップと構成は、ほとんどの場合、PowerShell スクリプトか ARM テンプレートを使用して実行できます。これにより、多数のクラスタをデプロイする場合に、反復的なタスクに費やされる時間を削減できます。

次の章では、プロダクション対応の Azure Red Hat OpenShift クラスタにサンプルアプリケーションをデプロイします。

第 7 章

サンプルアプリケーションのデプロイ

このガイドの内容は、主に、Azure Red Hat OpenShift を使用してプロダクションに移行するために必要なことを理解したいと考えている、開発者および運用者の役割を担う技術者を支援することを目的としています。Red Hat OpenShift のアーキテクチャ、管理ポータル、およびユーザーエクスペリエンスの多くは Azure Red Hat OpenShift と同一であり、このガイドは、読者が Red Hat OpenShift の基本的な知識を持っていることを前提としています。

本章では、簡単な教材として「Fruit Smoothies」アプリケーションという単純なサンプルアプリケーションを使用し、そのデプロイ方法について説明します。このアプリケーションは、Azure Red Hat OpenShift の使い方をより深く理解するための足掛かりや再確認のための資料として役立つでしょう。これは Azure Kubernetes Service 用に維持されているサンプルアプリケーションで、Azure Red Hat OpenShift 上にデプロイすることで、OpenShift が Kubernetes と 100% 互換性があることが証明されます。

本章は主に http://aroworkshop.io の内容に基づいており、一部、説明と文脈が追加されています。

評価アプリケーションの概要

アプリケーションのアーキテクチャはシンプルで、以下のとおりです。

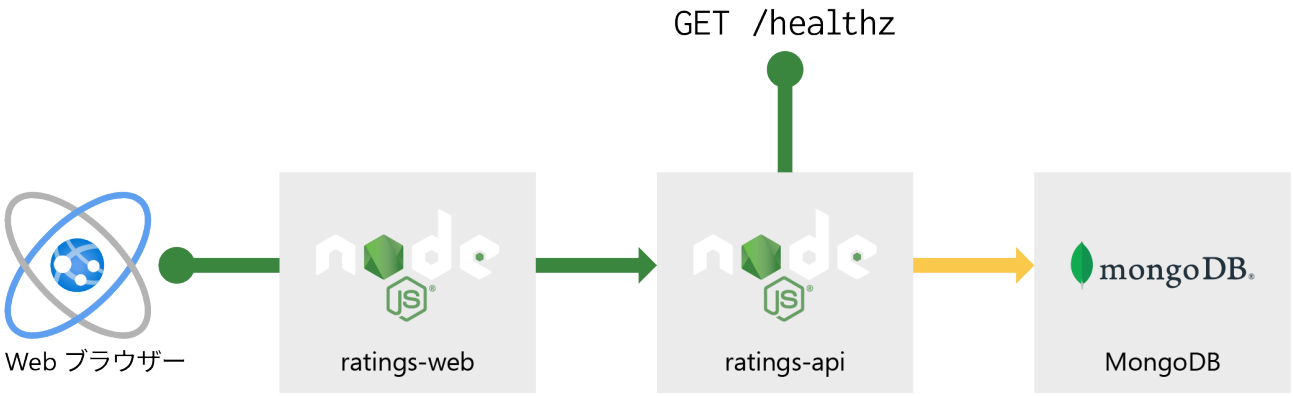


図 7.1：Fruit Smoothies アプリケーションのアーキテクチャ

前の図では、次の表に示すように、アプリケーションが 3 つのサービスと 2 つのパブリックエンドポイントで構成されていることがわかります。図の左側の最初のエンドポイントは、パブリック HTML Web アプリケーションを表示しているユーザーの Web ブラウザーを表しています。2 番目のエンドポイントである healthz は、ratings-api サービスのヘルスチェックです。個々のサービスの説明とそれらのリポジトリへのリンクは、次の表に記載されています。

コンポーネント	説明	GitHub リポジトリ
rating-web	公開されている Web フロントエンド、つまり「Web サイト」。	GitHub リポジトリ
rating-api	このサービスは、Web UI から入力を受け取り、それをデータベースに保存します。また、データベースからの結果をポート 3000 で Web アプリケーションに返します。	GitHub リポジトリ
mongodb	プリロードされたデータを含む NoSQL データベース。	データ

GitHub リポジトリのリンクにアクセスすると、これらのアプリケーションの詳細を確認できます。以下に続くガイダンスでは、これらの各アプリケーションのデプロイ方法を順を追って説明します。

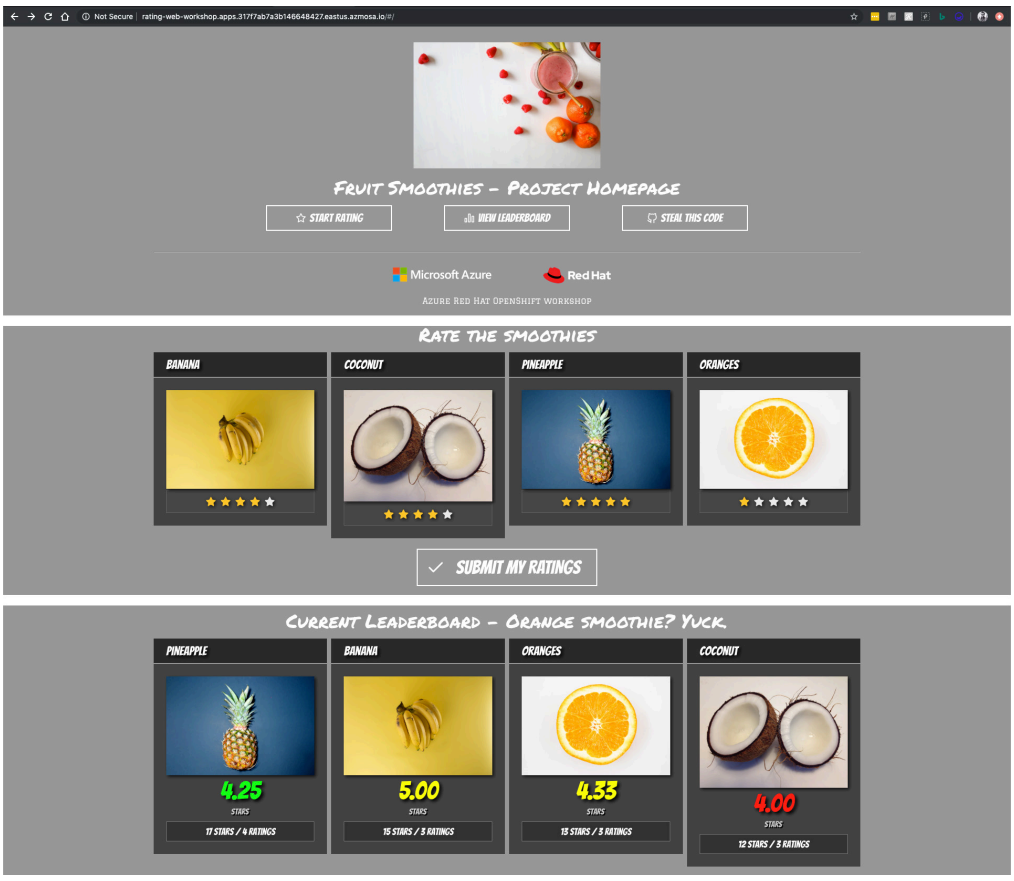


図 7.2：Fruit Smoothies アプリケーションの概要を示すスクリーンショット

完了したら、上記のスクリーンショットのような Web アプリケーションができています。開発者チームと運用チームがアプリケーションをどのようにして Azure Red Hat OpenShift にデプロイするのかについても、理解が深まっているはずです。これは、お客様ご自身がアプリケーションを Azure Red Hat OpenShift にデプロイする際の意思決定プロセスに役立ちます。

クラスタの作成と接続

以後、本章では、作業用の Azure Red Hat OpenShift 環境が稼働していることを前提とします。この評価アプリケーションに特別な要件はありません。また、これは空白で新規の Azure Red Hat OpenShift 環境にデプロイされます。クラスタのプロビジョニングをまだ行っていない場合は、以下の章を再確認してください。

- 第 4 章：プロビジョニングの準備 – エンタープライズ・アーキテクチャに関する考慮事項
- 第 5 章：Azure Red Hat OpenShift クラスタのプロビジョニング

第 5 章：Azure Red Hat OpenShift クラスタのプロビジョニングのクラスタへのアクセス のセクションは、すでにプロビジョニングが完了している可能性があるクラスタへのアクセス方法を再確認するのに役立ちます。

Web コンソールへのサインイン

各 Azure Red Hat OpenShift クラスタには、OpenShift Web コンソール用の DNS アドレスがあります。コマンド `az aro list` で、現在の Azure サブスクリプションのクラスタのリストを表示できます。

```
az aro list -o table
```

クラスタ Web コンソールの URL が表示されます。そのリンクを新しいブラウザータブで開き、`kubeadmin` ユーザー、またはプロジェクト作成のパーミッションを持つ別のユーザーアカウントでサインインします。

ログインすると、Azure Red Hat OpenShift Web コンソールが表示されます。

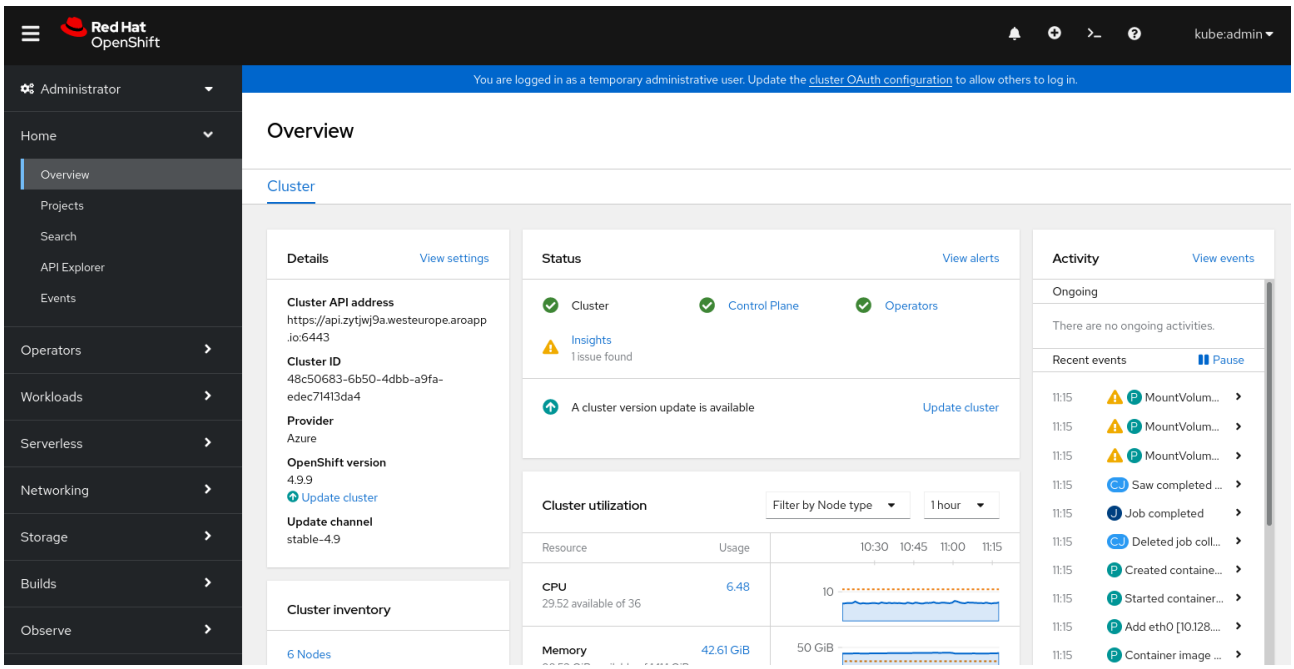


図 7.3：Azure Red Hat OpenShift Web コンソール

OpenShift クライアントのインストール

Azure Cloud Shell を開くか、ローカルの Linux Terminal を使用して、OpenShift コマンドライン・クライアントをインストールします。これは、コマンドラインからクラスタにアクセスするために必要です。これを行う手順は次のとおりです。

```
cd ~
curl https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-linux.tar.gz > openshift-client-linux.tar.gz

mkdir openshift

tar -zxvf openshift-client-linux.tar.gz -C openshift

echo 'export PATH=$PATH:~/openshift' >> ~/.bashrc && source ~/.bashrc
```

Windows または Mac の場合は、以下のリンクからダウンロードできます。

- https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-windows.zip
- https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-mac.tar.gz

ダウンロードしたパッケージで oc コマンドを実行できます。

ログインコマンドとトークンの取得

クライアントをインストールしたら、クラスタにログインするためのトークンを取得する必要があります。OpenShift Web コンソールにログインし、右上のユーザー名をクリックして **Copy Login Command** をクリックします。

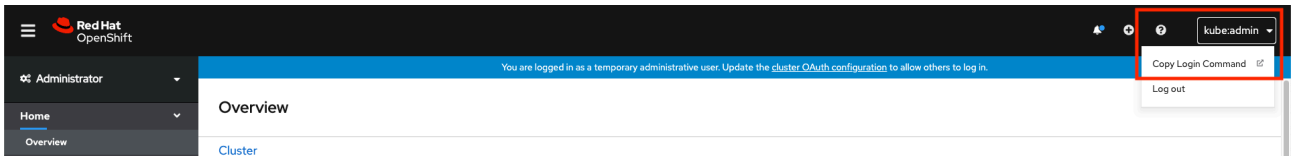


図 7.4：クラスタにログインするための Copy Login Command

ログインコマンドをシェル (ローカルの Linux Terminal または Azure Cloud Shell) に貼り付けます。これでクラスタに接続できます。

図 7.5：ログインコマンドを使用してクラスタに接続

接続したら、プロジェクトの作成に進むことができます。

プロジェクトの作成

OpenShift のプロジェクトは、評価アプリケーションを格納する論理フォルダーのようなものです。Red Hat OpenShift では、すべてのコンテナとアプリケーションが何らかのプロジェクト内に存在する必要があります。プロジェクトを使用して、アプリケーション、さらには部門を分離できます。次に、プロジェクトの作成方法をいくつか説明します。

Web インタフェースからプロジェクトを作成できますが、ここでは次のコマンドラインを使用します。

```
oc new-project workshop
```

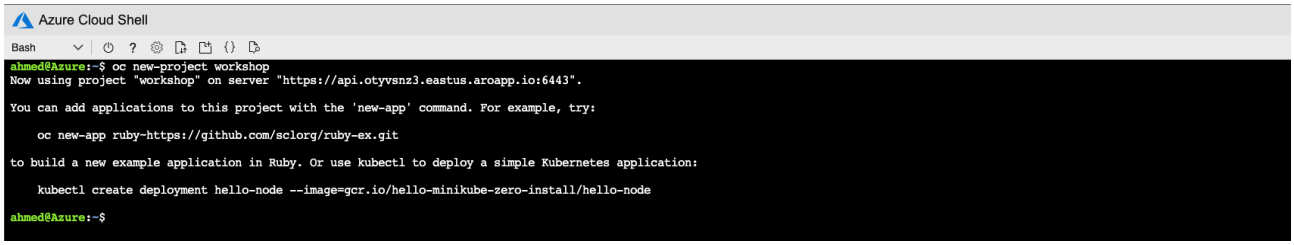


図 7.6：Azure Cloud Shell で新しい workshop を作成

プロジェクトが作成されたら、`oc project workshop` を使用してプロジェクトに切り替えることができます。次のステップは、本章の冒頭で説明した 3 つのマイクロサービスの 1 番目である MongoDB のデプロイです。先ほど作成したプロジェクトにデプロイします。

リソース

- Azure Red Hat OpenShift ドキュメント – CLI の使用を開始する
- Azure Red Hat OpenShift ドキュメント – プロジェクト

MongoDB のデプロイ

Azure Red Hat OpenShift には、MongoDB データベースサービスの新規作成を容易にするコンテナイメージとテンプレートが含まれています。テンプレートには、必須の環境変数をすべて定義するパラメーターフィールドがあり (ユーザー、パスワード、データベース名など)、自動生成されたパスワード値など、事前定義済みのデフォルト値が設定されます。また、テンプレートは、デプロイメント設定およびサービスの両方を定義します。

MongoDB インスタンスは、Docker Hub からコンテナイメージとして直接デプロイされます。OpenShift では、これをすべて Web コンソールを介して行うことができます。メニュー上部で開発者パースペクティブに切り替え、**Add** ページに移動して、**Container images** を選択します。

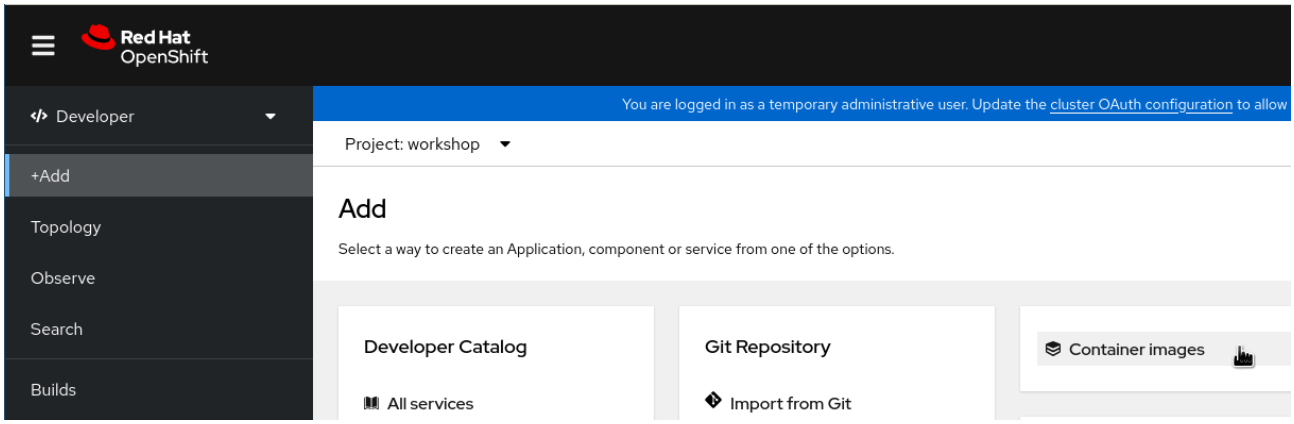


図 7.7 : コンテナイメージの追加

これにより、**Deploy Image** ページが表示されます。フォームに次のように入力します。

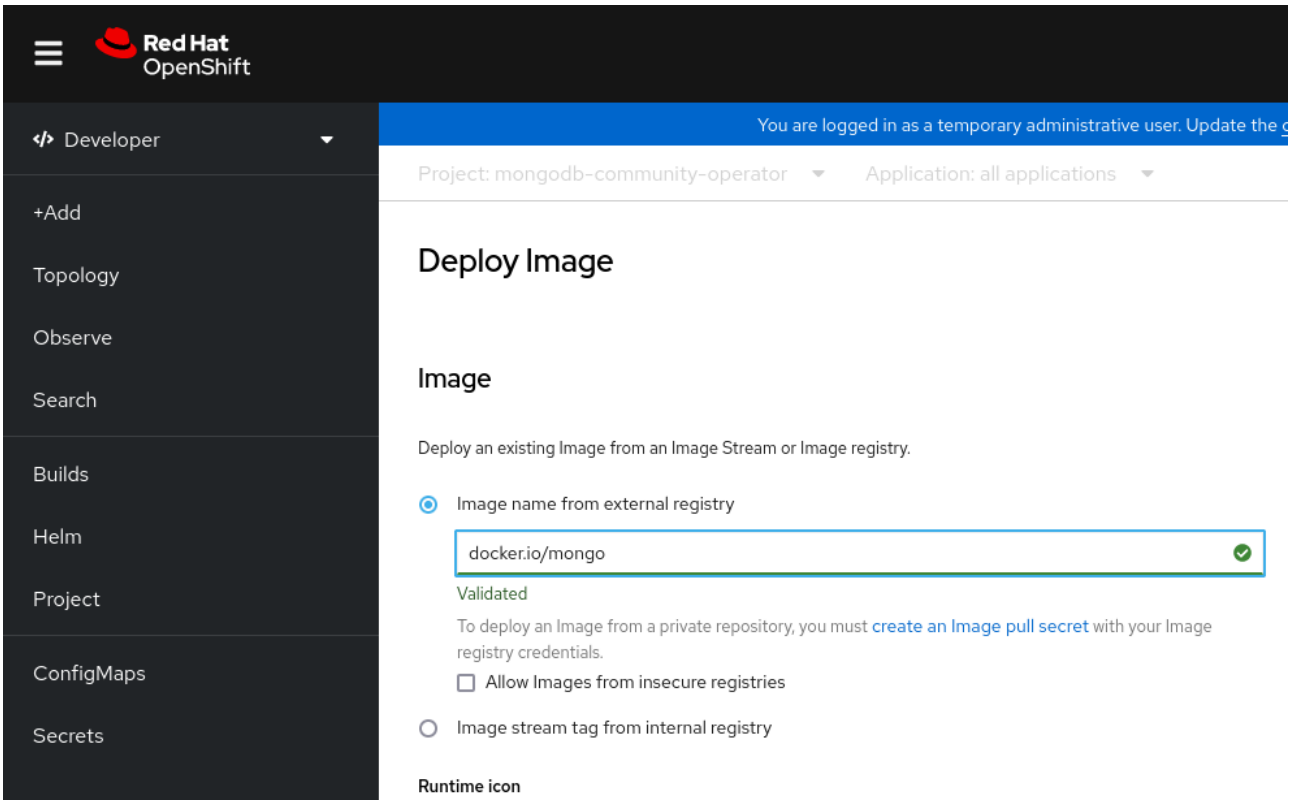


図 7.8：フォームに入力してイメージをデプロイ

フォームの値は次のように設定してください。

フィールド	値
Image name from external registry	`docker.io/mongo`
Runtime icon	`mongodb`
Application name	`ratings`
Name	`mongodb`
Create route to application	チェックしない — ルートを作成するとデータベースへの外部アクセスが許可されますが、このアプリケーションには必要ありません
Resource type	Deployment

フォームの最下部まで来たら、Deployment リンクを選択してフォームを展開し、環境変数を設定できるようにします。

次の環境変数は、データベースの最初の起動時にデータベースを初期化するためのデフォルトとして機能します。

名前	値
MONGO_INITDB_DATABASE	ratingsdb

MongoDB データベースにはユーザー名とパスワードが設定されていません。これはデフォルトの構成であり、認証は無効になります。

環境変数に間違いがないことを確認したら **Create** ボタンをクリックして続行し、MongoDB コンテナをデプロイします。

しばらくすると、MongoDB インスタンスが workspace プロジェクトで稼働します。このデプロイメントを表示するには、**Topology** ビューに切り替えます。

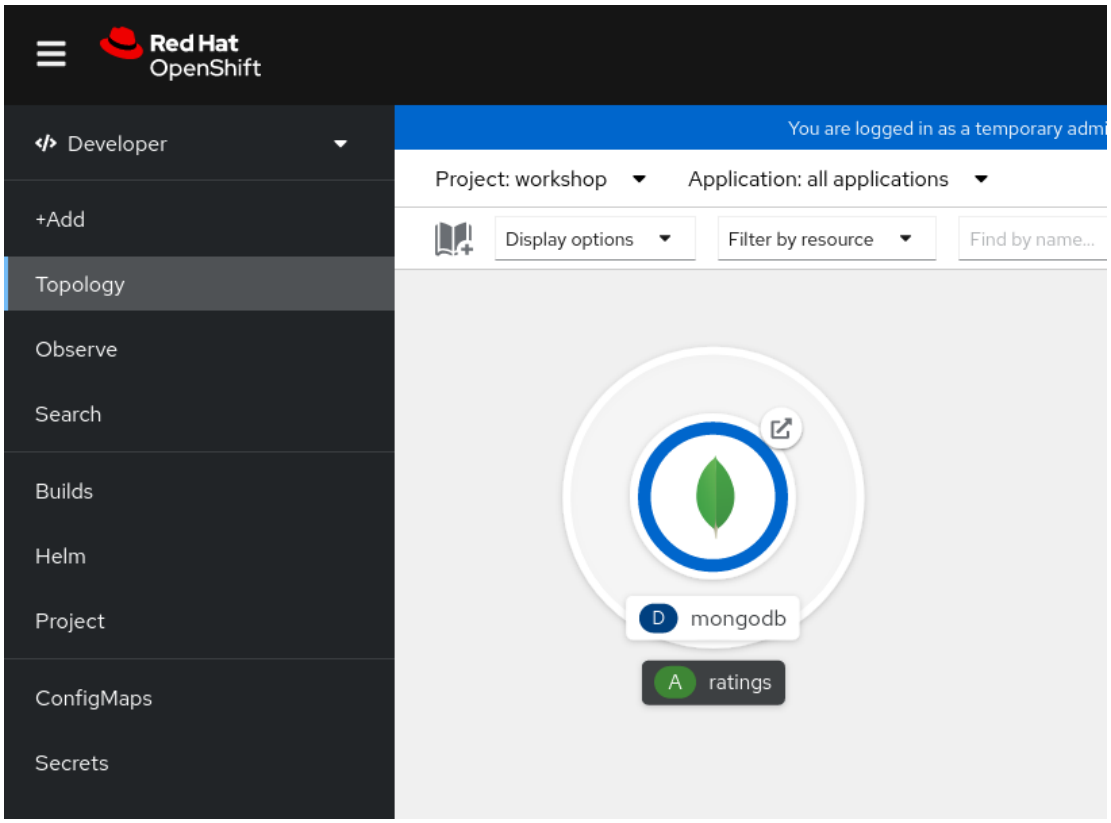


図 7.9：MongoDB インスタンスが稼働

oc get all コマンドを実行して、新しいアプリケーションのステータスを表示し、MongoDB テンプレートのデプロイメントが成功したかどうかを確認します。oc get all の出力例は次のとおりです。

```
user@host: oc get all
NAME                              READY    STATUS     RESTARTS    AGE
pod/mongo-6c6fcb45b8-8wpdm    1/1      Running    0           29s

NAME              TYPE          CLUSTER-IP       EXTERNAL-IP    PORT(S)      AGE
service/mongo     ClusterIP     172.30.88.119    <none>         27017/TCP    30s

NAME                       READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/mongo      1/1      1             1            30s

NAME                                    DESIRED    CURRENT    READY    AGE
replicaset.apps/mongo-6c6fcb45b8    1          1          1        30s

NAME                                        IMAGE REPOSITORY
TAGS      UPDATED
imagestream.image.openshift.io/mongo    image-registry.openshift-image-registry.svc:5000/workshop/mongo
latest    30 seconds ago
```

すべてが正常に機能している場合、STATUS 列にはコンテナが実行中 (Running) であることが表示されます。

MongoDB サービスのホスト名を取得する

デプロイが完了したら、クラスタ内からデータベースにアクセスできるように作成したサービスを見つける必要があります。svc は services の略です。

```
user@host: oc get svc mongodb
NAME     TYPE          CLUSTER-IP       EXTERNAL-IP    PORT(S)      AGE
mongo    ClusterIP     172.30.88.119    <none>         27017/TCP    77s
```

このサービスには、mongodb.workshop.svc.cluster.local という DNS 名でアクセスできます。DNS 名は [サービス名].[プロジェクト名].svc.cluster.local で構成されています。これはクラスタ内でのみ解決されます。

評価 API のデプロイ

次に、2 番目のアプリケーションである rating-api をデプロイします。これは、MongoDB インスタンスに接続してアイテムを取得および評価する Node.js アプリケーションです。このアプリケーションをデプロイするために必要な詳細の一部を以下に示します。

- GitHub の rating-api
- コンテナはポート 3000 を公開
- MongoDB 接続は、MONGODB_URI という環境変数を使用して設定される

以降のセクションで必要になるため、これらの詳細をメモしておいてください。

アプリケーションを独自の GitHub リポジトリにフォークする

CI/CD Webhook の追加などの変更を行えるようにするには、ratings-api コードの独自のコピーが必要です。Git では、これは「フォーク」と呼ばれます。アプリケーションを個人の GitHub リポジトリにフォークすることができます。前の GitHub リポジトリに移動し、**Fork** ボタンをクリックします。

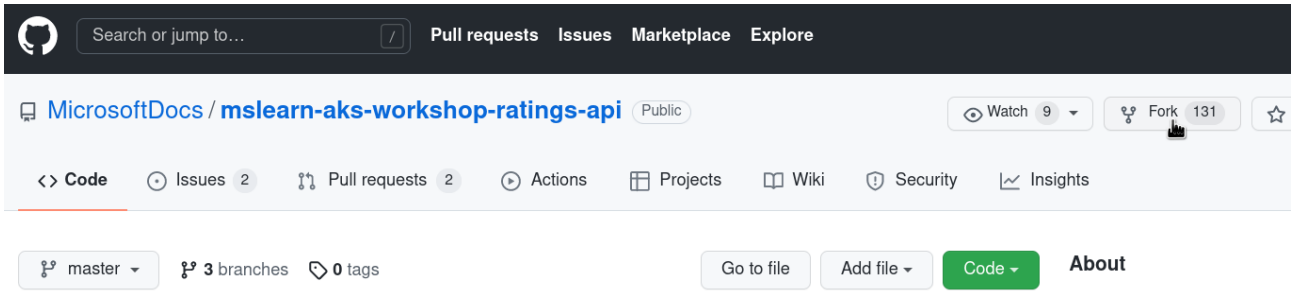


図 7.10：GitHub リポジトリでアプリケーションをフォーク

新しいリポジトリアドレスをメモします。これは、以降の手順で使用します。

OpenShift CLI を使用して rating-api をデプロイする

OpenShift は、コンテンツを確認し、そのコンテンツ (Java、PHP、Perl、Python など) に基づいて「ビルダーイメージ」を選択することで、Git リポジトリから直接コードをデプロイできます。この場合、rating-api は JavaScript アプリケーションです。ビルダーイメージは npm を使用して JavaScript 依存関係をダウンロードし、その結果、新しいコンテナイメージとして内部の OpenShift コンテナレジストリに保存されます。このビルドストラテジーは **Source-to-Image (S2I)** と呼ばれます。用語集で詳しく説明しています。

oc new-app を使用して、新規の S2I ビルドを開始できます。

```
user@host: oc new-app https://github.com/<your GitHub username>/mslearn-aks-workshop-ratings-api
--strategy=source --name=rating-api

--> Found image 0aea15f (3 weeks old) in image stream "openshift/nodejs" under tag "14-ubi8" for "nodejs"

    Node.js 14
    ----------
    Node.js 14 available as container is a base platform for building and running various Node.
js 14 applications and frameworks. Node.js is a platform built on Chrome's JavaScript runtime
for easily building fast, scalable network applications. Node.js uses an event-driven, non-
blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time
applications that run across distributed devices.

    Tags: builder, nodejs, nodejs14

    * The source repository appears to match: nodejs
    * A source build using source code from https://github.com/MicrosoftDocs/mslearn-aks-
workshop-ratings-api will be created
      * The resulting image will be pushed to image stream tag "rating-api:latest"
      * Use 'oc start-build' to trigger a new build
```

Web コンソール内の **Topology** ビューに切り替えるとアプリケーションのビルドが開始され、数分でデプロイメントが正常に完了します。

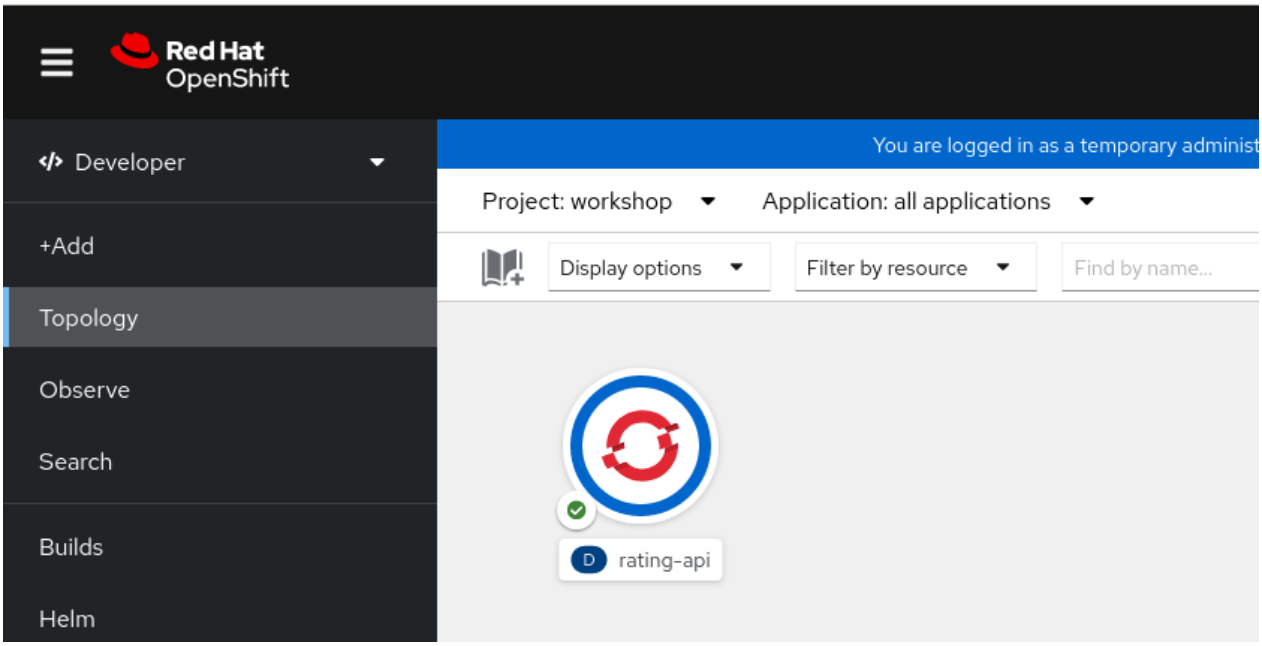


図 7.11：Topology ビュー

Pod のビルドと起動にかかる時間は、わずか 1 - 2 分です。

必要な環境変数を設定する

ここまでで、データベースと評価 API をデプロイできました。続けて、データベースへの接続方法を評価 API に指示する必要があります。コンテナベースのアプリケーションの構成は、通常、環境変数を使用して行います。

デプロイメントをクリックして編集します。次の環境変数を作成します。

名前	値
MONGODB_URI	mongodb://mongodb.workshop.svc.cluster.local:27017/ratingsdb

この新しい環境変数を保存すると、新しい環境変数を使用する ratings-api サービスの再デプロイがトリガーされます。

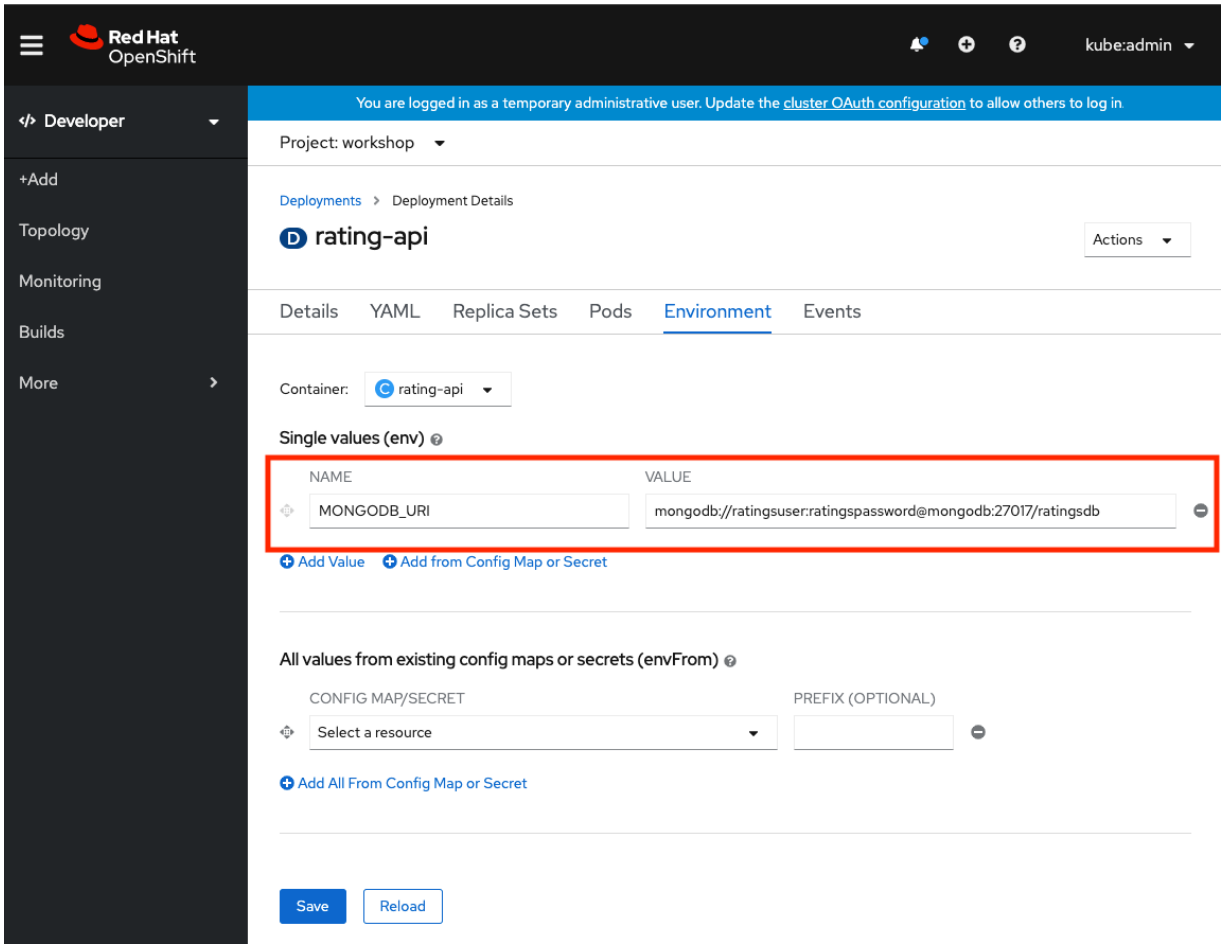


図 7.12：Web コンソールを介した MONGODB_URI 環境変数の設定

次のように、コマンドラインで実行することもできます。

```
oc set env deploy/rating-api MONGODB_URI=mongodb://mongodb.workshop.svc.cluster.local:27017/ratingsdb
```

どちらの方法を使用する場合でも、OpenShift は新しい環境変数を利用するコンテナを再起動する必要があります。

サービスが実行されていることを確認

`rating-api` デプロイメントのログに移動すると、コードが MongoDB に正常に接続できることを確認するログメッセージが表示されます。これを行うには、デプロイメントの詳細画面で **Pods** タブをクリックし、次に Pod のうちの 1 つをクリックします。

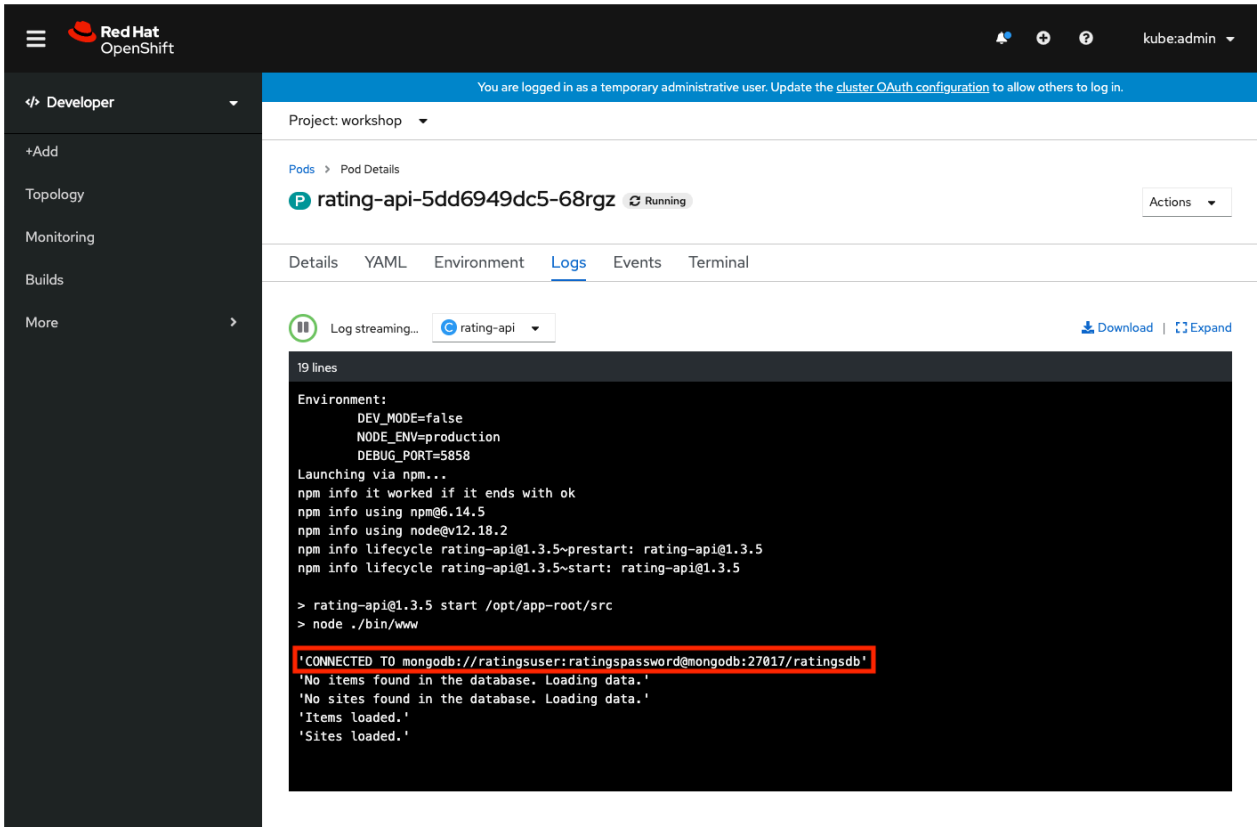


図 7.13：コードが MongoDB に正常に接続できることを確認するログメッセージ

rating-api サービスポートの調整

OpenShift は、ポート `8080` を使用してサービスを作成します。ただし、ライブラリの更新のため、このサービスはポート `3000` で実行されます。デフォルトのサービスを編集する必要があります。

Networking → **Services** メニューに移動し、サービスのリストから次のように rating-api サービスを編集します (8080 を 3000 に修正するだけです)。

```
  ports:
    - name: 3000-tcp
      protocol: TCP
      port: 3000
      targetPort: 3000
```

新しいポートを使用するには、サービスを再起動します。

```
user@host: oc rollout restart deploy/rating-api
```

これで、サービスは正しいポート (8080 ではなく 3000) にバインドされます。

rating-api サービスのホスト名の取得

次のセクションで rating-web アプリケーションをデプロイするときに使用するため、 rating-api サービスが存在することを検証する必要があります。

```
oc get service rating-api
```

このサービスには、ポート 3000 を介して rating-api.workshop.svc.cluster.local:3000 という DNS 名でアクセスできます。DNS 名は [サービス名].[プロジェクト名].svc.cluster.local という構成になっています。これはクラスタ内でのみ解決されます。

評価フロントエンドのデプロイ

rating-web は、rating-api に接続する Node.js アプリケーションです。このアプリケーションをデプロイするために必要な詳細の一部を以下に示します。

- GitHub の rating-web
- コンテナはポート 8080 を公開
- Web アプリは API という名前の環境変数を介してプロキシを使用し、内部クラスタ DNS を介して API に接続

OpenShift CLI を使用して rating-web をデプロイする

rating-api アプリケーションと同様に、このアプリケーションは oc new-app を使用して S2I でデプロイできます。ただし、今回は Git リポジトリに既に存在する Dockerfile を使用して、Dockerfile ストラテジーでアプリケーションを作成します。したがって、--strategy 引数は指定しないでください。

```
user@host: oc new-app https://github.com/<your GitHub username>/mslearn-aks-workshop-ratings-web
--name rating-web

--> Found container image e1495e4 (2 years old) from Docker Hub for "node:13.5-alpine"

    * An image stream tag will be created as "node:13.5-alpine" that will track the source image
    * A Docker build using source code from https://github.com/MicrosoftDocs/mslearn-aks-workshop-
ratings-web will be created
      * The resulting image will be pushed to image stream tag "rating-web:latest"
      * Every time "node:13.5-alpine" changes a new build will be triggered

--> Creating resources ...
    imagestream.image.openshift.io "node" created
    imagestream.image.openshift.io "rating-web" created
    buildconfig.build.openshift.io "rating-web" created
    deployment.apps "rating-web" created
    service "rating-web" created
--> Success
```

依存関係をコンテナにビルドするには少し時間がかかります。ビルドが完了してから **Topology** ビューに戻り、サービスがオンラインになるのを確認できるまで、2 分から 3 分程度かかります。

必要な環境変数を設定する

rating-web デプロイメント構成の API 環境変数を作成します。この変数の値は、rating-api サービスのホスト名/ポートになります。

環境変数は Azure Red Hat OpenShift Web コンソールではなく OpenShift CLI で設定することもできます。

```
oc set env deploy rating-web API=http://rating-api:3000
```

ルートを使用して rating-web サービスを公開する

サービスを公開するということは、ユーザーがサービスにアクセスするために使用できる、パブリックにアクセス可能な URL があるということです。サービスが公開されていない場合は、クラスタ内でのみアクセスできます。

```
user@host: oc expose svc/rating-web
route.route.openshift.io/rating-web exposed
```

最後に、公開されたサービスのURLを取得します。

```
user@host: oc get route rating-web
NAME          HOST/PORT                                                PATH    SERVICES      PORT
TERMINATION   WILDCARD
rating-web    rating-web-workshop.apps.zytjwj9a.westeurope.aroapp.io          rating-web    8080-tcp
None
```

デフォルトでは、**完全修飾ドメイン名 (FQDN)** がアプリケーション名とプロジェクト名で構成されていることに注意してください。FQDN の残りの部分であるサブドメインは、Azure Red Hat OpenShift クラスタ固有のアプリのサブドメインです。

サービスを使ってみる

ブラウザーでホスト名を開きます。評価アプリケーションのページが表示されます。投票するなど、いろいろと試して、スコアボードを見てみましょう。

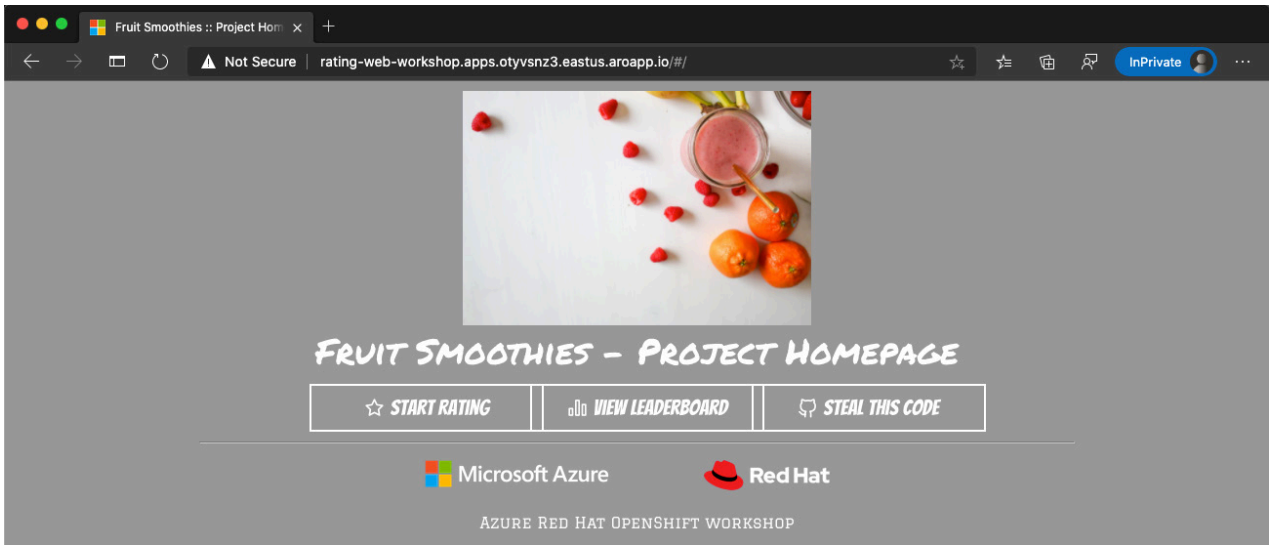


図 7.14：評価アプリケーションのサービスを使ってみる

GitHub Webhook の設定

コードを GitHub リポジトリにプッシュしたときに S2I ビルドをトリガーするには、GitHub Webhook を設定する必要があります。

1. GitHub Webhook トリガーシークレットを取得します。GitHub Webhook の URL でこのシークレットを使用する必要があります。

```
user@host: oc get bc/rating-web -o=jsonpath='{.spec.triggers..github.secret}'
3ffcc8d5-a243
```

シークレットキーは以降の作業で必要になるため、メモしておいてください。

2. ビルド構成から GitHub Webhook のトリガー URL を取得します。

```
user@host: oc describe bc/rating-web
...
Webhook GitHub:
        URL:    https://api.quwhfg7o.westeurope.aroapp.io:6443/apis/build.openshift.io/v1/
namespaces/workshop/buildconfigs/rating-web/webhooks/<secret>/github
...
```

3. <secret> プレースホルダーを、最初の手順で取得したシークレットに置き換えます。今回、シークレットは 3ffcc8d5-a243 なので URL は次のようになります。

```
https://api.quwhfg7o.westeurope.aroapp.io:6443/apis/build.openshift.io/v1/namespaces/workshop/
buildconfigs/rating-web/webhooks/3ffcc8d5-a243/github
```

この URL を使用して、GitHub リポジトリに Webhook を設定します。

4. GitHub リポジトリに移動します。**Settings** → **Webhooks** から **Add Webhook** を選択します。
5. **Payload URL** フィールドに、<secret> 部分にシークレットを挿入した GitHub URL をペーストします。
6. **Content type** フィールドで、デフォルトの application/x-www-form-urlencoded を application/json に変更します。

7. **Add webhook** をクリックします。

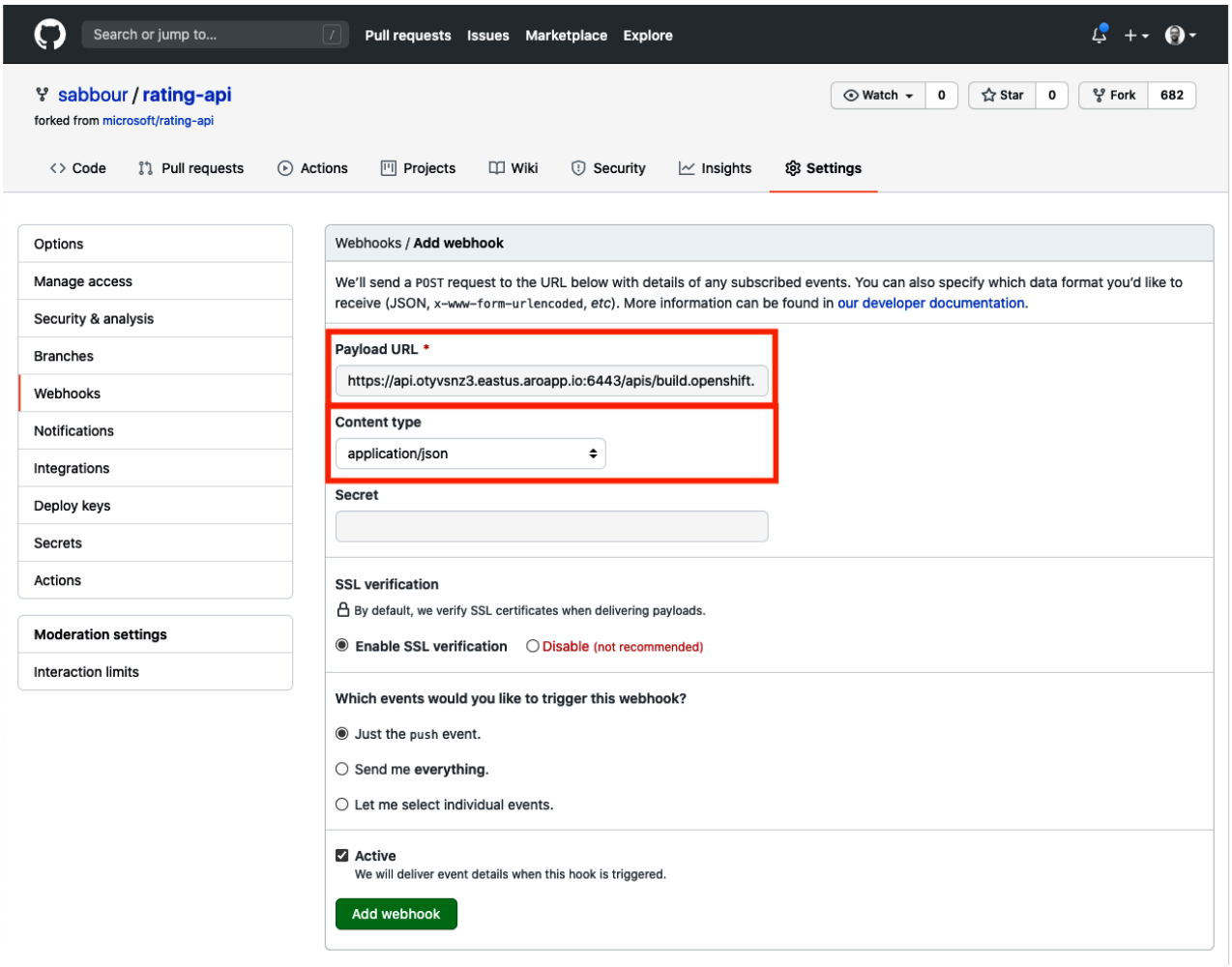


図 7.15：Webhook の追加

Webhook が正常に構成されたことを示す GitHub からのメッセージが表示されます。

これで、GitHub リポジトリに変更をプッシュするたびに、新規のビルドが自動的に開始され、ビルドが成功すると新しいデプロイが開始されます。

Web サイトのアプリケーションに変更を加えて、ローリングアップデートを確認する

GitHub のリポジトリにある https://github.com/<your GitHub username>/rating-web/blob/master/src/App.vue ファイルにアクセスします。

ファイルを編集します。background-color: #999; の行を background-color: #0071c5; に変更します。

ファイルへの変更を master ブランチにコミットします。

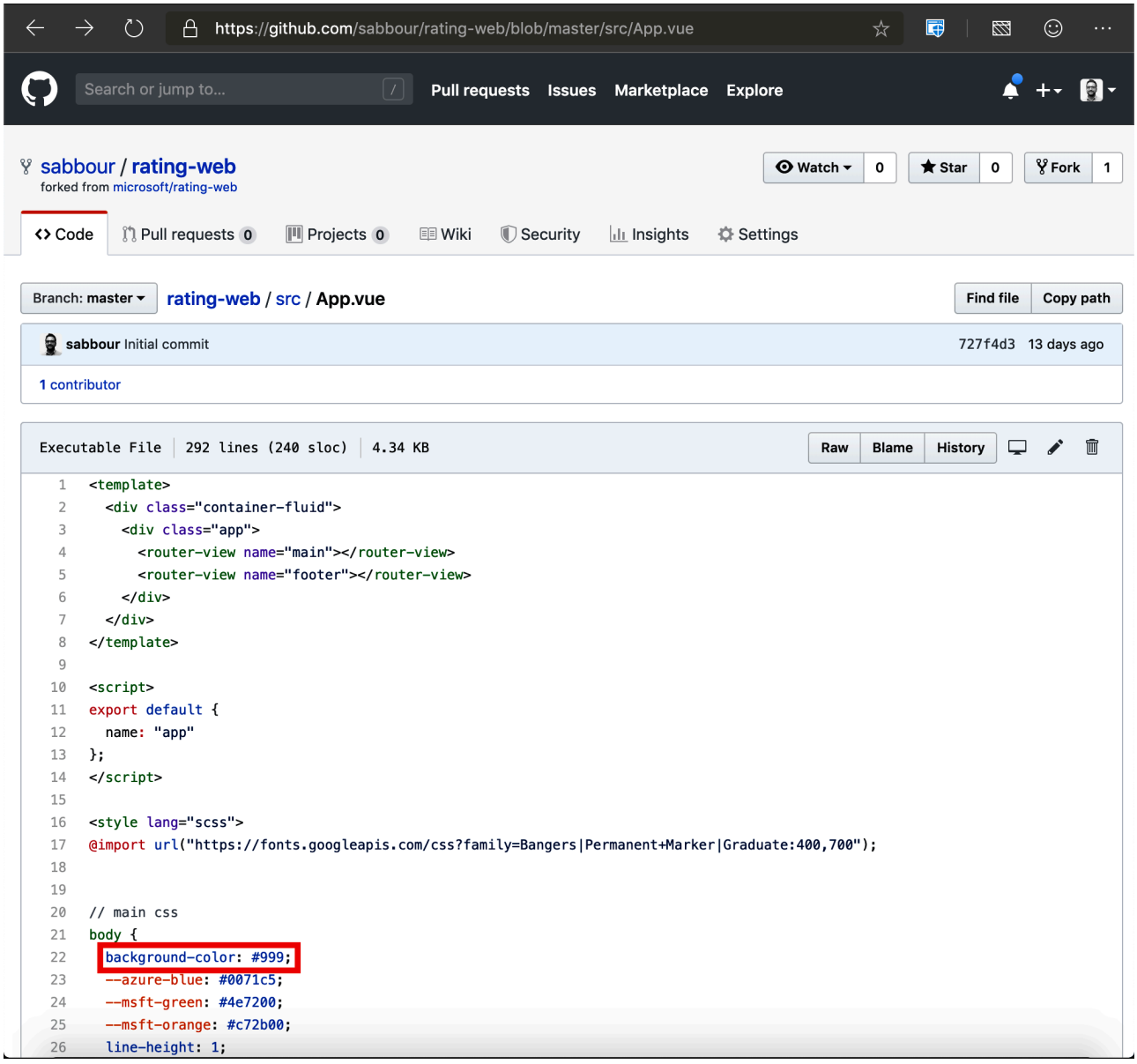


図 7.16：ファイルへの変更を master ブランチにコミット

すぐに、OpenShift Web コンソールの **Builds** タブに移動します。プッシュによってトリガーされた新しいビルドがキューに入っています。これが完了すると、新しいデプロイがトリガーされ、Web サイトの色が更新されていることを確認できます。

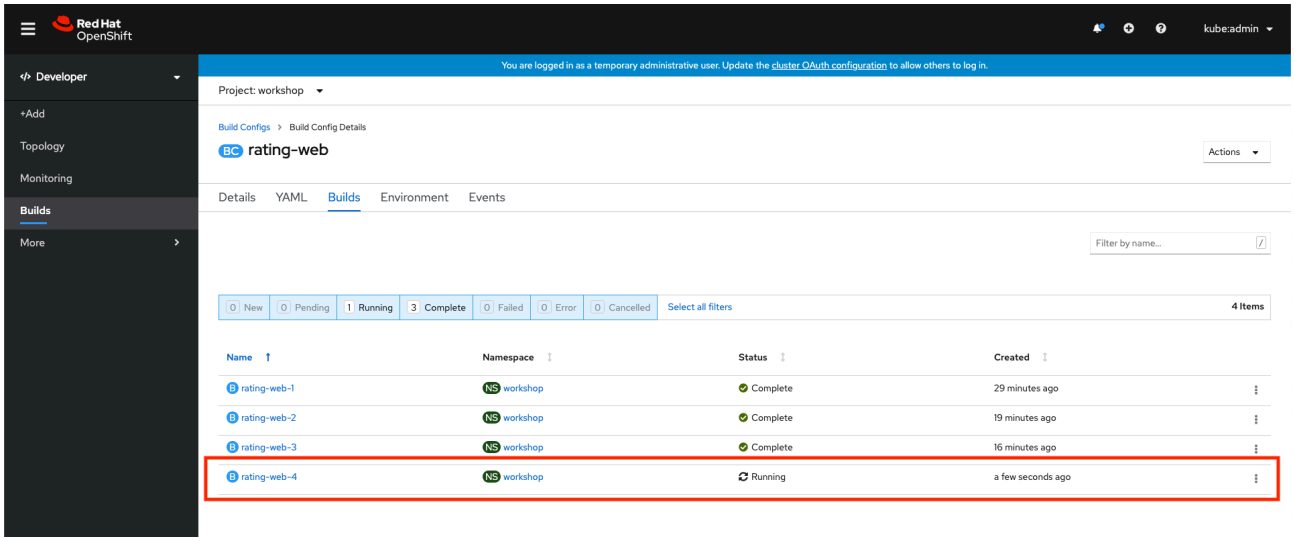


図 7.17：新しいビルドが実行中であることを示す Builds タブ

次に、ratings-web ページに戻ります。すべてうまく行っていれば背景色が変更されています。

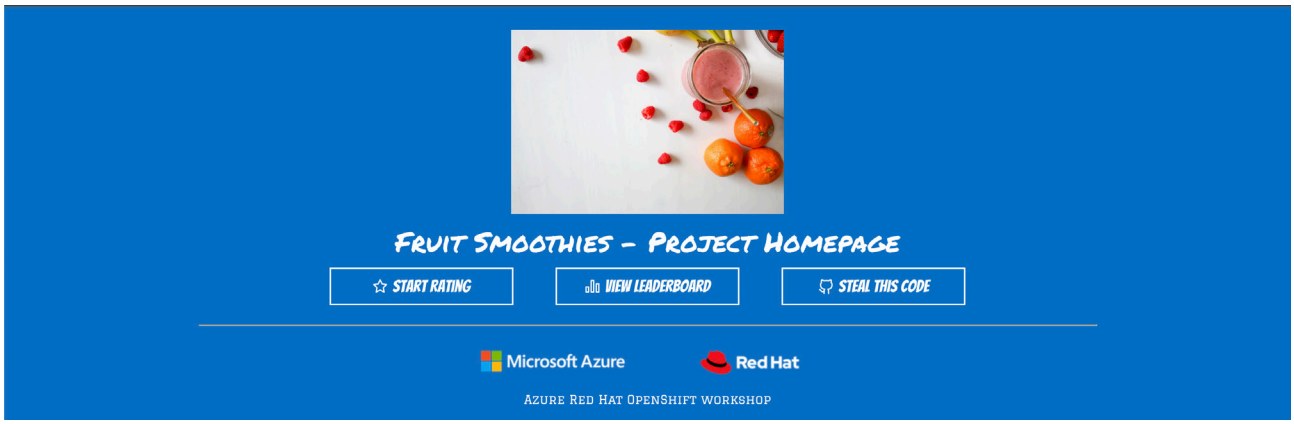


図 7.18：背景色が更新された Fruit Smoothies のホームページ

まとめ

本章では、MongoDB データベース、ratings-api、ratings-web コードの 3 つの小さなマイクロサービス・アプリケーションから成る基本的なアプリケーションをデプロイしました。これはプロダクションのアプリケーションとはかなり違うものではありますが、アプリケーションを Azure Red Hat OpenShift にデプロイする際の概念を簡単に振り返るのに役立ちます。独自のアプリケーションをデプロイする場合、この手順が基本になります。

Red Hat OpenShift は、OperatorHub、Helm Chart、および Azure DevOps のような外部 CI/CD システムからのアプリケーションのデプロイもサポートしています。組織に最適なデプロイメント・ストラテジーは、組織内で使用しているツールやテクノロジーによって異なります。

次の章では、Azure Red Hat OpenShift の付加価値、つまり Kubernetes 上に構築されたアプリケーション・プラットフォームとしての OpenShift の価値について説明します。エンタープライズ・アプリケーションの複雑なニーズをサポートすることを意図した特長や機能について掘り下げます。

第 8 章

アプリケーション・プラットフォームの機能紹介

前の章では、Red Hat OpenShift が Kubernetes 上に構築されたさまざまなサービスをどのように提供するかについて学びました。そうしたサービスはプラットフォームサービス、アプリケーションサービス、データサービス、開発者サービス、Kubernetes クラスタサービスの 5 つのカテゴリのいずれかに分類され、これらのサービスを組み合わせることで、真のアプリケーション・プラットフォームを作成できます。

*Red Hat OpenShift® には、一般的な言語/フレームワーク/データベースでサポートされるランタイムが含まれています。
その他の機能は、Red Hat Application および Data Services ポートフォリオに含まれています。

図 8.1：Azure Red Hat OpenShift に含まれるサービス

OpenShift Platform Plus に分類されるコンポーネント (Multicluster Management、Cluster Security、および Global Registry) は、サブスクリプションを必要とする付加的な製品です。これらは Azure Red Hat OpenShift と互換性がありますが、Azure Red Hat OpenShift オファリングには含まれていません。

次のセクションでは、これらのサービスが提供する主なメリットをいくつか取り上げ、詳細情報にアクセスできるリンクを紹介します。

クラスタサービス – コンテナレジストリの統合

Red Hat OpenShift は、クラスタがデプロイされると内部でコンテナレジストリを統合します。このレジストリは、クラスタ Operator のようなクラスタ内部のサービスに加え、デフォルトではお客様のアプリケーションコンテナにも使用されます。このレジストリは標準で、追加設定は必要なく、インフラストラクチャの Operator によって維持されます。

Kubernetes コンテナサービスをデプロイする場合、通常はコンテナイメージをプライベートにするために独自のコンテナレジストリもデプロイする必要があります。OpenShift では、組み込みのコンテナレジストリがクラスタ内ですでに利用可能になっているため、Day 2 の追加インストールおよびセットアップは必要ありません。これは、OpenShift が持つ、シンプルでありながら時間を節約する機能の一例です。

OpenShift Container Platform の統合レジストリの概要

クラスタ管理者はしばしば、OpenShift の外部のユーザーがコンテナイメージをレジストリにプッシュできるように、このコンテナレジストリをクラスタの外部に公開することを選択します。これは Azure Red Hat OpenShift 内で完全にサポートされています。その方法に関するドキュメントは、レジストリの公開に関する標準の OpenShift ドキュメントにあります。

プラットフォームサービス – OpenShift Pipelines

Red Hat OpenShift のお客様は、さまざまな方法でアプリケーションを構築することができます。Jenkins、CircleCI、GitHub Actions など、多くの一般的な CI/CD ツールは OpenShift をサポートするプラグインを備えています。しかし、OpenShift には、OpenShift Pipelines の Operator を介して、プラットフォーム上でクラウドネイティブのコンテナパイプラインを使用してアプリケーションを構築する機能もあります。

OpenShift Pipelines は Tekton というコミュニティプロジェクトに基づいています。Git リポジトリからのコードのプル、Java コンパイラーの実行、RPM パッケージのアセンブルなど、パイプラインの各プロセスはコンテナ内で実行されます。つまり、開発者と運用者は、コンテナが提供するすべての利点を活用して、洗練された複雑なパイプラインを形成し、ソフトウェアを構築することができます。

OpenShift Pipelines は、OperatorHub を介してインストールできます。**OperatorHub** に移動し、OpenShift Pipelines の Operator を選択するだけでインストールが開始されます。構成は必要なく、Operator のインストールは通常 1 分以内に完了します。

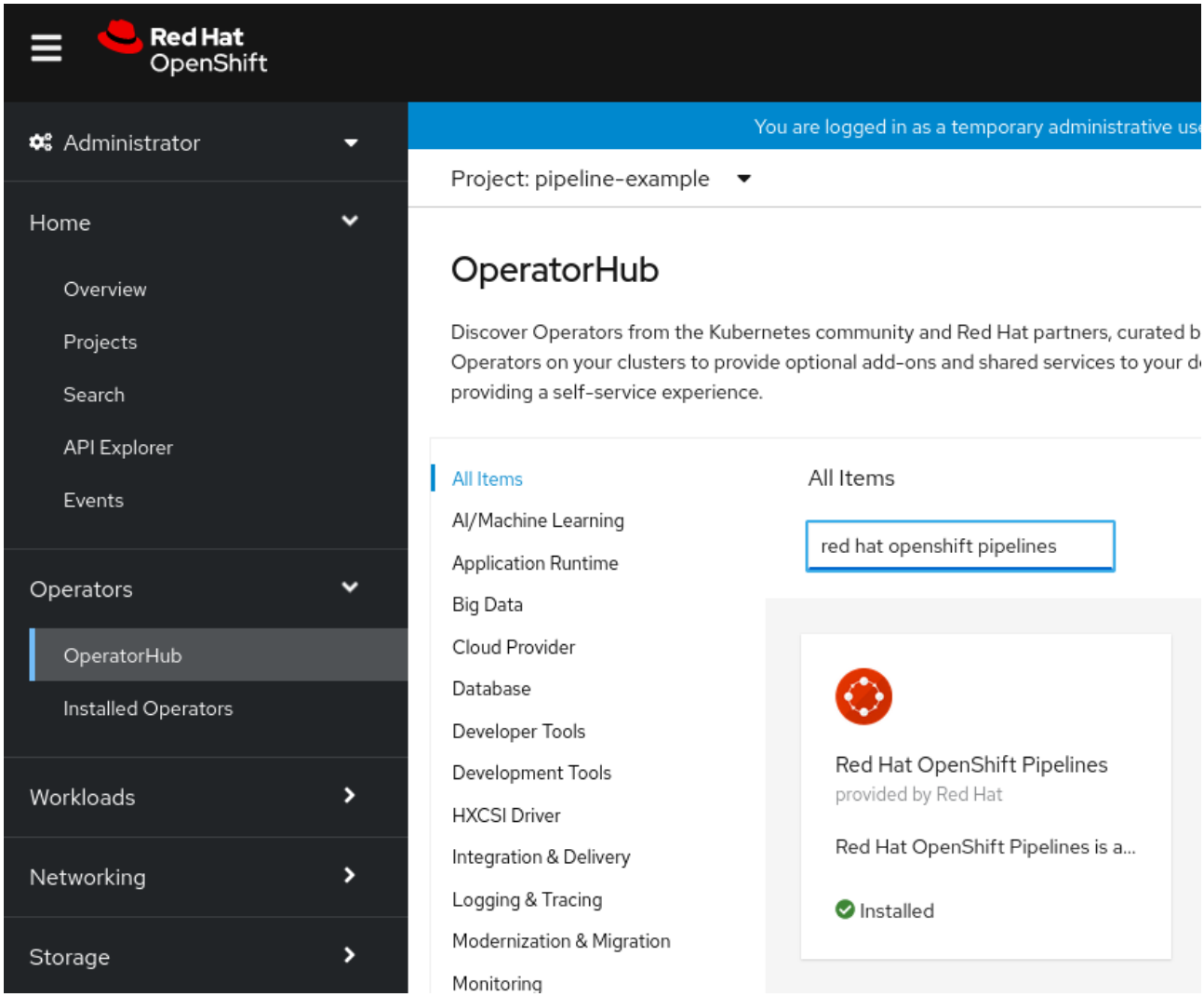


図 8.2：OperatorHub を介した OpenShift Pipelines のインストール

OpenShift Pipelines の Operator をインストールすると、サイドバーに新しく **Pipelines** セクションができ、Node.js の例を作成する場合などにパイプラインをカタログアイテムに追加するオプションが表示されます。

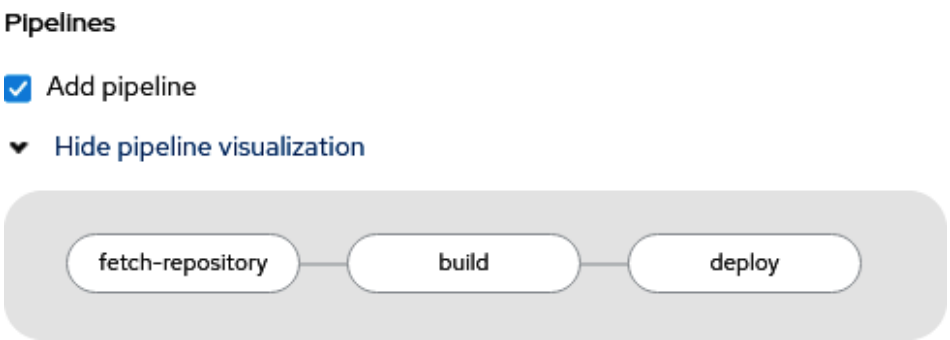


図 8.3：パイプラインの追加

OpenShift Pipelines を使用すると、ビジュアルビルダーを使って複雑な分岐パイプラインをセットアップしたり、構築したりすることもできます。例として、より複雑なパイプラインのスクリーンショットを次に示します。

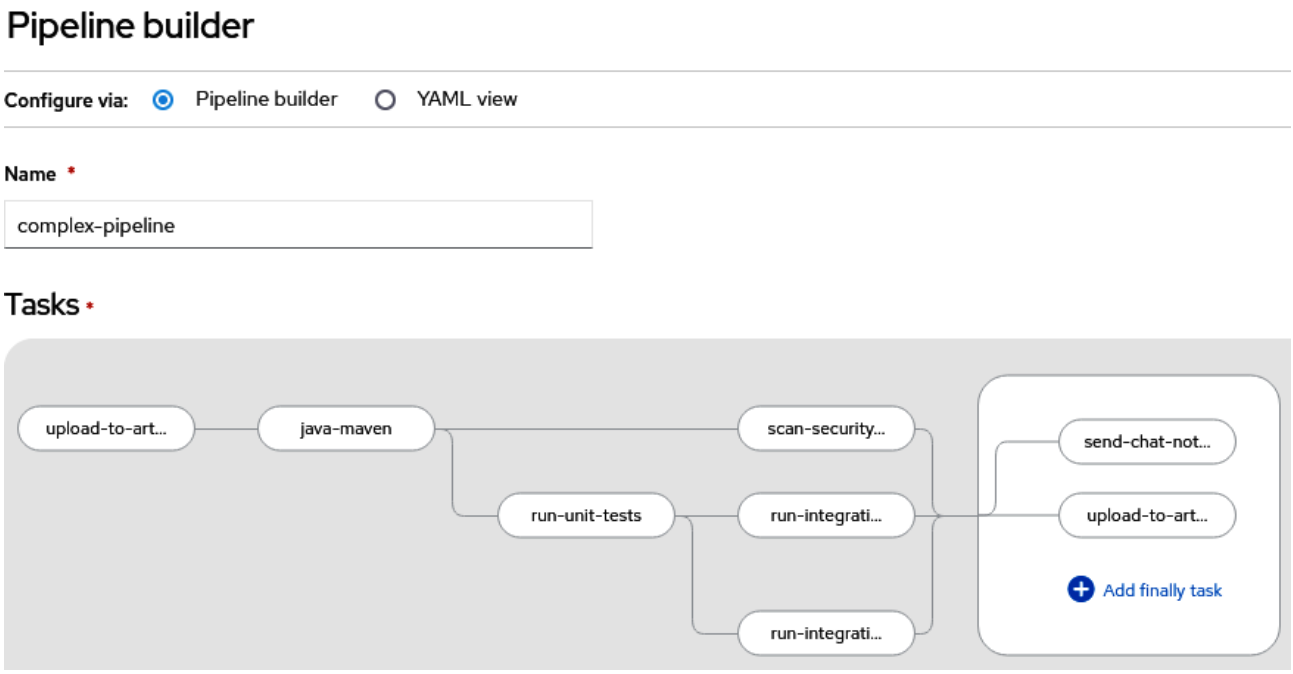


図 8.4：複雑なパイプライン

多くの組織が、さまざまなツールやテクノロジーを使用してソフトウェアを構築しますが、OpenShift Pipelines を使用すれば、コンテナが提供するあらゆる利点を活用し、OpenShift 内にビルドを直接、容易に統合できます。基盤のインフラストラクチャとして何を使用しているかに関係なく、セットアップの手間が極めて少なく、一貫性のある使いやすい CI/CD ソリューションを提供します。

参考資料

- OpenShift の OpenShift Pipelines について
- Tekton コミュニティサイト

プラットフォームサービス – OpenShift Serverless

コンテナは単に、長期的に実行するサービスに役立つテクノロジーであるというのはよくある誤解です。実際は、短期間で終了する多くのジョブやサーバーレス機能が同様に短期間で廃棄されるコンテナで実行されます。コンテナは、起動の速度、一貫性、シャットダウンの容易さの面で利点があるため、サーバーレスのワークロードにも適しています。もちろん、すべてのサーバーレスサービスにおいて、基盤となるサーバーがコードを実行する必要があります。このため、サーバーレスは「Function-as-a-Service (FaaS)」と呼ばれることもあります。

Red Hat OpenShift は、OpenShift Serverless の Operator を介して、サーバーレス (FaaS) のワークロードを処理します。この Operator は、Knative という人気のオープンソース・プロジェクトに基づいています。

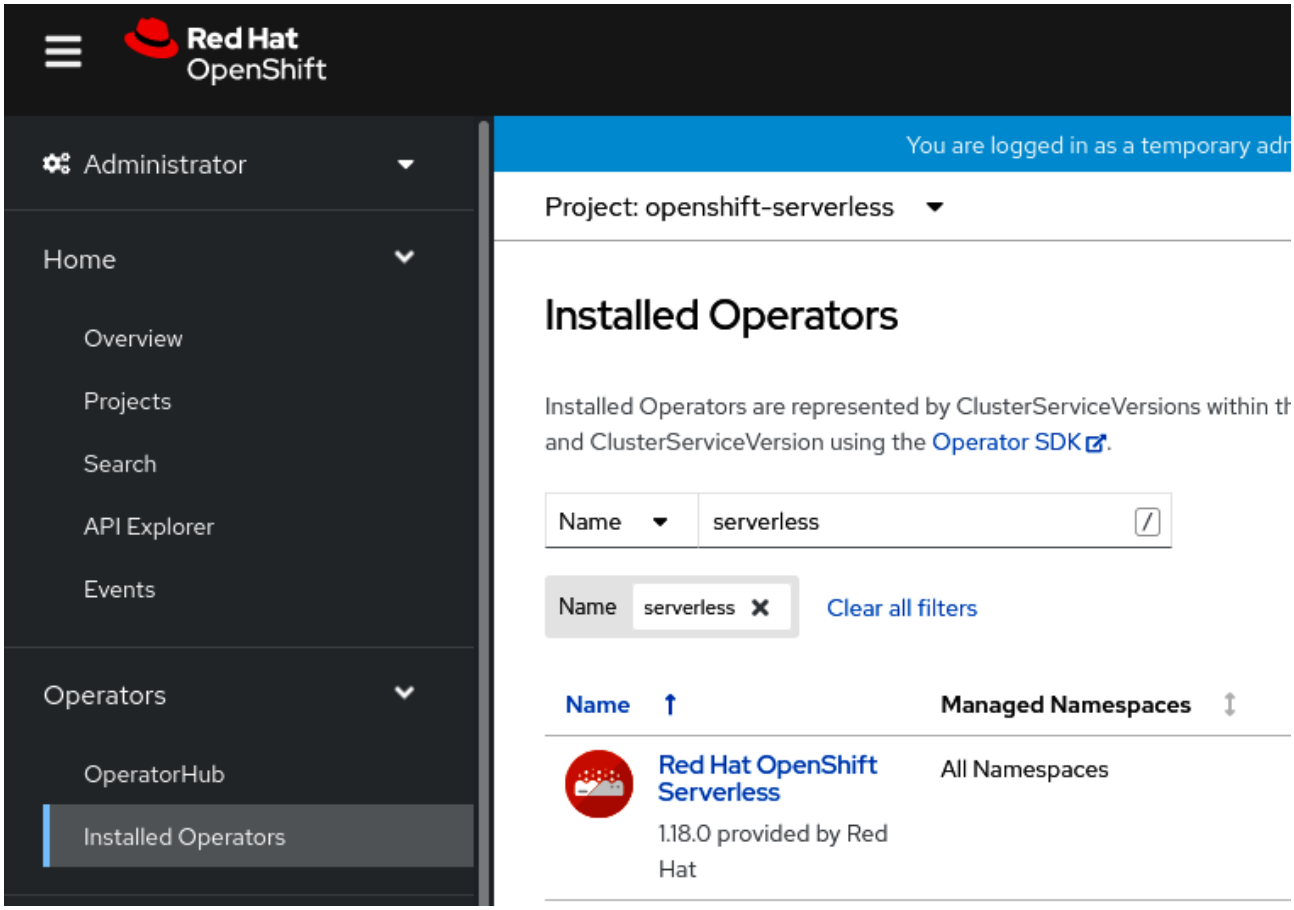


図 8.5：Installed Operators のビュー

クラスタにデプロイしたら (これも通常は 1 分から 2 分で終了します)、Operator 用のセットアップが少し必要になります。特に注意を払うべき**カスタムリソース定義 (CRD)** が 2 つあります。**Knative Serving** と **Knative Eventing** です。

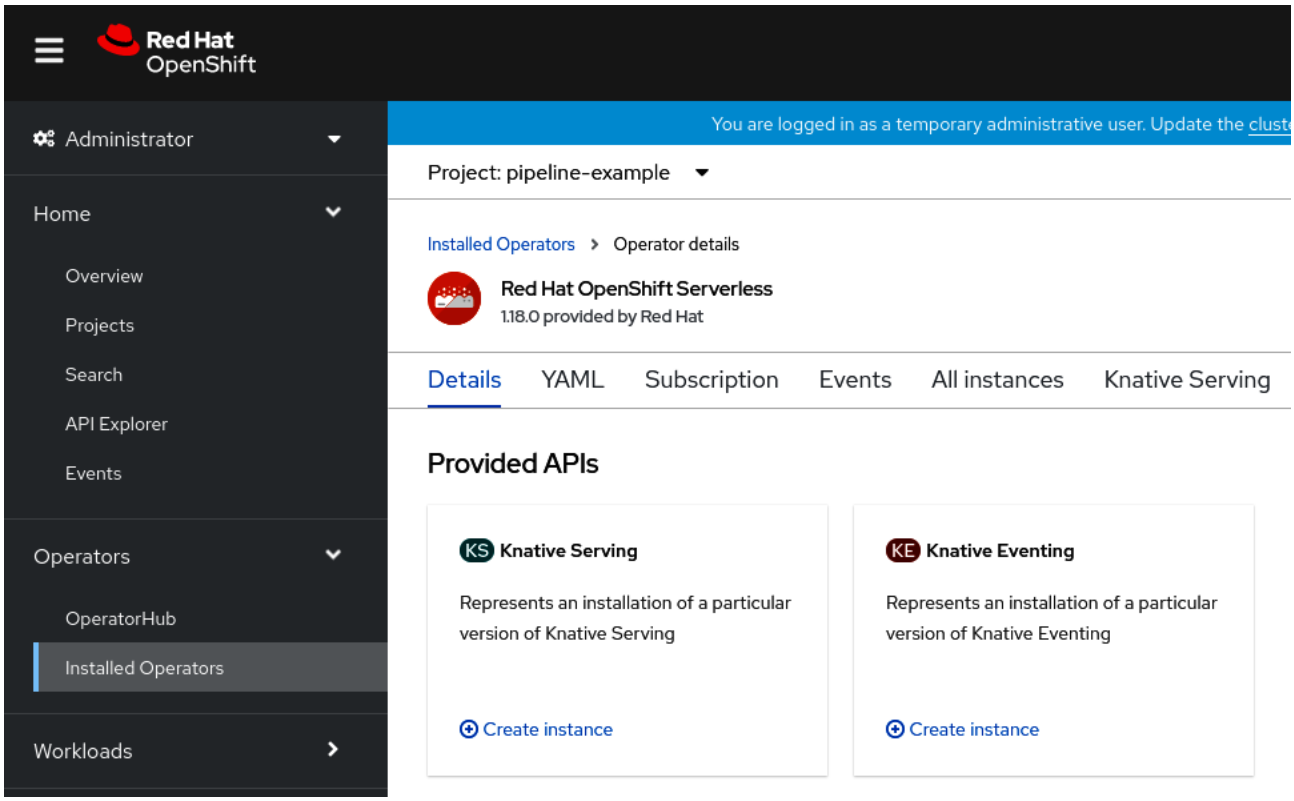


図 8.6：Operators メニューの Knative Serving と Knative Eventing

- **Knative Serving** は、アプリケーションのデプロイを単純化し、着信トラフィックに基づいて動的なスケーリングを行い、トラフィック分割によるカスタムのロールアウトストラテジーをサポートします。 例としては、イメージをストレージバケットにアップロードし、そのアップロードを NoSQL データベースにログとして記録する機能があります。
- **Knative Eventing** を使用すると、ビルド時ではなく、アプリケーションの実行時にイベントソースの「遅延バインディング」を実行できます。例としては、ストレージバケットへの新しいイメージのアップロードに応答するアプリケーションがあります。そのアプリケーションは、ビルド時またはイベントのデプロイ時にストレージバケットを認識している必要はありませんが、Knative Eventing によって、実行時にそのイベントソースをアプリケーションに「バインド」するのが容易になります。

次の 2 つのセクションで、これらの OpenShift 上のサーバーレス・アプリケーションの例を示します。

プラットフォームサービス – OpenShift Serverless – Knative Serving の例

Knative Serving はどのようにアプリケーションの自動スケーリングを行うのか、特に、要求がない場合にどのようにゼロへのスケーリングを行うのか、実際にやってみましょう。ここでは、ペットの ASCII 画像を Web ページとして表示する極めて単純なアプリケーションを使います。

- GitHub の php-ascii-pets リポジトリ

このリポジトリを Git から追加するのは簡単です。Red Hat OpenShift は、互換性のある PHP のビルダーイメージを自動的に検出します。

OpenShift は、このプロジェクトを自動的にビルドする方法を検出します。

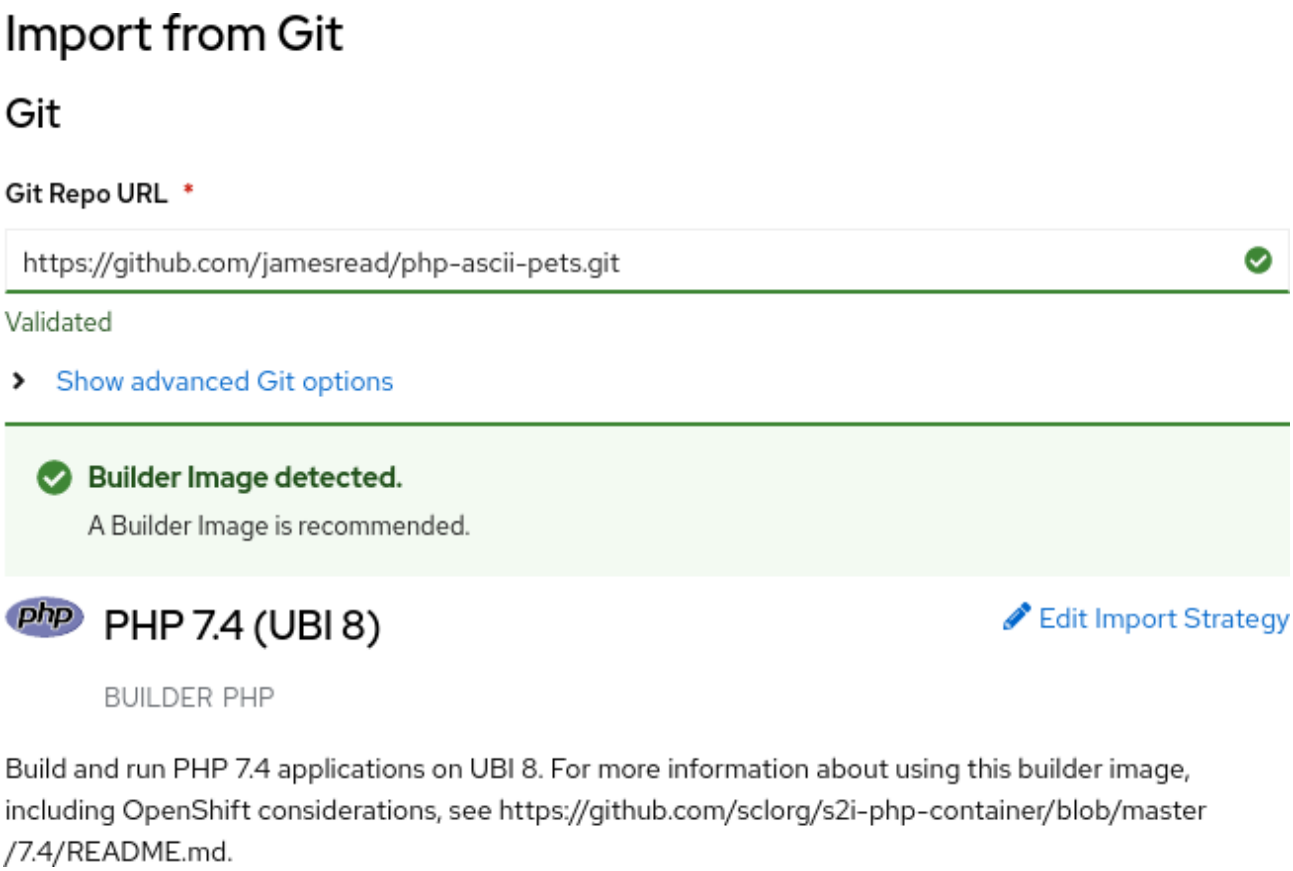


図 8.7：ビルダーイメージを自動的に検出して推奨

これを「サーバーレス」デプロイメントにするために重要なのは、デプロイメントタイプの選択です。OpenShift Serverless がインストールされると、「サーバーレス」デプロイメントが利用できるようになります。

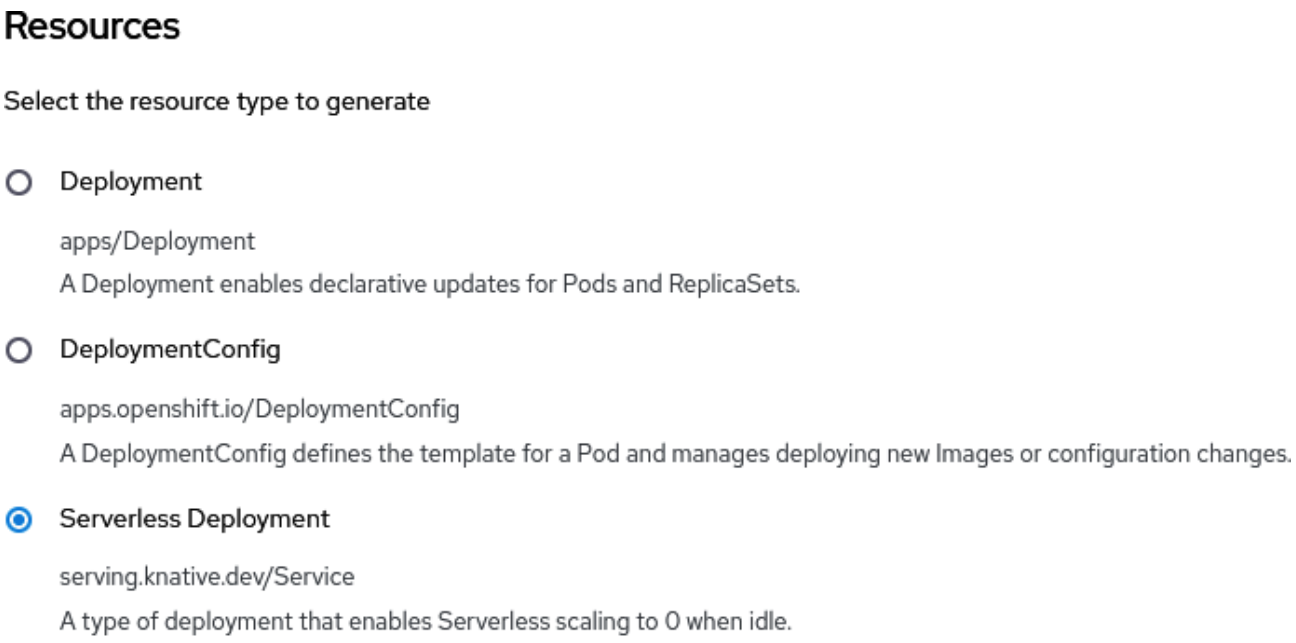


図 8.8：サーバーレス・デプロイメントの種類

最初のビルドが完了するまで少し時間がかかります。ビルドが完了すると、Pod がデプロイされています。トポロジービューは以下のようになります。

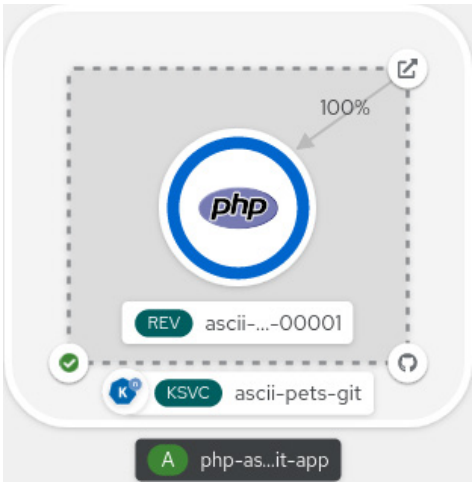


図 8.9：Knative Serving アプリケーション

実際のアプリケーションのページは図 8.9 のようになります。これは非常に単純なアプリケーションですが、ASCII アートの「ペット」は Pod のホスト名に関連付けられています。この単純なデモでは、アプリケーションにさらに多くの負荷をかけると、Knative Serving が追加の Pod を生成し、さまざまな「ペット」が表示されることを視覚的に確認できます。

しかし、アプリケーションに対する要求のない状態が 1 分間継続した場合、Knative Serving はこのアプリケーションをゼロにスケールダウンします。トポロジービューを見ると、アプリケーションが実行されていないことがわかります。

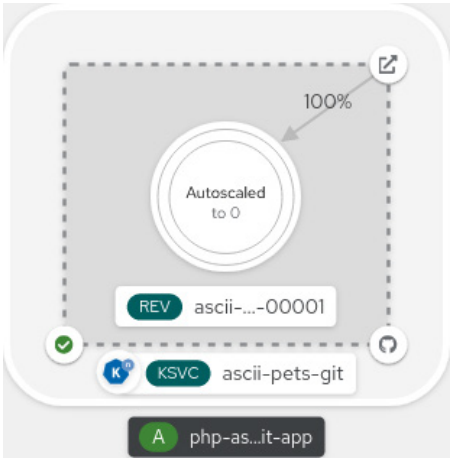


図 8.10：Knative Serving を使用してアプリケーションをゼロにスケーリングしたサーバーレス・デプロイメント

そして、誰かがページにアクセスすると、OpenShift が需要を満たすために必要と見なすインスタンスの数に応じて、アプリケーションは 1 レプリカ、2、3、またはそれ以上に自動スケーリングされます。

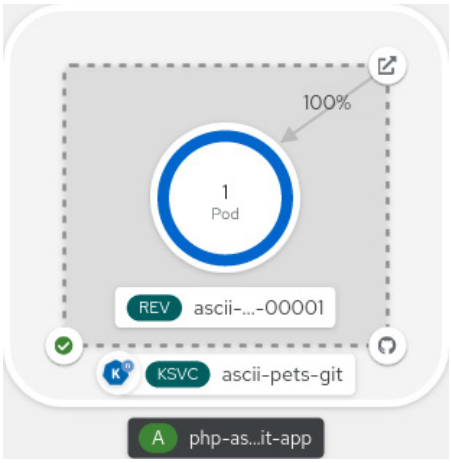


図 8.11：トラフィックに応じた自動スケーリング

Knative Serving と Red Hat OpenShift Serverless を使用すると、追加の構成がほぼ必要なく、アプリケーションを変更することなく、動的にスケールアップやスケールダウンを行えます。これは、デプロイメントの適切なサイズを保ち、リソースが不必要に使用されたり、課金されたりするのを防ぐために極めて有用です。

参考資料

- OpenShift Serverless について
- learn.openshift.com (OpenShift Serverless に関するコースがある)
- Knative コミュニティサイト

プラットフォームサービス – OpenShift Serverless – Knative Eventing の例

前の章のサンプルアプリケーションと基本を踏まえて、OpenShift Serverless は、Ingress トラフィックだけでなく、メトリクスに基づいてアプリケーションをスケーリングすることもできます。特にイベント駆動型アーキテクチャなどのシナリオでは、タイマーに応じてウェイクアップするなど、メッセージキューからのイベントを処理するために、アプリケーションのインスタンスをスケールアップできることが求められます。

同じデプロイメントを使用して、OpenShift コンソールでさまざまなイベントソースを簡単に設定することができます。

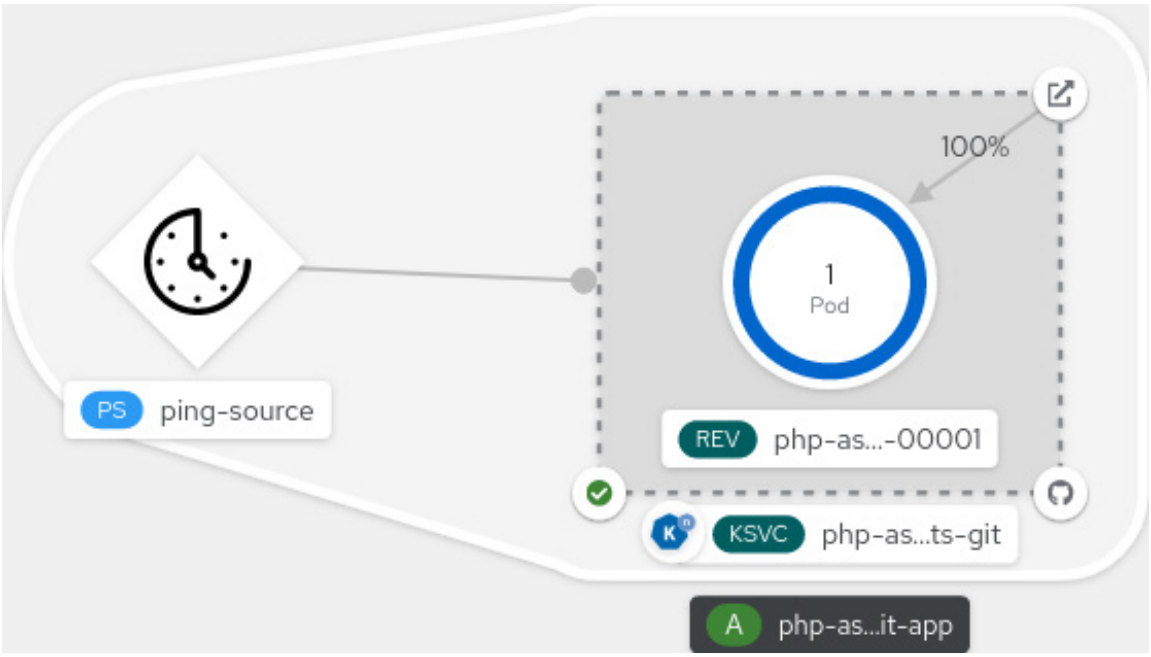


図 8.12：タイマーに従ってアプリケーションのネットワーク接続を確認する「ping」イベントソース

ping イベントソースの単純なユースケースには、毎晩深夜にバックアップジョブのコンテナを「ウェイクアップ」することなどがあります。また、この ping タイマーの別の用途としては、定期的に実行される監視コンテナがあります。

参考資料

- 5 分でわかる OpenShift Serverless のイベント処理
- OpenShift Serverless について

プラットフォームサービス – OpenShift Service Mesh

オープンソース・プロジェクトの Istio をベースとする Red Hat OpenShift Service Mesh は、既存の分散アプリケーションに透過的なレイヤーを追加します。サービスコードに変更を加える必要はありません。マイクロサービス間のすべてのネットワーク通信をインターセプトする特別なサイドカープロキシを環境全体にデプロイすることにより、Red Hat OpenShift Service Mesh サポートをサービスに追加します。サービスメッシュの構成と管理にはコントロールプレーン機能を使用します。

OpenShift Service Mesh で実現可能なユースケースには次のようなものがあります。

- 自動 mTLS による、サービス間通信への透過的な暗号化
- 複数バージョンのサービスの提供 (A/B テストなど)
- マイクロサービス間の通信における可観測性 (Kiali を使用)
- サービス間呼び出しのトレース (Jaeger を使用)

OpenShift の他のあらゆるプラットフォーム機能と同様、Service Mesh は Red Hat の Operator と一緒にインストールされます。

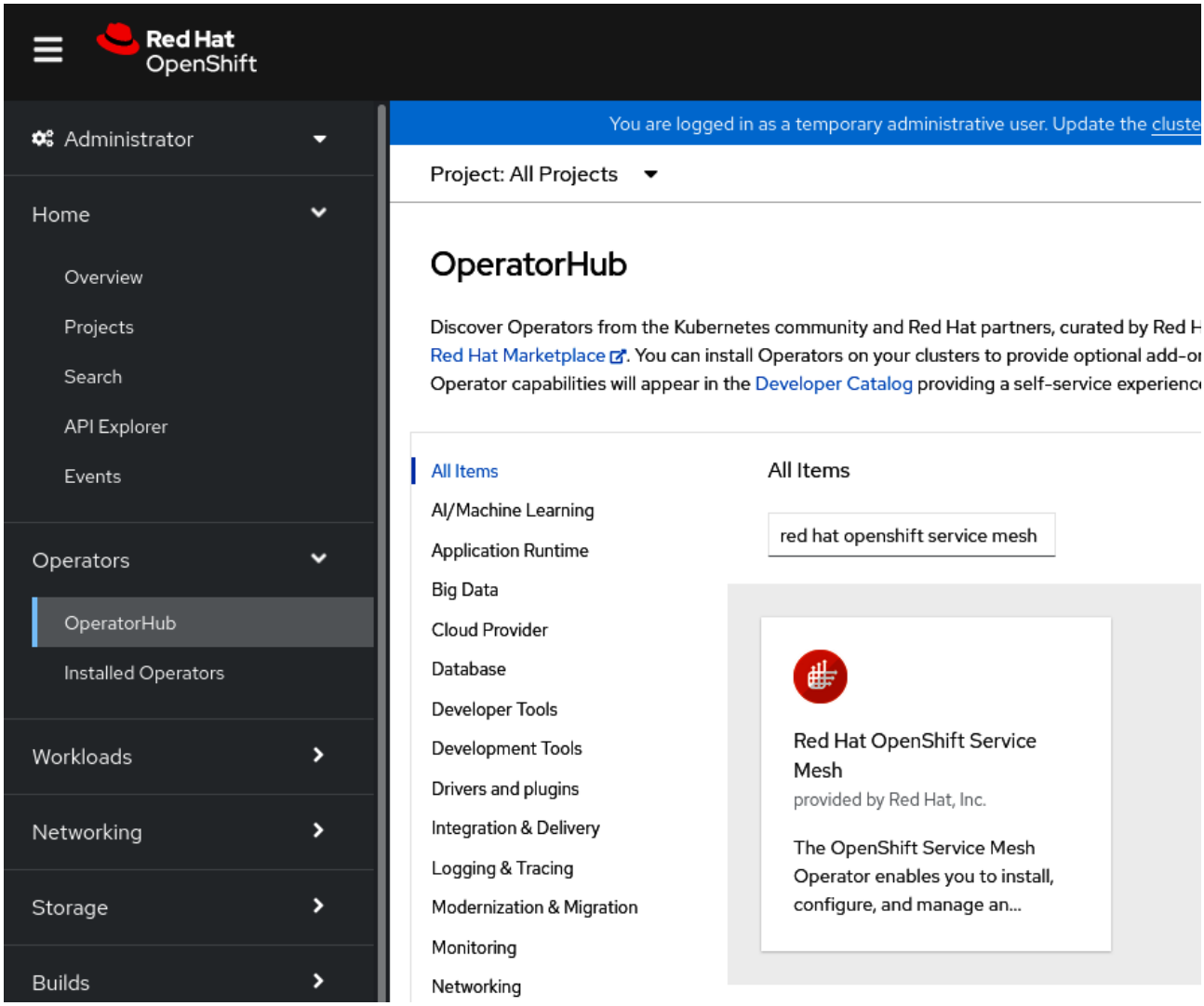


図 8.13：OperatorHub を介した Service Mesh のインストール

OpenShift Service Mesh のドキュメントには、BookInfo demo という素晴らしいサンプルアプリケーションが付属しています。これは書店をエミュレートする単純なアプリケーションで、OpenShift で実行されます。

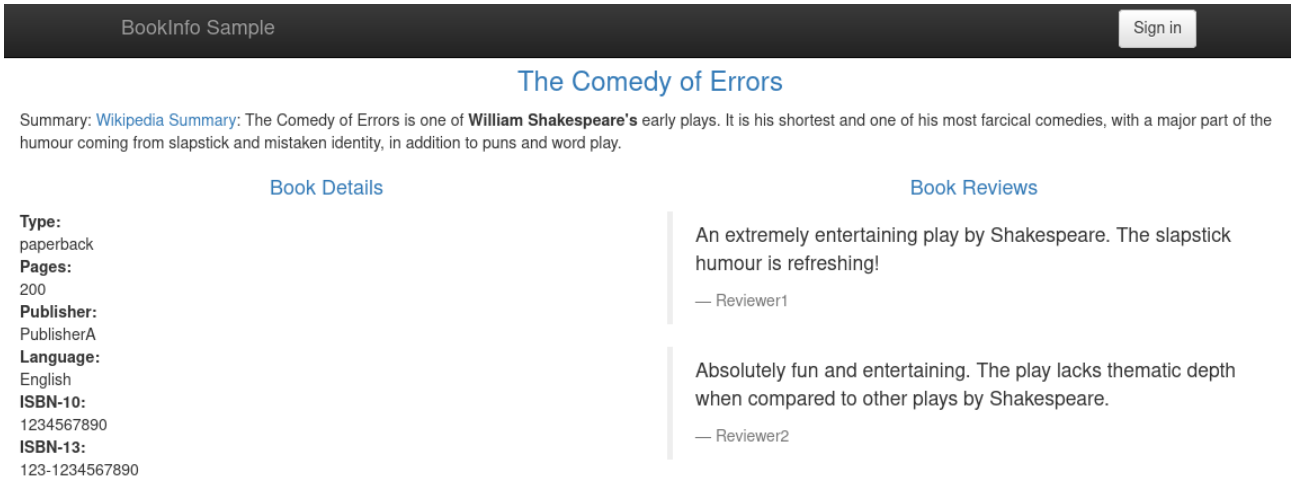


図 8.14：BookInfo アプリケーション

BookInfo アプリケーションを Service Mesh で動作させるには、Service Mesh コントロールプレーンを作成する必要があります。これは、図 8.15 に示すように、OpenShift グラフィカル・インタフェースを介して簡単に実行できます。

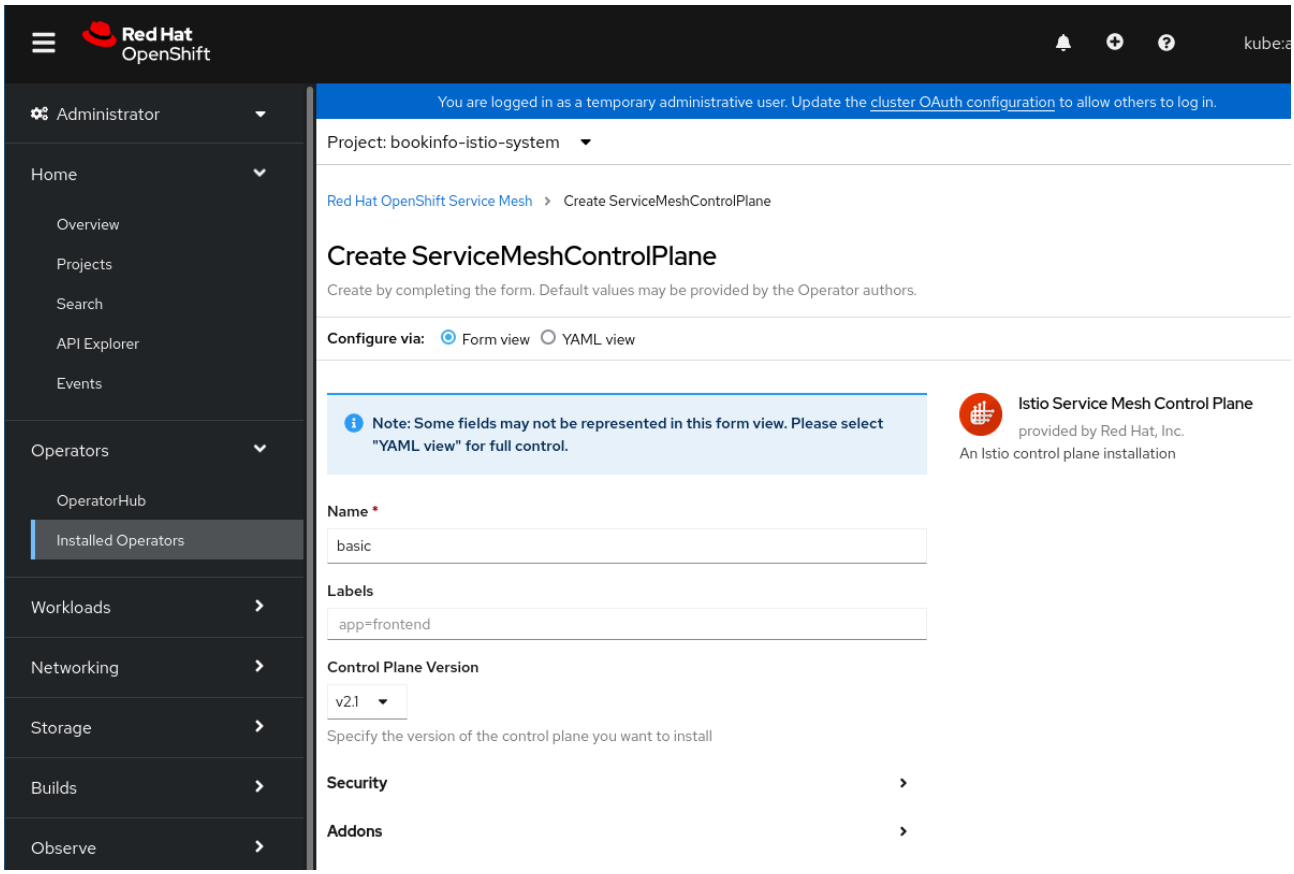


図 8.15：BookInfo bookinfo-istio-system プロジェクトでのコントロールプレーンのセットアップ

BookInfo プロジェクトは、コントロールプレーンを介して Ingress トラフィックを受け入れます。ここでは、bookinfo-istio-system という、コントロールプレーンを格納するための別のプロジェクトが作成されました。

Service Mesh コントロールプレーンとプロジェクトを分離する必要はありません。また、複数のプロジェクト間で Service Mesh コントロールプレーンを共有することも可能です。

次の 2 つのセクションでは、可観測性と分散トレースという 2 つの Service Mesh のユースケースについて詳しく説明します。

プラットフォームサービス – OpenShift Service Mesh – Kiali による可観測性

Red Hat OpenShift の Topology ビューでこのアプリケーションを構成している基本的な Pod を確認すると、6 つのマイクロサービスで構成されていることがわかります。

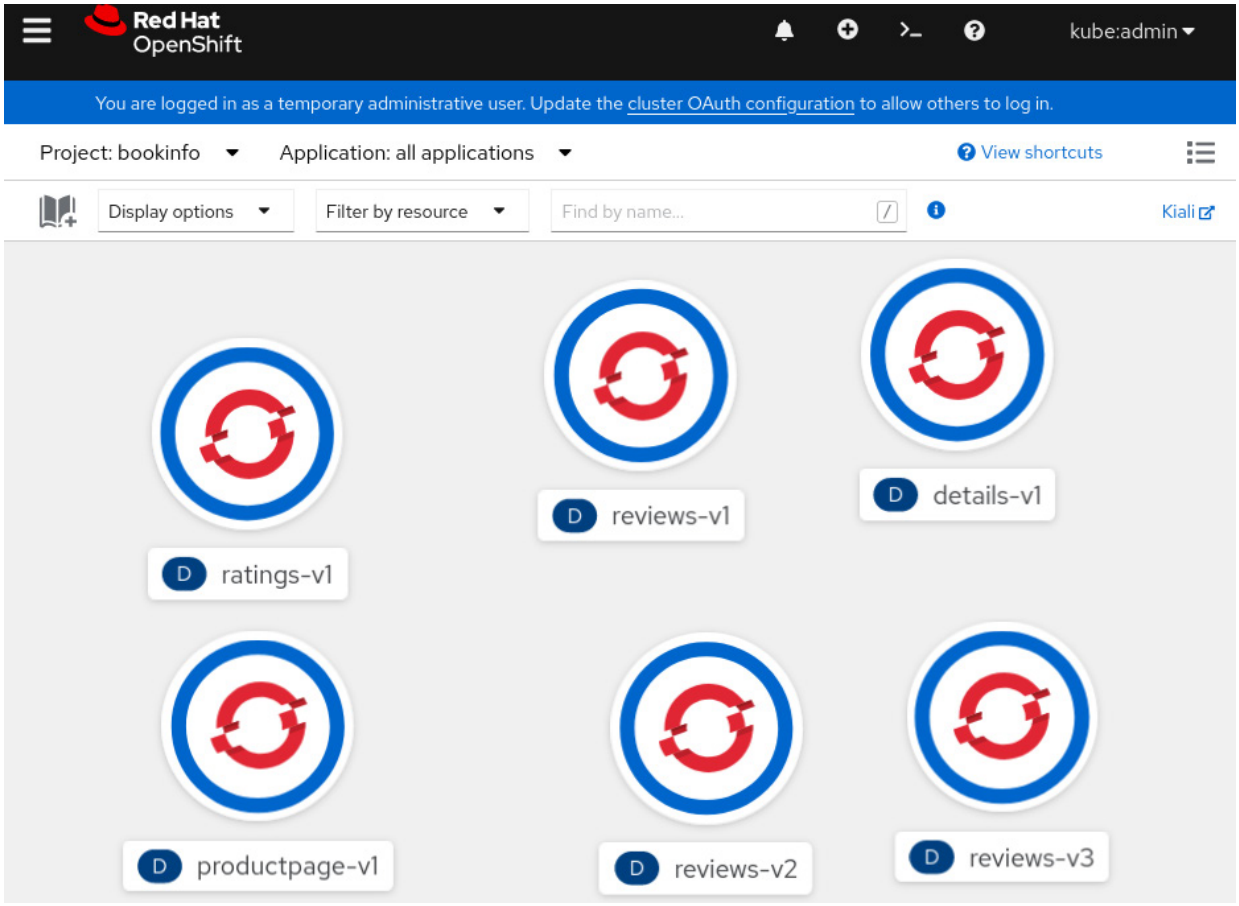


図 8.16：BookInfo アプリケーションを構成する 6 つのサービス

このビューは便利ですが、これらのサービスがどのように接続されているかを実際に伝えるものではありません。そこで、Service Mesh の第一のメリットである可観測性を利用します。Service Mesh には Kiali という、わずかに異なるコンソールが付帯しています。これを開くと、同じ 6 つのサービスのはるかに洗練されたアーキテクチャが表示されます。

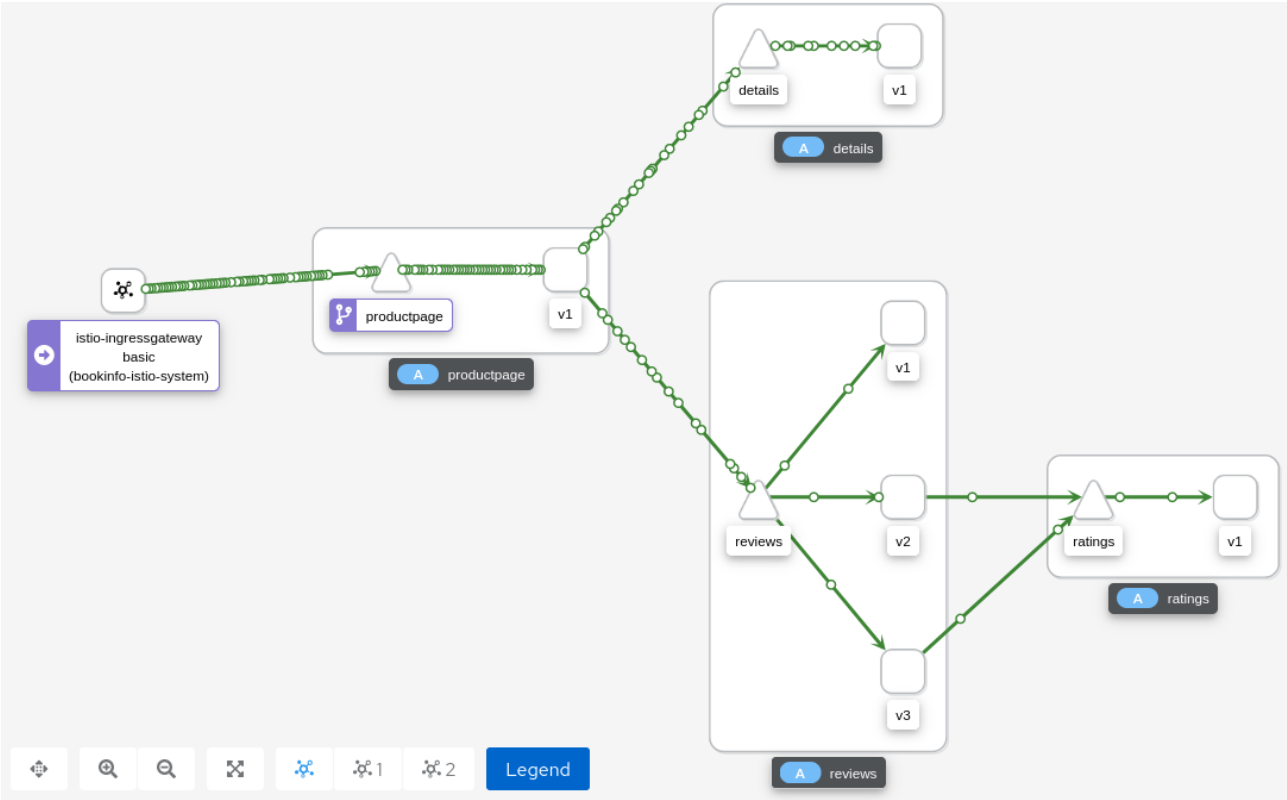


図 8.17：Service Mesh によって自動生成されたアプリケーション・アーキテクチャの図

このビューには、サービス間の実際の接続だけでなく、リアルタイムのトラフィック図も表示されます。この例では、アプリケーションはかなりの量のトラフィックを受信しています。各要求は接続を示すライン上の円のアニメーションで表されています。

この可観測性をさらに高めると、管理者はサービス接続をクリックしてそのトラフィックのプロファイルを確認することができます。この場合、トラフィックのほとんどが **HTTP OK** ステータスであることがわかります。

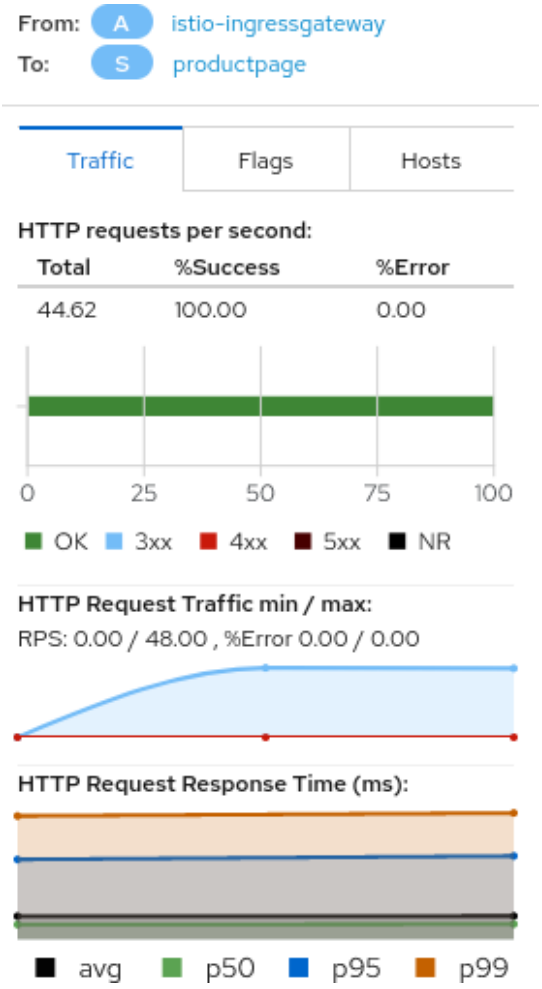


図 8.18：さまざまな HTTP ステータス

details マイクロサービスをシャットダウンし、レプリカをゼロにスケーリングすることで、このアプリケーションに人為的にエラーを発生させることができます。

図 8.19：details サービスのレプリカをゼロにして障害をシミュレート

ここで、Kiali のビューに戻ると、図にいくつかのコンポーネントが追加されていますが、最も注目すべきは、details サービスへの接続がエラーを示すために赤で強調表示されていることです。

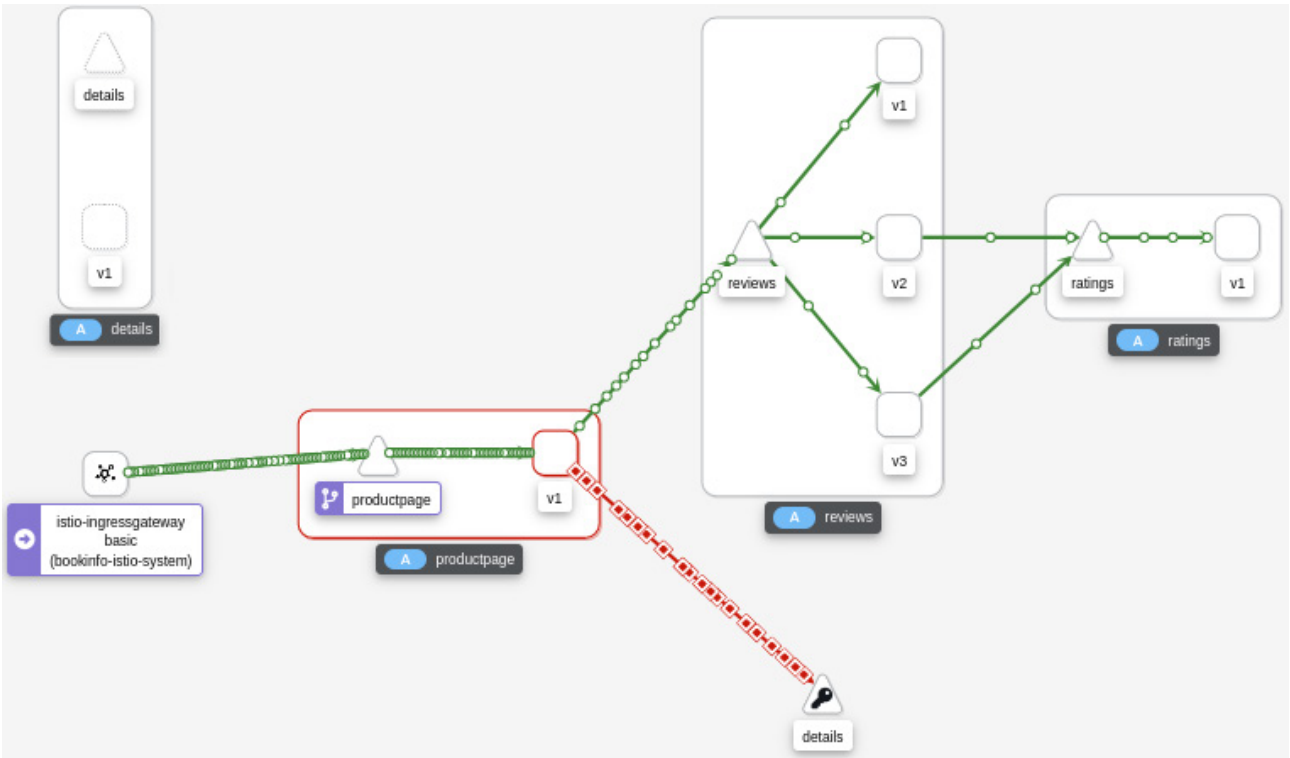


図 8.20：このサービスに問題があることを示す赤色の接続

ここまでで、Service Mesh の Kiali は、可観測性の獲得に極めて有用であることを学びました。BookInfo のような小規模のアプリケーションでは Service Mesh が役立ちますが、アプリケーションが大きく複雑になると、20 以上のマイクロサービスやさまざまな種類の対話が含まれる場合があり、Service Mesh に加えて Kiali などのツールを使用することは極めて有益であり、ほぼ不可欠です。

プラットフォームサービス – Red Hat OpenShift Service Mesh – Jaeger による分散トレース

Service Mesh が提供するもう 1 つの極めて有益なサービスは Jaeger です。これにより、複雑なマイクロサービス・アプリケーションの分散トレースが可能になります。前のセクションでは、BookInfo アプリケーションで `details` サービスがオフラインになると、要求が機能しなくなったことを確認しました。

自分で `details` サービスを利用できないようにしたわけですから、この障害の原因の特定は簡単です。しかし、原因が不可解でさらに調査が必要な場合は、Jaeger の分散トレースが役立ちます。

Jaeger は、システムを介した後続の接続を通じて最初のサービスからの要求を追跡します。Jaeger を有効にするには追加のアプリケーションコードが必要ですが、クライアントライブラリと最小限のコード変更により、開発者はこれを比較的簡単に行うことができます。

`productpage` サービス (Python で記述されている) を確認すると、そのサービスで Jaeger 分散トレースを有効にする関連コードを見つけることができます。このコードは、Jaeger クライアントライブラリを `productpage` サービスにインポートします。

```
from jaeger_client import Tracer, ConstSampler
from jaeger_client.reporter import NullReporter
from jaeger_client.codecs import B3Codec
```

クライアントライブラリがインポートされたら、製品ページで新しいトレーサーを設定する必要があります。次のコードは、productpage サービスで Jaeger の Tracer を初期化します。

```
tracer = Tracer(
    one_span_per_rpc=True,
    service_name='productpage',
    reporter=NullReporter(),
    sampler=ConstSampler(decision=True),
    extra_codecs={Format.HTTP_HEADERS: B3Codec()}
)
```

最後に、引き続き productpage サービスで、新しいスパン、つまり、いくつかのサービスに対する新しい要求を作成することを Jaeger に通知します。

```
span = tracer.start_span(
    operation_name='op', child_of=span_ctx, tags=rpc_tag
)
```

すると、Jaeger はいくつかのサービスを通じてこのトレーサーを追跡します。これらも Jaeger で動作するように調整する必要がありますが、クライアントライブラリはほとんどの一般的なプログラミング言語で利用できます。

productpage サービスの完全なソースコードは GitHub にあります。

コードをインストルメント化するには 2 つの方法があります。1 つは Jaeger クライアントライブラリを使用することですが、現在は非推奨とされています。おすすめは OpenTelemetry プロジェクトのオープンスタンダード・オプションを使用することです。

- OpenTelemetry ライブラリ

アプリケーションのインストルメント化の作業が完了すると、次のようなトレースが表示されます。

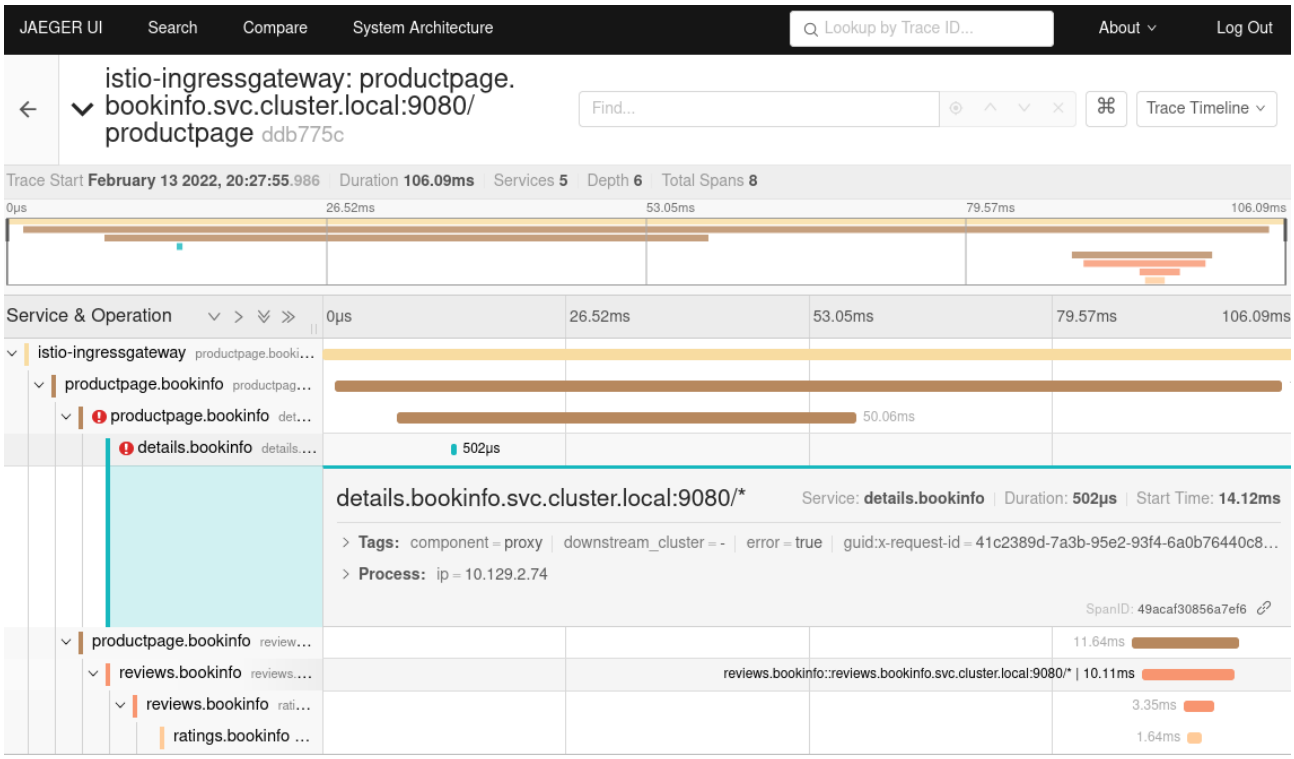


図 8.21：コールトレース

このビューにはコールトレースが表示されます。Kiali と似ていますが、階層ビューで表示されます。表示されている各行を展開すると、要求の期間、開始時間、送信元 IP アドレス、その他の詳細が表示されます。繰り返しになりますが、このレベルの情報は、複雑なマイクロサービス・アプリケーションを扱う場合にはほぼ不可欠です。

診断しようとしたエラーのことを思い返すと、このトレースのビューは、details マイクロサービスに問題が発生していることを明確に示しています。この場合、エラーは単純ですが、details ビューを展開すると、サービスが HTTP 503 エラーコードを発行していることがわかります。

図 8.22：エラーの詳細

HTTP 503 は、「サービス利用不可」を示す標準エラーコードです。この例では、原因は単にサービスのレプリカをゼロにスケーリングしていることです。再度スケールアップすれば、サービスは復旧します。

より複雑なマイクロサービス・アプリケーションの場合、Jaeger と Kiali を組み合わせると強力なツールになります。これらは Azure Red Hat OpenShift サービスの一部として提供されており、追加の費用やサブスクリプションは必要ありません。

まとめ

本章では、単なる素の Kubernetes 環境ではなく、アプリケーション・プラットフォームである Red Hat OpenShift によって提供される主要な付加価値サービスをいくつか取り上げました。Kubernetes にこれらのアップストリーム・オープンソース・プロジェクトをすべてインストールすることで、OpenShift と同様のレベルの機能を持たせることは可能ですが、Azure Red Hat OpenShift では統合、ライフサイクル、サポートがさらに提供されます。

これらのさまざまなプラットフォームサービス、アプリケーションサービス、データサービス、開発者サービス、およびクラスタサービスによって、Kubernetes を使用する、または複数の複雑なエンタープライズ・アプリケーションをデプロイする開発者や運用者の生産性が向上します。

第 9 章

他のサービスとの統合

通常、アプリケーションが Red Hat OpenShift 内だけで存続することはできません。ほとんどのアプリケーションは、何らかの形で Azure 上の外部データベースやサービスに依存しています。

Azure Red Hat OpenShift は、いかなる種類の接続も中断させません。たとえば、アプリケーションが Azure CosmosDB を使用している場合、そのアプリケーションに変更を加えなくても、Azure Red Hat OpenShift から Azure CosmosDB に接続できます。アプリケーションや組織に応じて、これらの外部依存関係の一部を Azure Red Hat OpenShift とは別にデプロイすることも、同時にデプロイすることもできます。

これらの外部依存関係をアプリケーションにデプロイすることが有益であると思われる場合は、Azure Service Operator を使用すると、このプロセスを大幅に単純化できます。Azure CLI、Azure Cloud Shell、または ARM テンプレートを使用する代わりに、Azure Service Operator を利用することで、これらのリソースを Kubernetes ネイティブであるかのようにデプロイできます。

Azure Service Operator

Azure Service Operator は、Azure Red Hat OpenShift にアプリケーションをデプロイする開発者または管理者の作業を楽にするもう 1 つの Operator です。その利点は、OpenShift コンソールを離れることなく Azure サービスを簡単にプロビジョニングできることです。これにより、Azure CosmosDB など、Azure サービスに依存している可能性のあるアプリケーションをより迅速かつ容易にデプロイできます。

Azure Service Operator を使用するもう 1 つの、おそらくより重要な利点は、Infrastructure-as-Code (IaC) のアプローチにおいて、標準の Kubernetes YAML ファイルを使用して Azure サービスでの依存関係を OpenShift アプリケーションの定義と一緒に保存できることです。

仕組みは簡単です。Azure Service Operator は、Azure 上のリソースの定義に一致する、YAML で定義された新しい **CRD** を探します。次に、Azure Service Operator は、この YAML を必要な Azure API 呼び出しに変換して、Azure で要求されているものを作成します。Operator がインストールされると、ユーザーは OpenShift Developer Catalog を参照して、Azure パブリック IP アドレス、Azure SQL データベース、Azure Firewall などの多くの一般的な Azure リソースの作成画面を確認できます。

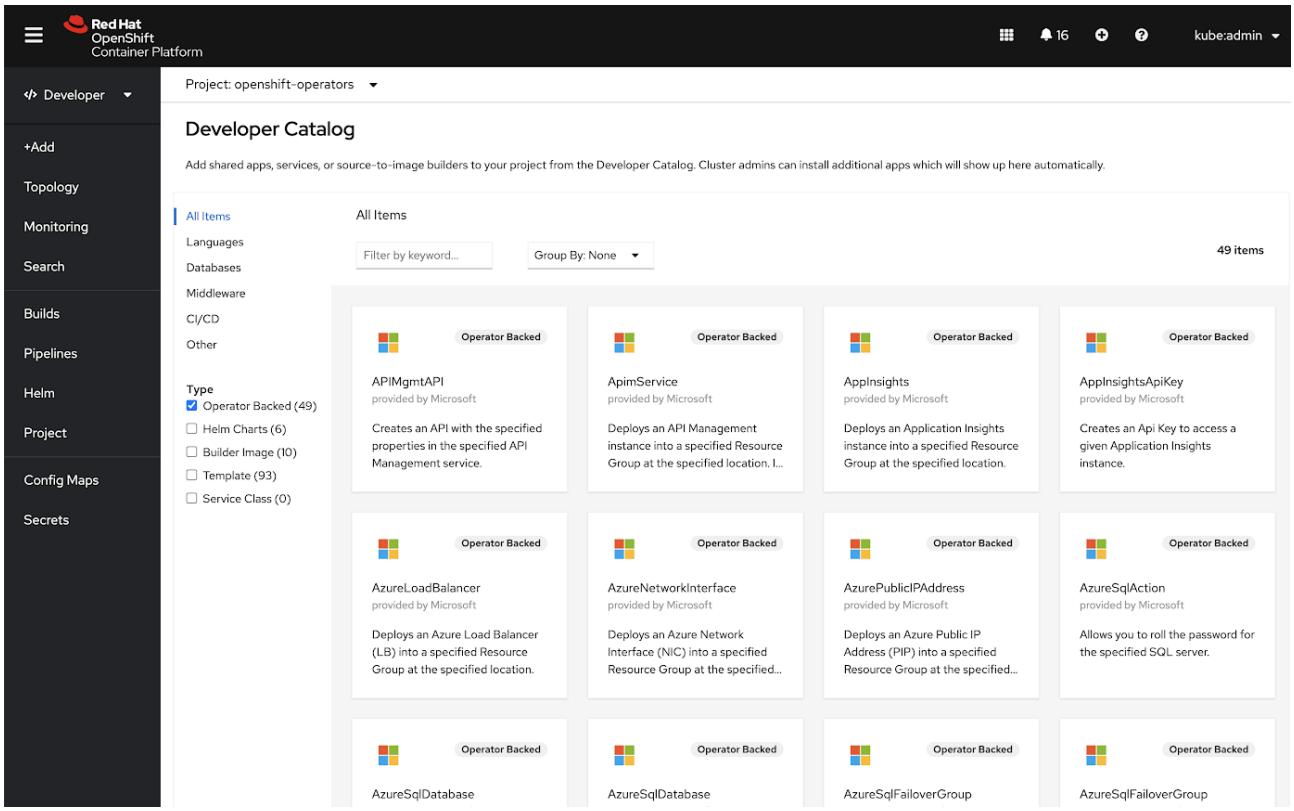


図 9.1：Azure Service Operator によって公開される Azure サービスの一部

Azure Service Operator は OperatorHub を介してインストールし、OpenShift のすべてのユーザーが利用できるようにすることができます。

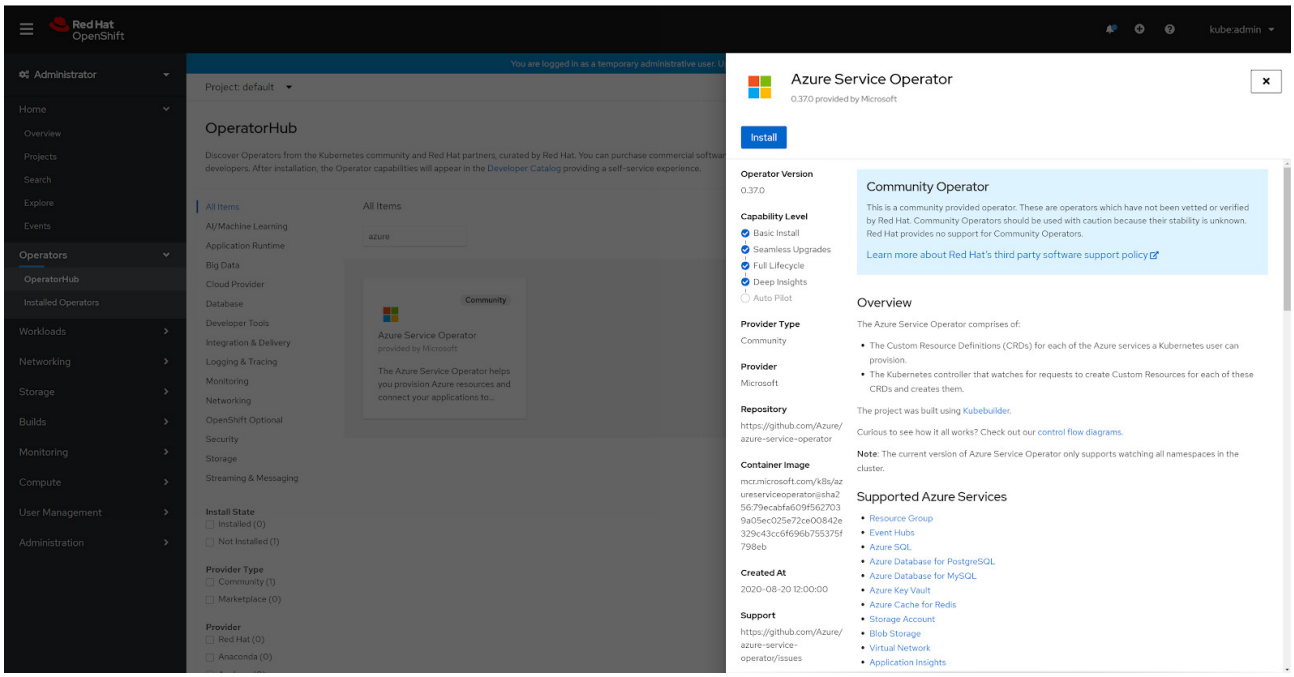


図 9.2：Azure Service Operator のインストール画面 (後ろの画面は OperatorHub)

Azure で実行されているサービスでの Azure Service Operator の使用は必須ではありません。また、基盤となる Azure サービスは、OpenShift ではなく Azure にデプロイされることを理解することが重要です。しかし、Azure Service Operator を使用すると、Azure サービスに依存するアプリケーションではこれがより迅速かつ容易になります。

Azure Service Operator の一般的なユースケースは、依存するデータベースをアプリケーションと一緒にデプロイすることです。たとえば、Azure CosmosDB のインスタンスを使用する Web アプリケーションの場合、OpenShift にアプリケーションをデプロイするのと同時に、Azure Service Operator を使用して Azure CosmosDB をデプロイします。Azure Service Operator には、Azure SQL Database、MySQL 用 の Azure Database、Azure PostgreSQL などのサポートが含まれています。

Azure Service Operator がインストールされていれば、次の Kubernetes マニフェストを OpenShift アプリケーションと一緒に保存し、MySQL データベースのプロビジョニングに使用できます。

ソース：Azure Service Operator のサンプルリポジトリから取得

```
apiVersion: azure.microsoft.com/v1alpha1
kind: MySQLServer
metadata:
  name: aso-wpdemo-mysqlserver
spec:
  location: eastus2
  resourceGroup: aso-wpdemo-rg
  serverVersion: "8.0"
  sslEnforcement: Disabled
  minimalTLSVersion: TLS10 # Possible values include: 'TLS10', 'TLS11', 'TLS12', 'Disabled'
  infrastructureEncryption: Enabled # Possible values include: Enabled, Disabled
  createMode: Default # Possible values include: Default, Replica, PointInTimeRestore (not
implemented), GeoRestore (not implemented)
  sku:
    name: GP_Gen5_4 # tier + family + cores eg. - B_Gen4_1, GP_Gen5_4
    tier: GeneralPurpose # possible values - 'Basic', 'GeneralPurpose', 'MemoryOptimized'
    family: Gen5
    size: "51200"
    capacity: 4
```

複雑な依存関係を持つ多くの Azure サービスが関係するサービスに Azure Service Operator を使用することはまれであり、そのような場合は Azure ARM テンプレートや Bicep などのツールの方が適している可能性があります。

Azure Service Operator の詳細は、次のブログ記事を参照してください。

- ブログ記事 – 2020 年 9 月
- OperatorHub.io の Azure Service Operator
- GitHub の Azure Service Operator

Azure DevOps との統合

多くのユーザーは、OpenShift Pipelines に加えて、Azure Red Hat OpenShift を Azure DevOps と統合したいと考えるでしょう。これらのソリューションは共存できます。いずれかのソリューションと他の何らかのソリューションを使用するプロジェクト、場合によっては両方を利用するプロジェクトもあります。どちらのソリューションをいつ使用するかは、いくつかの要因によって決まります。

一般的に、Azure DevOps は、他の Azure ツールとの高レベルの統合を提供しますが、OpenShift や他のコンピューティングリソースにも簡単にデプロイできます。

一方、OpenShift Pipelines は OpenShift に付属するオファリングで、OpenShift と緊密に統合されており、一貫性のあるマルチクラウド・エクスペリエンスを提供します。

Azure DevOps は、OpenShift を他の Kubernetes クラスタと同じように扱います。したがって、標準の Kubernetes インタフェースと API はすべて予期したとおりに機能します。

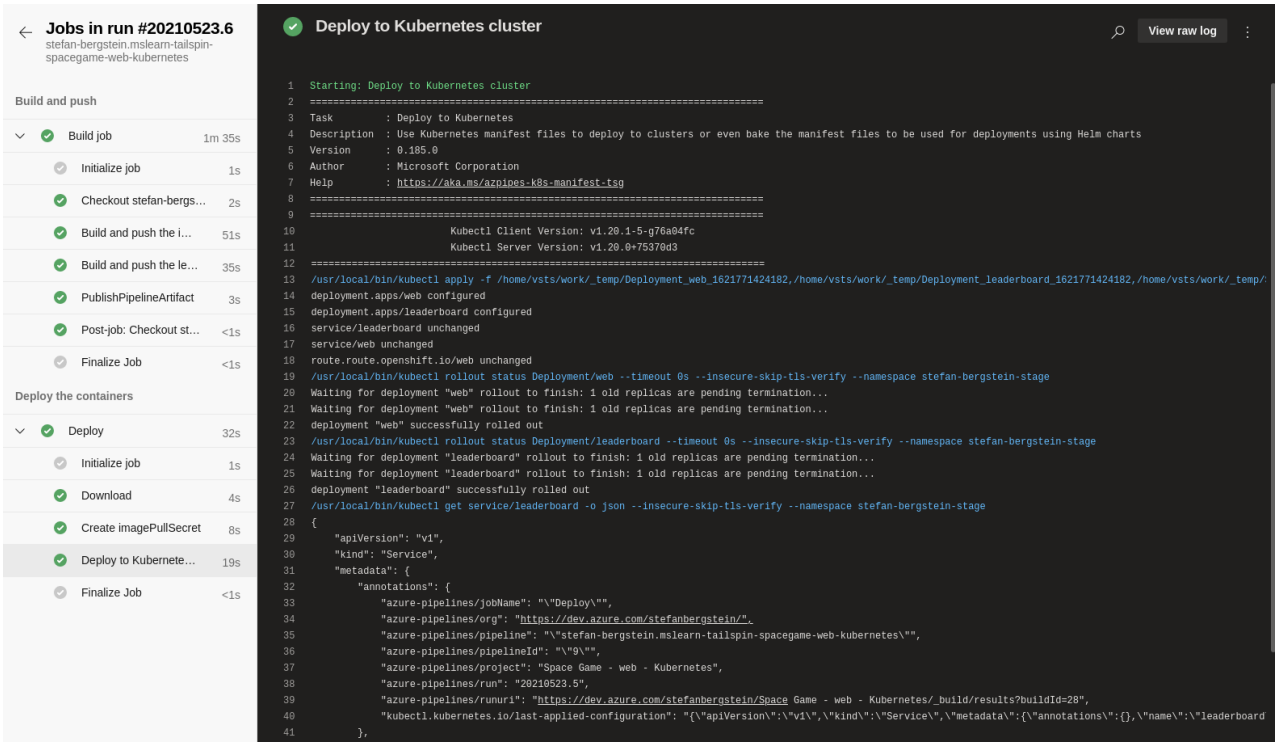


図 9.3：OpenShift にコンテンツをプッシュする Azure DevOps パイプライン

参考資料

以下のコミュニティによるブログ記事では、Azure Red Hat OpenShift と Azure DevOps の使用方法に関する優れたチュートリアルを提供しています。

- ブログ記事 – 2021 年 5 月

Visual Studio Code との統合

OpenShift の開発者統合の機能として、Visual Studio Code など、多くの一般的な IDE 用のプラグインがあります。これにより、開発者と管理者は、IDE を離れることなく Kubernetes と OpenShift のリソースにすばやく簡単にアクセスできます。

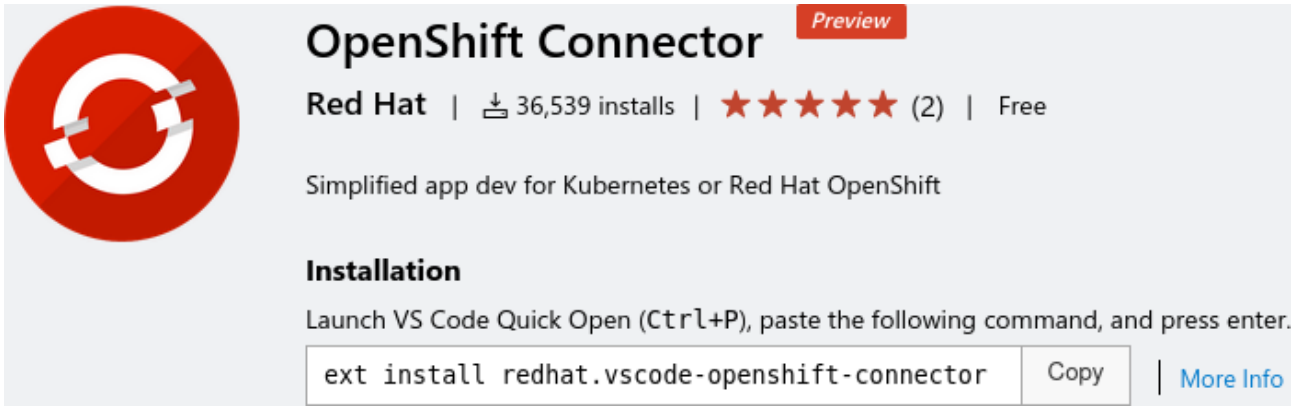


図 9.4：OpenShift Connector のインストール

コネクターをダウンロードして Visual Studio Code にインストールすると、OpenShift クラスタへのログインプロンプトが表示されます。以下は、Azure Red Hat OpenShift に接続した、単純な「hello world」スタイルのプロジェクトです。

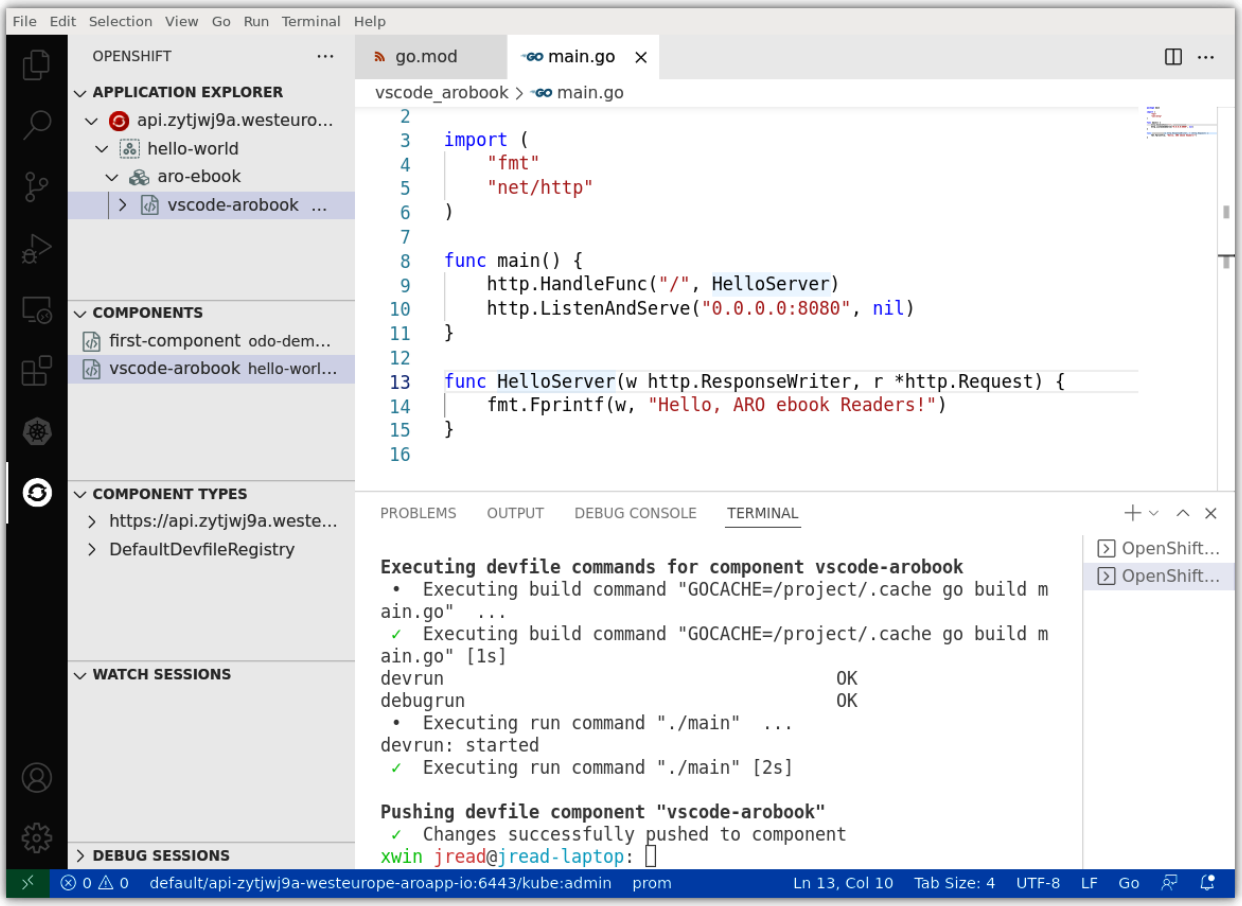


図 9.5：Visual Studio Code OpenShift コネクターを使用してデプロイされた Golang プロジェクト

Visual Studio Code で OpenShift コネクターを使用するメリットの 1 つは、一般的な OpenShift ツールの「OpenShift Do」(通常は単に「odo」と呼ばれる) のグラフィカル・フロントエンドを利用できることです。これにより、開発者は単なる素の Kubernetes コンポーネントよりも高いレベルの概念で考えることができます。開発者が Deployment、ReplicaSet などの詳細を心配する必要はあまりありません。前のスクリーンショットから、これは公開されている HTTP サービスを使用した単純なプロジェクトであることがわかります。

さらに、コネクターには、コード変更を OpenShift に直接プッシュする機能も含まれており、必ずしも最初にソース管理を使用する必要はありません。毎回ソース管理にプッシュし、新しいコンテナをビルドしてデプロイする必要がないため、迅速な開発に極めて役立ちます。

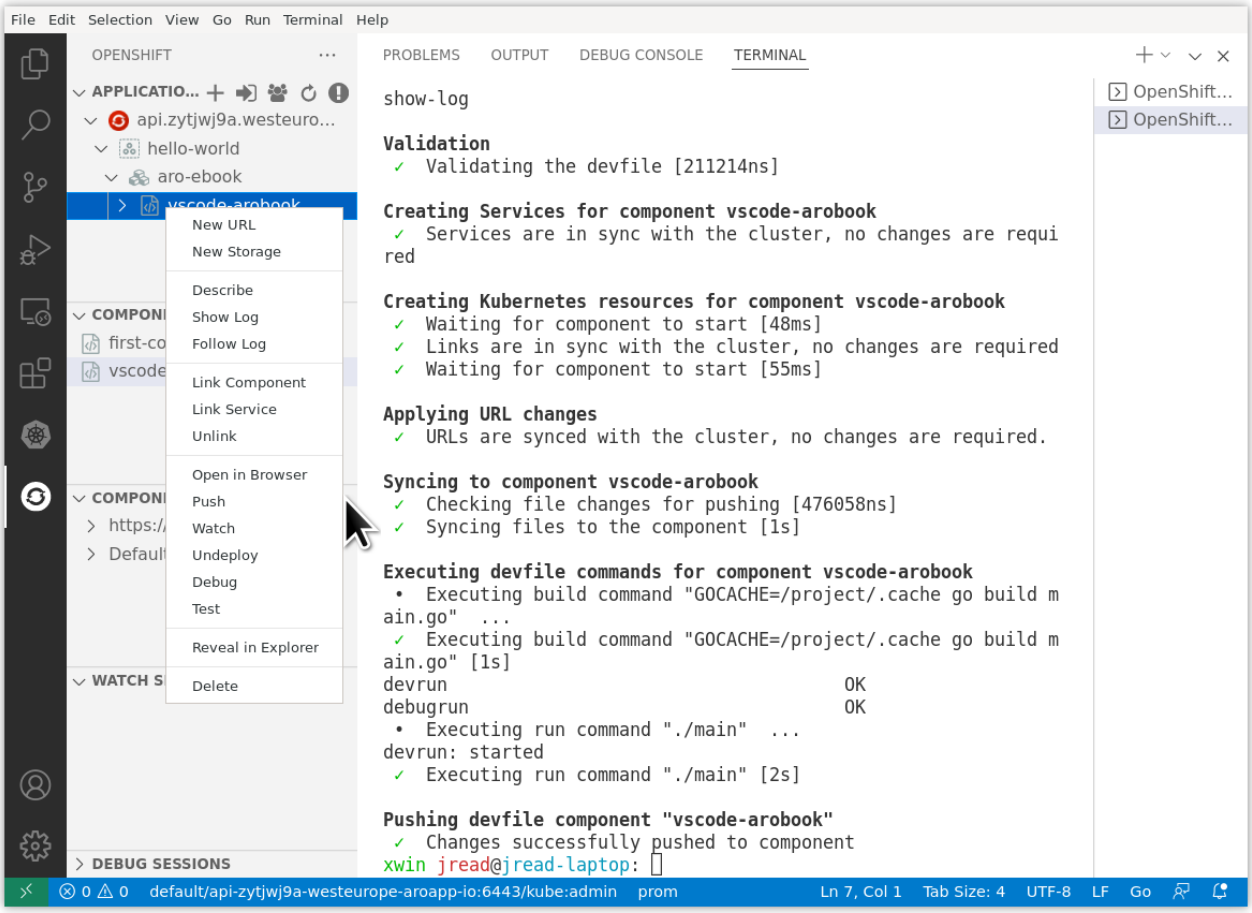


図 9.6：プロジェクトの「push」メニューと、ターミナルウィンドウに表示された最近のプッシュ

このコネクターは、Visual Studio Code の使用を好む開発者の開発およびテストのサイクルを加速します。

図 9.7：OpenShift で実行される基本的な Golang アプリケーション

もちろん、開発者が使用できるのは Visual Studio Code だけではありません。多くの開発者が Vim や Eclipse などのエディターを使用していますが、Visual Studio Code は「軽量 IDE」スタイルのエクスペリエンスを好む開発者の間で極めて高い人気を誇ります。

参考資料

- デモ動画 – OpenShift で Visual Studio Code を使用する
- Visual Studio Code OpenShift Connector

GitHub Actions との統合

GitHub Actions を使用して、Azure Red Hat OpenShift を含むあらゆる Red Hat OpenShift 環境にデプロイできるようになりました。任意の GitHub リポジトリから、**Actions** → **New Workflow** に移動し、使用可能なアクションのリストから **OpenShift** を選択します。

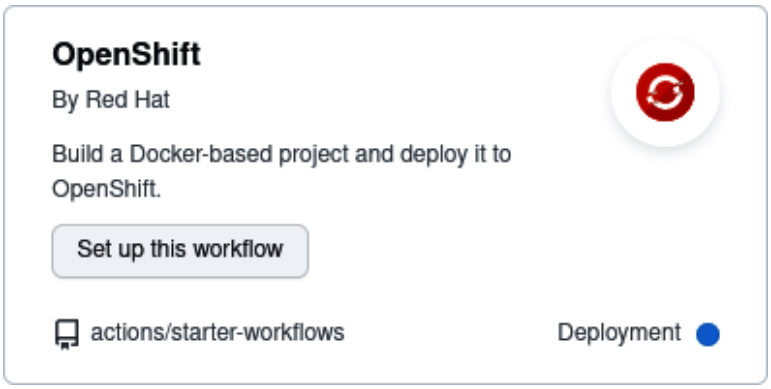


図 9.8：GitHub から OpenShift へのデプロイ

デフォルトのテンプレートにはアクションのサンプルセットが付属しており、OpenShift クラスタに接続するには環境変数をいくつか設定するだけです。

```
name: OpenShift

env:
  # ✏ EDIT your repository secrets to log into your OpenShift cluster and set up the context.
  # See https://github.com/redhat-actions/oc-login#readme for how to retrieve these values.
  # To get a permanent token, refer to https://github.com/redhat-actions/oc-login/wiki/Using-a-
Service-Account-for-GitHub-Actions
  OPENSHIFT_SERVER: ${{ secrets.OPENSHIFT_SERVER }}
  OPENSHIFT_TOKEN: ${{ secrets.OPENSHIFT_TOKEN }}
  # ✏ EDIT to set the kube context's namespace after login. Leave blank to use your user's
default namespace.
  OPENSHIFT_NAMESPACE: ""
  ...
```

その方法を説明する優れたブログ記事とデモ動画のリンクを以下に示します。

- ブログ記事
- デモ動画：GitHub Actions と OpenShift

まとめ

本章では、お客様が Azure Red Hat OpenShift で使用する一般的な統合の多くについて説明しました。章のはじめに述べたように、標準の OpenShift API を使用すると、本章で取り上げていない他の多くのサービスとの統合が可能になります。OperatorHub のリストにある Operator を使用して、膨大な数のサービスを簡単に統合することもできます。

次の章では、アプリケーションチームを立ち上げ、組織内でのサービスの導入を促進するためのベストプラクティスと推奨事項について説明します。

第 10 章

ワークロードとチームのオンボーディング

プロビジョニングの準備の手順、Azure Red Hat OpenShift のプロビジョニング、そして、プロビジョニング後に必要な手順が完了したら、あとは開発者チームとアプリケーションチームをサポートし、クラスタでの作業を開始してもらうだけです。これをどのように進めるかは、最終的には組織の文化によって決まります。自分たちで調べ上げていくことを楽しむ開発者チームやアプリケーションチームもあれば、できるだけ多くの指導を受けたいと考えるチームもあります。

本章は、開発者チームとアプリケーションチームが Azure Red Hat OpenShift を使用するにあたり、より迅速に仕事に取り掛かれるようにする優れた基盤を提供するためのチェックリストとして役立ちます。

このチェックリストを使用する際は、ご自分の組織の構造や文化に応じて適宜修正してください。

チェックリスト：オンボーディングの準備

新しいワークロードのオンボーディングの前に、アプリケーションチームまたはそのワークロードの所有者に対して、Azure Red Hat OpenShift は以下のように、組織のベストプラクティスに応じたデプロイおよび設計が完了していることを伝えられるようにしておくことが重要です。

- Azure Red Hat OpenShift は、Azure ランディングゾーンか、組織のアプリケーション用に承認された別の Azure 環境にデプロイされています。
- ストレージクラスや認証など、クラスタに必要な「Day 2」のセットアップが完了しています (第 6 章：プロビジョニングの後処理 - Day 2 で説明)。
- クラスタは完全に構成済みで、プラットフォームチームがサポートしており、Red Hat とマイクロソフトが提供する統合サポートによる支援があります。
- クラスタは利用可能な最新の安定したバージョンに更新されており、パッチが適用されています。今後もパッチが適用されます。

- Azure Red Hat OpenShift クラスタは、以下の極めて重要なリソースに接続できます。
    - Azure ExpressRoute または VPN 経由でのオンプレミス環境への接続
    - エンタープライズ・アーティファクト・リポジトリ (Java Maven リポジトリなど) への接続
    - アプリケーションのビルド時に依存関係をダウンロードするための、パブリックインターネットへの接続

試験的ワークロードのオンボーディング

Azure Red Hat OpenShift を組織に展開する場合、一般的なパターンは、少なくとも 2 つの試験的ワークロードをオンボーディングすることです。

- **最小規模の実務ワークロード** – SRE チームにとって理想的な最初の試験的ワークロードは、極めて単純な技術要件を持つ、小規模で誰もがよく知っているアプリケーションです。通常、組織内にこのアプリケーションの実際のビジネスオーナーがおり、その点で、これは単純な「Hello World」アプリケーションよりもわずかに高度なアプリケーションだと言えます。しかし、最初のワークロードでのポイントは、Azure Red Hat OpenShift クラスタがクラスタレベル (Azure の接続、Azure Active Directory のログイン) でビジネスニーズを満たしていることを確認することと、「目の前にぶら下がっている果実」(あらゆるアプリケーションで生じる可能性のある一般的な問題) を特定することです。
- **実務的なリファレンスワークロード** – 最小の単純なワークロードをデプロイした後は、続けて十分に複雑で意味のある (ビジネスにとって重要、またはビジネスクリティカルでさえある) ワークロードをデプロイすると、導入を進める上で大きく役立ちます。このワークロードは、重要なデータベースへの接続、パフォーマンステスト、大規模なロギング/メトリクスなどの問題を特定して解決するのに役立ちます。重要なのは、この実務的リファレンスワークロードを組織の**内部基準**として使用できることです。これにより、開発チームとアプリケーションチームが、将来、Azure Red Hat OpenShift プラットフォームを自信を持って使用できるようになります。

もちろん、組織の状況に合わせてワークロードを最初にいくつかオンボーディングするというパターンもあります。たとえば、間もなく廃止予定のデータセンター内のアプリケーションや、古い Java アプリケーションサーバーで実行されているアプリケーションを対象にすることが必要な場合があります。

チェックリスト：他のチームとのオンボーディング・ミーティング

新しい開発チームやアプリケーションチームにクラスタでの作業を開始してもらう際に、オンボーディング・ミーティングを開催することをお勧めします。

- チームが使用する namespace を 1 つまたは複数作成します。
- チームに対して Red Hat OpenShift の簡単なデモを行い、GitHub (または同等のもの) の Deploy などの主要な機能について説明します。
- クラスタにターゲットワークロード (CPU、RAM、ストレージなど) をデプロイするために十分な予備容量があることを確認します。
- チームのメンバーがクラスタにログインできることを確認します。Azure Active Directory に接続することで簡単に行えます。
- クラスタを管理している SRE チーム (またはそれに類するチーム) の連絡先情報を提供します。
- OpenShift スタートガイドのリンクを提供します。
    - http://learn.openshift.com
    - https://github.com/openshift-labs/starter-guides
    - http://docs.openshift.com

チェックリスト：定期的なヘルスチェックのためのミーティング

開発チームまたはアプリケーションチームがクラスタでの作業を開始したら、定期的なヘルスチェックのためのミーティングを持つことには、多くの場合、意義があります。毎日軽くミーティングを行う場合もあれば、週に 1 回、30 分のミーティングで十分な場合もあります。ミーティングでは次のことを確認します。

- チームの進捗状況について — 何かのアプリケーションがデプロイされたか
- クラスタの使用における問題 (Red Hat OpenShift に関する知識や接続性の問題) があったか
- Azure またはクラスタが停止して、エクスペリエンスに悪影響を及ぼすようなことはあったか
- 疑問が生じた場合に SRE チームに相談できることと SRE チームの連絡先情報を改めて通知

これ以外にも、同様のプロジェクトで行った定期ヘルスチェックミーティングでの確認事項を参考にして、適切と思われるものをこのチェックリストに追加してください。

アンチパターン：開発者の遊び場/サンドボックス

組織での開発者による導入を促進するためによく使われる「アンチパターン」では、一般に「開発者の遊び場」と呼ばれるサンドボックスタイプの環境を提供し、開発者チームとアプリケーションチームに Azure Red Hat OpenShift の導入の開始を促します。補足的なフォローアップサポートやリソースがなければ、Azure Red Hat OpenShift は手ごわく、理解できないほど複雑な環境になってしまうことがあります。「サンドボックス」パターンを導入すると、無駄なクラウド・コンピューティング費用が発生し、空のクラスタがいくつもできてしまうことがほとんどです。

しかし、開発チームが「サンドボックス化」した経験を持つ組織のために、これを可能な限り成功させるためのヒントを次に示します。

- 少なくとも 1 つ、Azure Red Hat OpenShift でできることに関する短いデモを提供します
- いくつかのドキュメント、そして、少なくとも上述したガイドのリンクを提供します
- ちょっとしたコンペ形式の「ハッカソン」イベントを開催し、開発者が OpenShift を使用して何かを構築する場を作ります
- より手厚いサポート提供の申し出と、SRE チームの紹介のほか、より詳細な指導を受けられるオンボーディング・エクスペリエンスをオプションとして提供します

これらの導入パターンのチェックリストに従うと、プラットフォームの導入がスムーズに進む可能性が高くなります。

Azure Red Hat OpenShift Developer Workshop

Azure Red Hat OpenShift Developer Workshop (https://aroworkshop.io) は、組織内で使用できる便利なワークショップとして事前に作成されたものであり、Azure Red Hat OpenShift の実践的なエクスペリエンスをガイド付きで説明します。ワークショップは 2 つのラボ演習に分かれています。どちらも OpenShift を初めて使用する開発者または運用者を対象としており、ガイド付きチュートリアル形式で提供されます。どちらのラボも、OpenShift の主要な機能に焦点を当て、その使用方法を紹介しています。ラボ 1 は先進的なマイクロサービス設計の概要、ラボ 2 はサービスの詳細について説明します。

組織内の Azure Red Hat OpenShift クラスタで aroworkshop.io コンテンツを使用すると、組織で Azure Red Hat OpenShift の認知度を高めるのに役立ちます。ワークショップ前のディスカッションでは、Azure Red Hat OpenShift が組織の既存の Azure Red Hat OpenShift サブスクリプション内にどのようにデプロイされているか、そして、ワークショップ内でのサービスの使い方は、Azure Red Hat OpenShift を使用してプロダクションのアプリケーションをホストするのと基本的に大きく変わらないことを説明するとよいでしょう。

まとめ

本章では、ワークロードとチームの Azure Red Hat OpenShift へのオンボーディングをスムーズに進めるために役立つチェックリストとガイダンスを提供しました。一般的なアンチパターン、つまり、サポートのない「サンドボックス」クラスタについて説明しました。あらゆる組織で利用できるリソースやチェックリスト項目を 1 冊のガイドで網羅することは難しいため、本章の内容をお客様のニーズに応じて修正することが重要です。

クラウドでは多くのの魅力的なサービスが提供されています。しかし、組織はテクノロジーとそのデプロイに重点を置いているため、多くは見過ごされているか、導入が不十分なままです。これらのトピックを理解することは重要ですが、そのサービスをプロダクションに移行して効率的に導入するとなれば、それらはさまざまな難しい問題のほんの一部にすぎません。本章では、Azure Red Hat OpenShift の導入を成功させるために必要なトレーニングや定期的な確認などにも言及しました。

第 11 章

総まとめ

開発チームと運用チームは、その作業時間のほとんどをクラスタと CI/CD パイプラインのプロビジョニング、セットアップ、保守、および監視に費やさなければならない状態に陥っている場合があります。その場合、彼らは自分たちの貴重な時間を自分たちがが最も得意とすること、すなわち、組織に最先端のアプリケーションをもたらすことに費やすことができません。

このガイドで見てきたように、Azure Red Hat OpenShift を使用すると、実行に必要なインフラストラクチャの構築と管理について心配することなく、フルマネージドの Red Hat OpenShift クラスタをデプロイできます。Kubernetes を単独で実行する場合は、いくつかの留意事項があります。それは主に、いくつものタスクで手動操作が必要になることに関連しており、それらのタスクは Azure Red Hat OpenShift を使えば自動化が可能です。

組織のクラスタ管理戦略を選択する際には、Kubernetes タイプのプラットフォームと、Kubernetes フレームワークに基づいて構築され、すぐに使える役立つ機能をセットにして提供する Azure Red Hat OpenShift を比較して、それぞれがもたらす長所と短所を考慮してください。

Azure Red Hat OpenShift の詳細は、製品ページにアクセスするか、ドキュメントのセクションをご覧ください。また、実践的なワークショップを受講したり、Web セミナーに登録して、都合のよい時に視聴することもできます。そして最も重要なことですが、Azure Red Hat OpenShift の評価を行う際にはマイクロソフトと Red Hat にぜひご連絡ください。できる限りのサポートをさせていただきます。

著者とバージョン

James Read (ジェームズ・リード) <james@redhat.com>
Red Hat プリンシパルソリューションアーキテクト
マイクロソフトに関する内容を担当 (AROv4 向けに更新および改訂)

Ahmed Sabbour (アーメッド・サブール) <asabbour@microsoft.com>
マイクロソフト シニアプロダクトマーケティングマネージャー
Azure Red Hat OpenShift に関する内容を担当 (AROv3 向けの初期バージョン)

前版の著者

Oren Kashi (オーレン・カシ) <okashi@redhat.com>
Red Hat シニアプリンシパルテクニカルプロダクトマーケティングマネージャー

Brooke Jackson、Nermina Miller、Jose Moreno、Ahmed Sabbour、Aditya Datar、Vince Power、Alex Patterson、そして、このガイドのレビューに時間とフィードバックを提供してくれた方々に感謝します。

第 12 章

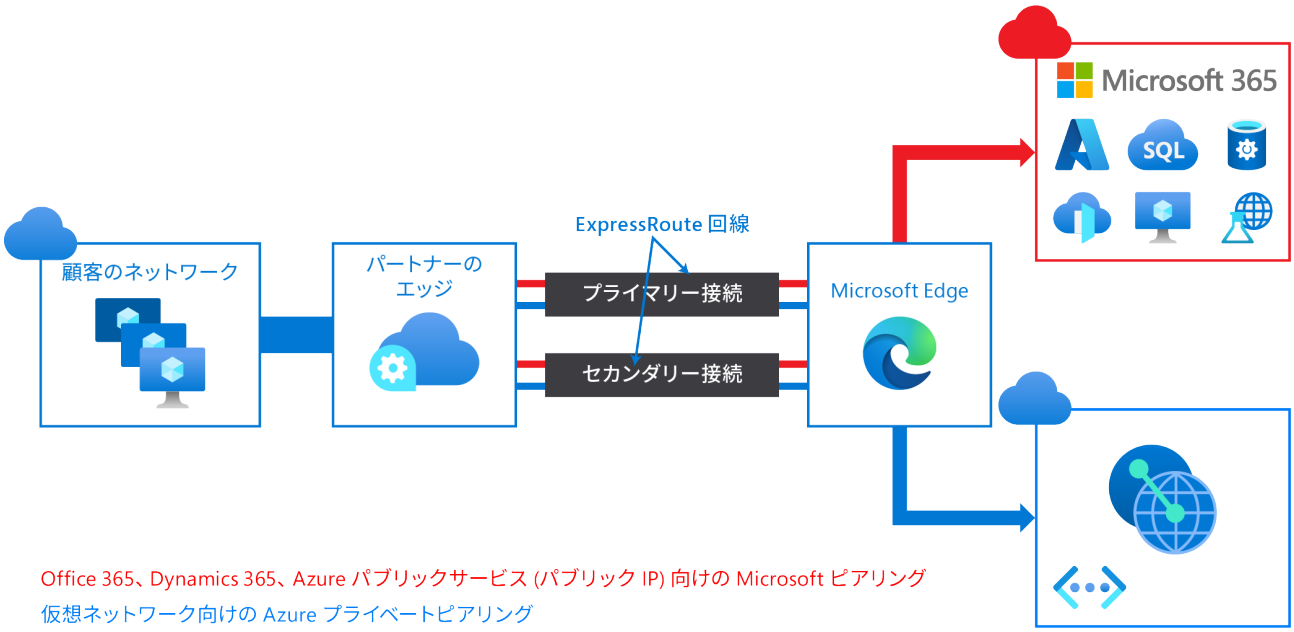
用語集

この用語集は、このガイドおよび Azure Red Hat OpenShift エコシステムで使用される用語の確認に役立つ、簡易的な資料です。見出しは読みの五十音順です。

Azure ExpressRoute

ExpressRoute により、接続プロバイダーを使用してプライベート接続を介し、マイクロソフトのクラウドにオンプレミスネットワークを拡張できます。ExpressRoute を使用すると、Microsoft Azure や Microsoft 365 などのマイクロソフトのクラウドサービスへの接続を確立できます。

以下は、Microsoft Azure のドキュメントから抜粋した ExpressRoute ネットワーク図です。



ExpressRoute の詳細は、Microsoft Azure のドキュメントの ExpressRoute とはのページを参照してください。

Azure での ExpressRoute の機能を理解するには、第 4 章：プロビジョニングの準備 – エンタープライズ・アーキテクチャに関する考慮事項を参照してください。

Azure ランディングゾーン

Azure ランディングゾーンは、Azure を使用した大規模なデプロイメントを計画している組織でよく利用されるデプロイメントパターンで、規模、セキュリティガバナンス、ネットワーキング、および ID が考慮されます。

Azure ランディングゾーンの概要

執筆時点で開発中のコミュニティ GitHub プロジェクトでは、Azure Red Hat OpenShift を Azure ランディングゾーン・アーキテクチャにデプロイする方法について、いくつかの推奨事項が示されています：https://github.com/Azure/Enterprise-Scale/tree/main/workloads/ARO

コンテナ

Azure Red Hat OpenShift アプリケーションの基本的な単位はコンテナと呼ばれています。Linux コンテナ・テクノロジーは、指定されたリソースのみと対話するために実行中のプロセスを分離する軽量なメカニズムです。

数多くのアプリケーション・インスタンスを単一ホストのコンテナで実行でき、かつそれぞれのプロセス、ファイル、ネットワークなどを相互に見えないようにできます。通常、各コンテナは Web サーバーまたはデータベースなどの (通常は「マイクロサービス」と呼ばれることの多い) 単一サービスを提供しますが、任意のワークロードの実行に使用することもできます。

コンテナイメージ

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

これはパッケージ化テクノロジーとして考えることができます。コンテナの作成時にコンテナに追加のアクセス権を付与しない限り、コンテナがアクセスできるのは、イメージで定義されたリソースのみです。同じイメージを複数のホストにまたがって複数のコンテナにデプロイし、それらの間で負荷を分散することにより、Azure Red Hat OpenShift はイメージにパッケージ化されたサービスの冗長性および水平的なスケーリングを提供できます。

コンテナレジストリ

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

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

Azure Red Hat OpenShift は、Docker Hub や Azure Container Registry など、コンテナイメージレジストリ APIを実装している任意のサーバーをイメージのソースとして利用することもできます。

ジョブ

ジョブは、その目的が特定の理由のために Pod を作成することである点で ReplicationController と似ています。違いは、レプリケーション・コントローラーの場合は、継続的に実行されている Pod を対象としていますが、ジョブは 1 回限りの Pod を対象としていることです。ジョブは正常な完了を追跡し、指定された完了数に達すると、ジョブ自体が完了します。

ジョブの使用方法の詳細は、ジョブのトピックを参照してください。

Source-to-Image (S2I)

S2I は、再生可能なコンテナイメージをソースコードから構築するツールキットおよびワークフローです。S2I は、ソースコードをコンテナイメージに挿入し、コンテナがそのソースコードを実行できる状態にして、実行可能なイメージを生成します。セルフアセンブルのビルダーイメージを作成することにより、コンテナイメージを使用してランタイム環境をバージョン管理するのとまったく同じように、ビルド環境のバージョン管理と制御を行うことができます。

Ruby のような動的言語の場合、ビルド時とランタイムの環境は通常同じです。この環境を記述するビルダーイメージ (Ruby、Bundler、Rake、Apache、GCC、およびインストールされた Ruby アプリケーションのセットアップと実行に必要な他のパッケージを使用) から開始して、S2I は次の手順を実行します。

1. 既知のディレクトリに挿入されたアプリケーションソースを使用して、ビルダーイメージからコンテナを起動します。
2. コンテナプロセスは、そのソースコードを適切で実行可能なセットアップに変換します。ここでは、Bundler で依存関係をインストールし、Apache が Ruby `config.ru` ファイルを検索するように事前構成されているディレクトリにソースコードを移動します。
3. 新しいコンテナをコミットし、イメージのエントリーポイントをスクリプト (ビルダーイメージによって提供される) に設定します。このスクリプトは、Apache を起動して Ruby アプリケーションをホストします。

C、Go、Java のようなコンパイル言語の場合、コンパイルに必要な依存関係が実際のランタイム・アーティファクトのサイズを大幅に上回る可能性があります。ランタイムイメージの肥大化を防ぐために、S2I は複数ステップのビルドプロセスを使用します。このプロセスでは、実行可能ファイルや Java WAR ファイルなどのバイナリー・アーティファクトが最初のビルダーイメージで作成され、抽出されて、実行可能ファイルを正しい場所に配置する 2 番目のランタイムイメージに注入されます。

たとえば、Tomcat (一般的な Java Web サーバー) と Maven の再現可能なビルドパイプラインは、次のようにして作成します。

1. WAR ファイルの注入先とする OpenJDK と Tomcat を含むビルダーイメージを作成します。
2. 最初のイメージの上に Maven とその他の標準的な依存関係をレイヤー化する 2 番目のイメージを作成し、Maven プロジェクトの挿入を想定します。
3. Java アプリケーションソースと Maven イメージを使用して S2I を呼び出し、目的のアプリケーション WAR を作成します。
4. 前の手順で作成した WAR ファイルと最初の Tomcat イメージを使用して、S2I をもう一度呼び出し、ランタイムイメージを作成します。

ビルドロジックをイメージ内に配置し、イメージを複数のステップに結合することで、ビルドツールをプロダクションにデプロイしなくても、ランタイム環境をビルド環境 (同じ JDK、同じ Tomcat JAR) に近づけることができます。

ビルドストラテジーとして S2I を使用する目標とメリットは次のとおりです。

- **再現性**：ビルド環境をコンテナイメージ内にカプセル化し、呼び出し元の単純なインタフェース (注入されたソースコード) を定義することで、ビルド環境の厳密なバージョン管理が可能になります。再現性のあるビルドは、セキュリティの更新とコンテナ化されたインフラストラクチャでの継続的インテグレーションを可能にするための重要な要件であり、ビルダーイメージは、再現性とランタイムの交換機能を確保するのに役立ちます。
- **柔軟性**：Linux で実行できる既存のビルドシステムはすべてコンテナ内で実行でき、個々のビルダーはより大きなパイプラインの一部にすることもできます。さらに、アプリケーションのソースコードを処理するスクリプトをビルダーイメージに注入できるため、作成者は既存のイメージを適応させてソース処理を行わせることができます。
- **スピード**：S2I は、単一の Docker ファイルで複数のレイヤーを構築するのではなく、単一のイメージレイヤーを単一のアプリケーションとするよう促します。これにより、作成とデプロイメントにかかる時間が削減され、最終的なイメージの出力をより適切に制御できるようになります。
- **セキュリティ**：Docker ファイルを使用したビルドは、コンテナの通常の操作制御機能の多くを使用することなく実行されます。通常は root として実行され、コンテナネットワークにアクセスできます。ビルドは単一のコンテナで起動されるため、S2I を使用して、ビルダーイメージに利用できるアクセス許可と特権を制御できます。OpenShift などのプラットフォームと連携して S2I を使用すると、管理者はビルド時に開発者が持つ特権を厳密に制御できます。

デプロイメントおよびデプロイメント設定

レプリケーション・コントローラーでビルドする Azure Red Hat OpenShift は、デプロイメントの概念を使用したソフトウェアの開発およびデプロイメント・ライフサイクルの拡張サポートを追加します。最も単純なケースでは、デプロイメントは新規の ReplicationController のみを作成し、それに Pod を起動させます。ただし、Azure Red Hat OpenShift デプロイメントは、イメージの既存デプロイメントから新規デプロイメントに移行する機能を提供し、ReplicationController の作成前後に実行するフックも定義します。

Azure Red Hat OpenShift DeploymentConfig オブジェクトはデプロイメントの以下の詳細を定義します。

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

デプロイヤー Pod は、デプロイメントがトリガーされるたびに、手動または自動であるかを問わず、(古い ReplicationController の縮小、新規の ReplicationController の拡大およびフックの実行などの) デプロイメントを管理します。デプロイメント Pod は、デプロイのログを維持するためにデプロイの完了後無期限で保持されます。デプロイメントが別のものに置き換えられると、以前の ReplicationController が維持されるので、必要に応じてロールバックを容易に実行できます。

デプロイメントの作成およびその対話方法についての詳細は、Deployment と DeploymentConfigを参照してください。

テンプレート

テンプレートは、Azure Red Hat OpenShift で作成されるオブジェクトの一覧を生成するためにパラメーター化し、処理されるオブジェクトのセットを記述できます。テンプレートを処理することで、ユーザーがプロジェクト内で作成する権限を持つあらゆるもの (サービス、ビルド設定、デプロイメント設定など) を作成できます。テンプレートは、テンプレートで定義されるすべてのオブジェクトに適用されるラベルのセットも定義します。

CLI を使用するか、プロジェクトまたはグローバル・テンプレート・ライブラリーにテンプレートがアップロードされている場合には、Web コンソールを使用して、テンプレートからオブジェクト一覧を作成できます。キュレートされたテンプレートについては、OpenShift イメージストリームおよびテンプレートライブラリーを参照してください。

ビルドとイメージストリーム

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

Azure Red Hat OpenShift は、ビルドイメージから Docker 形式のコンテナを作成し、それらをコンテナイメージレジストリにプッシュする際に Kubernetes を使用します。

ビルドオブジェクトには共通の特性があります。そうした特性には、ビルドの入力、ビルドプロセスの完了についての要件、ビルドプロセスのロギング、正常なビルドからのリソースのパブリッシュ、およびビルドの最終ステータスのパブリッシュなどがあります。ビルドはリソースの制限を利用し、CPU 使用、メモリー使用およびビルドまたは Pod の実行時間などのリソースの制限を指定します。

Azure Red Hat OpenShift ビルドシステムはビルドストラテジーをサポートしており、サポート対象は拡張可能です。ビルドストラテジーはビルド API でタイプを選択して指定します。利用可能なビルドストラテジーは主に 3 つあります。

- **Docker ビルド** – ソースリポジトリからアップロードまたはプルできる Dockerfile を使用します
- **Source-to-Image (S2I) ビルド** – ソースリポジトリ (Git など) を取得し、よく知られた言語ビルドファイル (Java プロジェクトの Maven .pom ファイルなど) からアプリケーションをビルドする方法を決定します
- **カスタムビルド**

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

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

プロジェクトとユーザー

プロジェクトは追加のアノテーションを持つ Kubernetes 名前空間であり、通常ユーザーのリソースへのアクセスを管理するための中心的な手段です。プロジェクトにより、ユーザーのコミュニティが他のコミュニティとは切り離してコンテンツを編成し、管理できます。ユーザーには、管理者によってプロジェクトへのアクセスが付与される必要があり、許可される場合はプロジェクトを作成でき、それらの独自のプロジェクトへのアクセスが自動的に付与されます。プロジェクトには、別個の name、`displayName`、および description を設定できます。

必須の name はプロジェクトの一意の ID であり、CLI ツールまたは API で最もよく使用されます。名前の長さは最大で 63 文字です。オプションの `displayName` は、Web コンソールでのプロジェクトの表示方法を示します (デフォルトは `name`に設定される)。オプションの description には、プロジェクトのさらに詳細な記述を使用でき、これも Web コンソールで表示できます。

開発者および管理者は、CLI または Web コンソールを使用してプロジェクトを操作できます。

Pod とサービス

Azure Red Hat OpenShift は、Kubernetes の Pod の概念 (1 つのホスト上にデプロイされる 1 つ以上のコンテナ) を使用しています。Pod は、Azure Red Hat OpenShift で定義され、デプロイされ、管理される最小のコンピュート単位です。

Pod はコンテナに対してマシンインスタンス (物理または仮想) とほぼ同じ機能を持ちます。各 Pod は独自の内部 IP アドレスを割り当てられるため、そのポート空間全体を所有しており、Pod 内のコンテナはローカルストレージおよびネットワークを共有できます。

Pod にはライフサイクルがあります。それらは定義された後にノードで実行されるために割り当てられ、コンテナが終了するまで実行されるか、その他の理由でコンテナが削除されるまで実行されます。ポリシーおよび終了コードによっては、Pod は終了後に削除されるか、コンテナのログへのアクセスを有効にするために保持される可能性があります。

Azure Red Hat OpenShift は Pod をほとんどイミュータブルなものとして処理します。Pod が実行中の場合は Pod に変更を加えることができません。Azure Red Hat OpenShift は既存 Pod を終了し、これを変更された設定、ベースイメージのいずれかまたはその両方で再作成して変更を実装します。Pod のランタイムコンポーネントは消費可能なものとして扱われ、定義されたコンテナイメージから再作成されます。そのため、通常 Pod はユーザーが直接管理するのでははく、より高次のレベルのコントローラーで管理します。

ルートと Ingress

Azure Red Hat OpenShift は、ルートと Ingress の両方をサポートします。どちらも www.example.com のような DNS 名を使用してサービスを公開し、外部のクライアントがサービスにアクセスできるようにするために使用されます。

ルートは元々 Red Hat OpenShift バージョン 3 で考案されました。そのリリース以来、Red Hat は Kubernetes コミュニティと連携して、現在 **Ingress** と呼ばれているものにおいてこの機能を標準化しました。

OpenShift Container Platform の Kubernetes Ingress リソースは、クラスタ内で Pod として実行される共有ルーターサービスを使用して Ingress Controller を実装します。Ingress トラフィックを管理する最も一般的な方法は、Ingress Controller を使用することです。この Pod は、他の通常の Pod と同じようにスケーリングおよび複製できます。このルーターサービスは、オープンソースのロードバランサー・ソリューションである HAProxy に基づいています。

OpenShift Container Platform のルートは、クラスタ内のサービスへの Ingress トラフィックを提供します。ルートは、TLS 再暗号化、TLS パススルー、ブルーグリーン・デプロイメントの分割トラフィックなど、標準の Kubernetes Ingress Controller ではサポートされない可能性のある高度な機能を提供します。

Ingress トラフィックは、ルートを介してクラスタ内のサービスにアクセスします。ルートと Ingress は、Ingress トラフィックを処理するための主要なリソースです。Ingress は、外部の要求を受け入れたり、ルートに基づいてそれらを委任したりするなど、ルートと同様の機能を提供します。ただし、Ingress で許可できるのは、特定の種類の接続 (HTTP/2、HTTPS と**サーバー名表示 (SNI)**、および証明書を使用した TLS) のみです。OpenShift Container Platform では、Ingress リソースで指定された条件を満たすようにルートが生成されます。

ReplicaSet

ReplicationController と同様に、ReplicaSet は、指定された数の Pod レプリカが常に実行されるようにします。ReplicaSet と ReplicationController の相違点は、ReplicaSet ではセットベースのセレクター要件をサポートし、ReplicationController は等価ベースのセレクター要件のみをサポートする点です。

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
    name: frontend-1
    labels:
        tier: frontend
spec:
    replicas: 3
    selector:
        matchLabels:
            tier: frontend
        matchExpressions:
        - {key: tier, operator: In, values: [frontend]}
    template:
        metadata:
            labels:
                tier: frontend
        spec:
            containers:
                - image: openshift/hello-openshift
                  name : helloworld
                  ports:
                  - containerPort: 8080
                    protocol: TCP
            restartPolicy: Always
```

上記のコードでは、次のことを確認できます。

- 一連のリソースに対するラベルのクエリ。matchLabels と matchExpressions の結果は論理的に結合されています。
- セレクターと一致するラベルでリソースを指定するための等式ベースのセレクター。
- キーをフィルタリングするセットベースのセレクター。これは、key が tier、value が frontend のリソースをすべて選択します。

ReplicationController

ReplicationController により、指定された数の Pod のレプリカが常に実行されます。Pod の終了または削除が行われた場合、ReplicationController は定義した数になるまでインスタンスを増やします。同様に、必要以上の数の Pod が実行されている場合には、定義された数に一致させるために必要な数の Pod を削除します。

ReplicationController 設定は以下で構成されています。

- 必要なレプリカ数 (ランタイム時に調整可能)。
- レプリケートされた Pod の作成時に使用する Pod 定義。
- 管理された Pod を識別するためのセレクター。
- セレクターは、ReplicationController が管理する Pod に割り当てられるラベルセットです。これらのラベルは、Pod 定義に組み込まれ、ReplicationController がインスタンス化します。ReplicationController は、必要に応じて調節するために、セレクターを使用して、すでに実行中の Pod 数を判断します。

ReplicationController は負荷やトラフィックに基づいて自動スケーリングを実行することはなく、追跡も行いません。このような処理を行う場合は、レプリカ数を外部の自動スケーラーで調整する必要があります。

ReplicationController は、Kubernetes のコアオブジェクトです。以下は、ReplicationController 定義の例です。

```
apiVersion: v1
kind: ReplicationController
metadata:
    name: frontend-1
spec:
    replicas: 1
    selector:
        name: frontend
    template:
        metadata:
            labels:
                name: frontend
        spec:
            containers:
                - image: openshift/hello-openshift
                  name: helloworld
                  ports:
                - containerPort: 8080
                  protocol: TCP
            restartPolicy: Always
```

上記のコードでは、次のことを確認できます。

- 実行する Pod のコピーの数。
- 実行する Pod のラベルセレクター。
- コントローラーが作成する Pod のテンプレート。
- Pod のラベルには、ラベルセレクターのラベルを含める必要があります。
- パラメーターを展開した後の名前の長さは最大で 63 文字です。