# Red Hat Ansible Automation Platform 2.4

## Red Hat OpenShift への Ansible Automation Platform 2 のデプロイ

Last Updated: 2023-11-16

# Red Hat Ansible Automation Platform 2.4 Red Hat OpenShift への Ansible Automation Platform 2 のデプロイ

Roger Lopez
ansible-feedback@redhat.com

## 法律上の通知



## 概要

このドキュメントでは、Red Hat OpenShift に Ansible Automation Platform 2 をデプロイするためのベストプラクティスを提供します。

# 目次

**コメントとフィードバック** ..... 3

**第1章 概要** ..... 4

**第2章 RED HAT OPENSHIFT で ANSIBLE AUTOMATION PLATFORM を使用する理由** ..... 6

**第3章 作業を開始する前の注意事項** ..... 7
3.1. POD とコンテナーのリソース管理 7
3.2. AUTOMATION CONTROLLER POD コンテナーの推奨サイズ 8
3.3. POSTGRES POD の推奨サイズ 10
3.4. 自動化ジョブ POD の推奨サイズ 11
3.5. AUTOMATION CONTROLLER の POD サイズの推奨事項の概要 13
3.6. AUTOMATION HUB POD の推奨サイズ 13
3.7. AUTOMATION CONTROLLER POD の専用ノードの指定 15
3.8. データベースの高可用性の処理 16

**第4章 前提条件** ..... 18

**第5章 ANSIBLE AUTOMATION PLATFORM OPERATOR のインストール** ..... 19

**第6章 AUTOMATION CONTROLLER のインストール** ..... 21

**第7章 AUTOMATION HUB のインストール** ..... 23

**第8章 AUTOMATION CONTROLLER ダッシュボードへのログイン** ..... 25

**第9章 AUTOMATION HUB ダッシュボードへのログイン** ..... 26

**第10章 ANSIBLE AUTOMATION PLATFORM のモニタリング** ..... 27
10.1. API メトリック監視に使用するもの 27
10.2. 得られるメトリクス 27
10.3. ANSIBLE PLAYBOOK によるインストール 27

**付録A 著者について** ..... 30

**付録B 以前の AAP インストールからの既存の PVC の削除** ..... 31

**付録C RED HAT OPENSHIFT ノードへのラベルとテイントの適用** ..... 32

**付録D AMAZON S3 バケットの作成** ..... 33

**付録E AWS S3 シークレットの作成** ..... 34

**付録F 管理された名前空間への AAP OPERATOR の追加** ..... 35

**付録G 参考資料** ..... 36

**付録H REVISION HISTORY** ..... 37

# コメントとフィードバック

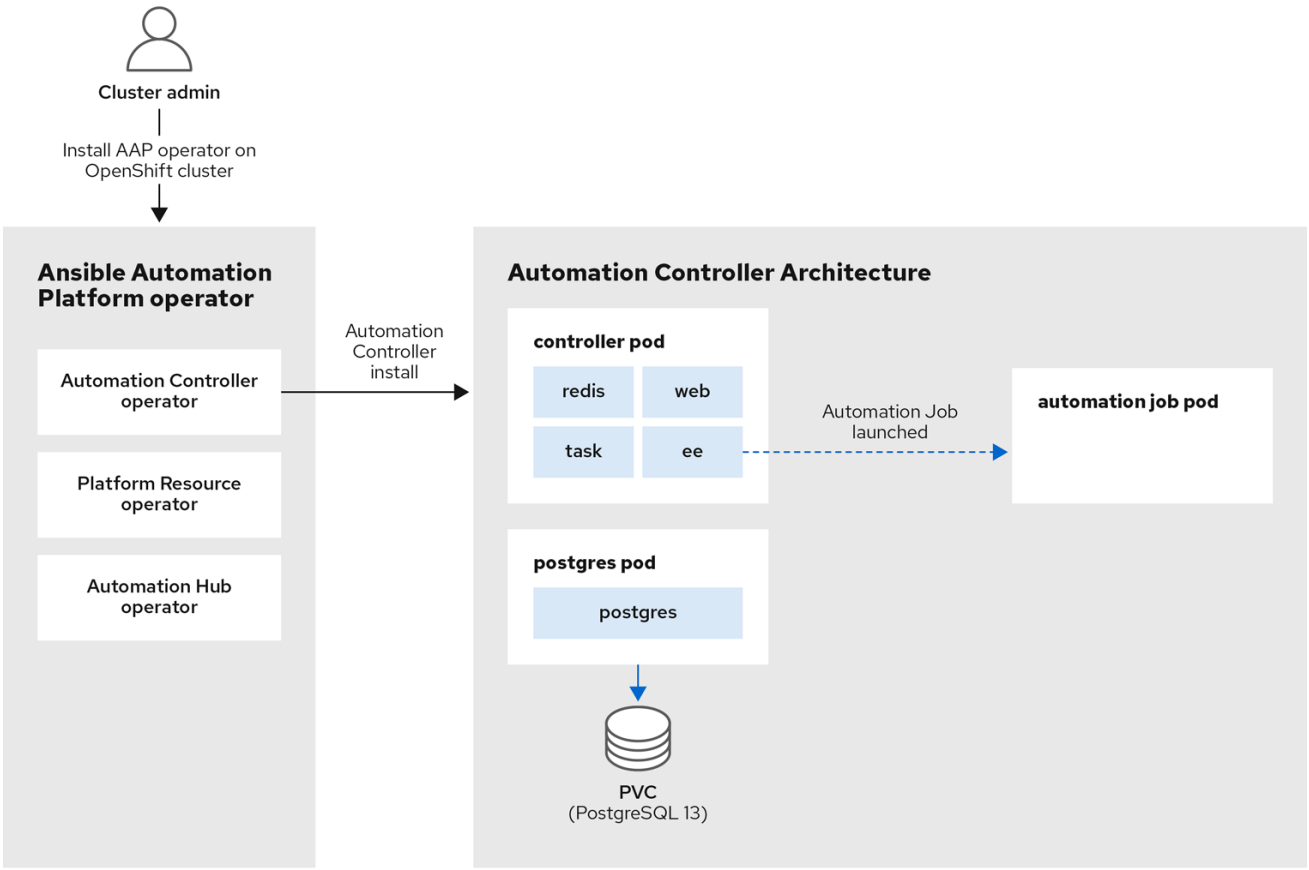
オープンソースの精神を大切にしています。リファレンスアーキテクチャーに関するフィードバックやコメントをお聞かせください。文書は社内で確認していますが、何らかの問題や誤字脱字があるかもしれません。フードバックにより、当社は作成文書の品質を向上でき、読者も改善点や内容の拡充などについてご意見いただけます。文書に関するフィードバックは、電子メールで ansible-feedback@redhat.com に送信してください。その際には、メール本文内で文書のタイトルを記載してください。

# 第1章 概要

Red Hat OpenShift リファレンスアーキテクチャーの Ansible Automation Platform (AAP) 2.3 は、Ansible Automation Platform 環境をデプロイする独自のセットアップを提供します。また、Ansible Automation Platform 2.3 をインストールおよび設定するための最新のベストプラクティスを含む段階的な展開手順について説明しています。これは、Red Hat OpenShift への Ansible Automation Platform のデプロイを検討しているシステムおよびプラットフォーム管理者に最適です。

Red Hat OpenShift の機能を活用して、Ansible Automation Platform のデプロイを合理化し、セットアップに必要な時間と労力を大幅に削減できます。

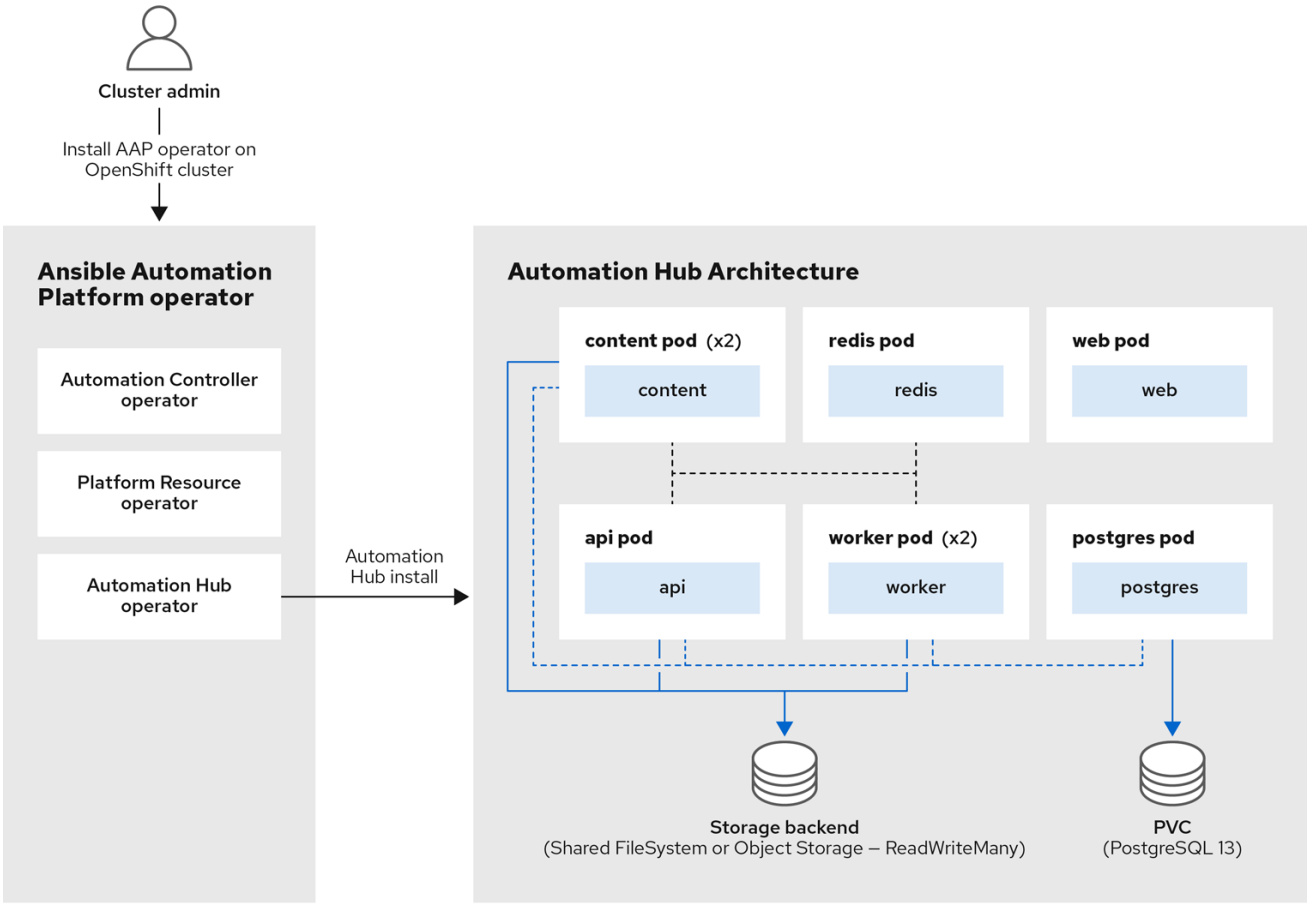
**図1.1 Automation Controller のアーキテクチャー**




280_Ansible_0323

図1.1「Automation Controller のアーキテクチャー」 は、Automation Controller コンポーネントをデプロイする Ansible Automation Platform (AAP) Operator のデプロイプロセスフローを示しています。大規模な Ansible Automation Platform Operator を設定する 3 つの Operator の 1 つである Automation Controller Operator は、コントローラー、postgres、および自動化ジョブ Pod など、さまざまな Pod のデプロイを行います。

図1.2 Automation Hub のアーキテクチャー




同様に、図1.2「Automation Hub のアーキテクチャー」 Automation Hub コンポーネントをデプロイする AAP Operator を示します。Automation Hub Operator は、相互に通信するさまざまな Pod をデプロイして Automation Hub を提供し、内部で生成されたコンテンツ、Red Hat Ansible 認定コンテンツ、実行環境、および Ansible 検証済みコンテンツをチームと共有します。

さらに、このリファレンスアーキテクチャーでは、あらゆる自動化の取り組みを行えるように、強固な基盤で提供する効率的でスケーラブルな環境を提供するにあたり主要な手順について説明します。

# 第2章 RED HAT OPENSHIFT で ANSIBLE AUTOMATION PLATFORM を使用する理由

Red Hat OpenShift 4.x で Ansible Automation Platform 2.3 を実行すると、プラットフォームをデプロイおよび管理するためのより効率的で自動化された方法が提供されるだけでなく、Red Hat OpenShift の組み込みの監視、ロギング、セキュリティー、およびスケーラビリティ機能との統合が向上します。

さらに、Red Hat OpenShift の Operator Lifecycle Manager (OLM) を使用して Ansible Automation Platform Operator をデプロイおよび管理すると、更新とアップグレードのプロセスが簡素化され、制御と柔軟性が向上します。

これらの利点の詳細な内訳を以下に示します。

- **自動化:** Ansible Automation Platform Operator は、Red Hat OpenShift での Ansible Automation Platform のデプロイと管理を自動化する方法を提供します。自動化により、プラットフォームのデプロイメントと管理に必要な時間と労力を削減して、一貫性があり、予測可能な方法で Ansible Automation Platform を実行できます。

- **スケーラビリティー:** Operator は、組織のニーズに合わせて Ansible Automation Platform をスケーリングするのに役立ちます。プラットフォームの複数のインスタンスを簡単にデプロイメントし、それらを 1 つの場所から管理できます。

- **柔軟性:** Operator は、Ansible Automation Platform をデプロイおよび管理するための柔軟な方法を提供します。特定のニーズに合わせてプラットフォームの設定をカスタマイズし、開発、ステージング、実稼働などのさまざまな環境にデプロイできます。

- **監視とトラブルシューティング:** Operator は、Prometheus や Grafana などの Red Hat OpenShift の組み込みの監視およびログツールと統合して、プラットフォームのリソース使用状況を監視し、ボトルネックを特定するために使用できます。

- **管理とアップグレード:** Red Hat OpenShift 4.x に付属の Operator Lifecycle Manager (OLM) を使用して、簡単にデプロイ、管理、およびアップグレードできます。[1] クラスター全体の Ansible Automation Platform Operator 。

[1] Ansible Automation Platform でサポートされるライフサイクルバージョン - https://access.redhat.com/support/policy/updates/ansible-automation-platform

# 第3章 作業を開始する前の注意事項

AAP を Red Hat OpenShift にデプロイする前に、インストール前に対処する必要がある重要な考慮事項を理解することが重要です。これらの要因により、AAP 環境のライフサイクル全体の正常性とスケーラビリティーが決まります。

このセクションでは、次のような重要な考慮事項の内訳を示します。

- Red Hat OpenShift リソース管理
- Automation controller Pod コンテナーのサイジングに関する推奨事項
- Postgres Pod のサイジングに関する推奨事項
- 自動化ジョブ Pod のサイジングに関する推奨事項
- Automation Hub Pod のサイジングに関する推奨事項

デプロイを成功させるための重要な側面の 1 つとして、Pod とコンテナーの適切なリソース管理が挙げられます。適切にリソース管理を行うことで Red Hat OpenShift クラスターの AAP アプリケーションの最適なパフォーマンスと可用性を確保します。

## 3.1. POD とコンテナーのリソース管理

リソース管理に関する 2 つの重要なリソースは、CPU とメモリー (RAM) です。Red Hat OpenShift は、リソース要求とリソース制限を使用して、コンテナーが Pod で消費できるリソースの量を制御します。

### 3.1.1. リソース要求とは

リソース要求は、コンテナーが適切に実行および機能するために最小限必要なリソースの量です。Kubernetes スケジューラーは、この値を使用して、コンテナーに使用できる十分なリソースがあることを確認します。

### 3.1.2. リソース制限とは

一方、リソース制限は、コンテナーが最大限消費できるリソースの量です。リソース制限を設定すると、コンテナーが必要以上のリソースを消費することがなくなり、他のコンテナーがリソース不足に陥る可能性があります。

### 3.1.3. リソース管理が重要な理由

AAP に関しては、正しいリソースリクエストと制限を設定することが重要です。リソースの割り当てが不十分な場合、制御 Pod が終了し、Automation controller 内のすべての自動化ジョブが失われる可能性があります。

### 3.1.4. リソースの計画

適切なリソース管理値を設定する一方で、組織は利用可能なリソースに基づいて、どのアーキテクチャーがニーズに最も適しているかを検討する必要があります。たとえば、自動化ジョブを実行する容量を最大化することよりも、Ansible Automation Platform 環境の高可用性が重要かどうかを判断します。

このリファレンスアーキテクチャーで使用されている既存の Red Hat OpenShift 環境を取り上げて、詳細に説明します。設定は次のとおりです。

- 3 コントロールプレーンノード
- 3 つのワーカーノード

これらの各ノードは、4 つの vCPU と 16 GiB の RAM で設定されています。

Red Hat OpenShift クラスターのコントロールプレーンノードはアプリケーションを実行しないため、この例では使用可能な 3 つのワーカーノードに焦点を当てています。

これら 3 つのワーカーノードを使用して、Ansible Automation Platform の可用性を最大化するか、できるだけ多くの自動化ジョブを実行するか、あるいはその両方か、どちらがより重要であるかを判断する必要があります。

可用性が最も重要な場合は、2 つの制御 Pod が別々のワーカーノード (例: **worker0** と **worker1**) で実行され、すべての自動化ジョブが残りのワーカーノード (例: **worker2**) 内で実行されるようにすることに重点が置かれます。

ただし、これにより、自動化ジョブの実行に使用できるリソースが半分に減少します。推奨の方法として、制御 Pod と自動化 Pod を分離して、同じワーカーノードで実行されないようにします。

実行する自動化ジョブの量を最大化することが主な目標である場合、制御 Pod に 1 つのワーカーノード (例: **worker0**) を使用し、残りの 2 つのワーカーノード (例: **worker1** と **worker2**) を自動化ジョブの実行に使用すると、使用可能なリソースが 2 倍になります。ジョブの実行はできますが、代わりに制御 Pod に冗長性がなくなります。

当然、どちらも同様に重要である場合があります。そういった場合には、解決策として、両方の要件を満たすには、追加のリソース (ワーカーノードを追加するなど) が必要になる場合があります。

## 3.2. AUTOMATION CONTROLLER POD コンテナーの推奨サイズ

概要セクションの 図1.1「Automation Controller のアーキテクチャー」 を確認すると、制御 Pod にコンテナーが 4 台含まれていることがわかります。

- web
- ee
- redis
- task

これらのコンテナーはそれぞれ、Ansible Automation コントローラーで独自の機能を実行するため、リソース設定が制御 Pod にどのように影響するかを理解することが重要です。デフォルトでは、Red Hat OpenShift にはテストとしてインストールする最低条件を満たす程度の低いリソース値が含まれていますが、実稼働環境で Ansible Automation Platform を実行するには最適ではありません。

Ansible Automation Platform の Red Hat OpenShift のデフォルトは次のとおりです。

- CPU: 100m
- メモリー: 128Mi

デフォルトでは、Red Hat OpenShift は最大リソース制限を設定せず、Ansible Automation Platform 制御 Pod によって要求されたすべての可能なリソースを割り当てようとします。この設定は、リソース

の枯渇を引き起こし、Red Hat OpenShift クラスターで実行されている他のアプリケーションに影響を与える可能性があります。

制御 Pod 内のコンテナーのリソースリクエストと制限に関するスタート地点を示すため、以下の前提条件を使用します。

- それぞれ 4 つの vCPU と 16GiB RAM を備えた Red Hat OpenShift クラスター内で使用可能なワーカーノード 3 つ
- 高可用性よりも、自動化ジョブのリソースを最大化することが重要である
- Automation controller を実行するための専用ワーカーノード 1 つ
- 自動化ジョブを実行するための残りのワーカーノード 2 つ

制御 Pod 内のコンテナーのサイジングに関しては、ワークロードの詳細を考慮することが重要です。このリファレンス環境に固有の推奨事項に沿ったパフォーマンステストを実施しましたが、これらの推奨事項はすべてのタイプのワークロードに適用できるわけではありません。

スタート地点として、Performance Collection playbooks (具体的には chatty_tasks.yml) を利用することにしました。

パフォーマンスベンチマークは、以下のとおりです。

- ホスト 1 台でインベントリーを作成する
- **chatty_tasks.yml** ファイルを実行するジョブテンプレートを作成する

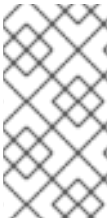
**chatty_tasks** ジョブテンプレートは、**ansible.builtin.debug** モジュールを利用して、ホストごとに設定された数のデバッグメッセージを生成し、必要なインベントリーを生成します。**ansible.builtin.debug** モジュールを利用することで、追加のオーバーヘッドを導入することなく、Automation controller のパフォーマンスを正確に表現できます。

ジョブテンプレートは、ジョブテンプレートの同時呼び出しの数を示す 10 ~ 50 の範囲の指定された同時実行レベルで実行されました。

以下に記載されている **リソース要求** および **リソース制限** は、パフォーマンスベンチマークの結果で、同様のリソースを使用する Red Hat OpenShift クラスターで AAP を実行する際の開始ベースラインとして使用できます。

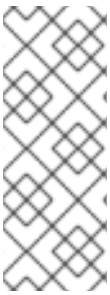
```
spec:
...
  ee_resource_requirements:
    limits:
      cpu: 500m
      memory: 400Mi
    requests:
      cpu: 100m
      memory: 400Mi
  task_resource_requirements:
    limits:
      cpu: 4000m
      memory: 8Gi
    requests:
      cpu: 1000m
      memory: 8Gi
  web_resource_requirements:
```

```
    limits:
      cpu: 2000m
      memory: 1.5Gi
    requests:
      cpu: 500m
      memory: 1.5Gi
  redis_resource_requirements:
    limits:
      cpu: 500m
      memory: 1.5Gi
    requests:
      cpu: 250m
      memory: 1.5Gi
```



**注記**

メモリーリソースの要求と制限を一致させ、Pod の Out Of Memory (OOM) Kill を引き起こす可能性がある Red Hat OpenShift クラスター内のメモリーリソースを過剰に使用しないようにします。リソース制限がリソース要求よりも大きい場合、Red Hat OpenShift ノードの過剰使用を許可するシナリオが発生する可能性があります。



**注記**

CPU リソースは圧縮可能とされているので、CPU リソースの要求と制限はメモリーとは異なります。つまり、Red Hat OpenShift は、リソース制限に達するとコンテナーの CPU を処理しようとしますが、コンテナーは終了しません。制御 Pod 内の上記のコンテナーでは、指定のワークロードに対しては十分な量の CPU 要求が提示されましたが、しきい値 (CPU 制限) をより高い値に設定して、負荷がかかっている状態で高い値でバーストできるようにしました。

**警告**

上記のシナリオでは、専用の Red Hat OpenShift ワーカーノードを使用するため、制御 Pod が存在するワーカーノード内のリソースを他のアプリケーションが使用していないことを前提としています。詳細は 「Automation controller Pod の専用ノードの指定」 を参照してください。

## 3.3. POSTGRES POD の推奨サイズ

**chatty_task** Playbook を使用してパフォーマンスベンチマークテストを実施した後、CPU リソース要求が 500m 未満になると、リソース制限よりも低いものの、初期リソース要求よりも多い場合には、追加のリソースが Pod に対して確保することが保証されないので、Postgres Pod で CPU スロットリングを引き起こす可能性があります。ただし、テスト中に 500m 要求を超えるバーストがあったため、CPU 制限は 1000m (1 vCPU) に設定されました。

メモリーに関しては、メモリーは圧縮可能なリソースではないため、**chatty_task** パフォーマンステスト中に、テストの最高レベルの Postgres Pod が 650Mi をわずかに超える RAM を消費したことがわかります。

したがって、このような結果をもとにすると、このリファレンス環境で推奨されるメモリーリソースの要求および制限は1Gi で、この程度あれば、十分なバッファーを提供し、Postgres Pod で Out Of Memory (OOM) Kill が発生する可能性を回避します。

以下に記載されている **リソース要求** および **リソース制限** は、パフォーマンスベンチマークの結果で、Postgres Pod を実行する際の開始ベースラインとして使用できます。

```
spec:
...
  postgres_resource_requirements:
    limits:
      cpu: 1000m
      memory: 1Gi
    requests:
      cpu: 500m
      memory: 1Gi
```

**警告**

以下の値は、このリファレンス環境に固有のものであり、実際のワークロードに対して不十分な場合があります。Postgres Pod のパフォーマンスを監視し、パフォーマンスのニーズを満たすようにリソースの割り当てを調整することが重要です。

## 3.4. 自動化ジョブ POD の推奨サイズ

Ansible Automation Platform ジョブは、ホストのインベントリーに対して Ansible Playbook を起動する Automation controller のインスタンスです。Ansible Automation Platform が Red Hat OpenShift で実行される場合、デフォルトの実行キューは、インストール時にOperator によって作成されたコンテナーグループです。

コンテナーグループは、Kubernetes 認証情報とデフォルトの Pod 仕様で設定されます。コンテナーグループに対してジョブが起動されると、Automation controller によって、コンテナーグループ Pod 仕様で指定された名前空間に Pod が作成されます。これらの Pod は、自動化ジョブ Pod と呼ばれます。

自動化ジョブ Pod の適切なサイズを決定するには、まず、Automation controller コントロールプレーンが同時に起動できるジョブ数の機能を理解する必要があります。

この例では、3 つのワーカーノードがあります (それぞれ 4 つの vCPU と 16GiB の RAM)。1 つのワーカーノードは制御 Pod をホストし、他の 2 つのワーカーノードは自動化ジョブに使用されます。

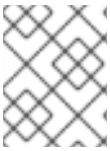
これらの値に基づいて、Automation controller コントロールプレーンが実行できる制御容量を決定できます。

次の式で、内訳がわかります。

合計制御容量 = 合計メモリー (MB)/フォークサイズ (MB)

ワーカーノードに基づいて、これは次のように表現できます。

総制御容量 = 16,000 MB/100 MB = 160



**注記**

この計算の詳細を確認するには、Resource Determination for Capacity Algorithm を参照してください。

これは、Automation controller が 160 個のジョブを同時に起動するように設定されていることを意味します。ただし、後ほど説明するコンテナーグループ/実行プレーンの容量に合わせて、この値を調整する必要があります。

**注記**

16GB は 16,000 MB に丸められ、1 つのフォークのサイズはデフォルトで 100MB とし、簡素化しています。

使用可能な制御容量を計算したので、同時自動化ジョブの最大数を決定できます。

これを判断するには、コンテナーグループ/実行プレーン内の自動化ジョブ Pod の仕様に、vCPU 250m および 100Mi の RAM のデフォルトの要求があることに注意する必要があります。

1 つのワーカーノードの合計メモリーの使用:

16,000 MB/100 MiB = 160 の同時ジョブ

1 つのワーカーノードの合計 CPU の使用:

4000 ミリ cpu / 250 ミリ cpu = 16 の同時ジョブ

上記の値に基づいて、ノード上の最大同時ジョブ数を 2 つの同時ジョブ値の最小値である 16 に設定する必要があります。この例では、自動化ジョブの実行に 2 つのワーカーノードが割り当てられているため、この数は 2 倍の 32 (ワーカーノードあたり 16 の同時ジョブ) になります。

Automation controller の設定は現在 160 の同時ジョブに設定されており、使用可能なワーカーノードの容量では 32 の同時ジョブしか許可されていません。数が偏っているため、これは問題です。

つまり、Automation controller のコントロールプレーンは、同時に 160 のジョブを起動できると認識しているにも拘らず、Kubernetes スケジューラーでは、コンテナーグループの名前空間で最大 32個の自動化ジョブ Pod しか同時にスケジュールされていません。

コントロールプレーンとコンテナーグループ/実行プレーンの間の値が不均衡であると、次のような問題が発生する可能性があります。

- コントロールプレーンの容量が、コンテナーグループでスケジュール可能な同時ジョブ Pod の最大数よりも多い場合、コントロールプレーンは、開始する Pod を送信することによって、ジョブの開始を試みます。ただし、これらの Pod は、リソースが利用可能になるまで実際に実行を開始しません。ジョブ Pod が **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** のタイムアウト内に開始されない場合、ジョブは中止されます (デフォルトは 2 時間)。

- コントロールプレーンが起動できるとみなす、同時自動化ジョブの数よりも多く、コンテナーグループがサポートできる場合に、Automation controller はコンテナーグループが最大サポート可能な同時自動化ジョブ数に到達するまで自動化ジョブを起動しないので、この容量は無駄になります。

中止されたジョブや未使用のリソースのリスクを回避するために、有効な制御容量と、デフォルトのコンテナーグループがサポートできる同時実行ジョブの最大数とのバランスを取ることを推奨します。

コントロールプレーンが起動するジョブの最大数は **AWX_CONTROL_NODE_TASK_IMPACT** と呼ば

れる設定の影響を受けるため、有効な制御容量という用語が使用されます。**AWX_CONTROL_NODE_TASK_IMPACT** 変数は、自動化ジョブごとに制御 Pod で使用可能な容量を定義し、実質的に、制御 Pod が起動を試すことのできる自動化ジョブ数を制御しています。

有効な制御能力と、利用可能な実行能力のバランスをとるには、**AWX_CONTROL_NODE_TASK_IMPACT** 変数を、Automation Controller コントロールプレーンで実行する同時ジョブ数を制限する値に設定して、コンテナーグループ/実行プレーンで起動される自動化ジョブ Pod 数と同じ数にします。

コンテナーグループがサポートできるよりも多くの同時自動化ジョブの起動を避けるために、**AWX_CONTROL_NODE_TASK_IMPACT** の最適値を計算するには、次の式を使用できます。

**AWX_CONTROL_NODE_TASK_IMPACT** = 制御容量/コンテナーグループが起動できる最大同時ジョブ

リファレンス環境の場合、これは次のとおりです。

**AWX_CONTROL_NODE_TASK_IMPACT** = 160 / 32 = 5

結論として、このリファレンス環境では、**AWX_CONTROL_NODE_TASK_IMPACT は** 5 に等しくなるはずである事がわかります。この値は 6章*Automation controller のインストール* の章の **extra_setting** で設定します。これについては、このドキュメントの後半で説明します。

## 3.5. AUTOMATION CONTROLLER の POD サイズの推奨事項の概要

コントロールプレーン (制御 Pod) とコンテナーグループ/実行プレーン (自動化ジョブ Pod) のリソース要求と制限を適切に設定することは、制御と実行の容量のバランスを取るために必要です。正しい設定は、次の方法で判断できます。

- 制御容量を計算する
- 同時に実行できる自動化ジョブの数を計算する
- Automation controller のインストール内で適切なバランス値を使用して **AWX_CONTROL_NODE_TASK_IMPACT** 変数を設定する

## 3.6. AUTOMATION HUB POD の推奨サイズ

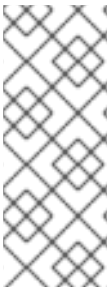
概要セクションの 図1.2「Automation Hub のアーキテクチャー」 にあるように、デプロイは 7 つの Pod で設定され、それぞれがコンテナーをホストしていることがわかります。

Pod のリストには、以下が含まれます。

- コンテンツ (x2)
- redis
- api
- postgres
- worker (x2)

Automation Hub アーキテクチャーを設定する 7 つの Pod は連携して機能し、コンテンツを効率的に管理および配布します。これらは、Automation Hub 環境の全体的なパフォーマンスとスケーラビリティにとって重要です。

このような Pod の中でも、ワーカー Pod はコンテンツの処理、同期、および配布を行うので、特に重要です。このような理由から、適切な量のリソースをワーカー Pod に設定して、確実にタスクを実行できるようにすることが重要です。



**注記**

以下は、Automation Hub 環境に必要なリソース要求と制限の見積もりを提供することを目的としたガイドラインです。実際に必要なリソースは、セットアップによって異なります。

たとえば、頻繁に更新または同期を実行しているリポジトリーが多数ある環境では、処理負荷を処理するためにより多くのリソースが必要になる場合があります。

このリファレンス環境では、Pod のサイズを決定するために、Automation Hub 環境で実行できるメモリー消費量が最も多いタスクの 1 つであるリモートリポジトリーの同期を使用して予備テストが行われました。

調査結果から、Automation Hub 内のリモートリポジトリーを正常に同期するには、次のリソースリクエストとリソース制限を各 Pod に設定する必要があることがわかりました。

```
spec:
...
content:
  resource_requirements:
    limits:
      cpu: 250mm
      memory: 400Mi
    requests:
      cpu: 100m
      memory: 400Mi

redis:
  resource_requirements:
    limits:
      cpu: 250m
      memory: 200Mi
    requests:
      cpu: 100m
      memory: 200Mi

api:
  resource_requirements:
    limits:
      cpu: 250m
      memory: 400Mi
    requests:
      cpu: 150m
      memory: 400Mi

postgres_resource_requirements:
  resource_requirements:
    limits:
      cpu: 500m
      memory: 1Gi
    requests:
```

```
      cpu: 200m
      memory: 1Gi

worker:
  resource_requirements:
    limits:
      cpu: 1000m
      memory: 3Gi
    requests:
      cpu: 400m
      memory: 3Gi
```

## 3.7. AUTOMATION CONTROLLER POD の専用ノードの指定

専用ノードで制御 Pod を実行することは、制御 Pod と自動化ジョブ Pod を分離し、これら 2 種類の Pod 間のリソース競合を防ぐために重要です。このように分離することで、リソースの制約が原因となるサービスの低下リスクなしに、制御 Pod とそれらの提供するサービスの安定性と信頼性を確保できます。

このリファレンス環境では、実行できる自動化ジョブの数を最大化することに重点が置かれています。つまり、Red Hat OpenShift 環境内で使用可能な 3 つのワーカーノードのうち、1 つのワーカーノードが制御 Pod の実行専用であり、他の 2 つのワーカーノードが自動化ジョブの実行に使用されます。

**警告**

制御 Pod を実行するワーカーノードを 1 つだけ専用にすると、専用のワーカーノードがダウンした場合に他に起動する場所がないため、サービスが失われる可能性があります。この状況を改善するための実行可能なオプションとして、自動化ジョブを実行するワーカーノードの数を減らすか、追加のワーカーノードを追加して、Red Hat OpenShift クラスター内で追加の制御 Pod レプリカを実行できます。

### 3.7.1. Automation controller の特定のワーカーノードへの制御 Pod の割り当て

Red Hat OpenShift で制御 Pod を特定のノードに割り当てるには、Pod 仕様の **node_selector** フィールドと **topology_spread_constraints** フィールドを組み合わせて使用します。**node_selector** フィールドを使用すると、Pod をホストする資格を得るためにノードが対応する必要があるラベル基準を指定できます。たとえば、ラベルが **aap_node_type: control** のノードを使用する場合、Pod 仕様で次のように指定して、Pod をこのノードに割り当てます。

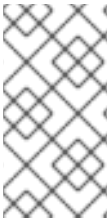
```
spec:
...
  node_selector: |
    aap_node_type: control
```

**topology_spread_constraints** は、ラベルが **aap_node_type: control** のノードでスケジュールできる Pod の最大数 (**maxSkew**) を 1 に設定します。**topologyKey** は、ノードのホスト名を示す組み込みラベルである **kubernetes.io/hostname** に設定されます。**whenUnsatisfiable** 設定が **ScheduleAnyway** に指定されているため、制約を満たした必須ラベルを持つノードが不足している場合に Pod をスケジューリングできます。**labelSelector** は、ラベルが **aap_node_type: control** の Pod と一致します。これにより、Red Hat OpenShift がノードごとに 1 つのコントローラー Pod のスケジューリングを優先

します。ただし、使用可能なワーカーノードよりも多くのレプリカリクエストがある場合、Red Hat OpenShift は、十分なリソースが使用可能であれば、同じ既存のワーカーノードで複数のコントローラー Pod をスケジュールすることができます。

**tolerations** セクションでは、**Dedicated: AutomationController** というラベルが付いたノードでのみ Pod をスケジュールできることを指定し、Toleration の効果を **NoSchedule** に設定して、必要なラベルが割り当てられていないノードで Pod がスケジュールされないようにします。これは **topology_spread_contstraints** と組み合わせて使用され、Pod をノード間で分散する方法を指定するだけでなく、Pod をスケジュールできるノードを示すためにも使用されます。

```
spec:
...
  topology_spread_constraints: |
    - maxSkew: 1
      topologyKey: "kubernetes.io/hostname"
      whenUnsatisfiable: "ScheduleAnyway"
      labelSelector:
        matchLabels:
          aap_node_type: control
  tolerations: |
    - key: "dedicated"
      operator: "Equal"
      value: "AutomationController"
      effect: "NoSchedule"
```



**注記**

ノードラベルとテイントの適用については、*付録C Red Hat OpenShift ノードへのラベルとテイントの適用* で説明しています。ノードセレクター、トポロジー制約、および容認を仕様ファイルに追加する手順は、*6章Automation controller のインストール* に記載しています。

## 3.8. データベースの高可用性の処理

Ansible Automation Platform 内の Automation controller および Automation Hub コンポーネントのデプロイでは、PostgreSQL データベースの PVC を利用します。実行中の Ansible Automation Platform の安定性を保つには、これらの PVC の可用性を確保することが極めて重要です。

Postgres Operator (PGO) および OpenShift Data Foundation (ODF) を介して Crunchy Data によって提供されるストラテジーなど、Red Hat OpenShift クラスター内で PVC の可用性を処理するために使用できるいくつかのストラテジーがあります。

Crunchy Data は、PGO (PostgreSQL クラスター) を自動管理する宣言型 PostgreSQL ソリューションを提案する Postgre Operator を提供します。PGO を使用すると、Postgres クラスターを作成し、高可用性 (HA) Postgres クラスターをスケーリングして作成し、Ansible Automation Platform などのアプリケーションに接続できます。

OpenShift Data Foundation (ODF) は、コンテナー化されたアプリケーションの永続ストレージを管理できる高可用性ストレージソリューションです。これは、Ceph、NooBaa、Rook など、複数のオープンソース Operator とテクノロジーで設定されています。これらのさまざまな Operator を使用すると、Ansible Automation Platform などのアプリケーションに接続できるファイル、ブロック、およびオブジェクトストレージをプロビジョニングおよび管理できます。

**注記**

PostgreSQL データベースに高可用性 PVC を提供する手順は、このリファレンスアーキテクチャーの範囲を超えています。

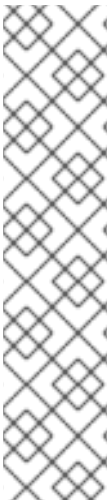
# 第4章 前提条件

このリファレンス環境用の Ansible Automation Platform 2.3 のインストールでは、以下を使用します。

- Red Hat OpenShift Platform 4.12
- 付録C *Red Hat OpenShift ノードへのラベルとテイントの適用*
- Automation Hub の **ReadWriteMany** ストレージ要件を処理する Amazon S3 バケット

**警告**

AAP 2.3 のインストールには **少なくとも** バージョン Red Hat OpenShift 4.9 が必要です。詳細は、Red Hat Ansible Automation Platform のライフサイクル ページを参照してください。



**注記**

Red Hat OpenShift をデプロイする方法は多数あります。Red Hat OpenShift クラスターのサイズは、(Ansible Automation Platform だけでなく) アプリケーションの特定の要件によって異なります。考慮すべき要因には、クラスターにアクセスするユーザーの数、アプリケーションの実行に必要なデータとリソースの量、およびスケーラビリティーと冗長性の要件が含まれます。

Red Hat OpenShift のデプロイ方法の詳細は、インストールガイド を参照してください。

上記のインストールガイドに加えて、OpenShift 4 Resources Configuration: Methodology and Tools ブログ記事を参照して、ニーズに合わせて適切なクラスターサイズを決定してください。

**注記**

Automation Hub には、複数の Pod がコレクションなど、共有コンテンツにアクセスできるように、**ReadWriteMany** ファイルベースのストレージ、Azure Blob ストレージ、または Amazon S3 準拠のストレージが必要です。

このリファレンス環境は、Amazon S3 バケットの作成を利用して、**ReadWriteMany** ストレージの要件に対応します。詳細は 付録D *Amazon S3 バケットの作成* を確認してください。

# 第5章 ANSIBLE AUTOMATION PLATFORM OPERATOR のインストール

Ansible Automation Platform Operator をインストールする場合に推奨されるデプロイ方法として、手動更新承認を使用して、対象の名前空間にクラスター範囲の Operator をインストールすることが挙げられます。

このデプロイメント方法の主な利点は、対象となる名前空間内のリソースのみに影響を与えることです。これにより、異なる名前空間での処理方法について AAP Operator の範囲を柔軟に制限できます。

たとえば、アップグレードのテスト中にさまざまな AAP デプロイメントを管理するために、個別の **devel** および **prod** 名前空間を使用する場合などです。

Ansible Automation Platform Operator をデプロイする手順は次のとおりです。

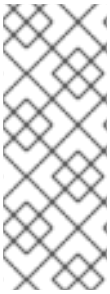
- クラスターの認証情報を使用して、Red Hat OpenShift Web コンソールにログインします。
- 左側のナビゲーションメニューで、Operators → OperatorHub を選択します。
- Ansible Automation Platform を検索して選択します。
- Ansible Automation Platform のインストールページで、インストールを選択します。
- Operator のインストールページで、
  - 適切な更新チャンネル **stable-2.3-cluster-scoped** を選択します。
  - 適切なインストールモード (**クラスター上の特定の名前空間**) を選択します。
  - 適切なインストール済み名前空間を選択します (**Operator が推奨する名前空間: aap**)。
  - 適切な更新承認を選択します (例: **Manual)**。
- **Install** をクリックします。
- Manual approval required で **Approve** をクリックします。

Ansible Automation Platform をインストールするプロセスは、利用可能になるまで数分かかる場合があります。

インストールが完了したら、**View Operator** ボタンを選択して、インストール中に指定された名前空間 (**aap** など) にインストールされたOperator を表示します。

**注記**

この AAP Operator のデプロイメントは、名前空間 **aap** のみを対象としています。追加の名前空間が AAP Operator によって対象となる (管理対象) 場合は、それらを **OperatorGroup spec** ファイルに追加する必要があります。詳細が 付録F *管理された名前空間への AAP Operator の追加* を確認してください。

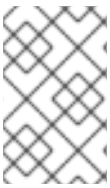


**注記**

Ansible Automation Platform Operator のデフォルトのリソース値は、一般的なインストールに適しています。ただし、多数の Automation controller および Automation Hub 環境をデプロイする場合は、**subscription.spec.config.resources** を使用して、サブスクリプション仕様内で Ansible Automation Platform Operator のリソースしきい値を増やすことを推奨します。これにより、Operator は増加したワークロードを処理してパフォーマンスの問題を防ぐのに十分なリソースを確保できます。

# 第6章 AUTOMATION CONTROLLER のインストール

Ansible Automation Platform Operator のインストールが完了したら、次の手順で Red Hat OpenShift クラスター内に Automation controller をインストールします。



**注記**

リソースの要求と制限の値は、このリファレンス環境に固有のものです。3章*作業を開始する前の注意事項* セクションを参照して、お使いの Red Hat OpenShift 環境の値を正しく計算してください。

**警告**

Automation controller のインスタンスが削除されても、関連付けられている Persistent Volume Claims (PVC) は自動的に削除されません。これにより、新しいデプロイメントが以前のデプロイメントと同じ名前である場合、移行中に問題が発生する可能性があります。同じ名前空間に新しい Automation controller インスタンスをデプロイする前に、古い PVC を削除することを推奨します。以前のデプロイメント PVC を削除する手順は 付録B *以前の AAP インストールからの既存の PVC の削除* を参照してください。

- クラスターの認証情報を使用して、Red Hat OpenShift Web コンソールにログインします。
- 左側のナビゲーションメニューで、**Operators → Installed Operators** を選択し、**Ansible Automation Platform** を選択します。
- **Automation Controller** タブに移動し、**Create AutomationController** をクリックします。
- フォームビュー内で、**名前** (例: **my-automation-controller**) を指定し、**Advanced configuration** を選択して追加オプションを展開します。
- **Additional configuration** 内で、**開始する前に** セクションから計算された各コンテナーの適切な **リソース要件** を設定します。
  - **Web コンテナーのリソース要件** を拡張する
    - 制限: CPU コア: 2000m、メモリー: 1.5Gi
    - 要求: CPU コア: 500m、メモリー: 1.5Gi
  - **タスクコンテナーのリソース要件** を拡張する
    - 制限: CPU コア: 4000m、メモリー: 8Gi
    - 要求: CPU コア: 1000m、メモリー: 8Gi
  - **EE コントロールプレーンコンテナーリソース要件** を拡張する
    - 制限: CPU コア: 500m、メモリー: 400Mi
    - 要求: CPU コア: 100m、メモリー: 400Mi

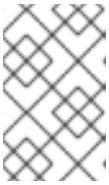
- **Redis コンテナーのリソース要件** を拡張する
  - 制限: CPU コア: 500m、メモリー: 1.5Gi
  - 要求: CPU コア: 250m、メモリー: 1.5Gi
- **PostgreSQL コンテナーのリソース要件** を拡張する
  - 制限: CPU コア: 1000m、メモリー: 1Gi
  - 要求: CPU コア: 500m、メモリー: 1Gi

- Create AutomationController ページの上部で、**YAML ビュー** を切り替えます
  - **spec:** セクション内に、**extra_settings** パラメーターを追加して、*3章 作業を開始する前の注意事項* セクションで計算した **AWX_CONTROL_NODE_TASK_IMPACT** の値を指定します。

```
spec:
...
  extra_settings:
    - setting: AWX_CONTROL_NODE_TASK_IMPACT
      value: "5"
```

- YAML view 内で、spec セクションに以下を追加して、制御 Pod の専用ノードを追加します。

```
spec:
...
 node_selector: |
   aap_node_type: control
 topology_spread_constraints: |
   - maxSkew: 1
     topologyKey: "kubernetes.io/hostname"
     whenUnsatisfiable: "ScheduleAnyway"
     labelSelector:
       matchLabels:
         aap_node_type: control
 tolerations: |
   - key: "dedicated"
     operator: "Equal"
     value: "AutomationController"
     effect: "NoSchedule"
```



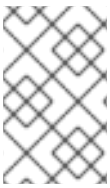
**注記**

制御 Pod を実行する適切な専用ワーカーノードに、ノードラベルとテイントが設定されていることを確認します。設定する詳細情報は、*付録C Red Hat OpenShift ノードへのラベルとテイントの適用* を参照してください。

- Create ボタンをクリックします

# 第7章 AUTOMATION HUB のインストール

Ansible Automation Platform Operator のインストールが完了したら、次の手順で Red Hat OpenShift クラスター内に Automation Hub をインストールします。

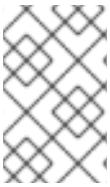


**注記**

リソースの要求と制限の値は、このリファレンス環境に固有のものです。3章*作業を開始する前の注意事項* セクションを参照して、お使いの Red Hat OpenShift 環境の値を正しく計算してください。

**警告**

Automation Hub のインスタンスが削除されても、関連付けられている永続ボリュームクレーム (PVC) は自動的に削除されません。これにより、新しいデプロイメントが以前のデプロイメントと同じ名前である場合、移行中に問題が発生する可能性があります。同じ名前空間に新しい Automation Hub インスタンスをデプロイする前に、古い PVC を削除することを推奨します。以前のデプロイメント PVC を削除する手順は 付録B *以前の AAP インストールからの既存の PVC の削除* を参照してください。



**注記**

複数の Pod がコレクションなどの共有コンテンツに確実にアクセスできるようにするには、Automation Hub の操作に **ReadWriteMany** ファイルベースのストレージ、Azure Blob ストレージ、または Amazon S3 準拠のストレージが必要です。

- クラスターの認証情報を使用して、Red Hat OpenShift Web コンソールにログインします。
- 左側のナビゲーションメニューで、**Operators → Installed Operators** を選択し、**Ansible Automation Platform** を選択します。
- **Automation Hub** タブに移動し、**Create AutomationHub** をクリックします。
- フォームビュー内
  - **Name** を指定します (例: **my-automation-hub**)。
  - **Storage type** 内で、**ReadWriteMany** 準拠のストレージを選択します。

**注記**

このリファレンス環境では、**ReadWriteMany** ストレージとして Amazon S3 を使用します。Amazon S3 バケットを作成するための詳細は、付録D *Amazon S3 バケットの作成* を参照してください。

    - **S3 ストレージシークレット** を指定します。内で作成する方法は 付録E *AWS S3 シークレットの作成* を参照してください。
  - **Advanced configuration** を選択して、追加のオプションをデプロイメントします。

- **PostgreSQL コンテナーのストレージ要件内 (マネージドインスタンスを使用する場合)**
  - ストレージ制限を 50Gi に設定する
  - ストレージ要求を 8Gi に設定する
- **PostgreSQL コンテナーのリソース要件内 (マネージドインスタンスを使用する場合)**
  - 制限: CPU コア: 500m、メモリー: 1Gi
  - 要求: CPU コア: 200m、メモリー: 1Gi
- **Redis deployment configuration** 内で、**Advanced configuration** を選択します。
  - **インメモリーデータストアのリソース要件** を選択する
    - 制限: CPU コア: 250m、メモリー: 200Mi
    - 要求: CPU コア: 100m、メモリー: 200Mi
- **API server configuration** 内で、**Advanced configuration** を選択します。
  - **API サーバーのリソース要件** を選択する
    - 制限: CPU コア: 250m、メモリー: 400Mi
    - 要求: CPU コア: 150m、メモリー: 400Mi
- **Content server configuration** 内で、**Advanced configuration** を選択します。
  - **コンテンツサーバーのリソース要件** を選択する
    - 制限: CPU コア: 250m、メモリー: 400Mi
    - 要求: CPU コア: 100m、メモリー: 400Mi
- **Worker configuration** 内で、**Advanced configuration** を選択します。
  - **ワーカーのリソース要件** を選択する
    - 制限: CPU コア: 1000m、メモリー: 3Gi
    - 要求: CPU コア: 500m、メモリー: 3Gi

- **Create** ボタンをクリックします

# 第8章 AUTOMATION CONTROLLER ダッシュボードへのログイン

Automation controller のインストールが成功すると、次の手順でダッシュボードにアクセスできます。

- Red Hat OpenShift Web コンソール内の左側のナビゲーションメニューで、**Operators → Installed Operators** を選択します。
- **Ansible Automation Platform** を選択します。
- Operator Details 内で、**Automation Controller** タブを選択します。
- インストールされている Automation controller の **Name** を選択します。
- **Automation Controller overview**内に、URL、管理者ユーザー、管理者パスワードなどの詳細が表示されます。

# 第9章 AUTOMATION HUB ダッシュボードへのログイン

- Red Hat OpenShift Web コンソール内の左側のナビゲーションメニューで、**Operators → Installed Operators** を選択します。
- **Ansible Automation Platform** を選択します。
- Operator Details 内で、**Automation Hub** タブを選択します。
- インストールされている Automation Hub の **Name** を選択します。
- **Automation Hub の概要** では、URL、管理者ユーザー、管理者パスワードなどの詳細が提供されます。

# 第10章 ANSIBLE AUTOMATION PLATFORM のモニタリング

Ansible Automation Platform のインストールが正常に完了したら、その正常性の維持と主要なメトリックを監視する機能を優先することが重要です。

このセクションでは、Red Hat OpenShift 内に、新しくインストールされた Ansible Automation Platform 環境によって提供される API メトリックを監視する方法に焦点を当てます。

## 10.1. API メトリック監視に使用するもの

Prometheus および Grafana

Prometheus は、メトリックを収集および集計するためのオープンソースの監視ソリューションです。カスタマイズ可能なダッシュボードでデータ分析の実行やメトリックの取得を行うオープンソースソリューションである Grafana と、Prometheus のモニタリング機能を連携すると、メトリックをリアルタイムで視覚化して、Ansible Automation Platform のステータスと正常性を追跡できます。

## 10.2. 得られるメトリクス

Grafana の事前構築済みダッシュボードには、次のものが表示されます。

- Ansible Automation Platform のバージョン
- コントローラーノードの数
- ライセンスで利用可能なホストの数
- 使用ホスト数
- 総ユーザー数
- ジョブの成功
- ジョブの失敗
- ジョブ実行の種類別の数量
- 実行中のジョブと保留中のジョブの数を示すグラフィック
- ワークフロー、ホスト、インベントリー、ジョブ、プロジェクト、組織などの量を示すツールの変化を示すグラフ。

この Grafana ダッシュボードは、関心のある他のメトリックを取得するようにカスタマイズできます。ただし、Grafana ダッシュボードのカスタマイズは、このリファレンスアーキテクチャーの範囲外です。

## 10.3. ANSIBLE PLAYBOOK によるインストール

カスタマイズされた Grafana ダッシュボードを使用して、Prometheus を介して Ansible Automation Platform を監視するプロセスは、数分でインストールできます。以下は、ビルド済みの Ansible Playbook を利用した Grafana ダッシュボードのカスタマイズ手順を示しています。

Ansible Playbook を正常に実行するには、次の手順が必要です。

- Automation controller 内でのカスタム認証情報タイプの作成

- Automation controller 内での kubeconfig 認証情報の作成
- Ansible Playbook を実行するためのプロジェクトとジョブテンプレートの作成

### 10.3.1. カスタム認証情報の種類の作成

Ansible Automation Platform ダッシュボード内で以下を行います。

1. **Administration→Credential Types** で、青色の **Add** ボタンをクリックします。
2. **Name** を入力します (例: **Kubeconfig**)。
3. 入力設定内で、次の YAML を入力します。

```
fields:
  - id: kube_config
    type: string
    label: kubeconfig
    secret: true
    multiline: true
```

4. インジェクター設定内で、次の YAML を入力します。

```
env:
  K8S_AUTH_KUBECONFIG: '{{ tower.filename.kubeconfig }}'
file:
  template.kubeconfig: '{{ kube_config }}'
```

5. **Save** をクリックします。

### 10.3.2. kubeconfig 認証情報の作成

Ansible Automation Platform ダッシュボード内で以下を行います。

1. **Resources→Credentials** の下で、青い **Add** ボタンをクリックします。
2. **Name** を入力します (例: **OpenShift-Kubeconfig)**。
3. **Credential Type** ドロップダウンで、**Kubeconfig** を選択します。
4. **Type Details** テキストボックス内に、Red Hat OpenShift クラスターの kubeconfig ファイルを挿入します。
5. **Save** をクリックします。

### 10.3.3. プロジェクトの作成

Ansible Automation Platform ダッシュボード内で以下を行います。

1. **Resources→Projects** の下で、青い **Add** ボタンをクリックします。
2. **Name** を入力してください (例: **Monitoring AAP Project**)。
3. 組織として **Default** を選択します。

4. **Execution Environment** として **Default execution environment** 環境を選択します。
5. **Source Control Credential Type** として **Git** を選択します。
6. **Type Details** で、以下を実行します。
    a. **Source Control URL** (https://github.com/ansible/aap_ocp_refarch) を追加します。
7. **Options** 内で以下を実行します。
    a. **Clean, Delete, Update Revision on Launch**を選択します。
8. **Save** をクリックします。

### 10.3.4. ジョブテンプレートを作成した Ansible Playbook の実行

Ansible Automation Platform ダッシュボード内で以下を実行します。

1. **Resources→Templates** の下で、青色の **Add→Add job template** をクリックします。
2. **Name** を入力します (例: **Monitoring AAP Job**)。
3. **Job Type** として **Run** を選択します。
4. **Inventory** として **Demo Inventory** を選択します。
5. **Project** として **Monitoring AAP Project** を選択します。
6. **Execution Environment** として **Default execution environment** 環境を選択します。
7. **Playbook** として **aap-prometheus-grafana/playbook.yml** を選択します。
8. **Credentials** を選択し、カテゴリーを **Machine** から **Kubeconfig** に切り替えます。
9. Red Hat OpenShift クラスターにアクセスするための適切な **kubeconfig** を選択します (例: **OpenShift-Kubeconfig**)。
10. **任意の手順**: **Variables** 内で、次の変数を変更できます。
    a. prometheus_namespace: **<your-specified-value>**
    b. ansible_namespace: **<your-specified-value>**
11. **Save** をクリックします。
12. **Launch** をクリックして、Ansible Playbook を実行します。
13. Grafana と Prometheus へのログイン情報は、ジョブ出力内に表示されます。

# 付録A 著者について

**Roger Lopez**

Roger Lopez is a Principal Technical Marketing Manager bringing 10+ years of computer industry experience delivering high-value solutions used by our sales, marketing and engineering teams to develop best practice documentation & methods for internal and external customers. He is a Red Hat Certified Engineer (RHCE) with experience building solutions around Ansible, OpenShift and OpenStack.

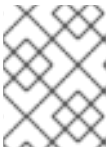
# 付録B 以前の AAP インストールからの既存の PVC の削除

1. ターミナルウィンドウを開き、Red Hat OpenShift クラスターにログインします。たとえば、KUBECONFIG ファイルをエクスポートします。

```
$ export KUBECONFIG=/path/to/kubeconfig
```

2. 指定された Ansible Automation Platform 名前空間 (**aap** など) の PVC のリストを確認します。

```
$ oc get pvc -n aap
```



**注記**

これは、名前、ステータス、容量、アクセスモード、およびストレージクラスを含む、**aap** 名前空間内のすべての PVC のリストを表示します。

3. リストから削除する PVC を特定します。

4. **oc delete** コマンドを使用して PVC を削除します。

```
oc delete pvc <pvc-name-to-remove> -n aap
```

**aap** 名前空間内で使用可能な PVC のリストを確認し、PVC が表示されなくなっていることを確認します。

```
$ oc get pvc -n aap
```

# 付録C RED HAT OPENSHIFT ノードへのラベルとテイントの適用

制御 Pod を専用の Red Hat OpenShift ノードで実行するには、指定したノードに適切なラベルとテイントを設定する必要があります。

この例では、ラベル **aap_node_type=control** の **worker** ロールを持つ Red Hat OpenShift ノードの 1 つを選択します。

1. 実行中のラベルを付けるいずれかのノードの名前を取得します。

```
$ oc get nodes
```

2. リストからノードを選択し、その名前をメモします (**worker1** など)。
3. **aap_node_type=control** ラベルをノードに適用します。

```
$ oc label node <node-name> aap_node_type=control
```

**注記**

**<node-name>** は、ラベル付けするノードの名前に置き換えます。

4. 次のように、ラベルの作成を確認します。

```
$ oc get nodes --show-labels | grep <node-name>
```

ラベルを作成したら、次のステップとして、すでにラベルを作成したワーカーノードに **NoSchedule** テイントを追加します。

次のコマンドは、ノードに **NoSchedule** テイントを追加します。

```
oc adm taint nodes <node-name> dedicated=AutomationController:NoSchedule
```

**dedicated**: これは、テイントのキー (テイントの識別用に提供される任意の文字列) です。

**AutomationController**: これはテイントに与えられる任意の値です。

**NoSchedule**: これは、このテイントを許容しない Pod がこのノードにスケジュールされないように指定するテイントの動作です。

このテイントをノードに適用することで、テイントを許容する特定のタイプのワークロード用にこのノードを予約するように Kubernetes スケジューラーに指示します。この場合、dedicated=AutomationController toleration を使用してワークロード用にノードを予約しています。

5. テイントが適用されたことを確認します。

```
$ oc get nodes \
-o jsonpath='{range.items[*]}{@.metadata.name}{"\t"}{@.spec.taints[*].key}:
{@.spec.taints[*].value}{"\n"}{end}' \
| grep AutomationController
```

# 付録D AMAZON S3 バケットの作成

1. ターミナルを開き、AWS CLI がインストールされ、AWS 認証情報で設定されていることを確認します。
2. 次のコマンドを実行して、新しい S3 バケットを作成します。

```
$ aws s3 mb s3://<bucket-name> --region <region-name>
```

**警告**

バケット名は一意の名前にする必要があります。

3. 次のコマンドを実行して、バケットが正常に作成されたことを確認します。

```
$ aws s3 ls | grep <bucket-name>
```

# 付録E AWS S3 シークレットの作成

Red Hat OpenShift では、シークレットを保存してそのシークレットを使用して Amazon S3 などの外部サービスで認証することができます。このリファレンス環境は、Amazon S3 バケットを利用して、Automation Hub を実行するための **ReadWriteMany** 要件を満たします。

以下の手順は、*7章Automation Hub のインストール* の章で使用するシークレットを Red Hat OpenShift クラスターで作成する方法を説明します。

```
$ cat s3-secret.yml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: s3-secret
  namespace: aap
stringData:
  s3-access-key-id: my_key
  s3-secret-access-key: my_access_key
  s3-bucket-name: my_bucket
  s3-region: my_region
```

```
$ oc create -f s3-secret.yml
```

# 付録F 管理された名前空間への AAP OPERATOR の追加

以下は、**aap** 名前空間に存在する既存の AAP Operator に、別の管理対象名前空間を追加する手順です。

- Red Hat OpenShift UI にログインします。
- **Operators→Installed Operators** 内で、**Ansible Automation Platform** を選択します。
- **Details** ページで、**ClusterServiceVersion Details** まで下にスクロールします。
  - 右側の列で、**OperatorGroup** をクリックします。
- **OperatorGroup** の詳細内で、**YAML** を選択します。
  - **spec** セクションの下に、**aap-devel** などの追加の名前空間を追加します。

```
spec:
...
  targetNamespaces:
    - aap
    - aap-devel
```

- **Save** をクリックします。

**注記**

OperatorGroup 仕様ファイルに追加する前に、対象の名前空間がすでに存在している必要があります。

# 付録G 参考資料

- What everyone should know about Kubernetes memory limits, OOMKilled pods, and pizza parties
- Pulp Project Hardware requirements
- Operator ベースのインストールにおける Red Hat Ansible Automation Platform のパフォーマンスに関する考慮事項
- Red Hat Ansible Automation Platform Operator の OpenShift Container Platform へのデプロイ
- Pulp Operator storage configuration
- AWX Operator
- Resources consumed by idle PostgreSQL connections
- Operator scoping with OperatorGroups
- Ansible Tower and Grafana Dashboards

# 付録H REVISION HISTORY

| | | |
|---|---|---|
| **改訂 1.0-0** | 2023-03-07 | **Roger Lopez** |

- **Initial Release**