OpenShift Container Platform 4.15

モニタリング

OpenShift Container Platform でのモニタリングスタックの設定および使用

Last Updated: 2024-04-16

OpenShift Container Platform 4.15 モニタリング

OpenShift Container Platform でのモニタリングスタックの設定および使用

法律上の通知

概要

本書では、OpenShift Container Platform で Prometheus モニタリングスタックを設定し、使用する方法について説明します。

目次

第1章 モニタリングの概要

1.1. OPENSHIFT CONTAINER PLATFORM モニタリングについて

OpenShift Container Platform には、コアプラットフォームコンポーネントのモニタリングを提供する事前に設定され、事前にインストールされた自己更新型のモニタリングスタックが含まれます。また、ユーザー定義プロジェクトのモニタリングを有効 にするオプションもあります。

クラスター管理者は、サポートされている設定で モニタリングスタックを設定 できます。OpenShift Container Platform は、追加設定が不要のモニタリングのベストプラクティスを提供します。

管理者にクラスターの問題について即時に通知するアラートのセットがデフォルトで含まれます。OpenShift Container Platform Web コンソールのデフォルトのダッシュボードには、クラスターの状態をすぐに理解できるようにするクラスターのメトリックの視覚的な表示が含まれます。OpenShift Container Platform Web コンソールを使用して、メトリクスの表示と管理、アラート、および モニタリングダッシュボードの確認 することができます。

OpenShift Container Platform Web コンソールの **Observe** セクションでは、metrics、alerts、monitoring dashboards、metrics targets などのモニタリング機能にアクセスして管理できます。

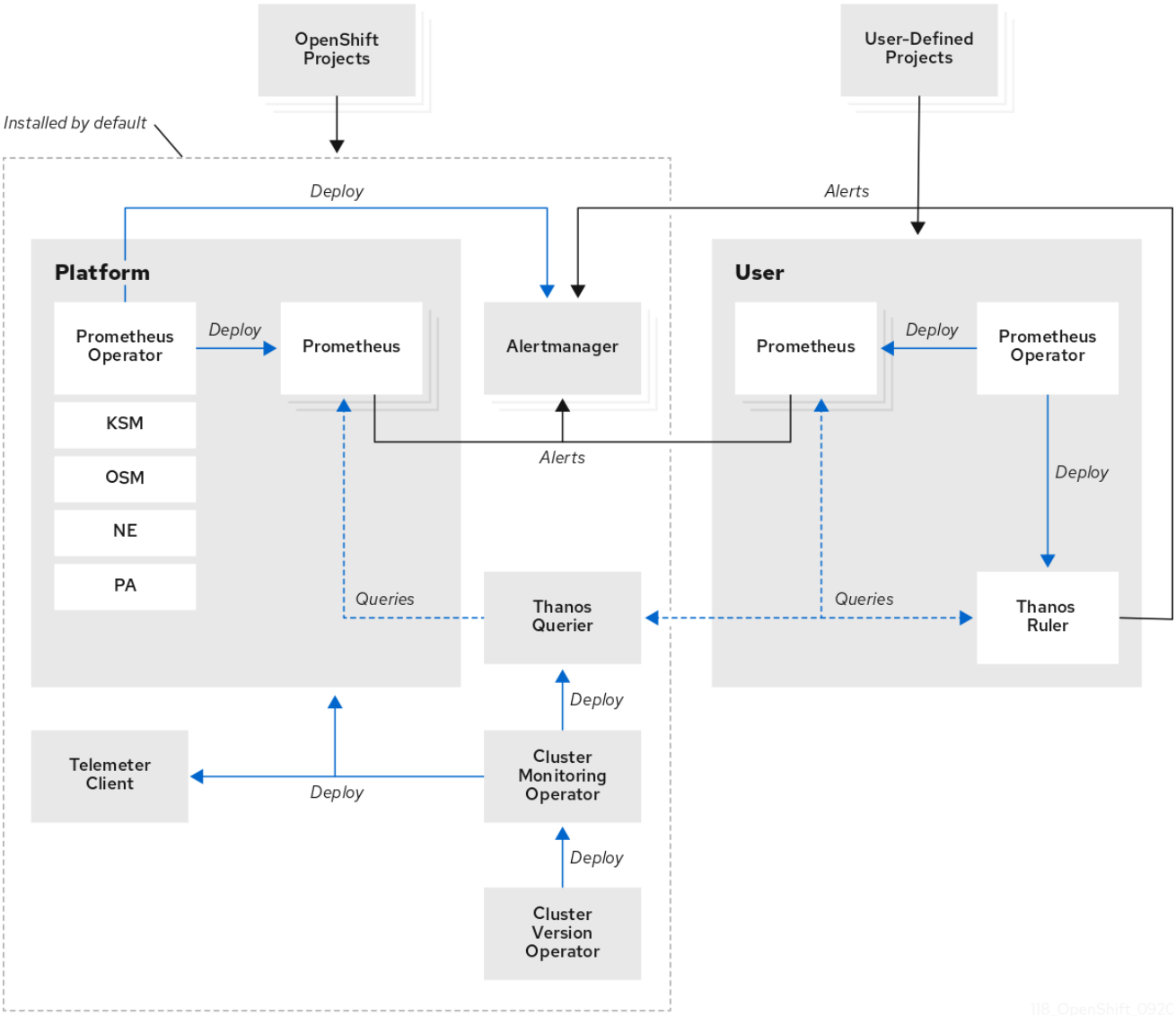
OpenShift Container Platform のインストール後に、クラスター管理者はオプションでユーザー定義プロジェクトのモニタリングを有効にできます。この機能を使用することで、クラスター管理者、開発者、および他のユーザーは、サービスと Pod を独自のプロジェクトでモニターする方法を指定できます。クラスター管理者は、Troubleshooting monitoring issues で、Prometheus によるユーザーメトリックの使用不可やディスクスペースの大量消費などの一般的な問題に対する回答を見つけることができます。

1.2. モニタリングスタックについて

OpenShift Container Platform モニタリングスタックは、Prometheus オープンソースプロジェクトおよびその幅広いエコシステムをベースとしています。モニタリングスタックには、以下のコンポーネントが含まれます。

- **デフォルトのプラットフォームモニタリングコンポーネント**。プラットフォームモニタリングコンポーネントのセットは、OpenShift Container Platform のインストール時にデフォルトで **openshift-monitoring** プロジェクトにインストールされます。これにより、Kubernetes サービスを含むコアクラスターコンポーネントのモニタリングが可能になります。デフォルトのモニタリングスタックは、クラスターのリモートのヘルスモニタリングも有効にします。
  これらのコンポーネントは、以下の図の **Installed by default** (デフォルトのインストール) セクションで説明されています。

- **ユーザー定義のプロジェクトをモニターするためのコンポーネント**。オプションでユーザー定義プロジェクトのモニタリングを有効にした後に、追加のモニタリングコンポーネントは **openshift-user-workload-monitoring** プロジェクトにインストールされます。これにより、ユーザー定義プロジェクトのモニタリング機能が提供されます。これらのコンポーネントは、以下の図の **User** (ユーザー) セクションで説明されています。



1.2.1. デフォルトのモニタリングコンポーネント

デフォルトで、OpenShift Container Platform 4.15 モニタリングスタックには、以下のコンポーネントが含まれます。

表1.1 デフォルトのモニタリングスタックコンポーネント

コンポーネント	説明
クラスターモニタリング Operator	Cluster Monitoring Operator (CMO) は、モニタリングスタックの中心的なコンポーネントです。Prometheus および Alertmanager インスタンス、Thanos Querier、Telemeter Client、およびメトリクスターゲットをデプロイ、管理、および自動更新します。CMO は Cluster Version Operator (CVO) によってデプロイされます。

コンポーネント	説明
Prometheus Operator	**openshift-monitoring** プロジェクトの Prometheus Operator(PO) は、プラットフォーム Prometheus インスタンスおよび Alertmanager インスタンスを作成、設定、および管理します。また、Kubernetes ラベルのクエリーに基づいてモニタリングターゲットの設定を自動生成します。
Prometheus	Prometheus は、OpenShift Container Platform モニタリングスタックのベースとなるモニタリングシステムです。Prometheus は Time Series を使用するデータベースであり、メトリックのルール評価エンジンです。Prometheus は処理のためにアラートを Alertmanager に送信します。
Prometheus アダプター	Prometheus アダプター (上記の図の PA) は、Prometheus で使用する Kubernetes ノードおよび Pod クエリーを変換します。変換されるリソースメトリックには、CPU およびメモリーの使用率メトリックが含まれます。Prometheus アダプターは、Horizontal Pod Autoscaling のクラスターリソースメトリック API を公開します。Prometheus アダプターは **oc adm top nodes** および **oc adm top pods** コマンドでも使用されます。
Metrics Server (テクノロジープレビュー)	有効にすると、Metrics Server コンポーネントはリソースメトリックを収集し、他のツールや API で使用できるように **metrics.k8s.io** Metrics API サービスで公開します。これにより、コアプラットフォームの Prometheus スタックによるこの機能の処理が不要になります。 **TechPreviewNoUpgrade** オプションを使用して **FeatureGate** カスタムリソースを設定すると、Prometheus Adapter の代わりに Metrics Server コンポーネントのテクノロジープレビューが自動的にインストールされます。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。
Alertmanager	Alertmanager サービスは、Prometheus から送信されるアラートを処理します。また、Alertmanager は外部の通知システムにアラートを送信します。
kube-state-metrics エージェント	kube-state-metrics エクスポーターエージェント (上記の図の KSM) は、Kubernetes オブジェクトを Prometheus が使用できるメトリックに変換します。

コンポーネント	説明
monitoring-plugin	monitoring-plugin 動的プラグインコンポーネントは、OpenShift Container Platform Web コンソールの **Observe** セクションにモニタリングページをデプロイします。Cluster Monitoring Operator (CMO) config map 設定を使用すると、Web コンソールページの monitoring-plugin リソースを管理できます。
openshift-state-metrics エージェント	openshift-state-metrics エクスポーター (上記の図の OSM) は、OpenShift Container Platform 固有のリソースのメトリクスを追加して、kube-state-metrics を拡張します。
node-exporter エージェント	ノードエクスポーターエージェント (上記の図の NE) は、クラスター内のすべてのノードに関するメトリクスを収集します。node-exporter エージェントはすべてのノードにデプロイされます。
Thanos Querier	Thanos Querier は、単一のマルチテナントインターフェイスで、OpenShift Container Platform のコアメトリクスおよびユーザー定義プロジェクトのメトリクスを集約し、オプションでこれらの重複を排除します。
Telemeter クライアント	Telemeter Client は、クラスターのリモートヘルスモニタリングを容易にするために、プラットフォーム Prometheus インスタンスから Red Hat にデータのサブセクションを送信します。

モニターリグスタックのすべてのコンポーネントはスタックによってモニターされ、OpenShift Container Platform の更新時に自動的に更新されます。

注記

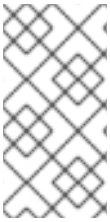
モニタリングスタックのすべてのコンポーネントは、クラスター管理者が一元的に設定する TLS セキュリティープロファイル設定を使用します。TLS セキュリティー設定を使用するモニタリングスタックコンポーネントを設定する場合、コンポーネントはグローバル OpenShift Container Platform **apiservers.config.openshift.io/cluster** リソースの **tlsSecurityProfile** フィールドにすでに存在する TLS セキュリティープロファイル設定を使用します。

1.2.2. デフォルトのモニタリングターゲット

スタック自体のコンポーネントに加え、デフォルトのモニタリングスタックは以下をモニターします。

- CoreDNS
- Elasticsearch (ロギングがインストールされている場合)
- etcd

- Fluentd (ロギングがインストールされている場合)
- HAProxy
- イメージレジストリー
- Kubelets
- Kubernetes API サーバー
- Kubernetes コントローラーマネージャー
- Kubernetes スケジューラー
- OpenShift API サーバー
- OpenShift Controller Manager
- Operator Lifecycle Manager (OLM)
- Vector (ロギングがインストールされている場合)



注記

各 OpenShift Container Platform コンポーネントはそれぞれのモニタリング設定を行います。OpenShift Container Platform コンポーネントのモニタリングに関する問題は、一般的なモニタリングコンポーネントに対してではなく、そのコンポーネントに対して Jira 問題 を開きます。

他の OpenShift Container Platform フレームワークのコンポーネントもメトリクスを公開する場合があります。詳細については、それぞれのドキュメントを参照してください。

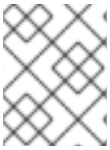
1.2.3. ユーザー定義プロジェクトをモニターするためのコンポーネント

OpenShift Container Platform には、ユーザー定義プロジェクトでサービスおよび Pod をモニターできるモニタリングスタックのオプションの拡張機能が含まれています。この機能には、以下のコンポーネントが含まれます。

表1.2 ユーザー定義プロジェクトをモニターするためのコンポーネント

コンポーネント	説明
Prometheus Operator	**openshift-user-workload-monitoring** プロジェクトの Prometheus Operator (PO) は、同じプロジェクトで Prometheus および Thanos Ruler インスタンスを作成し、設定し、管理します。
Prometheus	Prometheus は、ユーザー定義のプロジェクト用にモニタリング機能が提供されるモニタリングシステムです。Prometheus は処理のためにアラートを Alertmanager に送信します。

コンポーネント	説明
Thanos Ruler	Thanos Ruler は、別のプロセスとしてデプロイされる Prometheus のルール評価エンジンです。OpenShift Container Platform では、Thanos Ruler はユーザー定義プロジェクトをモニタリングするためのルールおよびアラート評価を提供します。
Alertmanager	Alertmanager サービスは、Prometheus および Thanos Ruler から送信されるアラートを処理します。Alertmanager はユーザー定義のアラートを外部通知システムに送信します。このサービスのデプロイは任意です。



注記

上記の表のコンポーネントは、モニタリングがユーザー定義のプロジェクトに対して有効にされた後にデプロイされます。

OpenShift Container Platform が更新されると、これらのコンポーネントはすべてスタックによってモニタリングされ、自動的に更新されます。

1.2.4. ユーザー定義プロジェクトのターゲットのモニタリング

モニタリングがユーザー定義プロジェクトについて有効にされている場合には、以下をモニターできます。

- ユーザー定義プロジェクトのサービスエンドポイント経由で提供されるメトリック。
- ユーザー定義プロジェクトで実行される Pod。

1.3. OPENSHIFT CONTAINER PLATFORM モニタリングの一般用語集

この用語集では、OpenShift Container Platform アーキテクチャーで使用される一般的な用語を定義します。

Alertmanager

Alertmanager は、Prometheus から受信したアラートを処理します。また、Alertmanager は外部の通知システムにアラートを送信します。

アラートルール

アラートルールには、クラスター内の特定の状態を示す一連の条件が含まれます。アラートは、これらの条件が true の場合にトリガーされます。アラートルールには、アラートのルーティング方法を定義する重大度を割り当てることができます。

クラスターモニタリング Operator

Cluster Monitoring Operator (CMO) は、モニタリングスタックの中心的なコンポーネントです。Thanos Querier、Telemeter Client、メトリクスターゲットなどの Prometheus インスタンスをデプロイおよび管理して、それらが最新であることを確認します。CMO は Cluster Version Operator (CVO) によってデプロイされます。

Cluster Version Operator

Cluster Version Operator (CVO) はクラスター Operator のライフサイクルを管理し、その多くはデフォルトで OpenShift Container Platform にインストールされます。

config map

ConfigMap は、設定データを Pod に注入する方法を提供します。タイプ **ConfigMap** のボリューム内の ConfigMap に格納されたデータを参照できます。Pod で実行しているアプリケーションは、このデータを使用できます。

Container

コンテナーは、ソフトウェアとそのすべての依存関係を含む軽量で実行可能なイメージです。コンテナーは、オペレーティングシステムを仮想化します。そのため、コンテナーはデータセンターからパブリッククラウド、プライベートクラウド、開発者のラップトップなどまで、場所を問わずコンテナーを実行できます。

カスタムリソース (CR)

CR は Kubernetes API のエクステンションです。カスタムリソースを作成できます。

etcd

etcd は OpenShift Container Platform のキーと値のストアであり、すべてのリソースオブジェクトの状態を保存します。

Fluentd

Fluentd は、各 OpenShift Container Platform ノードに常駐するログコレクターです。アプリケーション、インフラストラクチャー、および監査ログを収集し、それらをさまざまな出力に転送します。

注記

Fluentd は非推奨となっており、今後のリリースで削除される予定です。Red Hat は、現在のリリースのライフサイクル中にこの機能のバグ修正とサポートを提供しますが、この機能は拡張されなくなりました。Fluentd の代わりに、Vector を使用できます。

Kubelets

ノード上で実行され、コンテナーマニフェストを読み取ります。定義されたコンテナーが開始され、実行されていることを確認します。

Kubernetes API サーバー

Kubernetes API サーバーは、API オブジェクトのデータを検証して設定します。

Kubernetes コントローラーマネージャー

Kubernetes コントローラーマネージャーは、クラスターの状態を管理します。

Kubernetes スケジューラー

Kubernetes スケジューラーは Pod をノードに割り当てます。

labels

ラベルは、Pod などのオブジェクトのサブセットを整理および選択するために使用できるキーと値のペアです。

ノード

OpenShift Container Platform クラスター内のワーカーマシン。ノードは、仮想マシン (VM) または物理マシンのいずれかです。

Operator

OpenShift Container Platform クラスターで Kubernetes アプリケーションをパッケージ化、デプロイ、および管理するための推奨される方法。Operator は、人間による操作に関する知識を取り入れて、簡単にパッケージ化してお客様と共有できるソフトウェアにエンコードします。

Operator Lifecycle Manager (OLM)

OLM は、Kubernetes ネイティブアプリケーションのライフサイクルをインストール、更新、および管理するのに役立ちます。OLM は、Operator を効果的かつ自動化されたスケーラブルな方法で管理するために設計されたオープンソースのツールキットです。

永続ストレージ

デバイスがシャットダウンされた後でもデータを保存します。Kubernetes は永続ボリュームを使用して、アプリケーションデータを保存します。

永続ボリューム要求 (PVC)

PVC を使用して、PersistentVolume を Pod にマウントできます。クラウド環境の詳細を知らなくてもストレージにアクセスできます。

pod

Pod は、Kubernetes における最小の論理単位です。Pod には、ワーカーノードで実行される 1 つ以上のコンテナーが含まれます。

Prometheus

Prometheus は、OpenShift Container Platform モニタリングスタックのベースとなるモニタリングシステムです。Prometheus は Time Series を使用するデータベースであり、メトリックのルール評価エンジンです。Prometheus は処理のためにアラートを Alertmanager に送信します。

Prometheus アダプター

Prometheus アダプターは、Prometheus で使用するために Kubernetes ノードと Pod のクエリーを変換します。変換されるリソースメトリクスには、CPU およびメモリーの使用率が含まれます。Prometheus アダプターは、Horizontal Pod Autoscaling のクラスターリソースメトリック API を公開します。

Prometheus Operator

openshift-monitoring プロジェクトの Prometheus Operator(PO) は、プラットフォーム Prometheus インスタンスおよび Alertmanager インスタンスを作成、設定、および管理します。また、Kubernetes ラベルのクエリーに基づいてモニタリングターゲットの設定を自動生成します。

サイレンス

サイレンスをアラートに適用し、アラートの条件が true の場合に通知が送信されることを防ぐことができます。初期通知後はアラートをミュートにして、根本的な問題の解決に取り組むことができます。

storage

OpenShift Container Platform は、オンプレミスおよびクラウドプロバイダーの両方で、多くのタイプのストレージをサポートします。OpenShift Container Platform クラスターで、永続データおよび非永続データ用のコンテナーストレージを管理できます。

Thanos Ruler

Thanos Ruler は、別のプロセスとしてデプロイされる Prometheus のルール評価エンジンです。OpenShift Container Platform では、Thanos Ruler はユーザー定義プロジェクトをモニタリングするためのルールおよびアラート評価を提供します。

Vector

Vector は、各 OpenShift Container Platform ノードにデプロイするログコレクターです。各ノードからログデータを収集し、データを変換して、設定された出力に転送します。

Web コンソール

OpenShift Container Platform を管理するためのユーザーインターフェイス (UI)。

1.4. 関連情報

- リモートヘルスモニタリングについて

- ユーザーに対するユーザー定義のプロジェクトをモニターする権限の付与
- TLS セキュリティープロファイルの設定

1.5. 次のステップ

- モニタリングスタックの設定

第2章 モニタリングスタックの設定

OpenShift Container Platform インストールプログラムは、インストール前の少数の設定オプションのみを提供します。ほとんどの OpenShift Container Platform フレームワークコンポーネント (クラスターモニタリングスタックを含む) の設定はインストール後に行われます。

このセクションでは、サポートされている設定内容を説明し、モニタリングスタックの設定方法を示し、いくつかの一般的な設定シナリオを示します。

重要

モニタリングスタックのすべての設定パラメーターが公開されるわけではありません。設定では、Cluster Monitoring Operator の config map リファレンス にリストされているパラメーターとフィールドのみがサポートされます。

2.1. 前提条件

- モニタリングスタックには、追加のリソース要件があります。コンピューティングリソースの推奨事項については、Cluster Monitoring Operator のスケーリング を参照し、十分なリソースがあることを確認してください。

2.2. モニタリングのメンテナンスおよびサポート

モニタリングスタックのすべての設定オプションが公開されているわけではありません。唯一サポートされている OpenShift Dedicated モニタリング設定方法は、Cluster Monitoring Operator の Config map リファレンス で説明されているオプションを使用して Cluster Monitoring Operator を設定する方法です。**サポートされていない他の設定は使用しないでください。**

設定のパラダイムが Prometheus リリース間で変更される可能性があり、このような変更には、設定のすべての可能性が制御されている場合のみ適切に対応できます。Cluster Monitoring Operator の Config map リファレンス で説明されている設定以外の設定を使用すると、デフォルトの設定、および設計上、Cluster Monitoring Operator が自動的に差異を調整し、サポートされていない変更を元の定義済みの状態にリセットするため、変更は消えてしまいます。

2.2.1. モニタリングのサポートに関する考慮事項

注記

メトリクス、記録ルールまたはアラートルールの後方互換性を保証されません。

以下の変更は明示的にサポートされていません。

- **追加の ServiceMonitor、PodMonitor、および PrometheusRule オブジェクトを openshift-* および kube-* プロジェクトに作成します。**
- **openshift-monitoring または openshift-user-workload-monitoring プロジェクトにデプロイされるリソースまたはオブジェクト変更**OpenShift Container Platform モニタリングスタックによって作成されるリソースは、後方互換性の保証がないために他のリソースで使用されることは意図されていません。

注記

Alertmanager 設定は、**openshift-monitoring** namespace の **alertmanager-main** シークレットリソースとしてデプロイされます。ユーザー定義のアラートルーティング用に別の Alertmanager インスタンスを有効にしている場合、Alertmanager 設定も **openshift-user-workload-monitoring** namespace の **alertmanager-user-workload** シークレットリソースとしてデプロイされます。Alertmanager のインスタンスの追加ルートを設定するには、そのシークレットをデコードし、変更し、エンコードする必要があります。この手順は、前述のステートメントに対してサポートされる例外です。

- **スタックのリソースの変更**。OpenShift Container Platform モニタリングスタックは、そのリソースが常に期待される状態にあることを確認します。これらが変更される場合、スタックはこれらをリセットします。
- **ユーザー定義ワークロードの openshift-*、および kube-* プロジェクトへのデプロイ**。これらのプロジェクトは Red Hat が提供するコンポーネント用に予約され、ユーザー定義のワークロードに使用することはできません。
- **Prometheus Operator での Probe カスタムリソース定義 (CRD) による現象ベースのモニタリングの有効化**。
- **カスタム Prometheus インスタンスの OpenShift Container Platform へのインストール**。カスタムインスタンスは、Prometheus Operator によって管理される Prometheus カスタムリソース (CR) です。

2.2.2. Operator のモニタリングについてのサポートポリシー

モニタリング Operator により、OpenShift Container Platform モニタリングリソースの設定およびテスト通りに機能することを確認できます。Operator の Cluster Version Operator (CVO) コントロールがオーバーライドされる場合、Operator は設定の変更に対応せず、クラスターオブジェクトの意図される状態を調整したり、更新を受信したりしません。

Operator の CVO コントロールのオーバーライドはデバッグ時に役立ちますが、これはサポートされず、クラスター管理者は個々のコンポーネントの設定およびアップグレードを完全に制御することを前提としています。

Cluster Version Operator のオーバーライド

spec.overrides パラメーターを CVO の設定に追加すると、管理者はコンポーネントについての CVO の動作にオーバーライドのリストを追加できます。コンポーネントについて **spec.overrides[].unmanaged** パラメーターを **true** に設定すると、クラスターのアップグレードがブロックされ、CVO のオーバーライドが設定された後に管理者にアラートが送信されます。

> Disabling ownership via cluster version overrides prevents upgrades. Please remove overrides before continuing.

警告

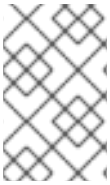
CVO のオーバーライドを設定すると、クラスター全体がサポートされていない状態になり、モニタリングスタックをその意図された状態に調整されなくなります。これは Operator に組み込まれた信頼性の機能に影響を与え、更新が受信されなくなります。サポートを継続するには、オーバーライドを削除した後に、報告された問題を再現する必要があります。

2.3. モニタリングスタックの設定の準備

モニタリング config map を作成し、更新してモニタリングスタックを設定できます。これらの config map により Cluster Monitoring Operator (CMO) が設定され、この CMO によりモニタリングスタックのコンポーネントが設定されます。

2.3.1. クラスターモニタリング config map の作成

openshift-monitoring プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを作成することで、コア OpenShift Container Platform モニタリングコンポーネントを設定できます。その後、Cluster Monitoring Operator (CMO) がモニタリングスタックのコアコンポーネントを設定します。



注記

変更を **cluster-monitoring-config ConfigMap** オブジェクトに保存すると、**openshift-monitoring** プロジェクトの Pod の一部またはすべてが再デプロイされる可能性があります。これらのコンポーネントが再デプロイするまで時間がかかる場合があります。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **cluster-monitoring-config ConfigMap** オブジェクトが存在するかどうかを確認します。

```
$ oc -n openshift-monitoring get configmap cluster-monitoring-config
```

2. **ConfigMap** オブジェクトが存在しない場合:

   a. 以下の YAML マニフェストを作成します。以下の例では、このファイルは **cluster-monitoring-config.yaml** という名前です。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
```

b. 設定を適用して **ConfigMap** を作成します。

```
$ oc apply -f cluster-monitoring-config.yaml
```

2.3.2. ユーザー定義のワークロードモニタリング config map の作成

openshift-user-workload-monitoring プロジェクトに **user-workload-monitoring-config ConfigMap** オブジェクトを作成することで、ユーザーワークロードモニタリングコンポーネントを設定できます。その後、Cluster Monitoring Operator (CMO) がユーザー定義プロジェクトをモニタリングするコンポーネントを設定します。

注記

変更を **user-workload-monitoring-config ConfigMap** オブジェクトに保存すると、**openshift-user-workload-monitoring** プロジェクトの Pod の一部またはすべてが再デプロイされる可能性があります。これらのコンポーネントが再デプロイするまで時間がかかる場合があります。ユーザー定義プロジェクトのモニタリングを最初に有効にする前に config map を作成し、設定することができます。これにより、Pod を頻繁に再デプロイする必要がなくなります。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **user-workload-monitoring-config ConfigMap** オブジェクトが存在するかどうかを確認します。

```
$ oc -n openshift-user-workload-monitoring get configmap user-workload-monitoring-config
```

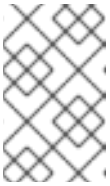
2. **user-workload-monitoring-config ConfigMap** オブジェクトが存在しない場合:

a. 以下の YAML マニフェストを作成します。以下の例では、このファイルは **user-workload-monitoring-config.yaml** という名前です。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

b. 設定を適用して **ConfigMap** を作成します。

```
$ oc apply -f user-workload-monitoring-config.yaml
```



注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

関連情報

- ユーザー定義プロジェクトのモニタリングの有効化

2.4. モニタリングスタックの設定

OpenShift Container Platform 4.15 では、**cluster-monitoring-config** または **user-workload-monitoring-config ConfigMap** オブジェクトを使用して、モニタリングスタックを設定できます。config map はクラスターモニタリング Operator (CMO) を設定し、その後にスタックのコンポーネントが設定されます。

前提条件

- **OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合**、以下を実行します。
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **OpenShift Container Platform のコアモニタリングコンポーネントを設定するには**、以下を実行します。
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

     ```
     $ oc -n openshift-monitoring edit configmap cluster-monitoring-config
     ```

     b. 設定を、**data/config.yaml** の下に値とキーのペア **<component_name>: <component_configuration>** として追加します。

     ```
     apiVersion: v1
     kind: ConfigMap
     metadata:
       name: cluster-monitoring-config
       namespace: openshift-monitoring
     ```

```
data:
  config.yaml: |
    <component>:
      <configuration_for_the_component>
```

<component> および **<configuration_for_the_component>** を随時置き換えます。

以下の **ConfigMap** オブジェクトの例は、Prometheus の永続ボリューム要求 (PVC) を設定します。これは、OpenShift Container Platform のコアコンポーネントのみをモニターする Prometheus インスタンスに関連します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s: 1
      volumeClaimTemplate:
        spec:
          storageClassName: fast
          volumeMode: Filesystem
          resources:
            requests:
              storage: 40Gi
```

1 Prometheus コンポーネントを定義し、後続の行はその設定を定義します。

- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定するには**、以下を実行します。

  a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

  ```
  $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
  ```

  b. 設定を、**data/config.yaml** の下に値とキーのペア **<component_name>: <component_configuration>** として追加します。

  ```
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: user-workload-monitoring-config
    namespace: openshift-user-workload-monitoring
  data:
    config.yaml: |
      <component>:
        <configuration_for_the_component>
  ```

  <component> および **<configuration_for_the_component>** を随時置き換えます。

以下の **ConfigMap** オブジェクトの例は、Prometheus のデータ保持期間および最小コンテナーリソース要求を設定します。これは、ユーザー定義のプロジェクトのみをモニターする Prometheus インスタンスに関連します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus: 1
      retention: 24h 2
      resources:
        requests:
          cpu: 200m 3
          memory: 2Gi 4
```

1. Prometheus コンポーネントを定義し、後続の行はその設定を定義します。
2. ユーザー定義プロジェクトをモニターする Prometheus インスタンスについて 24 時間のデータ保持期間を設定します。
3. Prometheus コンテナーの 200 ミリコアの最小リソース要求を定義します。
4. Prometheus コンテナーのメモリーの 2 GiB の最小 Pod リソース要求を定義します。

注記

Prometheus config map コンポーネントは、**cluster-monitoring-config ConfigMap** オブジェクトで **prometheusK8s** と呼ばれ、**user-workload-monitoring-config ConfigMap** オブジェクトで **prometheus** と呼ばれます。

2. ファイルを保存して、変更を **ConfigMap** オブジェクトに適用します。新規設定の影響を受けた Pod は自動的に再起動されます。

注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

警告

変更がモニタリング config map に保存されると、関連するプロジェクトの Pod およびその他のリソースが再デプロイされる可能性があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

関連情報

- **Cluster-monitoring-config** config map の設定リファレンス
- **user-workload-monitoring-config** config map の設定リファレンス
- モニタリング config map を作成する手順は、モニタリングスタックの設定の準備 を参照してください。
- ユーザー定義プロジェクトのモニタリングの有効化

2.5. 設定可能なモニタリングコンポーネント

以下の表は、設定可能なモニタリングコンポーネントと、**cluster-monitoring-config** および **user-workload-monitoring-config ConfigMap** オブジェクトでコンポーネントを指定するために使用されるキーを示しています。

表2.1 設定可能なモニタリングコンポーネント

コンポーネント	cluster-monitoring-config config map キー	user-workload-monitoring-config config map キー
Prometheus Operator	**prometheusOperator**	**prometheusOperator**
Prometheus	**prometheusK8s**	**prometheus**
Alertmanager	**alertmanagerMain**	**Alertmanager**
kube-state-metrics	**kubeStateMetrics**	
monitoring-plugin	**monitoringPlugin**	
openshift-state-metrics	**openshiftStateMetrics**	
Telemeter クライアント	**telemeterClient**	
Prometheus アダプター	**k8sPrometheusAdapter**	
Thanos Querier	**thanosQuerier**	

コンポーネント	cluster-monitoring-config config map キー	user-workload-monitoring-config config map キー
Thanos Ruler		**thanosRuler**

注記

Prometheus キーは、**cluster-monitoring-config ConfigMap** で **prometheusK8s** と呼ばれ、**user-workload-monitoring-config ConfigMap** オブジェクトで **prometheus** と呼ばれています。

2.6. ノードセレクターを使用したモニタリングコンポーネントの移動

ラベル付きノードで **nodeSelector** 制約を使用すると、任意のモニタリングスタックコンポーネントを特定ノードに移動できます。これにより、クラスター全体のモニタリングコンポーネントの配置と分散を制御できます。

モニタリングコンポーネントの配置と分散を制御することで、システムリソースの使用を最適化し、パフォーマンスを高め、特定の要件やポリシーに基づいてワークロードを分離できます。

2.6.1. ノードセレクターと他の制約の連携

ノードセレクターの制約を使用してモニタリングコンポーネントを移動する場合、クラスターに Pod のスケジューリングを制御するための他の制約があることに注意してください。

- Pod の配置を制御するために、トポロジー分散制約が設定されている可能性があります。
- Prometheus、Thanos Querier、Alertmanager、およびその他のモニタリングコンポーネントでは、コンポーネントの複数の Pod が必ず異なるノードに分散されて高可用性が常に確保されるように、ハードな非アフィニティールールが設定されています。

ノード上で Pod をスケジュールする場合、Pod スケジューラーは既存の制約をすべて満たすように Pod の配置を決定します。つまり、Pod スケジューラーがどの Pod をどのノードに配置するかを決定する際に、すべての制約が組み合わされます。

そのため、ノードセレクター制約を設定しても既存の制約をすべて満たすことができない場合、Pod スケジューラーはすべての制約をマッチさせることができず、ノードへの Pod 配置をスケジュールしません。

モニタリングコンポーネントの耐障害性と高可用性を維持するには、コンポーネントを移動するノードセレクター制約を設定する際に、十分な数のノードが利用可能で、すべての制約がマッチすることを確認してください。

関連情報

- ノードでラベルを更新する方法について
- ノードセレクターの使用による特定ノードへの Pod の配置
- アフィニティールールと非アフィニティールールの使用による他の Pod との相対での Pod の配置
- Pod トポロジー分散制約を使用した Pod 配置の制御

- モニタリングのための Pod トポロジー分散制約の設定
- ノードセレクターに関する Kubernetes ドキュメント

2.6.2. モニタリングコンポーネントの異なるノードへの移動

モニタリングスタックコンポーネントが実行されるクラスター内のノードを指定するには、ノードに割り当てられたラベルと一致するようにコンポーネントの **ConfigMap** オブジェクトの **nodeSelector** 制約を設定します。

注記

ノードセレクター制約を既存のスケジュール済み Pod に直接追加することはできません。

前提条件

- **OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合**、以下を実行します。
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. まだの場合は、モニタリングコンポーネントを実行するノードにラベルを追加します。

```
$ oc label nodes <node-name> <node-label>
```

2. **ConfigMap** オブジェクトを編集します。
   - **OpenShift Container Platform のコアプロジェクトをモニターするコンポーネントを移行するには**、以下を実行します。
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

     b. **data/config.yaml** でコンポーネントの **nodeSelector** 制約のノードラベルを指定します。

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    <component>: 1
      nodeSelector:
        <node-label-1> 2
        <node-label-2> 3
        <...>
```

1. **<component>** を適切なモニタリングスタックコンポーネント名に置き換えます。
2. **<node-label-1>** をノードに追加したラベルに置き換えます。
3. オプション: 追加のラベルを指定します。追加のラベルを指定すると、コンポーネントの Pod は、指定されたすべてのラベルを含むノード上でのみスケジュールされます。

注記

nodeSelector の制約を設定した後もモニタリングコンポーネントが **Pending** 状態のままになっている場合は、Pod イベントでテイントおよび容認に関連するエラーの有無を確認します。

- **ユーザー定義プロジェクトをモニターするコンポーネントを移動するには**、以下を実行します。
  a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

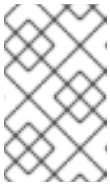
  b. **data/config.yaml** でコンポーネントの **nodeSelector** 制約のノードラベルを指定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>: 1
      nodeSelector:
        <node-label-1> 2
        <node-label-2> 3
        <...>
```

1. **<component>** を適切なモニタリングスタックコンポーネント名に置き換えます。

2 **<node-label-1>** をノードに追加したラベルに置き換えます。

3 オプション: 追加のラベルを指定します。追加のラベルを指定すると、コンポーネントの Pod は、指定されたすべてのラベルを含むノード上でのみスケジュールされます。



注記

nodeSelector の制約を設定した後もモニタリングコンポーネントが **Pending** 状態のままになっている場合は、Pod イベントでテイントおよび容認に関連するエラーの有無を確認します。

3. 変更を適用するためにファイルを保存します。新しい設定で指定されたコンポーネントは、新しいノードに自動的に移動されます。

注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

警告

monitoring config map への変更を保存すると、プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。そのプロジェクトで実行中のモニタリングプロセスも再起動する場合があります。

関連情報

- モニタリング config map を作成する手順は、モニタリングスタックの設定の準備 を参照してください。
- ユーザー定義プロジェクトのモニタリングの有効化
- ノードでラベルを更新する方法について
- ノードセレクターの使用による特定ノードへの Pod の配置
- **nodeSelector** 制約についての詳細は、Kubernetes ドキュメント を参照してください。

2.7. モニタリングコンポーネントへの容認 (TOLERATION) の割り当て

容認をモニタリングスタックのコンポーネントに割り当て、それらをテイントされたノードに移動することができます。

前提条件

- **OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合**、以下を実行します。

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **cluster-monitoring-configConfigMap** オブジェクトを作成している。

- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **容認をコア OpenShift Container Platform プロジェクトをモニターするコンポーネントに割り当てるには**、以下を実行します。
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

     b. コンポーネントの **tolerations** を指定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    <component>:
      tolerations:
        <toleration_specification>
```

     <component> および **<toleration_specification>** を随時置き換えます。

     たとえば、**oc adm taint nodes node1 key1=value1:NoSchedule** は、キーが **key1** で、値が **value1** の **node1** にテイントを追加します。これにより、モニタリングコンポーネントが **node1** に Pod をデプロイするのを防ぎます。ただし、そのテイントに対して許容値が設定されている場合を除きます。以下の例は、サンプルのテイントを容認するように **alertmanagerMain** コンポーネントを設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      tolerations:
```

```
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoSchedule"
```

- **ユーザー定義プロジェクトをモニターするコンポーネントに容認を割り当てるには**、以下を実行します。

  a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

  ```
  $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
  ```

  b. コンポーネントの **tolerations** を指定します。

  ```
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: user-workload-monitoring-config
    namespace: openshift-user-workload-monitoring
  data:
    config.yaml: |
      <component>:
        tolerations:
          <toleration_specification>
  ```

  <component> および **<toleration_specification>** を随時置き換えます。

  たとえば、**oc adm taint nodes node1 key1=value1:NoSchedule** は、キーが **key1** で、値が **value1** の **node1** にテイントを追加します。これにより、モニタリングコンポーネントが **node1** に Pod をデプロイするのを防ぎます。ただし、そのテイントに対して許容値が設定されている場合を除きます。以下の例では、サンプルのテイントを容認するように **thanosRuler** コンポーネントを設定します。

  ```
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: user-workload-monitoring-config
    namespace: openshift-user-workload-monitoring
  data:
    config.yaml: |
      thanosRuler:
        tolerations:
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoSchedule"
  ```

2. 変更を適用するためにファイルを保存します。新しいコンポーネントの配置設定が自動的に適用されます。

注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

警告

変更がモニタリング config map に保存されると、関連するプロジェクトの Pod およびその他のリソースが再デプロイされる可能性があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

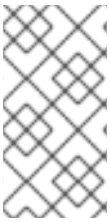
関連情報

- モニタリング config map を作成する手順は、モニタリングスタックの設定の準備 を参照してください。
- ユーザー定義プロジェクトのモニタリングの有効化
- テイントおよび許容値は、OpenShift Container Platform ドキュメント を参照してください。
- テイントおよび許容値については、Kubernetes ドキュメント を参照してください。

2.8. メトリクススクレイピング (収集) のボディーサイズ制限の設定

デフォルトでは、スクレイピングされたメトリックターゲットから返されるデータの圧縮されていない本文のサイズに制限はありません。スクレイピングされたターゲットが大量のデータを含む応答を返したときに、Prometheus が大量のメモリーを消費する状況を回避するために、ボディサイズの制限を設定できます。さらに、本体のサイズ制限を設定することで、悪意のあるターゲットが Prometheus およびクラスター全体に与える影響を軽減できます。

enforcedBodySizeLimit の値を設定した後、少なくとも 1 つの Prometheus スクレイプターゲットが、設定された値より大きい応答本文で応答すると、アラート **PrometheusScrapeBodySizeLimitHit** が発生します。



注記

ターゲットからスクレイピングされたメトリックデータの非圧縮ボディサイズが設定されたサイズ制限を超えていると、スクレイピングは失敗します。次に、Prometheus はこのターゲットがダウンしていると見なし、その **up** メトリック値を **0** に設定します。これにより、**TargetDown** アラートをトリガーできます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-monitoring** namespace で **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **enforcedBodySizeLimit** の値を **data/config.yaml/prometheusK8s** に追加して、ターゲットスクレイプごとに受け入れられるボディサイズを制限します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |-
    prometheusK8s:
      enforcedBodySizeLimit: 40MB (1)
```

(1) スクレイピングされたメトリックターゲットの最大ボディサイズを指定します。この **enforcedBodySizeLimit** の例では、ターゲットスクレイプごとの非圧縮サイズを 40 メガバイトに制限しています。有効な数値は、B (バイト)、KB (キロバイト)、MB (メガバイト)、GB (ギガバイト)、TB (テラバイト)、PB (ペタバイト)、および EB (エクサバイト) の Prometheus データサイズ形式を使用します。デフォルト値は **0** で、制限なしを指定します。値を **automatic** に設定して、クラスターの容量に基づいて制限を自動的に計算することもできます。

3. ファイルを保存して、変更を自動的に適用します。

警告

cluster-monitoring-config config map への変更を保存すると、**openshift-monitoring** プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。そのプロジェクトで実行中のモニタリングプロセスも再起動する場合があります。

関連情報

- Prometheus scrape configuration documentation

2.9. コンポーネントのモニタリングに使用する CPU およびメモリーリソースの管理

モニタリングコンポーネントを実行するコンテナーに十分な CPU リソースとメモリーリソースがあることを確認するには、これらのコンポーネントに対するリソース制限と要求の値を指定します。

これらの制限と要求は、**openshift-monitoring** namespace のコアプラットフォームモニタリングコンポーネント、および **openshift-user-workload-monitoring** namespace のユーザー定義プロジェクトを監視するコンポーネントに対して設定できます。

2.9.1. モニタリングコンポーネントの制限と要求の指定について

コアプラットフォームモニタリングコンポーネントと、次のコンポーネントを含むユーザー定義プロジェクトを監視するコンポーネントのリソース制限と要求を設定できます。

- Alertmanager (コアプラットフォームのモニタリングおよびユーザー定義プロジェクト用)
- kube-state-metrics
- monitoring-plugin
- node-exporter
- openshift-state-metrics
- Prometheus (コアプラットフォームのモニタリングおよびユーザー定義プロジェクト用)
- Prometheus アダプター
- Prometheus Operator とそのアドミッション Webhook サービス
- Telemeter クライアント
- Thanos Querier
- Thanos Ruler

リソース制限を定義すると、コンテナーのリソース使用量が制限され、コンテナーが CPU およびメモリーリソースの指定された最大値を超過しなくなります。

リソース要求を定義することで、要求されたリソースを満たすのに十分な CPU リソースとメモリーリソースが利用可能なノード上でのみコンテナーをスケジュールできるように指定します。

2.9.2. モニタリングコンポーネントの制限と要求の指定

CPU およびメモリーリソースを設定するには、モニタリングコンポーネントが配置されている namespace の適切な **ConfigMap** オブジェクトで、リソース制限と要求の値を指定します。

- コアプラットフォームのモニタリングに使用する **openshift-monitoring** namespace の **cluster-monitoring-config** config map
- ユーザー定義プロジェクトを関しするコンポーネントの **openshift-user-workload-monitoring** namespace 内の **user-workload-monitoring-config** config map

前提条件

- **コアプラットフォームモニタリングコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - これで、**cluster-monitoring-config** という名前の **ConfigMap** オブジェクトが作成されました。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:

- **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。

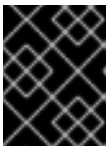
- OpenShift CLI (**oc**) がインストールされている。

手順

1. コアプラットフォームモニタリングコンポーネントを設定するには、**openshift-monitoring** namespace の **cluster-monitoring-config** config map オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. 値を追加して、設定する各コアプラットフォームモニタリングコンポーネントのリソース制限と要求を定義します。



重要

制限に設定された値が、常に要求に設定された値よりも大きいことを確認してください。そうでない場合、エラーが発生し、コンテナーは実行されません。

例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    prometheusK8s:
      resources:
        limits:
          cpu: 500m
          memory: 3Gi
        requests:
          cpu: 200m
          memory: 500Mi
    prometheusOperator:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    k8sPrometheusAdapter:
```

```
    resources:
      limits:
        cpu: 500m
        memory: 1Gi
      requests:
        cpu: 200m
        memory: 500Mi
  kubeStateMetrics:
    resources:
      limits:
        cpu: 500m
        memory: 1Gi
      requests:
        cpu: 200m
        memory: 500Mi
  telemeterClient:
    resources:
      limits:
        cpu: 500m
        memory: 1Gi
      requests:
        cpu: 200m
        memory: 500Mi
  openshiftStateMetrics:
    resources:
      limits:
        cpu: 500m
        memory: 1Gi
      requests:
        cpu: 200m
        memory: 500Mi
  thanosQuerier:
    resources:
      limits:
        cpu: 500m
        memory: 1Gi
      requests:
        cpu: 200m
        memory: 500Mi
  nodeExporter:
    resources:
      limits:
        cpu: 50m
        memory: 150Mi
      requests:
        cpu: 20m
        memory: 50Mi
  monitoringPlugin:
    resources:
      limits:
        cpu: 500m
        memory: 1Gi
      requests:
        cpu: 200m
        memory: 500Mi
  prometheusOperatorAdmissionWebhook:
```

```
    resources:
      limits:
        cpu: 50m
        memory: 100Mi
      requests:
        cpu: 20m
        memory: 50Mi
```

3. ファイルを保存して、変更を自動的に適用します。

重要

cluster-monitoring-config config map への変更を保存すると、**openshift-monitoring** プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。そのプロジェクトで実行中のモニタリングプロセスも再起動する場合があります。

関連情報

- Kubernetes の要求と制限に関するドキュメント

2.10. CONFIGURING PERSISTENT STORAGE

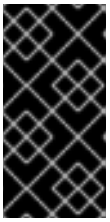
クラスターモニタリングを永続ストレージと共に実行すると、メトリクスは永続ボリューム (PV) に保存され、Pod の再起動または再作成後も維持されます。これは、メトリックデータまたはアラートデータをデータ損失から保護する必要がある場合に適しています。実稼働環境では、永続ストレージを設定することを強く推奨します。IO デマンドが高いため、ローカルストレージを使用することが有利になります。

2.10.1. 永続ストレージの前提条件

- ディスクが一杯にならないように、十分なローカル永続ストレージを確保します。必要な永続ストレージは Pod 数によって異なります。
- 永続ボリューム要求 (PVC) で要求される永続ボリューム (PV) が利用できる状態にあることを確認する必要があります。各レプリカに 1 つの PV が必要です。Prometheus と Alertmanager の両方に 2 つのレプリカがあるため、モニタリングスタック全体をサポートするには 4 つの PV が必要です。PV はローカルストレージ Operator から入手できますが、動的にプロビジョニングされるストレージが有効にされている場合は利用できません。
- 永続ボリュームを設定する際に、**volumeMode** パラメーターのストレージタイプ値として **Filesystem** を使用します。

注記

永続ストレージにローカルボリュームを使用する場合は、**LocalVolume** オブジェクトの **volumeMode: Block** で記述される raw ブロックボリュームを使用しないでください。Prometheus は raw ブロックボリュームを使用できません。



重要

Prometheus は、POSIX に準拠していないファイルシステムをサポートしません。たとえば、一部の NFS ファイルシステム実装は POSIX に準拠していません。ストレージに NFS ファイルシステムを使用する場合は、NFS 実装が完全に POSIX に準拠していることをベンダーに確認してください。

2.10.2. ローカル永続ボリューム要求 (PVC) の設定

モニタリングコンポーネントが永続ボリューム (PV) を使用できるようにするには、永続ボリューム要求 (PVC) を設定する必要があります。

前提条件

- **OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合**、以下を実行します。
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **OpenShift Container Platform のコアプロジェクトをモニターするコンポーネントの PVC を設定するには**、以下を実行します。
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

     b. コンポーネントの PVC 設定を **data/config.yaml** の下に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    <component>:
      volumeClaimTemplate:
        spec:
          storageClassName: <storage_class>
```

```
        resources:
          requests:
            storage: <amount_of_storage>
```

volumeClaimTemplate の指定方法については、PersistentVolumeClaims についての Kubernetes ドキュメント を参照してください。

以下の例では、OpenShift Container Platform のコアコンポーネントをモニターする Prometheus インスタンスのローカル永続ストレージを要求する PVC を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 40Gi
```

上記の例では、ローカルストレージ Operator によって作成されるストレージクラスは **local-storage** と呼ばれます。

以下の例では、Alertmanager のローカル永続ストレージを要求する PVC を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 10Gi
```

- **ユーザー定義プロジェクトをモニターするコンポーネントの PVC を設定するには**、以下を実行します。

  a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

b. コンポーネントの PVC 設定を **data/config.yaml** の下に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>:
      volumeClaimTemplate:
        spec:
          storageClassName: <storage_class>
          resources:
            requests:
              storage: <amount_of_storage>
```

volumeClaimTemplate の指定方法については、PersistentVolumeClaims についての Kubernetes ドキュメント を参照してください。

以下の例では、ユーザー定義プロジェクトをモニターする Prometheus インスタンスのローカル永続ストレージを要求する PVC を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 40Gi
```

上記の例では、ローカルストレージ Operator によって作成されるストレージクラスは **local-storage** と呼ばれます。

以下の例では、Thanos Ruler のローカル永続ストレージを要求する PVC を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
```

```
      resources:
        requests:
          storage: 10Gi
```

注記

thanosRuler コンポーネントのストレージ要件は、評価されルールの数や、各ルールが生成するサンプル数により異なります。

2. 変更を適用するためにファイルを保存します。新規設定の影響を受けた Pod は自動的に再起動され、新規ストレージ設定が適用されます。

注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

2.10.3. 永続ストレージボリュームのサイズ変更

OpenShift Container Platform は、使用される基礎となる **StorageClass** リソースが永続的なボリュームのサイズ変更をサポートしている場合でも、**StatefulSet** リソースによって使用される既存の永続的なストレージボリュームのサイズ変更をサポートしません。したがって、既存の永続ボリューム要求 (PVC) の **storage** フィールドをより大きなサイズで更新しても、この設定は関連する永続ボリューム (PV) に反映されません。

ただし、手動プロセスを使用して PV のサイズを変更することは可能です。Prometheus、Thanos Ruler、Alertmanager などのモニタリングコンポーネントの PV のサイズを変更する場合は、コンポーネントが設定されている適切な config map を更新できます。次に、PVC にパッチを適用し、Pod を削除して孤立させます。Pod を孤立させると、**StatefulSet** リソースがすぐに再作成され、Pod にマウントされたボリュームのサイズが新しい PVC 設定で自動的に更新されます。このプロセス中にサービスの中断は発生しません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合**、以下を実行します。
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
  - コア OpenShift Container Platform モニタリングコンポーネント用に少なくとも 1 つの PVC を設定しました。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。

- ユーザー定義プロジェクトを監視するコンポーネント用に少なくとも 1 つの PVC を設定しました。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **OpenShift Container Platform のコアプロジェクトをモニターするコンポーネントの PVC サイズを変更するには**、以下を実行します。
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

     b. **data/config.yaml** の下に、コンポーネントの PVC 設定用の新しいストレージサイズを追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    <component>: 1
      volumeClaimTemplate:
        spec:
          storageClassName: <storage_class> 2
          resources:
            requests:
              storage: <amount_of_storage> 3
```

     1 コアモニタリングコンポーネントを指定します。

     2 ストレージクラスを作成します。

     3 ストレージボリュームの新しいサイズを指定します。

     以下の例では、コア OpenShift Container Platform コンポーネントをモニターする Prometheus インスタンスのローカル永続ストレージを 100 ギガバイトに設定する PVC を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
```

```
        resources:
          requests:
            storage: 100Gi
```

次の例では、Alertmanager のローカル永続ストレージを 40 ギガバイトに設定する PVC を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 40Gi
```

- **ユーザー定義プロジェクトを監視するコンポーネントの PVC のサイズを変更するには**:

注記

ユーザー定義のプロジェクトを監視する Thanos Ruler および Prometheus インスタンスのボリュームサイズを変更できます。

a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

b. **data/config.yaml** 下のモニタリングコンポーネントの PVC 設定を更新します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>: 1
      volumeClaimTemplate:
        spec:
          storageClassName: <storage_class> 2
          resources:
            requests:
              storage: <amount_of_storage> 3
```

1 コアモニタリングコンポーネントを指定します。

2 ストレージクラスを作成します。

3 ストレージボリュームの新しいサイズを指定します。

次の例では、ユーザー定義のプロジェクトを監視する Prometheus インスタンスの PVC サイズを 100 ギガバイトに設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 100Gi
```

次の例では、Thanos Ruler の PVC サイズを 20 ギガバイトに設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 20Gi
```

注記

thanosRuler コンポーネントのストレージ要件は、評価されルールの数や、各ルールが生成するサンプル数により異なります。

2. 変更を適用するためにファイルを保存します。新規設定の影響を受けた Pod は自動的に再起動します。

警告

monitoring config map への変更を保存すると、関連プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。そのプロジェクトで実行しているモニタリングプロセスも再開される可能性があります。

3. 更新されたストレージ要求を使用して、すべての PVC に手動でパッチを適用します。以下の例では、**openshift-monitoring** namespace の Prometheus コンポーネントのストレージサイズを 100Gi に変更します。

```
$ for p in $(oc -n openshift-monitoring get pvc -l app.kubernetes.io/name=prometheus -o
jsonpath='{range .items[*]}{.metadata.name} {end}'); do \
  oc -n openshift-monitoring patch pvc/${p} --patch '{"spec": {"resources": {"requests":
{"storage":"100Gi"}}}}'; \
  done
```

4. **--cascade=orphan** パラメーターを使用して、基になる StatefulSet を削除します。

```
$ oc delete statefulset -l app.kubernetes.io/name=prometheus --cascade=orphan
```

2.10.4. Prometheus メトリックデータの保持期間およびサイズの変更

デフォルトで、Prometheus がメトリクスデータを保持する期間のデフォルトは以下のとおりです。

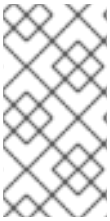
- **コアプラットフォームのモニタリング**: 15 日間
- **ユーザー定義プロジェクトの監視**: 24 時間

Prometheus の保持時間を変更して、データが削除されるまでの時間を変更できます。保持されるメトリクスデータが使用するディスク容量の最大量を設定することもできます。データがこのサイズ制限に達すると、使用するディスク領域が上限を下回るまで、Prometheus は最も古いデータを削除します。

これらのデータ保持設定は、以下の挙動に注意してください。

- サイズベースのリテンションポリシーは、**/prometheus** ディレクトリー内のすべてのデータブロックディレクトリーに適用され、永続ブロック、ライトアヘッドログ (WAL) データ、および m-mapped チャンクも含まれます。
- **wal** と **/head_chunks** ディレクトリーのデータは保持サイズ制限にカウントされますが、Prometheus はサイズまたは時間ベースの保持ポリシーに基づいてこれらのディレクトリーからデータをパージすることはありません。したがって、**/wal** ディレクトリーおよび **/head_chunks** ディレクトリーに設定された最大サイズよりも低い保持サイズ制限を設定すると、**/prometheus** データディレクトリーにデータブロックを保持しないようにシステムを設定している。
- サイズベースの保持ポリシーは、Prometheus が新規データブロックをカットする場合にのみ適用されます。これは、WAL に少なくとも 3 時間のデータが含まれてから 2 時間ごとに実行されます。

- **retention** または **retentionSize** の値を明示的に定義しない場合、保持期間のデフォルトは、コアプラットフォームの監視は 15 日間、ユーザー定義プロジェクトの監視は 24 時間です。保持サイズは設定されていません。
- **retention** および **retentionSize** の両方に値を定義すると、両方の値が適用されます。データブロックが定義された保持時間または定義されたサイズ制限を超える場合、Prometheus はこれらのデータブロックをパージします。
- **retentionSize** の値を定義して **retention** を定義しない場合、 **retentionSize** 値のみが適用されます。
- **retentionSize** の値を定義しておらず、p **retention** の値のみを定義する場合、**retention** 値のみが適用されます。
- **retentionSize** または **retention** の値を **0** に設定すると、デフォルト設定が適用されます。保持期間のデフォルト設定は、コアプラットフォームの監視の場合は 15 日間、ユーザー定義プロジェクトの監視の場合は 24 時間です。デフォルトでは、保持サイズは設定されていません。



注記

データコンパクションは 2 時間ごとに実行されます。そのため、コンパクションが実行される前に永続ボリューム (PV) がいっぱいになり、**retentionSize** 制限を超える可能性があります。その場合、PV 上のスペースが **retentionSize** 制限を下回るまで、**KubePersistentVolumeFillingUp** アラートが発生します。

前提条件

- OpenShift Container Platform **のコアモニタリングコンポーネントを設定する場合**、以下を実行します。
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - クラスター管理者は、ユーザー定義プロジェクトのモニタリングを有効にしている。
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **ConfigMap** オブジェクトを編集します。
   - OpenShift Container Platform **のコアプロジェクトをモニターする Prometheus インスタンスの保持時間とサイズを変更するに** は、以下を実行します。
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

b. 保持期間およびサイズ設定を **data/config.yaml** に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      retention: <time_specification> 1
      retentionSize: <size_specification> 2
```

1 保持時間: **ms** (ミリ秒)、**s** (秒)、**m** (分)、**h** (時)、**d** (日)、**w** (週)、**y** (年) が直接続く数値。**1h30m15s** などの特定の時間に時間値を組み合わせることもできます。

2 保持サイズ: **B** (バイト)、**KB** (キロバイト)、**MB** (メガバイト)、**GB** (ギガバイト)、**TB** (テラバイト)、**PB** (ペタバイト)、および **EB** (エクサバイト) が直接続く数値。

以下の例では、OpenShift Container Platform のコアコンポーネントをモニターする Prometheus インスタンスの保持期間を 24 時間に設定し、保持サイズを 10 ギガバイトに設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      retention: 24h
      retentionSize: 10GB
```

- **ユーザー定義プロジェクトをモニターする Prometheus インスタンスの保持時間とサイズを変更するには**、以下を実行します。

a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

b. 保持期間およびサイズ設定を **data/config.yaml** に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

```
prometheus:
  retention: <time_specification> 1
  retentionSize: <size_specification> 2
```

1 保持時間: **ms** (ミリ秒)、**s** (秒)、**m** (分)、**h** (時)、**d** (日)、**w** (週)、**y** (年) が直接続く数値。**1h30m15s** などの特定の時間に時間値を組み合わせることもできます。

2 保持サイズ: **B** (バイト)、**KB** (キロバイト)、**MB** (メガバイト)、**GB** (ギガバイト)、**TB** (テラバイト)、**PB** (ペタバイト)、または **EB** (エクサバイト) が直接続く数値。

次の例では、ユーザー定義プロジェクトを監視する Prometheus インスタンスについて、保持時間を 24 時間に、保持サイズを 10 ギガバイトに設定しています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      retention: 24h
      retentionSize: 10GB
```

2. 変更を適用するためにファイルを保存します。新規設定の影響を受けた Pod は自動的に再起動します。

警告

変更がモニタリング config map に保存されると、関連するプロジェクトの Pod およびその他のリソースが再デプロイされる可能性があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

2.10.5. Thanos Ruler メトリックデータの保持期間の変更

デフォルトでは、ユーザー定義のプロジェクトでは、Thanos Ruler は 24 時間にわたりメトリックデータを自動的に保持します。**openshift-user-workload-monitoring** namespace の **user-workload-monitoring-config** の Config Map に時間の値を指定して、このデータの保持期間を変更できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
- クラスター管理者は、ユーザー定義プロジェクトのモニタリングを有効にしている。

- **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. 保持期間の設定を **data/config.yaml** に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      retention: <time_specification> 1
```

1 保持時間 は、**ms** (ミリ秒)、**s** (秒)、**m** (分)、**h** (時)、**d** (日)、**w** (週)、**y** (年) が直後に続く数字で指定します。**1h30m15s** などの特定の時間に時間値を組み合わせることもできます。デフォルトは **24h** です。

以下の例では、Thanos Ruler データの保持期間を 10 日間に設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      retention: 10d
```

3. 変更を適用するためにファイルを保存します。新規設定が加えられた Pod は自動的に再起動します。

警告

モニタリング config map の変更を保存すると、モニタリングプロセスが再起動し、関連プロジェクトの Pod やその他のリソースが再デプロイされる場合があります。そのプロジェクトで実行中のモニタリングプロセスも再起動する場合があります。

関連情報

- クラスターモニタリング config map の作成
- Prometheus データベースのストレージ要件
- 設定可能な推奨のストレージ技術
- 永続ストレージについて
- ストレージの最適化
- ローカル永続ストレージの設定
- ユーザー定義プロジェクトのモニタリングの有効化

2.11. リモート書き込みストレージの設定

リモート書き込みストレージを設定して、Prometheus が取り込んだメトリックをリモートシステムに送信して長期保存できるようにします。これを行っても、Prometheus がメトリックを保存する方法や期間には影響はありません。

前提条件

- **OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合、以下を実行します。**
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。
- リモート書き込み互換性のあるエンドポイント (Thanos) を設定し、エンドポイント URL を把握している。リモート書き込み機能と互換性のないエンドポイントの情報では、Prometheus リモートエンドポイントおよびストレージについてのドキュメント を参照してください。

重要

Red Hat は、リモート書き込み送信側の設定に関する情報のみを提供し、受信側エンドポイントの設定に関するガイダンスは提供しません。お客様は、リモート書き込みと互換性のある独自のエンドポイントを設定する責任があります。エンドポイントレシーバー設定に関する問題は、Red Hat 製品サポートには含まれません。

- リモート書き込みエンドポイントの **Secret** オブジェクトに認証クレデンシャルを設定している。リモート書き込みを設定する Prometheus オブジェクトと同じ namespace にシークレットを作成する必要があります。デフォルトのプラットフォームモニタリングの場合は **openshift-**

monitoring namespace 、ユーザーのワークロードモニタリングの場合は **openshift-user-workload-monitoring** namespace です。

警告

セキュリティーリスクを軽減するには、HTTPS および認証を使用してメトリックをエンドポイントに送信します。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **コア OpenShift Container Platform プロジェクトをモニターする Prometheus インスタンスのリモート書き込みを設定するには、次の手順を実行します。**
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

     b. **data/config.yaml/prometheusK8s** に **remoteWrite:** セクションを追加します。

     c. このセクションにエンドポイント URL および認証情報を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com" 1
        <endpoint_authentication_credentials> 2
```

     1 リモート書き込みエンドポイントの URL。

     2 エンドポイントの認証方法およびクレデンシャル。現在サポートされている認証方法は、AWS 署名バージョン 4、**Authorization** リクエストヘッダーでの HTTP を使用した認証、基本認証、OAuth 2.0、および TLS クライアントです。**サポートされている認証方法のサンプル設定については、サポートされているリモート書き込み認証設定** を参照してください。

     d. 認証クレデンシャルの後に、書き込みの再ラベル設定値を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
```

```
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        <endpoint_authentication_credentials>
        <your_write_relabel_configs> 1
```

1 書き込みの再ラベル設定。

<your_write_relabel_configs> は、リモートエンドポイントに送信する必要のあるメトリクスの書き込みラベル一覧に置き換えます。

以下の例では、**my_metric** という単一のメトリックを転送する方法を紹介します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        writeRelabelConfigs:
        - sourceLabels: [__name__]
          regex: 'my_metric'
          action: keep
```

書き込み再ラベル設定オプションについては、Prometheus relabel_config documentation を参照してください。

- **ユーザー定義プロジェクトをモニターする Prometheus インスタンスのリモート書き込みを設定するには、次の手順を実行します。**

  a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

  ```
  $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
  ```

  b. **data/config.yaml/prometheus** に **remoteWrite:** セクションを追加します。

  c. このセクションにエンドポイント URL および認証情報を追加します。

  ```
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: user-workload-monitoring-config
    namespace: openshift-user-workload-monitoring
  data:
    config.yaml: |
      prometheus:
  ```

```
    remoteWrite:
    - url: "https://remote-write-endpoint.example.com" (1)
      <endpoint_authentication_credentials> (2)
```

(1) リモート書き込みエンドポイントの URL。

(2) エンドポイントの認証方法およびクレデンシャル。現在サポートされている認証方式は、AWS Signature Version 4、HTTP an **Authorization** リクエストヘッダーを用いた認証、Basic 認証、OAuth 2.0、TLS client です。サポートされる認証方法の設定例は、以下の**サポート対象のリモート書き込み認証設定**を参照してください。

d. 認証クレデンシャルの後に、書き込みの再ラベル設定値を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        <endpoint_authentication_credentials>
        <your_write_relabel_configs> (1)
```

(1) 書き込みの再ラベル設定。

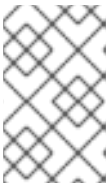
<your_write_relabel_configs> は、リモートエンドポイントに送信する必要のあるメトリクスの書き込みラベル一覧に置き換えます。

以下の例では、**my_metric** という単一のメトリックを転送する方法を紹介します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        writeRelabelConfigs:
        - sourceLabels: [__name__]
          regex: 'my_metric'
          action: keep
```

書き込み再ラベル設定オプションについては、Prometheus relabel_config documentation を参照してください。

2. 変更を適用するためにファイルを保存します。新規設定の影響を受けた Pod は自動的に再起動します。



注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

警告

モニタリング **ConfigMap** オブジェクトへの変更を保存すると、関連するプロジェクトの Pod およびその他のリソースが再デプロイされる可能性があります。また、変更を保存すると、そのプロジェクトで実行中のモニタリングプロセスも再起動する可能性があります。

2.11.1. サポート対象のリモート書き込み認証設定

異なる方法を使用して、リモート書き込みエンドポイントとの認証を行うことができます。現時点でサポートされている認証方法は AWS 署名バージョン 4、Basic 認証、認可、OAuth 2.0、および TLS クライアントです。以下の表は、リモート書き込みで使用するサポート対象の認証方法の詳細を示しています。

認証方法	config map フィールド	説明
AWS 署名バージョン 4	**sigv4**	この方法では、AWS Signature Version 4 認証を使用して要求を署名します。この方法は、認可、OAuth 2.0、または Basic 認証と同時に使用することはできません。
Basic 認証	**basicAuth**	Basic 認証は、設定されたユーザー名とパスワードを使用してすべてのリモート書き込み要求に承認ヘッダーを設定します。
認可	**認可**	Authorization は、設定されたトークンを使用して、すべてのリモート書き込みリクエストに **Authorization** ヘッダーを設定します。

認証方法	config map フィールド	説明
OAuth 2.0	**oauth2**	OAuth 2.0 設定は、クライアントクレデンシャル付与タイプを使用します。Prometheus は、リモート書き込みエンドポイントにアクセスするために、指定されたクライアント ID およびクライアントシークレットを使用して **tokenUrl** からアクセストークンを取得します。この方法を認可、AWS 署名バージョン 4、または基本認証と同時に使用することはできません。
TLS クライアント	**tlsConfig**	TLS クライアント設定は、TLS を使用してリモート書き込みエンドポイントサーバーで認証するために使用される CA 証明書、クライアント証明書、およびクライアントキーファイル情報を指定します。設定例は、CA 証明書ファイル、クライアント証明書ファイル、およびクライアントキーファイルがすでに作成されていることを前提としています。

2.11.2. リモート書き込み認証の設定例

次のサンプルは、リモート書き込みエンドポイントに接続するために使用できるさまざまな認証設定を示しています。各サンプルでは、認証情報やその他の関連設定を含む対応する **Secret** オブジェクトを設定する方法も示しています。それぞれのサンプルは、**openshift-monitoring** namespace でデフォルトのプラットフォームモニタリングで使用する認証を設定します。

例2.1 AWS 署名バージョン 4 認証のサンプル YAML

以下は、**openshift-monitoring** namespace の **sigv4-credentials** という名前の **sigv 4** シークレットの設定を示しています。

```
apiVersion: v1
kind: Secret
metadata:
  name: sigv4-credentials
  namespace: openshift-monitoring
stringData:
  accessKey: <AWS_access_key> 1
  secretKey: <AWS_secret_key> 2
type: Opaque
```

1 AWS API アクセスキー。

2 AWS API シークレットキー。

以下は、**openshift-monitoring** namespace の **sigv4-credentials** という名前の **Secret** オブジェクトを使用する AWS Signature Version 4 リモート書き込み認証のサンプルを示しています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
      - url: "https://authorization.example.com/api/write"
        sigv4:
          region: <AWS_region> 1
          accessKey:
            name: sigv4-credentials 2
            key: accessKey 3
          secretKey:
            name: sigv4-credentials 4
            key: secretKey 5
          profile: <AWS_profile_name> 6
          roleArn: <AWS_role_arn> 7
```

1 AWS リージョン。

2 4 AWS API アクセスクレデンシャルが含まれる **Secret** オブジェクトの名前。

3 指定された **Secret** オブジェクトに AWS API アクセスキーが含まれるキー。

5 指定された **Secret** オブジェクトに AWS API シークレットキーが含まれるキー。

6 認証に使用される AWS プロファイルの名前。

7 ロールに割り当てられた Amazon Resource Name (ARN) の一意の識別子。

例2.2 Basic 認証のサンプル YAML

以下は、**openshift-monitoring** namespace の **rw-basic-auth** という名前の **Secret** オブジェクトの基本的な認証設定例を示しています。

```
apiVersion: v1
kind: Secret
metadata:
  name: rw-basic-auth
  namespace: openshift-monitoring
stringData:
  user: <basic_username> 1
  password: <basic_password> 2
type: Opaque
```

1 ユーザー名

2 パスワード。

以下の例は、**openshift-monitoring** namespace の **rw-basic-auth** という名前の **Secret** オブジェクトを使用する **basicAuth** リモート書き込み設定を示しています。これは、エンドポイントの認証認証情報がすでに設定されていることを前提としています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
      - url: "https://basicauth.example.com/api/write"
        basicAuth:
          username:
            name: rw-basic-auth 1
            key: user 2
          password:
            name: rw-basic-auth 3
            key: password 4
```

1 3 認証クレデンシャルが含まれる **Secret** オブジェクトの名前。

2 指定の **Secret** オブジェクトのユーザー名が含まれるキー。

4 指定された **Secret** オブジェクトにパスワードが含まれるキー。

例2.3 Secret オブジェクトを使用したベアラートークンによる認証のサンプル YAML

以下は、**openshift-monitoring** namespace の **rw-bearer-auth** という名前の **Secret** オブジェクトのベアラートークン設定を示しています。

```
apiVersion: v1
kind: Secret
metadata:
  name: rw-bearer-auth
  namespace: openshift-monitoring
stringData:
  token: <authentication_token> 1
type: Opaque
```

1 認証トークン。

以下は、**openshift-monitoring** namespace の **rw-bearer-auth** という名前の **Secret** オブジェクトを使用するベアラートークン config map の設定例を示しています。

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
    prometheusK8s:
      remoteWrite:
      - url: "https://authorization.example.com/api/write"
        authorization:
          type: Bearer (1)
          credentials:
            name: rw-bearer-auth (2)
            key: token (3)
```

1. 要求の認証タイプ。デフォルト値は **Bearer** です。
2. 認証クレデンシャルが含まれる **Secret** オブジェクトの名前。
3. 指定された **Secret** オブジェクトに認証トークンが含まれるキー。

例2.4 OAuth 2.0 認証のサンプル YAML

以下は、**openshift-monitoring** namespace の **oauth2-credentials** という名前の **Secret** オブジェクトの OAuth 2.0 設定のサンプルを示しています。

```
apiVersion: v1
kind: Secret
metadata:
  name: oauth2-credentials
  namespace: openshift-monitoring
stringData:
  id: <oauth2_id> (1)
  secret: <oauth2_secret> (2)
  token: <oauth2_authentication_token> (3)
type: Opaque
```

1. Oauth 2.0 ID。
2. OAuth 2.0 シークレット。
3. OAuth 2.0 トークン。

以下は、**openshift-monitoring** namespace の **oauth2-credentials** という **Secret** オブジェクトを使用した **oauth2** リモート書き込み認証のサンプル設定です。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
```

```
config.yaml: |
  prometheusK8s:
    remoteWrite:
    - url: "https://test.example.com/api/write"
      oauth2:
        clientId:
          secret:
            name: oauth2-credentials (1)
            key: id (2)
        clientSecret:
          name: oauth2-credentials (3)
          key: secret (4)
        tokenUrl: https://example.com/oauth2/token (5)
        scopes: (6)
        - <scope_1>
        - <scope_2>
        endpointParams: (7)
          param1: <parameter_1>
          param2: <parameter_2>
```

1 3 対応する **Secret** オブジェクトの名前。**ClientId** は **ConfigMap** オブジェクトを参照することもできますが、**clientSecret** は **Secret** オブジェクトを参照する必要があることに注意してください。

2 4 指定された **Secret** オブジェクトの OAuth 2.0 認証情報が含まれるキー。

5 指定された **clientId** および **clientSecret** でトークンを取得するために使用される URL。

6 認可要求の OAuth 2.0 スコープ。これらのスコープは、トークンがアクセスできるデータを制限します。

7 認可サーバーに必要な OAuth 2.0 認可要求パラメーター。

例2.5 TLS クライアント認証のサンプル YAML

以下は、**openshift-monitoring** namespace 内の **mtls-bundle** という名前の **tlsSecret** オブジェクトに対する TLS クライアント設定のサンプルです。

```
apiVersion: v1
kind: Secret
metadata:
  name: mtls-bundle
  namespace: openshift-monitoring
data:
  ca.crt: <ca_cert> (1)
  client.crt: <client_cert> (2)
  client.key: <client_key> (3)
type: tls
```

1 サーバー証明書を検証する Prometheus コンテナーの CA 証明書。

2 サーバーとの認証用のクライアント証明書。

3 クライアントキー。

以下の例は、**mtls-bundle** という名前の TLS **Secret** オブジェクトを使用する **tlsConfig** リモート書き込み認証設定を示しています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        tlsConfig:
          ca:
            secret:
              name: mtls-bundle 1
              key: ca.crt 2
          cert:
            secret:
              name: mtls-bundle 3
              key: client.crt 4
          keySecret:
            name: mtls-bundle 5
            key: client.key 6
```

1 3 5 TLS 認証クレデンシャルが含まれる対応する **Secret** オブジェクトの名前。**ca** と **cert** は、代わりに **ConfigMap** オブジェクトを参照することができますが、**keySecret** は **Secret** オブジェクトを参照する必要があることに注意してください。

2 エンドポイントの CA 証明書が含まれる指定された **Secret** オブジェクトのキー。

4 エンドポイントのクライアント証明書が含まれる指定された **Secret** オブジェクトのキー。

6 クライアントシークレットが含まれる指定の **Secret** オブジェクトのキー。

関連情報

- リモート書き込み互換性のあるエンドポイント (Thanos など) を作成する手順は、リモート書き込み互換性のあるエンドポイントの設定 を参照してください。
- 各種のユースケースごとのリモート書き込みの最適化方法は、リモート書き込みの設定 を参照してください。
- OpenShift Container Platform で **Secret** オブジェクトを作成および設定する手順については、シークレット を参照してください。
- 追加のオプションフィールドは、リモート書き込みの Prometheus REST API リファレンス を参照してください。

2.12. クラスター ID ラベルのメトリックへの追加

複数の OpenShift Container Platform クラスターを管理し、リモート書き込み機能を使用してメトリックデータをこれらのクラスターから外部ストレージの場所に送信する場合、クラスター ID ラベルを追加して、異なるクラスターから送られるメトリックデータを特定できます。次に、これらのラベルをクエリーし、メトリックのソースクラスターを特定し、そのデータを他のクラスターによって送信される同様のメトリックデータと区別することができます。

これにより、複数の顧客に対して多数のクラスターを管理し、メトリックデータを単一の集中ストレージシステムに送信する場合、クラスター ID ラベルを使用して特定のクラスターまたはお客様のメトリックをクエリーできます。

クラスター ID ラベルの作成および使用には、以下の 3 つの一般的な手順が必要です。

- リモート書き込みストレージの書き込みラベルの設定。
- クラスター ID ラベルをメトリックに追加します。
- これらのラベルをクエリーし、メトリックのソースクラスターまたはカスタマーを特定します。

2.12.1. メトリックのクラスター ID ラベルの作成

デフォルトプラットフォームのモニタリングおよびユーザーワークロードモニタリングのメトリクスのクラスター ID ラベルを作成できます。

デフォルトのプラットフォームモニタリングの場合、**openshift-monitoring** namespace の **cluster-monitoring-config** config map でリモート書き込みストレージの **write_relabel** 設定でメトリクスのクラスター ID ラベルを追加します。

ユーザーワークロードモニタリングの場合、**openshift-user-workload-monitoring** namespace の **user-workload-monitoring-config** config map の設定を編集します。

注記

Prometheus が **namespace** ラベルを公開するユーザーワークロードターゲットをスクレイプすると、システムはこのラベルを **exported_namespace** として保存します。この動作により、最終的な namespace ラベル値がターゲット Pod の namespace と等しくなります。このデフォルトは、**PodMonitor** または **ServiceMonitor** オブジェクトの **honorLabels** フィールドの値を **true** に設定してオーバーライドすることはできません。

前提条件

- **デフォルトのプラットフォームモニタリングコンポーネントを設定する場合は、以下を実行します。**
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。

- **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。

- OpenShift CLI (**oc**) がインストールされている。
- リモート書き込みストレージを設定している。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **コア OpenShift Container Platform メトリクスのクラスター ID ラベルを作成するには、次の手順を実行します。**
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

     b. **data/config.yaml/prometheusK8s/remoteWrite** の下にある **writeRelabelConfigs:** セクションで、クラスター ID の再ラベル付け設定値を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        <endpoint_authentication_credentials>
        writeRelabelConfigs: 1
          - <relabel_config> 2
```

     1 リモートエンドポイントに送信するメトリックの書き込み再ラベル付け設定のリストを追加します。

     2 リモート書き込みエンドポイントに送信されるメトリックのラベル設定を置き換えます。

     以下の例は、デフォルトのプラットフォームモニタリングでクラスター ID ラベル **cluster_id** を持つメトリクスを転送する方法を示しています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        writeRelabelConfigs:
```

```
    - sourceLabels:
      - __tmp_openshift_cluster_id__ 1
      targetLabel: cluster_id 2
      action: replace 3
```

1 システムは最初に **__tmp_openshift_cluster_id__** という名前の一時的なクラスター ID ソースラベルを適用します。この一時的なラベルは、指定するクラスター ID ラベル名に置き換えられます。

2 リモート書き込みストレージに送信されるメトリックのクラスター ID ラベルの名前を指定します。メトリックにすでに存在するラベル名を使用する場合、その値はこのクラスター ID ラベルの名前で上書きされます。ラベル名には **__tmp_openshift_cluster_id__** は使用しないでください。最後の再ラベル手順では、この名前を使用するラベルを削除します。

3 **replace** 置き換えラベルの再設定アクションは、一時ラベルを送信メトリックのターゲットラベルに置き換えます。このアクションはデフォルトであり、アクションが指定されていない場合に適用されます。

- **ユーザー定義のプロジェクトメトリックのクラスター ID ラベルを作成するには、次の手順を実行します。**

  a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

  ```
  $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
  ```

  b. **data/config.yaml/prometheus/remoteWrite** の下にある **writeRelabelConfigs:** セクションで、クラスター ID の再ラベル付け設定値を追加します。

  ```
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: user-workload-monitoring-config
    namespace: openshift-user-workload-monitoring
  data:
    config.yaml: |
      prometheus:
        remoteWrite:
        - url: "https://remote-write-endpoint.example.com"
          <endpoint_authentication_credentials>
          writeRelabelConfigs: 1
            - <relabel_config> 2
  ```

  1 リモートエンドポイントに送信するメトリックの書き込み再ラベル付け設定のリストを追加します。

  2 リモート書き込みエンドポイントに送信されるメトリックのラベル設定を置き換えます。

  次のサンプルは、ユーザーワークロードのモニタリングでクラスター ID ラベル **cluster_id** を持つメトリックを転送する方法を示しています。

  -

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        writeRelabelConfigs:
        - sourceLabels:
          - __tmp_openshift_cluster_id__ 1
          targetLabel: cluster_id 2
          action: replace 3
```

1. システムは最初に **__tmp_openshift_cluster_id__** という名前の一時的なクラスター ID ソースラベルを適用します。この一時的なラベルは、指定するクラスター ID ラベル名に置き換えられます。

2. リモート書き込みストレージに送信されるメトリックのクラスター ID ラベルの名前を指定します。メトリックにすでに存在するラベル名を使用する場合、その値はこのクラスター ID ラベルの名前で上書きされます。ラベル名には **__tmp_openshift_cluster_id__** は使用しないでください。最後の再ラベル手順では、この名前を使用するラベルを削除します。

3. **replace** 置き換えラベルの再設定アクションは、一時ラベルを送信メトリックのターゲットラベルに置き換えます。このアクションはデフォルトであり、アクションが指定されていない場合に適用されます。

2. ファイルを保存して、変更を **ConfigMap** オブジェクトに適用します。更新された設定の影響を受ける Pod は自動的に再起動します。

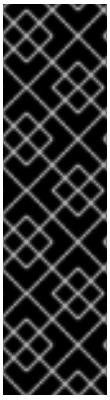
警告

モニタリング **ConfigMap** オブジェクトへの変更を保存すると、関連するプロジェクトの Pod およびその他のリソースが再デプロイされる可能性があります。また、変更を保存すると、そのプロジェクトで実行中のモニタリングプロセスも再起動する可能性があります。

関連情報

- 書き込みリラベル設定の詳細は、リモート書き込みストレージの設定 を参照してください。
- クラスター ID の取得方法について、詳細は クラスター ID の取得 を参照してください。

2.13. メトリック収集プロファイルの設定



重要

メトリックコレクションプロファイルの使用は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

デフォルトでは、Prometheus は OpenShift Container Platform コンポーネントのすべてのデフォルトのメトリックターゲットによって公開されたメトリックを収集します。ただし、特定のシナリオでは、Prometheus がクラスターから収集するメトリックを少なくしたい場合があります。

- クラスター管理者がアラート、テレメトリー、およびコンソールメトリックのみを必要とし、他のメトリックを使用可能にする必要がない場合。
- クラスターのサイズが大きくなり、収集されるデフォルトのメトリックデータのサイズが大きくなった場合、CPU とメモリーリソースを大幅に増やす必要があります。

メトリック収集プロファイルを使用して、デフォルトの量のメトリックデータまたは最小量のメトリックデータを収集できます。最小限のメトリックデータを収集すると、アラートなどの基本的なモニタリング機能は引き続き機能します。同時に、Prometheus が必要とする CPU およびメモリーリソースが減少します。

2.13.1. メトリック収集プロファイルについて

次の 2 つのメトリック収集プロファイルのいずれかを有効にできます。

- **full**: Prometheus は、すべてのプラットフォームコンポーネントによって公開されるメトリックデータを収集します。この設定がデフォルトです。
- **maximum**: Prometheus は、プラットフォームアラート、記録ルール、テレメトリー、およびコンソールダッシュボードに必要なメトリックデータのみを収集します。

2.13.2. メトリック収集プロファイルの選択

コア OpenShift Container Platform モニタリングコンポーネントのメトリック収集プロファイルを選択するには、**cluster-monitoring-config ConfigMap** オブジェクトを編集します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **FeatureGate** カスタムリソース (CR) を使用して、テクノロジープレビュー機能を有効にしました。
- **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。

警告

モニタリング config map の変更を保存すると、モニタリングプロセスが再起動し、関連プロジェクトの Pod やその他のリソースが再デプロイされる場合があります。そのプロジェクトで実行中のモニタリングプロセスも再起動する場合があります。

手順

1. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **data/config.yaml/prometheusK8s** の下にメトリック収集プロファイル設定を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      collectionProfile: <metrics_collection_profile_name> 1
```

1 メトリック収集プロファイルの名前。使用可能な値は **full** または **minimum** です。値を指定しない場合、または **collectionProfile** キー名が config map に存在しない場合は、デフォルト設定の **full** が使用されます。

次の例では、Prometheus のコアプラットフォームインスタンスのメトリックコレクションプロファイルを **minimal** に設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      collectionProfile: minimal
```

3. 変更を適用するためにファイルを保存します。新規設定の影響を受けた Pod は自動的に再起動します。

関連情報

- クラスターについて収集されているメトリックのリストを表示する手順については、使用可能なメトリックのリストの表示 を参照してください。
- テクノロジープレビュー機能を有効にする手順については、機能ゲートを使用して機能を有効にする を参照してください。

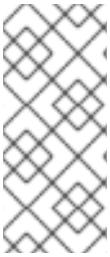
2.14. ユーザー定義プロジェクトでバインドされていないメトリクス属性の影響の制御

開発者は、キーと値のペアの形式でメトリックの属性を定義するためにラベルを作成できます。使用できる可能性のあるキーと値のペアの数は、属性について使用できる可能性のある値の数に対応します。数が無制限の値を持つ属性は、バインドされていない属性と呼ばれます。たとえば、**customer_id** 属性は、使用できる値が無限にあるため、バインドされていない属性になります。

割り当てられるキーと値のペアにはすべて、一意の時系列があります。ラベルに多数のバインドされていない値を使用すると、作成される時系列の数が指数関数的に増加する可能性があります。これは Prometheus のパフォーマンスに影響する可能性があり、多くのディスク領域を消費する可能性があります。

クラスター管理者は、以下の手段を使用して、ユーザー定義プロジェクトでのバインドされていないメトリクス属性の影響を制御できます。

- ユーザー定義プロジェクトでターゲット収集ごとに受け入れ可能なサンプル数を制限します。
- 収集されたラベルの数、ラベル名の長さ、およびラベル値の長さを制限します。
- 収集サンプルのしきい値に達するか、ターゲットを収集できない場合に実行されるアラートを作成します。



注記

収集サンプルを制限すると、多くのバインドされていない属性をラベルに追加して問題が発生するのを防ぐことができます。さらに開発者は、メトリクスに定義するバインドされていない属性の数を制限することにより、根本的な原因を防ぐことができます。使用可能な値の制限されたセットにバインドされる属性を使用すると、可能なキーと値のペアの組み合わせの数が減ります。

2.14.1. ユーザー定義プロジェクトの収集サンプルおよびラベル制限の設定

ユーザー定義プロジェクトで、ターゲット収集ごとに受け入れ可能なサンプル数を制限できます。収集されたラベルの数、ラベル名の長さ、およびラベル値の長さを制限することもできます。

警告

サンプルまたはラベルの制限を設定している場合、制限に達した後にそのターゲット収集についての追加のサンプルデータは取得されません。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
- ユーザー定義プロジェクトのモニタリングが有効になっている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. **enforcedSampleLimit** 設定を **data/config.yaml** に追加し、ユーザー定義プロジェクトのターゲットの収集ごとに受け入れ可能なサンプルの数を制限できます。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      enforcedSampleLimit: 50000 (1)
```

(1) このパラメーターが指定されている場合は、値が必要です。この **enforcedSampleLimit** の例では、ユーザー定義プロジェクトのターゲット収集ごとに受け入れ可能なサンプル数を 50,000 に制限します。

3. **enforcedLabelLimit**、**enforcedLabelNameLengthLimit**、および **enforcedLabelValueLengthLimit** 設定を **data/config.yaml** に追加し、収集されるラベルの数、ラベル名の長さ、およびユーザー定義プロジェクトでのラベル値の長さを制限します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      enforcedLabelLimit: 500 (1)
      enforcedLabelNameLengthLimit: 50 (2)
      enforcedLabelValueLengthLimit: 600 (3)
```

(1) 収集ごとのラベルの最大数を指定します。デフォルト値は **0** で、制限なしを指定します。

(2) ラベル名の最大長を指定します。デフォルト値は **0** で、制限なしを指定します。

(3) ラベル値の最大長を指定します。デフォルト値は **0** で、制限なしを指定します。

4. 変更を適用するためにファイルを保存します。制限は自動的に適用されます。

注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

警告

変更が **user-workload-monitoring-config ConfigMap** オブジェクトに保存されると、**openshift-user-workload-monitoring** プロジェクトの Pod および他のリソースは再デプロイされる可能性があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

2.14.2. 収集サンプルアラートの作成

以下の場合に通知するアラートを作成できます。

- ターゲットを収集できず、指定された **for** の期間利用できない
- 指定された **for** の期間、収集サンプルのしきい値に達するか、この値を上回る

前提条件

- **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
- ユーザー定義プロジェクトのモニタリングが有効になっている。
- **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- **enforcedSampleLimit** を使用して、ユーザー定義プロジェクトのターゲット収集ごとに受け入れ可能なサンプル数を制限している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. ターゲットがダウンし、実行されたサンプル制限に近づく際に通知するアラートを指定して YAML ファイルを作成します。この例のファイルは **monitoring-stack-alerts.yaml** という名前です。

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  labels:
    prometheus: k8s
```

```
    role: alert-rules
  name: monitoring-stack-alerts (1)
  namespace: ns1 (2)
spec:
  groups:
  - name: general.rules
    rules:
    - alert: TargetDown (3)
      annotations:
        message: '{{ printf "%.4g" $value }}% of the {{ $labels.job }}/{{ $labels.service
          }} targets in {{ $labels.namespace }} namespace are down.' (4)
      expr: 100 * (count(up == 0) BY (job, namespace, service) / count(up) BY (job,
        namespace, service)) > 10
      for: 10m (5)
      labels:
        severity: warning (6)
    - alert: ApproachingEnforcedSamplesLimit (7)
      annotations:
        message: '{{ $labels.container }} container of the {{ $labels.pod }} pod in the {{
$labels.namespace }} namespace consumes {{ $value | humanizePercentage }} of the
samples limit budget.' (8)
      expr: scrape_samples_scraped/50000 > 0.8 (9)
      for: 10m (10)
      labels:
        severity: warning (11)
```

(1) アラートルールの名前を定義します。

(2) アラートルールをデプロイするユーザー定義のプロジェクトを指定します。

(3) **TargetDown** アラートは、 **for** の期間にターゲットを収集できないか、利用できない場合に実行されます。

(4) **TargetDown** アラートが実行される場合に出力されるメッセージ。

(5) アラートが実行される前に、**TargetDown** アラートの条件がこの期間中 true である必要があります。

(6) **TargetDown** アラートの重大度を定義します。

(7) **ApproachingEnforcedSamplesLimit** アラートは、指定された **for** の期間に定義された収集サンプルのしきい値に達するか、この値を上回る場合に実行されます。

(8) **ApproachingEnforcedSamplesLimit** アラートの実行時に出力されるメッセージ。

(9) **ApproachingEnforcedSamplesLimit** アラートのしきい値。この例では、ターゲット収集ごとのサンプル数が実行されたサンプル制限 **50000** の 80% を超えるとアラートが実行されます。アラートが実行される前に、**for** の期間も経過している必要があります。式 **scrape_samples_scraped/<number> > <threshold>** の **<number>** は **user-workload-monitoring-config ConfigMap** オブジェクトで定義される **enforcedSampleLimit** 値に一致する必要があります。

(10) アラートが実行される前に、**ApproachingEnforcedSamplesLimit** アラートの条件がこの期間中 true である必要があります。

11 **ApproachingEnforcedSamplesLimit** アラートの重大度を定義します。

2. 設定をユーザー定義プロジェクトに適用します。

```
$ oc apply -f monitoring-stack-alerts.yaml
```

関連情報

- ユーザー定義のワークロードモニタリング config map の作成
- ユーザー定義プロジェクトのモニタリングの有効化
- 最高数の収集サンプルを持つメトリクスをクエリーする手順については、Prometheus が大量のディスク領域を消費している理由の特定 を参照してください。

第3章 外部 ALERTMANAGER インスタンスの設定

OpenShift Container Platform モニタリングスタックには、Prometheus からのアラートのルートなど、ローカルの Alertmanager インスタンスが含まれます。外部 Alertmanager インスタンスを追加して、コア OpenShift Container Platform プロジェクトまたはユーザー定義プロジェクトのアラートをルーティングできます。

複数のクラスターに同じ外部 Alertmanager 設定を追加し、クラスターごとにローカルインスタンスを無効にする場合には、単一の外部 Alertmanager インスタンスを使用して複数のクラスターのアラートルーティングを管理できます。

前提条件

- **openshift-monitoring プロジェクトで OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合:**
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-config**ConfigMap を作成している。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config**ConfigMap オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **OpenShift Container Platform のコアプロジェクトのルーティングアラート用に追加の Alertmanager を設定するには、以下を実行します。**
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config** ConfigMap を編集します。

     ```
     $ oc -n openshift-monitoring edit configmap cluster-monitoring-config
     ```

     b. **data/config.yaml/prometheusK8s** に **additionalAlertmanagerConfigs:** セクションを追加します。

     c. このセクションに別の Alertmanager 設定の詳細情報を追加します。

     ```
     apiVersion: v1
     kind: ConfigMap
     metadata:
       name: cluster-monitoring-config
       namespace: openshift-monitoring
     data:
       config.yaml: |
     ```

```
    prometheusK8s:
      additionalAlertmanagerConfigs:
      - <alertmanager_specification>
```

<alertmanager_specification> は、追加の Alertmanager インスタンスの認証およびその他の設定の詳細を置き換えます。現時点で、サポートされている認証方法はベアラートークン (**bearerToken**) およびクライアント TLS(**tlsConfig**) です。以下の config map は、クライアント TLS 認証でベアラートークンを使用して追加の Alertmanager を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      additionalAlertmanagerConfigs:
      - scheme: https
        pathPrefix: /
        timeout: "30s"
        apiVersion: v1
        bearerToken:
          name: alertmanager-bearer-token
          key: token
        tlsConfig:
          key:
            name: alertmanager-tls
            key: tls.key
          cert:
            name: alertmanager-tls
            key: tls.crt
          ca:
            name: alertmanager-tls
            key: tls.ca
        staticConfigs:
        - external-alertmanager1-remote.com
        - external-alertmanager1-remote2.com
```

- **ユーザー定義プロジェクトでルーティングアラート用に追加の Alertmanager インスタンスを設定するには、以下を実行します。**

  a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config** config map を編集します。

  ```
  $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
  ```

  b. **data/config.yaml/** の下に **<component>/additionalAlertmanagerConfigs:** セクションを追加します。

  c. このセクションに別の Alertmanager 設定の詳細情報を追加します。

  ```
  apiVersion: v1
  ```

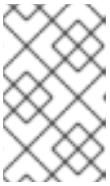
```
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>:
      additionalAlertmanagerConfigs:
      - <alertmanager_specification>
```

<component> には、サポート対象の外部 Alertmanager コンポーネント (**prometheus** または **thanosRuler**)2 つの内、いずれかに置き換えます。

<alertmanager_specification> は、追加の Alertmanager インスタンスの認証およびその他の設定の詳細を置き換えます。現時点で、サポートされている認証方法はベアラートークン (**bearerToken**) およびクライアント TLS(**tlsConfig**) です。以下の config map は、ベアラートークンおよびクライアント TLS 認証を指定した Thanos Ruler を使用して追加の Alertmanager を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      additionalAlertmanagerConfigs:
      - scheme: https
        pathPrefix: /
        timeout: "30s"
        apiVersion: v1
        bearerToken:
          name: alertmanager-bearer-token
          key: token
        tlsConfig:
          key:
            name: alertmanager-tls
            key: tls.key
          cert:
            name: alertmanager-tls
            key: tls.crt
          ca:
            name: alertmanager-tls
            key: tls.ca
        staticConfigs:
        - external-alertmanager1-remote.com
        - external-alertmanager1-remote2.com
```

2. ファイルを保存して、変更を **ConfigMap** オブジェクトに適用します。新しいコンポーネントの配置設定が自動的に適用されます。



注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

3. ファイルを保存して、変更を **ConfigMap** オブジェクトに適用します。新しいコンポーネントの配置設定が自動的に適用されます。

第4章 ALERTMANAGER のシークレットの設定

OpenShift Container Platform モニタリングスタックには、アラートを Prometheus からエンドポイントレシーバーにルーティングする Alertmanager が含まれています。Alertmanager がアラートを送信できるようにレシーバーで認証する必要がある場合は、レシーバーの認証認証情報を含むシークレットを使用するように Alertmanager を設定できます。

たとえば、シークレットを使用して、プライベート認証局 (CA) によって発行された証明書を必要とするエンドポイント受信者を認証するように Alertmanager を設定できます。また、基本 HTTP 認証用のパスワードファイルを必要とする受信者で認証するためにシークレットを使用するように Alertmanager を設定することもできます。いずれの場合も、認証の詳細は **ConfigMap** オブジェクトではなく **Secret** オブジェクトに含まれています。

4.1. ALERTMANAGER 設定へのシークレットの追加

openshift-monitoring プロジェクトの **cluster-monitoring-config** config map を編集することで、コアプラットフォームモニタリングコンポーネントの Alertmanager 設定にシークレットを追加できます。

config map にシークレットを追加すると、シークレットは、Alertmanager Pod の **alertmanager** コンテナー内の **/etc/alertmanager/secrets/<secret_name>** にボリュームとしてマウントされます。

前提条件

- **openshift-monitoring プロジェクトで OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合:**
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-config**ConfigMap を作成している。
  - **openshift-monitoring** プロジェクトの Alertmanager で設定するシークレットを作成しました。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - クラスター管理者は、ユーザー定義プロジェクトのモニタリングを有効にしている。
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **openshift-user-workload-monitoring** プロジェクトの Alertmanager で設定するシークレットを作成しました。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **コアプラットフォームのモニタリングのために Alertmanager にシークレット設定を追加するには、次の手順を実行します。**
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config** ConfigMap を編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

b. **data/config.yaml/alertmanagerMain** の下に **Secrets:** セクションを次の設定で追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      secrets: (1)
      - <secret_name_1> (2)
      - <secret_name_2>
```

(1) このセクションには、Alertmanager にマウントされるシークレットが含まれています。シークレットは、Alertmanager オブジェクトと同じ namespace 内に配置する必要があります。

(2) 受信者の認証認証情報を含む **Secret** オブジェクトの名前。複数のシークレットを追加する場合は、それぞれを新しい行に配置します。

次の config map 設定の例では、**test-secret-basic-auth** および **test-secret-api-token** という名前の 2 つの **Secret** オブジェクトを使用するように Alertmanager を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      secrets:
      - test-secret-basic-auth
      - test-secret-api-token
```

- **ユーザー定義のプロジェクトモニタリングのために Alertmanager にシークレット設定を追加するには、次の手順を実行します。**

a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config** config map を編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-
monitoring-config
```

b. **data/config.yaml/alertmanager/secrets** の下に **Secrets:** セクションを次の設定で追加します。

```
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    alertmanager:
      secrets: 1
      - <secret_name_1> 2
      - <secret_name_2>
```

1 このセクションには、Alertmanager にマウントされるシークレットが含まれています。シークレットは、Alertmanager オブジェクトと同じ namespace 内に配置する必要があります。

2 受信者の認証認証情報を含む **Secret** オブジェクトの名前。複数のシークレットを追加する場合は、それぞれを新しい行に配置します。

次の config map 設定の例では、**test-secret** および **test-secret-api-token** という名前の 2 つの **Secret** オブジェクトを使用するように Alertmanager を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    alertmanager:
      enabled: true
      secrets:
      - test-secret
      - test-api-receiver-token
```

注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

2. ファイルを保存して、変更を **ConfigMap** オブジェクトに適用します。新しい設定は自動的に適用されます。

4.2. 追加ラベルの時系列 (TIME SERIES) およびアラートへの割り当て

Prometheus の外部ラベル機能を使用して、Prometheus から送信されるすべての時系列とアラートにカスタムラベルを付けることができます。

前提条件

- **OpenShift Container Platform のコアモニタリングコンポーネントを設定する場合**、以下を実行します。

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **cluster-monitoring-configConfigMap** オブジェクトを作成している。

- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **カスタムラベルを、OpenShift Container Platform のコアプロジェクトをモニターする Prometheus インスタンスから出るすべての時系列およびアラートに割り当てるには**、以下を実行します。
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

     b. **data/config.yaml** の下にすべてのメトリックについて追加する必要のあるラベルのマップを定義します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      externalLabels:
        <key>: <value> 1
```

     1 **<key>: <value>** をキーと値のペアのマップに置き換えます。ここで、**<key>** は新規ラベルの一意の名前で、**<value>** はその値になります。

警告

- **prometheus** または **prometheus_replica** は予約され、上書きされるため、これらをキー名として使用しないでください。
- キー名に **cluster** または **managed_cluster** を使用しないでください。これらを使用すると、開発者ダッシュボードでデータが表示されなくなる問題が発生する可能性があります。

たとえば、リージョンと環境に関するメタデータをすべての時系列とアラートに追加するには、次の例を使用します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      externalLabels:
        region: eu
        environment: prod
```

- **カスタムラベルを、ユーザー定義のプロジェクトをモニターする Prometheus インスタンスから出るすべての時系列およびアラートに割り当てるには**、以下を実行します。

  a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

  ```
  $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
  ```

  b. **data/config.yaml** の下にすべてのメトリックについて追加する必要のあるラベルのマップを定義します。

  ```
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: user-workload-monitoring-config
    namespace: openshift-user-workload-monitoring
  data:
    config.yaml: |
      prometheus:
        externalLabels:
          <key>: <value> 1
  ```

  1 **<key>: <value>** をキーと値のペアのマップに置き換えます。ここで、**<key>** は新規ラベルの一意の名前で、**<value>** はその値になります。

警告

- **prometheus** または **prometheus_replica** は予約され、上書きされるため、これらをキー名として使用しないでください。
- キー名に **cluster** または **managed_cluster** を使用しないでください。これらを使用すると、開発者ダッシュボードでデータが表示されなくなる問題が発生する可能性があります。

注記

openshift-user-workload-monitoring プロジェクトでは、Prometheus はメトリックを処理し、Thanos Ruler はアラートおよび記録ルールを処理します。**user-workload-monitoring-config ConfigMap** オブジェクトで **prometheus** の **externalLabels** を設定すると、すべてのルールではなく、メトリックの外部ラベルのみが設定されます。

たとえば、リージョンと環境に関するメタデータを、ユーザー定義プロジェクトに関連するすべての時系列とアラートに追加するには、次の例を使用します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      externalLabels:
        region: eu
        environment: prod
```

2. 変更を適用するためにファイルを保存します。新しい設定は自動的に適用されます。

注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

警告

変更がモニタリング config map に保存されると、関連するプロジェクトの Pod およびその他のリソースが再デプロイされる可能性があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

関連情報

- モニタリング config map を作成する手順は、モニタリングスタックの設定の準備 を参照してください。
- ユーザー定義プロジェクトのモニタリングの有効化

第5章 モニタリングのための POD トポロジー分散制約の設定

Pod トポロジー分散制約を使用して、OpenShift Container Platform Pod が複数のアベイラビリティーゾーンにデプロイされている場合に、Prometheus、Thanos Ruler、および Alertmanager Pod がネットワークトポロジー全体にどのように分散されるかを制御できます。

Pod トポロジーの分散制約は、ノードがリージョンやリージョン内のゾーンなど、さまざまなインフラストラクチャーレベルに分散している階層トポロジー内で Pod のスケジューリングを制御するのに適しています。さらに、さまざまなゾーンで Pod をスケジュールできるため、特定のシナリオでネットワーク遅延を改善できます。

関連情報

- Pod トポロジー分散制約を使用した Pod 配置の制御
- Kubernetes Pod Topology Spread Constraints documentation

5.1. PROMETHEUS の POD トポロジー分散制約の設定

コア OpenShift Container Platform プラットフォームのモニタリングでは、Prometheus の Pod トポロジー分散制約を設定して、Pod レプリカがゾーン間でノードにスケジュールされる方法を微調整できます。そうすることで、Prometheus Pod の可用性が高くなり、より効率的に実行されるようになります。これは、ワークロードが異なるデータセンターまたは階層インフラストラクチャーゾーンのノードに分散されるためです。

cluster-monitoring-config config map で、Prometheus の Pod トポロジー分散制約を設定します。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-monitoring** namespace で **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **data/config.yaml/prometheusK8s** の下に次の設定の値を追加して、Pod トポロジー分散制約を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      topologySpreadConstraints:
```

```
- maxSkew: 1 (1)
  topologyKey: monitoring (2)
  whenUnsatisfiable: DoNotSchedule (3)
  labelSelector:
    matchLabels: (4)
      app.kubernetes.io/name: prometheus
```

(1) **maxSkew** の数値を指定します。これは、どの程度まで Pod が不均等に分散されることを許可するか定義します。このフィールドは必須で、値はゼロより大きい必要があります。指定された値は、**whenUnsatisfiable** に指定した値に応じて異なる効果を持ちます。

(2) **topologyKey** にノードラベルのキーを指定します。このフィールドは必須です。このキーと同じ値のラベルを持つノードは、同じトポロジーにあると見なされます。スケジューラーは、バランスのとれた数の Pod を各ドメインに配置しようとします。

(3) **whenUnsatisfiable** の値を指定します。このフィールドは必須です。利用可能なオプションは **DoNotSchedule** と **ScheduleAnyway** です。**maxSkew** 値で、ターゲットトポロジー内の一致する Pod の数とグローバル最小値との間で許容される最大差を定義する場合は、**DoNotSchedule** を指定します。スケジューラーが引き続き Pod をスケジュールするが、スキューを減らす可能性のあるノードにより高い優先度を与える場合は、**ScheduleAnyway** を指定します。

(4) **matchLabels** の値を指定します。この値は、制約の適用対象となる一致する Pod のセットを識別するために使用されます。

3. ファイルを保存して、変更を自動的に適用します。

> **警告**
>
> **cluster-monitoring-config** config map への変更を保存すると、**openshift-monitoring** プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。そのプロジェクトで実行中のモニタリングプロセスも再起動する場合があります。

5.2. ALERTMANAGER の POD トポロジー分散制約の設定

コア OpenShift Container Platform プラットフォームのモニタリングでは、Alertmanager の Pod トポロジー分散制約を設定して、Pod レプリカがゾーン間でノードにスケジュールされる方法を微調整できます。そうすることで、Alertmanager Pod の可用性が高くなり、より効率的に実行されるようになります。これは、ワークロードが異なるデータセンターまたは階層インフラストラクチャーゾーンのノードに分散されるためです。

cluster-monitoring-config config map で、Alertmanager の Pod トポロジー分散制約を設定します。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **cluster-monitoring-configConfigMap** オブジェクトを作成している。

- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-monitoring** namespace で **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **data/config.yaml/alertmanagermain** の下に次の設定の値を追加して、Pod トポロジー分散制約を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      topologySpreadConstraints:
      - maxSkew: 1 (1)
        topologyKey: monitoring (2)
        whenUnsatisfiable: DoNotSchedule (3)
        labelSelector:
          matchLabels: (4)
            app.kubernetes.io/name: alertmanager
```

(1) **maxSkew** の数値を指定します。これは、どの程度まで Pod が不均等に分散されることを許可するか定義します。このフィールドは必須で、値はゼロより大きい必要があります。指定された値は、**whenUnsatisfiable** に指定した値に応じて異なる効果を持ちます。

(2) **topologyKey** にノードラベルのキーを指定します。このフィールドは必須です。このキーと同じ値のラベルを持つノードは、同じトポロジーにあると見なされます。スケジューラーは、バランスのとれた数の Pod を各ドメインに配置しようとします。

(3) **whenUnsatisfiable** の値を指定します。このフィールドは必須です。利用可能なオプションは **DoNotSchedule** と **ScheduleAnyway** です。**maxSkew** 値で、ターゲットトポロジー内の一致する Pod の数とグローバル最小値との間で許容される最大差を定義する場合は、**DoNotSchedule** を指定します。スケジューラーが引き続き Pod をスケジュールするが、スキューを減らす可能性のあるノードにより高い優先度を与える場合は、**ScheduleAnyway** を指定します。

(4) **matchLabels** の値を指定します。この値は、制約の適用対象となる一致する Pod のセットを識別するために使用されます。

3. ファイルを保存して、変更を自動的に適用します。

警告

cluster-monitoring-config config map への変更を保存すると、**openshift-monitoring** プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。そのプロジェクトで実行中のモニタリングプロセスも再起動する場合があります。

5.3. THANOS RULER の POD トポロジー分散制約の設定

ユーザー定義のモニタリングの場合、Thanos Ruler の Pod トポロジー分散制約を設定して、Pod レプリカがゾーン間でノードにスケジュールされる方法を微調整できます。そうすることで、Thanos Ruler Pod の可用性が高くなり、より効率的に実行されるようになります。これは、ワークロードが異なるデータセンターまたは階層インフラストラクチャーゾーンのノードに分散されるためです。

user-workload-monitoring-config config map で、Thanos Ruler の Pod トポロジー分散制約を設定します。

前提条件

- クラスター管理者は、ユーザー定義プロジェクトのモニタリングを有効にしている。
- **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
- **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-user-workload-monitoring** namespaceで **user-workload-monitoring-config** config map を編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. **data/config.yaml/thanosRuler** の下に次の設定の値を追加して、Pod トポロジー分散制約を設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      topologySpreadConstraints:
      - maxSkew: 1 (1)
        topologyKey: monitoring (2)
```

```
    whenUnsatisfiable: ScheduleAnyway (3)
    labelSelector:
      matchLabels: (4)
        app.kubernetes.io/name: thanos-ruler
```

(1) **maxSkew** の数値を指定します。これは、どの程度まで Pod が不均等に分散されることを許可するか定義します。このフィールドは必須で、値はゼロより大きい必要があります。指定された値は、**whenUnsatisfiable** に指定した値に応じて異なる効果を持ちます。

(2) **topologyKey** にノードラベルのキーを指定します。このフィールドは必須です。このキーと同じ値のラベルを持つノードは、同じトポロジーにあると見なされます。スケジューラーは、バランスのとれた数の Pod を各ドメインに配置しようとします。

(3) **whenUnsatisfiable** の値を指定します。このフィールドは必須です。利用可能なオプションは **DoNotSchedule** と **ScheduleAnyway** です。**maxSkew** 値で、ターゲットトポロジー内の一致する Pod の数とグローバル最小値との間で許容される最大差を定義する場合は、**DoNotSchedule** を指定します。スケジューラーが引き続き Pod をスケジュールするが、スキューを減らす可能性のあるノードにより高い優先度を与える場合は、**ScheduleAnyway** を指定します。

(4) **matchLabels** の値を指定します。この値は、制約の適用対象となる一致する Pod のセットを識別するために使用されます。

3. ファイルを保存して、変更を自動的に適用します。

> **警告**
>
> 変更が **user-workload-monitoring-config** config map に保存されると、**openshift-user-workload-monitoring** プロジェクトの Pod および他のリソースは再デプロイされる可能性があります。そのプロジェクトで実行中のモニタリングプロセスも再起動する場合があります。

5.4. モニタリングコンポーネントのログレベルの設定

Alertmanager、Prometheus Operator、Prometheus、Alertmanager および Thanos Querier および Thanos Ruler のログレベルを設定できます。

cluster-monitoring-config および **user-workload-monitoring-configConfigMap** オブジェクトの該当するコンポーネントには、以下のログレベルを適用することができます。

- **debug**:デバッグ、情報、警告、およびエラーメッセージをログに記録します。
- **info**:情報、警告およびエラーメッセージをログに記録します。
- **warn**:警告およびエラーメッセージのみをログに記録します。
- **error**:エラーメッセージのみをログに記録します。

デフォルトのログレベルは **info** です。

前提条件

- **openshift-monitoring プロジェクトで** Alertmanager、Prometheus Operator、Prometheus、**または** Thanos Querier **のログレベルを設定する場合には、以下を実行します。**
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **openshift-user-workload-monitoring プロジェクトで** Prometheus Operator、Prometheus、**または** Thanos Ruler **のログレベルを設定する場合には、**以下を実行します。
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **ConfigMap** オブジェクトを編集します。
   - **openshift-monitoring プロジェクトのコンポーネントのログレベルを設定するには、**以下を実行します。
     a. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

     ```
     $ oc -n openshift-monitoring edit configmap cluster-monitoring-config
     ```

     b. コンポーネントの **logLevel: <log_level>** を **data/config.yaml** の下に追加します。

     ```
     apiVersion: v1
     kind: ConfigMap
     metadata:
       name: cluster-monitoring-config
       namespace: openshift-monitoring
     data:
       config.yaml: |
         <component>: 1
           logLevel: <log_level> 2
     ```

     1 ログレベルを設定するモニタリングスタックコンポーネント。デフォルトのプラットフォームモニタリングの場合、使用可能なコンポーネント値は **prometheusK8s**、**alertmanagerMain**、**prometheusOperator**、および **thanosQuerier** です。

     2 コンポーネントに設定するログレベル。使用可能な値は、**error**、**warn**、**info**、および **debug** です。デフォルト値は **info** です。

   - **openshift-user-workload-monitoring プロジェクトのコンポーネントのログレベルを設定するには、**以下を実行します。

     a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-**

a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

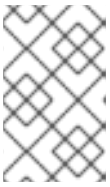
b. コンポーネントの **logLevel: <log_level>** を **data/config.yaml** の下に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>: 1
      logLevel: <log_level> 2
```

1 ログレベルを設定するモニタリングスタックコンポーネント。ユーザーワークロードのモニタリングの場合、使用可能なコンポーネントの値は、**alertmanager**、**prometheus**、**prometheusOperator**、および **thanosRuler** です。

2 コンポーネントに適用するログレベル。使用可能な値は、**error**、**warn**、**info**、および **debug** です。デフォルト値は **info** です。

2. 変更を適用するためにファイルを保存します。ログレベルの変更を適用する際に、コンポーネントの Pod は自動的に再起動します。



注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

警告

変更がモニタリング config map に保存されると、関連するプロジェクトの Pod およびその他のリソースが再デプロイされる可能性があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

3. 関連するプロジェクトでデプロイメントまたは Pod 設定を確認し、ログレベルが適用されていることを確認します。以下の例では、**openshift-user-workload-monitoring** プロジェクトの **prometheus-operator** デプロイメントでログレベルを確認します。

```
$ oc -n openshift-user-workload-monitoring get deploy prometheus-operator -o yaml | grep "log-level"
```

出力例

出力例

```
      - --log-level=debug
```

4. コンポーネントの Pod が実行中であることを確認します。以下の例は、**openshift-user-workload-monitoring** プロジェクトの Pod のステータスをリスト表示します。

```
$ oc -n openshift-user-workload-monitoring get pods
```

注記

認識されない **logLevel** 値が **ConfigMap** オブジェクトに含まれる場合は、コンポーネントの Pod が正常に再起動しない可能性があります。

5.5. PROMETHEUS のクエリーログファイルの有効化

エンジンによって実行されたすべてのクエリーをログファイルに書き込むように Prometheus を設定できます。これは、デフォルトのプラットフォームモニタリングおよびユーザー定義のワークロードモニタリングに対して行うことができます。

重要

ログローテーションはサポートされていないため、問題のトラブルシューティングが必要な場合にのみ、この機能を一時的に有効にします。トラブルシューティングが終了したら、**ConfigMap**オブジェクトに加えた変更を元に戻してクエリーログを無効にし、機能を有効にします。

前提条件

- **openshift-monitoring プロジェクトで Prometheus のクエリーログファイル機能を有効にしている場合**:
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
  - **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- **openshift-user-workload-monitoring プロジェクトで Prometheus のクエリーログファイル機能を有効にしている場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **user-workload-monitoring-config-edit** ロールを持つユーザーとして、クラスターにアクセスできる。
  - **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

- **openshift-monitoring プロジェクトで Prometheus のクエリーログファイルを設定するには**:
  1. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **data/config.yaml**の下の**prometheusK8s**の**queryLogFile: <path>** を追加します:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      queryLogFile: <path> 1
```

1 クエリーがログに記録されるファイルへのフルパス。

3. 変更を適用するためにファイルを保存します。

警告

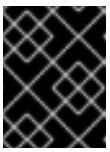
monitoring config map への変更を保存すると、関連プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

4. コンポーネントの Pod が実行中であることを確認します。次のコマンドの例は、**openshift-monitoring**プロジェクトの Pod のステータスを一覧表示します。

```
$ oc -n openshift-monitoring get pods
```

5. クエリーログを読みます。

```
$ oc -n openshift-monitoring exec prometheus-k8s-0 -- cat <path>
```



重要

ログに記録されたクエリー情報を確認した後、config map の設定を元に戻します。

- **openshift-user-workload-monitoringプロジェクトで Prometheus のクエリーログファイルを設定するには**:

1. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

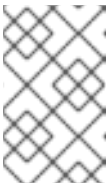
```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. **data/config.yaml** の下の **prometheus** の **queryLogFile: <path>** を追加します:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      queryLogFile: <path> 1
```

1 クエリーがログに記録されるファイルへのフルパス。

3. 変更を適用するためにファイルを保存します。



注記

user-workload-monitoring-config ConfigMap オブジェクトに適用される設定は、クラスター管理者がユーザー定義プロジェクトのモニタリングを有効にしない限りアクティブにされません。

警告

monitoring config map への変更を保存すると、関連プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

4. コンポーネントの Pod が実行中であることを確認します。以下の例は、**openshift-user-workload-monitoring** プロジェクトの Pod のステータスを一覧表示します。

```
$ oc -n openshift-user-workload-monitoring get pods
```

5. クエリーログを読みます。

```
$ oc -n openshift-user-workload-monitoring exec prometheus-user-workload-0 -- cat
<path>
```

重要

ログに記録されたクエリー情報を確認した後、config map の設定を元に戻します。

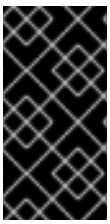
関連情報

- モニタリング config map を作成する手順は、モニタリングスタックの設定の準備 を参照してください。

- ユーザー定義のモニタリングを有効にする手順については、Enabling monitoring for user-defined projects を参照してください。

5.6. THANOS QUERIER のクエリーロギングの有効化

openshift-monitoring プロジェクトのデフォルトのプラットフォームモニタリングの場合、Cluster Monitoring Operator を有効にして Thanos Querier によって実行されるすべてのクエリーをログに記録できます。



重要

ログローテーションはサポートされていないため、問題のトラブルシューティングが必要な場合にのみ、この機能を一時的に有効にします。トラブルシューティングが終了したら、**ConfigMap**オブジェクトに加えた変更を元に戻してクエリーログを無効にし、機能を有効にします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **cluster-monitoring-configConfigMap** オブジェクトを作成している。

手順

openshift-monitoring プロジェクトで Thanos Querier のクエリーロギングを有効にすることができます。

1. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. 以下の例のように **thanosQuerier** セクションを **data/config.yaml** に追加し、値を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    thanosQuerier:
      enableRequestLogging: <value> 1
      logLevel: <value> 2
```

1. ロギングを有効にするには、値を **true** に設定し、ロギングを無効にするには **false** を設定します。デフォルト値は **false** です。

2. この値は **debug**、**info**、**warn**、または **error** に設定します。**logLevel** に値が存在しない場合、ログレベルはデフォルトで **error** に設定されます。

3. 変更を適用するためにファイルを保存します。

警告

monitoring config map への変更を保存すると、関連プロジェクトの Pod およびその他のリソースが再デプロイされる場合があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

検証

1. Thanos Querier Pod が実行されていることを確認します。次のコマンドの例は、**openshift-monitoring**プロジェクトの Pod のステータスを一覧表示します。

```
$ oc -n openshift-monitoring get pods
```

2. 以下のサンプルコマンドをモデルとして使用して、テストクエリーを実行します。

```
$ token=`oc create token prometheus-k8s -n openshift-monitoring`
$ oc -n openshift-monitoring exec -c prometheus prometheus-k8s-0 -- curl -k -H
"Authorization: Bearer $token" 'https://thanos-querier.openshift-
monitoring.svc:9091/api/v1/query?query=cluster_version'
```

3. 以下のコマンドを実行してクエリーログを読み取ります。

```
$ oc -n openshift-monitoring logs <thanos_querier_pod_name> -c thanos-query
```

注記

thanos-querier Pod は高可用性 (HA) Pod であるため、1 つの Pod でのみログを表示できる可能性があります。

4. ログに記録されたクエリー情報を確認したら、config map で **enableRequestLogging** の値を **false** に変更してクエリーロギングを無効にします。

関連情報

- モニタリング config map を作成する手順は、モニタリングスタックの設定の準備 を参照してください。

第6章 PROMETHEUS アダプターの監査ログレベルの設定

デフォルトのプラットフォームモニタリングでは、Prometheus アダプターの監査ログレベルを設定できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **cluster-monitoring-configConfigMap** オブジェクトを作成している。

手順

デフォルトの **openshift-monitoring** プロジェクトで Prometheus アダプターの監査ログレベルを設定できます。

1. **openshift-monitoring** プロジェクトで **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **k8sPrometheusAdapter/audit** セクションに **profile:** を **data/config.yaml** の下に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    k8sPrometheusAdapter:
      audit:
        profile: <audit_log_level> 1
```

1 Prometheus アダプターに適用する監査ログレベル。

3. **profile:** パラメーターに以下のいずれかの値を使用して、監査ログレベルを設定します。
   - **None**: イベントをログに記録しません。
   - **Metadata**: ユーザー、タイムスタンプなど、リクエストのメタデータのみをログに記録します。リクエストテキストと応答テキストはログに記録しないでください。**metadata** はデフォルトの監査ログレベルです。
   - **Request**: メタデータと要求テキストのみをログに記録しますが、応答テキストはログに記録しません。このオプションは、リソース以外の要求には適用されません。
   - **RequestResponse**: イベントのメタデータ、要求テキスト、および応答テキストをログに記録します。このオプションは、リソース以外の要求には適用されません。
4. 変更を適用するためにファイルを保存します。変更を適用すると、Prometheus Adapter 用の Pod が自動的に再起動します。

警告

変更がモニタリング config map に保存されると、関連するプロジェクトの Pod およびその他のリソースが再デプロイされる可能性があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

検証

1. config map の **k8sPrometheusAdapter/audit/profile** で、ログレベルを **Request** に設定し、ファイルを保存します。
2. Prometheus アダプターの Pod が実行されていることを確認します。以下の例は、**openshift-monitoring** プロジェクトの Pod のステータスを一覧表示します。

```
$ oc -n openshift-monitoring get pods
```

3. 監査ログレベルと監査ログファイルのパスが正しく設定されていることを確認します。

```
$ oc -n openshift-monitoring get deploy prometheus-adapter -o yaml
```

出力例

```
...
  - --audit-policy-file=/etc/audit/request-profile.yaml
  - --audit-log-path=/var/log/adapter/audit.log
```

4. 正しいログレベルが **openshift-monitoring** プロジェクトの **prometheus-adapter** デプロイメントに適用されていることを確認します。

```
$ oc -n openshift-monitoring exec deploy/prometheus-adapter -c prometheus-adapter -- cat
/etc/audit/request-profile.yaml
```

出力例

```
"apiVersion": "audit.k8s.io/v1"
"kind": "Policy"
"metadata":
  "name": "Request"
"omitStages":
- "RequestReceived"
"rules":
- "level": "Request"
```

注記

ConfigMap オブジェクトで Prometheus アダプターに認識されない **profile** 値を入力すると、Prometheus アダプターには変更が加えられず、Cluster Monitoring Operator によってエラーがログに記録されます。

5. Prometheus アダプターの監査ログを確認します。

```
$ oc -n openshift-monitoring exec -c <prometheus_adapter_pod_name> -- cat
/var/log/adapter/audit.log
```

関連情報

- モニタリング config map を作成する手順は、モニタリングスタックの設定の準備 を参照してください。

6.1. ローカル ALERTMANAGER の無効化

Prometheus インスタンスからのアラートをルーティングするローカル Alertmanager は、OpenShift ContainerPlatform モニタリングスタックの **openshift-monitoring** プロジェクトではデフォルトで有効になっています。

ローカル Alertmanager を必要としない場合、**openshift-monitoring** プロジェクトで **cluster-monitoring-config** config map を指定して無効にできます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **cluster-monitoring-config**ConfigMap を作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-monitoring** プロジェクトで **cluster-monitoring-config** ConfigMap を編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **data/config.yaml** の下に、**alertmanagerMain** コンポーネントの **enabled: false** を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      enabled: false
```

3. 変更を適用するためにファイルを保存します。Alertmanager インスタンスは、この変更を適用すると自動的に無効にされます。

関連情報

- Prometheus Alertmanager ドキュメント
- xref:[アラートの管理]

6.2. 次のステップ

- ユーザー定義プロジェクトのモニタリングの有効化
- リモート正常性レポート を確認し、必要な場合はこれをオプトアウトします。

第7章 ユーザー定義プロジェクトのモニタリングの有効化

OpenShift Container Platform 4.15 では、デフォルトのプラットフォームのモニタリングに加えて、ユーザー定義プロジェクトのモニタリングを有効にできます。追加のモニタリングソリューションを必要とせずに、Open Shift Container Platform で独自のプロジェクトをモニタリングできます。この機能を使用することで、コアプラットフォームコンポーネントおよびユーザー定義プロジェクトのモニタリングが一元化されます。

注記

Operator Lifecycle Manager (OLM) を使用してインストールされた Prometheus Operator のバージョンは、ユーザー定義のモニタリングと互換性がありません。そのため、OLM Prometheus Operator によって管理される Prometheus カスタムリソース (CR) としてインストールされるカスタム Prometheus インスタンスは OpenShift Container Platform ではサポートされていません。

7.1. ユーザー定義プロジェクトのモニタリングの有効化

クラスター管理者は、クラスターモニタリング **ConfigMap** オブジェクト に **enableUserWorkload: true** フィールドを設定し、ユーザー定義プロジェクトのモニタリングを有効にできます。

重要

OpenShift Container Platform 4.15 では、ユーザー定義プロジェクトのモニタリングを有効にする前に、カスタム Prometheus インスタンスを削除する必要があります。

注記

OpenShift Container Platform のユーザー定義プロジェクトのモニタリングを有効にするには、**cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできる必要があります。これにより、クラスター管理者は任意で、ユーザー定義のプロジェクトをモニターするコンポーネントを設定する権限をユーザーに付与できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-monitoring-configConfigMap** オブジェクトを作成している。
- オプションで **user-workload-monitoring-config ConfigMap** を **openshift-user-workload-monitoring** プロジェクトに作成している。ユーザー定義プロジェクトをモニターするコンポーネントの **ConfigMap** に設定オプションを追加できます。

注記

設定の変更を **user-workload-monitoring-config ConfigMap** に保存するたびに、**openshift-user-workload-monitoring** プロジェクトの Pod が再デプロイされます。これらのコンポーネントが再デプロイするまで時間がかかる場合があります。ユーザー定義プロジェクトのモニタリングを最初に有効にする前に **ConfigMap** オブジェクトを作成し、設定することができます。これにより、Pod を頻繁に再デプロイする必要がなくなります。

手順

1. **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **enableUserWorkload: true** を **data/config.yaml** の下に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true 1
```

1 **true** に設定すると、**enableUserWorkload** パラメーターはクラスター内のユーザー定義プロジェクトのモニタリングを有効にします。

3. 変更を適用するためにファイルを保存します。ユーザー定義プロジェクトのモニタリングは自動的に有効になります。

警告

変更が **cluster-monitoring-config ConfigMap** オブジェクトに保存されると、**openshift-monitoring** プロジェクトの Pod および他のリソースは再デプロイされる可能性があります。該当するプロジェクトの実行中のモニタリングプロセスも再起動する可能性があります。

4. **prometheus-operator**、 **prometheus-user-workload** および **thanos-ruler-user-workload** Pod が **openshift-user-workload-monitoring** プロジェクトで実行中であることを確認します。Pod が起動するまでに少し時間がかかる場合があります。

```
$ oc -n openshift-user-workload-monitoring get pod
```

出力例

```
NAME                                   READY   STATUS    RESTARTS   AGE
prometheus-operator-6f7b748d5b-t7nbg   2/2     Running   0          3h
prometheus-user-workload-0             4/4     Running   1          3h
prometheus-user-workload-1             4/4     Running   1          3h
thanos-ruler-user-workload-0           3/3     Running   0          3h
thanos-ruler-user-workload-1           3/3     Running   0          3h
```

関連情報

- ユーザー定義のワークロードモニタリング config map の作成

- モニタリングスタックの設定
- ユーザーに対するユーザー定義プロジェクトのモニタリングを設定するための権限の付与

7.2. ユーザーに対するユーザー定義のプロジェクトをモニターする権限の付与

クラスター管理者は、すべての OpenShift Container Platform のコアプロジェクトおよびユーザー定義プロジェクトをモニターできます。

クラスター管理者は、開発者およびその他のユーザーに、独自のプロジェクトをモニターする権限を付与できます。特権は、以下のモニタリングロールのいずれかを割り当てることで付与されます。

- **monitoring-rules-view** クラスターロールは、プロジェクトの **PrometheusRule** カスタムリソースへの読み取りアクセスを提供します。
- **monitoring-rules-edit** クラスターロールは、プロジェクトの **PrometheusRule** カスタムリソースを作成、変更、削除する権限をユーザーに付与します。また、ユーザーにアラートをサイレントにする機能も付与します。
- **monitoring-edit** クラスターロールは、**monitoring-rules-edit** クラスターロールと同じ特権を付与します。さらに、ユーザーはサービスまたは Pod の新規の収集 ターゲットを作成できます。このロールを使用すると、**ServiceMonitor** および **PodMonitor** リソースを作成し、変更し、削除することもできます。

また、ユーザー定義のプロジェクトをモニターするコンポーネントを設定する権限をユーザーに付与することもできます。

- **openshift-user-workload-monitoring** プロジェクトの user-workload-monitoring-config-edit ロールにより、**user-workload-monitoring-config ConfigMap** オブジェクトを編集できます。このロールを使用して、**ConfigMap** オブジェクトを編集し、ユーザー定義のワークロードモニタリング用に Prometheus、Prometheus Operator、および Thanos Ruler を設定できます。

また、ユーザー定義プロジェクトのアラートルーティングを設定する権限をユーザーに付与することもできます。

- **alert-routing-edit** クラスターロールは、プロジェクトの **AlertmanagerConfig** カスタムリソースを作成、更新、および削除する権限をユーザーに付与します。

このセクションでは、OpenShift Container Platform Web コンソールまたは CLI を使用してこれらのロールを割り当てる方法について説明します。

7.2.1. Web コンソールを使用したユーザー権限の付与

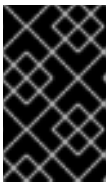
OpenShift Container Platform Web コンソールを使用して、独自のプロジェクトをモニターする権限をユーザーに付与できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- ロールを割り当てるユーザーアカウントがすでに存在している。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**User Management → RoleBindings → Create binding** に移動します。
2. **Binding Type**で、Namespace Role Binding タイプを選択します。
3. **Name** フィールドに、ロールバインディングの名前を入力します。
4. **Namespace** フィールドで、アクセスを付与するユーザー定義プロジェクトを選択します。



重要

モニタリングロールは、**Namespace** フィールドで適用するプロジェクトにバインドされます。この手順を使用してユーザーに付与する権限は、選択されたプロジェクトにのみ適用されます。

5. **Role Name** リストで、**monitoring-rules-view**、 **monitoring-rules-edit**、または **monitoring-edit** を選択します。
6. **Subject** セクションで、**User** を選択します。
7. **Subject Name** フィールドにユーザーの名前を入力します。
8. **Create** を選択して、ロールバインディングを適用します。

7.2.2. CLI を使用したユーザー権限の付与

OpenShift CLI (**oc**) を使用して、独自のプロジェクトをモニターする権限をユーザーに付与できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- ロールを割り当てるユーザーアカウントがすでに存在している。
- OpenShift CLI (**oc**) がインストールされている。

手順

- プロジェクトのユーザーにモニタリングロールを割り当てます。

```
$ oc policy add-role-to-user <role> <user> -n <namespace>
```
(1)

(1) **<role>** を **monitoring-rules-view**、**monitoring-rules-edit**、または **monitoring-edit** に置き換えます。

重要

選択したすべてのロールは、クラスター管理者が特定のプロジェクトにバインドする必要があります。

たとえば、**<role>** を **monitoring-edit** に、**<user>** を **johnsmith** に、**<namespace>** を **ns1** に置き換えます。これにより、ユーザー **johnsmith** に、メトリックコレクションをセットアップし、**ns1** namespace にアラートルールを作成する権限が割り当てられます。

7.3. ユーザーに対するユーザー定義プロジェクトのモニタリングを設定するための権限の付与

クラスター管理者は、**user-workload-monitoring-config-edit** ロールをユーザーに割り当てることができます。これにより、OpenShift Container Platform のコアモニタリングコンポーネントの設定および管理権限を付与せずに、ユーザー定義プロジェクトのモニタリングを設定および管理する権限が付与されます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- ロールを割り当てるユーザーアカウントがすでに存在している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **user-workload-monitoring-config-edit** ロールを **openshift-user-workload-monitoring** プロジェクトのユーザーに割り当てます。

```
$ oc -n openshift-user-workload-monitoring adm policy add-role-to-user \
  user-workload-monitoring-config-edit <user> \
  --role-namespace openshift-user-workload-monitoring
```

2. 関連するロールバインディングを表示して、ユーザーが **user-workload-monitoring-config-edit** ロールに正しく割り当てられていることを確認します。

```
$ oc describe rolebinding <role_binding_name> -n openshift-user-workload-monitoring
```

コマンドの例

```
$ oc describe rolebinding user-workload-monitoring-config-edit -n openshift-user-workload-monitoring
```

出力例

```
Name:         user-workload-monitoring-config-edit
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  user-workload-monitoring-config-edit
Subjects:
  Kind  Name  Namespace
  ----  ----  ---------
  User  user1          1
```

1 この例では、**user1** が **user-workload-monitoring-config-edit** ロールに割り当てられています。

7.4. カスタムアプリケーションについてのクラスター外からのメトリックへのアクセス

ユーザー定義プロジェクトを使用して独自のサービスを監視する場合は、クラスターの外部から Prometheus メトリクスをクエリーできます。このデータには、**thanos-querier** ルートを使用してクラスターの外部からアクセスします。

このアクセスは、認証に Bearer Token を使用することのみをサポートします。

前提条件

- "ユーザー定義プロジェクトのモニタリングの有効化" の手順に従い、独自のサービスをデプロイしている。
- Thanos Querier API へのアクセス権限を持つ **cluster-monitoring-view** クラスターロールでアカウントにログインしている。
- Thanos Querier API ルートの取得権限を持つアカウントにログインしています。

注記

アカウントに Thanos Querier API ルートの取得権限がない場合、クラスター管理者はルートの URL を提供できます。

手順

1. 次のコマンドを実行して、Prometheus に接続するための認証トークンを展開します。

```
$ TOKEN=$(oc whoami -t)
```

2. 次のコマンドを実行して、**thanos-querier** API ルート URL を展開します。

```
$ HOST=$(oc -n openshift-monitoring get route thanos-querier -ojsonpath={.spec.host})
```

3. 次のコマンドを使用して、サービスが実行されている namespace に namespace を設定します。

```
$ NAMESPACE=ns1
```

4. 次のコマンドを実行して、コマンドラインで独自のサービスのメトリクスに対してクエリーを実行します。

```
$ curl -H "Authorization: Bearer $TOKEN" -k "https://$HOST/api/v1/query?" --data-urlencode
"query=up{namespace='$NAMESPACE'}"
```

出力には、Prometheus がスクレイピングしている各アプリケーション Pod のステータスが表示されます。

出力例

```
{"status":"success","data":{"resultType":"vector","result":[{"metric":
{"__name__":"up","endpoint":"web","instance":"10.129.0.46:8080","job":"prometheus-
example-app","namespace":"ns1","pod":"prometheus-example-app-68d47c4fb6-
```

```
jztp2","service":"prometheus-example-app"},"value":[1591881154.748,"1"]}]}}
```

関連情報

- ユーザー定義プロジェクトのモニタリングの有効化

7.5. モニタリングからのユーザー定義のプロジェクトを除く

ユーザー定義のプロジェクトは、ユーザーワークロードモニタリングから除外できます。これを実行するには、**openshift.io/user-monitoring** ラベルに **false** を指定して、プロジェクトの namespace に追加します。

手順

1. ラベルをプロジェクト namespace に追加します。

```
$ oc label namespace my-project 'openshift.io/user-monitoring=false'
```

2. モニタリングを再度有効にするには、namespace からラベルを削除します。

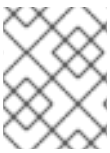
```
$ oc label namespace my-project 'openshift.io/user-monitoring-'
```

注記

プロジェクトにアクティブなモニタリングターゲットがあった場合、ラベルを追加した後、Prometheus がそれらのスクレイピングを停止するまでに数分かかる場合があります。

7.6. ユーザー定義プロジェクトのモニタリングの無効化

ユーザー定義プロジェクトのモニタリングを有効にした後に、クラスターモニタリング **ConfigMap** オブジェクトに **enableUserWorkload: false** を設定してこれを再度無効にできます。



注記

または、**enableUserWorkload: true** を削除して、ユーザー定義プロジェクトのモニタリングを無効にできます。

手順

1. **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

   a. **data/config.yaml** で **enableUserWorkload:** を **false** に設定します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
```

```
data:
  config.yaml: |
    enableUserWorkload: false
```

2. 変更を適用するためにファイルを保存します。ユーザー定義プロジェクトのモニタリングは自動的に無効になります。

3. **prometheus-operator**、 **prometheus-user-workload** および **thanos-ruler-user-workload** Pod が **openshift-user-workload-monitoring** プロジェクトで終了していることを確認します。これには少し時間がかかる場合があります。

```
$ oc -n openshift-user-workload-monitoring get pod
```

出力例

```
No resources found in openshift-user-workload-monitoring project.
```

注記

openshift-user-workload-monitoring プロジェクトの **user-workload-monitoring-config ConfigMap** オブジェクトは、ユーザー定義プロジェクトのモニタリングが無効にされている場合は自動的に削除されません。これにより、**ConfigMap** で作成した可能性のあるカスタム設定を保持されます。

7.7. 次のステップ

- メトリクスの管理

第8章 ユーザー定義プロジェクトのアラートルーティングの有効化

OpenShift Container Platform 4.15 では、クラスター管理者はユーザー定義プロジェクトのアラートルーティングを有効にできます。このプロセスは、以下の 2 つの一般的な手順で設定されています。

- ユーザー定義プロジェクトのアラートルーティングを有効にして、デフォルトのプラットフォーム Alertmanager インスタンスを使用するか、オプションでユーザー定義のプロジェクトに対してのみ別の Alertmanager インスタンスを使用できます。
- ユーザー定義プロジェクトのアラートルーティングを設定するための権限をユーザーに付与します。

これらの手順を完了すると、開発者およびその他のユーザーはユーザー定義のプロジェクトのカスタムアラートおよびアラートルーティングを設定できます。

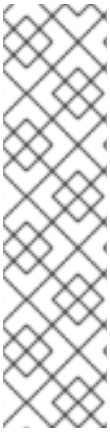
8.1. ユーザー定義プロジェクトのアラートルーティングについて

クラスター管理者は、ユーザー定義プロジェクトのアラートルーティングを有効にできます。この機能により、**alert-routing-edit** ロールを持つユーザーがユーザー定義プロジェクトのアラート通知ルーティングおよびレシーバーを設定できます。これらの通知は、デフォルトの Alertmanager インスタンスで指定されるか、有効にされている場合にユーザー定義のモニタリング専用のオプションの Alertmanager インスタンスによってルーティングされます。

次に、ユーザーはユーザー定義プロジェクトの **AlertmanagerConfig** オブジェクトを作成または編集して、ユーザー定義のアラートルーティングを作成し、設定できます。

ユーザーがユーザー定義のプロジェクトのアラートルーティングを定義した後に、ユーザー定義のアラート通知は以下のようにルーティングされます。

- デフォルトのプラットフォーム Alertmanager インスタンスを使用する場合、**openshift-monitoring** namespace の **alertmanager-main** Pod に対してこれを実行します。
- ユーザー定義プロジェクトの Alertmanager の別のインスタンスを有効にしている場合に、**openshift-user-workload-monitoring** namespace で **alertmanager-user-workload** Pod を行うには、以下を実行します。



注記

以下は、ユーザー定義プロジェクトのアラートルーティングの制限です。

- ユーザー定義のアラートルールの場合、ユーザー定義のルーティングはリソースが定義される namespace に対してスコープ指定されます。たとえば、namespace **ns1** のルーティング設定は、同じ namespace の **PrometheusRules** リソースにのみ適用されます。
- namespace がユーザー定義のモニタリングから除外される場合、namespace の **AlertmanagerConfig** リソースは、Alertmanager 設定の一部ではなくなります。

8.2. ユーザー定義のアラートルーティングのプラットフォーム ALERTMANAGER インスタンスの有効化

ユーザーは、Alertmanager のメインプラットフォームインスタンスを使用するユーザー定義のアラートルーティング設定を作成できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **cluster-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. **alertmanagerMain** セクションに **enableUserAlertmanagerConfig: true** を **data/config.yaml** の下に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      enableUserAlertmanagerConfig: true (1)
```

(1) **enableUserAlertmanagerConfig** 値を **true** に設定して、ユーザーが Alertmanager のメインプラットフォームインスタンスを使用するユーザー定義のアラートルーティング設定を作成できるようにします。

3. 変更を適用するためにファイルを保存します。

8.3. ユーザー定義のアラートルーティング用の個別の ALERTMANAGER インスタンスの有効化

クラスターによっては、ユーザー定義のプロジェクト用に専用の Alertmanager インスタンスをデプロイする必要がある場合があります。これは、デフォルトのプラットフォーム Alertmanager インスタンスの負荷を軽減するのに役立ちます。また、デフォルトのプラットフォームアラートとユーザー定義のアラートを分離することができます。このような場合、必要に応じて、Alertmanager の別のインスタンスを有効にして、ユーザー定義のプロジェクトのみにアラートを送信できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **openshift-monitoring** namespace の **cluster-monitoring-config** config map でユーザー定義プロジェクトのモニタリングを有効にしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. **data/config.yaml** の下にある **alertmanager** セクションに **enabled: true** および **enableAlertmanagerConfig: true** を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    alertmanager:
      enabled: true 1
      enableAlertmanagerConfig: true 2
```

1. **enabled** の値を **true** に設定して、クラスター内のユーザー定義プロジェクトの Alertmanager の専用インスタンスを有効にします。値を **false** に設定するか、キーを完全に省略してユーザー定義プロジェクトの Alertmanager を無効にします。この値を **false** に設定した場合や、キーを省略すると、ユーザー定義のアラートはデフォルトのプラットフォーム Alertmanager インスタンスにルーティングされます。

2. **enableAlertmanagerConfig** 値を **true** に設定して、ユーザーが **AlertmanagerConfig** オブジェクトで独自のアラートルーティング設定を定義できるようにします。

3. 変更を適用するためにファイルを保存します。ユーザー定義プロジェクトの Alertmanager の専用インスタンスが自動的に起動します。

検証

- **user-workload** Alertmanager インスタンスが起動していることを確認します。

```
# oc -n openshift-user-workload-monitoring get alertmanager
```

出力例

```
NAME            VERSION   REPLICAS   AGE
user-workload   0.24.0    2          100s
```

8.4. ユーザー定義プロジェクトのアラートルーティングを設定するためのユーザーへの権限の付与

ユーザー定義プロジェクトのアラートルーティングを設定する権限をユーザーに付与できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- **openshift-monitoring** namespace の **cluster-monitoring-config** config map でユーザー定義プロジェクトのモニタリングを有効にしている。
- ロールを割り当てるユーザーアカウントがすでに存在している。

- OpenShift CLI (**oc**) がインストールされている。

手順

- ユーザー定義プロジェクトのユーザーに **alert-routing-edit** クラスターロールを割り当てます。

```
$ oc -n <namespace> adm policy add-role-to-user alert-routing-edit <user> 1
```

1 **<namespace>**の場合は、ユーザー定義プロジェクトの代わりに namespace を使用します (例: **ns1**)。**<user>** の場合は、ロールを割り当てるアカウントの代わりにユーザー名を使用します。

関連情報

- Enabling monitoring for user defined projects
- ユーザー定義プロジェクトのアラートルーティングの作成

8.5. 次のステップ

- アラートの管理

第9章 メトリクスの管理

メトリックを使用すると、クラスターコンポーネントおよび独自のワークロードのパフォーマンスをモニターできます。

9.1. メトリクスについて

OpenShift Container Platform 4.15 では、クラスターコンポーネントはサービスエンドポイントで公開されるメトリクスを収集することによりモニターされます。ユーザー定義プロジェクトのメトリクスのコレクションを設定することもできます。メトリックを使用すると、クラスターコンポーネントおよび独自のワークロードの実行方法をモニターできます。

Prometheus クライアントライブラリーをアプリケーションレベルで使用することで、独自のワークロードに指定するメトリックを定義できます。

OpenShift Container Platform では、メトリックは **/metrics** の正規名の下に HTTP サービスエンドポイント経由で公開されます。**curl** クエリーを **http://<endpoint>/metrics** に対して実行して、サービスの利用可能なすべてのメトリックをリスト表示できます。たとえば、**prometheus-example-app** サンプルアプリケーションへのルートを公開し、以下のコマンドを実行して利用可能なすべてのメトリックを表示できます。

```
$ curl http://<example_app_endpoint>/metrics
```

出力例

```
# HELP http_requests_total Count of all HTTP requests
# TYPE http_requests_total counter
http_requests_total{code="200",method="get"} 4
http_requests_total{code="404",method="get"} 2
# HELP version Version information about this binary
# TYPE version gauge
version{version="v0.1.0"} 1
```

関連情報

- Prometheus クライアントライブラリーのドキュメント

9.2. ユーザー定義プロジェクトのメトリクスコレクションの設定

ServiceMonitor リソースを作成して、ユーザー定義プロジェクトのサービスエンドポイントからメトリックを収集できます。これは、アプリケーションが Prometheus クライアントライブラリーを使用してメトリックを **/metrics** の正規の名前に公開していることを前提としています。

このセクションでは、ユーザー定義のプロジェクトでサンプルサービスをデプロイし、次にサービスのモニター方法を定義する **ServiceMonitor** リソースを作成する方法を説明します。

9.2.1. サンプルサービスのデプロイ

ユーザー定義のプロジェクトでサービスのモニタリングをテストするには、サンプルサービスをデプロイできます。

手順

1. サービス設定の YAML ファイルを作成します。この例では、**prometheus-example-app.yaml** という名前です。

2. 以下のデプロイメントおよびサービス設定の詳細をファイルに追加します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns1
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: prometheus-example-app
  name: prometheus-example-app
  namespace: ns1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-example-app
  template:
    metadata:
      labels:
        app: prometheus-example-app
    spec:
      containers:
      - image: ghcr.io/rhobs/prometheus-example-app:0.4.2
        imagePullPolicy: IfNotPresent
        name: prometheus-example-app
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: prometheus-example-app
  name: prometheus-example-app
  namespace: ns1
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
    name: web
  selector:
    app: prometheus-example-app
  type: ClusterIP
```

この設定は、**prometheus-example-app** という名前のサービスをユーザー定義の **ns1** プロジェクトにデプロイします。このサービスは、カスタム **version** メトリックを公開します。

3. 設定をクラスターに適用します。

```
$ oc apply -f prometheus-example-app.yaml
```

サービスをデプロイするには多少時間がかかります。

4. Pod が実行中であることを確認できます。

```
$ oc -n ns1 get pod
```

出力例

```
NAME                                  READY     STATUS    RESTARTS   AGE
prometheus-example-app-7857545cb7-sbgwq   1/1       Running   0          81m
```

9.2.2. サービスのモニター方法の指定

サービスが公開するメトリクスを使用するには、OpenShift Container モニタリングを、**/metrics** エンドポイントからメトリクスを収集できるように設定する必要があります。これは、サービスのモニタリング方法を指定する **ServiceMonitor** カスタムリソース定義、または Pod のモニタリング方法を指定する **PodMonitor** CRD を使用して実行できます。前者の場合は **Service** オブジェクトが必要ですが、後者の場合は不要です。これにより、Prometheus は Pod によって公開されるメトリクスエンドポイントからメトリクスを直接収集することができます。

この手順では、ユーザー定義プロジェクトでサービスの **ServiceMonitor** リソースを作成する方法を説明します。

前提条件

- **cluster-admin** クラスターロールまたは **monitoring-edit** クラスターロールのあるユーザーとしてクラスターにアクセスできる。
- ユーザー定義プロジェクトのモニタリングが有効になっている。
- この例では、**prometheus-example-app** サンプルサービスを **ns1** プロジェクトにデプロイしている。

注記

prometheus-example-app サンプルサービスは TLS 認証をサポートしません。

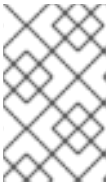
手順

1. **ServiceMonitor** リソース設定の YAML ファイルを作成します。この例では、ファイルは **example-app-service-monitor.yaml** という名前です。
2. 以下の **ServiceMonitor** リソース設定の詳細を追加します。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: prometheus-example-monitor
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - interval: 30s
```

```
    port: web
    scheme: http
  selector:
    matchLabels:
      app: prometheus-example-app
```

これは、**prometheus-example-app** サンプルサービスによって公開されるメトリクスを収集する **ServiceMonitor** リソースを定義します。これには **version** メトリクスが含まれます。



注記

ユーザー定義の namespace の **ServiceMonitor** リソースは、同じ namespace のサービスのみを検出できます。つまり、**ServiceMonitor** リソースの **namespaceSelector** フィールドは常に無視されます。

3. 設定をクラスターに適用します。

```
$ oc apply -f example-app-service-monitor.yaml
```

ServiceMonitor をデプロイするのに多少時間がかかります。

4. **ServiceMonitor** リソースが実行中であることを確認できます。

```
$ oc -n ns1 get servicemonitor
```

出力例

```
NAME                         AGE
prometheus-example-monitor   81m
```

関連情報

- ユーザー定義プロジェクトのモニタリングの有効化
- How to scrape metrics using TLS in a ServiceMonitor configuration in a user-defined project
- PodMonitor API
- ServiceMonitor API

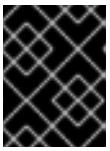
9.3. 利用可能なメトリクスのリストを表示する

クラスター管理者またはすべてのプロジェクトの表示権限を持つユーザーとして、クラスターで使用可能なメトリクスのリストを表示し、リストを JSON 形式で出力できます。

前提条件

- クラスター管理者であるか、**cluster-monitoring-view** クラスターロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Container Platform CLI (**oc**) をインストールしている。
- Thanos Querier の OpenShift Container Platform API ルートを取得しました。

- **oc whoami -t** コマンドを使用してベアラートークンを取得できます。



重要

Thanos Querier API ルートにアクセスするには、ベアラートークン認証のみを使用できます。

手順

1. Thanos Querier の OpenShift Container Platform API ルートを取得していない場合は、以下のコマンドを実行します。

```
$ oc get routes -n openshift-monitoring thanos-querier -o jsonpath='{.status.ingress[0].host}'
```

2. 次のコマンドを実行して、Thanos Querier API ルートから JSON 形式のメトリクスのリストを取得します。このコマンドは、**oc** を使用してベアラートークンで認証します。

```
$ curl -k -H "Authorization: Bearer $(oc whoami -t)"
https://<thanos_querier_route>/api/v1/metadata 1
```

1 **<thanos_querier_route>** を Thanos Querier の OpenShift Container Platform API ルートに置き換えます。

9.4. メトリクスのクエリー

OpenShift Container Platform モニタリングダッシュボードでは、Prometheus のクエリー言語 (PromQL) クエリーを実行し、プロットに可視化されるメトリクスを検査できます。この機能により、クラスターの状態と、モニターしているユーザー定義のワークロードに関する情報が提供されます。

クラスター管理者は、すべての OpenShift Container Platform のコアプロジェクトおよびユーザー定義プロジェクトのメトリックをクエリーできます。

開発者として、メトリクスのクエリー時にプロジェクト名を指定する必要があります。選択したプロジェクトのメトリクスを表示するには、必要な権限が必要です。

9.4.1. クラスター管理者としてのすべてのプロジェクトのメトリックのクエリー

クラスター管理者またはすべてのプロジェクトの表示権限を持つユーザーとして、メトリック UI ですべてのデフォルト OpenShift Container Platform およびユーザー定義プロジェクトのメトリックにアクセスできます。

前提条件

- **cluster-admin** クラスターロールまたはすべてのプロジェクトの表示権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

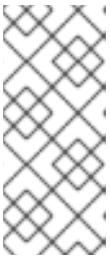
手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブから、**Observe → Metrics** を選択します。

2. 1 つ以上のクエリーを追加するには、次のいずれかを実行します。

オプション	説明
カスタムクエリーを作成します。	Prometheus Query Language (PromQL) クエリーを **Expression** フィールドに追加します。 PromQL 式を入力すると、オートコンプリートの提案がドロップダウンリストに表示されます。これらの提案には、関数、メトリクス、ラベル、および時間トークンが含まれます。キーボードの矢印を使用して提案された項目のいずれかを選択し、Enter を押して項目を式に追加できます。また、マウスポインターを推奨項目の上に移動して、その項目の簡単な説明を表示することもできます。
複数のクエリーを追加します。	**クエリーの追加** を選択します。
既存のクエリーを複製します。	オプションメニュー を選択します クエリーの横にある **Duplicate query** を選択します。
クエリーの実行を無効にします。	オプションメニュー を選択します クエリーの横にある **Disable query** を選択します。

3. 作成したクエリーを実行するには、**Run queries** を選択します。クエリーからのメトリクスはプロットで可視化されます。クエリーが無効な場合は、UI にエラーメッセージが表示されます。



注記

大量のデータで動作するクエリーは、時系列グラフの描画時にタイムアウトするか、ブラウザーをオーバーロードする可能性があります。これを回避するには、**Hide graph** を選択し、メトリクステーブルのみを使用してクエリーを調整します。次に、使用できるクエリーを確認した後に、グラフを描画できるようにプロットを有効にします。

注記

デフォルトでは、クエリーテーブルに、すべてのメトリクスとその現在の値をリスト表示する拡張ビューが表示されます。˅ を選択すると、クエリーの拡張ビューを最小にすることができます。

4. オプション: ページ URL には、実行したクエリーが含まれます。このクエリーのセットを再度使用できるようにするには、この URL を保存します。

5. 視覚化されたメトリクスを調べます。最初に、有効な全クエリーの全メトリクスがプロットに表示されます。次のいずれかを実行して、表示するメトリクスを選択できます。

オプション	説明
クエリーからすべてのメトリクスを非表示にします。	オプションメニューをクリックします  クエリーを選択し、**Hide all series** をクリックします。
特定のメトリクスを非表示にします。	クエリーテーブルに移動し、メトリクス名の近くにある色付きの四角形をクリックします。
プロットを拡大し、時間範囲を変更します。	次のいずれかになります。 • プロットを水平にクリックし、ドラッグして、時間範囲を視覚的に選択します。 • 左上隅のメニューを使用して、時間範囲を選択します。
時間範囲をリセットします。	**Reset zoom** を選択します。
特定の時点でのすべてのクエリーの出力を表示します。	その時点でプロット上にマウスカーソルを置きます。クエリーの出力はポップアップに表示されます。
プロットを非表示にします。	**Hide graph** を選択します。

関連情報

- PromQL クエリーの作成に関する詳細は、Prometheus クエリーについてのドキュメント を参照してください。

9.4.2. 開発者が行うユーザー定義プロジェクトのメトリクスのクエリー

ユーザー定義のプロジェクトのメトリクスには、開発者またはプロジェクトの表示権限を持つユーザーとしてアクセスできます。

Developer パースペクティブには、選択したプロジェクトの事前に定義された CPU、メモリー、帯域幅、およびネットワークパケットのクエリーが含まれます。また、プロジェクトの CPU、メモリー、帯域幅、ネットワークパケット、およびアプリケーションメトリクスについてカスタム Prometheus Query Language (PromQL) クエリーを実行することもできます。

注記

開発者は **Developer** パースペクティブのみを使用でき、**Administrator** パースペクティブは使用できません。開発者は、1度に1つのプロジェクトのメトリクスのみをクエリーできます。

前提条件

- 開発者として、またはメトリクスで表示しているプロジェクトの表示権限を持つユーザーとしてクラスターへのアクセスがある。

- ユーザー定義プロジェクトのモニタリングを有効にしている。
- ユーザー定義プロジェクトにサービスをデプロイしている。
- サービスのモニター方法を定義するために、サービスの **ServiceMonitor** カスタムリソース定義 (CRD) を作成している。

手順

1. OpenShift Container Platform Web コンソールの **Developer** パースペクティブから、**Observe → Metrics** を選択します。
2. **Project:** 一覧でメトリクスで表示するプロジェクトを選択します。
3. **Select query** 一覧からクエリーを選択するか、**Show PromQL** を選択して、選択したクエリーに基づいてカスタム PromQL クエリーを作成します。クエリーからのメトリクスはプロットで可視化されます。

注記

Developer パースペクティブでは、1 度に 1 つのクエリーのみを実行できます。

4. 次のいずれかを実行して、視覚化されたメトリクスを調べます。

オプション	説明
プロットを拡大し、時間範囲を変更します。	次のいずれかになります。 • プロットを水平にクリックし、ドラッグして、時間範囲を視覚的に選択します。 • 左上隅のメニューを使用して、時間範囲を選択します。
時間範囲をリセットします。	**Reset zoom** を選択します。
特定の時点でのすべてのクエリーの出力を表示します。	その時点でプロット上にマウスカーソルを置きます。クエリーの出力はポップアップに表示されます。

関連情報

- PromQL クエリーの作成に関する詳細は、Prometheus クエリーについてのドキュメント を参照してください。

9.5. メトリクスターゲットに関する詳細情報の取得を参照してください。

OpenShift Container Platform Web コンソールの **Administrator** パースペクティブでは、**Metrics Targets** ページを使用して、現在スクレイピングの対象となっているエンドポイントを表示、検索、およびフィルタリングできます。これは、問題の特定とトラブルシューティングに役立ちます。たとえば、ターゲットエンドポイントの現在のステータスを表示して、OpenShift Container Platform Monitoring がターゲットコンポーネントからメトリックをスクレイピングできないのはいつなのかを確認できます。

Metrics targets ページには、デフォルトの OpenShift Container Platform プロジェクトのターゲットとユーザー定義プロジェクトのターゲットが表示されます。

前提条件

- メトリックターゲットを表示するプロジェクトの管理者としてクラスターにアクセスできる。

手順

1. **Administrator** パースペクティブで、**Observe → Targets** を選択します。**Metrics targets** ページが開き、メトリクス用にスクレイピングされているすべてのサービスエンドポイントターゲットのリストが表示されます。
   このページには、デフォルトの OpenShift Container Platform のターゲットとユーザー定義プロジェクトの詳細が表示されます。このページには、ターゲットごとに以下の情報がリスト表示されます。
   - スクレイピングされるサービスエンドポイント URL
   - モニター対象の ServiceMonitor コンポーネント
   - ターゲットの **アップ** または **ダウン** ステータス
   - Namespace
   - 最後のスクレイプ時間
   - 最後のスクレイピングの継続期間
2. オプション: メトリクスターゲットのリストは長くなる場合があります。特定のターゲットを見つけるには、次のいずれかを実行します。

オプション	説明

オプション	説明
ステータスとソースによってターゲットをフィルタリングします。	**Filter** リストでフィルターを選択します。 以下のフィルタリングオプションが利用できます。 ● **ステータス** フィルター: ○ **Up**.ターゲットは現在 up で、メトリックに対してアクティブにスクレイピングされています。 ○ **Down**.ターゲットは現在 down しており、メトリック用にスクレイピングされていません。 ● **Source** フィルター: ○ **Platform**。プラットフォームレベルのターゲットは、デフォルトの Red Hat OpenShift Service on AWS プロジェクトにのみ該当します。これらのプロジェクトは、Red Hat OpenShift Service on AWS のコア機能を提供します。 ○ **User**。ユーザーターゲットは、ユーザー定義プロジェクトに関連します。これらのプロジェクトはユーザーが作成したもので、カスタマイズすることができます。
名前またはラベルでターゲットを検索します。	検索ボックスの横にある **Text** または **Label** フィールドに検索語を入力します。
ターゲットを並べ替えます。	**Endpoint Status**、**Namespace**、**Last Scrape**、および **Scrape Duration** 列ヘッダーの 1 つ以上をクリックします。

3. ターゲットの **Endpoint** 列の URL をクリックし、 **Target details** ページに移動します。このページには、次のようなターゲットに関する情報が表示されます。
   - メトリックのためにスクレイピングされているエンドポイント URL
   - 現在のターゲットのステータス (**Up** または **Down**)
   - namespace へのリンク
   - ServiceMonitor の詳細へのリンク
   - ターゲットに割り当てられたラベル
   - ターゲットがメトリック用にスクレイピングされた直近の時間

第10章 アラートの管理

OpenShift Container Platform 4.15 では、アラート UI を使用してアラート、サイレンス、およびアラートルールを管理できます。

- **アラートルール**。アラートルールには、クラスター内の特定の状態を示す一連の条件が含まれます。アラートは、これらの条件が true の場合にトリガーされます。アラートルールには、アラートのルーティング方法を定義する重大度を割り当てることができます。
- **アラート**。アラートは、アラートルールで定義された条件が true の場合に発生します。アラートは、一連の状況が OpenShift Container Platform クラスター内で明確であることを示す通知を提供します。
- **サイレンス**。サイレンスをアラートに適用し、アラートの条件が true の場合に通知が送信されることを防ぐことができます。初期通知後はアラートをミュートにして、根本的な問題の解決に取り組むことができます。

注記

アラート UI で利用可能なアラート、サイレンス、およびアラートルールは、アクセス可能なプロジェクトに関連付けられます。たとえば、**cluster-admin** ロールを持つユーザーとしてログインしている場合は、すべてのアラート、サイレント、およびアラートルールにアクセスできます。

管理者以外のユーザーは、次のユーザーロールが割り当てられていれば、アラートを作成して無効にできます。

- Alertmanager へのアクセスを許可する **cluster-monitoring-view** クラスターロール。
- **monitoring-alertmanager-edit** ロール。これにより、Web コンソールの **Administrator** パースペクティブでアラートを作成して無効にできます。
- **monitoring-rules-edit** クラスターロール。これにより、Web コンソールの **Developer** パースペクティブでアラートを作成して無効にできます。

10.1. ADMINISTRATOR および DEVELOPER パースペクティブでのアラート UI へのアクセス

アラート UI は、OpenShift Container Platform Web コンソールの **Administrator** および **Developer** パースペクティブからアクセスできます。

- **Administrator** パースペクティブで、**Observe** → **Alerting** に移動します。このパースペクティブのアラート UI には主要なページが 3 つあり、それが **Alerts** ページ、**Silences** ページ、**Alerting rules** ページです。
- **Developer** パースペクティブで、**Observe** → **<project_name>** → **Alerts** に移動します。このパースペクティブのアラートでは、サイレンスおよびアラートルールはすべて **Alerts** ページで管理されます。**Alerts** ページに表示される結果は、選択されたプロジェクトに固有のものです。

注記

Developer パースペクティブでは、コア OpenShift Container Platform と、 **Project: <project_name>** リスト内のアクセス可能なユーザー定義プロジェクトから選択できます。ただし、クラスター管理者としてログインしていない場合、コア OpenShift Container Platform プロジェクトに関連するアラート、サイレンス、およびアラートルールは表示されません。

10.2. アラート、サイレンスおよびアラートルールの検索およびフィルター

アラート UI に表示されるアラート、サイレンス、およびアラートルールをフィルターできます。このセクションでは、利用可能なフィルターオプションのそれぞれについて説明します。

アラートフィルターについて

Administrator パースペクティブでは、アラート UI の **Alerts** ページに、デフォルトの OpenShift Container Platform プロジェクトおよびユーザー定義プロジェクトに関連するアラートの詳細が提供されます。このページには、各アラートの重大度、状態、およびソースの概要が含まれます。アラートが現在の状態に切り替わった時間も表示されます。

アラートの状態、重大度、およびソースでフィルターできます。デフォルトでは、**Firing** の **Platform** アラートのみが表示されます。以下では、それぞれのアラートフィルターオプションについて説明します。

- **State** フィルター:
  - **Firing**。アラート条件が true で、オプションの **for** の期間を経過しているためにアラートが実行されます。条件が true である間、アラートの発生が続きます。
  - **Pending**。アラートはアクティブですが、アラート実行前のアラートルールに指定される期間待機します。
  - **Silenced**。アラートは定義された期間についてサイレンスにされるようになりました。定義するラベルセレクターのセットに基づいてアラートを一時的にミュートします。リストされたすべての値または正規表現に一致するアラートの土は送信されません。
- **Severity** フィルター:
  - **Critical**。アラートをトリガーした状態は重大な影響を与える可能性があります。このアラートには、実行時に早急な対応が必要となり、通常は個人または緊急対策チーム (Critical Response Team) に送信先が設定されます。
  - **Warning**。アラートは、問題の発生を防ぐために注意が必要になる可能性のある問題についての警告通知を提供します。通常、警告は早急な対応を要さないレビュー用にチケットシステムにルート指定されます。
  - **Info**。アラートは情報提供のみを目的として提供されます。
  - **None**。アラートには重大度が定義されていません。
  - また、ユーザー定義プロジェクトに関連するアラートの重大度の定義を作成することもできます。
- **Source** フィルター:
  - **Platform**。プラットフォームレベルのアラートは、デフォルトの OpenShift Container Platform プロジェクトにのみ関連します。これらのプロジェクトは OpenShift Container Platform のコア機能を提供します。

- **User**。ユーザーアラートはユーザー定義のプロジェクトに関連します。これらのアラートはユーザーによって作成され、カスタマイズ可能です。ユーザー定義のワークロードモニタリングはインストール後に有効にでき、独自のワークロードへの可観測性を提供します。

サイレンスフィルターについて
Administrator パースペクティブでは、アラート UI の **Silences** ページには、デフォルトの OpenShift Container Platform およびユーザー定義プロジェクトのアラートに適用されるサイレンスについての詳細が示されます。このページには、それぞれのサイレンスの状態の概要とサイレンスが終了する時間の概要が含まれます。

サイレンス状態でフィルターを実行できます。デフォルトでは、**Active** および **Pending** のサイレンスのみが表示されます。以下は、それぞれのサイレンス状態のフィルターオプションについて説明しています。

- **State** フィルター:
  - **Active**。サイレンスはアクティブで、アラートはサイレンスが期限切れになるまでミュートされます。
  - **Pending**。サイレンスがスケジュールされており、アクティブな状態ではありません。
  - **Expired**アラートの条件が true の場合は、サイレンスが期限切れになり、通知が送信されます。

アラートルールフィルターについて
Administrator パースペクティブでは、アラート UI の **Alerting rules** ページに、デフォルトの OpenShift Container Platform およびユーザー定義プロジェクトに関連するアラートルールの詳細が示されます。このページには、各アラートルールの状態、重大度およびソースの概要が含まれます。

アラート状態、重大度、およびソースを使用してアラートルールをフィルターできます。デフォルトでは、**プラットフォーム**のアラートルールのみが表示されます。以下では、それぞれのアラートルールのフィルターオプションを説明します。

- **Alert state** フィルター:
  - **Firing**。アラート条件が true で、オプションの **for** の期間を経過しているためにアラートが実行されます。条件が true である間、アラートの発生が続きます。
  - **Pending**。アラートはアクティブですが、アラート実行前のアラートルールに指定される期間待機します。
  - **Silenced**。アラートは定義された期間についてサイレンスにされるようになりました。定義するラベルセレクターのセットに基づいてアラートを一時的にミュートします。リストされたすべての値または正規表現に一致するアラートの土は送信されません。
  - **Not Firing**アラートは実行されません。
- **Severity** フィルター:
  - **Critical**。アラートルールで定義される状態は重大な影響を与える可能性があります。true の場合は、この状態に早急な対応が必要です。通常、ルールに関連するアラートは個別または緊急対策チーム (Critical Response Team) に送信先が設定されます。
  - **Warning**。アラートルールで定義される状態は、問題の発生を防ぐために注意を要する場合があります。通常、ルールに関連するアラートは早急な対応を要さないレビュー用にチケットシステムにルート指定されます。

- **Info**。アラートルールは情報アラートのみを提供します。
- **None**。アラートルールには重大度が定義されていません。
- ユーザー定義プロジェクトに関連するアラートルールのカスタム重大度定義を作成することもできます。

- **Source** フィルター:
  - **Platform**。プラットフォームレベルのアラートルールは、デフォルトの OpenShift Container Platform プロジェクトにのみ関連します。これらのプロジェクトは OpenShift Container Platform のコア機能を提供します。
  - **User**。ユーザー定義のワークロードアラートルールは、ユーザー定義プロジェクトに関連します。これらのアラートルールはユーザーによって作成され、カスタマイズ可能です。ユーザー定義のワークロードモニタリングはインストール後に有効にでき、独自のワークロードへの可観測性を提供します。

Developer パースペクティブでのアラート、サイレンスおよびアラートルールの検索およびフィルター

Developer パースペクティブ では、アラート UI の **Alerts** ページに、選択したプロジェクトに関連するアラートとサイレンスを組み合わせたビューが提供されています。規定するアラートルールへのリンクが表示されるアラートごとに提供されます。

このビューでは、アラートの状態と重大度でフィルターを実行できます。デフォルトで、プロジェクトへのアクセス権限がある場合は、選択されたプロジェクトのすべてのアラートが表示されます。これらのフィルターは **Administrator** パースペクティブについて記載されているフィルターと同じです。

10.3. アラート、サイレンスおよびアラートルールについての情報の取得

アラート UI は、アラートおよびそれらを規定するアラートルールおよびサイレンスについての詳細情報を提供します。

前提条件

- 開発者、またはアラートを表示するプロジェクトの表示権限を持つユーザーとして、クラスターにアクセスできる。

手順

Administrator パースペクティブでアラートの情報を取得するには、以下を実行します。

1. OpenShift Container Platform Web コンソールを開き、**Observe → Alerting → Alerts** ページに移動します。
2. オプション: 検索リストで **Name** フィールドを使用し、アラートを名前で検索します。
3. オプション: **Filter** リストでフィルターを選択し、アラートを状態、重大度およびソースでフィルターします。
4. オプション: 1 つ以上の **Name**、**Severity**、**State**、および **Source** 列ヘッダーをクリックし、アラートを並べ替えます。
5. アラートの名前をクリックして、**Alert details** ページを表示します。このページには、アラートの時系列データを示すグラフが含まれます。アラートに関する次の情報も提供されます。
   - アラートの説明

- アラートに関連付けられたメッセージ
- アラートに割り当てられるラベル
- アラートを規定するアラートルールへのリンク
- アラートが存在する場合のアラートのサイレンス

Administrator パースペクティブでサイレンスの情報を取得するには、以下を実行します。

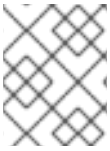
1. **Observe → Alerting → Silences** ページに移動します。
2. オプション: **Search by name** フィールドを使用し、サイレンスを名前でフィルターします。
3. オプション: **Filter** リストでフィルターを選択し、サイレンスをフィルターします。デフォルトでは、**Active** および **Pending** フィルターが適用されます。
4. オプション: **Name**、**Firing alerts**、**State**、**Creator** 列のヘッダーを 1 つ以上クリックして、サイレンスを並べ替えます。
5. サイレンスの名前を選択すると、その **Silence details** ページが表示されます。このページには、以下の詳細が含まれます。
   - アラート仕様
   - 開始時間
   - 終了時間
   - サイレンス状態
   - 発生するアラートの数およびリスト

Administrator パースペクティブでアラートルールの情報を取得するには、以下を実行します。

1. **Observe → Alerting → Alerting rules** ページに移動します。
2. オプション: **Filter** 一覧でフィルターを選択し、アラートルールを状態、重大度およびソースでフィルターします。
3. オプション: **Name**、**Severity**、**Alert State**、**Source** 列のヘッダーを 1 つ以上クリックし、アラートルールを並べ替えます。
4. アラートルールの名前を選択して、その **Alerting rule details** ページを表示します。このページには、アラートルールに関する以下の情報が含まれます。
   - アラートルール名、重大度、説明
   - アラートを発動する条件を定義する式
   - 条件が true で持続してアラートが発生するまでの期間
   - アラートルールで管理される各アラートのグラフ。アラートがh集うする値が表示されます。
   - アラートルールで管理されるすべてのアラートを示す表。

Developer パースペクティブでアラート、サイレンス、およびアラートルールの情報を取得するには、以下を実行します。

1. **Observe → <project_name> → Alerts** ページに移動します。
2. アラート、サイレンス、またはアラートルールの詳細を表示します。
   - **Alert details** を表示するには、アラート名の横にある大なり記号 (>) をクリックし、リストからアラートを選択します。
   - **Silence details** を表示するには、**Alert details** ページの **Silenced by** セクションでサイレンスを選択します。**Silence details** ページには、以下の情報が含まれます。
     - アラート仕様
     - 開始時間
     - 終了時間
     - サイレンス状態
     - 発生するアラートの数およびリスト
   - **Alerting rule details** を表示するには、**Alerts** ページのアラートの横にある メニュー をクリックし、次に **View Alerting Rule** をクリックします。



注記

選択したプロジェクトに関連するアラート、サイレンスおよびアラートルールのみが **Developer** パースペクティブに表示されます。

関連情報

- 特定の OpenShift Container Platform モニタリングアラートをトリガーする問題を診断および解決するには Cluster Monitoring Operator runbook を参照してください。

10.4. サイレンスの管理

OpenShift Container Platform Web コンソールの**Administrator** パースペクティブと**Developer** パースペクティブの両方で、アラートのサイレンスを作成できます。サイレンスを作成すると、アラートが発生したときにアラートに関する通知を受信しなくなります。

サイレントの作成は、最初のアラート通知を受信し、アラートの発生の原因となっている根本的な問題を解決するまでの間、さらなる通知を受け取りたくないシナリオで役立ちます。

サイレンスの作成時に、サイレンスをすぐにアクティブにするか、後にアクティブにするかを指定する必要があります。また、サイレンスの有効期限を設定する必要もあります。

サイレンスを作成した後、それらを表示、編集、および期限切れにすることができます。

10.4.1. アラートをサイレントにする

特定のアラート、または定義する仕様に一致するアラートのいずれかをサイレンスにすることができます。

前提条件

- クラスター管理者の場合は、**cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできます。
- 管理者以外のユーザーの場合は、次のユーザーロールを持つユーザーとしてクラスターにアクセスできます。
  - Alertmanager へのアクセスを許可する **cluster-monitoring-view** クラスターロール。
  - **monitoring-alertmanager-edit** ロール。これにより、Web コンソールの **Administrator** パースペクティブでアラートを作成して無効にできます。
  - **monitoring-rules-edit** クラスターロール。これにより、Web コンソールの **Developer** パースペクティブでアラートを作成して無効にできます。

手順

Administrator パースペクティブで特定のアラートをサイレントにするには、以下を行います。

1. OpenShift Container Platform Web コンソールで、**Observe → Alerting → Alerts** に移動します。
2. サイレントにしたいアラートに対して、 をクリックし、**Silence alert** を選択して、選択したアラートのデフォルト設定を含む **Silence alert** ページを開きます。
3. オプション: サイレントのデフォルト設定の詳細を変更します。

注記

サイレンスを保存する前にコメントを追加する必要があります。

4. サイレントを保存するには、**Silence** をクリックします。

Developer パースペクティブで特定のアラートをサイレントにするには、以下を行います。

1. OpenShift Container Platform Web コンソールで、**Observe → <project_name> → Alerts** に移動します。
2. 必要に応じて、アラート名の横にある大なり記号 (>) を選択し、アラートの詳細を展開します。
3. 展開されたビューでアラートメッセージをクリックすると、そのアラートの **Alert details** ページが開きます。
4. **Silence alert** をクリックして、アラートのデフォルト設定を含む **Silence alert** ページを開きます。
5. オプション: サイレントのデフォルト設定の詳細を変更します。

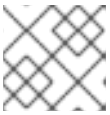
注記

サイレンスを保存する前にコメントを追加する必要があります。

6. サイレントを保存するには、**Silence** をクリックします。

Administrator パースペクティブでサイレンス設定を作成して一連のアラートをサイレントにするには、次の手順を実行します。

1. OpenShift Container Platform Web コンソールで、**Observe → Alerting → Silences** に移動します。
2. **Create silence** をクリックします。
3. **Create silence** フォームで、アラートのスケジュール、期間、およびラベルの詳細を設定します。



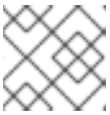
注記

サイレンスを保存する前にコメントを追加する必要があります。

4. 入力したラベルと一致するアラートのサイレンスを作成するには、**Silence** をクリックします。

Developer パースペクティブでサイレント設定を作成して一連のアラートをサイレンスにするには、次の手順を実行します。

1. OpenShift Container Platform Web コンソールで、**Observe → <project_name> → Silences** に移動します。
2. **Create silence** をクリックします。
3. **Create silence** ページで、アラートの期間とラベルの詳細を設定します。



注記

サイレンスを保存する前にコメントを追加する必要があります。

4. 入力したラベルと一致するアラートのサイレンスを作成するには、**Silence** をクリックします。

10.4.2. サイレンスの編集

サイレンスを編集すると、既存のサイレンスが期限切れになり、変更された設定で新しいサイレンスが作成されます。

前提条件

- クラスター管理者の場合は、**cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできます。
- 管理者以外のユーザーの場合は、次のユーザーロールを持つユーザーとしてクラスターにアクセスできます。
  - Alertmanager へのアクセスを許可する **cluster-monitoring-view** クラスターロール。
  - **monitoring-alertmanager-edit** ロール。これにより、Web コンソールの **Administrator** パースペクティブでアラートを作成して無効にできます。
  - **monitoring-rules-edit** クラスターロール。これにより、Web コンソールの **Developer** パースペクティブでアラートを作成して無効にできます。

手順

Administrator パースペクティブでサイレンスを編集するには、以下を実行します。

1. **Observe** → **Alerting** → **Silences** に移動します。
2. 変更するサイレンスの  をクリックして **Edit silence** を選択します。
   または、**Actions** をクリックし、サイレンスの **Silence details** ページで **Edit silence** を選択することもできます。
3. **Edit silence** ページで変更を加え、**Silence** をクリックします。これにより、既存のサイレンスが期限切れになり、更新された設定でサイレンスが作成されます。

Developer パースペクティブでサイレンスを編集するには、以下を実行します。

1. **Observe** → **<project_name>** → **Silences** に移動します。
2. 変更するサイレンスの  をクリックして **Edit silence** を選択します。
   または、**Actions** をクリックし、サイレンスの **Silence details** ページで **Edit silence** を選択することもできます。
3. **Edit silence** ページで変更を加え、**Silence** をクリックします。これにより、既存のサイレンスが期限切れになり、更新された設定でサイレンスが作成されます。

10.4.3. 有効期限切れにするサイレンス

単一のサイレンスまたは複数のサイレンスを期限切れにすることができます。サイレンスを期限切れにすると、そのサイレンスは永久に非アクティブ化されます。

注記

期限切れで沈黙したアラートは削除できません。120 時間を超えて期限切れになったサイレンスはガベージコレクションされます。

前提条件

- クラスター管理者の場合は、**cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできます。
- 管理者以外のユーザーの場合は、次のユーザーロールを持つユーザーとしてクラスターにアクセスできます。
  - Alertmanager へのアクセスを許可する **cluster-monitoring-view** クラスターロール。
  - **monitoring-alertmanager-edit** ロール。これにより、Web コンソールの **Administrator** パースペクティブでアラートを作成して無効にできます。
  - **monitoring-rules-edit** クラスターロール。これにより、Web コンソールの **Developer** パースペクティブでアラートを作成して無効にできます。

手順

Administrator パースペクティブでサイレンスを期限切れにするには、以下を行います。

1. **Observe** → **Alerting** → **Silences** に移動します。

2. 期限切れにするサイレンスについては、対応する行のチェックボックスを選択します。

3. **Expire 1 silence** をクリックして選択した 1 つのサイレンスを期限切れにするか、**Expire <n> silences** をクリックして複数の沈黙を期限切れにします (**<n>** は選択した沈黙の数になります)。
または、単一の沈黙を期限切れにするには、**Actions** をクリックし、サイレンスの**Silence details** ページで **Expire silence** を選択します。

Developer パースペクティブでサイレンスを期限切れにするには、以下を実行します。

1. **Observe → <project_name> → Silences** に移動します。

2. 期限切れにするサイレンスについては、対応する行のチェックボックスを選択します。

3. **Expire 1 silence** をクリックして選択した 1 つのサイレンスを期限切れにするか、**Expire <n> silences** をクリックして複数の沈黙を期限切れにします (**<n>** は選択した沈黙の数になります)。
または、単一の沈黙を期限切れにするには、**Actions** をクリックし、サイレンスの**Silence details** ページで **Expire silence** を選択します。

10.5. コアプラットフォームモニタリングのアラートルールの管理

OpenShift Container Platform 4.15 モニタリングには、プラットフォームメトリクスのデフォルトのアラートルールセットが同梱されます。クラスター管理者は、このルールセットを 2 つの方法でカスタマイズできます。

- しきい値を調整するか、ラベルを追加および変更して、既存のプラットフォームのアラートルールの設定を変更します。たとえば、アラートの **severity** ラベルを **warning** から **critical** に変更すると、アラートのフラグが付いた問題のルーティングおよびトリアージに役立ちます。

- **openshift-monitoring** namaespace のコアプラットフォームメトリックに基づいてクエリー式を作成することにより、新しいカスタムアラートルールを定義して追加します。

コアプラットフォームのアラートルールについての考慮事項

- 新規のアラートルールはデフォルトの OpenShift Container Platform モニタリングメトリクスをベースとする必要があります。

- アラートルールのみを追加および変更できます。新しい記録ルールを作成したり、既存の記録ルールを変更したりすることはできません。

- **AlertRelabelConfig** オブジェクトを使用して既存のプラットフォームのアラートルールを変更する場合、変更は Prometheus アラート API に反映されません。そのため、削除されたアラートは Alertmanager に転送されていなくても OpenShift Container Platform Web コンソールに表示されます。さらに、**severity** ラベルの変更など、アラートへの変更は Web コンソールには表示されません。

10.5.1. コアプラットフォームモニタリングのアラートルールを最適化するためのヒント

組織の特定のニーズに合わせてコアプラットフォームのアラートルールをカスタマイズする場合は、次のガイドラインに従って、カスタマイズされたルールが効率的かつ効果的であることを確認してください。

- **新しいルールの数を最小限に抑えます**。特定の要件に不可欠なルールのみを作成します。ルールの数を最小限に抑えることで、より管理しやすく、焦点を絞ったアラートシステムをモニタリング環境に作成できます。

- **原因ではなく症状に焦点を当てます**。根本的な原因ではなく症状をユーザーに通知するルールを作成します。このアプローチにより、関連する症状がユーザーに即座に通知され、アラートがトリガーされた後に根本原因を調査できるようになります。この戦略により、作成する必要があるルールの総数も大幅に削減されます。

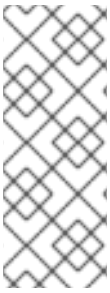
- **変更を実装する前に、ニーズを計画し、評価します**。まず、どの症状が重要であり、これらの症状が発生した場合にユーザーにどのようなアクションをとってもらいたいかを決定します。次に、既存のルールを評価し、症状ごとにまったく新しいルールを作成するのではなく、ニーズを満たすためにルールを変更できるかどうかを判断します。既存のルールを変更し、新しいルールを慎重に作成することで、アラートシステムを合理化できます。

- **クリアなアラートメッセージングを提供します**。アラートメッセージを作成するときは、症状、考えられる原因、推奨されるアクションを説明します。明確で簡潔な説明と、トラブルシューティング手順または詳細情報へのリンクを含めます。そうすることで、ユーザーは状況を迅速に評価し、適切に対応することができます。

- **重大度レベルを含めます**。ルールに重大度レベルを割り当てて、症状が発生してアラートがトリガーされたときにユーザーがどのように反応する必要があるかを示します。たとえば、アラートを **Critical** として分類すると、個人または重要な対応チームが直ちに対応する必要があることを示します。重大度レベルを定義することで、ユーザーがアラートへの対応方法を理解し、最も緊急性の高い問題に迅速な対応を確実に受けられるようになります。

10.5.2. 新規アラートルールの作成

クラスター管理者は、プラットフォームメトリックに基づいて新規のアラートルールを作成できます。これらのアラートルールは、選択したメトリックの値に基づいてアラートをトリガーします。



注記

- 既存のプラットフォームアラートルールに基づいてカスタマイズされた **AlertingRule** リソースを作成する場合は、元のアラートをサイレントに設定して、競合するアラートを受信しないようにします。

- ユーザーがアラートの影響と原因を理解できるように、アラートルールにアラートメッセージと重大度値が含まれていることを確認します。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできる。

- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-monitoring** namespace に **example-alerting-rule.yaml** という名前の新規 YAML 設定ファイルを作成します。

2. **AlertingRule** リソースを YAML ファイルに追加します。以下の例では、デフォルトの **Watchdog** アラートと同様に **example** という名前の新規アラートルールを作成します。

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: example
  namespace: openshift-monitoring
```

```
spec:
  groups:
  - name: example-rules
    rules:
    - alert: ExampleAlert 1
      expr: vector(1) 2
      labels:
        severity: warning 3
      annotations:
        message: This is an example alert. 4
```

1. 作成する必要のあるアラートルールの名前。
2. 新規ルールを定義する PromQL クエリー式。
3. アラートに割り当てられた重大度。
4. アラートに関連付けられたメッセージ。

3. 設定ファイルをクラスターに適用します。

```
$ oc apply -f example-alerting-rule.yaml
```

10.5.3. コアプラットフォームのアラートルールの変更

クラスター管理者は、Alertmanager がコアプラットフォームアラートをレシーバーにルーティングする前に変更できます。たとえば、アラートの重大度のラベルを変更したり、カスタムラベルを追加したり、アラートの送信から Alertmanager に送信されないようにしたりできます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **openshift-monitoring** namespace に **example-modified-alerting-rule.yaml** という名前の新規 YAML 設定ファイルを作成します。
2. **AlertRelabelConfig** リソースを YAML ファイルに追加します。以下の例では、デフォルトのプラットフォーム **watchdog** アラートルールの **severity** 設定を **critical** に変更します。

```
apiVersion: monitoring.openshift.io/v1
kind: AlertRelabelConfig
metadata:
  name: watchdog
  namespace: openshift-monitoring
spec:
  configs:
  - sourceLabels: [alertname,severity] 1
    regex: "Watchdog;none" 2
```

```
    targetLabel: severity 3
    replacement: critical 4
    action: Replace 5
```

1. 変更する値のソースラベル。
2. **sourceLabels** の値が一致する正規表現。
3. 変更する値のターゲットラベル。
4. ターゲットラベルを置き換える新しい値。
5. 正規表現の一致に基づいて古い値を置き換える再ラベルアクション。デフォルトのアクションは **Replace** です。その他に使用可能な値は、**Keep**、**Drop** 、**HshMod**、**LabelMap**、**LabelDrop**、および **LabelKeep** です。

3. 設定ファイルをクラスターに適用します。

```
$ oc apply -f example-modified-alerting-rule.yaml
```

関連情報

- OpenShift Container Platform 4.15 モニタリングアーキテクチャーに関する詳細は、モニタリングの概要 を参照してください。
- アラートルールの詳細は、Alertmanager ドキュメント を参照してください。
- 再ラベル付けの動作に関する詳細は、Prometheus の再ラベル付けに関するドキュメント を参照してください。
- アラートの最適化に関する追加のガイドラインは、Prometheus アラートのドキュメント を参照してください。

10.6. ユーザー定義プロジェクトのアラートルールの管理

OpenShift Container Platform モニタリングには、デフォルトのアラートルールのセットが同梱されます。クラスター管理者は、デフォルトのアラートルールを表示できます。

OpenShift Container Platform 4.15 では、ユーザー定義プロジェクトでアラートルールを作成、表示、編集、削除できます。

アラートルールについての考慮事項

- デフォルトのアラートルールは OpenShift Container Platform クラスター用に使用され、それ以外の目的では使用されません。
- 一部のアラートルールには、複数の意図的に同じ名前が含まれます。それらは同じイベントについてのアラートを送信しますが、それぞれ異なるしきい値、重大度およびそれらの両方が設定されます。
- 抑制 (inhibition) ルールは、高い重大度のアラートが実行される際に実行される低い重大度のアラートの通知を防ぎます。

10.6.1. ユーザー定義プロジェクトのアラートの最適化

アラートルールの作成時に以下の推奨事項を考慮して、独自のプロジェクトのアラートを最適化できます。

- **プロジェクト用に作成するアラートルールの数を最小限にします**。影響を与える状況を通知するアラートルールを作成します。影響を与えない条件に対して多数のアラートを生成すると、関連するアラートに気づくのがさらに困難になります。
- **原因ではなく現象についてのアラートルールを作成します**。根本的な原因に関係なく、状態を通知するアラートルールを作成します。次に、原因を調査できます。アラートルールのそれぞれが特定の原因にのみ関連する場合に、さらに多くのアラートルールが必要になります。そのため、いくつかの原因は見落される可能性があります。
- **アラートルールを作成する前にプランニングを行います**。重要な現象と、その発生時に実行するアクションを決定します。次に、現象別にアラートルールをビルドします。
- **クリアなアラートメッセージングを提供します**。アラートメッセージに現象および推奨されるアクションを記載します。
- **アラートルールに重大度レベルを含めます**。アラートの重大度は、報告される現象が生じた場合に取るべき対応によって異なります。たとえば、現象に個人または緊急対策チーム (Critical Response Team) による早急な対応が必要な場合は、重大アラートをトリガーする必要があります。

関連情報

- アラートの最適化に関する追加のガイドラインは、Prometheus アラートのドキュメント を参照してください。
- OpenShift Container Platform 4.15 モニタリングアーキテクチャーの詳細は、モニタリングの概要 を参照してください。

10.6.2. ユーザー定義プロジェクトのアラートルールの作成

ユーザー定義プロジェクトのアラートルールを作成する場合は、新しいルールを定義する際に次の主要な動作と重要な制限事項を考慮してください。

- ユーザー定義のアラートルールには、コアプラットフォームのモニタリングからのデフォルトメトリクスに加えて、独自のプロジェクトが公開したメトリクスを含めることができます。別のユーザー定義プロジェクトのメトリクスを含めることはできません。
  たとえば、**ns1** ユーザー定義プロジェクトのアラートルールでは、CPU やメモリーメトリクスなどのコアプラットフォームメトリクスに加えて、**ns1** プロジェクトが公開したメトリクスも使用できます。ただし、ルールには、別の **ns2** ユーザー定義プロジェクトからのメトリクスを含めることはできません。
- レイテンシーを短縮し、コアプラットフォームモニタリングコンポーネントの負荷を最小限に抑えるために、ルールに **openshift.io/prometheus-rule-evaluation-scope: leaf-prometheus** ラベルを追加できます。このラベルは、**openshift-user-workload-monitoring** プロジェクトにデプロイされた Prometheus インスタンスのみにアラートルールの評価を強制し、Thanos Ruler インスタンスによる評価を防ぎます。

重要

アラートルールにこのラベルが付いている場合、そのアラートルールはユーザー定義プロジェクトが公開するメトリクスのみを使用できます。デフォルトのプラットフォームメトリクスに基づいて作成したアラートルールでは、アラートがトリガーされない場合があります。

10.6.3. ユーザー定義プロジェクトのアラートルールの作成

ユーザー定義のプロジェクトに対してアラートルールを作成できます。これらのアラートルールは、選択したメトリクスの値に基づいてアラートをトリガーします。

注記

- アラートルールを作成すると、別のプロジェクトに同じ名前のルールが存在する場合でも、そのルールにプロジェクトラベルが適用されます。
- ユーザーがアラートの影響と原因を理解できるように、アラートルールにアラートメッセージと重大度値が含まれていることを確認します。

前提条件

- ユーザー定義プロジェクトのモニタリングが有効になっている。
- アラートルールを作成する必要のある namespace の **monitoring-rules-edit** クラスターロールを持つユーザーとしてログインしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. アラートルールの YAML ファイルを作成します。この例では、**example-app-alerting-rule.yaml** という名前です。
2. アラートルール設定を YAML ファイルに追加します。以下の例では、**example-alert** という名前の新規アラートルールを作成します。アラートルールは、サンプルサービスによって公開される **version** メトリクスが **0** になるとアラートを実行します。

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-alert
  namespace: ns1
spec:
  groups:
  - name: example
    rules:
    - alert: VersionAlert (1)
      expr: version{job="prometheus-example-app"} == 0 (2)
      labels:
        severity: warning (3)
      annotations:
        message: This is an example alert. (4)
```

1. 作成する必要のあるアラートルールの名前。
2. 新規ルールを定義する PromQL クエリー式。
3. アラートに割り当てられた重大度。
4. アラートに関連付けられたメッセージ。

3. 設定ファイルをクラスターに適用します。

```
$ oc apply -f example-app-alerting-rule.yaml
```

関連情報

- OpenShift Container Platform 4.15 モニタリングアーキテクチャーに関する詳細は、モニタリングの概要 を参照してください。

10.6.4. ユーザー定義プロジェクトのアラートルールへのアクセス

ユーザー定義プロジェクトのアラートルールを一覧表示するには、プロジェクトの **monitoring-rules-view** クラスターロールが割り当てられている必要があります。

前提条件

- ユーザー定義プロジェクトのモニタリングが有効になっている。
- プロジェクトの **monitoring-rules-view** クラスターロールを持つユーザーとしてログインしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **<project>** でアラートルールを一覧表示するには、以下を実行します。

```
$ oc -n <project> get prometheusrule
```

2. アラートルールの設定をリスト表示するには、以下を実行します。

```
$ oc -n <project> get prometheusrule <rule> -o yaml
```

10.6.5. 単一ビューでのすべてのプロジェクトのアラートルールのリスト表示

クラスター管理者は、OpenShift Container Platform のコアプロジェクトおよびユーザー定義プロジェクトのアラートルールを単一ビューでリスト表示できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **Administrator** パースペクティブで、**Observe → Alerting → Alerting rules** に移動します。
2. **Filter** ドロップダウンメニューで、**Platform** および **User** ソースを選択します。

注記

Platform ソースはデフォルトで選択されます。

10.6.6. ユーザー定義プロジェクトのアラートルールの削除

ユーザー定義プロジェクトのアラートルールを削除できます。

前提条件

- ユーザー定義プロジェクトのモニタリングが有効になっている。
- アラートルールを作成する必要のある namespace の **monitoring-rules-edit** クラスターロールを持つユーザーとしてログインしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

- **<namespace>** のルール **<foo>** を削除するには、以下を実行します。

```
$ oc -n <namespace> delete prometheusrule <foo>
```

関連情報

- Alertmanager ドキュメント を参照してください。

10.7. 外部システムへの通知の送信

OpenShift Container Platform 4.15 では、実行するアラートをアラート UI に表示できます。アラートは、デフォルトでは通知システムに送信されるように設定されません。以下のレシーバータイプにアラートを送信するように OpenShift Container Platform を設定できます。

- PagerDuty
- Webhook
- Email
- Slack

レシーバーへのアラートのルートを指定することにより、障害が発生する際に適切なチームに通知をタイムリーに送信できます。たとえば、重大なアラートには早急な対応が必要となり、通常は個人または緊急対策チーム (Critical Response Team) に送信先が設定されます。重大ではない警告通知を提供するアラートは、早急な対応を要さないレビュー用にチケットシステムにルート指定される可能性があります。

Watchdog アラートの使用によるアラートが機能することの確認

OpenShift Container Platform モニタリングには、継続的に実行される Watchdog アラートが含まれます。Alertmanager は、Watchdog のアラート通知を設定された通知プロバイダーに繰り返し送信します。通常、プロバイダーは Watchdog アラートの受信を停止する際に管理者に通知するように設定されます。このメカニズムは、Alertmanager と通知プロバイダー間の通信に関連する問題を迅速に特定するのに役立ちます。

10.7.1. アラートレシーバーの設定

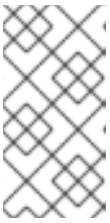
アラートレシーバーを設定して、クラスターについての重要な問題について把握できるようにします。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。

手順

1. **Administrator** パースペクティブで、**Administration → Cluster Settings → Configuration → Alertmanager** に移動します。



注記

または、通知ドロワーから同じページに移動することもできます。OpenShift Container Platform Web コンソールの右上にあるベルのアイコンを選択し、**AlertmanagerReceiverNotConfigured** アラートで **Configure** を選択します。

2. ページの **Receivers** セクションで、**Create Receiver** をクリックします。
3. **Create Receiver** フォームで、**Receiver name** を追加し、リストから **Receiver type** を選択します。
4. レシーバー設定を編集します。
   - PagerDuty receiver の場合:
     a. 統合のタイプを選択し、PagerDuty 統合キーを追加します。
     b. PagerDuty インストールの URL を追加します。
     c. クライアントおよびインシデントの詳細または重大度の指定を編集する場合は、**Show advanced configuration** をクリックします。
   - Webhook receiver の場合:
     a. HTTP POST リクエストを送信するエンドポイントを追加します。
     b. デフォルトオプションを編集して解決したアラートを receiver に送信する場合は、**Show advanced configuration** をクリックします。
   - メール receiver の場合:
     a. 通知を送信するメールアドレスを追加します。
     b. SMTP 設定の詳細を追加します。これには、通知の送信先のアドレス、メールの送信に使用する smarthost およびポート番号、SMTP サーバーのホスト名、および認証情報を含む詳細情報が含まれます。

c. TLS が必要かどうかを選択します。

d. 解決済みのアラートが receiver に送信されないようにデフォルトオプションを編集する、またはメール通知設定のボディーを編集する必要がある場合は、**Show advanced configuration** をクリックします。

- Slack receiver の場合:

a. Slack Webhook の URL を追加します。

b. 通知を送信する Slack チャネルまたはユーザー名を追加します。

c. デフォルトオプションを編集して解決済みのアラートが receiver に送信されないようにしたり、アイコンおよびユーザー設定を編集する必要がある場合は、**Show advanced configuration** を選択します。チャネル名とユーザー名を検索し、これらをリンクするかどうかについて選択することもできます。

5. デフォルトでは、すべてのセレクターに一致するラベルを持つ Firing アラートが receiver に送信されます。receiver に送信する前に、Firing アラートのラベル値を完全に一致させる場合は、次の手順を実行します。

a. フォームの **Routing Labels** セクションに、ルーティングラベルの名前と値を追加します。

b. **Add Label** を選択して、さらにルーティングラベルを追加します。

6. **Create** をクリックしてレシーバーを作成します。

10.7.2. ユーザー定義プロジェクトのアラートルーティングの作成

alert-routing-edit クラスターロールが付与されている管理者以外のユーザーの場合は、ユーザー定義プロジェクトのアラートルーティングを作成または編集できます。

前提条件

- クラスター管理者は、ユーザー定義プロジェクトのモニタリングを有効にしている。
- クラスター管理者が、ユーザー定義プロジェクトのアラートルーティングを有効にしている。
- アラートルーティングを作成する必要のあるプロジェクトの **alert-routing-edit** クラスターロールを持つユーザーとしてログインしている。
- OpenShift CLI (**oc**) がインストールされている。

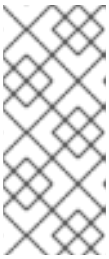
手順

1. アラートルーティングの YAML ファイルを作成します。この手順の例では、**example-app-alert-routing.yaml** という名前のファイルを使用します。

2. **AlertmanagerConfig** YAML 定義をファイルに追加します。以下に例を示します。

```
apiVersion: monitoring.coreos.com/v1beta1
kind: AlertmanagerConfig
metadata:
  name: example-routing
  namespace: ns1
spec:
```

```
  route:
    receiver: default
    groupBy: [job]
  receivers:
  - name: default
    webhookConfigs:
    - url: https://example.org/post
```



注記

ユーザー定義のアラートルールの場合、ユーザー定義のルーティングはリソースが定義される namespace に対してスコープ指定されます。たとえば、namespace **ns1** の **AlertmanagerConfig** オブジェクトで定義されるルーティング設定は、同じ namespace の **PrometheusRules** リソースにのみ適用されます。

3. ファイルを保存します。
4. リソースをクラスターに適用します。

```
$ oc apply -f example-app-alert-routing.yaml
```

この設定は Alertmanager Pod に自動的に適用されます。

10.8. カスタム ALERTMANAGER 設定の適用

Alertmanager のプラットフォームインスタンスの **openshift-monitoring** namespace で **alertmanager-main** シークレットを編集して、デフォルトの Alertmanager 設定を上書きできます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。

手順

CLI で Alertmanager 設定を変更するには、以下を実行します。

1. 現在アクティブな Alertmanager 設定をファイル **alertmanager.yaml** に出力します。

```
$ oc -n openshift-monitoring get secret alertmanager-main --template='{{ index .data
"alertmanager.yaml" }}' | base64 --decode > alertmanager.yaml
```

2. **alertmanager.yaml** で設定を編集します。

```
global:
  resolve_timeout: 5m
route:
  group_wait: 30s (1)
  group_interval: 5m (2)
  repeat_interval: 12h (3)
  receiver: default
  routes:
  - matchers:
    - "alertname=Watchdog"
```

```
    repeat_interval: 2m
    receiver: watchdog
  - matchers:
    - "service=<your_service>" 4
    routes:
    - matchers:
      - <your_matching_rules> 5
      receiver: <receiver> 6
receivers:
- name: default
- name: watchdog
- name: <receiver>
#  <receiver_configuration>
```

1. **group_wait** 値は、Alertmanager がアラートグループの初期通知を送信するまで待機する時間を指定します。この値は、Alertmanager が通知を送信する前に、同じグループの初期アラートを収集するまで待機する時間を制御します。

2. **group_interval** 値は、最初の通知がすでに送信されているアラートグループに追加された新しいアラートに関する通知を Alertmanager が送信するまでの時間を指定します。

3. **repeat_interval** の値は、アラート通知が繰り返される前に経過する必要のある最小時間を指定します。各グループの間隔で通知を繰り返す場合は、**repeat_interval** の値を **group_interval** の値よりも小さく設定します。ただし、たとえば特定の Alertmanager Pod が再起動または再スケジュールされた場合などは、繰り返し通知が遅延する可能性があります。

4. **service** の値は、アラートを発行するサービスを指定します。

5. **<your_matching_rules>** 値は、ターゲットアラートを指定します。

6. **receiver** 値は、アラートに使用するレシーバーを指定します。

注記

matchers キー名を使用して、ノードの照合でアラートが満たす必要のあるマッチャーを指定します。**match** または **match_re** キー名は使用しないでください。どちらも非推奨であり、今後のリリースで削除される予定です。

さらに、禁止ルールを定義する場合は、**target_matchers** キー名を使用してターゲットマッチャーを示し、**source_matchers** キー 名を使用してソースマッチャーを示します。**target_match**、**target_match_re**、**source_match**、または **source_match_re** キー名は使用しないでください。これらは非推奨であり、今後のリリースで削除される予定です。

以下の Alertmanager 設定例は、PagerDuty をアラートレシーバーとして設定します。

```
global:
  resolve_timeout: 5m
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
```

```
  routes:
  - matchers:
    - "alertname=Watchdog"
    repeat_interval: 2m
    receiver: watchdog
  - matchers:
    - "service=example-app"
    routes:
    - matchers:
      - "severity=critical"
      receiver: team-frontend-page*
receivers:
- name: default
- name: watchdog
- name: team-frontend-page
  pagerduty_configs:
  - service_key: "_your-key_"
```

この設定では、**example-app** サービスで実行される重大度が **critical** のアラートは、**team-frontend-page** receiver を使用して送信されます。通常、これらのタイプのアラートは、個別または緊急対策チーム (Critical Response Team) に送信先が設定されます。

3. 新規設定をファイルで適用します。

```
$ oc -n openshift-monitoring create secret generic alertmanager-main --from-file=alertmanager.yaml --dry-run=client -o=yaml |  oc -n openshift-monitoring replace secret --filename=-
```

OpenShift Container Platform Web コンソールから Alertmanager 設定を変更するには、以下を実行します。

1. Web コンソールの **Administration → Cluster Settings → Configuration → Alertmanager → YAML** ページに移動します。
2. YAML 設定ファイルを変更します。
3. **Save** をクリックします。

10.9. ユーザー定義のアラートルーティングの ALERTMANAGER へのカスタム設定の適用

ユーザー定義のアラートルーティング専用の Alertmanager の別のインスタンスを有効にしている場合は、**openshift-user-workload-monitoring** namespace で **alertmanager-user-workload** シークレットを編集して Alertmanager のこのインスタンスの設定を上書きできます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 現在アクティブな Alertmanager 設定をファイル **alertmanager.yaml** に出力します。

```
$ oc -n openshift-user-workload-monitoring get secret alertmanager-user-workload --
template='{{ index .data "alertmanager.yaml" }}' | base64 --decode > alertmanager.yaml
```

2. **alertmanager.yaml** で設定を編集します。

```
route:
  receiver: Default
  group_by:
  - name: Default
  routes:
  - matchers:
    - "service = prometheus-example-monitor" 1
    receiver: <receiver> 2
receivers:
- name: Default
- name: <receiver>
#  <receiver_configuration>
```

1 ルートに一致するアラートを指定します。この例では、**service="prometheus-example-monitor"** ラベルの付いたすべてのアラートを示しています。

2 アラートグループに使用するレシーバーを指定します。

3. 新規設定をファイルで適用します。

```
$ oc -n openshift-user-workload-monitoring create secret generic alertmanager-user-
workload --from-file=alertmanager.yaml --dry-run=client -o=yaml |  oc -n openshift-user-
workload-monitoring replace secret --filename=-
```

関連情報

- PagerDuty についての詳細は、PagerDuty の公式サイト を参照してください。
- **service_key** を取得する方法は、PagerDuty Prometheus Integration Guide を参照してください。
- 各種のアラートレシーバー経由でアラートを設定する方法については、Alertmanager configuration を参照してください。
- ユーザー定義のアラートルーティング用に Alertmanager の専用インスタンスを有効にする方法は、ユーザー定義プロジェクトのアラートルーティングの有効化 を参照してください。

10.10. 次のステップ

- モニタリングダッシュボードの確認

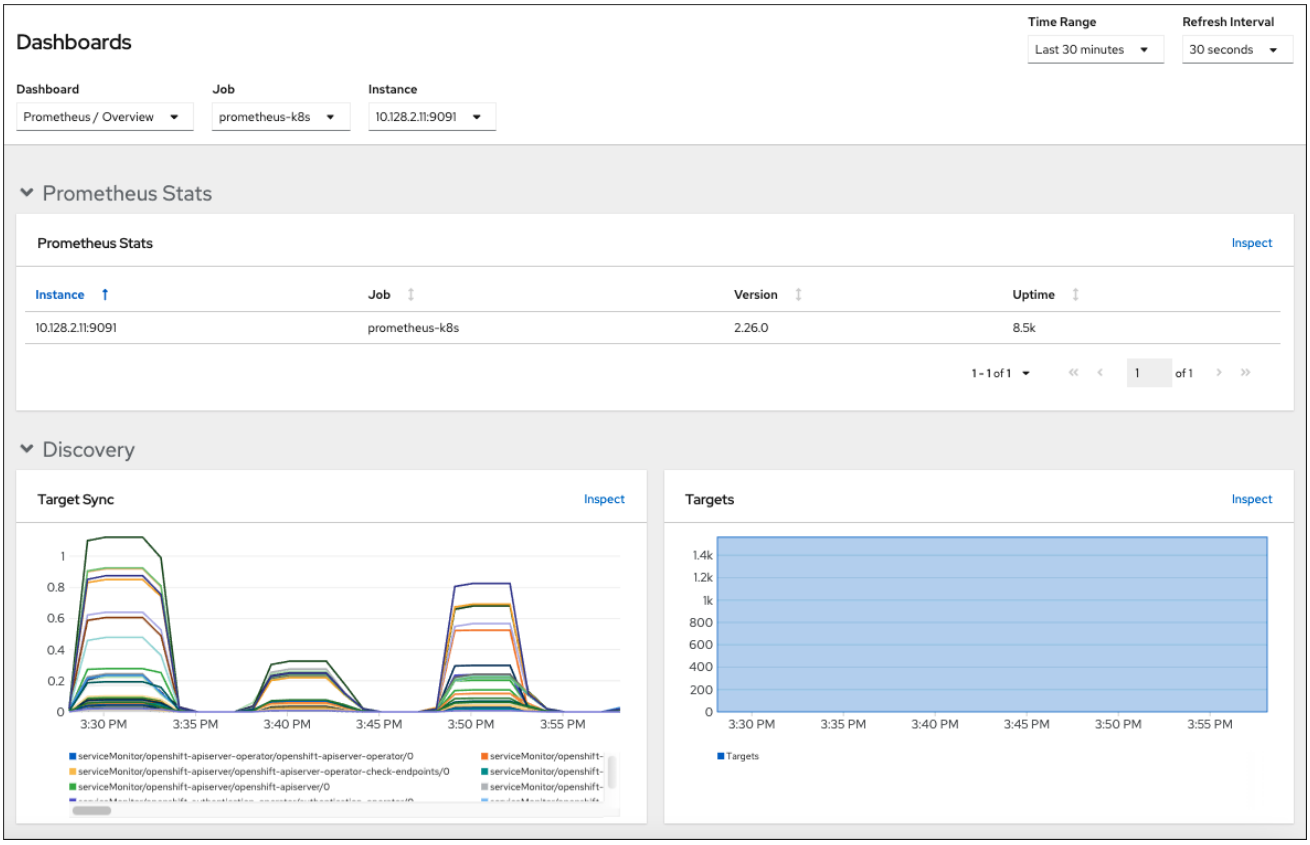
第11章 モニタリングダッシュボードの確認

OpenShift Container Platform 4.15 は、クラスターコンポーネントおよびユーザー定義ワークロードの状態を理解するために役立つ包括的なモニタリングダッシュボードのセットを提供します。

Administrator パースペクティブを使用して、以下を含む OpenShift Container Platform のコアコンポーネントのダッシュボードにアクセスします。

- API パフォーマンス
- etcd
- Kubernetes コンピュートリソース
- Kubernetes ネットワークリソース
- Prometheus
- クラスターおよびノードのパフォーマンスに関連する USE メソッドダッシュボード
- ノードのパフォーマンスメトリクス

図11.1 Administrator パースペクティブのダッシュボードの例

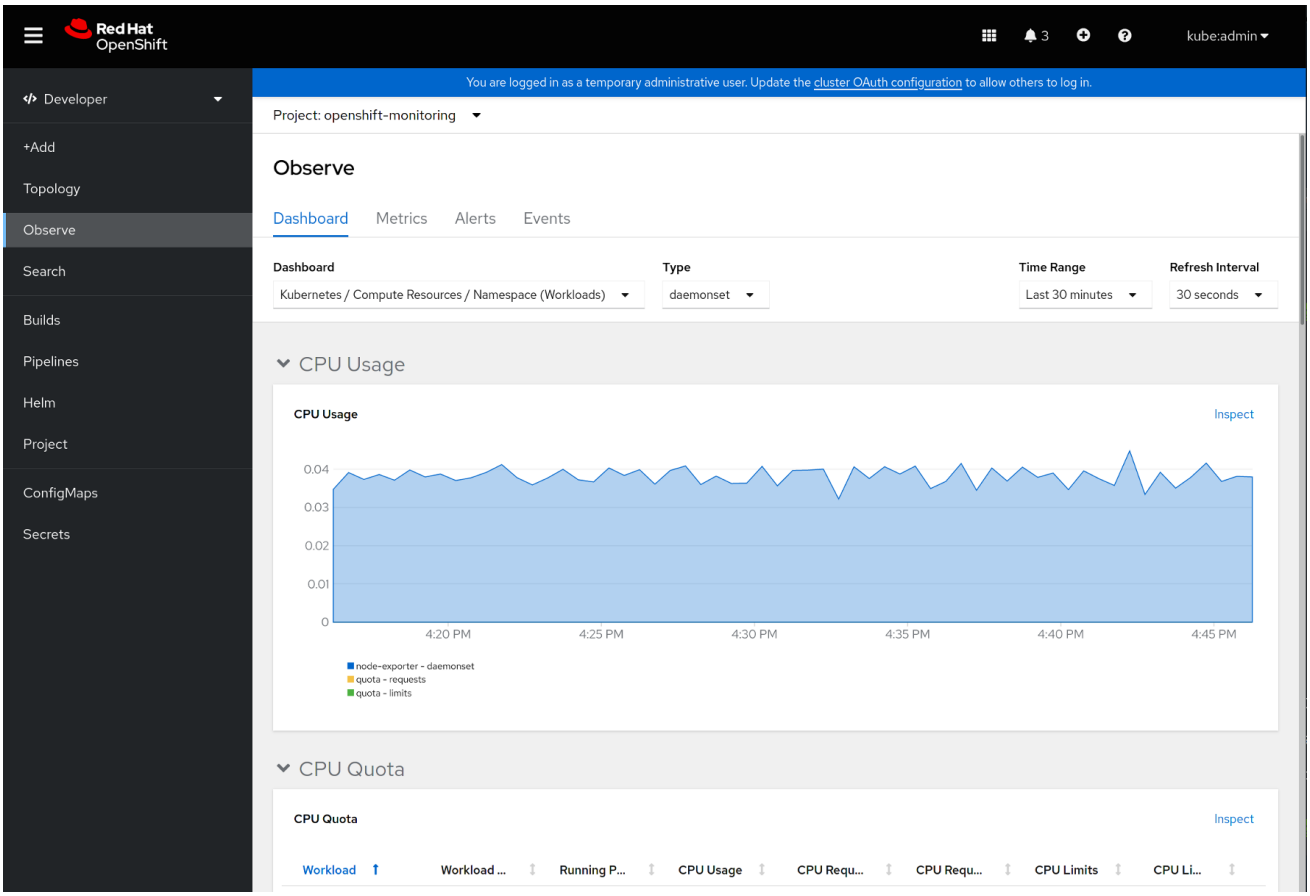


Developer パースペクティブを使用して、選択されたプロジェクトの以下のアプリケーションメトリックを提供する Kubernetes コンピュートリソースダッシュボードにアクセスします。

- CPU usage (CPU の使用率)
- メモリー使用量
- 帯域幅に関する情報

- パケットレート情報

図11.2 Developer パースペクティブのダッシュボードの例



注記

Developer パースペクティブでは、1度に1つのプロジェクトのみのダッシュボードを表示できます。

11.1. クラスター管理者としてのモニタリングダッシュボードの確認

Administrator パースペクティブでは、OpenShift Container Platform クラスターのコアコンポーネントに関連するダッシュボードを表示できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Observe → Dashboards** に移動します。
2. **Dashboard** 一覧でダッシュボードを選択します。**etcd** や **Prometheus** ダッシュボードなどの一部のダッシュボードは、選択時に追加のサブメニューを生成します。
3. 必要に応じて、**Time Range** 一覧でグラフの時間範囲を選択します。
   - 事前定義済みの期間を選択します。

- **時間範囲** リストで **カスタムの時間範囲** を選択して、カスタムの時間範囲を設定します。
    a. **From** および **To** の日付と時間を入力または選択します。
    b. **Save** をクリックして、カスタムの時間範囲を保存します。

4. オプション: **Refresh Interval** を選択します。
5. 特定の項目についての詳細情報を表示するには、ダッシュボードの各グラフにカーソルを合わせます。

11.2. 開発者が行うモニタリングダッシュボードの確認

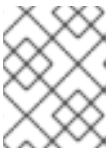
Developer パースペクティブでは、選択されたプロジェクトに関連するダッシュボードを表示できます。ダッシュボード情報を表示するには、プロジェクトをモニターするためのアクセスが必要になります。

前提条件

- 開発者またはユーザーとしてクラスターにアクセスできる。
- ダッシュボードを表示するプロジェクトの表示権限がある。

手順

1. OpenShift Container Platform Web コンソールの Developer パースペクティブで、**Observe → Dashboard** に移動します。
2. **Project:** ドロップダウンリストからプロジェクトを選択します。
3. **Dashboard** ドロップダウンリストからダッシュボードを選択し、フィルターされたメトリックを表示します。



注記

すべてのダッシュボードは、**Kubernetes / Compute Resources / Namespace(Pod)** を除く、選択時に追加のサブメニューを生成します。

4. 必要に応じて、**Time Range** 一覧でグラフの時間範囲を選択します。
    - 事前定義済みの期間を選択します。
    - **時間範囲** リストで **カスタムの時間範囲** を選択して、カスタムの時間範囲を設定します。
        a. **From** および **To** の日付と時間を入力または選択します。
        b. **Save** をクリックして、カスタムの時間範囲を保存します。
5. オプション: **Refresh Interval** を選択します。
6. 特定の項目についての詳細情報を表示するには、ダッシュボードの各グラフにカーソルを合わせます。

関連情報

- Developer パースペクティブを使用したプロジェクトおよびアプリケーションメトリクスのモニタリング

11.3. 次のステップ

- CLI を使用した API のモニタリング

第12章 CLI を使用した API のモニタリング

OpenShift Container Platform 4.15 では、コマンドラインインターフェイス (CLI) から一部のモニタリングコンポーネントの Web サービス API にアクセスできます。

重要

特定の状況では、特にエンドポイントを使用して大量のメトリックデータを取得、送信、またはクエリーする場合、API エンドポイントにアクセスするとクラスターのパフォーマンスとスケーラビリティーが低下する可能性があります。

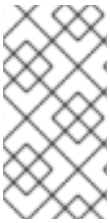
これらの問題を回避するには、以下の推奨事項に従ってください。

- エンドポイントに頻繁にクエリーを実行しないようにします。クエリーを 30 秒ごとに最大 1 つに制限します。
- Prometheus の **/federate** エンドポイントを介してすべてのメトリックデータを取得しようとしないでください。制限された集約されたデータセットを取得する場合にのみクエリーします。たとえば、各要求で 1,000 未満のサンプルを取得すると、パフォーマンスが低下するリスクを最小限に抑えることができます。

12.1. モニタリング WEB サービス API へのアクセスについて

次の監視スタックコンポーネントのコマンドラインから Web サービス API エンドポイントに直接アクセスできます。

- Prometheus
- Alertmanager
- Thanos Ruler
- Thanos Querier



注記

Thanos Ruler および Thanos Querier サービス API にアクセスするには、要求元のアカウントが namespace リソースに対するアクセス許可を **get** している必要があります。これは、アカウントに **cluster-monitoring-view** クラスターロールをバインドして付与することで実行できます。

モニタリングコンポーネントの Web サービス API エンドポイントにアクセスする場合は、以下の制限事項に注意してください。

- Bearer Token 認証のみを使用して API エンドポイントにアクセスできます。
- ルートの **/api** パスのエンドポイントにのみアクセスできます。Web ブラウザーで API エンドポイントにアクセスしようとすると、**Application is not available** エラーが発生します。Web ブラウザーでモニタリング機能にアクセスするには、OpenShift Container Platform Web コンソールを使用して、モニタリングダッシュボードを確認します。

関連情報

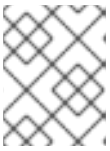
- モニタリングダッシュボードの確認

12.2. 監視 WEB サービス API へのアクセス

次の例は、コアプラットフォームの監視で使用される Alertmanager サービスのサービス API レシーバーをクエリーする方法を示しています。同様の方法を使用して、コアプラットフォーム Prometheus の **prometheus-k8s** サービスと Thanos Ruler の **thanos-ruler** サービスにアクセスできます。

前提条件

- **openshift-monitoring** 名前空間の **monitoring-alertmanager-edit** ロールにバインドされているアカウントにログインしています。
- Alertmanager API ルートを取得する権限を持つアカウントにログインしています。



注記

アカウントに Alertmanager API ルートの取得権限がない場合、クラスター管理者はルートの URL を提供できます。

手順

1. 次のコマンドを実行して認証トークンを抽出します。

```
$ TOKEN=$(oc whoami -t)
```

2. 次のコマンドを実行して、**alertmanager-main** API ルート URL を抽出します。

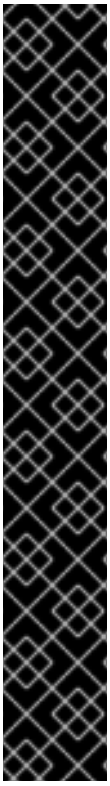
```
$ HOST=$(oc -n openshift-monitoring get route alertmanager-main -ojsonpath={.spec.host})
```

3. 次のコマンドを実行して、サービス API レシーバーに Alertmanager をクエリーします。

```
$ curl -H "Authorization: Bearer $TOKEN" -k "https://$HOST/api/v2/receivers"
```

12.3. PROMETHEUS のフェデレーションエンドポイントを使用したメトリックのクエリー

Prometheus のフェデレーションエンドポイントを使用して、クラスターの外部のネットワークの場所からプラットフォームとユーザー定義のメトリックを収集できます。これを実行するには、OpenShift Container Platform ルートを使用してクラスターの Prometheus /**federate** エンドポイントにアクセスします。



重要

メトリックデータの取得の遅延は、フェデレーションを使用すると発生します。この遅延は、収集されたメトリクスの精度とタイムラインに影響を与えます。

フェデレーションエンドポイントを使用すると、特にフェデレーションエンドポイントを使用して大量のメトリックデータを取得する場合に、クラスターのパフォーマンスおよびスケーラビリティーを低下させることもできます。これらの問題を回避するには、以下の推奨事項に従ってください。

- Prometheus のフェデレーションエンドポイントを介してすべてのメトリックデータを取得しようとしないでください。制限された集約されたデータセットを取得する場合にのみクエリーします。たとえば、各要求で 1,000 未満のサンプルを取得すると、パフォーマンスが低下するリスクを最小限に抑えることができます。
- Prometheus のフェデレーションエンドポイントに対して頻繁にクエリーすることは避けてください。クエリーを 30 秒ごとに最大 1 つに制限します。

クラスター外に大量のデータを転送する必要がある場合は、代わりにリモート書き込みを使用します。詳細は、**リモート書き込みストレージの設定**セクションを参照してください。

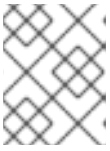
前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-monitoring-view** クラスターロールを持つユーザーとしてクラスターにアクセスできるか、**namespace** リソースの **get** 権限を持つベアラートークンを取得している。

注記

Prometheus フェデレーションエンドポイントへのアクセスには、ベアラートークン認証のみを使用できます。

- Prometheus フェデレーションルートを取得する権限を持つアカウントにログインしている。



注記

アカウントに Prometheus フェデレーションルートを取得する権限がない場合、クラスター管理者はルートの URL を提供できます。

手順

1. 次のコマンドを実行してベアラートークンを取得します。

```
$ TOKEN=$(oc whoami -t)
```

2. 次のコマンドを実行して、Prometheus フェデレーションルート URL を取得します。

```
$ HOST=$(oc -n openshift-monitoring get route prometheus-k8s-federate -ojsonpath=
{.spec.host})
```

3. **/federate** ルートからメトリックをクエリーします。次のコマンド例は、**up** メトリクスをクエリーします。

```
$ curl -G -k -H "Authorization: Bearer $TOKEN" https://$HOST/federate --data-urlencode
'match[]=up'
```

出力例

```
# TYPE up untyped
up{apiserver="kube-
apiserver",endpoint="https",instance="10.0.143.148:6443",job="apiserver",namespace="default
",service="kubernetes",prometheus="openshift-
monitoring/k8s",prometheus_replica="prometheus-k8s-0"} 1 1657035322214
up{apiserver="kube-
apiserver",endpoint="https",instance="10.0.148.166:6443",job="apiserver",namespace="default
",service="kubernetes",prometheus="openshift-
monitoring/k8s",prometheus_replica="prometheus-k8s-0"} 1 1657035338597
up{apiserver="kube-
apiserver",endpoint="https",instance="10.0.173.16:6443",job="apiserver",namespace="default",
service="kubernetes",prometheus="openshift-
monitoring/k8s",prometheus_replica="prometheus-k8s-0"} 1 1657035343834
...
```

12.4. カスタムアプリケーションについてのクラスター外からのメトリックへのアクセス

ユーザー定義プロジェクトを使用して独自のサービスを監視する場合は、クラスターの外部から Prometheus メトリクスをクエリーできます。このデータには、**thanos-querier** ルートを使用してクラスターの外部からアクセスします。

このアクセスは、認証に Bearer Token を使用することのみをサポートします。

前提条件

- "ユーザー定義プロジェクトのモニタリングの有効化" の手順に従い、独自のサービスをデプロイしている。
- Thanos Querier API へのアクセス権限を持つ **cluster-monitoring-view** クラスターロールでアカウントにログインしている。
- Thanos Querier API ルートの取得権限を持つアカウントにログインしています。

注記

アカウントに Thanos Querier API ルートの取得権限がない場合、クラスター管理者はルートの URL を提供できます。

手順

1. 次のコマンドを実行して、Prometheus に接続するための認証トークンを展開します。

```
$ TOKEN=$(oc whoami -t)
```

2. 次のコマンドを実行して、**thanos-querier** API ルート URL を展開します。

```
$ HOST=$(oc -n openshift-monitoring get route thanos-querier -ojsonpath={.spec.host})
```

3. 次のコマンドを使用して、サービスが実行されている namespace に namespace を設定します。

```
$ NAMESPACE=ns1
```

4. 次のコマンドを実行して、コマンドラインで独自のサービスのメトリクスに対してクエリーを実行します。

```
$ curl -H "Authorization: Bearer $TOKEN" -k "https://$HOST/api/v1/query?" --data-urlencode
"query=up{namespace='$NAMESPACE'}"
```

出力には、Prometheus がスクレイピングしている各アプリケーション Pod のステータスが表示されます。

出力例

```
{"status":"success","data":{"resultType":"vector","result":[{"metric":
{"__name__":"up","endpoint":"web","instance":"10.129.0.46:8080","job":"prometheus-
example-app","namespace":"ns1","pod":"prometheus-example-app-68d47c4fb6-
jztp2","service":"prometheus-example-app"},"value":[1591881154.748,"1"]}]}}
```

12.5. 関連情報

- ユーザー定義プロジェクトのモニタリングの有効化
- リモート書き込みストレージの設定
- メトリクスの管理
- アラートの管理

第13章 モニタリング関連の問題のトラブルシューティング

コアプラットフォームおよびユーザー定義プロジェクトのモニタリングに関する一般的な問題のトラブルシューティング手順を参照してください。

13.1. ユーザー定義のプロジェクトメトリクスが使用できない理由の調査

ServiceMonitor リソースを使用すると、ユーザー定義プロジェクトでサービスによって公開されるメトリックの使用方法を判別できます。**ServiceMonitor** リソースを作成している場合で、メトリック UI に対応するメトリックが表示されない場合は、この手順で説明されるステップを実行します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- ユーザー定義のワークロードのモニタリングを有効にし、設定している。
- **user-workload-monitoring-config ConfigMap** オブジェクトを作成している。
- **ServiceMonitor** リソースを作成している。

手順

1. サービスおよび **ServiceMonitor** リソース設定で、**対応するラベルの一致を確認** します。
   a. サービスに定義されたラベルを取得します。以下の例では、**ns1** プロジェクトの **prometheus-example-app** サービスをクエリーします。

```
$ oc -n ns1 get service prometheus-example-app -o yaml
```

出力例

```
  labels:
    app: prometheus-example-app
```

   b. **ServiceMonitor** リソース設定の **matchLabels** 定義が、直前の手順のラベルの出力と一致することを確認します。次の例では、**ns1** プロジェクトの **prometheus-example-monitor** サービスモニターをクエリーします。

```
$ oc -n ns1 get servicemonitor prometheus-example-monitor -o yaml
```

出力例

```
apiVersion: v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - interval: 30s
```

```
    port: web
    scheme: http
  selector:
    matchLabels:
      app: prometheus-example-app
```

注記

プロジェクトの表示権限を持つ開発者として、サービスおよび **ServiceMonitor** リソースラベルを確認できます。

2. **openshift-user-workload-monitoring** プロジェクトの **Prometheus Operator のログを検査**します。

   a. **openshift-user-workload-monitoring** プロジェクトの Pod をリスト表示します。

   ```
   $ oc -n openshift-user-workload-monitoring get pods
   ```

   出力例

   ```
   NAME                                  READY   STATUS    RESTARTS   AGE
   prometheus-operator-776fcbbd56-2nbfm   2/2     Running   0          132m
   prometheus-user-workload-0             5/5     Running   1          132m
   prometheus-user-workload-1             5/5     Running   1          132m
   thanos-ruler-user-workload-0           3/3     Running   0          132m
   thanos-ruler-user-workload-1           3/3     Running   0          132m
   ```

   b. **prometheus-operator** Pod の **prometheus-operator** コンテナーからログを取得します。以下の例では、Pod は **prometheus-operator-776fcbbd56-2nbfm** になります。

   ```
   $ oc -n openshift-user-workload-monitoring logs prometheus-operator-776fcbbd56-
   2nbfm -c prometheus-operator
   ```

   サービスモニターに問題がある場合、ログには以下のようなエラーが含まれる可能性があります。

   ```
   level=warn ts=2020-08-10T11:48:20.906739623Z caller=operator.go:1829
   component=prometheusoperator msg="skipping servicemonitor" error="it accesses file
   system via bearer token file which Prometheus specification prohibits"
   servicemonitor=eagle/eagle namespace=openshift-user-workload-monitoring
   prometheus=user-workload
   ```

3. OpenShift Container Platform Web コンソール UI の **Metrics targets** ページで、**エンドポイントのターゲットステータスを確認** します。

   a. OpenShift Container Platform の Web コンソールにログインし、**管理者** パースペクティブの **Observe → Targets** に移動します。

   b. リストでメトリックのエンドポイントを探し、**Status** 列でターゲットのステータスを確認します。

   c. **Status** が **Down** の場合、エンドポイントの URL をクリックすると、そのメトリックターゲットの **Target Details** ページで詳細情報を見ることができます。

4. **openshift-user-workload-monitoring** プロジェクトで Prometheus Operator **のデバッグレベルのロギングを設定** します。

   a. **openshift-user-workload-monitoring** プロジェクトで **user-workload-monitoring-config ConfigMap** オブジェクトを編集します。

   ```
   $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
   ```

   b. **prometheusOperator** の **logLevel: debug** を **data/config.yaml** に追加し、ログレベルを **debug** に設定します。

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: user-workload-monitoring-config
     namespace: openshift-user-workload-monitoring
   data:
     config.yaml: |
       prometheusOperator:
         logLevel: debug
   # ...
   ```

   c. 変更を適用するためにファイルを保存します。

   注記

   openshift-user-workload-monitoring プロジェクトの **prometheus-operator** は、ログレベルの変更時に自動的に再起動します。

   d. **debug** ログレベルが **openshift-user-workload-monitoring** プロジェクトの **prometheus-operator** デプロイメントに適用されていることを確認します。

   ```
   $ oc -n openshift-user-workload-monitoring get deploy prometheus-operator -o yaml |
   grep "log-level"
   ```

   出力例

   ```
           - --log-level=debug
   ```

   debug レベルのロギングにより、Prometheus Operator によって行われるすべての呼び出しが表示されます。

   e. **prometheus-operator** Pod が実行されていることを確認します。

   ```
   $ oc -n openshift-user-workload-monitoring get pods
   ```

   注記

   認識されない Prometheus Operator の **loglevel** 値が config map に含まれる場合、**prometheus-operator** Pod が正常に再起動されない可能性があります。

f. デバッグログを確認し、Prometheus Operator が **ServiceMonitor** リソースを使用しているかどうかを確認します。ログで他の関連するエラーの有無を確認します。

関連情報

- ユーザー定義のワークロードモニタリング config map の作成
- **ServiceMonitor** または **PodMonitor** の作成方法についての詳細は、サービスのモニター法の指定 を参照してください。
- メトリックターゲットに関する詳細情報の取得 を参照してください。

13.2. PROMETHEUS が大量のディスク領域を消費している理由の特定

開発者は、キーと値のペアの形式でメトリックの属性を定義するためにラベルを作成できます。使用できる可能性のあるキーと値のペアの数は、属性について使用できる可能性のある値の数に対応します。数が無制限の値を持つ属性は、バインドされていない属性と呼ばれます。たとえば、**customer_id** 属性は、使用できる値が無限にあるため、バインドされていない属性になります。

割り当てられるキーと値のペアにはすべて、一意の時系列があります。ラベルに多数のバインドされていない値を使用すると、作成される時系列の数が指数関数的に増加する可能性があります。これは Prometheus のパフォーマンスに影響する可能性があり、多くのディスク領域を消費する可能性があります。

Prometheus が多くのディスクを消費する場合、以下の手段を使用できます。

- どのラベルが最も多くの時系列データを作成しているか詳しく知るには **Prometheus HTTP API を使用して時系列データベース (TSDB) のステータスを確認** します。これを実行するには、クラスター管理者権限が必要です。
- 収集される **収集サンプルの数を確認** します。
- ユーザー定義メトリックに割り当てられるバインドされていない属性の数を減らすことで、**作成される一意の時系列の数を減らします**。

注記

使用可能な値の制限されたセットにバインドされる属性を使用すると、可能なキーと値のペアの組み合わせの数が減ります。

- ユーザー定義プロジェクト間で **収集可能なサンプル数の数に制限を適用します**。これには、クラスター管理者の権限が必要です。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **Administrator** パースペクティブで、**Observe → Metrics** に移動します。

2. **Expression** フィールドに、Prometheus Query Language (PromQL)クエリーを入力します。次のクエリー例は、ディスク領域の消費量の増加につながる可能性のある高カーディナリティメトリクスを識別するのに役立ちます。

   - 次のクエリーを実行すると、スクレイプサンプルの数が最も多い ジョブを 10 個特定できます。

     ```
     topk(10, max by(namespace, job) (topk by(namespace, job) (1,
     scrape_samples_post_metric_relabeling)))
     ```

   - 次のクエリーを実行すると、過去 1 時間に最も多くの時系列データを作成したジョブを 10 個特定して、時系列のチャーンを正確に特定できます。

     ```
     topk(10, sum by(namespace, job) (sum_over_time(scrape_series_added[1h])))
     ```

3. 想定よりもサンプルのスクレイプ数が多いメトリクスに割り当てられたラベルで、値が割り当てられていないものの数を確認します。

   - **メトリックがユーザー定義のプロジェクトに関連する場合**、ワークロードに割り当てられたメトリックのキーと値のペアを確認します。これらのライブラリーは、アプリケーションレベルで Prometheus クライアントライブラリーを使用して実装されます。ラベルで参照されるバインドされていない属性の数の制限を試行します。

   - **メトリクスが OpenShift Container Platform のコアプロジェクトに関連する場合**、Red Hat サポートケースを Red Hat カスタマーポータル で作成してください。

4. クラスター管理者としてログインしてから、次の手順に従い Prometheus HTTP API を使用して TSDB ステータスを確認します。

   a. 次のコマンドを実行して、Prometheus API ルート URL を取得します。

      ```
      $ HOST=$(oc -n openshift-monitoring get route prometheus-k8s -ojsonpath={.spec.host})
      ```

   b. 次のコマンドを実行して認証トークンを抽出します。

      ```
      $ TOKEN=$(oc whoami -t)
      ```

   c. 次のコマンドを実行して、Prometheus の TSDB ステータスをクエリーします。

      ```
      $ curl -H "Authorization: Bearer $TOKEN" -k "https://$HOST/api/v1/status/tsdb"
      ```

      出力例

      ```
      "status": "success","data":{"headStats":{"numSeries":507473,
      "numLabelPairs":19832,"chunkCount":946298,"minTime":1712253600010,
      "maxTime":1712257935346},"seriesCountByMetricName":
      [{"name":"etcd_request_duration_seconds_bucket","value":51840},
      {"name":"apiserver_request_sli_duration_seconds_bucket","value":47718},
      ...
      ```

関連情報

- CLI を使用した API のモニタリング

- ユーザー定義プロジェクトの収集サンプル制限の設定
- サポートケースの送信

第14章 CLUSTER MONITORING OPERATOR の CONFIGMAP 参照

14.1. CLUSTER MONITORING OPERATOR 設定リファレンス

OpenShift Container Platform クラスターモニタリングの一部は設定可能です。API には、さまざまな Config Map で定義されるパラメーターを設定してアクセスできます。

- モニタリングコンポーネントを設定するには、**openshift-monitoring** namespace で **cluster-monitoring-config** という名前の **ConfigMap** オブジェクトを編集します。このような設定は ClusterMonitoringConfiguration によって定義されます。
- ユーザー定義プロジェクトを監視するモニタリングコンポーネントを設定するには、**openshift-user-workload-monitoring** namespace で **user-workload-monitoring-config** という名前の **ConfigMap** オブジェクトを編集します。これらの設定は UserWorkloadConfiguration で定義されます。

設定ファイルは、常に config map データの **config.yaml** キーで定義されます。

重要

- モニタリングスタックのすべての設定パラメーターが公開されるわけではありません。このリファレンスにリストされているパラメーターとフィールドのみが設定でサポートされます。サポートされる設定の詳細は、メンテナンスおよび監視のサポート を参照してください。
- クラスターモニタリングの設定はオプションです。
- 設定が存在しないか、空の場合には、デフォルト値が使用されます。
- 設定が無効な場合、Cluster Monitoring Operator はリソースの調整を停止し、Operator のステータス条件で **Degraded=True** を報告します。

14.2. ADDITIONALALERTMANAGERCONFIG

14.2.1. 説明

AdditionalAlertmanagerConfig リソースは、コンポーネントが追加の Alertmanager インスタンスと通信する方法の設定を定義します。

14.2.2. 必須

- **apiVersion**

出現場所: PrometheusK8sConfig、PrometheusRestrictedConfig、ThanosRulerConfig

プロパティー	型	説明

プロパティー	型	説明
apiVersion	string	Alertmanager の API バージョンを定義します。使用できる値は **v1** または **v2** です。デフォルトは **v2** です。
bearerToken	*v1.SecretKeySelector	Alertmanager への認証時に使用するベアラートークンを含むシークレットキー参照を定義します。
pathPrefix	string	プッシュエンドポイントパスの前に追加するパス接頭辞を定義します。
scheme	string	Alertmanager インスタンスとの通信時に使用する URL スキームを定義します。使用できる値は **http** または **https** です。デフォルト値は **http** です。
staticConfigs	[]string	**<hosts>:<port>** の形式で静的に設定された Alertmanager エンドポイントの一覧。
timeout	*文字列	アラートの送信時に使用されるタイムアウト値を定義します。
tlsConfig	TLSConfig	Alertmanager 接続に使用する TLS 設定を定義します。

14.3. ALERTMANAGERMAINCONFIG

14.3.1. 説明

AlertmanagerMainConfig リソースは、**openshift-monitoring** namespace で Alertmanager コンポーネントの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
enabled	*bool	**openshift-monitoring** namespace のメイン Alertmanager インスタンスを有効または無効にするブール値フラグ。デフォルト値は **true** です。

プロパティー	型	説明
enableUserAlertmanagerConfig	bool	**AlertmanagerConfig** ルックアップのユーザー定義の namespace の選択を有効または無効にするブール値フラグ。この設定は、Alertmanager のユーザーワークロードモニタリングインスタンスが有効になっていない場合にのみ適用されます。デフォルト値は **false** です。
logLevel	string	Alertmanager のログレベル設定を定義します。使用できる値は、**error**、**warn**、**info**、**debug** です。デフォルト値は **info** です。
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	Alertmanager コンテナーのリソース要求および制限を定義します。
secrets	[]string	Alertmanager にマウントされるシークレットの一覧を定義します。シークレットは、Alertmanager オブジェクトと同じ namespace 内になければなりません。これらは **secret-<secret-name>** という名前のボリュームとして追加され、Alertmanager Pod の **alertmanager** コンテナーで **/etc/alertmanager/secrets/<secret-name>** にマウントされます。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Alertmanager の永続ストレージを定義します。この設定を使用して、ストレージクラス、ボリュームサイズ、名前などの永続ボリューム要求を設定します。

14.4. ALERTMANAGERUSERWORKLOADCONFIG

14.4.1. 説明

AlertmanagerUserWorkloadConfig リソースは、ユーザー定義プロジェクトに使用される Alertmanager インスタンスの設定を定義します。

表示場所: UserWorkloadConfiguration

プロパティー	型	説明
enabled	bool	**openshift-user-workload-monitoring** namespace のユーザー定義アラートの Alertmanager の専用インスタンスを有効または無効にするブール値フラグ。デフォルト値は **false** です。
enableAlertmanagerConfig	bool	**AlertmanagerConfig** ルックアップで選択されるユーザー定義の namespace を有効または無効にするブール値フラグ。デフォルト値は **false** です。
logLevel	string	ユーザーワークロードモニタリング用の Alertmanager のログレベル設定を定義します。使用できる値は、**error**、**warn**、**info**、および **debug** です。デフォルト値は **info** です。
resources	*v1.ResourceRequirements	Alertmanager コンテナーのリソース要求および制限を定義します。
secrets	[]string	Alertmanager にマウントされるシークレットの一覧を定義します。シークレットは、Alertmanager オブジェクトと同じ namespace 内に配置する必要があります。これらは **secret-<secret-name>** という名前のボリュームとして追加され、Alertmanager Pod の **alertmanager** コンテナーで **/etc/alertmanager/secrets/<secret-name>** にマウントされます。
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。

プロパティー	型	説明
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Alertmanager の永続ストレージを定義します。この設定を使用して、ストレージクラス、ボリュームサイズ、名前などの永続ボリューム要求を設定します。

14.5. CLUSTERMONITORINGCONFIGURATION

14.5.1. 説明

ClusterMonitoringConfiguration リソースは、**openshift-monitoring** namespace の **cluster-monitoring-config** ConfigMap を使用してデフォルトのプラットフォームモニタリングスタックをカスタマイズする設定を定義します。

プロパティー	型	説明
alertmanagerMain	*AlertmanagerMainConfig	**AlertmanagerMainConfig** は、**openshift-monitoring** namespace で Alertmanager コンポーネントの設定を定義します。
enableUserWorkload	*bool	**UserWorkloadEnabled** は、ユーザー定義プロジェクトのモニタリングを有効にするブール値フラグです。
k8sPrometheusAdapter	*K8sPrometheusAdapter	**K8sPrometheusAdapter** は、Prometheus Adapter コンポーネントの設定を定義します。
kubeStateMetrics	*KubeStateMetricsConfig	**KubeStateMetricsConfig** は、**kube-state-metrics** エージェントの設定を定義します。
metricsServer	*MetricsServerConfig	**MetricsServer** は、Metrics Server コンポーネントの設定を定義します。
prometheusK8s	*PrometheusK8sConfig	**PrometheusK8sConfig** は、Prometheus コンポーネントの設定を定義します。

プロパティー	型	説明
prometheusOperator	*PrometheusOperatorConfig	**PrometheusOperatorConfig** は、Prometheus Operator コンポーネントの設定を定義します。
prometheusOperatorAdmissionWebhook	*PrometheusOperatorAdmissionWebhookConfig	**PrometheusOperatorAdmissionWebhookConfig** は、Prometheus Operator のアドミッション Webhook コンポーネントの設定を定義します。
openshiftStateMetrics	*OpenShiftStateMetricsConfig	**OpenShiftMetricsConfig** は、**openshift-state-metrics** エージェントの設定を定義します。
telemeterClient	*TelemeterClientConfig	**TelemeterClientConfig** は、Telemeter Client コンポーネントの設定を定義します。
thanosQuerier	*ThanosQuerierConfig	**ThanosQuerierConfig** は、Thanos Querier コンポーネントの設定を定義します。
nodeExporter	NodeExporterConfig	**NodeExporterConfig** は、**node-exporter** エージェントの設定を定義します。
monitoringPlugin	*MonitoringPluginConfig	**MonitoringPluginConfig** は、モニタリング **console-plugin** コンポーネントの設定を定義します。

14.6. DEDICATEDSERVICEMONITORS

14.6.1. 説明

重要

この設定は非推奨であり、今後の OpenShift Container Platform バージョンで削除される予定です。この設定は、現在のバージョンにまだ存在しますが、効果はありません。

DedicatedServiceMonitors リソースを使用して、Prometheus アダプターの専用のサービスモニターを設定できます。

表示場所: K8sPrometheusAdapter

プロパティー	型	説明
enabled	bool	**enabled** が **true** に設定されている場合に、Cluster Monitoring Operator (CMO) は kubelet の **/metrics/resource** エンドポイントを公開する専用の Service Monitor をデプロイします。Service Monitor は **honorTimestamps: true** を設定し、Prometheus アダプターの Pod リソースクエリーに関連するメトリクスのみを保持します。さらに、Prometheus アダプターはこれらの専用メトリクスを使用するように設定されます。つまり、この機能は、**oc adm top pod** コマンドまたは Horizontal Pod Autoscaler が使用する Prometheus Adapter ベースの CPU 使用率の測定における一貫性を向上します。

14.7. K8SPROMETHEUSADAPTER

14.7.1. 説明

K8sPrometheusAdapter リソースは、Prometheus Adapter コンポーネントの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
audit	*Audit	Prometheus アダプターインスタンスによって使用される監査設定を定義します。使用できる値は **metadata**、**request**、**requestresponse**、および **none** です。デフォルト値は **metadata** です。
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	**PrometheusAdapter** コンテナーのリソース要求と制限を定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。

プロパティー	型	説明
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。
dedicatedServiceMonitors	*DedicatedServiceMonitors	専用のサービスモニターを定義します。

14.8. KUBESTATEMETRICSCONFIG

14.8.1. 説明

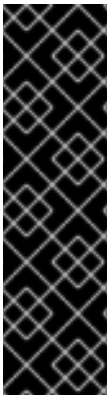
KubeStateMetricsConfig リソースは、**kube-state-metrics** エージェントの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	**KubeStateMetrics** コンテナーのリソースリクエストと制限を定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。

14.9. METRICSSERVERCONFIG

14.9.1. 説明



重要

Metrics Server はテクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

MetricsServerConfig リソースは、Metrics Server コンポーネントの設定を定義します。この設定は、**TechPreviewNoUpgrade** フィーチャーゲートが有効な場合にのみ適用されることに注意してください。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。
resources	*v1.ResourceRequirements	Metrics Server コンテナーのリソース要求および制限を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。

14.10. PROMETHEUSOPERATORADMISSIONWEBHOOKCONFIG

14.10.1. 説明

PrometheusOperatorAdmissionWebhookConfig リソースは、Prometheus Operator のアドミッション Webhook ワークロードの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
resources	*v1.ResourceRequirements	**prometheus-operator-admission-webhook** コンテナーのリソースリクエストと制限を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。

14.11. MONITORINGPLUGINCONFIG

14.11.1. 説明

MonitoringPluginConfig リソースは、**openshift-monitoring** namespace の Web コンソールプラグインコンポーネントの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	**console-plugin** コンテナーのリソースリクエストと制限を定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。

14.12. NODEEXPORTERCOLLECTORBUDDYINFOCONFIG

14.12.1. 説明

NodeExporterCollectorBuddyInfoConfig リソースは、**node-exporter** エージェントの **buddyinfo** コレクターのオン/オフスイッチとして機能します。デフォルトでは、**buddyinfo** コレクターは無効になっています。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**buddyinfo** コレクターを有効または無効にするブール値フラグ。

14.13. NODEEXPORTERCOLLECTORCONFIG

14.13.1. 説明

NodeExporterCollectorConfig リソースは、**node-exporter** エージェントの個別コレクターの設定を定義します。

表示場所: NodeExporterConfig

プロパティー	型	説明
cpufreq	NodeExporterCollectorCpufreqConfig	CPU 周波数の統計情報を収集する **cpufreq** コレクターの設定を定義します。デフォルトでは無効になっています。

プロパティー	型	説明
tcpstat	NodeExporterCollectorTcpStatConfig	TCP 接続の統計情報を収集する **tcpstat** コレクターの設定を定義します。デフォルトでは無効になっています。
netdev	NodeExporterCollectorNetDevConfig	ネットワークデバイスの統計情報を収集する **netdev** コレクターの設定を定義します。デフォルトでは有効です。
netclass	NodeExporterCollectorNetClassConfig	ネットワークデバイスに関する情報を収集する **netclass** コレクターの設定を定義します。デフォルトでは有効です。
buddyinfo	NodeExporterCollectorBuddyInfoConfig	**node_buddyinfo_blocks** メトリックからメモリー断片化に関する統計情報を収集する **buddyinfo** コレクターの設定を定義します。このメトリックは、**/proc/buddyinfo** からデータを収集します。デフォルトでは無効になっています。
mountstats	NodeExporterCollectorMountStatsConfig	NFS ボリューム I/O アクティビティーに関する統計を収集する **mountstats** コレクターの設定を定義します。デフォルトでは無効になっています。
ksmd	NodeExporterCollectorKSMDConfig	カーネルの同一ページ結合デーモンから統計を収集する **ksmd** コレクターの設定を定義します。デフォルトでは無効になっています。
processes	NodeExporterCollectorProcessesConfig	システム内で実行しているプロセスとスレッドから統計を収集する **processes** コレクターの設定を定義します。デフォルトでは無効になっています。
systemd	NodeExporterCollectorSystemdConfig	systemd デーモンとそのマネージドサービスに関する統計を収集する **systemd** コレクターの設定を定義します。デフォルトでは無効になっています。

14.14. NODEEXPORTERCOLLECTORCPUFREQCONFIG

14.14.1. 説明

NodeExporterCollectorCpufreqConfig リソースを使用して、**node-exporter** エージェントの **cpufreq** コレクターを有効または無効にします。デフォルトでは、**cpufreq** コレクターは無効になっています。特定の状況下で **cpufreq** コレクターを有効にすると、多数のコアを持つマシンの CPU 使用率が増加します。マシンに多数のコアがある場合にこのコレクターを有効にする際は、CPU の過剰使用がないかシステムを監視してください。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**cpufreq** コレクターを有効または無効にするブール値フラグ。

14.15. NODEEXPORTERCOLLECTORKSMDCONFIG

14.15.1. 説明

NodeExporterCollectorKSMDConfig リソースを使用して、**node-exporter** エージェントの **ksmd** コレクターを有効または無効にします。デフォルトでは、**ksmd** コレクターは無効になっています。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**ksmd** コレクターを有効または無効にするブールフラグ。

14.16. NODEEXPORTERCOLLECTORMOUNTSTATSCONFIG

14.16.1. 説明

NodeExporterCollectorMountStatsConfig リソースを使用して、**node-exporter** エージェントの **mountstats** コレクターを有効または無効にします。デフォルトでは、**mountstats** コレクターは無効になっています。コレクターを有効にすると、**node_mountstats_nfs_read_bytes_total**、**node_mountstats_nfs_write_bytes_total**、**node_mountstats_nfs_operations_requests_total** のメトリクスが使用可能になります。これらのメトリクスはカーディナリティが高くなる可能性があることに注意してください。このコレクターを有効にした場合は、**prometheus-k8s** Pod のメモリー使用量の増加を注意深く監視してください。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**mountstats** コレクターを有効または無効にするブールフラグ。

14.17. NODEEXPORTERCOLLECTORNETCLASSCONFIG

14.17.1. 説明

NodeExporterCollectorNetClassConfig リソースを使用して、**node-exporter** エージェントの **netclass** コレクターを有効または無効にします。デフォルトでは、**netclass** コレクターが有効になっています。無効にすると、次のメトリックが利用できなくなります (**node_network_info**、**node_network_address_assign_type**、**node_network_carrier**、**node_network_carrier_changes_total**、**node_network_carrier_up_changes_total**、**node_network_carrier_down_changes_total**、**node_network_device_id**、**node_network_dormant**、**node_network_flags**、**node_network_iface_id**、**node_network_iface_link**、**node_network_iface_link_mode**、**node_network_mtu_bytes**、**node_network_name_assign_type**、**node_network_net_dev_group**、**node_network_speed_bytes**、**node_network_transmit_queue_length**、および **node_network_protocol_type**)。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**netclass** コレクターを有効または無効にするブール値フラグ。
useNetlink	bool	**netclass** コレクターの **netlink** 実装をアクティブにするブール値フラグ。デフォルト値は **true** で、**netlink** モードがアクティブになります。この実装により、**netclass** コレクターのパフォーマンスが向上します。

14.18. NODEEXPORTERCOLLECTORNETDEVCONFIG

14.18.1. 説明

NodeExporterCollectorNetDevConfig リソースを使用して、**node-exporter** エージェントの **netdev** コレクターを有効または無効にします。デフォルトでは、**netdev** コレクターが有効になっています。無効にすると、次のメトリクスが利用できなくなります (**node_network_receive_bytes_total**、**node_network_receive_compressed_total**、**node_network_receive_drop_total**、**node_network_receive_errs_total**、**node_network_receive_fifo_total**、**node_network_receive_frame_total**、**node_network_receive_multicast_total**、**node_network_receive_nohandler_total**、**node_network_receive_packets_total**、**node_network_transmit_bytes_total**、**node_network_transmit_carrier_total**、**node_network_transmit_colls_total**、**node_network_transmit_compressed_total**、**node_network_transmit_drop_total**、**node_network_transmit_errs_total**、**node_network_transmit_fifo_total**、および **node_network_transmit_packets_total**)。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**netdev** コレクターを有効または無効にするブール値フラグ。

14.19. NODEEXPORTERCOLLECTORPROCESSESCONFIG

14.19.1. 説明

NodeExporterCollectorProcessesConfig リソースを使用して、**node-exporter** エージェントの **processes** コレクターを有効または無効にします。コレクターが有効な場合は、次のメトリクスが使用可能になります (**node_processes_max_processes**、**node_processes_pids**、**node_processes_state**、**node_processes_threads**、**node_processes_threads_state**)。メトリック **node_processes_state** と **node_processes_threads_state** には、プロセスとスレッドの状態に応じて、それぞれ最大 5 つのシリーズを含めることができます。プロセスまたはスレッドの可能な状態は、**D** (UNINTERRUPTABLE_SLEEP)、**R** (RUNNING & RUNNABLE)、**S** (INTERRUPTABLE_SLEEP)、**T** (STOPPED)、または **Z** (ZOMBIE) です。デフォルトでは、**processes** コレクターは無効になっています。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**processes** コレクターを有効または無効にするブールフラグ。

14.20. NODEEXPORTERCOLLECTORSYSTEMDCONFIG

14.20.1. 説明

NodeExporterCollectorSystemdConfig リソースを使用して、**node-exporter** エージェントの **systemd** コレクターを有効または無効にします。デフォルトでは、**systemd** コレクターは無効になっています。有効にすると、次のメトリクスが使用可能になります (**node_systemd_system_running**、**node_systemd_units**、**node_systemd_version**)。ユニットがソケットを使用する場合、次のメトリクスも生成します (**node_systemd_socket_accepted_connections_total**、**node_systemd_socket_current_connections**、**node_systemd_socket_refused_connections_total**)。**units** パラメーターを使用して、**systemd** コレクターに含める **systemd** ユニットを選択できます。選択したユニットは、各 **systemd** ユニットの状態を示す **node_systemd_unit_state** メトリックを生成するために使用されます。ただし、このメトリックのカーディナリティーは高くなる可能性があります (ノードごとのユニットごとに少なくとも 5 シリーズ)。選択したユニットの長いリストを使用してこのコレクターを有効にする場合は、過剰なメモリー使用量がないか **prometheus-k8s** デプロイメントを注意深く監視してください。**node_systemd_timer_last_trigger_seconds** メトリックは、**units** パラメーターの値を **logrotate.timer** として設定した場合にのみ表示されることに注意してください。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**systemd** コレクターを有効または無効にするブール値のフラグ。

プロパティー	型	説明
units	[]string	**systemd** コレクターに組み込まれる systemd ユニットに一致する正規表現 (regex) パターンのリスト。デフォルトでは、リストは空であるため、コレクターは systemd ユニットのメトリクスを公開しません。

14.21. NODEEXPORTERCOLLECTORTCPSTATCONFIG

14.21.1. 説明

NodeExporterCollectorTcpStatConfig リソースは、**node-exporter** エージェントの **tcpstat** コレクターのオン/オフスイッチとして機能します。デフォルトでは、**tcpstat** コレクターは無効になっています。

表示場所: NodeExporterCollectorConfig

プロパティー	型	説明
enabled	bool	**tcpstat** コレクターを有効または無効にするブール値フラグ。

14.22. NODEEXPORTERCONFIG

14.22.1. 説明

NodeExporterConfig リソースは、**node-exporter** エージェントの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
コレクター	NodeExporterCollectorConfig	有効にするコレクターと、それらの追加の設定パラメーターを定義します。

プロパティー	型	説明
maxProcs	uint32	node-exporter のプロセスが実行する CPU のターゲット数。デフォルト値は **0** で、node-exporter がすべての CPU で実行することを意味します。カーネルのデッドロックが発生した場合、または **sysfs** からの同時読み取り時にパフォーマンスが低下した場合は、この値を **1** に変更できます。これにより、node-exporter が 1 つの CPU で実行するように制限されます。CPU 数が多いノードの場合は、制限を低い数値に設定できます。これにより、Go ルーチンがすべての CPU で実行するようにスケジュールされなくなり、リソースが節約されます。ただし、**maxProcs** 値の設定が低すぎる場合や、収集するメトリクスが多数ある場合は、I/O パフォーマンスが低下します。
ignoredNetworkDevices	*[]string	**netdev** や **netclass** など、関連するコレクター設定から除外するネットワークデバイスのリスト (正規表現として定義)。リストが指定されていない場合、Cluster Monitoring Operator は、メモリー使用量への影響を最小限に抑えるために、除外されるデバイスの事前定義されたリストを使用します。リストが空の場合、デバイスは除外されません。この設定を変更する場合は、過剰なメモリー使用量がないか **prometheus-k8s** デプロイメントを注意深く監視してください。
resources	*v1.ResourceRequirements	**NodeExporter** コンテナーのリソースリクエストと制限を定義します。

14.23. OPENSHIFTSTATEMETRICSCONFIG

14.23.1. 説明

OpenShiftStateMetricsConfig リソースは、**openshift-state-metrics** エージェントの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	**OpenShiftStateMetrics** コンテナーのリソース要求と制限を定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。

14.24. PROMETHEUSK8SCONFIG

14.24.1. 説明

PrometheusK8sConfig リソースは、Prometheus コンポーネントの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Prometheus コンポーネントからアラートを受信する追加の Alertmanager インスタンスを設定します。デフォルトでは、追加の Alertmanager インスタンスは設定されません。
enforcedBodySizeLimit	string	Prometheus が取得したメトリクスに本体サイズの制限を適用します。収集された対象のボディーの応答が制限値よりも大きい場合には、スクレイピングは失敗します。制限なしを指定する空の値、Prometheus サイズ形式の数値 (**64MB**など)、または文字列 **automatic** (制限がクラスターの容量に基づいて自動的に計算されることを示す) などの値が有効です。デフォルト値は空で、制限なしを意味します。

プロパティー	型	説明
externalLabels	map[string]string	フェデレーション、リモートストレージ、Alertmanager などの外部システムと通信する際に、任意の時系列またはアラートに追加されるラベルを定義します。デフォルトでは、ラベルは追加されません。
logLevel	string	Prometheus のログレベル設定を定義します。使用できる値は、**error**、**warn**、**info**、および **debug** です。デフォルト値は **info** です。
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
queryLogFile	string	PromQL クエリーがログに記録されるファイルを指定します。この設定は、ファイル名 (クエリーが **/var/log/prometheus** の **emptyDir** ボリュームに保存される場合)、または **emptyDir** ボリュームがマウントされ、クエリーが保存される場所へのフルパスのいずれかです。**/dev/stderr**、**/dev/stdout**、または **/dev/null** への書き込みはサポートされていますが、他の **/dev/** パスへの書き込みはサポートされていません。相対パスもサポートされていません。デフォルトでは、PromQL クエリーはログに記録されません。
remoteWrite	[]RemoteWriteSpec	URL、認証、再ラベル付け設定など、リモート書き込み設定を定義します。
resources	*v1.ResourceRequirements	**Prometheus** コンテナーのリソース要求および制限を定義します。

プロパティー	型	説明
retention	string	Prometheus がデータを保持する期間を定義します。この定義は、次の正規表現パターンを使用して指定する必要があります (**0-9+(ms|s|m|h|d|w|y)** (ms=ミリ秒、s=秒、m=分、h=時間、d=日、w=週、y=年))。デフォルト値は **15d** です。
retentionSize	string	データブロックと先行書き込みログ (WAL) によって使用されるディスク領域の最大量を定義します。サポートされる値は、**B**、**KB**、**KiB**、**MB**、**MiB**、**GB**、**GiB**、**TB**、**TiB**、**PB**、**PiB**、**EB**、および **EiB** です。デフォルトでは、制限は定義されません。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。
collectionProfile	CollectionProfile	Prometheus がプラットフォームコンポーネントからメトリクスを収集するために使用するメトリクスコレクションプロファイルを定義します。使用可能な値は、**full** または **minimal** です。**full** プロファイル (デフォルト) では、Prometheus はプラットフォームコンポーネントが公開するメトリクスをすべて収集します。**minimal** プロファイルでは、Prometheus はデフォルトのプラットフォームアラート、レコーディングルール、Telemetry、およびコンソールダッシュボードに必要なメトリクスのみ収集します。
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Prometheus の永続ストレージを定義します。この設定を使用して、ストレージクラス、ボリュームサイズ、名前などの永続ボリューム要求を設定します。

14.25. PROMETHEUSOPERATORCONFIG

14.25.1. 説明

PrometheusOperatorConfig リソースは、Prometheus Operator コンポーネントの設定を定義します。

表示場所: ClusterMonitoringConfiguration、UserWorkloadConfiguration

プロパティー	型	説明
logLevel	string	Prometheus Operator のログレベル設定を定義します。使用できる値は、**error**、**warn**、**info**、および **debug** です。デフォルト値は **info** です。
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	**PrometheusOperator** コンテナーのリソース要求と制限を定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。

14.26. PROMETHEUSRESTRICTEDCONFIG

14.26.1. 説明

PrometheusRestrictedConfig リソースは、ユーザー定義プロジェクトをモニターする Prometheus コンポーネントの設定を定義します。

表示場所: UserWorkloadConfiguration

プロパティー	型	説明
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Prometheus コンポーネントからアラートを受信する追加の Alertmanager インスタンスを設定します。デフォルトでは、追加の Alertmanager インスタンスは設定されません。

プロパティー	型	説明
enforcedLabelLimit	*uint64	サンプルで受け入れられるラベルの数に、収集ごとの制限を指定します。メトリクスの再ラベル後にラベルの数がこの制限を超えると、スクレイプ全体が失敗として扱われます。デフォルト値は **0** で、制限が設定されていないことを意味します。
enforcedLabelNameLengthLimit	*uint64	サンプルのラベル名の長さにスクレイプごとの制限を指定します。ラベル名の長さがメトリクスの再ラベル付け後にこの制限を超える場合には、スクレイプ全体が失敗として扱われます。デフォルト値は **0** で、制限が設定されていないことを意味します。
enforcedLabelValueLengthLimit	*uint64	サンプルのラベル値の長さにスクレイプごとの制限を指定します。ラベル値の長さがメトリクスの再ラベル付け後にこの制限を超える場合、スクレイプ全体が失敗として扱われます。デフォルト値は **0** で、制限が設定されていないことを意味します。
enforcedSampleLimit	*uint64	受け入れられるスクレイプされたサンプル数のグローバル制限を指定します。この設定は、値が **enforcedTargetLimit** よりも大きい場合、ユーザー定義の **ServiceMonitor** または **PodMonitor** オブジェクトに設定された **SampleLimit** 値を上書きします。管理者は、この設定を使用して、サンプルの総数を制御できます。デフォルト値は **0** で、制限が設定されていないことを意味します。

プロパティー	型	説明
enforcedTargetLimit	*uint64	収集された対象数に対してグローバル制限を指定します。この設定は、値が **enforcedSampleLimit** よりも大きい場合、ユーザー定義の **ServiceMonitor** または **PodMonitor** オブジェクトに設定された **TargetLimit** 値を上書きします。管理者は、この設定を使用して、ターゲットの総数を制御できます。デフォルト値は **0** です。
externalLabels	map[string]string	フェデレーション、リモートストレージ、Alertmanager などの外部システムと通信する際に、任意の時系列またはアラートに追加されるラベルを定義します。デフォルトでは、ラベルは追加されません。
logLevel	string	Prometheus のログレベル設定を定義します。使用できる値は、**error**、**warn**、**info**、および **debug** です。デフォルト設定は **info** です。
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
queryLogFile	string	PromQL クエリーがログに記録されるファイルを指定します。この設定は、ファイル名 (クエリーが **/var/log/prometheus** の **emptyDir** ボリュームに保存される場合)、または **emptyDir** ボリュームがマウントされ、クエリーが保存される場所へのフルパスのいずれかです。**/dev/stderr**、**/dev/stdout**、または **/dev/null** への書き込みはサポートされていますが、他の **/dev/** パスへの書き込みはサポートされていません。相対パスもサポートされていません。デフォルトでは、PromQL クエリーはログに記録されません。
remoteWrite	[]RemoteWriteSpec	URL、認証、再ラベル付け設定など、リモート書き込み設定を定義します。

プロパティー	型	説明
resources	*v1.ResourceRequirements	Prometheus コンテナーのリソース要求および制限を定義します。
retention	string	Prometheus がデータを保持する期間を定義します。この定義は、次の正規表現パターンを使用して指定する必要があります (**0-9+(ms|s|m|h|d|w|y)** (ms=ミリ秒、s=秒、m=分、h=時間、d=日、w=週、y=年))。デフォルト値は **15d** です。
retentionSize	string	データブロックと先行書き込みログ (WAL) によって使用されるディスク領域の最大量を定義します。サポートされる値は、**B**、**KB**、**KiB**、**MB**、**MiB**、**GB**、**GiB**、**TB**、**TiB**、**PB**、**PiB**、**EB**、および **EiB** です。デフォルト値は **nil** です。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Prometheus の永続ストレージを定義します。この設定を使用して、ボリュームのストレージクラスおよびサイズを設定します。

14.27. REMOTEWRITESPEC

14.27.1. 説明

RemoteWriteSpec リソースは、リモート書き込みストレージの設定を定義します。

14.27.2. 必須

- **url**

出現場所: PrometheusK8sConfig、 PrometheusRestrictedConfig

プロパティー	型	説明
認可	*monv1.SafeAuthorization	リモート書き込みストレージの認証設定を定義します。

プロパティー	型	説明
basicAuth	*monv1.BasicAuth	リモート書き込みエンドポイント URL の Basic 認証設定を定義します。
bearerTokenFile	string	リモート書き込みエンドポイントのベアラートークンが含まれるファイルを定義します。ただし、シークレットを Pod にマウントできないため、実際にはサービスアカウントのトークンのみを参照できます。
headers	map[string]string	各リモート書き込み要求とともに送信されるカスタム HTTP ヘッダーを指定します。Prometheus によって設定されるヘッダーは上書きできません。
metadataConfig	*monv1.MetadataConfig	シリーズのメタデータをリモート書き込みストレージに送信するための設定を定義します。
name	string	リモート書き込みキューの名前を定義します。この名前は、メトリクスとロギングでキューを区別するために使用されます。指定する場合、この名前は一意である必要があります。
oauth2	*monv1.OAuth2	リモート書き込みエンドポイントの OAuth2 認証設定を定義します。
proxyUrl	string	オプションのプロキシー URL を定義します。
queueConfig	*monv1.QueueConfig	リモート書き込みキューパラメーターの調整を許可します。
remoteTimeout	string	リモート書き込みエンドポイントへの要求のタイムアウト値を定義します。

プロパティー	型	説明
sendExemplars	*bool	リモート書き込みによるエグザンプラーの送信を有効にします。この設定を有効にすると、最大 100,000 個のエグザンプラーをメモリーに保存するように Prometheus が設定されます。この設定はユーザー定義のモニタリングにのみ適用され、コアプラットフォームのモニタリンには適用されません。
sigv4	*monv1.Sigv4	AWS 署名バージョン 4 の認証設定を定義します。
tlsConfig	*monv1.SafeTLSConfig	リモート書き込みエンドポイントの TLS 認証設定を定義します。
url	string	サンプルの送信先となるリモート書き込みエンドポイントの URL を定義します。
writeRelabelConfigs	[]monv1.RelabelConfig	リモート書き込みの再ラベル設定のリストを定義します。

14.28. TLSCONFIG

14.28.1. 説明

TLSConfig リソースは、TLS 接続の設定を設定します。

14.28.2. 必須

- **insecureSkipVerify**

表示場所: AdditionalAlertmanagerConfig

プロパティー	型	説明
ca	*v1.SecretKeySelector	リモートホストに使用する認証局 (CA) を含む秘密鍵の参照を定義します。
cert	*v1.SecretKeySelector	リモートホストに使用する公開証明書を含む秘密鍵の参照を定義します。

プロパティー	型	説明
鍵 (key)	*v1.SecretKeySelector	リモートホストに使用する秘密鍵を含む秘密鍵の参照を定義します。
serverName	string	返された証明書のホスト名を確認するために使用されます。
insecureSkipVerify	bool	**true** に設定すると、リモートホストの証明書および名前の検証が無効になります。

14.29. TELEMETERCLIENTCONFIG

14.29.1. 説明

TelemeterClientConfig は、Telemeter Client コンポーネントの設定を定義します。

14.29.2. 必須

- **nodeSelector**
- **tolerations**

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	**TelemeterClient** コンテナーのリソース要求と制限を定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。

14.30. THANOSQUERIERCONFIG

14.30.1. 説明

ThanosQuerierConfig リソースは、Thanos Querier コンポーネントの設定を定義します。

表示場所: ClusterMonitoringConfiguration

プロパティー	型	説明
enableRequestLogging	bool	要求ロギングを有効または無効にするブール値フラグ。デフォルト値は **false** です。
logLevel	string	Thanos Querier のログレベル設定を定義します。使用できる値は、**error**、**warn**、**info**、および **debug** です。デフォルト値は **info** です。
enableCORS	bool	CORS ヘッダーの設定を可能にするブール型フラグ。ヘッダーにより、あらゆる発信元からのアクセスが許可されます。デフォルト値は **false** です。
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	Thanos Querier コンテナーのリソース要求および制限を定義します。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。

14.31. THANOSRULERCONFIG

14.31.1. 説明

ThanosRulerConfig リソースは、ユーザー定義プロジェクトの Thanos Ruler インスタンスの設定を定義します。

表示場所: UserWorkloadConfiguration

プロパティー	型	説明
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Thanos Ruler コンポーネントが追加の Alertmanager インスタンスと通信する方法を設定します。デフォルト値は **nil** です。

プロパティー	型	説明
logLevel	string	Thanos Ruler のログレベル設定を定義します。使用できる値は、**error**、**warn**、**info**、および **debug** です。デフォルト値は **info** です。
nodeSelector	map[string]string	Pod がスケジュールされるノードを定義します。
resources	*v1.ResourceRequirements	Alertmanager コンテナーのリソース要求および制限を定義します。
retention	string	Prometheus がデータを保持する期間を定義します。この定義は、次の正規表現パターンを使用して指定する必要があります (**0-9+ (ms|s|m|h|d|w|y)** (ms=ミリ秒、s=秒、m=分、h=時間、d=日、w=週、y=年))。デフォルト値は **15d** です。
tolerations	[]v1.Toleration	Pod の容認を定義します。
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Pod のトポロジー分散制約を定義します。
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Thanos Ruler の永続ストレージを定義します。この設定を使用して、ボリュームのストレージクラスおよびサイズを設定します。

14.32. USERWORKLOADCONFIGURATION

14.32.1. 説明

UserWorkloadConfiguration リソースは、**openshift-user-workload-monitoring** namespace の **user-workload-monitoring-config** Config Map でユーザー定義プロジェクトに対応する設定を定義します。userWorkload **Configuration** は、**openshift-monitoring** namespace の下にある **cluster-monitoring-config** Config Map で **enableUserWorkload** を **true** に設定した後にのみ有効にできます。

プロパティー	型	説明
alertmanager	*AlertmanagerUserWorkloadConfig	ユーザーワークロードモニタリングで Alertmanager コンポーネントの設定を定義します。

プロパティー	型	説明
prometheus	*PrometheusRestrictedConfig	ユーザーワークロードモニタリングで Prometheus コンポーネントの設定を定義します。
prometheusOperator	*PrometheusOperatorConfig	ユーザーワークロードモニタリングでの Prometheus Operator コンポーネントの設定を定義します。
thanosRuler	*ThanosRulerConfig	ユーザーワークロードモニタリングで Thanos Ruler コンポーネントの設定を定義します。

第15章 CLUSTER OBSERVABILITY OPERATOR

15.1. CLUSTER OBSERVABILITY OPERATOR リリースノート

重要

Cluster Observability Operator はテクノロジープレビュー機能としてのみ使用できます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Cluster Observability Operator (COO) は、オプションの OpenShift Container Platform Operator です。管理者はこれを使用して、さまざまなサービスやユーザーが使用できるように個別に設定できる、スタンドアロンのモニタリングスタックを作成できます。

COO は、OpenShift Container Platform のビルトインモニタリング機能を補完します。これは、Cluster Monitoring Operator (CMO) で管理されるデフォルトのプラットフォームおよびユーザーワークロードモニタリングスタックと並行してデプロイできます。

これらのリリースノートは、OpenShift Container Platform での Cluster Observability Operator の開発を追跡します。

15.1.1. Cluster Observability Operator 0.1.3

Cluster Observability Operator 0.1.3 については、次のアドバイザリーを利用できます。

- RHEA-2024:1744 Cluster Observability Operator 0.1.3

15.1.1.1. バグ修正

- 以前は、**http://<prometheus_url>:9090/graph** で Prometheus Web ユーザーインターフェイス (UI)にアクセスしようとすると、**Error open React index.html: open web/ui/static/react/index.html: no such file or directory** というエラーメッセージが表示されていました。本リリースでは問題が解決し、Prometheus Web UI が正しく表示されるようになりました。(**COO-34**)

15.1.2. Cluster Observability Operator 0.1.2

Cluster Observability Operator 0.1.2 では、次のアドバイザリーを利用できます。

- RHEA-2024:1534 Cluster Observability Operator 0.1.2

15.1.2.1. CVE

- CVE-2023-45142

15.1.2.2. バグ修正

- 以前は、特定のクラスターサービスバージョン (CSV) アノテーションが COO のメタデータに含まれていませんでした。これらのアノテーションが欠落していたため、COO の一部の特長と機能がパッケージマニフェストまたは OperatorHub ユーザーインターフェイスに表示されませんでした。このリリースで、欠落していたアノテーションが追加され、この問題が解決されました。(COO-11)

- 以前は、COO の自動更新が機能せず、OperatorHub で新しいバージョンが利用可能であっても、Operator の新しいバージョンによって古いバージョンが自動的に置き換えられませんでした。このリリースでこの問題が解決されました。(COO-12)

- 以前は、Thanos Querier が 127.0.0.1 (**localhost**) のポート 9090 でネットワークトラフィックのみをリッスンしていたため、Thanos Querier サービスにアクセスしようとすると **502 Bad Gateway** エラーが発生しました。このリリースで、Thanos Querier 設定が更新され、コンポーネントがデフォルトポート (10902) でリッスンするようになり、問題が解決されました。この変更の結果、必要に応じて、Server-Side Apply (SSA) を使用してポートを変更し、プロキシーチェーンを追加することもできるようになりました。(COO-14)

15.1.3. Cluster Observability Operator 0.1.1

Cluster Observability Operator 0.1.1 では、次のアドバイザリーを利用できます。

- 2024:0550 Cluster Observability Operator 0.1.1

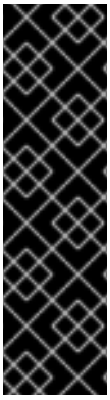
15.1.3.1. 新機能および機能拡張

このリリースでは、Cluster Observability Operator が更新され、制限されたネットワークまたは非接続環境での Operator のインストールがサポートされるようになりました。

15.1.4. Cluster Observability Operator 0.1

このリリースでは、Cluster Observability Operator のテクノロジープレビューバージョンが OperatorHub で利用できるようになります。

15.2. CLUSTER OBSERVABILITY OPERATOR の概要



重要

Cluster Observability Operator はテクノロジープレビュー機能としてのみ使用できます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Cluster Observability Operator (COO) は、オプションの OpenShift Container Platform コンポーネントです。これをデプロイして、さまざまなサービスやユーザーが使用できるように個別に設定可能なスタンドアロンのモニタリングスタックを作成できます。

COO は、次のモニタリングコンポーネントをデプロイします。

- Prometheus

- Thanos Querier (オプション)
- Alertmanager (オプション)

COO コンポーネントは、Cluster Monitoring Operator (CMO) でデプロイおよび管理されるデフォルトのクラスター内モニタリングスタックとは独立して機能します。2 つの Operator でデプロイされたモニタリングスタックは競合しません。CMO でデプロイされたデフォルトのプラットフォームモニタリングコンポーネントに加え、COO モニタリングスタックを使用できます。

15.2.1. Cluster Observability Operator について

Cluster Observability Operator (COO) で作成されたデフォルトのモニタリングスタックには、リモート書き込みを使用して外部エンドポイントにメトリクスを送信できる可用性の高い Prometheus インスタンスが含まれています。

各 COO スタックには、中央の場所から高可用性 Prometheus インスタンスをクエリーするために使用できるオプションの Thanos Querier コンポーネントと、さまざまなサービスのアラート設定をセットアップするために使用できるオプションの Alertmanager コンポーネントも含まれています。

15.2.1.1. Cluster Observability Operator を使用する利点

COO が使用する **MonitoringStack** CRD は、COO でデプロイされたモニタリングコンポーネントに対して独自のデフォルトモニタリング設定を提供しますが、より複雑な要件に合わせてカスタマイズすることも可能です。

COO で管理されるモニタリングスタックを導入すると、Cluster Monitoring Operator (CMO) でデプロイされたコアプラットフォームモニタリングスタックを使用したのでは対処することが難しい、または困難なモニタリングニーズを満たすことができます。COO を使用して導入されたモニタリングスタックには、コアプラットフォームとユーザーワークロードのモニタリングに比べて次の利点があります。

拡張性

ユーザーは、COO でデプロイされたモニタリングスタックにさらに多くのメトリクスを追加できますが、これをコアプラットフォームモニタリングで行った場合はサポートされません。さらに、COO で管理されるスタックは、フェデレーションを使用して、コアプラットフォームのモニタリングから特定のクラスター固有のメトリクスを受け取ることができます。

マルチテナンシーのサポート

COO は、ユーザー namespace ごとにモニタリングスタックを作成できます。namespace ごとに複数のスタックをデプロイしたり、複数の namespace に単一のスタックをデプロイしたりすることもできます。たとえば、クラスター管理者、SRE チーム、開発チームは、モニタリングコンポーネントの単一の共有スタックを使用するのではなく、独自のモニタリングスタックを単一のクラスターにデプロイできます。その後、各チームのユーザーは、アプリケーションやサービスのアラート、アラートルーティング、アラートレシーバーなどの機能を個別に設定できます。

スケーラビリティー

必要に応じて、COO で管理されるモニタリングスタックを作成できます。単一のクラスター上で複数のモニタリングスタックを実行できるため、手動シャーディングを使用することで非常に大規模なクラスターを容易に監視できます。この機能は、メトリクスの数が単一の Prometheus インスタンスのモニタリング能力を超える場合に対処します。

柔軟性

Operator Lifecycle Manager (OLM) を使用して COO をデプロイすると、COO リリースが OpenShift Container Platform リリースサイクルから切り離されます。このデプロイメント方法により、リリースイテレーションが短縮され、変化する要件や問題に迅速に対応できるようになります。さらに、COO で管理されるモニタリングスタックをデプロイすることで、ユーザーは OpenShift Container Platform のリリースサイクルとは別にアラートルールを管理できます。

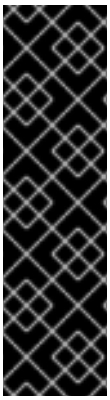
高度なカスタマイズが可能

COO は、カスタマイズを強化する Server-Side Apply (SSA) を使用して、カスタムリソース内にある 1 つの設定可能フィールドの所有権をユーザーに委譲できます。

関連情報

- Server-Side Apply (SSA) に関する Kubernetes ドキュメント

15.3. CLUSTER OBSERVABILITY OPERATOR のインストール



重要

Cluster Observability Operator はテクノロジープレビュー機能としてのみ使用できます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

クラスター管理者は、OpenShift Container Platform Web コンソールまたは CLI を使用して、OperatorHub から Cluster Observability Operator (COO) をインストールできます。OperatorHub は、クラスター上に Operator をインストールして管理する Operator Lifecycle Manager (OLM) と連動して動作するユーザーインターフェイスです。

OperatorHub を使用して COO をインストールするには、クラスターに Operator を追加する で説明されている手順に従います。

15.3.1. Web コンソールを使用して Cluster Observability Operator をアンインストールする

OperatorHub を使用して Cluster Observability Operator (COO) をインストールした場合は、OpenShift Container Platform Web コンソールでそれをアンインストールできます。

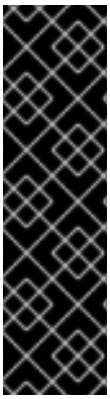
前提条件

- **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできます。
- OpenShift Container Platform Web コンソールにログインしている。

手順

1. **Operators → Installed Operators** に移動します。
2. リスト内で **Cluster Observability Operator** エントリーを見つけます。
3. このエントリーの  をクリックし、**Uninstall Operator** を選択します。

15.4. サービスを関しするための CLUSTER OBSERVABILITY OPERATOR 設定



重要

Cluster Observability Operator はテクノロジープレビュー機能としてのみ使用できます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Cluster Observability Operator (COO) で管理されるモニタリングスタックを設定することで、サービスのメトリクスを監視できます。

サービスのモニタリングをテストするには、次の手順に従います。

- サービスエンドポイントを定義するサンプルサービスをデプロイします。
- COO によるサービスのモニタリング方法を指定する **ServiceMonitor** オブジェクトを作成します。
- **ServiceMonitor** オブジェクトを検出するための **MonitoringStack** オブジェクトを作成します。

15.4.1. Cluster Observability Operator のサンプルサービスをデプロイする

この設定では、ユーザー定義の **ns1-coo** プロジェクトに **prometheus-coo-example-app** という名前のサンプルサービスをデプロイします。このサービスは、カスタム **version** メトリクスを公開します。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとして、または namespace の管理権限を持つユーザーとして、クラスターにアクセスできる。

手順

1. **prometheus-coo-example-app.yaml** という名前の YAML ファイルを作成します。このファイルには、namespace、デプロイメント、およびサービスに関する次の設定の詳細が含まれます。

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns1-coo
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: prometheus-coo-example-app
```

```
  name: prometheus-coo-example-app
  namespace: ns1-coo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-coo-example-app
  template:
    metadata:
      labels:
        app: prometheus-coo-example-app
    spec:
      containers:
      - image: ghcr.io/rhobs/prometheus-example-app:0.4.2
        imagePullPolicy: IfNotPresent
        name: prometheus-coo-example-app
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: prometheus-coo-example-app
  name: prometheus-coo-example-app
  namespace: ns1-coo
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
    name: web
  selector:
    app: prometheus-coo-example-app
  type: ClusterIP
```

2. ファイルを保存します。

3. 次のコマンドを実行して、設定をクラスターに適用します。

```
$ oc apply -f prometheus-coo-example-app.yaml
```

4. 次のコマンドを実行して出力を確認し、Pod が実行されていることを確認します。

```
$ oc -n -ns1-coo get pod
```

出力例

```
NAME                                  READY   STATUS    RESTARTS   AGE
prometheus-coo-example-app-0927545cb7-anskj   1/1     Running   0          81m
```

15.4.2. Cluster Observability Operator によるサービスのモニタリング方法を指定する

「Cluster Observability Operator のサンプルサービスをデプロイする」セクションで作成したサンプルサービスが公開するメトリクスを使用するには、/**metrics** エンドポイントからメトリクスを取得するようにモニタリングコンポーネントを設定する必要があります。

この設定は、サービスのモニタリング方法を指定する **ServiceMonitor** オブジェクト、または Pod のモニタリング方法を指定する **PodMonitor** オブジェクトを使用して作成できます。**ServiceMonitor** オブジェクトには **Service** オブジェクトが必要です。**PodMonitor** オブジェクトには必要ないため、**MonitoringStack** オブジェクトは Pod が公開するメトリクスエンドポイントから直接メトリクスを取得できます。

この手順は、**ns1-coo** namespace に **prometheus-coo-example-app** という名前のサンプルサービスの **ServiceMonitor** オブジェクトを作成する方法を示しています。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとして、または namespace の管理権限を持つユーザーとして、クラスターにアクセスできる。
- Cluster Observability Operator がインストールされている。
- **prometheus-coo-example-app** サンプルサービスを **ns1-coo** namespace にデプロイしている。

注記

prometheus-example-app サンプルサービスは、TLS 認証をサポートしていません。

手順

1. 次の **ServiceMonitor** オブジェクト設定の詳細を含む YAML ファイルを、**example-coo-app-service-monitor.yaml** という名前で作成します。

```
apiVersion: monitoring.rhobs/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: prometheus-coo-example-monitor
  name: prometheus-coo-example-monitor
  namespace: ns1-coo
spec:
  endpoints:
  - interval: 30s
    port: web
    scheme: http
  selector:
    matchLabels:
      app: prometheus-coo-example-app
```

   この設定は、**prometheus-coo-example-app** サンプルサービスが公開するメトリクスデータを収集するために **MonitoringStack** オブジェクトが参照する **ServiceMonitor** オブジェクトを定義します。

2. 次のコマンドを実行して、設定をクラスターに適用します。

```
$ oc apply -f example-app-service-monitor.yaml
```

3. 次のコマンドを実行して出力を観察し、**ServiceMonitor** リソースが作成されたことを確認します。

```
$ oc -n ns1-coo get servicemonitor
```

出力例

```
NAME                          AGE
prometheus-coo-example-monitor   81m
```

15.4.3. Cluster Observability Operator の MonitoringStack オブジェクトを作成する

ターゲット **prometheus-coo-example-app** サービスが公開するメトリクスデータを収集するには、「Cluster Observability Operator でサービスを監視する方法を指定する」セクションで作成した **ServiceMonitor** オブジェクトを参照する **MonitoringStack** オブジェクトを作成します。この **MonitoringStack** オブジェクトはサービスを検出し、そこから公開されているメトリクスデータを収集できます。

前提条件

- **cluster-admin** クラスターロールを持つユーザーとして、または namespace の管理権限を持つユーザーとして、クラスターにアクセスできる。
- Cluster Observability Operator がインストールされている。
- **prometheus-coo-example-app** サンプルサービスを **ns1-coo** namespace にデプロイしている。
- **ns1-coo** namespace に、**prometheus-coo-example-monitor** という名前の **ServiceMonitor** オブジェクトを作成している。

手順

1. **MonitoringStack** オブジェクト設定の YAML ファイルを作成します。この例では、ファイル名を **example-coo-monitoring-stack.yaml** にします。
2. 以下の **MonitoringStack** オブジェクト設定の詳細を追加します。

   MonitoringStack オブジェクトの例

```
apiVersion: monitoring.rhobs/v1alpha1
kind: MonitoringStack
metadata:
  name: example-coo-monitoring-stack
  namespace: ns1-coo
spec:
  logLevel: debug
  retention: 1d
  resourceSelector:
    matchLabels:
      k8s-app: prometheus-coo-example-monitor
```

3. 次のコマンドを実行して、**MonitoringStack** オブジェクトを適用します。

```
$ oc apply -f example-coo-monitoring-stack.yaml
```

4. 次のコマンドを実行し、出力で **MonitoringStack** オブジェクトが利用可能であることを確認します。

```
$ oc -n ns1-coo get monitoringstack
```

出力例

```
NAME                        AGE
example-coo-monitoring-stack   81m
```