OpenShift Dedicated 4

Monitoring

OpenShift Dedicated でのプロジェクトのモニタリング

Last Updated: 2024-07-23

OpenShift Dedicated 4 Monitoring

OpenShift Dedicated でのプロジェクトのモニタリング

法律上の通知

概要

このドキュメントは、OpenShift Dedicated でのプロジェクトのモニタリングに関する情報を提供します。

目次

第1章 モニタリングの概要

1.1. OPENSHIFT DEDICATED モニタリングについて

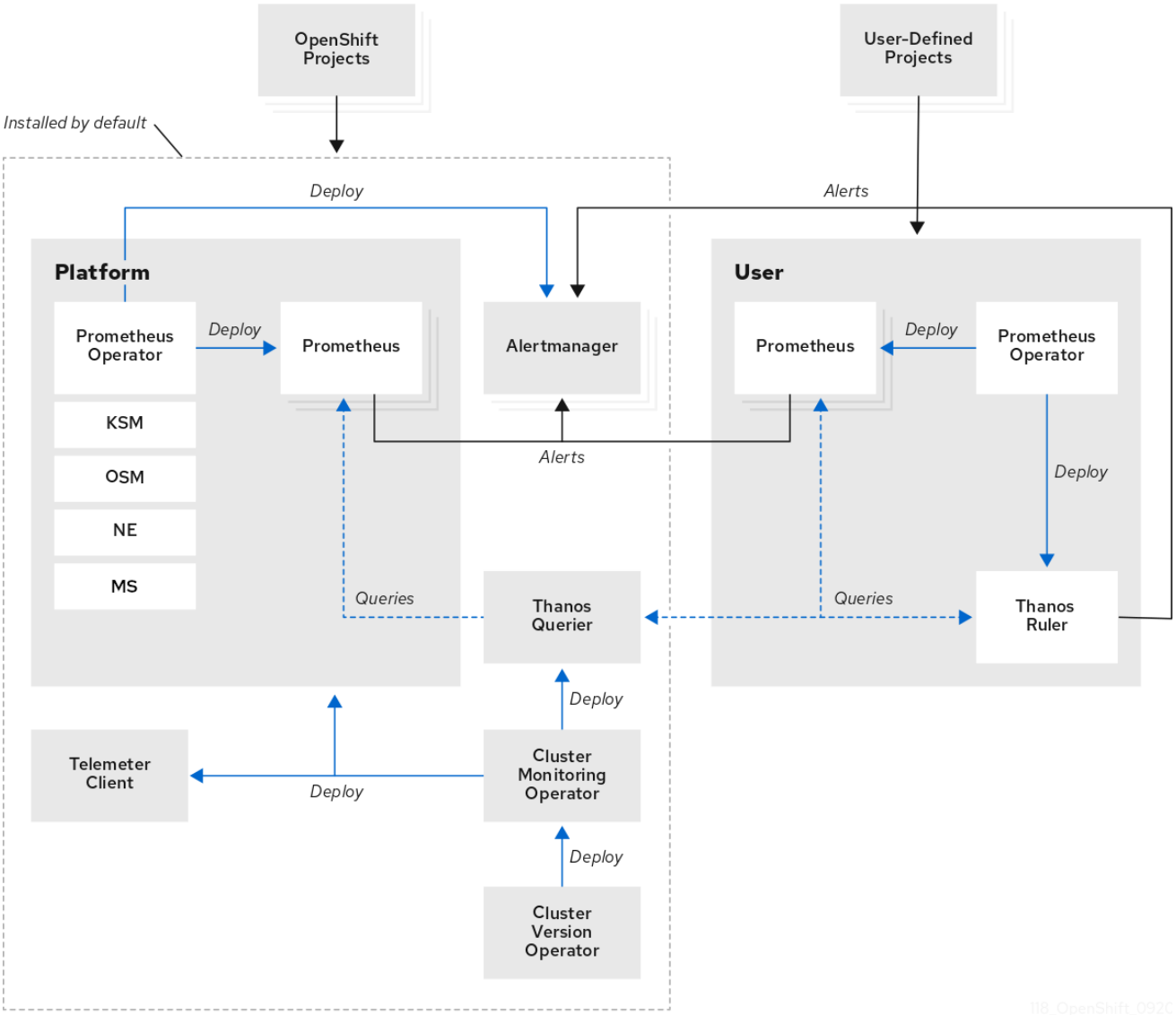
OpenShift Dedicated では、Red Hat Site Reliability Engineering (SRE) プラットフォームメトリクスから切り離して独自のプロジェクトをモニターできます。モニタリングソリューションを追加せずに独自のプロジェクトをモニターできます。

1.2. モニタリングスタックについて

OpenShift Dedicated 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 Dedicated 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。Red Hat Site Reliability Engineer (SRE) は、これらのコンポーネントを使用して、Kubernetes サービスを含むコアクラスターコンポーネントを監視します。これには、全 namespace に含まれるすべてのワークロードから収集された CPU やメモリーなどの重要なメトリクスが含まれます。
  これらのコンポーネントは、以下の図の **Installed by default** 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**。ユーザー定義のプロジェクトモニタリングコンポーネントのセットは、OpenShift Dedicated のインストール中にデフォルトで **openshift-user-workload-monitoring** プロジェクトにインストールされます。これらのコンポーネントを使用して、ユーザー定義プロジェクトのサービスと Pod をモニターできます。これらのコンポーネントは、以下の図の **User** セクションで説明されています。

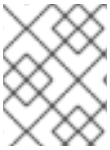


1.2.1. デフォルトのモニタリングターゲット

以下は、OpenShift Dedicated クラスターで Red Hat Site Reliability Engineer (SRE) によって監視されるターゲットの例です。

- CoreDNS
- etcd
- 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)



注記

ターゲットの正確なリストは、クラスターの機能とインストールされているコンポーネントによって異なる場合があります。

関連情報

- 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。

1.2.2. 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

OpenShift Dedicated 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1.1 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。
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 Dedicated では、Thanos Ruler はユーザー定義プロジェクトのモニタリングについて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。

これら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 Dedicated の更新時に自動的に更新されます。

1.2.3. 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

モニタリングは、OpenShift Dedicated のユーザー定義プロジェクトについてデフォルトで有効にされます。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クス。
- ユーザー定義プロジェクトで実行される Pod。

1.3. OPENSHIFT DEDICATED モニタリングの一般用語の用語集

この用語集では、OpenShift Dedicated 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。

Cluster Monitoring 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ます。クラスター Operator の多くは、デフォルトで OpenShift Dedicated にインストールされます。

config map

config map 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 config map 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。

コンテナー

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 Dedicated のキー/値ストアであり、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 Dedicated 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。

Metrics Server

Metrics Server モニタリングコンポーネントはリソースメトリクスを収集し、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。

ノード

OpenShift Dedicated クラスター内のワーカーマシンです。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 Dedicated 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です。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 Dedicated 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、メトリクス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 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。

ストレージ

OpenShift Dedicated は、AWS および GCP 上のさまざまなタイプのストレージをサポートします。OpenShift Dedicated クラスターでは、永続データと非永続データ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 Dedicated では、Thanos Ruler はユーザー定義プロジェクトのモニタリングについてのルールおよびアラート評価を提供します。

Vector

Vector は、各 OpenShift Dedicated ノードにデプロイされ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 Dedicated を管理するためのユーザーインターフェイス (UI)。

第2章 ユーザー定義プロジェクトのモニタリングのアクセス

OpenShift Dedicated クラスターをインストールすると、ユーザー定義プロジェクトのモニタリングがデフォルトで有効になります。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と、追加のモニタリングソリューションを必要とせずに、独自の OpenShift Dedicated プロジェクトをモニタリングできます。

dedicated-admin ユーザーには、ユーザー定義プロジェクトのモニタリングを設定し、アクセスするためのデフォルトのパーミッションがあります。

注記

カスタム Prometheus インスタンスおよび Operator Lifecycle Manager (OLM) でインストールされる Prometheus Operator では、ユーザー定義のプロジェクトモニタリングが有効である場合にこれに関する問題が生じる可能性があります。カスタム Prometheus インスタンスはサポートされません。

必要に応じて、クラスターのインストール中またはインストール後に、ユーザー定義プロジェクトの監視を無効にすることができます。

第3章 モニタリングスタックの設定

このセクションでは、サポートされる設定について説明し、ユーザー定義プロジェクトのモニタリングスタックを設定する方法を示し、いくつかの一般的な設定シナリオを示します。

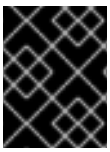
重要

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。

3.1. 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。OpenShift Dedicated モニタリング設定で唯一サポートされている方法は、Cluster Monitoring Operator (CMO) 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、デフォルトおよび設計により、CMO 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。



重要

別の Prometheus インスタンスのインストールは、Red Hat Site Reliability Engineers (SRE) ではサポートされていません。

3.1.1. 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。

- **カスタム Prometheus インスタンスの OpenShift Dedicated 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。
- **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変更します。cluster-monitoring-config** config map で定義されているコンポーネントは変更しないでください。Red Hat SRE は、これらのコンポーネントを使用して、コアクラスターコンポーネントと Kubernetes サービスをモニターします。

3.1.2. モニタリングコンポーネントのバージョンマトリックスのサポート

以下のマトリックスには、OpenShift Dedicated 4.12 以降のリリースのモニタリングコンポーネントのバージョンに関する情報が含まれています。

表3.1 OpenShift Dedicated およびコンポーネントのバージョン

OpenShift Dedicated	Prometheus Operator	Prometheus	Metrics Server	Alertmanager	kube-state-metrics エージェント	monitoring-plugin	node-exporter エージェント	Thanos
4.16	0.73.2	2.52.0	0.7.1	0.26.0	2.12.0	1.0.0	1.8.0	0.35.0
4.15	0.70.0	2.48.0	0.6.4	0.26.0	2.10.1	1.0.0	1.7.0	0.32.5
4.14	0.67.1	2.46.0	該当なし	0.25.0	2.9.2	1.0.0	1.6.1	0.30.2
4.13	0.63.0	2.42.0	該当なし	0.25.0	2.8.1	該当なし	1.5.0	0.30.2
4.12	0.60.1	2.39.1	該当なし	0.24.0	2.6.0	該当なし	1.4.0	0.28.1

注記

openshift-state-metrics エージェントと Telemeter Client は、OpenShift 固有のコンポーネントです。したがって、それらのバージョンは OpenShift Dedicated のバージョンに対応します。

3.2. モニタリングスタックの設定

OpenShift Dedicated では、**user-workload-monitoring-config ConfigMap** オブジェクトを使用してユーザー定義プロジェクトのワークロードをモニターするスタックを設定できます。config map が Cluster Monitoring Operator (CMO) を設定し、続いて CMO がスタックのコンポーネントを設定します。

前提条件

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。
   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。

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。

警告

ConfigMap オブジェクトの設定変更によって、結果も異なります。

- Pod は再デプロイされません。したがって、サービスの停止はありません。
- 変更された Pod が再デプロイされます。
  - 単一ノードクラスターの場合、一時的なサービスが停止します。
  - マルチノードクラスターの場合、高可用性であるため、影響を受ける Pod は徐々にロールアウトされ、モニタリングスタックは引き続き利用可能です。
  - 永続ボリュームの設定およびサイズ変更を行うと、高可用性であるかどうかに関係なく、常にサービスが停止します。

config map の変更を必要とする手順にはそれぞれ、想定される結果が含まれます。

関連情報

- **user-workload-monitoring-config** config map の設定リファレンス

3.3. 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、**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。

警告

Cluster-monitoring-config ConfigMap オブジェクト内のモニタリングコンポーネントを変更しないでください。Red Hat Site Reliability Engineer (SRE) は、これらのコンポーネントを使用して、コアクラスターコンポーネントと Kubernetes サービスをモニターします。

表3.2 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	user-workload-monitoring-config config map キー
Alertmanager	**alertmanager**
Prometheus Operator	**prometheusOperator**

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

3.4. 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。

3.4.1. 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

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

3.4.2. モニタリングコンポーネントの異なるノードへの移動

ユーザー定義プロジェクトのワークロードをモニターする任意のコンポーネントを特定のワーカーノードに移動できます。コンポーネントをコントロールプレーンまたはインフラストラクチャーノードに移動することは許可されていません。

前提条件

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。
   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自動的に新しいノードに移動され、新しい設定の影響を受ける Pod は再デプロイされます。

関連情報

- **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。

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

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許容値を割り当てて、テイントされたワーカーノードにプロジェクトを移動できるようにすることができます。コントロールプレーンまたはインフラストラクチャーノードでのスケジューリングは許可されていません。

前提条件

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトは、**openshift-user-workload-monitoring** namespace に存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。
   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。新しい設定の影響を受ける Pod は自動的に再デプロイされます。

関連情報

- テイントおよび許容値は、Kubernetes ドキュメント を参照してください。

3.6. 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。

3.6.1. 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)
- Metrics Server
- 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。

3.6.2. 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

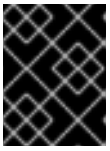
- **コアプラットフォームモニタリングコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとしてクラスターにアクセスできる。
  - これで、**cluster-monitoring-config** という名前の **ConfigMap** オブジェクトが作成されました。
- **ユーザー定義のプロジェクトをモニターするコンポーネントを設定する場合**:
  - **cluster-admin** クラスターロールを持つユーザーとして、または **openshift-user-workload-monitoring** プロジェクトの **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
metricsServer:
  resources:
    requests:
      cpu: 10m
      memory: 50Mi
    limits:
      cpu: 50m
      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. 変更を適用するためにファイルを保存します。新しい設定の影響を受ける Pod は自動的に再デプロイされます。

関連情報

- Kubernetes の要求と制限に関するドキュメント

3.7. CONFIGURING PERSISTENT STORAGE

永続ストレージを使用してクラスターモニタリングを実行すると、次の利点が得られます。

- メトリクスとアラートデータを永続ボリューム (PV) に保存することで、データ損失から保護します。その結果、Pod が再起動または再作成されても存続できます。
- Alertmanager Pod が再起動したときに、重複した通知を受信したり、アラートの無音が失われたりすることを回避します。

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。

3.7.1. 永続ストレージの前提条件

- ストレージのブロックタイプを使用します。

3.7.2. Persistent Volume Claim (永続ボリューム要求) の設定

コンポーネントの監視に永続ボリューム (PV) を使用するには、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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。
   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>: 1
      volumeClaimTemplate:
        spec:
          storageClassName: <storage_class> 2
          resources:
            requests:
              storage: <amount_of_storage> 3
```

1. PVC を設定するユーザー定義の監視のコンポーネントを指定します。
2. 既存のストレージクラスを指定します。ストレージクラスが指定されていない場合、デフォルトのストレージクラスが使用されます。
3. 必要なストレージの量を指定します。

volumeClaimTemplate の指定方法は、PersistentVolumeClaims に関する Kubernetes ドキュメント を参照してください。

以下の例では、Thanos Ruler の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: my-storage-class
          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。新しい設定の影響を受ける Pod は自動的に再デプロイされ、新しいストレージ設定が適用されます。

警告

PVC 設定で config map を更新すると、影響を受ける **StatefulSet** オブジェクトが再作成され、一時的なサービス停止が発生します。

3.7.3. Prometheus メトリクス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

データを削除するタイミングを変更するために、ユーザー定義のプロジェクトをモニターする Prometheus インスタンスの保持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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。
   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。新しい設定の影響を受ける Pod は自動的に再デプロイされます。

3.7.4. Thanos Ruler メトリクス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クス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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。新しい設定の影響を受ける Pod は自動的に再デプロイされます。

関連情報

- 永続ストレージについて

3.8. 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クス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 がメトリクスを保存する方法や期間には影響はありません。

前提条件

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。シークレットは **openshift-user-workload-monitoring** namespace に作成する必要があります。

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クス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。
   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クス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。新しい設定は自動的に適用されます。

3.8.1. 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

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。
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。

3.8.2. 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。各サンプルは、**openshift-user-workload-monitoring** namespace 内のユーザー定義プロジェクトのモニタリングで使用する認証を設定します。

例3.1 AWS 署名バージョン 4 認証のサンプル YAML

以下は、**openshift-user-workload-monitoring** namespace の **sigv4-credentials** という名前の **sigv4** 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-user-workload-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-user-workload-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: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      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3.2 Basic 認証のサンプル YAML

以下に、**openshift-user-workload-monitoring** namespace 内の **rw-basic-auth** という名前の **Secret** オブジェクトの基本認証設定のサンプルを示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: rw-basic-auth
  namespace: openshift-user-workload-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-user-workload-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: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      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3.3 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-user-workload-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-user-workload-monitoring
stringData:
  token: <authentication_token> 1
type: Opaque
```

1 認証トークン。

以下は、**openshift-user-workload-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設定マップの設定例を示しています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
    prometheus:
      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3.4 OAuth 2.0 認証のサンプル YAML

以下は、**openshift-user-workload-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-user-workload-monitoring
stringData:
  id: <oauth2_id> 1
  secret: <oauth2_secret> 2
type: Opaque
```

1 Oauth 2.0 ID。

2 OAuth 2.0 シークレット。

以下は、**openshift-user-workload-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: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      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3.5 TLS クライアント認証のサンプル YAML

以下は、**openshift-user-workload-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-user-workload-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: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      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。

3.9. クラスター ID ラベルのメトリクスへの追加

複数の OpenShift Dedicated 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クス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クス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、メトリクス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クス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、メトリクス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クス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 ラベルをメトリクスに追加します。
- これらのラベルをクエリーし、メトリクスのソースクラスターまたはカスタマーを特定します。

3.9.1. メトリクスのクラスター ID ラベルの作成

openshift-user-workload-monitoring namespace の **user-workload-monitoring-config** config map の設定を編集することで、メトリクスのクラスター ID ラベルを作成できます。

前提条件

- **dedicated-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。
- **user-workload-monitoring-config** ConfigMap オブジェクトを編集します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。

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クス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クス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** を持つメトリクス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クスのクラスター ID ラベルの名前を指定します。メトリクス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

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クス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. 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。

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。

3.10. 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クス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。

dedicated-admin 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。

3.10.1. 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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。

第4章 外部 ALERTMANAGER インスタンスの設定

OpenShift Dedicated モニタリングスタックには、Prometheus からアラートをルーティングするローカル Alertmanager インスタンスが含まれています。外部 Alertmanager インスタンスを追加して、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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。
   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. 変更を適用するためにファイルを保存します。新しい設定の影響を受ける Pod は自動的に再デプロイされます。

第5章 ALERTMANAGER のシークレットの設定

OpenShift Dedicated モニタリングスタックには、Prometheus からエンドポイントレシーバーにアラートをルーティングする 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。

5.1. ALERTMANAGER 設定へのシークレットの追加

openshift-user-workload-monitoring プロジェクトの **user-workload-monitoring-config** config map を編集することで、ユーザー定義プロジェクトの 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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- **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。
   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
```

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。

5.2. 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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。
   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クス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 はメトリクスを処理し、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、メトリクス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
```

c. 変更を適用するためにファイルを保存します。新しい設定の影響を受ける Pod は自動的に再デプロイされます。

第6章 モニタリングのための POD トポロジー分散制約の使用

OpenShift Dedicated 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、Pod トポロジー分散制約を使用して、ユーザー定義のモニタリング用の 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

- Kubernetes Pod Topology Spread Constraints documentation

6.1. 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用にすべての Pod に対して Pod トポロジーの拡散制約を設定し、ゾーン全体のノードに Pod レプリカをスケジュールする方法を制御できます。これにより、ワークロードが異なるデータセンターまたは階層型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、Pod の可用性が高まり、より効率的に実行されるようになります。

user-workload-monitoring-config config map を使用して、Pod を監視するための Pod トポロジーの分散制約を設定できます。

前提条件

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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** config map を編集します。

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

2. Pod トポロジーの分散制約を設定するには、**data/config.yaml** フィールドの下に次の設定を追加します。

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

```
      topologyKey: <key> 3
      whenUnsatisfiable: <value> 4
      labelSelector: 5
        <match_option>
```

1. Pod トポロジーの分散制約を設定するコンポーネントの名前を指定します。
2. **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。
3. **topologyKey** にノードラベルのキーを指定します。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、各ドメインにバランスの取れた数の Pod を配置しようとします。
4. **whenUnsatisfiable** の値を指定します。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。
5. 一致する Pod を見つけるには、**labelSelector** を指定します。このラベルセレクターに一致する Pod は、対応するトポロジードメイン内の Pod の数を決定するためにカウントされます。

Thanos Ruler の設定例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: monitoring
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            app.kubernetes.io/name: thanos-ruler
```

3. 変更を適用するためにファイルを保存します。新しい設定の影響を受ける Pod は自動的に再デプロイされます。

6.2. 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 および Thanos Ruler のログレベルを設定できます。

次のログレベルは、**user-workload-monitoring-config ConfigMap** オブジェクトの関連コンポーネントに適用できます。

- **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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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。
   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。新しい設定の影響を受ける Pod は自動的に再デプロイされます。
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

```
        - --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。

6.3. 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。

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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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. **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。

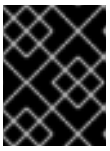
3. 変更を適用するためにファイルを保存します。新しい設定の影響を受ける Pod は自動的に再デプロイされます。

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。

第7章 ユーザー定義プロジェクトのモニタリングの無効化

関連情報

dedicated-admin として、ユーザー定義プロジェクトのモニタリングを無効にすることができます。ユーザーのワークロードモニタリングから個々のプロジェクトを除外することもできます。

7.1. ユーザー定義プロジェクトのモニタリングの無効化

デフォルトでは、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ます。ユーザー定義プロジェクトのモニタリングにビルトインモニタリングスタックを使用したくない場合は、それを無効にすることができます。

前提条件

- OpenShift Cluster Manager にログインしている。

手順

1. OpenShift Cluster Manager Hybrid Cloud Console からクラスターを選択します。
2. **Settings** タブをクリックします。
3. **Enable user workload monitoring**チェックボックスをクリックしてオプションの選択を解除し、**Save** をクリックします。
   ユーザーのワークロードモニタリングは無効になっています。Prometheus、Prometheus Operator、および Thanos Ruler コンポーネントは、**openshift-user-workload-monitoring** プロジェクトで停止されます。

7.2. 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。

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 Dedicated では、**dedicated-admin** により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することができます。このプロセスは、以下の 2 つの一般的な手順で構成されています。

- 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、別の Alertmanager インスタンスを使用します。
- 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。

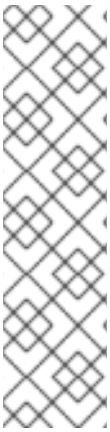
これらの手順を完了すると、開発者およびその他のユーザーはユーザー定義のプロジェクトのカスタムアラートおよびアラートルーティングを設定できます。

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

dedicated-admin として、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することができます。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、ユーザー定義の監視専用の 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。

ユーザー定義プロジェクトのアラートルーティングをユーザーが定義すると、ユーザー定義のアラート通知が **openshift-user-workload-monitoring** namespace の **alertmanager-user-workload** Pod にルーティングされます。



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用の個別の ALERTMANAGER インスタンスの有効化

OpenShift Dedicated では、ユーザー定義プロジェクト専用の Alertmanager インスタンスをデプロイして、デフォルトのプラットフォームアラートとは別のユーザー定義アラートを提供できます。このような場合は、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

- **dedicated-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。

- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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

- **alert-manager-user-workload** 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
```

出力例

```
NAME                                 READY   STATUS    RESTARTS   AGE
alertmanager-user-workload-0         6/6     Running   0          38s
alertmanager-user-workload-1         6/6     Running   0          38s
...
```

8.3. 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

- **dedicated-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。
- **user-workload-monitoring-config ConfigMap** オブジェクトが存在します。このオブジェクトは、クラスターの作成時にデフォルトで作成されます。
- 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

- 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

- 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

第9章 メトリクスの管理

メトリクスを使用すると、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 Dedicated 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。メトリクスを使用すると、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クスを定義できます。

OpenShift Dedicated では、メトリクスは **/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クス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クスを表示できます。

```
$ 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クスを収集できます。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クスを **/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** メトリクスを公開します。

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
```

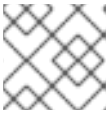
9.2.2. サービスのモニター方法の指定

サービスが公開するメトリクスを使用するには、OpenShift Dedicated 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

- **dedicated-admin** ロールまたは **monitoring-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。
- 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. **example-app-service-monitor.yaml** という名前の新しい YAML 設定ファイルを作成します。
2. **ServiceMonitor** リソースを YAML ファイルに追加します。以下の例では、**prometheus-example-monitor** という名前のサービスモニターを作成し、**ns1** namespace の **prometheus-example-app** サービスによって公開されるメトリクスを収集します。

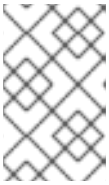
```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1 (1)
spec:
  endpoints:
  - interval: 30s
    port: web (2)
    scheme: http
```

```
  selector: 3
    matchLabels:
      app: prometheus-example-app
```

1 サービスが実行されるユーザー定義の namespace を指定します。

2 Prometheus によってスクレープされるエンドポイントポートを指定します。

3 メタデータラベルに基づいてサービスに一致するようにセレクターを設定します。



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** リソースが実行されていることを確認します。

```
$ oc -n <namespace> get servicemonitor
```

出力例

```
NAME                         AGE
prometheus-example-monitor   81m
```

9.2.3. サービスエンドポイント認証設定の例

ServiceMonitor および **PodMonitor** カスタムリソース定義 (CRD) を使用して、ユーザー定義のプロジェクト監視用のサービスエンドポイントの認証を設定できます。

次のサンプルは、**ServiceMonitor** リソースの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を示します。

9.2.3.1. ベアラートークンを使用した YAML 認証の例

以下の例は、**ns1** namespace の **example-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。

ベアラートークンシークレットの例

```
apiVersion: v1
kind: Secret
metadata:
  name: example-bearer-auth
```

```
  namespace: ns1
stringData:
  token: <authentication_token> 1
```

1 認証トークンを指定します。

以下の例は、**ServiceMonitor** CRD のベアラートークン認証設定を示しています。この例では、**example-bearer-auth** という名前の **Secret** オブジェクトを使用しています。

ベアラートークンの認証設定の例

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - authorization:
      credentials:
        key: token 1
        name: example-bearer-auth 2
    port: web
  selector:
    matchLabels:
      app: prometheus-example-app
```

1 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。

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。

重要

bearerTokenFile を使用してベアラートークンを設定しないでください。**bearerTokenFile** 設定を使用する場合、**ServiceMonitor** リソースは拒否されます。

9.2.3.2. Basic 認証用のサンプル YAML

次のサンプルは、**ns1** の **example-basic-auth** という名前の **Secret** オブジェクトの Basic 認証設定を示しています。

Basic 認証シークレットの例

```
apiVersion: v1
kind: Secret
metadata:
  name: example-basic-auth
  namespace: ns1
stringData:
  user: <basic_username> 1
  password: <basic_password> 2
```

1 認証のユーザー名を指定します。

2 認証のパスワードを指定します。

以下の例は、**ServiceMonitor** CRD の Basic 認証設定を示しています。この例では、 **example-basic-auth** という名前の **Secret** オブジェクトを使用しています。

Basic 認証の設定例

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - basicAuth:
      username:
        key: user (1)
        name: example-basic-auth (2)
      password:
        key: password (3)
        name: example-basic-auth (4)
    port: web
  selector:
    matchLabels:
      app: prometheus-example-app
```

1 指定の **Secret** オブジェクトのユーザー名が含まれるキー。

2 4 Basic 認証が含まれる **Secret** オブジェクトの名前。

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

9.2.3.3. OAuth 2.0 を使用した YAML 認証のサンプル

以下の例は、**ns1** namespace の **example-oauth2** という名前の **Secret** オブジェクトの OAuth 2.0 設定を示しています。

OAuth 2.0 シークレットの例

```
apiVersion: v1
kind: Secret
metadata:
  name: example-oauth2
  namespace: ns1
stringData:
  id: <oauth2_id> (1)
  secret: <oauth2_secret> (2)
```

1 Oauth 2.0 ID を指定します。

2 Oauth 2.0 シークレットを指定します。

以下の例は、**ServiceMonitor** CRD の OAuth 2.0 認証設定を示しています。この例では、**example-oauth2** という名前の **Secret** オブジェクトを使用します。

OAuth 2.0 認証の設定例

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - oauth2:
      clientId:
        secret:
          key: id 1
          name: example-oauth2 2
      clientSecret:
        key: secret 3
        name: example-oauth2 4
      tokenUrl: https://example.com/oauth2/token 5
    port: web
  selector:
    matchLabels:
      app: prometheus-example-app
```

1 指定された **Secret** オブジェクトの OAuth 2.0 ID が含まれるキー。

2 4 OAuth 2.0 認証情報を含む **Secret** オブジェクトの名前。

3 指定された **Secret** オブジェクトに OAuth 2.0 シークレットが含まれるキー。

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。

関連情報

- How to scrape metrics using TLS in a ServiceMonitor configuration in a user-defined project

9.3. メトリクスのクエリー

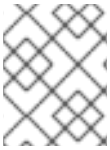
OpenShift Dedicated モニタリングダッシュボードを使用すると、Prometheus Query Language (PromQL) クエリーを実行して、プロット上に視覚化されたメトリクスを調べることができます。この機能により、クラスターの状態と、モニターしているユーザー定義のワークロードに関する情報が提供されます。

dedicated-admin として、ユーザー定義プロジェクトに関するメトリクスに対して、一度に 1 つ以上の namespace 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。

9.3.1. 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クスのクエリー

dedicated-admin またはすべてのプロジェクトの表示パーミッションを持つユーザーとして、メトリクス UI ですべてのデフォルト OpenShift Dedicated およびユーザー定義プロジェクトのメトリクスにアクセスできます。



注記

専任の管理者のみが、OpenShift Dedicated モニタリングで提供されるサードパーティーの UI にアクセスできます。

前提条件

- **dedicated-admin** ロールまたはすべてのプロジェクトの表示パーミッション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 Dedicated Web コンソールの **Administrator** パースペクティブで、**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。

9.3.2. 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。開発者は OpenShift Dedicated モニタリングで提供されるサードパーティーの UI にアクセスできません。

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が有効化されている。
- 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 Dedicated 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

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。

9.4. 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 Dedicated 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 Dedicated Monitoring がターゲットコンポーネントからメトリクス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、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

- **dedicated-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。

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 Dedicated プロジェクトとユーザー定義プロジェクトのターゲッ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
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、メトリクス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、メトリクス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。

- メトリクス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クス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 Dedicated 4 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、OpenShift Dedicated クラスター内で一連の状況が明らかであることを通知します。
- **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 Dedicated Web コンソールの **Administrator** パースペクティブおよび **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

Develper パースペクティブでは、**Project: <project_name>** リストでアクセス権のあるコア OpenShift Dedicated プロジェクトおよび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 Dedicated 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
管理者 パースペクティブでは、アラート UI の **Alerts** ページには、デフォルトの OpenShift Dedicated 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示されます。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 Dedicated 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 OpenShift Dedicated 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
管理者 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 Dedicated 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** ページには、デフォルトの OpenShift Dedicated 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 Dedicated 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 OpenShift Dedicated 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 Dedicated 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。アラートが発動され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 Dedicated モニタリングアラートをトリガーする問題の診断と解決に役立つ Cluster Monitoring Operator Runbook を参照してください。

10.4. サイレンスの管理

Administrator および **Developer** パースペクティブの両方で、OpenShift Dedicated Web コンソール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。

注記

サイレンスを作成すると、それらは Alertmanager Pod 全体に複製されます。ただし、Alertmanager の永続ストレージを設定しないと、サイレンスが失われる可能性があります。これは、たとえば、すべての Alertmanager Pod が同時に再起動した場合に発生する可能性があります。

関連情報

- Configuring persistent storage

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、**dedicated-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 Dedicated 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。

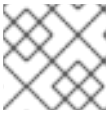
   注記

   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 Dedicated 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。



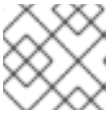
注記

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 Dedicated 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 Dedicated 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

前提条件

- クラスター管理者の場合は、**dedicated-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。

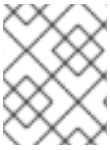
1. **Observe → Alerting → Silences** に移動します。
2. 変更するサイレンスの  をクリックして **Edit silence** を選択します。
   または、**Actions** をクリックし、サイレンスの **Silence details** ページで **Edit silence** を選択することもできます。
3. **Edit silence** ページで変更を加え、**Silence** をクリックします。これにより、既存のサイレンスが期限切れになり、更新された設定でサイレンスが作成されます。

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、**dedicated-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. ユーザー定義プロジェクトのアラートルールの管理

OpenShift Dedicated モニタリングには、一連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、デフォルトのアラートルールを表示できます。

OpenShift Dedicated 4 では、ユーザー定義プロジェクトでアラートルールの作成、表示、編集、削除ができます。

重要

ユーザー定義プロジェクトのアラートルールの管理は、OpenShift Dedicated バージョン 4.11 以降でのみ利用できます。

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 Dedicated クラスター専用に使用されます。

- 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。

10.5.1. 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。

10.5.2. 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。

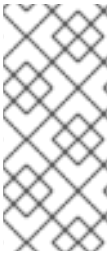
- レイテンシーを短縮し、コアプラットフォームモニタリングコンポーネントの負荷を最小限に抑えるために、ルールに **openshift.io/prometheus-rule-evaluation-scope: leaf-prometheus** ラベルを追加できます。このラベルは、**openshift-user-workload-monitoring** プロジェクトにデプロイされた Prometheus インスタンスのみにアラートルールの評価を強制し、Thanos Ruler インスタンスによる評価を防ぎます。

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。

10.5.3. 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化されている。
- 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)
      for: 1m (2)
      expr: version{job="prometheus-example-app"} == 0 (3)
      labels:
        severity: warning (4)
      annotations:
        message: This is an example alert. (5)
```

1. 作成する必要のあるアラートルールの名前。
2. アラートが発せられる前に条件が真である必要がある期間。
3. 新規ルールを定義する PromQL クエリー式。
4. アラートルールがアラートに割り当てる重大度。
5. 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 Dedicated 4 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。

10.5.4. 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化されている。
- 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
```

10.5.5. 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

dedicated-admin として、コア OpenShift Dedicated プロジェクトとユーザー定義プロジェクトのアラートルールを 1 つのビューにまとめてリストできます。

前提条件

- **dedicated-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。

10.5.6. 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化されている。
- 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。

10.6. 外部システムへの通知の送信

OpenShift Dedicated 4 では、アラートの起動をアラート UI で表示できます。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 Dedicated を設定できます。

- PagerDuty
- Webhook
- Email
- Slack
- Microsoft Teams

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 Dedicated モニタリングには、継続的に発生するウォッチドッグ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。

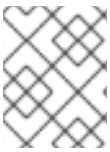
10.6.1. デフォルトのプラットフォームアラートとユーザー定義アラートに異なるアラートレシーバーを設定する

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アラートに異なるアラートレシーバーを設定して、次の結果を確実に得ることができます。

- すべてのデフォルトのプラットフォームアラートは、これらのアラートを担当するチームが所有する受信機に送信されます。
- すべてのユーザー定義アラートは別の受信者に送信されるため、チームはプラットフォームアラートにのみ集中できます。

これを実現するには、Cluster Monitoring Operator によってすべてのプラットフォームアラートに追加される **openshift_io_alert_source="platform"** ラベルを使用します。

- デフォルトのプラットフォームアラートを一致させるには、**openshift_io_alert_source="platform"** マッチャーを使用します。
- ユーザー定義のアラートを一致させるには、**openshift_io_alert_source!="platform"** または **'openshift_io_alert_source=""'** マッチャーを使用します。



注記

ユーザー定義アラート専用の Alertmanager の別のインスタンスを有効にしている場合、この設定は適用されません。

10.6.2. 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

- アラートルーティングがユーザー定義プロジェクトに対して有効になりました。
- 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。

10.7. 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

- **dedicated-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. 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は、Alertmanager configuration を参照してください。

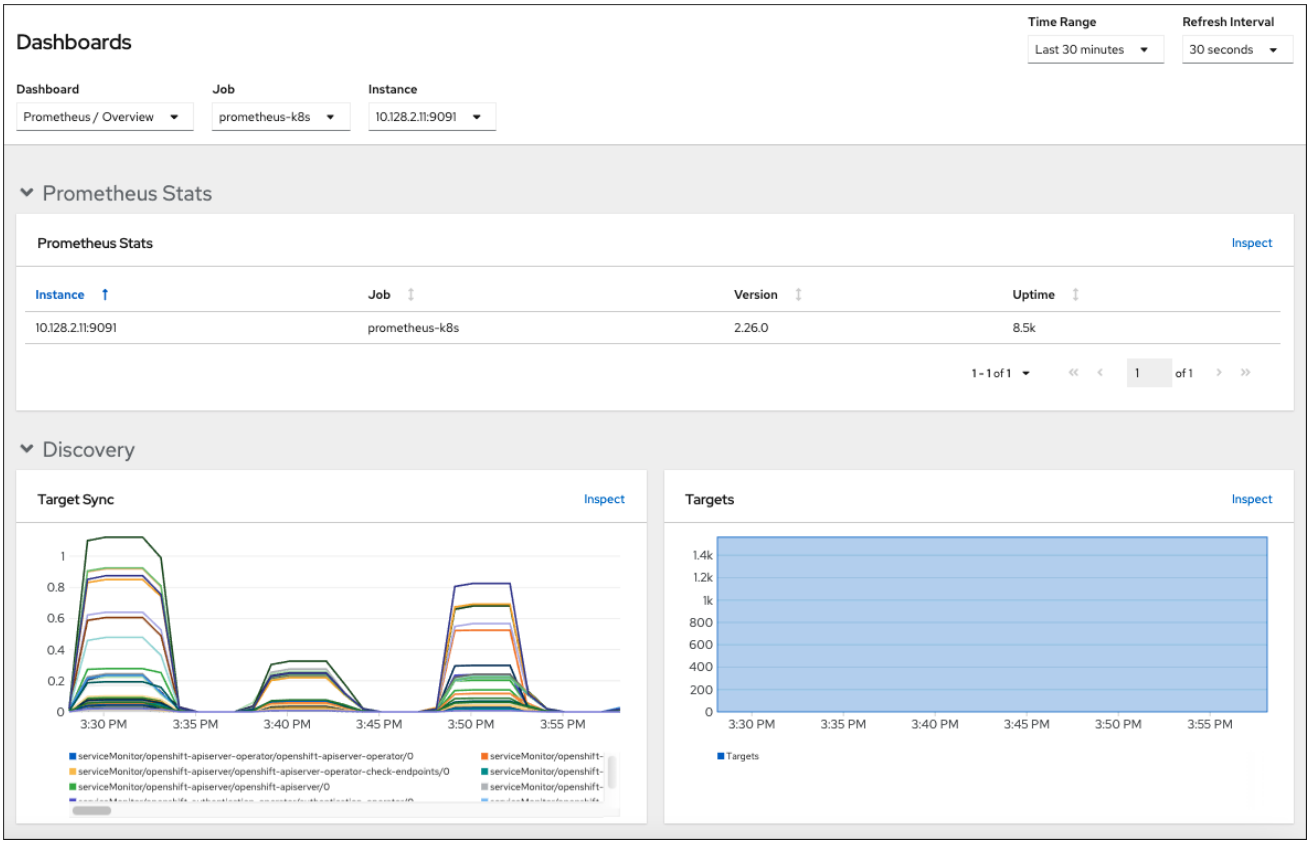
第11章 モニタリングダッシュボードの確認

OpenShift Dedicated は、ユーザー定義プロジェクトの状態を理解するのに役立つモニタリングダッシュボードを提供します。

Administrator パースペクティブを使用して、次の項目を含むコア OpenShift Dedicated 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

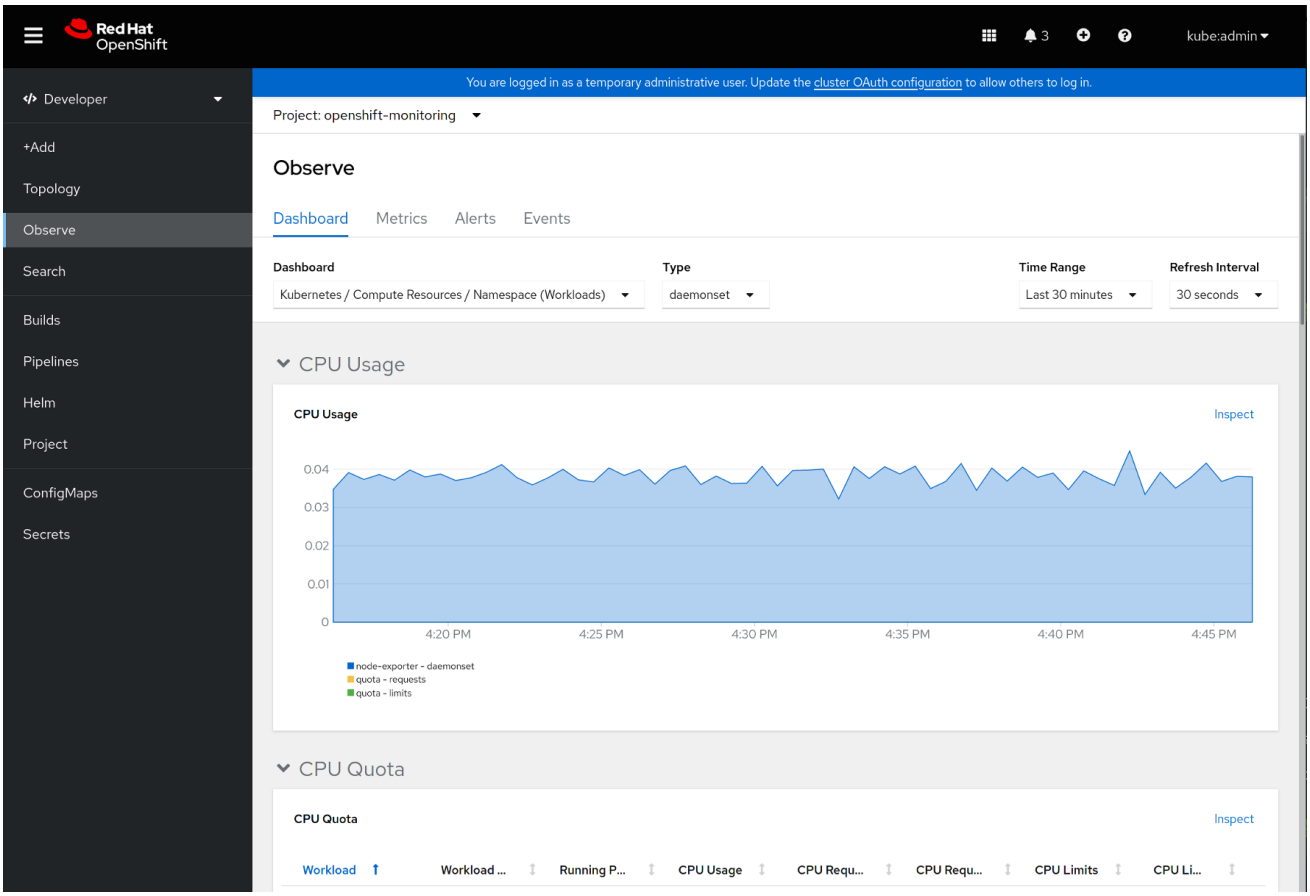


Developer パースペクティブを使用して、選択されたプロジェクトの以下のアプリケーションメトリクスを提供する Kubernetes コンピュートリソースダッシュボードにアクセスします。

- 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

開発者 パースペクティブでは、一度に 1 つのプロジェクトのダッシュボードのみを表示できます。

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、コア OpenShift Dedicated クラスター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

- **dedicated-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。

手順

1. OpenShift Dedicated 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 Dedicated 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クスを表示します。

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。

第12章 CLI を使用した API のモニタリング

OpenShift Dedicated 4 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クスデータを取得、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。

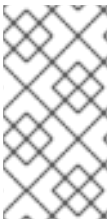
これらの問題を回避するには、以下の推奨事項に従ってください。

- 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クス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 Dedicated 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クスのクエリー

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クスを収集できます。これを行うには、OpenShift Dedicated ルートを介してクラスターの Prometheus /**federate** エンドポイントにアクセスします。

重要

メトリクス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クス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クス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** ルートからメトリクスをクエリーします。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クスへのアクセス

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

- 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

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. ユーザー定義プロジェクトのメトリクスが利用できない理由の判別

ユーザー定義プロジェクトのモニタリング時にメトリクスが表示されない場合は、以下の手順を実行して問題のトラブルシューティングを実行します。

手順

1. メトリクス名に対してクエリーを実行し、プロジェクトが正しいことを確認します。
   a. 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。
   b. **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。
   c. **Select query** 一覧からクエリーを選択するか、**Show PromQL** を選択してカスタム PromQL クエリーを実行します。
   メトリクスはグラフに表示されます。

   クエリーはプロジェクトごとに実行される必要があります。表示されるメトリクスは、選択したプロジェクトに関連するメトリクスです。

2. メトリクスが必要な Pod がアクティブにメトリクスを提供していることを確認します。以下の **oc exec** コマンドを Pod で実行し、**podIP**、**port**、および **/metrics** をターゲットにします。

```
$ oc exec <sample_pod> -n <sample_namespace> -- curl <target_pod_IP>:<port>/metrics
```

注記

curl がインストールされている Pod でコマンドを実行する必要があります。

以下の出力例は、有効なバージョンのメトリクスを含む結果を示しています。

出力例

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
# HELP version Version information about this binary-- --:--:-- --:--:--     0
# TYPE version gauge
version{version="v0.1.0"} 1
100   102  100   102    0     0  51000      0 --:--:-- --:--:-- --:--:-- 51000
```

無効な出力は、対応するアプリケーションに問題があることを示しています。

3. **PodMonitor** CRD を使用している場合は、**PodMonitor** CRD がラベル一致を使用して適切な Pod を参照するよう設定されていることを確認します。詳細は、Prometheus Operator のドキュメントを参照してください。

4. **ServiceMonitor** CRD を使用し、Pod の **/metrics** エンドポイントがメトリクスデータを表示している場合は、以下の手順を実行して設定を確認します。

a. サービスが正しい **/metrics** エンドポイントを参照していることを確認します。この出力のサービス **labels** は、後続の手順でサービスが定義するサービスモニターの **labels** と **/metrics** エンドポイントと一致する必要があります。

```
$ oc get service
```

出力例

```
apiVersion: v1
kind: Service (1)
metadata:
  labels: (2)
    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
```

(1) これがサービス API であることを指定します。

(2) このサービスに使用されるラベルを指定します。

b. **serviceIP**、**port**、および **/metrics** エンドポイントをクエリーし、以前に Pod で実行した **curl** コマンドと同じメトリクスがあるかどうかを確認します。

i. 以下のコマンドを実行してサービス IP を見つけます。

```
$ oc get service -n <target_namespace>
```

ii. **/metrics** エンドポイントをクエリーします。

```
$ oc exec <sample_pod> -n <sample_namespace> -- curl <service_IP>:
<port>/metrics
```

以下の例では、有効なメトリクスが返されます。

出力例

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   102  100   102    0     0  51000      0 --:--:-- --:--:-- --:--:--   99k
# HELP version Version information about this binary
# TYPE version gauge
version{version="v0.1.0"} 1
```

c. ラベルのマッチングを使用して、**ServiceMonitor** オブジェクトが必要なサービスを参照す

るように設定されていることを確認します。これを実行するには、**oc get service** 出力の **Service** オブジェクトを **oc get servicemonitor** 出力の **ServiceMonitor** オブジェクトと比較します。メトリクスを表示するには、ラベルが一致している必要があります。
たとえば、直前の手順の **Service** オブジェクトに **app: prometheus-example-app** ラベルがあり、**ServiceMonitor** オブジェクトに同じ **app: prometheus-example-app** 一致ラベルがある点に注意してください。

5. すべて有効になっていても、メトリクスが利用できない場合は、サポートチームにお問い合わせください。

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クス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。
- 収集されている **スクレイプサンプルの数を確認** します。
- ユーザー定義メトリクス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。

- ユーザー定義のプロジェクト全体で **スクレイピングできるサンプルの数に制限を適用** 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

- **dedicated-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 → 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

- 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。
   - **メトリクス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クス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。
   - **メトリクスがコア OpenShift Dedicated プロジェクトに関連している場合は**、Red Hat Customer Portal で Red Hat サポートケースを作成します。
4. 以下の手順に従い、**dedicated-admin** としてログインし、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},
...
```

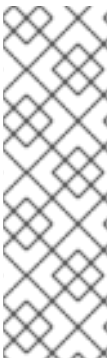
関連情報

- 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

13.3. PROMETHEUS に対する KUBEPERSISTENTVOLUMEFILLINGUP アラートの解決

クラスター管理者は、Prometheus に対してトリガーされている **KubePersistentVolumeFillingUp** アラートを解決できます。

openshift-monitoring プロジェクトの **prometheus-k8s-*** Pod によって要求された永続ボリューム (PV) の合計残り容量が 3% 未満になると、重大アラートが発生します。これにより、Prometheus の動作異常が発生する可能性があります。



注記

KubePersistentVolumeFillingUp アラートは 2 つあります。

- **重大アラート**: マウントされた PV の合計残り容量が 3% 未満になると、**severity="critical"** ラベルの付いたアラートがトリガーされます。
- **警告アラート**: マウントされた PV の合計空き容量が 15% 未満になり、4 日以内にいっぱいになると予想される場合、**severity="warning"** ラベルの付いたアラートがトリガーされます。

この問題に対処するには、Prometheus 時系列データベース (TSDB) のブロックを削除して、PV 用のスペースを増やすことができます。

前提条件

- **dedicated-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. 次のコマンドを実行して、すべての TSDB ブロックのサイズを古いものから新しいものの順にリスト表示します。

```
$ oc debug <prometheus_k8s_pod_name> -n openshift-monitoring \ (1)
-c prometheus --image=$(oc get po -n openshift-monitoring <prometheus_k8s_pod_name> \
(2)
-o jsonpath='{.spec.containers[?(@.name=="prometheus")].image}') \
-- sh -c 'cd /prometheus/;du -hs $(ls -dt */ | grep -Eo "[0-9|A-Z]{26}")'
```

(1) (2) **<prometheus_k8s_pod_name>** は、**KubePersistentVolumeFillingUp** アラートの説明に記載されている Pod に置き換えます。

出力例

```
308M    01HVKMPKQWZYWS8WVDAYQHNMW6
52M     01HVK64DTDA81799TBR9QDECEZ
102M    01HVK64DS7TRZRWF2756KHST5X
140M    01HVJS59K11FBVAPVY57K88Z11
90M     01HVH2A5Z58SKT810EM6B9AT50
152M    01HV8ZDVQMX41MKCN84S32RRZ1
354M    01HV6Q2N26BK63G4RYTST71FBF
```

```
156M    01HV664H9J9Z1FTZD73RD1563E
216M    01HTHXB60A7F239HN7S2TENPNS
104M    01HTHMGRXGS0WXA3WATRXHR36B
```

2. 削除できるブロックとその数を特定し、ブロックを削除します。次のコマンド例は、**prometheus-k8s-0** Pod から最も古い 3 つの Prometheus TSDB ブロックを削除します。

```
$ oc debug prometheus-k8s-0 -n openshift-monitoring \
-c prometheus --image=$(oc get po -n openshift-monitoring prometheus-k8s-0 \
-o jsonpath='{.spec.containers[?(@.name=="prometheus")].image}') \
-- sh -c 'ls -latr /prometheus/ | egrep -o "[0-9|A-Z]{26}" | head -3 | \
while read BLOCK; do rm -r /prometheus/$BLOCK; done'
```

3. 次のコマンドを実行して、マウントされた PV の使用状況を確認し、十分な空き容量があることを確認します。

```
$ oc debug <prometheus_k8s_pod_name> -n openshift-monitoring \ 1
--image=$(oc get po -n openshift-monitoring <prometheus_k8s_pod_name> \ 2
-o jsonpath='{.spec.containers[?(@.name=="prometheus")].image}') -- df -h /prometheus/
```

1 2 **<prometheus_k8s_pod_name>** は、**KubePersistentVolumeFillingUp** アラートの説明に記載されている Pod に置き換えます。

次の出力例は、**prometheus-k8s-0** Pod によって要求されるマウントされた PV に、63% の空き容量が残っていることを示しています。

出力例

```
Starting pod/prometheus-k8s-0-debug-j82w4 ...
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p4  40G   15G  40G  37% /prometheus

Removing debug pod ...
```

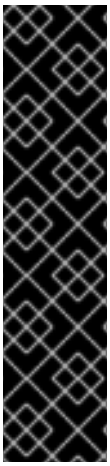
第14章 CLUSTER MONITORING OPERATOR の CONFIG MAP 参照

14.1. CLUSTER MONITORING OPERATOR 設定リファレンス

OpenShift Dedicated 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。
Toleration	[]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。
Toleration	[]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** config map 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。
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。
prometheusOperator	*PrometheusOperatorConfig	**PrometheusOperatorConfig** は、Prometheus Operator 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
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. KUBESTATEMETRICSCONFIG

14.6.1. 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。
Toleration	[]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。

14.7. METRICSSERVERCONFIG

14.7.1. 説明

MetricsServerConfig リソースは、Metrics Server 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	Metrics Server インスタンスで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。
Toleration	[]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。

14.8. MONITORINGPLUGINCONFIG

14.8.1. 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。

プロパティー	型	説明
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。
Toleration	[]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. NODEEXPORTERCOLLECTORBUDDYINFOCONFIG

14.9.1. 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。

14.10. NODEEXPORTERCOLLECTORCONFIG

14.10.1. 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** メトリクス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。このメトリクスは、**/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。

14.11. NODEEXPORTERCOLLECTORCPUFREQCONFIG

14.11.1. 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。

14.12. NODEEXPORTERCOLLECTORKSMDCONFIG

14.12.1. 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。

14.13. NODEEXPORTERCOLLECTORMOUNTSTATSCONFIG

14.13.1. 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。

14.14. NODEEXPORTERCOLLECTORNETCLASSCONFIG

14.14.1. 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、次のメトリクス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。

14.15. NODEEXPORTERCOLLECTORNETDEVCONFIG

14.15.1. 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。

14.16. NODEEXPORTERCOLLECTORPROCESSESCONFIG

14.16.1. 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**)。メトリクス **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。

14.17. NODEEXPORTERCOLLECTORSYSTEMDCONFIG

14.17.1. 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** メトリクスを生成するために使用されます。ただし、このメトリクス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** メトリクスは、**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。

14.18. NODEEXPORTERCOLLECTORTCPSTATCONFIG

14.18.1. 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。

14.19. NODEEXPORTERCONFIG

14.19.1. 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。

14.20. OPENSHIFTSTATEMETRICSCONFIG

14.20.1. 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。
Toleration	[]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。

14.21. PROMETHEUSK8SCONFIG

14.21.1. 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。
Toleration	[]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。

14.22. PROMETHEUSOPERATORCONFIG

14.22.1. 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。
Toleration	[]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。

14.23. PROMETHEUSOPERATORADMISSIONWEBHOOKCONFIG

14.23.1. 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。

14.24. PROMETHEUSRESTRICTEDCONFIG

14.24.1. 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。
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。

プロパティー	型	説明
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。
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。

プロパティー	型	説明
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。
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。
Toleration	[]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。

プロパティー	型	説明
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。

14.25. REMOTEWRITESPEC

14.25.1. 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。

14.25.2. 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。
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。

プロパティー	型	説明
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。
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。

14.26. TLSCONFIG

14.26.1. 説明

TLSConfig リソースは、TLS 接続の設定を設定します。

14.26.2. 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。
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。

14.27. TELEMETERCLIENTCONFIG

14.27.1. 説明

TelemeterClientConfig は、Telemeter Client コンポーネントの設定を定義します。

14.27.2. 必須

- **nodeSelector**
- **Toleration**

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。

プロパティー	型	説明
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。
Toleration	[]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。

14.28. THANOSQUERIERCONFIG

14.28.1. 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。
Toleration	[]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。

14.29. THANOSRULERCONFIG

14.29.1. 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。
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。
Toleration	[]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。

14.30. USERWORKLOADCONFIGURATION

14.30.1. 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。**UserWorkloadConfiguration** 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。
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。