OpenShift Container Platform 4.3

Jaeger

Jaeger のインストール、使用法、およびリリースノート

Last Updated: 2020-11-13

OpenShift Container Platform 4.3 Jaeger

Jaeger のインストール、使用法、およびリリースノート

法律上の通知

概要

本書では、OpenShift Container Platform で Jaeger を使用する方法について説明します。

目次

第1章 JAEGER リリースノート

1.1. JAEGER の概要

サービスの所有者は、Jaeger を使用してサービスをインストルメント化し、サービスアーキテクチャーに関する洞察を得ることができます。Jaeger は、最新のクラウドネイティブ、マイクロサービスベースのアプリケーションにおいてコンポーネント間の対話のモニタリング、ネットワークプロファイリングおよびトラブルシューティングに使用できる、オープンソースの分散トレースプラットフォームです。

Jaeger を使用すると、以下の機能を実行できます。

- 分散トランザクションの監視
- パフォーマンスとレイテンシーの最適化
- 根本原因分析の実行

Jaeger は特定のベンダーに依存しない OpenTracing API およびインストルメンテーションに基づいています。

1.2. サポート

このドキュメントで説明する手順に関連した問題が発生する場合は、Red Hat カスタマーポータル にアクセスしてください。カスタマーポータルから、以下を行うことができます。

- Red Hat 製品に関する技術サポート記事の Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。

注記

サポートケースを送信する際、Red Hat サポートのトラブルシューティングに役立つ以下の情報を提供していただくことをお勧めします。

  - **oc adm must-gather** コマンドを使用して収集されるデータ
  - 一意のクラスター ID。**(?) Help → Open Support Case** に移動すると、ケースの送信時にクラスター ID が自動入力されます。

- 他の製品ドキュメントへのアクセス。

本書の改善が提案されている場合や、エラーが見つかった場合は、**Documentation** コンポーネントの **OpenShift Container Platform** 製品に対して、Bugzilla レポート を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

1.2.1. OpenShift Jaeger 1.17.4 の新機能

OpenShift Jaeger の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2. OpenShift Jaeger 1.17.3 の新機能

OpenShift Jaeger の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.3. OpenShift Jaeger 1.17.2 の新機能

OpenShift Jaeger の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.4. OpenShift Jaeger 1.17.1 の新機能

本リリースの OpenShift Jaeger では、Red Hat OpenShift Service Mesh のコンポーネントとしてではなく、Jaeger をスタンドアロンソリューションとしてインストールするためのサポートが追加されました。

1.3. JAEGER の既知の問題

Jaeger には、以下の制限があります。

- Kafka パブリッシャーは Jaeger の一部として組み込まれていますが、サポートされていません。
- Apache Spark はサポートされていません。
- セルフプロビジョニングされた Elasticsearch インスタンスのみがサポートされます。本リリースでは、外部 Elasticsearch インスタンスはサポートされません。

Jaeger の既知の問題は以下のとおりです。

- TRACING-1166 現時点で、Jaeger ストリーミングストラテジーを非接続環境で使用することはできません。Kafka クラスターがプロビジョニングされる際に、以下のエラーが出されます: **Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076**
- Trace-809 Jaeger Ingester には Kafka 2.3 との互換性がありません。Jaeger Ingester のインスタンスが複数あり、十分なトラフィックがある場合、リバランスメッセージがログに継続的に生成されます。これは、Kafka 2.3.1 で修正された Kafka 2.3 のリグレッションによって生じます。詳細は、Jaegertracing-1819 を参照してください。

第2章 JAEGER アーキテクチャー

2.1. JAEGER アーキテクチャー

ユーザーがアプリケーションでアクションを実行するたびに、応答を生成するために多数の異なるサービスに参加を要求する可能性のあるアーキテクチャーによって要求が実行されます。Jaeger を使用すると、分散トレースを実行できます。これは、アプリケーションを構成するさまざまなマイクロサービスによる要求のパスを記録します。

分散トレースは、さまざまな作業ユニットの情報を連携させるために使用される技術です。これは、分散トランザクションでのイベントのチェーン全体を理解するために、通常さまざまなプロセスまたはホストで実行されます。分散トレースを使用すると、開発者は大規模なマイクロサービスアーキテクチャーで呼び出しフローを可視化できます。これは、シリアル化、並行処理、およびレイテンシーのソースについての理解にも役立ちます。

Jaeger はマイクロサービスのスタック全体での個々の要求の実行を記録し、トレースとして表示します。**トレース** とは、システムにおけるデータ/実行パスです。エンドツーエンドトレースは、1 つ以上のスパンで構成されます。

スパンは、オペレーション名、オペレーションの開始時間および期間を持ち、タグやログを持つ可能性もある Jaeger の作業の論理単位を表しています。スパンは因果関係をモデル化するためにネスト化され、順序付けられます。

2.1.1. Jaeger の概要

サービスの所有者は、Jaeger を使用してサービスをインストルメント化し、サービスアーキテクチャーに関する洞察を得ることができます。Jaeger は、最新のクラウドネイティブ、マイクロサービスベースのアプリケーションにおいてコンポーネント間の対話のモニタリング、ネットワークプロファイリングおよびトラブルシューティングに使用できる、オープンソースの分散トレースプラットフォームです。

Jaeger を使用すると、以下の機能を実行できます。

- 分散トランザクションの監視
- パフォーマンスとレイテンシーの最適化
- 根本原因分析の実行

Jaeger は特定のベンダーに依存しない OpenTracing API およびインストルメンテーションに基づいています。

2.1.2. Jaeger の機能

Jaeger のトレース機能には以下の機能が含まれます。

- Kiali との統合: 適切に設定されている場合、Kiali コンソールから Jaeger データを表示できます。
- 高いスケーラビリティー: Jaeger バックエンドは、単一障害点がなく、ビジネスニーズに合わせてスケーリングできるように設計されています。
- 分散コンテキストの伝播: さまざまなコンポーネントからのデータをつなぎ、完全なエンドツーエンドトレースを作成します。

- Zipkin との後方互換性: Jaeger には、Zipkin のドロップイン置き換えで使用できるようにする API がありますが、本リリースでは、Red Hat は Zipkin の互換性をサポートしていません。

2.1.3. Jaeger アーキテクチャー

Jaeger は、複数のコンポーネントで構成されており、トレースデータを収集し、保存し、表示するためにそれらが連携します。

- **Jaeger Client** (Tracer、Reporter、インストルメント化されたアプリケーション、クライアントライブラリー): Jaeger クライアントは、OpenTracing API の言語固有の実装です。それらは、手動または (Camel (Fuse)、Spring Boot (RHOAR)、MicroProfile (RHOAR/Thorntail)、Wildfly (EAP)、その他 OpenTracing にすでに統合されているものを含む) 各種の既存オープンソースフレームワークを使用して、分散トレース用にアプリケーションをインストルメント化するために使用できます。

- **Jaeger Agent** (Server Queue、Processor Worker): Jaeger エージェントは、User Datagram Protocol (UDP) で送信されるスパンをリッスンするネットワークデーモンで、コレクターにバッチ処理や送信を実行します。このエージェントは、インストルメント化されたアプリケーションと同じホストに配置されることが意図されています。これは通常、Kubernetes などのコンテナー環境にサイドカーコンテナーを配置することによって実行されます。

- **Jaeger Collector** (Queue、Worker): エージェントと同様に、コレクターはスパンを受信でき、これらを処理するために内部キューに配置できます。これにより、コレクターはスパンがストレージに移動するまで待機せずに、クライアント/エージェントにすぐに戻ることができます。

- **Storage** (Data Store): コレクターには永続ストレージのバックエンドが必要です。Jaeger には、スパンストレージ用のプラグ可能なメカニズムがあります。本リリースでは、サポートされているストレージは Elasticsearch のみであることに注意してください。

- **Query** (Query Service): Query は、ストレージからトレースを取得するサービスです。

- **Ingester** (Ingester Service): Jaeger は Apache Kafka をコレクターと実際のバッキングストレージ (Elasticsearch) 間のバッファーとして使用できます。Ingester は、Kafka からデータを読み取り、別のストレージバックエンド (Elasticsearch) に書き込むサービスです。

- **Jaeger Console**: Jaeger は、分散トレースデータを視覚化できるユーザーインターフェースを提供します。検索ページで、トレースを検索し、個別のトレースを構成するスパンの詳細を確認することができます。

第3章 JAEGER のインストール

3.1. JAEGER のインストール

Jaeger を OpenShift Container Platform にインストールするには、以下のいずれかの方法を使用できます。

- Jaeger は、Red Hat OpenShift Service Mesh の一部としてインストールできます。Jaeger はデフォルトでサービスメッシュインストールに含まれています。サービスメッシュの一部として Jaeger をインストールするには、「Red Hat Service Mesh のインストール」の手順に従います。
- サービスメッシュをインストールする必要がない場合は、Jaeger Operator を使用して Jaeger の Red Hat ビルド自体をインストールできます。サービスメッシュなしで Jaeger をインストールするには、以下の手順を実行します。

3.1.1. 前提条件

OpenShift Jaeger をインストールするには、インストールアクティビティーを確認し、前提条件を満たしていることを確認してください。

- お使いの Red Hat アカウントに有効な OpenShift Container Platform サブスクリプションを用意します。サブスクリプションをお持ちでない場合は、営業担当者にお問い合わせください。
- 「OpenShift Container Platform 4.3 の概要」を確認します。
- OpenShift Container Platform 4.3 をインストールします。
  - AWS への OpenShift Container Platform 4.3 のインストール
  - ユーザーによってプロビジョニングされた AWS への OpenShift Container Platform 4.3 のインストール
  - ベアメタルへの OpenShift Container Platform 4.3 のインストール
  - vSphere への OpenShift Container Platform 4.3 のインストール
- OpenShift Container Platform バージョンに一致する OpenShift Container Platform コマンドラインユーティリティーのバージョン (**oc** クライアントツール) をインストールし、これをパスに追加します。
- **cluster-admin** ロールを持つアカウントが必要です。

3.1.2. Jaeger インストールの概要

OpenShift Jaeger のインストール手順は以下のとおりです。

- 本書を確認し、デプロイメントストラテジーを確認します。
- デプロイメントストラテジーに永続ストレージが必要な場合は、OperatorHub を使用して Elasticsearch Operator をインストールします。
- OperatorHub を使用して Jaeger Operator をインストールします。
- Jaeger YAML ファイルをデプロイメントストラテジーをサポートするように変更します。

- Jaeger の 1 つ以上のインスタンスを OpenShift Container Platform 環境にデプロイします。

3.1.3. Elasticsearch Operator のインストール

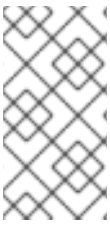
デフォルトの Jaeger デプロイメントはインメモリーストレージを使用します。それは、Jaeger の評価、デモの提供、またはテスト環境での Jaeger の使用を目的としてすぐにインストールできるように設計されているためです。実稼働環境で Jaeger を使用する予定がある場合、永続ストレージのオプション (この場合は Elasticsearch) をインストールする必要があります。

前提条件

- OpenShift Container Platform Web コンソールへのアクセスが可能です。
- **cluster-admin** ロールを持つアカウントが必要です。

警告

Operator のコミュニティーバージョンはインストールしないでください。コミュニティー Operator はサポートされていません。



注記

OpenShift クラスターロギングの一部として Elasticsearch Operator がすでにインストールされている場合、Elasticsearch Operator を再びインストールする必要はありません。Jaeger Operator はインストールされた Elasticsearch Operator を使用して Elasticsearch インスタンスを作成します。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. **Operators → OperatorHub** に移動します。
3. **Elasticsearch** とフィルターボックスに入力して、Elasticsearch Operator を検索します。
4. Red Hat が提供する **Elasticsearch Operator** をクリックし、Operator についての情報を表示します。
5. **Install** をクリックします。
6. **Create Operator Subscription** ページで **A specific namespace on the cluster**オプションを選択し、メニューから **openshift-operators-redhat** を選択します。
7. OpenShift Container Platform インストールに一致する **Update Channel** を選択します。たとえば、OpenShift Container Platform バージョン 4.5 にインストールしている場合は、4.5 更新チャネルを選択します。
8. **Automatic** Approval Strategy を選択します。

注記

手動の承認ストラテジーには、Operator のインストールおよびサブスクリプションプロセスを承認するための適切な認証情報を持つユーザーが必要です。

9. **Subscribe** をクリックします。
10. **Installed Operators** ページで、**openshift-operators-redhat** プロジェクトを選択します。Elasticsearch Operator が「InstallSucceeded」のステータスを表示するまで待機してから続行します。

3.1.4. Jaeger Operator のインストール

Jaeger をインストールするには、OperatorHub を使用して Jaeger Operator をインストールします。

デフォルトで、Operator は **openshift-operators** プロジェクトにインストールされます。

前提条件

- OpenShift Container Platform Web コンソールへのアクセスが可能です。
- **cluster-admin** ロールを持つアカウントが必要です。
- 永続ストレージが必要な場合、Jaeger Operator をインストールする前に Elasticsearch Operator もインストールする必要があります。

警告

Operator のコミュニティーバージョンはインストールしないでください。コミュニティー Operator はサポートされていません。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. **Operators → OperatorHub** に移動します。
3. **Jaeger** とフィルターに入力して、Jaeger Operator を検索します。
4. Red Hat が提供する **Jaeger Operator** をクリックし、Operator についての情報を表示します。
5. **Install** をクリックします。
6. **Create Operator Subscription** ページで、**All namespaces on the cluster (default)**を選択します。これにより、Operator がデフォルトの **openshift-operators** プロジェクトにインストールされ、Operator はクラスター内のすべてのプロジェクトで利用可能になります。
7. **stable** Update Channel を選択します。これにより、新しいバージョンがリリースされると Jaeger が自動的に更新されます。**1.17-stable** などのメンテナンスチャネルを選択すると、その

バージョンのサポートサイクルの期間、バグ修正およびセキュリティーパッチが送信されます。

- Approval Strategy を選択します。**Automatic** または **Manual** の更新を選択できます。インストールされた Operator について自動更新を選択する場合、Operator の新規バージョンが利用可能になると、Operator Lifecycle Manager (OLM) は人の介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。手動更新を選択する場合、Operator の新規バージョンが利用可能になると、OLM は更新要求を作成します。クラスター管理者は、Operator が新規バージョンに更新されるように更新要求を手動で承認する必要があります。

注記

手動の承認ストラテジーには、Operator のインストールおよびサブスクリプションプロセスを承認するための適切な認証情報を持つユーザーが必要です。

8. **Subscribe** をクリックします。
9. **Subscription Overview** ページで、**openshift-operators** プロジェクトを選択します。Jaeger Operator に「InstallSucceeded」のステータスが表示されるまで待機してから続行します。

3.2. JAEGER の設定およびデプロイ

Jaeger Operator には、Jaeger リソースのアーキテクチャーおよび設定を定義するカスタムリソース定義 (CRD) ファイルが含まれます。デフォルト設定をインストールするか、またはビジネス要件に合わせてファイルを変更することができます。

Jaeger には事前に定義されたデプロイメントストラテジーがあります。カスタムリソースファイルでデプロイメントストラテジーを指定します。Jaeger インスタンスの作成時に、Operator はこの設定ファイルを使用してデプロイメントに必要なオブジェクトを作成します。

デプロイメントストラテジーを表示する Jaeger カスタムリソースファイル

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production (1)
```

(1) Jaeger Operator は現時点で以下のデプロイメントストラテジーをサポートします。

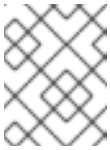
- **allInOne** (デフォルト): このストラテジーは、開発、テストおよびデモの目的で使用されることが意図されています。主なバックエンドコンポーネントである Agent、Collector、および Query サービスはすべて、インメモリーストレージを使用するように (デフォルトで) 設定された単一の実行可能ファイルにパッケージ化されます。

注記

インメモリーストレージには永続性がありません。つまり、Jaeger インスタンスがシャットダウンするか、再起動するか、または置き換えられると、トレースデータが失われます。各 Pod には独自のメモリーがあるため、インメモリーストレージはスケーリングできません。永続ストレージの場合、デフォルトのストレージとして Elasticsearch を使用する **production** または **streaming** ストラテジーを使用する必要があります。

- **production**: production ストラテジーは、実稼働環境向けのストラテジーであり、トレースデータの長期の保存が重要となり、より拡張性および高可用性のあるアーキテクチャーも必要になります。そのため、バックエンドコンポーネントはそれぞれ別々にデプロイされます。エージェントは、インストルメント化されたアプリケーションのサイドカーまたは Deamonset として挿入できます。Query および Collector サービスは、サポートされているストレージタイプ (現時点では Elasticsearch) で設定されます。これらの各コンポーネントの複数のインスタンスは、パフォーマンスと回復性を確保するために、必要に応じてプロビジョニングできます。

- **streaming**: streaming ストラテジーは、Collector とバックエンドストレージ (Elasticsearch) 間に効果的に配置されるストリーミング機能を提供することで、production ストラテジーを増強する目的で設計されています。これにより、負荷の高い状況でバックエンドストレージに加わる圧力を軽減し、他のトレース処理後の機能がストリーミングプラットフォーム (AMQ Streams/ Kafka) から直接リアルタイムのスパンデータを利用できるようにします。



注記

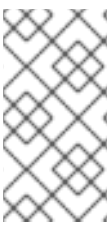
ストリーミングストラテジーには、AMQ Streams 用の追加の Red Hat サブスクリプションが必要です。

注記

サービスメッシュの一部としてか、またはスタンドアロンのコンポーネントとして Jaeger をインストールするには 2 つの方法があります。Jaeger を Red Hat OpenShift Service Mesh の一部としてインストールしている場合、ServiceMeshControlPlane の一部として Jaeger を設定し、デプロイする必要があります。

3.2.1. Web コンソールからのデフォルト Jaeger ストラテジーのデプロイ

カスタムリソース定義 (CRD) は、Jaeger のインスタンスをデプロイする際に使用される設定を定義します。Jaeger のデフォルト CR は **jaeger-all-in-one-inmemory** という名前で、デフォルトの OpenShift Container Platform インストールに正常にインストールできるように最小リソースで設定されます。このデフォルト設定を使用して、**AllInOne** デプロイメントストラテジーを使用する Jaeger インスタンスを作成するか、または独自のカスタムリソースファイルを定義できます。



注記

インメモリーストレージには永続性がありません。つまり、Jaeger Pod がシャットダウンするか、再起動するか、または置き換えられると、トレースデータが失われます。永続ストレージの場合、デフォルトのストレージとして Elasticsearch を使用する **production** または **streaming** ストラテジーを使用する必要があります。

前提条件

- Jaeger Operator がインストールされている必要があります。
- Jaeger インストールのカスタマイズ方法についての手順を確認します。
- **cluster-admin** ロールを持つアカウントが必要です。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. 新規プロジェクト (例: **jaeger-system**) を作成します。
   a. **Home → Projects** に移動します。
   b. **Create Project** をクリックします。
   c. **Name** フィールドに **jaeger-system** を入力します。
   d. **Create** をクリックします。
3. **Operators → Installed Operators** に移動します。
4. 必要な場合は、Project メニューから **jaeger-system** を選択します。Operator が新規プロジェクトにコピーされるまでに数分待機する必要がある場合があります。
5. OpenShift Jaeger Operator をクリックします。**Overview** タブの **Provided APIs** で、Operator は単一リンクを提供します。
6. **Jaeger** で **Create Instance** をクリックします。
7. **Create Jaeger** ページで、デフォルトを使用してインストールするには、 **Create** をクリックして Jaeger インスタンスを作成します。
8. **Jaegers** ページで、Jaeger インスタンスの名前 (例: **jaeger-all-in-one-inmemory**) をクリックします。
9. **Jaeger Details** ページで、**Resources** タブをクリックします。Pod のステータスが「Running」になるまで待機してから続行します。

3.2.1.1. CLI からのデフォルト Jaeger のデプロイ

以下の手順に従って、コマンドラインから Jaeger のインスタンスを作成します。

前提条件

- OpenShift Jaeger Operator がインストールされ、検証済みです。
- OpenShift Container Platform バージョンに一致する OpenShift CLI (**oc**) へのアクセスが可能です。
- **cluster-admin** ロールを持つアカウントが必要です。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。

```
$ oc login https://{HOSTNAME}:8443
```

2. **jaeger-system** という名前の新規プロジェクトを作成します。

```
$ oc new-project jaeger-system
```

3. 以下のテキストが含まれる **jaeger.yaml** という名前のカスタムリソースファイルを作成します。

例: jaeger-all-in-one.yaml

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

4. 以下のコマンドを実行して Jaeger をデプロイします。

```
$ oc create -n jaeger-system -f jaeger.yaml
```

5. 以下のコマンドを実行して、インストールプロセス時の Pod の進捗を確認します。

```
$ oc get pods -n jaeger-system -w
```

インストールプロセスが完了すると、以下のような出力が表示されるはずです。

```
NAME                                  READY   STATUS    RESTARTS   AGE
jaeger-all-in-one-inmemory-cdff7897b-qhfdx   2/2     Running   0          24s
```

第4章 WEB コンソールからの JAEGER PRODUCTION ストラテジーのデプロイ

production デプロイメントストラテジーは、実稼働環境向けのストラテジーであり、トレースデータの長期の保存が重要となり、より拡張性および高可用性のあるアーキテクチャーも必要になります。

前提条件

- Elasticsearch Operator がインストールされている必要があります。
- Jaeger Operator がインストールされている必要があります。
- Jaeger インストールのカスタマイズ方法についての手順を確認します。
- **cluster-admin** ロールを持つアカウントが必要です。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. 新規プロジェクト (例: **jaeger-system**) を作成します。
   a. **Home → Projects** に移動します。
   b. **Create Project** をクリックします。
   c. **Name** フィールドに **jaeger-system** を入力します。
   d. **Create** をクリックします。
3. **Operators → Installed Operators** に移動します。
4. 必要な場合は、Project メニューから **jaeger-system** を選択します。Operator が新規プロジェクトにコピーされるまでに数分待機する必要がある場合があります。
5. Jaeger Operator をクリックします。**Overview** タブの **Provided APIs** で、Operator は単一リンクを提供します。
6. **Jaeger** で **Create Instance** をクリックします。
7. **Create Jaeger** ページで、デフォルトの **all-in-one** yaml テキストを実稼働用の YAML 設定に置き換えます。以下は例になります。

Elasticsearch を含む jaeger-production.yaml ファイルの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-production
  namespace:
spec:
  strategy: production
  ingress:
    security: oauth-proxy
  storage:
```

```
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      redundancyPolicy: SingleRedundancy
    esIndexCleaner:
      enabled: true
      numberOfDays: 7
      schedule: 55 23 * * *
    esRollover:
      schedule: '*/30 * * * *'
```

8. **Create** をクリックして Jaeger インスタンスを作成します。
9. **Jaegers** ページで、Jaeger インスタンスの名前 (例: **jaeger-prod-elasticsearch**) をクリックします。
10. **Jaeger Details** ページで、**Resources** タブをクリックします。すべての Pod のステータスが「Running」になるまで待機してから続行します。

4.1. CLI からの JAEGER 実稼働環境のデプロイ

以下の手順に従って、コマンドラインから Jaeger のインスタンスを作成します。

前提条件

- OpenShift Jaeger Operator がインストールされ、検証済みです。
- OpenShift CLI (**oc**) へのアクセスが可能です。
- **cluster-admin** ロールを持つアカウントが必要です。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。

```
$ oc login https://{HOSTNAME}:8443
```

2. **jaeger-system** という名前の新規プロジェクトを作成します。

```
$ oc new-project jaeger-system
```

3. 直前の手順のサンプルファイルのテキストが含まれる **jaeger-production.yaml** という名前のカスタムリソースファイルを作成します。
4. 以下のコマンドを実行して Jaeger をデプロイします。

```
$ oc create -n jaeger-system -f jaeger-production.yaml
```

5. 以下のコマンドを実行して、インストールプロセス時の Pod の進捗を確認します。

```
$ oc get pods -n jaeger-system -w
```

インストールプロセスが完了すると、以下のような出力が表示されるはずです。

```
NAME                                                  READY   STATUS    RESTARTS   AGE
elasticsearch-cdm-jaegersystemjaegerproduction-1-6676cf568gwhlw   2/2     Running   0
10m
elasticsearch-cdm-jaegersystemjaegerproduction-2-bcd4c8bf5l6g6w   2/2     Running   0
10m
elasticsearch-cdm-jaegersystemjaegerproduction-3-844d6d9694hhst   2/2     Running   0
10m
jaeger-production-collector-94cd847d-jwjlj            1/1     Running   3          8m32s
jaeger-production-query-5cbfbd499d-tv8zf              3/3     Running   3          8m32s
```

第5章 WEB コンソールからの JAEGER ストリーミングストラテジーのデプロイ

streaming デプロイメントストラテジーは、実稼働環境向けのストラテジーであり、トレースデータの長期の保存が重要となり、より拡張性および高可用性のあるアーキテクチャーも必要になります。

streaming ストラテジーは、Collector とストレージ (Elasticsearch) 間に配置されるストリーミング機能を提供します。これにより、負荷の高い状況でストレージに加わる圧力を軽減し、他のトレースの後処理機能がストリーミングプラットフォーム (Kafka) から直接リアルタイムのスパンデータを利用できるようにします。

注記

ストリーミングストラテジーには、AMQ Streams 用の追加の Red Hat サブスクリプションが必要です。AMQ Streams サブスクリプションをお持ちでない場合は、営業担当者にお問い合わせください。

前提条件

- AMQ Streams Operator がインストールされている必要があります。バージョン 1.4.0 以降を使用している場合は、セルフプロビジョニングを使用できます。それ以外の場合は、Kafka インスタンスを作成する必要があります。
- Jaeger Operator がインストールされている必要があります。
- Jaeger インストールのカスタマイズ方法についての手順を確認します。
- **cluster-admin** ロールを持つアカウントが必要です。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. 新規プロジェクト (例: **jaeger-system**) を作成します。
   a. **Home → Projects** に移動します。
   b. **Create Project** をクリックします。
   c. **Name** フィールドに **jaeger-system** を入力します。
   d. **Create** をクリックします。
3. **Operators → Installed Operators** に移動します。
4. 必要な場合は、Project メニューから **jaeger-system** を選択します。Operator が新規プロジェクトにコピーされるまでに数分待機する必要がある場合があります。
5. Jaeger Operator をクリックします。**Overview** タブの **Provided APIs** で、Operator は単一リンクを提供します。
6. **Jaeger** で **Create Instance** をクリックします。
7. **Create Jaeger** ページで、デフォルトの **all-in-one** yaml テキストをストリーミング用の YAML 設定に置き換えます。以下は例になります。

例: jaeger-streaming.yaml ファイル

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          #Note: If brokers are not defined,AMQStreams 1.4.0+ will self-provision Kafka.
          brokers: my-cluster-kafka-brokers.kafka:9092
  storage:
    type: elasticsearch
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
```

1. **Create** をクリックして Jaeger インスタンスを作成します。
2. **Jaegers** ページで、 Jaeger インスタンスの名前 (例: **jaeger-streaming**) をクリックします。
3. **Jaeger Details** ページで、**Resources** タブをクリックします。すべての Pod のステータスが「Running」になるまで待機してから続行します。

5.1. CLI からの JAEGER ストリーミングのデプロイ

以下の手順に従って、コマンドラインから Jaeger のインスタンスを作成します。

前提条件

- OpenShift Jaeger Operator がインストールされ、検証済みです。
- OpenShift CLI (**oc**) へのアクセスが可能です。
- **cluster-admin** ロールを持つアカウントが必要です。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。

```
$ oc login https://{HOSTNAME}:8443
```

2. **jaeger-system** という名前の新規プロジェクトを作成します。

```
$ oc new-project jaeger-system
```

3. 直前の手順のサンプルファイルのテキストが含まれる **jaeger-streaming.yaml** という名前のカスタムリソースファイルを作成します。

4. 以下のコマンドを実行して Jaeger をデプロイします。

```
$ oc create -n jaeger-system -f jaeger-streaming.yaml
```

5. 以下のコマンドを実行して、インストールプロセス時の Pod の進捗を確認します。

```
$ oc get pods -n jaeger-system -w
```

インストールプロセスが完了すると、以下のような出力が表示されるはずです。

```
NAME                                                  READY   STATUS    RESTARTS   AGE
elasticsearch-cdm-jaegersystemjaegerstreaming-1-697b66d6fcztcnn   2/2     Running   0          5m40s
elasticsearch-cdm-jaegersystemjaegerstreaming-2-5f4b95c78b9gckz   2/2     Running   0          5m37s
elasticsearch-cdm-jaegersystemjaegerstreaming-3-7b6d964576nnz97   2/2     Running   0          5m5s
jaeger-streaming-collector-6f6db7f99f-rtcfm                       1/1     Running   0          80s
jaeger-streaming-entity-operator-6b6d67cc99-4lm9q                 3/3     Running   2          2m18s
jaeger-streaming-ingester-7d479847f8-5h8kc                        1/1     Running   0          80s
jaeger-streaming-kafka-0                                          2/2     Running   0          3m1s
jaeger-streaming-query-65bf5bb854-ncnc7                           3/3     Running   0          80s
jaeger-streaming-zookeeper-0                                      2/2     Running   0          3m39s
```

5.2. JAEGER デプロイメントのカスタマイズ

5.2.1. Jaeger のデフォルト設定オプション

Jaeger カスタムリソース (CR) は、Jaeger リソースの作成時に使用されるアーキテクチャーおよび設定を定義します。これらのパラメーターを変更して、Jaeger 実装をビジネスニーズに合わせてカスタマイズできます。

Jaeger 汎用 YAML の例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: name
spec:
  strategy: <deployment_strategy>
  allInOne:
    options: {}
    resources: {}
  agent:
    options: {}
    resources: {}
  collector:
    options: {}
    resources: {}
```

```
  sampling:
    options: {}
  storage:
    type:
    options: {}
  query:
    options: {}
    resources: {}
  ingester:
    options: {}
    resources: {}
  options: {}
```

表5.1 Jaeger パラメーター

パラメーター	説明	値	デフォルト値
apiVersion:	オブジェクトの作成時に使用する Application Program Interface のバージョン。	**jaegertracing.io/v1**	**jaegertracing.io/v1**
kind:	作成する Kubernetes オブジェクトの種類を定義します。	**jaeger**	
metadata:	**name** 文字列、**UID**、およびオプションの **namespace** などのオブジェクトを一意に特定するのに役立つデータ。		OpenShift は **UID** を自動的に生成し、オブジェクトが作成されるプロジェクトの名前で **namespace** を完成します。
name:	オブジェクトの名前。	Jaeger インスタンスの名前。	**jaeger-all-in-one-inmemory**
spec:	作成するオブジェクトの仕様。	Jaeger インスタンスのすべての設定パラメーターが含まれます。(Jaeger コンポーネントすべてに) 共通する定義が必要な場合、これは仕様ノードで定義されます。定義が個別のコンポーネントに関連する場合、これは spec/<component> ノードの下に配置されます。	該当なし
strategy:	Jaeger デプロイメントストラテジー	**allInOne**、**production**、または **streaming**	**allInOne**

パラメーター	説明	値	デフォルト値
allInOne:	allInOne イメージはエージェント、Collector、Query、Ingester, Jaeger UI を単一 Pod にデプロイするため、このデプロイメントの設定は、コンポーネント設定を allInOne パラメーターの下でネストする必要があります。		
agent:	Jaeger エージェントを定義する設定オプション。		
collector:	Jaeger Collector を定義する設定オプション。		
sampling:	トレース用のサンプリングストラテジーを定義する設定オプション。		
storage:	ストレージを定義する設定オプション。すべてのストレージ関連のオプションは、 **allInOne** または他のコンポーネントオプションではなく、**storage** に配置される必要があります。		
query:	Query サービスを定義する設定オプション。		
ingester:	Ingester サービスを定義する設定オプション。		

以下の YAML サンプルは、デフォルト設定を使用して Jaeger インスタンスを作成するための最小要件です。

最小要件の jaeger-all-in-one.yaml の例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

5.2.2. Jaeger Collector 設定オプション

Jaeger Collector は、トレーサーによってキャプチャーされたスパンを受信し、**production** ストラテジーを使用する場合はそれらを永続ストレージ(Elasticsearch) に書き込み、**streaming** ストラテジーを使用する場合は AMQ Streams に書き込むコンポーネントです。

コレクターはステートレスであるため、Jaeger Collector のインスタンスの多くは並行して実行できます。Elasticsearch クラスターの場所を除き、Collector では設定がほとんど必要ありません。

表5.2 Jaeger コレクターパラメーター

パラメーター	説明	値
spec: collector: options: {}	Jaeger Collector を定義する設定オプション。	
autoscale:	このパラメーターは、Collector の自動スケーリングを有効または無効にするかどうかを制御します。自動スケーリングを明示的に無効にするには、**false** に設定します。	**true**/**false**
kafka: producer: topic: jaeger-spans	**topic** パラメーターは、コレクターによってメッセージを生成するために使用され、Ingester によってメッセージを消費するために使用される Kafka 設定を特定します。	プロデューサーのラベル
kafka: producer: brokers: my-cluster-kafka-brokers.kafka:9092	メッセージを生成するために Collector によって使用される Kafka 設定を特定します。ブローカーが指定されていない場合で、AMQ Streams 1.4.0+ がインストールされている場合、Jaeger は Kafka をセルフプロビジョニングします。	
log-level:	コレクターのロギングレベル。	**trace**、**debug**、**info**、**warning**、**error**、**fatal**、 **panic**
maxReplicas:	Collector の自動スケーリング時に作成するレプリカの最大数を指定します。	整数 (例: **100**)。
num-workers:	キューからプルするワーカーの数。	整数 (例: **50**)。

パラメーター	説明	値
queue-size:	Collector キューのサイズ。	整数 (例: **2000**)。
replicas:	作成する Collector レプリカの数を指定します。	整数 (例: **5**)。

5.2.2.1. 自動スケーリングのための Collector の設定

Collector を自動スケーリングできるように設定できます。Collector は CPU および/またはメモリーの消費に基づいてスケールアップまたはスケールダウンします。Collector を自動スケーリングできるように設定すると、Jaeger 環境は負荷が増加するとスケールアップし、必要なリソースが少なくなるとスケールダウンし、これによってコストを節約できます。自動スケーリングを設定するには、**autoscale** パラメーターを **true** に設定し、**.spec.collector.maxReplicas** の値を、Collector の Pod が消費することが予想されるリソースの妥当な値と共に指定します。**.spec.collector.maxReplicas** の値を設定しない場合、Operator はこれを **100** に設定します。

デフォルトで、**.spec.collector.replicas** の値が指定されていない場合、Jaeger Operator は Collector の Horizontal Pod Autoscaler (HPA) 設定を作成します。HPA についての詳細は、Kubernetes ドキュメント を参照してください。

以下は、Collector の制限とレプリカの最大数を設定する自動スケーリングの設定例です。

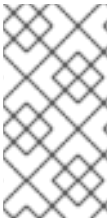
Collector の自動スケーリングの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  collector:
    maxReplicas: 5
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
```

5.2.3. Jaeger サンプリング設定オプション

この Operator は、リモートサンプラーを使用するように設定されているトレーサーに提供されるサンプリングストラテジーを定義するために使用できます。

すべてのトレースが生成される間に、それらの一部のみがサンプリングされます。トレースをサンプリングすると、追加の処理や保存のためにトレースにマークが付けられます。



注記

これは、トレースがサンプリングの意思決定が行われる際に Istio プロキシーによって開始されている場合には関連がありません。Jaeger サンプリングの意思決定は、トレースが Jaeger トレーサーを使用してアプリケーションによって開始される場合にのみ関連します。

サービスがトレースコンテキストが含まれていない要求を受信すると、Jaeger トレーサーは新しいトレースを開始し、これにランダムなトレース ID を割り当て、現在インストールされているサンプリングストラテジーに基づいてサンプリングの意思決定を行います。サンプリングの意思決定はトレース内の後続のすべての要求に伝播され、他のサービスが再度サンプリングの意思決定を行わないようにします。

Jaeger ライブラリーは以下のサンプラーをサポートします。

- **Constant**: サンプラーは、すべてのトレースについて常に同じ意思決定を行います。これは、すべてのトレースをサンプリングするか (sampling.param=1)、またはそれらのいずれもサンプリングしません (sampling.param=0)。
- **Probabilistic**: サンプラーは、**sampling.param** プロパティーの値と等しいサンプリングの確率で、ランダムなサンプリングの意思決定を行います。たとえば、sampling.param=0.1 の場合に、約 10 の内 1 のトレースがサンプリングされます。
- **Rate Limiting**: サンプラーは、リーキーバケット (leaky bucket) レートリミッターを使用して、トレースが一定のレートでサンプリングされるようにします。たとえば、sampling.param=2.0 の場合、1 秒あたり 2 トレースの割合で要求がサンプリングされます。
- **Remote**: サンプラーは Jaeger エージェントで現在のサービスで使用する適切なサンプリングストラテジーを参照します。これにより、Jaeger バックエンドの中央設定からサービス内のサンプリングストラテジーを制御できます。

表5.3 Jaeger サンプリングパラメーター

パラメーター	説明	値	デフォルト値
spec: sampling: options: {}	トレース用のサンプリングストラテジーを定義する設定オプション。		
sampling: type:	使用するサンプリングストラテジー。(上記の説明を参照してください。)	有効な値は **const**、**probabilistic**、**ratelimiting**、および **remote** です。	**remote**
sampling: options: type: param:	選択したサンプリングストラテジーのパラメーター(上記の例を参照してください。)	10 進値および整数値 (0、.1、1、10)	該当なし

この例では、トレースインスタンスをサンプリングする確率が 50% の、確率的なデフォルトサンプリングストラテジーを定義します。

確率的なサンプリングの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: with-sampling
spec:
  strategy: allInOne
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 50
```

5.2.4. Jaeger ストレージ設定オプション

spec:storage の下で Collector、Ingester、および Query サービスのストレージを設定します。これらの各コンポーネントの複数のインスタンスは、パフォーマンスと回復性を確保するために、必要に応じてプロビジョニングできます。

制限

- namespace ごとにセルフプロビジョニングされた Elasticsearch インスタンスを持つ 1 つの Jaeger のみを使用できます。
- namespace ごとに 1 つの Elasticsearch のみを使用できます。
- Jaeger で OpenShift Jaeger ロギング Elasticsearch インスタンスを共有したり、再利用したりすることはできません。Elasticsearch クラスターは単一の Jaeger インスタンスの専用のクラスターになります。

注記

Elasticsearch を OpenShift ロギングの一部としてインストールしている場合、Jaeger Operator はインストールされた Elasticsearch Operator を使用してストレージをプロビジョニングできます。

表5.4 Jaeger の一般的なストレージパラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  storage:`
`  options: {}`
	ストレージを定義する設定オプション。		

パラメーター	説明	値	デフォルト値
storage: type:	デプロイメントに使用するストレージのタイプ。	**memory** または **elasticsearch** メモリーストレージは、Pod がシャットダウンした場合にデータが永続化されないため、開発、テスト、デモ、および概念検証用の環境にのみに適しています。実稼働環境については、Jaeger は永続ストレージの Elasticsearch をサポートします。	**memory**

表5.5 Elasticsearch 設定パラメーター

パラメーター	説明	値	デフォルト値
General Elasticsearch configuration settings			
elasticsearch: server-urls:	Elasticsearch インスタンスの URL。	Elasticsearch サーバーの完全修飾ドメイン名。**spec:storage:type= elasticsearch** を指定しているが、**server-urls** パラメーターの値を指定していない場合、Jaeger Operator は Elasticsearch Operator を使用し、カスタムリソースファイルの **spec:storage:elasticsearch** セクションの設定を使用して Elasticsearch クラスターを作成します。	**http://elasticsearch.<namespace>.svc:9200**
es: max-num-spans:	Elasticsearch のクエリーごとに 1 度に取得するスパンの最大数。		10000
es: max-span-age:	Elasticsearch のスパンについての最大ルックバック。		72h0m0s

パラメーター	説明	値	デフォルト値
elasticsearch: secretname:	シークレットの名前 (例: **jaeger-secret**)		該当なし
es: sniffer:	Elasticsearch のスニファー設定。クライアントはスニファープロセスを使用してすべてのノードを自動的に検出します。デフォルトでは無効にされています。	**true**/ **false**	**false**
es: timeout:	クエリーに使用されるタイムアウト。ゼロに設定するとタイムアウトはありません。		0s
es: username:	Elasticsearch で必要なユーザー名。Basic 認証は、指定されている場合に CA も読み込みます。**es.password** も参照してください。		
es: password:	Elasticsearch で必要なパスワード。**es.username** も参照してください。		
es: version:	主要な Elasticsearch バージョン。指定されていない場合、値は Elasticsearch から自動検出されます。		0
Elasticsearch resource configuration settings			
elasticsearch: nodeCount:	Elasticsearch ノードの数。高可用性を確保するには、少なくとも 3 つのノードを使用します。「スプリットブレイン」の問題が生じる可能性があるため、2 つのノードを使用しないでください。	整数値。例: 概念実証用 = 1、最小デプロイメント = 3	1

パラメーター	説明	値	デフォルト値
elasticsearch: resources: requests: cpu:	ご使用の環境設定に基づく、要求に対する中央処理単位の数。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。例: 概念実証用 = 500m、最小デプロイメント = 1	1Gi
elasticsearch: resources: requests: memory:	ご使用の環境設定に基づく、要求に使用できるメモリー。	バイト単位で指定されます (例: 200Ki、50Mi、5Gi)。例: 概念実証用 = 1Gi、最小デプロイメント = 16Gi*	500m
elasticsearch: resources: limits: cpu:	ご使用の環境設定に基づく、中央処理単位数の制限。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。例: 概念実証用 = 500m、最小デプロイメント = 1	
elasticsearch: resources: limits: memory:	ご使用の環境設定に基づく、利用可能なメモリー制限。	バイト単位で指定されます (例: 200Ki、50Mi、5Gi)。例: 概念実証用 = 1Gi、最小デプロイメント = 16Gi*	
	各 Elasticsearch ノードはこれより低い値のメモリー設定でも動作しますが、これは実稼働環境でのデプロイメントには推奨されません。実稼働環境で使用する場合、デフォルトで各 Pod に割り当てる設定を 16Gi 未満にすることはできず、Pod ごとに最大 64Gi を割り当てることを推奨します。		
Elasticsearch data replication options			
elasticsearch: redundancyPolicy:	データレプリケーションポリシーは、Elasticsearch シャードをクラスター内のデータノードにレプリケートする方法を定義します。指定されていない場合、Jaeger Operator はノード数に基づいて最も適切なレプリケーションを自動的に判別します。	**ZeroRedundancy**（レプリカシャードなし）、**SingleRedundancy**（レプリカシャード 1 つ）、**MultipleRedundancy**（各インデックスはデータノードの半分に分散される）、**FullRedundancy** (各インデックスはクラスター内のすべてのデータノードに完全にレプリケートされます）	

パラメーター	説明	値	デフォルト値
es: num-replicas:	Elasticsearch のインデックスごとのレプリカ数。		1
es: num-shards:	Elasticsearch のインデックスごとのシャード数。		5
Elasticsearch index and index cleaner configuration options			
es: create-index-templates:	**true** に設定されている場合、アプリケーションの起動時にインデックステンプレートを自動的に作成します。テンプレートが手動でインストールされる場合は、**false** に設定されます。	**true**/ **false**	**true**
es: index-prefix:	Jaeger インデックスのオプションのプレフィックス。たとえば、これを「production」に設定すると、「production-jaeger-*」という名前のインデックスが作成されます。		
esIndexCleaner: enabled:	Elasticsearch ストレージを使用する場合、デフォルトでジョブが作成され、古いトレースをインデックスからクリーンアップします。このパラメーターは、インデックスクリーナージョブを有効または無効にします。	**true**/ **false**	**true**
esIndexCleaner: numberOfDays:	インデックスの削除を待機する日数。	整数値	**7**
esIndexCleaner: schedule:	Elasticsearch インデックスを消去する頻度についてのスケジュールを定義します。	cron 式	"55 23 * * *"

パラメーター	説明	値	デフォルト値
esRollover: schedule:	新規 Elasticsearch インデックスにロールオーバーする頻度についてのスケジュールを定義します。	cron 式	'*/30 * * * *'
Configuration settings for Elasticsearch bulk processor			
es: bulk: actions:	バルクプロセッサーがディスクへの更新のコミットを決定する前にキューに追加できる要求の数。		1000
es: bulk: flush-interval:	**time.Duration**: この後に、他のしきい値に関係なく一括要求がコミットされます。バルクプロセッサーのフラッシュ間隔を無効にするには、これをゼロに設定します。		200ms
es: bulk: size:	バルクプロセッサーがディスクへの更新をコミットするまでに一括要求が発生する可能性のあるバイト数。		5000000
es: bulk: workers:	一括要求を受信し、Elasticsearch にコミットできるワーカーの数。		1
Elasticsearch TLS configuration settings			
es: tls: ca:	リモートサーバーの検証に使用される TLS 認証局 (CA) ファイルへのパス。		デフォルトではシステムトラストストアを使用します。
es: tls: cert:	リモートサーバーに対するこのプロセスの特定に使用される TLS 証明書ファイルへのパス。		

パラメーター	説明	値	デフォルト値
es: tls: enabled:	リモートサーバーと通信する際に、トランスポート層セキュリティー (TLS) を有効にします。デフォルトでは無効にされています。	**true**/ **false**	**false**
es: tls: key:	リモートサーバーに対するこのプロセスの特定に使用される TLS 秘密鍵ファイルへのパス。		
es: tls: server-name:	リモートサーバーの証明書の予想される TLS サーバー名を上書きします。		
es: token-file:	ベアラートークンが含まれるファイルへのパス。このフラグは、指定されている場合は認証局 (CA) ファイルも読み込みます。		
Elasticsearch archive configuration settings			
es-archive: bulk: actions:	バルクプロセッサーがディスクへの更新のコミットを決定する前にキューに追加できる要求の数。		0
es-archive: bulk: flush-interval:	**time.Duration**: この後に、他のしきい値に関係なく一括要求がコミットされます。バルクプロセッサーのフラッシュ間隔を無効にするには、これをゼロに設定します。		0s
es-archive: bulk: size:	バルクプロセッサーがディスクへの更新をコミットするまでに一括要求が発生する可能性のあるバイト数。		0

パラメーター	説明	値	デフォルト値
es-archive: bulk: workers:	一括要求を受信し、Elasticsearch にコミットできるワーカーの数。		0
es-archive: create-index-templates:	**true** に設定されている場合、アプリケーションの起動時にインデックステンプレートを自動的に作成します。テンプレートが手動でインストールされる場合は、**false** に設定されます。	**true**/ **false**	**false**
es-archive: enabled:	追加ストレージを有効にします。	**true**/ **false**	**false**
es-archive: index-prefix:	Jaeger インデックスのオプションのプレフィックス。たとえば、これを「production」に設定すると、「production-jaeger-*」という名前のインデックスが作成されます。		
es-archive: max-num-spans:	Elasticsearch のクエリーごとに 1 度に取得するスパンの最大数。		0
es-archive: max-span-age:	Elasticsearch のスパンについての最大ルックバック。		0s
es-archive: num-replicas:	Elasticsearch のインデックスごとのレプリカ数。		0
es-archive: num-shards:	Elasticsearch のインデックスごとのシャード数。		0
es-archive: password:	Elasticsearch で必要なパスワード。**es.username** も参照してください。		

パラメーター	説明	値	デフォルト値
es-archive: server-urls:	Elasticsearch サーバーのコンマ区切りの一覧。完全修飾 URL（例: **http://localhost:9200**) として指定される必要があります。		
es-archive: sniffer:	Elasticsearch のスニファー設定。クライアントはスニファープロセスを使用してすべてのノードを自動的に検出します。デフォルトでは無効にされています。	**true**/ **false**	**false**
es-archive: timeout:	クエリーに使用されるタイムアウト。ゼロに設定するとタイムアウトはありません。		0s
es-archive: tls: ca:	リモートサーバーの検証に使用される TLS 認証局 (CA) ファイルへのパス。		デフォルトではシステムトラストストアを使用します。
es-archive: tls: cert:	リモートサーバーに対するこのプロセスの特定に使用される TLS 証明書ファイルへのパス。		
es-archive: tls: enabled:	リモートサーバーと通信する際に、トランスポート層セキュリティー (TLS) を有効にします。デフォルトでは無効にされています。	**true**/ **false**	**false**
es-archive: tls: key:	リモートサーバーに対するこのプロセスの特定に使用される TLS 秘密鍵ファイルへのパス。		

パラメーター	説明	値	デフォルト値
es-archive: tls: server-name:	リモートサーバーの証明書の予想される TLS サーバー名を上書きします。		
es-archive: token-file:	ベアラートークンが含まれるファイルへのパス。このフラグは、指定されている場合は認証局 (CA) ファイルも読み込みます。		
es-archive: username:	Elasticsearch で必要なユーザー名。Basic 認証は、指定されている場合に CA も読み込みます。**es-archive.password** も参照してください。		
es-archive: version:	主要な Elasticsearch バージョン。指定されていない場合、値は Elasticsearch から自動検出されます。		0

実稼働ストレージの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      resources:
        requests:
          cpu: 1
          memory: 16Gi
        limits:
          memory: 16Gi
```

ボリュームマウントを含むストレージの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
```

```
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200
        index-prefix: my-prefix
        tls:
          ca: /es/certificates/ca.crt
    secretName: jaeger-secret
  volumeMounts:
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

永続ストレージを含むストレージの例:

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 1
      storage: 1
        storageClassName: gp2
        size: 5Gi
      resources:
        requests:
          cpu: 200m
          memory: 4Gi
        limits:
          memory: 4Gi
      redundancyPolicy: ZeroRedundancy
```

1 永続ストレージの設定。この場合、AWS **gp2** のサイズは **5Gi** です。値の指定がない場合、Jaeger は **emptyDir** を使用します。Elasticsearch Operator は、Jaeger インスタンスで削除されない **PersistentVolumeClaim** および **PersistentVolume** をプロビジョニングします。同じ名前および namespace で Jaeger インスタンスを作成する場合は、同じボリュームをマウントできます。

5.2.5. Jaeger Query 設定オプション

Query とは、ストレージからトレースを取得し、ユーザーインターフェースをホストしてそれらを表示するサービスです。

表5.6 Jaeger Query パラメーター

パラメーター	説明	値	デフォルト値
spec: query: options: {} resources: {}	Query サービスを定義する設定オプション。		
query: additional- headers:	追加の HTTP 応答ヘッダー。複数回指定できます。	形式: "Key: Value"	
query: base-path:	すべての jaeger-query HTTP ルートのベースパスは、root 以外の値に設定できます。たとえば、**/jaeger** ではすべての UI URL が **/jaeger** で開始するようになります。これは、リバースプロキシーの背後で jaeger-query を実行する場合に役立ちます。	/{path}	
query: port:	クエリーサービスのポート。		16686
options: log-level:	Query のロギングレベル。	使用できる値は、**trace**、**debug**、**info**、**warning**、**error**、**fatal**、**panic** です。	

Query 設定の例

```
apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "my-jaeger"
spec:
  strategy: allInOne
  allInOne:
    options:
      log-level: debug
      query:
        base-path: /jaeger
```

5.2.6. Jaeger Ingester 設定オプション

Ingester は、Kafka トピックから読み取り、別のストレージバックエンド (Elasticsearch) に書き込むサービスです。**allInOne** または **production** デプロイメントストラテジーを使用している場合は、Ingester サービスを設定する必要はありません。

表5.7 Jaeger Ingester パラメーター

パラメーター	説明	値
spec: strategy: streaming ingester: options: {}	Ingester サービスを定義する設定オプション。	
autoscale:	このパラメーターは、Ingester の自動スケーリングを有効または無効にするかどうかを制御します。自動スケーリングはデフォルトで有効にされます。自動スケーリングを明示的に無効にするには、**false** に設定します。	**true**/**false**
kafka: consumer: topic:	**topic** パラメーターは、コレクターによってメッセージを生成するために使用され、Ingester によってメッセージを消費するために使用される Kafka 設定を特定します。	コンシューマーのラベル例: **jaeger-spans**
kafka: consumer: brokers:	メッセージを消費するために Ingester によって使用される Kafka 設定を特定します。	ブローカーのラベル (例: **my-cluster-kafka-brokers.kafka:9092**)
ingester: deadlockInterval:	Ingester が終了するまでメッセージを待機する間隔 (秒単位または分単位) を指定します。システムの初期化中にメッセージが到達されない場合に Ingester が終了しないようにするため、デッドロック間隔はデフォルトで無効に（0 に設定）されます。	分と秒 (例: **1m0s**)デフォルト値は **0** です。
log-level:	Ingester のロギングレベル。	使用できる値は、**trace**、**debug**、**info**、**warning**、**error**、**fatal**、**panic** です。

パラメーター	説明	値
maxReplicas:	Ingester の自動スケーリング時に作成するレプリカの最大数を指定します。	整数 (例: **100**)

ストリーミング Collector および Ingester の例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
      ingester:
        deadlockInterval: 5
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: http://elasticsearch:9200
```

5.2.6.1. 自動スケーリングのための Ingester の設定

Ingester を自動スケーリングできるように設定できます。Ingester は CPU および/またはメモリーの消費に基づいてスケールアップまたはスケールダウンします。Ingester を自動スケーリングできるように設定すると、Jaeger 環境は負荷が増加するとスケールアップし、必要なリソースが少なくなるとスケールダウンし、これによってコストを節約できます。自動スケーリングを設定するには、**autoscale** パラメーターを **true** に設定し、**.spec.ingester.maxReplicas** の値を、Ingester の Pod が消費することが予想されるリソースの妥当な値と共に指定します。**.spec.ingester.maxReplicas** の値を設定しない場合、Operator はこれを **100** に設定します。

デフォルトで、**.spec.ingester.replicas** の値が指定されていない場合、Jaeger Operator は Ingester の Horizontal Pod Autoscaler (HPA) 設定を作成します。HPA についての詳細は、Kubernetes ドキュメント を参照してください。

以下は、Ingester の制限とレプリカの最大数を設定する自動スケーリングの設定例です。

Ingester の自動スケーリングの例

```
apiVersion: jaegertracing.io/v1
```

```
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  ingester:
    maxReplicas: 8
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
```

5.3. サイドカーコンテナーの挿入

OpenShift Jaeger は、アプリケーションの Pod 内のプロキシーサイドカーコンテナーを使用してエージェントを提供します。Jaeger Operator は Jaeger Agent サイドカーを Deployment ワークロードに挿入できます。自動のサイドカーコンテナー挿入を有効にしたり、手動で管理したりできます。

第6章 サイドカーコンテナーの自動挿入

この機能を有効にするには、文字列 **true** または **oc get jaegers** で返される Jaeger インスタンス名のいずれかに設定されるアノテーション **sidecar.jaegertracing.io/inject** を追加します。**true** を指定する場合、デプロイメントとして単一の Jaeger インスタンスのみが同じ namespace に存在する必要があります。そうでない場合に、Operator は使用する Jaeger インスタンスを判別することができません。デプロイメントの特定の Jaeger インスタンス名は、その namespace に適用される **true** よりも優先されます。

以下のスニペットは、サイドカーコンテナーを挿入する単純なアプリケーションを示しています。Jaeger エージェントは、同じ namespace で利用可能な単一の Jaeger インスタンスを参照します。

サイドカーの自動挿入の例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    "sidecar.jaegertracing.io/inject": "true" 1
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: acme/myapp:myversion
```

サイドカーコンテナーが挿入されると、Jaeger エージェントは **localhost** のデフォルトの場所でアクセスできます。

第7章 サイドカーコンテナーの手動挿入

Deployments 以外のコントローラータイプ（**StatefulSets**、**DaemonSet** など）の場合、Jaeger エージェントサイドカーを仕様に手動で定義できます。

以下のスニペットは、Jaeger エージェントサイドカーのコンテナーセクションに追加できる手動の定義を示しています。

StatefulSet のサイドカー定義の例

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: example-statefulset
  namespace: example-ns
  labels:
    app: example-app
spec:

    spec:
      containers:
        - name: example-app
          image: acme/myapp:myversion
          ports:
            - containerPort: 8080
              protocol: TCP
        - name: jaeger-agent
          image: registry.redhat.io/distributed-tracing/jaeger-agent-rhel7:<version>
           # The agent version must match the operator version
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 5775
              name: zk-compact-trft
              protocol: UDP
            - containerPort: 5778
              name: config-rest
              protocol: TCP
            - containerPort: 6831
              name: jg-compact-trft
              protocol: UDP
            - containerPort: 6832
              name: jg-binary-trft
              protocol: UDP
            - containerPort: 14271
              name: admin-http
              protocol: TCP
          args:
            - --reporter.grpc.host-port=dns:///jaeger-collector-headless.example-ns:14250
            - --reporter.type=grpc
```

その後、Jaeger エージェントは localhost のデフォルトの場所でアクセスできます。

7.1. JAEGER のアップグレード

Operator Lifecycle Manager は、クラスター内の Operator のインストール、アップグレード、ロール

ベースのアクセス制御 (RBAC) を制御します。OLM はデフォルトで OpenShift Container Platform で実行されます。OLM は利用可能な Operator のクエリーやインストールされた Operator のアップグレードを実行します。OpenShift Container Platform のアップグレードの処理方法についての詳細は、Operator Lifecycle Manager のドキュメントを参照してください。

Jaeger Operator によって使用される更新方法により、管理された Jaeger インスタンスが Operator に関連付けられたバージョンにアップグレードされます。Jaeger Operator の新規バージョンがインストールされるたびに、Operator によって管理されるすべての Jaeger アプリケーションインスタンスがその Operator のバージョンにアップグレードされます。たとえば、バージョン 1.10 (Operator およびバックエンドコンポーネントの両方）がインストールされ、Operator がバージョン 1.11 にアップグレードされると、Operator のアップグレードが完了次第、Operator は実行中の Jaeger インスタンスをスキャンし、それらを 1.11 にアップグレードします。

7.2. JAEGER の削除

OpenShift Container Platform クラスターから Jaeger を削除する手順は、以下のとおりです。

1. Jaeger Pod をシャットダウンします。
2. Jaeger インスタンスを削除します。
3. Jaeger Operator を削除します。

7.2.1. Web コンソールを使用した Jaeger インスタンスの削除

注記

インメモリーストレージを使用するインスタンスを削除すると、すべてのデータが完全に失われます。永続ストレージ (Elasticsearch など）に保存されているデータは、Jaeger インスタンスが削除されても削除されません。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** に移動します。
3. Project メニューから Operator がインストールされているプロジェクトの名前 (例: **jaeger-system**) を選択します。
4. Jaeger Operator をクリックします。
5. **Jaeger** タブをクリックします。
6. Options メニュー  (削除するインスタンスの横にある) をクリックし、**Delete Jaeger** を選択します。
7. 確認ウィンドウで **Delete** をクリックします。

7.2.2. CLI からの Jaeger インスタンスの削除

1. OpenShift Container Platform CLI にログインします。

```
$ oc login
```

2. Jaeger インスタンスを表示するには、以下のコマンドを実行します。

```
$ oc get deployments -n <jaeger-project>
```

Operator の名前には、サフィックスの **-operator** が付きます。以下の例は、2 つの Jaeger Operator と 4 つの Jaeger インスタンスを示しています。

```
$ oc get deployments -n jaeger-system
```

以下のような出力が表示されるはずです。

```
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch-operator   1/1     1            1           93m
jaeger-operator          1/1     1            1           49m
jaeger-test              1/1     1            1           7m23s
jaeger-test2             1/1     1            1           6m48s
tracing1                 1/1     1            1           7m8s
tracing2                 1/1     1            1           35m
```

3. Jaeger のインスタンスを削除するには、以下のコマンドを実行します。

```
$ oc delete jaeger <deployment-name> -n <jaeger-project>
```

例:

```
$ oc delete jaeger tracing2 -n jaeger-system
```

4. 削除を確認するには、**oc get deployment** を再度実行します。

```
$ oc get deployments -n <jaeger-project>
```

例:

```
$ oc get deployments -n jaeger-system
```

以下のような出力が生成されるはずです。

```
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch-operator   1/1     1            1           94m
jaeger-operator          1/1     1            1           50m
jaeger-test              1/1     1            1           8m14s
jaeger-test2             1/1     1            1           7m39s
tracing1                 1/1     1            1           7m59s
```

7.2.3. Jaeger Operator の削除

手順

1. クラスターからの Operator の削除についての手順に従います。

- Jaeger Operator を削除します。
- Jaeger Operator の削除後、(該当する場合は) Elasticsearch Operator を削除します。

第8章 JAEGER の統合

8.1. OPENSHIFT SERVERLESS を使用した JAEGER とサーバーレスアプリケーションの統合

Jaeger を OpenShift Serverless で使用すると、OpenShift Container Platform でのサーバーレスアプリケーションの 分散トレース を有効にできます。

8.1.1. OpenShift Serverless で使用する Jaeger の設定

前提条件

OpenShift Serverless で使用する Jaeger を設定するには、以下が必要です。

- OpenShift Container Platform クラスターでのクラスター管理者パーミッション。
- OpenShift Serverless Operator および Knative Serving が現時点でインストールされていること。
- Jaeger Operator が現時点でインストールされていること。

手順

1. 以下のサンプル YAML を含む Jaeger カスタムリソース YAML ファイルを作成し、これを適用します。

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger
  namespace: default
```

2. **KnativeServing** リソースを編集し、トレース用に YAML 設定を追加して、Knative Serving のトレースを有効にします。

```
apiVersion: operator.knative.dev/v1alpha1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  config:
    tracing:
      sample-rate: "0.1" 1
      backend: zipkin 2
      zipkin-endpoint: http://jaeger-collector.default.svc.cluster.local:9411/api/v2/spans 3
      debug: "false" 4
```

1 1 **sample-rate** はサンプリングの可能性を定義します。**sample-rate: "0.1"** を使用すると、10 トレースの内の 1 つがサンプリングされます。

2 **backend** は **zipkin** に設定される必要があります。

3 **zipkin-endpoint** は **jaeger-collector** サービスエンドポイントを参照する必要があります。このエンドポイントを取得するには、Jaeger カスタムリソースが適用される namespace を置き換えます。

4 デバッグは **false** に設定する必要があります。**debug: "true"** を設定してデバッグモードを有効にすることで、サンプリングをバイパスしてすべてのスパンがサーバーに送信されるようにします。

検証手順

Jaeger Web コンソールにアクセスして、トレースデータを表示します。**jaeger** ルートを使用して Jaeger Web コンソールにアクセスできます。

1. 以下のコマンドを入力して **jaeger** ルートのホスト名を取得します。

```
$ oc get route jaeger
```

```
NAME     HOST/PORT                         PATH   SERVICES      PORT    TERMINATION
WILDCARD
jaeger   jaeger-default.apps.example.com          jaeger-query  <all>   reencrypt     None
```

2. ブラウザーでエンドポイントアドレスを開き、コンソールを表示します。