OpenShift Container Platform 4.12

分散トレース

OpenShift Container Platform での分散トレーシングの設定と使用

Last Updated: 2024-07-04

OpenShift Container Platform 4.12 分散トレース

OpenShift Container Platform での分散トレーシングの設定と使用

法律上の通知

概要

分散トレーシングを使用して、OpenShift Container Platform の分散システムを通過するマイクロサービストランザクションを保存、分析、視覚化します。

目次

第1章 リリースノート

1.1. RED HAT OPENSHIFT 分散トレーシングプラットフォーム 3.2.1 のリリースノート

1.1.1. 分散トレースの概要

サービスの所有者は、分散トレース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。Red Hat OpenShift 分散トレーシングプラットフォームを使用すると、最新のクラウドネイティブの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、トラブルシューティングが可能です。

分散トレースプラットフォームを使用すると、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

分散トレーシングプラットフォームは、Red Hat build of OpenTelemetry と組み合わせて使用 できます。

Red Hat OpenShift 分散トレーシングプラットフォームのこのリリースには、Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) と非推奨の Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) が含まれています。

1.1.2. CVE

このリリースでは、CVE-2024-25062 が修正されます。

1.1.3. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)

Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) は、Tempo Operator を通じて提供されます。

1.1.3.1. 既知の問題

現在、次のような既知の問題があります。

- 現在、IBM Z (**s390x**) アーキテクチャーでは、distributed tracing platform (Tempo) が失敗します。(TRACING-3545)

1.1.4. Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、Red Hat OpenShift 分散トレーシングプラットフォーム Operator を通じて提供されます。

重要

Jaeger は、FIPS 検証済みの暗号化モジュールを使用しません。

1.1.4.1. 既知の問題

現在、次のような既知の問題があります。

- 現在、Apache Spark はサポートされていません。
- 現在、AMQ/Kafka 経由のストリーミングデプロイメントは、{ibm-z-title} および {ibm-power-title} アーキテクチャーではサポートされていません。

1.1.5. サポート

本書で説明されている手順、または OpenShift Container Platform で問題が発生した場合は、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では、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションを対象とした 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。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager Hybrid Cloud Console で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

本書の改善への提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントの Jira Issue 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。

1.1.6. 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、用語の置き換えは、今後の複数のリリースにわたって段階的に実施されます。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

1.2. 以前にリリースされた RED HAT OPENSHIFT 分散トレーシングプラットフォームのリリースノート

1.2.1. 分散トレースの概要

サービスの所有者は、分散トレース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。Red Hat OpenShift 分散トレーシングプラットフォームを使用すると、最新のクラウドネイティブの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、トラブルシューティングが可能です。

分散トレースプラットフォームを使用すると、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

分散トレーシングプラットフォームは、Red Hat build of OpenTelemetry と組み合わせて使用できます。

1.2.2. Red Hat OpenShift 分散トレーシングプラットフォーム 3.2 のリリースノート

Red Hat OpenShift 分散トレーシングプラットフォームのこのリリースには、Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) と非推奨の Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) が含まれています。

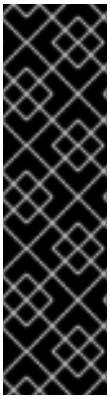
1.2.2.1. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)

Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) は、Tempo Operator を通じて提供されます。

1.2.2.1.1. テクノロジープレビュー機能

この更新では、次のテクノロジープレビュー機能が導入されています。

- Tempo モノリシックデプロイメントのサポート。



重要

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

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

1.2.2.1.2. 新機能および機能拡張

この更新では、次の機能拡張が導入されています。

- Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) 3.2 は、オープンソースの Grafana Tempo 2.4.1 をベースにしています。
- コンポーネントごとのリソースのオーバーライドが可能です。

1.2.2.1.3. バグ修正

この更新では、次のバグ修正が導入されています。

- この更新前は、Jaeger UI に表示されるサービスが、過去 15 分間にトレースを送信したサービスだけでした。この更新により、利用できるサービス名と操作名を、**spec.template.queryFrontend.jaegerQuery.servicesQueryDuration** フィールドを使用して設定できるようになりました。(TRACING-3139)
- この更新前は、大規模なトレースを検索した結果、メモリー不足 (OOM) になったときに、**query-frontend** Pod が停止する可能性がありました。この更新により、この問題を防ぐためにリソース制限を設定できるようになりました。(TRACING-4009)

1.2.2.1.4. 既知の問題

現在、次のような既知の問題があります。

- 現在、IBM Z (**s390x**) アーキテクチャーでは、distributed tracing platform (Tempo) が失敗します。(TRACING-3545)

1.2.2.2. Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、Red Hat OpenShift 分散トレーシングプラットフォーム Operator を通じて提供されます。

重要

Jaeger は、FIPS 検証済みの暗号化モジュールを使用しません。

1.2.2.2.1. OpenShift Elasticsearch Operator のサポート

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) 3.2 は、OpenShift Elasticsearch Operator 5.6、5.7、および 5.8 での使用がサポートされています。

1.2.2.2.2. 非推奨になった機能

Red Hat OpenShift 分散トレーシングプラットフォーム 3.2 では、引き続き Jaeger と Elasticsearch のサポートが非推奨となります。どちらも今後のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクル中、これらのコンポーネントのサポートと、重大度が重大以上の CVE およびバグに対する修正は提供しますが、これらのコンポーネントの機能拡張は提供しません。Tempo Operator と Red Hat build of OpenTelemetry が、分散トレーシングの収集と保存に推奨される Operator です。OpenTelemetry および Tempo 分散トレーシングスタックは、今後強化されるスタックであるため、ユーザーはこのスタックを採用する必要があります。

Red Hat OpenShift 分散トレーシングプラットフォーム 3.2 では、Jaeger エージェントが非推奨となりました。Jaeger エージェントは次のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクル中、Jaeger エージェントのバグ修正とサポートは提供しますが、機能拡張は提供せず、Jaeger エージェントを削除する予定です。Red Hat build of OpenTelemetry によって提供される OpenTelemetry Collector が、トレースコレクターエージェントの注入に推奨される Operator です。

1.2.2.2.3. 新機能および機能拡張

この更新では、分散トレーシングプラットフォーム (Jaeger) に次の機能拡張が導入されました。

- Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) 3.2 は、オープンソースの Jaeger リリース 1.57.0 をベースにしています。

1.2.2.2.4. 既知の問題

現在、次のような既知の問題があります。

- 現在、Apache Spark はサポートされていません。
- 現在、AMQ/Kafka 経由のストリーミングデプロイメントは、{ibm-z-title} および {ibm-power-title} アーキテクチャーではサポートされていません。

1.2.3. Red Hat OpenShift distributed tracing platform 3.1.1 のリリースノート

Red Hat OpenShift 分散トレーシングプラットフォームのこのリリースには、Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) と非推奨の Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) が含まれています。

1.2.3.1. CVE

このリリースでは、CVE-2023-39326 が修正されています。

1.2.3.2. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)

Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) は、Tempo Operator を通じて提供されます。

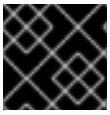
1.2.3.2.1. 既知の問題

現在、次のような既知の問題があります。

- 現在、Tempo Operator と併用すると、Jaeger UI には過去 15 分間にトレースを送信したサービスのみが表示されます。過去 15 分間にトレースを送信していないサービスの場合、トレースは保存されますが、Jaeger UI には表示されません。(TRACING-3139)
- 現在、IBM Z (**s390x**) アーキテクチャーでは、distributed tracing platform (Tempo) が失敗します。(TRACING-3545)

1.2.3.3. Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、Red Hat OpenShift 分散トレーシングプラットフォーム Operator を通じて提供されます。



重要

Jaeger は、FIPS 検証済みの暗号化モジュールを使用しません。

1.2.3.3.1. OpenShift Elasticsearch Operator のサポート

Red Hat OpenShift distributed tracing platform (Jaeger) 3.1.1 は、OpenShift Elasticsearch Operator 5.6、5.7、および 5.8 での使用がサポートされています。

1.2.3.3.2. 非推奨になった機能

Red Hat OpenShift distributed tracing platform 3.1.1 では、引き続き Jaeger と Elasticsearch のサポートが非推奨となり、どちらも今後のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクルにおいて、該当コンポーネントの「重大」以上の CVE に対するバグ修正とサポートを提供しますが、機能拡張は提供しません。

Red Hat OpenShift distributed tracing platform 3.1.1 では、Tempo Operator によって提供される Tempo と、Red Hat build of OpenTelemetry によって提供される OpenTelemetry Collector が、分散トレーシングの収集および保存に推奨される Operator です。OpenTelemetry および Tempo 分散トレーシングスタックは、今後の強化対象スタックとなっているため、すべてのユーザーが採用する必要があります。

1.2.3.3.3. 既知の問題

現在、次のような既知の問題があります。

- 現在、Apache Spark はサポートされていません。
- 現在、AMQ/Kafka 経由のストリーミングデプロイメントは、{ibm-z-title} および {ibm-power-title} アーキテクチャーではサポートされていません。

1.2.4. Red Hat OpenShift distributed tracing platform 3.1 のリリースノート

Red Hat OpenShift 分散トレーシングプラットフォームのこのリリースには、Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) と非推奨の Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) が含まれています。

1.2.4.1. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)

Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) は、Tempo Operator を通じて提供されます。

1.2.4.1.1. 新機能および機能拡張

この更新では、分散トレーシングプラットフォーム (Tempo) に次の機能拡張が導入されました。

- Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) 3.1 は、オープンソースの Grafana Tempo 2.3.1 をベースにしています。
- クラスター全体のプロキシー環境のサポート
- Gateway コンポーネントでの TraceQL のサポート

1.2.4.1.2. バグ修正

この更新では、分散トレーシングプラットフォーム (Tempo) の次のバグ修正が導入されています。

- この更新より前は、OpenShift Container Platform 4.15 で **monitorTab** を有効にして TempoStack インスタンスを作成した場合、必要な **tempo-redmetrics-cluster-monitoring-view** ClusterRoleBinding が作成されませんでした。この更新により、Operator が任意の namespace にデプロイされているときの Monitor タブの Operator RBAC が修正され、問題が解決されます。(TRACING-3786)
- この更新より前は、IPv6 ネットワークスタックのみを備えた OpenShift Container Platform クラスター上に TempoStack インスタンスが作成された場合、コンパクター Pod とインジェスター Pod が **CrashLoopBackOff** 状態で実行され、複数のエラーが発生していました。この更新では、IPv6 クラスターのサポートが提供されます。(TRACING-3226)

1.2.4.1.3. 既知の問題

現在、次のような既知の問題があります。

- 現在、Tempo Operator と併用すると、Jaeger UI には過去 15 分間にトレースを送信したサービスのみが表示されます。過去 15 分間にトレースを送信していないサービスの場合、トレースは保存されますが、Jaeger UI には表示されません。(TRACING-3139)
- 現在、IBM Z (**s390x**) アーキテクチャーでは、distributed tracing platform (Tempo) が失敗します。(TRACING-3545)

1.2.4.2. Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、Red Hat OpenShift 分散トレーシングプラットフォーム Operator を通じて提供されます。

重要

Jaeger は、FIPS 検証済みの暗号化モジュールを使用しません。

1.2.4.2.1. OpenShift Elasticsearch Operator のサポート

Red Hat OpenShift distributed tracing platform (Jaeger) 3.1 は、OpenShift Elasticsearch Operator 5.6、5.7、および 5.8 での使用がサポートされています。

1.2.4.2.2. 非推奨になった機能

Red Hat OpenShift distributed tracing platform 3.1 では、引き続き Jaeger と Elasticsearch のサポートが非推奨となり、どちらも今後のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクルにおいて、該当コンポーネントの「重大」以上の CVE に対するバグ修正とサポートを提供しますが、機能拡張は提供しません。

Red Hat OpenShift distributed tracing platform 3.1 では、Tempo Operator によって提供される Tempo と、Red Hat build of OpenTelemetry によって提供される OpenTelemetry Collector が、分散トレーシングの収集および保存に推奨される Operator です。OpenTelemetry および Tempo 分散トレーシングスタックは、今後の強化対象スタックとなっているため、すべてのユーザーが採用する必要があります。

1.2.4.2.3. 新機能および機能拡張

この更新では、分散トレーシングプラットフォーム (Jaeger) に次の機能拡張が導入されました。

- Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) 3.1 は、オープンソースの Jaeger リリース 1.53.0 に基づいています。

1.2.4.2.4. バグ修正

この更新では、分散トレーシングプラットフォーム (Jaeger) の次のバグ修正が導入されています。

- この更新より前は、**jager-query** Pod 内の **jaeger-agent** コンテナーの接続ターゲット URL が、OpenShift Container Platform 4.13 の別の namespace URL で上書きされていました。これは、**jaeger-operator** のサイドカーインジェクションコードのバグにより、非決定的な **jaeger-agent** インジェクションが発生することが原因でした。この更新により、ターゲットデプロイメントと同じ namespace からの Jaeger インスタンスを Operator が優先するようになりました。(TRACING-3722)

1.2.4.2.5. 既知の問題

現在、次のような既知の問題があります。

- 現在、Apache Spark はサポートされていません。
- 現在、AMQ/Kafka 経由のストリーミングデプロイメントは、{ibm-z-title} および {ibm-power-title} アーキテクチャーではサポートされていません。

1.2.5. Red Hat OpenShift distributed tracing platform 3.0 のリリースノート

1.2.5.1. Red Hat OpenShift distributed tracing platform 3.0 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.51.0
Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)	Tempo	2.3.0

1.2.5.2. Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

1.2.5.2.1. 非推奨になった機能

Red Hat OpenShift distributed tracing platform 3.0 では、Jaeger と Elasticsearch のサポートが非推奨となりました。どちらも今後のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクルにおいて、該当コンポーネントの「重大」以上の CVE に対するバグ修正とサポートを提供しますが、機能拡張は提供しません。

Red Hat OpenShift distributed tracing platform 3.0 では、Tempo Operator によって提供される Tempo と、Red Hat build of OpenTelemetry によって提供される OpenTelemetry Collector が、分散トレーシングの収集および保存に推奨される Operator です。OpenTelemetry および Tempo 分散トレーシングスタックは、今後の強化対象スタックとなっているため、すべてのユーザーが採用する必要があります。

1.2.5.2.2. 新機能および機能拡張

この更新では、分散トレーシングプラットフォーム (Jaeger) に次の機能拡張が導入されました。

- ARM アーキテクチャーのサポート。
- クラスター全体のプロキシー環境のサポート

1.2.5.2.3. バグ修正

この更新では、分散トレーシングプラットフォーム (Jaeger) の次のバグ修正が導入されています。

- この更新より前は、Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator が **relationshipImages** 以外のイメージを使用していました。そのため、非接続ネットワーク環境で **jaeger** Pod を起動するときに **ImagePullBackOff** エラーが発生していました。これは、**oc adm catalog mirror** コマンドが **relationshipImages** で指定されたイメージをミラーリングするためです。この更新により、**oc adm category mirror** CLI コマンドを使用する場合に非接続環境がサポートされるようになりました。(TRACING-3546)

1.2.5.2.4. 既知の問題

現在、次のような既知の問題があります。

- 現在、Apache Spark はサポートされていません。
- 現在、AMQ/Kafka 経由のストリーミングデプロイメントは、{ibm-z-title} および {ibm-power-title} アーキテクチャーではサポートされていません。

1.2.5.3. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)

1.2.5.3.1. 新機能および機能拡張

この更新では、分散トレーシングプラットフォーム (Tempo) に次の機能拡張が導入されました。

- ARM アーキテクチャーのサポート。
- スパン要求数、期間、およびエラー数 (RED) メトリクスのサポート。メトリクスは、Tempo の一部としてデプロイされた Jaeger コンソール、または Web コンソールの **Observe** メニューで表示できます。

1.2.5.3.2. バグ修正

この更新では、分散トレーシングプラットフォーム (Tempo) の次のバグ修正が導入されています。

- この更新より前は、CA 証明書を選択するオプションがあるにもかかわらず、**TempoStack** CRD がカスタム CA 証明書を受け入れませんでした。この更新により、オブジェクトストレージに接続するためのカスタム TLS CA オプションのサポートが修正されました。(TRACING-3462)
- この更新より前は、非接続クラスターで使用するために Red Hat OpenShift 分散トレーシングプラットフォームの Operator イメージをミラーレジストリーにミラーリングする場合、**tempo**、**tempo-gateway**、**opa-openshift**、および **tempo-query** に関連する Operator イメージがミラーリングされませんでした。この更新により、**oc adm catalog mirror** CLI コマンドを使用する場合の非接続環境のサポートが修正されました。(TRACING-3523)
- この更新より前は、ゲートウェイがデプロイされていない場合、Red Hat OpenShift 分散トレーシングプラットフォームのクエリーフロントエンドサービスが内部 mTLS を使用していました。これにより、エンドポイント障害エラーが発生していました。この更新により、ゲートウェイがデプロイされていない場合の mTLS が修正されました。(TRACING-3510)

1.2.5.3.3. 既知の問題

現在、次のような既知の問題があります。

- 現在、Tempo Operator と併用すると、Jaeger UI には過去 15 分間にトレースを送信したサービスのみが表示されます。過去 15 分間にトレースを送信していないサービスの場合、トレースは保存されますが、Jaeger UI には表示されません。(TRACING-3139)
- 現在、IBM Z (**s390x**) アーキテクチャーでは、distributed tracing platform (Tempo) が失敗します。(TRACING-3545)

1.2.6. Red Hat OpenShift distributed tracing platform 2.9.2 のリリースノート

1.2.6.1. Red Hat OpenShift distributed tracing platform 2.9.2 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.47.0

Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)	Tempo	2.1.1

1.2.6.2. CVE

このリリースでは、CVE-2023-46234 が修正されています。

1.2.6.3. Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

1.2.6.3.1. 既知の問題

現在、次のような既知の問題があります。

- Apache Spark はサポートされていません。
- AMQ/Kafka 経由のストリーミングデプロイメントは、{ibm-z-title} および {ibm-power-title} アーキテクチャーではサポートされません。

1.2.6.4. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)

重要

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

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

1.2.6.4.1. 既知の問題

現在、次のような既知の問題があります。

- 現在、オブジェクトストレージに接続するためのカスタム TLS CA オプションは実装されていません。(TRACING-3462)
- 現在、Tempo Operator と併用すると、Jaeger UI には過去 15 分間にトレースを送信したサービスのみが表示されます。過去 15 分間にトレースを送信していないサービスの場合、トレースは保存されますが、Jaeger UI には表示されません。(TRACING-3139)
- 現在、IBM Z (**s390x**) アーキテクチャーでは、distributed tracing platform (Tempo) が失敗します。(TRACING-3545)
- 現在、ゲートウェイがデプロイされていない場合、Tempo クエリーフロントエンドサービスは内部 mTLS を使用できません。この問題は Jaeger Query API には影響しません。回避策としては、mTLS を無効にします。(TRACING-3510)

  回避策

  次のように mTLS を無効にします。

1. 以下のコマンドを実行して、編集するために Tempo Operator ConfigMap を開きます。

```
$ oc edit configmap tempo-operator-manager-config -n openshift-tempo-operator (1)
```

(1) Tempo Operator がインストールされているプロジェクトです。

2. YAML ファイルを更新して、Operator 設定で mTLS を無効にします。

```
data:
  controller_manager_config.yaml: |
    featureGates:
      httpEncryption: false
      grpcEncryption: false
      builtInCertManagement:
        enabled: false
```

3. 以下のコマンドを実行して Tempo Operator Pod を再起動します。

```
$ oc rollout restart deployment.apps/tempo-operator-controller -n openshift-tempo-operator
```

- 制限された環境で Tempo Operator を実行するためのイメージがありません。Red Hat OpenShift distributed tracing platform (Tempo) CSV に、オペランドイメージへの参照がありません。(TRACING-3523)

回避策

ミラーリングツールに Tempo Operator 関連のイメージを追加して、イメージをレジストリーにミラーリングします。

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
archiveSize: 20
storageConfig:
  local:
    path: /home/user/images
mirror:
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.13
    packages:
    - name: tempo-product
      channels:
      - name: stable
  additionalImages:
  - name: registry.redhat.io/rhosdt/tempo-rhel8@sha256:e4295f837066efb05bcc5897f31eb2bdbd81684a8c59d6f9498dd3590c62c12a
  - name: registry.redhat.io/rhosdt/tempo-gateway-rhel8@sha256:b62f5cedfeb5907b638f14ca6aaeea50f41642980a8a6f87b7061e88d90fac23
  - name: registry.redhat.io/rhosdt/tempo-gateway-opa-rhel8@sha256:8cd134deca47d6817b26566e272e6c3f75367653d589f5c90855c59b2fab01e9

  - name: registry.redhat.io/rhosdt/tempo-query-rhel8@sha256:0da43034f440b8258a48a0697ba643b5643d48b615cdb882ac7f4f1f80aad08e
```

1.2.7. Red Hat OpenShift distributed tracing platform 2.9.1 のリリースノート

1.2.7.1. Red Hat OpenShift distributed tracing platform 2.9.1 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.47.0
Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)	Tempo	2.1.1

1.2.7.2. CVE

このリリースでは、CVE-2023-44487 が修正されています。

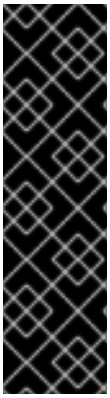
1.2.7.3. Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

1.2.7.3.1. 既知の問題

現在、次のような既知の問題があります。

- Apache Spark はサポートされていません。
- AMQ/Kafka 経由のストリーミングデプロイメントは、{ibm-z-title} および {ibm-power-title} アーキテクチャーではサポートされません。

1.2.7.4. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)



重要

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

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

1.2.7.4.1. 既知の問題

現在、次のような既知の問題があります。

- 現在、オブジェクトストレージに接続するためのカスタム TLS CA オプションは実装されていません。(TRACING-3462)
- 現在、Tempo Operator と併用すると、Jaeger UI には過去 15 分間にトレースを送信したサービスのみが表示されます。過去 15 分間にトレースを送信していないサービスの場合、トレースは保存されますが、Jaeger UI には表示されません。(TRACING-3139)

- 現在、IBM Z (**s390x**) アーキテクチャーでは、distributed tracing platform (Tempo) が失敗します。(TRACING-3545)

- 現在、ゲートウェイがデプロイされていない場合、Tempo クエリーフロントエンドサービスは内部 mTLS を使用できません。この問題は Jaeger Query API には影響しません。回避策としては、mTLS を無効にします。(TRACING-3510)

  回避策

  次のように mTLS を無効にします。

  1. 以下のコマンドを実行して、編集するために Tempo Operator ConfigMap を開きます。

     ```
     $ oc edit configmap tempo-operator-manager-config -n openshift-tempo-operator 1
     ```

     1 Tempo Operator がインストールされているプロジェクトです。

  2. YAML ファイルを更新して、Operator 設定で mTLS を無効にします。

     ```
     data:
       controller_manager_config.yaml: |
         featureGates:
           httpEncryption: false
           grpcEncryption: false
           builtInCertManagement:
             enabled: false
     ```

  3. 以下のコマンドを実行して Tempo Operator Pod を再起動します。

     ```
     $ oc rollout restart deployment.apps/tempo-operator-controller -n openshift-tempo-operator
     ```

- 制限された環境で Tempo Operator を実行するためのイメージがありません。Red Hat OpenShift distributed tracing platform (Tempo) CSV に、オペランドイメージへの参照がありません。(TRACING-3523)

  回避策

  ミラーリングツールに Tempo Operator 関連のイメージを追加して、イメージをレジストリーにミラーリングします。

  ```
  kind: ImageSetConfiguration
  apiVersion: mirror.openshift.io/v1alpha2
  archiveSize: 20
  storageConfig:
    local:
      path: /home/user/images
  mirror:
    operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.13
      packages:
      - name: tempo-product
        channels:
        - name: stable
    additionalImages:
  ```

```
  - name: registry.redhat.io/rhosdt/tempo-
rhel8@sha256:e4295f837066efb05bcc5897f31eb2bdbd81684a8c59d6f9498dd3590c62c12a
  - name: registry.redhat.io/rhosdt/tempo-gateway-
rhel8@sha256:b62f5cedfeb5907b638f14ca6aaeea50f41642980a8a6f87b7061e88d90fac23
  - name: registry.redhat.io/rhosdt/tempo-gateway-opa-
rhel8@sha256:8cd134deca47d6817b26566e272e6c3f75367653d589f5c90855c59b2fab01e9

  - name: registry.redhat.io/rhosdt/tempo-query-
rhel8@sha256:0da43034f440b8258a48a0697ba643b5643d48b615cdb882ac7f4f1f80aad08e
```

1.2.8. Red Hat OpenShift distributed tracing platform 2.9 のリリースノート

1.2.8.1. Red Hat OpenShift distributed tracing platform 2.9 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.47.0
Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)	Tempo	2.1.1

1.2.8.2. Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

1.2.8.2.1. バグ修正

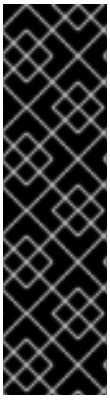
- この更新以前は、**jaeger-query** デプロイメントに gRPC ポートがないため、接続が拒否されていました。その結果、**transport: Error while dialing: dial tcp :16685: connect: connection refused** エラーメッセージが表示されていました。今回の更新により、Jaeger Query gRPC ポート (16685) が、Jaeger Query サービスで正常に公開されるようになりました。(TRACING-3322)
- 今回の更新以前は、**jaeger-production-query** で誤ったポートが公開されていたため、接続が拒否されていました。今回の更新では問題が修正され、Jaeger Query デプロイメントで Jaeger Query gRPC ポート (16685) が公開されています。(TRACING-2968)
- この更新以前は、非接続環境のシングルノード OpenShift クラスターに Service Mesh をデプロイすると、Jaeger Pod が頻繁に **Pending** 状態になりました。この更新により、問題が修正されました。(TRACING-3312)
- 今回の更新以前は、**OOMKilled** エラーメッセージが原因で、Jaeger Operator Pod はデフォルトのメモリー値で再起動されていました。今回の更新で、この問題はリソース制限を削除することで修正されています。(TRACING-3173)

1.2.8.2.2. 既知の問題

現在、次のような既知の問題があります。

- Apache Spark はサポートされていません。
- AMQ/Kafka 経由のストリーミングデプロイメントは、{ibm-z-title} および {ibm-power-title} アーキテクチャーではサポートされません。

1.2.8.3. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)



重要

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

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

1.2.8.3.1. 新機能および機能拡張

このリリースでは、distributed tracing platform (Tempo) に次の機能拡張が導入されました。

- Operator 成熟度 レベル IV、Deep Insights のサポート。これにより、TempoStack インスタンスと Tempo Operator のアップグレード、監視、アラートが可能になります。
- ゲートウェイの Ingress および Route 設定を追加します。
- **TempoStack** カスタムリソースの **managed** および **unmanaged** の状態をサポートします。
- Distributor サービスで、追加の取り込みプロトコル (Jaeger Thrift バイナリー、Jaeger Thrift コンパクト、Jaeger gRPC、Zipkin) を公開します。ゲートウェイが有効になっている場合は、OpenTelemetry プロトコル (OTLP) gRPC のみが有効になります。
- Query Frontend サービスで Jaeger Query gRPC エンドポイントを公開します。
- ゲートウェイの認証および認可なしでマルチテナンシーをサポートします。

1.2.8.3.2. バグ修正

- この更新以前は、Tempo Operator は非接続環境と互換性がありませんでした。今回の更新により、Tempo Operator は非接続環境をサポートするようになりました。(TRACING-3145)
- この更新以前は、TLS を使用する Tempo Operator を OpenShift Container Platform で起動できませんでした。今回の更新により、Tempo コンポーネント間で mTLS 通信が有効になり、Operand が正常に起動し、Jaeger UI にアクセスできるようになりました。(TRACING-3091)
- この更新以前は、Tempo Operator からのリソース制限により、**reason: OOMKilled** などのエラーメッセージが表示されていました。今回の更新では、このようなエラーを回避するために、Tempo Operator のリソース制限が削除されました。(TRACING-3204)

1.2.8.3.3. 既知の問題

現在、次のような既知の問題があります。

- 現在、オブジェクトストレージに接続するためのカスタム TLS CA オプションは実装されていません。(TRACING-3462)
- 現在、Tempo Operator と併用すると、Jaeger UI には過去 15 分間にトレースを送信したサービスのみが表示されます。過去 15 分間にトレースを送信していないサービスの場合、トレースは保存されますが、Jaeger UI には表示されません。(TRACING-3139)

- 現在、IBM Z (**s390x**) アーキテクチャーでは、distributed tracing platform (Tempo) が失敗します。(TRACING-3545)

- 現在、ゲートウェイがデプロイされていない場合、Tempo クエリーフロントエンドサービスは内部 mTLS を使用できません。この問題は Jaeger Query API には影響しません。回避策としては、mTLS を無効にします。(TRACING-3510)

  回避策

  次のように mTLS を無効にします。

  1. 以下のコマンドを実行して、編集するために Tempo Operator ConfigMap を開きます。

     ```
     $ oc edit configmap tempo-operator-manager-config -n openshift-tempo-operator 1
     ```

     1 Tempo Operator がインストールされているプロジェクトです。

  2. YAML ファイルを更新して、Operator 設定で mTLS を無効にします。

     ```
     data:
       controller_manager_config.yaml: |
         featureGates:
           httpEncryption: false
           grpcEncryption: false
           builtInCertManagement:
             enabled: false
     ```

  3. 以下のコマンドを実行して Tempo Operator Pod を再起動します。

     ```
     $ oc rollout restart deployment.apps/tempo-operator-controller -n openshift-tempo-operator
     ```

- 制限された環境で Tempo Operator を実行するためのイメージがありません。Red Hat OpenShift distributed tracing platform (Tempo) CSV に、オペランドイメージへの参照がありません。(TRACING-3523)

  回避策

  ミラーリングツールに Tempo Operator 関連のイメージを追加して、イメージをレジストリーにミラーリングします。

  ```
  kind: ImageSetConfiguration
  apiVersion: mirror.openshift.io/v1alpha2
  archiveSize: 20
  storageConfig:
    local:
      path: /home/user/images
  mirror:
    operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.13
      packages:
      - name: tempo-product
        channels:
        - name: stable
    additionalImages:
  ```

```
  - name: registry.redhat.io/rhosdt/tempo-
rhel8@sha256:e4295f837066efb05bcc5897f31eb2bdbd81684a8c59d6f9498dd3590c62c12a
  - name: registry.redhat.io/rhosdt/tempo-gateway-
rhel8@sha256:b62f5cedfeb5907b638f14ca6aaeea50f41642980a8a6f87b7061e88d90fac23
  - name: registry.redhat.io/rhosdt/tempo-gateway-opa-
rhel8@sha256:8cd134deca47d6817b26566e272e6c3f75367653d589f5c90855c59b2fab01e9

  - name: registry.redhat.io/rhosdt/tempo-query-
rhel8@sha256:0da43034f440b8258a48a0697ba643b5643d48b615cdb882ac7f4f1f80aad08e
```

1.2.9. Red Hat OpenShift distributed tracing platform 2.8 のリリースノート

1.2.9.1. Red Hat OpenShift distributed tracing platform 2.8 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.42
Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)	Tempo	0.1.0

1.2.9.2. テクノロジープレビュー機能

このリリースでは、Red Hat OpenShift distributed tracing platform (Tempo) のサポートが、Red Hat OpenShift distributed tracing platform の テクノロジープレビュー 機能として導入されています。

重要

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

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

この機能は、Red Hat OpenShift distributed tracing platform (Tempo) のバージョン 0.1.0 と、アップストリームの distributed tracing platform (Tempo) コンポーネントのバージョン 2.0.1 を使用します。

distributed tracing platform (Tempo) を使用して Jaeger を置き換え、ElasticSearch の代わりに S3 互換ストレージを使用できます。distributed tracing platform (Tempo) は Jaeger と同じ取り込みおよびクエリープロトコルをサポートし、同じユーザーインターフェイスを使用するため、Jaeger の代わりに distributed tracing platform (Tempo) を使用するほとんどのユーザーは機能の違いに気付きません。

このテクノロジープレビュー機能を有効にする場合は、現在の実装における以下の制限に注意してください。

- 現在、distributed tracing platform (Tempo) は非接続インストールをサポートしていません。(TRACING-3145)

- distributed tracing platform (Tempo) で Jaeger ユーザーインターフェイス (UI) を使用すると、Jaeger UI は過去 15 分以内にトレースを送信したサービスのみを一覧表示します。過去 15 分以内にトレースを送信していないサービスの場合、トレースは Jaeger UI に表示されませんが、保存はされます。(TRACING-3139)

Red Hat OpenShift distributed tracing platform の今後のリリースでは、Tempo Operator のサポートを拡張することが予定されています。追加される可能性のある機能には、TLS 認証、マルチテナンシー、複数クラスターのサポートが含まれます。Tempo Operator の詳細は、Tempo コミュニティーのドキュメント を参照してください。

1.2.9.3. バグ修正

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

1.2.10. Red Hat OpenShift distributed tracing platform 2.7 のリリースノート

1.2.10.1. Red Hat OpenShift distributed tracing platform 2.7 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.39

1.2.10.2. バグ修正

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

1.2.11. Red Hat OpenShift distributed tracing platform 2.6 のリリースノート

1.2.11.1. Red Hat OpenShift distributed tracing platform 2.6 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.38

1.2.11.2. バグ修正

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

1.2.12. Red Hat OpenShift distributed tracing platform 2.5 のリリースノート

1.2.12.1. Red Hat OpenShift distributed tracing platform 2.5 のコンポーネントバージョン

Operator	コンポーネント	バージョン

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.36

1.2.12.2. 新機能および機能拡張

このリリースでは、OpenTelemetry プロトコル (OTLP) を Red Hat OpenShift distributed tracing platform (Jaeger) Operator に取り込むためのサポートが導入されています。Operator は OTLP ポートを自動的に有効にするようになりました。

- OTLP gRPC プロトコル用のポート 4317。
- OTLP HTTP プロトコル用のポート 4318。

このリリースでは、Red Hat build of OpenTelemetry Operator に Kubernetes リソース属性を収集するためのサポートも追加されています。

1.2.12.3. バグ修正

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

1.2.13. Red Hat OpenShift distributed tracing platform 2.4 のリリースノート

1.2.13.1. Red Hat OpenShift distributed tracing platform 2.4 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.34.1

1.2.13.2. 新機能および機能拡張

このリリースでは、OpenShift Elasticsearch Operator を使用した証明書の自動プロビジョニングのサポートが追加されています。

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator を使用して、インストール中に OpenShift Elasticsearch Operator を呼び出すセルフプロビジョニング。

+

重要

Red Hat OpenShift distributed tracing platform 2.4 にアップグレードする場合、Operator は Elasticsearch インスタンスを再作成します。これには 5 - 10 分かかる場合があります。その期間、分散トレースは停止し、使用できなくなります。

1.2.13.3. テクノロジープレビュー機能

このリリースの テクノロジープレビュー 機能として、Elasticsearch インスタンスと証明書を作成してから証明書を使用するように distributed tracing platform (Jaeger) を設定できます。

1.2.13.4. バグ修正

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

1.2.14. Red Hat OpenShift distributed tracing platform 2.3 のリリースノート

1.2.14.1. Red Hat OpenShift distributed tracing platform 2.3.1 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.30.2

1.2.14.2. Red Hat OpenShift distributed tracing platform 2.3.0 のコンポーネントバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.30.1

1.2.14.3. 新機能および機能拡張

このリリースでは、Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator が、デフォルトで **openshift-distributed-tracing** namespace にインストールされるようになりました。この更新の前は、デフォルトのインストールは **openshift-operators** namespace にありました。

1.2.14.4. バグ修正

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

1.2.15. Red Hat OpenShift distributed tracing platform 2.2 のリリースノート

1.2.15.1. テクノロジープレビュー機能

2.1 リリースに含まれるサポート対象外の OpenTelemetry Collector コンポーネントが削除されました。

1.2.15.2. バグ修正

この Red Hat OpenShift distributed tracing platform のリリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.16. Red Hat OpenShift distributed tracing platform 2.1 のリリースノート

1.2.16.1. Red Hat OpenShift distributed tracing platform 2.1 のコンポーネントのバージョン

Operator	コンポーネント	バージョン

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.29.1

1.2.16.2. テクノロジープレビュー機能

- 本リリースでは、OpenTelemetry カスタムリソースファイルで証明書を設定する方法に重大な変更が加えられました。次の例に示すように、今回の更新では **ca_file** がカスタムリソースの **tls** の下に移動しました。

  OpenTelemetry バージョン 0.33 の CA ファイル設定

```
spec:
  mode: deployment
  config: |
    exporters:
      jaeger:
        endpoint: jaeger-production-collector-headless.tracing-system.svc:14250
        ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
```

  OpenTelemetry バージョン 0.41.1 の CA ファイル設定

```
spec:
  mode: deployment
  config: |
    exporters:
      jaeger:
        endpoint: jaeger-production-collector-headless.tracing-system.svc:14250
        tls:
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
```

1.2.16.3. バグ修正

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

1.2.17. Red Hat OpenShift distributed tracing platform 2.0 のリリースノート

1.2.17.1. Red Hat OpenShift distributed tracing platform 2.0 のコンポーネントのバージョン

Operator	コンポーネント	バージョン
Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)	Jaeger	1.28.0

1.2.17.2. 新機能および機能拡張

このリリースでは、以下の新機能および機能拡張が導入されました。

- Red Hat OpenShift Jaeger が Red Hat OpenShift distributed tracing platform としてリブランディングされました。
- Red Hat OpenShift distributed tracing platform が Jaeger 1.28 に更新されました。今後、Red Hat OpenShift distributed tracing platform は **stable** Operator チャネルのみサポートします。個別リリースのチャネルはサポートされなくなりました。
- OpenTelemetry プロトコル (OTLP) のサポートを Query サービスに追加されました。
- OperatorHub に表示される新しい分散トレースアイコンが導入されました。
- 名前の変更および新機能に対応するためのドキュメントへのローリング更新が含まれます。

1.2.17.3. テクノロジープレビュー機能

このリリースでは、Red Hat build of OpenTelemetry Operator を使用してインストールする Red Hat build of OpenTelemetry が、テクノロジープレビュー機能 として追加されます。Red Hat build of OpenTelemetry は、OpenTelemetry API とインストルメンテーションに基づいています。Red Hat build of OpenTelemetry には、OpenTelemetry Operator と Collector が含まれています。Collector を使用して、OpenTelemetry または Jaeger プロトコルでトレースを受信し、そのトレースデータを Red Hat OpenShift distributed tracing platform に送信できます。現時点では、Collector のその他の機能はサポートされていません。OpenTelemetry Collector を使用すると、開発者はベンダーに依存しない API でコードをインストルメント化し、ベンダーのロックインを回避して、可観測性ツールの拡大したエコシステムを有効にできます。

1.2.17.4. バグ修正

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

1.2.18. サポート

本書で説明されている手順、または OpenShift Container Platform で問題が発生した場合は、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では、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションを対象とした 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。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager Hybrid Cloud Console で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

本書の改善への提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントの Jira Issue 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。

1.2.19. 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、用語

の置き換えは、今後の複数のリリースにわたって段階的に実施されます。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第2章 分散トレースのアーキテクチャー

2.1. 分散トレースのアーキテクチャー

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。Red Hat OpenShift distributed tracing platform 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設定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。

Red Hat OpenShift distributed tracing platform は、マイクロサービスのスタック全体における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 つ以上のスパンで設定されます。

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 Red Hat OpenShift distributed tracing platform 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. 分散トレースの概要

サービスの所有者は、分散トレース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。Red Hat OpenShift 分散トレーシングプラットフォームを使用すると、最新のクラウドネイティブの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、トラブルシューティングが可能です。

分散トレースプラットフォームを使用すると、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

2.1.2. Red Hat OpenShift 分散トレーシングプラットフォームの機能

Red Hat OpenShift 分散トレースプラットフォームは、以下の機能を提供します。

- Kiali との統合: 適切に設定されている場合は、Kiali コンソールから分散トレースプラットフォームデータを表示できます。
- 高いスケーラビリティー: 分散トレースプラットフォームのバックエンドは、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できます。
- Zipkin との後方互換性: Red Hat OpenShift 分散トレースプラットフォームには、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、このリリースでは、Red Hat は Zipkin の互換性をサポートしていません。

2.1.3. Red Hat OpenShift 分散トレーシングプラットフォームアーキテクチャー

Red Hat OpenShift 分散トレースプラットフォームは、複数のコンポーネントで設定されており、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。

- **Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo)**: このコンポーネントは、オープンソースの Grafana Tempo プロジェクト に基づいています。
    - **Gateway**: ゲートウェイは、認証、認可、およびディストリビューターまたはクエリーフロントエンドサービスへのリクエストの転送を処理します。
    - **Distributor**: ディストリビューターは、Jaeger、OpenTelemetry、Zipkin などの複数の形式のスパンを受け入れます。**traceID** をハッシュ化し、分散コンシステントハッシュリングを使用して、スパンを Ingester にルーティングします。
    - **Ingester**: Ingester はトレースをブロックにバッチ化し、ブルームフィルターとインデックスを作成してすべてバックエンドにフラッシュします。
    - **Query Frontend**: Query Frontend は、受信クエリーの検索スペースをシャーディングします。次に、検索クエリーが Querier に送信されます。Query Frontend のデプロイメントでは、Tempo Query サイドカーを介して Jaeger UI が公開されます。
    - **Querier**: Querier は、Ingester またはバックエンドストレージで要求されたトレース ID を検索します。パラメーターに応じて、Ingester にクエリーを実行し、バックエンドから Bloom インデックスを取得して、オブジェクトストレージ内のブロックを検索できます。
    - **Compactor**: Compactor は、ブロックをバックエンドストレージとの間でストリーミングして、ブロックの総数を減らします。
- **Red Hat build of OpenTelemetry**- このコンポーネントは、オープンソースの OpenTelemetry プロジェクト に基づいています。
    - **OpenTelemetry Collector**: OpenTelemetry Collector は、テレメトリーデータを受信、処理、エクスポートするためのベンダーに依存しない方法です。OpenTelemetry Collector は、Jaeger や Prometheus などのオープンソースの可観測性データ形式をサポートし、1 つ以上のオープンソースまたは商用バックエンドに送信します。Collector は、インストルメンテーションライブラリーがテレメトリーデータをエクスポートするデフォルトの場所です。
- **Red Hat OpenShift 分散トレースプラットフォーム (Jaeger)**: このコンポーネントは、オープンソースの Jaeger プロジェクト に基づいています。
    - **クライアント** (Jaeger クライアント、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): 分散トレースプラットフォーム (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。
    - **エージェント** (Jaeger エージェント、Server Queue、Processor Worker): 分散トレースプラットフォーム (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、Collector 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** (Collector、Queue、Worker): Jaeger エージェントと同様に、Jaeger Collector はスパンを受信し、これらを処理するために内部キューに配置します。これにより、Jaeger Collector 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。Red Hat OpenShift 分散トレースプラットフォーム (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。Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、Elasticsearch ストレージをサポートしています。

- **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): Red Hat OpenShift 分散トレースプラットフォームは Apache Kafka を Collector と実際の Elasticsearch バッキングストレージ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、Elasticsearch ストレージバックエンドに書き込むサービスです。

- **Jaeger Console**: Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) ユーザーインターフェイスを使用すると、分散トレースデータを可視化できます。検索ページで、トレースを検索し、個別のトレースを設定するスパンの詳細を確認できます。

2.1.4. 関連情報

- Red Hat build of OpenTelemetry

第3章 分散トレーシングプラットフォーム (TEMPO)

3.1. インストール

分散トレーシングプラットフォーム (Tempo) をインストールするには、Tempo Operator が必要です。また、ユースケースに最適なデプロイメントの種類を選択する必要があります。

- マイクロサービスモードの場合は、専用の OpenShift プロジェクトに TempoStack インスタンスをデプロイします。
- モノリシックモードの場合は、専用の OpenShift プロジェクトに TempoMonolithic インスタンスをデプロイします。

重要

オブジェクトストレージを使用するには、TempoStack または TempoMonolithic インスタンスをデプロイする前に、サポートされているオブジェクトストアを設定し、オブジェクトストアの認証情報のシークレットを作成する必要があります。

3.1.1. Tempo Operator のインストール

Tempo Operator は、Web コンソールまたはコマンドラインを使用してインストールできます。

3.1.1.1. Web コンソールを使用した Tempo Operator のインストール

Tempo Operator は、Web コンソールの **Administrator** ビューからインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。
- サポートされているプロバイダーで必要なオブジェクトストレージの設定を完了している ({odf-full}、MinIO、Amazon S3、Azure Blob Storage、Google Cloud Storage)。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

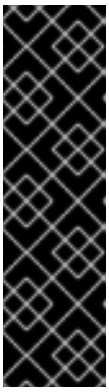
警告

オブジェクトストレージは必須ですが、分散トレーシングプラットフォーム (Tempo) には含まれていません。分散トレーシングプラットフォーム (Tempo) をインストールする前に、サポートされているプロバイダーによるオブジェクトストレージを選択して設定する必要があります。

手順

1. **Operators → OperatorHub** に移動し、**Tempo Operator** を検索します。

2. **Red Hat が提供** する **Tempo Operator** を選択します。



重要

次の選択は、この Operator のデフォルトのプリセットです。

- **Update channel** → **stable**
- **Installation mode** → **All namespaces on the cluster**
- **Installed Namespace** → **openshift-tempo-operator**
- **Update approval** → **Automatic**

3. **Enable Operator recommended cluster monitoring on this Namespace**チェックボックスを選択します。

4. **Install** → **Install** → **View Operator** を選択します。

検証

- インストール済み Operator ページの **Details** タブの **ClusterServiceVersion details** で、インストールの **Status** が **Succeeded** であることを確認します。

3.1.1.2. CLI を使用した Tempo Operator のインストール

Tempo Operator はコマンドラインからインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

ヒント

  - OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
  - **oc login** を実行します。

```
$ oc login --username=<your_username>
```

- サポートされているプロバイダーで必要なオブジェクトストレージの設定を完了している ({odf-full}、MinIO、Amazon S3、Azure Blob Storage、Google Cloud Storage)。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

警告

オブジェクトストレージは必須ですが、分散トレーシングプラットフォーム (Tempo) には含まれていません。分散トレーシングプラットフォーム (Tempo) をインストールする前に、サポートされているプロバイダーによるオブジェクトストレージを選択して設定する必要があります。

手順

1. 以下のコマンドを実行して、Tempo Operator のプロジェクトを作成します。

```
$ oc apply -f - << EOF
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  labels:
    kubernetes.io/metadata.name: openshift-tempo-operator
    openshift.io/cluster-monitoring: "true"
  name: openshift-tempo-operator
EOF
```

2. 以下のコマンドを実行して、Operator グループを作成します。

```
$ oc apply -f - << EOF
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-tempo-operator
  namespace: openshift-tempo-operator
spec:
  upgradeStrategy: Default
EOF
```

3. 以下のコマンドを実行して、サブスクリプションを作成します。

```
$ oc apply -f - << EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: tempo-product
  namespace: openshift-tempo-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: tempo-product
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

検証

- 次のコマンドを実行して、Operator のステータスを確認します。

```
$ oc get csv -n openshift-tempo-operator
```

3.1.2. TempoStack インスタンスのインストール

TempoStack インスタンスは、Web コンソールまたはコマンドラインを使用してインストールできます。

3.1.2.1. Web コンソールを使用した TempoStack インスタンスのインストール

Web コンソールの **Administrator** ビューから TempoStack インスタンスをインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。
- サポートされているプロバイダーで必要なオブジェクトストレージの設定を完了している ({odf-full}、MinIO、Amazon S3、Azure Blob Storage、Google Cloud Storage)。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

警告

オブジェクトストレージは必須ですが、分散トレーシングプラットフォーム (Tempo) には含まれていません。分散トレーシングプラットフォーム (Tempo) をインストールする前に、サポートされているプロバイダーによるオブジェクトストレージを選択して設定する必要があります。

手順

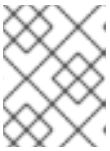
1. **Home → Projects → Create Project** に移動して、後続のステップで作成する TempoStack インスタンス用に任意のプロジェクトを作成します。
2. **Workloads → Secrets → Create → From YAML** に移動して、TempoStack インスタンス用に作成したプロジェクトに、オブジェクトストレージバケットのシークレットを作成します。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

Amazon S3 および MinIO ストレージのシークレット例

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
```

```
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. TempoStack インスタンスを作成します。



注記

同じクラスター上の別々のプロジェクトに、複数の TempoStack インスタンスを作成できます。

a. **Operators → Installed Operators** に移動します。

b. **TempoStack → Create TempoStack → YAML view** の順に選択します。

c. **YAML view** で、**TempoStack** カスタムリソース (CR) をカスタマイズします。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: sample
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage:
    secret: (1)
      name: <secret_name> (2)
      type: <secret_provider> (3)
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

(1) 前提条件の 1 つとして設定したオブジェクトストレージ用に、ステップ 2 で作成したシークレット。

(2) シークレットの **metadata** 内にある **name** の値。

(3) 許可される値は、Azure Blob Storage の場合は **azure**、Google Cloud Storage の場合は **gc**、Amazon S3、MinIO、または {odf-full} の場合は **s3** です。

AWS S3 および MinIO ストレージの TempoStack CR の例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
  namespace: <project_of_tempostack_instance>
spec:
```

```
    storageSize: 1Gi
    storage: 1
      secret:
        name: minio-test
        type: s3
    resources:
      total:
        limits:
          memory: 2Gi
          cpu: 2000m
    template:
      queryFrontend:
        jaegerQuery: 2
          enabled: true
          ingress:
            route:
              termination: edge
            type: route
```

1 この例のオブジェクトストレージは、前提条件の 1 つとして設定したものです。オブジェクトストレージのシークレットは、ステップ 2 で作成したものです。

2 この例にデプロイされたスタックは、HTTP および OpenTelemetry Protocol (OTLP) 経由で Jaeger Thrift を受信するように設定されています。これにより、Jaeger UI でデータが可視化されます。

d. **Create** を選択します。

検証

1. **Project:** ドロップダウンリストを使用して、**TempoStack** インスタンスのプロジェクトを選択します。
2. **Operators → Installed Operators** に移動して、**TempoStack** インスタンスの **Status** が **Condition: Ready** であることを確認します。
3. **Workloads → Pods** に移動して、**TempoStack** インスタンスのすべてのコンポーネント Pod が稼働していることを確認します。
4. Tempo コンソールにアクセスします。
   a. **Networking → Routes** に移動し、**Ctrl+F** で **tempo** を検索します。
   b. **Location** 列で URL を開き、Tempo コンソールにアクセスします。

注記

Tempo コンソールをインストールした直後は、Tempo コンソールにトレースデータは表示されません。

3.1.2.2. CLI を使用した TempoStack インスタンスのインストール

コマンドラインから TempoStack インスタンスをインストールできます。

前提条件

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

  ヒント

  - OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
  - **oc login** コマンドを実行します。

    ```
    $ oc login --username=<your_username>
    ```

- サポートされているプロバイダーで必要なオブジェクトストレージの設定を完了している ({odf-full}、MinIO、Amazon S3、Azure Blob Storage、Google Cloud Storage)。詳細は、「オブジェクトストレージのセットアップ」を参照してください。

警告

オブジェクトストレージは必須ですが、分散トレーシングプラットフォーム (Tempo) には含まれていません。分散トレーシングプラットフォーム (Tempo) をインストールする前に、サポートされているプロバイダーによるオブジェクトストレージを選択して設定する必要があります。

手順

1. 次のコマンドを実行して、後続の手順で作成する TempoStack インスタンス用に選択したプロジェクトを作成します。

   ```
   $ oc apply -f - << EOF
   apiVersion: project.openshift.io/v1
   kind: Project
   metadata:
     name: <project_of_tempostack_instance>
   EOF
   ```

2. 次のコマンドを実行して、TempoStack インスタンス用に作成したプロジェクトでオブジェクトストレージバケットのシークレットを作成します。

   ```
   $ oc apply -f - << EOF
   <object_storage_secret>
   EOF
   ```

   詳細は、「オブジェクトストレージのセットアップ」を参照してください。

   Amazon S3 および MinIO ストレージのシークレット例

   ```
   apiVersion: v1
   kind: Secret
   ```

```
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. TempoStack インスタンス用に作成したプロジェクトに TempoStack インスタンスを作成します。

注記

同じクラスター上の別々のプロジェクトに、複数の TempoStack インスタンスを作成できます。

a. **TempoStack** カスタムリソース (CR) をカスタマイズします。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: sample
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage:
    secret: (1)
      name: <secret_name> (2)
      type: <secret_provider> (3)
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

(1) 前提条件の 1 つとして設定したオブジェクトストレージ用に、ステップ 2 で作成したシークレット。

(2) シークレットの **metadata** 内にある **name** の値。

(3) 許可される値は、Azure Blob Storage の場合は **azure**、Google Cloud Storage の場合は **gc**、Amazon S3、MinIO、または {odf-full} の場合は **s3** です。

AWS S3 および MinIO ストレージの TempoStack CR の例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
```

```
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage: 1
    secret:
      name: minio-test
      type: s3
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  template:
    queryFrontend:
      jaegerQuery: 2
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

1 この例のオブジェクトストレージは、前提条件の 1 つとして設定したものです。オブジェクトストレージのシークレットは、ステップ 2 で作成したものです。

2 この例にデプロイされたスタックは、HTTP および OpenTelemetry Protocol (OTLP) 経由で Jaeger Thrift を受信するように設定されています。これにより、Jaeger UI でデータが可視化されます。

b. 次のコマンドを実行して、カスタマイズされた CR を適用します。

```
$ oc apply -f - << EOF
<tempostack_cr>
EOF
```

検証

1. 次のコマンドを実行して、すべての TempoStack **components** の **status** が **Running**、**conditions** が **type: Ready** になっていることを確認します。

```
$ oc get tempostacks.tempo.grafana.com simplest -o yaml
```

2. 次のコマンドを実行して、すべての TempoStack コンポーネント Pod が稼働していることを確認します。

```
$ oc get pods
```

3. Tempo コンソールにアクセスします。

   a. 以下のコマンドを実行してルートの詳細をクエリーします。

```
$ oc get route
```

   b. Web ブラウザーで **https://<route_from_previous_step>** を開きます。

注記

Tempo コンソールをインストールした直後は、Tempo コンソールにトレースデータは表示されません。

3.1.3. TempoMonolithic インスタンスのインストール

重要

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

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

TempoMonolithic インスタンスは、Web コンソールまたはコマンドラインを使用してインストールできます。

TempoMonolithic カスタムリソース (CR) は、モノリシックモードで Tempo デプロイメントを作成します。コンパクター、ディストリビューター、インジェスター、クエリー、クエリーフロントエンドなど、Tempo デプロイメントのすべてのコンポーネントが、単一のコンテナーに含まれます。

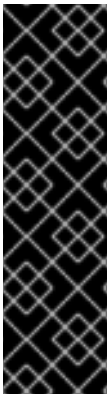
TempoMonolithic インスタンスは、インメモリーストレージ、永続ボリューム、またはオブジェクトストレージへのトレースの保存をサポートしています。

モノリシックモードでの Tempo デプロイメントは、小規模なデプロイメント、デモンストレーション、テスト、および Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) オールインワンデプロイメントの移行パスとして推奨されます。

注記

Tempo のモノリシックデプロイメントは水平方向にスケーリングできません。水平スケーリングが必要な場合は、マイクロサービスモードでの Tempo デプロイメント用の **TempoStack** CR を使用してください。

3.1.3.1. Web コンソールを使用した TempoMonolithic インスタンスのインストール



重要

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

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

Web コンソールの **Administrator** ビューから TempoMonolithic インスタンスをインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。

手順

1. **Home → Projects → Create Project** に移動して、後続のステップで作成する **TempoMonolithic** インスタンス用に任意のプロジェクトを作成します。
2. トレースの保存に使用するサポート対象のストレージのタイプ (インメモリーストレージ、永続ボリューム、オブジェクトストレージ) を決定します。

重要

オブジェクトストレージは分散トレースプラットフォーム(Tempo)に含まれず、サポートされているプロバイダーによるオブジェクトストアを設定する必要があります ({odf-full}、MinIO、Amazon S3、Azure Blob Storage、または Google Cloud Storage)。

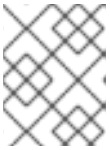
また、オブジェクトストレージを選択するには、**TempoMonolithic** インスタンス用に作成したプロジェクトにオブジェクトストレージバケットのシークレットを作成する必要があります。これは、**Workloads → Secrets → Create → From YAML** で実行できます。

詳細は、「オブジェクトストレージのセットアップ」を参照してください。

Amazon S3 および MinIO ストレージのシークレット例

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. **TempoMonolithic** インスタンスを作成します。



注記

同じクラスター上の別々のプロジェクトに複数の **TempoMonolithic** インスタンスを作成できます。

a. **Operators → Installed Operators** に移動します。

b. **TempoMonolithic → Create TempoMonolithic → YAML view** を選択します。

c. **YAML view** で、**TempoMonolithic** カスタムリソース (CR) をカスタマイズします。
次の **TempoMonolithic** CR は、OTLP/gRPC および OTLP/HTTP 経由のトレース取り込みを備えた TempoMonolithic デプロイメントを作成し、サポートされているタイプのストレージにトレースを保存し、ルート経由で Jaeger UI を公開します。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <metadata_name>
  namespace: <project_of_tempomonolithic_instance>
spec:
  storage:
    traces:
      backend: <supported_storage_type> 1
      size: <value>Gi 2
      s3: 3
        secret: <secret_name> 4
  jaegerui:
    enabled: true 5
    route:
      enabled: true 6
```

1. トレースを保存するストレージのタイプ (インメモリーストレージ、永続ボリューム、またはオブジェクトストレージ)。**tmpfs** インメモリーストレージの値は、**memory** です。永続ボリュームの値は **pv** です。オブジェクトストレージの値は、使用するオブジェクトストアのタイプに応じて、**s3**、**gcs**、または **azure** が受け入れられます。

2. メモリーサイズ: インメモリーストレージの場合、これは **tmpfs** ボリュームのサイズを意味します。デフォルトは **2Gi** です。永続ボリュームの場合、これは永続ボリューム要求のサイズを意味します。デフォルトは **10Gi** です。オブジェクトストレージの場合、これは Tempo WAL の永続ボリューム要求のサイズを意味し、デフォルトは **10Gi** です。

3. オプション: オブジェクトストレージの場合、オブジェクトストレージのタイプ。使用するオブジェクトストアのタイプに応じて、**s3**、**gcs**、および **azure** が値として受け入れられます。

4. オプション: オブジェクトストレージの場合、ストレージシークレットの **metadata** 内の **name** の値。ストレージシークレットは、TempoMonolithic インスタンスと同じ namespace にあり、「表 1. 必要なシークレットパラメーター」(「オブジェクトストレージのセットアップ」セクションを参照) で指定えているフィールドを含んでいる必要があります。

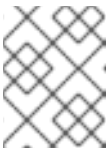
5. Jaeger UI を有効にします。

6. Jaeger UI のルートの作成を有効にします。

d. **Create** を選択します。

検証

1. **Project:** ドロップダウンリストを使用して、**TempoMonolithic** インスタンスのプロジェクトを選択します。
2. **Operator → Installed Operator** に移動して、**TempoMonolithic** インスタンスの **Status** が **Condition: Ready** であることを確認します。
3. **Workloads → Pod** に移動して、**TempoMonolithic** インスタンスの Pod が実行中であることを確認します。
4. Jaeger UI にアクセスします。
   a. **Networking → Routes** に移動し、**Ctrl+F** を押して **jaegerui** を検索します。



注記

Jaeger UI は、**tempo-<metadata_name_of_TempoMonolithic_CR>-jaegerui** ルートを使用します。

   b. **Location** 列で URL を開き、Jaeger UI にアクセスします。
5. **TempoMonolithic** インスタンスの Pod の準備ができたら、クラスター内の **tempo-<metadata_name_of_TempoMonolithic_CR>:4317** (OTLP/gRPC) および **tempo-<metadata_name_of_TempoMonolithic_CR>:4318** (OTLP/HTTP) エンドポイントにトレースを送信できます。
   Tempo API は、クラスター内の **tempo-<metadata_name_of_TempoMonolithic_CR>:3200** エンドポイントで利用できます。

3.1.3.2. CLI を使用した TempoMonolithic インスタンスのインストール

重要

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

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

コマンドラインから TempoMonolithic インスタンスをインストールできます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

ヒント

- OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
- **oc login** コマンドを実行します。

```
$ oc login --username=<your_username>
```

手順

1. 次のコマンドを実行して、後続のステップで作成する TempoMonolithic インスタンス用に任意のプロジェクトを作成します。

```
$ oc apply -f - << EOF
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: <project_of_tempomonolithic_instance>
EOF
```

2. トレースの保存に使用するサポート対象のストレージのタイプ (インメモリーストレージ、永続ボリューム、オブジェクトストレージ) を決定します。

重要

オブジェクトストレージは分散トレースプラットフォーム(Tempo)に含まれず、サポートされているプロバイダーによるオブジェクトストアを設定する必要があります ({odf-full}、MinIO、Amazon S3、Azure Blob Storage、または Google Cloud Storage)。

また、オブジェクトストレージを選択するには、TempoMonolithic インスタンス用に作成したプロジェクトにオブジェクトストレージバケットのシークレットを作成する必要があります。これを行うには、次のコマンドを実行します。

```
$ oc apply -f - << EOF
<object_storage_secret>
EOF
```

詳細は、「オブジェクトストレージのセットアップ」を参照してください。

Amazon S3 および MinIO ストレージのシークレット例

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. TempoMonolithic インスタンス用に作成したプロジェクト内に TempoMonolithic インスタンスを作成します。

ヒント

同じクラスター上の別々のプロジェクトに複数の TempoMonolithic インスタンスを作成できます。

a. **TempoMonolithic** カスタムリソース (CR) をカスタマイズします。
次の **TempoMonolithic** CR は、OTLP/gRPC および OTLP/HTTP 経由のトレース取り込みを備えた TempoMonolithic デプロイメントを作成し、サポートされているタイプのストレージにトレースを保存し、ルート経由で Jaeger UI を公開します。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <metadata_name>
  namespace: <project_of_tempomonolithic_instance>
spec:
  storage:
    traces:
      backend: <supported_storage_type> 1
      size: <value>Gi 2
      s3: 3
        secret: <secret_name> 4
  jaegerui:
    enabled: true 5
    route:
      enabled: true 6
```

1. トレースを保存するストレージのタイプ (インメモリーストレージ、永続ボリューム、またはオブジェクトストレージ)。**tmpfs** インメモリーストレージの値は、**memory** です。永続ボリュームの値は **pv** です。オブジェクトストレージの値は、使用するオブジェクトストアのタイプに応じて、**s3**、**gcs**、または **azure** が受け入れられます。

2. メモリーサイズ: インメモリーストレージの場合、これは **tmpfs** ボリュームのサイズを意味します。デフォルトは **2Gi** です。永続ボリュームの場合、これは永続ボリューム要求のサイズを意味します。デフォルトは **10Gi** です。オブジェクトストレージの場合、これは Tempo WAL の永続ボリューム要求のサイズを意味し、デフォルトは **10Gi** です。

3. オプション: オブジェクトストレージの場合、オブジェクトストレージのタイプ。使用するオブジェクトストアのタイプに応じて、**s3**、**gcs**、および **azure** が値として受け入れられます。

4. オプション: オブジェクトストレージの場合、ストレージシークレットの **metadata** 内の **name** の値。ストレージシークレットは、TempoMonolithic インスタンスと同じ namespace にあり、「表 1. 必要なシークレットパラメーター」(「オブジェクトストレージのセットアップ」セクションを参照) で指定えているフィールドを含んでいる必要があります。

5. Jaeger UI を有効にします。

6 Jaeger UI のルートの作成を有効にします。

b. 次のコマンドを実行して、カスタマイズされた CR を適用します。

```
$ oc apply -f - << EOF
<tempomonolithic_cr>
EOF
```

検証

1. 次のコマンドを実行して、すべての TempoMonolithic **components** の **status** が **Running** であり、**conditions** が **type: Ready** であることを確認します。

```
$ oc get tempomonolithic.tempo.grafana.com <metadata_name_of_tempomonolithic_cr> -o yaml
```

2. 次のコマンドを実行して、TempoMonolithic インスタンスの Pod が実行中であることを確認します。

```
$ oc get pods
```

3. Jaeger UI にアクセスします。

a. 次のコマンドを実行して、**tempo-<metadata_name_of_tempomonolithic_cr>-jaegerui** ルートのルート詳細をクエリーします。

```
$ oc get route
```

b. Web ブラウザーで **https://<route_from_previous_step>** を開きます。

4. TempoMonolithic インスタンスの Pod の準備ができたら、クラスター内の **tempo-<metadata_name_of_tempomonolithic_cr>:4317** (OTLP/gRPC) および **tempo-<metadata_name_of_tempomonolithic_cr>:4318** (OTLP/HTTP) エンドポイントにトレースを送信できます。
Tempo API は、クラスター内の **tempo-<metadata_name_of_tempomonolithic_cr>:3200** エンドポイントで利用できます。

3.1.4. オブジェクトストレージのセットアップ

サポートされているオブジェクトストレージを設定する際に、次の設定パラメーターを使用できます。

表3.1 必須のシークレットパラメーター

ストレージプロバイダー
Secret パラメーター
{odf-full}

ストレージプロバイダー
name: tempostack-dev-odf # example **bucket: <bucket_name> # requires an ObjectBucketClaim** **endpoint: https://s3.openshift-storage.svc** **access_key_id: <data_foundation_access_key_id>** **access_key_secret: <data_foundation_access_key_secret>**
MinIO
MinIO Operator を参照してください。 **name: tempostack-dev-minio # example** **bucket: <minio_bucket_name> # MinIO documentation** **endpoint: <minio_bucket_endpoint>** **access_key_id: <minio_access_key_id>** **access_key_secret: <minio_access_key_secret>**
Amazon S3
name: tempostack-dev-s3 # example **bucket: <s3_bucket_name> # Amazon S3 documentation** **endpoint: <s3_bucket_endpoint>** **access_key_id: <s3_access_key_id>** **access_key_secret: <s3_access_key_secret>**
Microsoft Azure Blob Storage
name: tempostack-dev-azure # example **container: <azure_blob_storage_container_name> # Microsoft Azure documentation** **account_name: <azure_blob_storage_account_name>** **account_key: <azure_blob_storage_account_key>**
Google Cloud Storage on Google Cloud Platform (GCP)

ストレージプロバイダー
name: tempostack-dev-gcs # example **bucketname: <google_cloud_storage_bucket_name> # requires a bucket created in a GCP project** **key.json: <path/to/key.json> # requires a service account in the bucket's GCP project for GCP authentication**

3.1.5. 関連情報

- クラスター管理者の作成
- OperatorHub.io
- Web コンソールへのアクセス
- Web コンソールを使用した OperatorHub からのインストール
- インストールされた Operator からのアプリケーションの作成
- OpenShift CLI の使用を開始

3.2. 設定

Tempo Operator は、distributed tracing platform (Tempo) の作成とデプロイで使用する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) ファイルを使用します。デフォルト設定をインストールするか、ファイルを変更できます。

3.2.1. デプロイメントのカスタマイズ

バックエンドストレージの設定については、永続ストレージについて と、選択したストレージに対応する設定トピックを参照してください。

3.2.1.1. デフォルトの設定オプション

TempoStack カスタムリソース (CR) は、分散トレーシングプラットフォーム (Tempo) リソースを作成するときに使用するアーキテクチャーと設定を定義します。これらのパラメーターを変更して、distributed tracing platform (Tempo) の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。

汎用 Tempo YAML ファイルの例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: name
spec:
  storage: {}
  resources: {}
  storageSize: 200M
  replicationFactor: 1
  retention: {}
```

```
  template:
    distributor:{}
    ingester: {}
    compactor: {}
    querier: {}
    queryFrontend: {}
    gateway: {}
```

表3.2 Ttempo 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 API バージョン。	**tempo.grafana.com/v1alpha1**	**tempo.grafana.com/v1alpha1**
kind:	作成する Kubernetes オブジェクトの種類を定義します。	**tempo**	
metadata:	**name** の文字列、**UID**、オプションの **namespace** などのオブジェクトを一意に識別するデータ。		OpenShift Container Platform 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** を完了します。
name:	オブジェクトの名前。	TempoStack インスタンスの名前。	**tempo-all-in-one-inmemory**
spec:	作成するオブジェクトの仕様。	TempoStack インスタンスのすべての設定パラメーターが含まれています。すべての Tempo コンポーネントの共通定義が必要な場合、**spec** ノードで定義されます。個々のコンポーネントに関連する定義は、**spec/template/<component>** ノードに置かれます。	該当なし
resources:	TempoStack インスタンスに割り当てられたリソース。		
storageSize:	Ingester PVC のストレージサイズ。		

パラメーター	説明	値	デフォルト値
replicationFactor:	レプリケーション係数の設定。		
retention:	トレースの保持に関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オプションは、**allInOne** や他のコンポーネントオプションではなく、**storage** の下に配置される必要があります。		
template.distributor:	Tempo **distributor** の設定オプション。		
template.ingester:	Tempo **ingester** の設定オプション。		
template.compactor:	Tempo **compactor** の設定オプション。		
template.querier:	Tempo **querier** の設定オプション。		
template.queryFrontend:	Tempo **query-frontend** の設定オプション。		
template.gateway:	Tempo **gateway** の設定オプション。		

最低限必要な設定

以下は、デフォルト設定で distributed tracing platform (Tempo) デプロイメントを作成するために必要な最小限の設定です。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
spec:
  storage: 1
    secret:
      name: minio
      type: s3
  resources:
```

```
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        ingress:
          type: route
```

1 このセクションでは、デプロイされたオブジェクトストレージバックエンドを指定します。そのためには、オブジェクトストレージにアクセスするための作成済みシークレットと認証情報が必要です。

3.2.1.2. ストレージの設定

spec.storage の下にある **TempoStack** カスタムリソースで、distributed tracing platform (Tempo) のオブジェクトストレージを設定できます。サポート対象である複数のストレージプロバイダーから選択できます。

表3.3 Tempo Operator が分散トレーシングストレージの定義に使用する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.secret.type	デプロイメントに使用するストレージのタイプ。	**memory**。Pod がシャットダウンされるとデータは保持されないため、memory ストレージは開発、テスト、デモンストレーション、概念実証環境にのみ適しています。	**memory**
storage.secretname	設定されたオブジェクトストレージタイプの認証情報を含むシークレットの名前。		該当なし
storage.tls.caName	CA は、CA 証明書が含まれる **ConfigMap** オブジェクトの名前です。		

表3.4 必須のシークレットパラメーター

ストレージプロバイダー
Secret パラメーター
{odf-full}

ストレージプロバイダー
name: tempostack-dev-odf # example **bucket: <bucket_name> # requires an ObjectBucketClaim** **endpoint: https://s3.openshift-storage.svc** **access_key_id: <data_foundation_access_key_id>** **access_key_secret: <data_foundation_access_key_secret>**
MinIO
MinIO Operator を参照してください。 **name: tempostack-dev-minio # example** **bucket: <minio_bucket_name> # MinIO documentation** **endpoint: <minio_bucket_endpoint>** **access_key_id: <minio_access_key_id>** **access_key_secret: <minio_access_key_secret>**
Amazon S3
name: tempostack-dev-s3 # example **bucket: <s3_bucket_name> # Amazon S3 documentation** **endpoint: <s3_bucket_endpoint>** **access_key_id: <s3_access_key_id>** **access_key_secret: <s3_access_key_secret>**
Microsoft Azure Blob Storage
name: tempostack-dev-azure # example **container: <azure_blob_storage_container_name> # Microsoft Azure documentation** **account_name: <azure_blob_storage_account_name>** **account_key: <azure_blob_storage_account_key>**
Google Cloud Storage on Google Cloud Platform (GCP)

ストレージプロバイダー
name: tempostack-dev-gcs # example **bucketname: <google_cloud_storage_bucket_name> # requires a bucket created in a GCP project** **key.json: <path/to/key.json> # requires a service account in the bucket's GCP project for GCP authentication**

3.2.1.3. クエリー設定オプション

分散トレースプラットフォーム (Tempo) の 2 つのコンポーネント、クエリアーとクエリーフロントエンドがクエリーを管理します。これらのコンポーネントは両方とも設定できます。

クエリアーコンポーネントは、インジェスターまたはバックエンドストレージで要求されたトレース ID を検索します。設定されたパラメーターに応じて、クエリアーコンポーネントはインジェスターの両方にクエリーを実行し、bloom またはインデックスをバックエンドからプルして、オブジェクトストレージ内のブロックを検索できます。クエリアーコンポーネントは **GET /querier/api/traces/<trace_id>** で HTTP エンドポイントを公開します。ただし、このエンドポイントを直接使用することは想定されていません。クエリーはクエリーフロントエンドに送信する必要があります。

表3.5 クエリアーコンポーネントの設定パラメーター

パラメーター	説明	値
nodeSelector	ノード選択制約の単純な形式。	type: object
replicas	コンポーネントに対して作成されるレプリカの数。	type: integer; format: int32
tolerations	コンポーネント固有の Pod 容認。	type: array

クエリーフロントエンドコンポーネントは、受信クエリーの検索スペースをシャーディングする役割を持ちます。クエリーフロントエンドは、単純な HTTP エンドポイント (**GET /api/traces/<trace_id>**) を介してトレースを公開します。内部的には、クエリーフロントエンドコンポーネントは **blockID** スペースを設定可能な数のシャードに分割し、これらのリクエストをキューに登録します。クエリアーコンポーネントは、ストリーミング gRPC 接続を介してクエリーフロントエンドコンポーネントに接続し、これらのシャードクエリーを処理します。

表3.6 クエリーフロントエンドコンポーネントの設定パラメーター

パラメーター	説明	値
component	クエリーフロントエンドコンポーネントの設定。	type: object
component.nodeSelector	ノード選択制約の単純な形式。	type: object

パラメーター	説明	値
component.replicas	クエリーフロントエンドコンポーネントに対して作成されるレプリカの数。	type: integer; format: int32
component.tolerations	クエリーフロントエンドコンポーネントに固有の Pod 容認。	type: array
jaegerQuery	Jaeger Query コンポーネントに固有のオプション。	type: object
jaegerQuery.enabled	**enabled** にすると、Jaeger Query コンポーネント **jaegerQuery** が作成されます。	type: boolean
jaegerQuery.ingress	Jaeger Query Ingress のオプション。	type: object
jaegerQuery.ingress.annotations	Ingress オブジェクトのアノテーション。	type: object
jaegerQuery.ingress.host	Ingress オブジェクトのホスト名。	type: string
jaegerQuery.ingress.ingressClassName	IngressClass クラスターリソースの名前。この Ingress リソースを提供する Ingress コントローラーを定義します。	type: string
jaegerQuery.ingress.route	OpenShift ルートのオプション。	type: object
jaegerQuery.ingress.route.termination	終端タイプ。デフォルトは **edge** です。	type: string (enum: insecure, edge, passthrough, reencrypt)
jaegerQuery.ingress.type	Jaeger Query UI の Ingress のタイプ。サポートされているタイプは、**ingress**、**route**、および **none** です。	type: string (enum: ingress, route)
jaegerQuery.monitorTab	Monitor タブの設定。	type: object
jaegerQuery.monitorTab.enabled	Jaeger コンソールの Monitor タブを有効にします。**PrometheusEndpoint** を設定する必要があります。	type: boolean

パラメーター	説明	値
jaegerQuery.monitorTab.prometheusEndpoint	スパンのレート、エラー、および期間 (RED) メトリクスを含む Prometheus インスタンスへのエンドポイント。たとえば、**https://thanos-querier.openshift-monitoring.svc.cluster.local:9091** です。	type: string

TempoStack CR のクエリーフロントエンドコンポーネントの設定例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
spec:
  storage:
    secret:
      name: minio
      type: s3
  storageSize: 200M
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

3.2.1.3.1. 関連情報

- テイントおよび容認 (Toleration) について

3.2.1.4. Jaeger UI の Monitor タブの設定

トレースデータには豊富な情報が含まれており、データはインストルメント化された言語およびフレームワーク間で正規化されます。したがって、リクエストのレート、エラー、および期間 (RED) メトリクスをトレースから抽出できます。メトリクスは、Jaeger コンソールの **Monitor** タブで可視化できます。

メトリクスは、ユーザーワークロード監視スタックにデプロイされた Prometheus により Collector から収集された OpenTelemetry コレクターのスパンから導出されます。Jaeger UI は、Prometheus エンドポイントからこれらのメトリクスをクエリーし、可視化します。

3.2.1.4.1. OpenTelemetry Collector の設定

OpenTelemetry Collector では、トレースからメトリクスを導出し、そのメトリクスを Prometheus 形式でエクスポートする **spanmetrics** コネクターの設定が必要です。

スパン RED の OpenTelemetry Collector カスタムリソース

```
kind: OpenTelemetryCollector
apiVersion: opentelemetry.io/v1alpha1
metadata:
  name: otel
spec:
  mode: deployment
  observability:
    metrics:
      enableMetrics: true (1)
  config: |
    connectors:
      spanmetrics: (2)
        metrics_flush_interval: 15s

    receivers:
      otlp: (3)
        protocols:
          grpc:
          http:

    exporters:
      prometheus: (4)
        endpoint: 0.0.0.0:8889
        add_metric_suffixes: false
        resource_to_telemetry_conversion:
          enabled: true # by default resource attributes are dropped

      otlp:
        endpoint: "tempo-simplest-distributor:4317"
        tls:
          insecure: true

    service:
      pipelines:
        traces:
          receivers: [otlp]
          exporters: [otlp, spanmetrics] (5)
        metrics:
          receivers: [spanmetrics] (6)
          exporters: [prometheus]
```

(1) **ServiceMonitor** カスタムリソースを作成して、Prometheus エクスポーターの収集を有効にします。

(2) Spanmetrics コネクターはトレースを受信し、メトリクスをエクスポートします。

(3) OpenTelemetry プロトコルのスパンを受信する OTLP レシーバー。

(4) Prometheus エクスポーターは、Prometheus 形式でメトリクスをエクスポートするために使用されます。

5 Spanmetrics コネクターは、トレースパイプラインのエクスポーターとして設定されています。

6 Spanmetrics コネクターは、メトリクスパイプラインのレシーバーとして設定されています。

3.2.1.4.2. Tempo の設定

TempoStack カスタムリソースでは、**Monitor** タブが有効で、ユーザー定義の監視スタックからデータをクエリーするように Prometheus エンドポイントを Thanos Querier サービスに設定する必要があります。

Monitor タブが有効な TempoStack カスタムリソース

```
kind: TempoStack
apiVersion: tempo.grafana.com/v1alpha1
metadata:
  name: simplest
spec:
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        monitorTab:
          enabled: true 1
          prometheusEndpoint: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 2
        ingress:
          type: route
```

1 Jaeger コンソールの監視タブを有効にします。

2 ユーザーワークロード監視からの Thanos Querier のサービス名。

3.2.1.4.3. Span RED メトリクスとアラートルール

spanmetrics コネクターによって生成されるメトリクスは、アラートルールで使用できます。たとえば、このコネクターは、サービスの速度低下に関するアラートの場合や、サービスレベル目標 (SLO) を定義する場合のために、**duration_bucket** ヒストグラムと **calls** カウンターメトリクスを作成します。これらのメトリクスには、サービス、API 名、操作タイプ、その他の属性を識別するラベルが付いています。

表3.7 spanmetrics コネクターで作成されるメトリクスのラベル

ラベル	説明	値
service_name	**otel_service_name** 環境変数によって設定されるサービス名。	**frontend**
span_name	操作の名前。	• / • **/customer**

ラベル	説明	値
span_kind	サーバー、クライアント、メッセージング、または内部操作を識別します。	• **SPAN_KIND_SERVER** • **SPAN_KIND_CLIENT** • **SPAN_KIND_PRODUCER** • **SPAN_KIND_CONSUMER** • **SPAN_KIND_INTERNAL**

フロントエンドサービスで 2000 ミリ秒以内に 95% の要求が処理されない場合の SLO のアラートルールを定義する PrometheusRule CR の例

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: span-red
spec:
  groups:
  - name: server-side-latency
    rules:
    - alert: SpanREDFrontendAPIRequestLatency
      expr: histogram_quantile(0.95, sum(rate(duration_bucket{service_name="frontend",
span_kind="SPAN_KIND_SERVER"}[5m])) by (le, service_name, span_name)) > 2000 (1)
      labels:
        severity: Warning
      annotations:
        summary: "High request latency on {{$labels.service_name}} and {{$labels.span_name}}"
        description: "{{$labels.instance}} has 95th request latency above 2s (current value: {{$value}}s)"
```

(1) 95% のフロントエンドサーバーの応答時間値が 2000 ミリ秒未満であるかどうかを確認する式。時間範囲 (**[5m]**) が収集間隔の 4 倍以上で、メトリクスの変化に対応できる十分な長さである必要があります。

3.2.1.5. マルチテナントへの対応

認証と認可を備えたマルチテナンシーは、Tempo Gateway サービスで提供されます。認証には、OpenShift OAuth と Kubernetes **TokenReview** API を使用します。認可には、Kubernetes **SubjectAccessReview** API を使用します。

注記

Tempo Gateway サービスは、OTLP/gRPC 経由のトレース取り込みのみをサポートしています。OTLP/HTTP はサポートされていません。

2 つのテナント (dev と prod) を使用したサンプル Tempo CR

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoStack
metadata:
  name: simplest
spec:
  tenants:
    mode: openshift (1)
    authentication: (2)
      - tenantName: dev (3)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (4)
      - tenantName: prod
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
  template:
    gateway:
      enabled: true (5)
    queryFrontend:
      jaegerQuery:
        enabled: true
```

1. **openshift** に設定する必要があります。
2. テナントのリスト。
3. テナント名。データを取り込む際に、**X-Scope-OrgId** ヘッダーで指定する必要があります。
4. 一意のテナント ID。
5. 認証と認可を実行するゲートウェイを有効にします。Jaeger UI は、**http://<gateway-ingress>/api/traces/v1/<tenant-name>/search** で公開されています。

認可設定では、Kubernetes ロールベースアクセス制御 (RBAC) の **ClusterRole** と **ClusterRoleBinding** を使用します。デフォルトでは、読み取りまたは書き込み権限を持っているユーザーはいません。

認証されたユーザーが dev テナントおよび prod テナントのトレースデータを読み取ることができる読み取り RBAC 設定のサンプル

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tempostack-traces-reader
rules:
  - apiGroups:
      - 'tempo.grafana.com'
    resources: (1)
      - dev
      - prod
    resourceNames:
      - traces
    verbs:
      - 'get' (2)
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
```

```
  name: tempostack-traces-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tempostack-traces-reader
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated (3)
```

(1) テナントをリスト表示します。

(2) 値が **get** の場合、読み取り操作が有効になります。

(3) 認証されたユーザー全員に、トレースデータの読み取り権限を付与します。

otel-collector サービスアカウントが dev テナントのトレースデータを書き込むことができる書き込み RBAC 設定のサンプル

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: otel-collector (1)
  namespace: otel
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tempostack-traces-write
rules:
  - apiGroups:
      - 'tempo.grafana.com'
    resources: (2)
      - dev
    resourceNames:
      - traces
    verbs:
      - 'create' (3)
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tempostack-traces
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tempostack-traces-write
subjects:
  - kind: ServiceAccount
    name: otel-collector
    namespace: otel
```

1. トレースデータのエクスポートでクライアントが使用するサービスアカウント名。クライアントは、サービスアカウントトークン **/var/run/secrets/kubernetes.io/serviceaccount/token** をベアラートークンヘッダーとして送信する必要があります。
2. テナントをリスト表示します。
3. 値が **create** の場合、書き込み操作が有効になります。

トレースデータは、データ書き込み用の RBAC を持つサービスアカウントを使用する OpenTelemetry Collector から Tempo インスタンスに送信できます。

OpenTelemetry CR 設定のサンプル

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: tracing-system
spec:
  mode: deployment
  serviceAccount: otel-collector
  config: |
    extensions:
      bearertokenauth:
        filename: "/var/run/secrets/kubernetes.io/serviceaccount/token"
    exporters:
      otlp/dev:
        endpoint: tempo-simplest-gateway.tempo.svc.cluster.local:8090
        tls:
          insecure: false
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
        auth:
          authenticator: bearertokenauth
        headers:
          X-Scope-OrgID: "dev"
    service:
      extensions: [bearertokenauth]
      pipelines:
        traces:
          exporters: [otlp/dev]
```

3.2.2. 監視とアラートの設定

Tempo Operator は、distributor や ingester などの各 TempoStack コンポーネントのモニタリングとアラートをサポートし、Operator 自体に関するアップグレードおよび運用のメトリクスを公開します。

3.2.2.1. TempoStack のメトリクスとアラートの設定

TempoStack インスタンスのメトリクスとアラートを有効にできます。

前提条件

- ユーザー定義プロジェクトのモニタリングがクラスターで有効にされている。

手順

1. TempoStack インスタンスのメトリクスを有効にするには、**spec.observability.metrics.createServiceMonitors** フィールドを **true** に設定します。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
spec:
  observability:
    metrics:
      createServiceMonitors: true
```

2. TempoStack インスタンスのアラートを有効にするには、**spec.observability.metrics.createPrometheusRules** フィールドを **true** に設定します。

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
spec:
  observability:
    metrics:
      createPrometheusRules: true
```

検証

Web コンソールの **Administrator** ビューを使用して、正常に設定されたことを確認できます。

1. **Observe → Targets** に移動して **Source: User** でフィルタリングし、**tempo-<instance_name>-<component>** 形式の **ServiceMonitor** のステータスが **Up** であることを確認します。

2. アラートが正しく設定されていることを確認するには、**Observe → Alerting → Alerting rules** に移動して **Source: User** でフィルタリングし、TempoStack インスタンスコンポーネントの **Alert rules** が利用可能であることを確認します。

3.2.2.1.1. 関連情報

- ユーザー定義プロジェクトのモニタリングの有効化

3.2.2.2. Tempo Operator のメトリクスとアラートの設定

Web コンソールから Tempo Operator をインストールする場合は、**Enable Operator recommended cluster monitoring on this Namespace** チェックボックスを選択すると、Tempo Operator のメトリクスおよびアラートを作成できます。

インストール時にチェックボックスを選択しなかった場合も、Tempo Operator をインストールした後にメトリクスとアラートを手動で有効にできます。

手順

- Tempo Operator がインストールされているプロジェクトに **openshift.io/cluster-monitoring: "true"** ラベルを追加します。デフォルトは **openshift-tempo-operator** です。

検証

Web コンソールの **Administrator** ビューを使用して、正常に設定されたことを確認できます。

1. **Observe → Targets** に移動して **Source: Platform** でフィルタリングし、**tempo-operator** を検索します。その場合は、ステータスは **Up** でなければなりません。
2. アラートが正しく設定されていることを確認するには、**Observe → Alerting → Alerting rules** に移動して **Source: Platform** でフィルタリングし、**Tempo Operator** の **Alert rules** を見つけます。

3.3. アップグレード

バージョンアップグレードの場合、Tempo Operator は Operator Lifecycle Manager (OLM) を使用します。これは、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。

Tempo Operator が新しいバージョンにアップグレードされると、その Operator が管理する実行中の TempoStack インスタンスをスキャンし、新しい Operator バージョンに対応するバージョンにアップグレードします。

3.3.1. 関連情報

- Operator Lifecycle Manager の概念およびリソース
- インストール済み Operator の更新

3.4. 削除中

OpenShift Container Platform クラスターから Red Hat OpenShift distributed tracing platform (Tempo) を削除する手順を、以下に示します。

1. すべての distributed tracing platform (Tempo) Pod をシャットダウンします。
2. TempoStack インスタンスを削除します。
3. Tempo Operator を削除します。

3.4.1. Web コンソールを使用して削除する

Web コンソールの **Administrator** ビューで、TempoStack インスタンスを削除できます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として、OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Dedicated の場合、**dedicated-admin** ロールを持つアカウントを使用してログインしている。

手順

1. **Operators → Installed Operators → Tempo Operator → TempoStack** に移動します。
2. TempoStack インスタンスを削除するには、 → **Delete TempoStack → Delete** を選択します。
3. オプション: Tempo Operator を削除します。

3.4.2. CLI を使用して削除する

コマンドラインで TempoStack インスタンスを削除できます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

  ヒント

  - OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
  - **oc login** を実行します。

    ```
    $ oc login --username=<your_username>
    ```

手順

1. 以下のコマンドを実行して、TempoStack インスタンスの名前を取得します。

   ```
   $ oc get deployments -n <project_of_tempostack_instance>
   ```

2. 以下のコマンドを実行して、TempoStack インスタンスを削除します。

   ```
   $ oc delete tempo <tempostack_instance_name> -n <project_of_tempostack_instance>
   ```

3. オプション: Tempo Operator を削除します。

検証

1. 以下のコマンドを実行して、出力に TempoStack インスタンスがないことを確認します。ない場合、正常に削除されています。

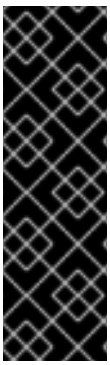
   ```
   $ oc get deployments -n <project_of_tempostack_instance>
   ```

3.4.3. 関連情報

- クラスターからの Operator の削除
- OpenShift CLI の使用を開始

第4章 DISTRIBUTED TRACING PLATFORM (JAEGER)

4.1. インストール



重要

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

Red Hat OpenShift distributed tracing platform 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。

- Red Hat OpenShift distributed tracing platform は、Red Hat OpenShift Service Mesh の一部としてインストールできます。分散トレースは、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 Red Hat OpenShift distributed tracing platform 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。Red Hat OpenShift distributed tracing platform はサービスメッシュと同じ namespace にインストールする必要があります。つまり、**ServiceMeshControlPlane** と Red Hat OpenShift distributed tracing platform リソースが同じ namespace にある必要があります。
- 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、Red Hat OpenShift distributed tracing platform Operator を使用して distributed tracing platform をインストールできます。サービスメッシュなしで Red Hat OpenShift distributed tracing platform をインストールするには、以下の手順を実行します。

4.1.1. 前提条件

Red Hat OpenShift distributed tracing platform をインストールする前に、インストールアクティビティーで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 4.16 の概要 を確認します。
- OpenShift Container Platform 4.12 をインストールします。
  - AWS への OpenShift Container Platform 4.12 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 4.12 のインストール
  - ベアメタルへの OpenShift Container Platform 4.12 のインストール
  - vSphere への OpenShift Container Platform 4.12 のインストール
- OpenShift Container Platform バージョンに一致する **oc** CLI ツールのバージョンをインストールし、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ある。

4.1.2. Red Hat OpenShift distributed tracing platform のインストール概要

Red Hat OpenShift distributed tracing platform は、次の手順でインストールできます。

- 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 OpenShift Elasticsearch Operator をインストールします。
- OperatorHub を使用して Red Hat OpenShift distributed tracing platform (Jaeger) Operator をインストールします。
- デプロイメントストラテジーをサポートするよう、カスタムリソース YAML ファイルを変更します。
- Red Hat OpenShift distributed tracing platform (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。

4.1.3. OpenShift Elasticsearch Operator のインストール

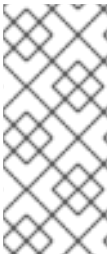
デフォルトの Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) のデプロイメントでは、インメモリーのストレージが使用されます。これは、Red Hat OpenShift 分散トレースの評価、デモの提供、またはテスト環境での Red Hat OpenShift 分散トレースプラットフォームの使用を希望するユーザー用に、迅速にインストール行うために設計されているためです。実稼働環境で Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) を使用する予定がある場合、永続ストレージのオプション (この場合は Elasticsearch) をインストールし、設定する必要があります。

前提条件

- OpenShift Container Platform Web コンソールにアクセスできる。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

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。

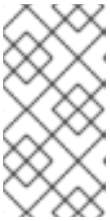


注記

OpenShift Logging の一部として OpenShift Elasticsearch Operator がすでにインストールされている場合は、OpenShift Elasticsearch Operator を再びインストールする必要はありません。Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator はインストールされた OpenShift 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。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
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、OpenShift Elasticsearch Operator を検索します。
4. Red Hat が提供する **OpenShift Elasticsearch Operator** をクリックし、Operator に関する情報を表示します。
5. **Install** をクリックします。
6. **Install Operator** ページで、**stable** 更新チャネルを選択します。これにより、新しいバージョンがリリースされると Operator が自動的に更新されます。
7. デフォルトの **All namespaces on the cluster (default)**を受け入れます。これにより、Operator がデフォルトの **openshift-operators-redhat** 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。



注記

Elasticsearch のインストールには、OpenShift Elasticsearch Operator 用の **openshift-operators-redhat** namespace が必要です。他の Red Hat OpenShift distributed tracing platform Operators は、**openshift-operators** namespace にインストールされます。

8. デフォルトの **Automatic** 承認ストラテジーを受け入れます。デフォルトを受け入れることで、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は、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。

9. **Install** 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。OpenShift Elasticsearch Operator が **InstallSucceeded** ステータスになるまで待機してから続行します。

4.1.4. Red Hat OpenShift 分散トレースプラットフォーム Operator のインストール

OperatorHub を使用して Red Hat OpenShift 分散トレーシング Platform Operator をインストールできます。

デフォルトでは、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 コンソールにアクセスできる。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
- 永続ストレージが必要な場合は、Red Hat OpenShift 分散トレーシング Platform Operator をインストールする前に OpenShift Elasticsearch Operator をインストールする必要があります。

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。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. **Operators → OperatorHub** に移動します。
3. 検索フィールドに **distributed tracing platform** と入力して、Red Hat OpenShift 分散トレーシング Platform Operator を検索します。
4. **Red Hat が提供** する **Red Hat OpenShift distributed tracing platform** Operator を選択し、Operator に関する情報を表示します。
5. **Install** をクリックします。
6. **Install Operator** ページの **Update channel** で **stable** を選択すると、新しいバージョンがリリースされたときに Operator が自動的に更新されます。
7. デフォルトの **All namespaces on the cluster (default)**を受け入れます。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。
8. デフォルトの **Automatic** 承認ストラテジーを受け入れます。

注記

このデフォルトを受け入れると、Operator の新しいバージョンが利用可能になったときに、Operator Lifecycle Manager (OLM) が、この 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。

Manual 更新を選択した場合、新しいバージョンの Operator が利用可能になると、OLM は更新リクエストを作成します。Operator を新しいバージョンに更新するには、クラスター管理者として更新リクエストを手動で承認する必要があります。**Manual** 承認ストラテジーでは、クラスター管理者が Operator のインストールとサブスクリプションを手動で承認する必要があります。

9. **Install** をクリックします。
10. **Operators → Installed Operators** に移動します。
11. **Installed Operators** ページで、**openshift-operators** プロジェクトを選択します。Red Hat OpenShift 分散トレーシング Platform Operator のステータスが **Succeeded** になるまで待機してから続行します。

4.2. 設定

重要

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

Red Hat OpenShift distributed tracing platform (Jaeger) Operator は、distributed tracing platform (Jaeger) リソースを作成してデプロイする際に使用される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) ファイルを使用します。デフォルト設定をインストールするか、ファイルを変更できます。

Red Hat OpenShift Service Mesh の一部として distributed tracing platform をインストールしている場合、ServiceMeshControlPlane の一部として基本的な設定を行なえますが、完全に制御するためには、Jaeger CR を設定してから ServiceMeshControlPlane の分散トレーシング機能設定ファイルを参照する必要があります。

Red Hat OpenShift distributed tracing platform (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。distributed tracing platform (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: MyConfigFile
spec:
  strategy: production (1)
```

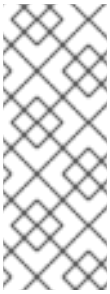
(1) デプロイメントストラテジー

4.2.1. サポート対象のデプロイメントストラテジー

Red Hat OpenShift 分散トレースプラットフォーム (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

- 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、デフォルトでインメモリーストレージを使用するように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

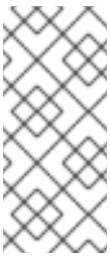
インメモリーストレージには永続性がありません。つまり、distributed tracing platform (Jaeger) インスタンスがシャットダウンまたは再起動するか、置き換えられると、トレースデータが失われます。各 Pod には独自のメモリーがあるため、インメモリーストレージはスケーリングできません。永続ストレージの場合は、デフォルトのストレージとして Elasticsearch を使用する **production** または **streaming** ストラテジーを使用する必要があります。

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 Elasticsearch バックエンドストレージ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

- streaming 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。
- IBM Z では、現在ストリーミングデプロイメントストラテジーはサポートされていません。

4.2.2. Web コンソールから distributed tracing platform のデフォルトストラテジーをデプロイする

カスタムリソース定義 (CRD) は、Red Hat OpenShift distributed tracing platform 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。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 Red Hat OpenShift distributed tracing platform (Jaeger) インスタンスを作成するか、独自のカスタムリソースファイルを定義できます。

注記

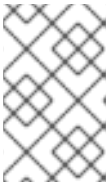
インメモリーストレージには永続性がありません。Jaeger Pod がシャットダウンするか、再起動するか、置き換えられると、トレースデータが失われます。永続ストレージの場合は、デフォルトのストレージとして Elasticsearch を使用する **production** または **streaming** ストラテジーを使用する必要があります。

前提条件

- Red Hat OpenShift distributed tracing platform (Jaeger) Operator がインストールされている。
- デプロイメントのカスタマイズ手順を確認している。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

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: **tracing-system**) を作成します。



注記

サービスメッシュの一部としてインストールする場合、distributed tracing platform リソースは、**istio-system** など、**ServiceMeshControlPlane** リソースと同じ namespace にインストールする必要があります。

   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 **tracing-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 **tracing-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が必要な場合があります。
5. Red Hat OpenShift distributed tracing platform (Jaeger) Operator をクリックします。**Details** 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 distributed tracing platform (Jaeger) インスタンスを作成します。
8. **Jaegers** ページで、distributed tracing platform (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。

4.2.2.1. CLI から distributed tracing platform のデフォルトストラテジーをデプロイする

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

前提条件

- Red Hat OpenShift distributed tracing platform (Jaeger) Operator がインストールされ、検証されている。

- デプロイメントのカスタマイズ手順を確認している。
- OpenShift Container Platform バージョンに一致する OpenShift CLI (**oc**) にアクセスできる。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 以下のコマンドを実行して、**cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインしてください。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:8443
```

2. 以下のコマンドを実行して、**tracing-system** という名前の新規プロジェクトを作成します。

```
$ oc new-project tracing-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、distributed tracing platform (Jaeger) をデプロイします。

```
$ oc create -n tracing-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 tracing-system -w
```

インストールプロセスが完了すると、出力は次の例のようになります。

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

4.2.3. Web コンソールから distributed tracing platform の production ストラテジーをデプロイする

production デプロイメントストラテジーは、よりスケーラブルで高可用性のあるアーキテクチャーを必要とし、トレースデータの長期保存が重要となる実稼働環境向けのものです。

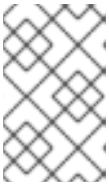
前提条件

- OpenShift Elasticsearch Operator がインストールされている。
- Red Hat OpenShift distributed tracing platform (Jaeger) Operator がインストールされている。

- デプロイメントのカスタマイズ手順を確認している。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

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: **tracing-system**) を作成します。



注記

サービスメッシュの一部としてインストールする場合、distributed tracing platform リソースは、**istio-system** など、**ServiceMeshControlPlane** リソースと同じ namespace にインストールする必要があります。

   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 **tracing-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 **tracing-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が必要な場合があります。
5. Red Hat OpenShift distributed tracing platform (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 distributed tracing platform (Jaeger) インスタンスを作成します。
9. **Jaegers** ページで、distributed tracing platform (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。

4.2.3.1. CLI から distributed tracing platform の production ストラテジーをデプロイする

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

前提条件

- OpenShift Elasticsearch Operator がインストールされている。
- Red Hat OpenShift distributed tracing platform (Jaeger) Operator がインストールされている。
- デプロイメントのカスタマイズ手順を確認している。
- OpenShift Container Platform バージョンに一致する OpenShift CLI (**oc**) にアクセスできる。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 以下のコマンドを実行して、**cluster-admin** ロールが割り当てられたユーザーとして OpenShift CLI (**oc**) にログインします。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:8443
```

2. 以下のコマンドを実行して、**tracing-system** という名前の新規プロジェクトを作成します。

```
$ oc new-project tracing-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、distributed tracing platform (Jaeger) をデプロイします。

```
$ oc create -n tracing-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 tracing-system -w
```

インストールプロセスが完了すると、以下の例ような出力が表示されます。

```
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
```

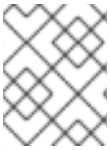
4.2.4. Web コンソールから distributed tracing platform の streaming ストラテジーをデプロイする

streaming デプロイメントストラテジーは、よりスケーラブルで高可用性のあるアーキテクチャーを必要とし、トレースデータの長期保存が重要となる実稼働環境向けのものです。

streaming ストラテジーは、Collector と Elasticsearch ストレージ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 Kafka ストリーミングプラットフォーム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

streaming 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。



注記

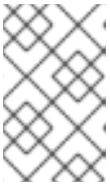
IBM Z では、現在ストリーミングデプロイメントストラテジーはサポートされていません。

前提条件

- AMQ Streams Operator がインストールされている。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。
- Red Hat OpenShift distributed tracing platform (Jaeger) Operator がインストールされている。
- デプロイメントのカスタマイズ手順を確認している。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

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: **tracing-system**) を作成します。



注記

サービスメッシュの一部としてインストールする場合、distributed tracing platform リソースは、**istio-system** など、**ServiceMeshControlPlane** リソースと同じ namespace にインストールする必要があります。

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 **tracing-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 **tracing-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が必要な場合があります。

5. Red Hat OpenShift distributed tracing platform (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
          brokers: my-cluster-kafka-brokers.kafka:9092 (1)
  storage:
    type: elasticsearch
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
```

(1) ブローカーが定義されていない場合、AMQStreams 1.4.0 以降は Kafka をセルフプロビジョニングします。

8. **Create** をクリックして distributed tracing platform (Jaeger) インスタンスを作成します。

9. **Jaegers** ページで、distributed tracing platform (Jaeger) インスタンスの名前 (例: **jaeger-streaming**) 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。

4.2.4.1. CLI から distributed tracing platform の streaming ストラテジーをデプロイする

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

前提条件

- AMQ Streams Operator がインストールされている。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。
- Red Hat OpenShift distributed tracing platform (Jaeger) Operator がインストールされている。
- デプロイメントのカスタマイズ手順を確認している。
- OpenShift Container Platform バージョンに一致する OpenShift CLI (**oc**) にアクセスできる。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 以下のコマンドを実行して、**cluster-admin** ロールが割り当てられたユーザーとして OpenShift CLI (**oc**) にログインします。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:8443
```

2. 以下のコマンドを実行して、**tracing-system** という名前の新規プロジェクトを作成します。

```
$ oc new-project tracing-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 tracing-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 tracing-system -w
```

インストールプロセスが完了すると、以下の例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
```

4.2.5. デプロイメントの検証

4.2.5.1. Jaeger コンソールへのアクセス

Jaeger コンソールにアクセスするには、Red Hat OpenShift Service Mesh または Red Hat OpenShift distributed tracing platform がインストールされ、Red Hat OpenShift distributed tracing platform (Jaeger) がインストール、設定、およびデプロイされている必要があります。

インストールプロセスにより、Jaeger コンソールにアクセスするためのルートが作成されます。

Jaeger コンソールの URL が分かっている場合は、これに直接アクセスできます。URL が分からない場合は、以下の指示を使用します。

Web コンソールからの手順

1. cluster-admin 権限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。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

2. **Networking → Routes** に移動します。

3. **Routes** ページで、**Namespace** メニューからコントロールプレーンプロジェクトを選択します (例:**tracing-system**)。
   Location 列には、各ルートのリンク先アドレスが表示されます。

4. 必要な場合は、フィルターを使用して **jaeger** ルートを検索します。ルートの **Location** をクリックしてコンソールを起動します。

5. **Log In With OpenShift** をクリックします。

CLI からの手順

1. 以下のコマンドを実行して、**cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインしてください。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

   ```
   $ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
   ```

2. コマンドラインを使用してルートの詳細をクエリーするには、以下のコマンドを入力します。この例では、**tracing-system** がコントロールプレーン namespace です。

   ```
   $ export JAEGER_URL=$(oc get route -n tracing-system jaeger -o jsonpath='{.spec.host}')
   ```

3. ブラウザーを起動し、**https://<JAEGER_URL>** に移動します。ここで、**<JAEGER_URL>** は直前の手順で検出されたルートです。

4. OpenShift Container Platform コンソールへアクセスするときに使用するものと同じユーザー名とパスワードを使用してログインします。

5. サービスメッシュにサービスを追加し、トレースを生成している場合は、フィルターと **Find Traces** ボタンを使用してトレースデータを検索します。
コンソールインストールを検証すると、表示するトレースデータはありません。

4.2.6. デプロイメントのカスタマイズ

4.2.6.1. デプロイメントのベストプラクティス

- Red Hat OpenShift 分散トレースプラットフォームインスタンスの名前は一意でなければなりません。複数の Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) インスタンスがあり、サイドカーが挿入されたエージェントを使用している場合、Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) インスタンスには一意の名前が必要となり、挿入 (injection) のアノテーションはトレースデータを報告する必要のある Red Hat OpenShift 分散トレースプラットフォームインスタンスの名前を明示的に指定する必要があります。

- マルチテナントの実装があり、テナントが namespace で分離されている場合は、Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) インスタンスを各テナント namespace にデプロイします。

永続ストレージの設定は、永続ストレージについて と、選択したストレージオプションに適した設定トピックを参照してください。

4.2.6.2. 分散トレースのデフォルト設定オプション

Jaeger カスタムリソース (CR) は、分散トレースプラットフォーム (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、分散トレースプラットフォーム (Jaeger) の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 CR の汎用 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4.1 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 API 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 Container Platform 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** を完了します。
name:	オブジェクトの名前。	分散トレースプラットフォーム (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。	分散トレースプラットフォーム (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。すべての Jaeger コンポーネントの共通定義が必要な場合、これは **spec** ノードで定義されます。定義が個々のコンポーネントに関連する場合は、**spec/<component>** ノードに置かれます。	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 Agent、Collector、Query、Ingester、および 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:	Agent 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 の例は、デフォルト設定を使用して Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) のデプロイメントを作成するために最低限必要なものです。

最小限必要な dist-tracing-all-in-one.yaml の例

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

4.2.6.3. 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 Elasticsearch ストレージに書き込み、**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。

Collector 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4.2 Operator によって使用される Jaeger Collector パラメーターを定義するためのパラメーター

パラメーター	説明	値
collector: replicas:	作成する Collector レプリカの数を指定します。	整数 (例: **5**)。

表4.3 Collector に渡される設定パラメーター

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。	
options: collector: num-workers:	キューからプルするワーカーの数。	整数 (例: **50**)。
options: collector: queue-size:	Collector キューのサイズ。	整数 (例: **2000**)。
options: kafka: producer: topic: jaeger-spans	**topic** パラメーターは、Collector 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。
options: 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、Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator は Kafka をセルフプロビジョニングします。	

パラメーター	説明	値
options: log-level:	Collector のロギングレベル。	使用できる値は、**debug**、**info**、**warn**、**error**、**fatal**、**panic** です。

options:
  otlp:
  enabled: true
  grpc:
  host-port: 4317
  max-connection-age: 0s
  max-connection-age-grace: 0s
  max-message-size: 4194304
  tls:
  enabled: false
  cert: /path/to/cert.crt
  cipher-suites: "TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256"
  client-ca: /path/to/cert.ca
  reload-interval: 0s
  min-version: 1.2
  max-version: 1.3
	OTLP/gRPC を受け入れるには、**otlp** を明示的に有効にします。他はすべて任意のオプションです。	

パラメーター	説明	値
options: otlp: enabled: true http: cors: allowed-headers: [<header-name>[, <header-name>]*] allowed-origins: * host-port: 4318 max-connection-age: 0s max-connection-age-grace: 0s max-message-size: 4194304 read-timeout: 0s read-header-timeout: 2s idle-timeout: 0s tls: enabled: false cert: /path/to/cert.crt cipher-suites: "TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256" client-ca: /path/to/cert.ca reload-interval: 0s min-version: 1.2 max-version: 1.3	OTLP/HTTP を受け入れるには、**otlp** を明示的に有効にします。他はすべて任意のオプションです。	

4.2.6.4. 分散トレースのサンプリング設定オプション

この Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) 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 Envoy プロキシーによって開始されている場合に関連がありません。Jaeger サンプリングの意思決定は、トレースがクライアント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、クライアント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。

分散トレーシングプラットフォーム (Jaeger) ライブラリーは、次のサンプラーをサポートしています。

- **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** を使用した場合は、約 10 のうち 1 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** を使用した場合は、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。

表4.4 Jaeger サンプリングのオプション

パラメーター	説明	値	デフォルト値
spec: sampling: options: {} default_strategy: service_strategy:	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。		設定を指定しない場合、Collector はすべてのサービスの確率 0.001 (0.1%) のデフォルトの確率的なサンプリングポリシーを返します。
default_strategy: type: service_strategy: 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 **probabilistic**、および **ratelimiting** です。	**probabilistic**
default_strategy: param: service_strategy: param:	選択したサンプリングストラテジーのパラメーター	10 進値および整数値 (0、.1、1、10)	1

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:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 0.5
```

```
    service_strategies:
      - service: alpha
        type: probabilistic
        param: 0.8
        operation_strategies:
          - operation: op1
            type: probabilistic
            param: 0.2
          - operation: op2
            type: probabilistic
            param: 0.4
      - service: beta
        type: ratelimiting
        param: 5
```

ユーザーによって指定される設定がない場合、分散トレースプラットフォーム (Jaeger) は以下の設定を使用します。

デフォルトのサンプリング

```
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 1
```

4.2.6.5. 分散トレースの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。

表4.5 分散トレースストレージを定義するために Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator によって使用される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:`
`  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。実稼働環境では、分散トレースプラットフォーム (Jaeger) は永続ストレージの Elasticsearch をサポートします。	**memory**

パラメーター	説明	値	デフォルト値
storage: secretname:	シークレットの名前 (例:**tracing-secret**)。		該当なし
storage: options: {}	ストレージを定義する設定オプション。		

表4.6 Elasticsearch インデックスクリーナーのパラメーター

パラメーター	説明	値	デフォルト値
storage: esIndexCleaner: enabled:	Elasticsearch ストレージを使用する場合は、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**
storage: esIndexCleaner: numberOfDays:	インデックスの削除を待機する日数。	整数値	**7**
storage: esIndexCleaner: schedule:	Elasticsearch インデックスを消去する頻度に関するスケジュールを定義します。	cron 式	"55 23 * * *"

4.2.6.5.1. Elasticsearch インスタンスの自動プロビジョニング

Jaeger カスタムリソースをデプロイする場合に、Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator は、OpenShift Elasticsearch Operator を使用して、カスタムリソースファイルの **ストレージ** セクションで提供される設定に基づいて Elasticsearch クラスターを作成します。Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator は、次の設定が指定されている場合、Elasticsearch をプロビジョニングします。

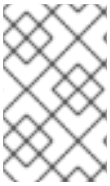
- **spec.storage:type** は **elasticsearch** に設定されている
- **spec.storage.elasticsearch.doNotProvision** は **false** に設定されている

- **spec.storage.options.es.server-urls** が定義されていない。つまり、OpenShift Elasticsearch Operator によってプロビジョニングされていない Elasticsearch インスタンスへの接続がない。

Elasticsearch をプロビジョニングする場合には、Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator は、Elasticsearch カスタムリソース **name** を Jaeger カスタムリソースの **spec.storage.elasticsearch.name** の値に設定します。**spec.storage.elasticsearch.name** に値を指定しない場合、Operator は **elasticsearch** を使用します。

制約

- namespace ごとにセルフプロビジョニングされた Elasticsearch インスタンスがある分散トレースプラットフォーム (Jaeger) 1 つだけを使用できます。Elasticsearch クラスターは単一の分散トレースプラットフォーム (Jaeger) インスタンスの専用のクラスターになります。
- namespace ごとに 1 つの Elasticsearch のみを使用できます。



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、Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator はインストールされた OpenShift 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。

以下の設定パラメーターは、**セルフプロビジョニングされた** Elasticsearch インスタンスに対するものです。これは、OpenShift Elasticsearch Operator を使用して Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator によって作成されるインスタンスです。セルフプロビジョニングされた Elasticsearch の設定オプションは、設定ファイルの **spec:storage:elasticsearch** の下で指定します。

表4.7 Elasticsearch リソース設定パラメーター

パラメーター	説明	値	デフォルト値
elasticsearch: properties: doNotProvision:	Elasticsearch インスタンスを Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator によってプロビジョニングするかどうかを指定するために使用します。	**true**/**false**	**true**
elasticsearch: properties: name:	Elasticsearch インスタンスの名前。Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator は、このパラメーターで指定された Elasticsearch インスタンスを使用して Elasticsearch に接続します。	string	**elasticsearch**

パラメーター	説明	値	デフォルト値
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	3
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	1
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。	バイト単位で指定します (例:200Ki、50Mi、5Gi)。例: 概念実証用 = 1Gi、最小デプロイメント = 16Gi*	16Gi
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。	バイト単位で指定します (例:200Ki、50Mi、5Gi)。例: 概念実証用 = 1Gi、最小デプロイメント = 16Gi*	
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、Red Hat OpenShift 分散トレースプラットフォーム (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
elasticsearch: useCertManagement:	分散トレーシングプラットフォーム (Jaeger) が OpenShift Elasticsearch Operator の証明書管理機能を使用するかどうかを指定するために使用します。この機能は、OpenShift Container Platform 4.7 の {logging-title} 5.2 に追加されたもので、新しい Jaeger デプロイメントに推奨される設定です。	**true**/**false**	**true**

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は、デフォルトで各 Pod に割り当てる設定を 16 Gi 未満にすることはできず、Pod ごとに最大 64 Gi を割り当てる必要があります。

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
```

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、distributed tracing platform (Jaeger) は **emptyDir** を使用します。OpenShift Elasticsearch Operator は、distributed tracing platform (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 で分散トレースプラットフォーム (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。

4.2.6.5.2. 既存の Elasticsearch インスタンスへの接続

分散トレースプラットフォームを使用したストレージには、既存の Elasticsearch クラスターを使用できます。**外部** Elasticsearch インスタンスとも呼ばれる既存の Elasticsearch クラスターは、Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator または OpenShift Elasticsearch Operator によってインストールされなかったインスタンスです。

次の設定が指定されている場合に、Jaeger カスタムリソースをデプロイすると、Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator は Elasticsearch をプロビジョニングしません。

- **spec.storage.elasticsearch.doNotProvision** が **true** に設定されている
- **spec.storage.options.es.server-urls** に値がある
- **spec.storage.elasticsearch.name** に値がある場合、または Elasticsearch インスタンス名が **elasticsearch** の場合。

Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator は、**spec.storage.elasticsearch.name** で指定された Elasticsearch インスタンスを使用して Elasticsearch に接続します。

制約

- distributed tracing platform (Jaeger) で OpenShift Container Platform 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分散トレースプラットフォーム (Jaeger) インスタンスの専用のクラスターになります。

以下の設定パラメーターは、**外部** Elasticsearch インスタンスとして知られる、既存の Elasticsearch インスタンス向けです。この場合は、カスタムリソースファイルの **spec:storage:options:es** で、Elasticsearch の設定オプションを指定します。

表4.8 汎用 ES 設定パラメーター

パラメーター	説明	値	デフォルト値
es: 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。	http://elasticsearch.<namespace>.svc:9200
es: max-doc-count:	Elasticsearch クエリーから返す最大ドキュメント数。これは集約にも適用されます。**es.max-doc-count** と **es.max-num-spans** の両方を設定する場合は、Elasticsearch は 2 つの内の小さい方の値を使用します。		10000
es: max-num-spans:	[**非推奨**: 今後のリリースで削除されます。代わりに **es.max-doc-count** を使用してください。] Elasticsearch のクエリーごとに、1 度にフェッチするスパンの最大数。**es.max-num-spans** と **es.max-doc-count** の両方を設定する場合、Elasticsearch は 2 つの内の小さい方の値を使用します。		10000
es: max-span-age:	Elasticsearch のスパンの最大ルックバック。		72h0m0s
es: sniffer:	Elasticsearch のスニファー設定。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効になっています。	**true**/ **false**	**false**

パラメーター	説明	値	デフォルト値
es: sniffer-tls-enabled:	Elasticsearch クラスターに対してスニッフィングする際に TLS を有効にするためのオプション。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効になっています。	**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

表4.9 ES データレプリケーションパラメーター

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

表4.10 ES インデックス設定パラメーター

パラメーター	説明	値	デフォルト値
es: create-index-templates:	**true** に設定されている場合は、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:	分散トレースプラットフォーム (Jaeger) インデックスのオプション接頭辞。たとえば、これを “production” に設定すると、“production-tracing-*” という名前のインデックスが作成されます。		

表4.11 ES バルクプロセッサー設定パラメーター

パラメーター	説明	値	デフォルト値
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

表4.12 ES TLS 設定パラメーター

パラメーター	説明	値	デフォルト値
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。		
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なっています。	**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。		

表4.13 ES アーカイブ設定パラメーター

パラメーター	説明	値	デフォルト値
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

パラメーター	説明	値	デフォルト値
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
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は、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:	分散トレースプラットフォーム (Jaeger) インデックスのオプション接頭辞。たとえば、これを “production” に設定すると、“production-tracing-*” という名前のインデックスが作成されます。		
es-archive: max-doc-count:	Elasticsearch クエリーから返す最大ドキュメント数。これは集約にも適用されます。		0

パラメーター	説明	値	デフォルト値
es-archive: max-num-spans:	[**非推奨**: 今後のリリースで削除されます。代わりに **es-archive.max-doc-count** を使用してください。] Elasticsearch のクエリーごとに、1度にフェッチするスパンの最大数。		0
es-archive: max-span-age:	Elasticsearch のスパン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。		
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ッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効になっています。	**true**/ **false**	**false**

パラメーター	説明	値	デフォルト値
es-archive: sniffer-tls- enabled:	Elasticsearch クラスターに対してスニッフィングする際に TLS を有効にするためのオプション。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効になっています。	**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なっています。	**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。		
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。		

パラメーター	説明	値	デフォルト値
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

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: tracing-secret
  volumeMounts:
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

以下の例は、ボリュームからマウントされる TLS CA 証明書およびシークレットに保存されるユーザー/パスワードを使用して外部 Elasticsearch クラスターを使用する Jaeger CR を示しています。

外部 Elasticsearch 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 (1)
        index-prefix: my-prefix
        tls: (2)
          ca: /es/certificates/ca.crt
    secretName: tracing-secret (3)
  volumeMounts: (4)
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

(1) デフォルト namespace で実行されている Elasticsearch サービスへの URL。

(2) TLS 設定。この場合は、CA 証明書のみを使用できますが、相互 TLS を使用する場合に es.tls.key および es.tls.cert を含めることもできます。

(3) 環境変数 ES_PASSWORD および ES_USERNAME を定義するシークレット。kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic により作成されます

(4) すべてのストレージコンポーネントにマウントされるボリュームのマウントとボリューム。

4.2.6.6. Elasticsearch を使用した証明書の管理

OpenShift Elasticsearch Operator を使用して、証明書を作成および管理できます。OpenShift Elasticsearch Operator を使用して証明書を管理すると、複数の Jaeger Collector で単一の Elasticsearch クラスターを使用することもできます。

重要

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

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

バージョン 2.4 以降、Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator は、Elasticsearch カスタムリソースで次のアノテーションを使用して、証明書の作成を OpenShift Elasticsearch Operator に委譲します。

- **logging.openshift.io/elasticsearch-cert-management: "true"**
- **logging.openshift.io/elasticsearch-cert.jaeger-<shared-es-node-name>: "user.jaeger"**
- **logging.openshift.io/elasticsearch-cert.curator- <shared-es-node-name>: "system.logging.curator"**

ここで、**<shared-es-node-name>** は Elasticsearch ノードの名前です。たとえば、 **custom-es** という名前の Elasticsearch ノードを作成する場合に、カスタムリソースは次の例のようになります。

アノテーションを表示する Elasticsearch CR の例

```
apiVersion: logging.openshift.io/v1
kind: Elasticsearch
metadata:
  annotations:
    logging.openshift.io/elasticsearch-cert-management: "true"
    logging.openshift.io/elasticsearch-cert.jaeger-custom-es: "user.jaeger"
    logging.openshift.io/elasticsearch-cert.curator-custom-es: "system.logging.curator"
  name: custom-es
spec:
  managementState: Managed
  nodeSpec:
    resources:
      limits:
        memory: 16Gi
      requests:
        cpu: 1
        memory: 16Gi
  nodes:
    - nodeCount: 3
      proxyResources: {}
      resources: {}
      roles:
        - master
        - client
        - data
      storage: {}
  redundancyPolicy: ZeroRedundancy
```

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- {logging-title} がデフォルト設定でクラスターにインストールされている。
- Elasticsearch ノードと Jaeger インスタンスが同じ namespace にデプロイされている。(例: **traceing-system**)。

Jaeger カスタムリソースで **spec.storage.elasticsearch.useCertManagement** を **true** に設定して、証明書管理を有効にします。

useCertManagement を示す例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      name: custom-es
      doNotProvision: true
      useCertManagement: true
```

Elasticsearch をプロビジョニングする場合には、Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator は、Elasticsearch カスタムリソース **name** を Jaeger カスタムリソースの **spec.storage.elasticsearch.name** の値に設定します。

証明書は OpenShift Elasticsearch Operator によってプロビジョニングされ、Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) Operator が証明書を挿入します。

4.2.6.7. クエリー設定オプション

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イスをホストしてそれらを表示するサービスです。

表4.14 Query を定義するために Red Hat OpenShift 分散トレースプラットフォーム (Jaeger) Operator によって使用されるパラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  query:`
`  replicas:`
	作成する Query レプリカの数を指定します。	整数 (例: **2**)	

表4.15 Query に渡される設定パラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  query:`
`  options: {}`
	Query サービスを定義する設定オプション。		

`options:` `  log-level:`	Query のロギングレベル。	使用できる値は、**debug**、**info**、**warn**、**error**、**fatal**、**panic** です。	

パラメーター	説明	値	デフォルト値
options: 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 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
```

4.2.6.8. Ingester 設定オプション

Ingester は、Kafka トピックから読み取り、Elasticsearch ストレージバックエンド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4.16 Ingester に渡される Jaeger パラメーター

パラメーター	説明	値
spec: ingester: options: {}	Ingester サービスを定義する設定オプション。	

パラメーター	説明	値
options: 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、デッドロックの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。
options: 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**
options: 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**)
options: log-level:	Ingester のロギングレベル。	使用できる値は、**debug**、**info**、**warn**、**error**、**fatal**、**dpanic**、**panic** です。

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
```

4.2.7. サイドカーコンテナーの挿入

Red Hat OpenShift distributed tracing platform (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。Red Hat OpenShift distributed tracing platform (Jaeger) Operator は、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。

4.2.7.1. サイドカーコンテナーの自動挿入

Red Hat OpenShift distributed tracing platform (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。サイドカーの自動挿入を有効にするには、**sidecar.jaegertracing.io/inject** アノテーションセットを文字列 **true** または **$ oc get jaegers** を実行して返される distributed tracing platform (Jaeger) インスタンス名に追加します。**true** を指定する場合、デプロイメントと同じ namespace に distributed tracing platform (Jaeger) インスタンスが 1 つだけ存在する必要があります。それ以外の場合、Operator はどの distributed tracing platform (Jaeger) インスタンスを使用するか判断できません。デプロイメントの distributed tracing platform (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。エージェントは、同じ namespace で利用可能な 1 つの distributed tracing platform (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
```

1 文字列 **true** または Jaeger インスタンスの名前のいずれかに設定します。

サイドカーコンテナーが挿入されると、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。

4.2.7.2. サイドカーコンテナーの手動挿入

Red Hat OpenShift distributed tracing platform (Jaeger) Operator は、Jaeger Agent サイドカーを Deployment ワークロードに自動挿入するだけです。**Deployments** 以外 (**StatefulSets**、**DaemonSets** など) のコントローラータイプの場合、仕様で Jaeger エージェントサイドカーを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、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。

4.3. アップグレード

重要

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

Operator Lifecycle Manager (OLM) 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。

更新時に、Red Hat OpenShift distributed tracing platform Operator は、マネージド distributed tracing platform インスタンスを Operator に関連付けられたバージョンにアップグレードします。Red Hat OpenShift distributed tracing platform (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 distributed tracing platform (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、Operator をインストールされている 1.10 から 1.11 にアップグレードすると、Operator は実行中の distributed tracing platform (Jaeger) インスタンスをスキャンし、それらも 1.11 にアップグレードします。

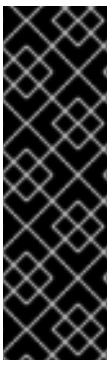
重要

Updating OpenShift Loggingの手順に従って OpenShift Elasticsearch Operator を更新していない場合は、Red Hat OpenShift distributed tracing platform (Jaeger) Operator を更新する前に更新を完了してください。

4.3.1. 関連情報

- Operator Lifecycle Manager の概念およびリソース
- インストール済み Operator の更新
- OpenShift Logging の更新

4.4. 削除中



重要

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger) は、非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

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

1. Red Hat OpenShift distributed tracing platform Pod をすべてシャットダウンします。
2. Red Hat OpenShift distributed tracing platform インスタンスをすべて削除します。
3. Red Hat OpenShift distributed tracing platform (Jaeger) Operator を削除します。
4. Red Hat build of OpenTelemetry Operator を削除します。

4.4.1. Web コンソールを使用して distributed tracing platform (Jaeger) インスタンスを削除する

Web コンソールの **Administrator** ビューで、distributed tracing platform (Jaeger) インスタンスを削除できます。

警告

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、すべてのデータが失われ、回復不能になります。Red Hat OpenShift distributed tracing platform (Jaeger) インスタンスが削除されても、永続ストレージ (Elasticsearch など) に保存されているデータは削除されません。

前提条件

- **cluster-admin** ロールを持つクラスター管理者として Web コンソールにログインしている。

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: **openshift-operators**) を選択します。
4. Red Hat OpenShift distributed tracing platform (Jaeger) Operator をクリックします。
5. **Jaeger** タブをクリックします。
6. 削除するインスタンスの横にある Options メニュー  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。

4.4.2. CLI を使用して distributed tracing platform (Jaeger) インスタンスを削除する

コマンドラインを使用して distributed tracing platform (Jaeger) インスタンスを削除できます。

前提条件

- **cluster-admin** ロールを持つクラスター管理者によるアクティブな OpenShift CLI (**oc**) セッション。

  ヒント

  - OpenShift CLI (**oc**) のバージョンが最新であり、OpenShift Container Platform バージョンと一致していることを確認してください。
  - **oc login** を実行します。

    ```
    $ oc login --username=<your_username>
    ```

手順

1. 以下のコマンドを実行して、OpenShift CLI (**oc**) でログインします。

   ```
   $ oc login --username=<NAMEOFUSER>
   ```

2. 以下のコマンドを実行して、distributed tracing platform (Jaeger) インスタンスを表示します。

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

   以下に例を示します。

   ```
   $ oc get deployments -n openshift-operators
   ```

   Operator の名前には、接尾辞の **-operator** が付きます。以下の例は、2 つの Red Hat OpenShift distributed tracing platform (Jaeger) Operator と 4 つの distributed tracing platform (Jaeger) インスタンスを示しています。

   ```
   $ oc get deployments -n openshift-operators
   ```

   以下のような出力が表示されます。

   ```
   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. distributed tracing platform (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>
   ```

   以下に例を示します。

   ```
   $ oc delete jaeger tracing2 -n openshift-operators
   ```

4. 削除を確認するには、**oc get deployments** コマンドを再度実行します。

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

以下に例を示します。

```
$ oc get deployments -n openshift-operators
```

以下の例のような出力が生成され、表示されます。

```
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
```

4.4.3. Red Hat OpenShift distributed tracing platform Operator を削除する

手順

1. クラスターからの Operator の削除 に記載された手順に従って、Red Hat OpenShift distributed tracing platform (Jaeger) Operator を削除します。

2. オプション: Red Hat OpenShift distributed tracing platform (Jaeger) Operator を削除してから、OpenShift Elasticsearch Operator を削除します。