



OpenShift Container Platform 4.16

Distributed tracing

Configuring and using the Network Observability Operator in OpenShift Container Platform

OpenShift Container Platform 4.16 Distributed tracing

Configuring and using the Network Observability Operator in OpenShift Container Platform

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Use the Network Observability Operator to observe and analyze network traffic flows for OpenShift Container Platform clusters.

Table of Contents

CHAPTER 1. RELEASE NOTES	6
1.1. RELEASE NOTES FOR RED HAT OPENSIFT DISTRIBUTED TRACING PLATFORM 3.2.1	6
1.1.1. Distributed tracing overview	6
1.1.2. CVEs	6
1.1.3. Red Hat OpenShift distributed tracing platform (Tempo)	6
1.1.3.1. Known issues	6
1.1.4. Red Hat OpenShift distributed tracing platform (Jaeger)	6
1.1.4.1. Known issues	6
1.1.5. Getting support	7
1.1.6. Making open source more inclusive	7
1.2. RELEASE NOTES FOR PAST RELEASES OF RED HAT OPENSIFT DISTRIBUTED TRACING PLATFORM	7
1.2.1. Distributed tracing overview	7
1.2.2. Release notes for Red Hat OpenShift distributed tracing platform 3.2	7
1.2.2.1. Red Hat OpenShift distributed tracing platform (Tempo)	8
1.2.2.1.1. Technology Preview features	8
1.2.2.1.2. New features and enhancements	8
1.2.2.1.3. Bug fixes	8
1.2.2.1.4. Known issues	8
1.2.2.2. Red Hat OpenShift distributed tracing platform (Jaeger)	9
1.2.2.2.1. Support for OpenShift Elasticsearch Operator	9
1.2.2.2.2. Deprecated functionality	9
1.2.2.2.3. New features and enhancements	9
1.2.2.2.4. Known issues	9
1.2.2.3. Release notes for Red Hat OpenShift distributed tracing platform 3.1.1	9
1.2.3.1. CVEs	9
1.2.3.2. Red Hat OpenShift distributed tracing platform (Tempo)	10
1.2.3.2.1. Known issues	10
1.2.3.3. Red Hat OpenShift distributed tracing platform (Jaeger)	10
1.2.3.3.1. Support for OpenShift Elasticsearch Operator	10
1.2.3.3.2. Deprecated functionality	10
1.2.3.3.3. Known issues	10
1.2.4. Release notes for Red Hat OpenShift distributed tracing platform 3.1	10
1.2.4.1. Red Hat OpenShift distributed tracing platform (Tempo)	11
1.2.4.1.1. New features and enhancements	11
1.2.4.1.2. Bug fixes	11
1.2.4.1.3. Known issues	11
1.2.4.2. Red Hat OpenShift distributed tracing platform (Jaeger)	11
1.2.4.2.1. Support for OpenShift Elasticsearch Operator	11
1.2.4.2.2. Deprecated functionality	12
1.2.4.2.3. New features and enhancements	12
1.2.4.2.4. Bug fixes	12
1.2.4.2.5. Known issues	12
1.2.5. Release notes for Red Hat OpenShift distributed tracing platform 3.0	12
1.2.5.1. Component versions in the Red Hat OpenShift distributed tracing platform 3.0	12
1.2.5.2. Red Hat OpenShift distributed tracing platform (Jaeger)	13
1.2.5.2.1. Deprecated functionality	13
1.2.5.2.2. New features and enhancements	13
1.2.5.2.3. Bug fixes	13
1.2.5.2.4. Known issues	13
1.2.5.3. Red Hat OpenShift distributed tracing platform (Tempo)	13
1.2.5.3.1. New features and enhancements	13

1.2.5.3.2. Bug fixes	14
1.2.5.3.3. Known issues	14
1.2.6. Release notes for Red Hat OpenShift distributed tracing platform 2.9.2	14
1.2.6.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.9.2	14
1.2.6.2. CVEs	14
1.2.6.3. Red Hat OpenShift distributed tracing platform (Jaeger)	14
1.2.6.3.1. Known issues	14
1.2.6.4. Red Hat OpenShift distributed tracing platform (Tempo)	15
1.2.6.4.1. Known issues	15
1.2.7. Release notes for Red Hat OpenShift distributed tracing platform 2.9.1	16
1.2.7.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.9.1	16
1.2.7.2. CVEs	16
1.2.7.3. Red Hat OpenShift distributed tracing platform (Jaeger)	17
1.2.7.3.1. Known issues	17
1.2.7.4. Red Hat OpenShift distributed tracing platform (Tempo)	17
1.2.7.4.1. Known issues	17
1.2.8. Release notes for Red Hat OpenShift distributed tracing platform 2.9	18
1.2.8.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.9	18
1.2.8.2. Red Hat OpenShift distributed tracing platform (Jaeger)	19
1.2.8.2.1. Bug fixes	19
1.2.8.2.2. Known issues	19
1.2.8.3. Red Hat OpenShift distributed tracing platform (Tempo)	19
1.2.8.3.1. New features and enhancements	19
1.2.8.3.2. Bug fixes	20
1.2.8.3.3. Known issues	20
1.2.9. Release notes for Red Hat OpenShift distributed tracing platform 2.8	21
1.2.9.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.8	21
1.2.9.2. Technology Preview features	22
1.2.9.3. Bug fixes	22
1.2.10. Release notes for Red Hat OpenShift distributed tracing platform 2.7	23
1.2.10.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.7	23
1.2.10.2. Bug fixes	23
1.2.11. Release notes for Red Hat OpenShift distributed tracing platform 2.6	23
1.2.11.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.6	23
1.2.11.2. Bug fixes	23
1.2.12. Release notes for Red Hat OpenShift distributed tracing platform 2.5	23
1.2.12.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.5	23
1.2.12.2. New features and enhancements	23
1.2.12.3. Bug fixes	24
1.2.13. Release notes for Red Hat OpenShift distributed tracing platform 2.4	24
1.2.13.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.4	24
1.2.13.2. New features and enhancements	24
1.2.13.3. Technology Preview features	24
1.2.13.4. Bug fixes	24
1.2.14. Release notes for Red Hat OpenShift distributed tracing platform 2.3	24
1.2.14.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.3.1	24
1.2.14.2. Component versions in the Red Hat OpenShift distributed tracing platform 2.3.0	24
1.2.14.3. New features and enhancements	25
1.2.14.4. Bug fixes	25
1.2.15. Release notes for Red Hat OpenShift distributed tracing platform 2.2	25
1.2.15.1. Technology Preview features	25
1.2.15.2. Bug fixes	25
1.2.16. Release notes for Red Hat OpenShift distributed tracing platform 2.1	25

1.2.16.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.1	25
1.2.16.2. Technology Preview features	25
1.2.16.3. Bug fixes	26
1.2.17. Release notes for Red Hat OpenShift distributed tracing platform 2.0	26
1.2.17.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.0	26
1.2.17.2. New features and enhancements	26
1.2.17.3. Technology Preview features	26
1.2.17.4. Bug fixes	27
1.2.18. Getting support	27
1.2.19. Making open source more inclusive	27
CHAPTER 2. DISTRIBUTED TRACING ARCHITECTURE	28
2.1. DISTRIBUTED TRACING ARCHITECTURE	28
2.1.1. Distributed tracing overview	28
2.1.2. Red Hat OpenShift distributed tracing platform features	28
2.1.3. Red Hat OpenShift distributed tracing platform architecture	29
2.1.4. Additional resources	30
CHAPTER 3. DISTRIBUTED TRACING PLATFORM (TEMPO)	31
3.1. INSTALLING	31
3.1.1. Installing the Tempo Operator	31
3.1.1.1. Installing the Tempo Operator by using the web console	31
3.1.1.2. Installing the Tempo Operator by using the CLI	32
3.1.2. Installing a TempoStack instance	33
3.1.2.1. Installing a TempoStack instance by using the web console	33
3.1.2.2. Installing a TempoStack instance by using the CLI	36
3.1.3. Installing a TempoMonolithic instance	39
3.1.3.1. Installing a TempoMonolithic instance by using the web console	40
3.1.3.2. Installing a TempoMonolithic instance by using the CLI	43
3.1.4. Object storage setup	46
3.1.5. Additional resources	47
3.2. CONFIGURING	47
3.2.1. Customizing your deployment	47
3.2.1.1. Default configuration options	47
3.2.1.2. Storage configuration	50
3.2.1.3. Query configuration options	52
3.2.1.3.1. Additional resources	55
3.2.1.4. Configuration of the monitor tab in Jaeger UI	55
3.2.1.4.1. OpenTelemetry Collector configuration	55
3.2.1.4.2. Tempo configuration	56
3.2.1.4.3. Span RED metrics and alerting rules	57
3.2.1.5. Multitenancy	58
3.2.2. Configuring monitoring and alerts	61
3.2.2.1. Configuring the TempoStack metrics and alerts	61
3.2.2.1.1. Additional resources	62
3.2.2.2. Configuring the Tempo Operator metrics and alerts	62
3.3. UPGRADING	62
3.3.1. Additional resources	62
3.4. REMOVING	62
3.4.1. Removing by using the web console	63
3.4.2. Removing by using the CLI	63
3.4.3. Additional resources	64
CHAPTER 4. DISTRIBUTED TRACING PLATFORM (JAEGER)	65

4.1. INSTALLING	65
4.1.1. Prerequisites	65
4.1.2. Red Hat OpenShift distributed tracing platform installation overview	66
4.1.3. Installing the OpenShift Elasticsearch Operator	66
4.1.4. Installing the Red Hat OpenShift distributed tracing platform Operator	67
4.2. CONFIGURING	68
4.2.1. Supported deployment strategies	69
4.2.2. Deploying the distributed tracing platform default strategy from the web console	70
4.2.2.1. Deploying the distributed tracing platform default strategy from the CLI	71
4.2.3. Deploying the distributed tracing platform production strategy from the web console	72
4.2.3.1. Deploying the distributed tracing platform production strategy from the CLI	74
4.2.4. Deploying the distributed tracing platform streaming strategy from the web console	75
4.2.4.1. Deploying the distributed tracing platform streaming strategy from the CLI	76
4.2.5. Validating your deployment	77
4.2.5.1. Accessing the Jaeger console	78
4.2.6. Customizing your deployment	78
4.2.6.1. Deployment best practices	78
4.2.6.2. Distributed tracing default configuration options	79
4.2.6.3. Jaeger Collector configuration options	82
4.2.6.4. Distributed tracing sampling configuration options	84
4.2.6.5. Distributed tracing storage configuration options	86
4.2.6.5.1. Auto-provisioning an Elasticsearch instance	87
4.2.6.5.2. Connecting to an existing Elasticsearch instance	91
4.2.6.6. Managing certificates with Elasticsearch	99
4.2.6.7. Query configuration options	101
4.2.6.8. Ingester configuration options	102
4.2.7. Injecting sidecars	104
4.2.7.1. Automatically injecting sidecars	104
4.2.7.2. Manually injecting sidecars	105
4.3. UPGRADING	106
4.3.1. Additional resources	106
4.4. REMOVING	106
4.4.1. Removing a distributed tracing platform (Jaeger) instance by using the web console	107
4.4.2. Removing a distributed tracing platform (Jaeger) instance by using the CLI	107
4.4.3. Removing the Red Hat OpenShift distributed tracing platform Operators	109

CHAPTER 1. RELEASE NOTES

1.1. RELEASE NOTES FOR RED HAT OPENSIFT DISTRIBUTED TRACING PLATFORM 3.2.1

1.1.1. Distributed tracing overview

As a service owner, you can use distributed tracing to instrument your services to gather insights into your service architecture. You can use the Red Hat OpenShift distributed tracing platform for monitoring, network profiling, and troubleshooting the interaction between components in modern, cloud-native, microservices-based applications.

With the distributed tracing platform, you can perform the following functions:

- Monitor distributed transactions
- Optimize performance and latency
- Perform root cause analysis

You can use the distributed tracing platform [in combination with](#) the [Red Hat build of OpenTelemetry](#).

This release of the Red Hat OpenShift distributed tracing platform includes the Red Hat OpenShift distributed tracing platform (Tempo) and the deprecated Red Hat OpenShift distributed tracing platform (Jaeger).

1.1.2. CVEs

This release fixes [CVE-2024-25062](#).

1.1.3. Red Hat OpenShift distributed tracing platform (Tempo)

The Red Hat OpenShift distributed tracing platform (Tempo) is provided through the Tempo Operator.

1.1.3.1. Known issues

There is currently a known issue:

- Currently, the distributed tracing platform (Tempo) fails on the IBM Z (**s390x**) architecture. ([TRACING-3545](#))

1.1.4. Red Hat OpenShift distributed tracing platform (Jaeger)

The Red Hat OpenShift distributed tracing platform (Jaeger) is provided through the Red Hat OpenShift distributed tracing platform Operator.



IMPORTANT

Jaeger does not use FIPS validated cryptographic modules.

1.1.4.1. Known issues

There is currently a known issue:

- Currently, Apache Spark is not supported.
- Currently, the streaming deployment via AMQ/Kafka is not supported on the IBM Z and IBM Power architectures.

1.1.5. Getting support

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the [Red Hat Customer Portal](#).

From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your cluster, you can use Insights in [OpenShift Cluster Manager](#). Insights provides details about issues and, if available, information on how to solve a problem.

If you have a suggestion for improving this documentation or have found an error, submit a [Jira issue](#) for the most relevant documentation component. Please provide specific details, such as the section name and OpenShift Container Platform version.

1.1.6. Making open source more inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

1.2. RELEASE NOTES FOR PAST RELEASES OF RED HAT OPENSIFT DISTRIBUTED TRACING PLATFORM

1.2.1. Distributed tracing overview

As a service owner, you can use distributed tracing to instrument your services to gather insights into your service architecture. You can use the Red Hat OpenShift distributed tracing platform for monitoring, network profiling, and troubleshooting the interaction between components in modern, cloud-native, microservices-based applications.

With the distributed tracing platform, you can perform the following functions:

- Monitor distributed transactions
- Optimize performance and latency
- Perform root cause analysis

You can use the distributed tracing platform [in combination with](#) the [Red Hat build of OpenTelemetry](#).

1.2.2. Release notes for Red Hat OpenShift distributed tracing platform 3.2

This release of the Red Hat OpenShift distributed tracing platform includes the Red Hat OpenShift distributed tracing platform (Tempo) and the deprecated Red Hat OpenShift distributed tracing platform (Jaeger).

1.2.2.1. Red Hat OpenShift distributed tracing platform (Tempo)

The Red Hat OpenShift distributed tracing platform (Tempo) is provided through the Tempo Operator.

1.2.2.1.1. Technology Preview features

This update introduces the following Technology Preview feature:

- Support for the Tempo monolithic deployment.



IMPORTANT

The Tempo monolithic deployment is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

1.2.2.1.2. New features and enhancements

This update introduces the following enhancements:

- Red Hat OpenShift distributed tracing platform (Tempo) 3.2 is based on the open source [Grafana Tempo 2.4.1](#).
- Allowing the overriding of resources per component.

1.2.2.1.3. Bug fixes

This update introduces the following bug fixes:

- Before this update, the Jaeger UI only displayed services that sent traces in the previous 15 minutes. With this update, the availability of the service and operation names can be configured by using the following field:
`spec.template.queryFrontend.jaegerQuery.servicesQueryDuration`. ([TRACING-3139](#))
- Before this update, the **query-frontend** pod might get stopped when out-of-memory (OOM) as a result of searching a large trace. With this update, resource limits can be set to prevent this issue. ([TRACING-4009](#))

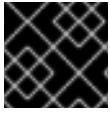
1.2.2.1.4. Known issues

There is currently a known issue:

- Currently, the distributed tracing platform (Tempo) fails on the IBM Z (**s390x**) architecture. ([TRACING-3545](#))

1.2.2.2. Red Hat OpenShift distributed tracing platform (Jaeger)

The Red Hat OpenShift distributed tracing platform (Jaeger) is provided through the Red Hat OpenShift distributed tracing platform Operator.



IMPORTANT

Jaeger does not use FIPS validated cryptographic modules.

1.2.2.2.1. Support for OpenShift Elasticsearch Operator

Red Hat OpenShift distributed tracing platform (Jaeger) 3.2 is supported for use with the OpenShift Elasticsearch Operator 5.6, 5.7, and 5.8.

1.2.2.2.2. Deprecated functionality

In the Red Hat OpenShift distributed tracing platform 3.2, Jaeger and support for Elasticsearch remain deprecated, and both are planned to be removed in a future release. Red Hat will provide support for these components and fixes for CVEs and bugs with critical and higher severity during the current release lifecycle, but these components will no longer receive feature enhancements. The Tempo Operator and the Red Hat build of OpenTelemetry are the preferred Operators for distributed tracing collection and storage. Users must adopt the OpenTelemetry and Tempo distributed tracing stack because it is the stack to be enhanced going forward.

In the Red Hat OpenShift distributed tracing platform 3.2, the Jaeger agent is deprecated and planned to be removed in the following release. Red Hat will provide bug fixes and support for the Jaeger agent during the current release lifecycle, but the Jaeger agent will no longer receive enhancements and will be removed. The OpenTelemetry Collector provided by the Red Hat build of OpenTelemetry is the preferred Operator for injecting the trace collector agent.

1.2.2.2.3. New features and enhancements

This update introduces the following enhancements for the distributed tracing platform (Jaeger):

- Red Hat OpenShift distributed tracing platform (Jaeger) 3.2 is based on the open source [Jaeger](#) release 1.57.0.

1.2.2.2.4. Known issues

There is currently a known issue:

- Currently, Apache Spark is not supported.
- Currently, the streaming deployment via AMQ/Kafka is not supported on the IBM Z and IBM Power architectures.

1.2.3. Release notes for Red Hat OpenShift distributed tracing platform 3.1.1

This release of the Red Hat OpenShift distributed tracing platform includes the Red Hat OpenShift distributed tracing platform (Tempo) and the deprecated Red Hat OpenShift distributed tracing platform (Jaeger).

1.2.3.1. CVEs

This release fixes [CVE-2023-39326](#).

1.2.3.2. Red Hat OpenShift distributed tracing platform (Tempo)

The Red Hat OpenShift distributed tracing platform (Tempo) is provided through the Tempo Operator.

1.2.3.2.1. Known issues

There are currently known issues:

- Currently, when used with the Tempo Operator, the Jaeger UI only displays services that have sent traces in the last 15 minutes. For services that did not send traces in the last 15 minutes, traces are still stored but not displayed in the Jaeger UI. ([TRACING-3139](#))
- Currently, the distributed tracing platform (Tempo) fails on the IBM Z (**s390x**) architecture. ([TRACING-3545](#))

1.2.3.3. Red Hat OpenShift distributed tracing platform (Jaeger)

The Red Hat OpenShift distributed tracing platform (Jaeger) is provided through the Red Hat OpenShift distributed tracing platform Operator.



IMPORTANT

Jaeger does not use FIPS validated cryptographic modules.

1.2.3.3.1. Support for OpenShift Elasticsearch Operator

Red Hat OpenShift distributed tracing platform (Jaeger) 3.1.1 is supported for use with the OpenShift Elasticsearch Operator 5.6, 5.7, and 5.8.

1.2.3.3.2. Deprecated functionality

In the Red Hat OpenShift distributed tracing platform 3.1.1, Jaeger and support for Elasticsearch remain deprecated, and both are planned to be removed in a future release. Red Hat will provide critical and above CVE bug fixes and support for these components during the current release lifecycle, but these components will no longer receive feature enhancements.

In the Red Hat OpenShift distributed tracing platform 3.1.1, Tempo provided by the Tempo Operator and the OpenTelemetry Collector provided by the Red Hat build of OpenTelemetry are the preferred Operators for distributed tracing collection and storage. The OpenTelemetry and Tempo distributed tracing stack is to be adopted by all users because this will be the stack that will be enhanced going forward.

1.2.3.3.3. Known issues

There are currently known issues:

- Currently, Apache Spark is not supported.
- Currently, the streaming deployment via AMQ/Kafka is not supported on the IBM Z and IBM Power architectures.

1.2.4. Release notes for Red Hat OpenShift distributed tracing platform 3.1

This release of the Red Hat OpenShift distributed tracing platform includes the Red Hat OpenShift distributed tracing platform (Tempo) and the deprecated Red Hat OpenShift distributed tracing platform (Jaeger).

1.2.4.1. Red Hat OpenShift distributed tracing platform (Tempo)

The Red Hat OpenShift distributed tracing platform (Tempo) is provided through the Tempo Operator.

1.2.4.1.1. New features and enhancements

This update introduces the following enhancements for the distributed tracing platform (Tempo):

- Red Hat OpenShift distributed tracing platform (Tempo) 3.1 is based on the open source [Grafana Tempo 2.3.1](#).
- Support for cluster-wide proxy environments.
- Support for TraceQL to Gateway component.

1.2.4.1.2. Bug fixes

This update introduces the following bug fixes for the distributed tracing platform (Tempo):

- Before this update, when a TempoStack instance was created with the **monitorTab** enabled in OpenShift Container Platform 4.15, the required **tempo-redmetrics-cluster-monitoring-view** ClusterRoleBinding was not created. This update resolves the issue by fixing the Operator RBAC for the monitor tab when the Operator is deployed in an arbitrary namespace. ([TRACING-3786](#))
- Before this update, when a TempoStack instance was created on an OpenShift Container Platform cluster with only an IPv6 networking stack, the compactor and ingestor pods ran in the **CrashLoopBackOff** state, resulting in multiple errors. This update provides support for IPv6 clusters. ([TRACING-3226](#))

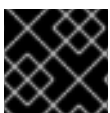
1.2.4.1.3. Known issues

There are currently known issues:

- Currently, when used with the Tempo Operator, the Jaeger UI only displays services that have sent traces in the last 15 minutes. For services that did not send traces in the last 15 minutes, traces are still stored but not displayed in the Jaeger UI. ([TRACING-3139](#))
- Currently, the distributed tracing platform (Tempo) fails on the IBM Z (**s390x**) architecture. ([TRACING-3545](#))

1.2.4.2. Red Hat OpenShift distributed tracing platform (Jaeger)

The Red Hat OpenShift distributed tracing platform (Jaeger) is provided through the Red Hat OpenShift distributed tracing platform Operator.



IMPORTANT

Jaeger does not use FIPS validated cryptographic modules.

1.2.4.2.1. Support for OpenShift Elasticsearch Operator

Red Hat OpenShift distributed tracing platform (Jaeger) 3.1 is supported for use with the OpenShift Elasticsearch Operator 5.6, 5.7, and 5.8.

1.2.4.2.2. Deprecated functionality

In the Red Hat OpenShift distributed tracing platform 3.1, Jaeger and support for Elasticsearch remain deprecated, and both are planned to be removed in a future release. Red Hat will provide critical and above CVE bug fixes and support for these components during the current release lifecycle, but these components will no longer receive feature enhancements.

In the Red Hat OpenShift distributed tracing platform 3.1, Tempo provided by the Tempo Operator and the OpenTelemetry Collector provided by the Red Hat build of OpenTelemetry are the preferred Operators for distributed tracing collection and storage. The OpenTelemetry and Tempo distributed tracing stack is to be adopted by all users because this will be the stack that will be enhanced going forward.

1.2.4.2.3. New features and enhancements

This update introduces the following enhancements for the distributed tracing platform (Jaeger):

- Red Hat OpenShift distributed tracing platform (Jaeger) 3.1 is based on the open source [Jaeger](#) release 1.53.0.

1.2.4.2.4. Bug fixes

This update introduces the following bug fix for the distributed tracing platform (Jaeger):

- Before this update, the connection target URL for the **jaeger-agent** container in the **jager-query** pod was overwritten with another namespace URL in OpenShift Container Platform 4.13. This was caused by a bug in the sidecar injection code in the **jaeger-operator**, causing nondeterministic **jaeger-agent** injection. With this update, the Operator prioritizes the Jaeger instance from the same namespace as the target deployment. ([TRACING-3722](#))

1.2.4.2.5. Known issues

There are currently known issues:

- Currently, Apache Spark is not supported.
- Currently, the streaming deployment via AMQ/Kafka is not supported on the IBM Z and IBM Power architectures.

1.2.5. Release notes for Red Hat OpenShift distributed tracing platform 3.0

1.2.5.1. Component versions in the Red Hat OpenShift distributed tracing platform 3.0

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.51.0
Red Hat OpenShift distributed tracing platform (Tempo)	Tempo	2.3.0

1.2.5.2. Red Hat OpenShift distributed tracing platform (Jaeger)

1.2.5.2.1. Deprecated functionality

In the Red Hat OpenShift distributed tracing platform 3.0, Jaeger and support for Elasticsearch are deprecated, and both are planned to be removed in a future release. Red Hat will provide critical and above CVE bug fixes and support for these components during the current release lifecycle, but these components will no longer receive feature enhancements.

In the Red Hat OpenShift distributed tracing platform 3.0, Tempo provided by the Tempo Operator and the OpenTelemetry Collector provided by the Red Hat build of OpenTelemetry are the preferred Operators for distributed tracing collection and storage. The OpenTelemetry and Tempo distributed tracing stack is to be adopted by all users because this will be the stack that will be enhanced going forward.

1.2.5.2.2. New features and enhancements

This update introduces the following enhancements for the distributed tracing platform (Jaeger):

- Support for the ARM architecture.
- Support for cluster-wide proxy environments.

1.2.5.2.3. Bug fixes

This update introduces the following bug fix for the distributed tracing platform (Jaeger):

- Before this update, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator used other images than **relatedImages**. This caused the **ImagePullBackOff** error in disconnected network environments when launching the **jaeger** pod because the **oc adm catalog mirror** command mirrors images specified in **relatedImages**. This update provides support for disconnected environments when using the **oc adm catalog mirror** CLI command. ([TRACING-3546](#))

1.2.5.2.4. Known issues

There is currently a known issue:

- Currently, Apache Spark is not supported.
- Currently, the streaming deployment via AMQ/Kafka is not supported on the IBM Z and IBM Power architectures.

1.2.5.3. Red Hat OpenShift distributed tracing platform (Tempo)

1.2.5.3.1. New features and enhancements

This update introduces the following enhancements for the distributed tracing platform (Tempo):

- Support for the ARM architecture.
- Support for span request count, duration, and error count (RED) metrics. The metrics can be visualized in the Jaeger console deployed as part of Tempo or in the web console in the **Observe** menu.

1.2.5.3.2. Bug fixes

This update introduces the following bug fixes for the distributed tracing platform (Tempo):

- Before this update, the **TempoStack** CRD was not accepting custom CA certificate despite the option to choose CA certificates. This update fixes support for the custom TLS CA option for connecting to object storage. ([TRACING-3462](#))
- Before this update, when mirroring the Red Hat OpenShift distributed tracing platform Operator images to a mirror registry for use in a disconnected cluster, the related Operator images for **tempo**, **tempo-gateway**, **opa-openshift**, and **tempo-query** were not mirrored. This update fixes support for disconnected environments when using the **oc adm catalog mirror** CLI command. ([TRACING-3523](#))
- Before this update, the query frontend service of the Red Hat OpenShift distributed tracing platform was using internal mTLS when gateway was not deployed. This caused endpoint failure errors. This update fixes mTLS when Gateway is not deployed. ([TRACING-3510](#))

1.2.5.3.3. Known issues

There are currently known issues:

- Currently, when used with the Tempo Operator, the Jaeger UI only displays services that have sent traces in the last 15 minutes. For services that did not send traces in the last 15 minutes, traces are still stored but not displayed in the Jaeger UI. ([TRACING-3139](#))
- Currently, the distributed tracing platform (Tempo) fails on the IBM Z (**s390x**) architecture. ([TRACING-3545](#))

1.2.6. Release notes for Red Hat OpenShift distributed tracing platform 2.9.2

1.2.6.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.9.2

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.47.0
Red Hat OpenShift distributed tracing platform (Tempo)	Tempo	2.1.1

1.2.6.2. CVEs

This release fixes [CVE-2023-46234](#).

1.2.6.3. Red Hat OpenShift distributed tracing platform (Jaeger)

1.2.6.3.1. Known issues

There are currently known issues:

- Apache Spark is not supported.

- The streaming deployment via AMQ/Kafka is unsupported on the IBM Z and IBM Power architectures.

1.2.6.4. Red Hat OpenShift distributed tracing platform (Tempo)



IMPORTANT

The Red Hat OpenShift distributed tracing platform (Tempo) is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

1.2.6.4.1. Known issues

There are currently known issues:

- Currently, the custom TLS CA option is not implemented for connecting to object storage. ([TRACING-3462](#))
- Currently, when used with the Tempo Operator, the Jaeger UI only displays services that have sent traces in the last 15 minutes. For services that did not send traces in the last 15 minutes, traces are still stored but not displayed in the Jaeger UI. ([TRACING-3139](#))
- Currently, the distributed tracing platform (Tempo) fails on the IBM Z (**s390x**) architecture. ([TRACING-3545](#))
- Currently, the Tempo query frontend service must not use internal mTLS when Gateway is not deployed. This issue does not affect the Jaeger Query API. The workaround is to disable mTLS. ([TRACING-3510](#))

Workaround

Disable mTLS as follows:

1. Open the Tempo Operator ConfigMap for editing by running the following command:

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

- 1** The project where the Tempo Operator is installed.

2. Disable the mTLS in the Operator configuration by updating the YAML file:

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

- Restart the Tempo Operator pod by running the following command:

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

- Missing images for running the Tempo Operator in restricted environments. The Red Hat OpenShift distributed tracing platform (Tempo) CSV is missing references to the operand images. ([TRACING-3523](#))

Workaround

Add the Tempo Operator related images in the mirroring tool to mirror the images to the registry:

```
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. Release notes for Red Hat OpenShift distributed tracing platform 2.9.1

1.2.7.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.9.1

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.47.0
Red Hat OpenShift distributed tracing platform (Tempo)	Tempo	2.1.1

1.2.7.2. CVEs

This release fixes [CVE-2023-44487](#).

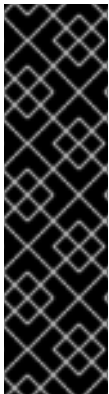
1.2.7.3. Red Hat OpenShift distributed tracing platform (Jaeger)

1.2.7.3.1. Known issues

There are currently known issues:

- Apache Spark is not supported.
- The streaming deployment via AMQ/Kafka is unsupported on the IBM Z and IBM Power architectures.

1.2.7.4. Red Hat OpenShift distributed tracing platform (Tempo)



IMPORTANT

The Red Hat OpenShift distributed tracing platform (Tempo) is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

1.2.7.4.1. Known issues

There are currently known issues:

- Currently, the custom TLS CA option is not implemented for connecting to object storage. ([TRACING-3462](#))
- Currently, when used with the Tempo Operator, the Jaeger UI only displays services that have sent traces in the last 15 minutes. For services that did not send traces in the last 15 minutes, traces are still stored but not displayed in the Jaeger UI. ([TRACING-3139](#))
- Currently, the distributed tracing platform (Tempo) fails on the IBM Z (**s390x**) architecture. ([TRACING-3545](#))
- Currently, the Tempo query frontend service must not use internal mTLS when Gateway is not deployed. This issue does not affect the Jaeger Query API. The workaround is to disable mTLS. ([TRACING-3510](#))

Workaround

Disable mTLS as follows:

1. Open the Tempo Operator ConfigMap for editing by running the following command:

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

- 1** The project where the Tempo Operator is installed.

2. Disable the mTLS in the Operator configuration by updating the YAML file:

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

3. Restart the Tempo Operator pod by running the following command:

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

- Missing images for running the Tempo Operator in restricted environments. The Red Hat OpenShift distributed tracing platform (Tempo) CSV is missing references to the operand images. ([TRACING-3523](#))

Workaround

Add the Tempo Operator related images in the mirroring tool to mirror the images to the registry:

```
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. Release notes for Red Hat OpenShift distributed tracing platform 2.9

1.2.8.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.9

Operator	Component	Version

Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.47.0
Red Hat OpenShift distributed tracing platform (Tempo)	Tempo	2.1.1

1.2.8.2. Red Hat OpenShift distributed tracing platform (Jaeger)

1.2.8.2.1. Bug fixes

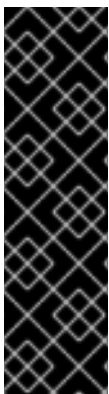
- Before this update, connection was refused due to a missing gRPC port on the **jaeger-query** deployment. This issue resulted in **transport: Error while dialing: dial tcp :16685: connect: connection refused** error message. With this update, the Jaeger Query gRPC port (16685) is successfully exposed on the Jaeger Query service. ([TRACING-3322](#))
- Before this update, the wrong port was exposed for **jaeger-production-query**, resulting in refused connection. With this update, the issue is fixed by exposing the Jaeger Query gRPC port (16685) on the Jaeger Query deployment. ([TRACING-2968](#))
- Before this update, when deploying Service Mesh on single-node OpenShift clusters in disconnected environments, the Jaeger pod frequently went into the **Pending** state. With this update, the issue is fixed. ([TRACING-3312](#))
- Before this update, the Jaeger Operator pod restarted with the default memory value due to the **reason: OOMKilled** error message. With this update, this issue is fixed by removing the resource limits. ([TRACING-3173](#))

1.2.8.2.2. Known issues

There are currently known issues:

- Apache Spark is not supported.
- The streaming deployment via AMQ/Kafka is unsupported on the IBM Z and IBM Power architectures.

1.2.8.3. Red Hat OpenShift distributed tracing platform (Tempo)



IMPORTANT

The Red Hat OpenShift distributed tracing platform (Tempo) is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

1.2.8.3.1. New features and enhancements

This release introduces the following enhancements for the distributed tracing platform (Tempo):

- Support the [operator maturity](#) Level IV, Deep Insights, which enables upgrading, monitoring, and alerting of the TempoStack instances and the Tempo Operator.
- Add Ingress and Route configuration for the Gateway.
- Support the **managed** and **unmanaged** states in the **TempoStack** custom resource.
- Expose the following additional ingestion protocols in the Distributor service: Jaeger Thrift binary, Jaeger Thrift compact, Jaeger gRPC, and Zipkin. When the Gateway is enabled, only the OpenTelemetry protocol (OTLP) gRPC is enabled.
- Expose the Jaeger Query gRPC endpoint on the Query Frontend service.
- Support multitenancy without Gateway authentication and authorization.

1.2.8.3.2. Bug fixes

- Before this update, the Tempo Operator was not compatible with disconnected environments. With this update, the Tempo Operator supports disconnected environments. ([TRACING-3145](#))
- Before this update, the Tempo Operator with TLS failed to start on OpenShift Container Platform. With this update, the mTLS communication is enabled between Tempo components, the Operand starts successfully, and the Jaeger UI is accessible. ([TRACING-3091](#))
- Before this update, the resource limits from the Tempo Operator caused error messages such as **reason: OOMKilled**. With this update, the resource limits for the Tempo Operator are removed to avoid such errors. ([TRACING-3204](#))

1.2.8.3.3. Known issues

There are currently known issues:

- Currently, the custom TLS CA option is not implemented for connecting to object storage. ([TRACING-3462](#))
- Currently, when used with the Tempo Operator, the Jaeger UI only displays services that have sent traces in the last 15 minutes. For services that did not send traces in the last 15 minutes, traces are still stored but not displayed in the Jaeger UI. ([TRACING-3139](#))
- Currently, the distributed tracing platform (Tempo) fails on the IBM Z (**s390x**) architecture. ([TRACING-3545](#))
- Currently, the Tempo query frontend service must not use internal mTLS when Gateway is not deployed. This issue does not affect the Jaeger Query API. The workaround is to disable mTLS. ([TRACING-3510](#))

Workaround

Disable mTLS as follows:

1. Open the Tempo Operator ConfigMap for editing by running the following command:

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


1 The project where the Tempo Operator is installed.

2. Disable the mTLS in the Operator configuration by updating the YAML file:

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

3. Restart the Tempo Operator pod by running the following command:

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

- Missing images for running the Tempo Operator in restricted environments. The Red Hat OpenShift distributed tracing platform (Tempo) CSV is missing references to the operand images. ([TRACING-3523](#))

Workaround

Add the Tempo Operator related images in the mirroring tool to mirror the images to the registry:

```
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. Release notes for Red Hat OpenShift distributed tracing platform 2.8

1.2.9.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.8

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.42
Red Hat OpenShift distributed tracing platform (Tempo)	Tempo	0.1.0

1.2.9.2. Technology Preview features

This release introduces support for the Red Hat OpenShift distributed tracing platform (Tempo) as a [Technology Preview](#) feature for Red Hat OpenShift distributed tracing platform.



IMPORTANT

The Red Hat OpenShift distributed tracing platform (Tempo) is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

The feature uses version 0.1.0 of the Red Hat OpenShift distributed tracing platform (Tempo) and version 2.0.1 of the upstream distributed tracing platform (Tempo) components.

You can use the distributed tracing platform (Tempo) to replace Jaeger so that you can use S3-compatible storage instead of Elasticsearch. Most users who use the distributed tracing platform (Tempo) instead of Jaeger will not notice any difference in functionality because the distributed tracing platform (Tempo) supports the same ingestion and query protocols as Jaeger and uses the same user interface.

If you enable this Technology Preview feature, note the following limitations of the current implementation:

- The distributed tracing platform (Tempo) currently does not support disconnected installations. ([TRACING-3145](#))
- When you use the Jaeger user interface (UI) with the distributed tracing platform (Tempo), the Jaeger UI lists only services that have sent traces within the last 15 minutes. For services that have not sent traces within the last 15 minutes, those traces are still stored even though they are not visible in the Jaeger UI. ([TRACING-3139](#))

Expanded support for the Tempo Operator is planned for future releases of the Red Hat OpenShift distributed tracing platform. Possible additional features might include support for TLS authentication, multitenancy, and multiple clusters. For more information about the Tempo Operator, see the [Tempo community documentation](#).

1.2.9.3. Bug fixes

This release addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.10. Release notes for Red Hat OpenShift distributed tracing platform 2.7

1.2.10.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.7

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.39

1.2.10.2. Bug fixes

This release addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.11. Release notes for Red Hat OpenShift distributed tracing platform 2.6

1.2.11.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.6

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.38

1.2.11.2. Bug fixes

This release addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.12. Release notes for Red Hat OpenShift distributed tracing platform 2.5

1.2.12.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.5

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.36

1.2.12.2. New features and enhancements

This release introduces support for ingesting OpenTelemetry protocol (OTLP) to the Red Hat OpenShift distributed tracing platform (Jaeger) Operator. The Operator now automatically enables the OTLP ports:

- Port 4317 for the OTLP gRPC protocol.
- Port 4318 for the OTLP HTTP protocol.

This release also adds support for collecting Kubernetes resource attributes to the Red Hat build of OpenTelemetry Operator.

1.2.12.3. Bug fixes

This release addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.13. Release notes for Red Hat OpenShift distributed tracing platform 2.4

1.2.13.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.4

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.34.1

1.2.13.2. New features and enhancements

This release adds support for auto-provisioning certificates using the OpenShift Elasticsearch Operator.

Self-provisioning by using the Red Hat OpenShift distributed tracing platform (Jaeger) Operator to call the OpenShift Elasticsearch Operator during installation.

+



IMPORTANT

When upgrading to the Red Hat OpenShift distributed tracing platform 2.4, the Operator recreates the Elasticsearch instance, which might take five to ten minutes. Distributed tracing will be down and unavailable for that period.

1.2.13.3. Technology Preview features

Creating the Elasticsearch instance and certificates first and then configuring the distributed tracing platform (Jaeger) to use the certificate is a [Technology Preview](#) for this release.

1.2.13.4. Bug fixes

This release addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.14. Release notes for Red Hat OpenShift distributed tracing platform 2.3

1.2.14.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.3.1

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.30.2

1.2.14.2. Component versions in the Red Hat OpenShift distributed tracing platform 2.3.0

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.30.1

1.2.14.3. New features and enhancements

With this release, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator is now installed to the **openshift-distributed-tracing** namespace by default. Before this update, the default installation had been in the **openshift-operators** namespace.

1.2.14.4. Bug fixes

This release addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.15. Release notes for Red Hat OpenShift distributed tracing platform 2.2

1.2.15.1. Technology Preview features

The unsupported OpenTelemetry Collector components included in the 2.1 release are removed.

1.2.15.2. Bug fixes

This release of the Red Hat OpenShift distributed tracing platform addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.16. Release notes for Red Hat OpenShift distributed tracing platform 2.1

1.2.16.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.1

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.29.1

1.2.16.2. Technology Preview features

- This release introduces a breaking change to how to configure certificates in the OpenTelemetry custom resource file. With this update, the **ca_file** moves under **tls** in the custom resource, as shown in the following examples.

CA file configuration for OpenTelemetry version 0.33

```
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"
```

CA file configuration for OpenTelemetry version 0.41.1

```
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. Bug fixes

This release addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.17. Release notes for Red Hat OpenShift distributed tracing platform 2.0

1.2.17.1. Component versions in the Red Hat OpenShift distributed tracing platform 2.0

Operator	Component	Version
Red Hat OpenShift distributed tracing platform (Jaeger)	Jaeger	1.28.0

1.2.17.2. New features and enhancements

This release introduces the following new features and enhancements:

- Rebrands Red Hat OpenShift Jaeger as the Red Hat OpenShift distributed tracing platform.
- Updates Red Hat OpenShift distributed tracing platform (Jaeger) Operator to Jaeger 1.28. Going forward, the Red Hat OpenShift distributed tracing platform will only support the **stable** Operator channel. Channels for individual releases are no longer supported.
- Adds support for OpenTelemetry protocol (OTLP) to the Query service.
- Introduces a new distributed tracing icon that appears in the OperatorHub.
- Includes rolling updates to the documentation to support the name change and new features.

1.2.17.3. Technology Preview features

This release adds the Red Hat build of OpenTelemetry as a [Technology Preview](#), which you install using the Red Hat build of OpenTelemetry Operator. Red Hat build of OpenTelemetry is based on the [OpenTelemetry](#) APIs and instrumentation. The Red Hat build of OpenTelemetry includes the OpenTelemetry Operator and Collector. You can use the Collector to receive traces in the OpenTelemetry or Jaeger protocol and send the trace data to the Red Hat OpenShift distributed

tracing platform. Other capabilities of the Collector are not supported at this time. The OpenTelemetry Collector allows developers to instrument their code with vendor agnostic APIs, avoiding vendor lock-in and enabling a growing ecosystem of observability tooling.

1.2.17.4. Bug fixes

This release addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.18. Getting support

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the [Red Hat Customer Portal](#).

From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your cluster, you can use Insights in [OpenShift Cluster Manager](#). Insights provides details about issues and, if available, information on how to solve a problem.

If you have a suggestion for improving this documentation or have found an error, submit a [Jira issue](#) for the most relevant documentation component. Please provide specific details, such as the section name and OpenShift Container Platform version.

1.2.19. Making open source more inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

CHAPTER 2. DISTRIBUTED TRACING ARCHITECTURE

2.1. DISTRIBUTED TRACING ARCHITECTURE

Every time a user takes an action in an application, a request is executed by the architecture that may require dozens of different services to participate to produce a response. Red Hat OpenShift distributed tracing platform lets you perform distributed tracing, which records the path of a request through various microservices that make up an application.

Distributed tracing is a technique that is used to tie the information about different units of work together – usually executed in different processes or hosts – to understand a whole chain of events in a distributed transaction. Developers can visualize call flows in large microservice architectures with distributed tracing. It is valuable for understanding serialization, parallelism, and sources of latency.

Red Hat OpenShift distributed tracing platform records the execution of individual requests across the whole stack of microservices, and presents them as traces. A *trace* is a data/execution path through the system. An end-to-end trace is comprised of one or more spans.

A *span* represents a logical unit of work in Red Hat OpenShift distributed tracing platform that has an operation name, the start time of the operation, and the duration, as well as potentially tags and logs. Spans may be nested and ordered to model causal relationships.

2.1.1. Distributed tracing overview

As a service owner, you can use distributed tracing to instrument your services to gather insights into your service architecture. You can use the Red Hat OpenShift distributed tracing platform for monitoring, network profiling, and troubleshooting the interaction between components in modern, cloud-native, microservices-based applications.

With the distributed tracing platform, you can perform the following functions:

- Monitor distributed transactions
- Optimize performance and latency
- Perform root cause analysis

2.1.2. Red Hat OpenShift distributed tracing platform features

Red Hat OpenShift distributed tracing platform provides the following capabilities:

- Integration with Kiali – When properly configured, you can view distributed tracing platform data from the Kiali console.
- High scalability – The distributed tracing platform back end is designed to have no single points of failure and to scale with the business needs.
- Distributed Context Propagation – Enables you to connect data from different components together to create a complete end-to-end trace.
- Backwards compatibility with Zipkin – Red Hat OpenShift distributed tracing platform has APIs that enable it to be used as a drop-in replacement for Zipkin, but Red Hat is not supporting Zipkin compatibility in this release.

2.1.3. Red Hat OpenShift distributed tracing platform architecture

Red Hat OpenShift distributed tracing platform is made up of several components that work together to collect, store, and display tracing data.

- **Red Hat OpenShift distributed tracing platform (Tempo)**- This component is based on the open source [Grafana Tempo project](#).
 - **Gateway** – The Gateway handles authentication, authorization, and forwarding requests to the Distributor or Query front-end service.
 - **Distributor** – The Distributor accepts spans in multiple formats including Jaeger, OpenTelemetry, and Zipkin. It routes spans to Ingesters by hashing the **traceID** and using a distributed consistent hash ring.
 - **Ingestor** – The Ingestor batches a trace into blocks, creates bloom filters and indexes, and then flushes it all to the back end.
 - **Query Frontend** – The Query Frontend is responsible for sharding the search space for an incoming query. The search query is then sent to the Queriers. The Query Frontend deployment exposes the Jaeger UI through the Tempo Query sidecar.
 - **Querier** – The Querier is responsible for finding the requested trace ID in either the Ingesters or the back-end storage. Depending on parameters, it can query the Ingesters and pull Bloom indexes from the back end to search blocks in object storage.
 - **Compactor** – The Compactors stream blocks to and from the back-end storage to reduce the total number of blocks.
- **Red Hat build of OpenTelemetry**- This component is based on the open source [OpenTelemetry project](#).
 - **OpenTelemetry Collector** – The OpenTelemetry Collector is a vendor-agnostic way to receive, process, and export telemetry data. The OpenTelemetry Collector supports open-source observability data formats, for example, Jaeger and Prometheus, sending to one or more open-source or commercial back-ends. The Collector is the default location instrumentation libraries export their telemetry data.
- **Red Hat OpenShift distributed tracing platform (Jaeger)**- This component is based on the open source [Jaeger project](#).
 - **Client** (Jaeger client, Tracer, Reporter, instrumented application, client libraries)- The distributed tracing platform (Jaeger) clients are language-specific implementations of the OpenTracing API. They can be used to instrument applications for distributed tracing either manually or with a variety of existing open source frameworks, such as Camel (Fuse), Spring Boot (RHOAR), MicroProfile (RHOAR/Thorntail), Wildfly (EAP), and many more, that are already integrated with OpenTracing.
 - **Agent** (Jaeger agent, Server Queue, Processor Workers) – The distributed tracing platform (Jaeger) agent is a network daemon that listens for spans sent over User Datagram Protocol (UDP), which it batches and sends to the Collector. The agent is meant to be placed on the same host as the instrumented application. This is typically accomplished by having a sidecar in container environments such as Kubernetes.
 - **Jaeger Collector** (Collector, Queue, Workers) – Similar to the Jaeger agent, the Jaeger Collector receives spans and places them in an internal queue for processing. This allows the Jaeger Collector to return immediately to the client/agent instead of waiting for the span to make its way to the storage.

- **Storage** (Data Store) - Collectors require a persistent storage backend. Red Hat OpenShift distributed tracing platform (Jaeger) has a pluggable mechanism for span storage. Red Hat OpenShift distributed tracing platform (Jaeger) supports the Elasticsearch storage.
- **Query** (Query Service) - Query is a service that retrieves traces from storage.
- **Ingester** (Ingester Service) - Red Hat OpenShift distributed tracing platform can use Apache Kafka as a buffer between the Collector and the actual Elasticsearch backing storage. Ingester is a service that reads data from Kafka and writes to the Elasticsearch storage backend.
- **Jaeger Console** - With the Red Hat OpenShift distributed tracing platform (Jaeger) user interface, you can visualize your distributed tracing data. On the Search page, you can find traces and explore details of the spans that make up an individual trace.

2.1.4. Additional resources

- [Red Hat build of OpenTelemetry](#)

CHAPTER 3. DISTRIBUTED TRACING PLATFORM (TEMPO)

3.1. INSTALLING

Installing the distributed tracing platform (Tempo) requires the Tempo Operator and choosing which type of deployment is best for your use case:

- For microservices mode, deploy a TempoStack instance in a dedicated OpenShift project.
- For monolithic mode, deploy a TempoMonolithic instance in a dedicated OpenShift project.



IMPORTANT

Using object storage requires setting up a supported object store and creating a secret for the object store credentials before deploying a TempoStack or TempoMonolithic instance.

3.1.1. Installing the Tempo Operator

You can install the Tempo Operator by using the web console or the command line.

3.1.1.1. Installing the Tempo Operator by using the web console

You can install the Tempo Operator from the **Administrator** view of the web console.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.
- You have completed setting up the required object storage by a supported provider: [Red Hat OpenShift Data Foundation](#), [MinIO](#), [Amazon S3](#), [Azure Blob Storage](#), [Google Cloud Storage](#). For more information, see "Object storage setup".

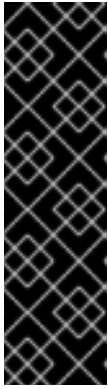


WARNING

Object storage is required and not included with the distributed tracing platform (Tempo). You must choose and set up object storage by a supported provider before installing the distributed tracing platform (Tempo).

Procedure

1. Go to **Operators** → **OperatorHub** and search for **Tempo Operator**.
2. Select the **Tempo Operator** that is **provided by Red Hat**



IMPORTANT

The following selections are the default presets for this Operator:

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

3. Select the **Enable Operator recommended cluster monitoring on this Namespace** checkbox.
4. Select **Install** → **Install** → **View Operator**.

Verification

- In the **Details** tab of the page of the installed Operator, under **ClusterServiceVersion details**, verify that the installation **Status** is **Succeeded**.

3.1.1.2. Installing the Tempo Operator by using the CLI

You can install the Tempo Operator from the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

TIP

- Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
- Run **oc login**:

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

- You have completed setting up the required object storage by a supported provider: [Red Hat OpenShift Data Foundation](#), [MinIO](#), [Amazon S3](#), [Azure Blob Storage](#), [Google Cloud Storage](#). For more information, see "Object storage setup".



WARNING

Object storage is required and not included with the distributed tracing platform (Tempo). You must choose and set up object storage by a supported provider before installing the distributed tracing platform (Tempo).

Procedure

1. Create a project for the Tempo Operator by running the following command:

```
$ 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. Create an Operator group by running the following command:

```
$ 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. Create a subscription by running the following command:

```
$ 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
```

Verification

- Check the Operator status by running the following command:

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

3.1.2. Installing a TempoStack instance

You can install a TempoStack instance by using the web console or the command line.

3.1.2.1. Installing a TempoStack instance by using the web console

You can install a TempoStack instance from the **Administrator** view of the web console.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.
- You have completed setting up the required object storage by a supported provider: [Red Hat OpenShift Data Foundation](#), [MinIO](#), [Amazon S3](#), [Azure Blob Storage](#), [Google Cloud Storage](#). For more information, see "Object storage setup".



WARNING

Object storage is required and not included with the distributed tracing platform (Tempo). You must choose and set up object storage by a supported provider before installing the distributed tracing platform (Tempo).

Procedure

1. Go to **Home** → **Projects** → **Create Project** to create a project of your choice for the TempoStack instance that you will create in a subsequent step.
2. Go to **Workloads** → **Secrets** → **Create** → **From YAML** to create a secret for your object storage bucket in the project that you created for the TempoStack instance. For more information, see "Object storage setup".

Example secret for Amazon S3 and MinIO storage

```
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. Create a TempoStack instance.



NOTE

You can create multiple TempoStack instances in separate projects on the same cluster.

- a. Go to **Operators** → **Installed Operators**.
- b. Select **TempoStack** → **Create TempoStack** → **YAML view**.
- c. In the **YAML view**, customize the **TempoStack** custom resource (CR):

```

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

```

- ❶ The secret you created in step 2 for the object storage that had been set up as one of the prerequisites.
- ❷ The value of the **name** in the **metadata** of the secret.
- ❸ The accepted values are **azure** for Azure Blob Storage; **gcs** for Google Cloud Storage; and **s3** for Amazon S3, MinIO, or Red Hat OpenShift Data Foundation.

Example of a TempoStack CR for AWS S3 and MinIO storage

```

apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage: ❶
    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 In this example, the object storage was set up as one of the prerequisites, and the object storage secret was created in step 2.
- 2 The stack deployed in this example is configured to receive Jaeger Thrift over HTTP and OpenTelemetry Protocol (OTLP), which permits visualizing the data with the Jaeger UI.

d. Select **Create**.

Verification

1. Use the **Project:** dropdown list to select the project of the **TempoStack** instance.
2. Go to **Operators** → **Installed Operators** to verify that the **Status** of the **TempoStack** instance is **Condition: Ready**.
3. Go to **Workloads** → **Pods** to verify that all the component pods of the **TempoStack** instance are running.
4. Access the Tempo console:
 - a. Go to **Networking** → **Routes** and **Ctrl+F** to search for **tempo**.
 - b. In the **Location** column, open the URL to access the Tempo console.



NOTE

The Tempo console initially shows no trace data following the Tempo console installation.

3.1.2.2. Installing a TempoStack instance by using the CLI

You can install a TempoStack instance from the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

TIP

- Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
- Run the **oc login** command:

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


- You have completed setting up the required object storage by a supported provider: [Red Hat OpenShift Data Foundation](#), [MinIO](#), [Amazon S3](#), [Azure Blob Storage](#), [Google Cloud Storage](#). For more information, see "Object storage setup".



WARNING

Object storage is required and not included with the distributed tracing platform (Tempo). You must choose and set up object storage by a supported provider before installing the distributed tracing platform (Tempo).

Procedure

- Run the following command to create a project of your choice for the TempoStack instance that you will create in a subsequent step:

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

- In the project that you created for the TempoStack instance, create a secret for your object storage bucket by running the following command:

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

For more information, see "Object storage setup".

Example secret for Amazon S3 and MinIO storage

```
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
```

- Create a TempoStack instance in the project that you created for it:

**NOTE**

You can create multiple TempoStack instances in separate projects on the same cluster.

- a. Customize the **TempoStack** custom resource (CR):

```

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

```

- ❶ The secret you created in step 2 for the object storage that had been set up as one of the prerequisites.
- ❷ The value of the **name** in the **metadata** of the secret.
- ❸ The accepted values are **azure** for Azure Blob Storage; **gcs** for Google Cloud Storage; and **s3** for Amazon S3, MinIO, or Red Hat OpenShift Data Foundation.

Example of a TempoStack CR for AWS S3 and MinIO storage

```

apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage: ❶
    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 In this example, the object storage was set up as one of the prerequisites, and the object storage secret was created in step 2.
- 2 The stack deployed in this example is configured to receive Jaeger Thrift over HTTP and OpenTelemetry Protocol (OTLP), which permits visualizing the data with the Jaeger UI.

b. Apply the customized CR by running the following command:

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

Verification

1. Verify that the **status** of all TempoStack **components** is **Running** and the **conditions** are **type: Ready** by running the following command:

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

2. Verify that all the TempoStack component pods are running by running the following command:

```
$ oc get pods
```

3. Access the Tempo console:

a. Query the route details by running the following command:

```
$ oc get route
```

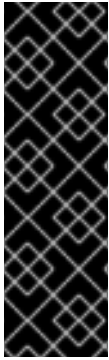
b. Open https://<route_from_previous_step> in a web browser.



NOTE

The Tempo console initially shows no trace data following the Tempo console installation.

3.1.3. Installing a TempoMonolithic instance



IMPORTANT

The TempoMonolithic instance is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

You can install a TempoMonolithic instance by using the web console or the command line.

The **TempoMonolithic** custom resource (CR) creates a Tempo deployment in monolithic mode. All components of the Tempo deployment, such as the compactor, distributor, ingester, querier, and query frontend, are contained in a single container.

A TempoMonolithic instance supports storing traces in in-memory storage, a persistent volume, or object storage.

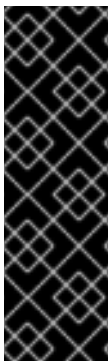
Tempo deployment in monolithic mode is preferred for a small deployment, demonstration, testing, and as a migration path of the Red Hat OpenShift distributed tracing platform (Jaeger) all-in-one deployment.



NOTE

The monolithic deployment of Tempo does not scale horizontally. If you require horizontal scaling, use the **TempoStack** CR for a Tempo deployment in microservices mode.

3.1.3.1. Installing a TempoMonolithic instance by using the web console



IMPORTANT

The TempoMonolithic instance is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

You can install a TempoMonolithic instance from the **Administrator** view of the web console.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.

Procedure

1. Go to **Home** → **Projects** → **Create Project** to create a project of your choice for the **TempoMonolithic** instance that you will create in a subsequent step.
2. Decide which type of supported storage to use for storing traces: in-memory storage, a persistent volume, or object storage.



IMPORTANT

Object storage is not included with the distributed tracing platform (Tempo) and requires setting up an object store by a supported provider: [Red Hat OpenShift Data Foundation](#), [MinIO](#), [Amazon S3](#), [Azure Blob Storage](#), or [Google Cloud Storage](#).

Additionally, opting for object storage requires creating a secret for your object storage bucket in the project that you created for the **TempoMonolithic** instance. You can do this in **Workloads** → **Secrets** → **Create** → **From YAML**.

For more information, see "Object storage setup".

Example secret for Amazon S3 and MinIO storage

```
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. Create a **TempoMonolithic** instance:



NOTE

You can create multiple **TempoMonolithic** instances in separate projects on the same cluster.

- a. Go to **Operators** → **Installed Operators**.
- b. Select **TempoMonolithic** → **Create TempoMonolithic** → **YAML view**.
- c. In the **YAML view**, customize the **TempoMonolithic** custom resource (CR).
The following **TempoMonolithic** CR creates a TempoMonolithic deployment with trace ingestion over OTLP/gRPC and OTLP/HTTP, storing traces in a supported type of storage and exposing Jaeger UI via a route:

```
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 Type of storage for storing traces: in-memory storage, a persistent volume, or object storage. The value for the **tmpfs** in-memory storage is **memory**. The value for a persistent volume is **pv**. The accepted values for object storage are **s3**, **gcs**, or **azure**, depending on the used object store type.
- 2 Memory size: For in-memory storage, this means the size of the **tmpfs** volume, where the default is **2Gi**. For a persistent volume, this means the size of the persistent volume claim, where the default is **10Gi**. For object storage, this means the size of the persistent volume claim for the Tempo WAL, where the default is **10Gi**.
- 3 Optional: For object storage, the type of object storage. The accepted values are **s3**, **gcs**, and **azure**, depending on the used object store type.
- 4 Optional: For object storage, the value of the **name** in the **metadata** of the storage secret. The storage secret must be in the same namespace as the TempoMonolithic instance and contain the fields specified in "Table 1. Required secret parameters" in the section "Object storage setup".
- 5 Enables the Jaeger UI.
- 6 Enables creation of a route for the Jaeger UI.

d. Select **Create**.

Verification

1. Use the **Project**: dropdown list to select the project of the **TempoMonolithic** instance.
2. Go to **Operators** → **Installed Operators** to verify that the **Status** of the **TempoMonolithic** instance is **Condition: Ready**.
3. Go to **Workloads** → **Pods** to verify that the pod of the **TempoMonolithic** instance is running.
4. Access the Jaeger UI:
 - a. Go to **Networking** → **Routes** and **Ctrl+F** to search for **jaegerui**.



NOTE

The Jaeger UI uses the **tempo-
<metadata_name_of_TempoMonolithic_CR>-jaegerui** route.

- b. In the **Location** column, open the URL to access the Jaeger UI.

- When the pod of the **TempoMonolithic** instance is ready, you can send traces to the **tempo-`<metadata_name_of_TempoMonolithic_CR>:4317`** (OTLP/gRPC) and **tempo-`<metadata_name_of_TempoMonolithic_CR>:4318`** (OTLP/HTTP) endpoints inside the cluster.

The Tempo API is available at the **tempo-`<metadata_name_of_TempoMonolithic_CR>:3200`** endpoint inside the cluster.

3.1.3.2. Installing a TempoMonolithic instance by using the CLI



IMPORTANT

The TempoMonolithic instance is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

You can install a TempoMonolithic instance from the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

TIP

- Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
- Run the **oc login** command:

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

Procedure

- Run the following command to create a project of your choice for the TempoMonolithic instance that you will create in a subsequent step:

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

- Decide which type of supported storage to use for storing traces: in-memory storage, a persistent volume, or object storage.

IMPORTANT

Object storage is not included with the distributed tracing platform (Tempo) and requires setting up an object store by a supported provider: [Red Hat OpenShift Data Foundation](#), [MinIO](#), [Amazon S3](#), [Azure Blob Storage](#), or [Google Cloud Storage](#).

Additionally, opting for object storage requires creating a secret for your object storage bucket in the project that you created for the TempoMonolithic instance. You can do this by running the following command:

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

For more information, see "Object storage setup".

Example secret for Amazon S3 and MinIO storage

```
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. Create a TempoMonolithic instance in the project that you created for it.

TIP

You can create multiple TempoMonolithic instances in separate projects on the same cluster.

- a. Customize the **TempoMonolithic** custom resource (CR).

The following **TempoMonolithic** CR creates a TempoMonolithic deployment with trace ingestion over OTLP/gRPC and OTLP/HTTP, storing traces in a supported type of storage and exposing Jaeger UI via a route:

```
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 Type of storage for storing traces: in-memory storage, a persistent volume, or object storage. The value for the **tmpfs** in-memory storage is **memory**. The value for a persistent volume is **pv**. The accepted values for object storage are **s3**, **gcs**, or **azure**, depending on the used object store type.
- 2 Memory size: For in-memory storage, this means the size of the **tmpfs** volume, where the default is **2Gi**. For a persistent volume, this means the size of the persistent volume claim, where the default is **10Gi**. For object storage, this means the size of the persistent volume claim for the Tempo WAL, where the default is **10Gi**.
- 3 Optional: For object storage, the type of object storage. The accepted values are **s3**, **gcs**, and **azure**, depending on the used object store type.
- 4 Optional: For object storage, the value of the **name** in the **metadata** of the storage secret. The storage secret must be in the same namespace as the TempoMonolithic instance and contain the fields specified in "Table 1. Required secret parameters" in the section "Object storage setup".
- 5 Enables the Jaeger UI.
- 6 Enables creation of a route for the Jaeger UI.

b. Apply the customized CR by running the following command:

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

Verification

1. Verify that the **status** of all TempoMonolithic **components** is **Running** and the **conditions** are **type: Ready** by running the following command:

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

2. Run the following command to verify that the pod of the TempoMonolithic instance is running:

```
$ oc get pods
```

3. Access the Jaeger UI:

- a. Query the route details for the **tempo-<metadata_name_of_tempomonolithic_cr>-jaegerui** route by running the following command:

```
$ oc get route
```

- b. Open **https://<route_from_previous_step>** in a web browser.

4. When the pod of the TempoMonolithic instance is ready, you can send traces to the **tempo-`<metadata_name_of_tempomonolithic_cr>:4317`** (OTLP/gRPC) and **tempo-`<metadata_name_of_tempomonolithic_cr>:4318`** (OTLP/HTTP) endpoints inside the cluster. The Tempo API is available at the **tempo-`<metadata_name_of_tempomonolithic_cr>:3200`** endpoint inside the cluster.

3.1.4. Object storage setup

You can use the following configuration parameters when setting up a supported object storage.

Table 3.1. Required secret parameters

Storage provider
Secret parameters
Red Hat OpenShift Data Foundation
name: <code>tempostack-dev-odf</code> # example bucket: <code><bucket_name></code> # requires an ObjectBucketClaim endpoint: <code>https://s3.openshift-storage.svc</code> access_key_id: <code><data_foundation_access_key_id></code> access_key_secret: <code><data_foundation_access_key_secret></code>
MinIO
See MinIO Operator .
name: <code>tempostack-dev-minio</code> # example bucket: <code><minio_bucket_name></code> # MinIO documentation endpoint: <code><minio_bucket_endpoint></code> access_key_id: <code><minio_access_key_id></code> access_key_secret: <code><minio_access_key_secret></code>
Amazon S3
name: <code>tempostack-dev-s3</code> # example bucket: <code><s3_bucket_name></code> # Amazon S3 documentation endpoint: <code><s3_bucket_endpoint></code> access_key_id: <code><s3_access_key_id></code> access_key_secret: <code><s3_access_key_secret></code>

Storage provider
<p>Microsoft Azure Blob Storage</p> <p>name: <code>tempostack-dev-azure</code> # example</p> <p>container: <code><azure_blob_storage_container_name></code> # Microsoft Azure documentation</p> <p>account_name: <code><azure_blob_storage_account_name></code></p> <p>account_key: <code><azure_blob_storage_account_key></code></p>
<p>Google Cloud Storage on Google Cloud Platform (GCP)</p> <p>name: <code>tempostack-dev-gcs</code> # example</p> <p>bucketname: <code><google_cloud_storage_bucket_name></code> # requires a bucket created in a GCP project</p> <p>key.json: <code><path/to/key.json></code> # requires a service account in the bucket's GCP project for GCP authentication</p>

3.1.5. Additional resources

- [Creating a cluster admin](#)
- [OperatorHub.io](#)
- [Accessing the web console](#)
- [Installing from OperatorHub using the web console](#)
- [Creating applications from installed Operators](#)
- [Getting started with the OpenShift CLI](#)

3.2. CONFIGURING

The Tempo Operator uses a custom resource definition (CRD) file that defines the architecture and configuration settings to be used when creating and deploying the distributed tracing platform (Tempo) resources. You can install the default configuration or modify the file.

3.2.1. Customizing your deployment

For information about configuring the back-end storage, see [Understanding persistent storage](#) and the appropriate configuration topic for your chosen storage option.

3.2.1.1. Default configuration options

The **TempoStack** custom resource (CR) defines the architecture and settings to be used when creating the distributed tracing platform (Tempo) resources. You can modify these parameters to customize your distributed tracing platform (Tempo) implementation to your business needs.

Example of a generic Tempo YAML file

```

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: {}

```

Table 3.2. Tempo parameters

Parameter	Description	Values	Default value
apiVersion:	API version to use when creating the object.	tempo.grafana.com/v1alpha1	tempo.grafana.com/v1alpha1
kind:	Defines the kind of Kubernetes object to create.	tempo	
metadata:	Data that uniquely identifies the object, including a name string, UID , and optional namespace .		OpenShift Container Platform automatically generates the UID and completes the namespace with the name of the project where the object is created.
name:	Name for the object.	Name of your TempoStack instance.	tempo-all-in-one-inmemory

Parameter	Description	Values	Default value
spec:	Specification for the object to be created.	Contains all of the configuration parameters for your TempoStack instance. When a common definition for all Tempo components is required, it is defined under the spec node. When the definition relates to an individual component, it is placed under the spec/template/<component> node.	N/A
resources:	Resources assigned to the TempoStack instance.		
storageSize:	Storage size for ingester PVCs.		
replicationFactor:	Configuration for the replication factor.		
retention:	Configuration options for retention of traces.		
storage:	Configuration options that define the storage. All storage-related options must be placed under storage and not under the allInOne or other component options.		
template.distributor:	Configuration options for the Tempo distributor .		
template.ingester:	Configuration options for the Tempo ingester .		
template.compactor:	Configuration options for the Tempo compactor .		

Parameter	Description	Values	Default value
template.querier:	Configuration options for the Tempo querier .		
template.queryFrontend:	Configuration options for the Tempo query-frontend .		
template.gateway:	Configuration options for the Tempo gateway .		

Minimum required configuration

The following is the required minimum for creating a distributed tracing platform (Tempo) deployment with the default settings:

```

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 This section specifies the deployed object storage back end, which requires a created secret with credentials for access to the object storage.

3.2.1.2. Storage configuration

You can configure object storage for the distributed tracing platform (Tempo) in the **TempoStack** custom resource under **spec.storage**. You can choose from among several storage providers that are supported.

Table 3.3. General storage parameters used by the Tempo Operator to define distributed tracing storage

Parameter	Description	Values	Default value
spec.storage.secret.type	Type of storage to use for the deployment.	memory . Memory storage is only appropriate for development, testing, demonstrations, and proof of concept environments because the data does not persist when the pod is shut down.	memory
storage.secretname	Name of the secret that contains the credentials for the set object storage type.		N/A
storage.tls.caName	CA is the name of a ConfigMap object containing a CA certificate.		

Table 3.4. Required secret parameters

Storage provider
Secret parameters
Red Hat OpenShift Data Foundation
<p>name: <code>tempostack-dev-odf # example</code></p> <p>bucket: <code><bucket_name> # requires an ObjectBucketClaim</code></p> <p>endpoint: <code>https://s3.openshift-storage.svc</code></p> <p>access_key_id: <code><data_foundation_access_key_id></code></p> <p>access_key_secret: <code><data_foundation_access_key_secret></code></p>
MinIO

Storage provider
<p>See MinIO Operator.</p> <p>name: <code>tempostack-dev-minio</code> # example</p> <p>bucket: <code><minio_bucket_name></code> # MinIO documentation</p> <p>endpoint: <code><minio_bucket_endpoint></code></p> <p>access_key_id: <code><minio_access_key_id></code></p> <p>access_key_secret: <code><minio_access_key_secret></code></p>
<p>Amazon S3</p> <p>name: <code>tempostack-dev-s3</code> # example</p> <p>bucket: <code><s3_bucket_name></code> # Amazon S3 documentation</p> <p>endpoint: <code><s3_bucket_endpoint></code></p> <p>access_key_id: <code><s3_access_key_id></code></p> <p>access_key_secret: <code><s3_access_key_secret></code></p>
<p>Microsoft Azure Blob Storage</p> <p>name: <code>tempostack-dev-azure</code> # example</p> <p>container: <code><azure_blob_storage_container_name></code> # Microsoft Azure documentation</p> <p>account_name: <code><azure_blob_storage_account_name></code></p> <p>account_key: <code><azure_blob_storage_account_key></code></p>
<p>Google Cloud Storage on Google Cloud Platform (GCP)</p> <p>name: <code>tempostack-dev-gcs</code> # example</p> <p>bucketname: <code><google_cloud_storage_bucket_name></code> # requires a bucket created in a GCP project</p> <p>key.json: <code><path/to/key.json></code> # requires a service account in the bucket's GCP project for GCP authentication</p>

3.2.1.3. Query configuration options

Two components of the distributed tracing platform (Tempo), the querier and query frontend, manage queries. You can configure both of these components.

The querier component finds the requested trace ID in the ingesters or back-end storage. Depending on the set parameters, the querier component can query both the ingesters and pull bloom or indexes from the back end to search blocks in object storage. The querier component exposes an HTTP

endpoint at **GET /querier/api/traces/<trace_id>**, but it is not expected to be used directly. Queries must be sent to the query frontend.

Table 3.5. Configuration parameters for the querier component

Parameter	Description	Values
nodeSelector	The simple form of the node-selection constraint.	type: object
replicas	The number of replicas to be created for the component.	type: integer; format: int32
tolerations	Component-specific pod tolerations.	type: array

The query frontend component is responsible for sharding the search space for an incoming query. The query frontend exposes traces via a simple HTTP endpoint: **GET /api/traces/<trace_id>**. Internally, the query frontend component splits the **blockID** space into a configurable number of shards and then queues these requests. The querier component connects to the query frontend component via a streaming gRPC connection to process these sharded queries.

Table 3.6. Configuration parameters for the query frontend component

Parameter	Description	Values
component	Configuration of the query frontend component.	type: object
component.nodeSelector	The simple form of the node selection constraint.	type: object
component.replicas	The number of replicas to be created for the query frontend component.	type: integer; format: int32
component.tolerations	Pod tolerations specific to the query frontend component.	type: array
jaegerQuery	The options specific to the Jaeger Query component.	type: object
jaegerQuery.enabled	When enabled , creates the Jaeger Query component, jaegerQuery .	type: boolean
jaegerQuery.ingress	The options for the Jaeger Query ingress.	type: object

Parameter	Description	Values
jaegerQuery.ingress.annotations	The annotations of the ingress object.	type: object
jaegerQuery.ingress.host	The hostname of the ingress object.	type: string
jaegerQuery.ingress.ingressClassName	The name of an IngressClass cluster resource. Defines which ingress controller serves this ingress resource.	type: string
jaegerQuery.ingress.route	The options for the OpenShift route.	type: object
jaegerQuery.ingress.route.termination	The termination type. The default is edge .	type: string (enum: insecure, edge, passthrough, reencrypt)
jaegerQuery.ingress.type	The type of ingress for the Jaeger Query UI. The supported types are ingress , route , and none .	type: string (enum: ingress, route)
jaegerQuery.monitorTab	The monitor tab configuration.	type: object
jaegerQuery.monitorTab.enabled	Enables the monitor tab in the Jaeger console. The PrometheusEndpoint must be configured.	type: boolean
jaegerQuery.monitorTab.prometheusEndpoint	The endpoint to the Prometheus instance that contains the span rate, error, and duration (RED) metrics. For example, https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 .	type: string

Example configuration of the query frontend component in a TempoStack CR

```

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. Additional resources

- [Understanding taints and tolerations](#)

3.2.1.4. Configuration of the monitor tab in Jaeger UI

Trace data contains rich information, and the data is normalized across instrumented languages and frameworks. Therefore, request rate, error, and duration (RED) metrics can be extracted from traces. The metrics can be visualized in Jaeger console in the **Monitor** tab.

The metrics are derived from spans in the OpenTelemetry Collector that are scraped from the Collector by the Prometheus deployed in the user-workload monitoring stack. The Jaeger UI queries these metrics from the Prometheus endpoint and visualizes them.

3.2.1.4.1. OpenTelemetry Collector configuration

The OpenTelemetry Collector requires configuration of the **spanmetrics** connector that derives metrics from traces and exports the metrics in the Prometheus format.

OpenTelemetry Collector custom resource for span RED

```

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 Creates the **ServiceMonitor** custom resource to enable scraping of the Prometheus exporter.
- 2 The Spanmetrics connector receives traces and exports metrics.
- 3 The OTLP receiver to receive spans in the OpenTelemetry protocol.
- 4 The Prometheus exporter is used to export metrics in the Prometheus format.
- 5 The Spanmetrics connector is configured as exporter in traces pipeline.
- 6 The Spanmetrics connector is configured as receiver in metrics pipeline.

3.2.1.4.2. Tempo configuration

The **TempoStack** custom resource must specify the following: the **Monitor** tab is enabled, and the Prometheus endpoint is set to the Thanos querier service to query the data from the user-defined monitoring stack.

TempoStack custom resource with the enabled Monitor tab

```

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
ingress:
  type: route

```

- 1 Enables the monitoring tab in the Jaeger console.
- 2 The service name for Thanos Querier from user-workload monitoring.

3.2.1.4.3. Span RED metrics and alerting rules

The metrics generated by the **spanmetrics** connector are usable with alerting rules. For example, for alerts about a slow service or to define service level objectives (SLOs), the connector creates a **duration_bucket** histogram and the **calls** counter metric. These metrics have labels that identify the service, API name, operation type, and other attributes.

Table 3.7. Labels of the metrics created in the **spanmetrics** connector

Label	Description	Values
service_name	Service name set by the otel_service_name environment variable.	frontend
span_name	Name of the operation.	<ul style="list-style-type: none"> • / • /customer
span_kind	Identifies the server, client, messaging, or internal operation.	<ul style="list-style-type: none"> • SPAN_KIND_SERVER • SPAN_KIND_CLIENT • SPAN_KIND_PRODUCER • SPAN_KIND_CONSUMER • SPAN_KIND_INTERNAL

Example **PrometheusRule** CR that defines an alerting rule for SLO when not serving 95% of requests within 2000ms on the front-end service

```

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** The expression for checking if 95% of the front-end server response time values are below 2000 ms. The time range (**[5m]**) must be at least four times the scrape interval and long enough to accommodate a change in the metric.

3.2.1.5. Multitenancy

Multitenancy with authentication and authorization is provided in the Tempo Gateway service. The authentication uses OpenShift OAuth and the Kubernetes **TokenReview** API. The authorization uses the Kubernetes **SubjectAccessReview** API.



NOTE

The Tempo Gateway service supports ingestion of traces only via the OTLP/gRPC. The OTLP/HTTP is not supported.

Sample Tempo CR with two tenants, dev and prod

```

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** Must be set to **openshift**.
- 2** The list of tenants.
- 3** The tenant name. Must be provided in the **X-Scope-OrgId** header when ingesting the data.

- 4 A unique tenant ID.
- 5 Enables a gateway that performs authentication and authorization. The Jaeger UI is exposed at <http://<gateway-ingress>/api/traces/v1/<tenant-name>/search>.

The authorization configuration uses the **ClusterRole** and **ClusterRoleBinding** of the Kubernetes Role-Based Access Control (RBAC). By default, no users have read or write permissions.

Sample of the read RBAC configuration that allows authenticated users to read the trace data of the dev and prod tenants

```

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 Lists the tenants.
- 2 The **get** value enables the read operation.
- 3 Grants all authenticated users the read permissions for trace data.

Sample of the write RBAC configuration that allows the otel-collector service account to write the trace data for the dev tenant

```

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
    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 The service account name for the client to use when exporting trace data. The client must send the service account token, `/var/run/secrets/kubernetes.io/serviceaccount/token`, as the bearer token header.
- ② Lists the tenants.
- ③ The **create** value enables the write operation.

Trace data can be sent to the Tempo instance from the OpenTelemetry Collector that uses the service account with RBAC for writing the data.

Sample OpenTelemetry CR configuration

```

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. Configuring monitoring and alerts

The Tempo Operator supports monitoring and alerts about each TempoStack component such as distributor, ingester, and so on, and exposes upgrade and operational metrics about the Operator itself.

3.2.2.1. Configuring the TempoStack metrics and alerts

You can enable metrics and alerts of TempoStack instances.

Prerequisites

- Monitoring for user-defined projects is enabled in the cluster.

Procedure

1. To enable metrics of a TempoStack instance, set the **spec.observability.metrics.createServiceMonitors** field to **true**:

```

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

```

2. To enable alerts for a TempoStack instance, set the **spec.observability.metrics.createPrometheusRules** field to **true**:

```

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

```

Verification

You can use the **Administrator** view of the web console to verify successful configuration:

1. Go to **Observe** → **Targets**, filter for **Source: User**, and check that **ServiceMonitors** in the format **tempo-<instance_name>-<component>** have the **Up** status.
2. To verify that alerts are set up correctly, go to **Observe** → **Alerting** → **Alerting rules**, filter for **Source: User**, and check that the **Alert rules** for the TempoStack instance components are available.

3.2.2.1.1. Additional resources

- [Enabling monitoring for user-defined projects](#)

3.2.2.2. Configuring the Tempo Operator metrics and alerts

When installing the Tempo Operator from the web console, you can select the **Enable Operator recommended cluster monitoring on this Namespace** checkbox, which enables creating metrics and alerts of the Tempo Operator.

If the checkbox was not selected during installation, you can manually enable metrics and alerts even after installing the Tempo Operator.

Procedure

- Add the **openshift.io/cluster-monitoring: "true"** label in the project where the Tempo Operator is installed, which is **openshift-tempo-operator** by default.

Verification

You can use the **Administrator** view of the web console to verify successful configuration:

1. Go to **Observe** → **Targets**, filter for **Source: Platform**, and search for **tempo-operator**, which must have the **Up** status.
2. To verify that alerts are set up correctly, go to **Observe** → **Alerting** → **Alerting rules**, filter for **Source: Platform**, and locate the **Alert rules** for the **Tempo Operator**.

3.3. UPGRADING

For version upgrades, the Tempo Operator uses the Operator Lifecycle Manager (OLM), which controls installation, upgrade, and role-based access control (RBAC) of Operators in a cluster.

The OLM runs in the OpenShift Container Platform by default. The OLM queries for available Operators as well as upgrades for installed Operators.

When the Tempo Operator is upgraded to the new version, it scans for running TempoStack instances that it manages and upgrades them to the version corresponding to the Operator's new version.

3.3.1. Additional resources

- [Operator Lifecycle Manager concepts and resources](#)
- [Updating installed Operators](#)

3.4. REMOVING

The steps for removing the Red Hat OpenShift distributed tracing platform (Tempo) from an OpenShift Container Platform cluster are as follows:

1. Shut down all distributed tracing platform (Tempo) pods.
2. Remove any TempoStack instances.
3. Remove the Tempo Operator.


3.4.1. Removing by using the web console

You can remove a TempoStack instance in the **Administrator** view of the web console.

Prerequisites

- You are logged in to the OpenShift Container Platform web console as a cluster administrator with the **cluster-admin** role.
- For Red Hat OpenShift Dedicated, you must be logged in using an account with the **dedicated-admin** role.

Procedure

1. Go to **Operators** → **Installed Operators** → **Tempo Operator** → **TempoStack**.
2. To remove the TempoStack instance, select  → **Delete TempoStack** → **Delete**.
3. Optional: Remove the Tempo Operator.

3.4.2. Removing by using the CLI

You can remove a TempoStack instance on the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

TIP

- Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
- Run **oc login**:

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

Procedure

1. Get the name of the TempoStack instance by running the following command:

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

2. Remove the TempoStack instance by running the following command:

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

3. Optional: Remove the Tempo Operator.

Verification

1. Run the following command to verify that the TempoStack instance is not found in the output, which indicates its successful removal:

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

3.4.3. Additional resources

- [Deleting Operators from a cluster](#)
- [Getting started with the OpenShift CLI](#)

CHAPTER 4. DISTRIBUTED TRACING PLATFORM (JAEGER)

4.1. INSTALLING



IMPORTANT

The Red Hat OpenShift distributed tracing platform (Jaeger) is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

You can install Red Hat OpenShift distributed tracing platform on OpenShift Container Platform in either of two ways:

- You can install Red Hat OpenShift distributed tracing platform as part of Red Hat OpenShift Service Mesh. Distributed tracing is included by default in the Service Mesh installation. To install Red Hat OpenShift distributed tracing platform as part of a service mesh, follow the [Red Hat Service Mesh Installation](#) instructions. You must install Red Hat OpenShift distributed tracing platform in the same namespace as your service mesh, that is, the **ServiceMeshControlPlane** and the Red Hat OpenShift distributed tracing platform resources must be in the same namespace.
- If you do not want to install a service mesh, you can use the Red Hat OpenShift distributed tracing platform Operators to install distributed tracing platform by itself. To install Red Hat OpenShift distributed tracing platform without a service mesh, use the following instructions.

4.1.1. Prerequisites

Before you can install Red Hat OpenShift distributed tracing platform, review the installation activities, and ensure that you meet the prerequisites:

- Possess an active OpenShift Container Platform subscription on your Red Hat account. If you do not have a subscription, contact your sales representative for more information.
- Review the [OpenShift Container Platform 4.16 overview](#).
- Install OpenShift Container Platform 4.16.
 - [Install OpenShift Container Platform 4.16 on AWS](#)
 - [Install OpenShift Container Platform 4.16 on user-provisioned AWS](#)
 - [Install OpenShift Container Platform 4.16 on bare metal](#)
 - [Install OpenShift Container Platform 4.16 on vSphere](#)
- Install the version of the **oc** CLI tool that matches your OpenShift Container Platform version and add it to your path.
- An account with the **cluster-admin** role.

4.1.2. Red Hat OpenShift distributed tracing platform installation overview

The steps for installing Red Hat OpenShift distributed tracing platform are as follows:

- Review the documentation and determine your deployment strategy.
- If your deployment strategy requires persistent storage, install the OpenShift Elasticsearch Operator via the OperatorHub.
- Install the Red Hat OpenShift distributed tracing platform (Jaeger) Operator via the OperatorHub.
- Modify the custom resource YAML file to support your deployment strategy.
- Deploy one or more instances of Red Hat OpenShift distributed tracing platform (Jaeger) to your OpenShift Container Platform environment.

4.1.3. Installing the OpenShift Elasticsearch Operator

The default Red Hat OpenShift distributed tracing platform (Jaeger) deployment uses in-memory storage because it is designed to be installed quickly for those evaluating Red Hat OpenShift distributed tracing platform, giving demonstrations, or using Red Hat OpenShift distributed tracing platform (Jaeger) in a test environment. If you plan to use Red Hat OpenShift distributed tracing platform (Jaeger) in production, you must install and configure a persistent storage option, in this case, Elasticsearch.

Prerequisites

- You have access to the OpenShift Container Platform web console.
- You have access to the cluster as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.



WARNING

Do not install Community versions of the Operators. Community Operators are not supported.



NOTE

If you have already installed the OpenShift Elasticsearch Operator as part of OpenShift Logging, you do not need to install the OpenShift Elasticsearch Operator again. The Red Hat OpenShift distributed tracing platform (Jaeger) Operator creates the Elasticsearch instance using the installed OpenShift Elasticsearch Operator.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

2. Navigate to **Operators** → **OperatorHub**.
3. Type **Elasticsearch** into the filter box to locate the OpenShift Elasticsearch Operator.
4. Click the **OpenShift Elasticsearch Operator** provided by Red Hat to display information about the Operator.
5. Click **Install**.
6. On the **Install Operator** page, select the **stable** Update Channel. This automatically updates your Operator as new versions are released.
7. Accept the default **All namespaces on the cluster (default)** This installs the Operator in the default **openshift-operators-redhat** project and makes the Operator available to all projects in the cluster.

**NOTE**

The Elasticsearch installation requires the **openshift-operators-redhat** namespace for the OpenShift Elasticsearch Operator. The other Red Hat OpenShift distributed tracing platform Operators are installed in the **openshift-operators** namespace.

8. Accept the default **Automatic** approval strategy. By accepting the default, when a new version of this Operator is available, Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention. If you select **Manual** updates, when a newer version of an Operator is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

**NOTE**

The **Manual** approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

9. Click **Install**.
10. On the **Installed Operators** page, select the **openshift-operators-redhat** project. Wait for the **InstallSucceeded** status of the OpenShift Elasticsearch Operator before continuing.

4.1.4. Installing the Red Hat OpenShift distributed tracing platform Operator

You can install the Red Hat OpenShift distributed tracing platform Operator through the [OperatorHub](#).

By default, the Operator is installed in the **openshift-operators** project.

Prerequisites

- You have access to the OpenShift Container Platform web console.
- You have access to the cluster as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.
- If you require persistent storage, you must install the OpenShift Elasticsearch Operator before installing the Red Hat OpenShift distributed tracing platform Operator.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.
2. Navigate to **Operators** → **OperatorHub**.
3. Search for the Red Hat OpenShift distributed tracing platform Operator by entering **distributed tracing platform** in the search field.
4. Select the **Red Hat OpenShift distributed tracing platform** Operator, which is **provided by Red Hat**, to display information about the Operator.
5. Click **Install**.
6. For the **Update channel** on the **Install Operator** page, select **stable** to automatically update the Operator when new versions are released.
7. Accept the default **All namespaces on the cluster (default)**. This installs the Operator in the default **openshift-operators** project and makes the Operator available to all projects in the cluster.
8. Accept the default **Automatic** approval strategy.



NOTE

If you accept this default, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of this Operator when a new version of the Operator becomes available.

If you select **Manual** updates, the OLM creates an update request when a new version of the Operator becomes available. To update the Operator to the new version, you must then manually approve the update request as a cluster administrator. The **Manual** approval strategy requires a cluster administrator to manually approve Operator installation and subscription.

9. Click **Install**.
10. Navigate to **Operators** → **Installed Operators**.
11. On the **Installed Operators** page, select the **openshift-operators** project. Wait for the **Succeeded** status of the Red Hat OpenShift distributed tracing platform Operator before continuing.

4.2. CONFIGURING



IMPORTANT

The Red Hat OpenShift distributed tracing platform (Jaeger) is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

The Red Hat OpenShift distributed tracing platform (Jaeger) Operator uses a custom resource definition (CRD) file that defines the architecture and configuration settings to be used when creating and deploying the distributed tracing platform (Jaeger) resources. You can install the default configuration or modify the file.

If you have installed distributed tracing platform as part of Red Hat OpenShift Service Mesh, you can perform basic configuration as part of the [ServiceMeshControlPlane](#), but for complete control, you must configure a Jaeger CR and then [reference your distributed tracing configuration file in the ServiceMeshControlPlane](#).

The Red Hat OpenShift distributed tracing platform (Jaeger) has predefined deployment strategies. You specify a deployment strategy in the custom resource file. When you create a distributed tracing platform (Jaeger) instance, the Operator uses this configuration file to create the objects necessary for the deployment.

Jaeger custom resource file showing deployment strategy

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: MyConfigFile
spec:
  strategy: production 1
```

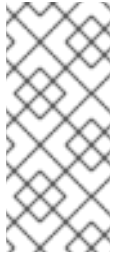
1 Deployment strategy.

4.2.1. Supported deployment strategies

The Red Hat OpenShift distributed tracing platform (Jaeger) Operator currently supports the following deployment strategies:

allInOne

- This strategy is intended for development, testing, and demo purposes; it is not intended for production use. The main backend components, Agent, Collector, and Query service, are all packaged into a single executable which is configured, by default, to use in-memory storage.

**NOTE**

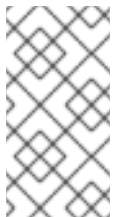
In-memory storage is not persistent, which means that if the distributed tracing platform (Jaeger) instance shuts down, restarts, or is replaced, that your trace data will be lost. And in-memory storage cannot be scaled, since each pod has its own memory. For persistent storage, you must use the **production** or **streaming** strategies, which use Elasticsearch as the default storage.

production

The production strategy is intended for production environments, where long term storage of trace data is important, as well as a more scalable and highly available architecture is required. Each of the backend components is therefore deployed separately. The Agent can be injected as a sidecar on the instrumented application. The Query and Collector services are configured with a supported storage type - currently Elasticsearch. Multiple instances of each of these components can be provisioned as required for performance and resilience purposes.

streaming

The streaming strategy is designed to augment the production strategy by providing a streaming capability that effectively sits between the Collector and the Elasticsearch backend storage. This provides the benefit of reducing the pressure on the backend storage, under high load situations, and enables other trace post-processing capabilities to tap into the real time span data directly from the streaming platform ([AMQ Streams](#)/[Kafka](#)).

**NOTE**

- The streaming strategy requires an additional Red Hat subscription for AMQ Streams.
- The streaming deployment strategy is currently unsupported on IBM Z®.

4.2.2. Deploying the distributed tracing platform default strategy from the web console

The custom resource definition (CRD) defines the configuration used when you deploy an instance of Red Hat OpenShift distributed tracing platform. The default CR is named **jaeger-all-in-one-inmemory** and it is configured with minimal resources to ensure that you can successfully install it on a default OpenShift Container Platform installation. You can use this default configuration to create a Red Hat OpenShift distributed tracing platform (Jaeger) instance that uses the **AllInOne** deployment strategy, or you can define your own custom resource file.

**NOTE**

In-memory storage is not persistent. If the Jaeger pod shuts down, restarts, or is replaced, your trace data will be lost. For persistent storage, you must use the **production** or **streaming** strategies, which use Elasticsearch as the default storage.

Prerequisites

- The Red Hat OpenShift distributed tracing platform (Jaeger) Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Create a new project, for example **tracing-system**.



NOTE

If you are installing as part of Service Mesh, the distributed tracing platform resources must be installed in the same namespace as the **ServiceMeshControlPlane** resource, for example **istio-system**.

- a. Go to **Home** → **Projects**.
 - b. Click **Create Project**.
 - c. Enter **tracing-system** in the **Name** field.
 - d. Click **Create**.
3. Navigate to **Operators** → **Installed Operators**.
 4. If necessary, select **tracing-system** from the **Project** menu. You may have to wait a few moments for the Operators to be copied to the new project.
 5. Click the Red Hat OpenShift distributed tracing platform (Jaeger) Operator. On the **Details** tab, under **Provided APIs**, the Operator provides a single link.
 6. Under **Jaeger**, click **Create Instance**.
 7. On the **Create Jaeger** page, to install using the defaults, click **Create** to create the distributed tracing platform (Jaeger) instance.
 8. On the **Jaegers** page, click the name of the distributed tracing platform (Jaeger) instance, for example, **jaeger-all-in-one-inmemory**.
 9. On the **Jaeger Details** page, click the **Resources** tab. Wait until the pod has a status of "Running" before continuing.

4.2.2.1. Deploying the distributed tracing platform default strategy from the CLI

Follow this procedure to create an instance of distributed tracing platform (Jaeger) from the command line.

Prerequisites

- The Red Hat OpenShift distributed tracing platform (Jaeger) Operator has been installed and verified.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role by running the following command:

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

2. Create a new project named **tracing-system** by running the following command:

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

3. Create a custom resource file named **jaeger.yaml** that contains the following text:

Example jaeger-all-in-one.yaml

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

4. Run the following command to deploy distributed tracing platform (Jaeger):

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

5. Run the following command to watch the progress of the pods during the installation process:

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

After the installation process has completed, the output is similar to the following example:

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

4.2.3. Deploying the distributed tracing platform production strategy from the web console

The **production** deployment strategy is intended for production environments that require a more scalable and highly available architecture, and where long-term storage of trace data is important.

Prerequisites

- The OpenShift Elasticsearch Operator has been installed.
- The Red Hat OpenShift distributed tracing platform (Jaeger) Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.

2. Create a new project, for example **tracing-system**.



NOTE

If you are installing as part of Service Mesh, the distributed tracing platform resources must be installed in the same namespace as the **ServiceMeshControlPlane** resource, for example **istio-system**.

- a. Navigate to **Home** → **Projects**.
 - b. Click **Create Project**.
 - c. Enter **tracing-system** in the **Name** field.
 - d. Click **Create**.
3. Navigate to **Operators** → **Installed Operators**.
 4. If necessary, select **tracing-system** from the **Project** menu. You may have to wait a few moments for the Operators to be copied to the new project.
 5. Click the Red Hat OpenShift distributed tracing platform (Jaeger) Operator. On the **Overview** tab, under **Provided APIs**, the Operator provides a single link.
 6. Under **Jaeger**, click **Create Instance**.
 7. On the **Create Jaeger** page, replace the default **all-in-one** YAML text with your production YAML configuration, for example:

Example jaeger-production.yaml file with Elasticsearch

```
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. Click **Create** to create the distributed tracing platform (Jaeger) instance.

9. On the **Jaegers** page, click the name of the distributed tracing platform (Jaeger) instance, for example, **jaeger-prod-elasticsearch**.
10. On the **Jaeger Details** page, click the **Resources** tab. Wait until all the pods have a status of "Running" before continuing.

4.2.3.1. Deploying the distributed tracing platform production strategy from the CLI

Follow this procedure to create an instance of distributed tracing platform (Jaeger) from the command line.

Prerequisites

- The OpenShift Elasticsearch Operator has been installed.
- The Red Hat OpenShift distributed tracing platform (Jaeger) Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift CLI (**oc**) as a user with the **cluster-admin** role by running the following command:

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

2. Create a new project named **tracing-system** by running the following command:

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

3. Create a custom resource file named **jaeger-production.yaml** that contains the text of the example file in the previous procedure.

4. Run the following command to deploy distributed tracing platform (Jaeger):

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

5. Run the following command to watch the progress of the pods during the installation process:

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

After the installation process has completed, you will see output similar to the following example:

```
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
jaeger-production-query-5cbfbd499d-tv8zf	3/3	Running	3
			8m32s
			8m32s

4.2.4. Deploying the distributed tracing platform streaming strategy from the web console

The **streaming** deployment strategy is intended for production environments that require a more scalable and highly available architecture, and where long-term storage of trace data is important.

The **streaming** strategy provides a streaming capability that sits between the Collector and the Elasticsearch storage. This reduces the pressure on the storage under high load situations, and enables other trace post-processing capabilities to tap into the real-time span data directly from the Kafka streaming platform.



NOTE

The streaming strategy requires an additional Red Hat subscription for AMQ Streams. If you do not have an AMQ Streams subscription, contact your sales representative for more information.



NOTE

The streaming deployment strategy is currently unsupported on IBM Z®.

Prerequisites

- The AMQ Streams Operator has been installed. If using version 1.4.0 or higher you can use self-provisioning. Otherwise you must create the Kafka instance.
- The Red Hat OpenShift distributed tracing platform (Jaeger) Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Create a new project, for example **tracing-system**.



NOTE

If you are installing as part of Service Mesh, the distributed tracing platform resources must be installed in the same namespace as the **ServiceMeshControlPlane** resource, for example **istio-system**.

- a. Navigate to **Home** → **Projects**.
- b. Click **Create Project**.
- c. Enter **tracing-system** in the **Name** field.

- d. Click **Create**.
3. Navigate to **Operators → Installed Operators**.
4. If necessary, select **tracing-system** from the **Project** menu. You may have to wait a few moments for the Operators to be copied to the new project.
5. Click the Red Hat OpenShift distributed tracing platform (Jaeger) Operator. On the **Overview** tab, under **Provided APIs**, the Operator provides a single link.
6. Under **Jaeger**, click **Create Instance**.
7. On the **Create Jaeger** page, replace the default **all-in-one** YAML text with your streaming YAML configuration, for example:

Example jaeger-streaming.yaml file

```

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 If the brokers are not defined, AMQStreams 1.4.0+ self-provisions Kafka.

8. Click **Create** to create the distributed tracing platform (Jaeger) instance.
9. On the **Jaegers** page, click the name of the distributed tracing platform (Jaeger) instance, for example, **jaeger-streaming**.
10. On the **Jaeger Details** page, click the **Resources** tab. Wait until all the pods have a status of "Running" before continuing.

4.2.4.1. Deploying the distributed tracing platform streaming strategy from the CLI

Follow this procedure to create an instance of distributed tracing platform (Jaeger) from the command line.

Prerequisites

- The AMQ Streams Operator has been installed. If using version 1.4.0 or higher you can use self-provisioning. Otherwise you must create the Kafka instance.
- The Red Hat OpenShift distributed tracing platform (Jaeger) Operator has been installed.
- You have reviewed the instructions for how to customize the deployment.
- You have access to the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift CLI (**oc**) as a user with the **cluster-admin** role by running the following command:

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

2. Create a new project named **tracing-system** by running the following command:

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

3. Create a custom resource file named **jaeger-streaming.yaml** that contains the text of the example file in the previous procedure.

4. Run the following command to deploy Jaeger:

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

5. Run the following command to watch the progress of the pods during the installation process:

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

After the installation process has completed, you should see output similar to the following example:

```
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. Validating your deployment

4.2.5.1. Accessing the Jaeger console

To access the Jaeger console you must have either Red Hat OpenShift Service Mesh or Red Hat OpenShift distributed tracing platform installed, and Red Hat OpenShift distributed tracing platform (Jaeger) installed, configured, and deployed.

The installation process creates a route to access the Jaeger console.

If you know the URL for the Jaeger console, you can access it directly. If you do not know the URL, use the following directions.

Procedure from the web console

1. Log in to the OpenShift Container Platform web console as a user with cluster-admin rights. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.
2. Navigate to **Networking** → **Routes**.
3. On the **Routes** page, select the control plane project, for example **tracing-system**, from the **Namespace** menu.
The **Location** column displays the linked address for each route.
4. If necessary, use the filter to find the **jaeger** route. Click the route **Location** to launch the console.
5. Click **Log In With OpenShift**

Procedure from the CLI

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role by running the following command. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

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

2. To query for details of the route using the command line, enter the following command. In this example, **tracing-system** is the control plane namespace.

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

3. Launch a browser and navigate to **https://<JAEGER_URL>**, where **<JAEGER_URL>** is the route that you discovered in the previous step.
4. Log in using the same user name and password that you use to access the OpenShift Container Platform console.
5. If you have added services to the service mesh and have generated traces, you can use the filters and **Find Traces** button to search your trace data.
If you are validating the console installation, there is no trace data to display.

4.2.6. Customizing your deployment

4.2.6.1. Deployment best practices

- Red Hat OpenShift distributed tracing platform instance names must be unique. If you want to have multiple Red Hat OpenShift distributed tracing platform (Jaeger) instances and are using sidecar injected agents, then the Red Hat OpenShift distributed tracing platform (Jaeger) instances should have unique names, and the injection annotation should explicitly specify the Red Hat OpenShift distributed tracing platform (Jaeger) instance name the tracing data should be reported to.
- If you have a multitenant implementation and tenants are separated by namespaces, deploy a Red Hat OpenShift distributed tracing platform (Jaeger) instance to each tenant namespace.

For information about configuring persistent storage, see [Understanding persistent storage](#) and the appropriate configuration topic for your chosen storage option.

4.2.6.2. Distributed tracing default configuration options

The Jaeger custom resource (CR) defines the architecture and settings to be used when creating the distributed tracing platform (Jaeger) resources. You can modify these parameters to customize your distributed tracing platform (Jaeger) implementation to your business needs.

Generic YAML example of the Jaeger CR

```
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: {}
```

Table 4.1. Jaeger parameters

Parameter	Description	Values	Default value
-----------	-------------	--------	---------------

Parameter	Description	Values	Default value
apiVersion:	API version to use when creating the object.	jaegertracing.io/v1	jaegertracing.io/v1
kind:	Defines the kind of Kubernetes object to create.	jaeger	
metadata:	Data that helps uniquely identify the object, including a name string, UID , and optional namespace .		OpenShift Container Platform automatically generates the UID and completes the namespace with the name of the project where the object is created.
name:	Name for the object.	The name of your distributed tracing platform (Jaeger) instance.	jaeger-all-in-one-inmemory
spec:	Specification for the object to be created.	Contains all of the configuration parameters for your distributed tracing platform (Jaeger) instance. When a common definition for all Jaeger components is required, it is defined under the spec node. When the definition relates to an individual component, it is placed under the spec/<component> node.	N/A
strategy:	Jaeger deployment strategy	allInOne , production , or streaming	allInOne

Parameter	Description	Values	Default value
allInOne:	Because the allInOne image deploys the Agent, Collector, Query, Ingester, and Jaeger UI in a single pod, configuration for this deployment must nest component configuration under the allInOne parameter.		
agent:	Configuration options that define the Agent.		
collector:	Configuration options that define the Jaeger Collector.		
sampling:	Configuration options that define the sampling strategies for tracing.		
storage:	Configuration options that define the storage. All storage-related options must be placed under storage , rather than under the allInOne or other component options.		
query:	Configuration options that define the Query service.		
ingester:	Configuration options that define the Ingester service.		

The following example YAML is the minimum required to create a Red Hat OpenShift distributed tracing platform (Jaeger) deployment using the default settings.

Example minimum required dist-tracing-all-in-one.yaml

```

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

```

4.2.6.3. Jaeger Collector configuration options

The Jaeger Collector is the component responsible for receiving the spans that were captured by the tracer and writing them to persistent Elasticsearch storage when using the **production** strategy, or to AMQ Streams when using the **streaming** strategy.

The Collectors are stateless and thus many instances of Jaeger Collector can be run in parallel. Collectors require almost no configuration, except for the location of the Elasticsearch cluster.

Table 4.2. Parameters used by the Operator to define the Jaeger Collector

Parameter	Description	Values
collector: replicas:	Specifies the number of Collector replicas to create.	Integer, for example, 5

Table 4.3. Configuration parameters passed to the Collector

Parameter	Description	Values
spec: collector: options: {}	Configuration options that define the Jaeger Collector.	
options: collector: num-workers:	The number of workers pulling from the queue.	Integer, for example, 50
options: collector: queue-size:	The size of the Collector queue.	Integer, for example, 2000
options: kafka: producer: topic: jaeger-spans	The topic parameter identifies the Kafka configuration used by the Collector to produce the messages, and the Ingestor to consume the messages.	Label for the producer.
options: kafka: producer: brokers: my-cluster-kafka-brokers.kafka:9092	Identifies the Kafka configuration used by the Collector to produce the messages. If brokers are not specified, and you have AMQ Streams 1.4.0+ installed, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator will self-provision Kafka.	

Parameter	Description	Values
<pre>options: log-level:</pre>	Logging level for the Collector.	Possible values: debug, info, warn, error, fatal, panic.
<pre>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</pre>	To accept OTLP/gRPC, explicitly enable the otlp . All the other options are optional.	

Parameter	Description	Values
<pre> 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_SHA 384,TLS_CHACHA20_POL Y1305_SHA256" client-ca: /path/to/cert.ca reload-interval: 0s min-version: 1.2 max-version: 1.3 </pre>	To accept OTLP/HTTP, explicitly enable the otlp . All the other options are optional.	

4.2.6.4. Distributed tracing sampling configuration options

The Red Hat OpenShift distributed tracing platform (Jaeger) Operator can be used to define sampling strategies that will be supplied to tracers that have been configured to use a remote sampler.

While all traces are generated, only a few are sampled. Sampling a trace marks the trace for further processing and storage.



NOTE

This is not relevant if a trace was started by the Envoy proxy, as the sampling decision is made there. The Jaeger sampling decision is only relevant when the trace is started by an application using the client.

When a service receives a request that contains no trace context, the client starts a new trace, assigns it a random trace ID, and makes a sampling decision based on the currently installed sampling strategy. The sampling decision propagates to all subsequent requests in the trace so that other services are not making the sampling decision again.

distributed tracing platform (Jaeger) libraries support the following samplers:

- **Probabilistic** - The sampler makes a random sampling decision with the probability of sampling equal to the value of the **sampling.param** property. For example, using **sampling.param=0.1** samples approximately 1 in 10 traces.
- **Rate Limiting** - The sampler uses a leaky bucket rate limiter to ensure that traces are sampled with a certain constant rate. For example, using **sampling.param=2.0** samples requests with the rate of 2 traces per second.

Table 4.4. Jaeger sampling options

Parameter	Description	Values	Default value
<pre>spec: sampling: options: {} default_strategy: service_strategy:</pre>	Configuration options that define the sampling strategies for tracing.		If you do not provide configuration, the Collectors will return the default probabilistic sampling policy with 0.001 (0.1%) probability for all services.
<pre>default_strategy: type: service_strategy: type:</pre>	Sampling strategy to use. See descriptions above.	Valid values are probabilistic , and ratelimiting .	probabilistic
<pre>default_strategy: param: service_strategy: param:</pre>	Parameters for the selected sampling strategy.	Decimal and integer values (0, .1, 1, 10)	1

This example defines a default sampling strategy that is probabilistic, with a 50% chance of the trace instances being sampled.

Probabilistic sampling example

```
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

```

If there are no user-supplied configurations, the distributed tracing platform (Jaeger) uses the following settings:

Default sampling

```

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

```

4.2.6.5. Distributed tracing storage configuration options

You configure storage for the Collector, Ingester, and Query services under **spec.storage**. Multiple instances of each of these components can be provisioned as required for performance and resilience purposes.

Table 4.5. General storage parameters used by the Red Hat OpenShift distributed tracing platform (Jaeger) Operator to define distributed tracing storage

Parameter	Description	Values	Default value
spec: storage: type:	Type of storage to use for the deployment.	memory or elasticsearch . Memory storage is only appropriate for development, testing, demonstrations, and proof of concept environments as the data does not persist if the pod is shut down. For production environments distributed tracing platform (Jaeger) supports Elasticsearch for persistent storage.	memory

Parameter	Description	Values	Default value
<code>storage: secretname:</code>	Name of the secret, for example tracing-secret .		N/A
<code>storage: options: {}</code>	Configuration options that define the storage.		

Table 4.6. Elasticsearch index cleaner parameters

Parameter	Description	Values	Default value
<code>storage: esIndexCleaner: enabled:</code>	When using Elasticsearch storage, by default a job is created to clean old traces from the index. This parameter enables or disables the index cleaner job.	true/ false	true
<code>storage: esIndexCleaner: numberOfDays:</code>	Number of days to wait before deleting an index.	Integer value	7
<code>storage: esIndexCleaner: schedule:</code>	Defines the schedule for how often to clean the Elasticsearch index.	Cron expression	"55 23 * * *"

4.2.6.5.1. Auto-provisioning an Elasticsearch instance

When you deploy a Jaeger custom resource, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator uses the OpenShift Elasticsearch Operator to create an Elasticsearch cluster based on the configuration provided in the **storage** section of the custom resource file. The Red Hat OpenShift distributed tracing platform (Jaeger) Operator will provision Elasticsearch if the following configurations are set:

- **spec.storage:type** is set to **elasticsearch**
- **spec.storage.elasticsearch.doNotProvision** set to **false**
- **spec.storage.options.es.server-urls** is not defined, that is, there is no connection to an Elasticsearch instance that was not provisioned by the OpenShift Elasticsearch Operator.

When provisioning Elasticsearch, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator sets the Elasticsearch custom resource **name** to the value of **spec.storage.elasticsearch.name** from the Jaeger custom resource. If you do not specify a value for **spec.storage.elasticsearch.name**, the Operator uses **elasticsearch**.

Restrictions

- You can have only one distributed tracing platform (Jaeger) with self-provisioned Elasticsearch instance per namespace. The Elasticsearch cluster is meant to be dedicated for a single distributed tracing platform (Jaeger) instance.
- There can be only one Elasticsearch per namespace.



NOTE

If you already have installed Elasticsearch as part of OpenShift Logging, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator can use the installed OpenShift Elasticsearch Operator to provision storage.

The following configuration parameters are for a *self-provisioned* Elasticsearch instance, that is an instance created by the Red Hat OpenShift distributed tracing platform (Jaeger) Operator using the OpenShift Elasticsearch Operator. You specify configuration options for self-provisioned Elasticsearch under **spec:storage:elasticsearch** in your configuration file.

Table 4.7. Elasticsearch resource configuration parameters

Parameter	Description	Values	Default value
<code>elasticsearch: properties: doNotProvision:</code>	Use to specify whether or not an Elasticsearch instance should be provisioned by the Red Hat OpenShift distributed tracing platform (Jaeger) Operator.	true/false	true
<code>elasticsearch: properties: name:</code>	Name of the Elasticsearch instance. The Red Hat OpenShift distributed tracing platform (Jaeger) Operator uses the Elasticsearch instance specified in this parameter to connect to Elasticsearch.	string	elasticsearch

Parameter	Description	Values	Default value
<code>elasticsearch:nodeCount:</code>	Number of Elasticsearch nodes. For high availability use at least 3 nodes. Do not use 2 nodes as "split brain" problem can happen.	Integer value. For example, Proof of concept = 1, Minimum deployment =3	3
<code>elasticsearch:resources:requests:cpu:</code>	Number of central processing units for requests, based on your environment's configuration.	Specified in cores or millicores, for example, 200m, 0.5, 1. For example, Proof of concept = 500m, Minimum deployment =1	1
<code>elasticsearch:resources:requests:memory:</code>	Available memory for requests, based on your environment's configuration.	Specified in bytes, for example, 200Ki, 50Mi, 5Gi. For example, Proof of concept = 1Gi, Minimum deployment = 16Gi*	16Gi
<code>elasticsearch:resources:limits:cpu:</code>	Limit on number of central processing units, based on your environment's configuration.	Specified in cores or millicores, for example, 200m, 0.5, 1. For example, Proof of concept = 500m, Minimum deployment =1	
<code>elasticsearch:resources:limits:memory:</code>	Available memory limit based on your environment's configuration.	Specified in bytes, for example, 200Ki, 50Mi, 5Gi. For example, Proof of concept = 1Gi, Minimum deployment = 16Gi*	
<code>elasticsearch:redundancyPolicy:</code>	Data replication policy defines how Elasticsearch shards are replicated across data nodes in the cluster. If not specified, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator automatically determines the most appropriate replication based on number of nodes.	ZeroRedundancy (no replica shards), SingleRedundancy (one replica shard), MultipleRedundancy (each index is spread over half of the Data nodes), FullRedundancy (each index is fully replicated on every Data node in the cluster).	

Parameter	Description	Values	Default value
elasticsearch: useCertManagement:	Use to specify whether or not distributed tracing platform (Jaeger) should use the certificate management feature of the OpenShift Elasticsearch Operator. This feature was added to <code>{logging-title} 5.2</code> in OpenShift Container Platform 4.7 and is the preferred setting for new Jaeger deployments.	true/false	true

Each Elasticsearch node can operate with a lower memory setting though this is NOT recommended for production deployments. For production use, you must have no less than 16 Gi allocated to each pod by default, but preferably allocate as much as you can, up to 64 Gi per pod.

Production storage example

```

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

```

Storage example with persistent storage:

```

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 Persistent storage configuration. In this case AWS **gp2** with **5Gi** size. When no value is specified, distributed tracing platform (Jaeger) uses **emptyDir**. The OpenShift Elasticsearch Operator provisions **PersistentVolumeClaim** and **PersistentVolume** which are not removed with distributed tracing platform (Jaeger) instance. You can mount the same volumes if you create a distributed tracing platform (Jaeger) instance with the same name and namespace.

4.2.6.5.2. Connecting to an existing Elasticsearch instance

You can use an existing Elasticsearch cluster for storage with distributed tracing platform. An existing Elasticsearch cluster, also known as an *external* Elasticsearch instance, is an instance that was not installed by the Red Hat OpenShift distributed tracing platform (Jaeger) Operator or by the OpenShift Elasticsearch Operator.

When you deploy a Jaeger custom resource, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator will not provision Elasticsearch if the following configurations are set:

- **spec.storage.elasticsearch.doNotProvision** set to **true**
- **spec.storage.options.es.server-urls** has a value
- **spec.storage.elasticsearch.name** has a value, or if the Elasticsearch instance name is **elasticsearch**.

The Red Hat OpenShift distributed tracing platform (Jaeger) Operator uses the Elasticsearch instance specified in **spec.storage.elasticsearch.name** to connect to Elasticsearch.

Restrictions

- You cannot share or reuse a OpenShift Container Platform logging Elasticsearch instance with distributed tracing platform (Jaeger). The Elasticsearch cluster is meant to be dedicated for a single distributed tracing platform (Jaeger) instance.

The following configuration parameters are for an already existing Elasticsearch instance, also known as an *external* Elasticsearch instance. In this case, you specify configuration options for Elasticsearch under **spec:storage:options:es** in your custom resource file.

Table 4.8. General ES configuration parameters

Parameter	Description	Values	Default value
es: server-urls:	URL of the Elasticsearch instance.	The fully-qualified domain name of the Elasticsearch server.	<a href="http://elasticsearch.<namespace>.svc:9200">http://elasticsearch.<namespace>.svc:9200

Parameter	Description	Values	Default value
<code>es: max-doc-count:</code>	The maximum document count to return from an Elasticsearch query. This will also apply to aggregations. If you set both es.max-doc-count and es.max-num-spans , Elasticsearch will use the smaller value of the two.		10000
<code>es: max-num-spans:</code>	[Deprecated - Will be removed in a future release, use es.max-doc-count instead.] The maximum number of spans to fetch at a time, per query, in Elasticsearch. If you set both es.max-num-spans and es.max-doc-count , Elasticsearch will use the smaller value of the two.		10000
<code>es: max-span-age:</code>	The maximum lookback for spans in Elasticsearch.		72h0m0s
<code>es: sniffer:</code>	The sniffer configuration for Elasticsearch. The client uses the sniffing process to find all nodes automatically. Disabled by default.	true/ false	false
<code>es: sniffer-tls-enabled:</code>	Option to enable TLS when sniffing an Elasticsearch Cluster. The client uses the sniffing process to find all nodes automatically. Disabled by default	true/ false	false
<code>es: timeout:</code>	Timeout used for queries. When set to zero there is no timeout.		0s

Parameter	Description	Values	Default value
<code>es:username:</code>	The username required by Elasticsearch. The basic authentication also loads CA if it is specified. See also es.password .		
<code>es:password:</code>	The password required by Elasticsearch. See also, es.username .		
<code>es:version:</code>	The major Elasticsearch version. If not specified, the value will be auto-detected from Elasticsearch.		0

Table 4.9. ES data replication parameters

Parameter	Description	Values	Default value
<code>es:num-replicas:</code>	The number of replicas per index in Elasticsearch.		1
<code>es:num-shards:</code>	The number of shards per index in Elasticsearch.		5

Table 4.10. ES index configuration parameters

Parameter	Description	Values	Default value
<code>es:create-index-templates:</code>	Automatically create index templates at application startup when set to true . When templates are installed manually, set to false .	true/ false	true
<code>es:index-prefix:</code>	Optional prefix for distributed tracing platform (Jaeger) indices. For example, setting this to "production" creates indices named "production-tracing-*".		

Table 4.11. ES bulk processor configuration parameters

Parameter	Description	Values	Default value
<code>es:bulk:actions:</code>	The number of requests that can be added to the queue before the bulk processor decides to commit updates to disk.		1000
<code>es:bulk:flush-interval:</code>	A time.Duration after which bulk requests are committed, regardless of other thresholds. To disable the bulk processor flush interval, set this to zero.		200ms
<code>es:bulk:size:</code>	The number of bytes that the bulk requests can take up before the bulk processor decides to commit updates to disk.		5000000
<code>es:bulk:workers:</code>	The number of workers that are able to receive and commit bulk requests to Elasticsearch.		1

Table 4.12. ES TLS configuration parameters

Parameter	Description	Values	Default value
<code>es:tls:ca:</code>	Path to a TLS Certification Authority (CA) file used to verify the remote servers.		Will use the system truststore by default.
<code>es:tls:cert:</code>	Path to a TLS Certificate file, used to identify this process to the remote servers.		
<code>es:tls:enabled:</code>	Enable transport layer security (TLS) when talking to the remote servers. Disabled by default.	true/ false	false

Parameter	Description	Values	Default value
es: tls: key:	Path to a TLS Private Key file, used to identify this process to the remote servers.		
es: tls: server-name:	Override the expected TLS server name in the certificate of the remote servers.		
es: token-file:	Path to a file containing the bearer token. This flag also loads the Certification Authority (CA) file if it is specified.		

Table 4.13. ES archive configuration parameters

Parameter	Description	Values	Default value
es-archive: bulk: actions:	The number of requests that can be added to the queue before the bulk processor decides to commit updates to disk.		0
es-archive: bulk: flush-interval:	A time.Duration after which bulk requests are committed, regardless of other thresholds. To disable the bulk processor flush interval, set this to zero.		0s
es-archive: bulk: size:	The number of bytes that the bulk requests can take up before the bulk processor decides to commit updates to disk.		0
es-archive: bulk: workers:	The number of workers that are able to receive and commit bulk requests to Elasticsearch.		0

Parameter	Description	Values	Default value
<code>es-archive: create-index- templates:</code>	Automatically create index templates at application startup when set to true . When templates are installed manually, set to false .	true/ false	false
<code>es-archive: enabled:</code>	Enable extra storage.	true/ false	false
<code>es-archive: index-prefix:</code>	Optional prefix for distributed tracing platform (Jaeger) indices. For example, setting this to "production" creates indices named "production-tracing-*".		
<code>es-archive: max-doc-count:</code>	The maximum document count to return from an Elasticsearch query. This will also apply to aggregations.		0
<code>es-archive: max-num-spans:</code>	[Deprecated - Will be removed in a future release, use es-archive.max-doc-count instead.] The maximum number of spans to fetch at a time, per query, in Elasticsearch.		0
<code>es-archive: max-span-age:</code>	The maximum lookback for spans in Elasticsearch.		0s
<code>es-archive: num-replicas:</code>	The number of replicas per index in Elasticsearch.		0

Parameter	Description	Values	Default value
<code>es-archive: num-shards:</code>	The number of shards per index in Elasticsearch.		0
<code>es-archive: password:</code>	The password required by Elasticsearch. See also, es.username .		
<code>es-archive: server-urls:</code>	The comma-separated list of Elasticsearch servers. Must be specified as fully qualified URLs, for example, http://localhost:9200 .		
<code>es-archive: sniffer:</code>	The sniffer configuration for Elasticsearch. The client uses the sniffing process to find all nodes automatically. Disabled by default.	true/ false	false
<code>es-archive: sniffer-tls- enabled:</code>	Option to enable TLS when sniffing an Elasticsearch Cluster. The client uses the sniffing process to find all nodes automatically. Disabled by default.	true/ false	false
<code>es-archive: timeout:</code>	Timeout used for queries. When set to zero there is no timeout.		0s
<code>es-archive: tls: ca:</code>	Path to a TLS Certification Authority (CA) file used to verify the remote servers.		Will use the system truststore by default.
<code>es-archive: tls: cert:</code>	Path to a TLS Certificate file, used to identify this process to the remote servers.		

Parameter	Description	Values	Default value
<code>es-archive: tls: enabled:</code>	Enable transport layer security (TLS) when talking to the remote servers. Disabled by default.	true/ false	false
<code>es-archive: tls: key:</code>	Path to a TLS Private Key file, used to identify this process to the remote servers.		
<code>es-archive: tls: server-name:</code>	Override the expected TLS server name in the certificate of the remote servers.		
<code>es-archive: token-file:</code>	Path to a file containing the bearer token. This flag also loads the Certification Authority (CA) file if it is specified.		
<code>es-archive: username:</code>	The username required by Elasticsearch. The basic authentication also loads CA if it is specified. See also es-archive.password .		
<code>es-archive: version:</code>	The major Elasticsearch version. If not specified, the value will be auto-detected from Elasticsearch.		0

Storage example with volume mounts

```

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

```

The following example shows a Jaeger CR using an external Elasticsearch cluster with TLS CA certificate mounted from a volume and user/password stored in a secret.

External Elasticsearch example

```

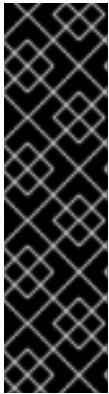
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200 ❶
        index-prefix: my-prefix
        tls: ❷
          ca: /es/certificates/ca.crt
        secretName: tracing-secret ❸
    volumeMounts: ❹
      - name: certificates
        mountPath: /es/certificates/
        readOnly: true
    volumes:
      - name: certificates
        secret:
          secretName: quickstart-es-http-certs-public

```

- ❶ URL to Elasticsearch service running in default namespace.
- ❷ TLS configuration. In this case only CA certificate, but it can also contain `es.tls.key` and `es.tls.cert` when using mutual TLS.
- ❸ Secret which defines environment variables `ES_PASSWORD` and `ES_USERNAME`. Created by `kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic`
- ❹ Volume mounts and volumes which are mounted into all storage components.

4.2.6.6. Managing certificates with Elasticsearch

You can create and manage certificates using the OpenShift Elasticsearch Operator. Managing certificates using the OpenShift Elasticsearch Operator also lets you use a single Elasticsearch cluster with multiple Jaeger Collectors.



IMPORTANT

Managing certificates with Elasticsearch is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

Starting with version 2.4, the Red Hat OpenShift distributed tracing platform (Jaeger) Operator delegates certificate creation to the OpenShift Elasticsearch Operator by using the following annotations in the Elasticsearch custom resource:

- **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"**

Where the `<shared-es-node-name>` is the name of the Elasticsearch node. For example, if you create an Elasticsearch node named **custom-es**, your custom resource might look like the following example.

Example Elasticsearch CR showing annotations

```
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

```

Prerequisites

- The Red Hat OpenShift Service Mesh Operator is installed.
- The {logging-title} is installed with default configuration in your cluster.
- The Elasticsearch node and the Jaeger instances must be deployed in the same namespace. For example, **tracing-system**.

You enable certificate management by setting **spec.storage.elasticsearch.useCertManagement** to **true** in the Jaeger custom resource.

Example showing useCertManagement

```

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

```

The Red Hat OpenShift distributed tracing platform (Jaeger) Operator sets the Elasticsearch custom resource **name** to the value of **spec.storage.elasticsearch.name** from the Jaeger custom resource when provisioning Elasticsearch.

The certificates are provisioned by the OpenShift Elasticsearch Operator and the Red Hat OpenShift distributed tracing platform (Jaeger) Operator injects the certificates.

4.2.6.7. Query configuration options

Query is a service that retrieves traces from storage and hosts the user interface to display them.

Table 4.14. Parameters used by the Red Hat OpenShift distributed tracing platform (Jaeger) Operator to define Query

Parameter	Description	Values	Default value
spec: query: replicas:	Specifies the number of Query replicas to create.	Integer, for example, 2	

Table 4.15. Configuration parameters passed to Query

Parameter	Description	Values	Default value
spec: query: options: {}	Configuration options that define the Query service.		
options: log-level:	Logging level for Query.	Possible values: debug , info , warn , error , fatal , panic .	
options: query: base-path:	The base path for all jaeger-query HTTP routes can be set to a non-root value, for example, /jaeger would cause all UI URLs to start with /jaeger . This can be useful when running jaeger-query behind a reverse proxy.	/<path>	

Sample Query configuration

```

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 configuration options

Ingester is a service that reads from a Kafka topic and writes to the Elasticsearch storage backend. If you are using the **allInOne** or **production** deployment strategies, you do not need to configure the Ingester service.

Table 4.16. Jaeger parameters passed to the Ingester

Parameter	Description	Values
spec: ingester: options: {}	Configuration options that define the Ingestor service.	
options: deadlockInterval:	Specifies the interval, in seconds or minutes, that the Ingestor must wait for a message before terminating. The deadlock interval is disabled by default (set to 0), to avoid terminating the Ingestor when no messages arrive during system initialization.	Minutes and seconds, for example, 1m0s . Default value is 0 .
options: kafka: consumer: topic:	The topic parameter identifies the Kafka configuration used by the collector to produce the messages, and the Ingestor to consume the messages.	Label for the consumer. For example, jaeger-spans .
options: kafka: consumer: brokers:	Identifies the Kafka configuration used by the Ingestor to consume the messages.	Label for the broker, for example, my-cluster-kafka-brokers.kafka:9092 .
options: log-level:	Logging level for the Ingestor.	Possible values: debug, info, warn, error, fatal, dpanic, panic .

Streaming Collector and Ingestor example

```

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
  ingestor:
    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. Injecting sidecars

The Red Hat OpenShift distributed tracing platform (Jaeger) relies on a proxy sidecar within the application's pod to provide the Agent. The Red Hat OpenShift distributed tracing platform (Jaeger) Operator can inject Agent sidecars into deployment workloads. You can enable automatic sidecar injection or manage it manually.

4.2.7.1. Automatically injecting sidecars

The Red Hat OpenShift distributed tracing platform (Jaeger) Operator can inject Jaeger Agent sidecars into deployment workloads. To enable automatic injection of sidecars, add the **sidecar.jaegertracing.io/inject** annotation set to either the string **true** or to the distributed tracing platform (Jaeger) instance name that is returned by running **\$ oc get jaegers**. When you specify **true**, there must be only a single distributed tracing platform (Jaeger) instance for the same namespace as the deployment. Otherwise, the Operator is unable to determine which distributed tracing platform (Jaeger) instance to use. A specific distributed tracing platform (Jaeger) instance name on a deployment has a higher precedence than **true** applied on its namespace.

The following snippet shows a simple application that will inject a sidecar, with the agent pointing to the single distributed tracing platform (Jaeger) instance available in the same namespace:

Automatic sidecar injection example

```

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 Set to either the string **true** or to the Jaeger instance name.

When the sidecar is injected, the agent can then be accessed at its default location on **localhost**.

4.2.7.2. Manually injecting sidecars

The Red Hat OpenShift distributed tracing platform (Jaeger) Operator can only automatically inject Jaeger Agent sidecars into Deployment workloads. For controller types other than **Deployments**, such as **StatefulSets** and **DaemonSets**, you can manually define the Jaeger agent sidecar in your specification.

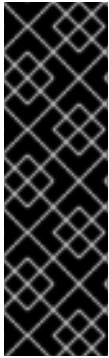
The following snippet shows the manual definition you can include in your containers section for a Jaeger agent sidecar:

Sidecar definition example for a StatefulSet

```
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
```

The agent can then be accessed at its default location on localhost.

4.3. UPGRADING



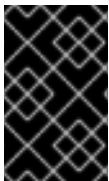
IMPORTANT

The Red Hat OpenShift distributed tracing platform (Jaeger) is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

Operator Lifecycle Manager (OLM) controls the installation, upgrade, and role-based access control (RBAC) of Operators in a cluster. The OLM runs by default in OpenShift Container Platform. OLM queries for available Operators as well as upgrades for installed Operators.

During an update, the Red Hat OpenShift distributed tracing platform Operators upgrade the managed distributed tracing platform instances to the version associated with the Operator. Whenever a new version of the Red Hat OpenShift distributed tracing platform (Jaeger) Operator is installed, all the distributed tracing platform (Jaeger) application instances managed by the Operator are upgraded to the Operator's version. For example, after upgrading the Operator from 1.10 installed to 1.11, the Operator scans for running distributed tracing platform (Jaeger) instances and upgrades them to 1.11 as well.



IMPORTANT

If you have not already updated your OpenShift Elasticsearch Operator as described in [Updating OpenShift Logging](#), complete that update before updating your Red Hat OpenShift distributed tracing platform (Jaeger) Operator.

4.3.1. Additional resources

- [Operator Lifecycle Manager concepts and resources](#)
- [Updating installed Operators](#)
- [Updating OpenShift Logging](#)

4.4. REMOVING



IMPORTANT

The Red Hat OpenShift distributed tracing platform (Jaeger) is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

The steps for removing Red Hat OpenShift distributed tracing platform from an OpenShift Container Platform cluster are as follows:

1. Shut down any Red Hat OpenShift distributed tracing platform pods.
2. Remove any Red Hat OpenShift distributed tracing platform instances.
3. Remove the Red Hat OpenShift distributed tracing platform (Jaeger) Operator.
4. Remove the Red Hat build of OpenTelemetry Operator.

4.4.1. Removing a distributed tracing platform (Jaeger) instance by using the web console

You can remove a distributed tracing platform (Jaeger) instance in the **Administrator** view of the web console.




WARNING

When deleting an instance that uses in-memory storage, all data is irretrievably lost. Data stored in persistent storage such as Elasticsearch is not deleted when a Red Hat OpenShift distributed tracing platform (Jaeger) instance is removed.

Prerequisites

- You are logged in to the web console as a cluster administrator with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators** → **Installed Operators**.
3. Select the name of the project where the Operators are installed from the **Project** menu, for example, **openshift-operators**.
4. Click the Red Hat OpenShift distributed tracing platform (Jaeger) Operator.
5. Click the **Jaeger** tab.
6. Click the Options menu  next to the instance you want to delete and select **Delete Jaeger**.
7. In the confirmation message, click **Delete**.

4.4.2. Removing a distributed tracing platform (Jaeger) instance by using the CLI

You can remove a distributed tracing platform (Jaeger) instance on the command line.

Prerequisites

- An active OpenShift CLI (**oc**) session by a cluster administrator with the **cluster-admin** role.

TIP

- Ensure that your OpenShift CLI (**oc**) version is up to date and matches your OpenShift Container Platform version.
- Run **oc login**:

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

Procedure

1. Log in with the OpenShift CLI (**oc**) by running the following command:

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

2. To display the distributed tracing platform (Jaeger) instances, run the following command:

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

For example,

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

The names of Operators have the suffix **-operator**. The following example shows two Red Hat OpenShift distributed tracing platform (Jaeger) Operators and four distributed tracing platform (Jaeger) instances:

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

You will see output similar to the following:

```
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. To remove an instance of distributed tracing platform (Jaeger), run the following command:

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

For example:

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

4. To verify the deletion, run the **oc get deployments** command again:

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

For example:


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

You will see generated output that is similar to the following example:

```
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. Removing the Red Hat OpenShift distributed tracing platform Operators

Procedure

1. Follow the instructions in [Deleting Operators from a cluster](#) to remove the Red Hat OpenShift distributed tracing platform (Jaeger) Operator.
2. Optional: After the Red Hat OpenShift distributed tracing platform (Jaeger) Operator has been removed, remove the OpenShift Elasticsearch Operator.