



OpenShift Container Platform 4.14

Red Hat build of OpenTelemetry

Red Hat build of OpenTelemetry

OpenShift Container Platform 4.14 Red Hat build of OpenTelemetry

Red Hat build of OpenTelemetry

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

오픈 소스 OpenTelemetry 프로젝트의 Red Hat 빌드를 사용하여 클라우드 네이티브 소프트웨어에 대한 통합, 표준화 및 벤더 중립 Telemetry 데이터 수집을 수집합니다.

Table of Contents

1장. 릴리스 노트	3
1.1. RED HAT BUILD OF OPENTELEMETRY 3.1 릴리스 노트	3
1.2. RED HAT BUILD OF OPENTELEMETRY의 이전 릴리스 노트	4
2장. OPENTELEMETRY RED HAT 빌드 설치	14
2.1. 웹 콘솔에서 RED HAT BUILD OF OPENTELEMETRY 설치	14
2.2. CLI를 사용하여 OPENTELEMETRY RED HAT 빌드 설치	15
2.3. 추가 리소스	18
3장. OPENTELEMETRY 수집기 구성	19
3.1. OPENTELEMETRY 수집기 구성 옵션	19
3.2. OPENTELEMETRY 수집기 구성 요소	22
3.3. 대상 AL CRYOSTAT	57
4장. 계측 구성	61
4.1. OPENTELEMETRY 조정 구성 옵션	61
5장. OPENTELEMETRY 수집기에 추적 및 메트릭 전송	70
5.1. 사이드카 삽입을 사용하여 추적 및 메트릭을 OPENTELEMETRY 수집기로 전송	70
5.2. 사이드카 삽입 없이 OPENTELEMETRY 수집기에 추적 및 메트릭 전송	72
6장. 모니터링 스택에 메트릭을 전송하기 위한 구성	76
7장. TEMPOSTACK으로 추적 전달	78
8장. OPENTELEMETRY 수집기 메트릭 구성	82
9장. 여러 클러스터에서 관찰 데이터 수집	83
10장. RED HAT BUILD OF OPENTELEMETRY 문제 해결	90
10.1. OPENTELEMETRY 수집기 로그 가져오기	90
10.2. 메트릭 노출	90
10.3. 디버그 내보내기	91
11장. 분산 추적 플랫폼(JAEGER)에서 OPENTELEMETRY의 RED HAT 빌드로 마이그레이션	93
11.1. 분산 추적 플랫폼(JAEGER)에서 사이드카를 사용하여 OPENTELEMETRY의 RED HAT 빌드로 마이그레이션	93
11.2. 분산 추적 플랫폼(JAEGER)에서 사이드카 없이 OPENTELEMETRY의 RED HAT 빌드로 마이그레이션	96
12장. OPENTELEMETRY RED HAT 빌드 업데이트	100
12.1. 추가 리소스	100
13장. OPENTELEMETRY RED HAT 빌드 제거	101
13.1. 웹 콘솔을 사용하여 OPENTELEMETRY 수집기 인스턴스 제거	101
13.2. CLI를 사용하여 OPENTELEMETRY 수집기 인스턴스 제거	102
13.3. 추가 리소스	103

1장. 릴리스 노트

1.1. RED HAT BUILD OF OPENTELEMETRY 3.1 릴리스 노트

1.1.1. Red Hat build of OpenTelemetry 개요

Red Hat build of OpenTelemetry는 오픈 소스 [OpenTelemetry 프로젝트](#)를 기반으로 하며, 이는 클라우드 네이티브 소프트웨어에 대한 통합, 표준화 및 벤더 중립 Telemetry 데이터 수집을 제공하는 것을 목표로 합니다. Red Hat build of OpenTelemetry 제품은 OpenTelemetry 수집기 배포 및 관리와 워크로드 조정 간소화를 지원합니다.

[OpenTelemetry 수집기](#)는 여러 형식으로 Telemetry 데이터를 수신, 처리 및 전달할 수 있으므로 Telemetry 처리 및 원격 분석 시스템 간의 상호 운용성에 이상적인 구성 요소입니다. 수집기는 메트릭, 추적 및 로그를 수집하고 처리하기 위한 통합 솔루션을 제공합니다.

OpenTelemetry 수집기에는 다음과 같은 여러 기능이 있습니다.

데이터 수집 및 처리 허브

다양한 소스의 지표 및 추적과 같은 Telemetry 데이터를 수집하는 중앙 구성 요소 역할을 합니다. 이러한 데이터는 조정된 애플리케이션 및 인프라에서 생성할 수 있습니다.

사용자 지정할 수 있는 Telemetry 데이터 파이프라인

OpenTelemetry 수집기는 사용자 지정이 가능하도록 설계되었습니다. 다양한 프로세서, 내보내기 및 수신자를 지원합니다.

자동 복원 기능

자동 계측은 애플리케이션에 관찰 기능을 추가하는 프로세스를 간소화합니다. 개발자는 기본 Telemetry 데이터를 위해 코드를 수동으로 조정할 필요가 없습니다.

다음은 OpenTelemetry 수집기의 몇 가지 사용 사례입니다.

중앙 집중식 데이터 수집

マイ크로 서비스 아키텍처에서 수집기를 배포하여 여러 서비스의 데이터를 집계할 수 있습니다.

데이터 강화 및 처리

데이터 분석 툴로 전달하기 전에 수집기는 이 데이터를 강화, 필터링 및 처리할 수 있습니다.

multi-backend 수신 및 내보내기

수집기는 여러 모니터링 및 분석 플랫폼에 동시에 데이터를 수신하고 보낼 수 있습니다.

1.1.2. 새로운 기능 및 개선 사항

이번 업데이트에서는 다음과 같은 향상된 기능이 도입되었습니다.

- Red Hat build of OpenTelemetry 3.1은 [OpenTelemetry](#) 0.93.0을 기반으로 합니다.
- OpenTelemetry 수집기에서 대상 후보 지원. 대상 al Cryostat는 OpenTelemetry Operator의 선택적 구성 요소이며 OpenTelemetry 수집기 인스턴스의 배포된 플랫폼에서 Prometheus 수집기 스크립트 대상을 분할합니다. 대상 al Cryostat는 Prometheus **PodMonitor** 및 **ServiceMonitor** 사용자 정의 리소스와의 통합을 제공합니다.

1.1.3. 지원 요청

이 문서에 설명된 절차 또는 일반적으로 OpenShift Container Platform에 문제가 발생하는 경우 [Red Hat 고객 포털](#)에 액세스하십시오.

고객 포털에서 다음을 수행할 수 있습니다.

- Red Hat 제품과 관련된 기사 및 솔루션에 대한 Red Hat 지식베이스를 검색하거나 살펴볼 수 있습니다.
- Red Hat 지원에 대한 지원 케이스 제출할 수 있습니다.
- 다른 제품 설명서에 액세스 가능합니다.

클러스터 문제를 식별하기 위해 [OpenShift Cluster Manager](#)에서 Insights를 사용할 수 있습니다. Insights는 문제에 대한 세부 정보 및 문제 해결 방법에 대한 정보를 제공합니다.

이 문서를 개선하기 위한 제안이 있거나 오류를 발견한 경우 가장 관련 문서 구성 요소에 대해 [Jira 문제](#)를 제출합니다. 섹션 이름 및 OpenShift Container Platform 버전과 같은 구체적인 정보를 제공합니다.

1.1.4. 보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 [CTO Chris Wright의 메시지](#)를 참조하십시오.

1.2. RED HAT BUILD OF OPENTELEMETRY의 이전 릴리스 노트

1.2.1. 분산 추적 개요

서비스 소유자로서 분산 추적을 사용하여 서비스 아키텍처에 대한 정보를 수집하도록 서비스를 조정할 수 있습니다. Red Hat OpenShift 분산 추적 플랫폼을 사용하여 최신 클라우드 네이티브, 마이크로서비스 기반 애플리케이션의 구성 요소 간의 상호 작용을 모니터링, 네트워크 프로파일링 및 문제 해결에 사용할 수 있습니다.

분산 추적 플랫폼을 사용하면 다음 기능을 수행할 수 있습니다.

- 분산 트랜잭션 모니터링
- 성능 및 대기 시간 최적화
- 근본 원인 분석 수행

분산 추적 플랫폼은 다음 세 가지 구성 요소로 구성됩니다.

- **Red Hat OpenShift 분산 추적 플랫폼(Tempo)**은 오픈 소스 [Grafana Tempo](#) 프로젝트를 기반으로 합니다.
- 오픈 소스 OpenTelemetry [프로젝트를 기반으로 하는 Red Hat build of OpenTelemetry](#).
- **Red Hat OpenShift distributed tracing platform(Jaeger)**은 오픈 소스 [Jaeger](#) 프로젝트를 기반으로 합니다.



중요

Jaeger는 FIPS 검증 암호화 모듈을 사용하지 않습니다.

1.2.2. Red Hat build of OpenTelemetry 3.0 릴리스 노트

Red Hat build of OpenTelemetry 3.0은 [OpenTelemetry](#) 0.89.0을 기반으로 합니다.

1.2.2.1. 새로운 기능 및 개선 사항

이번 업데이트에서는 다음과 같은 향상된 기능이 도입되었습니다.

- **OpenShift distributed tracing data collection Operator**는 **OpenTelemetry Operator**의 Red Hat 빌드로 이름이 변경됩니다.
- ARM 아키텍처 지원
- 메트릭 컬렉션에 대한 Prometheus 수신자를 지원합니다.
- Kafka에 추적 및 메트릭을 전송하는 Kafka 수신자 및 내보내기를 지원합니다.
- 클러스터 전체 프록시 환경 지원
- Red Hat build of OpenTelemetry Operator는 Prometheus 내보내기가 활성화된 경우 Prometheus **ServiceMonitor** 사용자 정의 리소스를 생성합니다.
- Operator는 업스트림 OpenTelemetry 자동 복원 라이브러리를 삽입할 수 있는 **Instrumentation** 사용자 정의 리소스를 활성화합니다.

1.2.2.2. 제거 알림

Red Hat build of OpenTelemetry 3.0에서는 Jaeger 내보내기가 제거되었습니다. 버그 수정 및 지원은 2.9 라이프사이클 종료 시에만 제공됩니다. Jaeger 수집기에 데이터를 전송하기 위한 Jaeger 내보내기 대신 OTLP 내보내기를 대신 사용할 수 있습니다.

1.2.2.3. 버그 수정

이번 업데이트에서는 다음과 같은 버그 수정이 도입되었습니다.

- **oc adm catalog mirror** CLI 명령을 사용할 때 연결이 끊긴 환경에 대한 지원이 수정되었습니다.

1.2.2.4. 확인된 문제

현재 알려진 문제가 있습니다.

- [TRACING-3761](#)에서는 버그로 인해 OpenTelemetry Operator의 Red Hat 빌드의 클러스터 모니터링이 비활성화됩니다. 버그로 인해 클러스터 모니터링 및 서비스 모니터 오브젝트에 필요한 레이블 **openshift.io/cluster-monitoring=true**로 인해 클러스터 모니터링이 OpenTelemetry Operator의 Red Hat 빌드에서 메트릭을 스킵하지 않습니다.

해결 방법

다음과 같이 클러스터 모니터링을 활성화할 수 있습니다.

1. Operator 네임스페이스에 다음 레이블을 추가합니다. **oc label namespace openshift-opentelemetry-operator openshift.io/cluster-monitoring=true**
2. 서비스 모니터, 역 할, 역 할 바인딩을 생성합니다.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
```

```

name: opentelemetry-operator-controller-manager-metrics-service
namespace: openshift-opentelemetry-operator
spec:
endpoints:
- bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
path: /metrics
port: https
scheme: https
tlsConfig:
  insecureSkipVerify: true
selector:
matchLabels:
  app.kubernetes.io/name: opentelemetry-operator
  control-plane: controller-manager
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: otel-operator-prometheus
  namespace: openshift-opentelemetry-operator
  annotations:
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
rules:
- apiGroups:
  - ""
resources:
- services
- endpoints
- pods
verbs:
- get
- list
- watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: otel-operator-prometheus
  namespace: openshift-opentelemetry-operator
  annotations:
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: otel-operator-prometheus
subjects:
- kind: ServiceAccount
  name: prometheus-k8s
  namespace: openshift-monitoring

```

1.2.3. Red Hat build of OpenTelemetry 2.9.2 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.9.2는 [OpenTelemetry](#) 0.81.0을 기반으로 합니다.

1.2.3.1. CVE

- 이 릴리스에서는 [CVE-2023-46234](#) 가 수정되었습니다.

1.2.3.2. 확인된 문제

현재 알려진 문제가 있습니다.

- 현재 [Operator 완료도](#) 를 Level IV, Deep Insights로 수동으로 설정해야 합니다. ([TRACING-3431](#))

1.2.4. Red Hat build of OpenTelemetry 2.9.1 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.9.1은 [OpenTelemetry](#) 0.81.0을 기반으로 합니다.

1.2.4.1. CVE

- 이 릴리스에서는 [CVE-2023-44487](#) 이 수정되었습니다.

1.2.4.2. 확인된 문제

현재 알려진 문제가 있습니다.

- 현재 [Operator 완료도](#) 를 Level IV, Deep Insights로 수동으로 설정해야 합니다. ([TRACING-3431](#))

1.2.5. Red Hat build of OpenTelemetry 2.9 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.9는 [OpenTelemetry](#) 0.81.0을 기반으로 합니다.

1.2.5.1. 새로운 기능 및 개선 사항

이번 릴리스에서는 OpenTelemetry의 Red Hat 빌드에 대해 다음과 같은 향상된 기능이 도입되었습니다.

- OTLP 메트릭 수집 지원. 메트릭은 Prometheus 내보내기를 통해 **user-workload-monitoring**에 전달 및 저장할 수 있습니다.
- **OpenTelemetry** 수집기 인스턴스 및 **OpenTelemetry** Operator의 Red Hat 빌드를 업그레이드할 수 있는 Operator 완성 수준 IV, Deep Insights를 지원합니다.
- OTLP 또는 HTTP 및 HTTPS를 사용하여 원격 클러스터의 추적 및 메트릭을 보고합니다.
- **resourcedetection** 프로세서를 통해 OpenShift Container Platform 리소스 속성을 수집합니다.
- **OpenTelemetryCollector** 사용자 지정에서 **managed** 및 **unmanaged** 상태를 지원합니다.

1.2.5.2. 확인된 문제

현재 알려진 문제가 있습니다.

- 현재 [Operator 완료도](#) 를 Level IV, Deep Insights로 수동으로 설정해야 합니다. ([TRACING-3431](#))

1.2.6. Red Hat build of OpenTelemetry 2.8 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.8은 [OpenTelemetry](#) Cryostat4.0을 기반으로 합니다.

1.2.6.1. 버그 수정

이 릴리스에서는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 처리합니다.

1.2.7. Red Hat build of OpenTelemetry 2.7 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.7은 [OpenTelemetry](#) 0.63.1을 기반으로 합니다.

1.2.7.1. 버그 수정

이 릴리스에서는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 처리합니다.

1.2.8. Red Hat build of OpenTelemetry 2.6 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.6은 [OpenTelemetry](#) 0.60을 기반으로 합니다.

1.2.8.1. 버그 수정

이 릴리스에서는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 처리합니다.

1.2.9. Red Hat build of OpenTelemetry 2.5 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.5는 [OpenTelemetry](#) 0.56을 기반으로 합니다.

1.2.9.1. 새로운 기능 및 개선 사항

이번 업데이트에서는 다음과 같은 향상된 기능이 도입되었습니다.

- OpenTelemetry Operator의 Red Hat 빌드에 Kubernetes 리소스 속성 수집 지원

1.2.9.2. 버그 수정

이 릴리스에서는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 처리합니다.

1.2.10. Red Hat build of OpenTelemetry 2.4 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.4는 [OpenTelemetry](#) 0.49를 기반으로 합니다.

1.2.10.1. 버그 수정

이 릴리스에서는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 처리합니다.

1.2.11. Red Hat build of OpenTelemetry 2.3 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.3.1은 [OpenTelemetry](#) 0.44.1을 기반으로 합니다.

Red Hat build of OpenTelemetry 2.3.0은 [OpenTelemetry](#) 0.44.0을 기반으로 합니다.

1.2.11.1. 버그 수정

이 릴리스에서는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 처리합니다.

1.2.12. Red Hat build of OpenTelemetry 2.2 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.2는 [OpenTelemetry](#) 0.42.0을 기반으로 합니다.

1.2.12.1. 기술 프리뷰 기능

2.1 릴리스에 포함된 지원되지 않는 OpenTelemetry 수집기 구성 요소가 제거됩니다.

1.2.12.2. 버그 수정

이 릴리스에서는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 처리합니다.

1.2.13. Red Hat build of OpenTelemetry 2.1 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.1은 [OpenTelemetry](#) 0.41.1을 기반으로 합니다.

1.2.13.1. 기술 프리뷰 기능

이번 릴리스에서는 OpenTelemetry 사용자 정의 리소스 파일에서 인증서를 구성하는 방법에 대한 주요 변경 사항이 추가되었습니다. 이번 업데이트를 통해 다음 예와 같이 **ca_file** 이 사용자 정의 리소스의 **tls** 아래에서 이동합니다.

OpenTelemetry 버전 0.33의 CA 파일 구성

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

OpenTelemetry 버전 0.41.1의 CA 파일 구성

```

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"

```

1.2.13.2. 버그 수정

이 릴리스에서는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 처리합니다.

1.2.14. Red Hat build of OpenTelemetry 2.0 릴리스 노트



중요

Red Hat build of OpenTelemetry는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry 2.0은 [OpenTelemetry](#) 0.33.0을 기반으로 합니다.

이번 릴리스에서는 OpenTelemetry Operator의 Red Hat 빌드를 사용하여 설치하는 [기술 프리뷰로](#) OpenTelemetry가 추가되었습니다. Red Hat build of OpenTelemetry는 [OpenTelemetry API](#) 및 계측을 기반으로 합니다. Red Hat build of OpenTelemetry에는 OpenTelemetry Operator 및 수집기가 포함되어 있습니다. 수집기를 사용하여 OpenTelemetry 또는 Jaeger 프로토콜에서 추적을 수신하고 OpenTelemetry의 Red Hat 빌드로 추적 데이터를 보낼 수 있습니다. 현재 수집기의 다른 기능은 지원되지 않습니다. OpenTelemetry Collector를 사용하면 개발자가 벤더와 무관한 API를 사용하여 코드를 조정할 수 있으며 벤더 종속을 방지하고 관찰 가능성 툴링의 에코시스템 확장을 활성화할 수 있습니다.

1.2.15. 지원 요청

이 문서에 설명된 절차 또는 일반적으로 OpenShift Container Platform에 문제가 발생하는 경우 [Red Hat 고객 포털](#)에 액세스하십시오.

고객 포털에서 다음을 수행 할 수 있습니다.

- Red Hat 제품과 관련된 기사 및 솔루션에 대한 Red Hat 지식베이스를 검색하거나 살펴볼 수 있습니다.
- Red Hat 지원에 대한 지원 케이스 제출할 수 있습니다.
- 다른 제품 설명서에 액세스 가능합니다.

클러스터 문제를 식별하기 위해 [OpenShift Cluster Manager](#)에서 Insights를 사용할 수 있습니다. Insights는 문제에 대한 세부 정보 및 문제 해결 방법에 대한 정보를 제공합니다.

이 문서를 개선하기 위한 제안이 있거나 오류를 발견한 경우 가장 관련 문서 구성 요소에 대해 [Jira 문제](#)를 제출합니다. 섹션 이름 및 OpenShift Container Platform 버전과 같은 구체적인 정보를 제공합니다.

1.2.16. 보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 [CTO Chris Wright의 메시지](#)를 참조하십시오.

2장. OPENTELEMETRY RED HAT 빌드 설치

OpenTelemetry의 Red Hat 빌드를 설치하려면 다음 단계를 수행해야 합니다.

1. OpenTelemetry Operator의 Red Hat 빌드 설치.
2. OpenTelemetry 수집기 인스턴스에 대한 네임스페이스 생성.
3. **OpenTelemetryCollector** 사용자 지정 리소스를 생성하여 OpenTelemetry 수집기 인스턴스를 배포합니다.

2.1. 웹 콘솔에서 RED HAT BUILD OF OPENTELEMETRY 설치

웹 콘솔의 관리자 보기에서 OpenTelemetry의 Red Hat 빌드를 설치할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 클러스터 관리자로 웹 콘솔에 로그인되어 있습니다.
- Red Hat OpenShift Dedicated의 경우 **dedicated-admin** 역할의 계정을 사용하여 로그인해야 합니다.

프로세스

1. OpenTelemetry Operator의 Red Hat 빌드를 설치합니다.
 - a. Operators → OperatorHub로 이동하여 **OpenTelemetry Operator**의 Red Hat 빌드를 검색합니다.
 - b. Red Hat → Install → Install → View Operator에서 제공하는 OpenTelemetry Operator의 Red Hat 빌드를 선택합니다.



중요

이렇게 하면 기본 사전 설정을 사용하여 Operator가 설치됩니다.

- 업데이트 채널 → stable
- 설치 모드 → 클러스터의 모든 네임스페이스
- 설치된 네임스페이스 → openshift-operators
- 업데이트 승인 → 자동

- c. 설치된 Operator 페이지의 세부 정보 탭에서 ClusterServiceVersion 세부 정보에서 설치 상태가 **Succeeded**인지 확인합니다.
2. 홈 → 프로젝트 → 프로젝트 생성으로 이동하여 다음 단계에서 생성할 OpenTelemetry 수집기 인스턴스에 대해 선택한 프로젝트를 생성합니다.
3. OpenTelemetry 수집기 인스턴스를 생성합니다.
 - a. Operator → 설치된 Operator로 이동합니다.
 - b. OpenTelemetry Collector → OpenTelemetry 수집기 → YAML 보기 선택합니다.

- c. YAML 보기에서 OTLP, Jaeger, Zipkin 수신기 및 디버그 내보내기를 사용하여 **OpenTelemetryCollector** CR(사용자 정의 리소스)을 사용자 지정합니다.

```

apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: <project_of_opentelemetry_collector_instance>
spec:
  mode: deployment
  config: |
    receivers:
      otlp:
        protocols:
          grpc:
          http:
    jaeger:
      protocols:
        grpc:
        thrift_binary:
        thrift_compact:
        thrift_http:
    zipkin:
  processors:
    batch:
      memory_limiter:
        check_interval: 1s
        limit_percentage: 50
        spike_limit_percentage: 30
  exporters:
    debug:
    service:
      pipelines:
        traces:
          receivers: [otlp,jaeger,zipkin]
          processors: [memory_limiter,batch]
          exporters: [debug]

```

- d. 생성을 선택합니다.

검증

1. **Project:** 드롭다운 목록을 사용하여 **OpenTelemetry 수집기** 인스턴스의 프로젝트를 선택합니다.
2. **Operator → 설치된 Operator**로 이동하여 **OpenTelemetry 수집기** 인스턴스의 상태가 **Condition: Ready**인지 확인합니다.
3. 워크로드 → Pod로 이동하여 **OpenTelemetry 수집기** 인스턴스의 모든 구성 요소 Pod가 실행 중인지 확인합니다.

2.2. CLI를 사용하여 OPENTELEMETRY RED HAT 빌드 설치

명령줄에서 OpenTelemetry의 Red Hat 빌드를 설치할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 클러스터 관리자가 활성 OpenShift CLI(**oc**) 세션입니다.

작은 정보

- OpenShift CLI(**oc**) 버전이 최신 버전인지 확인하고 OpenShift Container Platform 버전과 일치하는지 확인합니다.
- **oc login** 을 실행합니다.

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

프로세스

1. OpenTelemetry Operator의 Red Hat 빌드를 설치합니다.

- a. 다음 명령을 실행하여 Red Hat build of OpenTelemetry Operator 프로젝트를 생성합니다.

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

- b. 다음 명령을 실행하여 Operator group을 생성합니다.

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

- c. 다음 명령을 실행하여 서브스크립션을 생성합니다.

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

- d. 다음 명령을 실행하여 Operator 상태를 확인합니다.

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

2. 후속 단계에서 생성할 OpenTelemetry 수집기 인스턴스에 대해 선택한 프로젝트를 생성합니다.

- 메타데이터 없이 프로젝트를 생성하려면 다음 명령을 실행합니다.

```
$ oc new-project <project_of_opentelemetry_collector_instance>
```

- 메타데이터로 프로젝트를 생성하려면 다음 명령을 실행합니다.

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

3. 사용자가 생성한 프로젝트에서 OpenTelemetry 수집기 인스턴스를 생성합니다.



참고

동일한 클러스터의 별도의 프로젝트에서 여러 OpenTelemetry 수집기 인스턴스를 생성할 수 있습니다.

- a. OTLP, Jaeger 및 Zipkin 수신기 및 디버그 내보내기를 사용하여 **OpenTelemetry** 수집기 CR(사용자 정의 리소스)을 사용자 지정합니다.

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: <project_of_opentelemetry_collector_instance>
spec:
  mode: deployment
  config: |
    receivers:
      otlp:
        protocols:
          grpc:
          http:
    jaeger:
      protocols:
        grpc:
        thrift_binary:
        thrift_compact:
        thrift_http:
    zipkin:
  processors:
    batch:
    memory_limiter:
      check_interval: 1s
      limit_percentage: 50
```

```

spike_limit_percentage: 30
exporters:
  debug:
  service:
    pipelines:
      traces:
        receivers: [otlp,jaeger,zipkin]
        processors: [memory_limiter,batch]
        exporters: [debug]

```

- b. 다음 명령을 실행하여 사용자 지정된 CR을 적용합니다.

```

$ oc apply -f - << EOF
<OpenTelemetryCollector_custom_resource>
EOF

```

검증

1. 다음 명령을 실행하여 OpenTelemetry 수집기 Pod의 **status.phase** 이 **Running** 이고 **conditions** 가 **type: Ready** 인지 확인합니다.

```

$ oc get pod -l app.kubernetes.io/managed-by=opentelemetry-
operator,app.kubernetes.io/instance=<namespace>.<instance_name> -o yaml

```

2. 다음 명령을 실행하여 OpenTelemetry 수집기 서비스를 가져옵니다.

```

$ oc get service -l app.kubernetes.io/managed-by=opentelemetry-
operator,app.kubernetes.io/instance=<namespace>.<instance_name>

```

2.3. 추가 리소스

- 클러스터 관리자 생성
- OperatorHub.io
- 웹 콘솔에 액세스
- 웹 콘솔을 사용하여 OperatorHub에서 설치
- 설치된 Operator에서 애플리케이션 생성
- OpenShift CLI 시작하기

3장. OPENTELEMETRY 수집기 구성

Red Hat build of OpenTelemetry Operator는 OpenTelemetry 리소스의 Red Hat 빌드를 생성하고 배포할 때 사용할 아키텍처 및 구성 설정을 정의하는 CRD(사용자 정의 리소스 정의) 파일을 사용합니다. 기본 구성을 설치하거나 파일을 수정할 수 있습니다.

3.1. OPENTELEMETRY 수집기 구성 옵션

OpenTelemetry 수집기는 Telemetry 데이터에 액세스하는 5가지 유형의 구성 요소로 구성됩니다.

수신자

내보내기 또는 풀 기반이 될 수 있는 수신자는 데이터가 수집기로 들어오는 방법입니다. 일반적으로 수신자는 지정된 형식으로 데이터를 수락하고 내부 형식으로 변환한 다음 해당 파이프라인에 정의된 프로세서 및 내보내기에 전달합니다. 기본적으로 수신자는 설정되어 있지 않습니다. 하나 이상의 수신자를 구성해야 합니다. 수신자는 하나 이상의 데이터 소스를 지원할 수 있습니다.

프로세서

선택 사항: 프로세서는 수신 및 내보내기 사이에 데이터를 처리합니다. 기본적으로 프로세서는 사용할 수 없습니다. 모든 데이터 소스에 대해 프로세서가 활성화되어 있어야 합니다. 모든 프로세서가 모든 데이터 소스를 지원하는 것은 아닙니다. 데이터 소스에 따라 여러 프로세서가 활성화될 수 있습니다. 프로세서의 순서가 중요합니다.

내보내기

푸시 또는 풀 기반인 풀 기반이 될 수 있는 내보내기는 하나 이상의 백엔드 또는 대상에 데이터를 보내는 방법입니다. 기본적으로 내보내기는 구성되지 않습니다. 하나 이상의 내보내기를 구성해야 합니다. 내보내기는 하나 이상의 데이터 소스를 지원할 수 있습니다. 내보내기는 기본 설정과 함께 사용할 수 있지만 대상 및 보안 설정을 지정하려면 많은 내보내기 구성이 필요합니다.

커넥터

커넥터는 두 개의 파이프라인을 연결합니다. 이는 하나의 파이프라인이 끝날 때 내보내기자로 데이터를 사용하고 다른 파이프라인 시작 시 수신자로 데이터를 내보냅니다. 동일하거나 다른 데이터 유형의 데이터를 사용하고 내보낼 수 있습니다. 소비된 데이터를 요약하기 위해 데이터를 생성하고 내보낼 수 있거나, 단순히 데이터를 복제하거나 라우팅 할 수 있습니다.

확장

확장 기능은 수집기에 기능을 추가합니다. 예를 들어 인증을 수신자 및 내보내기에 자동으로 추가할 수 있습니다.

사용자 정의 리소스 YAML 파일에서 구성 요소의 여러 인스턴스를 정의할 수 있습니다. 구성하는 경우 YAML 파일의 **spec.config.service** 섹션에 정의된 파이프라인을 통해 이러한 구성 요소를 활성화해야 합니다. 필요한 구성 요소만 활성화하는 것이 좋습니다.

OpenTelemetry Collector 사용자 정의 리소스 파일의 예

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: tracing-system
spec:
  mode: deployment
  observability:
    metrics:
      enableMetrics: true
  config: |
    receivers:
```

```

otlp:
  protocols:
    grpc:
    http:
processors:
exporters:
  otlp:
    endpoint: jaeger-production-collector-headless.tracing-system.svc:4317
    tls:
      ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
prometheus:
  endpoint: 0.0.0.0:8889
  resource_to_telemetry_conversion:
    enabled: true # by default resource attributes are dropped
service: ①
  pipelines:
    traces:
      receivers: [otlp]
      processors: []
      exporters: [jaeger]
    metrics:
      receivers: [otlp]
      processors: []
      exporters: [prometheus]

```

① 구성 요소가 구성되었지만 **service** 섹션에 정의되지 않은 경우 구성 요소가 활성화되지 않습니다.

표 3.1. Operator에서 OpenTelemetry Collector를 정의하는 데 사용하는 매개변수

매개변수	설명	값	Default
receivers:	수신자는 데이터가 수집 기로 들어오는 방법입니다. 기본적으로 수신자는 설정되어 있지 않습니다. 구성이 유효한 것으로 간주되려면 하나 이상의 사용 가능한 수신자가 있어야 합니다. 파이프라인에 추가되면 수신자가 활성화됩니다.	otlp,jaeger,prometheus,zipkin,kafka,opencensus	없음
processors:	프로세서는 내보낸 데이터를 통해 실행됩니다. 기본적으로 프로세서는 사용할 수 없습니다.	batch, memory_limiter, resourcedetection, attributes, span, k8sattributes, filter, routing	없음

매개변수	설명	값	Default
exporters:	내보내기는 하나 이상의 백엔드 또는 대상에 데이터를 보냅니다. 기본적으로 내보내기는 구성되지 않습니다. 구성이 유효한 것으로 간주되려면 하나 이상의 활성화된 내보내기가 있어야 합니다. 파이프라인에 내보내기를 추가하여 사용할 수 있습니다. 내보내기는 기본 설정과 함께 사용할 수 있지만 대상 및 보안 설정을 지정하려면 많은 구성이 필요합니다.	otlp,otlphttp,debug,rometheus,kafka	없음
connectors:	Connectors는 데이터를 end-of-pipeline 내보내기 도구로 사용하고 데이터를 초기-op-pipeline 수신자로 내보내 파이프라인 쌍을 결합합니다. 커넥터를 사용하여 소비된 데이터를 요약, 복제 또는 라우팅 할 수 있습니다.	spanmetrics	없음
extensions:	Telemetry 데이터를 처리하지 않는 작업의 선택적 구성 요소입니다.	bearertokenauth,oauth2client,jaegerremotesAMPLIN,pprof,health_check,memory_barlast,zpages	없음
service: pipelines:	구성 요소는 services.pipeline 의 파이프라인에 추가하여 활성화됩니다.		
service: pipelines: traces: receivers:	service.pipelines.traces 에서 추적을 위해 수신자를 활성화합니다.		없음
service: pipelines: traces: processors:	service.pipelines.traces 에서 추적을 위해 프로세서를 활성화합니다.		없음

매개변수	설명	값	Default
service: pipelines: traces: exporters:	service.pipelines.traces traces에서 추적을 위해 내보내기를 활성화합니다.		없음
service: pipelines: metrics: receivers:	service.pipelines.metrics metrics에 추가하여 메트릭의 수신자를 활성화합니다.		없음
service: pipelines: metrics: processors:	service.pipelines.metrics metrics에 추가하여 metrics의 프로세서를 활성화합니다.		없음
service: pipelines: metrics: exporters:	service.pipelines.metrics metrics에서 메트릭에 내보내기를 활성화합니다.		없음

3.2. OPENTELEMETRY 수집기 구성 요소

3.2.1. 수신자

수신자는 수집기에 데이터를 가져옵니다.

3.2.1.1. OTLP 수신기

OTLP 수신자는 OTLP(OpenTelemetry 프로토콜)를 사용하여 추적 및 메트릭을 수집합니다.

활성화된 OTLP 수신자가 있는 OpenTelemetry 수집기 사용자 정의 리소스

```
config: |
  receivers:
    otlp:
      protocols:
        grpc:
          endpoint: 0.0.0.0:4317 ①
          tls: ②
            ca_file: ca.pem
            cert_file: cert.pem
            key_file: key.pem
            client_ca_file: client.pem ③
```

```

    reload_interval: 1h 4
    http:
      endpoint: 0.0.0.0:4318 5
      tls: 6

    service:
    pipelines:
      traces:
        receivers: [otlp]
    metrics:
      receivers: [otlp]

```

- 1** OTLP gRPC 끝점입니다. 생략하면 기본값 **0.0.0.0:4317** 이 사용됩니다.
- 2** 서버 측 TLS 구성입니다. TLS 인증서의 경로를 정의합니다. 생략하면 TLS가 비활성화됩니다.
- 3** 서버가 클라이언트 인증서를 확인하는 TLS 인증서의 경로입니다. 이렇게 하면 **TLSConfig**에서 **ClientCA** 및 **ClientAuth**의 값이 **RequireAndVerifyClientCert**로 설정됩니다. 자세한 내용은 [Golang TLS 패키지의 Config](#)를 참조하십시오.
- 4** 인증서를 다시 로드하는 시간 간격을 지정합니다. 값을 설정하지 않으면 인증서가 다시 로드되지 않습니다. **reload_interval**은 **ns,us** (또는 Cryostats), **ms,s,m,h** 와 같은 유효한 시간 단위를 포함하는 문자열을 허용합니다.

5

OTLP HTTP 끝점입니다. 기본값은 **0.0.0.0:4318**입니다.

6

서버 측 TLS 구성입니다. 자세한 내용은 **grpc** 프로토콜 구성 섹션을 참조하십시오.

3.2.1.2. Jaeger 수신자

Jaeger 수신자는 Jaeger 형식으로 추적을 수집합니다.

활성화된 Jaeger 수신자가 있는 OpenTelemetry 수집기 사용자 정의 리소스

```

config: |
receivers:
jaeger:
  protocols:
    grpc:
      endpoint: 0.0.0.0:14250 1
    thrift_http:

```

```

endpoint: 0.0.0.0:14268 ②
thrift_compact:
  endpoint: 0.0.0.0:6831 ③
  thrift_binary:
    endpoint: 0.0.0.0:6832 ④
  tls: ⑤

service:
pipelines:
traces:
receivers: [jaeger]

```

1

Jaeger gRPC 끝점입니다. 생략하면 기본값 0.0.0.0:14250 이 사용됩니다.

2

Jaeger Thrift HTTP 끝점입니다. 생략하면 기본값 0.0.0.0:14268 이 사용됩니다.

3

Jaeger Thrift Cryostat 끝점입니다. 생략하면 기본값 0.0.0.0:6831 이 사용됩니다.

4

Jaeger Thrift Binary 끝점입니다. 생략하면 기본값 0.0.0.0:6832 가 사용됩니다.

5

서버 측 **TLS** 구성입니다. 자세한 내용은 **OTLP** 수신기 구성 섹션을 참조하십시오.

3.2.1.3. Prometheus 수신자

Prometheus 수신자는 현재 [기술 프리뷰 기능 전용](#)입니다.

Prometheus 수신자는 지표 끝점을 스크랩합니다.

활성화된 **Prometheus** 수신자가 있는 **OpenTelemetry** 수집기 사용자 정의 리소스

```

config: |
  receivers:
    prometheus:
      config:
        scrape_configs: ①
          - job_name: 'my-app' ②
            scrape_interval: 5s ③
            static_configs:
              - targets: ['my-app.example.svc.cluster.local:8888'] ④
  service:
    pipelines:
      metrics:
        receivers: [prometheus]

```

1

Prometheus 형식을 사용하여 구성을 스크랩합니다.

2

Prometheus 작업 이름입니다.

3

메트릭 데이터를 스크랩하는 **Interval**입니다. 시간 단위를 허용합니다. 기본값은 **1m**입니다.

4

메트릭이 노출되는 대상입니다. 이 예제에서는 예제 프로젝트의 **my-app** 애플리케이션에서 메트릭을 스크랩합니다.

3.2.1.4. Zipkin 수신기

Zipkin 수신기는 **Zipkin v1** 및 **v2** 형식의 추적을 수집합니다.

활성화된 **Zipkin** 수신기가 있는 **OpenTelemetry** 수집기 사용자 정의 리소스

```

config: |
receivers:
  zipkin:
    endpoint: 0.0.0.0:9411 ①
    tls: ②

service:
pipelines:
  traces:
    receivers: [zipkin]

```

①

Zipkin HTTP 끝점입니다. 생략하면 기본값 **0.0.0.0:9411** 이 사용됩니다.

②

서버 측 TLS 구성입니다. 자세한 내용은 **OTLP** 수신기 구성 섹션을 참조하십시오.

3.2.1.5. Kafka 수신자

Kafka 수신자는 현재 기술 프리뷰 기능 전용입니다.

Kafka 수신자는 OTLP 형식으로 Kafka에서 추적, 메트릭 및 로그를 수신합니다.

활성화된 Kafka 수신자가 있는 OpenTelemetry 수집기 사용자 정의 리소스

```

config: |
receivers:
  kafka:
    brokers: ["localhost:9092"] ①
    protocol_version: 2.0.0 ②
    topic: otlp_spans ③
    auth:
      plain_text: ④
        username: example
        password: example
      tls: ⑤
        ca_file: ca.pem

```

```

cert_file: cert.pem
key_file: key.pem
insecure: false ⑥
server_name_override: kafka.example.corp ⑦
service:
pipelines:
traces:
receivers: [kafka]

```

1

Kafka 브로커 목록입니다. 기본값은 **localhost:9092**입니다.

2

Kafka 프로토콜 버전입니다. 예를 들면 **2.0.0**입니다. 이 필드는 필수 항목입니다.

3

읽을 Kafka 주제의 이름입니다. 기본값은 **otlp_spans**입니다.

4

일반 텍스트 인증 구성입니다. 생략하면 일반 텍스트 인증이 비활성화됩니다.

5

클라이언트 측 TLS 구성입니다. TLS 인증서의 경로를 정의합니다. 생략하면 TLS 인증이 비활성화됩니다.

6

서버의 인증서 체인과 호스트 이름을 확인하지 않습니다. 기본값은 **false**입니다.

7

ServerName은 가상 호스팅을 지원하기 위해 클라이언트에서 요청한 서버의 이름을 나타냅니다.

3.2.1.6. OpenCensus 수신자

OpenCensus 수신기는 조정된 코드베이스를 보다 쉽게 마이그레이션할 수 있도록 **OpenCensus** 프로젝트와 이전 버전과의 호환성을 제공합니다. gRPC 또는 HTTP 및 Json을 통해 **OpenCensus** 형식의 메트릭 및 추적을 수신합니다.

활성화된 **OpenCensus** 수신자가 있는 **OpenTelemetry** 수집기 사용자 정의 리소스

```
config: |
  receivers:
    opencensus:
      endpoint: 0.0.0.0:9411 ①
      tls: ②
      cors_allowed_origins: ③
        - https://*.example.com
  service:
    pipelines:
      traces:
        receivers: [opencensus]
        ...
...
```

①

OpenCensus 앤드포인트입니다. 생략하면 기본값은 0.0.0.0:55678입니다.

②

서버 측 **TLS** 구성입니다. 자세한 내용은 **OTLP** 수신기 구성 섹션을 참조하십시오.

③

HTTP JSON 끝점을 사용하여 선택적으로 이 필드에서 허용되는 **CORS** 목록을 지정하여 활성화되는 **CORS**를 구성할 수도 있습니다. * 가 있는 와일드카드는 **cors_allowed_origins**에서 허용됩니다. 모든 원본과 일치하려면 * 만 입력합니다.

3.2.2. 프로세서

프로세서가 수신되고 내보낼 때까지 데이터를 통해 실행됩니다.

3.2.2.1. 배치 프로세서

Batch 프로세서는 추적 및 메트릭을 배치하여 **Telemetry** 정보를 전송하는 데 필요한 발신 연결 수를 줄입니다.

Batch 프로세서를 사용할 때 **OpenTelemetry** 수집기 사용자 정의 리소스의 예

```
config: |
  processor:
    batch:
      timeout: 5s
      send_batch_max_size: 10000
    service:
      pipelines:
        traces:
          processors: [batch]
        metrics:
          processors: [batch]
```

표 3.2. **Batch** 프로세서에서 사용하는 매개변수

매개변수	설명	Default
timeout	배치 크기에 관계없이 특정 기간 후에 배치를 보냅니다.	200ms
send_batch_size	지정된 수의 범위 또는 메트릭 후에 원격 분석 데이터의 배치를 보냅니다.	8192
send_batch_max_size	배치의 최대 허용 크기입니다. send_batch_size 보다 크거나 같아야 합니다.	0
metadata_keys	활성화하면 client.Metadata 에 있는 각 고유한 값 집합에 대해 배쳐 인스턴스가 생성됩니다.	[]
metadata_cardinality_limit	metadata_keys 가 채워지면 이 구성은 프로세스 기간 동안 처리된 고유한 메타데이터 키-값 조합의 수를 제한합니다.	1000

3.2.2.2. 메모리 제한 프로세서

메모리 제한 프로세서는 주기적으로 수집기의 메모리 사용량을 확인하고 소프트 메모리 제한에 도달 할 때 데이터 처리를 일시 중지합니다. 이 프로세서는 추적, 메트릭 및 로그를 지원합니다. 일반적으로 수신자인 이전 구성 요소는 동일한 데이터 전송을 다시 시도해야 하며 들어오는 데이터에 백압을 적용할 수 있습니다. 메모리 사용량이 하드 제한을 초과하면 메모리 제한 프로세서가 가비지 수집을 강제 실행합니다.

메모리 제한 프로세서를 사용할 때 **OpenTelemetry** 수집기 사용자 정의 리소스의 예

```
config: |
  processor:
    memory_limiter:
      check_interval: 1s
      limit_mib: 4000
      spike_limit_mib: 800
    service:
      pipelines:
        traces:
          processors: [batch]
        metrics:
          processors: [batch]
```

표 3.3. 메모리 제한 프로세서에서 사용하는 매개변수

매개변수	설명	Default
check_interval	메모리 사용량 측정 사이의 시간입니다. 최적의 값은 1s 입니다. spiky 트래픽 패턴의 경우 check_interval 을 줄이거나 spike_limit_mib 를 늘릴 수 있습니다.	0s
limit_mib	힙에 할당된 MiB 단위의 최대 메모리 양인 하드 제한입니다. 일반적으로 OpenTelemetry 수집기의 총 메모리 사용량은 이 값보다 약 50MiB입니다.	0
spike_limit_mib	메모리 사용량의 최대 급증(MiB)입니다. 최적의 값은 limit_mib 의 약 20%입니다. 소프트 제한을 계산하면 limit_mib 에서 spike_limit_mib 를 뺀 값입니다.	limit_mib 의 20%

매개변수	설명	Default
limit_percentage	limit_mib 와 동일하지만 사용 가능한 총 메모리의 백분율로 표시됩니다. limit_mib 설정이 이 설정보다 우선합니다.	0
spike_limit_percentage	spike_limit_mib 와 동일하지만 사용 가능한 총 메모리의 백분율로 표시됩니다. limit_percentage 설정과 함께 사용하도록 설계되었습니다.	0

3.2.2.3. 리소스 탐지 프로세서

리소스 탐지 프로세서는 현재 기술 [프리뷰 기능 전용](#)입니다.

리소스 탐지 프로세서는 **OpenTelemetry**의 리소스 의미 체계 표준과 일치하는 호스트 리소스 세부 정보를 식별합니다. 감지된 정보를 사용하여 원격 분석 데이터에서 리소스 값을 추가하거나 교체할 수 있습니다. 이 프로세서는 추적, 메트릭을 지원하며 **Docket** 메타데이터 탐지기 또는 **OTEL_RESOURCE_ATTRIBUTES** 환경 변수 탐지기와 같은 여러 탐지기와 함께 사용할 수 있습니다.

리소스 탐지 프로세서에 필요한 **OpenShift Container Platform** 권한

```
kind: ClusterRole
metadata:
  name: otel-collector
rules:
- apiGroups: ["config.openshift.io"]
  resources: ["infrastructures", "infrastructures/status"]
  verbs: ["get", "watch", "list"]
```

리소스 탐지 프로세서를 사용한 **OpenTelemetry** 수집기

```
config: |
processor:
resourcedetection:
detectors: [openshift]
```

```

override: true
service:
pipelines:
traces:
  processors: [resourcedetection]
metrics:
  processors: [resourcedetection]

```

환경 변수 탐지기와 함께 리소스 탐지 프로세서를 사용한 OpenTelemetry 수집기

```

config: |
processors:
resourcedetection/env:
  detectors: [env] 1
  timeout: 2s
  override: false

```

1

사용할 탐지기를 지정합니다. 이 예에서는 환경 탐지기가 지정됩니다.

3.2.2.4. 특성 프로세서

특성 프로세서는 현재 [기술 프리뷰 기능 전용](#)입니다.

특성 프로세서는 범위, 로그 또는 메트릭의 속성을 수정할 수 있습니다. 입력 데이터를 필터링하고 일치하도록 이 프로세서를 구성하고 특정 작업에 대해 이러한 데이터를 포함하거나 제외할 수 있습니다.

프로세서는 작업 목록에서 작동하며 구성에 지정된 순서대로 실행됩니다. 지원되는 작업은 다음과 같습니다.

삽입

지정된 키가 없는 경우 입력 데이터에 새 특성을 삽입합니다.

업데이트

키가 이미 존재하는 경우 입력 데이터의 속성을 업데이트합니다.

Upsert

삽입 및 업데이트 작업 결합: 키가 아직 없는 경우 새 특성을 삽입합니다. 키가 이미 있는 경우 속성을 업데이트합니다.

delete

입력 데이터에서 특성을 제거합니다.

hash

기존 속성 값을 **SHA1**로 해시합니다.

extract

입력 키에서 규칙에 정의된 대상 키로 정규식 규칙을 사용하여 값을 추출합니다. 대상 키가 이미 존재하는 경우 기존 특성을 소스로 사용하여 **Span** 프로세서의 **to_attributes** 설정과 유사하게 재정의 됩니다.

convert

기존 특성을 지정된 형식으로 변환합니다.**Converts an existing attribute to a specified type.**

특성 프로세서를 사용한 OpenTelemetry 수집기

```
config: |
  processors:
    attributes/example:
      actions:
        - key: db.table
          action: delete
        - key: redacted_span
          value: true
          action: upsert
        - key: copy_key
          from_attribute: key_original
          action: update
        - key: account_id
          value: 2245
```

```

action: insert
- key: account_password
  action: delete
- key: account_email
  action: hash
- key: http.status_code
  action: convert
  converted_type: int

```

3.2.2.5. 리소스 프로세서

리소스 프로세서는 현재 [기술 프리뷰 기능](#) 전용입니다.

리소스 프로세서는 리소스 속성에 변경 사항을 적용합니다. 이 프로세서는 추적, 메트릭 및 로그를 지원합니다.

리소스 탐지 프로세서를 사용한 [OpenTelemetry 수집기](#)

```

config: |
processor:
  attributes:
    - key: cloud.availability_zone
      value: "zone-1"
      action: upsert
    - key: k8s.cluster.name
      from_attribute: k8s-cluster
      action: insert
    - key: redundant-attribute
      action: delete

```

속성은 특성 삭제, 특성 삽입 또는 특성 `upsert`와 같은 리소스 특성에 적용되는 작업을 나타냅니다.

3.2.2.6. 범위 프로세서

[Span](#) 프로세서는 현재 [기술 프리뷰 기능](#) 전용입니다.

Span 프로세서는 특성에 따라 범위 이름을 수정하거나 범위 이름에서 범위 속성을 추출합니다. 범위 상태를 변경할 수도 있습니다. 범위를 포함하거나 제외할 수도 있습니다. 이 프로세서는 추적을 지원합니다.

범위 이름을 변경하려면 **from_attributes** 구성을 사용하여 새 이름에 속성을 지정해야 합니다.

범위 이름을 변경하기 위한 **Span** 프로세서를 사용한 **OpenTelemetry** 수집기

```
config: |
  processor:
    span:
      name:
        from_attributes: [<key1>, <key2>, ...] ①
        separator: <value> ②
```

①

새 범위 이름을 구성할 키를 정의합니다.

②

선택적 구분 기호입니다.

프로세서를 사용하여 범위 이름에서 특성을 추출할 수 있습니다.

범위 이름에서 특성을 추출하기 위한 **Span** 프로세서를 사용하는 **OpenTelemetry** 수집기

```
config: |
  processor:
    span/to_attributes:
      name:
        to_attributes:
          rules:
            - ^/api/v1/document/(?P<documentId>.*)/update$ ①
```

1

이 규칙은 추출을 실행하는 방법을 정의합니다. 예를 들어 정규식이 이름과 일치하는 경우 **documentID attribute** 규칙을 정의할 수 있습니다. 이 예에서 입력 범위 이름이 `/api/v1/document/12345678/update` 이면 `/api/v1/document/{documentId}/update` 출력 범위 이름이 표시되고 새로운 "documentId"="12345678" 특성이 범위에 추가됩니다.

범위 상태를 수정할 수 있습니다.

상태 변경을 위한 **Span Processor**를 사용한 **OpenTelemetry Collector**

```
config: |
  processor:
    span/set_status:
      status:
        code: Error
        description: "<error_description>"
```

3.2.2.7. Kubernetes 속성 프로세서

Kubernetes 특성 프로세서는 현재 [기술 프리뷰 기능 전용](#)입니다.

Kubernetes 특성 프로세서를 사용하면 **Kubernetes** 메타데이터를 사용하여 범위, 메트릭 및 로그 리소스 특성을 자동으로 구성할 수 있습니다. 이 프로세서는 추적, 메트릭 및 로그를 지원합니다. 이 프로세서는 **Kubernetes** 리소스를 자동으로 식별하고, 해당 리소스에서 메타데이터를 추출하고, 이 추출된 메타데이터를 관련 범위, 지표 및 로그에 리소스 속성으로 통합합니다. **Kubernetes API**를 사용하여 클러스터 내에서 작동하는 모든 **Pod**를 검색하고 IP 주소, Pod UID 및 기타 관련 메타데이터의 레코드를 유지 관리합니다.

Kubernetes 속성 프로세서에 필요한 최소 **OpenShift Container Platform** 권한

```
kind: ClusterRole
```

```

metadata:
  name: otel-collector
rules:
  - apiGroups: []
    resources: ['pods', 'namespaces']
    verbs: ['get', 'watch', 'list']

```

Kubernetes 속성 프로세서를 사용한 OpenTelemetry 수집기

```

config: |
processors:
  k8sattributes:
    filter:
      node_from_env_var: KUBE_NODE_NAME

```

3.2.2.8. 필터 프로세서

필터 프로세서는 현재 [기술 프리뷰 기능 전용](#)입니다.

필터 프로세서는 [OpenTelemetry Cryostat Language](#)를 활용하여 Telemetry 데이터 삭제 기준을 설정합니다. 이러한 조건이 충족되면 Telemetry 데이터가 삭제됩니다. 조건은 논리 OR 연산자를 사용하여 결합할 수 있습니다. 이 프로세서는 추적, 메트릭 및 로그를 지원합니다.

활성화된 OTLP 내보내기가 활성화된 OpenTelemetry 수집기 사용자 정의 리소스

```

config: |
processors:
  filter/otlp:
    error_mode: ignore ①
    traces:
      span:
        - 'attributes["container.name"] == "app_container_1"' ②
        - 'resource.attributes["host.name"] == "localhost"' ③

```

1

오류 모드를 정의합니다. `ignore`로 설정하면 조건에서 반환된 오류를 무시합니다. 전파 하도록 설정하면 파이프라인의 오류를 반환합니다. 오류로 인해 수집기에서 페이로드가 삭제됩니다.

2

`container.name == app_container_1` 속성이 있는 범위를 필터링합니다.

3

`host.name == localhost` 리소스 속성이 있는 범위를 필터링합니다.

3.2.2.9. 라우팅 프로세서

라우팅 프로세서는 현재 [기술 프리뷰 기능](#) 전용입니다.

라우팅 프로세서는 로그, 메트릭 또는 추적을 특정 내보내기에 라우팅합니다. 이 프로세서는 들어오는 HTTP 요청(gRPC 또는 일반 HTTP)에서 헤더를 읽거나 리소스 특성을 읽은 다음 `read` 값에 따라 추적 정보를 관련 내보내기로 보낼 수 있습니다.

활성화된 OTLP 내보내기가 활성화된 OpenTelemetry 수집기 사용자 정의 리소스

```
config: |
processors:
routing:
  from_attribute: X-Tenant ①
  default_exporters: ②
    - jaeger
  table: ③
    - value: acme
    exporters: [jaeger/acme]
exporters:
  jaeger:
    endpoint: localhost:14250
  jaeger/acme:
    endpoint: localhost:24250
```

1

경로를 수행할 때 **lookup** 값에 대한 HTTP 헤더 이름입니다.

2

특성 값이 다음 섹션의 표에 없는 경우 기본 내보내기입니다.

3

내보내기로 라우팅 할 값을 정의하는 테이블입니다.

선택적으로 **from_attribute**에서 속성을 찾을 위치를 정의하는 **attribute_source configuration**을 생성할 수 있습니다. 허용되는 값은 HTTP 헤더를 포함하는 컨텍스트 또는 리소스 특성을 검색하는 리소스를 검색하는 컨텍스트입니다.

3.2.3. 내보내기

내보내기는 데이터를 하나 이상의 백엔드 또는 대상에 보냅니다.

3.2.3.1. OTLP 내보내기

OTLP gRPC 내보내기는 OTLP(OpenTelemetry 프로토콜)를 사용하여 추적 및 메트릭을 내보냅니다.

활성화된 OTLP 내보내기가 활성화된 OpenTelemetry 수집기 사용자 정의 리소스

```
config: |
  exporters:
    otlp:
      endpoint: tempo-ingester:4317 ①
      tls: ②
        ca_file: ca.pem
        cert_file: cert.pem
        key_file: key.pem
      insecure: false ③
```

```

insecure_skip_verify: false # 4
reload_interval: 1h 5
server_name_override: <name> 6
headers: 7
  X-Scope-OrgID: "dev"
service:
  pipelines:
    traces:
      exporters: [otlp]
  metrics:
    exporters: [otlp]

```

1

OTLP gRPC 끝점입니다. **https://** 스키마를 사용하는 경우 클라이언트 전송 보안이 활성화되고 **tls**의 **insecure** 설정을 덮어씁니다.

2

클라이언트 측 **TLS** 구성입니다. **TLS** 인증서의 경로를 정의합니다.

3

true로 설정된 경우 클라이언트 전송 보안을 비활성화합니다. 기본값은 기본적으로 **false**입니다.

4

true로 설정된 경우 인증서 확인을 건너뜁니다. 기본값은 **false**입니다.

5

인증서를 다시 로드하는 시간 간격을 지정합니다. 값을 설정하지 않으면 인증서가 다시 로드되지 않습니다. **reload_interval**은 **ns,us** (또는 **Cryostats**), **ms,s,m,h** 와 같은 유효한 시간 단위를 포함하는 문자열을 허용합니다.

6

요청의 권한 헤더 필드와 같은 권한의 가상 호스트 이름을 재정의합니다. 테스트에 이 값을 사용할 수 있습니다.

7

설정된 연결 중에 수행되는 모든 요청에 대해 헤더가 전송됩니다.

3.2.3.2. OTLP HTTP 내보내기

OTLP HTTP 내보내기는 OTLP(OpenTelemetry 프로토콜)를 사용하여 추적 및 메트릭을 내보냅니다.

활성화된 OTLP 내보내기가 활성화된 OpenTelemetry 수집기 사용자 정의 리소스

```
config:
  exporters:
    otlphttp:
      endpoint: http://tempo-ingester:4318 ①
      tls: ②
      headers: ③
        X-Scope-OrgID: "dev"
      disable_keep_alives: false ④

  service:
    pipelines:
      traces:
        exporters: [otlphttp]
      metrics:
        exporters: [otlphttp]
```

1

OTLP HTTP 끝점입니다. <https://> 스키마를 사용하는 경우 클라이언트 전송 보안이 활성화되고 `tls`의 `insecure` 설정을 덮어씁니다.

2

클라이언트 측 TLS 구성입니다. TLS 인증서의 경로를 정의합니다.

3

헤더는 모든 HTTP 요청에 전송됩니다.

4

true인 경우 **HTTP keep-alive**를 비활성화합니다. 단일 **HTTP** 요청에 대해서만 서버에 대한 연결을 사용합니다.

3.2.3.3. 디버그 내보내기

디버그 내보내기는 추적 및 메트릭을 표준 출력에 출력합니다.

활성화된 디버그 내보내기가 있는 **OpenTelemetry** 수집기 사용자 정의 리소스

```
config: |
  exporters:
    debug:
      verbosity: detailed ①
  service:
    pipelines:
      traces:
        exporters: [logging]
      metrics:
        exporters: [logging]
```

①

디버그 내보내기의 상세 정보 표시: 세부 정보 또는 일반 또는 기본. **detailed** 정보로 설정하면 파이프라인 데이터가 세부적으로 기록됩니다. 기본값은 **normal**입니다.

3.2.3.4. Prometheus 내보내기

Prometheus 내보내기는 현재 [기술 프리뷰 기능 전용](#)입니다.

Prometheus 내보내기는 **Prometheus** 또는 **OpenMetrics** 형식으로 지표를 내보냅니다.

활성화된 **Prometheus** 내보내기가 포함된 **OpenTelemetry** 수집기 사용자 정의 리소스

```

ports:
- name: promexporter 1
  port: 8889
  protocol: TCP
config: |
  exporters:
    prometheus:
      endpoint: 0.0.0.0:8889 2
      tls: 3
        ca_file: ca.pem
        cert_file: cert.pem
        key_file: key.pem
      namespace: prefix 4
      const_labels: 5
        label1: value1
      enable_open_metrics: true 6
      resource_to_telemetry_conversion: 7
        enabled: true
      metric_expiration: 180m 8
      add_metric_suffixes: false 9
service:
  pipelines:
    metrics:
      exporters: [prometheus]

```

1

수집기 Pod 및 서비스에서 Prometheus 포트를 노출합니다. ServiceMonitor 또는 PodMonitor 사용자 정의 리소스의 포트 이름을 사용하여 Prometheus에서 메트릭 스크랩을 활성화할 수 있습니다.

2

메트릭이 노출되는 네트워크 끝점입니다.

3

서버 측 TLS 구성입니다. TLS 인증서의 경로를 정의합니다.

4

설정된 경우 제공된 값 아래에 메트릭을 내보냅니다. 기본값이 없습니다.

5

6

`true` 인 경우 **OpenMetrics** 형식을 사용하여 메트릭을 내보냅니다. 예시는 **OpenMetrics** 형식으로만 내보내지며 히스토그램 및 단조적 합계 메트릭 (예: `counter`)에만 사용할 수 있습니다. 기본적으로 비활성되어 있습니다.

7

`enabled`가 `true` 이면 모든 리소스 속성이 기본적으로 지표 레이블로 변환됩니다. 기본적으로 비활성되어 있습니다.

8

업데이트 없이 메트릭이 노출되는 기간을 정의합니다. 기본값은 **5m**입니다.

9

지표 유형 및 단위 접미사를 추가합니다. **Jaeger** 콘솔의 모니터 탭이 활성화된 경우 비활성화해야 합니다. 기본값은 `true`입니다.

3.2.3.5. Kafka 내보내기

Kafka 내보내기는 현재 [기술 프리뷰 기능 전용](#)입니다.

Kafka 내보내기는 로그, 메트릭 및 추적을 Kafka로 내보냅니다. 이 내보내기에서는 메시지를 차단하고 배치하지 않는 동기 생산자를 사용합니다. 처리량과 복원력이 높아지려면 일괄 처리 및 대기열에 있는 재시도 프로세서와 함께 사용해야 합니다.

활성화된 Kafka 내보내기가 포함된 OpenTelemetry 수집기 사용자 정의 리소스

```
config: |
  exporters:
    kafka:
      brokers: ["localhost:9092"] ①
      protocol_version: 2.0.0 ②
      topic: otlp_spans ③
      auth:
        plain_text: ④
          username: example
          password: example
```

```

tls: ⑤
  ca_file: ca.pem
  cert_file: cert.pem
  key_file: key.pem
  insecure: false ⑥
    server_name_override: kafka.example.corp ⑦
service:
pipelines:
traces:
  exporters: [kafka]

```

1

Kafka 브로커 목록입니다. 기본값은 **localhost:9092**입니다.

2

Kafka 프로토콜 버전입니다. 예를 들면 **2.0.0**입니다. 이 필드는 필수 항목입니다.

3

읽을 Kafka 주제의 이름입니다. 다음은 추적의 경우 **otlp_spans**이며 지표의 경우 **otlp_metrics**, 로그의 경우 **otlp_logs**입니다.

4

일반 텍스트 인증 구성입니다. 생략하면 일반 텍스트 인증이 비활성화됩니다.

5

클라이언트 측 **TLS** 구성입니다. **TLS** 인증서의 경로를 정의합니다. 생략하면 **TLS** 인증이 비활성화됩니다.

6

서버의 인증서 체인과 호스트 이름을 확인하지 않습니다. 기본값은 **false**입니다.

7

ServerName은 가상 호스팅을 지원하기 위해 클라이언트에서 요청한 서버의 이름을 나타냅니다.

3.2.4. 커넥터

커넥터는 두 개의 파이프라인을 연결합니다.

3.2.4.1. Spanmetrics 커넥터

Spanmetrics 커넥터는 현재 [기술 프리뷰](#) 기능 전용입니다.

Spanmetrics 커넥터는 범위 데이터에서 요청, 오류 및 기간(R.E.D) OpenTelemetry 메트릭을 집계합니다.

활성화된 spanmetrics 커넥터가 있는 OpenTelemetry 수집기 사용자 정의 리소스

```
config: |
  connectors:
    spanmetrics:
      metrics_flush_interval: 15s ①
  service:
    pipelines:
      traces:
        exporters: [spanmetrics]
      metrics:
        receivers: [spanmetrics]
```

①

생성된 지표의 플러시 간격을 정의합니다. 기본값은 15s입니다.

3.2.5. 확장

확장 기능은 수집기에 기능을 추가합니다.

3.2.5.1. BearerTokenAuth 확장

BearerTokenAuth 확장은 현재 [기술 프리뷰](#) 기능 전용입니다.

BearerTokenAuth 확장은 **HTTP** 및 **gRPC** 프로토콜을 기반으로 하는 수신자 및 내보내기의 인증자입니다. **OpenTelemetry Collector** 사용자 정의 리소스를 사용하여 수신자 및 내보내기자 측에서 **BearerTokenAuth** 확장에 대한 클라이언트 인증 및 서버 인증을 구성할 수 있습니다. 이 확장에서는 추적, 메트릭 및 로그를 지원합니다.

BearerTokenAuth 확장에 대해 구성된 클라이언트 및 서버 인증이 포함된 **OpenTelemetry** 수집기 사용자 정의 리소스

```
config: |
  extensions:
    bearertokenauth:
      scheme: "Bearer" ①
      token: "<token>" ②
      filename: "<token_file>" ③
  receivers:
    otlp:
      protocols:
        http:
          auth:
            authenticator: bearertokenauth ④
  exporters:
    otlp:
      auth:
        authenticator: bearertokenauth ⑤
  service:
    extensions: [bearertokenauth]
    pipelines:
      traces:
        receivers: [otlp]
        exporters: [otlp]
```

1

BearerTokenAuth 확장을 구성하여 사용자 지정 스키마를 보낼 수 있습니다. 기본값은 **Bearer**입니다.

2

BearerTokenAuth 확장 토큰을 메타데이터로 추가하여 메시지를 확인할 수 있습니다.

3

모든 메시지와 함께 전송되는 권한 부여 토큰이 포함된 파일의 경로입니다.

4

인증자 구성은 **OTLP** 수신기에 할당할 수 있습니다.

5

인증자 구성은 **OTLP** 내보내기에 할당할 수 있습니다.

3.2.5.2. OAuth2Client 확장

OAuth2Client 확장은 현재 [기술 프리뷰](#) 기능 전용입니다.

OAuth2Client 확장은 **HTTP** 및 **gRPC** 프로토콜을 기반으로 하는 내보내기의 인증자입니다.
OAuth2Client 확장에 대한 클라이언트 인증은 **OpenTelemetry** 수집기 사용자 정의 리소스의 별도의 섹션에 구성되어 있습니다. 이 확장에서는 추적, 메트릭 및 로그를 지원합니다.

OAuth2Client 확장을 위해 구성된 클라이언트 인증이 있는 **OpenTelemetry** 수집기 사용자 정의 리소스

```
config: |
  extensions:
    oauth2client:
      client_id: <client_id> ①
      client_secret: <client_secret> ②
      endpoint_params: ③
        audience: <audience>
      token_url: https://example.com/oauth2/default/v1/token ④
      scopes: ["api.metrics"] ⑤
      # tls settings for the token client
      tls: ⑥
        insecure: true ⑦
        ca_file: /var/lib/mycert.pem ⑧
        cert_file: <cert_file> ⑨
        key_file: <key_file> ⑩
      timeout: 2s ⑪
```

```

receivers:
  otlp:
    protocols:
      http:

exporters:
  otlp:
    auth:
      authenticator: oauth2client ⑫

service:
  extensions: [oauth2client]
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [otlp]

```

1

ID 공급자가 제공하는 클라이언트 식별자입니다.

2

ID 공급자에 대한 클라이언트를 인증하는 데 사용되는 기밀 키입니다.

3

인증 중에 전송되는 키-값 쌍 형식의 추가 메타데이터입니다. 예를 들어 대상자는 토큰 수신자를 나타내는 액세스 토큰의 의도된 대상을 지정합니다.

4

수집기에서 액세스 토큰을 요청하는 OAuth2 토큰 끝점의 URL입니다.

5

범위는 클라이언트에서 요청한 특정 권한 또는 액세스 수준을 정의합니다.

6

토큰을 요청할 때 보안 연결을 설정하는 데 사용되는 토큰 클라이언트의 TLS(전송 계층 보안) 설정입니다.

7

true로 설정하면 비보안 또는 검증되지 않은 **TLS** 연결을 사용하여 구성된 토큰 끝점을 호출하도록 수집기를 구성합니다.

8

TLS 핸드셰이크 중 서버의 인증서를 확인하는 데 사용되는 **CA(인증 기관)** 파일의 경로입니다.

9

필요한 경우 클라이언트가 **OAuth2** 서버에 자신을 인증하는 데 사용해야 하는 클라이언트 인증서 파일의 경로입니다.

10

인증에 필요한 경우 클라이언트 인증서와 함께 사용되는 클라이언트의 개인 키 파일의 경로입니다.

11

토큰 클라이언트 요청에 대한 타임아웃을 설정합니다.

12

인증자 구성을 **OTLP** 내보내기에 할당할 수 있습니다.

3.2.5.3. Jaeger 원격 샘플링 확장

Jaeger 원격 샘플링 확장은 현재 [기술 프리뷰](#) 기능 전용입니다.

Jaeger 원격 샘플링 확장에서는 **Jaeger**의 원격 샘플링 **API** 후 샘플링 전략을 제공할 수 있습니다. **Jaeger** 수집기가 파이프라인 다운 또는 로컬 파일 시스템의 정적 **JSON** 파일과 같은 백업 원격 샘플링 서버에 요청을 프록시하도록 이 확장을 구성할 수 있습니다.

구성된 **Jaeger** 원격 샘플링 확장 기능이 있는 **OpenTelemetry** 수집기 사용자 정의 리소스

```
config: |
  extensions:
    jaegerremotesampling:
```

```

source:
  reload_interval: 30s ①
  remote:
    endpoint: jaeger-collector:14250 ②
    file: /etc/otelcol/sampling_strategies.json ③

receivers:
  otlp:
    protocols:
      http:

exporters:
  otlp:

service:
  extensions: [jaegerremotesampling]
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [otlp]

```

①

샘플링 구성이 업데이트되는 시간 간격입니다.

②

Jaeger 원격 샘플링 전략 공급자에 도달하기 위한 끝점입니다.

③

JSON 형식의 샘플링 전략 구성이 포함된 로컬 파일의 경로입니다.

Jaeger 원격 샘플링 전략 파일의 예

```
{
  "service_strategies": [
    {
      "service": "foo",
      "type": "probabilistic",
      "param": 0.8,
      "operation_strategies": [
        {
          "operation": "op1",

```

```

    "type": "probabilistic",
    "param": 0.2
  },
  {
    "operation": "op2",
    "type": "probabilistic",
    "param": 0.4
  }
]
},
{
  "service": "bar",
  "type": "ratelimiting",
  "param": 5
}
],
"default_strategy": {
  "type": "probabilistic",
  "param": 0.5,
  "operation_strategies": [
    {
      "operation": "/health",
      "type": "probabilistic",
      "param": 0.0
    },
    {
      "operation": "/metrics",
      "type": "probabilistic",
      "param": 0.0
    }
  ]
}
}

```

3.2.5.4. 성능 Cryostat 확장

Performance Cryostat 확장은 현재 [기술 프리뷰](#) 기능 전용입니다.

Performance Cryostat 확장을 사용하면 `Go net/http/pprof` 엔드포인트를 사용할 수 있습니다. 일반적으로 개발자는 성능 프로필을 수집하고 서비스 관련 문제를 조사하는 데 사용됩니다.

OpenTelemetry Collector 사용자 정의 리소스 및 성능 조정 확장

```
config: |
```

```

extensions:
pprof:
  endpoint: localhost:1777 ①
  block_profile_fraction: 0 ②
  mutex_profile_fraction: 0 ③
  save_to_file: test.pprof ④

receivers:
otlp:
protocols:
http:

exporters:
otlp:

service:
extensions: [pprof]
pipelines:
traces:
  receivers: [otlp]
  exporters: [otlp]

```

1

이 확장이 수신 대기하는 끝점입니다. **localhost:**를 사용하여 모든 네트워크 인터페이스에서 사용할 수 있도록 로컬 또는 ":" 만 사용할 수 있습니다. 기본값은 **localhost:1777**입니다.

2

이벤트의 프로파일링을 위해 차단 이벤트의 일부를 설정합니다. 프로파일링을 비활성화하려면 이 값을 0 또는 음수 정수로 설정합니다. **runtime** 패키지에 대한 [설명서](#)를 참조하십시오. 기본값은 0입니다.

3

뮤지션 경합 이벤트의 일부를 프로파일링하도록 설정합니다. 프로파일링을 비활성화하려면 이 값을 0 또는 음수 정수로 설정합니다. **runtime** 패키지에 대한 [설명서](#)를 참조하십시오. 기본값은 0입니다.

4

CPU 프로필을 저장할 파일의 이름입니다. 프로파일링은 수집기가 시작될 때 시작됩니다. 프로파일링은 수집기가 종료될 때 파일에 저장됩니다.

3.2.5.5. 상태 점검 확장

상태 점검 확장은 현재 [기술 프리뷰](#) 기능 전용입니다.

상태 점검 확장에서는 **OpenTelemetry** 수집기의 상태를 확인하기 위한 **HTTP URL**을 제공합니다. 이 확장 기능을 **OpenShift**에서 활성 상태 및 준비 상태 프로브로 사용할 수 있습니다.

구성된 **Health Check** 확장이 포함된 **OpenTelemetry** 수집기 사용자 정의 리소스

```
config: |
  extensions:
    health_check:
      endpoint: "0.0.0.0:13133" ①
      tls: ②
        ca_file: "/path/to/ca.crt"
        cert_file: "/path/to/cert.crt"
        key_file: "/path/to/key.key"
      path: "/health/status" ③
      check_collector_pipeline: ④
      enabled: true ⑤
      interval: "5m" ⑥
      exporter_failure_threshold: 5 ⑦

  receivers:
    otlp:
      protocols:
        http:

  exporters:
    otlp:

  service:
    extensions: [health_check]
    pipelines:
      traces:
        receivers: [otlp]
        exporters: [otlp]
```

1

상태 점검 상태를 게시하기 위한 대상 IP 주소입니다. 기본값은 0.0.0.0:13133 입니다.

2

3

4

수집기 파이프라인 상태 점검 설정

5

수집기 파이프라인 상태 점검을 활성화합니다. 기본값은 `false`입니다.

6

실패 횟수를 확인하는 시간 간격입니다. 기본값은 `5m`입니다.

7

컨테이너가 여전히 정상으로 표시될 때까지 다수의 실패 임계값입니다. 기본값은 `5`입니다.

3.2.5.6. 메모리 Ballast 확장

Memory Ballast 확장은 현재 [기술 프리뷰](#) 기능 전용입니다.

Memory Ballast 확장을 사용하면 애플리케이션에서 프로세스에 대한 메모리 볼트를 구성할 수 있습니다.

구성된 **Memory Ballast** 확장을 사용하여 OpenTelemetry 수집기 사용자 정의 리소스

```
config: |
  extensions:
    memory_ballast:
      size_mib: 64 ①
      size_in_percentage: 20 ②

  receivers:
    otlp:
      protocols:
        http:

  exporters:
```

```
otlp:
```

```
service:
  extensions: [memory_ballast]
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [otlp]
```

1

메모리 블트 크기를 MiB로 설정합니다. 둘 다 지정된 경우 `size_in_percentage` 보다 우선 순위를 지정합니다.

2

총 메모리의 메모리 블타를 백분율로 설정합니다. 1-100. 컨테이너화된 및 물리적 호스트 환경을 지원합니다.

3.2.5.7. zPages 확장

zPages 확장은 현재 [기술 프리뷰](#) 기능 전용입니다.

zPages 확장은 **zPages**를 제공하는 확장을 위한 HTTP 끝점을 제공합니다. 엔드포인트에서 이 확장은 조정된 구성 요소를 디버깅하기 위한 라이브 데이터를 제공합니다. 모든 코어 내보내기 및 수신자는 일부 **zPages** 계측을 제공합니다.

zPages는 추적 또는 메트릭을 검사하기 위해 백엔드에 의존하지 않고도 프로세스 내 진단에 유용합니다.

구성된 **zPages** 확장이 포함된 **OpenTelemetry** 수집기 사용자 정의 리소스

```
config: |
extensions:
  zpages:
    endpoint: "localhost:55679" ①
receivers:
```

```

otlp:
  protocols:
    http:
  exporters:
    otlp:

service:
  extensions: [zpages]
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [otlp]

```

1

zPages를 제공하는 HTTP 끝점을 지정합니다. **localhost:**를 사용하여 모든 네트워크 인터페이스에서만 사용할 수 있도록 로컬로 또는 ":"를 사용할 수 있도록 합니다. 기본값은 **localhost:55679**입니다.

3.3. 대상 AL CRYOSTAT

대상 al Cryostat는 OpenTelemetry 수집기 인스턴스의 배포된 플랫에서 대상을 스크랩하는 OpenTelemetry Operator의 선택적 구성 요소입니다. 대상 al Cryostat는 Prometheus PodMonitor 및 ServiceMonitor CR(사용자 정의 리소스)과 통합됩니다. 대상이 활성화된 경우 OpenTelemetry Operator는 대상 al Cryostat 서비스에 연결하는 활성화된 Prometheus 수신자에 **http_sd_config** 필드를 추가합니다.

활성화된 대상 al Cryostat를 사용하는 OpenTelemetryCollector CR의 예

```

apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: observability
spec:
  mode: statefulset 1
  targetAllocator:
    enabled: true 2
    serviceAccount: 3
  prometheusCR:
    enabled: true 4
    scrapeInterval: 10s
  serviceMonitorSelector: 5

```

```

name: app1
podMonitorSelector: ⑥
  name: app2
config: |
receivers:
  prometheus: ⑦
    config:
      scrape_configs: []
processors:
exporters:
  debug:
service:
pipelines:
  metrics:
    receivers: [prometheus]
    processors: []
    exporters: [debug]

```

1

대상 **ai Cryostat**가 활성화된 경우 배포 모드를 상태 저장 세트로 설정해야 합니다.

2

대상 **ai Cryostat**를 활성화합니다. 기본값은 **false**입니다.

3

대상 **ai Cryostat** 배포의 서비스 계정 이름입니다. 서비스 계정에는 **ServiceMonitor,PodMonitor** 사용자 정의 리소스 및 클러스터의 기타 오브젝트를 가져오려면 RBAC가 있어야 스크랩된 메트릭에 라벨을 올바르게 설정해야 합니다. 기본 서비스 이름은 <**collector_name>-target****ai Cryostat**입니다.

4

Prometheus PodMonitor 및 **ServiceMonitor** 사용자 정의 리소스와의 통합을 활성화합니다.

5

Prometheus ServiceMonitor 사용자 정의 리소스의 라벨 선택기입니다. 비어 있는 경우 모든 서비스 모니터를 활성화합니다.

6

7

최소 빈 `scrape_config: []` 구성 옵션이 있는 **Prometheus** 수신자.

대상 **al Cryostat** 배포에서는 **Kubernetes API**를 사용하여 클러스터에서 관련 오브젝트를 가져오므로 사용자 지정 **RBAC** 구성이 필요합니다.

대상 **al Cryostat** 서비스 계정에 대한 **RBAC** 구성

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: otel-targetallocator
rules:
  - apiGroups: [""]
    resources:
      - services
      - pods
    verbs: ["get", "list", "watch"]
  - apiGroups: ["monitoring.coreos.com"]
    resources:
      - servicemonitors
      - podmonitors
    verbs: ["get", "list", "watch"]
  - apiGroups: ["discovery.k8s.io"]
    resources:
      - endpointslices
    verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: otel-targetallocator
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: otel-targetallocator
subjects:
  - kind: ServiceAccount
    name: otel-targetallocator ①
    namespace: observability ②
```

1

대상 **ai Cryostat** 서비스 계정의 이름입니다.

2

대상 서비스 계정의 네임스페이스입니다.

4장. 계측 구성



중요

OpenTelemetry 계측 삽입은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

Red Hat build of OpenTelemetry Operator는 계측 구성을 정의하는 CRD(사용자 정의 리소스 정의) 파일을 사용합니다.

4.1. OPENTELEMETRY 조정 구성 옵션

Red Hat build of OpenTelemetry는 **OpenTelemetry** 자동 복구 라이브러리를 워크로드에 삽입하고 구성할 수 있습니다. 현재 이 프로젝트는 **Go, Java, Node.js, Python, .NET 및 Apache HTTP Server(httpd)**의 계측 라이브러리 삽입을 지원합니다.

OpenTelemetry의 자동 복원은 프레임워크가 수동 코드 변경 없이 애플리케이션을 자동으로 수행하는 기능을 나타냅니다. 이를 통해 개발자와 관리자는 기존 코드베이스에 대한 최소한의 노력과 변경 사항을 통해 애플리케이션을 관찰할 수 있습니다.



중요

Red Hat build of OpenTelemetry Operator는 계측 라이브러리의 삽입 메커니즘만 지원하지만 계측 라이브러리 또는 업스트림 이미지는 지원하지 않습니다. 고객은 자체 계측 이미지를 빌드하거나 커뮤니티 이미지를 사용할 수 있습니다.

4.1.1. 계측 옵션

계측 옵션은 **OpenTelemetryCollector** 사용자 지정 리소스에 지정됩니다.

샘플 OpenTelemetryCollector 사용자 정의 리소스 파일

```

apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: java-instrumentation
spec:
  env:
    - name: OTEL_EXPORTER_OTLP_TIMEOUT
      value: "20"
  exporter:
    endpoint: http://production-collector.observability.svc.cluster.local:4317
  propagators:
    - w3c
  sampler:
    type: parentbased_traceidratio
    argument: "0.25"
  java:
    env:
      - name: OTEL_JAVAAGENT_DEBUG
        value: "true"

```

표 4.1. Operator에서 Instrumentation을 정의하는 데 사용하는 매개변수

매개변수	설명	값
env	모든 계측을 정의하는 공통 환경 변수입니다.	
exporter	내보내기 구성.	
propagators	전파자는 프로세스 간 컨텍스트 전파 구성을 정의합니다.	tracecontext,b3,b3multi,jaeger,ottrace,none
resource	리소스 속성 구성.	
sampler	샘플링 구성.	

매개변수	설명	값
apacheHttpd	Apache HTTP Server 조정 구성.	
dotnet	.NET 계측에 대한 구성입니다.	
go	Go 계측을 위한 구성입니다.	
java	Java 계측을 위한 구성입니다.	
nodejs	Node.js 계측 구성.	
python	Python 계측을 위한 구성입니다.	

4.1.2. Service Mesh에서 계측 CR 사용

Red Hat OpenShift Service Mesh와 함께 조정 사용자 정의 리소스(CR)를 사용하는 경우 **b3multi** 전파기를 사용해야 합니다.

4.1.2.1. Apache HTTP Server 자동 복구 구성

표 4.2. .spec.apacheHttpd 필드에 대한 매개변수

이름	설명	Default
attrs	Apache HTTP Server와 관련된 속성입니다.	
configPath	Apache HTTP Server 구성의 위치입니다.	/usr/local/apache2/conf
env	Apache HTTP Server와 관련된 환경 변수.	
image	Apache SDK 및 자동 복구가 포함된 컨테이너 이미지입니다.	

이름	설명	Default
resourceRequirements	컴퓨팅 리소스 요구 사항입니다.	
version	Apache HTTP Server 버전.	2.4

삽입을 활성화하는 PodSpec 주석

`instrumentation.opentelemetry.io/inject-apache-httpd: "true"`

4.1.2.2. .NET 자동 복원 구성

이름	설명
env	.NET 고유의 환경 변수입니다.
image	.NET SDK 및 자동 복구가 포함된 컨테이너 이미지입니다.
resourceRequirements	컴퓨팅 리소스 요구 사항입니다.

.NET 자동 복원의 경우 내보내기의 끝점이 4317로 설정된 경우 필요한 OTEL_EXPORTER_OTLP_ENDPOINT 환경 변수를 설정해야 합니다. .NET autoinstrumentation에서 기본적으로 http/proto를 사용하고 Telemetry 데이터를 4318 포트로 설정해야 합니다.

삽입을 활성화하는 PodSpec 주석

`instrumentation.opentelemetry.io/inject-dotnet: "true"`

4.1.2.3. Go 자동 복원 구성

이름	설명
env	Go 관련 환경 변수.
image	Go SDK 및 자동 복구가 포함된 컨테이너 이미지입니다.
resourceRequirements	컴퓨팅 리소스 요구 사항입니다.

삽입을 활성화하는 **PodSpec** 주석

```
instrumentation.opentelemetry.io/inject-go: "true"
```

OpenShift 클러스터의 Go 자동 복원에 필요한 추가 권한

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: otel-go-instrumentation-scc
allowHostDirVolumePlugin: true
allowPrivilegeEscalation: true
allowPrivilegedContainer: true
allowedCapabilities:
- "SYS_PTRACE"
fsGroup:
  type: RunAsAny
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
seccompProfiles:
```

```
- **  
supplementalGroups:  
type: RunAsAny
```

작은 정보

OpenShift 클러스터에서 Go 자동 복구에 대한 권한을 적용하는 CLI 명령은 다음과 같습니다.

```
$ oc adm policy add-scc-to-user otel-go-instrumentation-scc -z <service_account>
```

4.1.2.4. Java 자동 복원 구성

이름	설명
env	Java와 관련된 환경 변수.
image	Java SDK 및 자동 복구가 포함된 컨테이너 이미지입니다.
resourceRequirements	컴퓨팅 리소스 요구 사항입니다.

삽입을 활성화하는 PodSpec 주석

```
instrumentation.opentelemetry.io/inject-java: "true"
```

4.1.2.5. Node.js 자동 복원 구성

이름	설명
env	Node.js와 관련된 환경 변수.

이름	설명
image	Node.js SDK 및 자동 복원이 포함된 컨테이너 이미지입니다.
resourceRequirements	컴퓨팅 리소스 요구 사항입니다.

삽입을 활성화하는 **PodSpec** 주석

```
instrumentation.opentelemetry.io/inject-nodejs: "true"
instrumentation.opentelemetry.io/otel-go-auto-target-exe: "/path/to/container/executable"
```

Instrumentation.opentelemetry.io/otel-go-auto-target-exe 주석은 필요한
OTEL_GO_AUTO_TARGET_EXE 환경 변수의 값을 설정합니다.

4.1.2.6. Python 자동 복원 구성

이름	설명
env	Python과 관련된 환경 변수.
image	Python SDK 및 자동 복구가 포함된 컨테이너 이미지입니다.
resourceRequirements	컴퓨팅 리소스 요구 사항입니다.

Python 자동 복원의 경우 내보내기 사용자의 끝점이 **4317**로 설정된 경우
OTEL_EXPORTER_OTLP_ENDPOINT 환경 변수를 설정해야 합니다. **Python** 자동 복원은 기본적으로
http/proto 를 사용하며 **Telemetry** 데이터를 **4318** 포트로 설정해야 합니다.

삽입을 활성화하는 **PodSpec** 주석

```
instrumentation.opentelemetry.io/inject-python: "true"
```

4.1.2.7. OpenTelemetry SDK 변수 구성

Pod의 OpenTelemetry SDK 변수는 다음 주석을 사용하여 구성할 수 있습니다.

```
instrumentation.opentelemetry.io/inject-sdk: "true"
```

모든 주석은 다음 값을 허용합니다.

true

네임스페이스에서 **Instrumentation** 리소스를 삽입합니다.

false

장치를 삽입하지 않습니다.

instrumentation-name

현재 네임스페이스에서 삽입할 계측 리소스의 이름입니다.

other-namespace/instrumentation-name

다른 네임스페이스에서 삽입할 계측 리소스의 이름입니다.

4.1.2.8. 멀티컨테이너 Pod

조정은 Pod 사양에 따라 기본적으로 사용 가능한 첫 번째 컨테이너에서 실행됩니다. 경우에 따라 삽입을 위해 대상 컨테이너를 지정할 수도 있습니다.

Pod 주석

instrumentation.opentelemetry.io/container-names: "<container_1>,<container_2>"



참고

Go 자동 복원은 다중 컨테이너 자동 복구 삽입을 지원하지 않습니다.

5장. OPENTELEMETRY 수집기에 추적 및 메트릭 전송

OpenTelemetry의 Red Hat 빌드를 설정하고 사용하여 OpenTelemetry 수집기 또는 TempoStack에 추적을 보낼 수 있습니다.

사이드카 삽입을 사용하거나 사용하지 않고 추적과 메트릭을 OpenTelemetry 수집기로 보낼 수 있습니다.

5.1. 사이드카 삽입을 사용하여 추적 및 메트릭을 OPENTELEMETRY 수집기로 전송

사이드카 삽입을 사용하여 OpenTelemetry 수집기 인스턴스로 Telemetry 데이터를 전송할 수 있습니다.

Red Hat build of OpenTelemetry Operator를 사용하면 설계의 배포 워크로드 및 자동 구성에 사이드 카 삽입을 통해 Telemetry 데이터를 OpenTelemetry 수집기에 보낼 수 있습니다.

사전 요구 사항

- Red Hat OpenShift distributed tracing platform(Tempo)이 설치되고 TempoStack 인스턴스가 배포됩니다.
- 웹 콘솔 또는 OpenShift CLI(oc)를 통해 클러스터에 액세스할 수 있습니다.
 - cluster-admin 역할의 클러스터 관리자로 웹 콘솔에 로그인되어 있습니다.
 - cluster-admin 역할의 클러스터 관리자가 활성 OpenShift CLI(oc) 세션입니다.
 - Red Hat OpenShift Dedicated의 경우 dedicated-admin 역할의 계정이 있어야 합니다.

프로세스

1.

OpenTelemetry 수집기 인스턴스에 대한 프로젝트를 생성합니다.

```
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: observability
```

2.

서비스 계정을 생성합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: otel-collector-sidecar
  namespace: observability
```

3.

k8sattributes 및 **resourcedetection** 프로세서에 대한 서비스 계정에 권한을 부여합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: otel-collector
rules:
- apiGroups: [""]
  resources: ["pods", "namespaces", "infrastructures", "infrastructures/status"]
  verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: otel-collector
subjects:
- kind: ServiceAccount
  name: otel-collector-sidecar
  namespace: observability
roleRef:
  kind: ClusterRole
  name: otel-collector
  apiGroup: rbac.authorization.k8s.io
```

4.

OpenTelemetry Collector를 사이드카로 배포합니다.

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: observability
```

```

spec:
  serviceAccount: otel-collector-sidecar
  mode: sidecar
  config: |
    serviceAccount: otel-collector-sidecar
    receivers:
      otlp:
        protocols:
          grpc:
          http:
    processors:
      batch:
      memory_limiter:
        check_interval: 1s
        limit_percentage: 50
        spike_limit_percentage: 30
    resourcedetection:
      detectors: [openshift]
      timeout: 2s
    exporters:
      otlp:
        endpoint: "tempo-<example>-gateway:8090" ①
        tls:
          insecure: true
    service:
      pipelines:
        traces:
          receivers: [jaeger]
          processors: [memory_limiter, resourcedetection, batch]
          exporters: [otlp]

```

①

이는 Tempo Operator를 사용하여 <example> 배포된 TempoStack 인스턴스의 게이트웨이를 가리킵니다.

5.

otel-collector-sidecar 서비스 계정을 사용하여 배포를 생성합니다.

6.

Deployment 오브젝트에 sidecar.opentelemetry.io/inject: "true" 주석을 추가합니다. 이렇게 하면 워크로드에서 OpenTelemetry 수집기 인스턴스로 데이터를 보내는 데 필요한 모든 환경 변수가 삽입됩니다.

5.2. 사이드카 삽입 없이 OPENTELEMETRY 수집기에 추적 및 메트릭 전송

여러 환경 변수를 수동으로 설정해야 하는 사이드카 삽입 없이 OpenTelemetry 수집기 인스턴스로 원격 분석 데이터를 보낼 수 있습니다.

사전 요구 사항

- Red Hat OpenShift distributed tracing platform(Tempo)이 설치되고 TempoStack 인스턴스가 배포됩니다.
- 웹 콘솔 또는 OpenShift CLI(oc)를 통해 클러스터에 액세스할 수 있습니다.
 - cluster-admin 역할의 클러스터 관리자로 웹 콘솔에 로그인되어 있습니다.
 - cluster-admin 역할의 클러스터 관리자가 활성 OpenShift CLI(oc) 세션입니다.
 - Red Hat OpenShift Dedicated의 경우 dedicated-admin 역할의 계정이 있어야 합니다.

프로세스

1.

OpenTelemetry 수집기 인스턴스에 대한 프로젝트를 생성합니다.

```
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: observability
```

2.

서비스 계정을 생성합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: otel-collector-deployment
  namespace: observability
```

3.

k8sattributes 및 resourcedetection 프로세서에 대한 서비스 계정에 권한을 부여합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: otel-collector
rules:
```

```

- apiGroups: [ "", "config.openshift.io" ]
  resources: ["pods", "namespaces", "infrastructures", "infrastructures/status"]
  verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: otel-collector
subjects:
- kind: ServiceAccount
  name: otel-collector
  namespace: observability
roleRef:
  kind: ClusterRole
  name: otel-collector
  apiGroup: rbac.authorization.k8s.io

```

4.

OpenTelemetryCollector 사용자 정의 리소스를 사용하여 **OpenTelemetry** 수집기 인스턴스를 배포합니다.

```

apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: observability
spec:
  mode: deployment
  serviceAccount: otel-collector-deployment
  config: |
    receivers:
      jaeger:
        protocols:
          grpc:
            thrift_binary:
            thrift_compact:
            thrift_http:
      opencensus:
      otlp:
        protocols:
          grpc:
          http:
      zipkin:
    processors:
      batch:
      k8sattributes:
      memory_limiter:
        check_interval: 1s
        limit_percentage: 50
        spike_limit_percentage: 30
      resourcedetection:
        detectors: [openshift]
    exporters:
      otlp:

```

```

endpoint: "tempo-<example>-distributor:4317" ①
  tls:
    insecure: true
  service:
    pipelines:
      traces:
        receivers: [jaeger, opencensus, otlp, zipkin]
        processors: [memory_limiter, k8sattributes, resourcedetection, batch]
        exporters: [otlp]

```

1

이는 Tempo Operator를 사용하여 <example> 배포된 TempoStack 인스턴스의 게이트웨이를 가리킵니다.

5.

조정된 애플리케이션을 사용하여 컨테이너의 환경 변수를 설정합니다.

이름	설명	기본값
OTEL_SERVICE_NAME	service.name 리소스 속성 값 을 설정합니다.	""
OTEL_EXPORTER_OTL_P_ENDPOINT	선택적으로 지정된 포트 번호가 있는 모든 신호 유형의 기본 끝점 URL입니다.	https://localhost:4317
OTEL_EXPORTER_OTL_P_CERTIFICATE	gRPC 클라이언트의 TLS 자격 증명에 대한 인증서 파일의 경로입니다.	https://localhost:4317
OTEL_TRACES_SAMPLER	추적에 사용할 샘플입니다.	parentbased_always_on
OTEL_EXPORTER_OTL_P_PROTOCOL	OTLP 내보내기를 위한 전송 프로토콜입니다.	grpc
OTEL_EXPORTER_OTL_P_TIMEOUT	OTLP 내보내기가 각 배치 내보내기에 대해 대기하는 최대 시간 간격입니다.	10s
OTEL_EXPORTER_OTL_P_INSECURE	gRPC 요청에 대한 클라이언트 전송 보안을 비활성화합니다. HTTPS 스키마는 이를 덮어씁니다.	False

6장. 모니터링 스택에 메트릭을 전송하기 위한 구성

OpenTelemetry Collector CR(사용자 정의 리소스)을 구성하여 수집기의 파이프라인 지표 및 활성화된 **Prometheus** 내보내기를 스크랩하는 **Prometheus ServiceMonitor CR**을 생성할 수 있습니다.

Prometheus 내보내기를 사용한 **OpenTelemetry** 수집기 사용자 정의 리소스의 예

```
spec:
  mode: deployment
  observability:
    metrics:
      enableMetrics: true ①
  config: |
    exporters:
      prometheus:
        endpoint: 0.0.0.0:8889
        resource_to_telemetry_conversion:
          enabled: true # by default resource attributes are dropped
  service:
    telemetry:
      metrics:
        address: ":8888"
  pipelines:
    metrics:
      receivers: [otlp]
      exporters: [prometheus]
```

①

수집기의 내부 지표 끝점 및 **Prometheus** 내보내기 메트릭 끝점을 스크랩하도록 **Prometheus ServiceMonitor CR**을 생성하도록 **Operator**를 구성합니다. 지표는 **OpenShift** 모니터링 스택에 저장됩니다.

또는 수동으로 생성한 **Prometheus PodMonitor**는 **Prometheus** 스크랩 중에 추가된 중복된 라벨 제거와 같은 미세한 제어를 제공할 수 있습니다.

수집기 메트릭을 스크랩하도록 모니터링 스택을 구성하는 **PodMonitor** 사용자 정의 리소스의 예

```

apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: otel-collector
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: `<cr_name>-collector` ①
  podMetricsEndpoints:
    - port: metrics ②
    - port: promexporter ③
  relabelings:
    - action: labeldrop
      regex: pod
    - action: labeldrop
      regex: container
    - action: labeldrop
      regex: endpoint
  metricRelabelings:
    - action: labeldrop
      regex: instance
    - action: labeldrop
      regex: job

```

①

OpenTelemetry 수집기 사용자 정의 리소스의 이름입니다.

②

OpenTelemetry 수집기의 내부 메트릭 포트의 이름입니다. 이 포트 이름은 항상 **metrics**입니다.

③

OpenTelemetry 수집기의 **Prometheus** 내보내기 포트의 이름입니다.

7장. TEMPOSTACK으로 추적 전달

TempoStack으로 전달 추적을 구성하려면 **OpenTelemetry** 수집기를 배포하고 구성할 수 있습니다. 지정된 프로세서, 수신자 및 내보내기를 사용하여 배포 모드에서 **OpenTelemetry** 수집기를 배포할 수 있습니다. 다른 모드는 추가 리소스에 연결된 **OpenTelemetry** 수집기 설명서를 참조하십시오.

사전 요구 사항

- **Red Hat build of OpenTelemetry Operator**가 설치되어 있습니다.
- **Tempo Operator**가 설치되어 있습니다.
- **TempoStack**은 클러스터에 배포됩니다.

프로세스

1. **OpenTelemetry** 수집기의 서비스 계정을 생성합니다.

서비스 계정의 예

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: otel-collector-deployment
```

2. 서비스 계정에 대한 클러스터 역할을 생성합니다.

클러스터 역할의 예

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```

metadata:
  name: otel-collector
rules:
  1
  2
  - apiGroups: [ "", "config.openshift.io" ]
    resources: ["pods", "namespaces", "infrastructures", "infrastructures/status"]
    verbs: ["get", "watch", "list"]

```

1

k8sattributesprocessor에는 Pod 및 네임스페이스 리소스에 대한 권한이 필요합니다.

2

resourcedetectionprocessor에는 인프라 및 상태에 대한 권한이 필요합니다.

3.

클러스터 역할을 서비스 계정에 바인딩합니다.

클러스터 역할 바인딩 예

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: otel-collector
subjects:
  - kind: ServiceAccount
    name: otel-collector-deployment
    namespace: otel-collector-example
roleRef:
  kind: ClusterRole
  name: otel-collector
  apiGroup: rbac.authorization.k8s.io

```

4.

YAML 파일을 생성하여 OpenTelemetryCollector CR(사용자 정의 리소스)을 정의합니다.

OpenTelemetry 수집기 예

```

apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
spec:
  mode: deployment
  serviceAccount: otel-collector-deployment
  config: |
    receivers:
      jaeger:
        protocols:
          grpc:
          thrift_binary:
          thrift_compact:
          thrift_http:
      opencensus:
      otlp:
        protocols:
          grpc:
          http:
      zipkin:
    processors:
      batch:
      k8sattributes:
      memory_limiter:
        check_interval: 1s
        limit_percentage: 50
        spike_limit_percentage: 30
      resourcedetection:
        detectors: [openshift]
    exporters:
      otlp:
        endpoint: "tempo-simplest-distributor:4317" ①
        tls:
          insecure: true
      service:
        pipelines:
          traces:
            receivers: [jaeger, opencensus, otlp, zipkin] ②
            processors: [memory_limiter, k8sattributes, resourcedetection, batch]
            exporters: [otlp]

```

1

수집기 내보내기는 OTLP를 내보내도록 구성되어 있으며 이 예에서는 이미 생성된 Tempo 배포자 엔드포인트인 "tempo-simplest-distributor:4317" 을 가리킵니다.

2

수집기는 Jaeger 추적을 위한 수신기로 구성되며 OpenCensus는 OpenCensus 프로토콜을 통해 추적, Zipkin 프로토콜을 통한 Zipkin 추적 및 GRPC 프로토콜을 통해 OTLP 추적을 사용합니다.

작은 정보

tracegen 을 테스트로 배포할 수 있습니다.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: tracegen
spec:
  template:
    spec:
      containers:
        - name: tracegen
          image: ghcr.io/open-telemetry/opentelemetry-collector-contrib/tracegen:latest
          command:
            - "./tracegen"
          args:
            - --otlp-endpoint=otel-collector:4317
            - --otlp-insecure
            - --duration=30s
            - --workers=1
        restartPolicy: Never
        backoffLimit: 4
```

추가 리소스

- [OpenTelemetry 수집기 문서](#)
- [GitHub의 배포 예](#)

8장. OPENTELEMETRY 수집기 메트릭 구성

OpenTelemetry 수집기 인스턴스의 메트릭 및 경고를 활성화할 수 있습니다.

사전 요구 사항

- 클러스터에서 사용자 정의 프로젝트에 대한 모니터링이 활성화됩니다.

프로세스

- OpenTelemetry Collector 인스턴스의 메트릭을 활성화하려면 `spec.observability.metrics.enableMetrics` 필드를 `true`로 설정합니다.

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: <name>
spec:
  observability:
    metrics:
      enableMetrics: true
```

검증

웹 콘솔의 관리자 보기를 사용하여 구성이 성공했는지 확인할 수 있습니다.

- Observe → Targets로 이동하여 Source: User로 필터링한 다음 `opentelemetry-collector-<instance_name>` 형식의 ServiceMonitor에 Up 상태가 있는지 확인합니다.

추가 리소스

- [사용자 정의 프로젝트 모니터링 활성화](#)

9장. 여러 클러스터에서 관찰 데이터 수집

다중 클러스터 구성의 경우 원격 클러스터 각각에 하나의 **OpenTelemetry** 수집기 인스턴스를 생성한 다음 모든 **Telemetry** 데이터를 하나의 **OpenTelemetry** 수집기 인스턴스로 전달할 수 있습니다.

사전 요구 사항

- **Red Hat build of OpenTelemetry Operator**가 설치되어 있습니다.
- **Tempo Operator**가 설치되어 있습니다.
- **TempoStack** 인스턴스는 클러스터에 배포됩니다.
- 다음 마운트된 인증서: 발급자, 자체 서명 인증서, **CA** 발급자, 클라이언트 및 서버 인증서입니다. 이러한 인증서를 생성하려면 1단계를 참조하십시오.

프로세스

1. **OpenTelemetry** 수집기 인스턴스에 다음 인증서를 마운트하고 이미 마운트된 인증서를 건너뜁니다.

- a. **cert-manager Operator for Red Hat OpenShift**를 사용하여 인증서를 생성하는 발행자입니다.

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
```

- b. 자체 서명된 인증서입니다.

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ca
```

```

spec:
  isCA: true
  commonName: ca
  subject:
    organizations:
      - Organization # <your_organization_name>
  organizationalUnits:
    - Widgets
  secretName: ca-secret
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io

```

c.

CA 발행자.

```

apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: test-ca-issuer
spec:
  ca:
    secretName: ca-secret

```

d.

클라이언트 및 서버 인증서입니다.

```

apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: server
spec:
  secretName: server-tls
  isCA: false
  usages:
    - server auth
    - client auth
  dnsNames:
    - "otel.observability.svc.cluster.local" ①
  issuerRef:
    name: ca-issuer
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: client
spec:
  secretName: client-tls
  isCA: false

```

```

usages:
- server auth
- client auth
dnsNames:
- "otel.observability.svc.cluster.local" 2
issuerRef:
  name: ca-issuer

```

1

OpenTelemetry 수집기 인스턴스의 솔버에 매핑될 정확한 DNS 이름 목록입니다.

2

클라이언트 **OpenTelemetry** 수집기 인스턴스에서 솔버에 매핑될 정확한 DNS 이름 목록입니다.

2.

OpenTelemetry 수집기 인스턴스에 대한 서비스 계정을 생성합니다.

서비스 계정의 예

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: otel-collector-deployment

```

3.

서비스 계정에 대한 클러스터 역할을 생성합니다.

클러스터 역할의 예

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: otel-collector
rules:
1
2

```

```

- apiGroups: [ "", "config.openshift.io" ]
  resources: ["pods", "namespaces", "infrastructures", "infrastructures/status"]
  verbs: ["get", "watch", "list"]

```

1

k8sattributesprocessor에는 Pod 및 네임스페이스 리소스에 대한 권한이 필요합니다.

2

resourcedetectionprocessor에는 인프라 및 상태에 대한 권한이 필요합니다.

4.

클러스터 역할을 서비스 계정에 바인딩합니다.

클러스터 역할 바인딩 예

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: otel-collector
subjects:
- kind: ServiceAccount
  name: otel-collector-deployment
  namespace: otel-collector-<example>
roleRef:
  kind: ClusterRole
  name: otel-collector
  apiGroup: rbac.authorization.k8s.io

```

5.

YAML 파일을 생성하여 엣지 클러스터에서 **OpenTelemetryCollector CR**(사용자 정의 리소스)을 정의합니다.

엣지 클러스터에 대한 **OpenTelemetryCollector** 사용자 정의 리소스의 예

```

apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: otel-collector-<example>
spec:
  mode: daemonset
  serviceAccount: otel-collector-deployment
  config: |
    receivers:
      jaeger:
        protocols:
          grpc:
            thrift_binary:
            thrift_compact:
            thrift_http:
        opencensus:
        otlp:
          protocols:
            grpc:
            http:
        zipkin:
    processors:
      batch:
      k8sattributes:
      memory_limiter:
        check_interval: 1s
        limit_percentage: 50
        spike_limit_percentage: 30
    resourcedetection:
      detectors: [openshift]
    exporters:
      otlphttp:
        endpoint: https://observability-cluster.com:443 ①
        tls:
          insecure: false
          cert_file: /certs/server.crt
          key_file: /certs/server.key
          ca_file: /certs/ca.crt
    service:
      pipelines:
        traces:
          receivers: [jaeger, opencensus, otlp, zipkin]
          processors: [memory_limiter, k8sattributes, resourcedetection, batch]
          exporters: [otlp]
    volumes:
      - name: otel-certs
        secret:
          name: otel-certs
    volumeMounts:
      - name: otel-certs
        mountPath: /certs

```

1

수집기 내보내기는 **OTLP HTTP**를 내보내도록 구성되며 중앙 클러스터에서 **OpenTelemetry** 수집기를 가리킵니다.

6.

YAML 파일을 생성하여 중앙 클러스터에서 **OpenTelemetryCollector CR**(사용자 정의 리소스)을 정의합니다.

중앙 클러스터에 대한 **OpenTelemetryCollector** 사용자 정의 리소스의 예

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otlp-receiver
  namespace: observability
spec:
  mode: "deployment"
  ingress:
    type: route
    route:
      termination: "passthrough"
  config: |
    receivers:
      otlp:
        protocols:
          http:
            tls: ①
              cert_file: /certs/server.crt
              key_file: /certs/server.key
              client_ca_file: /certs/ca.crt
    exporters:
      logging:
      otlp:
        endpoint: "tempo-<simplest>-distributor:4317" ②
        tls:
          insecure: true
    service:
      pipelines:
        traces:
          receivers: [otlp]
          processors: []
          exporters: [otlp]
    volumes:
      - name: otel-certs
        secret:
          name: otel-certs
```

```
volumeMounts:  
- name: otel-certs  
  mountPath: /certs
```

1

수집기 수신자는 첫 번째 단계에 나열된 인증서가 필요합니다.

2

수집기 내보내기는 OTLP를 내보내도록 구성되어 있으며 **Tempo** 배포자 끝점을 가리킵니다. 이 예제에서는 "tempo-simplest-distributor:4317"이며 이미 생성되었습니다.

10장. RED HAT BUILD OF OPENTELEMETRY 문제 해결

OpenTelemetry 수집기는 상태를 측정하고 데이터 수집 문제를 조사하는 여러 방법을 제공합니다.

10.1. OPENTELEMETRY 수집기 로그 가져오기

다음과 같이 OpenTelemetry 수집기의 로그를 가져올 수 있습니다.

프로세스

1.

OpenTelemetryCollector 사용자 정의 리소스(CR)에서 관련 로그 수준을 설정합니다.

```
config: |
  service: |
    telemetry: |
      logs:
        level: debug ①
```

①

수집기의 로그 수준입니다. 지원되는 값에는 **info**, **warn**, **error** 또는 **debug** 가 있습니다. 기본값은 **info** 입니다.

2.

oc logs 명령 또는 웹 콘솔을 사용하여 로그를 검색합니다.

10.2. 메트릭 노출

OpenTelemetry 수집기는 처리된 데이터 볼륨에 대한 지표를 표시합니다. 다음 메트릭은 기간용이지만 메트릭 및 로그 신호에 대해 유사한 메트릭이 노출됩니다.

otelcol_receiver_accepted_spans

파이프라인에 성공적으로 푸시된 기간 수입니다.

otelcol_receiver_refused_spans

파이프라인에 푸시할 수 없는 기간 수입니다.

`otelcol_exporter_sent_spans`

대상에 성공적으로 전송된 기간 수입니다.

`otelcol_exporter_enqueue_failed_spans`

전송 대기열에 범위를 추가하지 못했습니다.

Operator는 메트릭 끝점을 스크랩하는 데 사용할 수 있는 <cr_name>-collector-monitoring 원격 분석 서비스를 생성합니다.

프로세스

1.

OpenTelemetryCollector 사용자 정의 리소스에 다음 행을 추가하여 **Telemetry** 서비스를 활성화합니다.

```
config: |
  service:
    telemetry:
      metrics:
        address: ":8888" ①
```

①

내부 수집기 지표가 노출되는 주소입니다. 기본값은 :8888입니다.

1.

port-forwarding Collector Pod를 사용하는 다음 명령을 실행하여 지표를 검색합니다.

```
$ oc port-forward <collector_pod>
```

2.

<http://localhost:8888/metrics> 의 메트릭 끝점에 액세스합니다.

10.3. 디버그 내보내기

수집된 데이터를 표준 출력으로 내보내도록 디버그 내보내기를 구성할 수 있습니다.

프로세스

1.

다음과 같이 **OpenTelemetryCollector** 사용자 정의 리소스를 구성합니다.

```
config: |
  exporters:
    debug:
      verbosity: detailed
  service:
    pipelines:
      traces:
        exporters: [debug]
      metrics:
        exporters: [debug]
      logs:
        exporters: [debug]
```

2.

oc logs 명령 또는 웹 콘솔을 사용하여 로그를 표준 출력으로 내보냅니다.

11장. 분산 추적 플랫폼(JAEGER)에서 OPENTELEMETRY의 RED HAT 빌드로 마이그레이션



중요

Red Hat OpenShift distributed tracing platform(Jaeger)은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 **OpenShift Container Platform**에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 **OpenShift Container Platform** 릴리스 노트에서 더 이상 사용되지 않고 삭제된 기능 섹션을 참조하십시오.

애플리케이션에 **Red Hat OpenShift distributed tracing Platform(Jaeger)**을 이미 사용하고 있는 경우 **OpenTelemetry** 오픈 소스 프로젝트를 기반으로 하는 **OpenTelemetry**의 **Red Hat** 빌드로 마이그레이션할 수 있습니다.

Red Hat build of OpenTelemetry는 분산 시스템에서 관찰 기능을 용이하게 하는 API, 라이브러리, 에이전트 및 계측을 제공합니다. **OpenTelemetry**의 **Red Hat** 빌드의 **OpenTelemetry** 수집기는 **Jaeger** 프로토콜을 수집할 수 있으므로 애플리케이션의 **SDK**를 변경할 필요가 없습니다.

분산 추적 플랫폼(**Jaeger**)에서 **OpenTelemetry** 수집기로 **Red Hat** 빌드로 마이그레이션하려면 **OpenTelemetry** 수집기 및 애플리케이션이 추적을 원활하게 보고하도록 구성해야 합니다. 사이드카 및 사이드카 없는 배포를 마이그레이션할 수 있습니다.

11.1. 분산 추적 플랫폼(JAEGER)에서 사이드카를 사용하여 OPENTELEMETRY의 RED HAT 빌드로 마이그레이션

Red Hat build of OpenTelemetry Operator는 배포 워크로드에 사이드카 삽입을 지원하므로 분산 추적 플랫폼(**Jaeger**) 사이드카에서 **Red Hat** 빌드의 **OpenTelemetry** 사이드카로 마이그레이션할 수 있습니다.

사전 요구 사항

-

Red Hat OpenShift 분산 추적 플랫폼(**Jaeger**)은 클러스터에서 사용됩니다.

-

Red Hat build of OpenTelemetry가 설치되어 있습니다.

프로세스

1.

OpenTelemetry 수집기를 사이드카로 구성합니다.

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: <otel-collector-namespace>
spec:
  mode: sidecar
  config: |
    receivers:
      jaeger:
        protocols:
          grpc:
            thrift_binary:
            thrift_compact:
            thrift_http:
    processors:
      batch:
        memory_limiter:
          check_interval: 1s
          limit_percentage: 50
          spike_limit_percentage: 30
    resourcedetection:
      detectors: [openshift]
      timeout: 2s
    exporters:
      otlp:
        endpoint: "tempo-<example>-gateway:8090" ①
        tls:
          insecure: true
    service:
      pipelines:
        traces:
          receivers: [jaeger]
          processors: [memory_limiter, resourcedetection, batch]
          exporters: [otlp]
```

①

이 끝점은 **Tempo Operator**를 사용하여 배포된 **TempoStack** 인스턴스의 게이트웨이를 **<example>** 가리킵니다.

2.

애플리케이션 실행을 위한 서비스 계정을 생성합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: otel-collector-sidecar
```

3.

일부 프로세서에 필요한 권한에 대한 클러스터 역할을 생성합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: otel-collector-sidecar
rules:
  ①
  - apiGroups: ["config.openshift.io"]
    resources: ["infrastructures", "infrastructures/status"]
    verbs: ["get", "watch", "list"]
```

①

resourcedetectionprocessor에는 인프라 및 인프라/상태에 대한 권한이 필요합니다.

4.

ClusterRoleBinding 을 생성하여 서비스 계정에 대한 권한을 설정합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: otel-collector-sidecar
subjects:
  - kind: ServiceAccount
    name: otel-collector-deployment
    namespace: otel-collector-example
roleRef:
  kind: ClusterRole
  name: otel-collector
  apiGroup: rbac.authorization.k8s.io
```

5.

OpenTelemetry Collector를 사이드카로 배포합니다.

6.

Deployment 오브젝트에서 "sidecar.jaegertracing.io/inject": "true" 주석을 제거하여 애플리케이션에서 삽입된 Jaeger 에이전트를 제거합니다.

7.

Deployment 오브젝트의 `.spec.template.metadata.annotations` 필드에 `sidecar.opentelemetry.io/inject: "true"` 주석을 추가하여 **OpenTelemetry** 사이드카 자동 삽입을 활성화합니다.

8.

생성된 서비스 계정을 사용하여 애플리케이션이 배포되면 프로세서가 올바른 정보를 가져와서 추적에 추가할 수 있습니다.

11.2. 분산 추적 플랫폼(JAEGER)에서 사이드카 없이 OPENTELEMETRY의 RED HAT 빌드로 마이그레이션

사이드카 배포 없이 분산 추적 플랫폼(Jaeger)에서 **OpenTelemetry**의 Red Hat 빌드로 마이그레이션 할 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift** 분산 추적 플랫폼(Jaeger)은 클러스터에서 사용됩니다.
- **Red Hat build of OpenTelemetry**가 설치되어 있습니다.

프로세스

1.

OpenTelemetry 수집기 배포를 구성합니다.

2.

OpenTelemetry 수집기를 배포할 프로젝트를 생성합니다.

```
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: observability
```

3.

OpenTelemetry 수집기 인스턴스를 실행하기 위한 서비스 계정을 생성합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
name: otel-collector-deployment
namespace: observability
```

4.

프로세서에 필요한 권한을 설정하는 클러스터 역할을 생성합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: otel-collector
rules:
  1
  2
  - apiGroups: [""]
    resources: ["pods", "namespaces", "infrastructures", "infrastructures/status"]
    verbs: ["get", "watch", "list"]
```

1

`k8sattributesprocessor`에는 Pod 및 namespaces 리소스에 대한 권한이 필요합니다.

2

`resourcedetectionprocessor`에는 infrastructures 및 infrastructures/status에 대한 권한이 필요합니다.

5.

ClusterRoleBinding을 생성하여 서비스 계정에 대한 권한을 설정합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: otel-collector
subjects:
  - kind: ServiceAccount
    name: otel-collector-deployment
    namespace: observability
roleRef:
  kind: ClusterRole
  name: otel-collector
  apiGroup: rbac.authorization.k8s.io
```

6.

OpenTelemetry 수집기 인스턴스를 생성합니다.



참고

이 수집기는 추적을 **TempoStack** 인스턴스로 내보냅니다. **Red Hat Tempo Operator**를 사용하여 **TempoStack** 인스턴스를 생성하고 여기에 올바른 끝점을 배치해야 합니다.

```

apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: observability
spec:
  mode: deployment
  serviceAccount: otel-collector-deployment
  config: |
    receivers:
      jaeger:
        protocols:
          grpc:
            thrift_binary:
            thrift_compact:
            thrift_http:
    processors:
      batch:
      k8sattributes:
      memory_limiter:
        check_interval: 1s
        limit_percentage: 50
        spike_limit_percentage: 30
    resourcedetection:
      detectors: [openshift]
    exporters:
      otlp:
        endpoint: "tempo-example-gateway:8090"
        tls:
          insecure: true
    service:
      pipelines:
        traces:
          receivers: [jaeger]
          processors: [memory_limiter, k8sattributes, resourcedetection, batch]
          exporters: [otlp]
```

7.

추적 끝점을 **OpenTelemetry Operator**를 가리킵니다.

8.

애플리케이션에서 직접 추적을 **Jaeger**로 내보내는 경우 **API** 끝점을 **Jaeger** 끝점에서 **OpenTelemetry** 수집기 끝점으로 변경합니다.

Goland에서 **jaegerexporter** 를 사용하여 추적 내보내기 예

```
exp, err := jaeger.New(jaeger.WithCollectorEndpoint(jaeger.WithEndpoint(url))) ①
```

①

URL은 OpenTelemetry 수집기 API 끝점을 가리킵니다.

12장. OPENTELEMETRY RED HAT 빌드 업데이트

버전 업그레이드의 경우 **Red Hat build of OpenTelemetry Operator**는 OLM(Operator Lifecycle Manager)을 사용하여 클러스터에서 **Operator**의 설치, 업그레이드 및 역할 기반 액세스 제어(RBAC)를 제어합니다.

OLM은 기본적으로 **OpenShift Container Platform**에서 실행됩니다. 사용 가능한 **Operator** 및 설치된 **Operator**의 업그레이드에 대한 OLM 쿼리입니다.

Red Hat build of OpenTelemetry Operator가 새 버전으로 업그레이드되면 관리하는 **OpenTelemetry Collector** 인스턴스를 검사하고 **Operator**의 새 버전에 해당하는 버전으로 업그레이드합니다.

12.1. 추가 리소스

- [Operator Lifecycle Manager 개념 및 리소스](#)
- [설치된 Operator 업데이트](#)

13장. OPENTELEMETRY RED HAT 빌드 제거

OpenShift Container Platform 클러스터에서 **OpenTelemetry**의 **Red Hat** 빌드를 제거하는 단계는 다음과 같습니다.

1. **OpenTelemetry Pod**의 모든 **Red Hat** 빌드를 종료합니다.
2. **OpenTelemetryCollector** 인스턴스를 제거합니다.
3. **OpenTelemetry Operator**의 **Red Hat** 빌드를 제거합니다.

13.1. 웹 콘솔을 사용하여 OPENTELEMETRY 수집기 인스턴스 제거

웹 콘솔의 관리자 보기에서 **OpenTelemetry** 수집기 인스턴스를 제거할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 클러스터 관리자로 웹 콘솔에 로그인되어 있습니다.
- **Red Hat OpenShift Dedicated**의 경우 **dedicated-admin** 역할의 계정을 사용하여 로그인해야 합니다.

프로세스

1. **Operator** → 설치된 **Operator** → **OpenTelemetry Operator** → **OpenTelemetry Instrumentation** 또는 **OpenTelemetryCollector**로 이동합니다.
2. 관련 인스턴스를 제거하려면
 - ⋮
 - **Delete ...** → **Delete**를 선택합니다.

3.

선택 사항: OpenTelemetry Operator의 Red Hat 빌드를 제거합니다.

13.2. CLI를 사용하여 OPENTELEMETRY 수집기 인스턴스 제거

명령줄에서 OpenTelemetry 수집기 인스턴스를 제거할 수 있습니다.

사전 요구 사항

- cluster-admin 역할의 클러스터 관리자가 활성 OpenShift CLI(oc) 세션입니다.

작은 정보

- OpenShift CLI(oc) 버전이 최신 버전인지 확인하고 OpenShift Container Platform 버전과 일치하는지 확인합니다.
 - oc login 을 실행합니다.

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

프로세스

1.

다음 명령을 실행하여 OpenTelemetry 수집기 인스턴스의 이름을 가져옵니다.

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

2.

다음 명령을 실행하여 OpenTelemetry 수집기 인스턴스를 제거합니다.

```
$ oc delete opentelemetrycollectors <opentelemetry_instance_name> -n <project_of_opentelemetry_instance>
```

3.

선택 사항: OpenTelemetry Operator의 Red Hat 빌드를 제거합니다.

검증

-

OpenTelemetry 수집기 인스턴스가 성공적으로 제거되었는지 확인하려면 `oc get deployment`를 다시 실행합니다.

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

13.3. 추가 리소스

- [클러스터에서 Operator 삭제](#)
- [OpenShift CLI 시작하기](#)