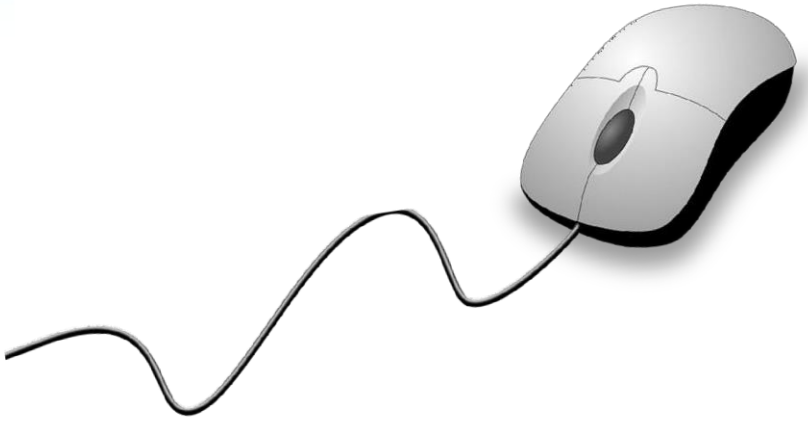
공개SW 솔루션 설치 & 활용 가이드

미들웨어 > 분산시스템SW



JAEGER

제대로 배워보자

How to Use Open Source Software

Open Source Software Installation & Application Guide

OPENUP

오픈소스 소프트웨어 통합지원센터

Open Source Software Support Center

CONTENTS

1. 개요

JAEGER

소개	• Dapper와 OpenZipkin을 기반으로 Uber Technologies에서 공개SW로 출시한 분산추적 시스템 • 마이크로 서비스 기반 분산시스템을 모니터링하고 문제를 해결하는 데 사용		
주요기능	• Distributed context propagation(전파) • Distributed transaction monitoring • Root cause analysis • Service dependency analysis • Performance/latency optimization		
대분류	• 미들웨어	**소분류**	• 분산시스템 SW
라이선스형태	• Apache License v2.0	**사전설치 솔루션**	• Go 1.9
		버전	• 1.15.1 (2019년 11월 기준)
특징	• High Scalability • OpenTracing Standard 준수 • 다수의 Storage지원 : NoSQL DB/other DB/In-memory Storage • React와 같은 Morden한 기술을 사용한 WEB UI		
개발회사/커뮤니티	• https://medium.com/jaegertracing • https://gitter.im/jaegertracing • https://twitter.com/JaegerTracing • https://github.com/jaegertracing/jaeger		
공식 홈페이지	• https://www.jaegertracing.io/		

2. 기능요약

- Apache 주요 기능

Span	• 작업명, 작업 시간 및 기간있는 논리적 작업 단위를 나타탐 • 중첩되어 있는 인과 관계를 모델링 정렬 가능
Trace	• 시스템을 통한 데이터/실행 경로이며, span의 방향성 비순환 그래프이다.
Client	• 클라이언트는 OpenTracing API로 수동 또는 Flask, Dropwizard, gRPC등과 같이 이미 OpenTracing 과 통합된 다양한 기존 오픈소스프레임 워크와 함께 분산 추적을 위한 응용프로그램을 계측하는데 사용할 수 있음.
Agent	• UDP를 통해 전송된 span을 수신 대기하는 네트워크 데몬으로, 이를 일괄 처리하여 수집기로 보냄 • 인프라 구성 요소로 모든 호스트에 배포되도록 설계
Collector	• Agent로 부터 추적 수신하여 파이프 라인을 통해 실행 • 현재 파이프 라인은 추적의 유효성을 검사하고 색인을 생성 변환 후 저장

3. 실행환경

- 하드웨어 제약이 거의 없음
- OS 플랫폼 종류에 따른 지원
  - macOS, Linux 및 Windows
- 요구사항
  - ⋆ 8 GB of RAM (Preferebly 16 GB) 50 GB Disk space

4. 설치 및 실행

1. Installing the Operator on Kubernetes

- 아래 명령어는 observability 네임스페이스를 작성하고 jaeger 운영자를 설치하는 화면

```
kubectl create namespace observability #<1>

kubectl create -f
https://raw.githubusercontent.com/jaegertracing/jaegeroperator/master/deploy/crds/jaegertracing.io_jaegers_crd.yaml # <2>

kubectl create -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/master/deploy/service_account.yaml

kubectl create -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/master/deploy/role.yaml

kubectl create -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/master/deploy/role_binding.yaml

kubectl create -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/master/deploy/operator.yaml
```

- #<1> 배포파일에서 기본적으로 사용되는 네임스페이스를 만듭니다.
- #<2> 이것은 apiVersion: jaegertracing.io/v1위한 "Custom Resource Definition" install

4. 설치 및 실행

1. Installing the Operator on Kubernetes

- jaeger-operator 사용 가능한 배포가 있어야 하며, 다음 명령을 실행하여 볼수 있습니다.

```
$ kubectl get deployment jaeger-operator -n observability
```

```
NAME              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
jaeger-operator   1         1         1            1           48s
```

4. 설치 및 실행

2. Installing the Operator on OKD/Openshift

- 이전 장표는 OKD 또는 OpenShift에 operator를 설치하는데 사용

RBAC(역활기반 액세스 제어)규칙, 사용자 정의 자원 및 운영자를 설치할때 권한있는 사용자로 로그인 했는지 확인

```
oc login -u <privileged user> oc new-project observability # <1>

oc create -f https://raw.githubusercontent.com/jaegertracing/jaeger-  o
perator/master/deploy/crds/jaegertracing.io_jaegers_crd.yaml # <2>

 oc create -f https://raw.githubusercontent.com/jaegertracing/jaeger-
operator/master/deploy/service_account.yaml

oc create -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/master/deploy/role.yaml

oc create -f https://raw.githubusercontent.com/jaegertracing/jaeger-
operator/master/deploy/role_binding.yaml

oc create -f  https://raw.githubusercontent.com/jaegertracing/jaeger-operator/master/deploy/operator.yaml
```

4. 설치 및 실행

2. Installing the Operator on OKD/Openshift

•operator가 설치되면, 개별 jaeger 인스턴스를 설치할 수 있는 사용자에게 "jaeger-operator" 권한을 승인. developer 사용자에게 jaeger-operator를 바인딩 하는 예제

```
oc create \

 rolebinding developer-jaeger-operator \

    --role=jaeger-operator \
    --user=developer
```

4. 설치 및 실행

3. Quick Start – Deploying the AllinOne image

•jaeger 인스턴스를 만드는 가장 간단한 방법은 YAML 파일을 만드는 것

단일포드를 사용하는 올인원 이미지를 설치하는 가장 쉬운 방법

```
apiVersion: jaegertracing.io/v1

 kind: Jaeger

metadata:
   name: simplest
```

- 다음으로 YAML파일을 사용

```
kubectl apply -f simplest.yaml
```

- jaeger 오브젝트 리스트하여 생성된 인스턴스를 확인

 # 빠른 데모 및 개발 목적에 적합한 올인원 인스턴스 jaeger가 출시 예정

```
$ kubectl get jaegers

 NAME      CREATED AT

simplest   28s
```

4. 설치 및 실행

3. Quick Start – Deploying the AllinOne image

- 포드 이름을 가져오려면 "simplest" jaeger 인스턴스에 속하는 포드를 쿼리

```
$ kubectl get pods -l app.kubernetes.io/instance=simplest
```

```
NAME                          READY   STATUS    RESTARTS   AGE
simplest-6499bb6cdd-kqx75     1/1     Running   0          2m
```

- 이전 예제에서 얻은 포드 이름을 사용하여 포드에서 직접 인스턴스에 속하는 모든 포드 로그를 쿼리할 수 있다.

```
$ kubectl logs -l app.kubernetes.io/instance=simplest ...

{"level":"info","ts":1535385688.0951214,"caller":"healthcheck/handler.go:133",
 "msg":"Health Check state change","status":"ready"}
```

4. 설치 및 실행

4. 배포전략

JAEGER

• jaeger 인스턴스를 작성하면 전략과 연관 됩니다.
이 전략은 사용자 지정 리소스 파일에 정의되어 있으며 jaeger 백엔드에 사용될 아키텍처를 결정합니다. 기본 전략은 " allinOne" 입니다. 그 외로 "production" 및 "streaming" 있습니다.

• AllInOne(default) strategy
이 전략은 개발, 테스트 및 데모 목적

• Production strategy
추적 데이터의 장기 저장이 중요하고 확장 가능한 가용성이 높은 아키텍처가 필요한 운영 환경을 대상으로 함. 따라서 각 백엔드 구성요소가 별도로 배포됨.

4. 설치 및 실행

5. production strategy

- 에이전트는 인스트루먼트 된 애플리케이션의 사이드카 또는 대몬셋으로 동작

쿼리 및 수집기 서비스는 Cassandra 또는 Elasticsearch 구성되어짐

각 구성 요소는 여러 인스턴스 성능 및 복원력 목적에 따라 프로비저닝

예) 주요 추가 요구사항에 대한 스토리지 유형 및 옵션의 세부사항 정의

```
storage:
     type: elasticsearch
     options:
        es:
           server-urls: http://elasticsearch:9200
```

4. 설치 및 실행

6. Streaming strategy

•production 수집기와 백엔드 스토리지(Cassandra 또는 Elasticsearch)사이에 효과적으로 위치한 스트리밍 기능을 제공

부하가 높을 때 백엔드 스토리지의 부담을 줄이는 이점을 제공

다른 추적 후 처리 기능을 통해 스트리밍 플랫폼(Kafka)에서 직접 실시간 데이터를 활용

collector 컴포넌트와 ingester 컴포넌트로 구성된 kafka 플랫폼에 액세스 하기위한 세부 정보를 제공해야 함

4. 설치 및 실행

4.6 Streaming strategy


```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
   name: simple-streaming
spec:
   strategy: streaming
   collector:
      options:
         kafka: # <1>
            producer:
               topic: jaeger-spans
               brokers: my-cluster-kafka-brokers.kafka:9092
   ingester:
      options:
         kafka: # <1>
           consumer:
              topic: jaeger-spans
              brokers: my-cluster-kafka-brokers.kafka:9092
         ingester:
            deadlockInterval: 0 # <2>
   storage:
      type: elasticsearch
      options:
         es:
            server-urls: http://elasticsearch:9200
```

4. 설치 및 실행

4.7 all-in-one instance 생성 방법


```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
   name: my-jaeger
spec:
   strategy: allInOne # <1>
   allInOne:
      image: jaegertracing/all-in-one:latest # <2>
      options: # <3>
         log-level: debug # <4>
   storage:
      type: memory # <5>
      options: # <6>
         memory: # <7>
            max-traces: 100000
      ingress:
         enabled: false # <8>
      agent:
         strategy: DaemonSet # <9>
      annotations:
         scheduler.alpha.kubernetes.io/critical-pod: "" # <10>
```

4. 설치 및 실행

7. all-in-one instance 생성 방법 – 구문별 설명

- <1> 기본 전략은 " allinone"
- <2> 일반 docker에서 사용할 이미지
- <3> --help 에서 사용 가능한 모든 옵션에 대해 참조할것
- <4> 해당 옵션은 바이너리에 전달할때 사용
- <5>디폴트 값이 memory, 그러나 Cassandra, Elasticsearch, Kafka 될 수도 있음
- <6> 모든 스토리지 관련 옵션 입력
- <8> 쿼리 서비스에 대한 수신 객체가 생성
- <9> DaemonSet으로 지정하면, operator가 agent를 daemonset으로 배포
- <10> 모든 배포에 적용할 주석을 정의

4. 설치 및 실행

8. Configuring the custom resource – Storage options

- Cassandra storage

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
   name: cassandra-without-create-schema
spec:
   strategy: allInOne
   storage:
      type: cassandra
      cassandraCreateSchema:
         enabled: false # <1>
```

4. 설치 및 실행

8. Configuring the custom resource – Storage options

- Elasticsearch storage – External Elasticsearch

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
   name: simple-prod
spec:
   strategy: production
   storage:
      type: elasticsearch # <1>
   options:
      es:
         server-urls: https://elasticsearch.default.svc:9200 # <2>
         tls: # <3>
            ca: /es/certificates/root-ca.pem
   secretName: jaeger-secret # <4>
volumeMounts: # <5>
   - name: certificates  mountPat
     h: /es/certificates/  readOn
     ly: true
volumes:
   - name: certificates
     secret:
        secretName: quickstart-es-http-certs-public
```

4. 설치 및 실행

9. Elasticsearch index cleaner job

- 기본적으로 elasticsearch는 old trace를 정리하기 위해서 cron을 사용한다.

```
storage:
   type: elasticsearch
   esIndexCleaner:
      enabled: true // turn the cron job deployment on and off  numberOf
      Days: 7 // number of days to wait before deleting a record
      schedule: "55 23 * * *" // cron expression for it to run
```

4. 설치 및 실행

9. Elasticsearch rollover

- 스토리지를 준비하는 초기화 작업과 인덱스 관리를 위한 2개의 cron 작업이 필요

```
storage:
   type: elasticsearch
   options:
      es:
         use-aliases: true
   esRollover:
      enabled: true // turn the cron job deployment on and off
      conditions: "{\"max_age\": \"2d\"}" // conditions when to rollover to a new index
      readTTL: 7d // how long should be old data available for reading
      schedule: "55 23 * * *" // cron expression for it to run
```

4. 설치 및 실행

10. Deriving dependencies

•dependencies를 도출하기 위한 프로세스는 스토리지에서 spans을 수집하고, 서비스간 링크를 분석한 후 나중에 UI에 표시하기 위함.

해당 작업은 Production 전략과 Cassandra 혹은 elasticsearch에서만 사용 가능

```
storage:
   type: elasticsearch
   dependencies:
      enabled: true // turn the job deployment on and off  sch
      edule: "55 23 * * *" // cron expression for it to run
      sparkMaster: // spark master connection string, when empty spark runs in
embedded local mode
```

4. 설치 및 실행

4.11 Installing the Agent as DaemonSet

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
   name: my-jaeger
spec:
   agent:
      strategy: DaemonSet
```

- 환경변수 "JAEGER_AGENT_HOST"를 Kubernetes노드의 IP값으로 설정(다음 페이지)

4. 설치 및 실행

11. Installing the Agent as DaemonSet

- 환경변수 "JAEGER_AGENT_HOST"를 Kubernetes노드의 IP값으로 설정

```
apiVersion: apps/v1
kind: Deployment  m
etadata:
   name: myapp \
spec:
   selector:
      matchLabels:
         app: myapp
   template:
      metadata:
         labels:
            app: myapp
      spec:
         containers:
         - name: myapp
           image: acme/myapp:myversion
           env:
           - name: JAEGER_AGENT_HOST
             valueFrom:
                fieldRef:
                   fieldPath: status.hostIP
```

4. 설치 및 실행

12. Configuring the UI

- 사용자 지정 리소스 내에서 UI 구성 변경 사항을 적용하기 위해 아래와 같이 yaml형식으로

```
ui:
   options:
      dependencies:  menuEn
         abled: false
      tracking:
         gaID: UA-000000-2
      menu:
      - label: "About Jaeger"
        items:
      - label: "Documentation"
        url: "https://www.jaegertracing.io/docs/latest"
      linkPatterns:
      - type: "logs"
        key: "customer_id"

url:/search?limit=20&lookback=1h&service=frontend&tags=%7B%22customer_id%22%3A%
22#{customer_id}%22%7D
text: "Search for other traces for customer_id=#{customer_id}"
```

4. 설치 및 실행

13. Defining Sampling Strategies

- 추적 인스턴스가 샘플링 될 확률이 50%인 샘플링 전략을 정의

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

4. 설치 및 실행

14. Finer grained configuration

JAEGER

•사용자 지정 리소스를 사용하면 모든 jaeger 구성요소 또는 개별 구성 요소 수준으로 kubernetes를 정의할 수 있음.

지원되는 구성 유형은 아래와 같음

- ✓ pod를 할당 할 수 있는 노드를 결정하는 선호도
- ✓ 주석
- ✓ 라벨
- ✓ cpu 및 메모리 리소스 제한
- ✓ 볼륨 및 볼륨 마운트
- ✓ 개별id로 실행되는 serviceAccount
- ✓ 실행중인 구성 요소의 권한을 정의하는 securityContext

4. 설치 및 실행

4.14 Finer grained configuration


```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
   name: simple-prod
spec:
   strategy: production
   storage:
      type: elasticsearch
      options:
         es:
            server-urls: http://elasticsearch:9200
   annotations:
      key1: value1
   labels:
      key2: value2
   resources:
      requests:
         memory: "64Mi"
         cpu: "250m"
      limits:
         memory: "128Mi"
         cpu: "500m"
```

4. 설치 및 실행

4.14 Finer grained configuration


```
affinity:
   nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
         nodeSelectorTerms:
         - matchExpressions:
            - key: kubernetes.io/e2e-az-name
              operator: In
              values:
              - e2e-az1
              - e2e-az2
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: another-node-label-key
            operator: In
            values:
            - another-node-label-value
```

4. 설치 및 실행

14. Finer grained configuration


```
tolerations:
   - key: "key1"  operato
     r: "Equal"  value: "
     value1"  effect: "No
     Schedule"
   - key: "key1"  operat
     or: "Equal"  value:
     "value1"  effect: "
     NoExecute"
serviceAccount: nameOfServiceAccount
securityContext:
   runAsUser: 1000
volumeMounts:
   - name: config-vol
     mountPath: /etc/config
volumes:
   - name: config-vol
     configMap:
        name: log-config
        items:
          - key: log_level
            path: log_level
```

4. 설치 및 실행

15. Accessing the Jaeger Console(UI)

- ingress 활성화

```
minikube addons ingress
```

- ingress 활성화 되면 ingress 쿼리를 통해 jaeger 콘솔 주소를 찾을수 있음

```
$ kubectl get ingress
NAME            HOSTS     ADDRESS         PORTS     AGE
simplest-query    *    192.168.122.34    80        3m
```

Q 종속성 페이지가 비어 있는 이유는 무엇입니까 ?

A

- 종속성 페이지에는 jaeger가 추적한 서비스 및 서비스 간 연결이 그래프로 표시됩니다. "all-in-one"메모리 내 저장소와 함께 메모리에 저장된 모든 추적 데이터를 필요시 그래프로 계산됩니다. 그러나 Cassandra 또는 Elasticsearch와 같은 실제 분산 스토리지를 사용하는 경우 서비스 그래프를 작성하기 위한 데이터베이스의 모든 데이터를 스캔하기에는 무리가 있습니다. 대신 jaeger프로젝트는 추적에서 서비스 그래프 데이터를 추출하는 데 사용할 수 있는 빅데이터 작업을 제공합니다.

Open Source Software Installation & Application Guide