SkyWalking 介绍与部署

简介

概念

SkyWalking 是什么?

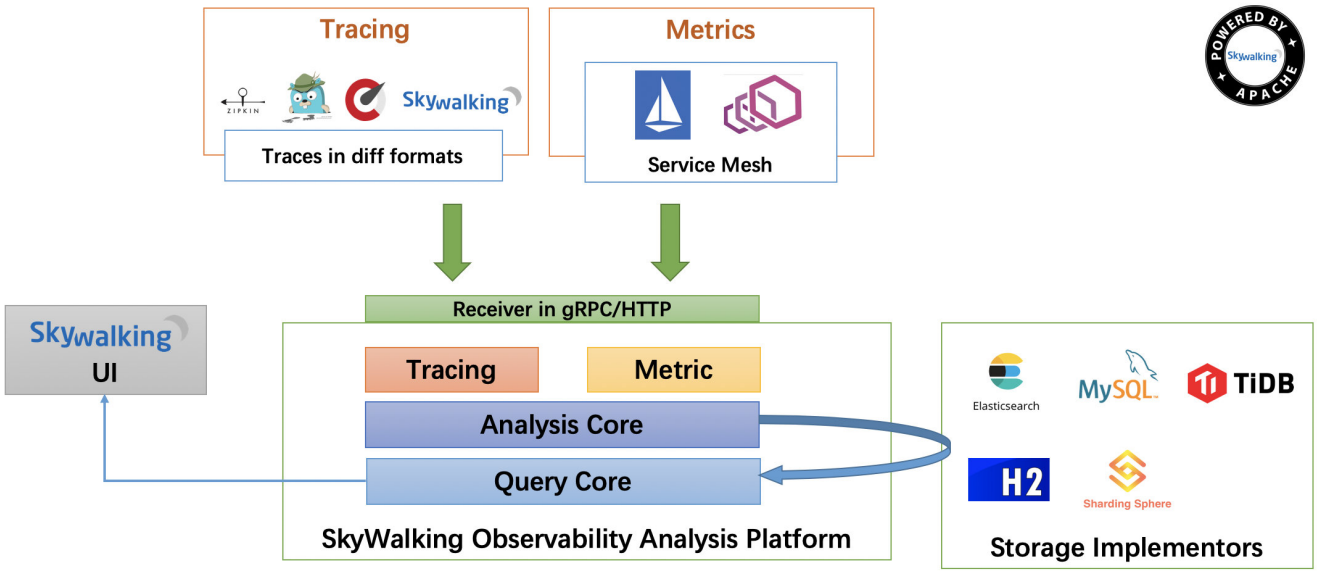
> 分布式系统的应用程序性能监视工具，专为微服务、云原生架构和基于容器（Docker、K8s、Mesos）架构而设计。 提供分布式追踪、服务网格遥测分析、度量聚合和可视化一体化解决方案。

功能

SkyWalking 功能有哪些?

- 多种监控手段。可以通过语言探针和 service mesh 获得监控是数据。
- 多个语言自动探针。包括 Java，.NET Core 和 Node.JS。
- 轻量高效。无需大数据平台，和大量的服务器资源。
- 模块化。UI、存储、集群管理都有多种机制可选。
- 支持告警。
- 优秀的可视化解决方案。

整体架构



整个架构，分成上、下、左、右四部分：

- 上部分 **Agent**： 负责从应用中，收集链路信息，发送给 SkyWalking OAP 服务器。目前支持 SkyWalking、Zikpin、Jaeger 等提供的 Tracing 数据信息。而我们目前采用的是，SkyWalking Agent 收集 SkyWalking Tracing 数据，传递给服务器。
- 下部分 **SkyWalking OAP**： 负责接收 Agent 发送的 Tracing 数据信息，然后进行分析(Analysis Core)，存储到外部存储器(Storage)，最终提供查询(Query)功能。

- 右部分 **Storage**： Tracing 数据存储。目前支持 ES、MySQL、Sharding Sphere、TiDB、H2 多种存储器。而我们目前采用的是 ES，主要考虑是 SkyWalking 开发团队自己的生产环境采用 ES 为主。
- 左部分 **SkyWalking UI**： 负责提供控台，查看链路等等。

在K8S中部署SkyWalking

本次文章使用Helm部署skywalking。 在k8s中使用helm的前提是需要安装helm客户端，关于helm的安装方式可以查看官方文档

> 安装 helm 官方文档地址：https://helm.sh/docs/intro/install/

部署elasticsearch

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: elasticsearch
  namespace: skywalking
  labels:
    app: skywalking-elasticsearch
spec:
  replicas: 1
  selector:
    matchLabels:
      app: skywalking-elasticsearch
  template:
    metadata:
      labels:
        app: skywalking-elasticsearch
    spec:
      containers:
      - image: elasticsearch:7.12.0
        name: skywalking-elasticsearch
        resources:
          limits:
            cpu: 2
            memory: 3Gi
          requests:
            cpu: 0.5
            memory: 500Mi
        env:
          - name: "discovery.type"
            value: "single-node"
          - name: ES_JAVA_OPTS
            value: "-Xms512m -Xmx2g"
        ports:
        - containerPort: 9200
          name: db
          protocol: TCP
        volumeMounts:
        - name: elasticsearch-data
          mountPath: /usr/share/elasticsearch/data
```

```
      volumes:
      - name: elasticsearch-data
        persistentVolumeClaim:
          claimName: es-pvc

---

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: es-pvc
  namespace: skywalking
spec:
#指定动态PV 名称
  storageClassName: "aks-azurefile"
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi

---

apiVersion: v1
kind: Service
metadata:
  name: skywalking-elasticsearch
  namespace: skywalking
spec:
  ports:
  - port: 9200
    protocol: TCP
    targetPort: 9200
  selector:
    app: skywalking-elasticsearch
```

部署SkyWalking

初始化SkyWalking的配置

```
# clone SkyWalking 的chart仓库
git clone https://github.com/apache/skywalking-kubernetes.git
# 进入目录
cd skywalking-kubernetes/chart/skywalking/
# 添加repo
export REPO=skywalking
helm repo add ${REPO} https://apache.jfrog.io/artifactory/skywalking-helm

# 环境变量
export SKYWALKING_RELEASE_NAME=skywalking  # change the release name according to
your scenario
```

```
export SKYWALKING_RELEASE_NAMESPACE=skywalking  # change the namespace to where
you want to install SkyWalking
```

修改skywalking的配置参数

初始化完成后需要自行调整配置文件，配置 oap-server 使用外部 ES

```
oap:
  image:
    tag: 9.2.0
  storageType: elasticsearch

ui:
  image:
    tag: 9.2.0
  service:
    type: LoadBalancer
    annotations:
      service.beta.kubernetes.io/azure-load-balancer-internal: "true" # 使用azure
的内部负载均衡
    externalPort: 80


elasticsearch:
  enabled: false
  config:               # For users of an existing elasticsearch cluster,takes
effect when `elasticsearch.enabled` is false
    host: skywalking-elasticsearch
    port:
      http: 9200
    user: "xxx"         # [optional]
    password: "xxx"     # [optional]
```

安装SkyWalking

```
helm install "${SKYWALKING_RELEASE_NAME}" ${REPO}/skywalking -n
"${SKYWALKING_RELEASE_NAMESPACE}" \
  -f values-my-es.yaml
```

使用 sidecar 将 pod 接入链路追踪

Java微服务接入skywalking 可以使用 SkyWalking Java Agent 来上报监控数据，这就需要 java 微服务在启动参数中通过 -javaagent:<skywalking-agent-path> 指定 skywalking agent 探针包，通常有以下三种方式集成：

- 使用官方提供的基础镜像 skywalking-base
- 将 agent 包构建到已存在的镜像中
- 通过 sidecar 模式挂载 agent

这里主要介绍如何使用 sidecar 将 pod 接入链路追踪，这种方式不需要修改原来的基础镜像，也不需要重新构建新的服务镜像，而是会以sidecar模式，通过共享的 volume 将 agent 所需的相关文件直接挂载到已经存在的服务镜像中。sidecar模式原理很简单，就是在 pod 中再部署一个初始容器，这个初始容器的作用就是将 skywalking agent 和 pod 中的应用容器共享。

- **什么是初始化容器 init container**

> Init Container 就是用来做初始化工作的容器，可以是一个或者多个，如果有多个的话，这些容器会按定义的顺序依次执行，只有所有的 Init Container 执行完后，主容器才会被启动。我们知道一个Pod里面的所有容器是共享数据卷和网络命名空间的，所以 Init Container 里面产生的数据可以被主容器使用到的。

自定义 skywalking agent 镜像

在开始以 sidecar 方式将一个 java 微服务接入 skywalking 之前，我们需要构建 skywalking agent 的公共镜像，具体步骤如下：

```
# 使用下面的命令下载 skywalking agent 并解压
wget https://www.apache.org/dyn/closer.cgi/skywalking/java-agent/8.12.0/apache-skywalking-java-agent-8.12.0.tgz
# 将下载的发布包解压到当前目录
tar -xzvf apache-skywalking-java-agent-8.12.0.tgz
```

在前面步骤中解压的 skywalking 发行包的同级目录编写 Dockerfile 文件，具体内容如下：

```
FROM busybox:latest
LABEL maintainer="baoxi"
COPY skywalking-agent/ /usr/skywalking/agent/
```

在上述 Dockefile 文件中使用的基础镜像是 bosybox 镜像，而不是 SkyWalking 的发行镜像，这样可以确保构建出来的sidecar镜像保持最小。

```
# 构建镜像
docker build -t democrdev.azurecr.cn/skywalking-agent-sidecar:9.2.0 .
docker push democrdev.azurecr.cn/skywalking-agent-sidecar:9.2.0
```

上面我们通过手工构建的方式构建了 SkyWalking Java Agent 的公共 Docker 镜像，接下来我们将演示如何通过编写 Kubernetes 服务发布文件，来将 Java 服务发布到 K8s 集群的过程中自动以 SideCar 的形式集成Agent 并接入 SkyWalking 服务。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-job
```

```yaml
    labels:
      app: demo-job
  spec:
    replicas: 3
    selector:
      matchLabels:
        app: demo-job
    template:
      metadata:
        labels:
          app: demo-job
      spec:
        nodeSelector:
          "kubernetes.io/os": linux
        initContainers:
          - image: democrdev.azurecr.cn/skywalking-agent-sidecar:9.2.0
            name: sw-agent-sidecar
            imagePullPolicy: IfNotPresent
            command: [ "sh" ]
            args:
              [
                  "-c",
                  "cp -R /usr/skywalking/agent/* /skywalking/agent",
              ]
            volumeMounts:
              - mountPath: /skywalking/agent
                name: sw-agent

        containers:
        - name: demo-job
          image: democrdev.azurecr.cn/demo-job:v202210131612
          ports:
            - containerPort: 9203
          volumeMounts:
          - name: sw-agent
            mountPath: /usr/skywalking/agent
          env:

            - name: JAVA_TOOL_OPTIONS
              value: -javaagent:/usr/skywalking/agent/skywalking-agent.jar
            - name: SW_AGENT_NAME
              value: demo-job
            - name: SW_AGENT_COLLECTOR_BACKEND_SERVICES
              value: skywalking-oap.skywalking.svc:11800

        volumes:
          - name: sw-agent
            emptyDir: { }
  ---

  apiVersion: v1
  kind: Service
  metadata:
    name: demo-job
```

```
    labels:
      app: demo-job
  spec:
    selector:
      app: demo-job
    ports:
    - port: 9203
      protocol: TCP
      targetPort: 9203
```

参考文档

https://zhaouncle.com/k8s环境-helm部署skywalking/#41-安装-es-7120

https://github.com/apache/skywalking-kubernetes/tree/master/chart/skywalking

https://www.jianshu.com/p/e7acc1da9e68

https://codeantenna.com/a/B3sOPDwB7p

https://itindex.net/detail/61361-k8s-skywalking-pod

https://tehub.com/a/8xl4knLGWm