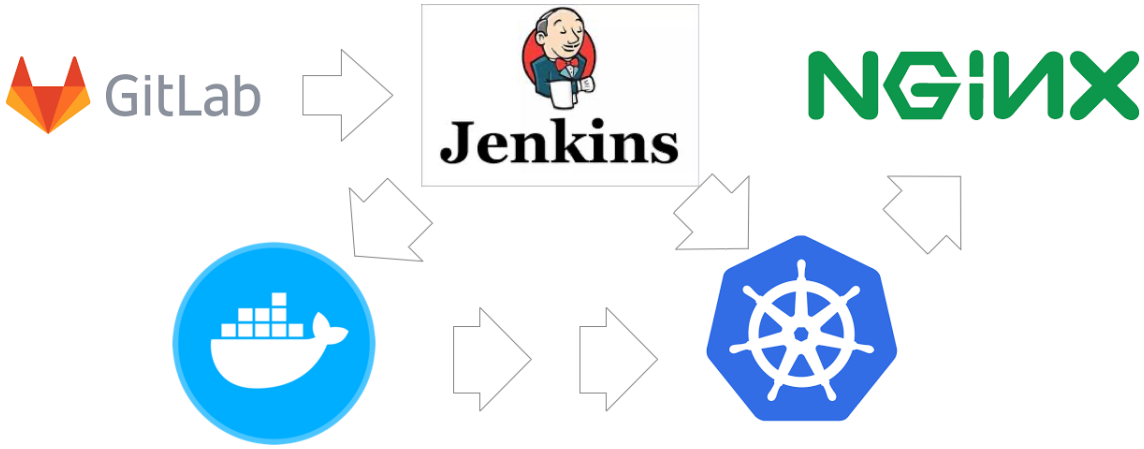
讲师(老司机)

容器虚拟化技术和自动化部署



k8s高级篇-service

service

简介

通过以前的学习，我们已经能够通过Deployment来创建一组Pod来提供具有高可用性的服务。虽然每个Pod都会分配一个单独的Pod IP，然而却存在如下两问题：

- Pod IP仅仅是集群内可见的虚拟IP，外部无法访问。
- Pod IP会随着Pod的销毁而消失，当Deployment对Pod进行动态伸缩时，Pod IP可能随时随地都会变化，这样对于我们访问这个服务带来了难度。
- Service能够提供负载均衡的能力，但是在使用上有以下限制。只提供 4 层负载均衡能力，而没有 7 层功能，但有时我们可能需要更多的匹配规则来转发请求，这点上 4 层负载均衡是不支持的

因此，Kubernetes中的Service对象就是解决以上问题的实现服务发现核心关键。

Service 在 K8s 中有以下四种类型

- ClusterIp：默认类型，自动分配一个仅 Cluster 内部可以访问的虚拟 IP
- NodePort：在 ClusterIP 基础上为 Service 在每台机器上绑定一个端口，这样就可以通过 : NodePort 来访
  问该服务
- LoadBalancer：在 NodePort 的基础上，借助 cloud provider 创建一个外部负载均衡器，并将请求转发
  到NodePort。是付费服务，而且价格不菲。
- ExternalName：把集群外部的服务引入到集群内部来，在集群内部直接使用。没有任何类型代理被创建，
  这只有 kubernetes 1.7 或更高版本的 kube-dns 才支持

详解4 种 Service 类型

Services 和 Pods

Kubernetes的Pods是有生命周期的。他们可以被创建，而且销毁不会再启动。如果您使用Deployment来运行您的应用程序，则它可以动态创建和销毁 Pod。

一个Kubernetes的Service是一种抽象，它定义了一组Pods的逻辑集合和一个用于访问它们的策略 - 有的时候被称之为微服务。一个Service的目标Pod集合通常是由Label Selector 来决定的

举个例子，想象一个处理图片的后端运行了三个副本。这些副本都是可以替代的 - 前端不关心它们使用的是哪一个后端。尽管实际组成后端集合的Pod可能会变化，前端的客户端却不需要知道这个变化，也不需要自己有一个列表来记录这些后端服务。Service抽象能让你达到这种解耦。

不像 Pod 的 IP 地址，它实际路由到一个固定的目的地，Service 的 IP 实际上不能通过单个主机来进行应答。相反，我们使用 iptables（Linux 中的数据包处理逻辑）来定义一个虚拟IP地址（VIP），它可以根据需要透明地进行重定向。当客户端连接到 VIP 时，它们的流量会自动地传输到一个合适的 Endpoint。环境变量和 DNS，实际上会根据 Service 的 VIP 和端口来进行填充。

kube-proxy支持三种代理模式: 用户空间，iptables和IPVS；它们各自的操作略有不同。

- **Userspace代理模式**

Client Pod要访问Server Pod时,它先将请求发给本机内核空间中的service规则，由它再将请求,转给监听在指定套接字上的kube-proxy，kube-proxy处理完请求，并分发请求到指定Server Pod后,再将请求递交给内核空间中的service,由service将请求转给指定的Server Pod。由于其需要来回在用户空间和内核空间交互通信，因此效率很差 。

当一个客户端连接到一个 VIP，iptables 规则开始起作用，它会重定向该数据包到 Service代理 的端口。Service代理 选择一个 backend，并将客户端的流量代理到 backend 上。

这意味着 Service 的所有者能够选择任何他们想使用的端口，而不存在冲突的风险。客户端可以简单地连接到一个 IP 和端口，而不需要知道实际访问了哪些 Pod。

- ***iptables代理模式***

当一个客户端连接到一个 VIP，iptables 规则开始起作用。一个 backend 会被选择（或者根据会话亲和性，或者随机），数据包被重定向到这个 backend。不像 userspace 代理，数据包从来不拷贝到用户空间，kube-proxy 不是必须为该 VIP 工作而运行，并且客户端 IP 是不可更改的。当流量打到 Node 的端口上，或通过负载均衡器，会执行相同的基本流程，但是在那些案例中客户端 IP 是可以更改的。

- ***IPVS代理模式***

在大规模集群（例如10,000个服务）中，iptables 操作会显着降低速度。IPVS 专为负载平衡而设计，并基于内核内哈希表。因此，您可以通过基于 IPVS 的 kube-proxy 在大量服务中实现性能一致性。同时，基于 IPVS 的 kube-proxy 具有更复杂的负载平衡算法（最小连接，局部性，加权，持久性）。

在 Kubernetes 集群中，每个 Node 运行一个 kube-proxy 进程。 kube-proxy 负责为 Service 实现了一种
VIP（虚拟 IP）的形式，而不是 ExternalName 的形式。 在 Kubernetes v1.0 版本，代理完全在 userspace。在
Kubernetes v1.1 版本，新增了 iptables 代理，但并不是默认的运行模式。 从 Kubernetes v1.2 起，默认就是
iptables 代理。 在 Kubernetes v1.8.0-beta.0 中，添加了 ipvs 代理在 Kubernetes 1.14 版本开始默认

使用 ipvs 代理。在 Kubernetes v1.0 版本， Service 是 “4层”（TCP/UDP over IP）概念。 在 Kubernetes v1.1 版本，新增了Ingress API（beta 版），用来表示 “7层”（HTTP）服务 。

这种模式，kube-proxy 会监视 Kubernetes Service 对象和 Endpoints ，调用 netlink 接口以相应地创建
ipvs 规则并定期与 Kubernetes Service 对象和 Endpoints 对象同步 ipvs 规则，以确保 ipvs 状态与期望一
致。访问服务时，流量将被重定向到其中一个后端 Pod与 iptables 类似，ipvs 于 netfilter 的 hook 功能，但使用哈希表作为底层数据结构并在内核空间中工作。这意味着 ipvs 可以更快地重定向流量，并且在同步代理规则时具有更好的性能。此外，ipvs 为负载均衡算法提供了更多选项。

ClusterIP

类型为ClusterIP的service，这个service有一个Cluster-IP，其实就一个VIP。具体实现原理依靠kubeproxy组件，通过iptables或是ipvs实现。

clusterIP 主要在每个 node 节点使用 iptables，将发向 clusterIP 对应端口的数据，转发到 kube-proxy 中。然
后 kube-proxy 自己内部实现有负载均衡的方法，并可以查询到这个 service 下对应 pod 的地址和端口，进而把
数据转发给对应的 pod 的地址和端口

这种类型的service 只能在集群内访问。

使用镜像

```
docker pull tomcat:9.0.20-jre8-alpine
```

部署service

service/clusteripdemo.yml：

```
apiversion: apps/v1
kind: Deployment
metadata:
  name: clusteripdemo
  labels:
    app: clusteripdemo
spec:
  replicas: 3
  template:
    metadata:
```

```
      name: clusteripdemo
      labels:
        app: clusteripdemo
    spec:
      containers:
        - name: clusteripdemo
          image: tomcat:9.0.20-jre8-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8080
      restartPolicy: Always
  selector:
    matchLabels:
      app: clusteripdemo
---
apiVersion: v1
kind: Service
metadata:
  name: clusterip-svc
spec:
  selector:
    app: clusteripdemo
  ports:
    - port: 8080
      targetPort: 8080
  type: ClusterIP
```

运行service

```
运行服务
kubectl apply -f clusteripdemo.yml

查看服务
kubectl get svc

访问服务
curl 10.1.15.24:8080

删除服务
kubectl delete -f clusteripdemo.yml
```

NodePort

我们的场景不全是集群内访问，也需要集群外业务访问。那么ClusterIP就满足不了了。NodePort当然是其中的一种实现方案。nodePort 的原理在于在 node 上开了一个端口，将向该端口的流量导入到 kube-proxy，然后由 kube-proxy 进一步到给对应的 pod 。

使用镜像

```
docker pull tomcat:9.0.20-jre8-alpine
```

部署service

service/nodeportdemo.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: clusteripdemo
  labels:
    app: clusteripdemo
spec:
  replicas: 3
  template:
    metadata:
      name: clusteripdemo
      labels:
        app: clusteripdemo
    spec:
      containers:
        - name: clusteripdemo
          image: tomcat:9.0.20-jre8-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8080
      restartPolicy: Always
  selector:
    matchLabels:
      app: clusteripdemo
---
apiVersion: v1
kind: Service
metadata:
  name: clusterip-svc
spec:
  selector:
    app: clusteripdemo
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30088
  type: NodePort
```

运行service

```
运行服务
kubectl apply -f nodeportdemo.yml

查看服务
kubectl get svc

访问服务
curl 10.1.61.126:8080

```

```
浏览器访问服务
http://192.168.198.156:30088

删除服务
kubectl delete -f nodeportdemo.yml
```

LoadBalancer

LoadBalancer类型的service 是可以实现集群外部访问服务的另外一种解决方案。不过并不是所有的k8s集群都会支持，大多是在公有云托管集群中会支持该类型。负载均衡器是异步创建的，关于被提供的负载均衡器的信息将会通过Service的status.loadBalancer字段被发布出去。

创建 LoadBalancer service 的yaml 如下：

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-lagou
spec:
  ports:
  - port: 3000
    protocol: TCP
    targetPort: 443
    nodePort: 30080
  selector:
    run: pod-lagou
  type: LoadBalancer
```

ExternalName

类型为 ExternalName 的service将服务映射到 DNS 名称，而不是典型的选择器，例如my-service或者cassandra。您可以使用spec.externalName参数指定这些服务。

这种类型的 Service 通过返回 CNAME 和它的值，可以将服务映射到 externalName 字段的内容(例如：
hub.lagouedu.com)。ExternalName Service 是 Service 的特例，它没有 selector，也没有定义任何的端口和
Endpoint。相反的，对于运行在集群外部的服务，它通过返回该外部服务的cname(别名)这种方式来提供服务

创建 ExternalName 类型的服务的 yaml 如下：

```yaml
kind: Service
apiVersion: v1
metadata:
  name: service-lagou
spec:
  ports:
  - port: 3000
    protocol: TCP
    targetPort: 443
  type: ExternalName
  externalName: www.lagou.com
```

ingress网络

什么是Ingress?

K8s集群对外暴露服务的方式目前只有三种：loadblancer、nodeport、ingress。前两种熟悉起来比较快，而且使用起来也比较方便，前面课程已经讲解过两种技术。在此就不进行介绍了。

下面详细讲解下ingress这个服务，ingress由两部分组成：ingress controller和ingress服务。

其中ingress controller目前主要有两种：基于nginx服务的ingress controller和基于traefik的ingress controller。

而其中traefik的ingress controller，目前支持http和https协议。由于对nginx比较熟悉，而且需要使用TCP负载，所以在此我们选择的是基于nginx服务的ingress controller。

在kubernetes集群中，我们知道service和pod的ip仅在集群内部访问。如果外部应用要访问集群内的服务，集群外部的请求需要通过负载均衡转发到service在Node上暴露的NodePort上，然后再由kube-proxy组件将其转发给相关的pod。

而Ingress就是为进入集群的请求提供路由规则的集合，通俗点就是提供外部访问集群的入口，将外部的HTTP或者HTTPS请求转发到集群内部service上。

Ingress-nginx组成

Ingress-nginx一般由三个组件组成：

- 反向代理负载均衡器：通常以service的port方式运行，接收并按照ingress定义的规则进行转发，常用的有nginx，Haproxy，Traefik等，本次实验中使用的就是nginx。
- Ingress Controller：监听APIServer，根据用户编写的ingress规则（编写ingress的yaml文件），动态地去更改nginx服务的配置文件，并且reload重载使其生效，此过程是自动化的（通过lua脚本来实现）。
- Ingress：将nginx的配置抽象成一个Ingress对象，当用户每添加一个新的服务，只需要编写一个新的ingress的yaml文件即可。

Ingress-nginx的工作原理

1. ingress controller通过和kubernetes api交互，动态的去感知集群中ingress规则变化。然后读取它，按照自定义的规则，规则就是写明了那个域名对应哪个service，生成一段nginx配置。
2. 再写到nginx-ingress-controller的pod里，这个Ingress controller的pod里运行着一个Nginx服务，控制器会把生成的nginx配置写入/etc/nginx.conf文件中。然后reload一下使配置生效，以此

达到分配和动态更新问题。

官网地址

基于nginx服务的ingress controller根据不同的开发公司，又分为k8s社区的ingres-nginx和nginx公司的nginx-ingress。

```
Ingress-Nginx github 地址:
https://github.com/kubernetes/ingress-nginx

Ingress-Nginx 官方网站:
https://kubernetes.github.io/ingress-nginx/
```

下载资源文件

根据github上的活跃度和关注人数，我们选择的是k8s社区的ingres-nginx。

下载ingrees-controller

```
打开github官网：选择某一个具体版本后进入deploy/static/目录中，复制mandatory.yaml文件内容。
https://github.com/kubernetes/ingress-nginx/tree/nginx-0.30.0/deploy/static/mandatory.yaml
```

下载ingress服务

```
https://github.com/kubernetes/ingress-nginx/blob/nginx-0.30.0/deploy/static/provider/baremetal/service-nodeport.yaml
```

下载镜像

需要将镜像pull到k8s集群各个node节点

```
docker pull registry.cn-hangzhou.aliyuncs.com/google_containers/nginx-ingress-controller:0.30.0

docker tag registry.cn-hangzhou.aliyuncs.com/google_containers/nginx-ingress-controller:0.30.0 quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.30.0

docker rmi -f registry.cn-hangzhou.aliyuncs.com/google_containers/nginx-ingress-controller:0.30.0
```

ingress与ingress-controller

要理解ingress，需要区分两个概念，ingress和ingress-controller：

- ingress对象：
  指的是k8s中的一个api对象，一般用yaml配置。作用是定义请求如何转发到service的规则，可以理解为配置模板。
- ingress-controller：
  具体实现反向代理及负载均衡的程序，对ingress定义的规则进行解析，根据配置的规则来实现请求转发。

简单来说，ingress-controller才是负责具体转发的组件，通过各种方式将它暴露在集群入口，外部对集群的请求流量会先到ingress-controller，而ingress对象是用来告诉ingress-controller该如何转发请求，比如哪些域名哪些path要转发到哪些服务等等。

ingress-controller

ingress-controller并不是k8s自带的组件，实际上ingress-controller只是一个统称，用户可以选择不同的ingress-controller实现，目前，由k8s维护的ingress-controller只有google云的GCE与ingress-nginx两个，其他还有很多第三方维护的ingress-controller，具体可以参考官方文档。但是不管哪一种ingress-controller，实现的机制都大同小异，只是在具体配置上有差异。一般来说，ingress-controller的形式都是一个pod，里面跑着daemon程序和反向代理程序。daemon负责不断监控集群的变化，根据ingress对象生成配置并应用新配置到反向代理，比如nginx-ingress就是动态生成nginx配置，动态更新upstream，并在需要的时候reload程序应用新配置。

ingress

ingress是一个API对象，和其他对象一样，通过yaml文件来配置。ingress通过http或https暴露集群内部service，给service提供外部URL、负载均衡、SSL/TLS能力以及基于host的方向代理。ingress要依靠ingress-controller来具体实现以上功能。

ingress网络实验一

使用镜像

```
docker pull tomcat:9.0.20-jre8-alpine
docker pull quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.30.0
```

运行ingress-controller

```
在mandatory.yaml文件的Deployment资源中增加属性sepc.template.sepc.hostNetWork
hostNetwork: true
hostNetwork网络，这是一种直接定义Pod网络的方式。如果在Pod中使用hostNetwork:true配置网络，那么Pod中运行的应用程序可以直接使用node节点的端口

运行ingress/mandatory.yaml文件

kubectl apply -f mandatory.yaml
```

运行ingress服务

```
运行ingress/service-nodeport.yaml文件

kubectl apply -f service-nodeport.yaml
```

部署tomcat-服务

ingress/tomcat-service.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat-deploy
  labels:
    app: tomcat-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: tomcat-deploy
      labels:
        app: tomcat-deploy
    spec:
      containers:
        - name: tomcat-deploy
          image: tomcat:9.0.20-jre8-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8080
      restartPolicy: Always
  selector:
    matchLabels:
      app: tomcat-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: tomcat-svc
spec:
  selector:
    app: tomcat-deploy
  ports:
    - port: 8080
      targetPort: 8080
  type: NodePort
```

运行tomcat-service

```
kubectl apply -f tomcat-service.yml
```

部署ingress规则文件

ingress/ingress-tomcat.yml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress-test
spec:
  backend:
    serviceName: tomcat-svc
    servicePort: 8080
```

运行ingress规则

```
kubectl apply -f ingress-tomcat.yml

查看ingress
kubectl get ingress

查看ingress服务:查看service的部署端口号
kubectl get svc -n ingress-nginx

查看ingress-controller运行在那个node节点
kubectl get pod -n ingress-nginx -o wide
```

通过ingress访问tomcat

```
http://192.168.198.158:31530/
```

ingress网络实验二

上边案例的部署方式只能通过ingress-controller部署的节点访问。集群内其他节点无法访问ingress规则。本章节通过修改mandatory.yaml文件的控制类类型，让集群内每一个节点都可以正常访问ingress规则。

ingress-controller

ingress/mandatory.yaml

```
修改mandatory.yaml配置文件

1.将Deployment类型控制器修改为：DaemonSet
2.属性：replicas: 1  # 删除这行
```

service-nodeport固定端口

ingress/service-nodeport.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  type: NodePort
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 31188
      protocol: TCP
    - name: https
      port: 443
      targetPort: 443
      nodePort: 31443
      protocol: TCP
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
```

域名访问ingress规则

service/ingress-tomcat.yml

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress-test
spec:
  rules:
    - host: ingress-tomcat.lagou.com
      http:
        paths:
          - path: /
            backend:
              serviceName: nodeporttomcat-svc
              servicePort: 8080
```

修改宿主机hosts文件

C:\Windows\System32\drivers\etc\hosts

```
增加ingress-tomcat.lagou.com 域名配置：

192.168.198.157 ingress-tomcat.lagou.com
```

部署服务

```
kubectl apply -f .
```

浏览器测试

```
http://ingress-tomcat.lagou.com:31188/
```

nginx-controller原理

```
查看ingress-nginx 命名空间下的pod
kubectl get pods -n ingress-nginx

进入ingress-nginx 的pod
kubectl exec -it nginx-ingress-controller-5gt4l -n ingress-nginx sh

查看nginx反向代理域名ingress-tomcat.lagou.com
cat nginx.conf
```