**k8s部署aspnetcore示例**

<https://www.cnblogs.com/lonelyxmas/p/10668432.html>

**1、Pod简介**

Kubernetes有很多技术概念，同时对应很多API对象，最重要的也是最基础的对象就是Pod。Pod是Kubernetes集群中运行部署应用的最小单元，并且是支持多个容器的。

Pod的设计理念是支持多个容器在一个Pod中共享网络地址和文件系统，可以通过进程间通信和文件共享这种简单高效的方式组合完成服务。Pod对多容器的支持是Kubernetes最基础的设计理念。比如你运行一个操作系统发行版的软件仓库，一个Nginx容器用来发布软件，另一个容器专门用来从源仓库做同步，这两个容器的镜像不太可能是一个团队开发的，但是他们一块儿工作才能提供一个微服务；这种情况下，不同的团队各自开发构建自己的容器镜像，在部署的时候组合成一个微服务对外提供服务。不过，在大多数情况下，我们只会在Pod中运行一个容器，本文中的例子也是这样的。

Docker是kubernetes中最常用的容器运行时，但是Pod也支持其他容器运行时。

总的来说，Pod的主要特征包括：

(1)-每个Pod可以在Kubernetes集群内拥有唯一的IP地址；

(2)-Pod可以拥有多个容器。这些容器共享同一个端口空间，所以他们可以通过localhost交流（可想而知它们无法使用相同的端口），与其他Pod内容器的交流可以通过结合Pod的IP来完成；

(3)-一个Pod内的容器共享同一个卷、同一个 IP、端口空间、IPC 命名空间。

**2、定义并创建Pod**

新建test\_pod.yaml文件，内容如下：

apiVersion: v1

kind: Pod # 定义Kubernetes资源的类型为Pod

metadata:

  name: test-pod # 定义资源的名称

  labels: # 为Pod贴上标签，后面会介绍其用处

    app: test-app

spec: # 定义资源的状态，对于Pod来说，最重要属性就是containers

  containers: # containers一个数组类型，如果你希望部署多个容器，可以添加多项

    - name: aspnetapp # 定义本Pod中该容器的名称

      image: mcr.microsoft.com/dotnet/core/samples:aspnetapp # 定义Pod启动的容器镜像

      ports:

        - containerPort: 80 # 定义容器监听的端口

**【创建Pod】**

kubectl create -f test\_pod.yaml



**【查看Pod】**

kubectl get pods



**3、kubenretes中的访问Pod的5种方式**

hostNetwork、hostPort、NodePort、LoadBalancer、Ingress

**【hostNetwork】**

当Pod 设置hostNetwork:true时候，Pod中的所有容器就直接暴露在宿主机的网络环境中，这时候，Pod的PodIP就是其所在Node的IP。**注意，每次重新启动pod时，Kubernetes都可能将pod重新安排到其他节点上，因此应用程序的IP地址将更改**。以下是使用主机网络的Pod的示例定义：

apiVersion: v1

kind: Pod # 定义Kubernetes资源的类型为Pod

metadata:

  name: test-pod # 定义资源的名称

  labels: # 为Pod贴上标签，后面会介绍其用处

    app: test-app

spec: # 定义资源的状态，对于Pod来说，最重要属性就是containers

hostNetwork: true # 定义Pod网络方式

  containers: # containers一个数组类型，如果你希望部署多个容器，可以添加多项

    - name: aspnetapp # 定义本Pod中该容器的名称

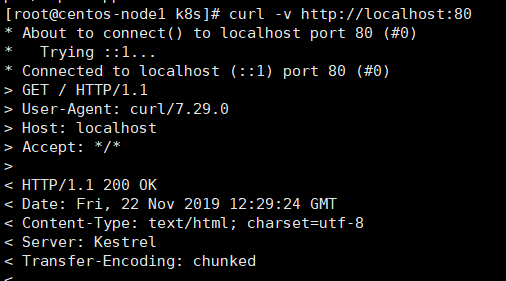
      image: mcr.microsoft.com/dotnet/core/samples:aspnetapp # 定义Pod启动的容器镜像

      ports:

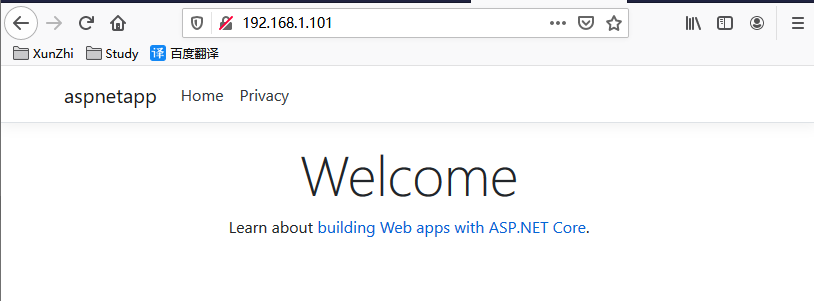
        - containerPort: 80 # 定义容器监听的端口（与Dockerfile中的EXPOSE类似）

内部访问Pod

curl -v <http://localhost:80>



外部访问：http://192.168.1.101:80



**【hostPort】**

apiVersion: v1

kind: Pod # 定义Kubernetes资源的类型为Pod

metadata:

  name: test-pod # 定义资源的名称

  labels: # 为Pod贴上标签，后面会介绍其用处

    app: test-app

spec: # 定义资源的状态，对于Pod来说，最重要属性就是containers

  containers: # containers一个数组类型，如果你希望部署多个容器，可以添加多项

    - name: aspnetapp # 定义本Pod中该容器的名称

      image: mcr.microsoft.com/dotnet/core/samples:aspnetapp # 定义Pod启动的容器镜像

      ports:

        - containerPort: 80 # 定义容器监听的端口（与Dockerfile中的EXPOSE类似）

hostPort:8080 # 定义节点端口

外部访问：http://192.168.1.101:8080

**【NodePort】**

NodePort在kubenretes里是一个广泛应用的服务暴露方式。Kubernetes中的Service默认情况下都是使用的ClusterIP这种类型，这样的service会产生一个ClusterIP，这个IP只能在集群内部访问，要想让外部能够直接访问service，需要将service type修改为 nodePort。

**定义pod：**

apiVersion: v1

kind: Pod # 定义Kubernetes资源的类型为Pod

metadata:

  name: test-pod # 定义资源的名称

  labels: # 为Pod贴上标签，后面会介绍其用处

    app: test-app

spec: # 定义资源的状态，对于Pod来说，最重要属性就是containers

  containers: # containers一个数组类型，如果你希望部署多个容器，可以添加多项

    - name: aspnetapp # 定义本Pod中该容器的名称

      image: mcr.microsoft.com/dotnet/core/samples:aspnetapp # 定义Pod启动的容器镜像

      ports:

        - containerPort: 80 # 定义容器监听的端口（与Dockerfile中的EXPOSE类似）

**定义service：**

apiVersion: v1

kind: Service # 定义Kubernetes资源的类型为Service

metadata:

  name: test-service # 定义资源的名称

spec:

  selector: # 指定对应的Pod

    app: test-app # 指定Pod的标签为test-app

  type: NodePort # 指定Service的类型，在这里使用NodePort来对外访问

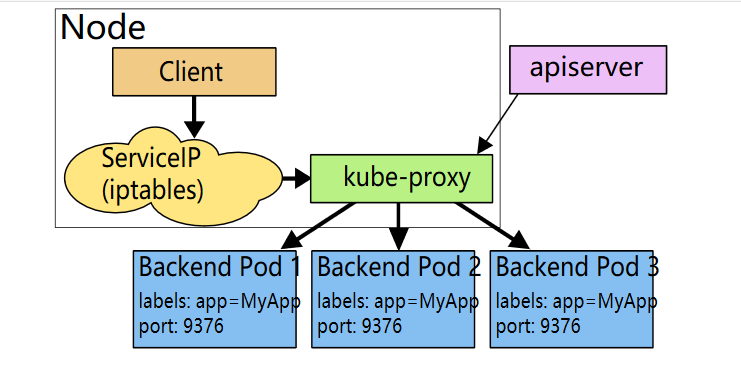
  ports:

  - port: 8080 # Service的端口,Service的8080端口映射到Pod的80端口

    targetPort: 80 # Pod的端口

    nodePort: 31000 #映射主机端口，不指定则在30000-32767范围分配

集群外就可以使用kubernetes任意一个节点的IP加上31000端口访问该服务了。kube-proxy会自动将流量以round-robin的方式转发给该service的每一个pod。



**3、定义并创建Service**

为了从外部访问应用程序，我们需要创建Kubernetes中的另外一种资源：Service。

Kubernetes中的Service资源可以作为一组提供相同服务的Pod的入口，这个资源负责发现服务和平衡Pod之间负荷。

在Kubernetes集群中，我们拥有提供不同服务的Pod，那么Service如何知道该处理哪个Pod呢？

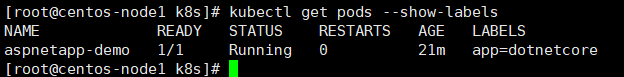
这个问题就用标签来解决的，具体分两个步骤：

(1)-给所有需要Service处理的对象Pod贴上标签。

(2)-在Service中使用一个选择器（Label Selector），该选择器定义了所有贴有对应的标签的对象Pod。

**【标签】**

标签提供了一种简单的方法用于管理Kubernetes中的资源。它们用一对键值表示，且可以用于所有资源。在上面的pod定义就用了标签,使用kubectl get pods --show-labels来查看Pod对应标签：



**【定义service】**

新建aspnetcore\_demo.yaml文件，内容如下：

apiVersion: v1

kind: Service # 定义Kubernetes资源的类型为Service

metadata:

  name: app-web-service # 定义资源的名称

spec:

  selector: # 指定对应的Pod

    app: aspnetapp-demo # 指定Pod的标签为demo-web

  type: NodePort # 指定Service的类型，在这里使用NodePort来对外访问

  ports:

  - protocol: TCP # 协议类型

    nodePort: 31000 #物理主机端口

    port: 80 # 指定Service访问的端口

    targetPort: 80 # 指定Service转发请求的端口

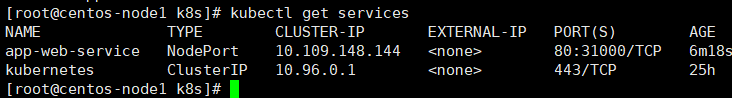
**【创建service】**

kubectl create -f app-web-service.yaml



**【查看service服务状态】**

kubectl get services



**【删除Service】**

kubectl delete -n default service app-web-service