13.如何创建一个Pod

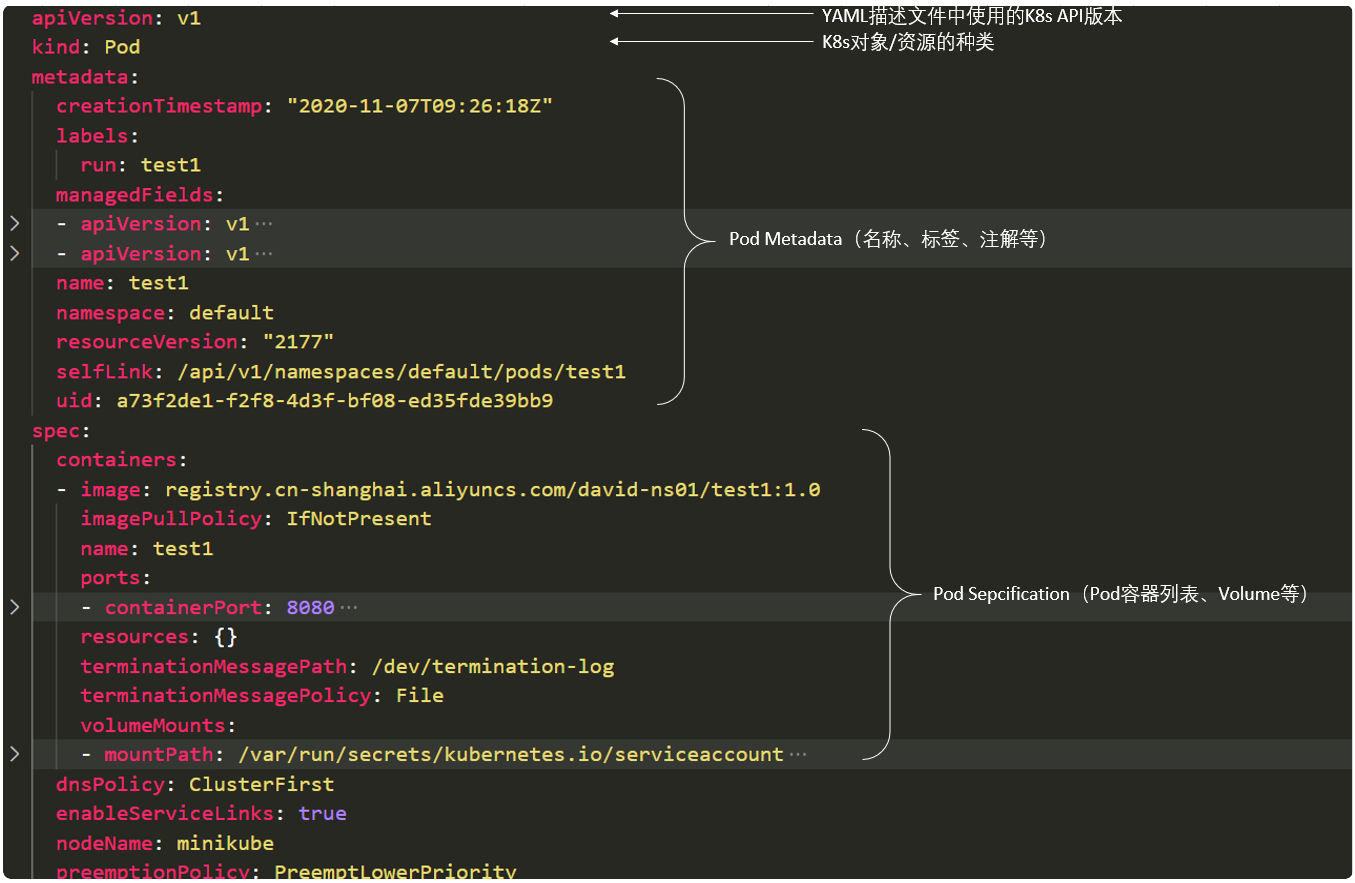
通常我们会发送一个JSON或者YAML格式的清单（manifest）到Kubernetes的REST API端点来创建Pod和其他Kubernetes资源。当然也可以使用其他更简单的方式来创建资源（resource）对象，比如在之前的文章中使用的kubectl run命令，但是这些方法通常只能配置一组有限的属性。
如果通过YAML文件定义所有Kubernetes对象，我们可以将它们上传到版本控制系统，从而可以充分利用版本控制系统带来的好处。
在配置每种类型资源的各种属性之前，我们还需要理解Kubernetes API对象定义。可以参考Kubernetes的API参考文档了解更多信息：https://kubernetes.io/docs/reference/

Pod的YAML描述文件

之前我们已经创建了一个名为test1的Pod资源，现在让我们来看看这个Pod的YAML定义信息到底是什么样子的。
可以在kubectl get 命令后加上 –o yaml选项来获取整个YAML定义信息：
kubectl get po test1 –o yaml

```
[david@dhr-demo root]$ kubectl get po test1 -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-11-07T09:26:18Z"
  labels:
    run: test1
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:metadata:
        f:labels:
          .: {}
          f:run: {}
      f:spec:
        f:containers:
          k:{"name":"test1"}:
            .: {}
            f:image: {}
            f:imagePullPolicy: {}
            f:name: {}
            f:ports:
              .: {}
              k:{"containerPort":8080,"protocol":"TCP"}:
                .: {}
                f:containerPort: {}
                f:protocol: {}
            f:resources: {}
            f:terminationMessagePath: {}
            f:terminationMessagePolicy: {}
        f:dnsPolicy: {}
        f:enableServiceLinks: {}
        f:restartPolicy: {}
```

下面的图片中对YAML描述文件中一些关键的信息进行了标注：



这个文件看起来比较复杂，但是在实际创建Pod的时候，我们需要编写的YAML内容其实不多。

Pod定义信息的组成

Pod定义信息由几部分组成。就拿YAML文件来说，首先是YAML遵循的Kubernetes API的版本（apiVersion）以及它描述的资源种类（kind）。其次是几乎所有Kubernetes资源的定义信息中都可以找到的三个重要的区段：

- Metadata – 元数据包含了名称、命名空间、标签以及关于Pod的其他信息
- Spec – 描述了Pod的实际内容，比如Pod内的容器、Volumes以及其他数据
- Status – 包含了Pod最近被观察到的状态，比如Pod所处的条件、每个容器的描述、状态以及Pod的内部IP地址和其他基本信息，但该数据不一定是最新的。

Status区段包含只读的运行时数据，显示资源在某一刻的状态。我们在创建一个新的Pod时，不需要在YAML文件中提供这部分信息。

这三个区段代表了Kubernetes API对象的典型结构。

创建YAML描述文件

现在我们来创建一个名为test1–manual.yml的文件：

touch test1–manual.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: test1-manual
spec:
  containers:
  - image: registry.cn-shanghai.aliyuncs.com/david-ns01/test1:1.0
    name: test1
    ports:
    - containerPort: 8080
      protocol: TCP
```

该YAML描述文件遵循Kubernetes API v1版本。资源的种类为Pod，名称为test-manual。这个Pod包含一个基于registry.cn-shanghai.aliyuncs.com/david-ns01/test1:1.0镜像的容器。容器的名称为test1，在8080端口上进行监听。
在Pod的定义文件中我们显示地指定了容器端口，这样其他人就能很方便地看到每个Pod暴露的端口。当然，如果不指定端口，客户端仍然能通过端口连接到Pod。如果容器通过一个绑定到0.0.0.0地址的端口来接收连接，那么就可以被其他Pod连接到，即使这个端口没有显示地在Pod的spec区段指定。
显示地定义端口的另外一个好处是让我们可以为每个端口指定一个名字。

创建Pod

在准备好YAML文件后，我们可以使用kubectl create命令来创建Pod：
kubectl create –f test1–manual.yaml

```
[david@dhr-demo ~]$ kubectl create -f test1-manual.yaml
pod/test1-manual created
```

除了创建Pod，kubectl create –f 命令还可以基于YAML或者JSON格式文件来创建任何资源。
创建完我们的Pod后，可以通过如下命令来查看该Pod的所有YAML描述信息：
kubectl get po test1–manual –o yaml

```
[david@dhr-demo root]$ kubectl get po test1-manual -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-11-14T04:04:45Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:spec:
        f:containers:
          k:{"name":"test1"}:
            .: {}
            f:image: {}
            f:imagePullPolicy: {}
            f:name: {}
            f:ports:
              .: {}
              k:{"containerPort":8080,"protocol":"TCP"}:
                .: {}
                f:containerPort: {}
                f:protocol: {}
            f:resources: {}
            f:terminationMessagePath: {}
            f:terminationMessagePolicy: {}
        f:dnsPolicy: {}
        f:enableServiceLinks: {}
        f:restartPolicy: {}
        f:schedulerName: {}
        f:securityContext: {}
        f:terminationGracePeriodSeconds: {}
```

虽然刚创建的Pod是基于YAML的，但是我们也可以通过如下命令让其返回JSON格式的Pod信息：

kubectl get pod test1–manual –o json

```
[david@dhr-demo root]$ kubectl get pod test1-manual -o json
{
    "apiVersion": "v1",
    "kind": "Pod",
    "metadata": {
        "creationTimestamp": "2020-11-14T04:04:45Z",
        "managedFields": [
            {
                "apiVersion": "v1",
                "fieldsType": "FieldsV1",
                "fieldsV1": {
                    "f:spec": {
                        "f:containers": {
                            "k:{\"name\":\"test1\"}": {
                                ".": {},
                                "f:image": {},
                                "f:imagePullPolicy": {},
                                "f:name": {},
                                "f:ports": {
                                    ".": {},
                                    "k:{\"containerPort\":8080,\"protocol\":\"TCP\"}": {
                                        ".": {},
                                        "f:containerPort": {},
                                        "f:protocol": {}
                                    }
                                },
                                "f:resources": {},
                                "f:terminationMessagePath": {},
                                "f:terminationMessagePolicy": {}
                            }
```

查看Pod的运行状态：

kubectl get pods

```
[david@dhr-demo root]$ kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
test1          1/1     Running   0          7d3h
test1-manual   1/1     Running   0          8h
[david@dhr-demo root]$
```

可以看到我们手动创建的名为test1-manual的Pod这个处于运行状态中。

查看应用的日志

我们的Node.js应用会将日志写到进程的标准输出。容器化的应用通常会将日志写到标准输出和标准错误流中，而不是写到文件中。通过这种方式，用户就能够以一种简单、标准的方式查看不同应用的日志信息。

容器运行时（Container Runtime），比如Docker，会将这些输出流重定向到文件中，使我们通过如下命令就可以获取到容器的日志：

docker logs <container id>

我们可以使用ssh登录到pod所在的工作节点，然后通过docker logs命令获取应用日志，但是Kubernetes为我们提供了一种更简单的方式。这种方式不需要通过ssh登录，我们只需要在本地机器上运行如下命令即可：

kubectl logs test1-manual

```
[david@dhr-demo root]$ kubectl logs test1-manual
开始运行...
[david@dhr-demo root]$
```

我们还未向这个应用程序发送任何请求，因此上面只输出了一条日志。可以发现，如果Pod只包含一个容器的话，在Kubernetes查看应用的日志是如此的简单。

每当容器日志文件达到10M时就会自动轮替。kubectl log命令只会显示最近一次轮替后的日志。

如果Pod包含多个容器的话，我们就需要显示地在kubectl log命令后加上-c 参数来指定容器名字：

kubectl logs test1-manual -c test1

发送请求到Pod

在之前的章节中，我们通过kubectl expose命令创建一个服务的方式来访问Pod。当然，为了测试以及调试的目的，我们还可以使用端口转发的方式。

可以执行如下命令来配置端口转发到Pod：

kubectl port-forward test1-manual 8003:8080

```
[david@dhr-demo root]$ kubectl port-forward test1-manual 8003:8080
Forwarding from 127.0.0.1:8003 -> 8080
Forwarding from [::1]:8003 -> 8080
```

上面的命令会将本地的8003端口转发到Pod test1-manual的8080端口。

现在我们就可以通过本地端口连接到Pod了。

curl localhost:8003

```
[root@dhr-demo ~]# curl localhost:8003
已发送消息至： test1-manual
[root@dhr-demo ~]#
```