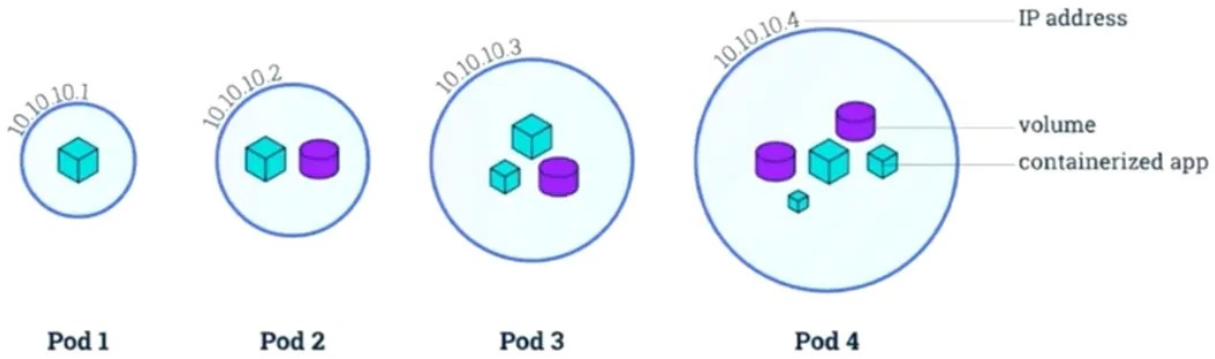
K8S最小调度单位Pod详解

k8s里面非常重要的一个概念pod，首先简单的介绍是pod是k8s最小的调度单位，一个pod里面可以包含一个或者多个container，一个pod共享一个namespace，它们之前可以通过localhost来进行通信。

> - docker：Namespace 做隔离，Cgroups 做限制，rootfs做文件系统。
> - 容器本质是进程，而k8s是操作系统。
> - pod就是类似于进程组。
> - 部署的一些应用有着类似“进程”和“进程组”的关系，必须部署在一台机器上，受限于容器的单进程。



为什么需要Pod?

Pod，是Kubernetes项目中的原子调度单位。容器就是未来云计算系统中的进程；容器镜像就是系统里的.exe安装包，Kubernetes就是操作系统。在一个真正的操作系统里，进程并不是“孤苦伶仃”地独自运行，而是以进程组的方式，“有原则”的组织在一起。在kubernetes中，“进程组”所映射的概念就是Pod。Google的工程师发现，他们部署的应用，往往都存在着类似于“进程和进程组”的关系，也就是说这些应用之间有着亲密的协作关系，使得他们必须部署在同一台机器上。

Pod的实现原理

Pod只是一个逻辑概念。

Pod其实是一组共享了某些资源的容器，具体来说，Pod里的所有容器，共享的是同一个Network Namespace，并且可以声明共享同一个Volume。在Kubernetes中，Pod的实现需要使用一个中间容器，这个容器叫做Infra容器。在Pod中，Infra容器永远是第一个被创建的容器，而其他用户定义的容器，则通过Join Network Namespace的方式与Infra容器关联在一起。

Infra容器使用的是一个非常特殊的镜像，叫做：k8s.gcr.io/pause。这个镜像是一个用汇编语言编写的、永远处于暂停状态的容器。在Infra容器“Hold住”NetworkNamespace之后，用户容器就可以加入到Infra容器的NetworkNamespace中了。这也就意味着，对于Pod中的容器A和容器B：

- 它们可以直接使用localhost进行通信
- 它们看到的网络设备和Infra容器看到的完全一样
- 一个Pod只有一个IP地址，也就是这个Pod的Network Namespace对应的IP地址
- 所有的网络资源，都是一个Pod一份，并且被该Pod内的所有容器共享
- Pod的生命周期只和Infra容器一致，与容器A和容器B无关

对于Pod共享Volume来说，Kubernetes只要把所有Volume定义设计在Pod层即可。例如：

Recommend
火龙果软件

```
apiVersion: v1
kind: Pod
metadata:
 name: two-containers
spec:
 restartPolicy: Never #Pod的重启策略。1.Always:容器失效时，kubelet自动重启该容器。2.OnFailure：容器终止运行且退出码不为0时重启。3.Never：无
 volumes:
 - name: shared-data
   hostPath:
    path : /data
 containers:
 - name: nginx-container
   image: nginx
   volumeMounts:
   - name: shared-data
     mountPath: /usr/share/nginx/html
 - name: debian-container
   image: debian
   volumeMounts:
   - name: shared-data
     mountPath: /pod-data
   command: ["/bin/sh"]
   args: ["-c","echo Hello from the debian container > /pod/data/index.html"]
```

Pod对象的生命周期

Pod生命周期变化主要体现在Pod API对象的Status部分。

- Pending，这个状态意味着Pod的YAML文件已经提交给了Kubernetes，API对象已经被创建并被保存在etcd中。但是，这个Pod里有些容器因为某种原因不能被成功创建，比如，调度不成功
- Running，这个状态下，Pod已经调度成功，和一个具体的节点绑定。它包含的容器都已经创建成功，并且至少有一个正在运行中。
- Succeed，这个状态意味着，Pod中所有容器都正常运行完毕，并且已经退出了。
- Failed，这个状态下，Pod里至少有一个容器以不正常的状态（非0状态码）退出。这个状态的出现意味着你要想办法Debug这个容器的应用，比如查看Pod的Events和日志
- Unknown，这是一个异常状态，意味着Pod的状态不能持续被kubelet汇报给kube-apiserver，这很可能是Master和kubelet间的通信出了问题。

演示pod

- pod_nginx.yml 文件

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: hub.c.163.com/library/nginx:latest
    ports:
    - containerPort: 80
```

- 开始运行pod节点

Recommend
火龙果软件

```
kubectl version
#创建pod
kubectl create -f pod_nginx.yml
#查看pod状态
kubectl get pods
#查看pod状态ip状态
kubectl get pods -o wide
```

```
[root@node51 ~]# kubectl version
Client Version: version.Info{Major:"1", Minor:"9", GitVersion:"v1.9.0", GitCommit:"925c127ec6b946659ad0fd596fa959be43f0cc05", GitTreeS
tate:"clean", BuildDate:"2017-12-15T21:07:38Z", GoVersion:"go1.9.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"9", GitVersion:"v1.9.0", GitCommit:"925c127ec6b946659ad0fd596fa959be43f0cc05", GitTreeS
```

```
[root@node51 ~]# kubectl get pods
NAME                                    READY     STATUS              RESTARTS   AGE
kubernetes-bootcamp-6b7849c495-v4fnk    0/1       ImagePullBackOff    0          11h
nginx                                   0/1       Pending             0          1m
[root@node51 ~]# kubectl get pods -o wide
NAME                                    READY     STATUS              RESTARTS   AGE       IP              NODE
```

```
[root@node51 ~]# kubectl get pods
NAME                                    READY     STATUS              RESTARTS   AGE
kubernetes-bootcamp-6b7849c495-v4fnk    1/1       Running             0          11h
nginx                                   0/1       ContainerCreating   0          2m
[root@node51 ~]# kubectl get pods
NAME                                    READY     STATUS    RESTARTS   AGE
kubernetes-bootcamp-6b7849c495-v4fnk    1/1       Running   0          11h
nginx                                   1/1       Running   0          2m
[root@node51 ~]#
[root@node51 ~]# kubectl get pods -o wide
NAME                                    READY     STATUS    RESTARTS   AGE       IP              NODE
kubernetes-bootcamp-6b7849c495-v4fnk    1/1       Running   0          11h       172.20.38.192   192.168.18.153
```

- 进入192.168.18.152查看docker ps，内部其实就是docker

```
minikube start
docker ps
docker exec -it 2ff4d852ed8e bash
exit
```

```
[root@node52 ~]# docker ps
CONTAINER ID        IMAGE                                                   COMMAND                  CREATED
 STATUS              PORTS               NAMES
2ff4d852ed8e        hub.c.163.com/library/nginx                             "nginx -g 'daemon of…"   5 hours ago
 Up 5 hours                              k8s_nginx_nginx_default_eed7e396-3748-11ea-a775-000c2927d058_0
547af6dcbdf4        registry.cn-hangzhou.aliyuncs.com/imooc/pause-amd64:3.0 "/pause"                 5 hours ago
 Up 5 hours                              k8s_POD_nginx_default_eed7e396-3748-11ea-a775-000c2927d058_0
```

```
[root@node52 ~]# docker exec -it 2ff4d852ed8e bash
root@nginx:/# ll
bash: ll: command not found
root@nginx:/# ls
```

- 查看pod nginx的网络

```
docker network ls
#查看到nginx的ip是172.17.0.4
docker network inspect bridge
exit
kubectl get pods -o wide
```

Recommend
火龙果软件

```
[root@node51 ~]# kubectl get pods -o wide
NAME                                    READY     STATUS    RESTARTS   AGE       IP              NODE
kubernetes-bootcamp-6b7849c495-v4fnk    1/1       Running   0          17h       172.20.38.192   192.168.18.153
nginx                                   1/1       Running   0          5h        172.20.2.65     192.168.18.152
[root@node51 ~]#
[root@node51 ~]# ssh 192.168.18.152
Last login: Wed Jan 15 16:24:07 2020 from 192.168.18.1
[root@node52 ~]#
[root@node52 ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
79b2dd9fa6f7        bridge              bridge              local
837db9172363        host                host                local
b56db6750a55        none                null                local
[root@node52 ~]# dokcer network inspect bridge
-bash: dokcer: 未找到命令
[root@node52 ~]# docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "79b2dd9fa6f7acaecfaed25efa9082356a4a3d425f699f69d65f4f5347423201",
        "Created": "2020-01-14T23:17:26.667031125+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
```

```
        "ConfigOnly": false,
        "Containers": {
            "3af57794d0e85cd3fc12522f7fb7be1baef7bf1be56f47dd08b0a812b3fdcd2f": {
                "Name": "k8s_POD_nginx_default_c744e679-98c0-11e8-a1da-080027087946_0",
                "EndpointID": "476c6b3639c08d78e55a94b9543379bf277b74d19f96343478d61d54d7551fda",
                "MacAddress": "02:42:ac:11:00:04",
                "IPv4Address": "172.17.0.4/16",
                "IPv6Address": ""
            },
            "496625351ee206ae9f74bc38b9e4ebe3377c1590ac0f7d0b47153e7729be63cd": {
                "Name": "k8s_POD_kubernetes-dashboard-685cfbd9f6-ghcn6_kube-system_036f53b1-980c-11e8-95f8-080027087946_2",
                "EndpointID": "ef57c74883b4b6a72cfd6eac19c6ff07128c4fa2cd1b7b092492f3a8f2af1b79",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            },
            "55e718f0b407c941a298d09fc7f0d368dcc3215c4c81dbbbfe0155849be75976": {
                "Name": "k8s_POD_kube-dns-b4bd9576-p9ffj_kube-system_fd01f700-980b-11e8-95f8-080027087946_2",
                "EndpointID": "10e3399f35942442003a87304ea7e14a37bfc5fada59e77961024b1d4c7ddee8",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
$ exit
logout

G:\docker\No.9\labs\pod-basic>kubectl get pods -o wide
NAME      READY     STATUS    RESTARTS   AGE       IP          NODE
nginx     1/1       Running   0          7m        172.17.0.4  minikube
```

Recommend
火龙果软件

上边这种是不是感觉很麻烦，如果想进入容器，还需要想进入虚拟机然后docker ps 找到id，最后docker exec的方式。其实不用那么复杂。

```
#进入nginx的容器
kubectl exec -it nginx sh
#了解下进入容器的所有的命令
kubectl exec -it -h
```

```
G:\docker\No.9\labs\pod-basic>kubectl exec -it nginx sh
# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
#
```

```
G:\docker\No.9\labs\pod-basic>kubectl exec -it -h
Execute a command in a container.

Examples:
  # Get output from running 'date' from pod 123456-7890, using the first container by default
  kubectl exec 123456-7890 date

  # Get output from running 'date' in ruby-container from pod 123456-7890
  kubectl exec 123456-7890 -c ruby-container date

  # Switch to raw terminal mode, sends stdin to 'bash' in ruby-container from pod 123456-7890
  # and sends stdout/stderr from 'bash' back to the client
  kubectl exec 123456-7890 -c ruby-container -i -t -- bash -il

  # List contents of /usr from the first container of pod 123456-7890 and sort by modification ti
  # If the command you want to execute in the pod has any flags in common (e.g. -i),
  # you must use two dashes (--) to separate your command's flags/arguments.
  # Also note, do not surround your command and its flags/arguments with quotes
  # unless that is how you would execute it normally (i.e., do ls -t /usr, not "ls -t /usr").
  kubectl exec 123456-7890 -i -t -- ls -t /usr

Options:
  -c, --container='': Container name. If omitted, the first container in the pod will be chosen
```

```
# nginx这个pods的所有详细信息
kubectl describe pods nginx
```

Recommend
火龙果软件

```
G:\docker\No.9\labs\pod-basic>kubectl describe pods nginx
Name:         nginx
Namespace:    default
Node:         minikube/10.0.2.15
Start Time:   Sun, 05 Aug 2018 23:03:58 +0800
Labels:       app=nginx
Annotations:  <none>
Status:       Running
IP:           172.17.0.4
Containers:
  nginx:
    Container ID:   docker://5db574698eeeb34f65db38baef36d512cd1916ef137478c8b8a5c0696f9fadcc
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:d85914d547a6c92faa39ce7058bd7529baacab7e0cd4255442b04577c4d1f424
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 05 Aug 2018 23:04:07 +0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-wxdkf (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  default-token-wxdkf:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-wxdkf
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                 Age   From               Message
  ----    ------                 ----  ----               -------
  Normal  Scheduled              14m   default-scheduler  Successfully assigned nginx to minikube
  Normal  SuccessfulMountVolume  14m   kubelet, minikube  MountVolume.SetUp succeeded for volume "default-token-wxdkf"
  Normal  Pulling                14m   kubelet, minikube  pulling image "nginx"
  Normal  Pulled                 14m   kubelet, minikube  Successfully pulled image "nginx"
  Normal  Created                14m   kubelet, minikube  Created container
```

通过ip的方式，看看nginx启动起来没？

```
minikube ssh
ping 172.17.0.4
curl 172.17.0.4
```

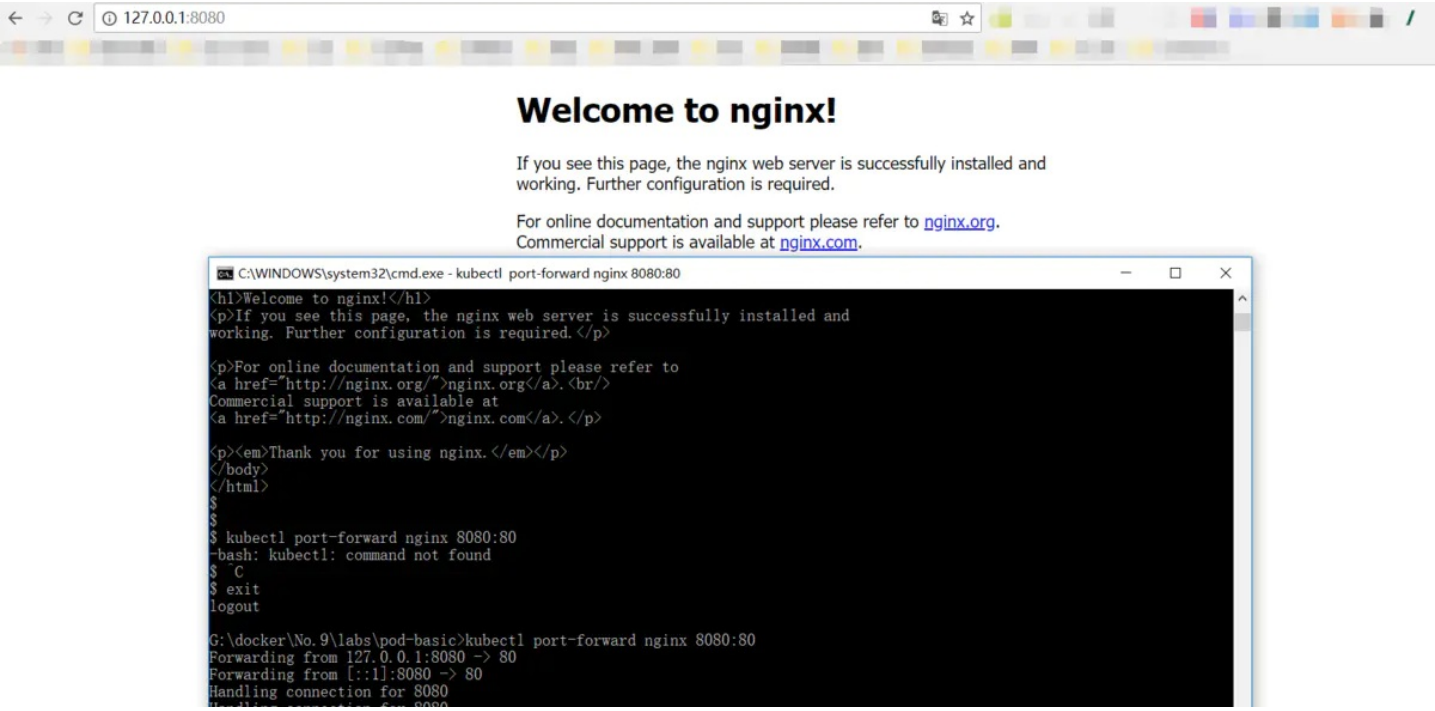
Recommend
火龙果软件

```
G:\docker\No.9\labs\pod-basic>minikube ssh

$ ping 172.17.0.4
PING 172.17.0.4 (172.17.0.4): 56 data bytes
64 bytes from 172.17.0.4: seq=0 ttl=64 time=0.219 ms
64 bytes from 172.17.0.4: seq=1 ttl=64 time=0.073 ms
^C
--- 172.17.0.4 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.073/0.146/0.219 ms
$ curl 172.17.0.4
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
```

> 老铁 现在你最想的是什么？我最想的是：在本机可以访问到nginx的服务，而不是在内部。在原来学docker的时候可以进行-p 端口映射的方式。在k8s的kubectl 同样可以看命令。

```
kubectl port-forward nginx 8080:80
```

Recommend
火龙果软件



PS：最后把容器内的nginx暴露出来了可以在外部访问了，是不是觉得美滋滋，这次主要讲了如果pod的方式创建容器，由k8s进行管理，获取到nginx的ip地址，映射容器内的端口，其实我感觉有点类似docker-swarm,但是这个好像比docker-swarm灵活点。也留个小问题，现在是用的port-forward它得一直开着，如果关闭了，直接就访问不了了，这种该如何解决。下次我来说！

附件：掌握Pod YAML文件定义详解

```
apiVersion:v1
kind: Pod
metadata:
  name: string
  namespace: string
  lables:
  - name:string
  annotations:
  - nameL string
spec:
  containers:
  - name: string
    image: string
    imagePullPolicy: [Always| Never| IFNotPresent]
    commadn: [string]
    args: [String]
    workingDir: string
    volumeMounts:
    - name: string
      mountPath: string
      readOnly: boolean
    ports:
    - name: string
      containerPort: int
      hostPort: int
      protocol: string
    env:
    - name:string
```

Recommend
火龙果软件

```
    value: string
  resources:
    limits:
      cpu: string
      memory: string
    requests:
      cpu: string
      memory: string
  livenessFrobe:
    exec:
      commadn: [string]
    httpGet:
      path: string
      port: number
      host: string
      scheme: string
      httpHeaders:
      - name: string
        value: string
      tcpSocket:
        port: number
      initialDelaySeconds: 0
      timeoutSeconds: 0
      periodSeconds: 0
      successThreshold: 0
      failureThreshold: 0
    securityContext:
      privileged: false
  restartFolicy: [Always| Never|| OnFaliure]
  nodeSelector: object
  imagePullSecrets:
  - name: string
  hostNetwork: false
  volumes:
  - name: string
    enptyDir: {}
    hostPath:
      path: string
    secret:
      secretName: string
      items:
      - key: string
        path: string
      configMap:
        name: string
        items:
        - key: string
          path: string
```

配置文件详细清单：

属性名称	取值范围	是否必选(1必选)	取值说明
[illegible]	string	1	版本号 例如：v1

Recommend
火龙果软件

属性名称	取值范围	是否必选(1必选)	取值说明
metadata	Object	1	元数据
metadata.name	String	1	Pod的名称，命令规范需要符合RFC 1035规范
metadata.namespace	String	1	Pod的命名空间，默认值为default
metadata.labels[]	List		自定义标签列表
metadata.annotation[]	List		自定义注释列表
Spec	Object	1	Pod中容器的详细定义
spec.containers[]	List	1	Pod中的容器列表
spec.containers[].name	String	1	容器的名称，需要符合RFC 1035规范
spec.containers[].image	String	1	容器的镜像名称
spec.containers[].imagePullPolicy	String		镜像拉取策略，可选值包括：Always、Nerver、IfNotPresent,默认值为Always。 (1) Always.表示每次都尝试重新拉取镜像。 (2)IfNotPresent:表示如果本地有该镜像，则使用本地的镜像，本地不存在时拉取镜像。 (3)Nerver:表示仅使用本地镜像。 包含如下设置，系统默认设置为Always，如下所述 (1)不设置imagePullPolicy,也未指定镜像的tag; (2)不设置imagePullPolicy,镜像tag为latest (3)启用名为AlwaysPullImages的准入控制器（Admission Controller）
spec.containers[].command[]	List		容器的启动命令列表，如果不指定，则使用镜像打包时使用的启动命令
spec.containers[].args[]	List		容器的启动命令参数列表
spec.containers[].workingDir	String		容器的工作目录
spec.containers[].volumeMounts[]	List		挂载到容器内部的存储卷配置
spec.containers[].volumeMounts[].name	String		引用Pod定义的共享存储卷的名称，需要使用volumes[]部分定义的共享存储卷名称
spec.containers[].volumeMounts[].mountPath	String		存储卷在容器内Mount的绝对路径，应少于512个字符
spec.containers[].volumeMounts[].readOnly	Boolean		是否为只读模式，默认为读写模式
spec.containers[].ports[]	List		容器需要暴露的端口号列表
spec.containers[].ports[].name	String		端口的名称
spec.containers[].ports[].containerPort	Int		容器需要监听的端口号
spec.containers[].ports[].hostPort	Int		容器所在主机需要监听的端口号，默认与containerPort相同，设置hostPort时，同一台宿主机将无法启动该容器的第2份副本
spec.containers[].ports[].protocol	String		端口协议，支持TCP和UDP，默认值为TCP
spec.containers[].env[]	List		容器运行前需要设置的环境变量列表
spec.containers[].env[].name	String		环境变量的名称
spec.containers[].env[].value	String		环境变量的值
spec.containers[].resources	Object		资源限制和资源请求的设置

Recommend
火龙果软件

属性名称	取值范围	是否必选(1必选)	取值说明
spec.containers[].resources.limits	Object		资源限制的设置
spec.containers[].resources.limits.cpu	String		CPU限制，单位为core数，将用于docker run --cpu-shares参数
spec.containers[].resources.limits.memory	String		内存限制，单位可以为MIB、GIB等。将用于docker run --memory
spec.containers[].resources.requests	Object		资源限制设置
spec.containers[].resources.requecsts.cpu	String		CPU请求，单位为core数，容器启动的初始可用数量
spec.containers[].resources.requests.memory	String		内存请求，单位可以为MIB、GIB等，容器启动的初始可用数量
spec.volumes[]	List		在该Pod上定义的共享存储卷列表
spec.volumes[].name	String		共享存储卷名称，在一个Pod中每个存储卷定义一个名称，容器定义部分的containers[].volumeMounts[].name将应用改共享存储卷的名称。 volume的类型包括：emptyDir、hostPath、gcePersistentDisk、awsElasticBlockStore、gitRepo、sercret、nfs、iscsi、glusterfs、persistentVolumeClaim、rbd、flexVolume、cinder、cephfs、flocker、downwardAPI、fc、azureFile、configMap、vsphereVolume，可以定义多个Volume，每个Volume的name保持唯一。
spec.volumes[].emptyDir	Object		类型为emptyDir的存储卷，表示与Pod同生命周期的一个临时目录，其值为一个空对象：emptyDir{}
spec.volumes[].hostPath	Object		类型为hostPath的存储卷，表示挂载Pod所在宿主机的目录，通过volumes[].hostPath.path指定
spec.volumes[].hostPath.path	String		Pod所在主机的目录，将被用于容器中的mount的目录
spec.volumes[].secret	Object		类型为secret存储卷，表示挂载集群预定义的secret对象到容器内部
spec.volumes[].configMap	Object		类型为configMap的存储卷，表示挂载激情预定义的configMap对象到容器内部
spec.volumes[].livenessProbc	Object		对Pod内各容器健康检查的设置，当探测无响应几次之后，系统将自动重启该容器，可以设置的方法包括：exec、httpGet、和tcpSocket。对一个容器仅需设置一种健康检查方法。
spec.volumes[].livenessProbe.exec	Object		对Pod内各容器健康检查的设置，exec方式
spec.volumes[].livenessProbe.exec.command[]	String		exec方式需要制定的命令或者脚本
spec.volumes[].livenessProbe.httpGet	Object		对Pod内各种容器健康检查设置，HTTPGet方式，需要指定path、port
spec.voulumes[].livenessProbe.tcpSocket	Object		对Pod内各容器健康检查的设置，tcpSocket方式
spec.volumes[].livenessProbe.initiaDelaySeconds	Number		容器启动完成后首次探测的时间，单位为s
spec.volumes[].livenessProbe.timeoutSeconds	Number		对容器健康检查的探测等待响应的超时时间设置。单位为s，默认值为1s。如超过该超时时间设置，则将认为该容器不健康，会重启该容器。
spec.volumes[].livenessProbe.PeriodSeconds	Number		对容器健康检查的定期探测时间设置，单位为s，默认10s探测一次

Recommend
火龙果软件

属性名称	取值范围	是否必选(1必选)	取值说明
spec.restartPolicy	String		Pod的重启策略。可选值为Always，OnFailure，Never 默认值为Always。 （1）Always：Pod一旦终止运行，则无论容器是如何终止的，kubectl都将重启它 （2）OnFailure:只有Pod以非零退出码终止时，kubectl才会重启该容器。如果容器正常结束，则kubectl将不会重启它 （3）Never:Pod终止后，kubectl将退出码报告给Master，不会再重启该Pod
spec.nodeSelector	Object		设置Node的Label，以key-value格式指定，Pod将被调度到具有这些Label的Node上
spec.imagePullSecrets	Object		pull镜像时使用的Secret名称，以name：secretkey格式指定
spec.hostNetwork	Boolean		是否使用主机网络模式，默认值为false，设置为true表示容器使用宿主机网络，不再使用Docker网桥，该Pod将无法在同一台宿主机上启动第2个副本。

Recommend
火龙果软件