## **1、流程**

master节点（Control Plane【控制面板】）：master节点控制整个集群

**master节点上有一些核心组件**：

- Controller Manager：控制管理器

- etcd：键值数据库（redis）【记账本，记事本】

- scheduler：调度器

- api server：api网关（所有的控制都需要通过api-server）

**node节点（worker工作节点）：**

- kubelet（监工）：每一个node节点上必须安装的组件。

- kube-proxy：代理。代理网络

**部署一个应用？**

程序员：调用CLI告诉master，我们现在要部署一个tomcat应用

● 程序员的所有调用都先去master节点的网关api-server。这是matser的唯一入口（mvc模式中的c层）

● 收到的请求先交给master的api-server。由api-server交给controller-mannager进行控制

● controller-mannager 进行 应用部署

● controller-mannager 会生成一次部署信息。 tomcat --image:tomcat6 --port 8080 ,真正不部署应用

●部署信息被记录在etcd中

● scheduler调度器从etcd数据库中，拿到要部署的应用，开始调度。看哪个节点合适，

●scheduler把算出来的调度信息再放到etcd中

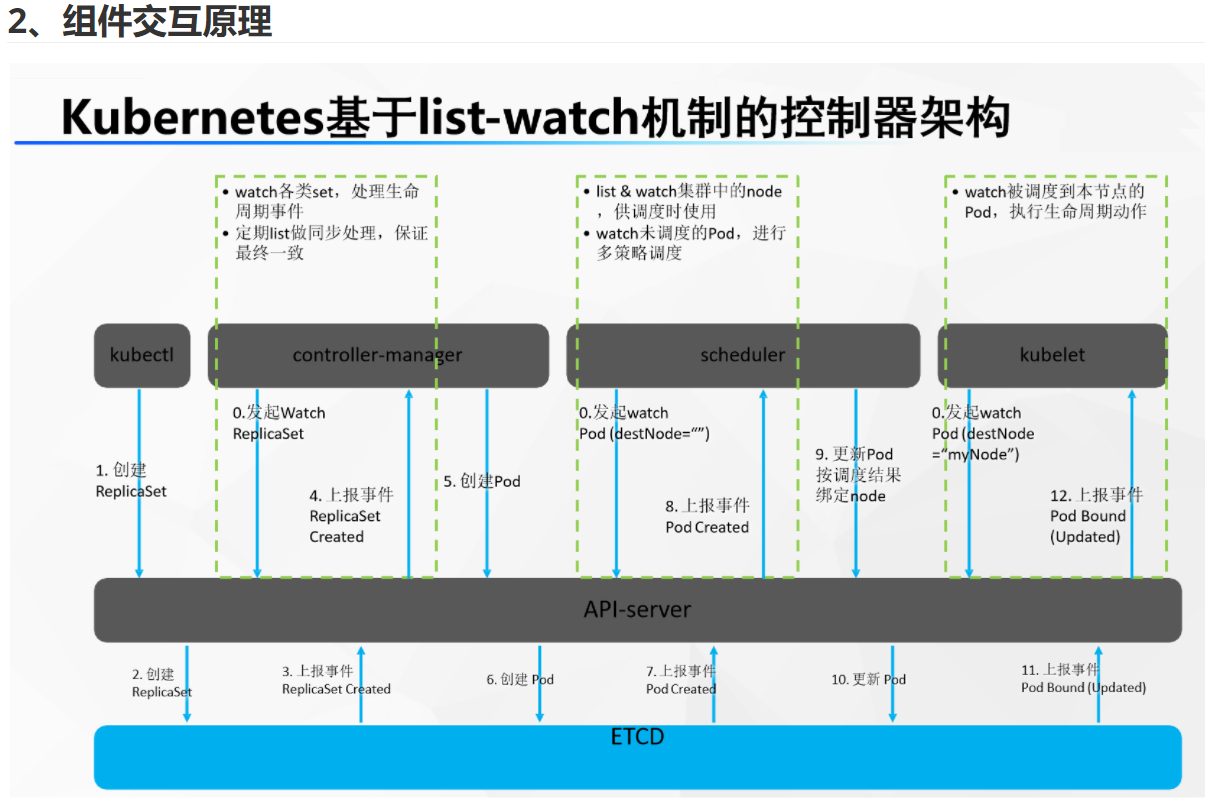
●每一个node节点的监控kubelet，随时和master保持联系的（给api-server发送请求不断获取最新数据），所有节点的kubelet就会从master

●假设node2的kubelet最终收到了命令，要部署。

●kubelet就自己run一个应用在当前机器上，随时给master汇报当前应用的状态信息，分配ip

●node和master是通过master的api-server联系的

●每一个机器上的kube-proxy能知道集群的所有网络。只要node访问别人或者别人访问node，node上的kube-proxy网络代理自动计算进行流量转发





api-server保存etcd，controller-manager最终解析数据，知道集群要my-nginx一份，保

存到etcd

**Kubernetes 中创建一个 Pod 涉及多个组件之间联动，主要流程如下：**

1. 客户端提交 Pod 的配置信息（可以是 yaml 文件定义的信息）到 kube-apiserver。

2. Apiserver 收到指令后，通知给 controller-manager 创建一个资源对象。

3. Controller-manager 通过 api-server 将 pod 的配置信息存储到 ETCD 数据中心中。

4. Kube-scheduler 检测到 pod 信息会开始调度预选，会先过滤掉不符合 Pod 资源配置要求的节点，

然后开始调度调优，主要是挑选出更适合运行 pod 的节点，然后将 pod 的资源配置单发送到

node 节点上的 kubelet 组件上。

5. Kubelet 根据 scheduler 发来的资源配置单运行 pod，运行成功后，将 pod 的运行信息返回给scheduler，scheduler 将返回的 pod 运行状况的信息存储到 etcd 数据中心。

## **2、Service的类型**

|  |
| --- |
| ClusterlP #向集群内部暴露服务 为当前Service分配或者不分配集群IP。负载均衡一组Pod  NodePort #通过宿主主机IP和端口向外部暴露服务 外界也可以使用机器ip+暴露的NodePort端口 访问。  ●nodePort端口由kube-proxy开在机器上  ●机器ip+暴露的NodePort 流量先来到 kube-proxy  LoadBalancer #通过弹性IP向外部暴露服务  ExtennalName #将外部的一个服务定义一个集群内部的别名 |

Service的名字还能当成域名被Pod解析，对宿主机无效。如curl service名字

## **3、NameSpace**

名称空间，用来对集群资源进行隔离划分。默认只隔离资源，不隔离网络

## **4、三种端口说明**

ports:

- name: http

  port: 80

  targetPort: 80

**nodePort: 30008**

  protocol: TCP

**nodePort**

外部机器（在windows浏览器）可以访问的端口；

比如一个Web应用需要被其他用户访问，那么需要配置type=NodePort，而且配置nodePort=30001，那么其他机器就可以通过浏览器访问scheme://node:30001访问到该服务；

**targetPort**

容器的端口，与制作容器时暴露的端口一致（Dockerfile中EXPOSE），例如docker.io官方的nginx暴露的是80端口；

**port**

Kubernetes集群中的各个服务之间访问的端口，虽然mysql容器暴露了3306端口，但外部机器不能访问到mysql服务，因为他没有配置NodePort类型，该3306端口是集群内其他容器需要通过3306端口访问该服务；

kubectl expose deployment tomcat6 --port=8912 --target-port=8080 --type=NodePort

## --port：集群内访问service的端口 8912

## --target-port： pod容器的端口 8080

## --nodePort： 每个机器开发的端口 30403

## **5、K8s 功能:**

**（1）自动装箱**

基于容器对应用运行环境的资源配置要求自动部署应用容器

**（2）自我修复(自愈能力)**

当容器失败时，会对容器进行重启

当所部署的 Node 节点有问题时，会对容器进行重新部署和重新调度

当容器未通过监控检查时，会关闭此容器直到容器正常运行时，才会对外提供服务

**（3）水平扩展**

通过简单的命令、用户 UI 界面或基于 CPU 等资源使用情况，对应用容器进行规模扩大

或规模剪裁

**（3）服务发现**

用户不需使用额外的服务发现机制，就能够基于 Kubernetes 自身能力实现服务发现和

负载均衡

**（4）滚动更新**

可以根据应用的变化，对应用容器运行的应用，进行一次性或批量式更新

**（5）版本回退**

可以根据应用部署情况，对应用容器运行的应用，进行历史版本即时回退

**（6）密钥和配置管理**

在不需要重新构建镜像的情况下，可以部署和更新密钥和应用配置，类似热部署。

**（7）存储编排**

自动实现存储系统挂载及应用，特别对有状态应用实现数据持久化非常重要

存储系统可以来自于本地目录、网络存储(NFS、Gluster、Ceph 等)、公共云存储服务

**（8）批处理**

提供一次性任务，定时任务；满足批量数据处理和分析的场景

## **6、PVC 和 PV**

PersistentVolume（PV）是集群中由管理员配置的一段网络存储。 它是集群中的资源，就

像节点是集群资源一样。 PV 是容量插件，如 Volumes，但其生命周期独立于使用 PV 的任

何单个 pod。 此 API 对象捕获存储实现的详细信息，包括 NFS，iSCSI 或特定于云提供程

序的存储系统。

PersistentVolumeClaim（PVC）是由用户进行存储的请求。 它类似于 pod。 Pod 消耗节点

资源，PVC 消耗 PV 资源。Pod 可以请求特定级别的资源（CPU 和内存）。声明可以请求特

定的大小和访问模式（例如，可以一次读/写或多次只读）。

## **7、探针**

**1、探针类型**

K8s 中存在两种类型的探针：liveness probe 和 readiness probe。

**2、liveness probe（存活探针）**

用于判断容器是否存活，即 Pod 是否为 running 状态，如果 LivenessProbe 探针探测到容

器不健康，则 kubelet 将 kill 掉容器，并根据容器的重启策略是否重启。如果一个容器不

包含 LivenessProbe 探针，则 Kubelet 认为容器的 LivenessProbe 探针的返回值永远成功。

有时应用程序可能因为某些原因（后端服务故障等）导致暂时无法对外提供服务，但应用

软件没有终止，导致 K8S 无法隔离有故障的 pod，调用者可能会访问到有故障的 pod，导致

业务不稳定。K8S 提供 livenessProbe 来检测应用程序是否正常运行，并且对相应状况进

行相应的补救措施。

**3、readiness probe（就绪探针）**

用于判断容器是否启动完成，即容器的 Ready 是否为 True，可以接收请求，如果

ReadinessProbe 探测失败，则容器的 Ready 将为 False，控制器将此 Pod 的 Endpoint 从对

应的 service 的 Endpoint 列表中移除，从此不再将任何请求调度此 Pod 上，直到下次探测

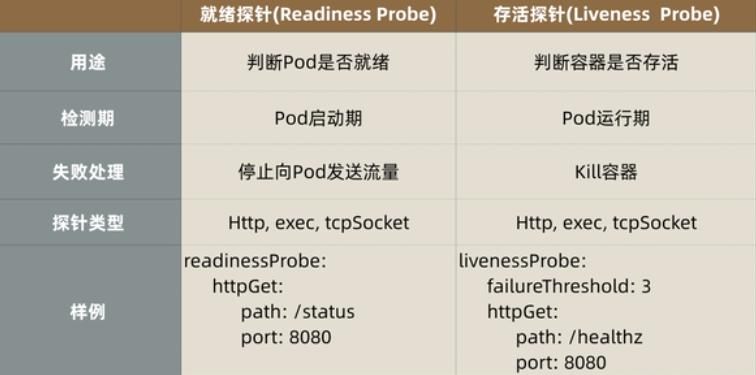
成功。通过使用 Readiness 探针，Kubernetes 能够等待应用程序完全启动，然后才允许服

务将流量发送到新副本。

比如使用 tomcat 的应用程序来说，并不是简单地说 tomcat 启动成功就可以对外提供服务

的，还需要等待 spring 容器初始化，数据库连接没连上等等。对于 spring boot 应用，默

认的 actuator 带有/health 接口，可以用来进行启动成功的判断

**4、每类探针都支持三种探测方法：**

（1）exec：通过执行命令来检查服务是否正常，针对复杂检测或无 HTTP 接口的服务，命

令返回值为 0 则表示容器健康。

（2）httpGet：通过发送 http 请求检查服务是否正常，返回 200-399 状态码则表明容器健

康。

（3）tcpSocket：通过容器的 IP 和 Port 执行 TCP 检查，如果能够建立 TCP 连接，则表明

容器健康。

**5、探针探测的结果**

（1）Success：Container 通过了检查。

（2）Failure：Container 未通过检查。

（3）Unknown：未能执行检查，因此不采取任何措施。

**6、Pod 重启策略：**

（1）Always: 总是重启 当容器失效时，由 kubelet 自动重启该容器

（2）OnFailure: 如果失败就重启 当容器终止运行且退出码不为 0 时，由 kubelet 自动重启该容器

（3）Never: 永远不重启 不论容器运行状态如何，kubelet 都不会重启该容器。

不同控制器的重启策略限制如下：

RC 和 DaemonSet：必须设置为 Always，需要保证该容器持续运行；

Job：OnFailure 或 Never，确保容器执行完成后不再重启；

kubelet：在 Pod 失效时重启，不论将 RestartPolicy 设置为何值，也不会对 Pod 进行健康检查。

**7、示例**

|  |
| --- |
| apiVersion: v1  kind: Pod  metadata:  name: goproxy  labels:  app: goproxy  spec:  containers:  - name: goproxy  image: k8s.gcr.io/goproxy:0.1  ports:  - containerPort: 8080  readinessProbe:  tcpSocket:  port: 8080  initialDelaySeconds: 5  periodSeconds: 10  livenessProbe:  tcpSocket:  port: 8080  initialDelaySeconds: 15  periodSeconds: 20 |

探针(Probe)有许多可选字段，可以用来更加精确的控制 Liveness 和 Readiness 两种探针

的行为。这些参数包括：

initialDelaySeconds：容器启动后第一次执行探测是需要等待多少秒。

periodSeconds：执行探测的频率。默认是 10 秒，最小 1 秒。

timeoutSeconds：探测超时时间。默认 1 秒，最小 1 秒。

successThreshold：探测失败后，最少连续探测成功多少次才被认定为成功。默认是 1。

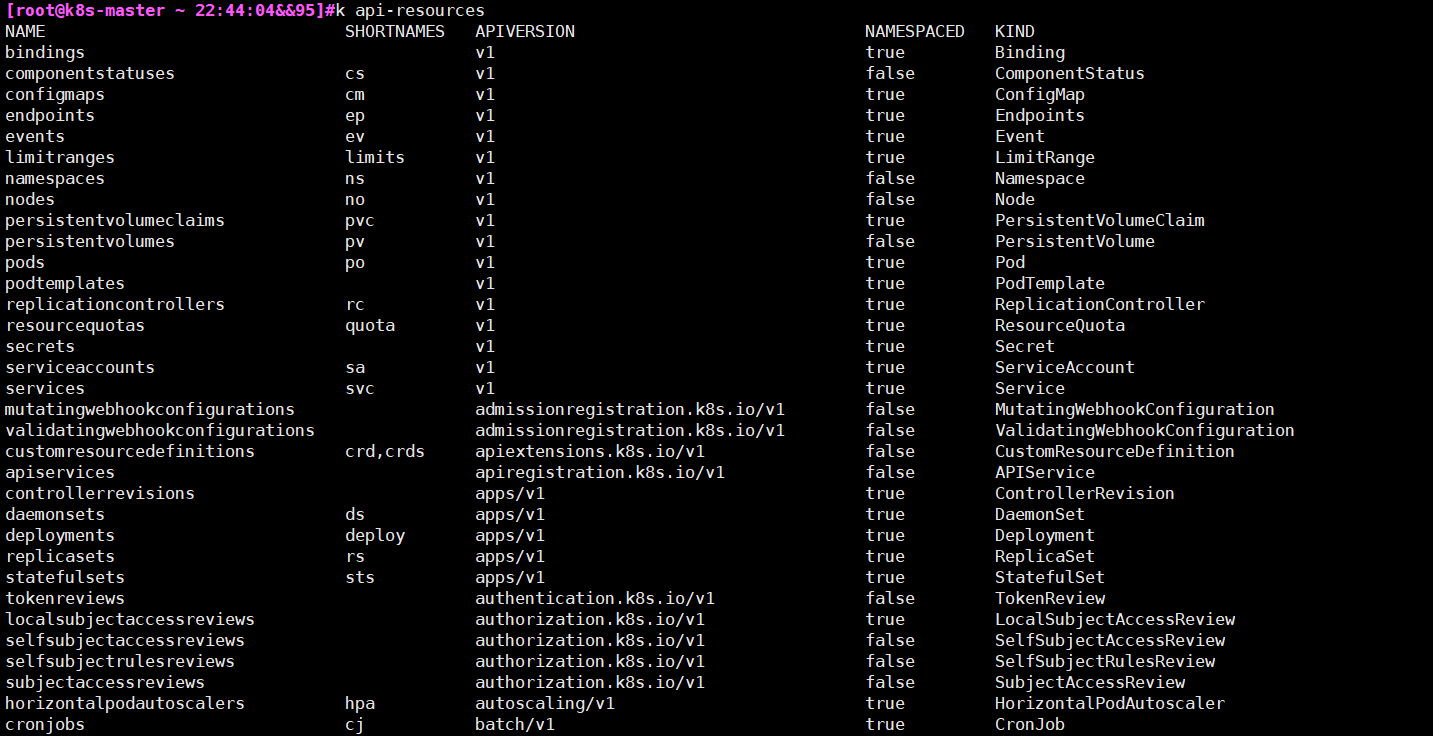
对于 liveness 必须是 1。最小值是 1。

failureThreshold：探测成功后，最少连续探测失败多少次才被认定为失败。默认是 3。

最小值是 1。

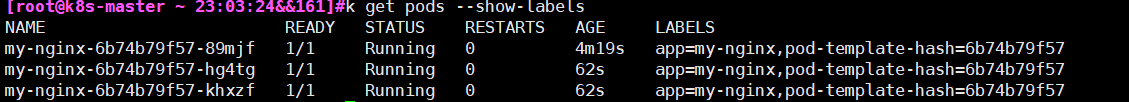
## **8、Kubelet启动的Pod每个都有Ip，全集群任意位置均可访问**

## **9、kubectl api-resources**



## **10、显示标签**

**k get pods --show-labels**



## **11、灰度发布：**

**新旧代码都有**

## **12、部署一个应用**

|  |
| --- |
| apiVersion: apps/v1 #与k8s集群版本有关，使用 kubectl api-versions 即可查看当前集群支持的版本  kind: Deployment #该配置的类型，我们使用的是 Deployment  metadata: #译名为元数据，即 Deployment 的一些基本属性和信息  name: nginx-deployment #Deployment 的名称  labels: #标签，可以灵活定位一个或多个资源，其中key和value均可自定义，可以定义多组，目前不需要理解  app: nginx #为该Deployment设置key为app，value为nginx的标签  spec: #这是关于该Deployment的描述，可以理解为你期待该Deployment在k8s中如何使用  replicas: 1 #使用该Deployment创建一个应用程序实例  selector: #标签选择器，与上面的标签共同作用，目前不需要理解  matchLabels: #选择包含标签app:nginx的资源  app: nginx  template: #这是选择或创建的Pod的模板  metadata: #Pod的元数据  labels: #Pod的标签，上面的selector即选择包含标签app:nginx的Pod  app: nginx  spec: #期望Pod实现的功能（即在pod中部署）  containers: #生成container，与docker中的container是同一种  - name: nginx #container的名称  image: nginx:1.7.9 #使用镜像nginx:1.7.9创建container，该container默认80端口可访问 |

## **13、暴露应用**

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: nginx-service #Service 的名称  labels: #Service 自己的标签  app: nginx #为该 Service 设置 key 为 app，value 为 nginx 的标签  spec: #这是关于该 Service 的定义，描述了 Service 如何选择 Pod，如何被访问  selector: #标签选择器  app: nginx #选择包含标签 app:nginx 的 Pod  ports:  - name: nginx-port #端口的名字  protocol: TCP #协议类型 TCP/UDP  port: 80 #集群内的其他容器组可通过 80 端口访问 Service  nodePort: 32600 #通过任意节点的 32600 端口访问 Service  targetPort: 80 #将请求转发到匹配 Pod 的 80 端口  type: NodePort #Serive的类型，ClusterIP/NodePort/LoaderBalancer |

## **14、编写yaml的黑科技**

## kubectl run my-nginx666 --image=nginx #启动一个Pod

1、kubectl get pod my-nginx666 -oyaml 集群中挑一个同类资源，获取出他的yaml。

2、kubectl run my-tomcat --image=tomcat --dry-run -oyaml 干跑一遍

当您在 Kubernetes 中创建一个对象时，您必须提供

该对象的 spec 字段，通过该字段描述您期望的 目标状态

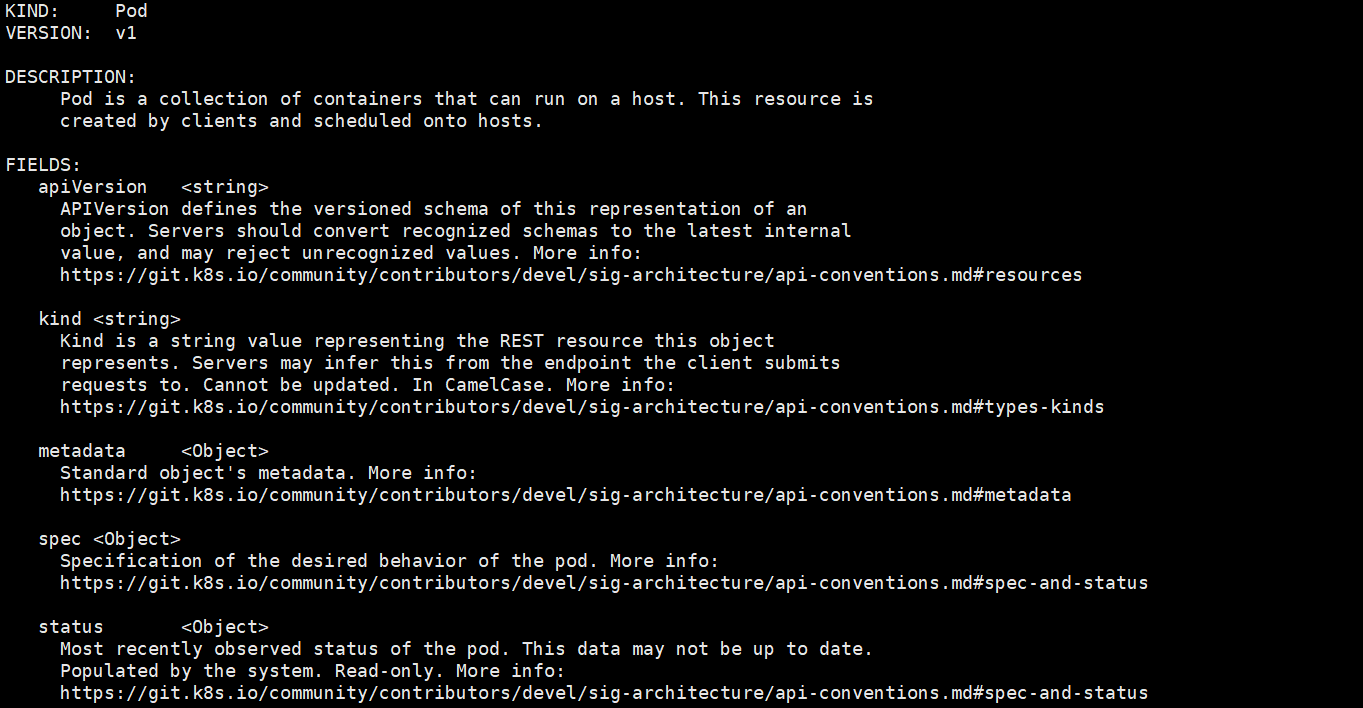
该对象的一些基本信息，例如名字

最重要的是containers内容的编写。

## **15、Explain**

比如不知道pod怎么编写

kubectl explain pod



比如不知道pod的metadata怎么编写

k explain pod.metadata

## **16、三个命令玩转所有的yaml写法**

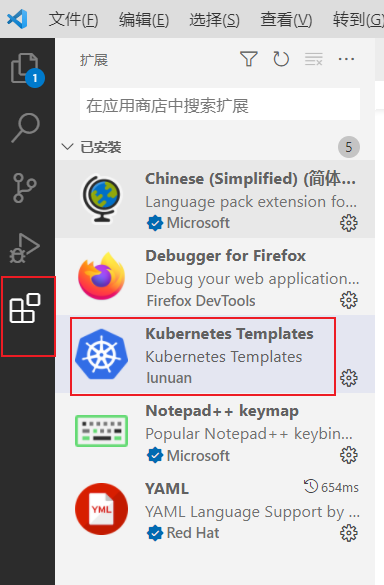
kubectl get xxx -oyaml

kubectl create deploy xxxxx --dry-run-client -oyaml

kubectl explain pod.spec.xx

写完yaml kubectl apply -f 即可

## **17、vscode编写yaml**



安装插件**Kubernetes Templates**

Getting Started

* Create a YAML file, enter prefix k8s or kube.
* Select the resource object you want to create from the pop-up list

创建个k8s或者kube开头 的yaml文件

然后输入要串讲的资源类型，如pod，service,deployment

## **18、kubectl的所有命令参考**

命令参考：**https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands**

pdf命令实战：**https://github.com/dennyzhang/cheatsheet-kubernetes-A4/blob/master/cheatsheet-kuber**

**netes-A4.pdf**

## **19、Pod里面有多个容器，进入某一个容器**

kubectl exec -it podName  -c nginx-container（容器名）-- /bin/sh

## **20、一个Deploy产生三个资源**

https://kubernetes.io/zh-cn/docs/tasks/run-application/run-stateless-application-deployment/

- Deployment资源

- replicaset资源

- Pod资源

Deployment控制RS，RS控制Pod的副本数

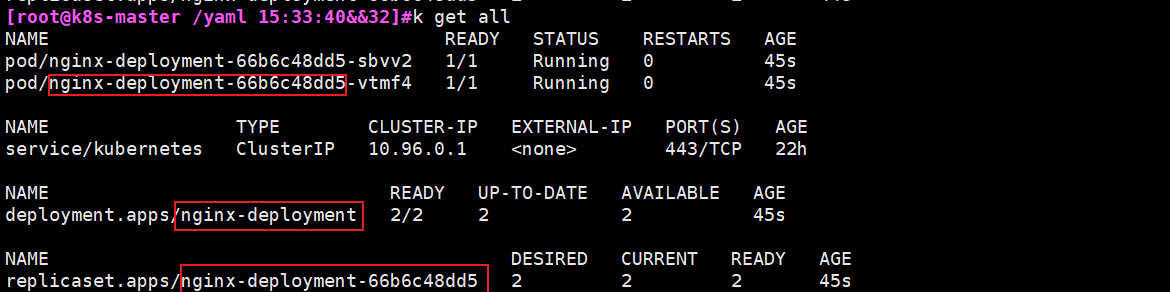
ReplicaSet： 只提供了副本数量的控制功能

Deployment： 每部署一个新版本就会创建一个新的副本集，利用他记录状态，回滚也是直接让指定的rs生效

--- rs1： 4 abc

--- rs2: 4 def

--- rsN: 4 eee



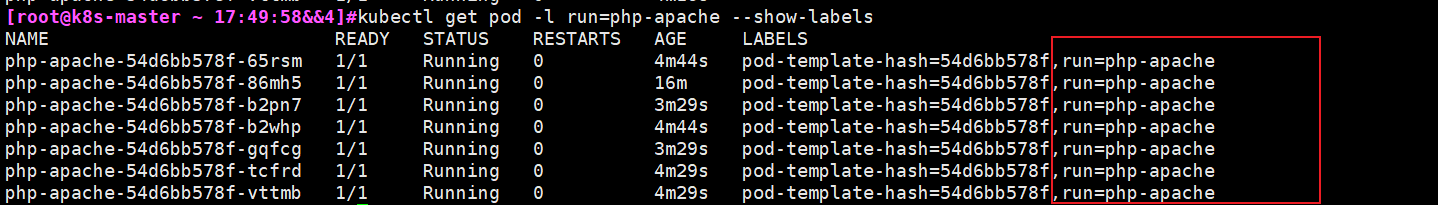
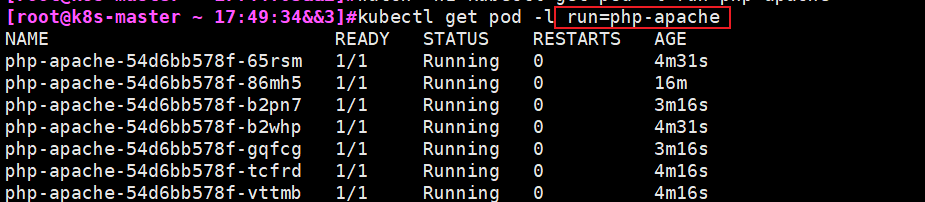
## **21、Deployment 更新机制**

- 仅当 Deployment Pod 模板（即 `.spec.template`）发生改变时，例如\*\*模板的标签或容器镜像被更新， 才会触发 Deployment 上线\*\*。 \*\*其他更新（如对 Deployment 执行扩缩容的操作）不会触发上线动作。\*\*

上线动作 原理： 创建新的rs，准备就绪后，替换旧的rs（此时不会删除，因为`revisionHistoryLimit` 指定了保留几个版本）\*\*

## **22、获取指定标签的pod**

kubectl get pod -l run=php-apache



## **23、蓝绿发布金丝雀**

**23.1 service版**

蓝绿有共同的标签 比如canary-nginx

各自又有不同的标签： 如蓝是v1 绿是v2

发布绿版本时可以控制一次发布多少个。等全部发布完成没问题后。可以根据实际需求删除v1版本。

k8s-canary-deploy-blue.yaml

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: canary-dep-v1  namespace: default  labels:  app: canary-dep-v1  spec:  selector:  matchLabels:  app: canary-nginx  v: v111  replicas: 2  template:  metadata:  labels:  app: canary-nginx  v: v111  spec:  # initContainers:  # Init containers are exactly like regular containers, except:  # - Init containers always run to completion.  # - Each init container must complete successfully before the next one starts.  containers:  - name: nginx  image: registry.cn-hangzhou.aliyuncs.com/lfy\_k8s\_images/nginx-test:env-msg |

k8s-canary-deploy-green.yaml

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: canary-dep-v2  namespace: default  labels:  app: canary-dep-v2  spec:  selector:  matchLabels:  app: canary-nginx  v: v222  replicas: 4  template:  metadata:  labels:  app: canary-nginx  v: v222  spec:  # initContainers:  # Init containers are exactly like regular containers, except:  # - Init containers always run to completion.  # - Each init container must complete successfully before the next one starts.  containers:  - name: nginx  image: nginx |

k8s-canary-service.yaml

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: canary-test  namespace: default  spec:  selector:  app: canary-nginx  type: NodePort ### 浏览器可以直接访问  ports:  - name: canary-test  port: 80 ###  targetPort: 80 ### Pod的访问端口  protocol: TCP  nodePort: 31666 ### 机器上开的端口，浏览器访问 |

**23.2 ingress版**

## **24、StatefulSet**

<https://kubernetes.io/zh-cn/docs/tutorials/stateful-application/basic-stateful-set/>

|  |
| --- |
| Deployment部署的应用我们一般称为无状态应用  StatefulSet部署的应用我们一般称为有状态应用  无状态应用：网络可能会变，存储可能会变，顺序可能会变。场景就是业务代码（  Deployment）  有状态应用：网络不变，存储不变，顺序不变。场景就是中间件（MySQL、Redis、MQ） |

## **25 卷挂载**

|  |
| --- |
| 容器声明卷挂载：容器的哪些位置做出去  Pod声明卷挂载：位置到底在哪里 |

## **26 默认存储**

|  |
| --- |
| apiVersion: storage.k8s.io/v1  kind: StorageClass  metadata:  name: nfs-storage  annotations:  storageclass.kubernetes.io/is-default-class: "true" |

## **27、亲和和反亲和**

节点只有亲和

Pod有亲和和反亲和

## **28、DaemonSet类型的资源特性**

|  |
| --- |
| DaemonSet资源对象会在每个Kubernetes集群中的节点上运行，并且每个节点只能运行一个pod，这  是它和deployment资源对象的最大也是唯一的区别。因此，在定义yaml文件中，不支持定义replicas。  它的一般使用场景如下：  在去做每个节点的日志收集工作。  监控每个节点的的运行状态。 |

## **29、Kubernetes镜像的下载策略**

K8s的镜像下载策略有三种：Always、Never、IFNotPresent。

Always：镜像标签为latest时，总是从指定的仓库中获取镜像。

Never：禁止从仓库中下载镜像，也就是说只能使用本地镜像。

IfNotPresent：仅当本地没有对应镜像时，才从目标仓库中下载。默认的镜像下载策略是：

当镜像标签是latest时，默认策略是Always；

当镜像标签是自定义时（也就是标签不是latest），那么默认策略是IfNotPresent。

## **30 kube-proxy ipvs 和 iptables 的异同**

iptables 与 IPVS 都是基于 Netfilter 实现的，但因为定位不同，二者有着本质的差别：iptables 是为防火墙而设计的；IPVS 则专门用于高性能负载均衡，并使用更高效的数据结构（Hash 表），允许几乎无限的规模扩张。

与 iptables 相比，IPVS 拥有以下明显优势：

1. 为大型集群提供了更好的可扩展性和性能；

2. 支持比 iptables 更复杂的复制均衡算法（最小负载、最少连接、加权等）；

3. 支持服务器健康检查和连接重试等功能；

4. 可以动态修改 ipset 的集合，即使 iptables 的规则正在使用这个集合

## **31 Kubernetes 中什么是静态 Pod**

静态 pod 是由 kubelet 进行管理的仅存在于特定 Node 的 Pod 上，他们不能通过 API Server 进行管理，无法与 ReplicationController、Deployment 或者 DaemonSet 进行关联，并且 kubelet 无法对他们进行健康检查。静态 Pod 总是由 kubelet 进行创建，并且总是在 kubelet 所在的 Node 上运行。

## **32 Kubernetes 数据持久化的方式有哪些？**

Kubernetes 通过数据持久化来持久化保存重要数据，常见的方式有：

EmptyDir（空目录）：没有指定要挂载宿主机上的某个目录，直接由 Pod 内保部映射到宿主机，emptyDir 的数据持久化的生命周期和使用的 pod 一致，一般是作为临时存储使用。

Hostpath：将宿主机上已存在的目录或文件挂载到容器内部。类似于 docker 中的 bind mount 挂载方式。特性：增加了 pod 与节点之间的耦合。

PersistentVolume（简称 PV）：如基于 NFS 服务的 PV，也可以基于 GFS 的 PV。它的作用是统一数据持久化目录，方便管理。