01 常见的部署方案

- 滚动更新

服务不会停止，但是整个pod会有新旧并存的情况。

- 重新创建

先停止旧的pod，然后再创建新的pod，这个过程服务是会间断的。

- 蓝绿

```
**无需停机，风险较小**

01-部署v1的应用（一开始的状态）
    所有外部请求的流量都打到这个版本上.

02-部署版本2的应用
    版本2的代码与版本1不同(新功能、Bug修复等).

03-将流量从版本1切换到版本2。

04-如版本2测试正常，就删除版本1正在使用的资源（例如实例），从此正式用版本2。
```

- 金丝雀

1.1 滚动更新

> 网盘/课堂源码/rollingupdate.yaml
>
> `maxSurge`：滚动升级时先启动的pod数量
>
> `maxUnavailable`：滚动升级时允许的最大unavailable的pod数量

```
kubectl apply -f rollingupdate.yaml
kubectl get pods
kubectl get svc
curl cluster-ip/dockerfile
```

修改rollingupdate.yaml文件，将镜像修改成v2.0

```
# 在w1上，不断地访问观察输出
while sleep 0.2;do curl cluster-ip/dockerfile;echo "";done
# 在w2上，监控pod
kubectl get pods -w
# 使得更改生效
kubectl apply -f rollingupdate.yaml
kubectl get pods
```

> `conclusion`：发现新旧pod是会共存的，并且可以访问测试看一下
>
> ```
> kubectl get pods -w
> kubectl get svc
> ```
>
> 可以发现，新老版本的确会共存

1.2 重新创建

> 网盘/课堂源码/recreate.yaml

```
kubectl apply -f recreate.yaml
kubectl get pods
```

修改recreate.yaml文件

```
kubectl apply -f recreate.yaml
kubectl get pods
```

> `conclusion`：发现pod是先停止，然后再创建新的
>
> ```
> NAME                         READY   STATUS        RESTARTS   AGE
> recreate-655d4868d8-5dqcz    0/1     Terminating   0          2m31s
> recreate-655d4868d8-sb688    0/1     Terminating   0          2m31s
> ```
>
> ```
> NAME                         READY   STATUS    RESTARTS   AGE
> recreate-6f74f4686d-4xkgl    1/1     Running   0          13s
> recreate-6f74f4686d-blrt7    1/1     Running   0          13s
> ```

Have a try

```
kubectl rollout pause deploy rollingupdate
kubectl rollout resume deploy rollingupdate
kubectl rollout undo deploy rollingupdate    # 回到上一个版本
```

1.3 蓝绿

> 网盘/课堂源码/bluegreen.yaml

```
kubectl apply -f bluegreen.yaml
kubectl get pods
```

> resources/deploy-strategy/bluegreen-service.yaml

```
kubectl apply -f bluegreen-service.yaml
kubectl get svc
# 在w1上不断访问观察
while sleep 0.3;do curl cluster-ip/dockerfile;echo "";done
```

修改bluegreen.yaml

```
01-deployment-name:blue    --->    green

02-image:v1.0--->   v2.0

03-version:v1.0    --->    v2.0
```

```
kubectl apply -f bluegreen.yaml
kubectl get pods
# 同时观察刚才访问的地址有没有变化
可以发现，两个版本就共存了，并且之前访问的地址没有变化
```

修改bluegreen-service.yaml

```
# 也就是把流量切到2.0的版本中
selector:
    app: bluegreen
    version: v2.0
```

```
kubectl apply -f bluegreen-service.yaml
kubectl get svc
# 同时观察刚才访问的地址有没有变化
发现流量已经完全切到了v2.0的版本上
```

1.4 金丝雀

> 修改上述 网盘/课堂源码/bluegreen-service.yaml
>
> ```
> selector:
> app: bluegreen
> version: v2.0    # 把version删除掉，只是根据bluegreen进行选择
> ```

```
kubectl apply -f bluegreen-service.yaml
# 同时观察刚才访问的地址有没有变化，istio中就更方便咯
此时新旧版本能够同时被访问到，AB测试，新功能部署少一些的实例
```

02 Log and Monitor

2.1 Log

2.1.1 容器级别

- docker命令查看

```
docker ps --->containerid
docker logs containerid --->查看容器的日志情况
```

- kubectl命令查看

```
kubectl logs -f <pod-name> -c <container-name>
```

2.1.2 Pod级别

```
kubectl describe pod springboot-demo-68b89b96b6-sl8bq
```

> 当然，kubectl describe除了能够查看pod的日志信息，还能查看比如Node、RC、Service、Namespace等信息。`注意`：要是想查看指定命名空间之下的，可以-n=namespace

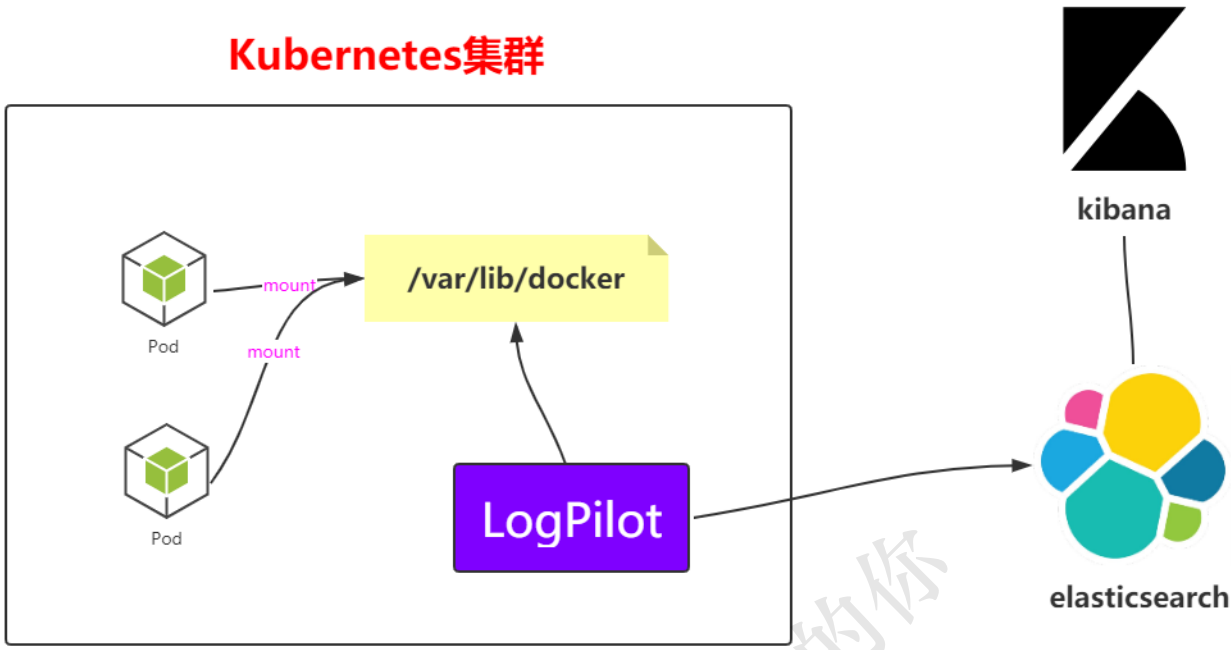
2.1.3 组件服务级别

> 比如kube-apiserver、kube-schedule、kubelet、kube-proxy、kube-controller-manager等
>
> 可以使用journalctl进行查看

```
journalctl -u kubelet
```

2.1.4 LogPilot+ES+Kibana

> https://github.com/AliyunContainerService/log-pilot



- 部署logpilot

> (1)根据网盘/课堂源码/log-pilot.yaml创建资源

```
kubectl apply -f log-pilot.yaml
```

> (2)查看pod和daemonset的信息

```
kubectl get pods -n kube-system
kubectl get pods -n kube-system -o wide | grep log
kubectl get ds -n kube-system
```

- 部署elasticsearch

> (1)根据网盘/课堂源码/elasticsearch.yaml创建资源

```
kubectl apply -f elasticsearch.yaml
kubectl get pods -n kube-system
kubectl get pods -n kube-system -o wide | grep ela
```

> (2)查看kube-system下的svc
>
> kubectl get svc -n kube-system

```
elasticsearch-api         ClusterIP   10.106.65.2      <none>        9200/TCP
elasticsearch-discovery   ClusterIP   10.101.117.180   <none>        9300/TCP
kube-dns                  ClusterIP   10.96.0.10       <none>
```

> (3)查看kube-system下的statefulset
>
> kubectl get statefulset -n kube-system

```
NAME            READY   AGE
elasticsearch   3/3     106s
```

- 部署kibana

> (1)根据网盘/课堂源码/kibana.yaml创建资源
>
> kibana主要是对外提供访问的，所以这边需要配置Service和Ingress
>
> **前提**：要有Ingress Controller的支持，比如Nginx Controller

```
kubectl apply -f kibana.yaml
```

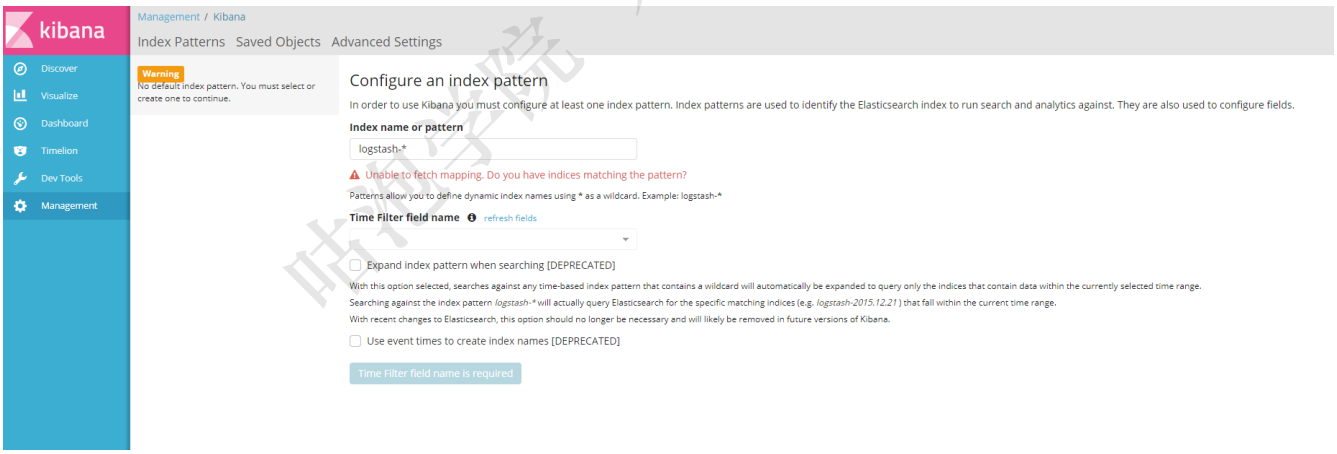
> (2)查看pod和deployment的信息

```
kubectl get pods -n kube-system | grep ki
kubectl get deploy -n kube-system
```

> (3)配置Ingress需要的域名
>
> 打开windows上的hosts文件

```
# 注意这边是worker01的IP
121.41.10.126 kibana.jack.com
```

> (4)在windows访问kibana.jack.com

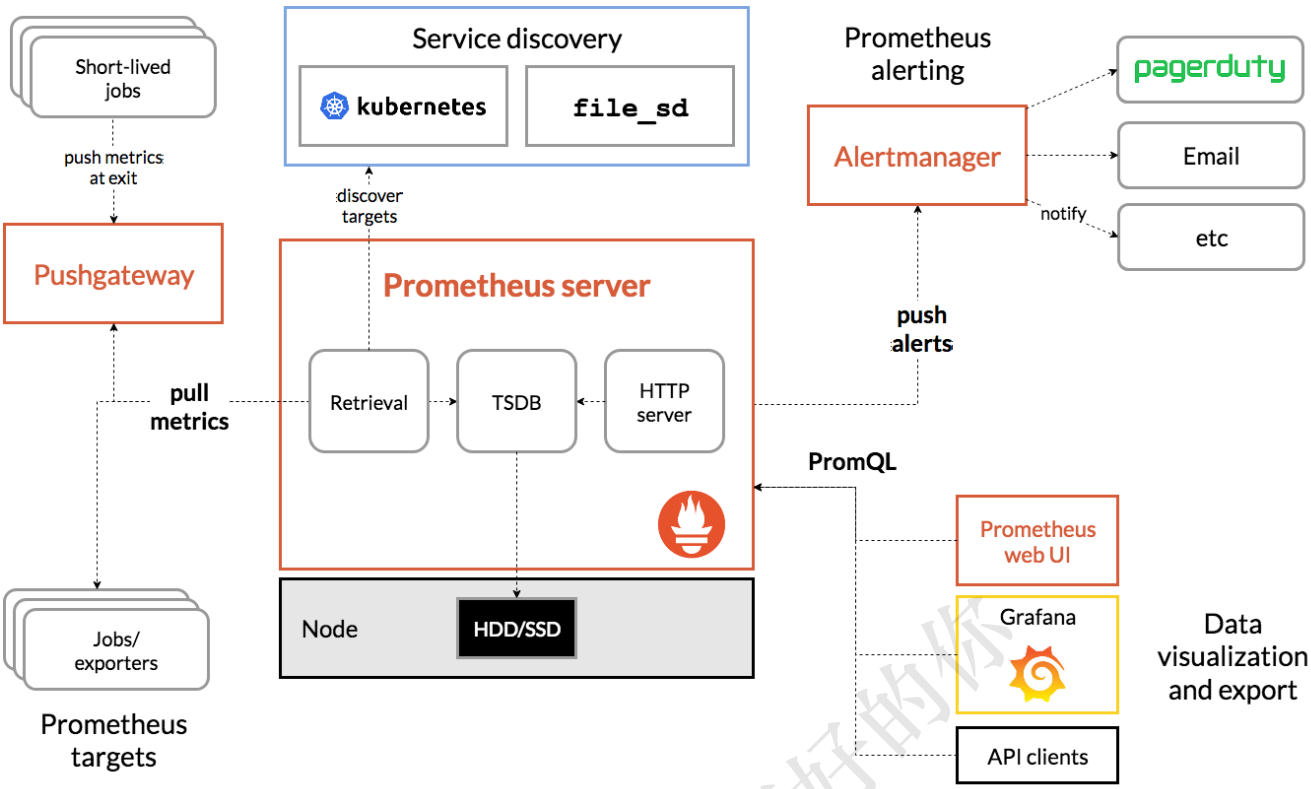


2.2 Monitor

2.2.1 Prometheus简介

> 官网：https://prometheus.io/
>
> github：https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/prometheus

2.2.2 Prometheus架构



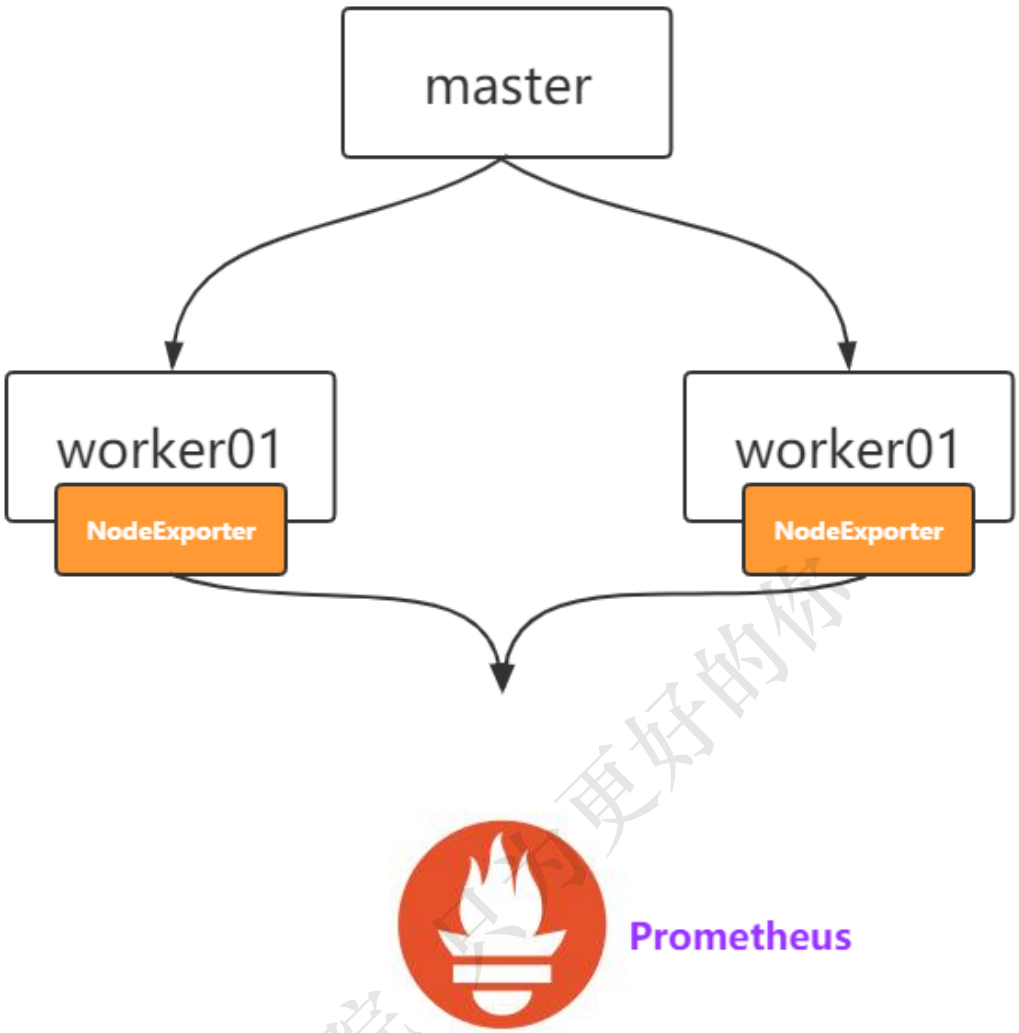
2.2.3 Prometheus知识普及

- 支持pull、push数据添加方式
- 支持k8s服务发现
- 提供查询语言PromQL
- 时序(time series)是由名字(Metric)以及一组key/value标签定义的
- 数据类型

2.2.4 数据采集

2.2.4.1 服务器数据

> 通过NodeExporter：https://github.com/prometheus/node_exporter



2.2.4.2 组件数据

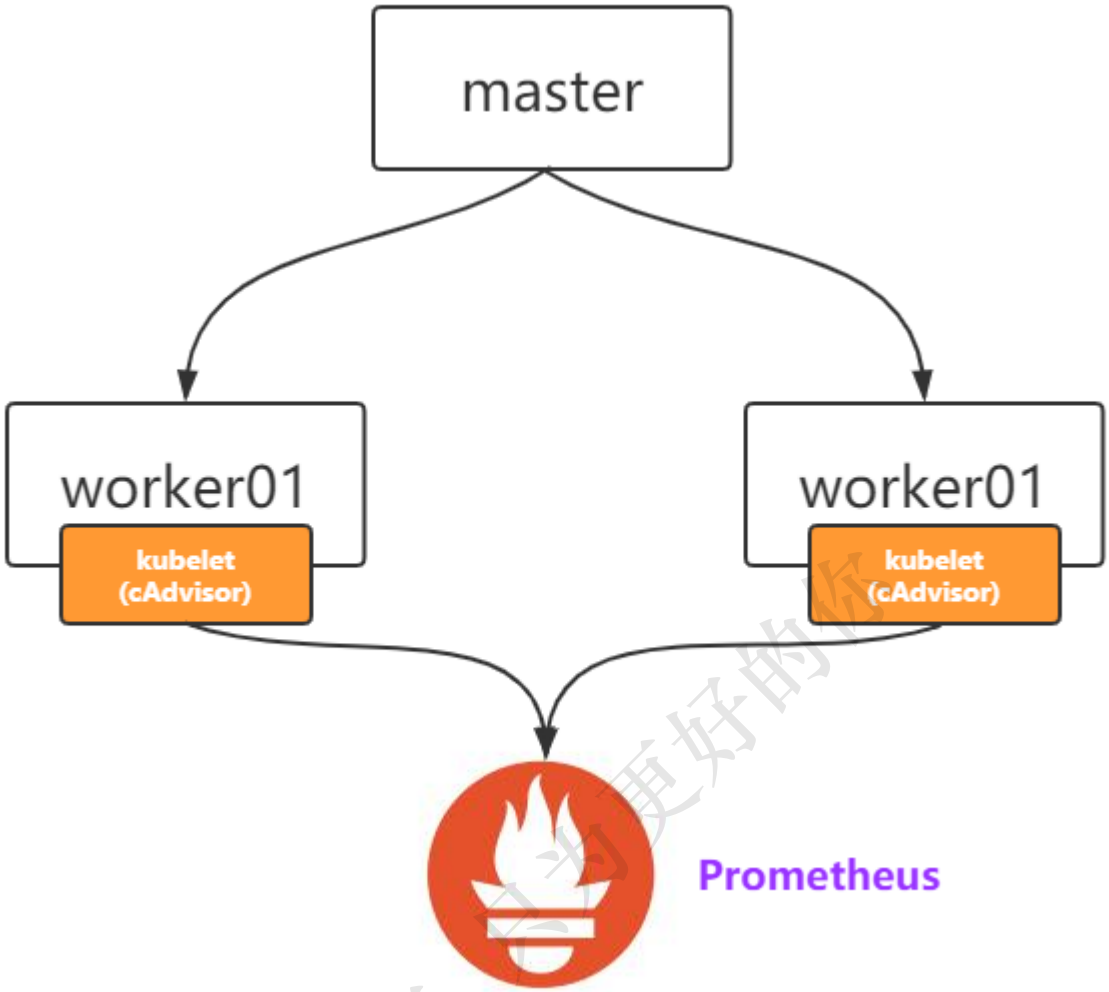
ETCD:https://ip:2379/metrics

APIServer:https://ip:6443/metrics

ControllerManager:https://ip:10252/metrics

Scheduler:https://ip:10251/metrics

2.2.4.3 容器数据



2.2.5 Prometheus+Grafana

> 在master上创建prometheus目录
>
> 网盘/课堂源码/*.yaml
>
> ```
> namespace.yaml
> node-exporter.yaml
> prometheus.yaml
> grafana.yaml
> ingress.yaml
> ```

> (1)创建命名空间ns-monitor

```
kubectl apply -f namespace.yaml
kubectl get namespace
```

> (2)创建node-exporter

```
kubectl apply -f node-exporter.yaml
kubectl get pod -n ns-monitor
kubectl get svc -n ns-monitor
kubectl get ds -n ns-monitor
win浏览器访问集群任意一个ip，比如http://121.41.10.126:31672  查看结果    # 这边是http协议，不能用https
```

> (3)部署prometheus pod
>
> 包含rbac认证、ConfigMap等
>
> 注意： 记得修改prometheus.yaml文件中的ip为master的ip和path[PV需要使用到]

```
kubectl apply -f prometheus.yaml
kubectl get pod -n ns-monitor
kubectl get svc -n ns-monitor
win浏览器访问集群任意一个ip:30222/graph  查看结果，比如http://121.41.10.126:30137
```

> (4)部署grafana

```
kubectl apply -f grafana.yaml
kubectl get pod -n ns-monitor
kubectl get svc -n ns-monitor
win浏览器访问集群任意一个ip:32405/graph/login
比如http://121.41.10.126:32727用户名密码:admin
```

> (5)增加域名访问[没有域名好像没有灵魂]
>
> 前提： 配置好ingress controller和域名解析

```
kubectl apply - ingress.yaml
kubectl get ingress -n ns-monitor
kubectl describe ingress -n ns-monitor
```

> (6)直接通过域名访问即可

03 Trouble Shooting

3.1 Master

> master上的组件共同组成了控制平面

```
01 若apiserver出问题了
    会导致整个K8s集群不可以使用，因为apiserver是K8s集群的大脑

02 若etcd出问题了
    apiserver和etcd则无法通信，kubelet也无法更新所在node上的状态

03 当scheduler或者controller manager出现问题时
    会导致deploy，pod，service等无法正常运行
```

> 解决方案：出现问题时，监听到自动重启或者搭建高可用的master集群

3.2 Worker

worker节点挂掉或者上面的kubelet服务出现问题时，w上的pod则无法正常运行。

3.3 Addons

dns和网络插件比如calico发生问题时，集群内的网络无法正常通信，并且无法根据服务名称进行解析。

3.4 系统问题排查

- 查看Node的状态

```
kubectl get nodes
kubectl describe node-name
```

- 查看集群master和worker组件的日志

```
journalctl -u apiserver
journalctl -u scheduler
journalctl -u kubelet
journalctl -u kube-proxy
...
```


3.5 Pod的问题排查

> K8s中最小的操作单元是Pod，最重要的操作也是Pod，其他资源的排查可以参照Pod问题的排查

> (1)查看Pod运行情况

```
kubectl get pods -n namespace
```

> (2)查看Pod的具体描述，定位问题

```
kubectl describe pod pod-name -n namespace
```

> (3)检查Pod对应的yaml是否有误

```
kubectl get pod pod-name -o yaml
```

> (4)查看Pod日志

```
kubectl logs ...
```

Pod可能会出现哪些问题及解决方案

```
01 处于Pending状态
    说明Pod还没有被调度到某个node上，可以describe一下详情。可能因为资源不足，端口被占用等。

02 处于Waiting/ContainerCreating状态
    可能因为镜像拉取失败，或者是网络插件的问题，比如calico，或者是容器本身的问题，可以检查一下容器的yaml文件内容和Dockerfile的书写。

03 处于ImagePullBackOff状态
    镜像拉取失败，可能是镜像不存在，或者没有权限拉取。

04 处于CrashLoopBackOff状态
    Pod之前启动成功过，但是又失败了，不断在重启。

05 处于Error状态
    有些内容不存在，比如ConfigMap，PV，没有权限等，需要创建一下。

06 处于Terminating状态
    说明Pod正在停止

07 处于Unknown状态
    说明K8s已经失去对Pod的管理监听。
```