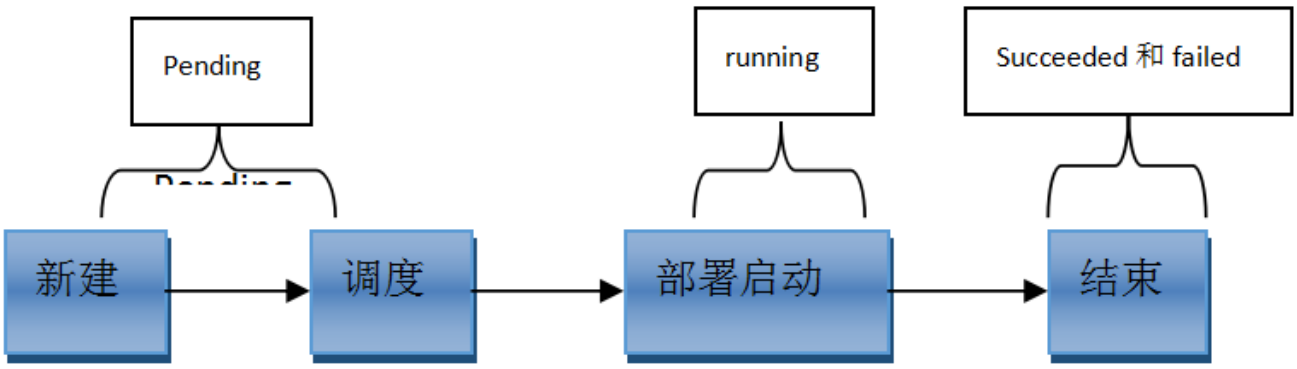
Pod生命周期和探针

一：Pod的状态和探针：

https://kubernetes.io/zh/docs/concepts/workloads/pods/pod-lifecycle/

1.1：Pod状态：



第一阶段：

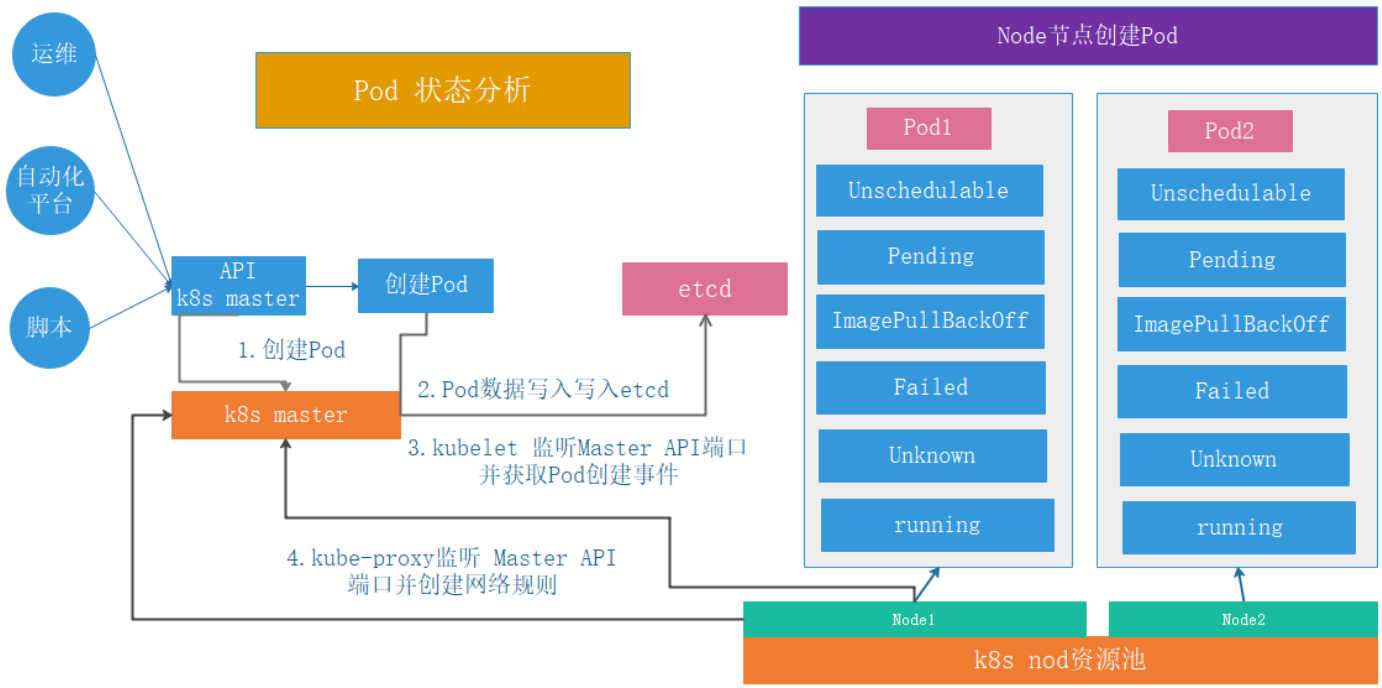
```
    Pending:
    #正在创建Pod但是Pod中的容器还没有全部被创建完成=[处于此状态的Pod应该检查Pod依赖的存储是否有权限挂载、镜像是否可以下载、调度是否正常等。

    Failed
    #Pod中有容器启动失败而导致pod工作异常。

    Unknown
    #由于某种原因无法获得pod的当前状态，通常是由于与pod所在的node节点通信错误。

    Succeeded
    #Pod中的所有容器都被成功终止即pod里所有的containers均已terminated。
```

第二阶段：

```
    Unschedulable:
    #Pod不能被调度，kube-scheduler没有匹配到合适的node节点

    PodScheduled
    #pod正处于调度中，在kube-scheduler刚开始调度的时候，还没有将pod分配到指定的node，在筛选出合适的节点后就会更新etcd数据，将pod分配到指定的node

    Initialized
    #所有pod中的初始化容器已经完成了

    ImagePullBackOff:
    #Pod所在的node节点下载镜像失败

    Running
```

#Pod内部的容器已经被创建并且启动。

Ready
#表示pod中的容器已经可以提供访问服务



Error：#pod 启动过程中发生错误
NodeLost：#Pod 所在节点失联
Unkown：#Pod 所在节点失联或其它未知异常
Waiting：#Pod 等待启动
Pending：#Pod 等待被调度
Terminating：#Pod 正在被销毁
CrashLoopBackOff：#pod，但是kubelet正在将它重启
InvalidImageName：#node节点无法解析镜像名称导致的镜像无法下载
ImageInspectError：#无法校验镜像，镜像不完整导致
ErrImageNeverPull：#策略禁止拉取镜像，镜像中心权限是私有等
ImagePullBackOff：#镜像拉取失败，但是正在重新拉取
RegistryUnavailable：#镜像服务器不可用，网络原因或harbor宕机
ErrImagePull：#镜像拉取出错，超时或下载被强制终止
CreateContainerConfigError：#不能创建kubelet使用的容器配置
CreateContainerError：#创建容器失败
PreStartContainer： #执行preStart hook报错，Pod hook(钩子)是由 Kubernetes 管理的 kubelet 发起的，当容器中的进程启动前或者容器中的进程终止之前运行，比如容器创建完成后里面的服务启动之前可以检查一下依赖的其它服务是否启动，或者容器退出之前可以把容器中的服务先通过命令停止。
PostStartHookError：#执行 postStart hook 报错

RunContainerError：#pod运行失败，容器中没有初始化PID为1的守护进程等
ContainersNotInitialized：#pod没有初始化完毕
ContainersNotReady：#pod没有准备完毕
ContainerCreating：#pod正在创建中
PodInitializing：#pod正在初始化中
DockerDaemonNotReady：#node节点decker服务没有启动

```
NetworkPluginNotReady：#网络插件还没有完全启动
```

1.2：Pod调度过程：

件k8s 实战案例 2.1.3：kube-scheduler：

1.3：Pod探针：

https://kubernetes.io/zh/docs/concepts/workloads/pods/pod-lifecycle/#%E5%AE%B9%E5%99%A8%E6%8E%A2%E9%92%88

1.3.1：探针简介：

探针 是由 kubelet 对容器执行的定期诊断，以保证Pod的状态始终处于运行状态，要执行诊断，kubelet 调用由容器实现的Handler(处理程序)，有三种类型的处理程序：

```
ExecAction
#在容器内执行指定命令，如果命令退出时返回码为0则认为诊断成功。

TCPSocketAction
#对指定端口上的容器的IP地址进行TCP检查，如果端口打开，则诊断被认为是成功的。

HTTPGetAction
#对指定的端口和路径上的容器的IP地址执行HTTPGet请求，如果响应的状态码大于等于200且小于 400，则诊断被认为是成功的。
```

每次探测都将获得以下三种结果之一：

```
成功：容器通过了诊断。
失败：容器未通过诊断。
未知：诊断失败，因此不会采取任何行动。
```

1.3.2：配置探针：

基于探针实现对Pod的状态检测

1.3.2.1：探针类型：

```
livenessProbe
#存活探针，检测容器容器是否正在运行，如果存活探测失败，则kubelet会杀死容器，并且容器将受到其重启策略的影响，如果容器不提供存活探针，则默认状态为 Success，livenessProbe用于控制是否重启pod。

readinessProbe
#就绪探针，如果就绪探测失败，端点控制器将从与Pod匹配的所有Service的端点中删除该Pod的IP地址，初始延迟之前的就绪状态默认为Failure(失败)，如果容器不提供就绪探针，则默认状态为 Success，readinessProbe用于控制pod是否添加至service。
```

1.3.2.2：探针配置：

https://kubernetes.io/zh/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

探针有很多配置字段，可以使用这些字段精确的控制存活和就绪检测的行为：

```
initialDelaySeconds: 120
#初始化延迟时间，告诉kubelet在执行第一次探测前应该等待多少秒，默认是0秒，最小值是0

periodSeconds: 60
#探测周期间隔时间，指定了kubelet应该每多少秒秒执行一次存活探测，默认是 10 秒。最小值是 1

timeoutSeconds: 5
#单次探测超时时间，探测的超时后等待多少秒，默认值是1秒，最小值是1。

successThreshold: 1
#从失败转为成功的重试次数，探测器在失败后，被视为成功的最小连续成功数，默认值是1，存活探测的这个值必须是1，最小值是 1。

failureThreshold: 3
#从成功转为失败的重试次数，当Pod启动了并且探测到失败，Kubernetes的重试次数，存活探测情况下的放弃就意味着重新启动容器，就绪探测情况下的放弃Pod 会被打上未就绪的标签，默认值是3，最小值是1。
```

HTTP 探测器可以在 httpGet 上配置额外的字段：

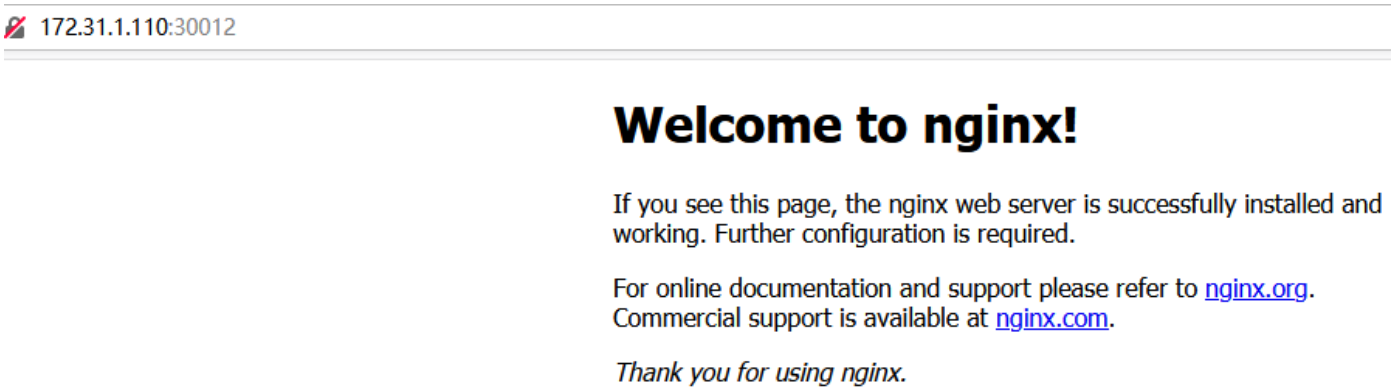
```
host:
#连接使用的主机名，默认是Pod的 IP，也可以在HTTP头中设置 “Host” 来代替。

scheme: http
#用于设置连接主机的方式（HTTP 还是 HTTPS），默认是 HTTP。

path: /monitor/index.html
#访问 HTTP 服务的路径。

httpHeaders:
#请求中自定义的 HTTP 头,HTTP 头字段允许重复。

port: 80
#访问容器的端口号或者端口名，如果数字必须在 1 ～ 65535 之间。
```

1.3.2.3：HTTP探针示例：

```
# cat nginx.yaml
#apiVersion: extensions/v1beta1
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels: #rs or deployment
      app: ng-deploy-80
    #matchExpressions:
    #  - {key: app, operator: In, values: [ng-deploy-80,ng-rs-81]}
  template:
    metadata:
      labels:
        app: ng-deploy-80
    spec:
      containers:
      - name: ng-deploy-80
        image: nginx:1.17.5
        ports:
        - containerPort: 80
        #readinessProbe:
        livenessProbe:
          httpGet:
            #path: /monitor/monitor.html
            path: /index.html
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 3
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 3


---
apiVersion: v1
kind: Service
metadata:
  name: ng-deploy-80
spec:
  ports:
  - name: http
    port: 81
    targetPort: 80
    nodePort: 40012
```

```
    protocol: TCP
  type: NodePort
  selector:
    app: ng-deploy-80
```

验证http探针：



1.3.2.4：TCP探针示例：

```
# cat nginx.yaml
#apiVersion: extensions/v1beta1
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels: #rs or deployment
      app: ng-deploy-80
    #matchExpressions:
    #  - {key: app, operator: In, values: [ng-deploy-80,ng-rs-81]}
  template:
    metadata:
      labels:
        app: ng-deploy-80
    spec:
      containers:
      - name: ng-deploy-80
        image: nginx:1.17.5
        ports:
        - containerPort: 80
        livenessProbe:
        #readinessProbe:
          tcpSocket:
            port: 80
            #port: 8080
          initialDelaySeconds: 5
```

```
          periodSeconds: 3
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 3

---
apiVersion: v1
kind: Service
metadata:
  name: ng-deploy-80
spec:
  ports:
  - name: http
    port: 81
    targetPort: 80
    nodePort: 40012
    protocol: TCP
  type: NodePort
  selector:
    app: ng-deploy-80
```

1.3.2.5：ExecAction探针：

可以基于指定的命令对Pod进行特定的状态检查。

```
# docker pull redis

# cat redis.yaml
#apiVersion: extensions/v1beta1
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-deployment
spec:
  replicas: 1
  selector:
    matchLabels: #rs or deployment
      app: redis-deploy-6379
    #matchExpressions:
    #  - {key: app, operator: In, values: [redis-deploy-6379,ng-rs-81]}
  template:
    metadata:
      labels:
        app: redis-deploy-6379
    spec:
      containers:
      - name: redis-deploy-6379
        image: redis
        ports:
```

```
        - containerPort: 6379
        livenessProbe:
        #readinessProbe:
          exec:
            command:
            #- /apps/redis/bin/redis-cli
            - /usr/local/bin/redis-cli
            - quit
          initialDelaySeconds: 5
          periodSeconds: 3
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 3

---
apiVersion: v1
kind: Service
metadata:
  name: redis-deploy-6379
spec:
  ports:
  - name: http
    port: 6379
    targetPort: 6379
    nodePort: 40016
    protocol: TCP
  type: NodePort
  selector:
    app: redis-deploy-6379
```

如果端口检测连续超过指定的三次都没有通过，则Pod状态如下：

```
root@k8s-master1:/opt/k8s-data/yaml/magedu/case2# kubectl  get pod
NAME                                READY   STATUS             RESTARTS   AGE
busybox                             1/1     Running            13         3d
mysql-0                             2/2     Running            2          2d22h
mysql-1                             2/2     Running            3          2d22h
mysql-2                             2/2     Running            3          2d22h
net-test1-5fcc69db59-4svkr          1/1     Running            1          2d16h
net-test1-5fcc69db59-gvfmn          1/1     Running            3          5d12h
net-test1-5fcc69db59-kxnmf          1/1     Running            1          2d16h
net-test1-5fcc69db59-qtskp          1/1     Running            3          5d12h
nginx-deployment-84bd8648f5-bchwj   1/1     Running            0          2m41s
redis-deployment-849d977d4c-dkndn   0/1     CrashLoopBackOff   3          67s
root@k8s-master1:/opt/k8s-data/yaml/magedu/case2#
```

1.3.2.6：livenessProbe和readinessProbe的对比：

```
配置参数一样
livenessProbe #连续探测失败会重启、重建pod，readinessProbe不会执行重启或者重建Pod操作
livenessProbe #连续检测指定次数失败后会将容器置于(Crash Loop BackOff)且不可用，readinessProbe不会
readinessProbe #连续探测失败会从service的endpointd中删除该Pod，livenessProbe不具备此功能，但是会将容器挂起livenessProbe

livenessProbe用户控制是否重启pod，readinessProbe用于控制pod是否添加至service

建议：
  两个探针都配置
```

1.4：Pod重启策略：

k8s在Pod出现异常的时候会自动将Pod重启以恢复Pod中的服务。

```
restartPolicy：
    Always：当容器异常时，k8s自动重启该容器，ReplicationController/Replicaset/Deployment。
    OnFailure：当容器失败时(容器停止运行且退出码不为0)，k8s自动重启该容器。
    Never：不论容器运行状态如何都不会重启该容器,Job或CronJob。
```

示例：

```
      containers:
      - name: magedu-tomcat-app1-container
        image: harbor.magedu.local/magedu/tomcat-app1:v1
        #command: ["/apps/tomcat/bin/run_tomcat.sh"]
        #imagePullPolicy: IfNotPresent
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
          protocol: TCP
          name: http
        env:
        - name: "password"
          value: "123456"
        - name: "age"
          value: "18"
        resources:
          limits:
            cpu: 1
            memory: "512Mi"
          requests:
            cpu: 500m
```

```
            memory: "512Mi"
        restartPolicy: Always
```

```
root@k8s-master1:/opt/k8s-data/yaml/magedu/nginx# kubectl  get pod
NAME                                READY   STATUS    RESTARTS   AGE
busybox                             1/1     Running   12         2d22h
mysql-0                             2/2     Running   2          2d20h
mysql-1                             2/2     Running   3          2d20h
mysql-2                             2/2     Running   3          2d20h
net-test1-5fcc69db59-4svkr          1/1     Running   1          2d14h
net-test1-5fcc69db59-gvfmn          1/1     Running   3          5d10h
net-test1-5fcc69db59-kxnmf          1/1     Running   1          2d14h
net-test1-5fcc69db59-qtskp          1/1     Running   3          5d10h
nginx-deployment-8c449b55f-c8fnm    1/1     Running   1          2d11h
root@k8s-master1:/opt/k8s-data/yaml/magedu/nginx#
```

1.5：镜像拉取策略：

https://kubernetes.io/zh/docs/concepts/configuration/overview/

```
imagePullPolicy: IfNotPresent #node节点没有此镜像就去指定的镜像仓库拉取，node有就使用node本地镜像。
imagePullPolicy: Always #每次重建pod都会重新拉取镜像
imagePullPolicy: Never #从不到镜像中心拉取镜像，只使用本地镜像
```