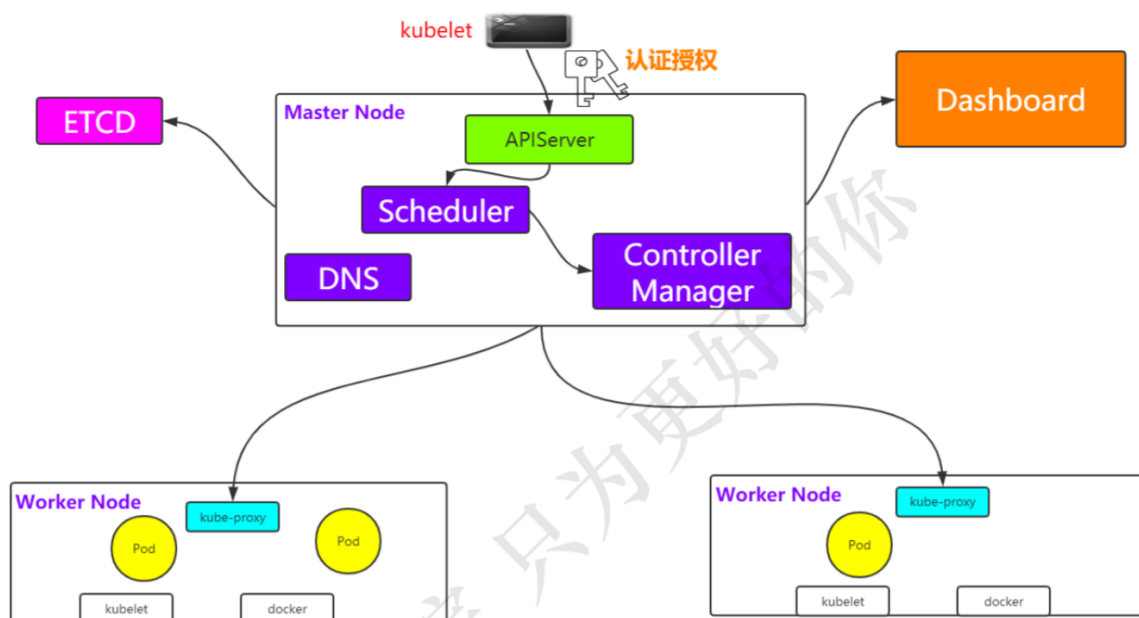


Kubernetes学习篇章

作者: shouzhi@duan

一、k8s核心组件



1、Kubectl

用于在master节点操控集群节点工具，比如说创建pod、svc、configmap等相关资源。

2、ApiServer

请求到达master之后，然后在分配给worker工作节点创建Pod之类的关键命令是通过kubectl过来之后，是不是需要授权一下，那么ApiServer就是用于授权的组件。

3、Scheduler

授权通过后，接下来具体操作那个worker节点，或者container之类的，得要有调度策略。那么scheduler就是起到一个调度策略的作用。

4、Controller Manager

调度器执行调度之后会有一个分发器，用来真正路由分发到哪个worker工作节点上。

5、Kubelet

分发器分发到具体的worker后，最终会由kubelet服务调用Docker Engine，创建对应的容器。

6、DNS域名解析

Calico、CoreDNS插件

7、ETCD分布式存储

etcd集群部署。

8、面板监控

Dashboard。

9、网络持、持久化

可以参考一下Docker方式，后面具体再做展开。

二、技术栈

1、k8s高可用部署方案

- [kubeadmin方式](#) (官网)
主人已实现kube-admin部署方式，[文档参考](#)。
- [kubespray方式](#) (官网)
主人已实现kube-spray部署方式，部署文档待编写。
- [kops方式](#) (官网)
主人未实现。
- [hard-way方式](#) (社区)
主人已实现hard-way部署方式，[文档参考](#)。

2、[k8s中文网](#)

3、[在线服务器](#)

注意：这个有效时间4个小时，仅仅用于测试学习使用。

三、k8s初体验

1、定义一个pod_nginx_rs.yaml文件，熟悉docker-compose的话这里就不用解释了。

```
cat > pod_nginx_rs.yaml <<EOF
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx
  labels:
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      name: nginx
      labels:
        tier: frontend
    spec:
```

```
containers:
  - name: nginx
    image: nginx
    ports:
      - containerPort: 80
EOF
```

2、启动yaml

```
kubectl apply -f pod_nginx_rs.yaml
```

3、查看pod

```
kubectl get pods
kubectl get pods -o wide
kubectl describe pod nginx
```

4、扩容

```
kubectl scale rs nginx --replicas=5
kubectl get pods -o wide
```

5、删除pod

```
kubectl delete -f pod_nginx_rs.yaml
```

6、查看pod详情

```
kubectl describe pod nginx-abc
```

```
Name:          nginx-abc-mhtjj
Namespace:     default
Priority:       0
Node:          node-3/192.168.10.123
Start Time:    Tue, 21 Dec 2021 16:59:20 +0800
Labels:        tier=frontend
Annotations:    cnf.projectcalico.org/containerID: 8802143744ad0adbaa1bf2b0e8a63e4a4ef2379d45da808dbf962312c03d08b2
                cnf.projectcalico.org/podIP: 10.200.139.106/32
                cnf.projectcalico.org/podIPs: 10.200.139.106/32
Status:        Running
IP:            10.200.139.106
IPs:
  IP:          10.200.139.106
Controlled By: ReplicaSet/nginx-abc
Containers:
  nginx-cname:
    Container ID:  containerd://fca76fd7a4d54e70e45eeb18b098a6f65611afc24e9de4e97c54610c3ad3f3b7
    Image:         nginx
    Image ID:      docker.io/library/nginx@sha256:af472ddb9a3f053b8559361f89cfb7c2eb49775acff64b0999c08446f7e79b82
    Port:         80/TCP
```

```

Host Port:      0/TCP
State:         Running
  Started:      Tue, 21 Dec 2021 16:59:23 +0800
Ready:         True
Restart Count: 0
Environment:    <none>
Mounts:
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-44qj2
(ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-44qj2:
    Type:          Secret (a volume populated by a Secret)
    SecretName:     default-token-44qj2
    Optional:       false
QoS Class:        BestEffort
Node-Selectors:    <none>
Tolerations:       node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                   node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:            <none>

```

四、YAML文件

[参考文档](#)

```

#test-pod
apiVersion: v1 #指定api版本, 此值必须在kubectl apiversion中
kind: Pod #指定创建资源的角色/类型
metadata: #资源的元数据/属性
  name: test-pod #资源的名字, 在同一个namespace中必须唯一
  labels: #设定资源的标签
    k8s-app: apache
    version: v1
    kubernetes.io/cluster-service: "true"
  annotations: #自定义注解列表
    - name: String #自定义注解名字
spec: #specification of the resource content 指定该资源的内容
  restartPolicy: Always #表明该容器一直运行, 默认k8s的策略, 在此容器退出后, 会立即创建一个
  相同的容器
  nodeSelector: #节点选择, 先给主机打标签kubectl label nodes kube-node1
  zone=node1
  zone: node1
  containers:
    - name: test-pod #容器的名字
      image: 10.192.21.18:5000/test/chat:latest #容器使用的镜像地址
      imagePullPolicy: Never #三个选择Always、Never、IfNotPresent, 每次启动时检查和更新
      (从registry) images的策略,
        # Always, 每次都检查
        # Never, 每次都不检查(不管本地是否有)
        # IfNotPresent, 如果本地有就不检查, 如果没有就拉取

```

```

command: ['sh'] #启动容器的运行命令, 将覆盖容器中的Entrypoint, 对应Dockerfile中的
ENTRYPOINT
args: ["$(str)"] #启动容器的命令参数, 对应Dockerfile中CMD参数
env: #指定容器中的环境变量
- name: str #变量的名字
  value: "/etc/run.sh" #变量的值
resources: #资源管理
  requests: #容器运行时, 最低资源需求, 也就是说最少需要多少资源容器才能正常运行
    cpu: 0.1 #CPU资源(核数), 两种方式, 浮点数或者是整数+m, 0.1=100m, 最少值为0.001
核(1m)
    memory: 32Mi #内存使用量
  limits: #资源限制
    cpu: 0.5
    memory: 1000Mi
ports:
- containerPort: 80 #容器开发对外的端口
  name: httpd #名称
  protocol: TCP
livenessProbe: #pod内容器健康检查的设置
  httpGet: #通过httpget检查健康, 返回200-399之间, 则认为容器正常
    path: / #URI地址
    port: 80
    #host: 127.0.0.1 #主机地址
    scheme: HTTP
  initialDelaySeconds: 180 #表明第一次检测在容器启动后多长时间后开始
  timeoutSeconds: 5 #检测的超时时间
  periodSeconds: 15 #检查间隔时间
#也可以用这种方法
#exec: 执行命令的方法进行监测, 如果其退出码不为0, 则认为容器正常
#  command:
#    - cat
#    - /tmp/health
#也可以用这种方法
#tcpSocket: //通过tcpSocket检查健康
#  port: number
lifecycle: #生命周期管理
  postStart: #容器运行之前运行的任务
    exec:
      command:
        - 'sh'
        - 'yum upgrade -y'
  preStop: #容器关闭之前运行的任务
    exec:
      command: ['service httpd stop']
volumeMounts: #挂载持久存储卷
- name: volume #挂载设备的名字, 与volumes[*].name 需要对应
  mountPath: /data #挂载到容器的某个路径下
  readOnly: True
volumes: #定义一组挂载设备
- name: volume #定义一个挂载设备的名字
  #emptyDir: {}
  hostPath:
    path: /opt #挂载设备类型为hostPath, 路径为宿主机下的/opt, 这里设备类型支持很多种
    #nfs

```

五、常见部署资源

1、ReplicationController(RC)

ReplicationController定义了一个期望的场景，即声明某种Pod的副本数量在任意时刻都符合某个预期值，所以RC的定义包含以下几个部分：

- Pod期待的副本数 (replicas)
- 用于筛选目标Pod的Label Selector
- 当Pod的副本数量小于预期数量时，用于创建新Pod的Pod模板 (template)

也就是说通过RC实现了集群中Pod的高可用，减少了传统IT环境中手工运维的工作。

Have a try

kind: 表示要新建对象的类型

spec.selector: 表示需要管理的Pod的label，这里表示包含app: nginx的label的Pod都会被该RC管理

spec.replicas: 表示受此RC管理的Pod需要运行的副本数

spec.template: 表示用于定义Pod的模板，比如Pod名称、拥有的label以及Pod中运行的应用等
通过改变RC里Pod模板中的镜像版本，可以实现Pod的升级功能

kubectl apply -f nginx-pod.yaml, 此时k8s会在所有可用的Node上，创建3个Pod，并且每个Pod都有一个app: nginx的label，同时每个Pod中都运行了一个nginx容器。

如果某个Pod发生问题，Controller Manager能够及时发现，然后根据RC的定义，创建一个新的Pod

扩缩容: kubectl scale rc nginx --replicas=5

(1)创建名为nginx_replication.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

(2)根据nginx_replication.yaml创建pod

```
kubectl apply -f nginx_replication.yaml
```

(3)查看pod

```
kubectl get po -A -o wide
```

```
root@node-1:~/kubernetes/deploy_work/test# kubectl get po -A -o wide
NAMESPACE   NAME             READY   STATUS    RESTARTS   AGE   IP            NODE     NOMINATED NODE   READINESS GATES
default     nginx-abc-mhtjj  1/1     Running   0           131m  10.200.139.106 node-3    <none>            <none>
default     nginx-abc-wj8ls  1/1     Running   0           131m  10.200.139.107 node-3    <none>            <none>
default     nginx-rc-7gxjx   1/1     Running   0           16m   10.200.139.110 node-3    <none>            <none>
default     nginx-rc-7nktx   1/1     Running   0           16m   10.200.247.46  node-2    <none>            <none>
```

(4)尝试删除一个pod

当我删除一个pod的时候，会发现k8s会帮我们重启一个来保证副本数的一致。

```
kubectl delete pods nginx-zzwz1
```

(5)动态扩容

```
# nginx-rc: 表示需要扩容资源的名称
kubectl scale rc nginx-rc --replicas=3
```

(6)删除pod

```
kubectl delete -f nginx_replication.yaml
```

2、ReplicaSet(RS)

官网：<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>

A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

在Kubernetes v1.2时，RC就升级成了另外一个概念：Replica Set，官方解释为“下一代RC”

ReplicaSet和RC没有本质的区别，kubectl中绝大部分作用于RC的命令同样适用于RS

RS与RC唯一的区别是：RS支持基于集合的Label Selector（Set-based selector），而RC只支持基于等式的Label Selector（equality-based selector），这使得Replica Set的功能更强

Have a try

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  matchLabels:
    tier: frontend
  matchExpressions:
    - {key: tier, operator: In, values: [frontend]}
  template:
    ...
```

注意：一般情况下，我们很少单独使用Replica Set，它主要是被Deployment这个更高的资源对象所使用，从而形成一整套Pod创建、删除、更新的编排机制。当我们使用Deployment时，无须关心它是如何创建和维护Replica Set的，这一切都是自动发生的。同时，无需担心跟其他机制的不兼容问题（比如ReplicaSet不支持rolling-update但Deployment支持）。

3、Deployment

官网: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

A Deployment provides declarative updates for Pods and ReplicaSets.

You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

Deployment相对RC最大的一个升级就是我们可以随时知道当前Pod“部署”的进度。

创建一个Deployment对象来生成对应的Replica Set并完成Pod副本的创建过程

检查Deployment的状态来看部署动作是否完成（Pod副本的数量是否达到预期的值）

(1)、nginx_deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

(2)创建pod

```
kubectl apply -f nginx_deployment.yaml
```

(3)查看pod

```
kubectl get pods -o wide
```



```
root@node-1:~/kubernetes/deploy_work/test# kubectl get pod -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	nginx-abc-mhtjj	1/1	Running	0	161m
default	nginx-abc-wj8ls	1/1	Running	0	161m
default	nginx-deployment-5d59d67564-k5p46	1/1	Running	0	6m8s
default	nginx-deployment-5d59d67564-sgzwn	1/1	Running	0	6m8s
default	nginx-deployment-5d59d67564-vpbz7	1/1	Running	0	6m8s
default	nginx-rc-7nktx	1/1	Running	0	46m
default	nginx-rc-ktmwp	1/1	Running	0	19m
default	nginx-rc-mv44p	1/1	Running	0	26m
kube-system	calico-kube-controllers-558995777d-7qrtv	1/1	Running	0	4d5h
kube-system	calico-node-2cqs4	1/1	Running	0	4d5h
kube-system	calico-node-pcxs5	1/1	Running	0	4d5h
kube-system	coredns-c46b5565f-985mx	1/1	Running	0	4d2h
kube-system	coredns-c46b5565f-w9bl8	1/1	Running	0	4d2h
kube-system	nginx-proxy-node-3	1/1	Running	0	4d5h
kube-system	nodelocaldns-hrr9x	1/1	Running	0	4d4h
kube-system	nodelocaldns-nsp9w	1/1	Running	0	4d4h

所有的POD

(4)查看所有的deployment

```
kubectl get deployment
或者
kubectl get deployment -o wide
```

```
root@node-1:~/kubernetes/deploy_work/test# kubectl get deployment -o wide
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
nginx-deployment	3/3	3	3	10m	nginx	nginx:1.7.9	app=nginx

(5)查看所得rs

```
kubectl get rs
```

```
root@node-1:~/kubernetes/deploy_work/test# kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
nginx-abc	2	2	2	167m
nginx-deployment-5d59d67564	3	3	3	11m

(6)查看所得rc

```
kubectl get rc
```

```
root@node-1:~/kubernetes/deploy_work/test# kubectl get rc
```

NAME	DESIRED	CURRENT	READY	AGE
nginx-rc	3	3	3	54m

(7)查看当前deployment部署的nginx版本

```
root@node-1:~/kubernetes/deploy_work/test# kubectl get deployment -o wide
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
nginx-deployment	3/3	3	3	15m	nginx	nginx:1.7.9	app=nginx

(8)动态更行nginx版本

```
kubectl set image deployment nginx-deployment nginx=nginx:1.9.1
```

```
root@node-1:~/kubernetes/deploy_work/test# kubectl get deployment -o wide
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
nginx-deployment	3/3	1	3	17m	nginx	nginx:1.9.1	app=nginx

(9)查看pod的标签

```
kubectl get pods -A --show-labels
```