**K8s网络模型 pod rc service**

**一、部署组合型的应用**

**1、使用配置文件启动replicas集合**

k8s通过Replication Controller来创建和管理各个不同的重复容器集合（实际上是重复的pods）。

Replication Controller会确保pod的数量在运行的时候会一直保持在一个特殊的数字，即replicas的设置。

这个功能类似于Google GCE的实例组管理和AWS的弹性伸缩。

在快速开始中，通过kubectl run以下的YAML文件创建了一个rc运行着nginx：

apiVersion: v1  
kind: ReplicationController  
metadata:  
  name: my-nginx  
spec:  
  replicas: 2  
  template:  
    metadata:  
      labels:  
        app: nginx  
    spec:  
      containers:  
      - name: nginx  
        image: nginx  
        ports:  
        - containerPort: 80

和定义一个pod的YAML文件相比，不同的只是kind的值为ReplicationController，replicas的值需要制定，pod的相关定义在template中，pod的名字不需要显式地指定，因为它们会在rc中创建并赋予名字，点击查看完整的rc定义字段列表：

[replication controller API object](https://htmlpreview.github.io/?https://github.com/kubernetes/kubernetes/HEAD/docs/api-reference/definitions.html" \l "_v1_replicationcontroller" \t "http://www.cnblogs.com/junneyang/p/_blank)

rc可以通过create命令像创建pod一样来创建：

$ kubectl create -f ./nginx-rc.yaml  
replicationcontrollers/my-nginx

和直接创建pod不一样，rc将会替换因为任何原因而被删除或者停止运行的Pod，比如说pod依赖的节点挂了。所以我们推荐使用rc来创建和管理复杂应用，即使你的应用只要使用到一个pod，在配置文件中忽略replicas字段的设置即可。

**2、查看Replication Controller的状态**

可以通过get命令来查看你创建的rc：

$ kubectl get rc  
CONTROLLER   CONTAINER(S)   IMAGE(S)   SELECTOR    REPLICAS  
my-nginx     nginx          nginx      app=nginx   2

这个状态表示，你创建的rc将会确保你一直有两个nginx的副本。

也可以和直接创pPod一样查看创建的Pod状态信息：

$ kubectl get pods  
NAME             READY     STATUS    RESTARTS   AGE  
my-nginx-065jq   1/1       Running   0          51s  
my-nginx-buaiq   1/1       Running   0          51s

**3、删除Replication Controller**

当你想停止你的应用，删除你的rc，可以使用：

$ kubectl delete rc my-nginx  
replicationcontrollers/my-nginx

默认的，这将会删除所有被这个rc管理的pod，如果pod的数量很大，将会花一些时间来完成整个删除动作，如果你想使这些pod停止运行，请指定--cascade=false。

如果你在删除rc之前尝试删除pod，rc将会立即启动新的pod来替换被删除的pod，就像它承诺要做的一样。

**4、Labels**

k8s使用用户自定义的key-value键值对来区分和标识资源集合（就像rc、pod等资源），这种键值对称为label。

在上面的例子中，定义pod的template字段包含了一个简单定义的label，key的值为app，value的值为nginx。所有被创建的pod都会卸载这个label，可以通过-L参数来查看：

$ kubectl get rc my-nginx -L app  
CONTROLLER   CONTAINER(S)   IMAGE(S)   SELECTOR    REPLICAS   APP  
my-nginx     nginx          nginx      app=nginx   2          nginx

默认情况下，pod的label会复制到rc的label，同样地，k8s中的所有资源都支持携带label。

更重要的是，pod的label会被用来创建一个selector，用来匹配过滤携带这些label的pods。

你可以通过kubectl get请求这样一个字段来查看template的格式化输出：

$ kubectl get rc my-nginx -o template --template="{{.spec.selector}}"

map[app:nginx]

你也可以直接指定selector，比如你想在pod template中指明那些不想选中的label，但是你应该确保selector将会匹配这些从pod template中创建的pod的label，并且它将不会匹配其他rc创建的pod。

确保后者最简单的方法是为rc创建一个唯一的label值，然后在pod template的label和selector中都指定这个label。

**二、连接应用到网络中**

**1、k8s中连接容器的模型**

现在，你已经有了组合的、多份副本的应用，你可以将它连接到一个网络上。在讨论k8s联网的方式之前，有必要和Docker中连接网络的普通方式进行一下比较。

默认情况下，Docker使用主机私有网络，所以容器之间可以互相交互，只要它们在同一台机器上。

为了让Docker容器可以进行跨节点的交流，必须在主机的IP地址上为容器分配端口号，之后通过主机IP和端口将信息转发到容器中。

这样一来，很明显地，容器之间必须谨慎地使用和协调端口号的分配，或者动态分配端口号。

在众多开发者之间协调端口号的分配是十分困难的，会将集群级别之外的复杂问题暴露给用户来处理。

在k8s中，假设Pod之间可以互相交流，无论它们是在哪个宿主机上。

我们赋予每个Pod自己的集群私有IP，如此一来你就不需要明确地在Pod之间创建连接，或者将容器的端口映射到主机的端口中。

这意味着，Pod中的容器可以在主机上使用任意彼此的端口，而且集群中的Pods可以在不使用NAT的方式下连接到其他Pod。

本章将会详细描述如何通过这样一个网络连接模型来运行稳定的Services。

本指南使用一个简单的nginx服务来验证以上的概念，在[Jenkins CI application](http://blog.kubernetes.io/2015/07/strong-simple-ssl-for-kubernetes.html" \t "http://www.cnblogs.com/junneyang/p/_blank)有对相同概念的更加详细的说明。

**2、在集群上将Pod连接到网络**

我们之前做过这个例子，但是让我们以网络连接的角度来重新做一次。

创建一个nginx Pod，注意，这个Pod有一个容器端口的说明：

$ cat nginxrc.yaml  
apiVersion: v1  
kind: ReplicationController  
metadata:  
  name: my-nginx  
spec:  
  replicas: 2  
  template:  
    metadata:  
      labels:  
        app: nginx  
    spec:  
      containers:  
      - name: nginx  
        image: nginx  
        ports:  
        - containerPort: 80

这使它可以进入集群上的任意节点，检查节点上运行的Pod：

$ kubectl create -f ./nginxrc.yaml  
$ kubectl get pods -l app=nginx -o wide  
my-nginx-6isf4   1/1       Running   0          2h        e2e-test-beeps-minion-93ly  
my-nginx-t26zt   1/1       Running   0          2h        e2e-test-beeps-minion-93ly

检查你的Pod的IPs：

$ kubectl get pods -l app=nginx -o json | grep podIP  
                "podIP": "10.245.0.15",

                "podIP": "10.245.0.14",

这时，你应该能够通过ssh登录任意节点然后使用curl来连接任一IP（如果节点之间没有处于同一个子网段，无法使用私有IP进行连接的话，就只能在对应节点上使用对应的IP进行连接测试）。

注意，容器并不是真的在节点上使用80端口，也没有任何的NAT规则来路由流量到Pod中。

这意味着你可以在同样的节点上使用同样的containerPort来运行多个nginx Pod，并且可以在集群上的任何Pod或者节点通过这个IP来连接它们。

和Docker一样，端口仍然可以发布到宿主机的接口上，但是因为这个网络连接模型，这个需求就变得很少了。

如果你好奇的话，可以通过这里查看我们是如何做到的：

[how we achieve this](https://github.com/kubernetes/kubernetes/blob/master/docs/admin/networking.md" \l "how-to-achieve-this" \t "http://www.cnblogs.com/junneyang/p/_blank)

**3、创建Service**

现在，在集群上我们有了一个运行着niginx并且有分配IP地址空间的的Pod。

理论上，你可以直接和这些Pod进行交互，但是当节点挂掉之后会发生什么？

这些Pod会跟着节点挂掉，然后RC会在另外一个健康的节点上重新创建新的Pod来代替，而这些Pod分配的IP地址都会发生变化，对于Service类型的服务来说这是一个难题。

k8s上的Service是抽象的，其定义了一组运行在集群之上的Pod的逻辑集合，这些Pod是重复的，复制出来的，所以提供相同的功能。

当Service被创建，会被分配一个唯一的IP地址（也称为集群IP）。这个地址和Service的生命周期相关联，并且当Service是运行的时候，这个IP不会发生改变。

Pods进行配置来和这个Service进行交互，之后Service将会自动做负载均衡到Service中的Pod。

你可以通过以下的YAML文件来为你的两个nginx容器副本创建一个Service：

$ cat nginxsvc.yaml  
apiVersion: v1  
kind: Service  
metadata:  
  name: nginxsvc  
  labels:  
    app: nginx  
spec:  
  ports:  
  - port: 80  
    protocol: TCP  
  selector:

    app: nginx

这个YAML定义创建创建一个Service，带有Label为app=nginx的Pod都将会开放80端口，并将其关联到一个抽象的Service端口。

（targetPort字段是指容器内开放的端口Service通过这个端口连接Pod，port字段是指抽象Service的端口，nodePort为节点上暴露的端口号，不指定的话为随机。）

点击这里查看完整的Service字段列表：

[service API object](https://htmlpreview.github.io/?https://github.com/kubernetes/kubernetes/HEAD/docs/api-reference/definitions.html" \l "_v1_service" \t "http://www.cnblogs.com/junneyang/p/_blank)

现在查看你创建的Service：

$ kubectl get svc  
NAME         CLUSTER\_IP       EXTERNAL\_IP       PORT(S)                SELECTOR     AGE  
kubernetes   10.179.240.1     <none>            443/TCP                <none>       8d

nginxsvc     10.179.252.126   122.222.183.144   80/TCP,81/TCP,82/TCP   run=nginx2   11m

和之前提到的一样，Service是以一组Pod为基础的。

这些Pod通过endpoints字段开放出来。

Service的selector将会不断地进行Pod的Label匹配，结果将会通知一个Endpoints Object，这里创建的也叫做nginxsvc。

当Pod挂掉之后，将会自动从Endpoints中移除，当有新的Pod被Service的selector匹配到之后将会自动加入这个Endpoints。

你可以查看这个Endpoint，注意，这些IP和第一步中创建Pods的时候是一样的：

$ kubectl describe svc nginxsvc  
Name:           nginxsvc  
Namespace:      default  
Labels:         app=nginx  
Selector:       app=nginx  
Type:           ClusterIP  
IP:         10.0.116.146  
Port:           <unnamed>   80/TCP  
Endpoints:      10.245.0.14:80,10.245.0.15:80  
Session Affinity:   None  
No events.

$ kubectl get ep  
NAME         ENDPOINTS

nginxsvc     10.245.0.14:80,10.245.0.15:80

你现在应该可以通过10.0.116.146:80这个IP从集群上的任何一个节点使用curl命令来连接到nginx的Service。

注意，Service的IP是完全虚拟的，如果你想知道它是怎么工作的，请点击：

[service proxy](https://github.com/kubernetes/kubernetes/blob/master/docs/user-guide/services.md" \l "virtual-ips-and-service-proxies" \t "http://www.cnblogs.com/junneyang/p/_blank)

**4、Pod发现并加入到Service中**

k8s提供了两种基本的方式来发现Service：环境变量和DNS。

环境变量是立即可以使用的，DNS则需要kube-dns集群插件。

**5、环境变量**

当Pod运行在一个节点上，kubelet将会为每个激活的Service添加一系列的环境变量。

为你的nginx Pod检查一下环境：

$ kubectl exec my-nginx-6isf4 -- printenv | grep SERVICE  
KUBERNETES\_SERVICE\_HOST=10.0.0.1

KUBERNETES\_SERVICE\_PORT=443

注意，这里没有显示和Service相关的东西，因为这个Pod是在Service之前创建的，这种做法另外一个缺点是，

Scheduler可能会将两个Pod都在同一个机器上启动，这样一来，当节点挂掉之后整个Service也会挂掉。

这里正确的做法是杀死两个Pod，等待RC重新启动新的Pod来代替。

这样一来，Service就在那些副本之前存在，并将环境变量设置到所有的Pod中。

正确的环境变量应为：

$ kubectl scale rc my-nginx --replicas=0; kubectl scale rc my-nginx --replicas=2;  
$ kubectl get pods -l app=nginx -o wide  
NAME             READY   STATUS     RESTARTS   AGE   NODE  
my-nginx-5j8ok   1/1     Running    0         2m    node1  
my-nginx-90vaf   1/1     Running   0          2m    node2  
  
$ kubectl exec my-nginx-5j8ok -- printenv | grep SERVICE  
KUBERNETES\_SERVICE\_PORT=443  
NGINXSVC\_SERVICE\_HOST=10.0.116.146  
KUBERNETES\_SERVICE\_HOST=10.0.0.1  
NGINXSVC\_SERVICE\_PORT=80

**6、DNS**

k8s提供了一个DNS集群服务插件，使用skydns来自动分配DNS给其他的Service。如果你的集群上有运行的话，你可以查看它的状态：

$ kubectl get services kube-dns --namespace=kube-system  
NAME       CLUSTER\_IP      EXTERNAL\_IP   PORT(S)         SELECTOR           AGE  
kube-dns   10.179.240.10   <none>        53/UDP,53/TCP   k8s-app=kube-dns   8d

如果集群上没有运行，那么你可以启用它：[enable it](http://releases.k8s.io/HEAD/cluster/addons/dns/README.md" \l "how-do-i-configure-it" \t "http://www.cnblogs.com/junneyang/p/_blank)

本节剩下的内容是在假设你拥有一个长时间可以使用IP的Service（nginxsvc）并且DNS域名已经通过dns服务（DNS集群服务插件）分配给这个IP的前提下。

所以你可以在集群上的任一节点中使用标准的请求方法来连接Service，现在来创建另外一个Pod进行测试：

$ cat curlpod.yaml  
apiVersion: v1  
kind: Pod  
metadata:  
  name: curlpod  
spec:  
  containers:  
  - image: radial/busyboxplus:curl  
    command:  
      - sleep  
      - "3600"  
    imagePullPolicy: IfNotPresent  
    name: curlcontainer  
  restartPolicy: Always

执行查找nginx Service：

$ kubectl create -f ./curlpod.yaml  
default/curlpod  
$ kubectl get pods curlpod  
NAME      READY     STATUS    RESTARTS   AGE  
curlpod   1/1       Running   0          18s  
  
$ kubectl exec curlpod -- nslookup nginxsvc  
Server:    10.0.0.10  
Address 1: 10.0.0.10  
Name:      nginxsvc  
Address 1: 10.0.116.146

**7、使用加密连接的Service**

到目前为止，我们仅仅是在集群中连接了nginx服务，在将服务开放到Internet上之前，你需要确认双方交流的通道是安全的。

你将会需要以下的东西来确保安全性：

* HTTPS自签名证书（或者你已经有了一个证书）
* 一个nginx服务配置好了使用这个证书
* 一个加密措施使得证书在Pod之间交流使用

你可以通过这里

[nginx https example](https://github.com/kubernetes/kubernetes/blob/master/examples/https-nginx/README.md" \t "http://www.cnblogs.com/junneyang/p/_blank)

来获得完整的配置方式，简要地说，方式如下：

$ make keys secret KEY=/tmp/nginx.key CERT=/tmp/nginx.crt SECRET=/tmp/secret.json  
$ kubectl create -f /tmp/secret.json  
secrets/nginxsecret  
$ kubectl get secrets  
NAME                  TYPE                                  DATA  
default-token-il9rc   [kubernetes.io/service-account-token](http://kubernetes.io/service-account-token" \t "http://www.cnblogs.com/junneyang/p/_blank)   1  
nginxsecret           Opaque                                2

现在修改你的nginx副本来启动一个使用这个证书的HTTPS服务和Service，并且都开放80和443端口：

$ cat nginx-app.yaml  
apiVersion: v1  
kind: Service  
metadata:  
  name: nginxsvc  
  labels:  
    app: nginx  
spec:  
  type: NodePort  
  ports:  
  - port: 8080  
    targetPort: 80  
    protocol: TCP  
    name: http  
  - port: 443  
    protocol: TCP  
    name: https  
  selector:  
    app: nginx  
---  
apiVersion: v1  
kind: ReplicationController  
metadata:  
  name: my-nginx  
spec:  
  replicas: 1  
  template:  
    metadata:  
      labels:  
        app: nginx  
    spec:  
      volumes:  
      - name: secret-volume  
        secret:  
          secretName: nginxsecret  
      containers:  
      - name: nginxhttps  
        image: bprashanth/nginxhttps:1.0  
        ports:  
        - containerPort: 443  
        - containerPort: 80  
        volumeMounts:  
        - mountPath: /etc/nginx/ssl  
          name: secret-volume

nginx-app中值得注意的点有：

* 其包括了RC和SVC的定义在同一个文件中
* nginx服务的HTTP通过80端口，HTTPS通过443端口
* 每个容器都通过挂载在/etc/nginx/ssl卷中的keys来连接。这步是在nginx服务启动之前完成的

$ kubectl delete rc,svc -l app=nginx; kubectl create -f ./nginx-app.yaml  
replicationcontrollers/my-nginx  
services/nginxsvc  
services/nginxsvc

replicationcontrollers/my-nginx

此时，你可以在任意节点上访问nginx服务

$ kubectl get pods -o json | grep -i podip  
    "podIP": "10.1.0.80",  
node $ curl -k [https://10.1.0.80](https://10.1.0.80/" \t "http://www.cnblogs.com/junneyang/p/_blank)  
...

<h1>Welcome to nginx!</h1>

注意在最后一步中，我们是如何使用-k这个参数的，因为我们不知道任何有关nginx服务运行的Pod的证书生成时刻，所以我们要告诉curl忽略别名不匹配。

通过创建一个Service，在Pod发现Service的时候我们将证书中使用的别名和真实地DNS域名关联起来，让我们通过一个Pod来测试（这里会重用这个证书，Pod连接Service只需要nginx.crt）：

$ cat curlpod.yaml  
vapiVersion: v1  
kind: ReplicationController  
metadata:  
  name: curlrc  
spec:  
  replicas: 1  
  template:  
    metadata:  
      labels:  
        app: curlpod  
    spec:  
      volumes:  
      - name: secret-volume  
        secret:  
          secretName: nginxsecret  
      containers:  
      - name: curlpod  
        command:  
        - sh  
        - -c  
        - while true; do sleep 1; done  
        image: radial/busyboxplus:curl  
        volumeMounts:  
        - mountPath: /etc/nginx/ssl  
          name: secret-volume  
  
$ kubectl create -f ./curlpod.yaml  
$ kubectl get pods  
NAME             READY     STATUS    RESTARTS   AGE  
curlpod          1/1       Running   0          2m  
my-nginx-7006w   1/1       Running   0          24m  
  
$ kubectl exec curlpod -- curl [https://nginxsvc](https://nginxsvc/" \t "http://www.cnblogs.com/junneyang/p/_blank) --cacert /etc/nginx/ssl/nginx.crt  
...  
<title>Welcome to nginx!</title>  
...

**8、开放Service**

在你的应用中可能需要将其开放到一个外部的IP中。

k8s提供了两种方式：NodePorts和LoadBalancers。

在上面的最后一部分中，我们已经通过NodePorts方式创建了Service，所以如果你有公网IP，你的nginx https副本就可以在Internet上开放了。

$ kubectl get svc nginxsvc -o json | grep -i nodeport -C 5  
            {  
                "name": "http",  
                "protocol": "TCP",  
                "port": 80,  
                "targetPort": 80,  
                "nodePort": 32188  
            },  
            {  
                "name": "https",  
                "protocol": "TCP",  
                "port": 443,  
                "targetPort": 443,  
                "nodePort": 30645  
            }  
   
$ kubectl get nodes -o json | grep ExternalIP -C 2  
                    {  
                        "type": "ExternalIP",  
                        "address": "104.197.63.17"  
                    }  
--  
                    },  
                    {  
                        "type": "ExternalIP",  
                        "address": "104.154.89.170"  
                    }  
$ curl [https://104.197.63.17:30645](https://104.197.63.17:30645/" \t "http://www.cnblogs.com/junneyang/p/_blank) -k  
...  
<h1>Welcome to nginx!</h1>

现在，我们将通过负载均衡器重新创建一个Service，只需要将nginx-app.yaml文件中的type字段从NodePort改为LoadBalancer即可：

$ kubectl delete rc, svc -l app=nginx  
$ kubectl create -f ./nginx-app.yaml  
$ kubectl get svc nginxsvc  
NAME      CLUSTER\_IP       EXTERNAL\_IP       PORT(S)                SELECTOR     AGE  
nginxsvc  10.179.252.126   162.222.184.144   80/TCP,81/TCP,82/TCP   run=nginx2   13m  
  
$ curl [https://162.22.184.144](https://162.22.184.144/" \t "http://www.cnblogs.com/junneyang/p/_blank) -k  
...  
<title>Welcome to nginx!</title>

EXTERNAL\_IP这列的IP即是可以在公网上访问的IP，CLUSTER\_IP只能在自己的集群上访问到。