《面向服务的软件系统》实验指导书

——实验三：微服务与容器技术

1. Kubernetes 及 docker 安装

（1） 关闭 swap 和防火墙、SeLinux

修改 /etc/fstab 文件，注释掉 swap 那行：

```
1. root@localhost:~ (ssh)

#
# /etc/fstab
# Created by anaconda on Tue Oct 30 05:09:58 2018
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/centos-root /                       xfs     defaults        0 0
UUID=02f8e660-1cd0-4ea1-b1f3-172f65def9e9 /boot                   xfs     defaul
ts        0 0
#/dev/mapper/centos-swap swap                    swap    defaults        0 0
~
~
~
~
~
~
~
~
~
~
~
~
"/etc/fstab" 11L, 467C written
```

保存。

命令执行：

swapoff -a

systemctl stop firewalld

systemctl disable firewalld

setenforce 0

（2）安装 Kubernetes 和 docker

复制执行下面内容：

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=http://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64
enabled=1
```

```
gpgcheck=0
repo_gpgcheck=0
gpgkey=http://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
        http://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

命令行执行：

```
Set SELinux in permissive mode (effectively disabling it)
setenforce 0
sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'
/etc/selinux/config

yum install -y kubelet kubeadm kubectl etcd
--disableexcludes=kubernetes

systemctl enable kubelet && systemctl start kubelet
cat <<EOF >  /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system
```

（3） 复制 docker 镜像压缩包至服务器

使用 scp 命令即可

接着使用命令：

```
docker load -i k8s-images.tar
```

导入镜像

（4） 集群搭建

首先执行命令：

mkdir /etc/cni/net.d –p

用一台云服务器作为 master 节点，使用 ifconfig 获取 IP 后，使用命令：

kubeadm init --pod-network-cidr=192.168.0.0/16 --kubernetes-version=v1.12.1 --apiserver-advertise-address=192.168.56.101 # 换成云服务器 master 的 IP

结果如下图所示：

```
[certificates] valid certificates and keys now exist in "/etc/kubernetes/pki"
[certificates] Generated sa key and public key.
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/admin.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/kubelet.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/controller-manager.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/scheduler.conf"
[controlplane] wrote Static Pod manifest for component kube-apiserver to "/etc/kubernetes/manifests/kube-apiserver.yaml"
[controlplane] wrote Static Pod manifest for component kube-controller-manager to "/etc/kubernetes/manifests/kube-controller-manager.yaml"
[controlplane] wrote Static Pod manifest for component kube-scheduler to "/etc/kubernetes/manifests/kube-scheduler.yaml"
[etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/manifests/etcd.yaml"
[init] waiting for the kubelet to boot up the control plane as Static Pods from directory "/etc/kubernetes/manifests"
[init] this might take a minute or longer if the control plane images have to be pulled
[apiclient] All control plane components are healthy after 25.506538 seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.12" in namespace kube-system with the configuration for the kubelets in the cluster
[markmaster] Marking the node host-0 as master by adding the label "node-role.kubernetes.io/master=''"
[markmaster] Marking the node host-0 as master by adding the taints [node-role.kubernetes.io/master:NoSchedule]
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "host-0" as an annotation
[bootstraptoken] using token: vlcgsa.yudakwdnd5xqpx9q
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstraptoken] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:

  kubeadm join 192.168.56.101:6443 --token vlcgsa.yudakwdnd5xqpx9q --discovery-token-ca-cert-hash sha256:b1111e583d36f2ddc6d932ed629064bb4dad4dbf4149d66b48367092119fdecc

[root@host-0 ~]# kubectl get node
The connection to the server localhost:8080 was refused - did you specify the right host or port?
[root@host-0 ~]#
```

注意上图中的两个红框。
第一个框是需要执行的代码，复制执行即可。
第二个框是其它机器加入集群的代码，复制以留后用。

接着 master 节点即搭建完毕。
可以使用命令：kubectl get nodes　查看集群节点情况：

```
[root@host-0 ~]# kubectl get node
NAME      STATUS    ROLES     AGE   VERSION
host-0    Ready     master    17m   v1.12.2
```

如果 STATUS 状态为 NotReady，请执行下面的代码：
`kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml`

接着在查看 节点状态是否变为 Ready
此问题详情请看：
https://github.com/kubernetes/kubeadm/issues/1031#issuecomment-410253279

其他服务器加入节点后，出现此问题也是如此解决。

接着打开其他服务器，执行第二个框内的命令，加入集群：

```
[root@host-0 ~]#   kubeadm join 192.168.56.101:6443 --token vlcgsa.yudakwdnd5xqpx9q --discovery-token-ca-cert-hash sha256:b1111e583d36f2ddc6d932ed629064bb4dad4dbf4149d66b48367092119fdecc
[preflight] running pre-flight checks
        [WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be used, because the following required kernel modules are not loaded: [ip_vs_rr ip_vs_wrr ip_vs_sh ip_vs] or no builtin ke
rnel ipvs support: map[nf_conntrack_ipv4:{} ip_vs:{} ip_vs_rr:{} ip_vs_wrr:{} ip_vs_sh:{}]
you can solve this problem with following methods:
 1. Run 'modprobe -- ' to load missing kernel modules;
2. Provide the missing builtin kernel ipvs support

[discovery] Trying to connect to API Server "192.168.56.101:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.56.101:6443"
[discovery] Requesting info from "https://192.168.56.101:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "192.168.56.101:6443"
[discovery] Successfully established connection with API Server "192.168.56.101:6443"
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.12" ConfigMap in the kube-system namespace
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[preflight] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "host-0" as an annotation

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

[root@host-0 ~]#
```

最后在 master 节点上，查看集群节点：

```
[root@host-0 ~]# kubectl get nodes
NAME      STATUS   ROLES    AGE     VERSION
host-0    Ready    master   29m     v1.12.2
host-1    Ready    <none>   3m57s   v1.12.2
host-2    Ready    <none>   29s     v1.12.2
[root@host-0 ~]#
```

注意：如果节点加入时提示成功，但是在 master 节点上无法看到该节点，可能是多个节点的 hostname 相同所致。
修改 hostname 后，重启云服务器，接着执行 kubeadm reset，然后重新加入集群即可。

2. Docker 镜像打包、上传

新建文件夹 hello_kube
进入该文件夹，新建文件：
server.js:

```
var http = require('http');
console.log('Heeeee');

var handleRequest = function(request, response) {
   console.log('Received request for URL: ' + request.url);
   response.writeHead(200);
   response.end('Hello World!');
};
var www = http.createServer(handleRequest);
www.listen(8080);
console.log('Listening')
```

Dockerfile：

```
FROM node:6.14.2
EXPOSE 8080
COPY server.js .
CMD node server.js
```

接着在该文件夹，执行命令：
docker build -t hello_world:v2 .

```
➜  hello_kube docker build -t hello_world:v2 .
Sending build context to Docker daemon 7.168 kB
Step 1/4 : FROM node:6.14.2
 ---> 00165cd5d0c0
Step 2/4 : EXPOSE 8080
 ---> Using cache
 ---> 4932330e9478
Step 3/4 : COPY server.js .
 ---> Using cache
 ---> dc7c2fd61af1
Step 4/4 : CMD node server.js
 ---> Using cache
 ---> 8a2f62018d5f
Successfully built 8a2f62018d5f
```

执行 docker images 查看镜像：

```
➜  hello_kube docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
docker.io/septemberhx/helloworld    v1                  8a2f62018d5f        5 days ago          660 MB
hello-node                          v1                  8a2f62018d5f        5 days ago          660 MB
hello_world                         v2                  8a2f62018d5f        5 days ago          660 MB
septemberhx/helloworld              v1                  8a2f62018d5f        5 days ago          660 MB
k8s.gcr.io/kube-proxy               v1.12.1             61afff57f010        3 weeks ago         96.6 MB
k8s.gcr.io/kube-apiserver           v1.12.1             dcb029b5e3ad        3 weeks ago         194 MB
k8s.gcr.io/kube-scheduler           v1.12.1             d773ad20fd80        3 weeks ago         58.3 MB
k8s.gcr.io/kube-controller-manager  v1.12.1             aa2dd57c7329        3 weeks ago         164 MB
k8s.gcr.io/etcd                     3.2.24              3cab8e1b9802        5 weeks ago         220 MB
docker.io/hello-world               latest              4ab4c602aa5e        7 weeks ago         1.84 kB
k8s.gcr.io/coredns                  1.2.2               367cdc8433a4        2 months ago        39.2 MB
docker.io/paulbouwer/hello-kubernetes 1.5               5e4b4221adf5        2 months ago        74.2 MB
docker.io/node                      6.14.2              00165cd5d0c0        4 months ago        660 MB
quay.io/coreos/flannel              v0.10.0-amd64       f0fad859c909        9 months ago        44.6 MB
k8s.gcr.io/pause                    3.1                 da86e6ba6ca1        10 months ago       742 kB
```

前往 https://hub.docker.com/ 注册，以上传自己的镜像
接着在命令行执行 docker login

```
➜  hello_kube docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username (septemberhx): septemberhx
Password:
Login Succeeded
```

接着执行命令上传镜像：
docker tag hello_world:v2 USERNAME/hello_world:v2
docker push USERNAME/hello_world:v2

```
➜  hello_kube docker tag hello_world:v2 septemberhx/hello_world:v2
➜  hello_kube docker push septemberhx/hello_world:v2
The push refers to a repository [docker.io/septemberhx/hello_world]
338b399e838c: Mounted from septemberhx/helloworld
aeaa1edefd60: Mounted from septemberhx/helloworld
6e650662f0e3: Mounted from septemberhx/helloworld
8c825a971eaf: Mounted from septemberhx/helloworld
bf769027dbbd: Mounted from septemberhx/helloworld
f3693db46abb: Mounted from septemberhx/helloworld
bb6d734b467e: Mounted from septemberhx/helloworld
5f349fdc9028: Mounted from septemberhx/helloworld
2c833f307fd8: Mounted from septemberhx/helloworld
v2: digest: sha256:c7c1e959df7a4a773f3b921af52c2f63b4a5a662ab34b7590c006d00350dd623 size: 2214
```

接着登录 https://hub.docker.com/ 即可看见新建的镜像

3. 部署到 Kubernetes 中

在 master 节点上，新建文件：注意将 image 修改为自己刚刚上传的。
hello_world.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: hello-world
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 31611
  selector:
    app: hello-world
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
```

```
containers:
- name: hello-world
  image: septemberhx/helloworld:v1
  ports:
  - containerPort: 8080
```

保存后，执行命令：kubectl create -f ./hello_world.yaml

```
[root@host-0 ~]# vim hello_world.yaml
[root@host-0 ~]# kubectl create -f ./hello_world.yaml
service/hello-world created
deployment.apps/hello-world created
```

接着即可使用命令查看创建情况：
kubectl get pods
kubectl get Deployment
kubectl get svc

```
[root@host-0 ~]# kubectl get pods
NAME                          READY   STATUS              RESTARTS   AGE
hello-world-686b6d9d9b-2szsr  0/1     ContainerCreating   0          36s
hello-world-686b6d9d9b-gl8kg  0/1     ContainerCreating   0          35s
hello-world-686b6d9d9b-pdm4h  0/1     ContainerCreating   0          35s
[root@host-0 ~]# kubectl get pods
NAME                          READY   STATUS              RESTARTS   AGE
hello-world-686b6d9d9b-2szsr  0/1     ContainerCreating   0          114s
hello-world-686b6d9d9b-gl8kg  0/1     ContainerCreating   0          113s
hello-world-686b6d9d9b-pdm4h  0/1     ContainerCreating   0          113s
[root@host-0 ~]# kubectl get Deployment
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-world   3         3         3            0           3m40s
[root@host-0 ~]# kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
hello-world   NodePort    10.111.61.233   <none>        80:31611/TCP   4m14s
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        49m
```

由于涉及到镜像的拉取，所以速度会比较慢

注意：如果创建 pod 过程中，发现 STATUS 一直为 ContainerCreating 状态，可以在 node 节点上，执行命令：systemctl status kubelet –l 进行错误排查。

如果出现下图错误：

```
10月 30 09:26:19 host-2 kubelet[22702]: E1030 09:26:19.598725   22702 cni.go:310] Error adding network: open /run/flannel/subnet.env: no such file or directory
10月 30 09:26:19 host-2 kubelet[22702]: E1030 09:26:19.598746   22702 cni.go:278] Error while adding to cni network: open /run/flannel/subnet.env: no such file or directory
10月 30 09:26:19 host-2 kubelet[22702]: E1030 09:26:19.728039   22702 remote_runtime.go:96] RunPodSandbox from runtime service failed: rpc error: code = Unknown desc = failed to set up sandbox container
"f83abcd0ec0a9a2f41c95850995077 6e897c05fd7f9d254316310c14ba26154e" network for pod "hello-world-686b6d9d9b-kqlmq": NetworkPlugin cni failed to set up pod "hello-world-686b6d9d9b-kqlmq_default" network: o
pen /run/flannel/subnet.env: no such file or directory
10月 30 09:26:19 host-2 kubelet[22702]: E1030 09:26:19.728096   22702 kuberuntime_sandbox.go:65] CreatePodSandbox for pod "hello-world-686b6d9d9b-kqlmq_default(5550f5fa-dc47-11e8-ad5c-080027ebc903)" fail
ed: rpc error: code = Unknown desc = failed to set up sandbox container "f83abcd0ec0a9a2f41c95850995077 6e897c05fd7f9d254316310c14ba26154e" network for pod "hello-world-686b6d9d9b-kqlmq": NetworkPlugin cn
i failed to set up pod "hello-world-686b6d9d9b-kqlmq_default" network: open /run/flannel/subnet.env: no such file or directory
10月 30 09:26:19 host-2 kubelet[22702]: E1030 09:26:19.728113   22702 kuberuntime_manager.go:657] createPodSandbox for pod "hello-world-686b6d9d9b-kqlmq_default(5550f5fa-dc47-11e8-ad5c-080027ebc903)" fai
led: rpc error: code = Unknown desc = failed to set up sandbox container "f83abcd0ec0a9a2f41c95850995077 6e897c05fd7f9d254316310c14ba26154e" network for pod "hello-world-686b6d9d9b-kqlmq": NetworkPlugin c
ni failed to set up pod "hello-world-686b6d9d9b-kqlmq_default" network: open /run/flannel/subnet.env: no such file or directory
10月 30 09:26:19 host-2 kubelet[22702]: E1030 09:26:19.728794   22702 pod_workers.go:186] Error syncing pod 5550f5fa-dc47-11e8-ad5c-080027ebc903 ("hello-world-686b6d9d9b-kqlmq_default(5550f5fa-dc47-11e8-
ad5c-080027ebc903)"), skipping: failed to "CreatePodSandbox" for "hello-world-686b6d9d9b-kqlmq_default(5550f5fa-dc47-11e8-ad5c-080027ebc903)" with CreatePodSandboxError: "CreatePodSandbox for pod \"hello
-world-686b6d9d9b-kqlmq_default(5550f5fa-dc47-11e8-ad5c-080027ebc903)\" failed: rpc error: code = Unknown desc = failed to set up sandbox container \"f83abcd0ec0a9a2f41c95850995077 6e897c05fd7f9d254316310
c14ba26154e\" network for pod \"hello-world-686b6d9d9b-kqlmq\": NetworkPlugin cni failed to set up pod \"hello-world-686b6d9d9b-kqlmq_default\" network: open /run/flannel/subnet.env: no such file or dire
ctory"
```

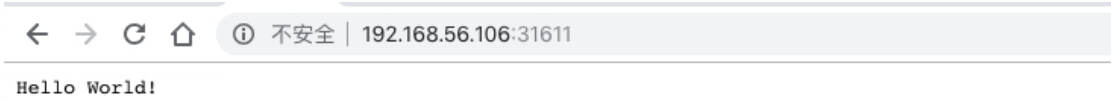
请检查在 master 节点上，执行 kubeadm init 时，是否添加了 –pod-network-cidr 选项。

最后查看 pod：

```
[root@host-0 ~]# kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
hello-world-686b6d9d9b-crskn      1/1     Running   0          3m57s
hello-world-686b6d9d9b-mgkt2      1/1     Running   0          3m57s
hello-world-686b6d9d9b-q4s8p      1/1     Running   0          3m57s
```

```
[root@host-0 ~]# kubectl get pod -o=custom-columns=NAME:.metadata.name,STATUS:.status.phase,NODE:.spec.nodeName --all-namespaces
NAME                      STATUS    NODE
hello-world-686b6d9d9b-crskn     Running   host-2
hello-world-686b6d9d9b-mgkt2     Running   host-2
hello-world-686b6d9d9b-q4s8p     Running   host-1
```

可以访问 node 节点的 31611 端口，即可看见 Hello world。



4. 对 hello-world 进行扩容

```
[root@host-0 ~]# kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
hello-world-686b6d9d9b-crskn      1/1     Running   0          7m49s
hello-world-686b6d9d9b-mgkt2      1/1     Running   0          7m49s
hello-world-686b6d9d9b-q4s8p      1/1     Running   0          7m49s
```

```
[root@host-0 ~]# kubectl get pod -o=custom-columns=NAME:.metadata.name,STATUS:.status.phase,NODE:.spec.nodeName --all-namespaces
NAME                      STATUS    NODE
hello-world-686b6d9d9b-crskn     Running   host-2
hello-world-686b6d9d9b-mgkt2     Running   host-2
hello-world-686b6d9d9b-q4s8p     Running   host-1
```

可以看到，现在一共有三个容器，host-1 上一个，host-2 上两个。

执行命令：
kubectl scale Deployment hello-world --replicas=10
即可将数量调整到十个：

```
[root@host-0 ~]# kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-world-686b6d9d9b-crskn   1/1     Running   0          7m49s
hello-world-686b6d9d9b-mgkt2   1/1     Running   0          7m49s
hello-world-686b6d9d9b-q4s8p   1/1     Running   0          7m49s
[root@host-0 ~]# kubectl scale Deployment hello-world --replicas=10
deployment.extensions/hello-world scaled
[root@host-0 ~]# kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-world-686b6d9d9b-4dsks   1/1     Running   0          22s
hello-world-686b6d9d9b-85798   1/1     Running   0          22s
hello-world-686b6d9d9b-b986k   1/1     Running   0          22s
hello-world-686b6d9d9b-crskn   1/1     Running   0          9m14s
hello-world-686b6d9d9b-dtvxn   1/1     Running   0          22s
hello-world-686b6d9d9b-fhf46   1/1     Running   0          22s
hello-world-686b6d9d9b-gpr7l   1/1     Running   0          22s
hello-world-686b6d9d9b-mgkt2   1/1     Running   0          9m14s
hello-world-686b6d9d9b-q4s8p   1/1     Running   0          9m14s
hello-world-686b6d9d9b-sj9dw   1/1     Running   0          22s
[root@host-0 ~]# kubectl get pod -o=custom-columns=NAME:.metadata.name,STATUS:.status.phase,NODE:.spec.nodeName --all-namespaces
NAME                            STATUS    NODE
hello-world-686b6d9d9b-4dsks    Running   host-1
hello-world-686b6d9d9b-85798    Running   host-2
hello-world-686b6d9d9b-b986k    Running   host-1
hello-world-686b6d9d9b-crskn    Running   host-2
hello-world-686b6d9d9b-dtvxn    Running   host-1
hello-world-686b6d9d9b-fhf46    Running   host-2
hello-world-686b6d9d9b-gpr7l    Running   host-2
hello-world-686b6d9d9b-mgkt2    Running   host-2
hello-world-686b6d9d9b-q4s8p    Running   host-1
hello-world-686b6d9d9b-sj9dw    Running   host-1
coredns-576cbf47c7-hlmcz        Running   host-1
```

即完成了扩容操作。