Exercise 16.2: Detailed Steps

Deploy a Load Balancer

While there are many options, both software and hardware, we will be using an open source tool **HAProxy** to configure a load balancer.

1. Deploy HAProxy. Log into the proxy node. Update the repos then install a the HAProxy software. Answer `yes`, should you the installation ask if you will allow services to restart.

```
student@ha-proxy:~$ sudo apt-get update ; sudo apt-get install -y haproxy vim
<output_omitted>
```

2. Edit the configuration file and add sections for the front-end and back-end servers. We will comment out the second and third cp node until we are sure the proxy is forwarding traffic to the known working cp.

```
student@ha-proxy:~$ sudo vim /etc/haproxy/haproxy.cfg
```

```
....
defaults
        log global                  #<-- Edit these three lines, starting around line 23
        option tcplog
        mode tcp
....
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

frontend proxynode                      #<-- Add the following lines to bottom of file
   bind *:80
   bind *:6443
   stats uri /proxystats
   default_backend k8sServers

backend k8sServers
   balance roundrobin
   server cp  10.128.0.24:6443 check  #<-- Edit these with your IP addresses, port, and hostname
#   server Secondcp  10.128.0.30:6443 check  #<-- Comment out until ready
#   server Thirdcp  10.128.0.66:6443 check   #<-- Comment out until ready
listen stats
   bind :9999
   mode http
   stats enable
   stats hide-version
   stats uri /stats
```

3. Restart the `haproxy` service and check the status. You should see the frontend and backend proxies report being started.

```
student@ha-proxy:~$ sudo systemctl restart haproxy.service
student@ha-proxy:~$ sudo systemctl status haproxy.service
<output_omitted>
Aug 08 18:43:08 ha-proxy systemd[1]: Starting HAProxy Load Balancer...
Aug 08 18:43:08 ha-proxy systemd[1]: Started HAProxy Load Balancer.
Aug 08 18:43:08 ha-proxy haproxy-systemd-wrapper[13602]: haproxy-systemd-wrapper:
```

```
Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy proxynode started.
Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy proxynode started.
Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy k8sServers started.
Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy k8sServers started.
```

4. **On the cp** Edit the `/etc/hosts` file and comment out the old and add a new `k8scp` alias to the IP address of the proxy server.

```
student@cp:~$ sudo vim /etc/hosts

10.128.0.64 k8scp      #<-- Add alias to proxy IP
#10.128.0.24 k8scp     #<-- Comment out the old alias, in case its needed
127.0.0.1 localhost
....
```

5. Use a local browser to navigate to the public IP of your proxy server. The http://34.69.XX.YY:9999/stats is an example your IP address would be different. Leave the browser up and refresh as you run following steps. You can find your public ip using **curl**. Your IP will be different than the one shown below.

```
ha-proxy$ curl ifconfig.io
```

```
34.69.73.159
```

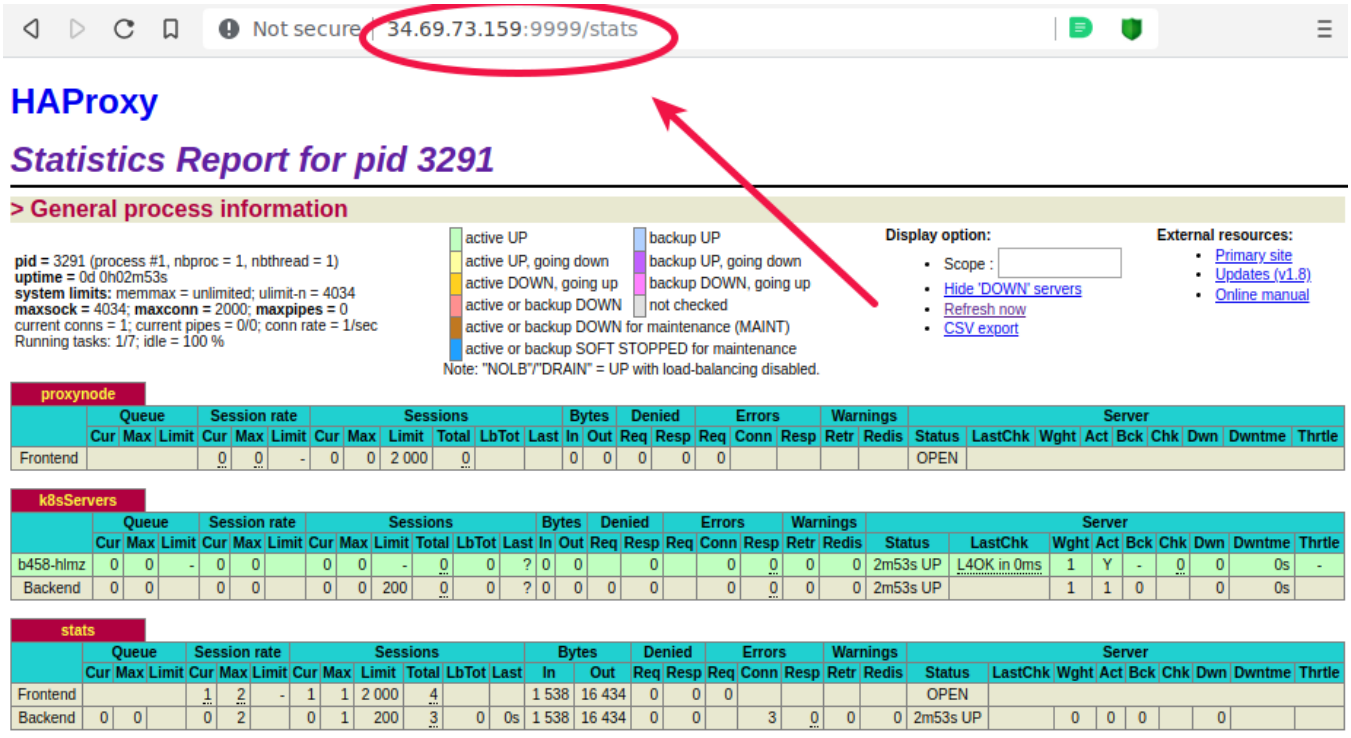

Figure 16.1: **Initial HAProxy Status**

6. Check the node status from the cp node then check the proxy statistics. You should see the byte traffic counter increase.

```
student@cp:~$ kubectl get nodes
```

```
NAME     STATUS   ROLES                  AGE    VERSION
k8scp    Ready    control-plane,master   2d6h   v1.21.1
worker   Ready    <none>                 2d3h   v1.21.1
```

Install Software

We will add two more control planes with stacked **etcd** databases for cluster quorum. You may want to open up two more PuTTY or SSH sessions and color code the terminals to keep track of the nodes.

> Initialize the second cp before adding the third cp

1. Configure and install the kubernetes software on the **second cp**. These are the same steps used when we first set up the cluster. The output to each command has been omitted to make the command clear. You may want to copy and paste from the output of **history** to make these steps easier.

```
student@Secondcp:~$ sudo -i
root@Secondcp:~$ apt-get update && apt-get upgrade -y
```

2. Install a text editor if not already installed.

```
root@Secondcp:~$ apt-get install -y vim
```

3. Install a container engine.

   (a) **IF** you chose Docker for the cp and worker:

```
root@Secondcp:~$ apt-get install -y docker.io
```

   (b) **IF** you chose cri-o for the cp and worker:

```
Please reference the installation lab for detailed installation
and configuration.
```

4. Configure the software repo then install kubernetes packages.

```
root@Secondcp:~$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" \
       >> /etc/apt/sources.list.d/kubernetes.list
root@Secondcp:~$ curl -s \
https://packages.cloud.google.com/apt/doc/apt-key.gpg \
| apt-key add -
root@Secondcp:~$ apt-get update
root@Secondcp:~$ apt-get install -y \
kubeadm=1.21.1-00 kubelet=1.21.1-00 kubectl=1.21.1-00
root@Secondcp:~$ apt-mark hold kubelet kubeadm kubectl
root@Secondcp:~$ exit
```

5. Install the software on the **third cp** using the same commands.

Join Control Plane Nodes

1. Edit the `/etc/hosts` file **ON ALL NODES** to ensure the alias of `k8scp` is set on each node to the proxy IP address. Your IP address may be different.

```
student@cp:~$ sudo vim /etc/hosts
```

```
10.128.0.64 k8scp
#10.128.0.24 k8scp
127.0.0.1 localhost
....
```

2. On the **first cp** create the tokens and hashes necessary to join the cluster. These commands may be in your **history** and easier to copy and paste.

3. Create a new token.

```
student@cp:~$ sudo kubeadm token create
```

```
jasg79.fdh4p2791320cz1g
```

4. Create a new SSL hash.

```
student@cp:~$ openssl x509 -pubkey \
 -in /etc/kubernetes/pki/ca.crt | openssl rsa \
 -pubin -outform der 2>/dev/null | openssl dgst \
 -sha256 -hex | sed 's/^.* //'
```

```
f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd
```

5. Create a new cp certificate to join as a cp instead of as a worker.

```
student@cp:~$ sudo kubeadm init phase upload-certs --upload-certs
```

```
[upload-certs] Storing the certificates in Secret "kubeadm-certs" in the "kube-system" Namespace
[upload-certs] Using certificate key:
5610b6f73593049acddee6b59994360aa4441be0c0d9277c76705d129ba18d65
```

6. On the **second cp** use the previous output to build a **kubeadm join** command. Please be aware that multi-line copy and paste from Windows and some MacOS has paste issues. If you get unexpected output copy one line at a time.

```
student@$Secondcp:~$ sudo kubeadm join k8scp:6443 \
 --token jasg79.fdh4p2791320cz1g \
 --discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd \
 --control-plane --certificate-key \
 5610b6f73593049acddee6b59994360aa4441be0c0d9277c76705d129ba18d65
```

```
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver \
    is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
<output_omitted>
```

7. Return to the first cp node and check to see if the node has been added and is listed as a cp.

```
student@cp:~$ kubectl get nodes
```

```
NAME         STATUS   ROLES                  AGE    VERSION
Secondcp     Ready    control-plane,master   10m    v1.21.1
k8scp        Ready    control-plane,master   2d6h   v1.21.1
worker       Ready    <none>                 2d3h   v1.21.1
```

8. Copy and paste the **kubeadm join** command to the third cp. Then check that the third cp has been added.

```
student@cp:~$ kubectl get nodes
```

```
NAME      STATUS   ROLES                  AGE    VERSION
Thridcp   Ready    control-plane,master   3m     v1.21.1
Secondcp  Ready    control-plane,master   13m    v1.21.1
k8scp     Ready    control-plane,master   2d6h   v1.21.1
worker    Ready    <none>                 2d3h   v1.21.1
```

9. Copy over the configuration file as suggested in the output at the end of the join command. Do this on both newly added cp nodes.

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

10. On the **Proxy node**. Edit the proxy to include all three cp nodes then restart the proxy.

```
student@ha-proxy:~$ sudo vim /etc/haproxy/haproxy.cfg
```

```
....
backend k8sServers
   balance roundrobin
   server cp   10.128.0.24:6443 check
   server Secondcp  10.128.0.30:6443 check  #<-- Edit/Uncomment these lines
   server Thirdcp  10.128.0.66:6443 check  #<--
....
```

```
student@ha-proxy:~$ sudo systemctl restart haproxy.service
```

11. View the proxy statistics. When it refreshes you should see three new back-ends. As you check the status of the nodes using **kubectl get nodes** you should see the byte count increase on each node indicating each is handling some of the requests.

proxynode

	Queue			Session rate			Sessions						Bytes		Deni	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	F
Frontend				0	68	-	11	68	2 000	76			85 805	145 550	0	

k8sServers

	Queue			Session rate			Sessions						Bytes		Denied	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp
master1	0	0	-	0	22		5	23	-	26	26	3s	28 029	37 193		0
master2	0	0	-	0	23		4	23	-	25	25	4m6s	26 015	31 374		0
master3	0	0	-	0	23		2	22	-	25	25	10s	31 761	76 983		0
Backend	0	0		0	68		11	68	200	76	76	3s	85 805	145 550	0	0

stats

	Queue			Session rate			Sessions						Bytes		Denied	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp
Frontend				1	2	-	1	1	2 000	7			3 205	56 260	0	
Backend	0	0		0	2		0	1	200	6	0	0s	3 205	56 260	0	

Figure 16.2: **Multiple HAProxy Status**

12. View the logs of the newest **etcd** pod. Leave it running, using the **-f** option in one terminal while running the following commands in a different terminal. As you have copied over the cluster admin file you can run **kubectl** on any cp.

```
student@cp:~$ kubectl -n kube-system get pods |grep etcd
```

```
etcd-cp              1/1     Running   0          2d12h
etcd-Secondcp        1/1     Running   0          22m
etcd-Thirdcp         1/1     Running   0          18m
```

```
student@cp:~$ kubectl -n kube-system logs -f etcd-Thirdcp
```

```
....
2019-08-09 01:58:03.768858 I | mvcc: store.index: compact 300473
2019-08-09 01:58:03.770773 I | mvcc: finished scheduled compaction at 300473 (took 1.286565ms)
2019-08-09 02:03:03.766253 I | mvcc: store.index: compact 301003
2019-08-09 02:03:03.767582 I | mvcc: finished scheduled compaction at 301003 (took 995.775μs)
2019-08-09 02:08:03.785807 I | mvcc: store.index: compact 301533
2019-08-09 02:08:03.787058 I | mvcc: finished scheduled compaction at 301533 (took 913.185μs)
```

13. Log into one of the **etcd** pods and check the cluster status, using the IP address of each server and port 2379. Your IP addresses may be different. Exit back to the node when done.

```
student@cp:~$ kubectl -n kube-system exec -it etcd-cp -- /bin/sh
```

etcd pod

```
/ # ETCDCTL_API=3 etcdctl -w table \
  --endpoints 10.128.0.66:2379,10.128.0.24:2379,10.128.0.30:2379 \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/server.crt \
  --key /etc/kubernetes/pki/etcd/server.key \
  endpoint status
+------------------+------------------+---------+---------+-----------+-----------+------------+
|     ENDPOINT     |        ID        | VERSION | DB SIZE | IS LEADER | RAFT TERM | RAFT INDEX |
+------------------+------------------+---------+---------+-----------+-----------+------------+
| 10.128.0.66:2379 | 2331065cd4fb02ff |  3.3.10 |   24 MB |      true |        11 |     392573 |
| 10.128.0.24:2379 | d2620a7d27a9b449 |  3.3.10 |   24 MB |     false |        11 |     392573 |
| 10.128.0.30:2379 | ef44cc541c5f37c7 |  3.3.10 |   24 MB |     false |        11 |     392573 |
+------------------+------------------+---------+---------+-----------+-----------+------------+
```

Test Failover

Now that the cluster is running and has chosen a leader we will shut down docker, which will stop all containers on that node. This will emulate an entire node failure. We will then view the change in leadership and logs of the events.

1. If you used Docker, Shut down the service on the node which shows `IS LEADER` set to `true`.

```
student@cp:~$ sudo systemctl stop docker.service
```

If you chose cri-o as the container engine then the cri-o service and conmon processes are distinct. It may be easier to reboot the node and refresh the HAProxy web page until it shows the node is down. It may take a while for the node to finish the boot process. The second and third cp should work the entire time.

```
student@cp:~$ sudo reboot
```

2. You will probably note the **logs** command exited when the service shut down. Run the same command and, among other output, you'll find errors similar to the following. Note the messages about losing the leader and electing a new one, with an eventual message that a peer has become inactive.

```
student@cp:~$ kubectl -n kube-system logs -f etcd-Thirdcp
....
2019-08-09 02:11:39.569827 I | raft: 2331065cd4fb02ff [term: 9] received a MsgVote message with higher \
                               term from ef44cc541c5f37c7 [term: 10]
2019-08-09 02:11:39.570130 I | raft: 2331065cd4fb02ff became follower at term 10
2019-08-09 02:11:39.570148 I | raft: 2331065cd4fb02ff [logterm: 9, index: 355240, vote: 0] cast MsgVote \
                               for ef44cc541c5f37c7 [logterm: 9, index: 355240] at term 10
2019-08-09 02:11:39.570155 I | raft: raft.node: 2331065cd4fb02ff lost leader d2620a7d27a9b449 at term 10
2019-08-09 02:11:39.572242 I | raft: raft.node: 2331065cd4fb02ff elected leader ef44cc541c5f37c7 at \
                               term 10
2019-08-09 02:11:39.682319 W | rafthttp: lost the TCP streaming connection with peer d2620a7d27a9b449 \
                               (stream Message reader)
2019-08-09 02:11:39.682635 W | rafthttp: lost the TCP streaming connection with peer d2620a7d27a9b449 \
                               (stream MsgApp v2 reader)
2019-08-09 02:11:39.706068 E | rafthttp: failed to dial d2620a7d27a9b449 on stream MsgApp v2 \
                               (peer d2620a7d27a9b449 failed to find local node 2331065cd4fb02ff)
2019-08-09 02:11:39.706328 I | rafthttp: peer d2620a7d27a9b449 became inactive (message send to peer failed)
....
```

3. View the proxy statistics. The proxy should show the first cp as down, but the other cp nodes remain up.

k8sServers

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings				
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	W
master1	0	0	-	0	22		0	23	-	173	129	12m18s	11 110 233	62 695 354		0		0	19	44	0	12m DOWN	L4CON in 0ms	
master2	0	0	-	0	23		6	23	-	129	129	12m6s	299 280	2 703 547		0		0	0	0	0	4h15m UP	L4OK in 0ms	
master3	0	0	-	0	23		5	22	-	128	128	12m23s	362 790	6 078 463		0		0	1	0	0	4h15m UP	L4OK in 0ms	
Backend	0	0		0	68		11	68	200	387	386	12m6s	11 772 303	71 477 364	0	0		0	20	44	0	4h15m UP		

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings				
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght
Frontend				1	2	-	1	1	2 000	10			4 885	93 693	0	0	0					OPEN		
Backend	0	0		0	2		0	1	200	9	0	0s	4 885	93 693	0	0		9	0	0	0	4h15m UP		0

Figure 16.3: **HAProxy Down Status**

4. View the status using **etcdctl** from within one of the running **etcd** pods. You should get an error for the endpoint you shut down and a new leader of the cluster.

```
student@Secondcp:~$ kubectl -n kube-system exec -it etcd-Secondcp -- /bin/sh
```

etcd pod

```
/ # ETCDCTL_API=3 etcdctl -w table \
  --endpoints 10.128.0.66:2379,10.128.0.24:2379,10.128.0.30:2379 \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/server.crt \
  --key /etc/kubernetes/pki/etcd/server.key \
  endpoint status
```

```
Failed to get the status of endpoint 10.128.0.66:2379 (context deadline exceeded)
+------------------+------------------+---------+---------+-----------+-----------+------------+
|     ENDPOINT     |        ID        | VERSION | DB SIZE | IS LEADER | RAFT TERM | RAFT INDEX |
+------------------+------------------+---------+---------+-----------+-----------+------------+
| 10.128.0.24:2379 | d2620a7d27a9b449 |  3.3.10 |   24 MB |      true |        12 |     395729 |
| 10.128.0.30:2379 | ef44cc541c5f37c7 |  3.3.10 |   24 MB |     false |        12 |     395729 |
+------------------+------------------+---------+---------+-----------+-----------+------------+
```

5. Turn the docker service back on. You should see the peer become active and establish a connection.

```
student@cp:~$ sudo systemctl start docker.service

student@cp:~$ kubectl -n kube-system logs -f etcd-ThirdControl Plane
```

```
....
2019-08-09 02:45:11.337669 I | rafthttp: peer d2620a7d27a9b449 became active
2019-08-09 02:45:11.337710 I | rafthttp: established a TCP streaming connection with peer\
                             d2620a7d27a9b449 (stream MsgApp v2 reader)
....
```

6. View the **etcd** cluster status again. Experiment with how long it takes for the **etcd** cluster to notice failure and choose a new leader with the time you have left.