Exercise 8.1: Troubleshooting: Monitor Applications

Overview

Troubleshooting can be difficult in a multi-node, decoupled and transient environment. Add in the rapid pace of change and it becomes more difficult. Instead of focusing and remembering a particular error and the fix it may be more useful to learn a flow of troubleshooting and revisit assumptions until the pace of change slows and various areas further mature.

1. View the `secondapp` pod, it should show as `Running`. This may not mean the application within is working properly, but that the pod is running. The restarts are due to the command we have written to run. The pod exists when done, and the controller restarts another container inside. The count depends on how long the labs have been running.

```
student@ckad-1/app2:~$ cd
student@ckad-1:~$ kubectl get pods secondapp
NAME                        READY     STATUS      RESTARTS   AGE
secondapp                   2/2       Running     49         2d
```

2. Look closer at the pod. Working slowly through the output check each line. If you have issues, are other pods having issues on the same node or volume? Check the state of each container. Both `busy` and `webserver` should report as `Running`. Note `webserver` has a restart count of zero while `busy` has a restart count of 49. We expect this as, in our case, the pod has been running for 49 hours.

```
student@ckad-1:~$ kubectl describe pod secondapp

Name:         secondapp
Namespace:    default
Node:         ckad-2-wdrq/10.128.0.2
Start Time:   Fri, 13 Apr 2018 20:34:56 +0000
Labels:       example=second
Annotations:  <none>
Status:       Running
IP:           192.168.55.91
Containers:
  webserver:
<output_omitted>
    State:          Running
      Started:      Fri, 13 Apr 2018 20:34:58 +0000
    Ready:          True
    Restart Count:  0
<output_omitted>

  busy:
<output_omitted>

    State:          Running
      Started:      Sun, 15 Apr 2018 21:36:20 +0000
    Last State:     Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Sun, 15 Apr 2018 20:36:18 +0000
      Finished:     Sun, 15 Apr 2018 21:36:18 +0000
    Ready:          True
    Restart Count:  49
    Environment:    <none>
```

3. There are three values for conditions. Check that the pod reports Initialized, Ready and scheduled.

```
<output_omitted>
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
<output_omitted>
```

4. Check if there are any events with errors or warnings which may indicate what is causing any problems.

```
Events:
  Type    Reason   Age                From                   Message
  ----    ------   ----               ----                   -------
  Normal  Pulling  34m (x50 over 2d)  kubelet, ckad-2-wdrq   pulling
image "busybox"
  Normal  Pulled   34m (x50 over 2d)  kubelet, ckad-2-wdrq   Successfully
pulled image "busybox"
  Normal  Created  34m (x50 over 2d)  kubelet, ckad-2-wdrq   Created
container
  Normal  Started  34m (x50 over 2d)  kubelet, ckad-2-wdrq   Started
container
```

5. View each container log. You may have to sift errors from expected output. Some containers may have no output at all, as is found with `busy`.

```
student@ckad-1:~$ kubectl logs secondapp webserver

192.168.55.0 - - [13/Apr/2018:21:18:13 +0000] "GET / HTTP/1.1" 200
612 "-" "curl/7.47.0" "-"
192.168.55.0 - - [13/Apr/2018:21:20:35 +0000] "GET / HTTP/1.1" 200
612 "-" "curl/7.53.1" "-"
127.0.0.1 - - [13/Apr/2018:21:25:29 +0000] "GET" 400 174 "-" "-" "-"
127.0.0.1 - - [13/Apr/2018:21:26:19 +0000] "GET index.html" 400 174
"-" "-" "-"
<output_omitted>


student@ckad-1:~$ kubectl logs secondapp busy

student@ckad-1:~$
```

6. Check to make sure the container is able to use DNS and communicate with the outside world. Remember we still have limited the UID for `secondapp` to be UID **2000**, which may prevent some commands from running. It can also prevent an application from completing expected tasks, and other errors.

```
student@ckad-1:~$ kubectl exec  -it  secondapp -c busy -- sh
```

On Container

```
/ $ nslookup www.linuxfoundation.org

/ $ nslookup www.linuxfoundation.org
Server:                10.96.0.10
Address:        10.96.0.10:53

Non-authoritative answer:
Name:        www.linuxfoundation.org
Address: 23.185.0.2

*** Can't find www.linuxfoundation.org: No answer


/ $ cat /etc/resolv.conf
```

```
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local
cluster.local c.endless-station-188822.internal
google.internal
options ndots:5
```

Test access to a remote node using **nc (NetCat)**. There are several options to **nc** which can help troubleshoot if the problem is the local node, something between nodes or in the target. In the example below the connect never completes and a **control-c** was used to interrupt.

7.
```
/ $ nc www.linux.com 25
^Cpunt!
```

8. Test using an IP address in order to narrow the issue to name resolution. In this case the IP in use is a well known IP for Google's DNS servers. The following example shows that Internet name resolution is working, but our UID issue prevents access to the `index.html` file.

```
/ $ wget http://www.linux.com/
Connecting to www.linux.com (151.101.45.5:80)
Connecting to www.linux.com (151.101.45.5:443)
wget: can't open 'index.html': Permission denied

/ $ exit
```

9. Make sure traffic is being sent to the correct Pod. Check the details of both the service and endpoint. Pay close attention to ports in use as a simple typo can prevent traffic from reaching the proper pod. Make sure labels and selectors don't have any typos as well.

```
student@ckad-1:~$ kubectl get svc
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1        <none>        443/TCP        10d
nginx        ClusterIP      10.108.95.67     <none>        443/TCP        10d
registry     ClusterIP      10.105.119.236   <none>        5000/TCP       10d
secondapp    LoadBalancer   10.109.26.21     <pending>     80:32000/TCP   1d
thirdpage    NodePort       10.109.250.78    <none>        80:31230/TCP   1h

student@ckad-1:~$ kubectl get svc secondapp -o yaml
<output_omitted>
  clusterIP: 10.109.26.21
  externalTrafficPolicy: Cluster
  ports:
  - nodePort: 32000
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    example: second
<output_omitted>
```

10. Verify an endpoint for the service exists and has expected values, including namespaces, ports and protocols.

```
student@ckad-1:~$ kubectl get ep
NAME         ENDPOINTS            AGE
kubernetes   10.128.0.3:6443      10d
nginx        192.168.55.68:443    10d
registry     192.168.55.69:5000   10d
secondapp    192.168.55.91:80     1d
thirdpage    192.168.241.57:80    1h
```

```
student@ckad-1:~$ kubectl get ep secondapp -o yaml

apiVersion: v1
kind: Endpoints
metadata:
  creationTimestamp: 2018-04-14T05:37:32Z
<output_omitted>
```

11. If the containers, services and endpoints are working the issue may be with an infrastructure service like **kube-proxy**. Ensure it's running, then look for errors in the logs. As we have two nodes we will have two proxies to look at. As we built our cluster with **kubeadm** the proxy runs as a container. On other systems you may need to use **journalctl** or look under `/var/log/kube-proxy.log`.

```
student@ckad-1:~$ ps -elf |grep kube-proxy

4 S root      2864  2847  0  80   0 - 14178 -      15:45 ?
 00:00:56 /usr/local/bin/kube-proxy --config=/var/lib/kube-proxy/config.conf
0 S student  23513 18282  0  80   0 -  3236 pipe_w 22:49 pts/0
 00:00:00 grep --color=auto kube-proxy


student@ckad-1:~$ journalctl -a | grep proxy

Apr 15 15:44:43 ckad-2-nzjr audit[742]: AVC apparmor="STATUS"
 operation="profile_load" profile="unconfined" \
  name="/usr/lib/lxd/lxd-bridge-proxy" pid=742 comm="apparmor_parser"
Apr 15 15:44:43 ckad-2-nzjr kernel: audit: type=1400
 audit(1523807083.011:11): apparmor="STATUS" \
  operation="profile_load" profile="unconfined" \
   name="/usr/lib/lxd/lxd-bridge-proxy" pid=742 comm="apparmor_parser"
Apr 15 15:45:17 ckad-2-nzjr kubelet[1248]: I0415 15:45:17.153670
 1248 reconciler.go:217] operationExecutor.VerifyControllerAttachedVolume\
  started for volume "xtables-lock" \
   (UniqueName: "kubernetes.io/host-path/e701fc01-38f3-11e8-a142-\
   42010a800003-xtables-lock") \
   pod "kube-proxy-t8k4w" (UID: "e701fc01-38f3-11e8-a142-42010a800003")
```

12. Look at both of the proxy logs. Lines which begin with the character **I** are `info`, **E** are `errors`. In this example the last message says access to listing an endpoint was denied by RBAC. It was because a default installation via Helm wasn't RBAC aware. If not using command line completion, view the possible pod names first.

```
student@ckad-1:~$ kubectl -n kube-system get pod

student@ckad-1:~$ kubectl -n kube-system logs kube-proxy-fsdfr

I0405 17:28:37.091224       1 feature_gate.go:190] feature gates: map[]
W0405 17:28:37.100565       1 server_others.go:289] Flag proxy-mode=""
unknown, assuming iptables proxy
I0405 17:28:37.101846       1 server_others.go:138] Using iptables Proxier.
I0405 17:28:37.121601       1 server_others.go:171] Tearing down
inactive rules.
<output_omitted>
E0415 15:45:17.086081       1 reflector.go:205] \
  k8s.io/kubernetes/pkg/client/informers/informers_generated/
  internalversion/factory.go:85: \
  Failed to list *core.Endpoints: endpoints is forbidden: \
    User "system:serviceaccount:kube-system:kube-proxy" cannot \
   list endpoints at the cluster scope:\
 [clusterrole.rbac.authorization.k8s.io "system:node-proxier" not found, \
   clusterrole.rbac.authorization.k8s.io "system:basic-user" not found,
 clusterrole.rbac.authorization.k8s.io \
 "system:discovery" not found]
```

13. Check that the proxy is creating the expected rules for the problem service. Find the destination port being used for the service, **30195** in this case.

```
student@ckad-1:~$ sudo iptables-save |grep secondapp

-A KUBE-NODEPORTS -p tcp -m comment --comment "default/secondapp:" \
-m tcp --dport 30195 -j KUBE-MARK-MASQ
-A KUBE-NODEPORTS -p tcp -m comment --comment "default/secondapp:" \
-m tcp --dport 30195 -j KUBE-SVC-DAASHM5XQZF5XI3E
-A KUBE-SERVICES ! -s 192.168.0.0/16 -d 10.109.26.21/32 -p tcp \
-m comment --comment "default/secondapp: \
      cluster IP" -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.109.26.21/32 -p tcp -m comment --comment \
"default/secondapp: cluster IP" -m tcp \
      --dport 80 -j KUBE-SVC-DAASHM5XQZF5XI3E
<output_omitted>
```

14. Ensure the proxy is working by checking the port targeted by **iptables**. If it fails open a second terminal and view the proxy logs when making a request as it happens.

```
student@ckad-1:~$ curl localhost:32000

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<output_omitted>
```