Troubleshooting

Lesson Topics:

- ❏ Logging
- ❏ Troubleshooting

References:

- ❏ kubernetes.io

1. Logging

- view logs from the node perspective
  - `journalctl`
- view logs from Pod perspective
  - `kubectl logs`
  - `kubetail`
  - sidecar container
    - read the logs of another container in a pod
    - push log to a centralize system (ElasticSearch, Splunk)

1.1. Log file locations

- systemd based Kubernetes cluster
  - `journalctl -u kubelet | less`
  - `journalctl -flu kubelet`
- major Kubernetes processes now run in containers
  - kube-apiserver, kube-dns, kube-proxy
  - `/var/log/containers` → various container logs
  - `/var/log/pods` → symlinks to `/var/log/containers`
    - `kube-apiserver.log` → responsible for serving the API
    - `kube-scheduler.log` → responsible for making scheduling decisions
    - `kube-controller-manager.log` → controller that manages replication controllers
    - `/var/log/kube-proxy.log` → responsible for service load balancing
- more readings:
  - https://kubernetes.io/docs/tasks/debug-application-cluster/debug-service/
  - https://kubernetes.io/docs/tasks/debug-application-cluster/determine-reason-pod-failure/

1.2. Viewing Pods logs output

- container standard out
  - `kubectl logs <Pod>`
  - if there is no standard out, you would not see any output
- logs are destroyed if the Pod is destroyed
- if a container got restarted, you can see the logs from the previous instance
  - `kubectl logs <Pod> --previous`
- view logs from a group of Pods based on a label selector
  - `kubectl -n kube-system logs -f --selector k8s-app=cilium`
  - `kubetail -n kube-system  --selector k8s-app=cilium`

2. Troubleshooting

- Troubleshooting can be difficult in kubernetes:
  - multi-node workers
  - decoupled environment
  - transient environment

2.1. Debug Services

- an issue that comes up rather frequently for new installations of Kubernetes is that a Service is not working properly
- background
  - you've run your `Deployment` and created a `Service`, but you get no response when you try to access it
  - setup the environment:
    - `kubectl apply -f namespace.yaml`
    - `kubectl apply -f deployment.yaml`
    - `kubectl get pods -l app=hostnames`

```
NAME                         READY   STATUS    RESTARTS   AGE
hostnames-745bc867c6-6k6vg   1/1     Running   0          64s
hostnames-745bc867c6-6prnm   1/1     Running   0          64s
hostnames-745bc867c6-fnnv7   1/1     Running   0          64s
```

2.1. Debug Services

- does the `Service` exist?
  - we did not actually create a Service yet - that is intentional
  - what would happen if I tried to access a non-existent Service?

```
kubectl run -it --rm --restart=Never alpine --image=alpine sh
wget -O- hostnames
---
wget: bad address 'hostnames'

kubectl get svc hostnames
---
Error from server (NotFound): services "hostnames" not found
```

- create the service

```
kubectl apply -f service.yaml
```

2.1. Debug Services

- does the Service work by DNS?

```
kubectl run -it --rm --restart=Never alpine --image=alpine sh
nslookup hostnames
wget -O- hostnames
```

2.1. Debug Services

- does any `Service` exist in DNS?
  - If the above still fails - DNS lookups are not working for your Service
  - we can take a step back and see what else is not working
  - the Kubernetes master Service should always work
  - `nslookup kubernetes.default`

```
Name:      kubernetes.default
Address 1: 10.96.0.1 kubernetes.default.svc.cluster.local
```

  - If this fails, you might need to go to the kube-proxy, but instead of debugging your own Service, debug DNS

2.1. Debug Services

- does the Service work by IP?
  - assuming we can confirm that DNS works, the next thing to test is whether your Service works at all
  - from a node in your cluster, access the Service's IP (from kubectl get above)
    - `curl 10.104.195.93:80`
  - If your Service is working, you should get correct responses. If not, there are a number of things that could be going wrong

2.1. Debug Services

- is the Service correct ?
  - `kubectl get service hostnames -o json`
    - is the port you are trying to access in `spec.ports[]` ?
    - is the targetPort correct for your Pods (many Pods choose to use a different port than the Service)?
    - is the port's protocol the same as the Pod's?

2.1. Debug Services

- does the Service have any Endpoints?
  - if you got this far, we assume that you have confirmed that your Service exists and is resolved by DNS
  - now let's check that the Pods you ran are actually being selected by the Service
  - `kubectl get pods -l app=hostnames`
    - check STATUS, RESTARTS, AGE
  - `kubectl get endpoints hostnames`

```
NAME          ENDPOINTS                                                  AGE
hostnames   10.244.1.21:9376,10.244.1.236:9376,10.244.1.97:9376   152m
```

  - this confirms that the endpoints controller has found the correct Pods for your Service
  - If the hostnames row is blank, you should check that the `spec.selector` field of your Service actually selects for `metadata.labels` values on your Pods

2.1. Debug Services

- Are the Pods working?
  - At this point, we know that your Service exists and has selected your Pods
  - Let's check that the Pods are actually working - we can bypass the Service mechanism and go straight to the Pods

```
kubectl get pods -l app=hostnames -o wide

wget -qO- 10.244.1.97:9376
wget -qO- 10.244.1.236:9376
wget -qO- 10.244.1.21:9376
```

  - we expect each Pod in the Endpoints list to return its own hostname
  - you might find kubectl logs to be useful or kubectl exec directly to your Pods and check service from there.

2.1. Debug Services

- is the kube-proxy working?
  - if you get here, your Service is running, has Endpoints, and your Pods are actually serving
  - at this point, the whole Service proxy mechanism is suspect
  - let's confirm it, piece by piece
- is kube-proxy running?
  - confirm that kube-proxy is running on your Nodes
  - `kubectl -n kube-system get pods`
  - next, confirm that it is not failing something obvious, like contacting the master
  - to do this, you'll have to look at the logs
  - `kubectl -n kube-system logs -f --selector k8s-app=kube-proxy`

2.1. Debug Services

- is kube-proxy writing iptables rules?
  - one of the main responsibilities of `kube-proxy` is to write the `iptables` rules which implement Services
  - let's check that those rules are getting written.

```
iptables-save | grep hostname
-A KUBE-SERVICES ! -s 10.244.0.0/16 -d 10.99.117.246/32 -p tcp -m comment --comment "kube-system/hostnames: cluster
IP" -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.99.117.246/32 -p tcp -m comment --comment "kube-system/hostnames: cluster IP" -m tcp --dport
80 -j KUBE-SVC-J7JSTV534HRZRQVN
```

```
iptables-save | grep KUBE-SVC-J7JSTV534HRZRQVN
:KUBE-SVC-J7JSTV534HRZRQVN - [0:0]
-A KUBE-SERVICES -d 10.99.117.246/32 -p tcp -m comment --comment "kube-system/hostnames: cluster IP" -m tcp --dport
80 -j KUBE-SVC-J7JSTV534HRZRQVN
-A KUBE-SVC-J7JSTV534HRZRQVN -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-YZUBK75TZVFH2PN7
-A KUBE-SVC-J7JSTV534HRZRQVN -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-C56UNLDVDVO5WCO7
-A KUBE-SVC-J7JSTV534HRZRQVN -j KUBE-SEP-7OPQNNUYFXXBG55
```

2.1. Debug Services

```
iptables-save | grep KUBE-SVC-J7JSTV534HRZRQVN
-A KUBE-SVC-J7JSTV534HRZRQVN -m statistic --mode random --probability 0.33332999982 -j
KUBE-SEP-YZUBK75TZVFH2PN7
-A KUBE-SVC-J7JSTV534HRZRQVN -m statistic --mode random --probability 0.50000000000 -j
KUBE-SEP-C56UNLDVDVO5WCO7
-A KUBE-SVC-J7JSTV534HRZRQVN -j KUBE-SEP-7OPQNNUYFXXBG55
```

```
 iptables-save | grep KUBE-SEP-YZUBK75TZVFH2PN7
:KUBE-SEP-YZUBK75TZVFH2PN7 - [0:0]
-A KUBE-SEP-YZUBK75TZVFH2PN7 -s 10.244.1.4/32 -j KUBE-MARK-MASQ
-A KUBE-SEP-YZUBK75TZVFH2PN7 -p tcp -m tcp -j DNAT --to-destination 10.244.1.4:9376
```

2.1. Debug Services

- Is kube-proxy proxying?
  - Assuming you do see the above rules, try again to access your Service by IP

```
curl 10.244.1.4:9376
---
hostnames-745bc867c6-6k6vg
```

- If this still fails, look at the kube-proxy logs for specific lines like:

```
Setting endpoints for default/hostnames:default to [10.244.0.5:9376
10.244.0.6:9376 10.244.0.7:9376]
```

- If you don't see those, try restarting kube-proxy with the -v flag set to 4, and then look at the logs again

2.2. Determine the Reason for Pod Failure

- Termination messages → write information about fatal events
- In most cases, information that you put in a termination message should also be written to the general `Kubernetes logs`

2.2. Determine the Reason for Pod Failure

- create a Pod based on the YAML configuration file:
  - `kubectl apply -f termination-demo-pod.yaml`
- Display information about the Pod
  - `kubectl get pod termination-demo`
- Display detailed information about the Pod
  - `kubectl get pod termination-demo --output=yaml`
  - the output includes the “Sleep expired” message

```
lastState:
  terminated:
    containerID: docker://f7b7c1d8aa8728fabc5fa554d671427245885afa6c2a33bced116912b51d65d1
    exitCode: 0
    finishedAt: "2019-08-14T21:04:44Z"
    message: |
      Sleep expired
    reason: Completed
    startedAt: "2019-08-14T21:04:34Z"
```

2.2. Determine the Reason for Pod Failure

- More structured information about the events, Pod Last State
  - `kubectl describe pod termination-demo`

```
    State:       Waiting
      Reason:    CrashLoopBackOff
    Last State:  Terminated
      Reason:    Completed
      Message:   Sleep expired

      Exit Code:    0
      Started:      Wed, 14 Aug 2019 21:07:47 +0000
      Finished:     Wed, 14 Aug 2019 21:07:57 +0000
    Ready:          False
    Restart Count:  5
    Environment:    <none>
```

2.2. Determine the Reason for Pod Failure

```
Events:
  Type     Reason     Age                    From               Message
  ----     ------     ----                   ----               -------
  Normal   Scheduled  7m2s                   default-scheduler  Successfully assigned kube-system/termination-demo
to node01
  Normal   Started    5m19s (x4 over 6m41s)  kubelet, node01    Started container termination-demo-container
  Normal   Pulling    4m26s (x5 over 7m1s)   kubelet, node01    Pulling image "debian"
  Normal   Pulled     4m23s (x5 over 6m41s)  kubelet, node01    Successfully pulled image "debian"
  Normal   Created    4m23s (x5 over 6m41s)  kubelet, node01    Created container termination-demo-container
  Warning  BackOff    2m (x16 over 6m16s)    kubelet, node01    Back-off restarting failed container
```

2.3. Check Pod CPU and Memory usage

- kubectl top <pods>

```
kubectl top pod
NAME                                    CPU(cores)   MEMORY(bytes)
cilium-fb874                            44m          187Mi
cilium-mrzl5                            33m          188Mi
cilium-operator-6df4996bbc-w6lpr        3m           16Mi
cilium-rzqkx                            37m          192Mi
coredns-5c98db65d4-ct7gz                4m           8Mi
etcd-master                             28m          67Mi
```

2.3. Check Liveness and Readiness probes

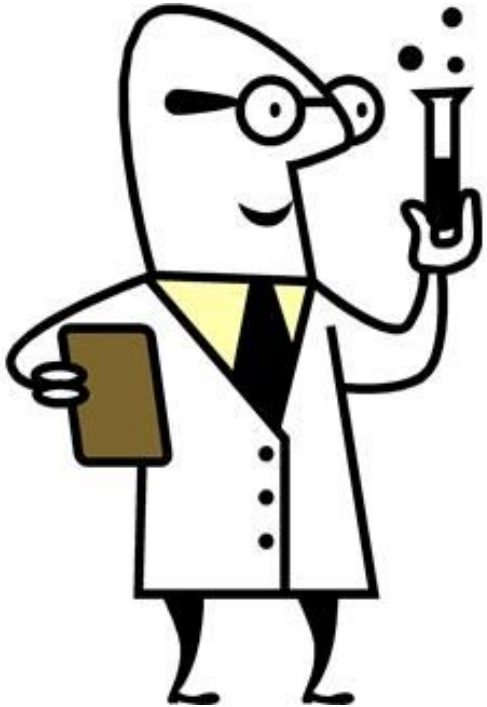
- `kubectl describe pod cilium-fb874`

```
Ready:          True
Restart Count:  2
Liveness:       exec [cilium status] delay=120s timeout=1s period=10s #success=1 #failure=10
Readiness:      exec [cilium status] delay=5s timeout=1s period=5s #success=1 #failure=3
```

Keywords

- `kubectl describe <object>`
- `kubectl logs <pod>`

Lab time!



2.4. Determine the Reason for Pod Failure

- customizing the termination message
  - kubernetes retrieves termination messages from the termination message file specified in the terminationMessagePath field of a Container, which as a default value of /dev/termination-log
  - by customizing this field, you can tell Kubernetes to use a different file
  - kubernetes use the contents from the specified file to populate the Container's status message on both success and failure.
  - in the following example, the container writes termination messages to /tmp/my-log for kubernetes to retrieve
    - kubectl apply -f custom-terminatin-demo.yaml