CKA Curriculum Part 2 - Logging and Monitoring

Understand how to monitor all cluster components

As a refresher, the following components reside in a Kubernetes cluster:

- Master Nodes
  - Etcd (unless external)
  - Kube-APIserver
  - Kube-Scheduler
  - Kube-Controller-Manager
- Worker Nodes
  - Some sort of CNI (Flannel, NSX-T, etc)
  - Kube-Proxy
  - Kubelet
  - Container runtime (Docker, RKT, containerd, etc)

Master Node(s)

ETCD

Usually, most etcd implementations also include etcdctl, which can aid in monitoring the state of the cluster. If you're unsure where to find it, execute the following:

```
find / -name etcdctl
```

Leveraging this tool to check the cluster status:

```
./etcdctl cluster-health

member 17f206fd866fdab2 is healthy: got healthy result from https://master-0.etcd.cfcr.internal:2379https://master-0.etcd.cfcr.internal:2379
```

The cluster this was executed on has only one master node, hence only one result from the script. You will normally receive a response for each etcd member in the cluster.

Alternatively, leverave kubectl get componentstatuses:

```
kubectl get componentstatuses

NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-1               Healthy   {"health":"true"}
etcd-0               Healthy   {"health":"true"}
```

Kube-APIserver

This is slightly dependent on the environment for which the Kubernetes platform has been installed on. For systemd based systems:

```
journalctl -u kube-apiserver
```

Or

```
cat /var/log/kube-apiserver.log
```

Or for instances where Kube-APIserver is running as a static pod:

```
kubectl logs kube-apiserver-k8s-master-03 -n kube-system
```

Kube-Scheduler

For systemd-based systems

```
journalctl -u kube-scheduler
```

Or

```
cat /var/log/kube-scheduler.log
```

Or for instances where Kube-Scheduler is running as a static pod:

```
kubectl logs kube-scheduler-k8s-master-03 -n kube-system
```

Kube-Controller-Manager

For systemd-based systems

```
journalctl -u kube-controller-manager
```

Or

```
cat /var/log/kube-controller-manager.log
```

Or for instances where Kube-controller manager is running as a static pod:

```
kubectl logs kube-controller-manager-k8s-master-03 -n kube-system
```

Worker Node(s)

CNI

Obviously this is dependent on the CNI in use for the cluster you're working on. However, using Flannel as an example:

```
journalctl -u flanneld
```

If running as a pod, however:

```
Kubectl logs --namespace kube-system <POD-ID> -c kube-flannel
kubectl logs --namespace kube-system weave-net-pwjkj -c weave
```

Kube-Proxy

For systemd-based systems

```
journalctl -u kube-proxy
```

Or

```
cat /var/log/kube-proxy.log
```

Kubelet

```
journalctl -u kubelet
```

Or

```
cat /var/log/kubelet.log
```

Container Runtime

Similarly to the CNI, this depends on which container runtime has been deployed, but using Docker as an example:

For systemd-based systems:

```
journalctl -u docker.service
```

Or

```
cat /var/log/docker.log
```

Hint : list the contents of `etc/systemd/system` if it's a systemd-based service (containerd.service may be here)

Understand how to monitor applications

This section is a bit open-ended as it highly depends on what you have deployed and the topology of an application. Typically, however, we have a application that runs as a number of inter-connected **services**, which in the world of Kubernetes is a container. So we monitor our applications by monitoring the pods/services/anything else we have deployed.

Applications are likely to (At least) consist of pods, replication controllers and services.

Monitor Pods

Leverage "kubectl describe pod" to get information pertaining to a specific pod.

```
kubectl describe pod nginx-65899c769f-2pgzk
```

Note at the end there are a list of events:

```
Events:
 Type    Reason                 Age   From                    Message
 ----    ------                 ----  ----                    -------
 Normal  Scheduled              18m   default-scheduler       Successfully assigned nginx-65899c769f-2pgzk to k8s-worker-01
 Normal  SuccessfulMountVolume  18m   kubelet, k8s-worker-01  MountVolume.SetUp succeeded for volume "default-token-rbt5s"
 Normal  Pulling                18m   kubelet, k8s-worker-01  pulling image "nginx"
 Normal  Pulled                 18m   kubelet, k8s-worker-01  Successfully pulled image "nginx"
 Normal  Created                18m   kubelet, k8s-worker-01  Created container
 Normal  Started                18m   kubelet, k8s-worker-01  Started container
```

This should point the user in the right direction should a pod have issues, either pre or post deployment.

We can extract logs that a pod generates providing it does so to stdout and/or stderr

```
kubectl logs nginx-65899c769f-2pgzk
```

Note that is is for pods that are running. For pods that have crashed:

```
kubectl logs --previous nginx-65899c769f-2pgzk
```

Monitor Replication Controllers

Replication controllers are dependent on the successful creation of pods. Therefore, if your pods won't deploy for whatever reason, the replication controller won't work. Therefore, concentrate on the successful deployment of pods.

We can also perform the following to get more information:

```
kubectl describe replicationcontroller
```

Monitor Services

Assuming a service has been successfully created, ensure it's initialised:

```
kubectl get services
```

```
Kubectl describe service nginx-service
```

The describe command is quite useful as it lists, amongst other information, the endpoints (pods) that are participating, and how it determines their inclusion (selector):

```
Selector:       env=test
Type:           ClusterIP
IP:             10.32.0.111
Port:           <unset>  80/TCP
TargetPort:     80/TCP
Endpoints:      10.200.0.3:80
```

Unless explicitly defined, services leverage the default type of ClusterIP, which can only be accessed internally. To test a

Manage application logs

Containers that log to stdout and stderr can have their logs extracted via kubectl:

```
kubectl logs nginx
```

Where “nginx” is the name of a pod. However, we can increase the scope by supplying labels:

```
kubectl logs -l app=nginx --all-containers=true
```

This will return all logs from containers in pods with a defined label “app=nginx”