Logging & Monitoring

Monitor Cluster Components

I'd like to know Node level metrics such as the number of nodes in the cluster, how many of them are healthy as well as performance metrics such as CPU. Memory, network and disk utilization. As well as POD level metrics such as the number of PODs, and performance metrics of each POD such as the CPU and Memory consumption on them. So we need a solution that will monitor these metrics store them and provide analytics around this data. As of this time, **Kubernetes does not come with a full featured built-in monitoring solution**.

However, there are a number of open-source solutions available today, such as the Metrics-Server, Prometheus, Elastic Stack, and proprietary solutions like Datadog and Dynatrace.

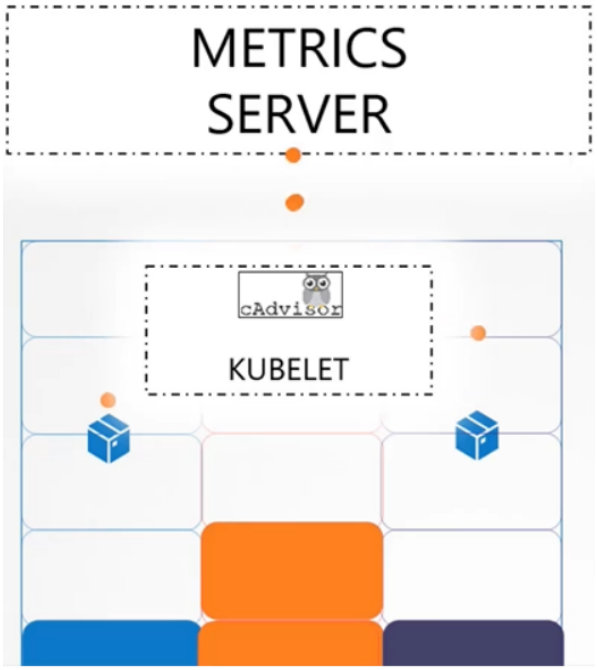
Heapster was one of the original projects that enabled monitoring and analysis features for kubernetes. You will see a lot of reference online when you look for reference architectures on monitoring Kubernetes. However, Heapster is now Deprecated and a slimmed down version was formed known as the **Metrics Server**.

You can have one metrics server per kubernetes cluster. The metrics server receives metrics from each kubernetes node and pods, processes them, and stores them in memory. Note that the metric server is only an in memory monitoring solution and does not store the metrics on the desk and as a result you cannot see historical performance data. For that you must rely on one of the advanced monitoring solutions we talked about earlier in this article.

So how are the metrics generated for the PODs on these nodes?

Kubernetes runs an agent on each node known as the **kubelet**, which is responsible for receiving instructions from the kubernetes API master server and running PODs on the nodes. The kubelet also contains a subcomponent known as as **cAdvisor** or Container Advisor.

cAdvisor - is responsible for retrieving performance metrics from pods, and exposing them through the kubelet API to make the metrics available for the Metrics Server.



- If you are using minikube for your local cluster, run the command:

```
minikube addons enable metrics-server
```

- For all other environments deploy the metrics server by cloning the metrics-server deployment files from the github repository:

```
git clone https://github.com/kodekloudhub/kubernetes-metrics-server.git
```

- And then deploying the required components using the kubectl create command:

```
cd root/kubernetes-metrics-server/
kubectl create -f .
```

This command deploys a set of pods, services and roles to enable metrics server to poll for performance metrics from the nodes in the cluster.

- Once deployed, give the metrics-server some time to collect and process data. Once processed, cluster performance can be viewed by running the command:

```
kubectl top nodeNAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
controlplane   197m         0%     1161Mi          0%
node01         34m          0%     272Mi           0%
```

This provides the CPU and Memory consumption of each of the nodes. As you can see 8% of the CPU on my master node is consumed, which is about 166 milli cores.

- **Identify the node that consumes the most CPU:**

```
kubectl top node --sort-by='cpu' --no-headers | head -1
```

- Use the kubectl top pod command to view performance metrics of pods in kubernetes.

```
kubectl top pod
```

- **Identify the POD that consumes the most Memory:**

```
kubectl top pod --sort-by='memory' --no-headers | head -1
```

- **Identify POD that consumes the least CPU.**

```
kubectl top pod --sort-by='cpu' --no-headers | tail -1
```

Managing Application Logs

If I were to run the docker container in the background, in a detached mode using the –d option, I wouldn't see the logs:

```
docker run -d kodekloud/event-simulator
```

If I wanted to view the logs. I could use the docker logs command followed by the **container ID**. The –f option helps us see the live log trail:

```
docker logs -f ecfgtyj01js
```

Now back to Kubernetes. We create a pod with the same docker image using the pod definition file:

```
apiVersion: v1
kind: Pod
metadata:
  name: event-simulator-pod
spec:
  containers:
  - name: event-simulator
    image: kodekloud/event-simulator
```

Once it’s the pod is running, we can view the logs using the kubectl logs command with the pod name. Use the –f option to stream the logs live just like the docker command:

```
kubectl logs -f event-simulator-pod
```

Now these logs are specific to the container running inside the pod. As we learned before, Kubernetes PODs can have multiple docker containers in them. In this case I modify my pod definition file to include an additional container called image-processor:

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: event-simulator-pod
spec:
  containers:
  - name: event-simulator
    image: kodekloud/event-simulator
  - name: image-processor
    image: some-image-processor
```

If you ran the kubectl logs command now with the pod name, which container's log would it show?

If there are multiple containers within a pod. You must specify the name of the container explicitly in the command. Otherwise it would fail asking you to specify a name. In this case I will specify the name of the first container event-simulator and that prints the relevant log messages:

```
kubectl logs -f even-simulator-pod event-simulator
```

Now, that is the simple logging functionality implemented within Kubernetes. And that is all that an application developer really needs to know to get started with Kubernetes.