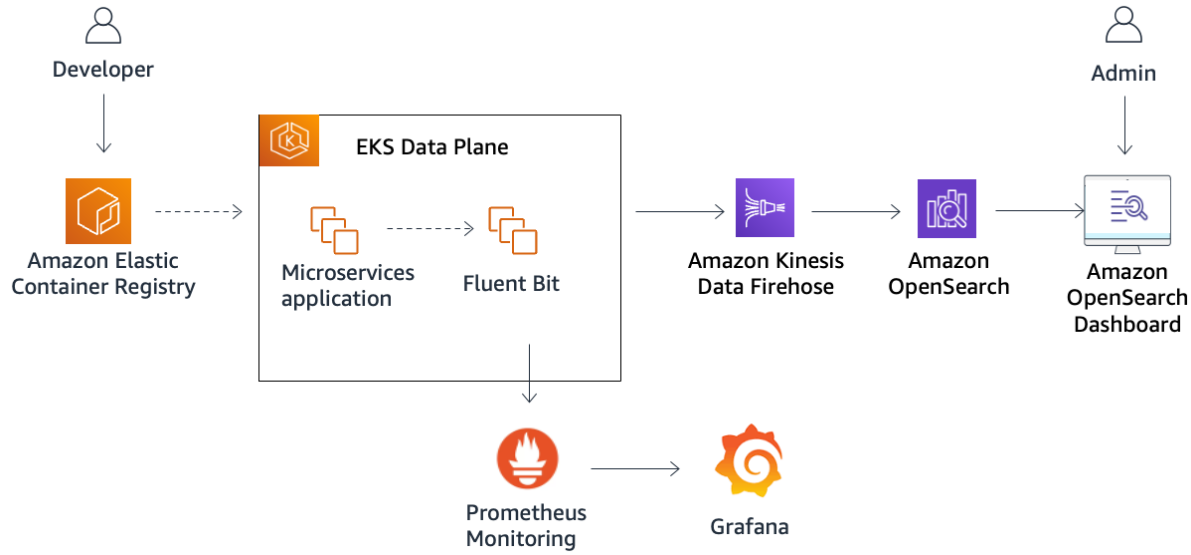
Lab 4: Monitoring Amazon EKS

Objectives:

1. Prometheus and Grafana: Collect and visualize application performance data using Prometheus and Grafana.



Lab only has prometheus and Grafana:--

1. Developer stores container images in ECR. The application is running in EKS clusters in form of microservices.
2. Fluent Bit is deployed to log data from the microservices on each node.
3. FB sends data to Firehose which transports data to Opensearch.
4. Opensearch allows search, store, analyze - data.
5. OS dashboard - Admin: The person who logs into the OpenSearch Dashboard to perform searches and analyze the log data stored in OpenSearch.
6. **Prometheus collects metrics data about the performance and health of the microservices running in the EKS cluster.**
7. **Grafana is used to visualize the metrics collected by Prometheus, allowing the admin to create dashboards for better insight into the application's performance.**

1.1 Connect to bastion host
Go into EC2,select running instances, connect to bastion host2.1

Task 2: Deploy and configure Prometheus

A typical Prometheus installation in Kubernetes includes these components:

1. Prometheus server
2. Node exporter
3. Push gateway
4. Alert manager
5. kube-state-metrics

In Kubernetes, the Prometheus server runs as a pod that is responsible for scraping metrics and saving them to a local time series database.

2.1 create a Prometheus namespace to logically group its monitoring components.

```
sh-4.2$ kubectl create namespace prometheus
namespace/prometheus created
sh-4.2$
```

2.2 To add the prometheus-community chart repository, enter the following command:
This chart installs the core Prometheus server, Alertmanager for alerts, exporters for collecting metrics, and PushGateway to support short-lived jobs.

```
sh-4.2$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

"prometheus-community" has been added to your repositories
sh-4.2$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. ⎈Happy Helming!⎈
```

`https://prometheus-community.github.io/helm-charts`

2.3 To deploy Prometheus, enter the following command:

helm upgrade -i prometheus prometheus-community/prometheus --version 23.1.0 \
  --namespace prometheus \
  --set alertmanager.persistentVolume.storageClass="gp2",server.persistentVolume.storageClass="gp2"

```
Release "prometheus" does not exist. Installing it now.

NAME: prometheus
LAST DEPLOYED: Fri May 31 16:40:47 2024
NAMESPACE: prometheus
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
```

2.4 Save the prometheus server url to a variable

PROMETHEUS_SERVER=http://prometheus-server.prometheus.svc.cluster.local

2.5 Verify all pods in prometheus namespace are running

```
sh-4.2$ kubectl get pods -n prometheus
NAME                                             READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-0                        1/1     Running   0          2m22s
prometheus-kube-state-metrics-6b77bdbb46-9pqdk   1/1     Running   0          2m22s
prometheus-prometheus-node-exporter-8wt86        1/1     Running   0          2m22s
prometheus-prometheus-node-exporter-k5221        1/1     Running   0          2m22s
prometheus-prometheus-node-exporter-r8pdd        1/1     Running   0          2m22s
prometheus-prometheus-pushgateway-f7f8778d7-7kqht   1/1     Running   0          2m22s
prometheus-server-65bc88c7c6-216gh               2/2     Running   0          2m22s
sh-4.2$
```

Task 3: Deploy and configure Grafana

provides you with tools to turn your time-series database (TSDB) data into beautiful graphs and visualizations.

3.1 Create Grafana namespace

```
sh-4.2$ kubectl create namespace grafana
namespace/grafana created
sh-4.2$
```

3.2 To add the Grafana chart repository, enter the following command:

```
sh-4.2$ helm repo add grafana https://grafana.github.io/helm-charts
"grafana" has been added to your repositories
sh-4.2$
```

`https://grafana.github.io/helm-charts`

3.3 view lines 411-420 of the Grafana Helm chart, which defines authentication for the admin user:

```
sh-4.2$ helm show values grafana/grafana > grafana.yaml

sh-4.2$ head -n 429 grafana.yaml | tail -n 10
    runAsUser: 0
    seccompProfile:
      type: RuntimeDefault
    capabilities:
      add:
        - CHOWN

# Administrator credentials when not using an existing secret (see below)
adminUser: admin
# adminPassword: strongpassword
```

3.4 We now establish the password for admin (skipping)

3.5 Now we use Helm to deploy Grafana into our cluster

...

3.6 create a yaml file grafana.yaml:

```
sh-4.2$ mkdir grafana
sh-4.2$ cat << EoF > grafana/grafana.yaml
> datasources:
>   datasources.yaml:
>     apiVersion: 1
>     datasources:
>     - name: Prometheus
>       type: prometheus
>       url: $PROMETHEUS_SERVER
>       access: proxy
>       isDefault: true
> EoF
sh-4.2$
```

This file:

1. Defines a Prometheus data source that Grafana will use to create visualizations
2. Points to the URL you saved to $PROMETHEUS_SERVER, which connects Grafana to Prometheus
3. Makes this Prometheus data source the default, so dashboards will use this data source by default when visualizing metrics

3.7 Verify all pods of grafana namespace

```
sh-4.2$ kubectl get all -n grafana
NAME                              READY   STATUS    RESTARTS   AGE
pod/grafana-7c9d6849f6-txdsx      1/1     Running   0          67s

NAME              TYPE           CLUSTER-IP       EXTERNAL-IP
                 PORT(S)        AGE
service/grafana   LoadBalancer   172.20.151.227   a14b7436af6f54a80828028
.amazonaws.com   80:32340/TCP   67s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/grafana      1/1     1            1           67s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/grafana-7c9d6849f6      1         1         1       67s
sh-4.2$
```

Task complete: You have successfully deployed and configured Grafana.

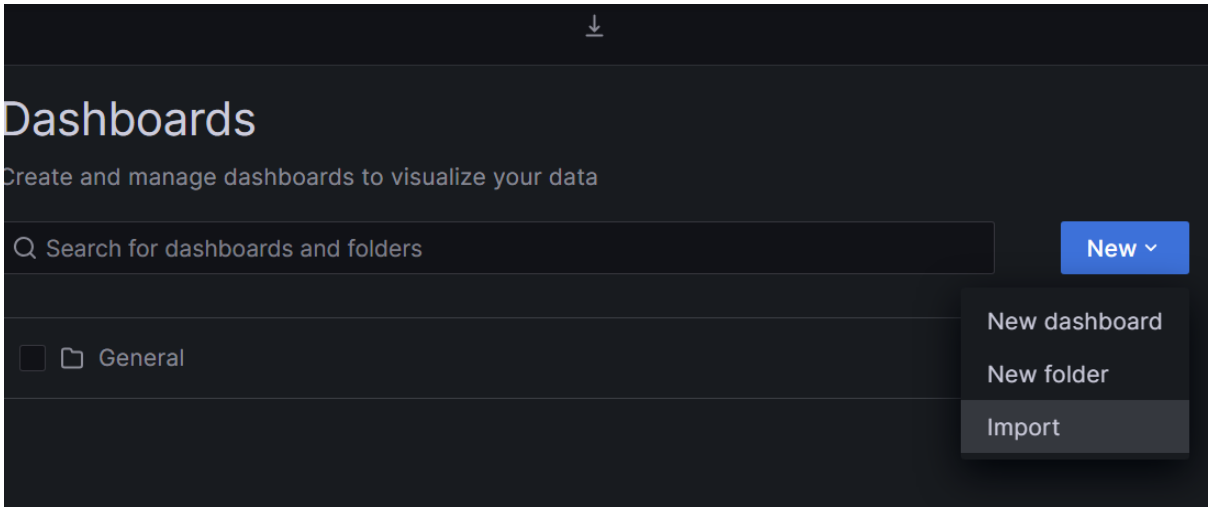
Task 4: Analyze metric data using Grafana

4.1 To retrieve the URL pointing to the Grafana user interface, enter the following command:

```
sh-4.2$ export ELB=$(kubectl get svc -n grafana grafana -o jsonpath='{.status.loadBalancer.ingress[0].hostn
ame}')
sh-4.2$
sh-4.2$ echo "http://$ELB"
http://a14b7436af6f54a80828028423a53460-1763789487.us-west-2.elb.amazonaws.com
sh-4.2$
```

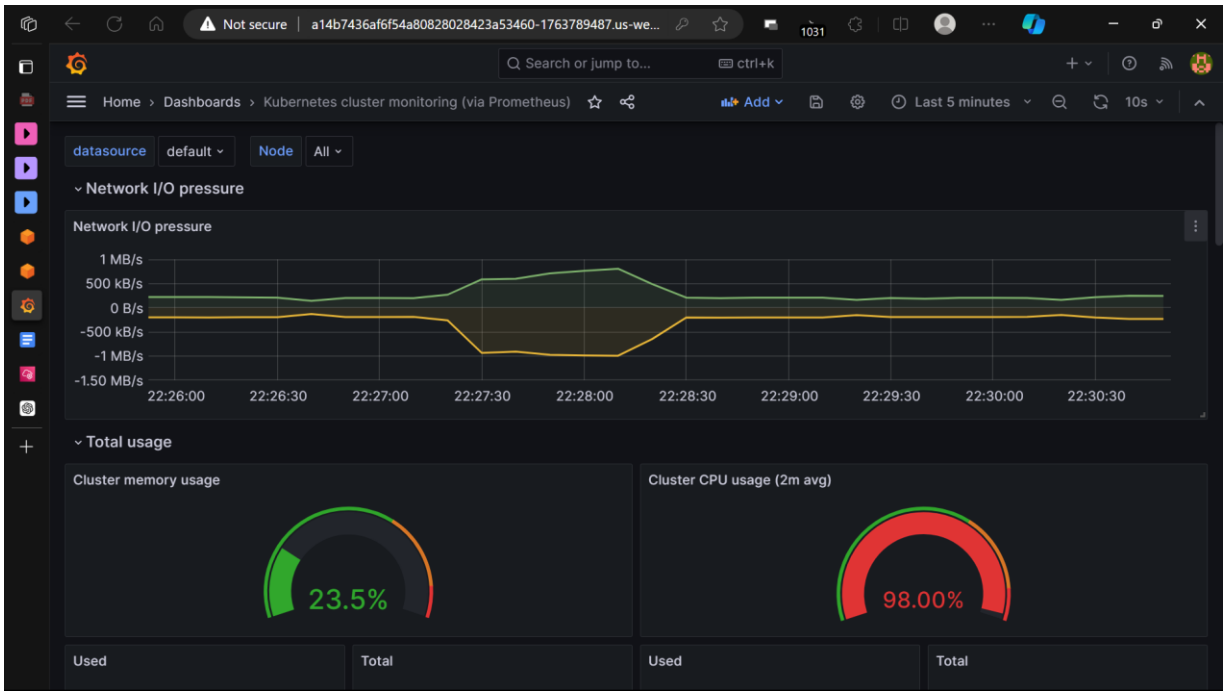
4.2 open the url, login to Grafana using credentials

4.3 Import a new dashboard



4.4 Upload and import the file provided by the lab

4.5 Dashboard -



NOTE - The Grafana dashboard monitors your cluster using Prometheus. It shows overall cluster CPU, memory, and filesystem

usage as well as individual statistics for pods, containers, and control plane as deployed by Kubernetes Operations (kOps).

4.6 The cluster CPU usage is over 90%. This is a cause for concern.
4.7 To review the EKS pods consuming high CPU, enter the following command:

```
sh-4.2$ kubectl top pods -A --sort-by=cpu | head
NAMESPACE       NAME                                       CPU(cores)   MEMORY(bytes)
malfunction     malfunction-658c9646db-tz77b               1952m        0Mi
malfunction     malfunction-658c9646db-s8lzb               944m         0Mi
malfunction     malfunction-658c9646db-f4r2g               685m         0Mi
stars           backend-6f94bb778c-zpvq5                   507m         79Mi
stars           frontend-7c9ccb8787-gc2lq                  505m         55Mi
management-ui   management-ui-7547fb8db9-4lbxt             487m         51Mi
client          client-7fc9b44789-gfr2z                    342m         34Mi
prometheus      prometheus-server-65bc88c7c6-2l6gh         7m           296Mi
grafana         grafana-7c9d6849f6-txdsx                   6m           97Mi
sh-4.2$
```

there are three pods whose names start with malfunction- that are consuming large amounts of CPU.

4.8 To view the assigned CPU limits for your pods, enter the following command:

cat -n /home/ssm-user/scripts/malfunction-cpu.yaml && echo

```
22              resources:
23                limits:
24                  cpu: "150m"
```

m = millicpus

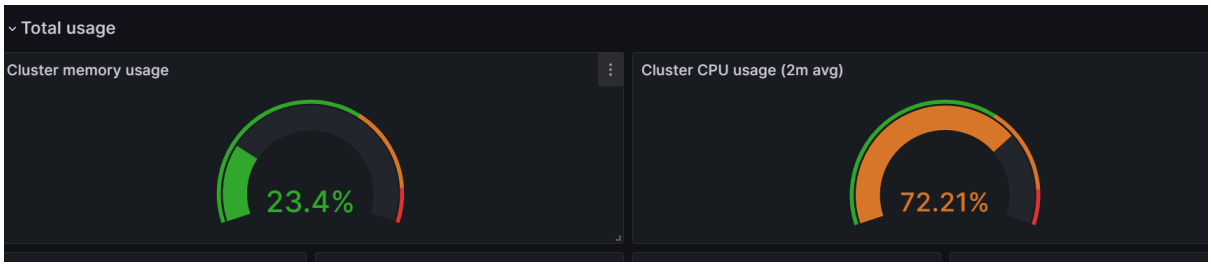
NOTE - the lab says that the user has set this limit of 150m, but I did not set any such limit, maybe it has already been set by the lab

4.9 Apply this manifest file to the pods, and then view the pods again

```
sh-4.2$ kubectl -n malfunction apply -f /home/ssm-user/scripts/malfunction-cpu.yaml
deployment.apps/malfunction configured
sh-4.2$ kubectl top pods -A --sort-by=cpu | head
NAMESPACE       NAME                                       CPU(cores)   MEMORY(bytes)
management-ui   management-ui-7547fb8db9-4lbxt             522m         90Mi
stars           frontend-7c9ccb8787-gc2lq                  504m         74Mi
stars           backend-6f94bb778c-zpvq5                   491m         63Mi
client          client-7fc9b44789-gfr2z                    365m         34Mi
malfunction     malfunction-6f5f6cb6c7-79hhm               150m         0Mi
malfunction     malfunction-6f5f6cb6c7-dfnx8               150m         1Mi
malfunction     malfunction-6f5f6cb6c7-h6nv7               150m         1Mi
prometheus      prometheus-server-65bc88c7c6-2l6gh         9m           208Mi
kube-system     ebs-csi-controller-6594597796-wp9fr        4m           60Mi
sh-4.2$
```

the malfunctioning pods are no longer consuming high amounts of CPU.

4.10 View the dashboard



Task complete: You have successfully analyzed metrics data using Grafana, and resolved the high CPU utilization problem affecting some of your pods.