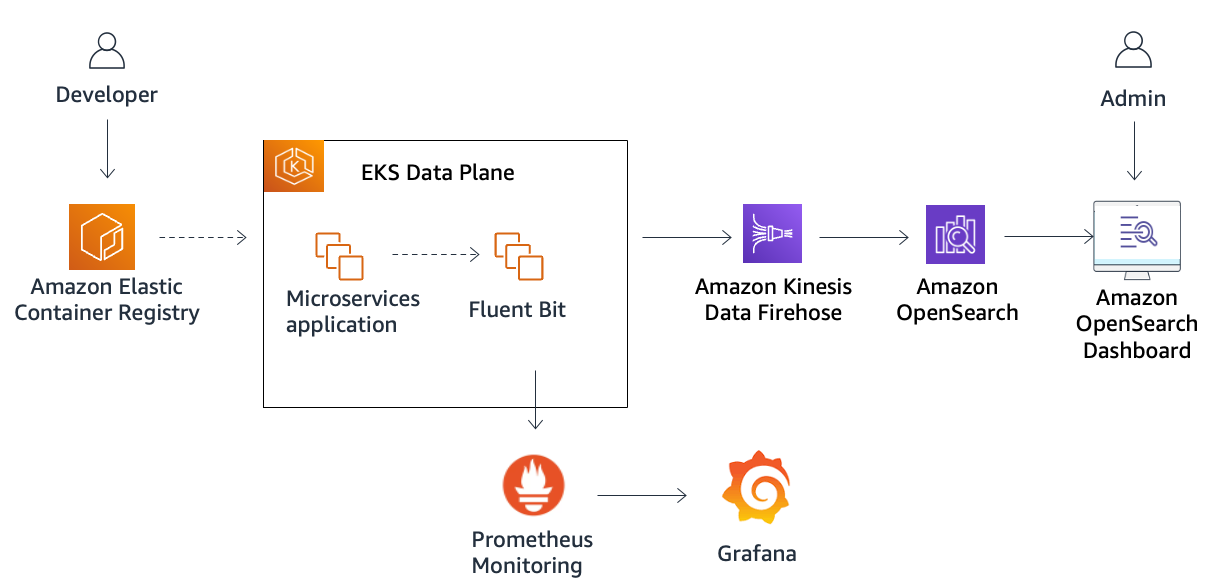
# Lab 4: Monitoring Amazon EKS

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

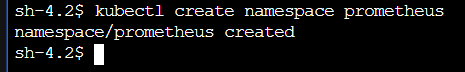
Task 2: Deploy and configure Prometheus

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

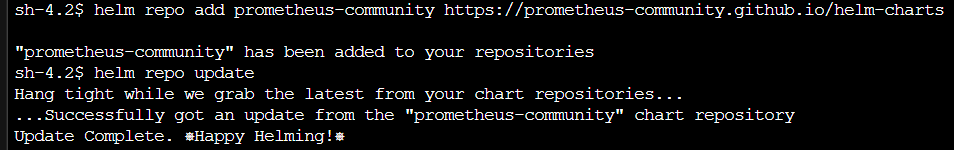
In Kubernetes, the Prometheus server runs as a pod that is responsible for scraping metrics and saving them to a local time series database.

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



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.



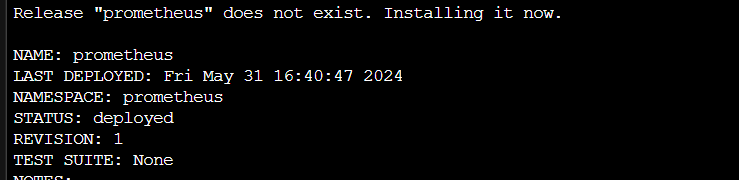
https://prometheus-community.github.io/helm-charts

2.3 To deploy Prometheus, enter the following command:

helm upgrade -i prometheus prometheus-community/prometheus --version 23.1.0 \

--namespace prometheus \

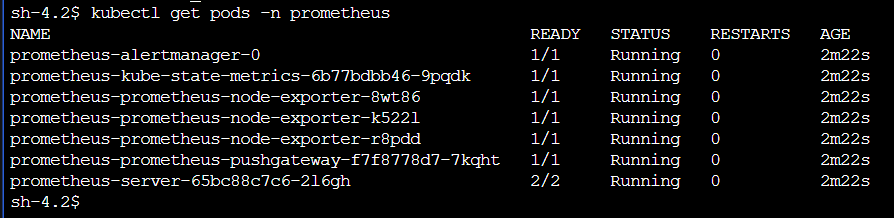
--set alertmanager.persistentVolume.storageClass="gp2",server.persistentVolume.storageClass="gp2"



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



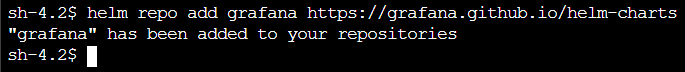
**Task 3: Deploy and configure Grafana**

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

3.1 Create Grafana namespace

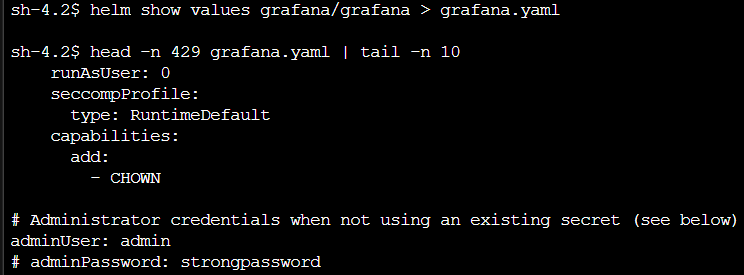


3.2 To add the Grafana chart repository, enter the following command:



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

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

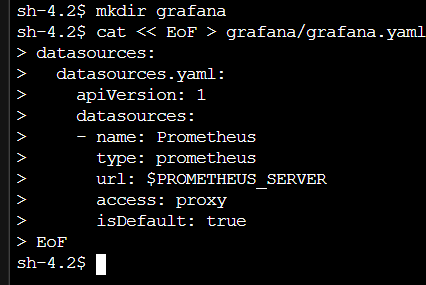


3.4 We now establish the password for admin (skipping)

3.5 Now we use Helm to deploy Grafana into our cluster

…

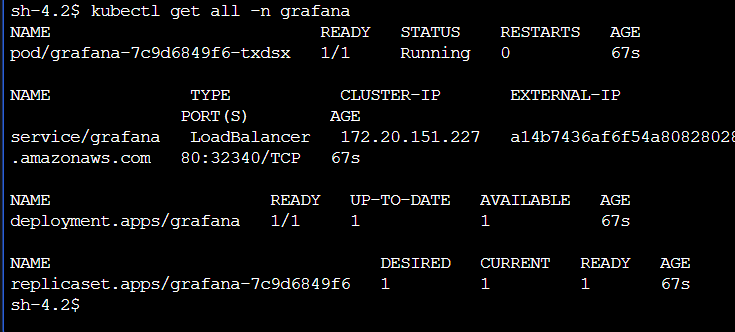
3.6 create a yaml file grafana.yaml:



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

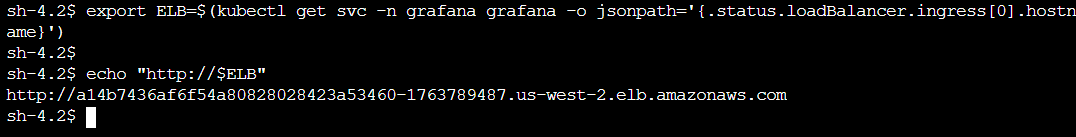
3.7 Verify all pods of grafana namespace





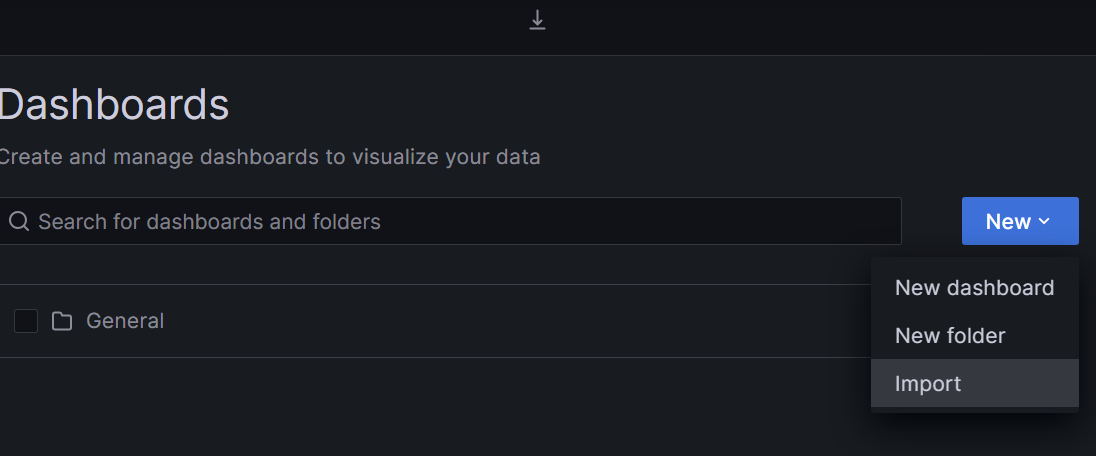
**Task 4: Analyze metric data using Grafana**

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



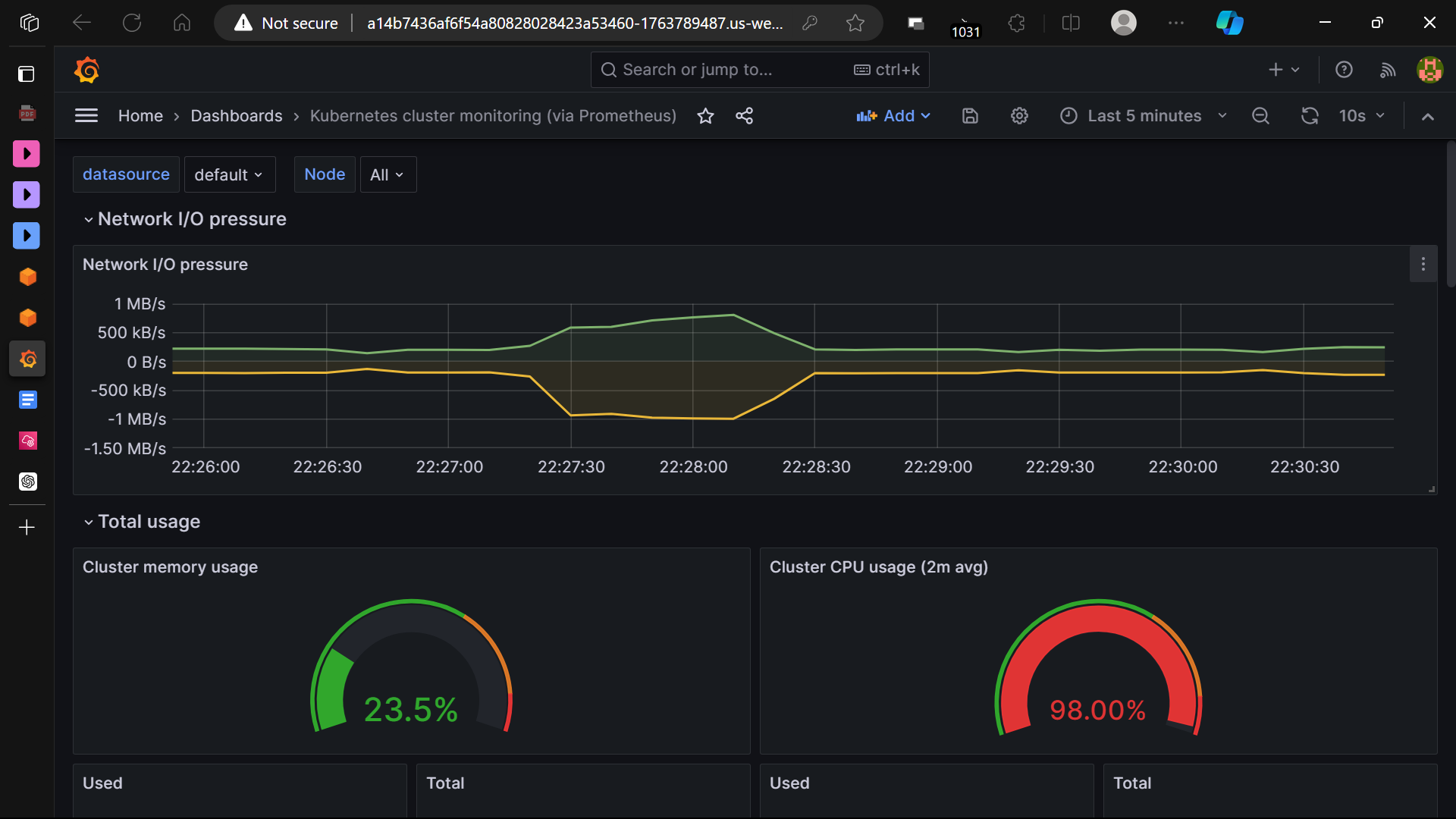
4.2 open the url, login to Grafana using credentials

4.3 Import a new dashboard



4.4 Upload and import the file provided by the lab

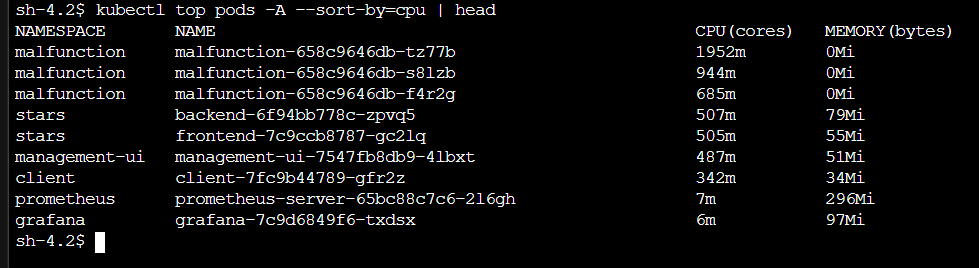
4.5 Dashboard -



**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.

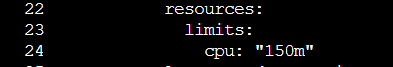
4.7 To review the EKS pods consuming high CPU, enter the following command:



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:

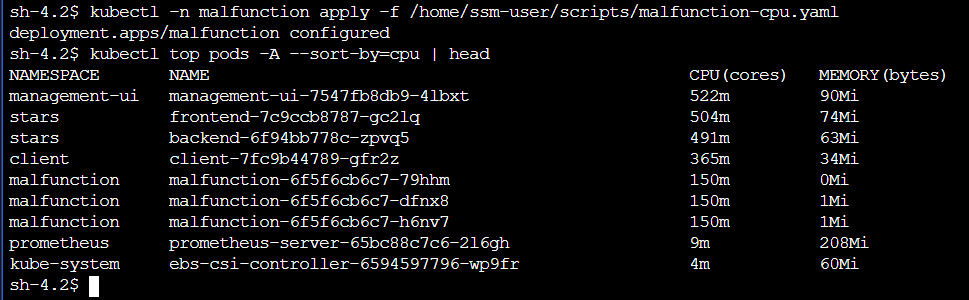
cat -n /home/ssm-user/scripts/malfunction-cpu.yaml && echo



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



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

4.10 View the dashboard



