Observability - A complete Guide: Part 2

Overview:

- What is Metrics?
- What is the difference between Metrics and Monitoring ?
- Examples of Metrics
- What is Prometheus ? Components
- Install Prometheus on EKS
- Login to Grafana
- How metrics are scraped ? What are Exporter? Two commonly used Exporters

What is Metrics ?

Metrics are your systems vital signs which provides a continuous pulse of its health and performance. They are essentially values, that are aggregated over time. Metrics can tell us the availability of a service, free CPU % of virtual machines or nodes, pod status, deployment status. Metrics is a sub part of monitoring.

What is the difference between Metrics and Monitoring ?

Metrics as discussed above are the values or data which is aggregated over time to monitor the performance of the system. While monitoring is process of scraping (pulling) the information from the metrics and representing these metrics in a well readable format using the dashboards. Also it is capable of firing the alerts in case of any issue with the system.

Examples of Metrics

Below are some of the most important metrics:

Infrastructure metrics: These metrics comes under the infrastructure level and they are CPU usage, memory consumption, network bandwidth.

Kubernetes metrics: These are the metrics which is related to the kubernetes cluster such as Pod status, deployment replicas.

Application metrics: These metrics are related to the application which are http_request, error rate, user signups which will be a more application specific.

Once these metrics are collected, either they are feed to the monitoring system, or scraped / pulled from the monitoring system.

What is Prometheus ?

Prometheus is one of the top open source time series database which is designed for real time monitoring and also used for alerting.

Where time series means along with the time, we are storing a key value pair. Now if you want to get these information from Prometheus, you can query http server of Prometheus using **PROMQL** which is Prometheus query language. Then Prometheus will give back these metrics which you requested for. You can also get custom metrics for a kubernetes cluster using a service discovery mechanism.

Key Components of Prometheus

- **Prometheus Server**: This is the core component which scrapes and stores the data.
- **Alertmanager**: Sending alerts on certain thresholds when they are crossed.
- **Exporters**: Services which exposes the metrics.
- **Pushgateway**: Push metrics to Prometheus.
- **Service Discovery**: Automatically detects services.

Install Prometheus on EKS

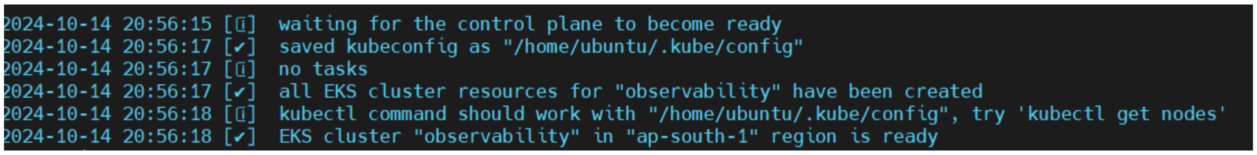
Pre-requisites:

1. AWS Account
2. AWS CLI
3. eksctl
4. kubectl
5. Helm

Steps:

1. Create cluster using the below command

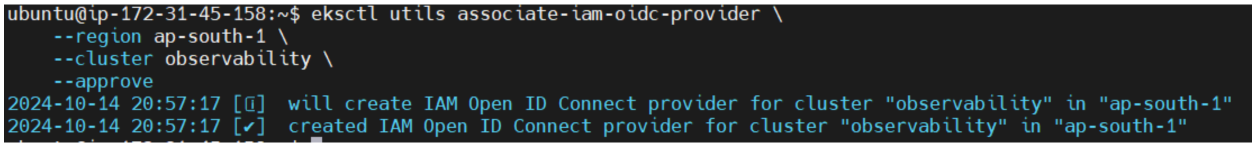
```
eksctl create cluster --name=observability \
                      --region=us-east-1 \
                      --zones=us-east-1a,us-east-1b \
                      --without-nodegroup
```



```
2024-10-14 20:56:15 [ℹ]  waiting for the control plane to become ready
2024-10-14 20:56:17 [✔]  saved kubeconfig as "/home/ubuntu/.kube/config"
2024-10-14 20:56:17 [ℹ]  no tasks
2024-10-14 20:56:17 [✔]  all EKS cluster resources for "observability" have been created
2024-10-14 20:56:18 [ℹ]  kubectl command should work with "/home/ubuntu/.kube/config", try 'kubectl get nodes'
2024-10-14 20:56:18 [✔]  EKS cluster "observability" in "ap-south-1" region is ready
```

2. Create the OIDC provider

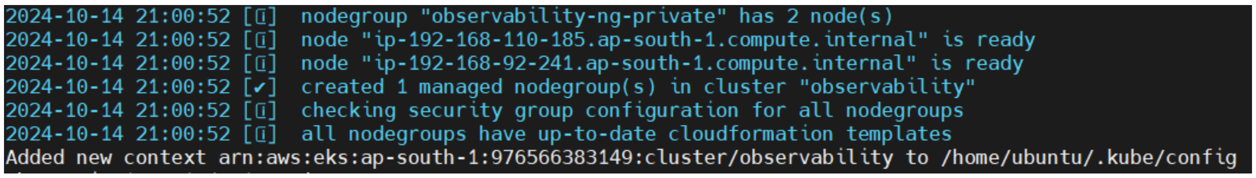
```
eksctl utils associate-iam-oidc-provider \
    --region ap-south-1 \
    --cluster observability \
    --approve
```



```
ubuntu@ip-172-31-45-158:~$ eksctl utils associate-iam-oidc-provider \
    --region ap-south-1 \
    --cluster observability \
    --approve
2024-10-14 20:57:17 [ℹ]  will create IAM Open ID Connect provider for cluster "observability" in "ap-south-1"
2024-10-14 20:57:17 [✔]  created IAM Open ID Connect provider for cluster "observability" in "ap-south-1"
```

3. Create the Node Group for your cluster, and update the context.

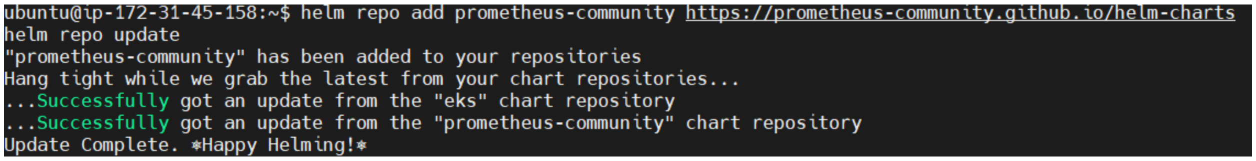
```
eksctl create nodegroup --cluster=observability \
                        --region=ap-south-1 \
                        --name=observability-ng-private \
                        --node-type=t3.medium \
                        --nodes-min=2 \
                        --nodes-max=3 \
                        --node-volume-size=20 \
                        --managed \
                        --asg-access \
                        --external-dns-access \
                        --full-ecr-access \
                        --appmesh-access \
                        --alb-ingress-access \
                        --node-private-networking

# Update ./kube/config file
aws eks update-kubeconfig --name observability
```



```
2024-10-14 21:00:52 [ℹ]  nodegroup "observability-ng-private" has 2 node(s)
2024-10-14 21:00:52 [ℹ]  node "ip-192-168-110-185.ap-south-1.compute.internal" is ready
2024-10-14 21:00:52 [ℹ]  node "ip-192-168-92-241.ap-south-1.compute.internal" is ready
2024-10-14 21:00:52 [✔]  created 1 managed nodegroup(s) in cluster "observability"
2024-10-14 21:00:52 [ℹ]  checking security group configuration for all nodegroups
2024-10-14 21:00:52 [ℹ]  all nodegroups have up-to-date cloudformation templates
Added new context arn:aws:eks:ap-south-1:976566383149:cluster/observability to /home/ubuntu/.kube/config
```

4. Install Kube prometheus stack.

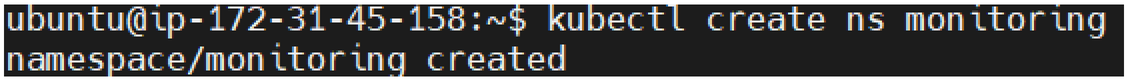
```
helm repo add prometheus-community https://prometheus-community
helm repo update
```



```
ubuntu@ip-172-31-45-158:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
"prometheus-community" has been added to your repositories
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "eks" chart repository
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. *Happy Helming!*
```

5. Create a new namespace.

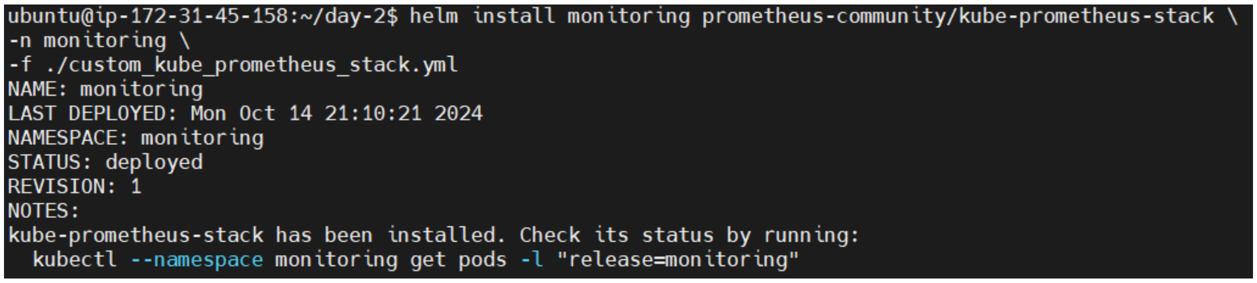
```
kubectl create ns monitoring
```



```
ubuntu@ip-172-31-45-158:~$ kubectl create ns monitoring
namespace/monitoring created
```

6. Install the chart into the monitoring namespace.

```
cd day-2

helm install monitoring prometheus-community/kube-prometheus-sta
-n monitoring \
-f ./custom_kube_prometheus_stack.yml
```



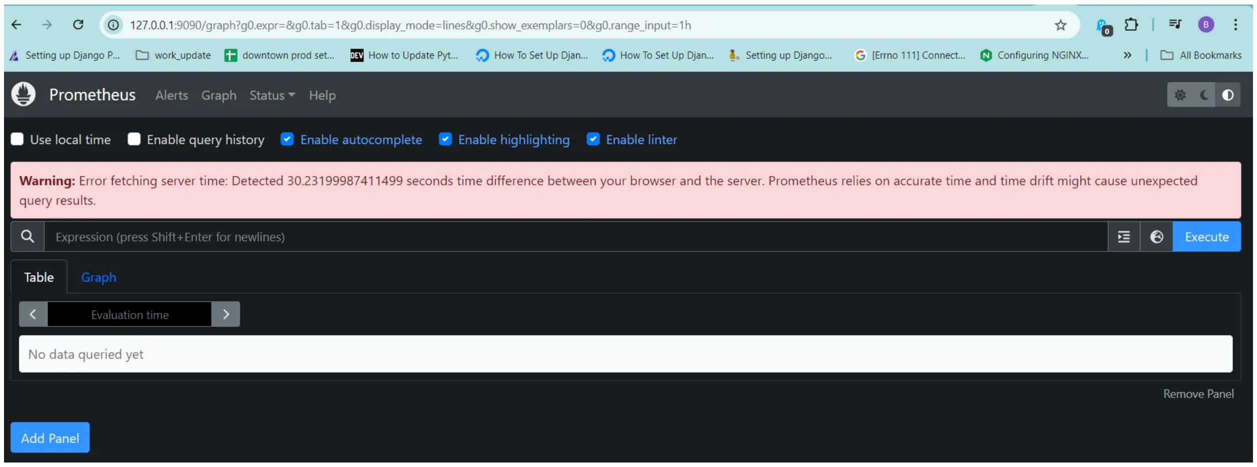
```
ubuntu@ip-172-31-45-158:~/day-2$ helm install monitoring prometheus-community/kube-prometheus-stack \
-n monitoring \
-f ./custom_kube_prometheus_stack.yml
NAME: monitoring
LAST DEPLOYED: Mon Oct 14 21:10:21 2024
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=monitoring"
```

7. Check for all the services if running in the monitoring namespace.

```
kubectl get all -n monitoring
```

8. Forward the port for Prometheus, Grafana and Alertmanager

```
kubectl port-forward service/prometheus-operated -n monitoring 9
```

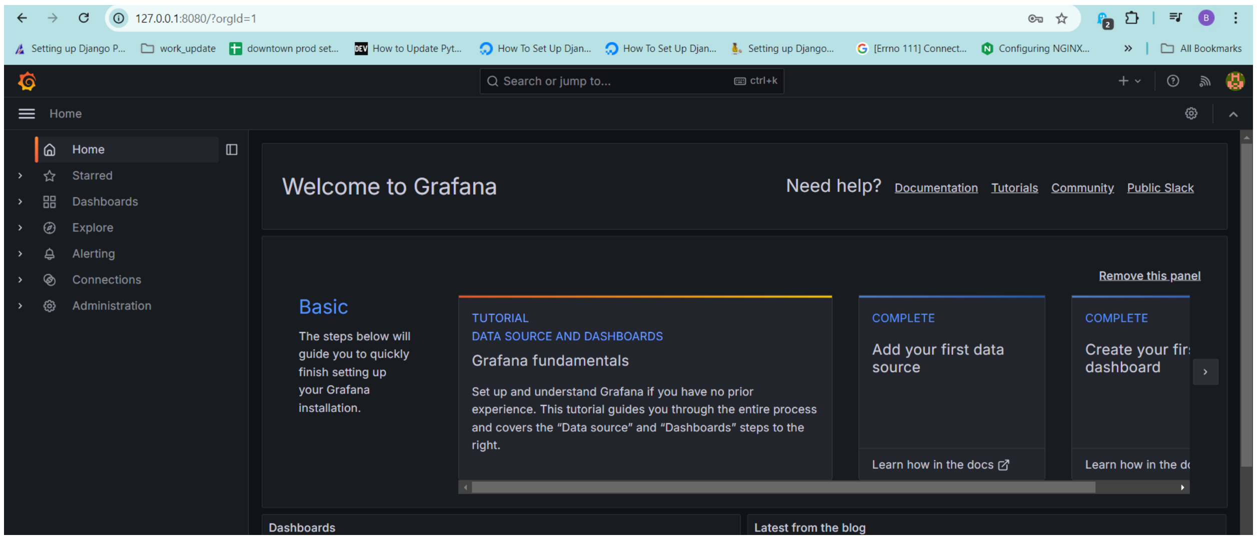


Warning: Error fetching server time: Detected 30.23199987411499 seconds time difference between your browser and the server. Prometheus relies on accurate time and time drift might cause unexpected query results.

We have set up Prometheus successfully. In the same way we will set up Grafana and Alertmanager.

```
kubectl port-forward service/monitoring-grafana -n monitoring 80
```

Login to Grafana: The username for the Grafana server is `admin` **and password is** `prom-operator`.

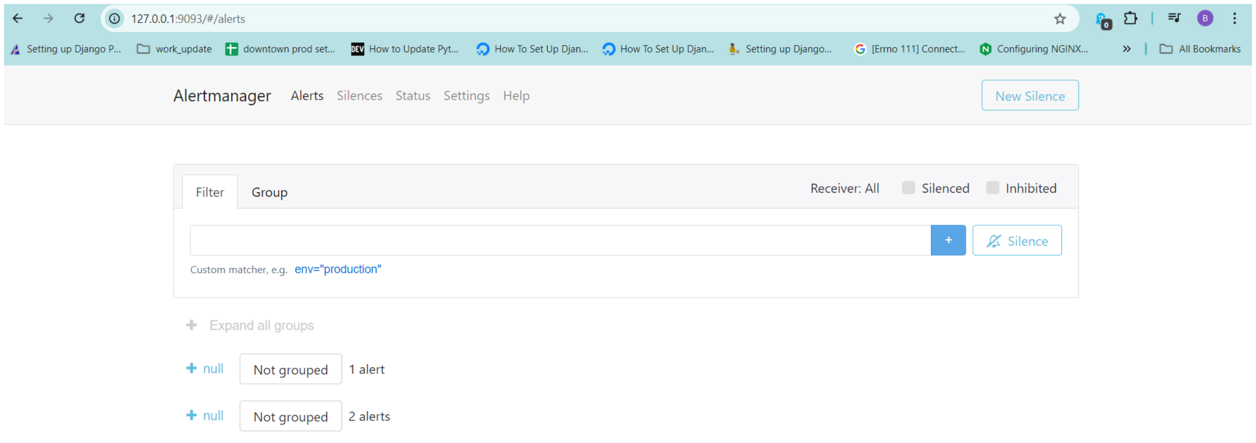


The steps below will guide you to quickly finish setting up your Grafana installation.

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

Alertmanager UI: Using the below command

```
kubectl port-forward service/alertmanager-operated -n monitorin
```



We have successfully installed all the 3 services in our kubernetes cluster. The only difference in production server will be "**ingress controller and ingress resource**", instead of using the port-forward command.

Once done with hands-on demo, clean up the complete infrastructure.

9. Uninstall the Helm chart.

```
helm uninstall monitoring --namespace monitoring
```

10. Delete the namespace.

```
kubectl delete ns monitoring
```

11. Finally delete the cluster.

```
eksctl delete cluster --name observability
```

How does Prometheus scrape metrics ?

How metrics are scraped ?

The first thing Prometheus needs is a target, target is the endpoint that supply the metrics that Prometheus stores. Once Prometheus has the endpoints, it can begin to retrieve the metrics from them. To scrape the metrics, Prometheus typically uses the simple HTTP request.

What are Exporters ?

Exporters are small, built-in programs or plugins which stands between prometheus and anything which you want to monitor that natively doesn't support prometheus. It acts as a translator between Prometheus and endpoint you want to monitor.

Two most commonly used Exporters

- **Node Exporters**: Node Exporters provide hardware and OS level system metrics exposed by UNIX Kernels. Some of the metrics are CPU (CPU Load), Memory (RAM Used, RAM Free) and Disk (Disk Space)
- **Kube State Metrics**: It is a service that generates metrics about the state of kubernetes objects through the Kubernetes API. It Exposes some of the important information such as the Node Status, Available Deployment Replicas, Pod Status.

Conclusion:

Monitoring is important. It helps identify when things have gone wrong, and it can show when things are going right. Prometheus is a powerful Open Source metrics tool. In this article we saw how to install Prometheus, Grafana and Alertmanager. We also saw how the metrics are scraped and information about the exporters. In the next article, we will deep dive into the Grafana and see how we can set the Prometheus as data source and create a dashboard out of it.

Stay Tuned!

If you found this post useful, give it a like👍

Repost♻️

Follow @Bala Vignesh for more such posts 💯🚀