What is Grafana: Grafana is a tool whose purpose is to compile and visualize data through dashboards from the data sources available throughout an organization. From these dashboards it handles a basic alerting functionality that generates visual alarms.

Prometheus: Prometheus is an open-source event monitoring and alerting tool.

it collects the data (it monitors resources) from different sources & passes to Grafana

Helm: Helm is a package manager tool in kubernetes.

1. **Imp installation & source code links:**

Helm official link: https://helm.sh/docs/intro/install/

Github Link for kube-promethus-stack: https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack

Github Link for prometheus-operator: https://github.com/prometheus-operator/prometheus-operator

Install helm on ubuntu: -

```
curl https://baltocdn.com/helm/signing.asc | sudo apt-key add -
sudo apt-get install apt-transport-https --yes
echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee
/etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

type helm to verify helm version: -

```
helm
helm version
```

2. Creating Monitor Namespace in k8s

```
kubectl create namespace prometheus
```

3. Installation of Prometheus, we can use the Helm chart of Prometheus to deploy Prometheus Grafana and many services that have been used to monitor kubernetes clusters.

3.1 Get and add helm repos

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

```
helm search repo prometheus-community

helm repo add stable https://charts.helm.sh/stable
helm repo update
```

3.2 Install helm chart

syntax: helm install [RELEASE_NAME] prometheus-community/kube-prometheus-stack

example: helm install prometheus prometheus-community/kube-prometheus-stack --namespace prometheus

```
kubectl get pods -n prometheus
```

3.3 Now we have installed Prometheus on the cluster we can visit Prometheus dashboard by the following command

example: kubectl port-forward -n prometheus prometheus-kube-prometheus-operator-6f76b6bfc5-ghb6g 9090

4. Configure Grafana Dashboard

```
kubectl get pods -n prometheus
kubectl port-forward -n prometheus prometheus-grafana-<pod name> 3000
kubectl get svc -n prometheus
```

Edit prometheus-grafana service and replace ClusterIP with NodePort

```
kubectl edit svc prometheus-grafana -n prometheus
kubectl get svc -n prometheus
```

Copy NodePort port number and access Grafana Dashboard from browser:

<VM IP>:<NodePort>

to get Password/Secret of Grafana:-

```
kubectl get secret --namespace prometheus
kubectl get secret --namespace prometheus prometheus-grafana -o yaml
```

Copy User Id Password and paste in browser.

(Default User ID and Password)

admin

prom-operator

5. Get The List Of Metrics To Monitor

When we deployed the Prometheus operator chart using Helm, and it includes not just Prometheus but these also deployed:

prometheus-operator

prometheus

alertmanager

node-exporter

kube-state-metrics

grafana

service monitors to scrape internal kubernetes components

kube-apiserver

kube-scheduler

kube-controller-manager

etcd

kube-dns/coredns

Kube-proxy

Tools included like Kube-state-metrics scrape internal system metrics and we can get a list of that by port forwarding the Kube state metrics pod.

```
kubectl get pod -n prometheus
syntax:
kubectl port-forward -n prometheus prometheus-kube-state-metrics-<pod name> 8080

ex:
kubectl port-forward -n prometheus prometheus-kube-state-metrics-95d956569-b7nrc
8080
```

Replace ClusterIp with NodePort:

```
kubectl edit svc -n prometheus prometheus-kube-state-metrics
kubectl get svc -n prometheus
```

Access from Brower:

```
<VM IP>:<NodePort of prometheus prometheus-kube-state-metrics  >/metrics
```

6. Metrics to Watch In Production

Now we do have all integration in place and we are ready to monitor the cluster. As we have some default dashboard available. We start by going to dashboard > manage and you will get the dashboard list

http://<Node IP>:<Grafana NodePort>dashboards

Watch for Nodes

You can monitor each node resource consumption graph separately so you will get the idea of each node performance.

You can import readymade dashboard

Grafana -> Dashboard -> import -> provide ID or URL

You can copy the readymade dashboard ID on the link below. (ex: 15661)

https://grafana.com/grafana/dashboards/?search=pod

history commands just for reference:

2 kubectl create namespace prometheus

3 kubectl get ns

4 helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

5 helm search repo prometheus-community

6 helm repo add stable https://charts.helm.sh/stable

7 helm repo update

8 helm install prometheus prometheus-community/kube-prometheus-stack --namespace prometheus

9 kubectl get pod -n prometheus

10 kubectl describe pod prometheus-prometheus-node-exporter-wbmnx -n prometheus

11 kubectl get pod -n prometheus

12 kubectl port-forward -n prometheus prometheus-kube-prometheus-operator-566c5c5dc7-mlwl4

13 kubectl port-forward -n prometheus prometheus-kube-prometheus-operator-566c5c5dc7-mlwl4 9090

14 kubectl get pods -n prometheus

15 kubectl port-forward -n prometheus prometheus-grafana-6f77bc5bc9-89jf6 3000

16 kubectl get svc -n prometheus

17 kubectl edit service prometheus-grafana -n prometheus

18 kubectl get svc -n prometheus

19 history

20 kubectl get pod -n prometheus

21 kubectl port-forward -n prometheus prometheus-kube-state-metrics-6cfd96f4c8-xj6kz 8080

22 kubectl edit svc -n prometheus prometheus-kube-state-metrics

23 kubectl get svc -n prometheus

Pod Troubleshooting based on alerts: -

Debugging Pods:

```
kubectl describe pods ${POD_NAME}
```

1. **Pod stays pending:** If a Pod is stuck in Pending it means that it cannot be scheduled onto a node. Generally, this is because there are insufficient resources of one type or another that prevent scheduling.

2. **Pod stays waiting**: if a Pod is stuck in the Waiting state, then it has been scheduled to a worker node, but it can't run on that machine. Again, the information from

kubectl describe ... should be informative. The most common cause of Waiting pods is a failure to pull the image.

3. **Pod is crashing** or otherwise unhealthy: -

4. **Pod is running but not doing what I told it to do:** If your pod is not behaving as you expected, it may be that there was an error in your pod description (e.g. mypod.yaml file on your local machine), and that the error was silently ignored when you created the pod. Often a section of the pod description is nested incorrectly, or a key name is typed incorrectly, and so the key is ignored. For example, if you misspelled command as command then the pod will be created but will not use the command line you intended it to use.

The first thing to do is to delete your pod and try creating it again with the --validate option. For example, run kubectl apply --validate -f mypod.yaml

5. **Debugging Replication Controllers**

Replication controllers are fairly straightforward. They can either create Pods or they can't. You can also use kubectl describe rc ${CONTROLLER_NAME} to introspect events related to the replication controller

6. **Debugging Services**: Services provide load balancing across a set of pods. There are several common problems that can make Services not work properly. The following instructions should help debug Service problems.

First, verify that there are endpoints for the service. For every Service object, the apiserver makes an endpoints resource available.

You can view this resource with:

```
kubectl get endpoints ${SERVICE_NAME}
```

Make sure that the endpoints match up with the number of pods that you expect to be members of your service. For example, if your Service is for an nginx container with 3 replicas, you would expect to see three different IP addresses in the Service's endpoints.

7. **Service is missing endpoints:** If you are missing endpoints, try listing pods using the labels that Service uses. Imagine that you have a Service where the labels are:

```
---
spec:
  - selector:
    name: nginx
    type: frontend
```

You can use:

kubectl get pods --selector=name=nginx,type=frontend

to list pods that match this selector. Verify that the list matches the Pods that you expect to provide your Service.

If the list of pods matches expectations, but your endpoints are still empty, it's possible that you don't have the right ports exposed. If your service has a containerPort specified, but the Pods that are selected don't have that port listed, then they won't be added to the endpoints list.

Verify that the pod's containerPort matches up with the Service's targetPort

8. **Network traffic is not forwarded**

If you can connect to the service, but the connection is immediately dropped, and there are endpoints in the endpoints list, it's likely that the proxy can't contact your pods.

There are three things to check:

Are your pods working correctly? Look for restart count, and debug pods.

Can you connect to your pods directly? Get the IP address for the Pod, and try to connect directly to that IP.

Is your application serving on the port that you configured? Kubernetes doesn't do port remapping, so if your application serves on 8080, the containerPort field needs to be 8080.

Ref Links: -

https://www.magalix.com/blog/monitoring-of-kubernetes-cluster-through-prometheus-and-grafana

https://logz.io/blog/prometheus-monitoring/#:-:text=The%20combination%20of%20Prometheus%20and,interface%20for%20analysis%20and%20visualization

https://www.sumologic.com/blog/prometheus-vs-grafana/

https://kubernetes.io/docs/tasks/debug-application-cluster/debug-application/

https://kubernetes.io/docs/tasks/debug-application-cluster/determine-reason-pod-failure/

https://docs.openshift.com/container-platform/4.5/support/troubleshooting/investigating-pod-issues.html