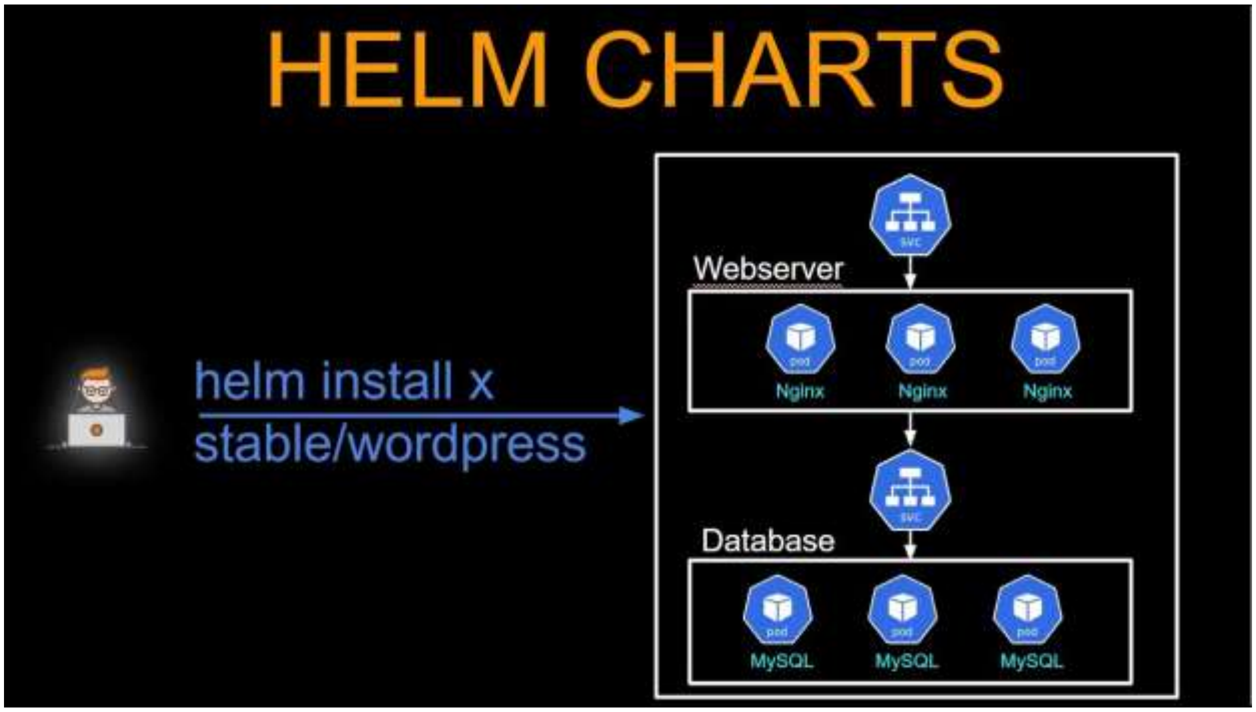
AmazingDevOps

Install Prometheus and Grafana Stack on GKE Cluster Using Helm

How to setup monitoring on Kubernetes Cluster using Prometheus and Grafana | Setup monitoring on EKS Cluster using Prometheus and Grafana



What is helm?

Imagine you are planning a big party, and you need to buy lots of different things, like decorations, food, and drinks. It can be hard to keep track of everything you need to buy, and make sure you get the right things in the right amounts.

In a similar way, deploying applications in Kubernetes can involve lots of different components, like pods, services, and volumes. It can be hard to manage all these components and ensure they are configured correctly.

This is where Helm comes in. Helm is like a party planning tool that helps you manage all of the different components you need to deploy your applications in Kubernetes. With Helm, you can define your application as a package, called a chart, which includes all of the components and configurations needed to run your application.

Once you have your chart defined, you can use Helm to install and manage your application on a Kubernetes cluster. Helm will take care of all of the details of deploying your application, including creating the necessary resources like pods and services, and setting the correct configurations.

Helm also includes a package manager, called the Helm Hub, where you can find and download pre-packaged charts for popular applications and services, like databases and web servers. This makes it easy to get started with deploying these applications in Kubernetes, without having to create your own charts from scratch.

Overall, Helm provides a way to simplify the deployment of applications in Kubernetes, by providing a package manager and tools to manage your application components and configurations.

What is helm Chart?

A Helm chart is like a party planning blueprint for deploying applications in Kubernetes. It's a package that includes all the different components and configurations needed to deploy your application, like pods, services, and volumes.

When you create a Helm chart, you define all the different pieces that make up your application, and how they should be configured. This includes things like which containers to use, how many replicas of each container should be running, and which ports should be exposed.

Once you have your Helm chart defined, you can use it to deploy your application in Kubernetes. Helm will take care of creating all the necessary resources in Kubernetes, based on the definitions in your chart. This makes it easy to deploy complex applications in Kubernetes, without having to manually create all the resources yourself.

Helm charts can also be shared with other people, just like sharing a party planning checklist. This means that other people can easily deploy the same application using your Helm chart, without having to figure out all the details themselves.

Overall, Helm charts provide a way to package up and deploy applications in Kubernetes, using a standardized blueprint that includes all the necessary resources and configurations.

<u>**Key components:**</u>

1. Prometheus server - Processes and stores metrics data

2. Alert Manager - Sends alerts to any systems/channels

3. Grafana - Visualize scraped data in UI

Installation Method:

There are many ways you can set up Prometheus and Grafana. You can install in following ways:

1. Create all configuration files of both Prometheus and Grafana and execute them in right order.

2. Prometheus Operator - to simplify and automate the configuration and management of the Prometheus monitoring stack running on a Kubernetes cluster

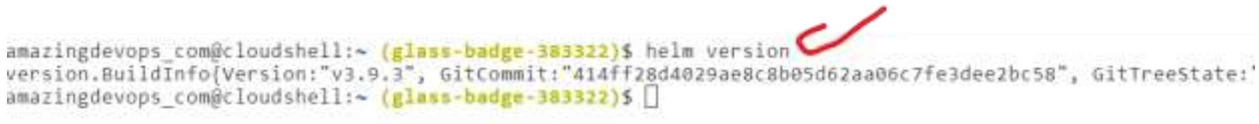
3. Helm chart (Recommended) - Using helm to install Prometheus Operator including Grafana

Why to use Helm?

Helm is a package manager for Kubernetes. Helm simplifies the installation of all components in one command. GKE already has a helm installed.

Check the version of your helm package.

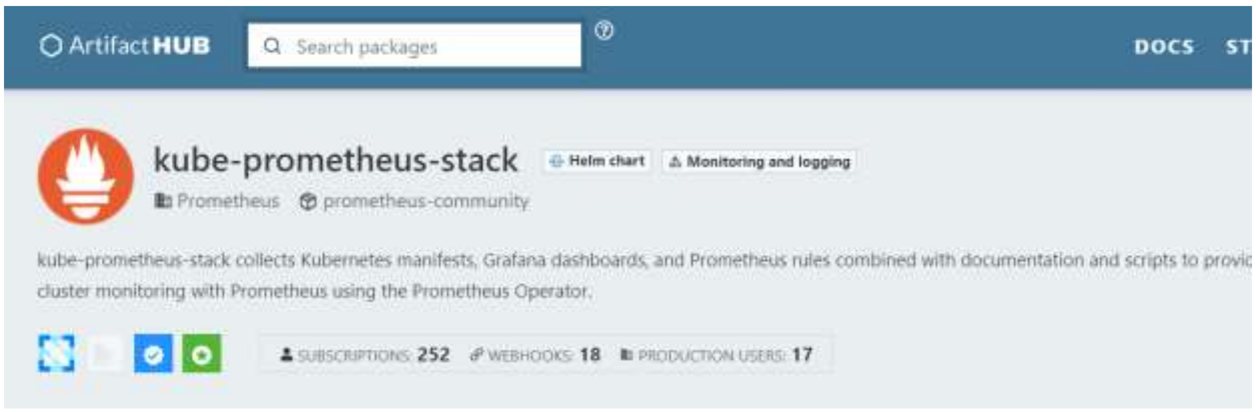
# helm version



```
amazingdevops_com@cloudshell:~ (glass-badge-383322)$ helm version
version.BuildInfo{Version:"v3.9.3", GitCommit:"414ff28d4029ae8c8b05d62aa06c7fe3dee2bc58", GitTreeState:'
amazingdevops_com@cloudshell:~ (glass-badge-383322)$
```

Click on the below helm artifact.

https://artifacthub.io/packages/helm/prometheus-community/kube-prometheus-stack



kube-prometheus-stack collects Kubernetes manifests, Grafana dashboards, and Prometheus rules combined with documentation and scripts to provic
cluster monitoring with Prometheus using the Prometheus Operator.

SUBSCRIPTIONS: 252 WEBHOOKS: 18 PRODUCTION USERS: 17

Overall, the above link provides an easy way to find and install a monitoring solution for Kubernetes using a pre-packaged Helm chart, which can save time and effort compared to deploying and configuring each component individually.

The two to be installed are Prometheus and Grafana.

Implementation steps

We need to add the Helm Stable Charts for your local client. Execute the below command:

# Commands

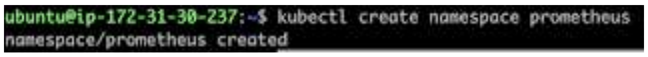
Get Helm Repository Info

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

# helm repo update

Create Prometheus namespace.

# kubectl create namespace prometheus

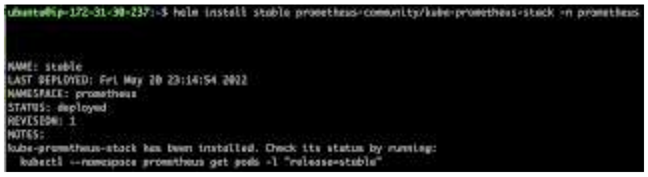


```
ubuntu@ip-172-31-30-237:~$ kubectl create namespace prometheus
namespace/prometheus created
```

Install kube-prometheus-stack

Below is helm command to install kube-prometheus-stack. The helm repo kube-stack-prometheus (formerly prometheus-operator) comes with a grafana deployment embedded.

# helm install stable prometheus-community/kube-prometheus-stack -n prometheus

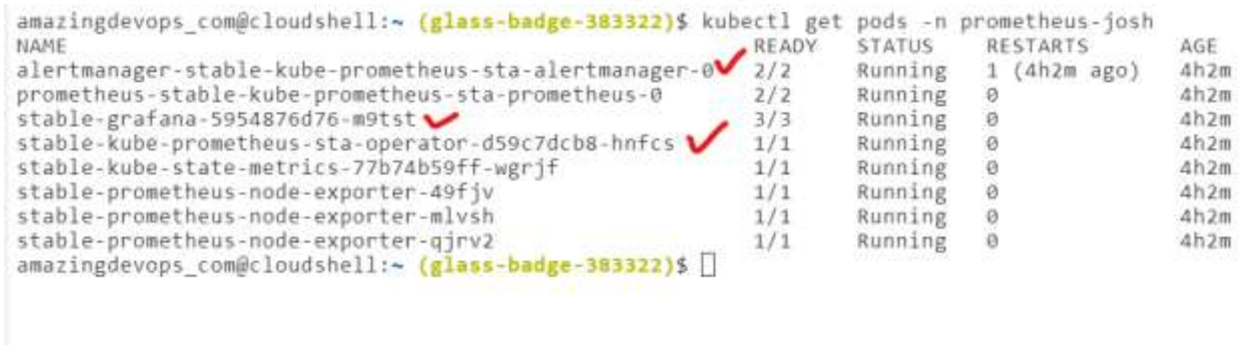


```
ubuntu@ip-172-31-30-237:~$ helm install stable prometheus-community/kube-prometheus-stack -n prometheus


NAME: stable
LAST DEPLOYED: Fri May 20 23:14:54 2022
NAMESPACE: prometheus
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace prometheus get pods -l "release=stable"
```

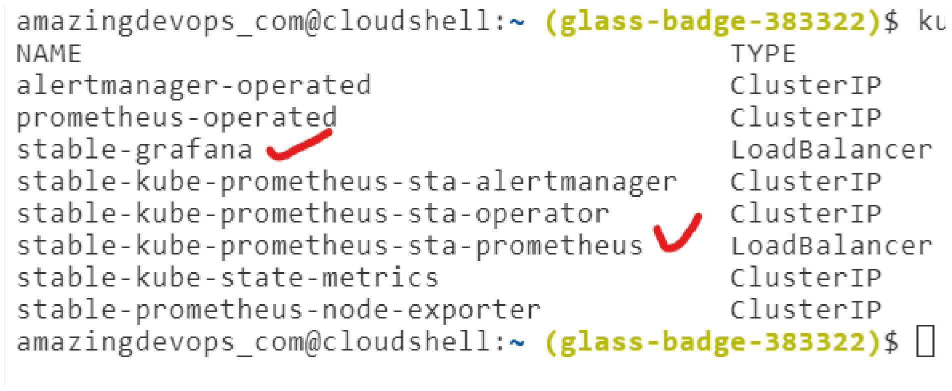
Lets check if prometheus and grafana pods are running already

# kubectl get pods -n prometheus



```
amazingdevops_com@cloudshell:~ (glass-badge-383322)$ kubectl get pods -n prometheus-josh
NAME                                                    READY   STATUS    RESTARTS      AGE
alertmanager-stable-kube-prometheus-sta-alertmanager-0  2/2     Running   1 (4h2m ago)  4h2m
prometheus-stable-kube-prometheus-sta-prometheus-0      2/2     Running   0             4h2m
stable-grafana-5954876d76-m9tst                         3/3     Running   0             4h2m
stable-kube-prometheus-sta-operator-d59c7dcb8-hnfcs     1/1     Running   0             4h2m
stable-kube-state-metrics-77b74b59ff-wgrjf              1/1     Running   0             4h2m
stable-prometheus-node-exporter-49fjv                   1/1     Running   0             4h2m
stable-prometheus-node-exporter-mlvsh                   1/1     Running   0             4h2m
stable-prometheus-node-exporter-qjrv2                   1/1     Running   0             4h2m
amazingdevops_com@cloudshell:~ (glass-badge-383322)$
```

Kubernetes Service or svc



```
amazingdevops_com@cloudshell:~ (glass-badge-383322)$ ku
NAME                                     TYPE
alertmanager-operated                    ClusterIP
prometheus-operated                      ClusterIP
stable-grafana                           LoadBalancer
stable-kube-prometheus-sta-alertmanager  ClusterIP
stable-kube-prometheus-sta-operator      ClusterIP
stable-kube-prometheus-sta-prometheus    LoadBalancer
stable-kube-state-metrics                ClusterIP
stable-prometheus-node-exporter          ClusterIP
amazingdevops_com@cloudshell:~ (glass-badge-383322)$
```

In Kubernetes, a Service is an abstraction layer that provides a stable IP address and DNS name for a set of one or more pods. A pod is a group of one or more containers that are deployed together on a single host.

When you create a Service in Kubernetes, you can specify the type of service you want to create, such as a ClusterIP, NodePort, or LoadBalancer service. Each type of service provides different levels of access and availability for your application.

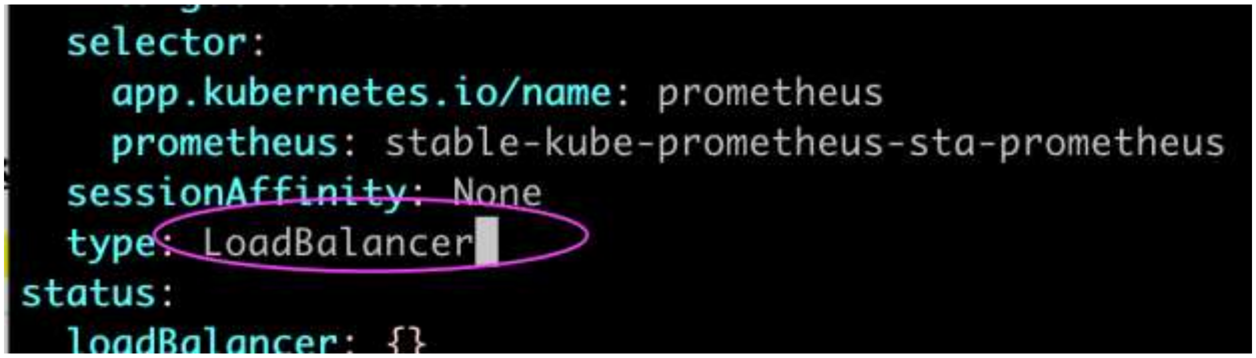
A ClusterIP service, for example, provides access to the pods within the cluster, but not to external traffic. A NodePort service, on the other hand, exposes the service on a specific port on each node in the cluster, making it accessible from outside the cluster. A LoadBalancer service provides a public IP address and DNS name for the service, and automatically routes traffic to the appropriate pods.

By creating a Service, you can decouple the logical name of your application from the specific IP addresses and ports used by the pods that make up your application. This makes it easier to manage your application and ensure that it remains available, even if pods are added or removed.

Overall, a Kubernetes Service provides a way to expose and access a set of pods running in a Kubernetes cluster, using a stable IP address and DNS name. This allows your application to be easily accessed and managed, even as it scales up or down over time.

Edit Prometheus Service

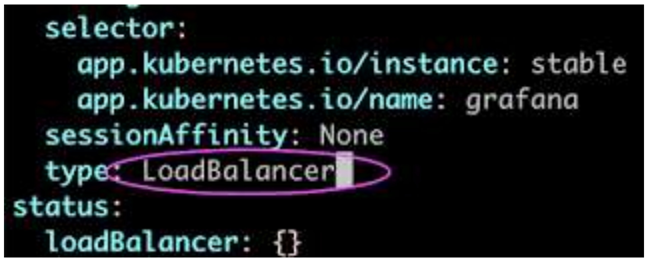
# kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus



```
  selector:
    app.kubernetes.io/name: prometheus
    prometheus: stable-kube-prometheus-sta-prometheus
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

<u>Edit Grafana Service</u>

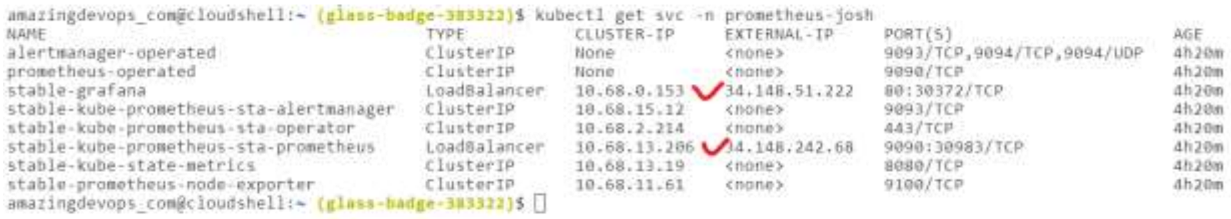
kubectl edit svc stable-grafana -n prometheus



```
  selector:
    app.kubernetes.io/instance: stable
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

Verify if service is changed to LoadBalancer and also to get the Load Balancer URL.

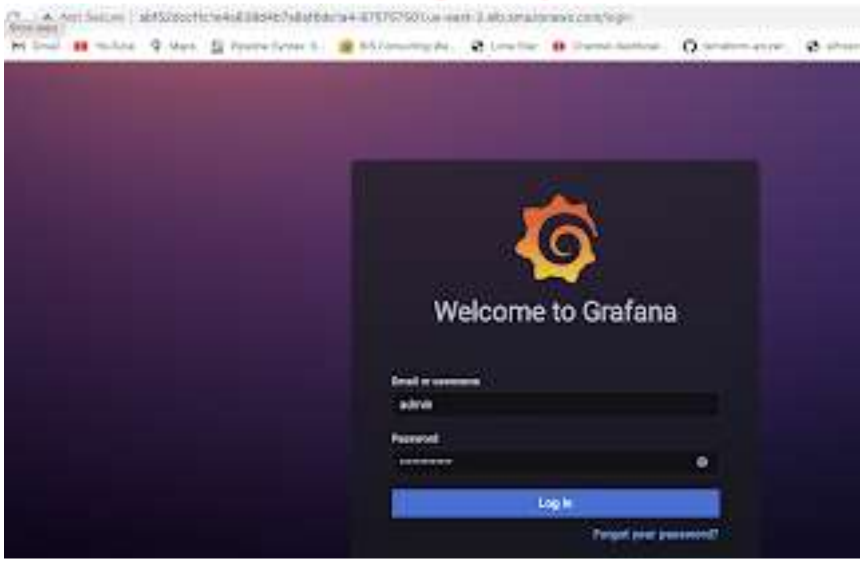
# kubectl get svc -n prometheus



```
amazingdevops_com@cloudshell:~ (glass-badge-383322)$ kubectl get svc -n prometheus-josh
NAME                                      TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)                        AGE
alertmanager-operated                     ClusterIP      None           <none>           9093/TCP,9094/TCP,9094/UDP     4h20m
prometheus-operated                       ClusterIP      None           <none>           9090/TCP                       4h20m
stable-grafana                            LoadBalancer   10.68.0.153    34.148.51.222    80:30372/TCP                   4h20m
stable-kube-prometheus-sta-alertmanager   ClusterIP      10.68.15.12    <none>           9093/TCP                       4h20m
stable-kube-prometheus-sta-operator       ClusterIP      10.68.2.214    <none>           443/TCP                        4h20m
stable-kube-prometheus-sta-prometheus     LoadBalancer   10.68.13.206   34.148.242.68    9090:30983/TCP                 4h20m
stable-kube-state-metrics                 ClusterIP      10.68.13.19    <none>           8080/TCP                       4h20m
stable-prometheus-node-exporter           ClusterIP      10.68.11.61    <none>           9100/TCP                       4h20m
amazingdevops_com@cloudshell:~ (glass-badge-383322)$
```

<u>Access Grafana UI in the browser</u>

Get the URL from the above screenshot and put in the browser.



UserName: admin

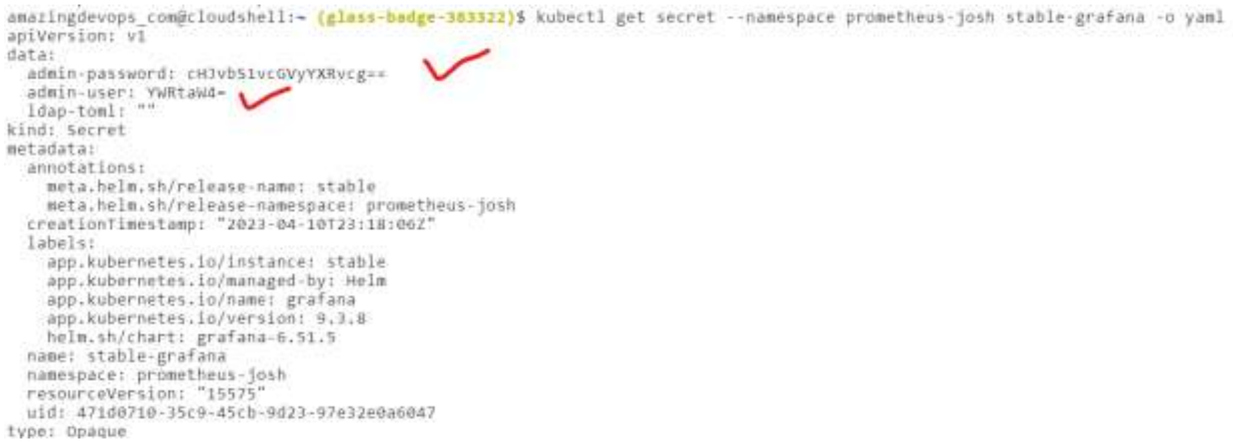
Password: prom-operator

How to get your Admin and Password

All usernames and passwords are stored in the secret manifests.

Run the command.

# kubectl get secret --namespace prometheus-josh stable-grafana -o yaml



```
amazingdevops_com@cloudshell:~ (glass-badge-383322)$ kubectl get secret --namespace prometheus-josh stable-grafana -o yaml
apiVersion: v1
data:
  admin-password: cHJvbS1vcGVyYXRvcg==
  admin-user: YWRtaW4=
  ldap-toml: ""
kind: Secret
metadata:
  annotations:
    meta.helm.sh/release-name: stable
    meta.helm.sh/release-namespace: prometheus-josh
  creationTimestamp: "2023-04-10T23:18:06Z"
  labels:
    app.kubernetes.io/instance: stable
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: grafana
    app.kubernetes.io/version: 9.3.8
    helm.sh/chart: grafana-6.51.5
  name: stable-grafana
  namespace: prometheus-josh
  resourceVersion: "15575"
  uid: 471d0710-35c9-45cb-9d23-97e32e0a6047
type: Opaque
```

Base 64 Encoding and Decoding

Using Base64 encoding is not an effective way to protect passwords. Base64 is a method of encoding data, which can be easily decoded by anyone who has access to the encoded text. Therefore, using Base64 encoding to protect passwords is not recommended as it does not provide any encryption or hashing to ensure the security of the password.

Instead, a more secure method for protecting passwords is to use a one-way hash function. A one-way hash function takes the password and transforms it into a fixed-size string of characters. The hash function is designed in such a way that it is easy to compute the hash value from the password, but it is practically impossible to reverse the process and determine the password from the hash value.

This means that even if a hacker gains access to the hashed password, they will not be able to determine the original password. When a user enters their password, the system will hash it and compare it to the stored hash value. If the hashes match, the user is granted access.

To further enhance the security of passwords, it is recommended to use a strong and unique password for each account, and to store the hashed passwords in a secure database that is not easily accessible by unauthorized users. Additionally, multi-factor authentication can be used to add an extra layer of security to the login process.

Command to encode a password.

```
# echo -n "password123" | base64
```

This command will encode the string "password123" in Base64. The **-n** option is used to prevent the **echo** command from adding a newline character to the end of the string, which can cause issues when decoding the Base64-encoded string.

Similarly, to decode a Base64-encoded string, you can use the following command:

Command to decode a password.

```
# echo "cGFzc3dvcmQxMjM=" | base64 -d
```

Let us now decode the admin-user and admin-password using base 64

```
data:
  admin-password: cHJvbS1vcGVyYXRvcg==
  admin-user: YWRtaW4=
  ldap-toml: ""
```

```
kind: Secret
metadata:
  annotations:
    meta.helm.sh/release-name: stable
    meta.helm.sh/release-namespace: prometheus-josh
  creationTimestamp: "2023-04-10T23:18:06Z"
```

```
# echo " cHJvbS1vcGVyYXRvcg==" | base64 -d
```

prom-operator

```
# echo " YWRtaW4=" | base64 -d
```

admin

Create Dashboard in Grafana

In Grafana, we can create various kinds of dashboards as per our needs.

How to Create Kubernetes Monitoring Dashboard?

For creating a dashboard to monitor the cluster:

Click '+' button on the left panel and select 'Import'.

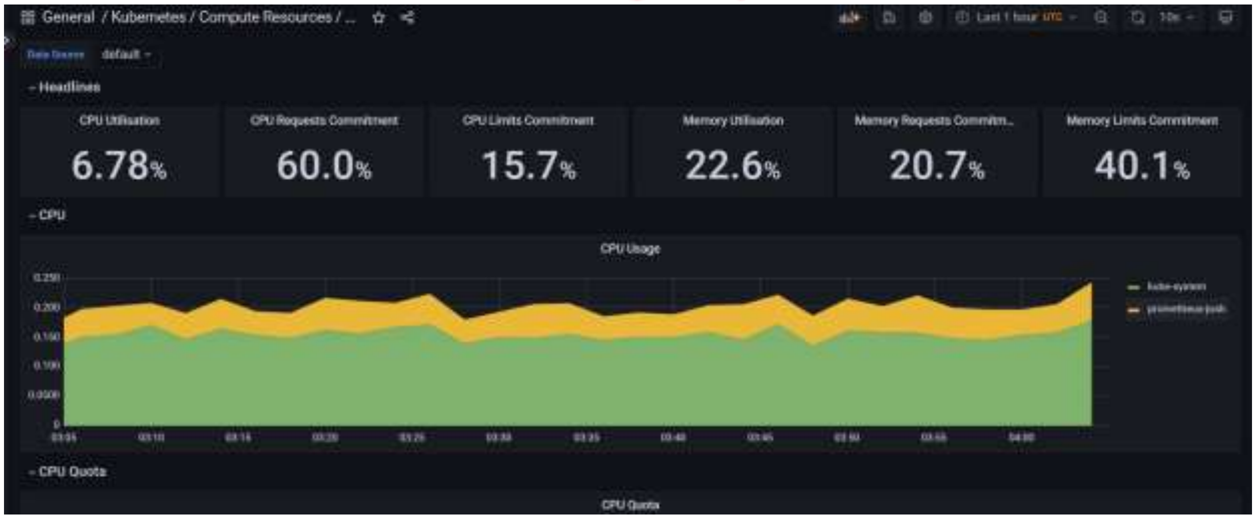
Enter 12740 dashboard id under Grafana.com Dashboard.

Click 'Load'.

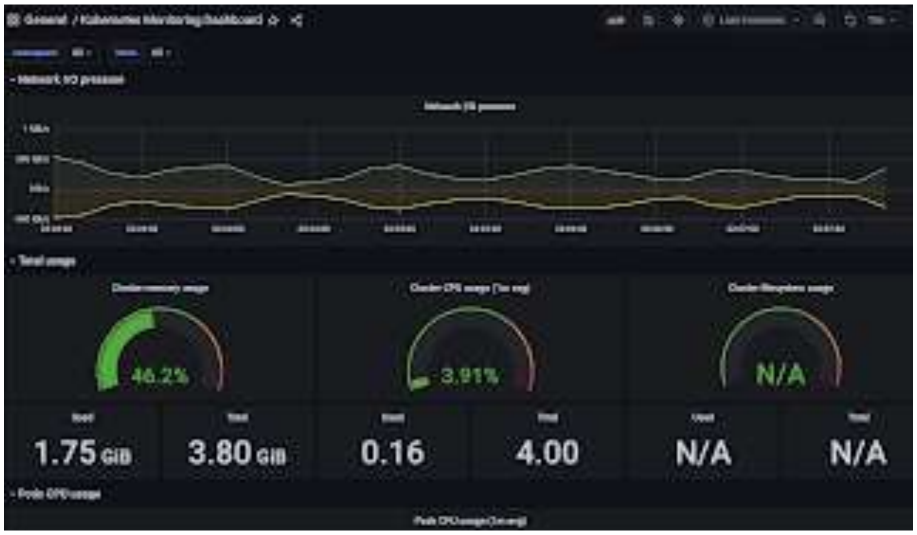
Select 'Prometheus' as the endpoint under prometheus data sources drop down.

Click 'Import'.

A dashboard will appear.



This will show a monitoring dashboard for all cluster nodes.

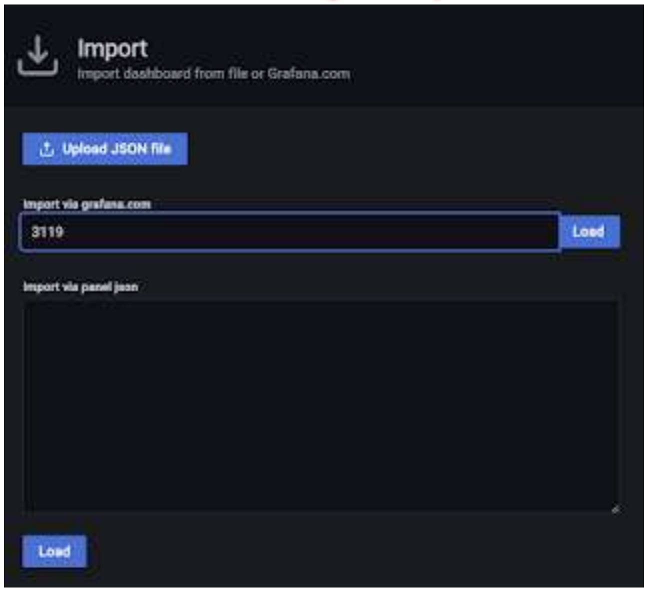


How to Create Kubernetes Cluster Monitoring Dashboard?

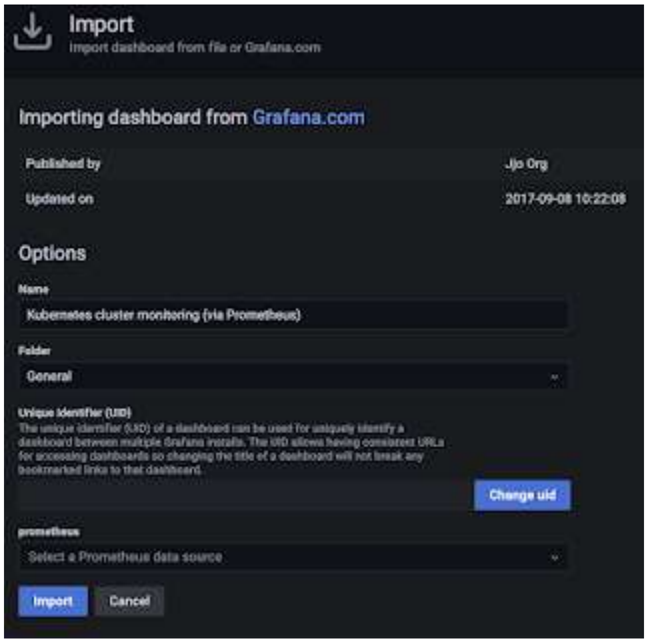
For creating a dashboard to monitor the cluster:

Click '+' button on left panel and select ‘Import’.
Enter 3119 dashboard id under Grafana.com Dashboard.
Click ‘Load’.
Select ‘Prometheus’ as the endpoint under prometheus data sources drop down.
Click ‘Import’.
This will show monitoring dashboard for all cluster nodes.

AmazingDevOps



3119



Importing dashboard from Grafana.com

Published by	Jjo Org
Updated on	2017-09-08 10:22:08

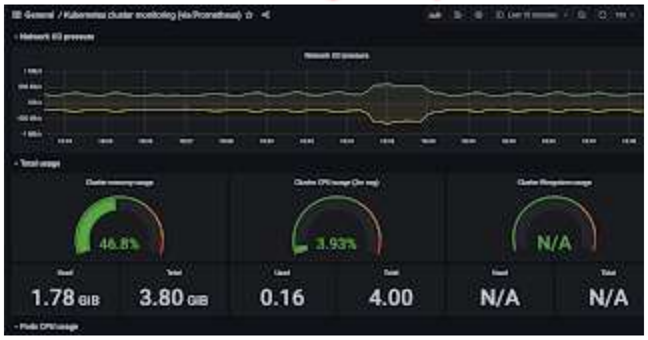
Options

Name: Kubernetes cluster monitoring (via Prometheus)

Folder: General

Unique identifier (UID)
The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

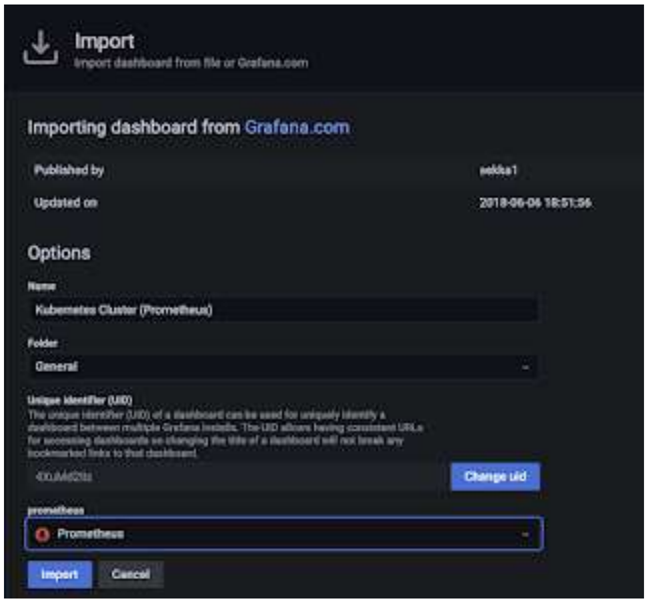
prometheus: Select a Prometheus data source



Create POD Monitoring Dashboard

For creating a dashboard to monitor the cluster:

Click '+' button on left panel and select ‘Import’.
Enter 6417 dashboard id under Grafana.com Dashboard.
Click ‘Load’.
Select ‘Prometheus’ as the endpoint under prometheus data sources drop down.
Click ‘Import’.



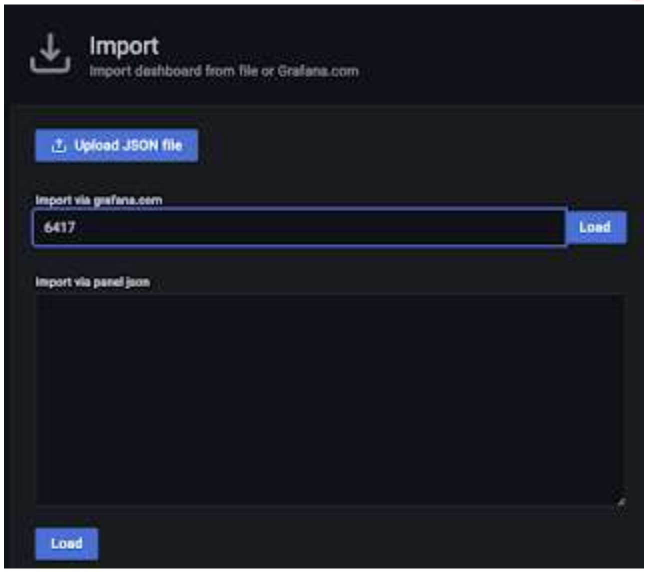
Import
Import dashboard from file or Grafana.com

Importing dashboard from Grafana.com

Published by: sekka1
Updated on: 2018-06-06 18:51:56

Options

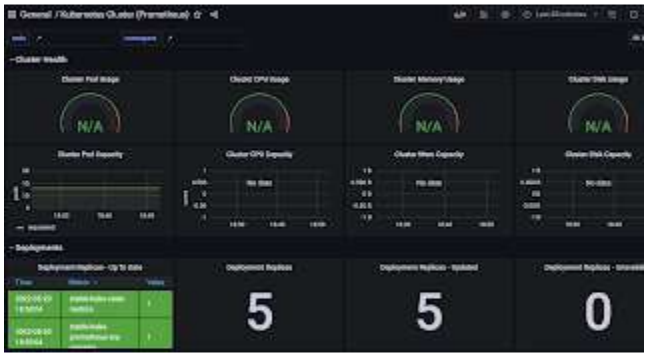
Name: Kubernetes Cluster (Prometheus)
Folder: General
Unique identifier (UID)
The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.
4XuMd2Iiz
prometheus: Prometheus



Import via grafana.com

6417

This will show a monitoring dashboard for all cluster nodes.


General / Kubernetes Cluster (Prometheus)
Cluster Health
N/A
N/A
N/A
N/A
Deployments
Deployment Replicas
5
5
0