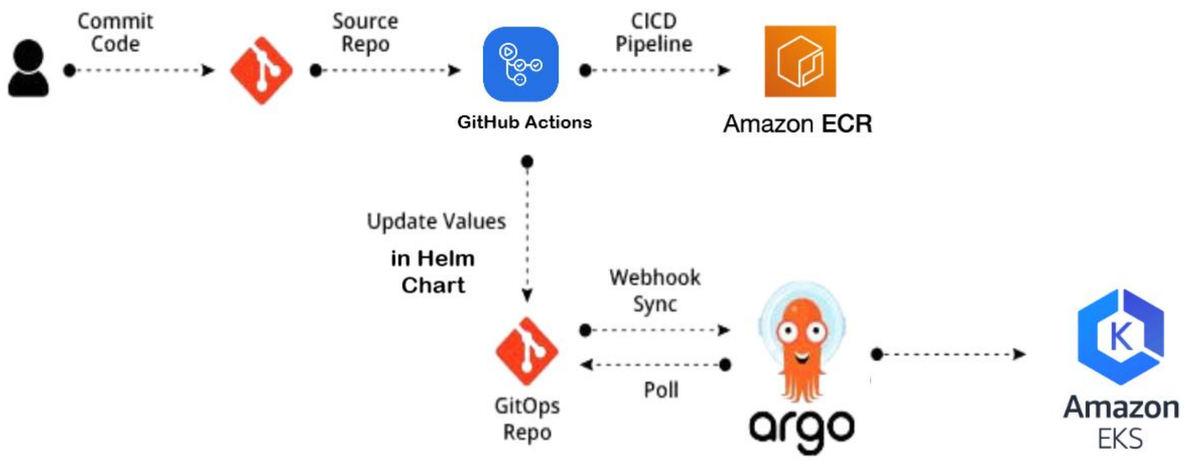
Project : GitOps: CI/CD using GitHub Actions and ArgoCD on Kubernetes

Description:

Creating GitOps (CI/CD) Workflow with GitHub Actions for Building and Pushing Docker Images and ArgoCD for deploying a NodeJS application on Amazon Elastic Kubernetes Cluster.

Technology Stack:

Github, Github Actions, ECR, ArgoCD, Helm and EKS



Source Code: https://github.com/AQtar-004/GitOps-CI-CD_Project.git

GitOps Principle
GitOps is a set of best practices and principles for managing infrastructure and deploying applications. It leverages Git as the single source of truth for both infrastructure and application code

GITHUB Actions
GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform provided by GitHub. It allows you to automate workflows, enabling you to build, test, and deploy your code directly from your GitHub repository. GitHub Actions is tightly integrated with the GitHub platform, making it easy to set up and manage automated workflows for your projects.

Amazon Elastic Container Registry (ECR) is a fully managed container registry service provided by Amazon Web Services (AWS). It allows you to store, manage, and deploy

Docker container images for your applications on AWS. ECR is tightly integrated with other AWS services, making it convenient for containerized applications running on AWS infrastructure.

ArgoCD

ArgoCD is an open-source declarative continuous delivery (CD) tool for Kubernetes. It enables GitOps practices by automating the deployment of applications and configurations to Kubernetes clusters based on definitions stored in Git repositories. ArgoCD helps ensure that the desired state of the applications in the Kubernetes clusters matches the state defined in the Git repositories.

ArgoCD Installation: https://argo-cd.readthedocs.io/en/stable/getting_started/

To get secret: kubectl get secret argocd-initial-admin-secret -n argocd -o json

To get the password: kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d

Key features of ArgoCD include:

- **Declarative Configuration:**
  ArgoCD uses a declarative approach, where the desired state of applications and configurations is specified in YAML files stored in Git repositories. These YAML files define Kubernetes manifests, version information, and other application-related settings.

- **GitOps Workflow:**
  ArgoCD follows the GitOps workflow, where changes to the applications are made through Git commits. When changes are pushed to a Git repository, ArgoCD automatically detects these changes and synchronizes the Kubernetes cluster with the new desired state.

- **Automated Synchronization:**
  ArgoCD continuously monitors the Git repositories for changes and automatically synchronizes the state of the applications in the Kubernetes clusters. This automated synchronization ensures that the actual state in the clusters aligns with the declared state in the Git repositories.

- **Rollback and History:**
  ArgoCD keeps track of the deployment history for each application. If issues arise with a new deployment, you can easily roll back to a previous version using the ArgoCD UI or CLI. This provides a way to revert to a known-good state in case of problems.

- **Multi-Cluster and Multi-Tenancy:**
  ArgoCD supports managing applications across multiple Kubernetes clusters, making it suitable for multi-cluster deployments. It also supports multi-tenancy, allowing different teams or users to manage their applications independently.

HELM
Helm is a package manager for Kubernetes applications. It simplifies the deployment and management of applications on Kubernetes clusters by providing a higher-level abstraction and a standardized way of defining, sharing, and managing Kubernetes manifests and configurations.
Helm packages are called charts. A chart is a collection of pre-configured Kubernetes resources, such as deployments, services, and ingress definitions, along with optional templates and values. Charts provide a consistent way to define and package applications.

Install a chart:
helm install <release-name> <chart-name>

Upgrade a release:
helm upgrade <release-name> <chart-name>

Uninstall a release:
helm uninstall <release-name>

Ingress:
https://artifacthub.io/packages/helm/ingress-nginx/ingress-nginx

Ingress is a Kubernetes resource that manages external access to services within a Kubernetes cluster. It provides HTTP and HTTPS routing to services based on defined rules, allowing you to expose multiple services under a single IP address or domain name.

Key Features of Ingress:

- Routing: Ingress allows you to define rules for routing external HTTP/S traffic to your internal services. This means you can route traffic based on URL paths or hostnames.
- SSL Termination: Ingress can manage SSL certificates and handle HTTPS traffic. This allows you to secure your applications without managing SSL certificates in individual services.
- Load Balancing: Ingress can distribute traffic among different backend services, enabling load balancing and high availability.
- Rewrites and Redirects: Ingress can rewrite URLs or redirect traffic based on specific conditions, making it easier to manage application endpoints.
- Authentication: Some Ingress controllers can enforce authentication mechanisms, adding an additional layer of security.

Kubectl cheat sheet: https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Horizontal Pod Autoscaler (HPA): https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

Create EKS cluster with EKSCTL:

EKSCTL is a command line interface (CLI) tool that allows users to create and manage Amazon EKS clusters
Features: Create clusters, update existing clusters, manage node groups, upgrade Kubernetes versions, and delete clusters

Install eksctl:
```
$ curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -
s)_amd64.tar.gz" | tar xz -C /tmp
$ sudo mv /tmp/eksctl /usr/local/bin
```

Create a Cluster Configuration YAML File

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-eks-cluster
  region: ap-south-1

nodeGroups:
  - name: ng-1
    instanceType: t3.medium
    desiredCapacity: 2
    minSize: 2
    maxSize: 4
    volumeSize: 20
    ssh:
      allow: true
      publicKeyName: eks-nodes
```

```
$ eksctl create cluster -f cluster-config.yml
```

If you only want to modify the scaling settings (min/max nodes, desired capacity) or the volume size use the following command

```
$ eksctl scale nodegroup --cluster=my-eks-cluster --name=ng-1 --nodes=4 --nodes-min=3 --
nodes-max=6
```

To delete the cluster: $ eksctl delete cluster --name my-eks-cluster

To install nginx-ingress controller run the following commands

```
$ helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
$ helm repo update
$ helm install nginx-ingress ingress-nginx/ingress-nginx
```

Setup monitoring for K8s Cluster using Prometheus and Grafana

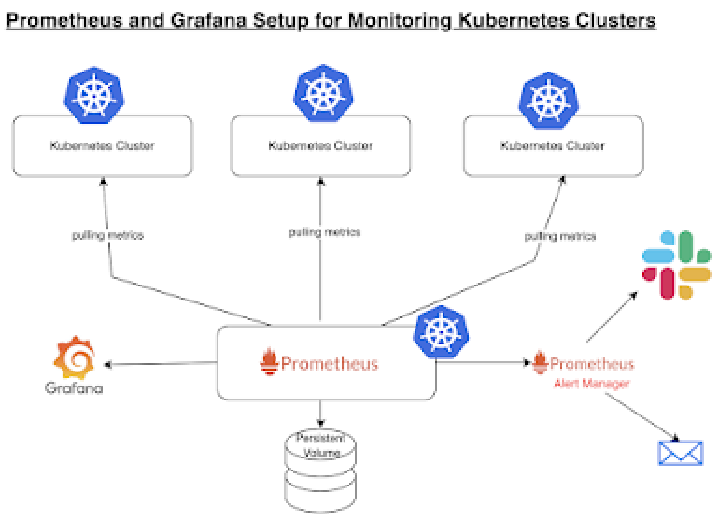
What is Prometheus?

- Prometheus is an open-source monitoring tool
- Provides out-of-the-box monitoring capabilities for the Kubernetes container orchestration platform. It can monitor servers and databases as well.
- Collects and stores metrics as time-series data, recording information with a timestamp
- It is based on pull and collects metrics from targets by scraping metrics HTTP endpoints.

What is Grafana?

- Grafana is an open-source visualisation and analytics software.
- It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored.

Prometheus Architecture



Key components:

- Prometheus server - Processes and stores metrics data
- Alert Manager - Sends alerts to any systems/channels
- Grafana - Visualize scraped data in UI

Installation Method:

The are many ways you can setup Prometheus and Grafana. You can install in following ways:

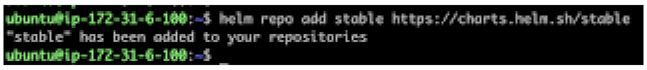
- Create all configuration files of both Prometheus and Grafana and execute them in right order.

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

- Helm chart - Using helm to install Prometheus Operator including Grafana

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

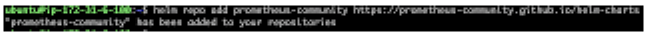
helm repo add stable https://charts.helm.sh/stable



```
ubuntu@ip-172-31-6-100:~$ helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
ubuntu@ip-172-31-6-100:~$ _
```

# Add Prometheus Helm repo

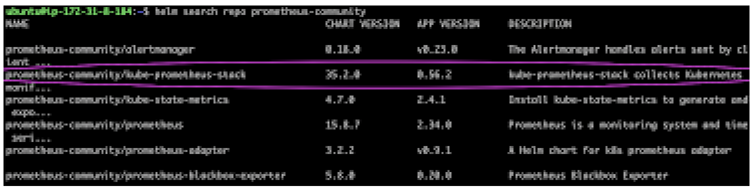
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts



```
ubuntu@ip-172-31-6-100:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
```

helm search repo prometheus-community

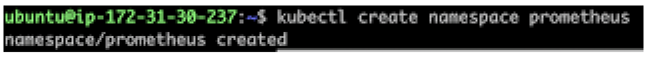
Prometheus and grafana helm chart moved to kube prometheus stack



```
ubuntu@ip-172-31-8-184:~$ helm search repo prometheus-community
NAME                                            CHART VERSION   APP VERSION     DESCRIPTION

prometheus-community/alertmanager               0.18.0          v0.23.0         The Alertmanager handles alerts sent by cl
lient ...
prometheus-community/kube-prometheus-stack      35.2.0          0.56.2          kube-prometheus-stack collects Kubernetes
manif...
prometheus-community/kube-state-metrics         4.7.0           2.4.1           Install kube-state-metrics to generate and
 expo...
prometheus-community/prometheus                 15.8.7          2.34.0          Prometheus is a monitoring system and time
 seri...
prometheus-community/prometheus-adapter         3.2.2           v0.9.1          A Helm chart for k8s prometheus adapter

prometheus-community/prometheus-blackbox-exporter   5.6.0       0.20.0          Prometheus Blackbox Exporter
```

Create Prometheus namespace

kubectl create namespace prometheus

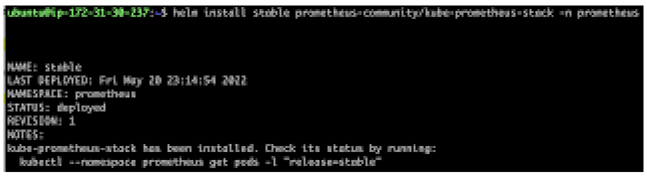


```
ubuntu@ip-172-31-30-237:~$ kubectl create namespace prometheus
namespace/prometheus created
```

Install kube-prometheus-stack

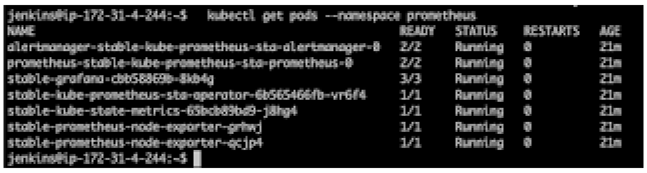
Below is helm command to install kube-prometheus-stack. The helm repo kube-stack-prometheus (formerly prometheus-operator) comes with a grafana deployment embedded.

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



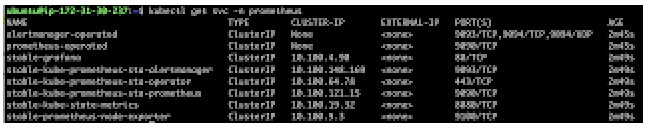
```
ubuntu@ip-172-31-30-237:~$ helm install stable prometheus-community/kube-prometheus-stack -n prometheus



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



```
jenkins@ip-172-31-4-244:~$   kubectl get pods --namespace prometheus
NAME                                                      READY   STATUS    RESTARTS   AGE
alertmanager-stable-kube-prometheus-sta-alertmanager-0   2/2     Running   0          21m
prometheus-stable-kube-prometheus-sta-prometheus-0       2/2     Running   0          21m
stable-grafana-cbb58869b-8kb4g                           3/3     Running   0          21m
stable-kube-prometheus-sta-operator-6b565466fb-vr6f4     1/1     Running   0          21m
stable-kube-state-metrics-65bcb89bd9-j8hg4               1/1     Running   0          21m
stable-prometheus-node-exporter-grhwj                    1/1     Running   0          21m
stable-prometheus-node-exporter-qcjp4                    1/1     Running   0          21m
jenkins@ip-172-31-4-244:~$
```

kubectl get svc -n prometheus



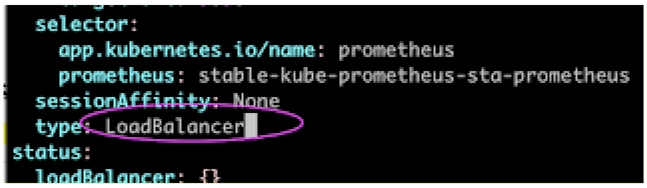
```
ubuntu@ip-172-31-30-237:~$ kubectl get svc -n prometheus
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   2m45s
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                     2m45s
stable-grafana                            ClusterIP   10.100.4.90      <none>        80/TCP                       2m49s
stable-kube-prometheus-sta-alertmanager   ClusterIP   10.100.148.160   <none>        9093/TCP                     2m49s
stable-kube-prometheus-sta-operator       ClusterIP   10.100.64.78     <none>        443/TCP                      2m49s
stable-kube-prometheus-sta-prometheus     ClusterIP   10.100.121.15    <none>        9090/TCP                     2m49s
stable-kube-state-metrics                 ClusterIP   10.100.29.32     <none>        8080/TCP                     2m49s
stable-prometheus-node-exporter           ClusterIP   10.100.9.3       <none>        9100/TCP                     2m49s
```

This confirms that prometheus and grafana have been installed successfully using Helm.

In order to make prometheus and grafana available outside the cluster, use LoadBalancer or NodePort instead of ClusterIP.

Edit Prometheus Service

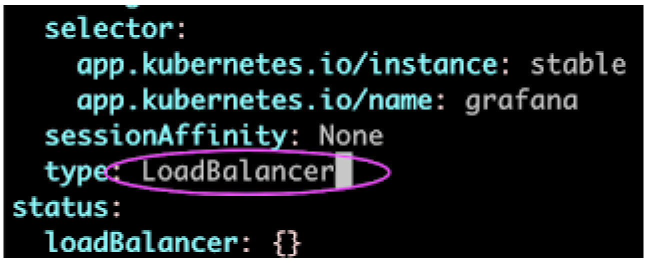
kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus



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

Edit Grafana Service

kubectl edit svc stable-grafana -n prometheus



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

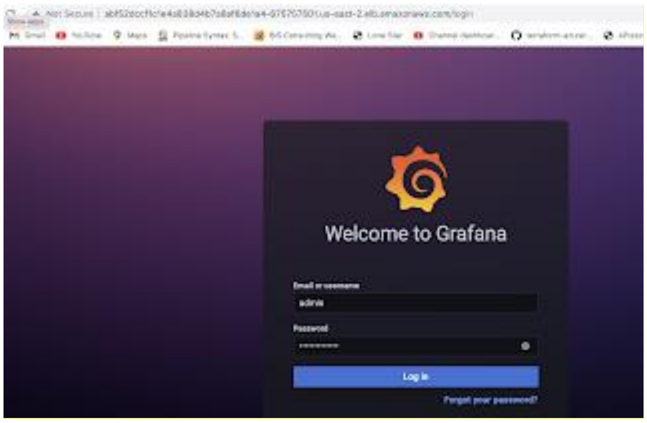
Verify if service is changed to LoadBalancer and also to get the Load Balancer URL.

kubectl get svc -n prometheus

```
[illegible]
```

<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

<u>Create Dashboard in Grafana</u>

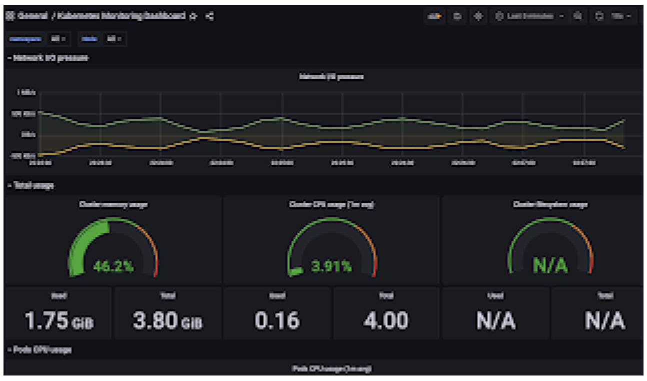
In Grafana, we can create various kinds of dashboards as per our needs.

<u>How to Create Kubernetes Monitoring Dashboard?</u>

For creating a dashboard to monitor the cluster:

Click '+' button on 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’.

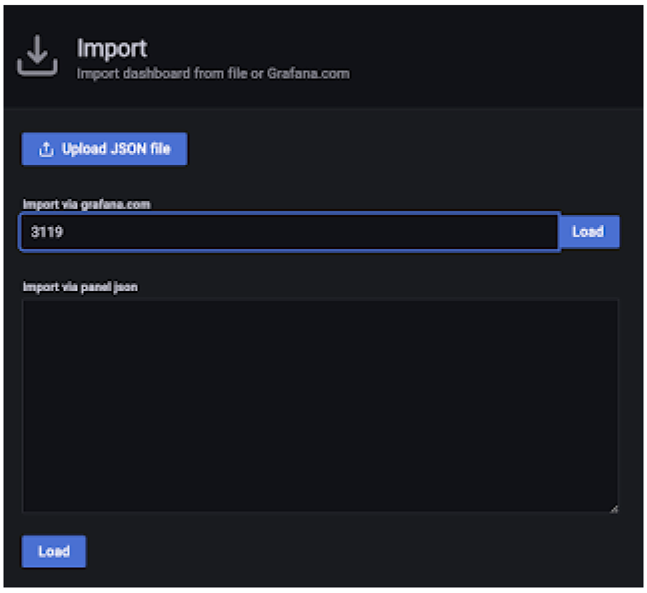
This will show monitoring dashboard for all cluster nodes



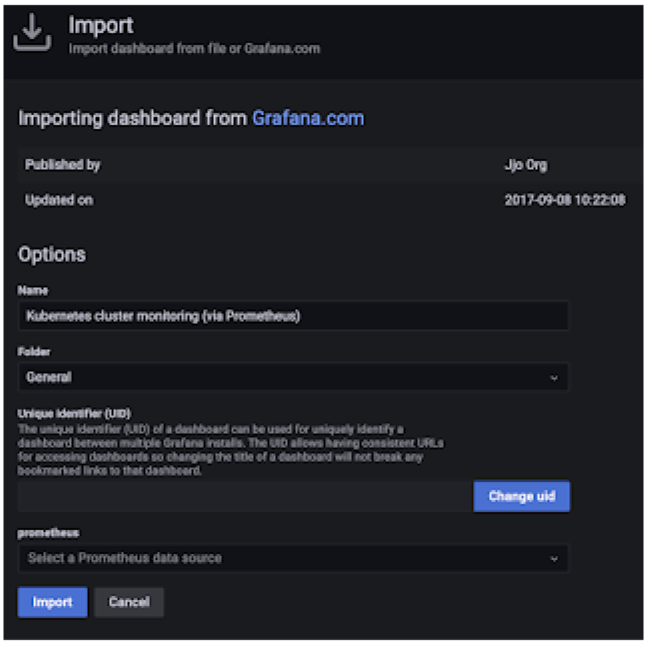
<u>How to Create Kubernetes Cluster Monitoring Dashboard?</u>

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



Import via grafana.com: 3119



Import

Import dashboard from file or Grafana.com

Importing dashboard from Grafana.com

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

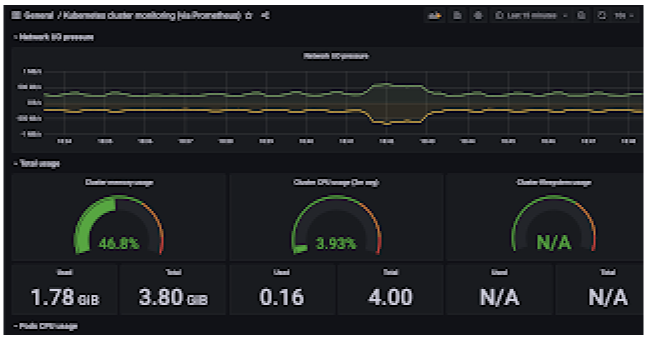
Options

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.

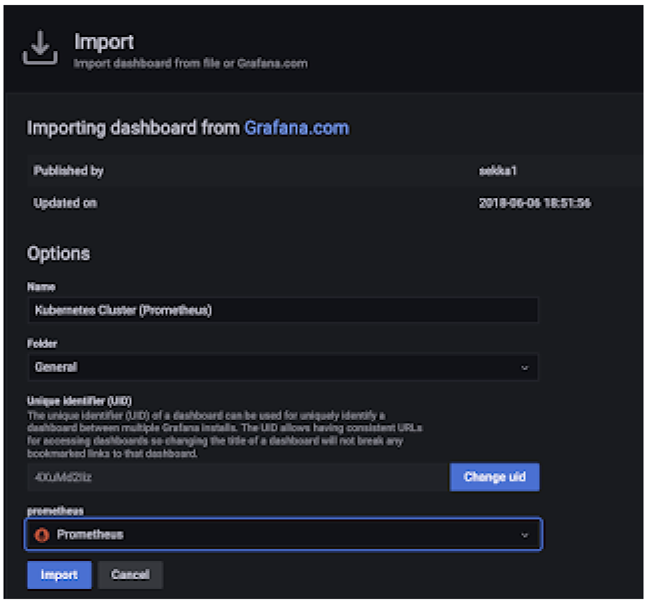
prometheus: Select a Prometheus data source



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

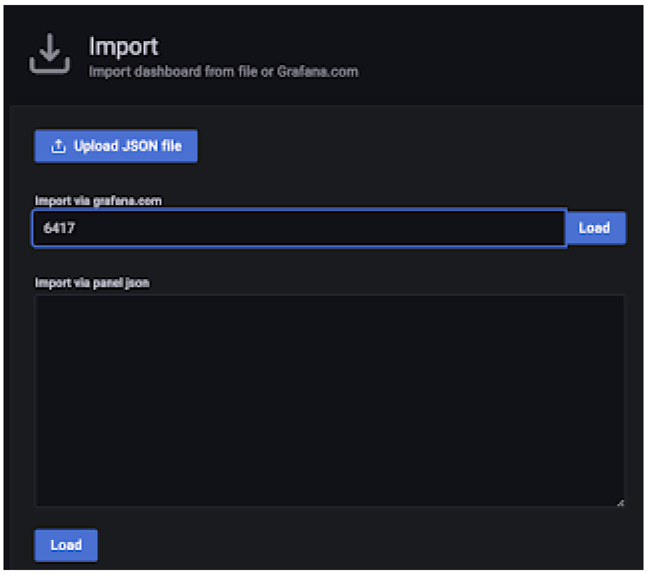
Name: Kubernetes Cluster (Prometheus)

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



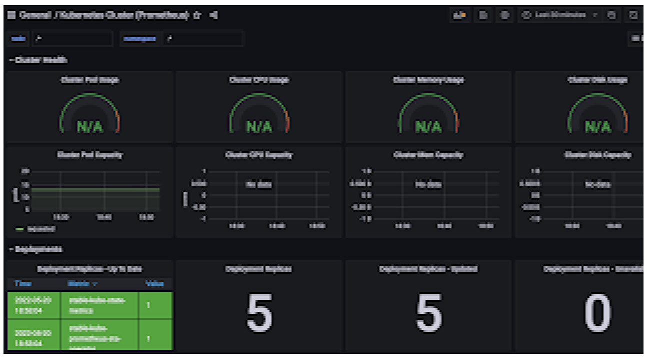
Import

Import dashboard from file or Grafana.com

Import via grafana.com: 6417

Import via panel json

This will show monitoring dashboard for all cluster nodes.



Prometheus query Cluster CPU Utilization:

To monitor the overall CPU utilization of your Kubernetes cluster, you can use the node_cpu_seconds_total metric. This metric provides the cumulative CPU time consumed by all CPUs on a node. You can calculate the cluster-wide CPU utilization by aggregating this metric across all nodes.

```
sum(rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100
```

This query calculates the CPU utilization by subtracting the idle time from 100% (assuming mode="idle" corresponds to idle CPU time). The rate() function calculates the per-second rate of change in CPU utilization.

To uninstall the package with helm

```
$ helm list
```

```
$ helm uninstall stable --namespace prometheus
```

Expected O/P: release "prometheus" uninstalled