**Installation and Setup Guide for Kubernetes Cluster with Multiple Masters and Nodes, Including Monitoring**

**RECORD OF REVISIONS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO** | **VERSION** | | **DATE OF CHANGE** | **CHANGES MADE** |
| **FROM** | **TO** |
| 1 | Draft |  |  | Initial Version |
| 2 |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Reference** |  | **Issue No.** |  |
| **Release Date** |  | **Total Number of pages** | 55 |
|  | **Name** | **Signature** | **Date** |
| **Prepared by** | Allampalli Sai Vishnu Vardhan,  S. N. S. K. Vamsi,  Dinesh Kumar,  Bolishetti Srinath | Allampalli Sai Vishnu Vardhan | 07-12-2023 |
| **Reviewed and approved by** |  |  |  |

Contents

[Problem Statement 5](#_Toc152862537)

[Single Point of Failure: 5](#_Toc152862538)

[Reduced High Availability: 5](#_Toc152862539)

[Limited Scalability: 5](#_Toc152862540)

[Reliability Concerns: 5](#_Toc152862541)

[Introduction 6](#_Toc152862542)

[What is Kubernetes: 6](#_Toc152862543)

[Master Node: 6](#_Toc152862544)

[Components of Master Node: 6](#_Toc152862545)

[API Server: 6](#_Toc152862546)

[etcd: 6](#_Toc152862547)

[Controller Manager: 6](#_Toc152862548)

[Scheduler: 7](#_Toc152862549)

[Worker Node: 7](#_Toc152862550)

[Components of Worker Node: 7](#_Toc152862551)

[Kubelet: 7](#_Toc152862552)

[Container Runtime: 7](#_Toc152862553)

[Kube Proxy: 7](#_Toc152862554)

[Pods: 7](#_Toc152862555)

[Installation of Haproxy: 10](#_Toc152862556)

[Namespace: 14](#_Toc152862557)

[What is Namespace: 14](#_Toc152862558)

[Creating Namespaces 14](#_Toc152862559)

[PV (Persistent Volume) and PVC (Persistent Volume Claim): 14](#_Toc152862560)

[Persistent Volume (PV): 14](#_Toc152862561)

[Creating a Persistent Volume (PV) 15](#_Toc152862562)

[Persistent Volume Claim (PVC): 15](#_Toc152862563)

[Creating a Persistent Volume Claim (PVC) 16](#_Toc152862564)

[Pod: 16](#_Toc152862565)

[Deployment: 16](#_Toc152862566)

[Creating Deployment 17](#_Toc152862567)

[Service 17](#_Toc152862568)

[The main types of Services in Kubernetes : 18](#_Toc152862569)

[1. ClusterIP: 18](#_Toc152862570)

[2. NodePort: 18](#_Toc152862571)

[3. LoadBalancer: 18](#_Toc152862572)

[4. ExternalName: 18](#_Toc152862573)

[5. Headless Service: 18](#_Toc152862574)

[Creating Service 19](#_Toc152862575)

[ReplicaSet: 19](#_Toc152862576)

[Desired Replicas: 19](#_Toc152862577)

[Selector: 19](#_Toc152862578)

[Pod Template: 19](#_Toc152862579)

[Self-Healing: 20](#_Toc152862580)

[Creating ReplicaSet 20](#_Toc152862581)

[StatefulSets: 20](#_Toc152862582)

[Stable Network Identifiers: 20](#_Toc152862583)

[Stable Storage: 21](#_Toc152862584)

[Ordered Deployment and Scaling: 21](#_Toc152862585)

[Headless Service: 21](#_Toc152862586)

[Creating StatefulSet 22](#_Toc152862587)

[ConfigMaps: 22](#_Toc152862588)

[Creating ConfigMap 23](#_Toc152862589)

[Using ConfigMap in a Pod: 23](#_Toc152862590)

[Secrets: 23](#_Toc152862591)

[Creating a Secret 23](#_Toc152862592)

[Using Secret in a Pod: 24](#_Toc152862593)

[Helm 25](#_Toc152862594)

[Prerequisites 25](#_Toc152862595)

[Helm Setup 25](#_Toc152862596)

[To create helm chart 25](#_Toc152862597)

[To apply helm chart 26](#_Toc152862598)

[Helm file: 26](#_Toc152862599)

[Ingress: 27](#_Toc152862600)

[How to install Nginx Ingress Controller: 27](#_Toc152862601)

[To install ingress controller 27](#_Toc152862602)

[Sample Flow Chart for Ingress 28](#_Toc152862603)

[Configuring ingress rules for path-based routing: 29](#_Toc152862604)

[Routing rules for ingress path-based routing: 29](#_Toc152862605)

[How to access application through path based: 30](#_Toc152862606)

[Name based virtual hosting in ingress: 30](#_Toc152862607)

[Routing rules for ingress name-based routing: 31](#_Toc152862608)

[Create a new A record for name based virtual hosting: 32](#_Toc152862609)

[Kubernetes Cluster Monitoring using Prometheus and Grafana 33](#_Toc152862610)

[What is Prometheus: 33](#_Toc152862611)

[What is Grafana: 33](#_Toc152862612)

[Installation of Prometheus and Grafana in Kubernetes-Cluster using helm: 34](#_Toc152862613)

[Prerequisites: 34](#_Toc152862614)

[Creating a Namespace: 34](#_Toc152862615)

[Installation of Prometheus & Grafana using helm charts: 34](#_Toc152862616)

[Grafana Dashboard creation for visual representation of Kubernetes Cluster Metrics: 40](#_Toc152862617)

[Setup alert manager on grafana: 44](#_Toc152862618)

# Problem Statement

Single Point of Failure: A single-master setup risks cluster downtime if the master node fails, affecting the availability of applications and services.

Reduced High Availability: Without multiple masters, achieving high availability becomes challenging, potentially leading to service disruptions during maintenance or failures.

Limited Scalability: Scaling the cluster might be constrained as a single master could become a bottleneck, limiting the number of nodes and overall cluster capacity.

Reliability Concerns: The single Kubernetes cluster can compromise the reliability of the Kubernetes control plane, impacting critical cluster management functions.

## Introduction

## What is Kubernetes:

* Kubernetes is an open-source container orchestration engine for automating deployment, scaling, and management of containerized applications.
* A Kubernetes cluster is a set of machines (nodes) that collectively run a set of containerized applications. It consists of at least one master node and one or more worker nodes.

## Master Node:

* + The master node is responsible for managing the cluster and making global decisions. It hosts components like the API server, controller manager, scheduler, and etcd (a distributed key-value store used for cluster state).

## Components of Master Node:

API Server: The central management hub that provides the Kubernetes API, acting as a frontend for controlling the cluster. It validates and processes RESTful API requests, then communicates with the etcd datastore.

etcd: A distributed key-value store that serves as Kubernetes' primary datastore. It stores cluster configuration data, including information about nodes, pods, configurations, and secrets, ensuring consistent and reliable storage for the entire cluster.

Controller Manager: This component manages various controllers that regulate the state of the cluster. Controllers watch the state of the cluster through the API server and work to maintain the desired state by reconciling any differences. Examples include Node Controller, Replication Controller, Endpoint Controller, etc.

Scheduler: Responsible for placing newly created pods onto nodes based on resource availability, constraints, and other policies. It schedules pods onto suitable nodes in the cluster.

## Worker Node:

* + Worker nodes are the machines where the application containers run. They are managed by the master node and are responsible for running the actual workloads.

## Components of Worker Node:

Kubelet: The primary agent running on each worker node that communicates with the Kubernetes master node. It is responsible for ensuring that containers (inside pods) are running and healthy on the node. Kubelet takes care of starting, stopping, and maintaining application containers as specified by the Kubernetes manifests.

Container Runtime: The software responsible for running containers. Kubernetes supports various container runtimes like Docker, containered, CRI-O, etc. This component manages the execution of containers on the node, handling image pulls, container creation, networking, and resource isolation.

Kube Proxy: Manages network communication for services running within the cluster. It maintains network rules (iptables) on the node, enabling communication between different pods and external traffic to reach the appropriate services.

Pods: Pods are the smallest deployable units in Kubernetes. They encapsulate one or more containers with shared networking and storage. Each worker node hosts multiple pods that collectively run the actual application workloads.

**Multi-Master Kubernetes Blueprint:**

A screenshot of a computer

Description automatically generated

**Prerequisites**

**Server Requirements:**

1. **Operating System:**
   * Linux-based distributions (e.g., Ubuntu, CentOS, Debian) are commonly used and recommended.
2. **Hardware:**
   * Sufficient CPU and RAM resources based on your expected workloads and data volume. Typically, at least 2 CPU cores and 4 GB of RAM are recommended for (Master) & 1 CPU and 2 GB of RAM for (Worker) small to medium setups.

**Networking**

**Network Access:**

• Ensure that the server has internet access to download software packages and updates during the installation process.

Setup for Haproxy and Kubernetes master and nodes:

## Installation of Haproxy:

Installing HAPROXY

1. $ sudo apt install haproxy -y

2. $ sudo systemctl status haproxy

3. $ sudo systemctl start haproxy

* configure a haproxy by going in to

4. $ sudo cd /etc/haproxy/haproxy.cfg

----> insert the front and backend data

frontend kubernetes

bind <private ip of haproxy server>:80

option tcplog

mode tcp

default\_backend kubernetes-master-nodes

backend kubernetes-master-nodes

mode tcp

balance roundrobin

option tcp-check

server kubernetes-master1 <private ip of master node -1>:6443 check fall 3 rise 2

server kubernetes-master2 <private ip of master node -2>:6443 check fall 3 rise 2

* Run on master and nodes

1. $ apt-get update && apt-get install -y docker.io

2. $ vi /etc/docker/daemon.json

{

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2"

}

1. $ sudo systemctl enable docker

$ sudo systemctl daemon-reload

$ sudo systemctl restart docker

$ sudo systemctl status docker

4. $ apt-get update && apt-get install -y apt-transport-https

5. $ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

6. $ sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"

7. $ apt-get update

8. $ apt-get install -y kubelet kubeadm kubectl

9. $ apt-mark hold kubelet kubeadm kubectl

Create kubernetes cluster (Run on Master)

* Run below command in k8s-master where haproxy is present after master and nodes are installed

1. $ sudo kubeadm init --control-plane-endpoint="<ip>:80" --upload-certs

2. Save ‘kubadm join’

(use the command “$ kubeadm token create --print-join-command” to regenerate kubeadm token)

3. $ cp /etc/kubernetes/admin.conf $HOME/

4. $ chown $(id -u):$(id -g) $HOME/admin.conf

5. $ echo 'export KUBECONFIG=/etc/kubernetes/admin.conf' >> $HOME/.bashrc

* Join Worker Nodes to the Kubernetes Cluster (Run on worker node)

1. Run ‘kubadm join’ command.

* Testing the Kubernetes Cluster(Run on Master)

1. $ kubectl get nodes

2. $ kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

3. $ kubectl get nodes

4. kubectl get pods -n kube-system

* Run following commands if ‘kubectl apply’ fails.

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

**Implementation**

# Namespace:

## What is Namespace:

* Namespace is a virtual cluster inside a Kubernetes cluster. It allows you to partition and isolate resources within the same physical cluster. Namespaces are a way to divide cluster resources between multiple users, teams, or projects (e.g., Deployments, Services, etc.) and not for cluster-wide objects (e.g., Storage Class, Nodes, Persistent Volumes, etc.).

## Creating Namespaces

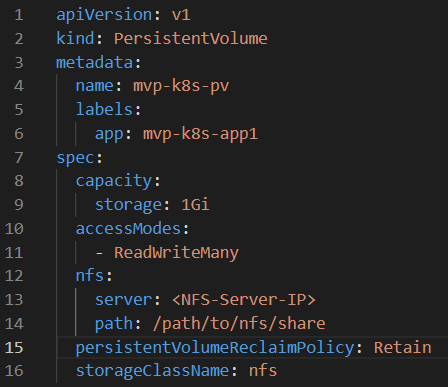
* kubectl create namespace <Mvp-DemoProject-K8s-Namespace-Dev>
* kubectl create namespace < Mvp-DemoProject-K8s-Namespace-Test >

# PV (Persistent Volume) and PVC (Persistent Volume Claim):

## Persistent Volume (PV):

* A Persistent Volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster, just like a node, that can be used to store data. PVs are typically used to represent physical storage resources in a cluster. These resources could be a network disk, a local disk, or a cloud storage system. PVs have a lifecycle independent of any individual pod using the PV.

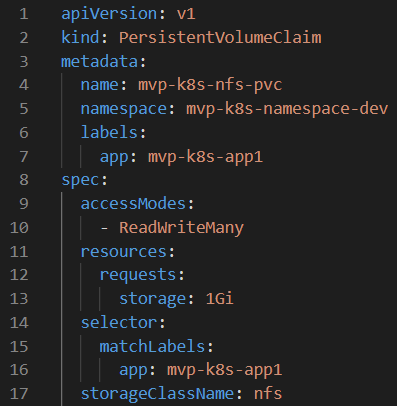
## Creating a Persistent Volume (PV)



## Persistent Volume Claim (PVC):

* A Persistent Volume Claim (PVC) is a request for storage by a user. It is used by pods to request a specific amount of storage from a PV. PVCs act as a request for storage resources. When a pod wants to use persistent storage, it makes a PVC specifying the desired storage size and access mode. The cluster then finds an appropriate PV and binds the PVC to it. Creating a Persistent Volume Claim (PVC)

## Creating a Persistent Volume Claim (PVC)



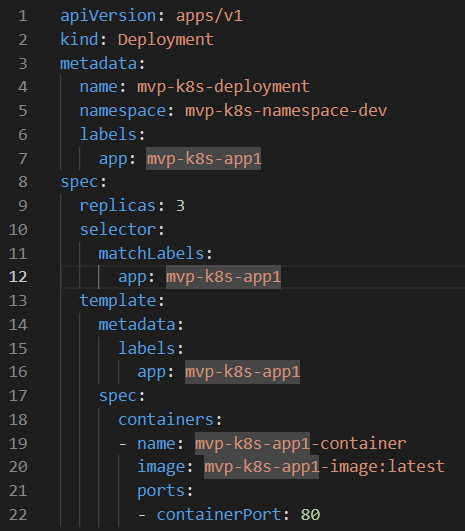
## Pod:

* Pod is the smallest deployable unit that represents a single instance of a running process in a cluster. A Pod can contain one or multiple containers, which share the same network namespace, storage, and configuration, and can communicate with each other.

## Deployment:

* Deployment is an object that provides declarative updates to applications. It allows you to describe an application's desired state, such as the number of desired replicas, and handles the deployment and scaling of the specified Pods. Deployments are a higher-level abstraction that simplifies the management and updates of applications in a Kubernetes cluster.

## Creating Deployment



To run Deployment file, execute the command below:

$ kubectl apply -f Mvp-DemoProject-K8s-Deployment-Pod-Frontend.yml

## Service

* Service is an abstraction that defines a logical set of Pods and a policy by which to access them. It enables communication and networking between different parts of an application, allowing them to work together even if they are running in different containers, on different nodes, or in different Pods.
* In Kubernetes, there are several types of Services, which define how networking is configured to enable access to the pods within the cluster. These Service types determine the rules for how external or internal clients can discover and communicate with the pods.

## The main types of Services in Kubernetes :

## ClusterIP:

`

* Exposes the Service on an internal IP within the cluster. It's only accessible from within the Kubernetes cluster.
* This is the default type if no type is specified.

## NodePort:

* Exposes the Service on a static port on each node's IP.
* Makes the Service accessible from outside the cluster using `<NodeIP>:<NodePort>`.

## LoadBalancer:

* Exposes the Service externally using a cloud provider's load balancer.
* It assigns an external IP to the Service, and the cloud provider's load balancer distributes incoming traffic to the Service.

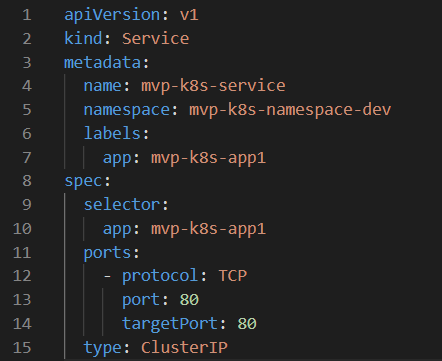
## ExternalName:

* Maps the Service to a DNS name rather than routing traffic through the cluster. It's used for integrating with external services by providing an alias to them.

## Headless Service:

* Doesn't assign a cluster IP to the service and instead provides DNS entries for the individual pods backing the service.

## Creating Service



## ReplicaSet:

* In Kubernetes, a ReplicaSet is a higher-level abstraction built on top of Pods. It ensures that a specified number of pod replicas are always running in your cluster. If Pods created by a ReplicaSet fail or are deleted, the ReplicaSet automatically replaces them to maintain the desired number of replicas.

## Desired Replicas:

* You define the desired number of replicas in the ReplicaSet's configuration. For example, if you specify that you want 3 replicas, the ReplicaSet will ensure that there are always 3 pods running.

## Selector:

* ReplicaSets use label selectors to identify the Pods they are supposed to manage. The ReplicaSet's selector field defines how to select the Pods it should manage. Pods created or modified with matching labels are considered part of the ReplicaSet.

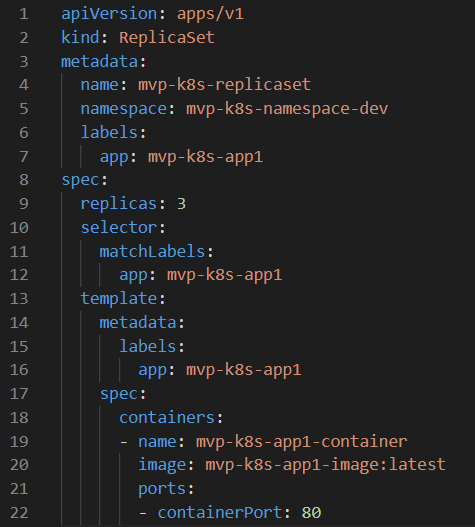
## Pod Template:

* A ReplicaSet uses a pod template to create new Pods when necessary. The pod template defines the specifications for the Pods, such as the container images, volumes, and environment variables. When a ReplicaSet needs to create new Pods, it uses the pod template as a blueprint.

## Self-Healing:

* One of the primary purposes of ReplicaSets is to provide high availability. If a Pod managed by a ReplicaSet crashes or is deleted, the ReplicaSet controller notices the change and creates a new Pod to replace it, ensuring that the desired number of replicas is always maintained.

## Creating ReplicaSet



## StatefulSets:

* StatefulSets is a higher-level abstraction in Kubernetes used to manage stateful applications. Unlike ReplicaSets or Deployments, StatefulSets provide guarantees about the ordering and uniqueness of pods. StatefulSets are particularly useful for applications that require stable network identities, stable storage, or ordered pod deployment and scaling.

## Stable Network Identifiers:

* StatefulSets give each pod a stable hostname based on its ordinal index and a stable network identity in the form of a DNS name. For example, if you have a StatefulSet named `web`, the pods will have hostnames like `web-0`, `web-1`, and so on. This stability is crucial for applications that rely on stable network identifiers.

## Stable Storage:

* StatefulSets can use Persistent Volumes (PVs) to provide stable storage for pods. When a StatefulSet is deleted, the PVs are not deleted by default, preserving the data. This is beneficial for databases and other stateful applications that need persistent storage.

## Ordered Deployment and Scaling:

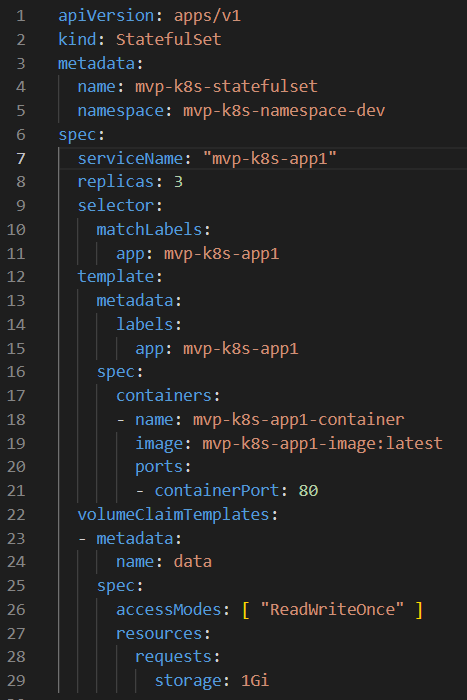
* StatefulSets deploy pods in order, starting with index 0 and incrementing sequentially. This ordered deployment and scaling can be important for applications that rely on a specific order during startup or shutdown processes.

## Headless Service:

* StatefulSets automatically create a Headless Service, which means that the service does not load balance traffic across pods. Instead, it allows direct communication with individual pods using their stable network identities.

`

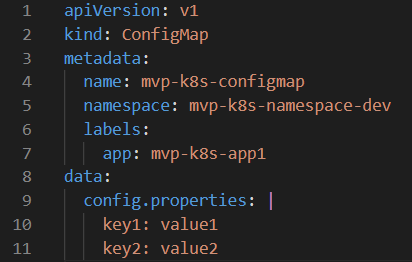
## Creating StatefulSet



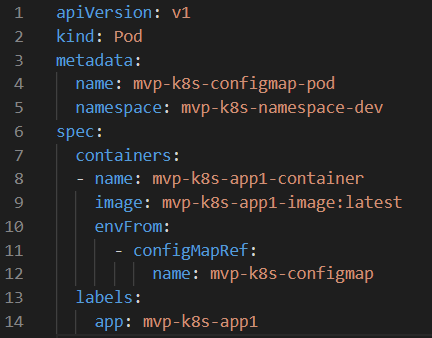
# ConfigMaps:

* ConfigMaps are used to store non-sensitive configuration data in key-value pairs. This data can be used by Pods and containers in a Kubernetes cluster. ConfigMaps are particularly useful for storing configuration files, environment variables, or any other configuration data that your application needs.

## Creating ConfigMap



# Using ConfigMap in a Pod:



# Secrets:

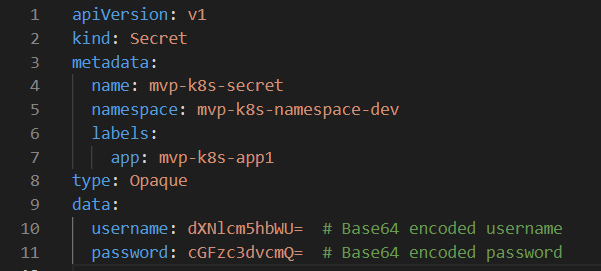
Secrets are used to store sensitive information, such as passwords, API keys, and certificates. Secrets are base64 encoded, but they are not encrypted, so it's essential to control access to secrets in your cluster.

## Creating a Secret

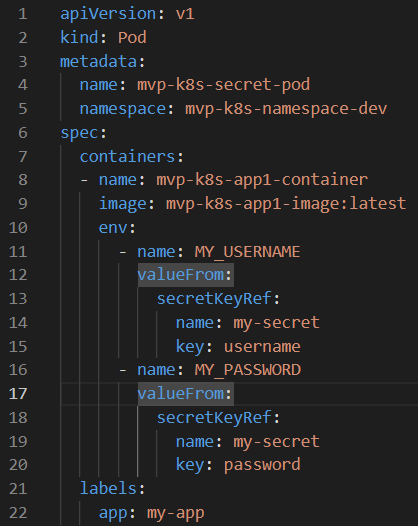
for encoding the required secret

syntax:

$ echo -n "secret" | base64 -i -  
example:  
$ echo -n "username" | base64 -i -  
$ echo -n "password" | base64 -i –



## Using Secret in a Pod:



## Helm

* Helm is a package manager for Kubernetes, designed to simplify the deployment and management of applications and services on Kubernetes clusters. It helps in defining, installing, and upgrading even the most complex Kubernetes applications.

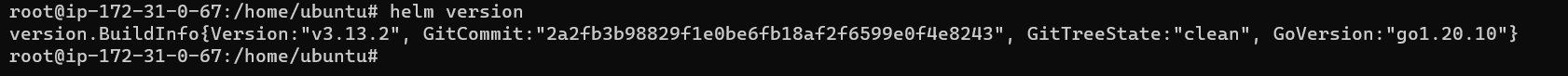
## Prerequisites

The following prerequisites are required for a successful and properly secured use of Helm.

* A Kubernetes cluster.
* Deciding what security configurations to apply to your installation, if any
* Installing and configuring Helm.

## Helm Setup

* curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
* sudo apt-get install apt-transport-https --yes
* echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] 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



## To create helm chart

* helm create <name>



* We give the values in values.yaml

A screenshot of a computer screen

Description automatically generated

* In templates directory we have the following files

A screen shot of a computer screen

Description automatically generated

* Edit the above files according to required configuration.

## To apply helm chart

* helm install <name of helm file> <helm chart name>

## Helm file:

* To run helm chart using helm file
* Enter the below command.

$ sudo helm sync

## Ingress:

* In Kubernetes (k8s), "Ingress" is an API object that manages external access to services within a Kubernetes cluster. It provides a way to manage HTTP and HTTPS routing from outside the cluster to services running inside the cluster.
* To use the ingress service, we have to install an ingress controller.

## How to install Nginx Ingress Controller:

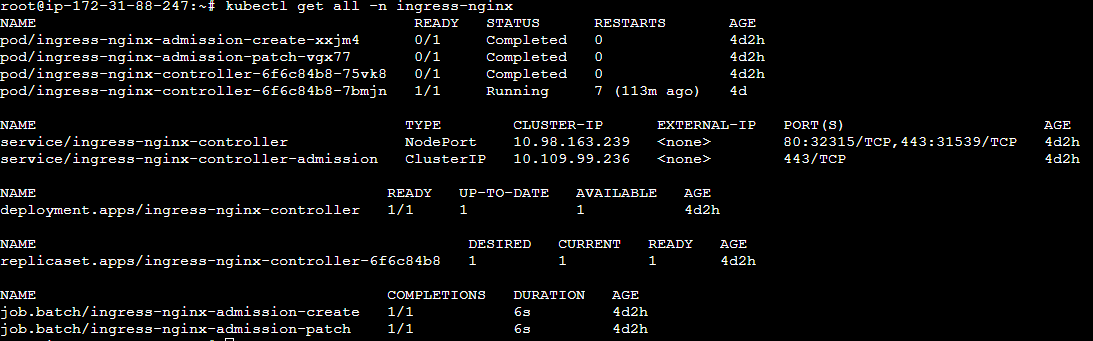
* As per the requirement we have to install ingress controller on a bare metal

## To install ingress controller

* Run below command on cluster.

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.8.2/deploy/static/provider/baremetal/deploy.yaml>

* while we execute this command it will create a separate namespace i.e. ingress-nginx



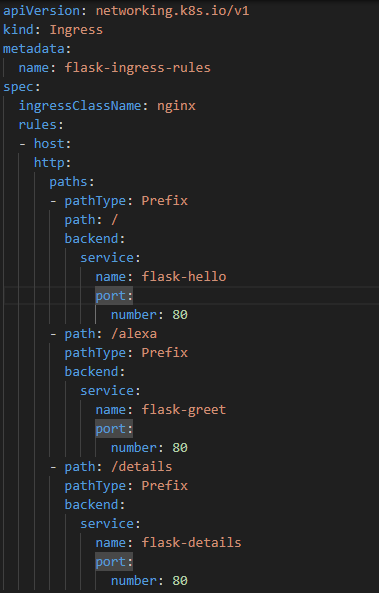
# Sample Flow Chart for Ingress

A diagram of a diagram

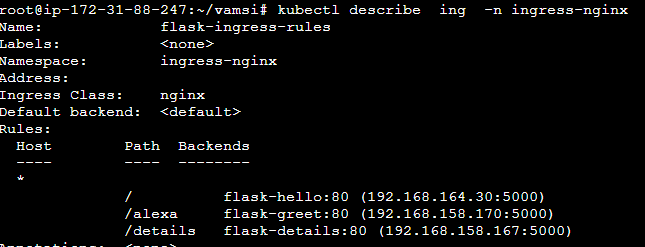
Description automatically generated

## Configuring ingress rules for path-based routing:

* Manifest file for ingress



## Routing rules for ingress path-based routing:

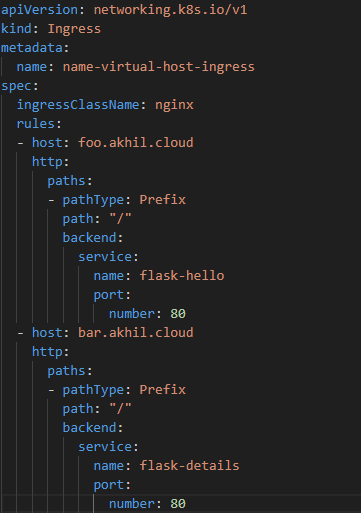


## How to access application through path based:

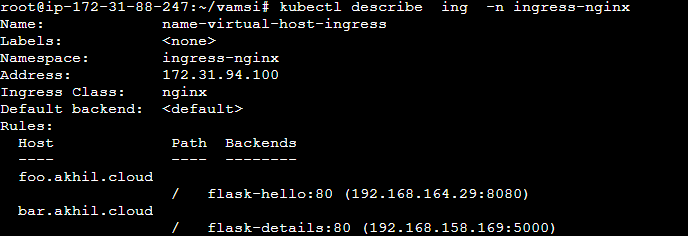
* <public Ip of an instance>:NodePort/path\_to\_route

## Name based virtual hosting in ingress:

Manifest file for name based virtual hosting.



## Routing rules for ingress name-based routing:



## Create a new A record for name based virtual hosting:

A screenshot of a computer

Description automatically generated

# Kubernetes Cluster Monitoring using Prometheus and Grafana

## What is Prometheus:

Prometheus is an open-source monitoring and alerting system. It has multiple tool components, including the main Prometheus server, which scrapes and stores time series data. client libraries that instrument the application code. a push gateway for handling short-term tasks. special-purpose exporters are provided for services like HAProxy, StatsD, Graphite, etc. alert-manager to handle alerts. various support tools. Graphite or other API consumers can be used to visualize the collected data.

## What is Grafana:

Grafana is an open-source analytics and monitoring platform that integrates with various data sources to visualize and analyze metrics, logs, and other data. It provides a flexible and customizable dashboarding solution, making it popular for monitoring and observability in the fields of infrastructure, application performance, and business analytics.

## Installation of Prometheus and Grafana in Kubernetes-Cluster using helm:

## Prerequisites:

**The following prerequisites are required for a successful and properly secured use of the Helm.**

* A Kubernetes Cluster
* Helm Package manager

## Creating a Namespace:

* Create a Namespace named "monitoring" in Kubernetes for best practice and isolation.
* $ kubectl create namespace monitoring

## Installation of Prometheus & Grafana using helm charts:

* **Add helm repo of Prometheus**.
  + $ helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>
* **Update the repository.**
* $ helm repo update
* **Install prometheus stack.**
* $ helm install prometheus prometheus-community/kube-prometheus-stack -n monitoring
* **Once the Installation done check all the prometheus pods are running fine or not using the following command**
* $ kubectl get all -n monitoring

A black background with many letters

Description automatically generated with medium confidence

* **Expose NodePort services for Prometheus and Grafana pods to access outside of the cluster, by default Prometheus run on 9090 and Grafana runs on 3000 ports.**
* $ kubectl expose service <service-name> --type=NodePort --name=<service-name> --target-port=3000 -n <namespace>



* Now we can access the “Prometheus Server” using cluster prometheus-publicIpaddress:NodePort

A screenshot of a computer

Description automatically generated

* We can access “Grafana Dashboard” using cluster grafana-publicIpaddress: NodePort

A screenshot of a computer

Description automatically generated

* **Log in using the default credentials (admin/admin) or the credentials you set during the installation. (**Default credentials may work or not, vary with the versions**)**
* **To change the credentials**
* Get into the Grafana pod to change the default Grafana credentials using the following command.
* $ kubectl exec -it <grafana-pod-name> -n <namespace> -- /bin/bash
* After getting into the pod run the below command by entering the new-password.
* grafana-cli admin reset-admin-password <new-password>
* **Add Prometheus Data Source:**
  + In the Grafana interface, click on the Toggle menu icon () on the left sidebar to click on the Connections > Add New Connection

A screenshot of a computer

Description automatically generated

* + In the “search all” type “Prometheus” click on Prometheus and click on “Add new data source”

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* **Configure Prometheus Connection:**
  + In the "HTTP" section, set the Prometheus server URL in the "URL" field. This is the address where your Prometheus server is running. It might look like <http://prometheus-server:port>

A screenshot of a computer

Description automatically generated

* **Save and Test:**
* Scroll down and click the "Save & Test" button to verify the connection to Prometheus. Grafana will display a success message if the connection is established successfully.

A screenshot of a computer

Description automatically generated

* After adding the data source, you can go back to the Grafana dashboard.
* Now that Prometheus is configured as a data source, you can create dashboards and panels that query data from Prometheus.

## Grafana Dashboard creation for visual representation of Kubernetes Cluster Metrics:

**Steps to Create the Dashboard:**

* Open any web browser and navigate to your Grafana instance.
* Log in with your credentials.
* After logging in, you’ll typically see a sidebar on the left. Click on the Toggle menu icon (). Then click on the Dashboards.

A screenshot of a computer

Description automatically generated

* Click on the “New” you’ll see three options “New Dashboard”, “New folder”, and “Import”.

A black rectangular object with a black line

Description automatically generated

A screenshot of a computer

Description automatically generated

* **“New Dashboard**” is to customize each panel as required by setting up necessary PromQL expressions and visualization patterns.
* **“New Folder”** is to create a folder which serves as a way to organize and group dashboards.
* **“Import”** allows users to bring external dashboards configurations into their grafana dashboards from official grafana website.
* We can Import an external dashboard using URL, or ID, or Json files from the official grafana website or existing pre-configured dashboard in Json format.

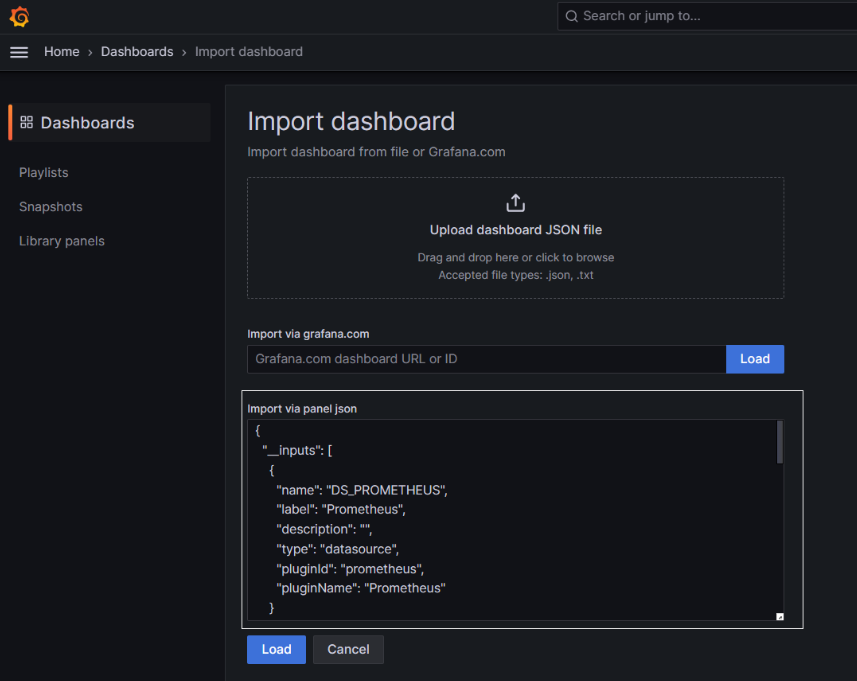
A screenshot of a computer

Description automatically generated

We have imported configured a dashboard with required panels and exported it in Json file. To import the dashboard, go to the GitHub repository link provided below.

* GitHub repo: [mvp-k8s/grafana-custome-dashboard/Kubernetes Cluster (Prometheus)-1701673775999.json at main · vishnuallampalli/mvp-k8s (github.com)](https://github.com/vishnuallampalli/mvp-k8s/blob/main/grafana-custome-dashboard/Kubernetes%20Cluster%20(Prometheus)-1701673775999.json)
* Go to the git repository link provided copy the json file.
* Paste the copied json file in the Import via panel json

+



* After pasting the Json file like shown above click on “Load”

A screenshot of a computer

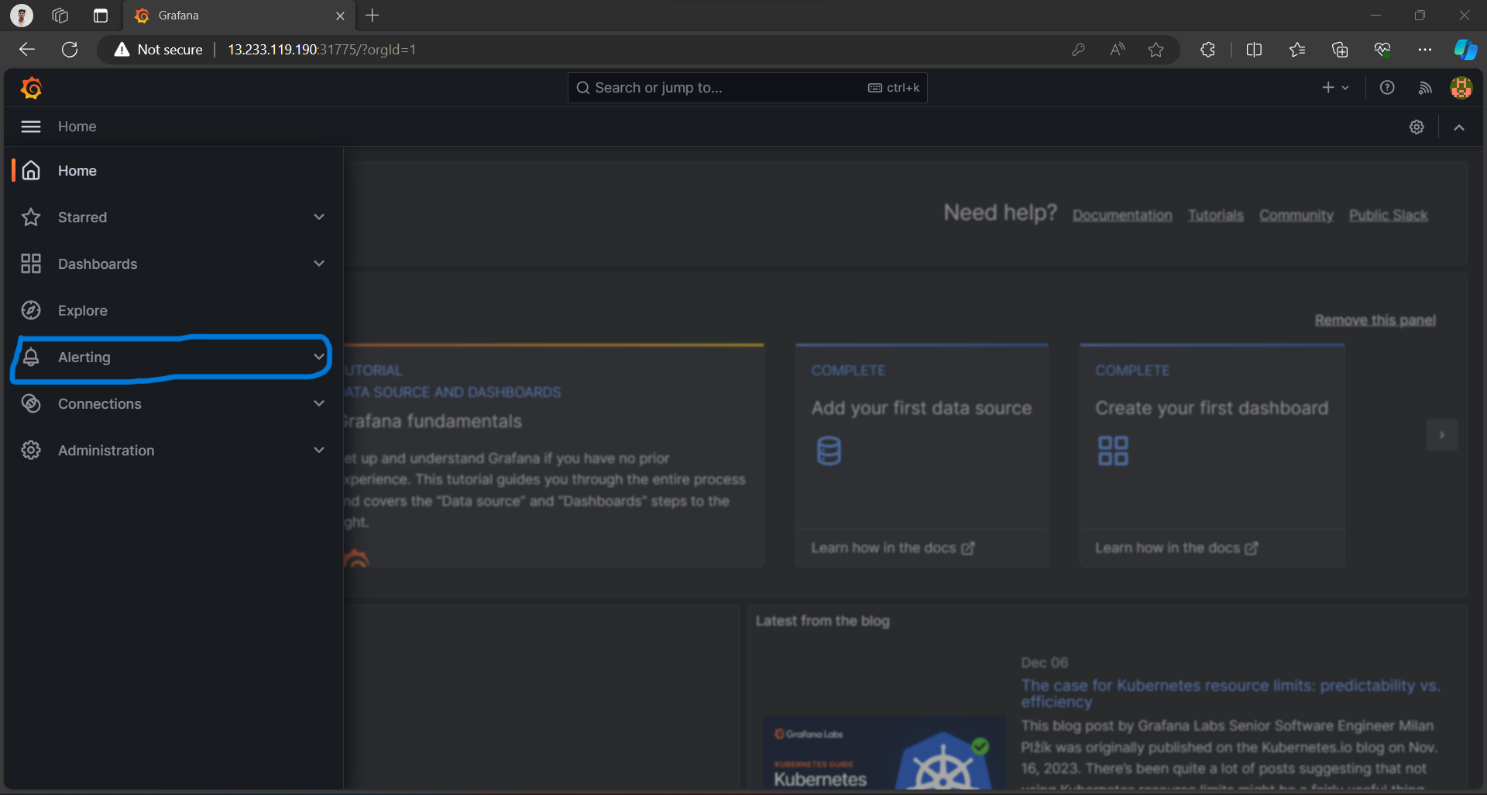
Description automatically generated

* In the options Name section Provide your dashboard name like “Kubernetes Cluster Monitoring”.
* Select the folder in which you want to organize your dashboard.
* Every Grafana dashboard is assigned a unique UID. This UID is used to reference the dashboard internally within Grafana. When you share a dashboard or when Grafana saves its configuration, it is the UID that is used to uniquely identify and retrieve that specific dashboard.
* In the UID section provide a UID generally it will come by default while importing. Change it if required.
* Select Prometheus data source in the last section and click on “Import”
* Now the imported dashboard looks like below.
* A screenshot of a computer

  Description automatically generated

## Setup alert manager on grafana:

* Go to Grafana dashboard and go to the left side menu and click on alerts.

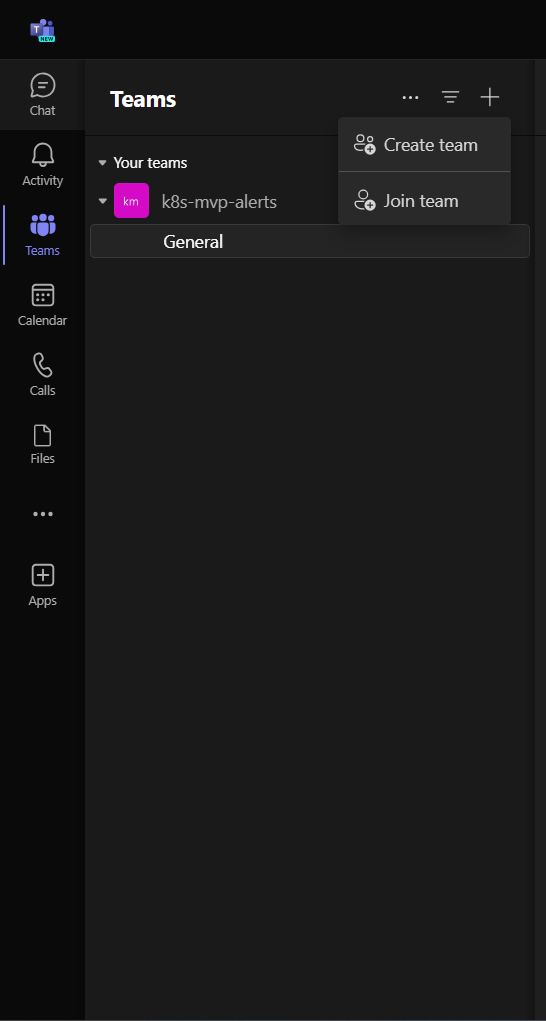


* Create contact points by clicking the contact points to where we want those alerts.

A screenshot of a computer

Description automatically generated

* Go to teams and follow the below steps to create a team.



* Select from scratch.

A screenshot of a computer

Description automatically generated

* Select public.

A screenshot of a computer

Description automatically generated

* Enter name and create.

A screenshot of a computer

Description automatically generated

* Adding members to the team.

A screenshot of a computer

Description automatically generated

* Now go to app and search for incoming webhook

A screenshot of a computer

Description automatically generated

* Then add the incoming webhook by clicking add to a team.

A screenshot of a computer

Description automatically generated

* Now select the team that created and set up a connector.

A screenshot of a computer

Description automatically generated

* Now we get the URL copy that URL and click on done.

A screenshot of a computer

Description automatically generated

* After clicking on done we can see the below image. That webhook is configured to the team.

A screenshot of a computer

Description automatically generated

* Now go back to Grafana dashboard and go to the contact points and paste the webhook URL that looks below.

A screenshot of a computer

Description automatically generated

* Go to the notifications policies and follow the below image and edit the notification policies.

A screenshot of a computer

Description automatically generated

* Select the mvp-alert and update the policy.

A screenshot of a computer

Description automatically generated

* Now we can configure the alert rules by clicking on new alert rule like following images.

A screenshot of a computer

Description automatically generated

* This is the rule for CPU usage in the following image.

A screenshot of a computer program

Description automatically generated

* This is the rule for disk usage in the following image.

A screenshot of a computer

Description automatically generated

* This is the rule for node-down alert in the following image.

A screenshot of a computer

Description automatically generated

* Now select the folder, group and give the pending period of an alert.

A screenshot of a computer

Description automatically generated

* Now we can get the alerts on teams like below.

A screenshot of a computer

Description automatically generated