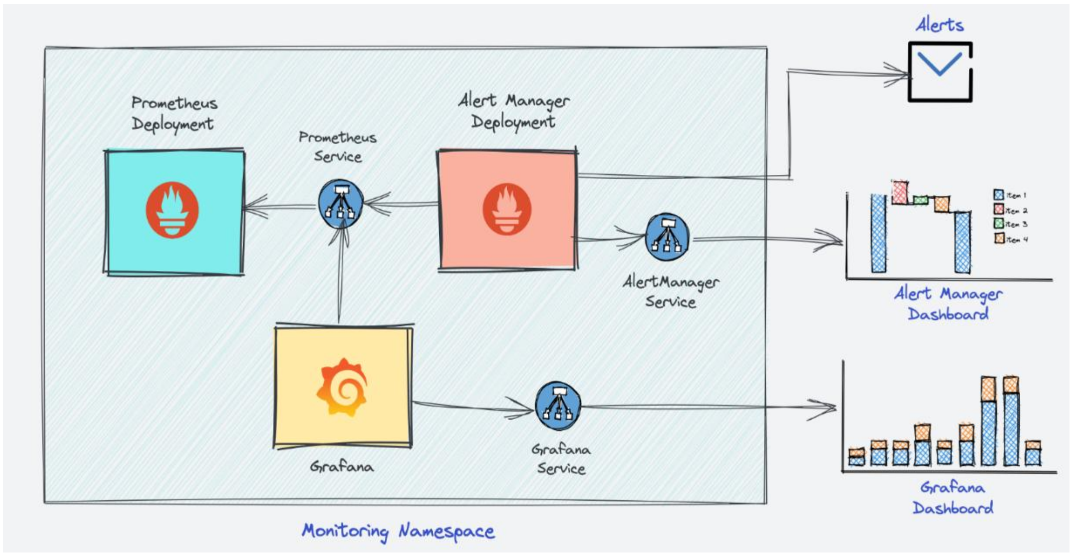
Setup Prometheus Monitoring On Kubernetes Cluster

Prometheus is a high-scalable open-source monitoring framework. It provides out-of-the-box monitoring capabilities for the Kubernetes container orchestration platform. Also, In the observability space, it is gaining huge popularity as it helps with metrics and alerts.

Prometheus Architecture: - The Kubernetes Prometheus monitoring stack has the following components.

1. Prometheus Server
2. Alert Manager
3. Grafana



Step 1: clone the git for Prometheus Kubernetes Manifest Files

```
$ git clone https://github.com/techiescamp/kubernetes-prometheus
```

Step 2: Create a Namespace & Cluster Role

```
$ kubectl create namespace monitoring
```

Step 3: Create a file named clusterRole.yaml and copy the following RBAC role.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: default
  namespace: monitoring
```

Step 4: Create the role using the following command.

```
$ kubectl create -f clusterRole.yaml
```

Step 5: create a config map to externalize Prometheus configurations

All configurations for Prometheus are part of `prometheus.yaml` file and all the alert rules for Alert manager are configured in `prometheus.rules`.

`prometheus.yaml`: This is the main Prometheus configuration which holds all the scrape configs, service discovery details, storage locations, data retention configs, etc)

`prometheus.rules`: This file contains all the Prometheus alerting rules

Step 6: Create a file called `config-map.yaml`

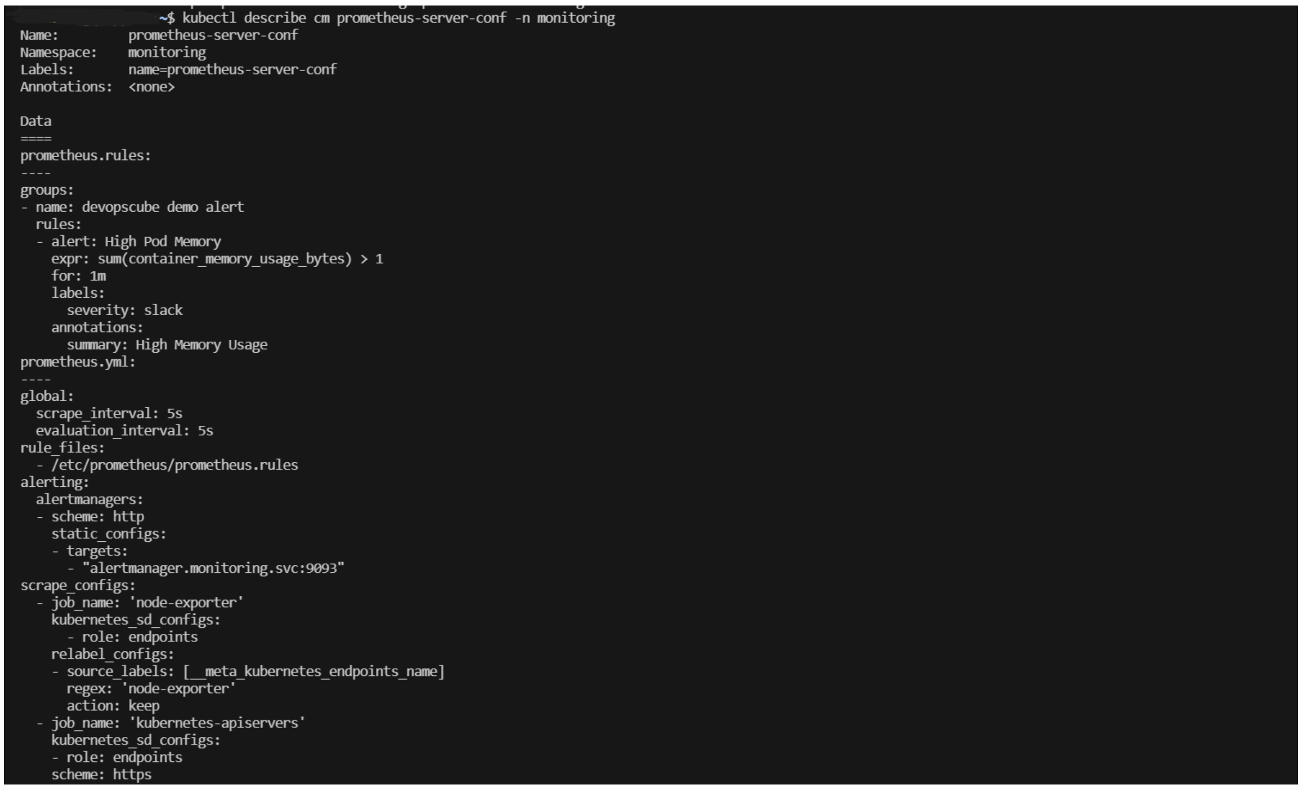
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-server-conf
  labels:
    name: prometheus-server-conf
  namespace: monitoring
data:
  prometheus.rules: |-
    groups:
    - name: devopscube demo alert
      rules:
      - alert: High Pod Memory
        expr: sum(container_memory_usage_bytes) > 1
        for: 1m
        labels:
          severity: slack
        annotations:
          summary: High Memory Usage
  prometheus.yml: |-
    global:
      scrape_interval: 5s
      evaluation_interval: 5s
    rule_files:
      - /etc/prometheus/prometheus.rules
    alerting:
      alertmanagers:
      - scheme: http
        static_configs:
        - targets:
          - "alertmanager.monitoring.svc:9093"
    scrape_configs:
      - job_name: 'node-exporter'
        kubernetes_sd_configs:
          - role: endpoints
        relabel_configs:
        - source_labels: [__meta_kubernetes_endpoints_name]
          regex: 'node-exporter'
          action: keep
      - job_name: 'kubernetes-apiservers'
        kubernetes_sd_configs:
        - role: endpoints
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
        relabel_configs:
        - source_labels: [__meta_kubernetes_namespace,
__meta_kubernetes_service_name, __meta_kubernetes_endpoint_port_name]
          action: keep
          regex: default;kubernetes;https
      - job_name: 'kubernetes-nodes'
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
        kubernetes_sd_configs:
        - role: node
        relabel_configs:
        - action: labelmap
          regex: __meta_kubernetes_node_label_(.+)
        - target_label: __address__
          replacement: kubernetes.default.svc:443
        - source_labels: [__meta_kubernetes_node_name]
          regex: (.+)
          target_label: __metrics_path__
          replacement: /api/v1/nodes/${1}/proxy/metrics
      - job_name: 'kubernetes-pods'
        kubernetes_sd_configs:
        - role: pod
        relabel_configs:
        - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
          action: keep
          regex: true
        - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
          action: replace
          target_label: __metrics_path__
          regex: (.+)
        - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
          action: replace
          regex: ([^:]+)(?::\d+)?;(\d+)
          replacement: $1:$2
          target_label: __address__
        - action: labelmap
          regex: __meta_kubernetes_pod_label_(.+)
        - source_labels: [__meta_kubernetes_namespace]
          action: replace
          target_label: kubernetes_namespace
        - source_labels: [__meta_kubernetes_pod_name]
          action: replace
          target_label: kubernetes_pod_name
      - job_name: 'kube-state-metrics'
        static_configs:
          - targets: ['kube-state-metrics.kube-system.svc.cluster.local:8080']
      - job_name: 'kubernetes-cadvisor'
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
        kubernetes_sd_configs:
        - role: node
        relabel_configs:
        - action: labelmap
          regex: __meta_kubernetes_node_label_(.+)
        - target_label: __address__
          replacement: kubernetes.default.svc:443
        - source_labels: [__meta_kubernetes_node_name]
          regex: (.+)
          target_label: __metrics_path__
          replacement: /api/v1/nodes/${1}/proxy/metrics/cadvisor
      - job_name: 'kubernetes-service-endpoints'
        kubernetes_sd_configs:
        - role: endpoints
        relabel_configs:
        - source_labels: 
[__meta_kubernetes_service_annotation_prometheus_io_scrape]
          action: keep
          regex: true
        - source_labels: 
[__meta_kubernetes_service_annotation_prometheus_io_scheme]
          action: replace
          target_label: __scheme__
          regex: (https?)
        - source_labels: 
[__meta_kubernetes_service_annotation_prometheus_io_path]
          action: replace
          target_label: __metrics_path__
          regex: (.+)
        - source_labels: [__address__, 
__meta_kubernetes_service_annotation_prometheus_io_port]
          action: replace
          target_label: __address__
          regex: ([^:]+)(?::\d+)?;(\d+)
          replacement: $1:$2
        - action: labelmap
          regex: __meta_kubernetes_service_label_(.+)
        - source_labels: [__meta_kubernetes_namespace]
          action: replace
          target_label: kubernetes_namespace
        - source_labels: [__meta_kubernetes_service_name]
          action: replace
          target_label: kubernetes_name
```

Step 7: Execute the following command to create the config map in Kubernetes.

```
$ kubectl create -f config-map.yaml
```

Step 8: Check wheather it is created or not

```
$ kubectl describe cm prometheus-server-conf –n monitoring
```



```
[redacted]~$ kubectl describe cm prometheus-server-conf -n monitoring
Name:         prometheus-server-conf
Namespace:    monitoring
Labels:       name=prometheus-server-conf
Annotations:  <none>

Data
====
prometheus.rules:
----
groups:
- name: devopscube demo alert
  rules:
  - alert: High Pod Memory
    expr: sum(container_memory_usage_bytes) > 1
    for: 1m
    labels:
      severity: slack
    annotations:
      summary: High Memory Usage
prometheus.yml:
----
global:
  scrape_interval: 5s
  evaluation_interval: 5s
rule_files:
  - /etc/prometheus/prometheus.rules
alerting:
  alertmanagers:
  - scheme: http
    static_configs:
    - targets:
      - "alertmanager.monitoring.svc:9093"
scrape_configs:
  - job_name: 'node-exporter'
    kubernetes_sd_configs:
      - role: endpoints
    relabel_configs:
    - source_labels: [__meta_kubernetes_endpoints_name]
      regex: 'node-exporter'
      action: keep
  - job_name: 'kubernetes-apiservers'
    kubernetes_sd_configs:
    - role: endpoints
    scheme: https
```

Step 9: Create a file named **`prometheus-deployment.yaml`** and copy the following contents onto the file.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus-deployment
  namespace: monitoring
  labels:
    app: prometheus-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-server
  template:
    metadata:
      labels:
        app: prometheus-server
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus
          args:
            - "--storage.tsdb.retention.time=12h"
            - "--config.file=/etc/prometheus/prometheus.yml"
            - "--storage.tsdb.path=/prometheus/"
          ports:
            - containerPort: 9090
          resources:
            requests:
              cpu: 500m
              memory: 500M
            limits:
              cpu: 1
              memory: 1Gi
          volumeMounts:
            - name: prometheus-config-volume
              mountPath: /etc/prometheus/
            - name: prometheus-storage-volume
              mountPath: /prometheus/
      volumes:
        - name: prometheus-config-volume
          configMap:
            defaultMode: 420
            name: prometheus-server-conf

        - name: prometheus-storage-volume
          emptyDir: {}
```

Step 10: Create a deployment on monitoring namespace using the above file.

```
$ kubectl create -f prometheus-deployment.yaml
```

Step 11: You can check the created deployment using the following command.

```
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
Prometheus-deployment   1/1     1            1           5d20h
```

Step 12: Create a file named **`prometheus-service.yaml`** and copy the following contents. We will expose Prometheus on all kubernetes node IP's on port `30000`.

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus-service
  namespace: monitoring
  annotations:
      prometheus.io/scrape: 'true'
      prometheus.io/port:   '9090'
spec:
  selector:
    app: prometheus-server
  type: NodePort
  ports:
    - port: 8080
      targetPort: 9090
      nodePort: 30000
```

Step 13: Create the service using the following command.

```
$ kubectl create -f prometheus-service.yaml --namespace=monitoring
```

Step 14: Now we can access to use IP address and Nodeport check in the Browers

Step 15: **kube state metrics setup** and Clone the Github repo

```
$ git clone https://github.com/devopscube/kube-state-metrics-configs.git
```

Step 16: Create all the objects by pointing to the cloned directory.

```
$ kubectl apply -f kube-state-metrics-configs/
```

Step 17: Check the deployment status using the following command.

```
$ kubectl get deployments kube-state-metrics -n kube-system
```

```
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
kube-state-metrics   1/1     1            1           5d4h
```

Step 18: Setup Node Exporter on Kubernetes Create a file name **daemonset.yaml** and copy the following content.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    app.kubernetes.io/component: exporter
    app.kubernetes.io/name: node-exporter
```

```yaml
  name: node-exporter
  namespace: monitoring
spec:
  selector:
    matchLabels:
      app.kubernetes.io/component: exporter
      app.kubernetes.io/name: node-exporter
  template:
    metadata:
      labels:
        app.kubernetes.io/component: exporter
        app.kubernetes.io/name: node-exporter
    spec:
      containers:
      - args:
        - --path.sysfs=/host/sys
        - --path.rootfs=/host/root
        - --no-collector.wifi
        - --no-collector.hwmon
        - --collector.filesystem.ignored-mount-
points=^/(dev|proc|sys|var/lib/docker/.+|var/lib/kubelet/pods/.+)($|/)
        - --collector.netclass.ignored-devices=^(veth.*)$
        name: node-exporter
        image: prom/node-exporter
        ports:
          - containerPort: 9100
            protocol: TCP
        resources:
          limits:
            cpu: 250m
            memory: 180Mi
          requests:
            cpu: 102m
            memory: 180Mi
        volumeMounts:
        - mountPath: /host/sys
          mountPropagation: HostToContainer
          name: sys
          readOnly: true
        - mountPath: /host/root
          mountPropagation: HostToContainer
          name: root
          readOnly: true
      volumes:
```

```yaml
      - hostPath:
          path: /sys
        name: sys
      - hostPath:
          path: /
        name: root
```

Step 19: Deploy the daemon set using the kubectl command.

```
$ kubectl create -f daemonset.yaml
```

Step 20: List the `daemon set` in the monitoring namespace and make sure it is in the available state.

```
$ kubectl get daemonset -n monitoring
```

Step 21: Create a file names `service.yaml` and copy the following contents.

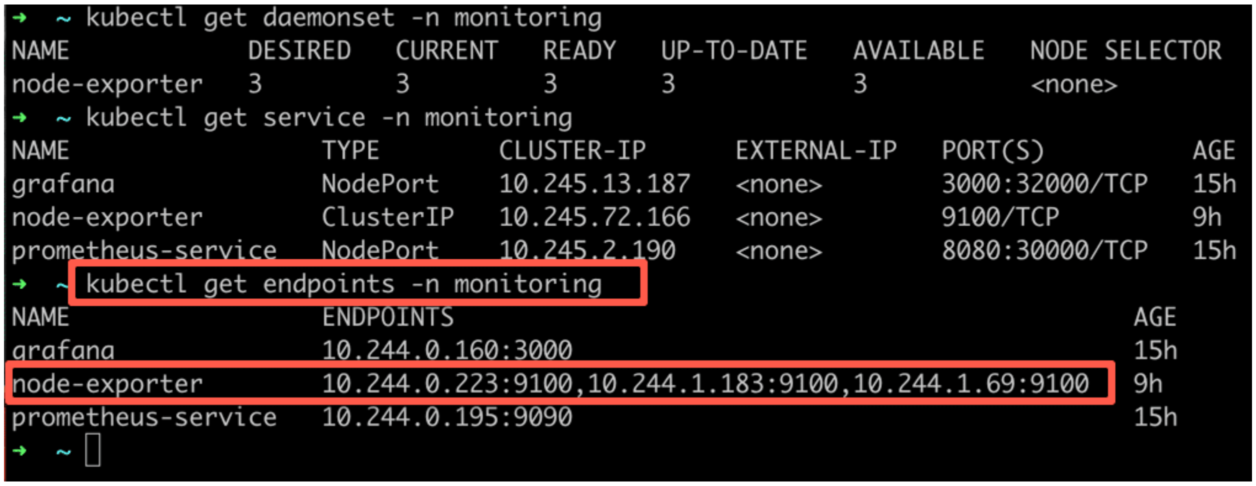
```yaml
kind: Service
apiVersion: v1
metadata:
  name: node-exporter
  namespace: monitoring
  annotations:
      prometheus.io/scrape: 'true'
      prometheus.io/port:   '9100'
spec:
  selector:
      app.kubernetes.io/component: exporter
      app.kubernetes.io/name: node-exporter
  ports:
  - name: node-exporter
    protocol: TCP
    port: 9100
    targetPort: 9100
```

Step 22: Create the service.

```
$ kubectl create -f service.yaml
```

Step 23: Now, check the service's endpoints and see if it is pointing to all the daemon set pods.

```
$ kubectl get endpoints -n monitoring
```



```
➜  ~ kubectl get daemonset -n monitoring
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
node-exporter   3         3         3       3            3           <none>
➜  ~ kubectl get service -n monitoring
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
grafana              NodePort    10.245.13.187   <none>        3000:32000/TCP   15h
node-exporter        ClusterIP   10.245.72.166   <none>        9100/TCP         9h
prometheus-service   NodePort    10.245.2.190    <none>        8080:30000/TCP   15h
➜  ~ kubectl get endpoints -n monitoring
NAME                 ENDPOINTS                                                   AGE
grafana              10.244.0.160:3000                                           15h
node-exporter        10.244.0.223:9100,10.244.1.183:9100,10.244.1.69:9100       9h
prometheus-service   10.244.0.195:9090                                           15h
➜  ~ 
```

As you can see from the above output, the node-exporter service has three endpoints. Meaning three node-exporter pods running on three nodes as part of Daemon set. Grafana

Grafana Setup on standalone server

Step 1: Install Required Packages

$ sudo apt-get install -y apt-transport-https software-propertiescommon wget

Step 2: Create Directory for GPG Key

Create a directory to store the GPG key used to authenticate the Grafana repository.

```
$ sudo mkdir -p /etc/apt/keyrings/
```

Step 3: Download and Store Grafana GPG Key

```
$ wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo tee
/etc/apt/keyrings/grafana.gpg > /dev/null
```

Step 4: **Add Grafana Stable Repository**

Add the stable Grafana repository to your system’s sources list.

```
$ echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg]
https://apt.grafana.com stable main" | sudo tee -a
/etc/apt/sources.list.d/grafana.list
```

Step 5: (Optional) Add Grafana Beta Repository

```
$ echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg]
https://apt.grafana.com beta main" | sudo tee -a
/etc/apt/sources.list.d/grafana.list
```

Step 6: Verify the Sources List

Navigate to the sources list directory and review the newly added Grafana repositories.

```
$ cd /etc/apt/sources.list.d/
```

```
sudo vim grafana.list
```

Step 7: Update Package Lists

```
$ sudo apt-get update
```

Step 8: Install Grafana

Now, install the Grafana package.

```
$ sudo apt-get install grafana
```

Step 10: Start Grafana Server

Start the Grafana server service.

```
$ sudo systemctl start grafana-server.service
```

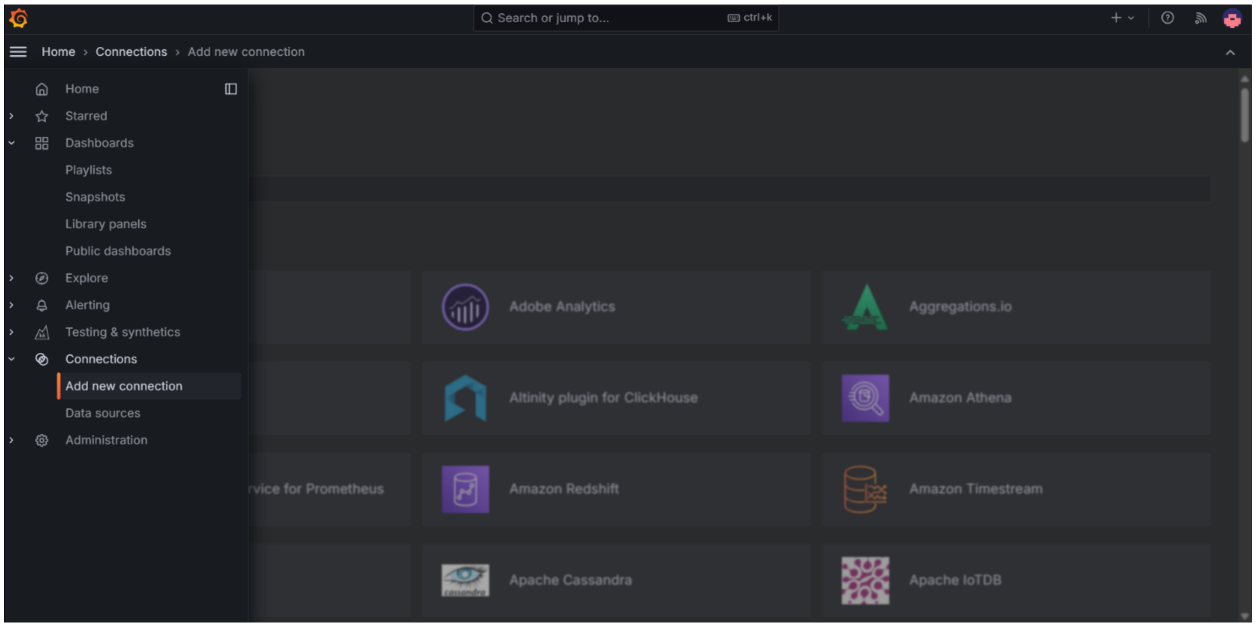
Step 11: Check Grafana Server Status

Check the status of the Grafana server to ensure it's running correctly.

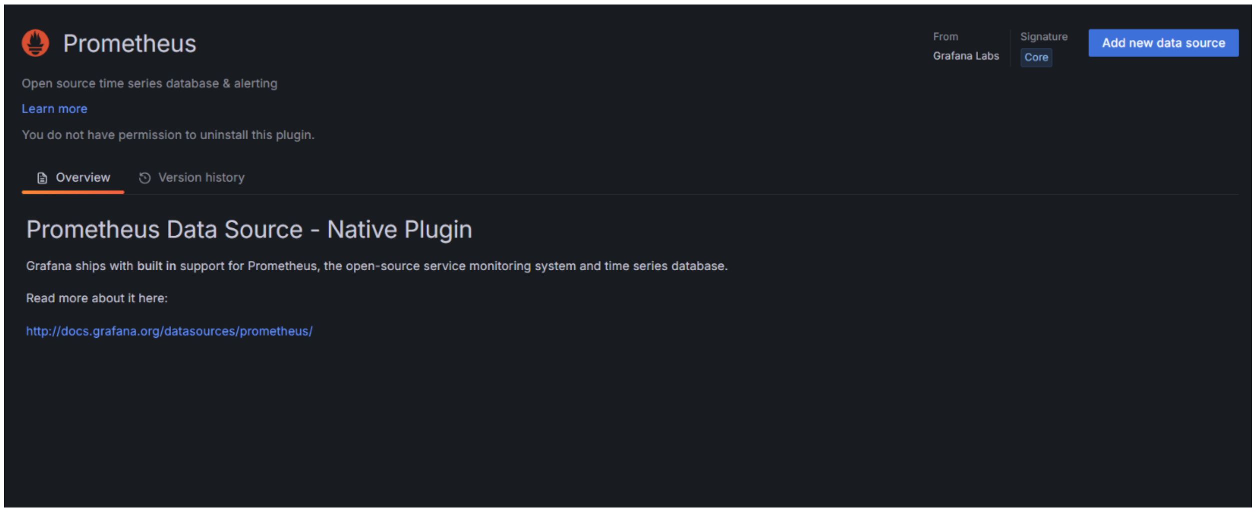
```
$ sudo systemctl status grafana-server.service
```

Step 12 : login with user and password

Step 13: add connection search the Prometheus and it



Step 14: add data source in Prometheus



Open source time series database & alerting

You do not have permission to uninstall this plugin.

Prometheus Data Source - Native Plugin

Grafana ships with built in support for Prometheus, the open-source service monitoring system and time series database.

Read more about it here:

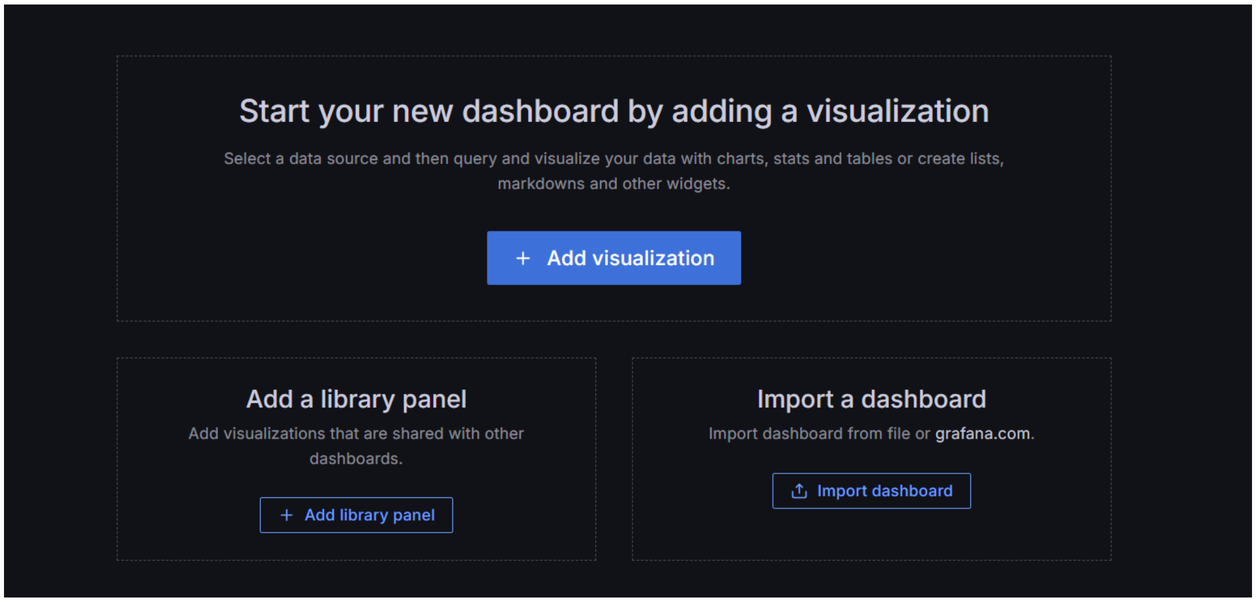
http://docs.grafana.org/datasources/prometheus/

Enter the fill details regarding Prometheus

Connection
Prometheus server URL *

Step 15: Goto dashboard add now dashboard

Step 16: go to import dashboard



Start your new dashboard by adding a visualization

Select a data source and then query and visualize your data with charts, stats and tables or create lists, markdowns and other widgets.

Add a library panel

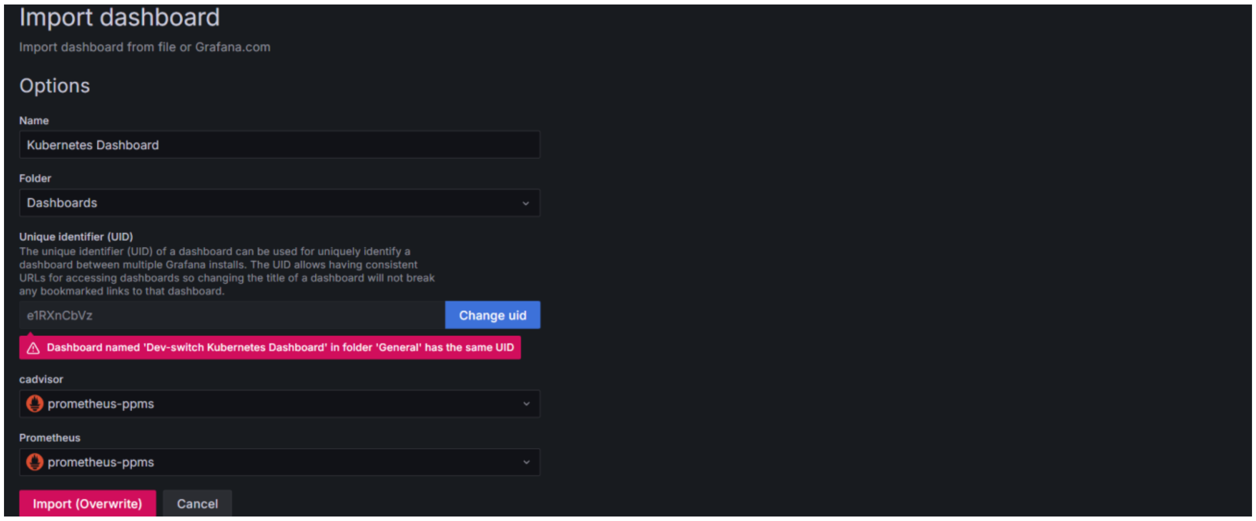
Add visualizations that are shared with other dashboards.

Import a dashboard

Import dashboard from file or grafana.com.

Step 17: In google search Grafana dashboard and download the dashboard as per requirement

Step 18: import the dashboard from download give Prometheus connection in dashboard



Import dashboard

Import dashboard from file or Grafana.com

Options

Name: Kubernetes Dashboard

Folder: Dashboards

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.

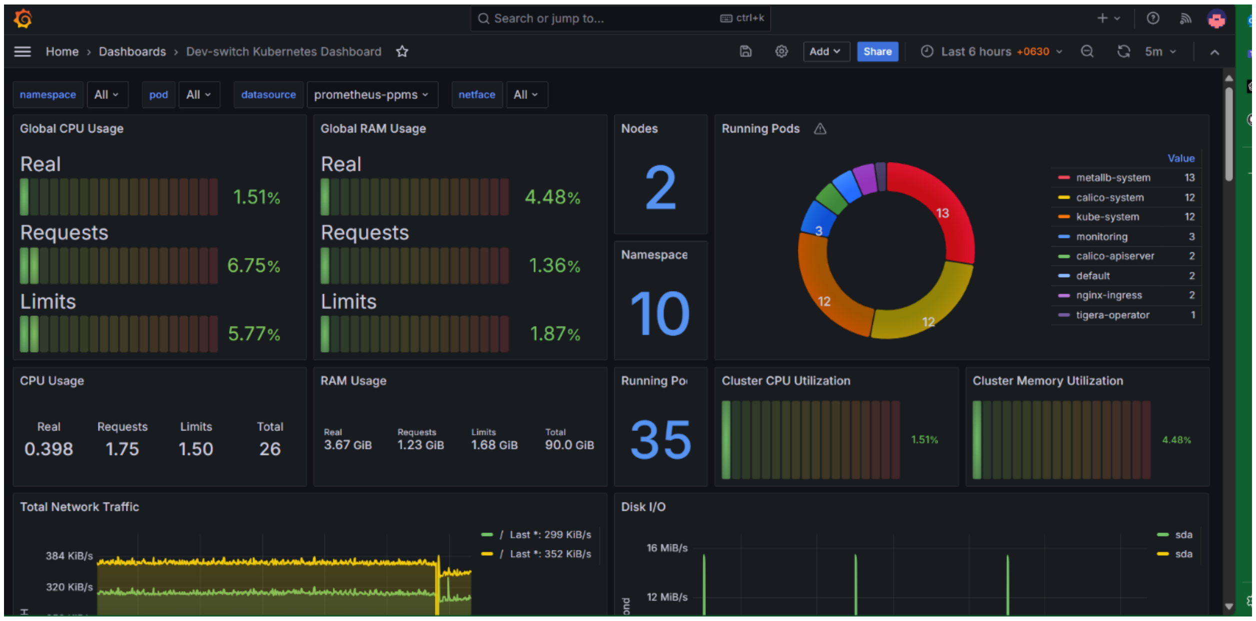
e1RXnCbVz

Dashboard named 'Dev-switch Kubernetes Dashboard' in folder 'General' has the same UID

cadvisor: prometheus-ppms

Prometheus: prometheus-ppms

Step 19:- we can see grafana dashboard



	Value
metallb-system	13
calico-system	12
kube-system	12
monitoring	3
calico-apiserver	2
default	2
nginx-ingress	2
tigera-operator	1

https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/

https://devopscube.com/setup-kube-state-metrics/

[How To Setup Prometheus Node Exporter On Kubernetes (devopscube.com)]()