Kubernetes Monitoring for Everyone: Step-by-Step with Prometheus & Grafana

🚀 A Complete Step-by-Step Guide for Beginners, Freshers, and Professionals

Press enter or click to view image in full size:

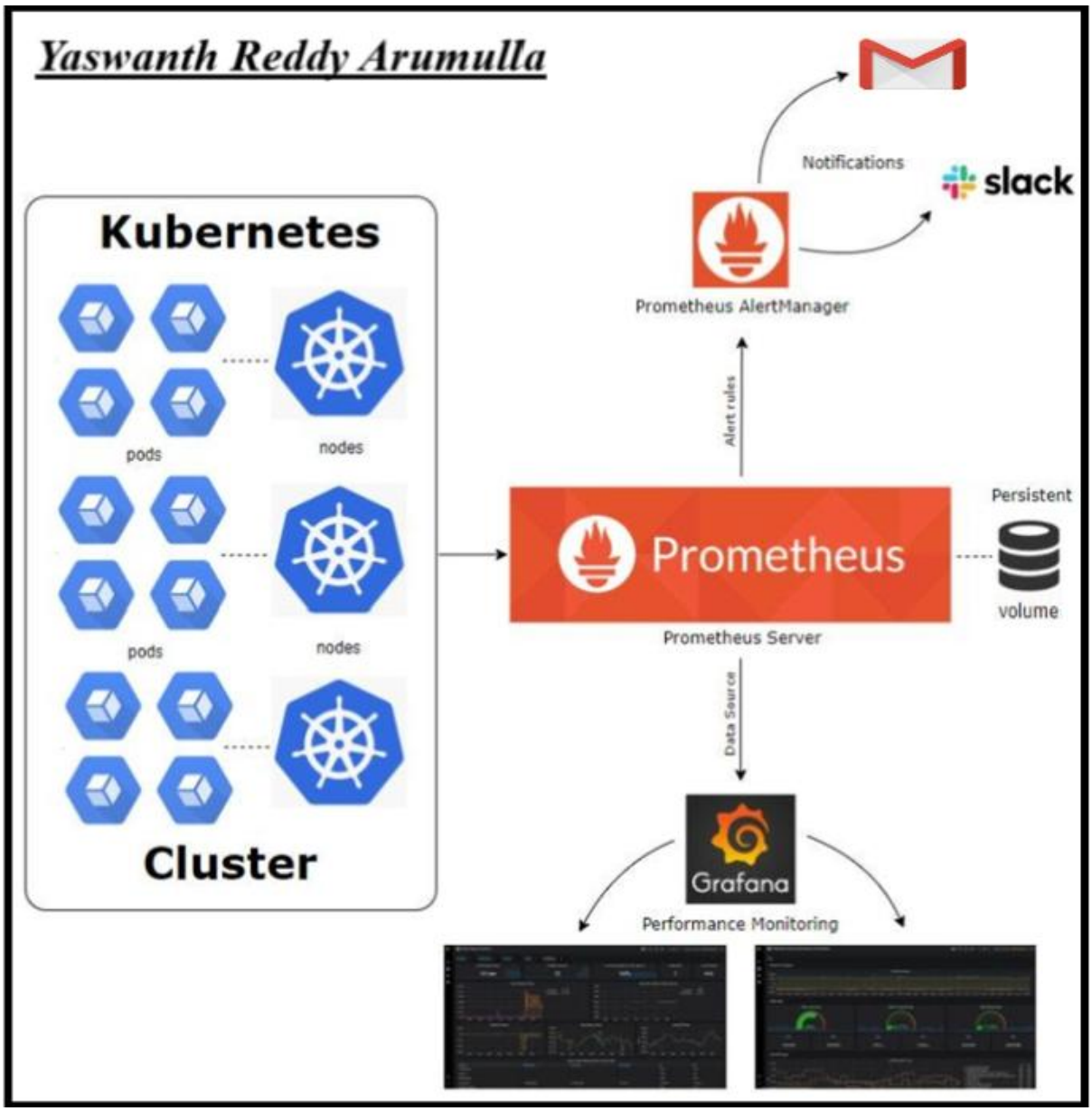
Prometheus & Grafana

🧠 **Why Monitoring Matters?**

In the world of cloud, microservices, and DevOps, monitoring your application and infrastructure is **critical**. Whether you're a fresher or a DevOps engineer, understanding how your systems behave in real time helps ensure availability, performance, and reliability.

This blog will walk you through setting up your **own monitoring stack** using **Prometheus and Grafana** from scratch. No prior monitoring experience required!

Architecture Diagram



🛠 What You'll Learn

➔ What is Monitoring
➔ What is Prometheus?
➔ What is Grafana?
➔ Why Use Prometheus + Grafana?
➔ How does Prometheus work?
➔ How Grafana works ?
➔ Monitor Kubernetes: CPU, RAM, pod status, etc.
➔ Visualize metrics using Grafana dashboards
➔ Access Grafana through a LoadBalancer
➔ Install everything using Helm (easiest way!)
➔ [Bonus] Monitor a sample app or Kubernetes cluster

1️⃣ What is Monitoring?

Monitoring is the continuous process of collecting, analyzing, and visualizing system and application metrics to understand the performance, availability, and overall health of your infrastructure.

2️⃣ What is Prometheus?

Prometheus is an open-source monitoring and alerting toolkit. It collects metrics from your systems and applications, stores them in a time-series database, and allows you to run powerful queries.

🔁 Pull-based metrics collection
🔍 Built-in querying language (PromQL)
📦 Lightweight and easy to install

3️⃣ What is Grafana ?

Grafana is a data visualization tool. It connects to Prometheus and allows you to create **interactive dashboards**, charts, and alerts — all through a simple UI.

📊 Dashboards and graphs
⏰ Alerts and thresholds
🔌 Plugin support for various data sources

4️⃣ Why Use Prometheus + Grafana?

When you use Prometheus and Grafana together, you get a powerful and easy way to monitor your system.

- Prometheus collects data like CPU usage, memory, disk space, and more.
- Grafana shows that data using beautiful graphs and charts. In short Prometheus watches your system.
- Grafana shows you what's happening clearly and beautifully. Together, they make system monitoring easy and smart.

5️⃣ How Prometheus works ?

- Prometheus asks other tools (called exporters) for system data — like "Hey, what's the CPU usage now?"
- Exporters give the data (like CPU: 30%, Memory: 40%)
- Prometheus stores this data with the time it was collected (this is called "timeseries data").
- You can see or analyze this data using its own web page or by connecting it to a tool like Grafana for beautiful dashboards.

6️⃣ How Grafana works ?

- Grafana connects to Prometheus (or other data sources)
- It reads the monitoring data (like CPU usage, memory, etc.)
- You can use Grafana to create dashboards and display that data in graphs
- You can also set alerts, so Grafana notifies you if something is wrong.

🎯 Goal: Install Prometheus + Grafana to Monitor Your K8s Cluster

📊 *Monitor your Argo CD–deployed website (running via LoadBalancer) — with **Prometheus + Grafana***
🔧 *View CPU, RAM, pod status, uptime, errors, etc.*

🧰 Prerequisites (Before We Start)

Make sure you have these ready 👇
1. A **Kubernetes Cluster** (EKS, GKE, Minikube — anything works)
2. kubectl is installed and connected to your cluster ✅
3. Helm is installed (helm version)

```
# Install Helm (if not installed)
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
helm version
```

4. Internet access to pull charts & Docker images
5. (Optional) Argo CD if you want GitOps deployment
If you're using GitOps, ensure:
☑ Argo CD is already deployed
☑ Your app is deployed using Argo CD
☑ Access to the app via Load Balancer

1️⃣ Create a Namespace for Monitoring

```
kubectl create namespace monitoring
```

2️⃣ Add Prometheus & Grafana Helm Chart Repo

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

3️⃣ Install the Kube Prometheus Stack (Includes Prometheus + Grafana)

```
helm install kube-prom-stack prometheus-community/kube-prometheus-stack \
  --namespace monitoring
```

🛠️ This installs:

- Prometheus (metrics collector)
- Grafana (dashboard visualizer)
- Alertmanager (for warnings)
- Node exporters (to get node metrics)

4️⃣. Check That Everything Is Running

```
kubectl get pods -n monitoring
```

You will get output like this :

NAME READY STATUS RESTARTS AGE
alertmanager-kube-prom-stack-kube-prome-alertmanager-0 2/2 Running 0 2m45s
kube-prom-stack-grafana-d5dfd9fd-m5j9t 3/3 Running 0 3m19s
kube-prom-stack-kube-prome-operator-6779bc5685-llmc8 1/1 Running 0 3m19s
kube-prom-stack-kube-state-metrics-6c4dc9d54-w48xj 1/1 Running 0 3m19s
kube-prom-stack-prometheus-node-exporter-vhncz 1/1 Running 0 3m19s
kube-prom-stack-prometheus-node-exporter-vx56f 1/1 Running 0 3m19s
prometheus-kube-prom-stack-kube-prome-prometheus-0 2/2 Running 0 2m45s

✅ Wait until STATUS is Running.

5️⃣. Accessing the Grafana UI Using LoadBalancer

Prometheus stack exposes Grafana as an internal service by default. Let's expose it to the world 🌍.

- Edit the Grafana Server File

kubectl edit svc kube-prom-stack-grafana -n monitoring

Change the Service Type :

- Find this line:

type: ClusterIP

- Change it to:

type: LoadBalancer

Save and exit (:wq for vi).

6️⃣. Get the Grafana LoadBalancer IP

kubectl get svc kube-prom-stack-grafana -n monitoring

- You will get output like this :

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kube-prom-stack-grafana LoadBalancer 172.20.174.208 abbda6b6f6c9345c6b017c020cf00122-1809047356.us-east-1.elb.amazonaws.com 80:32242/TCP 5m39s

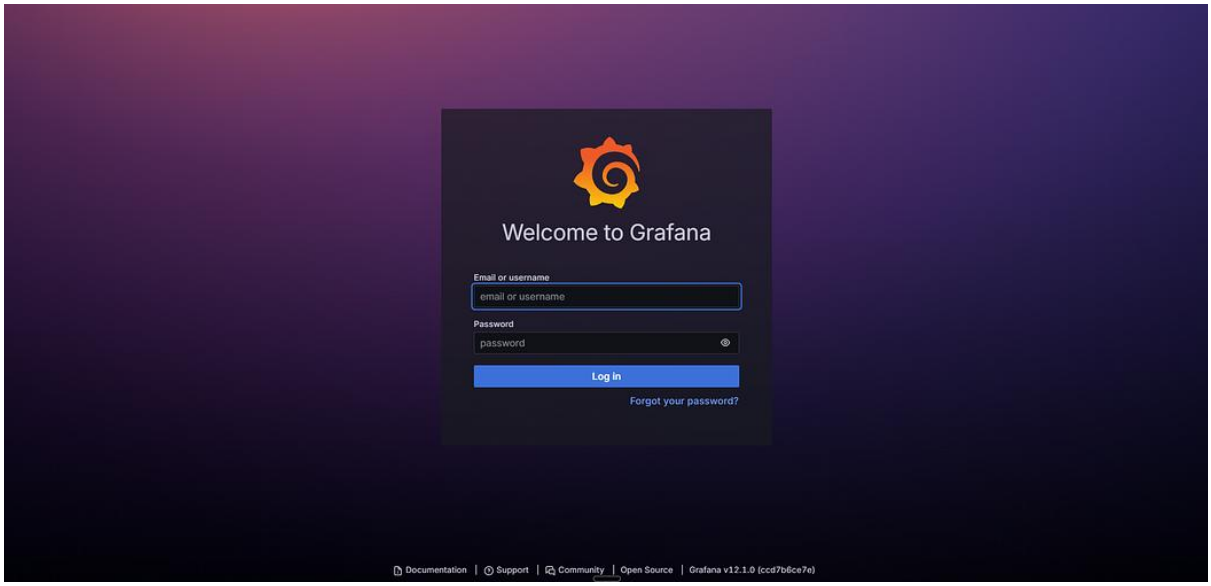
📌 Copy the **EXTERNAL-IP**, it will look like: http://a1b2c3d4.us-east-1.elb.amazonaws.com

7️⃣ Accessing the Grafana UI

1. Open Your Browser

- **Copy the EXTERNAL-IP** (the long .elb.amazonaws.com address).
- Paste it into your browser:

Press enter or click to view image in full size



You'll see the Grafana login page!

1. 🔐 Get the Initial Grafana Admin Password

kubectl get secret kube-prom-stack-grafana -n monitoring -o jsonpath="{.data.admin-password}" | base64 -d && echo

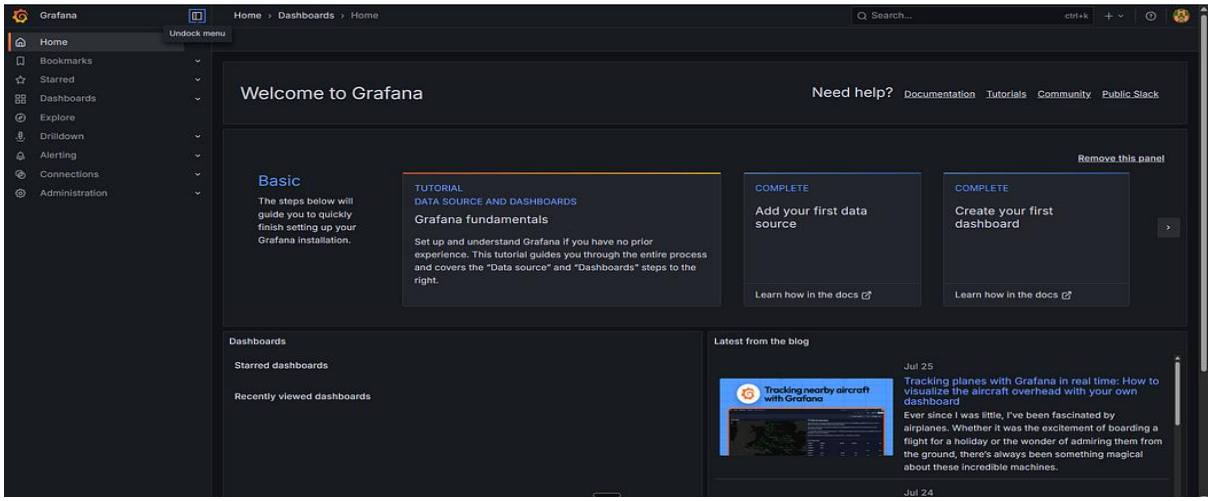
- You will get output like this :

prom-operator

Login to Grafana :

Go to your LoadBalancer IP in browser.
Default credentials:
***Username:** admin*
***Password:** prom-operator*
👉 Change the password when prompted.
You will like this

Press enter or click to view image in full size

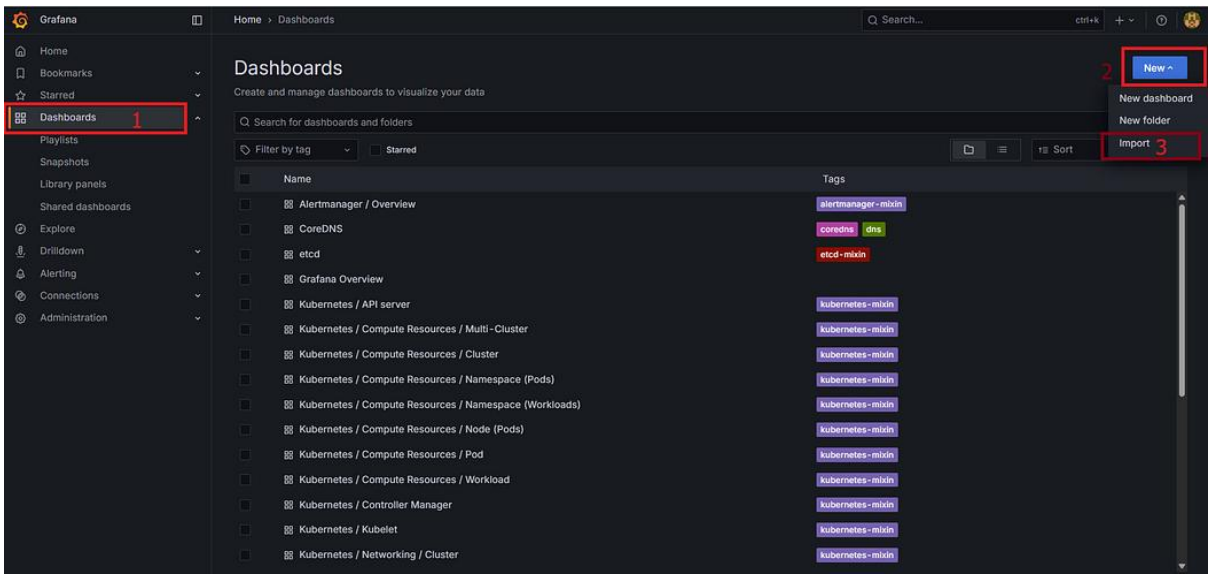


☸️ Add Kubernetes Dashboards in Grafana

📊 Go to:

1. Left menu → **Dashboards** → **+ Import** → New Dashboard
   Paste Dashboard ID:

Press enter or click to view image in full size



*In the text box under **“Import via Grafana.com”**, paste this number:*

Swiggy-GitOps-project/Grafana at master · arumullayaswanth/Swiggy-GitOps-project

Contribute to arumullayaswanth/Swiggy-GitOps-project development by creating an account on GitHub.

github.com

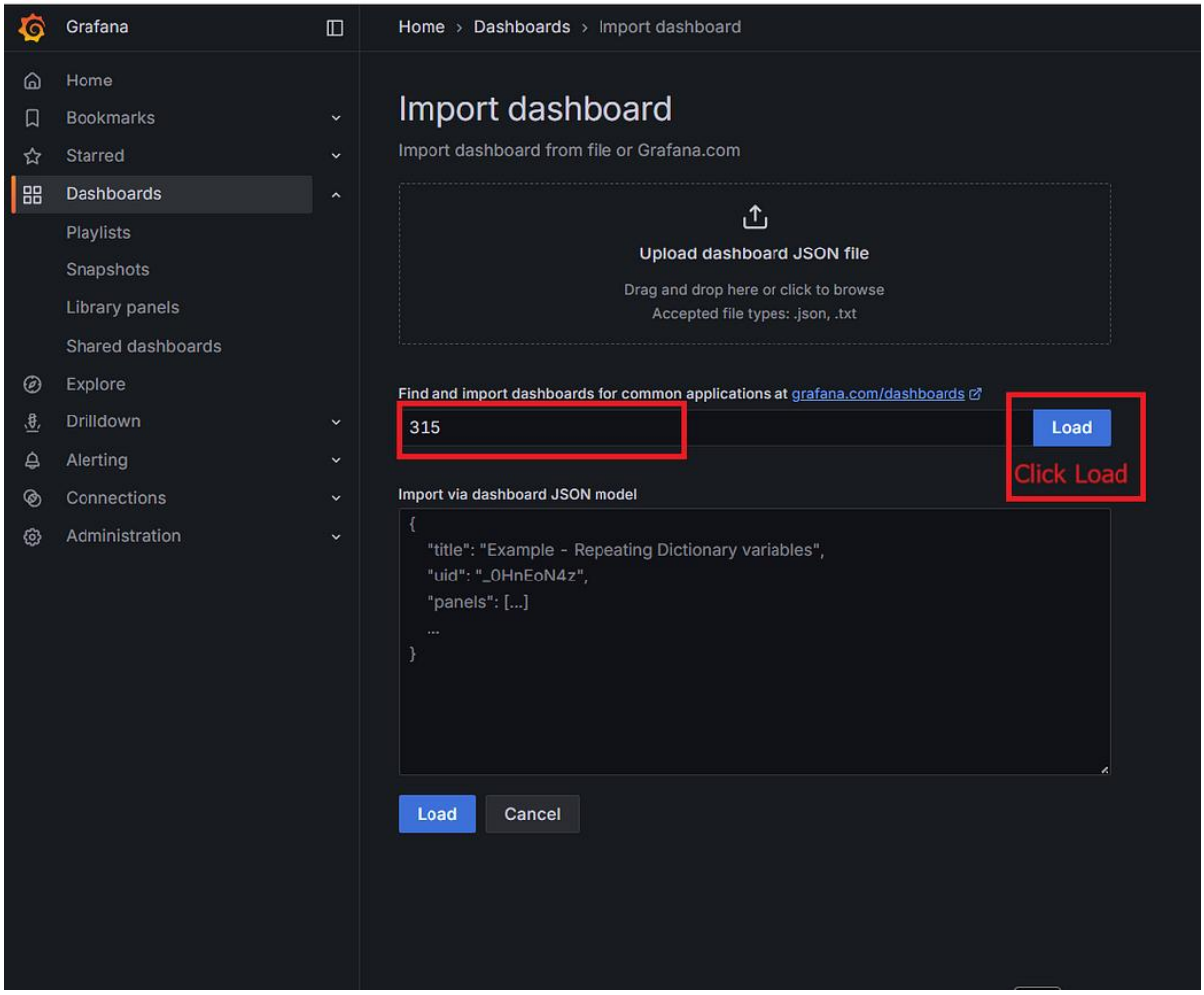
Otherwise you can use Json Files in this repository You can download Jason files and you can upload In the dashboard.

Kubernetes Cluster Monitoring (ID: 315)
Kubernetes Pods/Containers (ID: 3662)
Kubernetes Deployments (ID: 1621)
Kubernetes API Server (ID: 12006)
Kubernetes Nodes (ID: 6417)
Kubernetes Namespace Monitoring (ID: 10000)
Kubernetes Persistent Volumes (ID: 13602)
Kubernetes Networking (ID: 15758)
NGINX Ingress Controller (ID: 9614)

- Click **Load**

*Grafana expects **one dashboard ID at a time** in the "Grafana.com dashboard URL or ID" field.*
For example:
*1. You can enter **315**, click **Load**, and then import that dashboard.*
2. You must repeat this for each dashboard ID (e.g., 3662, 1621, etc.).
3.

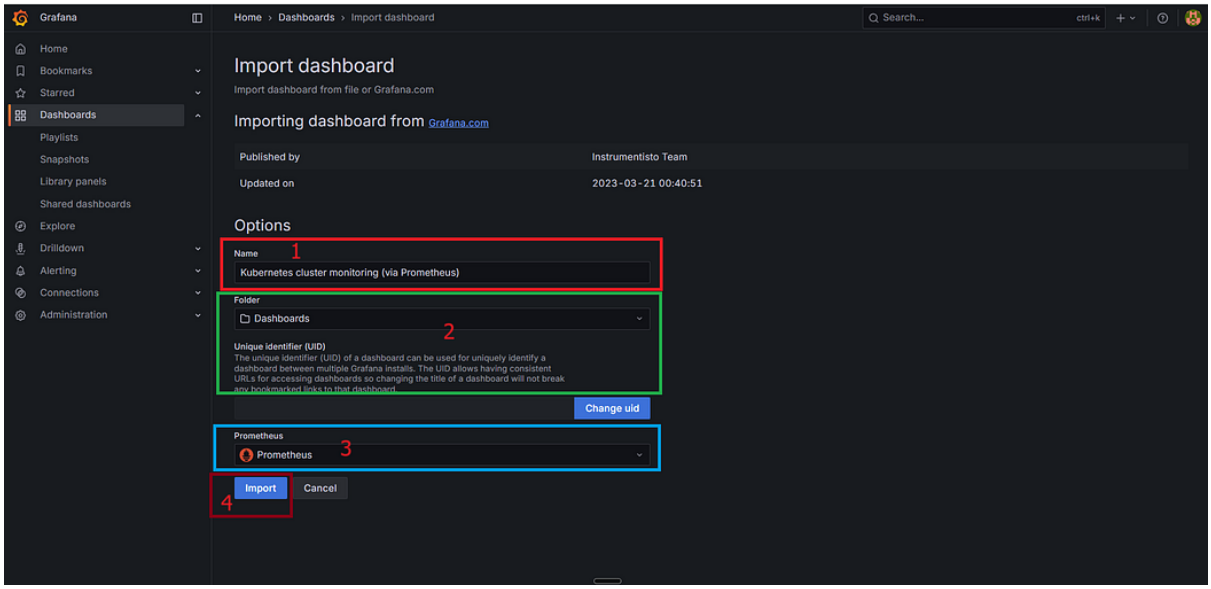
Press enter or click to view image in full size



2. Select Data source

- On the next screen:
- You'll see a dropdown **Prometheus Data Source**.
- Choose **Prometheus** (it's already installed with kube-prometheus-stack).
- Then click **Import**.

Press enter or click to view image in full size

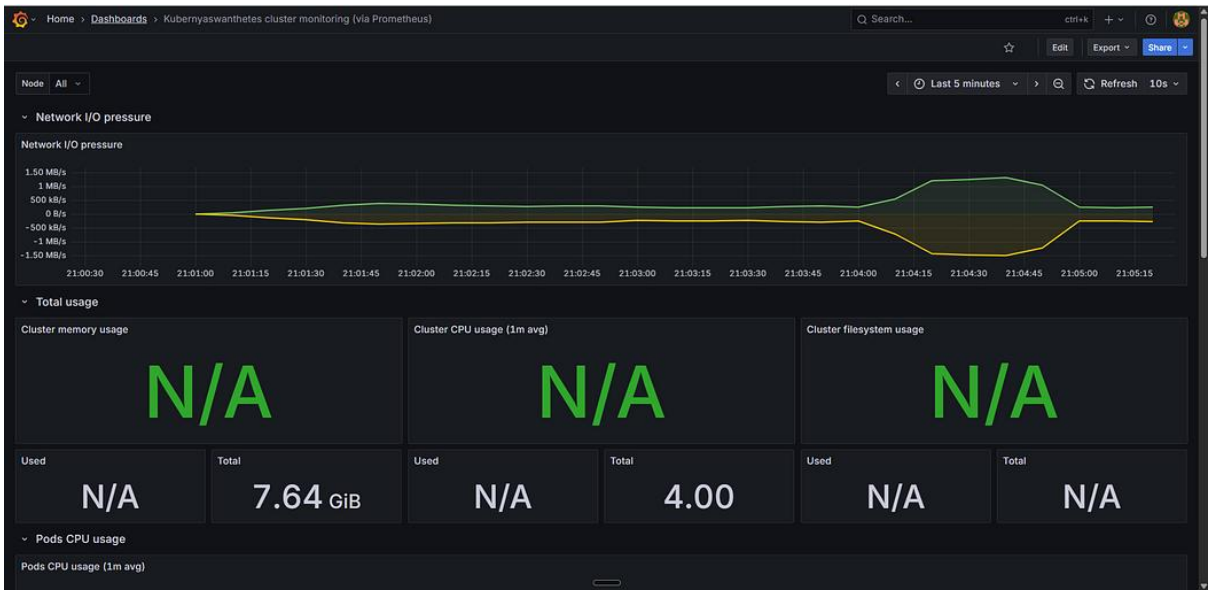


3. View the Dashboard

- After importing, the dashboard will automatically open.
- You'll now see:

CPU usage per Node *(which node is using the most CPU).*
Memory usage per Pod *(how much RAM each pod is using).*
Cluster Uptime.
Requests, errors, etc.

Press enter or click to view image in full size



💡 See Your Argo CD App Metrics!

All apps running in your cluster (including ones deployed via Argo CD) are automatically monitored!

Get Yaswanth Reddy Arumulla's stories in your inbox

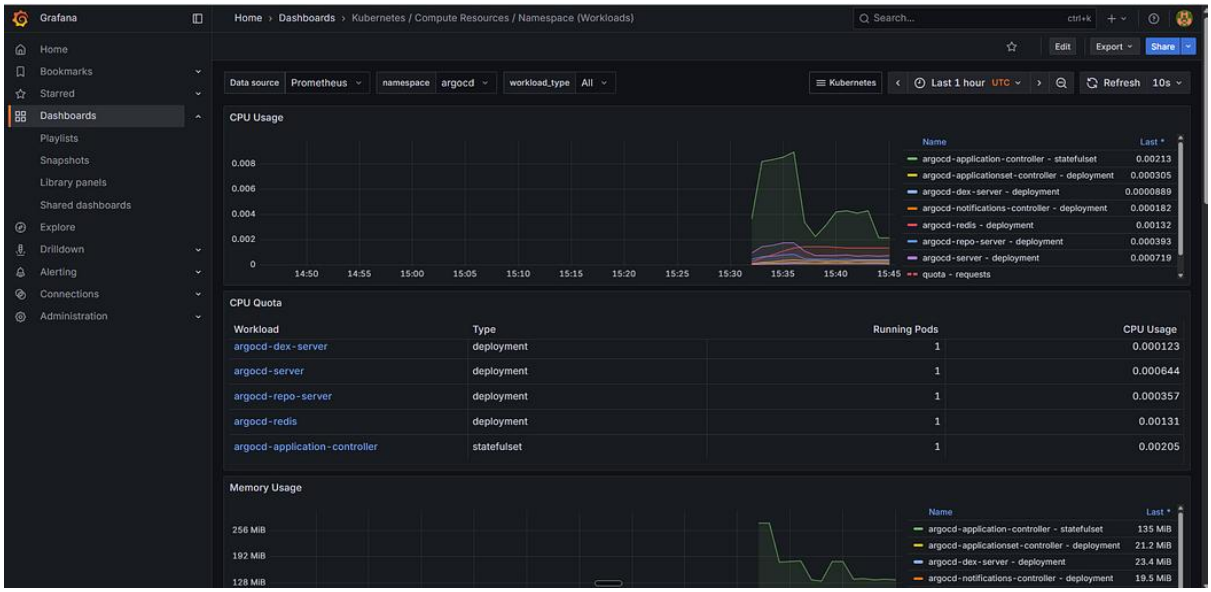
Join Medium for free to get updates from this writer.

Subscribe

You can import the **Argo CD dashboard from Grafana.com**:

1. Go to **Dashboards → Import**.
2. Use Dashboard ID **14584** (Argo CD Official Dashboard).
3. Select **Prometheus** as the data source.

Press enter or click to view image in full size



[Optional] Configure Alerts (Email Notifications)

🎯 Goal:

When CPU usage (or any other metric) goes beyond a threshold, you'll receive an **email alert** from **Alertmanager** (part of Prometheus stack).

① Confirm Alertmanager is Running

After installing the kube-prometheus-stack:

kubectl get pods -n monitoring

Look for something like:

alertmanager-kube-prom-stack-kube-prome-alertmanager-0
✅ If it's running, you're good to go.

② The Alertmanager dashboard provides:

- **Active Alerts** — A list of alerts currently firing (e.g., high CPU usage).
- **Silences** — You can configure silences to suppress certain alerts.
- **Status** — Displays cluster and configuration status.
- **Receivers** — Configured receivers like email, Slack, etc.

- **Routes** — The routing tree for alert notifications.

③ Edit the Alertmanager Config Map

The configuration for Alertmanager is stored in a **ConfigMap**.
Run:

kubectl get pods -n monitoring | grep alertmanager

Ensure:

alertmanager-kube-prom-stack-kube-prome-alertmanager-0 2/2 Running

✅ **To Access Alertmanager via Load Balancer (Externally):**

You need to **change the service type** from ClusterIP to LoadBalancer.

🔧 **Step 1: Edit the Alertmanager Service**

Run:

kubectl edit svc kube-prom-stack-kube-prome-alertmanager -n monitoring

🔁 **Step 2: Change This Line:**

Find:

type: ClusterIP

Change it to:

type: LoadBalancer

Save and exit (:wq if using vim)

⏳ **Step 3: Wait for External IP**

Check again with:

kubectl get svc kube-prom-stack-kube-prome-alertmanager -n monitoring

It will show something like:

kube-prom-stack-kube-prome-alertmanager LoadBalancer 172.20.51.43 abc123456789.elb.amazonaws.com 9093:xxxxx/TCP ...

✅ Copy the DNS under EXTERNAL-IP

🌐 **Step 4: Access Alertmanager in Browser**

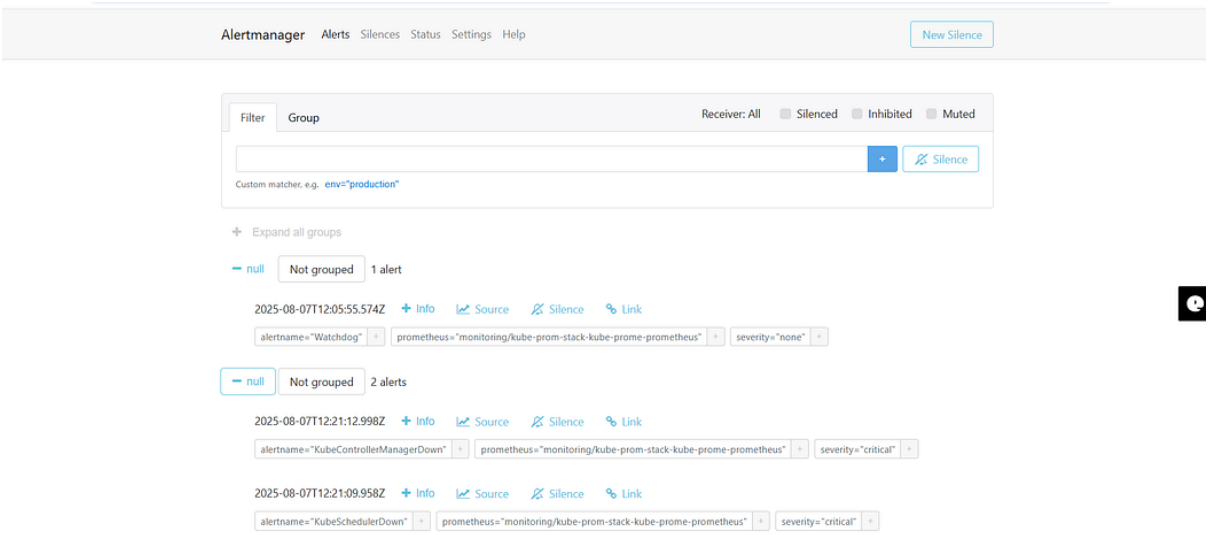
Use:

http://<external-dns>:9093

Example:

http://abc123456789.elb.amazonaws.com:9093

Press enter or click to view image in full size



✅ Optional: Open Port 9093 in Security Group

If it doesn't load:

- Go to **AWS EC2 Console → Load Balancers**
- Find the Alertmanager ELB
- Open the **Security Group**
- Edit **Inbound Rules**:
- Add rule for **TCP 9093**
- Source: 0.0.0.0/0 (or your IP)

run this commends

kubectl get secret alertmanager-kube-prom-stack-kube-prome-alertmanager -n monitoring -o jsonpath='{.data.alertmanager\.yaml}' | base64 --decode > alertmanager.yaml

ls

vim alertmanager.yaml

④ Add Email Configuration

Inside the alertmanager.yml section, add your SMTP email settings:
(Replace with your email SMTP provider details.)

```
global:
  smtp_smarthost: 'smtp.gmail.com:587'      # Your SMTP server
  smtp_from: 'yaswanth.arumulla@hmail.com'        # Sender email
  smtp_auth_username: 'yaswanth.arumulla@gmai.com'
  smtp_auth_password: 'your-app-password'   # Use app password (not your real password!)

route:
  receiver: 'email-alert'
```

```
receivers:
  - name: 'email-alert'
    email_configs:
      - to: 'yaswanth.arumulla@gmail.com'      # Where to send alerts
        send_resolved: true
```

⚠️ **For Gmail**:

- Enable **"Less Secure Apps"** or create an **App Password** from Google account security settings.

```
kubectl create secret generic alertmanager-kube-prom-stack-kube-prome-alertmanager \
  --from-file=alertmanager.yaml \
  -n monitoring \
  --dry-run=client -o yaml | kubectl apply -f -
```

⑤ Save and Restart Alertmanager

After editing, restart the Alertmanager pod to apply changes:

kubectl delete pod alertmanager-kube-prom-stack-kube-prome-alertmanager-0 -n monitoring

Wait for Restart

kubectl get pods -n monitoring -w

Look for:

alertmanager-kube-prom-stack-kube-prome-alertmanager-0 2/2 Running 0 30s

⑥ Create an Alert Rule (CPU Example)

We'll add an alert rule to **trigger an email** when CPU > 70%.

Create a new YAML file called cpu-alert-rule.yaml:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: cpu-alert
  namespace: monitoring
spec:
  groups:
    - name: cpu.rules
      rules:
        - alert: HighCPUUsage
          expr: sum(rate(container_cpu_usage_seconds_total[1m])) > 0.7
          for: 2m
          labels:
            severity: warning
          annotations:
            summary: "High CPU Usage detected"
            description: "CPU usage is above 70% for 2 minutes."
```

Apply the rule:

kubectl apply -f cpu-alert-rule.yaml

⑦ Test Your Alert

- Run a CPU-heavy process in a pod (simulate load).
- Wait 2–3 minutes.
- Check your email — you should receive an alert!

⑧ Verify Alerts in Prometheus

You can also see alerts in the Prometheus UI:

Step 1: Edit the Prometheus Service

Run:

kubectl edit svc kube-prom-stack-kube-prome-prometheus -n monitoring

Change the Service Type :

- Find this line:

type: ClusterIP

- Change it to:

type: LoadBalancer

Save and exit (:wq for vi).

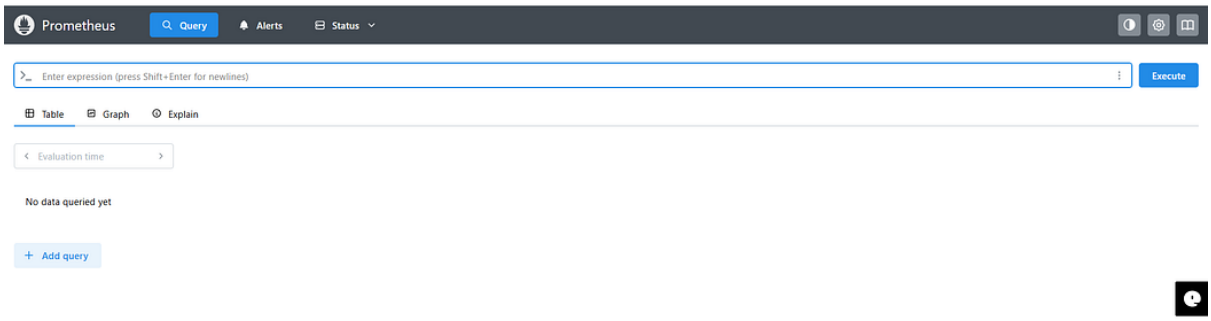
- Get the ***Prometheus LoadBalancer IP***

kubectl get svc kube-prom-stack-kube-prome-prometheus -n monitoring

NAME TYPE CLUSTER-IP EXTERNAL-IP
PORT(S) AGE
kube-prom-stack-kube-prome-prometheus LoadBalancer 172.20.146.205
a17530e645b134734ba1cff112072526-1666914053.us-east-1.elb.amazonaws.com
9090:32606/TCP,8080:31797/TCP 3h35m

📌 Copy the **EXTERNAL-IP**, it will look like: http://a1b2c3d4.us-east-1.elb.amazonaws.com:9090

Press enter or click to view image in full size



🎉 Done!

Now, you'll **get email alerts** whenever CPU usage crosses the limit.

🎉 Final Checklist

- ✅ Prometheus & Grafana Installed
- ✅ Grafana Accessible via LoadBalancer
- ✅ Kubernetes Metrics Visible
- ✅ Argo CD Deployed App Visible
- ✅ Dashboards Working
- ✅ Optional Alerts Configured

💡 Bonus: What You Can Monitor

- ✅ CPU/RAM of your Argo CD app
- ✅ Pod crashes/restarts
- ✅ Node health
- ✅ Cluster capacity
- ✅ Response times
- ✅ Resource usage per container

🏁 Conclusion

Monitoring Kubernetes is **not just a luxury — it's a necessity** in modern cloud-native environments. With **Prometheus and Grafana**, you can gain real-time insights into your applications, nodes, and infrastructure performance.

- **Prometheus** ensures that metrics are collected and stored efficiently.
- **Grafana** makes those metrics meaningful with beautiful, actionable dashboards.
- With **Alertmanager**, you can proactively respond to issues like high CPU or memory usage before they affect users.

By following the steps in this guide — from installation using Helm, exposing Grafana via LoadBalancer, importing dashboards, and setting up email alerts — you've built a **complete end-to-end monitoring stack** suitable for both beginners and experienced DevOps engineers.

Next Steps:

- Add more dashboards tailored to your apps.
- Set up advanced alerts (e.g., latency, error rates).
- Explore integrations like Loki for log monitoring and Tempo for tracing.

🚀 **With Prometheus + Grafana, you're now ready to monitor, visualize, and optimize your Kubernetes applications like a pro!**