To install Prometheus and Grafana on an Amazon EKS (Elastic Kubernetes Service) cluster and configure Grafana to use Prometheus as a data source for collecting metrics, follow these step-by-step instructions.

**Prerequisites**

1. **EKS Cluster**: You must have an EKS cluster running. If not, create one using eksctl or AWS Console.
2. **kubectl Installed**: Ensure kubectl is installed and configured to communicate with your EKS cluster .
3. Helm Installed: Install Helm if not already installed.
4. **Eks cluster creation & kubectl Installed & Helm Installed:**

To create an **Amazon EKS** cluster, install kubectl, and configure it to communicate with the cluster, follow these **step-by-step** instructions.

**Step 1: Install AWS CLI, eksctl & kubectl**

Before creating an EKS cluster, you need to install the required tools.

**1. Install AWS CLI**

If you haven’t installed the AWS CLI, install it using the following command:

**Linux & macOS**

curl "https://awscli.amazonaws.com/AWSCLIV2-$(uname -s | tr '[:upper:]' '[:lower:]')-$(uname -m).zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

**Windows**

Download and install from:  
<https://awscli.amazonaws.com/AWSCLIV2.msi>

**Verify Installation**

aws --version

**2. Configure AWS CLI with IAM User or Role**

Run:

aws configure

Enter your AWS **Access Key ID**, **Secret Access Key**, **Region**, and **Output format**.

To verify:

aws sts get-caller-identity

**3. Install eksctl**

**Linux & macOS**

curl -sSL "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /usr/local/bin

**Windows**

choco install eksctl

or download from <https://github.com/weaveworks/eksctl/releases>.

**Verify Installation**

eksctl version

**4. Install kubectl**

EKS requires a specific version of kubectl. To install the latest compatible version:

**Linux**

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

**macOS**

brew install kubectl

**Windows**

choco install kubernetes-cli

**Verify Installation**

kubectl version --client

**Step 2: Create an EKS Cluster**

Now, create an **EKS cluster** using eksctl.

**1. Create an EKS Cluster with eksctl**

eksctl create cluster \

--name my-eks-cluster \

--region us-east-1 \

--version 1.27 \

--nodegroup-name my-node-group \

--node-type t3.medium \

--nodes 2 \

--nodes-min 1 \

--nodes-max 3 \

--managed

* --name my-eks-cluster → Name of the cluster.
* --region us-east-1 → AWS region (Change if needed).
* --version 1.27 → Kubernetes version.
* --nodegroup-name my-node-group → Name of the worker node group.
* --node-type t3.medium → EC2 instance type for nodes.
* --nodes 2 → Initial node count.
* --nodes-min 1 and --nodes-max 3 → Auto-scaling settings.
* --managed → Managed node group.

This process takes about **10–15 minutes**.

**2. Verify the Cluster Creation**

After the cluster is created, check its status:

eksctl get cluster --name my-eks-cluster --region us-east-1

**Step 3: Configure kubectl to Connect to EKS**

Once your EKS cluster is ready, configure kubectl to communicate with it.

**1. Update the Kubeconfig**

Run:

aws eks --region us-east-1 update-kubeconfig --name my-eks-cluster

This updates the Kubernetes configuration file so kubectl can connect to the EKS cluster.

**2. Verify the Connection**

kubectl get nodes

You should see your worker nodes in the **Ready** state.

**Final Steps**

Now, your **EKS cluster is running**, and kubectl is configured! 🎉

From here, you can:

* Deploy applications on **EKS**.
* Install **Prometheus and Grafana** (as per the previous guide).
* Implement **Ingress and LoadBalancer** to expose services.

**To install helm**

Helm Installed: Install Helm if not already installed

curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

chmod 700 get\_helm.sh

./get\_helm.sh

Let me know if you need further assistance! 🚀

**Deployment of Prometheus & Grafana**

**Step 1: Deploy Prometheus & Grafana Using Helm**

We will use the **Prometheus Community Helm Chart** to deploy both Prometheus and Grafana.

**1. Add the Prometheus Helm Repository**

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

helm repo update

**2. Create a Namespace for Monitoring**

kubectl create namespace monitoring

**3. Install Prometheus Stack**

Use the following Helm command to install **Prometheus and Grafana**:

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

This installs **Prometheus**, **Grafana**, and the necessary exporters to collect Kubernetes metrics.

**Step 2: Expose Prometheus & Grafana Services**

By default, these services are available inside the cluster. You can access them externally using **kubectl port-forward** or an **ELB/ALB Ingress Controller**.

**1. Port-forward Prometheus**

kubectl port-forward -n monitoring svc/prometheus-kube-prometheus-prometheus 9090:9090

after this it will start forwarding the traffic from local host to the pod , for next execution process use new terminal , u cant use the same one beacouse it is in process of forwarding .

You can now access Prometheus at [http://localhost:9090](http://localhost:9090/).

**2. Port-forward Grafana**

kubectl port-forward -n monitoring svc/prometheus-grafana 3000:80 .

after this it will start forwarding the traffic from local host to the pod , for next execution process use new terminal , u cant use the same one beacouse it is in process of forwarding .

Grafana will be available at [http://localhost:3000](http://localhost:3000/).

**Step 3: Get Grafana Admin Credentials**

The default username is admin. To get the password, run:

kubectl get secret -n monitoring prometheus-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

Use these credentials to log in to Grafana.

**Step 4: Configure Prometheus as a Data Source in Grafana**

1. **Log in to Grafana** at http://localhost:3000
2. Navigate to **Configuration → Data Sources**.
3. Click **"Add Data Source"**.
4. Select **Prometheus**.
5. In the **URL** field, enter:

http://prometheus-kube-prometheus-prometheus.monitoring.svc.cluster.local:9090

1. Click **Save & Test**.

**Step 5: Import Kubernetes Dashboards**

1. In Grafana, go to **Dashboards** → **Import**.
2. Use an existing **Kubernetes Dashboard** from Grafana’s repository (e.g., ID 3119 for Kubernetes cluster monitoring).
3. Select **Prometheus** as the data source.
4. Click **Import**.

Now, you should see Kubernetes metrics visualized in Grafana!

**Step 6: (Optional) Expose Services Using LoadBalancer or Ingress**

To access **Grafana** externally:

kubectl patch svc prometheus-grafana -n monitoring -p '{"spec": {"type": "LoadBalancer"}}'

Get the external IP:

kubectl get svc -n monitoring prometheus-grafana

Use the external IP to access Grafana.

But After this if you try to access this external IP it will not open this is beacouse u haven’t opened the inbound rule,

If u are using aks , Go to the aks vms agent pool ,and go to the networking and add the inbound rules and the port which have to open and accessed . After it will connect using with node ip and port .

If u are using eks go to the security group attached to the vm’s and add inbound rules.

**Final Notes**

* **Prometheus** collects Kubernetes metrics automatically.
* **Grafana** visualizes the data from Prometheus.
* Use **Helm values.yaml** if you need custom configurations.

Let me know if you need any clarifications! 🚀

To expose Prometheus and Grafana in EKS, you can use NodePort, LoadBalancer, or Ingress. Below are detailed steps for each method.

**1️⃣ Expose Services Using NodePort**

A **NodePort** exposes services on a static port (30000–32767) on all cluster nodes.

**Modify Prometheus Service to Use NodePort**

kubectl edit svc -n monitoring prometheus-kube-prometheus-prometheus

Find the spec.type field and change it to NodePort:

spec:

type: NodePort

Find the **Prometheus port section** and add a nodePort value:

ports:

- name: web

port: 9090

targetPort: 9090

nodePort: 30090

Save and exit.

**Modify Grafana Service to Use NodePort**

kubectl edit svc -n monitoring prometheus-grafana

Change the spec.type to NodePort and set a nodePort:

spec:

type: NodePort

ports:

- name: service

port: 80

targetPort: 3000

nodePort: 30030

Save and exit.

**Access Prometheus & Grafana**

Find a worker node’s public IP:

kubectl get nodes -o wide

Then access:

* **Prometheus**: http://<NODE\_PUBLIC\_IP>:30090
* **Grafana**: http://<NODE\_PUBLIC\_IP>:30030
* But After this if you try to access this external IP it will not open this is beacouse u haven’t opened the inbound rule,
* If u are using aks , Go to the aks vms agent pool ,and go to the networking and add the inbound rules and the port which have to open and accessed . After it will connect using with node ip and port .
* If u are using eks go to the security group attached to the vm’s and add inbound rules.

**2️⃣ Expose Services Using LoadBalancer**

A **LoadBalancer** exposes the service externally via an AWS ELB.

**Modify Prometheus Service to Use LoadBalancer**

kubectl patch svc prometheus-kube-prometheus-prometheus -n monitoring -p '{"spec": {"type": "LoadBalancer"}}'

**Modify Grafana Service to Use LoadBalancer**

kubectl patch svc prometheus-grafana -n monitoring -p '{"spec": {"type": "LoadBalancer"}}'

**Get External Access URLs**

kubectl get svc -n monitoring prometheus-kube-prometheus-prometheus prometheus-grafana

Look for the **EXTERNAL-IP** field.

Access:

* **Prometheus**: http://<EXTERNAL-IP>:9090
* **Grafana**: http://<EXTERNAL-IP>
* But After this if you try to access this external IP it will not open this is beacouse u haven’t opened the inbound rule,
* If u are using aks , Go to the aks vms agent pool ,and go to the networking and add the inbound rules and the port which have to open and accessed . After it will connect using with node ip and port .
* If u are using eks go to the security group attached to the vm’s and add inbound rules.

**3️⃣ Expose Services Using Ingress**

Ingress provides a single entry point using **AWS ALB Ingress Controller**.

**1. Install AWS ALB Ingress Controller**

helm repo add eks https://aws.github.io/eks-charts

helm repo update

helm install aws-load-balancer-controller eks/aws-load-balancer-controller \

--set clusterName=my-eks-cluster \

--set serviceAccount.create=false \

--set region=us-east-1 \

--set vpcId=<VPC\_ID> \

-n kube-system

**2. Create an Ingress Resource**

Create a file grafana-ingress.yaml:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: grafana-ingress

namespace: monitoring

annotations:

alb.ingress.kubernetes.io/scheme: internet-facing

spec:

ingressClassName: alb

rules:

- host: grafana.mydomain.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: prometheus-grafana

port:

number: 80

Apply it:

kubectl apply -f grafana-ingress.yaml

**3. Get the Ingress URL**

kubectl get ingress -n monitoring .

Once the ALB is provisioned, you can access Grafana at <http://grafana.mydomain.com>.

* But After this if you try to access this external IP it will not open this is beacouse u haven’t opened the inbound rule,
* If u are using aks , Go to the aks vms agent pool ,and go to the networking and add the inbound rules and the port which have to open and accessed . After it will connect using with node ip and port .
* If u are using eks go to the security group attached to the vm’s and add inbound rules.

**Conclusion**

| **Method** | **Access Type** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| **NodePort** | Local only via Node IP | Easy to set up | Requires manually finding node IP |
| **LoadBalancer** | Public AWS ELB | Direct external access | Creates a new ELB per service |
| **Ingress** | Single domain-based access | Centralized entry point, supports HTTPS | Requires ALB Ingress Controller |

Let me know if you need more details! 🚀