

Comisión 2403 - Grupo 02

Curso Devops

PIN FINAL

German Montori <gerlm86@gmail.com >

Guillermo Getar <guillermo.getar@unc.edu.ar>

Mauro Pereira <mauro.a.pereira@gmail.com>

Santiago Gordillo <arielgordillolucas@gmail.com>



Introducción

Este proyecto tiene como idea principal el aprendizaje sobre distintos temas y la puesta en práctica mediante un laboratorio que permita integrar diferentes herramientas y tecnologías.

Durante la primera parte nos centramos en la creación de una instancia de EC2 en AWS para poder desde allí realizar todas las tareas necesarias.

Por último, configuramos la parte de monitoreo de pods con el stack de Prometheus y Grafana.

Flujo de trabajo

- Creación de cuenta de AWS junto a los permisos necesarios.
- Instalación y configuración de EKS.
- Creación del cluster.
- Chequeo de cluster.
- Despliegue de Prometheus.
- Chequeo de despliegue de Prometheus.
- Despliegue de Grafana.
- Chequeo de despliegue de Grafana.
- Configuración de dashboard.
- Conclusión.

Creación de cuenta de AWS junto a los permisos necesarios

Se reutiliza el usuario del PIN2 `aws_cli_user` pero agregando la política **AmazonEC2FullAccess**.

aws

Global

mauro.a.pereira

IAM

Personas

aws_cli_user

Permisos

Grupos (1)

Etiquetas (1)

Credenciales de seguridad

Último acceso

Políticas de permisos (4)

Eliminar

Agregar permisos

Los permisos se definen mediante políticas asociadas a la persona directamente o a través de grupos.

Filtrar por Tipo

Buscar

Todos los tipos

< 1 >

Nombre de la política

Tipo

AdministradorAccess

Administrada por AWS: función de trabajo

AmazonEC2FullAccess

Administrada por AWS

AmazonEKSClusterPolicy

Administrada por AWS

AmazonEKSWorkerNodePolicy

Administrada por AWS

CloudShell

Comentarios

Privacidad

Términos

Preferencias de cookies

© 2025, Amazon Web Services, Inc. o sus filiales.

Instalación y configuración de EKS

La instalación de programas como **kubectl**, **docker-compose**, **helm** y **terraform** se lleva a cabo a través de un script llamado **install_tools.sh**, el cual también configura los PATH para que **eksctl** pueda acceder a ellos. **eksctl** es un cli para crear y administrar los cluster en **Amazon EKS** por comandos. Es importante configurar bien el usuario de **aws cli**.

```
$ install_tools.sh U X
00_install_tools > $ install_tools.sh
24 kubectl version --client
25 echo "kubectl installed successfully"
26
27
28 # Instalación de eksctl
29 echo "Installing eksctl"
30 curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname)
31 if [ $? -ne 0 ]; then
32     echo "Failed to download eksctl"
33     exit 1
34 fi
35 sudo mv /tmp/eksctl /usr/local/bin
36 if [ $? -ne 0 ]; then
37     echo "Failed to move eksctl to /usr/local/bin"
38     exit 1
39 fi
40 export PATH=$PATH:/usr/local/bin
41 echo 'export PATH=$PATH:/usr/local/bin' >> ~/.bashrc
42 eksctl version
43
44 # Instalación de Docker
45 echo "Installing Docker"
46 sudo apt install -y docker
47 sudo systemctl enable docker.service
```

```
mauro@map-xu22-z170xpsli:~$ aws sts get-caller-identity
{
  "UserId": "AIDAYUQGTFIMKYQW2PUED",
  "Account": "593793067544",
  "Arn": "arn:aws:iam::593793067544:user/aws_cli_user"
}
```

Creación del cluster

A través del script `create_cluster.sh` se crea un cluster cuyas propiedades se definen a través de variables, como el nombre, el nombre del grupo al que pertenece, la contraseña, las zonas, etc. También importa las claves ssh e incluye la opción `--delete` que se encarga de eliminar todo lo creado por el script, para no incurrir en gastos innecesarios o en la etapa desarrollo, para ir probando las distintas configuraciones.

```
01_cluster > $ create_cluster.sh
1  #!/bin/bash
2  set -e # Detiene el script si cualquier comando falla
3
4  # Configuración de Variables
5  CLUSTER_NAME="eks-grupo-2"
6  NODEGROUP_NAME="ng-grupo-2"
7  AWS_REGION="us-east-1"
8  NODE_TYPE="t3.small"
9  NODE_COUNT=3
10 SSH_KEY="terraform-key"
11 ZONES="us-east-1a,us-east-1b,us-east-1c"
12 SSH_DIR="$HOME/.ssh"
13 SSH_KEY_PATH="$SSH_DIR/$SSH_KEY"
14
15 # Asegurarse de que /usr/local/bin está en el PATH
16 export PATH=$PATH:/usr/local/bin
17
18 # Función para mostrar mensajes de error
19 Codeium: Refactor | Explain | Generate Function Comment | ✕
20 function error_exit {
21     echo "Error: $1"
22     exit 1
23 }
24
25 # Verificar si eksctl está disponible
26 if ! command -v eksctl &> /dev/null
27 then
28     error_exit "eksctl no está instalado o no está en el PATH."
29 fi
30
31 # Verificar si AWS CLI está configurado y si las credenciales son válidas
32 aws sts get-caller-identity > /dev/null
33 if [ $? -eq 0 ]; then
34     echo "Credenciales de AWS validadas, procediendo con la creación del clúster..."
35 fi
```

```
aws ec2 describe-key-pairs --key-names "$SSH_KEY" --region "$AWS_REGION" > /dev/null 2>&1
if [ $? -ne 0 ]; then
    echo "Clave SSH '$SSH_KEY' no encontrada. Generando nueva clave..."

    # Generar clave SSH si no existe localmente
    if [ ! -f "$SSH_KEY_PATH" ]; then
        ssh-keygen -t rsa -b 4096 -f "$SSH_KEY_PATH" -N ""
        echo "Clave SSH generada en $SSH_KEY_PATH"
    fi

    # Subir clave a AWS
    aws ec2 import-key-pair --key-name "$SSH_KEY" --public-key-material fileb://"$SSH_KEY_PATH".pub --region "$AWS_REGION"
    if [ $? -eq 0 ]; then
        echo "Clave SSH '$SSH_KEY' importada a AWS correctamente."
    else
        error_exit "Error al importar la clave SSH en AWS."
    fi
fi

# Si el parámetro --delete está presente, eliminar el clúster y salir
if [[ "$1" == "--delete" ]]; then
    echo "Eliminando el clúster de EKS '$CLUSTER_NAME'..."
    eksctl delete cluster --name "$CLUSTER_NAME"
    exit 0
fi

# Crear el clúster en EKS
echo "Creando el clúster de EKS '$CLUSTER_NAME' en la región $AWS_REGION..."
eksctl create cluster \
    --name "$CLUSTER_NAME" \
    --region "$AWS_REGION" \
    --nodegroup-name "$NODEGROUP_NAME" \
    --node-type "$NODE_TYPE" \
    --node-count "$NODE_COUNT" \
    --zones "$ZONES"
```


Chequeo de cluster

Gracias a la integración de **eksctl** con **kubectl**, se puede chequear fácilmente lo desplegado como por ejemplo los nodos:

```
2025-03-02 22:38:20 [✓] created 1 managed nodegroup(s) in cluster "eks-grupo-2"
2025-03-02 22:38:21 [i] kubectl command should work with "/home/mauro/.kube/config", try 'kubectl get nodes'
2025-03-02 22:38:21 [✓] EKS cluster "eks-grupo-2" in "us-east-1" region is ready
El clúster 'eks-grupo-2' fue creado exitosamente.
mauro@map-xu22-lenovo:~/repos/map/PIN_FINAL$ kubectl get nodes
NAME                                STATUS    ROLES    AGE     VERSION
ip-192-168-56-245.ec2.internal      Ready    <none>    4m16s   v1.30.9-eks-5d632ec
ip-192-168-6-124.ec2.internal       Ready    <none>    4m14s   v1.30.9-eks-5d632ec
ip-192-168-73-234.ec2.internal      Ready    <none>    4m9s    v1.30.9-eks-5d632ec
```

Despliegue de Prometheus

La última parte consiste en desplegar el monitoreo de los pods a través de **Prometheus** y **Grafana**, esto se logra con la integración de **helm** con **eksctl**. El primer paso es desplegar prometheus a través de un script que chequea previamente si ya existe el namespace, en caso de que exista, se borra, es una opción a utilizar `--delete`.

```
02_monitoreo > $ install_prometheus.sh
1
2
3
4  NAMESPACE="prometheus"
5
6  # Eliminar Prometheus si existe
7  if helm ls -n "$NAMESPACE" | grep -q prometheus; then
8  |   echo "🗑 Eliminando Prometheus..."
9  |   helm uninstall prometheus -n "$NAMESPACE"
10 else
11 |   echo "⚠ Prometheus ya estaba eliminado."
12 fi
13
14 # Asegurar que el namespace no existe
15 if kubectl get namespace "$NAMESPACE" &>/dev/null; then
16 |   echo "🗑 Eliminando namespace $NAMESPACE..."
17 |   kubectl delete namespace "$NAMESPACE"
18 else
19 |   echo "⚠ Namespace $NAMESPACE ya estaba eliminado."
20 fi
21
22 # Crear namespace
23 kubectl create namespace "$NAMESPACE"
24
25 # Instalar Prometheus sin almacenamiento persistente
26 helm install prometheus prometheus-community/prometheus \
27 |   --namespace "$NAMESPACE" \
28 |   --set alertmanager.persistentVolume.enabled=true \
29 |   --set server.persistentVolume.enabled=true
30
31 echo "✅ Instalación de Prometheus completada."
```

Chequeo de despliegue de Prometheus

Como se mencionó antes, se puede pedir información de los pods que pertenecen al namespace **prometheus**. Haciendo un port-forward al pod **prometheus-server**, se puede abrir temporalmente una conexión a la interfaz web del mismo para asegurarnos que está funcionando bien. Esos avisos que se ven en tiempo real son las conexiones que realiza el navegador web.

```
mauro@xu24-map-latitude-7390:~/repos/map/PIN_FINAL$ kubectl get pods -n prometheus
NAME                                READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-0           1/1     Running   0           98m
prometheus-kube-state-metrics-5c7f9cf685-95g9c  1/1     Running   0           98m
prometheus-prometheus-node-exporter-7jkrl      1/1     Running   0           98m
prometheus-prometheus-node-exporter-wb9br      1/1     Running   0           98m
prometheus-prometheus-node-exporter-wn8kt      1/1     Running   0           98m
prometheus-prometheus-pushgateway-79964b5788-dfwzl  1/1     Running   0           98m
prometheus-server-748d4668d-t46xl             2/2     Running   0           98m
mauro@xu24-map-latitude-7390:~/repos/map/PIN_FINAL$ kubectl port-forward -n prometheus pod/prometheus-server-748d4668d-t46xl 8080:9090 --address 0.0.0.0
Forwarding from 0.0.0.0:8080 -> 9090
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

Endpoint	Labels
----------	--------

Chequeo de despliegue de Prometheus

La dirección es localhost:8080. Se observa que Prometheus está scrapeando métricas de los nodos del clúster y monitoreando tanto el estado del clúster como la API de Kubernetes.

The screenshot shows the Prometheus web interface at localhost:8080/targets. The interface includes a navigation bar with 'Prometheus', 'Query', 'Alerts', and 'Status > Target health'. Below the navigation bar, there are filters for 'Select scrape pool', 'Filter by target health', and 'Filter by endpoint or labels'. The main content area displays two sections: 'kubernetes-apiservers' and 'kubernetes-nodes'. Each section shows a table of targets with columns for 'Endpoint', 'Labels', 'Last scrape', and 'State'.

Endpoint	Labels	Last scrape	State
https://192.168.121.43/metrics	instance="192.168.121.43:443" job="kubernetes-apiservers"	25.481s ago 308ms	UP
https://192.168.185.153/metrics	instance="192.168.185.153:443" job="kubernetes-apiservers"	54.342s ago 160ms	UP

Endpoint	Labels	Last scrape	State
https://kubernetes.default.svc/api/v1/nodes/ip-192-168-83-39.ec2.internal/proxy/metrics	alpha_eksctl_io_cluster_name="eks-grupo-2" alpha_eksctl_io_nodegroup_name="ng-grupo-2" beta_kubernetes_io_arch="amd64" beta_kubernetes_io_instance_type="t3.small" beta_kubernetes_io_os="linux" eks_amazonaws_com_capacityType="ON_DEMAND" eks_amazonaws_com_nodegroup="ng-grupo-2" eks_amazonaws_com_nodegroup_image="ami-060ebf819759fa062" eks_amazonaws_com_sourceLaunchTemplateId="lt-032b816398aa299eb" eks_amazonaws_com_sourceLaunchTemplateVersion="1" failure_domain_beta_kubernetes_io_region="us-east-1" failure_domain_beta_kubernetes_io_zone="us-east-1c" instance="ip-192-168-83-39.ec2.internal" job="kubernetes-nodes" k8s_io_cloud_provider_aws="55525ae3d63c058272be8d51b601c486" kubernetes_io_arch="amd64" kubernetes_io_hostname="ip-192-168-83-39.ec2.internal" kubernetes_io_os="linux" node_kubernetes_io_instance_type="t3.small" topology_ebs_csi_aws_com_zone="us-east-1c" topology_k8s_aws_zone_id="use1-az1" topology_kubernetes_io_region="us-east-1" topology_kubernetes_io_zone="us-east-1c"	33.097s ago 85ms	UP
https://kubernetes.default.svc/api/v1/nodes/ip-192-168-19-105.ec2.internal/proxy/metrics	alpha_eksctl_io_cluster_name="eks-grupo-2" alpha_eksctl_io_nodegroup_name="ng-grupo-2" beta_kubernetes_io_arch="amd64" beta_kubernetes_io_instance_type="t3.small" beta_kubernetes_io_os="linux" eks_amazonaws_com_capacityType="ON_DEMAND" eks_amazonaws_com_nodegroup="ng-grupo-2" eks_amazonaws_com_nodegroup_image="ami-060ebf819759fa062" eks_amazonaws_com_sourceLaunchTemplateId="lt-032b816398aa299eb" eks_amazonaws_com_sourceLaunchTemplateVersion="1" failure_domain_beta_kubernetes_io_region="us-east-1" failure_domain_beta_kubernetes_io_zone="us-east-1a" instance="ip-192-168-19-105.ec2.internal" job="kubernetes-nodes" k8s_io_cloud_provider_aws="55525ae3d63c058272be8d51b601c486" kubernetes_io_arch="amd64" kubernetes_io_hostname="ip-192-168-19-105.ec2.internal" kubernetes_io_os="linux" node_kubernetes_io_instance_type="t3.small" topology_ebs_csi_aws_com_zone="us-east-1c" topology_k8s_aws_zone_id="use1-az1" topology_kubernetes_io_region="us-east-1" topology_kubernetes_io_zone="us-east-1c"	10.245s ago 89ms	UP

Despliegue de Grafana

Se despliega **Grafana** con dos archivos. El script **grafana_deploy.sh** que utiliza **kubect1** para crear en namespace y llamar a **helm**. Este último utiliza el archivo de configuración de valores personalizados **grafana.yaml**.

```
monitoreo > $ grafana_deploy.sh

# Verificar si el archivo de configuración grafana.yml existe
if [ ! -f "$GRAFANA_VALUES" ]; then
    echo "Error: El archivo grafana.yaml no existe en la ruta $GRAFANA_VALUES"
    exit 1
fi

# Verificar si el script fue llamado con --delete
if [[ "$1" == "--delete" ]]; then
    echo "Eliminando todos los recursos de Grafana..."

    # Eliminar cualquier instalación previa de Grafana
    helm uninstall grafana -n $NAMESPACE

    # Eliminar el namespace
    kubect1 delete namespace $NAMESPACE

    exit 0
fi

# Comprobar si el namespace grafana ya existe, si no, crearlo
if ! kubect1 get namespace $NAMESPACE &>/dev/null; then
    kubect1 create namespace $NAMESPACE
else
    echo "El namespace $NAMESPACE ya existe."
fi

# Instalar Grafana usando Helm y el archivo de configuración local
helm install grafana grafana/grafana \
    --namespace $NAMESPACE \
    --set persistence.enabled=true \
    --set adminPassword="$ADMIN_PASSWORD" \
    --values "$GRAFANA_VALUES" \
    --set service.type=LoadBalancer
```

```
02_monitoreo > ! grafana.yaml
1  datasources:
2    datasources.yaml:
3      apiVersion: 1
4      datasources:
5        - name: Prometheus
6          type: prometheus
7          url: http://prometheus-server.prometheus.svc.cluster.local
8          access: proxy
9          isDefault: true
```

Chequeo de despliegue de Grafana

Nuevamente se puede utilizar **kubectl** para obtener información, en este caso de todo lo relacionado al namespace **grafana**. Se creo un pequeño script **get-grafana-info.sh** el cual devuelve la dirección URL y contraseña del servicio para poder conectarse con el navegador web, útil para el paso final.

```
02_monitoreo > $ get-grafana-info.sh
1  #!/bin/bash
2
3  # Obtener el hostname del LoadBalancer de Grafana
4  ELB=$(kubectl get svc -n grafana grafana -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')
5
6  # Verificar si se obtuvo un resultado válido
7  if [[ -z "$ELB" ]]; then
8      echo "⚠ No se pudo obtener el hostname de Grafana. Puede que el LoadBalancer aún esté aprovisionando."
9      exit 1
10 fi
11
12 # Obtener la contraseña del usuario admin
13 ADMIN_PASSWORD=$(kubectl get secret --namespace grafana grafana -o jsonpath="{.data.admin-password}" | base64 --decode)
14
15 # Mostrar la URL de acceso y la contraseña de Grafana
16 echo "🔗 URL de Grafana: http://$ELB"
17 echo "🔑 Contraseña de admin: $ADMIN_PASSWORD"
18 |
```

```
mauro@xu24-map-latitude-7390:~/repos/map/PIN_FINAL$ kubectl get pods,svc,pvc -n grafana
```

NAME	Connections	READY	STATUS	RESTARTS	AGE
pod/grafana-85587967bc-4hfzw		1/1	Running	0	12m

```
mauro@xu24-map-latitude-7390:~/repos/map/PIN_FINAL$ kubectl get svc -n grafana
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/grafana	LoadBalancer	10.100.161.204	af152bedc5efc4f888ebd400af8d3f08-675295343.us-east-1.elb.amazonaws.com	80:30973/TCP	12m

```
mauro@xu24-map-latitude-7390:~/repos/map/PIN_FINAL$ ./05_monitoreo_alt/get-grafana-info.sh
```

```
🔗 URL de Grafana: http://af152bedc5efc4f888ebd400af8d3f08-675295343.us-east-1.elb.amazonaws.com
```

```
🔑 Contraseña de admin: grupo-02
```

```
mauro@xu24-map-latitude-7390:~/repos/map/PIN_FINAL$
```

Chequeo de despliegue de Grafana

Grafana sin dashboards aún.

The screenshot shows the Grafana web interface in a browser. The address bar displays the URL: `af152bedc5efc4f888ebd400af8d3f08-675295343.us-east-1.elb.amazonaws.com/?orgId=1&from=now-6h&to=now&timezone=browser`. The interface has a dark theme. On the left is a sidebar with the Grafana logo and navigation links: Home, Bookmarks, Starred, Dashboards, Explore, Alerting, Connections, and Administration. The main content area has a header with 'Welcome to Grafana' and links for 'Need help?' (Documentation, Tutorials, Community, Public Slack). Below this is a 'Basic' section with a tutorial titled 'TUTORIAL DATA SOURCE AND DASHBOARDS Grafana fundamentals'. The tutorial text says: 'The steps below will guide you to quickly finish setting up your Grafana installation. Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.' There are two buttons: 'Add your first data source' and 'Create your first dashboard', both with links to 'Learn how in the docs'. A 'Remove this panel' link is also present. At the bottom, there is a 'Dashboards' section with 'Starred dashboards' and 'Recently viewed dashboards', and a 'Latest from the blog' section with a post dated 'mar. 03' titled 'How to monitor your Shopify store with Grafana Cloud Frontend Observability'.

Home

Search or jump to... ctrl+k

Welcome to Grafana

Need help? [Documentation](#) [Tutorials](#) [Community](#) [Public Slack](#)

Basic

The steps below will guide you to quickly finish setting up your Grafana installation.

TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

COMPLETE
Add your first data source
Learn how in the docs

DASHBOARDS
Create your first dashboard
Learn how in the docs

[Remove this panel](#)

Dashboards

Starred dashboards

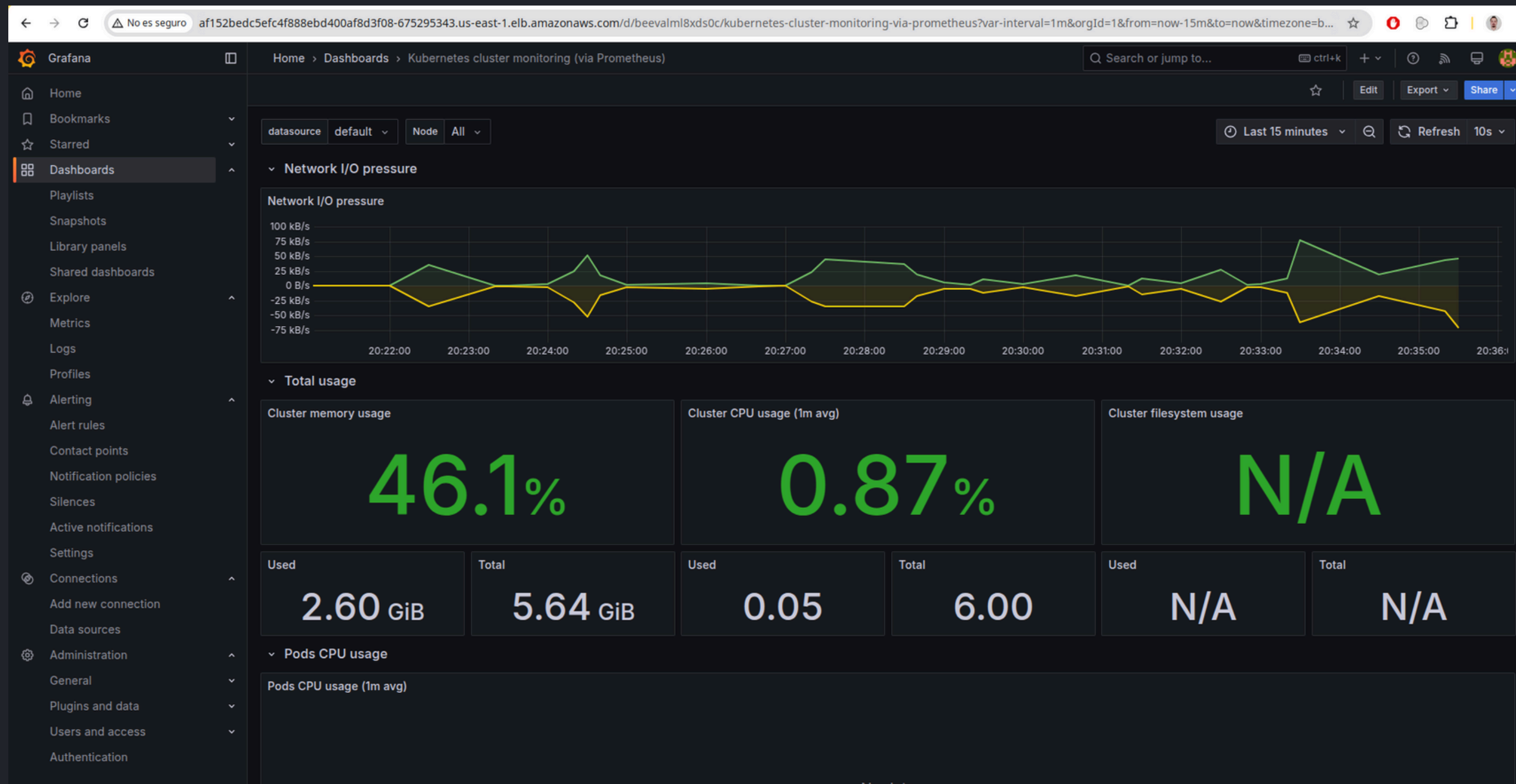
Recently viewed dashboards

Latest from the blog

mar. 03
How to monitor your Shopify store with Grafana Cloud Frontend Observability
Shopify is a fantastic tool for organizations who want to sell products, but don't want to build or maintain an e-commerce platform themselves. Even some of the largest brands that have built

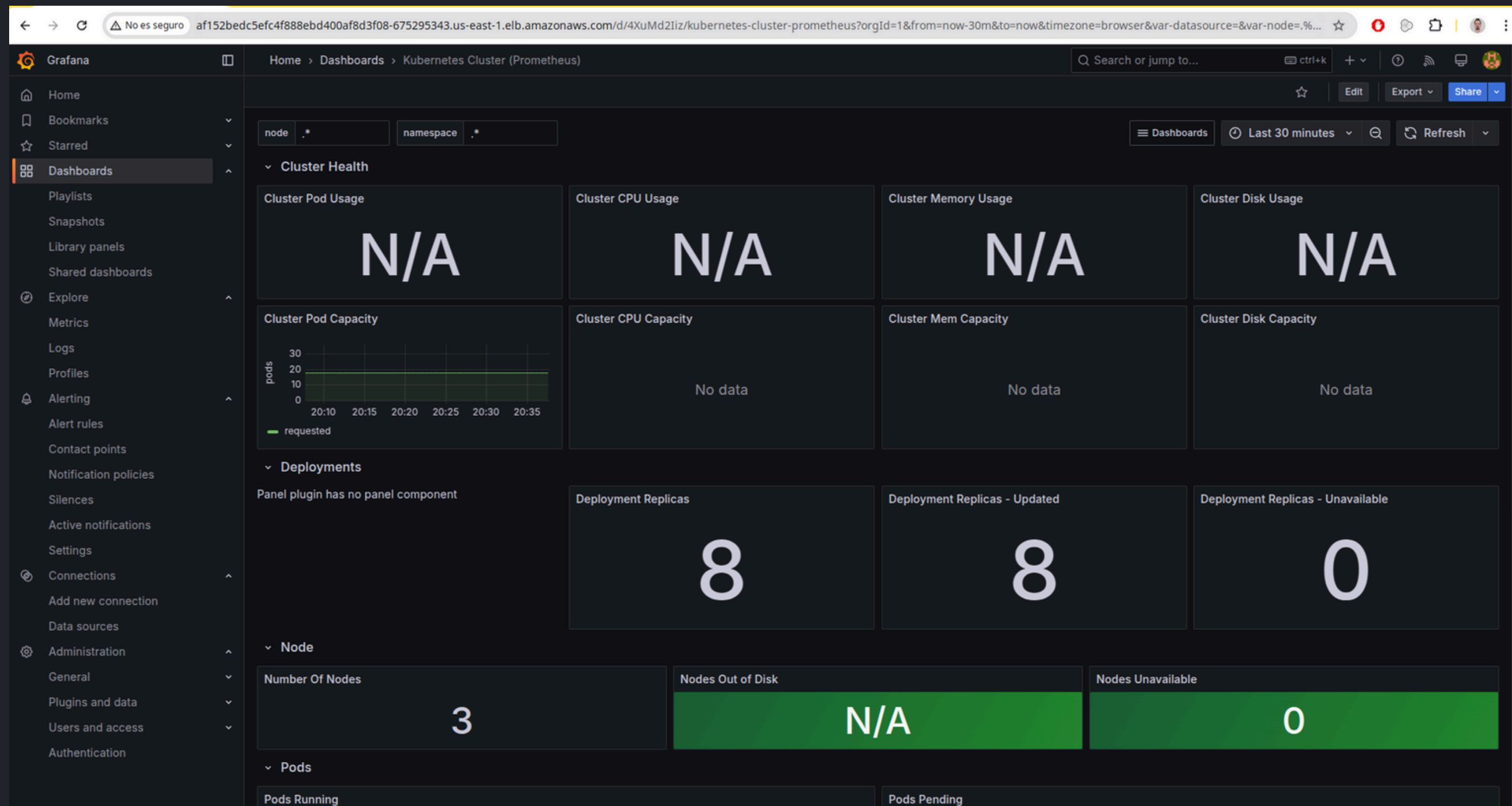
Configuración de dashboard

Unas de las ventajas de usar Grafana es que se pueden importar dashboards con parámetros predefinidos, esto se logra clickeando en New dashboard -> Import. Especificando el número 3119 y seleccionando a **Prometheus** como datasource se obtiene el **Cluster Monitoring Dashboard**.



Configuración de dashboard

Repitiendo los pasos pero esta vez agregando el número 6417 se obtiene el **Pods Monitoring Dashboard**.



Conclusión

- Elegir una buena herramienta de monitoreo facilita mucho el despliegue del mismo. La importación de dashboard nos permite ganar tiempo, sin embargo al tener configuraciones predefinidas es necesario retocar algunas, por eso algunos indicadores figuran como N/A.
- Es imprescindible tener un buen manejo de costos a la hora de probar diferentes despliegues, estar muy atentos a **Cost Explorer** de **AWS** y configurar las alertas.
- Se debe controlar que se hayan eliminado los clusteres si no se usan más o no se van a usar por un buen tiempo, mantener limpia la nube, para no incurrir en costos desmedidos. Siempre incluir algún algoritmo que se encargue de la eliminación, en nuestro caso el parámetro **--delete** cumple dicho propósito. Si hubiéramos usado **terraform** tendríamos que ejecutar la opción **destroy**.
- Otra buena práctica tener presente los permisos asignados a los usuarios. Tratar de siempre darles los mínimos privilegios, manteniendo políticas ordenadas y limpias.