

Proyecto Integrador Final - DevOps

Objetivo del Proyecto

El objetivo de este trabajo es aplicar diferentes herramientas y tecnologías de DevOps mediante un laboratorio práctico, integrando conceptos clave como:

- **Gestión de infraestructura:** Configuración de Kubernetes con Minikube.
 - **Automatización:** Uso de `kubectl` y `helm` para la gestión de aplicaciones.
 - **Despliegue de aplicaciones:** Implementación de un servidor NGINX en Kubernetes.
 - **Monitoreo y observabilidad:** Uso de Prometheus y Grafana.
 - **Resolución de problemas:** Solución de errores en la configuración del clúster.
 - **Gestión del ciclo de vida de los recursos:** Creación, configuración y eliminación de componentes.
-

Paso 1: Instalación de herramientas

Antes de comenzar, debemos instalar las siguientes herramientas:

1.1 Instalar Minikube

- `curl -LO`
`https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64`
- `sudo install minikube-linux-amd64 /usr/local/bin/minikube`

1.2 Instalar kubectl

- `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/`

1.3 Instalar Docker

- `sudo apt update`
- `sudo apt install -y docker.io`

1.4 Instalar Helm

- `curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash`

si estamos en mac:

- *brew install minikube kubectl docker helm*

```
gcuenya@Guillermos-MacBook-Pro Projects % brew install minikube kubectl docker helm

==> Auto-updating Homebrew...
Adjust how often this is run with HOMEBREW_AUTO_UPDATE_SECS or disable with
HOMEBREW_NO_AUTO_UPDATE. Hide these hints with HOMEBREW_NO_ENV_HINTS (see 'man brew').
==> Auto-updated Homebrew!
Updated 2 taps (homebrew/core and homebrew/cask).
==> New Formulae
adapterremoval  code2prompt      exomizer          garnet            identme           libpostal         nping            rustic            symfony-cli       xk6
aqua            cspell           fancy-cat         git-mob           infisical         libpostal-rest    nuitka           rustywind         taskflow          yamlfix
arello          dbg-macro        feluda            gowall            jira-cli          mac               mdpdf           pdfly             sitefetch         tgpt
baton-is        dockerfilegraph  flow-control      havener           jupyter           mdo               ratarmount       soft-serve        snowflake-cli     vkit
bazel07         dtoroll          foundry           hk                keeper-commander  mummer            reuse            sql-formatter     visidata          vscli
bombarrier      cloudfoundry-cli dyff              fricas            koji              nak               ruby-lsp         sv2v              wfs2-lib
==> New Casks
automounterhelper  font-big-shoulders-stencil  granola              obscura-vpn          ui-tars
badgeify           font-boldonse               istatistica-core    opera-air            vezer
batfi              font-bytesized              jumpcloud-password-manager  pairpods            windsurf@next
block-goose        font-comic-relief           kunlun               pdl
font-aporetic       font-sf-mono-nerd-font-ligaturized  losslesswitcher
font-big-shoulders  fuse-t                      luanti
font-big-shoulders-inline  gologin                    mitti

You have 10 outdated formulae installed.

Warning: Trusting docker as a formula. For the cask, use homebrew/cask/docker or specify the '--cask' flag. To silence this message, use the '--formula' flag.
==> Downloading https://ghcr.io/v2/homebrew/core/minikube/manifests/1.35.0
==> Fetching dependencies for minikube: kubernetes-cli
==> Downloading https://ghcr.io/v2/homebrew/core/kubernetes-cli/manifests/1.32.2
==> Fetching kubernetes-cli
==> Downloading https://ghcr.io/v2/homebrew/core/kubernetes-cli/blobs/sha256:aaf45466833d93aec0e1a938d68d8bbde124a9005b79c1273be0aa8e1c104169
==> Downloading https://ghcr.io/v2/homebrew/core/minikube/blobs/sha256:3b4fdb423c0e9a216cc41eb9330550b5e23b6e4c069df40b2635fe99a2a4d1a9
==> Downloading https://ghcr.io/v2/homebrew/core/kubernetes-cli/manifests/1.32.2
Already downloaded: /Users/gcuenya/Library/Caches/Homebrew/downloads/199ec85b43a5f338fac8ff6b2bf65685bf71643a9a16ca3c96f62558118d--kubernetes-cli-1.32.2.bottle_manifest.json
==> Downloading https://ghcr.io/v2/homebrew/core/docker/manifests/28.0.0
==> Fetching dependencies for docker
```

Paso 2: Iniciar Minikube

Ejecuta el siguiente comando para iniciar el clúster local:

- *minikube start --driver=docker*

```
gcuenya@Guillermos-MacBook-Pro PINFinal-Diplo % minikube start --driver=docker
minikube v1.35.0 on Darwin 14.5
Using the docker driver based on user configuration
Using Docker Desktop driver with root privileges
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.46 ...
Creating docker container (CPUs=2, Memory=4000MB) ...
Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
Configuring bridge CNI (Container Networking Interface) ...
Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: default-storageclass, storage-provisioner

! /Applications/Docker.app/Contents/Resources/bin/kubectl is version 1.30.5, which may have incompatibilities with Kubernetes 1.32.0.
  - Want kubectl v1.32.0? Try 'minikube kubectl -- get pods -A'
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
gcuenya@Guillermos-MacBook-Pro PINFinal-Diplo % minikube dashboard
```

Containers [Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage ⓘ

38.30% / 800% (8 CPUs available)

Container memory usage ⓘ

436.9MB / 7.48GB

Show charts

🔍 Search






☰

☐ Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Memory (%)	Actions
<input type="checkbox"/>	minikube	a5e6adee251e	k8s-minikul	64143:22 ↗ Show all ports (5)	38.3%	2	<div> <div></div> <div>:</div> <div></div> </div> <div></div>

Containers / minikube

minikube

<  a5e6adee251e   [k8s-minikube/kicbase](#) (was [k8s-minikube/kicbase:v0.0](#)) **STATUS**
64143:22  64144:2376  [Show all ports \(5\)](#) Running (47 seconds ago)

Logs Inspect Bind mounts Exec Files Stats

```
2025-03-01 09:17:41 Starting containerd container runtime...
2025-03-01 09:17:41 Starting minikube automount...
2025-03-01 09:17:41 Starting Podman auto-update service...
2025-03-01 09:17:41 Starting Podman Start All ...estart Policy Set To Always...
2025-03-01 09:17:41 Starting Podman API Service...
2025-03-01 09:17:41 Starting OpenBSD Secure Shell server...
2025-03-01 09:17:41 [ OK ] Started Podman API Service.
2025-03-01 09:17:41 [ OK ] Finished minikube automount.
2025-03-01 09:17:41 [ OK ] Started OpenBSD Secure Shell server.
2025-03-01 09:17:42 [ OK ] Started containerd container runtime.
2025-03-01 09:17:42 Starting Docker Application Container Engine...
2025-03-01 09:17:42 [ OK ] Finished Podman Start All ... Restart Policy Set To Always.
2025-03-01 09:17:42 [ OK ] Finished Podman auto-update service.
2025-03-01 09:17:42 [ OK ] Started Docker Application Container Engine.
2025-03-01 09:17:42 [ OK ] Reached target Multi-User System.
2025-03-01 09:17:42 [ OK ] Reached target Graphical Interface.
2025-03-01 09:17:42 Starting Record Runlevel Change in UTMP...
2025-03-01 09:17:42 [ OK ] Finished Record Runlevel Change in UTMP.
```

- *minikube dashboard*

Verificamos que el clúster esté activo:

- *kubectl get nodes*

```
gcuenya@Guillermos-MacBook-Pro ~ % kubectl get nodes
NAME          STATUS    ROLES          AGE   VERSION
minikube      Ready     control-plane   10m   v1.32.0
```

Paso 3: Desplegar NGINX

Crear el despliegue:

- *kubectl create deployment nginx --image=nginx*

Exponer el servicio:

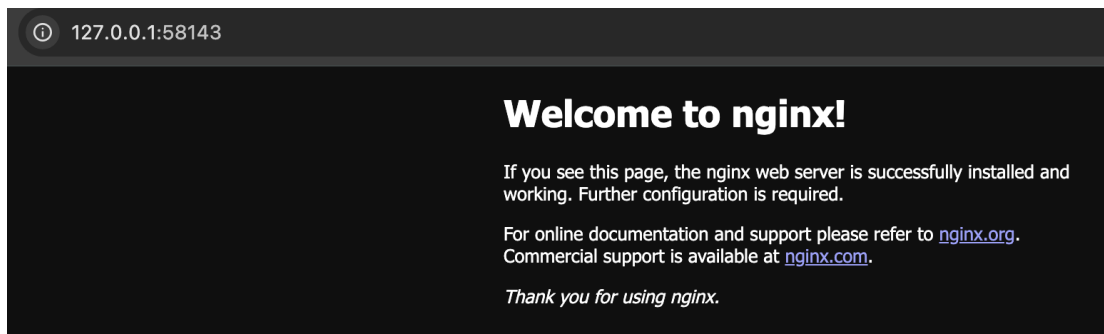
- *kubectl expose deployment nginx --type=NodePort --port=80*

```
gcuenya@Guillermos-MacBook-Pro ~ % kubectl create deployment nginx --image=nginx
deployment.apps/nginx created
gcuenya@Guillermos-MacBook-Pro ~ % kubectl expose deployment nginx --type=NodePort --port=80
service/nginx exposed
```

Obtener la URL de acceso:

- *minikube service nginx --url*

Vemos que nginx esta corriendo:



Paso 4: Instalación de Prometheus y Grafana

4.1 Agregamos los repositorios de Helm

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

4.2 Instalamos Prometheus

- *kubectl create namespace monitoring*
- *helm install prometheus prometheus-community/prometheus --namespace monitoring*

Verificamos los pods:

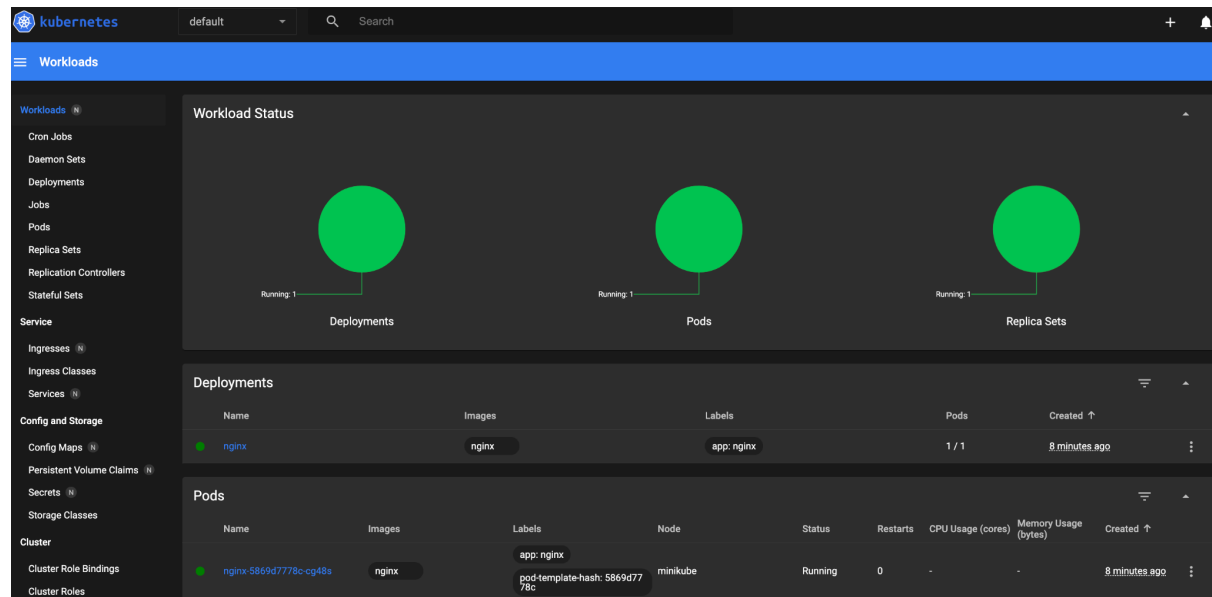
- *kubectl get pods -n monitoring*

```
gcuenya@Guillermos-MacBook-Pro ~ % kubectl get pods -n monitoring

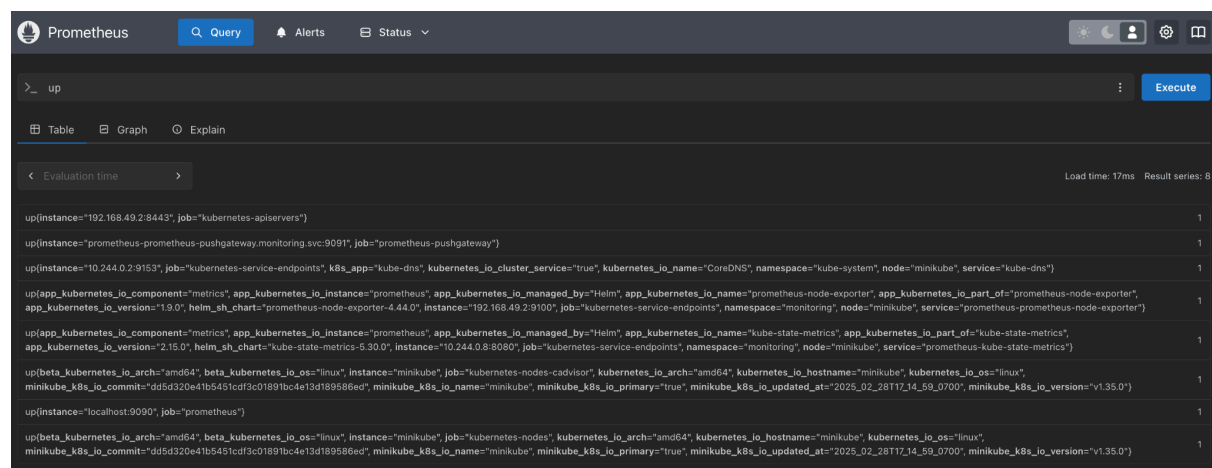
NAME                                READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-0          1/1     Running   0           78s
prometheus-kube-state-metrics-5bd466f7f6-2zm22  1/1     Running   0           79s
prometheus-prometheus-node-exporter-6g92d    1/1     Running   0           79s
prometheus-prometheus-pushgateway-544579d549-slbz9  1/1     Running   0           79s
prometheus-server-596945876b-s7vr8          1/2     Running   0           79s
```

Para acceder a la interfaz de Prometheus:

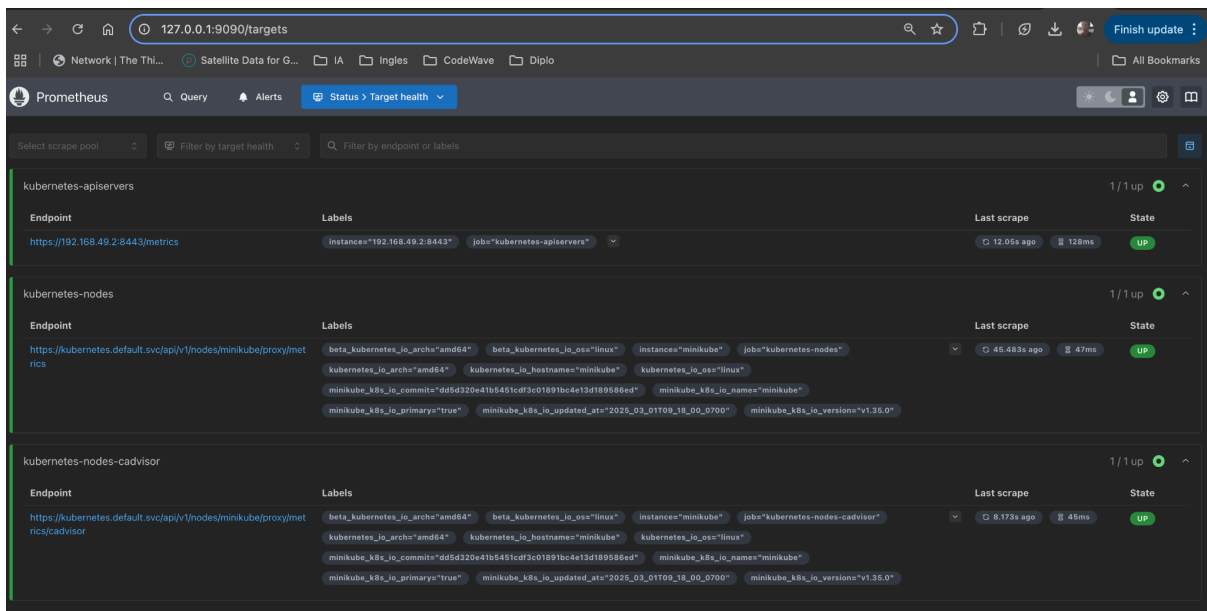
- `kubectl port-forward -n monitoring svc/prometheus-server 9090:80`



Accediendo a <http://localhost:9090>



Vemos que todos los endpoints están UP



4.3 Instalamos Grafana

- `kubectl create namespace grafana`
- `helm install grafana grafana/grafana --namespace grafana --set service.type=NodePort`
<http://localhost:3000/dashboards>

Verificamos que Grafana esté corriendo:

- `kubectl get pods -n grafana`

```
nginx-5869d7778c-cg48s    1/1    Running    0        11m
gcuenya@Guillermos-MacBook-Pro ~ % kubectl get pods -n grafana
NAME                                READY   STATUS    RESTARTS   AGE
grafana-96bf44858-1466g           1/1    Running    0          47s
```

Obtenemos la url de grafana:

- `minikube service -n grafana grafana --url`

Para obtener la contraseña de Grafana:

- `kubectl get secret --namespace grafana grafana -o jsonpath="{.data.admin-password}" | base64 --decode`

Para acceder a Grafana:

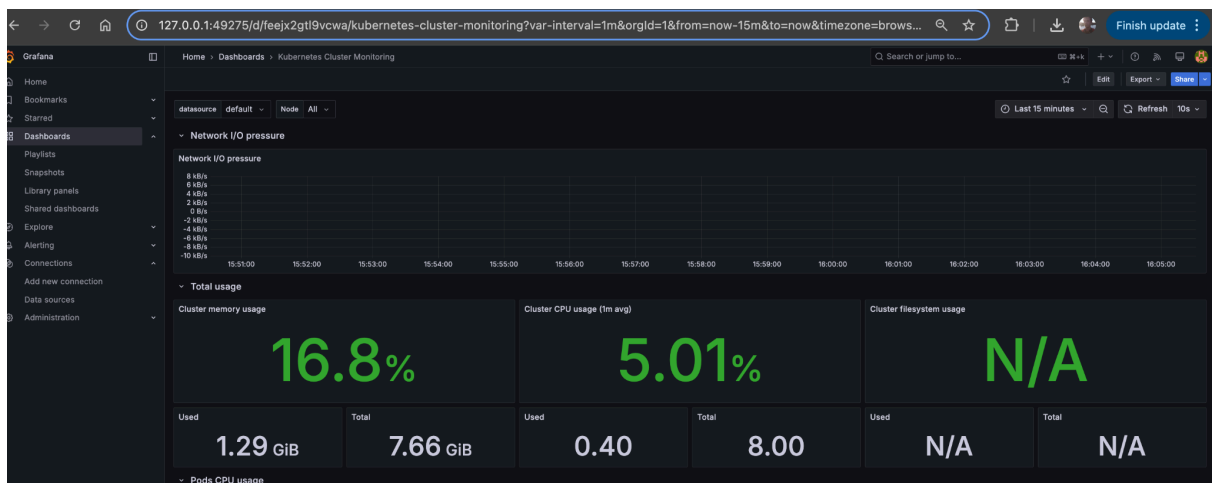
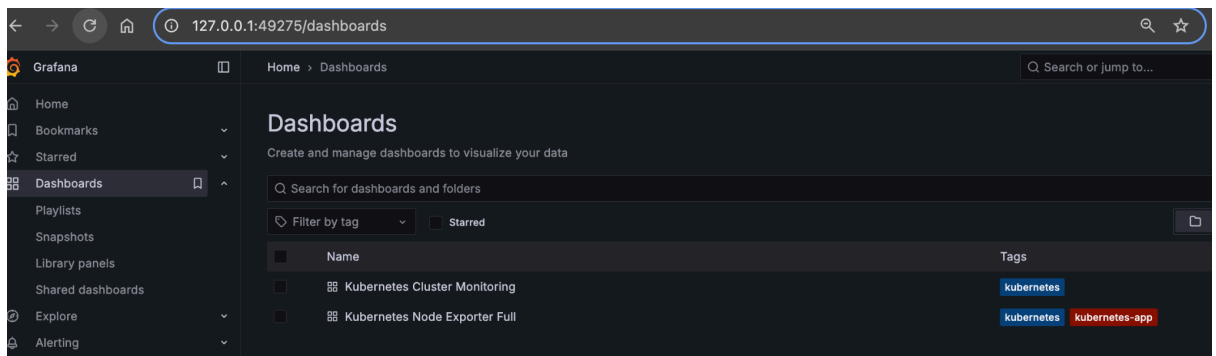
- `kubectl port-forward -n grafana svc/grafana 3000:80`

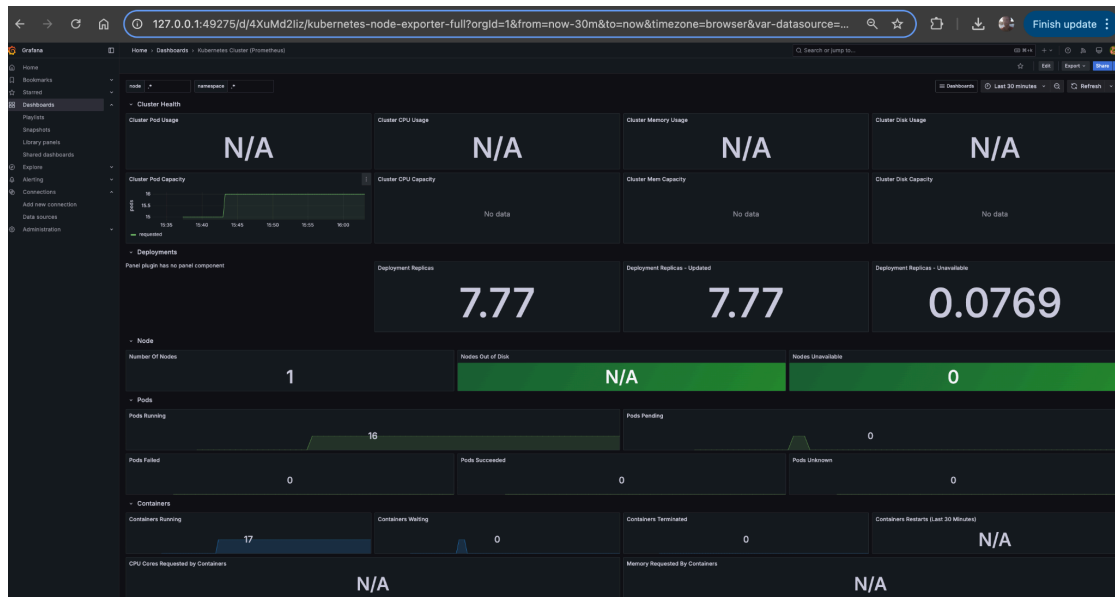
En el navegador, accedemos a Grafana (<http://localhost:3000>) e iniciamos sesión con:

- **Usuario:** admin
- **Contraseña:** (la obtenida en el comando anterior - TTXuJeJpfDXng91jeJTRyZH3rXdFyC8bzYJMvBe)

Paso 5: Importar Dashboards en Grafana

1. Iniciamos sesión en Grafana.
2. En **Dashboards > Importar**.
3. Introducimos los siguientes IDs:
 - **3119** → Kubernetes Cluster Monitoring
 - **6417** → Kubernetes Node Exporter Full
4. Configuramos la fuente de datos como **Prometheus**.
5. Guarda y visualiza las métricas.





Paso 6: Cleanup de Recursos

Para eliminar todos los recursos creados:

Eliminar Prometheus y Grafana

```
helm uninstall prometheus --namespace monitoring
```

```
kubectl delete ns monitoring
```

```
helm uninstall grafana --namespace grafana
```

```
kubectl delete ns grafana
```

Eliminar NGINX

```
kubectl delete deployment nginx
```

```
kubectl delete service nginx
```

Detener y eliminar Minikube

```
minikube stop
```

```
minikube delete
```

Conclusión

Este laboratorio práctico permitió la implementación y configuración de un entorno Kubernetes local utilizando Minikube, junto con herramientas clave para la observabilidad y monitoreo del clúster. A lo largo del desarrollo, se abordaron conceptos fundamentales de DevOps, incluyendo la automatización de despliegues, la gestión de infraestructura como código y la supervisión de servicios en un entorno de contenedores.

Principales Logros

✓ Configuración de Kubernetes con Minikube

Se instaló y configuró Minikube como un clúster local de Kubernetes, proporcionando una plataforma de pruebas y desarrollo sin depender de entornos en la nube. Se aseguraron los complementos esenciales, como metrics-server, para habilitar la recopilación de métricas.

✓ Despliegue de NGINX como aplicación de prueba

Se desplegó un servidor web NGINX en Kubernetes utilizando kubectl, lo que permitió comprender el manejo de Deployments, Pods y Services dentro del clúster. Se expuso el servicio correctamente para que fuera accesible desde el host local.

✓ Implementación de monitoreo con Prometheus y Grafana

Se integró un stack de monitoreo en Kubernetes utilizando Helm, con Prometheus para la recopilación de métricas y Grafana para su visualización. Se configuraron Node Exporter y Kube State Metrics para obtener información detallada sobre el estado del clúster y los recursos utilizados por los contenedores y nodos.

✓ Visualización y análisis de métricas del clúster

Se implementaron dashboards preconfigurados en Grafana para monitorear la infraestructura en tiempo real. A través de consultas en Prometheus, se validó la disponibilidad de servicios y el consumo de recursos, permitiendo detectar posibles cuellos de botella o fallas en los componentes del clúster.

Desafíos y Soluciones

⚠ Acceso a servicios internos de Kubernetes

Inicialmente, los servicios de Prometheus y Grafana estaban configurados como ClusterIP, lo que impedía su acceso externo. Para solucionar esto, se modificaron los servicios a NodePort, permitiendo su exposición en puertos accesibles desde el host.

⚠ Errores en la inicialización de pods

Algunos pods, como los de Prometheus, quedaron en estado Pending o CrashLoopBackOff debido a la falta de recursos o configuraciones incorrectas. Esto se resolvió revisando los logs de los pods y asegurando que el clúster de Minikube tuviera suficiente memoria asignada.

⚠ Persistencia de datos en Grafana

La configuración inicial no incluía persistencia, lo que provocaba la pérdida de dashboards al reiniciar el clúster. Para evitar esto, se debería agregar una configuración de PersistentVolumeClaim en futuras iteraciones.

📌 Conocimientos Aplicados

- ♦ **Infraestructura como Código:** Se utilizó Helm para gestionar instalaciones complejas de servicios en Kubernetes.
- ♦ **Gestión de Contenedores:** Se trabajó con kubectl y helm para desplegar y administrar

pods, servicios y recursos del clúster.

- ♦ **Monitoreo y Observabilidad:** Se integraron herramientas como Prometheus y Grafana para la supervisión de recursos y rendimiento.

- ♦ **Automatización:** Se desarrolló un script en Bash para desplegar todo el stack de forma automatizada, asegurando consistencia y eficiencia en el proceso de instalación.