



Proyecto integrador DevOps

Grupo 10

Integrantes

- Marquinho Moya
- Carlos Navarro
- Gabriel Barrera
- Federico Pascarella
- Julio Sejas

Requisitos

- aws cli
- kubectl
- eksctl
- helm

Planificación del Cluster EKS

- Numero de nodos: 3
- Tipo de instancia: t3.small
- Region: us-east-1
- Zonas:
 - us-east-1a
 - us-east-1b
 - us-east-1c

Configuración del Entorno Local para Desplegar un Cluster de EKS en AWS

| OS: Linux, Ubuntu 24.04.1 LTS

Instalar los paquete necesarios en requisitos

- aws cli

Se ejecuto lo siguiente

```
sudo apt install unzip -y
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscli.zip"
unzip awscli.zip
sudo ./aws/install
```

al finalizar se debería ver lo siguiente

```
inflating: aws/dist/docutils/writers/latex2e/titlingpage.tex
inflating: aws/dist/docutils/writers/latex2e/default.tex
inflating: aws/dist/docutils/writers/latex2e/docutils.sty
inflating: aws/dist/docutils/writers/html4css1/html4css1.css
inflating: aws/dist/docutils/writers/html4css1/template.txt
inflating: aws/dist/docutils/writers/odf_odt/styles.odt
You can now run: /usr/local/bin/aws --version
```

Y para verificar ejecutamos

```
aws --version
```

Debería imprimir lo siguiente

```
jsm@JSM-AMD:~$ aws --version  
aws-cli/2.24.16 Python/3.12.9 Linux/5.15.167.4-microsoft-standard-WSL2 exe/x86_64.ubuntu.24
```

- Kubectl

Se ejecuto lo siguiente

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2024-12-20/bin/linux/amd64/kubectl  
chmod +x ./kubectl  
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH  
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Ahora ejecutamos lo siguiente para verificar que esta todo correcto

```
kubectl version --client
```

Debería imprimir lo siguiente

```
jsm@JSM-AMD:~$ kubectl version --client  
Client Version: v1.32.0-eks-5ca49cb  
Kustomize Version: v5.5.0
```

- eksctl

Se ejecuto lo siguiente

```
# for ARM systems, set ARCH to: `arm64`, `armv6` or `armv7`  
ARCH=amd64  
PLATFORM=$(uname -s)_$ARCH  
  
curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_${PLATFORM}.tar.gz"  
  
# (Optional) Verify checksum  
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.txt" | grep $PLATFORM | sha256sum --check  
  
tar -xzf eksctl_${PLATFORM}.tar.gz -C /tmp && rm eksctl_${PLATFORM}.tar.gz
```

```
sudo mv /tmp/eksctl /usr/local/bin
```

Ahora ejecutamos lo siguiente para verificar que esta todo correcto

```
eksctl info
```

Debería imprimir lo siguiente

```
jsm@JSM-AMD:~$ eksctl info
eksctl version: 0.205.0
kubectl version: v1.32.0-eks-5ca49cb
OS: linux
```

- Helm

Se ejecuto lo siguiente

```
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /
usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https --yes
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/ke
yrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | s
udo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm -y
```

Ahora ejecutamos lo siguiente para verificar que esta todo correcto

```
helm version
```

Debería imprimir lo siguiente


```
jsm@JSM-AMD:~$ helm version
version.BuildInfo{Version:"v3.17.1", GitCommit:"980d8ac1939e3913810136440756af2bdee1da5", GitTreeState:"clean", GoVersi
on:"go1.23.5"}
```

Configurar AWS

Ahora vamos a configurar el AWS cli con las credenciales. Para mas información consulte

Setting up the AWS CLI - AWS Command Line Interface

The AWS CLI is an open source tool built using the AWS SDK for Python (Boto) that provides commands for interacting with AWS services. With minimal configuration, you can start using

 <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html>



Una vez que tenemos el KEY ID y el ACCESS KEY, lo configuramos de la siguiente manera

```
jsm@JSM-AMD:~$ aws configure
AWS Access Key ID [None]: AKIA47CR[REDACTED]
AWS Secret Access Key [None]: kjPV/e6qAhg7Yp6ZAXiWQ[REDACTED]
Default region name [None]: us-east-1
Default output format [None]: json
```

Por seguridad pintamos las keys

Una vez que tenemos configurado el AWS cli podemos verificar con el siguiente comando

```
aws sts get-caller-identity
```

Debería imprimir

```
jsm@JSM-AMD:~$ aws sts get-caller-identity
{
  "UserId": "A[REDACTED]A2",
  "Account": "8[REDACTED]34",
  "Arn": "arn:aws:iam::8[REDACTED]34:user/cloud_user"
}
```

Con esto tenemos todo configurado para continuar

Crear el Cluster EKS

Creando Key Pair

Antes de crear en cluster necesitamos crear un ssh-key y subirlo a AWS.

Creamos una carpeta en \$HOME y generamos la key

Se ejecuto lo siguiente

```
mkdir $HOME/eks && cd $HOME/eks && ssh-keygen -t rsa -b 4096 -C "eks-ssh" -f ./eks-ssh -N ""
```

Una ves finalizado debería imprimir los siguiente y con  verificamos que este

```
jsm@JSM-AMD:~$ mkdir $HOME/eks && cd $HOME/eks && ssh-keygen -t rsa -b 4096 -C "eks-ssh" -f ./eks-ssh -N ""
Generating public/private rsa key pair.
Your identification has been saved in ./eks-ssh
Your public key has been saved in ./eks-ssh.pub
The key fingerprint is:
SHA256:fLPCQV+5pWlLXE0YkFMv+UmgaUsHDrLPgILEtDemx8 eks-ssh
The key's randomart image is:
+---[RSA 4096]-----+
|..          o+=o+=o|
|..          + +oBo +|
|..          o + = +=o|
|..          . + o  oBo|
|..          .o S o . +.|
|..          .ooEo  o  +|
|..          ....+ o  |
|..          .. o .    |
+---[SHA256]-----+
jsm@JSM-AMD:~/eks$ ls
eks-ssh  eks-ssh.pub
```

Iniciar Cluster

Para la creación del cluster eks usamos este comando

```
eksctl create cluster \
--name eks-tp-m \
--region us-east-1 \
--node-type t3.small \
--nodes 3 \
--with-oidc \
--ssh-access \
--ssh-public-key $HOME/eks/eks-ssh.pub \
--managed \
--full-ecr-access \
--zones us-east-1a,us-east-1b,us-east-1c
```

Si todo esta correcto deberíamos tener lo siguiente a la espera que se termine de ejecutar la creación.

```
2025-03-04 21:35:47 [i] eksctl version 0.205.0
2025-03-04 21:35:47 [i] using region us-east-1
2025-03-04 21:35:47 [i] subnets for us-east-1a - public:192.168.0.0/19 private:192.168.96.0/19
2025-03-04 21:35:47 [i] subnets for us-east-1b - public:192.168.32.0/19 private:192.168.128.0/19
2025-03-04 21:35:47 [i] subnets for us-east-1c - public:192.168.64.0/19 private:192.168.160.0/19
2025-03-04 21:35:47 [i] nodegroup "ng-310da273" will use "" [AmazonLinux2/1.30]
2025-03-04 21:35:47 [i] using SSH public key "/home/jsm/eks/eks-ssh.pub" as "eksctl-eks-tp-m-nodegroup-ng-310da273-d6:c0:11:73:96:ec:d2:78:9c:fc:7b:70:1a:02:98:6e"
2025-03-04 21:35:49 [i] using Kubernetes version 1.30
2025-03-04 21:35:49 [i] creating EKS cluster "eks-tp-m" in "us-east-1" region with managed nodes
2025-03-04 21:35:49 [i] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2025-03-04 21:35:49 [i] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=eks-tp-m'
2025-03-04 21:35:49 [i] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "eks-tp-m" in "us-east-1"
2025-03-04 21:35:49 [i] Cloudwatch logging will not be enabled for cluster "eks-tp-m" in "us-east-1"
2025-03-04 21:35:49 [i] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-1 --cluster=eks-tp-m'
2025-03-04 21:35:49 [i] default addons metrics-server, vpc-cni, kube-proxy, coredns were not specified, will install them as EKS addons
2025-03-04 21:35:49 [i]
2 sequential tasks: { create cluster control plane "eks-tp-m",
  2 sequential sub-tasks: {
    5 sequential sub-tasks: {
      1 task: { create addons },
      wait for control plane to become ready,
      associate IAM OIDC provider,
      no tasks,
      update VPC CNI to use IRSA if required,
    },
    create managed nodegroup "ng-310da273",
  }
}
2025-03-04 21:35:49 [i] building cluster stack "eksctl-eks-tp-m-cluster"
2025-03-04 21:35:50 [i] deploying stack "eksctl-eks-tp-m-cluster"
2025-03-04 21:36:21 [i] waiting for CloudFormation stack "eksctl-eks-tp-m-cluster"
2025-03-04 21:36:54 [i] waiting for CloudFormation stack "eksctl-eks-tp-m-cluster"
2025-03-04 21:37:57 [i] waiting for CloudFormation stack "eksctl-eks-tp-m-cluster"
2025-03-04 21:39:01 [i] waiting for CloudFormation stack "eksctl-eks-tp-m-cluster"
```

Esto podría tardar entre 15m-40m

Una vez finalizado deberíamos ver lo siguiente

```
2025-03-04 21:50:27 [i] waiting for at least 3 node(s) to become ready in "ng-310da273"
2025-03-04 21:50:27 [i] nodegroup "ng-310da273" has 3 node(s)
2025-03-04 21:50:27 [i] node "ip-192-168-1-37.ec2.internal" is ready
2025-03-04 21:50:27 [i] node "ip-192-168-62-116.ec2.internal" is ready
2025-03-04 21:50:27 [i] node "ip-192-168-64-242.ec2.internal" is ready
2025-03-04 21:50:27 [i] created 1 managed nodegroup(s) in cluster "eks-tp-m"
2025-03-04 21:50:29 [i] kubectl command should work with "/home/jsm/.kube/config", try 'kubectl get nodes'
2025-03-04 21:50:29 [i] EKS cluster "eks-tp-m" in "us-east-1" region is ready
jsm@JSM-AMD:~$ kubectl get nodes
NAME                                STATUS    ROLES    AGE     VERSION
ip-192-168-1-37.ec2.internal        Ready    <none>    4m15s   v1.30.9-eks-5d632ec
ip-192-168-62-116.ec2.internal      Ready    <none>    4m19s   v1.30.9-eks-5d632ec
ip-192-168-64-242.ec2.internal      Ready    <none>    4m8s    v1.30.9-eks-5d632ec
```

Con `kubectl get nodes` podemos ver los nodos disponibles

Levantar Servicios

Ahora vamos a levantar varios servicios

Nginx

Para levantar nginx creamos un archivo `nginx.yaml` con el siguiente contenido

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
labels:
  app: nginx
```



```
spec:
  externalTrafficPolicy: Local
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

Ahora levantamos con

```
kubectl apply -f nginx.yaml
```

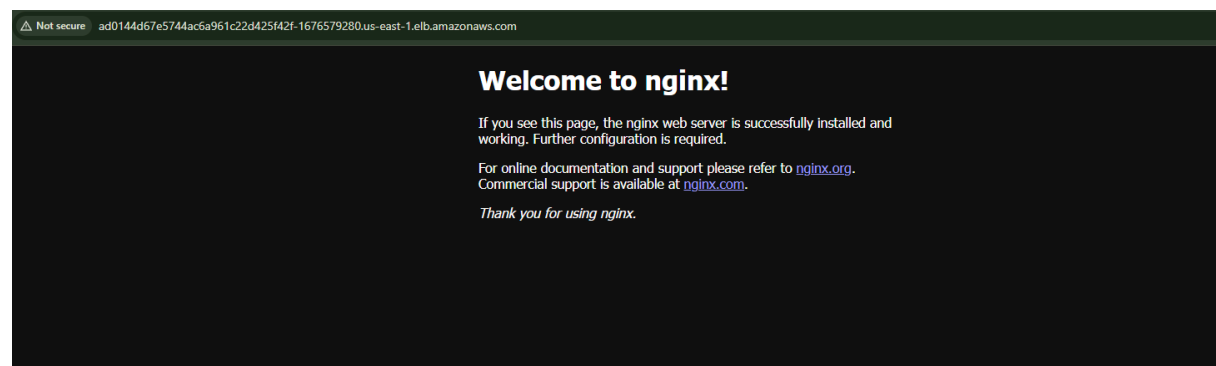
Una vez levantado podemos acceder desde el dns de elb. Para traer los dns de los elb podemos ejecutar el siguiente comando

```
kubectl get svc -n default
```

Vamos a tener dos entranos al EXTERNAL-IP de nginx

```
jsm@JSM-AMD:~$ kubectl get svc -n default
NAME            TYPE           CLUSTER-IP   EXTERNAL-IP                                PORT(S)          AGE
kubernetes      ClusterIP      10.100.0.1    <none>                                     443/TCP          54m
nginx            LoadBalancer  10.100.83.51  a394cbda0c2654bac91ec5079f5b8fb5-1555875173.us-east-1.elb.amazonaws.com  80:30392/TCP     2m12s
```

accedemos y listo ya tenemos nginx



Prometheus

Para levantar Prometheus primero vamos a configurar el storage EBS

Primero instalamos el driver con el siguiente comando

```
kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/overlays/stable/?ref=release-1.30"
```

Ahora verificamos con el siguiente comando

```
kubectl get pods -n kube-system -l app.kubernetes.io/name=aws-ebs-csi-driver
```

Deberíamos tener algo así

```
jsm@JSM-AMD:~$ kubectl get pods -n kube-system -l app.kubernetes.io/name=aws-ebs-csi-driver
```

NAME	READY	STATUS	RESTARTS	AGE
ebs-csi-controller-5ff7cc7d88-5s9zt	6/6	Running	0	5m31s
ebs-csi-controller-5ff7cc7d88-mqlfx	6/6	Running	0	5m31s
ebs-csi-node-545r9	3/3	Running	0	5m20s
ebs-csi-node-b2ksb	3/3	Running	0	5m24s
ebs-csi-node-c8g7v	3/3	Running	0	5m31s

Una vez que tenemos los driver ahora configuramos los permisos con el siguiente comando

```
eksctl create iamserviceaccount \
--name ebs-csi-controller-sa \
--namespace kube-system \
--cluster eks-tp-m \
--attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
--approve \
--role-only \
--role-name AmazonEKS_EBS_CSI_DriverRole
```

Y aplicamos actualizamos el add-on que administra el driver del EBS y lo asociamos al rol creado con el siguiente comando

```
eksctl create addon \
--name aws-ebs-csi-driver \
--cluster eks-tp-m \
```

```
--service-account-role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output text):role/AmazonEKS_EBS_CSI_DriverRole --force
```

Ahora que tenemos el EBS configurado vamos a levantar el Prometheus primero creamos el namespace con el siguiente comando

```
kubectl create namespace prometheus
```

Agregamos los repositorios con el siguiente comando

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

Ahora implementamos Prometheus con el siguiente comando

```
helm upgrade -i prometheus prometheus-community/prometheus \
--namespace prometheus \
--set alertmanager.persistence.storageClass="gp2" \
--set server.persistentVolume.storageClass="gp2"
```

Ahora verificamos que todos los pods este **Running** con el siguiente comando

```
kubectl get pods -n prometheus
```

Deberíamos tener lo siguiente

```
j3m@JSM-AMD:~$ kubectl get pods -n prometheus
NAME                                READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-0          1/1     Running   0           19m
prometheus-kube-state-metrics-5c7f9cf685-tn444  1/1     Running   0           19m
prometheus-prometheus-node-exporter-7pzl2      1/1     Running   0           19m
prometheus-prometheus-node-exporter-96dms      1/1     Running   0           19m
prometheus-prometheus-node-exporter-mrgrk      1/1     Running   0           19m
prometheus-prometheus-pushgateway-79964b5788-r4x5v  1/1     Running   0           19m
prometheus-server-68d87cf7c5-rvmlq            2/2     Running   0           19m
```

Ahora para acceder usamos **kubectl** para el enrutamiento del puerto con el siguiente comando

```
kubectl --namespace=prometheus port-forward deploy/prometheus-serve
```

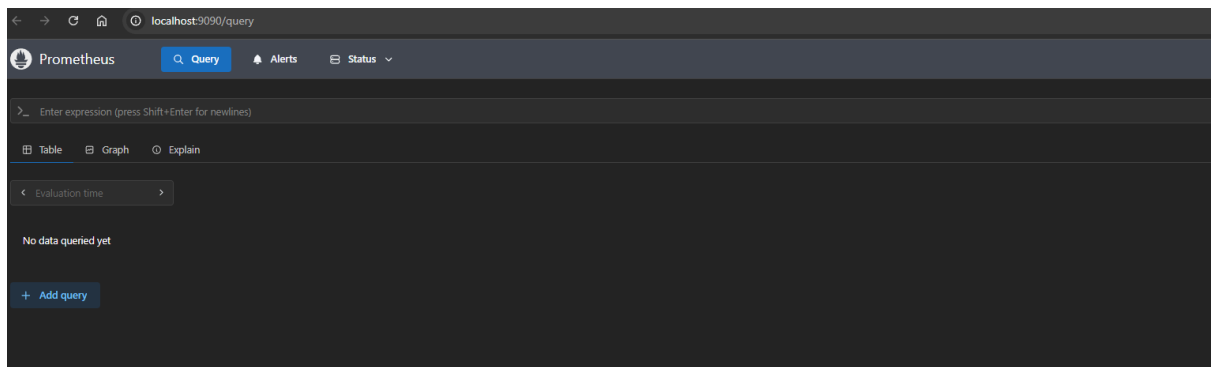
```
r 9090
```

deberíamos ver esto

```
jsm@JSM-AMD:~$ kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
Forwarding from 127.0.0.1:9090 -> 9090
```

Ahora accedermos desde <http://localhost:9090>

Listo tenemos Prometheus



Grafana

Primero vamos a crear un namespace para grafana con el comando:

```
kubectl create namespace grafana
```

Ahora dentro de `$HOME/eks` creamos una carpeta para las configs de grafana

```
mkdir ${HOME}/eks/grafana
```

Ahora dentro de `$HOME/eks/grafana` creamos el archivo `grafana.yaml` con los siguientes datos

```
datasources:
datasources.yaml:
  apiVersion: 1
  datasources:
  - name: Prometheus
    type: prometheus
```

```
url: http://prometheus-server.prometheus.svc.cluster.local
access: proxy
isDefault: true
```

Ahora agregamos el repo de grafana

```
helm repo add grafana https://grafana.github.io/helm-charts
```

Ahora desplegamos `grafana` con helm con el siguiente comando

```
helm install grafana grafana/grafana \
--namespace grafana \
--set persistence.storageClassName="gp2" \
--set persistence.enabled=true \
--set adminPassword='PASSWD' \
--values ${HOME}/eks/grafana/grafana.yaml \
--set service.type=LoadBalancer
```

Deberíamos ver lo siguiente

```
jasm@JSM-AMD:~$ helm install grafana grafana/grafana --namespace grafana --set persistence.storageClassName="gp2" --set persistence.enabled=true --set adminPassword='PASSWD' --values ${HOME}/eks/grafana/grafana.yaml --set service.type=LoadBalancer
NAME: grafana
LAST DEPLOYED: Wed Mar  5 00:19:58 2025
NAMESPACE: grafana
STATUS: deployed
REVISION: 1
NOTES:
1. Get your 'admin' user password by running:

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

2. The Grafana server can be accessed via port 80 on the following DNS name from within your cluster:

  grafana.grafana.svc.cluster.local

  Get the Grafana URL to visit by running these commands in the same shell:
  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
  You can watch the status of by running 'kubectl get svc --namespace grafana -w grafana'
  export SERVICE_IP=$(kubectl get svc --namespace grafana grafana -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
  http://$SERVICE_IP:80

3. Login with the password from step 1 and the username: admin
```

ahora con el siguiente comando verificamos que este correctamente levantado

```
kubectl get pods -n grafana
```

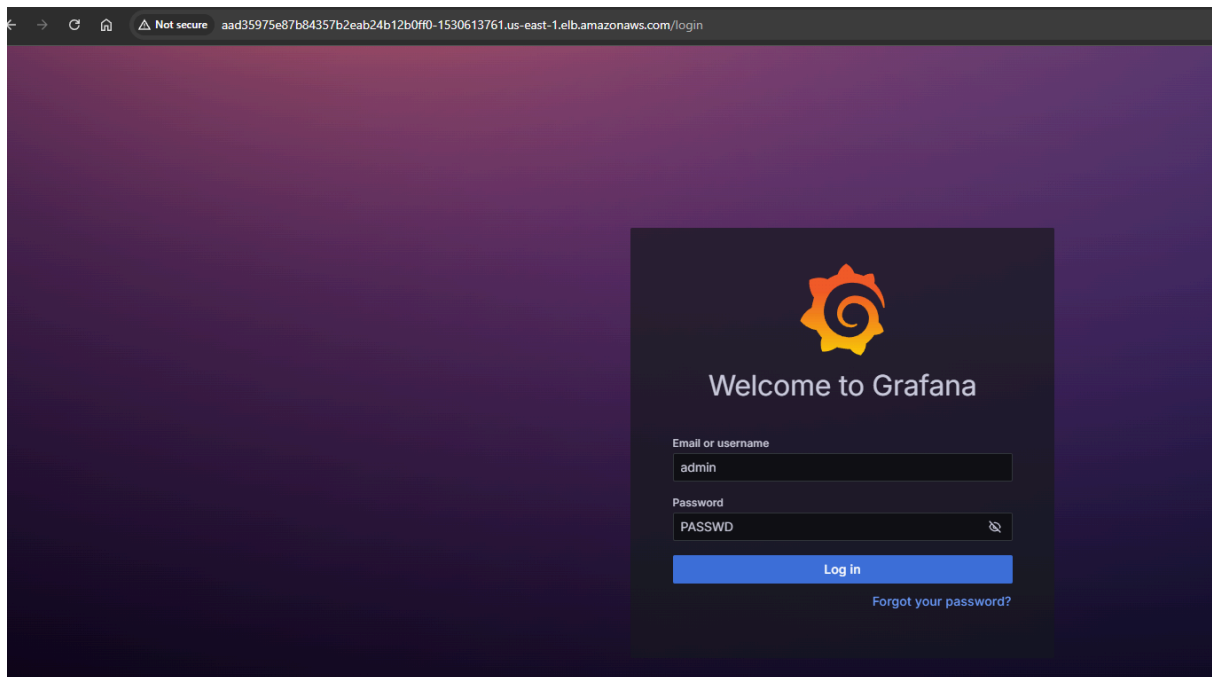
```
jasm@JSM-AMD:~$ kubectl get pods -n grafana
NAME                                READY   STATUS    RESTARTS   AGE
grafana-fdc6dcf8f-9c5c8            1/1     Running   0           2m1s
```

Buscamos el DNS del ELB para acceder con el siguiente comando

```
kubectl get svc -n grafana
```

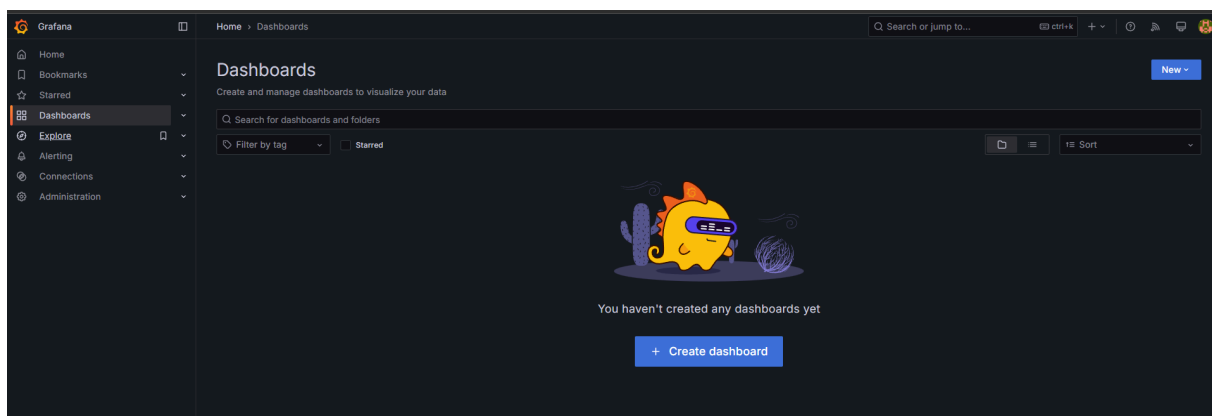
```
jasm@JSM-AMD:~$ kubectl get svc -n grafana
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
grafana   LoadBalancer  10.100.139.151  aad35975e87b84357b2eab24b12b0ff0-1530613761.us-east-1.elb.amazonaws.com  80:30377/TCP  6m39s
```

Ahora accedemos a grafana

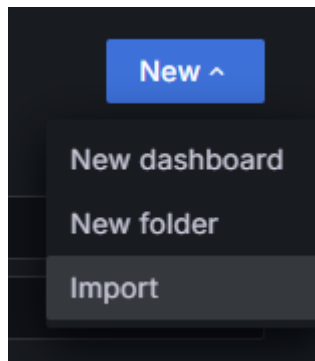


usamos PASSWD como password declarado

Ahora vamos a **dashboard**




Ahora desde **New** importamos el dashboard



Usamos el ID 3119

Import dashboard

Import dashboard from file or Grafana.com


Upload dashboard JSON file
Drag and drop here or click to browse
Accepted file types: .json, .txt

Find and import dashboards for common applications at grafana.com/dashboards

Load

Import via dashboard JSON model

```
{
  "title": "Example - Repeating Dictionary variables",
  "uid": "_0HnEoN4z",
  "panels": [...]
  ...
}
```

LoadCancel

Hacemos click en Load

Ahora en la ultima opción ponemos Prometheus y hacemos click en [Import](#)

Import dashboard

Import dashboard from file or Grafana.com

Importing dashboard from Grafana.com

Published by	Jjo Org
Updated on	2017-09-08 12:22:08

Options

Name


Folder

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.

[Change uid](#)

prometheus

 Prometheus

default Prometheus

[Open advanced data source picker →](#)

Vamos a ver el Dashboard

