Proyecto Devops Integrador

Grupo N°2

Integrantes:

Sepulveda Javier

Garbero Diego

Mastramico Gerardo

Gazzanego Walter

Muller Frank

Los siguientes ejercicios se basaron en las pautas mencionadas en el PDF

[Nuevo Proyecto Integrador DevOps.pdf](https://github.com/Diegocel/mundose/blob/a518a1b5ce4e30119e69d6911a4988585759b994/Nuevo%20Proyecto%20Integrador%20DevOps.pdf)

Contenido

[Máquina EC2 - Bastion 3](#_Toc128158195)

[Creación de la instancia 3](#_Toc128158196)

[Conexión a la instancia EC2 4](#_Toc128158197)

[Configuración de la instancia y cliente aws 4](#_Toc128158198)

[Cluster EKS 6](#_Toc128158199)

[Creación del cluster 6](#_Toc128158200)

[Crear cluster con eksctl 6](#_Toc128158201)

[Creación del cluster con Terraform 7](#_Toc128158202)

[MapUsers 8](#_Toc128158203)

[Azure Devops 9](#_Toc128158204)

[Service connections 9](#_Toc128158205)

[Azure Pipeline 11](#_Toc128158206)

[Monitoreo 13](#_Toc128158207)

[Instalación 13](#_Toc128158208)

[Creación del Cluster de ElasticSearch 15](#_Toc128158209)

[Configurar Acceso ElasticSearch 17](#_Toc128158210)

[Despliegue Fluent Bit 17](#_Toc128158211)

[Configuración de Kibana 18](#_Toc128158212)

# Máquina EC2 - Bastion

## Creación de la instancia

Se creo una instancia EC2 para utilizarla como Bastion con las siguientes características.

Se le configuro en user data, un script para la configuración de las herramientas necesarias para los pasos posteriores (aws cli, eksctl, Helm, terraform, etc)

**Name**: Jenkins

**Region**: us-east-2  
**S.O.** : Ubuntu Server 20.04  
**Family** (**Tipo**): t2.small

**Storage**: 8GB General Purpose SSD (gp2)

**Network** **Security** **Group**: TCP\_8080 y TCP\_22 abiertos

**User** **data**: Contenido del archivo ec2\_user\_data.sh

**Key** **Pair**: Se genero una keypair con formato pem llamada Jenkins.pem

[Interfaz de usuario gráfica, Texto, Aplicación, Teams

Descripción generada automáticamente](https://github.com/Diegocel/mundose/blob/612b0ba1c6f4a9609233b16d56072d8185f47780/Capturas/Generacion%20EC2.png)  
*(Selecciones realizadas al crear la instancia EC2 –* [*LINK PARA VER COMPLETO*](https://github.com/Diegocel/mundose/blob/612b0ba1c6f4a9609233b16d56072d8185f47780/Capturas/Generacion%20EC2.png)*)*

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

*(confirmación de creación)*

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente con confianza media

*(instancia en ejecución)*

## Conexión a la instancia EC2

A dicha instancia nos conectamos mediante ssh usando la keypair generada anteriormente.

Texto

Descripción generada automáticamente

## Configuración de la instancia y cliente aws

Mediante IAM, generamos un rol con los permisos de [AdministratorAccess](https://us-east-1.console.aws.amazon.com/iam/home#/policies/arn:aws:iam::aws:policy/AdministratorAccess) y se lo asigno a la EC2 creada.

**Trusted entity type**: AWS Service

**Use case**: EC2

**Permissions policies**: [AdministratorAccess](https://us-east-1.console.aws.amazon.com/iam/home#/policies/arn:aws:iam::aws:policy/AdministratorAccess)

**RoleName**: ec2-admin-role

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Falta foto de asignación del ROLE

Conectado a la instancia, configuramos el ***cliente de AWS*** mediante **aws configure** y pudiendo comprobar la conexión mediante el comando **aws ec2 describe-instances**

Texto

Descripción generada automáticamente

Falta foto de **aws ec2 describe-instances**

# Cluster EKS

## Creación del cluster

Para la creación del cluster de EKS, se realizó de las dos maneras (EKS Cli y Terraform) para practicarlas.

### Crear cluster con eksctl

Este paso solo se realizo a modo practica verificar el comando. Se utilizo el comando ***eksctl create cluster*** , confirmando el funcionamiento se procedió a su eliminación con ***eksctl delete cluster***.

*eksctl create cluster --name eks-mundos-e --region us-east-2 --node-type t2.small --with-oidc --ssh-access --ssh-public-key jenkins --managed --full-ecr-access --zones us-east-2a,us-east-2b,us-east-2c*

Texto

Descripción generada automáticamente

*(Ejecución del comando eksctl create cluster)*

Verificamos en el portal del CloudFormation el estado

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Y procedimos a la eliminación del mismo

### Creación del cluster con Terraform

Se utilizaron los siguientes archivos de terraform para crear un cluster con las estas características principales

[Archivos de Terraform](https://github.com/Diegocel/mundose/tree/612b0ba1c6f4a9609233b16d56072d8185f47780/terraform)

**cluster\_name** = "mundose-eks"

**cluster\_version** = "1.24"

**instance\_types** = "t2.small"

**ami\_type** = "AL2\_x86\_64"

**Cant. Nodes** = 2

Lo pasos que realizamos fueron:

*# Inicializamos el directorio de trabajo e instalamos complementos para los proveedores requeridos*

**terraform init**

Texto

Descripción generada automáticamente

#Verificamos el plan de ejecución

**terraform plan**

Texto

Descripción generada automáticamente

#Aplicamos los manifiestos y verificamos que se hayan creados los recursos solicitados

**terraform apply**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## MapUsers

Para autorizar nuestro usuario, tuvimos que obtener el ARN del mismo, mediante la consola IAM, y agregarlo al EKS mediante la edición del configmap/aws-auth:

kubectl edit -n kube-system configmap/aws-auth

Texto

Descripción generada automáticamente

(Archivo configmap/aws-auth)

# Azure Devops

En Azure Devops, se generó un proyecto público, llamado mundose.

[Proyecto Mundose](https://dev.azure.com/bitcomar/mundose)

Interfaz de usuario gráfica, Aplicación

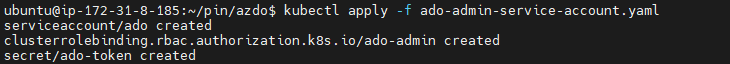
Descripción generada automáticamente

En dicho proyecto se creó un [repositorio](https://dev.azure.com/bitcomar/_git/mundose), donde se subió el archivo [nginx-deployment.yaml](https://dev.azure.com/bitcomar/_git/mundose?path=/nginx-deployment.yaml) , el cual contine el manifiesto para la creación de un deployment y un Service type loadbalancer para un Nginx de prueba

### Service connections

Para generar la conexión entre Azure Devops y nuestro EKS, primero tuvimos que generar un ServiceAccount en el EKS para utilizarlo con Azure Devops. Para dicha creación se aplicó el manifiesto [ado-admin-service-account.yaml](https://github.com/Diegocel/mundose/blob/c908c51ffa72718dea1f65885ffb7ea370a8ca12/ado-admin-service-account.yaml) , donde se declara un tipo ServiceAccount, ClusterRoleBinding vinculado al rol cluster-admin y un Secret.

*Kubectl apply -f ado-admin-service-account.yaml*



Luego se obtuvo el secret generado con el siguiente comando

*kubectl get secret ado-token -n kube-system -o json*

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Adicionalmente se obtuvo la API URL de nuestro EKS

*kubectl cluster-info | grep -E 'Kubernetes master|Kubernetes control plane' | awk '/http/ {print $NF}'*



Con dichos datos se procedió a la creación de la Service connections con EKS en Azure Devops.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

### Azure Pipeline

Se genero un [pipeline](https://dev.azure.com/bitcomar/_git/mundose?path=/azure-pipelines.yml), indicando que se utiliza el repositorio de Azure, configurando como trigger el branch main y la utilización de un pool con agentes propios.

Posee una sola tarea ([KubernetesManifest](https://learn.microsoft.com/en-us/azure/devops/pipelines/tasks/reference/kubernetes-manifest-v0?view=azure-pipelines)@0) la cual aplica el yaml del nginx que subimos anteriormente en el EKS.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Al guardar los cambios, dispara la ejecución del mismo, debido al trigger configurado.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para ingresar el nginx creado, se extrajo del log de la ejecución, la URL del hostname.

Captura de pantalla de computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Monitoreo

Para realizar el monitoreo, utilizamos el Stack de ELK.

## Instalación

Para la instalación seguimos los siguientes pasos:

* *Definición de las siguientes* ***variables***

*export AWS\_REGION='us-east-2'*

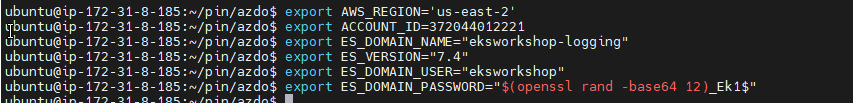
*export ACCOUNT\_ID=372044012221*

*export ES\_DOMAIN\_NAME="eksworkshop-logging"*

*export ES\_VERSION="7.4"*

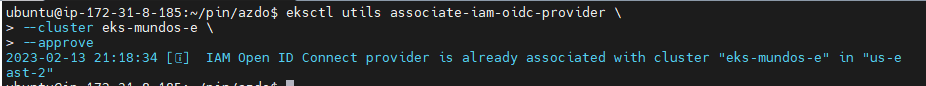
*export ES\_DOMAIN\_USER="eksworkshop"*

*export ES\_DOMAIN\_PASSWORD="$(openssl rand -base64 12)\_Ek1$"*



* Configurar **OpenID Connect**

*eksctl utils associate-iam-oidc-provider --cluster eks-mundos-e --approve*



* *Crear* ***IAM policy*** *con AWS CLI*

Generamos el JSON para la IAM policy, utilizando el comando CAT para que realice la creación del archivo ya con las variables sustituidas.

*cat «EoF> ~/environment/logging/fluent—bit—policy.json*

*{*

*"Version": "2012-10-17",*

*"Statement":[*

*{*

*"Action":[*

*"es:ESHttp\*"*

*],*

*"Resource":*

*"arn:aws:es:${AWS\_REGION}:${ACCOUNT\_ID>:domain/${ES\_DOHAIN\_NAME>",*

*"Effect": "Allow"*

*}*

*]*

*}*

*EoF*

FALTA FOTO DEL COMANDO

Archivo generado [fluent—bit—policy.json](https://github.com/Diegocel/mundose/blob/d7597c609032c0ad6e6043f6184fe37c1160214b/fluent-bit-policy.json)

Texto

Descripción generada automáticamente

* Creamos la IAM policy aplicando el JSON

*aws iam create-policy --policy-name fluent-bit-policy --policy-document file://~/environment/logging/fluent-bit-policy.json*

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* Creación del namespace

Se creo un namespace con el nombre logging

*kubectl create namespace logging*



* Creación de la cuenta de servicio

Creamos una cuenta de servicio en el namespace anterior, y le adjuntamos la policy creada.

*eksctl create iamserviceaccount --name fluent-bit --namespace logging --cluster eks-mundos-e --attach-policy-arn "arn:aws:iam::${ACCOUNT\_ID}:policy/fluent-bit-policy" --approve --override-existing-serviceaccounts*

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Verificamos la creación

*kubectl get serviceaccount —n Logging*

Imagen que contiene Texto

Descripción generada automáticamente

*kubectl -n logging describe sa fluent-bit*

Falta foto del comando

## Creación del Cluster de ElasticSearch

Nos descargamos el siguiente [template](https://www.eksworkshop.com/intermediate/230_logging/deploy.files/es_domain.json) y remplazamos las variables declaradas anteriormente utilizando el siguiente comando.

*curl -sS https://archive.eksworkshop.com/intermediate/230\_logging/deploy.files/es\_domain.json | envsubst > ~/environment/logging/es\_domain.json*

[*es\_domain.json*](https://github.com/Diegocel/mundose/blob/aea345832fee41f6c8a4ab8ec3ddcdc78e8ca904/es_domain.json)



*Tuvimos que corregirle el InstanceType ya que el que figuraba en el JSON original, no existe más.*

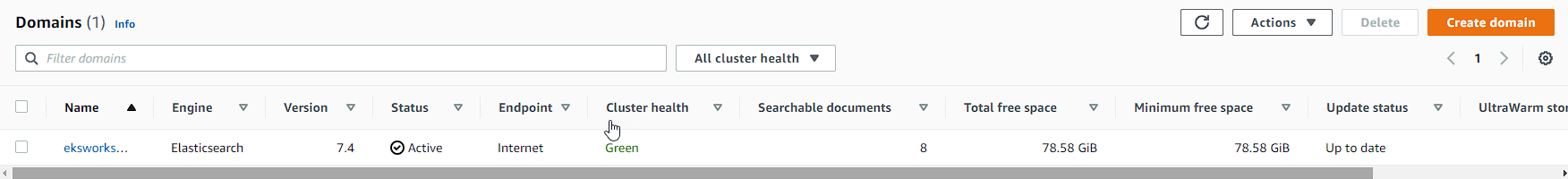
Ejecutamos la creación del Cluster de Elasticsearch indicando en el argumento que utilice el json creado anteriormente.

*aws es create-elasticsearch-domain --cli-input-json file://~/environment/logging/es\_domain.json*

Texto

Descripción generada automáticamente

Confirmamos la creación desde la consola de AWS



Y opcionalmente desde el cli

Texto

Descripción generada automáticamente

## Configurar Acceso ElasticSearch

Configuramos el acceso a ElasticSearch ejecutando el siguiente SH, el cual actualiza los permisos agregando el ARN del Service account

configure-elastic-access.sh

Texto

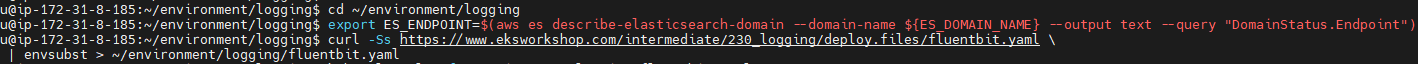
Descripción generada automáticamente

## Despliegue Fluent Bit

Generamos el manifiesto para el deploy de Fluent Bit, con el siguiente SH, el cual declara una variable con el endpoint del Elasticsearch, descarga un yaml con el manifiesto y remplaza las variables. Generando un nuevo archivo fluentbit.yaml

[fluentbit-deployment-generator.sh](https://github.com/Diegocel/mundose/blob/83b22980f1e31b1f2a0085c0df8e145631fcaebc/Scripts/fluentbit-deployment-generator.sh)

[fluentbit.yaml](https://github.com/Diegocel/mundose/blob/83b22980f1e31b1f2a0085c0df8e145631fcaebc/fluentbit.yaml)



Para ejecutar el deploy, aplicamos el manifiesto creado anteriormente

*kubectl apply -f ~/environment/logging/fluentbit.yaml*

Texto

Descripción generada automáticamente

Verificamos la creación de los pods

*kubectl --namespace=logging get pods*

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## Configuración de Kibana

Para ingresar, debemos obtener las credenciales y URL de acceso a Kibana, mediante los comandos

echo "Kibana URL: https://${ES\_ENDPOINT}/\_plugin/kibana/

Kibana user: ${ES\_DOMAIN\_USER}

Kibana password: ${ES\_DOMAIN\_PASSWORD}"

Falta Foto comandos

Falta Foto Login

///////////////////////HASTA LLEGUE ACOMODANDO////////////////////////////////////////////

//

Create index pattern

Generamos un index, con el siguiente patrón \*fluent-bit\* y verificamos que nos muestre los datos

Foto index

Foto Visor de datos

Desplegar Prometheus

Generemos un namespace llamado prometheus

kubectl create namespace Prometheus

Foto comando

Utilizamos helm para instalar Prometheus, en el namespame creado y definiendo el tipo de storage en gp2

helm install prometheus prometheus-community/prometheus \

--namespace prometheus \

--set alertmanager.persistentVolume.storageClass="gp2" \

--set server.persistentVolume.storageClass="gp2"

Foto comando

Generamos un port-forward para exponer prometheus en la instancia de EC2 en el puerto 8080

kubectl port-forward -n prometheus deploy/prometheus-server 8080:9090 --address 0.0.0.0

Foto comando

La dirección la obtenemos, desde las propiedades de la instancia de EC2 que aloja el deploy

Foto consola de EC2 Falta

Comprobamos el funcionamiento

Foto prometeus y de targets

Desplegar Grafana

Para el deploy de Grafana, seguimos los siguientes pasos:

generamos un YAML ( ~/environment/grafana/grafana.yaml) donde declaramos el datasource

Foto proceso

LINK ARCHIVO grafana.yaml

generamos un namespace llamado grafana

kubectl create namespace grafana

Foto proceso

Mediante HELM instalamos Grafana, seleccionando el tipo de almacenamiento en gp2, establecimos la clave del administrado, aplicamos la configuración de grafana.yaml y el tipo de servicio como loa balancer

helm install grafana grafana/grafana --namespace grafana --set persistence.storageClassName="gp2" --set persistence.enabled=true --set adminPassword='EKS!sAWSome' --values ${HOME}/environment/grafana/grafana.yaml --set service.type=LoadBalancer

Foto proceso

Verificamos que los recursos se hayan iniciada correctamente

kubectl get all -n grafana

Foto proceso

Obtener acceso a Grafana

Para ingresar, debemos obtener la dirección asignada.

export ELB=$(kubectl get svc -n grafana grafana -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')

echo "http://$ELB"

Foto proceso

Y decodificar la contraseña asignada

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

Foto proceso

Foto portal Grafana

Configurar Grafana

Importamos Cluster Monitoring Dashboard (Codigo 3119)

Foto proceso

Foto dashboard OK

Importamos Pods Monitoring Dashboard (Codigo 6417)

Foto proceso

Foto dashboard OK

Cleanup de recursos

Se borraron todos los recursos creado en AWS

Borrar FluentBit y Elastic

cd ~/environment/

kubectl delete -f ~/environment/logging/fluentbit.yaml

aws es delete-elasticsearch-domain --domain-name ${ES\_DOMAIN\_NAME}

eksctl delete iamserviceaccount --name fluent-bit --namespace logging --cluster eksworkshop-eksctl --wait

aws iam delete-policy --policy-arn "arn:aws:iam::${ACCOUNT\_ID}:policy/fluent-bit-policy"

kubectl delete namespace logging

Borrar Prometheus y Grafana

helm uninstall prometheus --namespace prometheus

kubectl delete ns prometheus

helm uninstall grafana --namespace grafana

kubectl delete ns grafana

rm -rf ${HOME}/environment/grafana

Borrar Cluster EKS

Terraform destroy