Despliegue de EKS usando CrossPlane + ArgoCD

Roberto Rodríguez Márquez

Índice

Índice **1**
1. Objetivos que se quieren conseguir **2**
1.2. ¿Por qué Crossplane? **2**
2. Fundamentos teóricos y conceptos **5**
2.1 Kubernetes 5
2.1.1 Plano de control 5
2.2 GitOps 5
2.3 ArgoCD 6
2.4 Infraestructura como código 7
2.5 Crossplane 7
2.5.1 Proveedor 8
3. Escenario que se ha realizado **9**
3.1 Instalaciones 9
3.1.1 Clúster local de k8s 9
3.1.2 ArgoCD 9
3.1.3 Crossplane 10
3.2 Configuraciones 10
3.2.1 AWS 10
3.2.1 Proveedores de Crossplane 10
3.2.1.1 AWS 10
3.2.1.2 kubernetes 11
3.2.2 Recursos gestionados 12
3.2.3 Ngrok 12
3.2.4 Webhook 13
3.2.4 Preparación previa a la demo 13
4. Demostraciones **15**
4.1 Modificación del número de nodos 15
4.2 Despliegue en el clúster 16
4.2.1 Configuración del proveedor de kubernetes 16
4.2.2 Script 17
4.2.3 Aplicación de ArgoCD 17
5. Dificultades que se han encontrado **19**
5.1 Despliegue sobre AWS 19
5.2 Despliegue sobre Kubernetes 19
6. Conclusión **20**
7. Bibliografía **21**

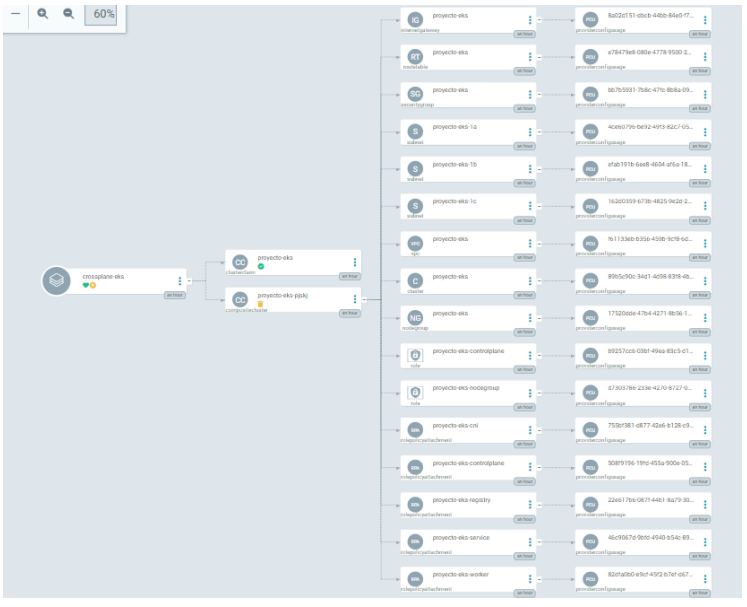
1. Objetivos que se quieren conseguir

Los objetivos que se quieren conseguir son los siguientes:

- Lograr una mayor **comprensión** de lo que es **Crossplane** y sus posibles aplicaciones.
- **Automatización**: La definición de la infraestructura como código y la automatización del proceso de implementación y gestión de la infraestructura permite a los equipos de desarrollo y operaciones centrarse en tareas más importantes y reducir los errores humanos.
- **Escalabilidad**: Otra ventaja de Kubernetes es su capacidad de escalar automáticamente, tanto hacia arriba como hacia abajo, en función de las necesidades del sistema. Al utilizar **Crossplane** para gestionar la infraestructura de AWS, se pueden aprovechar las capacidades de escalabilidad de AWS y de Kubernetes para garantizar que el sistema siempre tenga los recursos necesarios para funcionar correctamente.
- **Portabilidad**: Al utilizar Crossplane, se puede asegurar que la infraestructura es portátil y se puede mover fácilmente a otras nubes o proveedores de infraestructura en el futuro. Esto permite a las organizaciones evitar el bloqueo de proveedores y aprovechar las ventajas de diferentes proveedores de nube según sea necesario.
- **Costes**: La utilización de una infraestructura basada en la nube puede ser costosa, por lo que es importante asegurarse de que los recursos se utilizan de manera eficiente y se minimizan los costos innecesarios. Al utilizar Crossplane para gestionar la infraestructura de AWS, se puede optimizar el uso de los recursos y evitar pagar por recursos no utilizados.

1.2. ¿Por qué Crossplane?

La **principal ventaja** de Crossplane es que, aún existiendo frameworks que permitan desplegar la misma infraestructura, Crossplane traslada esas interacciones a **kubernetes**, utilizando sus comandos propios y sus métodos. Esto quiere decir que, por ejemplo, habiendo desplegado un clúster en AWS, se crean automáticamente los siguientes objetos de kubernetes enlazados a la api de AWS:



Además, utilizando kubectl podemos obtener aún más información acerca de estos objetos, dentro del propio clúster local:

```
Deletion Policy:  Delete
For Provider:
  Ami Type:       AL2_x86_64
  Capacity Type:  ON_DEMAND
  Cluster Name:   proyecto-eks
  Cluster Name Ref:
    Name:  proyecto-eks
  Cluster Name Selector:
    Match Controller Ref:  true
  Disk Size:               20
  Instance Types:
    t2.micro
  Node Role:  arn:aws:iam::691741190841:role/proyecto-eks-nodegroup
  Node Role Ref:
    Name:  proyecto-eks-nodegroup
  Node Role Selector:
    Match Controller Ref:  true
    Match Labels:
      Role:         proyecto-eks-nodegroup
  Region:           us-east-1
  Release Version:  1.26.4-20230513
  Scaling Config:
    Desired Size:  4
    Max Size:      10
    Min Size:      4
  Subnet Refs:
    Name:  proyecto-eks-1a
    Name:  proyecto-eks-1b
    Name:  proyecto-eks-1c
  Subnet Selector:
    Match Labels:
      Access:  public
```

En la imagen se puede obtener toda la información referente a el objeto indicado, en este caso un nodegroup, y podemos ver, por ejemplo, el rol de nodo que tiene asignado, la región en la que se encuentra, las subredes disponibles…

Aparte de eso, las ventajas que tiene frente el principal competidor, terraform, son:

- **Ficheros más entendibles (YAML)**: Crossplane utiliza archivos YAML para definir la infraestructura, lo que los hace más fáciles de entender y leer para los desarrolladores que pueden estar más familiarizados con el formato YAML. Terraform, por otro lado, utiliza su propio lenguaje de configuración llamado HCL (HashiCorp Configuration Language), que puede tener una curva de aprendizaje para aquellos que no están familiarizados con él.
- **Permite desplegar indistintamente en los proveedores cloud**: Crossplane se enfoca en la gestión de la infraestructura multi-nube, lo que significa que permite a los usuarios definir y gestionar recursos en múltiples proveedores de nube utilizando la misma sintaxis de Kubernetes. Terraform también

soporta múltiples proveedores de nube, pero requiere la definición de cada recurso utilizando un proveedor de nube específico.

- **Metodología GitOps**: Crossplane está diseñado para trabajar en una metodología GitOps, lo que significa que todas las definiciones de recursos se almacenan en un repositorio Git y los cambios se aplican automáticamente al clúster de Kubernetes utilizando un proceso de integración y entrega continua (CI/CD). Terraform no está diseñado específicamente para trabajar con GitOps, aunque puede integrarse con sistemas de control de versiones como Git para almacenar y gestionar el código de infraestructura.

2. Fundamentos teóricos y conceptos

Se van a definir los siguientes fundamentos teóricos y conceptos relacionados con el proyecto y necesarios para su entendimiento, en orden de uso y de complejidad.

2.1 Kubernetes

Kubernetes o k8s para acortar, es una plataforma de sistema distribuido de código libre para la automatización del despliegue, ajuste de escala y manejo de aplicaciones. Una de las principales ventajas de Kubernetes es que ofrece una plataforma común para el desarrollo y la producción de aplicaciones. Esto significa que los equipos de desarrollo pueden crear aplicaciones en sus equipos y luego trasladarlas sin problemas a un entorno de producción utilizando las mismas herramientas y procesos. Kubernetes proporciona una capa de abstracción entre la infraestructura subyacente y las aplicaciones que se ejecutan en ella, lo que facilita la portabilidad de las aplicaciones entre diferentes plataformas y proveedores de nube. Además, Kubernetes ofrece características de autoreparación, lo que significa que las aplicaciones pueden recuperarse automáticamente de fallos en tiempo de ejecución, sin necesidad de intervención humana.

2.1.1 Plano de control

El plano de control de Kubernetes es el conjunto de componentes que se encargan de gestionar el estado del clúster y de coordinar todas las operaciones en el mismo. Estos componentes son responsables de tomar decisiones sobre la orquestación y el escalado de los contenedores y aplicaciones en el clúster, y de garantizar que el estado deseado del clúster se mantenga en todo momento.

2.2 GitOps

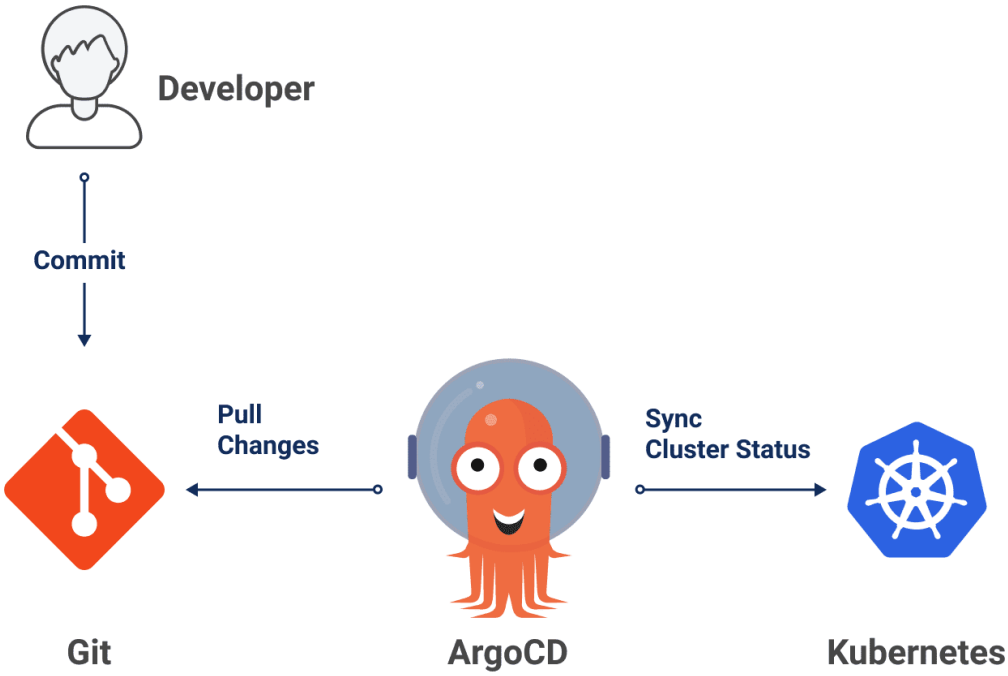
La metodología GitOps es un enfoque para la entrega continua de aplicaciones en la nube que utiliza Git como fuente de verdad para la configuración y la implementación de la infraestructura y las aplicaciones. En la metodología GitOps, todas las definiciones de la infraestructura y las aplicaciones se almacenan en un repositorio Git centralizado. Los cambios en el repositorio Git son automáticamente detectados por una herramienta de despliegue, que se encarga de implementar los cambios en la infraestructura y las aplicaciones.

La metodología GitOps se basa en los principios de la automatización, la colaboración y la transparencia. En primer lugar, la metodología GitOps

- **Automatización** de la gestión de la infraestructura y las aplicaciones, lo que permite la implementación continua de cambios en un entorno controlado y seguro.
- **Colaboración** entre los miembros del equipo, ya que todos los cambios se realizan en el repositorio Git centralizado, lo que permite a los miembros del equipo trabajar en conjunto de manera más eficiente.
- **Transparencia**, ya que todos los cambios y versiones se registran en el repositorio Git centralizado, lo que permite a los miembros del equipo revisar y rastrear el historial de cambios.

2.3 ArgoCD

Argo CD es una herramienta de entrega continua (Continuous Delivery) y de operaciones de infraestructura (Infrastructure Operations) que se ejecuta en Kubernetes. Permite la automatización y el control del proceso de implementación y despliegue de aplicaciones en un clúster de Kubernetes.



Una de las características más destacadas de Argo CD es su capacidad para automatizar la gestión de versiones y el despliegue de aplicaciones. Permite la definición de flujos de trabajo (workflows) para la implementación de cambios, lo que garantiza que los cambios se realicen de manera controlada y segura. Argo CD también incluye características de seguridad, como el control de acceso basado en roles y la autenticación de usuarios.

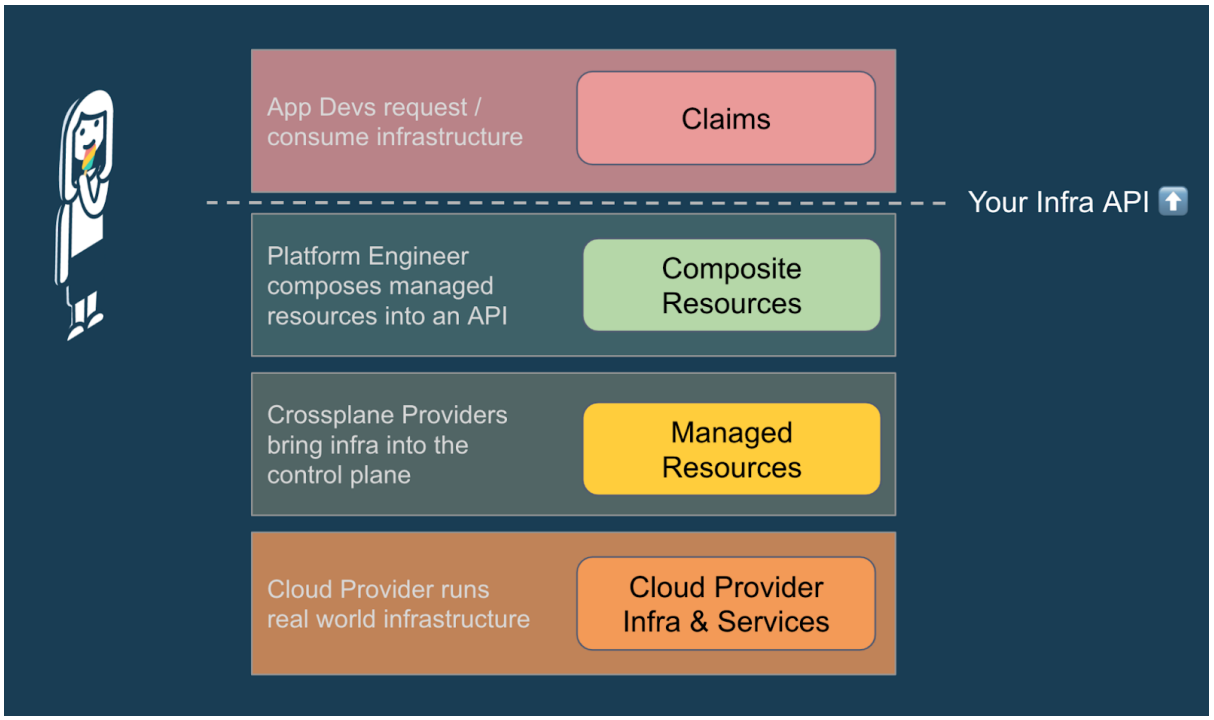
2.4 Infraestructura como código

Es una práctica que consiste en definir la infraestructura de una aplicación o sistema de forma programática utilizando un lenguaje de programación o una sintaxis específica. En lugar de configurar manualmente servidores, redes y otros recursos de infraestructura, los desarrolladores y los equipos de operaciones pueden definir la infraestructura como código y utilizar herramientas de automatización para gestionar y desplegar la infraestructura en un entorno reproducible y escalable.

2.5 Crossplane

Según la documentación oficial, crossplane es una extensión de kubernetes que transforma kubernetes en un **plano de control Universal**. Permite a los usuarios utilizar cualquier API como si fuera nativa de Kubernetes. Esto se logra mediante el uso de los recursos personalizados de Kubernetes (CRD), que permiten a los desarrolladores definir y extender los recursos de Kubernetes con sus propias definiciones. Puede crear y gestionar recursos que no son nativos de Kubernetes, como bases de datos, servicios de almacenamiento en la nube, cuentas de usuario y cualquier otra cosa que una API pueda ofrecer.

Además, Crossplane es compatible con **múltiples proveedores de nube**, lo que significa que los usuarios pueden utilizar cualquier servicio de cualquier proveedor de nube, incluidos Amazon Web Services, Google Cloud Platform y Microsoft Azure, entre otros. Esto permite a los usuarios crear **aplicaciones que utilizan servicios de varios proveedores de nube** sin tener que preocuparse por la complejidad de gestionar múltiples APIs de nube diferentes.



2.5.1 Proveedor

Un proveedor de Crossplane le permite aprovisionar una **infraestructura en un servicio externo**. se utiliza, entre otras cosas, para la autenticación, la realización de llamadas a API externas y utilizando APIs de kubernetes.

Para utilizar un proveedor además, es necesaria una configuración. Ésta se realiza por medio del uso de **providerConfig**. Normalmente se utilizan para autenticarse con la API con la que se está comunicando, En este proyecto, se utilizan tanto con las credenciales de **AWS** como con el **clúster de EKS**.

3. Escenario que se ha realizado

El escenario consiste en lo siguiente:

- Clúster local de kubernetes, en el que se han instalado las siguientes aplicaciones:
  - ArgoCD
  - Crossplane, con los proveedores de AWS y Kubernetes.
- Clúster de EKS creado y gestionado por Crossplane, en el que hay 3 nodos.

3.1 Instalaciones

Para la realización del proyecto, se necesita instalar los siguientes recursos:

3.1.1 Clúster local de k8s

Para desplegar un clúster local de k8s, se usará **kind**, ya que el provider de aws de crossplane consume muchos recursos, y kind es más liviano.. Para su instalación se seguirá la documentación oficial: Instalación kind. Para gestionar el clúster también es necesario **kubectl**, y la documentación oficial de instalación es la siguiente: instalación kubectl. Es muy importante partir de un **clúster nuevo** y sin configuraciones previas.

3.1.2 ArgoCD

Para instalar ArgoCD se ejecutan los siguientes comandos:

```
kubectl create namespace argocd
kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

Para acceder a la página de administración, necesito la contraseña inicial. Se obtiene con el siguiente comando; el usuario es admin:

```
kubectl -n argocd get secret argocd-initial-admin-secret -o
jsonpath="{.data.password}" | base64 -d;echo
```

Y para acceder a la página, se realiza el siguiente port-forward:

```
kubectl port-forward svc/argocd-server -n argocd 8080:80
--address=0.0.0.0
```

Tras eso ya está configurado ArgoCD.

3.1.3 Crossplane

Para instalar crossplane utilizamos helm

```
helm repo add \
crossplane-stable https://charts.crossplane.io/stable
helm repo update

helm install crossplane \
crossplane-stable/crossplane \
--namespace crossplane-system \
--create-namespace
```

3.2 Configuraciones

3.2.1 AWS

Es necesaria una cuenta con los permisos de AWS para crear y gestionar los siguientes recursos:

- instancias EC2
- clúster de EKS
- redes VPC
- Tokens de acceso a la propia cuenta

Una vez se disponga de dicha cuenta, creamos el token en el siguiente apartado:

credenciales de seguridad > crear clave de acceso > seleccionamos el cuadro y crear clave de acceso > descargar archivo.csv

3.2.1 Proveedores de Crossplane

3.2.1.1 AWS

Ahora instalamos la última versión del provider de AWS de crossplane

```
cat <<EOF | kubectl apply -f -
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: provider-aws
spec:
  package:
xpkg.upbound.io/crossplane-contrib/provider-aws:v0.39.0
EOF
```

Tras eso, creamos el fichero **aws-credentials.txt** con las claves generadas en AWS con el siguiente formato:

```
[default]
aws_access_key_id = <aws_access_key>
aws_secret_access_key = <aws_secret_key>
```

Creamos un **secret** de kubernetes con las credenciales

```
kubectl create secret \
generic aws-creds \
-n crossplane-system \
--from-file=creds=./aws-credentials.txt
```

Finalmente, se crea un **providerconfig** con el secret que hemos creado

```
cat <<EOF | kubectl apply -f -
apiVersion: aws.crossplane.io/v1beta1
kind: ProviderConfig
metadata:
  name: default
spec:
  credentials:
    source: Secret
    secretRef:
      namespace: crossplane-system
      name: aws-creds
      key: creds
EOF
```

3.2.1.2 kubernetes

Instalamos el proveedor con el siguiente comando:

```
cat <<EOF | kubectl apply -f -
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: crossplane-provider-kubernetes
spec:
  package: crossplane/provider-kubernetes:main
EOF
```

3.2.2 Recursos gestionados

Para desplegar un clúster de aws de forma sencilla, se van a utilizar dos recursos gestionados:

- **eks.yaml**, de tipo **composition** que contiene parámetros para la creación del clúster, como lo son las redes que se crearán, en qué servidores, qué tipo de máquinas pueden crearse, etc..
- **definition.yaml**, de tipo **compositeResourceDefinition,** contiene los parámetros que se le van a pasar con el manifiesto final, así como los recursos a los que corresponden en composition. También contiene valores por defecto.

Ambos ficheros se encuentran en el repositorio del proyecto en la carpeta crossplane:
https://github.com/robertorodriguez98/proyecto-integrado/tree/main/crossplane
Para añadirlos, se utiliza kubectl:

```
kubectl apply -f eks.yaml && kubectl apply -f definition.yaml
```

3.2.3 Ngrok

Para que ArgoCD esté atento a los commits de github y siga así la metodología GitOps, al tenerlo instalado en un clúster local, es necesario instalar ngrok, exponiendo el puerto 8080:

```
curl -s https://ngrok-agent.s3.amazonaws.com/ngrok.asc | sudo
tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null && echo "deb
https://ngrok-agent.s3.amazonaws.com buster main" | sudo tee
/etc/apt/sources.list.d/ngrok.list && sudo apt update && sudo
apt install ngrok
```

Añadimos el token:

```
ngrok config add-authtoken [TOKEN]
```

y exponemos el puerto:

```
ngrok http https://localhost:8080
```

3.2.4 Webhook

Para que GitHub notifique a ArgoCD en el evento de un push al repositorio, tenemos que añadirlo en la configuración del mismo. Para hacerlo, en el repositorio accedemos al apartado **Settings**; en la barra lateral seleccionamos **Webhooks** y **Add webhook**.
Dentro de la página, tenemos que rellenar los siguientes campos:

- **Payload URL**: la url generada por Ngrok, con la cadena /api/webhook añadida al final.
- **Content type**: application/json.
- **Event**: Cuando se produce un push.

3.2.4 Preparación previa a la demo

Para preparar la demo, iniciaremos la aplicación de argocd **app.yaml** que se encuentra en el repositorio, ésta despliega el siguiente manifiesto:

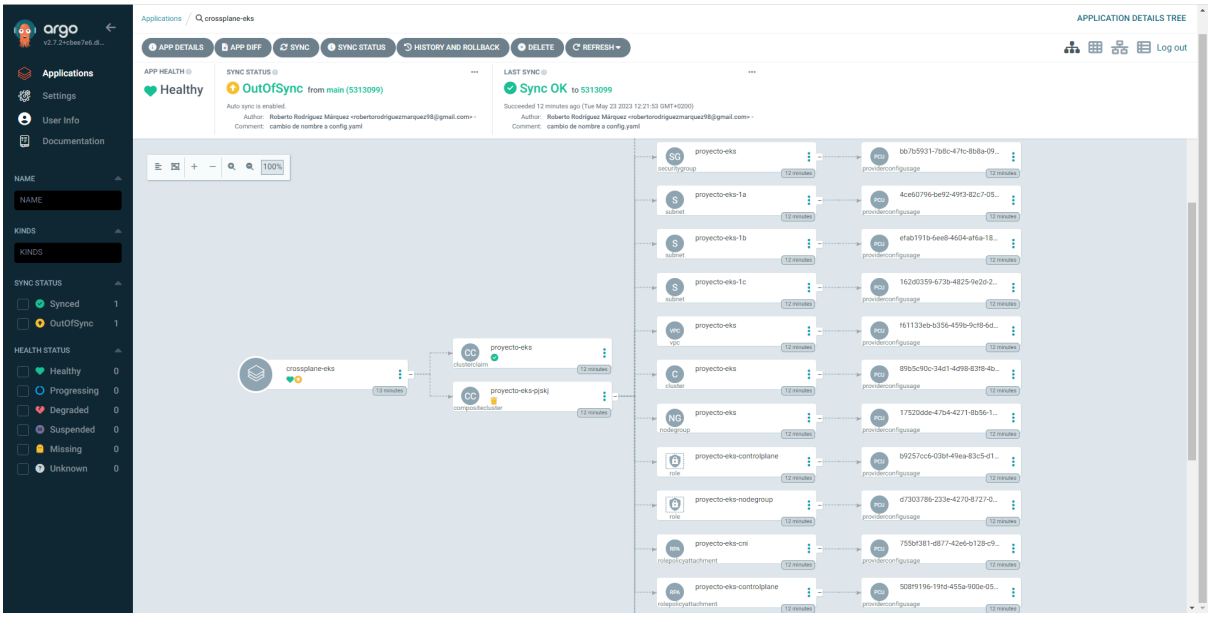
```
apiVersion: prodready.cluster/v1alpha1
kind: ClusterClaim
metadata:
  name: proyecto-eks
  labels:
    cluster-owner: robertorm
spec:
  id: proyecto-eks
  compositionSelector:
    matchLabels:
      provider: aws
      cluster: eks
  parameters:
    nodeSize: medium
    minNodeCount: 3
```

Donde podemos modificar los siguientes parámetros:

- **name**: Nombre del clúster.
- **nodesize**: tipo de máquina que se van a utilizar en los nodos. En el fichero eks.yaml se establecen los siguientes tamaños:
    - small: t2:micro
    - medium: t3.micro
    - large: t3.medium

  Los dos primeros tamaños, entran dentro de la capa gratuita de AWS.
- **minNodeCount**: mínimo de nodos que tienen que crearse en el clúster.

Tras aproximadamente **15 minutos**, el clúster estará creado junto a sus nodos. Si observamos el panel de ArgoCD de la aplicación, podemos ver todos los recursos de AWS que se han desplegado:



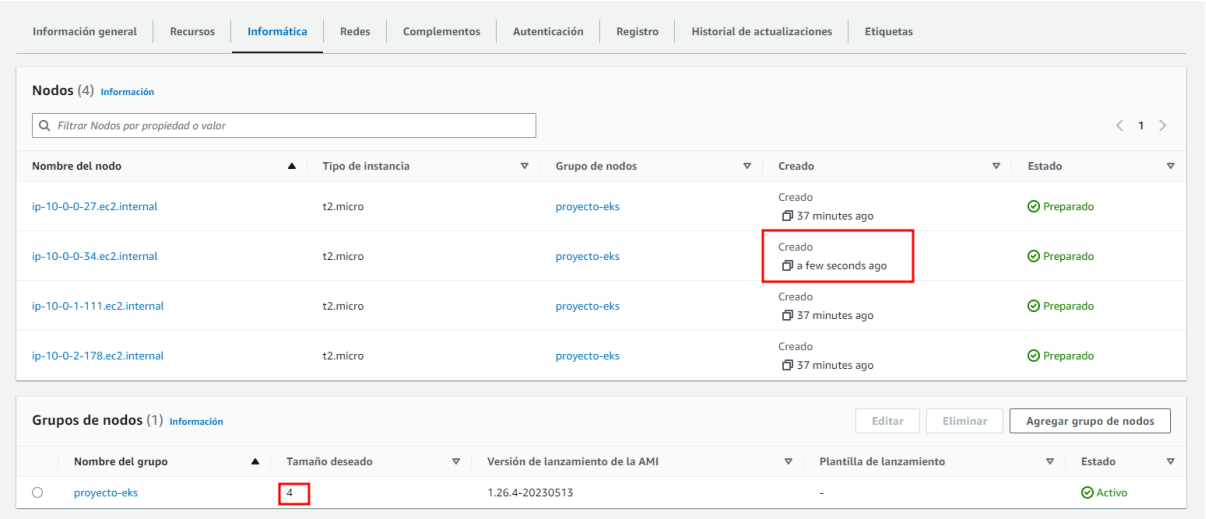
4. Demostraciones

4.1 Modificación del número de nodos

El escenario final de la demo es el siguiente:

Y el flujo de trabajo en el que consiste la primera demo, con el clúster ya desplegado es el siguiente:

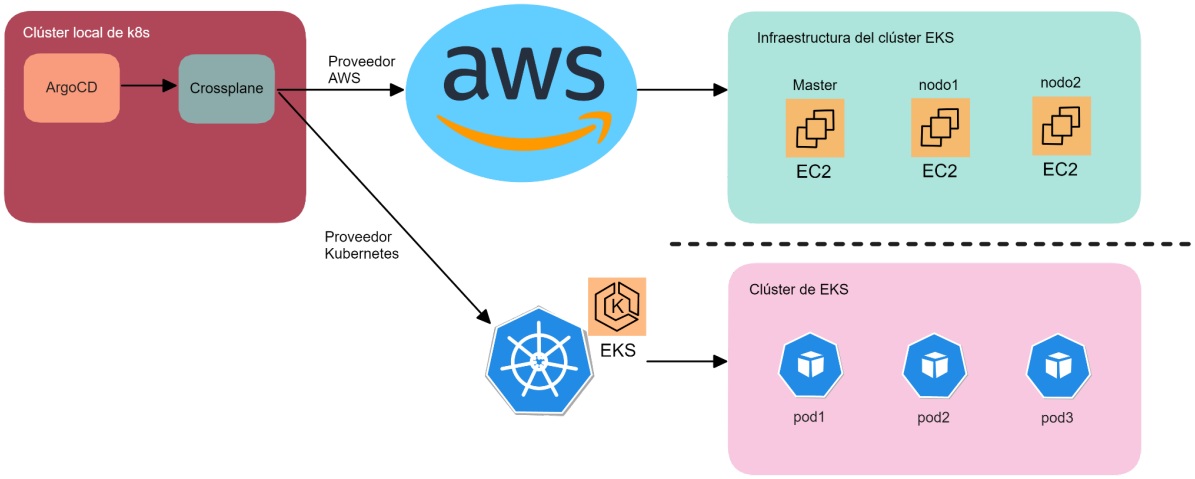
1. Se realiza un commit a **github** con un cambio en el manifiesto **aws-eks.yaml**, concretamente se cambia el número de nodos que va a tener el clúster de EKS.
2. **ArgoCD** se da cuenta de que se han realizado cambios en el repositorio, y, aplicando la metodología **gitops**, hace que en los recursos que gestiona se vean reflejados dichos cambios. Compruebo que en el panel de la aplicación se ve reflejado el cambio en el recurso **nodegroup**.
3. **Crossplane**, haciendo uso del provider de AWS, se comunica con la API referente a EKS, e indica que el número de nodos ha cambiado.
4. Finalmente, en **AWS** se hacen efectivos los cambios, por lo que podemos comprobarlo accediendo a la consola de **EKS** y viendo el nuevo nodo.



Tras ello, para demostrar el funcionamiento de Crossplane, y como este asegura que se siga el marco de trabajo **GitOps**, se añade un nodo desde la consola de AWS, mostrando como Crossplane lo detecta y vuelve a dejarlo como está definido en los recursos.

4.2 Despliegue en el clúster

Una vez con el clúster, se va a desplegar una aplicación sobre él, utilizando también crossplane. El escenario es el siguiente:



4.2.1 Configuración del proveedor de kubernetes

Para configurar el proveedor de kubernetes para que tenga acceso al clúster que acabamos de crear, se ejecutan los siguientes comandos; Primero obtenemos el **kubeconfig** y lo guardamos en un fichero:

```
kubectl --namespace crossplane-system \
    get secret proyecto-eks-cluster \
    --output jsonpath="{.data.kubeconfig}" \
    | base64 -d >kubeconfig.yaml
```

Enviamos el contenido del fichero a la variable KUBECONFIG:

```
KUBECONFIG=$(<kubeconfig.yaml)
```

Usando la variable, creamos un secret que pueda usar el proveedor:

```
kubectl -n crossplane-system create secret generic
cluster-config --from-literal=kubeconfig="${KUBECONFIG}"
```

Ahora, añadimos la configuración del proveedor usando el secret (el fichero está en la raiz del repositorio). El archivo es: [config-kubernetes.yaml]:

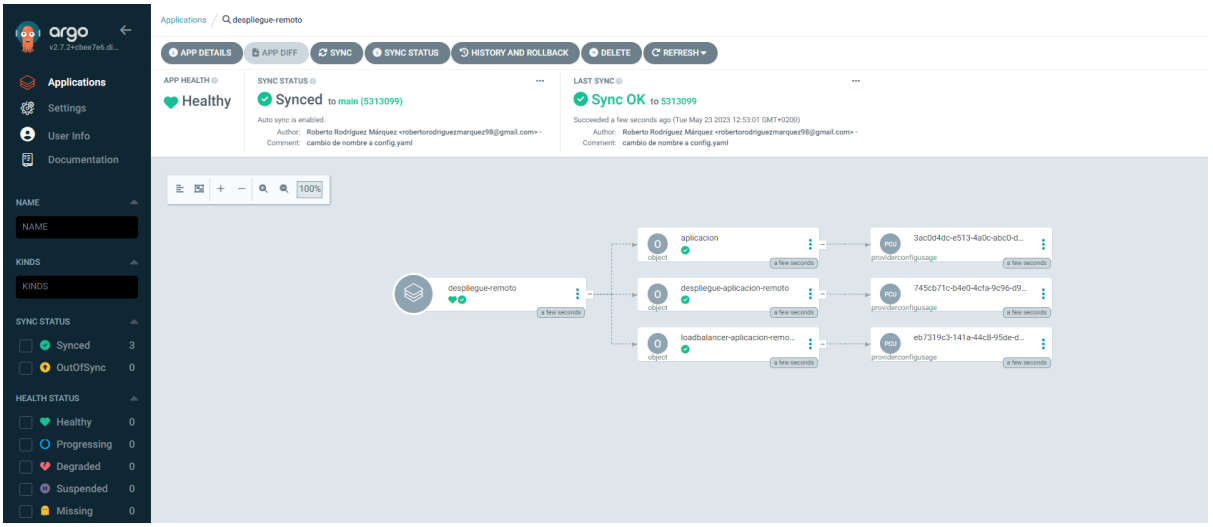
```
kubectl apply -f config-kubernetes.yaml
```

4.2.2 Script

Para facilitar la ejecución de la demo, se ha creado el script [configurar-k8s.sh] que aúna los pasos anteriores.

4.2.3 Aplicación de ArgoCD

Una vez realizados los pasos anteriores, solo queda desplegar la aplicación. Para ello se va a volver a utilizar ArgoCD; en este caso se va a utilizar la aplicación [despliegue-remoto.yaml]. Una vez desplegado, se ve así en ArgoCD



para obtener la dirección en la que está la aplicación desplegada (gracias al loadbalancer) se utiliza el siguiente comando:

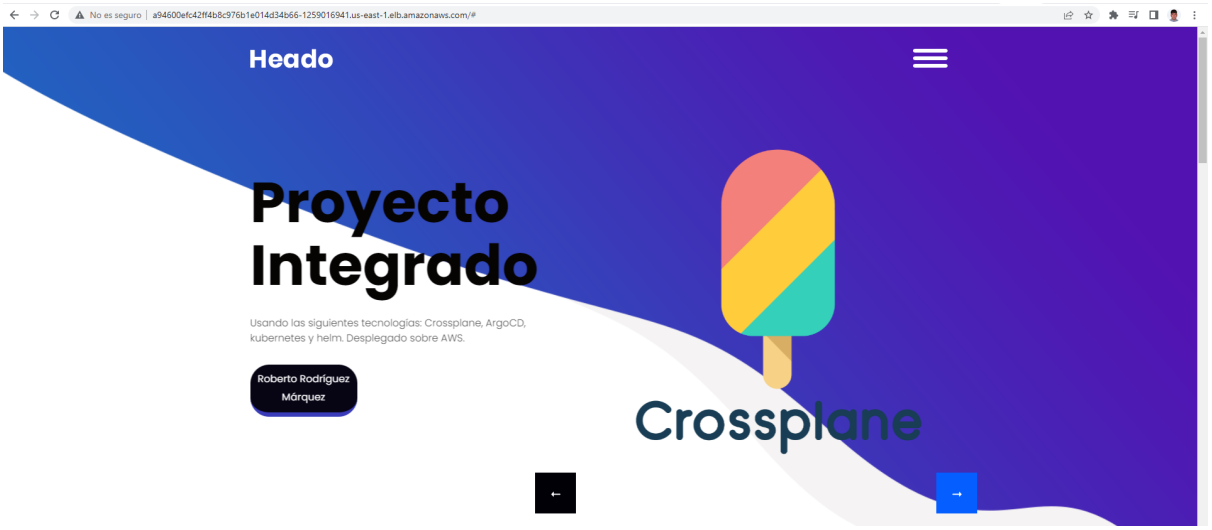
```
k describe object loadbalancer-aplicacion-remoto
```

O si queremos la dirección directamente:

```
k describe object loadbalancer-aplicacion-remoto | egrep
"Hostname"
```

```
❯ k describe object loadbalancer-aplicacion-remoto | egrep "Hostname"
            Hostname:  a94600efc42ff4b8c976b1e014d34b66-1259016941.us-east-1.elb.amazonaws.com
```

Finalmente, si accedemos a la dirección podemos visualizar la aplicación desplegada:



5. Dificultades que se han encontrado

5.1 Despliegue sobre AWS

Las principales dificultades que se han encontrado a la hora de desplegar sobre AWS son las siguientes:

- **Falta de documentación/ejemplos**: Al tratarse de una tecnología relativamente nueva y con una comunidad todavía en crecimiento, no ha sido fácil encontrar ejemplos que aplicaran la api de **AWS** para desplegar específicamente sobre **EKS**.
- **Múltiples proveedores**: A la hora de elegir el proveedor en el marketplace, resulta que para Amazon Web Services existen **dos diferentes**, y en la documentación, dependiendo de la versión que se esté consultando, usan uno u otro, teniendo éstos **diferentes métodos y llamadas**.Se ha utilizado finalmente el siguiente, debido a que hay más documentación sobre él: https://marketplace.upbound.io/providers/crossplane-contrib/provider-aws/v0.39.0
- **Coste**: El uso de EKS **no** se encuentra dentro de la **capa gratuita de AWS**, por lo que desplegar un clúster conlleva un gasto económico (aunque no muy alto).

5.2 Despliegue sobre Kubernetes

Desplegando sobre kubernetes he tenido otras problemáticas. Son las siguientes:

- **Falta de documentación/ejemplos**: Al ser una extensión de **Crossplane**, los problemas relacionados con la falta de comunidad también se aplican al proveedor de kubernetes.
- **Pocas explicaciones**: Existe un repositorio oficial con ejemplos, pero para que estos funcionen sobre un clúster en **AWS** hay que realizar **pasos extra** que no explican.
- **Caducidad**: Las credenciales que se utilizan para crear el **secret** de kubernetes que permite la conexión, caducan a los **30 minutos**.

6. Conclusión

Con la realización del proyecto hemos podido comprender mejor la **infraestructura como código**, así como la implementación de ésta por medio de una nueva tecnología que es **Crossplane**, aplicando este conocimiento a desplegar un clúster de EKS en Amazon Web Services y una aplicación dentro del mismo, haciendo uso de un entorno con **múltiples proveedores**, aprovechando las funcionalidades que este ofrece. Al ser Crossplane una herramienta tan versátil y potente, hay muchas cosas que se pueden hacer que no se han llegado a tocar en el proyecto.

También, y, a consecuencia del tema del proyecto, hemos aprendido acerca de la metodología **GitOps** y cómo aplicarla utilizando **ArgoCD**, que sirve a la perfección para este propósito gracias al funcionamiento de sus aplicaciones basadas en repositorios de GitHub.

Finalmente, se ha profundizado en el funcionamiento de **kubernetes**, aprendiendo acerca del plano de control y de las diferentes APIs que gestionan los recursos.

En conclusión, Crossplane es una herramienta muy **polifacética y útil**, siendo su principal virtud, **trasladar la infraestructura como código al terreno de kubernetes**, haciendo así que sea especialmente atractiva, y, tras un poco de aprendizaje, una herramienta a tener en cuenta de cara a **desplegar infraestructura**.

7. Bibliografía

- Crossplane on Amazon EKS (canal Containers from the Couch): https://www.youtube.com/live/aWRWKnniqeM?feature=share
- Documentación de Crossplane: https://docs.crossplane.io/v1.12/
- Proveedores de Crossplane: https://docs.crossplane.io/latest/concepts/providers/
- Documentación de la api de los diferentes proveedores:
    - AWS: https://doc.crds.dev/github.com/crossplane/provider-aws
    - Helm: https://doc.crds.dev/github.com/crossplane-contrib/provider-helm
    - K8s: https://doc.crds.dev/github.com/crossplane-contrib/provider-kubernetes
- AWS Quickstart: https://docs.crossplane.io/v1.12/getting-started/provider-aws/
- Production ready EKS Cluster with Crossplane: https://www.kloia.com/blog/production-ready-eks-cluster-with-crossplane
- GitOps model for provisioning and bootstrapping Amazon EKS clusters using Crossplane and Flux: https://aws.amazon.com/es/blogs/containers/gitops-model-for-provisioning-and-bootstrapping-amazon-eks-clusters-using-crossplane-and-flux/
- Ejemplos del proveedor de kubernetes: https://github.com/crossplane-contrib/provider-kubernetes