

**Load Balancing with**

**DevOps Arquitecture**

**ÍNDICE**

1. Introducción
2. Conceptos previos sobre DevOps
3. Tecnologías usadas
4. ¿Qué es un despliegue blue-green?
5. Ventajas y desventajas del blue-green
6. Proyecto técnico
7. Git Branching
8. Construcción de las imágenes Docker
9. Inicialización del clúster

10. Implementación de Istio

11. Despliegue de la aplicación

12. Introducción de ArgoCD

13. Desenlace

14. Bibliografía

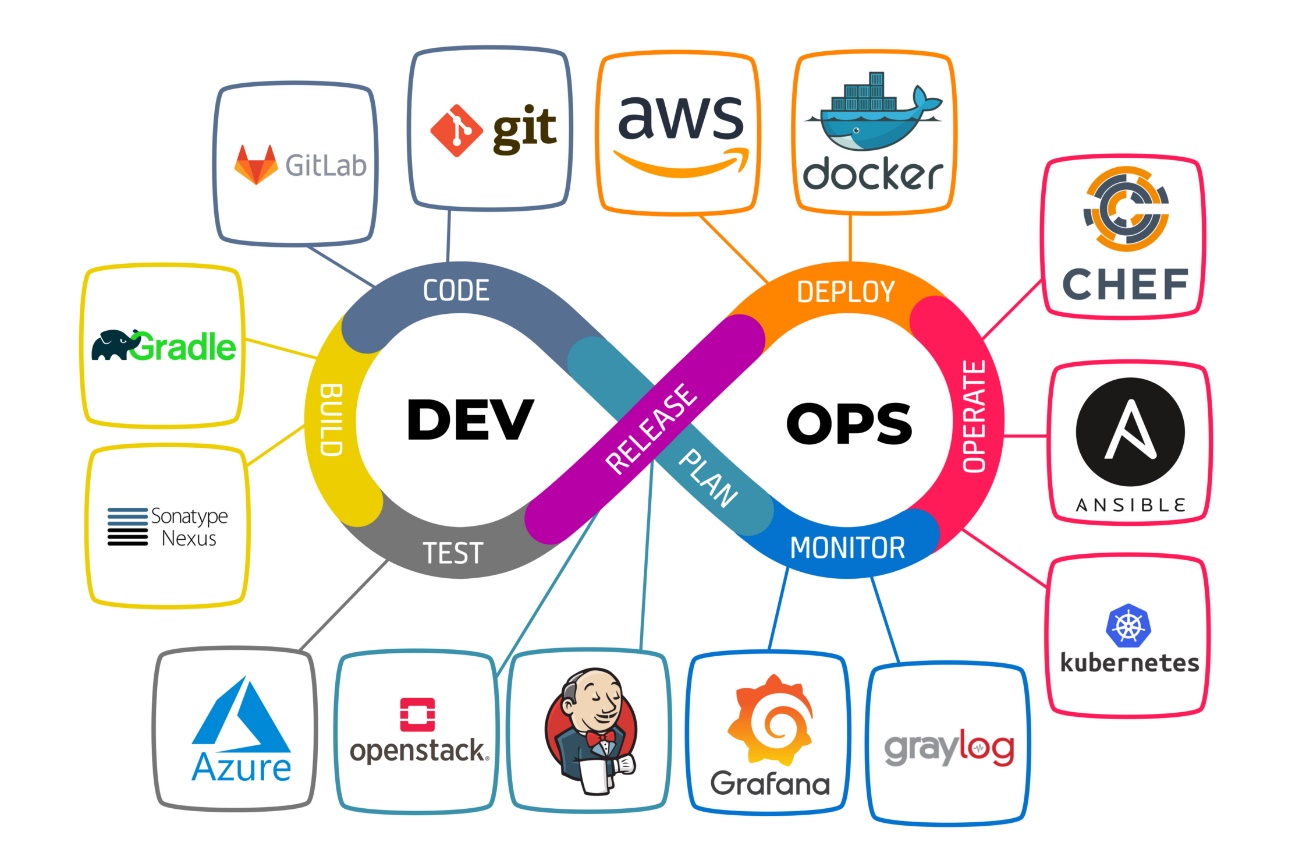
**Introducción**

Esta va a ser la documentación que voy a seguir para desarrollar mi proyecto integrado de empresa al terminar el grado superior de Administración en Sistemas Informáticos y Redes (ASIR) en el instituto I.E.S Cánovas del Castillo. Antes de meternos a empezar a explicar en que consiste mi proyecto y en enseñar como se organiza toda la infraestructura, vamos a tener un momento para aclarar ciertos conceptos que a lo mejor no entran dentro de tu ámbito o que no has oído hablar mucho sobre ellos. Para resumir antes de empezar, esta documentación va a hablar en su totalidad de explicar como se realiza un despliegue blue-green y como se le ha implementado ciertas herramientas que facilitan el arreglo de problemas al administrador y así escalar y actualizar mejor la aplicación.

**Conceptos DevOps**

Antes de empezar con el despliegue de la aplicación vamos a pararnos a ver, ¿qué es DevOps?

DevOps es una filosofía y práctica que busca integrar de manera efectiva el desarrollo de software (Dev) con las operaciones de sistemas (Ops) en un ciclo de vida de desarrollo continuo. La palabra "DevOps" proviene de la combinación de "Development" (Desarrollo) y "Operations" (Operaciones). El objetivo principal de DevOps es mejorar la colaboración y la eficiencia entre los equipos de desarrollo y operaciones para acelerar la entrega de software y mejorar la calidad del producto final.



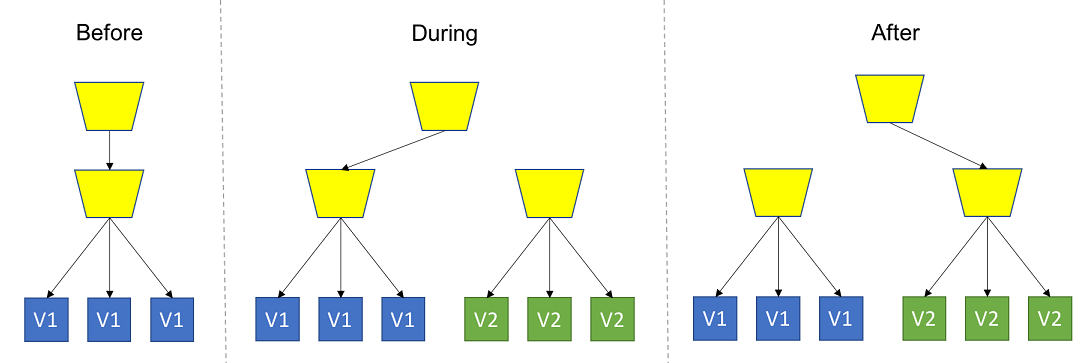
**¿Qué es un blue-green?**

El despliegue Blue-Green (azul-verde) es una estrategia utilizada en el desarrollo de software para realizar actualizaciones de aplicaciones de manera segura y sin interrupciones significativas para los usuarios. En este enfoque, se mantienen dos entornos separados, uno llamado "azul" y otro llamado "verde", y se alterna entre ellos durante las actualizaciones.

Funciona de la siguiente manera:

1. **Entorno Azul (Blue**): Este es el entorno de producción actual, que contiene la versión en funcionamiento de la aplicación. Los usuarios acceden a esta versión en el día a día.

2. **Entorno Verde (Green**): Este es el nuevo entorno que contiene la versión actualizada de la aplicación. Se prepara y prueba completamente antes de realizar la transición.



El proceso de despliegue Blue-Green; sigue estos pasos:

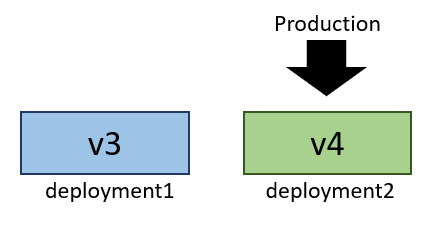
- Despliegue Inicial (Blue): Inicialmente, la versión actual (azul) está en producción y en uso.

- Despliegue de la Nueva Versión (Green): La nueva versión de la aplicación se implementa y se prueba en el entorno verde sin afectar a los usuarios finales.

- Conmutación de Tráfico: Una vez que la versión verde se ha probado y está lista, se redirige el tráfico de usuarios del entorno azul al verde. Ahora, los usuarios acceden a la nueva versión sin interrupciones en el servicio.

- Monitoreo y Validación: Se realiza un monitoreo continuo para asegurar que la nueva versión en el entorno verde funcione correctamente y no tenga problemas en producción.

- Despliegue Futuro: Si hay futuras actualizaciones, se realiza un nuevo despliegue en el entorno que no está en uso (ya sea azul o verde). Este proceso se repite, alternando entre entornos azul y verde.



**Ventajas y desventajas de un blue-green**

El despliegue Blue-Green ofrece varias ventajas, pero también tiene algunas consideraciones y desventajas:

***Ventajas:***

1. **Cero Tiempo de Inactividad (Zero Downtime**): Uno de los principales beneficios es la capacidad de realizar actualizaciones sin afectar el tiempo de actividad del sistema. Los usuarios pueden ser redirigidos de manera transparente de la versión antigua a la nueva.

2. **Reversión Rápida**: Si se encuentra algún problema con la nueva versión en el entorno "verde", es fácil revertir rápidamente al entorno "azul" para mantener la estabilidad del sistema.

3. **Pruebas Exhaustivas**: Permite realizar pruebas completas en el entorno "verde" antes de conmutar el tráfico. Esto reduce el riesgo de errores en producción.

4. **Facilita la Validación y Monitoreo**: Se puede realizar una validación continua y monitoreo durante y después del despliegue para asegurar el buen funcionamiento de la nueva versión.

5. **Escalabilidad**: Puede facilitar la escalabilidad horizontal, ya que es posible agregar más instancias al entorno "verde" para manejar cargas crecientes antes de redirigir el tráfico.

***Desventajas:***

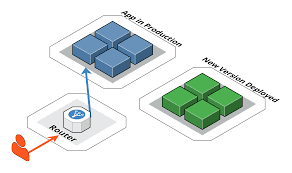
1. **Infraestructura Duplicada**: Se requiere mantener dos conjuntos de infraestructura (entornos "azul" y "verde"), lo que puede aumentar los costos y la complejidad de la gestión.

2. **Sincronización de Datos**: Si hay cambios en la base de datos u otros componentes que requieren sincronización, puede ser necesario implementar estrategias adicionales para asegurar la consistencia.

3. **Complejidad Operativa**: La gestión de dos entornos puede resultar compleja y requerir herramientas y procesos adicionales para garantizar una transición suave y la coexistencia de las dos versiones.

4. **Posible Latencia**: Durante la conmutación de tráfico, puede haber una breve latencia mientras los usuarios son redirigidos de un entorno a otro.

5. **Requiere Automatización**: Para aprovechar al máximo las ventajas del despliegue Blue-Green, se necesita una automatización robusta para gestionar la implementación y conmutación de tráfico.



A pesar de estas desventajas, el despliegue Blue-Green sigue siendo una estrategia popular y eficaz para implementar actualizaciones de software de manera segura y con mínimo impacto en la experiencia del usuario. La elección de utilizar esta estrategia dependerá de los requisitos específicos del proyecto y de las preferencias del equipo de desarrollo y operaciones.

**Tecnologías usadas**

A continuación, voy a hacer una breve descripción de todas las tecnologías que he usado para llevar a cabo este proyecto:

Aquí tienes descripciones breves de cada una de las tecnologías mencionadas:

1. **Docke**r:

- **Descripción**: Docker es una plataforma de contenedorización que permite a los desarrolladores empaquetar una aplicación y todas sus dependencias en un contenedor único. Los contenedores son unidades ligeras y portátiles que pueden ejecutarse de manera consistente en cualquier entorno que tenga Docker instalado.

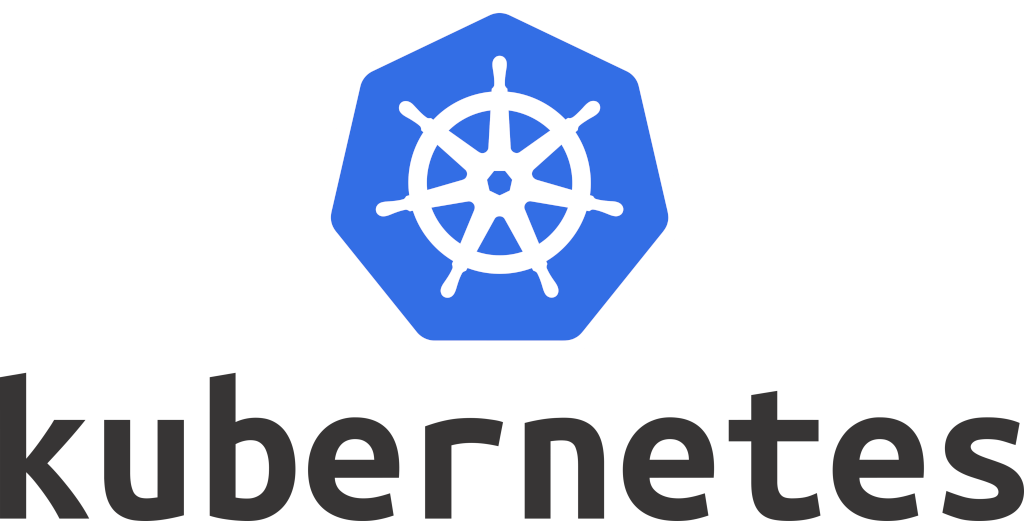
- **Características clave**: Portabilidad, aislamiento, eficiencia de recursos.



2. **Kubernetes**:

- **Descripción**: **Kubernetes** es un sistema de orquestación de contenedores de código abierto que automatiza el despliegue, la escala y la gestión de aplicaciones en contenedores. Permite la gestión eficiente de aplicaciones distribuidas y microservicios en entornos escalables.

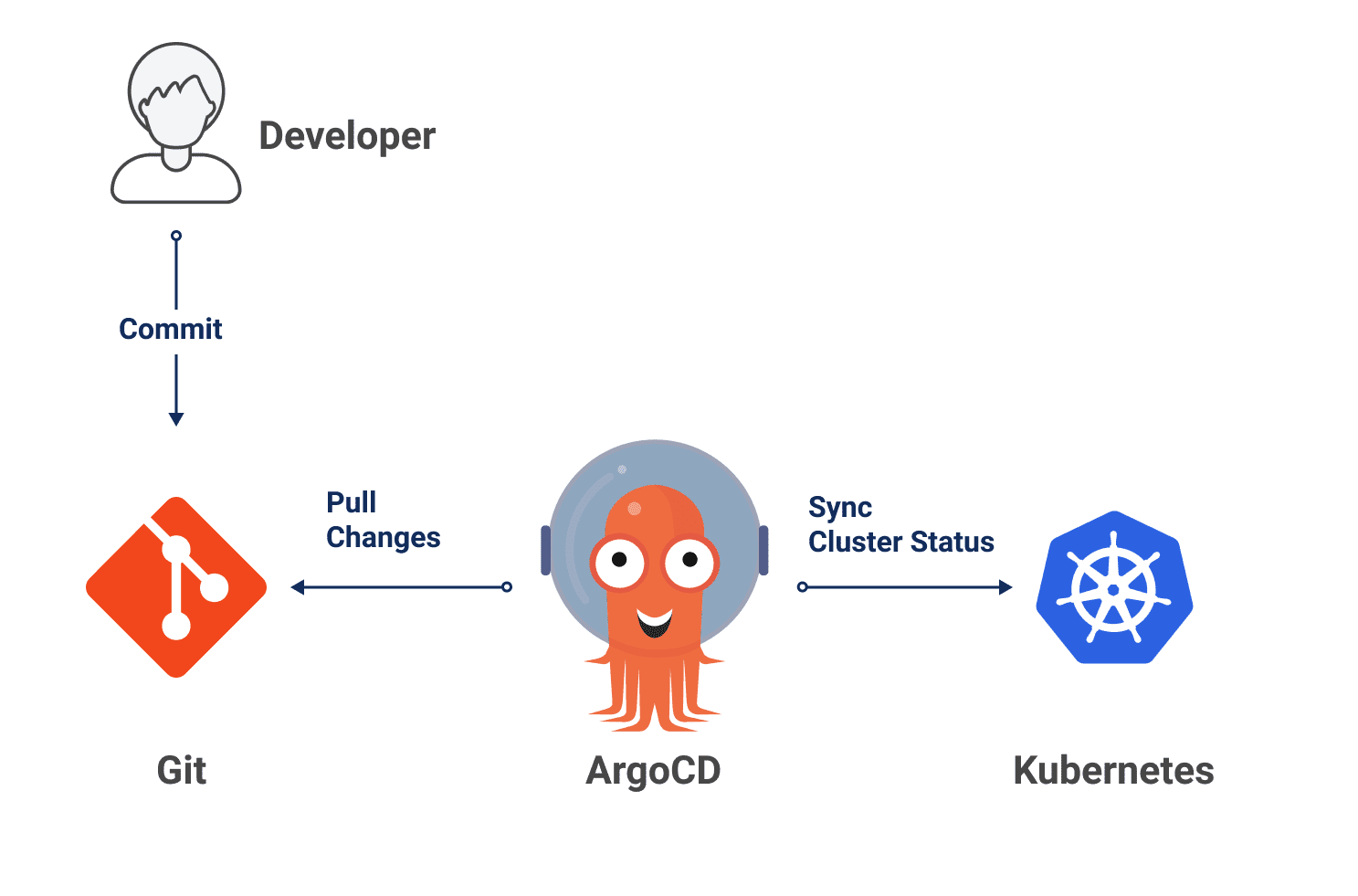
- **Características** **clave**: Orquestación, escalabilidad, alta disponibilidad.



3. **ArgoCD**:

- **Descripción**: ArgoCD es una herramienta de entrega continua (Continuous Delivery) que se utiliza para implementar y mantener aplicaciones en entornos Kubernetes. Permite declarar y mantener el estado deseado de las aplicaciones en Kubernetes a través de archivos de configuración declarativos almacenados en repositorios Git.

- **Características** **clave**: Entrega continua para Kubernetes, sincronización de estado, automatización de implementaciones.



4. **Istio**:

- **Descripción**: Istio es una plataforma de servicio de malla (Service Mesh) de código abierto que facilita la conexión, la seguridad y el control de tráfico entre servicios en una arquitectura de microservicios. Proporciona funciones como el enrutamiento inteligente, la gestión de tráfico y la observabilidad.

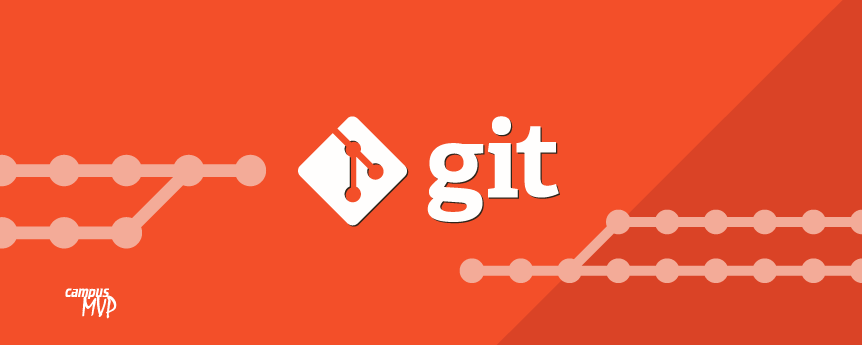
- **Características** **clave**: Control de tráfico, seguridad, monitoreo.



5. **Git**:

- **Descripción**: Git es un sistema de control de versiones distribuido ampliamente utilizado para el seguimiento de cambios en el código fuente durante el desarrollo de software. Permite a los equipos de desarrollo colaborar de manera eficiente y rastrear la historia de los cambios en el código.

- **Características** **clave**: Control de versiones distribuido, ramificación, fusión.



6. **GitHub**:

- **Descripción**: GitHub es una plataforma basada en la web que utiliza Git para el control de versiones y proporciona herramientas adicionales para la colaboración en equipo. Permite a los desarrolladores alojar, revisar y colaborar en proyectos de software de manera eficiente.

- **Características** **clave**: Repositorios Git alojados, seguimiento de problemas, colaboración social.



**Proyecto técnico**

Como he mencionado antes mi proyecto técnico va a ser un despliegue blue-green. Una vez visto de manera teórica que es y visto también todas las tecnologías podemos pasar a exponer de manera gráfica como se monta un servicio blue-green.

Empezaremos exponiendo la organización que he llevado acabado en git para trabajar en el proyecto, pasando por todos los pasos a seguir para montar nuestro despliegue. Toda la documentación y código estarán disponibles en el siguiente repositorio de GitHub, por si queréis desplegarlo por vuestra cuenta.

<https://github.com/Totinbolo45/Proyecto>

También de forma ilustrativa dejó un gráfico donde se muestra el funcionamiento del proyecto en si con sus tecnologías y su infraestructura

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Git Branching**

Para organizar el proyecto he utilizado la creación de ramas que ofrece git para ir subiendo los distintos ficheros de configuración y organizar el proyecto para futuras implementaciones que se le hagan y también prepararlo por si se quisiera implementar más participantes al proyecto.

La jerarquía de ramas es la siguiente:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Tenemos nuestra rama main, en la cuál esta el proyecto desarrollado y puesto en producción es decir es el producto final que se le ofrece al cliente.

La rama de Pre-Producción es la rama que se va a usar para subir los cambios hechos en desarrolló, es la rama que también se va a usar para testear y debuggear el proyecto antes de subirlo a producción/main

Por último, la rama de desarrolló, es la rama que vamos a usar para desarrollar cualquier nueva implementación al proyecto o para testeos previos de alguna nueva futura versión (Ejemplo; implementación de un clúster en AWS en vez de en local)

**Construcción de las imágenes Docker**

**Requisitos previos:**

* Tener Docker desktop o Docker engine instalado en una máquina virtual o en tu máquina personal
* Habilitar una cuenta en Docker Hub
* Conocimiento de comandos básicos de construcción de imágenes en Docker y como subirlas a Docker Hub.

**Apartado práctico**

Comenzaremos con la primera parte de nuestro proyecto, que es la construcción de imágenes Docker para poder buildear las dos webs que vamos a usar.

Previamente instalaremos Docker desktop o engine en nuestra máquina, siguiendo los pasos de la documentación oficial de Docker.

-Enlace a la documentación: <https://docs.docker.com/get-docker/>

Una vez instalado prepararemos el código que contendrá la página web

En mi caso usare este código y da la siguiente página web

Texto

Descripción generada automáticamente

De resultado la siguiente página web:

Interfaz de usuario gráfica

Descripción generada automáticamente

Una vez tengamos esto pasaremos a buildear la imagen Docker que usaremos para montar nuestro contenedor. Usaremos un servidor apache y introduciremos en su configuración la web que acabamos de usar.

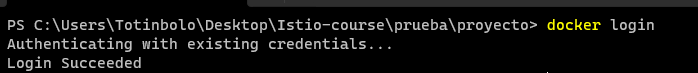
Texto

Descripción generada automáticamente

Abriremos una terminal

Y nos logearemos en Docker con nuestro usuario

**Docker login**



Una vez logeados ejecutaremos el siguiente comando para buildear las imágenes;

“**docker build -t blue .”**

Texto

Descripción generada automáticamente

Esto nos creara el contenedor que hemos especificado, una vez así podemos comprobar haciendo un localhost al puerto 80 que nos aparecerá nuestra web

Previamente ejecutaremos el comando;

**Docker run -p 80:80 [Hash de la imagen docker]**

Texto

Descripción generada automáticamente

En el navegador buscaremos <http://localhost>

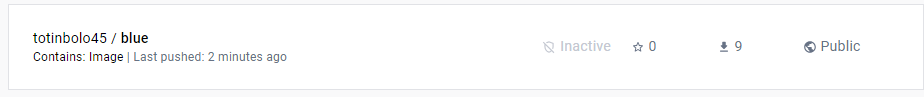
Gráfico

Descripción generada automáticamente con confianza baja

Y efectivamente comprobamos que nos lleva a nuestra web.

Ahora subiremos la imagen a Docker hub para poder implementarla de manera futura en nuestro proyecto

Previamente crearemos un repositorio en Docker hub, teniendo de nombre blue



Le asignaremos una etiqueta a nuestra imagen, usando el siguiente comando

“**docker tag blue totinbolo45/blue:v2**”



Y pushearemos la tag a nuestro repositorio en Docker hub

“**Docker push totinbolo45/blue:v2**”

Texto

Descripción generada automáticamente

Nos iremos a Docker hub y comprobaremos que tenemos la imagen subida correctamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Una vez comprobado que está subido repetiremos los pasos con la imagen que contendrá nuestra imagen green

Crearemos la página green:

Texto

Descripción generada automáticamente

Muestra por el navegador:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Buildearemos ahora la imagen

Texto

Descripción generada automáticamente

La lanzaremos para comprobar que funcione

Texto

Descripción generada automáticamente

Buscamos en el navegador <http://localhost>

Texto

Descripción generada automáticamente

Crearemos el repositorio green en Docker hub



Y ahora crearemos la etiqueta y la pushearemos al repositorio de Docker hub

Texto

Descripción generada automáticamente

Y comprobaremos que se ha subido correctamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Inicialización del clúster**

Como hemos comentado vamos a utilizar los servicios de Kubernetes para poder implementar todos los servicios necesarios para los contenedores y aprovechar la fácil escalabilidad que ofrece el servicio de Kubernetes.

**Requisitos previos:**

* Tener instalado Kubernetes-cli en tu máquina virtual o en tu propia máquina personal
* Instalar los servicios de clúster de minikube
* Previamente haber hecho los pasos anteriores de Docker.

**Apartado práctico**

Empezaremos la instalación de Kubernetes-cli en nuestra máquina. Para ello tendremos dos maneras de hacerlo. La primera es siguiendo la documentación de Kubernetes.

-Enlace a la documentación: <https://kubernetes.io/docs/tasks/tools/>

Una vez hayamos instalado los servicios de Kubernetes, necesitaremos un clúster para poder utilizar los servicios. Para esto tenemos dos opciones posibles:

1. Usamos un clúster local; como minikube.
2. Utilizamos un servicio de Kubernetes en un entorno cloud; como un EKS (AWS kubernetes Service) o un AKS (Azure Kubernetes Service)

En nuestro caso por ahorrar costes económicos vamos a usar un clúster local con minikube.

Para instalar minikube, seguiremos la documentación oficial de Kubernetes para instalarlo en nuestra máquina

-Enlace a la documentación: <https://minikube.sigs.k8s.io/docs/start/>

Una vez tenemos minikube instalado, iniciaremos el minikube con la siguiente sentencia

“**minikube start --memory=6000 --driver=virtualbox --no-vtx-check**”

Texto

Descripción generada automáticamente

Esperaremos hasta que se inicié

Texto

Descripción generada automáticamente

Una vez se inicié podremos empezar a aplicar la configuración de los servicios que vamos a usar.

**Implementación de Istio**

Uno de los servicios que vamos a implementar es Istio, para poder facilitarnos la implementación del despliegue blue-green y poder aplicar las reglas del balanceo del tráfico de una manera más fácil. No obstante veremos una prueba como se podría implementar este servicio sin Istio y así demostrar la facilidad del uso de este.

**Requisitos previos:**

* Tener los ficheros de instalación de Istio (<https://github.com/DickChesterwood/istio-fleetman/tree/master/_course_files/x86_amd64/1-Telemetry>)
* Previamente haber instalado minikube y kubernetes-cli

**Apartado práctico**

Para poder implementar Istio en nuestro clúster tenemos que aplicar los ficheros de configuración. Para ello he dejado los ficheros ordenados en función de cuál debemos aplicar primero.

Aplicaremos todos los ficheros con el comando: “**kubectl apply -f**”

Texto

Descripción generada automáticamente

El primer fichero lo que hará será instalar las características de Istio para poder implementarlo en nuestro servicio de Kubernetes

Ahora aplicaremos el siguiente fichero de documentación

Texto

Descripción generada automáticamente

Este fichero de configuración instalará en el clúster herramientas para poder usar Istio, entre ellas el inyector de proxys para usar Istio, la consola Kiali para visualizar el tráfico de nuestra aplicación, grafana y prometeus para monitorizar el uso de recursos de nuestro clúster.

Imagen que contiene Texto

Descripción generada automáticamente

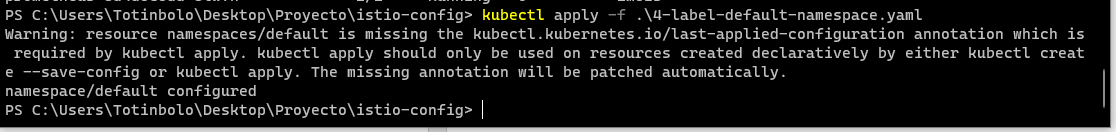
Esperaremos a que los contenedores estén inicializados para poder empezar a implementar el proyecto

Texto

Descripción generada automáticamente con confianza baja

Una vez iniciados los PODs pasaremos a implementar el inyector en el namespace default

El tercer fichero de configuración es redundante ya que su función es permitirte logearte en la consola Kiali, pero Kiali ya no utiliza login para poder entrar por lo cuál no hace falta implementarlo



El cuarto fichero nos permitirá aplicar los proxys de Istio para poder aplicar las configuraciones. Si hacemos un describe del namespace default, veremos que el inyector esta aplicado

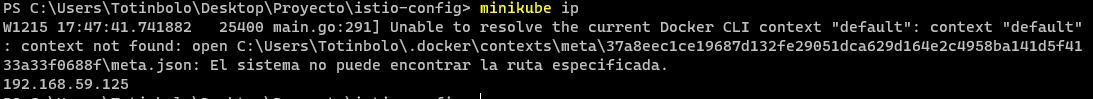
“**kubectl describe ns default**”

Texto

Descripción generada automáticamente

Una vez aplicado todo lo que sería la configuración de Istio podremos acceder a las diferentes herramientas mediante la IP que nos brinda minikube y los puertos asignados

Ejecutar: “**minikube ip**”, para obtener la ip del minikube



Una vez obtenida la IP miramos en los distintos puertos

Kiali: 31000

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Jaeger UI: 31001

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Grafana-Prometheus: 31002

Captura de pantalla de computadora

Descripción generada automáticamente

Ahora que conocemos las herramientas pasemos a desplejar nuestra aplicación y ver como la vamos a implementar usando solo Kubernetes y luego implementaremos Istio para aplicar unas mejores prácticas

**Despliegue de la aplicación**

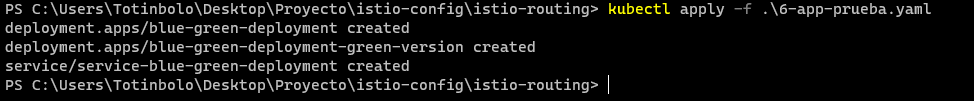
Ahora que hemos desplegadas las herramientas para montar y monitorear nuestra aplicación, vamos a pasar a ir creando nuestra aplicación y prepararla para el despliegue y la implementación de Istio.

Primero que nada, crearemos el despliegue de nuestra aplicación en un fichero de extensión “yaml”

Texto

Descripción generada automáticamente

Una vez lo tenemos listo lo vamos a aplicar en el clúster



Comprobamos que los pods estén levantados correctamente

Texto

Descripción generada automáticamente

Una vez están levantados, con la IP del minikube entraremos al puerto especificado en el fichero (30080 en este caso)

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Esto nos mostrará por pantalla de manera aleatoria uno de los dos websites creados y si refrescamos irá balanceando la carga entre ellos de manera aleatoria o respetando un “50% 50%”

Interfaz de usuario gráfica

Descripción generada automáticamente

Como podemos ver en el mismo puerto se muestra la otra página

Observamos también en el Kiali que el tráfico es entrante por ambos PODs sin problema

Interfaz de usuario gráfica

Descripción generada automáticamente

Hasta aquí podríamos decir que tenemos un balanceo de carga realizado, pero es bastante inestable en cuanto a recibir peticiones, ya que realiza un 50%50%, bastante extraño. Una duda que surge es, ¿Y si no quiero hacer que el balanceador haga un 50-50, como lo puedo implementar?

Imagen que contiene Texto

Descripción generada automáticamente

Pues es posible hacerlo, pero digamos que es un procedimiento un tanto tedioso, como primer paso se deberían crear tantas replicas o PODs para crear ese otro porcentaje. Supongamos que queremos hacer un 70%-30%. El número de PODs a crear debería ser de 70 en el despliegue del blue y de 30 PODs en el despliegue del green, y digamos que no es que sea una buena práctica y que consumiría muchos más recursos de lo que deseamos nosotros. Es aquí donde empezaremos a aplicar las configuraciones de Istio para poder llevar a cabo el enrutamiento que nosotros queremos.

Cabe aclarar primero, que Kiali permite crear las configuraciones de enrutamiento desde una opción en los detalles del servicio creado.

Interfaz de usuario gráfica

Descripción generada automáticamente

Igualmente yo haré toda la configuración a mano para este proceso.

Crearemos un fichero para crear todo el enrutamiento (servicio virtual, reglas de entrada y reglas de destino) que usará Istio para poder generar este balanceo de carga.

Este es el código que vamos a aplicar para crear el enrutamiento:

Servicio Virtual

Texto

Descripción generada automáticamente

Regla de Destino

Texto

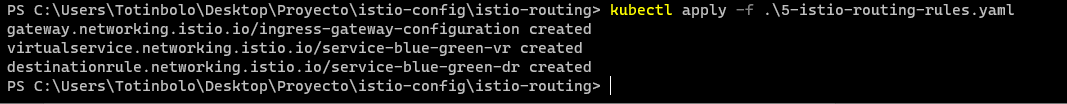
Descripción generada automáticamente

Regla de Entrada para el routing

Texto

Descripción generada automáticamente

Una vez tenemos todo esto creado, vamos a aplicar las reglas de enrutamiento en el clúster



Si nos dirigimos a Kiali comprobaremos que el servicio tiene un logo encima aplicado

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Eso significa que la configuración de enrutamiento esta aplicada al servicio. Ahora si volvemos a la página de nuestra web veremos que algo raro ocurre.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Nosotros hemos especificado que el blue tiene un 100 del tráfico, pero siguen entrando peticiones al green. ¿Qué ocurre aquí? ¿Por qué no se han aplicado los cambios? ¿Por qué Kiali entonces muestra que está aplicado la configuración del enrutamiento?

Esto tiene solución fácil, si hacemos el siguiente comando:

“kubectl get svc -n istio-system”

Esto nos mostrará los servicios de Istio instalados en el clúster.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Si miramos el servicio de “istio-ingressgateway” y miramos el tipo y los puertos asignados, comprobaremos que Istio configura un balanceador de carga en el que se aplican las reglas de enrutamiento y los puertos por los que sale. Podemos ver que el puerto es el 31380 y nosotros al estar en un clúster local nuestro balanceador cogerá la IP del clúster local, es decir, la IP del minikube.

Nuestro fichero yaml actual del servicio, esta configurado por nodePort por lo cuál deberemos cambiar esa configuración. La configuración por aplicar será cambiar nodePort por ClusterIP.

Texto

Descripción generada automáticamente

Una vez cambiado esto, con la IP del minikube y con el puerto del balanceador de carga comprobaremos que el despliegue blue es el que está aplicado.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Comprobamos que efectivamente muestra por pantalla el blue como le hemos configurado.

Texto

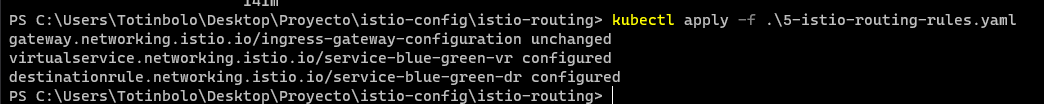
Descripción generada automáticamente con confianza media

Vamos a hacer la prueba ahora aplicando un 100 al weight del green y aplicaremos los cambios y veremos que ahora nos debería mostrar el green por la misma IP y puerto.

Texto

Descripción generada automáticamente

Aplicaremos los cambios mediante un apply de las reglas de enrutamiento en el clúster



Comprobaremos que las reglas se habrán cambiado y ahora si refrescamos la web anterior…

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Como se puede comprobar ahora se ha desplegado el green y ahora si refrescamos vamos a comprobar que solo muestra el green y no el blue

Texto

Descripción generada automáticamente

Comprobaremos en el Kiali que al poner el balanceador de carga ahora se ha creado una nueva figura en la consola que indica que se está usando ese balanceador para acceder a la web en función de lo que le hayamos configurado al fichero.

Interfaz de usuario gráfica

Descripción generada automáticamente

Podemos comprobar también con Jaeger UI todas las trazas entrantes al despliegue y comprobamos que son trazas que no han tardado mucho en responder y pasan todas por este balanceador de carga antes de llegar a los PODs

Captura de pantalla de computadora

Descripción generada automáticamente

Una vez aquí ya tenemos el balanceador configurado y nos quedaría una última función que agregar a nuestro proyecto.

**Introducción de ArgoCD**

**Requisitos previos:**

* Instalar ArgoCD en tu clúster
* Tener el clúster de Kubernetes funcionando correctamente y con tu aplicación desplegada dentro de él
* Tener un repositorio en GitHub con toda la configuración de tu aplicación

**Apartado práctico**

Vamos a aplicar una última configuración a nuestro clúster y es implementarle ArgoCD. ArgoCD vamos a utilizarlo para implementar nuevos cambios que se realicen en el proyecto o para desplegar de manera manual la nueva configuración que se vaya agregando.

Para ello vamos a instalar el ArgoCD en nuestro clúster.

Crearemos primero el namespace de ArgoCD, que es donde instalaremos todos los PODs y servicios que requiere ArgoCD

“**kubectl create ns argocd**”



Una vez tenemos el namespace de ArgoCD, vamos a instalar los servicios y PODs de argocd. Para ello ejecutaremos el siguiente comando con la URL del repositorio de descarga oficial de ArgoCD

“kubectl apply -f <https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml> -n argocd”

Texto

Descripción generada automáticamente

Esperaremos hasta que los PODs se creen correctamente

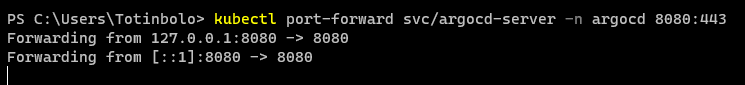
Texto

Descripción generada automáticamente

Una vez iniciados entraremos en argocd y sacaremos la contraseña que necesitamos para acceder a él

Abrimos otra terminal y ejecutaremos el siguiente comando

“**kubectl port-forward svc/argocd-server -n argocd 8080:443**”



Esto lo que hará es abrir el puerto 8080 para poder acceder al ArgoCD y que podamos operar con él

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Esta ventana es la que aparecerá cuando entramos a <http://localhost:8080>

El usuario será “admin” pero la contraseña viene establecida de manera secreta en el yaml. Para sacarla ejecutaremos el siguiente comando.

“**kubectl get secret argocd-initial-admin-secret -n argocd -o yaml**”

Texto

Descripción generada automáticamente

Esa contraseña que nos da viene encriptada por el propio ArgoCD, tendremos que desencriptarla usando una consola de bash. Le pasaremos este comando

“**echo "TC1oWC02b1dvTUd4QXlSLQ==" | base64 -d**”

Y nos dará la contraseña desencriptada

Texto

Descripción generada automáticamente

Ahora nos iremos al ArgoCD y nos logearemos

Captura de pantalla de un celular

Descripción generada automáticamente

Una vez logeados ya podremos empezar a crear nuestros despliegues de la aplicación

Le daremos a “new app”

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Nos llevará a un panel que tendremos que rellenar para que despliegue la aplicación

En el panel de general lo rellenaremos con estos campos:

Captura de pantalla de un celular

Descripción generada automáticamente

El panel de Source lo rellenaremos con estos campos:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Estos campos tendrán que ser la URL de nuestro repositorio de Github donde estamos subiendo los cambios y los diferentes ficheros de configuración con la rama en la que se esta configurando todo y la ruta donde se encuentran nuestros ficheros con la configuración de los servicios y PODs

El resto de la configuración se dejará por defecto y le daremos a crear

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Esto creará nuestra aplicación en el argocd y desde ahí podremos operar con ella

Captura de pantalla de un teléfono celular

Descripción generada automáticamente

Entraremos dentro de ella y veremos la estructura que tiene

Captura de pantalla de un celular

Descripción generada automáticamente

Ahí podremos ver todos los servicios y PODs configurados. Comprobaremos que algunos de ellos muestran que están desincronizados, vamos a sincronizarlos para que este ArgoCD funcionando correctamente

Texto

Descripción generada automáticamente

Pulsaremos en sincronizar y veremos como ahora todos los servicios se sincronizan correctamente

Captura de pantalla de un celular

Descripción generada automáticamente

Comprobaremos que todo está sincronizado correctamente (la regla de destino y servicio virtual se quedan desincronizado por ser configuración de Istio que no reconoce Argo correctamente ocurre también Visual Studio Code)



Vamos a hacer una prueba de sincronización. Vamos a subir a nuestro repositorio los nuevos cambios en los PODs con las versiones.

Texto

Descripción generada automáticamente

Aquí muestro el proceso entero de pusheo de cambios al repositorio.

Nos dirigimos a ArgoCD, y sincronizaremos la aplicación

Captura de pantalla de computadora

Descripción generada automáticamente

Comprobaremos que ArgoCD comienza a sincronizar el despliegue de la aplicación y a cerrar los PODs antiguos y levantar los nuevos con la nueva versión que se le ha subido a GitHub

Comprobamos que la web sigue funcionando correctamente

Interfaz de usuario gráfica

Descripción generada automáticamente

Llegados aquí nuestra aplicación estaría levantada por completo y funcionando todas sus herramientas implementadas de momento.

**Desenlace**

Para terminar con el proyecto expuesto, voy a dar unas futuras implementaciones del proyecto con futuras versiones que se van a agregar de cara a mejorar el proyecto.

* Versión 2.0.0: Integración de la aplicación en un servicio de Kubernetes de AWS (EKS), para así poder mejorar la auto escalabilidad del clúster a la hora de ir añadiendo más PODs y mejoras al proyecto



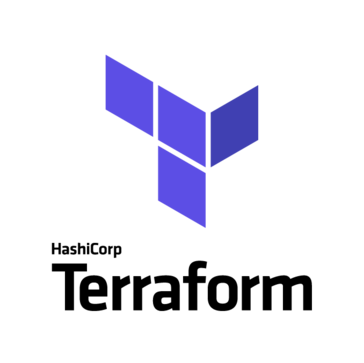
* Versión 2.5.0: Mejora de las implementaciones de Istio, agregando más ventajas de Istio, ya que las tratadas aquí son solo una de las pocas características que viene a ofrecer



* Versión 3.0.0: Aplicación de CI, mediante el uso de herramientas como pueden ser Gitlab, Azure Devops o Jenkins (está por ver cuál usar)



* Versión 4.0.0: Integración de Terraform, como IAC para automatizar el montaje de cierta infraestructura



**Bibliografía**

-Cuenta Github personal: <https://github.com/Totinbolo45>

-Cursos usados para aprender las tecnologías del proyecto:

-Kubernetes: <https://www.udemy.com/course/learn-kubernetes/>

-Istio-Service Meshing with Kubernetes: <https://www.udemy.com/course/istio-hands-on-for-kubernetes/?referralCode=36E4FA521FB5D6124156>

-Docker: <https://www.udemy.com/course/learn-docker/>

-Git y Github: <https://www.udemy.com/course/git-github/>