Universidad Zaragoza
1542

Trabajo Fin de Grado

# Soporte y estandarización de una arquitectura tecnológica de referencia para el despliegue de microservicios complejos mediante técnicas DevOps y GitOps

# Support and standarization of a reference technology architecture for the deployment of complex microservices using DevOps and GitOps techniques


Autor

Jorge Sanclemente Vilda

Director

David Román Esteban

Ponente

Francisco Javier Zarazaga Soria



ESCUELA DE INGENIERÍA Y ARQUITECTURA
2024

# AGRADECIMIENTOS

Quiero expresar mi gratitud a todas las personas que, de una manera u otra, han contribuido a la realización de este Trabajo de Fin de Grado.

En primer lugar, quiero expresar mi profunda gratitud a mis padres, por su apoyo incondicional y por creer en mí en todo momento. Su confianza, amor y apoyo emocional han sido fundamentales para la realización de este Trabajo de Fin de Grado.

Agradezco también a mi tutor David Román por su orientación, sugerencias y experiencia, pero sobre todo por darme la oportunidad de poder realizar este TFG en este departamento, ha supuesto un gran aprendizaje para mí.

Extiendo mi agradecimiento a mi ponente, Francisco Javier Zarazaga Soria, por su disposición continua y la gran ayuda que me ha proporcionado en la estructuración y realización de la memoria de este trabajo de fin de grado. Su orientación ha sido clave para la culminación de esta memoria.

Además, me gustaría agradecer también a todos los profesores que he tenido durante todos estos años de universidad por compartir sus conocimientos y por despertar el interés en aprender continuamente cosas nuevas.

Por último, agradecer a todos los compañeros de "hiberus"por su paciencia y por compartir su experiencia y conocimientos conmigo.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

# DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe remitirse a seceina@unizar.es dentro del plazo de depósito)

D./Dª. Jorge Sanclemente ,

en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de Estudios de la titulación de

Grado en Ingeniería Informática (Título del Trabajo)

"Soporte y estandarización de una arquitectura tecnológica de referencia para el despliegue de microservicios complejos mediante técnicas DevOps y GitOps"

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 14 de Junio de 2024

Fdo: Jorge Sanclemente Vilda

# RESUMEN

Este TFG realizado en la empresa "Hiberus Tecnología", tiene como objetivo dos propósitos fundamentales. Por un lado, y como objetivo principal, pretende estandarizar el flujo de CI/CD del departamento para proyectos nuevos, estableciendo un stack tecnológico de referencia en los proyectos promovidos desde hiberus.

Por otro lado, pretende ser una guía de referencia para aquellas personas que se incorporan al departamento de DevOps dentro de la compañía y nunca han trabajado con esta metodología, con el objetivo de que adquieran una visión general de cómo estas prácticas otorgan beneficios fundamentales a las empresas que las adoptan.

La elaboración de este trabajo se ha dividido en tres fases bien diferenciadas. La primera de ellas se corresponde con una fase de investigación de las herramientas actuales y cómo estas satisfacen o no las necesidades propias de la empresa.

Una vez concluyó este paso, se realizó una segunda fase de implementación de una arquitectura DevOps de referencia en un entorno local con máquinas virtuales simulando un entorno de desarrollo con el stack tecnológico propuesto por mi director del TFG. El objetivo de esta implementación de la arquitectura es demostrar cómo se acelera el ciclo de vida del desarrollo de software, acortando notablemente los periodos de desarrollo y por tanto aumentando la frecuencia con la que se pone el nuevo software en manos del cliente. Para ello se ha desarrollado y desplegado una simple aplicación (Hola Mundo) en Java.

Por último, las herramientas estudiadas en esta segunda fase se llevan a un entorno de producción real, siendo esta la tercera fase, desplegando la misma aplicación en una infraestructura en el cloud de Azure, para observar cómo funciona el flujo de despliegue en un entorno real con el objetivo de estudiar su comportamiento y hacer pruebas de cara a estandarizar el flujo de CI/CD del departamento.

# Índice

# Capítulo 1

# Introducción

## 1.1. Contexto del trabajo

Este proyecto se ha realizado en la empresa Hiberus Tecnología, una consultora tecnológica con sede central en Zaragoza, que se especializa en la prestación de servicios de consultoría de negocio, desarrollo tecnológico, transformación digital y outsourcing. En la actualidad, cuenta con más de 3000 empleados en plantilla, y en el año 2023 facturaron más de 180M de euros, lo que la situa en el top 5 de las empresas del sector tecnológico de capital español. Además de su sede central en Zaragoza, hiberus cuenta con oficinas en territorio nacional como Barcelona, Madrid, Bilbao, Sevilla... así como con oficinas en Europa como Londres, Munich, Milán y en el resto del mundo como Buenos Aires, Medellín, Bogotá, Miami...

Dentro de la empresa, he trabajado en el área denominada Hiberus Sistemas que se dedica a ofrecer soluciones tecnológicas de calidad, proporcionando apoyo integral a diversas empresas y negocios para alcanzar sus metas financieras. Su enfoque abarca proyectos de infraestructura tecnológica, servicios en la nube y servicios gestionados, lo que les permite ofrecer una amplia gama de servicios en el sector TI.

Hiberus Sistemas cuenta con diversos departamentos, entre los cuales se encuentra el conocido como "DevOps", que es donde he desarrollado este TFG. Este es un departamento totalmente nuevo, creado el 1 de enero de este mismo año (2024). El motivo de la creación del mismo es una apuesta total por las soluciones cloud aplicando la filosofía DevOps (ver Anexo A) en sus soluciones. Antes de la creación de este departamento, se trabajaba más con soluciones on-premise utilizando tecnologías más tradicionales. La decisión de dar un paso hacia la innovación surgió como respuesta a la creciente demanda del mercado y la necesidad de adaptarse a un entorno tecnológico en constante evolución con la llegada de los servicios cloud. Esta transición hacia soluciones en la nube no solo permite a Hiberus Sistemas mantenerse a la vanguardia de la tecnología, sino también satisfacer de manera más efectiva las necesidades de sus

clientes, ofreciendo soluciones ágiles, flexibles y escalables.

Desde que el departamento se constituyó como departamento independiente, hace 6 meses, ha experimentado un crecimiento significativo, pasando de 10 a 40 personas hasta la fecha. Además, se ha incrementado la facturación total de la compañía entre un 10 y un 15 por ciento. Estos datos han generado una mayor satisfacción entre los clientes, quienes cada vez confían más en las soluciones ofrecidas por hiberus.

## 1.2. Contexto tecnológico

Antes de la creación del departamento DevOps, el proceso de desarrollo de nuevo software, bien para el cliente, bien como producto propio, era muy diferente a lo que se está realizando actualmente. A la hora de desarrollar nuevo software, se operaba de la siguiente manera:

- Por un lado, el equipo de desarrollo se encargaba de construir el software de las aplicaciones y de encargarse de que ese código funcionara a la perfección y sin errores. Después de días, semanas o tal vez meses, el equipo de desarrollo finalizaba el proceso de creación de código y pondría ese software en manos del equipo de operaciones (en este caso, este sería el equipo donde he realizado este proyecto).

- Por otro lado, el equipo de operaciones se encargaba de proveer y configurar la infraestructura necesaria (máquinas, firewalls, bases de datos, servidores, redes...) para ejecutar esas aplicaciones, además de monitorizar ese software para comprobar que todo funcionaba según lo previsto.

En teoría, se creía que era un proceso bien calculado y libre de errores, pero en la práctica resultó que no era así. Los ciclos de desarrollo de software eran largos, y cuando había algún fallo en producción suponía devolver la aplicación al equipo de desarrollo para verificar y corregir esos errores, cuando el error podía venir por la falta de alguna librería o dependencia, o por alguna diferencia en las versiones del sistema operativo, lo que prolongaba los tiempos de entrega de software al cliente.

Ahora que se ha explicado ligeramente el contexto tecnológico de este proyecto, voy a definir de manera breve qué es DevOps ya que aparecerá en numerosas ocasiones durante este documento para que el lector pueda comprender mejor lo que se está explicando.

El movimiento DevOps como tal, empezó a fraguarse entre el 2007 y el 2008 aproximadamente, cuando las comunidades de operaciones de TI y desarrollo de software se pronunciaron sobre lo que consideraban una disfunción gravísima del sector.

Se alzaron contra el modelo tradicional de desarrollo de software, que exigía que los que escribían el código se mantuvieran al margen, en términos de organización y operación, de los que implementaban y mantenían dicho código. El término DevOps se corresponde con la combinación de las palabras inglesas Development y Operations. DevOps es un marco de trabajo, una filosofía, un conjunto de prácticas que agrupan el desarrollo de software (Dev) y las operaciones de TI (Ops) cuyo objetivo es promover la colaboración y comunicación entre estos dos equipos para reducir el ciclo de vida de desarrollo y desplegar software de calidad de la forma más automatizada y productiva posible.

Es una filosofía reciente, que están adoptando muchas grandes empresas y que está ayudando enormemente en la transformación digital de los clientes de hiberus, con varios casos de éxito recientes en la compañía.

## 1.3. Motivación y problema que se aborda

Tal y como se ha explicado anteriormente, el equipo DevOps apenas lleva 6 meses operando como un equipo independiente y, debido a su reciente creación, tiene todavía un gran número de procesos por estandarizar y automatizar. Además, hiberus se encuentra en constante expansión de su plantilla (actualmente en Junio de 2024, se está incorporando una media de noventa a cien personas al mes a la compañía), por lo que resulta crucial estandarizar los procesos para asegurar que este crecimiento se desarrolle de la manera más organizada posible.

El objetivo último de estas estandarizaciones es conseguir que, cuando se realizan los flujos de despliegue de aplicaciones, se lleven a cabo, en la medida de lo posible, de manera sistemática en el uso de las mismas tecnologías y prácticas para cada cliente.

Además, no se cuenta con un on-boarding dentro del departamento para la gente que viene nueva, ya sea de prácticas o contratada. Por ello se pretende también que este proyecto sirva como una primera toma de contacto con estas herramientas y con el trabajo que desempeña este departamento.

Para ello, el trabajo que se ha realizado es la implementación de una arquitectura tecnológica de referencia para el despliegue de todo tipo de aplicaciones Java (la mayoría de aplicaciones desarrolladas por hiberus utilizan este lenguaje de programación) en dos entornos: un entorno de desarrollo empleando máquinas virtuales, y otro entorno de producción alojado en el cloud de Azure. Este último pretende imitar un entorno real que podría asemejarse a un entorno de cualquier cliente (staging, pre-producción o producción) que Hiberus implementa para desplegar las aplicaciones de los clientes en el cloud, pero utilizando un stack tecnológico que sirva como prueba de concepto para poder analizar estas herramientas y ver si satisfacen los requisitos de la compañía. Así

mismo, con la creación de estos dos entornos, se pueden configurar otras herramientas que se incorporen al flujo de desarrollo y despliegue para poder comprobar cómo se integran unas con otras y de esta forma tener una base para poder estandarizar el uso de estas herramientas teniendo dos entornos que se asemejan totalmente a las infraestructuras construidas para los clientes.

En estas dos fases, que se explican con profundidad en la sección 2, se detalla paso a paso cómo se ha configurado y contruido la arquitectura, explicando el uso y función de cada tecnología, y profundizando en el detalle de cómo funciona el flujo de CI/CD para este proyecto, que no deja de ser una demostración de "prueba"de cómo se opera de cara al cliente.

## 1.4. Herramientas de trabajo

Durante las fases del trabajo desarrollado que se explican en profundidad en el capítulo 2, se ha hecho uso de una gran variedad de herramientas y tecnologías:

- Como repositorio de código y sistema de control de versiones se ha trabajado con GitLab, para alojar los dos repositorios necesarios para la realización de este proyecto. Uno para alojar el repositorio de código fuente de la aplicación y otro para alojar el código de configuración de la aplicación, en lo que se conoce como repositorio GitOps (ver Anexo C).

- Para construir la primera infraestructura en un entorno local con máquinas virtuales se ha empleado la herramienta Vagrant[1].

- Como lenguaje de programación para desarrollar la aplicación de prueba se ha empleado Java[2].

- Como herramienta de gestión de dependencias y de construcción de código se ha empleado Maven[3].

- Se ha empleado Docker para construir una imagen portable e independiente de la aplicación desarrollada[4].

- Como servidor de automatización y operador de Continous Integration se ha utilizado la herramienta Jenkins[5].

[1]Documentación oficial Vagrant: `https://developer.hashicorp.com/vagrant/docs`
[2]"Que es Java": `https://www.java.com/en/download/help/whatis_java.html`
[3]Documentación oficial Maven: `https://maven.apache.org/what-is-maven.html`
[4]Documentación oficial Docker: `https://www.docker.com/`
[5]Documentación oficial Jenkins: `https://www.jenkins.io/doc/`

- Para analizar el código estático de la aplicación en busca de vulnerabilidades o malas prácticas y para conseguir lo que se conoce como "Clean code"se ha empleado la herramienta SonarQube[6].

- Trivy se ha empleado para analizar las vulnerabilidades de las diferentes capas de la imagen Docker de nuestra aplicación. Gracias a Trivy y SonarQube conseguimos aplicar una capa de seguridad en cada fase del ciclo de vida del software, adoptando prácticas que se conocen como "DevSecOps"(ver Anexo D).

- Como repositorio de imágenes de contenedor de Docker se ha empleado Nexus Repository Server [7] y Azure Container Registry [8]

- Para automatizar el despliegue de las aplicaciones en los diferentes entornos se utiliza la herramienta de Continous Deployment conocida como ArgoCD[9].

- Para construir los entornos donde se ha desplegado la aplicación, se ha empleado Kubernetes[10], en concreto Minikube[11] en la versión con máquinas locales y Azure Kubernetes Service (AKS)[12] en la última fase del proyecto.

Por último, como proveedor de cloud para construir toda la infraestructura en la última fase, se ha empleado Azure [13].

[6]Documentación SonarQube: `https://docs.sonarsource.com/sonarqube/latest/`
[7]Documentación Nexus: `https://www.sonatype.com/products/sonatype-nexus-repository`
[8]`https://azure.microsoft.com/es-es/products/container-registry`
[9]Documentación ArgoCD: `https://argo-cd.readthedocs.io/en/stable/`
[10]Kubernetes: `https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/`
[11]Minikube: `https://minikube.sigs.k8s.io/docs/start/`
[12]Azure Kubernetes Service: `https://learn.microsoft.com/en-us/azure/aks/what-is-aks`
[13]Azure: `https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure`

## 1.5. Esquema general de la memoria

En esta memoria se documenta el trabajo de fin de grado, el cual se divide en tres capítulos principales. A continuación, se describe de manera resumida el contenido de cada sección para ofrecer una visión general de la estructura del documento.

**El Capítulo 1, Introducción**: Se corresponde con la introducción y establece el contexto y la motivación del trabajo realizado. Aquí se presenta el ámbito del proyecto, su relevancia y la situación actual en el campo de estudio. Se describen las tecnologías y herramientas principales utilizadas, se expone la problemática específica que se pretende resolver y se detallan las herramientas empleadas durante el desarrollo del proyecto. Además, se proporciona esta visión general de la estructura del documento para ayudar al lector a entender la organización del contenido.

**Capítulo 2, Trabajo Desarrollado**: Detalla el trabajo realizado en tres fases distintas. La primera fase se centra en la investigación de la literatura y las herramientas relevantes para el proyecto. La segunda fase describe la implementación de una arquitectura DevOps de referencia utilizada en hiberus para la construcción de un entorno de desarrollo a modo de prueba, creado con máquinas virtuales Vagrant. La tercera fase se enfoca en la implementación de la misma arquitectura DevOps, simulando un entorno de producción en el cloud de Azure.

**Capítulo 3, Lecciones aprendidas y conclusiones**: Este capítulo recopila los conocimientos adquiridos y aspectos más complejos enfrentados durante el desarrollo del proyecto. Además, se discuten ideas futuras que podrían derivarse de este trabajo y se presentan las conclusiones finales.

# Capítulo 2

# Trabajo desarrollado

## 2.1. Fase 1: Investigación de la literatura y herramientas

Durante esta primera fase, que abarcó aproximadamente dos semanas, se llevaron a cabo las siguientes actividades y se procedió de la manera que se describe a continuación.

Inicialmente, me dediqué a comprender el funcionamiento del equipo del departamento, lo cual incluyó la lectura de documentación sobre su organización interna y la familiarización con diversos conceptos fundamentales. Entre estos conceptos se encuentran DevOps, administración de sistemas, redes (Networking) y computación en la nube (Cloud).

Después de haber obtenido una visión general de los conceptos mencionados anteriormente, mi director de TFG me proporcionó un primer boceto de arquitectura de referencia. Este diagrama representaba el stack tecnológico utilizado en las soluciones ofrecidas al cliente para agilizar, automatizar y estandarizar el ciclo de vida del desarrollo de software y los despliegues de sus aplicaciones.

El diagrama propuesto fue el que se muestra en la figura 2.1. A partir del mismo, mi primera tarea fue realmente entender el flujo de este diagrama y entender la función de cada tecnología. Tal y como se puede ver en la figura, los repositorios de código se encuentran alojados en GitLab y por cada aplicación del cliente que se quiere desplegar se cuenta con dos repositorios: El repositorio de código fuente y el repositorio de configuración, que es donde se almacenan todos los ficheros yaml que definen los objetos a desplegar en el clúster de Kubernetes. Jenkins es el servidor de automatización que se encarga de clonar ese repositorio de código fuente para su compilación, análisis estático del código con la herramienta SonarQube y la ejecución de los tests unitarios y test de integración. La herramienta Nexus Repository es un repositorio de artefactos que se emplea para almacenar las imágenes de contenedor de las diferentes versiones

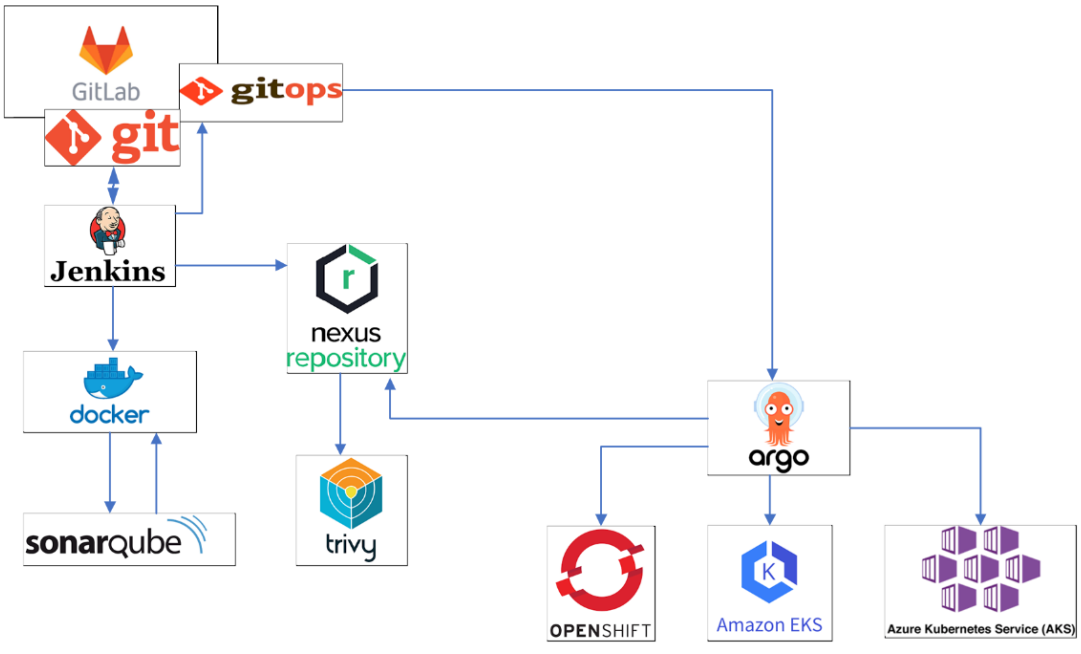



Figura 2.1: Arquitecura tecnológica de referencia

de las aplicaciones. La parte de Docker y Sonarqube refleja que en la mayoría de las instalaciones, los diferentes componentes que componen Sonarqube se ejecutan como contenedores de Docker. SonarQube es una herramienta que también se emplea de forma estandarizada en los flujos CI/CD de hiberus y se encarga de mantener la cálidad del código asegurando que se construye software de manera eficiente, segura y empleando buenas prácticas. Además, tambien analiza posibles vulnerabilidades y brechas de seguridad en el código. Trivy es la herramienta que se encarga de escanear vulnerabilidades dentro de las imágenes de contenedor de las aplicaciones. Por último, ArgoCD es el operador de Continous Deployment y se encarga de que el clúster siempre se mantenga en el mismo estado que reflejan los ficheros alojados en el repositorio de configuración en GitLab, encargándose de desplegar dichos cambios en el clúster.

Es importante resaltar que, para implementar buenas prácticas, los administradores de los clústers de Kubernetes deben aprovechar al máximo la infraestructura como código (IaC, ver Anexo 4) que ofrece ArgoCD, y siempre se debería cambiar la configuración del clúster mediante los ficheros del repositorio de Git, es decir, nos beneficiamos de la filosofía GitOps. Esto evita cambios ad-hoc en el clúster, reduciendo así la posibilidad de errores humanos, y mejorando la auditabilidad de los cambios, pudiendo hacer un sencillo rollback en caso de errores, restaurando el estado previo del sistema. Cuando ArgoCD detecta cambios en el repositorio de configuración, se encarga de desplegar esos cambios en el clúster correspondiente que puede estar alojado en Azure Kubernetes Service, OpenShift Kubernetes Engine, o Elastic Kubernetes

Service de Amazon. Como se puede observar, se intentan alojar esos clústers siempre en entornos de cloud, lo que aporta escalabilidad, agilidad, tolerancia a fallos y menor esfuerzo de mantenimiento, permitiendo responder de manera rápida a las necesidades del cliente.

Para poder afrontar las siguientes fases del proyecto, me dediqué a entender qué papel jugaba cada una de las tecnologías de esta arquitectura, gracias a leer documentación y reunirme con compañeros del departamento. Cabe destacar que las tecnologías de este diagrama no son un estándar, y uno de los objetivos de este TFG era probar los entornos construidos para ver cómo se integran las tecnologías entre ellas y proponer otras alternativas. Una vez se comprendidó el uso y posibles beneficios de estas herramientas construí un diagrama de flujo del despliegue automatizado para entender mejor la infraestructura a implementar y no perder de vista el conjunto de tareas que se querían automatizar. El diagrama presentado en la figura 2.2 me sirvió de gran ayuda y guía en las siguientes fases. Además, ha servido también de modelo en el departamento para el propósito de estandarizar el flujo de despliegue de las aplicaciones.

El proceso de despliegue creado en la figura 2.2 es continuo y se ejecuta cada vez que se aceptan nuevos cambios en el repositorio de código fuente, acortando los periodos de ciclo de vida del desarrollo del software, haciendo la vida mucho más fácil a los desarrolladores, permitiéndoles dedicarse exclusivamente a escribir código de calidad abstrayéndose de la infraestructura subyacente. Este flujo se explica a continuación:

1. Una vez el desarollador añade nuevas features o funcionalidades a la aplicación, este sube esos cambios al repositorio de código mediante una pull request. Cuando la PR se acepta, se desencadena el proceso automático de despliegue del software.

2. A continuación, se ejecuta el análisis estático del código con SonarQube y se compila el código fuente. Si, el análisis falla, hay que volver a revisar el código y corregir los errores. Si el análisis es exitoso, se continúa con la siguiente fase.

3. En esta fase se ejecutan los tests correspondientes escritos por los desarrolladores.

4. Si el resultado de la fase de testing es el esperado, se crea una imagen de Docker con las dependencias requeridas por la aplicación.

5. A continuación, se ejecuta un escaner de la imágen de Docker construida en la fase anterior en busca de vulnerabilidades. Este escáner lo realiza la herramienta Trivy.

6. Si la imagen es segura, se sube la imagen al repositorio de artefactos.

7. En este paso comienza el flujo de CD. Una vez se ha subido la imagen de la aplicación al repositorio de imágenes, Jenkins clona el repositorio Git de configuración (también llamado repositorio GitOps), modifica el fichero donde se especifica la versión de la imagen y se suben los cambios en el repositorio GitOps.

8. En el último paso, la herramienta ArgoCD detecta que ha habido cambios en el repositorio de GitLab y que no coincide el estado actual del clúster con el estado reflejado en los ficheros de configuración. Es en este paso, cuando Argo despliega los nuevos cambios en el clúster para asegurarse que el estado actual coincide con el deseado.

Los procesos que estan en azul más claro en la figura 2.2 se corresponden con el proceso de Continous Integration. Todos estos procesos los ejecuta el servidor de Jenkins.

Los que estan en azul más oscuro se corresponden con el Continous Deployment.

Como primera aproximación, el director de este TFG me pidó que implementara la arquitectura de la figura 2.1 con máquinas virtuales utilizando la herramienta "Vagrant". Esta fase se explica con detalle en la siguiente sección.

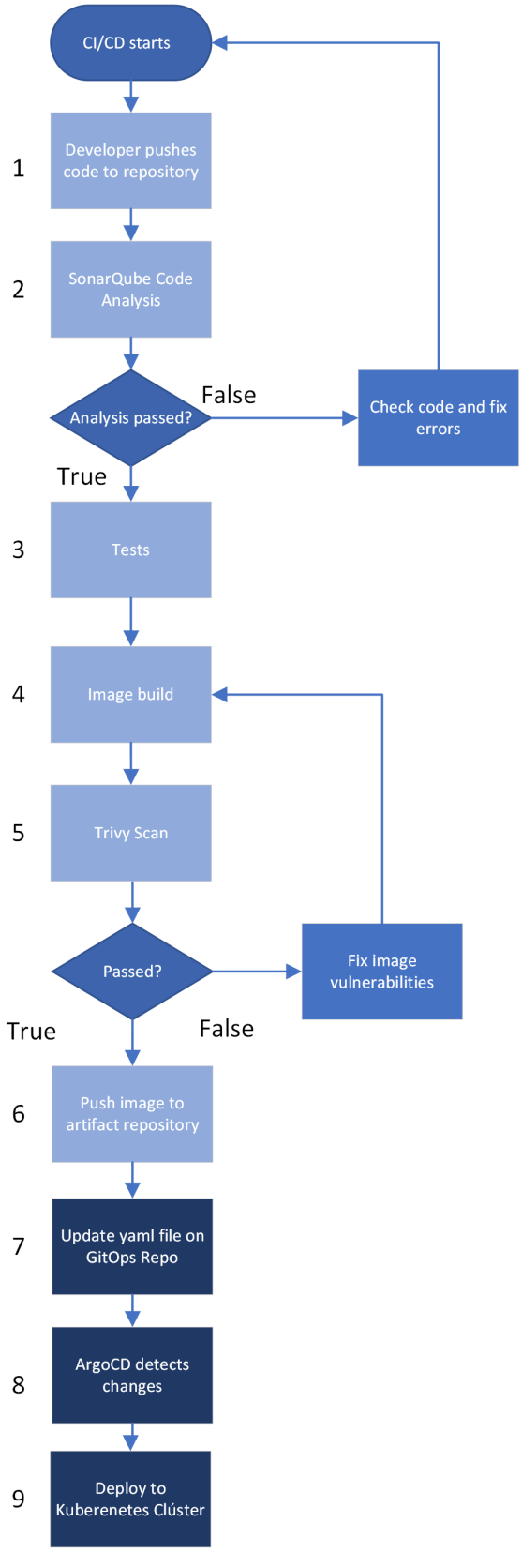



Figura 2.2: Flujo de despliegue estandarizado en hiberus

## 2.2. Fase 2: Implementación de la arquitectura DevOps de referencia empleada en hiberus para la construcción de un entorno de desarrollo a modo de prueba creado con máquinas virtuales Vagrant.

Este periodo duró aproximadamente un mes y fue sin duda la fase más intensa de trabajo y ha supuesto un reto para mí ya que nunca me había enfrentado a estas herramientas y tecnologías. El director de este TFG me había pedido construir una infraestructura con máquinas virtuales de Vagrant para implementar la arquitectura de referencia. Vagrant es una herramienta que utiliza VirtualBox (también puede trabajar con otros software de virtualización como VMware) y emplea su API para construir máquinas virtuales de manera declarativa, con ficheros de configuración.

El objetivo de esta fase era desarrollar una simple aplicación en Java que escribiera "hola mundo"por pantalla (esta decisión se toma porque la mayoría de aplicaciones que utilizan microservicios en hiberus se construyen en Java) y comprobar cómo una vez construida la infraestructura y los scripts de automatización, con hacer un cambio en el código fuente, se lanza automáticamente todo el proceso de CI/CD y se despliega la aplicación en el clúster de Kubernetes con esos cambios. Para ejecutar el proceso de despliegue de aplicaciones, primero fue necesario crear la infraestructura que se detalla a continuación.

Decidí que utilizar cuatro máquinas era la opción más óptima y eficiente:

- Una máquina como servidor de GitLab donde se alojan los repositorios (en hiberus tambien emplean un servidor dedicado para alojar los repositorios necesarios).
- Un servidor de Jenkins que se encarga de ejecutar todas las tareas del proceso de Continous Integration, donde además se ejecuta SonarQube y Trivy.
- Otra máquina para el repositorio de imágenes Nexus.
- La última máquina se ha empleado para alojar el clúster de Kubernetes, en este caso Minikube y la herramienta ArgoCD.

El diagrama con la infraestructura construida sería el que se muestra en la figura 2.3

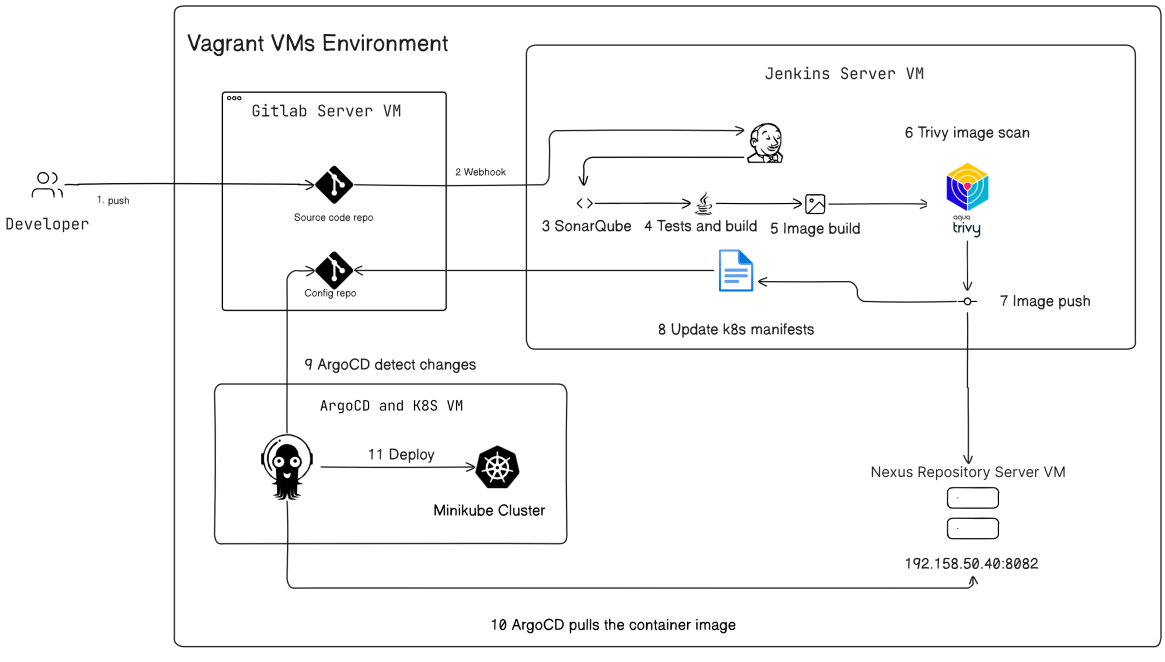



Figura 2.3: Entorno creado con máquinas virtuales Vagrant

Este proceso de construcción del entorno se ha automatizado con varios scripts gracias a las opciones de aprovisionamiento que ofrece Vagrant para que ejecutando un solo comando se levante la infraestructura (instalación de Jenkins, Docker, levantar SonarQube con Docker Compose, ArgoCD, Trivy en la máquina correspondiente y configuración de la red privada y reenvío de puertos).

Una vez finalizada la construcción automatizada del entorno se procedió a configurar la creación y comunicación de las tecnologías del proyecto en el siguiente orden:

- Creación de cuenta gitlab y configuración de los dos repositorios. Creación del código fuente y los tests de la aplicación.
- Creación del Dockerfile de la aplicación.
- Integración de Jenkins con GitLab para que una vez se suban los cambios al repositorio, se envie un webhook a Jenkins para que comience el proceso de CI.
- Integración de Jenkins y SonarQube para que este avise si el análisis del código ha sido exitoso.
- Integración de Nexus Repository con el pipeline de Jenkins.
- Escribir los ficheros de yaml de Kubernetes necesarios para crear el despliegue de los pods que corren la app y subirlos al repositorio de configuración.

- Configuración de la aplicación de ArgoCD para que escuche el repositorio de configuración.

- Por último, la creación del Jenkinsfile que es el fichero que contiene el script que se encarga de ejecutar la lógica del proceso de desliegue.

En este punto, teníamos creada toda la infraestructura necesaria para poder ejecutar el flujo de despliegue. Por último, faltaba escribir el script de automatización que ejecuta Jenkins y se encarga de realizar todos los pasos necesarios del proceso de despliegue. Este fichero, por convención se crea en el repositorio de código y contiene un script en el lenguaje ruby.

El script de Jenkins se presenta a continuación. Se divide en varios snippets de código en este documento por simplicidad, pero todos los fragmentos forman parte del mismo fichero denominado Jenkinsfile.


Figura 2.4: Primera parte del Jenkinsfile

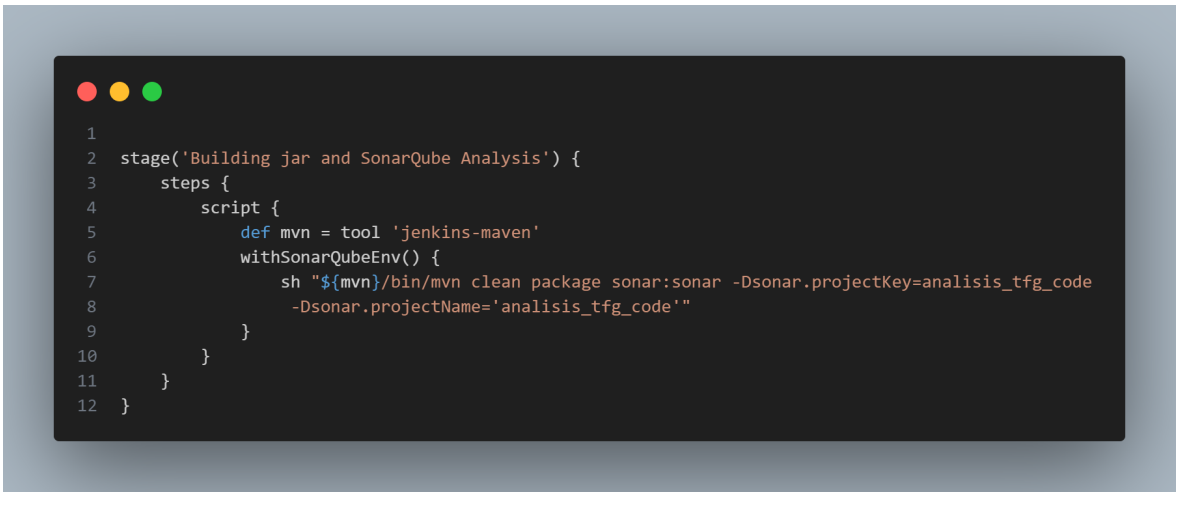



Figura 2.5: Compilación del código fuente, ejecución de tests y análisis con SonarQube

En la figura 2.4 y 2.5 se ven dos fases del script que recordemos se encarga de desplegar la aplicación en el entorno correspondiente. En la figura 2.4, Jenkins clona el repositorio de código que habíamos alojado en la primera máquina virtual. En nuestro caso es el repositorio de codigo fuente. En la figura 2.5 con un solo comando se compila el código fuente, se ejecutan los tests y se analiza el código estático con SonarQube.

En la figura 2.6 se ve el código necesario para que el pipeline espere a que la aplicación de SonarQube finalice el análisis del código de la aplicación y aborte el script en caso de que el análisis no sea exitoso.

```
stage ('Quality gate SonarQube Analysis') {
            steps {
                timeout(time: 2, unit: 'MINUTES') {
                    waitForQualityGate abortPipeline: true
                }
            }
        }
```

Figura 2.6: Quality Gate de SonarQube

En la figura 2.7 se construye la imagen de contenedor que contiene nuestra aplicación. Las variables en mayúscula son variables de entorno que se definen al principio del script.

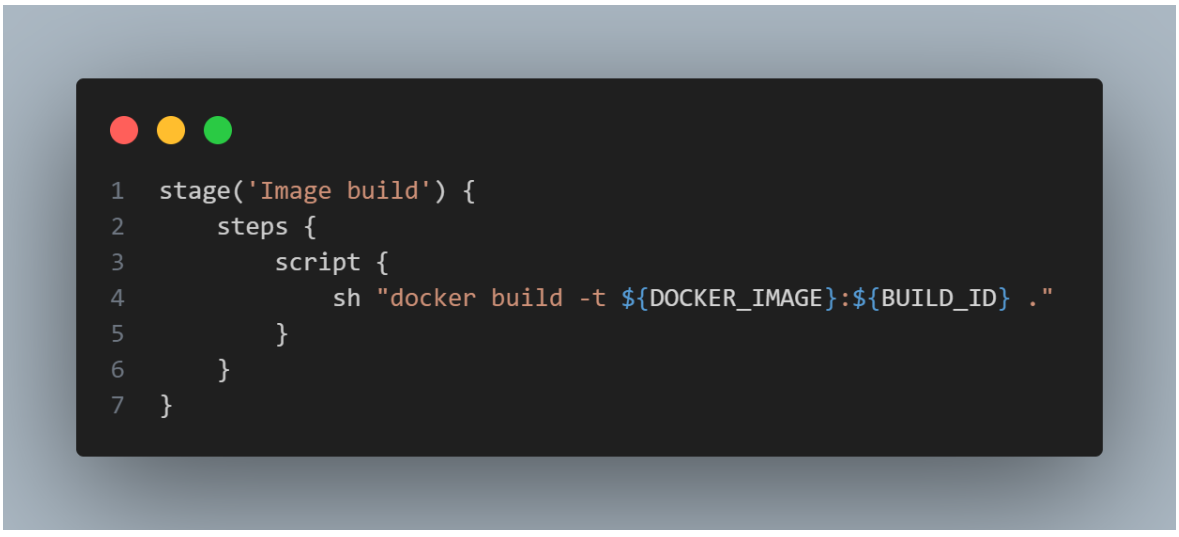
La variable BUILD-ID contiene el número de ejecución del pipeline, por tanto, si es la décima vez que se ejecuta, la variable contendrá el valor "10". Se utiliza esta convención para que el nombre de la imagen tenga un valor único que se corresponde con las distintas versiones de la aplicación.

En los entornos reales, se suele utilizar el hash del commit de Git que identifica esos cambios, seguidos de la fecha de construcción de la imagen.

En la figura 2.8 se realiza el análisis de las vulnerabilidades de la imagen construida. En este pipeline se analizan las vulnerabilidades que son altas y críticas, siendo bastante estrictos en la seguridad del entorno.

A continuación, en la figura 2.9 nos logeamos con el registro de Nexus que hemos desplegado, etiquetamos la imagen y hacemos el push de esa imagen al repositorio.

Por último, queda hacer un checkout del repositorio de configuración, modificar en



```
stage('Image build') {
    steps {
        script {
            sh "docker build -t ${DOCKER_IMAGE}:${BUILD_ID} ."
        }
    }
}
```

Figura 2.7: Construcción de la imagen de docker

```
stage('Trivy scan') {
    steps {
        script {
            sh "trivy image --no-progress --exit-code 1 --severity HIGH,CRITICAL ${DOCKER_IMAGE}:${BUILD_ID}"
        }
    }
}
```

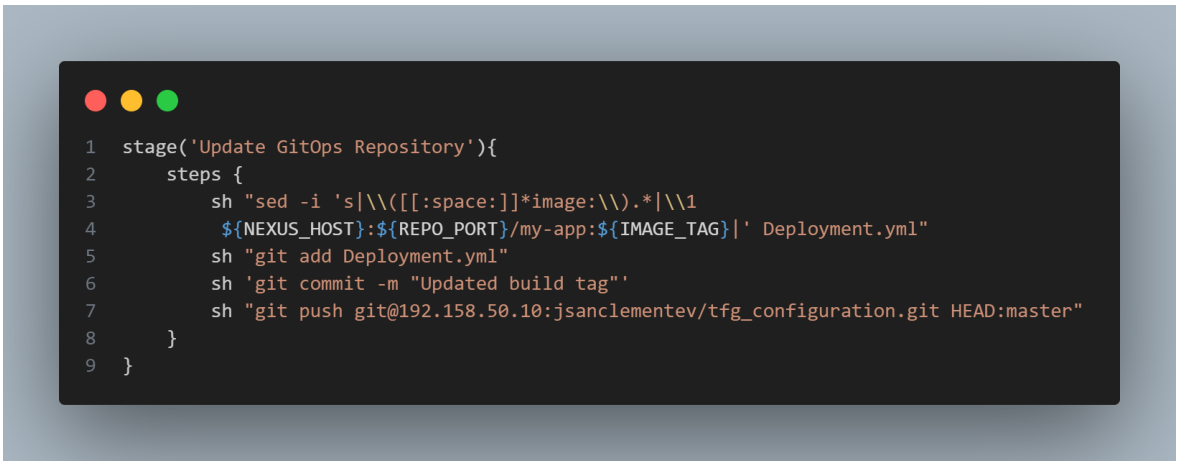
Figura 2.8: Escáner de la imagen con Trivy

el fichero del despliegue de Kubernetes el nuevo tag de la imagen y subir los cambios al repositorio. Todas estas acciones se ejecutan en la Figura 2.10

Es a partir de este paso, cuando de manera automatizada, ArgoCD detecta los cambios que ha habido en este repositorio y re-despliega la nueva versión de la aplicación en el clúster sin la necesidad de intervención humana.

```
stage('Push to Nexus') {
    environment {
      NEXUS_CREDS = credentials('nexus-credentials')
    }
    steps {
        script {
            sh 'echo $NEXUS_CREDS_PSW | docker login -u $NEXUS_CREDS_USR --password-stdin  $REPO_HOST:$REPO_PORT'
            sh "docker tag ${DOCKER_IMAGE}:${BUILD_ID} ${REPO_HOST}:${REPO_PORT}/${DOCKER_IMAGE}:${BUILD_ID}"
            sh "docker push ${REPO_HOST}:${REPO_PORT}/${DOCKER_IMAGE}:${BUILD_ID}"
        }
    }
}
```

Figura 2.9: Push de la imagen al repositorio



```
stage('Update GitOps Repository'){
    steps {
        sh "sed -i 's|\\([[:space:]]*image:\\).*|\\1
         ${NEXUS_HOST}:${REPO_PORT}/my-app:${IMAGE_TAG}|' Deployment.yml"
        sh "git add Deployment.yml"
        sh 'git commit -m "Updated build tag"'
        sh "git push git@192.158.50.10:jsanclementev/tfg_configuration.git HEAD:master"
    }
}
```

Figura 2.10: Modificación del repositorio GitOps

**Demostración de caso de uso de prueba**

Para comprobar que toda la infraestructura construida despliega las aplicaciones correctamente, debemos hacer cambios en el repositorio de código fuente. Por ejemplo, cambiaremos el mensaje que escribe la aplicación Java por pantalla:

- En este punto, nuestra aplicación escribe "hola mundo"por pantalla. Cambiaremos el mensaje a Hola hiberus! y subimos los cambios al repositorio.

App.java 306 Bytes

```java
package com.mycompany.app;

/**
 * Hello world!
 */
public class App {

    private static final String MESSAGE = "Hola mundo!";

    public App() {}

    public static void main(String[] args) {
        System.out.println(MESSAGE);
    }

    public String getMessage() {
        return MESSAGE;
    }
}
```

Figura 2.11: Fichero App.java de la apliación

- Cuando se suben los cambios al repositorio, se ejecuta el pipeline de Jenkins automáticamente, gracias al webhook enviado por GitLab. El script de Jenkins ejecuta todos los stages explicados anteriormente que coinciden además con las fases explicadas en el diagrama de flujo de la Figura 2.2, estos stages aparecen en verde indicando que se han ejecutado correctamente. Si hubiera un error en el código, o vulnerabilidades en las imágenes de contenedor construidas, el pipeline aparecería en rojo en ese punto y se detendría su ejecución, indicando en los logs el motivo del error. Cabe destacar que no sólo se implementa un flujo de despliegue automatizado si no que además se añaden capas adicionales de seguridad con SonarQube y Trivy para que las aplicaciones sean lo más seguras posibles.

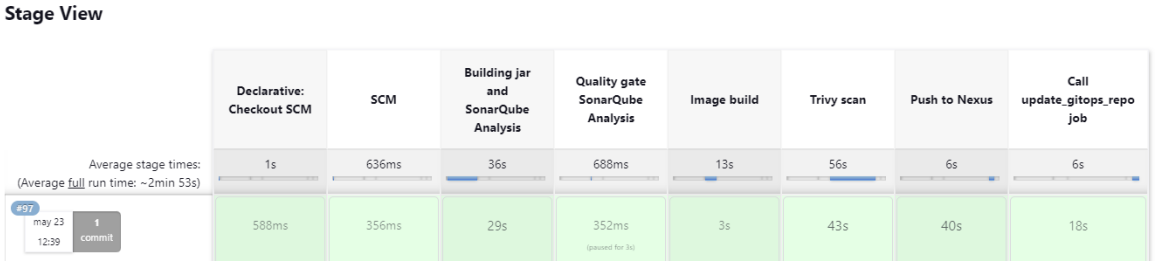



Figura 2.12: Información en tiempo real de los stages del pipeline en Jenkins

- En una media de tiempo de 2 minutos y medio, si no hay errores, los cambios hechos en el código se reflejan en el clúster. Para comprobar esto, en la máquina

virtual en la que se aloja el clúster de Minikube podemos observar que hay tres pods corriendo nuestra aplicación (ver Figura 2.13).

```
vagrant@argocd:~$ k get po
NAME                      READY   STATUS    RESTARTS   AGE
my-app-765b786759-lvqz2   1/1     Running   0          2m35s
my-app-765b786759-pf6tt   1/1     Running   0          2m48s
my-app-765b786759-x5mdw   1/1     Running   0          4m36s
```

Figura 2.13: Información de los pods que están ejecutando nuestra aplicación

- Para ver si se ha desplegado la nueva versión, es decir, la versión en la que nuestra aplicación escribe hola hiberus por pantalla, podemos obtener los logs de los pods. Como podemos observar en la figura 2.14, el flujo de despliegue funciona según lo previsto. Se han aplicado los cambios que hemos hecho en el repositorio de código fuente de manera totalmente automatizada.

```
vagrant@argocd:~$ k logs my-app-765b786759-lvqz2
Hola hiberus!
```

Figura 2.14: Logs de los pods

## 2.3. Fase 3: Implementación de la arquitectura DevOps de referencia empleada en hiberus simulando un entorno de producción en la nube de Azure.

Una vez terminada la fase anterior, el director del tfg me propuso implementar la misma infraestructura pero en la nube de Azure.

Esta fase se realizó en aproximadamente dos semanas y fue el último periodo de trabajo técnico de este TFG. Para ello se me proporcionó una cuenta en el directorio de Azure del departamento para poder alojar los recursos necesarios.

Antes de ponerme a implementar la arquitectura, tuve que familiarizarme con los conceptos básicos del cloud para que los gastos no estuvieran por encima de lo previsto. Entender los conceptos básicos de azure como región, zona de disponibilidad, red virtual, máquinas virtuales, interfaces de red virtuales, security groups de las máquinas virtuales, etcétera.

Una vez entendidos estos conceptos, al igual que en el Capítulo 2 se diseñó un diagrama de la infraestructura a construir.

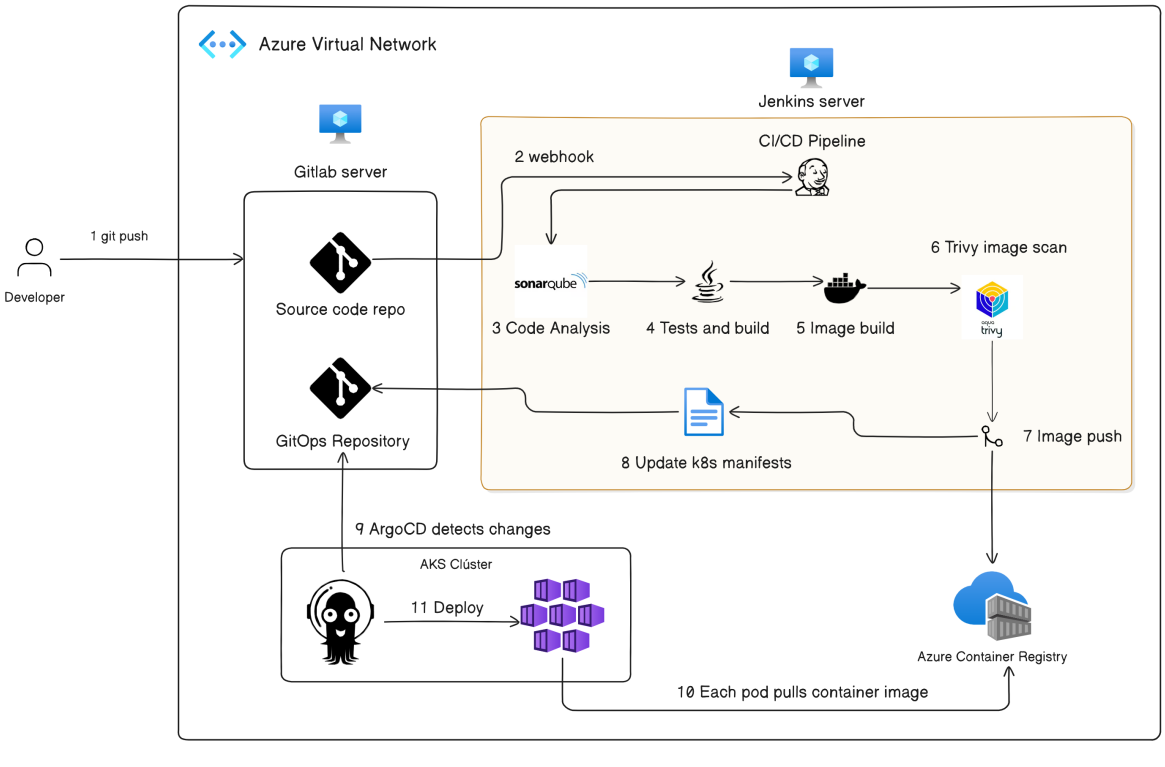



Figura 2.15: Arquitectura construida en Azure

Esta infraestuctura realmente no deja de ser una analogía de la arquitectura construida en la fase anterior, pero, al tratarse de un entorno de cloud, cambian algunos

componentes con respecto a la infraestructura de máquinas virtuales de la fase 2.

A continuación se expican los distintos elementos que componen esta arquitectura:

- Una máquina virtual de Azure como servidor de GitLab para alojar los repositorios necesarios.
- Una máquina virtual de Azure como servidor de Jenkins que se encarga de ejecutar todas las tareas del proceso de Continous Integration, donde además se ejecuta SonarQube y Trivy.
- El servicio de Azure Container Registry para el repositorio de imágenes de Docker.
- En vez de una máquina virtual donde instalar Minukube, se crea el servicio gestionado Azure Kubernetes Service para ejecutar la aplicación desarrollada y donde además se ejecuta la aplicación de ArgoCD.

Esta infraestructura no se construye de manera automatizada como era el caso de Vagrant, si no que este aspecto se comentará brevemente en la sección 3.3 de ideas futuras. En cambio, la instalación de herramientas como por ejemplo GitLab no fue necesaria ya que Azure proporciona imágenes de máquina virtual que ya contienen GitLab instalado. A continuación, una vez se finalizó la instalación de todos los paquetes necesarios (SonarQube, Docker, Trivy...) en cada máquina virtual, tuve que configurar los credenciales del registro de imágenes y configurar el servicio de Kubernetes de Azure donde había que indicar cuantos nodos crear en el clúster, la imagen de máquina virtual que iba a utilizar cada nodo (Linux, Debian...) y otras configuraciones del clúster.

Una vez instalados los paquetes, al igual que en la fase anterior habia que configurar la comunicación entre herramientas. Este paso simplemente fue replicar lo que ya había hecho en la fase 2 de este proyecto. Tenía todo documentado y simplemente fue seguir los pasos que ya había seguido. En cuanto al script que se encargaba de ejecutar todos los pasos del proceso de despliegue (Jenkinsfile), utilicé de plantilla el mismo que se explica en la fase anterior pero empleando los endpoints de Azure Container Registry en vez de emplear el repositorio de imágenes Nexus.

Esta arquitectura implementada ha servido como prueba de concepto en el departamento para tener una infraestructura en la que poder realizar pruebas con los compañeros para presentar una propuesta firme sobre la estandarización del flujo de CI/CD y presentarla a los responsables del departamento. (Ver Sección 3.3).

# Capítulo 3

# Lecciones aprendidas y conclusiones

## 3.1. Aspectos más complejos abordados

Mencionar aquí que nunca había trabajado con la parte de sistemas, exceptuando la asignatura de Administración de Sistemas en segundo de carrera. Además, no conocía prácticas y filosofías nuevas en el desarrollo de software como DevOps, CI/CD ni su propósito.

Conocía pocas herramientas tecnológicas utilizadas. Había trabajado con Docker en la asignatura de Ingeniería Web y durante la asignatura Cloud Computing en el Erasmus, pero de manera básica. Además, nunca había trabajado con Kubernetes, Vagrant, ArgoCD, SonarQube o Trivy. He de decir que la curva de aprendizaje fue muy alta en las primeras semanas del proyecto.

Cuando ya llevaba unas cuantas semanas trabajando en en el proyecto y había finalizado la fase 2, tuve que enfrentarme al aprendizaje y configuración de entornos en proveedores de servicios en la nube, en este caso Azure. Nunca había trabajado con proveedores de servicios en la nube como (AWS, Azure o Google Cloud).

Por último, me gustaría destacar el volumen de trabajo autónomo que he tenido que desarrollar. Como ya he mencionado, este TFG pretende ser el primer paso para una aproximación novedosa dentro de la empresa, lo que ha hecho que no haya podido contar con conocimientos ya asentados en la misma.

## 3.2. Conocimientos adquiridos

Durante el desarrollo de este proyecto, he aprendido varias habilidades o conceptos que me pueden servir de mucho para mi desarrollo profesional:

- **Cómo funciona un departamento de una consultora tecnológica**: Trabajar en este proyecto me ha permitido entender mejor el funcionamiento de un departamento en una empresa como hiberus. He visto cómo se gestionan los

proyectos, se coordinan los equipos y se implementan las soluciones tecnológicas para los clientes.

- **Ser más productivo**: He aprendido a ser más productivo al gestionar mejor mi tiempo y optimizar mis tareas. La necesidad de cumplir con los plazos fijados por mi director del proyecto para cada una de las tres fases, me ha obligado a desarrollar técnicas para mantenerme enfocado y eficiente en mi trabajo diario.
- **Mejorar mi capacidad de organización y planificación**: Este proyecto me ha enseñado la importancia de la organización y la planificación detallada. He mejorado significativamente en la estructuración de mis tareas y en la elaboración de planes que aseguren el cumplimiento de los objetivos establecidos.
- **La importancia de agilizar la frecuencia con la que se pone software en manos del cliente y responder rápidamente a sus necesidades**: Trabajar en una consultora tecnológica que ofrece servicios y soporte a otras compañías me ha enseñado la importancia de entregar software de manera rápida y eficiente. Responder rápidamente a las necesidades del cliente es esencial para mantener su satisfacción y confianza en los servicios ofrecidos.
- **Mejorar mis capacidades de comunicación**: Al tener que interactuar con distintos compañeros para pedir accesos a cuentas y otros recursos, he mejorado mis habilidades de comunicación. Este aspecto ha sido crucial para garantizar que todos los elementos necesarios para el proyecto estuvieran disponibles y en orden.

Estas lecciones no solo han mejorado mis habilidades técnicas y profesionales, sino que también me han proporcionado una perspectiva más amplia sobre la gestión de proyectos y la importancia de la colaboración y la comunicación efectiva en un entorno profesional.

## 3.3. Ideas futuras

En esta sección se va a comentar brevemente cuál va a ser el devenir de este proyecto en el futuro y como va a ayudar en el departamento de DevOps.

Como se mencionó en el resumen de este trabajo, el objetivo principal de este TFG era poder estandarizar el flujo de CI/CD del departamento.

El objetivo a medio-largo plazo de este proyecto, es poder hacer pruebas de despliegues con la infraestructura creada en Azure durante la fase 3 y poder presentarlo a los responsables del departamento como prueba de concepto. Una vez aprobada

la propuesta de las herramientas presentadas, la idea es poder crear un proyecto con los compañeros del departamento para poder automatizar la construcción de la infraestructura con Terraform. Terraform es una herramienta de infraestructura como código (ver Anexo 4) que permite definir la infraestructura de un sistema a través de código, lo que hace posible versionar, reutilizar y compartir configuraciones, de manera similar al código de software. Además, Terraform es compatible con múltiples proveedores de nube y servicios, incluyendo AWS, Azure, Google Cloud Platform que son los proveedores de cloud con los que trabaja hiberus.

Automatizar la construcción de la infraestructura que soporta el despliegue de software utilizando Terraform [1] y la filosofía DevOps sería extremadamente beneficioso para el departamento y para la compañía. Esta automatización agilizaría el tiempo necesario para replicar la arquitectura para cada cliente, permitiendo parametrizar el código según la propuesta específica de cada uno. Como resultado, el tiempo desde que un cliente contrata el servicio hasta que recibe una primera versión de la infraestructura se reduciría al mínimo, lo que aumentaría la satisfacción del cliente y, en consecuencia, traería beneficios significativos para la compañía.

[1] Página oficial Terraform: `https://www.terraform.io/`

## 3.4. Conclusiones

Durante estos años en la carrera, he aprendido gran cantidad de aspectos técnicas sobre redes, administración de sistemas, algoritmia y programación, gestión de proyectos software, sistemas operativos, y más. Pero, si soy sincero, lo más valioso que me llevo de la universidad es la forma en la que nos han enseñado a pensar y a trabajar.

**Trabajo en equipo:** He trabajado en muchos proyectos con mis compañeros en la universidad y ahí aprendí la importancia de una comunicación efectiva, repartir las tareas de manera justa y resolver problemas de manera conjunta. Esta habilidad no solo es clave en la universidad, sino también en el trabajo, donde la colaboración y el trabajo en equipo son esenciales para sacar adelante cualquier proyecto.

**Capacidad de análisis:** Durante mi formación, me he enfrentado a problemas complejos y he aprendido a analizarlos y encontrar soluciones efectivas. Descomponer problemas grandes en partes más pequeñas y manejables, evaluar diferentes enfoques y elegir el mejor es algo que he practicado mucho. Esta habilidad me ha sido de gran ayuda en la elaboración de este trabajo de fin de grado.

**Capacidad de comunicación:** Comunicar ideas y soluciones de manera clara y efectiva es otra habilidad que he desarrollado. Esto es crucial no solo en presentaciones académicas, sino también en el día a día laboral.

Además, la informática es un campo en el que siempre debes seguir formándote y aprendiendo cosas nuevas. Los conocimientos técnicos que adquirí en la universidad son una buena base, pero lo más importante es haber desarrollado la capacidad de seguir aprendiendo y la predisposición al cambio y a nuevas metodologías.

Después de terminar el TFG, he seguido trabajando en hiberus, formando parte de equipos de trabajo en proyectos para el cliente. Gracias a lo que aprendí en la universidad, me siento preparado para enfrentar las tareas diarias y confío en poder llevarlas a cabo con éxito. Todo lo que se me ha enseñado durante estos años de universidad me ha dado una muy buena base para comenzar mi carrera profesional y afrontarla con confianza en mis capacidades y conocimientos.

# Anexos

# Anexos A

# ¿Qué es DevOps?

El término DevOps se corresponde con la combinación de las palabras inglesas Development y Operations. DevOps es un marco de trabajo, una filosofía, un conjunto de prácticas que agrupan el desarrollo de software (Dev) y las operaciones de TI (Ops) cuyo objetivo es promover la colaboración y comunicación entre estos dos equipos para reducir el ciclo de vida de desarrollo y desplegar software de calidad de la forma más automatizada y productiva posible.

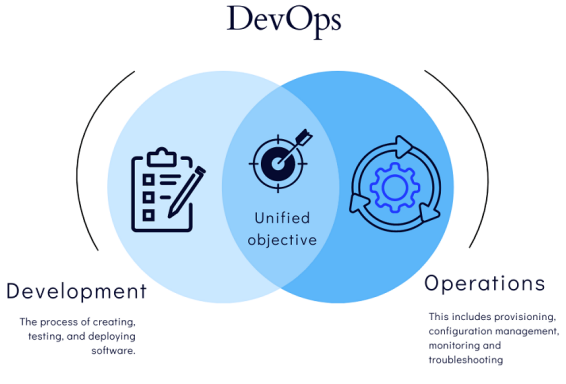



Figura A.1: La convergencia de estos dos equipos es el propósito central de DevOps.[1]

La principal característica del movimiento DevOps es defender activamente la automatización y el monitoreo en todos los pasos de la construcción del software, desde la integración, las pruebas, el despliegue, hasta la implementación y la administración de la infraestructura. DevOps apunta a ciclos de desarrollo más cortos, mayor frecuencia de implementación, lanzamientos más efectivos, en estrecha alineación con los objetivos comerciales.

[1]Imagen tomada de: `https://www.manageengine.com/latam/applications_manager/tech-topics/que-es-devops.html` (última visita: Junio 2024)

# Anexos B

# Principios y prácticas claves en DevOps

En esta sección se analizarán y estudiaran las prácticas claves que se realizan en la filosofía DevOps y que cualquier empresa que quiera adpotar esta filosofía o que ya la haya adoptado pero quiera aprovecharse al máximo de sus ventajas deberían incorporar en sus flujos de trabajo.

Cómo ya se ha ido comentando a lo largo del texto, DevOps parte de la idea de la automatización y la mejora continua del producto que se está desarrollando. Las prácticas por las que aboga la filosofía DevOps son las siguientes:

1. **Continous Development**

   Esta práctica se centra en la idea de que el código debe ser desarrollado en pequeñas y frecuentes iteraciones en lugar de hacerlo todo de una sola vez. El desarrollo continuo es esencial en DevOps porque optimiza la eficiencia cada vez que se crea, prueba, construye y despliega un fragmento de código en producción. Este enfoque de desarrollo continuo mejora la calidad del código y acelera la identificación y corrección de errores, vulnerabilidades y defectos. Además, permite que los desarrolladores se enfoquen en producir código de alta calidad.

2. **Continous Integration / CI**

   Esta es una de las prácticas más conocidas y mundialmente aceptadas por toda la comunidad del sector tecnológico. La integración continua sostiene que los desarrolladores deben fusionar con regularidad los cambios en el código en un repositorio central, tras lo cual se ejecutan compilaciones y pruebas automatizadas. Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidez del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones

[1]Imagen tomada de: `https://www.geeksforgeeks.org/devops-lifecycle/` (última visita: Junio 2024)

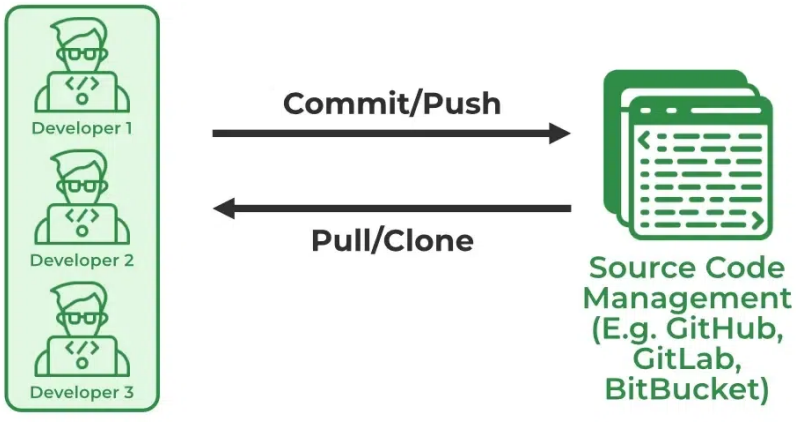



Figura B.1: El código se publica en un repositorio de manera frecuente y continua[1]

Anteriormente, era común que los desarrolladores de un equipo trabajasen aislados durante un largo periodo de tiempo y solo intentasen combinar los cambios en la versión de producción una vez que habían acabado el trabajo.

Como consecuencia, la combinación de los cambios en el código resultaba difícil y ardua, además de dar lugar a la acumulación de errores durante mucho tiempo que no se corregían. Estos factores hacían que resultase más difícil proporcionar las actualizaciones a los clientes con rapidez.

Con la llegada de la integración continua (CI), los desarrolladores envían los cambios de forma periódica a un repositorio compartido utilizando un sistema de control de versiones como Git. Un servidor de integración continua como Jenkins, compila el código y ejecuta los test de manera automática una vez se detectan cambios en ese repositorio compartido, lo que permite identificar y corregir inmediatamente cualquier error.

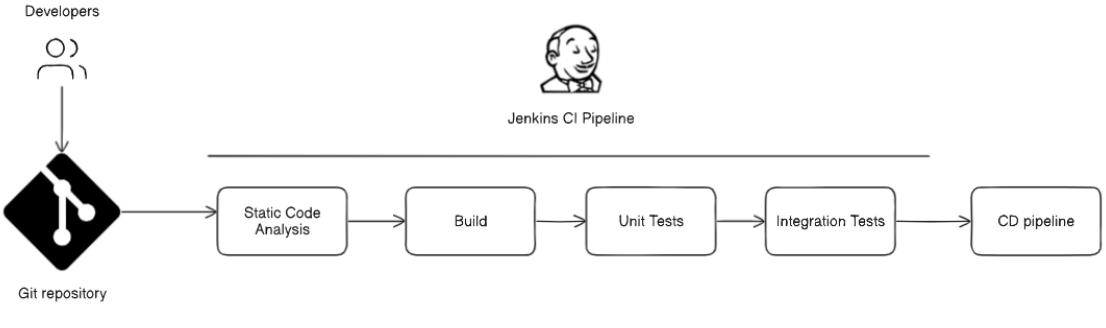



Figura B.2: Flujo de trabajo típico de Continous Integration con Jenkins

La integración continua mejora notablemente la productividad en el desarrollo de software, permite encontrar y arreglar los errores con mayor rapidez y permite entregar el software al cliente con mayor rapidez.

3. **Continous Delivery / Continous Deployment** Estos dos conceptos, frecuentemente referidos como "CD", son comúnmente confundidos en la industria debido a su notable similitud.

   A continuación, detallaré la diferencia principal entre ellos y la importancia de integrar estas prácticas en nuestro proceso de desarrollo de software bajo la filosofía DevOps.

   Por un lado, Continous Delivery es el proceso de desplegar una aplicación en producción manualmente, cuando esta ha completado el proceso de build y test. Con esta práctica se consigue automatizar el proceso un paso más allá de CI, sin embargo, se requiere la acción manual de un ingeniero para hacer las últimas comprobaciones antes de desplegar en producción. Por otro lado, Continous

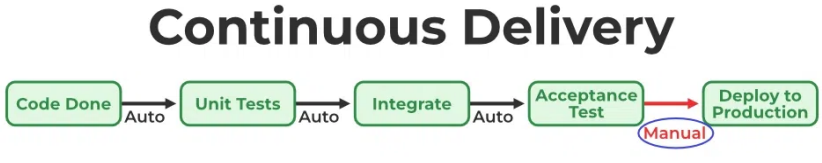



Figura B.3: El despliegue en producción requiere la acción manual de un profesional [2]

   Deployment va aún un paso más allá que Continous Delivery. Es el proceso de automáticamente desplegar una aplicación en algún entorno de desarrollo, staging, pre-producción o producción cuando se han completado las fases de build y test. En este caso se automatiza absolutamente todo el proceso (ver figura B.4), desde la obtención del código fuente hasta el despliegue de la aplicación en algún entorno. No se requiere ninguna acción manual en el proceso. La herramienta más utilizada para CD y que es la que más se utiliza en hiberus es ArgoCD.

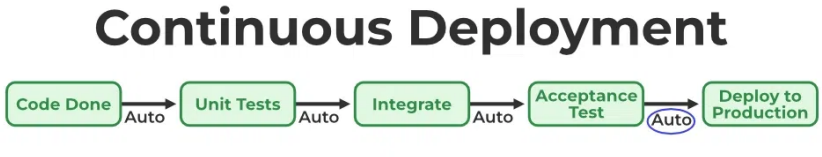



Figura B.4: El despliegue en producción se realiza de forma automática [3]

   En la práctica, en hiberus emplean el Continous Delivery para desplegar una aplicación en producción debido a que es un entorno crítico y hay que asegurar

[2]Imagen tomada de: `https://www.geeksforgeeks.org/devops-lifecycle/` (última visita: Junio 2024

[3]Imagen tomada de: `https://www.geeksforgeeks.org/devops-lifecycle/` (última visita: Junio 2024

que no haya fallos y todo funcione a la perfección. El Continous Deployment es muy útil para levantar entornos de desarrollo o pre-producción en los que si que conviene que el proceso esté lo más automatizado posible para poder verificar que la aplicación desplegada funciona según lo esperado.

4. **Infraestructura como código (IaC)**

   La infraestructura como código es el proceso de gestionar y provisionar infraestructura tecnológica a través de código en vez de realizarla en un proceso manual. Con IaC, se crean archivos de configuración que contienen las especificaciones de la infraestructura tecnológica, lo que facilita la edición y distribución de las configuraciones. También garantiza que siempre se aprovisione el mismo entorno. Al codificar y documentar sus especificaciones de configuración, IaC facilita la gestión de la configuración y ayuda a evitar cambios de configuración no documentados y ad hoc.

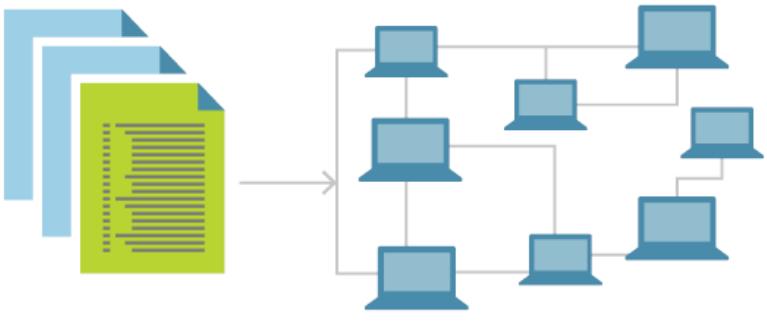


Figura B.5: Se crea la infraestructura necesaria a partir del IaC Fuente.[4]

[4]Imagen tomada de: `https://learn.microsoft.com/es-es/devops/deliver/what-is-in` (última visita: Junio 2024)

# Anexos C

# ¿Qué es GitOps?

GitOps es una metodología que utiliza los flujos de trabajo de Git para administrar la infraestructura y la configuración de las aplicaciones.

Al emplear los repositorios de Git como única fuente de verdad, GitOps permite a los equipos DevOps almacenar y gestionar el estado completo de la configuración de la infraestructura en Git. Esto garantiza que cualquier cambio en la infraestructura sea transparente y verificable.

GitOps radica en su capacidad para garantizar la consistencia, la trazabilidad y la automatización en la gestión de la infraestructura y las aplicaciones, permitiendo a los equipos de desarrollo y operaciones mantener un control preciso de los recursos, realizar despliegues confiables y fomentar la colaboración al utilizar repositorios de Git como fuente única de verdad.

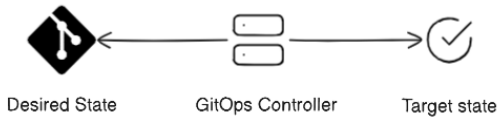



Figura C.1: El controlador gitops lleva la aplicación al estado deseado

El proceso de despliegue de una aplicación utilizando GitOps se describe a continuación:

Como Ingeniero DevOps o desarrollador, es necesario definir la infraestructura de la aplicación a desplegar, como la creación de los manifiestos de Kubernetes requeridos. Posteriormente, se deben subir estos cambios al repositorio de Git designado, comúnmente conocido como “Repositorio de Configuración”.

Una vez que los cambios se han subido al repositorio, un operador de GitOps se encarga de monitorear dicho repositorio de configuración. Detecta cualquier cambio realizado y, al identificar una modificación, procede a implementar automáticamente la infraestructura necesaria según la configuración reflejada en los archivos del repositorio.

# Anexos D

# ¿Qué es DevSecOps?

DevSecOps es la práctica de integrar las pruebas de seguridad en cada etapa del proceso de desarrollo de software. Incluye herramientas y procesos que fomentan la colaboración entre los desarrolladores, los especialistas en seguridad y los equipos de operaciones para crear un software que sea eficiente y seguro. DevSecOps aporta una transformación cultural que hace de la seguridad una responsabilidad compartida para todos los que crean el software. [1]

**¿Qué significa DevSecOps?**

DevSecOps significa desarrollo, seguridad y operaciones. Es una extensión de la práctica de DevOps. Cada término define diferentes funciones y responsabilidades de los equipos de software a la hora de crear aplicaciones de software.

**Desarrollo**

El desarrollo es el proceso de planificación, codificación, creación y prueba de la aplicación.

**Seguridad** La seguridad significa introducir la seguridad en una etapa temprana del ciclo de desarrollo de software. Por ejemplo, los programadores se aseguran de que el código esté libre de vulnerabilidades de seguridad y los profesionales de la seguridad prueban el software más a fondo antes de que la empresa lo publique.

**Operaciones** El equipo de operaciones publica, supervisa y corrige cualquier problema que surja del software.

[1] `https://aws.amazon.com/es/what-is/devsecops/#:~:text=DevSecOps%20es%20la%20pr%C3%A1ctica%20de,que%20sea%20eficiente%20y%20seguro.`