
Universidad
Tecmilenio®

Universidad Tecmilenio

Empaquetado en DevOps

Introducción a la plataforma de empaquetado Helm

Semana 11

Introducción

Helm ofrece un marco de referencia para desplegar una aplicación en Kubernetes, logrando ejecutar un montaje sin tener que empezar desde cero, ya que combina soluciones y procedimientos que previamente han sido compartidos a nivel comunidad.

Es de uso libre y esto ayuda a que pueda evolucionar por sí mismo gracias a los conocimientos compartidos que añaden funcionalidades a lo ya existente.

Explicación

Metodología DevOps

DevOps es un acrónimo en inglés de *development* (desarrollo) y *operations* (operaciones), que se refiere a un movimiento cultural y profesional enfocado en la comunicación, colaboración e integración entre profesionales que desarrollan el software y los profesionales en las operaciones de tecnologías de la información (López, 2021).

Según Freeman (2019), la metodología DevOps se centra en los siguientes principios:

- Pensamiento sistémico.
- Aceptar el fracaso.
- Comunicación y más comunicación.
- Disminuir el trabajo aislado.
- Aceptar la retroalimentación.
- Fomentar el trabajo en equipo.
- Automatizar procesos.

¿Qué es Helm y para qué sirve?

Universidad Tecmilenio

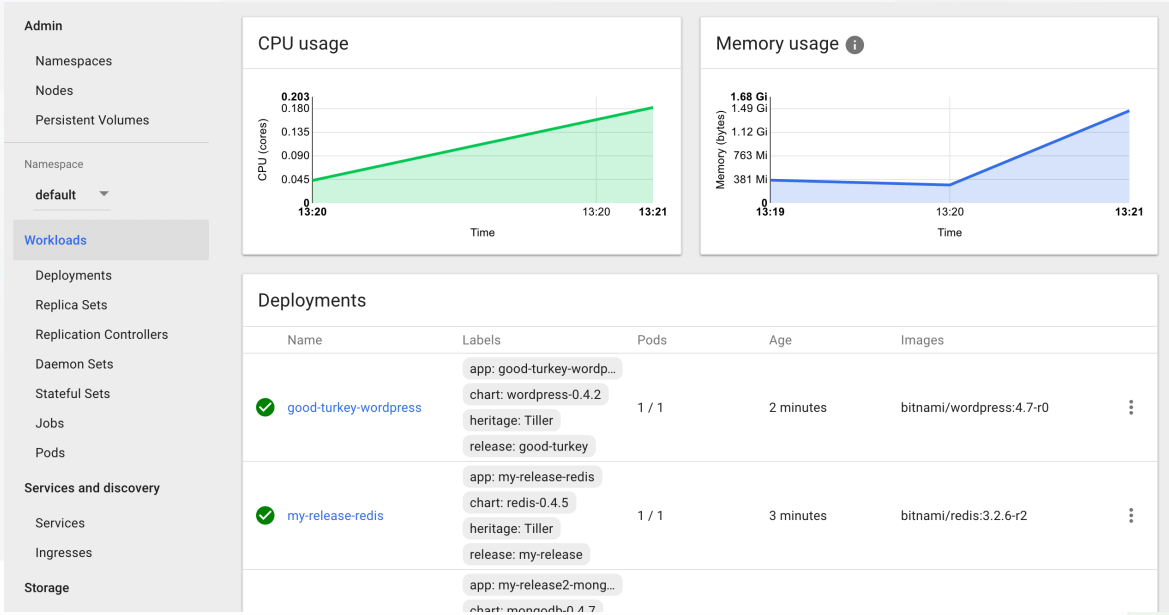
Explicación

De acuerdo con Cornejo (2019), Helm es una herramienta que se utiliza para la gestión de paquetes Kubernetes que, a su vez, se denominan charts.

Un chart es una colección de ficheros que describen a un conjunto de recursos de la API de Kubernetes.

¿Qué es posible realizar con Helm?

- Empaquetar las aplicaciones como charts.
- Interactuar con repositorios de charts (públicos o privados).
- Instalar y desinstalar charts de los clústeres de Kubernetes.
- Gestionar el ciclo de vida de despliegue de charts instalados previamente con Helm.



Explicación

Ventajas de Helm

Los charts en Helm tienen la capacidad de aprovechar los paquetes de Kubernetes con solo hacer clic en un botón o con un solo comando. También se pueden incluir gráficos de Helm dentro de otros gráficos de Helm y tener varias dependencias.

Gráficos de Helm	Es la escalabilidad, ya que todos los gráficos de las imágenes que usa Helm se almacenan en un registro llamado Helm Workspace.
Personalizar las configuraciones de la aplicación.	Se pueden proporcionar configuraciones para todos los recursos de Kubernetes incluidos en la aplicación y también configurar todos los requisitos específicos del entorno para esos recursos.
Cuenta con Chart Hooks de CI/CD	Permiten a los equipos automatizar ciertas acciones para que se lleven a cabo de forma predeterminada. Incluso se pueden programar verificaciones de estado para saber si una implementación se completó con éxito.

Instalación de Helm

Explicación

Paso 1

Instalar la utilidad de línea de comandos Helm en el equipo local. Helm proporciona una secuencia de comandos que gestiona el proceso de instalación en MacOS, Windows o Linux.

```
cd /tmp
curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > install-helm.sh
```

```
chmod u+x install-helm.sh
```

```
./install-helm.sh
```

Paso 2.

Tiller es un complemento del comando Helm que se ejecuta en su clúster; recibe comandos de Helm y se comunica directamente con la API de Kubernetes para hacer el trabajo de crear y eliminar recursos. Para darle a Tiller los permisos que necesita para ejecutarse en el clúster, debemos crear un recurso *serviceaccount* de Kubernetes.

```
kubectl -n kube-system create serviceaccount tiller
```

```
kubectl create clusterrolebinding tiller --clusterrole cluster-admin --serviceaccount=kube-system:tiller
```

```
helm init --service-account tiller
```

```
kubectl get pods --namespace kube-system
```

Paso 3.

Los charts son paquetes de software y Helm viene preconfigurado con un repositorio seleccionado que se llama stable. Para conocer los gráficos disponibles, revisa el siguiente enlace: https://github.com/helm/charts/tree/master/stable

```
$ helm install stable/kubernetes-dashboard --name dashboard-demo
```

```
$ helm list
```

Ejercicio

1. Describe con tus propias palabras los siguientes conceptos:

- DevOps
- Helm
- Chart Helm
- Tiller

2. Analiza el siguiente caso:

En la empresa de desarrollo de software Yellow Tech hay un área de TI donde coexisten dos equipos de trabajo: un área de desarrollo de código y un área de producción. Actualmente, solo hay una persona desarrollando código y en el área de producción hay dos personas que se encargan de la revisión de este, así como de realizar las pruebas de ejecución.

En el último mes, el área de producción ha revisado dos paquetes de códigos del “proyecto A”, que tiene un retraso de entrega de tres días y se ha revisado un paquete de código del “proyecto B” que tiene fecha de entrega para dentro de diez días. Los dos paquetes del “proyecto A” se regresaron al área de programación por fallas.

La comunicación entre las áreas se lleva a cabo solo por correo electrónico. Actualmente, el área de producción está en espera de recibir dos paquetes más de código para el nuevo proyecto. El área de programación está trabajando en estos dos paquetes.

Es importante aclarar que la lectura de correos por parte de los programadores es relativamente baja y el “proyecto A” debe concluirse lo más pronto posible, por lo que el encargado de producción acude al lugar de los programadores para comentarles que urge la corrección de los paquetes devueltos del “proyecto A” y ellos mencionan que lo tienen en la fila porque están trabajando en los paquetes del “proyecto B” faltantes, ya que son los que continúan en su programación de actividades.

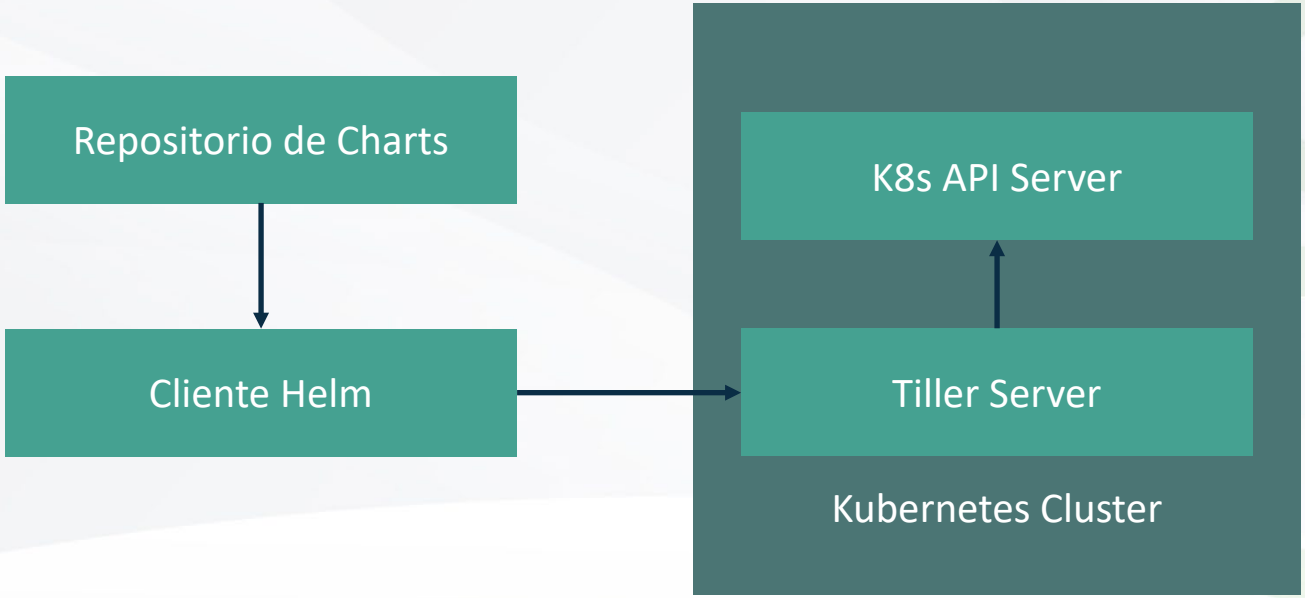
Con base en los principios centrales de la metodología DevOps, desarrolla cuál sería la mejor forma de abordar el caso para cada uno de ellos, con el fin de lograr la finalización de ambos proyectos.

Universidad Tecmilenio

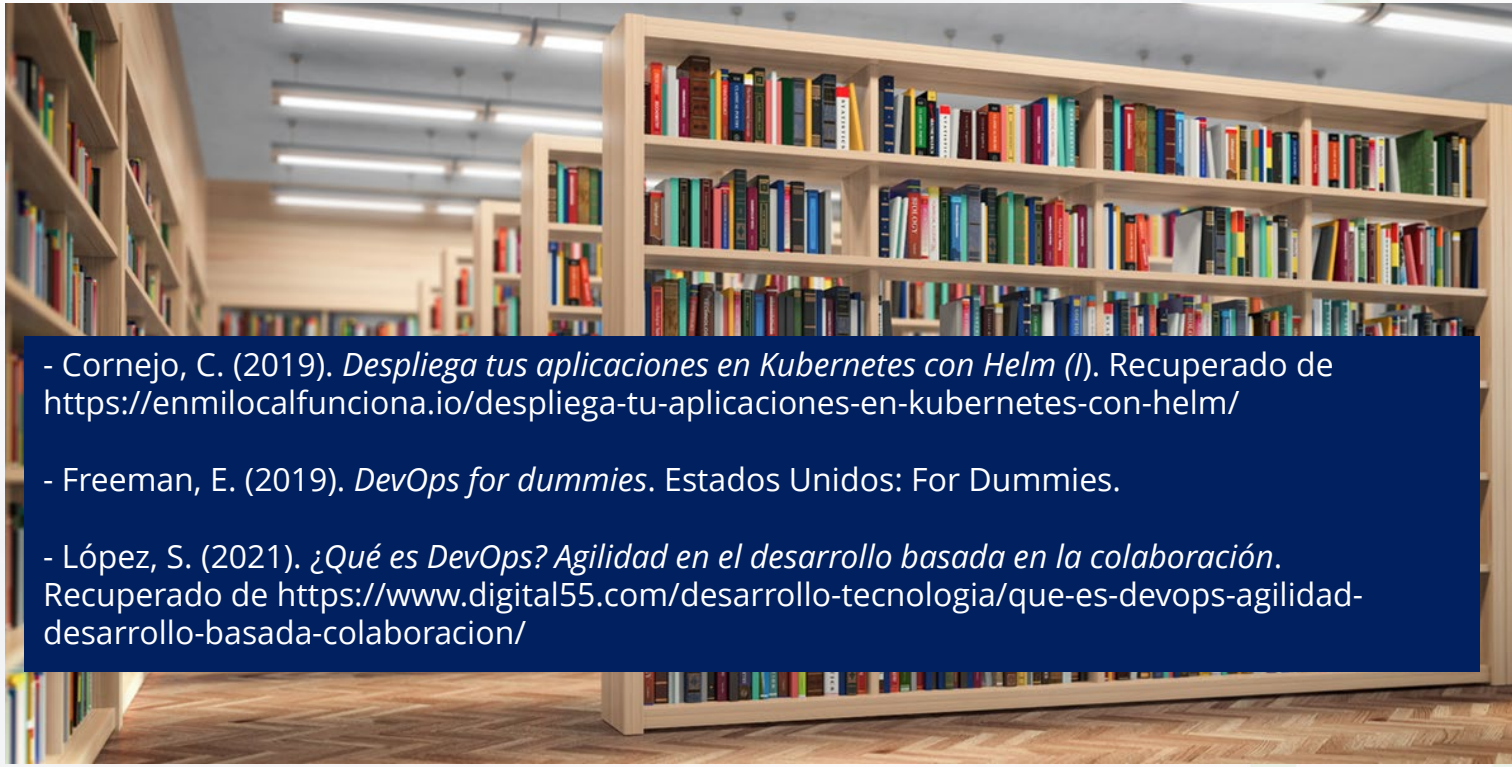
Cierre

Helm facilita el trabajo con Kubernetes y ofrece flexibilidad al poder configurar todos los despliegues a nuestra forma de trabajo.

Se convierte en una herramienta muy poderosa para compartir a la comunidad nuestras propias implementaciones de despliegues, permitiendo que puedan evolucionar por sí mismas y llegar a convertirse en un estándar. Realmente detona el despliegue de Kubernetes a otro nivel.



Bibliografía re



- Cornejo, C. (2019). *Despliega tus aplicaciones en Kubernetes con Helm (I*). Recuperado de https://enmilocalfunciona.io/despliega-tu-aplicaciones-en-kubernetes-con-helm/

- Freeman, E. (2019). *DevOps for dummies*. Estados Unidos: For Dummies.

- López, S. (2021). *¿Qué es DevOps? Agilidad en el desarrollo basada en la colaboración*. Recuperado de https://www.digital55.com/desarrollo-tecnologia/que-es-devops-agilidad-desarrollo-basada-colaboracion/

Universidad Tecmilenio

Empaquetado en DevOps

Implementando Helm

Semana 11

Introducción

Para la arquitectura de servicios, cada uno de los procedimientos definidos son servicios independientes.

Si nos enfocamos, por ejemplo, en el sistema de facturación, dependiendo del tamaño de la empresa que está ofreciendo estos servicios, puede haber un equipo que realice el desarrollo y control de calidad de dicho microservicio y, en consecuencia, este procedimiento puede tener su propio calendario de operaciones y publicaciones, así como manuales y estrategias de implementación

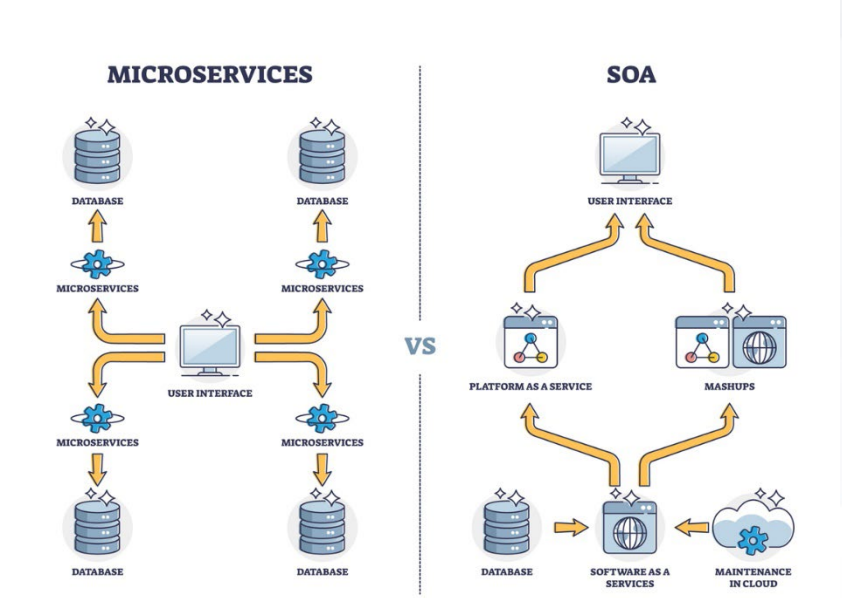
Explicación

Implementación en una arquitectura de microservicios

El concepto de arquitectura de microservicios, según Hiberus (2021), es un método de desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños servicios con operaciones bien definidas e independientes entre sí.

Una de las principales diferencias entre la estructura tradicional y la de microservicios es que la primera se realiza de manera monolítica, donde todas las partes se encuentran juntas en la misma aplicación y de forma integral.

En la arquitectura de microservicios se descompone la aplicación en sus funciones principales y cada una de estas se llama ahora microservicios, con la ventaja de que se pueden diseñar e implementar de forma independiente.



Explicación

Arquitectura de microservicios

En una arquitectura de microservicios, cada uno de ellos hace una tarea simple y puede comunicarse con otros microservicios o con los clientes a través de mecanismos como una API.

Es posible lograr la persistencia de datos mediante una base de datos, mientras que los archivos de imagen y medios para la aplicación se pueden almacenar en la nube.

Ventajas:

- Son más fáciles para realizar tests y en el tema de mantenimiento.

- No se encuentran integrados en el sistema principal.

- Cada servicio puede utilizar diferentes tecnologías.

Desventajas:

- Será necesario implementar comunicación interna entre los servicios.

- Complejidad de despliegue: la administración de microservicios es muy compleja por definición.

- Uso elevado de recursos.

Explicación

Desarrollo en microservicios

Uno de los principales aspectos es definir una estrategia de desarrollo en microservicios que sea viable y permita obtener los beneficios previstos en el ciclo de vida. Por otra parte, el ciclo de vida del desarrollo de microservicios se divide en tres fases (Hiberus, 2021):

Fase de lanzamiento.	Fase de expansión.	Fase de estabilización.
Se debe confirmar la estrategia definida para el desarrollo con base en microservicios.	Se caracteriza por centrarse en el control y administración de la expansión del uso de los microservicios.	En esta fase el aspecto principal es enfocarse en la eficiencia y al mismo tiempo mantener la calidad de los desarrollos.

Explicación

Creación de Helm charts

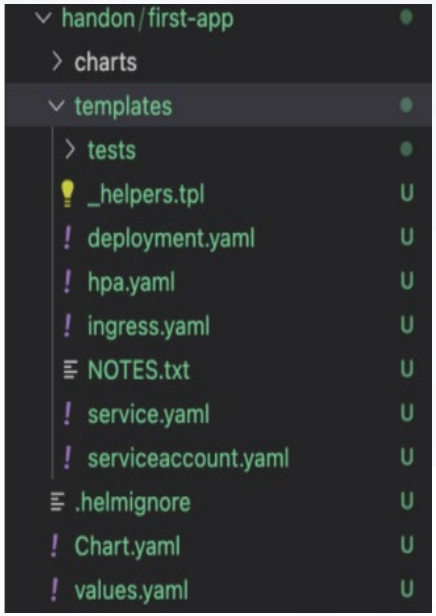
Estructura de un chart
Se conforma internamente como un árbol de directorios.

Paso 1
La creación de los charts en Helm se lleva a cabo a través de un comando muy sencillo:

```
helm create <NOMBRE DEL CHART>
```

Paso 2
Ir al directorio en el que se desea crear el chart y ejecutar el siguiente comando:

```
helm create first-app
```



Un chart es una aplicación que puede instalarse en un clúster y se compone principalmente de ficheros YAML de Kubernetes (objetos de Kubernetes), como un service, un ingress, un deployment, etc.

A continuación, se enlistan los archivos con metadatos que siempre deben incluirse en un chart:

Chart.yaml: incluye toda la información.

Values.yaml: determinan los valores por defecto de la aplicación.

README.md: se incluye la descripción de todos los valores personalizables y los comandos.

Cierre

1. De acuerdo con el concepto de arquitectura de microservicios, explica un ejemplo real de una aplicación que cuente con este tipo de estructura y justifica tu respuesta.

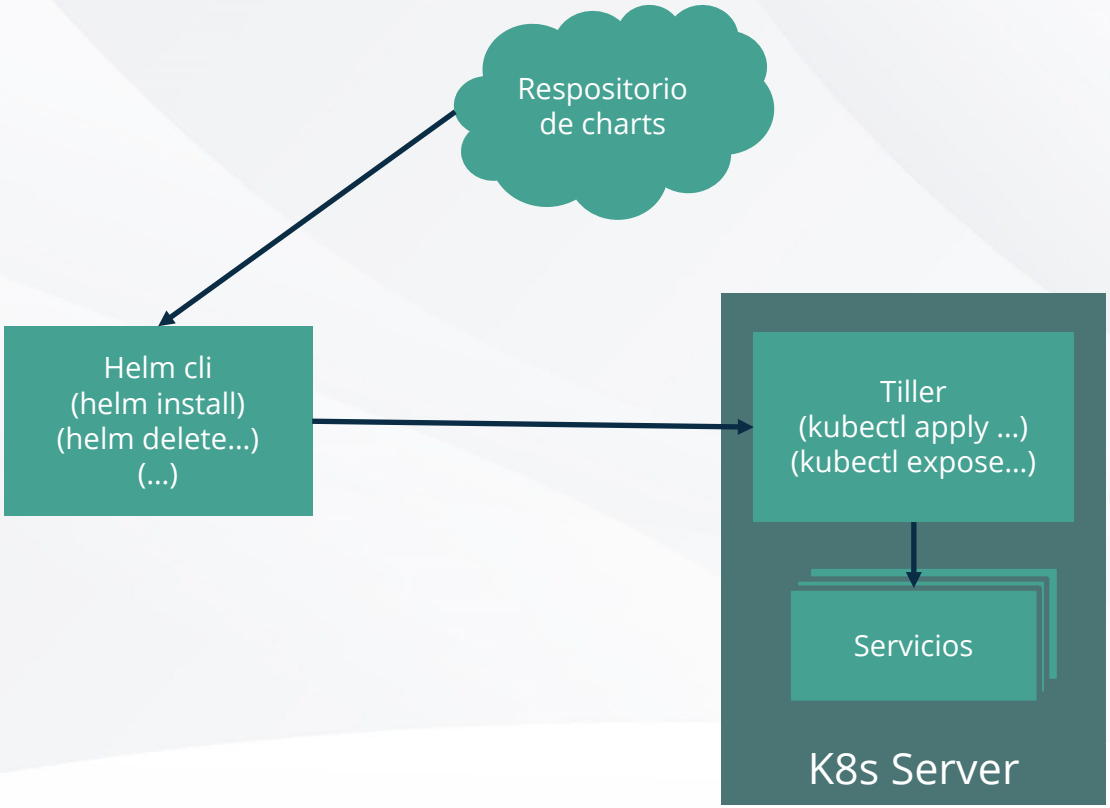
2. Con base en las tres fases del ciclo de vida en el desarrollo de microservicios, analiza y realiza la solución más adecuada para migrar a una operación por microservicios (con base en el caso de la actividad del tema 11) .

• Actividades de cambio para cada fase.
• Realiza un ejemplo de Helm chart que apoye la organización del nuevo equipo de trabajo bajo una metodología DevOps.

Cierre

A lo largo de la presentación se han visto algunas características y ventajas de la creación de charts con Helm para Kubernetes.

También se presentó el despliegue de la aplicación haciendo uso de Helm y sus templates, lo que ayudará mucho en la creación de tareas repetitivas o cuando se tengan que realizar despliegues de aplicaciones con características parecidas.



Bibliografía

- Hiberus. (2021). *De una arquitectura tradicional a una arquitectura microservicios*. Recuperado de https://www.hiberus.com/crecemos-contigo/de-una-arquitectura-tradicional-a-microservicios/

Universidad Tecmilenio

Empaquetado en DevOps

GitHub Actions - CI/CD (avanzado)

Semana 11

Universidad Tecmilenio

Introducción

Apoyarse en GitHub Actions ayudará a que las soluciones creadas para una aplicación no queden aisladas, sino que puedan ser compartidas y convertidas a otras aplicaciones sin la necesidad de partir desde cero.

Será muy importante que el líder del equipo de trabajo pueda establecer líneas de trabajo apoyadas por la combinación de procesos de integración continua (CI) y procesos de entrega continua (CD), considerando sus beneficios para una mejor administración de proyectos donde las implementaciones serán la entrada para el siguiente ciclo de trabajo de desarrollo.

Explicación

GitHub

GitHub es una plataforma de integración y entregas continuas (CI/CD) que permite automatizar los procesos de compilación, prueba e implementación.

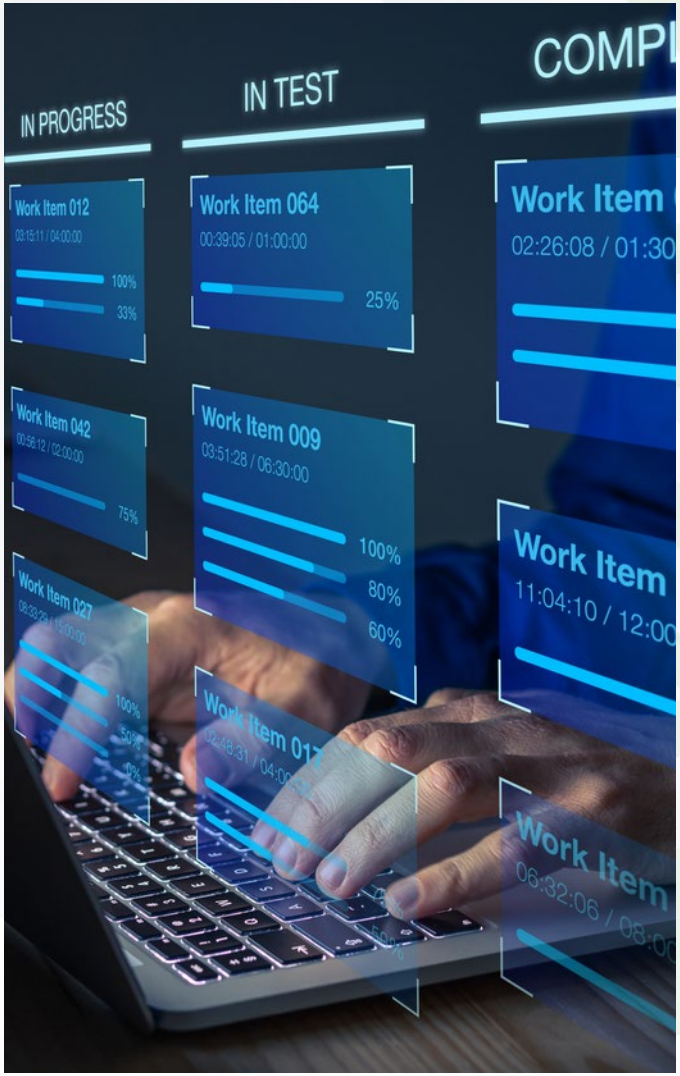
Permite contar con flujos de trabajo cuando otros eventos están realizándose en el repositorio.

Flujos de trabajo

Son un proceso automatizado configurable que realiza la ejecución de los jobs.

Se definen mediante un archivo YAML que se verifica en el repositorio y se ejecuta cuando lo activa un evento dentro del mismo.

Puede haber muchos flujos de trabajo en un mismo repositorio y cada uno de ellos puede ejecutar diferentes pasos. Se puede tener un flujo para probar solicitudes de cambio, desplegar la aplicación y agregar etiquetas (GitHub Docs, s.f.).



Explicación

YAML files

Son un formato de serialización de datos que indica que YAML no es un lenguaje de marcado (Ghimire, 2021).

Aspectos principales:
- Es un lenguaje que se basa en datos con características de C, HTML, Perl y otros lenguajes.
- Es un superconjunto derivado de JSON.
- Tiene una infraestructura de alto rendimiento.

Explicación

Workflow

Workflow: se considera un procedimiento automatizado que se añade a un repositorio. A través de su uso se puede realizar el *build*, *test*, *package*, *release* o *deploy* de un proyecto dentro de GitHub.

Job: es un conjunto de pasos (*steps*) que se ejecutan en el *runner* del proceso que se está realizando.

Step: es una actividad individual que puede ejecutar comandos dentro de un job.

Action: son los comandos de ejecución del proceso que se van a ejecutar en un step para crear un job. Es el bloque de instrucción más pequeño que existe.

Etapas de CI/CD

Universidad Tecmilenio

Explicación

CI/CD se considera como un método para distribuir aplicaciones a los clientes con cierta frecuencia a través de la automatización.

CI se refiere a la integración continua (proceso de automatización de desarrolladores).
CD se enfoca en la implementación continua, teniendo su mayor uso en la automatización de las etapas posteriores del proceso.

Etapa de compilación	El código se recupera tanto de fuentes internas como de externas y luego se compila si es necesario.
Etapa de prueba	Donde se revisa el código para encontrar defectos.
Etapa de lanzamiento	Análisis de sensometría, minería de datos y ecuaciones estructurales.

Ejercicio

1. Investiga tres herramientas que apoyen la implementación de un sistema CI/CD para una empresa de tecnologías de la información que tiene poco más de un año de estar desarrollando proyectos para pymes. Explica cada una de ellas con tus propias palabras.

2. Basándote en las etapas de CI/CD, analiza y realiza la solución más adecuada para el siguiente departamento de tecnologías de la información.

La empresa Red Tech, con más de un año de operaciones, cuenta con un equipo de desarrolladores que realizan todas sus actividades a través de un plan de administración de proyectos tradicional, donde la comunicación se da a través de correo electrónico y de manera verbal, lo que, en ocasiones, puede provocar que no se guarde evidencia de los acuerdos entre el equipo de desarrolladores y el área de producción.

De acuerdo con la filosofía de microservicios, no se debe presentar la situación de contar con una larga serie de versiones en la que todos los equipos tengan que hacer fila, es decir, un equipo encargado de compilar un servicio “A” podría publicar una actualización en cualquier momento sin necesidad de esperar a que los cambios realizados en un servicio “B” sean combinados, probados e implementados.

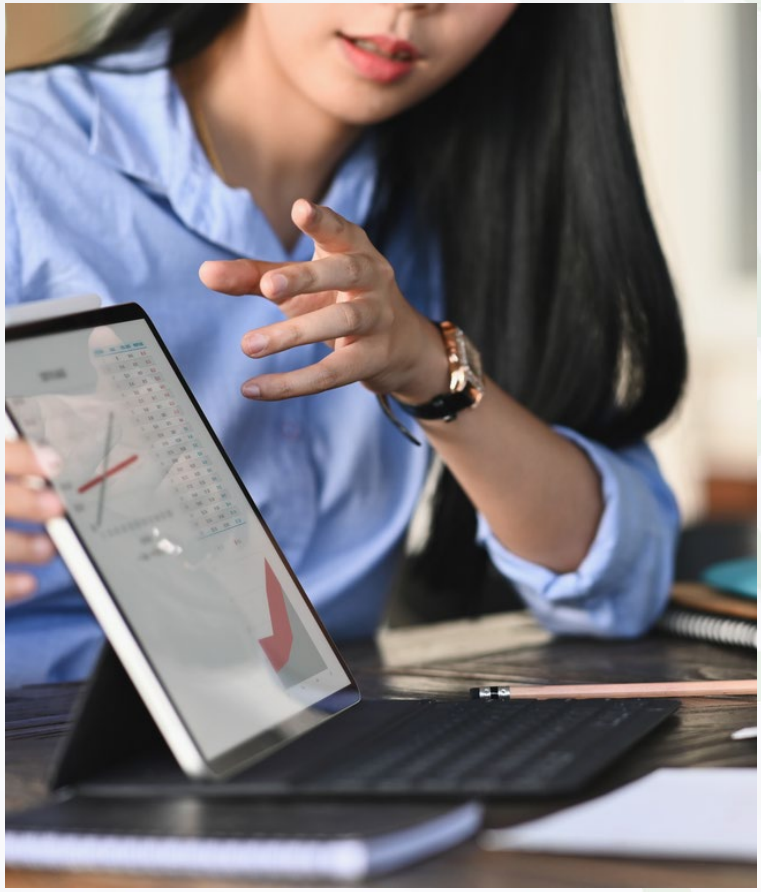
Con base en las etapas CI/CD, determina lo siguiente:

a. ¿Cómo implementarías las etapas de CI/CD en este departamento con el fin de dar solución al caso?
b. Para cada uno de los siguientes desafíos a los que se enfrenta el departamento, establece la forma de mitigarlos.

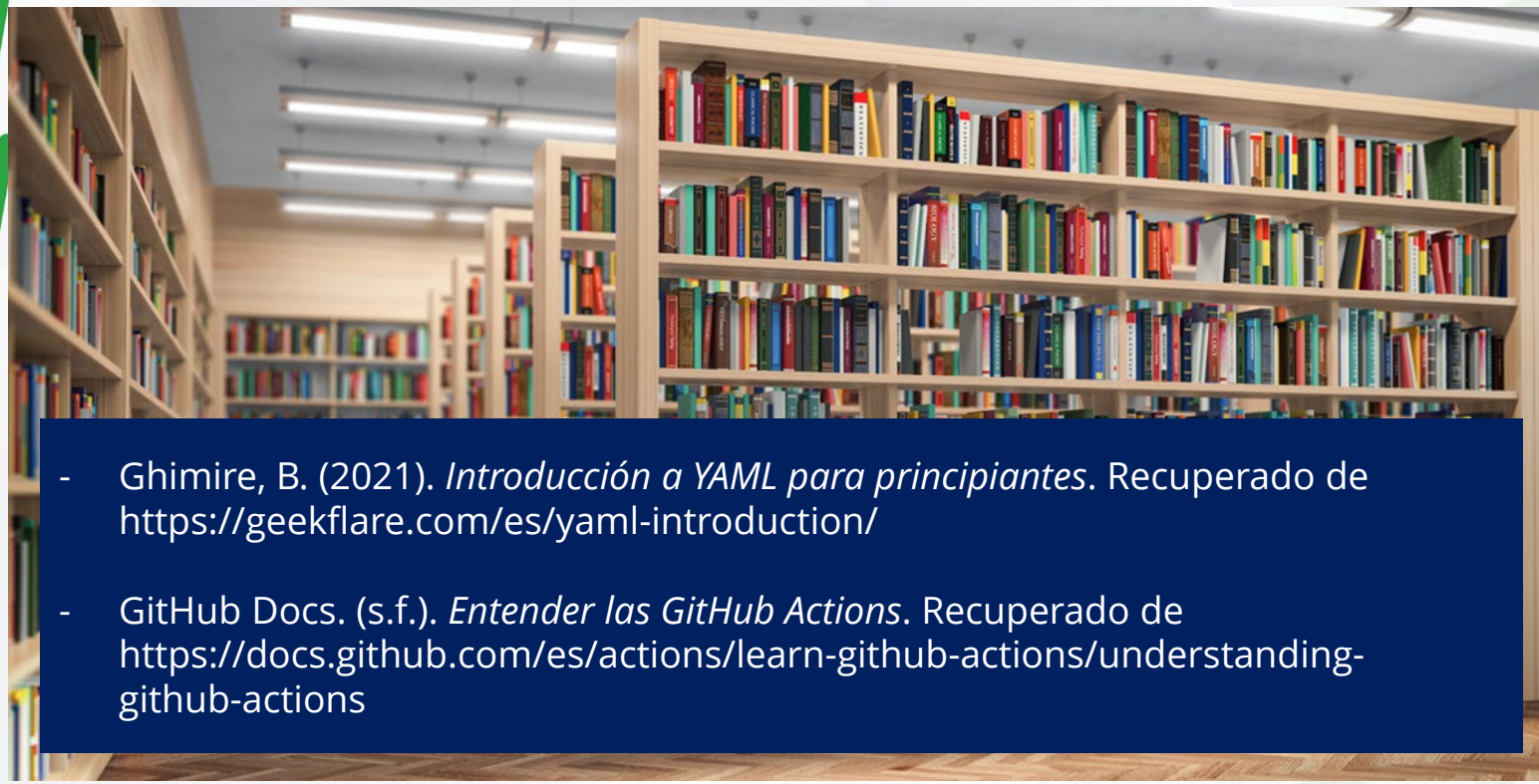
Cierre

Contar con herramientas como GitHub nos ayuda a potenciar cada una de las etapas de DevOps, tanto los equipos de trabajo como la productividad y, aunado a esto, la metodología CI/CD aporta elementos valiosos para no solo lograr producir aplicaciones en tiempos adecuados, sino también para lograr la total satisfacción del cliente, permitiendo un ciclo virtuoso entre programadores, producción y clientes.

A través de los archivos YAML se pueden lograr interacciones que facilitan la conectividad con otras aplicaciones que se estén desarrollando por diferentes equipos de trabajo y todos estos elementos tendrán como resultado equipos de trabajo bien gestionados con pocas probabilidades de fallo o retrabajo.



Bibliografía



- Ghimire, B. (2021). *Introducción a YAML para principiantes*. Recuperado de https://geekflare.com/es/yaml-introduction/

- GitHub Docs. (s.f.). *Entender las GitHub Actions*. Recuperado de https://docs.github.com/es/actions/learn-github-actions/understanding-github-actions

Universidad Tecmilenio

Empaquetado en DevOps

Pipeline de despliegue

Semana 11

Introducción

En una empresa, los desarrolladores trabajan en la realización de un software confiable e innovador en un lapso corto. Por otra parte, el área de operaciones de TI se encarga de implementar, configurar y optimizar los recursos para que el software en desarrollo funcione correctamente y sea estable.

Sin embargo, estas áreas regularmente trabajan por separado, por lo que necesitan alternativas para agilizar estos procesos.

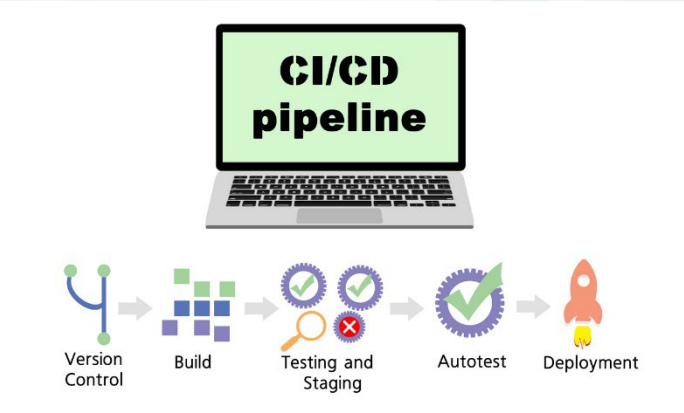
En este punto entran los *pipelines* de despliegue, que se convierten en una parte crucial para el desarrollo de software de calidad.

Explicación

Pipeline

Es un principio fundamental en DevOps que nos permite estructurar el ciclo de vida completo de un desarrollo de software.

El pipeline se forma por un conjunto de procesos o herramientas automatizadas que permiten el desarrollo paso a paso de manera coherente para implementar los códigos de los desarrolladores en un ambiente de trabajo real.



Despliegue

Es la automatización que implementan las aplicaciones a los diferentes entornos de trabajo.

El pipeline de CD más común tiene etapas de compilación, pruebas e implementación, pero también existen algunos pipelines más avanzados que incluyen extracción de código, ejecución de los pasos de infraestructura, mover código, administrar las variables de entorno y configurarlas, entre otros.

Explicación

Etapas principales

Existen tres etapas principales para los pipelines de despliegue, las cuales tienen tareas que son repetitivas y que pueden automatizarse a través de scripts escritos en YAML (Oquendo, 2021).

Integración.	Entrega a un ambiente de pruebas.	Despliegue a producción.
Implica realizar las tareas de compilación del código, ejecución de pruebas y el análisis de la calidad del código.	a. Despliegue de archivos de compilación creados en la primera etapa. b. Automatizar pruebas de integración y aceptación. c. Realizar pruebas no funcionales (usabilidad, seguridad y rendimiento). d. Realizar pruebas manuales de usuario.	Incluye las siguientes actividades: - Despliegue de los archivos generados después de la compilación de la primera etapa. - Validar las nuevas funcionalidades. - Validar las funcionalidades anteriores.

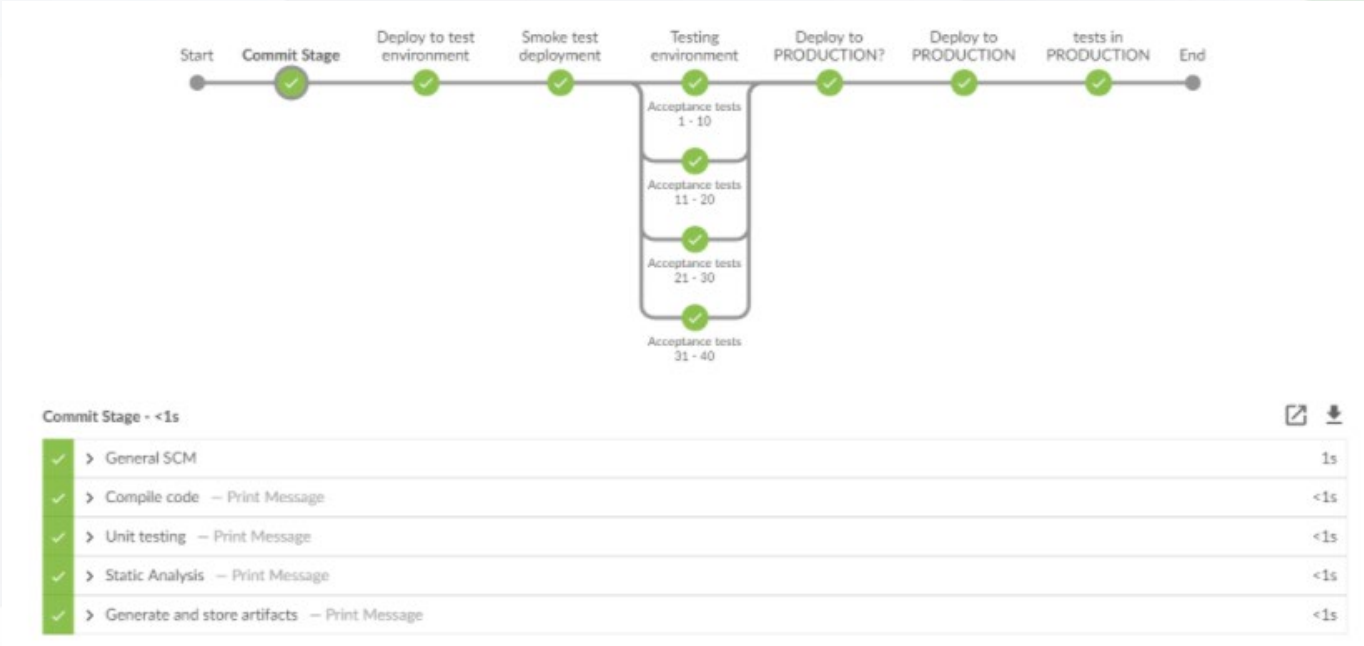
Explicación

Implementación de pipelines de despliegue

Al operar pipelines de CI/CD en la nube se pueden utilizar contenedores y sistemas de orquestación como Kubernetes, los cuales permiten empaquetar y desplegar aplicaciones de manera portátil y estándar.

El uso de pipelines de CI/CD está enfocado para el uso empresarial, donde se pueden modificar las aplicaciones con mayor frecuencia y van a requerir procesos de entregas confiables.

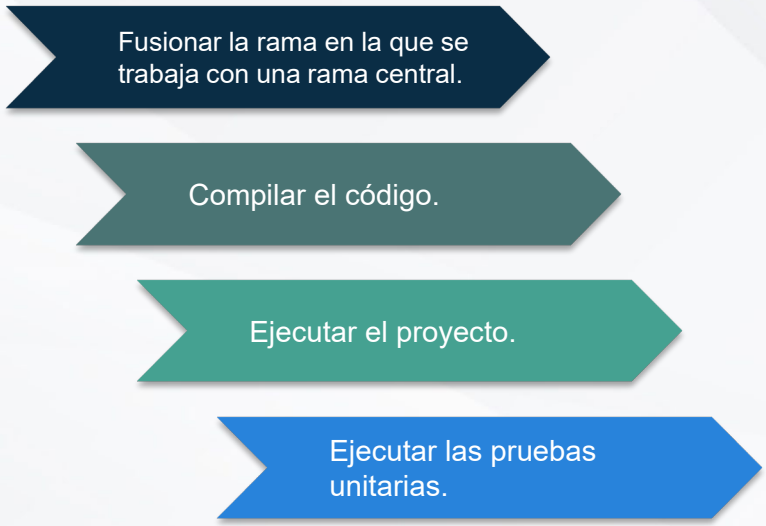
Un aspecto para considerar es que es posible medir el impacto de la implementación de pipelines de CI/CD, como un indicador clave de rendimiento (KPI) en DevOps.



Explicación

Postdespliegue

El pipeline se activará al hacer *push* a una rama del repositorio central que se encuentre en alguna plataforma en línea como GitHub, que tendrá como objetivo principal ejecutar de manera automática las siguientes actividades (Lorenzo, 2021).



Explicación

Consideraciones al implementar un pipeline

Es importante analizar las partes que se podrán automatizar y si es posible modelarlo con una herramienta que permita automatizar tareas y visualizarlas en forma gráfica.

Asegurar que el estado del pipeline sea siempre correcto, es decir, todo el equipo es responsable del pipeline y de su estado.

Es recomendable comenzar con un "esqueleto" que presente las etapas del pipeline y de manera sucesiva añadir más tareas dentro de las etapas.

Ejercicio

1. Investiga en Internet tres ejemplos de herramientas que puedas utilizar para realizar una implementación de un pipeline de despliegue y describe las características de cada una.

2. Analiza el caso.

En la empresa de desarrollo de software BlackTech se desarrollan aplicaciones para clientes de la industria del marketing; en ella trabaja un grupo de desarrolladores que deben compartir información acerca del funcionamiento de la aplicación y sus características.

Deberán realizar actualizaciones de forma periódica, así como integraciones y pruebas del código integral. El tema de planificar la incorporación del código y realizar dichas pruebas llevaría mucho tiempo de desarrollo, por lo que es necesario un sistema que ayude con la integración continua.

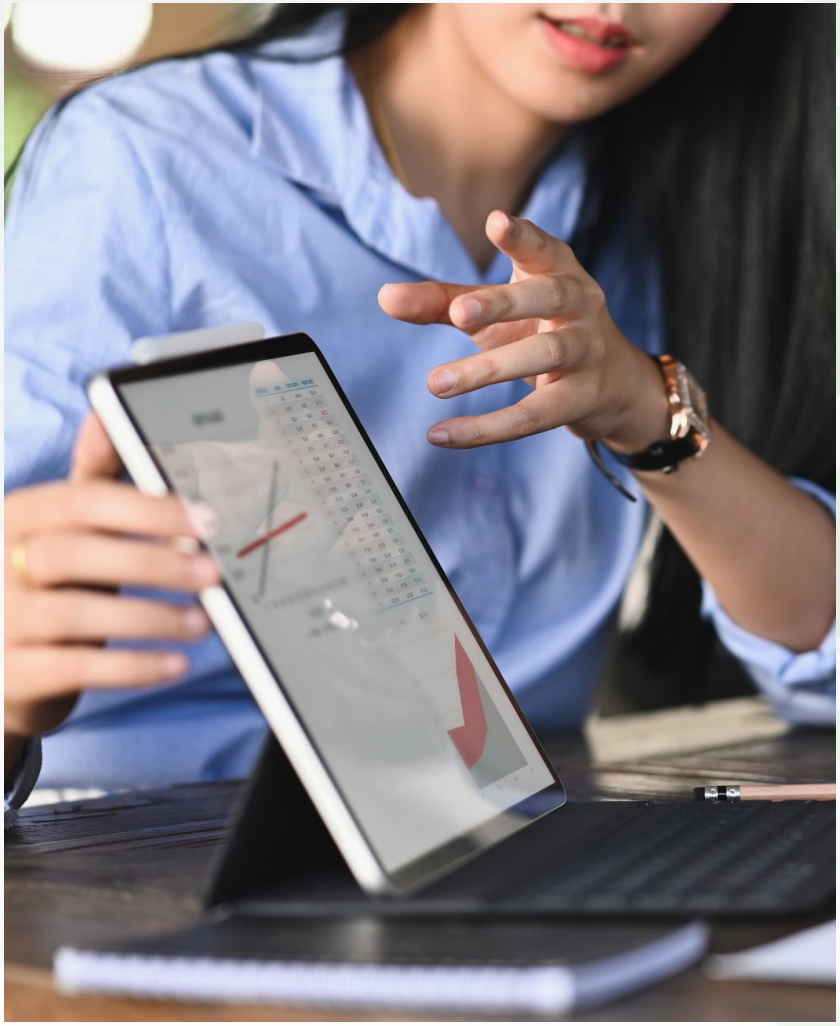
Para que el desarrollo de las aplicaciones esté listo en tiempo y forma, incluyendo los puntos mencionados anteriormente, será necesario considerar los siguientes aspectos:

1. ¿Cuándo se debe empezar a probar la integración del código?
2. ¿Cómo realizar pruebas para asegurar que la integración sea exitosa?
3. ¿Cómo se deben comunicar los resultados al equipo?

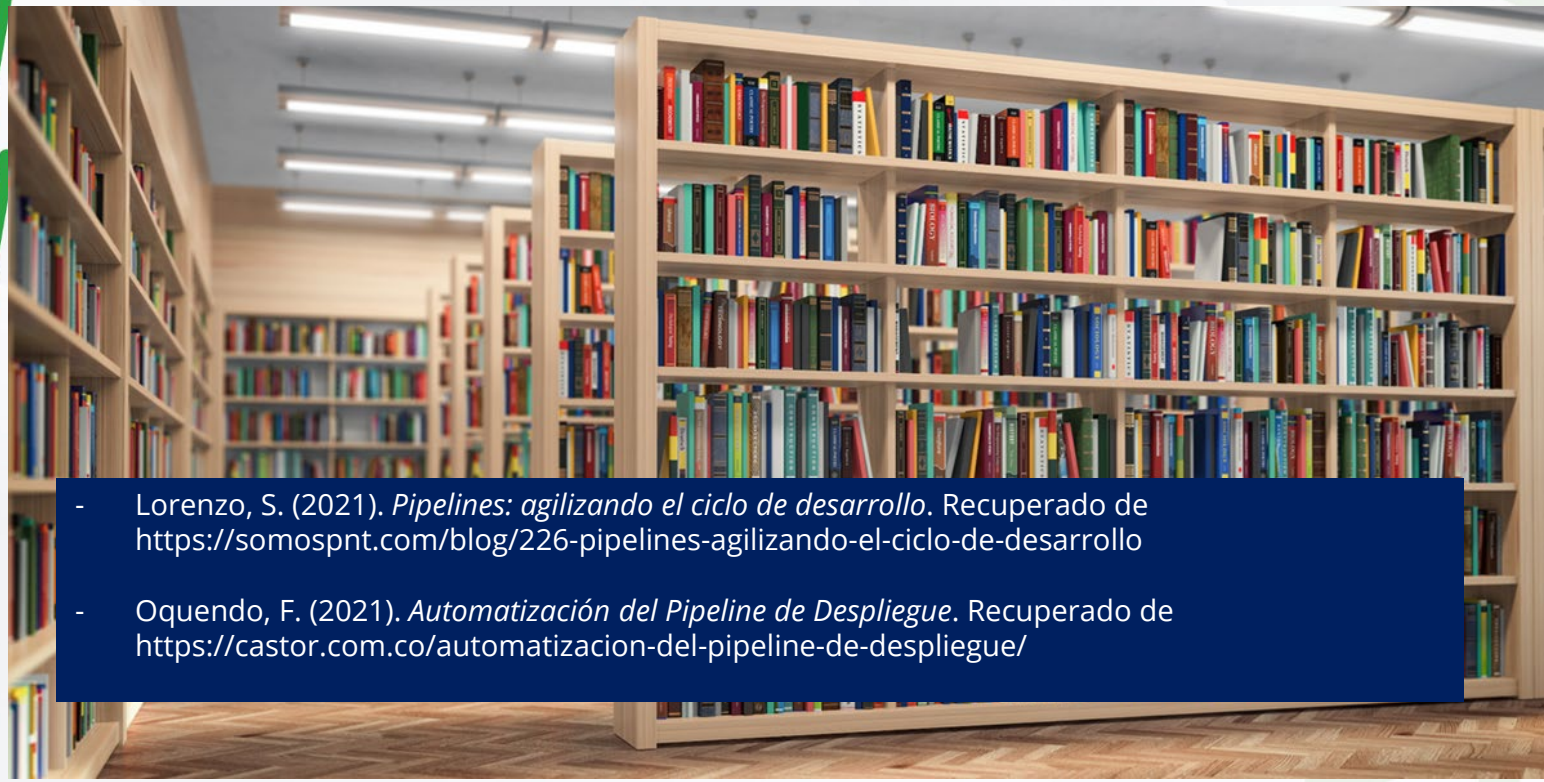
Cierre

Un pipeline en un equipo de desarrollo es sumamente importante, ya que no solo nos permite tener el control de cada uno de los aspectos de una aplicación, sino de cómo se pueden optimizar los recursos en cada una de sus etapas, teniendo siempre en mente que la finalidad de una adecuada administración de proyectos es generar sistemas eficientes y eficaces con altos índices de calidad.

Se recomienda contar con implementaciones exitosas a través de líneas de despliegue, que apoyen a los equipos de TI a generar software confiable y en pequeños lapsos de tiempo, permitiendo la integración de las áreas más cruciales del proyecto, con el fin de agilizar procesos y aprovechar al máximo los recursos disponibles.



Bibliografía



- Lorenzo, S. (2021). *Pipelines: agilizando el ciclo de desarrollo*. Recuperado de https://somospnt.com/blog/226-pipelines-agilizando-el-ciclo-de-desarrollo

- Oquendo, F. (2021). *Automatización del Pipeline de Despliegue*. Recuperado de https://castor.com.co/automatizacion-del-pipeline-de-despliegue/