

### Módulo 3 - Clase 13

#### **Introducción al mundo de los pipelines**

##### **Pipelines ¿qué son y para qué sirven?**

Hoy en día, uno de los factores clave para los negocios es la velocidad con la que se desarrollan los productos de software. ¡Los viejos procesos manuales ya no son compatibles con los tiempos que corren! La automatización viene a resolver en parte este nuevo paradigma mediante la concatenación de procesos de desarrollo. Para entender de qué se trata... ¡Hablemos de Pipelines!

##### ***Video***

Imaginamos una fábrica que produce un bien físico necesita materia prima ciertas máquinas y personal que las controle y procesos que definen de forma lógica la secuencia de pasos que deben ejecutar para lograr el producto que finalmente saldrá de la fábrica y llevar a cabo algún mercado para su consumo en infraestructura como código estos procesos lógicos que permiten la elaboración de un producto son los que se conocen como Pipeline

Un pipeline es uno o más procesos automatizados que sirven para construir implementar cierto código de software de forma rápida, escalable, flexible y segura minimizando errores humanos que derivan de la tarea manual por ejemplo sería el siguiente

Tenemos una plataforma cloud que hospeda un sitio web y queremos implementar cierta feature una nueva opción para ello necesitamos los siguientes elementos

- -pieza nueva de software
- -los pasos que el pipeline realizara
- -Accesos apropiados al ambiente de testing
- -destino donde esta feature será implementada

Vamos a git y pusheamos el código ya revisado por alguien mas a un repositorio remoto un lugar con otra carpeta a otro servidor nuestro producto de pipelines recibe el código y este es interpretado según una serie de reglas preestablecidas

##### ***¿Cuáles podrían ser estas reglas?***

1. Validar que exista un motor de software que pueda compilar el código e decir si nuestro código de input es hcl entonces vamos a precisar el binario de terraform para que pueda interpretarlo correctamente si nuestro código viene hecho el Go vamos a necesitar tener instalado el compilador de GO
2. Validar que no exista errores de sintaxis
3. Validar que todos los módulos, referencias o ramificaciones estén presentes
4. Validar que no se esté violando la seguridad
5. Versionar y guardar el o los elementos resultantes de cada etapa

Nuestro código pasará por cada una de estas etapas y puede o no ir arrojando un resultado que luego será consumido por la próxima etapa de pipeline finalmente el código puede ser compilado, impactado en el sitio web que será actualizado automáticamente, es importante destacar que según la madurez del proceso y del equipo la salida a nuestro pipeline puede no impactar a ni siquiera a nuestro ambiente de testing si no que se frena el proceso un paso antes y se requiere una acción manual para realizar el pasaje al sitio web

Los pipelines dentro de la infraestructura moderna nos brinda una serie de ventajas como

- velocidad al ser un proceso automatizado donde no hay que ejecutar comandos o hacer click en opciones determinadas permite correr a velocidad de procesamiento que se haya destinado a tal proceso

- consistencia siempre vamos a obtener el mismo resultado logrando la integridad del producto final
- seguridad con el proceso automatizado y en un ambiente controlado vamos a tener la certeza de que no estamos introduciendo ninguna vulnerabilidad
- versionado como mencionamos anteriormente podemos ir guardando los elementos resultantes por cada etapa o iteración con determinados tags o números de versión

El uso de pipeline es fundamental en cualquier proceso de infraestructura como código

**Pipelines** → procesos lógicos que permiten la elaboración de un producto. Este es uno o más procesos automatizados que sirven para construir e implementar cierto código de software de forma rápida, escalable, flexible y segura, minimizando errores humanos que derivan de la tarea manual.

por ejemplo: tenemos una plataforma cloud que hospeda un sitio web y queremos implementar cierta Feature, una nueva opción, para ello necesitamos los siguientes elementos: nuestra nueva pieza de software, los pasos que el pipeline realizará, acceso apropiados al ambiente de testing y el destino donde está feature será implementada.

Via git pushiamos el código ya revisado por alguien más a un repositorio remoto, nuestro producto de pipelines recibe el código y este es interpretado según una serie de reglas preestablecidas; estas reglas podrían ser: primero validar que exista un motor de software que pueda compilar el código, es decir, si nuestro código de input es HCL, entonces vamos a necesitar el binario de terraform para que pueda interpretarlo correctamente,

Si nuestro código viene hecho en GO vamos a necesitar tener instalado el compilador de GO, la segunda validar que no existan errores de sintaxis, la tercera validar que todos los módulos, referencias o ramificaciones están presentes, la cuarta validar que no se estén violando las condiciones específicas de seguridad, la quinta versionar y guardar el o los elementos resultantes de cada etapa, nuestro código pasara por cada una de estas etapas y puede o no ir arrojando un resultado que luego será consumido por la próxima etapa del pipeline, finalmente el código puede ser compilado, impactado en el sitio web que será actualizado automáticamente.

Es importante destacar que según la madurez del proceso y del equipo, la salida de nuestro pipeline puede no impactar ni siquiera en nuestro ambiente de testing, sino que se frena el proceso un paso antes y se requiere una acción manual para realizar el pasaje al sitio web.

Los pipelines dentro de la infraestructura moderna nos brindan una serie de ventajas como velocidad, al ser generalmente un proceso automatizado donde no hay que ejecutar comandos o hacer clic en opciones determinadas, permite correr a la velocidad de procesamiento que se haya destinado a tal proceso, consistencia, siempre vamos a obtener el mismo resultado logrando la integridad del producto final, seguridad, con el proceso automatizado y en un ambiente controlado vamos a tener la certeza de que no estamos introduciendo ninguna vulnerabilidad, versionado, podemos ir guardando los elementos resultantes por cada etapa o iteración con determinados tags o números de versión.

El uso de estos es fundamental en cualquier proceso de infraestructura como código y en esencia toda persona que trabaje en software tarde o temprano se va a encontrar con este tipo de arquitectura de release.

**Pipelines** es una práctica que nos sirve para agilizar los procesos de desarrollo e implementación de software que se vale de automatizar pasos repetitivos según un procedimiento preestablecido. Esta arquitectura es bastante común en el desarrollo de software porque tiene la propiedad de agilizar los procesos de **build, deploy y release**.

Para hacernos una idea más concreta, lo podemos pensar como una serie de comandos encadenados donde cada comando está en un nivel determinado. Cuando el comando finaliza su ejecución, el resultante pasa al siguiente nivel donde otro lo espera para ejecutar una función predeterminada.

Por ejemplo:

- Una etapa puede ser el chequeo de que aquello que ingresó en el proceso cumpla con ciertos requisitos iniciales de seguridad. Si estos requisitos no satisfacen ciertas reglas predefinidas de antemano, entonces no puede continuar adelante.
- En una etapa posterior se puede, por ejemplo, compilar el código.
- En la siguiente puede ser almacenado en algún lugar específico. Así, sucesivamente, hasta llegar al final de la línea.

Un pipeline no es un proceso monolítico sino que puede consistir en una o varias etapas que están encadenadas o secuenciadas y que van transformando a nuestro elemento inicial de acuerdo con ciertas reglas hasta llegar al producto final.

#### Cuestionario

- 1. Un pipeline es una práctica de software destinada a:**
  - Desarrollos de aplicaciones web
  - Desarrollos para infraestructura de redes.
- 2. Un pipeline puede mejorar significativamente el tiempo de despliegue de una aplicación. Esto dependerá de:** Cómo esté construido el workflow.
- 3. Un pipeline puede mejorar significativamente la seguridad de una aplicación. Esto dependerá de:** el producto elegido contenga la etapa de iteración encargada de testear el código.
- 4. Nuestro código ha ingresado a un pipeline, se ha procesado correctamente y ha actualizado nuestro negocio: un sitio web. ¿Cuál de las siguientes opciones es válida?**

Sin haber validado el resultado de cada etapa del pipeline, no se debió continuar con el proceso.

Al ser código que tiene impacto directo en el negocio, revisar la seguridad es un punto crucial antes de deployar.

Ninguna de las anteriores.

- 5. Finalicé mi código y lo subí al repositorio correspondiente. De forma automática, comienza la ejecución de las distintas etapas ya definidas. Inmediatamente, la salida muestra un error de sintaxis. ¿Qué procedimiento debo emplear?**
  - Cancelar la ejecución del pipeline (si es que no se ha cancelado ya), corregir mi código y volver a pushear. o
  - Cancelar la ejecución del pipeline (si es que no se ha cancelado ya), instalar el compilador correcto y volver a pushear.

#### ¿Qué tecnologías existen?

Cuando hablamos de “collaboration” en términos de archivos, quizás en lo primero que pensamos es en Dropbox o Google Drive. Estos sistemas son fáciles de utilizar, poseen una interfaz intuitiva y permiten controlar archivos. Sin embargo, es bastante difícil adaptarlos a las necesidades con las que nos podemos encontrar al trabajar en tecnología. La industria del software requiere de productos o soluciones que puedan:

- Actualizar código en tiempo real.

- Controlar conflictos en el código.
- Disponer de “Roadmaps” para planificar y determinar milestones.
- Sincronizar versiones de archivos.
- Almacenar grandes cantidades de archivos.
- Automatizar procesos de compilación de archivos.

Este tipo de requerimientos son contemplados por **productos muy específicos** creados para tal función. ¡Veamos algunos de los más conocidos!

Exploremos productos de CI: GITHUB; GITLAB; BITBUCKET



### GITHUB

- Es un servicio de repositorio remoto y control de versiones para colaboración en proyectos de código de software.
- Tiene una característica llamada **github issues** que funciona como herramienta de ticketing.
- Dispone de **boards** para hacer seguimiento sobre “issues” creados.

Más información en:  
<https://github.com/about>



### GITLAB

- Al igual que GitHub, es una herramienta de repositorio que permite a los usuarios poder colaborar en un proyecto de software.
- Se ha hecho tan popular que existen imágenes de Docker certificadas de tal forma que se puede descargar y utilizar en localmente.
- Posee una característica llamada **CI** que permite crear y hacer uso de pipelines para actualizar el código de nuestro proyecto.

Más información en:  
<https://about.gitlab.com>



### BITBUCKET

- Repositorio que permite colaborar en proyectos de software. Es muy similar a los anteriores, pero sin opción free.
- Al ser un producto de Atlassian, se integra muy bien JIRA y todos los productos de la misma línea. No necesita de procesos complejos de integración.
- Dispone de **boards** para hacer seguimiento sobre “issues” creados.

Más información en:  
<https://bitbucket.org>

## ¿Qué

### tecnologías existen?

Todos estos productos disponen de **flujos de trabajo** altamente configurables según las necesidades del usuario. Pero, ¿qué son los flujos de trabajo? ¿De qué se tratan? Los flujos de trabajo dan cuenta de las características innatas o facilidades que estos sistemas nos ofrecen:

- Almacenamiento de código
- Versionado
- CI/CD
- Control de cambios
- Colaboración y revisión

### ¿Qué es Jenkins y para qué sirve?

**Jenkins** es un servidor de automatización de código abierto escrito en Java capaz de organizar una cadena de acciones que ayudan a lograr el proceso de integración continua —y mucho más!— de manera automatizada. ¿Para qué lo usamos? La clave está en una palabra: **automatizar**. Jenkins nos ayuda a automatizar procesos de desarrollo ¡para que te puedas concentrar en escribir el código de tu aplicación! Algunos de sus usos más populares:

- Ejecutar builds.
- Correr tests para detectar bugs u otros problemas antes de que puedan llegar al usuario.
- Hacer análisis estático de código.
- Desplegar a distintos ambientes.

Y todo esto de manera automática, ahorrando tiempo en tareas repetitivas y optimizando el proceso de desarrollo.

### ¿Qué es Jenkins?

Jenkins es un servidor diseñado para la integración continua (CI). Es gratuito y de código abierto (open source). ¡Y se ha convertido en el software más utilizado para esta tarea!

**¿Qué podemos hacer en Jenkins?** Nos permite organizar una cadena de acciones que ayudan a lograr el proceso de integración continua (¡mucho más!) de manera automatizada.

### **Algunas de sus características**

- **Automatización** Es el software de automatización más usado para la integración continua
- **Comunidad** La comunidad ha desarrollado más de 15000 plugins para agilizar las tareas de los equipos de desarrollo.
- **Tareas** Jenkins puede orquestar cualquier tipo de proceso y ejecutar tareas manuales, periódicas o automáticas.
- **Fácil de usar** Su uso es sencillo y puede aumentar su capacidad de cómputo añadiendo nuevos agentes o servidores

### **Configuración de Jenkins**

**En la sección de configuración podemos personalizar varias opciones**

**SMTP y otros canales** Podemos configurar a Jenkins para que se comuniquen con nosotros por mail, cargando la información de nuestro SMTP o WebHooks para usar canales como Slack.

**Variables de entorno** En nuestra instancia de Jenkins podemos guardar información sensible (por ejemplo, los datos de conexión a nuestros servidores como par de llaves para ssh).

**¡Seguridad primero!** Podemos configurar los roles y permisos de nuestro Jenkins para limitar el acceso de usuarios. Es importante que cada usuario tenga los permisos mínimos para realizar sus tareas.

### **Múltiples nodos**

Jenkins puede distribuir el trabajo en varias máquinas, lo que ayuda a realizar las compilaciones, pruebas e implementaciones en varias plataformas con mayor rapidez.



### **Plugins**

Con cientos de complementos en el Centro de Actualización, Jenkins se integra con prácticamente todas las herramientas en la cadena de la integración y entrega continua. Algunos ejemplos son...

- -Docker
- -Pipelines
- -Slack notifications
- -Azure

## Jenkinsfile

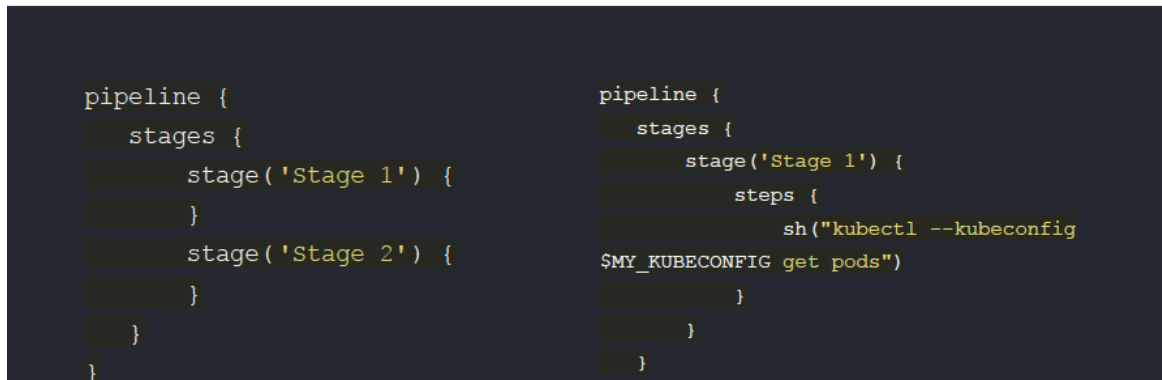
Al crear un archivo llamado Jenkinsfile podemos definir nuestro pipeline a través de código escrito en Groovy (un derivado de Java).

Esto nos brinda algunos beneficios:

- Extender el uso de los plugins.
- Personalizar algunos usos que no se encuentran en plugins.
- Que nuestro pipeline tenga la misma lógica que una aplicación a través de estructuras de control.

### Jenkinsfile: Dividimos en pasos

Los pipelines se encuentran divididos en distintos pasos que se encadenan entre sí para obtener el resultado esperado al final del pipeline. Los llamamos Stages



```
pipeline {
  stages {
    stage('Stage 1') {
    }
    stage('Stage 2') {
    }
  }
}
```

```
pipeline {
  stages {
    stage('Stage 1') {
      steps {
        sh("kubectl --kubeconfig
$MY_KUBECONFIG get pods")
      }
    }
  }
}
```

## Video

Un **jenkisfile** es un archivo que contiene las reglas o pasos así ejecutados en el pipeline este puede ser declarativos o con script

**Veamos el declarativo** un jenkinsfile declarativo está compuesto por varios bloques el pipeline es el bloque principal donde reciben los restos los bloques el bloque stages engloba un subconjunto de tareas que se realizan a través de todo el ciclo del pipeline

- -construir
- -probar
- -desplegar

Es utilizado por varios plugins para visualizar o Mostrar el estado y progreso del proceso cada una de las tareas que componen una etapa se denominan steps, básicamente cada paso le dice a jenkins que hacer en cada uno de los puntos concretos del ciclo al realizar

Veamos un jenkinsfile simple aquí podemos ver todos los bloques mencionados dentro del bloque pipeline se encuentran todos los subbloques y además una línea agent any que hace referencia a la gente slat en que se quiere ejecutar ese pipeline luego tenemos el bloque stage que contiene las distintas etapas del pipeline en este caso hay tres etapas bien definidas

- -"Build"
- -"test"
- -"deploy"

dentro de cada uno de estos bloques están las acciones a llevar a cabo en este ejemplo solo colocamos un texto en la salida pero en el bloque stage podemos ejecutar comandos usando sh Comando o incluso hacer uso de distintos pipelines esto se implementa en los pipeline

jenkis es alguien que hace lo que se le solicita en este caso lleva la ejecución del pipeline

**software** : construye nuestro software lo testea y lo **despliega**, pero para poder hacer todo esto jenkins necesita saber qué es exactamente lo que queremos que haga es ahí donde entró juego El jenkinsfile este archivo es una guía que jenkins tiene que seguir para que todo salga como queremos por ejemplo en esta guía podríamos solicitar que

contruya el software para obtener un ejecutable para Windows y para Linux y luego que se ejecuten los test para verificar que el software hace todo lo necesario para su correcto funcionamiento y además que no tenga errores a continuación si los testeos son exitosos queremos que suba los ejecutables a un repositorio para su almacenamiento y finalmente deseamos que ese ejecutable sea descargado en el servidor correspondiente y se ejecute teniendo así una nueva versión del Software funcionando y todo esto de manera automática si viene hemos usado en Jenkinsfile declarativo debemos tener en cuenta que no es la única forma que también existen los Jenkinsfile scripted estos ofrecen un mayor control y mayor poder a la hora de indicar los pasos de un pipeline sin embargo tiene la desventaja de que están en un lenguaje de programación poco utilizado y además suele ser más complejo con una curva de aprendizaje mayor a la de los Jenkinsfile declarativos.

Cabe destacar también que estos archivos están siempre con el código fuente del Software lo que ofrece todas las ventajas del uso del controlador de versiones como la colaboración entre el equipo, transparencia, versionado, etc. Según los principios de DevOps, tenemos que eliminar la dependencia de las personas que desarrollan hacia las personas encargadas de operaciones; esto permite que quienes desarrollan puedan desplegar sus aplicaciones de forma automática y sin intervenciones de sysadmins, lo que agiliza el proceso de desarrollo de software.

---

## Clase 14 Pipelines: build y continuous integration

### **El proceso de build**

El **pipeline** es un componente fundamental en el desarrollo de software automatizado. Si bien el término se ha utilizado para describir muchos aspectos diferentes de la informática, en la industria de DevOps usamos pipelines para ilustrar las amplias aplicaciones de comportamientos y procesos involucrados en integración continua.

Pensemos entonces en los pipelines y sus procesos. Podemos reconocer distintas etapas que conforman un pipeline. Vamos a dedicarnos a una de ellas: el proceso de **build** o construcción de una aplicación. Es decir, la transformación del código fuente en una aplicación funcional para ser ejecutada por usuarios.

Comencemos el recorrido por este proceso. Veremos cómo se organiza, la manera en que se compone, y cómo se integra con Jenkins como herramienta de integración continua.

### **Video 03:45**

Vamos a hablar del proceso de compilación: build, de una aplicación utilizando como herramienta de automatización Jenkins.

Cuando hablamos de **build** = Construcción = compilación

Es decir, la transformación de un código fuente a través de una aplicación funcional para ser ejecutadas por usuarios.

Lenguajes interpretados: El código fuente es interpretado tal cual como se da la ejecución del programa.

Lenguajes compilables: Tienen como condición necesaria tener una etapa de compilación de la aplicación para ser utilizadas, algunos ejemplos son: C, C#, C++ y Java

Para que una aplicación Java pueda ser utilizada, el archivo Java debe transformarse en un archivo ejecutable .class

En este proceso el archivo no pierde su nombre, solo se modifica la extensión.

Por qué se compilan los programas: La compilación es necesaria para que la máquina comprenda nuestro código, para esto el programa que estamos desarrollando se transforma en: Bytecode: un lenguaje de máquina que el procesador de la computadora puede comprender y ejecutar.

Archivo compilado puede variar según el lenguaje y la finalidad de la aplicación

Este proceso de Build es automatizable dentro de la metodología DEVOPS.

## Etapas - proceso de Build:

1. **Entrada:** Etapa de codificación: Cuando el desarrollador está escribiendo el programa.
2. **Proceso:** Compilación
3. **Salida:** Entregable para las pruebas

Con Jenkins tenemos la oportunidad de hacer todo el ciclo devops de 8 etapas.

Para centrarnos en la etapa de compilación de una aplicación java debemos hacer lo siguiente:

1. Crear una tarea de estilo libre
2. La tenemos que nombrar de la forma más descriptiva posible.
3. Dentro de un grupo de estilo libre vamos a escribir una breve descripción.
4. Luego elegimos el tipo de ejecución que necesitamos para nuestro proyecto
5. Hacemos clic en guardar, ya tendríamos nuestro proyecto configurado, ahora podemos ejecutarlo cada vez que se requiera con la opción: Construir ahora
6. Al observar la salida vemos que está marcada como exitosa y vemos lo que escribimos en nuestro código y el script que utilizamos para ejecutar esta tarea.

## **Continuous integration**

¿Qué es la integración continua? ¿Y si la pensamos en acción? ¡Vamos a intentarlo en este breve relato!

Pensemos en una aplicación cuyo código está almacenado en un repositorio remoto. Los desarrolladores suben cambios al código todos los días, varias veces al día. Por cada envío al repositorio se ejecutan un conjunto de pasos automáticos que terminan construyendo una aplicación.

Estos pasos no solo arman la aplicación, ¡también la testean! Entonces...

La **integración continua (CI)** es una práctica de desarrollo que requiere que los desarrolladores integren código en un repositorio compartido varias veces al día. Al integrar contenido con cierta frecuencia, detectamos errores rápidamente y los localizamos de manera sencilla.





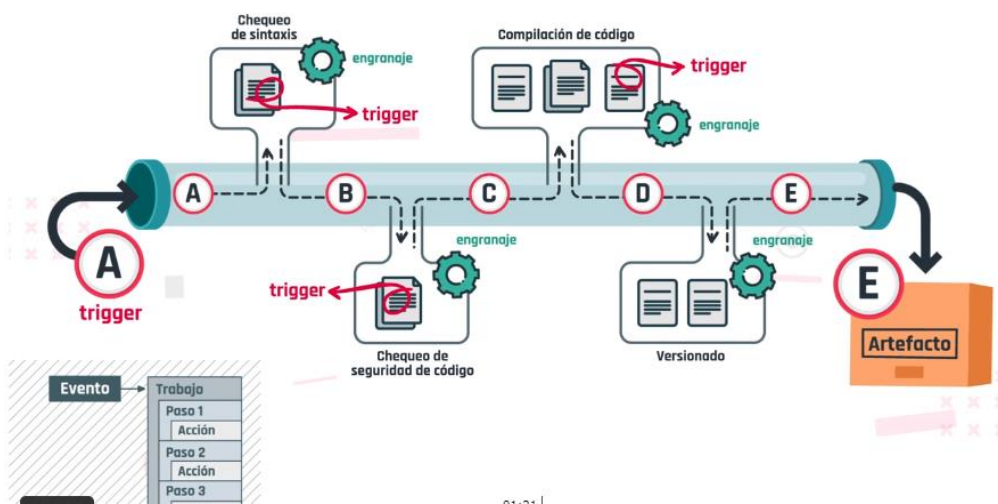
## ¿Qué son los triggers?

Un trigger es un “disparador”: algo que hace que una primera cosa active una segunda y así sucesivamente. Los triggers son parte de la familia de la “integración continua” o “CI” y se utilizan para ejecutar un pipeline mediante una llamada de una API u otro proceso en línea. Son sumamente útiles para concatenar procesos que, en líneas generales, constituirán un flujo de trabajo que servirá para construir algo. Veamos algunos ejemplos en este video.

### Video 02:15

Un Trigger es un disparador de eventos, un evento es la aparición de un estímulo que puede dar comienzo a una serie de estados posteriores, un objeto pasara de un estado a otro dentro de nuestro pipeline.

Un trigger se va utilizar para iniciar un pipeline de forma automática, y va a decidir que código ejecutar cuando haya un evento específico, en general cuando hablamos de procesos por eventos hay un bucle principal que escucha los nuevos eventos entrantes disparados por el drif, de esta forma se concatenan los procesos.



De acuerdo a como este configurado el pipeline puede haber diferentes **TIPOS DE triggers:**

1. Basados en CI: Integración continua: Estos hacen que se ejecute un pipeline cada vez que se envía una actualización a un branch específico o cuando se envía etiquetas específicas.
2. Basados en pipelines: Disparan un pipeline habiendo completado un pipeline anterior
3. Scheduled: Disparan un pipeline de acuerdo a un evento programado
4. Externos: Refieren a la integración con diferentes productos, generalmente mediante el uso de APIs.

## ¿Qué es un artefacto?

Varias veces hemos tenido que compilar código fuente para obtener un producto utilizable. Esos archivos resultantes del proceso de compilación llevan un nombre puntual: ¡Artefactos!

**Los artefactos son el resultado de la compilación —la transformación— de un código fuente en un archivo binario no modificable.**

Navegando entre artefactos:

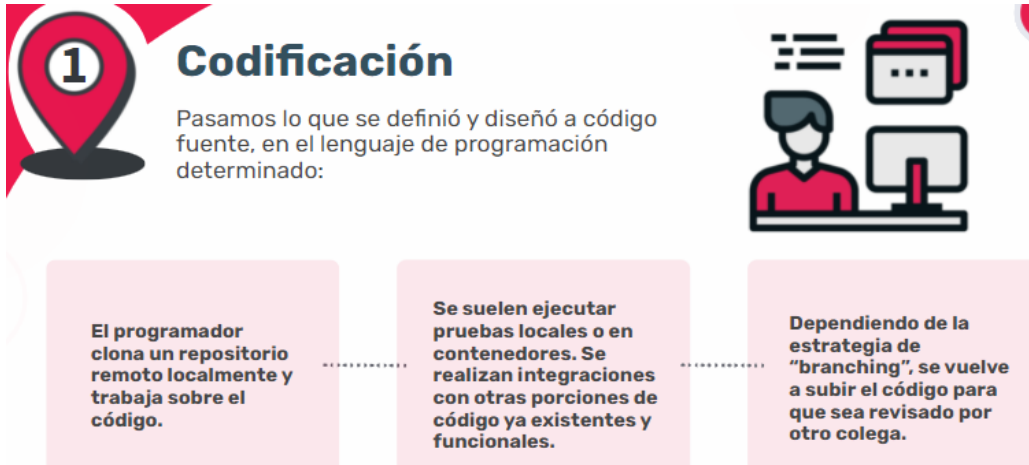
Las aplicaciones suelen estar fragmentadas en distintos componentes individuales que luego se pueden ensamblar hacia un producto completo. Esto generalmente sucede durante la fase de compilación, cuando son creados fragmentos más pequeños de la misma. Esto suele hacerse para tener un uso más eficiente de los recursos, reducir los tiempos de compilación, hacer un mejor seguimiento para la depuración binaria, etc. El tipo de artefacto o artifact va a depender del producto con el que se programe la aplicación. Veamos el tipo de artifact por producto:

- Java: jar, ear, war, etc.
- .Net: dll, exe, etc.
- Python: .py, sdist.

- 1) También pueden existir tipos de artefactos no relacionados con el producto específico, pero necesarios para el funcionamiento. Por ejemplo archivos, YAML, XML, TXT, MD, lib, so, bin, etc. ¡Un momento! Para complejizar un poco más el panorama, un artefacto también puede ser un script, un diagrama, un modelo de datos, etc. Pero entonces, ¿un artifact es cualquier tipo de archivo? En términos estrictos, es todo aquello que resulta de un proceso de build. Administración Un producto de repositorios de artefactos nos permite básicamente efectuar las siguientes operaciones: - mover - copiar - eliminar ... artefactos para mantener la consistencia en cada repo. Cuando un artefacto se mueve, copia o elimina, el producto debe actualizar automáticamente los llamados “descriptores de metadatos”. Ejemplo de ello pueden ser: maven-metadata.xml, RubyGems, Npm, etc. Metadata ¡Los metadatos son datos acerca de los datos! Cada artefacto contendrá metadatos que son esenciales para la reutilización de código. Una característica existente es la de permitir a los desarrolladores compartir código y utilizar componentes de terceros. En este sentido, los metadatos cumplen un rol clave en la colaboración. ¿Por qué?
- 2) Por ejemplo, al chequear los datos asociados a cada artefacto podemos ver si fueron alterados. Esto será de utilidad para validar su descripción, por ejemplo, la versión de un producto. Podremos saber -acerca de ese cambio- quien lo hizo, cuándo, a qué hora exactamente, qué dependencias tiene, etc. Sin embargo, la combinación de muchos tipos de metadatos puede llegar a complejizar el proceso de colaboración. ¡Una buena estrategia inicial es vital para evitar caer en este tipo de situaciones! Almacenamiento ¿Dónde se almacena todo este código y sus respectivos artefactos? En este punto está claro que el código suele almacenarse en sistemas de control de versiones como GitHub, GitLab o BitBucket. Pero este no es el caso para los artefactos que pueden -por ejemplo- ser archivos binarios. Y quizás no tiene mucho sentido utilizar un repositorio de proyectos para almacenar archivos binarios que son ilegibles para el ser humano y pueden ser bastante grandes en tamaño. Es aquí donde los repositorios de binarios son una parte tan vital del proceso de integración continua. El repositorio binario puede permitir alojar todo esto en un solo lugar haciendo que su administración sea más simple. Ejemplo de productos típicos que podemos encontrar en el mercado hoy día son: Artifactory, Nexus, Harbor, etc. También vamos a encontrar aquellos que son basados en Cloud: Azure Artifact, Artifact Registry, etc. Accediendo a nuestros artefactos Como todo producto de software, su sintaxis variará pero su propiedad será la misma: ofrecer una forma sencilla de realizar consultas que especifique un criterio de búsqueda, filtros, opciones de clasificación y parámetros de salida.
- 3) Esto se logra mediante la exposición de una API RESTful, ya que siendo objetos que deban utilizarse de inmediato para proporcionar datos de salida, el tiempo de acceso y respuesta debe ser extremadamente rápido y con bajo consumo de memoria. Todo alrededor de una filosofía La importancia de un repo de artefactos se puede entender en relación con la filosofía DevOps: “desarrollar mejores aplicaciones, más rápido y con constantes entregas de funciones o productos de software”. Una práctica común en integración continua es la de construir el binario una sola vez, subirlo a un repositorio de binarios y luego llamarlo desde allí para implementarlo en los diferentes entornos. De esa manera nos aseguramos de que el código base que ya funciona en el ambiente de desarrollo es la misma base que se introdujo en producción.

#### 4 pasos para el desarrollo de software:

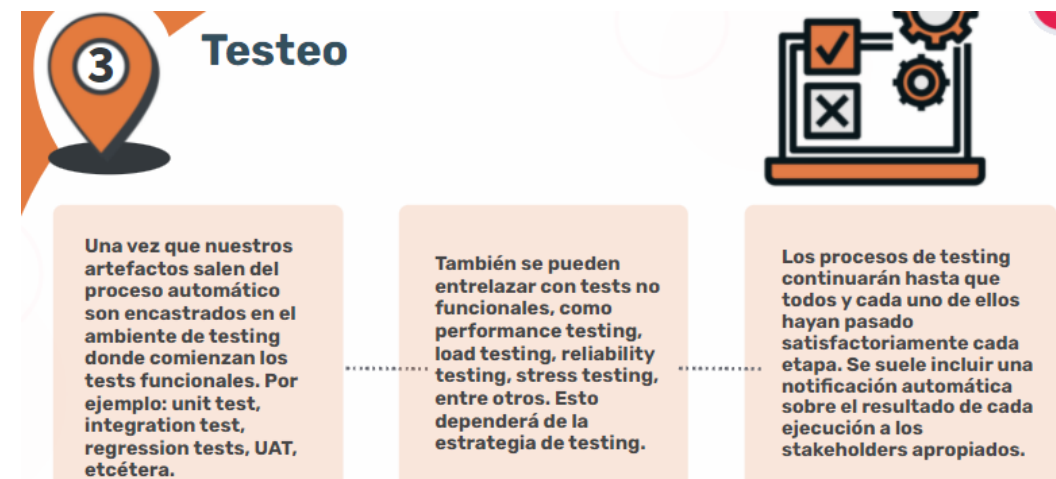
1. .



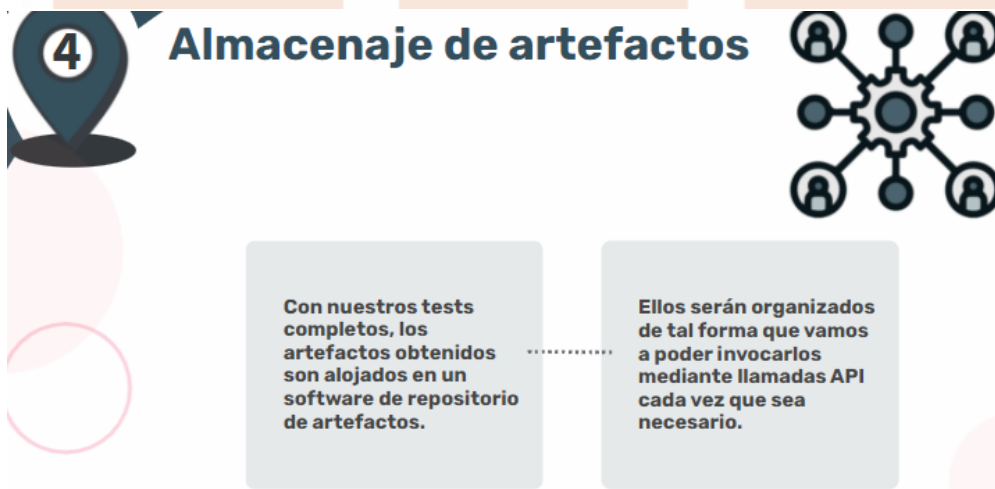
2. .



3. .



4. .



## Bonus: El principio de inmutabilidad

Inmutabilidad: “sin mutaciones”, “sin cambios”. En el sentido de DevOps, significa que una vez que creamos un artefacto —ya sea una imagen de contenedor o un paquete de código compilado— no debería ser necesario modificarlo. Y si se requieren cambios, se creará una nueva versión del objeto.

**¿Por qué nos resulta útil que un objeto o artefacto sea inmutable?** Porque cuando lo copiamos —por ejemplo, desde un entorno de desarrollo a la producción— ya sabemos cómo se va a comportar. No solo el artefacto tiene que ser inmutable para que funcione bien en todos los entornos, o ambientes, sino también la infraestructura. ¡Veámoslo en un ejemplo

## Repaso de la semana

- **Integración continua:** Es una práctica común en el desarrollo de software. Los desarrolladores fusionan regularmente cambios de código en un repositorio central para ejecutar pruebas y compilaciones automatizadas. Facilita los ciclos de retroalimentación entre equipos de operaciones y desarrollo para que se puedan implementar iteraciones de actualizaciones más rápidamente en las aplicaciones en producción.
- **Pipeline:** Conjunto de prácticas que implementan los equipos de desarrollo (Dev) y operaciones (Ops) para crear, probar e implementar software de forma más rápida y sencilla. El flujo de trabajo contiene fases que impulsan el desarrollo continuo, integración, pruebas y posterior retroalimentación para comenzar de nuevo.
- **Automatización:** Utilización de tecnología que reduce el uso de asistencia humana sobre los procesos haciéndolos más confiables, rápidos, seguros y escalables.
- **Testing:** Proceso organizativo dentro del desarrollo de software en el que se verifica la corrección, calidad y rendimiento del software crítico para el negocio.
- **Build:** Proceso que convierte archivos con código en un producto de software en su forma final o consumible.
- **Triggers:** Es ese “algo” que activa la ejecución de otro proceso. Un desencadenante para lograr que algo se ponga en movimiento.
- **Artefactos:** Es ese “algo” que activa la ejecución de otro proceso. Un desencadenante para lograr que algo se ponga en movimiento.

---

## Módulo 3 - Clase 15

**Pipelines** Conjunto de prácticas que implementan los equipos de desarrollo (Dev) y operaciones (Ops) para crear, probar e implementar software de forma más rápida y sencilla. El flujo de trabajo contiene fases que impulsan el desarrollo continuo, integración, pruebas y posterior retroalimentación para comenzar de nuevo

**Automatización** Utilización de tecnología que reduce el uso de asistencia humana sobre los procesos haciéndolos más confiables, rápidos, seguros y escalables.

**Testing** Proceso organizativo dentro del desarrollo de software en el que se verifica la corrección, calidad y rendimiento del software crítico para el negocio.

**Build:** Proceso que convierte archivos con código en un producto de software en su forma final o consumible.

**Triggers** Es ese “algo” que activa la ejecución de otro proceso. Un desencadenante para lograr que algo se ponga en movimiento.

**Artefactos** Es uno de los muchos tipos de subproductos tangibles que se producen durante el desarrollo de software.

**Pipelines:** Los pipelines en Jenkins proporcionan un conjunto de herramientas que son flexibles a través de código. Esto nos permite modelar una implementación de la metodología DevOps muy simple o para una organización grande y compleja.

---

## Clase 16 - Pipelines reales y continuos delivery

### El Despliegue Continuo (CD)

En el **Módulo III** iniciamos nuestro recorrido en el mundo de los **pipelines** y aprendimos qué es la **Integración Continua (CI)** y las etapas que involucra. Pero el desarrollo de un producto o artefacto no termina allí, sino que ¡recién inicia!

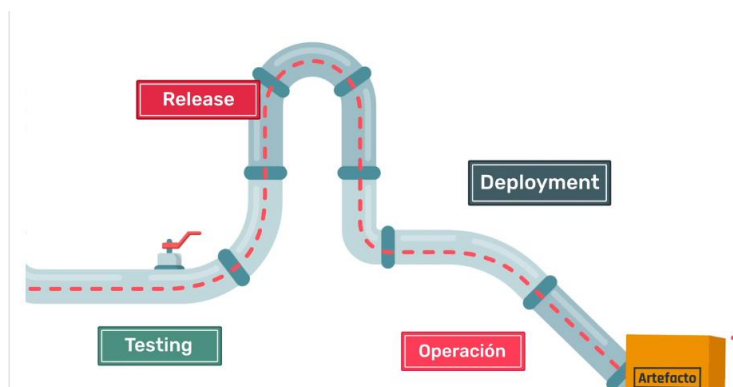
En esta clase verás cómo tu producto, recién salido de la fábrica, queda disponible para su consumo. Pero lograrlo no es tan sencillo, ¡involucra varias etapas que te permitirán comprobar la calidad del desarrollo y asegurar su correcta entrega! El **Despliegue Continuo (CD)** es ese segmento del pipeline que se encarga de llevar un producto ya construido a un ambiente productivo: ahí donde se utilizará.

Una vez terminada la elaboración del producto, es decir la culminación del proceso CI Integración continua donde ya tenemos la aplicación compilada, en donde comienza la distribución para su consumo pero antes debe pasar por una serie de etapas que comprueba su calidad y asegura su correcta entrega esto es lo que se conoce como CD o despliegue continuo.

**CD:** hace referencia al segmento de pipeline final que se encarga de llevar un producto ya construido a un ambiente productivo, este proceso es continuo, porque al estar en un pipeline automatizado nos permite realizar esta tarea de forma continua siempre que sea necesario.

El proceso entero se compone de las siguientes etapas:

1. **Testing:** Donde se realiza pruebas funcionales de integración, seguridad, performance y carga que comprueban el completo funcionamiento y calidad del producto.
2. **Release:** En ella se lleva a cabo la creación, la calificación y subida de guardado en una bodega de paquetes lo que llamamos repositorio donde quedará disponible para su despliegue en el entorno que se requiera. En este paquete además de nuestra aplicación hay otra data como archivos de configuración, programa de iteración y documentación.
3. **Deployment:** Acá se mueve el producto, se realiza el despliegue, su instalación y se pone en funcionamiento y queda listo para que pueda ser usado por los usuarios.
4. **Operación:** En esta se utiliza el producto, aquí se debe preparar el entorno para despliegue y configurar servidores entre otras



El despliegue continuo nos permite entregar y garantizado en el menos tiempo posible cumpliendo con todos los requisitos de las personas usuarias.

### ¡Una diferencia clave!

Los procesos de continuous delivery (o entrega continua) y continuous deployment (despliegue continuo) siguen las mismas etapas, ¡pero con una pequeña gran diferencia!

La **entrega** continua se focaliza en la automatización de los pasos para que nuestro software esté disponible para ser aplicado en los ambientes productivos en cualquier momento, ¡pero no hace la implementación automática en producción!

**El despliegue va un paso más allá.** En esta práctica, todo es automático en los ambientes, ¡también en producción! Buscamos que no exista intervención humana en ninguno de los procesos de trabajo.

Esta es la diferencia más importante con respecto a la entrega continua.

Para lograrlo, el pipeline de producción tiene una serie de pasos que deben ejecutarse en orden y forma satisfactoria ¡y de manera automática! Si alguno de esos pasos no finaliza de forma esperada, el proceso de despliegue no se llevará a cabo.



---

## Clase 17 - Pipelines: End-to-End

### Pipelines: CI / CD

¿Pipelines End-to-End? ¿Pensaste a qué refiere este término? Llegamos a la última clase que nos presenta contenidos del Módulo III. Un encuentro que se trata también de los cierres: vamos a experimentar como finalizan —y vuelven siempre a iniciar— nuestros pipelines. ¡Vamos a trabajar algunos conceptos y definiciones que nos permitirán comprender de manera completa los procesos de un pipeline!

¿Recapitulamos? CI/CD es un método para distribuir aplicaciones de software con cierta frecuencia mediante el uso de procesos automatizados. ¡Pero no es solo eso! Es también una forma eficaz de promover la cultura ágil dentro de un equipo de desarrollo y operaciones.

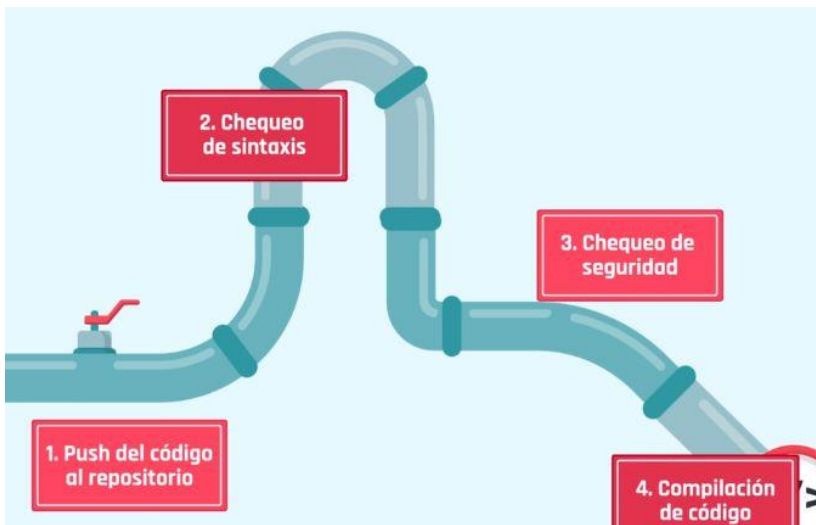
El objetivo principal de CI/CD es reducir el tiempo de comercialización que, de otro modo, implicaría procesos largos y una colaboración mínima entre el desarrollo y el área de operaciones. ¡De aquí surge el concepto de DevOps! ¡Veamos el proceso completo en el siguiente video!

Video 02:56

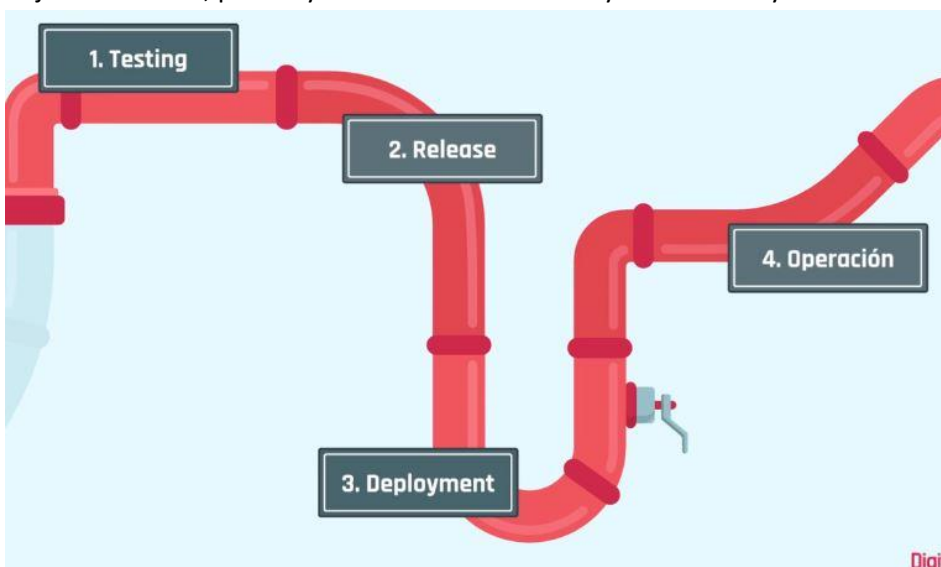
## Pipelines: CI + CD

Imaginemos que tenemos una fábrica que produce un bien físico. Necesita materia prima, ciertas máquinas, personal que las controle y procesos que definan de forma lógica la secuencia de pasos que se deben ejecutar para liberar el producto que finalmente saldrá de la fábrica y llegará a algún mercado para su consumo. Una vez terminada la elaboración del producto comienza su distribución, pero antes, debe pasar por una serie de etapas que comprueben su calidad y se aseguren de la correcta entrega a los usuarios que van a usar dicho producto. Todo este proceso automatizado de principio a fin dentro de la elaboración de un producto es la unificación de dos conjuntos de pipelines que garantizan su calidad y tiempo de entrega desde la creación del código hasta su operación: CI (integración continua) y CD (despliegue continuo).

Iniciemos con CI o integración continua. Que no es mas que la práctica de fusionar o mergear copias de trabajo de los proyectos de cada desarrollador en un proyecto principal. La idea es realizar integraciones de código constantemente y quizás varias veces al día dependiendo del enfoque estratégico de la compañía. Por cada integración pueden existir uno o mas testeos para detectar posibles errores lo más rápido posible.



Luego seguimos con CD o entrega/despliegue continua que es el eslabón que le sigue a CI. Este es un enfoque o práctica en ingeniería de software en el que los equipos producen software en ciclos cortos. Esto va a garantizar que el software se pueda lanzar o entregar de manera confiable en cualquier momento y de forma automatizada. Su objetivo es crear, probar y lanzar software con mayor velocidad y frecuencia.



Si bien el proceso CI y CD finaliza, al estar operativo el producto en realidad el ciclo vuelve a comenzar. El uso de CI y CD nos aporta los siguientes beneficios:

- Los cambios de código son más pequeños y más simples, más atómicos y tienen menos consecuencias no deseadas.



- Detectar fallas entonces se hace mas simple y rápido. Reducción de errores.
- El tiempo medio de resolución MTTR es más corto justamente debido a cambios de códigos mas pequeños.
- Estos cambios mas pequeños permiten pruebas positivas y negativas más precisas.
- El tiempo transcurrido para detectar y corregir los escapes de producción es más corto.
- Con una tasa de liberación mucho más rápida.
- Entonces el producto mejora a través de la rápida introducción de funciones y una respuesta a los cambios de funciones más veloz.
- Los ciclos de lanzamiento a producción son más cortos.
- La participación y los comentarios del usuario final durante el desarrollo y el testeo conducen a mejoras en la usabilidad del producto final. Retroalimentación constante con el usuario

Hoy en día es común ver a las áreas de operaciones y de desarrollo trabajar en conjunto en el desarrollo de un producto de software, que es lo que hoy se conoce como practicas de DevOps. En este sentido, CI y CD son laos pilares fundamentales de estas practicas que trabajan sobre los principios de integración continua, entrega continua y desarrollo a través de varias pruebas automatizadas que garantizan un producto de calidad y su correcta entrega.

Un pipeline consiste en diversos pasos o etapas pero, ¿tenemos realmente en claro en qué orden va cada uno? ¡Vamos a averiguarlo!

## ¡A ordenar!

En el siguiente juego, vamos a organizar cada etapa en el orden correcto siguiendo el ciclo de desarrollo que vimos usando pipelines en CI/CD.



### Combinamos deployments

Deployment es... ¿implantar o implementar? ¿Cuáles son las formas de realizar deploy? ¿Cómo integramos laC dentro del ciclo DevOps de nuestra aplicación?

En este video vamos a conocer el ciclo de vida de una aplicación desde su planificación hasta su despliegue y operación, teniendo presente qué herramientas se utilizan para cada una de las etapas. Además, aprenderemos los tipos de deployments más utilizados en el mercado y cómo agregamos infraestructura como código en este ciclo.

**video 04:49**

### **Deployments / Desde la Infraestructura como Código al deploy de la aplicación**

En términos generales, la palabra Deploy es utilizada para describir que algo fue colocado en su posición cuando un sistema es habilitado para su uso, ya sea en un

- Ambiente de desarrollo
- Pruebas
- Producción



Pensemos por ejemplo en el caso de haber creado un sitio web en nuestra computadora y lo dejamos estático, sin publicar. En el momento en que incorporamos la pagina a un servidor Web Hosting, ese proceso será considerado Deploy. Otro ejemplo, es cuando se genere una nueva versión de un programa y el desarrollador quiere implantarlo en un servidor de Aplicaciones, es decir, subir los archivos actualizados a un servidor web que puede ser un ambiente de prueba o de producción. Esto también significa hacer un Deploy. Ahora bien, no debemos confundir el termino Implantar con Implementar, estas son dos cosas distintas. Cuando decimos implantar hacemos referencia a iniciar algo, mientras que implementar es el acto de poner algo en práctica. Si lo aplicamos al mundo de la programación cuando se inicia el desarrollo de un sistema este esta siendo implantado, en el momento en el que el proyecto comienza a ser usado por los usuarios podemos decir que ha sido implementado.

¿Cuáles son las formas de realizar Deploy? Hoy existen 3 formas de realizarlo:

- 1) De forma Manual: por ejemplo, el protocolo de transferencia de archivo conocidos como FTP (File Transfer Protocol) Se trata de un tipo de conexión que permite que dos computadoras con acceso a Internet intercambien archivos. Al realizar de forma Manual, se necesita una persona ejecutando esta transferencia.
- 2) De forma Parcialmente automatizada: un ejemplo puede ser el push de BranchMaster que se realiza en el repositorio Git el cual opera un pequeño Trigger y actualiza el servidor de Web Hosting. Aunque necesite algunos comandos el proceso ocurre de manera automática. La ventaja es el Control de la Versión y el estado de cada Deploy.
- 3) Completamente automatizada: Es la tendencia más moderna en el desarrollo web. Esta no solamente copia los cambios de forma automática en el servidor, sino que también está íntimamente conectada con el concepto de Integración Continua (CI). La herramienta de Deploy en este caso realiza todas las pruebas necesarias para que no existan problemas a la hora de juntar las integraciones de la producción.
  - Actualizar bibliotecas
  - Probar la conectividad de los servidores
  - Simular visitas y entradas de datos.
  - Una de las herramientas mas utilizadas para el Deploy automatizado es Jenkins. Entre sus beneficios se destacan:
    - Alto nivel de productividad
    - Seguridad que nos brinda
    - Calidad en el desarrollo de software.

Existen también 3 estrategias muy simples que pueden ser implementadas y que permiten hacer Deploy en el día a día:

- Rolling → consiste en subir los servicios con la nueva versión del código. Pudiendo coexistir o no con la versión antigua.
- Blue-Green → Se caracteriza por tener dos ambientes idénticos conocidos como mirror, que tienen un Load Balancer (balanceador de carga) permitiendo así redireccionar el trafico para el ambiente deseado. El beneficio de esta estrategia es que nos permite subir una nueva versión de la aplicación que esta en producción mientras que la versión actual, Blue, solo recibe las solicitudes. De esta forma tan pronto como sean realizadas las pruebas en la nueva versión, Green, es posible realizar otras solicitudes que apunten hacia esta.
- Canary → Se trata de la estrategia mas compleja. Consiste en colocar la nueva versión en producción para una pequeña parte de los usuarios, es posible, por ejemplo, liberar una función solamente para el público de sexo femenino menor a 30 años, esto nos permite probar la versión antes de liberar el acceso total.

Ahora, ¿Cómo integramos i a c (¿??) dentro del ciclo de DevOps de nuestra aplicación?

Los equipos de infraestructura utilizan Pipelines para infraestructura como código por las mismas razones que los equipos de desarrollo usan Pipelines para sus códigos de aplicación, ya que garantizan que cada cambio se haya aplicado a cada entorno y que las pruebas automatizadas hayan pasado.

Con este enfoque las nuevas instancias de entorno pueden activarse bajo pedido, lo cual tiene varios beneficios. Los desarrolladores pueden crear sus propias instancias de Sandbox para que puedan implementar y probar aplicaciones basadas en infraestructura en la nube o trabajar en los cambios de las definiciones de entorno sin entrar en conflicto con nosotros mismos y los del equipo.

Los cambios y las implementaciones se pueden manejar con un enfoque Blue-Green crean una nueva instancia, la prueba, luego se intercambia el tráfico y se elimina la instancia anterior.

Quienes prueban, revisan y otros miembros del equipo pueden crear entornos según necesario eliminándolos cuando no se utilizan. De esta manera tenemos integrado la gestión de la infraestructura como código dentro de nuestro ciclo DevOps donde Jenkins administra las tareas que, para este caso, cierran el Deploy y la operación como etapas del ciclo. Jenkins interactúa con ambas herramientas, para que cada una realice la tarea necesaria. Dichas herramientas devuelven una respuesta si la tarea finalizó correctamente o no, y según lo que tenga definido en el pipeline continua o lo finaliza. Es importante entonces conocer el ciclo de vida de una aplicación desde su planificación hasta su despliegue y operación teniendo presente cuales herramientas se utilizan para cada una de las etapas.

### **Cuestionario**

1. La diferencia principal entre CI y CD radica en:
  - CI toma el producto final y lo hace disponible a la fase de testing. CD continúa el proceso y lo hace disponible a la fase de release.
2. ¿Qué problemas viene a resolver un software de administración de artefactos?
  - → Centraliza los archivos para ser consumidos por mi aplicación en tiempo real. Esta respuesta es correcta. Mediante el uso de pipelines, puedo integrar el software administrador de artefactos para que almacene de forma centralizada mis artefactos y luego poder ser utilizados mediante llamadas en tiempo real.
3. El testeo de carga de mi aplicación arroja resultados negativos enviando correos a todos los involucrados y deteniendo el ciclo de continuous delivery. Aún con estos resultados nuestro project manager solicita que el producto sea deployado en producción, dado que nuestro cliente lo necesita ya mismo. Bajo esta situación ¿cuál debería ser nuestra recomendación?
  - → No proceder, ya que si bien podemos deployar nuestro software el mismo puede comenzar a operar de forma correcta hasta cierto punto en el que comenzará a fallar o a operar de forma limitada.
  - Esta respuesta es correcta. Si nuestros testeos de carga nos arrojan resultados negativos, lo aconsejable es no deployar y volver a definir las características de carga que esa infraestructura deberá soportar antes de montar software y hacerlo productivo.
4. ¿En cuál de las prácticas mencionadas debajo podemos incluir una etapa de testeo? →
  - Continuous integration
  - Continuous deployment
  - Continuous deliveryTodas las anteriores → Esta respuesta es correcta. El testing puede ubicarse en cualquiera de las etapas. En CI, encontraremos testeos de integración, por ejemplo! En continuous delivery, encontraremos testeos de carga, en continuous deployment, tests unitarios, etcétera.
5. Distinguí los beneficios de CI / CD. → Resolución rápida de defectos de software mediante la colaboración del equipo en los llamados “code reviews”.
  - Esta respuesta es correcta. Uno de los beneficios encontrados es la posibilidad de efectuar revisiones de código con colaboradores antes de que el producto sea compilado.

## ¿Cómo hacer un pipeline de CI / CD exitoso?

Tenemos nuestro pipeline configurado. ¡Genial! Pero ahora... ¿Podemos asegurarnos que sea un éxito? ¿Cómo? ¿Qué podemos hacer para reducir al máximo cualquier inconveniente o retroceso que pueda aparecer?

Construir un pipeline completo de un extremo a otro no es tarea sencilla, ya que involucra multitud de procesos y herramientas. Por eso es bueno que podamos generar y tener siempre presente estas **buenas prácticas** para el trabajo en DevOps.



## Repaso de la semana

- **CD:** Existen dos términos clave cuando hablamos de CD y nos sirven para revisar cómo se lanzan releases de software a un entorno de producción: la entrega continua (continuous delivery) y el despliegue continuo (continuous deployment).
- **Entrega continua:** Se produce software en ciclos cortos, lo que garantiza que se pueda lanzar de manera confiable en cualquier momento. Su objetivo es crear, probar y lanzar software con mayor velocidad y frecuencia.
- **Despliegue continuo:** El software se entrega con frecuencia en producción a través de implementaciones de pipelines totalmente automatizadas de inicio a fin.
- **El proceso de release:** Este proceso consta de una serie de acciones que nos llevan a tener disponible un paquete completo.
- En esta instancia es importante enfocarse en las múltiples [versiones](#) y en las etapas que lo componen.
- **Arquitectura recomendada sobre pipelines:** Con el objetivo de no “reinventar” la rueda, hemos visto algunas prácticas fuertemente testeadas en el mundo de DevOps que nos ayudan a arquitecturar un pipeline de forma eficiente:
  - Fallar rápido.
  - Tener etapas bien definidas.
  - Hacer pruebas fuertes.
  - Activar el Rollback.
  - Bajar la dependencia
  - Ejecutar fácilmente.
  - Contar con credenciales seguras

- **Pipelines end-to-end:** Hablamos del conjunto completo de procesos: CI+CD. Los pipelines abarcan flujos de trabajo que van coordinando las diversas etapas. Todo esto se define en archivos de configuración de proyecto e involucra triggers.



---

## Clase 19 – Monitoreo

Disponer de las prácticas de CI/CD y tener un ambiente totalmente automatizado no servirá de nada si no podemos ver y medir qué sucede en nuestro ambiente productivo.

Qué es el monitoreo y la diferencia entre monitorear infraestructura y monitorear aplicaciones.

**Monitoreo:** proceso que permite recopilar métricas sobre operaciones en aplicaciones o infraestructura para garantizar que todo funcione correctamente.

El término Monitoreo puede estar asociado a infra, aplicaciones y negocio. Siempre el objetivo q se persigue es el seguimiento de parámetros críticos, tener observabilidad sobre lo q ocurre en el ambiente o plataforma, el registro, almacenamiento de datos y la exportación de datos p su posterior análisis /procesamiento; prevenir problemas; o encontrar la solución.

- Cuando hablamos de **monitoreo de infraestructura** generalmente hablamos de monitoreo de: servidores, redes, software, BDD.
- Cuando hablamos de **monitoreo de aplicaciones:** de todo un software q brinda un servicio
- Cuando hablamos de **monitoreo de negocio:** sobre la utilización en la toma de decisiones

Para que sea efectivo necesitamos de **Herramientas** tecnológicas que nos permitan realizarlo

- De observación: ayudan a **supervisar la efectividad** operacional del software, hardware que se estén utilizando
- Análíticas: una vez obtenidos los datos de los activos digitales las aplicaciones nos ayudan a **analizar** la información para detectar problemas y ver cómo y por qué están ocurriendo
- Engagement: permiten **Accionar**, tomar acciones concisas a partir de las herramientas y análisis anteriores

### Como funcionan estas herramientas:

En la mayoría de los casos se dividen en 2 partes:

- **Un agente:** que se instala en el recurso que vamos a monitorear → escucha los eventos que queramos y luego los envía a la consola de monitoreo y poder visualizar la captura efectuada, ya sea por medio de gráficos, histogramas, etc.
- **Consola de monitoreo /administracion**

**¿Cómo controlamos nuestros sistemas?** Mediante las métricas. Vamos a conseguir: monitorear, gestionar, optimizar y generar informes de todos nuestros sistemas.

Para ello tenemos que definir nuestra necesidad → *ejemplo:* que para una carga concurrente de 100 usuarios, los recursos hardware de un servidor web no superen el 60% de su capacidad de uso. Luego hay que asegurarnos de que le sistema funcione bien bajo los parámetros establecidos (y si no funciona, agregar más recursos). Por último hay que establecer alertas a ser enviadas en caso de que se rompan ciertos umbrales

Es un proceso clave para garantizar que todo funcione de la manera que se espera y necesitamos! Nos da información para saber si estamos avanzando en el sentido esperado y nos alerta acerca de cambios o correcciones que deberíamos hacer. En definitiva, El monitoreo es un proceso indispensable cuando desarrollamos un sistema y cuánto más grande sea el proyecto más importancia tendrá.

- Responde a las preguntas ¿vamos por buen camino?, ¿Cómo debería modificar mi intervención para lograr los resultados esperados?

Monitoreo es un proceso sistemático que se encarga de recolectar, analizar y utilizar información con el objetivo de mantener de manera óptima los recursos computacionales y el software de acuerdo a los requerimientos del negocio.

### **¿Qué buscamos al monitorear?**

Como venimos descubriendo, el monitoreo se trata de generar información para poder controlar y verificar que nuestros procesos marchan de acuerdo a lo planificado.

Desarmando esta idea, **podemos identificar cuatro objetivos o propósitos que el monitoreo viene a cubrir:**

1. Evitar y/o prevenir los problemas que puedan surgir ¡Y poder anticiparnos!
2. Entender qué está sucediendo en tiempo real en nuestros recursos y mantenerlos siempre alineados a nuestras necesidades.
3. Hacer análisis porque nos permite almacenar eventos para una revisión posterior.
4. Observar y rendir cuentas comunicando al personal interviniente.

Como vimos, el monitoreo nos guía en la toma de decisiones de gestión, nos permite generar reportes de desempeño y crear indicadores de uso y rendimiento. Pero esos no son los únicos beneficios.

### **Beneficios del monitoreo:**

1. **Observar infraestructura y aplicaciones:** Las aplicaciones modernas están montadas y se ejecutan mediante todo tipo de arquitecturas - monolíticas, distribuidas, de microservicios, entre otras, que generan grandes cantidades de datos en forma de métricas, registros y eventos. Mediante las herramientas de monitoreo podemos recopilar, visualizar y correlacionar en un único punto los datos de todos los recursos, aplicaciones y servicios que funcionan en servidores en la nube y datacenters.
2. **Recopilar métricas para su posterior análisis:** Monitorizar recursos y aplicaciones puede efectuarse en tiempo real y ser consumido mediante una consola centralizada que formatee y presente esos datos de forma legible. También puede hacerse mediante el uso asincrónico, es decir, recopilación y almacenamiento para su uso posterior.
3. **Manipular datos procesables:** Las herramientas actuales de monitoreo nos permiten exportar los datos crudos, también llamados en inglés raw data, para poder ser graficados, almacenados y/o procesados por otros motores de datos con el fin de obtener la visualización que necesitemos.
4. **Alcanzar resultados específicos:** Cada proyecto se construye bajo la influencia de diversas interacciones. El monitoreo es clave en la gestión de cada proyecto y nos ayuda a alcanzar los objetivos tomando en consideración la evolución del contexto, las estrategias, las hipótesis, las suposiciones, etcétera.
5. **Mejorar el rendimiento operativo:** Al poder establecer alarmas y automatizar acciones en función de umbrales predefinidos, podemos saber con exactitud cuándo un recurso o serie de recursos de mi plataforma operativa se están desalineando para poder tomar acciones correctivas y volverlos a la normalidad.
6. **Visibilizar las operaciones diarias:** Tener una vista operativa unificada nos permite optimizar el rendimiento y el uso de los recursos. Al disponer de datos detallados, y en tiempo real, operadores, gerentes de proyecto y stakeholders podrán actuar y tomar decisiones de acuerdo a la información que reciben.

### **Monitoreo de infraestructura y de aplicaciones**

¡No es lo mismo medir recursos hardware que servicios de software, procesos, o la comunicación de una API!

A la hora de definir el monitoreo de nuestro ambiente, existen dos grandes separaciones que tenemos que considerar:

- **Por un lado, la infraestructura.** Es usual que necesitemos monitorear recursos como sistemas de almacenamiento, redes, bases de datos, servidores, etc.
- **Por otro lado, vamos a encontrar aplicaciones.** Allí nos va a interesar monitorear elementos como el número de peticiones o requests, la cantidad de fallos producidos por un servicio web, eventos de llamadas API, etc.

### **¡Herramientas para monitorear!**

Nuestra decisión va a depender de qué queremos monitorear, el costo, la complejidad y la curva de aprendizaje.

Todas las herramientas de monitoreo van a cumplir las mismas funciones sobre aplicaciones e infraestructura, estas funciones son:

- Trazabilidad y observabilidad en tiempo real.
- Alertas.
- Almacenar para su posterior análisis.
- Métricas integradas.
- Reportes.

No obstante, cada herramienta ofrecerá prestaciones específicas dependiendo de su versión. ¡Vamos a conocer cuáles son las más populares y las funcionalidades de cada una!

- **DATADOG**: Plataforma de seguridad y monitoreo para aplicaciones en la nube. Reúne seguimientos, métricas y registros de un extremo a otro para que aplicaciones, infraestructura y servicios de terceros sean completamente observables. Está orientada a proveer servicios en la nube.

Puntos fuertes:

- Madura integración con distintos tipos de software mediante el uso de APIs.
- Gran flexibilidad y versatilidad para configurar de diversas formas los tableros de control.
- Granularidad extrema sobre los servicios a elegir para monitorear.

- **NAGIOS**: Sistema de monitorización de redes ampliamente utilizado y de código abierto. Vigila los equipos (hardware) y servicios (software) que se especifiquen, alertando cuando el comportamiento no sea el deseado. Mayormente utilizado en ambientes on-premises, aunque dispone de servicios adicionales que apuntan a tener presencia en la nube.

Puntos fuertes:

- Descubrimiento automático de recursos sin uso de agentes.
- Sistema de ticketing.
- Monitoreo ambiental

- **PROMETHEUS**: Base de series de tiempo y un sistema de monitoreo y alertas. Las series de tiempo almacenan datos ordenados cronológicamente, midiendo variables a lo largo del tiempo. Las bases de datos enfocadas a series de tiempo son especialmente eficientes en almacenar y consultar estos datos. Dispone de funcionalidad para cloud y on-premises.

Puntos fuertes:

- Modelado de datos.
- Lenguaje PromQL.

- **CLOUDWATCH**: Servicio de monitorización y administración que suministra datos e información procesable para aplicaciones y recursos de infraestructura locales, híbridos y de AWS. Permite recopilar y obtener acceso a todos los datos de rendimiento y operaciones en formato de registros y métricas a partir de una sola plataforma. AWS Nativo, con lo cual se integra fácil y naturalmente con cualquier servicio de AWS.

Puntos fuertes:

- Correlación de registros con métricas.
- Métricas de flujo.
- Análisis de registros mediante inteligencia artificial.

## **La importancia de las métricas**

A la hora de monitorear nuestro ambiente surge una pregunta fundamental

¿Qué métricas debo considerar para saber el estado de mi infraestructura y de las aplicaciones que he montado sobre ella?

**¿Qué es medir?** La medición es el proceso que se basa en comparar una unidad de medida preestablecida con el elemento cuya magnitud se desea medir y, de esta forma, comprender cuántas veces la unidad está contenida en esa magnitud. ¿Qué nos proponemos? Definir una estructura coherente de métricas a monitorear para poder gestionar, optimizar y generar informes de todos nuestros servicios de forma regular.

**¿Qué son los indicadores?** Son el resultado de manipular las métricas para obtener información sobre el comportamiento basado en diferentes variables. Tanto las métricas como los indicadores nos pueden dar información para tomar decisiones de negocio relacionadas con las funcionalidades del producto o la plataforma donde se encuentra. ¿Qué nos proponemos? Convertir datos crudos en indicadores nos permitirá conocer nuestro ambiente, controlarlo y medir los recursos eficientemente, poder anticiparnos al mercado y optimizar el esfuerzo en el desarrollo de productos.

**¿Qué es la degradación?** Es la desviación de las métricas predefinidas que se suelen manifestar mediante la baja performance de un servicio. El tiempo y el uso son factores que inciden directamente sobre la performance de nuestro ambiente.

Esta desviación en las métricas configuradas nos ayudará a identificar problemas en nuestros sistemas con suficiente tiempo antes de que se produzcan problemas mayores.

**¿Qué hacer cuando nuestros sistemas tienen degradación?**

- Evaluar los recursos disponibles vs. lo consumido en función de los límites preestablecidos.
- Disponer de mapas de arquitectura que muestran el flujo de datos dentro del sistema.
- CDM (CPU, disco y memoria) son recursos esenciales a chequear en nuestras mediciones.
- Revisar logs e historial sobre cualquier cambio acontecido recientemente.

Buenas y malas prácticas

- Buenas: ¡Medir con criterio! Segmentar analíticamente cada servicio a monitorear.
- Malas: ¡Medir en exceso! Esto nos tapaná la visibilidad sobre lo que realmente necesitamos medir

## **Tipos de métricas**

### **1. Métricas de trabajo**

- a. **Rendimiento:** La cantidad de trabajo que realiza el sistema por unidad de tiempo. El rendimiento generalmente se registra como un número absoluto.
- b. **Success/ Error:** Las métricas que representan el porcentaje de trabajo que se ejecutó correctamente y las que han tenido fallo.
- c. **Performance:** Es la cuantificación de la eficiencia con la que un componente está haciendo su trabajo. Se suele comparar con medidas preestablecidas.

### **2. Métricas de recursos:**

- a. **Utilización:** Es el porcentaje de tiempo que un recurso está ocupado o en uso.
- b. **Saturación:** Es la medida de la cantidad de trabajo que el recurso aún no puede atender, a menudo en espera o queue.
- c. **Disponibilidad:** Representa el porcentaje de tiempo que el recurso tiene para ofrecer a las solicitudes.

### **3. Eventos:**

- a. **Cambios en el código:** Involucra todo tipo de cambio en el código: modificaciones, compilaciones y/o fallas.
- b. **Alertas:** Eventos generados en función de algún parámetro preestablecido sobre algún recurso.
- c. **Autoescalado:** La adición y/o sustracción automática de recursos en función a la carga u otra configuración preestablecida

---

## **Módulo 4 - Clase 20**

### **Monitorear infraestructura**

Dato de la realidad: las organizaciones dependen cada día más de herramientas que se apoyan en infraestructuras digitales. ¡Gestionar y monitorear estos recursos es cada vez más importante para el correcto desempeño de un proyecto o negocio!



La infraestructura está compuesta por una serie de elementos usualmente ligados al hardware. Pero, como venimos viendo, con la utilización de software de virtualización el límite entre hardware y aplicaciones o servicios es cada vez más difuso.

¡Hagamos un breve repaso! ¿Qué podemos monitorear? Estos son algunos de los recursos en hardware físico o virtual:

- Redes: enrutadores, switches, firewalls, vpc, balanceadores, etc.
- Máquinas o servidores: CPU, memoria, almacenamiento, autoescalado de instancias.
- Servicios administrados: dependerá del tipo de servicio la medición a realizar.

## Amazon CloudWatch

¡Enfoquemos en una herramienta! CloudWatch es el servicio de monitoreo de Amazon Web Services. Esta solución cubre muchas de las necesidades que nos encontramos al encarar problemas de monitoreo de infraestructura. Es, además, un gran recurso para realizar ajustes de mejora y reducción de costos.

### ¿Qué es CloudWatch?

CloudWatch es un servicio de monitoreo y administración de AWS que nos permite centralizar todos los datos generados por infraestructuras, servicios y aplicaciones locales, en la nube, e híbridos.

Tiene la capacidad de generar alarmas programables, eventos que desencadenan acciones automáticas, y exponer estos datos para su análisis.

### Como trabaja

CloudWatch nos propone un flujo de trabajo con las siguientes partes:

1. **colecta:** Se recolectan los datos a visualizar o analizar. CloudWatch permite datos en formato de métricas, registros y eventos.
2. **Monitoreo** Los datos recolectados se almacenan para ser mostrados en paneles visuales o dashboards.
3. **Acción** Luego de la colecta es posible establecer actuadores como alarmas o disparadores de acciones.
4. **Análisis** Por último nos brinda una serie de herramientas para poder realizar análisis sobre los datos recolectados.



### Obtención de datos

Existen tres tipos de datos:

1. **Métricas** Valores de magnitudes medibles con una firma de tiempo e identificador.
2. **Registros** Trozos de texto que representan una información (mensajes de consola de una app).
3. **Eventos** Notificaciones sobre cambios en la infraestructura, servicios externos o alarmas activadas.

### Obtención de datos

Amazon cuenta con servicios diseñados para la obtención de datos y su envío a CloudWatch. ¡Solo hay que configurarlo!

También ofrece la posibilidad de subir datos mediante su agente (aplicación que podemos instalar) o a través de su propia API.

Dentro de este servicio vamos a acceder a distintas secciones: paneles, logs, métricas, eventos, etc., en las que podremos visualizar la información que necesitamos.

## Monitoreo

CloudWatch -como toda buena herramienta de monitoreo- nos permite **Crear paneles a medida, gráficos e indicadores** que visibilicen mejor el estado de los recursos.

También tenemos disponibles paneles por defecto para la mayoría de los servicios existentes en AWS.

CloudWatch nos facilita crear **alarmas simples**-que se configuran con umbrales específicos de acuerdo a la necesidad- y **alarmas compuestas** activadas por un conjunto de otras alarmas.

Entre otras características posee detección de anomalías, utilizando aprendizaje automático para determinar comportamientos poco comunes.

**Acciones** → Mediante CloudWatch podemos **disparar acciones automáticas** utilizando tanto las alarmas ya vistas como eventos.

Los eventos pueden ser propios de AWS -como cambios en recursos, por ejemplo se pueden programar para que se disparen cada un determinado periodo de tiempo, como un despertador

**Autoescalado** → Un ejemplo es el **autoescalado de recursos** cuando se detecta un alto consumo de CPU, se dispara la acción de escalar horizontalmente (hacer crecer) la cantidad de máquinas virtuales para satisfacer la demanda. A su vez, si el consumo es mínimo, podemos hacer decrecer la cantidad de servidores.

**Análisis** → CloudWatch ofrece operaciones sobre métricas, para obtener información en tiempo real y analizarla en un panel gráfico.

#### Para tener en cuenta

- Almacena hasta **15 meses** de métricas.
- Recolecta de datos en intervalo de hasta **un** segundo.

Mediante **cloudWatch Logs Insighth** permite analizar los logs, realizar consultas con filtros y exportar a los paneles obteniendo una visibilidad operativa completa.

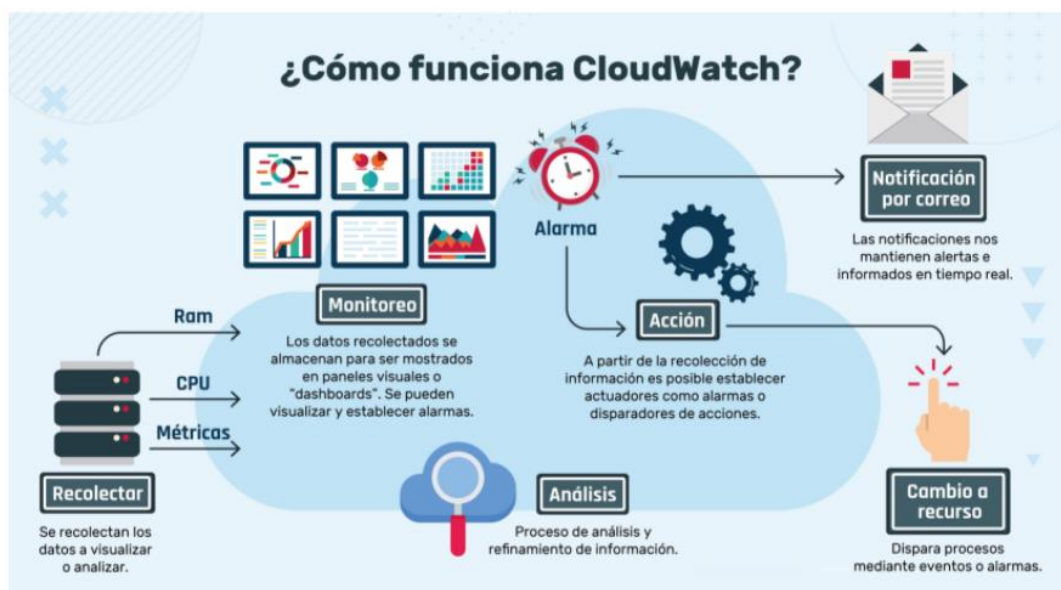
#### Conclusión

En esta presentación conocimos los distintos grupos de características que ofrece el servicio de **Amazon CloudWatch** y sus posibilidades con respecto al monitoreo de infraestructura.

¡Vamos ahora a pasar a la acción y ponerla en uso!

#### ¿Cómo funciona CloudWatch?

Veamos gráficamente cómo funciona la herramienta para monitoreo de Amazon Web Services



- Recolector
- Monitoreo

- Acción
- Análisis
- Notificación por correo
- Cambia a recurso

**Autoescalado y elasticidad** → Una de las características más interesantes que nos ofrecen los proveedores de cloud es el autoescalado. Esta funcionalidad nos permite administrar el crecimiento y decrecimiento de nuestra infraestructura de manera automática —de acuerdo a la demanda— permitiéndonos atender de manera satisfactoria momentos específicos en los que nuestra aplicación es más requerida. Ahora, ¿cómo lo hace? ¡Vamos a conocer más!

## Video

el autoescalado y la elasticidad el autoescalado es la posibilidad de dimensionar nuestra infraestructura de manera responsiva ante la demanda de los clientes basándonos en la elasticidad de nuestro proveedor. Dónde la elasticidad no es más que la capacidad pero servicios en la nube de entregar o eliminar recursos automáticamente proporcionando la cantidad justa de activo para cada proyecto en una tienda física para disminuir el tiempo de espera entre una compra y otra en un momento gran demanda de único que podemos hacer es habilitar más cajas o puntos para pagar no bueno espero que dependerá del espacio del lugar y lo más importante es habilitar o deshabilitar estas cajas o puntos de pago de forma dinámica cada vez que la demanda sube o baja.

en una tienda online si podemos hacerlo siempre y cuando su infraestructura se base en un proveedor en la nube que sea elástico y pueda autoescalar nuestros recursos nuestra infraestructura base esta con un balanceador de carga y detrás dos servidores web esa infraestructura es suficiente para sostener una demanda habitual.

## Que pasa con esa infraestructura en momentos de mayor carga

-se generan demoras o caídas perdiéndose ventas

El auto escalado propone un crecimiento horizontal de los recursos a medida que estos son más demandados entonces de manera reactiva se agregan más servidores al esquema de balanceo siempre monitoreando monitoreando la performance de nuestra tienda y actuando en consecuencia.

El autoescalado nos permite que este esquema de monitoreo y crecimiento se realice de manera automática ya que AWS se encargara de monitorear determinados parámetros que le indiquemos y en función donde ellos nuestra infraestructura crecerá o decrecerá según la necesidad mantiene la infraestructura lo más ajustado posible evitando costos innecesarios un producto o servicio auto escalable proporcionado por la elasticidad nuestro proveedor de servicios en la nube es fundamental para evitar demoras y caídas por falta de recursos además de los gastos de la infraestructura crecerán solamente cuando la demanda vida evitando que existan recursos ociosos el aumento innecesario de Los costos.

---

## Clase 21ª-Cierre de semana

**Monitoreo** → Definimos qué es monitorear y qué buscamos exactamente con esta práctica: el monitoreo es una tarea periódica que permite documentar y utilizar resultados, procesos y experiencias como base para dirigir la buena toma de decisiones de forma continua.

**Herramientas** → Exploramos algunas características propias de las herramientas de monitoreo más populares del mercado.

**Pipelines y monitoreo** → El ciclo CI/CD no abarca solamente el planear, codificar, testear, deployar y liberar, sino también el monitoreo, práctica sin la cual nuestro ambiente no podría mantenerse en pie por los errores que puedan aparecer.

**Beneficios** → Recorrimos los beneficios adicionales que podemos obtener con esta práctica y la importancia de los reportes para el análisis y la gestión del negocio.

**Métricas e indicadores** → Nos detuvimos en las métricas y los indicadores como elementos de información vital y decisiva

**Estructura/aplicaciones** → Distinguimos las particularidades del monitoreo para la infraestructura y para las aplicaciones.

---

## Clase 22 - Monitoreo de Aplicaciones

Infraestructura y aplicaciones son dos capas distintas a controlar. Cuando nuestra aplicación está deployada, necesitamos cuidar de ella.

El monitoreo de aplicaciones permite:

- Garantiza que nuestras aplicaciones corporativas se comporten del modo esperado ¡en todo momento y con mínima supervisión humana!
- Genera informes de los problemas de rendimiento.
- Nos acerca análisis de gestión adecuados.
- Sirve para comprender cómo se comporta nuestra aplicación y qué necesita para su correcta operación en términos de performance, disponibilidad y seguridad.

### Video

El monitoreo de aplicaciones proporciona un seguimiento distribuido de los dispositivos frontend nuestro web servers o interfaces de aplicación hasta las bases de datos, al monitorear las aplicaciones estaremos vigilando que nuestros servicios estén siempre alineados con las métricas predeterminadas y que su comportamiento es lo que se espera en todo momento Y durante el ciclo de vida es exactamente lo mismo que el monitoreo de infraestructura salvo que acá estamos en una capa superior es decir el software montado sobre la infraestructura pero que cosa tenemos monitoreado principalmente el rendimiento como los tiempos de respuesta a las aplicaciones las trazas lentas que en función de la configuración preestablecida nos permiten ver las peticiones lentas y trazarlas internamente y las query de base de datos que nos permiten saber si tenemos query que deben estar impactando en la aplicación y cuales son mejorables también deben monitorear la continuidad como los errores de aplicación donde podemos dotar a nuestro sistema con la capacidad de filtrar los errores y mostrar cuándo y cuántas veces han ocurrido ahora

**cómo seleccionamos un software de monitoreo de aplicaciones** para esto debemos tener presente algo muy importante el **análisis es siempre lo primero** que necesitamos, queremos que nuestra aplicación siempre esté arriba que se nos alerte si esto no sucede queremos tener trazabilidad sobre su comportamiento queremos que se guarden todos los eventos que surjan, tener reportes tener un buen soporte, de nuestro análisis saldrá la decisión apropiada para elegir el producto que más se adecue a nuestras necesidades para asegurarnos de que nuestras aplicaciones cumplan con su función sin problemas una vez que esté productiva es necesario comprender cómo se comportan y que se necesita para su correcta operación en términos de performance de disponibilidad y seguridad y la única forma de lograr esto es con auditoria

### ¿Qué miramos cuando monitoreamos aplicaciones?

El monitoreo de aplicaciones y el monitoreo de infraestructura tienen mucho en común, ¡y una diferencia importante! Las métricas a las que apuntan.

### # Qué monitoreamos

La accesibilidad, la cantidad de llamadas APIs o si existen errores en las páginas que estamos sirviendo.

### ## Métricas

Para cada tipo de aplicación vamos a usar diferentes métricas.

Estas son algunas de las métricas más populares cuando hacemos monitoreo de aplicaciones:

- Usuarios activos
- Uso de CDM ( CPU, Memory, Disk ) de la aplicación
- Cantidad y duración de sesiones concurrentes, picos, etc.
- Tiempo de disponibilidad online de la aplicación

- Eventos registrados (errores, warnings, etc.)
- Número de hits sobre el sitio
- Estadística en bases de datos
- Tasa de retención de usuarios
- Llamadas a eventos o servicios externos

Las métricas dependerán de factores como el producto empleado y las tecnologías a monitorear.

Es importante tener un mapa de arquitectura, como el modelo OSI de capas, que muestre el sistema lógico y el flujo de datos.

Con el mapa y el flujo podemos tener una visión holística de lo que estamos monitoreando.

## # CloudWatch

Vamos a utilizar la misma herramienta para hacer el monitoreo de una aplicación web.

Es un servicio de monitoreo y observación que ofrece datos e información procesable de nuestra infraestructura y también de nuestras aplicaciones, con la finalidad de optimizar los recursos, detectar fallas y errores de manera temprana y encontrar soluciones.

## ## Beneficios

- Optimizar recursos
- Detectar fallas y errores
- Encontrar soluciones

## ## Métricas - Para monitorear una aplicación web se utilizan estas métricas:

- códigos de estado de respuesta HTTP de las peticiones
- Cantidad de peticiones sobre recursos específicos
- Contenido de la aplicación
- Latencia en las rtas

El monitoreo se realiza de forma diferente que el de la infraestructura

- La aplicación web debe ser publicada con alguno de los servicios que ofrece AWS como **`Elastic Beanstalk`**. Permite crear aplicaciones y desplegar un conjunto de servicios como: EC2, S3, SNS, Cloudwatch, Autoescalado y Balanceador.
- Soporta lenguajes como: NET core, JAVA, PHP, Python, entre otros.
- Al desplegar nuestra aplicación usando **`ElasticBeanstalk`** AWS nos da total acceso a las métricas que necesitamos para monitorearla.

## ## Pasos

1. Seleccionar que métricas vamos a exponer en nuestra app: este paso se realiza en ElasticBeanstalk.
2. Recolectar los datos de las métricas en CloudWatch.
3. Establecemos umbrales críticos y fatales y a cada uno de ellos le asignamos acciones. Esto también se realiza en CloudWatch.

Las acciones de notificaciones son tomadas por el servicio SNS (Simple Notification Service). Por ejemplo podemos programar que ante una cierta cantidad de peticiones se dispare el envío de un correo electrónico.

## # Actividad

Ejercitación S8 - Monitoreamos una App e integramos Slack.pdf

[https://drive.google.com/file/d/1il\\_FnM8MvLxZfyETW7EJJEYVLKi\\_D05q/view?usp=drivesdk](https://drive.google.com/file/d/1il_FnM8MvLxZfyETW7EJJEYVLKi_D05q/view?usp=drivesdk)

---

## Clase 23 – Cierre semana

Definimos el **monitoreo** como el proceso sistemático que se encarga de recolectar, analizar y utilizar información con el objetivo de mantener de manera óptima los recursos computacionales y el software de acuerdo a los requerimientos del negocio.

### Que buscamos monitorear:

- Evitar y/o prevenir los problemas que puedan surgir ¡Y poder anticiparnos!
- Entender qué está sucediendo en tiempo real en nuestros recursos y mantenerlos siempre alineados a nuestras necesidades.
- Hacer análisis porque nos permite almacenar eventos para una revisión posterior.
- Observar y rendir cuentas comunicando al personal interviniente.

**Herramientas:** Presentamos las herramientas más populares para hacer monitoreos (CloudWatch, Nagios, Prometheus, DataDog) y profundizamos en CloudWatch, trabajando con la herramienta en distintas prácticas

A la hora de definir el monitoreo de nuestro ambiente, existen dos grandes divisiones que tenemos que considerar:

- **En el monitoreo de infraestructura** vamos a mirar recursos como sistemas de almacenamiento, redes, bases de datos, servidores, entre otros.
- **En el de las aplicaciones**, nos va a interesar monitorear elementos como el número de peticiones o requests, la cantidad de fallos producidos por un servicio web, eventos de llamadas API, entre otros.

**Autoescalado (Elasticidad):** Esta funcionalidad nos permite administrar el crecimiento y decrecimiento de nuestra infraestructura de manera automática —de acuerdo a la demanda— permitiéndonos atender de manera satisfactoria momentos específicos en los que nuestra aplicación es más requerida.

**Indicadores y métricas:** Los indicadores resultan de manipular las métricas para obtener información sobre el comportamiento de diferentes variables. Tanto las métricas como los indicadores nos dan información para tomar decisiones de negocio relacionadas con las funcionalidades del producto o la plataforma donde se encuentra.

Presentamos el monitoreo de un sitio web de comercio electrónico, revisando qué indicadores podemos considerar en el front end, el back end y en la visibilidad de los procesos. Aprendimos que:

- Las métricas puntuales dependerán de factores como el producto empleado y las tecnologías a monitorear.
- Es importante disponer siempre de un mapa de arquitectura que muestre el sistema lógico y, dentro de él, el flujo de datos.
- Mapa y flujo nos permiten tener una visión holística de lo que estamos monitoreando o necesitamos monitorear.