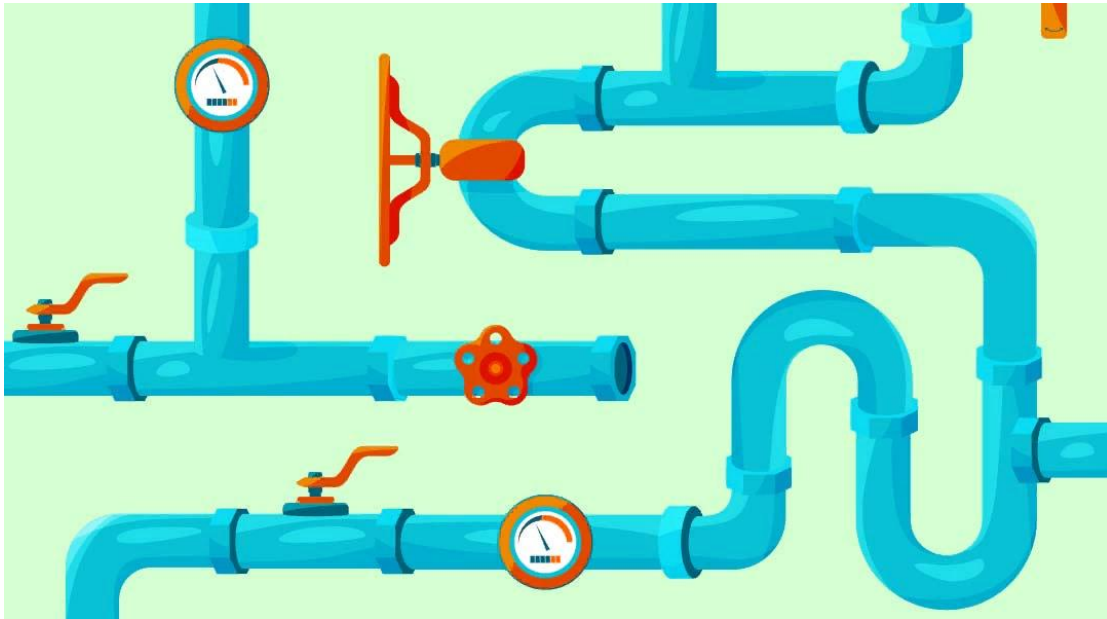


Spring Boot

Microservicios y despliegue continuo



Entrega rápida y frecuente



Fuente de la imagen: devclass.com

A medida que las industrias buscan **la transformación digital**, surge la necesidad de encontrar formas de **mejorar y acelerar la entrega** de nuevas aplicaciones, servicios y capacidades.

Empresas de todo el mundo se **esfuerzan por aprovechar las ventajas competitivas** de la disrupción digital.

Las empresas están adoptando **estrategias de transformación ágil y digital** que les ayudan a centrarse firmemente en ofrecer servicios en **menos tiempo manteniendo una alta calidad**.

Cuál es **el papel de los oleoductos** en este escenario?

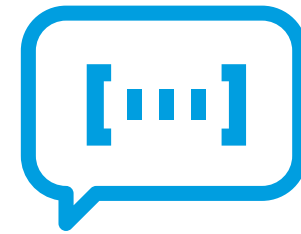


DISCUSSION

**No nos gusta la lentitud humana...
¿Cómo podemos automatizar la
entrega de la aplicación?**

01

CI/CD

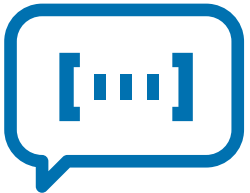


OBJETIVO

Iinsights sobre la entrega workflow

INSTRUCCIONES

1. Pensar acerca de el proceso sigue una aplicación de el punto cuando código es listo y la aplicación está entregado al usuario final.
2. **Encontrarse** con su compañero y tratar de respuesta **estos preguntas** :
 - **Cómo muchas veces en un año pensar ¿ Se debe entregar una aplicación ?**
 - **Cómo debería aquellos momentos ser como ?**
 - **¿Cómo hacemos ? mezcla código de el varios desarrolladores ? ¿ Dónde ?**
 - **Debe el entero funcionalidades terminadas _ a entregar una aplicación?**
 - **Debería nosotros ir de el desarrollador computadora a el servidor de producción ?**
3. Usa post-its a **recolectar** las ideas.
4. **Prepárate** para compartir tu perspectivas con el descansar de el clase.



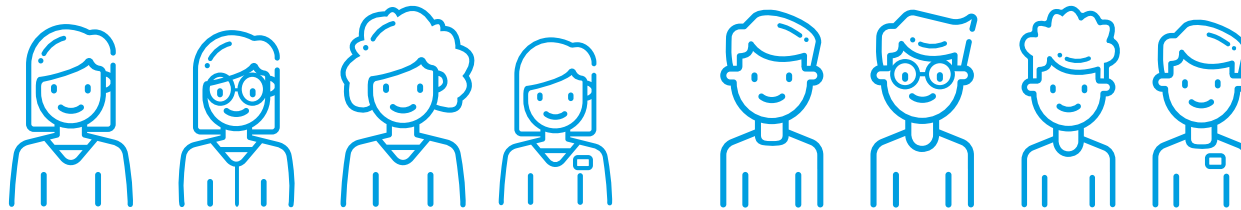
10 min

OBJETIVO

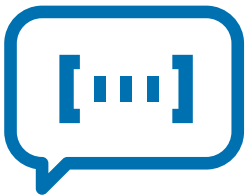
Comparte tus ideas !

INSTRUCCIONES

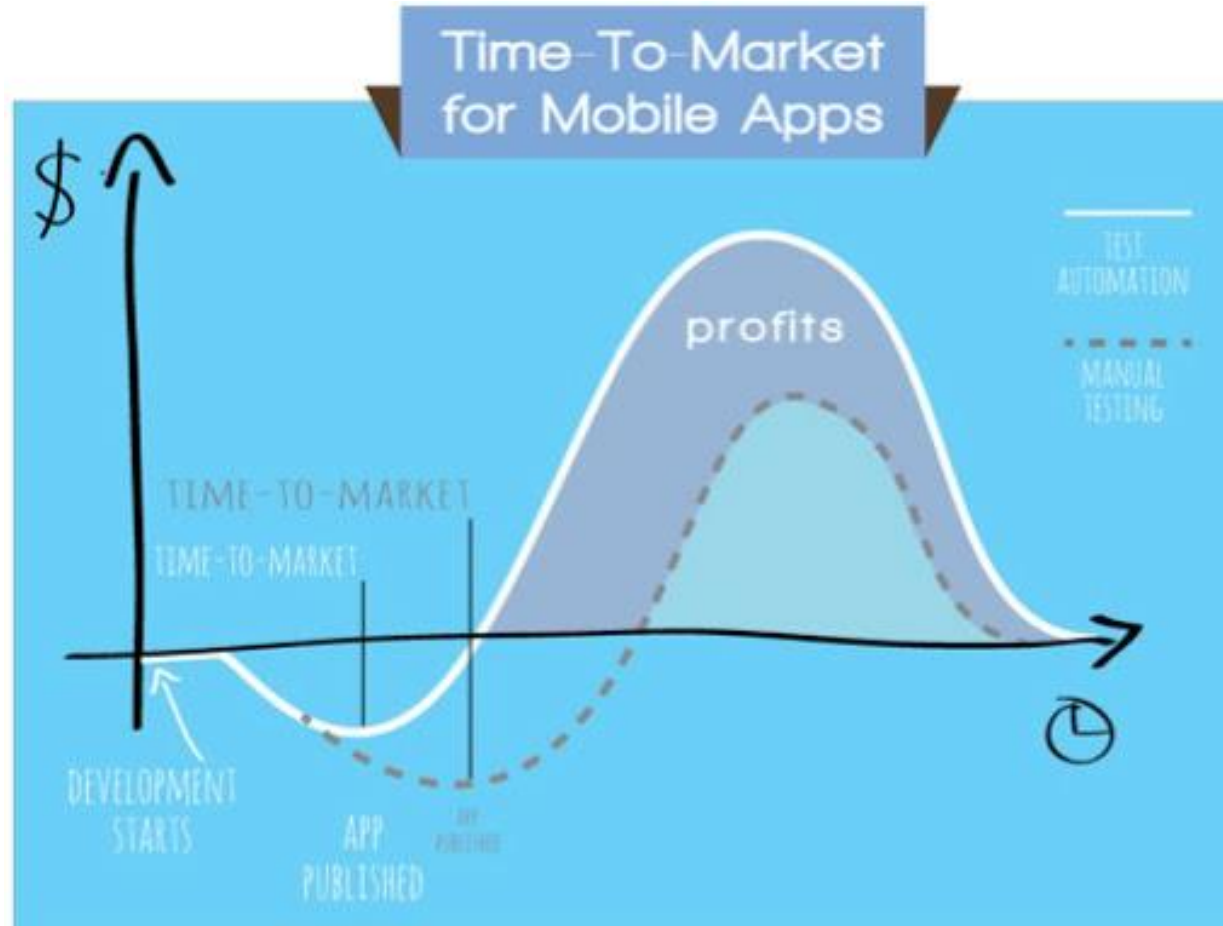
1. **Comparte** tu perspectivas con el resto de la clase.
2. Generar **conclusiones comunes**.



15 min



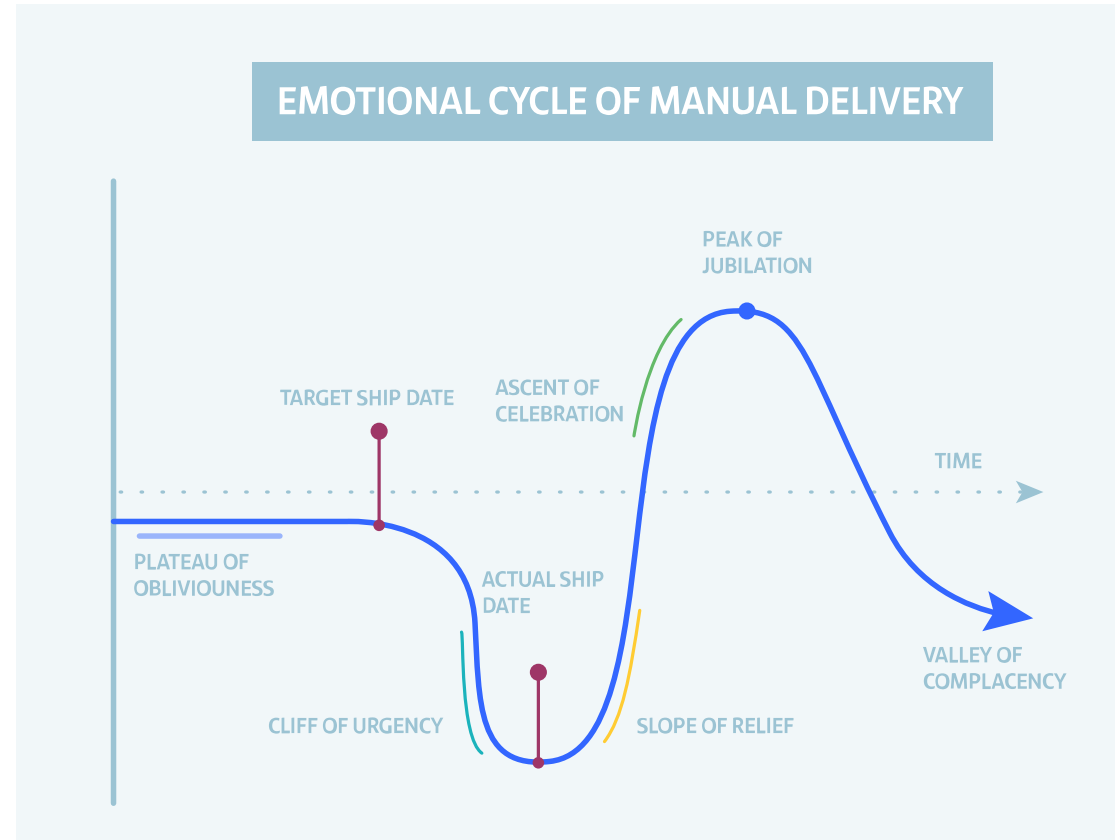
La necesidad de velocidad



“El software se está
comiendo el
mundo”

Con un Time-to-market bajo, las organizaciones tienen
más posibilidades de superar a la competencia y
permanecer en el negocio.

Liberación



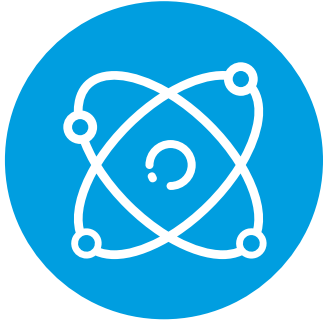
Si tu equipo todavía vive con pruebas manuales para prepararse para las entregas y despliegues manuales o usando scripts para realizarlas, sus sentimientos pueden estar más cerca del "pavor" y la "ira ciega".

Avanzando hacia la continuidad



- **En el paradigma continuo, los productos de calidad se lanzan a los clientes de forma frecuente y predecible.**
- Por lo tanto, se reduce la ceremonia y el riesgo de liberación.
- **Esto se logra a través de un proceso de entrega automatizado.**
- Si se depende de pipelines a diario, se identificarán (¡y resolverán!) las deficiencias mucho más rápido que si fluyeran una vez cada pocas semanas o meses.
- Es decir, se reduce la dificultad aumentando la frecuencia de los lanzamientos de los productos.
- Por tanto, **la continuidad está estrechamente relacionada con las tareas automatizadas.** Examinemos cómo.

Rasgos de continuidad



Velocidad

Los pipelines de entrega de software automatizados ayudan a las organizaciones a responder mejor a los cambios del mercado. Hoy en día contamos con herramientas para automatizar casi todos los procesos del proceso de entrega.



Productividad

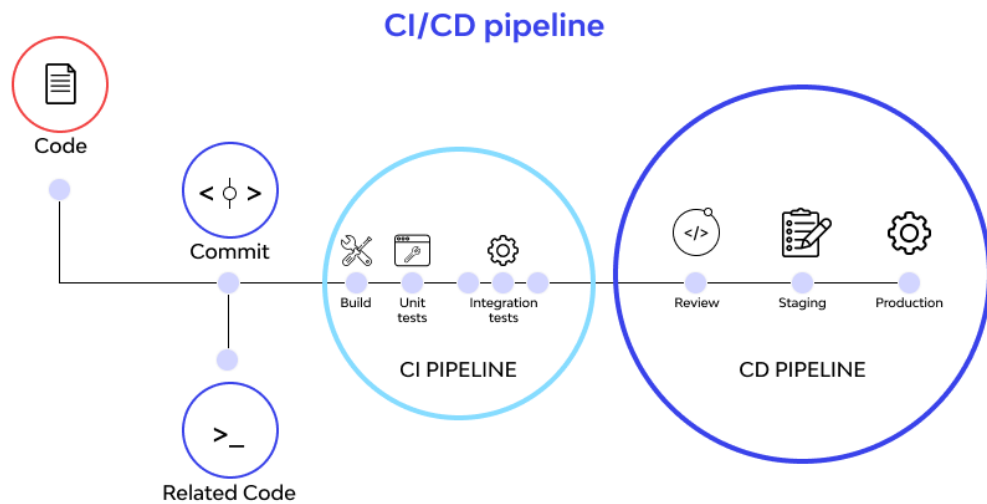
La productividad aumenta cuando los pipelines, en lugar de los humanos, pueden realizar tareas tediosas y repetitivas, como completar un informe de error por cada defecto descubierto.



Sostenibilidad

Mantenerse constantemente a la cabeza de la competencia puede ser aún más difícil. Se necesita disciplina y rigor. Trabajar duro las 24 horas del día, los 7 días de la semana, provocará un agotamiento prematuro.

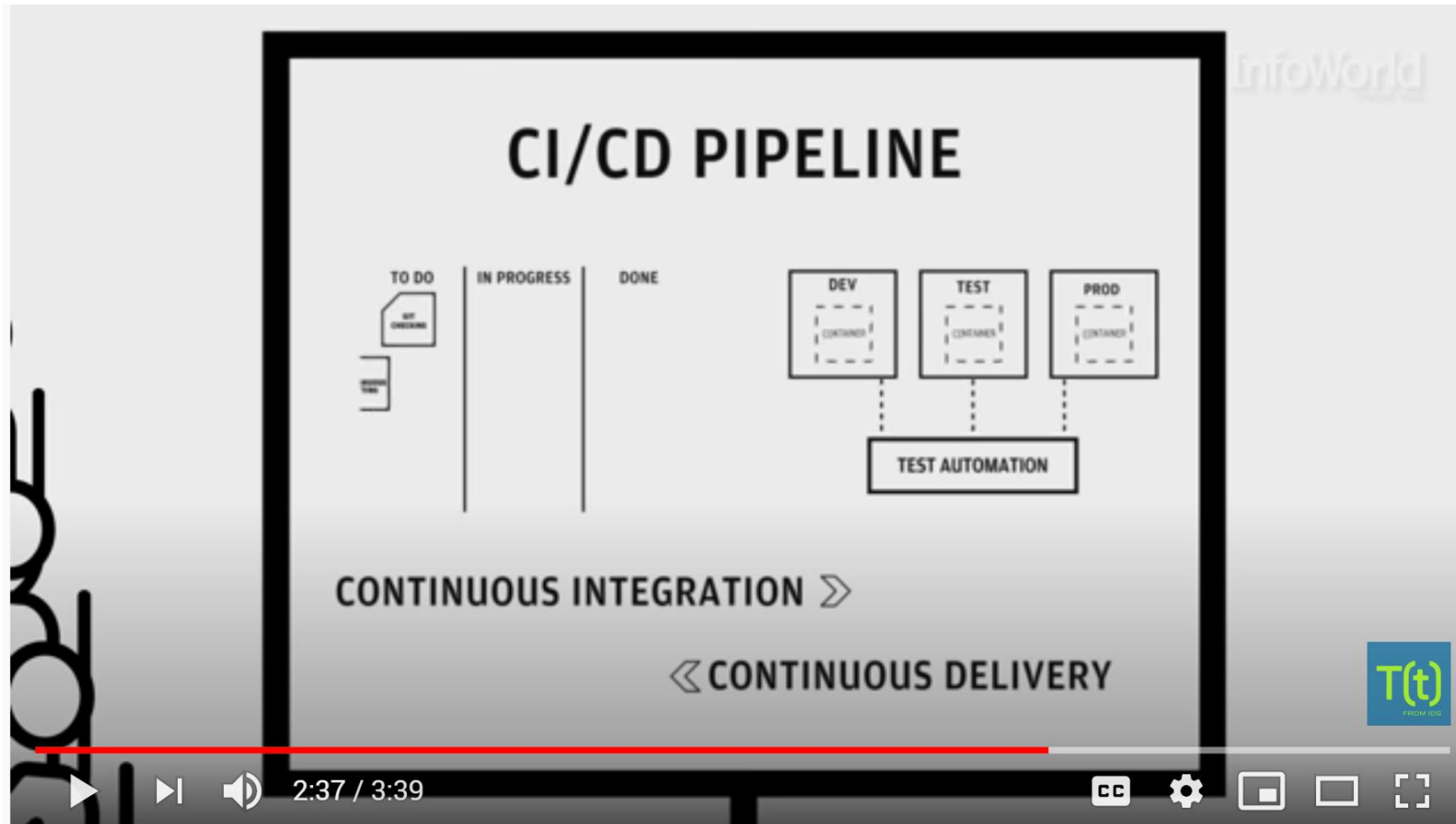
CI/CD



Fuente de la imagen: wallarm.com

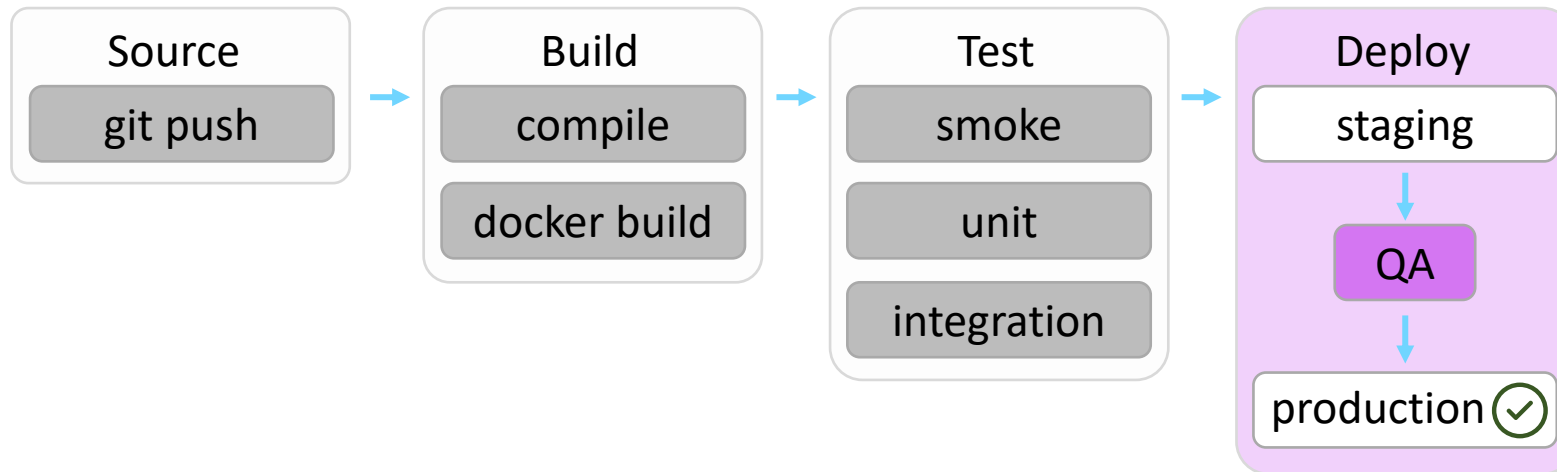
- La continuidad y la entrega frecuente se ponen en práctica en las organizaciones mediante las prácticas de CI/CD.
- **CI/CD es una colección de principios y prácticas operativas que ayudan a los equipos de desarrollo a realizar cambios frecuentes de código de manera confiable.**
- Es la automatización y el monitoreo continuos a lo largo del ciclo de vida de la aplicación, desde la integración y las pruebas hasta las fases de entrega e implementación del producto.

Cómo entregar código más rápido con CI/CD



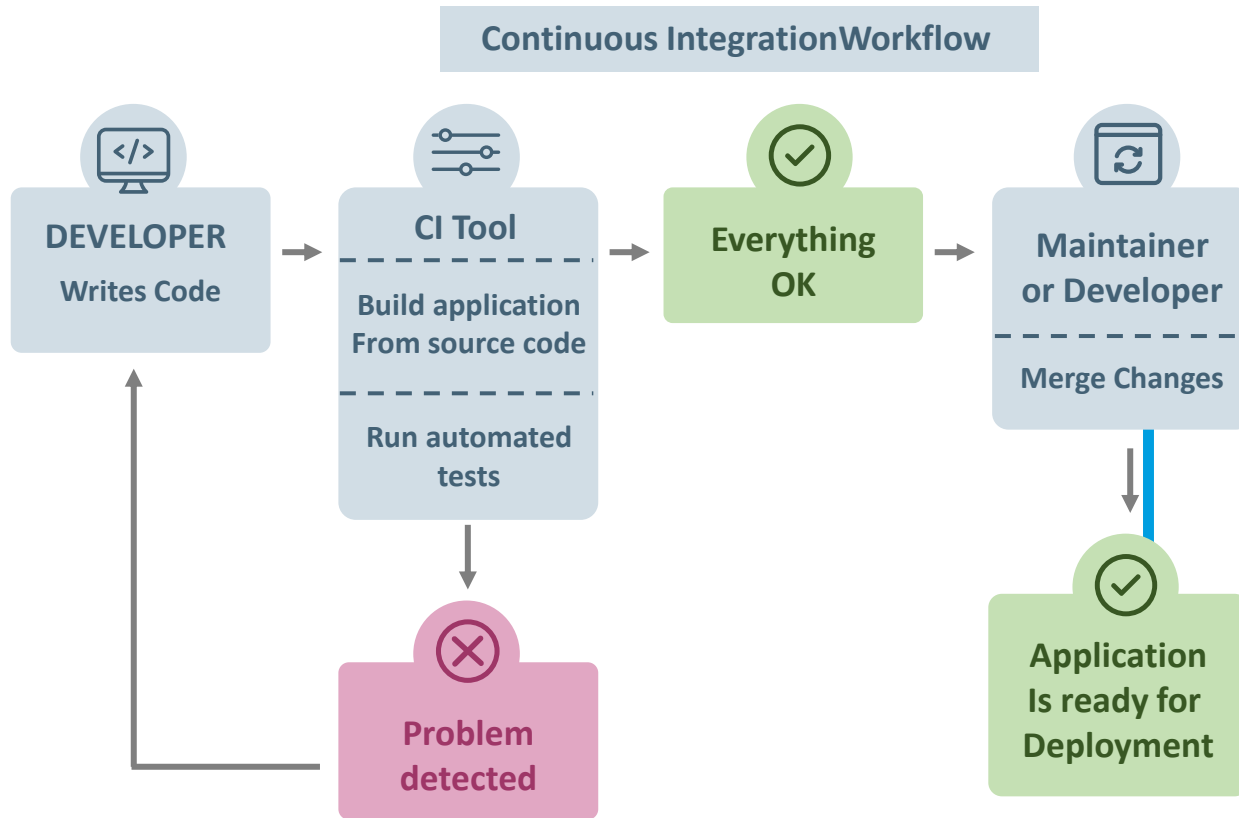
<https://www.youtube.com/watch?v=Rq5TQIPyr8g>

pipeline de CI/CD



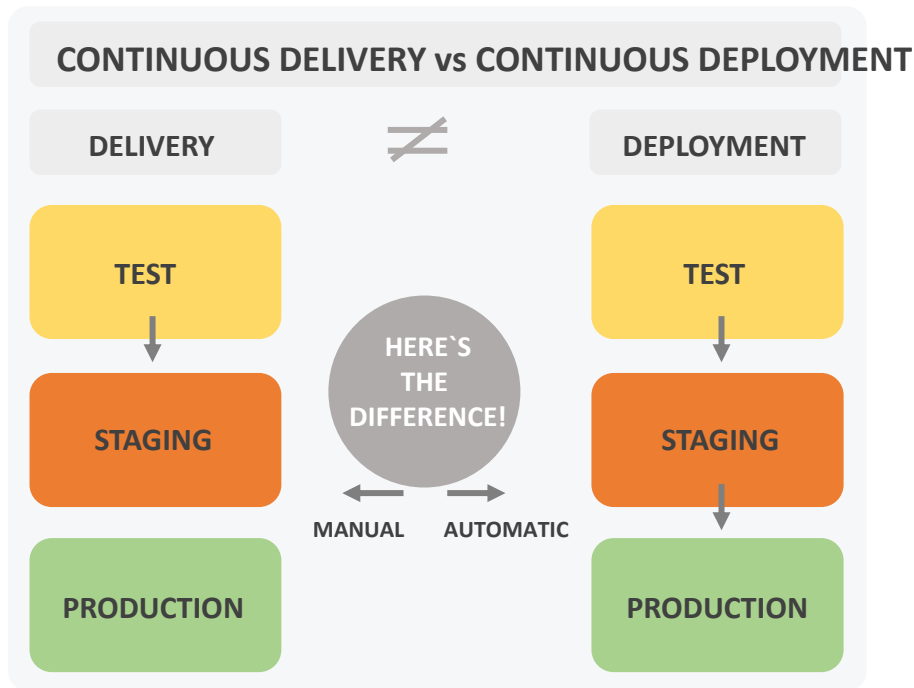
Un pipeline CI/CD puede parecer una tarea costosa, pero en realidad no lo es.
Es esencialmente una especificación ejecutable de los pasos que deben realizarse para entregar una nueva versión de un producto de software.
En ausencia de un proceso automatizado, los ingenieros aún necesitarían realizar estos pasos manualmente y, por lo tanto, de manera mucho menos productiva.

Integración continua



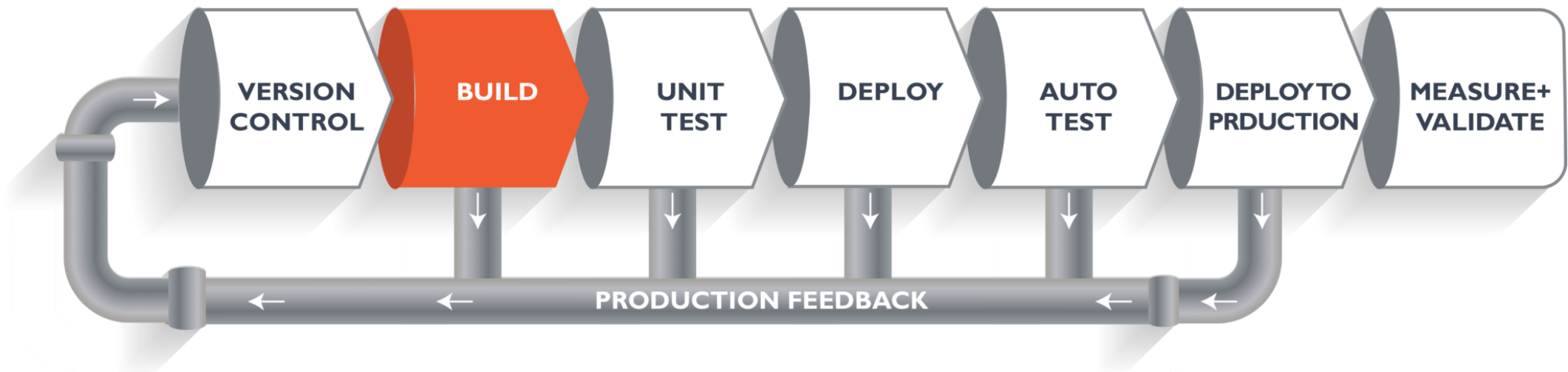
- CI es la práctica de automatizar la **integración de cambios de código** de múltiples contribuyentes en un solo proyecto de software.
- El proceso de CI consta de herramientas automáticas que **validan la exactitud** del nuevo código antes de la integración.
- La integración continua pone un fuerte énfasis en la automatización de pruebas para verificar que la aplicación no se rompa cada vez que se integran nuevas confirmaciones en la rama principal.

Entrega continua y Despliegue continuo



- CI y CD son dos acrónimos que se mencionan a menudo cuando la gente habla de prácticas de desarrollo modernas.
- **CI** es sencillo y significa **integración continua**, una práctica que se centra en facilitar la preparación de un lanzamiento.
- Pero **CD** puede significar **entrega continua** o **despliegue continuo** y, si bien esas dos prácticas tienen mucho en común, también tienen una diferencia significativa que puede tener consecuencias críticas para una empresa.

Etapas de CI/CD



Fuente de la imagen: dzone.com

Las prácticas de CI/CD y sus herramientas relacionadas se implementan en **etapas concretas de entrega de software** en proceso, como confirmar cambios, construir, probar, etc.

OBJETIVO

Las 4 etapas de CI/CD

INSTRUCCIONES

1. Ve al siguiente enlace para leer sobre las etapas básicas de CI/CD:
 - <https://haritechworld.com/2020/08/13/concepts-the-4-stages-of-the-ci-cd-pipeline/>
2. Complete la siguiente tabla.

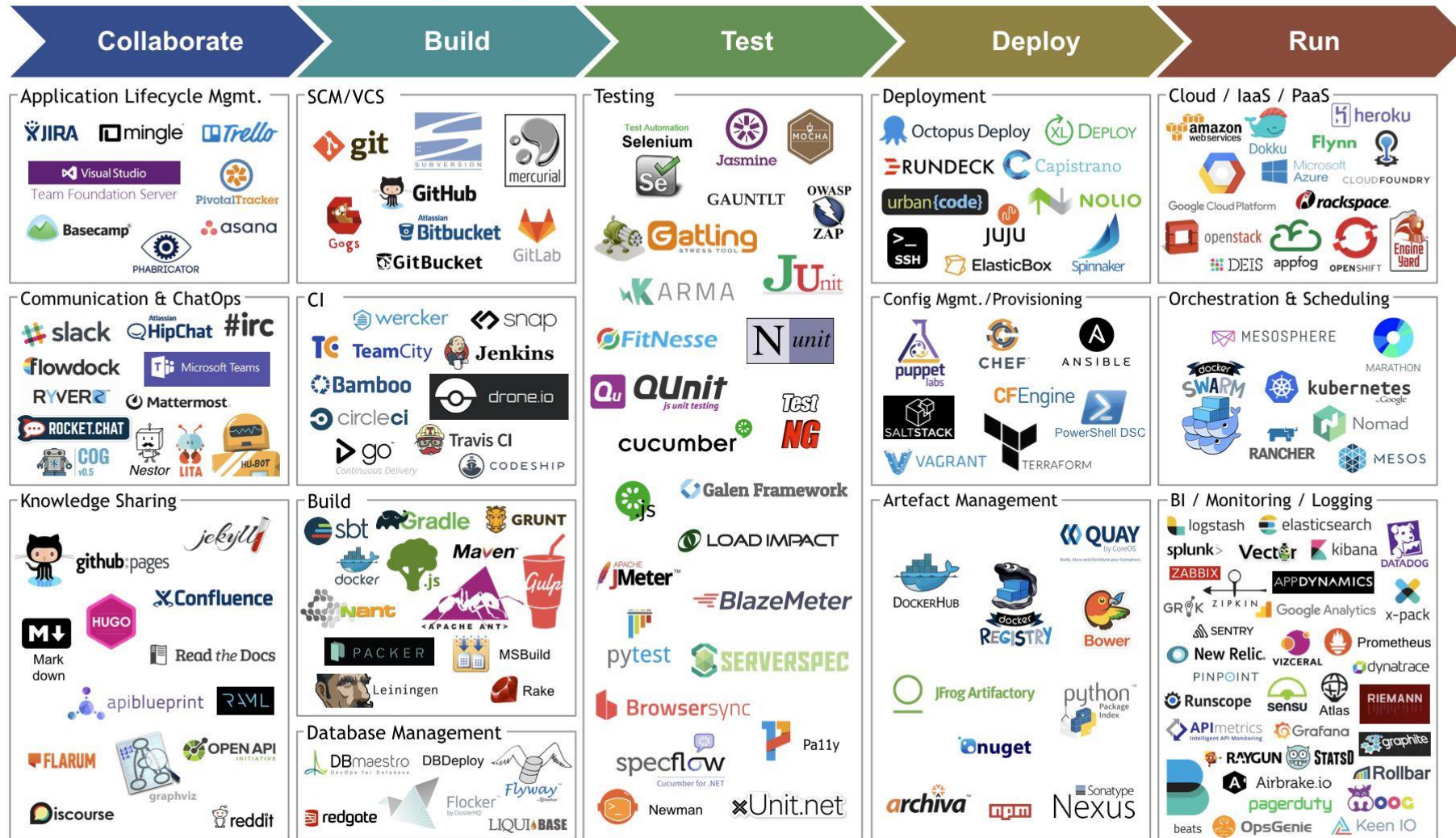


20 min

Escenario	Proceso	Elementos clave	Herramientas
<un escenario>			
<un escenario>			
<un escenario>			
<un escenario>			
<un escenario>			



Herramientas CI/CD



te de la imagen:
ppmmg.com

Hay literalmente decenas de herramientas para CI/CD,
varias opciones para cada etapa.

OBJETIVO

Revisión de algunas herramientas de CI/CD

INSTRUCCIONES

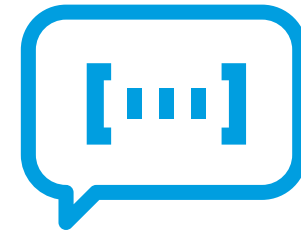
1. Mira el detalle del mapa de herramientas CI/CD:
 - <https://i.pinimg.com/originals/d2/08/84/d208846d818b1ec49acc8e2e4a01858f.jpg>
2. En cada uno de los bloques (como Ci, Build , Artefact gestión ,...) , busca en Google al menos dos herramientas que no conozca.



15 mín

02

Jenkins



Jenkins



- Es un **servidor de Integración Continua de código abierto** escrito en Java para orquestar una **cadena de acciones** para lograr el proceso de Integración Continua de forma automatizada.
- Jenkins **soporta el ciclo de vida completo de desarrollo** del software, desde la creación, las pruebas, la documentación del software, el despliegue y otras etapas del ciclo de vida del desarrollo del software.
- Es una aplicación ampliamente utilizada en todo el mundo.
- **Las empresas pueden acelerar su proceso de desarrollo de software**, ya que Jenkins puede automatizar el build y las pruebas a un ritmo rápido.

OBJETIVO

Accediendo a Jenkins

INSTRUCCIONES

1. Abra su navegador e ingrese la siguiente URL:
 1. `https://localhost 8080`
2. Desbloquea Jenkins siguiendo las instrucciones (ingresando la **contraseña inicial de administrador**).
3. Instala los plugins sugeridos.
4. Finaliza el proceso de configuración inicial.
5. Revisa la interfaz.



15 min

plugins de Jenkins



- Jenkins tiene **un excelente soporte para plugins**. Hay miles de plugins de aplicaciones de terceros disponibles en su sitio web:
<https://plugins.jenkins.io/>
- Jenkins **necesita tener** instalado el plugin GitHub para poder extraer código del repositorio de GitHub.
 - No necesitas instalar un plugin de GitHub si ya lo instalaste en respuesta al mensaje durante la configuración de instalación de Jenkins.

OBJETIVO

Instalación del plugin Git en Jenkins

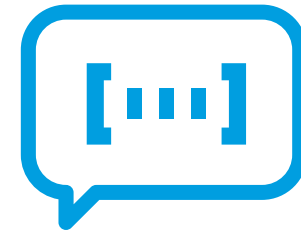
INSTRUCCIONES

1. Sigue este tutorial de Guru99 para instalar el plugin git en Jenkins:
 - [https://www.guru99.com/jenkins-github-integration.html#how to install git plugin in jenkins](https://www.guru99.com/jenkins-github-integration.html#how_to_install_git_plugin_in_jenkins)



15 min

03




Proyectos Freestyle


Proyecto de estilo libre

Enter an item name


» This field cannot be empty, please enter a valid name

**Freestyle project**


This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Pipeline**


Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**External Job**

This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

**Multi-configuration project**

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



- Es una **tarea de compilación repetible**, script o pipeline repetible que contiene pasos y acciones post-compilación.
- Es un trabajo o tarea mejorada que **puede abarcar múltiples operaciones**.
- Permite **configurar triggers de compilación** y ofrece seguridad a nivel de proyecto.
- También **ofrece plugins** para ayudar en los pasos de compilación y las acciones post compilación.

Secciones del proyecto de estilo libre

The screenshot shows the 'General' tab of a project configuration interface. At the top, there are tabs for 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build', and 'Post-build Actions'. The 'General' tab is active. Below the tabs, there is a 'Description' field with the text 'This is a demo freestyle project.' and a '[Plain text] Preview' button. Below the description field, there is a list of checkboxes for various project settings: 'Commit agent's Docker container', 'Define a Docker template', 'Discard old builds', 'GitHub project', 'This build requires lockable resources', 'This project is parameterized', 'Throttle builds', 'Disable this project', and 'Execute concurrent builds if necessary'. Each checkbox has a blue question mark icon to its right. At the bottom right of the 'General' tab, there is an 'Advanced...' button.

Visitemos las secciones del proyecto de estilo libre y su utilidad:

- 1. General**
- 2. Gestión de código fuente**
- 3. Triggers de compilación**
- 4. Entorno de compilación**
- 5. Build**
- 6. Acciones post-compilación**

OBJETIVO

Creando un proyecto de estilo libre de Jenkins

INSTRUCCIONES

1. Sigue este tutorial de dwops.com para crear un proyecto de estilo libre en Jenkins y generar un build para una fuente de código Java:
 - <https://dwops.com/courses/beginners-jenkins-tutorial-basics-freestyle-projects-ci-cd-pipeline/lesson/creating-a-freestyle-project-using-a-github-repository/>



20 min

OBJETIVO

Configurar notificaciones email

INSTRUCCIONES

1. Sigue los siguientes pasos para configurar notificaciones email cuando se hace build de un proyecto en Jenkins usando una cuenta de Gmail.
2. Recuerda habilitar el **Acceso a aplicaciones menos segura**. Vaya a esta página [aquí](#).



20 min

Ve a Manage Jenkins > Configure System, luego busque Notificación por email extendida y completa los siguientes detalles:

1. ✓ Servidor SMTP – smtp.gmail.com
2. ✓ Puerto SMTP – 465
 - Haga clic en avanzado debajo del cuadro Puerto SMTP.
3. ✓ Nombre de usuario SMTP: cualquier identificación de Gmail. Esta cuenta de Gmail se utilizará para enviar notificaciones por correo electrónico.
4. ✓ Contraseña SMTP: contraseña de la identificación de Gmail utilizada anteriormente
5. ✓ Marcar la casilla Usar SSL.
6. ✓ Destinatario predeterminado: agregar una identificación de correo electrónico predeterminada.



OBJETIVO

Crear un trabajo parametrizado

INSTRUCCIONES

1. Vamos a crear un trabajo de Jenkins que recibirá información del menú desplegable utilizando parámetros.



20 min

En el Trabajo de estilo libre > Sección General:

1. Crear parámetros (en el menú desplegable).
 1. ✓ Marcar "Este proyecto está parametrizado" en la sección General
 2. ✓ Haga clic en Agregar parámetro.
 3. ✓ Elija el parámetro de elección
2. Configurar el parámetro de elección
 1. ✓ Nombre: proporcione un nombre adecuado para su parámetro. Por ej. SO
 2. ✓ Opciones: proporcione los valores uno por línea que aparecerán en la opción desplegable.
 3. ✓ Descripción: describe el parámetro.
3. Configurar el paso de compilación
 1. ✓ Elija Ejecutar Shell e imprima algo usando el parámetro. Por ej.
echo 'Jenkins está instalado en ' \$OS.
4. Hacer build del proyecto y elegir el parámetro.
5. Verificar la salida de la consola



OBJETIVO

Concatenar varios proyectos de estilo libre

INSTRUCCIONES

1. Sísigue las siguientes instrucciones para generar un proyecto de estilo libre de varios pasos.

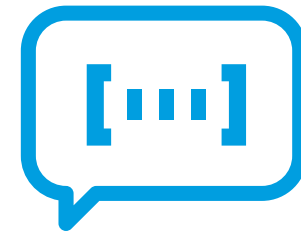


20 min

1. Genera un proyecto de estilo libre (Paso 1) que clone el siguiente repositorio:
 - <https://github.com/ricardoahumada/SimuladorCoches>
 - Cambiar a una carpeta independiente en el repositorio clonado.
2. Generar otro proyecto de estilo libre (Paso 2) que ejecute el siguiente orden en Shell:
 - `mvn clean test`
 - Vuelve al proyecto Paso 1 y agrega "Acciones posteriores a el build" que apunte al Paso 2.
3. Genera un proyecto final de estilo libre (Paso 3) que ejecute el siguiente orden en Shell:
 - `mvn package`
 - Vuelve al proyecto Paso 3 y agregue "Acciones posteriores a el build" que apunte al Paso 3.
4. Build del proyecto
5. Verificar la salida de la consola

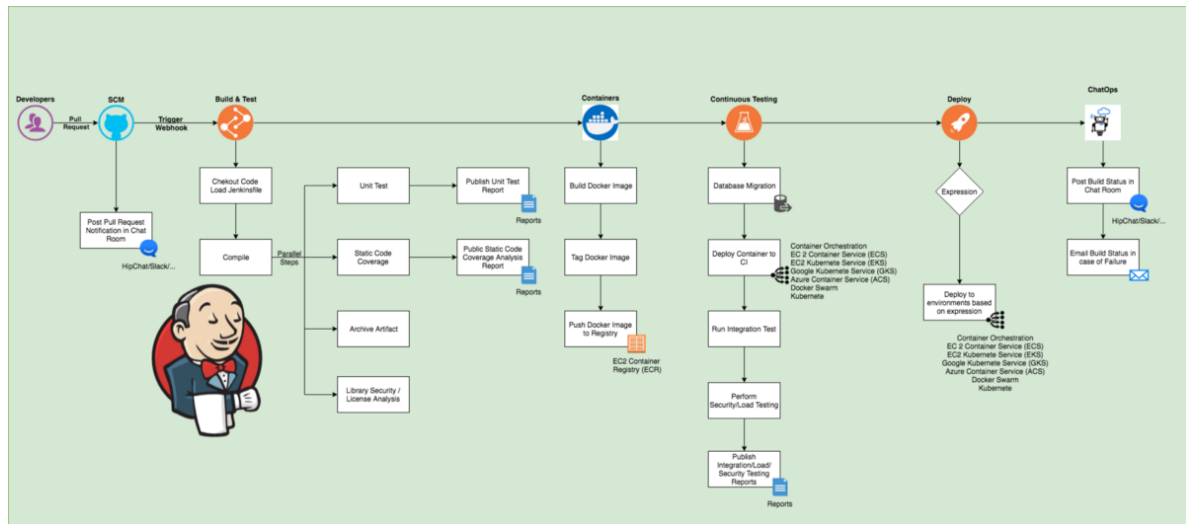


04



Pipelines Jenkins

pipelines Jenkins



Fuente de la imagen: medium.com

- Jenkins Pipeline (o simplemente "Pipeline" con "P" mayúscula) es un **conjunto de plugins** que admite la implementación e integración de pipelines de entrega continua en Jenkins.
- Pipeline **proporciona un conjunto extensible de** herramientas para modelar pipelines de entrega de simples a complejas **"como código"** a través de la sintaxis del lenguaje específico de dominio de Pipeline (**DSL**).
- La definición de Jenkins Pipeline se codifica en un archivo de texto (llamado **Jenkinsfile**) que a su vez se puede enviar al repositorio de control de código fuente de un proyecto.
- Ésta es la base de **" Pipeline-as-code "**; Tratar el pipeline de CI/CD como parte de la aplicación para ser versionada y revisada como cualquier otro código.

Conceptos pipeline

- **Pipeline:** bloque que contiene todos los procesos como build, pruebas, implementación, etc. Es una colección de todas las etapas en un Jenkinsfile.
- **Node:** un nodo es una máquina que ejecuta un workflow completo. Es una parte clave de la sintaxis de la pipeline con script.
- **Agent:** indica a Jenkins que asigne un ejecutor para las compilaciones. Ayuda a distribuir la carga de trabajo a diferentes agentes y ejecutar varios proyectos dentro de una única instancia de Jenkins. Puede ser Cualquiera, Ninguno, Etiqueta, Docker.
- **Stages:** El trabajo se concreta en forma de etapas. Cada etapa realiza una tarea específica.
- **Steps:** Se pueden definir una serie de pasos dentro de un bloque de etapa. Estos pasos se llevan a cabo en secuencia para ejecutar una etapa.

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Building..'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing..'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying....'
      }
    }
  }
}
```

Archivo Jenkins

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Building..'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing..'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying....'
      }
    }
  }
}
```

- Un Jenkinsfile se puede escribir usando dos tipos de sintaxis: **declarativa** y en **secuencia de comandos (scripted)**.
- Las pipelines declarativas y scripted se construyen de forma fundamentalmente diferente.
- **Pipeline Declarativa** es una característica más reciente de Jenkins Pipeline que:
 - proporciona **características sintácticas más ricas** que la sintaxis de Scripted Pipeline, y
 - está diseñado para **facilitar la escritura y lectura** del código Pipeline.
- Muchos de los componentes sintácticos individuales (o "steps") escritos en un Jenkinsfile son comunes tanto para Declarative como para Scripted Pipeline.

OBJETIVO

Proyectos de pipeline scripted y declarativos

INSTRUCCIONES

1. Sigue los siguientes pasos para crear ejemplos de proyectos de pipeline scripted y declarativos.



15 min

1. Crea un nuevo Item de tipo pipeline.
 - Elige '**pipeline script**' para scripted pipeline.
 - Luego copiar y pegar el código anterior.
 - Hazbuild del proyecto
 - Verifica la salida de la consola

2. Crea otro Item de tipo pipeline.
 - Selecciona '**pipeline script from SCM**' , para pipeline declarativa.
 - Elige el SCM: <https://github.com/Zulaikha12/gitnew.git>
 - Haz build del proyecto
 - Verifica la salida de la consola



Pipeline declarativa

- En la sintaxis de Declarative Pipeline, el bloque de pipeline define todo el trabajo realizado a lo largo de todo el pipeline.

```
pipeline {
  agent any
  stages {
    stage('One') {
      steps {
        echo 'Hi, this is Zulaikha from edureka'
      }
    }
    stage('Two') {
      steps {
        input('Do you want to proceed?')
      }
    }
    stage('Three') {
      when {
        not {
          branch "master"
        }
      }
      steps {
        echo "Hello"
      }
    }
    stage('Four') {
      parallel {
        stage('Unit Test') {
          steps {
            echo "Running the unit test..."
          }
        }
        stage('Integration test') {
          agent {
            docker {
              reuseNode true
              image 'ubuntu'
            }
          }
          steps {
            echo "Running the integration test..."
          }
        }
      }
    }
  }
}
```

OBJETIVO

Probando el proyecto de pipeline declarativo

INSTRUCCIONES

1. Crea un nuevo Item de tipo pipeline.
2. Copia y pega el código anterior.
3. Haz build del proyecto
4. Verifica la salida de la consola



5 min

Scripted Pipeline

- En la sintaxis de Scripted Pipeline, uno o más bloques de nodos realizan el trabajo principal en todo el Pipeline.
- Aunque esto no es un requisito obligatorio de la sintaxis de Scripted Pipeline, limitar el trabajo del pipeline dentro de un bloque de nodos logra dos cosas:
 - Programar la ejecución de los pasos contenidos en el bloque agregando un elemento a la cola de Jenkins. Tan pronto como un ejecutor esté libre en un nodo, se ejecutarán los pasos.
 - Crear un espacio de trabajo (un directorio específico para esa pipeline en particular) donde se puede trabajar en archivos extraídos del control de código fuente.

```
node {
    for (i=0; i<2; i++) {
        stage "Stage #"+i
        print 'Hello, world !'
        if (i==0)
        {
            git "https://github.com/Zulaikha12/gitnew.git"
            echo 'Running on Stage #0'
        }
        else {
            build 'Declarative pipeline'
            echo 'Running on Stage #1'
        }
    }
}
```

OBJETIVO

Probando un proyecto de pipeline scripted

INSTRUCCIONES

1. Crea un nuevo artículo de tipo pipeline.
2. Copia y pega el anterior código.
3. Haz build del proyecto
4. Verifica la salida de la consola



5 min

OBJETIVO

Comparación de pipelines declarativas y scripted

INSTRUCCIONES

1. Mira el siguiente código para el dos tipos de pipelines. Son equivalentes. _
2. Cual uno es declarativo y que uno es scripted?
3. Qué hace este código?



5 min

```

pipeline {
  agent any
  options {
    skipStagesAfterUnstable()
  }
  stages {
    stage('Build') {
      steps {
        sh 'make'
      }
    }
    stage('Test'){
      steps {
        sh 'make check'
        junit 'reports/**/*.xml'
      }
    }
    stage('Deploy') {
      steps {
        sh 'make publish'
      }
    }
  }
}

```

```

node {
  stage('Build') {
    sh 'make'
  }
  stage('Test') {
    sh 'make check'
    junit 'reports/**/*.xml'
  }
  if (currentBuild.currentResult == 'SUCCESS') {
    stage('Deploy') {
      sh 'make publish'
    }
  }
}

```



Usando variables de entorno

- Jenkins Pipeline expone variables de entorno a través de la variable global **env** , que está disponible desde cualquier lugar dentro de un Jenkinsfile.

```
pipeline {
  agent any
  stages {
    stage('Example') {
      steps {
        echo "Running ${env.BUILD_ID} on ${env.JENKINS_URL}"
      }
    }
  }
}
```


Usando variables de entorno

La lista completa de variables de entorno accesibles desde Jenkins Pipeline está documentada en **`${YOUR_JENKINS_URL}/pipeline-syntax/globals#env`**.

BUILD_ID

The current build ID, identical to BUILD_NUMBER for builds created in Jenkins versions 1.597+

BUILD_NUMBER

The current build number, such as "153"

BUILD_TAG

String of jenkins-\${JOB_NAME}-\${BUILD_NUMBER}. Convenient to put into a resource file, a jar file, etc for easier identification

BUILD_URL

The URL where the results of this build can be found (for example `http://buildserver/jenkins/job/MyJobName/17/`)

EXECUTOR_NUMBER

The unique number that identifies the current executor (among executors of the same machine) performing this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1

JAVA_HOME

If your job is configured to use a specific JDK, this variable is set to the JAVA_HOME of the specified JDK. When this variable is set, PATH is also updated to include the bin subdirectory of JAVA_HOME

JENKINS_URL

Full URL of Jenkins, such as `https://example.com:port/jenkins/` (NOTE: only available if Jenkins URL set in "System Configuration")

JOB_NAME

Name of the project of this build, such as "foo" or "foo/bar".

NODE_NAME

The name of the node the current build is running on. Set to 'master' for the Jenkins controller.

WORKSPACE

The absolute path of the workspace

Configuración de variables de entorno

- La configuración de una variable de entorno dentro de una pipeline de Jenkins se logra de manera diferente dependiendo de si se utiliza una pipeline declarativa o scripted.
- Los pipelines declarativos admiten una **directiva de entorno** , mientras que los Scripted Pipeline deben usar el step **withEnv**.

```
pipeline {
  agent any
  environment {
    CC = 'clang'
  }
  stages {
    stage('Example') {
      environment {
        DEBUG_FLAGS = '-g'
      }
      steps {
        sh 'printenv'
      }
    }
  }
}
```

```
node {
  /* .. snip .. */
  withEnv(["PATH+MAVEN=${tool 'M3'}/bin"]) {
    sh 'mvn -B verify'
  }
}
```

Establecer variables de entorno dinámicamente

- Las variables de entorno se pueden configurar en tiempo de ejecución y pueden ser utilizadas por scripts de Shell (sh), scripts por lotes de Windows (bat) y scripts de PowerShell (powershell).
- Cada script puede usar **returnStatus** o **returnStdout**.
- Más información sobre scripts en:
 - <https://www.jenkins.io/doc/pipeline/steps/workflow-durable-task-step/>
- A continuación se muestra un ejemplo en una pipeline declarativa que utiliza sh (shell) con returnStatus y returnStdout.

```
pipeline {
  agent any
  environment {
    // Using returnStdout
    CC = """${sh(
      returnStdout: true,
      script: 'echo "clang"'
    )}"""
    // Using returnStatus
    EXIT_STATUS = """${sh(
      returnStatus: true,
      script: 'exit 1'
    )}"""
  }
  stages {
    stage('Example') {
      environment {
        DEBUG_FLAGS = '-g'
      }
      steps {
        sh 'printenv'
      }
    }
  }
}
```

Manejo de credenciales

- Las credenciales configuradas en Jenkins se pueden acceder en los pipelines para su uso inmediato.
- **Para textos secretos, nombres de usuario y contraseñas, y archivos secretos**
 - La sintaxis declarativa de Pipeline de Jenkins tiene el método auxiliar **credenciales()** (utilizado dentro de la directiva de entorno) que admite texto secreto, nombre de usuario y contraseña, así como credenciales de archivos secretos.
- **Texto secreto**
 - El siguiente código de Pipeline muestra un ejemplo de cómo crear un Pipeline utilizando variables de entorno para credenciales de texto secreto.

```
pipeline {
    agent {
        // Define agent details here
    }
    environment {
        AWS_ACCESS_KEY_ID = credentials('jenkins-aws-secret-key-id')
        AWS_SECRET_ACCESS_KEY = credentials('jenkins-aws-secret-access-key')
    }
    stages {
        stage('Example stage 1') {
            steps {
                echo $AWS_SECRET_ACCESS_KEY
            }
        }
        stage('Example stage 2') {
            steps {
                //
            }
        }
    }
}
```

Nombres de usuario y contraseñas

Este código establece las siguientes tres variables de entorno:

- **BITBUCKET_COMMON_CREDS:** contiene un nombre de usuario y una contraseña separados por dos puntos en el formato nombre de usuario:contraseña.
- **BITBUCKET_COMMON_CREDS_USR :** una variable adicional que contiene únicamente el componente de nombre de usuario.
- **BITBUCKET_COMMON_CREDS_PSW:** una variable adicional que contiene únicamente el componente de contraseña.

```
environment {  
    BITBUCKET_COMMON_CREDS = credentials('jenkins-bitbucket-common-creds')  
}  
...  
  
echo $BITBUCKET_COMMON_CREDS  
  
echo $BITBUCKET_COMMON_CREDS_USR  
  
echo $BITBUCKET_COMMON_CREDS_PSW
```

Archivos secretos

Un archivo secreto es una credencial que se almacena en un archivo y se carga en Jenkins. Los archivos secretos se utilizan para credenciales que son:

- demasiado difícil de manejar para entrar directamente en Jenkins, y/o
- en formato binario, como un archivo GPG.

```
pipeline {
  agent {
    // Define agent details here
  }
  environment {
    // The MY_KUBECONFIG environment variable will be assigned
    // the value of a temporary file. For example:
    // /home/user/.jenkins/workspace/cred_test@tmp/secretFiles/546a5cf3-9b56-4165-a0fd-19e2afe6b31f/kubeconfig.txt
    MY_KUBECONFIG = credentials('my-kubeconfig')
  }
  stages {
    stage('Example stage 1') {
      steps {
        sh("kubectl --kubeconfig $MY_KUBECONFIG get pods")
      }
    }
  }
}
```

OBJETIVO

SSH en Jenkins

INSTRUCCIONES

1. Crea un nombre de usuario/contraseña para conectar al servidor ssh de tu compañero.
2. Genera un archivo Jenkins para cargar y ejecutar un jar en el servidor.
3. Revisar el referencia en:
 - <https://www.jenkins.io/doc/pipeline/steps/ssh-steps/>



20 min

Interpolación de strings

- La interpolación de strings Groovy puede resultar confuso para muchos recién llegados al lenguaje.
- Groovy soporta la declaración de una cadena con comillas simples o dobles.
- **¡La interpolación de cadenas maravillosa nunca debe usarse con credenciales!**

```
def username = 'Jenkins'
echo 'Hello Mr. ${username}'
echo "I said, Hello Mr. ${username}"
```

Would result in:

```
Hello Mr. ${username}
I said, Hello Mr. Jenkins
```

```
pipeline {
    agent any
    environment {
        EXAMPLE_CREDS = credentials('example-credentials-id')
    }
    stages {
        stage('Example') {
            steps {
                /* WRONG! */
                sh("curl -u ${EXAMPLE_CREDS_USR}:${EXAMPLE_CREDS_PSW} https://example.com/")
            }
        }
    }
}
```


Manejo de parámetros

- Los Pipeline Declarativos admiten parámetros listos para usar, lo que permite que Pipeline acepte parámetros especificados por el usuario en tiempo de ejecución a través de la directiva de parámetros.
- La configuración de parámetros con Scripted Pipeline se realiza con el paso de propiedades, que se puede encontrar en el Generador de fragmentos.
- Si configuraste tu pipeline para aceptar parámetros usando la opción Build with Parameters, se puede acceder a esos parámetros como miembros de la variable params.
- Suponiendo que se ha configurado un parámetro de cadena llamado "Greeting" en Jenkinsfile , se puede acceder a ese parámetro a través de `${params.Greeting}`:

```
pipeline {
    agent any
    parameters {
        string(name: 'Greeting', defaultValue:
'Hello', description: 'How should I greet the
world?')
    }
    stages {
        stage('Example') {
            steps {
                echo "${params.Greeting} World!"
            }
        }
    }
}
```

Gestión de fallos

- Los Pipeline Declarativos admiten una gestión sólida de fallos de forma predeterminada a través de **post section**, que permite declarar una serie de "condiciones de publicación" diferentes, como:
 - always, unstable, success, failure, and changed.
- La sección Sintaxis de pipeline proporciona más detalles sobre cómo utilizar las distintas condiciones de post.

```
pipeline {
  agent any
  stages {
    stage('Test') {
      steps {
        sh 'make check'
      }
    }
  }
  post {
    always {
      junit '**/target/*.xml'
    }
    failure {
      mail to: team@example.com, subject: 'The Pipeline failed :('
    }
  }
}
```

Usando múltiples agentes

- Pipeline permite utilizar múltiples agentes en el entorno Jenkins desde el mismo Jenkinsfile, lo que puede ser útil para casos de uso más avanzados, como la ejecución de compilaciones/pruebas en múltiples plataformas.

```
pipeline {
  agent none
  stages {
    stage('Build') {
      agent any
      steps {
        checkout scm
        sh 'make'
        stash includes: '**/target/*.jar', name: 'app'
      }
    }
    stage('Test on Linux') {
      agent {
        label 'linux'
      }
      steps {
        unstash 'app'
        sh 'make check'
      }
      post {
        always {
          junit '**/target/*.xml'
        }
      }
    }
    stage('Test on Windows') {
      agent {
        label 'windows'
      }
      steps {
        unstash 'app'
        bat 'make check'
      }
      post {
        always {
          junit '**/target/*.xml'
        }
      }
    }
  }
}
```

Argumentos de pasos opcionales

- Pipeline sigue la convención del lenguaje Groovy de permitir que se omitan los paréntesis alrededor de los argumentos del método.
- Muchos pasos de Pipeline también utilizan la **sintaxis de parámetro con nombre** como abreviatura para crear un mapa en Groovy, que utiliza la sintaxis **[clave1: valor1, clave2: valor2]**.
- Por conveniencia, al llamar a pasos que toman solo un parámetro (o solo un parámetro obligatorio), se puede omitir el nombre del parámetro.

```
git url: 'git://example.com/amazing-project.git', branch: 'master'  
git([url: 'git://example.com/amazing-project.git', branch: 'master'])
```

```
sh 'echo hello' /* short form */  
sh([script: 'echo hello']) /* long form */
```

Ejecución paralela

- Pipeline tiene una funcionalidad incorporada para ejecutar partes de Scripted Pipeline en paralelo, implementada en el paso paralelo, apropiadamente llamado.
- Ejemplo: en lugar de ejecutar las pruebas en los nodos "linux" y "windows" en serie, ahora se ejecutarán en paralelo asumiendo que existe la capacidad necesaria en el entorno Jenkins.

```
stage('Build') {
    /* .. snip .. */
}

stage('Test') {
    parallel linux: {
        node('linux') {
            checkout scm
            try {
                unstash 'app'
                sh 'make check'
            }
            finally {
                junit '**/target/*.xml'
            }
        }
    },
    windows: {
        node('windows') {
            /* .. snip .. */
        }
    }
}
```

Programando jobs en Jenkins



- La función de programación permite programar trabajos para que se ejecuten automáticamente fuera del horario laboral o en tiempos de inactividad.
- Programar trabajos puede ayudar a escalar el entorno a medida que aumenta el uso de Jenkins. El siguiente vídeo proporciona información sobre la función de programación y sus diversas opciones de configuración.

OBJETIVO

Programar un trabajo cada 10 min

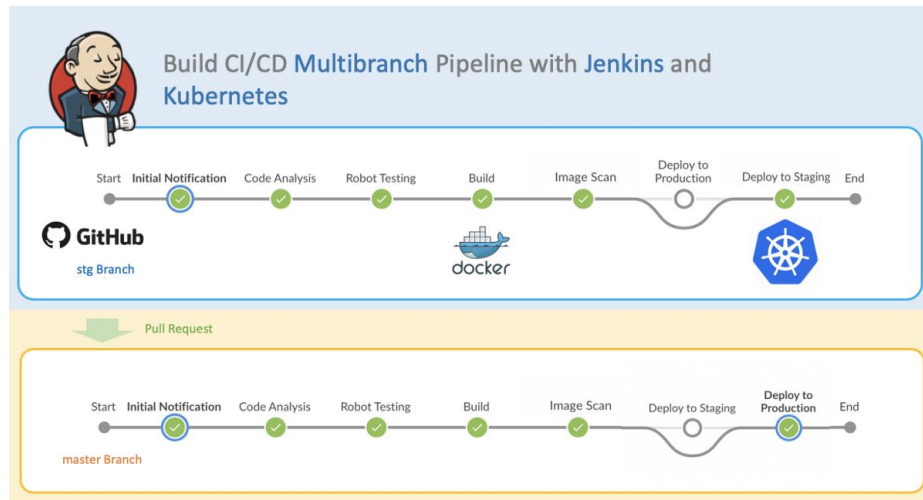
INSTRUCCIONES

1. Mira el siguiente vídeo sobre cómo para programar trabajos en Jenkins:
 - <https://www.youtube.com/watch?v=JhvVJtYFUm0>
2. Programa un trabajo para que se ejecute cada 10 min.



20 min

Branches y Pull Requests



Fuente de la imagen: [peiruwang.medium.com](https://medium.com/@peiruwang)

- El tipo de proyecto Multibranch Pipeline permite implementar diferentes Jenkinsfiles para diferentes ramas del mismo proyecto.
- En un proyecto Multibranch Pipeline, Jenkins descubre, administra y ejecuta automáticamente Pipelines para ramas que contienen un archivo Jenkins en el control de código fuente.

OBJETIVO

Generando un Pipeline multirama

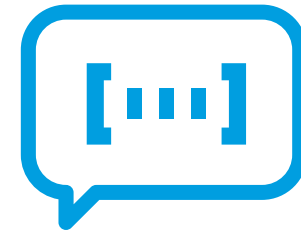
INSTRUCCIONES

1. Sigue los pasos del siguiente artículo para crear un pipeline multirama en Jenkins:
 - <https://www.jenkins.io/doc/book/pipeline/multibranch/>
2. Usa tus ramas de repositorio para crear un pipeline multirama.



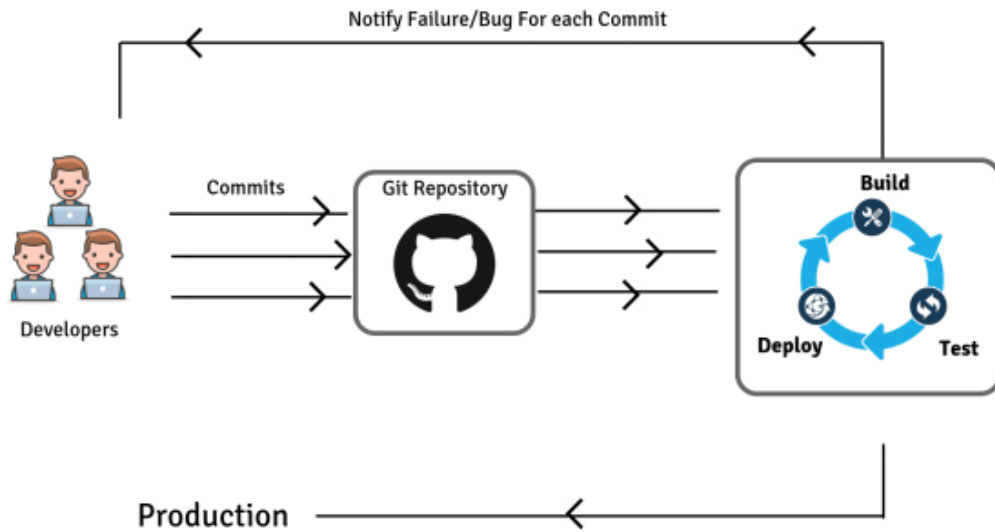
20 min

05



Integración continua con Jenkins y GitHub

Qué es un WebHook?

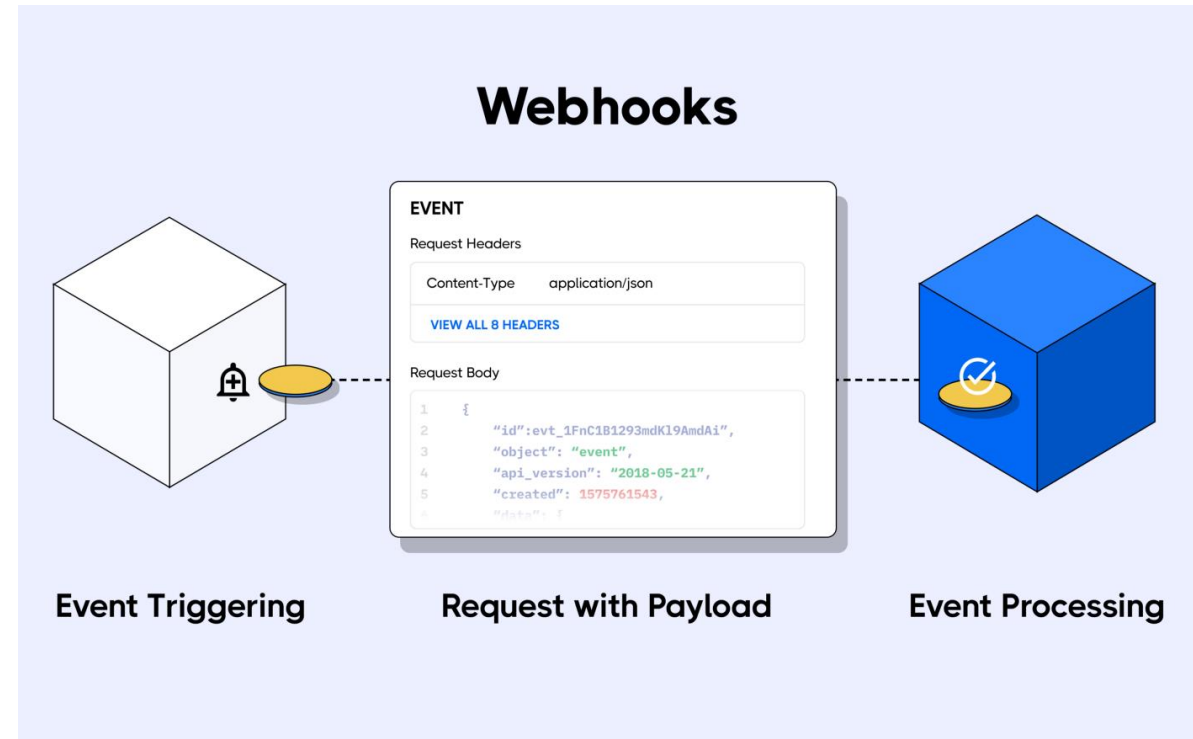


Fuente de la imagen: qatouch.com

- Uno de los pasos básicos para implementar CI/CD es **integrar la herramienta de gestión de control de código fuente (SCM) con la herramienta de CI.**
- Esto **ahorra tiempo y mantiene el proyecto actualizado** todo el tiempo.
- Queremos **activar automáticamente cada compilación** en el servidor Jenkins, después de que cada **commit** o **pull request** se fusione en el repositorio Git.

Qué es un WebHook?

- Si estás familiarizado con las API RESTful, los webhooks son **una solicitud POST** realizada por un servicio web a otro servicio.
- En nuestro caso, el servicio web que crea esta solicitud es **GitHub** y el servicio donde se realiza esta solicitud es nuestro servidor jenkins.



Fuente de la imagen: medium.com

OBJETIVO

Configurando el proyecto de estilo libre de Jenkins

INSTRUCCIONES

1. Sigue las siguientes instrucciones para configurar Jenkins para recibir llamadas usando webhooks.



10 min

1. Crea un nuevo Item freestyle.
2. Ve a la pestaña "Administración de código fuente" para cambiar la configuración de **Jenkins GitHub Webhook**.
3. Muévete a la sección "Administración de código fuente", elige la opción "**Git**" para informarle a Jenkins que el nuevo trabajo es para Jenkins GitHub Webhook.
4. **Pega la URL** del repositorio deseado en el campo de "URL del repositorio".
5. Ahora, ve a la pestaña "**Build Triggers**" donde se puede configurar qué acción debe realizar Jenkins GitHub Webhook si Jenkins detecta algún cambio en el repositorio de GitHub.
6. Aquí, elige la opción "**GitHub hook trigger for GITScm pulling**" , que escuchará los triggers del repositorio de GitHub proporcionado.
7. Ahora, haz clic en el botón "Aplicar" para guardar los cambios y crea un nuevo Webhook Jenkins GitHub para tu repositorio.



OBJETIVO

Configurar el webhook de GitHub

INSTRUCCIONES

1. Sigye los instrucciones para configurar GitHub para enviar llamadas webhook.



10 min

1. ve a la opción "Configuración" en la esquina derecha.
2. Selecciona la opción "Webhooks" y luego haz clic en el botón "Agregar Webhook".
3. Ve a tu pestaña Jenkins y copia la URL, luego péguala en el campo de texto llamado "**Payload URL**".
 1. Agregar **"/github-webhook/"** al final de la URL.
 2. La URL final tendrá este formato **" http://address:port/github-webhook/ "**.
 1. **¡La dirección debe ser una dirección IP o accesible a Internet!**
 3. Selecciona el "Content type" al formato "application/json"
4. El campo "Secreto" es opcional. Dejémoslo en blanco para este webhook de Jenkins GitHub.
5. A continuación, elige una opción en **"Which events would you like to trigger this webhook?"**.
6. Haz clic en el botón **"Add Webhook"** para guardar las configuraciones del Webhook de Jenkins GitHub.



OBJETIVO

Prueba la conexión GitHub-Jenkins

INSTRUCCIONES

1. Sigue las instrucciones para probar la conexión Github -Jenkins.



10 min

Prueba con un push simple

1. Configura el wekhook de GitHub para un push simple.
2. Ahora haz algunos cambios en tu código.
3. Haz push a GitHub
4. Verifica que el build de Jenkins se activa automáticamente.

Prueba con pull request

1. Configura el wekhook de GitHub para “pull-request merge”.
2. Ahora haz algunos cambios en tu código.
3. Entra en el proceso de pull request con su compañero
4. Verifica que el build de Jenkins se activa automáticamente.



OBJETIVO

Pruebe la conexión GitHub-Jenkins con ramas

INSTRUCCIONES

1. Crea una configuración de webhook para una rama específica en tu repositorio.



10 min

OBJETIVO

Integración del webhook de GitHub en Jenkinsfile

INSTRUCCIONES

1. Seguir las instrucciones para integrar el webhook de GitHub en un Jenkinsfile.



15 min

1. Crea un proyecto de pipeline en Jenkins
2. Elige **el script Pipeline de SCM**
3. En el menú desplegable de SCM, selecciona **Git** y, debajo, en la URL del repositorio, escribe (o pega) la URL completa del repositorio de GitHub.
4. Ahora, escribe la **ruta del script** , que será la ruta del archivo Jenkins.
 1. Agregue un script de pipeline.
5. Prueba el pipeline manualmente.
6. Haz push de algunos cambios en tu repositorio.
7. Verifica que el build de Jenkins se activa automáticamente.





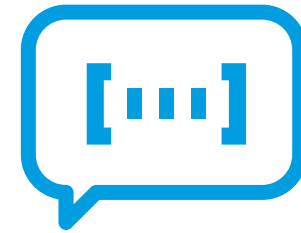
EXERCISE

Más sobre Jenkins

- Visita la página siguiente para ver más ejemplos de jenkinsfile :
 - <https://www.jenkins.io/doc/pipeline/examples/>
- Prueba algunos de ellos.



06



Prácticas de Calidad CI/CD

Continuous Testing - Pruebas continuas



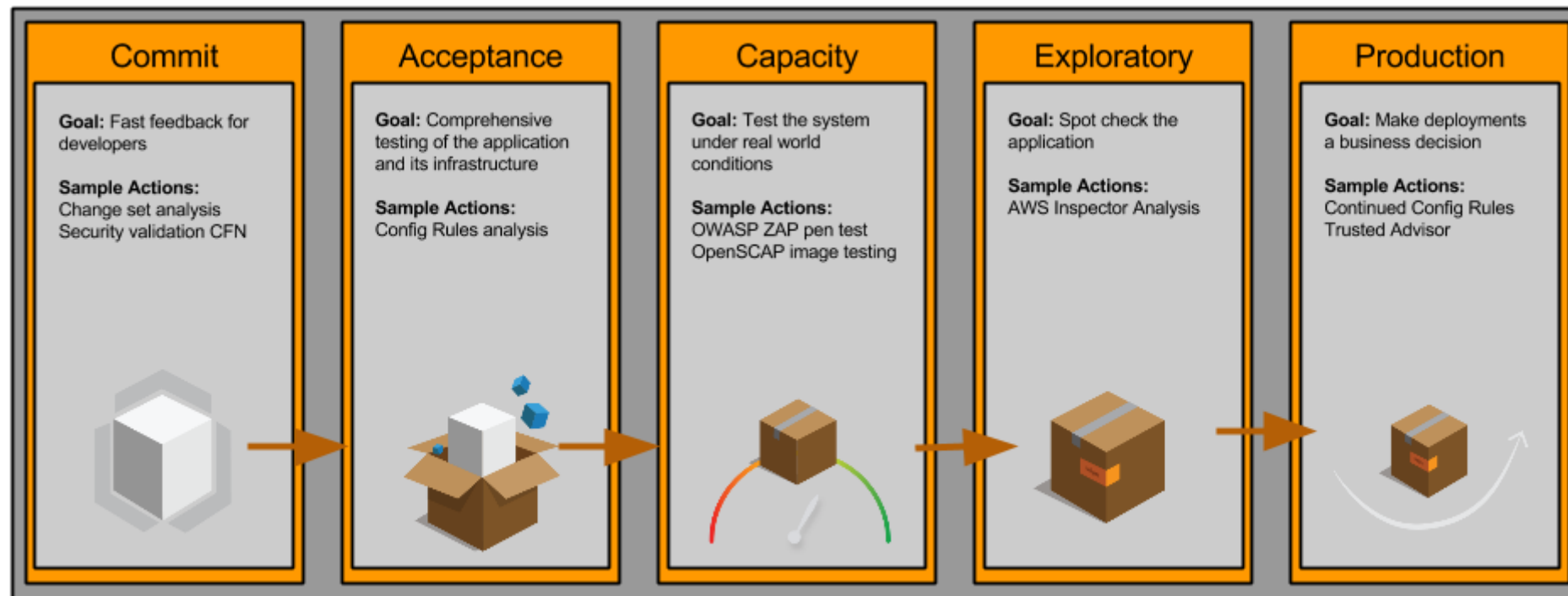
- La automatización es la piedra angular de un gran workflow de desarrollo. Cada tarea que pueda hacer una máquina debería hacerla.
- **Pero la automatización sin calidad puede aumentar el impacto de los errores, incluso llegando al cliente en el pipeline de entrega.**
- Las pruebas continuas proporcionan **información rápida y continua** sobre el estado de la última versión de la aplicación.
 - Se puede utilizar para determinar si la **aplicación está lista** para avanzar a través del pipeline de entrega en cualquier momento.
- Debido a que las pruebas comienzan a pasarse pronto en el ciclo de de entrega y se ejecutan continuamente, los errores se exponen poco después de su introducción, lo que reduce el tiempo y el esfuerzo necesario para localizarlos y solucionarlos.
- En consecuencia, es posible **aumentar la velocidad y la frecuencia** con la que se entrega software libre de errores y de alta calidad, así como disminuir la deuda técnica.

Ejecutar las pruebas continuamente

- ✓ Los **humanos** cometemos muchos errores.
 - ✓ A veces las personas consideran que si un cambio es pequeño no puede romper la aplicación, por lo que no realiza pruebas. Entonces el producto se cae (deja de funcionar) y nadie sabe por qué.
- ✓ Las pruebas automatizadas solo pueden mostrar su verdadero poder cuando las **ejecutas continuamente** y para cada cambio.
 - ✓ Deben ejecutarse en un sistema automatizado y separado para prevenir el síndrome "Funciona en mi máquina". Solo si la infraestructura de Integración continúa le indica que las pruebas pasan, las pruebas pasan.
- ✓ La ventaja de **ejecutar todas las pruebas de inmediato** para cada cambio es que se sabe de inmediato si algo se rompió.
 - ✓ Por ejemplo, si se trabaja en un sprint de 2 semanas: En el segundo día del sprint, uno de sus desarrolladores cambia algo que rompe una prueba. Si no se ejecuta continuamente todas sus pruebas, existe la posibilidad de que nadie detecte el problema. El equipo sigue trabajando, y al final del sprint, se vuelve a ejecutar todo el conjunto de pruebas. La prueba falla. Han pasado 2 semanas desde que se comenzó el cambio radical. Encontrar el cambio que rompió las pruebas es increíblemente difícil, ya que ha habido muchos compromisos desde entonces.

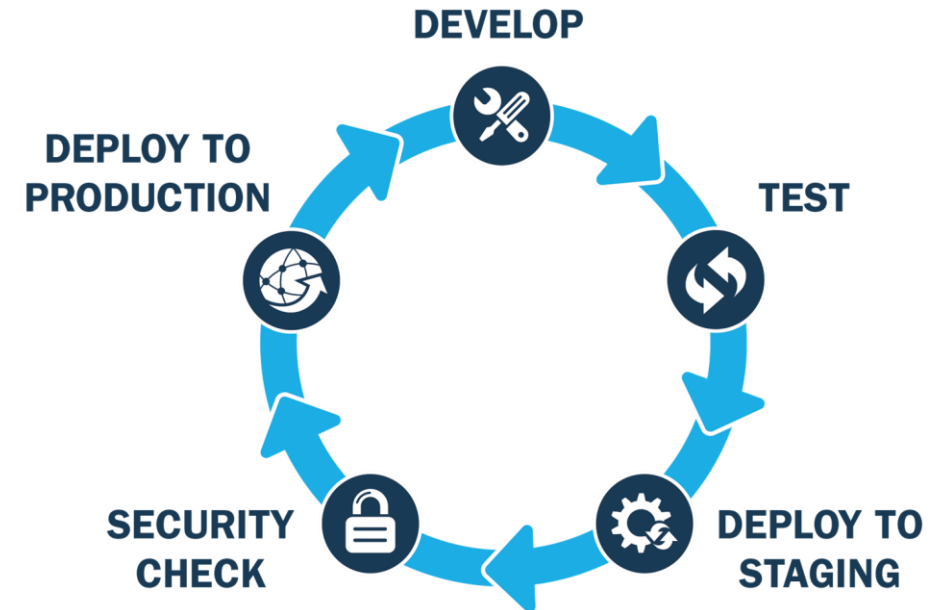
Pruebas continuas

Ejecutar las pruebas continuamente



Seguridad Continua - Seguridad continua

- *¿Qué tiene que ver esto con la seguridad? ¡Hacer!*
- La necesidad de un software seguro es primordial, de lo contrario, nuestros **medios de vida basados en la tecnología** estarán en riesgo.
- Las **violaciones de seguridad** son una de las mayores amenazas que enfrentan las organizaciones y nuestros gobiernos en la actualidad.
 - Varias organizaciones importantes han sido hackeadas en los últimos tiempos, causando grandes consecuencias y renuncias abruptas de los últimos tiempos. directivos (C-suite).
- **Integrar** las comprobaciones de seguridad automatizadas. en el oleoducto para recibir alertas tempranas.
- **Supervisar** las vulnerabilidades de seguridad sin descanso.



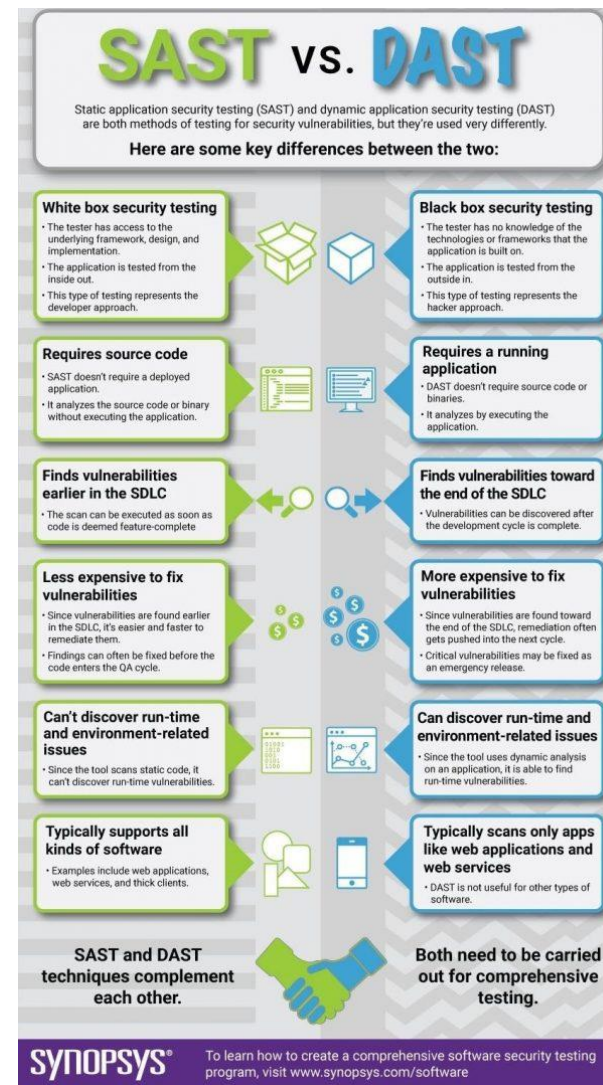
Seguridad continua: SAST y DAST

SAST

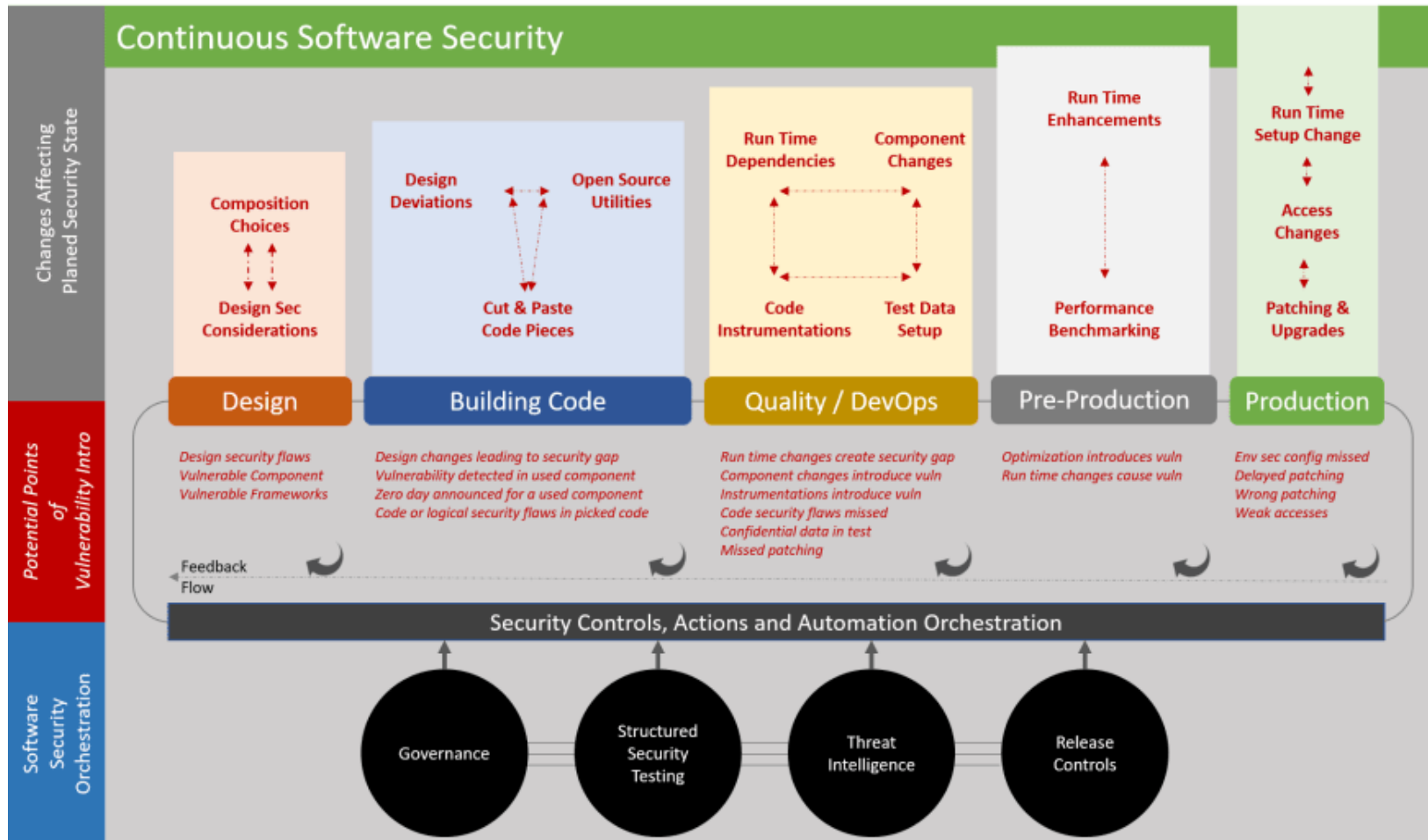
- **de seguridad de análisis estático : pruebas** de seguridad de análisis estático.
- Además de detectar violaciones de las mejores prácticas de codificación, los analizadores de código estático detectan vulnerabilidades de seguridad en el código que posee y en las bibliotecas (posiblemente inseguras) que importan.
- Las herramientas modernas se integran bien con pipelines. de entrega continua.

DAST

- Los componentes **débilmente acoplados** (acoplados débilmente) forman un subsistema. Los subsistemas se pueden implementar y probar para detectar vulnerabilidades de seguridad utilizando DAST (**dynamic analysis security testing**: pruebas de seguridad de análisis dinámico).
- A diferencia de SAST, DAST examina una aplicación desde el exterior en su estado de ejecución, muy similar a lo que haría un atacante.



Seguridad continua en el oleoducto



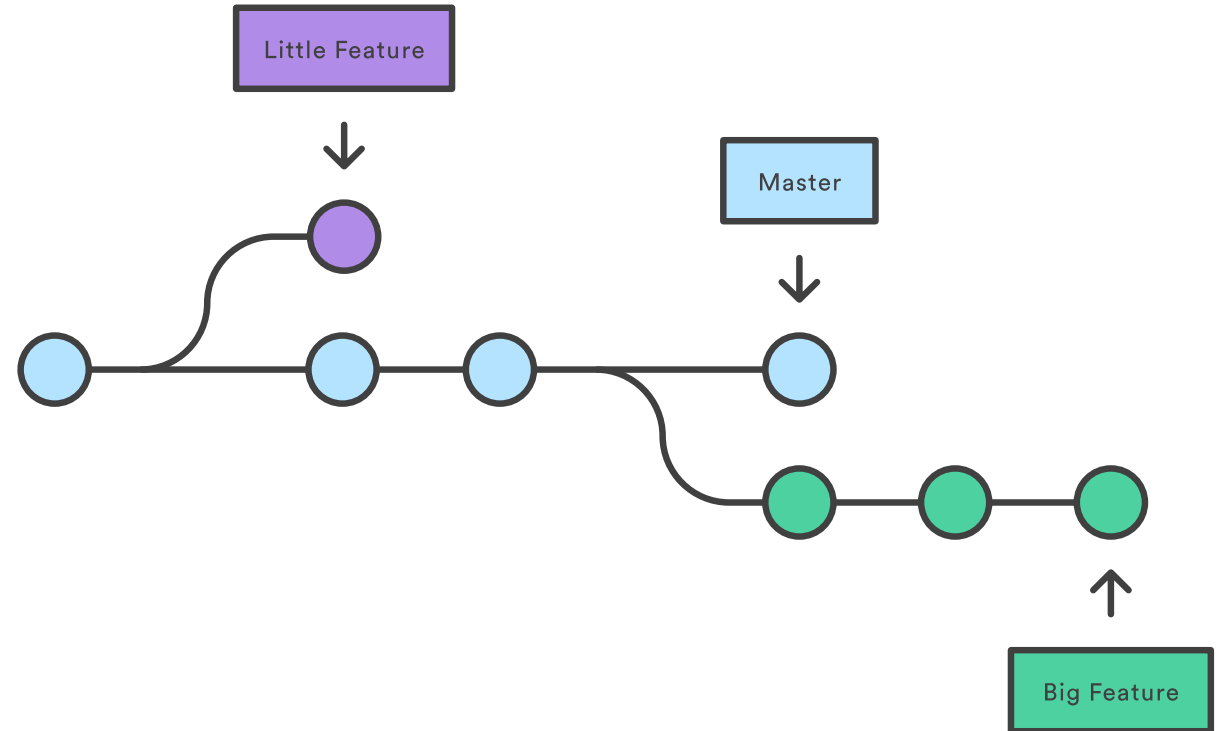
Mejores prácticas



- **Comprometerse** pronto ya menudo
- Mantener las **compilaciones en verde**
- Compilador **una sola vez**
- **simplificar** las pruebas
- **Revisar** cambios antes de comprometerse
- Usar **Ramas**
- **Limpiar** los entornos
- Convertir CI/CD en el **único modo** de implementar una producción
- **Monitorizar y medir** su proceso
- **Acordar** el workflow / Trabajar como un equipo

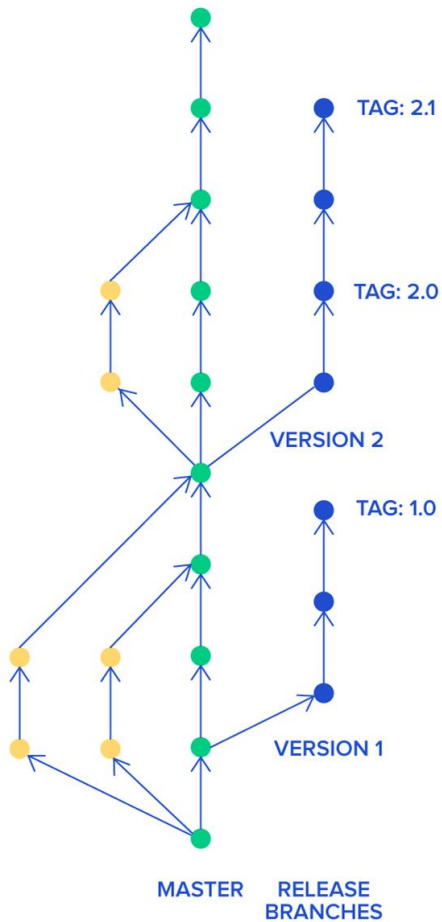
Branching

- Una rama representa **una línea de desarrollo independiente**.
- Las ramas sirven como una abstracción del proceso **de edición/etapa/compromiso**.
- En Git, las ramas son parte del proceso de desarrollo diario: son un **indicador de una instantánea de los cambios**.
- Cuando desea **agregar una nueva función o corregir un error**, sin importar cuán grande o pequeño sea, **se genera una nueva rama** para encapsular estos cambios.

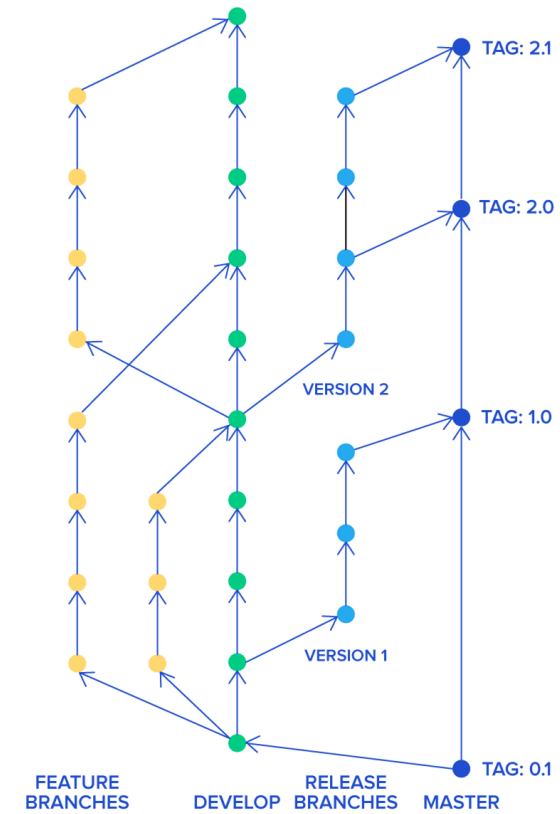


Branching workflows

Ramas de larga duración: workflow de desarrollo basado en troncales

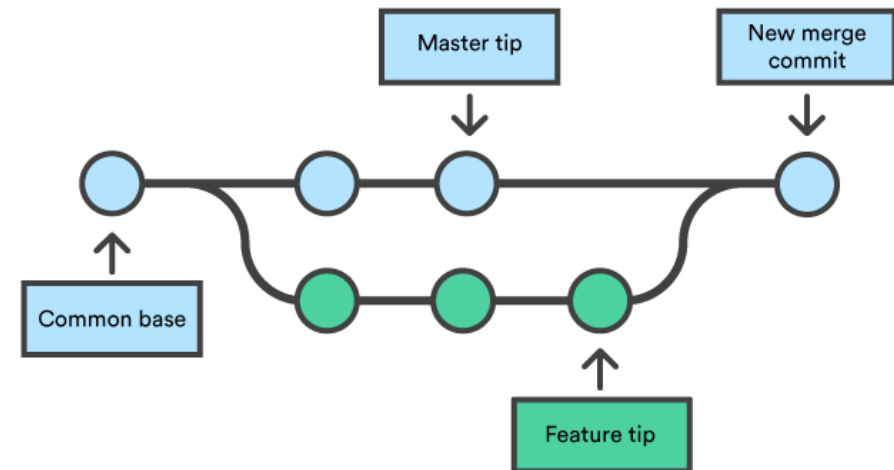


Ramas temáticas: Git Flow



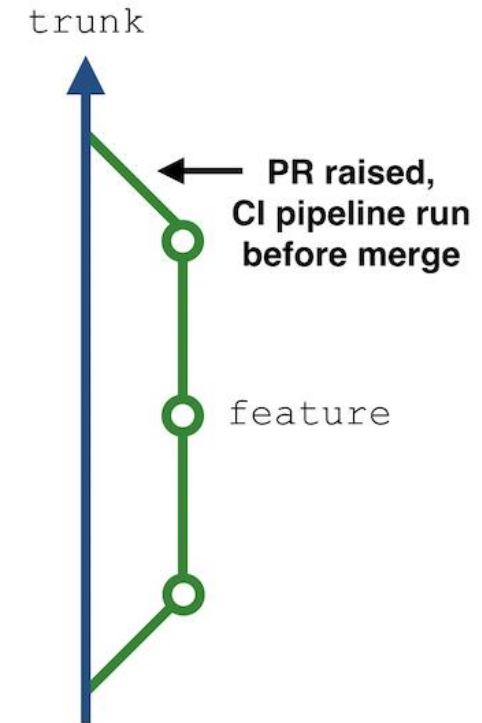
Branch merging

- El merge es la forma en que git **vuelve a unirse a una historia bifurcada**.
- El **comando git merge** permite tomar las líneas de desarrollo separadas creadas por la rama git e integrarlas en una sola rama.
- La fusión permite algunas buenas prácticas de codificación como "**pull request**" y "**programación en pares**".

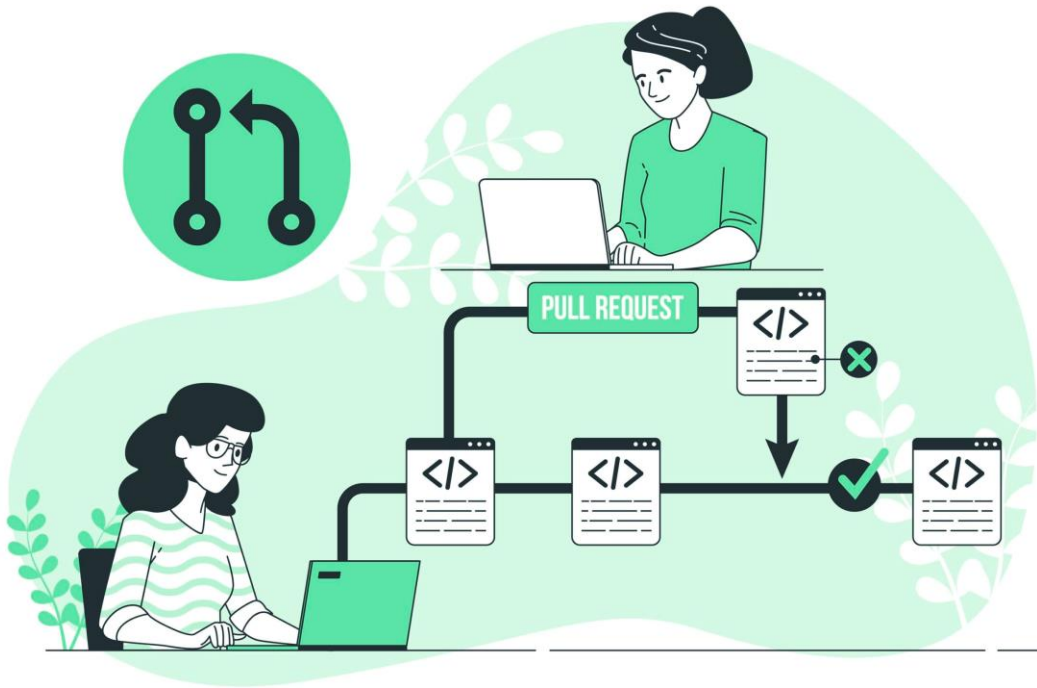


CI/CD branching strategy: Streamline

- Streamline es un enfoque combinado para ramificación, integración continua y (opcionalmente) implementación continua.
- Agilizar **el workflow** consta de:
 1. **Limite las ramas a funciones y troncales** (equivalente a master). la corrección de errores es opcional.
 2. **Utilice solicitudes de extracción** para activar una pipeline de CI que ejecute todas las pruebas funcionales.
 3. **Una combinación de avance rápido** en el tronco marca una nueva "versión" de su aplicación.
 4. La **nueva versión se puede implementar en cualquier lugar**, desde entornos de 'demostración' o 'prueba de penetración' hasta producción.



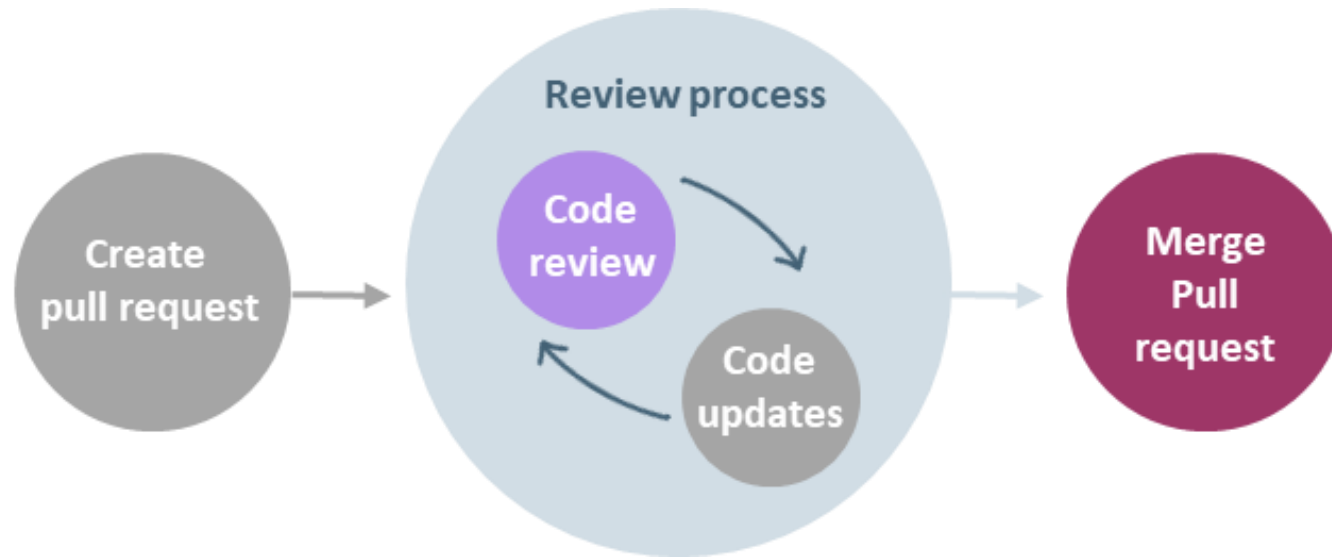
Pull request



Fuente de la imagen:
gitalential.com

1. Un Pull Request es un **método para enviar contribuciones a un proyecto.**
2. En su forma más simple, las solicitudes de extracción son un mecanismo para que un **desarrollador notifique a los miembros del equipo que han completado una función.**
3. Una vez que su rama de funciones está lista, el desarrollador **presenta una pull request a través de sus mecanismos de repositorio.** Esto permite que todos los involucrados sepan que deben revisar el código y fusionarlo en la rama maestra.
4. **Es un método de workflow** ; no es una característica del sistema de control de versiones. Permite una revisión constante del código que mejora la calidad del código y contribuye al trabajo en equipo.

Proceso de pull request



1. Un desarrollador **crea la función** en una rama dedicada en su repositorio local.
2. El **desarrollador envía la rama** a un repositorio central.
3. El **desarrollador envía una pull request**.
4. El resto del equipo (**revisores**) **revisa el código, lo discute y lo modifica**.
5. El **director del proyecto fusiona la función** en el repositorio oficial y cierra la pull request.

OBJETIVO

Juguemos a hacer pull request

INSTRUCCIONES

1. Emparéjate con un compañero.
2. Comparte un repositorio (con algunos archivos de texto).
3. Uno interpretará al desarrollador y el otro al revisor. Sigue el proceso de la siguiente diapositiva.
4. Luego intercambia el papel y repita el siguiente proceso.



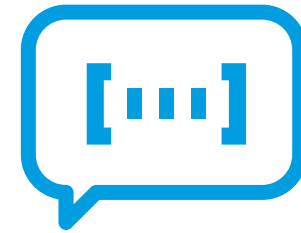
30 min

Proceso de pull request:

- El desarrollador debe:
 1. Generar una rama de características
 2. Haz un cambio... luego agrega, confirma y haz push del cambio.
 3. Genera una pull request en Github.
- El revisor debe:
 1. Aceptar la pull request.
 2. Revisar los cambios.
 3. Hacer algunos comentarios y sugerencias al desarrollador (a través de Github y cara a cara).
- El desarrollador debe:
 1. Leer/escuchar los comentarios y sugerencias del revisor.
 2. Ajustar el código y presiónelo nuevamente.
- El revisor debe:
 1. Aceptar los cambios y fusiona la rama.

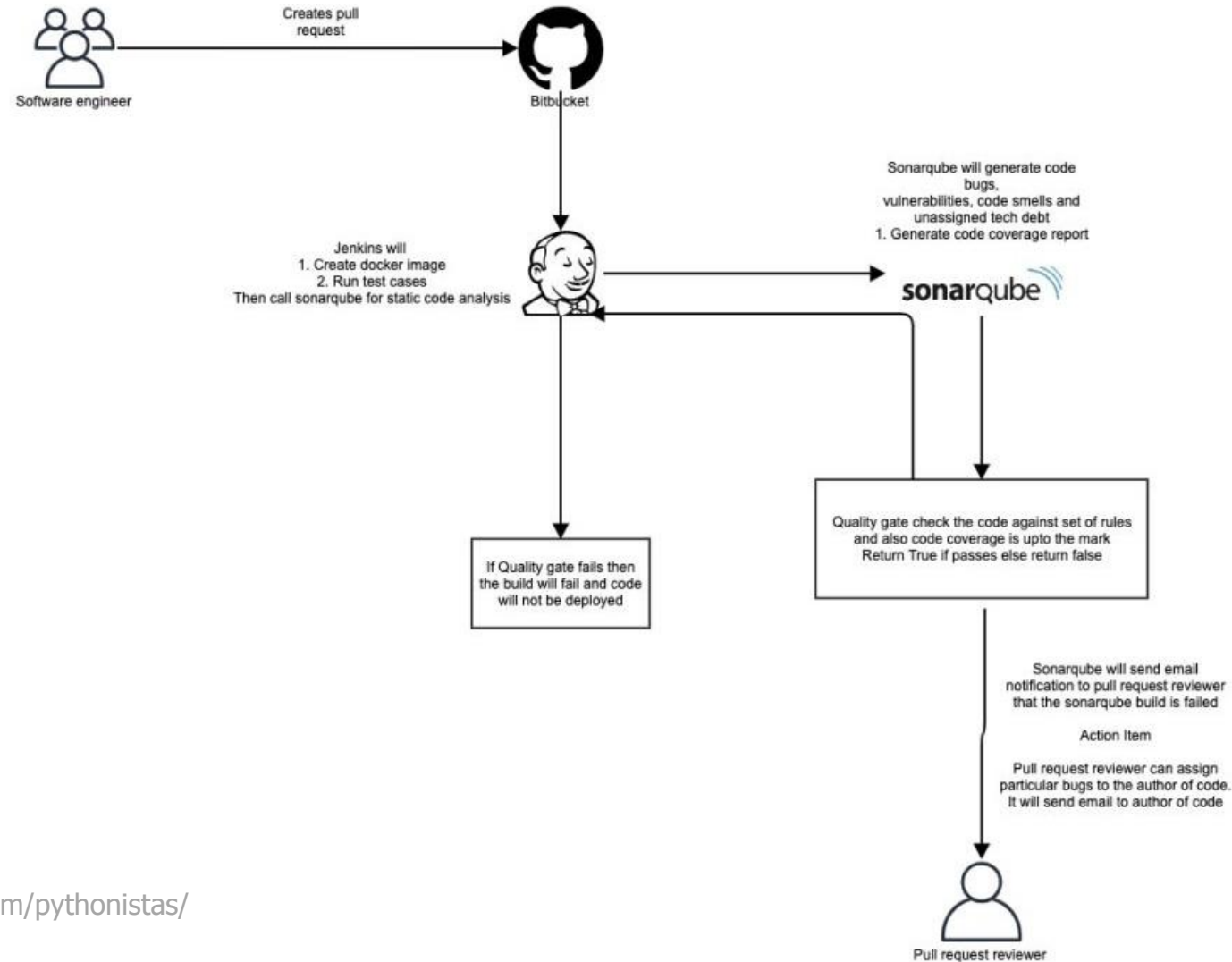


07



Integrar SonarQube con Jenkins

Sonarqube funciona muy bien con las pipelines de Jenkins



la imagen: medium.com/pythonistas/

Análisis en ejecución con Jenkins

- Integra SonarQube en un paso de tu pipeline.
- Agrega **run_sonarqube_script.sh** o **run_sonarqube_script.ps1** en el directorio raíz del repositorio con el comando del escáner:

```
sonar-scanner - Dsonar.projectKey =< clave_proyecto > - Dsonar.sources =. -  
Dsonar.host.url= http://localhost:9000 - Dsonar.login =<token>
```

- Llama al script en un paso de pipeline.
- Guarda la pipeline y ejecuta la tarea.

OBJETIVO

Integrando SonarQube en Jenkins

INSTRUCCIONES

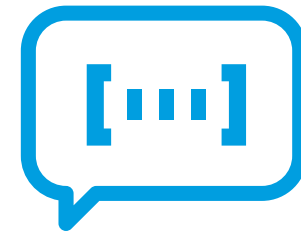
1. Integra el análisis de SonarQube como un paso del proceso de Jenkins.



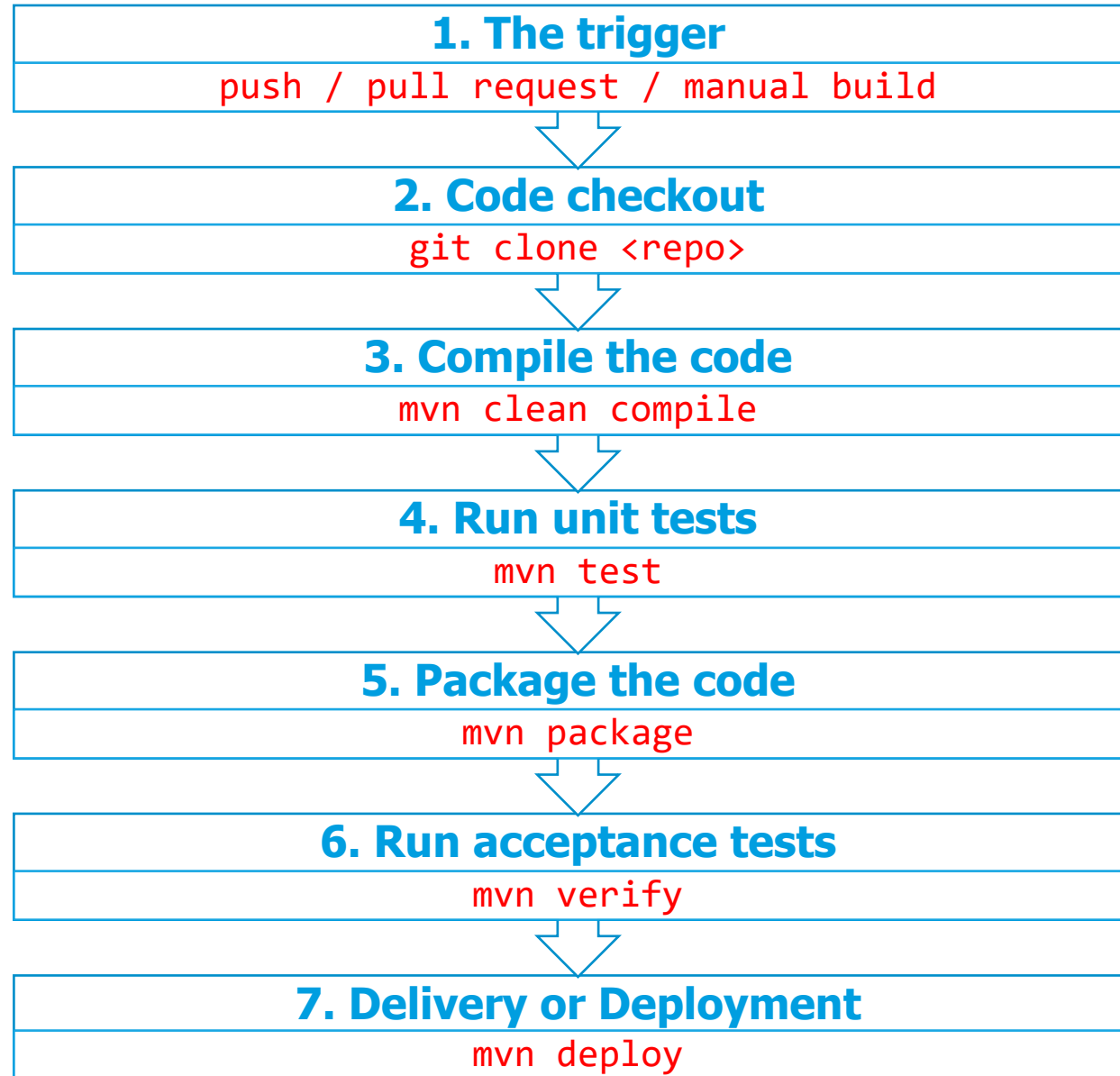
30 min

A1

Guía



7 etapas esenciales de un proceso de CI/CD



Jenkinsfile e estructura

La estructura de nuestra pipeline debe corresponder a las etapas significativas de nuestro proyecto.

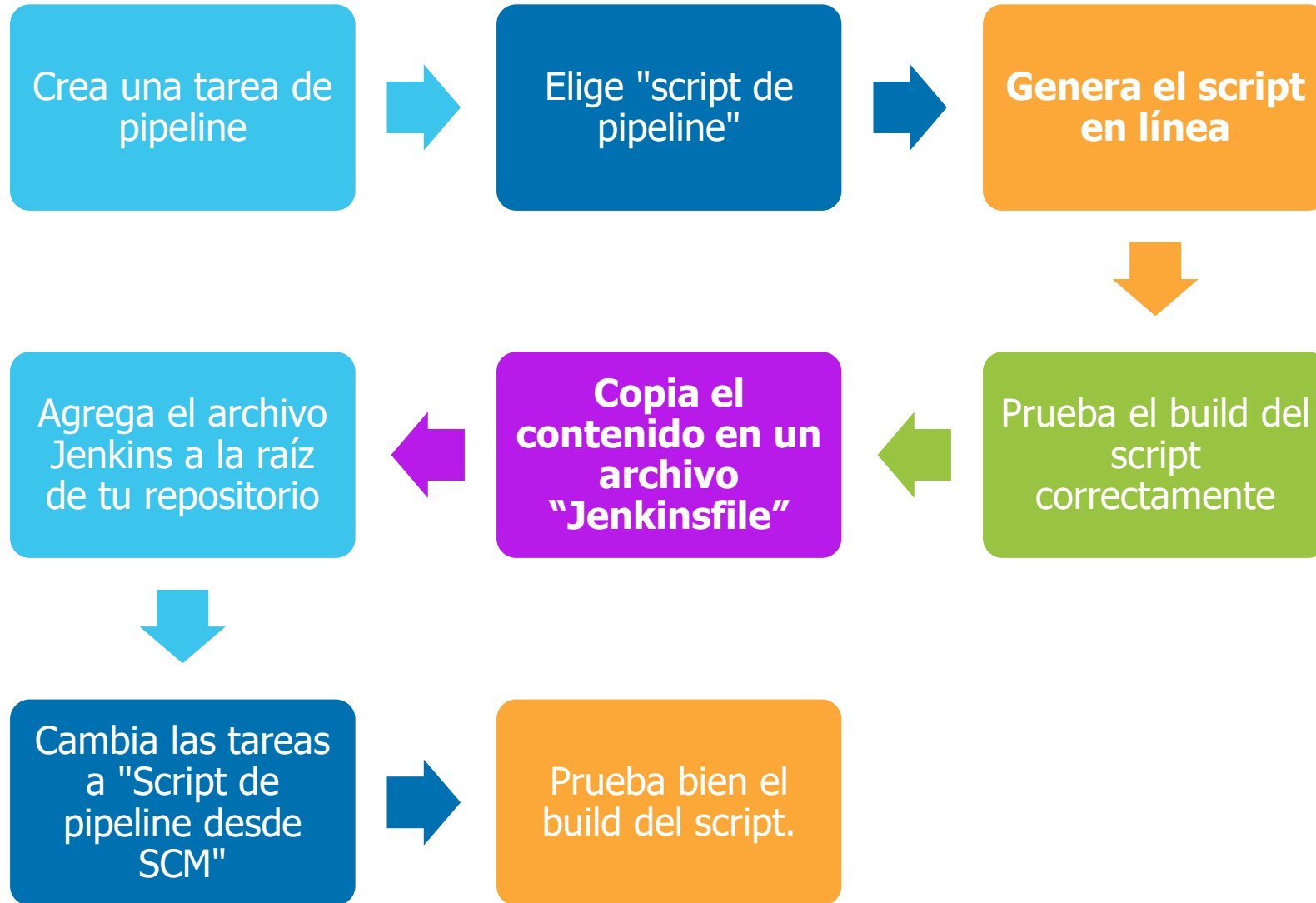
Pipeline:

- Checkout: this is needed to clone the code
- Compile
- Test
- ...

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                echo 'Checkout ...'
            }
        }
        stage('Compile') {
            steps {
                echo 'Building...'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
            }
        }
        stage('Package') {
            steps {
                echo 'Packaging...'
            }
        }
        stage('Acceptance test') {
            steps {
                echo 'Acceptance...'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

¿Cómo crear un Jenkinsfile?



Comandos comunes

Checkout

```
git branch: '<branch>',url: '<repo>'
```

execute a shell or bat command

```
bat '<command>'  
sh '<command>'
```

change dir and execute steps

```
dir ('standalone'){  
  bat 'dir'  
  bat 'mvn clean test'  
}
```

environment variables

```
environment {  
  DB_ENGINE = 'sqlite'  
}  
...  
steps {  
  bat 'set | grep JAVA_HOME'  
  echo "Database engine is ${DB_ENGINE}"  
}
```


Comandos comunes

post action

```
post {
  always {
    echo 'One way or another, I have finished'
    deleteDir() /* clean up our workspace */
  }
  success {
    echo 'I succeeded!'
  }
  failure {
    echo 'I failed :('
  }
}
```

execution parameters

```
parameters {
  choice(
    choices: ['greeting', 'silence'],
    description: "",
    name: 'REQUESTED_ACTION')
}
```

Comandos comunes

conditionals

```
stage ('Speak') {  
  when {  
    // Only say hello if a "greeting" is requested  
    expression { params.REQUESTED_ACTION == 'greeting' }  
  }  
  steps {  
    echo "Hello, bitwiseman!"  
  }  
}
```



Next steps



¡Nos gustaría saber tu opinión!

Por favor, cuéntanos qué piensas sobre el contenido.
Desde Netmind queremos darte las gracias, agradecemos
el tiempo y esfuerzo que dedicas a responder todo lo que es
importante para mejorar nuestros planes de formación y que
siempre estés satisfecho de habernos elegido.
calidad@netmind.es

Thanks!

Follow us:

