Logotipo, Esquemático

Descripción generada automáticamente



**Manual de usuario**

IC /DC JENKINS

TCC – Remesas Descumplidas

**CONTROL DE VERSIONES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Elaborado por:** | Juan David Martínez | **No. de Versión:** | 1.0.0 |
| **Revisado por:** |  | **Fecha de revisión:** |  |
| **Aprobado por:** |  | **Fecha de Aprobación:** |  |

**Historia de Modificaciones**

| **No. de Versión** | **Fecha de Versión** | **Autor** | **Revisado por** | **Aprobado por** | **Descripción** |
| --- | --- | --- | --- | --- | --- |
| 1.0.0 | 26/01/2024 | Juan David Martínez |  |  | Documento Original |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Lista de distribución**

| **Para** | **Acción\*** | **Empresa** | **Firma/Medio de Entrega** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

*\* Tipos de acción: Aprobar, Revisar, Informar, Archivar, Complementar, Asistir a junta, Otras (por favor especificar)*

Contenido

[INTRODUCCION 4](#_Toc150935687)

[PRERREQUISITOS 4](#_Toc150935688)

[ARQUITECTURA: 5](#_Toc150935689)

[Versión Semántica: 6](#_Toc150935690)

[**Prefijo de etiqueta** 7](#_Toc150935691)

[PROCESO DE PUBLICACIÓN: 9](#_Toc150935692)

[Estructura del flujo Pipeline: 10](#_Toc150935693)

[Certificación del Proyecto. 11](#_Toc150935694)

[Fase de Rollback. 13](#_Toc150935695)

# INTRODUCCION

El presente documento muestra los pasos a seguir para trabajar en JENKINS como orquestador en la solución de IC/DC, para las aplicaciones de los proyectos de TCC desplegados en Weblogic.

El proyecto contiene solución para 3 Ambientes (dllo, QA y PDN).

# PRERREQUISITOS

* Disponer de una cuenta GITLAB para acceder a los repositorios de Remesas Descumplidas.
* Jenkins como herramienta de orquestación.
* Plugins necesarios, para la ejecución de los Pipelines.
* Contar con la salida de correo para el envío de notificaciones.
* Acceso a la cuenta de GCP donde este alojado el servicio de Jenkins.

# ARQUITECTURA:

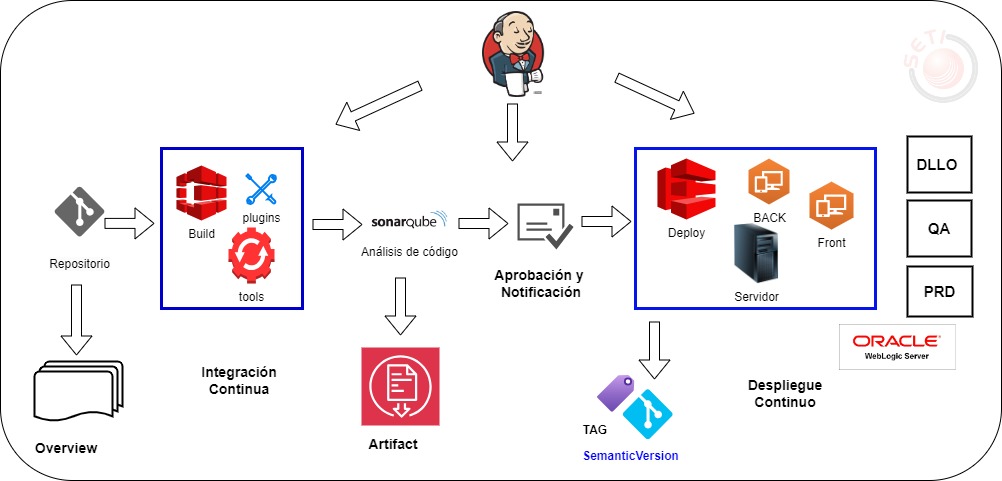


Figura 1. Proceso de publicación de la App Java.

En el proceso de IC/DC se implementó la solución de compilación de la Aplicación con las siguientes fases:

1. Repositorio de código. Se implemento la solución desde la herramienta de GitLab
2. Cuando se aplica un cambio en el repositorio del tipo pull-request este genera una acción en el orquestador Jenkins para el análisis del proceso del despliegue.
3. Cuando el orquestador valida la rama origen activa el pipeline definido por la rama origen. Es decir que cada rama del repositorio tiene un pipeline dedicado para su ejecución.
4. El pipeline realiza la Fase de Análisis de código estático con la herramienta SonarQube. Para esta fase el pipeline publica el resultado.
5. En la Fase de Integracion Continua el pipeline versiona el proyecto bajo el modelo expuesto en el item “Versión Semántica”. Este modelo garantiza que en cada proceso de ejecución se genere una nueva versión.
6. Para los pipelines de QA y PDN, existe un modelo de Notificaciones y Aprobaciones. En esta fase el Pipeline envía un mail de notificación solicitando una acción sobre una actividad ya sea de tipo Gitflow o Build.
7. En la Fase de generación de Artefactos el Pipeline genera los archivos requeridos por el Nodo de Weblogic para su despliegue.
8. La fase de Build entrega a los diferentes ambientes del servidor de Aplicaciones weblogic los artefactos compilados, versionados y Analizados. En esta fase se requiere de un plugin de weblogic ya cargado en los repositorios para su gestión.
9. En la fase de Notificación, se envía el resumen vía mail y grupos de teams la descripción de los procesos generados y adicional se crea un Tag en GIT del proyecto desplegado.

Todos estos pipelines están almacenados en GitLab en la ruta

<https://gitlab.com/tcc-devops/deployments.git>

Para cada proyecto se tiene un folder definido según la tecnología, para weblogic se tiene:

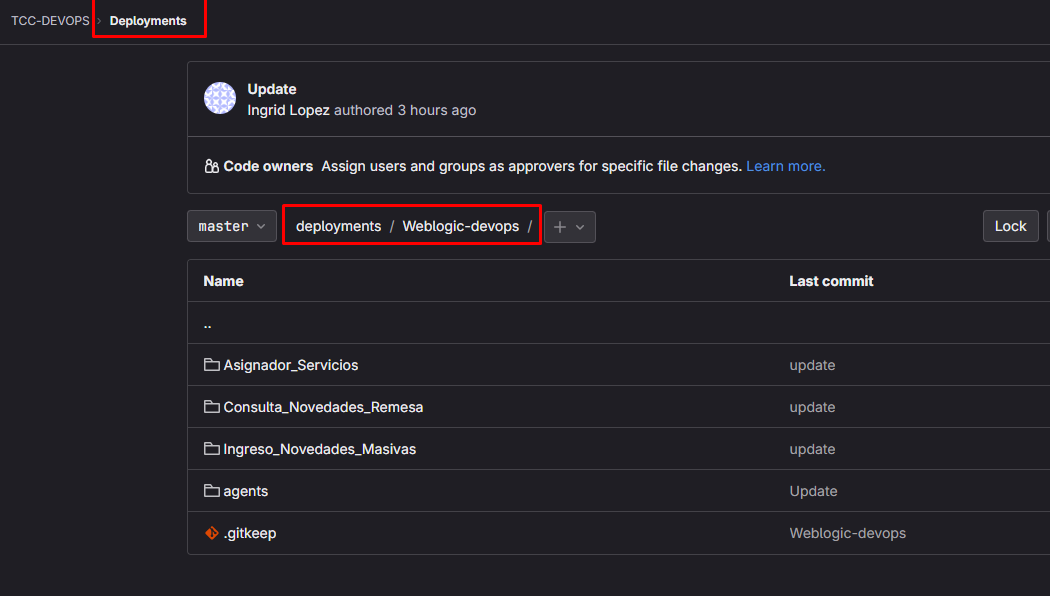


Figura 2. Repositorio de Pipeline.

# Versión Semántica:

Para la implementación de la versión Semántica se implemento el modelo de gitflow. GitFlow se define como un **sistema de***branching***o ramificación o modelo de manejo de ramas en Git**, en el que se usan las ramas principales y la feature. De modo que la rama featurela crean los desarrolladores para fusionarla con la rama principal, únicamente cuando cumpla con sus labores.

Deben existir dos ramas principales para que el flujo de trabajo funcione correctamente:

* Master
* qa
* develop

GitFlow creará por defecto los siguientes prefijos para las ramas auxiliares, los cuales ayudan a identificar y tener control en el repositorio:

* Breaking/
* feature/
* fix/

### **Prefijo de etiqueta**

Una expresión regular que se usa para recortar las etiquetas de Git antes del procesamiento (p. ej., v1.0.0). El valor predeterminado es [vV], aunque esto es solo para fines ilustrativos, ya que hacemos una coincidencia IgnoreCase y podría ser v.

Para los commits en cada cambio se requiere dejar clara la siguiente nemotecnia:

* brk: La expresión regular con la que hacer coincidir los mensajes de confirmación para realizar un incremento de versión principal.
* feat: La expresión regular con la que hacer coincidir los mensajes de confirmación para realizar un incremento de versión menor.
* fix: La expresión regular con la que hacer coincidir los mensajes de confirmación para realizar un incremento de la versión del parche.

Imagen que contiene Patrón de fondo

Descripción generada automáticamente

Figura 3. Flujo del proceso de gitflow.

En la Figura 3 se evidencia el proceso de entrega de los desarrollos en las 3 ramas y su versión semántica.

Para el proceso de desarrollo se requieren validar los siguientes datos en el mensaje del commit:

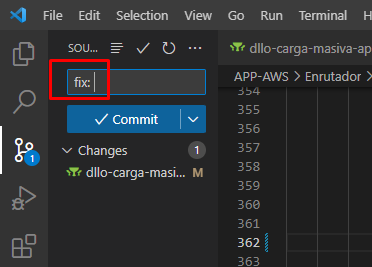


Figura 4. Estructura SemanticVersion.

Este mensaje es obligatorio para que el desarrollo pueda asignar la versión correcta.

En la ruta de GITLAB se genera un archivo version.json el cual lleva la versión definida por los flujos.

Texto

Descripción generada automáticamente

Figura 5. Estructura SemanticVersion JSON.

# PROCESO DE PUBLICACIÓN:

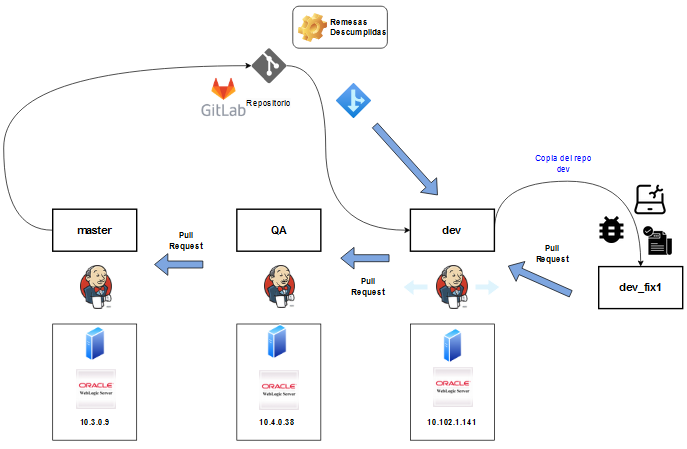


Figura 6. Proceso de publicación.

1. Para realizar un cambio se requiere generar una copia de la rama **dev** y trabajar sobre esta nueva rama (Ejemplo dev\_fix1). Cuando los cambios estén listos se debe realizar un pull\_request a la rama **dev** y buscando eliminar la rama fuente (Solo eliminar las ramas copia).
2. Cuando se realiza el pull\_request en la rama dev esta es aprobada por el mismo desarrollador (No tiene flujo de aprobación).
3. Si se realiza un pull\_request a las ramas QA o master estas tienen un proceso de aprobación. Este proceso dispara el Master pipeline para realizar la validación de las versiones y el tipo de rama.
4. Al realizar el merge de las ramas (dev, qa o master) el Master pipeline es activado y este genera la compilación y despliegue en los diferentes ambientes.
5. Cada merge que es aprobado o ejecutado genera un tag de la versión desplegada.

# Estructura del flujo Pipeline:

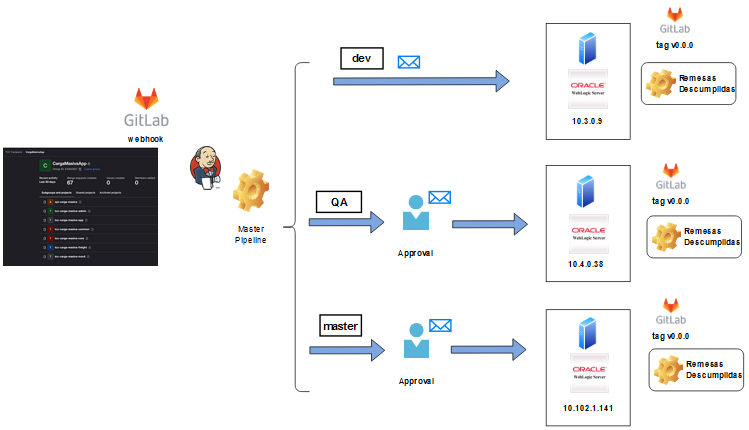


Figura 7. Estructura Pipeline Jenkins.

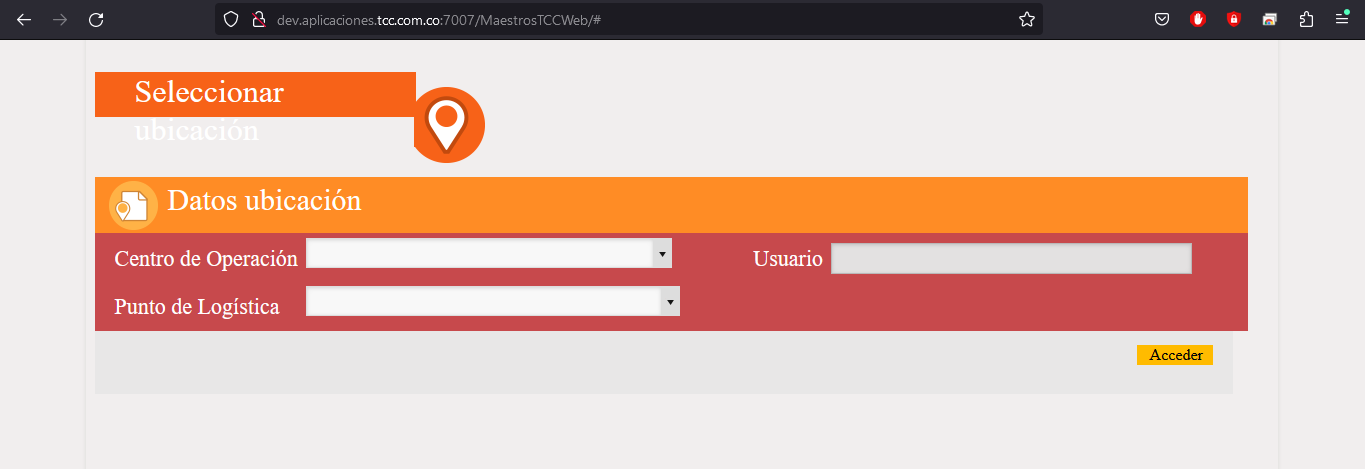
1. En GitLab se tiene configurado un webhook que se dispara en el momento que se recibe una actividad de estado pull\_request.
2. El máster pipeline, analiza la rama de origen y el tipo de cambio.
3. Si el cambio es tipo Pull, este valida la versión y el tipo de commit, con esto envía la notificación al usuario si es merge o Pull\_request.
4. Cuando es Pull\_request, para las ramas QA y PDN se genera un correo de notificación solicitando la aprobación del pull y cambiar a estado merge.
5. Cuando se realiza el merge, se envía el correo al usuario con la notificación del cambio.
6. Cada versión desplegada genera un tag con la estructura antes descrita.

**NOTA: todos los pipelines de Producción y QA contienen una fase de aprobación que esta definida por el usuario dueño del proceso.**

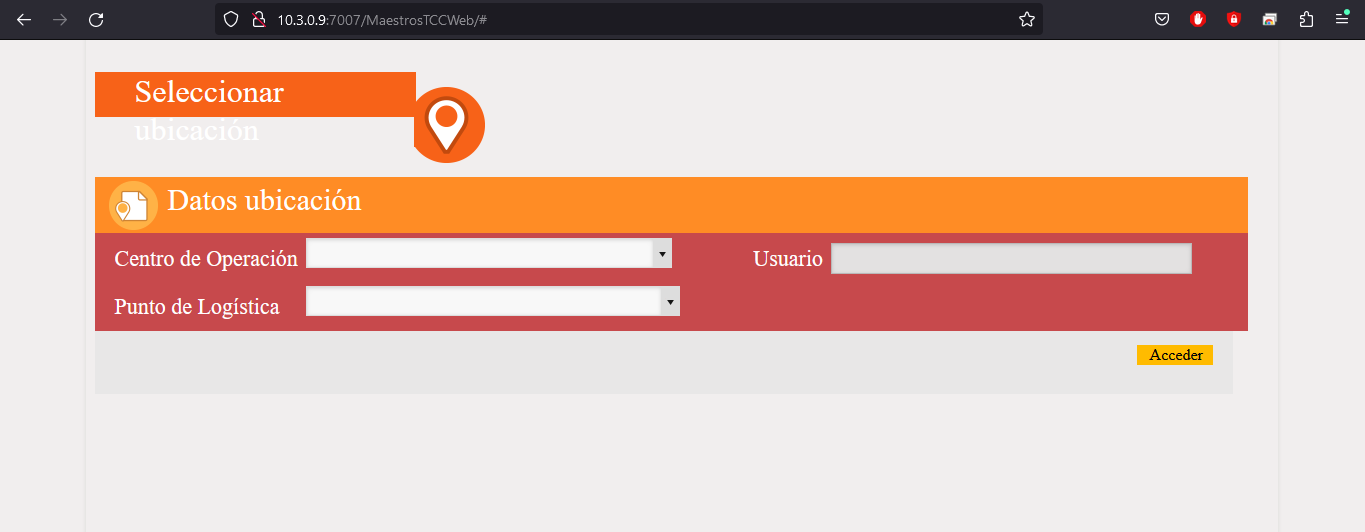
# Certificación del Proyecto.

El siguiente Capitulo Documenta las pruebas realizadas para la Aplicación a nivel del flujo Automático:

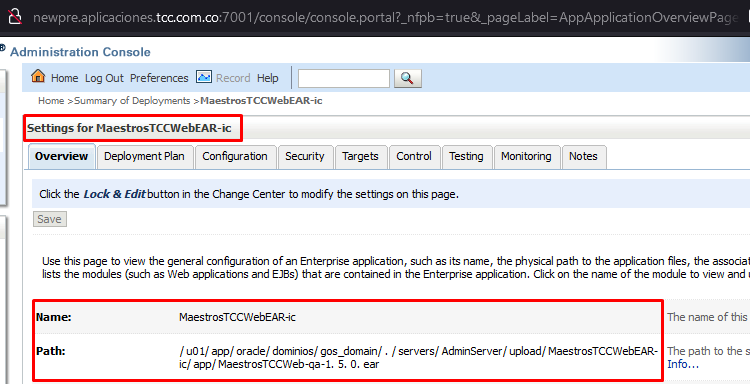
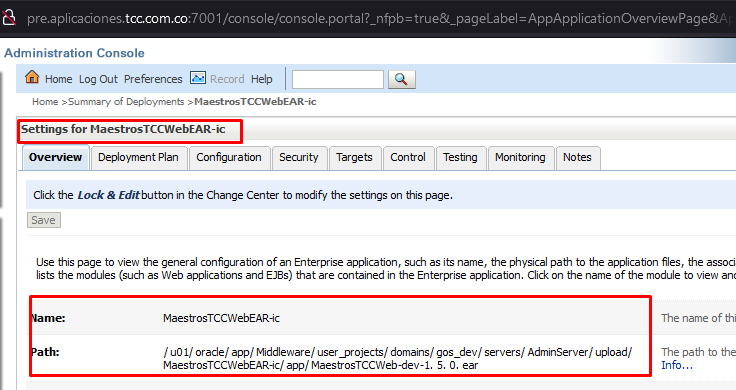
dllo:



QA:

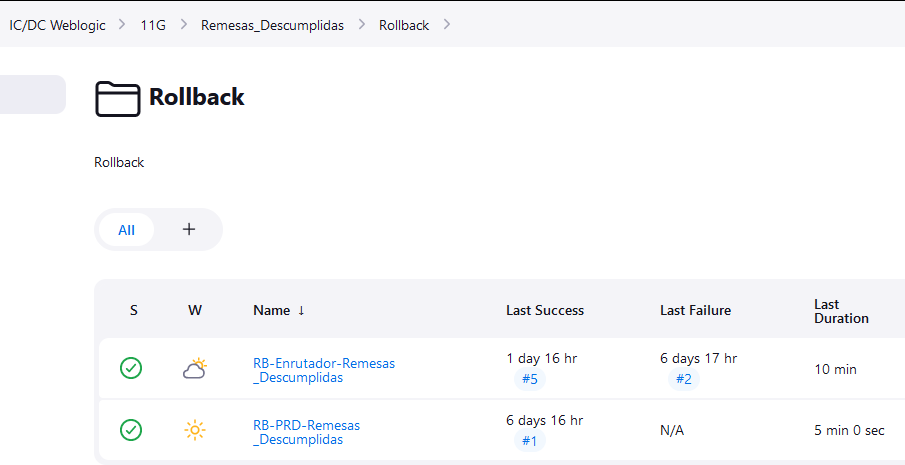


Validación Weblogic:

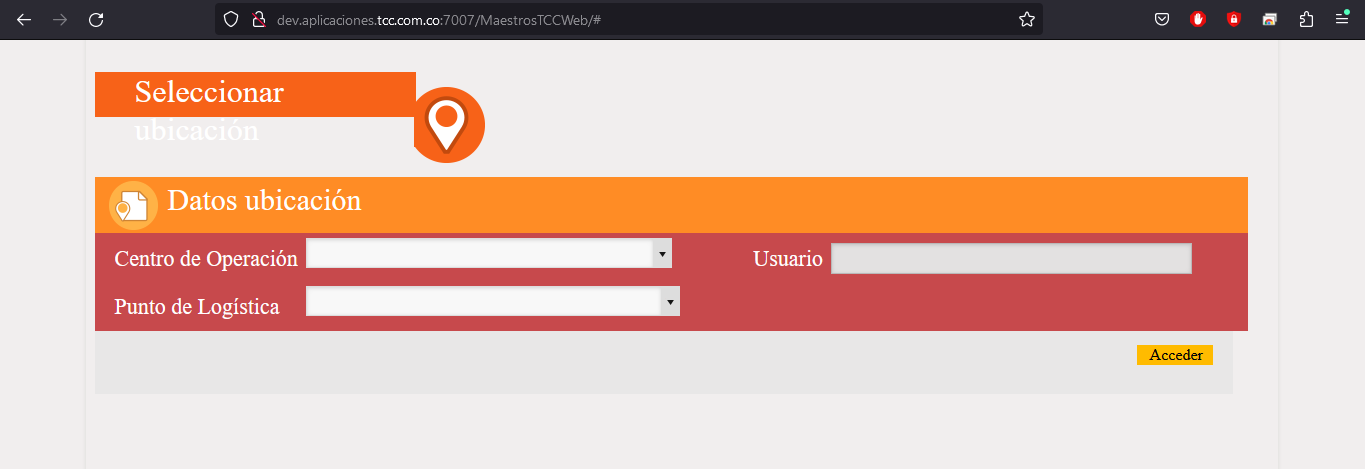


# Fase de Rollback.

Rollback Dllo y QA:



DLLO:



QA:

