Industria y Comercio
SUPERINTENDENCIA

LINEAMIENTOS DE IMPLEMENTACIÓN DEVOPS

LINEAMIENTOS TÉCNICOS PARA LA IMPLEMENTACIÓN Y OPERACIÓN DE LA METODOLOGÍA DEVOPS

SUPERINTENDENCIA DE INDUSTRIA Y COMERCIO

CONTROL DE CAMBIOS

FECHA	AUTOR	VERSIÓN	DESCRIPCIÓN
16/03/2020	Victor Rodriguez	1.0	Versión inicial del documento
07/04/2020	Victor Rodriguez	1.1	Se ajusta sección de pruebas unitarias
28/05/2020	Victor Rodriguez	1.2	Se adiciona Git desde Eclipse
13/07/2020	Victor Rodriguez	1.3	Se adiciona lineamiento de pipeline de integración continua.
11/08/2020	Victor Rodriguez	1.4	Se adicionan lineamientos de despliegue continuo
30/11/2020	Victor Rodriguez	1.5	Se adicionan comentarios de aplicabilidad por proyecto y tecnología.

Contenido

INTRODUCCIÓN

La adopción de nuevas metodologías de trabajo requiere la definición de estándares y formas de aplicarla, pues de esta forma se garantiza que todos los equipos ejecuten las tareas de la misma forma, este documento presenta las consideraciones técnicas, buenas prácticas y lineamientos definidos para la implementación de la metodología DevOps en la Superintendencia de Industria y Comercio.

Este es un documento en su mayoría técnico, el cual debe ser usado en conjunto con la Metodología de Desarrollo de Software para la construcción de los sistemas de información de la SIC.

1. GENERAL

Los lineamientos generarles para la implementación de DevOps consisten en el uso estandarizado de las herramientas de desarrollo para todos los proyectos nuevos, para los antiguos se debe realizar un análisis y un plan de trabajo para establecer el estado objetivo de cada uno y los paso para lograr dicho estado.

Las fases del ciclo de trabajo de DevOps están alineadas a la metodología de desarrollo de software y a las etapas del plan de acción, por lo tanto, no va en contravía de ningún otro lineamiento de desarrollo.

Asi mismo es de recordad que el uso de DevOps es un lineamiento estratégico de la Oficina de Tecnología de la Información de la SIC, y es necesario que todos los proyectos de desarrollo lo adopten.

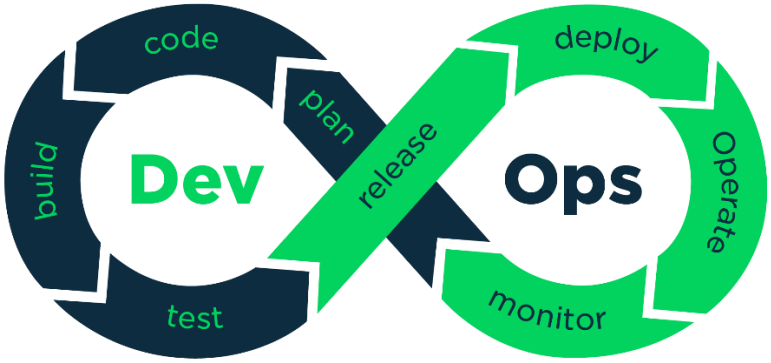


Imagen 1. Ciclo de trabajo de DevOps

ID	LINEAMIENTO	RESPONSABLE	DESCRIPCIÓN
LD.01	Cultural	Equipo de Arquitectura Lider OTI Coordinadores	Debe existir una estrecha colaboración entre los desarrolladores y el personal de operaciones de TI
LD.02	Cultural	Equipo de Arquitectura Lider OTI Coordinadores	Intercambio frecuente de conocimiento y uso de mejores prácticas de los desarrolladores y el personal de operaciones de TI
LD.03	Cultural	Equipo de Arquitectura Lider OTI Coordinadores	Establecer objetivos, incentivos y valores compartidos entre los desarrolladores y personal de operaciones de TI
LD.04	Cultural	Lider OTI Coordinadores	Debe haber una integración total entre el equipo de desarrollo y los grupos

		Equipo de desarrollo Área funcional	funcionales para que los requerimientos se desarrollen de la mejor forma
LD.05	Cultural	Equipo de Arquitectura Lider OTI Coordinadores	Empezar el proceso de transición poco a poco, es decir, que en lugar de tratar de automatizar una compilación complicada existente que aporta mucho al negocio y es difícil de administrar por todo el equipo, es mejor comenzar con una compilación simple y trivial para establecer un nuevo proceso limpio de Integración y Despliegue Continuo que todo el equipo pueda entender. Estos cambios deben ser Incrementales, Iterativos y continuos
LD.06	Cultural	Equipo de Arquitectura Lider OTI Coordinadores	Todo cambio en producción debe ser aprobado por el equipo (desarrollo + operación + funcional), y notificar al comité de cambios
LD.07	Automatización	Equipo de Arquitectura Coordinadores Equipo de desarrollo	Se favorecerá la automatización de procesos (Construcción, Pruebas, Despliegue) que garanticen la entrega de valor,
LD.08	Agilismo	Equipo de desarrollo	Realizar entrega de valor rápida
LD.09	Automatización	Lider OTI Coordinadores	Fomentar la autogestión en los equipos de trabajo
LD.10	Automatización	Equipo de Arquitectura Equipo de desarrollo Ingeniero DevOps	Las ayudas graficas son muy útiles a la hora de trabajar con DevOps. Por ejemplo, un gráfico del proceso de puesta en producción del software y como las actividades de los desarrolladores y operadores de TI interactúan entre si.
LD.11	Métricas y Monitoreo	Equipo de Arquitectura Equipo de desarrollo Ingeniero DevOps	Generar métricas dentro de las aplicaciones y realizar monitoreo de esta en tiempo real
LD.12	Colaboración	Líder OTI, Coordinadores	Realizar seminarios o talleres de mejora de la comunicación entre los equipos de Desarrollo y operadores de TI

LD.13	Agilismo	Equipo de Arquitectura Lider OTI Coordinadores Equipo de desarrollo	Se pondrá en práctica la metodología de desarrollo ágil definida por la OTI.
LD.14	Agilismo	Coordinadores Equipo de desarrollo Área funcional	Los equipos de trabajo tendrán ciclos de realimentación muy cortos desde lo funcional y operativo, para poder incluir mejoras rápidamente
LD.15	Colaboración	Lider OTI, Coordinadores	Generar espacios de esparcimiento entre los equipos de desarrollo y operaciones de TI
LD.16	Colaboración	Lider OTI, Coordinadores	Utilizar herramientas de comunicación que hagan fluida la comunicación entre equipos
LD.17	Colaboración	Lider OTI, Coordinadores Equipo de desarrollo	Se establecé un mecanismo de comunicación asíncrona que permita consultar conversaciones y decisiones previas, como MS Teams o Slack
LD.18	Automatización	Equipo de Arquitectura Equipo de desarrollo Ingeniero DevOps	Los ambientes de despliegue deben en lo posible ser aprovisionados de forma automática, para poder hacer pruebas o migraciones de infraestructura.
LD.19	Automatización	Equipo de Arquitectura Coordinadores Equipo de desarrollo Ingeniero DevOps	Se establecen los escenarios de pruebas unitarias, funcionales, de integración y de carga que se deben hacer construir y ejecutar en cada proyecto
LD.20	Automatización	Equipo de Arquitectura Equipo de desarrollo Ingeniero DevOps	Se debe hacer el análisis de código estático como parte del proceso de construcción, con el fin de garantizar la calidad del software.
LD.21	Automatización	Equipo de Arquitectura Equipo de desarrollo Ingeniero DevOps	El no cumplimiento de cualquier Quality Gate, impedirá la construcción y despliegue del código desarrollado, con esto se garantiza la correcta construcción y estabilidad de los ambientes.
LD.22	Arquitectura y diseño	Equipo de Arquitectura	La arquitectura de los sistemas de información estará diseñada con componentes desacoplados, favoreciendo los despliegues

			independientes y migración a microservicios.
LD.23	Arquitectura y diseño	Equipo de Arquitectura	Los mecanismos de trazabilidad de los sistemas se centralizan para poder llevar a cabo un diagnóstico y gestión más ágil.
LD.24	Desarrollo	Coordinadores Equipo de desarrollo	El código fuente de todos los componentes y proyectos debe ser gestionado de forma centralizada en la herramienta Azure DevOps y serán repositorios tipo GIT.
LD.25	Desarrollo	Coordinadores Equipo de desarrollo	Se usa una metodología de gestión de ramas en el repositorio para los desarrollos, todos los dias debería haber por lo menos un push a las ramas feature.
LD.26	Desarrollo	Equipo de Arquitectura Equipo de desarrollo Ingeniero DevOps	Cada push/commit en las ramas Develop o Master debe iniciar el proceso automatizado de CI y CD
LD.27	Desarrollo	Lider OTI Equipo de Arquitectura Coordinadores Equipo de desarrollo Área funcional	Se priorizan esfuerzos de refactorización de los sistemas de información con el fin de evitar la obsolescencia y potencia el uno de nuevas y más eficientes tecnologías.
LD.28	Cultural	Lider OTI Coordinadores Equipo de desarrollo	Se fomenta una cultura de aprendizaje continuo y de innovación para el desarrollo de nuevos sistemas y soluciones.
LD.29	Estandarización	Lider OTI Equipo de Arquitectura Equipo de desarrollo	Se cuenta con la arquitectura de referencia y el stack tecnológico de la SIC, para que los sistemas de información desarrollados y refactorizados se alineen a esta definición y potenciar su mantenibilidad.
LD.30	Estandarización	Lider OTI Equipo de Arquitectura Coordinadores Equipo de desarrollo Ingeniero DevOps	La gestión de DevOps se hará soportada en la herramienta Azure DevOps
LD.31	Automatización	Equipo de Arquitectura Equipo de desarrollo	Los procesos de despliegue se hacen de forma automática, para los

		Ingeniero DevOps	ambientes de desarrollo, pruebas (UAT) y producción
LD.32	Desarrollo	Coordinadores Equipo de desarrollo	Los despliegues en ambientes de prueba (UAT) son aprobados por los equipos funcionales o de desarrollo, previamente a ser desplegados en ambiente productivo.
LD.33	Desarrollo	Comité de cambios	Los despliegues en ambientes de productivos son aprobados y priorizados por el comité de cambios de la OTI.
LD.34	Automatización	Equipo de Arquitectura Equipo de desarrollo Ingeniero DevOps	Se tiene un pipeline de despliegue para la rama master y uno para la rama develop.

Tabla 1. Lista de lineamientos

NOTA: La aplicación de los diferentes componentes del presente lineamiento se hará evaluando la capacidad de los equipos de desarrollo, la tecnología usada y la antigüedad de los mismos, sin embargo para todos los proyectos nuevos (iniciados de cero a partir del 2021) se recomienda aplicarlos en su totalidad; También es necesario recomendar que de acuerdo al mapa de ruta de implementación de los proyectos, a la capacidad y tecnología usada se debe garantizar un cumplimiento mínimo de las etapas.

ETAPA	ACTIVIDAD	OBLIGATORIEDAD	
		PRODUCTOS NUEVOS	PRODUCTOS VIEJO
PLANEACIÓN	DASHBOARD	SI	SI
	BACKLOG	SI	SI
CODIFICACIÓN	REPOSITORIOS	SI	SI
	SONARLINT	RECOMENDADO	RECOMENDADO
CONSTRUCCIÓN	HERRAMIENTA DE CONSTRUCCIÓN	SI	RECOMENDADO
	CI PIPELINES	SI	RECOMENDADO
	QUALITY GATES	SI	RECOMENDADO
PRUEBAS	TEST PLANS	SI	SI
	PRUEBAS UNITARIAS	SI	RECOMENDADO
LIBERACIÓN	ARTEFACTOS	SI	RECOMENDADO
DESPLIEGUE	CD PIPELINES	SI	RECOMENDADO
OPERACIÓN	POR DEFINIR	POR DEFINIR	POR DEFINIR
MONITOREO	POR DEFINIR	POR DEFINIR	POR DEFINIR

Tabla 2. Obligatoriedad por etapas

1.1.Azure DevOps

La herramienta que soporta el ciclo de vida de DevOps es **Azure DevOps**, esta es la evolución de Team Foundation Services, y se usa como servicio en la nube, esta tiene dentro de sus funcionalidades la capacidad de gestión de proyectos, repositorio de código, automatización de flujos de compilación y de despliegue (CI/CD), entre otros.

La herramienta tiene una estructura jerárquica de dos niveles principales, el primero es la Organización, esta se crea para agrupar todos los proyectos de una entidad, en este caso la SIC, el segundo nivel son los proyectos y se crea uno por cada proyecto de desarrollo que se lleva dentro de la entidad.

El acceso a la herramienta se hace a través del siguiente enlace:

https://dev.azure.com/SUPER-INDUSTRIA/

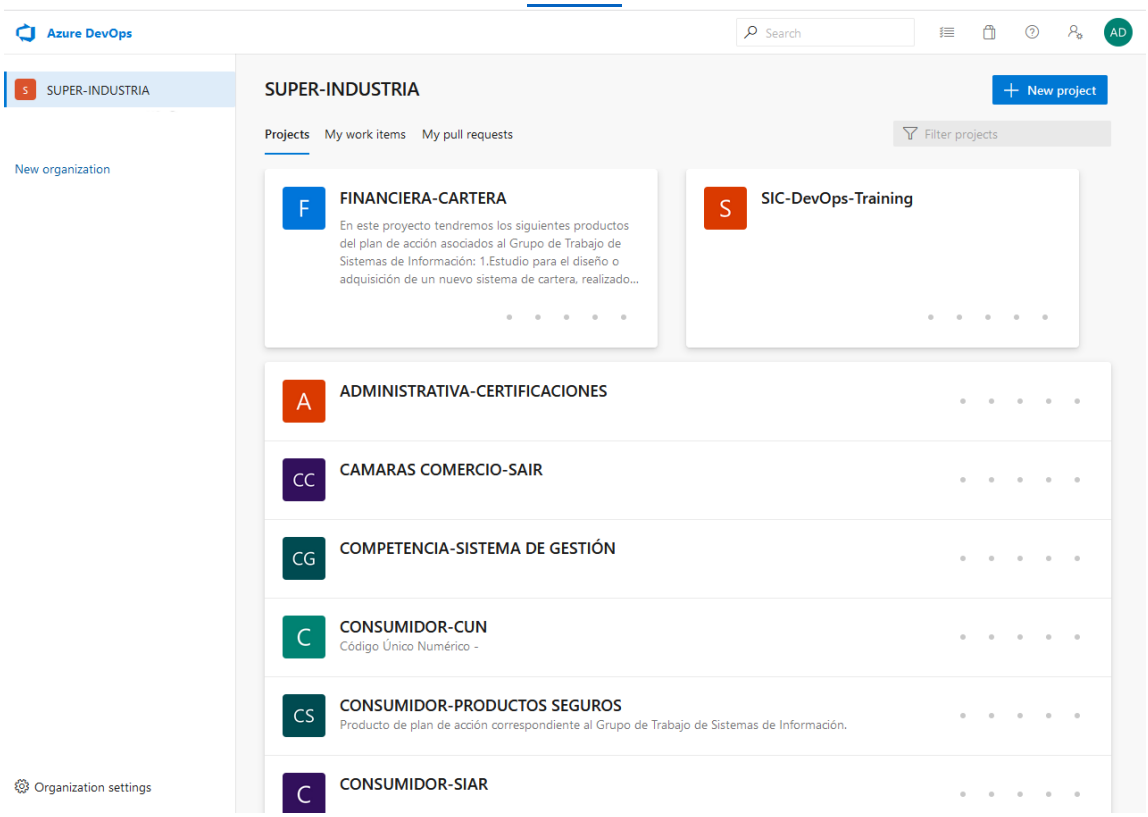


Imagen 2. Vista de la organización

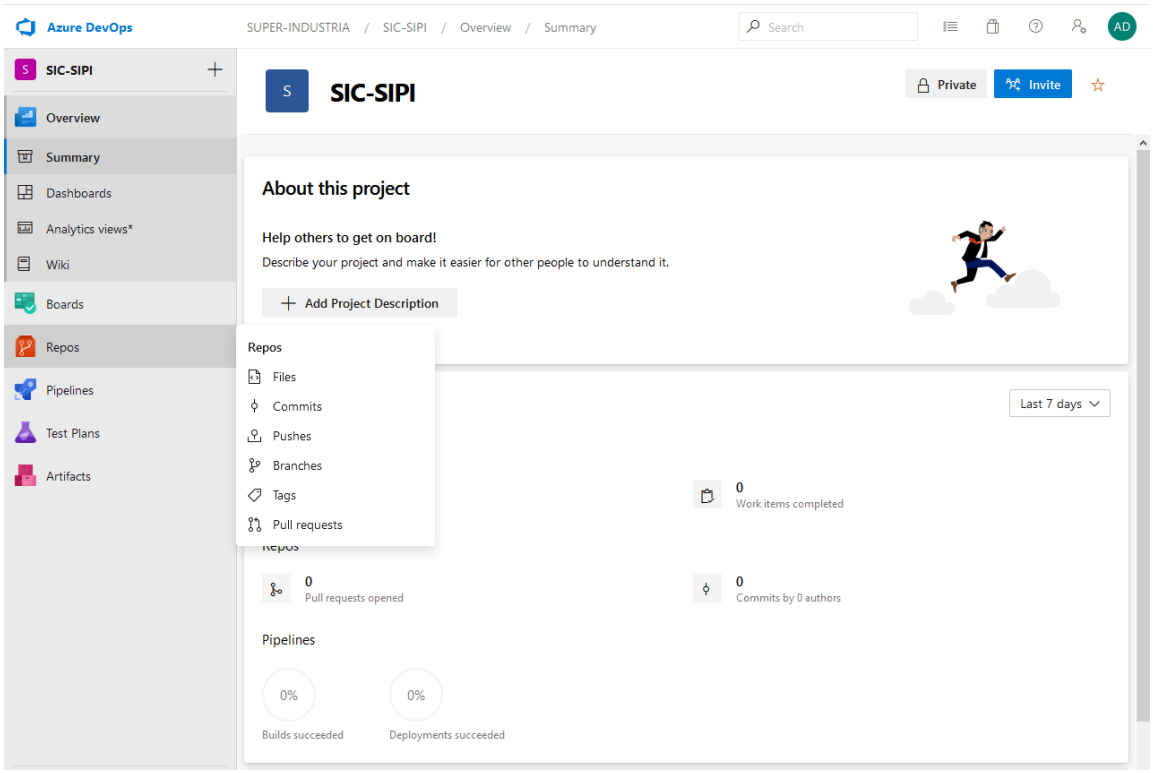


Imagen 3. Vista de proyecto

1.2.Quality Gates

Un Quality Gate es una medición de calidad que debe cumplir el software para su proceso de compilación y despliegue, se establecen para garantizar la calidad de los procesos de desarrollo, se definen dentro de la herramienta Azure DevOps y se miden en los procesos de integración y despliegue continuo.

Dentro de los más comunes se tienen el porcentaje de cubrimiento de pruebas automáticas, deuda técnica por revisión estática de código, vulnerabilidades de seguridad, cumplimiento de plan de pruebas.

Para la implementación de DevOps, los Quality Gates se establecerán desde un comienzo dentro de la herramienta, pero su nivel de exigencia se va a ir incrementando para poder garantizar su cumplimiento.

NOTA: Para los proyectos viejos (iniciados antes del 2021), los quality gates van a ser de aplicación progresiva de acuerdo con la capacidad del proyecto para implementar los ajustes necesarios para cumplirlos.

2. CODIFICACIÓN

El proceso de codificación, si bien es específico para cada proyecto se debe alinear a los estándares definidos para la construcción, los cuales son los siguientes.

2.1.Repositorio

Todos los proyectos de desarrollo de software (incluido el mantenimiento) que soporten los actuales o nuevos sistemas de información deben usar el repositorio GIT provisto en Azure DevOps y se debe hacer uso de una extensión del IDE para el versionamiento con GIT.

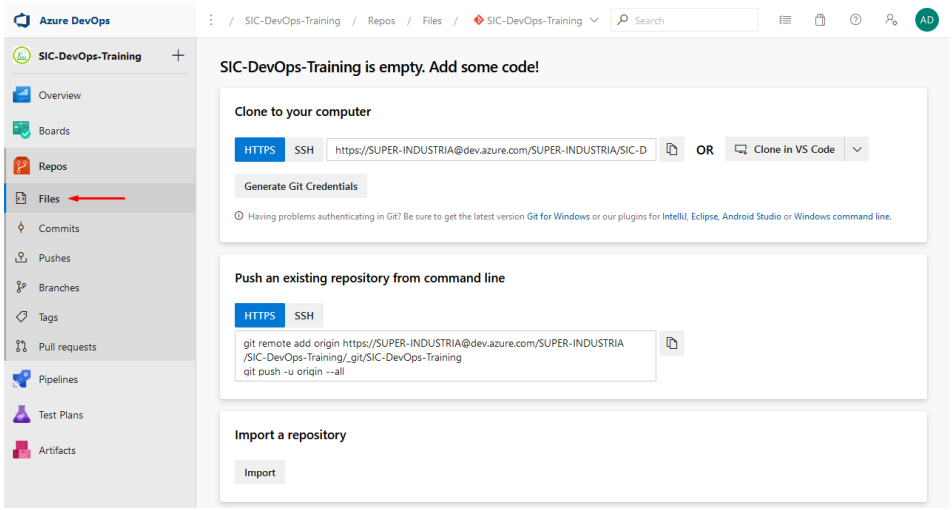


Imagen 4. Acceso a los repositorios del proyecto

Cada proyecto en Azure DevOps cuenta con la opción de creación de repositorios, para la creación de estos se deben seguir los siguientes lineamientos:

2.1.1. Creación de repositorios:

La creación de repositorios se hace desde la opción Repos de cada proyecto, usando la opción de gestión de repositorios que se encuentra en la parte superior de la pantalla, como se muestra a continuación.

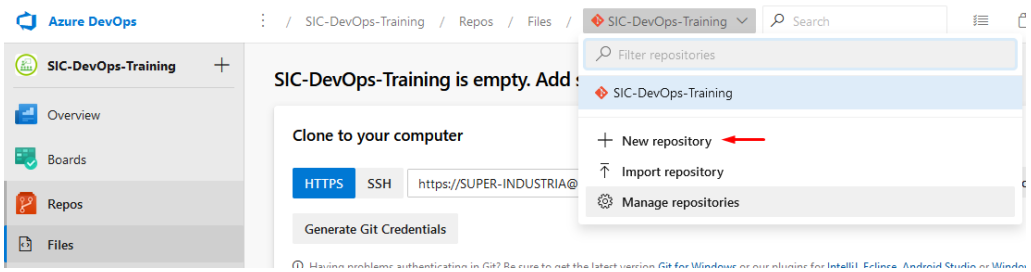


Imagen 5. Creación de repositorio (1)

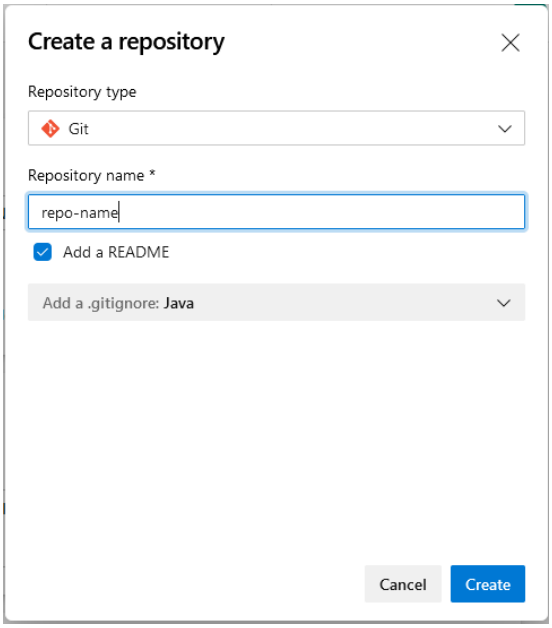


Imagen 6. Creación de repositorio (2)

Para crear el repositorio se deben diligenciar todos los campos asi:

Tipo de repositorio: Para todos los proyectos se usará GIT
Nombre de repositorio: Se debe ajustar al lineamiento definido en el punto 2.1.2
Add a README: Se debe seleccionar para agregar la información que describe el repositorio.
Add a .gitignore: Se debe seleccionar el lenguaje de desarrollo para el archivo que indica que elementos nos se deben versionar

¿Cuándo crear un repositorio?, cuando se necesite versionar el código de cualquier componente que se deba desplegar de forma independiente, un microservicio, un desarrollo front, un api de servicios backend, una librería reutilizable, etc.

2.1.2. Nombramiento de repositorios:

Se debe crear el repositorio de acuerdo con sus características como **alcance** (si es especifico del proyecto, del área o transversal a toda la entidad), **tipo** si es una librería, un proyecto backend, o frontend, si es desarrollo en base de datos, microservicio, servicio de interoperabilidad entre otros (esta lista se refinará con la evolución de la metodología), y la **descripción** la cual debe describir la funcionalidad, se usa el carácter "-" (guion) para separar los componentes del nombre, en la siguiente tabla se presenta un ejemplo de nombramiento y en las secciones posteriores se describen los criterios de nombramiento.

ALCANCE	TIPO	DESCRIPCIÓN	NOMBRE
fin	back	compensaciones	fin-back-compensaciones
adm	datos	depuracion	adm-datos-depuracion
jur	móvil	appconsultas	jur-móvil-appconsultas
sic	web	portal	sic-web-portal
fin	lib	calculadora	fin-lib-calculadora

Tabla 3. Ejemplos de nombramiento

2.1.2.1. Alcance:

El alcance de los repositorios esta dado por su capacidad de ser reutilizados dentro de la entidad, este es un principio de la arquitectura de referencia de la entidad, el código no debe repetirse dentro de los proyectos, sino que si es susceptible de ser reutilizado se deberá crear un repositorio de acuerdo con el alcance de reutilización.

Todos estos componentes deberán ser liberados y mantenidos de acuerdo con el esquema de versionamiento de software definido en la sección 2.6 de este documento.

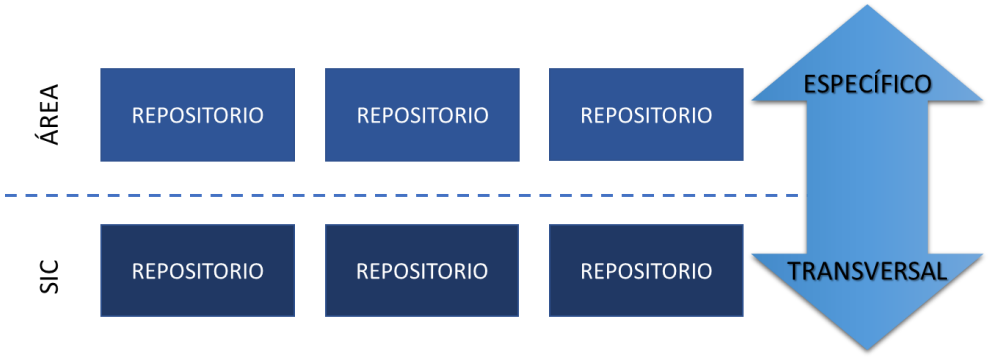


Imagen 7. Descripción de alcance de los repositorios

2.1.2.1.1. Repositorios del área:

Se consideran todos los repositorios de código con funcionalidades transversales para los proyectos del área, estos se pueden reutilizar como librerías de componentes dentro de los proyectos del área, por ejemplo “Componente para la generación de reportes Financieros”, para el nombramiento se usan las tres primeras letras del nombre del área para la cual se desarrolla el proyecto, ejemplo Financiera **fin**, Administrativa **adm**.

2.1.2.1.2. Repositorios de la entidad:

Se consideran todos los repositorios de código con funcionalidades transversales para los proyectos de toda la SIC, el objetivo es poder tener un catálogo de librerías que soluciona problemas técnicos de forma genérica para todos los proyectos, por ejemplo “Componente de autenticación centralizado.”, cuando el repositorio es transversal a toda la entidad se usara la abreviatura **sic**.

2.1.2.2. Tipo:

El tipo de repositorio depende del tipo de la funcionalidad construida en este, y de acuerdo con la capa de la arquitectura de referencia en la cual se despliega el componente, en la imagen 8 se presentan las posibilidades identificadas hasta el momento, estos podrían extenderse al evolucionar la implementación de la metodología DevOps.

TIPO	DESCRIPCIÓN	ABREVIATURA
Front Web	Componente Front Web que se despliegue de forma independiente.	web
Front Móvil	Aplicación móvil para el acceso a los servicios	móvil
Front Windows	Aplicación Windows que corre en los PC y consume servicios en una capa de integración	windows
API REST	Desarrollo para la capa de gestión de APIs (API Management)	restapi
Servicio SOAP	Servicio SOAP separado de un backend funcional	soap
Bus de Integración	Desarrollo de servicio proxy para ser desplegado en el bus de integración	bus
Microservicio	Componente independiente de código que expone su funcionalidad a través de una interfaz tipo REST.	ms
Backend Funcional	Componente de código funcional que se ejecuta del lado del servicio, puede contener la capa de UI si es un proyecto JAVA Web o PHP con las vistas integradas en el proyecto	back
PL/SQL	Desarrollo de procedimientos almacenados que se pueden ejecutar directamente en la base de datos.	plsql
ETL / Procesamiento de datos	Código o utilidades que permiten hacer procesamiento de datos incluido la extracción, transformación y carga de datos.	datos
Librería	Todos los elementos de código reutilizables entre proyectos y capas de la arquitectura de la SIC.	lib

Tabla 4. Tipos de proyectos de código

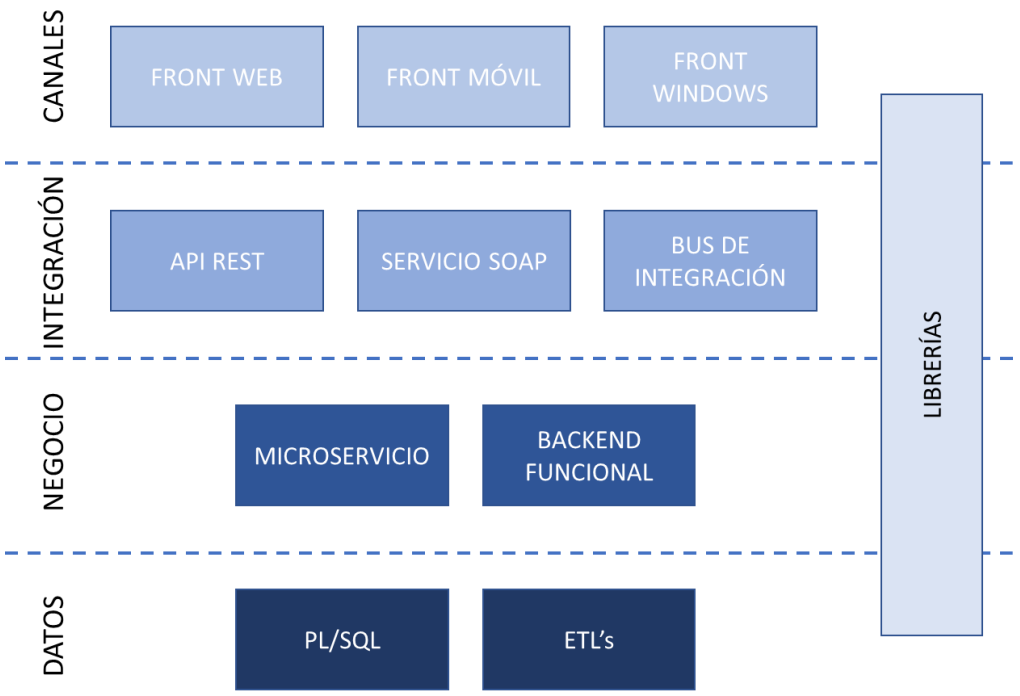


Imagen 8. Tipos de proyectos vs Arquitectura

2.1.2.3. Descripción:

Este atributo es autónomo de cada equipo de trabajo, sin embargo, no debe ser muy largo (máximo de 25 caracteres), y debe describir el contenido del código o de la funcionalidad desarrollada, para separar palabras se usa el carácter guion "-".

2.2.GitFlow

Para el manejo del esquema de ramas del repositorio se define el uso del estándar GIT Flow, el cual es ampliamente usado en proyectos OpenSource y propietarios alrededor del mundo, para este modelo se usan dos ramas principales que siempre deben existir, y tres tipos de ramas auxiliares que se usan para necesidades específicas del proceso, estas son:

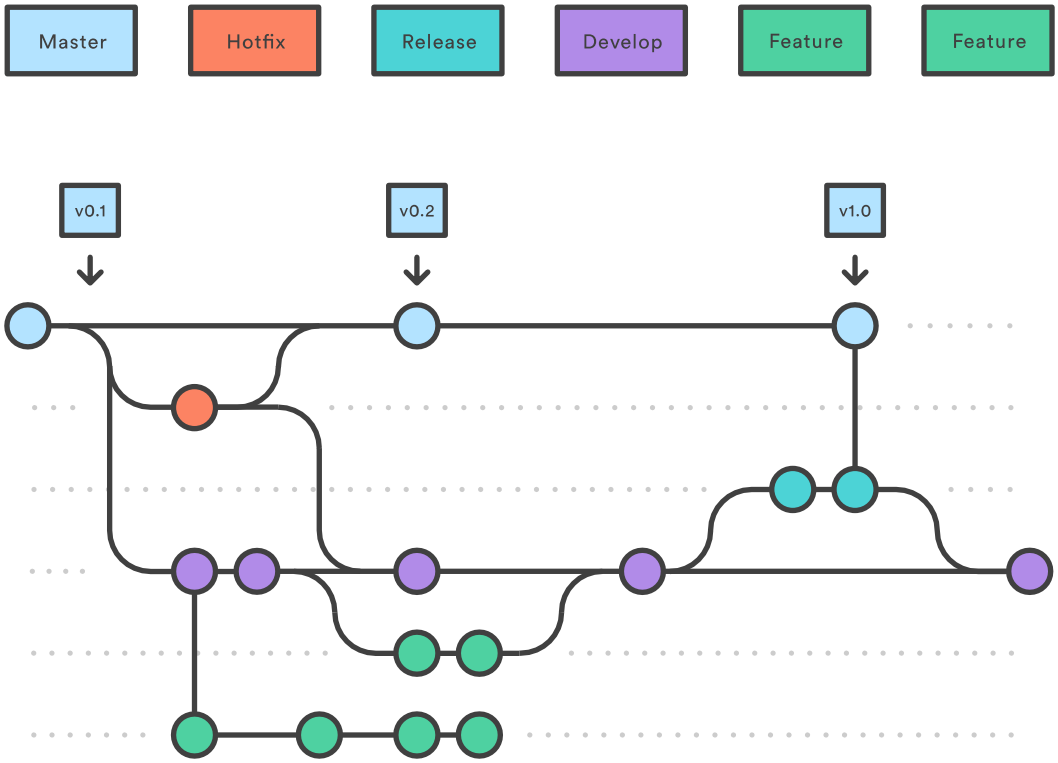


Imagen 9. Ramas y proceso GitFlow

- **Master:** Es la rama que contiene el código estable desplegado en el ambiente productivo.

- **Develop:** Es la rama en la cual se integran los desarrollos de las funcionalidades, no pueden llegar Push sin Pull Request desarrolladas por cada desarrollador, no pueden llegar cambios sin Pull Request.

- **Feature:** Es una rama en la que cada desarrollador realiza el desarrollo de su historia o corrección de defectos dentro del Sprint, debe tener Push diarios.

- **Release:** Se usa para mantener una versión previa al paso a producción, los cambios solo pueden llegar con Pull Request

- **Hotfix:** Se usa para hacer correcciones sobre la versión en Master, los cambios se pueden hacer directamente en la rama, pero se deben pasar a Master y Develop por medio de Pull Request.

2.3.IDE

Para todos los desarrollos se debe usar un IDE moderno, que tenga integrado el versionamiento de código con GIT, para favorecer los casos que requieran hacer Merge,

también debe soporta la integración de la herramienta SonarLint (descrita en el punto 2.5), a continuación, se presenta una tabla con la relación entre los IDE's recomendados por lenguaje de desarrollo.

LENGUAJE	ECLIPSE	VS CODE	VISUAL STUDIO
JAVA	SI	NO	NO
PHP	NO	SI	NO
ANGULAR	NO	SI	NO
.NET (C#, VB)	NO	NO	SI

Tabla 5. Lenguajes vs IDE

2.4.TDD

Test Driven Development o desarrollo guiado por pruebas es un estilo de desarrollo en el cual primero se diseña la prueba unitara de acuerdo al requerimiento funcional, después se desarrolla el código hasta que la ejecución de la prueba sea exitosa y posteriormente se refactoriza el código optimizándolo y depurándolo.

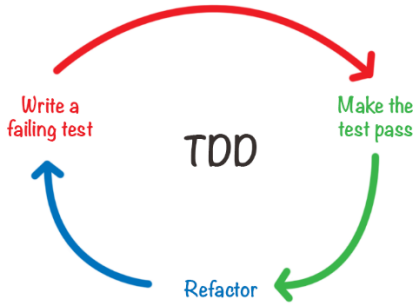


Imagen 10. Flujo TDD

Este estilo de desarrollo se debe usar para todo el código que sea suceptible de ser probado usando pruebas unitarias, como lógica en backend, servicios web, utilidades, librerias, controladores en MVC, etc; Su aplicación permitirá agilizar, simplificar y garantizar el uso de pruebas unitarias dentro de los procesos de construcción de software en la SIC.

2.5.SonarLint

SonarLint es una extensión o plugin para los IDEs, la cual permite el análisis estático de código en tiempo de desarrollo, con lo cual se optimiza el proceso de revisión y estandarización de código, los IDEs usados por los desarrolladores lo deben tener instalado, toda vez que en el paso de construcción de código se ejecutará el análisis estático como un Quality Gate, el uso de SonarLint disminuirá los retrabajos como resultado de dicho análisis.

La información se puede consultar en https://www.sonarlint.org/ para la instalación se accede a la sección del IDE a configurar:

- Eclipse: https://www.sonarlint.org/eclipse/
- Visual Studio: https://www.sonarlint.org/visualstudio/
- Visual Studio Code (VS Code): https://www.sonarlint.org/vscode/

2.6. Versiones del software

Para la numeración de versiones se usa el versionamiento tradicional el cual usa un identificador numérico de tres posiciones ##.##.## (mayor.menor.micro) con las siguientes consideraciones:

- **Mayor:** El software sufre grandes cambios y mejoras cuyo alcance es mayoritario dentro de la funcionalidad del sistema.

- **Menor:** El software sufre pequeños cambios y/o correcciones de errores, puede ser la adiciona de una característica o funcionalidad nueva.

- **Micro:** Se aplica una corrección al software, y a su vez sufre pocos cambios, usualmente corrección de defectos o ajustes mínimos de funcionalidad.

La primera versión de cualquier sistema ser la 1.0.0 y se asignara en el momento de salida a producción la primera vez, para los sistemas que ya tienen una versión actualmente, esta se deberá ajustar a este lineamiento continuando con la numeración o versión que esté en producción.

3. CONSTRUCCIÓN

La construcción de los artefactos de código es un mecanismo que permite habilitar la aplicación de la metodología DevOps, es por esto por lo que se estandariza el uso de una herramienta para versionar el código y construirlo, de tal forma que los flujos de construcción automatizados puedan descargar el código y correr la tarea de construcción sin intervención humana.

3.1. GIT Básico

En esta sección se describe el funcionamiento básico de GIT, sin embargo, es necesario que cada desarrollador lo aprenda a profundidad y que en la medida de los posible se apoye del uso del IDE para la gestión de versiona miento.

Para configurar la cuenta de Azure DevOps en el equipo local se deben ejecutar los siguientes comandos en la consola.

```
git config --global credential.helper wincred

git config --global user.name "Nombre Funcionario"
git config --global user.email correo.funcionario@its2sicgov.onmicrosoft.com
```

Los datos en azul deben ser reemplazados por los datos del usuario que esté haciendo la configuración.

3.1.1. Clone

Es el evento que permite hacer una copia local de un repositorio remoto, la URL se obtiene desde la sección de repositorios en Azure DevOps, a través del botón Clone.

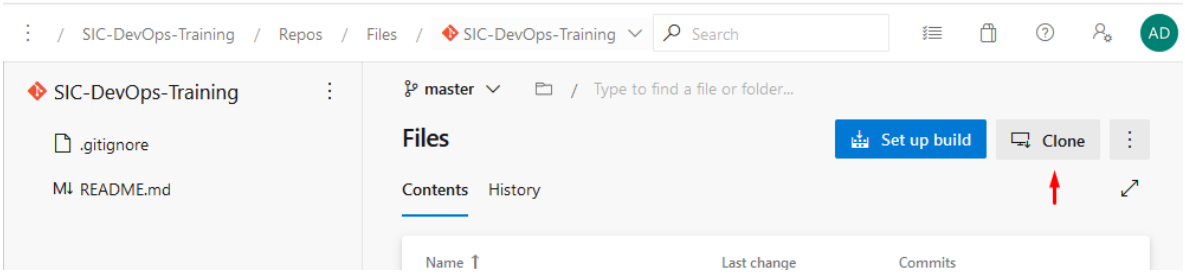


Imagen 11. Obtener URL para hacer Clone

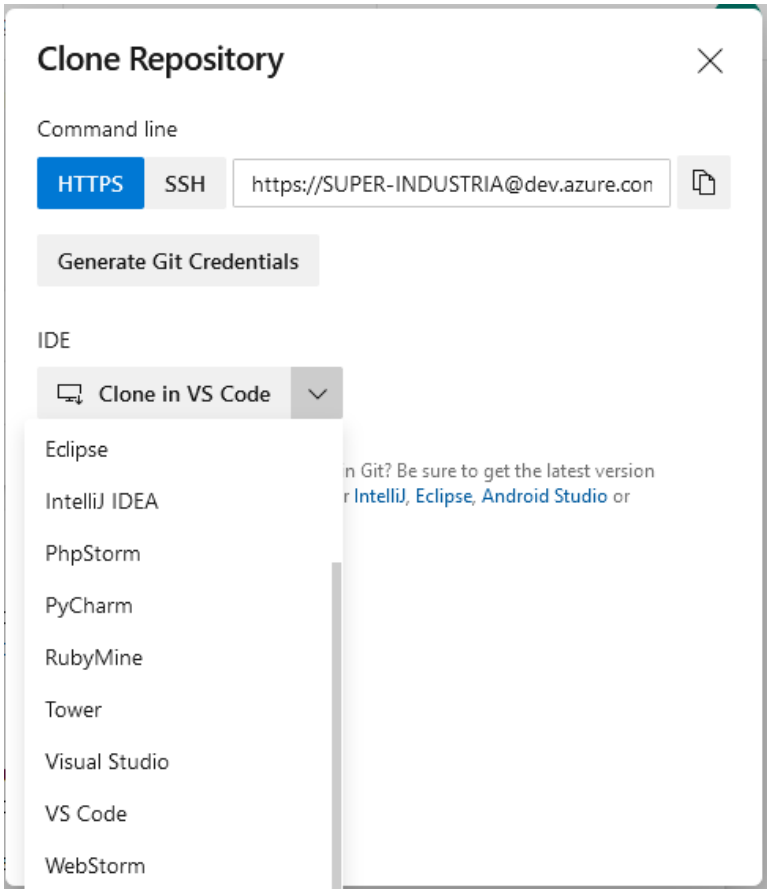


Imagen 12. Opciones para clonar

En la imagen anterior se Azure DevOps nos permite copiar la URL de Clonación o iniciar el proceso directamente en el IDE, a través del botón **Clone in XXX** este despliega la lista de IDEs compatibles para el proceso.

3.1.2. Commit Push

El proceso para hacer la publicación de un cambio en Azure DevOps se componente dos pasos, el primero es ejecutar el comando **commit** en el repositorio local, después de esto se debe ejecutar el comando **push** el cual sube el cambio al repositorio remoto.

En la siguiente imagen se muestra el detlle paso a paso de las secuencia para llevar el código al repositorio remoto, si bien el IDE ejecuta los pasos listados los eventos de Commit y Push si los debe generar el desarrollador.

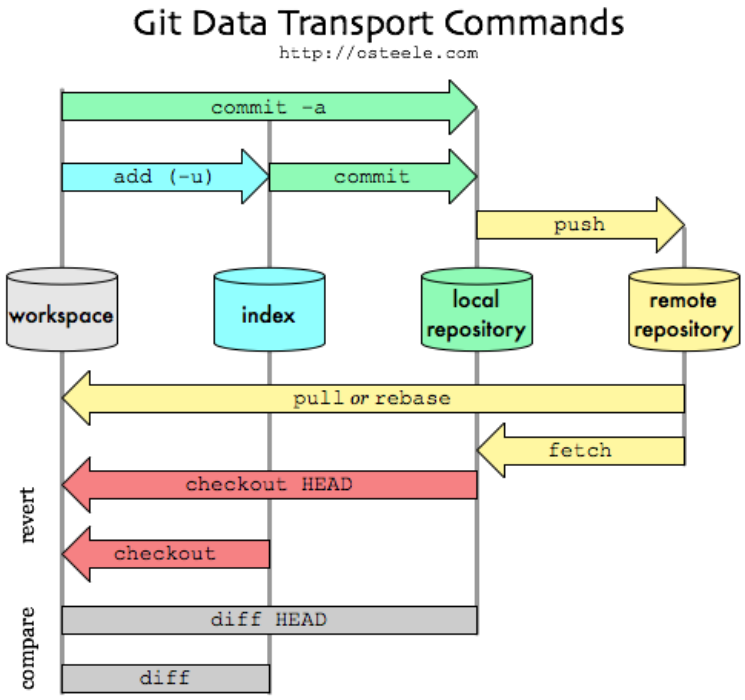


Imagen 13. Secuencias de interacción con repositorio remoto

3.1.3. Crear Branch

Como se describió en la sección 2.2 el uso de ramas es obligatorio para el desarrollo usando la metodología DevOps, para la creación de reamas se puede hacer desde la consola de comandos o desde Azure DevOps asi:

```
git branch nombre/branch
```

Nota: este comando crea la rama localmente, es necesario hace un push para que esta se cree en el repositorio remoto.

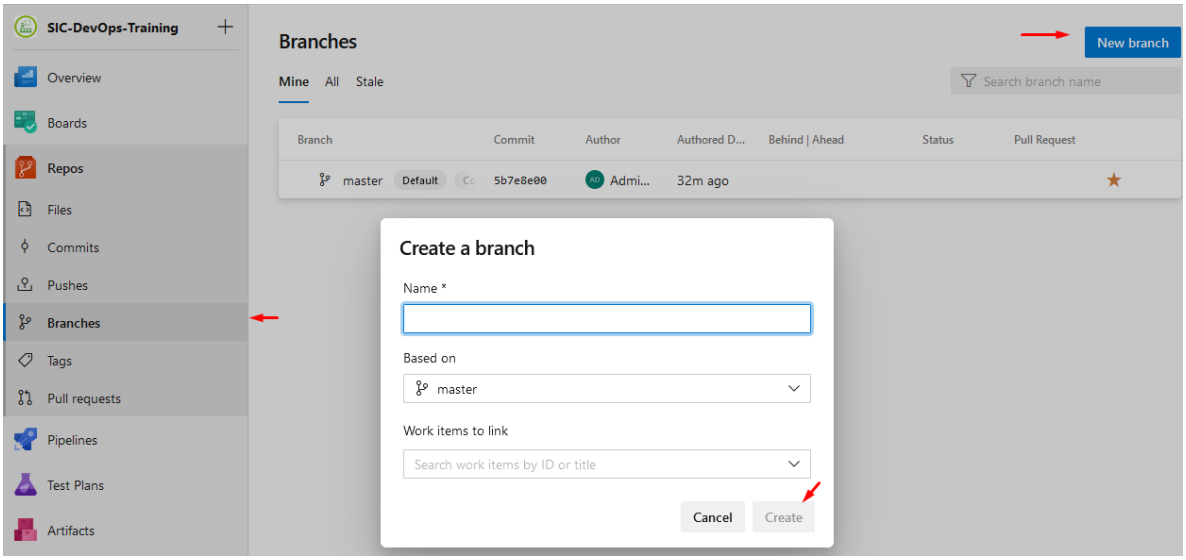


Imagen 14. Creación de ramas en Azure DevOps

Para la creación de ramas en el repositorio remoto, dentro de Azure DevOps, en la opción Repos -> Branches, se habilita la opción New Branch, la cual hace la creación de la rama, para esto pide el nombre, se debe seguir el lineamiento de la sección 2.2, basado en, depende de la rama padre de la cual se quiera sacar la copia y Work Item to Link permite asociar un Item de trabajo, para que cuando se haga el push, Azure DevOps identifique y automáticamente le cambie el estado.

3.1.3.1. Políticas de Branch

Para mantener la consistencia y estabilidad del código se establecen controles para el proceso de Merge a las ramas Master y Develop, para eso se usa la opción Branch policies.

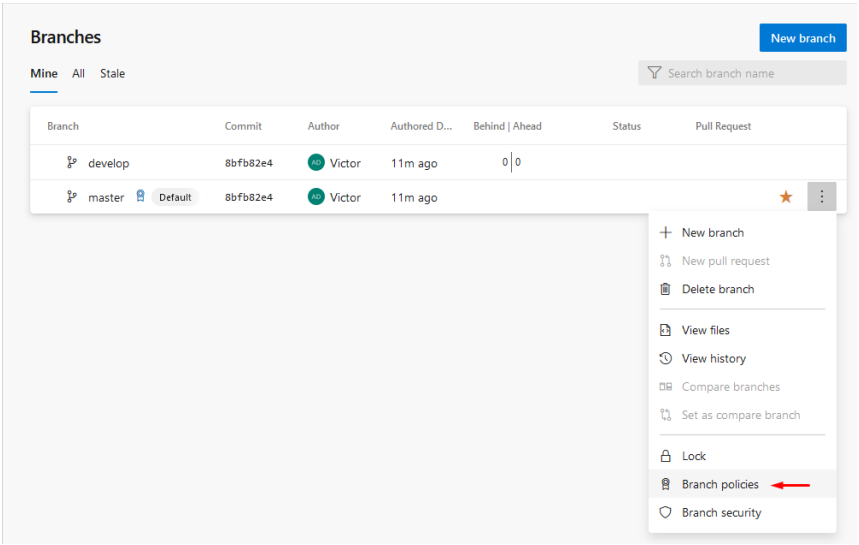


Imagen 15. Políticas de las ramas

Las ramas Master y Develop deben existir para cada repositorio y deben tener configurar las siguientes políticas:

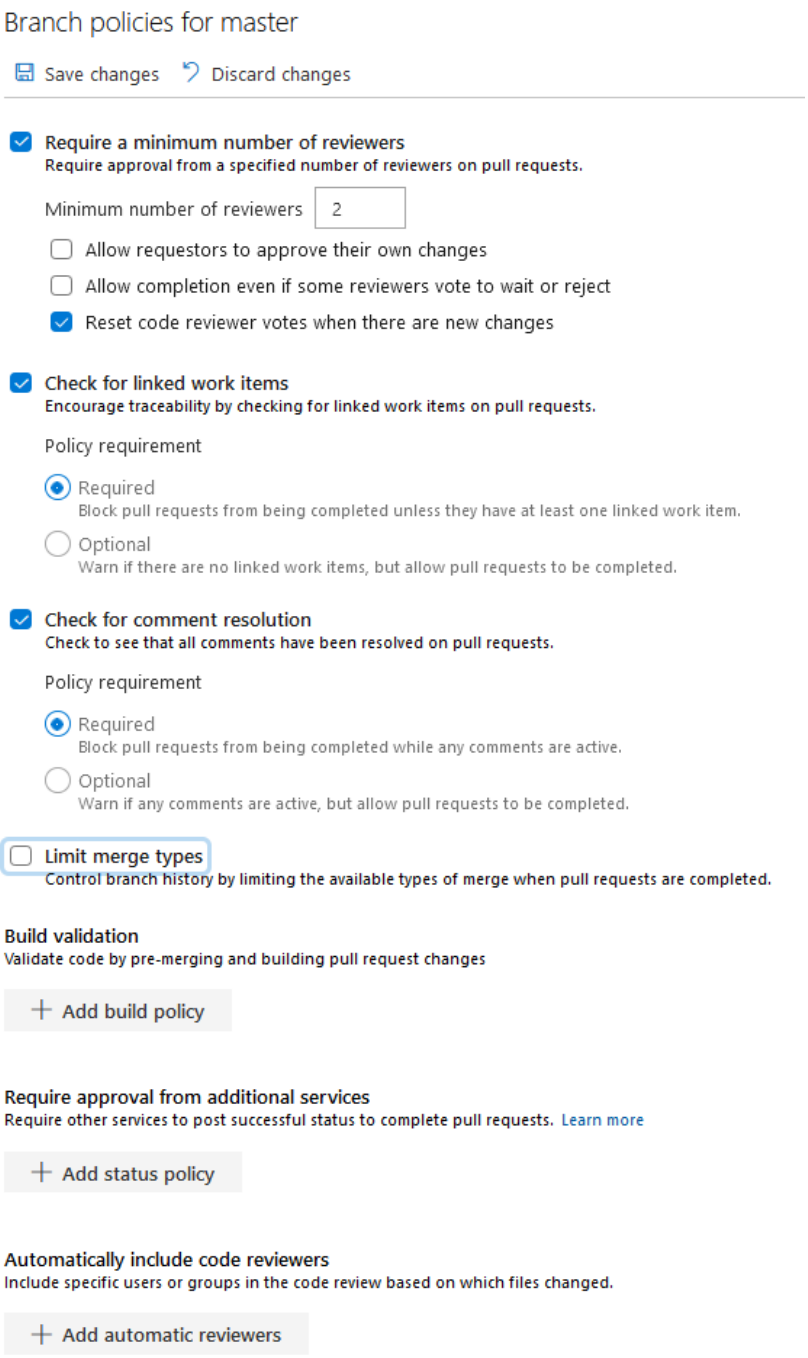


Imagen 16. Creación de políticas en ramas

NOTA: La aplicación de la política de revisión de pares (aprobadores), se aplicará solo si el equipo tiene la capacidad (perfiles y recursos) para hacer estas revisiones.

3.1.4. Pull Request

El evento Pull Reques permite volver a unir dos ramas automáticamente, para esto se debe entrar por la opción Repos -> Pull Request y hacer click en el boton New Pull request.

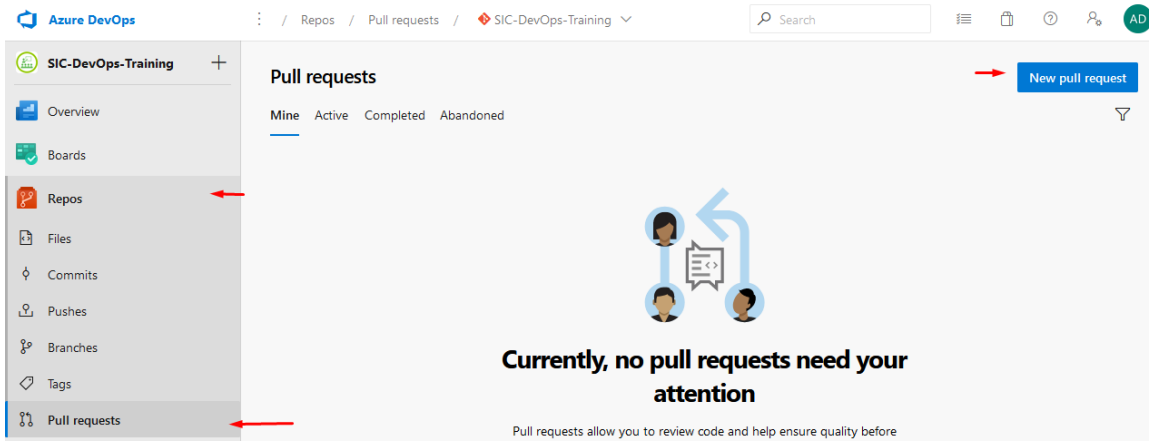


Imagen 17. Creación de pull request

El Pull Request solicitará el Branch origen y destino para hacer la unificación, una vez creado presenta los cambios que se van a unir, y pide la aprobación de los revisores, este revisor debe ser un par del desarrollador y no el mismo.

Por configuración de la política se pide por lo menos un revisor, sin embargo, en los proyectos que tengan más de 3 desarrolladores, se recomienda que sean mínimo dos revisores.

3.2. Integración Eclipse con Azure DevOps

Para lograr una integración sencilla desde Eclipse se utiliza el cliente nativo de GIT en Eclipse, sin embargo, este cliente no tiene acceso a las credenciales de Windows, para lograr la autenticación se necesita generar unas credenciales de GIT para el usuario del dominio, de la siguiente manera:

Al clonar el repositorio hacer click en el botón **Generate Git Credentials**

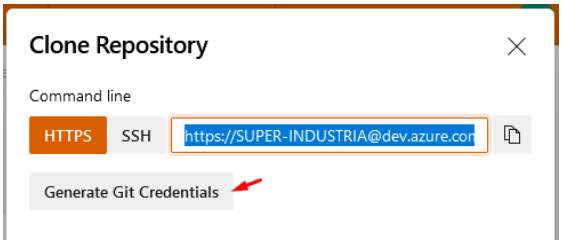


Imagen 18. Generación de credenciales de Git

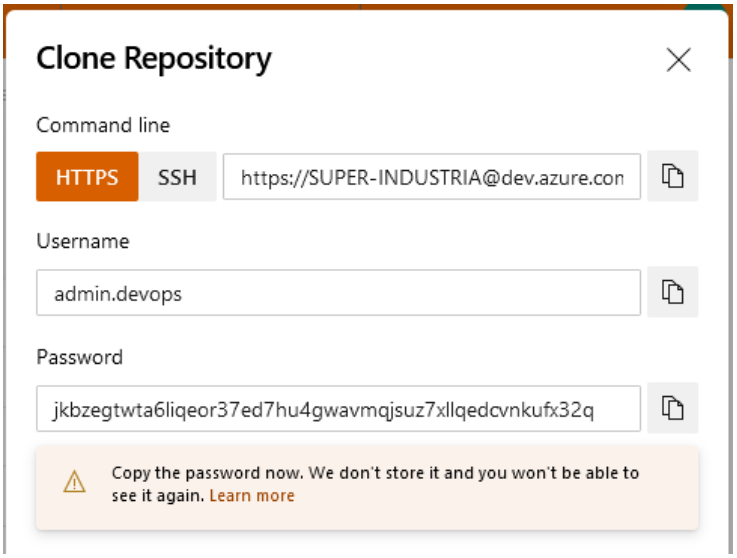


Imagen 19. Credenciales generadas

Una vez generadas se debe clonar el repositorio desde Eclipse, para esto se hace lo siguiente:

En la vista de repositorios (Window->Show View->Other-> Git Repositories

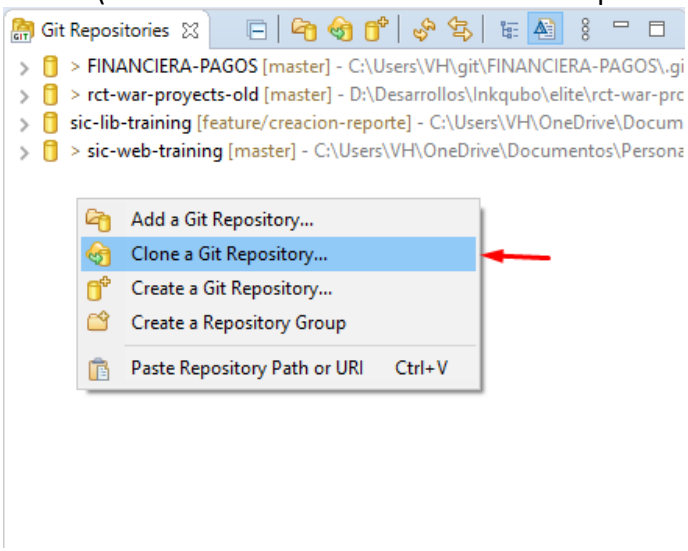


Imagen 20. Clonación del repositorio

La URL que se usa se basa en la que nos genera Azure DevOps pero se cambia la organización por el usuario generado, ejemplo:

Generada:

https://SUPER-INDUSTRIA@dev.azure.com/SUPER-INDUSTRIA/SIC-DevOps-Training/_git/sic-lib-training

Modificada (la que se usa desde Eclipse)

https://admin.devops@dev.azure.com/SUPER-INDUSTRIA/SIC-DevOps-Training/_git/sic-lib-training

3.3. Herramienta de construcción

La herramienta de construcción de artefactos se debe unificar para los proyectos de la SIC, esta depende del lenguaje y se describe a continuación.

3.3.1. JAVA

Para JAVA, se define que la herramienta estándar a utilizar es **Maven**, dado que es la herramienta que tiene el primer lugar en MarketShare para JAVA empresarial, esto hace que haya una gran cantidad de librerías soportadas en repositorios públicos, y documentación para su uso.

Es necesario aclarar que la implementación de Maven dentro de los proyectos existentes se hará con un diagnostico que permita identificar la mejor manera para hacerlo, puede ser que los proyectos ya cuenten con **ANT** u otra herramienta que permita hacerlo.

3.3.2. Angular

Para angular se estandariza la construcción de artefactos usando el cliente de línea de comandos de Angular NG-CLI, usando el comando **ng build**.

3.3.3. PHP

En el caso de PHP, se usa el proceso de empaquetado provisto por el framework usado, para los proyectos nuevos se recomienda el uso de Laravel, en los proyectos viejo se debe hacer un análisis para identificar el mecanismo adecuado de empaquetamiento.

3.4. Parametrizaciones

Como el despliegue automático se hace desde Azure DevOps, el artefacto generado en la construcción debería estar aislado del ambiente de despliegue, esto quiere decir que toda la parametrización necesaria para su funcionamiento no debe depender de archivos externos en el sistema sino de archivos internos del artefacto, cuando exista una dependencia con el ambiente esta configuración se debe hacer a través de las variables de entorno del sistema.

4. PRUEBAS

La metodología DevOps basa su éxito en garantizar que cada versión que se libera cumple con un ciclo de pruebas automatizado que permite identificar errores rápidamente y de esta forma mantener la estabilidad del sistema por impacto en funcionalidades fuera del alcance del desarrollo, básicamente es tener un set de pruebas automatizado que permita hacer las pruebas de regresión del sistema, las cuales se describen a continuación.

4.1. Pruebas Unitarias

Las pruebas unitarias son las que lleva a cabo el desarrollador, estas se deben construir para todas las funcionalidades del código y se debe evaluar el cubrimiento del código desarrollado, esto significa que las pruebas unitarias garantizan que si una funcionalidad no modificada genera un resultado diferente al esperado hay que hacer un ajuste en el código no modificado, para mantener la estabilidad del sistema.

El porcentaje de cubrimiento de pruebas unitarias representa la cantidad de líneas que se prueban de forma unitaria versus las que no, este es un indicador que permite al equipo estar tranquilo con la estabilidad del sistema, a mayor porcentaje mayor estabilidad.

A continuación, se presenta el listado de herramientas que se deben usar para la construcción y ejecución de pruebas unitarias.

LENGUAJE	PRUEBAS	CUBRIMIENTO
JAVA	JUNIT	JACOCO
PHP	PHPUnit	
ANGULAR	KARMA	KARMA, JASMIN, ISTANBUL
.NET (C#, VB)	VISUAL STUDIO	VISUAL STUDIO

Tabla 6. Herramientas pruebas unitarias y cubrimiento

NOTA: De acuerdo con la nota de la sección 1, los proyectos nuevos que usan las tecnologías descritas deben ejecutar las pruebas unitarias dentro de su proceso de desarrollo, los proyectos viejos deberán hacer un análisis de la viabilidad y plan de implementación.

4.2. Pruebas de integración

Una vez se despliegan los componentes en los ambientes de pruebas o producción, es posible lanzar pruebas a los servicios para identificar si el despliegue se hizo satisfactoriamente, por ejemplo, hacer un HTTP GET a la pagina del login, o consumir un servicio que trae información parametriza.

Para esto se usa la herramienta SOAPUI, la cual permite hacer peticiones HTTP SOAP/REST y ejecutarlas por la línea de comandos en el pipeline de despliegue, una vez este se complete.

4.3. Pruebas Funcionales.

Las pruebas funcionales buscan disminuir la carga de trabajo para los testers manuales, quienes se perfilan hacia el rol de Automatizador de Pruebas, con estas se prueban de forma automática los escenarios más comunes de uso de la aplicación, de tal forma que es un script el que prueba la interfaz de los sistemas, reduciendo los tiempos de los ciclos de pruebas y aumentando su confiabilidad.

Para esto se usa Selenium, parar crear los Scripts de pruebas, personalizarlos y posteriormente ejecutarlos con datos que permitan probar todos los flujos de los escenarios.

5. INTEGRACIÓN CONTINUA

El proceso de integración continua (CI) se hace a través de la herramienta Azure DevOps, usando la opción de Pipelines del menú Pipelines, esta permite configurar paso a paso la secuencia de construcción, de los flujos de integración continua.

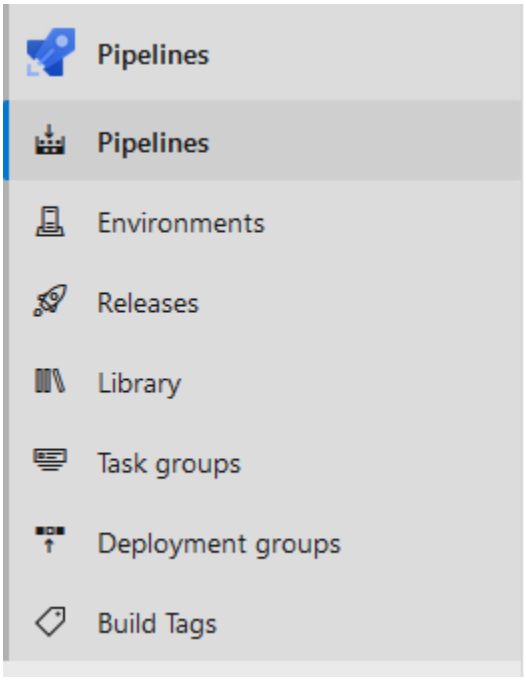


Imagen 21. Menú de integración continua

5.1. Análisis estático

El análisis estático de código se hace usando la herramienta SonarQube que es aprovisionada por la SIC y se integrará desde el pipeline de construcción, esto complementado con la herramienta SonarLint permite generar código de excelente calidad.

5.2. Repositorio de artefactos

El repositorio de artefactos creados como libreras reutilizables tipo Maven, nuGet, npm para los proyectos es Azure Artifacts provisto por Azure DevOps, en ese se mantienen las versiones de las librerías usadas de forma transversal por los proyectos.

5.3. Análisis de seguridad

PENDIENTE DE DEFINIR, ES NECESARIO VALIDAR CON EL ÁREA DE SEGURIDAD DE LA SIC LA PERTINENCIA DE SONARQUBE O LA HERRAMIENTA USADA ACTUALMENTE

5.4. Conexión a SonarQube

El servidor SonarQube esta instalado en la infraestructura de la SIC, y por lo tanto se debe configurar una conexión en cada proyecto que requiera ejecutar análisis de código, para esto se deben seguir estos pasos:

- Entrar a la configuración del proyecto.

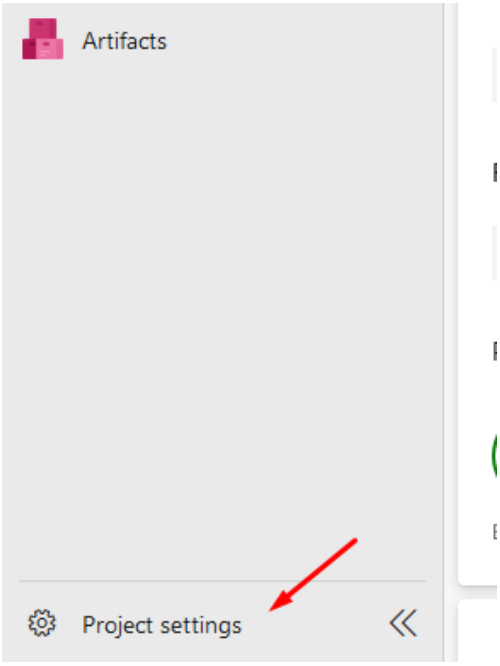


Imagen 22. Configuración del proyecto

- Entrar a la opción de conexiones a servicios externos para Pipelines.

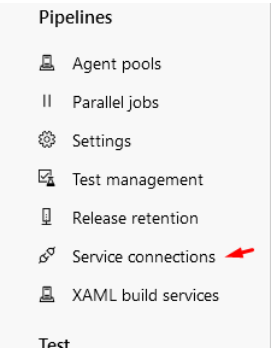
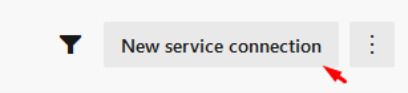


Imagen 23. Conexión a servicios externos

- Crear una nueva conexión y buscar SonarQube.



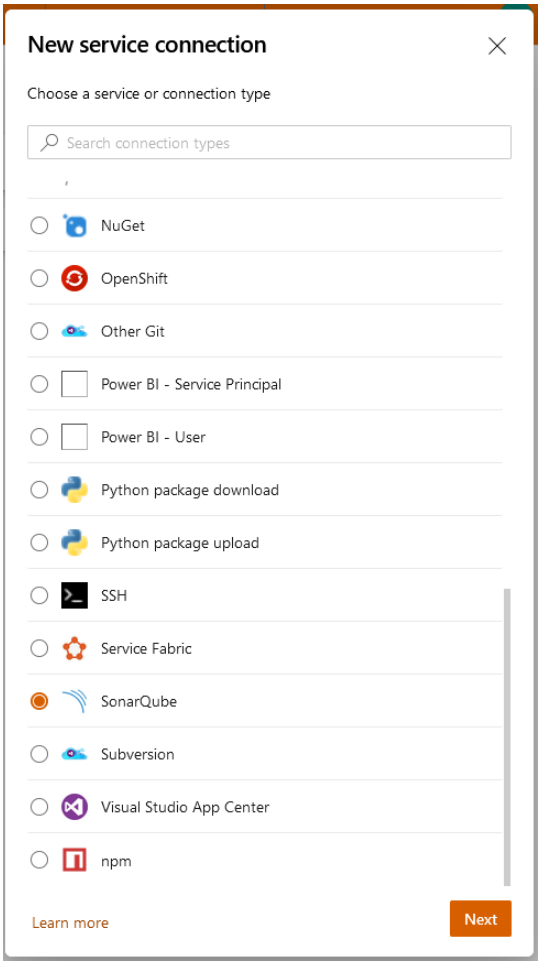


Imagen 24. Conexión a SonarQube

- Configurar la conexión con la siguiente información.

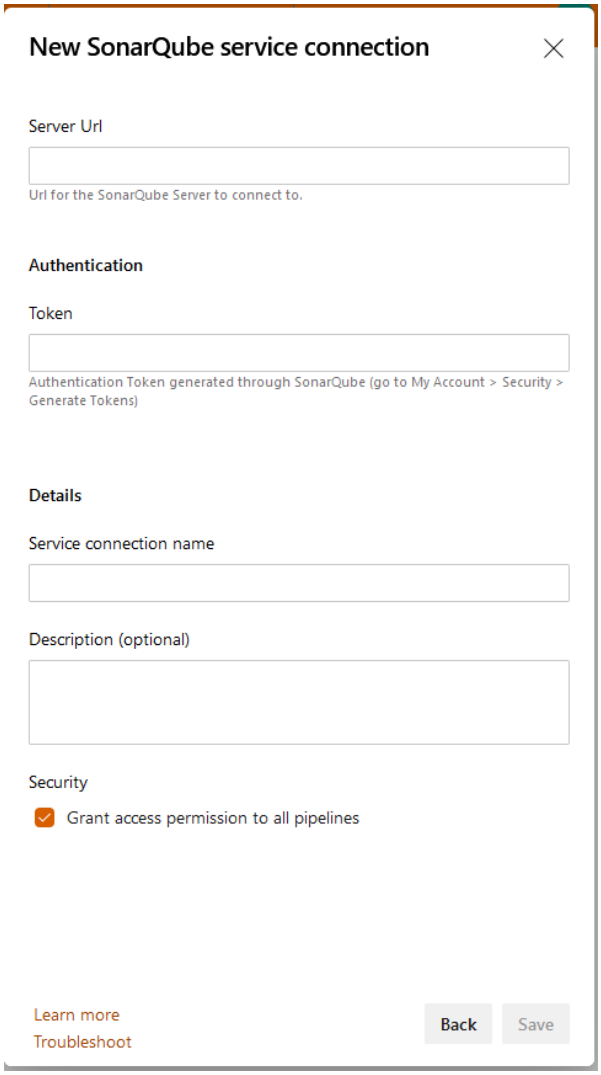


Imagen 25. Creación de conexión

Server URL: http://190.216.132.110:9000/sonar
Authentication Token: Se proveerá por correo electrónico a solicitud, porque cambia en el tiempo
Details Service connection name: SIC SonarQube
Details Description: Servidor SonarQube, para el análisis de código estático, alojado en la SIC
Security Grant Access permission to all pipelines: SI

5.5. Pipeline de Construcción

5.5.1. Pasos del Pipeline

Los pasos genéricos que deben llevar los pipelines garantizan la correcta construcción de los proyectos son los siguientes:

- Configuración de SonarQube
- Construcción del artefacto
- Ejecución de análisis de SonarQube (no se usa con Maven)
- Publicación de resultados SonarQube
- Validación de resultados SonarQube
- Publicar artefacto de despliegue

5.5.2. Proceso de construcción

Los flujos de construcción de los proyectos de código son soportados en la herramienta Azure DevOps, a través de los Pipelines, estos permiten la creación de la secuencia de pasos de automatización de construcción.

Para crear un flujo de construcción (Pipeline), se hace click en el botón Create Pipeline, este nos abre dirige a la funcionalidad para crear el flujo para el tipo de proyecto que se esté configurando.

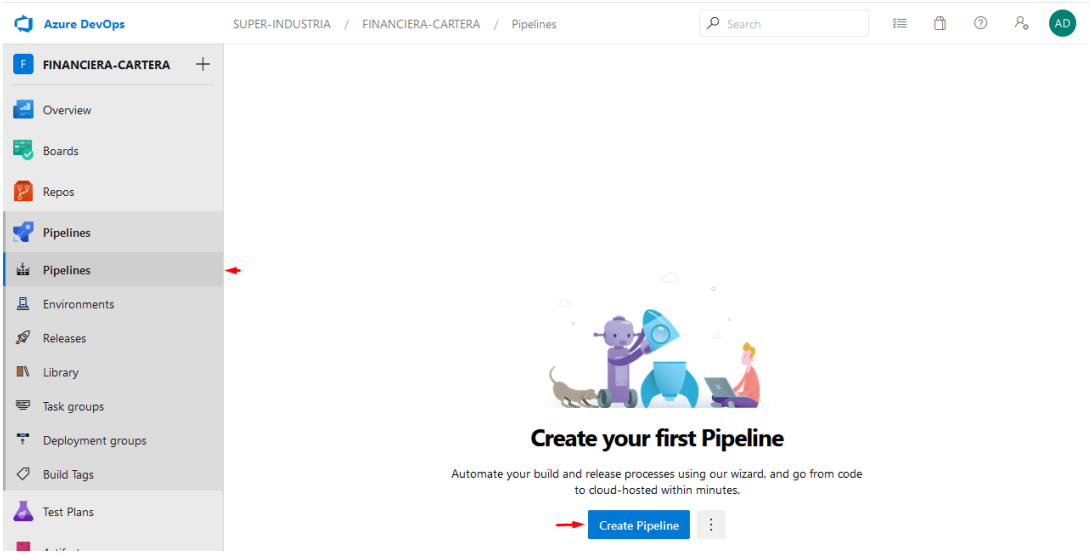


Imagen 26. Creación de flujo de integración continua

El primer paso para la creación del pipeline de construcción es seleccionar el origen del código fuente, para esto se usa el repositorio git del proyecto que se quiera construir.

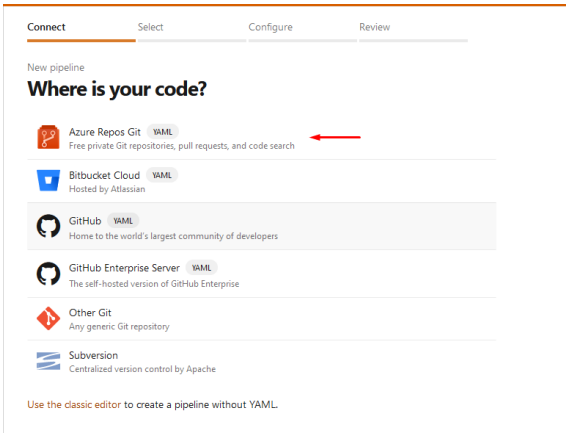


Imagen 27. Selección de repositorio para pipeline de CI

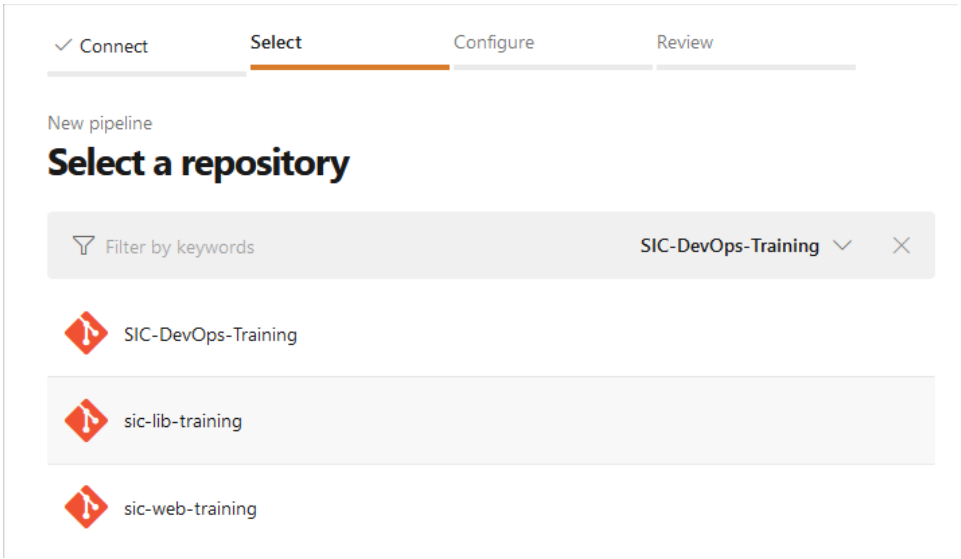


Imagen 28. Selección de repositorio para pipeline de CI

La herramienta Azure DevOps detecta el tipo de proyecto dentro del repositorio, para el caso de los proyectos Java se usa la configuración por defecto de Maven, con la cual

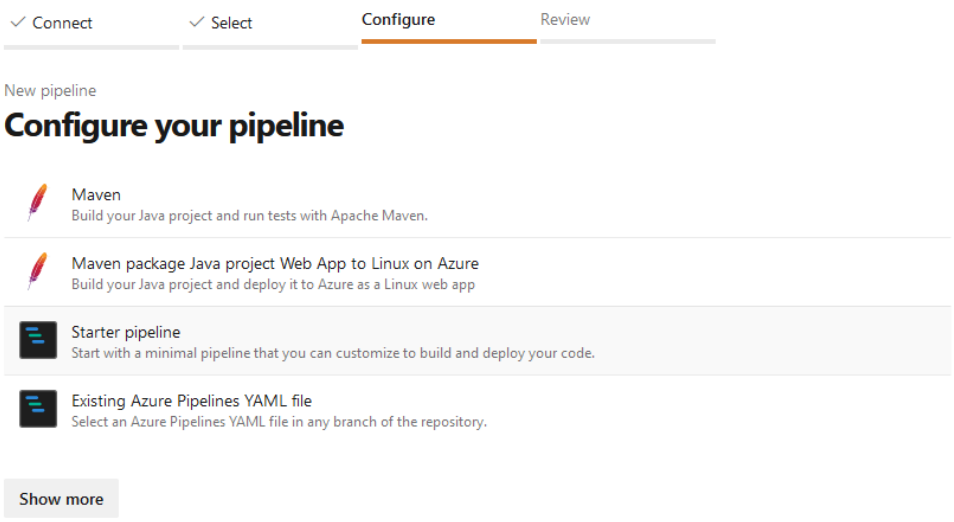


Imagen 29. Selección de repositorio para pipeline de CI

La configuración básica permite la creación del pipeline básico de construcción para proyectos Maven.

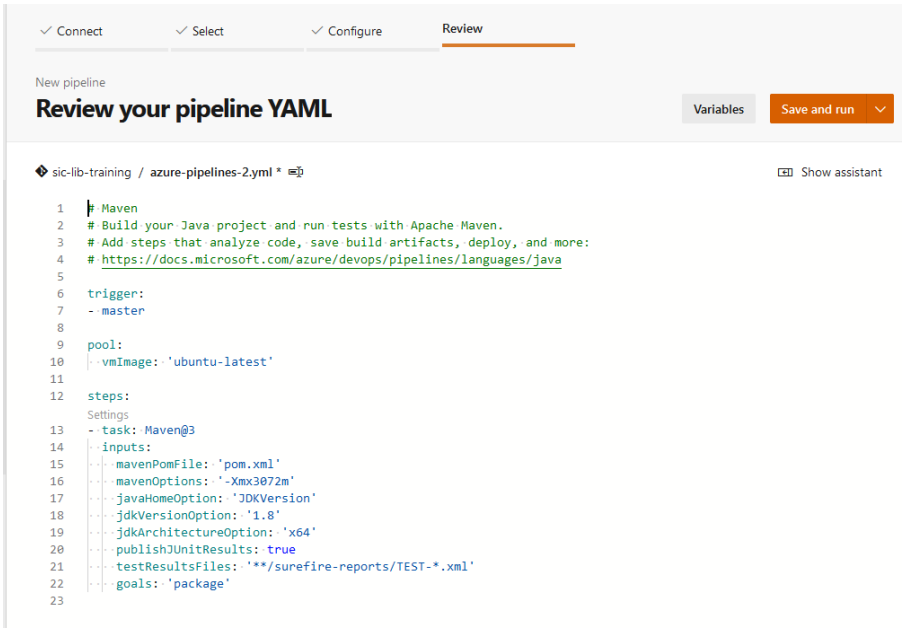


Imagen 30. Pipeline con la tarea de construcción de Maven

A la tarea básica de Maven se le deben agregar las siguientes líneas, de acuerdo con los pasos descritos en la sección 5.5.1:

- Configuración de SonarQube, se usa la tarea SonarQubePrepare versión 4

```
- task: SonarQubePrepare@4
  inputs:
    SonarQube: 'SIC SonarQube'
    scannerMode: 'Other'
```

- Construcción del artefacto (Maven) se usan las tareas DownloadSecureFile versión 1, Bash versión 3 y Maven versión 3, estas permiten descargar y copiar el archivo de configuración de Maven para publicar y consumir dependencias en Azure Artifacts, y la tare Maven construye el artefacto y permite ejecutar el análisis de cubrimiento de las pruebas unitarias y de Sonar Qube.

```
- task: DownloadSecureFile@1
  name: mavenSettings
  displayName: 'Download Maven Settings'
  inputs:
    secureFile: 'settings.xml'

- task: Bash@3
  inputs:
    targetType: 'inline'
    script: |
      # Write your commands here

      echo 'Configurar Maven Settings'
      mkdir ${HOME}/.m2
      mv $(mavenSettings.secureFilePath) ${HOME}/.m2/settings.xml

- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.8'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: true
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    codeCoverageToolOption: 'JaCoCo'
    codeCoverageFailIfEmpty: true
    goals: 'clean install'
    sonarQubeRunAnalysis: true
    sqMavenPluginVersionChoice: 'latest'
```

- Ejecución de análisis de SonarQube (no se usa con Maven), en caso de que no sea un proyecto Java-Maven, se debe ejecutar la tarea SonarQubeAnalyze versión 4

```
- task: SonarQubeAnalyze@4
  displayName: 'Run Code Analysis'
```

- Publicación de resultados SonarQube, se usa para generar el reporte en SonarQube con la tare SonarQubePublish versión 4

```
- task: SonarQubePublish@4
  inputs:
    pollingTimeoutSec: '300'
```

- Validación de resultados SonarQube, se usa la tarea sonar-buildbreaker versión 8, para indicarle a Azure DevOps que la construcción fue fallida en caso de que no se cumplan los quality gates configurados.

```
- task: sonar-buildbreaker@8
  inputs:
    SonarQube: 'SIC SonarQube'
```

- Publicar artefacto de despliegue (si no es una librería que se publica en Artifacts), este artefacto queda disponible para ser desplegado por un pipeline de despliegue, se usa la tarea PublishPipelineArtifact versión 1.

```
- task: PublishPipelineArtifact@1
  inputs:
    targetPath: '$(Build.Repository.LocalPath)/target/NOMBRE_DE_ARCHIVO.EXT'
    artifact: 'NOMBRE DE ARTEFACTO'
    publishLocation: 'pipeline'
```

Un archivo completo de despliegue es el siguiente:

```
# Maven
# Build your Java project and run tests with Apache Maven.
# Add steps that analyze code, save build artifacts, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/java

trigger:
- master
```

```yaml
pool:
  vmImage: 'ubuntu-latest'

steps:
- task: SonarQubePrepare@4
  inputs:
    SonarQube: 'SIC SonarQube'
    scannerMode: 'Other'

- task: DownloadSecureFile@1
  name: mavenSettings
  displayName: 'Download Maven Settings'
  inputs:
    secureFile: 'settings.xml'

- task: Bash@3
  inputs:
    targetType: 'inline'
    script: |
      # Write your commands here

      echo 'Configurar Maven Settings'
      mkdir ${HOME}/.m2
      mv $(mavenSettings.secureFilePath) ${HOME}/.m2/settings.xml

- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.8'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: true
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    codeCoverageToolOption: 'JaCoCo'
    codeCoverageFailIfEmpty: true
    goals: 'clean install'
    sonarQubeRunAnalysis: true
    sqMavenPluginVersionChoice: 'latest'

- task: SonarQubePublish@4
  inputs:
```

```
    pollingTimeoutSec: '300'

- task: sonar-buildbreaker@8
  inputs:
    SonarQube: 'SIC SonarQube'
```

NOTA: Esta secuencia de pasos se puede usar de forma gerencia, sin embargo, por la naturaleza de los proyectos es posible que se necesite una personalización de los pasos, pero en general se debe cumplir con los pasos aca descritos.

6. DESPLIEGUE CONTINUO

El proceso de despliegue continuo es el encargado de tomar los artefactos de construcción (CI) y desplegarlos de forma automáticamente en los ambientes dispuestos para las pruebas y operación de los sistemas, por lo tanto, es necesario que existan previamente los pipelines de integración continua y que estos publiquen los artefactos de despliegue de forma

Para la creación de los Pipelines de despliegue se deben seguir los siguientes pasos.

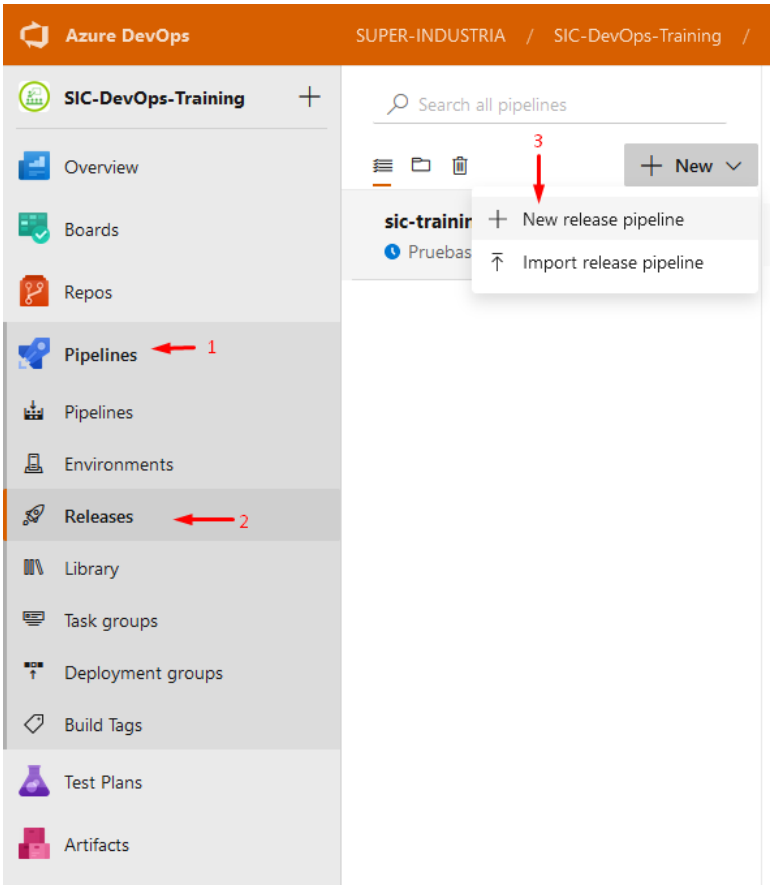


Imagen 31. Acceso a la creación de Pipeline de Despliegue

Una vez creado, se debe iniciar con un proceso vacio, cuando se estandaricen las plantillas de despliegue por tecnología se podrá seleccionar la plantilla que ahorra el trabajo de configuración.

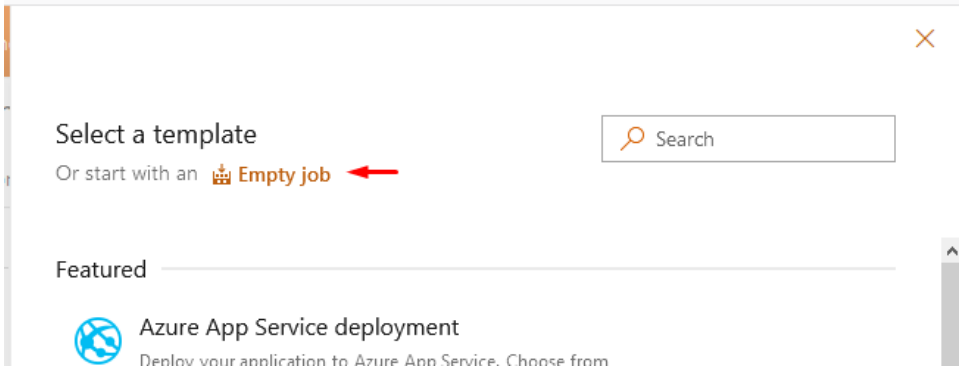


Imagen 32. Seleccionar proceso vacio

Una vez creado seleccionado el proceso se debe crear el primer Stage (etapa o ambiente) de despliegue, si es para la rama develop, se debe nombrar como Desarrollo, si es para la rama master, se debe nombrar como Pruebas

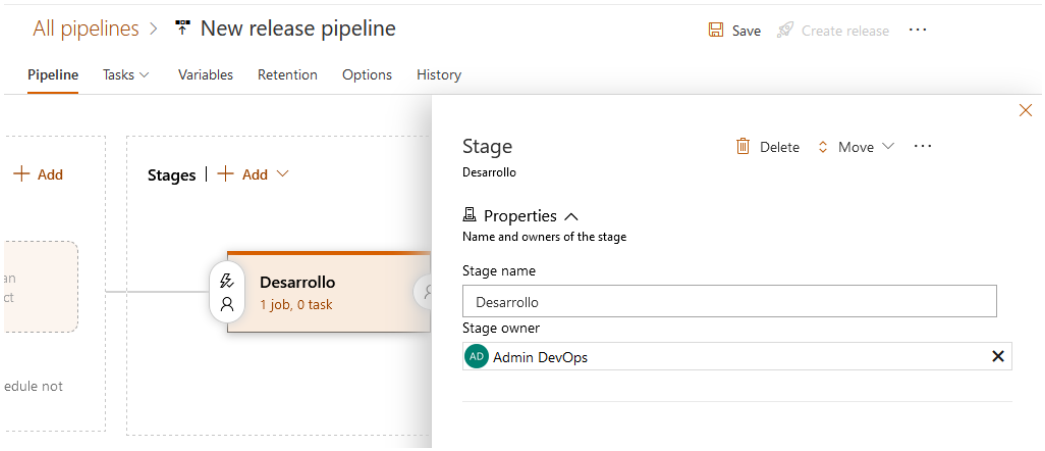


Imagen 33. Crear primer Stage

Imagen 33. Nombramiento del Pipelina

El pipeline que se esté creando debe ser nombrado con la siguiente estructura, para lograr identificarlo fácilmente.

[nombre-repositorio]-[rama]

Ejemplo:

sic-lib-training-develop

Una vez nombrado el pipeline se debe seleccionar el pipeline de integración que originará el despliegue del componente

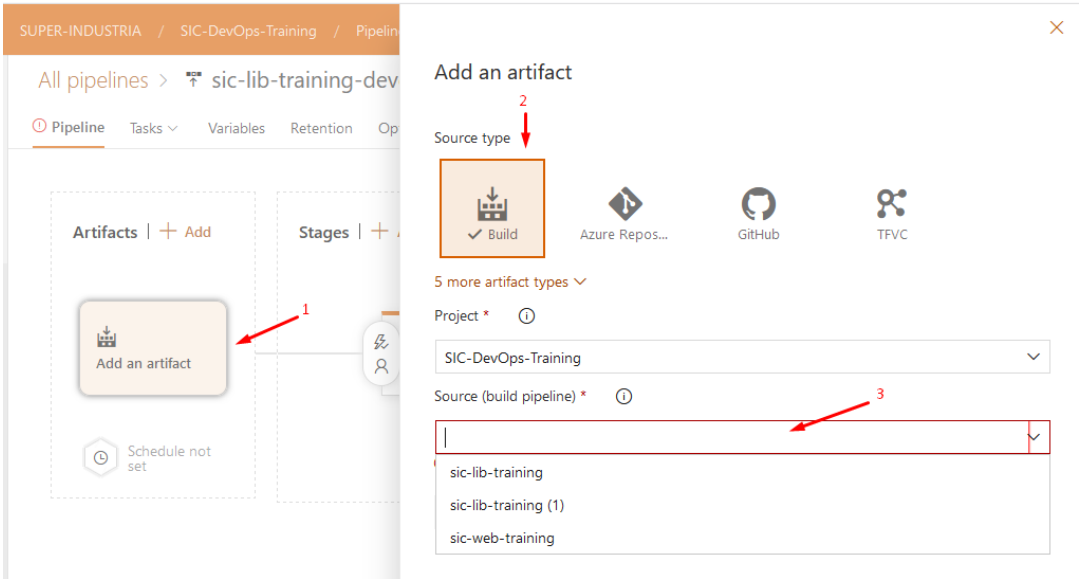


Imagen 34. Selección del pipeline de integración

Se debe configurar el lanzador (trigger) para que cada build exitoso desencadene un proceso de despliegue automático

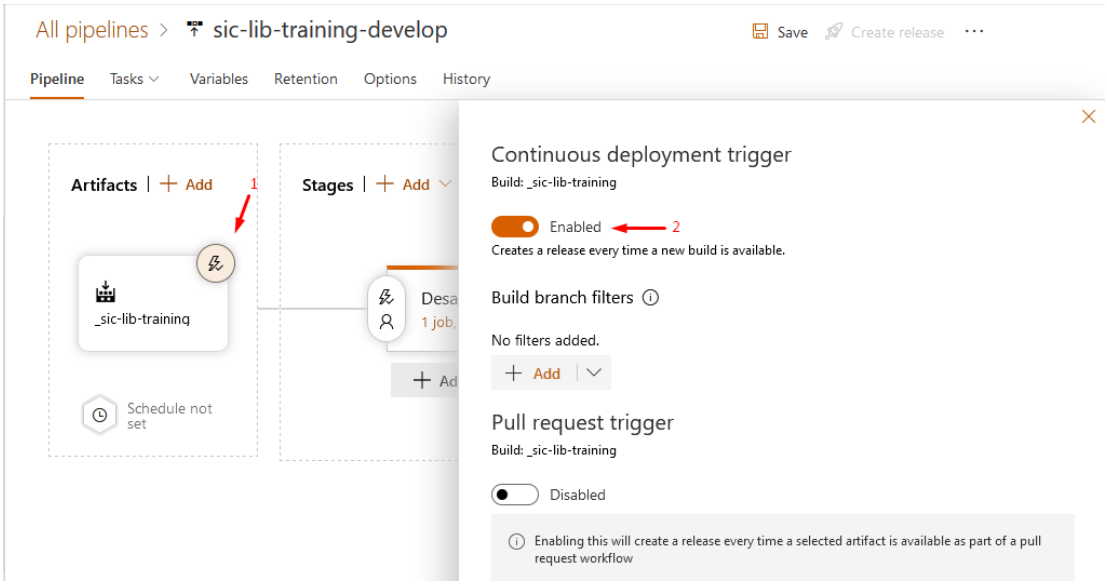


Imagen 35. Trigger de ejecución automática

Para agregar los pasos específicos del proceso de despliegue, se debe seleccionar el Stage y agregar los pasos necesarios.

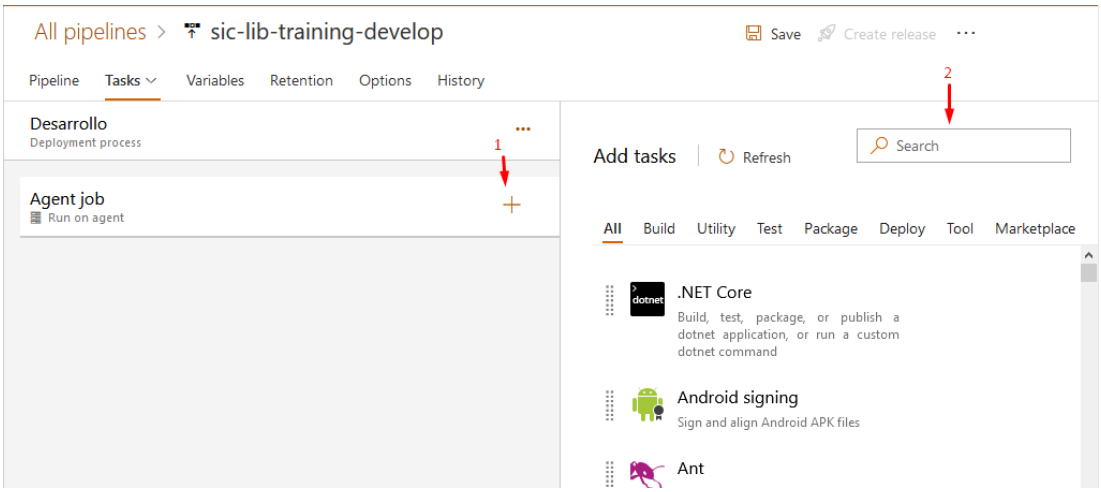


Imagen 36. Adición de pasos de despliegue

6.1. Pipeline de despliegue

El pipeline de despliegue se configura para cada rama, es decir debe existir por lo menos uno para desarrollo y uno para master (esto no implica que no se pueda tener uno para las features/*), y se encargan de tomar el artefacto generado por el pipeline de CI, y

desplegarlo según sea el caso, como el pipeline de integración continua se puede llegar a reutilizar para las dos ramas se tienen dos posibles esquemas, son los siguientes.

NOTA: La configuración y ejecución de los pipelines, asi como se describe en la nota de la sección 1, se hará de acuerdo con la capacidad y análisis de viabilidad y plan de implementación de cada proyecto.

6.1.1. CI Exclusivo

Corresponde al esquema en el cual cada rama tiene si propio pipeline de integración continua, por lo que en consecuencia el pipeline de despliegue continuo toma el artefacto correspondiente a la construcción especifica de la rama.

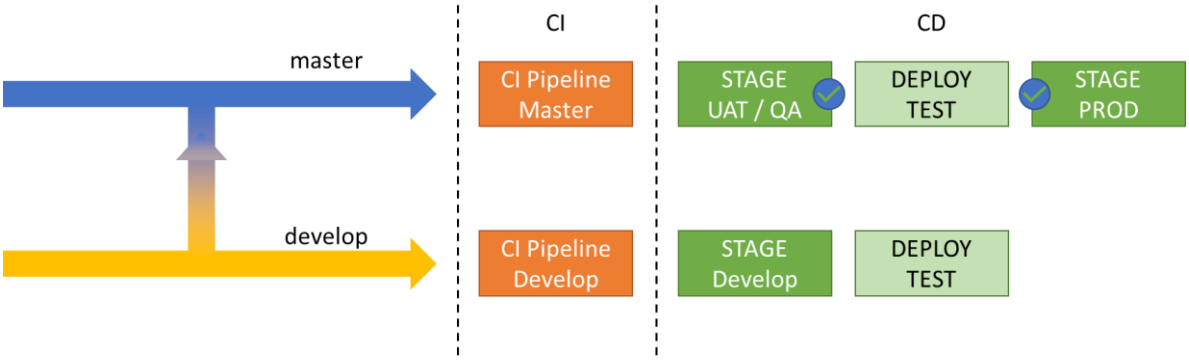


Imagen 37. Esquema de CI exclusivo

6.1.2. CI Compartido.

El esquema de CI compartido permite tener un único pipeline de integración continua, y que el trigger del despliegue identifique la rama origen para iniciarse, en este caso también se deben tener mínimo dos pipelines de despliegue 1 para master y 1 para develop

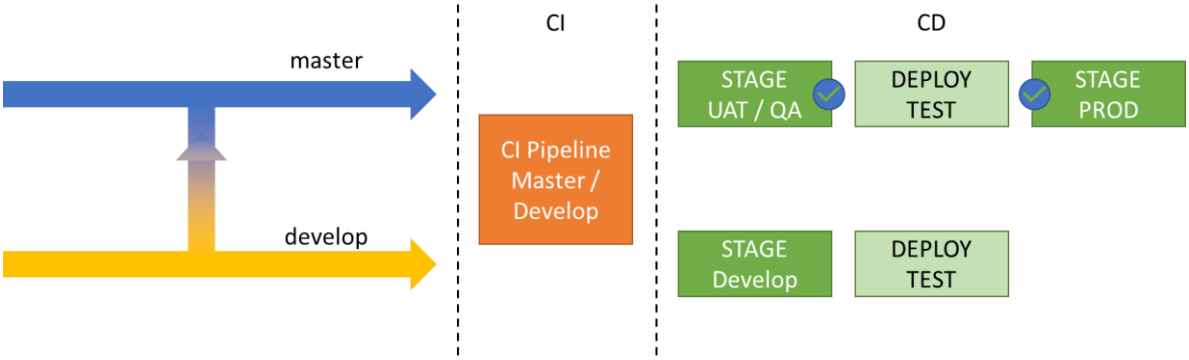


Imagen 38. Esquema de CI exclusivo

En este caso hay cos consideraciones a tener en cuenta:

1. El trigger del pipeline de integración (build), debe incluir las ramas master y develop asi:

```
trigger:
- master
- develop
```

2. El pipeline de despliegue de acuerdo con lo requerido deberá ejecutarse solo para la rama adecuada asi:

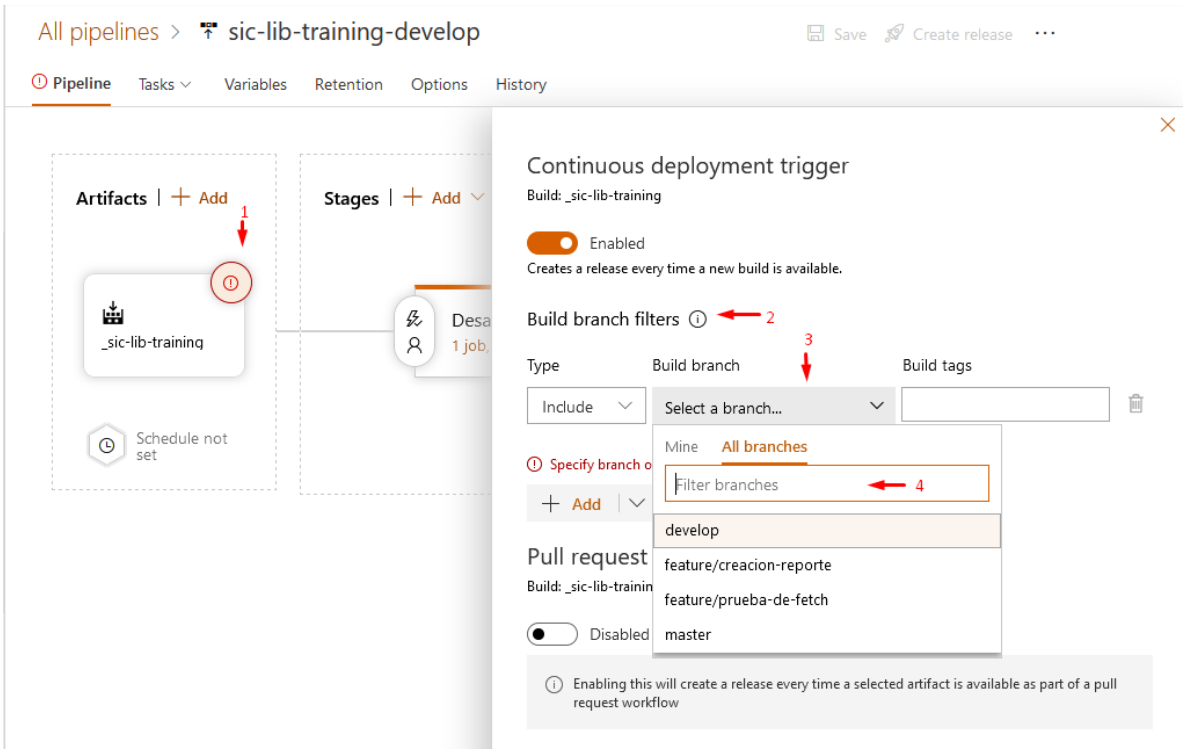


Imagen 39. Selección de rama en esquema compartido

6.2.Aprovisionamiento de ambientes

Con la habilitación de tecnologías como contenedores y OpenShift, la cual se encuentra en proceso de establecimiento, los ambientes podrán ser aprovisionados como código, pero hasta que no se cuente con esta capacidad completamente implementada, se deberá hacer la solicitud de aprovisionamiento de ambientes de forma manual a través de los procesos de la mesa de servicios.

6.3. Proceso de autorización

Los procesos de despliegue en los ambientes productos requieren de dos etapas de autorización, la primera es la aprobación funcional del desarrollo desplegado en el ambiente de pruebas como soporte de estar de acuerdo con la funcionalidad la segunda es la aprobación por parte del comité de cambios previo al despliegue automático del ambiente productivo de los sistemas.

Esta configuración se hace dentro del pipeline en los eventos post-deploy del Stage de Pruebas y pre-deploy del ambiente productivo.

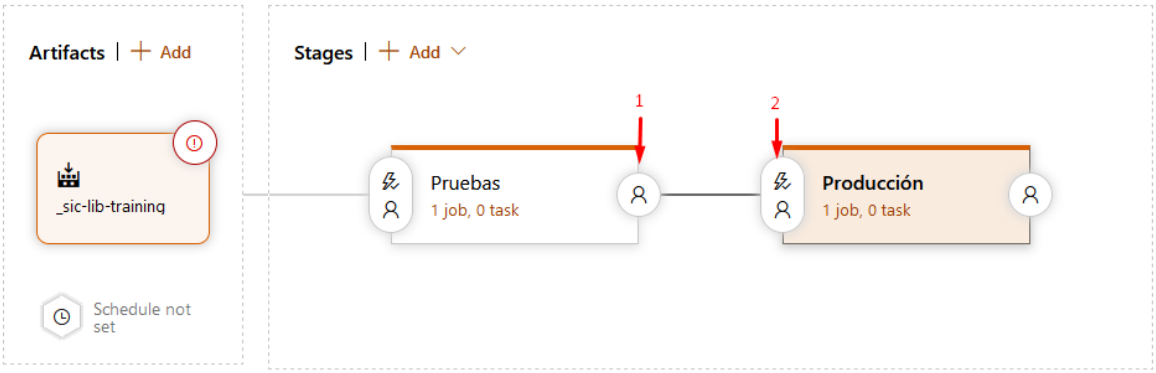


Imagen 40. Etapas de autorización

6.3.1. Aprobación funcional

Es el proceso que busca que los equipos funcionales o producto owners de los sistemas de información de él visto bueno de las funcionalidades desarrolladas y desplegadas en el ambiente de pruebas previo a un despliegue productivo, esta autorización se configura asi.

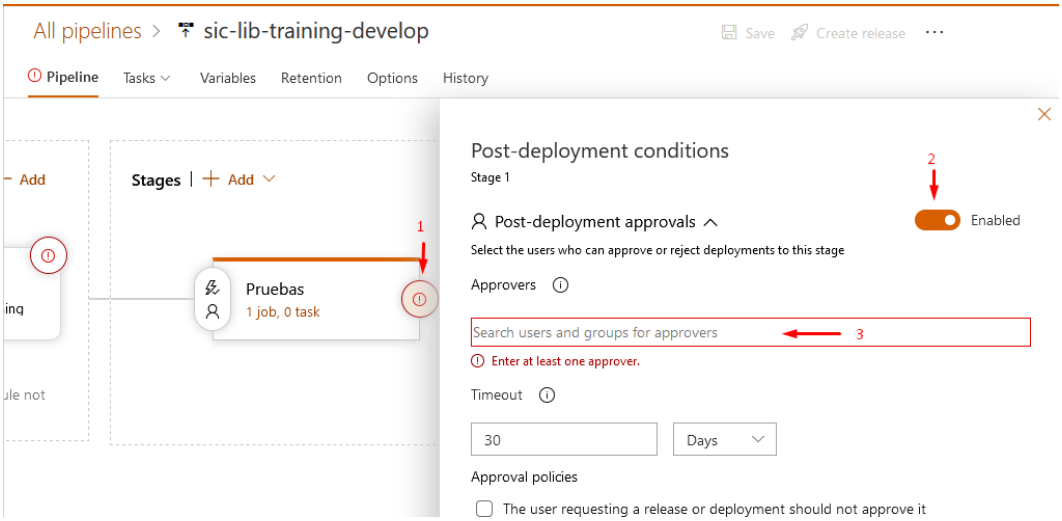


Imagen 40. Aprobación funcional post-deploy Pruebas

Se deben agregar los usuarios que tengan el rol o la autoridad de aprobar funcionalmente los despliegues productivos.

6.3.2. Aprobación comité de cambios

La aprobación por aparte del comité de cambios busca desde la perspectiva técnica garantizar el conocimiento de los despliegues y por lo tanto la operación de los cambios incluidos, para configurar dicha aprobación se deben seguir estos pasos.

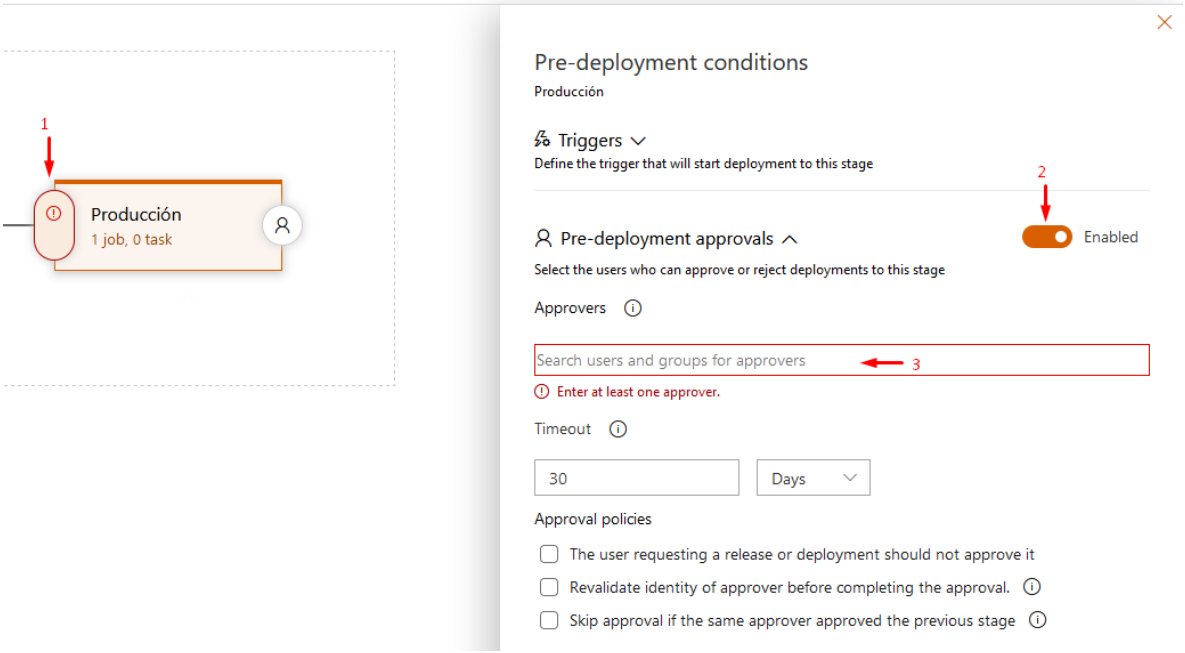


Imagen 41. Aprobación funcional pre-deploy Producción

Se deben agregar los usuarios que tengan el rol o la autoridad de aprobar técnicamente el despliegue, usualmente los miembros del comité de cambios.

7. REFERENCIAS

DevOps LifeCycle, tomado de https://co.pinterest.com/pin/858428378956341017/

Infografías de GitFlow, tomado de https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Información GitFlow, tomado de https://medium.com/snappler/c%C3%B3mo-ser-un-super-desarrollador-introducci%C3%B3n-a-git-flow-parte-1-7a3f7027d3fd

Información TDD, tomado de https://rubentejera.com/algoritmo-tdd/

Versiones de software, tomado de https://es.wikipedia.org/wiki/Versionado_de_software

Commit y Push, tomado de https://es.stackoverflow.com/questions/28344/cu%C3%A1l-es-la-diferencia-entre-commit-y-push-en-git

Guía detallada GIT, https://rogerdudler.github.io/git-guide/index.es.html

SOAPUI desde la consola de comandos. https://www.soapui.org/test-automation/running-from-command-line/functional-tests.html

Angular Testing. https://angular.io/guide/testing

PHPUnit, https://phpunit.de/getting-started/phpunit-9.html

PHPUnit Coverage, https://phpunit.readthedocs.io/es/latest/code-coverage-analysis.html#

Visual Studio Unit Testing, https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019

Junit, https://junit.org/junit5/

JaCoCO, https://www.jacoco.org/

Selenium, https://www.selenium.dev/documentation/en/getting_started/quick/#ide

Plugin TEE, https://azuredevopslabs.com/labs/vstsextend/eclipse/