INNOVACIÓN TECNOLÓGICA ST

WORKSHOP GIT FLOW

Flujo de Trabajo en Git y Paso entre Entornos

ATLANTIS

Transformando lo esencial

ÍNDICE

ATLANTIS

Definición Git Flow

ATLANTIS

- Modelo alternativo de creación de ramas en Git en el que se utilizan ramas de función y varias ramas principales.
- Siempre puedes confirmar cambios y crear una nueva rama de funciones cuando necesites interrumpir tu trabajo. Puedes volver a tu función en cualquier momento.
- Las ramas de corrección permiten realizar cambios urgentes. No tienes que preocuparte de fusionar accidentalmente nuevos desarrollos al mismo tiempo.
- Disminuyen los errores humanos en la mezcla de las ramas.
- Elimina la dependencia de funcionalidades al momento de entregar código para ser puesto en producción.

Definición ramas Git

ATLANTIS

Ramas principales

- **Main** Rama principal, alineada con versiones de **producción**.
- **Release** Rama estable, alineada con entorno de **pre-producción** para validación por parte de Product Owners / Clientes finales colaboradores
- **Test** Rama en desarrollo, alineada con entorno de **test**, para validación por parte del equipo de QA.
- **Develop** Rama principal de **desarrollo**, donde se irán uniendo los desarrollos finalizados. Para validación por parte del equipo de Desarrollo.

Ramas de Función

- **Feature/...** Ramas de función derivadas de Develop para tareas de nuevos desarrollos.
- **Bugfix/...** Ramas de función derivadas de Develop para tareas de corrección de errores en Desarrollo
- **Hotfix/...** Ramas de función derivadas de Main para tareas de corrección de errores críticos en Producción

Definición ramas Git

ATLANTIS

- Las ramas de función deberán nombrarse con el ID de la tarea correspondiente de Jira y opcionalmente una descripción corta.

  Ejemplos:
  - feature/**KAN-123**
  - bugfix/**TASK-345**-corrección_error
  - feature/**ATL-789**_descripción-tarea

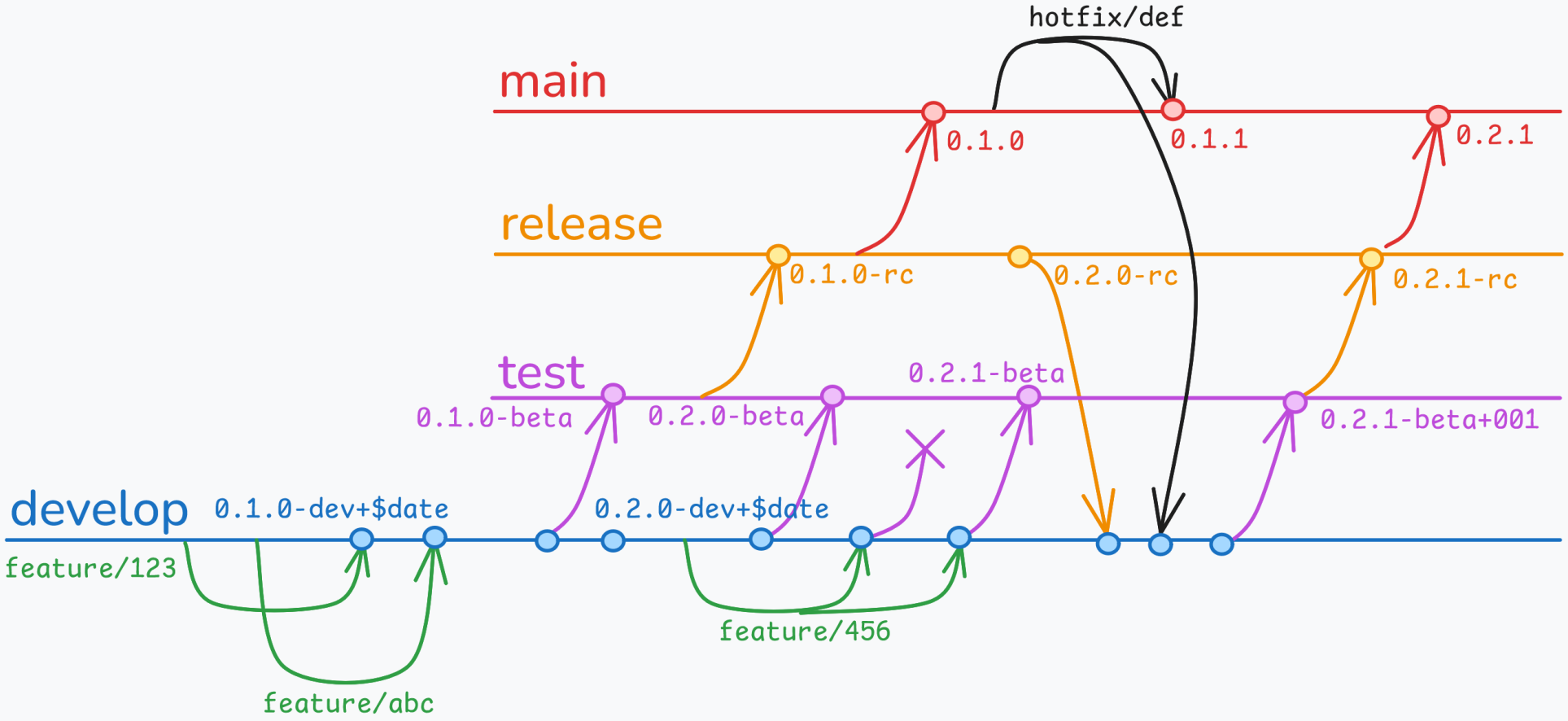
- Si hay más de una tarea de Jira que corresponda con el desarrollo a realizar, utilizar el ID de la épica.
  - En su defecto, usar el ID de la primera tarea a desarrollar.

- Los mensajes de los commits deberán comenzar con el ID de la tarea correspondiente de Jira y opcionalmente una descripción corta.

  Ejemplos:
  - **KAN-123** Implementación de nueva vista
  - **ATL-789** - conexión con db

ATLANTIS

SemVer: Major.Minor.Patch(-label)



Artefactos generados

ATLANTIS

Durante el flujo de CI/CD se generan distintos artefactos para el despliegue de la aplicación y sus dependencias:

- **PAQUETE DE LA APLICACIÓN**
  - JAR/WAR, NPM, NuGet ...
  - Agnóstico a sistema operativo, configuración y entornos
  - Almacenada en repositorio GitLab / Sonatype Nexus
- **IMAGEN DE CONTENEDOR**
  - En formato Docker
  - Con configuración de sistema operativo (RHEL 9/Alpine), agnóstico a entorno de despliegue
  - Almacenada en repositorio de imágenes GitLab / Harbor
- **PAQUETE PARA DESPLIEGUE**
  - En formato Helm Chart
  - Con configuración del entorno de despliegue
  - Almacenada en repositorio GitLab / Sonatype Nexus

Versionado y Etiquetado

SemVer se compone de tres valores principales:

- **MAJOR [X.0.0]**
  Subida de la versión Major cuando hay cambios no retro-compatibles, como cambios en la definición de la API o subida de la versión del framework.

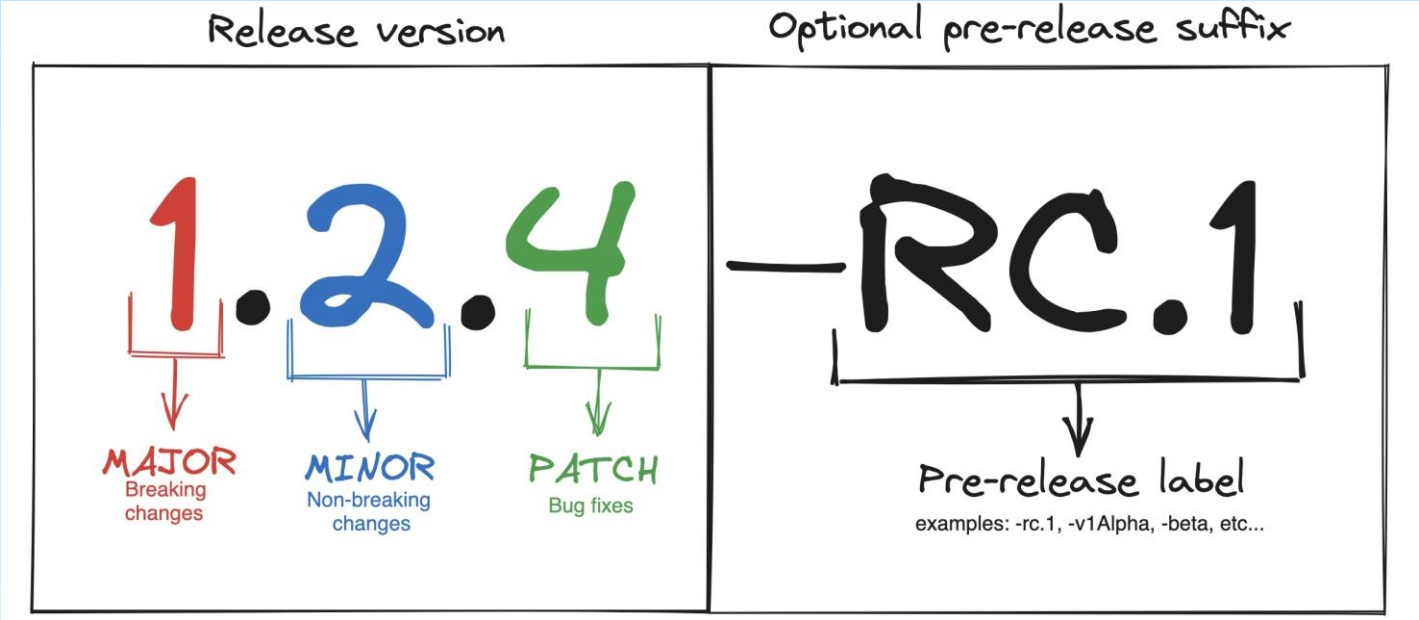
- **MINOR [0.X.0]**
  Subida de la versión Minor cuando hay nuevos desarrollos o implementaciones, o cambios retro-compatibles en la API o funciones marcadas como deprecadas.

  Se debe establecer a 0 cuando se sube la versión MAJOR.

- **PATCH [0.0.X]**
  Subida de la versión Patch cuando hay desarrollos que corrigen errores o comportamientos incorrectos introducidos en versiones anteriores.

  Se debe establecer a 0 cuando se sube la versión MINOR o MAJOR.

Versionado y Etiquetado



https://victorpierre.dev/blog/beginners-guide-semantic-versioning/

Versionado y Etiquetado

ATLANTIS

Cada artefacto generado tendrá su propia versión, que podrá coincidir o no con la versión de la aplicación:

- **PAQUETE DE LA APLICACIÓN**

  artifact-1.2.3-LABEL.(nuget|jar|war|npm...) notifications-1.0.1-develop.nupkg

- **IMAGEN DE CONTENEDOR (DOCKER)**

  repositorio/artifact:1.2.3-LABEL grupos/atlantis/back/notifications:1.0.1-develop

- **PAQUETE PARA DESPLIEGUE (HELM CHART)**

  Dos campos diferenciados: uno para identificar versión de la aplicación y otro para la versión del Helm.

  appVersion puede ser cualquier texto notifications-1.0.1-develop
  version debe estar en formato SemVer 1.0.1

¡version puede ser distinto a appVersion!
El Helm Chart puede haber sido modificado independientemente de la aplicación

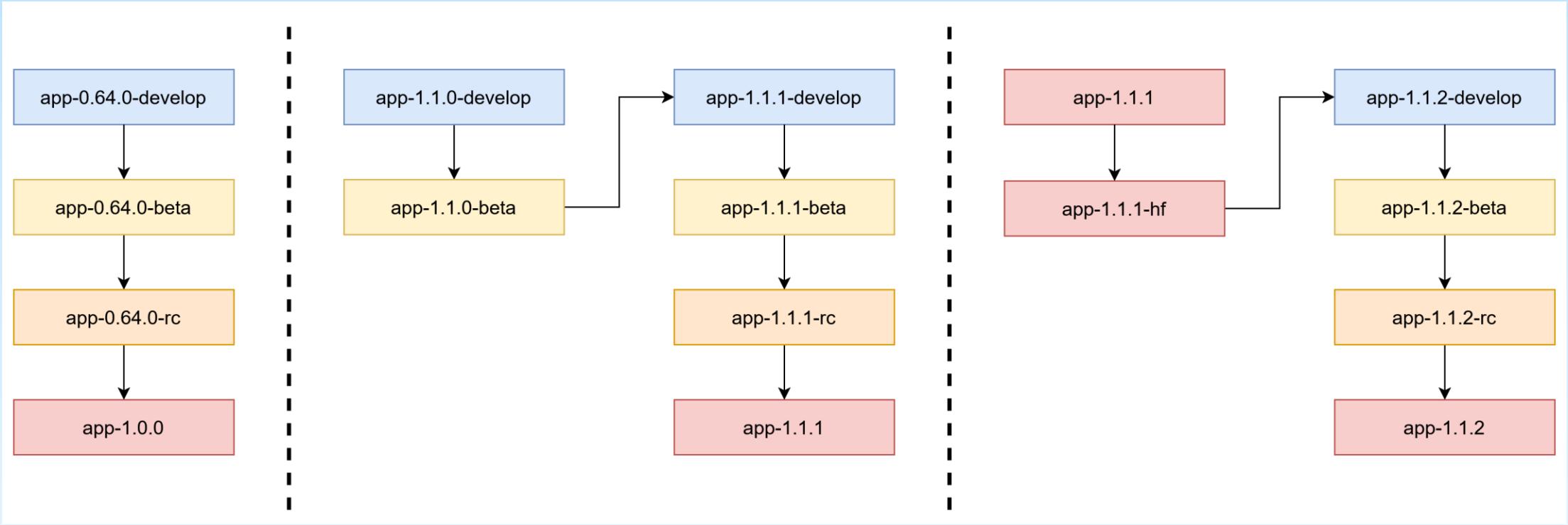
Versionado y Etiquetado

ATLANTIS

- En el paso a cada entorno se creará una etiqueta en el repositorio de Git para identificar rápidamente el punto del histórico del código desplegado.
- Estas etiquetas tendrán el número de versión junto con el sufijo correspondiente al entorno.
  - Desarrollo -develop
  - Test -beta
  - Pre-producción -rc
  - Producción Sin sufijo
- Coincidirán con la etiqueta de la imagen Docker de los registros de contenedores.

Esquema de Versionado

ATLANTIS



Reglas de los Repositorios

ATLANTIS

Se aplicarán las siguientes reglas en todos los repositorios de Back y Front del proyecto Atlantis:

- Las 4 ramas principales estarán protegidas debidamente según el entorno.
  - Esta protección implica que no se podrá hacer merge o subir código sin los permisos necesarios.

- Los merge requests a las ramas principales deberán cumplir con las reglas de aprobación y de seguridad.
  - Los escaneos de vulnerabilidades no deben tener vulnerabilidades altas o críticas.
  - Debe cumplirse con los niveles de calidad del código establecidos.
  - Debe haber aprobación manual por parte de las personas responsables.

- Se rechazarán las subidas de código si no se validan reglas como:
  - Email del usuario es válido y con cuenta en el sistema.
  - El nombre de la rama de función cumple con la nomenclatura.
  - El mensaje del commit comienza con un ID de tarea de Jira.
  - Se suben ficheros con extensiones prohibidas (exe, msi, deb, zip, tar.gz, etc.)
  - Se detecta algún patrón coincidente con información sensible como claves y tokens de API.

Reglas de los Repositorios

ATLANTIS

```
15:49 dvelasco ~/.../CICD/templates  arreglo-error  129
$ git commit -m 'Prueba commit' --allow-empty
[arreglo-error b080164] Prueba commit
15:51 dvelasco ~/.../CICD/templates  arreglo-error
$ git push
fatal: The current branch arreglo-error has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin arreglo-error

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

15:51 dvelasco ~/.../CICD/templates  arreglo-error  128
$ git push --set-upstream origin arreglo-error
Enter passphrase for key '/home/dvelasco/.ssh/id_ed25519':
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 198 bytes | 39.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
remote: GitLab: Commit message does not follow the pattern '^[A-Z][A-Z]+-[0-9]+.*'
To gitlab.com:grupost/atlantis/arquitectura/cicd/templates.git
 ! [remote rejected] arreglo-error -> arreglo-error (pre-receive hook declined)
error: failed to push some refs to 'gitlab.com:grupost/atlantis/arquitectura/cicd/templates.git'
```

```
16:07 dvelasco ~/.../CICD/templates  desarrollo ↑1 ↓4  1
$ git push  --set-upstream origin desarrollo
Enter passphrase for key '/home/dvelasco/.ssh/id_ed25519':
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 206 bytes | 34.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
remote: GitLab: Branch name 'desarrollo' does not follow the pattern '(feature|bugfix|hotfix)\/[A-Z][A-Z]+-\d+([-_].*)?'
To gitlab.com:grupost/atlantis/arquitectura/cicd/templates.git
 ! [remote rejected] desarrollo -> desarrollo (pre-receive hook declined)
error: failed to push some refs to 'gitlab.com:grupost/atlantis/arquitectura/cicd/templates.git'
16:08 dvelasco ~/.../CICD/templates  desarrollo ↑1 ↓4  1
$
```

Reglas de Aprobación

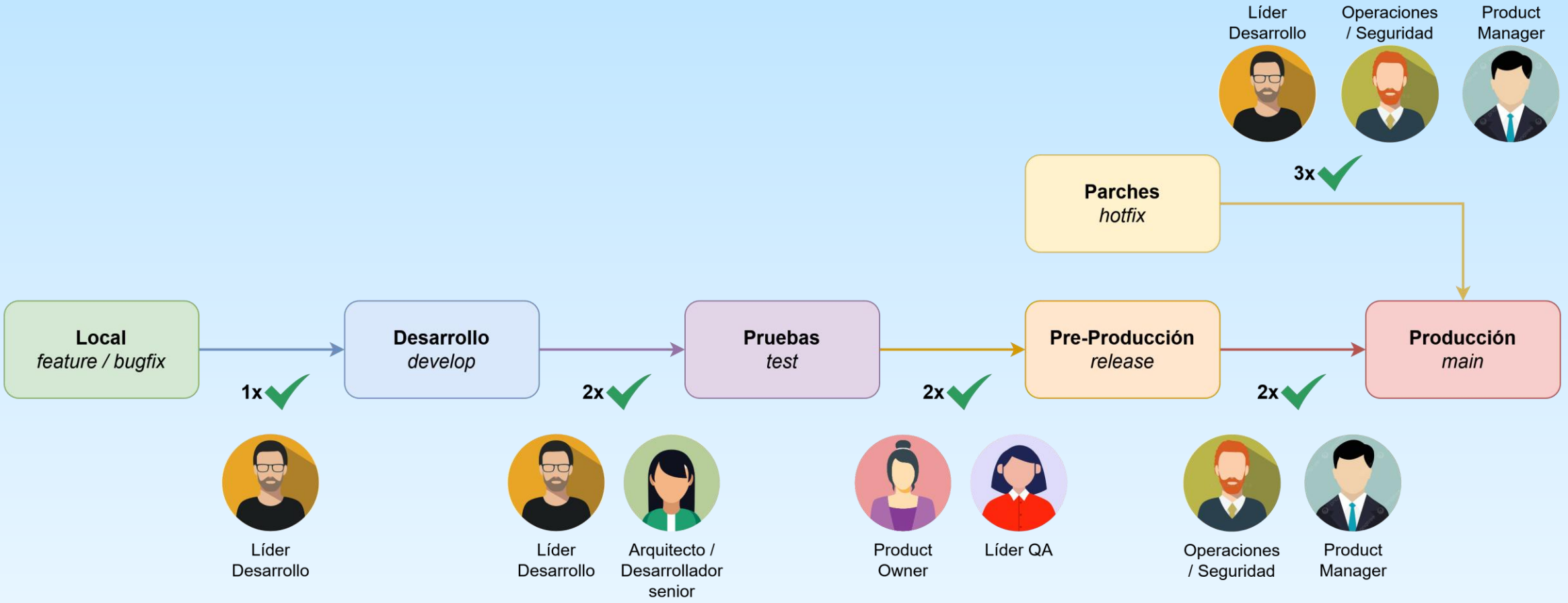
ATLANTIS

El paso entre entornos de los desarrollos deberá ser aprobado por los miembros autorizados correspondientes.

❖ Desarrollo	Líder Equipo Desarrollo	Marcos Anguís
❖ Test	Líder Equipo Desarrollo Arquitecto / Desarrollador senior	Marcos Anguís Eduardo Bugedo
❖ Pre-Producción	Líder Equipo QA Product Owner	Marcos Anguís Lorena Benito
❖ Producción	Product Manager Sistemas / Seguridad	Mariano López Ángel Molina / Manuel Pedrosa
❖ Hotfix Producción	Líder Equipo Desarrollo Product Manager Sistemas / Seguridad	Marcos Anguís Mariano López Ángel Molina / Manuel Pedrosa

Reglas de Aprobación

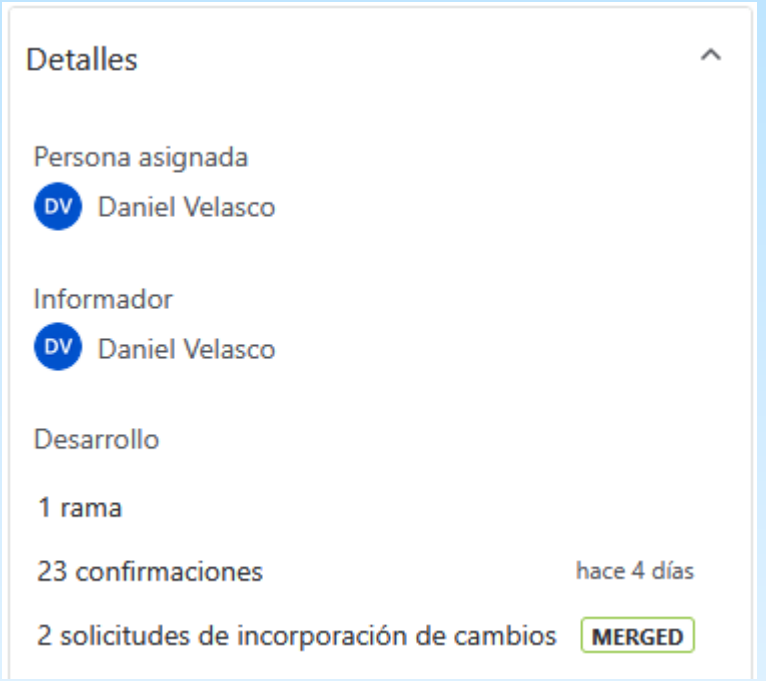
ATLANTIS



Integración con Jira

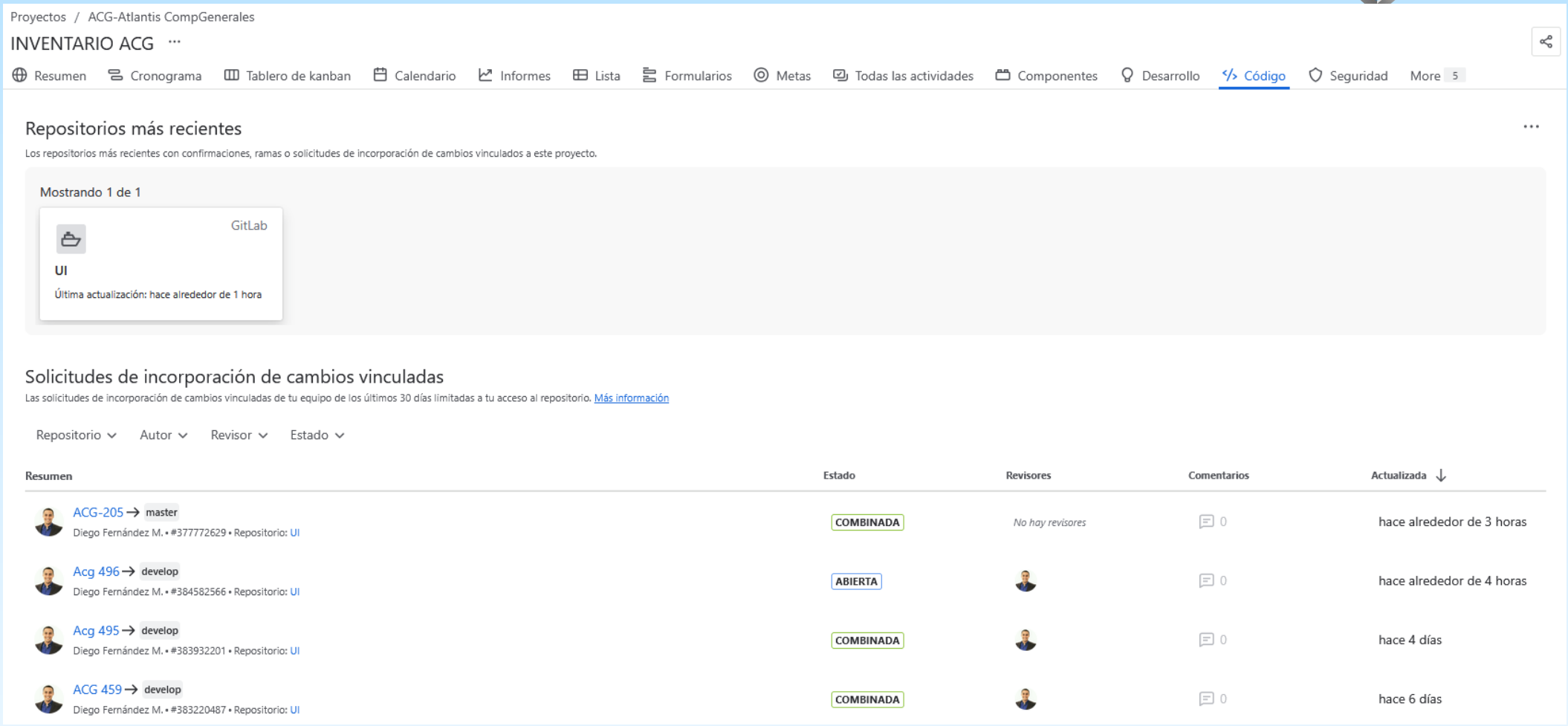
ATLANTIS

- La integración de GitLab con Jira ofrece información de las fusiones de código realizadas (Solicitudes de incorporación de cambios) y su estado.
- En los detalles de una tarea podremos ver el histórico de las confirmaciones (commits) enlazadas a la tarea y los autores.
- En el futuro también podremos ver desde Jira las versiones publicadas y tener un seguimiento de las mismas.
- También será posible disparar la creación de ramas asignadas a las tareas y las peticiones de fusión de código al pasar de una columna a otra
- Para que la integración funcione correctamente, tanto las ramas como los commits deben incluir el ID de la tarea de Jira.



Integración con Jira

ATLANTIS



Creación del repositorio de una nueva aplicación

ATLANTIS

- El equipo de Sistemas crea un nuevo repositorio en GitLab, creando las ramas principales y aplicando todas las reglas de seguridad y aprobación correspondientes.
- El equipo de DevOps añade el código del pipeline correspondiente al repositorio de código fuente, teniendo en cuenta el tipo de proyecto y frameworks a utilizar.
- El equipo de Desarrollo ya puede clonar el proyecto y comenzar con los desarrollos.

Creación del despliegue de una nueva aplicación

ATLANTIS

- El equipo de Desarrollo indicará al equipo de DevOps la configuración de la aplicación necesaria para su despliegue, por ejemplo, tipo de aplicación y versión del framework, carpetas y volúmenes necesarios, ficheros de configuración, variables de entorno, conexiones a servicios externos como bases de datos, puertos a exponer, etc.
- El equipo de DevOps crea el Dockerfile para la construcción de la imagen de la nueva aplicación.
- El equipo de DevOps crea un paquete de Helm para la nueva aplicación, modificando los valores correspondientes.
- El equipo de DevOps junto con Sistemas debe modificar el fichero de valores de cada entorno con estas configuraciones.

Construcción en Desarrollo

ATLANTIS

- El equipo de Desarrollo sube un cambio al código fuente de la aplicación.
- El pipeline de GitLab CI ejecuta los tests y escaneos de vulnerabilidades configurados.
- Si los tests pasan, el pipeline genera los artefactos (aplicación, imagen y chart) con un nuevo versionado y las etiquetas correspondientes.
- El equipo de desarrollo despliega la nueva versión de aplicación con la herramienta de despliegue continuo (ArgoCD) en el entorno de desarrollo para realizar las pruebas necesarias.
- Una vez verificado el desarrollo, el equipo realiza una solicitud de fusión (merge request) sobre la rama de Test para pasar la aplicación al entorno de Test / QA.

Paso a Test / QA

ATLANTIS

- La(s) persona(s) responsable(s) aprueba(n) la solicitud de fusión y el código pasa de la rama de develop a test.
- La herramienta de CI ejecuta el pipeline para analizar las vulnerabilidades y etiquetar los artefactos en su versión para test.
- La herramienta de CD despliega la aplicación en el entorno de test a través de un pipeline.
- El equipo de Test / QA realiza las pruebas pertinentes para validar los desarrollos realizados.
- Una vez verificada la aplicación, el equipo de desarrollo realiza una solicitud de fusión (merge request) para pasar la aplicación al entorno de pre-producción.

Paso a Pre-producción (Staging)

ATLANTIS

- La(s) persona(s) responsable(s) aprueba(n) la solicitud de fusión y el código pasa de la rama de test a release.
- La herramienta de CI ejecuta el pipeline para analizar las vulnerabilidades y etiquetar los artefactos en su versión para pre-producción.
- La herramienta de CD despliega la aplicación en el entorno de pre-producción a través de un pipeline.
- El equipo de producto realiza las pruebas pertinentes para validar los desarrollos realizados.
- Una vez verificada la aplicación, el equipo de desarrollo realiza una solicitud de fusión (merge request) para pasar la aplicación al entorno de producción.

Despliegue en Producción

ATLANTIS

- La(s) persona(s) responsable(s) aprueba(n) la solicitud de fusión y el código pasa de la rama de release a main.
- La herramienta de CI ejecuta el pipeline para analizar las vulnerabilidades y etiquetar los artefactos en su versión para producción.
- La herramienta de CI crea la aplicación con ArgoCD, obteniendo el fichero correspondiente del repositorio de Git y aplicándolo sobre el clúster.
- El equipo responsable de producción aplica manualmente los cambios de la nueva versión sobre el clúster de producción sincronizando ArgoCD.

Despliegue en Producción

ATLANTIS

- El despliegue de nuevas versiones se ejecuta en formato Rolling, actualizando uno a uno los pods de la aplicación para no tener pérdida de servicio.
- ArgoCD permite desplegar las versiones en formato blue/green automatizando el balanceo de carga entre las dos versiones, sin tener que realizar cambios en la infraestructura subyacente.
- También es posible realizar rollbacks desde ArgoCD para volver a versiones anteriores rápidamente.

Guías y Buenas Prácticas

ATLANTIS

- Git – la guía sencilla: rogerdudler.github.io/git-guide
- Git – branching learngitbranching.js.org
- Manteniendo el registro de cambios (changelog): keepachangelog.com
- SemVer 2: https://semver.org/lang/es/

ATLANTIS

Transformando lo esencial