Quality

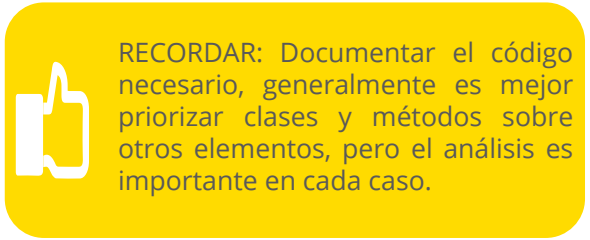
IT BOARDING

BOOTCAMP

Documentar el código: Documentación Idiomática

La documentación idiomática nos permite utilizar la sintaxis y expresividad de cada lenguaje para realizar comentarios explicativos sobre nuestro código, agregándole una capa de documentación de bajo costo y alta expresividad para quien necesita entenderlo.

En el caso de Java, este tipo de comentario solemos llamarlo JavaDoc y nos permite documentar casi todos los elementos del lenguaje, desde variables hasta clases y métodos.




```java
/**
* Hero is the main entity we'll be using to . . .
*
* Please see the {@link com.baeldung.javadoc.Person} class for true identity
* @author Captain America
*
*/
public class SuperHero extends Person {
    // fields and methods
}
```

Índice

IT BOARDING BOOTCAMP

// LA IMPORTANCIA DE DOCUMENTAR

IT BOARDING

BOOTCAMP

// ¡Documentar es una inversión!

“Documentar es aburrido”, “Documentar es costoso” son frases preferidas de muchos desarrolladores contemporáneos. Y en cierta manera tienen razón, un mal uso o abuso de la documentación puede convertirse más en un problema que en una solución.

Documentar con criterio y priorizando, sin dudas es una inversión que nos acerca a tener un **código más sustentable/mantenible**.

Documentar nos hace más **ágiles** y produce **desarrolladores felices**. Nos ayuda a **entendernos interna** y **externamente** y que el **onboarding** y la **colaboración** sea más **directa**

La clave es elegir qué, cuándo y cómo documentar. Maximizando el entendimiento y minimizando el costo.

Qué hacer y qué no hacer cuando documentamos

Aprovechar el repositorio es importante, un buen README es clave para un primer contacto con la aplicación.

Elegir los recursos justos y necesarios para explicar gráficamente mi aplicación.

Documentar funcionalmente lo justo y necesario pensando en mis usuarios, que necesitan y cómo representarlo.

Documentar el código utilizando buenas prácticas del lenguaje lo hace más expresivo.

Documentar la aplicación por fuera del repositorio. Dificulta la colaboración, mantenimiento, actualización y accesibilidad.

Sobredosis de diagramas puede confundir y generar un alto costo de mantenimiento.

Grandes y largos manuales de uso son costosos de crear y muy difícilmente sean usables para quienes los consumen.

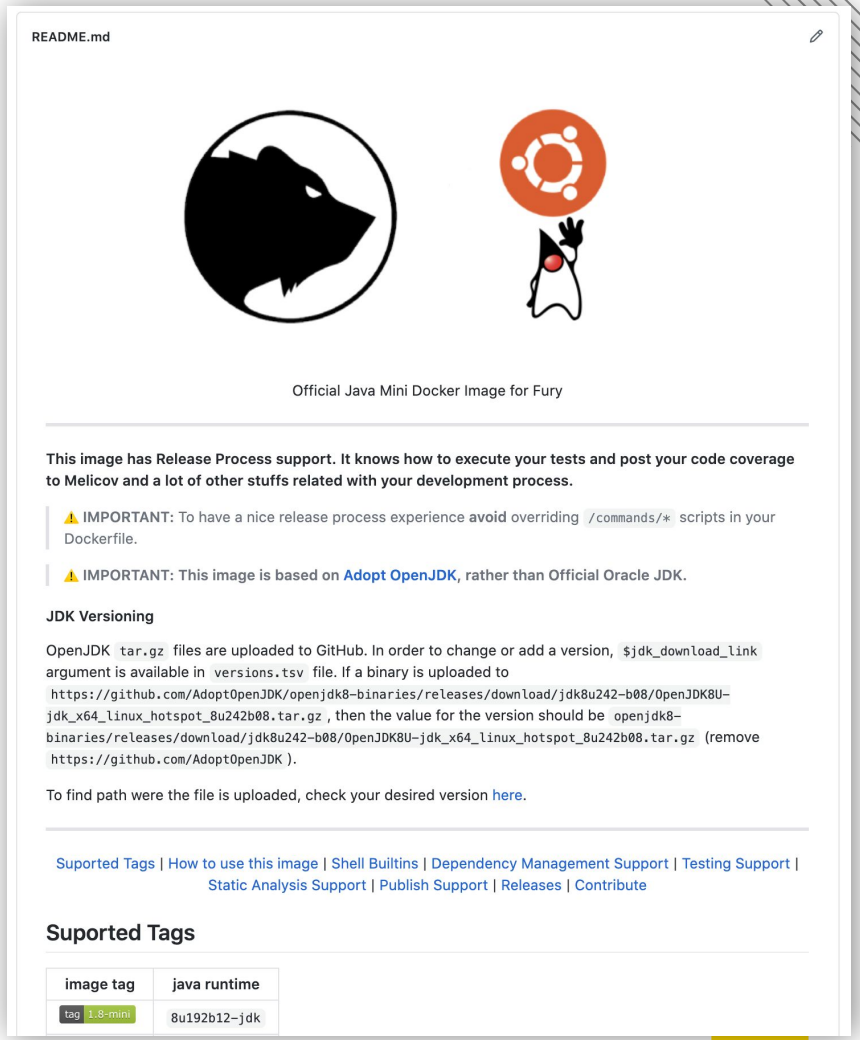
Documentar cada línea de código resulta ilegible, poco expresivo y difícil de mantener.

Documentar el repositorio: README

README.md es un archivo de texto situado en la mayoría de los repositorios de código que actúa como carta de presentación de dicho repositorio, conteniendo información útil e introductoria sobre lo que puedo o no encontrarme en dicho repositorio.

.md es la extensión del lenguaje de markup *Markdown*. Markdown es un lenguaje que nos permite de manera simple y ágil formatear texto utilizando un editor de texto plano.

Para más información sobre cómo crear un buen README, visitar el sitio:
https://www.makeareadme.com/


README.md

Official Java Mini Docker Image for Fury

This image has Release Process support. It knows how to execute your tests and post your code coverage to Melicov and a lot of other stuffs related with your development process.

> ⚠ **IMPORTANT:** To have a nice release process experience **avoid** overriding `/commands/*` scripts in your Dockerfile.

> ⚠ **IMPORTANT: This image is based on Adopt OpenJDK, rather than Official Oracle JDK.**

JDK Versioning

OpenJDK `tar.gz` files are uploaded to GitHub. In order to change or add a version, `$jdk_download_link` argument is available in `versions.tsv` file. If a binary is uploaded to `https://github.com/AdoptOpenJDK/openjdk8-binaries/releases/download/jdk8u242-b08/OpenJDK8U-jdk_x64_linux_hotspot_8u242b08.tar.gz` , then the value for the version should be `openjdk8-binaries/releases/download/jdk8u242-b08/OpenJDK8U-jdk_x64_linux_hotspot_8u242b08.tar.gz` (remove `https://github.com/AdoptOpenJDK`).

To find path were the file is uploaded, check your desired version here.

Suported Tags | How to use this image | Shell Builtins | Dependency Management Support | Testing Support | Static Analysis Support | Publish Support | Releases | Contribute

Suported Tags

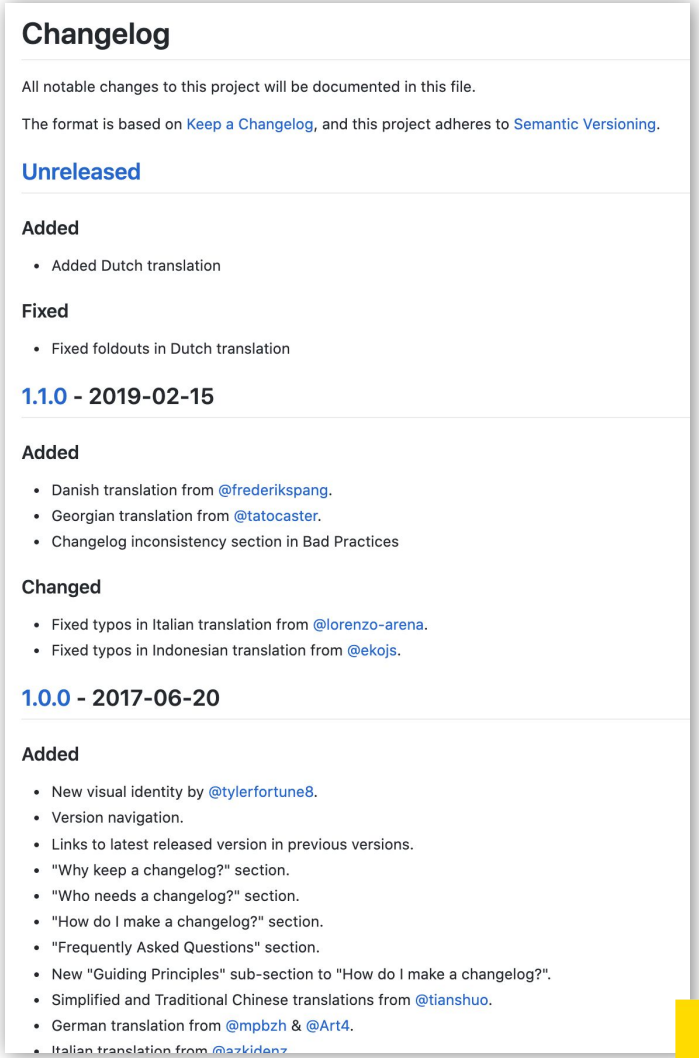
image tag	java runtime
tag 1.8-mini	`8u192b12-jdk`

Documentar el repositorio: CHANGELOG

CHANGELOG.md es un archivo de texto situado en la mayoría de los repositorios de código que nos permite conocer la información más relevante de cada una de nuestras releases, que se agregó, que se modificó, que se eliminó, que está deprecado y demás.

El concepto de CHANGELOG está íntimamente ligado a SEMVER, dado que el formato en el nombre de la versión nos permite entender si los cambios son o no retrocompatibles, si se trata de un fix o un feature, etc.

Para más información sobre cómo crear un buen CHANGELOG, visitar el sitio:
https://keepachangelog.com/



Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

Unreleased

Added

- Added Dutch translation

Fixed

- Fixed foldouts in Dutch translation

1.1.0 - 2019-02-15

Added

- Danish translation from @frederikspang.
- Georgian translation from @tatocaster.
- Changelog inconsistency section in Bad Practices

Changed

- Fixed typos in Italian translation from @lorenzo-arena.
- Fixed typos in Indonesian translation from @ekojs.

1.0.0 - 2017-06-20

Added

- New visual identity by @tylerfortune8.
- Version navigation.
- Links to latest released version in previous versions.
- "Why keep a changelog?" section.
- "Who needs a changelog?" section.
- "How do I make a changelog?" section.
- "Frequently Asked Questions" section.
- New "Guiding Principles" sub-section to "How do I make a changelog?".
- Simplified and Traditional Chinese translations from @tianshuo.
- German translation from @mpbzh & @Art4.
- Italian translation from @azkidenz

Documentar los cambios: Commit messages

El commit message es una descripción que dejamos sobre los cambios que estamos haciendo sobre el código, cada vez que aplicamos los mismos sobre el repositorio.

No existe una única convención sobre la manera de escribirlo, pero sí es importante aprovecharlo para agregar el contexto que ayude a quien esté leyendo el código a comprender el motivo del cambio. De esa manera facilitamos la colaboración y el debugging.

RECORDAR: No es necesario hacer esto para TODOS los commits. Usar siempre el criterio y analizar cada caso en particular

```
commit eb0b56b19017ab5c16c745e6da39c53126924ed6
Author: Pieter Wuille <pieter.wuille@gmail.com>
Date:   Fri Aug 1 22:57:55 2014 +0200

    Simplify serialize.h's exception handling

    Remove the 'state' and 'exceptmask' from serialize.h's stream
    implementations, as well as related methods.

    As exceptmask always included 'failbit', and setstate was always
    called with bits = failbit, all it did was immediately raise an
    exception. Get rid of those variables, and replace the setstate
    with direct exception throwing (which also removes some dead
    code).

    As a result, good() is never reached after a failure (there are
    only 2 calls, one of which is in tests), and can just be replaced
    by !eof().

    fail(), clear(n) and exceptions() are just never called. Delete
    them.
```

// ¡HANDS ON TIME!

IT BOARDING

BOOTCAMP

// INTRO A DIAGRAMAS

IT BOARDING

BOOTCAMP

// Diagrama

Es un lenguaje para el modelado de software, entre otras cosas. Que utiliza recursos gráficos para describir componentes, métodos y procesos.

Nos ayuda a **entender** el big picture de nuestro sistema de una manera **ágil** y **visual**, sin necesidad de conocer en profundidad la implementación de bajo nivel.

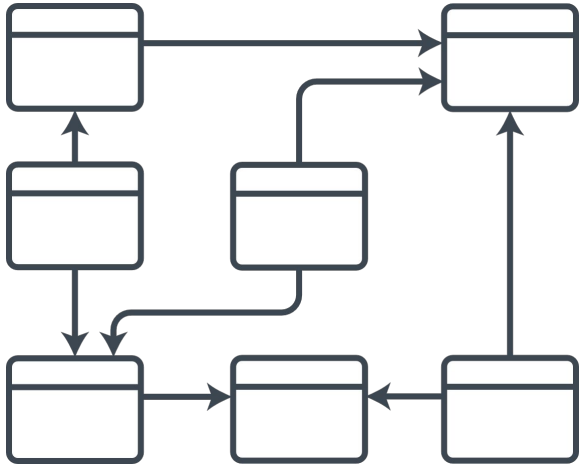
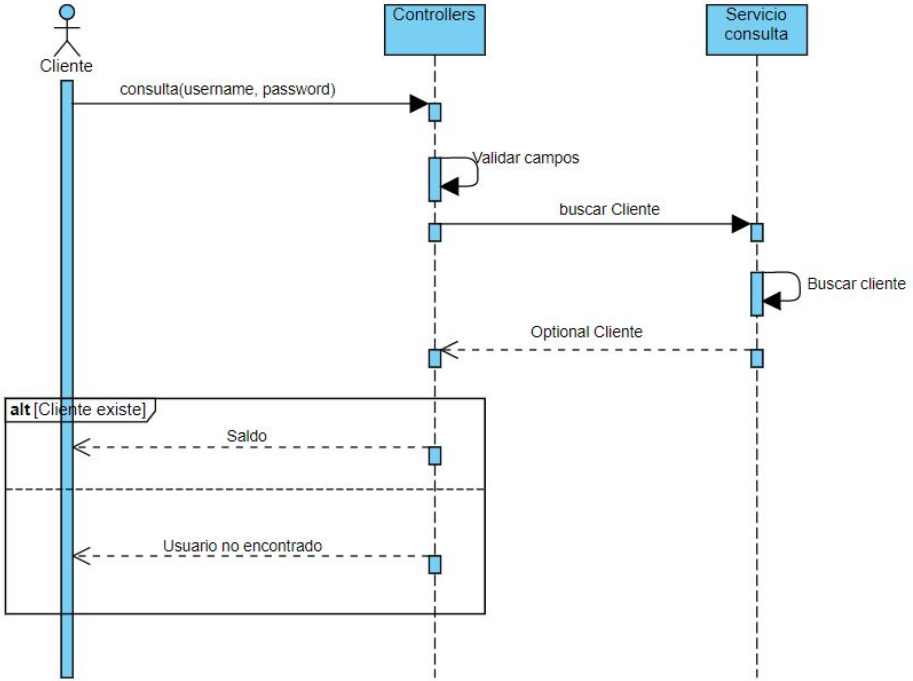


Diagrama de secuencia

Es un diagrama que representa los eventos en orden cronológico, contenidos en líneas de vida, mostrando la interacción entre un conjunto de objetos, a través de mensajes.

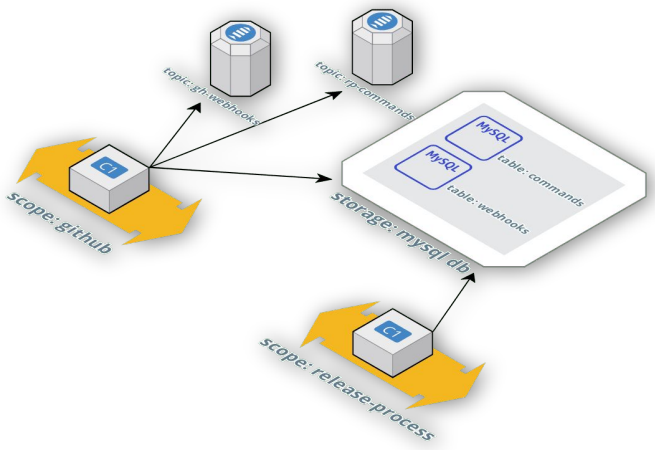


Tool online https://online.visual-paradigm.com/drive/#diagramlist:proj=0&new=SequenceDiagram

Diagrama de arquitectura

Un diagrama de arquitectura es la primer herramienta de diseño que tenemos disponible para empezar tanto a crear una nueva aplicación o colaborar en una ya existente.

Es un diagrama sumamente importante porque nos ayuda a tener una visión global de la estructura, posibles puntos de falla, performance, vulnerabilidades o puntos propensos a ataques cibernéticos.



// ¡HANDS ON TIME!

IT BOARDING

BOOTCAMP

// DOCUMENTATION SERVICE

IT BOARDING

BOOTCAMP

// Documentar es ágil

El servicio de documentación de Fury es un servicio desarrollado in-house que nos permite documentar nuestras aplicaciones de manera ágil, versionada y dentro del repositorio. Aprovechando el resto de los servicios de Fury como por ejemplo Release Process.

Está basado en dos pilares básicos:

Agilizar la gestión

Una de las cosas más costosas cuando hacemos ingeniería de software es la gestión de la documentación, tanto escribirla como mantenerla o publicarla y distribuirla. El servicio toma todos estos puntos y los lleva al punto máximo de agilidad.

Maximizar la trazabilidad

Cuando hablamos de documentación es clave la trazabilidad, entender qué cosa estamos documentando en cada release y cuando lo disponibilizamos es muy importante para que no se generen fugas o problemas de comunicación.

Documentación funcional

La documentación funcional nos permite realizar una explicación en lenguaje natural sobre las características de nuestra aplicación.

Este tipo de documentación es muy útil cuando queremos brindar un manual de uso, buenas prácticas o troubleshooting a quienes utilizan normalmente nuestra aplicación.

Por ejemplo la documentación funcional de Fury fue creada utilizando su propio servicio de documentación.

Para este tipo de documentación utilizamos docsify, que nos permite crear una documentación visualmente muy atractiva escribiendo únicamente archivos de texto plano en formato markdown (como los README o el CHANGELOG)

Documentación de interfaz REST

La documentación de interfaz nos permite documentar los contratos de nuestras APIs REST.

Este tipo de documentación es muy usada cuando queremos exponer información sobre qué aceptamos como requests y que retornamos como response en cada una de nuestras APIs.

Esto es extremadamente útil cuando necesitamos que nuestra API sea consumida por diversos clientes, para que ellos puedan adaptar sus aplicaciones a tiempo.

Para este tipo de documentación utilizamos Swagger, que nos permite crear una documentación visualmente muy atractiva escribiendo únicamente archivos YAML en el repositorio o mismo utilizando librerías utilitarias.

```yaml
paths:
  /allowlist:
    post:
      consumes:
      - application/json
      description: Create an allowed configuration for a given repository
      parameters:
      - description: Authorization Header
        in: header
        name: x-auth-token
        required: true
        type: string
      - description: Allowed configuration
        in: body
        name: allowedConfiguration
        required: true
        schema:
          $ref: '#/definitions/allowlist.AllowedConfigurationAPIPayload'
      responses:
        "201":
          description: Created
          schema:
            $ref: '#/definitions/gin.H'
        "400":
          description: Bad Request
          schema:
            $ref: '#/definitions/errorx.Err'
        "401":
          description: Unauthorized
          schema:
            $ref: '#/definitions/errorx.Err'
        "404":
          description: Not Found
          schema:
            $ref: '#/definitions/errorx.Err'
        "500":
          description: Internal Server Error
          schema:
            $ref: '#/definitions/errorx.Err'
      summary: Create an allowed configuration
      tags:
      - Allowlist
  /allowlist/{id}:
    delete:
      description: Delete an allowed configuration for a given repository
      parameters:
      - description: Authorization Header
        in: header
        name: x-auth-token
        required: true
        type: string
      - description: Configuration ID
        in: path
```

Allowlist

POST `/allowlist` Create an allowed configuration

Create an allowed configuration for a given repository

Parameters Try it out

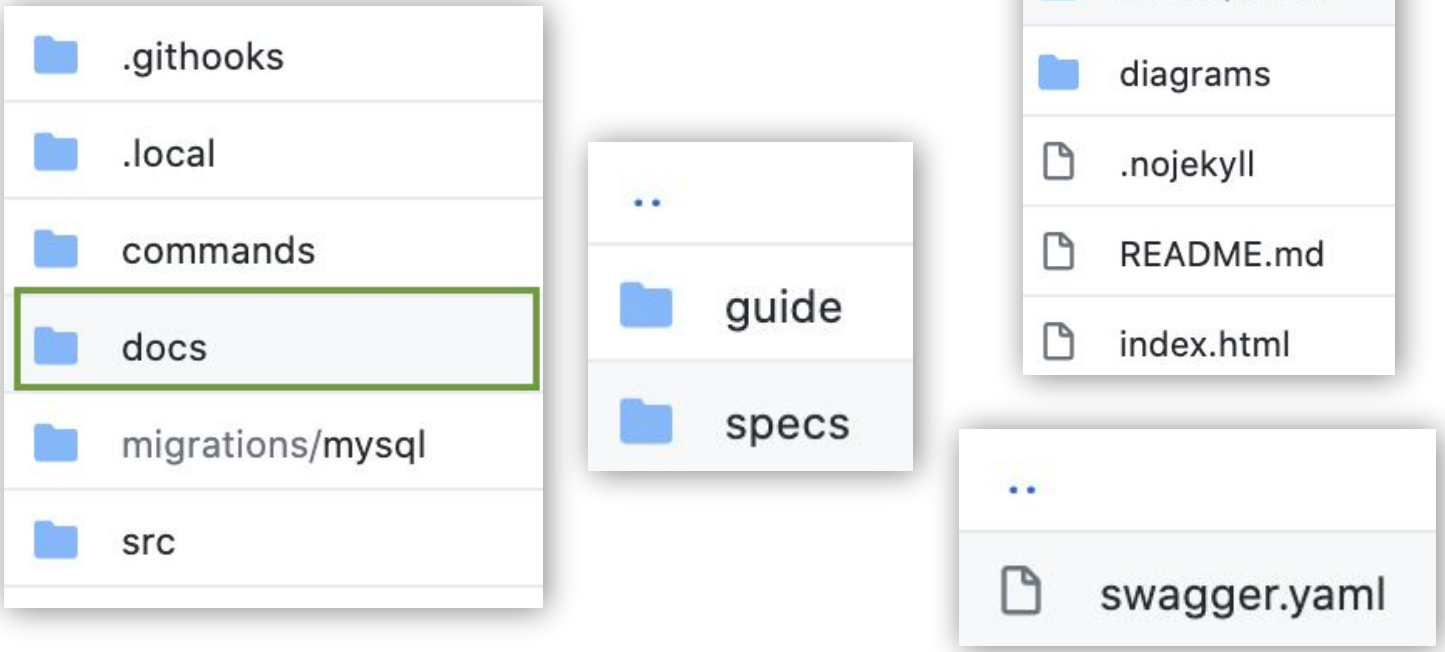
Name	Description
x-auth-token * required string *(header)*	Authorization Header
allowedConfiguration * required *(body)*	Allowed configuration Example Value | Model `{ "configuration_id": "fury_rp-configs-api", "dev_image": "go:1.15-mini", "expires_at": "2021-01-02T00:00:00Z", "runtime_image": "go:runtime-mini" }` Parameter content type: application/json

Responses Response content type: application/json

Code	Description
201	Created

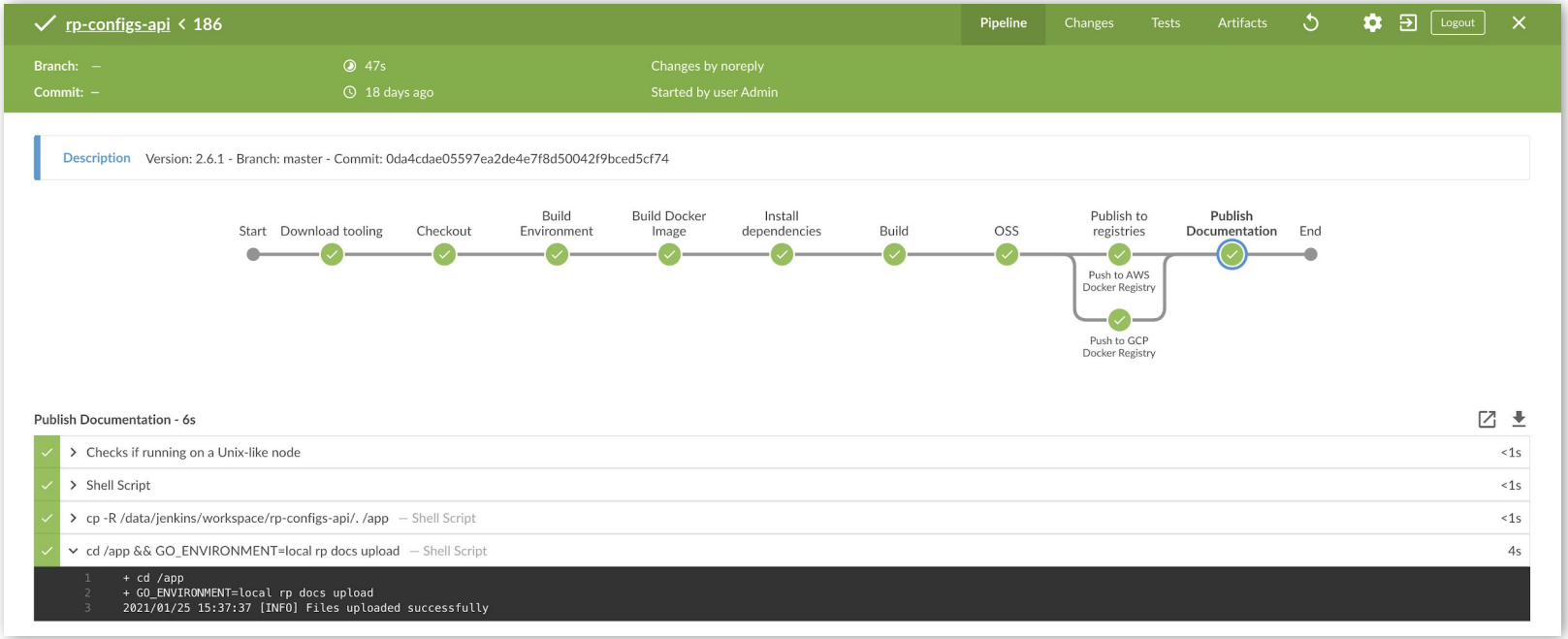
Feature: Estructura de directorio

El servicio se encuentra totalmente integrado con el repositorio y para que su gestión sea ágil y entendible utiliza como convención una estructura de directorio determinada.



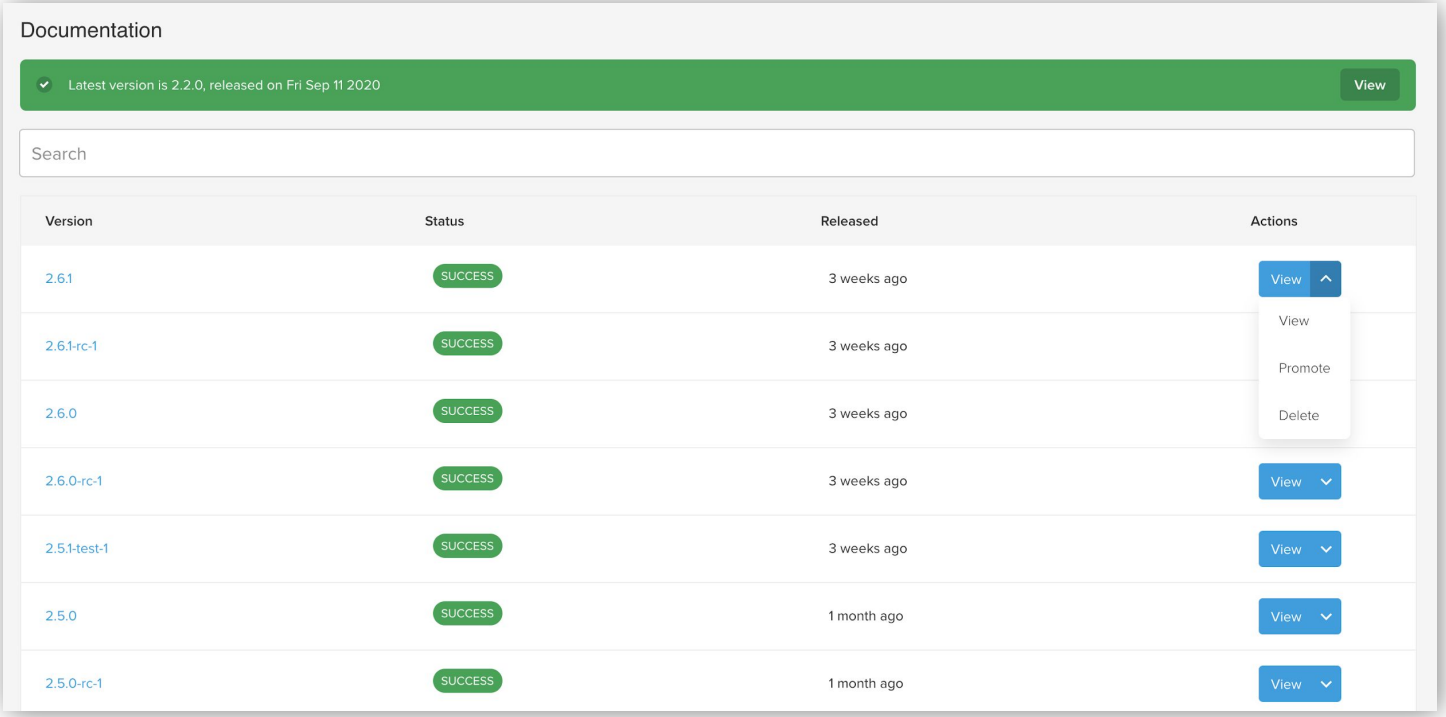
Feature: Proceso de subida

Cada vez que se crea una versión en una aplicación con la estructura de directorio mencionada anteriormente, se dispara automáticamente el proceso de upload de dicha documentación.



Feature: Proceso de publicación/gestión

La gestión es íntegramente realizada a través de Fury en la sección **“Servicios -> Documentation”**



// ¡HANDS ON TIME!

IT BOARDING

BOOTCAMP

// ¡DEMO TIME!

IT BOARDING

BOOTCAMP

// ¡GAME TIME!

IT BOARDING

BOOTCAMP

Los invitamos a completar la siguiente encuesta sobre el módulo QUALITY del Bootcamp.
¡Es muy muy importante para nosotros contar con su feedback!
Solamente les tomará unos minutos completarla :)

Link a la encuesta

Gracias.

IT BOARDING

BOOTCAMP