**Documento Lineamientos Técnicos APIs de Servicios**

**Elaborado para Coordinador Eléctrico Nacional**



Diciembre 2021

Tabla de Contenido

[**Prefacio**](#_1z6dsh353lkn) **5**

[Confidencialidad, derechos de autor y extención de responsabilidad](#_a5yrox6sx5iu) 5

[Acerca de este documento](#_xy1y0go8efu9) 5

[Contactos del equipo de Red Hat para este proyecto](#_4qhibqapl10e) 5

[**Introducción**](#_9qu8dsqe94ki) **6**

[**Alcance**](#_7v7gloxze8n1) **6**

[Alcance gestión de versiones](#_6lsckz2w8cz5) 6

[Alcance estándar de desarrollo](#_pj5ticptbhov) 6

[**Gestión de versiones**](#_84rd8beujtsu) **6**

[Flujo de ejecución general](#_euvlq0tzpum5) 7

[Flujo de ejecución de error](#_mn8iuiajasr9) 7

[Merge Request Desarrollo a QA](#_vo0xbydoqs2l) 7

[Merge Request QA a Producción](#_gwhtho5y4fwe) 9

[Diagramas](#_qm418puftbmw) 10

[Estructura de Repositorio](#_k2gv9oilnkx5) 13

[Contenido de Repositorios](#_ey3p3v2727sk) 14

[Documentación](#_iks7nylqdhxv) 15

[Acceso a Git mediante consola](#_q1n3m8ujear0) 15

[Numeración de Versiones](#_uw21f6y8rxn5) 16

[Criterio para modificación de versiones](#_lgh0uyxw1h12) 16

[Flujo de Ramificación Git](#_74v9lelqr7ai) 17

[Gestión de ramas sistemas en producción](#_ee2kmm8xrk11) 17

[Gestión de ramas nuevas](#_dbbf8hl3gxrn) 19

[Gestión de ramas evolutivos](#_tva01qf7dl4e) 21

[**Especificaciones de OpenAPI**](#_qtf13aeq5bbf) **22**

[Fases del ciclo de vida de la OpenAPI](#_jiq3dfgyq5it) 22

[Estrategias de migración de consumidores](#_91088jfy3gli) 23

[Formato](#_ggp6bu3ce0i) 23

[Tipos de Datos](#_j6b2gwrzhl60) 24

[Campos fijos](#_pyt46kee9jnz) 26

[Objeto de rutas](#_r6qx8moa3g16) 27

[Campos Estandarizados](#_976vqhozdu47) 27

[Coincidencia según plantillas de rutas](#_37xak11lgp99) 28

[Objeto de rutas de Ejemplo](#_rn5h9cwm4m9) 29

[Objeto de elemento de ruta](#_xj4llzni95hl) 30

[Campos fijos](#_y1v7q26whlkj) 30

[Objetos de flujos OAuth](#_wxz2hvtn9qvu) 31

[Campos fijos](#_32slujsx15af) 32

[Objeto de flujo OAuth](#_8aefknw0x5r1) 32

[Campos fijos](#_l1h7rk5zk1sn) 32

[Referencia relativa en URLs](#_nem4g2pyhu5g) 34

[**Especificaciones de API REST**](#_cbnccuz15fjd) **34**

[Motivaciones](#_cradwuhp7kl) 34

[Restricciones](#_e9wv7iy9wml3) 35

[Los 4 principios de REST](#_cnffxpz0gmsj) 36

[REST utiliza los métodos HTTP de manera explícita](#_ncvoqwlitrh2) 36

[REST no mantiene estados](#_mbs9cjhoyy8u) 36

[Responsabilidad del servidor](#_ebrh47fn20yf) 37

[Responsabilidades del cliente de la aplicación](#_tl8o1ko0rom) 37

[REST expone URIs con forma de directorios](#_l05j17b9cbdy) 38

[REST transfiere XML, JSON, o ambos](#_24613iv3i3p1) 38

[Reglas de diseño](#_e3mgkh4svmk6) 39

[Reglas generales](#_jiyuvhwmlfk9) 39

[Reglas de nombrado de URIs y dominio](#_vh5rnicm5m2v) 40

[Reglas de métodos HTTP](#_ria43429lexy) 42

[Reglas relativas a la respuesta HTTP](#_uwa4g6aaq8re) 43

[Reglas relativas al Header HTTP](#_dr4cihlvm215) 47

[**Tipos de fecha y moneda**](#_3sgbforhuv2l) **48**

[Monedas](#_rm0atauv2ke1) 48

[Fechas](#_fwi8nduvyh7t) 48

[**Especificación de configuración Openshift**](#_m9vvngewf0gw) **49**

[Configuración de Scaling](#_khlgtsmzs9x0) 49

[Configuración recomendada para Scaling](#_11fkz8vkbr2x) 49

[Configuraciones opcionales para Scaling](#_z47yumvqdxs3) 52

[Horizontal Pod Autoscalers](#_nr89fwingly8) 52

[Vertical Pod Autoscalers](#_xnhk97m1pmpq) 53

[**Gobernabilidad y Categorización de APIs**](#_da7qy3iapd5x) **55**

[Despliegues de componentes](#_gesidihpcvlf) 55

[Tabla de herramientas y framework](#_p0a2spvton08) 56

[Lenguaje y framework de desarrollo por defecto del COORDINADOR](#_ir4epvg07xhu) 56

[Mnemotecnia de diseño](#_as6nlni3mim) 58

[Nomenclatura de datos](#_johz46isxxwd) 58

[Nomenclatura para desarrollo web](#_rwbu7eex3ckh) 60

[Nomenclatura para desarrollo de microservicios](#_1zsscej41h63) 60

[Desarrollo Java Spring Boot y componentes core](#_o9qk4fsqwuox) 61

[Documentación de diseño](#_dncrnke6mz4g) 62

[Descripción del entorno tecnológico](#_e8xsb5hu75qa) 62

[Definición de Arquitectura del Sistema](#_3lmjpl6ndz58) 62

[Diseño del Modelo de Clases del Sistema](#_dq3ws7a0o1b6) 62

[Modelamiento de datos](#_t7xz5a8xx6n7) 62

[Definición de la Interfaz de Servicios del Sistema:](#_qacd4w22023b) 62

[Revisión y aprobación](#_uo6r6bevtfic) 63

[Ambientes cloud / on-Premise](#_ozllqmsdxyi0) 63

[Componentes cloud](#_1wztm9tk9n23) 63

[Gestión de código fuente](#_9rqyqw74k5yy) 64

[Indicaciones adicionales](#_q2l67yc28dku) 64

[**Consideraciones generales de desarrollo de software**](#_2lbc0trgr89u) **66**

[**Estándares de Interoperabilidad**](#_dfnjbq68jfnr) **68**

[Principios SOLID](#_54o6z7s5vu9f) 68

[Arquitectura de Referencia](#_6byxrex4zzbt) 68

[Consideraciones generales de integración](#_jpclafcrtvpl) 69

[Estándar CIM](#_63ahj6dm9adn) 70

[Modelo de Integración](#_oa6s5yjm7ju9) 70

[Protocolos SOAP-REST](#_9pd2kgukojbv) 71

[Procesos (ETL)](#_xrvprtupnrz8) 72

[Diseño](#_tl1qlkdo1kgj) 73

[Orquestación de Servicio](#_40ip4wd1bo70) 73

[Microservicios orquestados:](#_mokya2mbqcc4) 74

[Orquestar Microservicios desacoplados:](#_sq51nm2qdapj) 74

[Herramientas de Interoperabilidad](#_k9s3ypc4rdss) 74

[**DevSecOps**](#_ceyn35nh5p3a) **75**

[Integración Continua:](#_e51tqvhi3m3v) 75

[Revisión de Código:](#_d8bb2xsd5f4) 75

[**BPM**](#_7vna7i9jrll) **75**

[Entregables BPM](#_jdauh205elo) 76

[**Estándares para el uso de sistemas y aplicaciones**](#_n7v4re2yr9d2) **76**

[**Consideraciones generales para implementación y/o configuración de plataformas propietarias**](#_ohg1b916asp1) **77**

[**Utilización de repositorios Nexus**](#_1il04nmypcw2) **78**

[**Metodologías de Cifrado**](#_7yxqn3i5q9wj) **79**

[Ruta Edge](#_jaw6vqbtqf21) 79

[Prerrequisitos](#_xzgew4yh1znn) 79

[Procedimiento](#_hkbjngapbujz) 79

[**Ejemplo de Diseños**](#_nk18bbwkp78t) **81**

[Diagrama de Componentes](#_cze3rasvz75u) 81

[Diagrama de Clases](#_15goh5hu83ol) 82

[Diagrama de Secuencia](#_myelf74a4sn5) 83

[Diagrama de Despliegue](#_2dwthsv958n0) 84

# Prefacio

## Confidencialidad, derechos de autor y extención de responsabilidad

Se trata de un documento orientado al cliente entre Red Hat, Inc. y Coordinador Eléctrico Nacional

Copyright © 2021 Red Hat, Inc. Todos los derechos reservados. Ninguna parte del trabajo cubierto por los derechos de autor puede ser reproducida o utilizada de ninguna forma ni por ningún medio - gráfico, electrónico o mecánico, incluyendo fotocopiado, grabación o sistemas de almacenamiento y recuperación de información - sin el permiso por escrito de Red Hat, excepto cuando sea necesario para compartir esta información, tal y como se ha proporcionado a las partes confidenciales mencionadas anteriormente.

## Acerca de este documento

Este documento es el resultado de sesiones entre Red Hat y el Coordinador Eléctrico Nacional, donde se exponen las condiciones del proyecto desde el punto de vista de la migración y modernización de servicios. Este documento está dirigido para quienes deban comprender el contexto de este proyecto y que además les permita contar con una guía de migración y de modernización de servicios.

La versión de este documento referencia el alcance del proyecto el cual fue definido por los equipos de trabajo, tanto Red Hat como CEN (Coordinador Eléctrico Nacional) y de este modo generar una guía de trabajo para los equipos de consultoría.

## Contactos del equipo de Red Hat para este proyecto

Para obtener información adicional sobre la contenida en este documento, puede contactar con uno de los siguientes miembros del equipo de Red Hat:

| **Name** | **Title** | **Number** | **Email** |
| --- | --- | --- | --- |
| Cristian Quezada | Senior Consultant | +56 9 6340 9153 | cquezada@redhat.com |
| Hugo Cerda | Consultant | +56 9 5321 6795 | hcerdaja@redhat.com |
| Nathalia Patiño | Engagement Lead | +56 9 7518 1603 | npatinom@redhat.com |
| Daniele Stefenello | Project Manager | +56 9 7705 8527 | dstefene@redhat.com |
| Raul Escobar | Consultant | +56 9 4903 6593 |  |

# Introducción

Este documento describe las reglas de diseño y mejores prácticas sobre hacer uso del API REST con el objetivo de definir una forma homogénea de utilizarla, siguiendo lineamientos que ayuden a la reutilización de servicios, hacerlos auto descriptivos y mejorar el mantenimiento de los mismos.

# Alcance

## Alcance gestión de versiones

El documento abordará la gestión de versiones de forma que aplique para todos los sistemas y aplicaciones que se implementarán en las dependencias de Coordinador Eléctrico Nacional, tanto para desarrollo de nuevos proyectos, evolutivos y correctivos. Este documento es parte integral de los requerimientos de ingeniería.

## Alcance estándar de desarrollo

El documento abordará los estándares de desarrollo a aplicar para todo los sistemas y aplicaciones que se implementarán en las dependencias de Coordinador Eléctrico Nacional, tanto para desarrollo de nuevos proyectos, evolutivos y correctivos. Este documento es parte integral de los requerimientos de ingeniería.

# Gestión de versiones

Los desarrolladores de software deberán tener configurado en sus equipos locales el software Git con el cual se podrán comunicar con la plataforma Git del Coordinador.

Es responsabilidad del equipo de desarrollo (individualizado por cada desarrollador) el mantener diariamente el código fuente actualizado en las ramas del proyecto asignado por el Coordinador en la plataforma Git.

## Flujo de ejecución general

* + 1. El desarrollador deberá hacer un “push” del último commit a la rama de “desarrollo” del Git Coordinador asignada. Cada commit creado localmente debe tener un mensaje descriptivo de lo que se está subiendo a la rama.
    2. Git a su vez actualizará la rama correspondiente a “desarrollo”, quedando el último código actualizado.
    3. De manera automática, Git mediante un servicio “Webhook” gatilla la llamada a Jenkins para la ejecución del “pipeline” correspondiente a desarrollo.
    4. Jenkins gatilla el “pipeline” asignado a desarrollo y procede a:
    5. Realizar el build de los componentes
    6. Comunicar con el servicio de “revisión de código” Fortify

## Flujo de ejecución de error

* + 1. El desarrollador de debe realizar un rollback del código:

Ubicar el ID del commit al cual queremos regresar

| git reset --hard <commit-id> |
| --- |

Indicar que este commit será el nuevo

| git reset --soft HEAD@{1} |
| --- |

Subir los cambios

| git commit -m "rollback commit-id-origen commit-id-destino motivo-rollback" |
| --- |

## 

## Merge Request Desarrollo a QA

Desde este procedimiento en adelante, ya no hay intervención del equipo de desarrollo ya que estos no deberán tener injerencia en ambientes de pruebas QA y menos en ambientes productivos.

El proveedor será el responsable de crear la solicitud “Merge Request” en Git del Coordinador.

Esta solicitud debe contener:

* + 1. El título de la solicitud “merge” debe ser claro y preciso.
    2. Detalle de los cambios o funcionalidades que se están pasando a la rama de pruebas QA, con el objetivo de que se declare qué se está entregando (Release Note).
    3. Adjunta informe de revisión de código.
    4. Tener el cuidado de que la solicitud esté entre la rama “desarrollo” hacia la rama “QA”.
    5. En lo posible asignarlo al responsable de pruebas o QA asignado al proyecto, de lo contrario dejar sin asignar.
    6. Velar por que este sin seleccionar la casilla de “Delete source Branch”
    7. La solicitud de “merge” quedará pendiente de ejecutarse hasta que el responsable de pruebas QA lo acepte y ejecute.
    8. Si se estima necesario es posible iterar con mensajes aclaratorios.

El responsable de pruebas QA asignado, cuando estime conveniente podrá aprobar el “merge”, con esto se actualizará el código en la rama de pruebas QA.

Una vez aprobado y ejecutado el “merge” GitLab gatillará mediante Webhook la llamada a Jenkins para ejecutar el pipeline correspondiente a pruebas QA, el cual:

* + 1. Realizar el build de los componentes
    2. Comunicar con el servicio de “revisión de código” Fortify
    3. Realiza el “deploy” de la aplicación en el ambiente de pruebas QA.

En caso de error:

En este caso o en cualquier momento se podrá realizar un “revert merge” del mismo “merge request” ya aprobado, el cual dejará el código de la rama en la versión anterior y gatillará el pipeline de despliegue correspondiente dejando tanto código como ambientación en la versión anterior.

El “revert merge” puede ser ejecutado por el responsable de pruebas QA o por el líder técnico correspondiente.

## Merge Request QA a Producción

El Líder Técnico asignado será el responsable de crear la solicitud “Merge Request” en Git del Coordinador.

Esta solicitud debe contener:

* + 1. El título de la solicitud “merge” debe ser claro y preciso.
    2. Detalle de los cambios o funcionalidades que se están pasando a la rama de producción, con el objetivo de que se establezca en control de versiones lo que se está solicitando pasar.
    3. Adjuntar toda la documentación solicitada por la CAB para los despliegues y formalizaciones de paso a producción.
    4. Tener el cuidado de que la solicitud es entre la rama “pruebas QA” hacia la rama de “master”.
    5. La asignación deberá quedar pendiente de asignación hasta la aprobación del comité CAB correspondiente.
    6. Velar por que este sin seleccionar la casilla de “Delete source Branch”

La solicitud de “merge” quedará pendiente de ejecutarse hasta que el responsable de soporte y continuidad lo acepte y ejecute.

En el caso de haber observaciones estas se podrán gestionar en el mismo “merge” con mensajes aclaratorios.

El responsable de Soporte y Continuidad asignado, según la coordinación del comité CAB deberá aprobar y ejecutar el “merge”.

Una vez aprobado y ejecutado el “merge” Git gatillará mediante Webhook la llamada a Jenkins al pipeline correspondiente a producción, el cual:

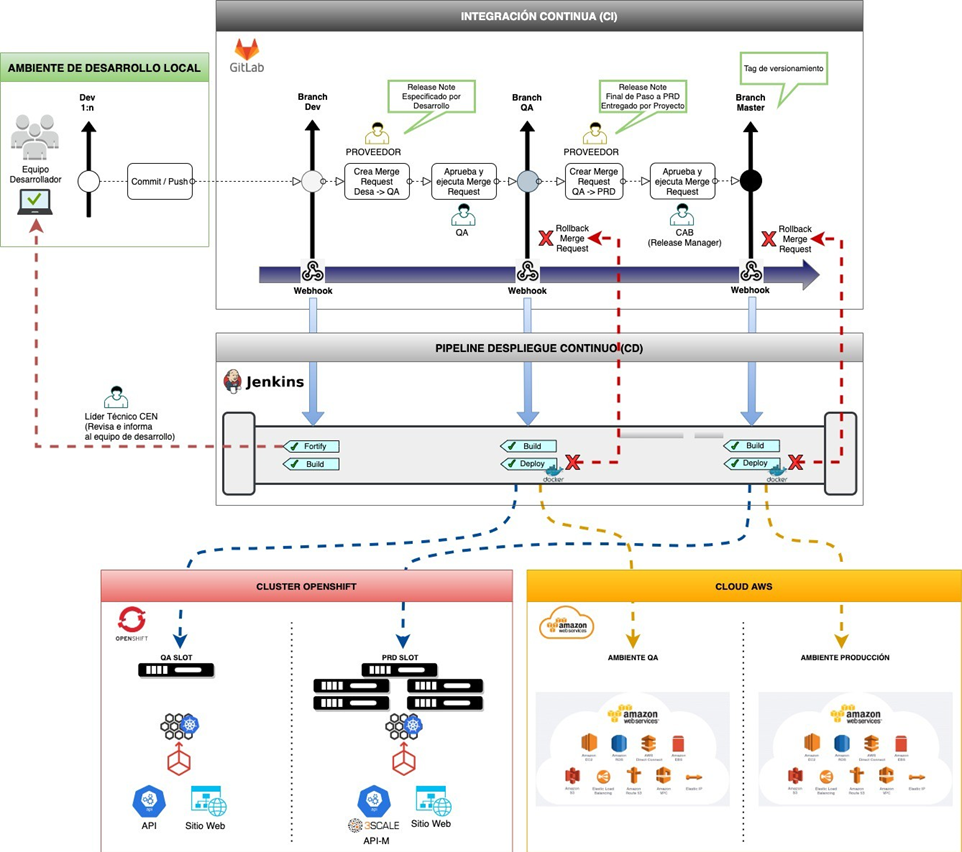
* + 1. Realizar el build de los componentes
    2. Comunicar con el servicio de “revisión de código” Fortify
    3. Realiza el “deploy” de la aplicación en el ambiente de producción.

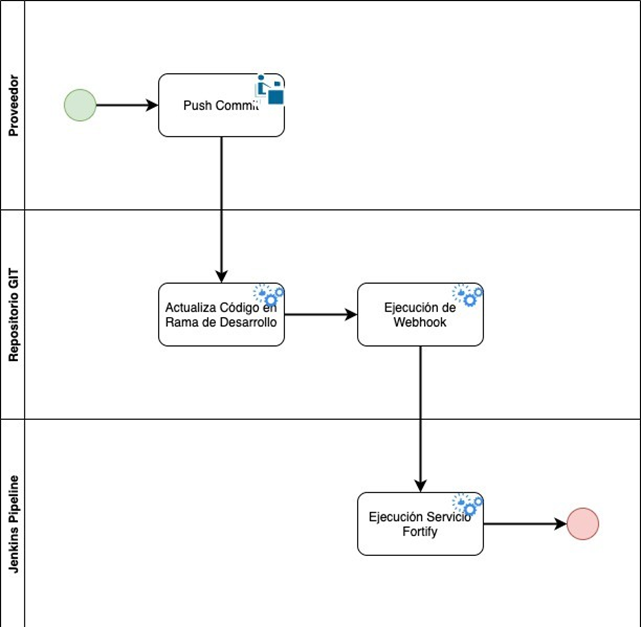
En caso de error:

En este caso o en cualquier momento se podrá realizar un “revert merge” del mismo “merge request” ya aprobado, el cual dejará el código de la rama en la versión anterior y gatillará el pipeline de despliegue correspondiente dejando tanto código como ambientación en la versión anterior.

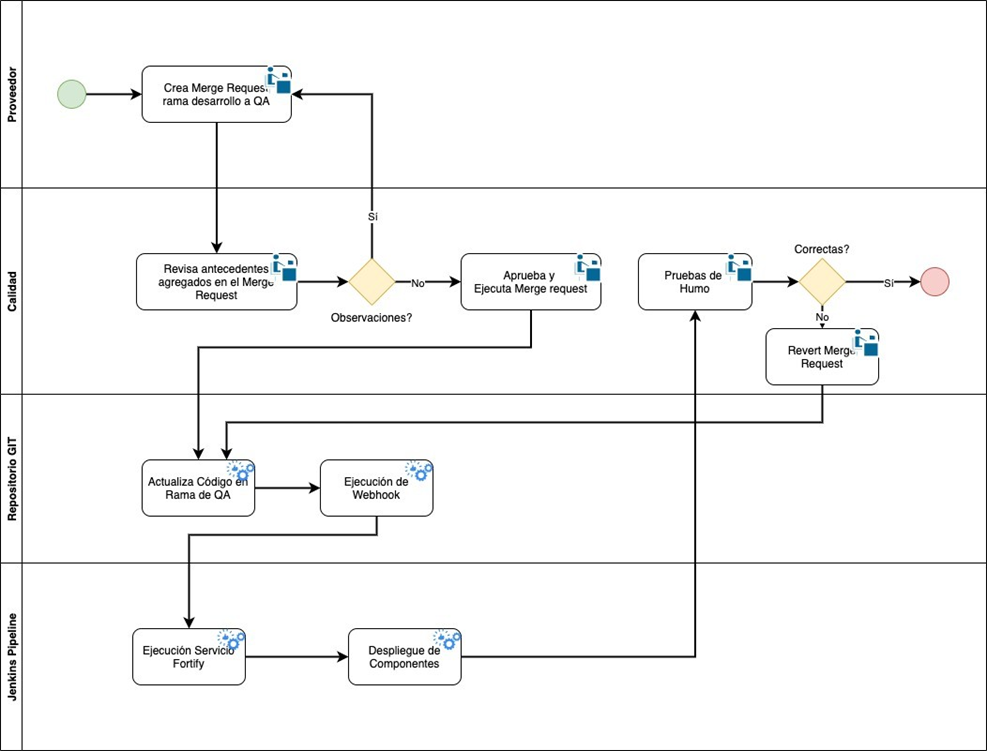
El “revert merge” debe ser ejecutado por la persona asignada de Soporte y Continuidad.

## Diagramas

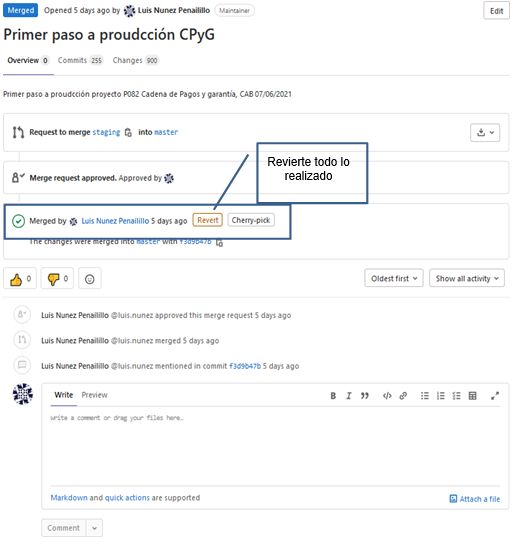
*Ilustración 1 - Crear Merge Request*

**

*Ilustración 2 - Seleccionar ramas*

**

*Ilustración 4 - Completar Formulario*



*Ilustración 5 - Revertir Merge*

## Estructura de Repositorio

La estructura de creación de repositorios contendrá la siguiente nomenclatura:

**{unidad de negocio} – {producto} – {tipo} – {repositorio código}**

* **{unidad de negocio}**: Grupo que corresponde a la unidad o área de negocio a quién pertenece el producto o sistema.
* **{producto}**: Grupo que corresponde al sistema que da soporte a la operación de la unidad de negocio.
* **{tipo}**: Grupo que corresponde a la separación técnica de arquitectura (FrontEnd, Backend)
* **{repositorio código}**: Repositorio que contendrá las ramas correspondientes al versionado del código desarrollado (desarrollo, qa y master).

## Contenido de Repositorios

Cada pieza de código deberá tener su repositorio independiente, esto permite mantener aislado cualquier intervención innecesaria del código producto de mantenciones y/o evolutivo.

Los repositorios de código deberán contener solo archivos que correspondan a las piezas de desarrollo como tal.

Los repositorios no deben contener documentos en formatos binarios como documentos de ofimática.

Se debe gestionar con cada proyecto el archivo “.gitignore” con el objetivo de no mantener información innecesaria en los repositorios git del Coordinador.

Ejemplo “.gitignore” para proyectos java spring-boot

\*#

\*.iml

\*.ipr

\*.iws

\*.jar

\*.sw?

\*~

.#\*

.\*.md.html

.DS\_Store

.attach\_pid\*

.classpath

.factorypath

.gradle

.idea

.metadata

.project

.recommenders

.settings

.springBeans

.vscode

/code MANIFEST.MF

\_site/ activemq-data bin

build

!/\*\*/src/\*\*/bin

!/\*\*/src/\*\*/build build.log

dependency-reduced-pom.xml dump.rdb

interpolated\*.xml lib/

manifest.yml out overridedb.\* target

transaction-logs

.flattened-pom.xml secrets.yml

.gradletasknamecache

.sts4-cache

## Documentación

Cada repositorio de código debe ser iniciado con al menos un archivo “README”, este archivo debe contener toda la información técnica necesaria referente al código que almacena dicho repositorio.

En el caso de que el repositorio no contenga dicha información, deberá ser considerado dentro del backlog de deuda técnica.

La información básica que debe tener este archivo “README” se compone de:

* Descripción de la funcionalidad
* Comenzando
* Pre-requisitos
* Build
* Ejecutando pruebas
* Despliegue
* Versionado
* Autores

## Acceso a Git mediante consola

* Como pre-requisito se debe tener instalado Open-SSH
* El usuario debe ser miembro del repositorio a trabajar
* Abrir una consola de comandos y ejecutar lo siguiente:
  + ssh-keygen -t ed25519 -C {nombre\_correo}@coordinador.cl
  + Referencia: https://gitlab.coordinadorelectrico.cl/help/ssh/README#generating-a-new-ssh-key-pair
* Cuando pida “Enter file in which to save key …”, solo presionar “Enter”
  + Posteriormente ingresar una clave y repetirla
  + Con un editor de texto abrir el archivo C:\Users\{user}\.ssh\ id\_ed25519
  + Copiar el contenido del archivo
  + Ingresar con el usuario definido a la configuración de usuario y seleccionar “Settings”
  + Seleccionar del menú lateral izquierdo “SSH Key”
  + Agregar el contenido copiado en el cuadro key y presionar “Add key”
* Probar la conexión: ssh -T git@gitlab.coordinadorelectrico.cl
  + Les pedirá la password ingresada:
  + Enter passphrase for key 'C:\Users\luis.nunez/.ssh/id\_ed25519'
* Finalmente clonar mediante ssh:
  + git clone [git@gitlab.coordinadorelectrico.cl](mailto:git@gitlab.coordinadorelectrico.cl)…

## Numeración de Versiones

La notación numérica será compuesta por tres números separados por puntos:

**{major}.{minor}.{revision}**

* **{major}**: Versión principal del software, consiste en un conjunto de funcionalidades concretas que son requeridas y cubiertas por dicha versión.
* **{minor}**: Indican las funcionalidades menores cubiertas en la versión de software entregada referente a evolutivos.
* **{revision}**: Se incrementa cuando hay revisiones o modificaciones en el código ante fallos y correcciones de esta.

## Criterio para modificación de versiones

Los criterios para el incremento de cada uno de los contadores de las etiquetas de versión son los siguientes:

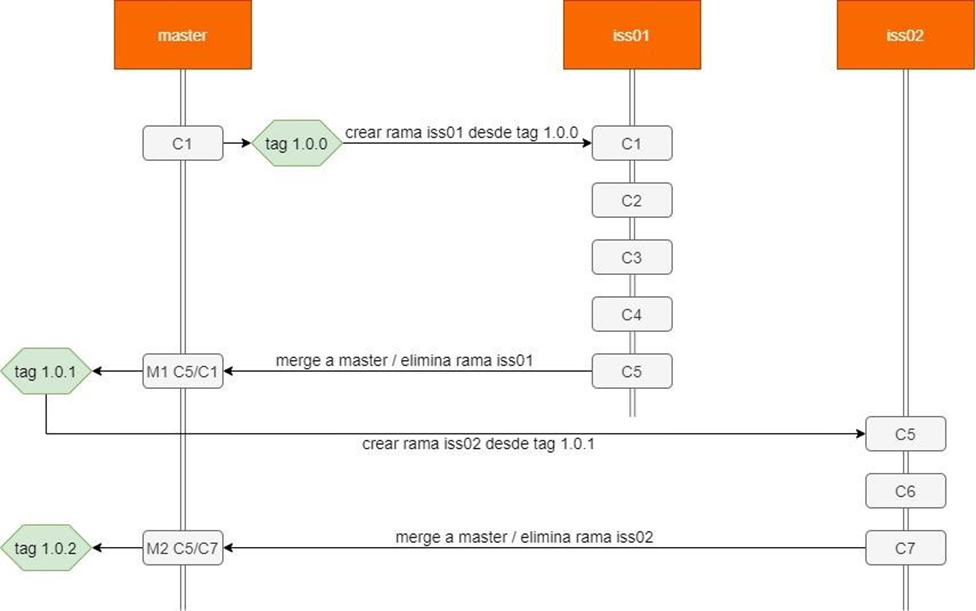
* **{major}**: Nuevas funcionalidades claves de la aplicación respecto a la versión anterior debido a la inclusión de nuevos requerimientos para el sistema, como la inclusión de nuevos módulos o una revisión completa de los existentes.
* **{minor}**: Cambios significativos en la forma en la que se ofrece la funcionalidad existente, corrección de grandes fallos del sistema o nuevas versiones evolutivas que modifican significativamente la funcionalidad ofrecida.
* **{revision}**: Se modifica por cada entrega de software que se realice.

## Flujo de Ramificación Git

### Gestión de ramas sistemas en producción

Este flujo corresponde a los aplicativos que se encuentran en operación o producción y bajo el modelo de soporte correctivo. Como premisa se debe contar con el código fuente en la rama “master” (producción).

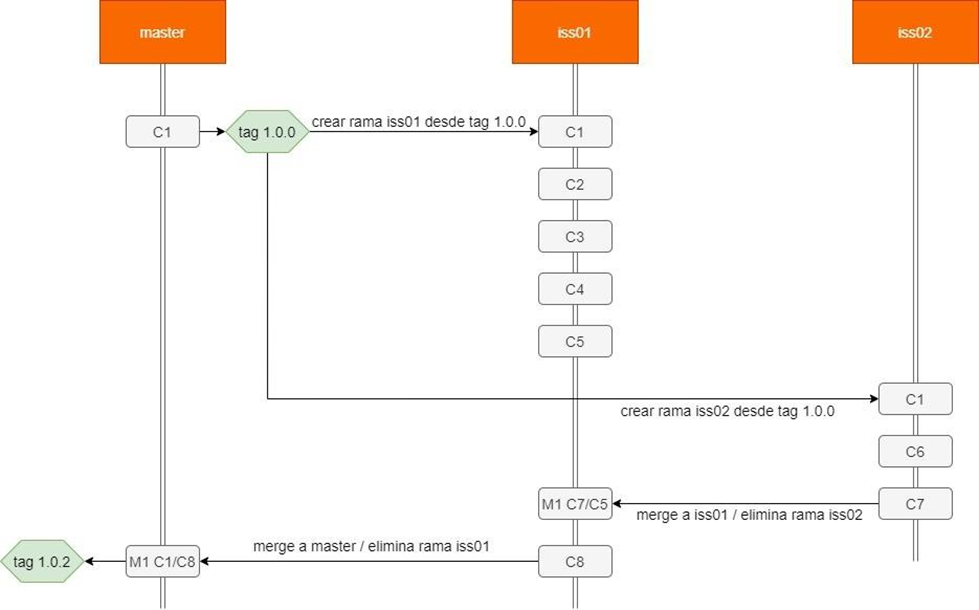
Para el ejemplo, se consta con una versión “taggeada” del código definida por el primer commit “C1” realizado a la rama “master”, bajo este commit se crea un “tag 3”, el cual nos indicará la última versión aprobada y que se encuentra desplegada en ambientes de producción.



*Ilustración 2 Gestión de Ramas Correctivos (Issues)*

La imagen anterior ejemplifica las actividades a realizar al momento de levantarse 2 errores (issues), el “iss01” y “iss02” en tiempos distintos.

* El primer paso, es solicitar la creación de las ramas (branches) con el nombre (nomenclatura) de los “issues” a abordar, para este caso se crean dos ramas que tienen la versión latest del código de la rama “master” (tag 1.0.0).
* Una vez que se tienen los accesos a las ramas “issues” los equipos de desarrollo
* Correctivo: aquellas correcciones de piezas de software que requieren de tiempos de solución urgentes (issues).
* Un “tag” es una versión estática (snapshot) del código, la cual no puede ser modificada y nos asegura la inmutabilidad de la versión. El “tag” debe ser construido según el criterio de numeración de versiones tratado con anterioridad.
* comienzan a trabajar en las correcciones del código y realizan uno o más commit de estas en las ramas correspondientes.
* El equipo de corrección del “iss01” tiene certificada su corrección por tanto se debe generar un “merge request” desde la rama “iss01” a la rama “master”, este “merge” queda con un correlativo automático.
* Como actividad final de este “issue”, se debe generar el “tag” con el objetivo de dejar marcada la última versión de código generado, para este caso “tag 1.0.1”.
* Si se genera un “issues” del nuevo reléase (tag), se debe proceder de la misma manera al paso 1.
* La eliminación de las ramas (branches) según la imagen es opcional.

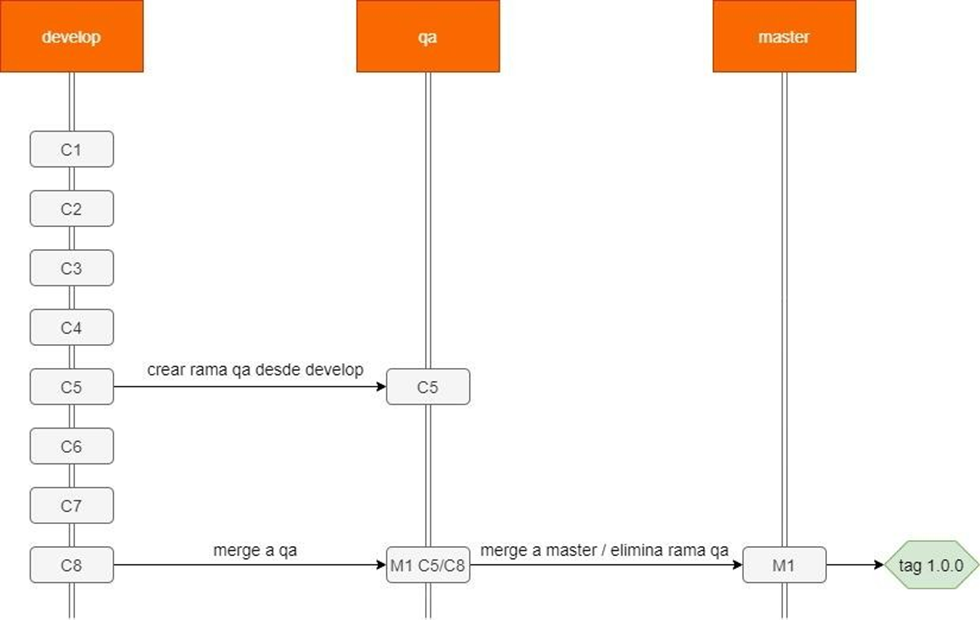
*Ilustración 3 Gestión de Ramas Correctivos (Issues) caso 2*

Este segundo caso se da cuando se requiera pasar a producción más de una corrección a la vez y realizadas por equipos de desarrollo distinto.

Aquí la variante es que se realiza el “merge request” a la rama del “issue” que concentrará las demás correcciones de “issues”.

### Gestión de ramas nuevas

Este flujo es para aquellos sistemas que son nuevos.

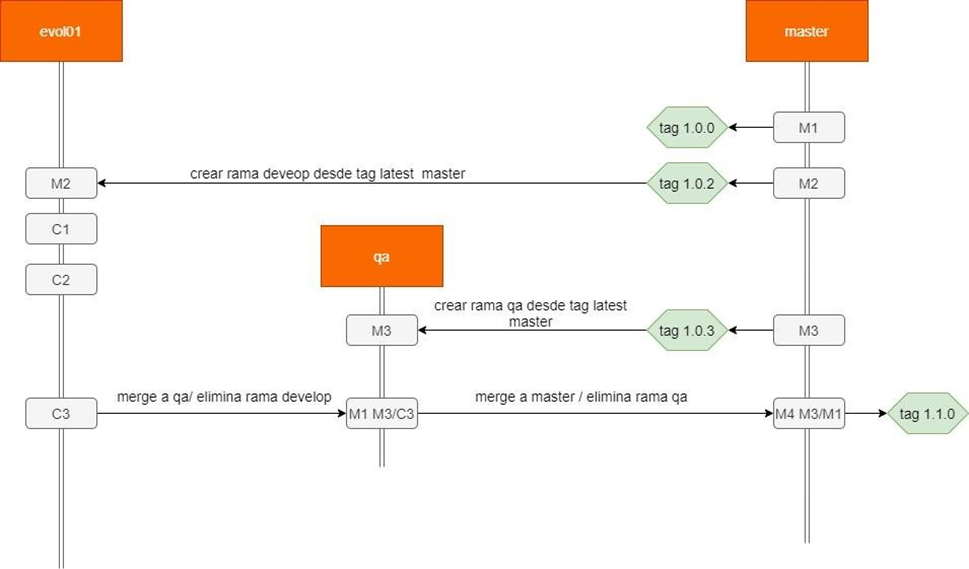


*Ilustración 4 Gestión de Ramas Sistemas Nuevos*

* Aquí el caso es más simple, ya que el equipo de desarrollo comienza con el acceso a la rama “develop” (desarrollo), en la cual va subiendo sus “commit” (C1, C2, C3, …C8).
* En el ejemplo se define que el “commit” C5 debe ser promocionado a la rama “qa”, ya sea para ser probado o certificado (MVP).
* En este caso, se debe solicitar la creación de la rama “qa” (teniendo en cuenta que es un desarrollo desde cero), el cual se lleva de forma automática el último “commit” en esa línea de tiempo (C5).
* El equipo de desarrollo continúa con la construcción y/o correcciones de “issues” informados al revisar el “commit” C5.
* Una vez finalizado el desarrollo, se debe realizar un “merge request” hacia la rama de “qa”, para su certificación.
* Si se encuentra todo “ok” en “qa”, se debe crear la rama “master” a partir de la rama certificada de “qa”.
* Finalmente se debe crear el “tag” con la versión correspondiente, para este caso “tag 1.0.0”.

### Gestión de ramas evolutivos

Este flujo define cómo se debe realizar la gestión de ramas en sistemas que sea necesario realizar mejoras evolutivas.



*Ilustración 5 Gestión de Ramas Evolutivos*

* Se debe crear la rama del evolutivo “evol01” directamente desde la última versión (latest) de la rama “master”, para este ejemplo es la “tag 1.0.2” de la línea de tiempo.
* El equipo de desarrollo trabaja en el evolutivo generando varios “commit” (C1, C2 y C3).
* Se define que el “commit” C3 es el que contiene el evolutivo desarrollado.
* Durante el tiempo de desarrollo del evolutivo se han generado nuevas versiones de la rama “master” debido a correcciones realizadas (issues). Por tanto, se debe crear la rama “qa” con la última versión de la rama “master” (tag 1.0.3, que es diferente a la línea base con la cual se comenzó el desarrollo del evolutivo).
* Como ya se tiene la última versión en la rama “qa” se debe realizar un “merge request” hacia esta rama desde la rama del evolutivo “evol01”.
* Una vez combinados las correcciones y las mejoras en la rama “qa”. Se procede a generar un nuevo “merge request” hacia la rama “master” teniendo así una nueva versión del código. Como esta es una versión que nace del evolutivo mayor y según el apartado de “Numeración de Versiones” esta debe promoverse en su nueva versión “minor” (tag 1.0.3 => tag 1.1.0).

Evolutivo: Mejora mayor a un sistema en operación que se prolonga por más de 2 meses de desarrollo.

# Especificaciones de OpenAPI

Una Open API corporativa atraviesa por distintas fases o estados durante su desarrollo. Es decir, transita desde la fase de Definición hasta su fase Operativa, donde quedará lista para ser utilizada en Producción.

La complejidad del cambio determina cual es el curso que debe recorrer en ciclo de vida de la Open

API para poder evolucionar. Es importante indicar que una Open API corporativa va evolucionar desde su estado operativo. Independientemente del cambio, todas las versiones anteriores deben ser deprecadas tan pronto como la nueva versión alcance el estado de Definición. Esto permitirá que los nuevos consumidores siempre seleccionen la versión más reciente de la Open API.

## Fases del ciclo de vida de la OpenAPI

A continuación se describen las fases del ciclo de vida de la Open API que se recorren para cada tipo de release generado:

* Versión mayor: Se debe comenzar con el proceso de recopilación, descubrimiento y análisis de los requisitos que deben ser satisfechos. Finalmente, se realizan las restantes fases de desarrollo hasta alcanzar el estado operacional.
* Versión menor: Seguramente este tipo de release necesitará la adición de una nueva funcionalidad. Por lo tanto, se iniciará en la fase de Definición. Es en esta fase donde se determina el alcance del servicio y cómo impactará en la interfaz de la Open API corporativa. Finalmente, se realizarán las restantes fases de desarrollo hasta alcanzar el estado operacional.
* Versión parche: Se inicia desde la fase de diseño porque un release de parche no debería afectar la definición del servicio. Este cambio necesitará ser diseñado, implementado y probado para finalmente volverlo a llevar al estado operacional.

## Estrategias de migración de consumidores

La gestión de cambios siguiendo las reglas anteriores, puede ocasionar una gran cantidad de versiones

de una Open API. Para evitar disponer de cúmulos de versiones, se debe adoptar una estrategia de migración de los clientes, la cual ayudará a reducir el número de versiones existentes de la Open API corporativa.

Las siguientes estrategias pueden llevar a los clientes de la Open API a su última versión disponible:

* Automática: Para realizar este tipo de migración la nueva versión debe ser compatible con las versiones anteriores, no requiriendo de cambios en los clientes de la Open API.
* Controlada: Esta estrategia de migración requiere que la nueva versión de la Open API sea desplegada en paralelo con sus versiones anteriores. Debe establecerse un periodo de salida de las versiones anteriores, el cual debe anunciarse oportunamente. Puede ocurrir que el periodo sea diferido debido a que los clientes no alcanzaron a realizar las actividades de migración en el plazo dado.
* Forzada: El proceso consiste en desplegar la nueva versión junto a la migración de los clientes. Es importante considerar que la liberación de la versión quedará atada a la espera de las nuevas versiones de los clientes que consumen esta Open API.

## Formato

Un documento de OpenAPI que se ajusta a la especificación de OpenAPI es en sí mismo un objeto JSON, que puede representarse en formato JSON o YAML.

Por ejemplo, si un campo tiene un valor de matriz, se utilizará la representación de matriz JSON:

{

"field": [ 1, 2, 3 ]

}

Todos los nombres de campo de la especificación distinguen entre mayúsculas y minúsculas. Esto incluye todos los campos que se utilizan como claves en un mapa, excepto donde se indique explícitamente que las claves no distinguen entre mayúsculas y minúsculas.

El esquema expone dos tipos de campos: campos fijos, que tienen un nombre declarado, y campos con patrón, que declaran un patrón de expresiones regulares para el nombre del campo.

Los campos con patrón DEBEN tener nombres únicos dentro del objeto contenedor.

Para preservar la capacidad de ida y vuelta entre los formatos YAML y JSON, se RECOMIENDA la versión 1.2 de YAML junto con algunas restricciones adicionales:

* Las etiquetas DEBEN limitarse a las permitidas por el conjunto de reglas del esquema JSON.
* Las claves utilizadas en los mapas YAML DEBEN estar limitadas a una cadena escalar, según lo definido por el conjunto de reglas del esquema YAML Failsafe.

**Nota:** Si bien los documentos de OpenAPI pueden definir las API en formato YAML o JSON, no es necesario que los cuerpos de solicitud y respuesta de la API y otro contenido sean JSON o YAML.

## Tipos de Datos

Los tipos de datos en la OEA se basan en los tipos admitidos por el Borrador de especificación de esquema JSON 2020-12. Tenga en cuenta que también se admite el número entero como tipo y se define como un número JSON sin una fracción o parte de exponente. Los modelos se definen mediante el objeto de esquema, que es un superconjunto del borrador de especificación de esquema JSON 2020-12.

Según lo definido por el vocabulario de validación de esquemas JSON, los tipos de datos pueden tener una propiedad de modificador opcional: formato. OAS define formatos adicionales para proporcionar detalles finos para tipos de datos primitivos.

Los formatos definidos por la OEA son:

| **Tipo** | **Formato** | **Comentarios** |
| --- | --- | --- |
| integer | int32 | signed 32 bits |
| integer | int64 | signed 64 bits (a.k.a long) |
| number | float |  |
| number | double |  |
| string | password | A hint to UIs to obscure input. |

### 

#### Campos fijos

##### 

| **Field Name** | **Type** | **Description** |
| --- | --- | --- |
| openapi | string | ***REQUIRED***. Esta cadena DEBE ser el número de versión de la especificación de OpenAPI que utiliza el documento de OpenAPI. El campo openapi DEBE ser utilizado por herramientas para interpretar el documento OpenAPI. Esto no está relacionado con la cadena de API info.version. |
| info | [Info Object](https://spec.openapis.org/oas/v3.1.0#infoObject) | ***REQUERIDO.*** *Proporciona metadatos sobre la API. Los metadatos PUEDEN ser utilizados por herramientas según sea necesario.* |
| jsonSchemaDialect | string | El valor predeterminado para la palabra clave $ esquema dentro de los Objetos de esquema contenidos en este documento de la OEA. DEBE tener la forma de un URI. |
| servers | [[Server Object](https://spec.openapis.org/oas/v3.1.0#serverObject)] | Una matriz de objetos de servidor, que proporciona información de conectividad a un servidor de destino. Si no se proporciona la propiedad de los servidores, o es una matriz vacía, el valor predeterminado sería un Objeto de servidor con un valor de URL de /. |
| paths | [Paths Object](https://spec.openapis.org/oas/v3.1.0#pathsObject) | Las rutas y operaciones disponibles para la API. |
| webhooks | Map[string, [Path Item Object](https://spec.openapis.org/oas/v3.1.0#pathItemObject) | [Reference Object](https://spec.openapis.org/oas/v3.1.0#referenceObject)] ] | Los webhooks entrantes que PUEDEN recibirse como parte de esta API y que el consumidor de API PUEDE optar por implementar. Estrechamente relacionada con la función de devolución de llamadas, esta sección describe las solicitudes iniciadas de otra manera que no sea una llamada API, por ejemplo, por un registro fuera de banda. El nombre de la clave es una cadena única para hacer referencia a cada webhook, mientras que el objeto de elemento de ruta (al que se hace referencia opcionalmente) describe una solicitud que puede iniciar el proveedor de API y las respuestas esperadas. Hay un ejemplo disponible. |
| components | [Components Object](https://spec.openapis.org/oas/v3.1.0#componentsObject) | Un elemento para contener varios esquemas para el documento. |
| security | [[Security Requirement Object](https://spec.openapis.org/oas/v3.1.0#securityRequirementObject)] | Una declaración de qué mecanismos de seguridad se pueden utilizar en la API. La lista de valores incluye objetos de requisitos de seguridad alternativos que se pueden utilizar. Solo se debe satisfacer uno de los objetos de requisito de seguridad para autorizar una solicitud. Las operaciones individuales pueden anular esta definición. Para que la seguridad sea opcional, se puede incluir un requisito de seguridad vacío ({}) en la matriz. |
| tags | [[Tag Object](https://spec.openapis.org/oas/v3.1.0#tagObject)] | Una lista de etiquetas utilizadas por el documento con metadatos adicionales. El orden de las etiquetas se puede utilizar para reflejar su orden mediante las herramientas de análisis. No se deben declarar todas las etiquetas que utiliza el objeto de operación. Las etiquetas que no están declaradas PUEDEN estar organizadas al azar o según la lógica de las herramientas. Cada nombre de etiqueta en la lista DEBE ser único. |
| externalDocs | [External Documentation Object](https://spec.openapis.org/oas/v3.1.0#externalDocumentationObject) | Documentación externa adicional. |

Este objeto puede ampliarse con extensiones de especificación.

## Objeto de rutas

Contiene las rutas relativas a los puntos finales individuales y sus operaciones. La ruta se agrega a la URL del Objeto de servidor para construir la URL completa. Las rutas PUEDEN estar vacías debido a restricciones de la lista de control de acceso (ACL).

### Campos Estandarizados

##### 

| **Field Pattern** | **Type** | **Description** |
| --- | --- | --- |
| /{path} | [Path Item Object](https://spec.openapis.org/oas/v3.1.0#pathItemObject) | Una ruta relativa a un punto final individual. El nombre del campo DEBE comenzar con una barra inclinada (/). La ruta se agrega (sin resolución de URL relativa) a la URL expandida desde el campo de URL del Objeto de servidor para construir la URL completa. Se permite la creación de plantillas de ruta. Al hacer coincidir las URL, las rutas concretas (sin plantilla) coincidirían antes que sus contrapartes con plantilla. Las rutas con plantilla con la misma jerarquía pero con nombres de plantilla diferentes NO DEBEN existir, ya que son idénticas. En caso de una coincidencia ambigua, depende de las herramientas decidir cuál utilizar. |

### Coincidencia según plantillas de rutas

Asumiendo los siguientes caminos, la definición concreta, /pets/mine, will be matched first if used:

/pets/{petId}

/pets/mine

Las siguientes rutas se consideran idénticas e inválidas:

/pets/{petId}

/pets/{name}

Lo siguiente puede dar lugar a una resolución ambigua:

/{entity}/me

/books/{id}

### Objeto de rutas de Ejemplo

{

"/pets": {

"get": {

"description": "Returns all pets from the system that the user has access to",

"responses": {

"200": {

"description": "A list of pets.",

"content": {

"application/json": {

"schema": {

"type": "array",

"items": {

"$ref": "#/components/schemas/pet"

}

}

}

}

}

}

}

}

}

/pets:

get:

description: Returns all pets from the system that the user has access to

responses:

'200':

description: A list of pets.

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/pet'

### Objeto de elemento de ruta

Describe las operaciones disponibles en una única ruta. Un elemento de ruta PUEDE estar vacío debido a restricciones de ACL. La ruta en sí todavía está expuesta al visor de documentación, pero no sabrán qué operaciones y parámetros están disponibles.

#### Campos fijos

##### 

| **Nombre de Campo** | **Tipo** | **Descripción** |
| --- | --- | --- |
| $ref | string | Permite una definición referenciada de este elemento de ruta. La estructura referenciada DEBE tener la forma de un objeto de elemento de ruta. En caso de que aparezca un campo Objeto de elemento de ruta tanto en el objeto definido como en el objeto referenciado, el comportamiento no está definido. Consulte las reglas para resolver referencias relativas. |
| summary | string | Un resumen de cadena opcional, destinado a aplicarse a todas las operaciones en esta ruta. |
| description | string | Una descripción de cadena opcional, destinada a aplicarse a todas las operaciones en esta ruta. La sintaxis CommonMark PUEDE usarse para la representación de texto enriquecido. |
| get | [Operation Object](https://spec.openapis.org/oas/v3.1.0#operationObject) | Una definición de una operación GET en esta ruta. |
| put | [Operation Object](https://spec.openapis.org/oas/v3.1.0#operationObject) | Una definición de una operación PUT en esta ruta. |
| post | [Operation Object](https://spec.openapis.org/oas/v3.1.0#operationObject) | Una definición de una operación POST en esta ruta. |
| delete | [Operation Object](https://spec.openapis.org/oas/v3.1.0#operationObject) | Una definición de una operación DELETE en esta ruta. |
| options | [Operation Object](https://spec.openapis.org/oas/v3.1.0#operationObject) | Una definición de una operación OPTIONS en esta ruta. |
| head | [Operation Object](https://spec.openapis.org/oas/v3.1.0#operationObject) | Una definición de una operación HEAD en esta ruta. |
| patch | [Operation Object](https://spec.openapis.org/oas/v3.1.0#operationObject) | Una definición de una operación PATCH en esta ruta. |
| trace | [Operation Object](https://spec.openapis.org/oas/v3.1.0#operationObject) | Una definición de una operación TRACE en esta ruta. |
| servers | [[Server Object](https://spec.openapis.org/oas/v3.1.0#serverObject)] | Una matriz de servidores alternativa para dar servicio a todas las operaciones en esta ruta. |
| parameters | [[Parameter Object](https://spec.openapis.org/oas/v3.1.0#parameterObject) | [Reference Object](https://spec.openapis.org/oas/v3.1.0#referenceObject)] | Una lista de parámetros que son aplicables para todas las operaciones descritas en esta ruta. Estos parámetros se pueden anular en el nivel de operación, pero no se pueden eliminar allí. La lista NO DEBE incluir parámetros duplicados. Un parámetro único se define mediante una combinación de nombre y ubicación. La lista puede usar el objeto de referencia para vincular a los parámetros que se definen en los componentes / parámetros del objeto OpenAPI. |

This object *MAY* be extended with [Specification Extensions](https://spec.openapis.org/oas/v3.1.0#specificationExtensions).

### Objetos de flujos OAuth

Permite la configuración de los flujos OAuth soportados.

#### Campos fijos

##### 

| **Field Name** | **Type** | **Description** |
| --- | --- | --- |
| implicit | [OAuth Flow Object](https://spec.openapis.org/oas/v3.1.0#oauthFlowObject) | Configuración para el flujo implícito de OAuth |
| password | [OAuth Flow Object](https://spec.openapis.org/oas/v3.1.0#oauthFlowObject) | Configuración para el flujo de contraseña de propietario de recurso de OAuth |
| clientCredentials | [OAuth Flow Object](https://spec.openapis.org/oas/v3.1.0#oauthFlowObject) | Configuración para el flujo de credenciales de cliente OAuth. Aplicación anteriormente llamada en OpenAPI 2.0. |
| authorizationCode | [OAuth Flow Object](https://spec.openapis.org/oas/v3.1.0#oauthFlowObject) | Configuración para el flujo del código de autorización OAuth. Anteriormente llamado accessCode en OpenAPI 2.0. |

This object *MAY* be extended with [Specification Extensions](https://spec.openapis.org/oas/v3.1.0#specificationExtensions).

#### Objeto de flujo OAuth

Detalles de configuración soportada para flujo OAuth

##### Campos fijos

##### 

| **Field Name** | **Type** | **Applies To** | **Description** |
| --- | --- | --- | --- |
| authorizationUrl | string | oauth2 ("implicit", "authorizationCode") | ***REQUERIDO.*** *La URL de autorización que se utilizará para este flujo. DEBE tener la forma de una URL. El estándar OAuth2 requiere el uso de TLS.* |
| tokenUrl | string | oauth2 ("password", "clientCredentials", "authorizationCode") | ***REQUERIDO.*** *La URL del token que se utilizará para este flujo. DEBE tener la forma de una URL. El estándar OAuth2 requiere el uso de TLS.* |
| refreshUrl | string | oauth2 | La URL que se utilizará para obtener tokens de actualización. DEBE tener la forma de una URL. El estándar OAuth2 requiere el uso de TLS. |
| scopes | Map[string, string] | oauth2 | ***REQUERIDO.*** *Los ámbitos disponibles para el esquema de seguridad OAuth2. Un mapa entre el nombre del alcance y una breve descripción del mismo. El mapa PUEDE estar vacío.* |

## Referencia relativa en URLs

A menos que se especifique lo contrario, todas las propiedades que son URL PUEDEN ser referencias relativas según lo definido por [RFC3986]. A menos que se especifique lo contrario, las referencias relativas se resuelven utilizando las URL definidas en el Objeto de servidor como URL base. Tenga en cuenta que estos mismos PUEDEN ser relativos al documento de referencia.

# Especificaciones de API REST

REST define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

## Motivaciones

De la arquitectura subyacente a la Web surge el modelo REST motivado por lo siguiente:

* Escalabilidad de la interacción con los componentes, teniendo como ejemplo la Web ha crecido exponencialmente sin degradar su rendimiento. Generalidad de interfaces: gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial.
* Puesta en funcionamiento independiente: este hecho es una realidad que debe tratarse cuando se trabaja en Internet.
* Los clientes y servidores pueden ser puestos en funcionamiento con agilidad, ya que HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs, a través de la habilidad para crear nuevos métodos y tipos de contenido.
* Compatibilidad con componentes intermedios: los más populares intermediarios son de proxys para Web, las cachés, por su lado se utilizan para mejorar el rendimiento, otros permiten reforzar las políticas de seguridad como firewalls. Por tanto, la compatibilidad con componentes intermediarios nos permite reducir la latencia de interacción, reforzar la seguridad y encapsular otros sistemas.

## Restricciones

* Identificación de recursos y manipulación de ellos a través de representaciones: esto se consigue mediante el uso de URIs. HTTP es un protocolo centrado en URIs. Los recursos son los objetos lógicos a los que se le envían mensajes. Los recursos no pueden ser directamente accedidos o modificados. Más bien se trabaja con representaciones de ellos. Cuando se utiliza un método PUT para enviar información, se obtiene una representación. Internamente el estado del recurso puede ser cualquier cosa desde una base de datos relacional a un fichero de texto.

* Mensajes autodescriptivos: REST establece que los mensajes HTTP deberían ser tan descriptivos como sea posible. Esto hace posible que los intermediarios interpreten los mensajes y ejecuten servicios en nombre del usuario. Uno de los modos en que HTTP logra esto, es por medio del uso de varios métodos estándares, muchos encabezamientos y un mecanismo de direccionamiento. Por ejemplo, las cachés Web saben que por defecto el comando GET es cacheable (ya que es side-effect-free) en cambio POST no lo es. Además saben cómo consultar las cabeceras para controlar la caducidad de la información. HTTP es un protocolo sin estado y cuando se utiliza adecuadamente, es posible interpretar cada mensaje sin ningún conocimiento de los mensajes precedentes. Por ejemplo, en vez de loguearse del modo que lo hace el protocolo FTP, HTTP envía esta información en cada mensaje.
* Utilización de hipermedia como un mecanismo para gestionar el estado de la aplicación. El estado actual de una aplicación Web debería ser capturado en uno o más documentos de hipertexto, residiendo tanto en el cliente como en el servidor. El servidor conoce sobre el estado de sus recursos, aunque no intenta seguirle la pista a las sesiones individuales de los clientes. Esta es la misión del navegador, el sabe como navegar de recurso a recurso, recogiendo información que él necesita o cambiar el estado que él necesita cambiar.

## Los 4 principios de REST

### REST utiliza los métodos HTTP de manera explícita

Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP.

De acuerdo a esta asociación:

* se usa POST para crear un recurso en el servidor
* se usa GET para obtener un recurso
* se usa PUT para cambiar el estado de un recurso o actualizarlo
* se usa DELETE para eliminar un recurso

### REST no mantiene estados

* El uso de servidores intermedios para mejorar la escalabilidad hace necesario que los clientes de servicios web REST envíen peticiones completas e independientes; es decir, se deben enviar peticiones que incluyan todos los datos necesarios para cumplir el pedido, de manera que los componentes en los servidores intermedios puedan redireccionar y gestionar la carga sin mantener el estado localmente entre las peticiones.
* Una petición completa e independiente hace que el servidor no tenga que recuperar ninguna información de contexto o estado al procesar la petición. Una aplicación o cliente de servicio web REST debe incluir dentro del encabezado y del cuerpo HTTP de la petición todos los parámetros, contexto y datos que necesita el servidor para generar la respuesta. De esta manera, el no mantener estado mejora el rendimiento de los servicios web y simplifica el diseño e implementación de los componentes del servidor, ya que la ausencia de estado en el servidor elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.
* Un servicio sin estado mueve la responsabilidad de mantener el estado al cliente de la aplicación. En un servicio web REST, el servidor es responsable de generar las respuestas y proveer una interfaz que le permita al cliente mantener el estado de la aplicación por su cuenta. Por ejemplo, en el mismo ejemplo de una petición de datos en múltiples páginas, el cliente debería incluir el número de página a recuperar en vez de pedir "la siguiente".
* Este aspecto del diseño de un servicio web REST puede descomponerse en dos conjuntos de responsabilidades, como una separación de alto nivel que clarifica cómo puede mantenerse un servicio sin estado.

#### Responsabilidad del servidor

* Genera respuestas que incluyen enlaces a otros recursos para permitirle a la aplicación navegar entre los recursos relacionados. Este tipo de respuestas tiene enlaces embebidos. De la misma manera, si la petición es hacia un padre o un recurso contenedor, entonces una respuesta REST típica debería también incluir enlaces hacia los hijos del padre o los recursos subordinados, de manera que se mantengan conectados.
* Genera respuestas que indican si son susceptibles de caché o no, para mejorar el rendimiento al reducir la cantidad de peticiones para recursos duplicados, y para lograr eliminar algunas peticiones completamente. El servidor utiliza los atributos Cache-Control y Last-Modified de la cabecera en la respuesta HTTP para indicarlo.

#### Responsabilidades del cliente de la aplicación

* Utiliza el atributo Cache-Control del encabezado de la respuesta para determinar si debe cachear el recurso (es decir, hacer una copia local del mismo) o no. El cliente también lee el atributo Last-Modified y envía la fecha en el atributo If-Modified-Since del encabezado para preguntarle al servidor si el recurso cambió desde entonces. Esto se conoce como GET Condicional, y ambos encabezados van de la mano con la respuesta del servidor 304 (No Modificado) y se omite al recurso que se había solicitado si no hubo cambios desde esa fecha. Una respuesta HTTP 304 significa que el cliente puede seguir usando la copia local de manera segura, evitando así realizar las peticiones GET hasta tanto el recurso no cambie.
* Envía peticiones completas que pueden ser serviciadas en forma independiente a otras peticiones. Esto implica que el cliente hace uso completo de los encabezados HTTP tal como está especificado por la interfaz del servicio web, y envía las representaciones del recurso en el cuerpo de la petición. El cliente envía peticiones que hacen muy pocas presunciones sobre las peticiones anteriores, la existencia de una sesión en el servidor, la capacidad del servidor para agregarle contexto a una petición, o sobre el estado de la aplicación que se mantiene entre las peticiones.
* Esta colaboración entre el cliente y el servicio es esencial para crear un servicio web REST sin estado. Mejora el rendimiento, ya que ahorra ancho de banda y minimiza el estado de la aplicación en el servidor.

### REST expone URIs con forma de directorios

Las URI de los servicios web REST deben ser intuitivas, hasta el punto de que sea fácil adivinarlas. Pensemos en las URI como una interfaz auto-documentada que necesita de muy poca o ninguna explicación o referencia para que un desarrollador pueda comprender a lo que apunta, y a los recursos derivados relacionados.

### REST transfiere XML, JSON, o ambos

* La representación de un recurso en general refleja el estado actual del mismo y sus atributos al momento en que el cliente de la aplicación realiza la petición. La representación del recurso son simples "fotos" en el tiempo. Esto podría ser una representación de un registro de la base de datos que consiste en la asociación entre columnas y tags XML, donde los valores de los elementos en el XML contienen los valores de las filas. O, si el sistema tiene un modelo de datos, la representación de un recurso es una fotografía de los atributos de una de las cosas en el modelo de datos del sistema.
* Por último, es bueno construir los servicios de manera que usen el atributo HTTP Accept del encabezado, en donde el valor de este campo es de tipo MIME. De esta manera, los clientes pueden pedir por un contenido en particular.

Algunos de los tipos MIME más usados para los servicios web REST son:

| **MIME-Type** | **Content-Type** |
| --- | --- |
| **JSON** | application/json |
| **XML** | application/xml |
| **XHTML** | application/xhtml+xml |

Esto permite que el servicio sea utilizado por distintos clientes escritos en diferentes lenguajes, corriendo en diversas plataformas y dispositivos. El uso de los tipos MIME y del encabezado HTTP Accept es un mecanismo conocido como negociación de contenido, el cual le permite a los clientes elegir qué formato de datos puedan leer, y minimiza el acoplamiento de datos entre el servicio y las aplicaciones que lo consumen.

## Reglas de diseño

En este apartado se especificarán las distintas reglas de diseño aplicables para la utilización de APIs REST.

### Reglas generales

| **REGLA** | **DESCRIPCIÓN** |
| --- | --- |
| **G1** | El API REST impondrá requisitos para lograr el bajo acoplamiento y también lograra desvincular el servicio de la tecnología de aplicación. |
| **G2** | El API REST contiene todos los aspectos del protocolo de transferencia HTTP, incluyendo sus métodos, códigos de respuesta y encabezados de mensajes. |
| **G3** | El API REST se compone de 4 arquetipos y cada recurso debe estar alineado a sólo uno de los siguientes:   * **Documento** (Recurso Individual) - por ejemplo, un registro de base de datos -. Su representación incluye: campos con valores y enlaces a recursos relacionados. Estos pueden tener recursos hijos de diferente tipo y son, por lo general, candidatos a ser recurso raíz. * **Colección**: directorio de recursos gestionado por el servidor. * **Repositorio**: repositorio de recursos gestionado por el cliente. El cliente puede agregar, recuperar y eliminar recursos. * **Controller** - recursos que son funciones ejecutables - con entrada y salida asociada a parámetros. Necesario cuando la acción requerida no puede ser mapeada a los métodos CRUD. |
| **G4** | JSON debe ser la tecnología determinada para la representación de los recursos. |
| **G5** | XML y otros formatos pueden, opcionalmente, pueden ser usados para la negociación de contenido con el cliente. |
| **G6** | Las entidades comunes deben ser identificadas en la URI que las identifica con la palabra clave “common”. Ejemplo:  AddressCommon |
| **G7** | La versión se indicará con "v" tras el nombre del API, por ejemplo:  APIName/v1/resource. |
| **G8** | El esquema asociado con el API tendrá su número de versión alineada con la del API y en el mismo formato, por ejemplo:  targetNamespace = "http://www.telefonica.com/schemas/UNICA/REST/common/v2" |
| **G9** | Debe tener un artefacto que le permita describir su contrato, documentar sus reglas y sus especificaciones. Este artefacto puede encontrarse descrito en formato JSON o YAML. |
| **G10** | Deben adherirse a la especificación **OpenAPI 2.0**(https://swagger.io/specification/v2/) o **OpenAPI 3.0** (https://swagger.io/specification/). |
| **G11** | Deben ser sin estado (stateless). |
| **G12** | Deben segurizarse usando el estándar abierto OAuth o algún otro mecanismo de seguridad. |
| **G13** | Debe limitarse la cantidad de datos retornados en una consulta GET usando los campos limit (cantidad de elementos a recibir) y offset (comenzando desde). |

### Reglas de nombrado de URIs y dominio

| **REGLA** | **DESCRIPCIÓN** |
| --- | --- |
| **N1** | El path de la URI debe transmitir el modelo de recursos del API, con cada ‘/’  indica división de segmentos del dominio que corresponde a un recurso único dentro de la jerarquía del modelo. |
| **N2** | Los nombres de la URI (colecciones, recursos individuales, etc.) deben ser en minúsculas, por ejemplo:  /account/billSummary/billDetail  /account/billSummary/billdetail |
| **N3** | La URI del recurso debe estar formado de acuerdo al siguiente patrón base:    {apiRoot} / {resourcePath}  o  {apirRoot} / {resourceID\*} / {resourcePath}  o  {apiRoot} / {controller}    Donde:  {resourcePath} = {collectionResourceName} [( / {resourceID} [ / {resourcePath} o {controller} ] )  o  ( /{controller} )]    {**collectionResourceName**} es el nombre de la colección de recursos (colección o  repositorio)  {**resourceID**} es el identificador único del recurso dentro de una colección o repositorio particular.  {**resourceID\*}** es el identificador único del recurso que posee recursos internos (por ejemplo, un ID de usuario que posee los recursos bajo este dominio)  {**controller**} es un acceso directo para realizar una llamada a procedimiento remoto en el servidor del API.    Además    {**apiRoot**} es definido como: {serverRoot}/{apiName}/{apiVersion}    Donde    {**apiName**} es el nombre del API  {**apiVersion**} es la versión del API (ejemplo v1)  {**serverRoot**} es específico de la implementación (por ejemplo https://api.service.telefonica.com). |
| **N4** | La URI puede contener ‘-’, pero no se recomiendan ‘\_’. |
| **N5** | La barra ‘/’ debe ser utilizada para indicar una relación jerárquica. |
| **N6** | El nombre de la URI no debe contener nombres de verbos HTTP (el controlador debe utilizarse si es necesario para mayor claridad).    Por ejemplo, **no utilizar** esto:  <http://www.service.com/entities/getEntity?id=001>    donde, getEntity es un verbo. Se recomienda utilizar el estilo REST:  <http://www.service.com/entities/001> |
| **N7** | El nombre de la URI no debe contener nombres de las funciones CRUD. |
| **N8** | La paginación, filtrado y búsqueda debe ser solicitada a través de los parámetros dentro del path. |
| **N9** | El nombre de recursos de tipo repositorio y colección debe ser en plural (por ejemplo, /productos) |
| **N10** | Crear una URL para cada recurso. |

### Reglas de métodos HTTP

| **REGLA** | **DESCRIPCIÓN** |
| --- | --- |
| **M1** | El GET debe ser utilizado para recuperar una representación de un recurso. |
| **M2** | El HEAD debe ser utilizado para recuperar las cabeceras de las respuestas. Su uso es opcional. |
| **M3** | El POST debe ser utilizado para crear un nuevo recurso dentro de una colección. El recurso creado es identificado por una URI generada por el servicio. |
| **M4** | El POST debe ser utilizado para ejecutar controladores. |
| **M5** | El PUT debe ser utilizado para crear un recurso identificado por una URI particular. |
| **M6** | El PUT debe ser utilizado para actualizar un recurso identificado por una URI particular. |
| **M7** | El DELETE debe ser utilizado para eliminar un recurso. |

## 

### Reglas relativas a la respuesta HTTP

| **REGLA** | **DESCRIPCIÓN** |
| --- | --- |
| **RS1** | Los faults siempre se devuelven en forma de códigos de error HTTP. |
| **RS2** | Se deben respetar los siguientes códigos de respuesta, interpretados en muchos casos como faults:     | **CODIGO** | **REGLA** | | --- | --- | | **2xx** | **Indica que la solicitud del cliente fue aceptado satisfactoriamente** | | **200** | OK - should be used to indicate nonspecific success | | **200** | OK - must not be used to communicate errors in the response body | | **201** | Created - must be used to indicate successful resource creation. Return message SHOULD contain a resource representation and a Location header with the created resource’s URI | | **202** | Accepted - must be used to indicate successful start of an asynchronous action | | **204** | No Content - should be used when the response body is intentionally empty | | **3xx** | **Indica que el cliente debe tomar algunas medidas adicionales con el fin de completar su petición.** | | **301** | Moved Permanently - should be used to relocate resources | | **302** | Found - should not be used | | **303** | See Other - should be used to refer the client to a different URI – can be used with a Location header containing the URI of a resource that shows the outcome of an asynchronous task. | | **304** | Not Modified - should be used to preserve bandwidth | | **307** | Temporary Redirect - should be used to tell clients to resubmit the request to another URI | | **4xx** | **Esta categoría de códigos de error señala problemas en el cliente.** | | **400** | Bad Request - may be used to indicate nonspecific failure.  The request could not be understood by the server. The client SHOULD NOT repeat the request without modifications | | **401** | Unauthorized - must be used when there is a problem with the client’s credentials | | **403** | Forbidden - should be used to forbid access regardless of authorization state. *For example, a client may be authorized to interact with some, but not all of a REST API’s resources. If the client attempts a resource interaction that is outside of its permitted scope, the REST API should respond with 403.* | | **404** | Not Found - must be used when a client’s provided URI cannot be mapped to a resource URI | | **405** | Method Not Allowed - must be used when the HTTP method is not supported | | **406** | Not Acceptable - must be used when the requested media type cannot be served | | **409** | Conflict - should be used to indicate a violation of resource state. *For example, a REST API may return this response code when a client tries to delete a non-empty store resource.* | | **412** | Precondition Failed - should be used to support conditional operations | | **415** | Unsupported Media Type - must be used when the media type of a request’s payload cannot be processed | | **5xx** | **El servidor es responsable de estos códigos de error.** | | **500** | Internal Server Error - should be used to indicate API server malfunction | |
| **RS3** | Las respuestas de error 4XX y 5XX o faults deben incluir una frase con la razón de error en el campo exceptionText (ver a continuación) siendo tan descriptiva como sea posible con el fin de ayudar a la resolución de errores. |
| **RS4** | Las respuestas de error o faults deben estar siempre acompañadas de un elemento default que incluya la siguiente información:     | **CAMPO** | **TIPO** | **OBLIGATORIEDAD** | **DESCRIPCIÓN** | | --- | --- | --- | --- | | **code** | string | Mandatorio | Código de la excepción. | | **message** | string | Mandatorio | Descripción legible del error | | **moreInfo** | URI | Opcional | Una URI que proporciona más información acerca de la excepción | | **userMessage** | string | Opcional | Un mensaje que puede ser mostrado al usuario de la aplicación que implementa el cliente del API. | |
| **RS5** | Las respuestas OK con un body vacío deben retornar el código de estado HTTP 204. |
| **RS6** | Los recursos que contengan campos binarios, como archivos o imágenes deben poder recuperarse en fragmentos. La API debería soportar el header Accept-Ranges para consultas GET con grandes binarios. También, debería implementar una consulta HEAD para retornar los headers (Accept-Ranges, Content-Type, Content-Length) que describen el recurso binario. El Content-Length indica el tamaño total del recurso, mientras que el Accept-Ranges indica si la operación soporta resultados por fragmento. |
| **RS7** | Las respuestas, que contienen datos retornados en fragmento, deben retornar el código de estado HTTP 206 y los header Accept-Ranges, Content-Type, Content-Length y Content-Range. El header Content-Length especifica el número de bytes retornados en el body del mensaje y el header Content-Range indica la parte enviada con el total de bytes que conforman el recurso. |

## 

### Reglas relativas al Header HTTP

| **REGLA** | **DESCRIPCIÓN** |
| --- | --- |
| **H1** | El **Content-Type** debe ser utilizado para indicar el tipo de tecnología utilizada, sea json, xml o xhtml+xml. |
| **H2** | **Content-Length** se debe utilizar para que el cliente pueda saber si se ha leído el número correcto de bytes y además se puede hacer un request HEAD para averiguar cuán grande es el cuerpo de la entidad, sin descargarla. |
| **H3** | **Last-Modified** debe ser utilizado en la respuesta, para determinar si la copia de la caché necesita ser actualizada con la última respuesta. |
| **H4** | **Location** debe ser utilizado para especificar la URI de un recurso recientemente creado y podría ser utilizado para redireccionar a los clientes al estado operacional de un controlador de recursos asincrónico. Este es el mejor método para implementar comportamiento asincrónico. |
| **H5** | **Cache-Control, Expire, y Date** del header HTTP de respuesta debe ser usado para mejorar el almacenamiento y manejo de caché. |
| **H6** | El Caching debe ser utilizado y correctamente gestionado de acuerdo a los parámetros anteriormente enumerados. |
| **H7** | La expiración de los headers se debe usar cuando el código de las respuestas a solicitudes GET y HEAD son 200 ("OK"). |
| **H8** | La expiración de los headers cacheados pueden manejar la caducidad utilizando las respuestas con código 3xx y 4xx, esto ayuda a reducir la cantidad de redirecciones y errores debido a cargas de recursos innecesarias. |
| **H9** | Ninguna representación debería estar aislada, es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener más información. |
| **H10** | Los pedidos y las respuestas deben contener el header x-correlator (opcional) de tipo String, el cual representa el ID de correlación de los servicios. |
| **H11** | Las operaciones de creación de recursos deben retornar el header Location con el código de estado HTTP 201. El valor almacenado en el header Location debería contener la url donde acceder al recurso creado. |
| **H12** | Las respuestas a solicitudes procesadas asincrónicamente deben retornar el header Location con el código de estado HTTP 202. El valor almacenado en el header Location debería contener la url donde consultar el estado de la solicitud. |
| **H13** | Las respuestas, que contienen datos paginados, deben incluir un header x-total-count que indica el número total de elementos disponible en la colección. |

# Tipos de fecha y moneda

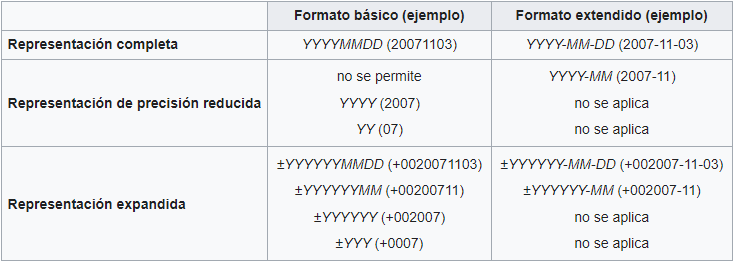
Aquí se especificarán los formatos tanto para fecha y moneda.

## Monedas

| **TF1** | El formato del campo moneda debe adherirse al estándar **ISO 4217** (https://www.iso.org/iso-4217-currency-codes.html). | | | |
| --- | --- | --- | --- | --- |
| Código: CLP | Núm: 152 | Dec: 0 | [Peso chileno](https://es.wikipedia.org/wiki/Peso_chileno) | Bandera de Chile [Chile](https://es.wikipedia.org/wiki/Chile) |

## Fechas

| **TM1** | El formato del campo de tipo fechas debe adherirse al estándar **ISO 8601** (https://www.iso.org/iso-8601-date-and-time-format.html). |
| --- | --- |



Para especificar una fecha primero se escribe el año, posteriormente el mes y a continuación el día. Por ejemplo, para especificar la fecha 14 de diciembre de 2021, se usa la notación «2021-12-14».

# Especificación de configuración Openshift

## Configuración de Scaling

Como se definió en los lineamientos de arquitectura, hay dos apartados para las configuraciones, las que deben ir y unas opcionales.

## Configuración recomendada para Scaling

Dentro de cada deployment config, es necesario configurar una cantidad mínima de réplicas y los “limits y request” para la aplicación. Además de un ejemplo de un Selector con el Label “node-role.kubernetes.io/produccion”, donde se le especifica que la aplicación debe correr en un nodo en específico, el que también contenga ese Label.

A continuación se describe un Deployment Config del producto SSO, en donde se detalla las secciones antes mencionadas:

| kind: DeploymentConfig  apiVersion: apps.openshift.io/v1  metadata:  name: sso  namespace: rh-sso  labels:  application: sso  rhsso: 7.3.6.GA  spec:  strategy:  type: Recreate  recreateParams:  timeoutSeconds: 600  resources: {}  activeDeadlineSeconds: 21600  triggers:  - type: ImageChange  imageChangeParams:  containerNames:  - sso  from:  kind: ImageStreamTag  namespace: rh-sso  name: 'sso'  lastTriggeredImage: >-  docker-registry-default.example.com/rh-sso/sso@sha256:17aeee3cef75be0df745791482949b69d026eaeb6a6364953a82c17756738eff  replicas: 2  revisionHistoryLimit: 10  test: false  selector:  deploymentConfig: sso  template:  metadata:  name: sso  creationTimestamp: null  labels:  app: sso73-x509-postgresql-persistent  application: sso  deploymentConfig: sso  annotations:  openshift.io/generated-by: OpenShiftNewApp  spec:  volumes:  - name: sso-x509-https-volume  secret:  secretName: sso-x509-https-secret  defaultMode: 420  - name: sso-x509-jgroups-volume  secret:  secretName: sso-x509-jgroups-secret  defaultMode: 420  containers:  - resources:  limits:  memory: 1Gi  cpu: 1  request:  memory: 1Gi  cpu: 1  readinessProbe:  exec:  command:  - /bin/bash  - '-c'  - /opt/eap/bin/readinessProbe.sh  timeoutSeconds: 1  periodSeconds: 10  successThreshold: 1  failureThreshold: 3  terminationMessagePath: /dev/termination-log  name: sso  livenessProbe:  exec:  command:  - /bin/bash  - '-c'  - /opt/eap/bin/livenessProbe.sh  initialDelaySeconds: 60  timeoutSeconds: 1  periodSeconds: 10  successThreshold: 1  failureThreshold: 3  env:  - name: JGROUPS\_PING\_PROTOCOL  valueFrom:  secretKeyRef:  name: rh-sso-secrets  key: JGROUPS\_PING\_PROTOCOL  ...  ports:  - name: jolokia  containerPort: 8778  protocol: TCP  ...  imagePullPolicy: Always  volumeMounts:  - name: sso-x509-https-volume  readOnly: true  mountPath: /etc/x509/https  - name: sso-x509-jgroups-volume  readOnly: true  mountPath: /etc/x509/jgroups  terminationMessagePolicy: File  image: >-  docker-registry-default.example.com/rh-sso/sso@sha256:17aeee3cef75be0df745791482949b69d026eaeb6a6364953a82c17756738eff  restartPolicy: Always  terminationGracePeriodSeconds: 75  dnsPolicy: ClusterFirst  securityContext: {}  schedulerName: default-scheduler  nodeSelector:  node-role.kubernetes.io/produccion: '' |
| --- |

Documentación de manejo de deployment: <https://docs.openshift.com/container-platform/4.7/applications/deployments/managing-deployment-processes.html>

### Configuraciones opcionales para Scaling

Dentro de las recomendaciones opcionales, se describió el HPA y VPA (Horizontal Pod Autoscalers y Vertical Pods Autoscalers respectivamente), lo cual se describe a continuación:

#### Horizontal Pod Autoscalers

Se puede definir como un recurso más de configuración (como un config map o deployment config), en donde es posible guardarlos como .yaml para realizar respaldos y además de forma automática mediante algún pipeline

Dentro de la configuración que a continuación se detalla, es importante definir la cantidad mínima (minReplicas) y máxima (maxReplicas) de réplicas y además el target de referencia, que en este ejemplo es con “DeploymentConfig” el cual tenga el nombre de “image-registry”. Además definir el porcentaje que debe superar para realizar el escalado (targetCPUUtilizationPercentage), que en este caso es 75%

**Nota: El tiempo aproximado que debe superar sobre el umbral es de 5 a 10 min para realizar el escalado**

| apiVersion: autoscaling/v1  kind: HorizontalPodAutoscaler  metadata:  name: image-registry  namespace: default  spec:  maxReplicas: 5  minReplicas: 2  scaleTargetRef:  apiVersion: apps.openshift.io/v1  kind: DeploymentConfig  name: image-registry  targetCPUUtilizationPercentage: 75 |
| --- |

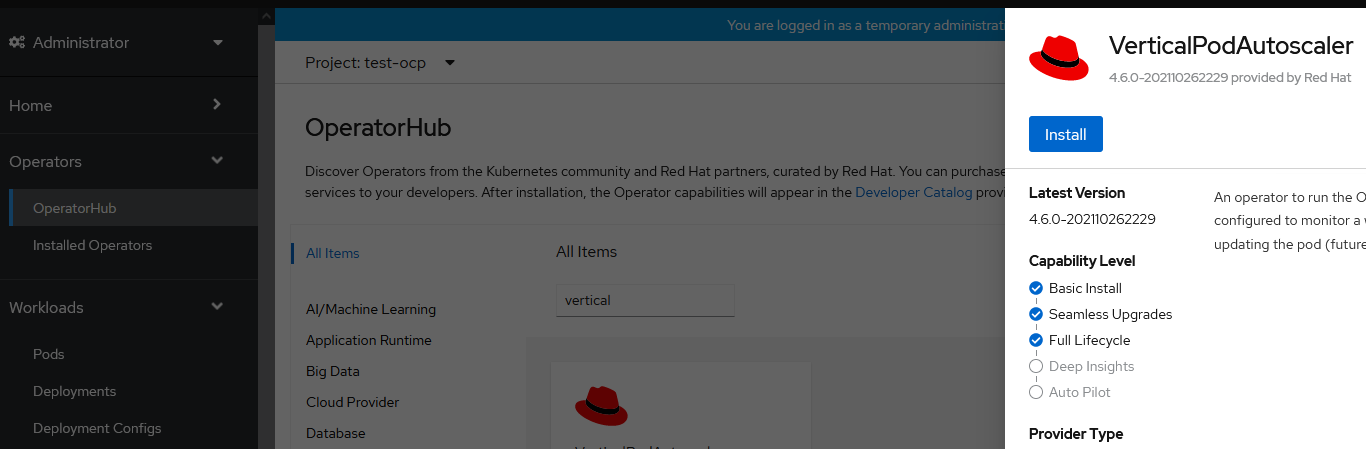
Documentación oficial:

<https://docs.openshift.com/container-platform/4.7/nodes/pods/nodes-pods-autoscaling.html>

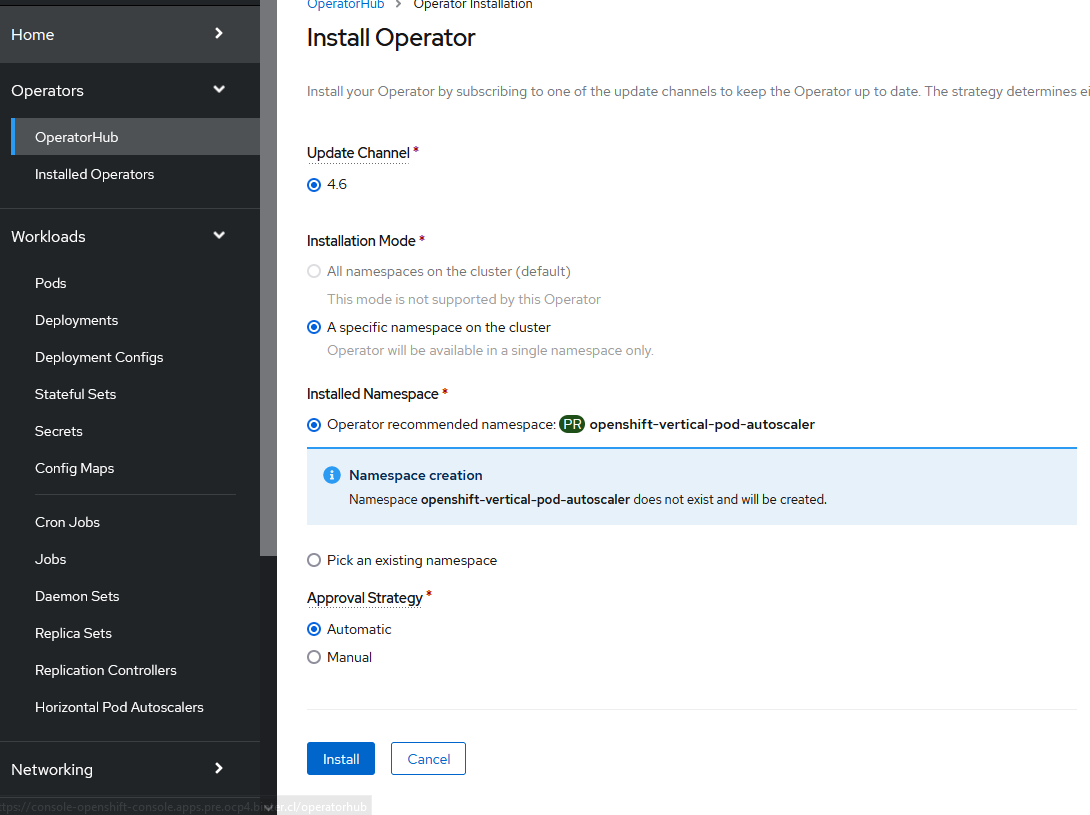
#### Vertical Pod Autoscalers

Para realizar esta configuración, es necesario tener instalado el Operator “VerticalPodAutoscaler”, el cual se encarga de monitorear el histórico las aplicaciones para luego automáticamente aplicar o recomendar la cantidad de recursos para las aplicaciones

Primero para instalar el Operator, desde el Operator Hub buscar por “VerticalPodAutoscaler” y seleccionar en instalar.



Luego asegurarse de que esté seleccionado el namespace de instalación recomendado



Una vez instalado se deben crear los recursos “VerticalPodAutoscaler” (similar a HorizontalPodAutoscaler), en donde también se hace referencia al recurso al cual debe monitorear como: deployment, stateful set, job, daemon set, replica set, o replication controller.

Ejemplo:

| apiVersion: autoscaling.k8s.io/v1  kind: VerticalPodAutoscaler  metadata:  name: vpa-recommender  spec:  targetRef:  apiVersion: "apps/v1"  kind: Deployment  name: frontend  updatePolicy:  updateMode: "Auto" |
| --- |

En donde:

* **Name**: Hace referencia al nombre del recurso VerticalPodAutoscaler
* **Kind y name**: Hace referencia a que recurso se está asociando (Deployment, job, etc) y su nombre
* **updateMode**: hace referencia a la políticas que aplicará al recurso a monitorear, en donde existen 3 distintas.
* **Auto**: donde automáticamente aplica recomendaciones de CPU y memoria al pod, este elimina los pods del proyecto y los crea con los las recomendaciones basadas en el historial de las métricas
* **Initial**: Automáticamente aplica recomendaciones solo a la creación de los Pods
* **Off**: Solamente provee las recomendaciones sin aplicarlas

# Gobernabilidad y Categorización de APIs

## Despliegues de componentes

Todos los desarrollos deben ser dockerizados y desplegados en la plataforma Openshift del

COORDINADOR, mediante despliegue continuo.

A su vez, todo servicio ya sea REST o SOAP, debe ser expuesto mediante el API Manager

(3Scale).

Se debe considerar la definición y configuración del auto-scalling el cual depende de la definición de sizing del proyecto.

## Tabla de herramientas y framework

A continuación, se listan las herramientas que deberán ser contempladas en los desarrollos, con

el objetivo de mantener un estándar para COORDINADOR.

### Lenguaje y framework de desarrollo por defecto del COORDINADOR

* Front-End: Framework Angular en su versión 8 o superior.
* Back-End: Lenguaje de desarrollo Java framework SpringBoot en su versión estable.

Cabe señalar que las versiones de estas deben ser las denominadas como “estables”.

| **Tópico** | **Herramienta** | **Versión** | **Observación** |
| --- | --- | --- | --- |
| Virtualización | VMWare |  |  |
| Cloud | AWS |  |  |
| DataLake | AWS S3 |  | Corporativo |
| Desarrollo – IDE | Spring ToolSuite | 4 + |  |
| Visual Studio Code | 1.45 + |  |
| Basado en Eclipse | Versión del año |  |
| Sublime Text | 3 |  |

| Desarrollo - Framework | Angular | 8 + |  |
| --- | --- | --- | --- |
| Spring Boot | Última versión estable |  |
| Java JDK | 8 |  |
| Python | 3.8.3 + | Debe ser justificado |
| Gestor de paquetes | Maven | Nexus | Corporativo |
| NPM |  |  |
| Biblioteca Gráfica | Bootstrap Material Design | Latest |  |
| Librerías | OpenApi | 3.0 + |  |
| Lombok | 1.18 + |  |
| SLF4J | 1.7 + |  |
| Persistencia | PostgreSQL | 10 + |  |
| Ms SqlServer | 2019 | Always On |
| MongoDB | 3.6 + |  |
| ORM | JPA | 2.0 + |  |
| MyBatis | 3.5.4 + |  |
| Mensajería | Apache Kafka  (RedHat AMQ) | Latest | Sin implementar |
| Contenedores y Gestor de Contenedores | Docker | Latest |  |
| Kubernetes | 15. + | Bajo Openshift |
| Suite de Integración | API Management  3Scale |  |  |
| JBoss Fuse |  | Sin Implementar |
| RedHat Openshift | 4.7 + |  |
| Herramienta de calidad  de datos | Informática  PowerCenter | IDQ 10.4.1 |  |

## Mnemotecnia de diseño

Todos los diagramas y diseños deben ser compatibles con la herramienta Enterprise Architect

(EA).

La nomenclatura de los desarrollos y definiciones debe seguir el siguiente estándar:

### Nomenclatura de datos

- Los nombres de bases de datos, esquemas, tablas, campos, llaves primarias y

foráneas, índices y secuencias, etc, deben ir en estilo

“lower\_case\_underscore\_separated\_identifiers”. Ej: invoice\_details

- El nombre de la base de datos debe contener el nombre de la empresa seguido

por el nombre del aplicativo. Ej: company\_application.

- El nombre de la tabla debe estar en plural y tener relación con los datos que se

van a almacenar. Ej: invoices, invoices\_details, products.

- Toda tabla debe tener una llave primaria la cual debe ser id.

- El campo que se utilizará como nexo para la relación foránea con una tabla

principal, debe ser el nombre de dicha tabla en singular seguido del sufijo \_id. Ej:

products -> product\_id, invoices -> invoice\_id, users -> user\_id.

- El resto de los campos de la tabla deben estar en singular y llevar directa relación

con los datos almacenados. Ej: users -> name, last\_name, email, password.

- Los nombres de las llaves foráneas deben estar compuestas del nombre de la

tabla donde subsiste la relación seguido del campo en cuestión y el sufijo -foreign.

Ej: invoice\_details\_invoice\_foreign, roles\_student\_foreign.

- Para las relaciones de muchos a muchos la tabla pivote lleva el nombre de las dos

tablas en orden alfabéticos (se puede cambiar el nombre por uno que represente

la relación) y contiene las columnas id de las mismas. Ej: Tabla 1: users, Tabla 2:

roles, Tabla pivote: role\_user, con las columnas user\_id y role\_id.

- El nombre de los índices debe estar compuesta de la tabla dueña del índice,

seguida del o los campos participantes del índice y el sufijo \_index. Ej:

invoice\_details\_product\_index, invoice\_details\_invoice\_product\_index.

- El nombre de las restricciones únicas debe estar compuesta del nombre de la

tabla, seguida del o los campos participantes y el sufijo \_unique. Ej:

user\_email\_unique.

- Las funciones trigger y los trigger de las tablas deben estar formadas por prefijo

tgr\_ seguida del nombre de la tabla y el nombre del trigger. Ej:

tgr\_update\_product\_provider.

- Las funciones deben estar formadas por el prefijo fun\_ seguido del nombre de la

función. Ej: fun\_convert\_unit

- Los procedimientos almacenados deben estar formados por el prefijo sp\_ seguido

del nombre del procedimiento. Ej: sp\_list\_user.

### Nomenclatura para desarrollo web

- Los nombres de las clases, archivos html, componentes, funciones, etc deben ir

en estilo “lower\_case\_underscore\_separated\_identifiers”.

- Los nombres de los componentes deben estar en singular y tener relación con la

información que va a contener o manejar seguido del sufijo .component. Ej:

product.component.html, product.component.ts.

- Los nombres de los servicios deben estar en singular y tener relación con la

información que va a contener o manejar seguido del sufijo .service. Ej:

product.service.ts.

- Para nombres compuestos debe seguir la nomenclatura anterior y entre ellos tener

una separación por “-“. Ej: product-details.service.ts.

- Las funciones o métodos deben estar formados por la acción en minúscula y el

nombre descriptivo de ámbito de la acción capitalizada. Ej: getProduct,

getProductAll, getProductFindById.

- Las carpetas de organización dentro del código deben estar formadas por el prefijo

“\_” y el tipo de componente a contener con excepción de la carpeta “pages”. Ej:

\_models, \_services, \_helpers, \_components, \_directives.

### Nomenclatura para desarrollo de microservicios

- Los nombres de las clases deben estar en estilo capitalización. Ej:

ProductController.java, ProductDTO.java, ProductService.java.

- Las funciones o métodos deben estar formados por la acción en minúscula y el

nombre descriptivo de ámbito de la acción capitalizada. Ej: getProduct,

getProductAll, getProductFindById.

## Desarrollo Java SpringBoot y componentes core

Los desarrollos base deberán ser en lenguaje Java con framework Spring Boot.

Dependencias requeridas:

* Lombok
* Spring Web
* Rest Repositories
* Spring Data JPA
* Driver de datos que corresponda
* H2 Database (solo desarrollo)

Los componentes core son objetos de desarrollos reutilizables que encapsulan ciertas reglas o

comportamientos que son utilizadas en todos los desarrollos de manera estándar, con esto se

evita la redundancia de código y se controla de manera centralizada siendo solo un punto de

corrección en caso de problemas.

Las estructuras para los desarrollos front-end serán las definidas por cada uno de los framework

de desarrollo, por ejemplo: angular/cli.

La programación de las piezas de software tanto front-end como back-end, se debe realizar

utilizando nomenclatura autocontenida y en idioma inglés.

En el repositorio Git corporativo se encuentra un ejemplo de código para el desarrollo de

microservicios:

<https://gitlab.coordinadorelectrico.cl/arquitectura/demos/ms-person>

El manejo de excepciones será provisto por un componente core para el caso de desarrollos

back-end en Java SpringBoot, el cual se encuentra disponible en el repositorio Nexus (11

Utilización de repositorio Nexus) corporativo.

La estructura del proyecto debe regirse por la siguiente nomenclatura:

pom.xml

<groupId>cl.coordinador.{aplicación}</groupId>

<artifactId>{nombre de la funcionalidad}</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>{nombre de la funcionalidad}</name>

<description>{Descripción del proyecto}</description>

## Documentación de diseño

Para la recepción de todo desarrollo software, se enumera a continuación, la documentación

mínima de la cual debe estar acompañado:

### Descripción del entorno tecnológico

* Elementos de Infraestructura
* Restricciones técnicas
* Planificación de capacidades

### Definición de Arquitectura del Sistema

* Definición de Niveles de Arquitectura del Sistema
* Modelo de Despliegue
* Patrones de Diseño y Buenas Prácticas

### Diseño del Modelo de Clases del Sistema

* Diseño de la Lógica de Negocio (Model)
* Diseño del Controlador (Controller)
* Diseño de la Vista (View)
* Diseño de Persistencia

### Modelamiento de datos

* Modelo de datos, el cual debe incluir modelamiento de privilegios de acceso.
* Diccionario lógico de datos, el cual debe contener como mínimo el nombre de entidad, descripción de entidad, atributos de entidad y llaves.

### Definición de la Interfaz de Servicios del Sistema:

* Exposición de Interfaces APIs.
* Consumo de Interfaces APIs.

El proveedor deberá suministrar en formato digital los manuales correspondientes, los cuales

deben ser compatibles con las herramientas de visualización vigentes en COORDINADOR.

## Revisión y aprobación

El proveedor deberá solicitar la validación de los documentos de diseño de arquitectura de

sistema, en la etapa de aprobación de ingeniería detalles por el área de arquitectura de la

Subgerencia de Ingeniería de Software y Arquitectura.

Adicionalmente durante la etapa de construcción (desarrollo) el código fuente debe ser cargado en GIT o repositorio, para su validación por parte del COORDINADOR de acuerdo con los estándares definidos.

## Ambientes cloud / on-Premise

Las soluciones podrán ser implementadas tanto en ambientes cloud como en ambientes on-

premise siendo este último la opción por defecto a seleccionar, sin embargo, la definición y diseño debe ser evaluada previa presentación al comité de arquitectura del Coordinador, y validada por el área de arquitectura e infraestructura.

Como premisa, todo desarrollo y software propietario deberá ser compatible con ambos ambientes, con el objeto de tener la posibilidad de migrar entre ellos sin impactos en los

componentes desarrollados.

Sin perjuicio de lo anterior, en el caso de que se presenten desarrollos o software propietarios

que solo sean compatibles en un ambiente, esto deberá ser explicado en detalle por el proyecto

justificando la necesidad exclusiva de componentes, además de identificar los costos asociados

en el caso de no contar con un dimensionamiento contratado por el COORDINADOR, llámese licencias y/o crecimiento de infraestructura.

### Componentes cloud

Dentro del diseño arquitectónico de la solución se podrán recomendar y/o utilizar componentes

cloud como servicios SaaS, IaaS y PaaS, así como también otros componentes específicos que

son parte de la solución propuesta. No obstante, el uso de estos componentes debe ser validado tanto por los costos asociados como también por su implementación, configuración y posterior administración. Por lo tanto, el proveedor deberá justificar el uso de dichos componentes siempre y cuando no hayan sido solicitados explícitamente por COORDINADOR.

Sin perjuicio de lo anterior, el proveedor deberá entregar las recomendaciones de escalamiento

y/o dimensionamientos orientados a la solución a desarrollar o implementar, así como también,

deberá indicar el plan de costos asociados a usos de los componentes en la modalidad que

corresponda.

## Gestión de código fuente

Todo código debe ser incorporado en el repositorio Gitlab del COORDINADOR desde la fase de

desarrollo, por ende, este será el único repositorio válido para el o los proyectos.

Como práctica los desarrolladores deberán realizar al menos un “commit y push” a la rama de

desarrollo de gitlab del Coordinador diariamente, con esto se procederá a tener control del

código y a su vez se procederá de forma automática a la revisión de este por parte de la

herramienta Fortify (7 DevSecOps), con el objetivo de detectar de manera temprana posibles

brechas de seguridad y así ser corregidas por el equipo de desarrollo.

Para el cumplimiento de lo anterior, se deben solicitar a Gestión de Versionamiento el o los

repositorios necesarios junto con los accesos al mismo. A su vez, se debe solicitar la

incorporación del proyecto en la plataforma de revisión de código Fortify.

## Indicaciones adicionales

* Todo producto complementario fuera de los descrito anteriormente deberá ser consultado a COORDINADOR para su aprobación.
* Las versiones indicadas en la tabla de herramientas podrán sufrir cambios siempre y cuando sean en primera instancia las versiones estables y que no impliquen conflictos con los desarrollos ya existentes en el COORDINADOR.
* Para cada proceso de desarrollo de una aplicación se deberá indicar cada una de las herramientas de software a utilizar.
* El diseño de pantallas deberá seguir los estándares que se indiquen en cada propuesta, pudiendo ser estos estándares Web, Windows o Mobile.
* El diseño visual de las aplicaciones web deberá ser adaptable o adaptativo (Responsive Web Design), de manera tal que la apariencia de los desarrollos, se adapten a los dispositivos que se estén utilizando para visualizarlos. Se debe considerar dispositivos con sistemas operativos Android, IOs y Microsoft, como también equipos Smart como celulares y Tablet.
* En caso de integrar las soluciones a los portales corporativos, las dimensiones y componentes de la aplicación deberán ajustarse a las dimensiones destinadas para funcionamiento.
* Las aplicaciones deberán contar con mensajes de alerta o advertencia en la ejecución de procesos de grabación y eliminación de información.
* Deberán contar con manejo de errores para indisponibilidad, denegación de acceso y validación de datos.
* Las aplicaciones deberán poseer un menú de ayuda (Ayuda de Referencia) en todas las pantallas, de manera que sirvan como libros de referencia en línea.
* Todos los proyectos tanto de desarrollo como implementación de software propietarios que tengan interacción directa con usuarios finales deberán tener al menos un análisis de usabilidad contemplando aspectos básicos bajo los principios de Jakob Nielsen.
  + Visibilidad del estado del sistema: el sistema siempre debería mantener informados a los usuarios de lo que está ocurriendo, a través de retroalimentación apropiada dentro de un tiempo razonable.
  + Relación entre el sistema y el mundo real: el sistema debería hablar el lenguaje de los usuarios mediante palabras, frases y conceptos que sean familiares al usuario, más que con términos relacionados con el sistema. Seguir las convenciones del mundo real, haciendo que la información aparezca en un orden natural y lógico.
  + Control y libertad del usuario: hay ocasiones en que los usuarios elegirán las funciones del sistema por error y necesitarán una salida de emergencia claramente marcada para dejar el estado no deseado al que accedieron, sin tener que pasar por una serie de pasos. Se deben apoyar las funciones de deshacer y rehacer.
  + Consistencia y estándares: los usuarios no deberán cuestionarse si acciones, situaciones o palabras diferentes significan en realidad la misma cosa; siga las convenciones establecidas.
  + Prevención de errores: mucho mejor que un buen diseño de mensajes de error es realizar un diseño cuidadoso que prevenga la ocurrencia de problemas.
  + Reconocimiento antes que recuerdo: se deben hacer visibles los objetos, acciones y opciones. El usuario no tendría que recordar la información que se le da en una parte del proceso, para seguir adelante. Las instrucciones para el uso del sistema deben estar a la vista o ser fácilmente recuperables cuando sea necesario.
  + Flexibilidad y eficiencia de uso: la presencia de aceleradores, que no son vistos por los usuarios novatos, puede ofrecer una interacción más rápida a los usuarios expertos que la que el sistema puede proveer a los usuarios de todo tipo. Se debe permitir que los usuarios adapten el sistema para usos frecuentes.
  + Estética y diseño minimalista: los diálogos no deben contener información que es irrelevante o poco usada. Cada unidad extra de información en un diálogo, compite con las unidades de información relevantes y disminuyen su visibilidad relativa.
  + Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores: los mensajes de error se deben entregar en un lenguaje claro y simple, indicando en forma precisa el problema y sugerir una solución constructiva al problema.
  + Ayuda y documentación: incluso en los casos en que el sistema pueda ser usado sin documentación, podría ser necesario ofrecer ayuda y documentación. Dicha información debería ser fácil de buscar, estar enfocada en las tareas del usuario, con una lista concreta de pasos a desarrollar y no ser demasiado extensa.

# Consideraciones generales de desarrollo de software

Las principales consideraciones a los desarrollos de proyectos informáticos corresponden a:

* El desarrollo del software informático debe contemplar una etapa de sincronización o Tuning de la aplicación, donde se deben realizar esfuerzos por disminuir al máximo los tiempos de respuesta, ya sea en querys o consultas y/o cargas de datos, estos tiempos no deben superar los 3 a 5 segundos, en caso contrario se debe declarar al usuario el retardo de la consulta.
* El proyecto de software debe contar con tiempos de respuestas razonables en los procesos y acordes a una aplicación de características Web o Mobile, sin que el usuario final deba esperar tiempos prolongados para obtener el resultado de la información, en caso de que acontezca, la aplicación deberá mostrar mensajes correspondientes al retraso, aduciendo medidas que corrijan la consulta como técnicas de paginación.
* Para validar los tiempos de respuesta, el proveedor deberá entregar los tiempos promedio de respuesta de las acciones que realicen los usuarios sobre la solución, los cuales deberán ser presentados a COORDINADOR, que evaluará y solicitará las mejoras que estime necesarias.
* El software debe poseer programación ordenada y documentada (descripción de funciones, procedimientos y procedimientos almacenados donde corresponda), con el fin de que COORDINADOR pueda realizar modificaciones futuras si así lo requiriera. El código debe ser documentado y referenciar cada proceso.
* El control y mensajería de errores no puede ser críptico o indicar solo codificaciones. La comunicación de errores debe utilizar mensajes genéricos claros y sugerir cursos de acción, además de indicar código y error.
* El software debe poseer interfaces amigables, fáciles de usar y entendibles por el usuario final.
* No se podrá instalar software de ninguna clase en COORDINADOR, a excepción del propio proyecto de software desarrollado.
* La aplicación no debe contar con llamados a programas de administración de Base de Datos (ejemplo: uso de bcp).
* El proveedor será el responsable de la correcta instalación y ejecución del software con el resto de las aplicaciones corporativas existentes en la plataforma de software de COORDINADOR.
* El diseño de la solución deberá ajustarse al nivel de conectividad de su aplicación bajo los protocolos existentes en la red de COORDINADOR, y bajo ningún caso podrá implementar protocolos que no estén ejecutándose en la Red Corporativa.
* Toda clave de acceso utilizada por la aplicación deberá estar encriptada, debiendo el proveedor entregar el algoritmo, programa de encriptación y su correspondiente documentación, la propuesta de modelo de seguridad debe validarse previamente por COORDINADOR.
* El proveedor deberá entregar el plan y método de compilación o despliegue de la solución para que COORDINADOR los valide y ejecute en los respectivos ambientes de Test, Preproducción y Producción.
* El proveedor es responsable de entregar la documentación técnica, de diseño y manuales requerido de un desarrollo.

# Estándares de Interoperabilidad

Todos los proyectos deben considerar en la etapa de diseño la especificación de puntos de integración con las actuales aplicaciones corporativas y departamentales.

Para los puntos de integración definidos, se dispondrá de un catálogo de servicios web que deberán ser integrados a las soluciones que desarrolle el proveedor.

Junto con el catálogo de servicios que dispondrá COORDINADOR, el proveedor podrá integrar al catálogo existente, nuevos servicios que deberán ser especificados en forma y estructura, para su aprobación por parte del COORDINADOR.

El proveedor será responsable de que las aplicaciones que desarrolle operen en forma correcta e integrada con el resto de las aplicaciones corporativas de COORDINADOR.

## Principios SOLID

S – Single Responsibility Principle (SRP)

O – Open/Closed Principle (OCP)

L – Liskov Substitution Principle (LSP)

I – Interface Segregation Principle (ISP)

D – Dependency Inversion Principle (DIP)

Los principios SOLID se encuentran relacionados con los patrones de diseño que nos permiten mantener una alta cohesión y por tanto, un bajo acoplamiento de software.

* **Acoplamiento:** Grado de dependencia que tienen dos unidades de software (clases, subtipos, métodos, módulos, funciones, librería, etc) entre sí.
* **Cohesión:** Grado en que elementos diferentes de un sistema permanecen unidos para alcanzar un mejor resultado que estando separados. Es referente a la forma de agrupar diferentes unidades de software con el objetivo de crear una unidad mayor.

## Arquitectura de Referencia

COORDINADOR cuenta con una arquitectura de referencia definida en 6 capas;

* **Acceso:** La cual indica las herramientas y métodos de autenticación de las aplicaciones.
* **Legados y Herramientas de desarrollo:** Define la capa de sistemas que son parte del ecosistema del COORDINADOR y la definición de herramientas para su desarrollo, como también la utilización de servicios PaaS, SaaS y Aplicaciones World Class.
* **Interoperabilidad e infraestructura como software:** stack de herramientas en cluster que dan soporte a herramientas de integración como también a las aplicaciones como servicios.
* **Bases de Datos:** Definición de motores de bases de datos que están disponibles para el uso.
* **Infraestructura:** Capa de soporte de infraestructura OnPremise y Cloud (AWS).
* **DevSecOps:** Herramientas que soportan los procesos integración y despliegue continuo del COORDINADOR, además de revisión de código sobre prácticas de desarrollo seguro.

|  |  |
| --- | --- |
|  |  |

*Ilustración 1 Arquitectura de Referencia*

## Consideraciones generales de integración

Todos los proyectos deben considerar en la etapa de diseño la especificación de puntos de

integración con las actuales aplicaciones corporativas y departamentales.

Para los puntos de integración definidos, se dispondrá de un catálogo de servicios web que

deberán ser integrados a las soluciones que desarrolle el proveedor.

Junto con el catálogo de servicios que dispondrá COORDINADOR, el proveedor podrá integrar

al catálogo existente, nuevos servicios que deberán ser especificados en forma y estructura, para su aprobación por parte del COORDINADOR.

El proveedor será responsable que las aplicaciones que desarrolle operen en forma correcta e

integrada con el resto de las aplicaciones corporativas de COORDINADOR.

## Estándar CIM

Los sistemas de gestión de energía y más específicamente los sistemas de distribución, cuentan

con numerosos módulos que requieren compartir información de un modelo de datos común. En general, estos modelos son desarrollados por separado y la información utilizada puede requerir de una adaptación.

En este contexto, EPRI (Electric Power Research Institute) estableció un modelo de datos estándar para estos sistemas denominado CIM (Common Information Model). CIM está descrito mediante diagramas de clases y paquetes, cuya composición arquitectónica ha sido ampliamente documentada.

COORDINADOR debe operar bajo CIM, teniendo en consideración que la interoperabilidad entre los diferentes componentes debe ser bajo este estándar sin perder de vista las diferentes

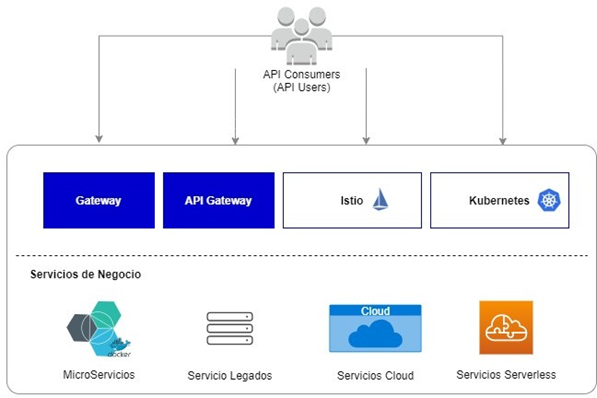
versiones de CIM que los componentes a interoperar puedan tener, siendo la “interoperabilidad” quien se encargue de homologarlas.

## Modelo de Integración

COORDINADOR para la integración de sistemas o intercambio de datos ha establecido que se

debe realizar mediante APIs. Estas deben ser desarrolladas de acuerdo con los estándares

definidos en los puntos de este documento.



### Protocolos SOAP-REST

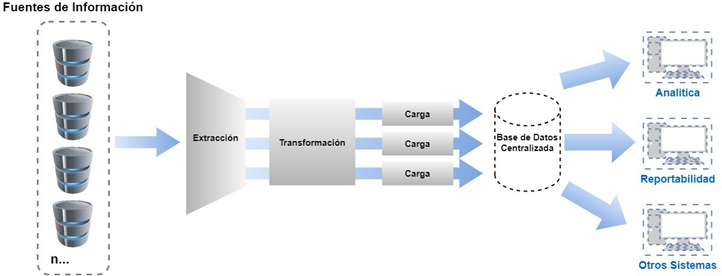
* Para el desarrollo de modelos de integración a través de SOAP se debe documentar la estructura del Web Services Description Language (WSDL), esquema de seguridad, además de contemplar log de transacción y servicios de notificación del servicio.
* Para los desarrollos que requieran orquestación de servicios, se utilizará ESB que el COORDINADOR proveerá y los end-point serán expuestos mediante API Manager correspondientes.
* Los estándares de programación de servicios REST/JSON, se limita al uso de los lineamientos Google JSON Style Guide y su respuesta debe ser un JSON válido.
* Como requerimiento base los tiempos de respuesta la solicitud sea por debajo de los 5 segundos, siendo este un parámetro de resultado e2e (end to end) esto es, el tiempo transcurrido desde la llamada hasta el renderizado en el front-end.
* Los servicios web deben estar protegidos mediante diferentes protocolos.
  + Filtros por lista blanca (whitelist) de IPs.
    - Pueden entregarse los IPs desde dónde será consultado el servicio web.
  + Autenticación APIs, esta debe ser definida por arquitectura al momento del diseño de la solución.
  + OAuth2 – OpenID
    - Todos los servicios expuestos deberán ser securitizados mediante protocolos Oauth2, el cual será provisto por el API Manager respectivo. Por otra parte, en el caso de los microservicios o desarrollos similares que se expongan como servicio, deberán al menos tener un método de seguridad básico (usuario password) definido para cada servicio o para un producto en general. Con lo anterior tenemos dos barreras de seguridad implementadas, la primera de cara al consumidor del api y la segunda de cara al API y los servicios y entre ellos.
    - Autorización mediante token en la cabecera: Bearer token
      * 'Authorization: Bearer 5c53563a5174cf2416ee026847b'
  + API Key
    - i) Al menos se debe considerar la utilización de un API Key provisto por CEN para el consumo de las API’s mediante API Manager.
    - ii) El parámetro para utilizar será user\_key con su respectivo token.
* La respuesta para las salidas XML deben seguir los lineamientos de XML SCHEMA.
  + Las tablas por defecto deberán venir informadas en el elemento <NewDataSet>. En caso de que se utilice otro nombre de elemento debe ser especificado.

## Procesos (ETL)

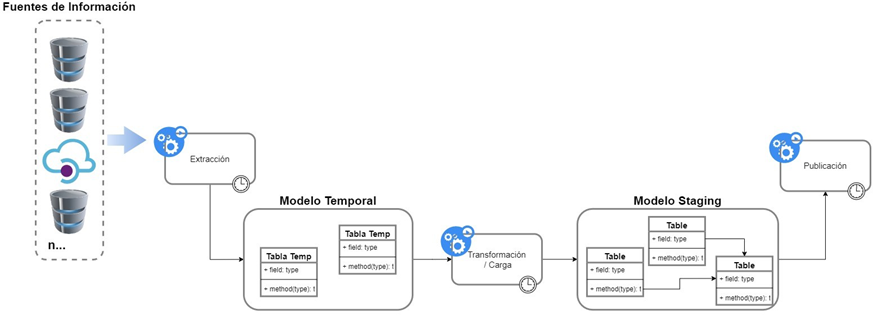
Los procesos ETL (extracción, transformación y carga) se determinarán por los siguientes parámetros:

* Los mensajes a enviar superan los 2 mb
* No son transferencias de información transaccional
* Procesos programables
* Riesgo de sobrecarga del sistema origen y/o destino
* Cantidad de llamados concurrentes

### Diseño

Se deberán considerar dos modelos de diseño de ETL, los cuales contemplan estrategias base para el diseño y construcción de estos.

* ETL Base: Contempla un diseño sin persistencia, este diseño puede ser utilizado cuando las cargas de datos no implican reintentos y las puntas (origen/destino) se encargan de la validación de la información. Por otra parte, la transformación se puede realizar en tiempo real y no requiere de tiempos de procesos diferidos.
* ETL Persistente: Contempla el diseño con persistencia, lo cual permite almacenamiento en un modelo temporal de datos extraídos, con el objetivo de realizar la transformación de manera secuencial.

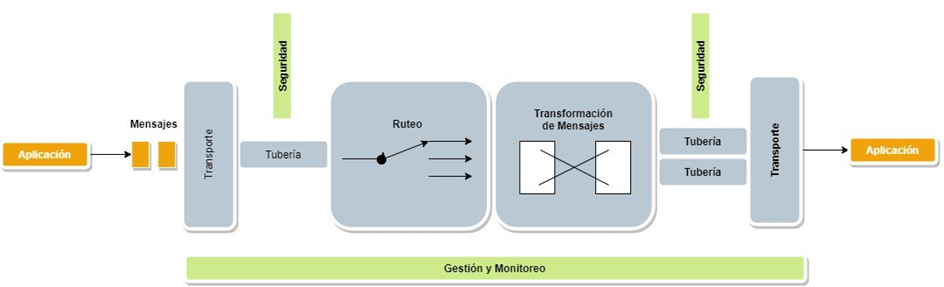


## Orquestación de Servicio

La orquestación de los servicios se deberá realizar teniendo en cuenta dos miradas:

### Microservicios orquestados:

Esto indica que, al momento de desarrollar con arquitectura de microservicios, estos deberán tener la lógica de intercomunicación entre ellos con el objetivo de conformar un sistema integrado desde su génesis.



### Orquestar Microservicios desacoplados:

En el caso que se requiera conformar un “sistema” con microservicios ya existentes pero que no están intercomunicados internamente, por tanto, es necesario orquestarlos mediante métodos externos.

Para el caso de Orquestar Microservicios desacoplados se debe orquestar mediante **Service Bus** (ESB), el cual permite diseñar la comunicación entre los servicios existentes definiendo una entrada y salida en común

## Herramientas de Interoperabilidad

CEN cuenta con la suite de interoperabilidad de Red Hat, considerando como base API Management 3Scale, ESB Fuse, soporte Openshift entre otros.



*Ilustración 2 Arquitectura de Referencia*

# DevSecOps

## Integración Continua:

Todo proyecto deberá estar alineado con integración continua, por tanto, su código fuente debe estar en el repositorio Gitlab del COORDINADOR desde la fase de desarrollo. Y será revisado por especialistas técnicos del COORDINADOR en cualquier momento.

## Revisión de Código:

Se realizarán revisiones de código de forma constante cada vez que se ejecute un “push” a la rama de desarrollo asignada al proyecto.

* 1. Despliegue Continuo:

Para aquellos proyectos que lo permitan, se evaluará la incorporación de métodos DevSecOps, con el objetivo de centralizar, coordinar y en lo posible automatizar las diferentes ambientaciones teniendo en consideración los procesos de Integración Continua y Entrega Continua (CI/CD).

Actualmente se cuenta con Jenkins para la automatización de los pipelines de compilación, revisión y despliegue continuo. Por lo que es parte del proyecto la configuración y construcción del pipeline correspondientes al o los desarrollos. De igual manera debe considerar la documentación necesaria del mismo.

* + 1. Revisión de código:

Se cuenta con Fortify para revisión de código en los aspectos de prácticas de seguridad, los estándares de revisión serán como base los relacionados con CWE TOP 25 Y OWASP TOP 10 (WEB O MOBILE O API O NUBE) dependiendo del proyecto.

# BPM

Todo sistema que contemple flujos de proceso, evaluación de kpi de los estados del proceso, validaciones, notificaciones, etc, deberán ser modelados y desarrollados en la plataforma de BPM Bizagi.

La herramienta BPM debe permitir desarrollos front-end o en el caso que solo se requiera utilización del back-end del aplicativo BPM, deberá exponer servicios Rest (API) para la comunicación entre el aplicativo y el workflow definido. Todos basados en los estándares de este documento.

Los lenguajes de programación en el caso de que el desarrollo sea en la misma herramienta BPM, será la misma que la herramienta Bizagi.

## Entregables BPM

Se deben considerar los diseños técnicos asociados con el producto a configurar indicando de manera detalla:

* Diseños de flujos de procesos
* Diccionarios de datos
* Integraciones
* Diseños de formularios y control de mensajes informativos
* Diseño de interfaces de comunicación

# Estándares para el uso de sistemas y aplicaciones

Todos los desarrollos deben considerar en la etapa de ingeniería de detalle una instancia de validación de su arquitectura de software y modelo de datos en conjunto al COORDINADOR.

Además, deben incluir un dimensionamiento o sizing de los recursos que utilizará el sistema o aplicación, tales como:

* Capacidad de Memoria
* Capacidad de procesadores
* Capacidad de almacenamiento

Debe incluir un dimensionamiento de la Base de Datos e infraestructura donde el sistema residirá, el cual debe considerar la estimación de crecimiento del volumen de información de la base de datos en forma diaria, semanal, mensual y anual, bajo dos aspectos:

* Cantidad de registros
* Almacenamiento físico en disco medido en MB (megabytes).
* VCPU
* VMemoria

El dimensionamiento deberá ser entregado al COORDINADOR en la etapa de validación de la ingeniería de detalle, antes de la solicitud de creación física de la Base de Datos.

* Las tablas de la base de datos deben contar con integridad referencial y éstas deben poseer llaves primarias e índices asociadas para realizar búsquedas óptimas.
* Será creada una cuenta de conexión o acceso a la Base de Datos del proyecto de software, como dbo, con la finalidad de que el proveedor realice labores de administración de la Base de Datos. Se crearán cuentas de usuarios en el motor de Base de Datos, las cuales serán utilizadas para que la aplicación interactúe con el motor, en forma independiente o coincidente con la creación de usuarios y sus respectivos permisos, privilegios, claves y contraseñas de acceso propias de la aplicación.
* El proveedor deberá realizar las optimizaciones (tuning) de la Base de Datos una vez que el proyecto finalice en su desarrollo y antes de que se realice su recepción provisional o final.
* Software de exploración por internet (Browser):
  + Las aplicaciones desarrolladas deben correr indistintamente en los siguientes exploradores.
    - Google Chrome
    - Firefox
    - Safari (Para aplicaciones IOs)
    - Microsoft EDGE

# Consideraciones generales para implementación y/o configuración de plataformas propietarias

* El sistema para implementar debe estar alineado con los requerimientos técnicos transversales indicados en los apartados anteriores.
* El sistema para implementar debe ser configurable en lo funcional con el objetivo de cubrir el o los requerimientos de negocio indicados en las bases.
* En el caso de que se requieran desarrollos adicionales con el objetivo de cumplir con el punto anterior, estos deberán estar alineados con los requerimientos de arquitectura de desarrollos descritos en este documento en los apartados posteriores. Sin embargo, en el caso de que sea necesario realizar cambios, estos deben ser justificados.
* Se debe considerar todo tipo de documentación necesaria para la implementación, operación y uso del sistema a implementar, así como también toda la documentación técnica de configuraciones y/o desarrollos adicionales.
* El proveedor deberá entregar en su propuesta el detalle de la arquitectura de despliegue del sistema a implementar.
* El proveedor deberá entregar en su propuesta el diseño arquitectónico del sistema, identificando puntos de comunicación entre componentes o módulos, además de componentes externos que sean necesarios.
* Todo sistema debe estar integrado a la plataforma de SSO que posea el Coordinador.

# Utilización de repositorios Nexus

pom.xml

<dependency>

<groupId>coordinador</groupId>

<artifactId>core</artifactId>

<version>1.0.1-RELEASE</version>

</dependency>

$HOME/.m2/settings.xml

<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns[="http://maven.apache.org/SETTINGS/1.0.0"](http://maven.apache.org/SETTINGS/1.0.0) [xmlns:xsi="http://www.w3.org/2001/XMLSchema](http://www.w3.org/2001/XMLSchema-instance)-instance" [xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0](http://maven.apache.org/SETTINGS/1.0.0)

https://maven.apache.org/xsd/settings-1.0.0.xsd">

<activeProfiles>

<!--make the profile active all the time -->

<activeProfile>nexus</activeProfile>

</activeProfiles>

<mirrors>

<mirror>

<id>central</id>

<name>central</name>

[<url>http://nexus3](http://nexus3-arquitectura.apps.prod-os-1.coordinador.cl/repository/maven-)-[arquitectura.apps.prod](http://nexus3-arquitectura.apps.prod-os-1.coordinador.cl/repository/maven-)-os-[1.coordinador.cl/repository/maven-](http://nexus3-arquitectura.apps.prod-os-1.coordinador.cl/repository/maven-) group/</url>

<mirrorOf>\*</mirrorOf>

</mirror>

</mirrors></settings>

# Metodologías de Cifrado

Para la integración de certificados en OCP, existen tres metodologías las cuales son soportadas por Red Hat.

## Ruta Edge

Es posible configurar una ruta segura utilizando edge TLS con un certificado personalizado usando el comando **oc create route.** Con una ruta edge, el controlador de ingreso finaliza el cifrado TLS antes de reenviar el tráfico al pod de destino. La ruta específica el certificado TLS y la clave que el controlador de ingreso usa para la ruta.

### Prerrequisitos

* Se debe contar con un par certificado/clave en archivos codificados PEM, donde el certificado sea válido para el host de la ruta.
* Se debe contar con un certificado de CA independiente en un archivo codificado PEM que complete la cadena de certificados.
* Se debe contar con un servicio a exponer.

### Procedimiento

Este procedimiento crea un recurso de ruta con un certificado personalizado y una terminación TLS de borde. Lo siguiente supone que el par certificado / clave está en los archivos tls.crt y tls.key en el directorio de trabajo actual. También puede especificar un certificado CA si es necesario para completar la cadena de certificados. Sustituya los nombres de ruta reales por tls.crt, tls.key y (opcionalmente) ca.crt. Sustituya el nombre del servicio que desea exponer para frontend. Sustituya el nombre de host adecuado por www.example.com.

* Cree un recurso de ruta seguro utilizando la terminación TLS de borde y un certificado personalizado.
* $ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=www.example.com
* Si examina el recurso de ruta resultante, debería tener un aspecto similar al siguiente:
* Definición de YAML de la ruta segura

| apiVersion: v1  kind: Route  metadata:  name: frontend  spec:  host: www.example.com  to:  kind: Service  name: frontend  tls:  termination: edge  key: |-  -----BEGIN PRIVATE KEY-----  [...]  -----END PRIVATE KEY-----  certificate: |-  -----BEGIN CERTIFICATE-----  [...]  -----END CERTIFICATE-----  caCertificate: |-  -----BEGIN CERTIFICATE-----  [...]  -----END CERTIFICATE----- |
| --- |

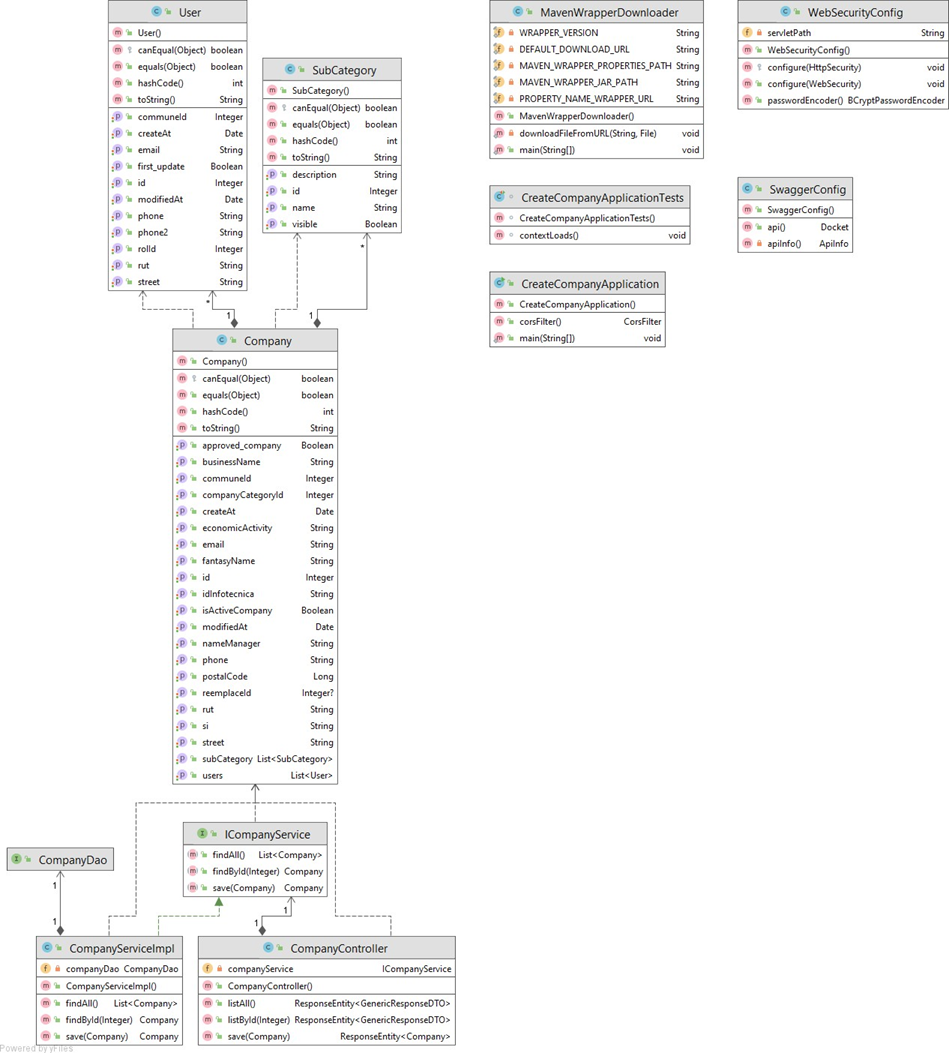
See oc create route edge --help for more options.

# Ejemplo de Diseños

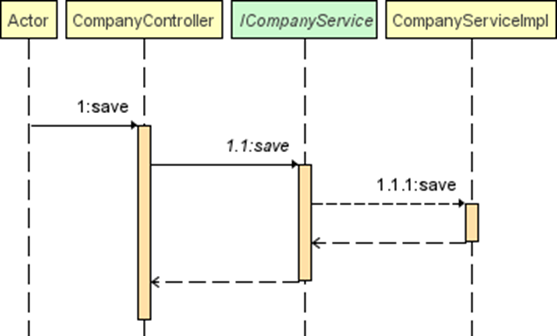
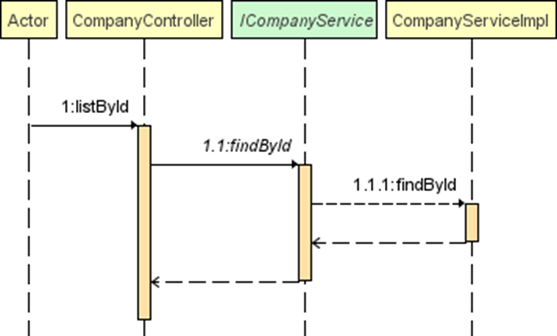
## Diagrama de Componentes

# 

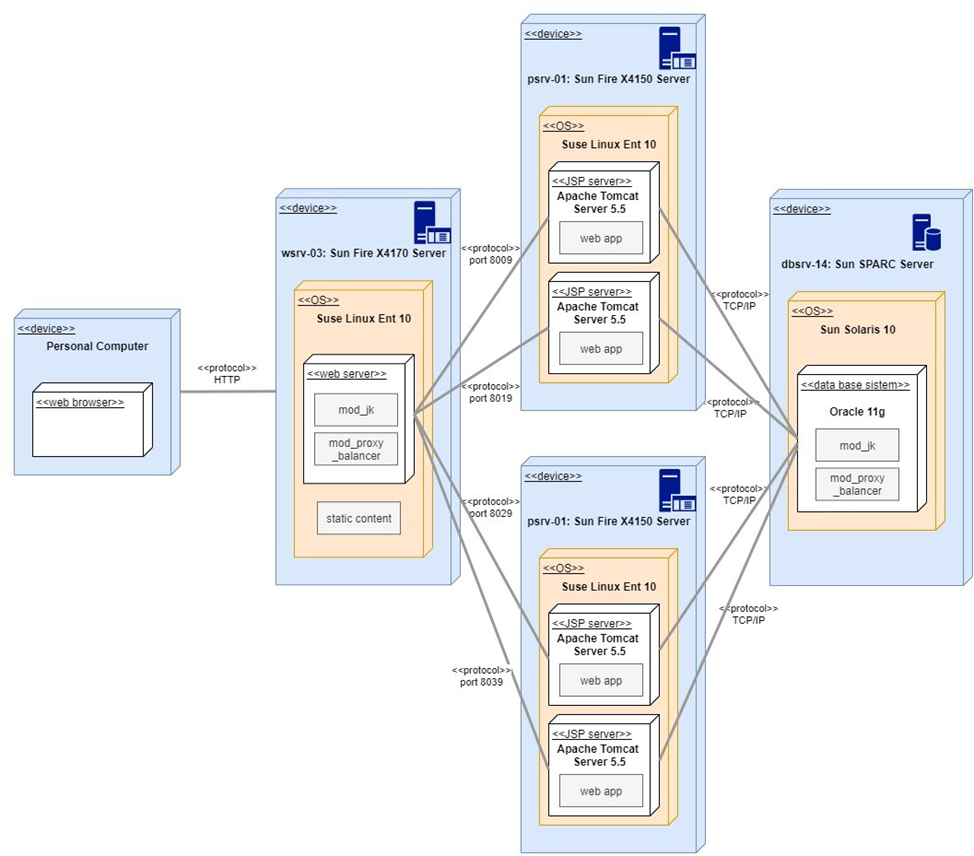
## Diagrama de Clases



## Diagrama de Secuencia

**

## Diagrama de Despliegue



## **Fase de Desarrollo**

Esta fase corresponde a todas las actividades cuyo resultado aporta directamente a la implementación de la aplicación, en específico, su escritura y despliegue de código en entornos controlados (dev/qa), en esta fase se obtienen los artefactos desplegables que darán vida a la funcionalidad comprometida.

Lo importante de esta fase yace en las consideraciones que se deben abordar para implementar el DRP desde el inicio, esta consideración es la siguiente. Uno de los pilares de la estratégia de DRP es el de poder desplegar aplicaciones en múltiples sitios, para lograr esto se ha definido las siguientes reglas para los los componentes que se materializan en Openshift mediante un *DeploymentConfig* puedan ser desplegados en distintos clusters Openshift 4 (independiente de su versión minor):

1. Las aplicaciones deben contar con un repositorio GIT.
2. Las aplicaciones deben ser compiladas mediante una estrategia *Source-To-Image*[[1]](#footnote-0) y su *BuildConfig* debe ser parte de los artefactos a subir al repositorio GIT.
3. Cada aplicación deberá contar con una carpeta -llamada por convención “**openshift**” - con todos los yamls que permitan su correcto despliegue en Openshift. Estos yaml pueden contener cualquier tipo de contenido soportado por Openshift/Kubernetes. Un despliegue típico debería contener generalmente los siguientes:
   1. DeploymentConfig
   2. BuildConfig
   3. ConfigMap
   4. Secrets (solo de entornos no productivo)
   5. Service
   6. Route
   7. ImageStream
   8. PersistenVolumeClaim
   9. StatefulSet

Un aspecto crítico de la anterior convención, es que el contenido de los archivos YAML que se almacenen en dicha carpeta solo deben contener **una definición de objeto**, por lo que se debe guardar precaución con aquellas definiciones multivaluadas que son del tipo **List,**  estas no deben ser utilizadas tanto porque no existe una necesidad específica para ello y los componentes que acceden a estos archivos no están preparados para un componente del tipo **List.** Cada archivo debe tener una sola definición de objeto, el nombre de archivo deberá ser constituido únicamente por una combinación de minúsculas y guiones medios finalizando con una extensión de archivo del tipo “.yaml” o “.yml”

Con las reglas anteriores se estará aplicando **Infraestructura como código[[2]](#footnote-1)**, por lo que cualquier cambio a los componentes de Openshift deberá, necesariamente, partir por un cambio a los YAML del código fuente para hacerlo persistente. Si se realizan cambios de forma directa a Openshift, procure incorporar estos al repositorio GIT o de lo contrario se sobrescribirán cuando se realicen un despliegue en las fase de despliegue.

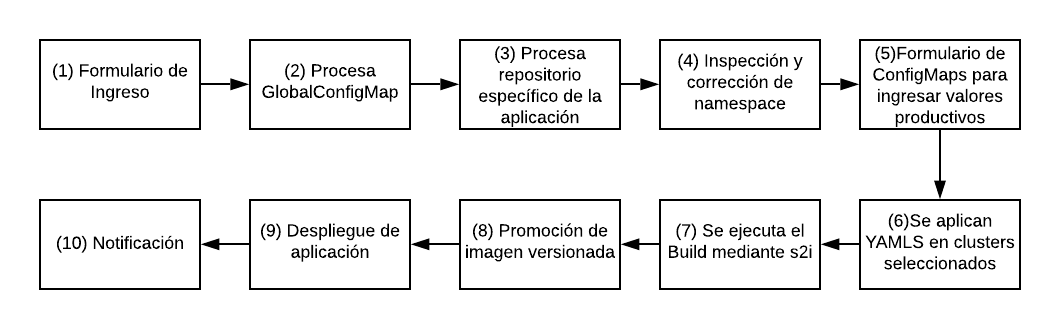
## **Fase de Despliegue**

Esta fase entra en acción cuando ya se ha tomado la decisión de realizar un despliegue en producción. Esta fase está fuertemente relacionada con Jenkins y sus configuraciones realizadas para desplegar aplicaciones tanto en el cluster Openshift ubicado en ESC como en Azure.

En esta parte del DRP, se construyeron Pipelines haciendo uso de utilitarios oficiales de Red Hat[[3]](#footnote-2) que facilitan la labor de comunicación con distintos clusters y que además permiten leer la carpeta convenida en la fase de desarrollo, por lo que es crucial que la definición de objetos allí almacenada esté de acuerdo con la forma en la que la aplicación se debe desplegar.

Para hacer efectiva esta fase, se requiere que el equipo responsable de los despliegues en producción ingrese a Jenkins y gatille el trabajo de despliegue ejecutando el pipeline correspondiente a la aplicación que se desea llevar a producción. Luego de esta acción, el pipeline se ejecutará según la siguiente pauta:

1. Se muestra un formulario para solicitar la información del despliegue a realizar
2. Se procesa en primera instancia el repositorio global de ConfigMap para cargar los YAML que allí estén almacenados
3. Se procesa el repositorio específico de la aplicación que se está desplegando de la siguiente forma
   1. Clona el repositorio
   2. Procesa los YAML ubicados en la carpeta **Openshift** del repositorio
   3. Agrega metadata correspondiente a producción a los YAMLS procesados
   4. Ajusta los objetos de tipo BuildConfig eliminado triggers, agregando credenciales genéricas de GIT entre otros aspectos
   5. Ajusta los Objetos de tipo DeploymentConfig quitando los triggers y poniendo en pausa los despliegues
4. Se inspeccionan los namespaces correspondientes y ajustan los parámetros y labels correspondientes de cada uno, para estar alineados a las definiciones de entornos productivos.
5. Se presenta un formulario con el contenido original del repositorio (es decir **NO PRODUCTIVO**) de cada objeto del tipo ConfigMap, con el fin de actualizar los datos con los productivos
6. Se aplican los artefactos antes recolectados en los clusters seleccionados
7. Se ejecutan los BuildConfigs detectados desde el repositorio
8. Se espera la confirmación exitosa de la finalización de la compilación
9. Se promueve la imagen en el mismo namespace productivo de forma que el despliegue de la imagen quede versionada
10. Se despliega la aplicación usando la imagen versionada
11. Se notifica al correo el resultado de la operación

**Diagrama de flujo del pipeline**

**NOTA: Se debe considerar que los pasos de despliegue se realizan en Serie para ambos clusters**

Al finalizar esta fase, la aplicación se encontrará desplegada en los clusters correspondientes de acuerdo a los parámetros ingresados en el formulario y al código YAML almacenado en conjunto con la aplicación.

## **Fase de Ejecución**

Esta fase depende única y enteramente de las particularidades de cada aplicación, existen algunas condiciones que sugieren ejecutar procedimientos mientras la aplicación se encuentra dando pleno servicio a sus usuarios.

## **Fase de Failover**

Ha ocurrido un desastre relacionado a caída de base de datos o pérdida del sitio principal, por lo que es necesario ejecutar los procedimientos de DRP definidos por cada aplicación.

Se deben considerar que cada aplicación podría estar involucrada con alguno de los siguientes tópicos al momento de sobrellevar un desastre:

1. Modificaciones al stack de redes tales como cambios de Pool en los balanceadores
2. Ejecutar playbooks en Ansible Tower que realicen cambios a nivel de Openshift principalmente
3. Ejecutar los procedimientos establecidos para conmutar los repositorios de datos en caso del escenario de falla de base de datos
4. Ejecutar los procedimientos establecidos para conmutar componentes externos

En esta fase del DRP resulta altamente crítico que las tareas anteriores a esta fase hayan sido ejecutadas sin problemas. Esto significa que las aplicaciones deberían estar desplegadas en ambos sitios, con sus VIPS configuradas, consumiendo recursos externos mediante DNS+Virtual Server (según corresponda), con los procedimientos de sincronización ejecutados.

## **Fase de Failback**

Se ha superado la condición de desastre y es factible recuperar los servicios en los clusters principales, para eso se deben seguir los procedimientos definidos por cada aplicación, para restablecer la funcionalidad en los cluster que corresponden. Al finalizar la ejecución de esta Fase, la aplicación debería quedar en el estado regular de ejecución, tal como estaba antes de la aparición del evento que provocó la falla.

Algunos aspectos típicos de este failback son los siguientes:

1. Re-establecer pool en F5
2. Re-configurar aplicaciones para habilitar el regreso
3. Sincronizar datos entre el clúster secundario y el primario

El siguiente capítulo busca compartir un listado de tópicos a partir del entendimiento del equipo de Red Hat que implementó el proyecto respecto de los aspectos que se pueden explorar, reforzar, mejorar y con ello los próximos pasos para lograr la maduración de las prácticas de DRP y CD.

1. Mejorar el conocimiento y desarrollo empírico de las aplicaciones
2. Continuar estandarizando el proceso de despliegue de las aplicaciones
3. Empoderarse de los artefactos técnicos pudiendo agregar y/o ajustar funcionalidades tanto al pipeline como a los playbooks
4. Automatizar el proceso de creación de aplicaciones, como por ejemplo:
   1. Creación automática los espacios en bitbucket
   2. Creación automática los espacios en Openshift
   3. Creación automática los espacios en Jenkins con Pipelines
5. Mejorar la visibilidad del capacity planning de Openshift
6. Recolectar feedback continuo entre los equipos responsable de las aplicaciones (dev y ops) para mejorar no tan solo la funcionalidad de negocio, sino las diferentes fases que permiten a que el negocio se facilite (como despliegue, calidad de código, y en general mejora continua)
7. Explorar diferentes posibilidades y opciones de comunicación/automatización para las aprobaciones del pipelines
8. Establecer un roadmap para todos los artefactos entregados durante este proyecto, para que se expanda su aplicación en toda la organización.

# 

### Metadata generada por el usuario

Este tipo de metadata es necesaria para modificar ciertos aspectos en la ejecución del pipeline. Esta información debe ser incorporada por el desarrollador y debe estar en los Artefactos YAML que correspondan, a continuación se detallan los annotations permitidos por tipo de artefactos YAML:

* **ConfigMap:**
  + multisite-config.jenkins.devops.cclh.io/override-values: Esta annotation indica al Pipeline que advierta y solicite al usuario atributos dentro de un ConfigMap cuyos valores son diferentes entre clusters. Los valores permitido son textos separados por coma, por ejemplo:
    - multisite-config.jenkins.devops.coordinador.io/override-values: ‘valor1,esto.es.una.llave,esto-es-otra-llave’

Con este ejemplo, el pipeline indica al usuario que el ConfigMap debe poseer valores distintos dependiendo del clusters y que los valores que se debe tener especial atención son: 1) valor1, 2) esto.es.una.llave y 3) esto-es-otra-llave. Ante esto, el pipeline en esta versión presentará 2 formularios, el operador deberá completar el yaml completo, con los valores que correspondan en los formularios que correspondan.

* **Route:**
  + custom-hostname-route-config.jenkins.devops.coordinador.io/override-values: Esta annotation indica al pipeline el valor productivo a utilizar cuando se realice el despliegue en Producción. Reemplaza el valor original del yaml por el definido en esta anotación, fundamentalmente debido a que se acordó que los valores funcionales[[4]](#footnote-3) de los yamls que se suban al repositorio deberían ser todos no-productivo, este annotation da la oportunidad al desarrollador/operador de poder definirlo honrando el acuerdo anterior. Cuando una ruta exista en el repositorio, este valor es REQUERIDO, su ausencia provocará un error en el pipeline. Por otro lado, este pipeline no está preparado para rutas cuyos certificados SSL se encuentren en el YAML explícitamente, debido a que al momento del desarrollo no existía ninguna ruta con estas características por lo que no fue necesario programarlo.

### Metadata generada por el pipeline

Por otro lado, el pipeline incrusta anotaciones para entregar información sobre el último Job de Jenkins que modificó el artefacto. A continuación los significados de la metadata que se incrusta en todos los objetos sujetos a manipulación en pipelines Jenkins:

1. [jenkins.devops.coordinador.io/id](http://jenkins.devops.cclh.io/id): Contiene la ID del Build Number asignado por Jenkins, es un correlativo que se incrementa por cada ejecución de pipeline realizada.
2. [jenkins.devops.coordinador.io/build](http://jenkins.devops.cclh.io/build): Link directo al build al que manipuiló este objeto
3. [jenkins.devops.coordinador.io/job](http://jenkins.devops.cclh.io/job): Link directo al JOB específico que se está ejecutando
4. [jenkins.devops.coordinador.io/logs](http://jenkins.devops.cclh.io/logs): Link directo a la salida (log) del pipeline

Adicionalmente, para las rutas, se agrega específicamente esta annotation

1. jenkins.devops.coordinador.io/original-route-from-sourcecode: Este annotation aparece cuando se ha sobreescrito la ruta original (no productiva) con la final (productiva), sirve para tener un registro de la ruta que originalmente estaba en los repositorios antes de su reemplazo automático.

## 

1. https://docs.openshift.com/container-platform/4.3/builds/build-strategies.html#images-create-s2i\_build-strategies [↑](#footnote-ref-0)
2. https://www.redhat.com/en/blog/helping-you-get-infrastructure-code [↑](#footnote-ref-1)
3. https://plugins.jenkins.io/openshift-client/ [↑](#footnote-ref-2)
4. Que tengan un significado operativo para openshift [↑](#footnote-ref-3)