**Guidance Microservicios sobre EKS**

**Overview Microservices sobre EKS**

El desarrollo de microservicios implica considerar los **principios básicos** para **diseño**de microservicios:

* Altamente mantenibles y probables (testing).
* Débilmente acoplados.
* Desplegables de forma independiente.
* Organizado en torno a capacidades empresariales (Usando dominios de negocio).
* Propiedad de un pequeño equipo. (You build it, you run it).

La arquitectura de **microservicios**permite la **entrega rápida, frecuente y confiable de aplicaciones grandes y complejas.** También permite que una organización evolucione su pila de tecnología.

Para cumplir con estos principios, **Itaú**apalanca los **microservicios sobre Kubernetes** como plataforma; *Kubernetes* es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios.  Kubernetes facilita la automatización y la configuración declarativa (manifiestos, código, cli, apis). Tiene un ecosistema grande y en rápido crecimiento. Existen múltiples vendors en la industria de TI que ofrecen su producto out of the box de Kubernetes. Para**Itaú Chile**, usamos el Kubernetes de **AWS**: **EKS (Elastic Kubernetes Services).**

En este documento declararemos las **buenas prácticas y definiciones**, que**permitirán mantener la integridad del diseño de apps con estilo de arquitectura de microservicios. Evitando vendor lock-in y manteniendo la portabilidad de sus aplicaciones.**

Microservice Architecture Style:  
Reference: https://microservices.io

**Buenas practicas**

**Establezcamos las buenas prácticas que nos guiarán a mantener la consistencia con el diseño de microservicios:**

**Evitar Vendor Lock-in:**

* Evitar el uso de SKDs, Librerías, APIs, u otra interfaz que consuma directamente servicios externos al Kubernetes, para evitar acoplamiento entre componentes, que eviten la portabilidad de aplicaciones a diferentes clusters, regiones, cuentas y/o cloud vendors como Azure o Google.  
  **Ejemplo:**En vez de usar los SDKs de AWS secret manager para ir a obtener secretos, usamos secretos nativos de AWS, e integramos el EKS con Secret manager usando AWS Secret Manager CSI Driver. Permitiendonos más adelante cambiar el proveedor de secretos, y no acoplar una app/microservicio directamente con una herramienta de Vault Credentials.

**Consistencia**

* Uso de los pipelines CI/CD para despliegue o modificación de cualquier recurso sobre los cluster de EKS. Permitiendo la consistencia entre los fuentes y lo desplegado.

Para mantener alineados al diseño de la arquitectura de microservicios se establecen algunas definiciones a continuación.

**Convenciones utilizadas en este guidance.**

Las palabras clave del nivel de requisitos "**DEBEN**", "**NO DEBEN**", "**REQUERIDOS**", "**DEBERÁN**", "**NO DEBERÁN**", "**DEBERÍAN**", "**NO DEBERÍAN**", "**RECOMENDADOS**", "**PUEDEN**" y "**OPCIONALES**" utilizadas en este documento (sin distinción entre mayúsculas y minúsculas) deben interpretarse como se describe en el estandar [RFC 2119](https://www.ietf.org/rfc/rfc2119.txt).

**1- Contenerización de Apps**

**Repositorio de imágenes**

En el banco usamos el repo de imagenes de JFrog, que se encuentra gobernado por el equipo de devops. Y posee cargada las imagenes base permitidas en nuestra organización para la construcción de microservicios.

**Imágenes base**

|  |  |  |  |
| --- | --- | --- | --- |
| **Framework** | **Lenguaje** | **Imagen Base usada** | **Caso de uso** |
| * .net 6 * .net 5 | * C# 10 * C# 9 | * mcr.microsoft.com/dotnet/aspnet:6.0 * mcr.microsoft.com/dotnet/aspnet:5.0 (Obsoleto) | BFF, Microservicio |
| * Nodejs con Express | * Typescript | * node:14-alpine | BFF, Microservicio |
| * SpringBoot\* | * Java | * PENDIENTE POR DEFINIR | BFFs de New APP \* |

**2- Recursos de Kubernetes**

Kubernetes disponibiliza diferentes tipos de recursos declarativos que permiten a la infraestructura orquestar el estado deseado de una aplicación. Se describen las consideraciones sobre las definiciones de cada uno de los recursos a continuación:

**Namespaces**

Los namespaces, permiten agrupar-aislar recursos por equipos distribuidos. Permitiendo lógica y físicamente(en algunos casos) permitir o restringir el uso de recursos con políticas de accesos RBAC, uso de recursos límites con Resource Quotas, y visibilidad entre componentes.

* Para Outer/BFFs y fronts los namespaces **DEBEN**ser por aplicación (Segun registro formal en el inventario de aplicaciones).
* Para Inner los Namespaces **DEBEN**ser por BusinessDomain. Basandose en BIAN.

**Deployments**

Los deployments proporcionan las definiciones del estado deseado del despliegue, a traves de los replica sets del contenedor.

**Replicas**

Para aplicaciones que requieran alta disponibilidad se recomienda establecer un **minimo de 2 replicas**. Que le indicarán a Kubernetes que debe tener 2 Pods en state running. Adicionalmente, pueden establecer las replicas de escalamiento con HPA.

**Imagenes**

El origen de las imagenes solo es posible establecerlos usando el repositorio aprobado y gobernado por el equipo de devops. (JFrog). Por lo que las imagenes son descargadas post-build desde el repositorio de imagenes del pipeline de microservicios.

**Recursos**

Se han definido los siguientes recursos base para soportar cada pod.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tecnología** | **Requests** | | **Limits** | |
| Memoria (Mb) | CPU (milicores) | Memoria (Mb) | CPU (milicores) |
| TypeScript | 128 | 100 | 512 | 300 |
| Net Core | 74 | ​​​​​​​100 | 479 | 270 |

En caso de requerir aumentar los recursos de Pods por Deployment, las comunidades deberán en base a benchmarks y/o pruebas de cargas establecer los recursos request/limits. Entendiendo y considerando el aumento de costos y disminuyendo el tamaño de recursos definidos para su namespace.

En caso de requerir aumento de recursos asignados en namespace (By Resource Quotas), deberán previamente haber realizado una evaluación y contar con la aprobación de FinOps con presupuesto aprobado para el aumento de recursos.

**Services**

Los servicios son balanceadores layer 4 (TCP/UDP), de cargas de trabajo que pueden disponibilizar servicios internamente en el cluster, para ser expuestos por un ingress.

Considerar los siguientes puntos:

* Se debe usar ClusterIP, como definición.

**Ingress**

Los ingress son las capas de exposición (Layer 7) de las APIs vía HTTP. Un Ingress permite la exposición y el enrutamiento de trafico hacia las cargas de trabajo necesarias. Para soportar esta capacidad el Cluster, debe poseer un Ingress Controller.

Es necesario considerar los siguientes puntos:

* Usamos AWS ALB Ingress Controller. Que permite la creación de ALB (Balanceadores layer 7).
* Solo exponer tráfico HTTPs.
* Siempre usar DNS de Itaú. Nunca usar los DNS autoasignados por AWS. Estos pueden cambiar e impactar a los consumidores, creando indisponibilidad y requiriendo esfuerzo extra para resolver la problemática de cambiar los endpoints de exposición e invocación.
* Se debe asignar el ingressClassName según el nivel de exposición del MS, por defecto, es interno.
* No existe por defecto un Ingress Class que permita exposición de servicios a internet. En caso de tener una solución con este requerimiento, se debe analizar con Arquitectura de Infraestructura y Arquitectura de seguridad, las capas de exposición, volumetria, y seguridad a aplicar (WAF, CDN, NetworkFirewall, entre otras).

**Ingress Class:**

Todos los ingressClass respetan las IngressClass (diferentes instancias de ingress), se implementan Ingress Controller según el nivel de exposición:

|  |  |  |  |
| --- | --- | --- | --- |
| **IngressClassName** | **Provider** | **Annotation Requires** | **Uso** |
| ingress-internal | AWS Ingress Controller | **annotations**:  alb.ingress.kubernetes.io/target-type: 'ip' alb.ingress.kubernetes.io/group.name: **inner-bcl** | Para Ingress dentro del EKS transversal de BCL |
| ingress-internal | AWS Ingress Controller | **annotations**: alb.ingress.kubernetes.io/target-type: 'ip' alb.ingress.kubernetes.io/group.name: **outer-bff** | Para Ingress dentro del EKS transversal de BFF |

**Host&DNS**

Cada ingress tiene la responsabilidad de enrutar hacia los diferentes servicios. Para evitar problemas de mantenibilidad entre rutas "basepath", en el banco cada microservicio es enrutado por su nombre, asignando un nombre en los DNS internos del banco.

Lineamientos para exposición de ingress usando registros de DNS:

* No se debe usar en los ingress y/o capas de exposición en balanceadores, usando los dns autogenerados por los ALB/NLB. Los autogenerados pueden cambiar en la recreación, o movimiento de cargas.
* Todo los ingress DEBEN de tener un host Itau, Itauchile o clouditauchile definido. A continuación la nomenclatura que se debe seguir:  
  La nomenclatura para registro host en DNS:

**Desarrollo**  
{Nombre MS}-{ProjectIDGitLab}.bff-dev-ms.clouditauchile.cl**​​​​​​​**  
{Nombre MS}-{ProjectIDGitLab}.bcl-dev-ms.clouditauchile.cl  
{Nombre-MF}-{ProjectIDGitLab}.mf-dev-ms.clouditauchile.cl

**QA**  
{Nombre MS}-{ProjectIDGitLab}.bff-qa-ms.clouditauchile.cl  
{Nombre MS}-{ProjectIDGitLab}.bcl-qa-ms.clouditauchile.cl  
{Nombre-MF}-{ProjectIDGitLab}.mf-qa-ms.clouditauchile.cl

**Producción**  
{Nombre MS}-{ProjectIDGitLab}.bff-prd-ms.clouditauchile.cl  
{Nombre MS}-{ProjectIDGitLab}.bcl-prd-ms.clouditauchile.cl  
{Nombre-MF}-{ProjectIDGitLab}.mf-prd-ms.clouditauchile.cl

**3- Seguridad, Autenticación, Encriptación de Trafico con TLS, Secretos, API Keys**

Cada microservicio debe ser disponibilizado de forma segura con TLS para evitar ataques de man-in-the-middle, invocaciones no permitidas, o permisos de applicaciones para el consumo de APIs/Microservicios. Se describen a continuación las consideraciones para el desarrollo de microservicios:

1. La exposición de microservicios siempre debe realizarse a través de API Connect. Que posee como requisitos minimos, exposición TLS para encriptación de data en transito, autenticación entre aplicaciones vía API Keys, segmentación por redes y analítica.
2. Los microservicios no son finalizadores de trafico TLS. Minimizando la carga y el uso de recursos de hardware sobre los pods.

**APIKeys**

El flujo de creación de api Keys de API connect está automatizado por el equipo de devops a traves de un pipeline, que crea la aplicación en API Connect, y luego registra los secrets con esta nomenclatura en el secret manager de accesos. La estructura con la que se escriben estos secrets en el secret manager es un json con el esquema:

{

"clientid": "4n4b1kf15d48w",

"clientsecret": "4322ygug23yu4gvghvhg2v34g234442151"

}

**Secrets**

Para el uso de secretos (api keys, conectionstrings, BD passwords) se debe realizar la implementación de [secrets de k8s](https://kubernetes.io/es/docs/concepts/configuration/secret/" \o "https://kubernetes.io/es/docs/concepts/configuration/secret/" \t "_blank), que son recursos que permiten tener una colección de llave valor, con N secretos. Estos secretos son sincronizados segun la definición de un [SecretProviderClass](https://github.com/aws/secrets-store-csi-driver-provider-aws" \l "secretproviderclass-options" \o "https://github.com/aws/secrets-store-csi-driver-provider-aws#secretproviderclass-options" \t "_blank)por el [Kubernetes secret CSI driver](https://aws.amazon.com/blogs/security/how-to-use-aws-secrets-configuration-provider-with-kubernetes-secrets-store-csi-driver/" \o "https://aws.amazon.com/blogs/security/how-to-use-aws-secrets-configuration-provider-with-kubernetes-secrets-store-csi-driver/" \t "_blank).

Considerar los siguientes puntos para el uso de secretos dentro de sus deployments:

1. La relación entre secretos y deployments/microservicios debe ser contemplado usando el principio de minimo privilegio. Por lo que cada deployment, debe tener:   
   1 SecretProviderClassObject - 1 Secret - N Keys as Secrets | 1 Service Account - 1 Role IAM - 1 Politica IAM - N Secretos.  
   [Ver demo](http://gitlab.itauchile.cl/architecture-center-of-excellence/workshops/masa-use-case/microservicios-bcl/currency-exchange/currency-exchange-management) y "Anexo 1. Relación de recursos de desarrollo con IAM y Secrets." para obtener mayor detalle.
2. El secreto ya debe existir previamente para que kubernetes pueda levantar el deployment con los secrets de los que depende.
3. Para usar secretos, se crea debe un  [SecretProviderClass object](https://secrets-store-csi-driver.sigs.k8s.io/getting-started/usage.html" \o "https://secrets-store-csi-driver.sigs.k8s.io/getting-started/usage.html" \t "_blank) donde se mapea n los secretos a los que requieren ser inyectados como variables de entorno y montados como volumen.
4. Los secretos, son almacenados en las cuentas de custodia (diferentes a las de desarrollo o donde están los EKS), y se registran usando la siguiente nomenclatura según su tipo.
5. Las key de los secretos no deberán contener caracteres especiales, y se deberá usar lowerCamelCase.

**Nomenclaturas de registro de secretos en el secret manager**

Cada tipo de secreto tiene una nomenclatura definida, se especifica a continuación:

**API Keys**

APIKey/{qa, prod}/{org}/{catalogo}/{applicationName}​​​​​​​

**Passwords**

Password/dev/Filenet/fncbrkflt  
Password/qa/Filenet/fncbrkflt  
Password/prod/Filenet/fncbrkflt

**Proveedores**

Password/{ambiente}/ext/{nombreproveedor}/{nombre usuario}

**Ejemplos:**  
Password/desa/ext/digevo/17011778  
Password/qa/ext/digevo/192376123  
Password/prod/ext/digevo/28871887

**TrustStore Certificate de CA interna Itaú**

La firma de certificados internos para dominios itauchile siempre es realizada con una CA interna/privada de Itaú Chile. Por lo que se deben agregar a las imágenes de contenedores, dentro del docker file las CA de Itachile, de la siguiente forma:

**mcr.microsoft.com/dotnet/aspnet:​​​​​​**

1

​

2

ADD http://gitlab.itauchile.cl/architecture-center-of-excellence/api-connect/certificates/-/raw/main/itauchile/CAPrivate.crt "/usr/share/ca-certificates/itauchile/CAPrivate.crt"

3

RUN echo itauchile/CAPrivate.crt >> /etc/ca-certificates.conf && update-ca-certificates

4

​

**node:14-alpine:**

1

​

2

RUN apk add openssl

3

WORKDIR /usr/local/share/ca-certificates/

4

​

5

ADD http://gitlab.itauchile.cl/architecture-center-of-excellence/api-connect/certificates/-/raw/main/itauchile/CAPrivate.crt "/usr/local/share/ca-certificates/CAPrivate.crt"

6

RUN openssl x509 -in /usr/local/share/ca-certificates/CAPrivate.crt -out /usr/local/share/ca-certificates/ca.pem

7

RUN update-ca-certificates

Por defecto, NodeJs, usa su propio TrustStore, para configurar NodeJs y las librerias importadas, como Axios, deberán configurar en las variables de entorno la opción de Node para usar las CA del SO:

1

​

2

- name: NODE\_OPTIONS

3

     value: "--use-openssl-ca"

4

​

**4- Integración de MS con ecosistema de APIs**

Capacidades de Observability, Seguridad, Reusabilidad de la malla de aplicaciones y servicios:

* En capas de **exposición**de**Outer las aplicaciones consumidoras**en API Connect**DEBEN ser las aplicaciones por Canales.** (Portal, NewApp, ItauPass, Rappi, Itú, entre otras, usando el nombre de aplicaciones registradas en el inventario de aplicaciones del banco).
* En **capas de exposición internas**(Inner/ACL, Inner/BLC, Inner/Providers), se **DEBEN crear/usar aplicaciones**en API Connect **consumidoras por comunidad.  
  ​​​​​​​**Ejemplo: Fraude, Inversiones, Rappi, Cuenta Digital.

**5 - Automatización**

Para automatizar la creación de los manifiestos de kubernetes, se debe usar HELM, declarando los valores del Chart de despliegue, que permitirán al pipeline, usar helm para despliegue de todos los componentes relacionados al despliegue.

[Usar las definiciones de devops, para la declaración de values](https://itauchile.sharepoint.com/sites/CoEDevSecOpsPblicoITA/Documentos%20compartidos/Forms/AllItems.aspx?ct=1685127133454&or=Teams-HL&ga=1&id=/sites/CoEDevSecOpsPblicoITA/Documentos%20compartidos/Capacitaciones/Capacitaci%C3%B3n%20Automatizaci%C3%B3n%20Microservicios%20sobre%20EKS%20Transversal&viewid=2b11f74e-d771-485e-9efe-1db7658ef90e).

**6 - Chassis Pattern**

El patron de chassis permite realizar un "template" de codigo fuente con las definiciones base de una aplicación para que un proyecto pueda copiar y comenzar a desarrollar un nuevo microservicio, sin necesidad de reescribir estás definiciones:

**Loggin**

* Se debe implementar un patron de logs, independientemente del tipo de tecnología, framework o lenguaje. Esto permite realizar explotación y trazabilidad de la aplicación desde las herramientas de monitoreo (Dynatrace, Cloudwatch).
* El formato de logs debe contemplar:

|  |  |  |  |
| --- | --- | --- | --- |
| **Pregunta** | **Campo** | **Descripción** | **Ejemplo** |
| Cuando | timestamp | El timestamp en formato ISO8589 | 2022-09-12T15:48:39.000-4:00 |
| Quien | transactionId | Identificador del valor único por cada petición generado desde la aplicación consumidora (ej: aplicación de canal). Se debe traspasar a las cadenas de componentes subsiguientes. | db1b0c549e804e53a2198cc2a6b76d63 |
| Quien | spanId | Identificador del ID que representa donde se encuentra la llamada en toda la traza. | fc85aab146bf41d6baca410789c41eda |
| Quien | parentId | Identificador del invocador del servicio. En multiples llamadas, el spanId de la llamada previa se convierte en parentId | a0cb1ee5e8e6450cab5121551ed68a4c |
| Que | severity | Corresponde al nivel o Level del log | INFO |
| Donde | env | Ambiente donde se ejecuta el microservicio que loguea (ej: dev, qa, prod) | dev |
| Quien | consumer | Código con identificador de la aplicación consumidora. | 11 |
| Quien | component | Nombre y versión del componente que loguea | Cardpayment:1.0.0 |
| Quien | type | Tipo del componente (ej: bff, bcl) | bcl |
| Quien | method | Nombre del método | pay |
| Que | message | Mensaje del log | Pago realizado |
| Que | payload | Paylod de la operación | {} |
| Que | responseTime | Tiempo de respuesta en milisegundos | 100 |

Ejemplo de un registro:

1

{ "timestamp": "2022-09-12T15:48:39.000-4:00", "correlationId": "db1b0c549e804e53a2198cc2a6b76d63", "transactionId": "fc85aab146bf41d6baca410789c41eda", "spanId": "a0cb1ee5e8e6450cab5121551ed68a4c", "severity": "INFO", "env": "dev", "consumer": "11", "component": "cardpayment:1.0.0", "type": "bcl", "method": "pay", "message": "Pago realizado", “responseTime”: “100”, "payload": “{}” }

El sentido de la traza distribuída es lograr mantener un registro (trazabilidad de una petición)

​​​​​​​**Healthcheck**

* Se debe implementar patron de health, exponiendo un endpoint en***{host}/health****,*que indique con HttpStatus codes el status de salud del contenedor.

**OpenAPI**

* Las Aplicaciones/MS, deben de exponer sus contratos de API auto-generados, documentados bajo OpenAPI 2.0.

**Exception Handler**

* Manejo de errores no controlados, para loguear y manejar http status code estandar que mapee, filtre las excepciones.

**Autenticación/Seguridad**

* Por seguridad los MS deben considerar la autorización de invocaciones por medio de flujos de autenticación de clientes. Para esto es necesario entender el caso de uso, y establecer cual es el IDP (Identity Provider) que se usará. Para Clientes Itaú PingId, para Colaboradores el Azure AD.
* La exposición de los MS se realiza por HTTP dentro del cluster, pero para exponer fuera del cluster se deben usar IngressControllers que actuan como finalizadores de trafico TLS.
* En caso de MS, que requieran ser finalizadores de trafico por limitantes (PCI compliance u otro), se debe evaluar una arquitectura que permita la administración del ciclo de vida de los certificados y llaves dentro de los contenedores.

**7 - Versionamiento semantico**

* La compilación de imagenes se versiona usando versionamiento semantico. (SemVer 2.0.0)
* Dado un número de versión MAYOR.MINIOR.PATCH, se incrementa:  
  - La versión MAYOR cuando realizas un cambio incompatible en la API.   
  - La versión MENOR cuando añades funcionalidad compatible con versiones anteriores.  
  - La versión PARCHE cuando reparas errores compatibles con versiones anteriores.

Siguiendo el versionamiento semantico, el ciclo de vida de MS y sus APIs, establece las siguientes reglas para el versionamiento:

* MAYOR. Deja obsoleto a la versión anterior, con fecha de retiro planificada, todos los consumidores deberán modificar su implementación para consumir nueva versión.
* MENOR. Actualización sobre el mismo componente, sin modificación del endpoint. Los consumidores no deberán realizar cambios. Update in place.
* PARCHE. Actualización sobre el mismo componente, sin modificación del endpoint. Los consumidores no deberán realizar cambios. Update in place.

El **versionamiento semantico**, es **usado**para establecer en la URI la**ruta de acceso según la versión de un deployment**.

* Se usa el ***Mayor Version***para establecer la ruta de acceso en el base path de enrutamiento del ingress. Ejemplos
  + {ingress host}/**v1**/{path}
  + {ingress host}/**v2**/{path}
  + {ingress host}/**v3**/{path}

**8 - Convención de nombre para proyectos de microservicios en Gitlab**

* ​​​​​​​Todos los componentes del tipo microservicios, definidos por banco Itaú, deben encontrarse dispuestos bajo el grupo padre "microservicios-api" de Gitlab Itauchile.
* La nomenclatura usada para especificar repositorios y grupos de Gitlab debe ser kebab-case (minusculas-separadas-por-guion).
* **La tercera posición de izquierda a derecha, especifica el namespace de kubernetes/eks, donde se desplegarán los recursos.**
* La ruta de proyectos sigue el siguiente template:

**microservicios-api/<tipo-de-microservicio>/<namespace>/…/<proyecto-ms>**

A partir de este punto, y tomando como base los tipos de microservicios definidos por banco Itaú, se debe identificar el **<tipo-de-microservicio>** a desarrollar, entre los cuales se encuentran:

|  |
| --- |
| [**Tipo de Microservicios**](https://itauchile.sharepoint.com/sites/CenterOfExcellence/SitePages/Arquitectura-MASA-Ita%C3%BA.aspx#%3a~%3atext=nuestras%20Inner%20API.-%2cBCL%2c-Los%20servicios%20en) |
| Microservicios-bff |
| Microservicios-bcl |
| Microservicios-bcl-itu |
| Microservicios-microfronts |

Dependiendo del **tipo-de-microservicio**, se debe establecer el path, de acuerdo a lo siguiente:​​​​​​​

**8.1 Microservicios-fronts**

1

+ {aplicacion}

2

  - {proyecto-module-app}

Path: **microservicios-api/microservicios-microfronts/<aplicación>/<{proyecto-ms}>**​​​​​​​

* **{aplicación}** corresponde al nombre de la aplicación registrada en el inventario de aplicaciones del banco Itaú. Ejemplos: Superapp, Portal, DBS Portal, DBS Onboarding, KYC.  
  NOTA: La aplicación es tomada por el pipeline para generar el namespace en EKS donde se desplegarán los artefactos.
* **{proyecto-ms}**corresponde a los proyectos que componen la aplicación, en microfronts, separados por nombres de capacidad de negocio. Ejemplos: Credit Card Management, Consumer Loans, Account Management, AML and Compliance.

**NOTA:**En caso de no existir registro de la aplicación en el inventario de aplicaciones , se debe generar el registro , el cual DEBE ser revisado y acordado con el arquitecto de soluciones y arquitectura empresarial.

**8.2 Microservicios-bff:**

1

 + {aplicación}

2

 - {proyecto-module-app}

Path: **microservicios-api/microservicios-bff/<aplicación>/<{proyecto-ms}>**

* **{aplicación}** corresponde al nombre de la aplicación registrada en el inventario de aplicaciones del banco Itaú. Ejemplos: Superapp, Portal, DBS Portal, DBS Onboarding, KYC.  
  NOTA: La aplicación es tomada por el pipeline para generar el namespace en EKS donde se desplegarán los artefactos.
* **{proyecto-ms}**corresponde a los proyectos que componen los componentes de la aplicación, el nombre es orientado al Producto/Servicio.

**8.3 Microservicios-bcl:**

1

+ {business-domain}

2

+ {capability}

3

  - {proyecto-microservices-service-domain-ref-bian}

Path: **microservicios-api/microservicios-bcl/<{business-domain}>/<{capability}>/<proyecto-ms>**

* **{Business-domain}** corresponde al nombre del Business Domain, gobernado por el equipo de Arquitectura empresarial (Se usa como referencia los Business Domain de BIAN). En caso de no existir, aplicar procedimiento de registro de Business Domain y capacidades con el equipo de EA.  
  **NOTA:** El business domain es tomado por el pipeline para generar el namespace en EKS donde se desplegarán los artefactos.
* **{Capability}** o service domain, corresponde a la capacidad que estarán soportando los microservicios. Se deben usar los aprobados por EA, en caso de no existir, aplicar procedimiento para registro de capacidades. Ejemplos: Credit Card Management, Consumer Loans, Account Management, AML and Compliance.
* **{proyecto-ms}**corresponde a los proyectos que componen los componentes de la aplicación, el nombre es orientado al Producto/Servicio.

**Ejemplos:**

* microservicios-api/microservicios-bcl/**cross-channel**/**party-authentication**/**onboarding-auth.git**
* microservicios-api/microservicios-bcl/**consumer-services**/**currency-exchange**/**coin-bridge.git**
* microservicios-api/microservicios-bcl/**party**/**customer-profile**/**customer-insights.git**

**Casuísticas presentes en repositorio:**

* microservicios-api/microservicios-bcl/**consumer-services**/**currency-exchange**/**currency-exchange.git**
* microservicios-api/microservicios-bcl/**document-library.git**

**Forma correcta de manejar repositorio:**

* microservicios-api/microservicios-bcl/**consumer-services**/**currency-exchange**/**currency-exchange.git**
* microservicios-api/microservicios-bcl/**consumer-services**/**currency-exchange**/**coin-bridge.git**
* microservicios-api/microservicios-bcl/**consumer-services**/**currency-exchange**/**cash-mate.git**
* microservicios-api/microservicios-bcl/**document-management-and-archive**/**document-library**/**document-library.git**

**Observación:**existen algunos microservicios bcl que pueden representar la capacidad/service domain  establecida/o, por ejemplo currency-exchange en ese caso la estructura del repositorio debe contener la capacidad/service domaine duplicado como indica el siguiente ejemplo:​​​​​​​

* microservicios-api/microservicios-bcl/**consumer-services**/**currency-exchange**/**currency-exchange.git**

Forma correcta de manejar repositorio:

* microservicios-api/microservicios-bcl/**consumer-services** /**currency-exchange** /**currency-exchange.git**
* microservicios-api/microservicios-bcl/**consumer-services**/**currency-exchange**/**coin-bridge.git**
* microservicios-api/microservicios-bcl/**consumer-services**/**currency-exchange**/**cash-mate.git**

De esta manera se permite que otros microservicios relacionados a la misma capacidad/service domain puedan incorporarse a la estructura de repo que lo contiene.