Java
Microservicios

Ing. Rafael Benedettelli

Rafael Benedettelli

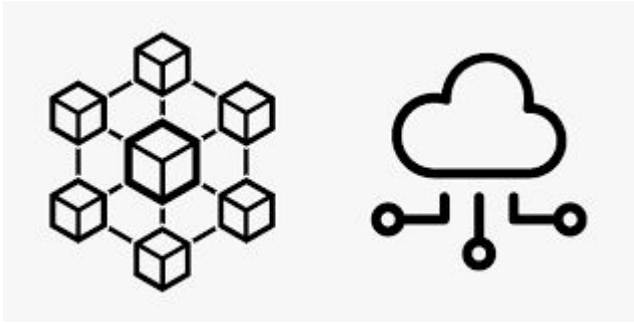
https://www.linkedin.com/in/rafael-benedettelli-34080113/

- Ingeniero en sistemas
- Arquitecto de software

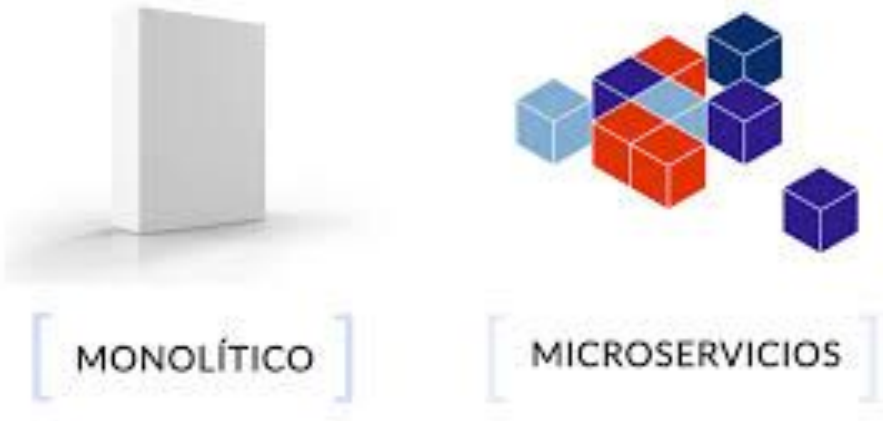
Java 11, JEE, Spring, Microservicios, Maven, Gradle, Eureka, Ribbon, Feign, git, log4j, postgresql, mongo, jongo, Rest, Swagger, Angular, CSS, LESS, C, PHP, Bash, Alfresco...

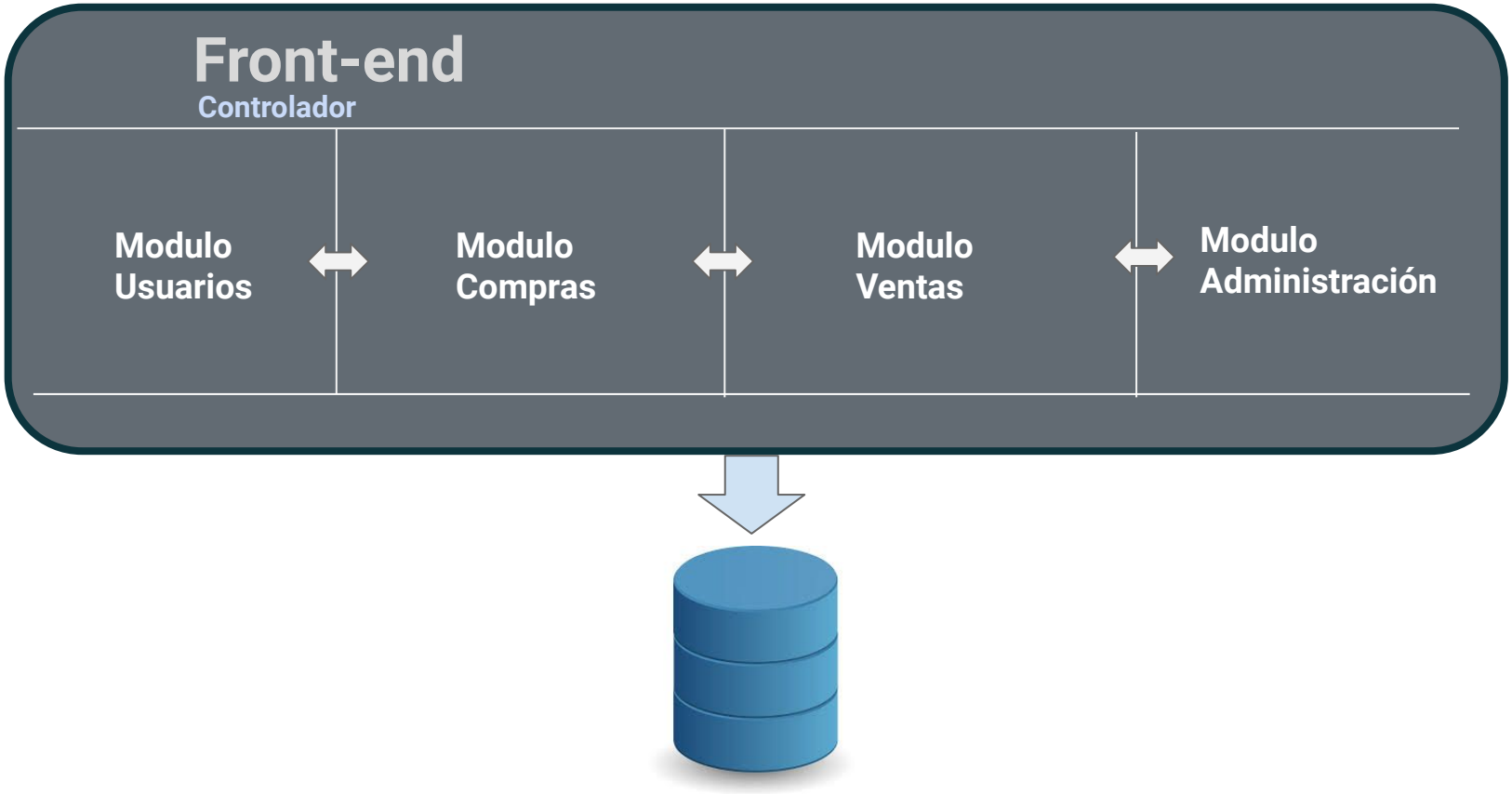
1. Arquitectura orientada a microservicios
2. Integración y despliegue continuo
3. Java (Situación actual) - de 8 a 14
4. Spring BOOT
5. API REST CRUD
6. Postman Consumo de servicios
7. Lombok (Generación de código)
8. Externalización de parámetros
9. Taller de ejercicios

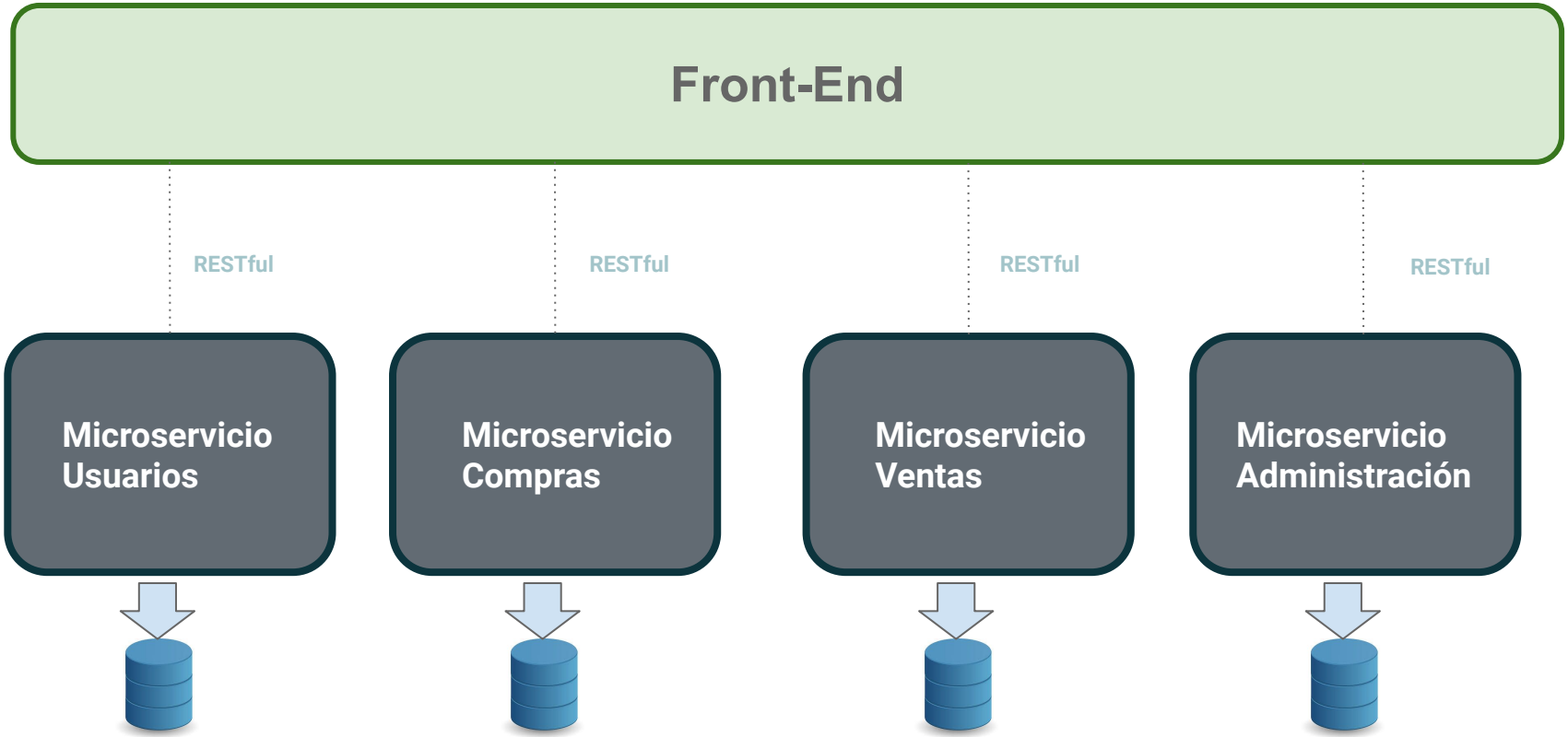
Arquitectura orientada a Microservicios

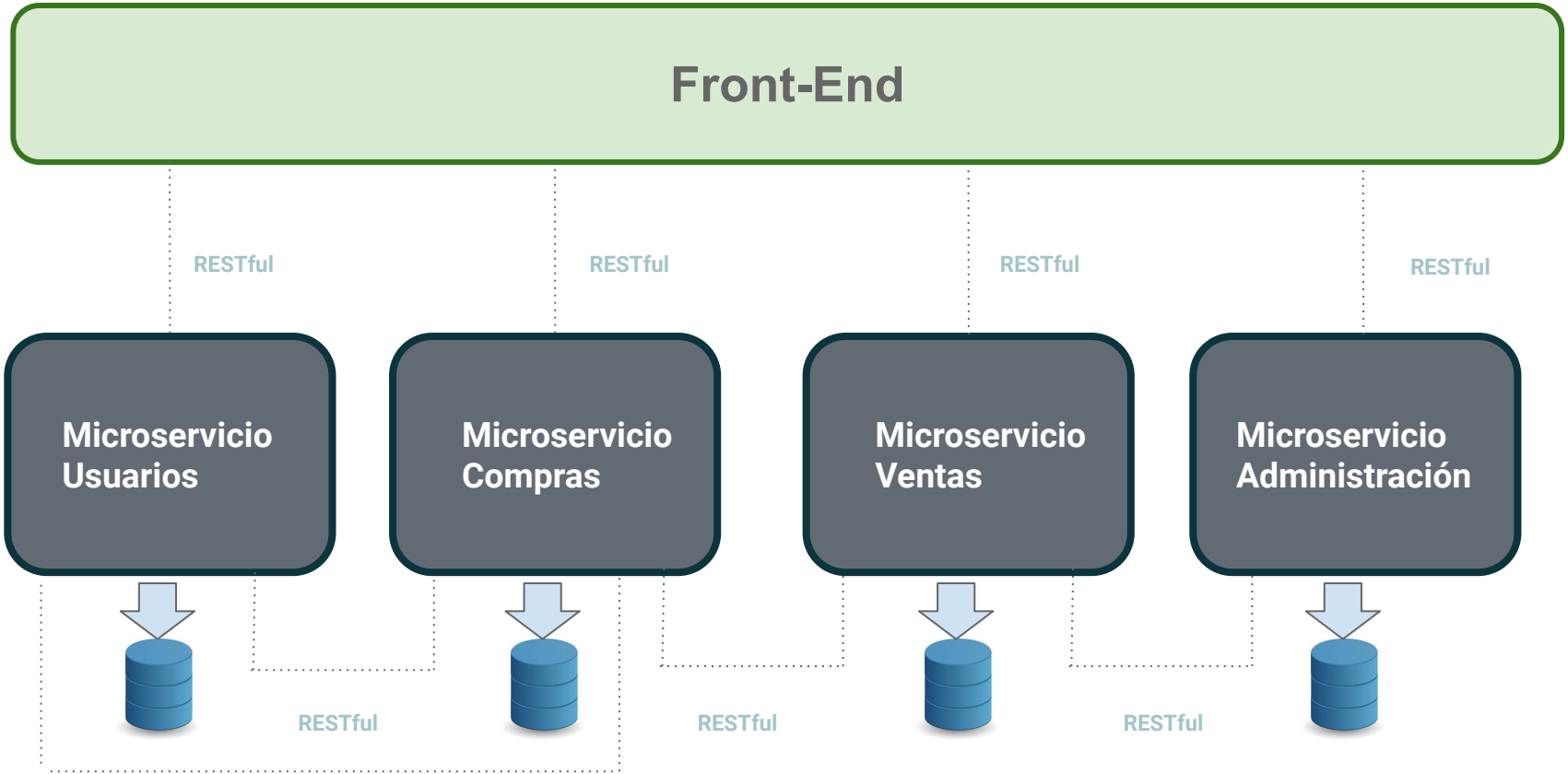


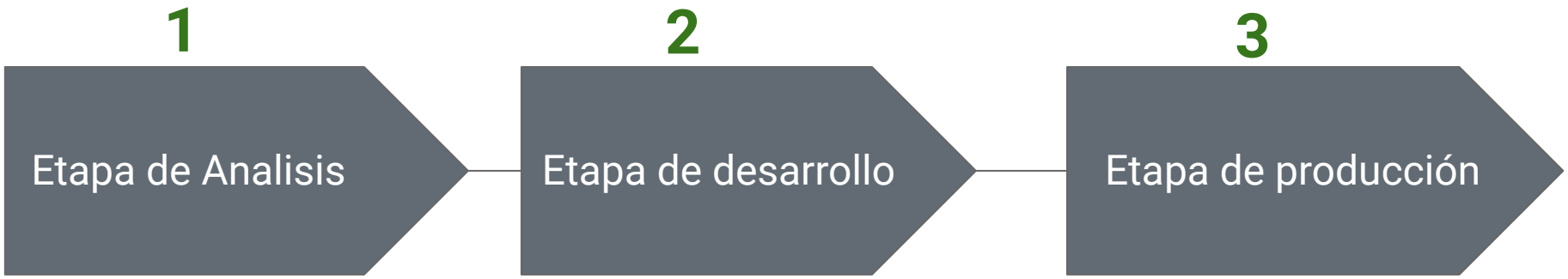
Arquitectura orientada a Microservicios




Front-end
Controlador
Modulo Usuarios
Modulo Compras
Modulo Ventas
Modulo Administración


Front-End
RESTful
RESTful
RESTful
RESTful
Microservicio Usuarios
Microservicio Compras
Microservicio Ventas
Microservicio Administración

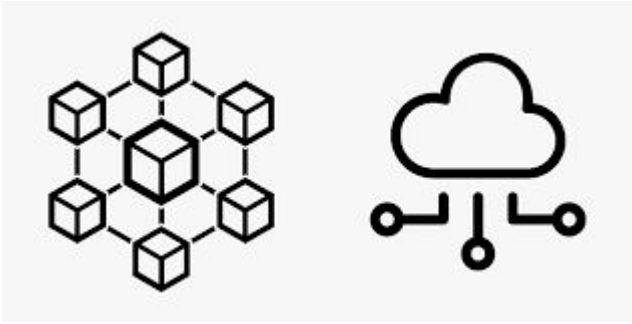

Front-End
RESTful
RESTful
RESTful
RESTful
Microservicio Usuarios
Microservicio Compras
Microservicio Ventas
Microservicio Administración
RESTful
RESTful
RESTful

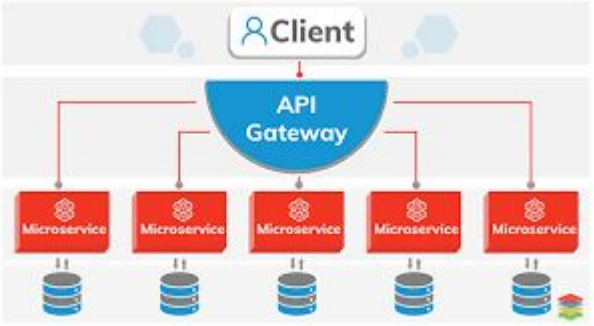

1
Etapa de Analisis
2
Etapa de desarrollo
3
Etapa de producción

Arquitectura orientada a Microservicios

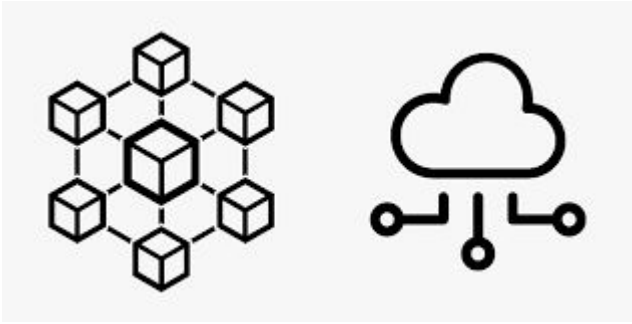
Diseño y arquitectura

- Separación por unidades logicas
- Reduce notablemente el **acoplamiento**
- Reduce el impacto de los fallos en un microservicios en la solucion general
- La comunicación entre microservicios es a travez de protocolos ligeros (Rest, Message Broker)





Arquitectura orientada a Microservicios



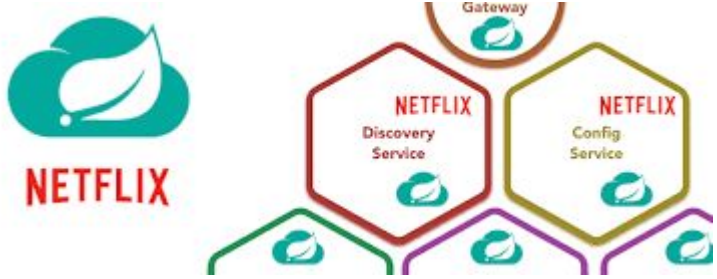
Desarrollo

División de roles por microservicio
Enfoque en el microservicio
Agnostico al lenguaje

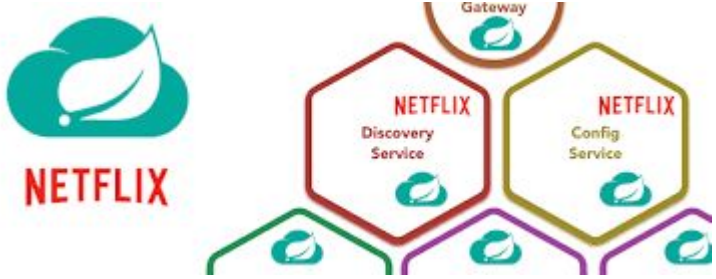
Despliegue

Mayor complejidad de despliegue
Mayor flexibilidad en entorno operativo
Asignación de instancias/recursos por concurrencia

GitLab ♡ docker

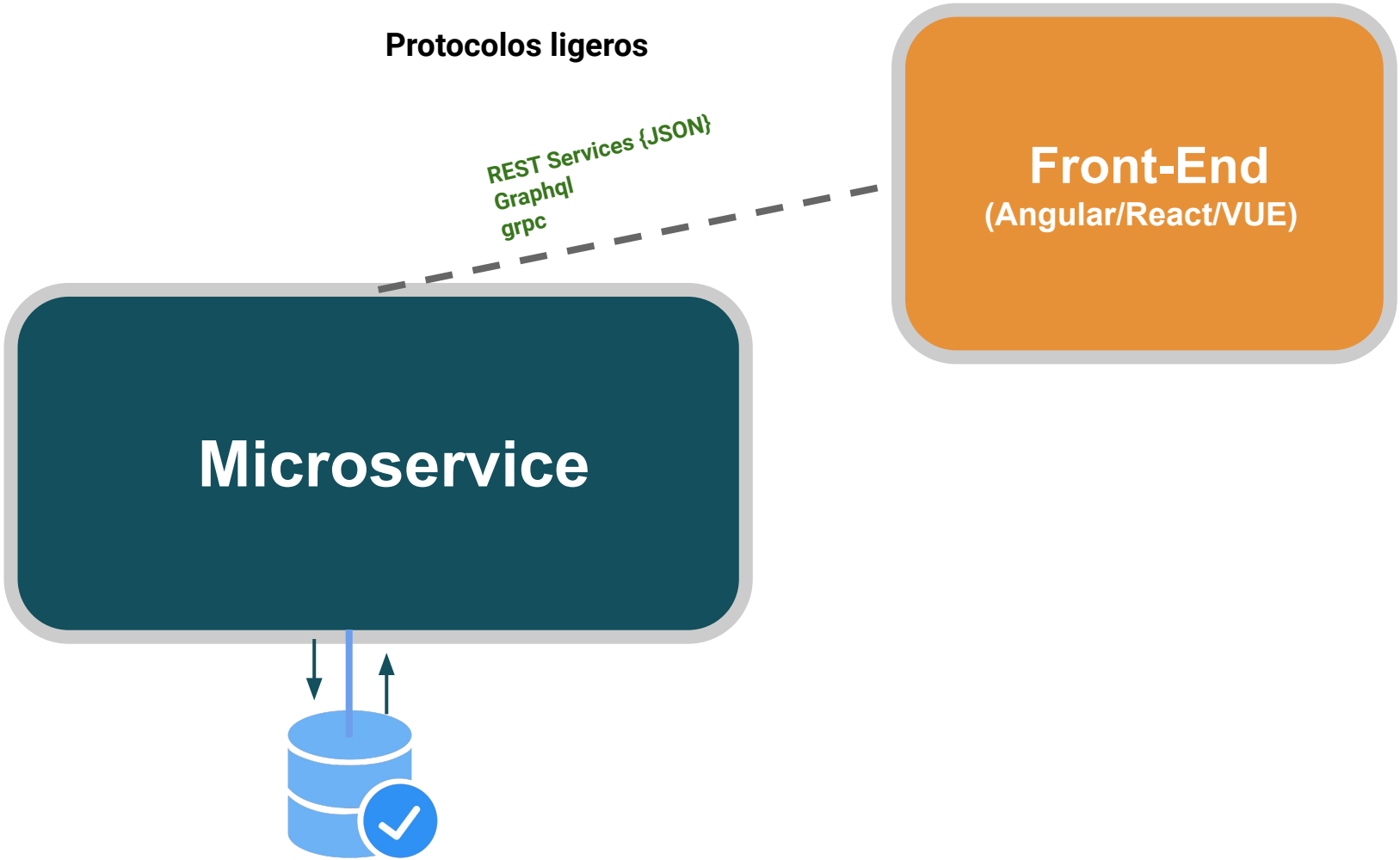


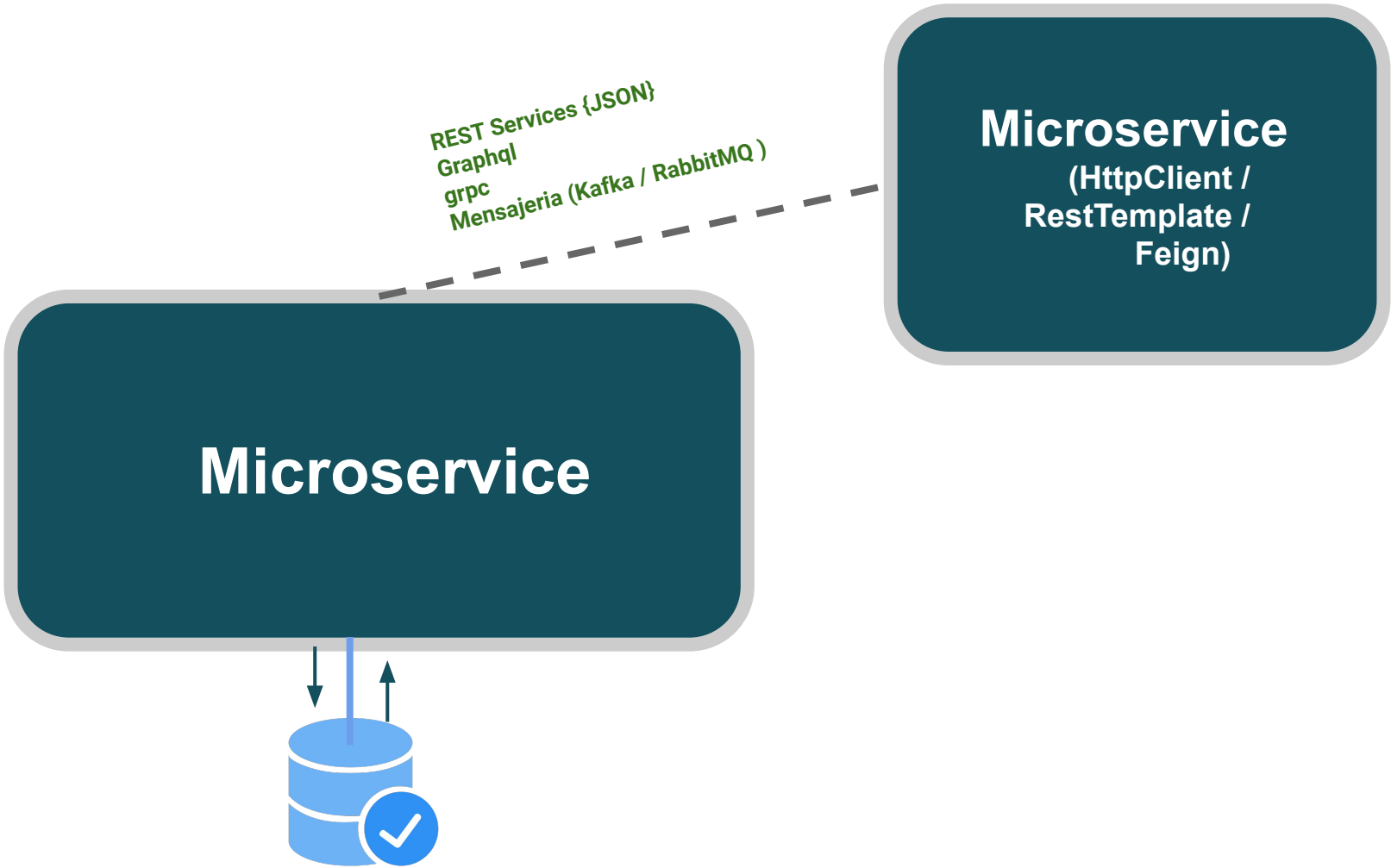
Portabilidad y Reutilizacion



GitLab ♡ docker

Arquitectura orientada a Microservicios

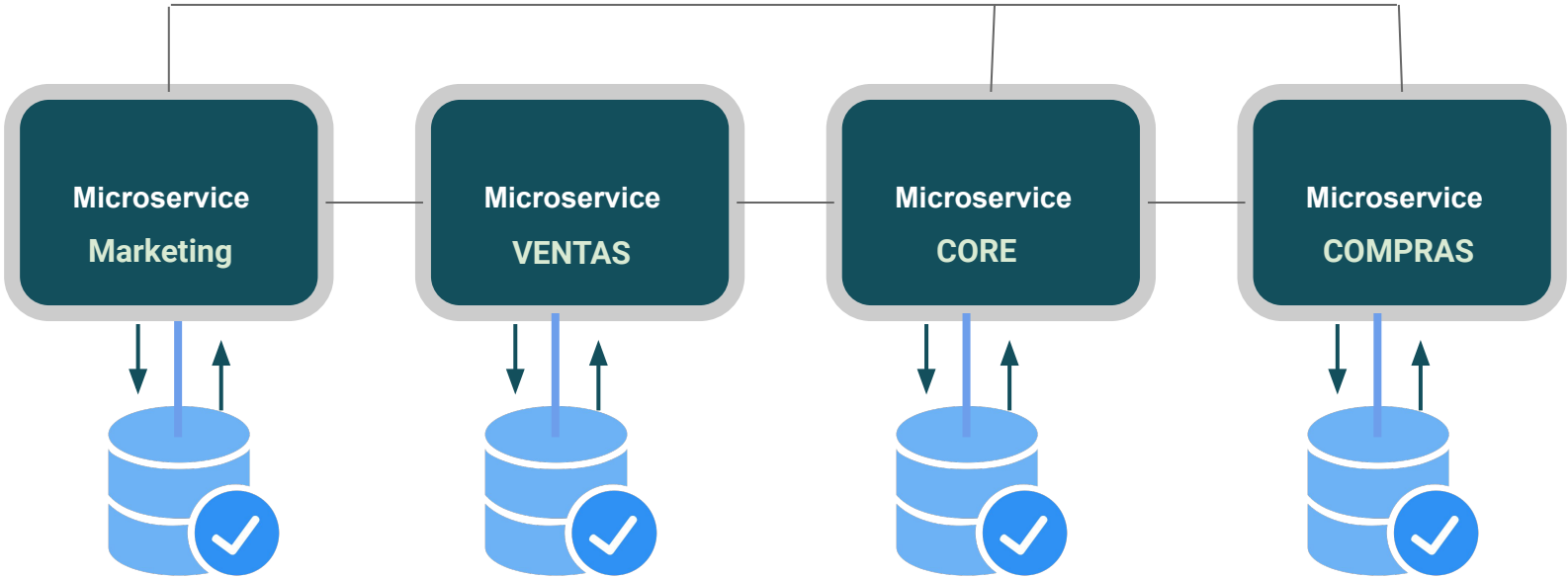

Protocolos ligeros
REST Services {JSON}
Graphql
grpc
Front-End
(Angular/React/VUE)
Microservice


REST Services {JSON}
Graphql
grpc
Mensajeria (Kafka / RabbitMQ)
Microservice
(HttpClient /
RestTemplate /
Feign)
Microservice

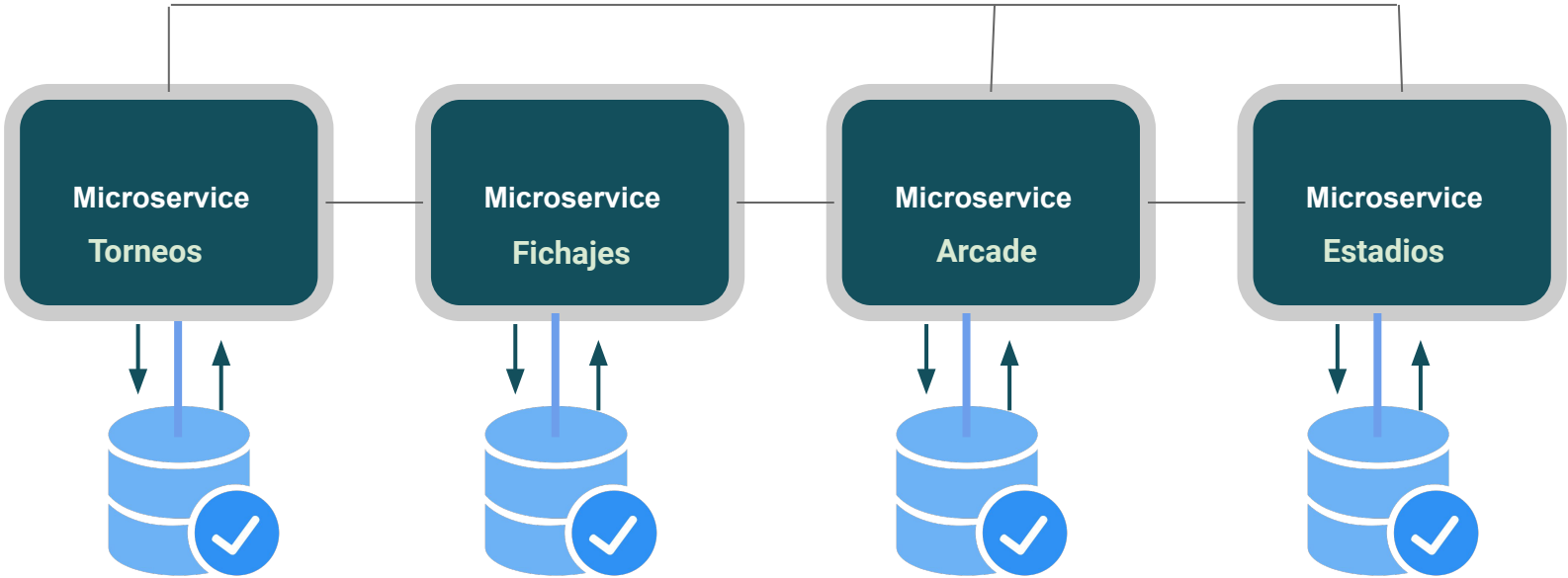
Arquitectura orientada a Microservicios

Patrones y coreografía de microservicios

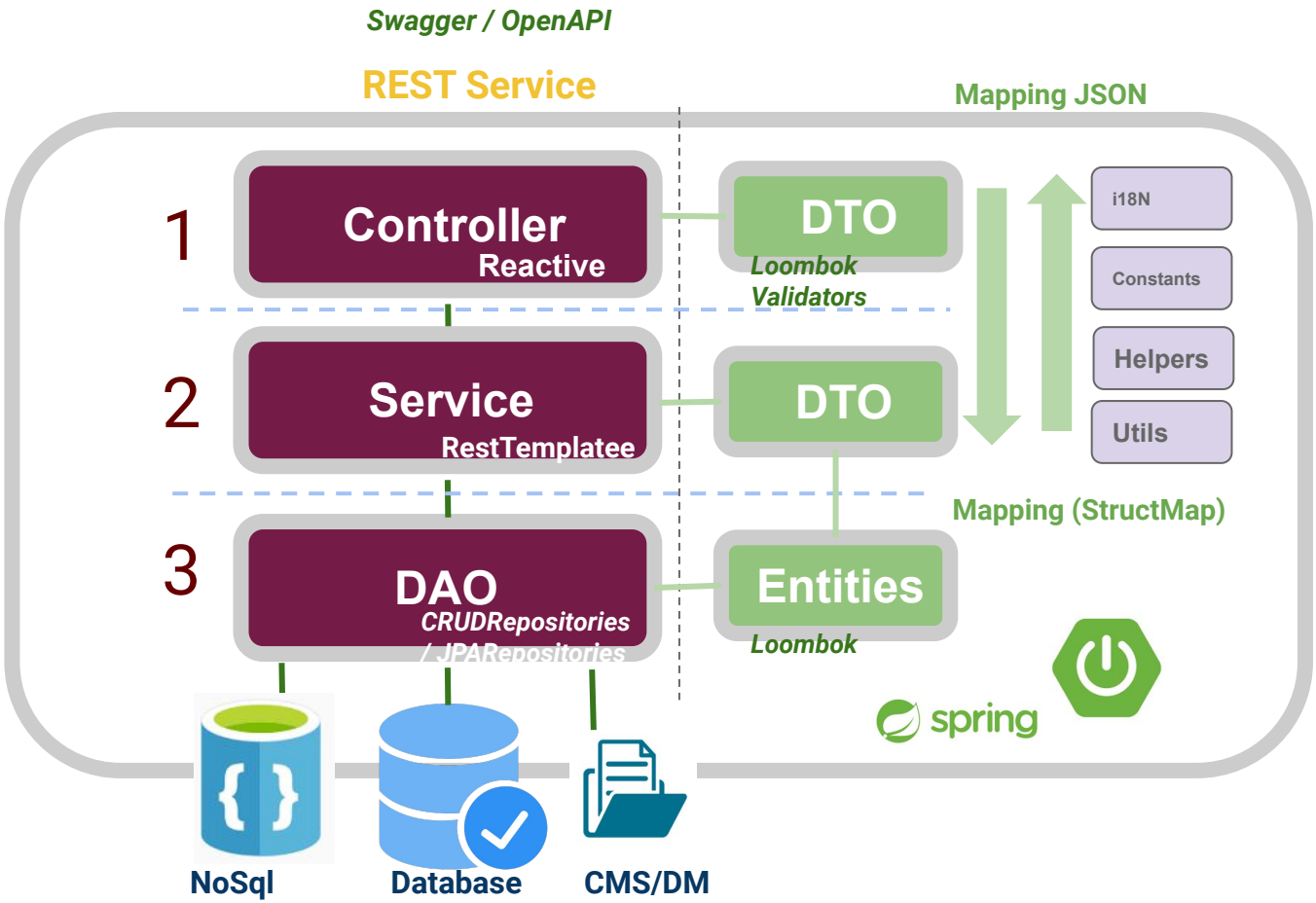
REST Services {JSON}
Graphql
grpc (HTTP/2)
Mensajeria (Kafka / RabbitMQ (amqp))



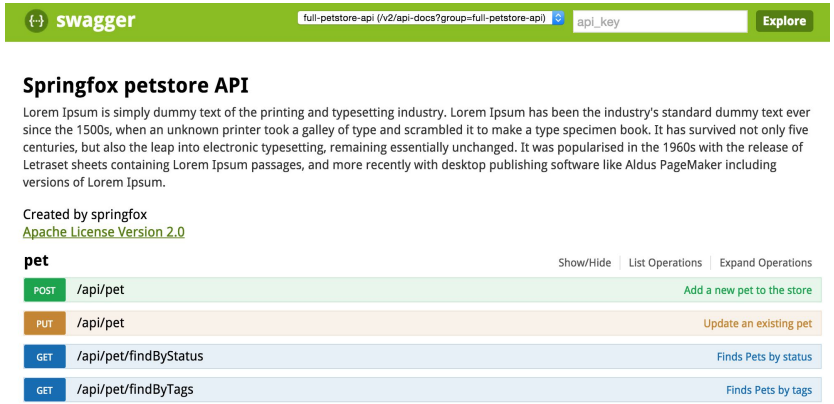
Patrones y coreografía de microservicios



Anatomía de un microservicio



API-FIRST Design

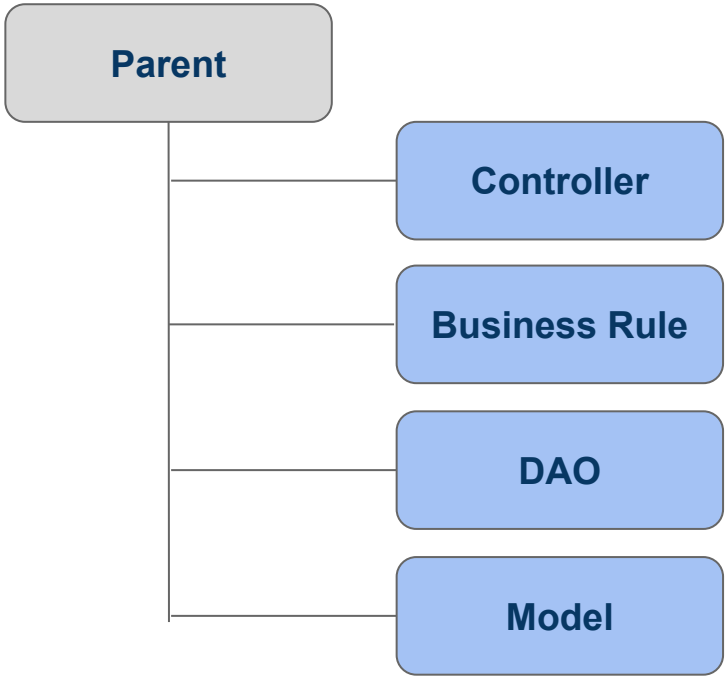


Spring Rest / Hatoeas / Springfox API

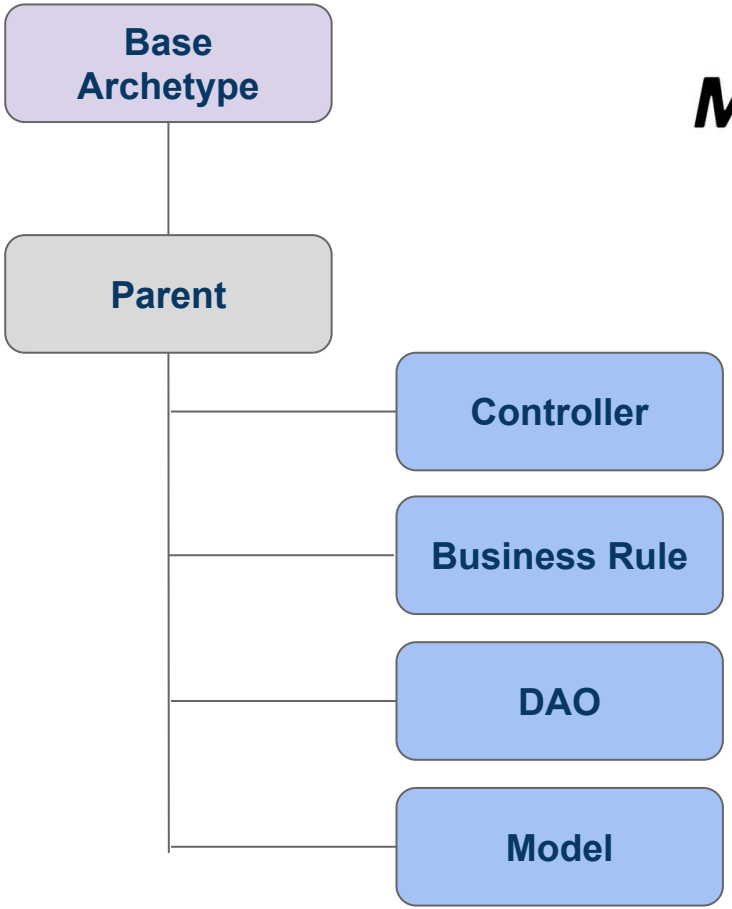
Microservice

Desarrollo

Maven™



Desarrollo



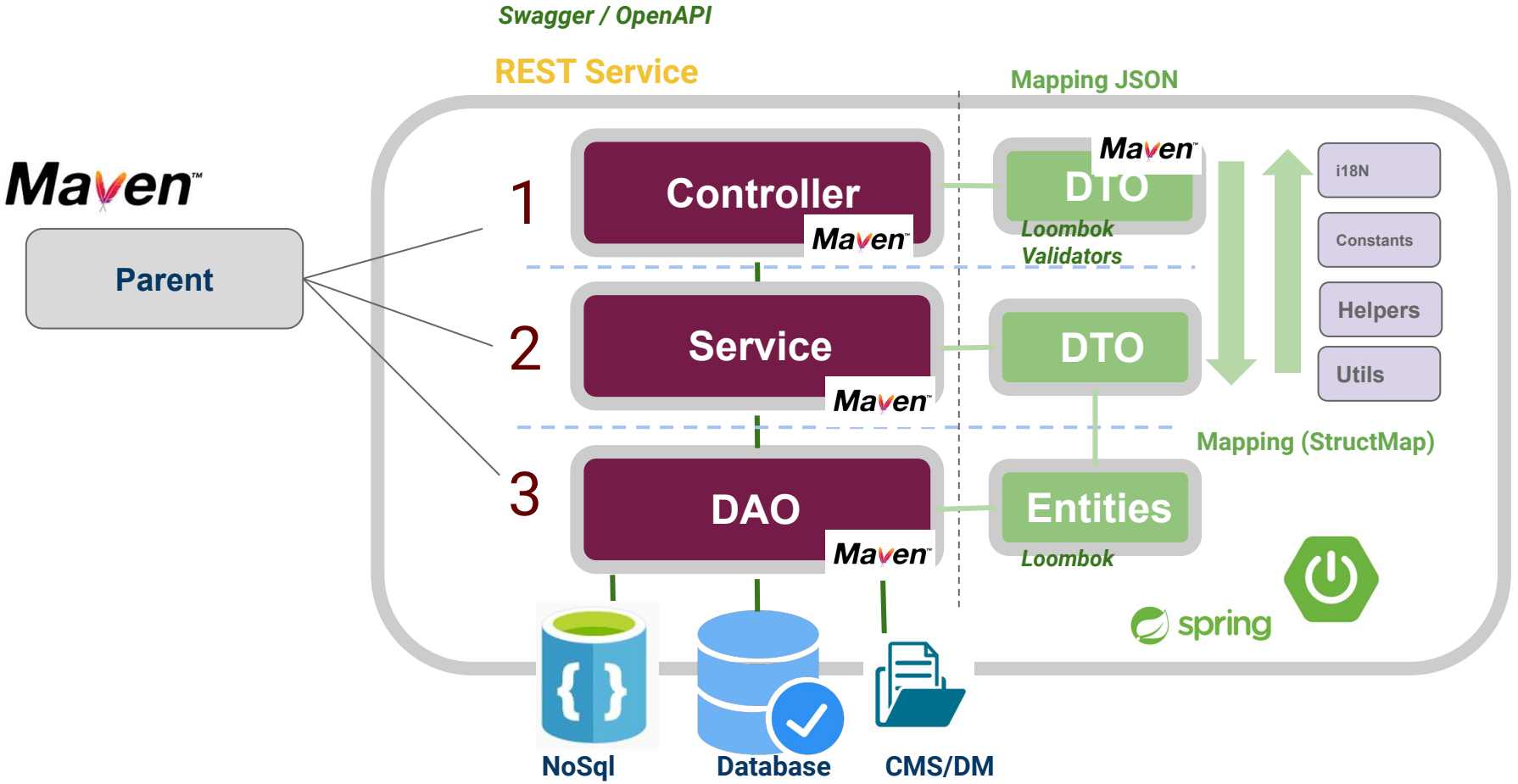
Maven™

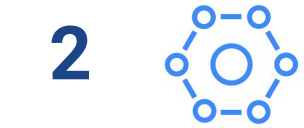
Maven™

Artifactory

NoSql Database CMS/DM

Desarrollo

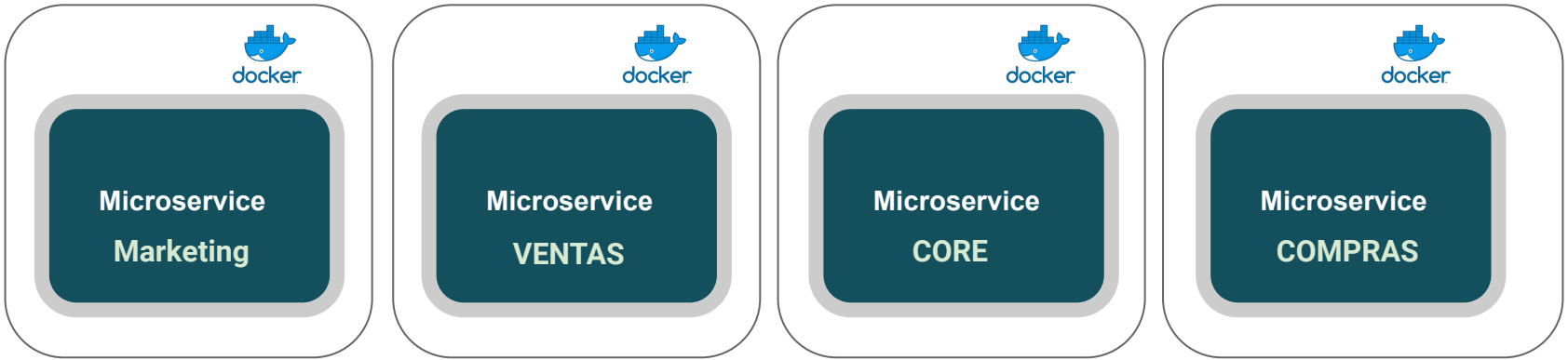

Swagger / OpenAPI
REST Service
Mapping JSON
Maven
Parent
1
Controller
Maven
DTO
Maven
Loombok
Validators
i18N
Constants
Helpers
Utils
2
Service
Maven
DTO
Mapping (StructMap)
3
DAO
Maven
Entities
Loombok
spring
NoSql
Database
CMS/DM



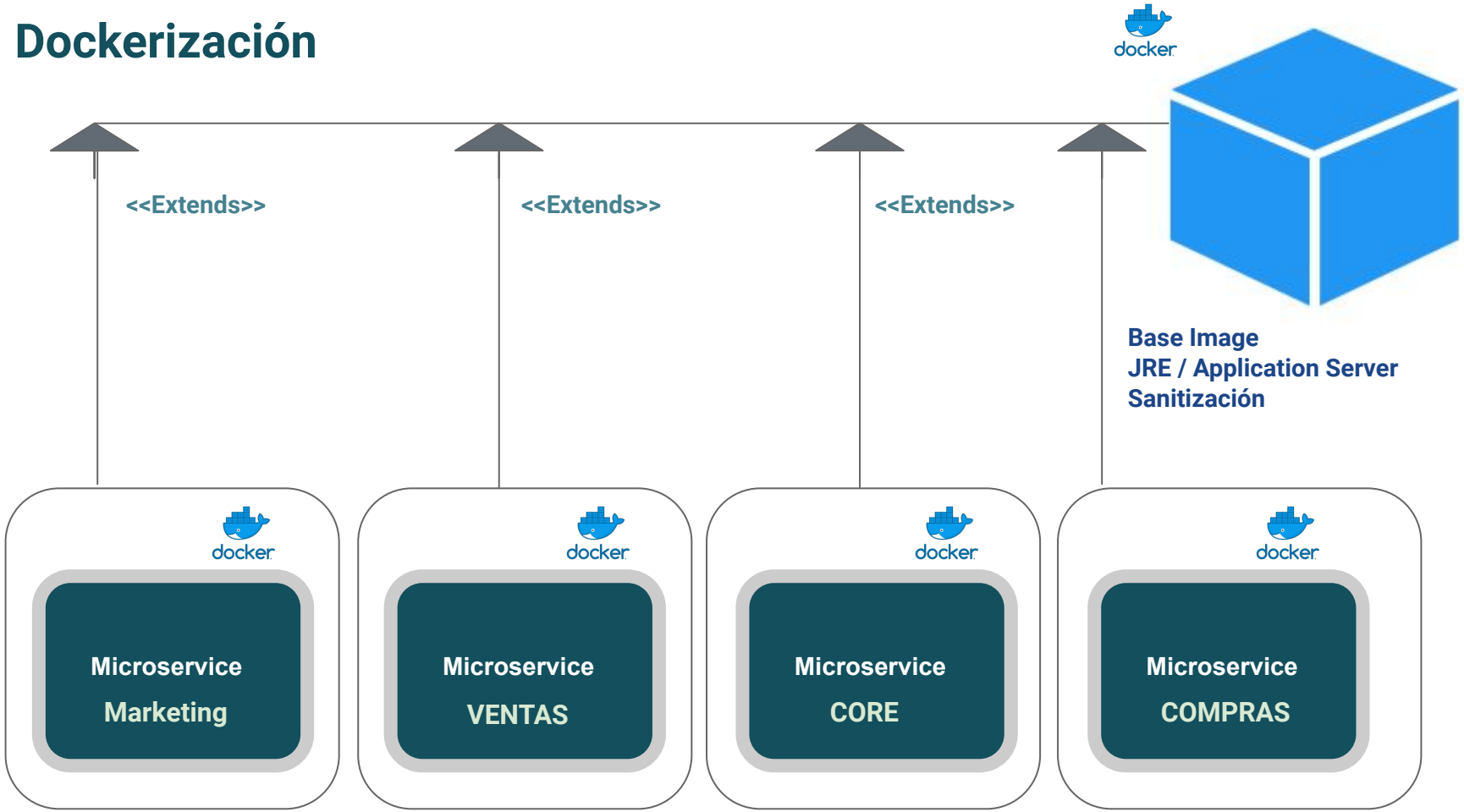
CI/CD

GitLab ♡ docker

Dockerización

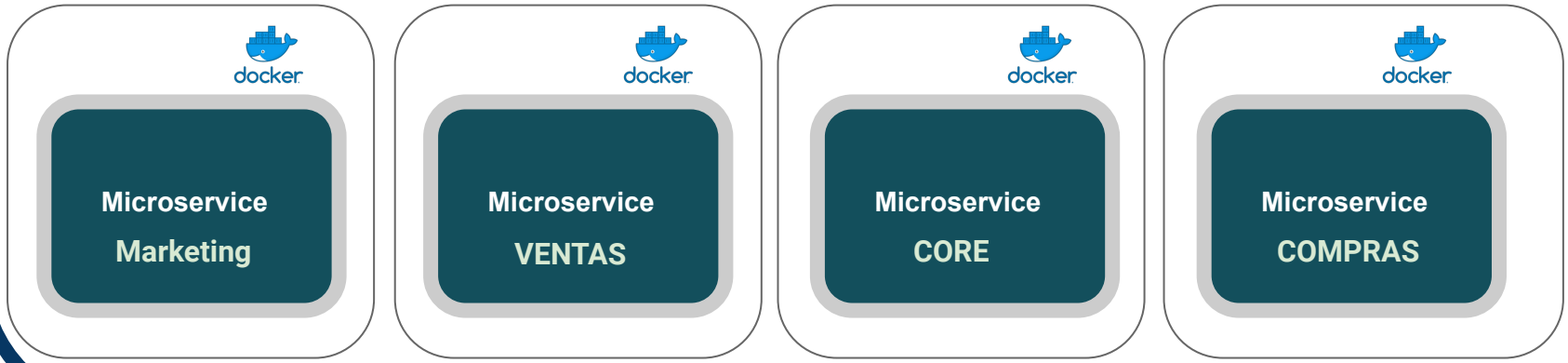


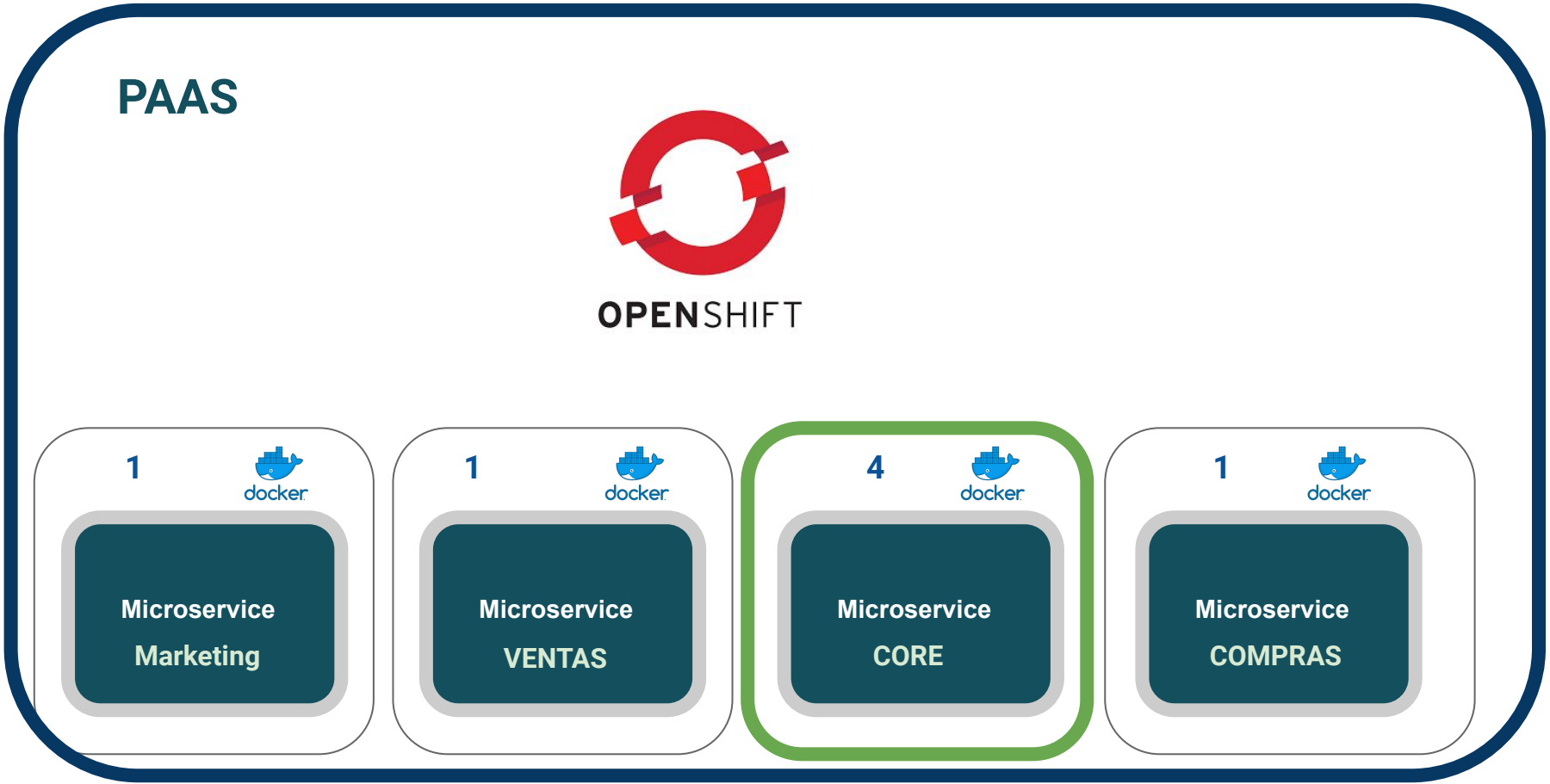
Dockerización



PAAS


OPENSHIFT


docker
Microservice
Marketing
docker
Microservice
VENTAS
docker
Microservice
CORE
docker
Microservice
COMPRAS


PAAS
OPENSHIFT
1
docker
Microservice
Marketing
1
docker
Microservice
VENTAS
4
docker
Microservice
CORE
1
docker
Microservice
COMPRAS

PostgreSQL vs MySql

Mejor rendimiento ante consultas complejas (joins y subconsultas)
Orientado a desarrolladores PL / JAVA /C
Licencia totalmente libre
Instalación mas compleja
Proyectos Medios-Grandes

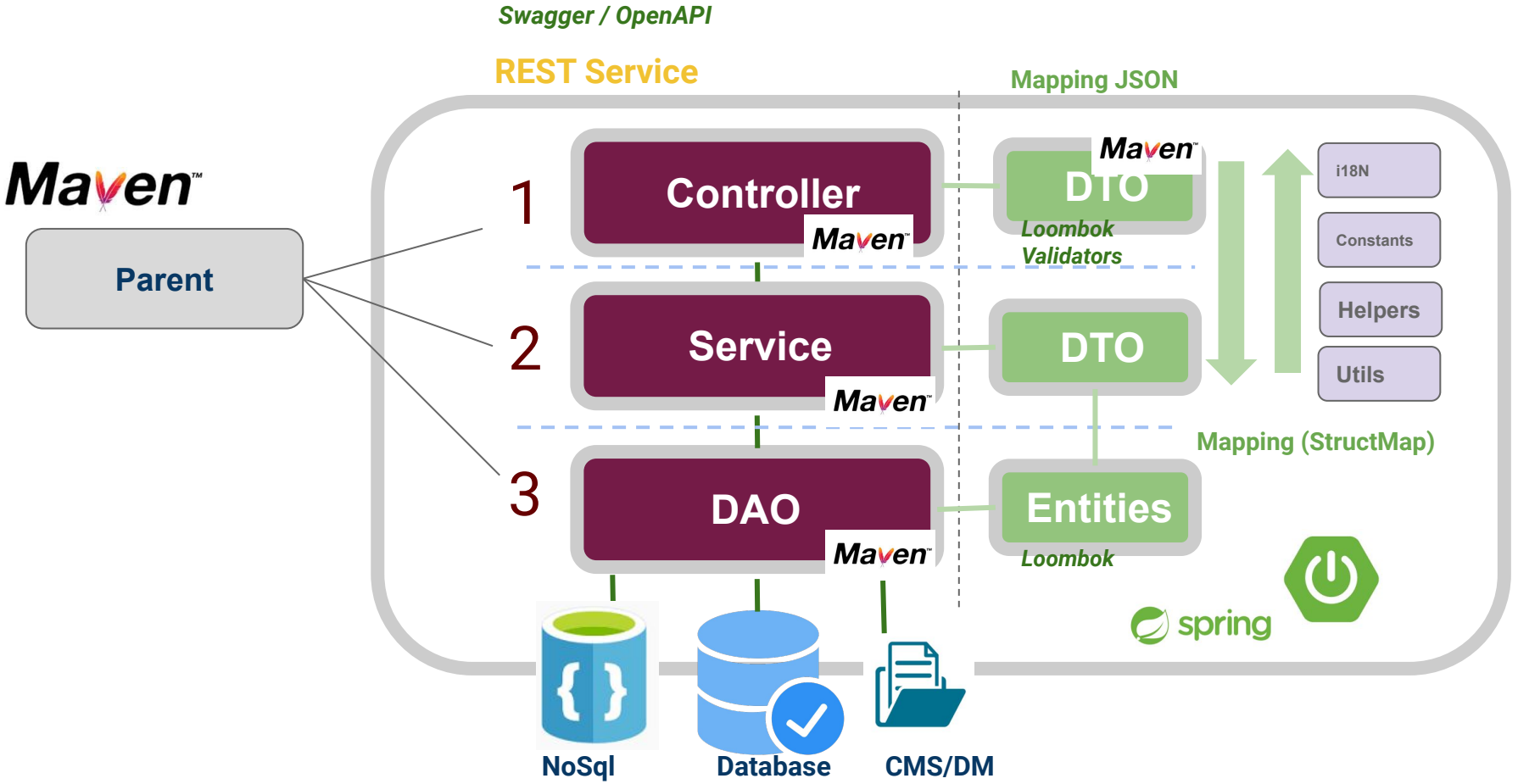
Alto rendimiento en consultas simples
GUI mas amigable
Puesta a punto simple
Licencia Abierta pero bajo el ala de Oracle
Proyectos chicos / web

Roadmap de versiones

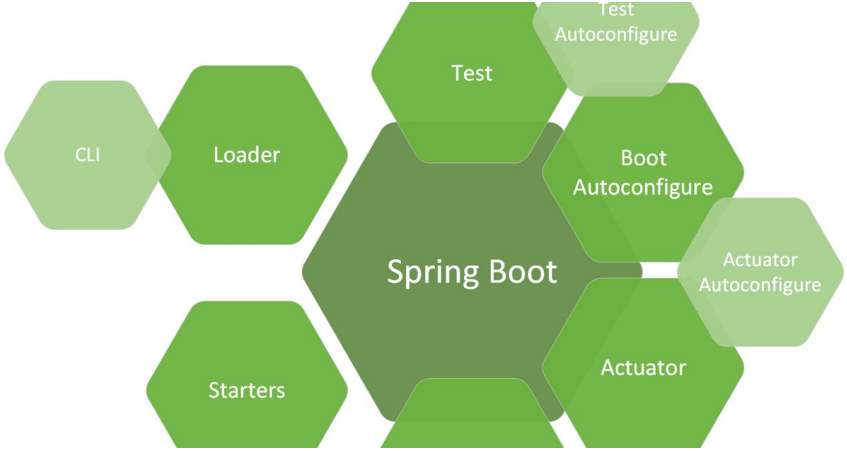
Java de 8 a 15

Java Politicas de Versionado

Desarrollo


Swagger / OpenAPI
REST Service
Mapping JSON
Maven
Parent
1
Controller
Maven
DTO
Maven
Loombok
Validators
i18N
Constants
Helpers
Utils
2
Service
Maven
DTO
Mapping (StructMap)
3
DAO
Maven
Entities
Loombok
spring
NoSql
Database
CMS/DM

Spring Boot



spring boot

Java Microservicios

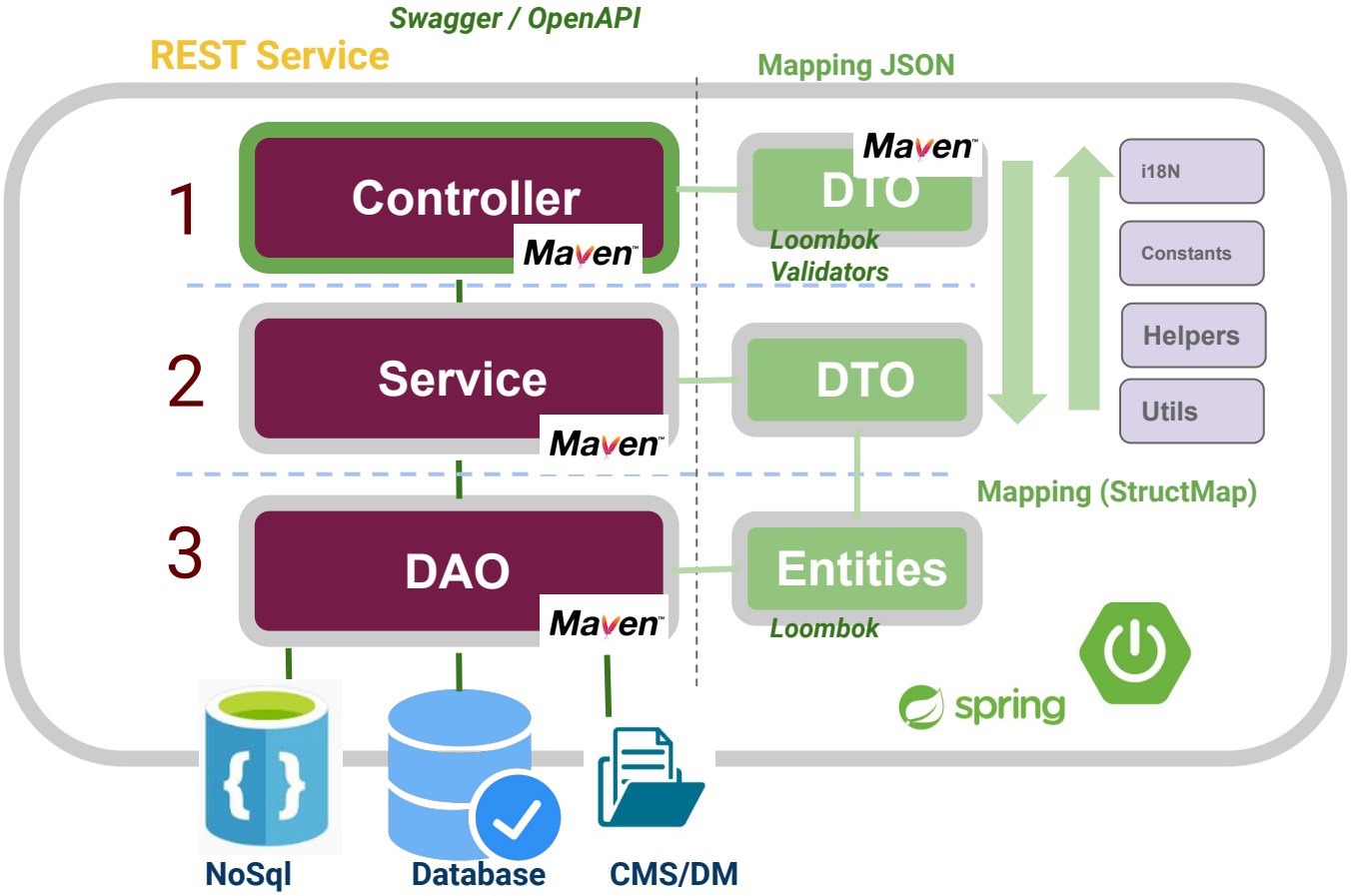
Spring Boot

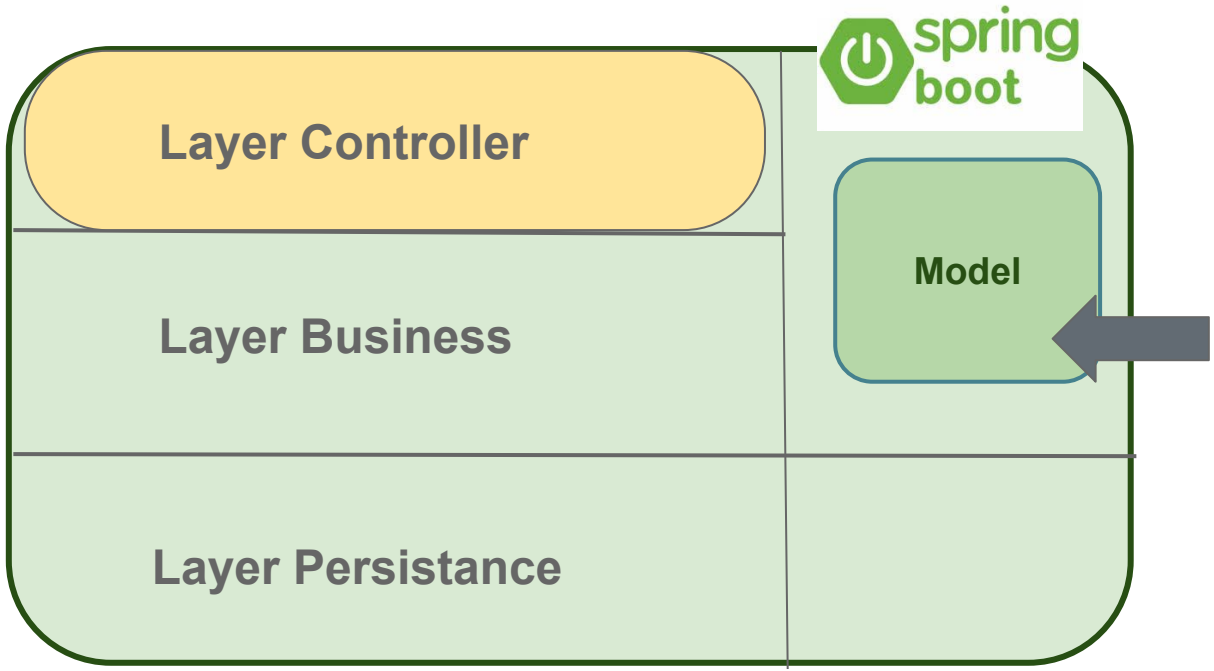
- Estandarización de proyectos
- Injección de dependencias
- Ejecución de proyectos
- Manejo de errores, validaciones, repositories

Spring Boot Startup!!!

Java Microservicios

Desarrollo

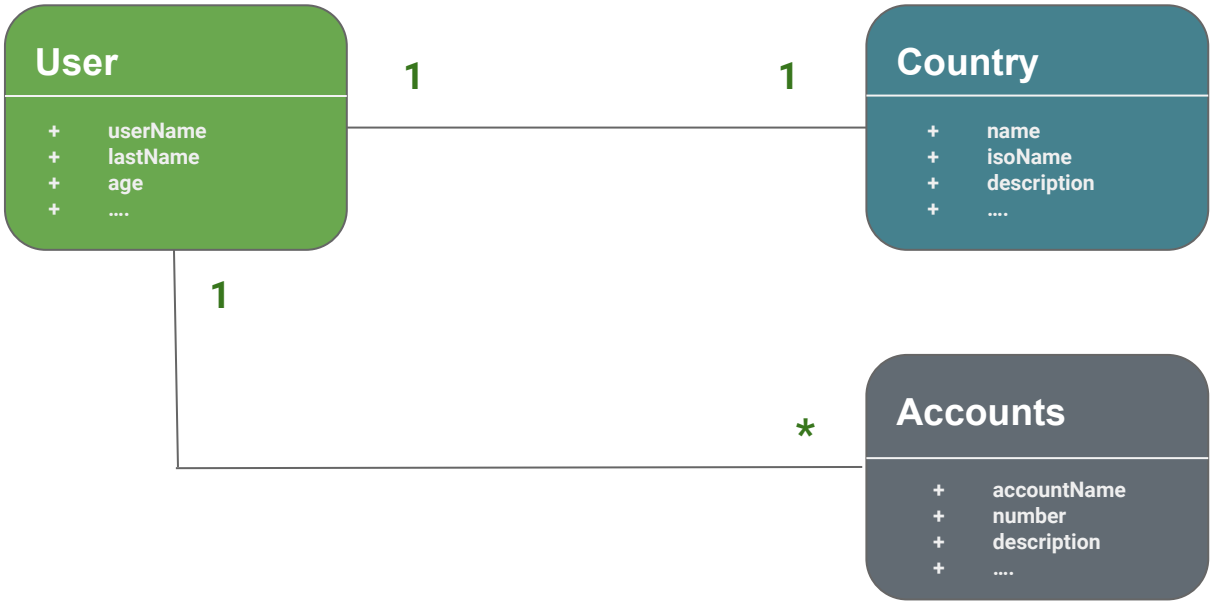

Swagger / OpenAPI
REST Service
Mapping JSON
1
Controller
Maven
2
Service
Maven
3
DAO
Maven
Maven
DTO
Loombok
Validators
DTO
Entities
Loombok
i18N
Constants
Helpers
Utils
Mapping (StructMap)
spring
NoSql
Database
CMS/DM



Taller de Spring Boot

Hola Mundo REST

Modelado


User
+ userName
+ lastName
+ age
+ ....
1
1
Country
+ name
+ isoName
+ description
+ ....
1
*
Accounts
+ accountName
+ number
+ description
+ ....

Project Lombok

Lombok

- Generación automatica de codigo fuente
- Asistencia para codigo repetitivo y comun
- Reduce el código fuente significativamente
- Generalmente usado para POJO's
- Getter y Setters
- Constructores
- equals y hashcode
- toString
- Otras Features
- Se debe instalar un plugin para el IDE (eclipse, IntelliJ, etc)

CRUD

spring boot

Rest User

getById	→	retorna un { user } por id
listAll	→	retorna lista de [{user},{user},{user}]
create	→	crea un { user }
update	→	Actualiza un {user}
delete	→	Elimina un {user}

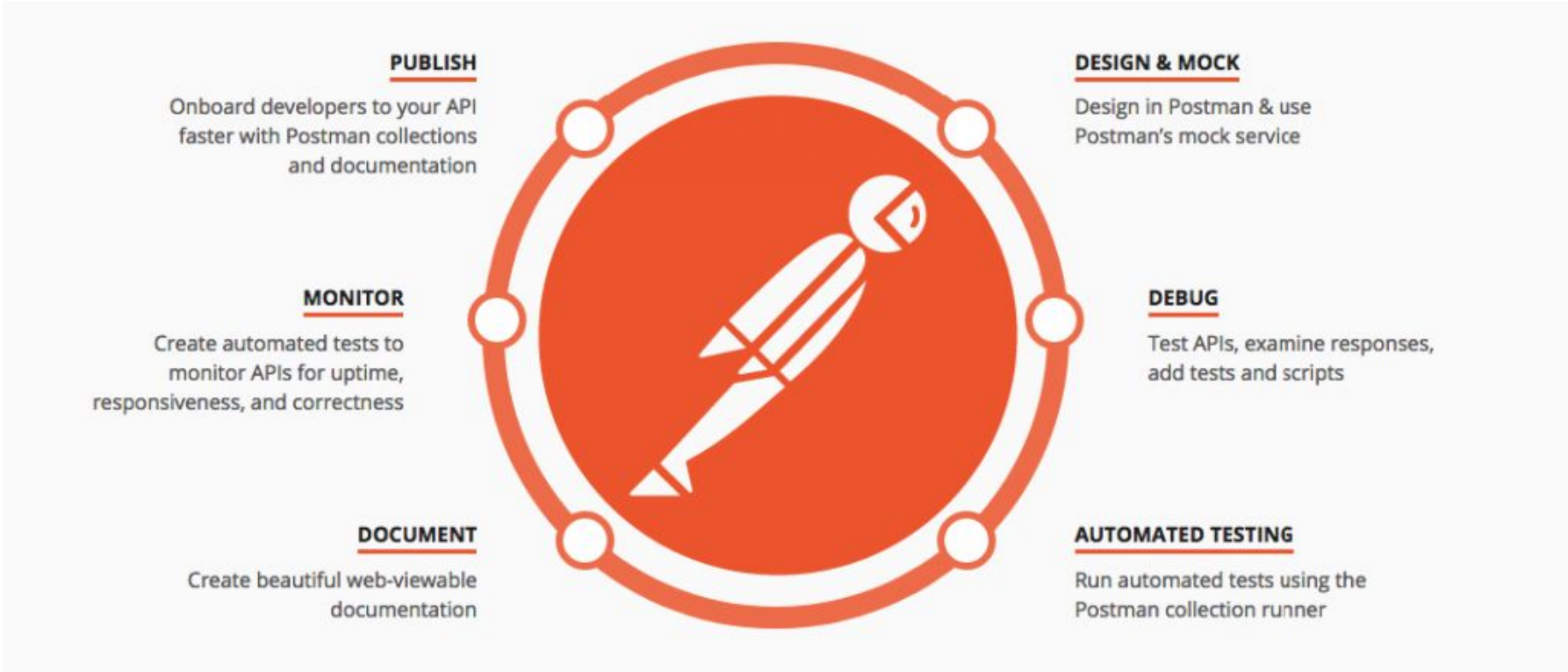
Rest User

getById	→	GET
listAll	→	GET
create	→	POST
update	→	PUT
delete	→	DELETE

Obtener URI:

```
URI location = ServletUriComponentsBuilder.
                                        fromCurrentRequest()
                                        .path("/{id}")
                                        .buildAndExpand(userDTO.getId())
                                        .toUri();
```

POSTMAN


PUBLISH
Onboard developers to your API faster with Postman collections and documentation
DESIGN & MOCK
Design in Postman & use Postman's mock service
MONITOR
Create automated tests to monitor APIs for uptime, responsiveness, and correctness
DEBUG
Test APIs, examine responses, add tests and scripts
DOCUMENT
Create beautiful web-viewable documentation
AUTOMATED TESTING
Run automated tests using the Postman collection runner

POSTMAN

- Environments y variables
- Runners y Testing

Rest User

		Código Errores HTTP
getById	GET	200 - OK
listAll	GET	200 - OK
create	POST	201 - CREATED
update	PUT	200 - OK
delete	DELETE	200 - OK

Rest User (Not Found)

getById	→	GET	404 - OK
listAll	→	GET	404 - OK
create	→	POST	404 - OK
update	→	PUT	404 - OK
delete	→	DELETE	404 - OK

application.properties


```
server.port=8081
server.servlet.context-path=/escuelait/api/v1/microservices
```

Unificar rutas por controller


```
@RequestMapping("/users")
```

Códigos de error HTTP ResponseEntity

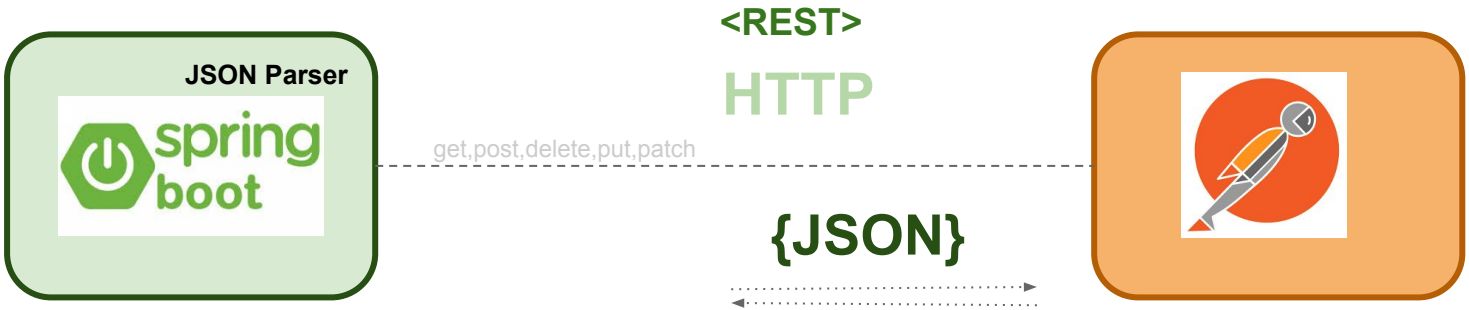
200 - OK
201 - Created
400 - Bad Request
401 - Unauthorized
404 - Not Found
500 - Internal error server

Not Found
Recurso no encontrado

Anidación de recursos

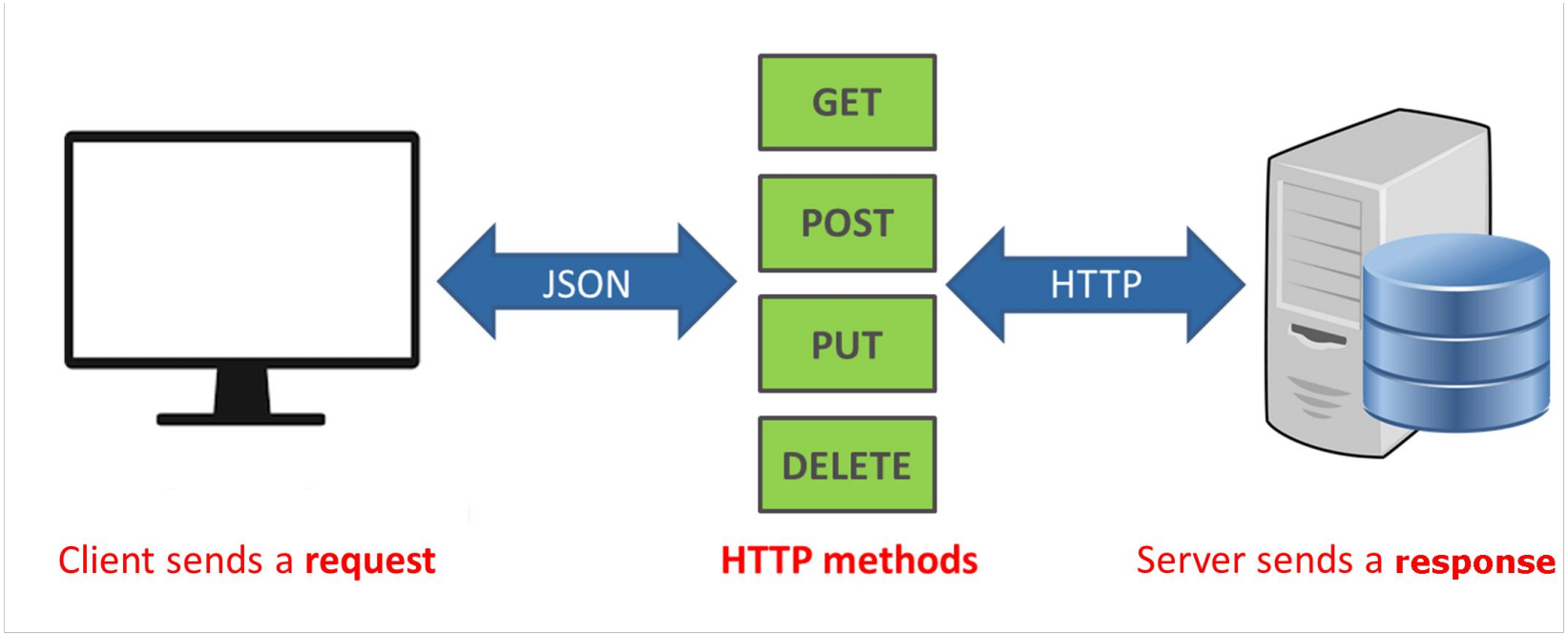
users/{id}/accounts

REST Protocolo y JSON



Taller de Spring Boot

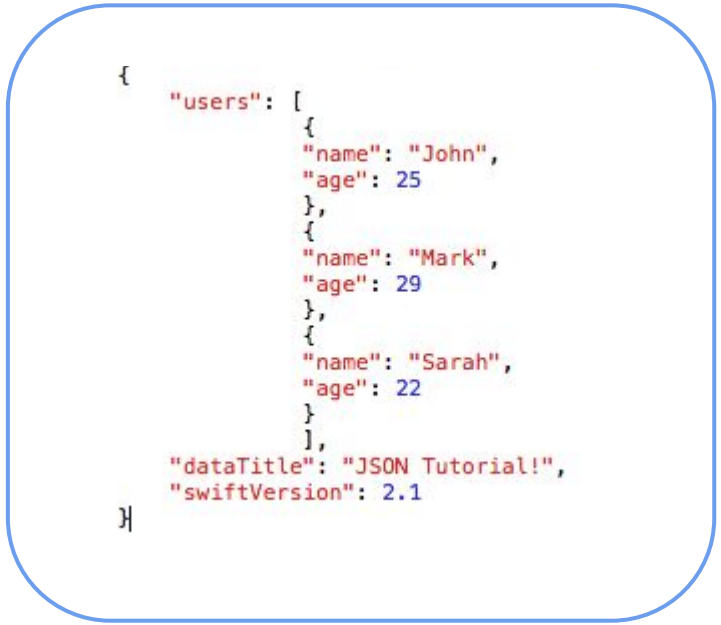
{ REST:API }


GET
POST
PUT
DELETE
JSON
HTTP
Client sends a **request**
HTTP methods
Server sends a **response**

{ REST:API }

- Ligero
- Facilidad de parseo
- Legibilidad
- Transferencia
- Configuración

{ JSON }



```
{
    "users": [
                {
                "name": "John",
                "age": 25
                },
                {
                "name": "Mark",
                "age": 29
                },
                {
                "name": "Sarah",
                "age": 22
                }
                ],
    "dataTitle": "JSON Tutorial!",
    "swiftVersion": 2.1
}
```

```
public class Users{

 List<User> users;


}
```

Spring REST

Jackson / gson


```
{
    "users": [
                {
                "name": "John",
                "age": 25
                },
                {
                "name": "Mark",
                "age": 29
                },
                {
                "name": "Sarah",
                "age": 22
                }
                ],
    "dataTitle": "JSON Tutorial!",
    "swiftVersion": 2.1
}
```

URL

http://localhost:8080/escit/api/v1/microservices/users/

Protcol domain port domain api version subdomain service

Rest Buenas Prácticas

Rest Buenas Practicas

REST es orientado a resources

- Usar sustantivos (no verbos)
- Usar plural

POST **/usuarios** 

/crearUsuarios 

Rest Buenas Practicas

REST concatenación de recursos de manera apropiada

GET	**/usuarios/**	(todos los usuarios)
GET	**/usuarios/{id}**	(el usuario con {id}
GET	**/usuarios/{id}/accounts/**	Todas la cuentas del usuario con id
GET	**/usuarios/{id}/accounts/{idAcc}**	La cuenta con id del usuario id

Rest Buenas Practicas

Entregar codigos HTTP adecuados

No mas de tres niveles de resources

equipos/{1}/jugadores/{1}/estadisticas/

JSON claros y no mas de 3 niveles

```
{
nombre: “Real Madrid”
jugadores: [
  {nombre: “Benzema”, estadisticas: [
                .....
        ]}
]
}
```

La cuenta con id del usuario id

Rest Buenas Practicas

Evitar!!!

/equipos/1/jugadores/1/partidos/2/goles/1/multimedia/1

/users/1/accounts/1/orders/1/products/1/

Rest Buenas Practicas

No entregar JSON “Enormes”

- Paginar
- Sortear
- Filtrar
- Crear un rest query con filters

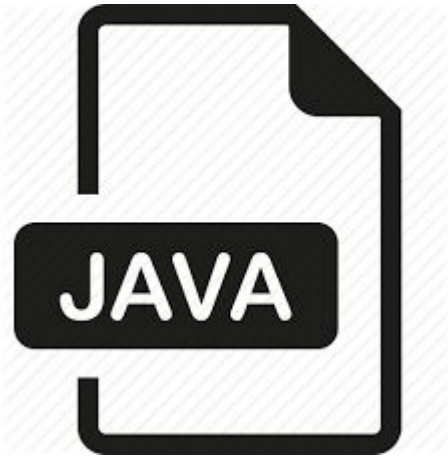
Rest Buenas Practicas

- Documentar la API
  - Open API - Swagger
- Generar links de navegación
  - HATEOAS
- Asegurar las apis definiendo roles

Rest Buenas Practicas

- Exportar clientes de demostración
  - Ej: POSTMAN
- Entregar errores definidos y verbosos
- Validación de campos
- Versionado apropiado de la API

Externalización



application.properties

Externalización

- @Value
- @ConfigurationProperties Prefix
- Listas de valores
- Variables de entornos
- Valor constante
- Cambio de archivo de propiedades:
  @PropertySource(value = "classpath:mensajes.properties")

Práctica

Práctica

1. Generar un proyecto Spring Boot desde cero
    a. Incorporando lombok y web
2. Crear un modelo Equipo -> Jugadores
    a. Lombok: Generar constructores/getter/setter/toString/Equals
    b. Para jugadores generar un constructor exclusivo para los campos Numero/Nombre
3. Crear un controlador CRUD Rest para la entidad equipo
4. Retornar los codigos de error apropiados
5. Generar un proyecto POSTMAN con todas las llamadas
    a. Generar variables de entorno para base url
    b. Exportar el proyecto
6. Agregar un servicio teams/{id}/players que devuelva un array con todos sus jugadores
7. Crear un CRUD controller para Players
8. Externalizar los siguientes datos:
    a. Nombre del juego
    b. Edicion
    c. Año
    d. Liga