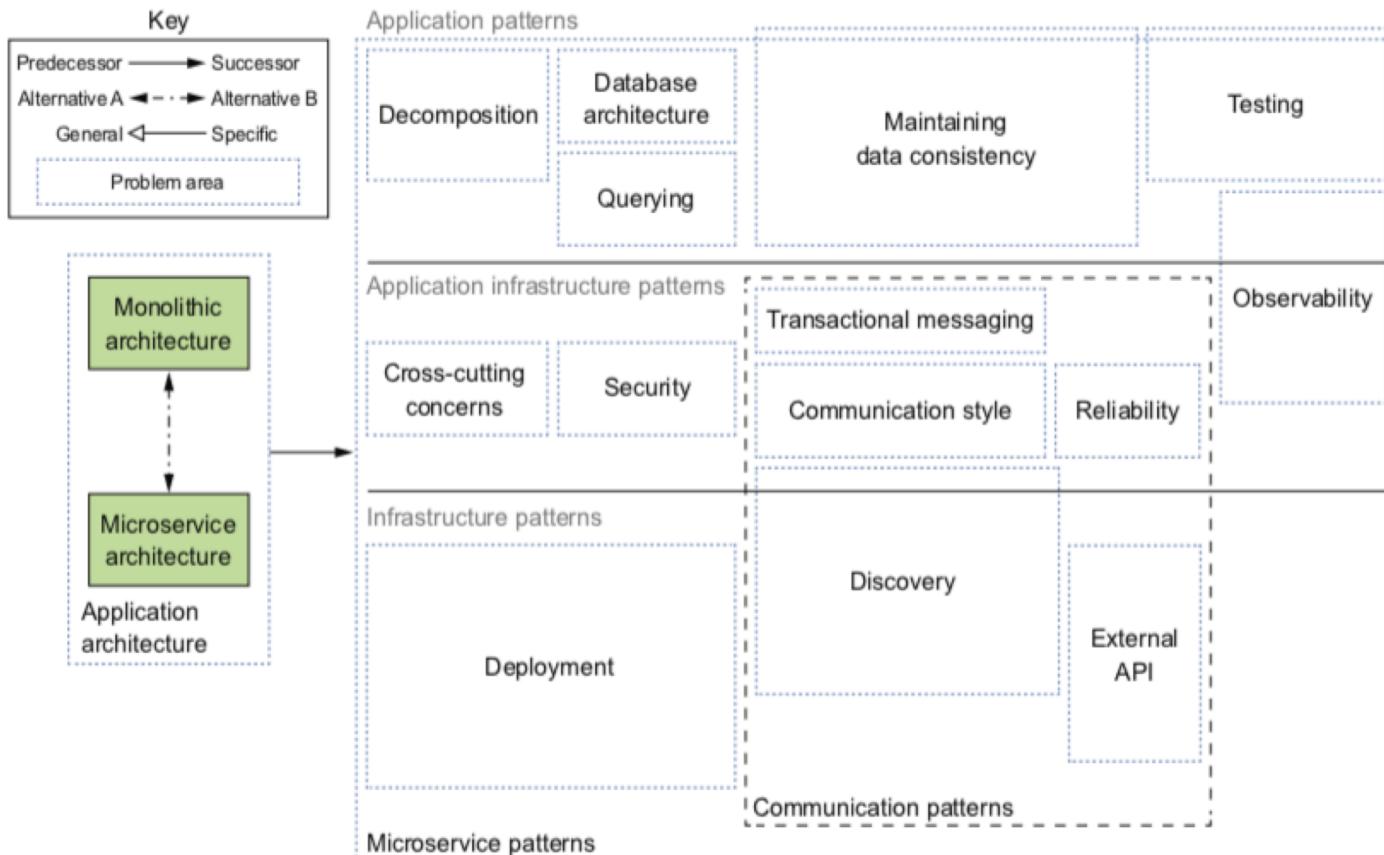


Arquitectura e Implementación de Microservicios

Sesión 2

PATRONES DE DISEÑO DE MICROSERVICIOS

Clasificación



- Patrones de infraestructura
- Patrones de infraestructura de la aplicación
- Patrones de la aplicación

Patrones de Comunicación inter-servicios

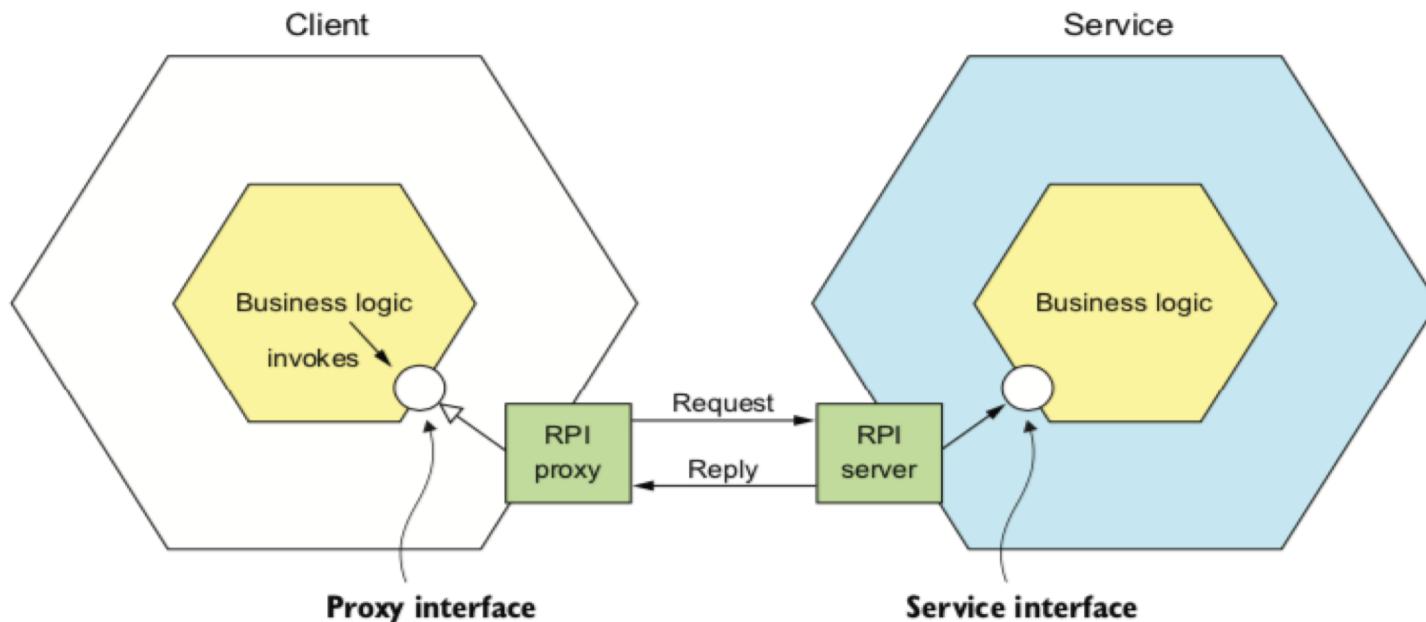
- Servicios colaboran.
- Instancias en diferentes servidores
- Lo común es REST
- Alternativas:
 - Request/reponse
 - REST , GRPC
 - Asincrona AMQP , STOMP, etc

Tipos de comunicación

- Cantidad de servicios
 - Uno a uno
 - Uno a muchos
 - Publica/suscribe
- Estilo de comunicación
 - Sincrona
 - Cliente espera
 - Ejemplo RPC
 - Asincrona
 - Cliente no espera

Comunicación Síncrona usando RPC

- Interfaz proxy



REST es un ejemplo de este tipo de comunicación

REST. Representational state transfer

- Es la moda
- REST es una estilo de arquitectura para el desarrollo de servicios web.
- Es popular debido a su simplicidad y a que esta construido sobre sistemas existentes y funciones del protocolo http.
- Recuso= objeto de negocio
 - Cliente, producto, etc
- Usa verbos para manipular recursos
- verbos:
 - Get, put, post, delete
 - Head, options
 - Verbos Indican al servidor que hacer con los datos
 - Get: representación del recurso
 - Post: crea recurso
 - Put: actualiza recurso

Codigos de respuesta http

- Indican si se ha completado satisfactoriamente una solicitud http.
- Agrupadas por rango
 - Informativas.
 - 1xx
 - Status parcial
 - Exito.
 - 2xx
 - Termino en exito
 - Redireccion.
 - 3xx
 - Recurso en otra ubicacion
 - Error del cliente.
 - 4xxx
 - Error del servidor.
 - 5xx
- <https://www.restapitutorial.com/httpstatuscodes.html>

Ejemplos rest

- Post /users
 - crea un usuario
- Get orders/{orderid}
 - devuelve una orden por id

Problema: como mapear operaciones de negocio usando los verbos http

- Problema de diseño de API rest
- Sabemos que Put para actualizaciones pero hay multiples formas de actualizar
 - cancelar una orden, revisar una orden, etc.
- Solucion: usar sub-recurso
 - Post orders/{orderid}/cancel
 - Post orders/{orderid}/revise

Beneficios de rest

- Es simple y familiar.
- Facil de testear
- Soporta directamente estilo de comunicación request/response
- http no es bloqueado por el firewall
- no es necesario usar un bróker intermediario

Desventajas

- Solo soporta el estilo de comunicación request/reponse
- Clientes deben conocer las URLs de las instancias de los servicios.
 - Para solucionar este problema esta el patrón Service Discovery.
- Algunas veces es difícil mapear operaciones de negocio a verbos http.

gRPC

- Alternativa a REST popular
- grpc usa un formato de mensaje binario.
 - google protocol buffer. protobuf.
- Beneficios:
 - fácil diseñar un api con una amplia variedad de operaciones de negocio.
 - eficiente especialmente cuando se intercambia mensajes muy largos.
 - interoperatividad entre clientes y servicios
- Desventajas
 - Toma mas trabajo para los clientes javascript consumir grpc comparado con rest
 - Esta basado en http2 y algunos firewalls aun no lo soportan.

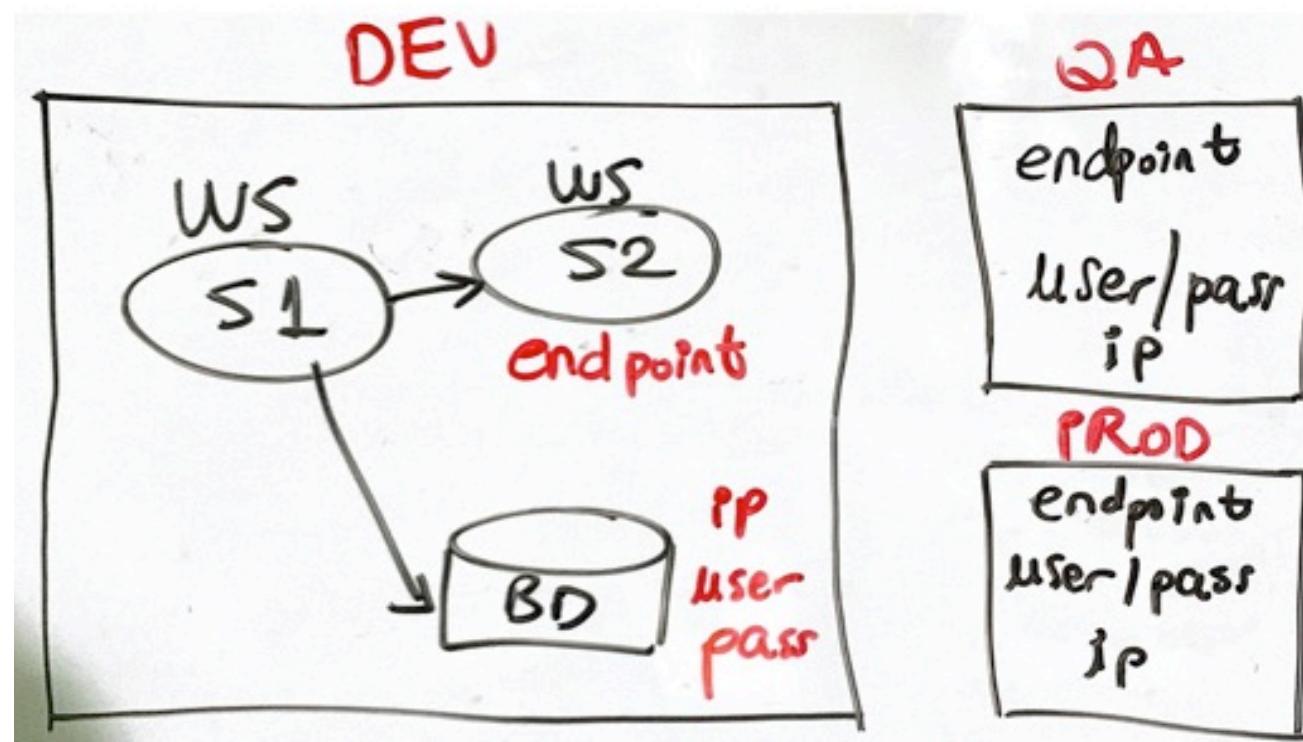
LABORATORIO

- sistema operativo ubuntu
- plataforma java 8
- herramientas
 - Git
 - Postman
 - Sublime
 - eclipse
 - maven
 - Docker
 - open-office

DEMO REST SERVICE

Controlando configuración con Spring cloud configuration server

Administracion de la configuracion



Administracion de la configuracion

- Propiedades de configuración: donde?
 - Clase constantes
 - Problema: re-compilar y re-deployar
- Debes separar completamente
 - No tienes que recompilar
- YAML, JSON, XML?
 - Trabaja para pocas aplicaciones
 - Falla para Cloud=100+ microservicios e instancias

Administracion de la configuracion

- Microservicios cloud nativos enfatizan:
 - Separar completamente configuración
 - Servidor no cambian entre ambientes
 - Inyectar configuración en el arranque:
 - Variables de entorno
 - Leer repositorio central
- Administración de la configuración:
 - Critico para los microservicios cloud
 - Instancias Deployadas rápidamente

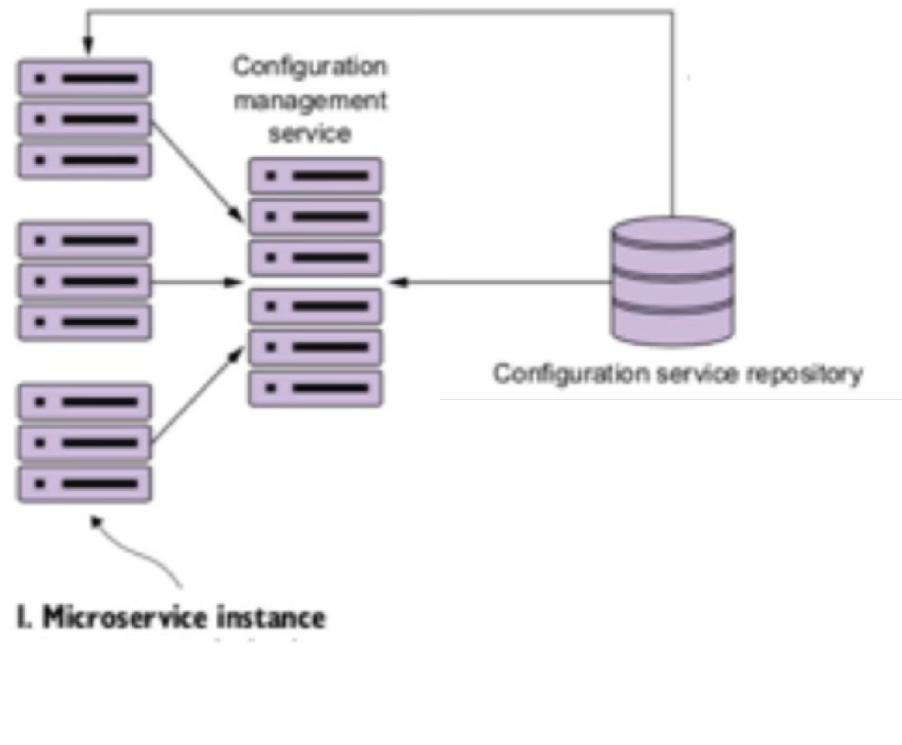
4 Objetivos

- Separacion
 - Completamente del despliegue físico
 - No deployada
 - pasada en arranque
- Abstraccion
 - Ocultar bajo interface
 - No acceder directamente repositorio
 - Usar servicio REST/JSON
- Centralizacion
 - Aplicación = cientos microservicios
 - Minimizar repositorios
- Fortalecimiento
 - Completamente separada y centralizada
 - Critico alta disponibilidad

Es una dependencia externa: administrada y versionada.

- Sino: bugs , caídas, etc

Esquema general



1. Servicios inician
2. Url del servidor de configuración
3. Repositorio

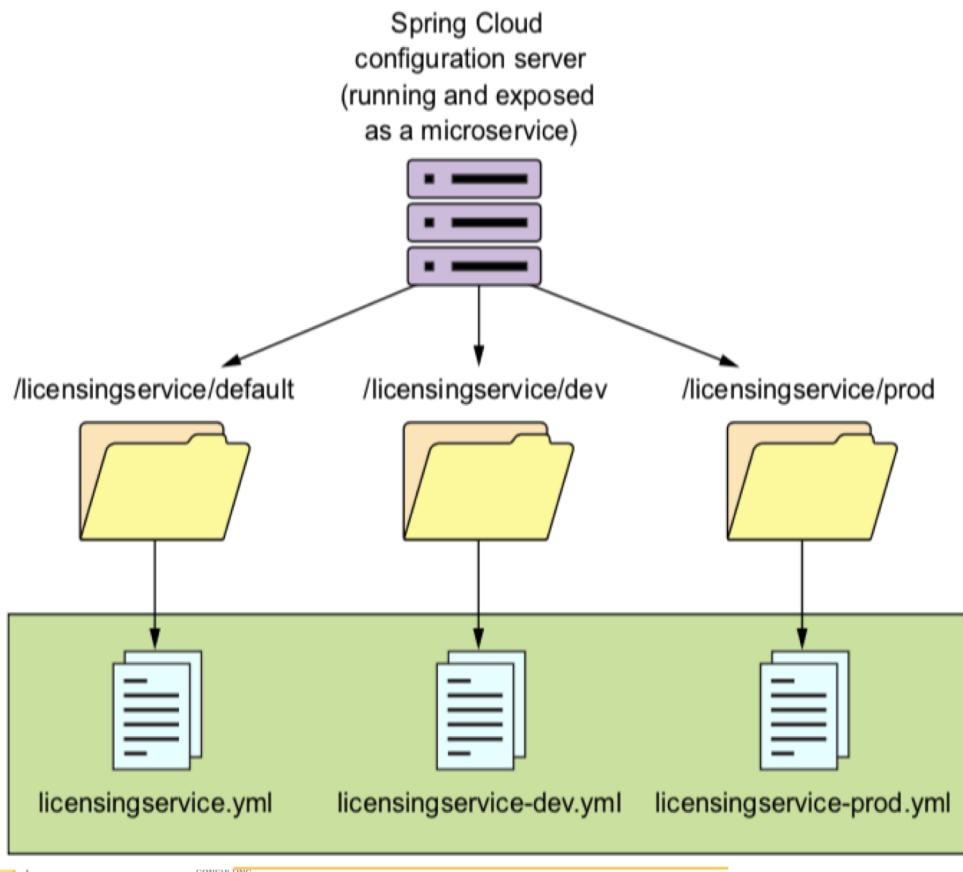
Alternativas implementacion

Project Name	Description	Characteristics
Etcd	Open source project written in Go. Used for service discovery and key-value management. Uses the raft (https://raft.github.io/) protocol for its distributed computing model.	Very fast and scalable Distributable Command-line driven Easy to use and setup
Eureka	Written by Netflix. Extremely battle-tested. Used for both service discovery and key-value management.	Distribute key-value store. Flexible; takes effort to set up Offers dynamic client refresh out of the box
Consul	Written by Hashicorp. Similar to Etcd and Eureka in features, but uses a different algorithm for its distributed computing model (SWIM protocol; https://www.cs.comell.edu/~asdas/research/dsn02-swim.pdf).	Fast Offers native service discovery with the option to integrate directly with DNS Doesn't offer client dynamic refresh right out of the box
ZooKeeper	An Apache project that offers distributed locking capabilities. Often used as a configuration management solution for accessing key-value data.	Oldest, most battle-tested of the solutions The most complex to use Can be used for configuration management, but should be considered only if you're already using ZooKeeper in other pieces of your architecture
Spring Cloud configuration server	An open source project that offers a general configuration management solution with different back ends. It can integrate with Git, Eureka, and Consul as a back end.	Non-distributed key/value store Offers tight integration for Spring and non-Spring services Can use multiple back ends for storing configuration data including a shared filesystem, Eureka, Consul, and Git

1. usando repositorio de configuración el filesystem

- Aclaracion:
 - No recomendable aplicaciones grandes o medianas
 - Necesita shared filesystem para instancias
 - Es tu responsabilidad mantenerlo
- Spring config server, rest/ spring boot
- Embebido en ya existente o nuevo
- confsvr

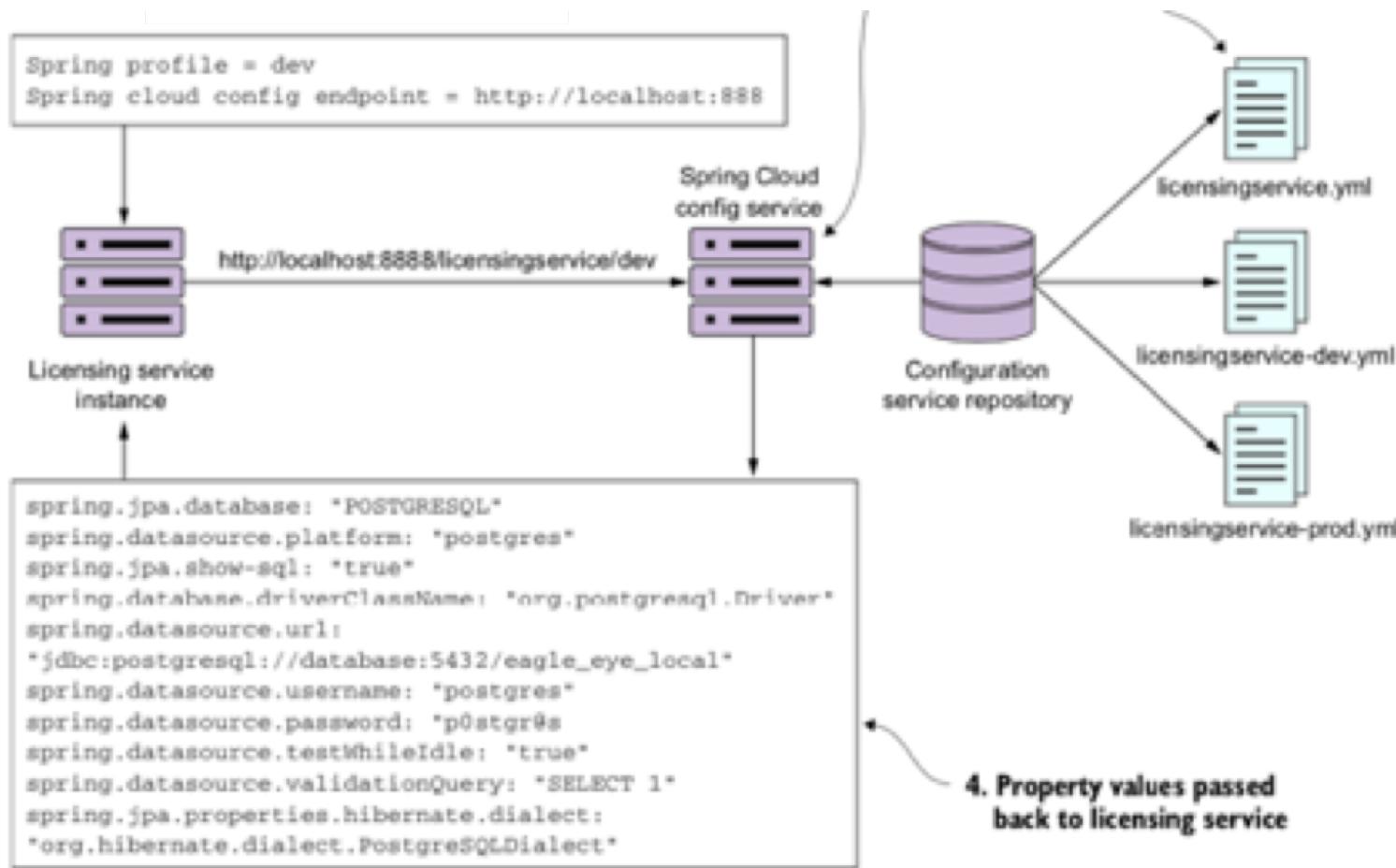
Esquema de confsvr



- Un microservicio mas
- Rest http endpoint
- 3 ambientes: default, dev, prod
- url: nombre aplicación + ambiente
- Archivos yml: appname-env.yml

Laboratorio

2. Integrando un cliente



Laboratorio

3. Usar repositorio git

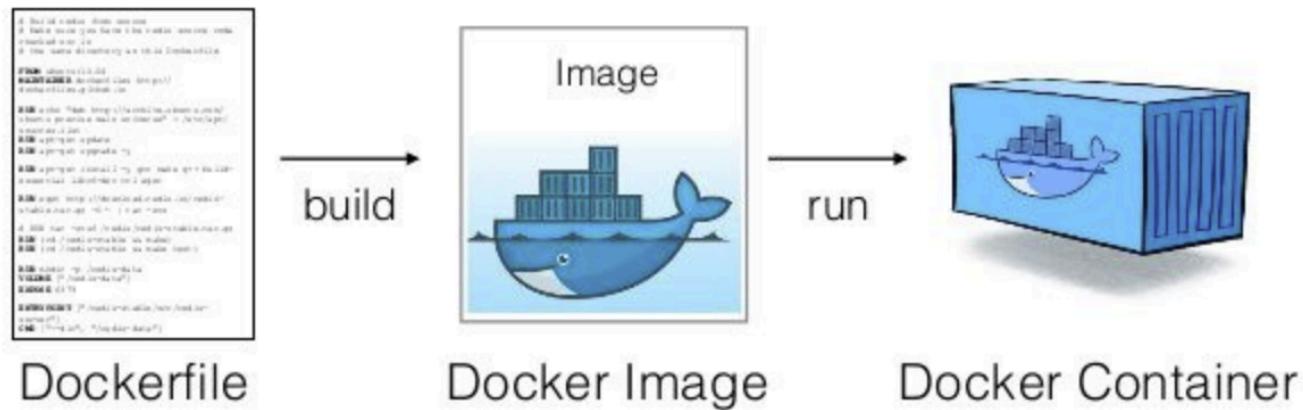
- Repositorio filesystem impráctico
 - montar filesystem en instancias del config server
- Alternativa: repositorio git
 - Beneficios: Configuracion app versionada

Docker



- Docker es una herramienta que permite a desarrolladores, sysadmin, etc desplegar fácilmente aplicaciones en un entorno aislado (llamado contenedor),
- para que se ejecuten dentro del sistema operativo anfitrión. eg Linux
- permite empaquetar una aplicación con todas sus dependencias dentro de una pieza estandarizada de software.
- a diferencia de máquinas virtuales, contenedores son mucho más ligeros y por ende más eficientes

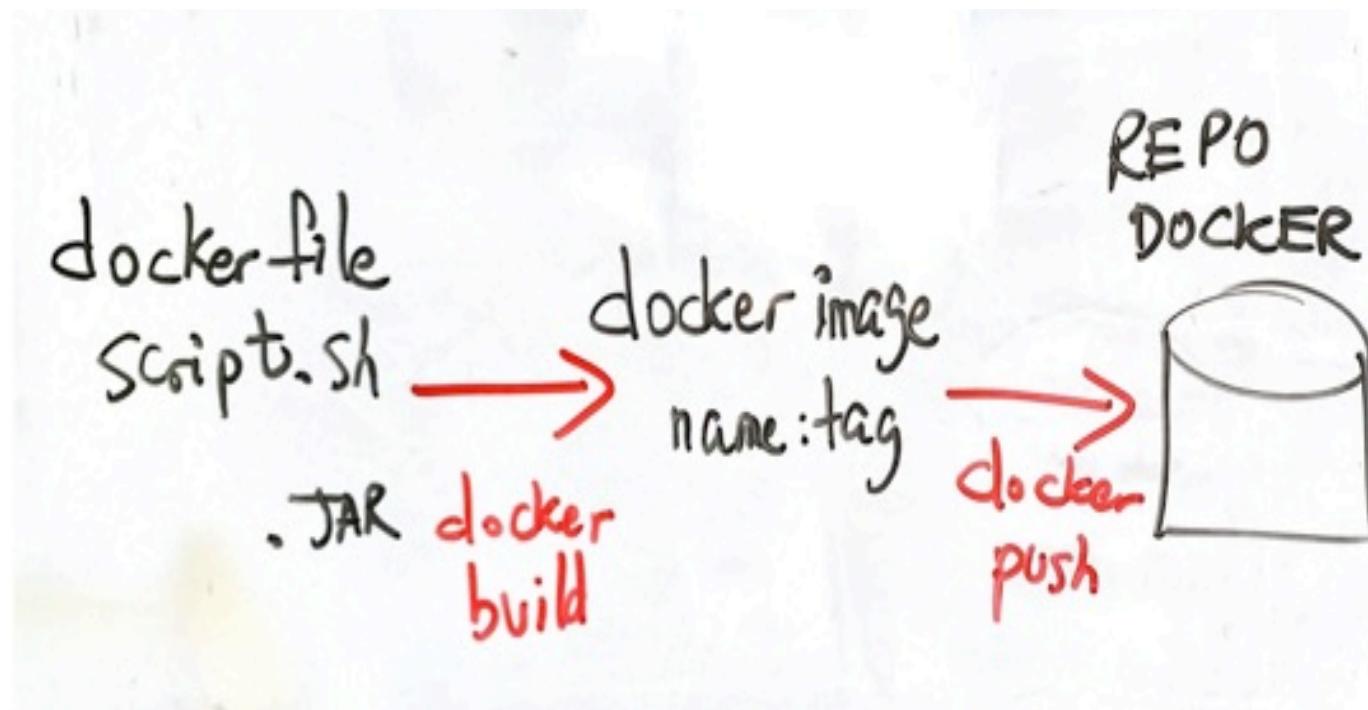
Componentes



Dockerfile: comandos a ejecutarse en el Container
Docker Image: contruido a partir del dockerfile
Docker Container: cuando se ejecuta un Docker Image

Laboratorio

Docker build



Nc command

