



INFRASTRUCTURE DE PRODUCTION

—

EXPOSER UNE
MULTITUDE DE
COMPOSANTS

00.00.2017

Photo by Diogène Morin on [Unsplash](#)

TABLE DES MATIÈRES

1 | PRÉPARER SON APPLICATION

- A. ARCHITECTURE MICROSERVICES
- B. Les 12 facteurs

2 | SPRING CLOUD

2 | REVERSE PROXY

Photo by Martha Dominguez de Gouveia on [Unsplash](#)

SCALABILITÉ

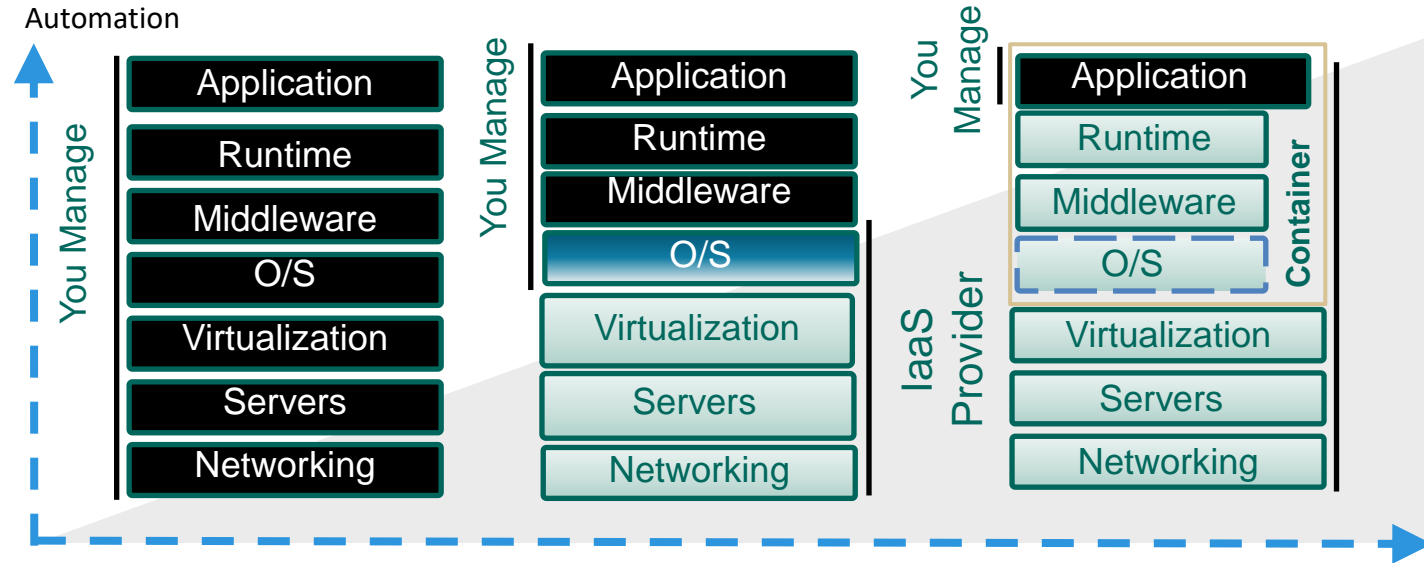


- ***Stateless vs stateful n'est plus une question dans le cloud***

- *La scalabilité est essentielle: vous devez être stateless*
- *Votre application doit s'arrêter et démarrer rapidement*

ABSTRACTION

Agility and
Cost Savings





1

PRÉPARER SON APPLICATION POUR LE CLOUD

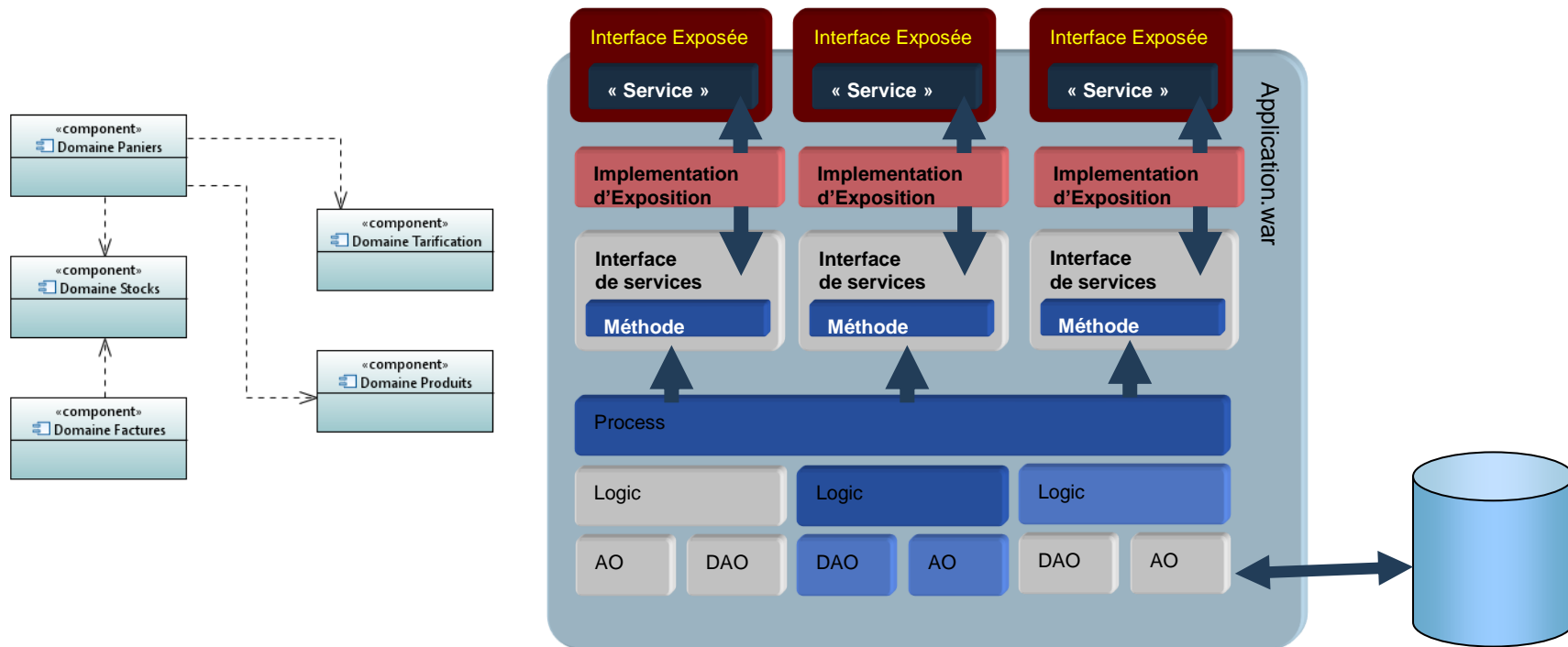
MICRO SERVICES



LES CLÉS D'UNE APPLICATION CLOUD READY

DU MONOLITHIQUE AUX ARCHITECTURES MICRO SERVICES

- UNE APPLICATION MONOLITHIQUE EST UNE APPLICATION QUI EST DÉVELOPPÉE EN UN SEUL BLOC (WAR, JAR, EAR) ET DÉPLOYÉE D'UNE MANIÈRE UNITAIRE DANS UN SERVEUR D'APPLICATION



PROBLÈMES

- Elles **centralisent** tous les besoins fonctionnels
- Elles sont réalisées dans **une seule technologie**
- Chaque modification nécessite de :
 - Tester les **régressions**
 - Redéployer **toute l'application**
- Difficile à faire évoluer au niveau fonctionnel
- Livraison **en bloc** (Le client attend beaucoup de temps pour commencer à voir les premières versions)

ARCHITECTURE MICROSERVICE

- Les microservices sont très récents (3 à 4 ans). Théorisé par Martin Fowler dans de nombreux articles sur la componentization des applications logicielles
- Concevoir une application unique basé sur un ensemble de **petits services indépendants**
- Communique via un protocole léger, le plus souvent à base de ressource HTTP
- Chaque élément correspond à un service (componentization), Chaque service est **responsable d'une fonctionnalité**
- L'objectif principal est de pouvoir déployer les microservices indépendamment
 - DURS (Deploy, Update, Replace, Scale) indépendants : Chaque service peut être indépendamment déployé, mis à jour, remplacé, scalé

“Loosely coupled service oriented architecture with bounded contexts”

BÉNÉFICES

- **Scaling indépendant** : les services les plus sollicités (cadence de requête, mutualisation d'application) peuvent être scalés indépendamment (CPU/mémoire ou sharding)
- **Mise à jour indépendantes** : Les changements locaux à un service peuvent se faire sans coordination avec les autres équipes
- **Maintenance facilitée** : Le code d'un micro-service est limité à une seule fonctionnalité
- **Hétérogénéité des langages** : Utilisation des langages les plus appropriés pour une fonctionnalité donnée
- **Isolation des fautes** : Un service dysfonctionnant ne pénalise pas obligatoirement le système complet
- **Communication inter-équipe renforcée** : Full-stack team

FACTEURS INDISPENSABLES À LA COMPONENTIZATION

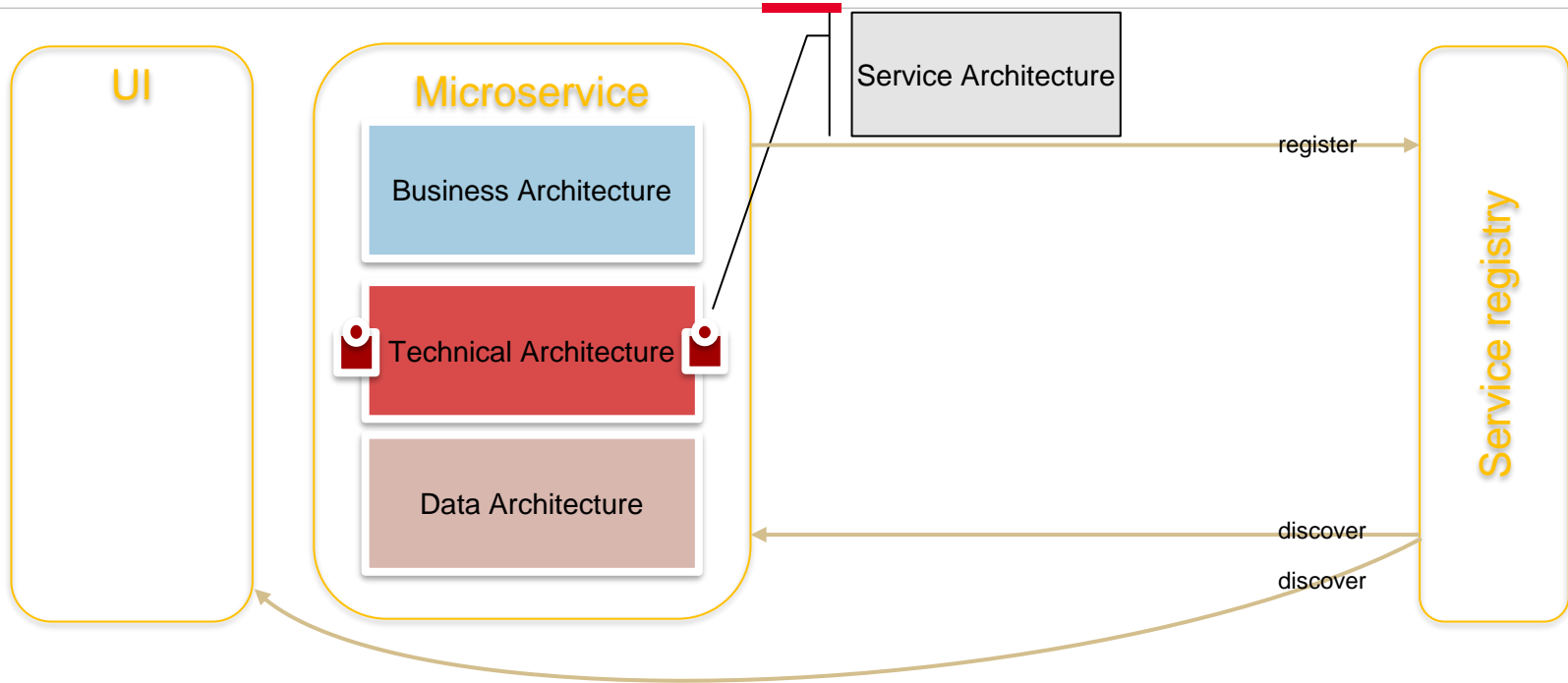
Facteurs

- **Cohérence** et logique du système : tout service doit correspondre à une fonctionnalité précise ;
- **Spécificité des fonctionnalités** : chaque service ne fait qu'une chose et il la fait bien ;
- **Autonomie des services** : chaque service dispose de son propre code, gère ses propres données et ne les partage pas (pas directement en tout cas) avec d'autres services ;
- **Indépendance** des microservices : chaque microservice ne doit pas inclure de couplage fort avec un autre microservice.

Contrainte

- **Réplication** : Un micro-service doit être scalable facilement, cela a des impacts sur le design (stateless, etc...)
- **Découverte automatique** : Les services sont typiquement distribués dans un environnement PaaS. Le scaling peut être automatisé selon certaines métriques. Les points d'accès aux services doivent alors s'enregistrer dans un annuaire afin d'être localisés automatiquement
- **Monitoring** : Les services sont surveillés en permanence. Des traces sont générées et éventuellement agrégées
- **Résilience** : Les services peuvent être en erreur. L'application doit pouvoir résister aux erreurs.
- **DevOps** : L'intégration et le déploiement continu sont indispensables pour le succès.

COMPOSITION



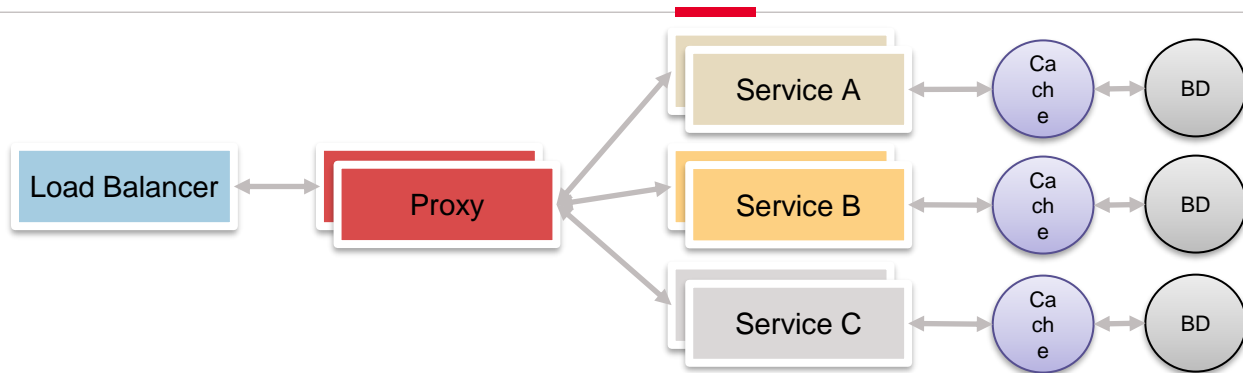
- Un micro service combine les trois couches « **métier, technique et données** » en une seule, accessible via des API.

DESIGN PATTERNS

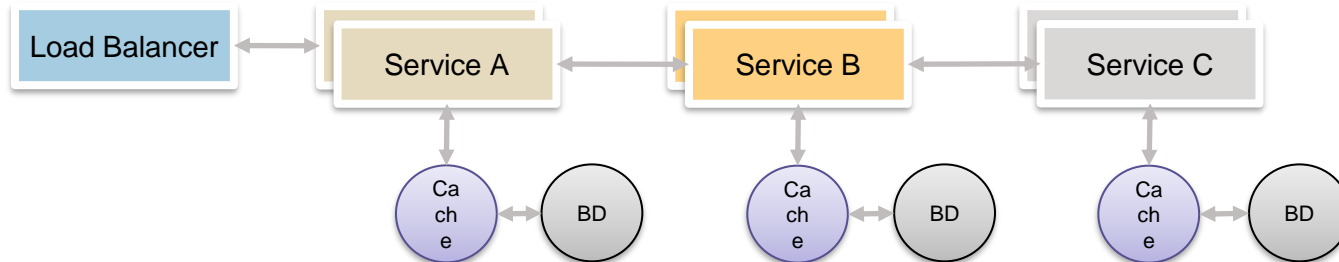
- Stateless
- Abstraction
- API Gateway
- Service Registry
- Config Server
- Circuit Breaker
- 12 factors
- **L'agrégateur** : Agrégation de plusieurs microservice et fourniture d'une autre API RESt
- **Proxy** : Délégation à un service caché avec éventuellement une transformation
- **Chaîne** : Réponse consolidée à partir de plusieurs sous-services
- **Branche** : Idem agrégateur avec le parallélisme

PATTERNS

Proxy

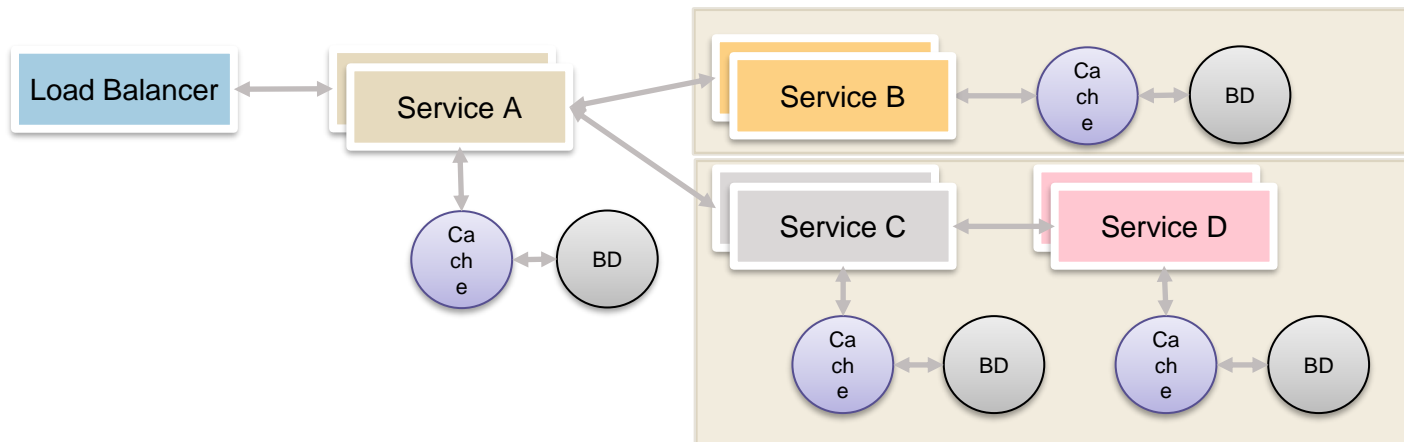


Chaine



PATTERNS

Branche



MICRO-SERVICES TECHNIQUES

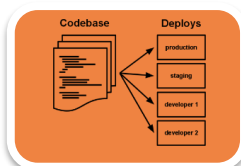
- Les architectures micro-services basées sur une distribution massive nécessitent des micro-services d'infrastructure :
- Service **d'annuaire** permettant de localiser un micro-service disponible
- Service de centralisation de **configuration** facilitant la configuration et l'administration des micro-services
- Services **d'authentification** offrant une fonctionnalité de SSO parmi l'ensemble des micro-services
- Service de **monitoring** surveillant la disponibilité, centralisant les fichiers journaux
- Service de **répartition** de charge, de gestion d'appartenance au cluster, d'élection de maître, ...
- Service de session applicative, horloge synchronisée,

12 FACTEURS



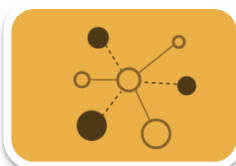
LES CLÉS D'UNE APPLICATION CLOUD READY

THE TWELVE-FACTOR APP



I. Codebase

- One codebase tracked in revision control, many deploys



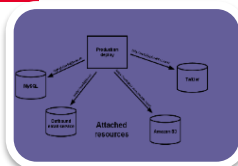
II. Dependencies

- Explicitly declare and isolate dependencies



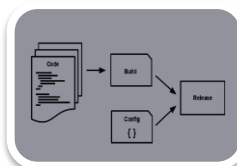
III. Config

- Store config in the environment



IV. Backing services

- Treat backing services as attached resources



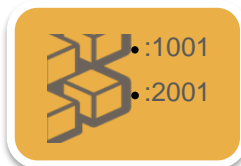
V. Build, release, run

- Strictly separate build and run stages



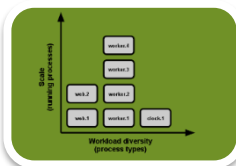
VI. Processes

- Execute the app as one or more stateless processes



VII. Port binding

- Export services via port binding



VIII. Concurrency

- Scale out via the process model



IX. Disposability

- Maximize robustness with fast startup and graceful shutdown



X. Dev/prod parity

- Keep development, staging, and production as similar as possible



XI. Logs

- Treat logs as event streams



XII. Admin processes

- Run admin/management tasks as one-off processes

■ Une application « cloud » ne s'improvise pas

■ L'application « 12 Factor »


- <http://12factor.net/>
- Méthodologie documentant 12 pratiques qui permettent de créer un logiciel « as a service »
- Couvre le développement, le déploiement, la gestion de la production

RÉPARTITION




Arrêt & démarrage

- Processes
- Disposability




élasticité

- Processes
- Concurrency
- Logs
- Admin processes



Plomberie technique

- Config
- Backing services
- Port binding
- Logs

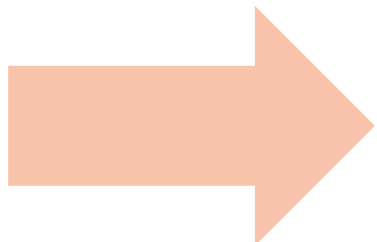
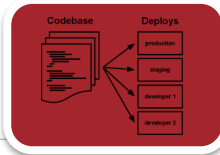


Création & installation

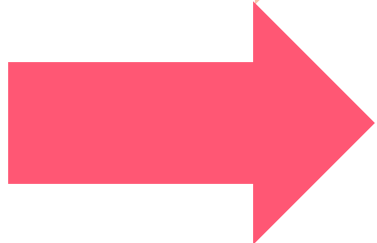
- Code base
- Dependencies
- Build, release, run
- Dev / prod parity



I – BASE DE CODE



Centralisation du code pour un composant
Plusieurs déploiement possible

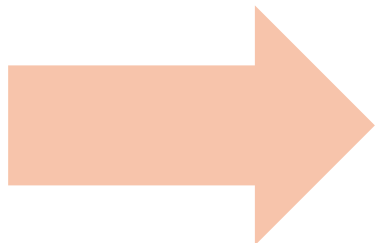


La duplication de code pour partager des
fonctionnalités est une violation

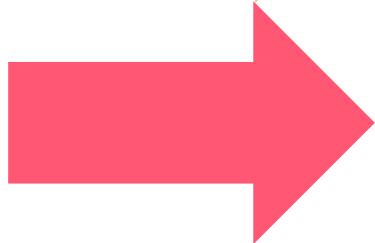


Un composant = une seule zone de stockage pour gérer le versionning de code

II – DÉPENDANCES



Le développeur est autonome pour sa compréhension du périmètre du composant.



Appuyer sur l'existence implicite d'outils système, par exemple ImageMagick, curl ou télécharger un jar est une violation

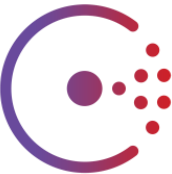
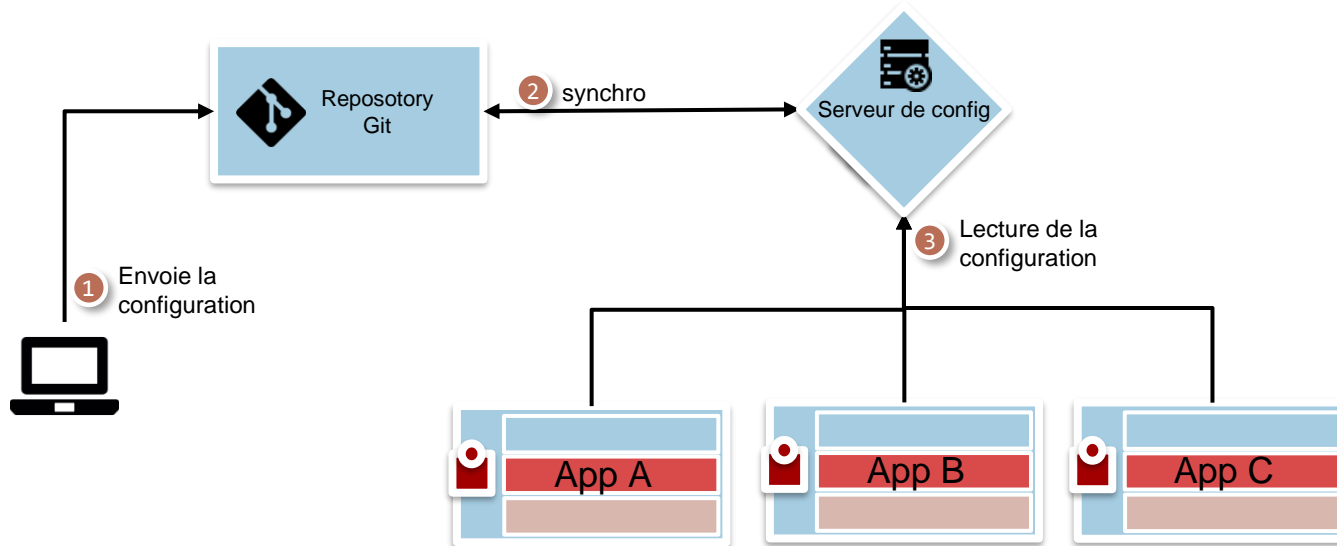


Les dépendances doivent être déclarées et isolées

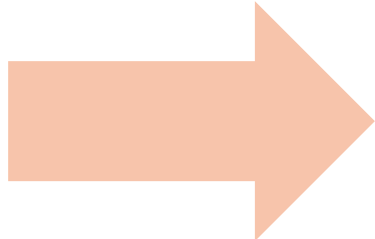
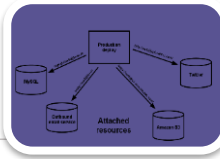


Découplage du code et de l'environnement d'exécution

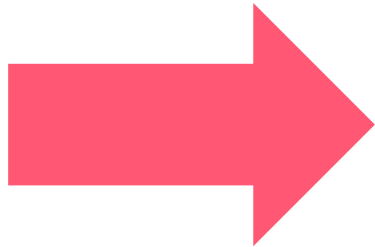
La configuration est stockée dans l'environnement



IV – SERVICES EXTERNES



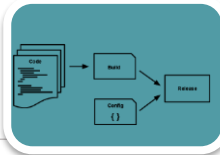
Pas de distinction entre les services locaux et les services tiers.



Aucune modification de code ne doit être nécessaire pour passer d'une ressource local à une ressource distante



Les ressources doivent être traiter comme des services



Séparation entre les différentes étapes de build et exécution des composants

3 étapes :

- L'étape d'assemblage (ou "build") = le code + les dépendances
- L'étape de publication (ou "release") = le build + la configuration
- L'étape d'exécution (ou "runtime") = exécution de la release

Aucune intervention humaine entre les 3 étapes

Il est impossible de faire des changements dans le code au moment de son exécution, car il n'y a pas moyen de propager ces changements vers l'étape de build.



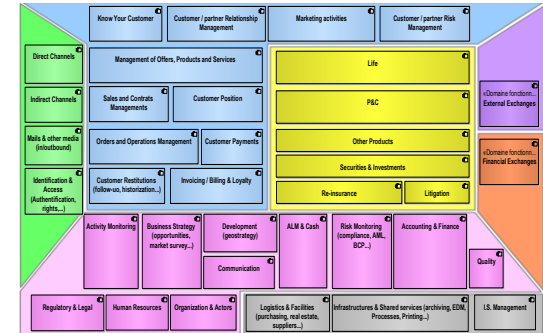
Jenkins



Les composants doivent être sans état

Les composants ne partagent rien. Si un stockage est nécessaire il doit être fait dans une ressource statefull.

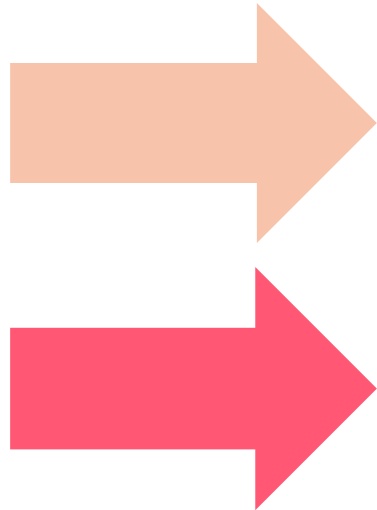
Les caches partagés sont une violation



VII – ASSOCIATION DE PORTS

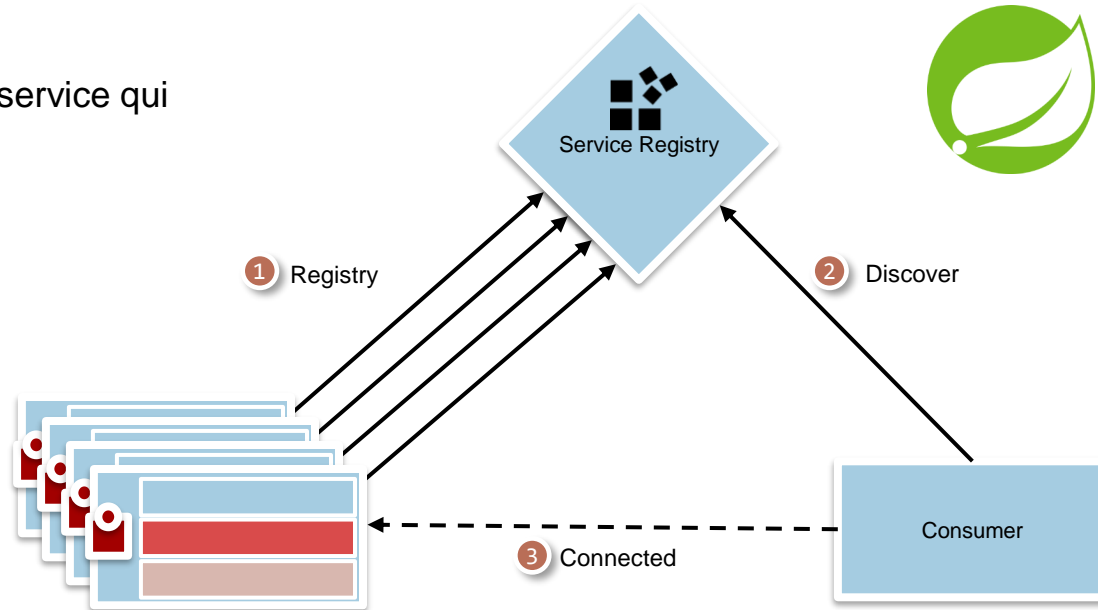


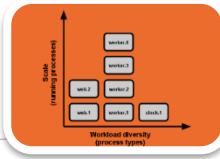
Tous les services doivent être considérés comme un service réseau



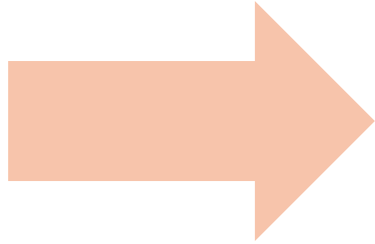
L'application fournit un service qui écoute sur un port

La gestion des ports ne doit pas être gérée par des opérationnels



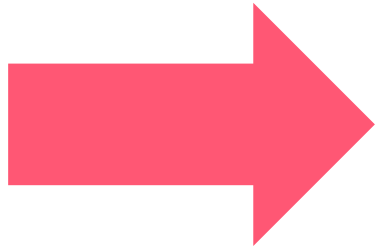


Capacité de montée à l'échelle



L'ajout plus de composant doit être une opération simple et fiable

- Instancier un nouveau container
- Lancer un nouveau processus

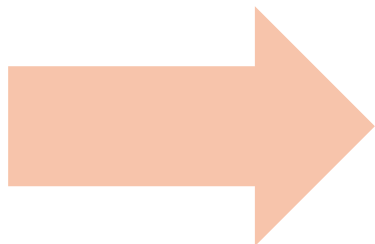


Ne pas écrire les flux de sortie dans des fichiers

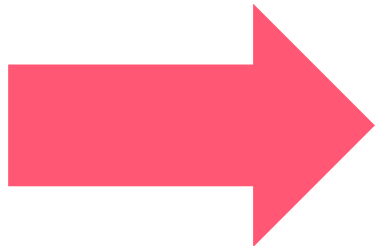




Les composants doivent être compatible avec l'infrastructure qui la supporte



Pet vs cattle
Démarrage rapide
Arrêt fluide
Robuste au crash



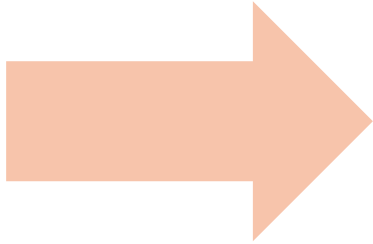
Pas de chargement de données au démarrage



OPENSIFT

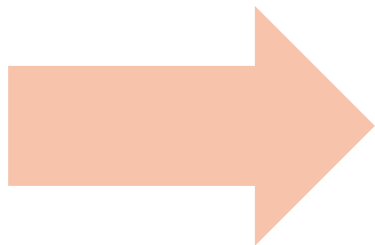


Les environnements doivent être alignés



Tous les environnements doivent être les plus proches possible

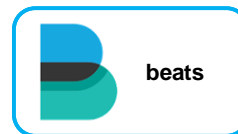




Les applications doivent externaliser leurs journaux pour la visualisation et l'archivage à long terme ELK



Traitez les logs comme des flux d'évènements



beats



logstash



es-hadoop



elasticsearch



x-pack

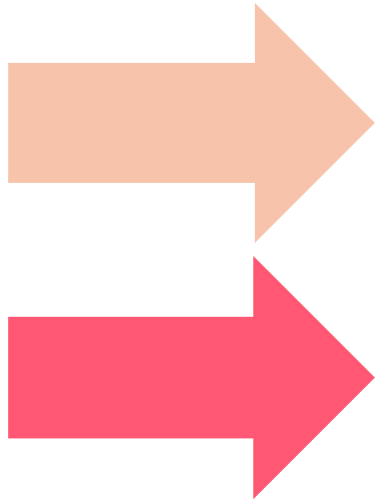


kibana



x-pack

XII – PROCESSUS D'ADMINISTRATION



Doter les composants d'outils pour permettre leur administration et leur supervision

Modifier la configuration en base de données manuellement est une violation



Lancez les processus d'administration et de maintenance de manière unique

MONITORING

■ Le monitoring et les logs sont essentiels dans le cloud

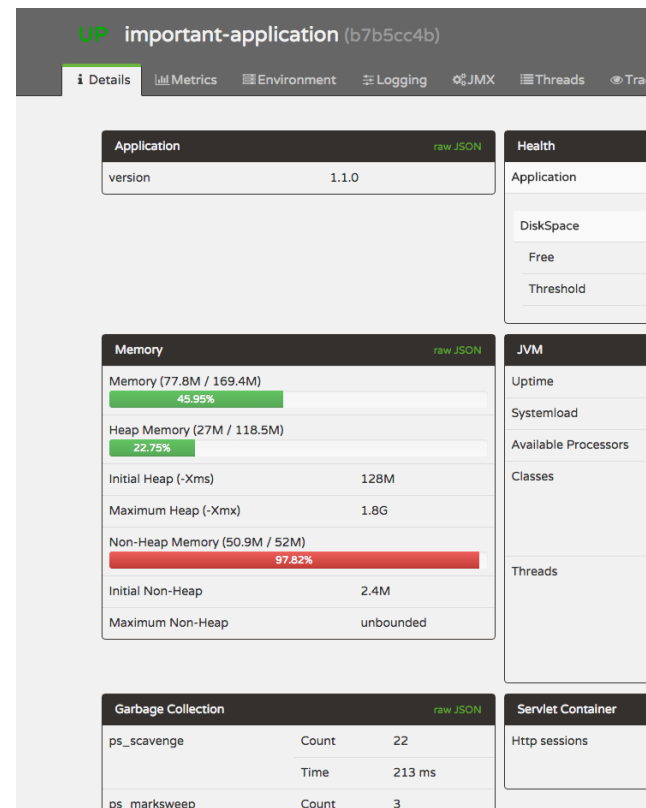
- Feedback rapide sur l'état des applications et du cloud

■ Suivi des métriques

- JVM, requêtes HTTP, pool de connexion, cache
- Statistiques avancées sur les objets Java et les Beans Spring

■ Santé des ressources externes

- Base de données, serveur de mail






2

SPRING CLOUD

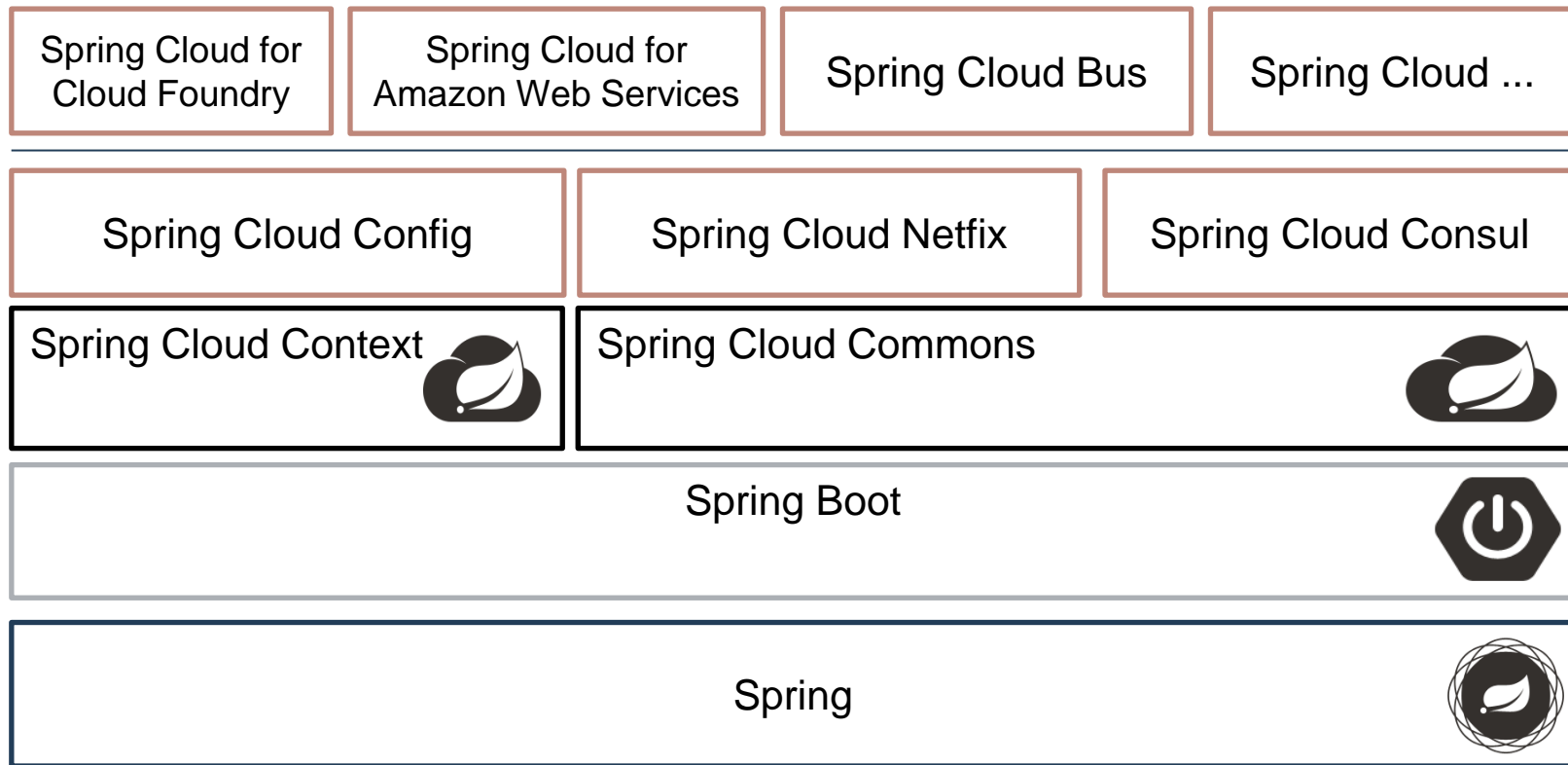


SPRING



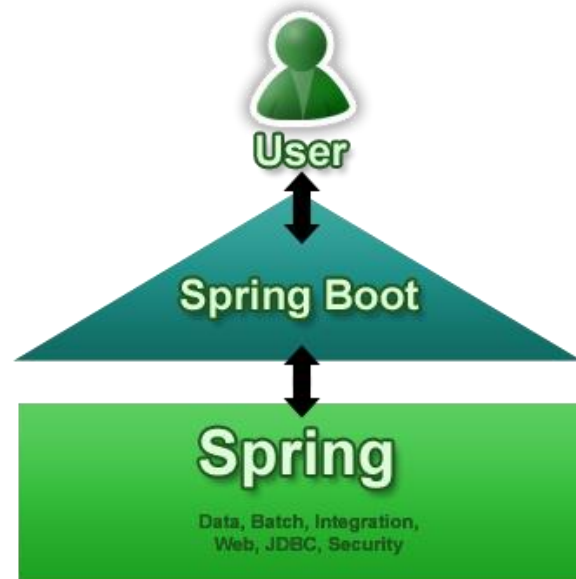
LES MODULES

SPRING CLOUD



SPRING BOOT ?

- **Sur-couche** à spring framework
- Permet de créer des **applications** basées sur spring core
- Orienté sur la simplicité de la gestion des **dépendances**
 - Utilisation de « starter » pour ajouter une fonctionnalité
- Orienté sur la simplicité de la **configuration**
 - Généralisation des annotations et utilisation des autoconfigure
 - Utilisation de fichiers de configuration
- Application **Standalone**. Ne nécessite pas d'installation de serveur d'application
- Pas de génération de code



TOUS LES TYPES D'APPLICATION

- Web
- command line
- Batch
- redis/gemfire/jpa/elasticsearch
- Integration
- JMS, AMQP
- etc

```
@Component
public class Startup implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception
    {
        System.out.println("Hello World!");
    }
}
```

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project ▾ with Java ▾ and Spring Boot 1.5.7 ▾

Project Metadata

Artifact coordinates

Group

com.example



Artifact

demo

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Security ×

Config Client ×

Consul Configuration ×

JPA ×

MySQL ×

Generate Project alt + ↵

Don't know what to look for? Want more options? [Switch to the full version.](#)

UNE APPLICATION SPRINGBOOT

```
@Configuration
@EnableAutoConfiguration
@ComponentScan
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Classe principale

```
@RestController
@RequestMapping("/api")
public class RestServices {
    @RequestMapping(method = RequestMethod.GET)
    public @ResponseBody String ping() {
        return "{\"status\":\"OK\",\"timestamp\":\"" +
System.currentTimeMillis() + "\"}";
    }
}
```

Services REST

Voilà, c'est tout !

LES STARTER

■ Définir des collections de dépendances pour un usage donné

- Pattern commun : spring-boot-starter-*

■ Support de Maven et Gradle

spring-boot-starter	Core du composant, inclus auto-configuration, logging, etc
spring-boot-starter-web	Support pour les applications web (spring-mvc et tomcat)
spring-boot-starter-tomcat	Support pour le conteneur web tomcat
spring-boot-starter-test	Support pour les dépendances de test
spring-boot-starter-actuator	Fonctionnalités pour la production (métrique, monitoring)
spring-boot-starter-data-jpa	Support pour JPA, Spring Data et Hibernate
spring-boot-starter-batch	Support du module spring-batch en utilisant HSQLDB
spring-boot-starter-security	Support du module spring-security

AUTOCONFIGURE

```
@Configuration
@ComponentScan
@EnableAutoConfiguration
public class MyApplication { }
```

@EnableAutoConfiguration

```
@SpringBootApplication
public class MyApplication { }
```

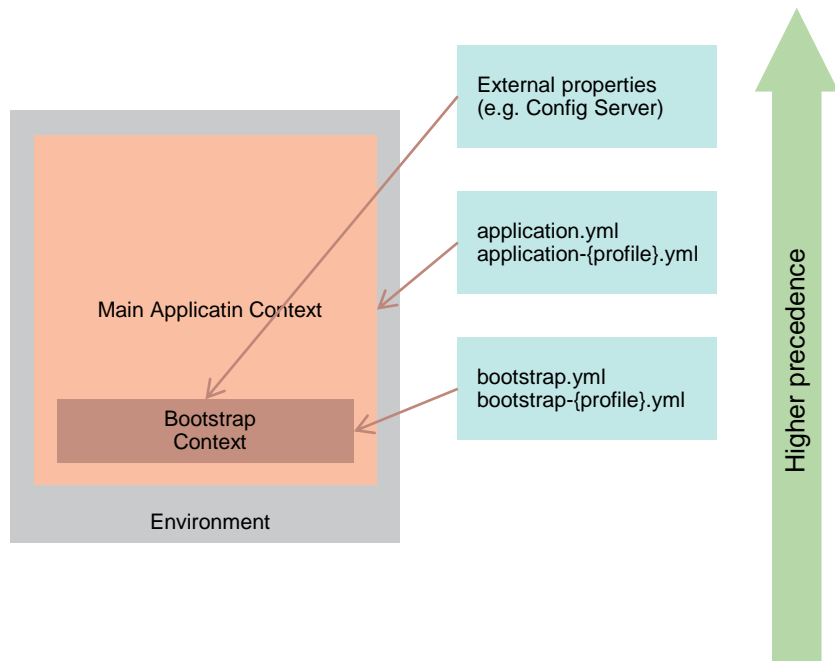
- Essaye de configure automatiquement votre application
- Si vous définissez vos propres beans, l'autoconfigure ne démarre pas
- Définie par @Configuration classes, @Conditional* annotations

```
@Configuration
@Conditional(CustomCondition.class)
public class AppConfiguration {
    // @Bean methods
}
```

@Conditional

```
// defined in Boot
@ConditionalOnClass(ObjectMapper.class)
@ConditionalOnMissingBean("aBeanName")
```

LES FICHIERS DE CONFIGURATION



Bootstrap Context

- Parent of Main Application Context
- Used to load properties from external sources
- Out of the box loads properties from Config Server
- Can be configured to do anything you want
- Handling environment changes
 - Re-bind @ConfigurationProperties
 - Set log levels for logging.level.*
 - @RefreshScope

ACTUATOR

- Voici une liste non exhaustive des indicateurs disponibles par défaut :
- **metrics**: métriques de l'application (CPU, mémoire, ...)
- **beans**: liste des BEANS Spring
- **trace**: liste des requêtes HTTP envoyées à l'application
- **dump**: liste des threads en cours
- **health**: état de santé de l'application
- **env**: liste des profils, des propriétés et des variables d'environnement

AJOUTER LA SÉCURITÉ

■ Ajouter le starter dans le pom Maven

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

■ Ajouter la configuration de la sécurité

```
@Configuration
public class SecurityConfig extends GlobalAuthenticationConfigurerAdapter {

    @Override
    public void init(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("hero").password("hero").roles("HERO", "USER").and()
            .withUser("user").password("user").roles("USER");
    }
}
```

LES TESTS

■ Ajouter la dépendance spring-boot-starter-test

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

■ Développer les tests avec spring-test

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@SpringApplicationConfiguration(classes = Application.class)
public class HomeControllerTest {

    @Autowired private WebApplicationContext wac;

    @Test public void testHome() throws Exception {

        MockMvc mvc = webAppContextSetup(this.wac).build();
        mvc
            .perform(get("/"))
            .accept(MediaType.TEXT_PLAIN)
            .andExpect(status().isOk())
            .andExpect(content().string("Hello, world."));
    }
}
```

SPRING CLOUD

- **Spring cloud propose des starter pour faciliter l'écriture d'application native cloud**
- **Les principaux composants nécessaires dans un système distribué sont**
 - configuration distribué/versionné
 - Enregistrement et découverte de Service
 - Routage
 - Appel de Service-to-service
 - Load balancing
 - Circuit Breakers
 - Global locks
 - Election d'un master et gestion de l'état d'un cluster
 - Message Distribué

LES STARTER SPRING CLOUD

■ Définir des collections de dépendances pour un usage donné

- Pattern commun : spring-boot-starter-*

■ Support de Maven et Gradle

spring-cloud-starter-config	Importe les composants permettant l'utilisation de propriétés distribuées
spring-cloud-starter-eureka	Importe la stack Netflix (Eureka, ribbon, Hystrix, Archaius, Zuul)
spring-cloud-starter-consul-all	Importe les composants liés au produit Consul
spring-cloud-starter-sleuth	Importe les composants permettant la mise en œuvre de traces distribuées
spring-cloud-zookeeper-discovery	Importe les composants liés au produit zookeeper
spring-cloud-starter-stream-kafka	Importe les composants de lecture des données dans Kafka
spring-cloud-dataflow-server-local	Ce n'est pas un starter mais une application

STACK NETFLIX



DÉPLOYER DES MICROSERVICES

LA STACK NETFLIX



**Service
Discovery**



Eureka

**REST
client**



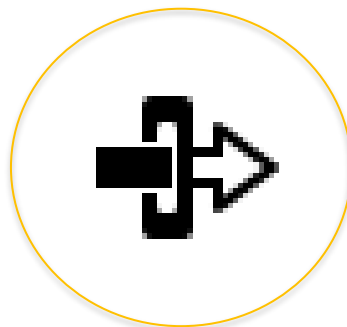
Ribbon

**Circuit
breaker**



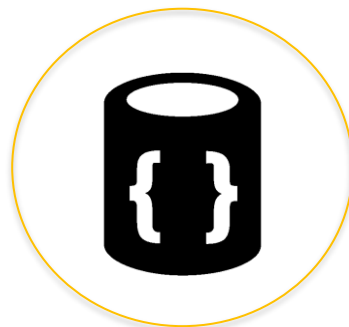
resilience4j

**Gateway
Service**



Zuul

**Dynamic
Properties**



Archaius



Service Discovery



Découvrir
l'emplacement
de services

Health Checking



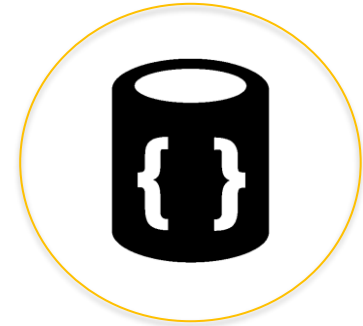
Monitor
Reroute les flux vers
les noeuds actifs

Multiple Datacenter



Support du multi-
datacenter en
standard

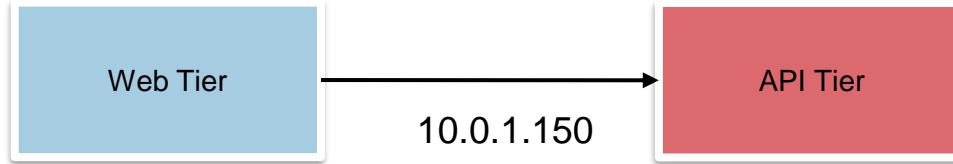
Key/Value Store



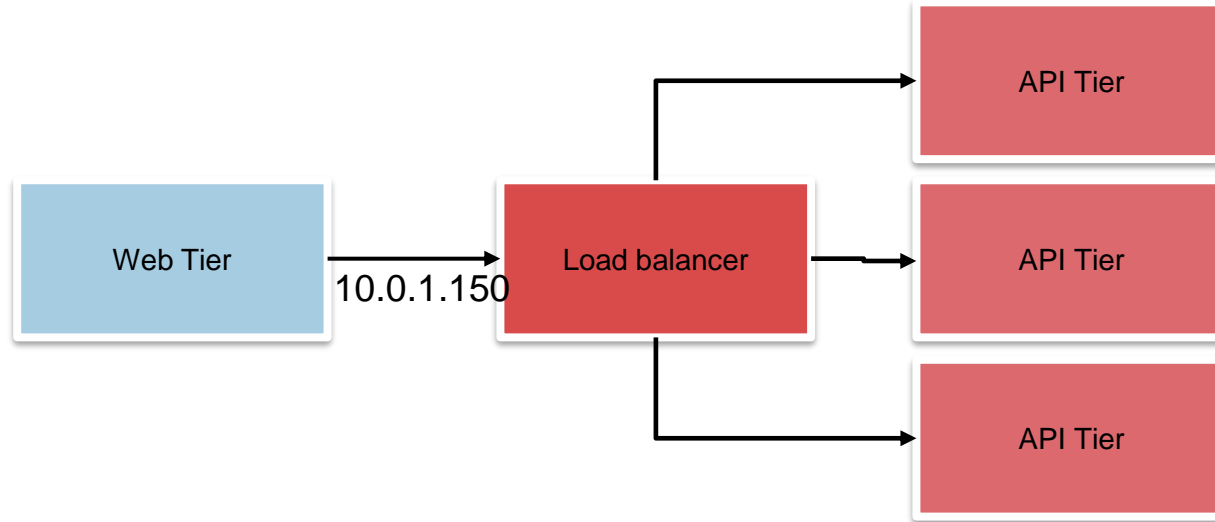
configuration dynamic
Coordination
Leader election



DISCOVERY

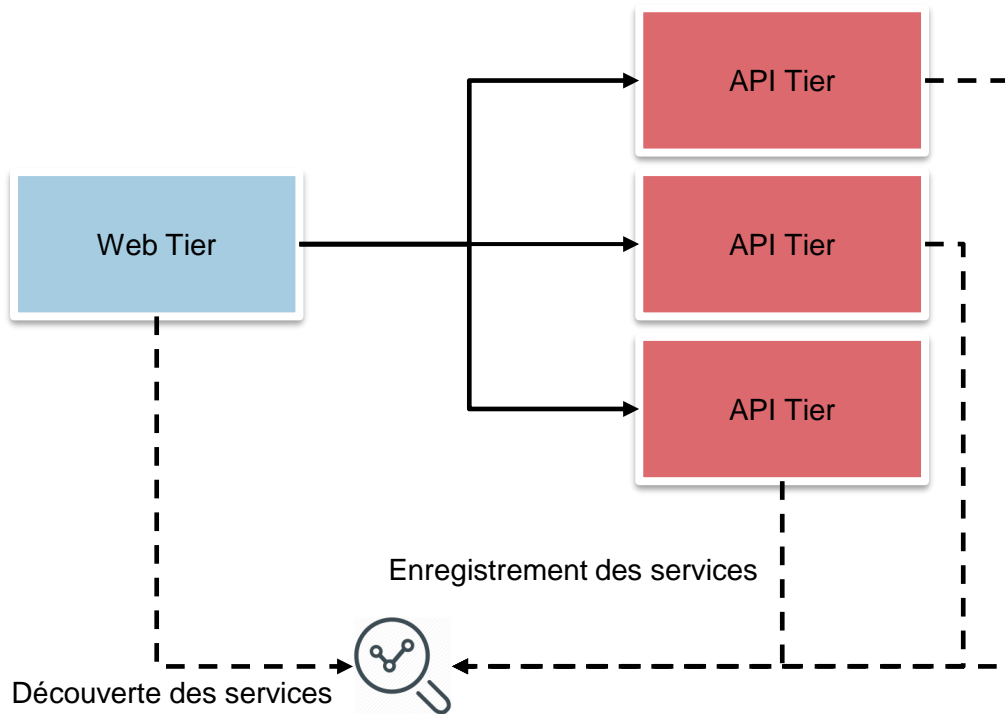


Adresse écrite en dur dans la configuration





DISCOVERY





DISCOVERY

■ Service simple via API REST

- Qui tourne ?
- Où ?
- Sur quelle IP ? Quel port ? TCP, UDP, les deux ?
- Quel est son nom ?

■ Interface DNS

- Round-robin DNS
- Filtre sur la santé des services

■ Vérification de la santé des services enregistrés

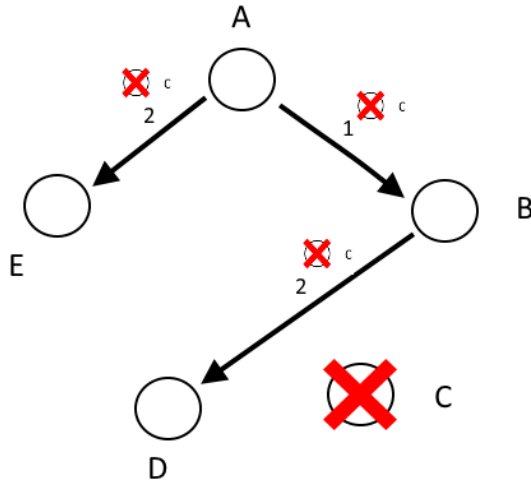
- Scalable
- Compatible avec les applications de monitoring du marché

DISCOVERY : FONCTIONNEMENT

- **Consul est construit autour d'un exécutable, l'Agent**
 - **L'Agent**
L'agent est un daemon qui s'exécute sur tous les membres du cluster Consul.
 - **Clients**
Les Clients sont des agents dont les seuls rôles sont de faire suivre les requêtes
 - **Serveur**
Les Serveurs ont la charge de participer à la gestion du consensus

- **Le GOSSIP Protocol est au cœur des échanges dans un cluster consul**

GOSSIP PROTOCOL OU EPIDEMIC PROTOCOL



Un ensemble de serveur sont dans un réseau. Ils se regroupent en petits groupes aléatoires et partagent la même donnée.

Au début de la journée, A échange avec le B une information à propos de C.

A la réunion suivante, B va partager cette information avec D, tandis que A fera de même avec E.

On peut grossièrement dire qu'à chaque réunion, le nombre de serveurs informées de la donnée a doublé, jusqu'à atteindre l'ensemble des serveurs présentes.

La robustesse de ce protocole réside dans sa diffusion de l'information. En effet, si D n'a pas compris ce que B lui a dit, il sera informé lors d'une des réunions suivantes.



DISCOVERY

< All Nodes

machine

127.0.0.1

Health Checks Service Instances Lock Sessions Metadata

✓ Serf Health Status

NodeName	CheckID	Type	Notes
machine	serfHealth		-

Output

Agent alive and reachable



✓ Service 'petclinic-admin' check

ServiceName	CheckID	Type	Notes
petclinic-admin	service:petclinic-admin-8c0abedb66715831577d4b1358969ba8	http	-

Output

HTTP GET http://host.docker.internal:8080/actuator/health: 200 OK Output: {"status":"UP"}





KEY/VALUE STORAGE

- A flexible key/value store
 - enables storing dynamic configuration,
 - feature flagging,
 - coordination,
 - leader election
-
- The simple HTTP API makes it easy to use anywhere.



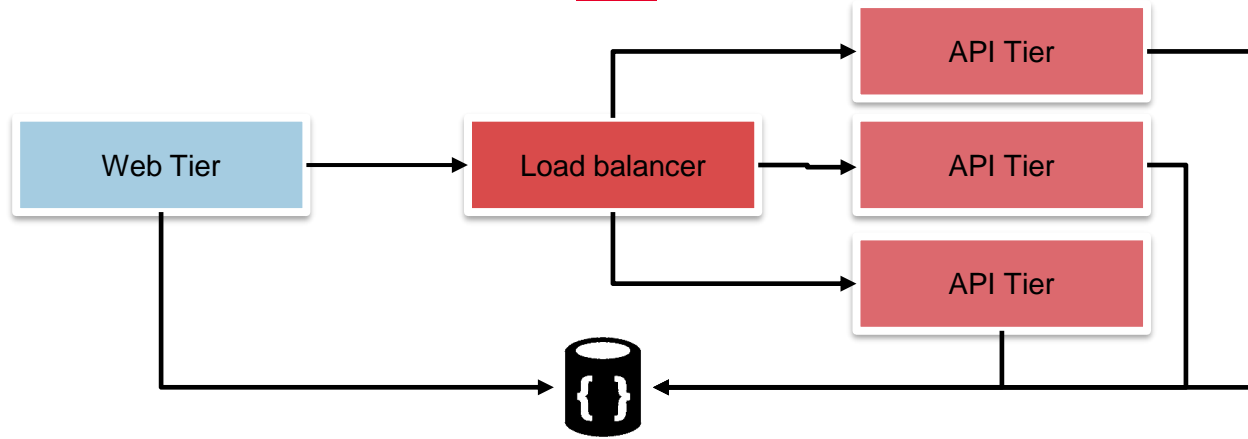
CONFIGURATION DISTRIBUÉ/VERSIONNÉ



- Dans une installation classique chaque application possède ses fichiers de propriétés
 - Données applicatives
 - Données environnementales
- Dans le cloud il faut pouvoir lancer à la volé une nouvelle instance
- Dans le cloud, un composant peut être déployé de multiple fois
- Comment appliquer une modification de propriétés à toutes les instances



CONFIGURATION DISTRIBUÉ/VERSIONNÉ



- La configuration est centralisée
- Seul l'information sur le serveur est déployé en même temps que les composants



CONFIGURATION DISTRIBUÉ/VERSIONNÉ

Config Server

- HTTP, API de lecture des ressources
- Support des formats JSON/YML/properties
- Possibilité d'utiliser Git (default)
- Intégré a spring security

Config Client

- Config-first bootstrap
- Discovery-first bootstrap
- Fail-fast option
- Like reading local application*.yml family with extra dimation "label"

`/ {application} / {profile} / {label}`
`/ {application} / {profile} [/ {label}]`
`/ {application} - {profile} . yml`
`/ {label} / {application} - {profile} . yml`
`/ {application} - {profile} . properties`
`/ {label} / {application} - {profile} . properties`

`${spring.application.name}`
`${spring.profiles.active}`
`master`

`spring.cloud.config.[name|env|label]`



CONFIGURATION DISTRIBUÉ/VERSIONNÉ DANS SPRING BOOT

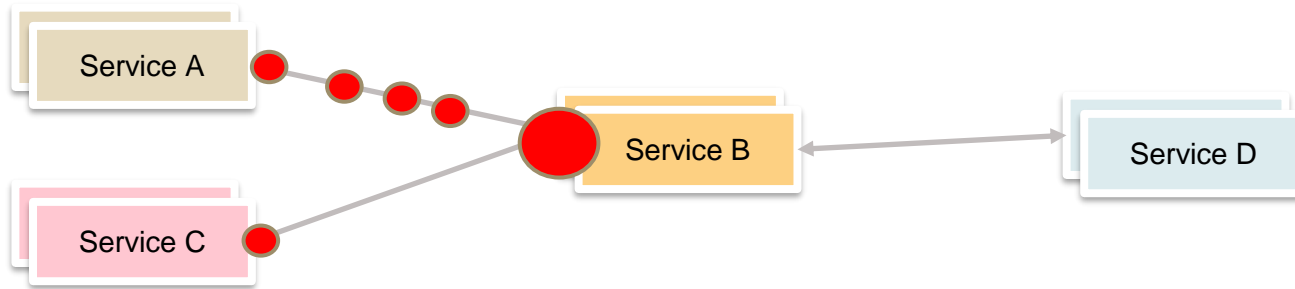
```
@SpringBootApplication
@EnableDiscoveryClient
public class SpringCloudConsulStudentApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringCloudConsulStudentApplication.class, args);
    }
}
```

```
spring:
  application:
    name: kivala-server
  cloud:
    consul:
      host: hostname.domaine
      port: 8500
    discovery:
      healthCheckPath: ${server.contextPath}/health
      health-check-interval: 15s
      scheme: https
      hostname: hostname.domaine
```

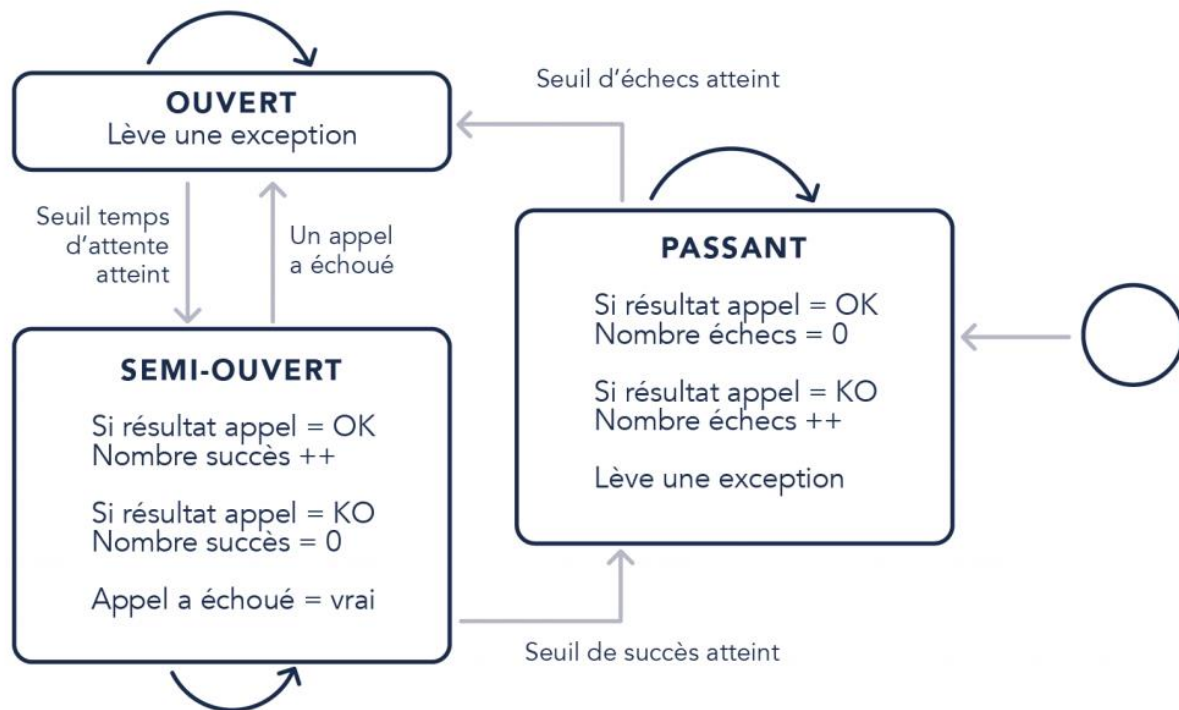
bootstrap.yml

CIRCUIT BREAKER



- Un client de service doit pouvoir invoquer un service distant a travers un proxy qui fonctionne de la même manière qu'un coupe circuit électrique

CIRCUIT BREAKER





RESILIENCE4J

- **Resilience4j est une librairie léger de gestion de tolérance d'erreur inspiré de hystrix mais écrit pour java 8 et en programmation fonctionnelle**

```
CircuitBreaker circuitBreaker = CircuitBreaker.ofDefaults("backendService");

Retry retry = Retry.ofDefaults("backendService");

Bulkhead bulkhead = Bulkhead.ofDefaults("backendService");

Supplier<String> supplier = () -> backendService.doSomething(param1, param2);

Supplier<String> decoratedSupplier = Decorators.ofSupplier(supplier)
    .withCircuitBreaker(circuitBreaker)
    .withBulkhead(bulkhead)
    .withRetry(retry)
    .decorate();
```

- **Circuit breaker**
- **Rate limiter**
- **Retry**
- **Cache**
- **...**

CONSOLE D'ADMINISTRATION

- Supervise votre application springboot et toutes ses interactions
- La console est une application springboot qui peut être démarré séparément
- En intégré dans l'application a superviser

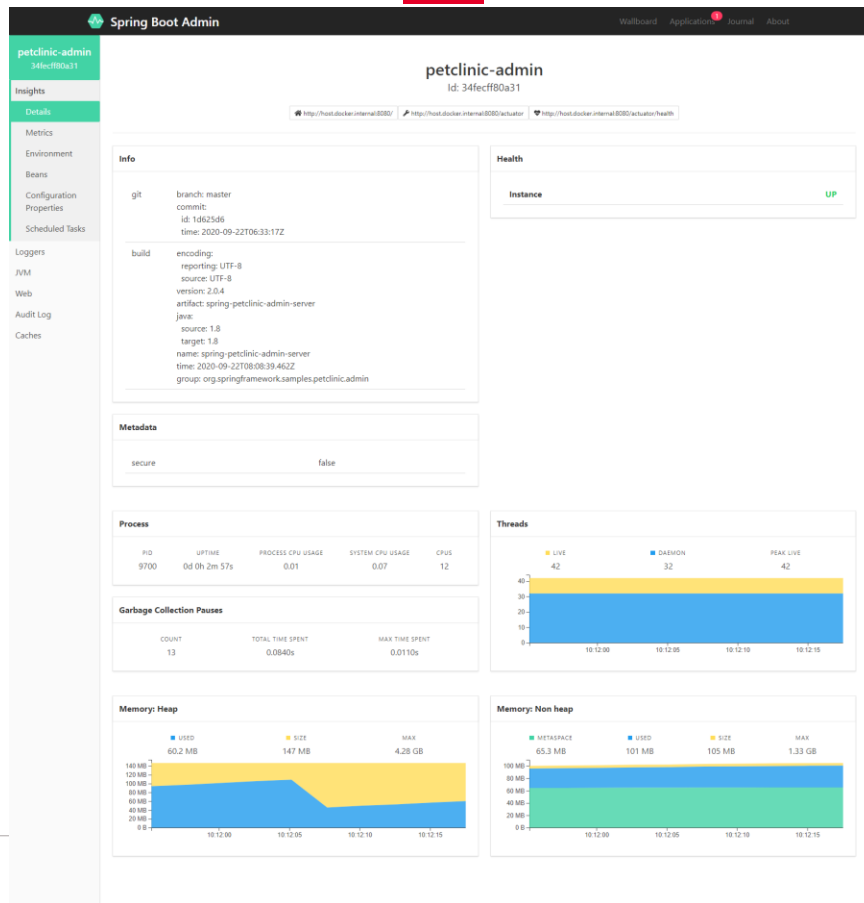
```
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-server-ui</artifactId>
  <version>1.5.0</version>
</dependency>
```

spring.boot.admin.url: <http://localhost:8080>

CONSOLE D'ADMINISTRATION

- **Changement à la volé du niveau de log (changement du niveau en mode debug par exemple)**
- **Utilisation et lecture des métriques JMX sur la performance**
- **Visualisation de toutes les métriques de l'application**
- **Visualisation de toutes les requêtes (header compris)**
- **Visualisation des éléments d'environnement, stack trace, ...**

- **Basiquement, toutes les informations exposés par actuator**





3

REVERSE PROXY

Photo by [Imgix](#) on [Unsplash](#)

CAS D'USAGE

- **Netflix utilise Zuul pour :**
 - **L'authentification et la sécurité**
 - **Des tests de stress**
 - **Canary Testing : Déploiement de nouvelles fonctionnalités sur un ensemble restreint d'utilisateurs**
 - **Du délestage de charge**
 - **Du routage dynamique**
 - **De la migration de service**
 - **Le traitement des réponses statiques**
 - **Gestion active du trafic**

ZUUL

- Zuul Routing (également Netflix) permet d'automatiquement mapper des routes (URLs) vers des services enregistrés
- => Pas de configuration
- => Peut ajouter des filtres, etc.
- Il inclut également un répartiteur de charge
- Les règles de routage sont exprimées dans des langages JVM (essentiellement Java et Groovy)

FONCTIONNEMENT PAR DÉFAUT

- Le proxy utilise **Ribbon** pour localiser un instance
- Toutes les requêtes sont exécutées dans une commande Hystrix
 - => Ainsi, les erreurs sont remontées dans les métriques Hystrix
 - => Une fois que le circuit est ouvert, le proxy n'essaie plus de contacter le service.
- Par défaut, un service nommé **users** reçoit les requêtes de **/users**

A person with dark hair and a beard, wearing a dark blue t-shirt and a black neck strap, is seen from behind, sitting at a dark wooden table in what appears to be a cafe or co-working space. They are using a silver laptop. To their left, another person's hands are visible typing on a laptop. To their right, another silver laptop is open, displaying a document with text and a sidebar. A small potted plant with green leaves sits on the table between the two laptops. In the background, other people are partially visible, and a water bottle is on the table. The overall atmosphere is casual and productive.

TP