Spring - Core Microservices

Marwen Saidi : marwen.saidi@orange.com

orange™

Spring-core

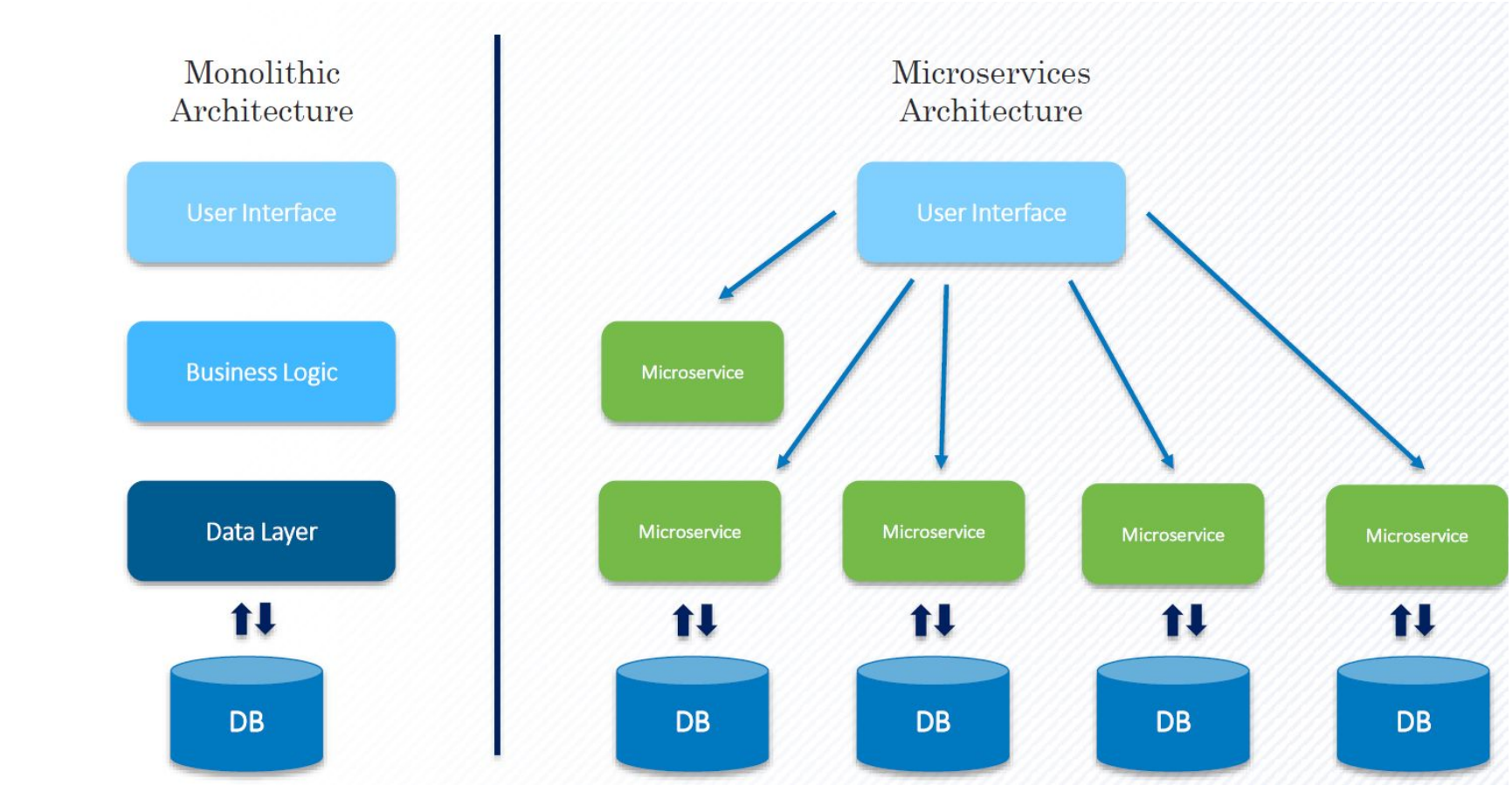
Spring-Boot

Microservices

une approche de l'ingénierie centrée sur le building de modules à fonction unique avec des interfaces et opérations.

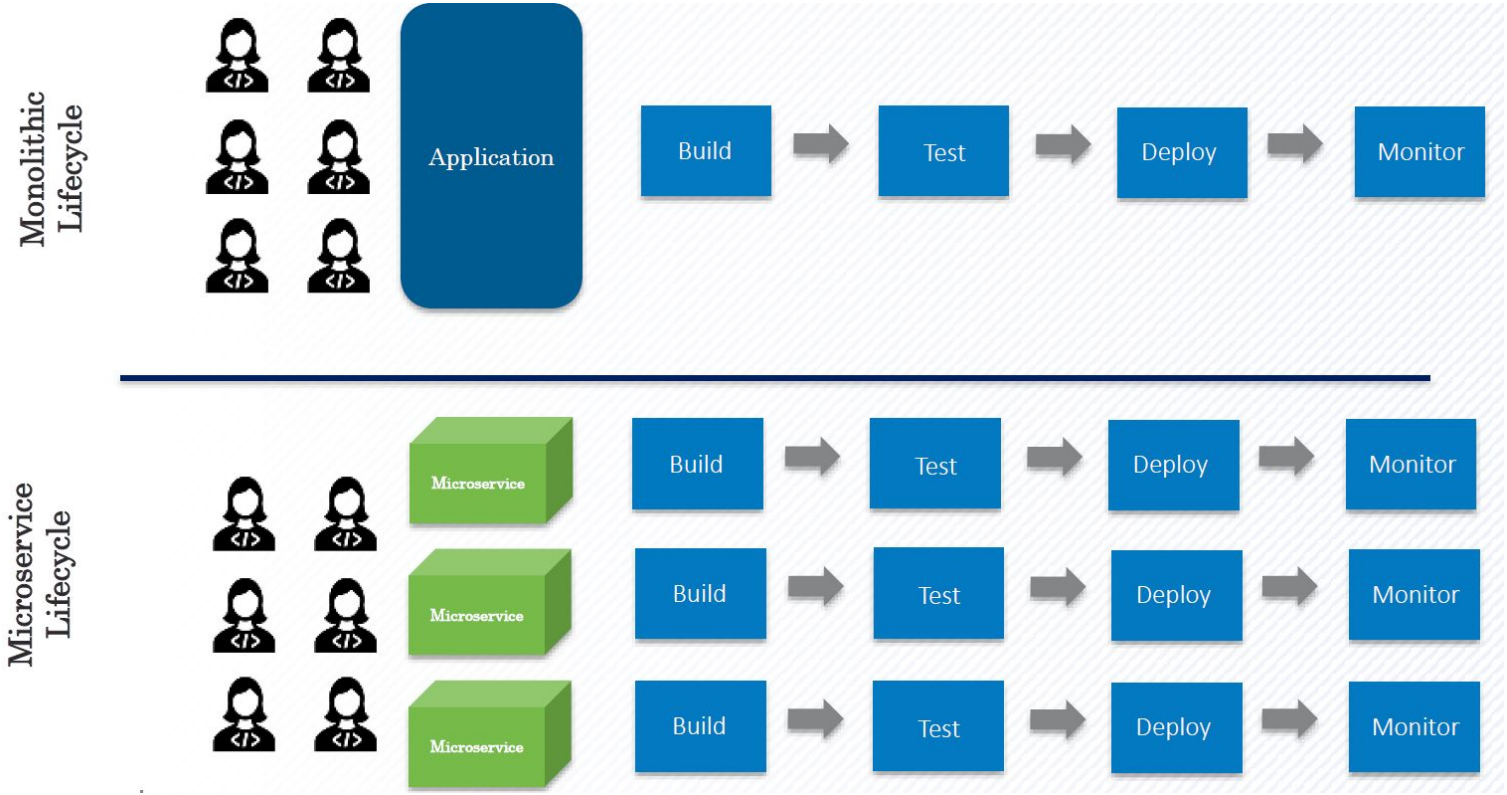
- Minimiser les risques et la portée du changement
- Facile à déployer
- Facile à comprendre dans l’ensemble du business

Microservices



Microservices

Do One Thing. And Do It Well.



Microservices

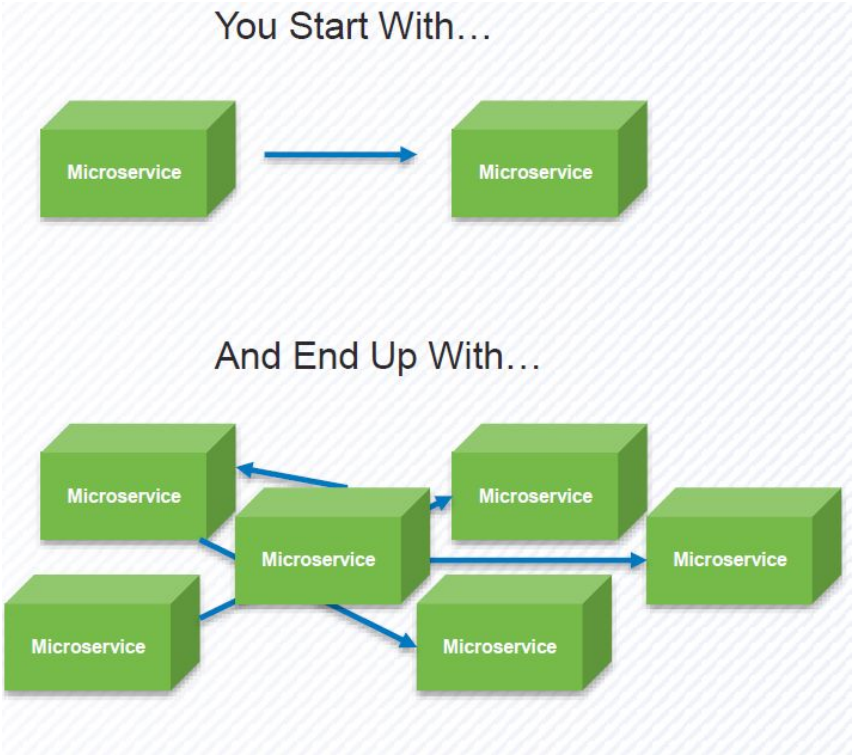
Avantages des microservices

- Séparation des services - Les services se concentrent sur une seule fonction
- Technologie facile à changer - Aucun engagement important envers une tech stack
- Simple à comprendre - Les équipes réparties peuvent comprendre plus facilement les petites fonctions
- Disponibilité accrue - Améliorer l'isolation des pannes et la résilience du système
- Réutilisabilité des services - Réutiliser les microservices dans votre organisation
- Données décentralisées - Chaque microservice est responsable de sa base de données
- Facile à déployer - Déployez littéralement en morceaux

Microservices

En plus de la complexité normale de l'API:

- Nombre exponentiel de connexions
- Précision des abstractions API
- Pas entièrement normalisé

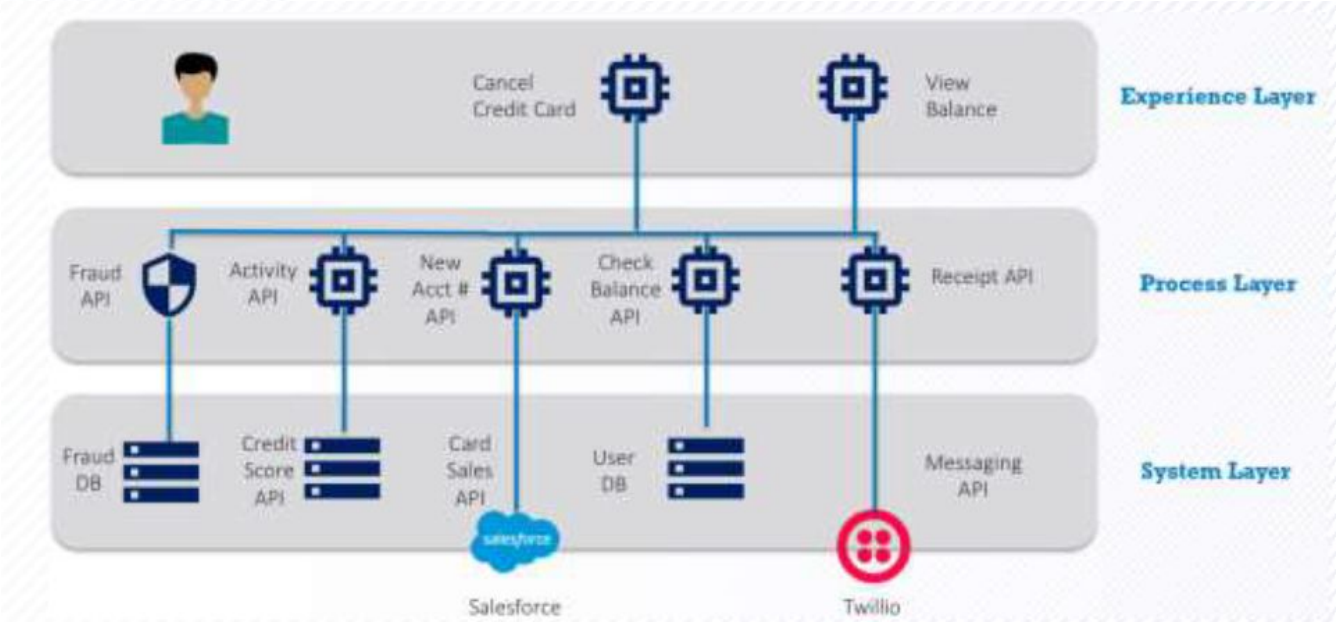


Microservices

Complexity | **Automation** | Availability | Performance

Test Automation

- Test fonctionnel
- Tester les states et les datas
- Chaînage de services
- Refactoring de test

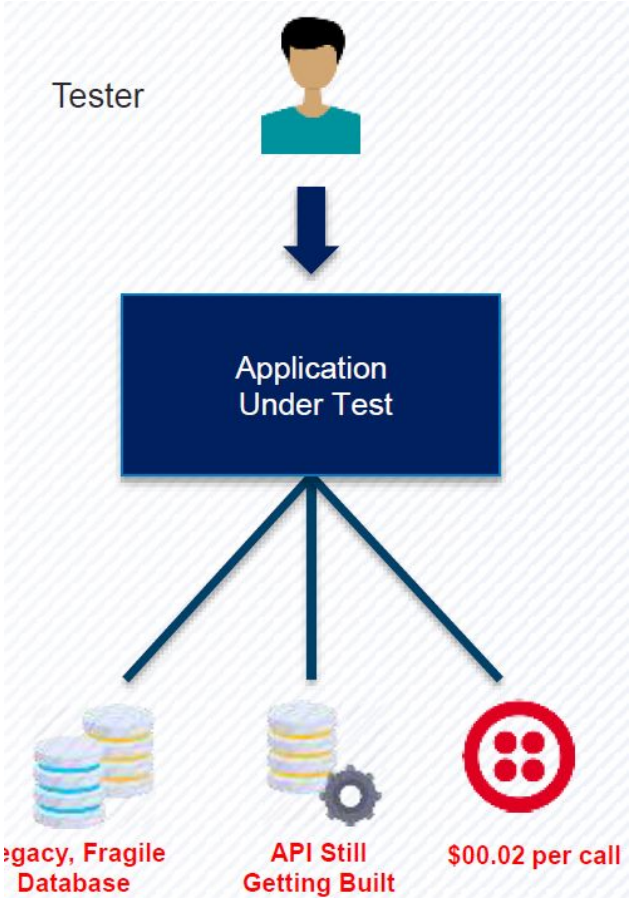


Microservices

Complexity | Automation | **Availability** | Performance

Availability

- 84% des équipes QA signalent des retards dans l'attente des services,
- composants, API, etc.
- 81% des équipes de développement signalent des retards en attente
- services, composants, API, etc.
- Accès aux systèmes requis
- Nombre moyen de systèmes nécessaires pour le développement/test = 52
- Nombre moyen de systèmes disponibles pour le développement/test = 32
- Temps d'attente moyen pour accéder aux systèmes requis = 32 jours

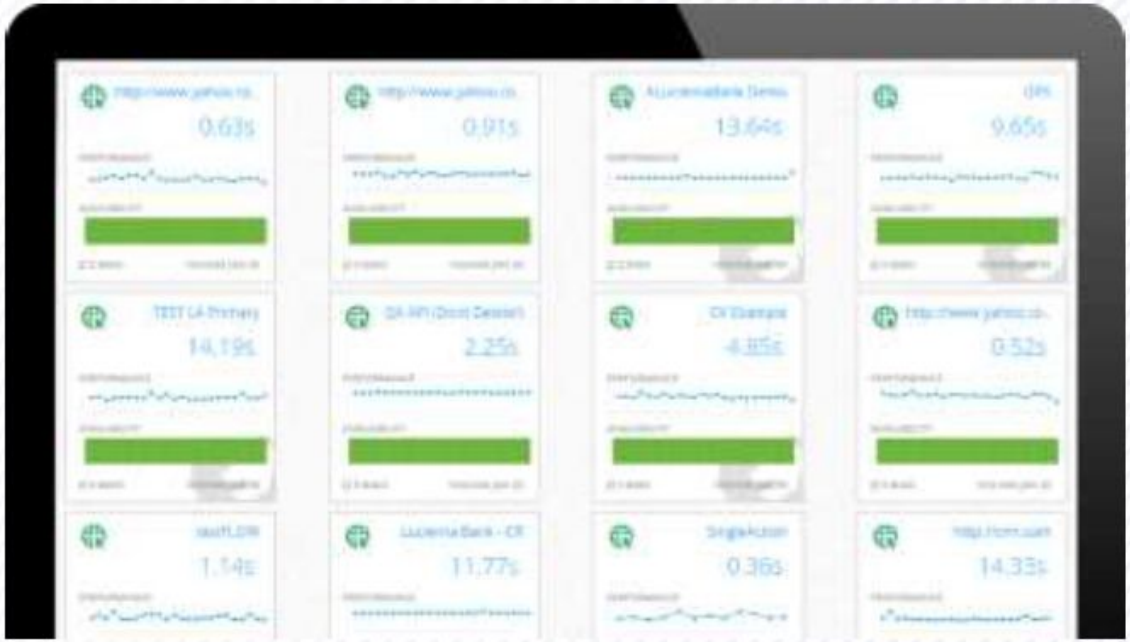


Microservices

Complexity | Automation | Availability | **Performance**

Performance

- Tester les performances sur un seul service versus tests à l'échelle du système
- Modélisation économique d'une utilisation à grande échelle
- Conduire une approche de test basée sur les données



Monitoring


Explore
Overview by Service
Container Count by Image
Request Time by Image
Request Count by Image
CPU % by Image
CPU Container Limit % by Image

Monitoring

Xymon

System Monitoring

Virtualization

Virtualization

Pourquoi ?

- Ne pas tester en Prod
- simuler le comportement d’un système
- multiplier les scénario
- Ne pas interagir et influencer la Prod
- ...

ELK

ELK

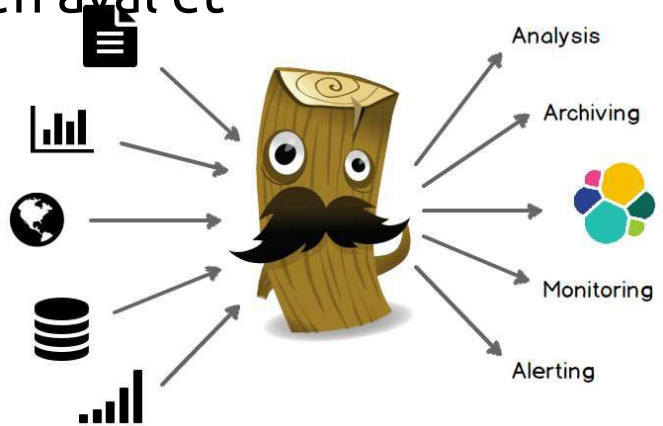
Elasticsearch

Elasticsearch est une recherche en texte intégral open source hautement évolutive et moteur d'analyse. Il vous permet de stocker, rechercher et Analyser rapidement et en temps quasi réel de gros volumes de données.
Il est généralement utilisé comme moteur / technologie sous-jacent qui alimente les applications qui ont des fonctionnalités de recherche complexes et exigences.

ELK

Logstach

Logstash est un moteur de collecte de données open source avec temps réel capacités de pipelining. Logstash peut unifier dynamiquement données provenant de sources disparates et normaliser les données en destinations de votre choix. Nettoyez et démocratiser tous vos données pour diverses analyses avancées en aval et cas d'utilisation de visualisation.

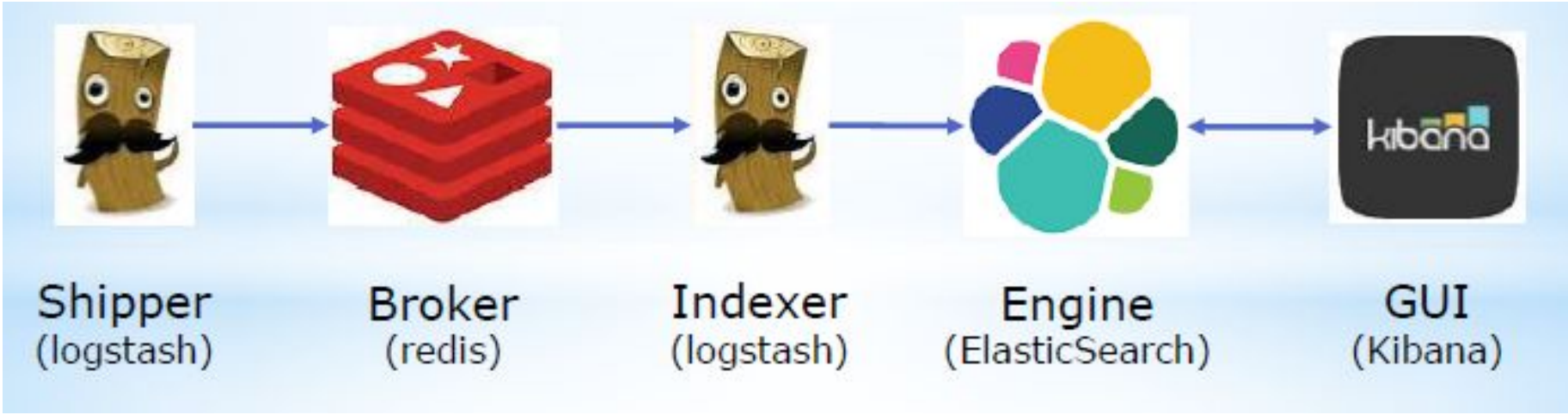


ELK

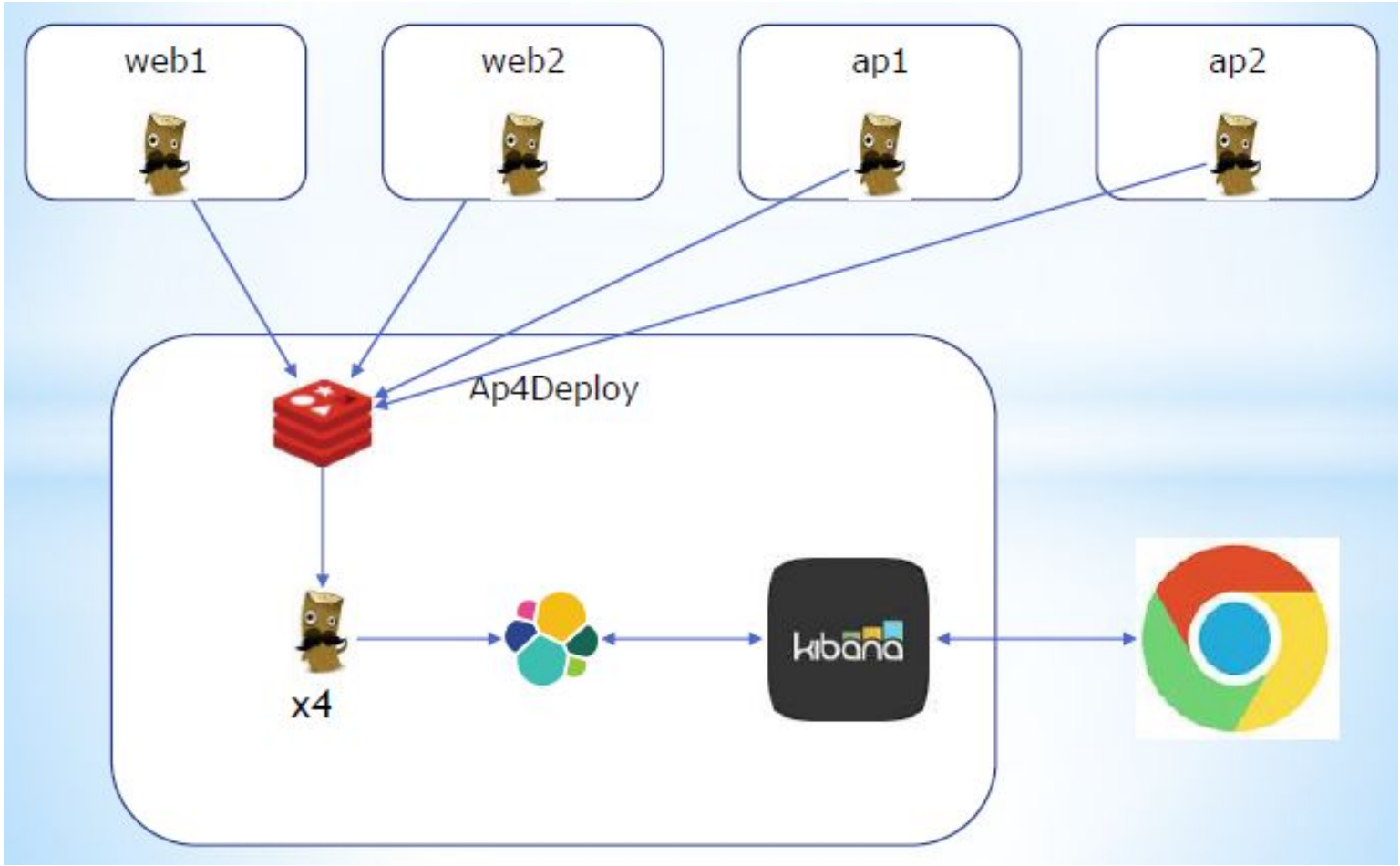
Kibana

Kibana est une plateforme d'analyse et de visualisation open source conçu pour fonctionner avec Elasticsearch. Vous utilisez Kibana pour rechercher, afficher et interagir avec les données stockées dans Elasticsearch indices. Vous pouvez facilement effectuer une analyse avancée des données et visualisez vos données dans une variété de graphiques, de tableaux et de cartes.

ELK


kibana
Shipper
(logstash)
Broker
(redis)
Indexer
(logstash)
Engine
(ElasticSearch)
GUI
(Kibana)

ELK



ELK

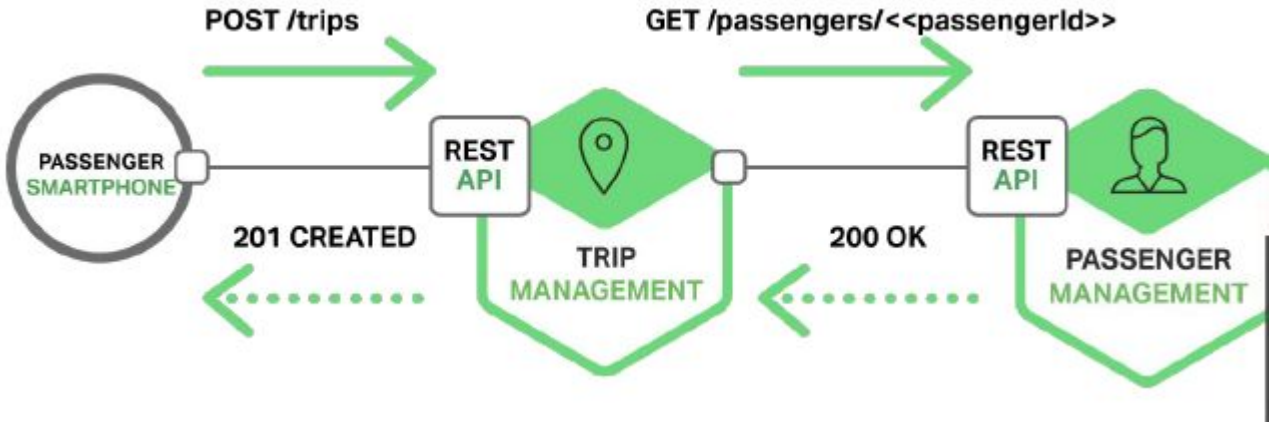

kibana
Discover
Visualize
Dashboard
Settings
1 minute
Yesterday
Dashboard 2
Login User Count
Legend
Count
Count
@timestamp per 30 minutes
库存查询 (PC/Mobile)
Legend
Mobile
PC
Count
@timestamp per 30 minutes
AS400 Queries S4/S7 and Else
Count
@timestamp per 30 minutes
WCPAPI Formats
Legend
1024IN...
PC vs Mobile
Legend
PC
Mobile
Waiting for AS400
Legend
Count
Count
@timestamp per 30 minutes

Enjoy Directly code

SC : Netflix Service Discovery

App Traditionnel

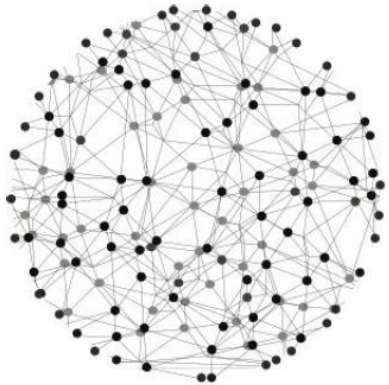
Pour effectuer la communication entre les services, nous devons connaître l'emplacement du service (port, hôte). Dans les applications traditionnelles, c'est une tâche simple car les services fonctionner dans un emplacement fixe et connu.



SC : Netflix Service Discovery

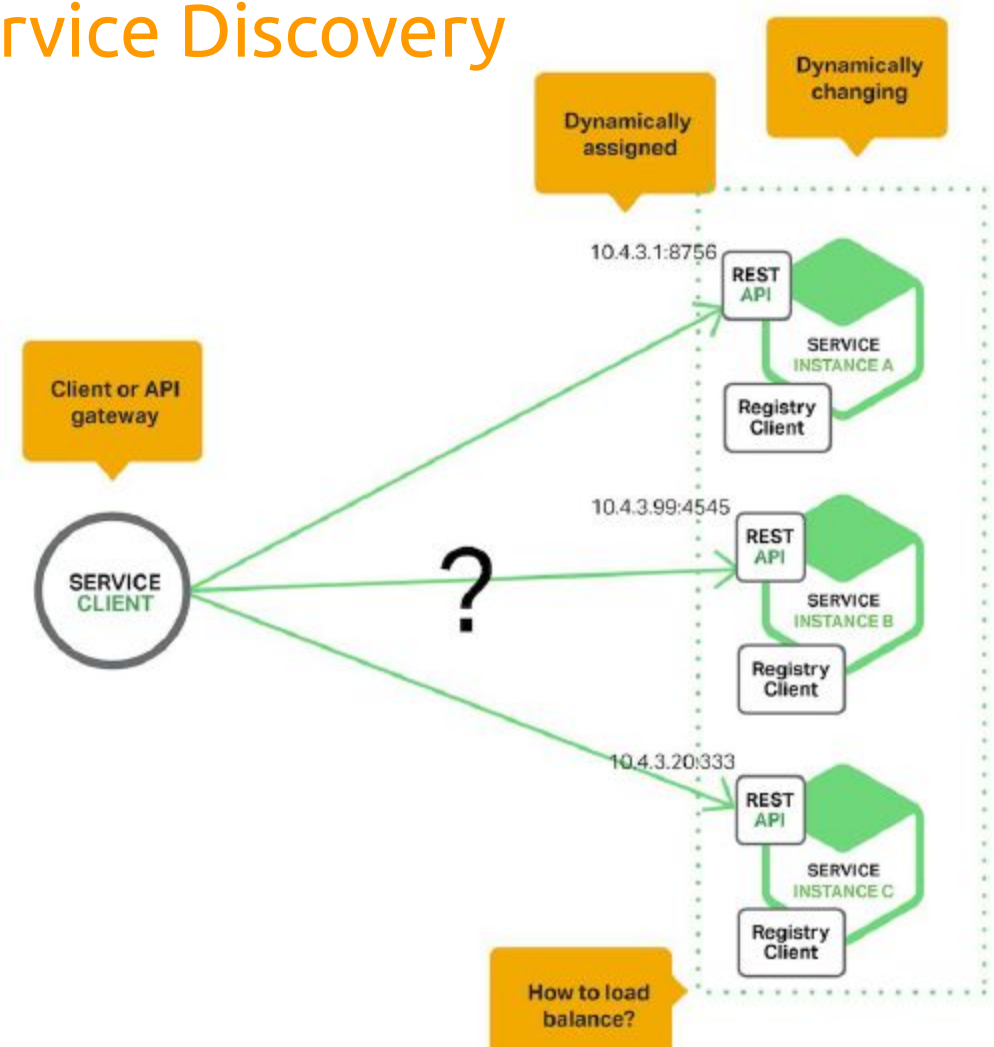
Application moderne

Dans les applications modernes, les services fonctionnent dans un environnement dynamique. Le service peut avoir N instances s'exécutant sur N machines différentes. Dans ce cas, pour connaître l'hôte et le port de chaque service est très pénible.



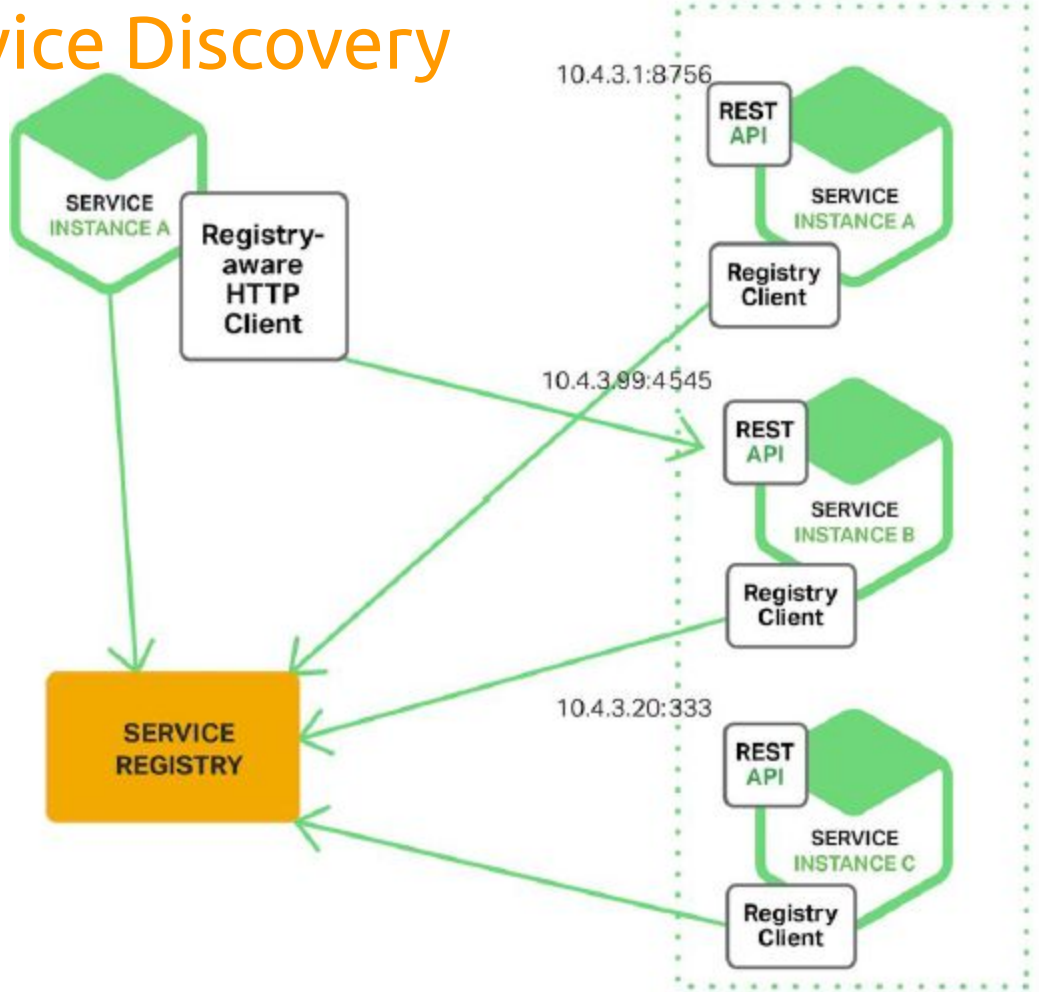
SC : Netflix Service Discovery

Le problème



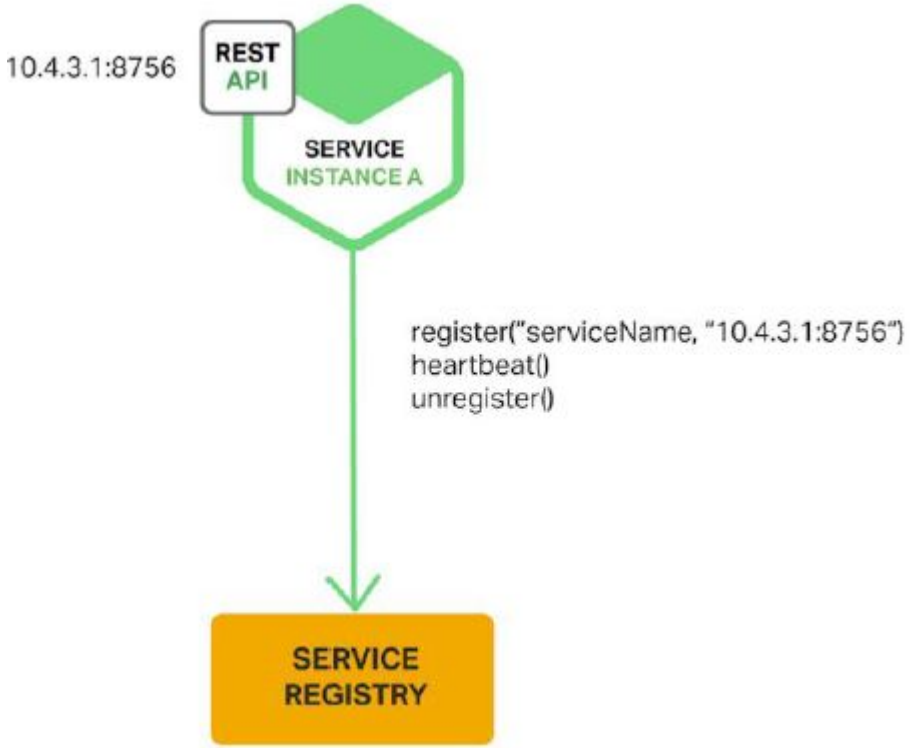
SC : Netflix Service Discovery

Service Discovery



SC : Netflix Service Discovery

Service Registry



SC : Netflix Service Discovery

Eurekaaaaaaaaaa ...

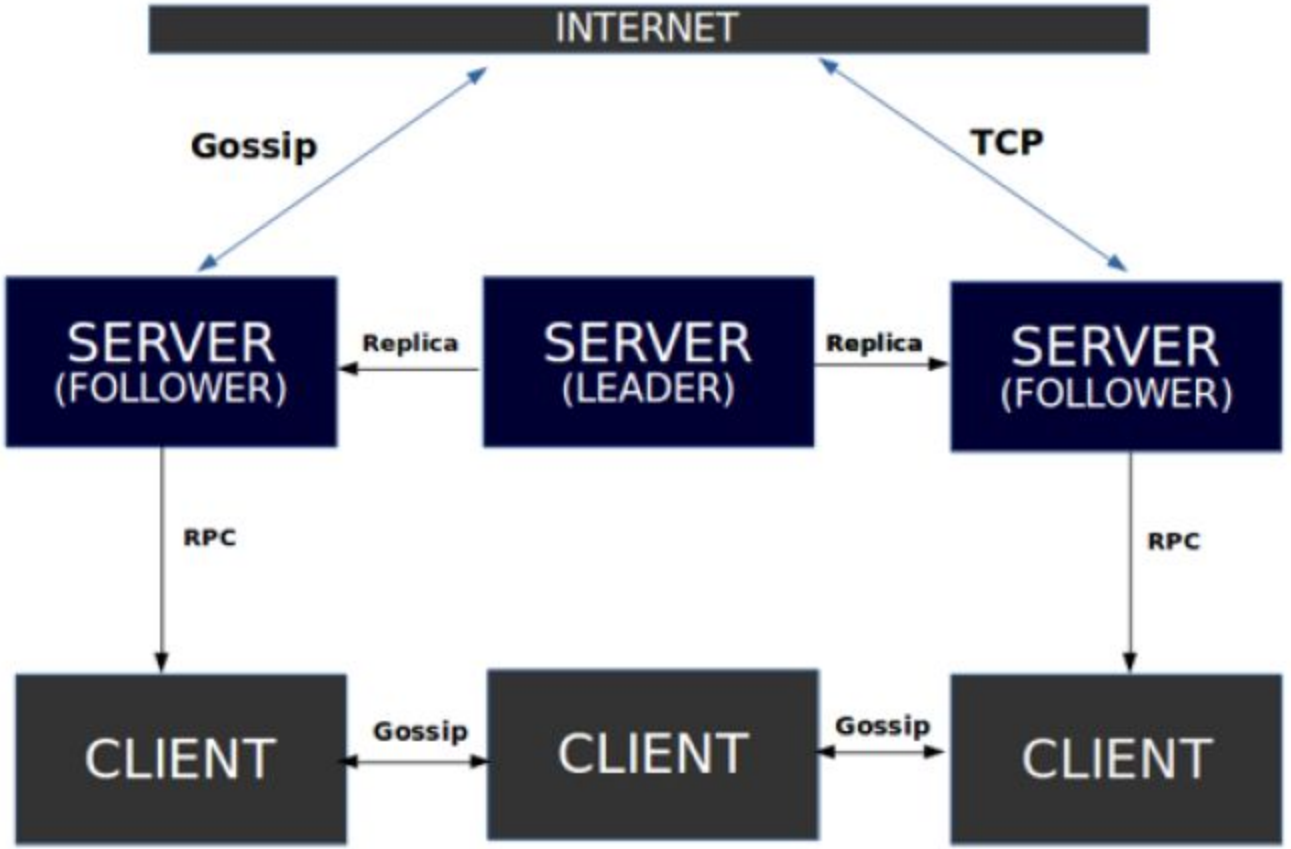
Eureka est un service basé sur REST (Representational State Transfer) qui est principalement utilisé dans le cloud AWS pour localiser les services à des fins d'équilibrage de charge et basculement des serveurs de niveau intermédiaire. »

SC : Consul Service Discovery

- Multicentre de données (datacenter). C'est un outil simplifier la vie des administrateurs système. C'est prévu de base pour fonctionner sur plusieurs datacenters. Les données se répliquent, et en toute logique c'est en cluster. Et ça fonctionne super bien.
- Une base de donnée clé/valeur. Dans Consul tout est stocké dans la base de données. Chaque valeur y est stockée dans une clé. Il y a une hiérarchie avec des dossiers et tout. Et accessible via une API REST.
- Service de découverte. Dans ces valeurs, on peut y trouver des inventaires de services. Dans l'exemple que je prendrais plus tard, je veux mettre X serveurs d'application Python. Pour configurer mon Haproxy, je vais aller chercher la liste des serveurs dans Consul.
- Test de vie. Et justement, pour avoir la liste de mes serveurs d'application Python, je vais installer un agent Consul sur chaque machine et lui ajouter un test de vie. Dés que le serveur est prêt, il en informe consul qui va mettre à jours la configuration haproxy automagiquement. Et pareil quand le serveur sera arrêté.

SC : Consul Service Discovery

Archi



SC : Consul Service Discovery

comparaison

Properties	Consul	Etcd	Zookeeper
User Interface	Available		
RPC	Available	Available	
Health Check	HTTP API	HTTP API	TCP
Key Value	3 Consistency modes	Good Consistency	Strong Consistency
Token System	Available		
Language	Golang	Golang	Java

SC : Consul Service Discovery

Setup

Download consul from https://www.consul.io/downloads.html

Extract binary package from the downloaded folder.

> cd Downloads

> chmod +x consul

> sudo mv consul / usr /

SC : Consul Service Discovery

Consul Template

Daemon that queries the Consul instance and updates any number of specified templates on the file system.

Download & Extract from https://releases.hashicorp.com/consul-template/.

> cd Downloads

> chmod +x consul-template

> sudo mv consul template / usr /share/

SC : Consul Service Discovery

Consul-UI

Divided into three important parts, which are

ACL - Set of Rules to easily lock your clusters easily

Datacenter - Enables you to easily manage datacenters and work out with your cluster

Nodes - Quick update on the nodes that Consul cluster is using

> mkdir /opt/consul-ui

> cd /opt/consul-ui

> wget https://releases hashicorp com/consul/0.7.2/0.7.2 _web_ui zip

> unzip consul_ 0.7.2 _web_ui zip

> sudo consul agent -dev -ui -data-dir tmp/consul

SC : Consul Service Discovery

How to use it ?

//Running Consul

> sudo consul agent -dev -data-dir=tmp/consul

// Listing members

> consul members

// Joining Nodes

> consul join <Node1><Node2>

// Using Docker

> docker run -p 8400:8400 -p 8500:8500 -p 8600:53/ udp -h node1 progrium/consul-

server -bootstrap

SC : Consul Service Discovery

How to use it ?

//Running Consul UI

> docker run -p 8400:8400 -p 8500:8500 -p 8600:53/udp -h node1 progrium /consul-server-bootstrap-ui-dir-ui

// Digging

> dig @127.0.0.1 -p 8600 web.service.consul

// Monitor

> consul monitor

// Watch

> consul watch -type=service service=consul

SC : Consul Service Discovery

How to use it ?

//Registering External Services on Consul

> sudo curl X PUT d '{"Datacenter": "dc1", "Node": "amazon","Address": www.amazon.com

"Service": {"Service": "shop", "Port":80}}' http://127.0.0.1:8500/v1/catalog/register

// Removing External Service

> curl X PUT d '{"Datacenter": "dc1", "Node": "amazon"}' http://127.0.0.1:8500/v1/catalog/deregister

// Info

> sudo consul info

SC : Hystrix Circuit Breaker

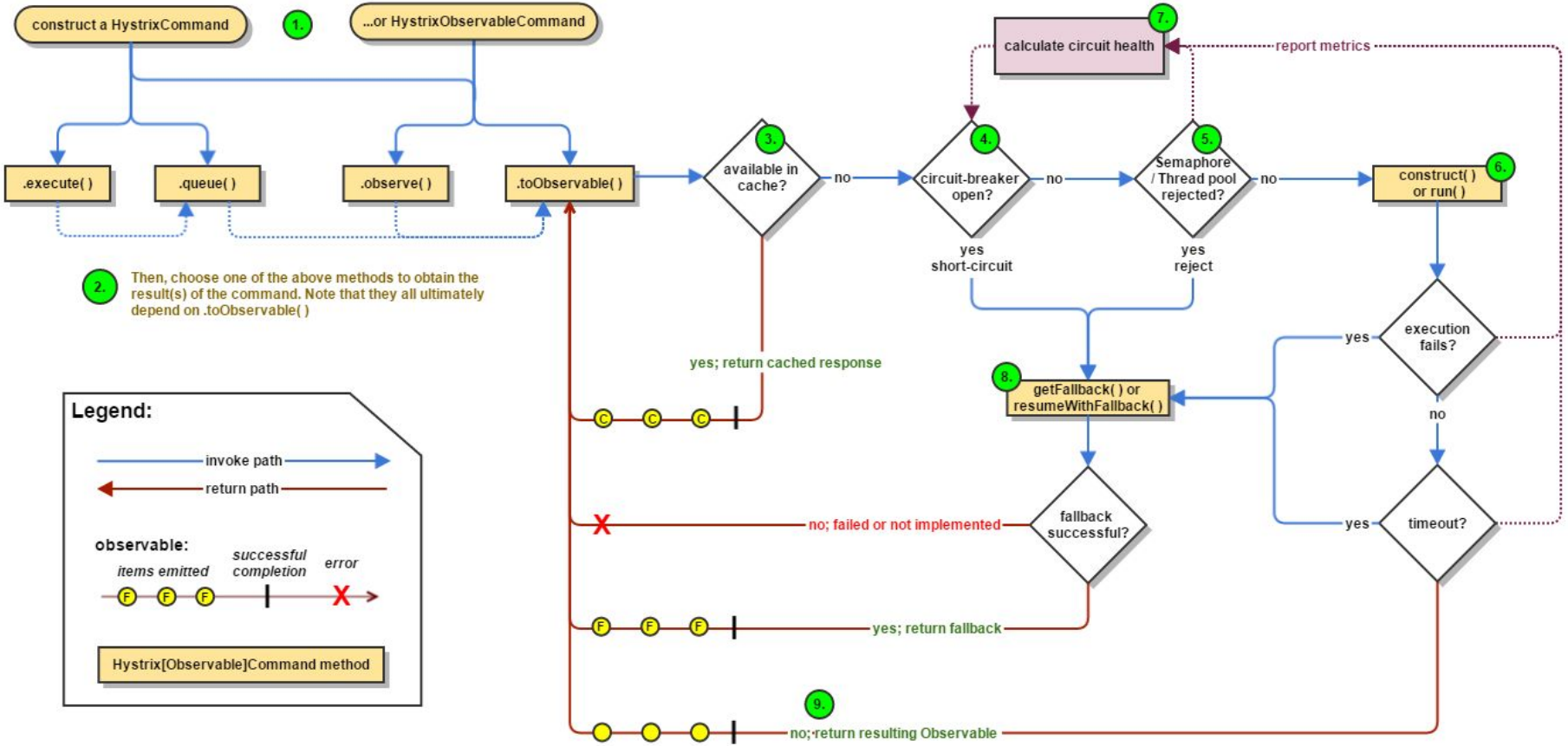
- Hystrix est une bibliothèque de latence et de tolérance aux pannes de Netflix.
- Conçu pour isoler les points d'accès aux systèmes, services et Bibliothèques tierces
- Aide à arrêter les échecs en cascade
- Permettre la résilience dans les systèmes distribués complexes où la défaillance est inévitable.

SC : Hystrix Circuit Breaker

Fonctionnement

- Execute Hystrix command
- Circuit-breaker
- Fallback
- Threads and thread pools
- Request-caching
- Request collapsing

SC : Hystrix Circuit Breaker



SC : Hystrix Circuit Breaker

Execute cmd :

execute() - bloque, puis retourne la seule réponse reçue du

dépendance (ou lève une exception en cas d'erreur)

queue() - renvoie un Future avec lequel vous pouvez obtenir la réponse unique de la dépendance

observe() - souscrit à l'Observable qui représente la (les) réponse (s) du

dépendance et retourne un Observable qui réplique cette source Observable

toObservable() - renvoie un Observable qui, lorsque vous y souscrivez,

exécuter la commande Hystrix et émettre ses réponses

SC : Hystrix Circuit Breaker

Circuit Breaker

Lorsque vous exécutez la commande, Hystrix vérifie avec le circuit breaker

pour voir si le circuit est ouvert.

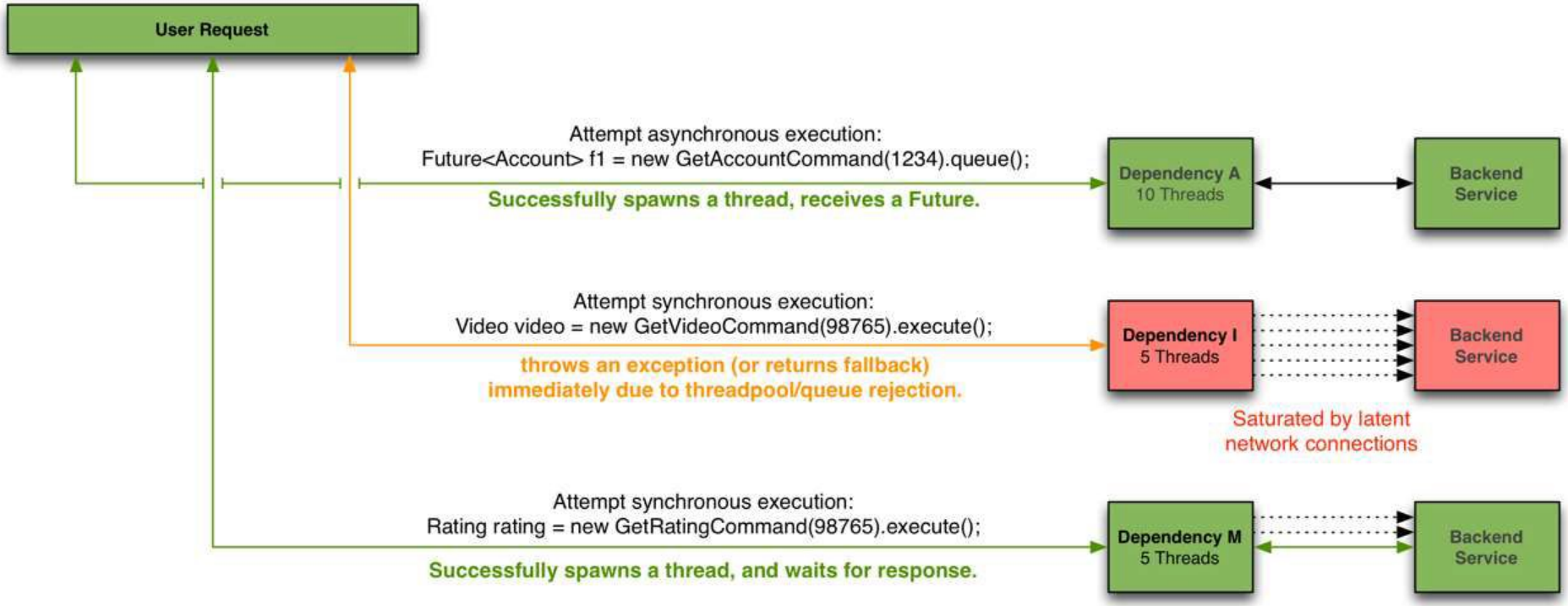
Si le circuit est ouvert -> Hystrix ne s'exécutera pas et ira en repli

Si le circuit est fermé -> le flux passe au pool de threads / sémaphore

étape de rejet pour vérifier s'il y a de la capacité disponible pour exécuter la cmd

SC : Hystrix Circuit Breaker

Thread Pool



SC : Hystrix Circuit Breaker

Fallback

Fallback s'exécute lorsqu'une exécution de commande échoue:

- lorsqu'une exception est levée par construct () ou run ()
- lorsque la commande est court-circuitée car le circuit est ouvert
- lorsque le pool de threads et la file d'attente ou le sémaphore de la commande sont à pleine capacité
- lorsque la commande a dépassé sa durée d'expiration.

HystrixCommand.getFallback () est utilisé pour donner du repli

SC : Zipkin and Sleuth