Examen : Micro-services & Intégration

- Date : 12 novembre 2018
- Durée : 13h30 – 17h30 (4 heures)
- Aucun document autorisé ; barème donné à titre indicatif.

Lisez tout le sujet (3 pages) avant de commencer à répondre aux questions ;
Les questions sont identifiées en gras dans le texte du sujet ;
Les différentes parties (au nombre de 3) sont indépendantes ;
Pour toutes les questions, la justification de vos réponses
est tout aussi importance que votre proposition.

Vous devez répondre aux questions posées dans le sujet sur la copie d'examen fournie, avec d'éventuelles feuilles intercalaires. Vous ne pouvez pas sortir de la salle d'examen durant la première heure (avant 14h30), ni durant le dernier quart d'heure (après 17h15). **Toute fraude identifiée sera systématiquement transmise au conseil de discipline de l'UNS**.

Partie 1	Partie 2	Partie 3			
		Q1	Q2	Q3	Q4
/4	/4	/3	/5	/2	/2

Partie #1 : Monitoring (Supervision) (0h45)

Le chapitre 2 du livre « *Production Ready Microservices* », de Susan J. Fowler (donné dans la bibliographie du cours) est dédié aux différentes dimensions à prendre en compte pour déployer une architecture de micro-services capable de prendre en compte les besoins d'un contexte de production :

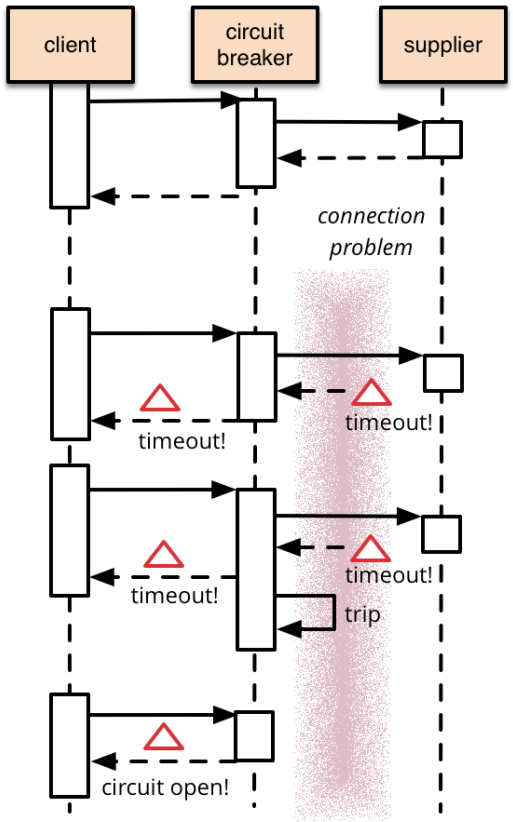
- Stabilité ;
- Fiabilité ;
- Passage à l'échelle ;
- Tolérance aux catastrophes ;
- Performance ;
- Supervision ;
- Et pour finir documentation.

On s'intéresse ici à la dimension de la *supervision* (le chapitre 6 de l'ouvrage est dédié à cette dimension). **Expliquez en une page maximum (300 mots) comment mettre en œuvre de la supervision dans une architecture micro-service, et comment l'exploiter sur l'infrastructure de production.**

Partie #2 : Circuit Breaker (Disjoncteur) (0h45)

Le patron Circuit Breaker est un classique de mise en œuvre d'architecture micro-services. Martin Fowler le décrit de la manière suivante sur son site https://martinfowler.com/bliki/CircuitBreaker.html.

It's common for software systems to make remote calls to software running in different processes, probably on different machines across a network. One of the big differences between in-memory calls and remote calls is that remote calls can fail, or hang without a response until some timeout limit is reached. What's worse if you have many callers on a unresponsive supplier, then you can run out of critical resources leading to cascading failures across multiple systems. In his excellent book « Release It », Michael Nygard popularized the Circuit Breaker pattern to prevent this kind of catastrophic cascade.



The basic idea behind the circuit breaker is very simple. You wrap a protected function call in a circuit breaker object, which monitors for failures. Once the failures reach a certain threshold, the circuit breaker trips, and all further calls to the circuit breaker return with an error, without the protected call being made at all. Usually you'll also want some kind of monitor alert if the circuit breaker trips.

- **Définissez en pseudo-code (ou dans le langage de votre choix) une implémentation de ce patron de conception.**
- **Proposez une analyse critique et synthétique de ce patron (environ 1 page, 300 mots**. Pistes que vous pouvez suivre dans votre analyse (non exhaustives) : Quel est son intérêt, quelles sont les différentes possibilités pour le mettre en œuvre dans une architecture logicielle, à quels types de scénarios peut-il répondre, …

Partie #3 : Trot'hipster (2h30)

Les trottinettes électriques en libre-service sont ***LE*** marché de l'année 2018. Surfant sur cette vague *hipster*, la société *Trot'hipster* souhaite déployer une architecture micro-service en support à son infrastructure logistique.

Du point de vue de l'utilisateur, une application mobile permet de repérer la géolocalisation des trottinettes à disposition sur les parking « deux roues » de la ville, ainsi que leur niveau de batterie. Comme les localisations GPS sont approximatives, il est possible de sélectionner une des trottinettes à proximité pour en déterminer l'emplacement précis (selon l'heure et la luminosité ambiante, la trottinette va alors klaxonner et/ou clignoter). Une fois la trottinette choisie, l'utilisateur scanne le QR-code scotché sur le châssis depuis son téléphone, et la trottinette se déverrouille. Il peut alors l'utiliser pour se déplacer. À tout moment il peut consulter son téléphone pour savoir combien lui coûte son déplacement. Quand il a terminé, il utilise l'appli mobile pour « libérer » la trottinette, ce qui enregistre sa localisation, verrouille l'engin, et déclenche la facturation sur la carte bleue enregistrée sur le profil du client.

Du point de vue des *juicers* (les employés de *Trot'hipster* chargés de recharger les batteries des trottinettes), ils peuvent voir à tout moment les trottinettes libres qui ont moins de 20% de batterie à proximité, et peuvent les préempter pour en bloquer la réservation. Passé 19h, ils peuvent récupérer n'importe quelle trottinette disponible dans la ville. Le *juicer* signale qu'il a pris la trottinette, la recharge (il est payé au nombre de trottinettes rechargée), et reçoit un emplacement dans son rayon d'action où il devra redéposer l'engin avant 7h le lendemain matin.

Du point de vue des administrateurs, il faut à tout moment connaitre la position des trottinettes dans la ville, pour déterminer les zones blanches et potentiellement déclencher des déplacements. Ces déplacements peuvent être fait via les *juicers*, mais aussi par la mise en place de « missions » pour les utilisateurs si ceux-ci déposent leur trottinette en fin de déplacement dans une zone blanche. Les utilisateurs reçoivent des alertes directement sur leur téléphone quand de telles opportunité sont disponible, et gagnent des coupons de réduction à utiliser leur leurs prochains trajets.

1. **Définissez en pseudo-code les interfaces des services** (objets métiers inclus) dont vous avez besoin pour répondre à cette étude de cas. **Justifiez**-le (ou les) style de conception de services choisis
2. Sur la base de cette description, **définissez les évènements d'intérêt** dans votre système. En prenant comme base les cas d'utilisations principaux de la description précédente, **décrivez des scénarios de bout en bout** permettant de réaliser les besoins décrit avec vos services : quel service est appelé initialement, comment les évènements sont consommés / produits par les services, …
3. **Que pouvez-vous mettre en œuvre** dans votre architecture pour être tolérant aux fautes ?
4. En aout 2013, Github est arrivé à un pic de déploiement de 175 mises en production dans la même journée (une toute les 8 minutes environ). Quelles solutions proposeriez-vous à Trot'Hipster pour atteindre de telles capacités, en exploitant au mieux son architecture micro-services.