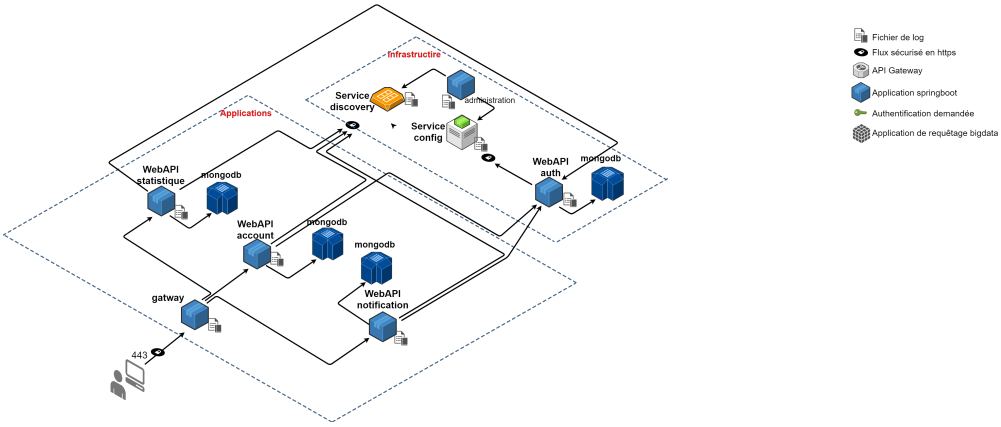
TP7 - Architecture résiliente

Infrastructure de production

ARCHITECTURE RÉSILIENTE



L'objectif de ce TP est d'installer et d'utiliser tous composants vus pour réaliser une architecture scalable et résiliente a la panne. Pour cela on va installer tous les composants nécessaires.

Ce TP doit être réalisé uniquement sous Docker et avec plusieurs machines car il y a beaucoup de composant à lancer.

Les éléments de notation

Envoyer par email les éléments suivant :

- Un document texte avec toutes les actions effectué et des captures d'écran du monitoring, et des logs.

Les sources

Vous trouverez les sources de l'application a installer : https://github.com/diogene/PiggyMetrics. Regarder le contenu des fichiers et des répertoires pour vous aider. Lors de la mise en place des éléments, tous les composants ne sont pas modifié pour prendre en compte les éléments attendus.

Vous aurez aussi besoin de http://nilhcem.com/FakeSMTP/download.html pour les mails.

Préparation

il faut mettre les variables d'environnement.

sous linux

```
export CONFIG_SERVICE_PASSWORD=123456\@!
export NOTIFICATION_SERVICE_PASSWORD=123456\@!
export STATISTICS_SERVICE_PASSWORD=123456\@!
export ACCOUNT_SERVICE_PASSWORD=123456\@!
export MONGODB_PASSWORD=123456\@!
```

Les mots de passe ne sont pas a changer

Les composants

docker

creation d'un réseau :

```
docker network create -d bridge piggyMetrics
```

Root

les deux composants de base sont le serveur de configuration et le serveur registry.

bases de données

ensuite il y a les quatres instances de mongodb

- account-mongodb
- auth-mongodb
- statistics-mongodb
- notification-mongodb
- RabbitMq

Lancer la récupération des images docker, qui ne changeront pas, avec :

```
docker-compose -f docker-compose.db.yml up --no-start
```

Par défaut le docker compose utilise les images préexistant sur le docker hub. lorsque vous faite des modifications des sources il faut changer le docker compose pour utiliser les éléments locals

exemple avec la partie admin

```
  admin:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
    image: java
    restart: always
    depends_on:
      - registry
    ports:
      - 9510:8080
    volumes:
      - /home/etud/diogene.moulron/TP/08/PiggyMetrics/admin/target/admin-1.0-SNAPSHOT.jar:/usr/share/admin
/admin-1.0-SNAPSHOT.jar
    command: bash -c 'java -jar /usr/share/admin/admin-1.0-SNAPSHOT.jar --server.port=8080 -Dlogging.level.org.
springframework=DEBUG '
```

Les containers peuvent demarrer que après le demarage complet de config

Pour ce faire, la ligne de commande doit évoluer et intégrer un heathcheck :

```
command: >
      /bin/sh -c "
        echo Waiting for config service start...;
        while ! curl --user "user:$CONFIG_SERVICE_PASSWORD" --write-out %{http_code} --silent --output /dev
/null -f http://config:8888/actuator/health;
        do
          sleep 1;
        done;
        echo Connected!
        bash -c 'java -jar /usr/share/auth-service/auth-service-1.0-SNAPSHOT.jar -Dlogging.level.org.
springframework=DEBUG';
      "
```

Les services d'infrastructure

les service d'infrastructure correspondent aux services permettant les échanges et l'authentification :

- Auth Service

Les services métiers

Coeur de l'application :

- account-service
- statistics-service
- notification-service
- Gateway

La partie notification a une spécificité : il faut un serveur smtp pour fonctionner.

```
command: >
      /bin/sh -c "
        echo Waiting for config service start...;
        while ! curl --user "user:$CONFIG_SERVICE_PASSWORD" --write-out %{http_code} --silent --output /dev
/null -f http://config:8888/actuator/health;
        do
          sleep 1;
        done;
        echo Connected!
        bash -c 'java -jar  /usr/share/notification-service/fakeSMTP-2.0.jar -s -b -p 2525 -a 127.0.0.1  > /dev
/null' &
        bash -c 'java -jar /usr/share/notification-service/notification-service-1.0.0-SNAPSHOT.jar -Dlogging.
level.org.springframework=DEBUG'
      "
```

Ce lancement intègre **fakeSMTP-2.0.jar**.

Les services de support

Les services de support sont les services permettant la gestion opérationnel de la plateforme

- spring-administrator
- cadvisor pour gerer les composants dans docker
- prometheus
  prendre le fichier PiggyMetrics/registry/prometheus-config.yml comme exemple
- grafana
- Une stack elastic pour visualiser les logs
  Stocker les log dans le système de fichier hôte pour que logstash puisse les lire

la partie docker compose de prometheus :

```
  prometheus:
    image: quay.io/prometheus/prometheus
    ports:
      - 9090:9090
    networks:
      - default
    volumes:
      - /home/etud/diogene.moulron/TP/08/PiggyMetrics/config/src/main/resources/shared/prometheus-config.yml:
/etc/prometheus/prometheus.yml
```

La configuration doit être modifiée pour que tous les composants soient pris :

Dans le fichier config/src/main/resources/shared/application.yml :

```
eureka:
  instance:
    prefer-ip-address: false
    leaseRenewalIntervalInSeconds: 10
    metadata-map:
      management.context-path: ${management.context-path}
```

puis dans chaque configuration (exemple config/src/main/resources/shared/auth-service.yml)

```
management:
  context-path: /uaa/actuator
```

et il faut changer la configuration de prometheus :

```
  # Added to get services from Eureka-Consul-Adapter
  - job_name: 'eureka'
    scrape_interval: 20s
    metrics_path: /actuator/prometheus
    static_configs:
    consul_sd_configs:
      - server: 'registry:8761'
    relabel_configs:
      - source_labels: ['__meta_consul_tags']
        action: keep
      - source_labels: ['__meta_consul_service']
        target_label: job
      - source_labels: [__meta_consul_metadata_management_context_path]
        regex: (.*)
        replacement: '${1}/prometheus'
        target_label: __metrics_path__
```

ce qui donne dans prometheus/target :

eureka (6/6 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://172.19.0.4:34329/accounts/actuator/prometheus	UP	instance="172.19.0.4:34329" job="ACCOUNT-SERVICE"	12.165s ago	21.55ms	
http://172.19.0.12:8080/actuator/prometheus	UP	instance="172.19.0.12:8080" job="ADMIN-SERVICE"	9.395s ago	18.65ms	
http://172.19.0.3:5000/uaa/actuator/prometheus	UP	instance="172.19.0.3:5000" job="AUTH-SERVICE"	2.983s ago	17.55ms	
http://172.19.0.9:4000/actuator/prometheus	UP	instance="172.19.0.9:4000" job="GATEWAY"	1.713s ago	16.44ms	
http://172.19.0.7:45249/notifications/actuator/prometheus	UP	instance="172.19.0.7:45249" job="NOTIFICATION-SERVICE"	7.63s ago	60.05ms	
http://172.19.0.11:37209/statistics/actuator/prometheus	UP	instance="172.19.0.11:37209" job="STATISTICS-SERVICE"	7.745s ago	44.26ms	

Préparation des tests pour visualiser les changements dans l'infra

1. Créé un script jmeter de test de votre application avec le scénario : Identification, visualisation des comptes, changement de type d'une dépense.
2. Lancer le test.
3. Visualiser les utilisations des services dans grafana
4. Ou sont stockés les fichiers de configuration de chaque composant ?
5. Donnez tous les ports de démarrage
6. Donnez une capture d'écran du wallboard de spring boot administrator, augmentez le nombre de noeud pour le composant de statistique et donnez une nouvelle capture d'écran

Visualiser la scallabilité

1. Lancer le test fait précédemment
2. Visualiser les utilisations des services dans le turbine et dans grafana
3. Comment se comporte votre application ? a mettre dans le document a rendre.
4. Lancer une nouvelle instance de account-service.
5. Cette instance doit être visible dans le serveur registry.
6. Quelle est le changement constaté ? a mettre dans le document a rendre.

Visualiser la résilience

1. Lancer le test fait précédemment
2. Visualiser les utilisations des services dans grafana
3. couper brutalement une instance
4. Quelle est le changement constaté ? a mettre dans le document a rendre

Intégration des bases de données

Intégré les bases de données dans le registry

Sécurisation

Pour ceux qui ont terminer avant, reprenez les travaux et sécurisé tous les services avec du SSL.

Architecture résiliente

Les éléments d'infrastructure

Il est possible de faire ce TP entièrement sous docker en utilisant le mode swarm qui permet de mettre en cluster plusieurs machines.

```
docker swarm init --advertise-addr <ip d'une machine>
```

La sortie de cette commande donne :

```
Sortie :
Swarm initialized: current node (pf981eg2bpeapmb2wmgz474zj) is now a manager.
To add a worker to this swarm, run the following command:
    docker swarm join --token SWMTKN-1-1j3j2jniehxt2s615l1o919v3vra5c68dtrp9ftiidmj1igdfk-
15dc8dkqiwkdslo534mxz2p2g 192.168.46.70:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

sur deux autres machines :

```
docker swarm join
```

le résultat doit être, à la fin :

```
docker node ls

ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
xun4avhi8wzkt6bca5zo7fjkl     pcl-e01-18          Ready               Active
iw006c0kvc8hcivvki38vxx1f     pcl-e01-19          Ready               Active
pf981eg2bpeapmb2wmgz474zj *   pcl-e01-20          Ready               Active              Leader
```

Ensuite, le tableau d'équivalence :

dans docker standard	dans docker swarm
docker run	docker service create
docker-compose -f /path/to/docker-compose.yml up	docker stack deploy --compose-file /path/to/docker-compose.yml mystack