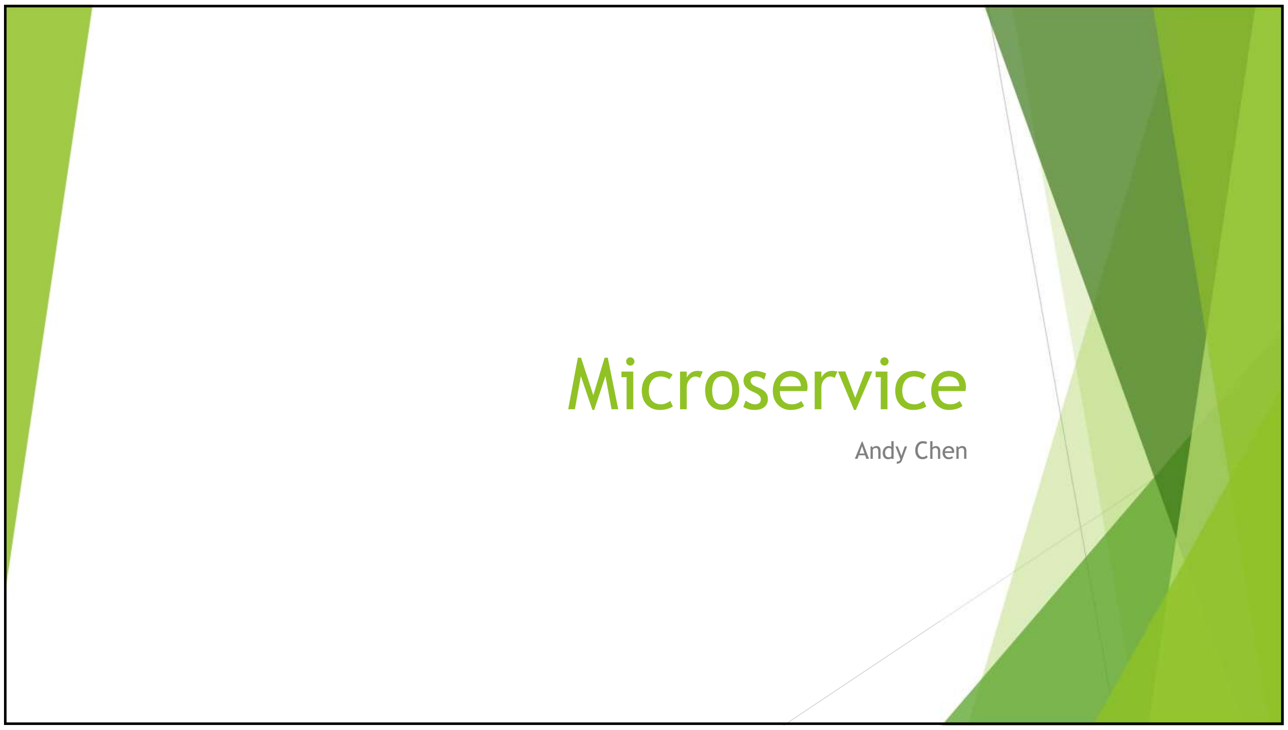
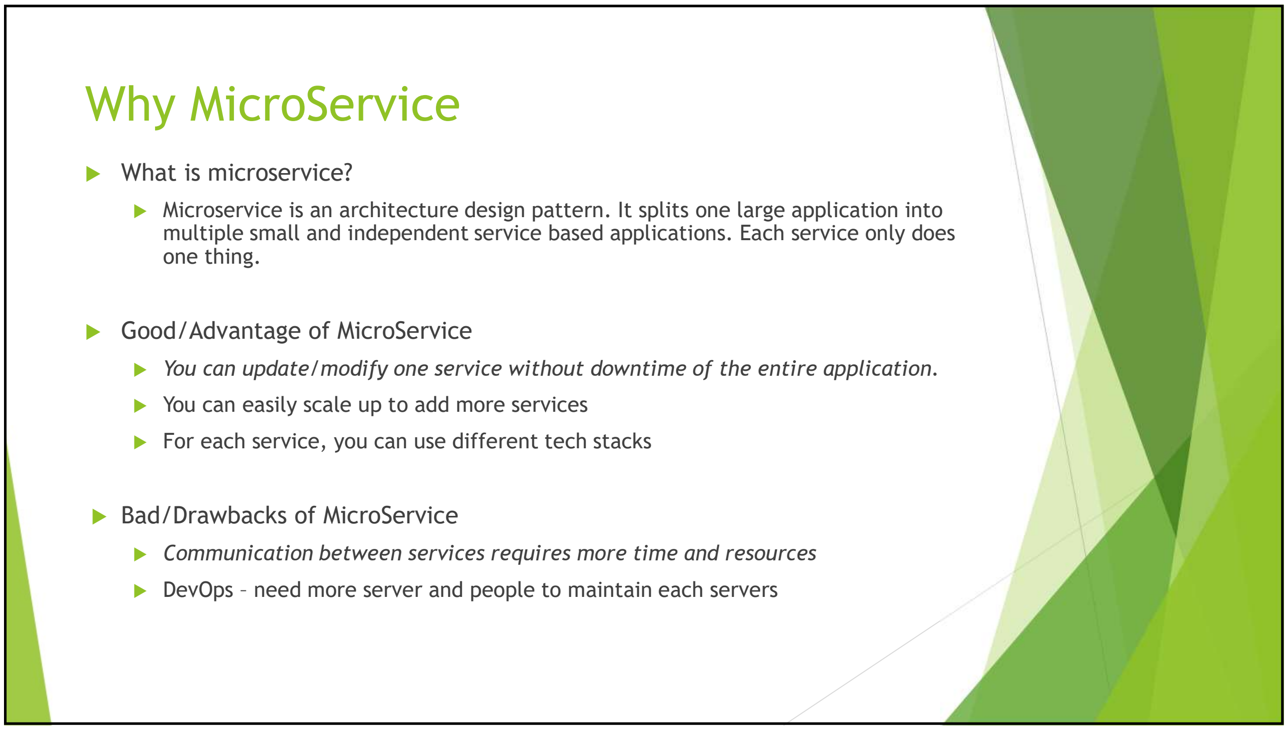


Microservice

Andy Chen

1



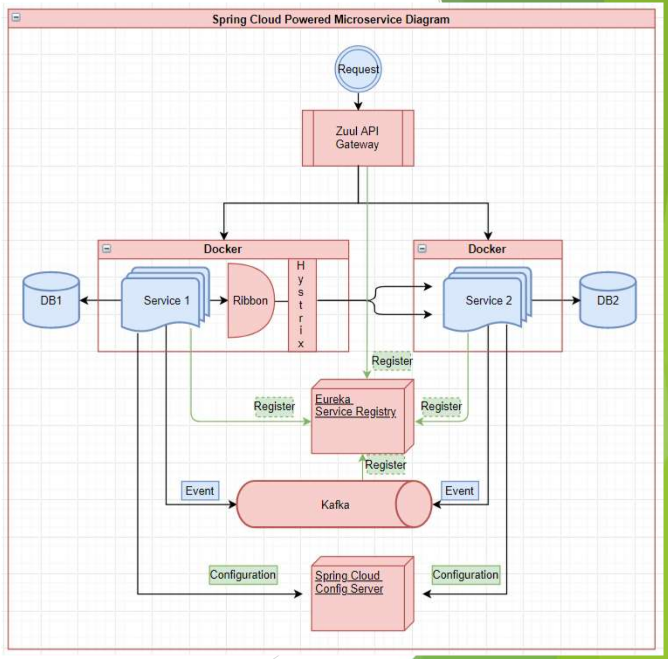
Why MicroService

- What is microservice?
  - Microservice is an architecture design pattern. It splits one large application into multiple small and independent service based applications. Each service only does one thing.
- Good/Advantage of MicroService
  - *You can update/modify one service without downtime of the entire application.*
  - You can easily scale up to add more services
  - For each service, you can use different tech stacks
- Bad/Drawbacks of MicroService
  - *Communication between services requires more time and resources*
  - DevOps – need more server and people to maintain each servers

2

Component

- Spring Cloud
- Zuul API Gateway
- Eureka
- Ribbon Load Balancer
- Hystrix Circuit Breaker
- Config Server
- Kafka
- Docker

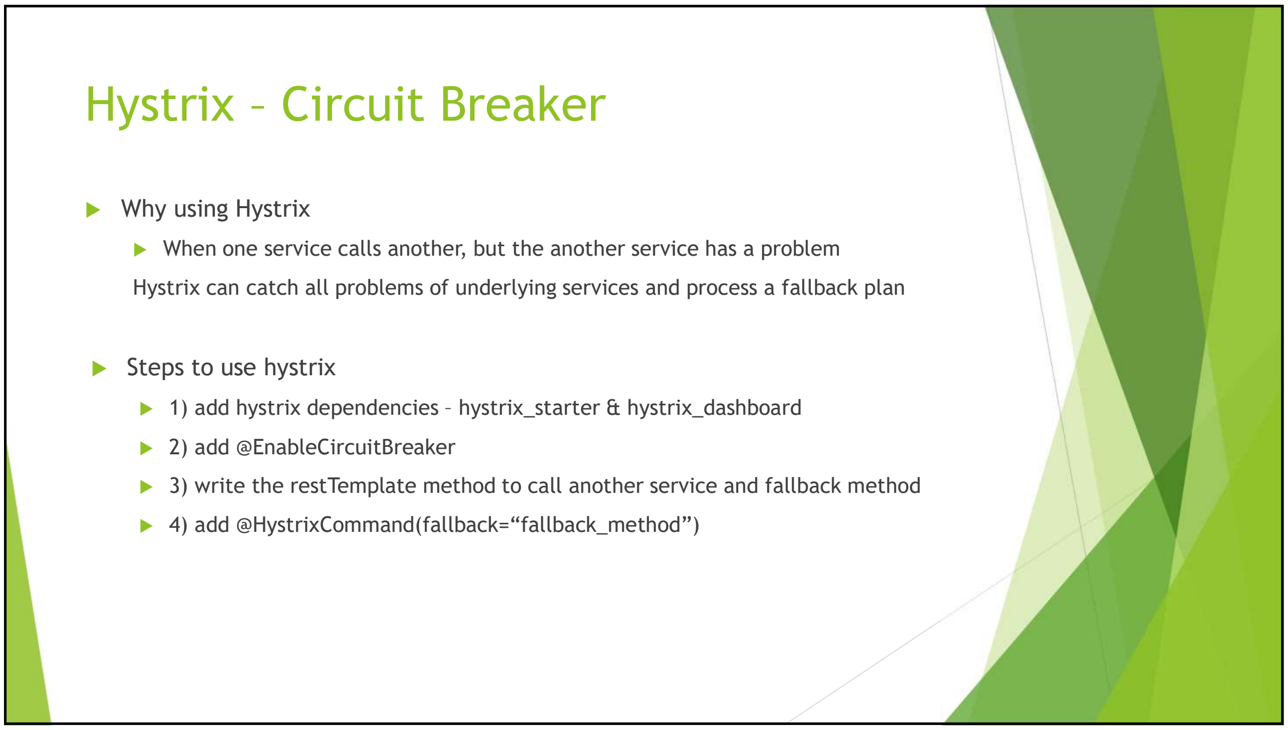


3

Eureka

- Why using Eureka and how to register service in Eureka?
- Why?
  - Eureka register service by serviceid, monitor service health status, auto-register new services
- How to register service?
  - 1) @EnableEurekaClient in client service
  - 2) Give service a service id – spring.application.name = serviceid
  - 3) In client service application.properties, set eureka server default url

4



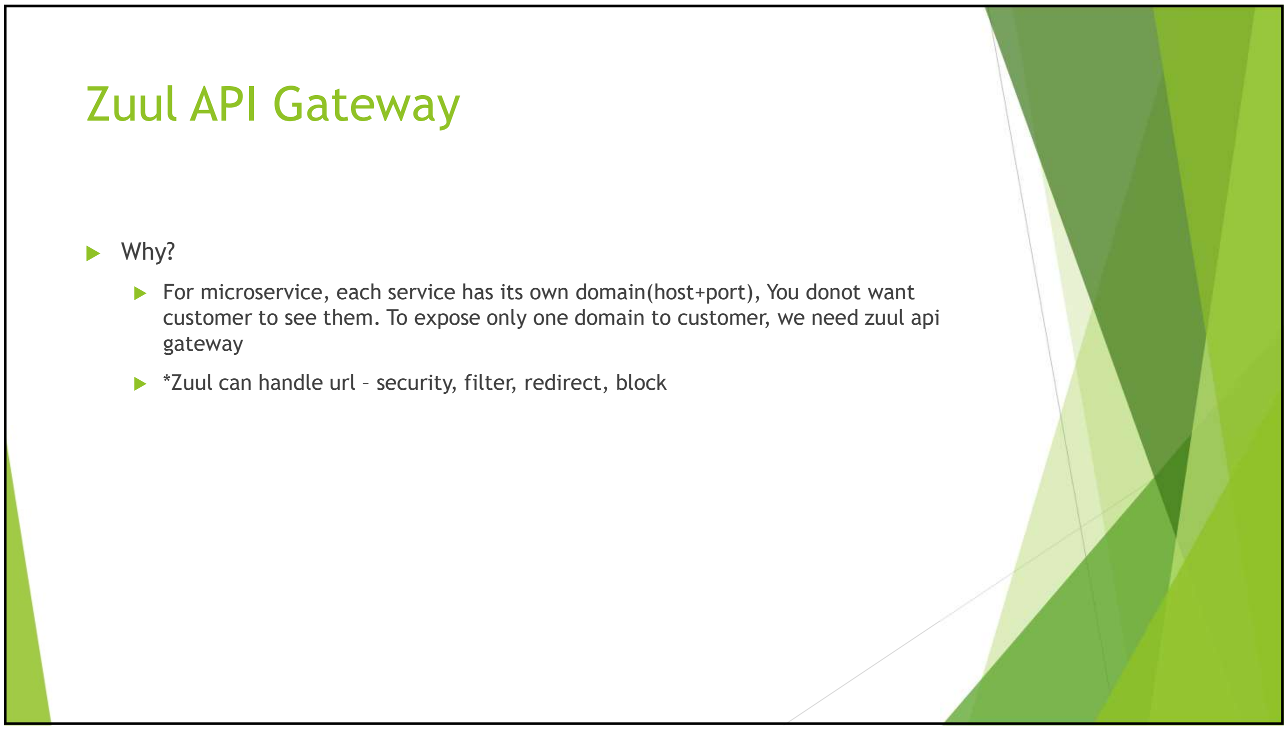
Hystrix – Circuit Breaker

- Why using Hystrix
  - When one service calls another, but the another service has a problem

    Hystrix can catch all problems of underlying services and process a fallback plan

- Steps to use hystrix
  - 1) add hystrix dependencies – hystrix_starter & hystrix_dashboard
  - 2) add @EnableCircuitBreaker
  - 3) write the restTemplate method to call another service and fallback method
  - 4) add @HystrixCommand(fallback="fallback_method")

5



Zuul API Gateway

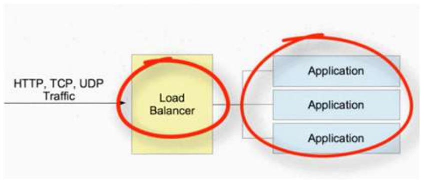
- Why?
  - For microservice, each service has its own domain(host+port), You donot want customer to see them. To expose only one domain to customer, we need zuul api gateway
  - *Zuul can handle url – security, filter, redirect, block

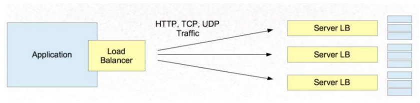
6

Load Balancer

- Server Side Load Balancer
- Client Side Load Balancer

- Why client side is better?



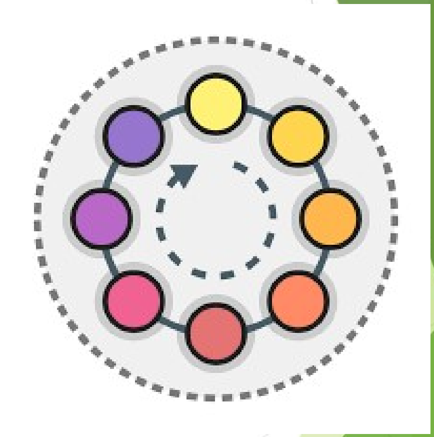


7

Load Balancer Algorithms

Some most popular Ribbon load balancer rules

- Round Robin
- Availability Filtering
- Weighted Response Time
- Random



8

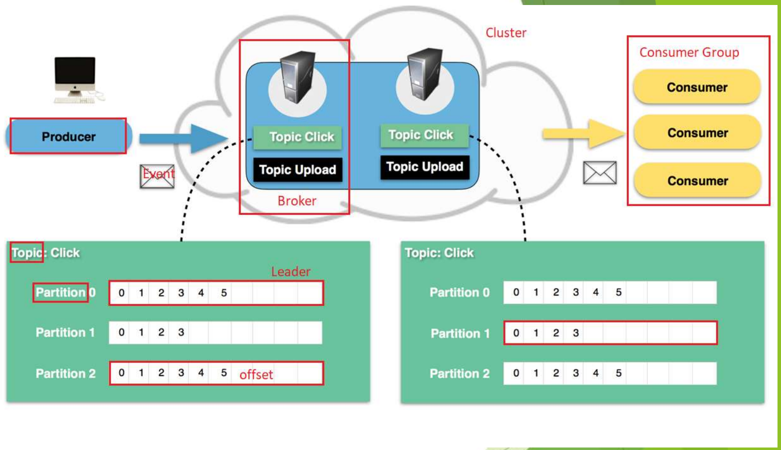
Config Server

9

KAFKA – important notes

- Explain kafka
  - Message broker/agent, producer and consumer, producer sent message to kafka server and kafka server dispatch message to different queue based on routing key, consumer will get message from queue
- Benefit using kafka
  - 1. very fast
  - 2. large amount of data
  - 3. Asynchronous and Concurrent
  - 4. Replay
- Kafka vs RabbitMQ vs JMS

10



KAFKA Diagram

Producer
Consumer/Consumer Group
Cluster
Broker
Topic
Partition(leader/replica/offset)
Event/Message

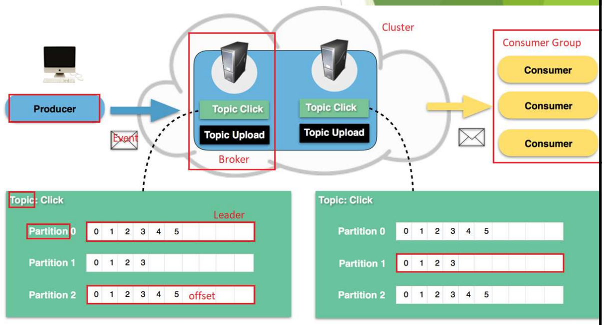
11

KAFKA: Topic and Fault Tolerance

- No Queue, only Topics
- Producer sends message to topic, consumer group listens to all partitions inside each topic.
- Consumer sends heartbeat to kafka cluster for updating its status
- Consumer subscribes to different partitions

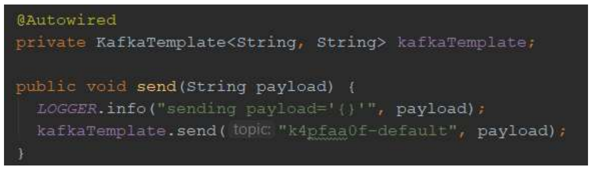
- **FAULT TOLERANCE**: Partition has replica with replica factor, only leader is consumed by consumer group, if leader is down, replica will become leader

- **MAINTAIN ORDER**: events have same key so that they will send to same partition and consumed in order



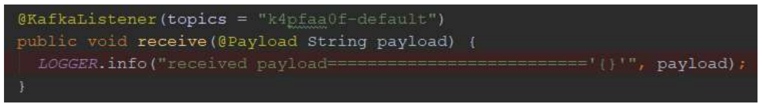
12

KAFKA: Producer and Consumer



```
@Autowired
private KafkaTemplate<String, String> kafkaTemplate;

public void send(String payload) {
    LOGGER.info("sending payload='{}'", payload);
    kafkaTemplate.send( topic: "k4pfaa0f-default", payload);
}
```



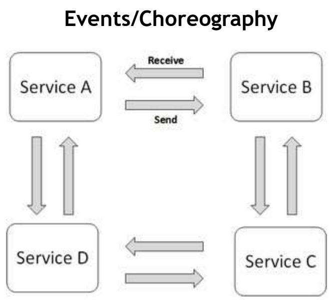
```
@KafkaListener(topics = "k4pfaa0f-default")
public void receive(@Payload String payload) {
    LOGGER.info("received payload=========================='{}'", payload);
}
```

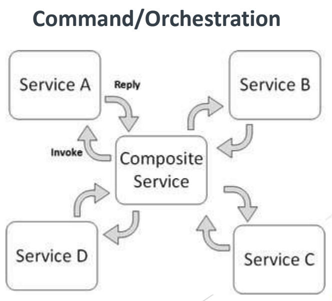
Kafka version: kafka2 was released in middle 2018

13

Transaction: Saga Pattern

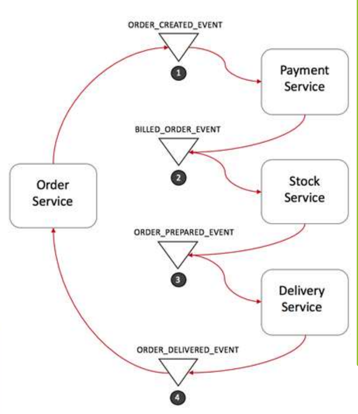
- History: Traditional global transaction Pattern: 2-phase commit and why it may not be good for microservice
- What is Saga Pattern – a sequence of independent local transactions
- How to implement Saga Pattern



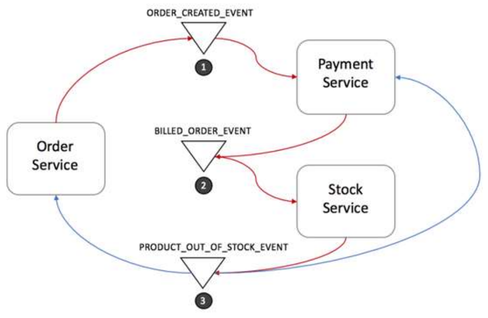


14

Saga: Events/Choreography



Normal Forward flow



Rollback flow

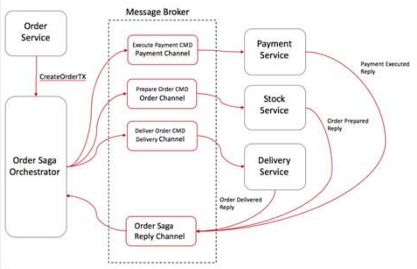
Pro:
- Easy to understand
- Easy to build

Con:
- Hard to scale – hard to track who is listening to whom
- Cyclic dependency

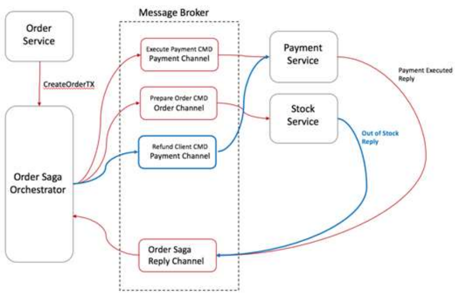
Pros & Cons

15

Saga: Command/Orchestration



Normal Forward flow



Rollback flow

Pro:
- No cyclic dependency
- Centralized orchestrator – easy to add new service
- Service are independent

Con:
- Relying too much on Orchestrator
- Increase DevOps for new service

Pros & Cons

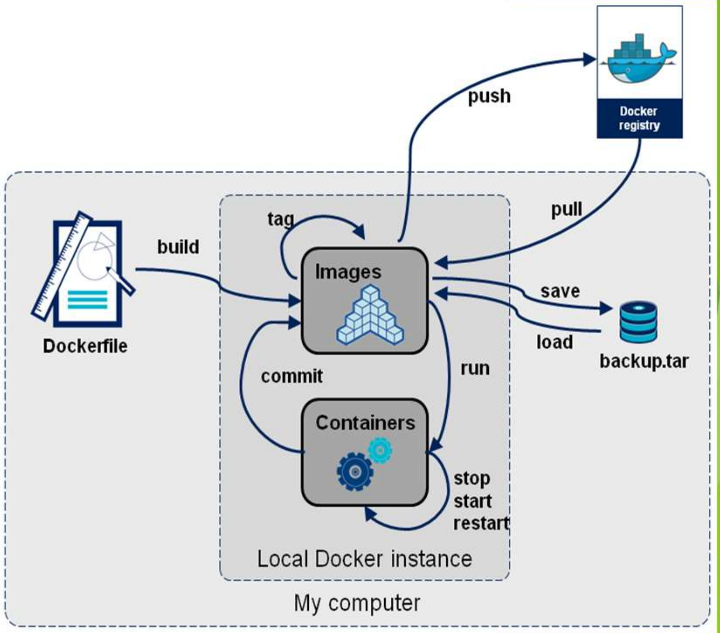
16

Docker

Docker vs Virtual Machine

Docker pull
Docker commit / push

Docker run



17

End of MicroService

18