schaerer
swiss coffee competence

zühlke
empowering ideas

COFFEE TO GO WITH A «CLOUDLY» FOAM

CLOUD BASED FUNCTIONALITIES OF PREMIUM COFFEE MACHINES WITH KUBERNETES AND KAFKA

Who are we?

Patrick Wachsmuth

Senior Software Engineer @ Schaerer

https://www.linkedin.com/in/patrick-wachsmuth/

https://www.schaerer.com/

Jonas Alder

Lead DevOps Engineer @ Zühlke

https://www.linkedin.com/in/jonas-alder/

https://www.zuehlke.com/

Agenda

What we do

Why we choose Microservices with Spring & Kafka

QR-Code based payment

Integration tests with Testcontainers

Running everything on Kubernetes

Headquarter in Switzerland

- Subsidiaries in Germany, Belgium, USA

450 employees

Operational in over 70 countries worldwide

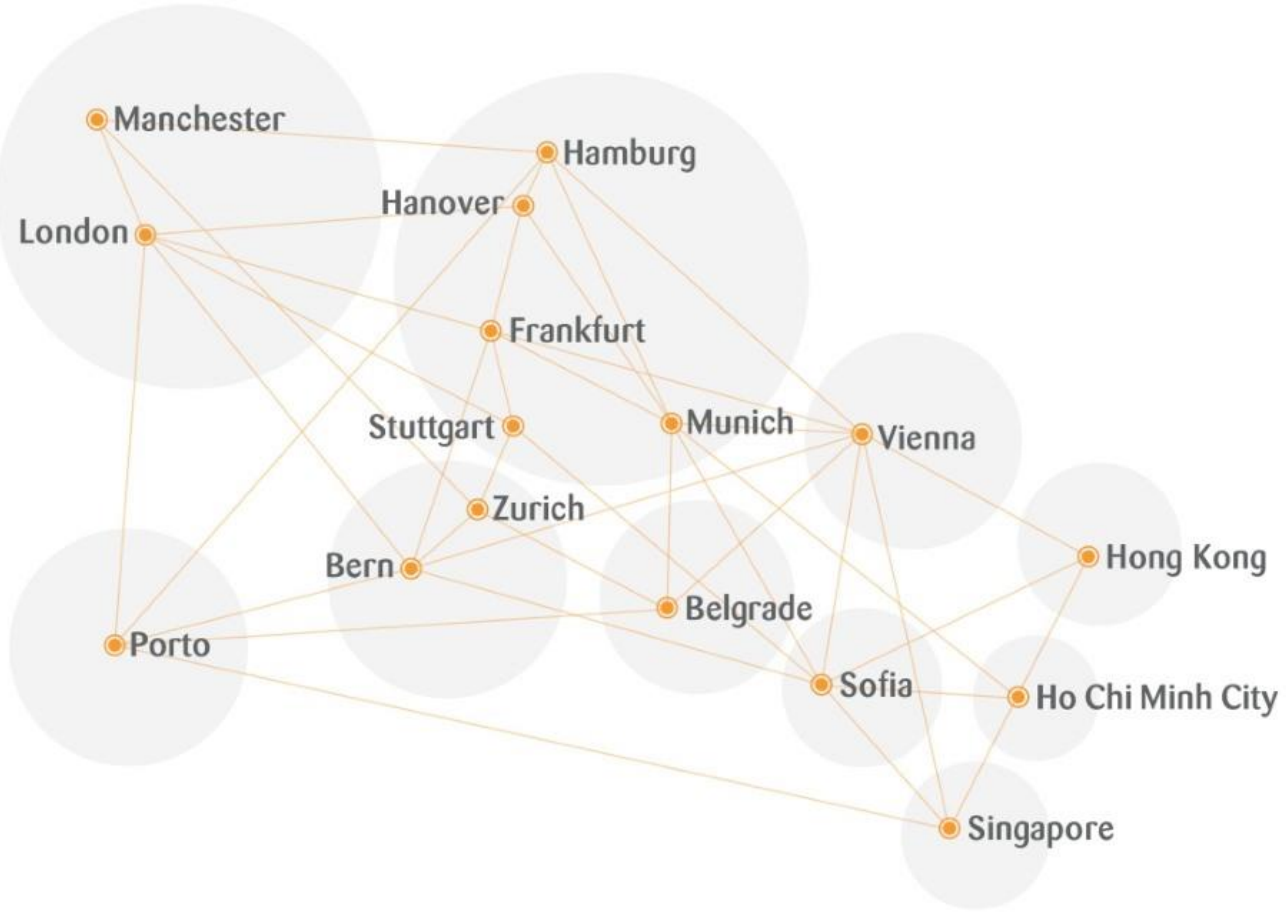
65’000 machines

3’000’000 coffees per day

3’000’000 machine events per day (bin full, milk low, …)

schaerer

swiss coffee competence



zühlke
empowering ideas

Founded in 1968

Over 10,000 software and product development projects

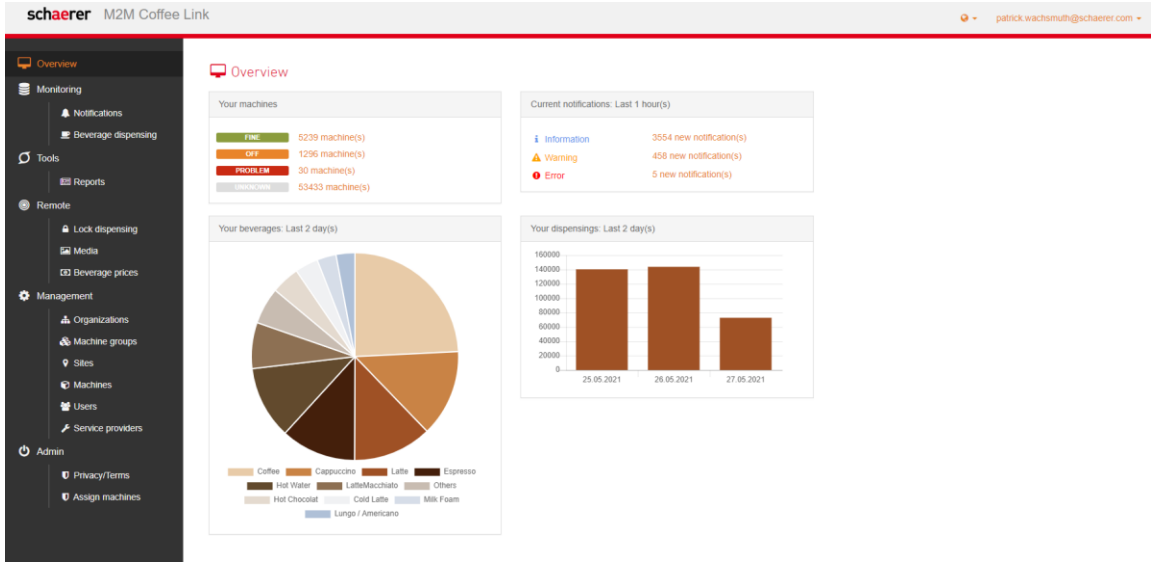
1300 employees

10% of turnover is invested into training & development

What we do – Build together

Coffeelink 3 (Legacy system)

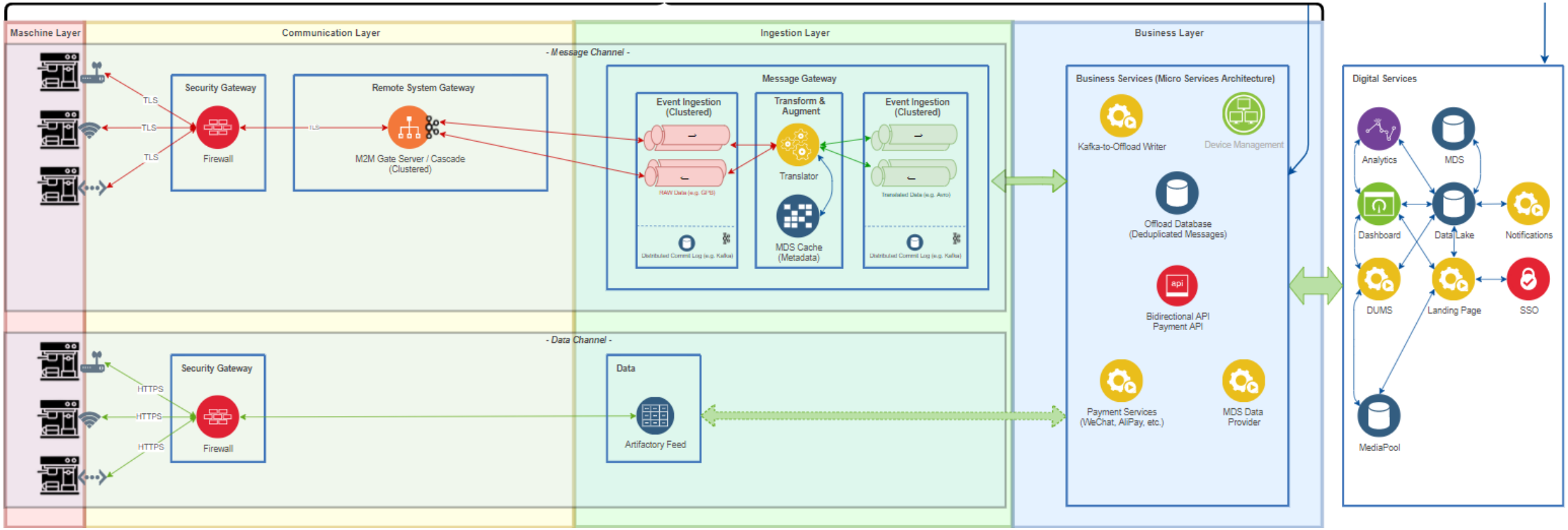
- Database centered monolithic application with UI
- Improvements
  - Decoupled reading and processing of data with messaging queue
- Blockers
  - Database performance due to high coupling on different tables
  - Long running queries for historical data and reports
  - Message-Payload not optimized for priorities
  - Monolithic architecture
  - Digital Platform (New UI driven by WMF for the same data)



New Telemetry Stack

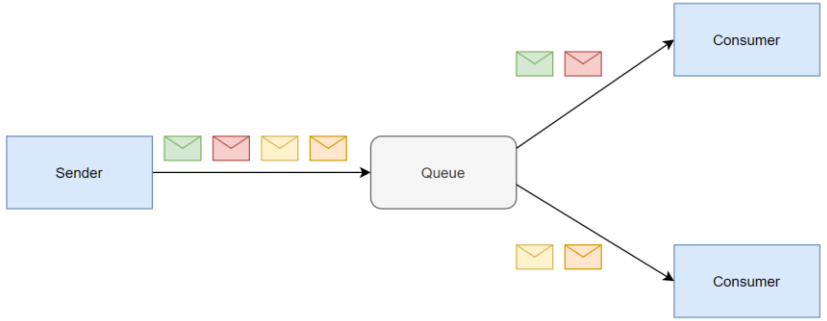
- Started in 2019
- Built platform with improved performance, flexibility and features
- Build & run platform with 3-4 DevOps engineers

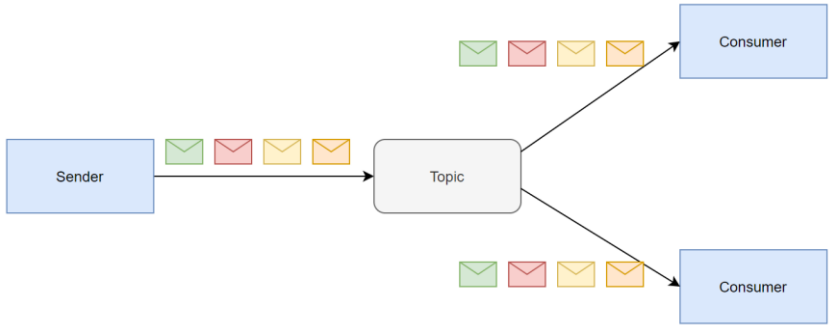
Architecture

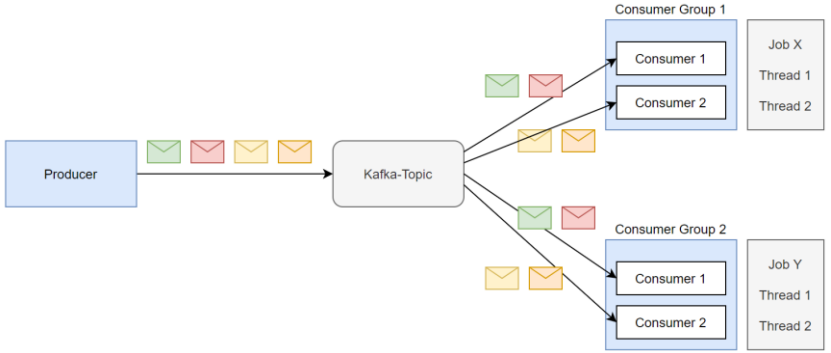


Architecture – Why Kafka?

What makes Kafka better than a Message Queue/Topic?



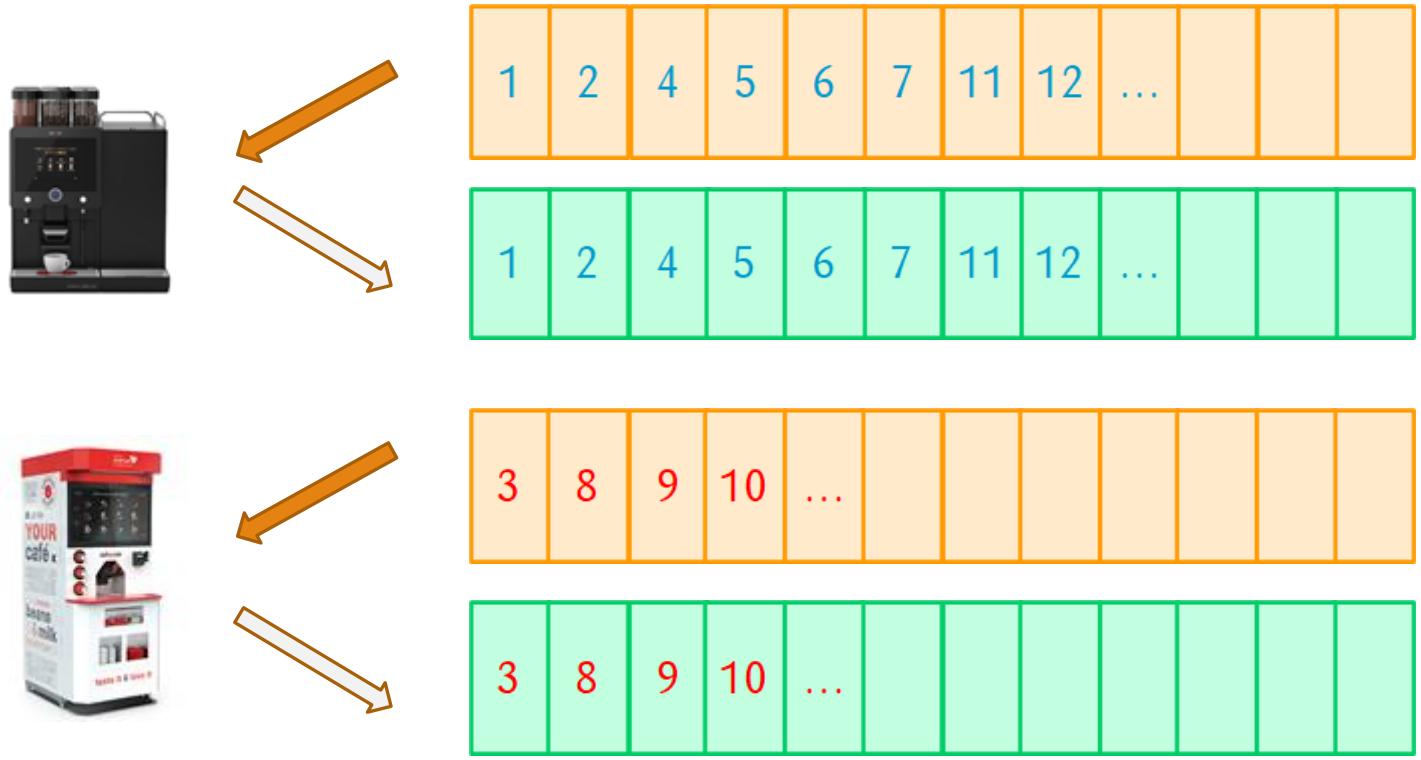


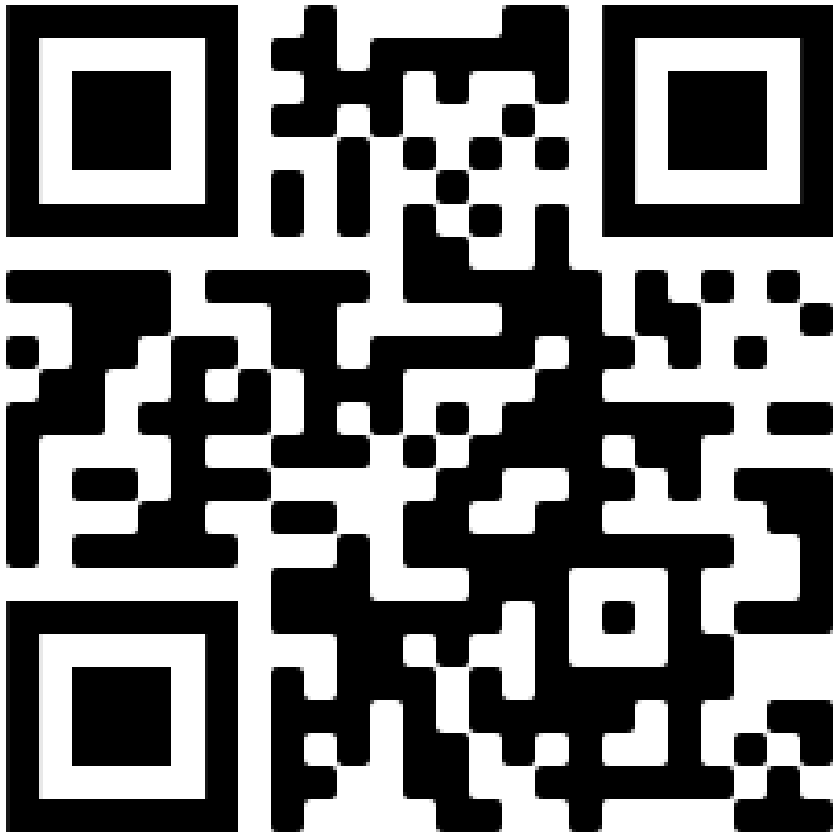


What is Kafka anyways?

- Distributed log
- Highly available
- High throughput – Horizontally scalable
- Replayable processing (persistence)
- Streaming
- Compaction

Architecture – Why Kafka?



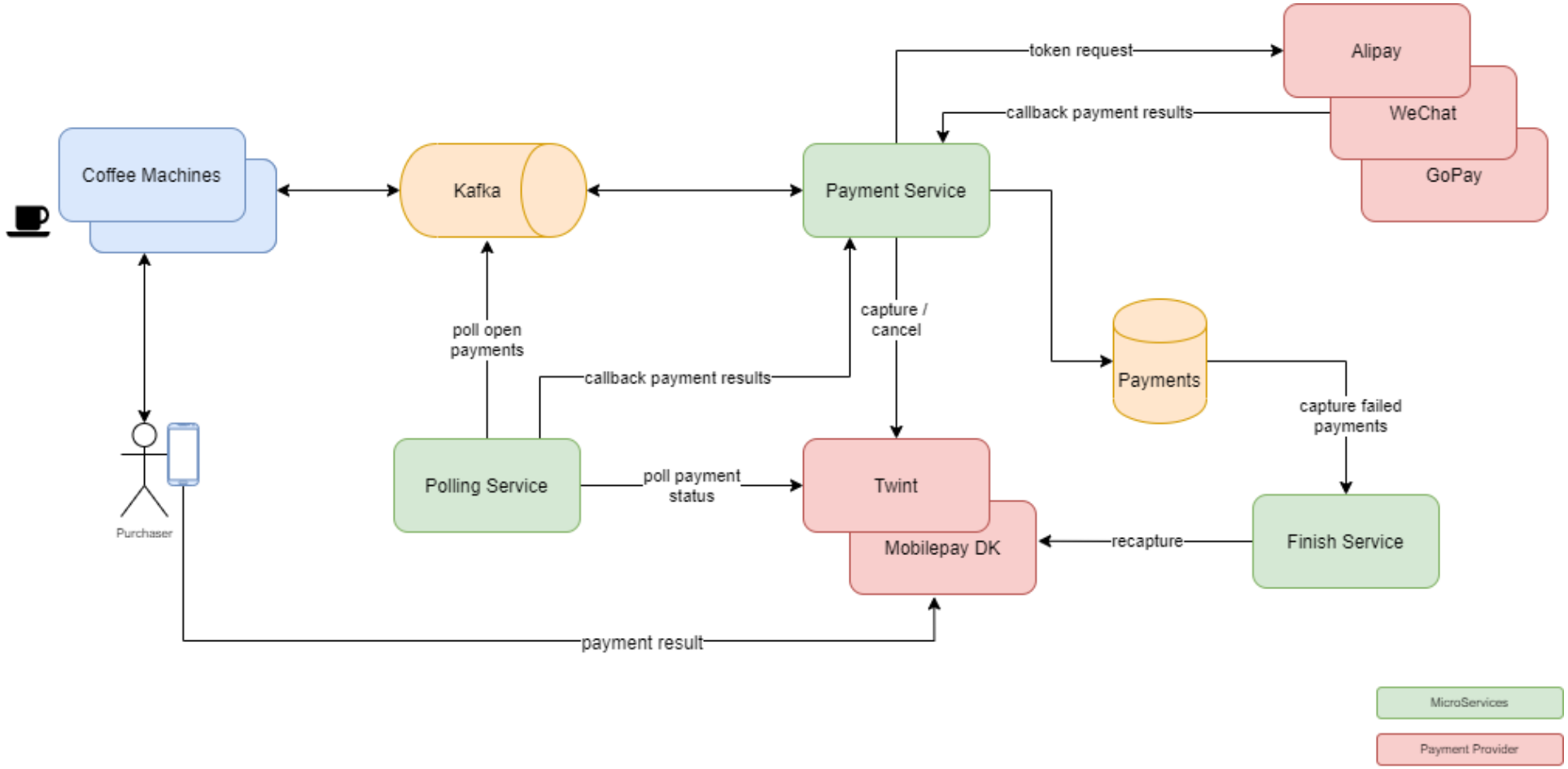


Architecture – Why Spring Boot?

Let's build a word cloud

Payment Providers - Flow

Payment Provider - Implementation



Payment Provider - Challenges

Feature diversity

- Some providers have hundreds of functions & flows
- Polling vs Notifications vs hybrid

Credentials management

- Different types (Keys, Certificates)
- Multiple credentials per payment provider
- Multiple environments
- Transfer of credentials to/from customers

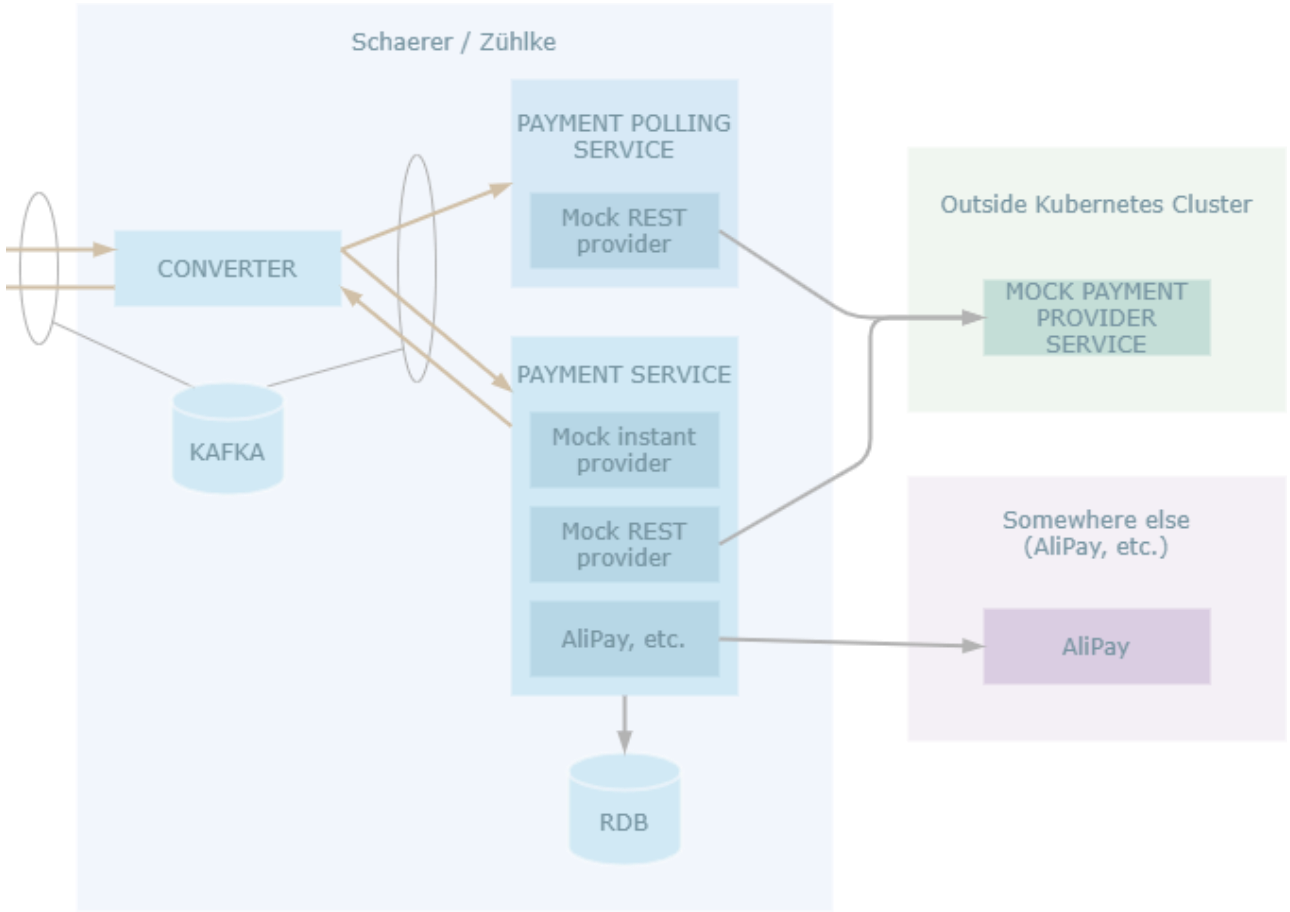
Payment Provider - Challenges

Chinese providers for the local market

WeChat: token request return code was FAIL, result code was null, unified order response = {return_msg=错误的签名，验签失败, return_code=FAIL}

Payment Provider - Challenges

Load Testing - Simulating external provider with desired behavior



MockServer

mock-server.com

Testcontainers for integration tests

Testcontainers is a JVM based utility to control and manage containers from code.

All developers need is a JVM and Docker installed.

Our testing scope:

- single Microservices
- Input and output either Kafka messages or REST calls

testcontainers.org

Testcontainers for integration tests

```
final var zookeeperContainer : ZookeeperContainer = new ZookeeperContainer().withNetwork(network);
final var kafkaContainer : KafkaContainer = new KafkaContainer(zookeeperContainer.getZookeeperConnect()).withNetwork(network);
final var schemaRegistryContainer : SchemaRegistryContainer = new SchemaRegistryContainer(zookeeperContainer.getZookeeperConnect()).withNetwork(network);

Startables
  .deepStart(Stream.of(zookeeperContainer, kafkaContainer, schemaRegistryContainer))
  .join();

assureTopicsAreAvailable(zookeeperContainer.getZookeeperConnectExternal());

TestPropertyValues.of(
  "spring.kafka.bootstrap-servers=" + kafkaContainer.getBootstrapServers(),
  "spring.kafka.properties.schema.registry.url=" + schemaRegistryContainer.getServiceURL()
).applyTo(applicationContext.getEnvironment());
```

```
@ContextConfiguration(initializers = {KafkaContextInitializer.class})
```

Testcontainers for integration tests

```
public ZookeeperContainer() throws IOException {
  super(getImage( imageName: "confluentinc/cp-zookeeper:5.5.1"));

  exposedPort = ContainerUtils.getRandomFreePort();

  final var env = new HashMap<String, String>();
  env.put("ZOOKEEPER_CLIENT_PORT", Integer.toString(ZOOKEEPER_INTERNAL_PORT));
  env.put("ZOOKEEPER_TICK_TIME", Integer.toString(ZOOKEEPER_TICK_TIME));
  withEnv(env);

  addFixedExposedPort(exposedPort, ZOOKEEPER_INTERNAL_PORT);
}

public String getZookeeperConnectExternal() {
  return String.format("%s:%d", getContainerIpAddress(), exposedPort);
}
```

Testcontainers for integration tests

What works well?

- Spring testing integration
- Mark context as dirty, containers get recreated
- No more integration test configuration hooks
- No more matching embedded version with production version.

```
@DirtiesContext
```

```
public class KafkaContextInitializer implements ApplicationContextInitializer<ConfigurableApplicationContext> {

  @Override
  public void initialize(final ConfigurableApplicationContext applicationContext) {
```

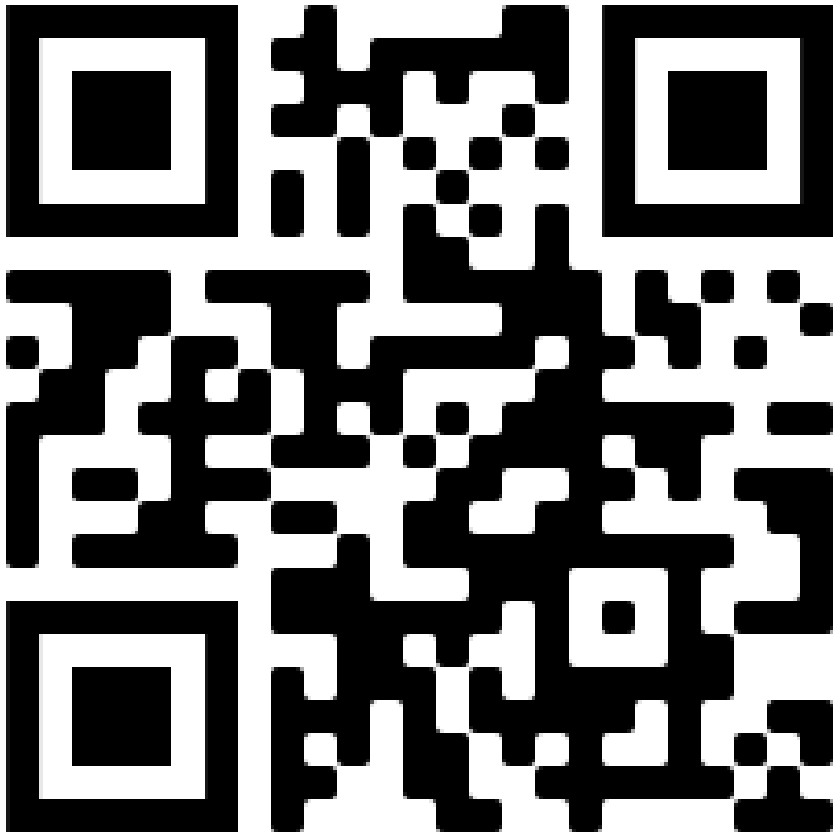
Testcontainers for integration tests

What causes pain?

- Containers with state might need to be recreated between tests
- Container creation speed is essential

Outlook & Ideas

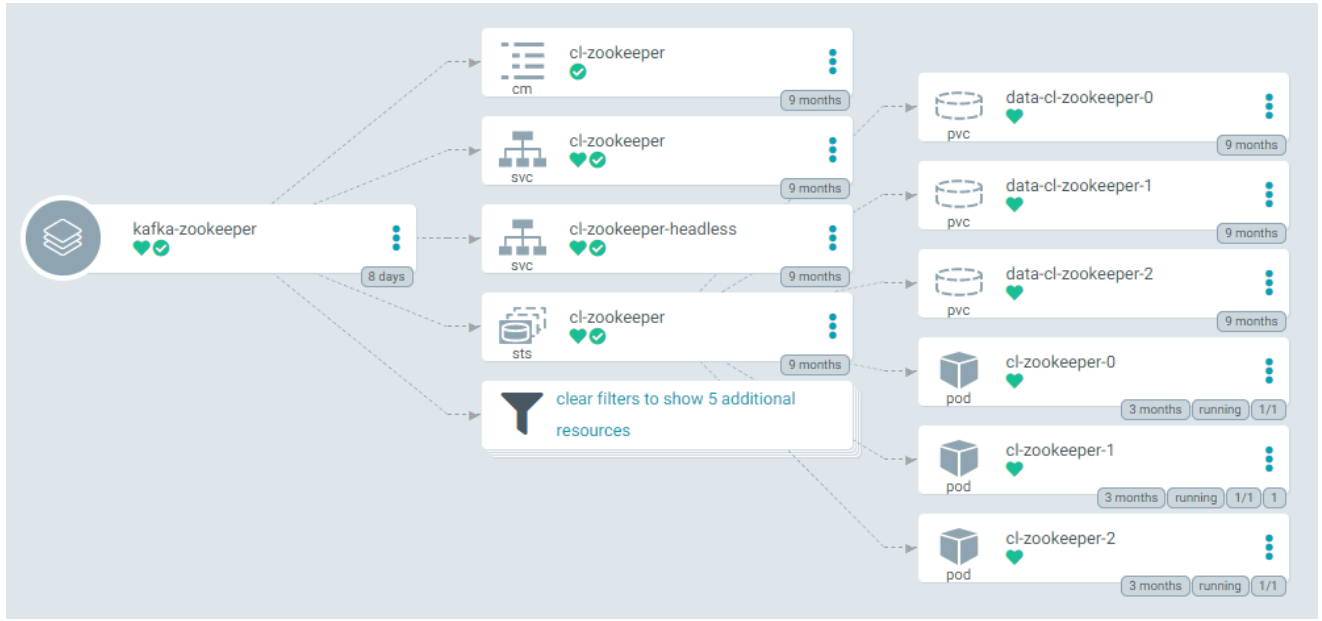
- Extend usage to MariaDB (still using H2)
- Testing multiple services together

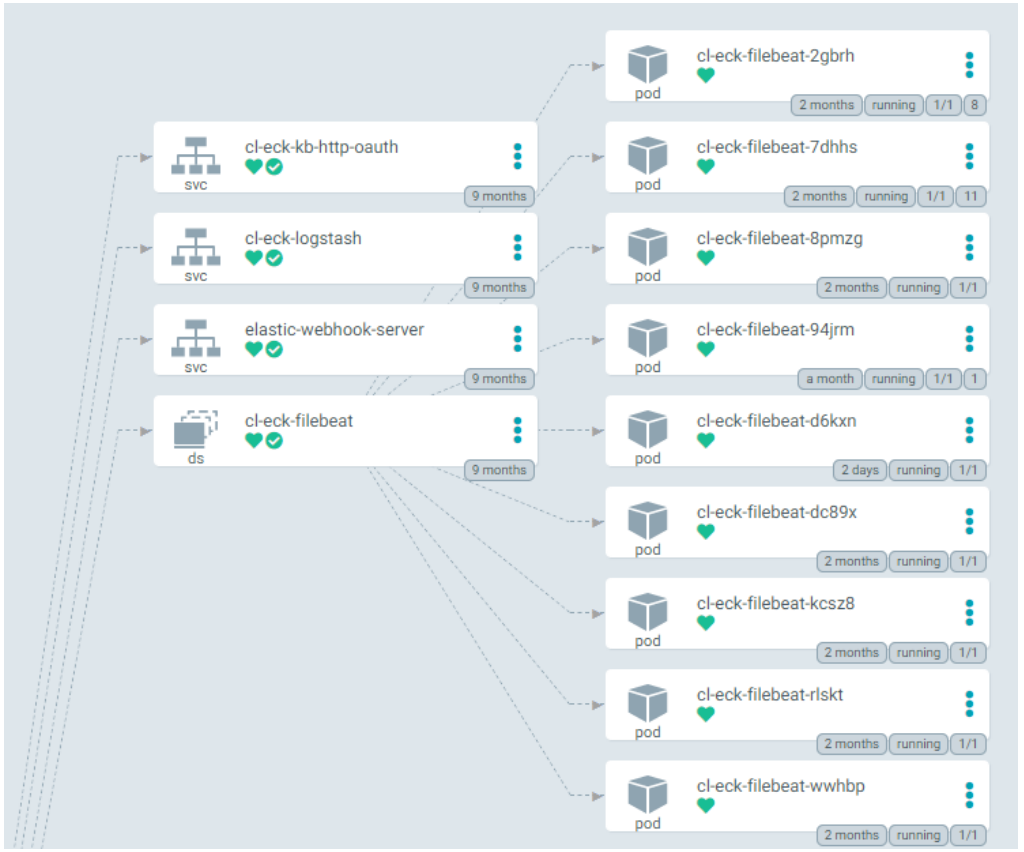


Running everything on Kubernetes

Test your basic knowledge what Kubernetes is

Running everything on Kubernetes



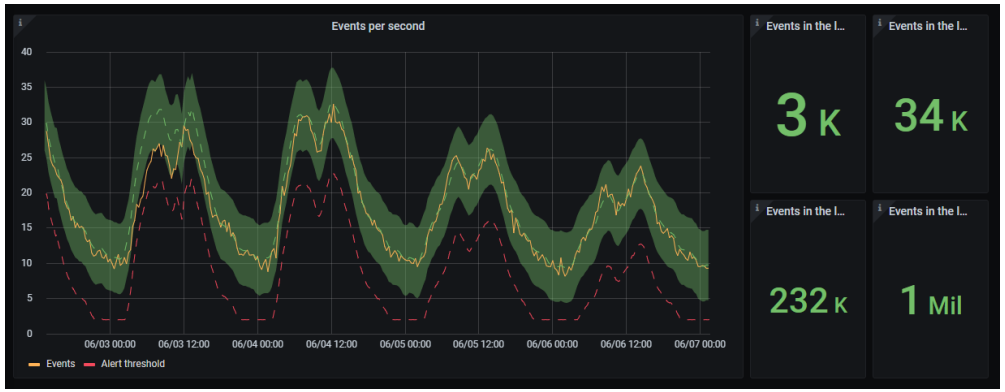


Running everything on Kubernetes

Why we decided to use Kubernetes as our Platform

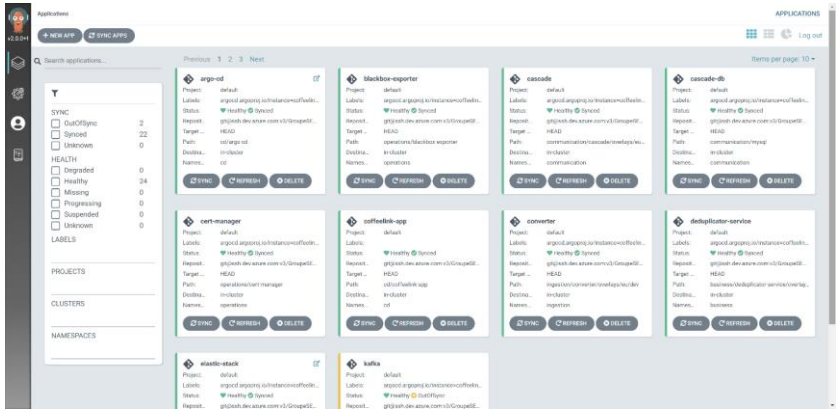
- Runs in Cloud or on Premise
- Scalability and availability
- IT costs optimization
- Faster time to market
- Zero-Downtime updates
- Control what we run with a small Team
- Security

Running everything on Kubernetes



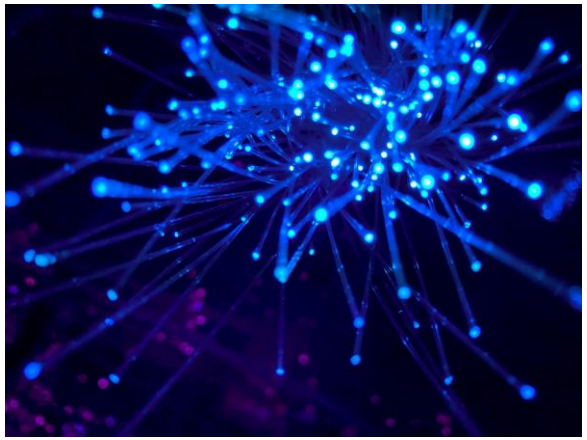
Monitoring is key

GitOps



Persistence impacts flexibility

Secret handling



Secured connections

Summary

Resilient and scalable Architecture thanks to Kafka & Kubernetes

Fast and easy development with Microservice Framework (SpringBoot)

Integration tests with same components as in production (Testcontainers)

GitOps approach with ArgoCD very helpful for small teams for fast deployments

Avoid single point of failures with monolithic applications or primary/secondary setup

Q&A