Chaos@Fire

GOALS

O1. Chaos engineering test our platform
- Test **broker and database performance** while scaling up the volume of messages
- Test **replication persistence** resistance

O2. Dimension the infrastructure required for scaling up

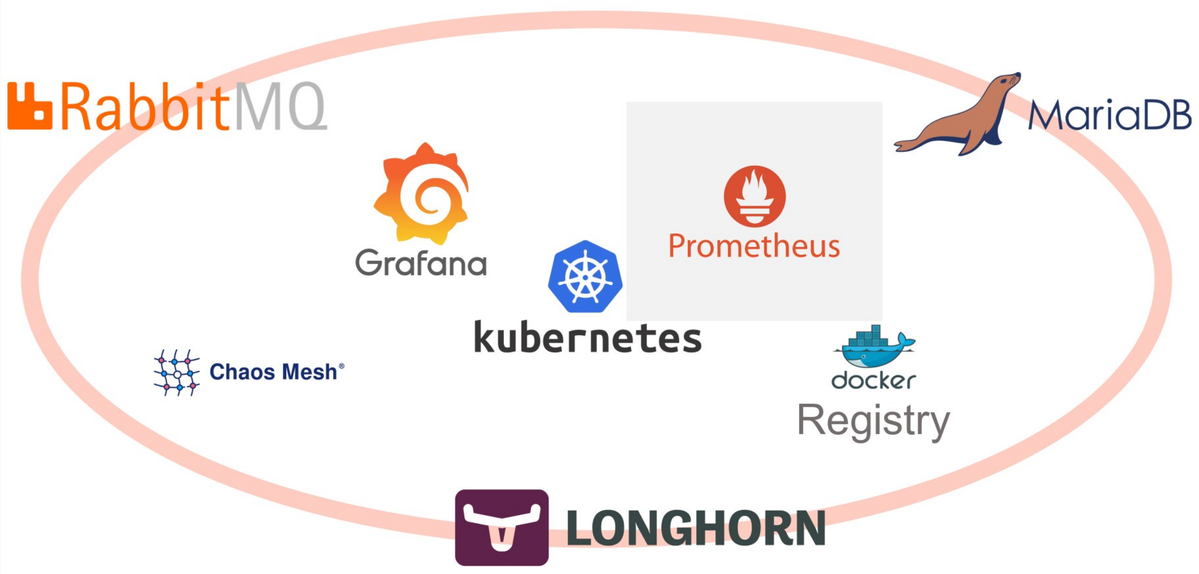
O3. Develop a **chaos engineering workflow**

CHALLENGES

- **Automatize** Kubernetes cluster creation
- Deploy **HA** database and broker
- **Replicate persistence**
- **Stress** the cluster using Chaos Mesh

DEMO SETUP

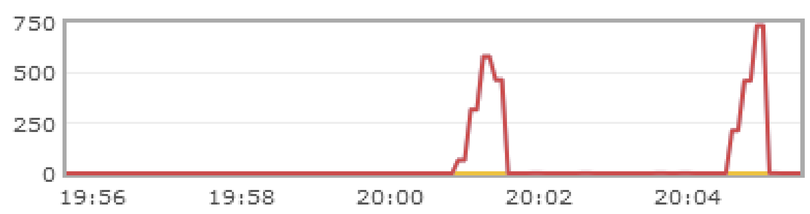
Stress broker and database with a message feeder

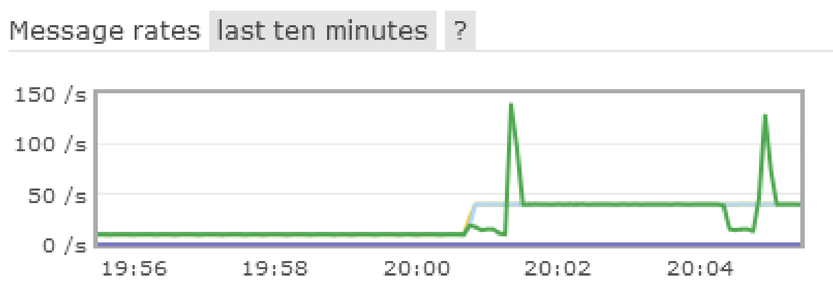


Services should remain available at any time

RESULTS

force injection latency → acummulation of messages → dump rate drop

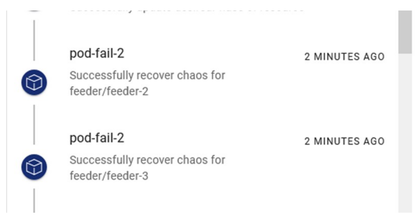


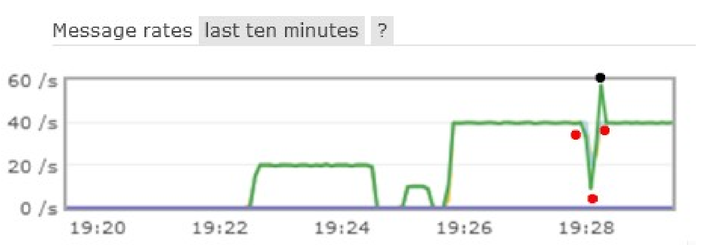


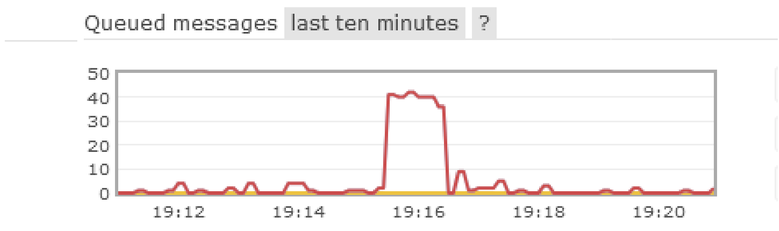
MORE RESULTS

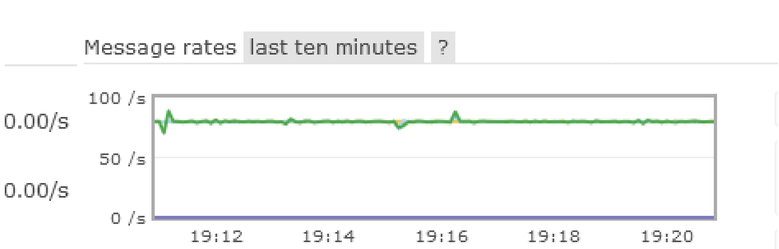
Automate scaling of consumers

feeder Pod fail on feeder → feeder automatically restarts pods → Drop of feeded messages and automatic recovery
mariadb Pod fail → mariadb cluster malfunctioning and little message queueing → Need of manual restoration









CONCLUSIONS

- **High availability** is a must in a reliable microservice architechture
- Chaos engineering is an extension of software testing. Need to implement a full **Chaos engineering cycle**
- Push platform to break point, check its **limits**
- **Grid5000** is a great infrastructure → test different **software possibilities**

POST MORTEM

Open questions:

- Deploy and test a HA **Kafka** cluster
- Best practices on replicating **influxDB**
- Develop **high availability** for our own **code** and test its resilience with learnt chaos engineering methods
- Create **self-scalable** microservices