Model-driven Generation of Microservice Architectures for Benchmarking Performance and Resilience Engineering Approaches

Thomas F. Düllmann
André van Hoorn
Motivation

- Microservice architectures are steadily gaining more adopters in practice (e.g., Netflix, Amazon, Zalando, OTTO)
- No known real-world examples for microservice architectures
- No known benchmarking applications for performance and resilience engineering approaches in microservice architectures
- Define meta model for microservice architecture
- Generate actual microservices based on model instances
- Run instrumented synthetic microservices
- Use monitoring data for benchmarking your approaches
Model-driven Generation of Microservice Architectures for Benchmarking Performance and Resilience Engineering Approaches
Model-driven Generation of Microservice Architectures for Benchmarking Performance and Resilience Engineering Approaches

Ecore Metamodel Instance

- Database (v 2.0.0)
  - getData '/data'

- Portal (v 1.0.0)
  - getIndex '/'

- Ecore Metamodel
- Instance

- platform/resource/MicroserviceMetamodel/model/MetaModelStructure.xmi
  - Meta Model Structure
    - Infrastructure Model
    - Configuration
      - Microservice 0000001
      - Microservice 0000002
    - Dependency Model
  - Operation To Operation Calling Dependency
    - Time Series
    - Microservice Repository
      - Microservice Type portal
        - REST Operation getIndex
        - Version 1.0.0
      - Microservice Type database
        - REST Operation getData
        - Version 2.0.0

- platform/resource/MicroserviceMetamodel/model/anotherMicroserviceMetamodel.ecore

- Properties
  - Called Microservice: Microservice Type database
    - Called Operation: REST Operation getData
    - Calling Microservice: Microservice Type portal
    - Calling Operation: REST Operation getIndex
    - Calling Version: Version 1.0.0
• Xtend template feature used for including data from meta model instance
• Generated microservices are based on Spring Boot [1]
• Monitoring is included by default (Kieker)
• Prepared for delay injection
• Further generation artifacts
  • Dockerfile to create Docker [2] container for the microservice
  • Kubernetes [3] kubernetes.yaml file for deployment on a cluster
• Result
  • Folder for each service with gen-<ServiceName>

[3] https://kubernetes.io
Based on the artifacts, the following steps lead to runnable microservices (example for the generated portal microservice):

- Pull dependencies for Spring Boot and build JAR file
  - `mvn clean package`

- Create Docker image
  - `docker build -t my/portal .`

- Deploy on Kubernetes cluster
  - `kubectl create -f kubernetes.yaml`
Execution

- Artifacts can be used to deploy generated microservices to Kubernetes cluster
- Supplemental microservices for evaluating the system under test (SUT)
  - Load generation (JMeter)
  - Monitoring data collection (JMS Server) and storing (Monitoring Server)
  - Delay injection (Registry, Injector)
Model-driven Generation of Microservice Architectures for Benchmarking Performance and Resilience Engineering Approaches
Future Work / Limitations

Feature set still limited but may be a base for **future work**:

- Improve the generator to use more properties of the model instance
- Use generated microservice architectures in the CASPA approach
- Dashboard for injections and status of microservices in SUT
- Extraction of meta model instance based on monitoring data
- More kinds of injections for different (performance) problems
  - Delay patterns (e.g., the ramp)
  - Resource demands (e.g., CPU, memory)
Conclusion

- Microservice generation based on Ecore meta model instance
- Actual microservices that are prepared to be deployed on a Kubernetes cluster
- Enables to run and monitor pre-defined microservice architectures
- Possible base for extensions in the area of performance and resilience engineering benchmarking
- Available on GitHub: 
  github.com/orcas-elite/arch-gen

- Questions/Discussion
  Contact: duellmann@informatik.uni-stuttgart.de