Microservices Architectures and Technical Debt: A Self-adaptation View

Ananya Chhonker and Rogério de Lemos
University of Kent, UK
Outline

- Motivation
- MSA and technical debt
- MSA and self-adaptation
- MSA and self-protection (security)
- Conclusions
Motivation

- MSA and self-adaptation
  - Self-healing, autonomic controller for deployment, regression testing, resilience, controller failure
    - Distributed nature of MSA
  - Insight taken: emergent and fuzzy
    - Centralised vs decentralised
- MSA and technical debt
  - There are no primary or secondary studies analysing the principles of MSA in light of TD
  - No guidelines for evaluating the debt capacity
Motivation: State-of-the-art

- Not much significant has been published on MSA and self-adaptation
  - Relative new field, not yet stable in terms of what is the key foci of change
  - Some misguided proposals
- (Surprisingly!!) Both are about change
  - Possible explanation: in software engineering, in particular, software architectures the basis has bee MAPE-K loop
    - Essentially, a centralized entity
  - Architecture vs deployment
    - Tools for handling change at deployment
      - Eg, Kubernetes
- 3 physical servers
- 10 VMs
- 22 micro-services
- 3 OSS packages
- 120+ Open API REST services
- ~300,000 lines of code (Java)

All deployed by a Continuous Delivery engine
MSA and Technical Debt

- Technical debt
  - design decisions that consciously or unconsciously compromise system-wide quality attributes
- Assumptions or methods can negatively affect an MSA-based software system during its lifetime
- Awareness of TD and its implications
  - Developers can be more cautious while making decisions
- To reduce TD
  - Quality of attributes of an MSA-based system should be analysed at architecture level
    - Testing, security and reliability
MSA and Technical Debt

- **Testing**
  - Large systems with many connections between services makes integration testing more challenging

- **Security**
  - Attack area increases since there are more points of entry, lack of global view since containers deployed in the cloud, individual microservices may not be trustworthy, system complexity, etc

- **Reliability**
  - Complex deployments with many services and connections tend to reduce the reliability of services, nothing to handle failures in connections
**Self-adaptation**

- A system is able to modify its behaviour and/or structure in response to changes

- Feedback control loop, and software engineering techniques

**Testing**

- Generate automatically new integration test plans when changes in the architecture

**Security**

- Identify malicious components and protect the MSA-based system

**Reliability**

- Known as self-healing
Type of Control

- From centralised to decentralised
  - Single component or distributed amongst several components
- Self-adaptation
  - Feedback control loop, relies on system models from different perspectives
- Self-organization
  - Multi-agents, ant colony optimisation, swarms
- There is no single unique solution for self-adaptive MSA
  - Orchestration vs choreography
  - Several factors involved in controlling functional and non-functional attributes
    - Specialisation vs generalization of controllers
Patterns for Decentralized Control in Self-Adaptive Systems [Weyns 2013]
Security services should rely on orchestration

- Information sharing, each node cannot maintain a model of system, compromised nodes lack self-awareness, etc.

Detection and protection against malicious containers

- Containers may have the right to access other services, but they abuse
- Their malicious behaviour should be detected
- Containers should be isolated from the system
  - Impact on reliability or availability is another analysis
7b. Adapt containers access

Identity Services

6a. Containers rights

4. Validate credentials

3. Request access decision

2. Request services using credentials

1. Authenticate & get credentials

6b. Monitor Resource usage

Container

5. Send access decision

6a. Monitor Access

4. Validate credentials

3. Request access decision

2. Request services using credentials

1. Authenticate & get credentials

7a. Adapt access criteria

6b. Monitor Resource usage

Criteria for access

Container access rights

Criteria for credential release

Self-protection against Malicious Microservices

Container access rights

Criteria for credential release

Autonomic Controller

Identity Services

Authorisation Services

Container

MSA 2019 – February 2019 – 12
Conclusions

- Design decisions lead to trade-offs
  - MSA has benefits, but also has some drawbacks
  - Lack of discussion regarding technical debt
    - What about lessons learned from similar SE initiatives?

- Self-adaptation offers an opportunity for handling change
  - A single MAPE-K loop is not the solution
  - Resilience of MSA-based microservices comes with a cost related to complexity

- The notion of ‘autonomy’ is misplaced
  - Functional and non-functional

- How to protect MSA-based solutions against malicious microservices
Thank you!