Microservices beyond COVID-19

Antonio Brogi

Department of Computer Science
University of Pisa, Italy
Q: Beyond COVID-19?

A: Sorry, just a dirty trick to attract audience :)

15:00 Coffee Break
15:30 Keynote: Microservices beyond COVID-19
16:30 Closing
Microservices, microservices, microservices ...
Microservices

Main motivations

(1) Shorten lead time for new features/updates
   - accelerate rebuild and redeployment
   - reduce chords across functional silos

(2) Need to scale, effectively
   - millions of users
OK but ... what are microservices?
Microservices

Applications = sets of services
  + each running in its own process container
  + communicating with lightweight mechanisms
  + built around business capabilities
  + decentralizing data management
  + independently deployable
  + horizontally scalable
  + fault resilient
  + DevOps culture and tools!

(service-orientation done right?)
Microservices

- shorter lead time
- scaling
Does my app respect the "microservices principles"?

If not, how can I refactor it?
Microservices, microservices, microservices ...
Design principles, architectural smells and refactoring
How can architectural **smells** affecting design **principles** of microservices be detected and resolved via **refactoring**?
A multivocal review

Recent review of white and grey literature aimed at identifying

- the most recognised *architectural smells* for microservices, and
- the architectural *refactorings* to resolve them

(review of 41 studies presenting architectural smells & refactorings for resolving them)

Design principles

Independent deployability
The microservices forming an application should be independently deployable

Horizontal scalability
The microservices forming an application should be horizontally scalable

Isolation of failures
Failures should be isolated

Decentralization
Decentralisation should occur in all aspects of microservice-based applications, from data management to governance
Architectural smells

- multiple services in one container
- no API gateway
- endpoint-based service interactions
- wobbly service interactions
- independent deployability
- horizontal scalability
- isolation of failures
- decentralisation

- ESB misuse
- shared persistence
- single-layer teams
Multiple services in one container

- independent deployability
- multiple services in one container
- package each service in a separate container
Endpoint-based service interactions

Horizontal scalability

- Add service discovery
- Add message router
- Add message broker

Invocation to specific instance

(e.g. message queue)

- Add message broker (w:14%; o:4)
- Add service discovery (w:55%; o:16)
- Add message router (w:31%; o:9)

(e.g. load balancer)
No API gateway

App clients must invoke directly app services
(similar to endpoint-based service interaction smell)

Refactoring: add API gateway (that can be useful also for authentication, throttling, ...)
The interaction of m1 with m2 is *wobbly* when a failure of m2 can trigger a failure of m1.
Shared persistence

Multiple services access/manage the same DB
Shared persistence

- **split database**
  - db splitted
  - small changes to s1, s2
  - not always possible/easy to implement
  - eventual data consistency for replicated data

- **add data manager**
  - dm added
  - very small changes to s1, s2
  - added communication overhead

- **merge services**
  - s1 and s2 merged into single service
  - not always easy to implement
ESB misuse may lead to undesired centralisation of business logic and dumb services

Smart endpoints & dumb pipes!
Single-layer teams
Microservices, microservices, microservices ...
Design principles, architectural smells and refactoring
- μFreshener
μFreshener

A web-based GUI for
• editing app specifications
• automatically identifying architectural smells
• applying architectural refactorings to resolve the identified smells

Excerpted principle-smell-refactoring taxonomy

- **horizontal scalability**
  - no API gateway
    - add API gateway
  - endpoint-based service interaction
    - add service discovery
    - add message router
    - add message broker
    - add circuit breaker
    - use timeout
  - wobbly service interaction
    - split data store
    - add data manager
    - merge services

- **isolation of failures**
  - wobbly service interaction

- **decentralisation**
  - shared persistence
    - add data manager
    - merge services
Modelling application architecture

Graphical representation (of μTOSCA model)

- service
- data store
- message broker
- message router
- edge
- dynamic discovery
- circuit breaker
- timeout

Example
μFreshener: horizontal scalability

endpoint-based service interaction
μFreshener: isolation of failures

wobbly service interaction

<table>
<thead>
<tr>
<th>add circuit breaker</th>
<th>use timeout</th>
<th>add message broker</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Add Circuit Breaker" /></td>
<td><img src="image2" alt="Use Timeout" /></td>
<td><img src="image3" alt="Add Message Broker" /></td>
</tr>
</tbody>
</table>
μFreshener: decentralisation

shared persistence
Remarks 1/2

- mFreshener (freely) usable to analyse & refactor microservice-based apps
  - industrial case study
    - 4 no API gateway smells
    - 1 shared persistence smell
  - controlled experiment (100% vs. 49% smells identified, 83% vs. 1% resolved all smells)

- a smell is not necessarily a principle violation

- “let it be” refactoring supported
Remarks 2/2

• $\mu$Freshener works at the architecture level
  concrete implementation of refactoring left to application manager – much like in design patterns

• scalability: $\mu$Freshener features team-based view

• ongoing work: dealing with container orchestration
Can I play with \(\mu\)Freshener?

https://github.com/di-unipi-socc/microFreshener
Microservices, microservices, microservices ...
Design principles, architectural smells and refactoring
From incomplete specs to running apps
Motivations

- Microservice-based applications integrate many interacting services

→ Need to select an appropriate runtime environment for each microservice

→ Need to package each microservice into the selected runtime environment
Idea (1/2)

Exploit the TOSCA-based representation of microservice-based applications to specify only the application components and the software support they need.
node_filter:
  type: tosker.nodes.Container
  properties:
    - supported_sw:
      - mvn: 3.x
      - java: 1.8.x
      - git: x
    - ports:
      - 8080: 8000
      - os_distribution: ubuntu

node_filter:
  type: tosker.nodes.Container
  properties:
    - supported_sw:
      - node: 6.x
      - npm: 3.x
      - git: x
    - ports:
      - 3000: 8080
Idea (2/2)

Develop a tool for **automatically completing (and updating)** TOSCA application specifications by discovering and including **Docker-based runtime environments** providing the software support needed by each microservice.
$ toskerise thinking.csar --policy size
$ toserise thinking.completed.csar -f --policy most_used

Motivations

- Microservice-based applications integrate many interacting services

→ Need to select an appropriate runtime environment for each microservice

→ Need to package each microservice into the selected runtime environment
Idea

Develop a tool to **automate the deployment on top of existing container orchestrators**

**Ingredients:**
- a *process management* system inside containers
- a *service* for component-aware orchestration
- a *packager* capable of deploying on existing container orchestrators
Case studies

Thinking

Sock Shop

- 7 software components
- 14 containers (7 standalone)
- Deploy on Cluster of 4 VMs with Docker Swarm
Can I play with these tools too?

https://github.com/di-unipi-socc/TosKeriser
https://github.com/di-unipi-socc/DockerFinder
https://github.com/di-unipi-socc/toskose
One sec … do I have to write myself the TOSCA spec of my app?
Microservices, microservices, microservices ...
Design principles, architectural smells and refactoring
From incomplete specs to running apps
Mining the architecture of microservice-based apps
Automatically deriving the architecture of **black-box** applications

- **Step 1: Static Mining**
  - Partial topology graph
  - Eliciting services & databases

- **Step 2: Dynamic Mining**
  - Complete topology graph
  - Monitoring interactions among services & databases

- **Step 3: Refinement**
  - Refined topology graph
  - Identifying integration components

+ Marshalling obtained architecture to TOSCA

---


[https://github.com/di-unipi-socc/microMiner](https://github.com/di-unipi-socc/microMiner)
Case studies

Online boutique

Sock shop

Robot shop
Microservices, microservices, microservices ...
Design principles, architectural smells and refactoring
From incomplete specs to running apps
Mining the architecture of microservice-based apps
Concluding remarks
Take-home message: A (minimal) modelling of microservice-based applications can considerably simplify their design and analysis and allow automating their container-based completion and deployment.
Many interesting research directions on microservices (non-exhaustive, biased list):

- DSLs for microservices
- Security
- Monitoring
- Identifying failure causalities
- Continuous reasoning
- Green computing

• ...
... and thanks to

J. Soldani  D. Neri  O. Zimmermann  M. Bogo  G. Muntoni  L. Rinaldi
Microservices beyond COVID-19

Antonio Brogi

Department of Computer Science
University of Pisa, Italy