AARHUS UNIVERSITET

# Microservices and DevOps

## Scalable Microservices

### MicroService exercises

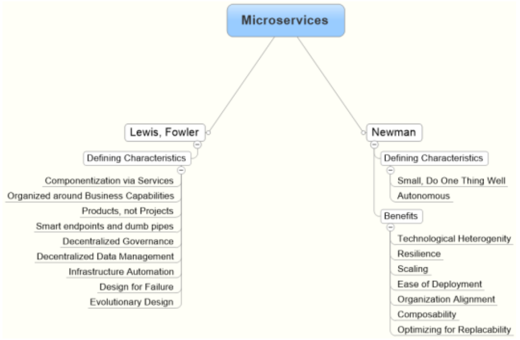
Henrik Bærbak Christensen

# Exercise 1

- Using the Fowler/Newman definitions of microservices, analyze SkyCave's architecture and…

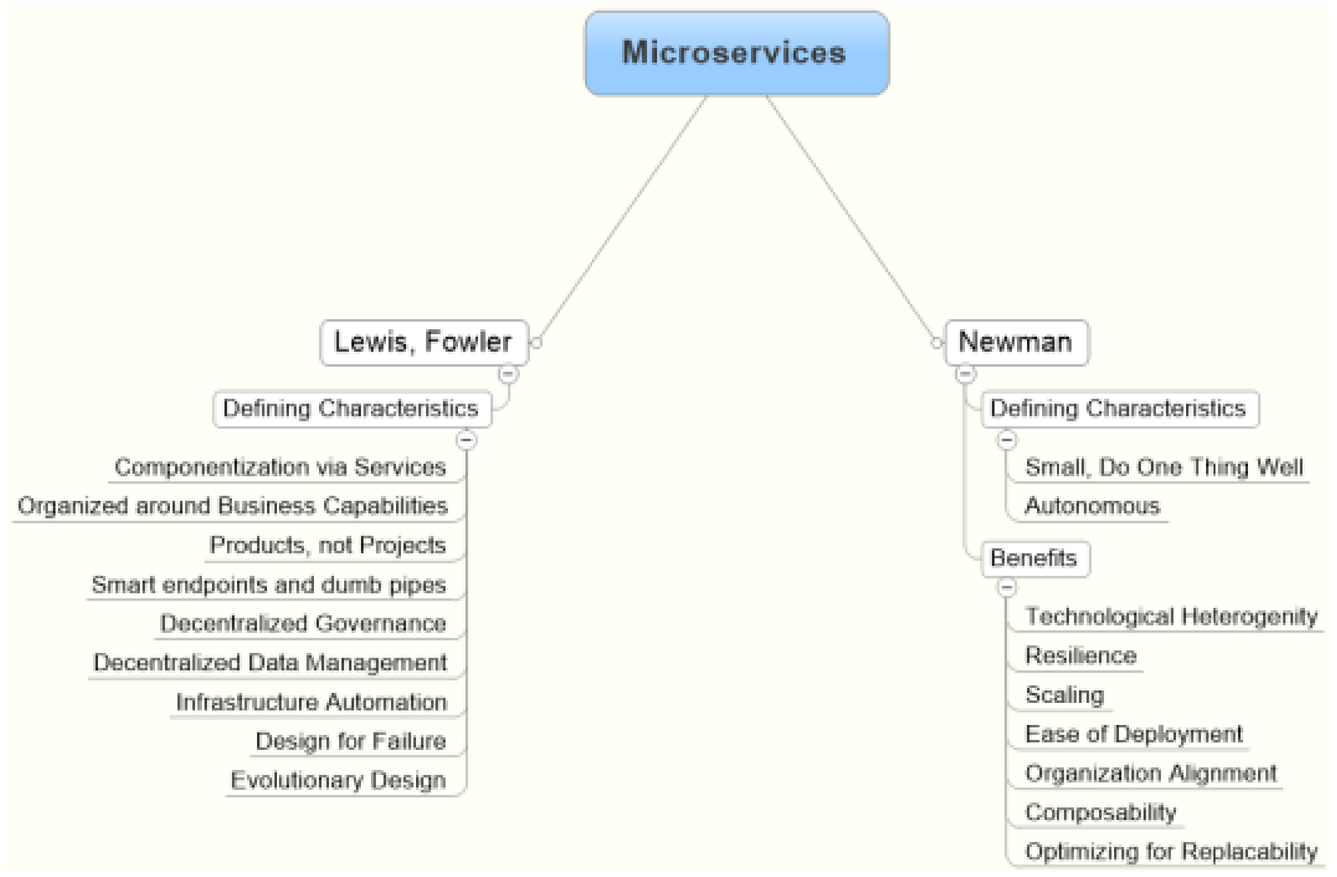
- *Argue that SkyCave is a MS architecture*
  - *Based upon the aspects it has, that are MS style*

- *Argue that SkyCave is **not** a MS architecture*
  - *Based upon the aspects, that collide with the MS style*



# Exercise 1

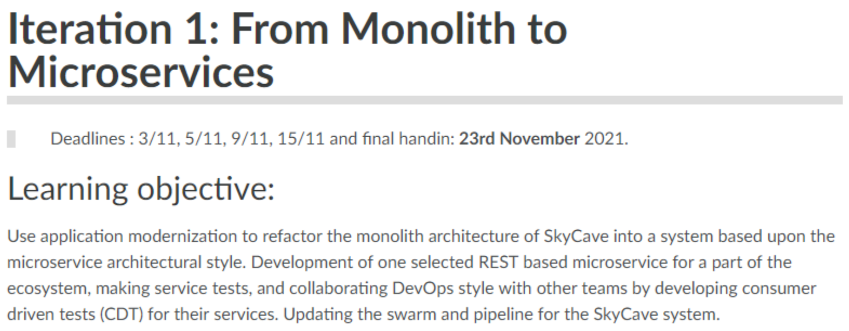
- Pro? Con?



AARHUS UNIVERSITET

## Exercise 2

- Start on the First Mandatory Exercise' *Strangling aspect*



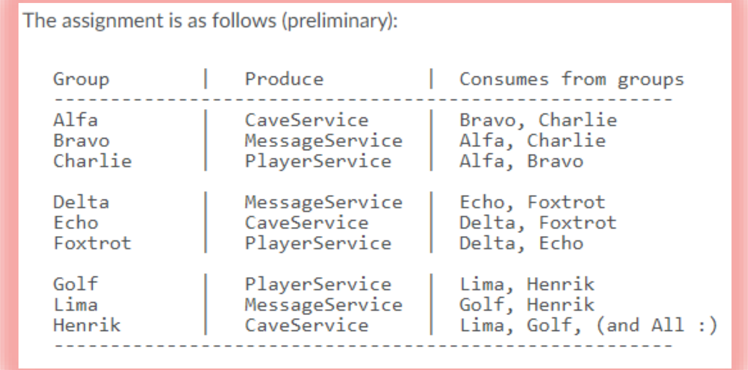
### Iteration 1: From Monolith to Microservices

Deadlines : 3/11, 5/11, 9/11, 15/11 and final handin: **23rd November** 2021.

#### Learning objective:

Use application modernization to refactor the monolith architecture of SkyCave into a system based upon the microservice architectural style. Development of one selected REST based microservice for a part of the ecosystem, making service tests, and collaborating DevOps style with other teams by developing consumer driven tests (CDT) for their services. Updating the swarm and pipeline for the SkyCave system.

Who are the groups?



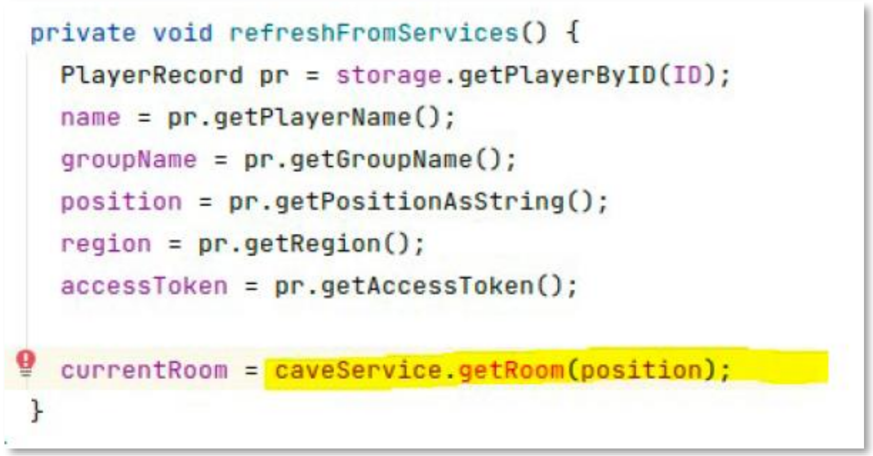
The assignment is as follows (preliminary):

| Group | Produce | Consumes from groups |
|---|---|---|
| Alfa | CaveService | Bravo, Charlie |
| Bravo | MessageService | Alfa, Charlie |
| Charlie | PlayerService | Alfa, Bravo |
| Delta | MessageService | Echo, Foxtrot |
| Echo | CaveService | Delta, Foxtrot |
| Foxtrot | PlayerService | Delta, Echo |
| Golf | PlayerService | Lima, Henrik |
| Lima | MessageService | Golf, Henrik |
| Henrik | CaveService | Lima, Golf, (and All :) |

AARHUS UNIVERSITET

## Exercise 2

- As argued, you can start strangling right away!
- Example:
  - *My group must develop 'CaveService'*
    - *Obvious "small step" is to strangle all Player calls to CaveStorage that do "cave service responsibilities' into calling a FakeCaveService*
  - I.e.
    - storage.getRoom(p)
    - →
    - caveService.issueGETrequest(p)

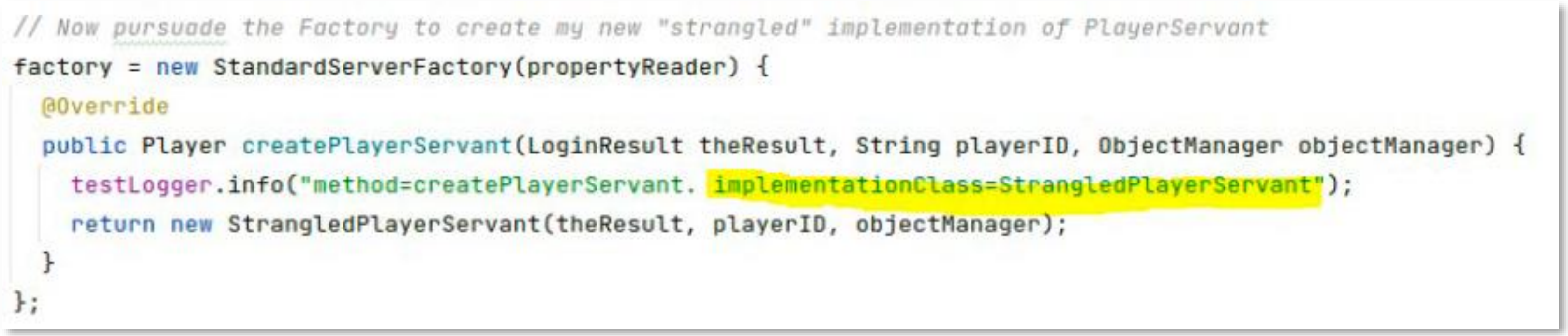


```
private void refreshFromServices() {
  PlayerRecord pr = storage.getPlayerByID(ID);
  name = pr.getPlayerName();
  groupName = pr.getGroupName();
  position = pr.getPositionAsString();
  region = pr.getRegion();
  accessToken = pr.getAccessToken();

  currentRoom = caveService.getRoom(position);
}
```

# Exercise 2

- I advice to
  - Branch your repo to a ‘strangling branch’
    - Support ‘do over’ – all is shit code!!!
  - Maintain old PlayerServant in parallel; by…
  - … using the factory system to create the new implementation:

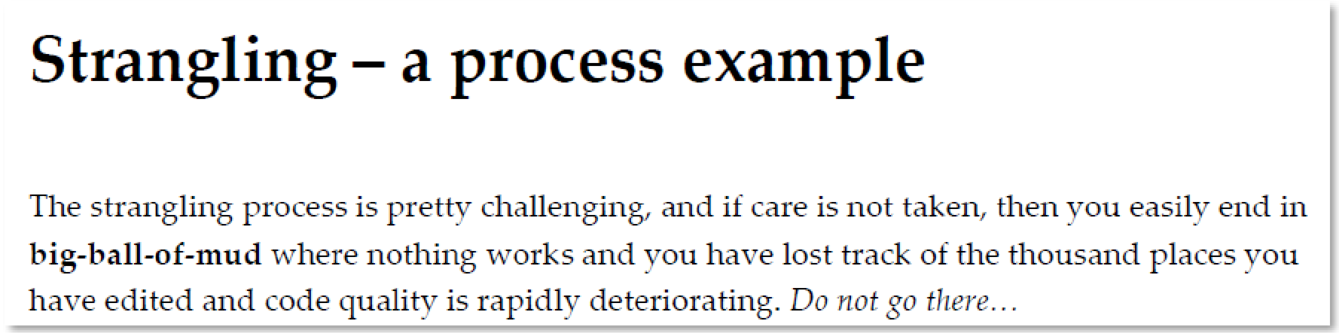


```java
// Now pursuade the Factory to create my new "strangled" implementation of PlayerServant
factory = new StandardServerFactory(propertyReader) {
  @Override
  public Player createPlayerServant(LoginResult theResult, String playerID, ObjectManager objectManager) {
    testLogger.info("method=createPlayerServant, implementationClass=StrangledPlayerServant");
    return new StrangledPlayerServant(theResult, playerID, objectManager);
  }
};
```

AARHUS UNIVERSITET

## Exercise 2

- Have a look at my guide, if it seems a bit scary…



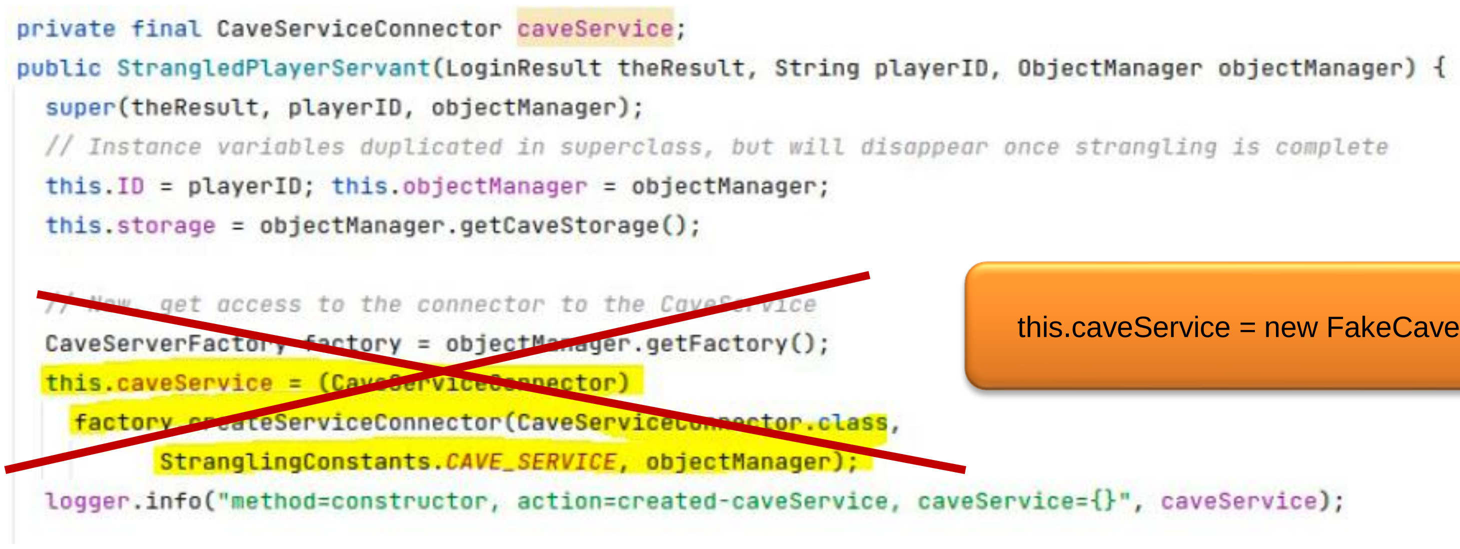
### Strangling – a process example

The strangling process is pretty challenging, and if care is not taken, then you easily end in **big-ball-of-mud** where nothing works and you have lost track of the thousand places you have edited and code quality is rapidly deteriorating. *Do not go there…*

- However, just 'hardwire the FakeCaveService()' instead of using the CFP system, it is a *smaller step*…
  - *Fake it till you make it…*

# Exercise 2

- Fake it till you make it…



```
private final CaveServiceConnector caveService;
public StrangledPlayerServant(LoginResult theResult, String playerID, ObjectManager objectManager) {
  super(theResult, playerID, objectManager);
  // Instance variables duplicated in superclass, but will disappear once strangling is complete
  this.ID = playerID; this.objectManager = objectManager;
  this.storage = objectManager.getCaveStorage();

~~  // Now, get access to the connector to the CaveService~~
~~  CaveServerFactory factory = objectManager.getFactory();~~
~~  this.caveService = (CaveServiceConnector)~~
~~    factory.createServiceConnector(CaveServiceConnector.class,~~
~~          StranglingConstants.CAVE_SERVICE, objectManager);~~
  logger.info("method=constructor, action=created-caveService, caveService={}", caveService);
```

this.caveService = new FakeCaveService();

## Exercise 3

- Create one or more QAS for SkyCave that express reasonable architectural requirements for
  - Availability QA
  - Modifiability QA
  - Performance QA
  - Testability QA

- Next, evaluate if SkyCave meets these requirements