AARHUS UNIVERSITET

Microservices and DevOps

Scalable Microservices

Mandatory 2.1

Splitting SkyCave – From Three tier to µService

Henrik Bærbak Christensen

SkyCave Architecture

- SkyCave was never conceived as a MS architecture
  - And designed as a classic *three-tier architecture*
    - UI -> Application -> Persistence
    - Cmd -> PlayerServant -> CaveStorage

- But I design in the *responsibility-centric perspective*
  - Roles that encapsulate cohesive responsibilities
  - Roles as *Program to an Interface*
    - *Cave, Player, CaveStorage, …*
  - Collaboration through *Favor Object Composition*

SkyCave Architecture

- If you review the central *Player* role, it is actually kind of a **Mediator**, API Gateway, type role
  - A player deals with different aspects of the cave experience, and the Player role encapsulate the interaction with these ‘sub roles’

• I dig a room to the north	Modifying the Room Matrix/Cave
• I post a message here	Modifying the Wall messages
• I move to a new room	Modifying the Player’s own data

- These sub-responsibilities *(sh/c)ould have been* extracted into sub-roles, each with their own interface…

Sub-Responsibilities

- … related to **Rooms**
  - CaveStorage: addRoom, getRoom, getSetOfExitsFromRoom, updateRoom
- ... related to **Players**
  - CaveStorage: getPlayerByID, computeListOfPlayersAt, updatePlayerRecord
- ... related to **Messages**
  - CaveStorage: addMessage, updateMessage, getMessageList
- And they are actually **completely orthogonal** to each other
  - Ideal borderlines for services (But, no real *shared models* ☹)

Service Responsibilities

- … related to Rooms
  - CaveStorage: addRoom, getRoom, getSetOfExitsFromRoom, updateRoom

CaveService

- ... related to Players
  - CaveStorage: getPlayerByID, computeListOfPlayersAt, updatePlayerRecord

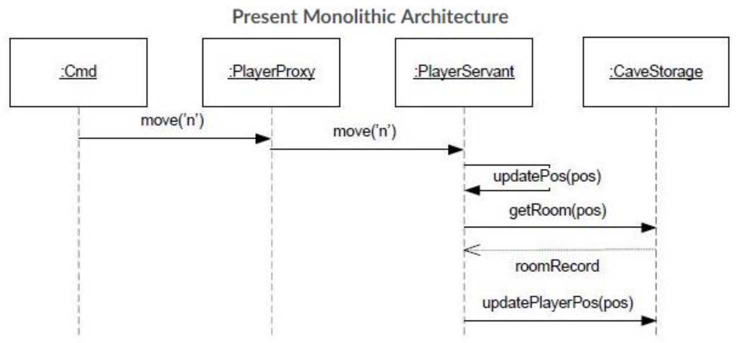
PlayerService

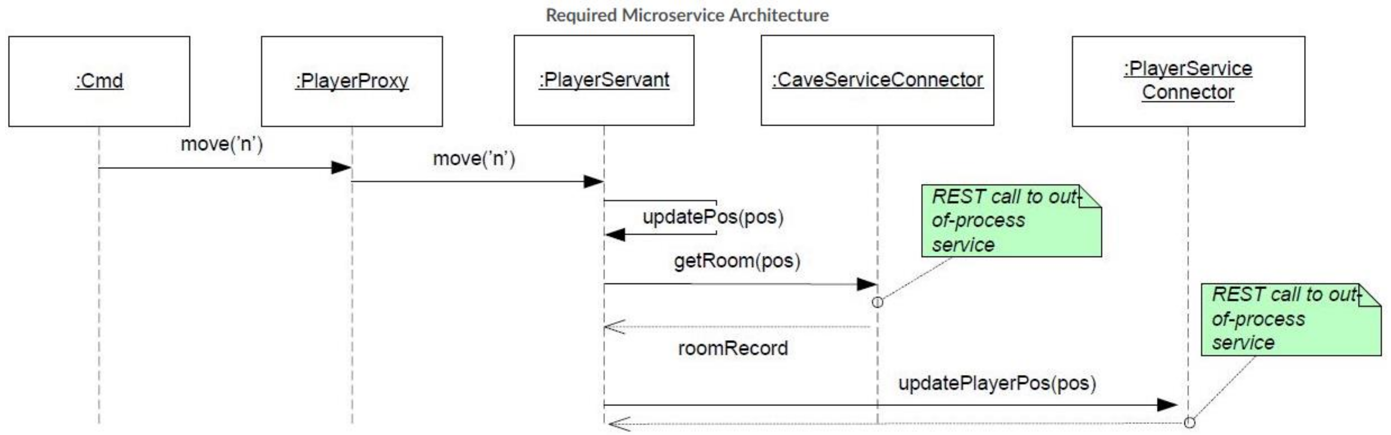
- ... related to Messages on the wall
  - CaveStorage: addMessage, updateMessage, getMessageList

MessageService

So – The Exercise is…

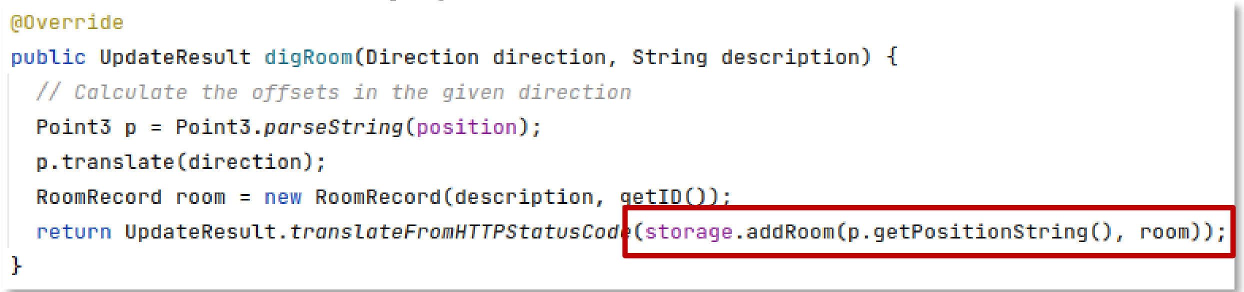
- To modernize/migrate SkyCave to a µService architecture…
- From…
- To…





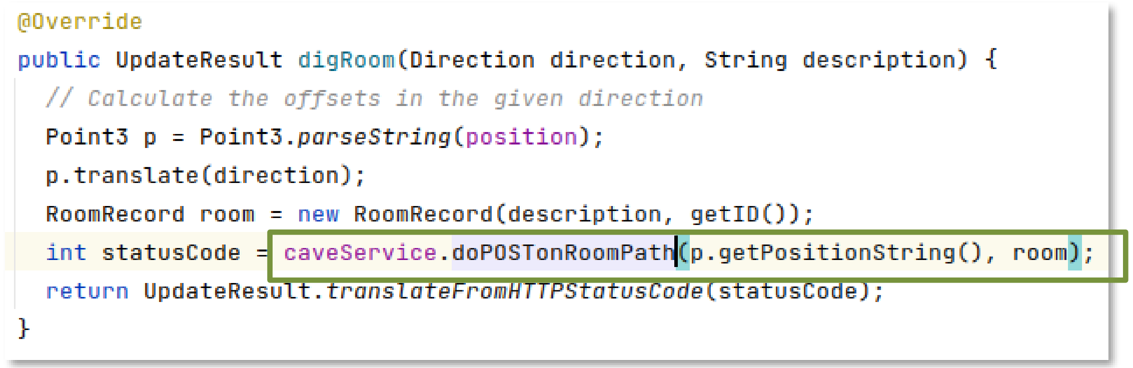
NewPlayerServant

- Presently PlayerServant simply interact with CaveStorage



```
@Override
public UpdateResult digRoom(Direction direction, String description) {
  // Calculate the offsets in the given direction
  Point3 p = Point3.parseString(position);
  p.translate(direction);
  RoomRecord room = new RoomRecord(description, getID());
  return UpdateResult.translateFromHTTPStatusCode(storage.addRoom(p.getPositionString(), room));
}
```

- Refactoring PlayerServant into 'API Gateway' kind of abstraction
  - Strangler pattern



```
@Override
public UpdateResult digRoom(Direction direction, String description) {
  // Calculate the offsets in the given direction
  Point3 p = Point3.parseString(position);
  p.translate(direction);
  RoomRecord room = new RoomRecord(description, getID());
  int statusCode = caveService.doPOSTonRoomPath(p.getPositionString(), room);
  return UpdateResult.translateFromHTTPStatusCode(statusCode);
}
```

AARHUS UNIVERSITET

Which boils down to

… a number of parts

DevOps…

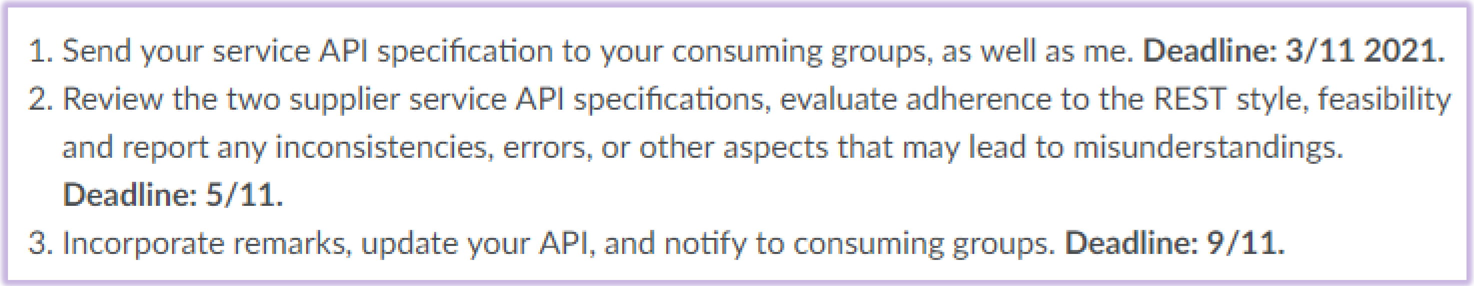
- You do not do it alone
  - Three groups collaborate
    - A produces CaveService and associated CDT
    - B produces MessageService + CDT
    - C produces PlayerService + CDT
  - And all validate the API protocol by review
  - And then you strangle the Player by consuming the three micro services…
    - Creating connectors to Cave-, Message- and PlayerService…
    - And refactor the PlayerImplemention…

The Parts

- *Each* group will solve
  1. *Create and Document a REST API* for your assigned MicroService; and incorporate improvements and suggestions from your consuming groups.
  2. *Develop Contract Tests (CDTs)* in Java/JUnit/TestContainers that verify and document the developed REST API - and provide your consuming groups with these CDTs.
  3. *Develop* your assigned micro service (either Cave-, Message-, or PlayerService), and provide it as a docker hub image to your consuming groups.
  4. Strangle the SkyCave daemon, so a new implementation of the Player interface (a "StrangledPlayerServant") becomes an API Gateway that interacts with Cave-, Player-, and MessageServices (your own + two consumed services; and notably the CaveStorage interface and implementations are eliminated. This will also entail developing *Connector Tests* for the connectors to the three services.
  5. *Deploy to production.* Crunch will come by…

Create/Document API

- To get going, I have defined a milestone plan with dates



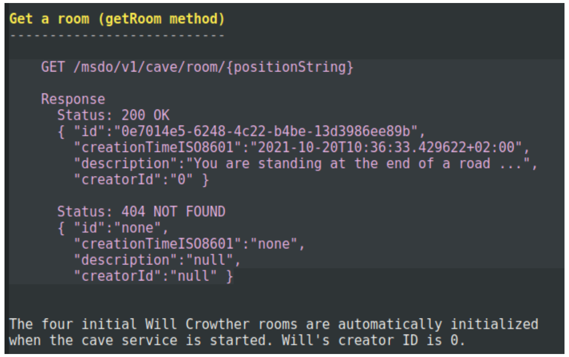
1. Send your service API specification to your consuming groups, as well as me. **Deadline: 3/11 2021.**
2. Review the two supplier service API specifications, evaluate adherence to the REST style, feasibility and report any inconsistencies, errors, or other aspects that may lead to misunderstandings. **Deadline: 5/11.**
3. Incorporate remarks, update your API, and notify to consuming groups. **Deadline: 9/11.**

- If you have problems with the dates, please talk to each other (and me) about finding new dates…

- **APIs are required to be REST format**
  - Use POST and GET, do not just do URI Tunneling

Format

- **API specification in ‘FRDS §7.7’ format** unless all three groups decides otherwise (OpenAPI or what have you).
  - Not a precise format but…

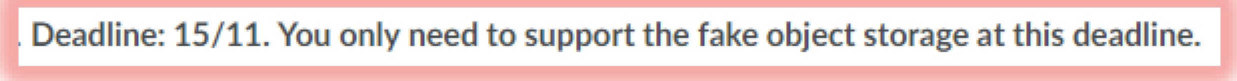


```
Get a room (getRoom method)
---------------------------

    GET /msdo/v1/cave/room/{positionString}

    Response
      Status: 200 OK
      { "id":"0e7014e5-6248-4c22-b4be-13d3986ee89b",
        "creationTimeISO8601":"2021-10-20T10:36:33.429622+02:00",
        "description":"You are standing at the end of a road ...",
        "creatorId":"0" }

      Status: 404 NOT FOUND
      { "id":"none",
        "creationTimeISO8601":"none",
        "description":"null",
        "creatorId":"null" }


The four initial Will Crowther rooms are automatically initialized
when the cave service is started. Will's creator ID is 0.
```

- I do not want to force yet another learning curve on you
  - If all agree you can do as you please, but only if all agree…

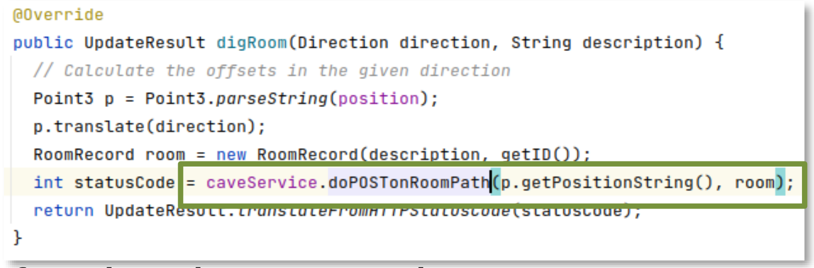
CDT and Service Development

- Develop CDT
  - In Java and TestContainers !
    - And provide that to your consuming groups
      - Git repo, or source files + gradle dependencies
- Develop the Service itself
  - In any toolstack you wish
  - Provide a docker image for it, to consuming groups (+me)
    - Code and docker image may be *public*
  - ***Must be configurable*** to allow either
    - FakeObject- or NoSQL persistence
      - Env variables, command line argument, CPF file, what-have-you


Deadline: 15/11. You only need to support the fake object storage at this deadline.

Connector Tests

- Do *not* issue Http requests directly in you new PlayerServant
  - *Program to an Interface !*



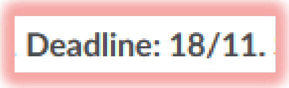
```
@Override
public UpdateResult digRoom(Direction direction, String description) {
  // Calculate the offsets in the given direction
  Point3 p = Point3.parseString(position);
  p.translate(direction);
  RoomRecord room = new RoomRecord(description, getID());
  int statusCode = caveService.doPOSTonRoomPath(p.getPositionString(), room);
  return UpdateResult.translateFromHTTPStatusCode(statusCode);
}
```

- *So*
  - Develop XServiceConnectors for the three services
    - (Interface (+ FakeObject) + Real REST connector)
      - *The interface for each is almost given by cutting CaveStorage into three…*
  - *Develop out-of-process Integration tests (connector tests)*

- Hint: By making a FakeObject implementation, you can start the Strangling even before you get the external services!

Finalize Service Persistence

- Develop the persistence layer
  - Using any NoSQL of your choose
    - Redis, MongoDB, Cassandra, Riak, …

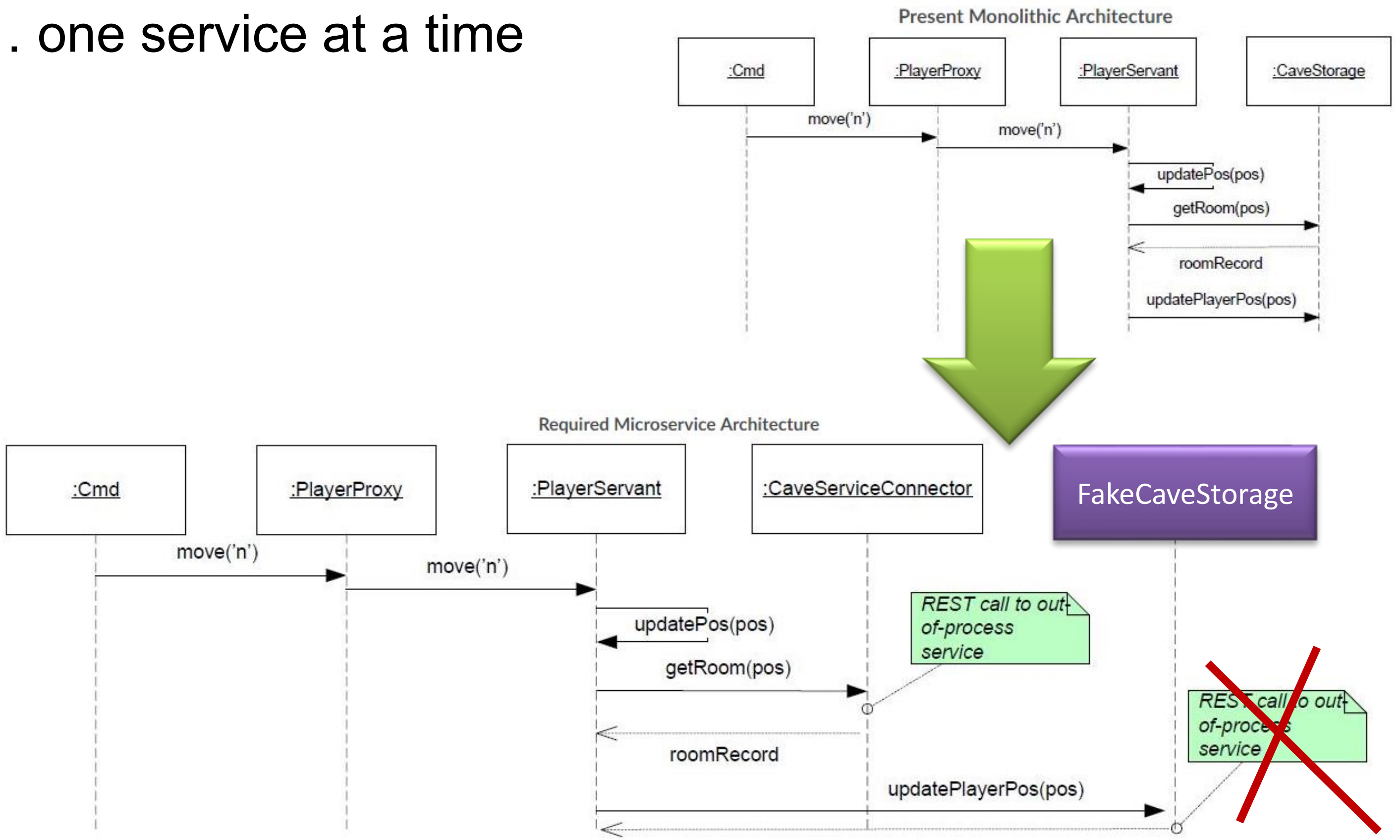

Deadline: 18/11.

- Meaning real deployment requires a compose-file

- *Optional for 1-person groups*

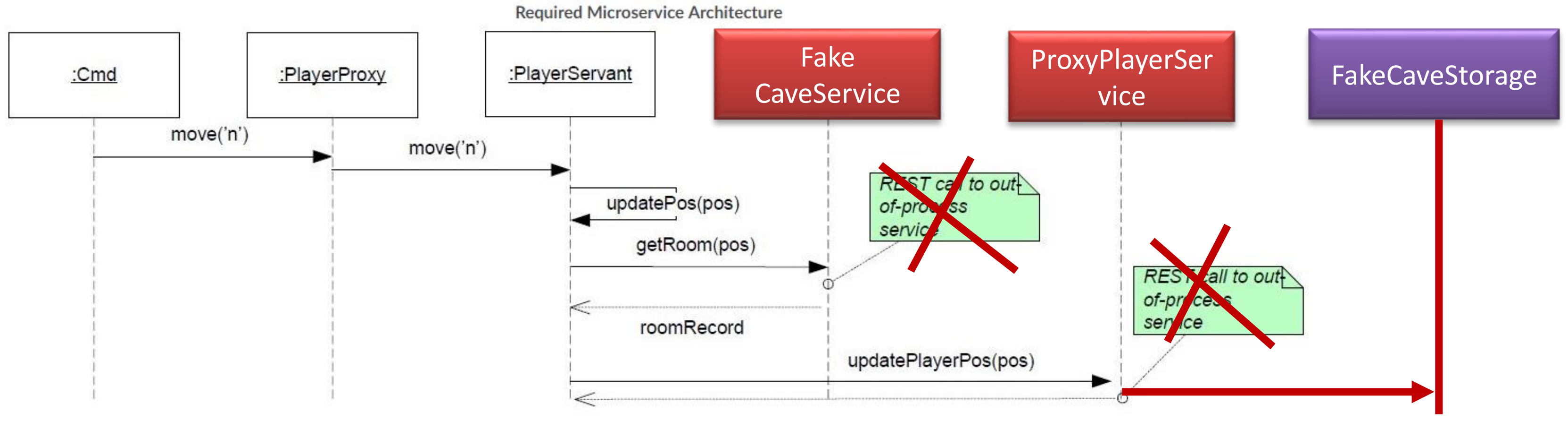
Strangling

- Do a step-wise strangling of the Player role
  - … one service at a time



Program To Interface…

- Note: These individual parts/steps can be done in (almost) any order!
  - You can start Strangling, before you have the service API and images!



Deliver…

- Develop a compose-file for the full SkyCave daemon stack
- … and deploy it to your prod server…
  - Free to forget old data, migrating data is hell-on-earth…
- Two options actually…
  - Your stack deploys all services
  - Your stack refers to external group's deployed services
    - Will have an interesting implication on what users see ☺

AARHUS UNIVERSITET

Hints and Help

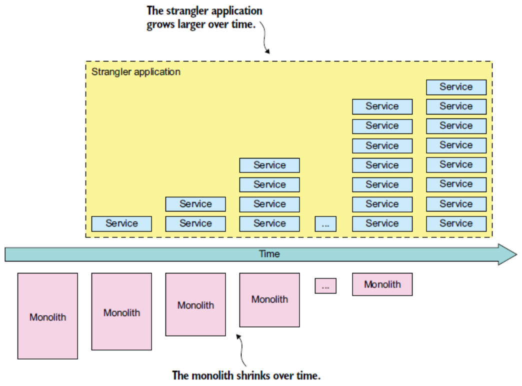
AARHUS UNIVERSITET

Experience from Last Time

- The Connector + Strangling part is not ‘difficult’ but requires some time
  - Last time, people suddenly got very busy 4 days before deadline…
- Take care while Strangling
  - It is easy to ‘start in 10 ends and end up in *big-ball-of-mud’*
    - *“I introduce the message connector in my player servant which becomes a slow out-of-process test, while fiddling with the CPF and having to reset the Redis every, oh, and I also need to catch this UniRest Exception which…”*

PlayerServant Strangling

- Strangler pattern
  - *Present:* All 'micro-roles' are in 'CaveStorage'
  - *Strangling step 1:*
    - Replace 'room' calls with related
      - *caveservice.doThatRESTCall(….);*
    - … while keeing the *FakeCaveStorage* to handle the two other 'micro-roles's responsibilities….
  - *Strangling step 2:*
    - Replace micro-role 2
    - Etc.

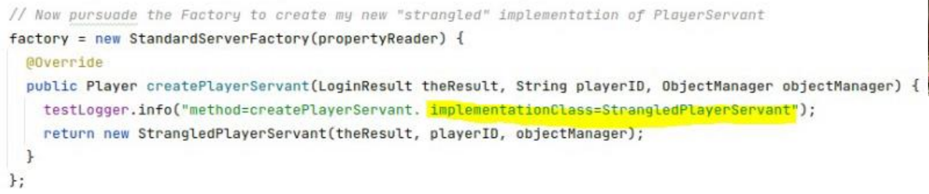

The strangler application grows larger over time.
Strangler application
Service
...
Time
Monolith
...
The monolith shrinks over time.

Strangle your own service first!

Scaffolding

- Do a bit of extra work to keep tests running all the time!
  - Use intermediaries, keep stuff that will eventually disappear…

- If you ‘fall into the abyss’, consult my own first steps of strangling…

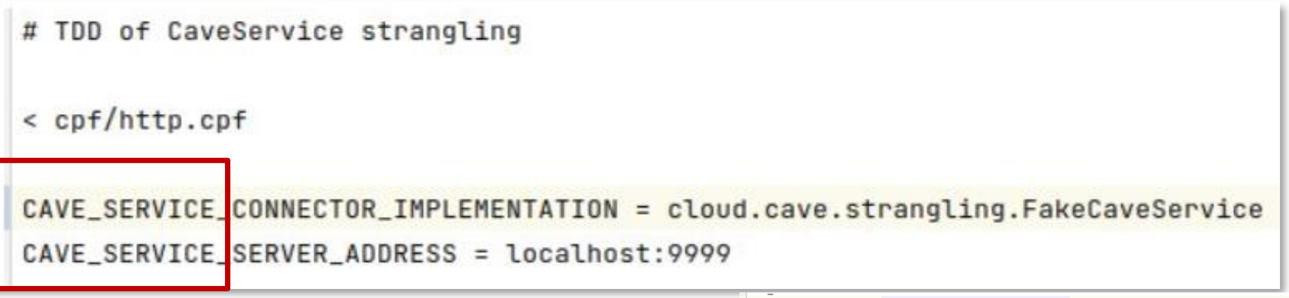


```
// Now pursuade the Factory to create my new "strangled" implementation of PlayerServant
factory = new StandardServerFactory(propertyReader) {
  @Override
  public Player createPlayerServant(LoginResult theResult, String playerID, ObjectManager objectManager) {
    testLogger.info("method=createPlayerServant.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

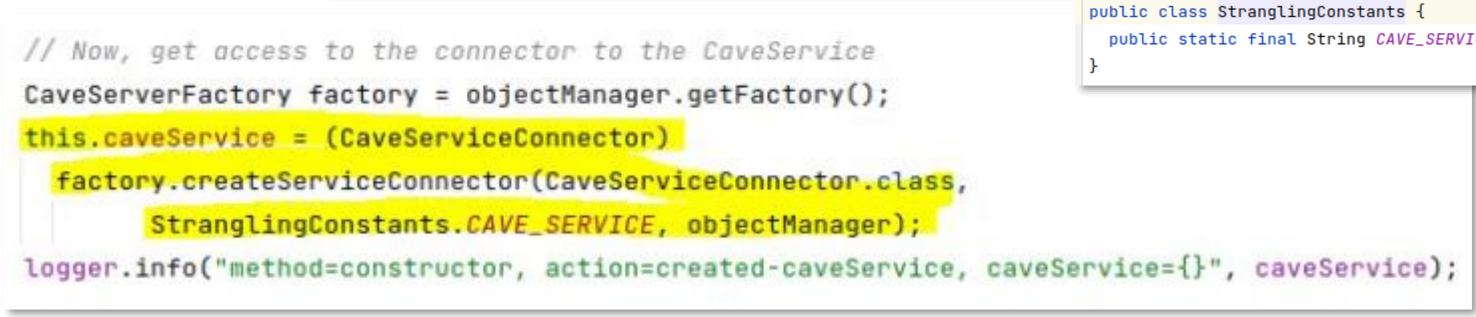
- *Find a link to a document in the exercise hint section…*

Configuring new services

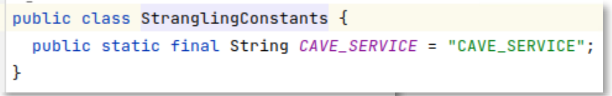
- SkyCave's Factory system can read *any* service specification, not just the QuoteService etc.
  - Obey the 'naming' convention, with a prefix



```
# TDD of CaveService strangling

< cpf/http.cpf

CAVE_SERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.strangling.FakeCaveService
CAVE_SERVICE_SERVER_ADDRESS = localhost:9999
```



```
// 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);
```

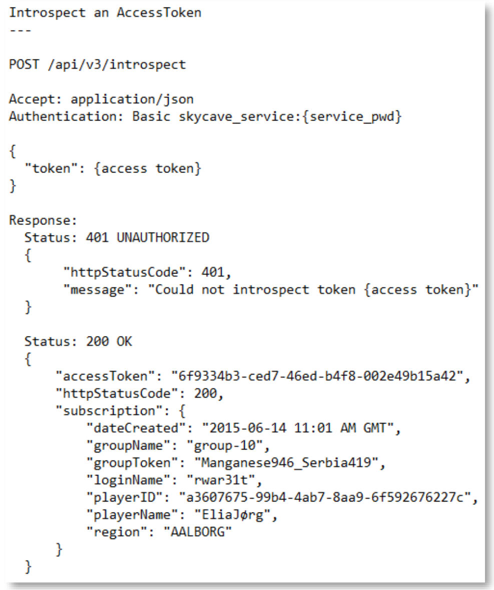


```
public class StranglingConstants {
  public static final String CAVE_SERVICE = "CAVE_SERVICE";
}
```

AARHUS UNIVERSITET

Security

- You *could* ensure only authenticated SkyCave players are allowed to access your group’s service…
- By
  - Requiring ‘Authorization: Bearer (accessToken)’ header in the request…
  - Do /introspect on cavereg.baerbak.com
- But…
  - Uhum, do not…
    - The exercise is big enough as it stands.



```
Introspect an AccessToken
---

POST /api/v3/introspect

Accept: application/json
Authentication: Basic skycave_service:{service_pwd}

{
  "token": {access token}
}

Response:
  Status: 401 UNAUTHORIZED
  {
       "httpStatusCode": 401,
       "message": "Could not introspect token {access token}"
  }

  Status: 200 OK
  {
      "accessToken": "6f9334b3-ced7-46ed-b4f8-002e49b15a42",
      "httpStatusCode": 200,
      "subscription": {
          "dateCreated": "2015-06-14 11:01 AM GMT",
          "groupName": "group-10",
          "groupToken": "Manganese946_Serbia419",
          "loginName": "rwar31t",
          "playerID": "a3607675-99b4-4ab7-8aa9-6f592676227c",
          "playerName": "EliaJørg",
          "region": "AALBORG"
      }
  }
```