Mandatory Project: SkyCave Microservice Architecture

<<Names>>
Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark
<<Group name>>
<<Student names>>
{..., ...}@cs.au.dk

<<Date>>

Contents

Abstract

The SkyCave System implements a massive multiuser online experience. This report outlines the work done by the group to solve the course's three mandatory exercises.

Chapter 1

From Monolith to Microservice

This section represents the group's solution to the first mandatory exercise.

1.1 Service Assignment

[Present the service assignment: Which microservice is your group responsible for? Which two groups supply the other services?]

1.2 Service APIs

Below the final service APIs are documented using the notation of [Either choose FRDS Â§7.7 [Christensen, 2019] or the OpenAPI initiative].

[You only need to fully fill in the section of your group's service, for the other two services, just refer to the supplier group's name.]

1.2.1 PlayerService

1.2.2 CaveService

1.2.3 MessageService

1.3 Consumer-Driven Tests (Faked Storage)

[Shortly explain your CDTs for your developed REST service, by focusing on an "interesting, and not too complex" test. Remember to include the test itself. Perhaps use 'Given-When-Then' comments or other guides for the reader.]

1.4 Reflections on DevOps

[Shortly outline experiences in the DevOps Process - exchanging APIs, exchanging CDTs, collaborations with other groups, ...]

1.5 Integration (Connector) Tests

[Shortly explain your Integration tests for your developed driver/connector to your REST service, by focusing on an "interesting, and not too complex" test. Remember to include the test itself. Perhaps use 'Given-When-Then' comments or other guides for the reader.]

1.6 Service with Data Layer

[This exercise is optional for one-person groups.]

[Include compose-file for your full service, including whatever NoSQL database you have choosen for the job.]

1.7 Strangling Process

[Outline your strangling process—how SkyCave was migrated into replacing a central CaveStorage storage tier with the three services. Include source code from one of the methods in the final, refactored, PlayerServant, and explain it shortly.]

1.8 Operations

[Include your swarm compose-file]

[Include screen snapshots that document that your swarm is fully operational, and can be operated by a 'cmd']

1.9 Artefacts

The artefacts developed can be found here:

- (Docker hub SkyCave image, including source code)
- (Docker hub REST service image)
- (Link to REST service code base (bitbucket, zip, whatever))
- (Link to CDT's, if not included in the above link)

- (potential other developed artefacts)

Chapter 2

Design for Failure

2.1 Design for Failure

[Pick the most complex integration point to your group's REST service, and provide a short argumentation for why this integration point was choosen (detailing it in the sections below)]

2.1.1 Safe failure mode

[Argumentation for the choosen safe failure mode behavior: 'try later' reply, cached reply, eventually consistent reply, retries, or . . .]

2.1.2 Code fragments

[Include the safe failure mode/graceful degradation code fragment(s) and shortly explain the code]

2.1.3 Demo

[Include a set of screenshots (avoid font sizes that are too small!) or (perhaps better) a link to a screencast of 4-8 minutes, that demonstrate a scenario in which the REST service is working correctly (Cmd doing the happy path stuff), a failure in the REST service is introduced, and next demonstrated how the safe failure mode works (Cmd being gracefully degradated, server log output, . . .)]

[Screencast = video with voice over explaining what is happening]

2.2 Health Check

[Include the compose-file and highlight and explain the health check and the restart policy for your group's service]

[Include the /health path handling code in the REST service, and shortly explain]

2.3 Reflections on "Design for Failure"

[Shortly outline the group's experiences with increasing availability/stability of SkyCave using the techniques above. Examples of issues may be effort invested, particular difficulties, trade-offs, relations to theory, etc.]

Chapter 3

Microservice Architecture Outlook

3.1 Problem statement/Hypothesis One

[Make a short problem statement or hypothesis: what do you want to investigate and what is the expected outcome? Try as best possible to make the statement as measurable as possible.]

3.1.1 Method

[Outline shortly how you want to conduct the experiment, how to measure/reason that the hypothesis is true, how to verify that the expected outcome is achieved, what systematics you use to ensure the results is not due to something (false positives)]

3.1.2 Experiments

[Describe experiments and include experimental output: refactored code, compose files, graphs of measurements, architectural diagrams, whatever suits your problem statement and experiments]

3.1.3 Results and Conclusion

[Present results and tie the conclusion to the initial problem statement. What did you learn?]

3.1.4 Reflections on the Process

[Reflect on the exercise/process of solving the exercise.]

3.2 Problem statement/Hypothesis Two

[Repeat above template for second issue investigated.]

Bibliography

[Christensen, 2019] Christensen, H. B. (2019). *Flexible, Reliable, Distributed Software—Still Using Patterns and Agile Development*. LeanPub.com.