Testing Microservices

Techniques for Automation

European Testing Conference 2018
Emily Bache
@emilybache
Emily Bache

Practice Lead for Test Automation
Consultant
Author of
“The Coding Dojo Handbook”

@emilybache
emily.bache@praqma.com
Pagero Online Cloud Service

Invoice Issuer

Invoice Recipient

XML

PDF
Some numbers…

• We were around 45 developers, in 7 teams
• We had around 60 microservices
• Our monolith was in production for nearly 10 years before we started with microservices, 3-4 years ago
Pagero Online’s architecture
A Microservice

- Provides one business capability
- Independently Deployable
- Lightweight API
- Not too big: fits in my head

*Dockert is used to containerise the service*
Testing a Microservices Architecture

In my experience
End-To-End tests exercise the whole system

Unit tests are for individual functions

https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid
Testing Strategy for the Monolith

Unit tests and end-to-end GUI tests still valid for Microservices architecture
Testing Strategy

Testing a monolith
- Unit tests
- Data-layer tests
- Service-layer tests
- GUI tests

Testing a bunch of microservices
- Unit tests
- Individual microservice tests
- System tests
- GUI tests
Testing a Microservice
Microservice Tests

• Deploy your service on localhost (and the database if it has one)

• Test uses the public API to trigger functionality

• Mock a response to any calls made to other services or queues
Faster execution

• In-process testing: Deploy your service with an in-memory database and in-memory api
• ‘internal api’ to set up test data and query internals
More production-like

- Out-of-process testing: Deploy your service in one process, and the test case in another
- Security needed for ‘internal api’ so only test/non-production code can access
- Easier to performance test
Business-Facing Feature Tests

Test a whole scenario across several microservices
Testing a whole Feature

This is where it starts getting really complicated…
Example End-to-End Test Strategy

• Test crucial workflows, not every detail
• Use APIs instead of the GUI for most tests
• Techniques:
  • Selective Deployment
  • Approval Testing
  • Event Monitoring

Keep testing costs as low as possible
Service-layer tests in a monolith

- Test cases are for different business-facing workflows
- Will exercise different components within the monolith
Feature tests in microservices

- Test cases are for different business-facing workflows
- Will exercise different groups of microservices
- Need not deploy everything for every test case
Selective Deployment

- Need not deploy everything for every test case
- But: all test cases needed a few essential infrastructure services.
Parts of a test case

(Aprroximately) all the tests involve these elements:

- Invoice Issuer
- Document
- Invoice Recipient
- Test Workflow
- Recipient Presentation

Arrange -> Act -> Assert
Approval Testing specifies the Assert: compare against an ‘Approved’ result.
The “Approved” Result as Text

- Find the important outputs. Convert them to plain text.
- Use textual diff to decide if the actual output matches the approved version.
Elements of a Test Case

- Minimise test creation work
- Minimise test maintenance work
- Maximise serendipity - find bugs you didn’t anticipate
Debugging a failing test?

Which part broke?
Correlation ID

Pick out all events with ‘123’
Event log recorded in a test case

Helps you to debug what happened when the test fails
Test Case Elements

- Recorded traffic is stored with the approved output

Input data varies by test case

Invoice Issuer  Document  Invoice Recipient

Test Workflow common to many tests

Event Logs to debug the test

Approved Output to determine pass/fail

Recipient Presentation for reference
Independent Microservices & Teams

Organizing Testing efforts
Testing in the pipeline

- Pair Programming, TDD
- version control
- unit test
- microservice test
- system test
- manual test
- production

- more production-like environment
- test larger pieces of code together
- Test costs increase
Multi-team development

Component A

Infrastructure & Architecture

Components C&D

Component B

Components E&F
Testing in the pipeline

Pair Programming, TDD

version control

unit test

component test

system test

manual test

production

monitoring
These tests are often broken!
Team pipelines

One environment shared by all teams

version control → unit test → micro-service test → team-system test

system test → manual test

production

monitoring
Teams Deploy Independently

Team A
- version control
- microservice test
- team-system test
- team-manual test

Team B

incremental roll-out to production
monitoring & manual testing
Testing in a Microservices Architecture

System Tests

- Invoice Issuer
- Document
- Invoice Recipient
- Test Workflow
- Approved Output
- Event Logs

Input data varies by test case
- Common to many tests
- To determine pass/fail
- To debug the test
Testing in a Microservices Architecture

System Tests

- Invoice Issuer
- Document
- Invoice Recipient
- Test Workflow
- Approved Output
- Event Logs

Input data varies by test case

common to many tests
to determine pass/fail
to debug the test