TESTING ASYNCHRONOUS APIS

Presented by Heinz N. Gies
Agenda

• The system under test – what is Project-FiFo
• The Problem
• Building the model
• Conclusion
What is Project-FiFo

- Cloud Orchestration (think Open Stack)
- Manage private and public clouds
- Based on SmartOS / Solaris Containers (aka Zones)
- OSS and Commercial Support
What is Project-FiFo

- Self hosted
- Distributed
- Highly available
- Eventually consistent
Architecture

HTTP / REST API

AAA: OAuth2, RBAC

Business logic, database, tracking

Agents to manage physical components
Testing

• Testing components and libraries in isolation
• Unit tests
• EQC properties
• riak_test (testing for distributed components)
Let's test
Lets test

The problem

Login → Create

API
AAA
Mgmt.
Agent
Agent
Agent
Let's test

The problem

Login → Create → GET

API
AAA
Mgmt.
Agent
Agent
Agent
Let's test

Login → Create → GET

This fails, create is asynchronous
Let's test

The problem

OK

fail

Login → Create → 3m → GET → API → AAA → Mgmt. → Agent → Agent → Agent
The problem

- Tests in isolation
- Limited interaction
- OS, hardware, network are part of the system
- We don’t know when a command finishes
Create a VM

Building the model

- Login
- Create
- 1s
- starting
- start
- stopped
- GET
- busy
- running
- fail
- stop
- stopping
- 1s
- GET
- busy

API
AAA
Mgmt.
Agent
Agent
Agent
Wait for a VM to be running
Stop a VM
Wait for a VM to stop
Starts VM

Building the model
Building the model
States

- VM’s powering up
- Running VM’s
- VM’s that are being stopped
- Stopped VM’s
Transitions

- Starting
  - Started -> running
  - Timeout -> failed
  - Still starting -> starting

Building the model
transitions

- Stopping
  - stopped -> stopped
  - Timeout -> failed
  - Still stopping -> stopping
What do we get out of it

• Concurrent creations
• More transitions
• Concurrency of multiple users
• High test density
  • 458 LOC
  • Including the described model
  • VM deletion
  • Permission validation (user A doesn’t see / access user B’s VMs)
Testing the full stack, upside

- Interactions between components
- More complex error cases can be caught
- Observe behavior in real conditions
Testing the full stack, downside

- More complex test cases
- Need to understand the system well
- Need really good logging
- Erlang observability is a big help
- Long running tests (hours not seconds)
What is next

- Extending the model
- Testing for failures
- Test automation
- Test coverage
Beyond development

- **Deployment testing**
  - After initial installation.
  - Validates functionality.
  - Part of customer handoff.

- **Continuous testing**
  - System build for concurrent use/load.
  - Never ending tests.
  - Constant validation of functionality.
  - Part of alerting/monitoring strategy.
  - Possibility to get very complex test cases.