accenture technology

Dynamic Languages

Scalable Dynamic Lightweight Service Layer using CoffeeScript

Strategy | Consulting | Digital | Technology | Operations

Me

Ilkka Anttonen

Technology Architecture Science Manager
Emerging Technologies Nordic Lead
Senior Technology Architect

Blog: http://sirile.github.io
Mail: ilkka.anttonen@accenture.com
Twitter: IlkkaAnttonen

Agenda

Modern (Lightweight) Runtime Architectures

Dynamic vs Static

Developer Happiness

Case: Returning JSON over REST for complex database structure

- Chosen tools and libraries
- Project Structure and dependency management
- Configurations for different environments
- Model with Sequelize
- REST service using Restify
- Local TDD with file watching
- Test coverage
- Packaging to container and running
- Full build and tests

Conclusion and Q&A

Modern (Lightweight) Runtime Architectures

- Polyglot architectures are becoming a real option
  - Modern runtimes allow for better contract based boundaries between components
  - As services can be self-contained there are less (technical) dependencies between services
- Use the correct tool for the correct problem, for example
  - For rapid prototyping dynamic languages may be a good fit
  - For stream handling, Scala might be a good bet
  - For maximum throughput go or c(++) might fit
  - Depending on available skills, Java and SpringBoot could work

Dynamic vs Static

- Arguments can go whichever way
  - Both ideologies have their merits and they aren't mutually exclusive
- Prototyping and exploring new ideas is often faster with dynamic languages
  - As things are (often) less strict, the team needs to have necessary discipline and experience to keep everything organized and understandable
- Less code means inherently less bugs introduced
- Smaller amount of code is easier to refactor and/or replace
  - If the solution is clean enough, almost anyone can understand it regardless of previous experience with the given language instead of having to learn a huge boilerplate containing framework around the solution

Developer Happiness

- What makes me a happy developer
  - (Extremely) fast feedback cycle
  - Elegant and simple solutions
  - Ability to pick the tools that fit my development style
  - Ability to solve the problem without boilerplate (convention over configuration)
  - Visibility on what's happening
- A happy developer is a productive (and creative.. within limits) developer!

CASE: Returning JSON over REST for complex database structure

Requirements

- Needs to support a model with complex relationships
  - Associations and different mappings
- Backed by a SQL database
  - MariaDB was a good candidate
- End-to-end tests
  - Integration through the service API to the DB
- Publish as a self-contained service (Docker)
  - Inherent support for scaling

Chosen Tools and Libraries

- CoffeeScript
  - Nicer syntax than pure JS, personal preference
- Restify
  - Powerful REST library
- Sequelize
  - Practically a full blown ORM implementation on NodeJS
- Mocha + Chai (+ Sinon) + Istanbul
  - Simple and effective tests + coverage, mocking (Sinon) not used for now
- Bunyan
  - JSON logging
- Npm
  - Building and package management
- Docker

Project Structure and Dependency Management

- Npm, Sequelize and Mocha drive towards unified structure
  - Even dynamic languages are getting opinionated frameworks that are becoming widely accepted – “industry standard”
- Installation is `git clone` followed with `npm install`
  - This installs all the required dependencies with working versions
  - **Note:** This has to be run in both the root and in the service/users-directory
- Local running with `npm run db` followed by `npm start`
  - Everything is controlled through a single tool
- (Demo of project structure and running)

Configurations for Different Environments

- Support for development (local shell), test (Docker) and production
  - Localhost means a different thing in Docker and when running on shell
  - The convention for this project is set by Sequelize
- Environment is set through environment variables (NODE_ENV)
  - They are exposed to the nodejs application through a standard mechanism
  - Works well with Docker based runtime that supports environment variables
- All configuration should really be externalized to for example Consul
  - Allows for really runtime agnostic containers
  - Not in scope of this exercise, but I'm writing a library to help

Model with Sequelize

- Sequelize is a promise based ORM for nodejs
  - Support for PostgreSQL, MySQL, MariaDB, SQLite and MSSQL
- Good support for different datatypes
  - General as well as database dialect specific
- Supports both lazy and eager evaluations
  - Defined at query execution time
- Good model validation support
  - The contents of the model can be validated as part of operations or by calling a validate() method
  - Custom validators can be defined for complex cases and cross validations

REST Service using Restify

- "Framework that gives control over interactions with HTTP and observability into the latency and characteristics of applications"
  - Borrows heavily from Express but is intended for just API services
  - Native support for DTrace for low level latency information
- Supports plugins and handlers
  - Body parsing, static file serving, auditing, ..
- Efficient, simple and lightweight
  - Configurable and extendable in case default functionality isn't enough

Local TDD with file watching

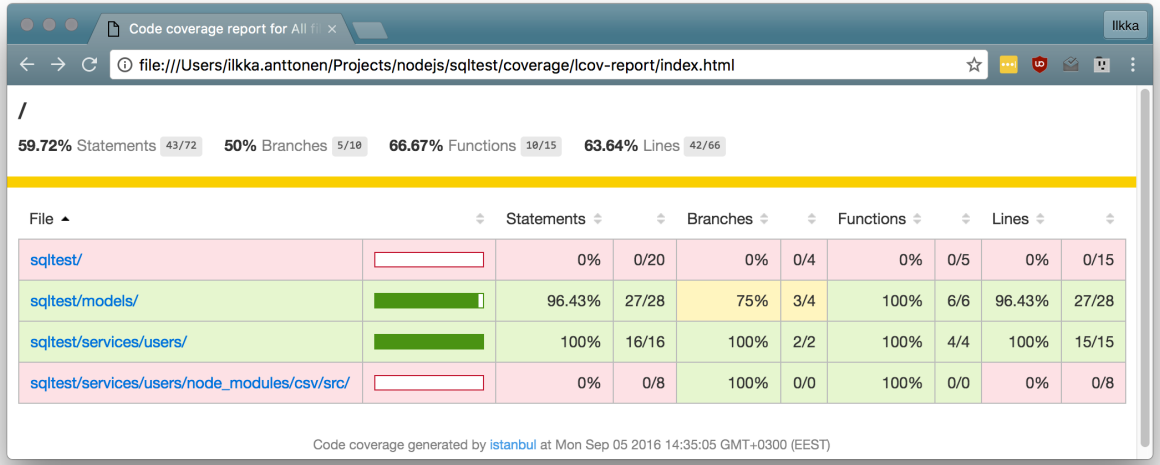
- Tests are run every time files change with `mocha --watch`
- Mocha and Chai tests are readable

```
describe 'if the user exists', ->
  it 'returns http statuscode 200 and a JSON object for the user', (done) ->
    client.get '/users/ile', (err, req, res, obj) ->
      res.statusCode.should.equal 200
      obj.userid.should.equal 'ile'
      done()
```

- Allows for instant feedback when combined with nodemon providing instantaneous code compilation after save
- Use `npm run testwatch` to run the testing framework
- (Demo of adding a new attribute to the model)

Test Coverage

- Test code coverage reports can be generated with Istanbul
- Running the coverage report is done with `npm test` which also generates the reports that are viewable through the browser



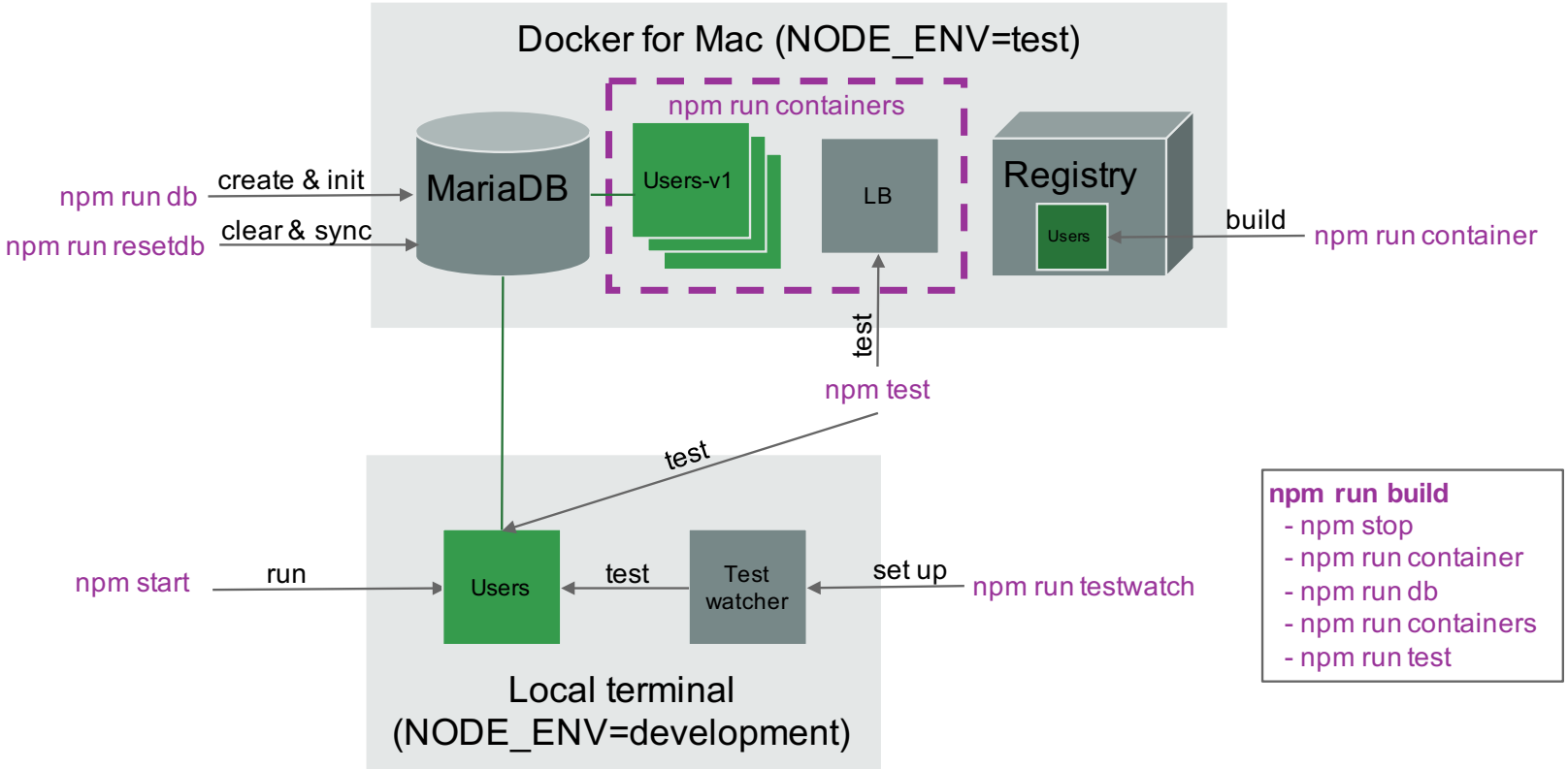
Packaging to Container and Running

- Uses Docker for Mac
  - In Windows Docker for Windows should work as well
  - Docker Machine and VirtualBox based solution should also work
  - Container is built with `npm run container`
- Base image is a minimized Alpine based official NodeJS image
  - Helps to control the size of the service image
- Tests can be run against the container runtime
  - If the environment variable NODE_ENV is TEST, then the target for the tests is the loadbalancer instead of the local port 8080
- The container runtime can be started with npm run containers
  - This also starts the loadbalancer

Full Build and Tests

- Full build with tests is done with `npm run build` which combines
  - `npm stop`
  - `npm run container`
  - `npm run db`
  - `npm run containers`
  - `npm run test`
- After that would come tagging the image and pushing to production registry and notifying the orchestration system that a new version is available, followed by either a rolling upgrade over existing containers or a new version running side-by-side

Bringing it Together



Conclusion & Q&A

Thanks for listening!