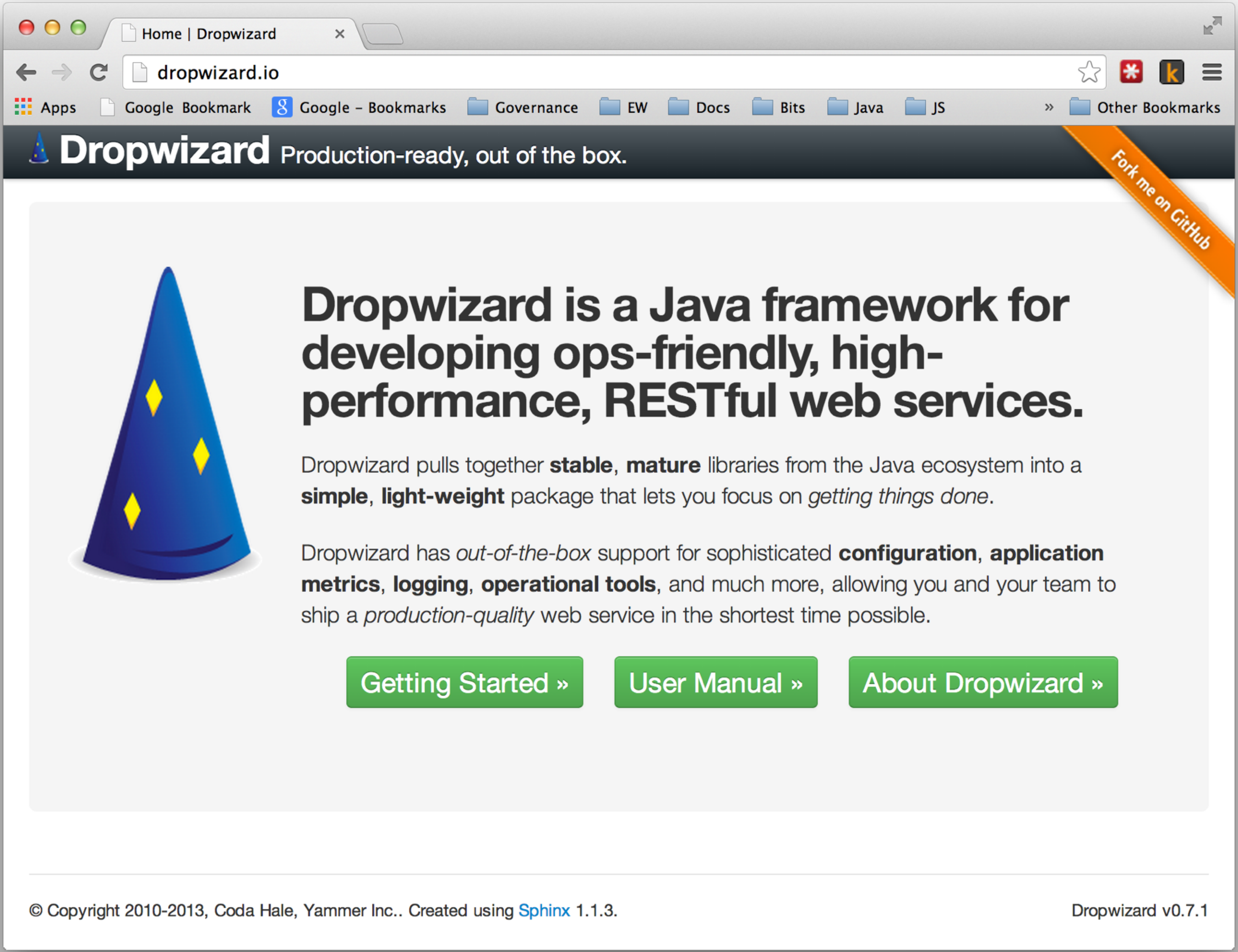
Component Testing with Dropwizard and WireMock

@TomAkehurst



Dropwizard Production-ready, out of the box.

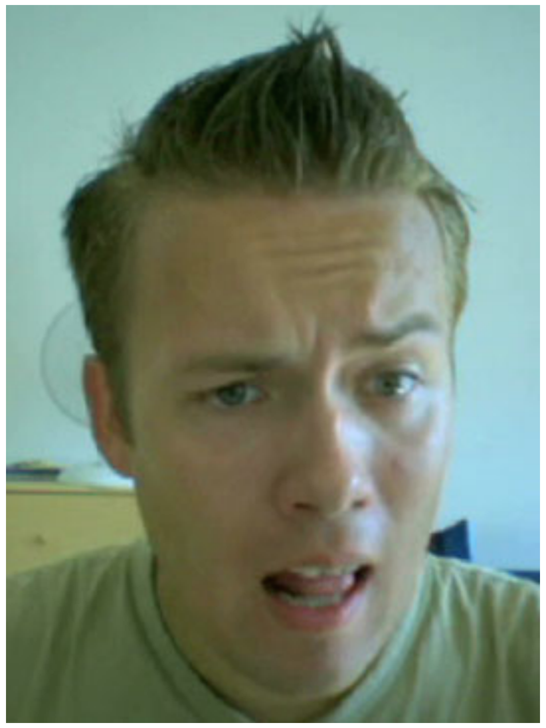
Dropwizard is a Java framework for developing ops-friendly, high-performance, RESTful web services.

Dropwizard pulls together **stable**, **mature** libraries from the Java ecosystem into a **simple**, **light-weight** package that lets you focus on *getting things done*.

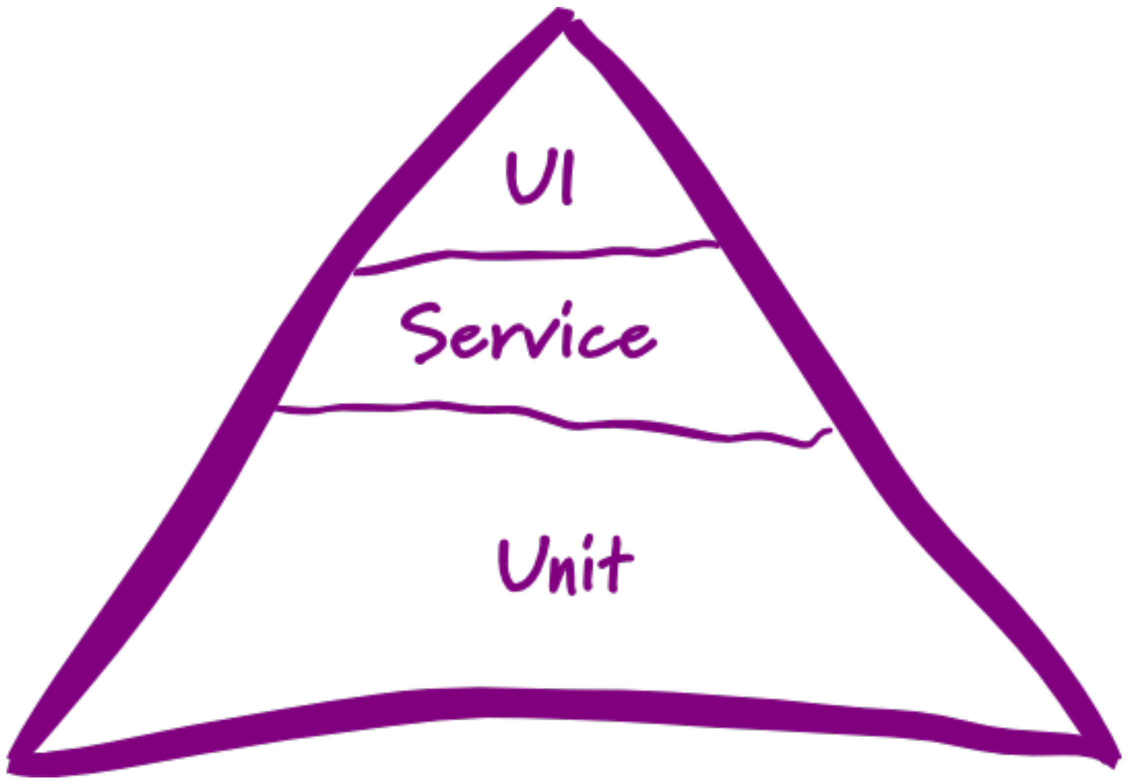
Dropwizard has *out-of-the-box* support for sophisticated **configuration**, **application metrics**, **logging**, **operational tools**, and much more, allowing you and your team to ship a *production-quality* web service in the shortest time possible.

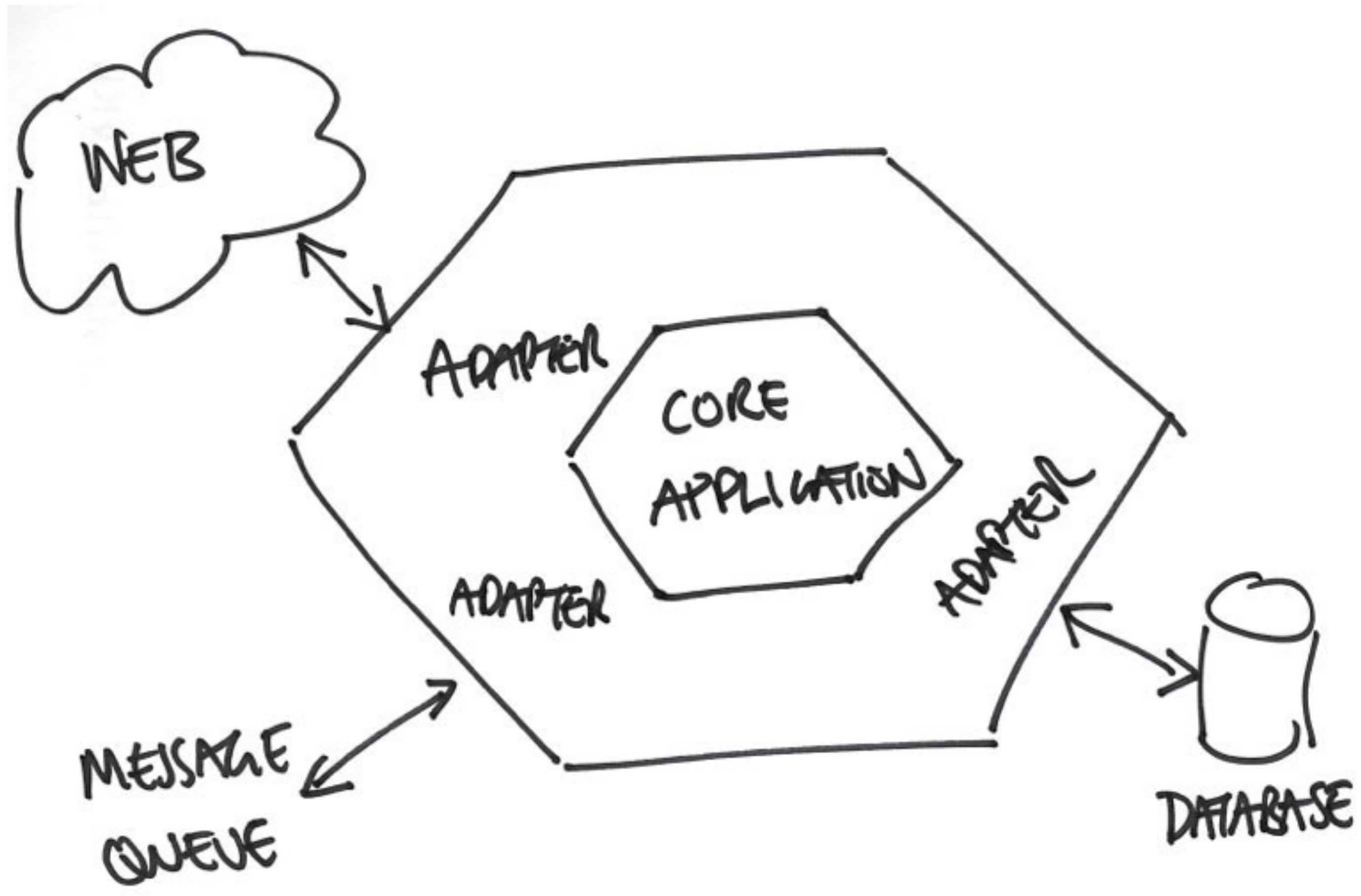
© Copyright 2010-2013, Coda Hale, Yammer Inc.. Created using Sphinx 1.1.3.

Dropwizard v0.7.1



TDD is dead!





* With thanks to Martin Fowler and Matt Wynne, neither of whom are aware I've stolen their images

Context: μ-services

Constrained size

Few & static ports

Simple core domain

API – REST, messaging etc.

Component Testing

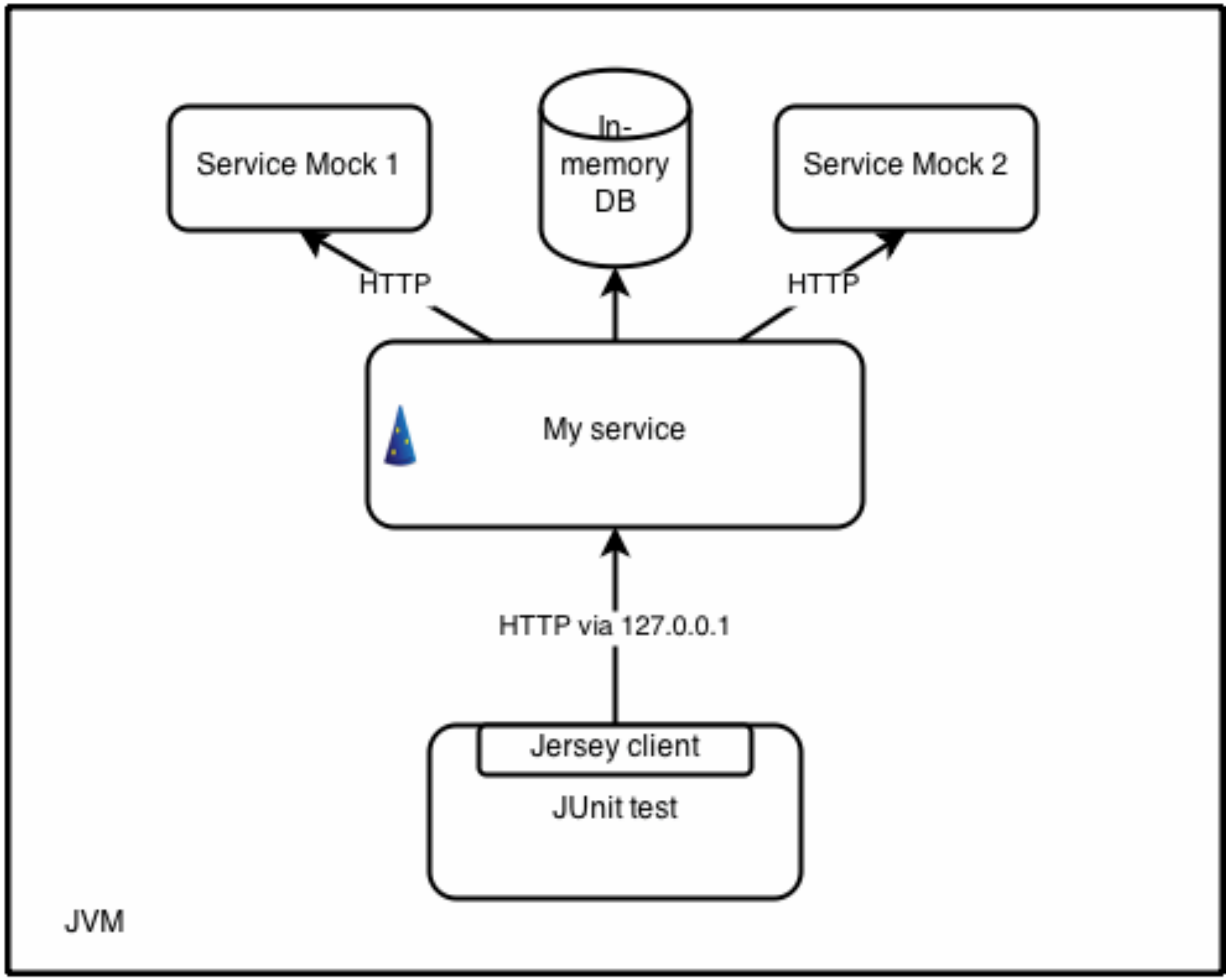
Component Testing

Run the entire app/service, not just bits of it

Mock external services, not objects

No deployment required – everything in one JVM

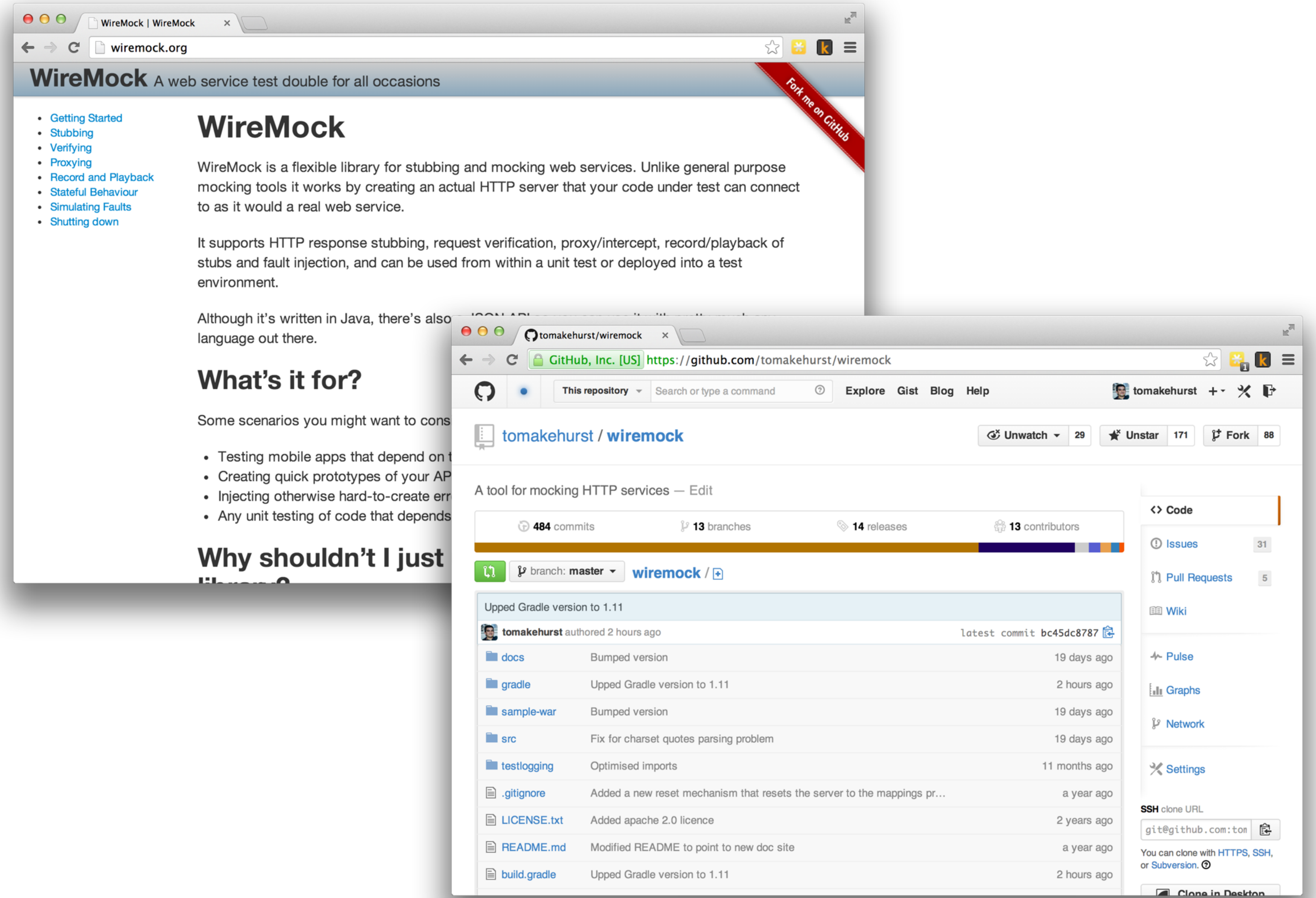
Concerned with component (e.g. REST), rather than object interfaces


Service Mock 1
In-memory DB
Service Mock 2
HTTP
HTTP
My service
HTTP via 127.0.0.1
Jersey client
JUnit test
JVM

```java
public class MyTestEnvironment implements TestRule {

    private final DropwizardAppRule<MyConfig> app =
            new DropwizardAppRule<>(MyApp.class, "test.yaml");

    private final InMemoryH2Rule db = new InMemoryH2Rule();
    private final MockService1Rule service1 = new MockService1Rule();
    private final MockService2Rule service2 = new MockService2Rule();

    private final RuleChain rules =
        RuleChain
                .outerRule(db)
                .around( service1)
                .around( service2)
                .around( app);

    @Override
    public Statement apply(Statement base, Description description) {
        return rules.apply(base, description);
    }

    public DropwizardAppRule<MyConfig> app() {
        return app;
    }

    public Client buildTestClient() {
        return app.getConfiguration().buildTestClient( app.getEnvironment());
    }
}
```

```java
public class AcceptanceTest {

    @Rule
    public MyTestEnvironment testEnvironment = new MyTestEnvironment();
    Client client;

    @Before
    public void init() {
        client = testEnvironment.buildTestClient(); // Build using YAML config
    }

    @Test
    /unchecked/
    public void should_return_3_list_items_initially() throws Exception {
        ClientResponse response =
                client.resource("http://localhost:" + port() + "/items")
                .accept(APPLICATION_JSON)
                .get(ClientResponse.class);

        assertThat(response.getStatus(), is(200));
        with(response.getEntityInputStream())
                .assertThat("$.items", is(collectionWithSize(equalTo(2))));
    }

    private int port() {
        return testEnvironment.app().getLocalPort();
    }
}
```

Service Mocks?



WireMock | WireMock

wiremock.org

WireMock A web service test double for all occasions

Fork me on GitHub

- Getting Started
- Stubbing
- Verifying
- Proxying
- Record and Playback
- Stateful Behaviour
- Simulating Faults
- Shutting down

WireMock

WireMock is a flexible library for stubbing and mocking web services. Unlike general purpose mocking tools it works by creating an actual HTTP server that your code under test can connect to as it would a real web service.

It supports HTTP response stubbing, request verification, proxy/intercept, record/playback of stubs and fault injection, and can be used from within a unit test or deployed into a test environment.

Although it's written in Java, there's also
language out there.

What's it for?

Some scenarios you might want to cons

- Testing mobile apps that depend on t
- Creating quick prototypes of your AP
- Injecting otherwise hard-to-create err
- Any unit testing of code that depends

Why shouldn't I just

tomakehurst/wiremock

GitHub, Inc. [US] https://github.com/tomakehurst/wiremock

This repository Search or type a command Explore Gist Blog Help tomakehurst

tomakehurst / wiremock

Unwatch 29 Unstar 171 Fork 88

A tool for mocking HTTP services — Edit

484 commits 13 branches 14 releases 13 contributors

branch: master wiremock /

Upped Gradle version to 1.11

tomakehurst authored 2 hours ago latest commit bc45dc8787

docs	Bumped version	19 days ago
gradle	Upped Gradle version to 1.11	2 hours ago
sample-war	Bumped version	19 days ago
src	Fix for charset quotes parsing problem	19 days ago
testlogging	Optimised imports	11 months ago
.gitignore	Added a new reset mechanism that resets the server to the mappings pr…	a year ago
LICENSE.txt	Added apache 2.0 licence	2 years ago
README.md	Modified README to point to new doc site	a year ago
build.gradle	Upped Gradle version to 1.11	2 hours ago

SSH clone URL

git@github.com:ton

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

```java
@Rule
public WireMockRule mockThingStore = new WireMockRule(wireMockConfig().port(9090));

@Test
public void should_return_first_item_from_thing_store() throws Exception {

    mockThingStore.stubFor(get(urlEqualTo("/user-items/1"))
            .willReturn(aResponse()
                    .withStatus(200)
                    .withHeader("Content-Type", "application/json")
                    .withBodyFile("user-items-1.json")));

    ClientResponse response =
            client.resource("http://localhost:" + port() + "/items")
                    .accept(APPLICATION_JSON)
                    .get(ClientResponse.class);

    with(response.getEntityInputStream())
            .assertThat("$.items[0].name", is("Item 1"))
            .assertThat("$.items[0].id", is("1168471"));
}
```

```java
@Test
public void should_push_an_item_to_the_thing_store_when_added_by_user() throws Exception {

    mockThingStore.stubFor(post(urlEqualTo("/user-items"))
            .willReturn(aResponse()
                    .withStatus(201)));

    client.resource("http://localhost:" + port() + "/items")
            .post(ClientResponse.class, "{ \"id\": \"71283756\", \"name\": \"New Thing\" }");

    mockThingStore.verify(postRequestedFor(urlEqualTo("/user-items"))
            .withRequestBody(containing("71283756")));
}
```

```
@Test
public void health_check_should_fail_meaningfully_when_thing_store_returns_bad_http() {

    mockThingStore.stubFor(get(urlEqualTo("/user-items/1"))
            .willReturn(aResponse()
                    .withFault(MALFORMED_RESPONSE_CHUNK)));

    ClientResponse response =
            client.resource("http://localhost:" + adminPort() + "/healthcheck")
                    .get(ClientResponse.class);

    assertThat(response.getStatus(), is(500));
    assertThat(response.getEntity(String.class),
            containsString("Thing Store returned an unparseable HTTP response"));
}
```

And that’s pretty much it…thanks!