Streamline Integration Testing with Testcontainers

Andrew Guibert
Software Engineer
IBM

@andrew_guibert
Why do we write automated tests?
Why do we write automated tests?

To have confidence that our application works the way we want it to.
What type of testing gives us the most confidence?
Which type of tests contribute more to your confidence in your automated tests?

- Integration (i.e. "live server") tests: 76%
- Unit/Mock tests: 24%
4/4 Unit Tests Passing

- Faucet turns on
- Faucet turns off
- Drain works
- Sink does not overflow
What we have time to do

Integration Testing

Unit Testing

Manual testing if I remember and am not swamped
The dilemma of testing

Confidence vs. Time given
The dilemma of testing

Confidence vs. Time given

Confidence level in tests

@andrew_guibert
The Twelve-Factor App
The Twelve-Factor App

I. Codebase
II. Dependencies
III. Config
IV. Backing Services
V. Build, Release, Run
VI. Stateless Processes
VII. Port Binding
VIII. Concurrency
IX. Disposability
X. Dev-Prod Parity
XI. Logs
XII. Admin Processes
The Twelve-Factor App

I. Codebase
II. Dependencies
III. Config
IV. Backing Services
V. Build, Release, Run
VI. Stateless Processes
VII. Port Binding
VIII. Concurrency
IX. Disposability
X. Dev-Prod Parity
XI. Logs
XII. Admin Processes
Dev-Prod Parity

Keep development, staging, and production as similar as possible

Comprised of three common issues:

- The time gap
- The personnel gap
- The tools gap
Dev-Prod Parity

“The twelve-factor developer resists the urge to use different backing services between development and production, even when adapters theoretically abstract away any differences in backing services.”
It works on my machine.
public class CustomConfigSource implements ConfigSource {
  String fileLocation = System.getProperty("user.dir").split("target")[0] + "resources/CustomConfigSource.json";

  @Override
  public int getOrdinal() {
    return Integer.parseInt(getProperties().get("config_ordinal"));
  }

  @Override
  public Set<String> getPropertyNames() {
    return getProperties().keySet();
  }

  @Override
  public String getValue(String key) {
    return getProperties().get(key);
  }

  @Override
  public String getName() {
    return "Custom Config Source";
  }
}
Can you spot what’s wrong here?

```java
public class CustomConfigSource implements ConfigSource {
    String fileLocation = System.getProperty("user.dir").split("target")[0] + "resources/CustomConfigSource.json";

    @Override
    public int getOrdinal() {
        return Integer.parseInt(getProperties().get("config_ordinal"));
    }

    @Override
    public Set<String> getPropertyNames() {
        return getProperties().keySet();
    }

    @Override
    public String getValue(String key) {
        return getProperties().get(key);
    }

    @Override
    public String getName() {
        return "Custom Config Source";
    }
}
```
Testing with databases

H2 → PostgreSQL

Apache Derby → IBM DB2
Why do we do this?

- Easy to set up
- Faster than the real thing
- Been doing it this way for >10 years
Containers

- Combines everything needed to run a service into a single unit
- Standard way to manage
- Lightweight and disposable
- Typically configured via env
public class ExampleTest {

    @ClassRule
    public static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>()
        .withDatabaseName("testdb")
        .withUsername("testuser")
        .withPassword("testpass");

    // test methods....
}
Using Testcontainers

- Databases
- Messaging Brokers
- External Services
- Anything in a container
Taking things a step further

MicroShed Testing
Motivation

Devs lack confidence in their tests because...

• Tests do not reflect prod
• Writing tests takes too long
• Setup is too complex

What factors are limiting the confidence in your automated tests?

- Setting up or configuring tests is too complicated: 9
- The tests we have do not accurately represent how...: 18
- Debugging a test that fails takes too long: 7
- Running the tests we have takes too long: 10
- Our app is too big/complex to test: 11
- Writing tests is too complicated or time...: 11
1) Easy to set up

2) Works with
   • Java EE
   • Jakarta EE
   • MicroProfile

3) Provide true-to-production tests
MicroShed Testing

Test client JVM

- User test code
- MicroShed Testing
- Convenience libs (e.g. REST client)
- Testcontainers
- docker-java

Docker network *

- App container
- Other container resources (e.g. DB)

Test request/responses

controls

@andrew_guibert
@Path("/people")
@ApplicationScoped
@Produces(MediaType.APPLICATION_JSON)
public class PersonService {

    private final Map<Long, Person> personRepo = // ...

    @GET
    public Collection<Person> getAllPeople() {
        return personRepo.values();
    }

    @GET
    @Path("/{personId}")
    public Person getPerson(@PathParam("personId") long id) {
        Person foundPerson = personRepo.get(id);
        if (foundPerson == null)
            personNotFound(id);
        return foundPerson;
    }
}
@MicroShedTest
public class JaxrsJsonTest {

    @Container
    public static MicroProfileApplication app = new MicroProfileApplication()
        .withAppContextRoot("/myservice")
        .withReadinessPath("/myservice/app/people");

    @Inject
    public static PersonService personSvc;

    @Test
    public void testGetPerson() {
        Long bobId = personSvc.createPerson("Bob", 24);
        Person bob = personSvc.getPerson(bobId);
        assertEquals("Bob", bob.name);
        assertEquals(24, bob.age);
        assertNotNull(bob.id);
    }
}
Automatically discover, build, and start services

a) Provide container label

b) Locate Dockerfile

c) Use default image

```java
@MicroShedTest
public class JaxrsJsonTest {

    @Container
    public static MicroProfileApplication app = ...
        .withAppContextRoot("/myservice")
        .withReadinessPath("/myservice")
        ...

    @Inject
    public static PersonService personSvc;

    @Test
    public void testGetPerson() {
        Long bobId = personSvc.createPerson("Bob");
        Person bob = personSvc.getPerson(bobId);
        assertEquals("Bob", bob.name);
        assertEquals(24, bob.age);
        assertNotNull(bob.id);
    }
}
```
Wait for app container to be ready

- Defaults to app context root
- Can use MP Health 2.0 readiness check
- Can supply path manually

```java
t
 JaxrsJsonTest {

  static MicroProfileApplication app = new MicroProfileApplication
    .withAppContextRoot("/myservice")
    .withReadinessPath("/myservice/app/people")
  ...

  static PersonService personSvc;

  @test GetPerson()
  {
    bobId = personSvc.createPerson("Bob", 24);
    bob = personSvc.getPerson(bobId);
    assertEquals("Bob", bob.name);
    assertEquals(24, bob.age);
    assertNotNull(bob.id);
  }
```
Inject REST client

- Can re-use class from app
- Can define own client
- Automatically configured for app container

```java
@MicroShedTest
class JaxrsJsonTest {

    @Container
    public static MicroProfileApplication app = new ...
    .withAppContextRoot("/myservice")
    .withReadinessPath("/myservice")

    @Inject
    public static PersonService personSvc;

    @Test
    public void testGetPerson() {
        Long bobId = personSvc.createPerson("Bob")
        Person bob = personSvc.getPerson(bobId);
        assertEquals("Bob", bob.name);
        assertEquals(24, bob.age);
        assertNotNull(bob.id);
    }
}
```
Drive requests on running app container

- Real HTTP requests
- Parameters and return values converted with JSON-B
- JWT tokens applied automatically

```java
@MicroShedTest
public class JaxrsJsonTest {

    @Container
    public static MicroProfileApplication app = ...
        .withAppContextRoot("/myservice")
        .withReadinessPath("/myservice");

    @Inject
    public static PersonService personSvc;

    @Test
    public void testGetPerson() {
        Long bobId = personSvc.createPerson("Bob");
        Person bob = personSvc.getPerson(bobId);
        assertEquals("Bob", bob.name);
        assertEquals(24, bob.age);
        assertNotNull(bob.id);
    }
}
```
Where to learn more

**MicroShed**
- microshed.org/microshed-testing/
- github.com/MicroShed/microshed-testing

**Testcontainers**
- testcontainers.org
- github.com/testcontainers/testcontainers-java

**Demo Used**
- github.com/aguibert/oc1-demo

Andrew Guibert
@andrew_guibert
@aguibert