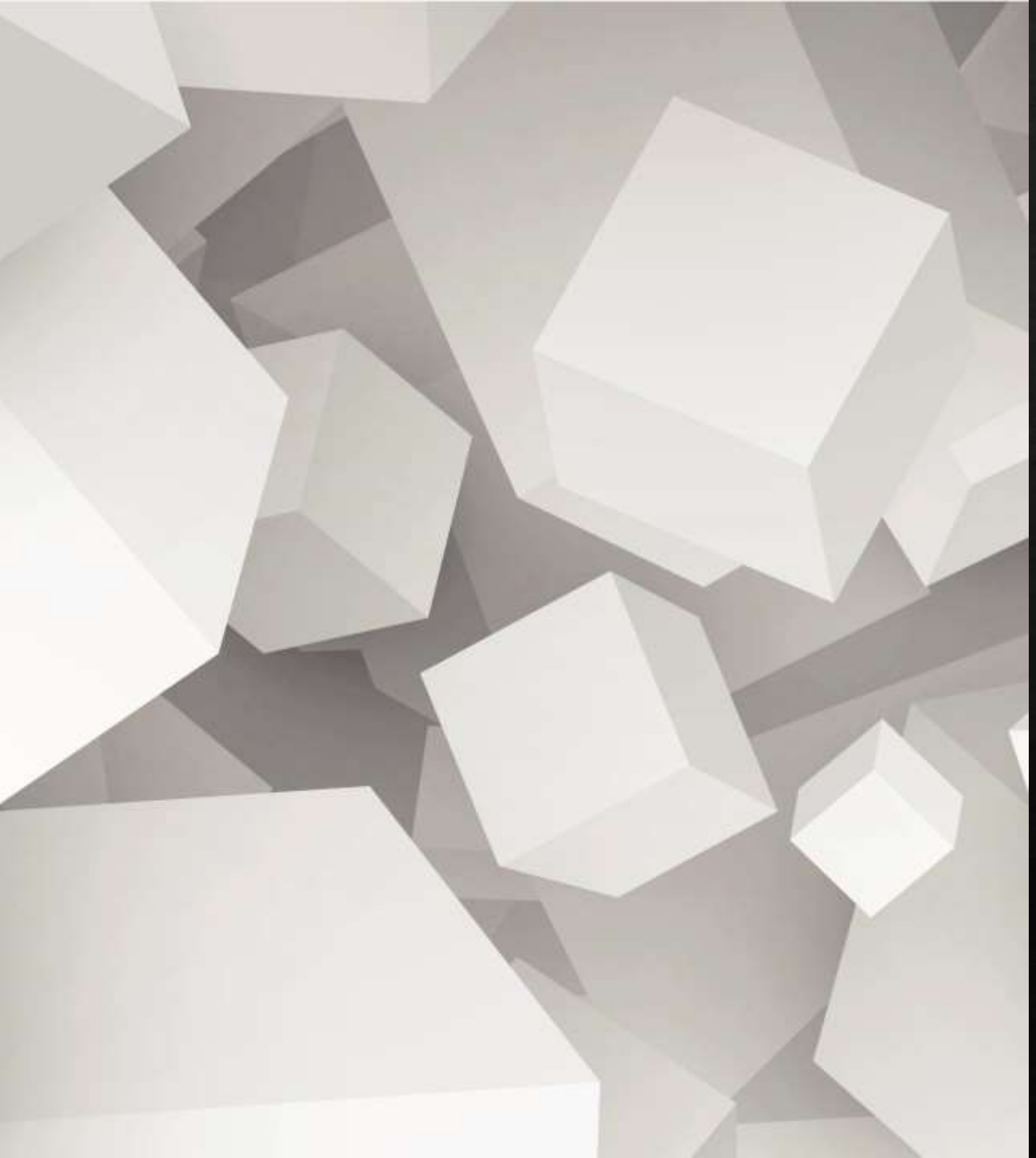
Replicating production on your laptop using the magic of containers

Jamie Lee Coleman

Software Engineer/Advocate @ IBM

Twitter: @Jamie_Lee_C



What we will cover

- How it begun
- The problems we are trying to solve
- Testing with Containers
- Testcontainers
- MicroShed Testing
- Open Liberty and MicroShed
- Interactive Demo
- Overview and Links

In the beginning…

Alright maybe not quite the beginning

1979

UnixV7

History of containers

1979 - UnixV7

History of containers

1979 - UnixV7

2006 – Control Groups (CGroups) CPU, memory, disk I/O, network

History of containers

1979 - UnixV7

2006 – Control Groups (CGroups) CPU, memory, disk I/O, network

2008 – LXC LinuX Containers

History of containers

1979 - UnixV7

2006 – Control Groups (CGroups) CPU, memory, disk I/O, network

2008 – LXC LinuX Containers

2013 – Docker

History of containers

1979 - UnixV7

2006 – Control Groups (CGroups) CPU, memory, disk I/O, network

2008 – LXC LinuX Containers

2013 – Docker

2014 - Kubernetes

What the Magic of Containers Provide

- Virtualization from OS level
  - Lightweight
  - Faster start-up
- Portable with images and config files such as a Dockerfile
- Run Anywhere
- Isolation
- Complexity!

What the Magic of Kubernetes Provide

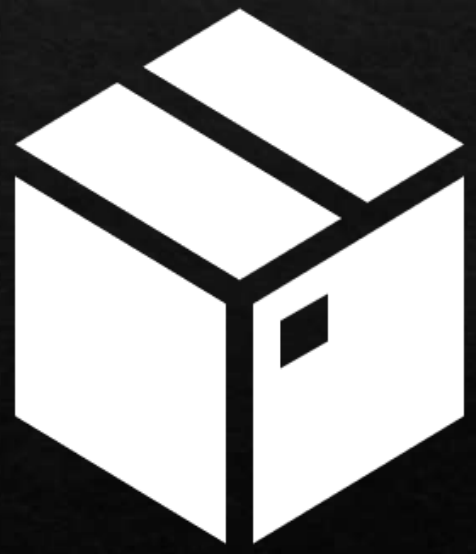
- Automated Rollouts and rollbacks
- Automatic scaling of services (containers)
- Health monitoring
- Declarative management
- Deploy anywhere (Hybrid deployments)

What Magic can Developers use?

- Isolated development environments
- Portable
- Preconfigured images available
- Fast start-up of applications
- No need to have so many pre-reqs installed
- Version control of dependencies
- Can develop easily in the cloud
- True-to production testing!

In the beginning…

We had the application…



Then we had the test…

Problems with Testing that Containers can solve

- Data access such access to databases
- Integration testing
- Automatic updating and version control
- Complex setup on local machines
- Portable testing environments (So others can run the same test
- …

Testing with Containers

History of containers

1979 - UnixV7

2006 – Control Groups (CGroups) CPU, memory, disk I/O, network

2008 – LXC LinuX Containers

2013 – Docker

2014 – Kubernetes

2015 - Testcontainers

Problems that Containers once had with Testing

Containers where never really designed for testing purposes but they have the capacity to help with many testing issues developers encounter.

For example containers are not designed for GUI interfaces and they did not have support for many different OS such as Windows and Mac. This has now all changed!

Testcontainers

What is Testcontainers?

Testcontainers is a Java library that provides lightweight, throwaway instances of anything that can run in a Docker container and supports Junit.

Testcontainers make a variety of different testing easier such as:

- Data access layer integration tests
- Application integration tests
- UI/Acceptance tests
- Much more via contributed modules
- Supports JUnit 4/5 and Spok

Browser Testing

Testcontainers give you the ability to do UI/Acceptance testing in a container. This has great benefits such as:

- Fresh instance of a browser
- No browser state, plugin variations or browser upgrades
- Video recoding of test session or just failed test sessions

Some of the Testcontainer Modules Available

Database Modules:

- CouchBase
- DB2
- MongoDB
- MySQL
- Postgress etc

Other Modules:

- Docker Compose
- Elasticsearch
- Kafka
- Nginx
- RabbitMQ

People Love Testcontainers

Testcontainers Retweeted

Andres Almiray @aalmiray · Oct 7

How do you test with a db? Please comment if different option.

@testcontainers ✓	47.4%
Local db running	33.7%
Db in the cloud	6.9%
I don't	11.9%

1,408 votes · 4 days left

41 24 33

Testcontainers Retweeted

Mohammed Aboullaite @laytoun · Apr 1

I like @testcontainers a lot! A feeling of excitement comes up whenever I see some project using it.

1 1 17

Testcontainers Retweeted

Maciej Walkowiak @maciejwalkowiak · Apr 21

Till not so long ago, integration tests were tricky - but since we have Docker and @testcontainers there is literally no excuse to cover whole code with tests - including messaging, persistence, search, cache, EVERYTHING!

2 3 25

Show this thread

Testcontainers Retweeted

Trisha Gee @trisha_gee · Jun 10

Used @testcontainers for the first time today. Holy crap! They make integration testing MUCH more reproducible.

20 30 219

Introducing MicroShed Testing

What is MicroShed Testing?

The Microshed Testing framework allows for true-to-production integration tests on your local machine. This enables you as a developer to minimize the amount of test failures due to differences in dev/test and production environments. Using the magic of containers, you can replicate what you have in your production environment without much setup at all.

MicroShed Example

The @MicroShedTest annotation is required so that the build knows this is a MicroShed test

The @Container annotation is required for setting up your container to run your tests inside

The @RESTClient annotation is needed to give us something to test against

The @Test annotation like you would in a normal JUnit test.

```
@MicroShedTest
public class MyTest {

    // Search for Dockerfile.
    // Start app in Container.
    // Wait for Container before running tests.
    @Container
    public static MicroProfileApplication app
        = new MicroProfileApplication()
                .withAppContextRoot("/myservice");

    // Inject JAX-RS REST Client
    @RestClient
    public static MyService mySvc;

    // A test method like any other
    @Test
    public void testMyService() {
        ...
    }
}
```

Sharable Configuration

If multiple test classes can share the same container instances, they can offload their @Container annotated fields to a sperate class like so:

```
public class AppContainerConfig implements SharedContainerConfiguration {
    @Container
    public static ApplicationContainer app = new ApplicationContainer()
                    .withAppContextRoot("/myservice");
}

@MicroShedTest
@SharedContainerConfig(AppContainerConfig.class)
public class MySimpleTestA {
    // ...
}

@MicroShedTest
@SharedContainerConfig(AppContainerConfig.class)
public class MySimpleTestB {
    // ...
}
```

Different Runtimes Available

- Open Liberty
- Payara Micro
- Payara Sever
- Wildfly
- Quarkus

Open Liberty and MicroShed

Open Liberty™

What you can do with MicroShed and Open Liberty

mvn liberty:dev

```
[INFO] Response from server: -3746667975365372121
[INFO] Response from server: {"age":42,"id":-3746667975365372121,"name":"Ha"}
[INFO] Response from server: 1851947796734870224
[INFO] Response from server: {"age":0,"id":1851947796734870224,"name":"Newborn"}
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 7.328 s - in io.openliberty.guides.testing.PersonServiceIT
[INFO] Running io.openliberty.guides.testing.ErrorPathIT
[INFO] Using ApplicationEnvironment class: org.microshed.testing.testcontainers.config.HollowTestcontainersConfiguration
[INFO] Exposing fixed port 9080 for container HollowApplicationContainer
[INFO] All containers started in 1ms
[INFO] Building rest client for class io.openliberty.guides.testing.PersonService with base path: http://localhost:9080/guide-microshed-testing/ and providers: [class
 org.microshed.testing.jaxrs.JsonBProvider]
[INFO] [WARNING ] Value 'NameTooLongPersonNameTooLongPersonNameTooLongPerson' of PersonService$Proxy$_$$_WeldClientProxy.createPerson.name: size must be between 2 and
 50
[INFO] [WARNING ] Value (null) of PersonService$Proxy$_$$_WeldClientProxy.createPerson.name: must not be empty
[INFO] [WARNING ] Value '-1' of PersonService$Proxy$_$$_WeldClientProxy.createPerson.age: must be greater than or equal to 0
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.28 s - in io.openliberty.guides.testing.ErrorPathIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] Integration tests finished.
[INFO] Press the Enter key to run tests on demand. To stop the server and quit dev mode, use Ctrl-C or type 'q' and press the Enter key.
```

Simple to Get Started

Maven Dependencies

```
<dependency>
    <groupId>org.microshed</groupId>
    <artifactId>microshed-testing-liberty</artifactId>
    <version>0.9</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.6.2</version>
    <scope>test</scope>
</dependency>
```

Java Imports

```
import org.junit.jupiter.api.Test;
import org.microshed.testing.jaxrs.RESTClient;
import org.microshed.testing.jupiter.MicroShedTest;
import org.microshed.testing.testcontainers.ApplicationContainer;
import org.testcontainers.junit.jupiter.Container;
```

Interactive Demo

Turning a JUnit test into a MicroShed test

If you would like to follow along at the same time please go to the following link and select the “Testing a MicroProfile or Jakarta EE Application” Module

https://openliberty.skillsnetwork.site/cloud-native-java-made-easy-microprofile-jakarta-ee

Summary

Containers are great!

They make developers lives easier (portable, configurable, lightweight…)

Containers sometimes make things more complicated (for the better)

But they can help solve some issues with testing such as dev/test prod parity

Speeding up time to productions and making everyone happier!

Links

Open Liberty - https://openliberty.io

MicroShed - https://microshed.org

Testcontainers - https://www.testcontainers.org/

MicroProfile - https://microprofile.io/

Jakarta EE - https://jakarta.ee/

MicroShed Testing Guide - https://openliberty.io/guides/microshed-testing.html

Reactive MicroShed Testing Guide - https://openliberty.io/guides/reactive-service-testing.html

Thank You

Jamie L Coleman

Role: Software Developer/Advocate

Technologies: Open Liberty, MicroProfile, Jakarta EE & OpenJ9

Twitter: @Jamie_Lee_C

LinkedIn: https://www.linkedin.com/in/jamie-coleman/