SAFECHARGE

Arquillian & ShrinkWrap

Integration testing made simple

November 2014

What do they do?

- ShrinkWrap to package your test artifacts
- Arquillian deploys the package to target container
- Arquillian enriches and executes the test inside your container
- NO MOCKS
- All done through Eclipse

What is ShrinkWrap?

- Java API analogue to the “jar” tool, a virtual file system
- What is a traditional build process?
- For the AS , it doesn’t matter if the bytes are received form disk or memory
- So?

What is ShrinkWrap?

› ShrinkWrap lets us to programmatically build our artifacts, effectively skipping the build

```
JavaArchive archive =
   ShrinkWrap.create(JavaArchive.class, "arc.jar")
.addClasses(MyClass.class, MyOtherClass.class)
.addResource("mystuff.properties");
```

ShrinkWrap packaging

- Java classes
- A Java package (which adds all the Java classes in the package)
- Classpath resources
- File system resources
- A programmatically-defined file
- Java libraries (JAR files)
- Other Java archives defined by ShrinkWrap

What is Arquillian?

- Arquillian is a

  test harness that can be used to produce an integration tests for Java applications
- Abstracts out server lifecycle and deployment
- Tests become container agnostic
- Works with JUnit and TestNG
- Works trough our IDE

How Arquillian test looks?

A:Just like a regular JUnit or TestNG test case with two declarative enhancements:

- The class contains a static method annotated with @Deployment that returns a JavaArchive
- The class is annotated with @RunWith(Arquillian.class) (JUnit) or extends Arquillian (TestNG)

How Arquillian test looks?

```java
package org.arquillian.example;

import javax.inject.Inject;

@RunWith(Arquillian.class)
public class GreeterTest {

    @Deployment
    public static JavaArchive createDeployment() {
        return ShrinkWrap.create(JavaArchive.class)
            .addClasses(Greeter.class, PhraseBuilder.class)
            .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");

    }

    @Inject
    Greeter greeter;

    @Test
    public void should_create_greeting() {
        Assert.assertEquals("Hello, Earthling!",
            greeter.createGreeting("Earthling"));
        greeter.greet(System.out, "Earthling");
    }
}
```

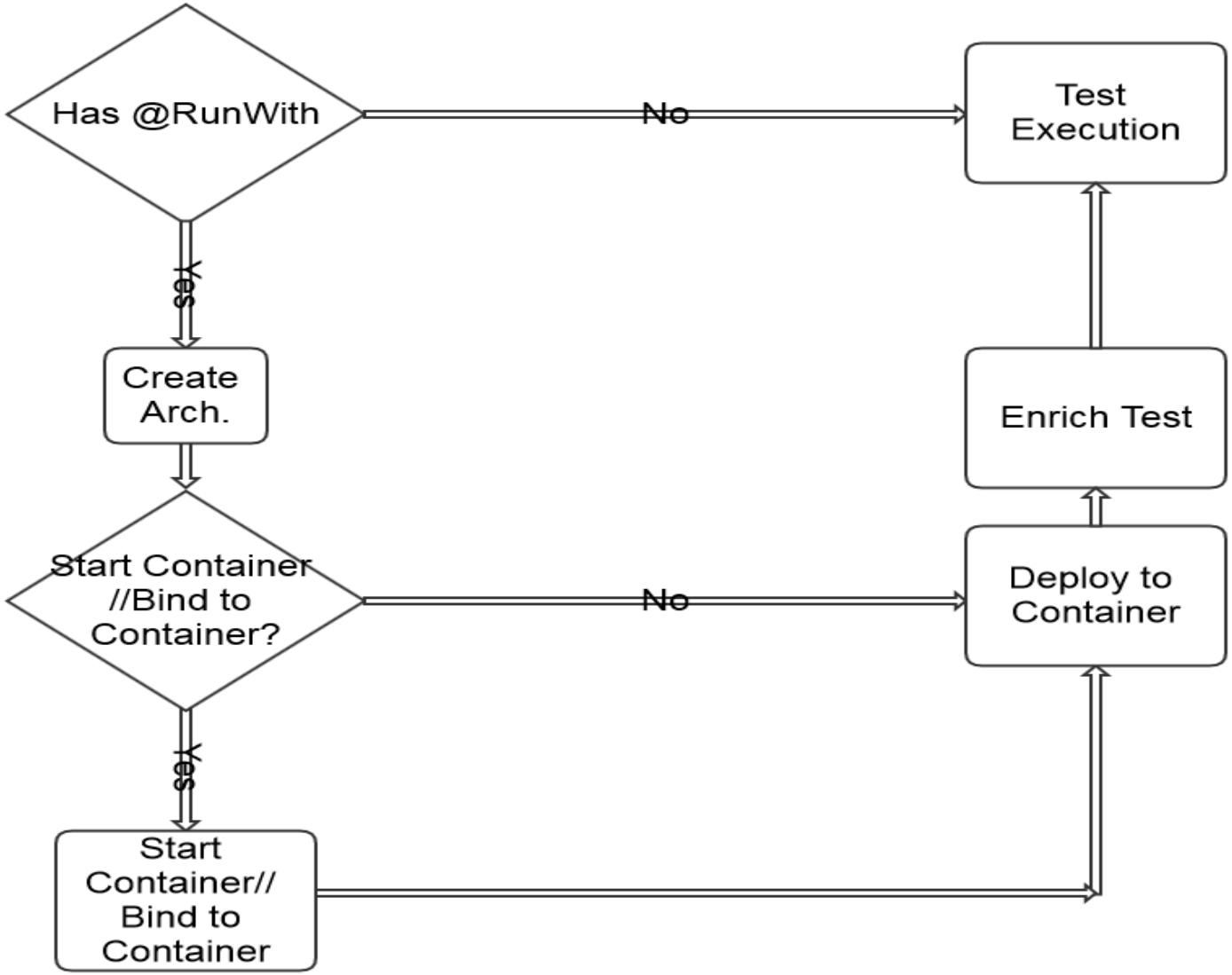
Tell JUnit to use Arquillian test

Make a virtual archive

Injected Bean, Managed by container

Normal Test?

Arquillian test algorithm



Container Types

- A *remote container* - separate JVM from the test runner – remote protocol Servlet, JMX , slower or faster?
- A *managed container* is similar to a remote container , but Arquillian controls its lifecycle
- An *embedded container* resides in the same JVM and is managed by Arquillian.

Lifecycle and container types

- Bundle Test Archive Uses ShrinkWrap archive
- Startup - Managed and Embedded OR
- Bind - Remote Container
- Deploy archive to container
- Enrich test class
- After execution, undeploy
- Stop Container OR
- Unbind from it
- Display Results

What container?

- Container adapters for most major servers:

  JBoss AS, GlassFish or OpenEJB, GlassFish Embedded or Weld SE
- We can test with one container, or another with the same correct results
- Extendable SPI , if no container adapter is provided

Single test in different containers?

- Container selection occurs at runtime - add the container adapter as a test-scoped dependency for the desired container.
- Arquillian delegates to an SPI (Service Provider Interface) for starting / stopping container and deploying / undeploying - org.jboss.arquillian.spi.client.DeployableContainer
- Maven Profiles – for swapping containers

Test enrichment

- @Inject
  - CDI implementation
  - your deployment is a valid bean archive
- @EJB and @Resource: Provide injection of a container-managed resource into your test code.

Test enrichment

- @Resource - Java EE resource injections
- @EJB - EJB session bean reference injections
- @Inject - CDI injections
- @PersistenceUnit
- @PersistenceContext

Container Adapters

- All Arquillian container adapter extensions must implement the LoadableExtension SPI. (register) the

DeployableContainer

- setup(T configuration)
- start()
- deploy(org.jboss.shrinkwrap.api.Archive<?> archive)

Container Adapters

› A container adapter declares the default set of test enrichers as classpath dependencies

```
public class MyContainerExtension implements
    LoadableExtension {
     @Override
     public void register(ExtensionBuilder builder) {
         builder.service(DeployableContainer.class,
    MyDeployableContainer.class);
     }
```

Examples

SAFECHARGE

Testing an EJB

```
@RunWith(Arquillian.class)
    public class InjectionTestCase {
        @Deployment
        public static JavaArchive createTestArchive() {
            return Archives.create("test.jar", JavaArchive.class)
                .addClasses(GreetingManager.class, GreetingManagerBean.class);
        }
        @EJB
        private GreetingManager greetingManager;
        @Test
        public void shouldBeAbleToInjectEJB() throws Exception {
            String userName = "Earthlings";
            Assert.assertEquals(Hello " + userName, greetingManager.greet(userName));
        }
    }
```

Testing JMS

```
@RunWith(Arquillian.class)
    public class InjectionTestCase {
       @Deployment  public static JavaArchive createTestArchive() {
         return Archives.create("test.jar", JavaArchive.class).addClasses(MessageEcho.class, QueueRequestor.class);
       }
       @Resource(mappedName = "/queue/DLQ") private Queue dlq;
       @Resource(mappedName = "/ConnectionFactory") private ConnectionFactory factory;
       @Test
       public void shouldBeAbleToSendMessage() throws Exception {
          String messageBody = "ping";
          Connection connection = factory.createConnection();
          Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
          QueueRequestor requestor = new QueueRequestor((QueueSession) session, dlq);
          connection.start();
          Message request = session.createTextMessage(messageBody);
          Message response = requestor.request(request, 5000);
          Assert.assertEquals("Should have responded with same message", messageBody, ((TextMessage) response).getText())
       }
    }
```

THANK YOU

SAFECHARGE