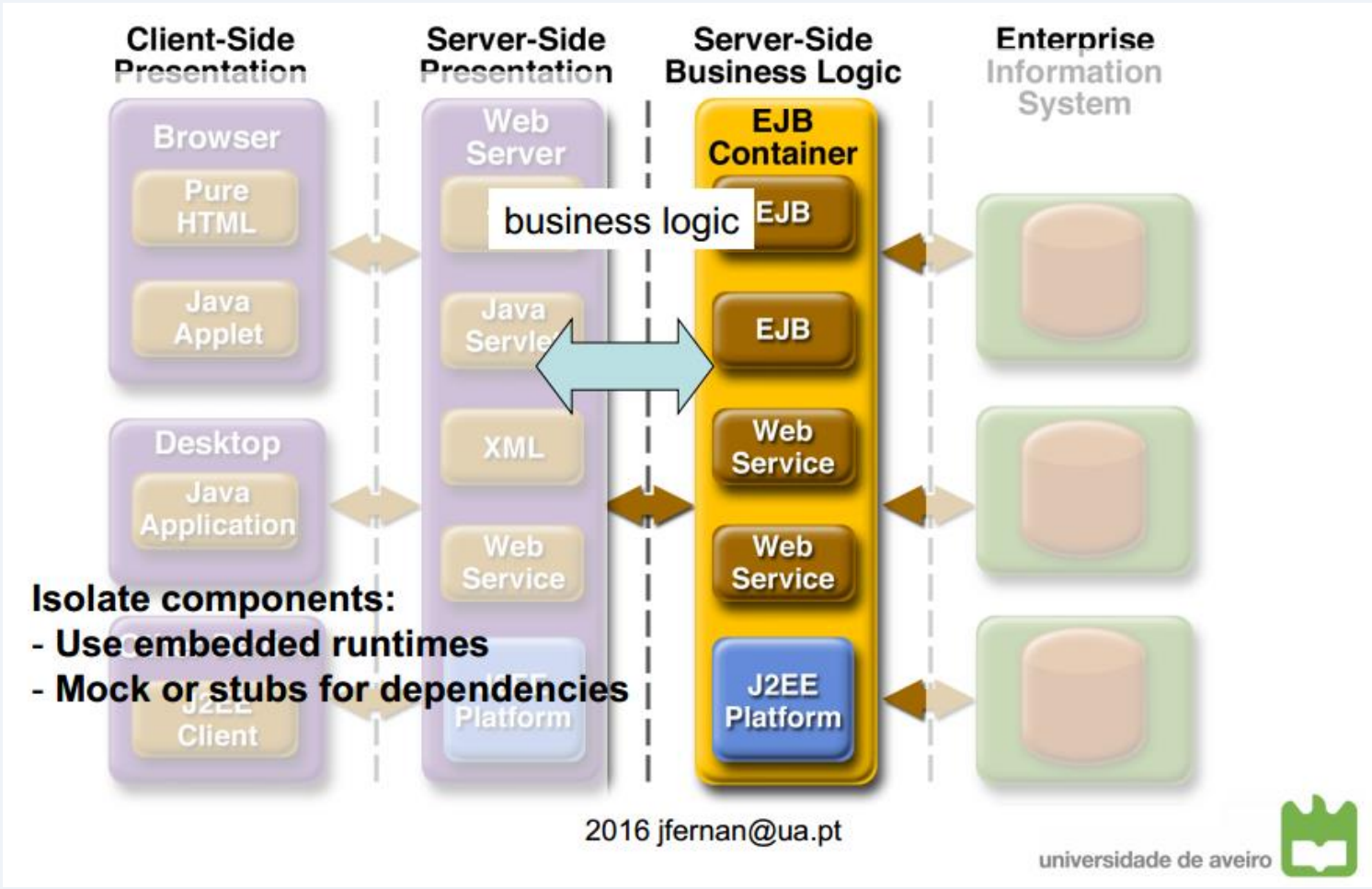
Integration testing in Java EE

DETI-UA/TQS
Ilídio Oliveira (ico@ua.pt)
v2019-03-12

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

Integration testing is difficult

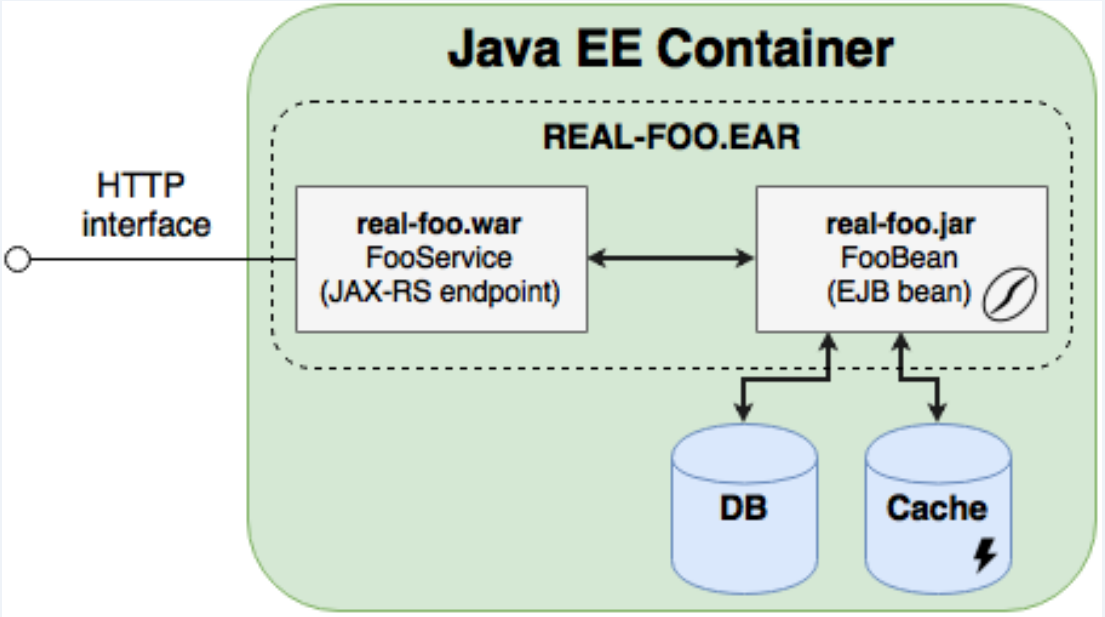


Java EE layers.

Test components that live in containers

Strategies

1. Unit testing, by mocking the containers services
2. Integration testing, via endpoint access
3. Embedded containers (without Arquillian)
4. Embedded containers (with Arquillian)



JavaEE application, with endpoint and EJB.

http://develorium.com/2016/10/testing-of-javaee-applications-with-arquillian/

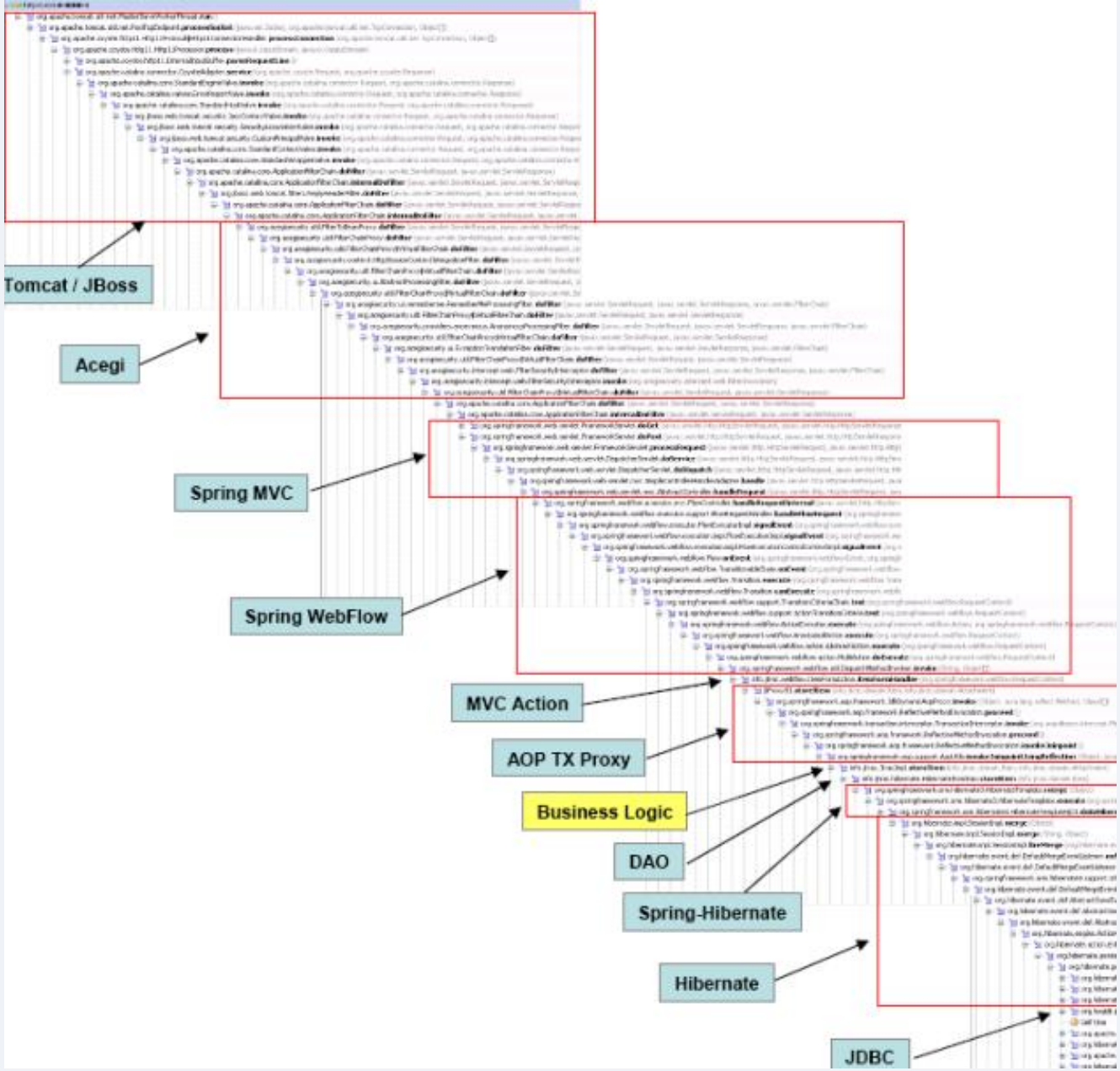
Strategies

Junit and mocking the complex services

- See AGonçalves blog article
- ☹

Still not the real thing…

Wait… containers?…



https://ptrthomas.wordpress.com/2006/06/06/java-call-stack-from-http-upto-jdbc-as-a-picture/

Using Arquillian

Define target containers with configuration

- POM and profiles

Create artifacts

- ShrinkWrap framework

Start container

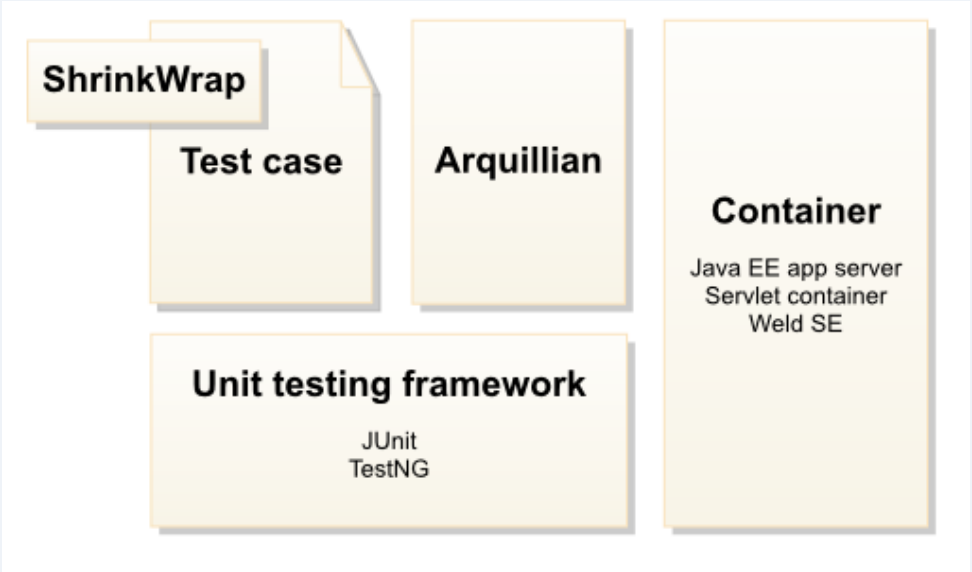
- Remote, Managed, Embedded

Deploy SuT

Run tests

- Client vs container

Stop container



http://arquillian.org/

Anatomy of an Arquillian test

1. A @RunWith(Arquillian.class) annotation on the class
2. A public static method annotated with @Deployment that returns a ShrinkWrap archive
3. At least one method annotated with @Test

```java
@RunWith(Arquillian.class)
public class GreeterTest {

    @Deployment
    public static JavaArchive createDeployment() {
        return ShrinkWrap.create(JavaArchive.class)
            .addClass(Greeter.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");
    }
}
```

Container lifecycle management

Remote

- AS running in a remote process
- Connect and run tests

Managed

- AS running in a different process
- Lifecycle is managed by Arquillian

Embedded

- Life cycle controlled by Arquillian (same JVM)

Execution modes

Run tests in the container

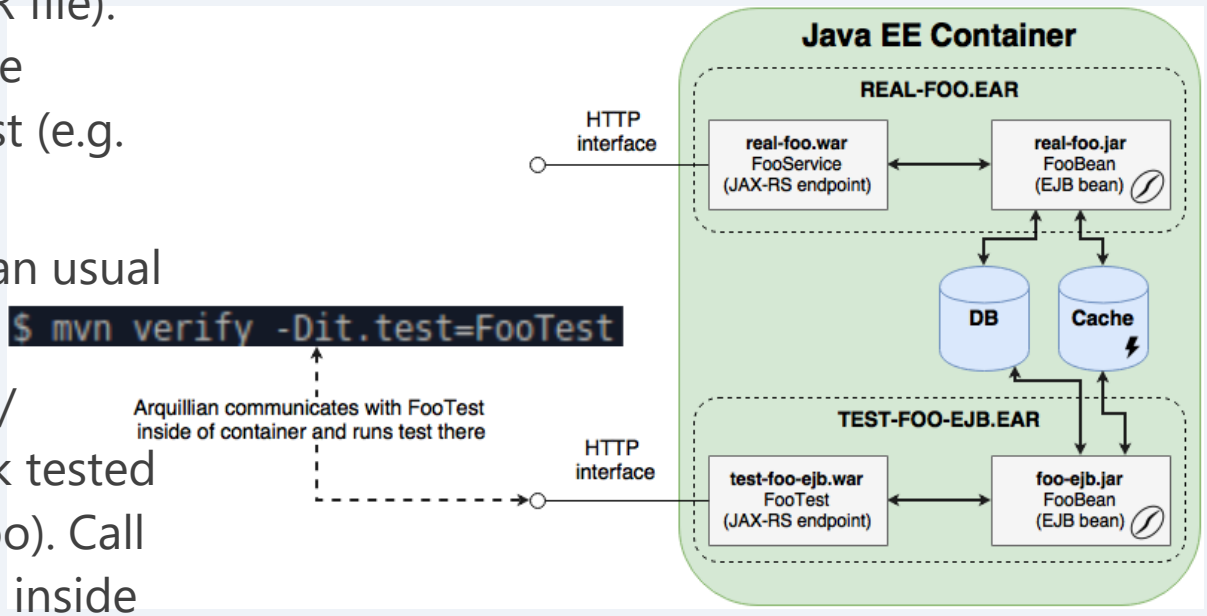
- Test the internals EJB, queues, JPA
- CDI ok

Run as a client

- e.g. client to REST api

Arquillian style: "real" tests

- Craft test artifact (e.g. WAR file). Include into this artifact the component we want to test (e.g. FooBean)
- Write test class as it were an usual unit test.
- Inside test class use @EJB / @Inject annotations to link tested modules (e.g. @EJB Foo foo). Call methods of linked module inside @Test methods
- Deploy resulting test artifact to a real container and execute test inside of this container



SpringBoot

Spring Boot Test Starter

- a Starter for testing Spring Boot applications with libraries including JUnit, Hamcrest and Mockito

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

SpringBoot way

```java
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HelloControllerIT {

    @LocalServerPort
    private int port;

    private URL base;

    @Autowired
    private TestRestTemplate template;

    @Before
    public void setUp() throws Exception {
        this.base = new URL( spec: "http://localhost:" + port + "/");
    }

    @Test
    public void getHello() throws Exception {
        ResponseEntity<String> response = template.getForEntity(base.toString(),
                String.class);
        assertThat(response.getBody(), equalTo( operand: "Greetings from Spring Boot!"));
    }
}
```

Unit Test launching the complete Spring Context using @MockBean

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class BusinessServicesMockSpringContextTest {

	@MockBean
	DataService dataServiceMock;

	@Autowired
	BusinessService businessImpl;

	@Test
	public void testFindTheGreatestFromAllData() {
		when(dataServiceMock.retrieveAllData()).thenReturn(new int
		assertEquals(24, businessImpl.findTheGreatestFromAllData()
	}

	@Test
	public void testFindTheGreatestFromAllData_ForOneValue() {
		when(dataServiceMock.retrieveAllData()).thenReturn(new int
		assertEquals(15, businessImpl.findTheGreatestFromAllData()
	}

	@Test
	public void testFindTheGreatestFromAllData_NoValues() {
		when(dataServiceMock.retrieveAllData()).thenReturn(new int
		assertEquals(Integer.MIN_VALUE, businessImpl.findTheGreates
	}
```