AARHUS UNIVERSITET

Microservices and DevOps

DevOps and Container Technology

TestContainers

Henrik Bærbak Christensen

Motivation

- Service tests, Consumer Driven Tests, test journeys
  - Fowler: Component tests out-of-process, contract tests,…
- … all requires that you do *automated testing of multiple services being started for testing purposes*

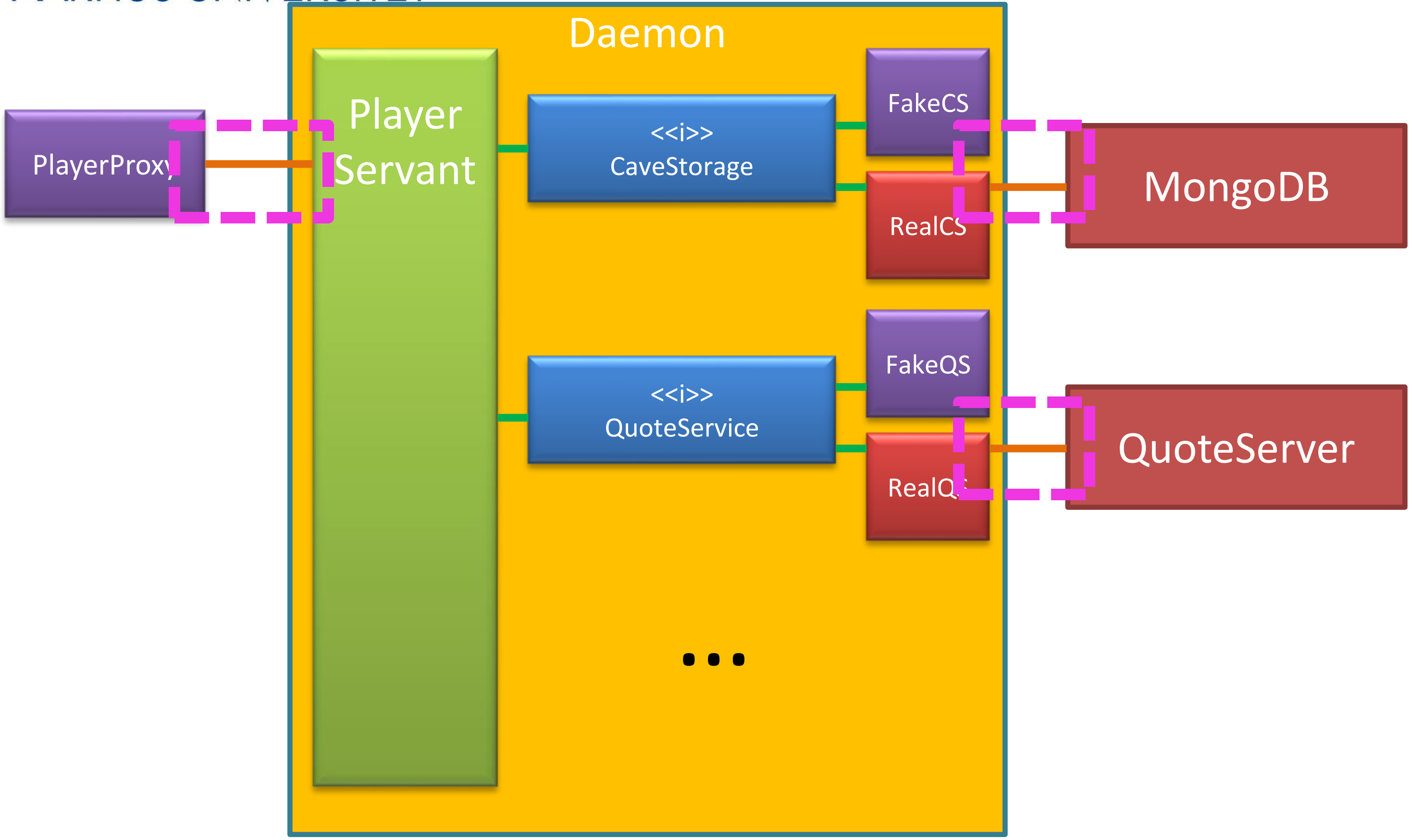
- *Which means you have to start services from within JUnit, which is seldom easy…*

AARHUS UNIVERSITET

The Pesky Network

What are our Options?

Test Across Network



🟩 = Local call

🟧 = Network call

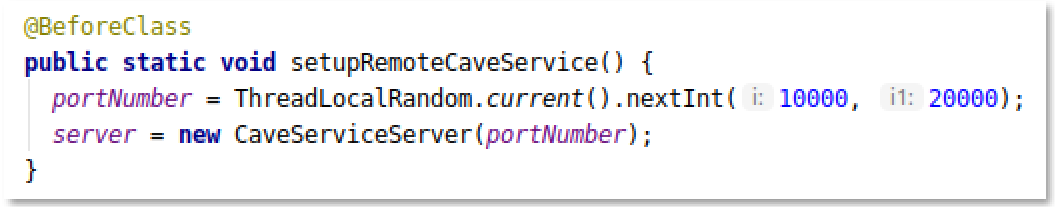
AARHUS UNIVERSITET

The Easy Case

- One case is do-able, however.

- Starting a Spark-Java web server can easily be done in JUnit.

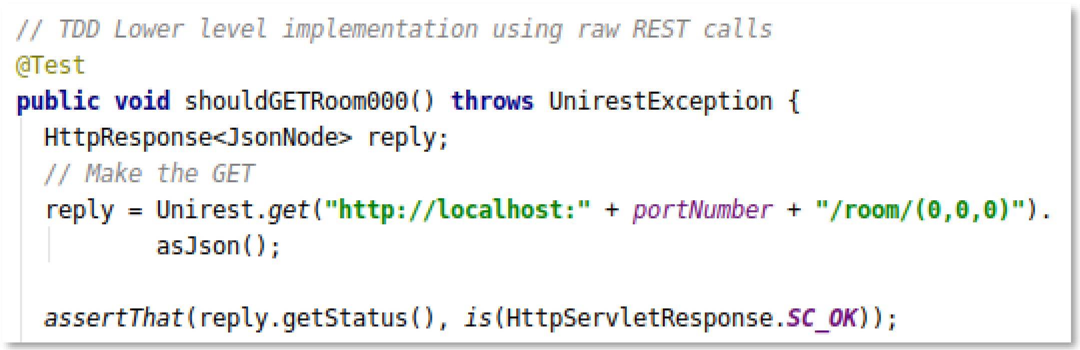
- Linux Issue!
  - Releasing a port takes a loong time.
  - Use *random port in each @Before*



```
@BeforeClass
public static void setupRemoteCaveService() {
  portNumber = ThreadLocalRandom.current().nextInt( i: 10000,  i1: 20000);
  server = new CaveServiceServer(portNumber);
}
```

The Easy Case

- You may now formulate JUnit test cases that directly contact that server



```
// TDD Lower level implementation using raw REST calls
@Test
public void shouldGETRoom000() throws UnirestException {
  HttpResponse<JsonNode> reply;
  // Make the GET
  reply = Unirest.get("http://localhost:" + portNumber + "/room/(0,0,0)").
          asJson();

  assertThat(reply.getStatus(), is(HttpServletResponse.SC_OK));
```

- Actually runs reasonably fast …

The Tricky Cases

- But What do I do if the external service is
  - A database, like MariaDB, MongoDB, Redis, …?
  - A web service I do not develop myself?
    - Like the other groups in the course and you need to validate their service using CDTs
  - Even worse – some external service that only runs in one instance???

Some Options

- PowerMock
  - mock(UniRest) cumbersome
- Provided mock frameworks
  - Fongo for Mongo available?
- Test double services
  - Mountebank only http
- Call system/OS processes
  - ProcessBuilder() os dependent/flaky
- TestContainers
  - Run docker within JUnit ***cool stuff!***

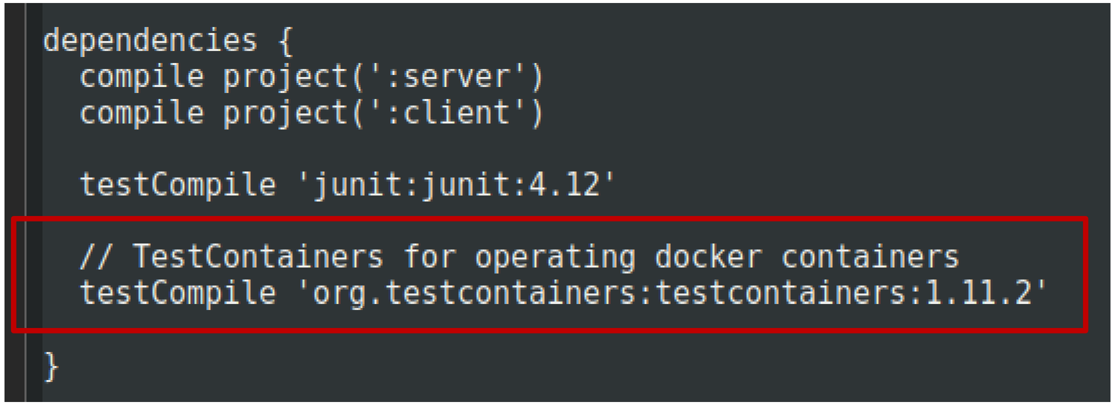
AARHUS UNIVERSITET

TestContainers

Running Docker from JUnit

Getting It

- Gradle to the rescue

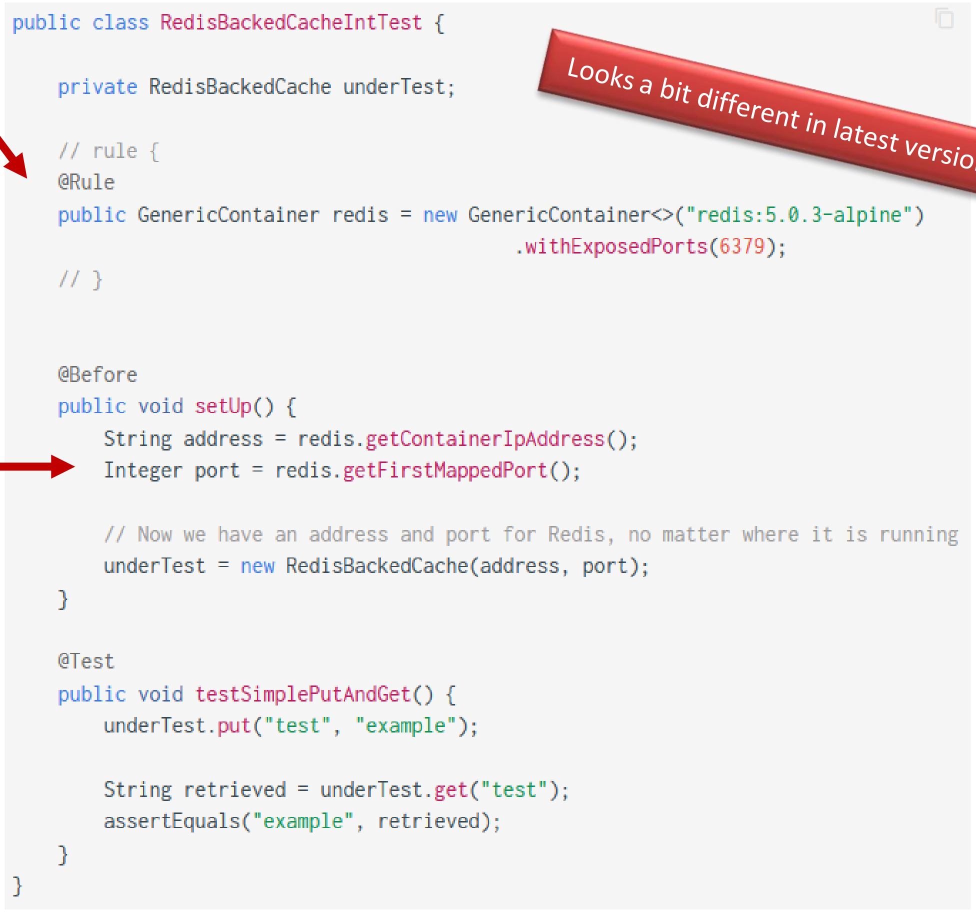


```
dependencies {
  compile project(':server')
  compile project(':client')

  testCompile 'junit:junit:4.12'

  // TestContainers for operating docker containers
  testCompile 'org.testcontainers:testcontainers:1.11.2'

}
```

1.15.2 / 1.16.0

GenericContainer

- @Rule
  - Creates a ‘redis’ for each test
  - Tell exposed port (-p)
- Randomized port mapping
  - Configure our ‘driver’
- Write your tests as normal

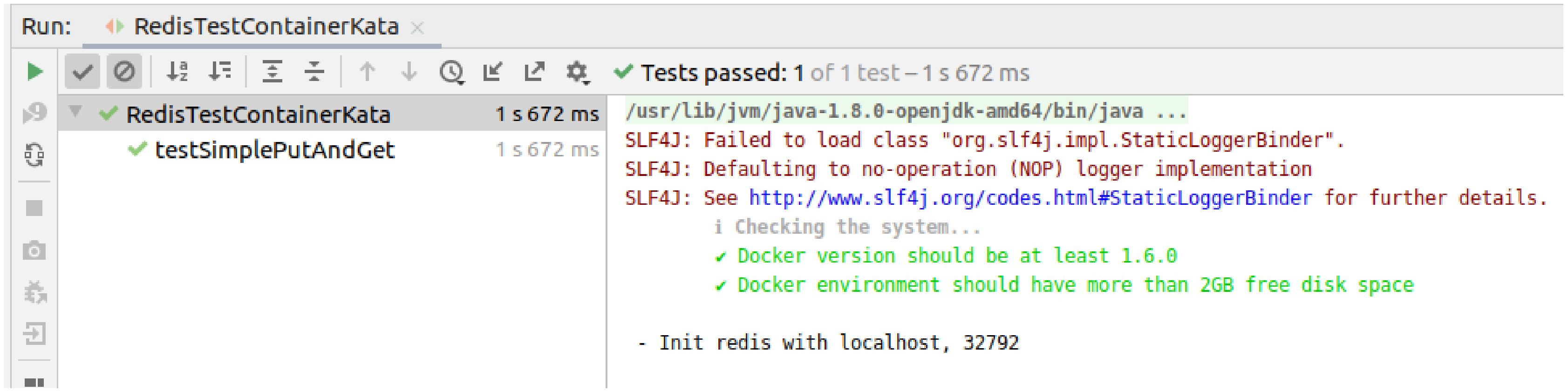


Looks a bit different in latest version

```
public class RedisBackedCacheIntTest {

    private RedisBackedCache underTest;

    // rule {
    @Rule
    public GenericContainer redis = new GenericContainer<>("redis:5.0.3-alpine")
                                            .withExposedPorts(6379);
    // }


    @Before
    public void setUp() {
        String address = redis.getContainerIpAddress();
        Integer port = redis.getFirstMappedPort();

        // Now we have an address and port for Redis, no matter where it is running
        underTest = new RedisBackedCache(address, port);
    }

    @Test
    public void testSimplePutAndGet() {
        underTest.put("test", "example");

        String retrieved = underTest.get("test");
        assertEquals("example", retrieved);
    }
}
```

And Run…

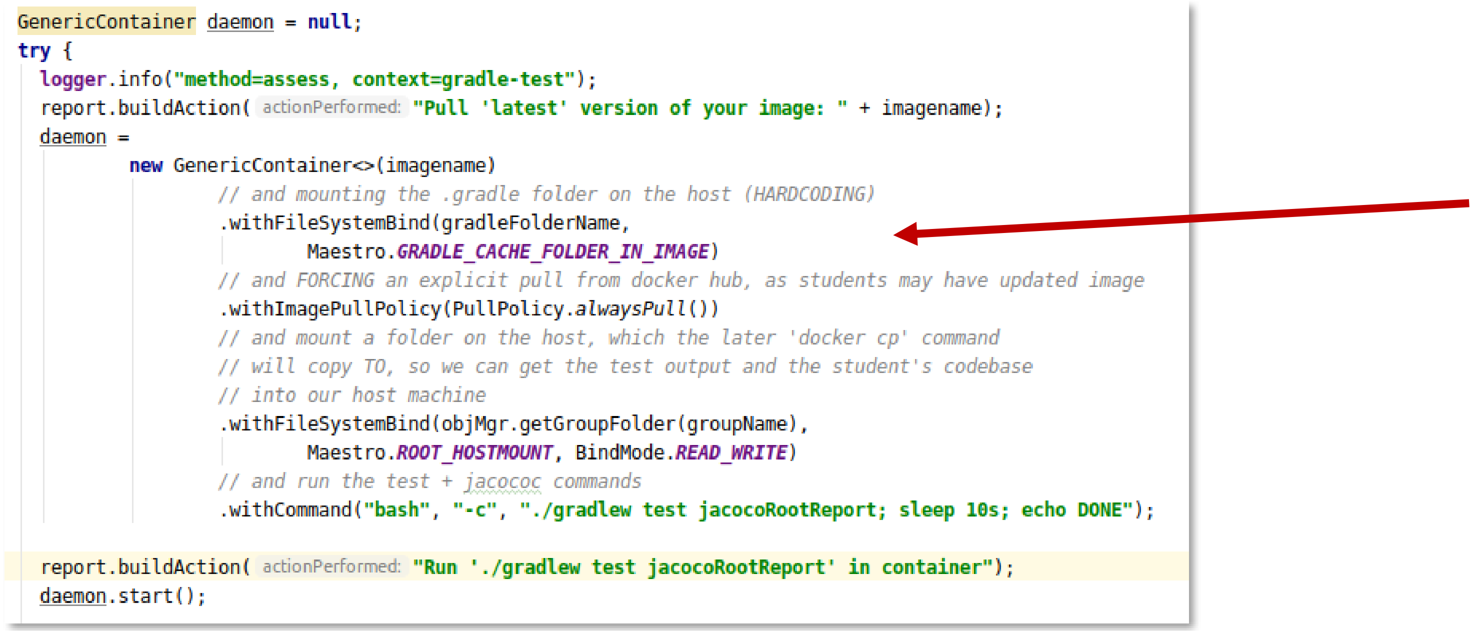
- Run slowly, but still…


Run: RedisTestContainerKata
Tests passed: 1 of 1 test – 1 s 672 ms
RedisTestContainerKata 1 s 672 ms
testSimplePutAndGet 1 s 672 ms

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
        i Checking the system...
        ✔ Docker version should be at least 1.6.0
        ✔ Docker environment should have more than 2GB free disk space

 - Init redis with localhost, 32792
```

Lots of Control, Vast API

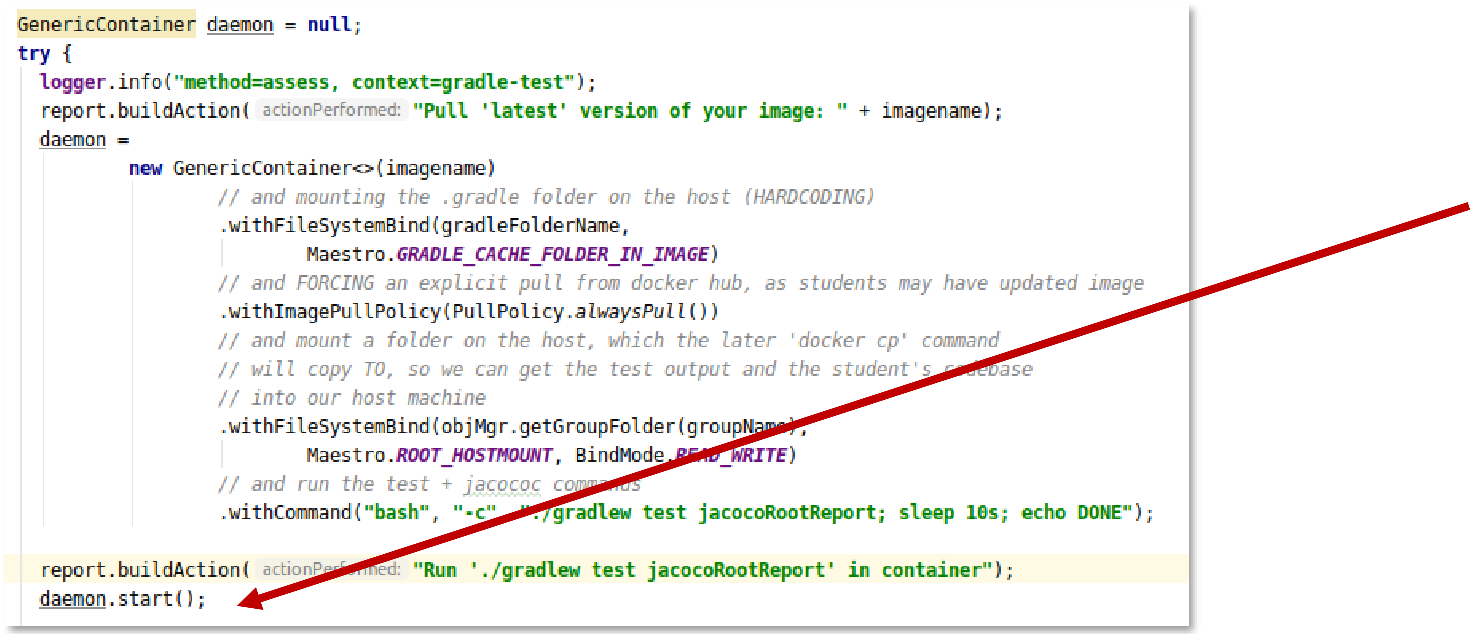
- You can configure your docker 'run' detailed
  - Force image pull, mount folders, set run command



```
GenericContainer daemon = null;
try {
  logger.info("method=assess, context=gradle-test");
  report.buildAction( actionPerformed: "Pull 'latest' version of your image: " + imagename);
  daemon =
          new GenericContainer<>(imagename)
                  // and mounting the .gradle folder on the host (HARDCODING)
                  .withFileSystemBind(gradleFolderName,
                          Maestro.GRADLE_CACHE_FOLDER_IN_IMAGE)
                  // and FORCING an explicit pull from docker hub, as students may have updated image
                  .withImagePullPolicy(PullPolicy.alwaysPull())
                  // and mount a folder on the host, which the later 'docker cp' command
                  // will copy TO, so we can get the test output and the student's codebase
                  // into our host machine
                  .withFileSystemBind(objMgr.getGroupFolder(groupName),
                          Maestro.ROOT_HOSTMOUNT, BindMode.READ_WRITE)
                  // and run the test + jacococ commands
                  .withCommand("bash", "-c", "./gradlew test jacocoRootReport; sleep 10s; echo DONE");

  report.buildAction( actionPerformed: "Run './gradlew test jacocoRootReport' in container");
  daemon.start();
```

Lots of Control, Vast API

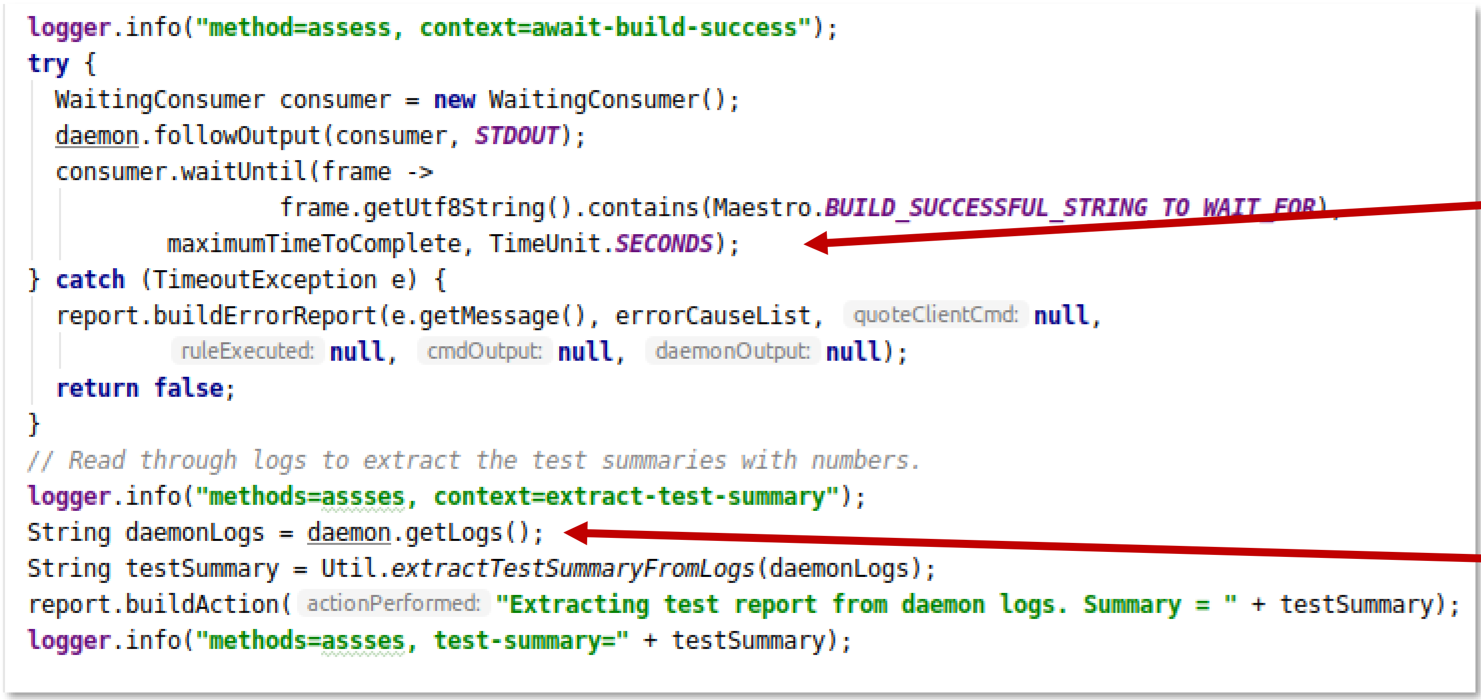
- You can start/stop containers



```
GenericContainer daemon = null;
try {
  logger.info("method=assess, context=gradle-test");
  report.buildAction( actionPerformed: "Pull 'latest' version of your image: " + imagename);
  daemon =
          new GenericContainer<>(imagename)
                  // and mounting the .gradle folder on the host (HARDCODING)
                  .withFileSystemBind(gradleFolderName,
                          Maestro.GRADLE_CACHE_FOLDER_IN_IMAGE)
                  // and FORCING an explicit pull from docker hub, as students may have updated image
                  .withImagePullPolicy(PullPolicy.alwaysPull())
                  // and mount a folder on the host, which the later 'docker cp' command
                  // will copy TO, so we can get the test output and the student's codebase
                  // into our host machine
                  .withFileSystemBind(objMgr.getGroupFolder(groupName),
                          Maestro.ROOT_HOSTMOUNT, BindMode.READ_WRITE)
                  // and run the test + jacococ commands
                  .withCommand("bash", "-c", "./gradlew test jacocoRootReport; sleep 10s; echo DONE");

  report.buildAction( actionPerformed: "Run './gradlew test jacocoRootReport' in container");
  daemon.start();
```

You can monitor output

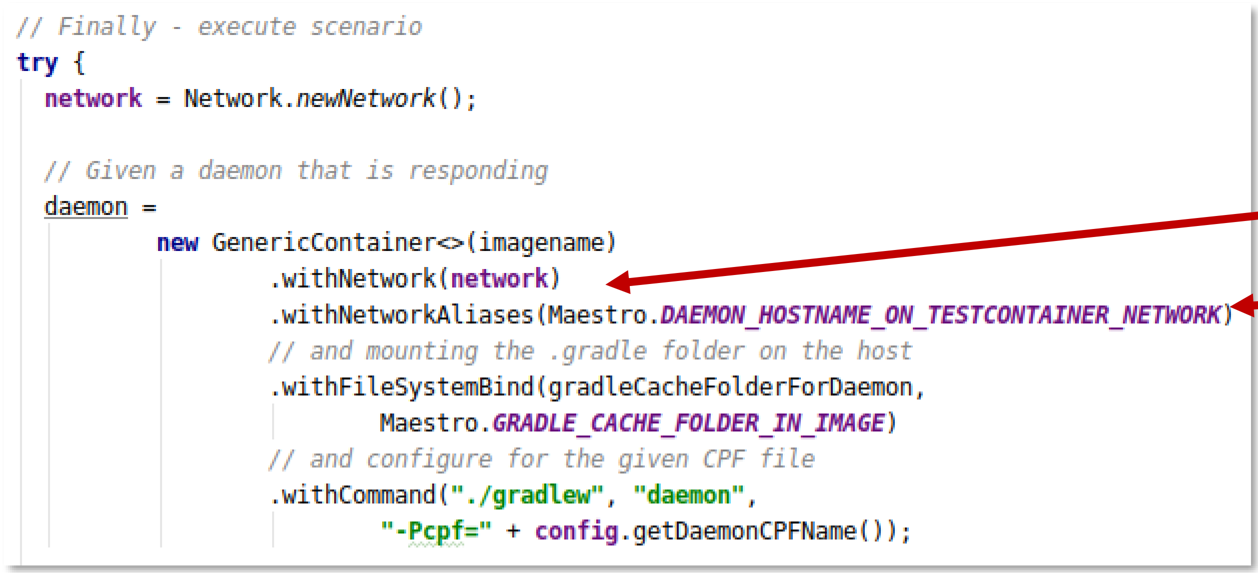
- By ‘consumers’



```
logger.info("method=assess, context=await-build-success");
try {
  WaitingConsumer consumer = new WaitingConsumer();
  daemon.followOutput(consumer, STDOUT);
  consumer.waitUntil(frame ->
                frame.getUtf8String().contains(Maestro.BUILD_SUCCESSFUL_STRING_TO_WAIT_FOR),
        maximumTimeToComplete, TimeUnit.SECONDS);
} catch (TimeoutException e) {
  report.buildErrorReport(e.getMessage(), errorCauseList,  quoteClientCmd: null,
          ruleExecuted: null,  cmdOutput: null,  daemonOutput: null);
  return false;
}
// Read through logs to extract the test summaries with numbers.
logger.info("methods=assses, context=extract-test-summary");
String daemonLogs = daemon.getLogs();
String testSummary = Util.extractTestSummaryFromLogs(daemonLogs);
report.buildAction( actionPerformed: "Extracting test report from daemon logs. Summary = " + testSummary);
logger.info("methods=assses, test-summary=" + testSummary);
```

You can control networks

- ‘docker network create’, and ‘—name (myname)’



```
// Finally - execute scenario
try {
  network = Network.newNetwork();

  // Given a daemon that is responding
  daemon =
          new GenericContainer<>(imagename)
                  .withNetwork(network)
                  .withNetworkAliases(Maestro.DAEMON_HOSTNAME_ON_TESTCONTAINER_NETWORK)
                  // and mounting the .gradle folder on the host
                  .withFileSystemBind(gradleCacheFolderForDaemon,
                          Maestro.GRADLE_CACHE_FOLDER_IN_IMAGE)
                  // and configure for the given CPF file
                  .withCommand("./gradlew", "daemon",
                          "-Pcpf=" + config.getDaemonCPFName());
```

Note: Only 16 networks available ☹. Unless you tweak Docker…

Failure Modes

- Sometimes TestContainers choke
  - ‘IllegalStateException: could not connect to ryuk’

- Do the following
  - **Restart your Docker service!**
    - Either restart your VM; or ‘sudo service docker restart’
  - ‘docker pull testcontainers/ryuk’
    - I have experience that ryuk (a support container) is not pulled and thus missing

Summary

- Out-of-process integration testing is tedious…
- TestContainers provides a robust way of making all the docker power available directly in JUnit…