CASSANDRA
SUMMIT 2016

Effective Testing in DSE

Predrag Knežević

predrag.knezevic@datastax.com

@pedjakknezevic

Why Taking Care?

- Automated testing for quality control of shipped product/service
- Number of tests and total testing times increase over time
- Shorter delivery cycles → continuous testing

- Run tests on each pre-merge check, but
- Keep feedback cycles short
- Ensure repeatable test execution anywhere

CASSANDRA SUMMIT2016

DSE Build Facts

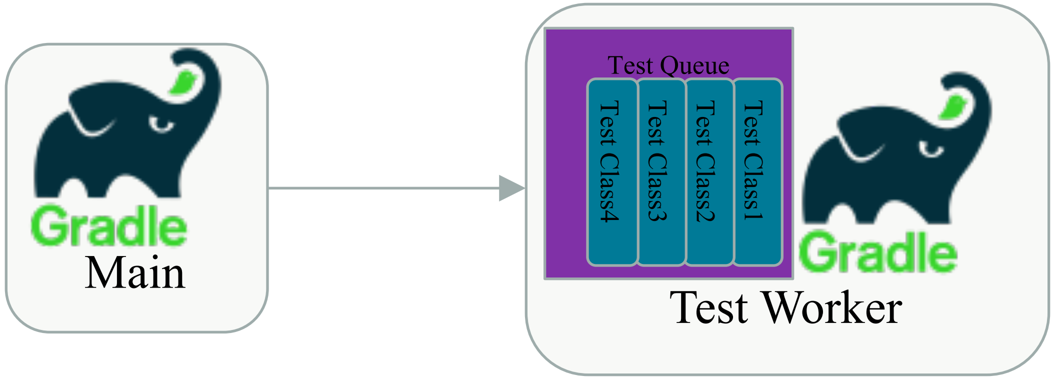
- Junit based test infrastructure
- December 2014 (DSE 4.6)
  - Ant based build system
  - ~5h for running all tests on Jenkins, with a rather complicated job layout
- July 2016 (DSE 4.7+)
  - Gradle based build system
  - 40-60mins for running all tests on Jenkins
  - 16 hours of total testing time
  - The number of tests doubled!
  - Repeatable test execution across all machines
  - Simple setup

Why Moving to Gradle?

- Built-in support for parallel test execution
- Readable - build scripts based on Groovy (easy learning for Java devs)
- Repeatable builds/environment setup across machines
- Powerful dependency management
- Sane conventions, but configurable when needed
- Easy project modularization
- Excellent Eclipse/IntelliJ support
- Easy extendable through plugins or additional Java/Groovy code living in the script or project
- All Ant tasks still available

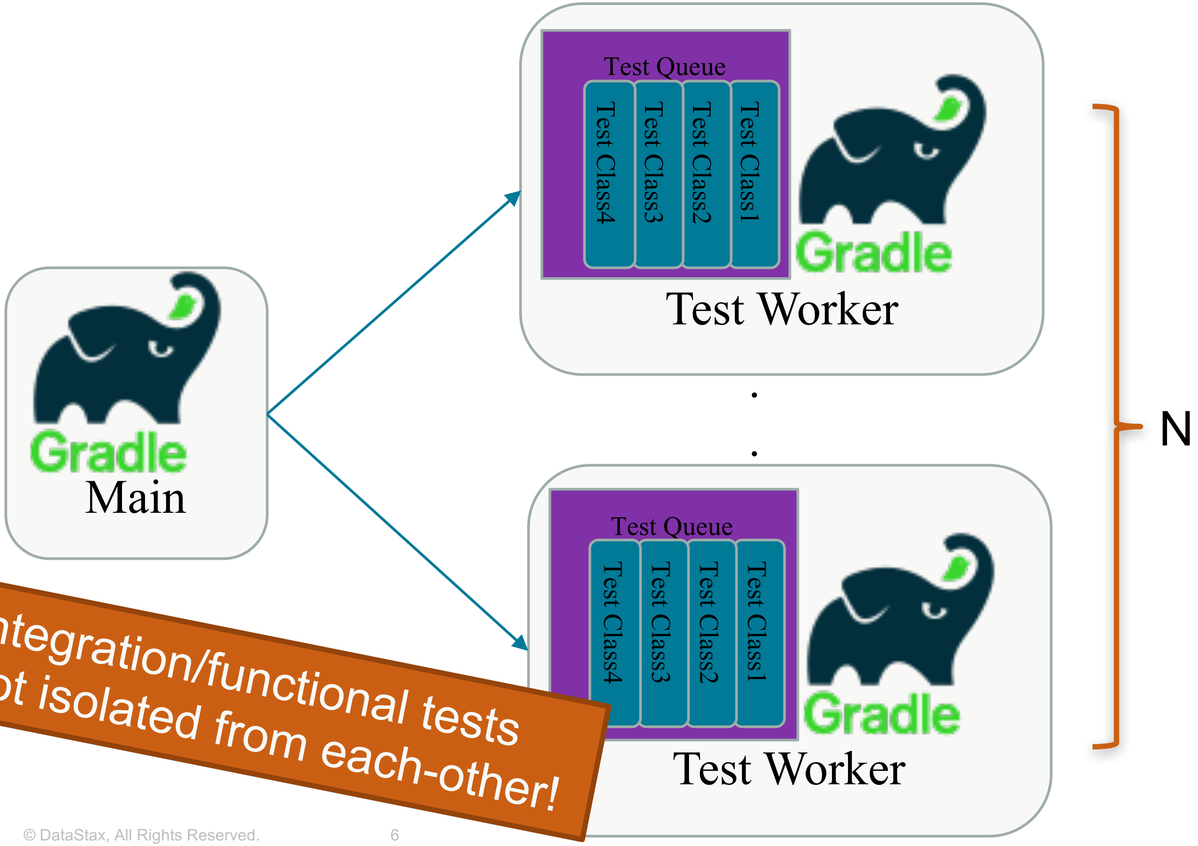
Running Tests

- All: `gradlew test`
- Single: `gradlew test -Dtest.single=FooTest`
- By default sequential execution
  - Low resources usage on modern multicore hardware
  - Long test round duration



CASSANDRA SUMMIT 2016

Parallel Test Execution



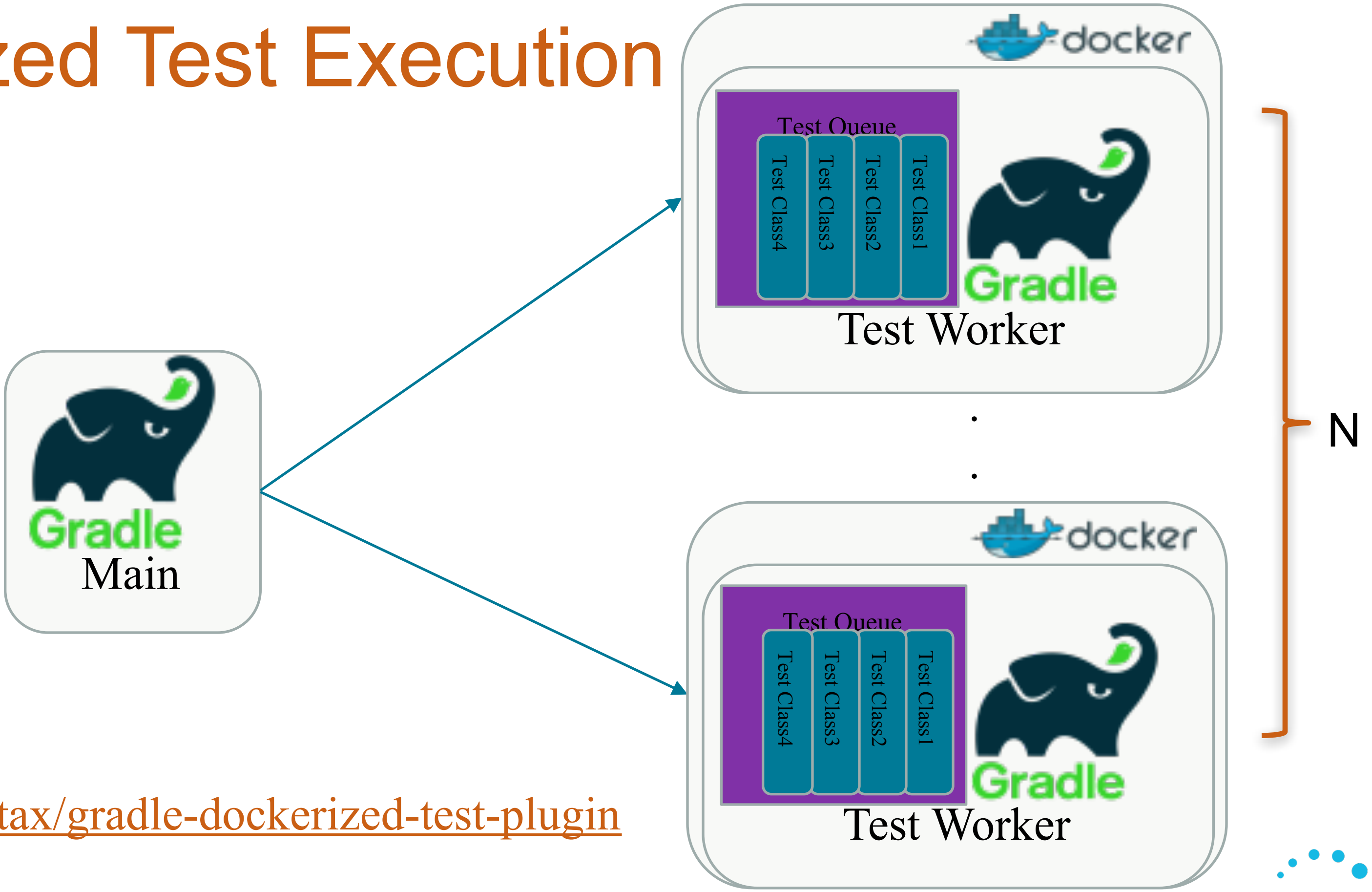
build.gradle

```
test {
  maxParallelForks = N
}
```

Integration/functional tests not isolated from each-other!

CASSANDRA SUMMIT 2016

Dockerized Test Execution



github.com/datastax/gradle-dockerized-test-plugin

CASSANDRA SUMMIT 2016

Dockerized Test Execution (2)

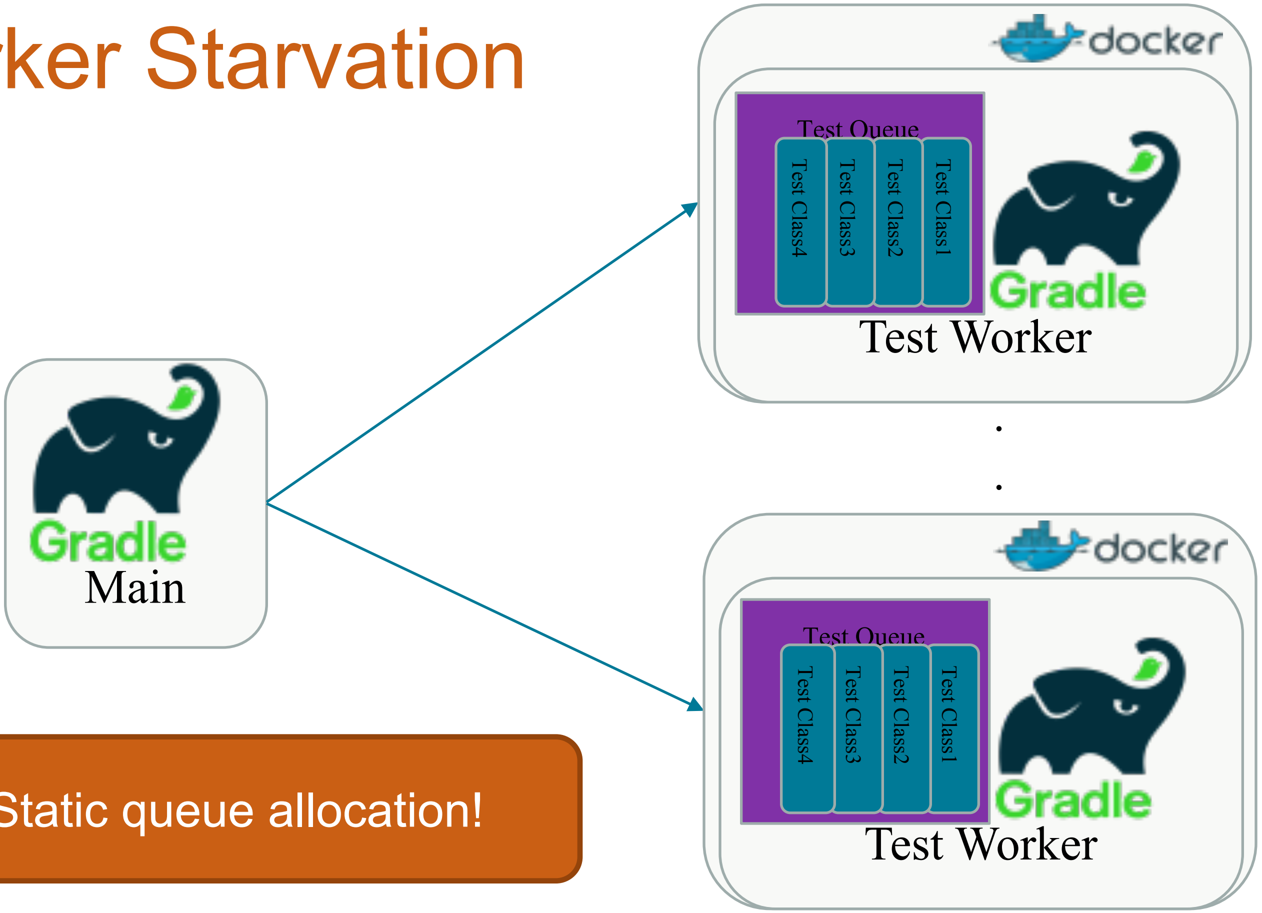
- Install Docker locally (native or boot2docker)
- No changes on production or test code required
- Test environment
  - Consistent across all machines (dev + CI)
    → no more “it works on my machine”
  - Managed as code (Dockerfiles) within project
  - Easy machine bootstraping
  - Fully isolated
- Easy testing against
  several and/or appropriate environment

build.gradle

```
test {
  maxParallelForks = N
  docker {
    image = ’test-image’
  }
}
```

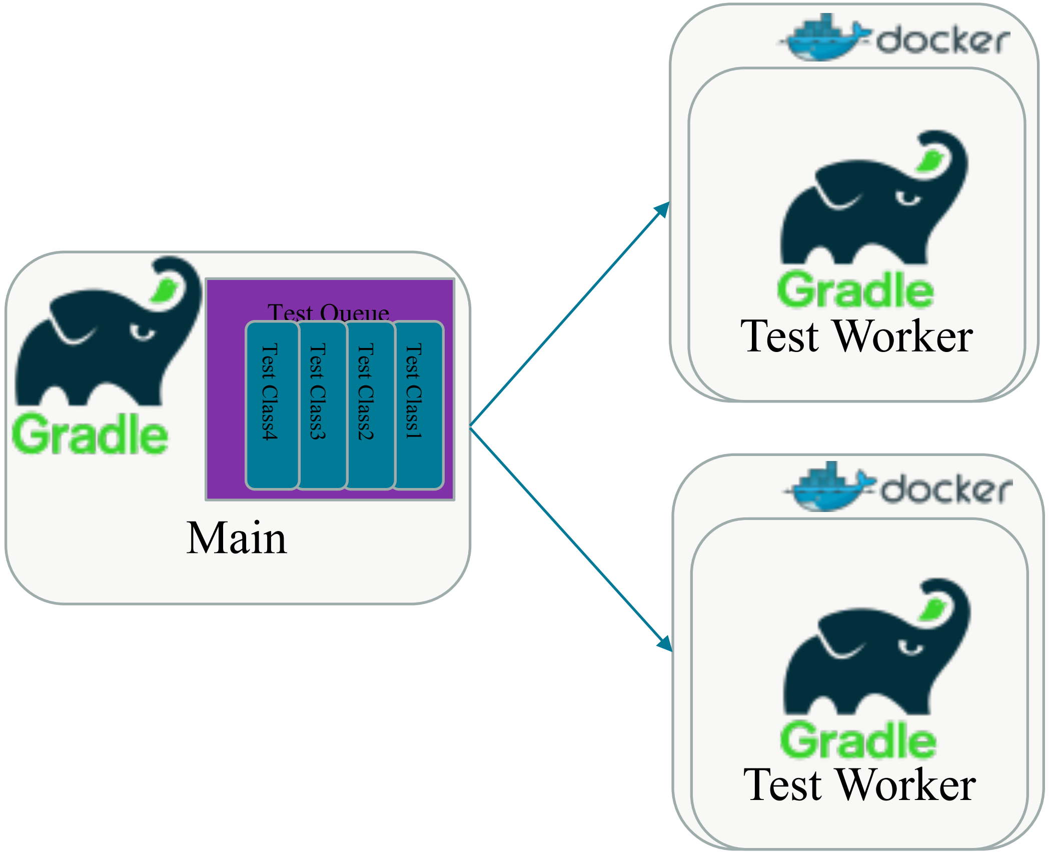
CASSANDRA SUMMIT2016

Worker Starvation



Static queue allocation!

Improved Queue Management

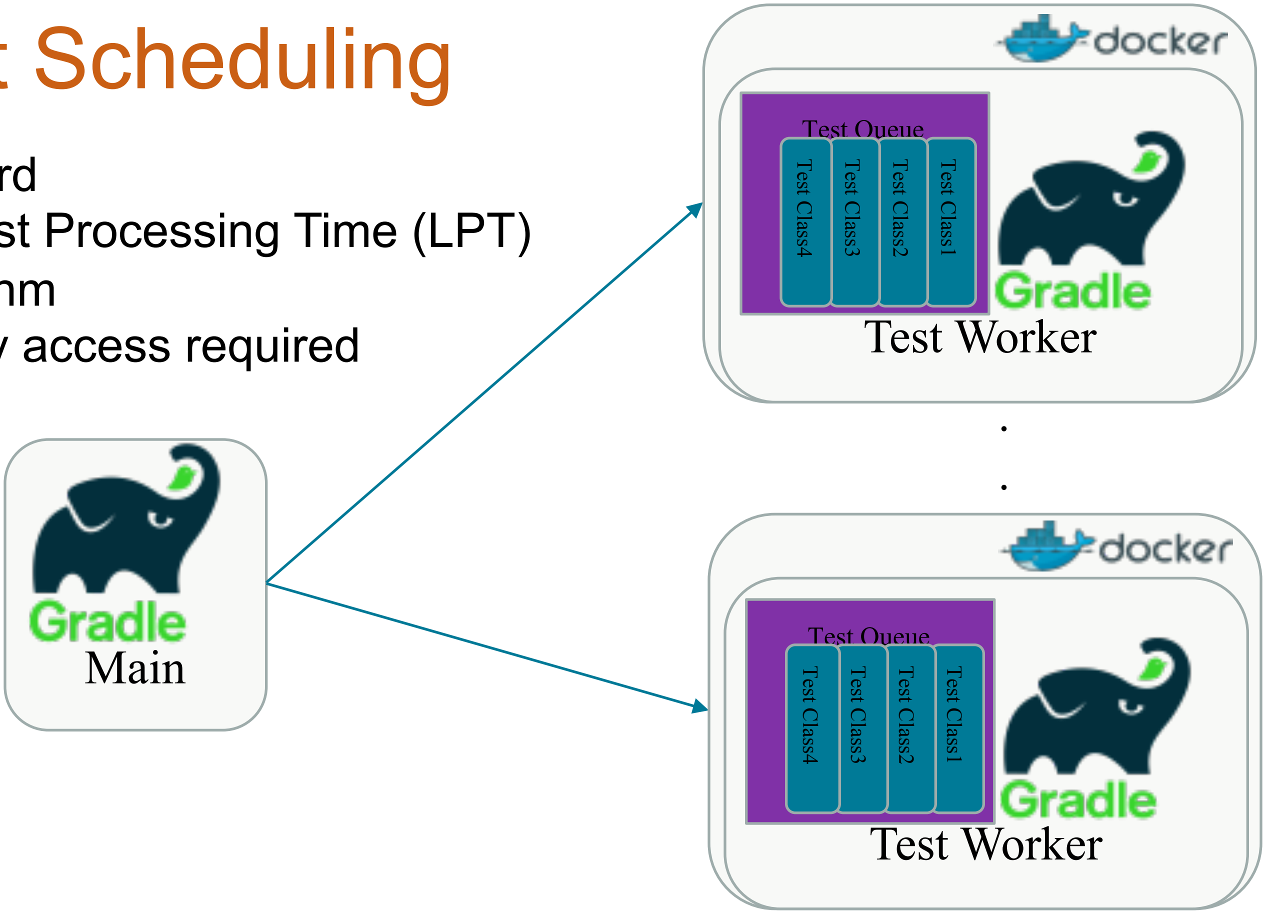


The duration of test round depends on the order of the test classes in the queue!

CASSANDRA SUMMIT 2016

Test Scheduling

- NP-hard
- Longest Processing Time (LPT) algorithm
- History access required

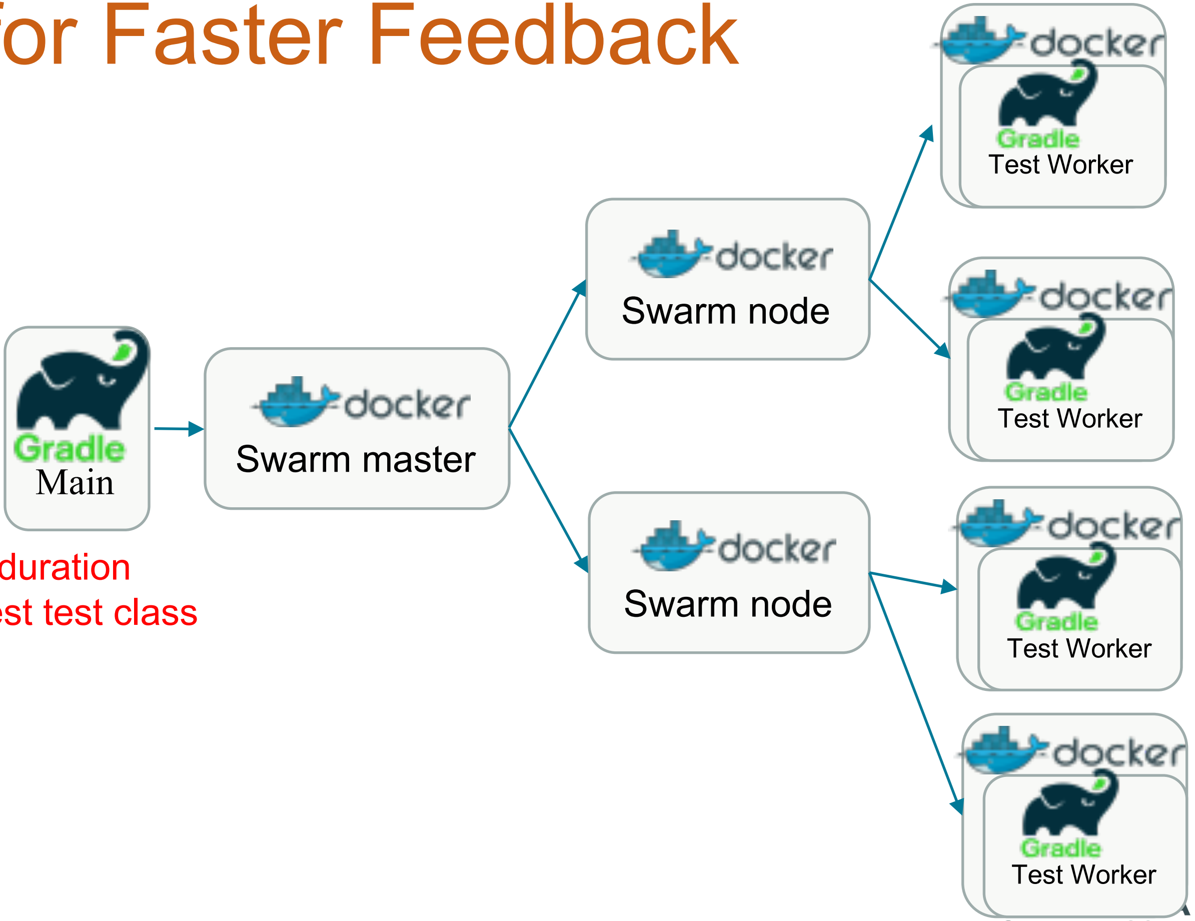


CASSANDRA SUMMIT2016

More Workers for Faster Feedback

- Powerful multi-core machine
- Docker Swarm
  - Virtual Docker engine
  - Test workers run on a Swam node
  - Cluster can shrink or grow

- Lower bound for the test round duration is equal to the duration of slowest test class



Split Slow Test Classes for More Throughput

- Manual
  - Group by common fixture
  - Extreme: one test per class
- Bytecode manipulations (auto split annotated class)
- Grouping tests classes into JUnit suites forces their sequential execution!

```
class FooTest {
@Test
void bar1() {
}

@Test
void bar2() {
}

}
```

```
class Foo1Test {
@Test
void bar1() {
}
}

class Foo2Test {
@Test
void bar2() {
}

}
```

CASSANDRA SUMMIT2016

No Embedded DSE Nodes

- Running cluster within single test worker JVM possible only by using separate classpath loaders
  - Still requires a good amount of hacking to make it work decently
  - Node shutdowns might still be problematic
    - Thread can hang around
    - Some objects cannot be garbage collected → memory leaks
- Standalone nodes enable reusage across test classes

Remote Code Execution

- MobilityRPC library (http://github.com/npgall/mobility-rpc)
- Remote JUnit Testrunner (http://github.com/datastax/remote-junit-runner)
  - Useful for integration tests requiring application context, but application cannot be easily embedded into test JVM
  - Support any existing JUnit runner on the remote side (Parametrized, Spock, Suites)

```
@RunWith(Remote.class)
public class RemoteTest {

  @Test
  public void foo() {
    // test
  }
}
```

Injecting Faults

- JBoss Byteman (http://byteman.jboss.org)
- Inject at runtime
  - Exceptions
  - Delays
  - Arbitrary side-effects
- Enables/simplifies testing of edge/uncommon cases

Logging

- Logback (http://logback.qos.ch/) based infrastructure
- Log file per a test case
  - Send all log messages to the single logback server asynchronously (reactor-logback adapter) keeping DSE nodes responsive
  - Route log statements to the proper file using SiftingAdapter and appropriate discriminator
  - Use JUnit rules to mark the beginning and the end of a test and propagate this information to the logback discriminator
  - Write thread dumps in case of a failure
- Scan log files for known issues and fail tests if they occur (e.g. Netty memory leaks)
- Turn DEBUG messages on to help later digging in a case of failure

Log Event Sender

```
<appender name="SOCKET" class="com.datastax.bdp.logback.SocketAppender">
    <remoteHost>${logbackServer}</remoteHost>
    <port>12345</port>
    <reconnectionDelay>1 seconds</reconnectionDelay>
    <nodeId>${nodeid}</nodeId>
    <eventDelayLimit>120 seconds</eventDelayLimit>
</appender>

<appender name="ASYNC" class="reactor.logback.AsyncAppender">
    <includeCallerData>true</includeCallerData>
    <appender-ref ref="SOCKET"/>
</appender>

<root level="DEBUG">
    <appender-ref ref="ASYNC"/>
</root>
```

CASSANDRA SUMMIT2016

Log Event Router

```
<appender name="SIFT" class="ch.qos.logback.classic.sift.SiftingAppender">
    <discriminator class="com.datastax.bdp.test.ng.LogFileDiscriminator">
        <key>TEST_LOG_FILE</key>
        <defaultValue>system.log</defaultValue>
    </discriminator>
    <sift>
        <appender name="FILE-${TEST_LOG_FILE}" class="ch.qos.logback.core.FileAppender">
            <filter class="com.datastax.bdp.test.ng.ForbiddenLogEventsDetector">
              <logFile>${TEST_LOG_FILE}</logFile>
              <detector class="com.datastax.bdp.test.ng.NettyLeakDetector"/>
            </filter>
            <encoder>
                <pattern>%X{nodeid} %5p [%t] %d{ISO8601} %F \(line %line\) %m%n</pattern>
                <immediateFlush>false</immediateFlush>
            </encoder>
            <file>${LOG_DIR}/${TEST_LOG_FILE}</file>
            <append>true</append>
        </appender>
    </sift>
</appender>
```

Test Start/End Detection

```
@Rule
public TestRule watcher = new TestWatcher() {

    private String logFile;

    @Override
    protected void starting(Description description) {
        logFile = description.getClassName()+"/"+description.getMethodName()+".log";
        logToFile(logFile);
    }

    protected void failed(Throwable e, Description description) {
        threadDumpLogger.error("Test {}.{} failed, thread dump:\n{}\n", description.getClassName(),
                description.getMethodName(), getThreadDumps());
        logToFile(description.getClassName()+"/after.log");
    }

    protected void finished(Description description) {
        logToFile(description.getClassName()+"/after.log");
        ForbiddenLogEventsDetector.checkForIssues(logFile);
    }
};
```

CASSANDRA SUMMIT 2016

Resources

- Gradle (gradle.org)
- Docker (docker.com)
- Gradle Dockerized Test Plugin (github.com/datastax/gradle-dockerized-test-plugin)
- MobilityRPC (http://github.com/npgall/mobility-rpc)
- Remote JUnit Testrunner (http://github.com/datastax/remote-junit-runner)
- JBoss Byteman (http://byteman.jboss.org)
- Logback (http://logback.qos.ch/)
- Project Reactor Addons (github.com/reactor/reactor-addons), Logback adapter

CASSANDRA SUMMIT2016

CASSANDRA SUMMIT 2016

Thank you!

Questions?