OBJECT
COMPUTING
HOME TO GRAILS & MICRONAUT

WEBINAR

Micronaut Testing Best Practices

MICRONAUT

Iván López - @ilopmar

Senior Software Engineer

Iván López (@ilopmar)

OBJECT
COMPUTING
HOME TO GRAILS & MICRONAUT

Why testing?

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

Unit tests

vs.

Integration tests

vs.

E2E tests

OBJECT
COMPUTING
HOME TO GRAILS & MICRONAUT

What About Micronaut?

Tests in Micronaut

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Avoid artificial separation of tests because of slow start and memory consumption
- Micronaut is a test agnostic framework
- **Spock**, JUnit 4, **JUnit 5**, Spek 2, **Kotlin Test**
- https://launch.micronaut.io

Spock

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
plugins {
    id "groovy"
    ...
}

dependencies {
    ...
    testImplementation("org.spockframework:spock-core") {
        exclude group: "org.codehaus.groovy", module: "groovy-all"
    }
    testImplementation "io.micronaut:micronaut-inject-groovy"
    testImplementation "io.micronaut.test:micronaut-test-spock"

    ...
}
```

JUnit 5

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
dependencies {
    ...
    testAnnotationProcessor "io.micronaut:micronaut-inject-java"

    testImplementation "org.junit.jupiter:junit-jupiter-api"
    testImplementation "io.micronaut.test:micronaut-test-junit5"

    testRuntimeOnly "org.junit.jupiter:junit-jupiter-engine"
}

test {
    useJUnitPlatform()
}
```

Kotlin Test

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
dependencies {
    ...
    kaptTest "io.micronaut:micronaut-inject-java"

    testImplementation "io.kotlintest:kotlintest-runner-junit5:3.3.2"
    testImplementation "io.micronaut.test:micronaut-test-kotlintest"
    testImplementation "io.mockk:mockk:1.9.3"
}

test {
    useJUnitPlatform()
}
```

Kotlin Test

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

Different approaches for tests

1.- DIY: Manual, old approach

2.- Micronaut Test

3.- DIY on Steroids

DIY, manual, old approach

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Start/stop manually `ApplicationContext` & `EmbeddedServer`
- Control about how everything is created
- Verbose

DIY, manual, old approach

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
public class HelloControllerTest {

    private static EmbeddedServer server;
    private static HttpClient client;

    @BeforeClass
    public static void setupServer() {
        server = ApplicationContext
                    .run(EmbeddedServer.class);
        client = server
                    .getApplicationContext()
                    .createBean(HttpClient.class,server.getURL());
    }

    @AfterClass
    public static void stopServer() {
        if (server ≠ null) {
            server.stop();
        }
        if (client ≠ null) {
            client.stop();
        }
    }
```

```
    @Test
    public void testHello() throws Exception {
        HttpRequest request = HttpRequest.GET("/hello");
        String body = client.toBlocking().retrieve(request);

        assertNotNull(body);
        assertEquals("Hello World", body);
    }
}
```

DIY, manual, old approach

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
class HelloControllerSpec extends Specification {

    @Shared
    @AutoCleanup
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer)

    @Shared
    @AutoCleanup
    RxHttpClient client = embeddedServer.applicationContext.createBean(RxHttpClient,
embeddedServer.getURL())

    void "test hello world response"() {
        when:
        HttpRequest request = HttpRequest.GET('/hello')
        String rsp = client.toBlocking().retrieve(request)

        then:
        rsp == "Hello World"
    }

}
```

DIY, manual, old approach

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

Pros

- Total control

Cons

- Repetitive setup
- Lots of copy-paste
- JUnit pain
- More code “around” test than testing

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

Different approaches for tests

1.- DIY: Manual, old approach

2.- Micronaut Test

3.- DIY on Steroids

Micronaut Test

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Extensions for Spock, JUnit 5 and Kotlin Test
- Automatically start and stop the server for the scope of a test suite
- Allow dependency injection into a test instance
- `@MockBean`: Define mock beans that replace existing beans for the scope of the test
- https://micronaut-projects.github.io/micronaut-test/latest/guide/index.html

Micronaut Test

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Micronaut Test **DOES NOT MOCK** any part of Micronaut
- Running a test with `@MicronautTest` is running your **REAL** application
- `TestPropertyProvider`

```
@MicronautTest
public class BookControllerTest implements TestPropertyProvider {

    @Nonnull
    @Override
    public Map<String, String> getProperties() {
        return Map.of("foo.bar", "abcd");
    }

    ...
}
```

- `@Property(name="foo.bar", value="abcd")`

Micronaut Test

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
public class HelloControllerTest {

    private static EmbeddedServer server;
    private static HttpClient client;

    @BeforeClass
    public static void setupServer() {
        server = ApplicationContext
                .run(EmbeddedServer.class);
        client = server
                .getApplicationContext()
                .createBean(HttpClient.class,server.getURL());
    }

    @AfterClass
    public static void stopServer() {
        if (server != null) {
            server.stop();
        }
        if (client != null) {
            client.stop();
        }
    }

    @Test
    public void testHello() throws Exception {
        HttpRequest request = HttpRequest.GET("/hello");
        String body = client.toBlocking().retrieve(request);

        assertNotNull(body);
        assertEquals("Hello World", body);
    }
}
```

@MicronautTest

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
@MicronautTest
public class HelloControllerTest {

    @Inject
    @Client("/")
    RxHttpClient client;

    @Test
    public void testHello() {
        HttpRequest<String> request = HttpRequest.GET("/hello");
        String body = client.toBlocking().retrieve(request);

        assertNotNull(body);
        assertEquals("Hello World", body);
    }
}
```

@MicronautTest

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
class HelloControllerSpec extends Specification {

    @Shared
    @AutoCleanup
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer)

    @Shared
    @AutoCleanup
    RxHttpClient client = embeddedServer.applicationContext.createBean(RxHttpClient,
embeddedServer.getURL())

    void "test hello world response"() {
        when:
        HttpRequest request = HttpRequest.GET('/hello')
        String rsp  = client.toBlocking().retrieve(request)

        then:
        rsp == "Hello World"
    }

}
```

@MicronautTest

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
@MicronautTest
class HelloControllerSpec extends Specification {

    @Inject
    @Client("/")
    RxHttpClient client

    void "test hello world response"() {
        when:
        HttpRequest request = HttpRequest.GET('/hello')
        String rsp = client.toBlocking().retrieve(request)

        then:
        rsp == "Hello World"
    }
}
```

@MicronautTest

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Pros

- Automatic start/stop `EmbbededServer` & `ApplicationContext`
- Automatic Bean Injection
- `@MockBean`
- `TestPropertyProvider`
- `@Property`

Cons

- Everything is automatic → less control
- `@MicronautTest` everywhere
- ES/AC started even when not necessary

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

Different approaches for tests

1.- DIY: Manual, old approach

2.- Micronaut Test

3.- DIY on Steroids

HOME TO GRAILS & MICRONAUT

We ❤ Spock

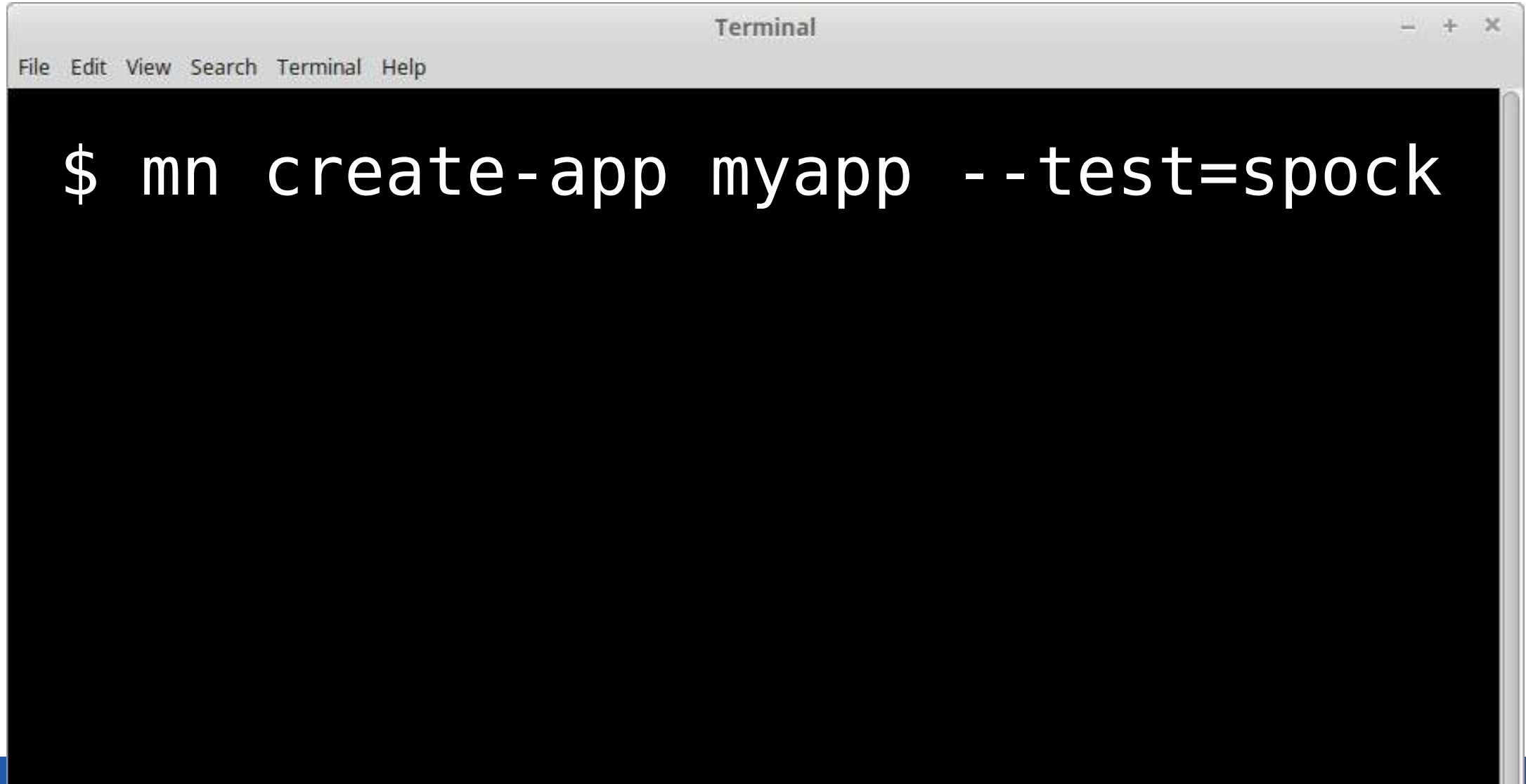
DIY on Steroids

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

- Spock
- Parent `abstract` classes
- Groovy traits
- Groovy getters
- Common configuration
- Data fixtures

Java code, Spock tests



ApplicationContextSpecification

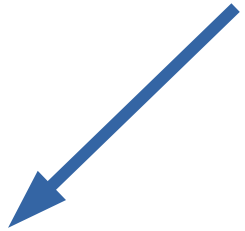
HOME TO GRAILS & MICRONAUT

```
abstract class ApplicationContextSpecification extends Specification
                                                implements ConfigurationFixture {

    @AutoCleanup
    @Shared
    ApplicationContext applicationContext = ApplicationContext.run(configuration)

}
```

ConfigurationFixture

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
trait ConfigurationFixture {

    Map<String, Object> getConfiguration() {
        Map<String, Object> m = [:]

        if (specName) {
            m['spec.name'] = specName
        }

        m
    }

    String getSpecName() {
        null
    }
}
```



EmbeddedServerSpecification

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
abstract class EmbeddedServerSpecification extends Specification implements ConfigurationFixture {

    @AutoCleanup
    @Shared
    EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer, configuration)

    @AutoCleanup
    @Shared
    ApplicationContext applicationContext = embeddedServer.applicationContext

    @AutoCleanup
    @Shared
    HttpClient httpClient = applicationContext.createBean(HttpClient, embeddedServer.URL)

    BlockingHttpClient getClient() {
        httpClient.toBlocking()
    }
}
```

Example ApplicationContext


```
class RouteServiceImplSpec extends ApplicationContextSpecification implements SensorFixture {

    @Subject
    @Shared
    RouteService routeService = applicationContext.getBean(RouteService)

    void 'fetch the route related to a shipment'() {
        given: 'a sensor-shipment'
        saveSensor('1709286', shipmentId)

        when: 'fetching the route related to the shipment'
        Optional<Route> optRoute = routeService.fetchRouteByShipmentId(shipmentId)

        then: 'it exists'
        noExceptionThrown()
        optRoute
        optRoute.isPresent()

        ...
    }
}
```

Example EmbeddedServer


```
class OrganizationControllerSpec extends EmbeddedServerSpecification {

    void 'it is possible to fetch all organizations with basic auth'() {
        given: 'an http request with auth to get the organizations'
        HttpRequest request = HttpRequest.GET('/organizations') basicAuth('abc.ware', 'foo')

        when: 'executing the http call'
        HttpResponse<List<Organization>> response =
                                              client.exchange(request, Argument.listOf(Organization))

        then: 'status is OK'
        response.status() == HttpStatus.OK

        when: 'fetching the organizations'
        List<Organization> organizations = response.body()

        then: 'they are correct'
        organizations
        organizations.size() == 1
        organizations[0].name == 'name'
        organizations[0].id == 'id'
        organizations[0].address == 'address'
    }
}
```

DIY on Steroids

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- What about a Data layer?
- Let’s add Micronaut Data (or JPA/Hibernate, or MongoDB, or ...)
- Test “Leakage”

LeakageDetector

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
trait RepositoriesFixture {

    abstract ApplicationContext getApplicationContext()

    SensorRepository getSensorRepository() {
        applicationContext.getBean(SensorRepository)
    }

    ...
}
```

```
abstract class ApplicationContextSpecification ... {

    @AutoCleanup
    @Shared
    ApplicationContext applicationContext = ApplicationContext.run(configuration)

}
```

LeakageDetector

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
trait LeakageDetector extends RepositoriesFixture {

    boolean hasLeakage() {
        if (sensorRepository.count() > 0) {
            println "there are still sensors"
        }
        ...

        sensorRepository.count() > 0 || ...
    }
}
```

```
trait RepositoriesFixture {
    SensorRepository getSensorRepository() {
        applicationContext.getBean(SensorRepository)
    }
}
```

LeakageDetector


```
abstract class ApplicationContextSpecification ... implements LeakageDetector ... {
    ...

    def cleanup() {
        assert !hasLeakage()
    }
}
```

```
abstract class EmbeddedServerSpecification ... implements LeakageDetector ... {
    ...

    def cleanup() {
        assert !hasLeakage()
    }
}
```

LeakageDetector

Tests failed: 1 of 1 test – 1 s 249 ms

```
Condition not satisfied:

!hasLeakage()
||
|true
false

    at com.                    ApplicationContextSpecification.cleanup(ApplicationContextSpecification.groovy:20)
```

Database for the tests

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- H2 by default
- H2 for production?
- Use same database for production and tests
- Excuses:
    - “I need to setup the db locally”
    - “I need to install it also in my CI environment”
    - “I need to...”

Stop making excuses for not testing properly!

Testcontainers

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Testcontainers

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Dependencies

```
testImplementation "org.testcontainers:spock:1.14.3"
testImplementation "org.testcontainers:mysql:1.14.3"
```

```
testImplementation "org.testcontainers:testcontainers:1.14.3"
testImplementation "org.testcontainers:junit-jupiter:1.14.3"
testImplementation "org.testcontainers:mysql:1.14.3"
```

- Annotate test class with `@Testcontainers`

- But I don’t want to put `@Testcontainers` everywhere...

Testcontainers

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
class MySQL {
    static MySQLContainer mysqlContainer

    static init() {
        if (mysqlContainer == null) {
            mysqlContainer = new MySQLContainer()
                .withDatabaseName('mydb')
                .withUsername('myuser')
                .withPassword('mypasswd')
            mysqlContainer.start()
        }
    }
}
```

Testcontainers

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
trait MySQLContainerFixture {

    Map<String, Object> getMySQLConfiguration() {
        if (MySQL.mysqlContainer == null || !MySQL.mysqlContainer.isRunning()) {
            MySQL.init()
        }
        [
            'datasources.default.url'      : MySQL.mysqlContainer.getJdbcUrl(),
            'datasources.default.password': MySQL.mysqlContainer.getPassword(),
            'datasources.default.username': MySQL.mysqlContainer.getUsername(),
        ]
    }
}
```

Testcontainers

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
trait ConfigurationFixture {

    Map<String, Object> getConfiguration() {
        Map<String, Object> m = [:]

        if (specName) {
            m['spec.name'] = specName
        }

        m
    }




}
```

Testcontainers

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
trait ConfigurationFixture implements MySqlFixture {

    Map<String, Object> getConfiguration() {
        Map<String, Object> m = [:]

        if (specName) {
            m['spec.name'] = specName
        }

        m+= mySqlConfiguration

        m
    }


}
```

Mocking collaborators

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

- Always use interfaces
- `@Singleton` & `@Primary`
- `@Requires(property = 'spec.name', value = 'xxxxxxSpec')`

Mocking collaborators

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
class HandoverControllerStartHandoverSpec extends EmbeddedServerSpecification implements HandoverFixture {

    @Override
    String getSpecName() {
        'HandoverControllerStartHandoverSpec'
    }

    void 'test start handover works and do not throws exception'() {
        when:
        HttpResponse rsp = client.exchange(
            HttpRequest
                .POST('/shipments/1/handover', createStartHandoverRequest("2"))
                .basicAuth('abc.ware', 'foo')
        )

        then:
        rsp.status() == HttpStatus.NO_CONTENT
    }

    @Singleton
    @Primary
    @Requires(property = 'spec.name', value = 'HandoverControllerStartHandoverSpec')
    static class CustomStartHandoverService implements StartHandoverService {

        @Override
        void startHandover(@Valid @NotNull StartHandover startHandover) {
        }
    }
}
```

```
trait ConfigurationFixture {
    Map<String, Object> getConfiguration() {
        Map<String, Object> m = [:]

        if (specName) {
            m['spec.name'] = specName
        }

        m
    }
}
```

Global Mocking

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Mocks for all the tests

- Option to enable/disable them per test

Global Mocking

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
@Primary
@Singleton
@Requires(env = Environment.TEST)
@Requires(property = 'mockShipmentSecurityService', value = 'true')
class MockShipmentSecurityService implements ShipmentSecurityService {

    @Override
    boolean canUserAccessToShipment(@NotBlank String username, @NotBlank String shipmentId) {
        true
    }

    @Override
    boolean canUserAccessToShipment(@NotBlank String username, @NotNull Shipment shipment) {
        true
    }
}
```

Global Mocking

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
trait ConfigurationFixture ... {

    ...

    Map<String, Object> getConfiguration() {
        Map<String, Object> m = [
            ...
            'mockShipmentSecurityService': mockShipmentSecurityServiceEnabled(),
        ]

        m
    }

    boolean mockShipmentSecurityServiceEnabled() {
        true
    }

    ...
}
```

What about 3rd party APIs?

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Start another `EmbeddedServer` (or even more)
- It has its own `ApplicationContext` and beans
- Controllers with the responses needed

What about 3rd party APIs?

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
abstract class EmbeddedServerSpecification ... {

    @Shared
    int blkServerPort = SocketUtils.findAvailableTcpPort()
    ...
}
```

```
abstract class BlkEmbeddedServerSpecification extends EmbeddedServerSpecification {

    @AutoCleanup
    @Shared
    EmbeddedServer blkEmbeddedServer = ApplicationContext.run(EmbeddedServer, configuration + [
        'micronaut.server.port': blkServerPort,
        'spec.name'            : blkSpecName,
    ])
}
```

What about 3rd party APIs?

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
trait ConfigurationFixture ... {

    abstract int getBlkServerPort()

    Map<String, Object> getConfiguration() {
        Map<String, Object> m = [:]

        if (specName) {
            m['spec.name'] = specName
        }
        if (blkSpecName) {
            m['eos.chain-api-base-url'] = "http://localhost:$blkServerPort"
        }
        m += mySQLConfiguration

        m
    }
    ...
}
```

What about 3rd party APIs?

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
class ShipmentsFetcherImplShipmentByIdServiceSpec extends BlkEmbeddedServerSpecification {

    @Subject
    @Shared
    ShipmentsFetcher shipmentsFetcher = applicationContext.getBean(ShipmentsFetcher)

    void 'it is possible to fetch a shipment by id'() {
        when:
        Optional<Shipment> optShipment = shipmentsFetcher.fetchShipmentById('1')

        then:
        noExceptionThrown()
        optShipment.isPresent()

        when:
        Shipment shipment = optShipment.get()

        then:
        shipment.id == '1'
        ...
    }

    ...
    ...
}
```

What about 3rd party APIs?

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
class ShipmentsFetcherImplShipmentByIdServiceSpec extends BlkEmbeddedServerSpecification {

    ...
    ...

    @Override
    String getBlkSpecName() {
        'ShipmentsFetcherImplShipmentByIdServiceSpec'
    }

    @Requires(property = 'spec.name', value = 'ShipmentsFetcherImplShipmentByIdServiceSpec')
    @Controller('/v1/chain/get_table_rows')
    @Secured(SecurityRule.IS_ANONYMOUS)
    static class GetTableRows {
        @Post
        String index() {
            '{"rows":[{"id":1,"serial_num":"4006381333931",...}],"more":false}'
        }
    }

}
```

Is this fast?

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Is this fast?

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

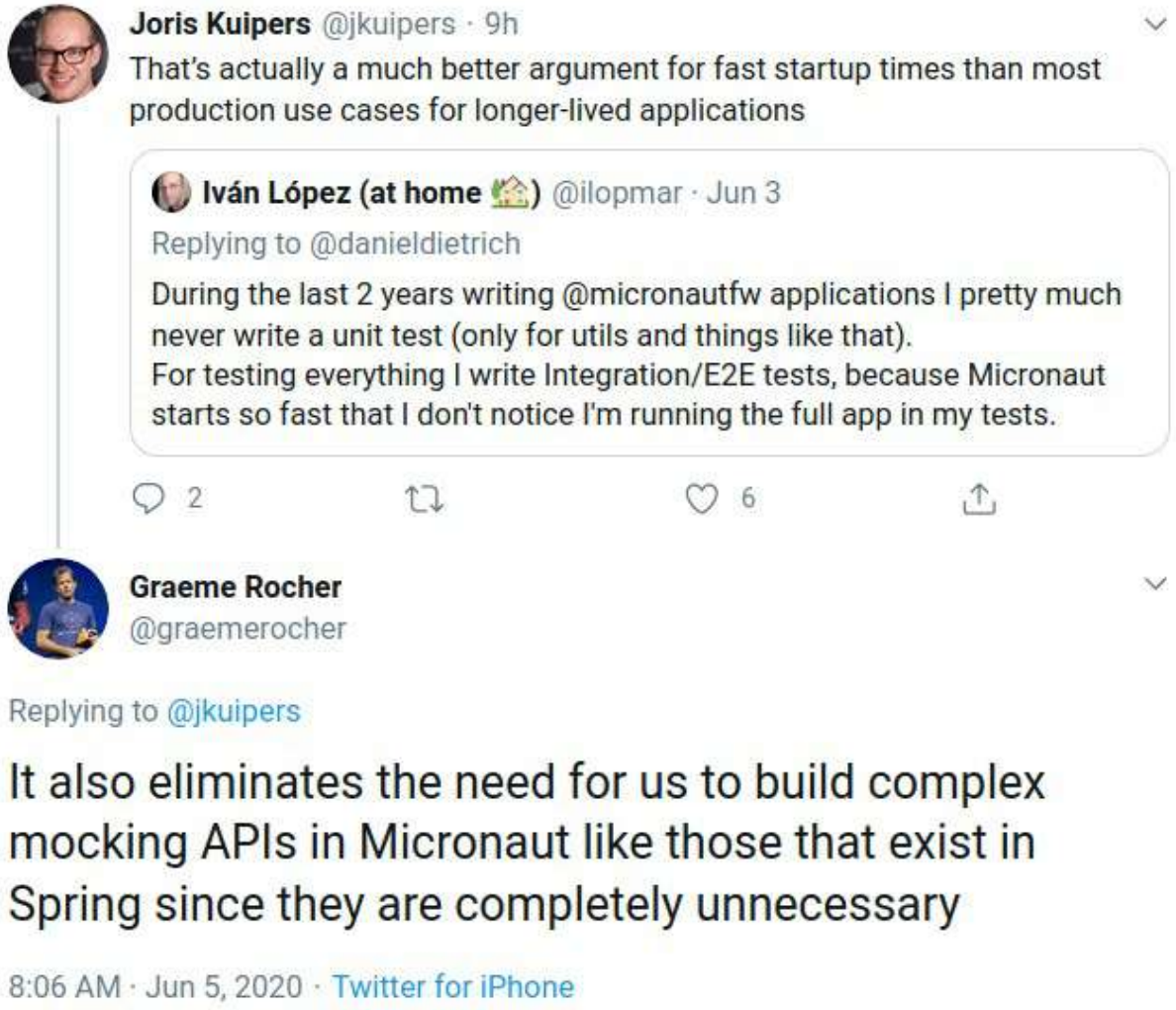
```
✔ Test Gradle Test Run :api:test Executed: 519/519/0/0

BUILD SUCCESSFUL in 1m 37s
```

- 15% Unit
- 62% Integration
- 23% E2E

85 EmbeddedServers started in 1m 37s!!!

Does it really matter?

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



Joris Kuipers @jkuipers · 9h

That's actually a much better argument for fast startup times than most production use cases for longer-lived applications

> **Iván López (at home 🏠)** @ilopmar · Jun 3
>
> Replying to @danieldietrich
>
> During the last 2 years writing @micronautfw applications I pretty much never write a unit test (only for utils and things like that).
> For testing everything I write Integration/E2E tests, because Micronaut starts so fast that I don't notice I'm running the full app in my tests.

2 6

Graeme Rocher
@graemerocher

Replying to @jkuipers

It also eliminates the need for us to build complex mocking APIs in Micronaut like those that exist in Spring since they are completely unnecessary

8:06 AM · Jun 5, 2020 · Twitter for iPhone

DIY on Steroids

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Pros

- Total control
- Mock collaborators
- Mock 3rd party APIs
- Add new dependencies easily: database, testcontainers,...
- Easy to read because of Groovy
- After using in 2 projects, works really great!

Cons

- Initial configuration
- Not `@Inject` but get beans from `ApplicationContext`

OBJECT
COMPUTING
HOME TO GRAILS & MICRONAUT

Conclusions

Conclusions

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Different ways to approach testing Micronaut apps
- Running Integration and E2E tests is really fast
- Nice developer experience
- As project evolves, easy to integrate everything in tests
- Spock FTW
- Groovy: traits, getters,…

Please, do write tests.

LEARN MORE ABOUT OCI EVENTS AND TRAINING

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Events:

- objectcomputing.com/events

Training:

- objectcomputing.com/training
- grailstraining.com
- micronauttraining.com

Or email 2GM@objectcomputing.com to schedule a custom training program for your team online, on site, or in our state-of-the-art, Midwest training lab.

Questions?

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

CONNECT WITH US

1+ (314) 579-0066

@micronautfw

info@micronaut.io