OBJECT COMPUTING

WEBINAR

Micronaut & GraalVM

Iván Lopez - @ilopmar

Iván López (@ilopmar)

OBJECT COMPUTING

GRAILS

MICRONAUT

Groovy MADRID

Groovy Users Group

OBJECT
COMPUTING

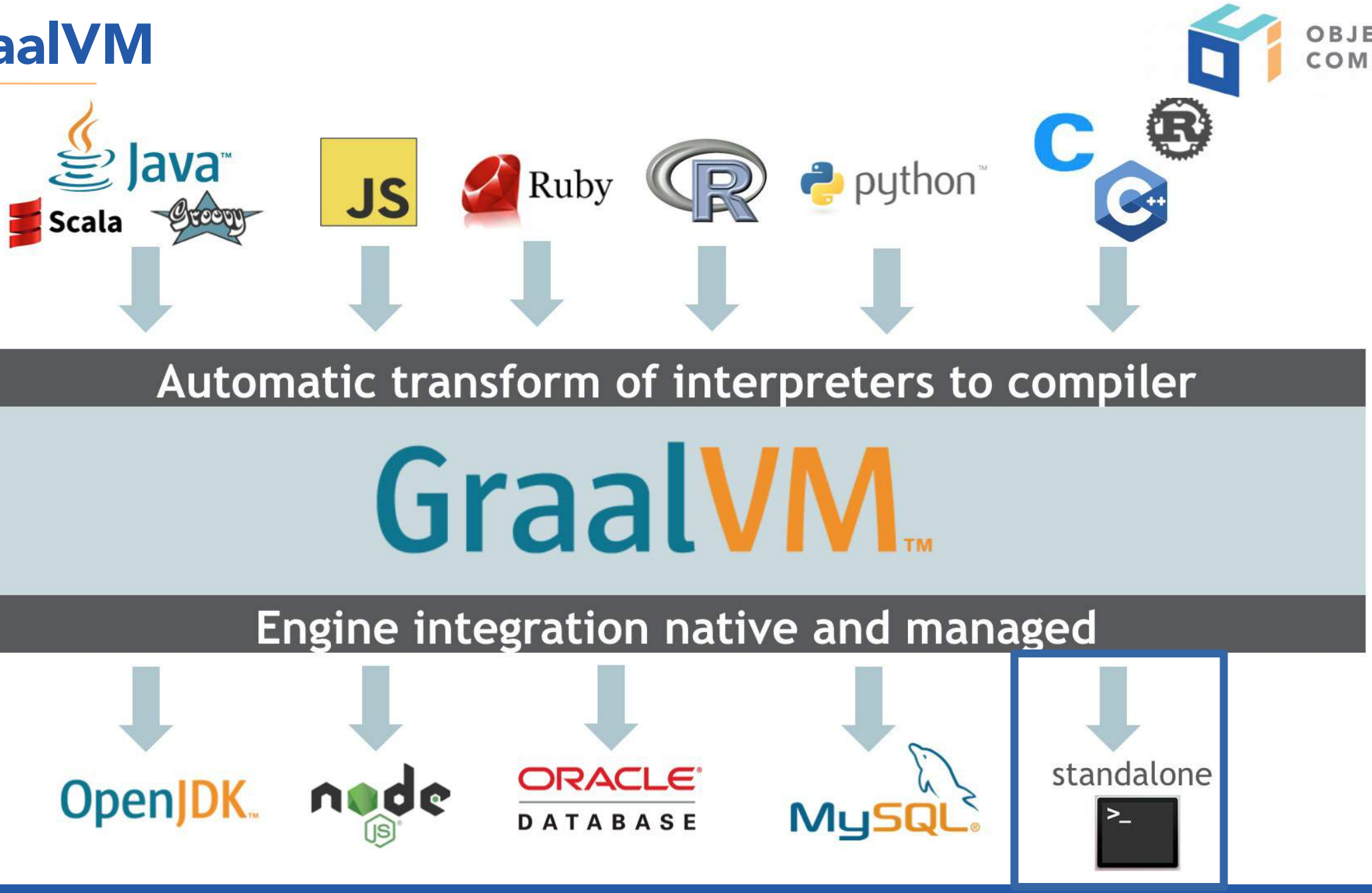
GraalVM™

GraalVM

- Universal Polyglot VM from Oracle
- JVM languages + Ruby, Python, JS, R
- Graal Compiler / JVMCI
- Truffle
- Substrate VM

GraalVM™

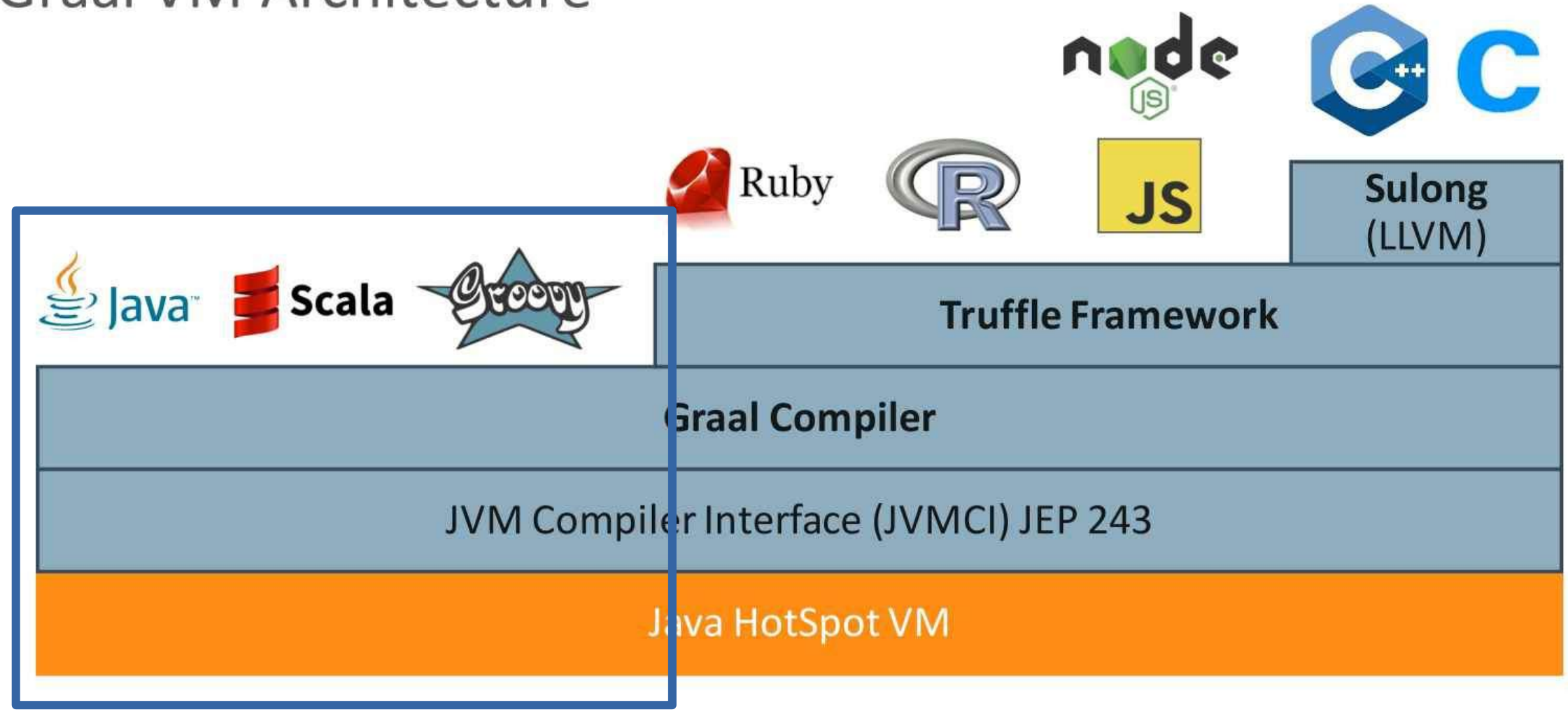
GraalVM


OBJECT COMPUTING
Java
Scala
Groovy
JS
Ruby
R
python
C
C++
Automatic transform of interpreters to compiler
GraalVM
Engine integration native and managed
OpenJDK
node
ORACLE
DATABASE
MySQL
standalone

GraalVM

OBJECT COMPUTING

Graal VM Architecture



GraalVM on production?

OBJECT COMPUTING

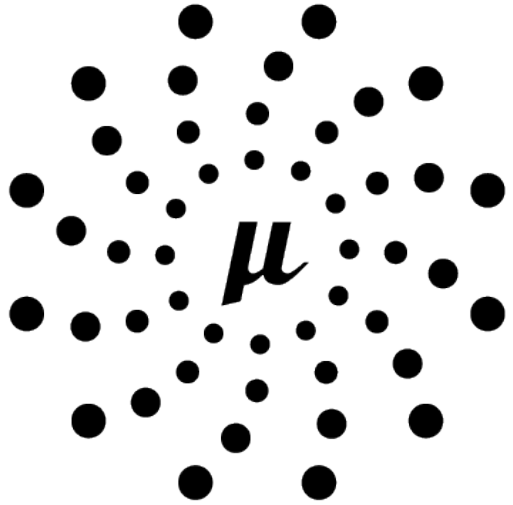
MICRONAUT

OBJECT COMPUTING

Micronaut

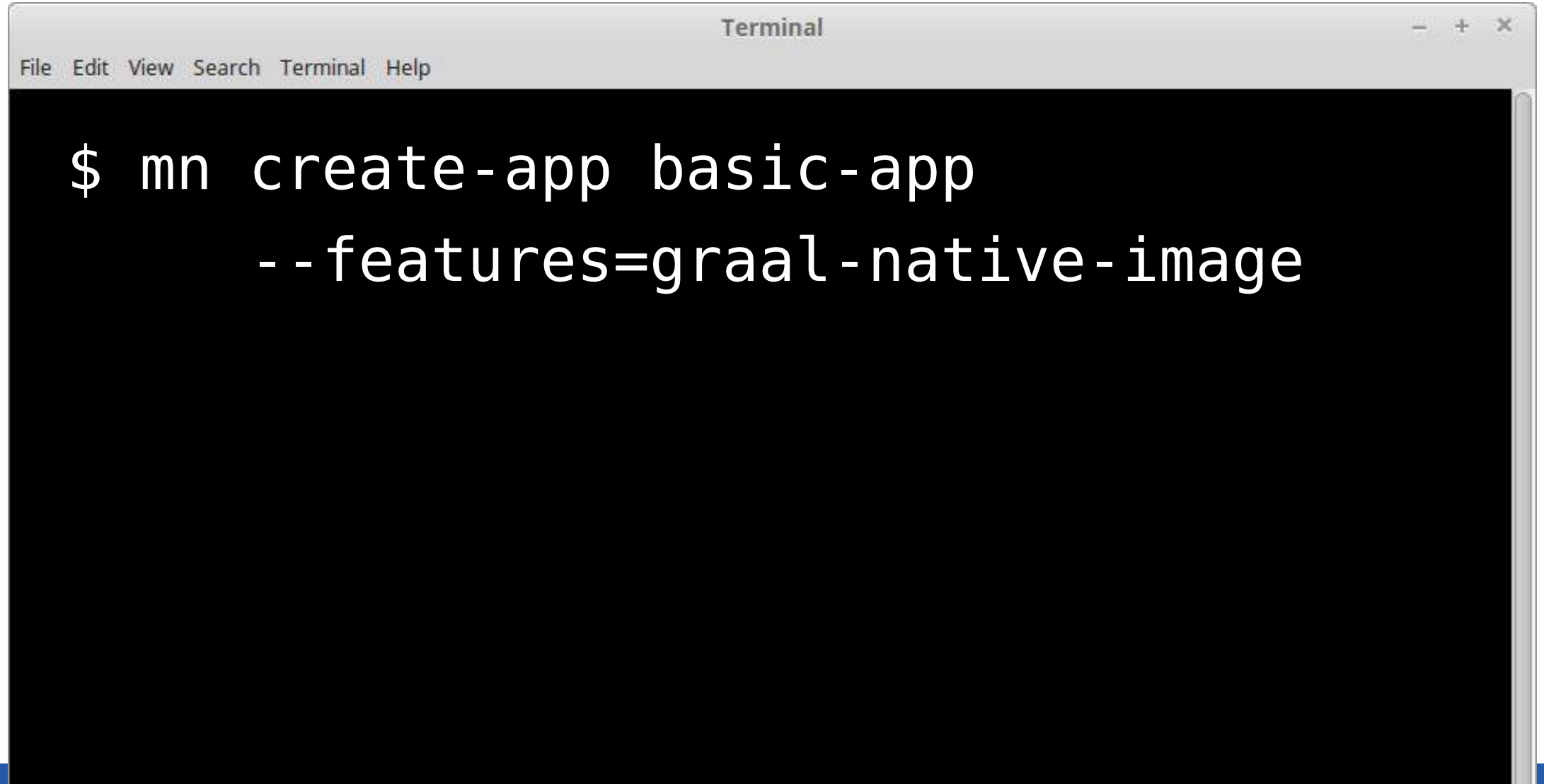
OBJECT COMPUTING

- Framework for microservices in the JVM
- Ultra-lightweight & reactive (Netty-based)
- Java, Groovy & Kotlin
- Ahead of Time compilation (AoT)
- No reflection, no runtime proxies
- Fast startup
- Low memory footprint
- Natively Cloud Native
- Support for GraalVM



MICRONAUT

Creating app with GraalVM support



Micronaut support for GraalVM

- Dependencies

```
annotationProcessor "io.micronaut:micronaut-graal"
compileOnly "com.oracle.substratevm:svm"
```

- Dockerfile

```
FROM oracle/graalvm-ce:1.0.0-rc15 as graalvm
COPY . /home/app/basic-app
WORKDIR /home/app/basic-app
RUN native-image --no-server -cp build/libs/basic-app-*.jar

FROM frolvlad/alpine-glibc
EXPOSE 8080
COPY --from=graalvm /home/app/basic-app .
ENTRYPOINT ["./basic-app"]
```

Micronaut support for GraalVM (II)

- native-image.properties

```
Args = -H:IncludeResources=logback.xml|application.yml \
       -H:Name=basic-app \
       -H:Class=example.micronaut.Application
```

Native image

OBJECT COMPUTING

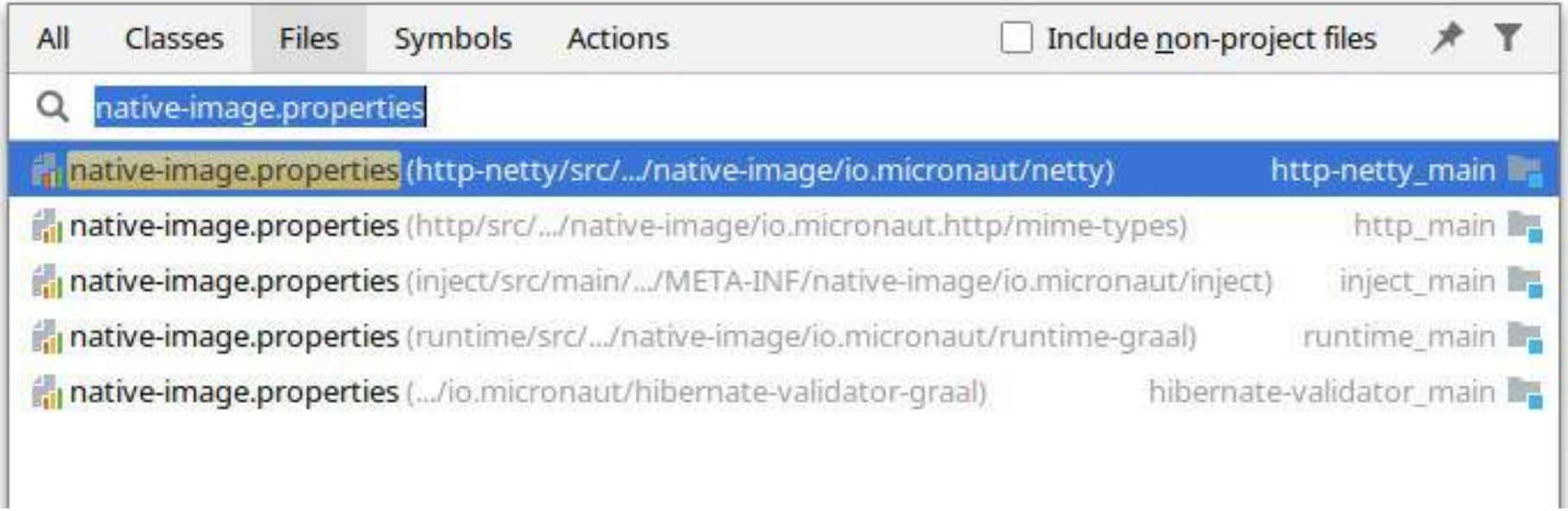
```
native-image --no-server -cp build/libs/basic-app-*.jar
```

```
# build-native-image.sh
native-image --no-server
       --class-path build/li       sic-app-0.1-all.jar \
       -H:ReflectionConfigurati       es=reflect.json \
       -H:EnableURLProtocols=http
       -H:IncludeResources="logback.       pplicatio       \
       -H:Name=basic-app \
       -H:Class=example.micronaut.Applica
       -H:+ReportUnsupportedElementsAtRunt
       -H:+AllowVMInspection \
       --allow-incomplete-classpath \
       --rerun-class-initializatio       untime=sun.       ity.jca.JCAUtil$CachedSecureRandomHolder,
                                         javax.n       l.SSLContext \
       --delay-class-initiali       n-to-runtime=io.netty.h       r.codec.http.HttpObjectEncoder,
                                  io.netty.handler.codec.htt       ocketx.WebSocket00FrameEncoder,
                                  io.netty.handler.ssl.util.Th       calInsecureRandom,
                                  com.sun.jndi.dns.DnsClient
```

How does it work?

OBJECT COMPUTING

- Composable `native-image.properties`
- `src/main/resources/META-INF/native-image/example.micronaut/app`



Micronaut packages everything


```
# inject/native-image.properties
Args = -H:+ReportUnsupportedElementsAtRuntime \
       --allow-incomplete-classpath
```

```
# http/native-image.properties
Args = -H:IncludeResources=META-INF/http/mime.types
```

```
# http-netty/native-image.properties
Args = --rerun-class-initialization-at-runtime=
                  sun.security.jca.JCAUtil$CachedSecureRandomHolder,javax.net.ssl.SSLContext \
       --delay-class-initialization-to-runtime=
io.netty.handler.codec.http.HttpObjectEncoder,io.netty.handler.codec.http.websocketx.WebSocket
00FrameEncoder,io.netty.handler.ssl.util.ThreadLocalInsecureRandom,com.sun.jndi.dns.DnsClient,
io.netty.handler.ssl.ConscryptAlpnSslEngine,io.netty.handler.ssl.JettyNpnSslEngine,io.netty.ha
ndler.ssl.ReferenceCountedOpenSslEngine,io.netty.handler.ssl.JdkNpnApplicationProtocolNegotiat
or,io.netty.handler.ssl.ReferenceCountedOpenSslServerContext,io.netty.handler.ssl.ReferenceCou
ntedOpenSslClientContext,io.netty.handler.ssl.util.BouncyCastleSelfSignedCertGenerator \
       -H:EnableURLProtocols=http,https
```

Micronaut support

- `@Introspected`
- `@TypeHint`
- `@ReflectiveAccess`
- Custom `src/main/graal/reflect.json`
- Graal TypeElementVisitor

GraalTypeElementVisitor

- `micronaut-core/graal/src/main/java/io/micronaut/graal/reflect/GraalTypeElementVisitor.java`
- Generates `reflection-config.json` and `native-image.properties`

GraalTypeElementVisitor (II)

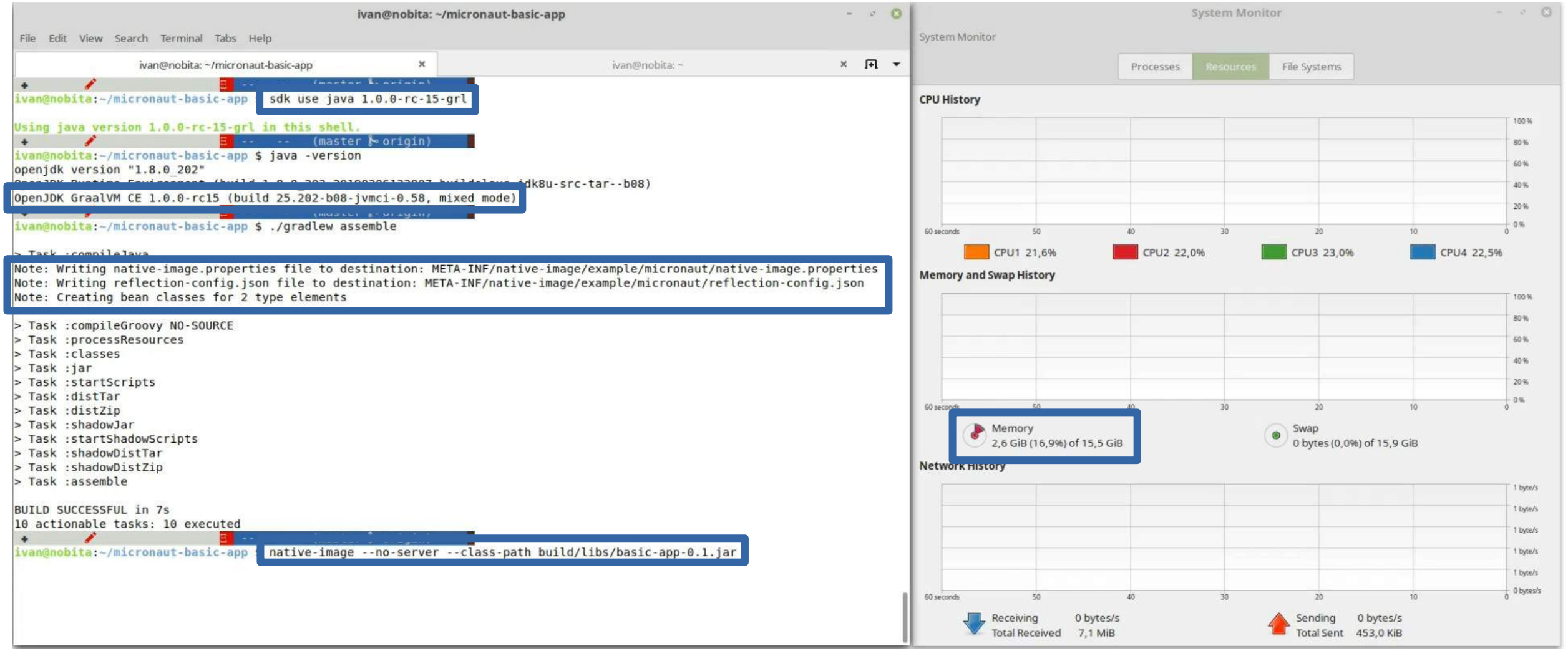

```
...
:http:compileJava
Note: Writing native-image.properties file to destination:
        META-INF/native-image/io.micronaut/http/native-image.properties
Note: Writing reflection-config.json file to destination:
        META-INF/native-image/io.micronaut/http/reflection-config.json

:http-netty:compileJava
Note: Writing native-image.properties file to destination:
        META-INF/native-image/io.micronaut/http-netty-channel/native-image.properties
Note: Writing reflection-config.json file to destination:
        META-INF/native-image/io.micronaut/http-netty-channel/reflection-config.json

:runtime:compileJava
Note: Writing native-image.properties file to destination:
        META-INF/native-image/io.micronaut/health/native-image.properties
Note: Writing reflection-config.json file to destination:
        META-INF/native-image/io.micronaut/health/reflection-config.json
...
```

OBJECT
COMPUTING

DEMO

Demo

OBJECT COMPUTING



How do we test this?

- Creating a native-image takes a lot of time
- And needs a lot of RAM
- GraalVM changes so fast (and they break things)...
- ...and we also do it ;-)
- We use Travis for Micronaut, but...
- ...Travis has limits for RAM
- We needed another way

GitLab

OBJECT
COMPUTING

Tests using Gitlab CI

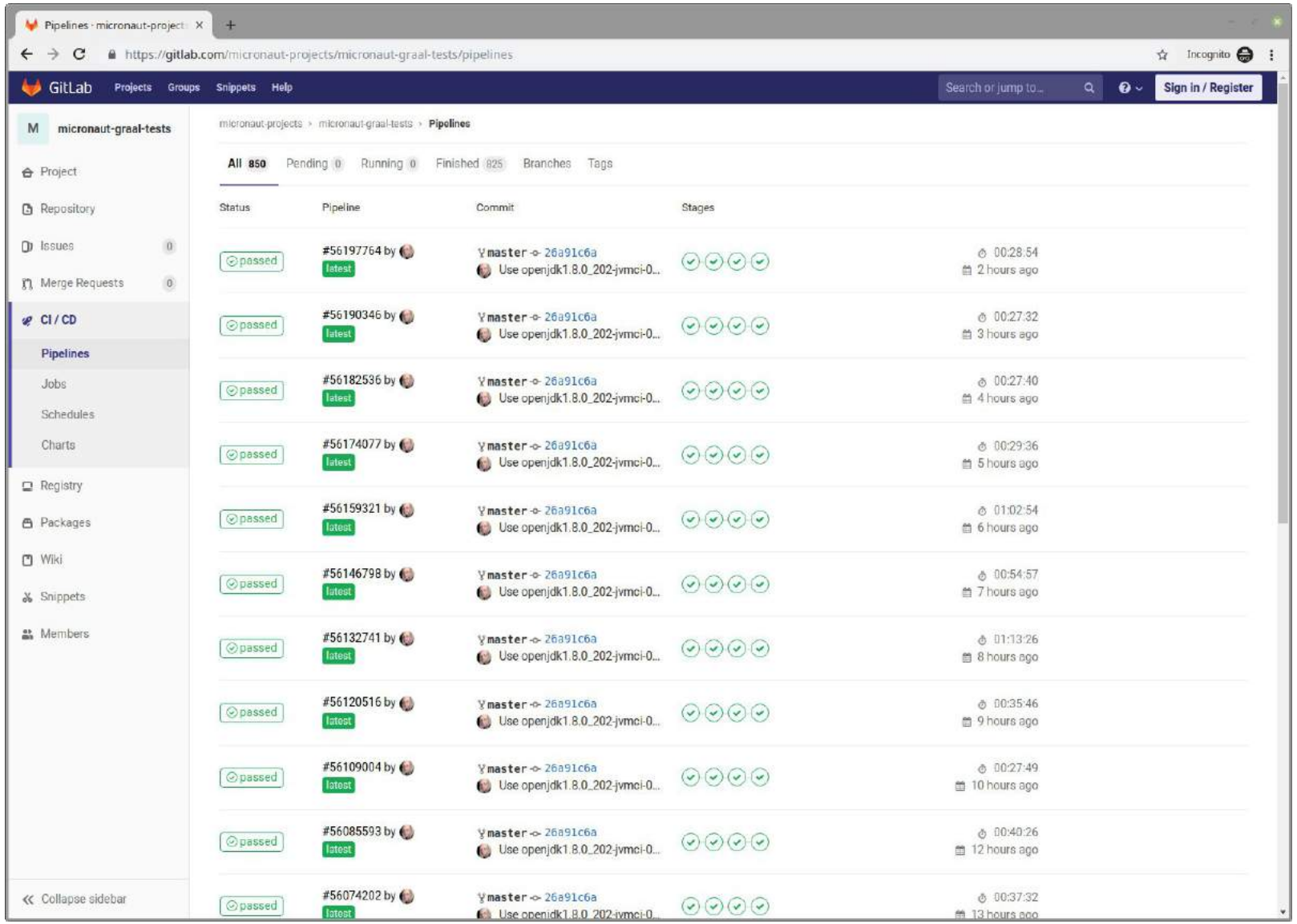
- Scheduled job every hour
- Only if new commits in Micronaut or Graal
- Compile Graal from source code
- Create native image for Micronaut test applications
- Run functional tests
- https://gitlab.com/micronaut-projects/micronaut-graal-tests

Micronaut Test applications

- Basic application: DI, POJO serialization, reactive type
- Micronaut function: AWS Function
- Service discovery: Consul support & HTTP Client
- Static resources & views
- Management & metrics endpoints
- Distributed tracing: Zipkin
- RabbitMQ: Fire-and-forget & RPC

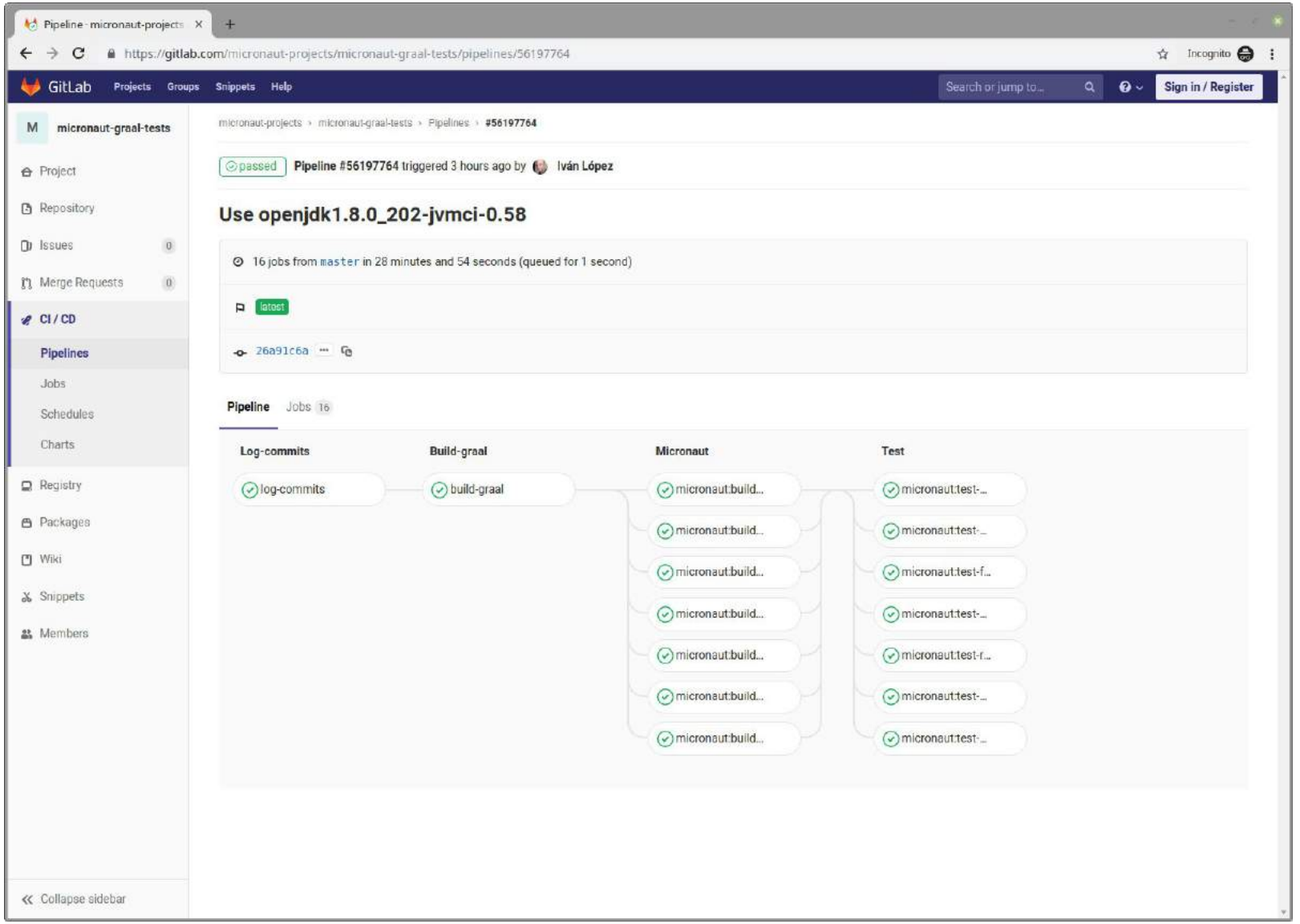
Gitlab CI

OBJECT COMPUTING



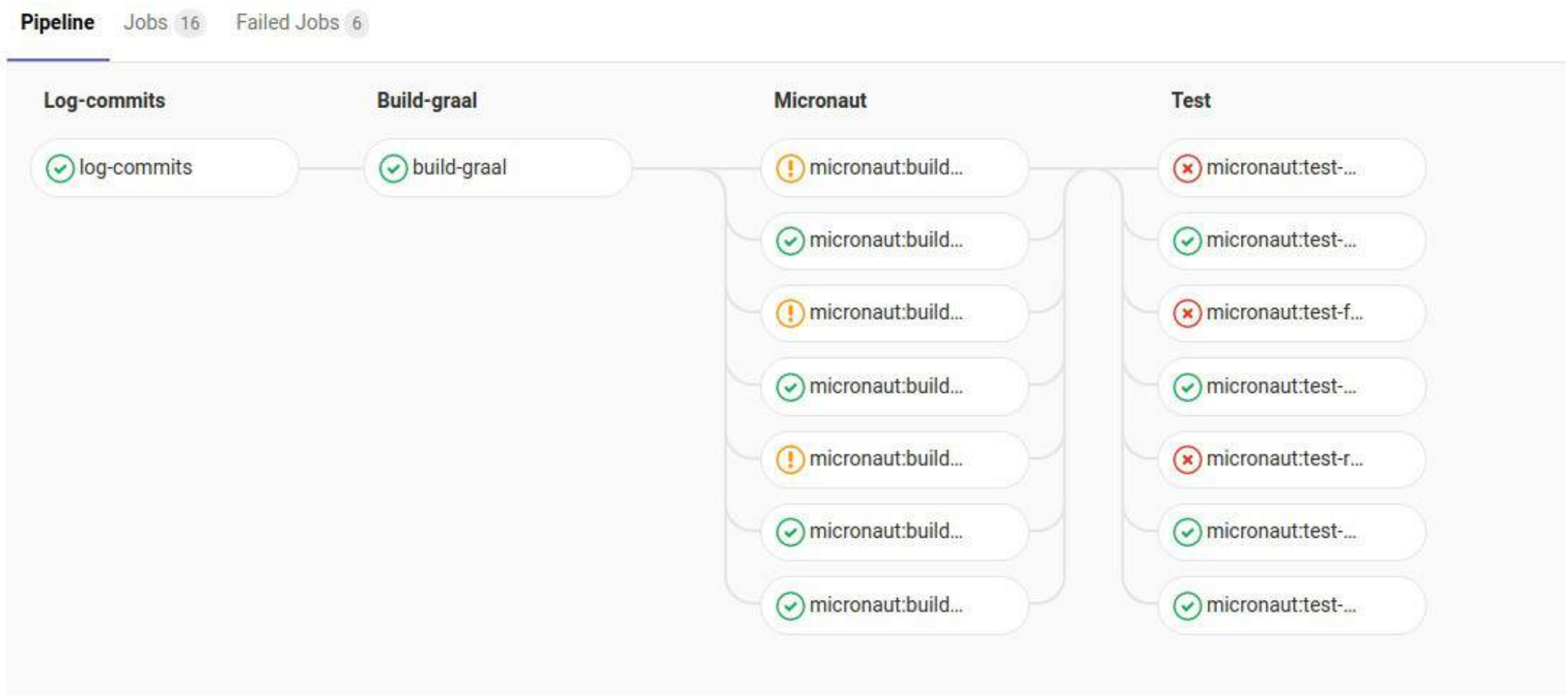
Gitlab CI

OBJECT COMPUTING



Gitlab CI

OBJECT COMPUTING



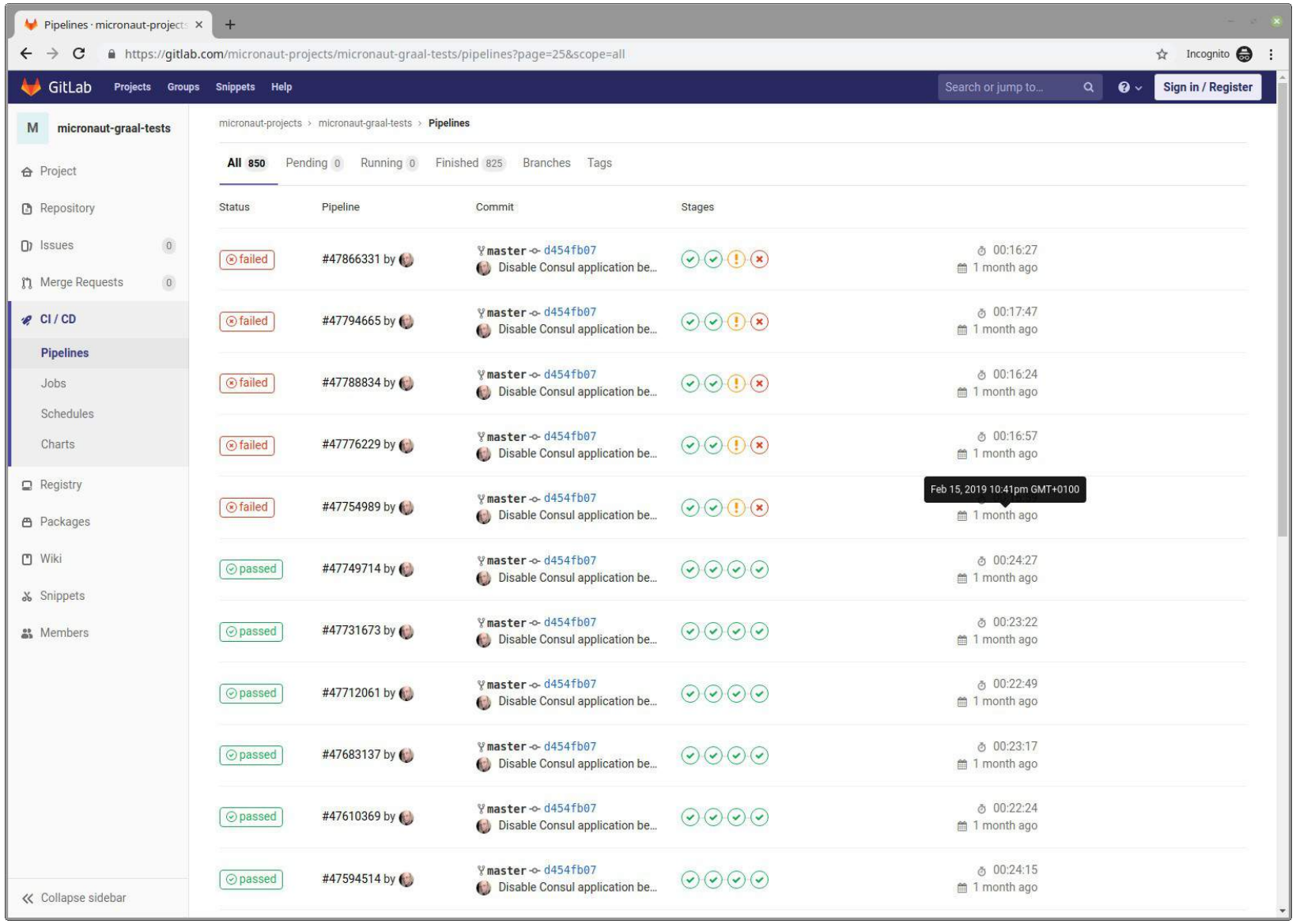
Gitlab CI

OBJECT COMPUTING

```
> Task :compileJava
Note: Writing native-image.properties file to destination: META-INF/native-image/example/micronaut/native-image.properties
Note: Writing reflection-config.json file to destination: META-INF/native-image/example/micronaut/reflection-config.json
Note: Creating bean classes for 1 type elements
/builds/micronaut-projects/micronaut-graal-tests/micronaut-basic-app/src/main/java/example/micronaut/GreetingService.java:5: error:
package javax.inject does not exist
import javax.inject.Singleton;
                   ^
/builds/micronaut-projects/micronaut-graal-tests/micronaut-basic-app/src/main/java/example/micronaut/GreetingService.java:7: error:
cannot find symbol
@Singleton
 ^
  symbol: class Singleton
2 errors

> Task :compileJava FAILED
```

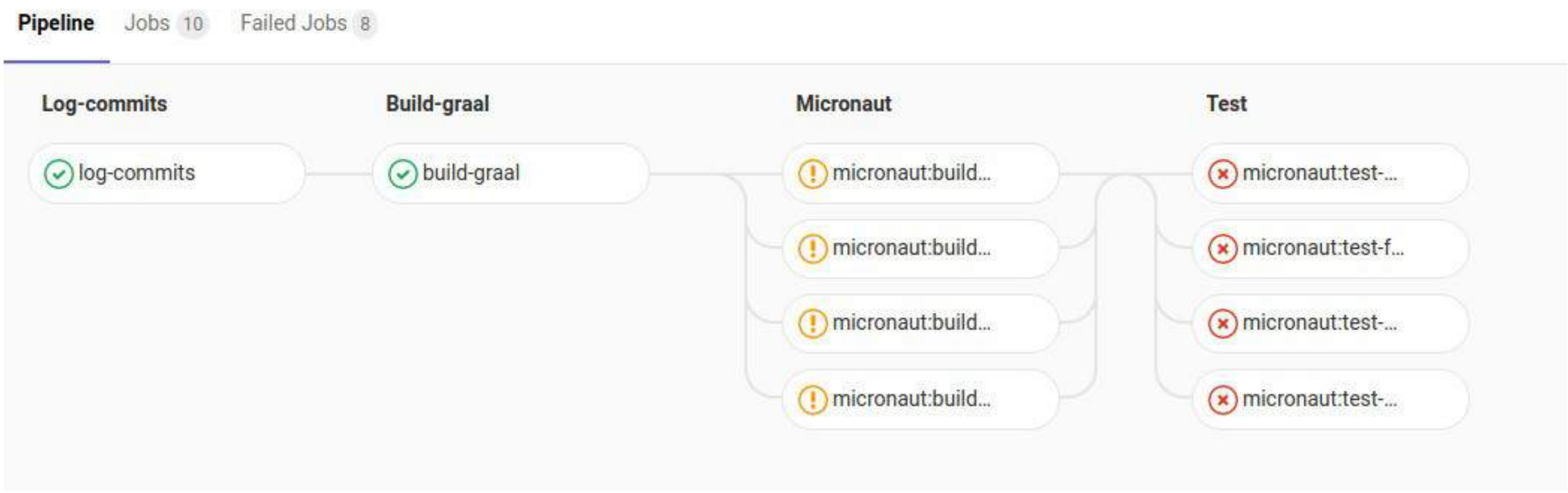
Gitlab CI

OBJECT COMPUTING



Gitlab CI

OBJECT COMPUTING



Gitlab CI


```
Graal Class Loading Analysis Enabled.
Graal Class Loading Analysis Enabled.
Writing reflect.json file to destination: build/reflect.json
[basic-app:96]    classlist:   9,785.03 ms
[basic-app:96]        (cap):   3,001.95 ms
[basic-app:96]        setup:   6,848.65 ms
[basic-app:96]     analysis:  47,532.28 ms
Fatal error: java.lang.IndexOutOfBoundsException: index 2
        at java.util.concurrent.atomic.AtomicReferenceArray.checkedByteOffset(AtomicReferenceArray.java:78)
        at java.util.concurrent.atomic.AtomicReferenceArray.get(AtomicReferenceArray.java:125)
        at com.oracle.graal.pointsto.flow.context.object.AnalysisObject.getInstanceFieldTypeStore(AnalysisObject.java:213)
        at com.oracle.graal.pointsto.flow.context.object.AnalysisObject.getInstanceFieldFlow(AnalysisObject.java:199)
        at com.oracle.graal.pointsto.flow.LoadFieldTypeFlow$LoadInstanceFieldTypeFlow.onObservedUpdate(LoadFieldTypeFlow.java:159)
        at com.oracle.graal.pointsto.flow.TypeFlow.notifyObservers(TypeFlow.java:347)
        at com.oracle.graal.pointsto.flow.TypeFlow.update(TypeFlow.java:389)
        at com.oracle.graal.pointsto.BigBang$2.run(BigBang.java:508)
        at com.oracle.graal.pointsto.util.CompletionExecutor.lambda$execute$0(CompletionExecutor.java:169)
        at java.util.concurrent.ForkJoinTask$RunnableExecuteAction.exec(ForkJoinTask.java:1402)
        at java.util.concurrent.ForkJoinTask.doExec(ForkJoinTask.java:289)
        at java.util.concurrent.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1056)
        at java.util.concurrent.ForkJoinPool.runWorker(ForkJoinPool.java:1692)
        at java.util.concurrent.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:157)
Error: Image building with exit status 1
ERROR: Job failed: exit code 1
```

Gitlab CI

OBJECT COMPUTING

Regression introduced: IndexOutOfBoundsException creating Micronaut native-image apps #985

New issue

Closed **ilopmar** opened this issue on Feb 16 · 3 comments

ilopmar commented on Feb 16

Our CI has detected a regression introduced in this commit: `d4558fb`

All of our Micronaut Graal test applications fail when building the native image with the following error:

```
[basic-app:96]    classlist:   9,785.03 ms
[basic-app:96]        (cap):   3,001.95 ms
[basic-app:96]        setup:   6,848.65 ms
[basic-app:96]     analysis:  47,532.28 ms
Fatal error: java.lang.IndexOutOfBoundsException: index 2
        at java.util.concurrent.atomic.AtomicReferenceArray.checkedByteOffset(AtomicReferenc
        at java.util.concurrent.atomic.AtomicReferenceArray.get(AtomicReferenceArray.java:12
        at com.oracle.graal.pointsto.flow.context.object.AnalysisObject.getInstanceFieldType
        at com.oracle.graal.pointsto.flow.context.object.AnalysisObject.getInstanceFieldFlow
        at com.oracle.graal.pointsto.flow.LoadFieldTypeFlow$LoadInstanceFieldTypeFlow.onObse
        at com.oracle.graal.pointsto.flow.TypeFlow.notifyObservers(TypeFlow.java:347)
        at com.oracle.graal.pointsto.flow.TypeFlow.update(TypeFlow.java:389)
        at com.oracle.graal.pointsto.BigBang$2.run(BigBang.java:508)
        at com.oracle.graal.pointsto.util.CompletionExecutor.lambda$execute$0(CompletionExec
        at java.util.concurrent.ForkJoinTask$RunnableExecuteAction.exec(ForkJoinTask.java:14
        at java.util.concurrent.ForkJoinTask.doExec(ForkJoinTask.java:289)
        at java.util.concurrent.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1056)
        at java.util.concurrent.ForkJoinPool.runWorker(ForkJoinPool.java:1692)
        at java.util.concurrent.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:157)
Error: Image building with exit status 1
```

In this CI execution https://gitlab.com/micronaut-projects/micronaut-graal-tests/pipelines/47754989 you can check that all the applications fail in the "Micronaut" stage. This is the stage we use to build native images with latest GraalVM built from master in the "Build-graal" pipeline stage.

To reproduce the issue you can use any of our test applications used in CI, for example:

- `git clone https://github.com/micronaut-graal-tests/micronaut-basic-app`
- `cd micronaut-basic-app/`
- Use GraalVM built from master
- `./build-native-image.sh`

👍 1

Assignees

vjovanov

cstancu

Labels

bug

substrate

Milestone

No milestone

4 participants

cstancu commented on Feb 16 — Member

Thanks for the quick report! I caught this on our own integration tests too and will have a fix soon.

🎉 2

thomaswue assigned **cstancu** on Feb 17

thomaswue added bug substrate labels on Feb 17

cstancu referenced this issue on Feb 19

[native-image] wrong type in AnalysisObject #995 — Closed

vjovanov self-assigned this on Feb 19

dougxc pushed a commit that referenced this issue on Feb 20

`[GR-14006] Fix DynamicHub.getProtectionDomain() substitution [#985, #995` ··· `9f05467`

cstancu commented on Feb 20 — Member

The issue is fixed in `9f05467`. The problem was with the `DynamicHub.getProtectionDomain()` substitution: it declared `Object` return type instead of `ProtectionDomain`, so the points-to analysis was not applying the expected type narrowing. I also improved the error reporting: instead of an `ArrayIndexOutOfBoundException` now you would get `AnalysisError$FieldNotPresentError: Field java.security.ProtectionDomain.codesource is not present on type java.lang.Object. Error encountered while analysing java.security.ProtectionDomain.getCodeSource()`.

cstancu closed this on Feb 20

ilopmar commented on Feb 20 — Author

Thanks! I can confirm that our integration tests pass again 👍

👍 1

Summary

OBJECT COMPUTING

GraalVM is new and exciting technology

Micronaut AoT is the perfect fit

Micronaut support out-of-the-box

We'll keep improving the support

Fast start-up & low memory consumption

Developers are happy & productive

Questions?

OBJECT
COMPUTING

CONNECT WITH US

1+ (314) 579-0066

@objectcomputing

objectcomputing.com