

GraalVM™

ConFoo.ca
DEVELOPER CONFERENCE

Unleash the power of your
applications with Micronaut®
and GraalVM

Álvaro Sánchez-Mariscal

Principal Member of Technical Staff
Oracle



@alvaro_sanchez





@alvaro_sanchez



Álvaro Sánchez-Mariscal

Principal Member of Technical Staff, Oracle



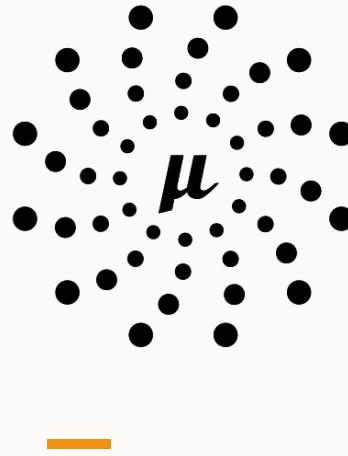
- Coming from Madrid 🇪🇸
- Developer since 2001 (Java ☕ stack)
- Micronaut core developer since its inception (2017).
 - Author: Maven Plugin, Object Storage, Kubernetes, Cache, Control Panel.
 - Others: Core, Gradle, AWS, GCP, Azure, Security, Test, etc.
- Currently at Oracle Labs:
 - Micronaut and Graal Cloud Native (GCN).
 - GraalVM Native Build Tools.
- Speaker at conferences: Devoxx, GeeCON, JavaLand, JavaZone, jPrime, Codemotion, Commit Conf, etc.



ORACLE
Labs

Slides available on Speaker Deck





MICRONAUT®



Introduction to Micronaut



Modern Java Framework

Micronaut has been designed from scratch in 2017, focused on modern architectures like microservices and serverless, and with the cloud in mind.



All application types

Micronaut is a complete solution for any type of application: microservices, message-driven producers or consumers, command-line apps, serverless functions, etc.



Highly optimised

Micronaut leverages Java annotation processors and other optimisations to compute the framework infrastructure at compile-time, drastically reducing startup time and memory consumption.

AOT: Ahead Of Time

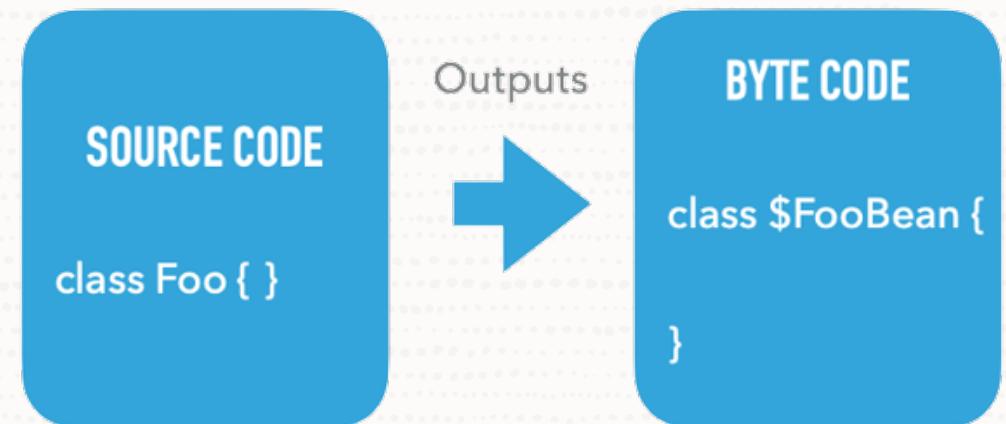


Micronaut computes at **build time**:

- All dependency and configuration injection.
- Annotation metadata (meta annotations)
- AOP proxies.
- Bean introspections.
- All other framework infrastructure.

At **runtime**:

- No reflection.
- No proxy generation.
- No dynamic classloading.
- No classpath scanning.



Any language, build tool, test framework, reactive library, cloud, ...



Maven™



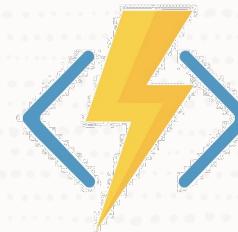
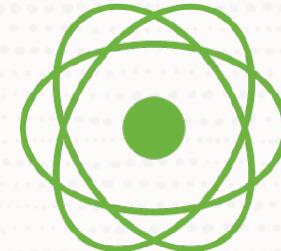
Gradle



GraalVM™



Kotlin Test



Google Cloud



Micronaut Hello World



```
@Controller("/hello")
public class HelloController {

    @Get(produces = MediaType.TEXT_PLAIN)
    public String index() {
        return "Hello World";
    }
}
```

Micronaut features: HTTP Client



Implemented at compile time

```
@Client("/hello")
public interface HelloClient {

    @Get
    String index();
}
```

Micronaut features: message-driven applications



Kafka

- Producers.
- Consumers.
- Kafka Streams.



RabbitMQ

- Producers.
- Consumers.
- RPC.



MQTT

- Publishers.
- Subscribers.



NATS

- Producers.
- Consumers.

Micronaut features: data access



Micronaut features: data access



Interface implemented at compile time



```
@JdbcRepository(dialect = Dialect.ORACLE)
public interface UserRepository extends CrudRepository<User, Long> {
    List<User> findByNameOrderByName(String name);
    @Query("SELECT * FROM user WHERE enabled = false")
    List<User> findDisabled();
}
```

Query generated from the
method name, compile-
time validated

Can have custom queries,
inserts, etc

Micronaut features: security



1. Authentication providers.

- Built-in support for LDAP and OAuth 2.0 password grant flow.

2. Security rules.

- Configuration or annotation-based.

3. Authorization strategies.

- Basic auth, session, JWT (JWKS, JWS), X.509.

4. OAuth 2.0.

- Authorization code, client credentials and password grants.
- OpenID Connect.
- Ease of integration with providers such as Okta, Auth0, AWS Cognito, Keycloak and more.

Micronaut features: Test Resources



1. Automatic provisioning of external resources needed during **development or testing**.
 - Containers, binaries, etc.
2. Built-in support:
 - MariaDB
 - MySQL
 - Oracle
 - PostgreSQL
 - SQL Server
 - Elasticsearch
 - Redis
 - MongoDB
 - Neo4j
 - Hashicorp Vault
 - Kafka
 - RabbitMQ
 - MQTT
 - Localstack
3. Integrated with JUnit Platform (JUnit / Spock / Kotest) for zero-configuration Testcontainers.

Micronaut features



1. Dependency injection.

- Can use JSR-330 (@Inject) or Spring (@Autowired) annotations.

2. Configuration.

- Properties, YAML, TOML, Groovy, Config4K.

3. Validation support.

- Built-in: reflection free, faster startup, smaller JAR, reduced memory.
- Hibernate Validation: full Bean Validation API compliance.

4. AOP: Aspect-Oriented Programming.

- Compile-time, reflection free.

Micronaut features: misc



1. Distributed Tracing.

- Zipkin, Jaeger.

2. Service Discovery and Distributed Configuration.

- Consul, Eureka, Kubernetes.

3. Monitoring.

- Micrometer, JMX, Elasticsearch.

4. API development.

- GraphQL, GRPC, Open API.

5. Containers.

- Docker, Kubernetes, Testcontainers.

6. Cache.

- Jcache, Redis, Ehcache, Hazelcast, Infinispan, Oracle Coherence.

7. Email.

8. Error handling.

Getting Started: Micronaut Launch

A screenshot of the Micronaut Launch application interface. The interface is dark-themed with white text and light-colored buttons. At the top left is the Micronaut logo (a stylized 'μ' symbol). To its right is the text 'MICRONAUT® LAUNCH'. On the far right are social media sharing icons for GitHub, Twitter, LinkedIn, and Email. Below the header, there are two main sections: 'Application Type' and 'Java Version'. Under 'Application Type', 'Micronaut Application' is selected. Under 'Java Version', '11' is selected. In the center, there's a 'Name' field containing 'demo', a 'Base Package' field containing 'com.example', and several configuration dropdowns for 'Micronaut Version' (3.4.3), 'Language' (Java), 'Build Tool' (Gradle), and 'Test Framework' (JUnit). At the bottom, there are four buttons: '+ FEATURES', 'DIFF', 'PREVIEW', and 'GENERATE'. A section titled 'Included Features (1)' shows a single feature: 'graalvm' with a close button ('X').

MICRONAUT®
LAUNCH

Application Type
Micronaut Application

Name
demo

Java Version
11

Base Package
com.example

Micronaut Version
 3.4.3
 3.4.4-SNAPSHOT
 2.5.13

Language
 Java
 Groovy
 Kotlin

Build Tool
 Gradle
 Gradle Kotlin
 Maven

Test Framework
 JUnit
 Spock
 Kotest

+ FEATURES DIFF PREVIEW GENERATE

Included Features (1)
graalvm X

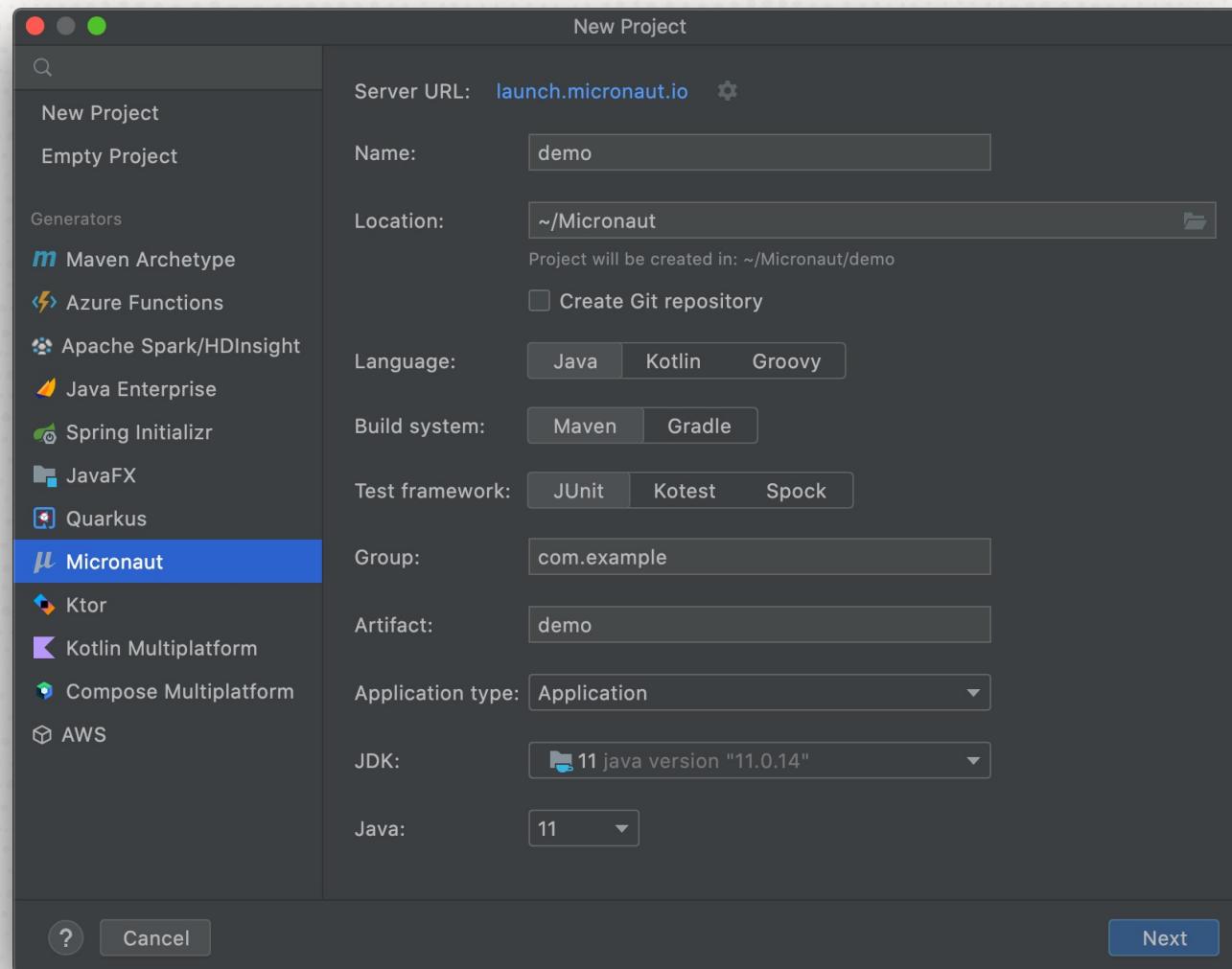
<https://launch.micronaut.io>

Getting Started: Micronaut CLI



```
$ mn create-app com.example.demo --features graalvm  
| Application created at /tmp/demo
```

Getting Started: IntelliJ IDEA



Getting Started: Visual Studio Code



The screenshot shows the Visual Studio Code interface with the Micronaut Tools extension installed. The sidebar on the left has icons for file operations, search, navigation, and other extensions. The main area is titled "Micronaut Tools" and displays "Micronaut Support for Visual Studio Code. Integrated with GraalVM to get the most from your applications." A modal window titled "Create Micronaut Project (1/9)" is open, showing a dropdown menu for "Pick Micronaut version" with options: 3.4.3 (selected), 3.4.4-SNAPSHOT, and 3.4.3 (using local CLI). Below the modal, there are sections for "Project" (Create new Micronaut project..., Open existing Micronaut project...), "Native Image" (Build native image, Learn about native image), "Help" (Extension documentation, Request a feature, File a bug, Send feedback), and "Getting Started" (Sample application, Create your first Micronaut application). There are also sections for "Documentation" (Micronaut documentation, User guide, API reference, and much more on Micronaut) and "Micronaut guides" (Learn all Micronaut features). At the bottom of the screen, the status bar shows "Indexing completed." and various status icons.

Micronaut 4: what's new



1. Java 17 required.

- Virtual threads support (Project Loom).
- @Client implementation based on java.net.http.HttpClient (JEP 321)
- For older versions of Java use Micronaut 3.x.
- Other languages
 - Kotlin 1.8, KSP support.
 - Apache Groovy 4.

2. GraalVM improvements.

- Reachability metadata enabled.
- Improved build times and reduced memory consumption.

3. Micronaut Data v4

- Hibernate 6.

Micronaut 4: what's new



3. Smaller, lighter runtime.

- Performance closer to raw Netty (~200ms startup).
- Some modules spun off core: Service Discovery, Retry, Session, Validation, Websocket.
- Some dependencies now optional: Jackson, SnakeYAML, Caffeine.

4. Other features:

- HTTP client/server filter methods: annotation-based and without reactive APIs.
- Annotation-based CORS configuration.
- HTTP/3 support.
- Prefer Micronaut Serialization over Jackson.
- Disabled cloud environment deduction by default.

Micronaut 4: what's new



Checked at compile time

5. Compile-time expression language

```
@Scheduled(fixedRate = "1s", condition = "#{!jc.paused}")
public void run(ExampleJobControl jc) {
    System.out.println("Job Running");
    this.jobRan = true;
}
```

Micronaut 4: what's new



5. Compile-time expression language

```
● ● ●  
  
@Controller  
public class MyController {  
  
    @Get  
    @RouteCondition( "#{request.headers.getFirst('Authorization')?.contains('foo')}" )  
    String hello() {  
        return "ok";  
    }  
}
```

Micronaut 4: what's new



6. Bean mappers: built-in, type-safe alternative to MapStruct for type mapping built on EL.

Micronaut 4 Control Panel



The screenshot shows the Micronaut Control Panel interface. On the left, there's a sidebar with 'Dashboard' and 'My Application' buttons. The main area has tabs for 'Dashboard' and 'Bean Definitions'. Under 'Bean Definitions', it says '248' and lists sections for 'Other beans' (with 'Package com.example' and 'Package io.netty.channel' options) and 'Micronaut Framework beans' (with 'Package aop', 'Package buffer', 'Package context', 'Package runtime', 'Package scheduling', and 'Package reactor' options). A modal window titled 'Package com.example' displays a dependency graph. Nodes include 'com.example.\$Application\$AllPlainEnvironmentEndpointFilter\$Definition\$Intercepted', 'com.example.ApplicationControlPanel', 'com.example.MyController', 'io.micronaut.context.BeanContext', 'io.micronaut.context.BeanRegistration', 'io.micronaut.context.BeanResolutionContext', and 'io.micronaut.context.Qualifier'. Arrows show dependencies from the application filter node to BeanContext, BeanRegistration, and BeanResolutionContext, and from the ApplicationControlPanel node to Qualifier. At the bottom of the modal is a 'Close' button. Below the modal, a terminal window shows Java system properties:

```
sun.boot.library.path = /Users/alvaro/.sdkman/candidates/java/22.3.r17-grl/jdk
sun.java.command = com.example.Application
com.sun.management.jmxremote =
http://127.0.0.1:160-254/161-160-254/16
```

Micronaut 4 coming soon: source generation

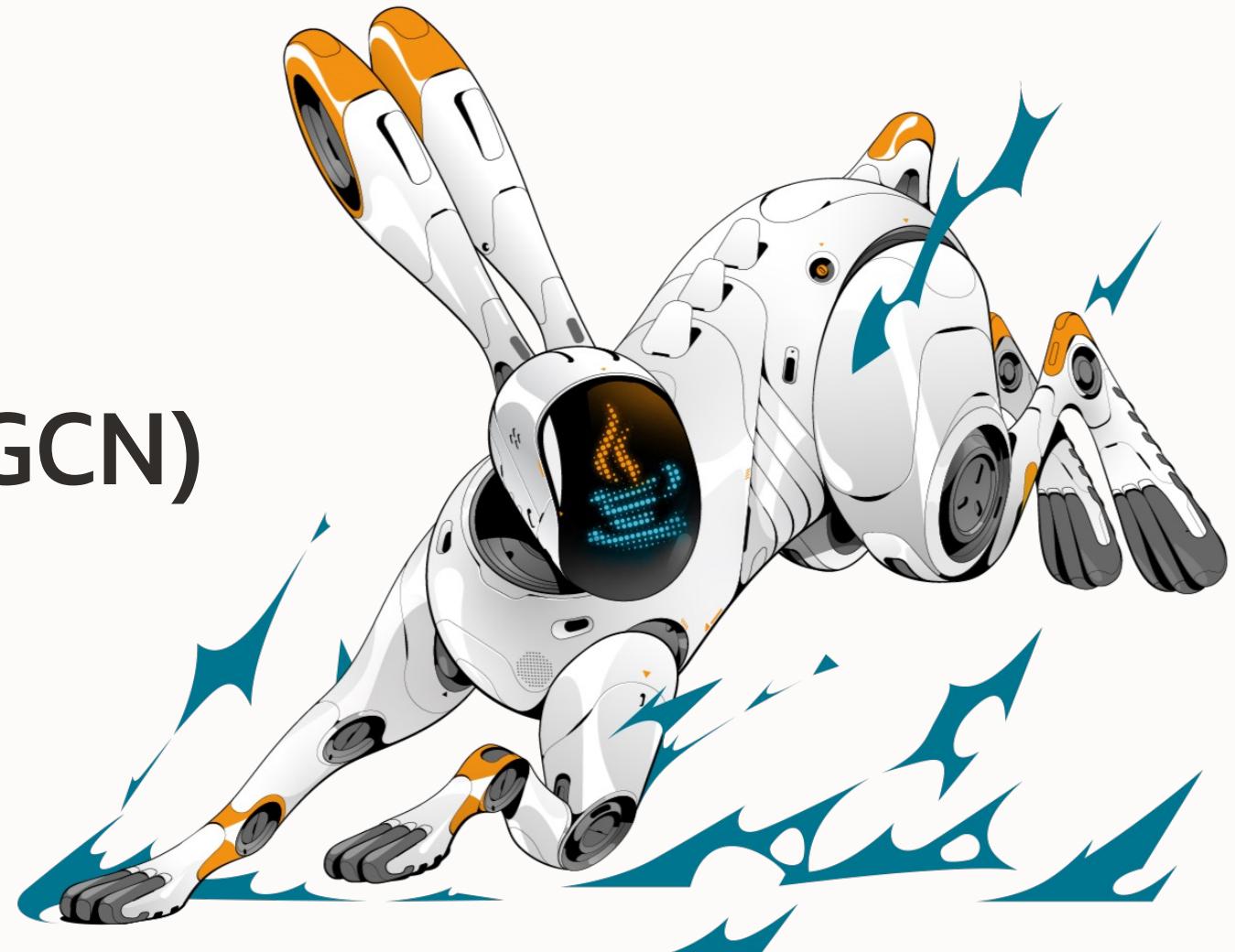


Generates
PersonBuilder at
compile-time

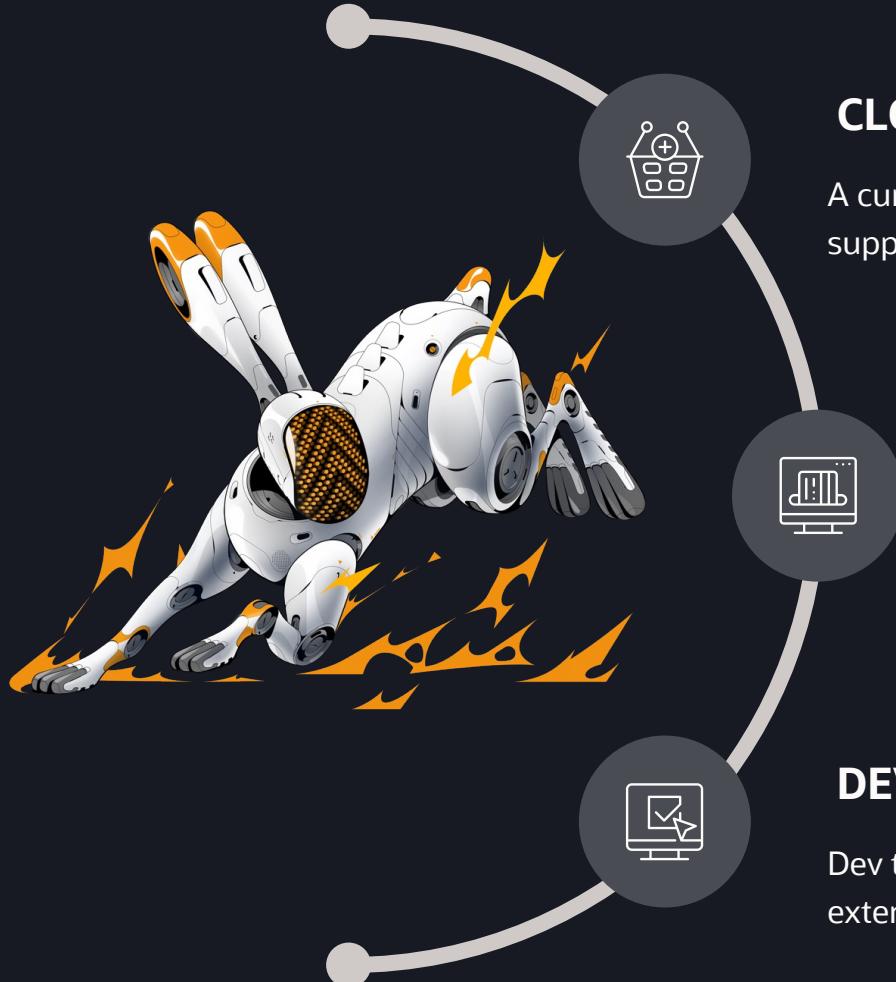
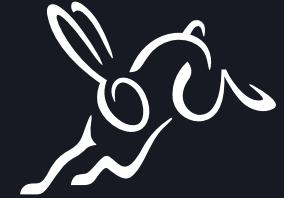
```
import io.micronaut.sourcegen.annotations.Builder;  
  
@Builder  
public record Person(Long id, String name) {}
```

```
@Test  
public void buildsPerson() {  
    var person = PersonBuilder.builder()  
        .id(123L)  
        .name("Cédric")  
        .build();  
    assertEquals("Cédric", person.name());  
    assertEquals(123L, person.id());  
}
```

Graal Cloud Native (GCN)



What is Graal Cloud Native?



CLOUD-AGNOSTIC MODULES

A curated set of Micronaut modules that provide platform-independent support for core cloud services – object store, secrets, streaming, and more.

BUILT FOR GRAALVM NATIVE IMAGE

Easily compile ahead-of-time with GraalVM Native Image into small executables that start instantly, and use less memory/CPU.

DEV TOOLS AND VS CODE EXTENSIONS

Dev tools to generate multicloud application starter templates. VS Code extensions for local development and cloud deployment.

<https://graal.cloud>



GraalVM™



What is GraalVM



java MyMainClass
OpenJDK™

native-image MyMainClass
./mymainclass



Native Image Goals



Start Fast



Low Resource Usage



Minimize Vulnerability



Compact Packaging

Micronaut and GraalVM



As low as
6ms
startup time

Increased throughput up to
>30%
and reduced latency

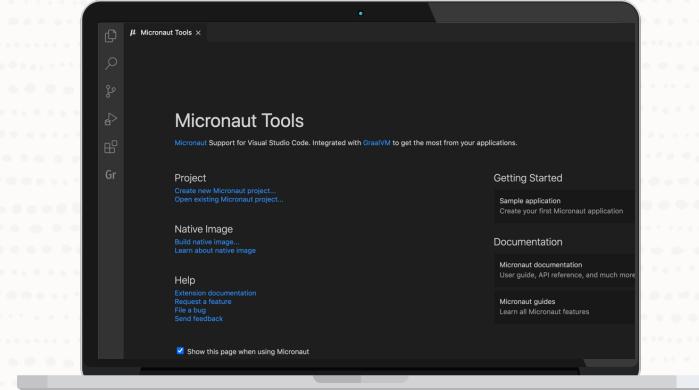
Consuming
18MB
memory footprint

Micronaut and GraalVM



1. The best integration possibly, supported by teams working together at Oracle Labs.
 - GraalVM EE includes optimisations for Micronaut for increased performance and throughput.
 - GraalVM EE license included in Oracle Cloud.
2. Micronaut is ready for GraalVM Native Image since day 1.
 - No reflection, no runtime proxies, no bytecode generation, no dynamic classloading.
3. GraalVM Extension Pack for Visual Studio Code.

GraalVMTM
Extension Pack



GraalVM Reachability Metadata Repository



Community-driven collection of GraalVM reachability metadata for open-source libraries.

- Actively maintained by a collective effort of VMware, Oracle and others.
- Enabled in GraalVM Native Build Tools (Maven and Gradle plugins).
- Integrated by default in Spring Boot and Micronaut.



```
# Maven  
$ mvn -Pnative native:compile  
  
# Gradle  
$ gradle nativeCompile
```

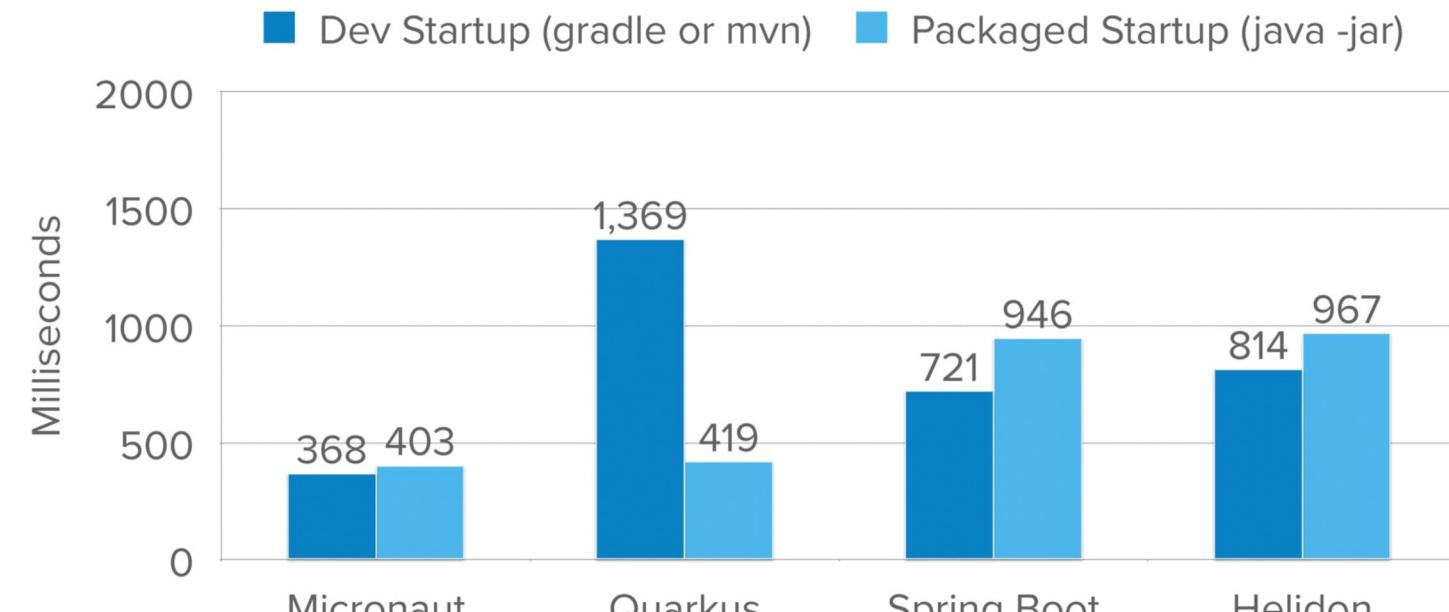


```
# Maven  
$ mvn package -Dpackaging=native-image  
  
# Gradle  
$ gradle nativeCompile
```

Micronaut is the fastest to startup



Startup Performance



@mraible

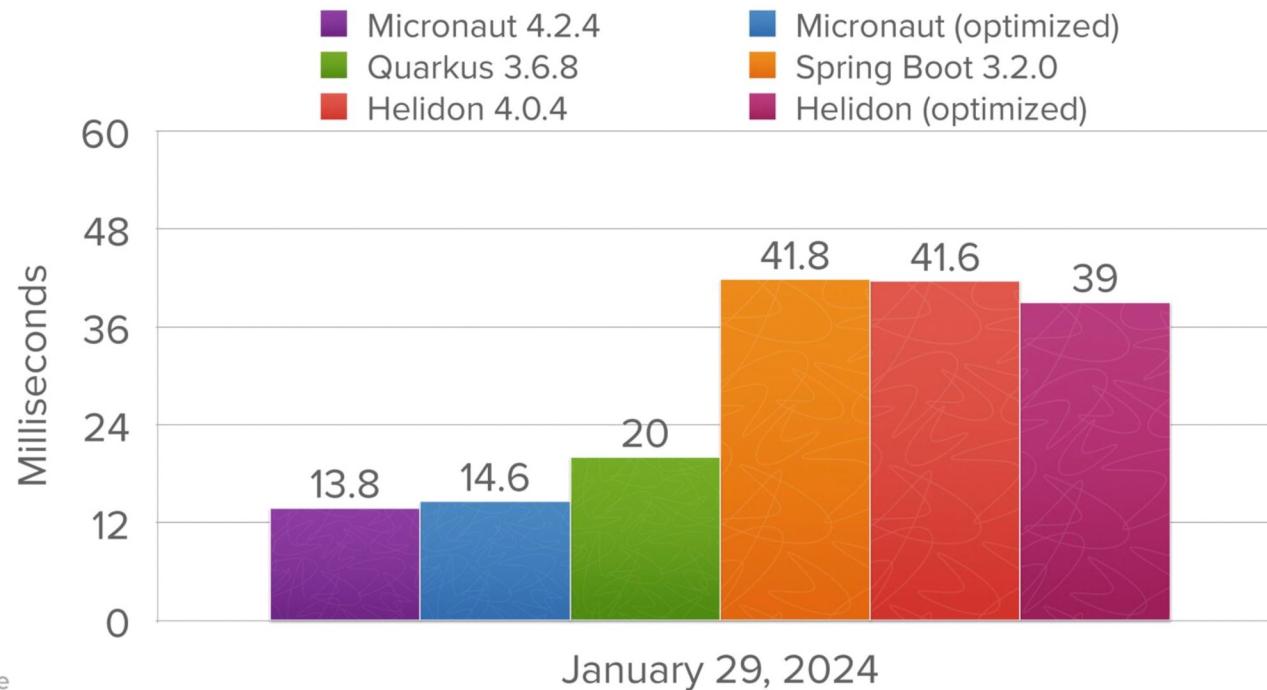


Source: <https://speakerdeck.com/mraible/comparing-native-java-rest-api-frameworks-jchampions-conference-2024>

Micronaut is the fastest to startup

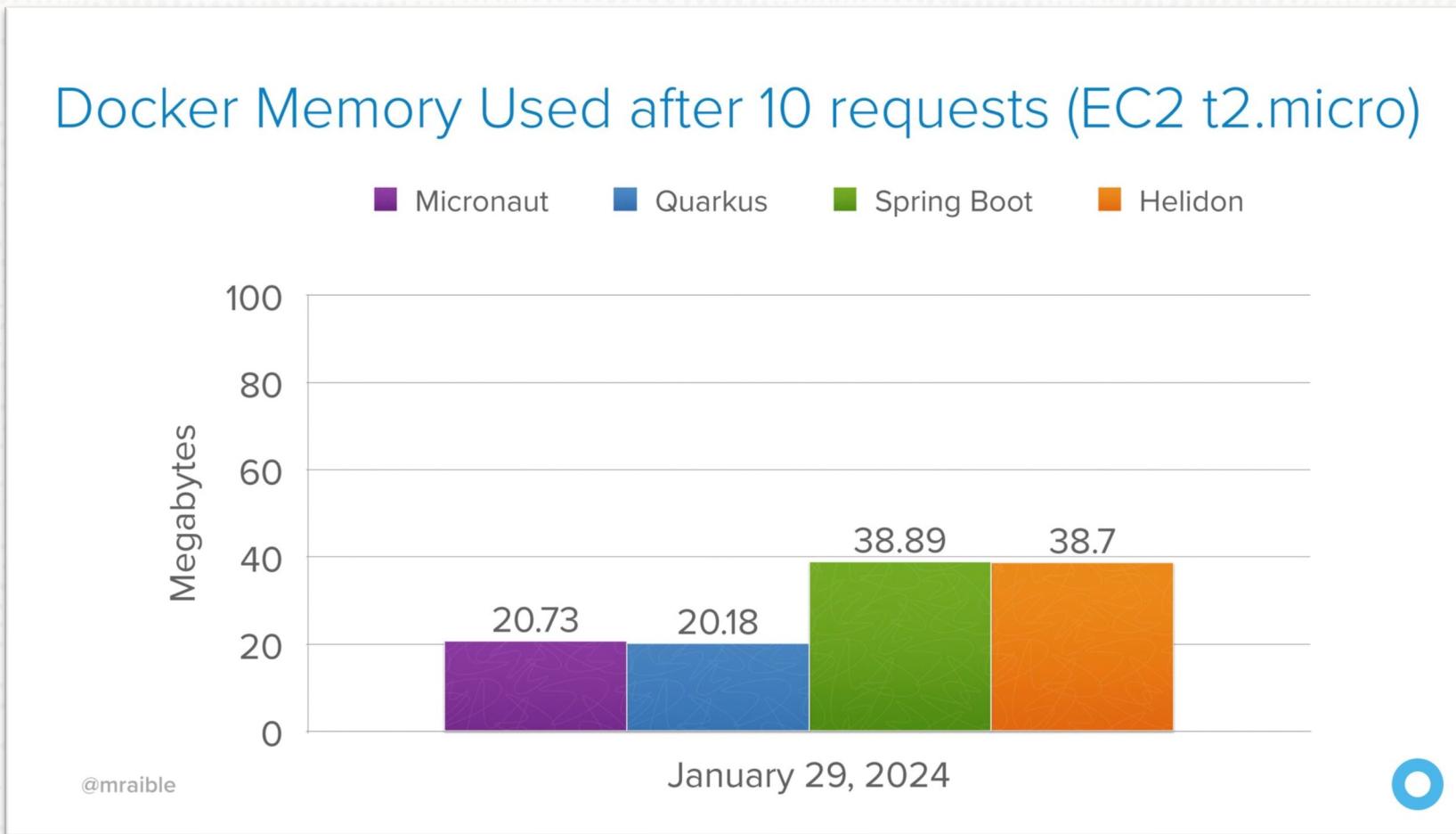


Native Startup Performance (M1; time from console)



Source: <https://speakerdeck.com/mraible/comparing-native-java-rest-api-frameworks-jchampions-conference-2024>

Micronaut consumes the less memory



Source: <https://speakerdeck.com/mraible/comparing-native-java-rest-api-frameworks-jchampions-conference-2024>

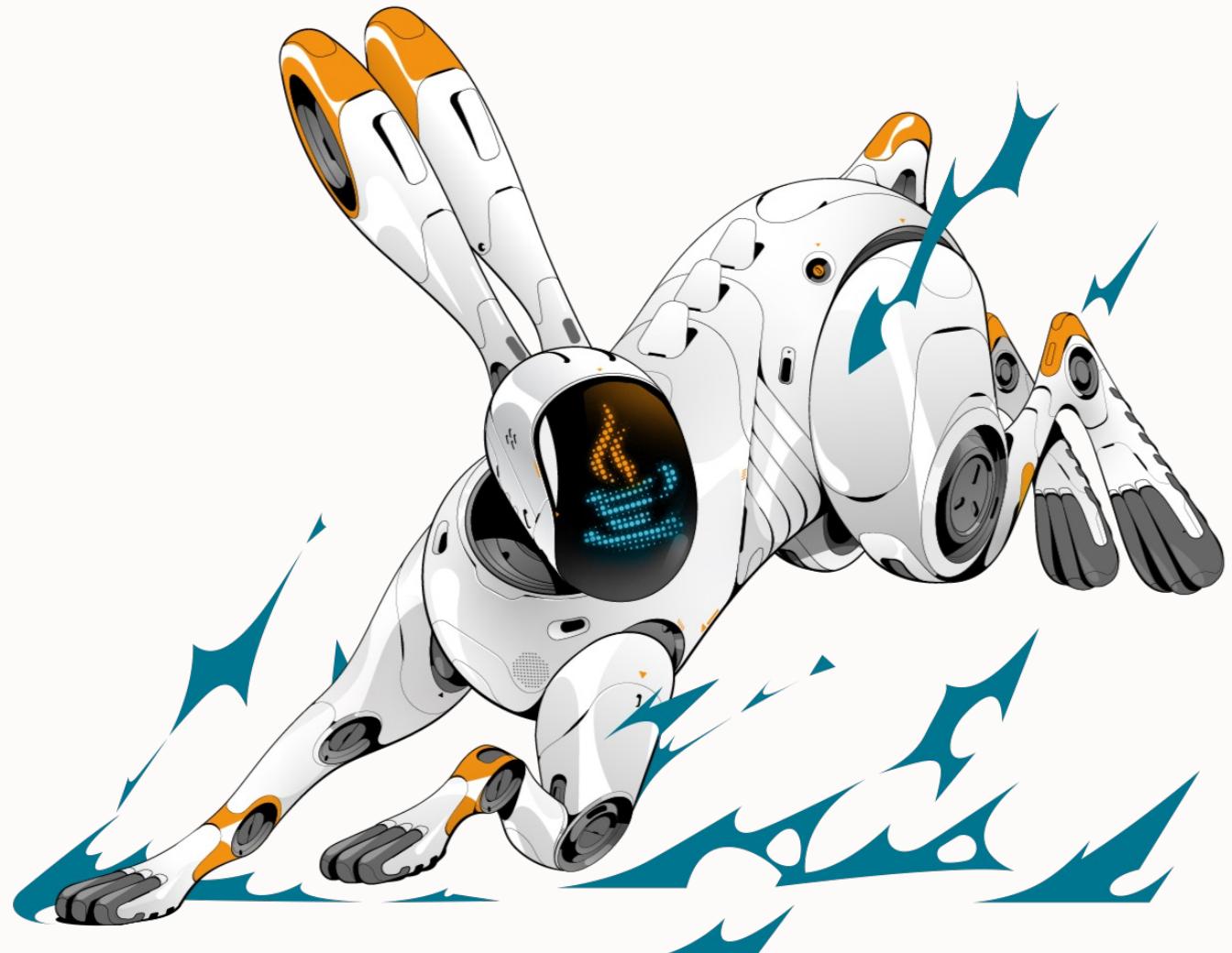
Micronaut and GraalVM powering Disney+



<https://aws.amazon.com/blogs/opensource/improving-developer-productivity-at-disney-with-serverless-and-open-source/>

Demo

Micronaut and GraalVM



Q &A



Rate me 🙏

Álvaro Sánchez-Mariscal



@alvaro_sanchez

Thank
you

ConFoo.ca
DEVELOPER CONFERENCE

ORACLE

