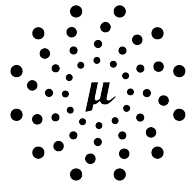
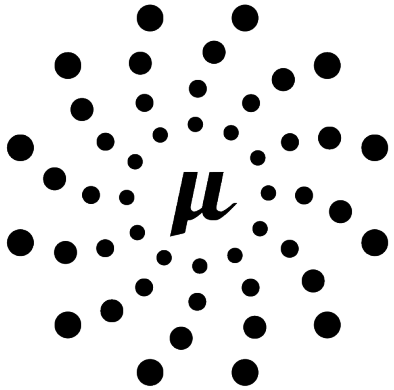
August 2022

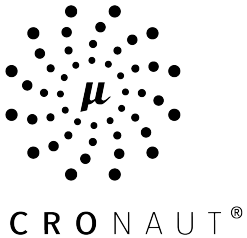


Micronaut Essentials

micronaut.io



MICRONAUT®

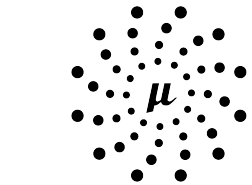


OCI
12140 Woodcrest Exec. Dr., Ste. 300
Saint Louis, MO 63141 USA

Limits of Liability and Disclaimer of Warranty

While every precaution has been taken in preparing this material, including research, development and testing, OCI assumes no responsibility for errors or omissions. No liability is assumed by OCI for any damages resulting from the use of this information.

OCI Engineering Training



- 24+ years experience
- Over 50,000 trained
- 150 current courses
- More than 40 instructors on staff
- All training delivered by practitioners and SME's in their respective fields
- Customized to fit your specific needs
- Flexible training delivery
- Training assessments

Instructor

MICRONAUT®

Principal Software Engineer

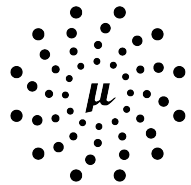
2GM Team member

Contributor to Grails & Micronaut frameworks

@ZacharyAKlein

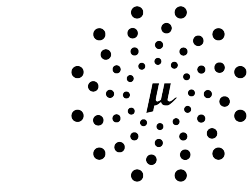
kleinz@objectcomputing.com

Micronaut Team At OCI



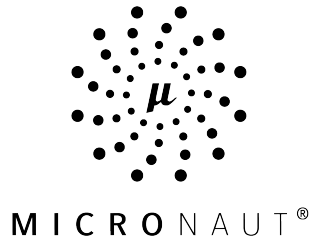
- OCI is Home to the Grails/Micronaut (2GM) Team
  - "Team" Is Singular
  - Not Separate Teams
  - Micronaut framework brought to you by the same folks who brought you the Grails framework
  - Micronaut framework:
    - Development team lead - Sergio Del Amo
    - Micronaut Foundation Technology Advisory Board
    - micronaut.io/foundation

Course Overview

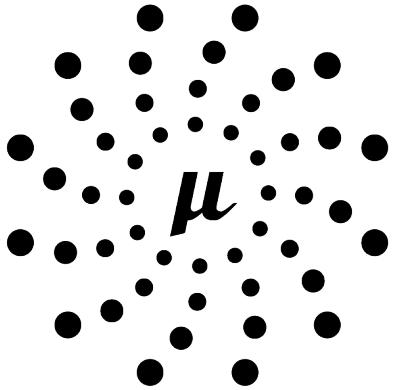


- High Level Agenda:
  - Introduction to the Micronaut framework
  - Core Micronaut Fundamentals
  - Micronaut Data
- Lots of Demos & Exercises!
- Q & A

Requirements



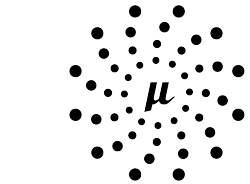
- JDK 11+
- Docker
- Optional: Experience with any Java MVC framework
  - Spring Boot/Grails
  - Play!
  - JSF
  - Etc.



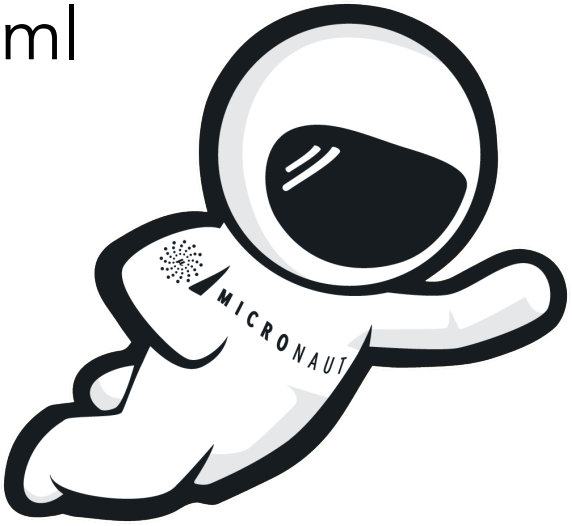
MICRONAUT®

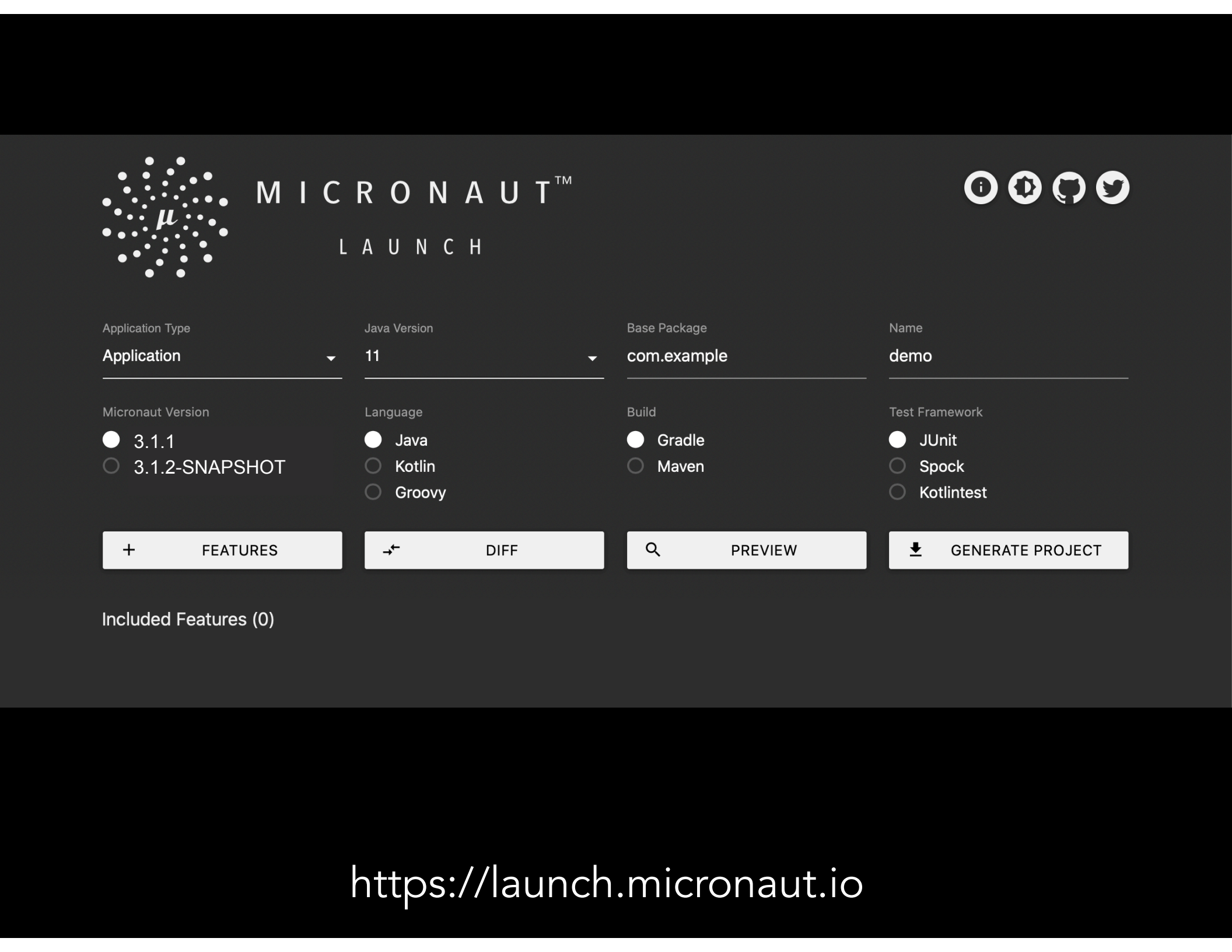
CREATE APP

Creating a Micronaut App



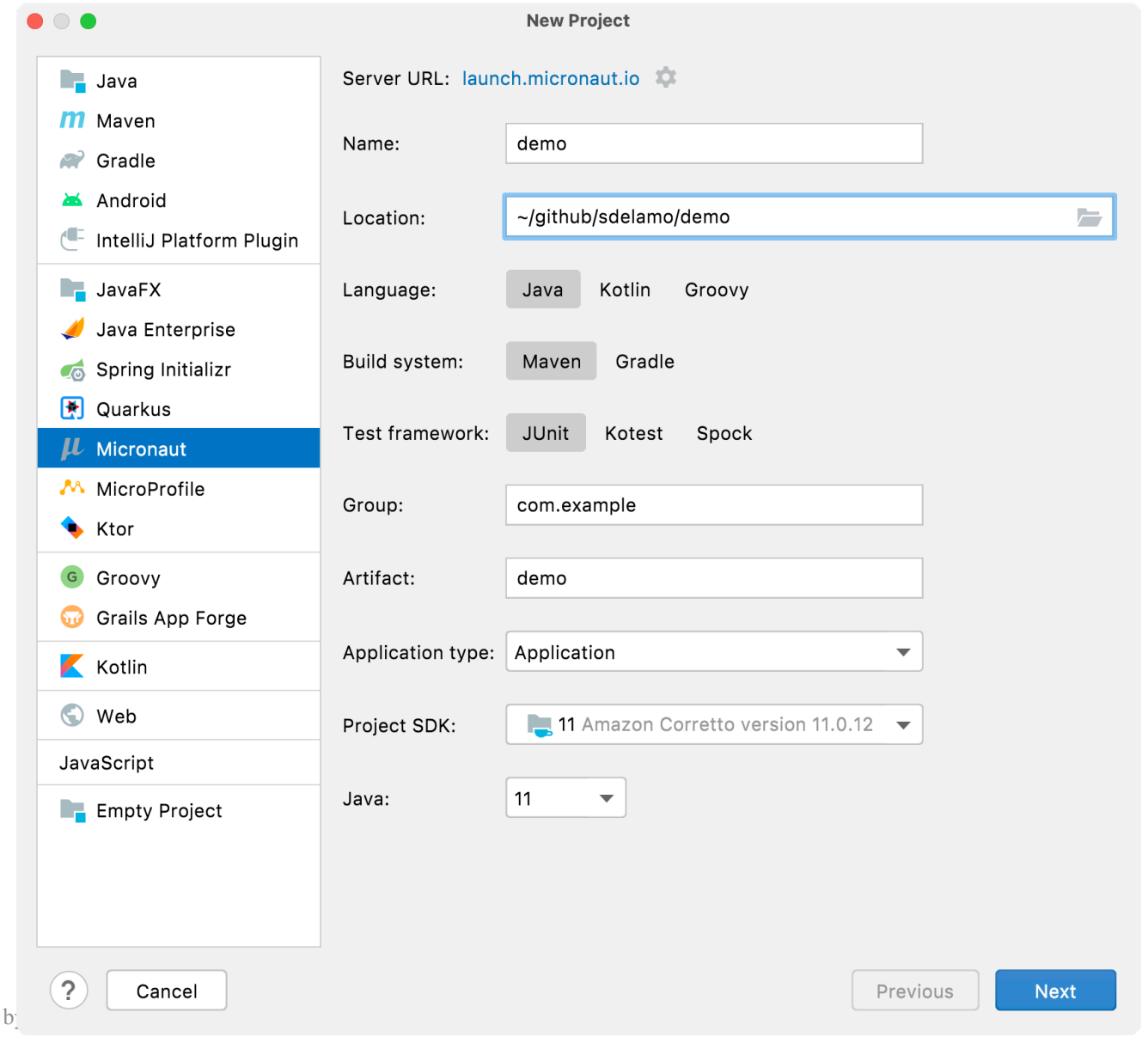
- Micronaut Launch: https://launch.micronaut.io
- SDKMan: https://sdkman.io
- Homebrew/MacPorts
- Binary or Source: https://micronaut.io/download.html

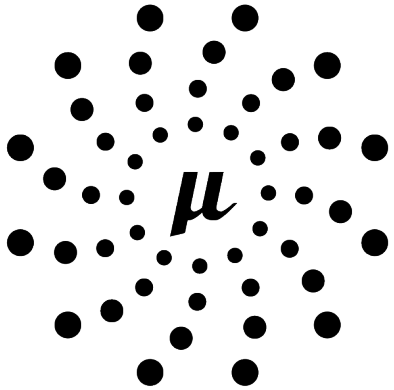




https://launch.micronaut.io

IntelliJ IDEA

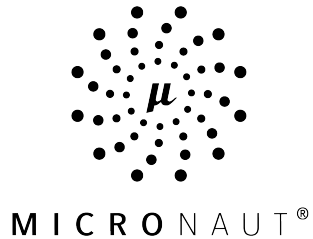




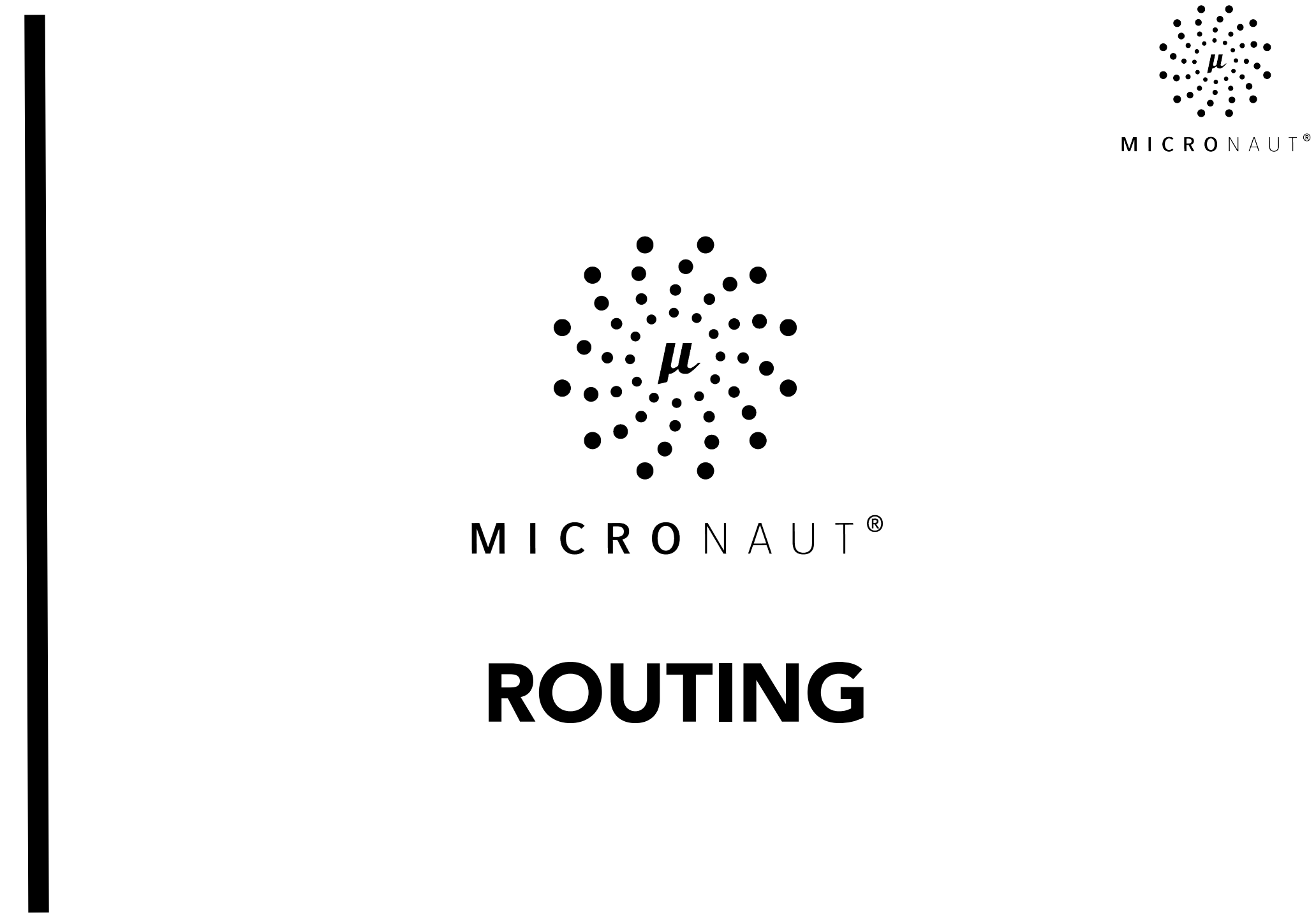
MICRONAUT®

DEVELOPMENT

Hot Reloading



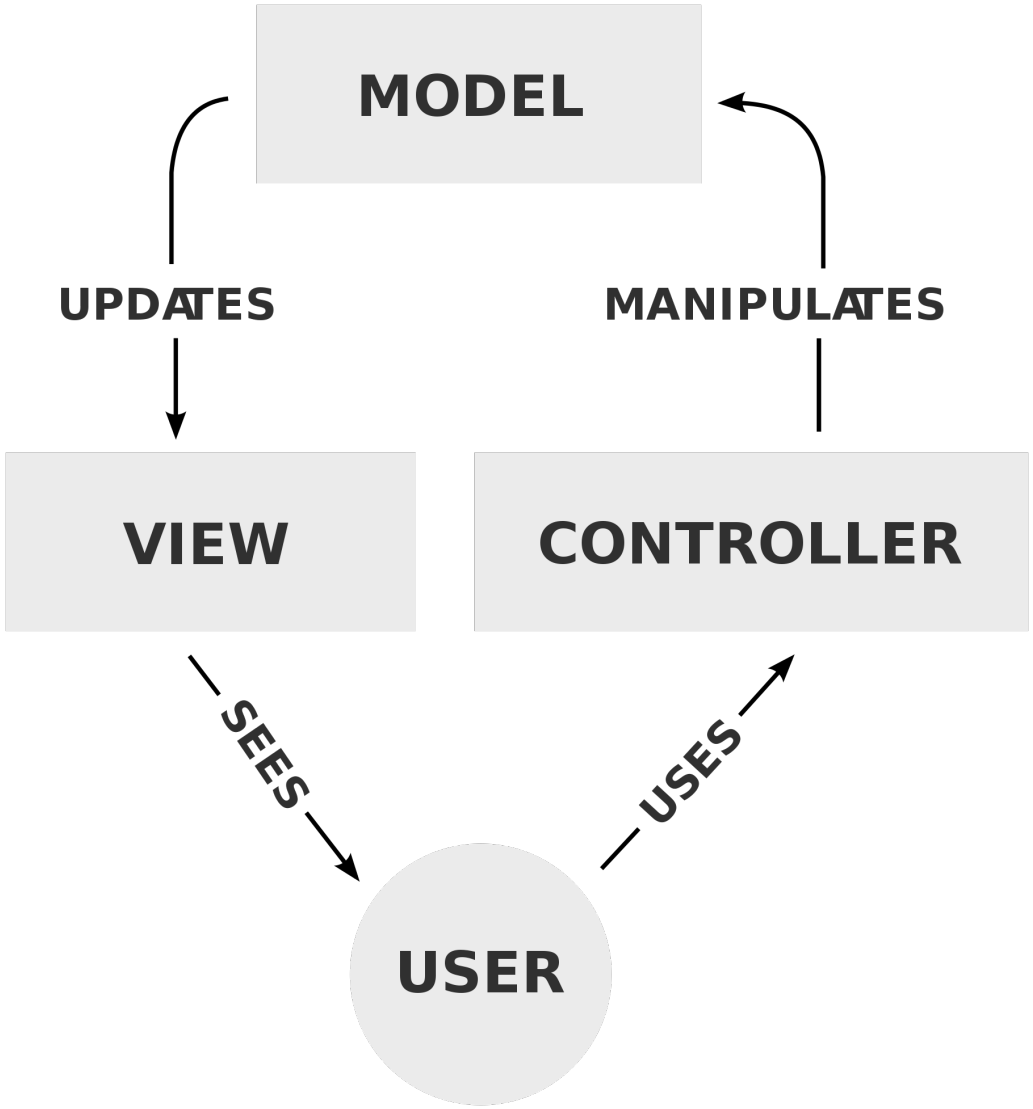
- Gradle via continuous mode (./gradlew -t)
- Maven via Micronaut plugin (new in 2.0)
- Other options are Springloaded (deprecated), JRebel (proprietary)



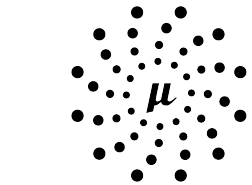
ROUTING

Controllers

MICRONAUT®

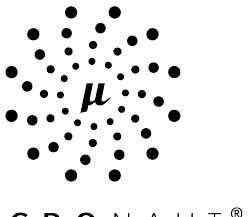


Micronaut Controllers



- Annotated With `@Controller`
- Configured As Beans
  - Subjected To DI
- Conceptually Similar To Controllers In Other Frameworks
  - Grails
  - Spring
  - Etc…

Hello World Controller

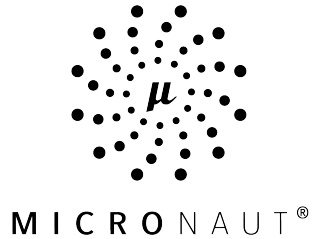



```
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

@Controller
public class HelloController {

    @Get("/hello/{name}")
    public String greet(String name) {
        return "Hello " + name;
    }
}
```

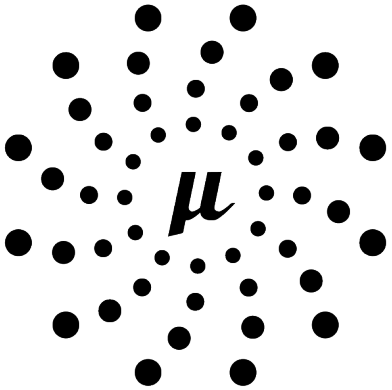
RFC-6570 URI Template
https://tools.ietf.org/html/rfc6570

```
$ curl http://localhost:8080/hello/World
Hello World
```

URI Templates



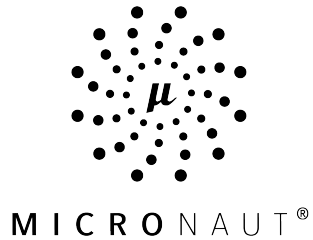
`/books/{id}`	Simple match `Integer id`	`/books/1`
`/books/{id:2}`	A variable of 2 characters max `Integer id`	`/books/10`
`/books{/id}`	An optional URI variable `@Nullable Integer id`	`/books/10` or `/books`
`/book{/id:[a-zA-Z]+}`	An optional URI variable with regex `@Nullable String id`	`/books/foo`
`/books{?max,offset}`	Optional query parameters `@Nullable Integer max`	`/books?max=10&offset=10`
`/books{/path:.*}{.ext}`	Regex path match with extension `@Nullable String path`	`/books/foo/bar.xml`



MICRONAUT®

INTRODUCTION

Introduction To The Micronaut Framework

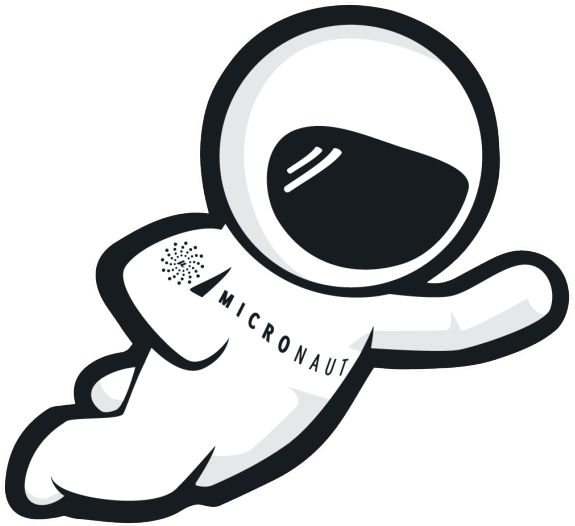


- A framework for microservice & serverless applications
- Leverages Ahead Of Time (AOT) for DI, AOP, and configuration injection
- Reactive HTTP layer built on Netty
- Declarative HTTP Client
- “Natively Cloud Native”
- Accompanied by a suite of official and community-contributed integrations and libraries

What Is The Micronaut Framework?

MICRONAUT®

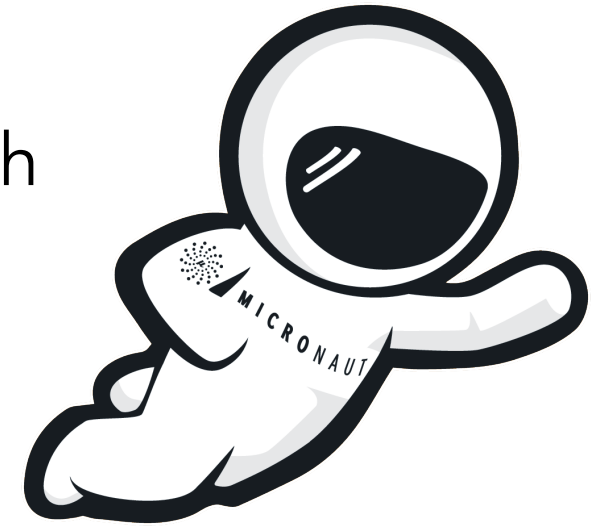
- A Microservices and Serverless-focused framework
- A Complete Application Framework - suitable for any type of Application
- A Set of Libraries providing Dependency Injection, Aspect Oriented Programming (AOP), Configuration Management, Bean Introspection and more..



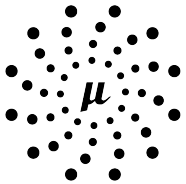
What Can You Build With The Micronaut Framework?

MICRONAUT®

- Microservices
- Serverless Applications
- Message-Driven Applications with Kafka/Rabbit
- CLI Applications
- Android Applications
- Anything with `static void main(String..args)`



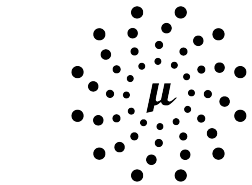
Why Should I Care About Micronaut?



- Java and Java frameworks were not designed with microservices & serverless in mind
- JVM-based DI/AOP solutions make heavy use of reflection, caches, and proxies
- Not optimized for fast cold starts (severless) or low memory usage (microservices/containers)

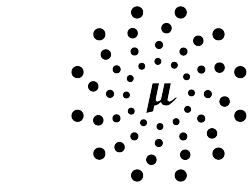
What's Wrong With Reflection?

What Makes The Micronaut Framework Different?



- Ahead of Time (AOT) Compilation
- No Reflection, Runtime Proxies, or Dynamic Classloading
- Optimized for GraalVM (and standard JVM)
- Natively Reactive
- Capable of running in low-memory environments with sub second startup time
- Polyglot: supports Java, Kotlin, and Groovy

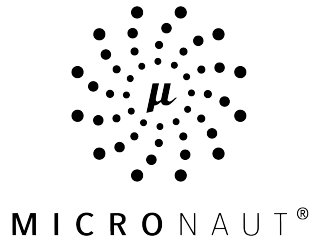
What is a Micronaut App?



- Micronaut-Inject: Core DI, AOP
- Micronaut-Runtime: ApplicationContext, configuration
- HTTP-Server & Client: Reactive HTTP layer*
- A suite of configurations & libraries providing advanced features (security, data access, messaging, etc)

* Since Micronaut 2.0, servlet containers are supported

What is a Micronaut App?

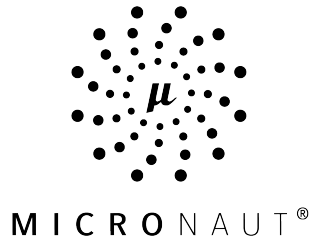


- Micronaut-Inject: Core DI, AOP
- Micronaut-Runtime: ApplicationContext, configuration
- HTTP-Server & Client: Reactive HTTP layer*
- A suite of configurations & libraries providing advanced features (security, data access, messaging, etc)

Can be used in any JVM-based application!

* Since Micronaut 2.0, servlet containers are supported

What is a Micronaut App?

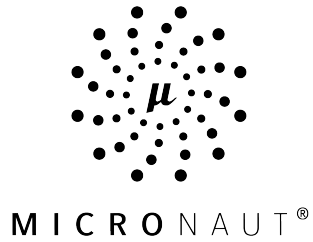


- Micronaut-Inject: Core DI, AOP
- Micronaut-Runtime: ApplicationContext, configuration
- HTTP-Server & Client: Reactive HTTP layer*
- A suite of configurations & libraries providing advanced features (security, data access, messaging, etc)

Included in a "stock" Micronaut Application (via CLI or Launch)

* Since Micronaut 2.0, servlet containers are supported

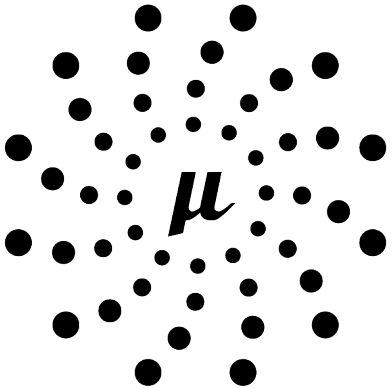
What is a Micronaut App?



- Micronaut-Inject: Core DI, AOP
- Micronaut-Runtime: ApplicationContext, configuration
- HTTP-Server & Client: Reactive HTTP layer*
- A suite of configurations & libraries providing advanced features (security, data access, messaging, etc)

Can be added to your project manually or specified when creating your Micronaut app

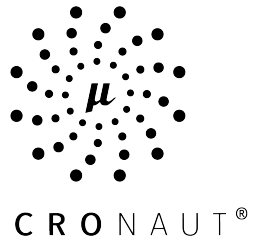
* Since Micronaut 2.0, servlet containers are supported



MICRONAUT®

TESTING

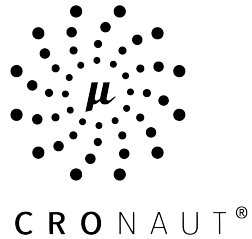
Testing Micronaut



- Micronaut is test-framework agnostic
- No special tooling required for unit or integration/ functional tests
- Simplest approach: start the `ApplicationContext` and test away!

```
try (ApplicationContext context = ApplicationContext.run()) {
    MyBean myBean = context.getBean(MyBean.class);
    // test your MyBean
}
```

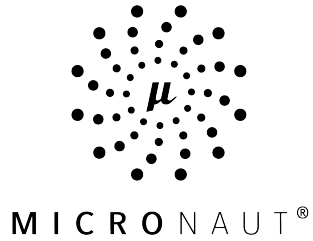
Test Hello Controller



Could Also Write a POJO Unit Test Without Running The Server

```
import io.micronaut.context.ApplicationContext
import io.micronaut.http.client.HttpClient
import io.micronaut.runtime.server.EmbeddedServer
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification

class HelloControllerSpec extends Specification {

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

    @Shared
    HttpClient client = HttpClient.create(embeddedServer.URL)

    void 'test greeting'() {
        expect:
        client.toBlocking().retrieve('/hello/James') == 'Hello James'
    }
}
```

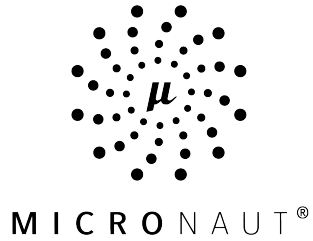
Micronaut Test



- Testing library for Spock & JUnit 5
- Starts up application (inc. embedded server) and shuts it down automatically
- Allows direct injection of beans into test classes
- Set configuration properties with `@Property`

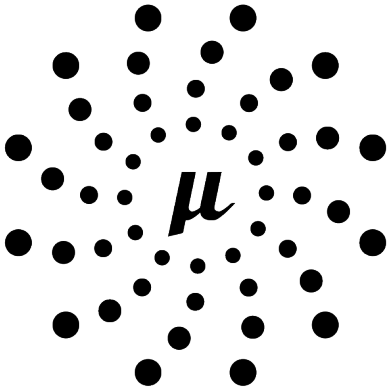
```
@MicronautTest
@Property(name = "greeting.prefix", value = "Hello")
class HelloControllerSpec extends Specification {

    @Inject HelloClient client

    void 'test greeting'() {
        expect:
        client.toBlocking().retrieve('/hello/James') == 'Hello James'
    }
}
```

Test Hello Controller




```
import io.micronaut.http.client.HttpClient
import io.micronaut.http.client.annotation.Client
import io.micronaut.test.annotation.MicronautTest
import spock.lang.Specification

import javax.inject.Inject

@MicronautTest
class HelloControllerSpec extends Specification {

    @Inject
    @Client('/')
    HttpClient client

    void 'test greeting'() {
        expect:
        client.toBlocking().retrieve('/hello/James') == 'Hello James'
    }
}
```

```
testCompile “io.micronaut.test:micronaut-test-spock:$mnTestVersion"
```



MICRONAUT®

DEPENDENCY INJECTION

Dependency Injection

MICRONAUT®

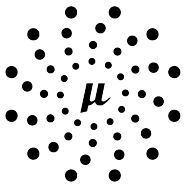
(...for five-year olds)

"When you go and get things out of the refrigerator for yourself, you can cause problems. You might leave the door open, you might get something Mommy or Daddy don't want you to have. You might even be looking for something we don't even have or which has expired.

What you should be doing is stating a need, "I need something to drink with lunch," and then we will make sure you have something when you sit down to eat."

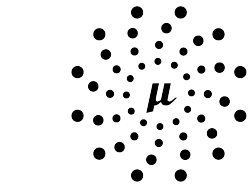
— John Munsch (https://stackoverflow.com/a/1638961)

Dependency Injection



- Micronaut Contains A Full Featured Dependency Injection (DI) Container
- Uses jakarta.inject.* Annotations
- JSR-330 Compliant
  - Passes The Full TCK
- Supports Constructor Injection As Well As Property And Field Injection
  - *insert heated debate here…*

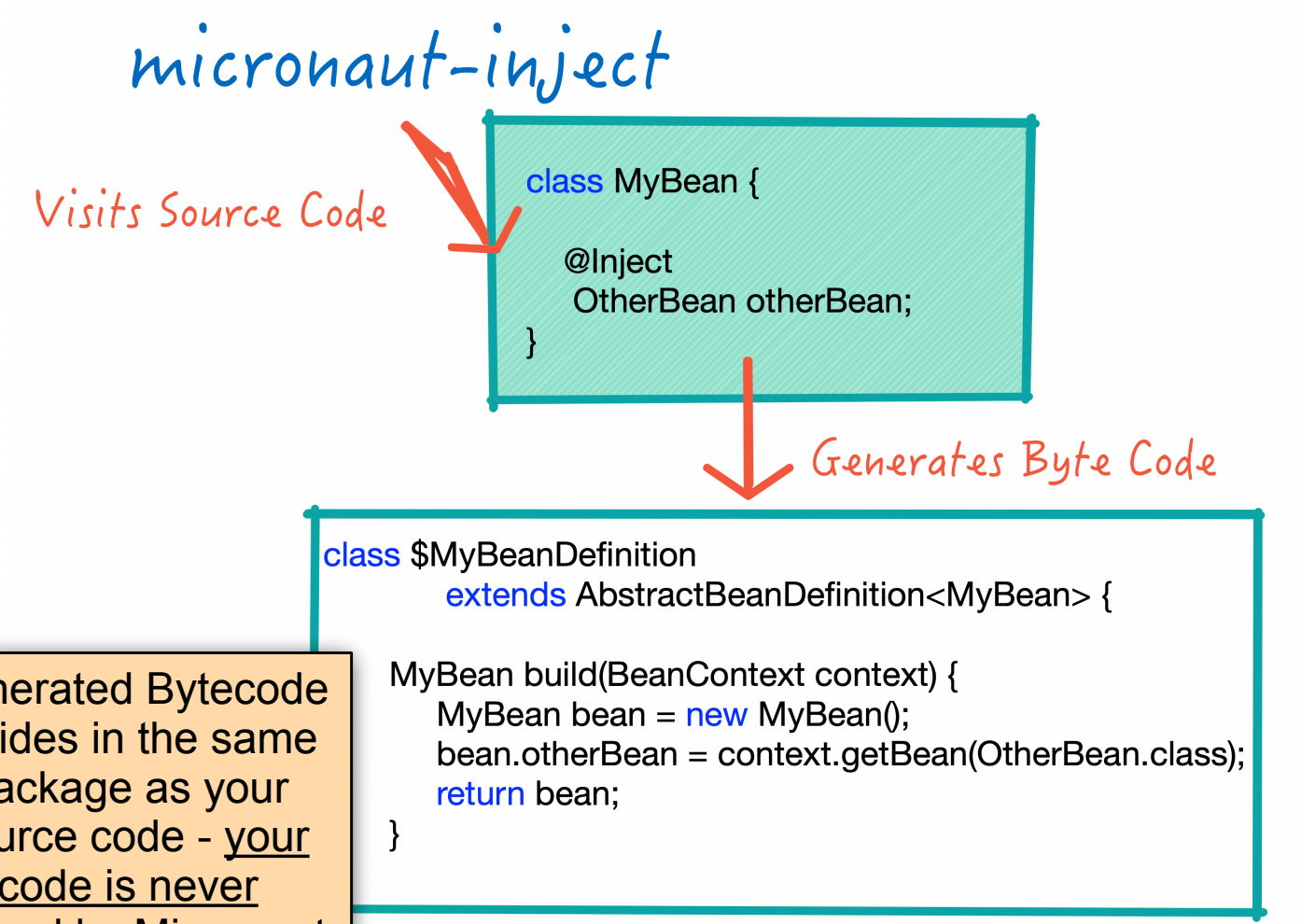
DI Implementation



- Annotations Are Processed At Compile Time
- No Runtime Reflection Unless Necessary
- More Performant At Runtime
- Less Memory Consumption

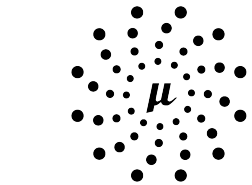
Compile-Time DI

MICRONAUT®



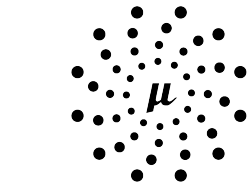
Generated Bytecode resides in the same package as your source code - your code is never altered by Micronaut

Constructor Injection



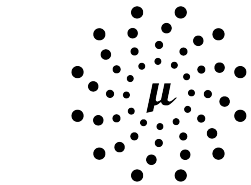

```
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

@Controller
public class HelloController {

    private MessageHelper messageHelper;

    public HelloController(MessageHelper messageHelper) {
        this.messageHelper = messageHelper;
    }

    // ...

}
```

Property Injection



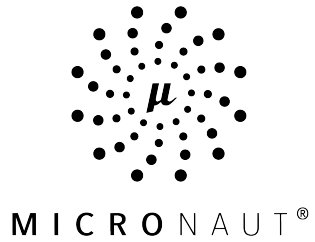

```
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

import javax.inject.Inject;

@Controller
public class HelloController {

    private MessageHelper messageHelper;

    @Inject
    public void setMessageHelper(MessageHelper messageHelper) {
        this.messageHelper = messageHelper;
    }

    // ...
}
```

Field Injection




```
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

import javax.inject.Inject;

@Controller
public class HelloController {

    @Inject
    // Could be private, protected,
    // package-protected or public
    private MessageHelper messageHelper;

    // ...
}
```

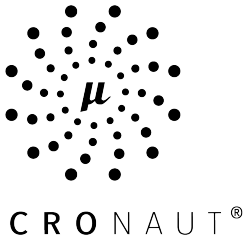
Constructor Injection




```
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;

@Controller("/")
public class HelloController {

    private final MessageHelper messageHelper;

    public HelloController(MessageHelper messageHelper) {
        this.messageHelper = messageHelper;
    }

    // ...

}
```

RECOMMENDED!

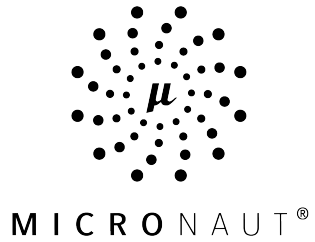
Dependency Injection



- Bean Scopes

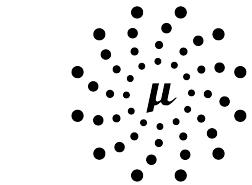
@Singleton	Only one instance of the bean should exist
@Context	Eagerly instantiated
@Prototype	New instance is created each time it is injected
@Infrastructure	Bean cannot be replaced
@ThreadLocal	Bean per thread via a ThreadLocal
@Refreshable	Bean's state to be refreshed via the `/refresh` endpoint
@RequestScope	New instance is created and associated with each HTTP request

Dependency Injection



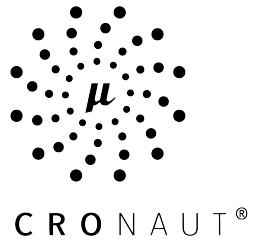
- Should Be Used For All Micronaut Artifacts
- Should Never Create Artifacts Directly
  - ~~JavaHelloController controller = new JavaHelloController();~~
- Most Helpers Should Be Configured As Beans
  - Utilities From 3rd Party Libraries May Require Use Of `@Factory`

Bean Factories




```
import io.micronaut.context.annotation.Bean;
import io.micronaut.context.annotation.Factory;

@Factory
public class SomeFactory {

    @Bean
    SomeThirdPartyUtility anyMethodName() {
        return new SomeThirdPartyUtility();
    }
}
```

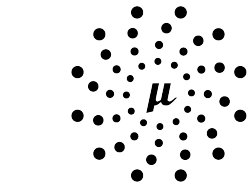
Conditional Beans



- Bean Definitions May Be Marked With `@Requires` To Conditionally Load Them
- Examples:

```
@Requires(classes = org.hibernate.classic.Lifecycle.class)
@Requires(property = “some.property")
@Requires(property = “some.property”, value = "some value")
@Requires(beans = javax.sql.DataSource.class)
@Requires(env = "test")
@Requires(notEnv = "test")
@Requires(sdk = Requires.Sdk.JAVA, version = “1.8")
```

Replaceable Beans

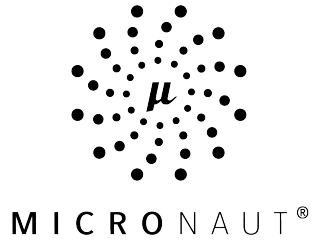


- `@Replaces` May Be Used In Conjunction With `@Requires` To Optionally Replace Beans

```java
@Singleton
@Requires(beans = DataSource.class)
public class JdbcBookService implements BookService {
    // Real BookService Impl...
}
```

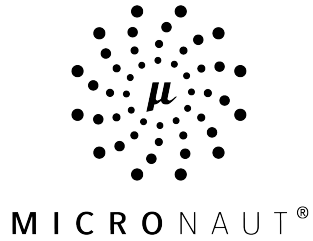
```java
@Singleton
@Requires(env = Environment.TEST)
@Replaces(JdbcBookService.class)
public class MockBookService implements BookService {
    // Mock BookService Impl...
}
```

PostConstruct




```
@Refreshable
public static class WeatherService {
    private String forecast;

    @PostConstruct
    public void init() {
        forecast = "Scattered Clouds "
            + new SimpleDateFormat("dd/MMM/yy HH:mm:ss.SSS”)
            .format(new Date());
    }

    public String latestForecast() {
        return forecast;
    }
}
```

PreDestroy




```
import javax.annotation.PreDestroy;
import javax.inject.Singleton;
import java.util.concurrent.atomic.AtomicBoolean;

@Singleton
public class PreDestroyBean implements AutoCloseable {

    AtomicBoolean stopped = new AtomicBoolean(false);

    @PreDestroy
    @Override
    public void close() throws Exception {
        stopped.compareAndSet(false, true);
    }
}
```