**To create a new Quarkus project using Maven command line, you can use the Quarkus Maven plugin.**

**1. mvn io.quarkus.platform:quarkus-maven-plugin:3.15.1:create \**

**-DprojectGroupId=org.acme \**

**-DprojectArtifactId=my-quarkus-app \**

**-DclassName="org.acme.GreetingResource" \**

**-Dpath="/hello"**

**Explanation of parameters:**

* **io.quarkus.platform:quarkus-maven-plugin:3.15.1:create → The Quarkus plugin for generating the project. Replace 3.15.1 with the latest stable version.**
* **-DprojectGroupId=org.acme → Maven groupId (like a package root).**
* **-DprojectArtifactId=my-quarkus-app → Maven artifactId (the project name).**
* **-DclassName="org.acme.GreetingResource" → Optional. It creates a sample REST endpoint class.**
* **-Dpath="/hello" → Optional. Defines the default REST path for the example endpoint.**

**2. quarkus create app org.acme: my-quarkus-ap**

**Explanation:**

**org.acme:my-quarkus-app → groupId:artifactId of your app.**

**3. Runing Quarkus Project:**

**cd my-quarkus-app**

**./mvnw compile quarkus:dev**

**1. Bootstrap Process Overview**

When you run a Quarkus application (mvn quarkus:dev, ./mvnw quarkus:dev, or a packaged JAR/native binary), the following phases happen:

1. **Bootstrap class starts**
   * The entry point is io.quarkus.runner.GeneratedMain.main(String[] args).
   * This class is generated at build time by Quarkus augmentation.
   * It delegates to io.quarkus.runtime.Quarkus.run(...).
2. **Application Bootstrap**
   * Quarkus.run() calls into QuarkusApplication implementations and sets up the runtime.
   * QuarkusApplication is like CommandLineRunner in Spring Boot, if you define one.
3. **Static Init Phase (Build Time Augmentation)**
   * During the build (mvn package or quarkus:dev startup), Quarkus extensions analyze your code and prepare metadata.
   * Things like CDI bean discovery, JAX-RS endpoints, Hibernate mappings, configuration defaults are processed here.
   * Much of what Spring Boot does at runtime is shifted here to reduce startup time.
4. **Runtime Init Phase**
   * At actual runtime, Quarkus initializes things that can’t be decided at build time:
     + Database connections
     + HTTP server (Vert.x event loop, RESTEasy Reactive, gRPC, etc.)
     + Logging
     + Metrics / Health endpoints
5. **Main Loop Running**
   * The Quarkus runtime stays alive, typically running the HTTP server.
   * Application code (REST controllers, services, event loops, etc.) is now active.
   * If it’s a CLI app, your QuarkusApplication.run() logic executes and then may exit.
6. **Shutdown**
   * Triggered by SIGTERM (e.g., CTRL+C, Kubernetes shutdown).
   * Quarkus.asyncExit() / shutdown hooks clean up beans, stop HTTP server, close DB pools.

**2. Developer Mode (quarkus:dev)**

When you use dev mode:

* Quarkus starts a **hot reload loop**.
* The core runtime runs in one classloader, and your app code runs in another.
* When you change code, Quarkus only reloads the app classloader, keeping the core runtime warm (fast reload).

**3. Native Image**

If you build a native image:

* GraalVM compiles the app into a binary.
* The **static init phase** runs at image build time.
* At runtime, only the lightweight **runtime init phase** happens → resulting in **milliseconds startup**.

**4. Execution Flow (Simplified)**

GeneratedMain.main()

↓

Quarkus.run()

↓

Static Init (build time augmentation)

↓

Runtime Init (start HTTP server, DB pools, CDI contexts)

↓

Your app is running

⚡ So the key difference:

* **Spring Boot**: does almost all bean scanning, reflection, proxies, etc. **at runtime**.
* **Quarkus**: does most of it **at build time**, leaving a very light runtime boot.

**Quarkus modes (dev, build, test)**

Quarkus Dev Mode:

* **Objective:** Optimize for developer experience and rapid iteration.
* **Features:**
  + **Live Reload:** Automatically detects code changes (Java files, resources, configuration) and reloads the application without a full restart, allowing instant feedback.
  + **Dev UI:** A web-based interface providing insights into the running application, configuration, extensions, and more.
  + **Dev Services:** Automatically starts and configures external services (like databases, message queues) needed for development, simplifying setup.
  + **Continuous Testing:** Integrates with your testing framework to automatically re-run tests on code changes.
* **Performance:** Prioritizes developer convenience over runtime performance and footprint. It loads more classes and performs build-time operations during live reloads.

Quarkus Build Mode (Production Application):

* **Objective:**

Optimize for minimal memory consumption, fast startup time, and high performance in production environments.

* **Process:**

Involves compiling the application into an executable JAR or a native executable (using GraalVM).

* **Features:**
  + **Ahead-of-Time (AOT) Compilation:** For native executables, Quarkus compiles the application to a native binary, significantly reducing startup time and memory footprint.
  + **Build-time Optimizations:** Performs various optimizations at build time, such as removing unused classes and methods, resulting in a more efficient and compact application.
  + **No Dev Tools:** Dev UI, Dev Services, and live reload features are not included in the production build, as they are unnecessary and would add overhead.
* **Performance:**

Achieves superior performance, lower memory usage, and faster startup times compared to dev mode.