Comparison Between Spring Boot and Quarkus

This document outlines the key differences between Spring Boot and Quarkus from both functional and technical perspectives. It also provides equivalent settings and annotations in Quarkus for common Spring Boot features.

# 1. Functional Differences

|  |  |  |
| --- | --- | --- |
| Aspect | Spring Boot | Quarkus |
| Philosophy | Provides batteries-included framework for microservices with rich ecosystem. | Optimized for cloud-native and Kubernetes-native applications, with focus on fast startup and low memory. |
| Startup Time | Relatively slower due to reflection-heavy classpath scanning. | Extremely fast (milliseconds) due to build-time optimizations and GraalVM native image support. |
| Memory Usage | Higher footprint (100–200 MB typical). | Lower footprint (~20–50 MB for native). |
| Ecosystem | Rich ecosystem with Spring projects (Spring Data, Spring Security, etc.). | Smaller ecosystem but rapidly growing with Quarkus extensions. |
| Cloud-Native Fit | Works with containers but not optimized for serverless. | Optimized for serverless, Kubernetes, and containers. |
| Developer Experience | Spring Initializr, wide adoption, mature documentation. | Dev mode with hot reload (quarkus:dev), faster feedback loop. |
| Community & Adoption | Very large community, industry standard. | Smaller but fast-growing community with Red Hat support. |

# 2. Technical Differences

|  |  |  |
| --- | --- | --- |
| Aspect | Spring Boot | Quarkus |
| Dependency Injection | @Autowired, @Component, @Service, @Repository | CDI (@Inject, @ApplicationScoped, @Singleton) |
| Configuration | application.properties / application.yml | application.properties (same format) |
| Profiles | spring.profiles.active=dev | %dev.quarkus.datasource.username=... |
| Logging | Logback/SLF4J | JBoss Logging (bridges available) |
| Security | Spring Security framework | Quarkus Security extension |
| Data Access | Spring Data JPA, JDBC, R2DBC | Panache, Hibernate ORM, Reactive SQL clients |
| REST API | @RestController, @RequestMapping | JAX-RS (@Path, @GET, @POST) |
| Validation | @Valid, @NotNull (JSR-303) | Same JSR-303 validation supported |
| Events | @EventListener | CDI Events with @Observes |
| AOP | Spring AOP with proxies | CDI interceptors (@AroundInvoke) |
| Testing | JUnit + Spring TestContext | JUnit + Quarkus JUnit 5 extension |
| Native Compilation | Spring Native (experimental/stable in Spring 3) | First-class GraalVM support |

# 3. Equivalent Settings & Annotations

### Dependency Injection  
- Spring Boot: @Autowired  
- Quarkus: @Inject

### Bean Scopes  
- Spring Boot: @Component, @Service, @Repository  
- Quarkus: @ApplicationScoped, @Singleton, @Dependent

### REST Endpoints  
Spring Boot:  
```java  
@RestController  
@RequestMapping("/hello")  
public class HelloController {  
 @GetMapping  
 public String hello() {  
 return "Hello";  
 }  
}  
```  
Quarkus:  
```java  
@Path("/hello")  
public class HelloResource {  
 @GET  
 public String hello() {  
 return "Hello";  
 }  
}  
```

### Configuration  
Spring Boot:  
```properties  
spring.datasource.url=jdbc:postgresql://localhost:5432/mydb  
spring.datasource.username=admin  
```  
Quarkus:  
```properties  
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/mydb  
quarkus.datasource.username=admin  
```

### Profiles  
Spring Boot:  
```properties  
spring.profiles.active=dev  
```  
Quarkus:  
```properties  
%dev.quarkus.http.port=8081  
%prod.quarkus.http.port=80  
```

### Events  
Spring Boot:  
```java  
@EventListener  
public void onEvent(CustomEvent event) {  
 // handle event  
}  
```  
Quarkus:  
```java  
public void onEvent(@Observes CustomEvent event) {  
 // handle event  
}  
```

### Security  
Spring Boot:  
```java  
@PreAuthorize("hasRole('ADMIN')")  
```  
Quarkus:  
```java  
@RolesAllowed("admin")  
```

# 4. Summary

- Spring Boot = mature, feature-rich, widely adopted.  
- Quarkus = modern, cloud-native, optimized for performance.  
- Many concepts are similar, but Quarkus leans on CDI, JAX-RS, and build-time optimizations.  
- Migration requires mapping annotations and configurations, but most features have direct equivalents.