

# Table of Contents

1. Quarkus Foundation .....	1
1.1. Key Quarkus Capabilities .....	1
1.2. Arc (Quarkus CDI) .....	1
1.3. Native Compilation with GraalVM .....	2
1.4. Extension Ecosystem .....	2
1.5. Quantum's Quarkus Integration .....	2
1.6. Getting Started with Quarkus .....	3
1.7. See Also .....	3

# 1. Quarkus Foundation

Quantum builds on Quarkus, a Kubernetes-native Java stack optimized for fast startup, low memory footprint, and developer productivity.

## 1.1. Key Quarkus Capabilities

### Developer Experience

- **Live reload:** Code changes are reflected immediately in dev mode
- **Unified config:** Single `application.properties` for all configuration
- **Dev UI:** Rich development interface at <http://localhost:8080/q/dev/>
- **Test-first ergonomics:** Built-in test support with `@QuarkusTest`

### Build-time Optimizations

- **Classpath indexing:** Aggressive build-time analysis reduces runtime scanning
- **Ahead-of-time processing:** Framework initialization moved to build time
- **Jandex indexes:** Annotation discovery without runtime reflection

### Cloud Native

- **Container integration:** Seamless Docker and Kubernetes deployment
- **Health checks:** Built-in health and readiness endpoints
- **Metrics:** Micrometer and Prometheus integration
- **Configuration:** Environment-aware config with profiles

## 1.2. Arc (Quarkus CDI)

Arc is Quarkus' CDI implementation, focused on build-time analysis and small runtime overhead.

### Common Scopes

Scope	Usage
<code>@ApplicationScoped</code>	Stateless services, repositories (recommended default)
<code>@RequestScoped</code>	Per-request context holders
<code>@Singleton</code>	Single instance without CDI proxies
<code>@Dependent</code>	Short-lived helpers (default if no scope declared)

### Bean Discovery

- Classes become beans with scope annotations or producer methods
- Use `@Qualifier` to disambiguate multiple implementations

- `Instance<T>` for programmatic bean selection

### Example

```
@ApplicationScoped
public class ProductService {
    @Inject ProductRepo repo;

    public List<Product> findActive() {
        return repo.findByActive(true);
    }
}
```

## 1.3. Native Compilation with GraalVM

### Benefits

- Millisecond startup times
- Drastically reduced memory footprint
- Ideal for serverless and microservices

### Constraints

- Reflection needs explicit configuration
- Dynamic proxies require substitution
- Some dynamic classloading limitations

### Configuration

Use `@RegisterForReflection` for classes accessed via reflection:

```
@RegisterForReflection
@Entity
public class Product extends BaseModel {
    // Ensures reflection metadata in native image
}
```

## 1.4. Extension Ecosystem

Quarkus provides extensions for: - **Data**: MongoDB, PostgreSQL, Redis, Elasticsearch - **Security**: JWT, OIDC, OAuth2, RBAC - **Messaging**: Kafka, AMQP, JMS - **Observability**: Metrics, tracing, health checks - **Cloud**: AWS, Azure, Google Cloud integrations

## 1.5. Quantum's Quarkus Integration

Quantum leverages Quarkus for: - **CDI**: Dependency injection for repositories and services - **JAX-RS**:

REST endpoint framework via BaseResource - **Security:** JWT/OIDC integration for authentication - **MongoDB:** Morphia integration for persistence - **Configuration:** MicroProfile Config for settings - **OpenAPI:** Automatic API documentation

## 1.6. Getting Started with Quarkus

### Create Project

```
mvn io.quarkus:quarkus-maven-plugin:create \
  -DprojectGroupId=com.example \
  -DprojectArtifactId=my-app \
  -DclassName="com.example.MyResource" \
  -Dpath="/hello"
```

### Development Mode

```
./mvnw quarkus:dev
```

### Native Build

```
./mvnw package -Pnative
```

## 1.7. See Also

- [Quarkus Guides](#)
- [CDI Reference](#)
- [Native Image Guide](#)