

EIGENE QUARKUS EXTENSIONS NUTZEN

EINFÜHRUNG VON CDI-LITE

Standardisierung von CDI 4.0 in Jakarta EE 10

Wichtige Änderungen:

- Aufteilung in CDI Lite und CDI Full
- Build-kompatible Erweiterungen
- Änderungen in `beans.xml`

UNTERSCHIEDE

Lite	Full
abgespeckt, nur Features für kleine Runtimes	Lite
Core-Profile	Decorators, Specialization und weitere Scopes
<i>Build Compatible Extensions</i>	Portable Extensions

BUILD COMPATIBLE EXTENSIONS

Neue Erweiterungsart für CDI Lite

Können bereits zur Buildzeit verarbeitet werden

→ Startzeit-Reduktion

Funktionieren ohne Reflection

→ ermöglichen die Ausführung als nativen Code

IMPLEMENTIERUNG

```
public class EmptyBCE implements BuildCompatibleExtension {  
  
    @Discovery  
    public void discovery() { ... }  
  
    @Enhancement(types = Object.class, withSubtypes = true)  
    public void enhancement(...) { ... }  
  
    @Registration(types = Object.class)  
    public void registration(...) { ... }  
  
    @Synthesis  
    public void synthesis() { ... }  
  
    @Validation  
    public void validation() { ... }  
  
}
```

ANWENDUNGSBEISPIEL: STARTUP-SERVICES

Startup-Services direkt instanziiieren

- Buildzeit: Beans mit zusätzlichem Qualifier versehen
- Laufzeit: Qualifizierte Beans instanziiieren

```
public class StartupBCE implements BuildCompatibleExtension {  
  
    @Enhancement(types = Object.class, withSubtypes = true)  
    public void addStartupQualifier(ClassConfig classConfig) {  
        ...  
  
        if (className.startsWith("de.gedoplan.")  
            && className.endsWith("StartupService")) {  
            classConfig.addAnnotation(Startup.class);  
        }  
    }  
}
```

BCE DEMO



WARUM QUARKUS EXTENSIONS?

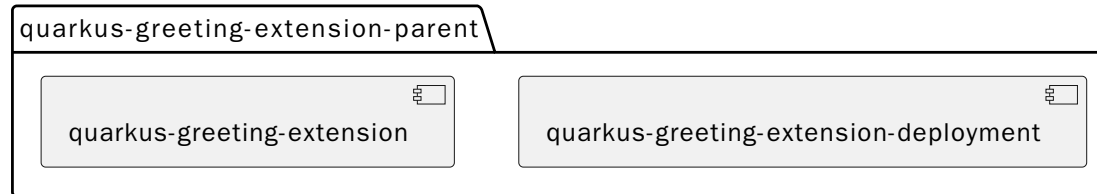
Funktionsergänzung

- Einbindung von Third Party Libraries / Frameworks
- Generieren von zusätzlichem Code
- ...

Quarkus unterstützt keine CDI Portable Extensions

Build Time Augmentation

MULTI-MODULE PROJEKT



Erstellen eines Projektes

```
$ mvn io.quarkus.platform:quarkus-maven-plugin:3.29.4:create-extension \
  -N \
  -DgroupId=de.gedoplan.showcase \
  -DextensionId=quarkus-greeting-extension \
  -DwithoutTests
```

RUNTIME MODULE

Normale Bibliothek mit Quarkus Abhängigkeiten

```
<dependency>  
  <groupId>io.quarkus</groupId>  
  <artifactId>quarkus-arc</artifactId>  
</dependency>  
<dependency>  
  <groupId>io.quarkus</groupId>  
  <artifactId>quarkus-undertow</artifactId>  
</dependency>
```

DEPLOYMENT MODUL

Abhängigkeit zum Runtime-Modul und zu den Deployment-Modulen der Runtime-Dependencies

```
<dependency>
  <groupId>de.gedoplan.showcase</groupId>
  <artifactId>quarkus-greeting-extension</artifactId>
  <version>${project.version}</version>
</dependency>

<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-arc-deployment</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-undertow-deployment</artifactId>
</dependency>
```

QUARKUS BOOTSTRAP

Augmentation

- Möglichkeit Code zur Anwendung hinzuzufügen
- Laden und Scannen des Bytecodes
- Lesen der Konfigurationen
- Scannen von Klassen nach Annotationen
- Alle Anpassungen werden direkt dem Bytecode hinzugefügt

Static Init

- Initialisierung und Konfiguration statischer Ressourcen
- CDI Container verfügbar

Runtime Init

BUILD ITEMS

Während Augmentation aufruf der @BuildStep-Methoden

```
class QuarkusExtensionProcessor {  
  
    @BuildStep  
    FeatureBuildItem feature() {  
        return new FeatureBuildItem("gedoplan-greeting");  
    }  
  
    @BuildStep  
    ServletBuildItem createServlet() {  
        return ServletBuildItem.builder("greeting-servlet",  
            GreetingExtensionServlet.class.getName())  
            .addMapping("/gedoplan/greeting")  
            .build();  
    }  
}
```

RUNTIME

Ein einfaches Servlet, dass zur Laufzeit bereitgestellt wird.

```
public class GreetingExtensionServlet extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
        resp.getWriter().write("Hello");  
    }  
}
```

TEST DER EXTENSION

```
public class GreetingExtensionTest {  
  
    @RegisterExtension  
    static final QuarkusUnitTest sut  
        = new QuarkusUnitTest().withEmptyApplication();  
  
    @Test  
    public void testGreeting() {  
        RestAssured  
            .when().get("/gedoplan/greeting")  
            .then().statusCode(200).body(containsString("Hello"));  
    }  
}
```

CUSTOM EXTENSIONS DEMO



FAZIT

Wann sollte ich eine eigene Extension schreiben?

- Eigentlich immer wenn Funktionalität integriert werden soll
- Optimierung von Ressourcenverbrauch
- Verbesserung der Startzeit

 Beispiele auf GitHub: <https://github.com/GEDOPLAN/quarkus-custom-extensions>