Quarkus-Spring compatibility

While users are encouraged to use Java EE, MicroProfile, or Quarkus annotations/extensions, Quarkus provides an API compatibility layer for some of the Spring projects. If you are coming from Spring development, these integrations might make you a smooth transition to Quarkus.

Important Please note that the Spring support in Quarkus does not start a Spring Application Context nor are any Spring infrastructure classes run. Spring classes and annotations are only used for reading metadata and/or are used as user code method return types or parameter types.

SPRING DI

```
./mvnw quarkus:add-extension
  -Dextensions="quarkus-spring-di"
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfiguration {

   @Bean(name = "capitalizeFunction")
   public StringFunction capitalizer() {
      return String::toUpperCase;
   }
}
```

Or as a component:

```
import org.springframework.stereotype.Component;

@Component("noopFunction")
public class NoOpSingleStringFunction
   implements StringFunction {
}
```

Also as a service and injecting configuration properties from the application.properties file.

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

@Service
public class MessageProducer {

   @Value("${greeting.message}")
   String message;

}
```

And you can inject using Autowired or constructor in a component and in a JAX-RS resource.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
public class GreeterBean {

   private final MessageProducer messageProducer;

   @Autowired @Qualifier("noopFunction")
```

```
    StringFunction noopStringFunction;

    public GreeterBean(MessageProducer messageProducer) {
        this.messageProducer = messageProducer;
    }
}
```

SPRING WEB

```
./mvnw quarkus:add-extension
  -Dextensions="quarkus-spring-web"
```

It supports the REST related features.

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/greeting")
public class GreetingController {

    private final GreetingBean greetingBean;

    public GreetingController(GreetingBean greetingBean) {
        this.greetingBean = greetingBean;
    }

    @GetMapping("/{name}")
    public Greeting hello(@PathVariable(name = "name")
                          String name) {
        return new Greeting(greetingBean.greet(name));
    }
}
```

Supported annotations are: RestController, RequestMapping, GetMapping, PostMapping, PutMapping, DeleteMapping, PatchMapping, RequestParam, RequestHeader, MatrixVariable, PathVariable, CookieValue, RequestBody, ResponseStatus, ExceptionHandler and RestControllerAdvice. org.springframework.http.ResponseEntity is also supported as a return type.

SPRING BOOT CONFIGURATION

```
./mvnw quarkus:add-extension
  -Dextensions="quarkus-spring-boot-properties"
```

```
import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties("example")
public final class ClassProperties {

    private String value;
    private AnotherClass anotherClass;

    // getters/setters
}
```

```
example.value=class-value
example.anotherClass.value=true
```

SPRING SECURITY

```
./mvnw quarkus:add-extension
  -Dextensions="spring-security"
```

You need to choose a security extension to define user, roles, ... such as openid-connect, oauth2, properties-file or security-jdbc. Then you can use Spring Security annotations to protect classes or methods:

```
import org.springframework.security.access.annotation.Secured;

@Secured("admin")
@GetMapping
public String hello() {
   return "hello";
}
```

Quarkus provides support for some of the most used features of Spring Security's @PreAuthorize annotation.
Some examples:

hasRole

@PreAuthorize("hasRole('admin')")

@PreAuthorize("hasRole(@roles.USER)") where roles is a bean defined with @Component annotation and USER is a public field of the class.

hasAnyRole

@PreAuthorize("hasAnyRole(@roles.USER, 'view')")

Permit and Deny All

@PreAuthorize("permitAll()")

@PreAuthorize("denyAll()")

Anonymous and Authenticated

@PreAuthorize("isAnonymous()")

@PreAuthorize("isAuthenticated()")

Expressions

Checks if the current logged in user is the same as the username method parameter:

```
import org.springframework.security.access.prepost.PreAuthorize;

@PreAuthorize("#person.name == authentication.principal.username")
public void doSomethingElse(Person person){}
```

Checks if calling a method if user can access:

```
import org.springframework.security.access.prepost.PreAuthorize;

@PreAuthorize("@personChecker.check(#person, authentication.principal.username)")
public void doSomething(Person person){}

@Component
public class PersonChecker {
   public boolean check(Person person, String username) {
      return person.getName().equals(username);
   }
}
```

Combining expressions:

```
@PreAuthorize("hasAnyRole('user', 'admin') AND #user == principal.username")
public void allowedForUser(String user) {}
```

SPRING DATA JPA

```
./mvnw quarkus:add-extension
  -Dextensions="quarkus-spring-data-jpa"
```

```
import org.springframework.data.repository.CrudRepository;

public interface FruitRepository
      extends CrudRepository<Fruit, Long> {
   List<Fruit> findByColor(String color);
   List<Fruit> findByNameOrderByExpire(String name);
   Stream<Fruit> findNameByNameAndOriginAllIgnoreCase(String name, String origin);
}
```

And then you can inject it either as shown in Spring DI or in Spring Web.
Interfaces supported:

org.springframework.data.repository.Repository

org.springframework.data.repository.CrudRepository

org.springframework.data.repository.PagingAndSortingRepository

org.springframework.data.jpa.repository.JpaRepository.

INFO: Generated repositories are automatically annotated with @Transactional.
Repository fragments are also supported:

```
public interface PersonFragment {
   void makeNameUpperCase(Person person);
}
public class PersonFragmentImpl implements PersonFragment {
   @Override
   public void makeNameUpperCase(Person person) {}
}

public interface PersonRepository
   extends JpaRepository<Person, Long>, PersonFragment {
}
```

User defined queries:

```
@Query("select m from Movie m where m.rating = ?1")
Iterator<Movie> findByRating(String rating);

@Modifying
@Query("delete from Movie where rating = :rating")
void deleteByRating(@Param("rating") String rating);

@Query(value = "SELECT COUNT(*), publicationYear FROM Book GROUP BY publicationYear")
List<BookCountByYear> findAllByPublicationYear2();

interface BookCountByYear {
   int getPublicationYear();

   Long getCount();
}
```

What is currently unsupported:

Methods of org.springframework.data.repository.query.QueryByExampleExecutor

QueryDSL support

Customizing the base repository

java.util.concurrent.Future as return type

Native and named queries when using @Query

SPRING DATA REST

```
./mvnw quarkus:add-extension
  -Dextensions="spring-data-rest"
```

```
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;

@RepositoryRestResource(exported = false, path = "/my-fruits")
public interface FruitsRepository extends CrudRepository<Fruit, Long> {
    @RestResource(exported = true)
    Optional<Fruit> findById(Long id);
    @RestResource(exported = true)
    Iterable<Fruit> findAll();
}
```

The spring-data-jpa extension will generate an implementation for this repository. Then the spring-data-rest extension will generate a REST CRUD resource for it.
The following interfaces are supported:

org.springframework.data.repository.CrudRepository

org.springframework.data.repository.PagingAndSortingRepository

org.springframework.data.jpa.repository.JpaRepository

SPRING CACHE

```
./mvnw quarkus:add-extension
  -Dextensions="spring-cache"
```

```
@org.springframework.cache.annotation.Cacheable("someCache")
public Greeting greet(String name) {}
```

Quarkus provides compatibility with the following Spring Cache annotations:

@Cacheable

@CachePut

@CacheEvict

SPRING SCHEDULE

```
./mvnw quarkus:add-extension
  -Dextensions="spring-scheduled"
```

```
@org.springframework.scheduling.annotation.Scheduled(cron="*/5 * * * * ?")
void cronJob() {
   System.out.println("Cron expression hardcoded");
}

@Scheduled(fixedRate = 1000)
@Scheduled(cron = "{cron.expr}")
```

SPRING CLOUD CONFIG CLIENT

```
./mvnw quarkus:add-extension
  -Dextensions="quarkus-spring-cloud-config-client"
```

```
quarkus.spring-cloud-config.uri=http://localhost:8089
quarkus.spring-cloud-config.username=user
quarkus.spring-cloud-config.password=pass
quarkus.spring-cloud-config.enabled=true
```

```
@ConfigProperty(name = "greeting.message")
String greeting;
```

Possible configuration options. Prefix is quarkus.spring-cloud-config.

uri

Base URI where the Spring Cloud Config Server is available. (default: localhost:8888)

username

Username to be used if the Config Server has BASIC Auth enabled.

password

Password to be used if the Config Server has BASIC Auth enabled.

enabled

Enables read configuration from Spring Cloud Config Server. (default: false)

fail-fast

True to not start application if cannot access to the server. (default: false)

connection-timeout

The amount of time to wait when initially establishing a connection before giving up and timing out. (default: 10S)

read-timeout

The amount of time to wait for a read on a socket before an exception is thrown. (default: 60S)

label

The label to be used to pull remote configuration properties.