Ticker Module Documentation

Table of Contents

1. General

[Compile status] [Maven central version of Parent pom project] [License of Parent pom project] [Use JakartaEE project] [Commits] [Supported jvms] [GitHub Repo Stars]

The goal of the project is to provide a modular solution for triggering processes by calling specific API endpoints using cron scheduling.

Components

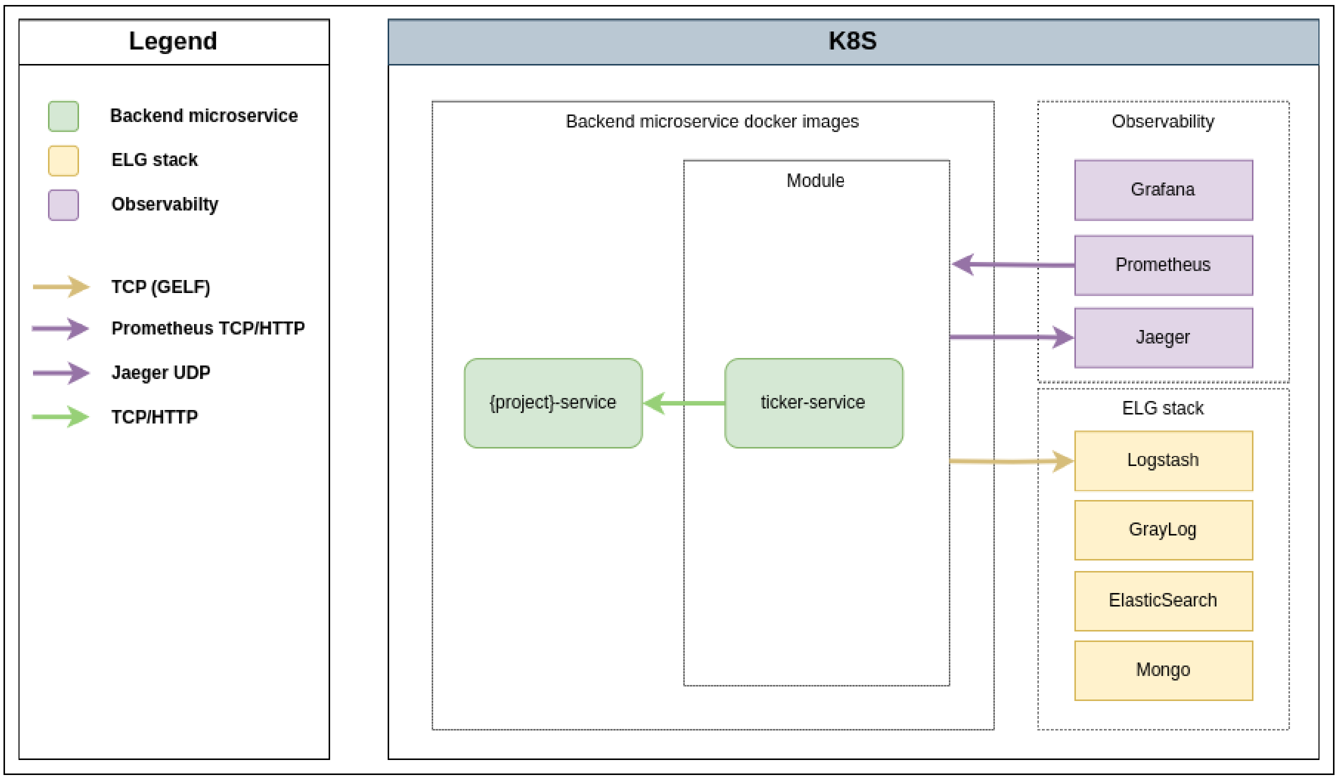

Figure 1. Architecture

Table 1. Ticker backend {page-version}

Type	Module	Version	maven
External			
Internal	Coffee	-	2.9.0

2. Backend

The Ticker project can be divided into two parts. One is a sampler and its associated testsuite subproject, and the other part is the ticker-core-quartz module.

2.1. Service

The sampler demonstrates how to use the module to create a service where you only need to link the **apit** at the dependency level with the **ticker-core-quartz module**.

The first step is to include the ticker-core-quartz module in the dependencies.

```xml
<dependency>
    <groupId>hu.icellmobilsoft.ticker</groupId>
    <artifactId>ticker-core-quartz</artifactId>
</dependency>
```

Secondly, use the API dependency containing the interfaces that the module will call:

Example API dependency

```xml
<dependency>
    <groupId>hu.icellmobilsoft.sampler.api</groupId>
    <artifactId>api</artifactId>
</dependency>
```

It is important that since this is a Quarkus-based module, the implementing module must also use an application with the same version of Quarkus.

After that, when the application starts, the event defined in the `SchedulerController` will be triggered.

At this point, the `SchedulerController` will see the indexed interfaces and will start them according to the provided yml config.

2.2. Ticker sampler

Mock GET Rest endpoint

```
GET /test
```

Mock POST Rest endpoint

```
POST /test
```

The two mock endpoints serve as examples to test the module; there is no functionality behind them.

3. Configuration

3.1. Backend

The module configuration is done through application.yml, which allows specifying the necessary values based on Quarkus.

The basic configuration is provided through application.yml, which can be expanded and may vary by environment, but you only need to specify values that differ from the default settings.

Possible methods in order of priority:

- System variables
- Environment variables

3.2. Backend

The module's configuration is managed via MicroProfile Config, which allows specifying necessary values in multiple ways.

MicroProfile Config can retrieve a given key from all available sources and uses the highest priority value.

The base configuration is provided via `project-defaults.yml`, which can be extended and may vary per environment. It's not necessary to specify every value; only those that differ from the default settings.

Possible modes in order of priority:

- System variables
- Environment variables

3.3. Job Configuration

The jobs to be executed are configured in the `application.yml` file.

> Basic configurations are set at the `microprofile-config.yml` level in the `ticker-core-quartz` dependency, so this file should not be overridden in the implementing project.

application.yml - Defining Quarkus REST clients

```
quarkus:
    rest-client:
        ITickerTestRestRegisteredClient: ①
```

```
          url: http://localhost:8080 ②
          scope: jakarta.enterprise.context.ApplicationScoped ③
          read-timeout: 5000 ④
          connect-timeout: 5000 ⑤
```

> It is a general solution in Quarkus that the definitions of REST clients to be used in the application can be specified in this way.

The example contains the most basic configurations:

① Under the quarkus.rest-client config, the identifier of the REST client interface must be given, which is the interface with its package, or if the configKey is provided in the `@RegisterRestClient` annotation, then it should be referenced here based on that.

② This is the place to specify the baseUri.

③ Determination of the scope of the injected RestClient interface. It's recommended to use ApplicationScoped, as it can avoid many issues (rest client closure, memory leaks, etc.); be careful not to try to destroy the bean if multiple jobs use the interface, as it will lead to errors. Increasing the thread pool is recommended if many jobs need to run on the same REST client.

④ The response waiting timeout can be defined.

⑤ The connection timeout can be defined.

For increasing the thread pool mentioned in point 3, the `DefaultRestClientBuilderListener` class should be inherited and the `onNewBuilder` method overridden, where the thread pool and the max-pooled-per-route parameters can be adjusted:

```
    @Override
    public void onNewBuilder(RestClientBuilder builder) {
        super.onNewBuilder(builder);
        builder.property("resteasy.connectionPoolSize",75);
        builder.property("resteasy.maxPooledPerRoute",50);
```

For further options, see: https://download.eclipse.org/microprofile/microprofile-rest-client-3.0/microprofile-rest-client-spec-3.0.html

application.yml - Defining the list of Jobs

```
ticker:
    timer:
        activeJobs:
            - TEST_REST
            - TEST_REST_2
            - ..
```

3.3.1. Using Microprofile Rest-based Job

The job allows HTTP calls to be initiated with the help of an MP REST client.

The following example demonstrates its use:

application.yml

```
ticker:
    timer:
        activeJobs:
            - TEST_REST ①
        job:
            TEST_REST: ②
                code: REST_QUARTZ_TEST ③
                cron: "*/10 * * ? * *" ④
                actionClass:
hu.icellmobilsoft.ticker.quartz.service.timer.job.mprestclient.MicroprofileRestClientJ
ob ⑤
                config:
                    mpRestClientClass:
hu.icellmobilsoft.ticker.sample.service.rest.test.api.ITickerTestRestRegisteredClient
⑥
                    method:
getTest(java.lang.String,java.lang.Integer,java.lang.Long,java.lang.Boolean,java.time.
OffsetDateTime) ⑦
                    parameters: ⑧
                        - config
                        - 50
                        - 30
                        - true
                        - 2023-06-07T13:45:27.893013372Z
```

① - under ticker.timer.activeJobs, you can define which job should be active

② - under ticker.time.job, the jobs referenced in the first point can be defined

③ - Using the code, you can search for the given job in the logs. It’s found in Health as <code>-Job, and in logs like: `<<< End quartz job type [REST_QUARTZ_TEST job]`.

④ - Cron settings

⑤ - Action class that defines the job’s process.

⑥ - The action configuration where the mpRestClientClass can be specified, which is the REST client interface.

⑦ - The action configuration where the method can be specified within the REST client interface.

⑧ - The action configuration where the parameters for the method call can be specified. Any list element can be defined with a static method call using { at the beginning and } at the end.

3.3.2. Using HTTP Call-based Job

The job allows for the definition of basic HTTP calls. The only task to be solved is if a custom body setup is needed, for example for a POST call, which can be solved with the method definition so far.

The following example demonstrates its use:

application.yml

```
ticker:
    timer:
        activeJobs:
            - TEST_APACHE_HTTP_CLIENT ①
        job:
            TEST_APACHE_HTTP_CLIENT: ②
                code: TEST_APACHE_HTTP_CLIENT ③
                cron: "*/1 * * ? * *" ④
                actionClass:
hu.icellmobilsoft.ticker.quartz.service.timer.job.httpclient.HttpClientJob ⑤
            config:
                baseUrl: http://localhost:8080/test/ticker ⑥
                method: Get ⑦
                body:
"&{hu.icellmobilsoft.ticker.common.util.version.BaseRequestUtil.generate}" # static
method call ⑧
                headers:
                    Content-Type: "application/xml"
                    Accept: "application/json"
                queryParams:
                    testString: value
                    testInteger: 1000
                    testLong: 50000
                    testBoolean: true
                    testOffsetDateTime: 2023-06-07T13:45:27.893013372Z
```

① - under ticker.timer.activeJobs, you can define which job should be active

② - under ticker.time.job, the jobs referenced in the first point can be defined

③ - Using the code, you can search for the given job in the logs. It’s found in Health as <code>-Job, and in logs like: `<<< End quartz job type [TEST_APACHE_HTTP_CLIENT job]`.

④ - Cron settings

⑤ - Action class that defines the job’s process, in this example, one can use the HTTP Call-based job.

⑥ - The action configuration where the baseUrl can be specified for the HTTP call

⑦ - The action configuration where the method can be specified for the HTTP call

⑧ - The action configuration where the body can be specified for the HTTP call. A static method call can also be defined in the body using &{ at the beginning and } at the end.

⑨ - The action configuration where the headers can be specified for the HTTP call

⑩ - The action configuration where the queryParams can be specified for the HTTP call

4. Installation, Deployment

4.1. General Service Settings

4.2. Ticker core quartz service configuration

The Ticker Service must be accessible in the environments of the project(s) that intend to use it. To achieve this, each instance of the service - along with its infrastructural requirements - must be deployed and configured for each (development/test/production) environment.

4.2.1. Service Configuration

Application

```
COFFEE_APP_NAME=${quarkus.application.name}
COFFEE_CONFIG_XML_CATALOG_PATH=xsd/hu/icellmobilsoft/cfg/dto/super.catalog.xml
COFFEE_CONFIG_RESOURCE_BUNDLES= i18n.common-messages,i18n.messages
CONSOLE_LOGGING_ENABLED=true
```

4.2.1.1. Kubernetes deployment

- Recommended configuration

Parameter	Value	Description
TICKER_LOG_CONSOLE_ENABLE	true	disable logging to console, default: true
TICKER_LOG_FILE_ENABLE	false	disable logging to file, default: false
TICKER_LOGSTASH_K8S_NAMESPACE	ticker-core-quartz-service	set K8S_NAMESPACE, default unknown
CFG_LOGSTASH_MODULE_VERSION		set moduleVersion key, default unknown
TICKER_JAEGER_AGENT_HOST_PORT	jaeger:6831	jaeger agent host, default localhost
TICKER_JAEGER_SERVICE_NAME	ticker-core-quartz-service	Service name visible on the Jaeger interface (default ROOT.war)

4.2.2. Quarkus based configs

Since the application is Quarkus based, the default Quarkus settings can be used in it.

The description can be found here: https://quarkus.io/version/3.15/guides/all-config

From the configuration list, only those elements are active that are included in the project at the dependency level.

Important elements that are already defined by default with the project:

Quarkus config key	Description	Env variable	Default value
quarkus.arc.remove-unused-beans	Arc setting - remove unused beans: Link	-	false
quarkus.log.category."hu.icellmobilsoft".level	hu.icellmobilsoft category log level	TICKER_LOG_HU_ICELLMOBILSOFT_LEVEL	INFO
quarkus.log.console.enable	Console logging enable	TICKER_LOG_CONSOLE_ENABLE	true
quarkus.log.console.json	Json logging enable	TICKER_LOG_CONSOLE_JSON_ENABLED	false
quarkus.log.console.json.additional-field."moduleVersion".value	JSON log - **moduleVersion** additional-field value	TICKER_JSON_MODULE_VERSION	<POM_VERSION> Set in dockerfile. It is preferred for projects implementing ticker-core-quartz and building their own image to set it as well.
quarkus.log.console.json.additional-field."moduleId".value	JSON log - **moduleId** additional-field value	TICKER_JSON_MODULE_ID	<POM_ARTIFACT_ID> Set in dockerfile. It is preferred for projects implementing ticker-core-quartz and building their own image to set it as well.
quarkus.log.console.json.additional-field."K8S_NAMESPACE".value	JSON log - **K8S_NAMESPACE** additional-field value	TICKER_JSON_K8S_NAMESPACE	unknown
quarkus.log.console.json.additional-field."JSON_MDC_FIELDS".value	JSON log - **JSON_MDC_FIELDS** additional-field value	TICKER_JSON_MDC_FIELDS	none
quarkus.log.console.json.exception-output-type	JSON log - The exception output type to specify Link	TICKER_JSON_EXCEPTION_OUTPUT_TYPE	formatted
quarkus.log.console.json.date-format	JSON log - The date format to use Link	TICKER_JSON_DATE_FORMAT	`yyyy-MM-dd HH:mm:ss,SSSZ`

Quarkus config key	Description	Env variable	Default value
quarkus.log.console.format	Console log format	-	`%d{yyyy-MM-dd HH:mm:ss.SSS} %-5p [traceId=%X{traceId}] [spanId=%X{spanId}] [thread:%t] [%c{10}] [sid:%X{extSessionId}] - %s%E%n`
quarkus.log.handler.gelf.additional-field."moduleVersion".value	Gelf log - **moduleVersion** additional-field value	TICKER_LOGSTASH_MODULE_VERSION	<POM_VERSION> Set in dockerfile. It is preferred for projects implementing ticker-core-quartz and building their own image to set it as well.
quarkus.log.handler.gelf.additional-field."moduleId".value	Gelf log - **moduleId** additional-field value	TICKER_LOGSTASH_MODULE_ID	<POM_ARTIFACT_ID> Set in dockerfile. It is preferred for projects implementing ticker-core-quartz and building their own image to set it as well.
quarkus.log.handler.gelf.additional-field."K8S_NAMESPACE".value	Gelf log - **K8S_NAMESPACE** additional-field value	TICKER_LOGSTASH_K8S_NAMESPACE	unknown
quarkus.handler.gelf.include-full-mdc	Gelf log - Whether to include all fields from the MDC.	-	false
quarkus.log.level	Quarkus log level: Link	TICKER_LOG_LEVEL	INFO
quarkus.log.min-level	Quarkus min log level: Link	TICKER_LOG_MIN_LEVEL	ALL
quarkus.otel.enabled	OpenTelemetry - enabled config: Link	QUARKUS_OTEL_ENABLED	false
quarkus.otel.traces.exporter	OpenTelemetry trace exporter - : Link	QUARKUS_OTEL_TRACES_EXPORTER	jaeger
quarkus.otel.exporter.otlp.traces.endpoint	OpenTelemetry endpoint - : Link	QUARKUS_OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	http://localhost:4317
quarkus.package.jar.add-runner-suffix	Quarkus package add runner suffix: Link	-	false
quarkus.package.jar.type	Quarkus package JAR type: Link	-	uber-jar

Quarkus config key	Description	Env variable	Default value
quarkus.quartz.clustered	Quartz - clustered : Link	-	false
quarkus.quartz.thread-count	Quartz - thread count : Link	TICKER_QUARTZ_THREAD_COUNT	25
quarkus.scheduler.start-mode	Quartz - start mode : Link	-	FORCED
quarkus.smallrye-openapi.info-title	Openapi - info title : Link	-	Ticker service
quarkus.smallrye-openapi.info-version	Quartz - info version : Link	-	${quarkus.application.version}
quarkus.smallrye-openapi.info-description	Quartz - info version : Link	-	`REST endpoints for operations. <br/> General responses in case of error: <br/> * __400__ - Bad Request <br/> * __401__ - Unauthorized <br/> * __404__ - Not found <br/> * __418__ - Database object not found <br/> * __500__ - Internal Server Error <br/>`
quarkus.swagger-ui.enable	Enable swagger ui: Link	-	false

4.2.2.1. Observability

Metrics

The `hu.icellmobilsoft.ticker.quartz.service.quartz.util.QuartzJobUtil` class provides metrics about Quartz Jobs. We add our own `hu.icellmobilsoft.ticker.quartz.service.quartz.health.metric.MetricJobListener`, which implements `org.quartz.JobListener`, to the `org.quartz.Scheduler` via the `org.quartz.ListenerManager` interface.

Currently, the following Quartz Job metrics are available:

- Quartz job prev fire time
  - The time of the previous Job execution
  - key: `quartz_job_prev_fire_time`
- Quartz job next fire time
  - The time of the next Job execution
  - key: `quartz_job_next_fire_time`
- Quartz job run time
  - The duration of the most recent Job execution
  - key: `quartz_job_run_time`

The application server provides metrics at the `<host:port>/q/metrics` endpoint with the `application_` prefix.

example

```
application_quartz_job_prev_fire_time{configKey="REST_QUARTZ_TEST-Job",quantile="0.5"}
1.66921282E12
application_quartz_job_next_fire_time{configKey="REST_QUARTZ_TEST-Job",quantile="0.5"}
1.66921283E12
application_quartz_job_run_time{configKey="REST_QUARTZ_TEST-Job",quantile="0.5"} 41.0
```

Health - startup/liveness/readiness

The service supports the use of k8s probes.

- started http://localhost:9000/q/health/started
- live http://localhost:9000/q/health/live
- ready http://localhost:9000/q/health/ready

4.3. Helm Config Guidelines

Quarkus supports the dev and test profiles and allows the creation of other profiles. However, the values of configurations, such as a URL, can vary across different environments.

Therefore, you need to set a custom microprofile-config.yml file at the Helm config level to override the configurations in the application.

Quarkus provides the opportunity to set config sources via environment variables: https://quarkus.io/guides/config-reference#quarkus-config-config_quarkus.config.locations

So, for helm values, the following should be set:

values.yaml

```
configMountPath: /deployments/app/config
...
```

```
extraEnv:
- name: QUARKUS_CONFIG_LOCATIONS
value: {{ .Values.configMountPath }}/microprofile-config.yml
```

5. Additional Information

5.1. Useful Commands and Accesses

Commands used for development purposes, which are used for setting up and starting developer environments.

The application can be started in several ways:

- Starting Quarkus dev with Maven
- Creating a Quarkus uber-jar and running this jar file using `java -jar`
  - The same jar placed into a java docker image and run (using Dockerfile.uber-jar.jvm)

Docker-compose is used for creating and running Docker images.

The project contains a sampler service that demonstrates the use of the module. This example is capable of running entirely on a local development machine. So there are no external dependencies.

5.1.1. Starting ticker-core-quartz-service Server in Different Ways

IDE included Quarkus run config

```
Several IDEs offer native support for Quarkus, as they do for Spring Boot projects,
recognizing and creating their own run configuration.
```

Maven quarkus:dev

```
mvn clean compile quarkus:dev
```

The project consists of more than one module, as expected by Quarkus, therefore compile is necessary.

With the help of the Quarkus Maven plugin, the project can be started in dev mode, activating several dev tools. More information: https://quarkus.io/guides/dev-mode-differences.

Running Quarkus uber-jar in Docker

```
mvn clean install ①
docker-compose -f <PROJECT_PATH>/ticker-backend/etc/docker-compose/docker-
```

```
compose.local.ticker-service.yml up --build --force-recreate ②
```

① is necessary for generating the jar that will be included in the Docker image.

② The docker compose command, issued in the project's root, initiates the docker-compose build (forcing a rebuild of the image if needed with the force recreate parameter), and starts up.

> Quarkus processes and optimizes beans at **build-time**, unlike traditional runtime dependency injection models. Thanks to this: Only the actually used classes are included in the final application, while unnecessary ones are automatically removed.

> MP Rest Client can only be configured at **build-time**, they cannot be dynamically registered or injected at runtime, therefore at build-time it can be packaged with a `Ticker core` module.

5.2. Validating job configuration existence with test

We provide an abstract test class which can be extended on project side to validate that all expected jobs have been configured.

Test scoped dependency

```xml
<dependency>
    <groupId>hu.icellmobilsoft.ticker</groupId>
    <artifactId>ticker-core-quartz</artifactId>
    <version>${version}</version>
    <classifier>tests</classifier>
    <scope>test</scope>
</dependency>
```

Using of AbstractQuarkusTest class

```java
@QuarkusTest ①
public class ProjectQuarkusTestIT extends AbstractQuarkusTest { ②

    @Override
    public List<String> getExpectedJobKeys() { ③
        return List.of("...", "...");
    }
}
```

① add the QuarkusTest annotation

② extend AbstractQuarkusTest class

③ override the getExpectedJobKeys method to return with the key list

- The expected job keys are the `ticker.timer.job.[jobName].code` configurations.
- The expected jobs should be active as well, thus listed in `ticker.timer.activeJobs`

configuration.

5.3. Validating job configuration on startup

If the `ticker.config.validation` configuration is set to true, the job configuration will be validated at startup, and the application will fail to start in case of any errors.

Validations:

- at least one job must be configured in `ticker.timer.activeJobs`
- the cron expression configured in `ticker.timer.job.[jobName].cron` must be valid
- the action class configured in `ticker.timer.job.[jobName].actionClass` must exist and implement the `org.quartz.Job` interface
- in case of a `MicroprofileRestClientJob`:
  - the class configured in `ticker.timer.job.[jobName].config.mpRestClientClass` must exist
  - the method configured in `ticker.timer.job.[jobName].config.method` must be callable

6. Release notes

6.1. ticker 1.3.0

6.1.1. Changes

- 🎉 Establishment of the open source project

6.1.2. Migration

The changes are backward compatible and do not require migration.

6.2. ticker 1.4.0

6.2.1. Changes

- GH documents translated to english.

6.2.1.1. `coffee` version upgrade `2.6.0` → `2.9.0`:

- 2.6.0 → 2.7.0
- 2.7.0 → 2.8.0
- 2.8.0 → 2.9.0

Migration

The changes are backward compatible and do not require migration.

6.2.1.2. `roaster` version upgrade `2.1.0` → `2.5.0`:

- 2.1.0 → 2.2.0
- 2.2.0 → 2.3.0
- 2.3.0 → 2.4.0
- 2.4.0 → 2.5.0

Migration

The changes are backward compatible and do not require migration.

6.2.1.3. Quarkus version upgrade `3.2.5.Final` → `3.15.3`

- Observability: OpenTracing → OpenTelemetry
- Quarkus configuration changes

Migration

- Observability Jaeger configuration:
  - Enable OTLP collector and set port, see `docker-compose.local.observability.yml`
- MicroProfile (`microprofile-config.yml`) and environment variables (`docker-compose.local.ticker-service.yml`):
  - `quarkus.jaeger.enabled` → `quarkus.otel.enabled` (`QUARKUS_OTEL_ENABLED`)
  - new `quarkus.otel.traces.exporter` (`QUARKUS_OTEL_TRACES_EXPORTER`)
  - `quarkus.jaeger.endpoint` → `quarkus.otel.exporter.otlp.traces.endpoint` (`QUARKUS_OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`)
  - `quarkus.package.add-runner-suffix` → `quarkus.package.jar.add-runner-suffix`
  - `quarkus.package.type` → `quarkus.package.jar.type`
  - more details: https://quarkus.io/version/3.15/guides/all-config

6.2.2. Bug Fixes

6.2.2.1. `docker/bake-action` version upgrade: `v4.3.0` → `v5`

Migration

The changes are backward compatible and do not require migration. It does not affect application functionality, only the docker-deploy workflow running on GitHub.

6.3. ticker 1.5.0

6.3.1. Changes

- uber-jar → fast-jar

- JSON logging enhancements - new configuration keys (detailed description: Quarkus based configs):
  - TICKER_JSON_MODULE_VERSION (Set in dockerfile. It is preferred for projects implementing ticker-core-quartz and building their own image to set it as well.)
  - TICKER_JSON_MODULE_ID (Set in dockerfile. It is preferred for projects implementing ticker-core-quartz and building their own image to set it as well.)
  - TICKER_JSON_K8S_NAMESPACE
  - TICKER_JSON_MDC_FIELDS
  - TICKER_JSON_EXCEPTION_OUTPUT_TYPE
  - TICKER_JSON_DATE_FORMAT
- GELF logging: moduleId and moduleVersion values set in dockerfile. (It is preferred for projects implementing ticker-core-quartz and building their own image to set it as well.)
- `quarkus.log.console.enable` configuration mapped to TICKER_LOG_CONSOLE_ENABLE environment variable (default true)
- README translated to english

6.3.2. Migration

The changes are backward compatible and do not require migration.

6.4. ticker 1.6.0

6.4.1. Changes

- Configuration prevalidation on startup (MicroprofileRestClientJob configuration only). See: Additional Information
- Coffee 2.11.0 upgrade
- AbstractQuarkusTest has been added for testing job configuration. See: Additional Information
- Parent pom upgrade: 1.4.0 → 1.5.0 so the release process is using the new `central-publishing-maven-plugin` 0.8.0.

6.4.2. Migration

The changes are backward compatible and do not require migration.

> If the `ticker.config.validation` configuration key is set to true, the job configuration will be validated at startup, and the application will fail to start in case of any errors.

6.5. ticker 1.7.0

6.5.1. Changes

- `docker-base` upgrade **1.5.0** → **1.7.0**
  - including `builder-nexus-download` and setting the `NEXUS_REPOSITORY_TYPE=central` environment variable in release Dockerfiles

6.5.2. Migration

The changes are backward compatible and do not require migration.