



Open Liberty 2020

Release Notes for Open Liberty 20.0.0.6 on Red Hat OpenShift Container Platform

Release Notes for Open Liberty 2020 on Red Hat OpenShift Container Platform

Open Liberty 2020 Release Notes for Open Liberty 20.0.0.6 on Red Hat OpenShift Container Platform

Release Notes for Open Liberty 2020 on Red Hat OpenShift Container Platform

Legal Notice

Copyright © 2020 IBM Corp

Code and build scripts are licensed under the Eclipse Public License v1 Documentation files are licensed under Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND 4.0)

Abstract

These release notes contain the latest information about new features, enhancements, fixes, and issues contained in Open Liberty 2020 on Red Hat OpenShift Container Platform release.

Table of Contents

CHAPTER 1. FEATURES	3
1.1. RUN YOUR APPS USING 20.0.0.6	3
1.2. USE GRAPHQL WITH OPEN LIBERTY	3
1.3. SUPPORT PROVISIONING FEATURES FROM MAVEN REPOSITORY	5
1.4. CONTROL APPLICATION START ORDER IN OPEN LIBERTY	5
1.5. REST VISUALIZATIONS AND HOVER-OVER DESCRIPTIONS IN OPEN LIBERTY GRAFANA DASHBOARDS	6
1.6. WEB APPLICATION STARTUP CHANGES IN OPEN LIBERTY	6
CHAPTER 2. RESOLVED ISSUES	7
CHAPTER 3. FIXED CVES	8
CHAPTER 4. KNOWN ISSUES	9

CHAPTER 1. FEATURES

With Open Liberty 20.0.0.6 you can now develop "code-first" GraphQL applications. In addition, you can now provision features from Maven repository and control the start up of applications in the server configuration.

In [Open Liberty 20.0.0.6](#):

- [Use GraphQL with Open Liberty](#)
- [Support provisioning features from Maven repository](#)
- [Control application start order in Open Liberty](#)
- [REST visualizations and hover-over descriptions in Open Liberty Grafana Dashboards](#)
- [Web application startup changes in Open Liberty](#)

View the list of fixed bugs in [20.0.0.6](#).

1.1. RUN YOUR APPS USING 20.0.0.6

If you're using [Maven](#), here are the coordinates:

```
<dependency>
  <groupId>io.openliberty</groupId>
  <artifactId>openliberty-runtime</artifactId>
  <version>20.0.0.6</version>
  <type>zip</type>
</dependency>
```

Or for [Gradle](#):

```
dependencies {
  libertyRuntime group: 'io.openliberty', name: 'openliberty-runtime', version: '[20.0.0.6,)'
}
```

Or if you're using Docker:

```
FROM open-liberty
```

1.2. USE GRAPHQL WITH OPEN LIBERTY

Open Liberty officially supports [MicroProfile GraphQL](#). This allows you to write "code-first" GraphQL applications putting clients in control of the data they receive. Use annotations like **@Query** and **@Mutation** to turn POJOs into HTTP-based GraphQL endpoints. Those query/mutation methods can then return existing entity objects and the client can specify which fields it is interested in - reducing network bandwidth and client-side processing.

Here's an example:

```
@GraphQLApi
public class MovieService {
```

```

AtomicInteger nextId = new AtomicInteger();
Map<Integer, Movie> movieDB = new HashMap<>();

@Query("movieById")
public Movie getMovieById(int id) throws UnknownMovieException {
    return Optional.ofNullable(movieDB.get(id)).orElseThrow(UnknownMovieException::new);
}

@Query("allMoviesDirectedBy")
public List<Movie> getAllMoviesWithDirector(String directorName) {
    return movieDB.values().stream()
        .filter(m -> m.getDirector().equals(directorName))
        .collect(Collectors.toList());
}

@Mutation("newMovie")
public int createNewMovie(@Name("movie") Movie movie) {
    int id = nextId.incrementAndGet();
    movie.setId(id);
    movieDB.put(id, movie);
    return id;
}
}

```

This will create a GraphQL application with two queries ("movieById" and "allMoviesDirectedBy") and a mutation ("newMovie"). This will allow a client to execute a query like:

```

query {
  allMoviesDirectedBy(directorName: "Roland Emmerich") {
    id, title, actors
  }
}

```

and see a result like:

```

{
  "data": {
    "allMoviesDirectedBy": [
      {
        "id": 1,
        "title": "Independence Day",
        "actors": [
          "Will Smith",
          "Bill Pullman",
          "Jeff Goldblum",
          ...
        ]
      },
      ...
    ]
  }
}

```

Liberty's GraphQL APIs were developed within the MicroProfile community and have broad industry support. The implementation is based on SmallRye GraphQL. Liberty's GraphQL feature goes beyond

the MicroProfile specification and adds support for metrics collection, authorization checks, and request/response logging of query and mutation methods.

Next week, we will be releasing a blog post with more details on how to use MicroProfile GraphQL in Open Liberty. Until then, check out [this sample](#) to get started - and join the growing [landscape of GraphQL adopters](#) and write your first GraphQL application today!

1.3. SUPPORT PROVISIONING FEATURES FROM MAVEN REPOSITORY

Users are now able to install desired features onto their Open Liberty runtimes from Maven Central or an on-premises Maven repository, such as one served on Artifactory or Nexus, using a convenient command line tool.

Use the **wlp/bin/featureUtility** command to find, get information about, and install assets that are in a Maven repository.

Command	Description
featureUtility help installFeature	Display help information for the installFeature action
featureUtility installFeature mpHealth-2.2 or featureUtility installFeature io.openliberty.features:mpHealth-2.2	Install the MicroProfile Health 2.2 feature from Maven Central
featureUtility installServerFeatures myserver	Install server features for the myserver server
featureUtility installFeature mpHealth-2.2 --noCache	Install the MicroProfile Health 2.2 feature without caching the feature to the local Maven repository
featureUtility installServerFeatures myserver --noCache	Install server features for the myserver server without caching the features to the local Maven repository
featureUtility installFeature adminCenter-1.0 -acceptLicense	Install the Admin Center feature from Maven Central
featureUtility installServerFeatures defaultServer --verbose	Install features for the myserver server with debug enabled
featureUtility viewSettings	View a template of your featureUtility.properties file
featureUtility find mpHealth-2.2	Search for the MicroProfile Health 2.2 feature from Maven Central and all configured Maven repositories
featureUtility find	Search for all available features from Maven Central and all configured Maven repositories

1.4. CONTROL APPLICATION START ORDER IN OPEN LIBERTY

By default, applications start in parallel and can finish starting in random order. This update provides the ability to prevent any application from starting until one or more other applications have started.

Separate applications can often have implicit dependencies on each other. For example, a single Open Liberty server might contain a front end application that provides a user interface and a back end application that accesses a database. If the front end application is available before the back end application has started, users may run into errors. This feature would allow administrators to prevent the front end application from starting until the back end is ready so that users would no longer see those errors.

Application dependencies can be defined in the configuration using the **startAfter** attribute on the **application** element. The **startAfter** attribute should contain a comma separated list of ID values for applications that should start before that application can begin starting. For example:

```
<webApplication id="frontend" location="myFrontend.war" startAfter="backend1, backend2"/>
<enterpriseApplication id="backend1" location="myBackend.ear"/>
<enterpriseApplication id="backend2" location="myUtilities.ear"/>
```

1.5. REST VISUALIZATIONS AND HOVER-OVER DESCRIPTIONS IN OPEN LIBERTY GRAFANA DASHBOARDS

The Grafana dashboard provides a wide range of time-series visualizations of MicroProfile Metrics data such as CPU, REST, Servlet, Connection Pool, and Garbage Collection metrics. It is powered by a Prometheus datasource which is configured to ingest data from one or more Open Liberty servers' **/metrics** endpoint, enabling users to view on Grafana dashboards in near real-time.

With the release of **mpMetrics-2.3** and its addition of JAX-RS metrics, we've introduced a new set of visualizations to our Open Liberty Grafana dashboards under a new tab labelled "REST". In addition, hover-over descriptions have been added to help provide a short summary about each visualization and their function. These updates apply to our Grafana dashboard for Open Liberty deployed to OKD or Red Hat OpenShift Container Platform, as well as our dashboard for standalone Open Liberty instances.

If you do not already have Grafana and Prometheus set up, there is a Kabanero guide for [Red Hat OpenShift Container Platform 4.3](#) as well as a blog post for [standalone](#) Open Liberty to help get started.

Grafana dashboards for Liberty on OKD or Red Hat OpenShift Container Platform can be found in our [open-liberty-operator repository](#). In addition, the dashboard for standalone Open Liberty can be found on the [Grafana Labs website](#).

1.6. WEB APPLICATION STARTUP CHANGES IN OPEN LIBERTY

Open Liberty has been updated to consider Web applications started after calls to the **ServletContainerInitializers** and **ServletContextListeners** have completed. This has the effect of moving more of the application initialization into the server startup route and may make applications and the server appear to take longer to start. It doesn't affect how long it takes for applications to start processing requests, it just moves it to run prior to the ports opening. In addition, you can now configure the **server.xml** so a failure in a **ServletContextListener** will cause application startup to fail. To do so, add the following:

```
<webContainer stopAppStartUponListenerException="true"/>
```

Find out more about application properties [here](#).

CHAPTER 2. RESOLVED ISSUES

See the [Open Liberty 20.0.0.6 issues that were resolved for this release](#) .

CHAPTER 3. FIXED CVES

For a list of CVEs that were fixed in Open Liberty 20.0.0.6, see [security vulnerabilities](#).

CHAPTER 4. KNOWN ISSUES

See the [list of issues that were found but not fixed during the development of 20.0.0.6](#) .