



Open Liberty 2020

Release Notes for Open Liberty 20.0.0.1 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 20.0.0.1 on Red Hat OpenShift Container Platform

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

Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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 20.0.0.1	3
1.2. USING LIBERTY SOCIAL LOGIN WITH RED HAT OPENSIFT	3
1.3. MONITOR THE PROCESS CPU TIME (MICROPROFILE METRICS 2.0)	5
1.4. FASTER APPLICATION STARTUPS WITH LIBERTY ANNOTATION CACHING	6
1.5. BUG FIXES IN JAVASERVER FACES 2.3	7
CHAPTER 2. RESOLVED ISSUES	8
CHAPTER 3. FIXED CVES	9
CHAPTER 4. KNOWN ISSUES	10

CHAPTER 1. FEATURES

In Open Liberty 20.0.0.1, you can configure the Social Login feature to use Red Hat OpenShift's OAuth server for authentication. In addition, there is a new MicroProfile Metric to measure CPU time, memory heap and response time.

In [Open Liberty 20.0.0.1](#):

- [Support OpenShift OAuth server for authentication and authorization](#)
- [Monitor the process CPU time \(MicroProfile Metrics 2.0\)](#)
- [Faster application startups with Liberty annotation caching](#)
- [Updated JavaServer Faces](#)

View the list of fixed bugs in [20.0.0.1](#).

1.1. RUN YOUR APPS USING 20.0.0.1

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

```
<dependency>
  <groupId>io.openliberty</groupId>
  <artifactId>openliberty-runtime</artifactId>
  <version>20.0.0.1</version>
  <type>zip</type>
</dependency>
```

Or for [Gradle](#):

```
dependencies {
  libertyRuntime group: 'io.openliberty', name: 'openliberty-runtime', version: '[20.0.0.1,)'
}
```

Or if you're using Docker:

```
docker pull open-liberty
```

1.2. USING LIBERTY SOCIAL LOGIN WITH RED HAT OPENSIFT

The Social Login feature **socialLogin-1.0** can now be configured to use OpenShift's built-in OAuth server and OAuth Proxy sidecar as authentication providers. The Social Login feature has several pre-configured providers (e.g. Google, GitHub, Facebook) but you can also configure additional providers (e.g. Instagram). OpenShift's OAuth server and OAuth Proxy sidecar can now be configured as additional providers too. The first is a standard OAuth Authorization Code flow, where a web browser accessing an app running in Liberty is redirected to the OpenShift OAuth server to authenticate. The second is accepting an inbound token from the OpenShift OAuth Proxy sidecar or obtained from an OpenShift API call. This approach requires less cluster-specific configuration.

Most users of this will run Liberty in a pod; however, in the Authorization Code flow, Liberty can run outside the OpenShift cluster. In either mode, an optional JWT can be created for propagation to downstream services.

Using OpenShift as a provider differs slightly from other OAuth providers, it requires a service account token to obtain information about the OAuth tokens. Once the: client ID, secret, and token have been obtained from OpenShift, Liberty can be configured as shown below.

To enable the feature add it to the **server.xml**

Server configuration to use OpenShift OAuth server:

```
<server description="social">

  <!-- Enable features -->
  <featureManager>
    <feature>appSecurity-3.0</feature>
    <feature>socialLogin-1.0</feature>
  </featureManager>

  <logging traceSpecification="com.ibm.ws.security.*=all=enabled" maxFiles="8" maxFileSize="200"/>

  <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="8941" httpsPort="8946" > <tcpOptions
soReuseAddr="true" /> </httpEndpoint>

  <!-- specify your clientId, clientSecret and userApiToken as liberty variables or environment
variables -->
  <oauth2Login id="openshiftLogin"
    scope="user:full"
    clientId="{myclientId}"
    clientSecret="{myclientSecret}"
    authorizationEndpoint="https://oauth-openshift.apps.papains.os.fyre.ibm.com/oauth/authorize"
    tokenEndpoint="https://oauth-openshift.apps.papains.os.fyre.ibm.com/oauth/token"
    userNameAttribute="username"
    groupNameAttribute="groups"
    userApiToken="{serviceAccountToken}"
    userApiType="kube"
    userApi="https://api.papains.os.fyre.ibm.com:6443/apis/authentication.k8s.io/v1/tokenreviews">
  </oauth2Login>

  <keyStore id="defaultKeyStore" password="keyspass" />

  <!-- more application config would go here -->

</server>
```

In the sidecar scenario, the configuration changes to accept an inbound token from the sidecar. Server configuration to use OAuth proxy sidecar:

```
<!-- specify your userApiToken as a liberty variable or environment variable -->
<!-- note that no clientId or clientSecret are needed -->
<oauth2Login id="openshiftLogin"
  scope="user:full"
  userNameAttribute="username"
  groupNameAttribute="groups"
  userApiToken="{serviceAccountToken}"
  userApiType="kube"
  accessTokenHeaderName="X-Forwarded-Access-Token"
```



```

    accessTokenRequired="true"
    userApi="https://kubernetes.default.svc/apis/authentication.k8s.io/v1/tokenreviews">
</oauth2Login>

```

Use of HTTPS communication requires that the server either have a key signed by a well-known certificate authority, which Liberty can trust automatically or that the server's public key be added to the Liberty trust store. OpenShift does not come with CA-signed keys by default, so the Red Hat OpenShift OAuth server's public key will need to be added. The most convenient way to do this is to specify an environment variable in **server.env**. That identifies the file containing the public key in PEM format. Liberty will read the file and add the key to its trust store.

```

# server.env

# OAuth sidecar scenario: causes the Kubernetes default certificate that is pre-installed in pods to be
added to Liberty trust store.
cert_defaultKeyStore=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt

# OAuth server scenario: causes the public keys from /tmp/trustedcert.pem (obtained seperetly) to be
added to Liberty trust store.
cert_defaultKeyStore=/tmp/trustedcert.pem

```

1.3. MONITOR THE PROCESS CPU TIME (MICROPROFILE METRICS 2.0)

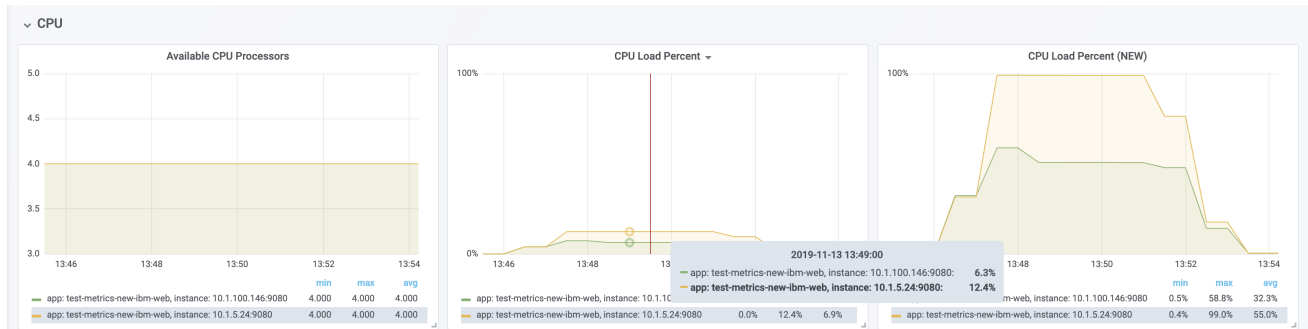
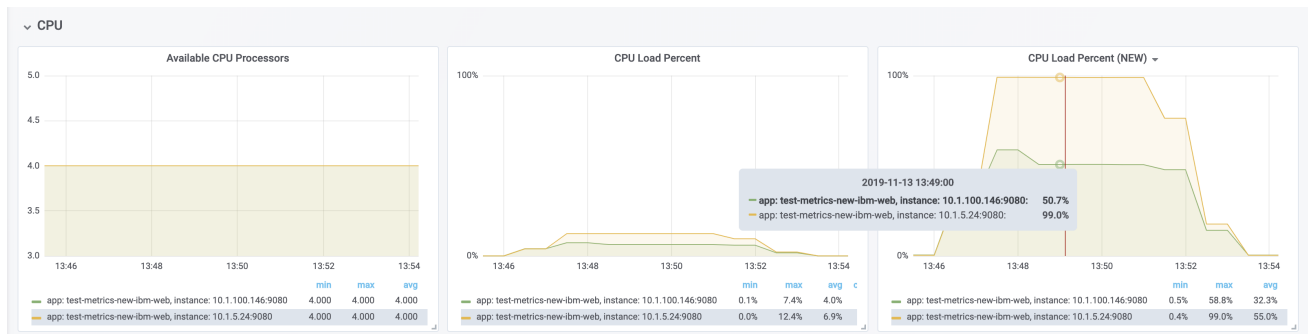
A new metric, **processCpuTime**, which returns the CPU time used by the process on which the JVM is running. The MicroProfile Metrics feature provides information monitoring an application, such as CPU time used, memory heap, response time of servlets.

The new **processCpuTime** metric provides a more accurate CPU load percentage on cloud platforms via Grafana. Previously, the CPU load percentage was shown with the metric **processCpuLoad**. However, the load percentage was calculated using the total number of cores allocated to the deployment. If the deployment has a restricted number of cores, the **processCpuLoad** ends up showing a plateau on Grafana when the maximum number of cores is reached. For example, on a deployment with 32 cores allocated but restricted to four cores, the **processCpuLoad** graph shows a plateau at 12.5% when all four cores are used.

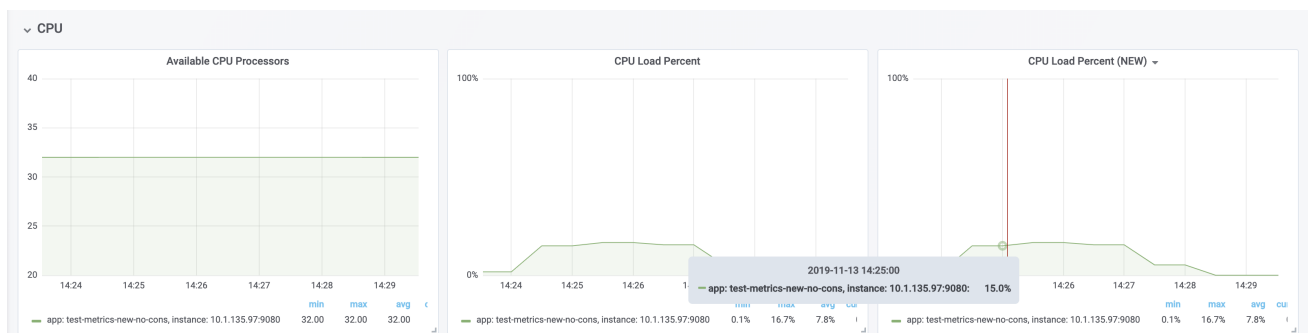
The new metric, **processCpuTime**, can be manipulated on Grafana to create a more accurate representation of the CPU being used. **rate(processCpuTime)[1m]** shows the average rate of increase in CPU time over one minute. Dividing this by the total number of CPU cores, we can see a more accurate percentage of the CPU used, taking into account the constraints.

The new **processCpuTime** metric is displayed on the **/metrics** endpoint with the MicroProfile Metrics 2.0 and 2.2 features. On the dashboard, a new panel can be created with the following PromQL query: **(rate(base:cpu_process_cpu_time[2m])/1e9) / base:cpu_available_processors{app=~}**. [View full dashboard](#).

The following images show that the old metric, **processCpuLoad**, plateaus at 12.5% (4/23), while the new metric, **processCpuTime**, more accurately represents the percentage of CPU used.



With all the machines cores being used and there are no constraints on the processors (32 processors) - The old version and new version display the same data.

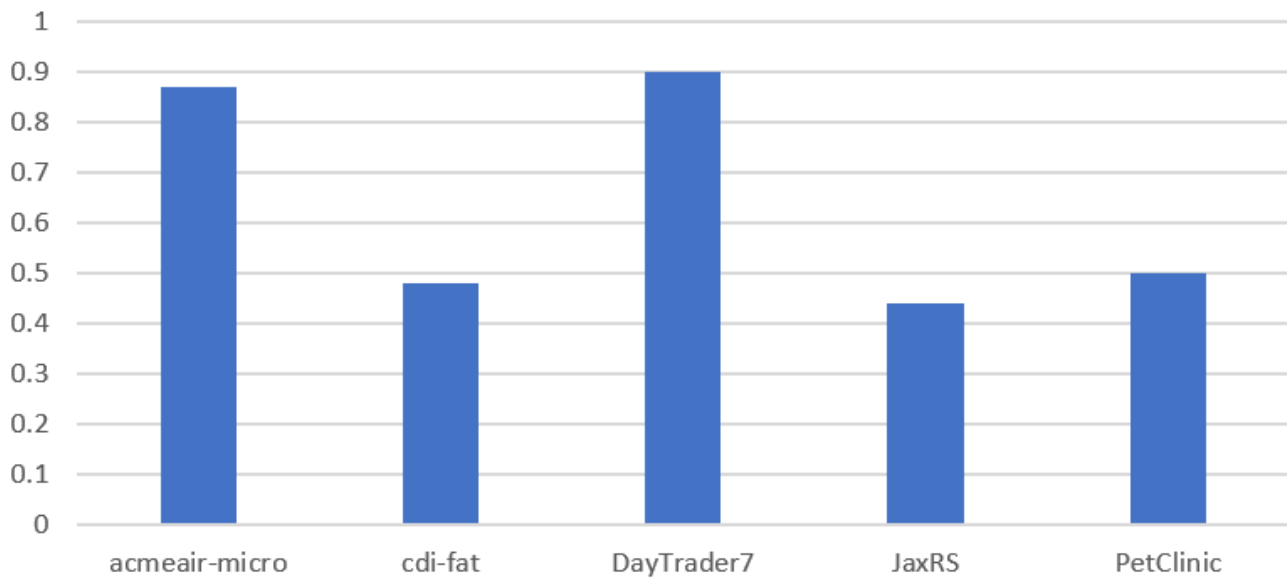


1.4. FASTER APPLICATION STARTUPS WITH LIBERTY ANNOTATION CACHING

Application startup times have been made faster by adding cache capabilities **annotation caching** to core class and annotation scanning function. Depending on application characteristics, startup times are reduced by 10% to more than 50%. Applications with many jar files, or which use CDI or JAX-RS function, see the best improvements:

Open Liberty - v19.0.0.11

Relative Startup Time with Annotation Caching



Good news! Annotation caching is enabled by default.

Annotation cache data is stored in the server workarea. Cache of application class data is cleared when performing a clean server start (starting the server with the **--clean** option). In normal operations, the clearing of cache data is not necessary, since the cache automatically regenerates cache data for changed application classes.

In container environments, for annotation caching to be effective, the server image must be "warmed" when the container image is created. Warming the server can be done by starting and stopping the server during the container build. Warming the image moves the annotation scan into the container build meaning you get optimal startup on the container deployment. Using the `configure.sh` file in the base open-liberty docker images causes the server to be started and stopped during the container build.

1.5. BUG FIXES IN JAVASERVER FACES 2.3

JavaServer Faces 2.3 contains a new feature to get bug fixes that are in Apache MyFaces 2.3.6. The jsf-2.3 feature pulls in the Apache MyFaces implementation and integrates it into the Liberty runtime.

The Apache MyFaces 2.3.6 release contains bug fixes. View [the release notes for more information](#).

To use the JSF 2.3, enable the **jsf-2.3** feature to leverage the latest Apache MyFaces 2.3. release For more information about the JavaServer Feature, view the [Apache website](#).

CHAPTER 2. RESOLVED ISSUES

See the [Open Liberty 20.0.0.1 issues that were resolved for this release](#) .

CHAPTER 3. FIXED CVES

For a list of CVEs that were fixed in Open Liberty 20.0.0.1, see [security vulnerabilities](#).

CHAPTER 4. KNOWN ISSUES

See the [list of issues that were found but not fixed during the development of 20.0.0.1](#) .