

## Lab 4: Using Open Liberty Tools with VS Code



---

# Contents

## Contents

- LAB 4 USING OPEN LIBERTY TOOLS (DEV MODE) WITH VS CODE..... 1**
  - 1.1 OBJECTIVES ..... 1
  - 1.2 LAB REQUIREMENTS..... 2
  - 1.4 INTRODUCTION – OPEN LIBERTY TOOLS EXTENSION FOR VS CODE ..... 3
    - 1.4.1 LIBERTY MAVEN PLUGIN ..... 4**
    - 1.4.2 INTERACTING WITH DEV MODE ..... 4**
    - 1.4.3 LIBERTY DEV MODE COMMANDS ..... 5**
  - 1.5 THE LAB ENVIRONMENT ..... 6
    - 1.5.1 LOGIN TO THE "LIBERTY VPOT ... DESKTOP" VM AND GET STARTED..... 7**
  - 1.6 GETTING STARTED WITH OPEN LIBERTY TOOLS IN VS CODE ..... 10
    - 1.6.1 REVIEW THE VS CODE EXTENSIONS AND PROJECTS POM.XML FILE USED FOR THIS PROJECT..... 10**
  - 1.7 USING OPEN LIBERTY TOOLS IN VS CODE ..... 15
    - 1.7.1 DEVELOPER EXPERIENCE USING OPEN LIBERTY TOOLS IN VS CODE..... 17**
    - 1.7.2 RUNNING TESTS USING THE OPEN LIBERTY TOOLS IN VS CODE ..... 27**
  - 1.8 INTEGRATED DEBUGGING USING THE OPEN LIBERTY TOOLS IN VS CODE..... 33

---

## Lab 4 Using Open Liberty Tools (dev mode) with VS Code

### 1.1 Objectives

In this exercise, you will learn how developers can use Liberty in “dev” mode with VS Code Integrated Development Environment for achieving efficient iterative develop, test, debug cycle when developing Java based applications and microservices.

At the end of this lab you should be able to:

- Experience using the Open Liberty Tools extension available in VS Code to efficiently develop, test, and debug Java cloud native applications.
- Experience hot reloading of application code and configuration changes using dev mode

You will need an estimated **60 to 90 minutes** to complete this lab. Keep this in mind when requesting your Skytap cloud environment as documented in the **Prerequisites** stage of this lab.

**Reserve only for the time you need.** This is a finite resource and limiting your reservation time ensures more IBMers can leverage the resources for their client engagements.

## 1.2 Lab requirements

1. Use the lab environment that we prepared for this lab. It already has the prerequisite software installed and configured.

## 1.3 Prerequisites

### IMPORTANT!

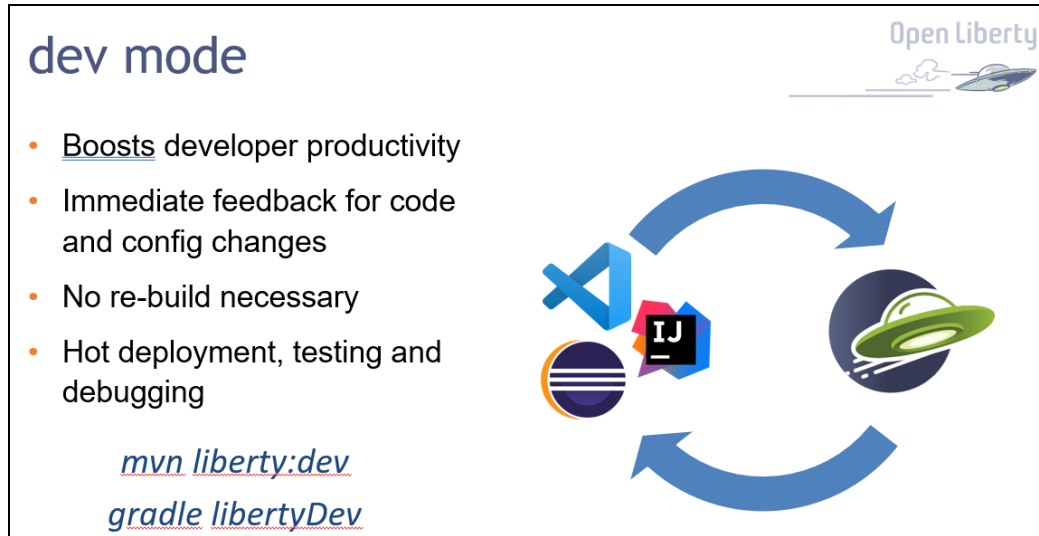


In **self-paced mode**, you are required to provision a lab environment that we made available for this lab.

Otherwise, in an **instructor led** lab, the lab instructor will provide access to pre-provisioned lab environment specific for completing this lab.

## 1.4 Introduction – Open Liberty Tools extension for VS Code

In a separate lab, you learned how Open Liberty dev mode can be run from a command line while allowing you to edit your code with any text editor or IDE.



In this lab, you will use the “**Open Liberty Tools**” **VS Code extension** to start Open Liberty in development mode, make changes to your application while the server is up, run tests and view results, and even debug the application without leaving the editor.

Your code is automatically compiled and deployed to your running server, making it easy to iterate on your changes.

The Open Liberty Tools for VS Code contains the following key Features

- View **liberty-maven-plugin projects** in the workspace (version 3.1 or higher)
- View **liberty-gradle-plugin projects** in the workspace (version 3.0 or higher)
- Start/Stop Open Liberty Server in dev mode
- Start Open Liberty Server dev mode with custom parameters
- Run Unit and Integration tests
- View unit and integration test reports

The Open Liberty Tools for VS Code has a dependency on the **Tools for MicroProfile** VS Code extension to support the development of MicroProfile based microservices.

The **Tools for MicroProfile** VS Code extension has dependencies on the following:

- Java JDK (or JRE) 11 or more recent
- Language Support for Java by Red Hat VS Code extension.

### 1.4.1 Liberty Maven Plugin

The **Liberty Maven Plugin** provides several goals for managing a Liberty server and applications.

Maven 3.5.0 or later is recommended to use the Liberty Maven Plugin.

Enabling the Liberty Maven Plugin in your project, simply add the following XML Stanza to your **pom.xml** file.

```
<project>
  ...
  <build>
    <plugins>
      <!-- Enable liberty-maven-plugin -->
      <plugin>
        <groupId>io.openliberty.tools</groupId>
        <artifactId>liberty-maven-plugin</artifactId>
        <version>[3.3.4,)</version>
        <!-- Specify configuration, executions for liberty-maven-plugin -->
        ...
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

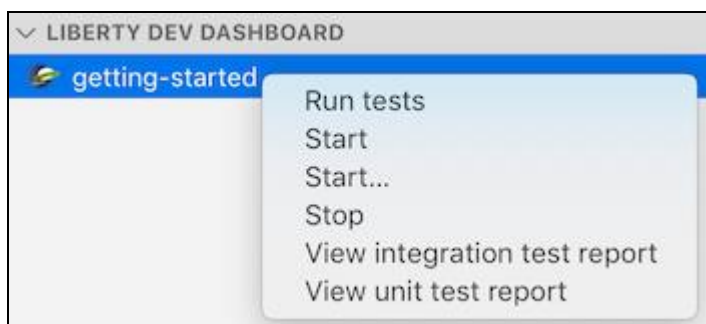
For detailed information about the Maven goals supported by the Liberty Maven Plugin, visit:

<https://github.com/OpenLiberty/ci.maven>

### 1.4.2 Interacting with dev mode

Once the **Liberty Maven Plugin** is specified in your **pom.xml** file, your project name is then listed under the **Liberty Dev Dashboard** in the side panel in VS Code, as illustrated below.

You can interact with dev mode by right-clicking on your project name and selecting one of the commands supported by the Open Liberty Tools extension.



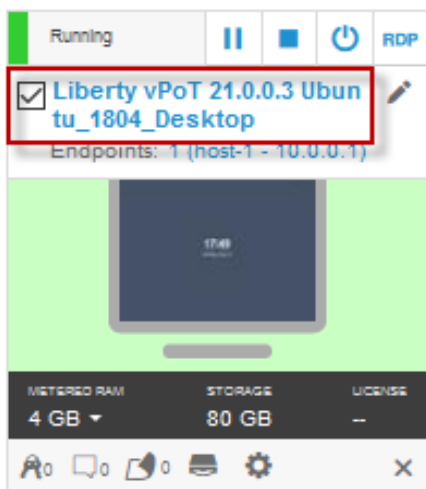
### 1.4.3 Liberty dev mode Commands

The following commands can be selected from the drop-down menu after right-clicking on your project name in the Liberty Dev Dashboard.

COMMAND	DESCRIPTION
Run tests	Runs the unit tests and integration tests that are configured for your project. This command requires dev mode to be already started.
Start	Starts dev mode.
Start...	Starts dev mode with custom parameters. For example, to run tests automatically after every change, include the <code>-DhotTests</code> parameter. Additional supported parameters can be found in the documentation for the <a href="#">dev goal of the Liberty Maven Plugin</a> .
Stop	Stops dev mode.
View integration test report	Views the integration test report file.
View unit test report	Views the unit test report file.

## 1.5 The lab environment

One (1) Linux VM has been provided for this lab.



The “**Liberty vPOT ... Desktop**” VM has the following software available:

- Application Project with Liberty
- Maven 3.6.0
- The login credentials for the **Liberty vPOT ... Desktop** VM are:

User ID: **ibmdemo**

Password: **passw0rd** (That is a numeric zero in passw0rd)

### IMPORTANT:



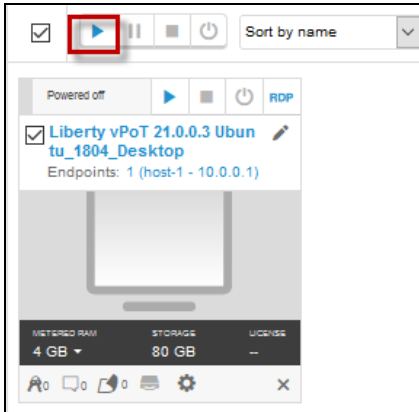
Using the lab environment provided, all the required VS code extensions and dependencies have been installed for you.

This allows you to focus on the value of using the capabilities of the tools for fast, efficient inner-loop development, test, debug, of Java based applications and Microservices using Open Liberty in dev mode.

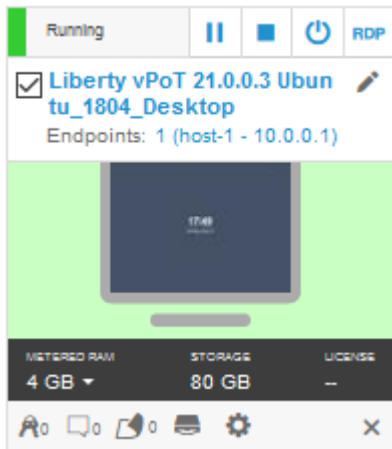


## 1.5.1 Login to the "Liberty vPOT ... Desktop" VM and Get Started

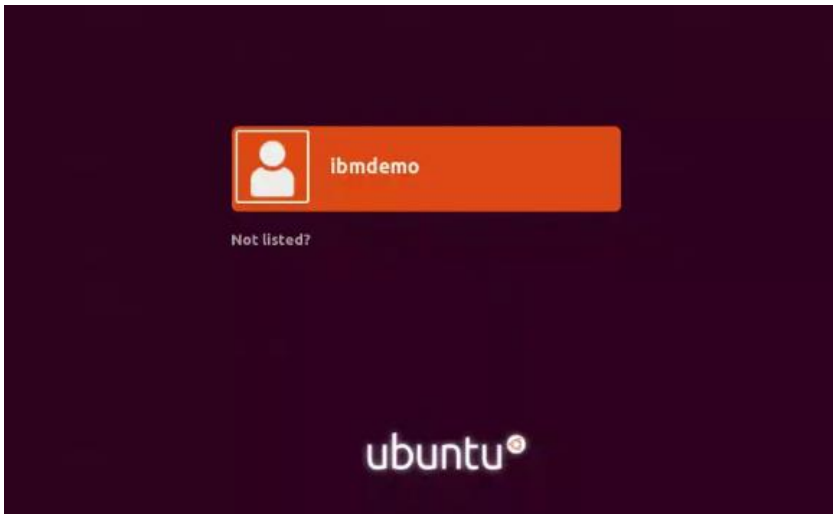
- \_\_1. If the VM is **not** already started, start it by clicking the **Play** button.



- \_\_2. After the VM is started, click the “**Liberty vPOT ... Desktop**” VM icon to access it.

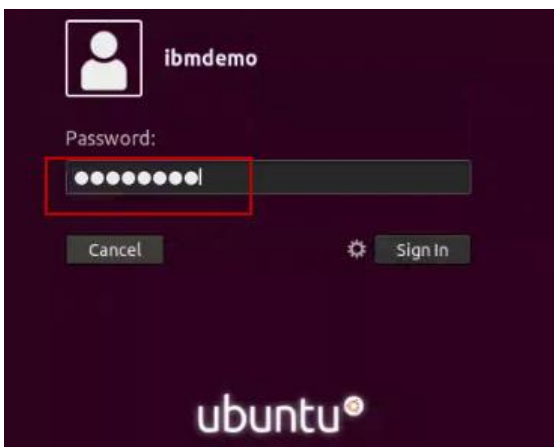


- \_\_3. Login with **ibmdemo** ID.
- \_\_a. Click on the “**ibmdemo**” icon on the Ubuntu screen.



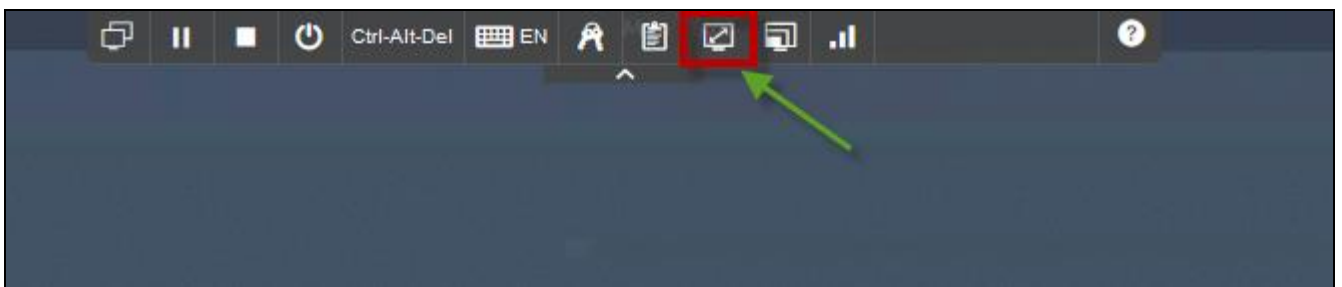
\_\_b. When prompted for the password for “ibmdemo” user, enter “**passw0rd**” as the password:

Password: **passw0rd** (lowercase with a zero instead of the o)



\_\_4. Resize the Skytap environment window for a larger viewing area while doing the lab.

From the Skytap menu bar, click on the “**Fit to Size**”  icon. This will enlarge the viewing area to fit the size of your browser window.

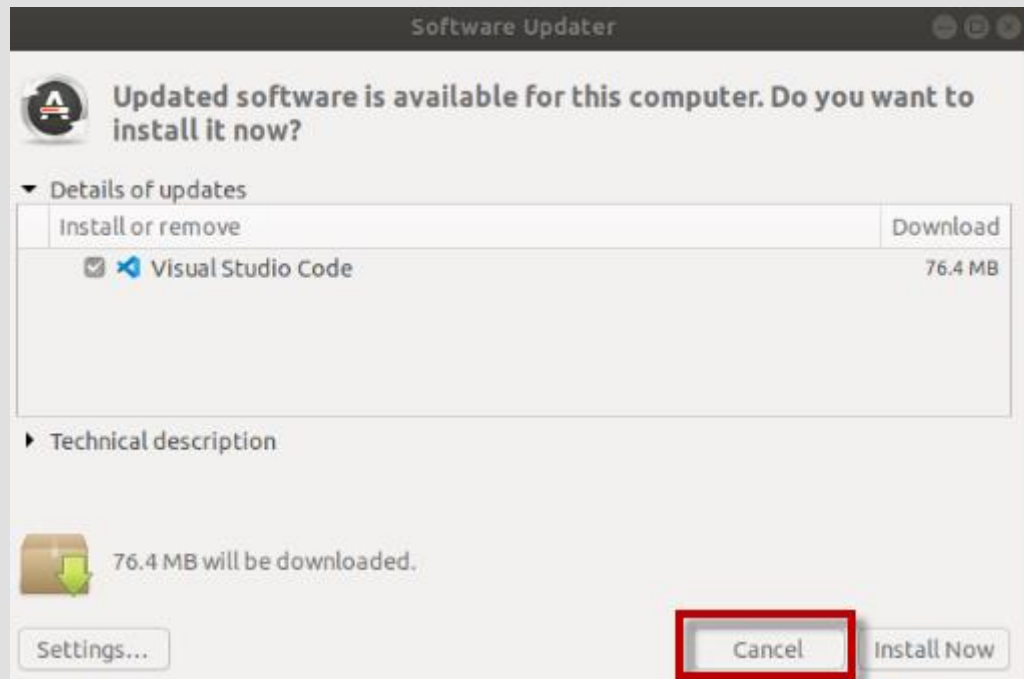


**Important:**

**Click CANCEL....** If, at any time during the lab, you get a pop-up asking to install updated software onto the Ubuntu VM.

The one we experience is an update available for VS Code.

**CLICK CANCEL!**



## 1.6 Getting Started with Open Liberty Tools in VS Code

**Liberty Dev mode** allows you, as the developer, to focus on your code. When Open Liberty is running in dev mode, your code is automatically compiled and deployed to the running server, making it easy to iterate on your changes.

In this lab, as a developer, you will experience using the **Open Liberty Tools** extension in **VS Code** to work with your code, run tests on demand, so that you can get immediate feedback on your changes.

You will also work with integrated debugging tools and attach a Java debugger to debug your running application.

From a developer perspective, this is a huge gain in efficiency, as all these iterative inner-loop development activities occur without ever leaving the integrated development environment (IDE).

### 1.6.1 Review the VS Code extensions and projects pom.xml file used for this project

The sample application used in this lab is configured to be built with Maven. Every Maven-configured project contains a pom.xml file, which defines the project configuration, dependencies, plug-ins, and so on.

Your pom.xml file is in the root directory of the project and is configured to include the **liberty-maven-plugin**, which allows you to install applications into Open Liberty and manage the server instances.

To begin, navigate to the project directory and review the IDE extensions and pom.xml file that is used for the “**system**” microservice that is provided in the lab.

First, add the project folder to a VS Code Workspace

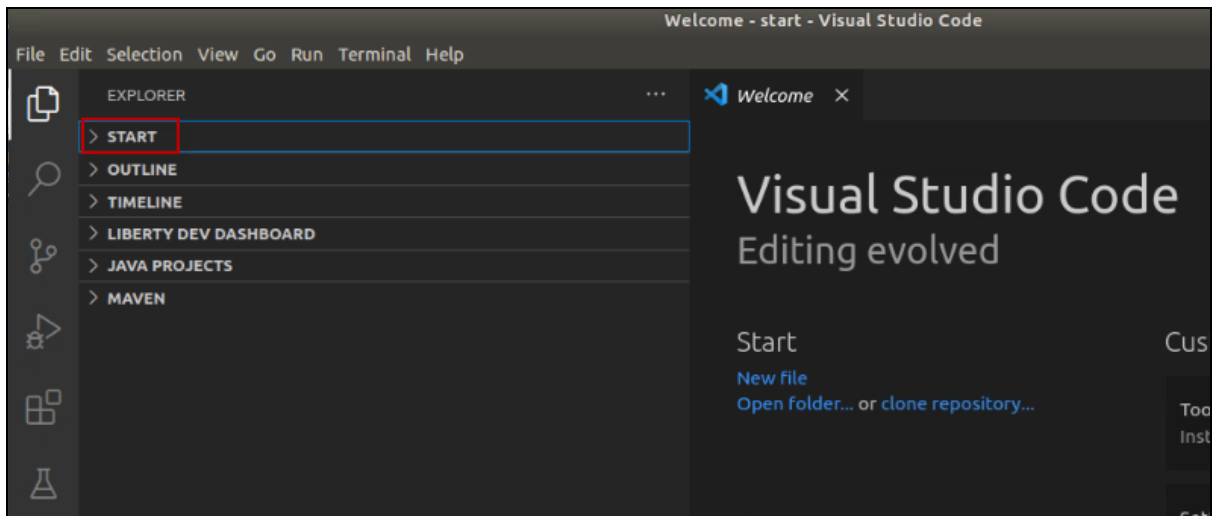
- \_\_1. **Close** all **Terminal** windows and **Brower** Tabs used in any previous lab.
- \_\_2. Navigate to the project directory and launch VS Code from the “**start**” folder of the project.
  - a. Open a terminal window and change to the following directory:

```
cd /home/ibmdemo/Student/labs/devmode/guide-getting-started/start
```

- b. Launch VS Code using the current directory as the root folder for the workspace

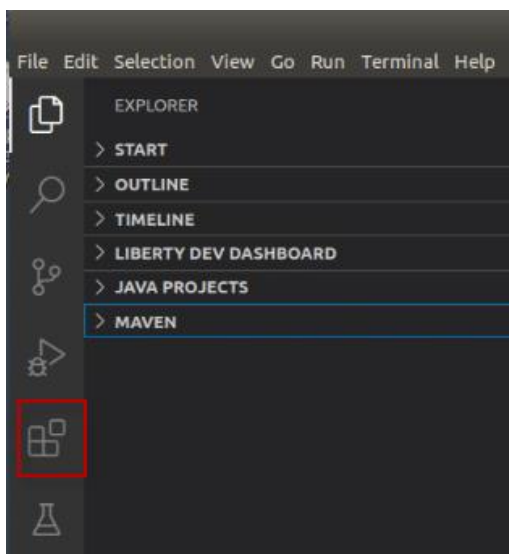
```
code .
```

When the VS Code UI launches, the Explorer view is shown. The “START” folder contains the source code for the project.



\_\_3. Review the installed extensions in VS Code that are used for this lab.

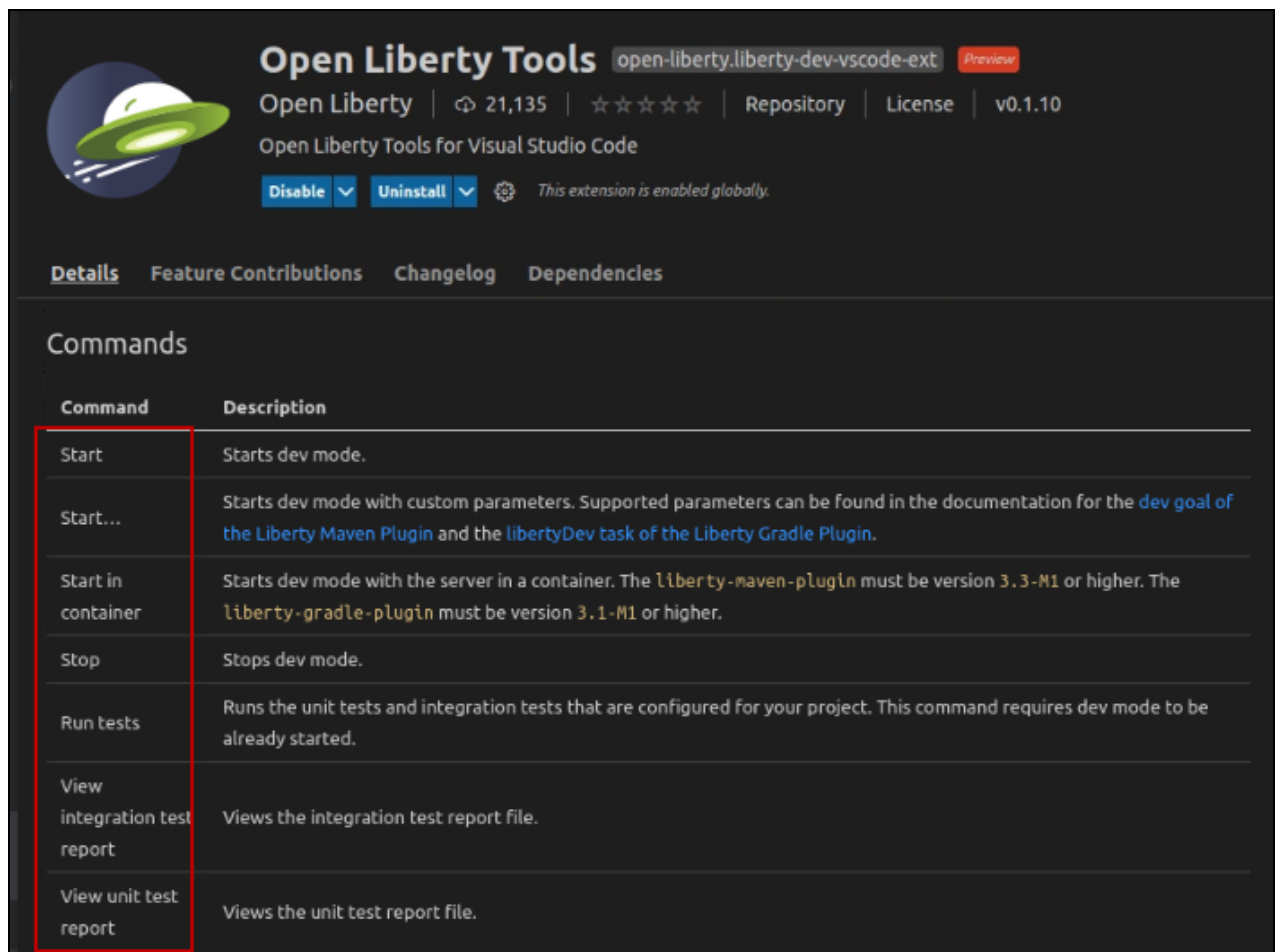
a. Click on the **Extensions** icon in the left navigation bar in VS Code.



b. Expand the “INSTALLED” extensions section to list the extensions that are currently installed in this environment. The notable extensions used in this lab are:

- i. Open Liberty Tools
- ii. Tools for MicroProfile
- iii. Language Support for Java
- iv. Debugger for Java

- c. Click on the “**open Liberty Tools**” extension to view its details.
- d. Notice the list of commands that are supported by the Open Liberty Tools extension.



**Open Liberty Tools** `open-liberty.liberty-dev-vscode-ext` Preview

Open Liberty | 21,135 | ★★★★★ | Repository | License | v0.1.10

Open Liberty Tools for Visual Studio Code

[Disable](#) [Uninstall](#) [Settings](#) This extension is enabled globally.

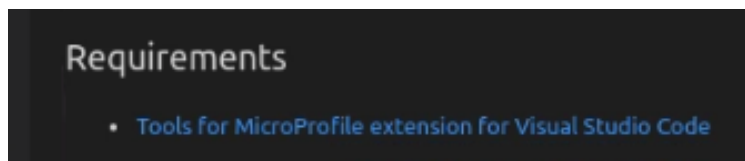
[Details](#) [Feature Contributions](#) [Changelog](#) [Dependencies](#)

### Commands

Command	Description
Start	Starts dev mode.
Start...	Starts dev mode with custom parameters. Supported parameters can be found in the documentation for the <a href="#">dev goal of the Liberty Maven Plugin</a> and the <a href="#">libertyDev task of the Liberty Gradle Plugin</a> .
Start in container	Starts dev mode with the server in a container. The <code>liberty-maven-plugin</code> must be version 3.3-M1 or higher. The <code>liberty-gradle-plugin</code> must be version 3.1-M1 or higher.
Stop	Stops dev mode.
Run tests	Runs the unit tests and integration tests that are configured for your project. This command requires dev mode to be already started.
View integration test report	Views the integration test report file.
View unit test report	Views the unit test report file.

- e. Scroll down to the “**Requirements**” section of the Open Liberty Tools details page.

Notice the requirement for “Tools for MicroProfile” to support development of Microservices that use MicroProfile APIs with Open Liberty.



### Requirements

- [Tools for MicroProfile extension for Visual Studio Code](#)

### Information:

The **Tools for MicroProfile** extension requires the components to be installed in the environment:




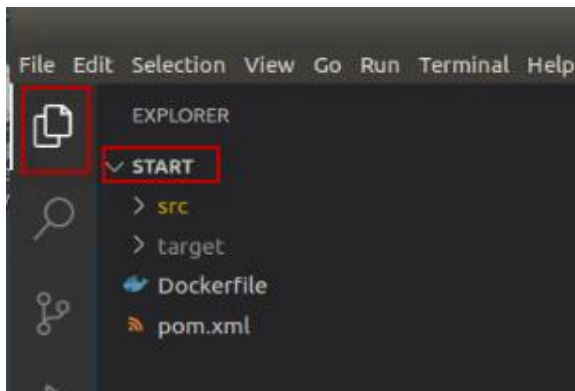
#### Requirements

- Java JDK (or JRE) 11 or more recent
- [Language Support for Java\(TM\) by Red Hat](#)

- f. **Close** the Open Liberty Tools Extension details page.

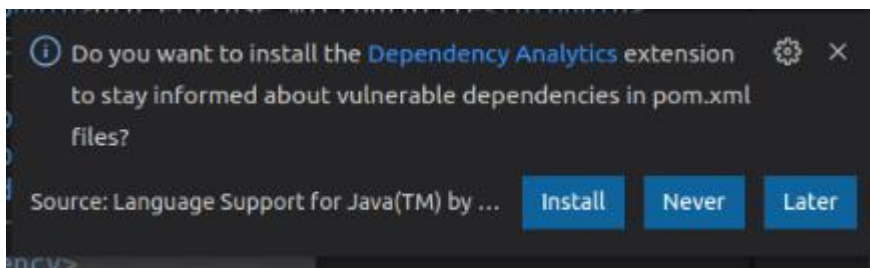
- \_\_4. Review the **pom.xml** file used to configure and build the “system” microservice.

- a. Click on the **Explorer** icon  located on the left navigation bar in VS Code.
- b. Expand the **START** folder if it is not already expanded



- c. Click on the **pom.xml** file to open it in the editor pane
- d. Close any Pop-up boxes asking if you want to install extensions or switch views.

**Note:** You may see additional pop-ups, just close them, or ignore them.



- e. Note the binary packaging of the Java application war file that is produced from the Maven Build. The WAR file produced will be named **guide-getting-started** version 1.0-SNAPSHOT.

```
7    <groupId>io.openliberty.guides</groupId>
8    <artifactId>guide-getting-started</artifactId>
9    <version>1.0-SNAPSHOT</version>
10   <packaging>war</packaging>
11
```

- f. Default HTTP and HTTPS Ports are defined, and substituted into the server.xml file

```
17    <!-- Liberty configuration -->
18    <liberty.var.default.http.port>9080</liberty.var.default.http.port>
19    <liberty.var.default.https.port>9443</liberty.var.default.https.port>
20    </properties>
21
```

- g. The Open Liberty Tools Plugin is enabled, with a supported version of 3.3.4

```
67    <!-- Enable liberty-maven plugin -->
68    <plugin>
69        <groupId>io.openliberty.tools</groupId>
70        <artifactId>liberty-maven-plugin</artifactId>
71        <version>3.3.4</version>
72    </plugin>
```

- h. Plugin for running Tests is also added to the Maven configuration, that leverage the testing dependencies also defined in the pom.xml file.

```
83    <!-- Plugin to run functional tests -->
84    <plugin>
85        <groupId>org.apache.maven.plugins</groupId>
86        <artifactId>maven-failsafe-plugin</artifactId>
87        <version>2.22.2</version>
88        <configuration>
89            <systemPropertyVariables>
90                <http.port>${liberty.var.default.http.port}</http.port>
91            </systemPropertyVariables>
92        </configuration>
93    </plugin>
```

- i. **Close** the pom.xml file



**Information:**

**Tip:** Additional information on the liberty-maven-plugin can be found here:

<https://github.com/OpenLiberty/ci.maven>



## 1.7 Using Open Liberty Tools in VS Code

In this section of the lab, you will use the **Open Liberty Tools in VS Code** to work with your code and run tests on demand, so that you can get immediate feedback on your changes.

### Important:

#### For Open Liberty Tools (LIBERTY DEV DASHBOARD)

VS Code provides extensions for Java to support the Java language features.

VS Code for Java supports two modes.

- Lightweight mode
- Standard mode



VS Code has a default configuration called “hybrid mode” where a workspace is opened in Lightweight mode, but as needed, you are prompted to switch to Standard mode.

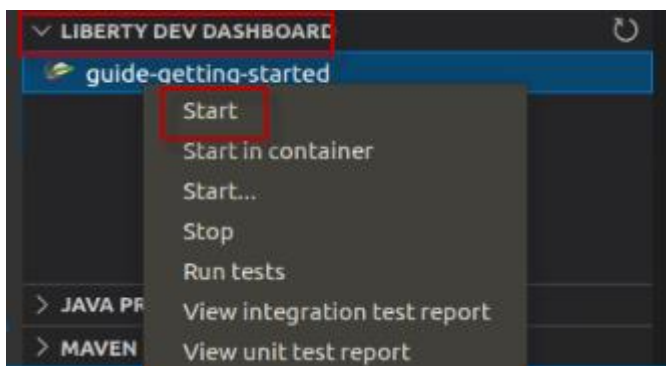
The **Tools for MicroProfile** Extension, which is required for the **Open Liberty Tools** extension, requires the Java workspace to be opened in “**STANDARD**” mode. Otherwise the LIBERTY DEV DASHBOARD will not function properly.

**Tip:** In this lab environment, the workspace is already configured to use Standard mode.

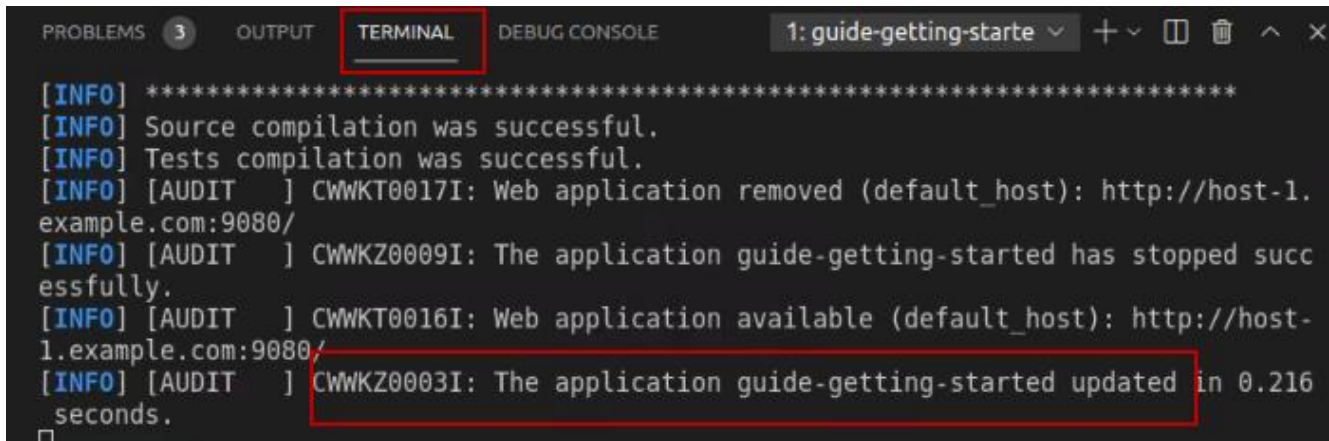
For more details on VS Code for Java is available here:

<https://code.visualstudio.com/docs/java/java-project>

- \_\_\_1. Use the Liberty Dev Dashboard to **start** the Liberty Server in dev mode
  - a. In VS Code, expand the LIBERTY DEV DASHBOARD section
  - b. Right-mouse click on the **guide-getting-started** Liberty Server
  - c. Select **Start** from the menu to start the server

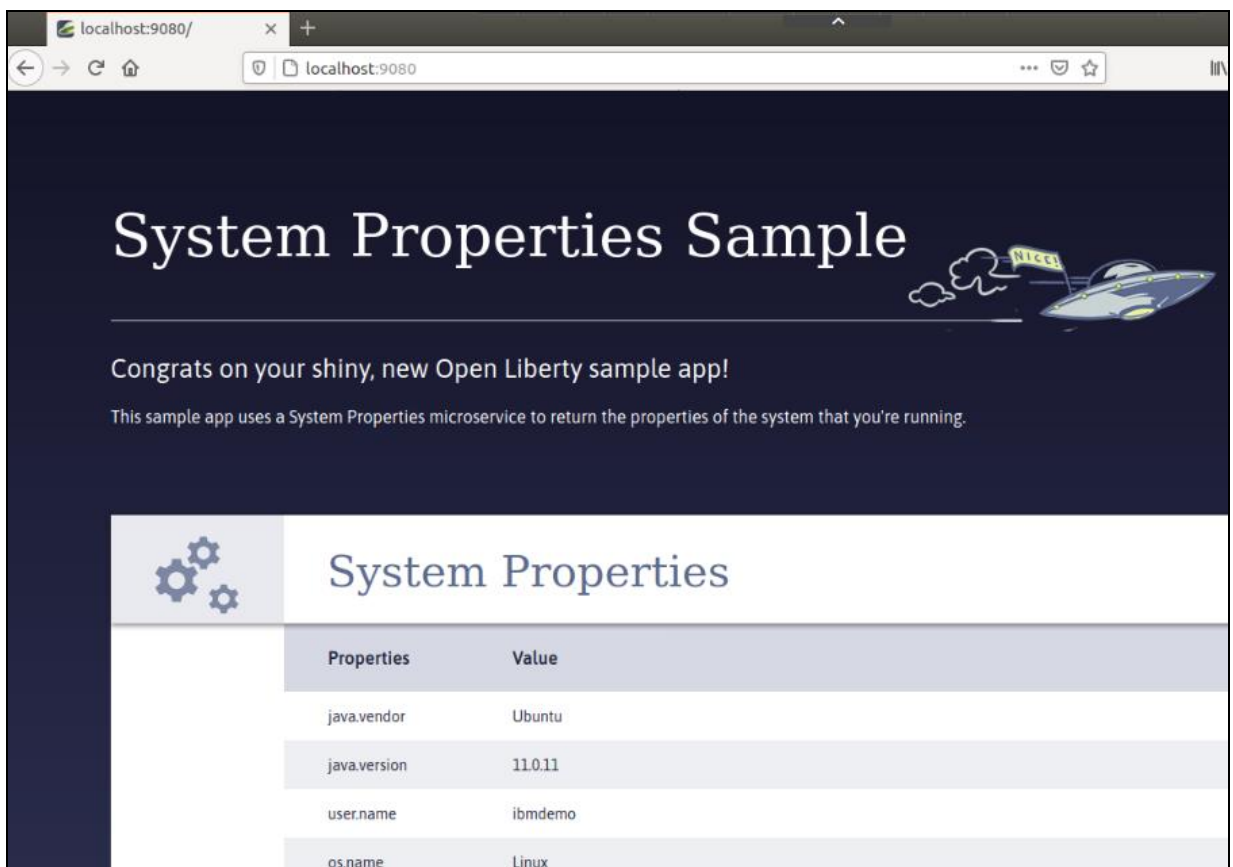


- d. The Terminal view opens, and you see the server log messages as the server starts. When the following message appears in the Terminal, the Liberty server is started.



```
[INFO] ****
[INFO] Source compilation was successful.
[INFO] Tests compilation was successful.
[INFO] [AUDIT ] CWWKT0017I: Web application removed (default_host): http://host-1.
example.com:9080/
[INFO] [AUDIT ] CWWKZ0009I: The application guide-getting-started has stopped succ
essfully.
[INFO] [AUDIT ] CWWKT0016I: Web application available (default_host): http://host-
1.example.com:9080/
[INFO] [AUDIT ] CWWKZ0003I: The application guide-getting-started updated in 0.216
seconds.
```

- \_\_2. Run the system Properties sample application from a web browser
- Open the Firefox Browser from inside of the VM
  - Go to <http://localhost:9080> to verify the application is running.



### 1.7.1 Developer experience Using Open Liberty Tools in VS Code

The System Properties Sample application is up and running in the Liberty server.

Next, as a developer, you want to implement a health check for the application.

The developer experience is frictionless, as all code and configuration change the developer introduces, are automatically detected and the server and application are dynamically updated in the running server to reflect the updated code and configuration.

Let's explore a couple of examples of the very efficient developer experience by implementing some new capability into our service.

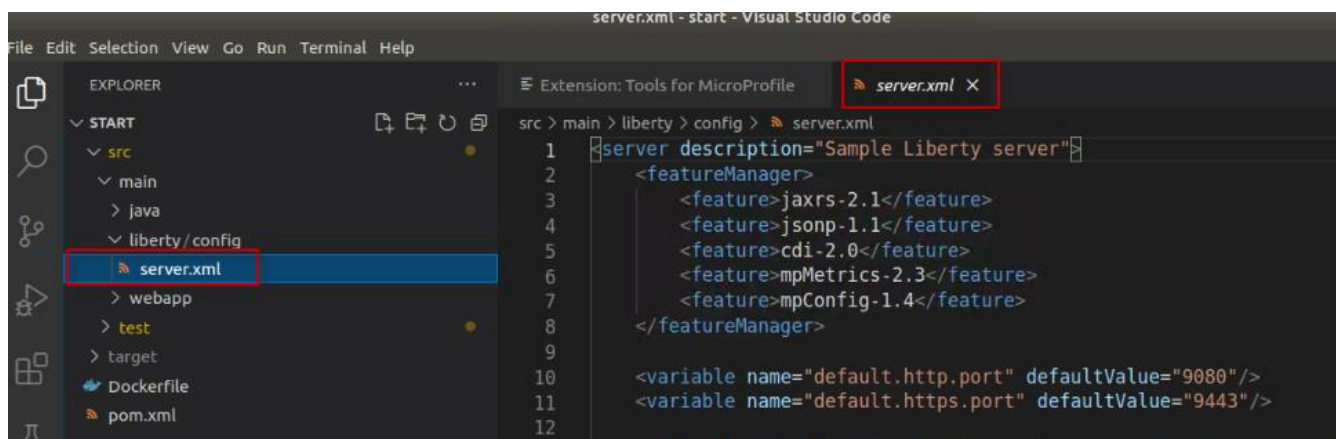
In this example, you will leverage the **mpHealth-2.2** feature in Open Liberty, which implements the MicroProfile mpHealth-2.2 API, to implement the new health checks for the application.

The **mpHealth-2.2** feature provides a **/health** endpoint that represents a binary status, either UP or DOWN, of the microservices that are installed.

To learn more about the MicroProfile mpHealth feature, visit:

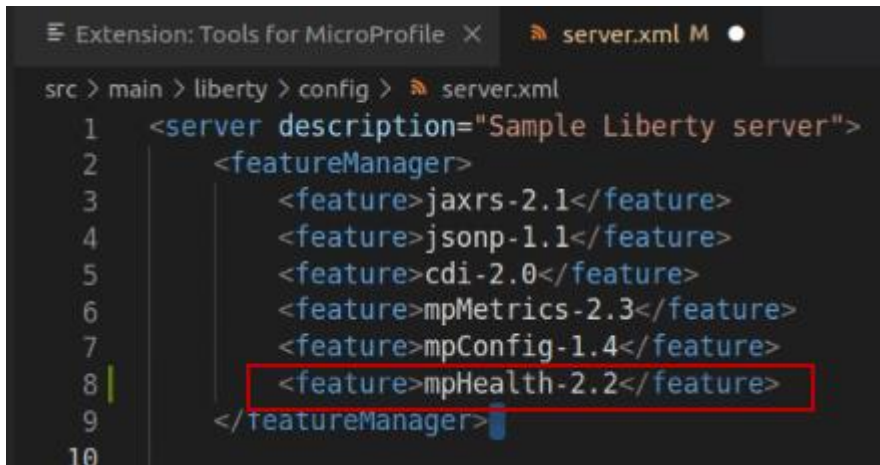
<https://www.openliberty.io/docs/21.0.0.4/health-check-microservices.html>

- \_\_\_1. Update the Liberty server configuration file (server.xml) to include the mpHealth-2.2 feature to begin implementing the health checks for the application.
  - a. In the VS Code Explorer view, navigate to **START > src > main > liberty / config**
  - b. Click on **server.xml** to open the file in the editor pane



- c. Add the mpHealth-2.2 feature to the server.xml file using the text below:

```
<feature>mpHealth-2.2</feature>
```

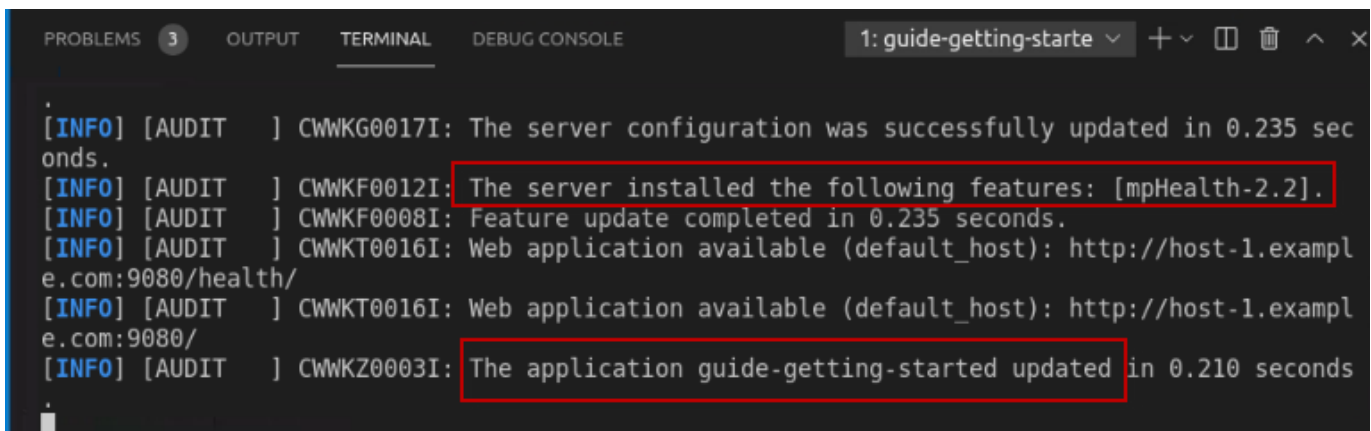


```
src > main > liberty > config > server.xml
1  <server description="Sample Liberty server">
2      <featureManager>
3          <feature>jaxrs-2.1</feature>
4          <feature>jsonp-1.1</feature>
5          <feature>cdi-2.0</feature>
6          <feature>mpMetrics-2.3</feature>
7          <feature>mpConfig-1.4</feature>
8          <feature>mpHealth-2.2</feature>
9      </featureManager>
10
```

d. **Save** and **Close** the server.xml file

When the server.xml file is saved, the configuration changes are detected, and the server is dynamically updated, installing the new feature and updating the application in the running server.

- \_\_2. View the messages in the **Terminal** view, showing the feature being installed and the application being updated.

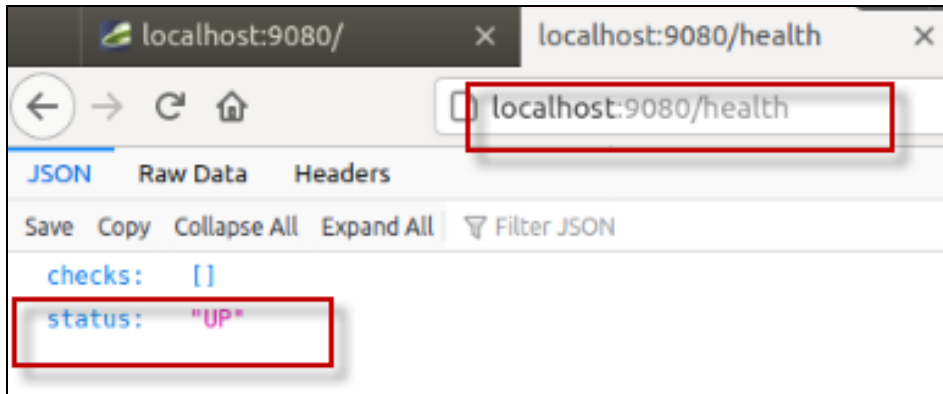


```
PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE 1: guide-getting-started
[INFO] [AUDIT ] CWWKG0017I: The server configuration was successfully updated in 0.235 seconds.
[INFO] [AUDIT ] CWWKF0012I: The server installed the following features: [mpHealth-2.2].
[INFO] [AUDIT ] CWWKF0008I: Feature update completed in 0.235 seconds.
[INFO] [AUDIT ] CWWKT0016I: Web application available (default_host): http://host-1.example.com:9080/health/
[INFO] [AUDIT ] CWWKT0016I: Web application available (default_host): http://host-1.example.com:9080/
[INFO] [AUDIT ] CWWKZ0003I: The application guide-getting-started updated in 0.210 seconds
```

Once the changes are saved, and the server is automatically updated, the new **/health** endpoint is available.

- \_\_3. From the Firefox browser in the VM access the **/health** endpoint to view the health status of the application.

```
http://localhost:9080/health
```

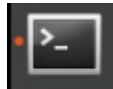


Currently, the basic health check provides a simple status indicating if the service is running, but not if it is healthy.

In the next steps, you will implement a **liveness** check that implements logic that gathers memory and cpu usage information and reports the service DOWN in the health check if the system resources exceed a certain threshold.

You will also implement a **readiness** check that checks external property configuration in the server.xml file, that is used to place the service in maintenance mode. And if the service is in maintenance mode, the service is marked DOWN from the health check.

\_\_4. Copy an implementation of the **SystemReadinessCheck.java** to the project



- a. Open a Terminal window on the VM
- b. Run the following command to copy the **SystemReadinessCheck.java** to the project

```
cp /home/ibmdemo/Student/labs/devmode/guide-getting-started/finish/src/main/java/io/openliberty/sample/system/SystemReadinessCheck.java /home/ibmdemo/Student/labs/devmode/guide-getting-started/start/src/main/java/io/openliberty/sample/system/SystemReadinessCheck.java
```



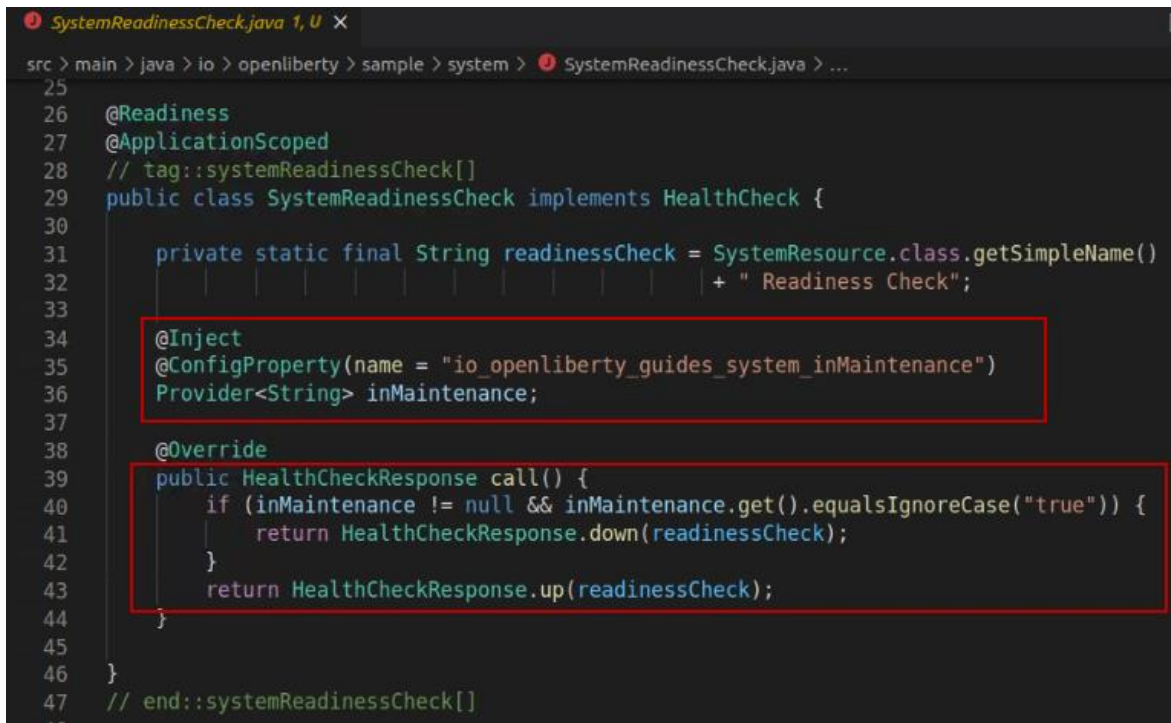
**Information:**

For the purposes of the lab, the copy command above copies a fully implemented Readiness check from the “finished” project, into the current working project.

\_\_5. Review the **SystemReadinessCheck.java** implementation

- a. Return to the VS Code Explorer view

- b. Navigate to **START > main > java / io / openliberty / sample / system**
- c. Click on the **SystemReadinessCheck.java** file to open it in the editor pane



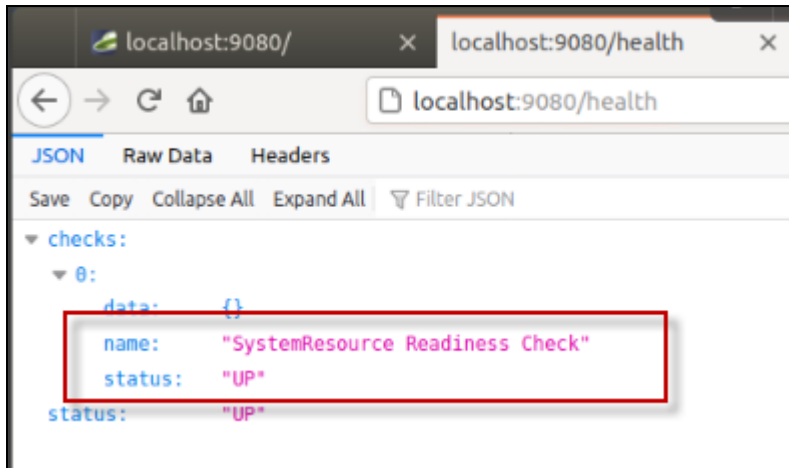
```
25
26 @Readiness
27 @ApplicationScoped
28 // tag::systemReadinessCheck[]
29 public class SystemReadinessCheck implements HealthCheck {
30
31     private static final String readinessCheck = SystemResource.class.getSimpleName()
32         + " Readiness Check";
33
34     @Inject
35     @ConfigProperty(name = "io_openliberty_guides_system_inMaintenance")
36     Provider<String> inMaintenance;
37
38     @Override
39     public HealthCheckResponse call() {
40         if (inMaintenance != null && inMaintenance.get().equalsIgnoreCase("true")) {
41             return HealthCheckResponse.down(readinessCheck);
42         }
43         return HealthCheckResponse.up(readinessCheck);
44     }
45
46 }
47 // end::systemReadinessCheck[]
```

The SystemReadinessCheck simply evaluates the “**inMaintenance**” ConfigProperty, which is implemented via the mpConfig MicroProfile feature, and configured in the Liberty Server’s server.xml file.

- If the “inMaintenance” property is set to “**false**” the readiness check sets the Health Status to UP.
- If the inMaintenance property is set to “**true**” the status is set to DOWN.

- \_\_\_6. From the Firefox Browser in the VM, rerun the **/health** endpoint to view the health status of the application.

```
http://localhost:9080/health
```



#### Information:



Did you notice that while implementing the new readiness check code in the application, that you did not have to restart the application or Liberty Server?

The Open Liberty Tools detected the code changes in the project, and dynamically updated the application in the running server.

\_\_7. Copy an implementation of the **SystemLivenessCheck.java** to the project



- a. Open a Terminal window on the VM
- b. Run the following command to copy the **SystemLivenessCheck.java** to the project

```
cp /home/ibmdemo/Student/labs/devmode/guide-getting-started/finish/src/main/java/io/openliberty/sample/system/SystemLivenessCheck.java /home/ibmdemo/Student/labs/devmode/guide-getting-started/start/src/main/java/io/openliberty/sample/system/SystemLivenessCheck.java
```

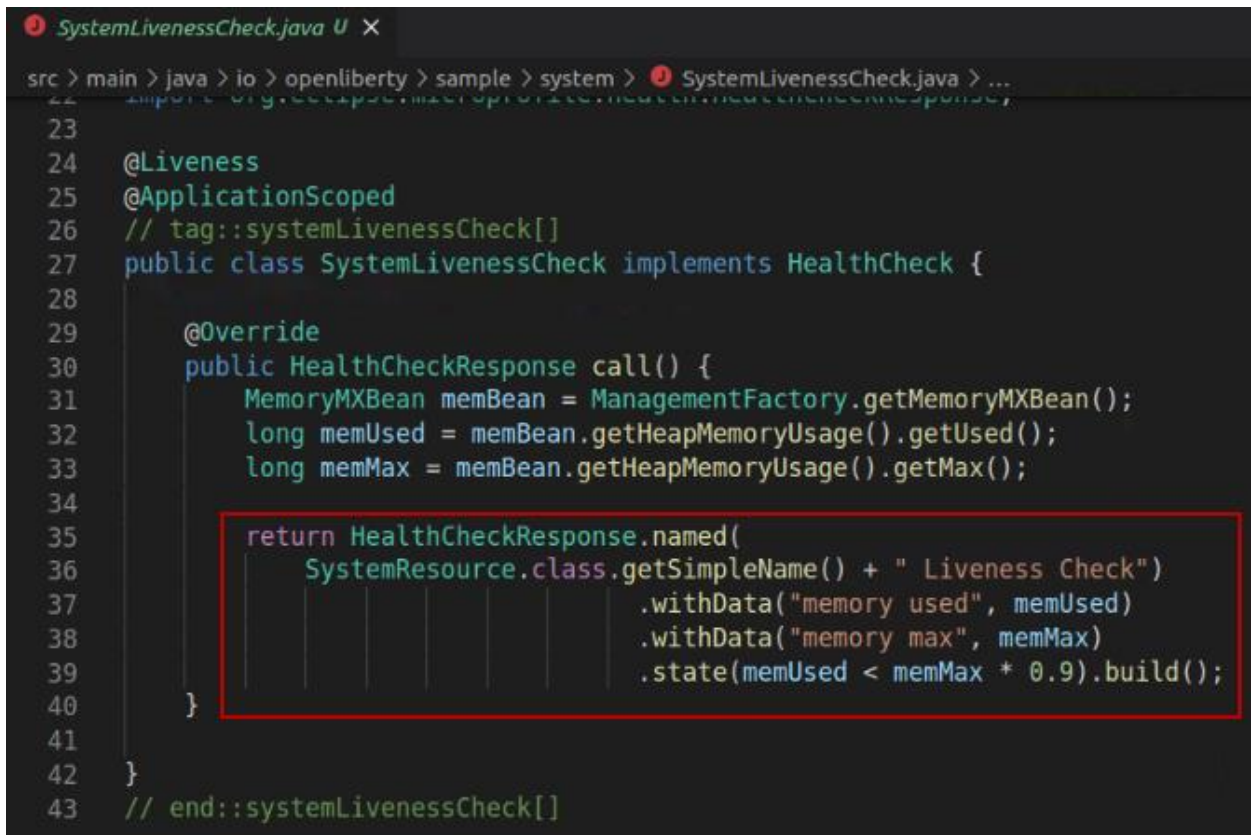
#### Information:



For the purposes of the lab, the copy command above copies a fully implemented Liveness check from the “finished” project, into the current working project.



- \_\_8. Review the **SystemLivenessCheck.java** implementation
- Return to the VS Code Explorer view
  - Navigate to **START > main > java / io / openliberty / sample / system**
  - Click on the **SystemLivenessCheck.java** file to open it in the editor pane



```
22 import org.eclipse.microprofile.health.HealthCheckResponse;
23
24 @Liveness
25 @ApplicationScoped
26 // tag::systemLivenessCheck[]
27 public class SystemLivenessCheck implements HealthCheck {
28
29     @Override
30     public HealthCheckResponse call() {
31         MemoryMXBean memBean = ManagementFactory.getMemoryMXBean();
32         long memUsed = memBean.getHeapMemoryUsage().getUsed();
33         long memMax = memBean.getHeapMemoryUsage().getMax();
34
35         return HealthCheckResponse.named(
36             SystemResource.class.getSimpleName() + " Liveness Check")
37             .withData("memory used", memUsed)
38             .withData("memory max", memMax)
39             .state(memUsed < memMax * 0.9).build();
40     }
41
42 }
43 // end::systemLivenessCheck[]
```

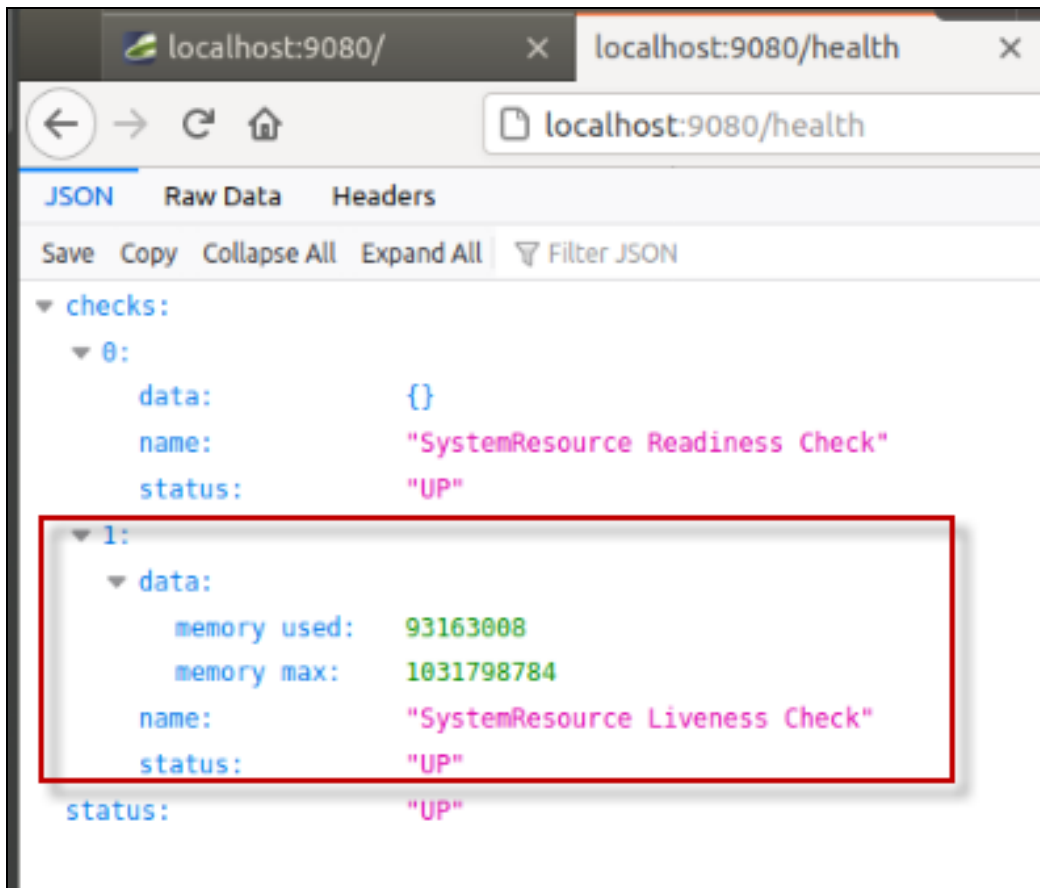
The SystemLivenessCheck evaluates the “**memory**” and “**cpu**” resources used.

- If the “memory” used is less than 90%, the liveness probe sets the status to UP.
- If the “memory” used is greater than 90%, the liveness probe sets the status to DOWN.

- \_\_9. From the Firefox Browser in the VM, rerun the **/health** endpoint to view the health status of the application.

```
http://localhost:9080/health
```





**Note:** in the case where there are multiple health checks being performed, as in our example, ALL the health checks must have the UP status for the service to be marked UP.

**So, what happens when we change the inMaintenance property to “true”?**

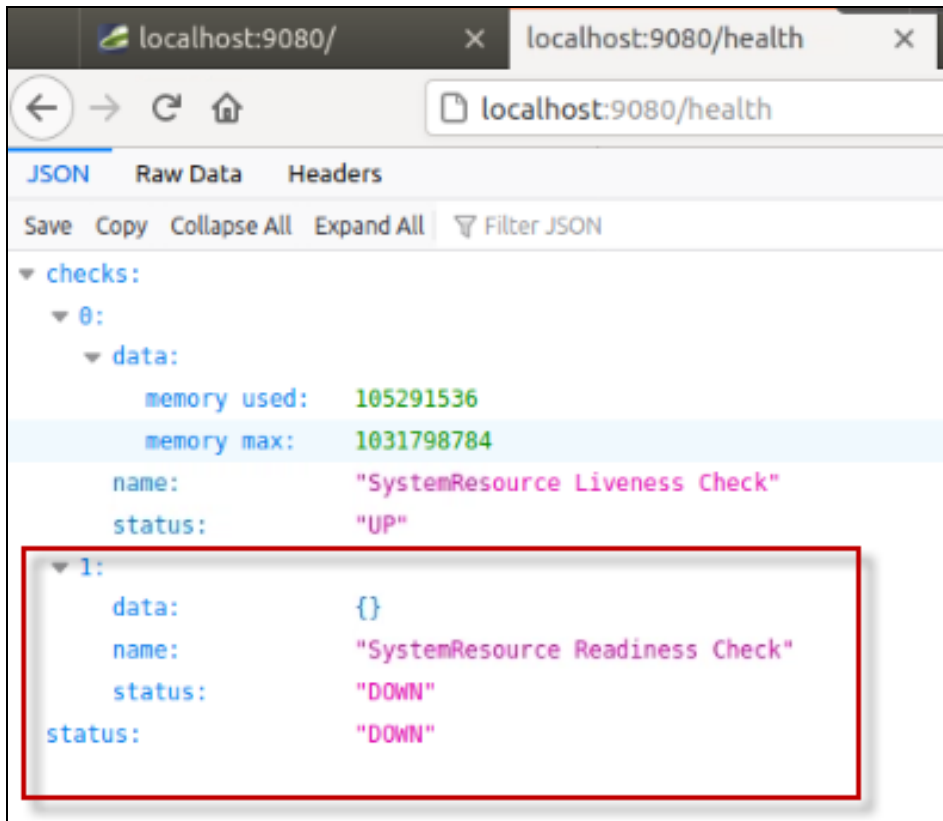
Let's modify the external configuration to set the service in maintenance mode and see the results of the health checks.

- \_\_10. Modify the inMaintenance property in the server.xml file
  - a. Return to the VS Code console and navigate to **START > src > main > liberty / config**
  - b. Click on **server.xml** to open the file in the editor
  - c. Modify the inMaintenance variable value to “**true**” as illustrated below
  - d. **Save** the server.xml file. The server configuration is dynamically updated to reflect the update.

```
server.xml M ●
src > main > liberty > config > server.xml
1  <server description="Sample Liberty server">
2    <featureManager>
3      <feature>jaxrs-2.1</feature>
4      <feature>jsonp-1.1</feature>
5      <feature>cdi-2.0</feature>
6      <feature>mpMetrics-2.3</feature>
7      <feature>mpConfig-1.4</feature>
8      <feature>mpHealth-2.2</feature>
9    </featureManager>
10
11    <variable name="default.http.port" defaultValue="9080"/>
12    <variable name="default.https.port" defaultValue="9443"/>
13
14    <webApplication location="guide-getting-started.war" contextRoot="/" />
15
16    <mpMetrics authentication="false"/>
17
18    <httpEndpoint host="*" httpPort="${default.http.port}"
19      httpsPort="${default.https.port}" id="defaultHttpEndpoint"/>
20
21    <variable name="io_openliberty_guides_system_inMaintenance" value="true"/>
22  </server>
23
```

- \_\_11. From the Firefox Browser in the VM, rerun the **/health** endpoint to view the health status of the application.

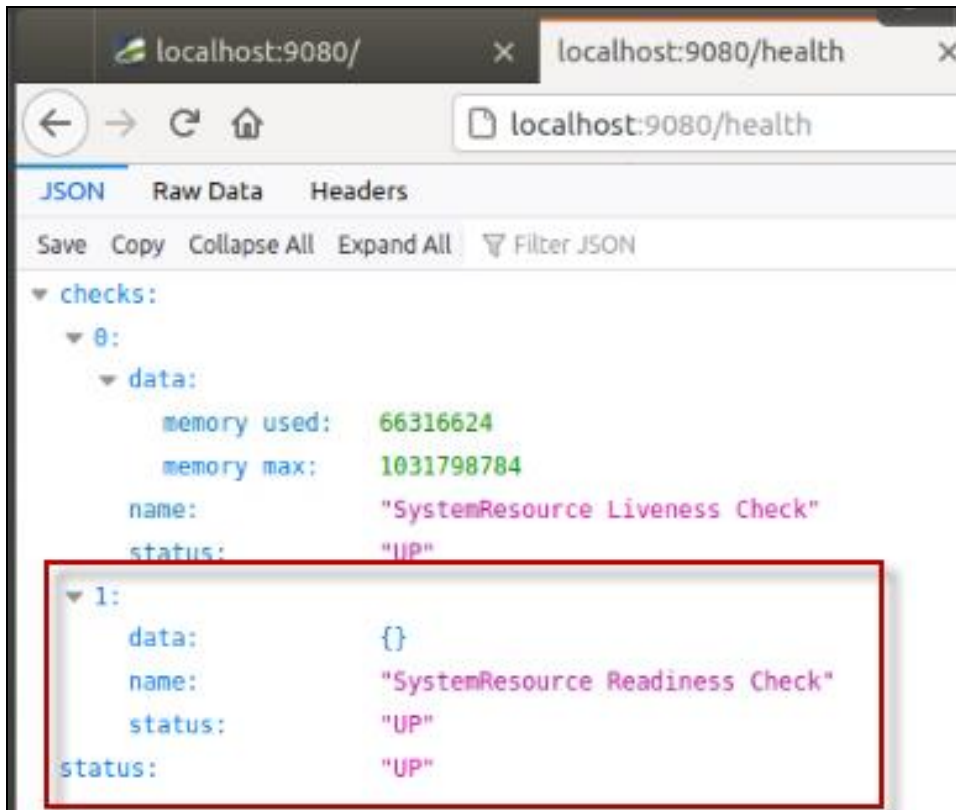
```
http://localhost:9080/health
```



- \_\_12. In the `server.xml` file, change the `inMaintenance` variable back to `false`
- Save** the `server.xml` file
  - Close** the `server.xml` editor view

```
<variable name="io_openliberty_guides_system_inMaintenance" value="false"/>
```

\_\_13. Rerun the **/health** endpoint to verify the service is now marked UP again.



## 1.7.2 Running Tests using the Open Liberty Tools in VS Code

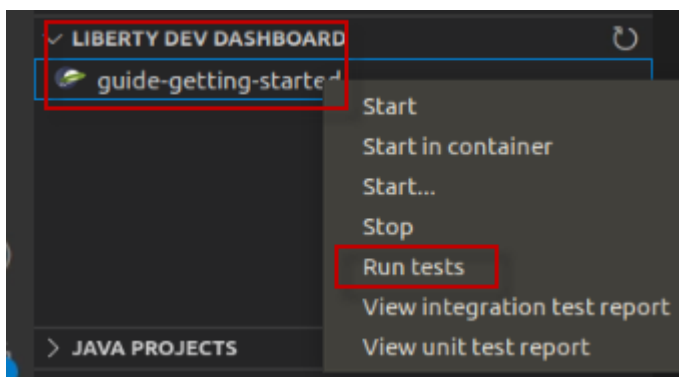
In this section of the lab, you will make some simple changes to the sample application code and run test cases directly from the VS Code IDE using the built-in capabilities in the Open Liberty tools.

To simulate a breaking change in the application code, you will modify the path to the service endpoint from **/properties** to **/all-properties**.

Because the test case attempts to run the system service using the **/properties** path, the test case will fail and return an HTTP Code of 404, rather than the expected response code of 200.

Since the developer is purposely introducing this change, the test case needs to be updated to reflect the new path to the service for the tests to pass.

- \_\_\_1. Use the Liberty Dev Dashboard to **Run Tests** against the System Properties Sample service.
  - a. In VS Code, expand the LIBERTY DEV DASHBOARD section
  - b. Right-mouse click on the **guide-getting-started** Liberty Server
  - c. Select **Run Test** from the menu to run the tests

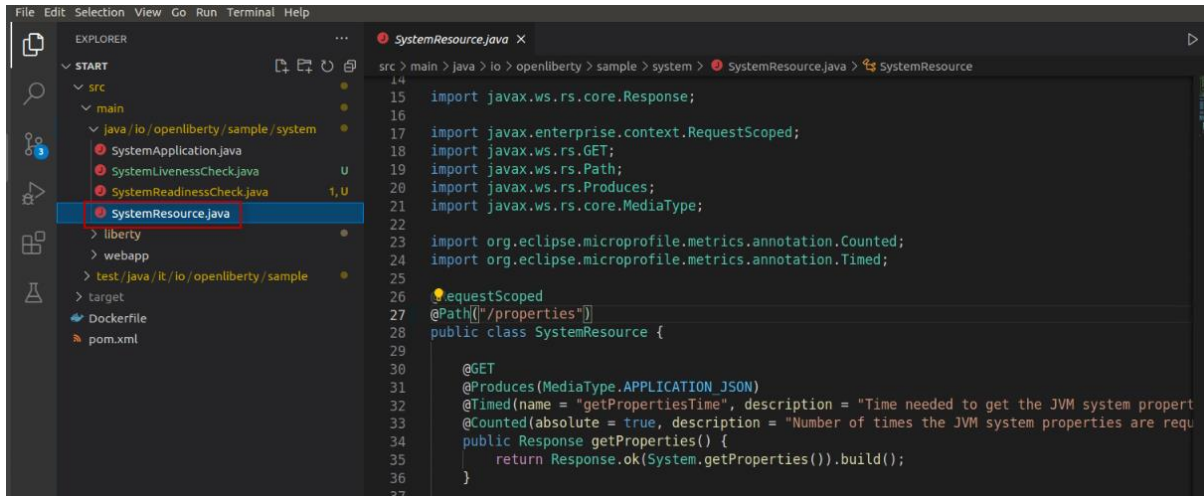


- d. In the Terminal view, you will see the results of the tests. One test was executed, and one test PASSED.

```
PROBLEMS 4 OUTPUT TERMINAL DEBUG CONSOLE
[INFO] -----
[INFO] Running it.io.openliberty.sample.PropertiesEndpointIT
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 s
[INFO] opertiesEndpointIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] Integration tests finished.
[INFO] To run tests on demand, press Enter.
```

Next, as a developer on the project, you have been asked to change the code to specify a different path to the “properties” service. Doing so, has an impact on the tests. In the next few steps, you will make the code change, and update the tests to match the NEW expected results.

- \_\_2. Open the **systemResources.java** in VS Code editor
  - a. In VS Code Explorer view, expand **START > src > main > java / io / openliberty / sample / system**
  - b. Click on **SystemResources.java** to open it in the editor



- \_\_3. Update the **@Path** to the system properties service to specify a different service path
  - a. From the editor, make the following change to the **systemResources.java** file:

**Change the highlighted line:**

```
23 import org.eclipse.microprofile.metrics.annotation.Counted;
24 import org.eclipse.microprofile.metrics.annotation.Timed;
25
26 @RequestScoped
27 @Path("/properties")
28 public class SystemResource {
29
```

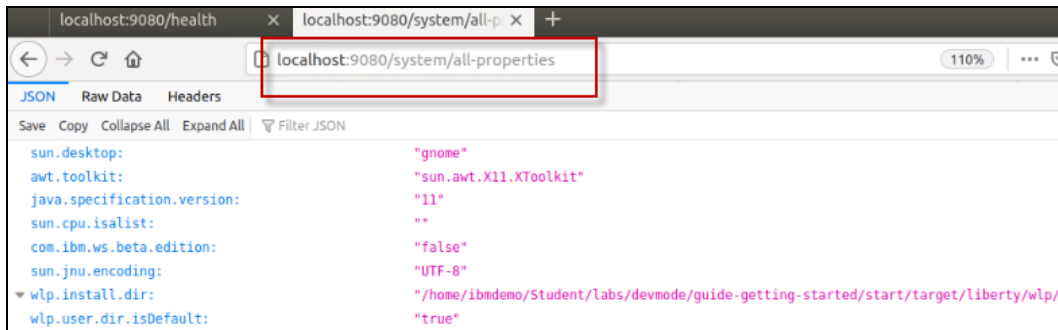
**Updated to read: @Path("/all-properties")**

```
22
23 import org.eclipse.microprofile.metrics.annotation.Counted;
24 import org.eclipse.microprofile.metrics.annotation.Timed;
25
26 @RequestScoped
27 @Path("/all-properties")
28 public class SystemResource {
29
```

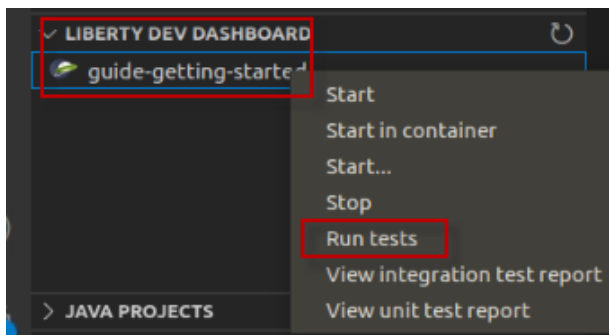
- b. **SAVE** the file. The Liberty server and application are dynamically updated.
- c. **Close** the editor view for the **SystemResource.java** file

\_\_2. From the Firefox browser, run the service using the NEW endpoint URL.

```
http://localhost:9080/system/all-properties
```



- \_\_3. Use the Liberty Dev Dashboard to **Run Tests** against the System Properties Sample service.
  - a. In VS Code, expand the LIBERTY DEV DASHBOARD section
  - b. Right-mouse click on the **guide-getting-started** Liberty Server
  - c. Select **Run Test** from the menu to start the server

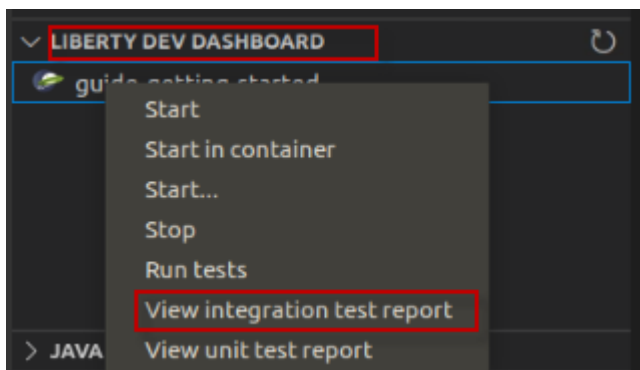


- d. Alternatively, you can run the tests by simply pressing the **ENTER** key in the Terminal window. Give it a try. **The tests now FAIL.**

```
[ERROR] PropertiesEndpointIT.testGetProperties:45 Incorrect response code from http://localhost:9000
ected: <200> but was: <404>
[INFO]
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[ERROR] Integration tests failed: There are test failures.

Please refer to /home/ibmdemo/Student/labs/devmode/guide-getting-started/start/target/failsafe-repo
individual test results.
Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream.
[INFO] To run tests on demand, press Enter.
```

- \_\_4. Use the Liberty Dev Dashboard to **View integration test report**.
  - a. In VS Code, expand the LIBERTY DEV DASHBOARD section
  - b. Right-mouse click on the **guide-getting-started** Liberty Server
  - c. Select **View integration test report** from the menu



- \_\_5. View the test results details in the “**guide-getting-started Failsafe report**” that is now displayed the editor pane
  - a. Notice that the test case failed

### Failsafe Report

#### Summary

[\[Summary\]](#) [\[Package List\]](#) [\[Test Cases\]](#)

Tests	Errors	Failures	Skipped	Success Rate	Time
1	0	1	0	0%	0.369

- b. Scroll to the bottom of the report to see the ERROR message that was produced from the failing test.



Test Cases		
<a href="#">[Summary]</a> <a href="#">[Package List]</a> <a href="#">[Test Cases]</a>		
PropertiesEndpointIT		
	<a href="#">testGetProperties + [ Detail ]</a>	0.347
	Incorrect response code from http://localhost:9080/ ==> expected: <200> but was: <404>	

c. The issue is obvious. Since we changed the endpoint path, the test case assertion failed because it got a HTTP response code of 404 (Not Found) when attempting to run the service using the original path of /properties.

d. **Close** the Failsafe Report in the Editor pane

**NOTE:** In this case, we expected the test case to fail. And as the developer, you must update the test case to match the expected results based on to your code change.

- \_\_\_6. Modify the test case that is included in the application project to invoke the updated path to the service.
  - a. From the Explorer view in VS Code, navigate to **START > src > test / java / it /io /openliberty / sample**
  - b. Click on **PropertiesEndpointIT.java** to open it in an editor pane
  - c. From the editor, make the following change to the **PropertiesEndpointIT.java** file:

**Change the highlighted line: "system/properties"**

```

40 // request
41 WebTarget target = client.target(url + "system/properties");
42 Response response = target.request().get();
43
44 // response
45 assertEquals(200, response.getStatus(), "Incorrect response code from " + url);
46

```

**Updated to read: "system/all-properties"**

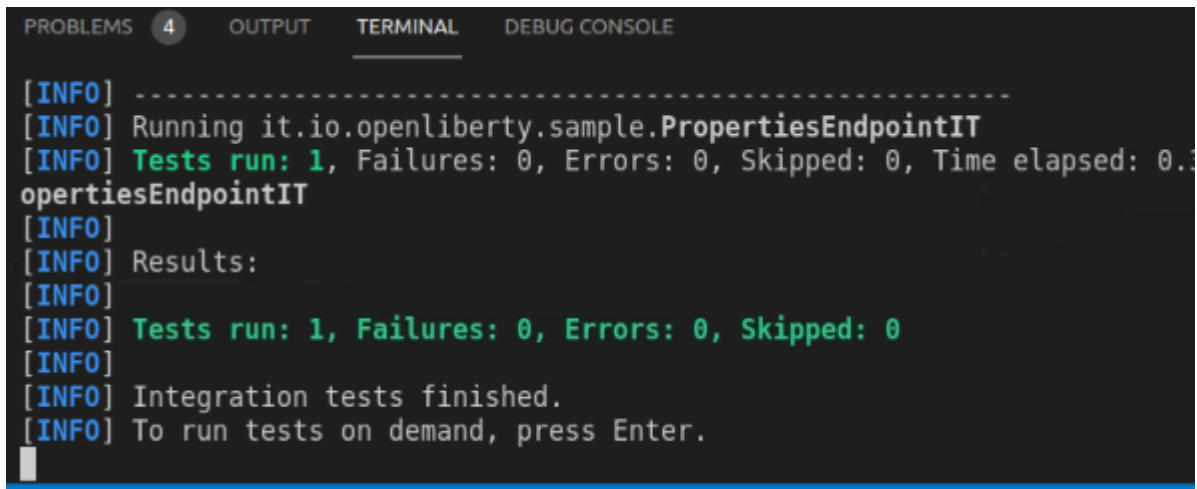
```

40 // request
41 WebTarget target = client.target(url + "system/all-properties");
42 Response response = target.request().get();
43

```

d. **SAVE** and **CLOSE** the file. The Liberty server and application are dynamically updated.

\_\_\_7. Rerun the tests by Pressing the **ENTER** key in the Terminal view. The test PASS.

A screenshot of the Visual Studio Code interface, specifically the Terminal view. The terminal shows the output of a test run. The text is as follows:

```
PROBLEMS 4 OUTPUT TERMINAL DEBUG CONSOLE
[INFO] -----
[INFO] Running it.io.openliberty.sample.PropertiesEndpointIT
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.1
PropertiesEndpointIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] Integration tests finished.
[INFO] To run tests on demand, press Enter.
```

At this point, you have explored using the Liberty Developer Tools to develop code, make server configuration changes, and run test cases to get immediate feedback on the updates.

Using the Open Liberty Tools in VS Code provides an integrated development environment where your updates were automatically detected and dynamically applied to the running server. This provides a rapid inner-loop development cycle for development and testing.

In the next section of the lab, you will explore how simple it is to integrate application debugging in the same development environment without having to restart the Liberty server.

## 1.8 Integrated debugging using the Open Liberty Tools in VS Code

Application debugging is an important part of application development. Developers expect to easily and quickly iterate through **dev – test – debug** without having to leave the development environment or having to restart servers and applications for debugging.

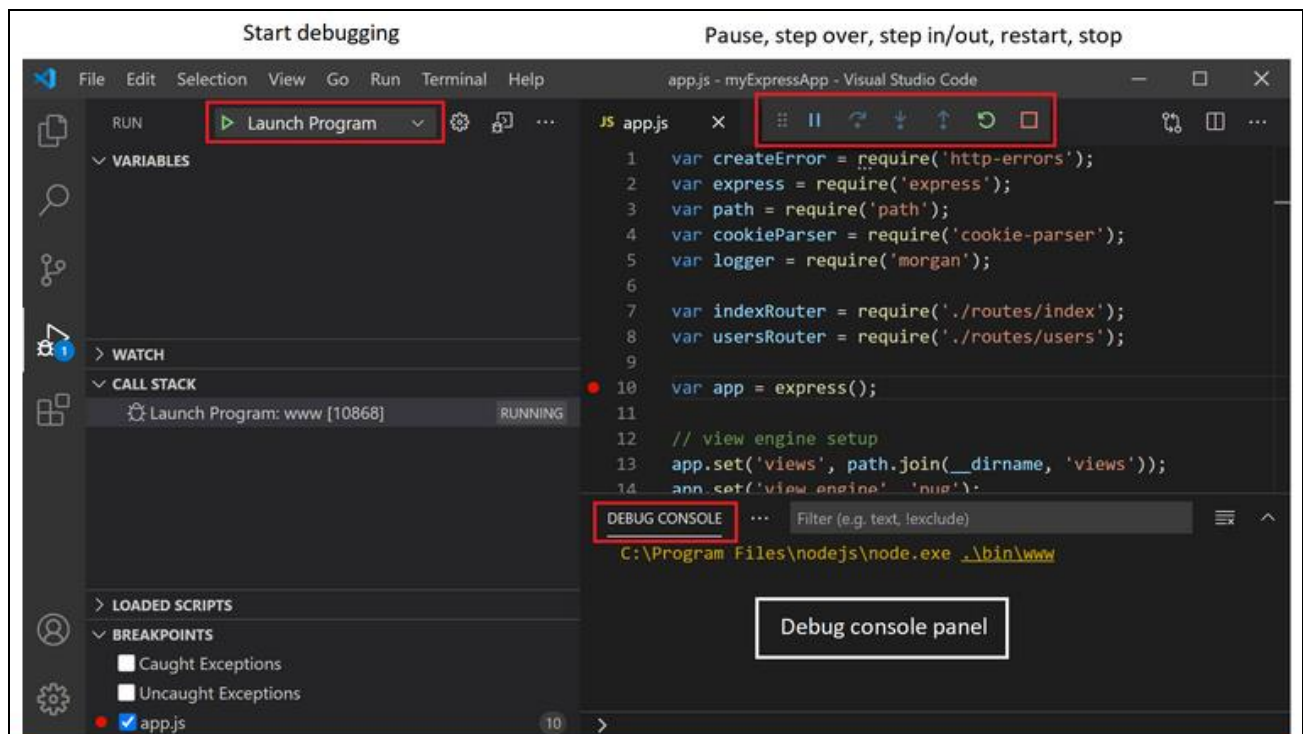
In this section of the lab, you will explore how easy it is for developers to debug their Java application using the integrated development environment and Open Liberty.

### Here are the basics steps for debugging

- Set a breakpoint in the source code
- Add a “Java Attach” in the launch configuration and set the debug port
- Go to Debug view and select the “Attach” configuration
- Click the Start debugging icon
- Run the application in the Browser
- The application stops at the breakpoint
- Step through the app in debug mode to explore the variables and code to resolve issues

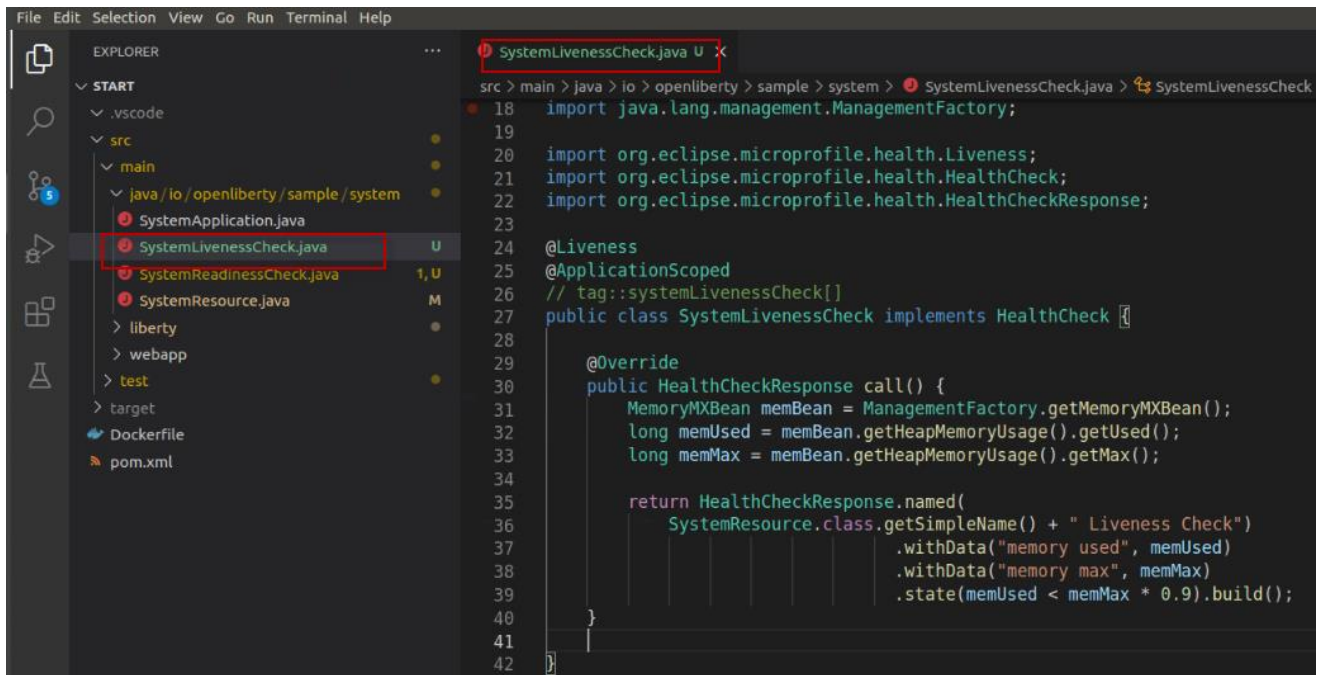
One of the key features of Visual Studio Code is its great debugging support.

In this section of the lab, you will use VS Code debugger to debug the Java application running on Liberty server.



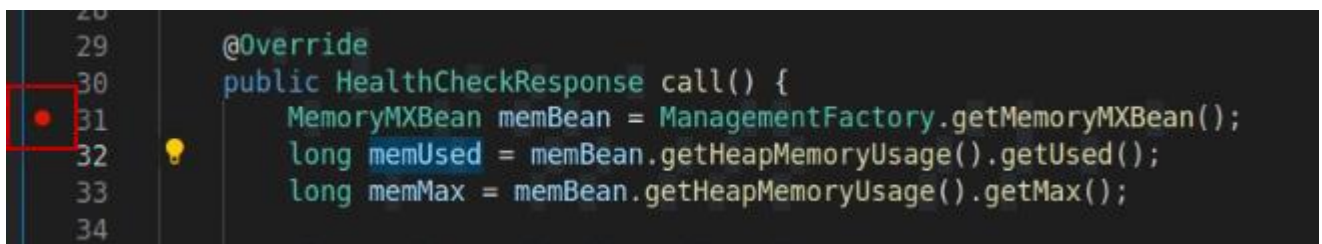
In this scenario, you will set a breakpoint and debug the `SystemLivenessCheck.java` code that is executed when running the `/health` endpoint in the application.

- \_\_1. Open the **SystemLivenessCheck.java** in VS Code editor
  - a. In VS Code Explorer view, expand **START > src > main > java / io / openliberty / sample / system**
  - b. Click on **SystemLivenessCheck.java** to open it in the editor

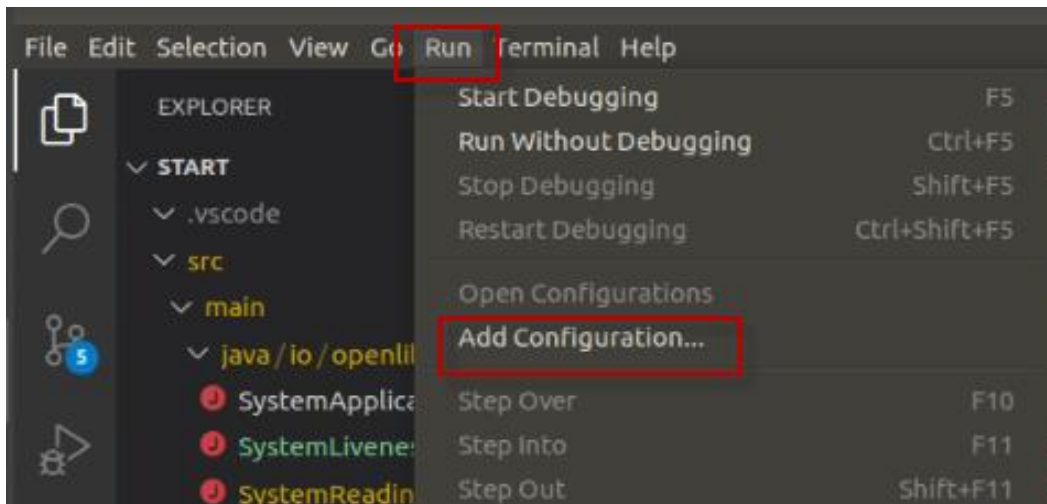


- \_\_2. Set a breakpoint in the code where the **MemoryMaxBean** variable is set
  - a. Locate the line with the text:  

```
MemoryMXBean memBean = ManagementFactory.getMemoryMXBean();
```
  - b. **Left-mouse click** on the left side of the Line Number (31 in the screen shot) to set a breakpoint. A RED dot will appear, indicating the breakpoint is set

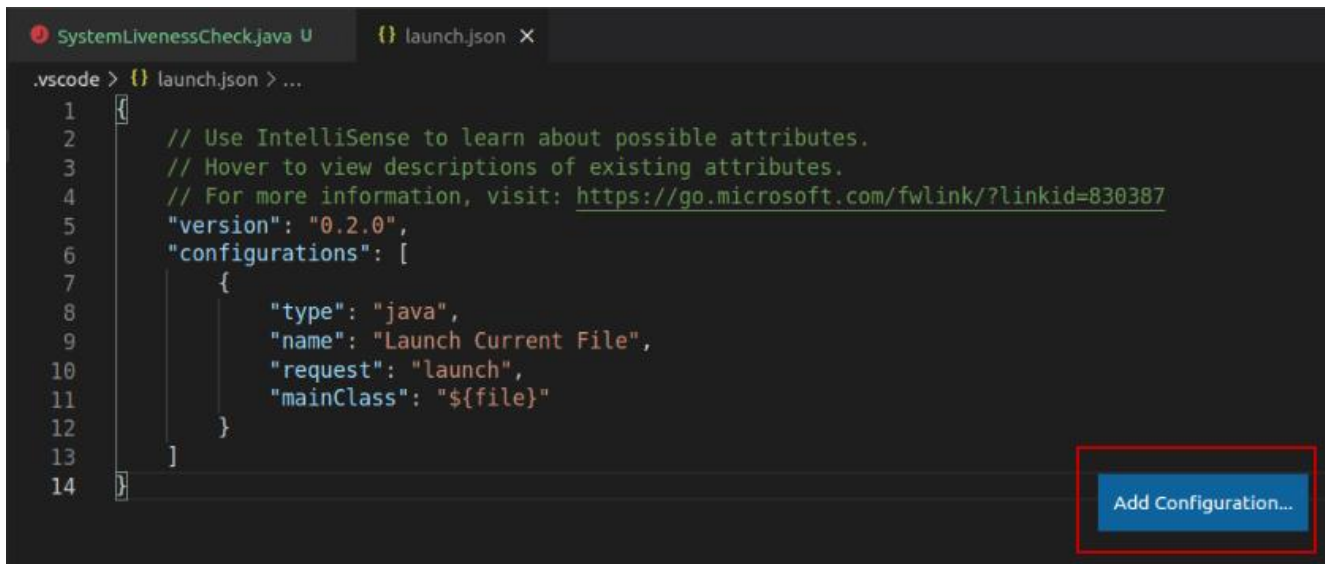


- \_\_3. Create a new **Java Attach configuration** and specify the debug port **7777**
- a. Select **Run > Add Configuration...** from the main menu in VS Code

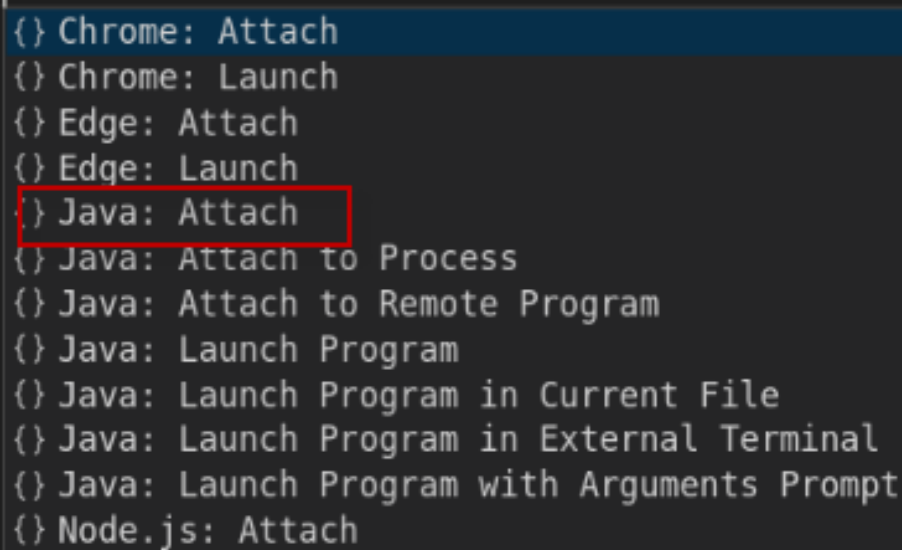


A new file named **launch.json** file was created in the **.vscode** directory. You can see the new file in the explorer view.

- b. In the **launch.json** file that opened in the Editor view, click on the “**Add Configuration**” button located on the lower right corner of the screen.



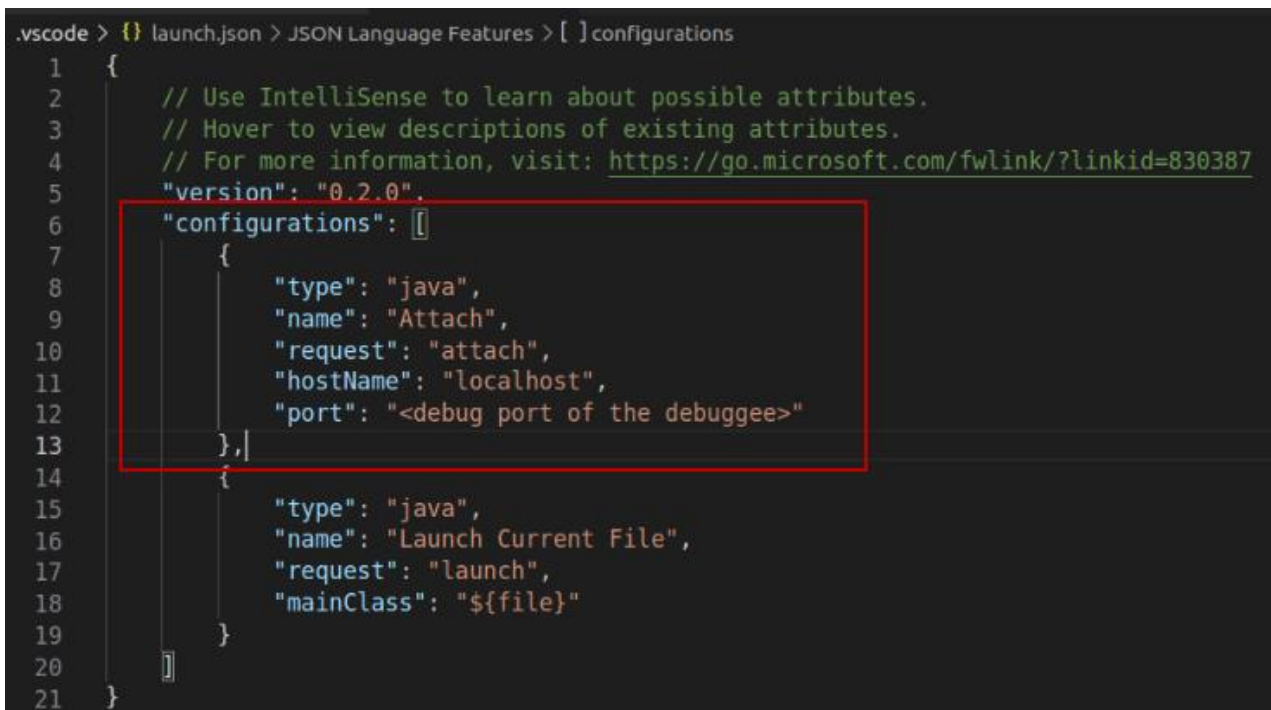
- c. Select **Java: Attach** from the menu.



```
{ } Chrome: Attach
{ } Chrome: Launch
{ } Edge: Attach
{ } Edge: Launch
{ } Java: Attach
{ } Java: Attach to Process
{ } Java: Attach to Remote Program
{ } Java: Launch Program
{ } Java: Launch Program in Current File
{ } Java: Launch Program in External Terminal
{ } Java: Launch Program with Arguments Prompt
{ } Node.js: Attach
```

- d. A new configuration is added to the launch.json file, that includes a “**port**” parameter to attach the debugger for Open Liberty.

**Note:** Open Liberty is configured to use debug port 7777 by default.



```
.vscode > { } launch.json > JSON Language Features > [ ] configurations
1  {
2      // Use IntelliSense to learn about possible attributes.
3      // Hover to view descriptions of existing attributes.
4      // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "type": "java",
9              "name": "Attach",
10             "request": "attach",
11             "hostName": "localhost",
12             "port": "<debug port of the debuggee>"
13         },
14         {
15             "type": "java",
16             "name": "Launch Current File",
17             "request": "launch",
18             "mainClass": "${file}"
19         }
20     ]
21 }
```



\_\_4. Change the “port” parameter to 7777

a. From the editor, make the following change to the **lauch.json** file:

**Change the highlighted line:** “port”: “<debug port of the debugger>”

```
6      "configurations": [  
7          {  
8              "type": "java",  
9              "name": "Attach",  
10             "request": "attach",  
11             "hostName": "localhost",  
12             "port": "<debug port of the debugger>"  
13         },  
14     ]  
15 }
```

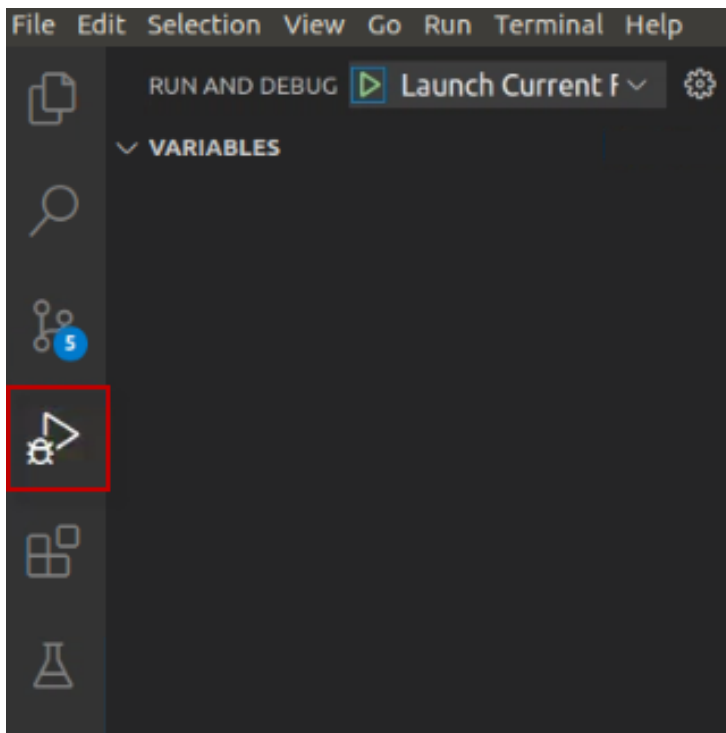
**Updated to read:** “port”: 7777

**Note** Be sure to REMOVE the double quotes around 7777, as illustrated below.

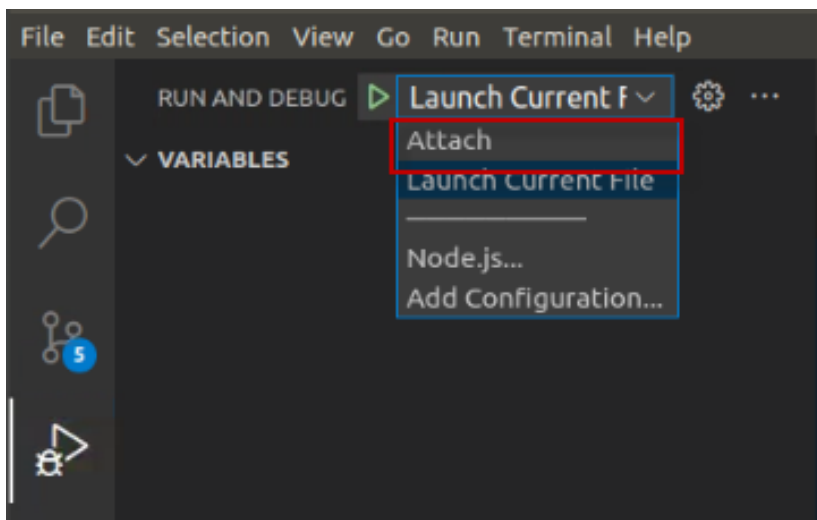
```
6      "configurations": [  
7          {  
8              "type": "java",  
9              "name": "Attach",  
10             "request": "attach",  
11             "hostName": "localhost",  
12             "port": 7777  
13         },  
14     ]  
15 }
```

b. **SAVE and CLOSE** the file. The Liberty server and application are dynamically updated.

- \_\_5. Now, attach the new Java Attach configuration
- Switch to the **Debug** perspective in VS Code, by selecting the **Debug Icon** on the left side navigation menu

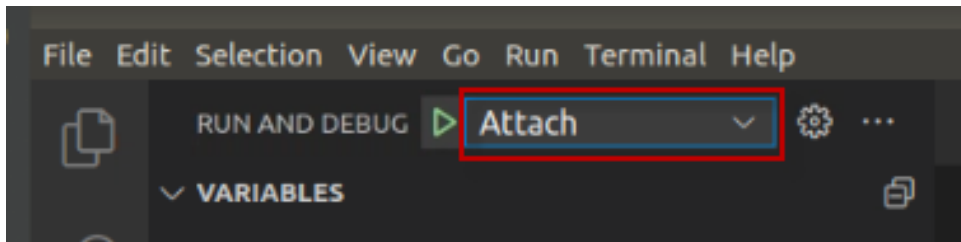


- Using the launch Drop-down menu in the Debug perspective, set the **Launch action** to the “**Attach**” configuration that you created.

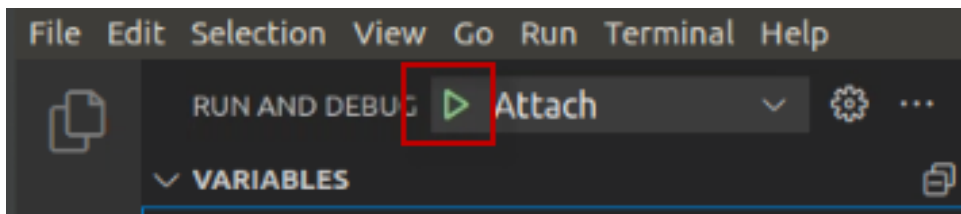




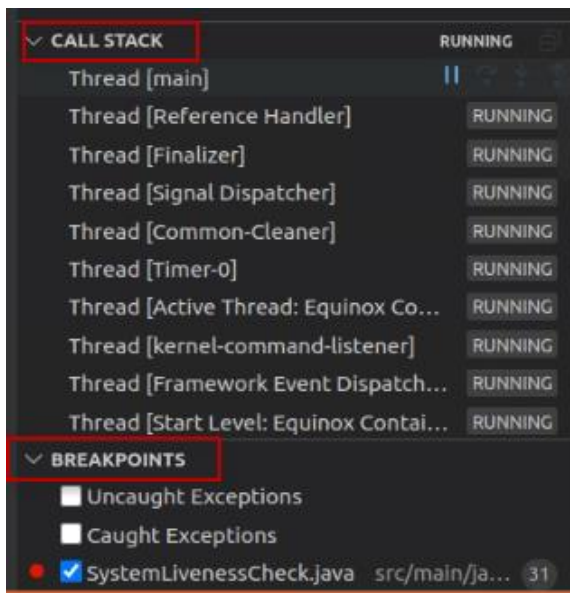
c. The “Attach” configuration is now selected. You are ready to debug.



\_\_6. Click on the **Start** Icon to start the debugger.



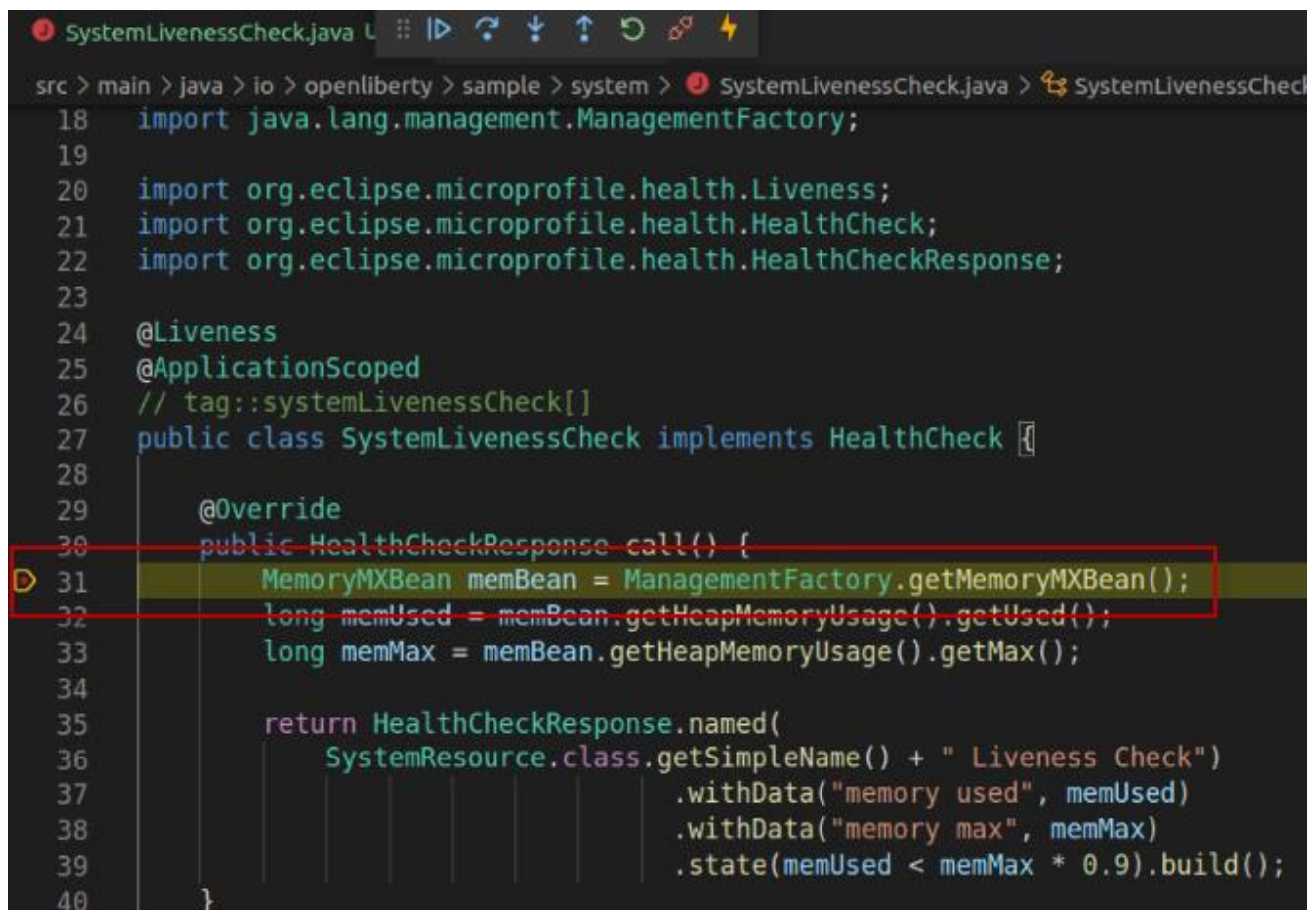
The debugger is now attached, and the CALL STACK and BREAKPOINTS are displayed in the Debug perspective, as illustrated below:



- \_\_\_7. From the Firefox Browser in the VM, run the **/health** endpoint to view the health status of the application. The application will stop at the breakpoint in the SystemLivenessCheck.java code.

```
http://localhost:9080/health
```

In VS Code's Debugger perspective, the application stopped at the breakpoint you set in the SystemLivenessCheck.java, as illustrated below.



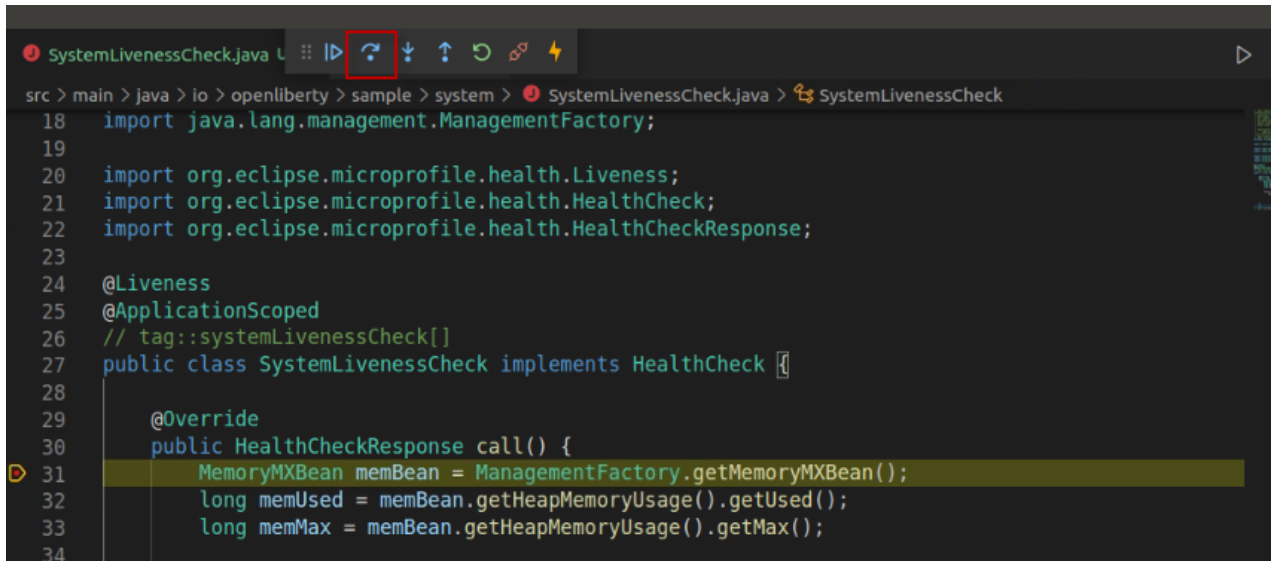
The screenshot shows the VS Code editor with the file SystemLivenessCheck.java open. The code is as follows:

```
18 import java.lang.management.ManagementFactory;
19
20 import org.eclipse.microprofile.health.Liveness;
21 import org.eclipse.microprofile.health.HealthCheck;
22 import org.eclipse.microprofile.health.HealthCheckResponse;
23
24 @Liveness
25 @ApplicationScoped
26 // tag::systemLivenessCheck[]
27 public class SystemLivenessCheck implements HealthCheck {
28
29     @Override
30     public HealthCheckResponse call() {
31         MemoryMXBean memBean = ManagementFactory.getMemoryMXBean();
32         long memUsed = memBean.getHeapMemoryUsage().getUsed();
33         long memMax = memBean.getHeapMemoryUsage().getMax();
34
35         return HealthCheckResponse.named(
36             SystemResource.class.getSimpleName() + " Liveness Check")
37             .withData("memory used", memUsed)
38             .withData("memory max", memMax)
39             .state(memUsed < memMax * 0.9).build();
40     }
41 }
```

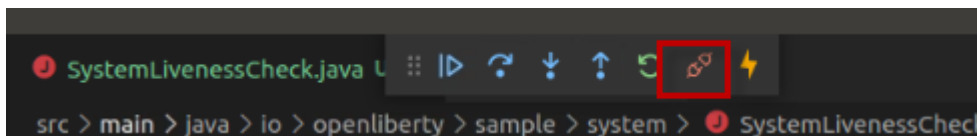
A red bracket highlights the code block from line 31 to line 39. A yellow breakpoint icon is placed on the left margin next to line 31.

\_\_8. Now you can use the “step Over”, “Step In” “Step Out”,” Run” or “Disconnect” actions.

- a. Click the “Step Over” to execute the existing line of code and step to the next line of code in the application.

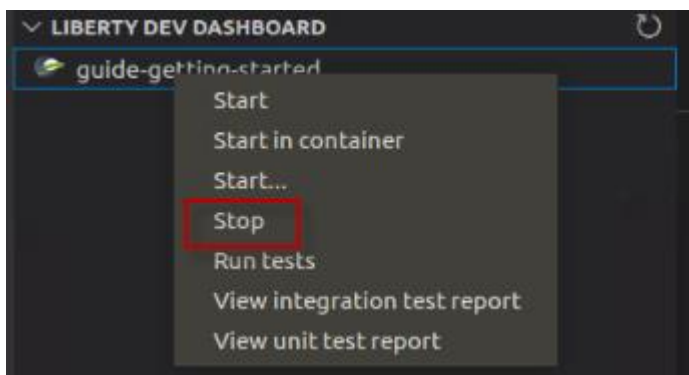


\_\_9. When you are finished stepping through the debugger and exploring the local variables, click the **Disconnect** icon to disconnect the debugger



\_\_10. Use the Liberty Dev Dashboard to **STOP** the Liberty Server in dev mode

- c. In VS Code, expand the LIBERTY DEV DASHBOARD section
- d. Right-mouse click on the **guide-getting-started** Liberty Server
- e. Select **Stop** from the menu to stop the server



- \_\_11. **Exit** the VS Code UI
  - a. Select **File > Exit** from the main menu in VS Code to Exit the UI
- \_\_12. **Close** all opened **Terminal** Windows and **Browser** tabs

Congratulations! You have successfully used the **Liberty Dev VS Code extension** to start Open Liberty in development mode, make changes to your application and Liberty server configuration while the server is up, run tests and view results, and even debug the application without leaving the editor.

As you explored the fast and efficient inner-loop development experience using the Open Liberty tools and VS Code IDE, your code was automatically compiled and deployed to your running server, making it easy to iterate on your changes.

**===== END OF LAB =====**