

Lab 3: Getting Started with Liberty and Dev Mode



Contents

Contents

LAB 3 GETTING STARTED WITH LIBERTY AND DEV MODE 1

- 1.1 OBJECTIVES 1
- 1.2 LAB REQUIREMENTS..... 2
- 1.4 INTRODUCTION - LIBERTY AND "DEV" MODE 3
- 1.5 THE LAB ENVIRONMENT 4
 - 1.5.1 LOGIN TO THE "LIBERTY vPOT ... DESKTOP" VM AND GET STARTED..... 5**
- 1.6 GETTING STARTED WITH LIBERTY AND DEV MODE 8
 - 1.6.1 BUILDING AND RUNNING THE APPLICATION USING MAVEN AND THE LIBERTY-MAVEN-PLUGIN..... 8**
 - 1.6.2 UPDATING THE APPLICATION WITHOUT RESTARTING THE SERVER..... 11**
 - 1.6.3 UPDATING THE SERVER CONFIGURATION WITHOUT RESTARTING THE SERVER 15**
- 1.7 DEVELOPING AND RUNNING THE APPLICATION IN A DOCKER CONTAINER AND IN LIBERTY DEV MODE 19
 - 1.7.1 INVESTIGATE DOCKER COMMANDS AND DOCKERFILE FOR BUILDING IMAGES 19**
 - 1.7.2 RUNNING THE APPLICATION IN A CONTAINER..... 23**
 - 1.7.3 USING DEV MODE TO DEVELOP AN APPLICATION IN A DOCKER CONTAINER 31**
- 1.8 LAB CLEANUP AND COMPLETION..... 37

Lab 3 Getting Started with Liberty and Dev Mode

1.1 Objectives

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

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

- Use Liberty dev mode (stand-alone) without an IDE
- Experience hot reloading of application code and configuration changes using dev mode
- Work with Liberty dev mode in containers
- Run integrated unit/integration tests from Liberty dev mode

You will need an estimated **45 to 60 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 have 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 - Liberty and "dev" Mode

Open Liberty is an application server designed for the cloud. It's small, lightweight, and designed with modern cloud-native application development in mind.

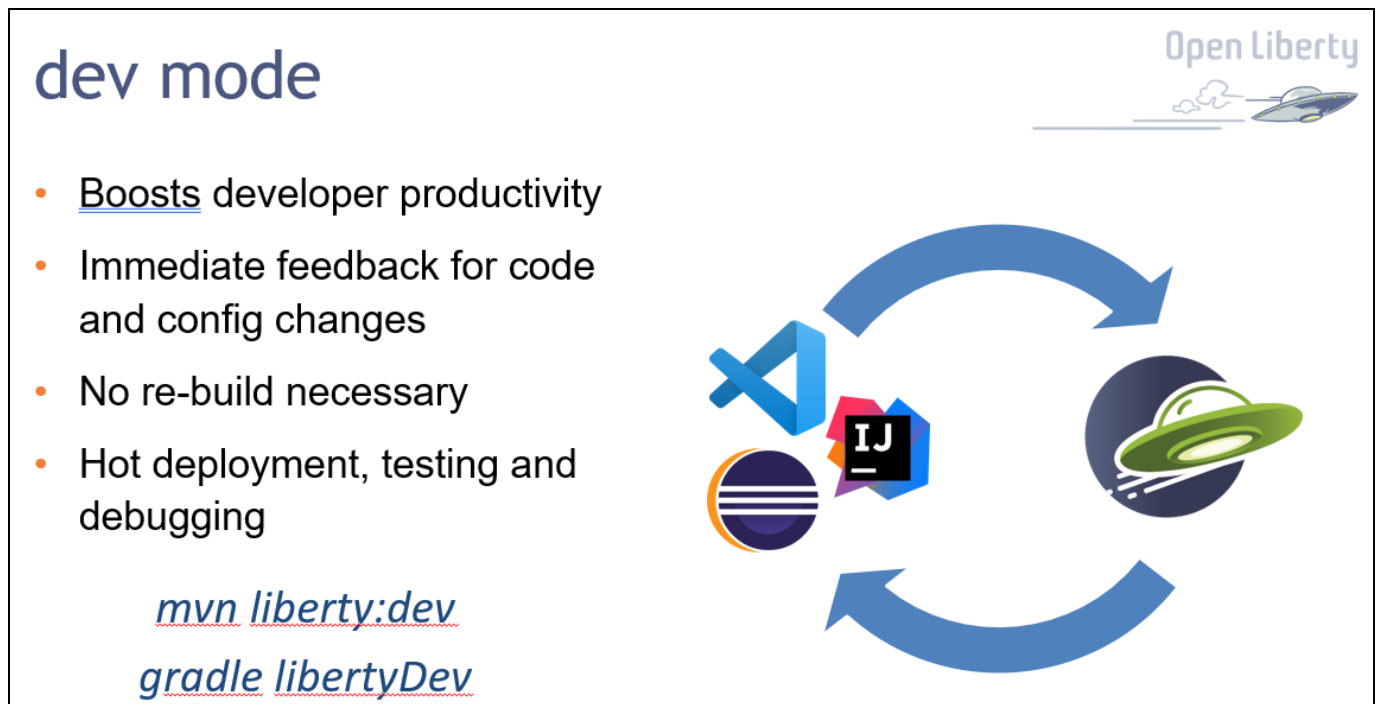
Open Liberty supports the full MicroProfile and Jakarta EE APIs and is composable, meaning that you can use only the features that you need, keeping the server lightweight, which is great for microservices. It also deploys to every major cloud platform, including Docker, Kubernetes, and Cloud Foundry.

Maven is an automation build tool that provides an efficient way to develop Java applications. Using Maven, you will build a simple microservice, called **system**, that collects basic system properties from your laptop and displays them on an endpoint that you can access in your web browser.

Open Liberty **development mode**, or dev mode, allows you to develop applications with any text editor or IDE by providing hot reload and deployment, on demand testing, and debugger support. Open Liberty Dev Mode is enabled through **Maven** and **Gradle** projects.

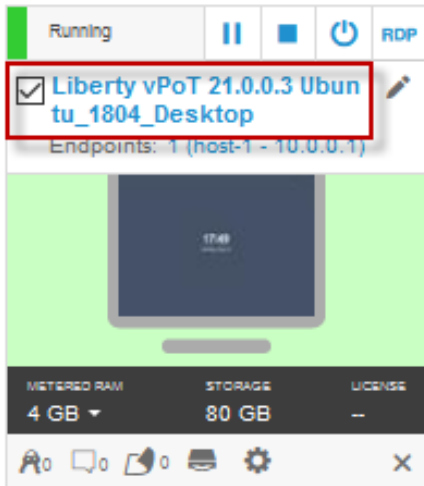
Your code is automatically compiled and deployed to your running server, making it easy to iterate on your changes.

You can run tests on demand or even automatically so that you can get immediate feedback on your changes. You can also attach a debugger at any time to debug your running application.



1.5 The lab environment

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

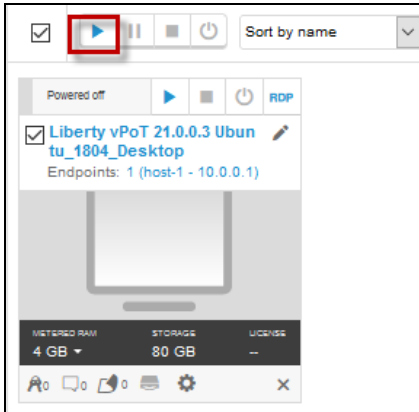


The “**Liberty vPOT ... Desktop**” VM has the following software available:

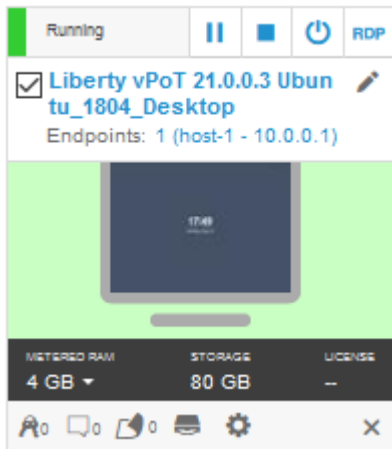
- Application Project with Liberty
- Maven 3.6.0
- The login credentials for the **Liberty vPOT ... Desktop** VM are:
 - User ID: **ibmdemo**
 - Password: **passw0rd** (That is a numeric zero in passw0rd)

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

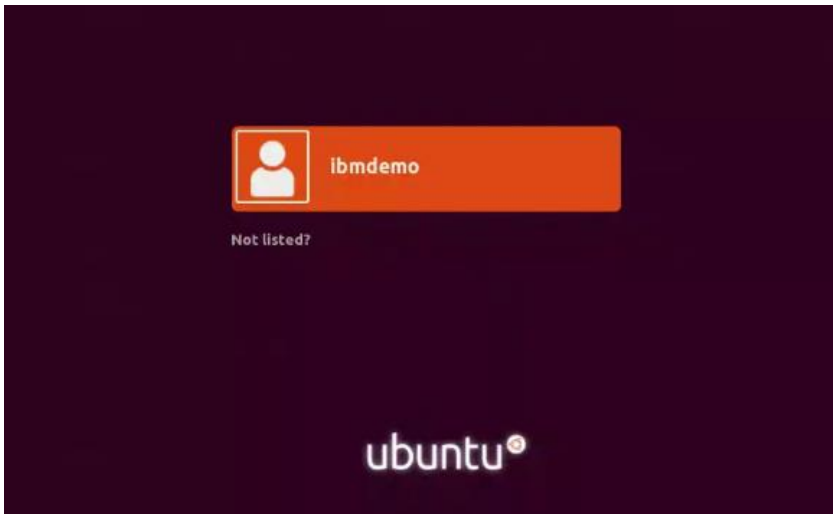
- __1. If the VM is **not** already started, start it by clicking the **Play** button.



- __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.



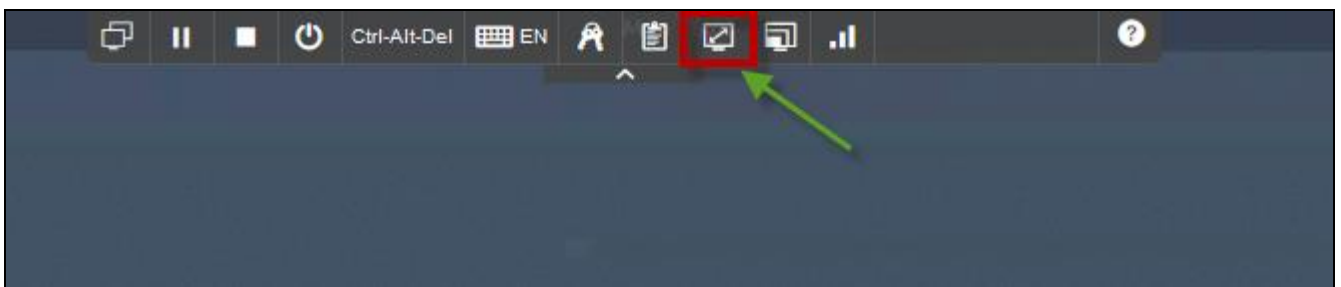
__b. When prompted for the password for “ibmdemo” user, enter “**passw0rd**” as the password:

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.

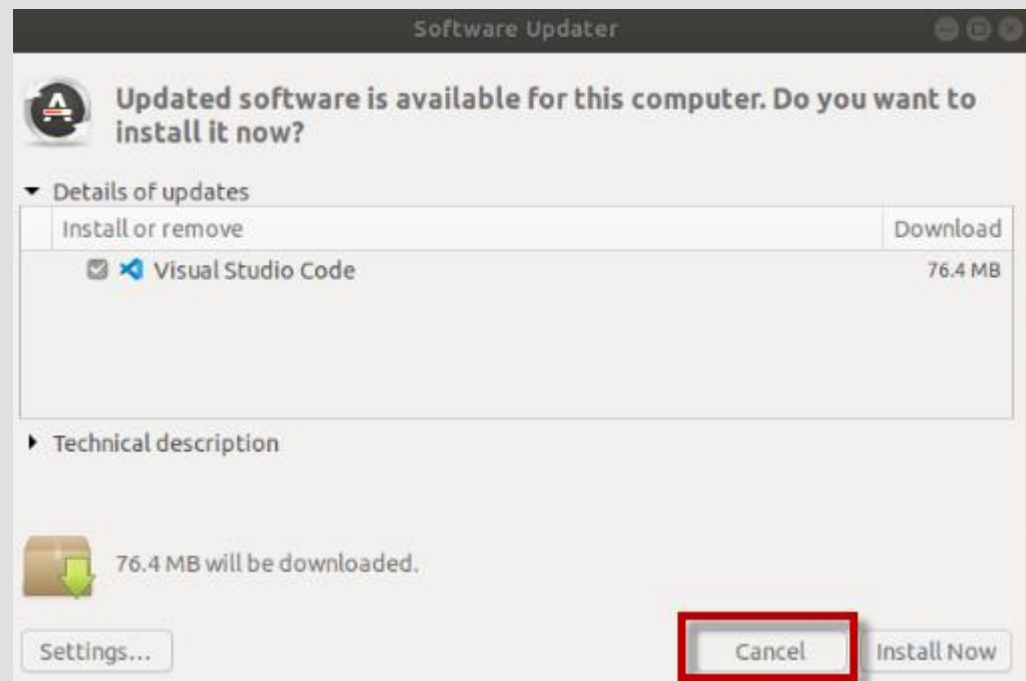


Important:

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 Liberty and Dev Mode

In this lab, you will learn how to run and update a simple REST microservice on an Open Liberty server using the developer mode (dev mode). You will use Maven throughout the guide to build and run the microservice as well as to interact with the running server instance.

1.6.1 Building and running the application using Maven and the liberty-maven-plugin

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 located 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. Build the “**system**” microservice that is provided and deploy it to Open Liberty by running the Maven **liberty:run** goal:

- ___1. Navigate to the project directory. Build the “**system**” microservice that is provided and deploy it to Open Liberty by running the Maven **liberty:run** goal:
 - a. Open a terminal window and change to the directory
`/home/ibmdemo/Student/labs/devmode/demo-project`

```
cd /home/ibmdemo/Student/labs/devmode/demo-project
```

- b. Run the mvn liberty:run command to start the Liberty server

```
mvn liberty:run
```

The mvn command initiates a **Maven build**, during which the target directory is created to store all build-related files.

The **liberty:run** argument specifies the Open Liberty run goal, which starts an Open Liberty server instance in the foreground.

As part of this phase, an Open Liberty server runtime is downloaded and installed into the target/liberty/wlp directory, a server instance is created and configured in the target/liberty/wlp/usr/servers/defaultServer directory, and the application is installed into that server.

When the server begins starting up, various messages display in your command-line session. Wait for the following message, which indicates that the server startup is complete:

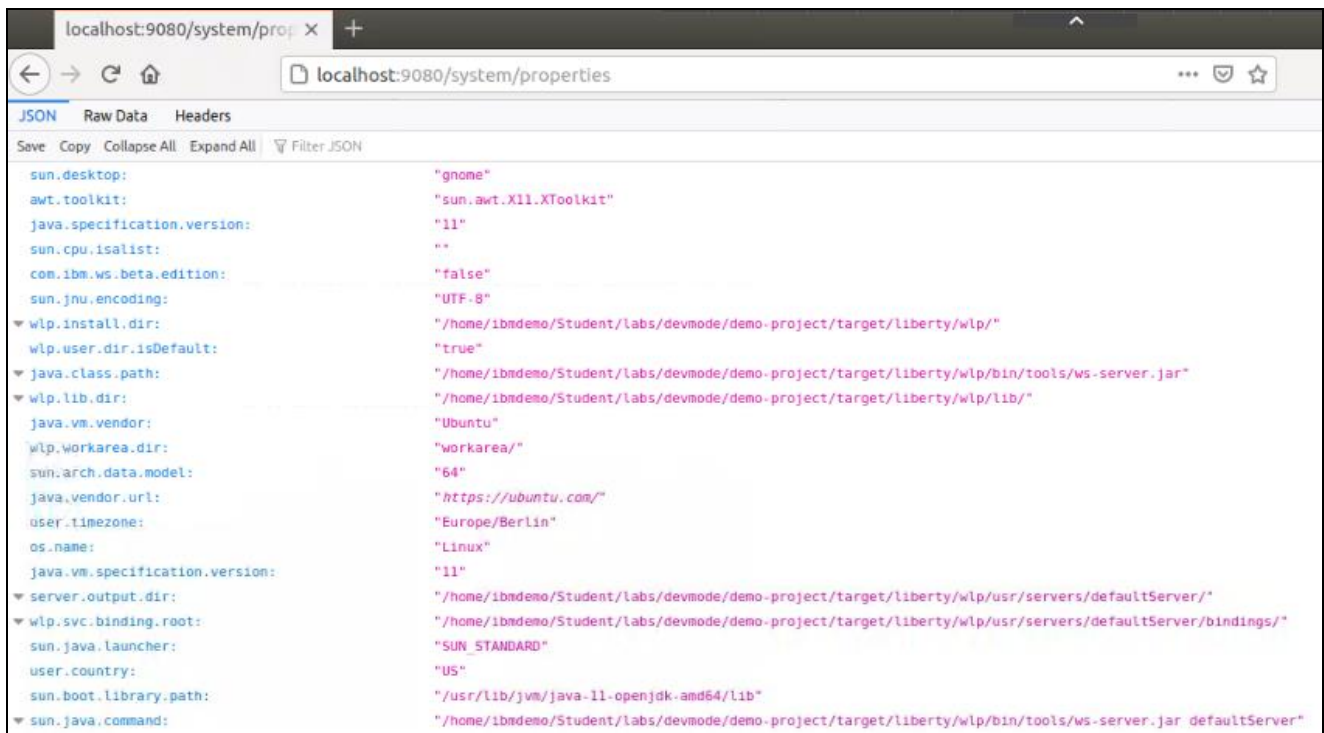
The server defaultServer is ready to run a smarter planet.

```
CNPKI0820A: The default keystore has been created using the 'keystore_password' environment variable.
CWWT0016I: Web application available (default_host): http://host-1.example.com:9080/ibm/api/
CWWT0016I: Web application available (default_host): http://host-1.example.com:9080/metrics/
CWWT0016I: Web application available (default_host): http://host-1.example.com:9080/
CWKZ0001I: Application guide-getting-started started in 1.359 seconds.
CWKF0012I: The server installed the following features: [cdi-2.0, distributedMap-1.0, jaxrs-2.1, jaxrsClient-2.1, jndi-1.0
1.1, monitor-1.0, mpConfig-1.4, mpMetrics-2.3, servlet-4.0, ssl-1.0].
CWKF0011I: The defaultServer server is ready to run a smarter planet. The defaultServer server started in 4.281 seconds.
```

__2. Access the “system” microservice that was deployed to the Liberty server.

- a. Open the Firefox Web Browser from inside of the VM and go to the URL below. The microservice lists various system properties of your JVM.

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



- ___3. **Stop** the Liberty Server by pressing the **CTRL+C** in the command-line session where you ran the server.
- ___4. Start and Stop the Liberty server in the background

Although you can start and stop the server in the foreground by using the Maven `liberty:run` goal, you can also start and stop the server in the background with the Maven **liberty:start** and **liberty:stop** goals:

```
mvn liberty:start
```

```
mvn liberty:stop
```

- ___5. View the **pom.xml** file to see the liberty-maven-plugin that was used by the previous steps.
 - a. From a Terminal window, navigate to the following directory

```
cd /home/ibmdemo/Student/labs/devmode/demo-project
```

- b. View the relevant plugin in the pom.xml file. The `-A` and `-B` options on the `grep` command display the specified number of lines before and after the location of the search text string.

```
cat pom.xml | grep -B 4 -A 2 liberty-maven-plugin
```

```
<plugins>
  <!-- Enable liberty-maven plugin -->
  <plugin>
    <groupId>io.openliberty.tools</groupId>
    <artifactId>liberty-maven-plugin</artifactId>
    <version>3.3.4</version>
  </plugin>
```



Information:

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

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

1.6.2 Updating the application without restarting the server

The Open Liberty Maven plug-in includes a **dev** goal that listens for any changes in the project, including application source code or configuration changes. The Open Liberty server automatically reloads the application and configuration without restarting. This goal allows for quicker turnarounds and an improved developer experience.

__1. Start the Liberty server in “dev” mode

- a. From a Terminal window, navigate to the following directory

```
cd /home/ibmdemo/Student/labs/devmode/demo-project
```

- b. Ensure the Liberty server is STOPPED!

```
mvn liberty:stop
```

- c. Start Liberty in dev mode

```
mvn liberty:dev
```

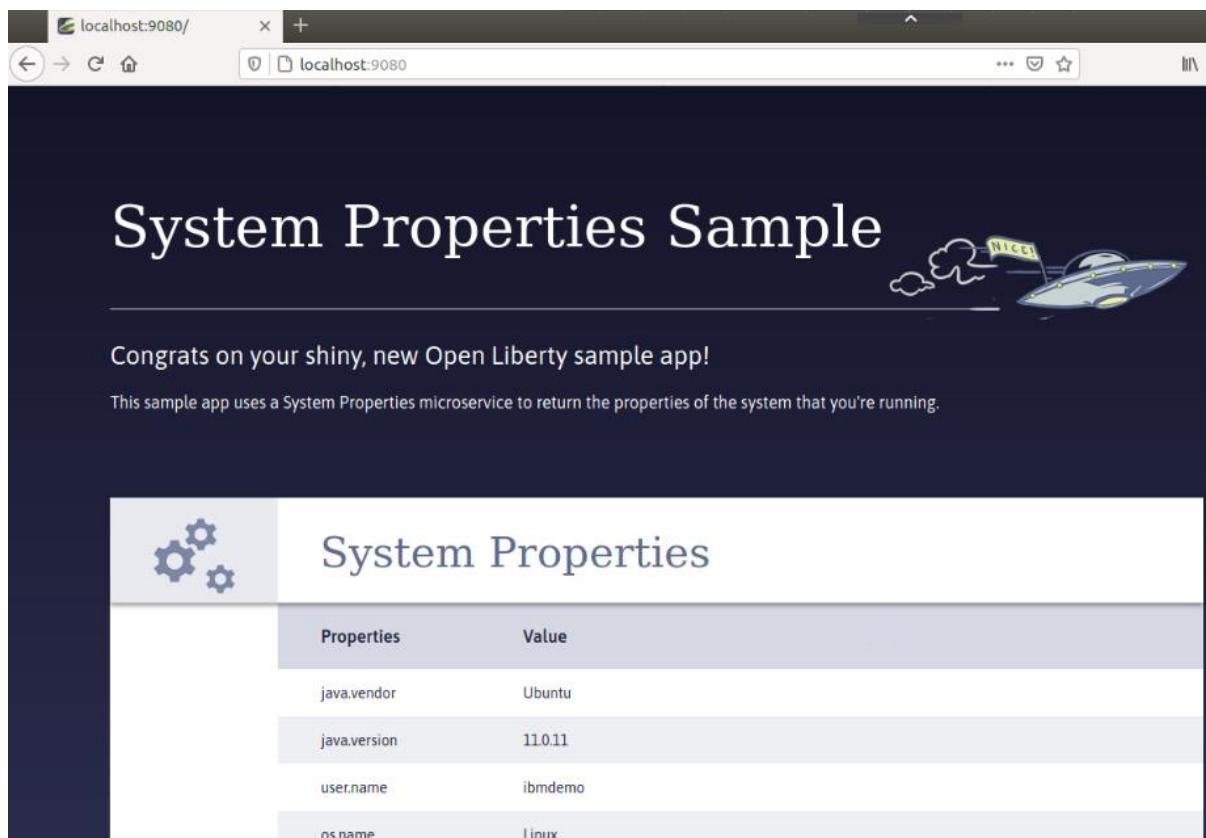
Note: The Liberty is now started in dev mode.

```
[INFO] [AUDIT] CWWK0020M: The default keystore has been created using the 'keystore_password' environment variable.
[INFO] [AUDIT] CWWKT0016I: Web application available (default_host): http://host-1.example.com:9080/metrics/
[INFO] [AUDIT] CWWKT0016I: Web application available (default_host): http://host-1.example.com:9080/ibm/api/
[INFO] [AUDIT] CWWKT0016I: Web application available (default_host): http://host-1.example.com:9080/
[INFO] [AUDIT] CWWKZ0001I: Application guide-getting-started started in 1.331 seconds.
[INFO] [AUDIT] CWWKF0012I: The server installed the following features: [cdi-2.0, distributedMap-1.0, jaxrs-2.1, jaxrsClient-2.1, jndi-1.0, json-1.0, jsonp-1.1, monitor-1.0, mpConfig-1.4, mpMetrics-2.3, servlet-4.0, ssl-1.0].
[INFO] [AUDIT] CWWKF0011I: The defaultServer server is ready to run a smarter planet. The defaultServer server started in 4.591 seconds.
[INFO] CWWKM2015I: Match number: 1 is [5/10/21, 19:30:46:358 CEST] 00000022 com.ibm.ws.kernel.feature.internal.FeatureManager A CW
[INFO] CWWKF0011I: The defaultServer server is ready to run a smarter planet. The defaultServer server started in 4.591 seconds..
[INFO] *****
[INFO] * Liberty is running in dev mode.
[INFO] * To run tests on demand, press Enter.
[INFO] * To restart the server, type 'r' and press Enter.
[INFO] * To stop the server and quit dev mode, press Ctrl-C or type 'q' and press Enter.
[INFO] *
[INFO] * Liberty server port information:
[INFO] * Liberty server HTTP port: [ 9080 ]
[INFO] * Liberty server HTTPS port: [ 9443 ]
[INFO] * Liberty debug port: [ 7777 ]
[INFO] *
[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 successfully.
[INFO] [AUDIT] CWWKT0016I: Web application available (default_host): http://host-1.example.com:9080/
[INFO] [AUDIT] CWWKZ0003I: The application guide-getting-started updated in 0.205 seconds.
```

Dev mode automatically picks up changes that you make to your application and allows you to run tests by pressing the **enter/return** key in the active command-line session. When you're working on your application, rather than rerunning Maven commands, press the enter/return key to verify your change, which executes your tests.

- __2. Access the "system" microservice that was deployed to the Liberty server.
 - a. Open the Firefox Web Browser from inside of the VM and go to the URL below to display the main application web page

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



- __3. Make a minor change to the "system Properties Sample" source code while running in dev mode, to see the changes dynamically picked up and applied to the running server.
 - a. Open a **new terminal window** and navigate to **src/main/webapp** folder of the application as illustrated below.


```
cd /home/ibmdemo/Student/labs/devmode/demo-  
project/src/main/webapp
```

- b. Use the gedit editor to open the **index.html** file in edit mode

```
gedit index.html
```

- c. Make the following minor change to the index.html page

Change the highlighted line:

```
<html>  
  <head>  
    <script src="js/mpData.js"></script>  
    <link href="https://fonts.googleapis.com/css?family=Asap" rel="stylesheet">  
    <link rel="stylesheet" href="css/main.css">  
  </head>  
  <body>  
    <section id="appIntro">  
      <div id="titleSection">  
        <h1 id="appTitle">System Properties Sample</h1>  
        <div class="line"></div>  
        <div class="headerImage"></div>  
        <h2>Congrats on your shiny, new Open Liberty sample app!</h2>  
        <p>This sample app uses a System Properties microservice to return th  
      </div>
```

Updated to read: System Properties Demo

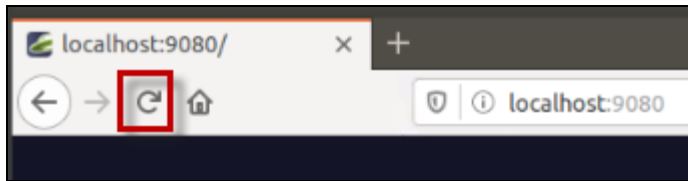
```
<html>  
  <head>  
    <script src="js/mpData.js"></script>  
    <link href="https://fonts.googleapis.com/css?family=Asap" rel="stylesheet">  
    <link rel="stylesheet" href="css/main.css">  
  </head>  
  <body>  
    <section id="appIntro">  
      <div id="titleSection">  
        <h1 id="appTitle">System Properties Demo</h1>  
        <div class="line"></div>  
        <div class="headerImage"></div>  
        <h2>Congrats on your shiny, new Open Liberty sample app!</h2>  
        <p>This sample app uses a System Properties microservice to return th  
      </div>
```

- d. **Save** the file and **close** the gedit editor.

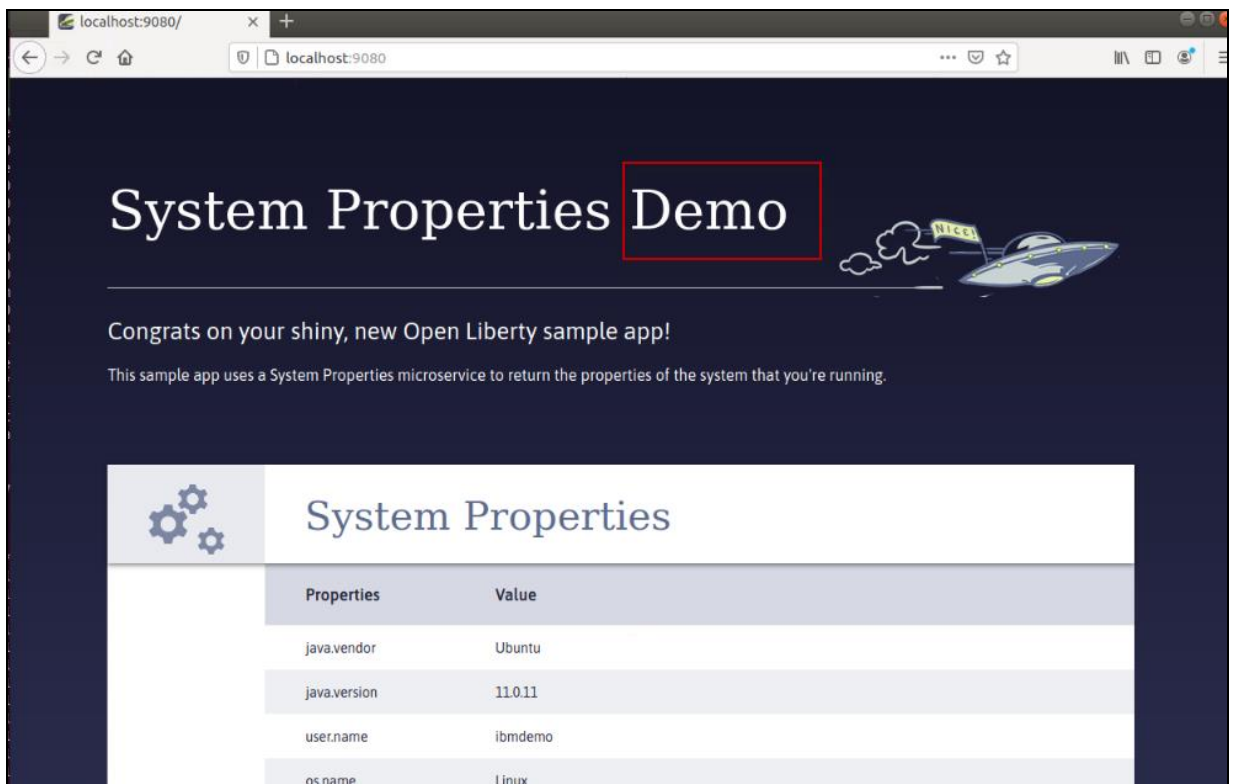
- ___4. Access the **System Properties Sample** microservice that was deployed to the Liberty server.
- From the Firefox Web Browser inside of the VM and go to the URL below to display the main application web page

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

- IMPORTANT:** Click the **RELOAD** icon in the Firefox browser to reload the page. Browsers cache content, so you need to reload the page.



- The updated index.html page is displayed with your changes dynamically picked up



The application code changes were detected and dynamically applied to the running instance of the Liberty server.

1.6.3 Updating the Server configuration without restarting the server

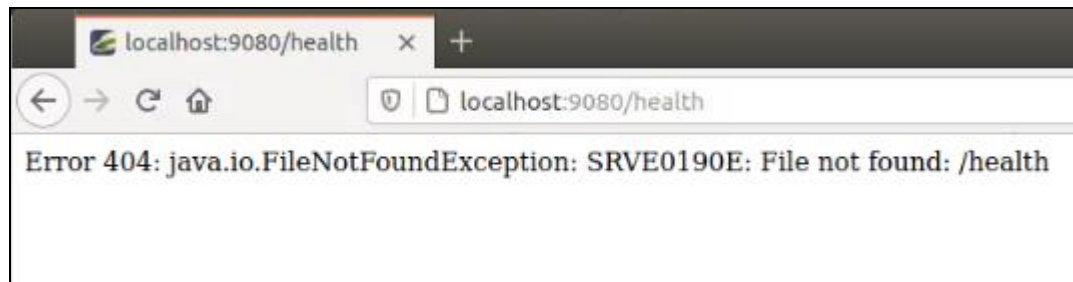
The Open Liberty Maven plug-in's **dev goal** not only listens for application code changes, but also **configuration changes** in the project. The Open Liberty server automatically reloads the configuration without restarting. This goal allows for quicker turnarounds and an improved developer experience.

In this section, you make a simple configuration change in the Liberty Server configuration file (server.xml) of the project (src), to include a **/health** endpoint for the service. You will notice that the changes are detected by maven and dynamically updated on the target server.xml. The running Liberty instance automatically picks up the changed target configuration

If you try to access this /health endpoint now, you see a 404 error because the **/health** endpoint does not yet exist:

- __1. From the Firefox browser in the VM, try to access the applications /health endpoint at:

<http://localhost:9080/health/>



- __2. Make a simple change to the Liberty Server configuration file to add **the mpHealth-2.2** feature to the server.xml file, which enables the health check endpoint.
 - a. From a terminal window, navigate to **liberty/config** folder of the application

```
cd /home/ibmdemo/Student/labs/devmode/demo-project/src/main/liberty/config
```

- b. Use the gedit editor to open the **server.xml** file in edit mode

```
gedit server.xml
```

- c. Add the Make the following minor change to the server.xml file

Change the highlighted line:

```
<server description="Sample Liberty server">
  <featureManager>
    <feature>jaxrs-2.1</feature>
    <feature>jsonp-1.1</feature>
    <feature>cdi-2.0</feature>
    <feature>mpMetrics-2.3</feature>
    <feature>mpConfig-1.4</feature>
  </featureManager>
```

Updated to include: `<feature>mpHealth-2.2</feature>`

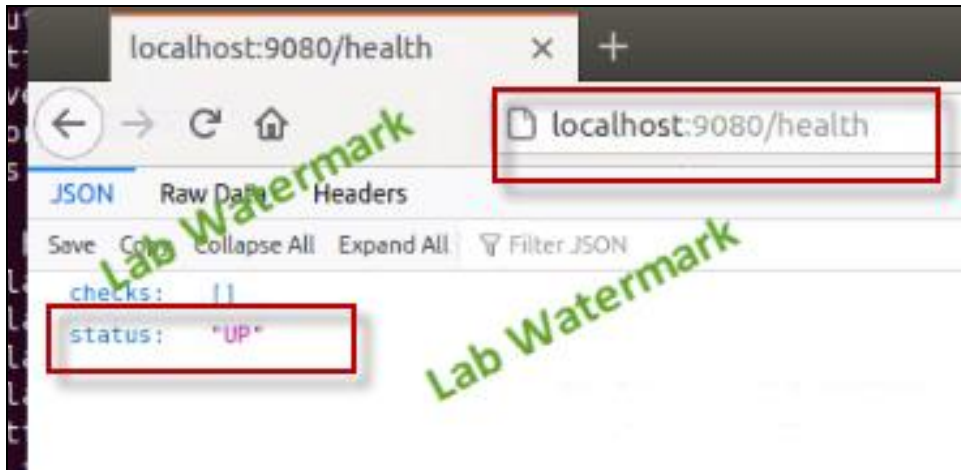
```
<server description="Sample Liberty server">
  <featureManager>
    <feature>jaxrs-2.1</feature>
    <feature>jsonp-1.1</feature>
    <feature>cdi-2.0</feature>
    <feature>mpMetrics-2.3</feature>
    <feature>mpConfig-1.4</feature>
    <feature>mpHealth-2.2</feature>
  </featureManager>
```

- d. **Save** the file and **close** the gedit editor.

__3. Access the new **Health Endpoint** for the application.

- a. Open the Firefox Web Browser from inside of the VM and go to the URL below to display the health endpoint.

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



- __4. View the Liberty server console log to see the **mpHealth-2.2** feature was installed, and the health endpoint enabled.
 - a. Return to the Terminal window where the **mvn liberty:dev** is running.
 - b. Review the messages that indicate the **mpHealth-22** feature was dynamically installed and the **/health** endpoint enabled.

```
ibmdemo@ibmdemo-virtual-machine: ~/Student/labs/devmode/demo-project
File Edit View Search Terminal Help
~/Student/labs/devmode/demo-project/target/liberty/wlp/usr/servers/defaultServer/server.xml
[INFO] Running liberty:deploy
[INFO] CWWM2102I: Using artifact based assembly archive : io.openliberty:openliberty-kernel:null:21.0.0.4:zip.
[INFO] CWWM2102I: Using installDirectory : /home/ibmdemo/Student/labs/devmode/demo-project/target/liberty/wlp.
[INFO] CWWM2102I: Using serverName : defaultServer.
[INFO] CWWM2102I: Using serverDirectory : /home/ibmdemo/Student/labs/devmode/demo-project/target/liberty/wlp/usr/servers/defaultServer.
[INFO] Copying 1 file to /home/ibmdemo/Student/labs/devmode/demo-project/target/liberty/wlp/usr/servers/defaultServer
[INFO] CWWM2144I: Update server configuration file server.xml from /home/ibmdemo/Student/labs/devmode/demo-project/src/main/liberty/config/server.xml.
[INFO] CWWM2185I: The liberty-maven-plugin configuration parameter "appsDirectory" value defaults to "apps".
[INFO] CWWM2160I: Installing application guide-getting-started.war.xml.
[INFO] CWWM2010I: Searching for CWWKZ0001I.*guide-getting-started in /home/ibmdemo/Student/labs/devmode/demo-project/target/liberty/wlp/usr/servers/defaultServer/logs/messages.log. This search will timeout after 40 seconds.
[INFO] CWWM2015I: Match number: 1 is [5/11/21, 17:29:48:085 CEST] 00000023 com.ibm.ws.app.manager.AppMessageHelper
A CWWKZ0001I: Application guide-getting-started started in 1.481 seconds..
[INFO] [AUDIT] CWWK0017I: Web application removed (default_host): http://host-1.example.com:9080/
[INFO] [AUDIT] CWWKZ0009I: The application guide-getting-started has stopped successfully.
[INFO] [AUDIT] CWWK0016I: Starting server configuration update.
[INFO] [AUDIT] CWWK0093A: Processing configuration drop-ins resource: /home/ibmdemo/Student/labs/devmode/demo-project/target/liberty/wlp/usr/servers/defaultServer/configDropins/overrides/liberty-plugin-variable-config.xml
[INFO] [AUDIT] CWWK0016I: Web application available (default host): http://host-1.example.com:9080/
[INFO] [AUDIT] CWWKZ0003I: The application guide-getting-started updated in 0.210 seconds.
[INFO] [AUDIT] CWWK0017I: Web application removed (default_host): http://host-1.example.com:9080/
[INFO] [AUDIT] CWWKZ0009I: The application guide-getting-started has stopped successfully.
[INFO] [AUDIT] CWWK0017I: The server configuration was successfully updated in 0.346 seconds.
[INFO] [AUDIT] CWWK0016I: Web application available (default host): http://host-1.example.com:9080/health/
[INFO] [AUDIT] CWWK0012I: The server installed the following features: [mpHealth-2.2].
[INFO] [AUDIT] CWWK0000I: Feature update completed in 0.357 seconds.
[INFO] [AUDIT] CWWK0016I: Web application available (default host): http://host-1.example.com:9080/
[INFO] [AUDIT] CWWKZ0003I: The application guide-getting-started updated in 0.170 seconds.
```

The Liberty server configuration changes were detected and dynamically applied to the running instance of the Liberty server.

The **/health** endpoint reports whether the server is running, but the endpoint doesn't provide any details on the microservices that are running inside of the server.

MicroProfile Health offers health checks for both readiness and liveness.

- A **readiness** check allows third-party services, such as Kubernetes, to know if the microservice is ready to process requests.
- A **liveness** check allows third-party services to determine if the microservice is running.

Note: Working with MicroProfile Health is beyond the scope of this lab and is introduced in a subsequent lab.

1.7 Developing and Running the application in a Docker Container and in Liberty Dev Mode

When developing an application that will eventually be deployed to production in **containers**, you can avoid potential issues by ensuring that your development and production environments are as similar as possible. This aligns with [Twelve Factor App](#) methodology, particularly factor 10, which calls for dev/prod parity. For cloud-native applications, part of this issue is addressed by using containers where your environment can be codified to provide consistency between development and production.

In this section of the lab, you will use **Liberty dev mode with containers**. With container support, you can develop applications on your local environment while your Open Liberty server runs in a container.

The development container image is kept as similar as possible to the production container image, while still allowing for iterative development. Your code changes are automatically hot deployed to the container and picked up by the running server. Additionally, dev mode allows you run tests either automatically or on demand, and you can attach a debugger at any time to debug your application.

With container support for Open Liberty dev mode, you can use the same **Dockerfile** for both development and production. You use the same base image and customizations, and you specify the exact configuration files that you need for your application in your Dockerfile. This prevents you from running into any surprises when you deploy your application to production.

Dev mode changes how the image is built and run to enable iterative development, so the images are identical except for how application and configuration files are mounted into the containers. With dev mode, simply save a source file in any text editor or IDE, and it is recompiled and picked up without any need to rebuild the image or restart the server.

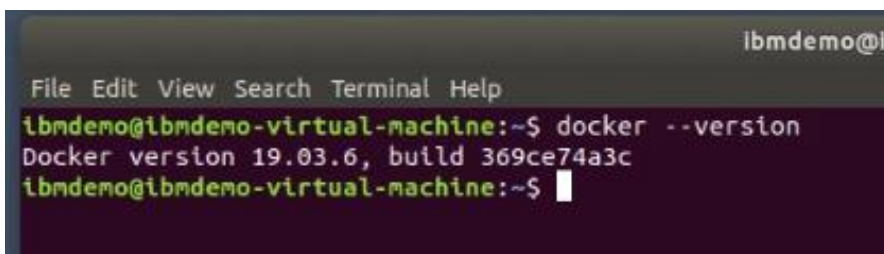
1.7.1 Investigate Docker Commands and Dockerfile for building images

This section is a primer on how to use, build, deploy and run Liberty in a Docker container. You will pull Liberty from Docker hub, install it, review information about the container, access resources inside the container, add an application then test it.

To run the application in a container, Docker needs to be installed and the Docker daemon running. In this lab environment, these prerequisites have been configured.

- __1. Open a Terminal window and verify that Docker is running

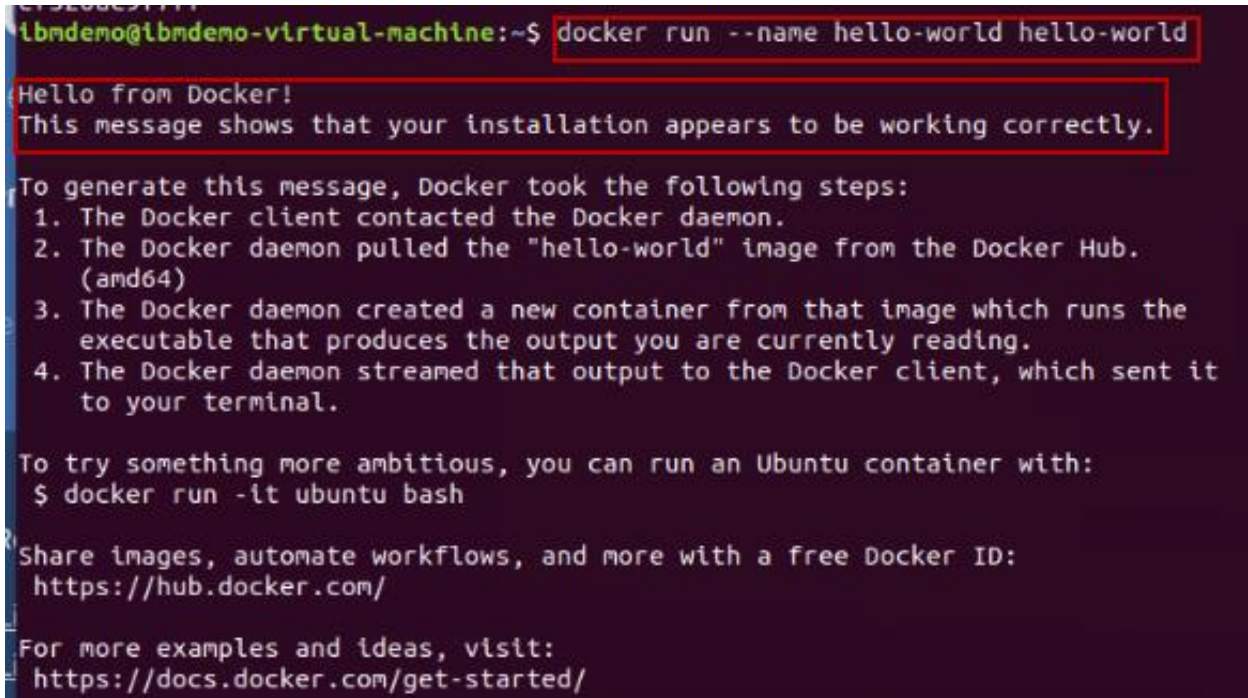
```
docker --version
```



```
ibmdemo@ibmdemo-virtual-machine:~$ docker --version
Docker version 19.03.6, build 369ce74a3c
ibmdemo@ibmdemo-virtual-machine:~$
```


- __2. Run the hello-world docker sample. The docker image will be pulled from Dockerhub if it is not already on the local machine. If Docker is functioning properly, you will see the highlighted message illustrated below, stating that Docker is working properly.

```
docker run --name hello-world hello-world
```

A terminal window with a dark background and light green text. The prompt is 'ibmdemo@ibmdemo-virtual-machine:~\$'. The command 'docker run --name hello-world hello-world' is entered and highlighted with a red box. The output is: 'Hello from Docker! This message shows that your installation appears to be working correctly.' This output is also highlighted with a red box. Below this, a list of four steps explains how Docker generated the message. At the bottom, there are links to Docker Hub and Docker documentation.

```
ibmdemo@ibmdemo-virtual-machine:~$ docker run --name hello-world hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

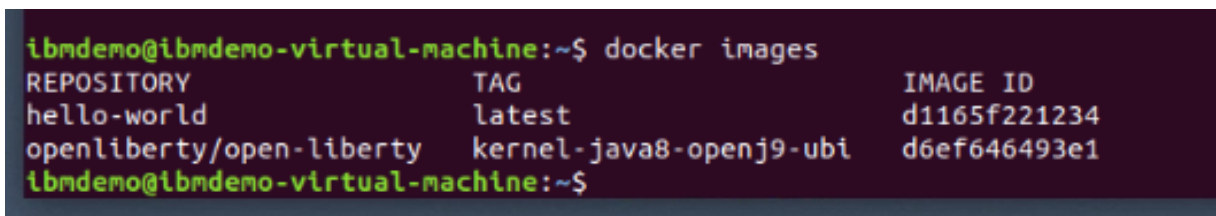
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

- __3. Use the **docker images** command to find out which Docker images are in the local docker repository

```
docker images
```

A terminal window with a dark background and light green text. The prompt is 'ibmdemo@ibmdemo-virtual-machine:~\$'. The command 'docker images' is entered. The output is a table with three columns: REPOSITORY, TAG, and IMAGE ID.

```
ibmdemo@ibmdemo-virtual-machine:~$ docker images
REPOSITORY          TAG                 IMAGE ID
hello-world          latest              d1165f221234
openliberty/open-liberty  kernel-java8-openj9-ubi  d6ef646493e1
ibmdemo@ibmdemo-virtual-machine:~$
```

- __4. Use the docker history command to view the layers that make up the Docker image

```
docker history hello-world
```

```
ibmdemo@ibmdemo-virtual-machine: ~  
File Edit View Search Terminal Help  
ibmdemo@ibmdemo-virtual-machine:~$ docker history hello-world  
IMAGE          CREATED          CREATED BY          SIZE  
d1165f221234    2 months ago    /bin/sh -c #(nop)  CMD ["/hello"]    0B  
<missing>       2 months ago    /bin/sh -c #(nop)  COPY file:7bf12aab75c3867a... 13.3kB  
ibmdemo@ibmdemo-virtual-machine:~$
```

- __5. Let's run a Liberty image Docker container. Docker will check if there is an image in the repository. If not, it will download the latest image, then run it

```
docker run -d -p 9086:9080 --name wlp websphere-liberty
```

```
ibmdemo@ibmdemo-virtual-machine:~$ docker run -d -p 9086:9080 --name wlp websphere-liberty  
Unable to find image 'websphere-liberty:latest' locally  
latest: Pulling from library/websphere-liberty  
01bf7da0a88c: Pull complete  
f3b4a5f15c7a: Pull complete  
57ffbe87baa1: Pull complete  
3efb27499fe2: Pull complete  
7050a00d84ec: Pull complete  
83b032ab6a73: Pull complete  
f51afc71ced3: Pull complete  
177ddd3816a7: Pull complete  
a563485cf947: Pull complete  
6dec3091c20b: Pull complete  
0a2a8dd613d6: Pull complete  
6c6e8c0abd5b: Pull complete  
2d740348188a: Pull complete  
787a0fc3436a: Pull complete  
Digest: sha256:7c7074d5d875389a350b96c1e69ad7e718c95168ca22ea8eb293b4569dac5fc3  
Status: Downloaded newer image for websphere-liberty:latest  
e5a23342880b89c70d719a3fe05afa1dcb0460e2e4a0721af2dc35a1c364bbc0  
ibmdemo@ibmdemo-virtual-machine:~$
```

__6. Review the containers information.

- a. The **docker ps** command lists only **running** containers. The **docker ps -a** command shows all containers, running or stopped.

```
docker ps
```

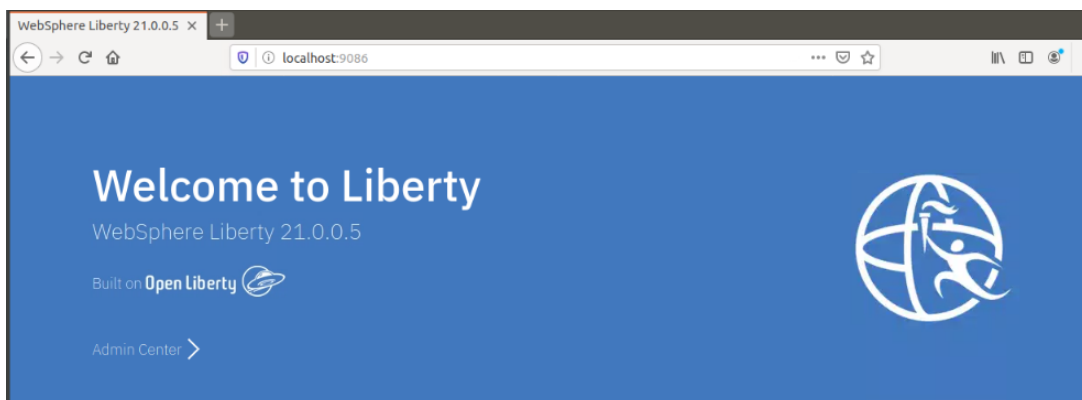
```
ibmdemo@ibmdemo-virtual-machine:~$ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED
e5a23342880b   websphere-liberty "/opt/ibm/helpers/ru...  7 minutes ago
tcp, 9443/tcp   wlp
```

- b. The **docker stats** command shows resource usage of the running containers. Use the **Ctrl + C** keys to stop stats

```
docker stats
```

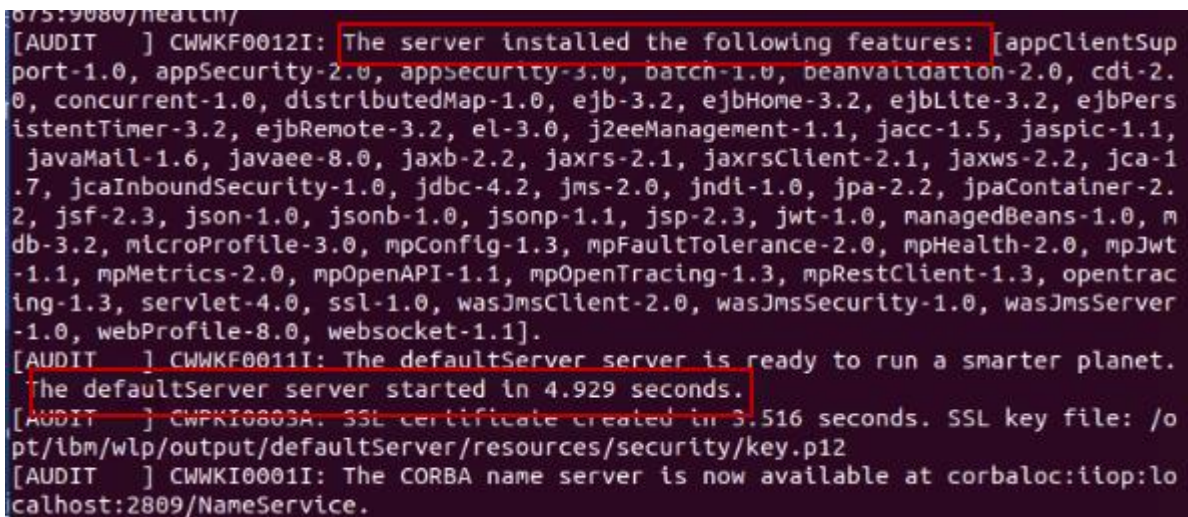
```
ibmdemo@ibmdemo-virtual-machine: ~
File Edit View Search Terminal Help
CONTAINER ID   NAME             CPU %             MEM USAGE / LIMIT  MEM %
NET I/O       BLOCK I/O        PIDS
e5a23342880b   wlp              0.40%            228.3MiB / 3.844GiB  5.80%
4.95kB / 286B  0B / 13.1MB      70
```

__7. Open the Firefox browser on the VM and access Liberty running in the container:
<http://localhost:9086>



- __8. Look at the Liberty logs in the running container

```
docker logs -f wlp
```



The screenshot shows the output of the command `docker logs -f wlp`. The logs are from the Open Liberty server. Key messages include: [AUDIT] CWWKF0012I: The server installed the following features: [appClientSupport-1.0, appSecurity-2.0, appSecurity-3.0, batch-1.0, beanValidation-2.0, cdi-2.0, concurrent-1.0, distributedMap-1.0, ejb-3.2, ejbHome-3.2, ejbLite-3.2, ejbPersistentTimer-3.2, ejbRemote-3.2, el-3.0, j2eeManagement-1.1, jacc-1.5, jaspic-1.1, javaMail-1.6, javaee-8.0, jaxb-2.2, jaxrs-2.1, jaxrsClient-2.1, jaxws-2.2, jca-1.7, jcaInboundSecurity-1.0, jdbc-4.2, jms-2.0, jndi-1.0, jpa-2.2, jpaContainer-2.2, jsf-2.3, json-1.0, jsonb-1.0, jsonp-1.1, jsp-2.3, jwt-1.0, managedBeans-1.0, mdb-3.2, microProfile-3.0, mpConfig-1.3, mpFaultTolerance-2.0, mpHealth-2.0, mpJwt-1.1, mpMetrics-2.0, mpOpenAPI-1.1, mpOpenTracing-1.3, mpRestClient-1.3, opentracing-1.3, servlet-4.0, ssl-1.0, wasJmsClient-2.0, wasJmsSecurity-1.0, wasJmsServer-1.0, webProfile-8.0, websocket-1.1]. [AUDIT] CWWKF0011I: The defaultServer server is ready to run a smarter planet. The defaultServer server started in 4.929 seconds. [AUDIT] CWPXI0803A: SSL certificate created in 3.516 seconds. SSL key file: /opt/ibm/wlp/output/defaultServer/resources/security/key.p12 [AUDIT] CWWKI0001I: The CORBA name server is now available at corbaloc:iiop:localhost:2809/NameService.

- __9. **Stop** and **remove** the docker containers used in this section of the lab. Then use the **docker ps -a** command to verify the containers are removed.

```
docker stop wlp  
  
docker rm wlp  
  
docker rm hello-world  
  
docker ps -a
```

1.7.2 Running the application in a container

To run the application in a container, Docker needs to be installed and the Docker daemon running. In this lab environment, these prerequisites have been configured.

First, to containerize the application, you need a **Dockerfile**. This file contains a collection of instructions that define how a Docker image is built, what files are packaged into it, what commands run when the image runs as a container, and other information.

For this lab, a **Dockerfile** has been provided to build the docker image for the System Properties Sample. This Dockerfile copies the **.war** file into a Docker image that contains the Java runtime and a preconfigured Open Liberty server.

- __1. From a Terminal window, STOP the running Liberty Server from the previous section of the lab, using the commands below:

```
cd /home/ibmdemo/Student/labs/devmode/demo-project  
mvn liberty:stop
```

- __2. View the Dockerfile that is used to build the docker image.

- a. Open a terminal window and change to the directory:
/home/ibmdemo/Student/labs/devmode/demo-project

```
cd /home/ibmdemo/Student/labs/devmode/demo-project
```

- b. Investigate the Dockerfile

```
cat Dockerfile
```

The Dockerfile performs the following tasks:

```
FROM openliberty/open-liberty:kernel-java8-openj9-ubi  
  
ARG VERSION=1.0  
ARG REVISION=SNAPSHOT  
  
LABEL \  
  org.opencontainers.image.authors="Your Name" \  
  org.opencontainers.image.vendor="IBM" \  
  org.opencontainers.image.url="local" \  
  org.opencontainers.image.source="https://github.com/OpenLiberty/guide-getting-started" \  
  org.opencontainers.image.version="$VERSION" \  
  org.opencontainers.image.revision="$REVISION" \  
  vendor="Open Liberty" \  
  name="system" \  
  version="$VERSION-$REVISION" \  
  summary="The system microservice from the Getting Started guide" \  
  description="This image contains the system microservice running with the Open Liberty runtime."  
  
COPY --chown=1001:0 src/main/liberty/config/ /config/  
COPY --chown=1001:0 target/*.war /config/apps/  
  
RUN configure.sh
```

Read below for a short description and details of the commands in the Dockerfile:

- FROM [openliberty/open-liberty:kernel-java8-openj9-ubi](#)

FROM specifies the Docker image that is to be used. If this is not in the local repository, this will be pulled from Docker Hub.

In this example, we pull the liberty kernel image that contains Java 8, and uses the RedHat Universal base Images for deployments to OpenShift.

The **kernel** image contains just the Liberty kernel and no additional runtime features. This image is the recommended basis for custom built images, so that they can contain only the features required for a specific application

- ARG [VERSION=1.0](#) and ARG [REVISION=SNAPSHOT](#)

The ARG instruction defines variables that can be passed at build time. Once it is defined in the Dockerfile, you can pass it with the flag **--build-arg**.

- LABEL

Labels are used in Dockerfile to help organize your Docker Images. Labels are key-value pairs and simply add custom metadata to your Docker Images.

- COPY [--chown=1001:0 src/main/liberty/config/ /config/](#)

The COPY command copies the liberty configuration file (server.xml) to the /config folder in the Liberty container.

The RedHat Universal Base Images (UBI) are built such that Liberty does not run as root. To ensure that Liberty can access the files that are copied to the image, they must be owned by a non-root user. All Liberty images from IBM contain a non-root user defined as **1001:0**. So the copy command copies the files as the non-root user that is known to exist on the LIBERTY image.

- COPY [--chown=1001:0 target/*.war /config/apps/](#)

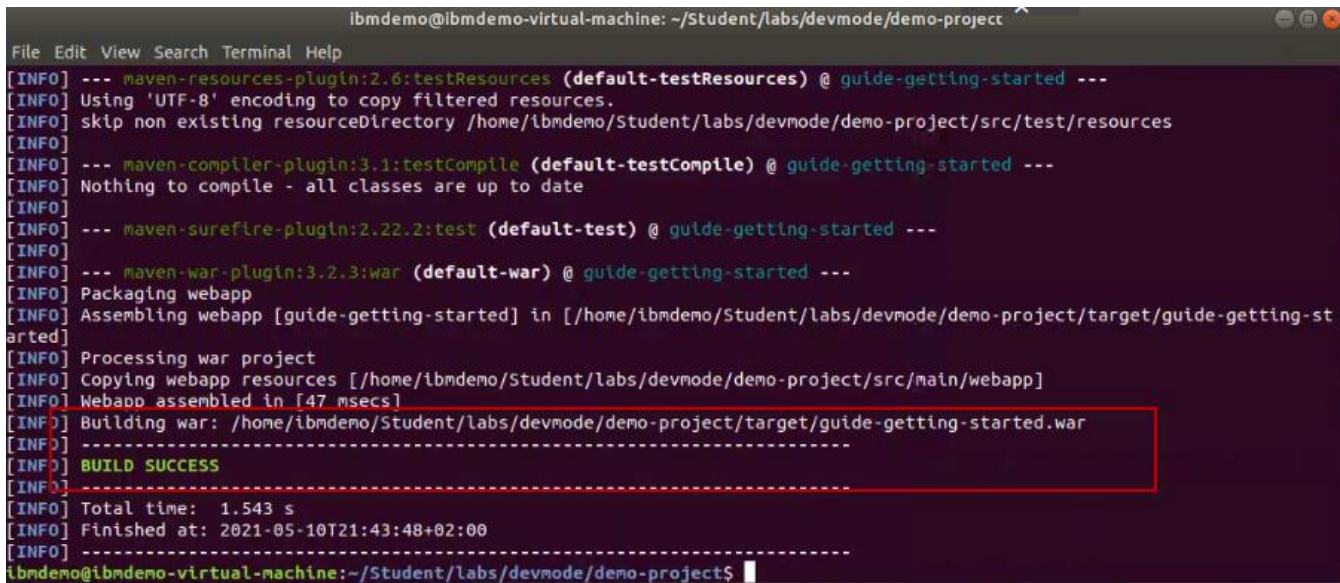
The COPY command copies the application WAR file to the /config/apps directory on the image.

- RUN [configure.sh](#)

The configure.sh script installs the required Liberty features based on the features defined in the server configuration file (server.xml). This script is baked into ALL Liberty and tWAS Docker images provided by IBM.

- ___3. Run the **mvn package** command from the **demo-project** directory. The command will produce a war file named “**guide-getting-started.war**” and copied to the “**target**” directory.

```
mvn package
```

A terminal window titled 'ibmdemo@ibmdemo-virtual-machine: ~/Student/labs/devmode/demo-project' showing the output of the 'mvn package' command. The output includes several informational messages from Maven plugins: maven-resources-plugin, maven-compiler-plugin, maven-surefire-plugin, and maven-war-plugin. The final message is 'BUILD SUCCESS' in green, followed by 'Total time: 1.543 s' and 'Finished at: 2021-05-10T21:43:48+02:00'. A red box highlights the 'BUILD SUCCESS' message and the path to the war file: '/home/ibmdemo/Student/labs/devmode/demo-project/target/guide-getting-started.war'.

```
ibmdemo@ibmdemo-virtual-machine: ~/Student/labs/devmode/demo-project
File Edit View Search Terminal Help
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ guide-getting-started ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/ibmdemo/Student/labs/devmode/demo-project/src/test/resources
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ guide-getting-started ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ guide-getting-started ---
[INFO] --- maven-war-plugin:3.2.3:war (default-war) @ guide-getting-started ---
[INFO] Packaging webapp
[INFO] Assembling webapp [guide-getting-started] in [/home/ibmdemo/Student/labs/devmode/demo-project/target/guide-getting-started]
[INFO] Processing war project
[INFO] Copying webapp resources [/home/ibmdemo/Student/labs/devmode/demo-project/src/main/webapp]
[INFO] Webapp assembled in [47 msecs]
[INFO] Building war: /home/ibmdemo/Student/labs/devmode/demo-project/target/guide-getting-started.war
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.543 s
[INFO] Finished at: 2021-05-10T21:43:48+02:00
[INFO] -----
ibmdemo@ibmdemo-virtual-machine:~/Student/labs/devmode/demo-project$
```

The Maven package command builds the application and produces an application deployable binary “.war” file that is copied into the Docker image via the Dockerfile.

- ___4. Run the following command to download or update to the latest Open Liberty Docker image used in our Dockerfile:

```
docker pull openliberty/open-liberty:kernel-java8-openj9-ubi
```

Note: The image has already been pulled onto this VM. So, you may only see a message indicating the image is already up to date, rather than pulling the image again.

```

kernel-java8-openj9-ubi: Pulling from openliberty/open-liberty
ec1681b6a383: Pull complete
c4d668e229cd: Pull complete
ab5c4f4bffe6: Pull complete
5fb7bfe3044f: Pull complete
371fe28dd907: Pull complete
8844ac3b8359: Pull complete
f44d037b7eb6: Pull complete
f87b15f2bf73: Pull complete
64dfa9407640: Pull complete
d8b93da6f830: Pull complete
5d5b852ec707: Pull complete
7205b542686c: Pull complete
d16ad2a56062: Pull complete
Digest: sha256:5446158e6a252ee0c069eee8e0aa3af694e41106e30047018eb3addfc5c6f0a9
Status: Downloaded newer image for openliberty/open-liberty:kernel-java8-openj9-ubi
docker.io/openliberty/open-liberty:kernel-java8-openj9-ubi
ibmdemo@ibmdemo-virtual-machine:~/Student/labs/devmode/demo-project$

```

OR, you might see this message if the latest image is already pulled.

```

[INFO] -----
ibmdemo@ibmdemo-virtual-machine:~/Student/labs/devmode/demo-project$ docker pull openliberty/open-liberty:kernel-java8-openj
9-ubi
kernel-java8-openj9-ubi: Pulling from openliberty/open-liberty
Digest: sha256:5446158e6a252ee0c069eee8e0aa3af694e41106e30047018eb3addfc5c6f0a9
Status: Image is up to date for openliberty/open-liberty:kernel-java8-openj9-ubi
docker.io/openliberty/open-liberty:kernel-java8-openj9-ubi
ibmdemo@ibmdemo-virtual-machine:~/Student/labs/devmode/demo-project$

```

- ___5. To build and containerize the application, run the following Docker build command. Be sure you are in the directory that contains the Docker file.

Note: The **dot** at the end of the docker build command is part of the command, indicating to use the current directory path.

```
cd /home/ibmdemo/Student/labs/devmode/demo-project
```

```
docker build -t openliberty-getting-started:1.0-SNAPSHOT .
```



```
lbmdemo@lbmdemo-virtual-machine: ~/Student/labs/devmode/demo-project
File Edit View Search Terminal Help
---> 7108113d84df
Step 3/7 : ARG REVISION=SNAPSHOT
---> Running in 7aefcad83143
Removing intermediate container 7aefcad83143
---> ba8c6479ccde
Step 4/7 : LABEL org.opencontainers.image.authors="Your Name" org.opencontainers.image.vendor="IBM" org
.image.url="local" org.opencontainers.image.source="https://github.com/OpenLiberty/guide-getting-started"
ners.image.version="$VERSION" org.opencontainers.image.revision="$REVISION" vendor="Open Liberty" name=
ion="$VERSION-$REVISION" summary="The system microservice from the Getting Started guide" description="Th
ns the system microservice running with the Open Liberty runtime."
---> Running in 5fc8196e5f70
Removing intermediate container 5fc8196e5f70
---> 735bdafec1fd
Step 5/7 : COPY --chown=1001:0 src/main/liberty/config/ /config/
---> bd5cfec64dba
Step 6/7 : COPY --chown=1001:0 target/*.war /config/apps/
---> 78b7c415316c
Step 7/7 : RUN configure.sh
---> Running in 4668e07cdf47
Removing intermediate container 4668e07cdf47
---> 8d5fe80d4a10
Successfully built 8d5fe80d4a10
Successfully tagged openliberty-getting-started:1.0-SNAPSHOT
lbmdemo@lbmdemo-virtual-machine:~/Student/labs/devmode/demo-project$
```

The Docker **openliberty-getting-started:1.0-SNAPSHOT** image is built from the **Dockerfile**.

- __6. To verify that the image is built, run the **docker images** command to list all local Docker images

```
docker images
```

Your image should be listed as “**openliberty-getting-started**” with the TAG of “**1.0-SNAPSHOT**”

```
Successfully tagged openliberty-getting-started:1.0-SNAPSHOT
lbmdemo@lbmdemo-virtual-machine:~/Student/labs/devmode/demo-project$ docker images
REPOSITORY              TAG                IMAGE ID           CREATED
openliberty-getting-started 1.0-SNAPSHOT      8d5fe80d4a10      3 minutes ago
openliberty/open-liberty  kernel-java8-openj9-ubi d0ef646493e1      6 months ago
lbmdemo@lbmdemo-virtual-machine:~/Student/labs/devmode/demo-project$
```

- __7. Next, run the container from the image.

```
docker run -d --name gettingstarted-app -p 9080:9080
openliberty-getting-started:1.0-SNAPSHOT
```

```
lbmdemo@lbmdemo-virtual-machine:~/Student/labs/devmode/demo-project$ docker run -d --name gettingstarted-app -p 9080:9080
openliberty-getting-started:1.0-SNAPSHOT
58a923209dd98fd8bf157ad63f74431930fd0932af808feb7a5de641fc0772c7
lbmdemo@lbmdemo-virtual-machine:~/Student/labs/devmode/demo-project$
```

There is a bit going on in that command; so, here is a short description of the parameters on the command. And the final argument on the command is the Docker image name.

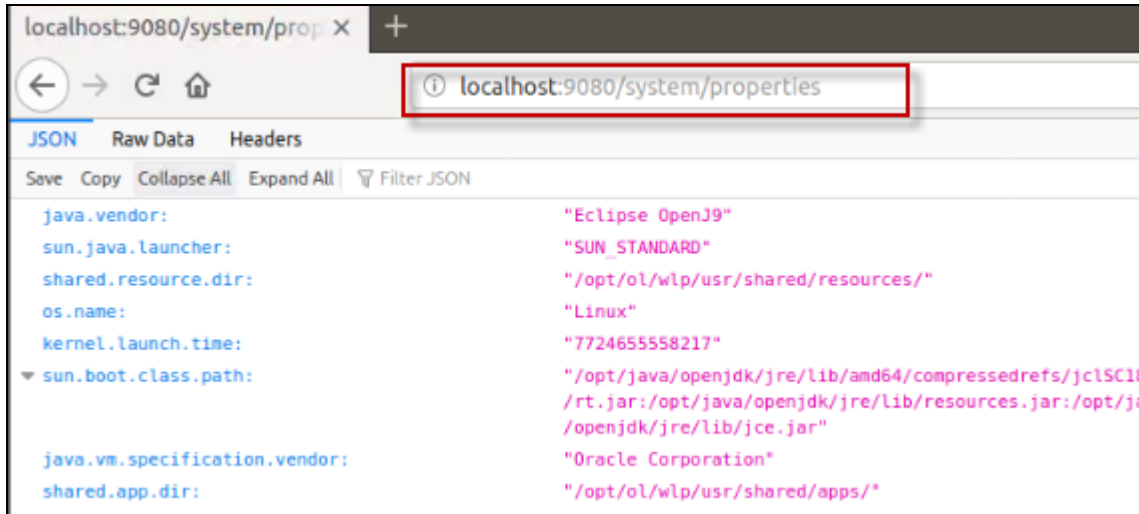
Flag	Description
-d	Runs the container in the background.
--name	Specifies a name for the container.
-p	Maps the container ports to the host ports.

- __8. Run the **docker ps** command to verify your docker container ins running

```
docker ps
```

```
58a923209dd910801157a003174431930700932a18081e07a50e041fc0772c7
lbdemo@lbdemo-virtual-machine:~/Student/labs/devnode/demo-project$ docker ps
CONTAINER ID        IMAGE                                     COMMAND                  CREATED
PORTS              NAMES
58a923209dd9       openliberty-getting-started:1.0-SNAPSHOT  "/opt/ol/helpers/run..." 2 minutes ago
0.0.0.0:9080->9080/tcp, 9443/tcp  gettingstarted-app
```

- __9. From a Web Browser, access the application using: <http://localhost:9080/system/properties>



- __10. Stop and Remove the Docker container

```
docker stop gettingstarted-app
docker rm gettingstarted-app
```

- __11. Remove the Docker image

```
docker rmi openliberty-getting-started:1.0-SNAPSHOT
```


1.7.3 Using Dev mode to develop an application in a Docker container

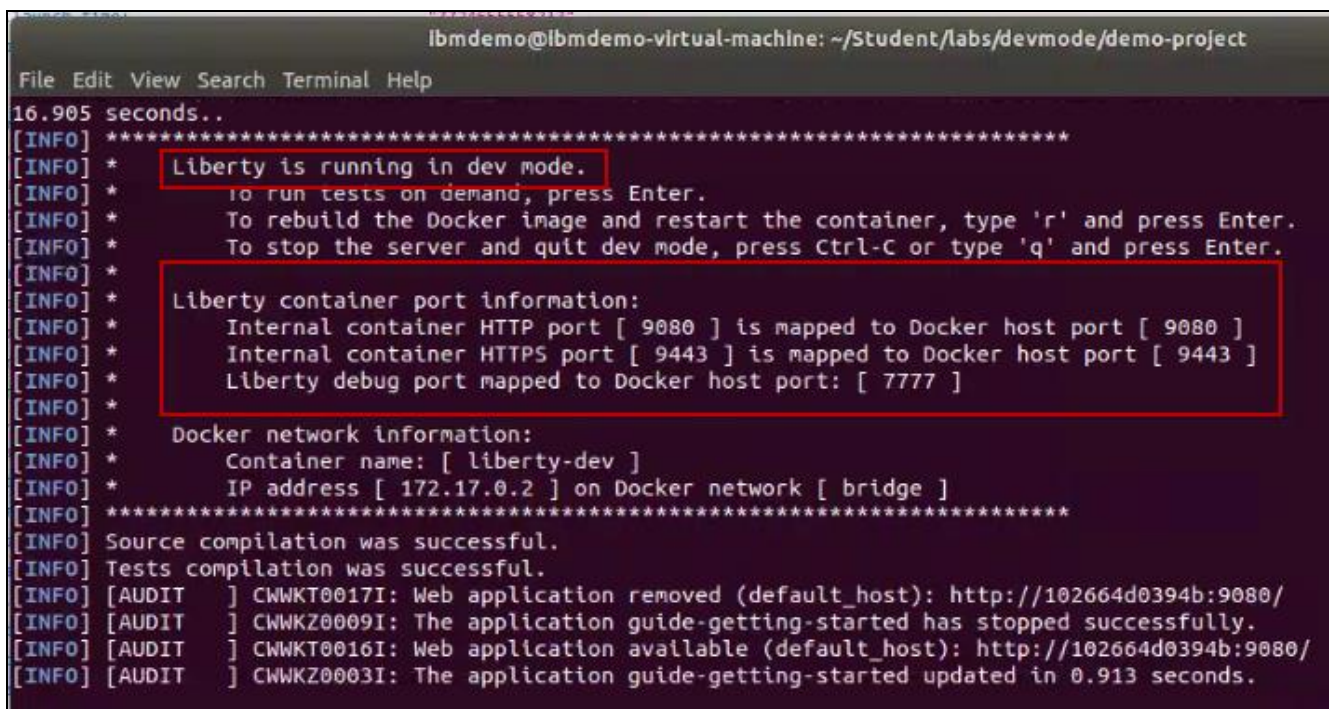
The Open Liberty Maven plug-in includes a **devc** goal that simplifies developing your application in a Docker container by starting dev mode with container support.

This **devc** goal builds a Docker image, mounts the required directories, binds the required ports, and then runs the application inside of a container. Dev mode also listens for any changes in the application source code or configuration and rebuilds the image and restarts the container as necessary.

- __1. Build and run the container by running the devc goal from the start directory:

```
mvn liberty:devc
```

- a. When you see the message: **"liberty is running in dev mode"**, Open Liberty is ready to run in dev mode:



```
ibmdemo@ibmdemo-virtual-machine: ~/Student/labs/devmode/demo-project
File Edit View Search Terminal Help
16.905 seconds..
[INFO] *****
[INFO] * Liberty is running in dev mode.
[INFO] * To run tests on demand, press Enter.
[INFO] * To rebuild the Docker image and restart the container, type 'r' and press Enter.
[INFO] * To stop the server and quit dev mode, press Ctrl-C or type 'q' and press Enter.
[INFO] *
[INFO] * Liberty container port information:
[INFO] * Internal container HTTP port [ 9080 ] is mapped to Docker host port [ 9080 ]
[INFO] * Internal container HTTPS port [ 9443 ] is mapped to Docker host port [ 9443 ]
[INFO] * Liberty debug port mapped to Docker host port: [ 7777 ]
[INFO] *
[INFO] * Docker network information:
[INFO] * Container name: [ liberty-dev ]
[INFO] * IP address [ 172.17.0.2 ] on Docker network [ bridge ]
[INFO] *****
[INFO] Source compilation was successful.
[INFO] Tests compilation was successful.
[INFO] [AUDIT ] CWWKT0017I: Web application removed (default_host): http://102664d0394b:9080/
[INFO] [AUDIT ] CWWKZ0009I: The application guide-getting-started has stopped successfully.
[INFO] [AUDIT ] CWWKT0016I: Web application available (default_host): http://102664d0394b:9080/
[INFO] [AUDIT ] CWWKZ0003I: The application guide-getting-started updated in 0.913 seconds.
```

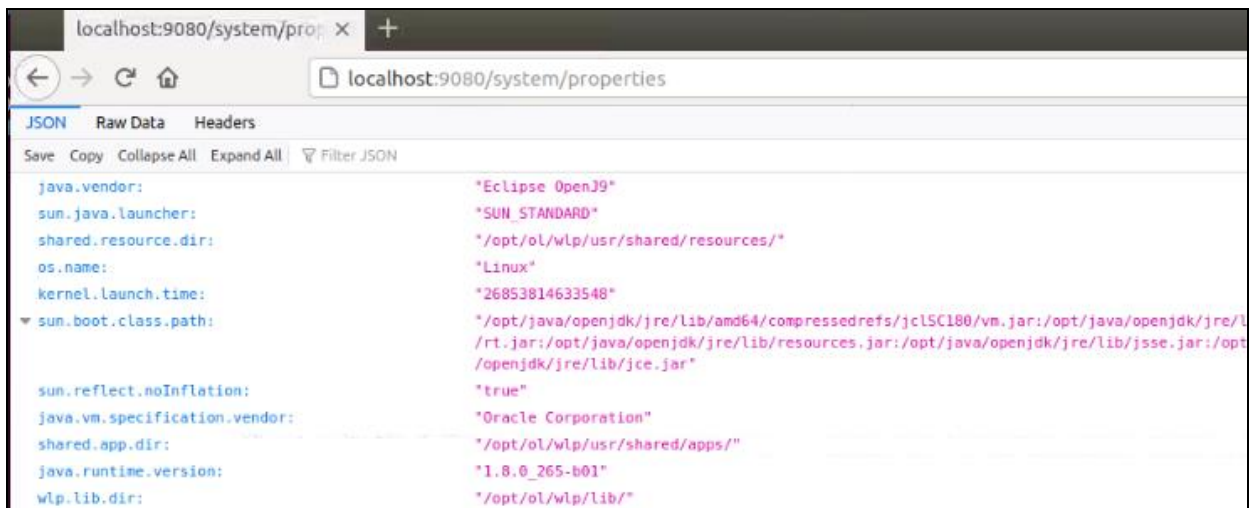
- __2. Open another Terminal window and run the **docker ps** command to verify that the container is started.

```
docker ps
```

A terminal window titled 'ibmdemo@ibmdemo-virtual-machine: ~' showing the output of the 'docker ps' command. The output is a table with columns: CONTAINER ID, IMAGE, COMMAND, CREATED, NAMES, and STATUS. One container is listed with ID '102664d0394b', image 'guide-getting-started-dev-mode', and status 'Up 2 minutes'. A red box highlights the 'COMMAND', 'CREATED', and 'NAMES' columns for this container.

CONTAINER ID	IMAGE	COMMAND	CREATED	NAMES	STATUS
102664d0394b	guide-getting-started-dev-mode	"/opt/ol/helpers/run..."	2 minutes ago	liberty-dev	Up 2 minutes

- __3. From a Web Browser, access the application using: <http://localhost:9080/system/properties>



Dev mode automatically picks up changes that you make to your application and allows you to run tests by pressing the **enter/return** key in the active command-line session.

- __4. Run the integration test that is included in the application project.

Note: The test case simply creates a client and invokes the <http://localhost:9080/sste/properties> endpoint. The expected HTTP response code is 200, which indicates a successful http response from the system/properties service. The test case will fail if the response code is anything other than 200.

- a. From the Terminal window that the **mvn liberty:devc** is running, press the **ENTER** key to run the tests. You will see that the test runs with no failures, as illustrated below.

```
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running it.io.openliberty.sample.PropertiesEndpointIT
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.493
[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.
```

IMPORTANT:



Next, you will introduce a change to the system/properties endpoint, which will break the test.

In this lab, you will not go into detail on the test cases themselves because you will explore testing in greater detail in a subsequent lab when working with Liberty dev mode in the VS Code IDE.

__5. Update the **server.xml** file to change the context root from **/** to **/dev**

- a. From a terminal window, navigate to the **liberty config** folder of the application

```
cd /home/ibmdemo/Student/labs/devmode/demo-  
project/src/main/liberty/config
```

- b. Use the gedit editor to open the **index.html** file in edit mode

```
gedit server.xml
```

- c. Make the following minor change to the index.html page

Change the highlighted line:

```
<server description="Sample Liberty server">
  <featureManager>
    <feature>jaxrs-2.1</feature>
    <feature>jsonp-1.1</feature>
    <feature>cdi-2.0</feature>
    <feature>mpMetrics-2.3</feature>
    <feature>mpConfig-1.4</feature>
    <feature>mpHealth-2.2</feature>
  </featureManager>

  <variable name="default.http.port" defaultValue="9080"/>
  <variable name="default.https.port" defaultValue="9443"/>

  <webApplication location="guide-getting-started.war" contextRoot="/" />

  <mpMetrics authentication="false"/>

  <httpEndpoint host="*" httpPort="${default.http.port}"
    httpsPort="${default.https.port}" id="defaultHttpEndpoint"/>

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

Updated to read: contextRoot="/dev" />

```
<server description="Sample Liberty server">
  <featureManager>
    <feature>jaxrs-2.1</feature>
    <feature>jsonp-1.1</feature>
    <feature>cdi-2.0</feature>
    <feature>mpMetrics-2.3</feature>
    <feature>mpConfig-1.4</feature>
    <feature>mpHealth-2.2</feature>
  </featureManager>

  <variable name="default.http.port" defaultValue="9080"/>
  <variable name="default.https.port" defaultValue="9443"/>

  <webApplication location="guide-getting-started.war" contextRoot="/dev" />

  <mpMetrics authentication="false"/>

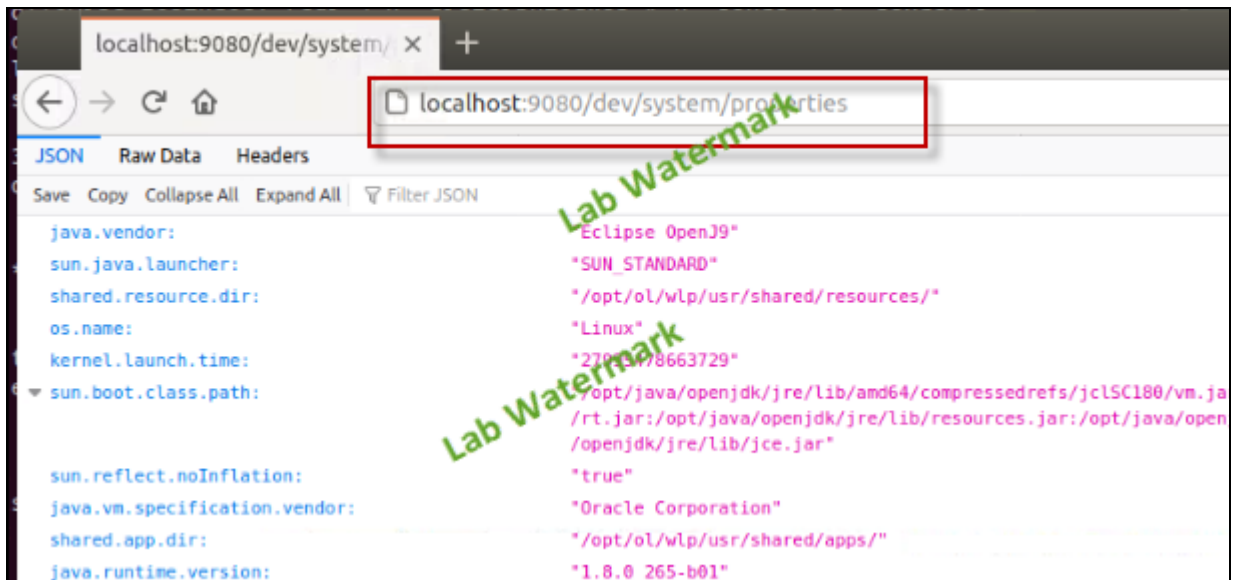
  <httpEndpoint host="*" httpPort="${default.http.port}"
    httpsPort="${default.https.port}" id="defaultHttpEndpoint"/>

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

- d. **Save** the file and **close** the gedit editor.

- ___6. After you **SAVE** the file changes, Open Liberty automatically reloads its configuration. You can access the application at the <http://localhost:9080/dev/system/properties> URL.

Notice that context root is now **/dev**



IMPORTANT:

We encountered situations where the container got mapped to Docker host port 9081 instead of 9080.



```

ibmdemo@ibmdemo-virtual-machine:~/Student/labs/devmode/deno-project/src/main/liberty/config$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  NAMES                  CREATED          STATUS
d0bdf93bb11a   guide-getting-started-dev-mode      "/opt/ol/helpers/run..." 49 seconds ago       Up 47
0:7777->7777/tcp, 0.0.0.0:9443->9443/tcp, 0.0.0.0:9081->9080/tcp liberty-dev

```

This happens when the new container comes up before the old container ports have been freed. To get the container back to port 9080, devc or do a minor change to the server.xml..

- ___7. Rerun the integration test that is included in the application project.
 - a. From the Terminal window that the **mvn liberty:devc** is running, press the **ENTER** key to run the tests. You will see that the test runs, but now FAILS because a 404 HTTP response code was returned, using the original context root.


```

[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running it.io.openliberty.sample.PropertiesEndpointIT
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.421 s <<< FAILURE! - in it.io.openliberty.sample.PropertiesEndpointIT
[ERROR] testGetProperties Time elapsed: 0.411 s <<< FAILURE!
org.opentest4j.AssertionFailedError: Incorrect response code from http://localhost:9080/ ==> expected: <200> but was: <404>
    at it.io.openliberty.sample.PropertiesEndpointIT.testGetProperties(PropertiesEndpointIT.java:45)

[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   PropertiesEndpointIT.testGetProperties:45 Incorrect response code from http://localhost:9080/ ==> expected: <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/demo-project/target/failsafe-reports for the 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.

```

Note: For the test case to PASS, the test case itself would need to be updated to reflect invoking the system/properties service using the new /dev context root. This activity is beyond the scope of this lab. However, you will explore the tests in greater detail in a subsequent lab.

1.8 Lab cleanup and completion

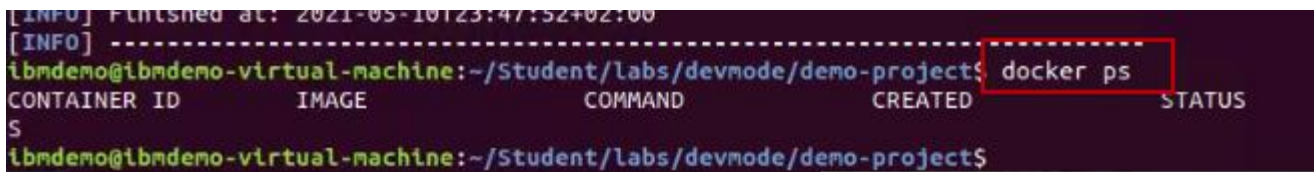
___1. Exit Liberty devc mode

- a. When you are finished, **exit** dev mode by pressing **CTRL+C** in the command-line session that the container was started from, or by typing `q` and then pressing the `enter/return` key.

Note: Either of these options stops and removes the container.

- b. To check that the container was stopped, run the **docker ps** command.

```
docker ps
```



```
[INFO] Finished at: 2021-05-10T23:47:52+02:00
[INFO] -----
ibndemo@ibndemo-virtual-machine:~/Student/Labs/devmode/demo-project$ docker ps
CONTAINER ID    IMAGE    COMMAND    CREATED    STATUS
5               ibndemo  S          5m         S
ibndemo@ibndemo-virtual-machine:~/Student/Labs/devmode/demo-project$
```

Congratulations! You have successfully completed the lab “Getting Started with Liberty and Dev Mode.

In this exercise, you learned how developers can use Liberty in “dev” mode for achieving efficient iterative develop, test, debug cycle when developing Java based applications / microservices.

You explored:

- Using Liberty dev mode (stand-alone) without an IDE
- Hot reloading of application code and configuration changes using dev mode
- Working with Liberty dev mode in containers
- Running integrated unit/integration tests from Liberty dev mode

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