

Lab 01 Getting Started with Docker

Contents

LAB 01	GETTING STARTED WITH DOCKER.....	6
1.1	INTRODUCTION	6
1.2	APPLICATION MODERNIZATION WITH IBM CLOUD PAK FOR APPLICATIONS – PROOF OF TECHNOLOGY	6
1.3	WHAT IS DOCKER	7
1.4	AUDIENCE FOR THIS PROOF OF TECHNOLOGY	7
1.5	LAB ENVIRONMENT	7
1.6	USER ID AND PASSWORDS	8
1.7	INSTALLATION OF IBM CLOUD PAK FOR APPLICATIONS	8
1.8	LINUX TIPS.....	8
1.9	REVIEW SCRIPT BEFORE RUNNING	8
1.10	LET’S GET STARTED	9
APPENDIX: SKYTAP TIPS FOR LABS		21
1.1	HOW TO USE COPY / PASTE BETWEEN LOCAL DESKTOP AND SKYTAP VM?.....	21

1.1 Introduction

This is “**Lab 01– Getting started with Docker**”, from an IBM Cloud Pak for Applications & App Modernization Proof of technology (PoT). The labs are not required to be executed in order. And, you may skip labs, and only perform the labs that suit your desired learning objectives.

The full set of labs in the PoT are:

Lab01 - Getting started with Docker

Lab02 - Explore RedHat OpenShift Container Platform

Lab03 - Getting started with Kubernetes

Lab04 – Liberty application deployment using Operators

Lab05 – IBM Cloud Pak for Applications - App Modernization using Transformation Advisor

Lab06 – App Modernization with Java EE Microservices and Liberty

Lab07 – Using Tekton pipelines for CI/CD of microservices to RedHat OpenShift Container Platform

1.2 Application Modernization with IBM Cloud Pak for Applications – Proof of Technology

The goal of this lab is to provide background working with standalone Docker images and containers running a IBM WebSphere® Liberty application. Since Docker is application runtime used by Kubernetes and in turn by IBM Cloud Pak for Applications (CP4Apps) running on Red Hat OpenShift Container Platform (RHOC). You can start by learning some basic concepts.

1.3 What is Docker

Docker is a Linux kernel extension that provides for operating system virtualization, also known as “containerization”. Applications are packaged in Docker images and then deployed as containers. Containers are isolated from each other and bundle their own programs, libraries, applications and configuration, which can communicate with other containers. A typical example for an application would be one container with the web or application server and a second container running a database.

1.4 Audience for this Proof of Technology

This Proof of Technology is mainly designed for database administrators (DBAs), Linux system administrators, application architects, solution designers and infrastructure professionals. IBM Cloud Pak for Apps is a collection of many different technologies and not every aspect can be covered in a single Proof of Technology. In this session, we cover certain aspects that will be useful to a specific role. For example:

Role	Capabilities
Database administrators	Learn a new paradigm of database software deployment and database management.
Application architects	Deploy application servers using a push-button approach and evaluate existing applications suitability to deploy in new paradigm of using cloud capabilities.
Solution designers	Learn new greenfield architecture of microservices and implement those on your platform.
Infrastructure professionals	Learn about Kubernetes and Docker containers orchestration using IBM Cloud Pak for Apps on OpenShift Container Platform in your own environment.

1.5 Lab environment

We are using a single virtual machine (VM) to demonstrate IBM Cloud Pak for Apps. the lab environment was built to the following specifications:

- The operating system is Red Hat Enterprise Linux Server 7.7
- The kernel version is: Linux 3.10.0-1062.4.1.el7.x86_64 x86_64
- The docker version installed is: Docker version 1.13.1, build 4ef4b30/1.13.1
- The Red Hat OpenShift Container Platform version is: RHOC 3.11.153
- The recommended host memory to run the VM is 20 GB RAM
- The minimum SSD space required to build and run the labs is 100 GB of free space

1.6 User ID and passwords

The logon credentials for the primary Operating System user is:

Username: `ibmdemo`

Password: `passw0rd` (with a zero), which is also the `sudo` password.

1.7 Installation of IBM Cloud Pak for Applications

IBM Cloud Pak for Applications, which pre-reqs Red Hat OCP and Docker, is already installed on an IBM-provided laptop.

1.8 Linux tips

If you are new to the Linux environment, the following tips should help.

- As opposed to the Microsoft Windows, you must click in a Linux GUI window to get it to focus, so you can type the commands.
- You can type the `clear` command from the command line window to clear the contents.
- When you are typing in a command shell, after few characters, click the **tab** key to autocomplete instead of typing the whole command. This will save some typing.
- To close a command line window, you can either type `exit` followed by `Enter` or press `CTRL-D`.
- The command that you need to type are bold faced such as:
`$ ls -l`
- If the command shown in this lab documents starts with `$`, please assume that you are running that command as `ibmdemo` user.
- You should not need to run as root as `ibmdemo` should have sufficient rights. However, you may need to prefix the command with `sudo` to execute as a privileged (root) user. For example: `sudo kubectl (command goes here)`.
- It is assumed that you will click `Enter` after typing the command.

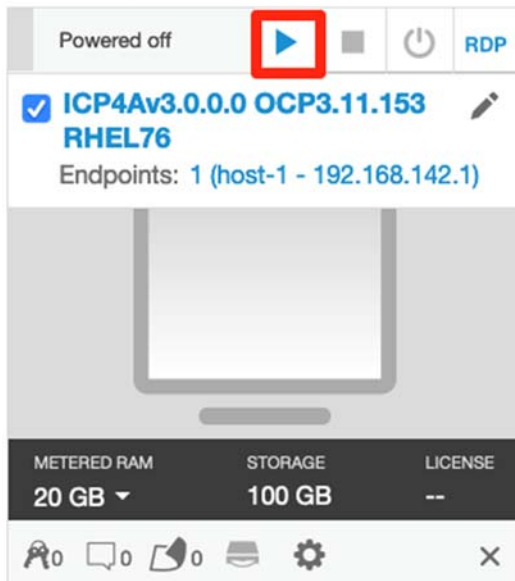
1.9 Review script before running

1. While doing the lab exercises, we ask you to type the name of the script or file to run. We recommend that you view the contents of the script before running it.
2. You can use a method of your choice to view the contents of the script, such as one of the following commands.
`$ more <script name>` → See the contents one screen at a time
`$ cat <script name>` → Equivalent of Windows type command.
3. In most cases the lab exercises do not have explicit instructions requesting you to view the contents of the script. It is good practice to view the script before you run it.

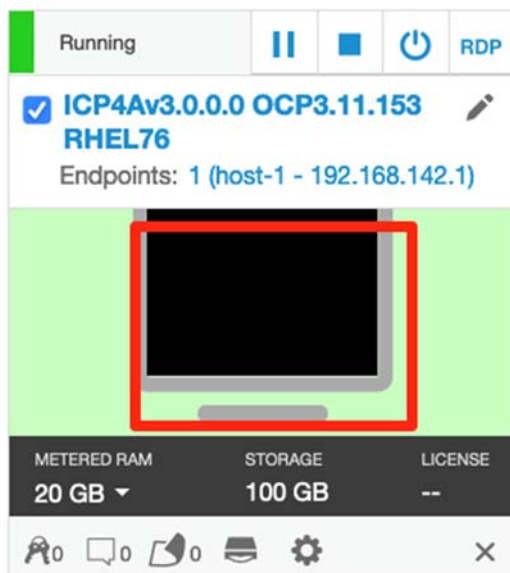
1.10 Let's get started

On your laptop/workstation, locate the [ICP4Av3.0.0.0 OCP3.11.153 RHEL76](#) virtual machine

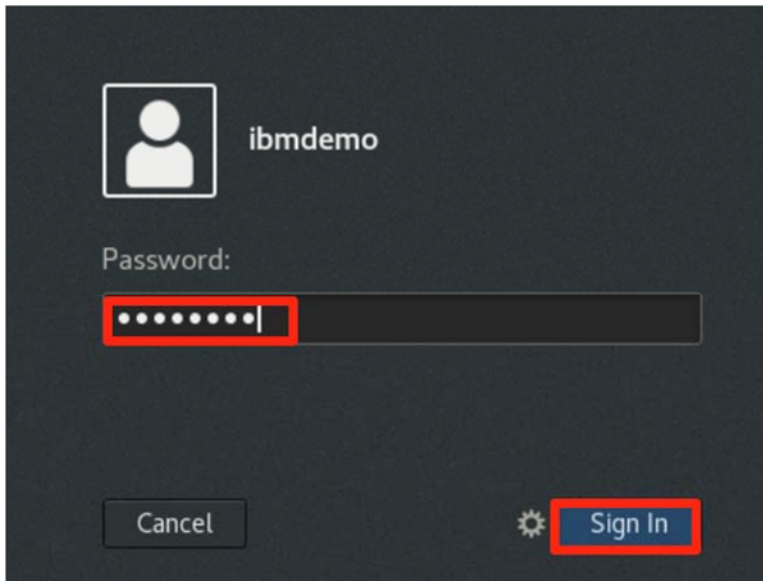
1. The VM should already be running. If not, Launch the Lab environment by clicking the **Run this VM** icon.



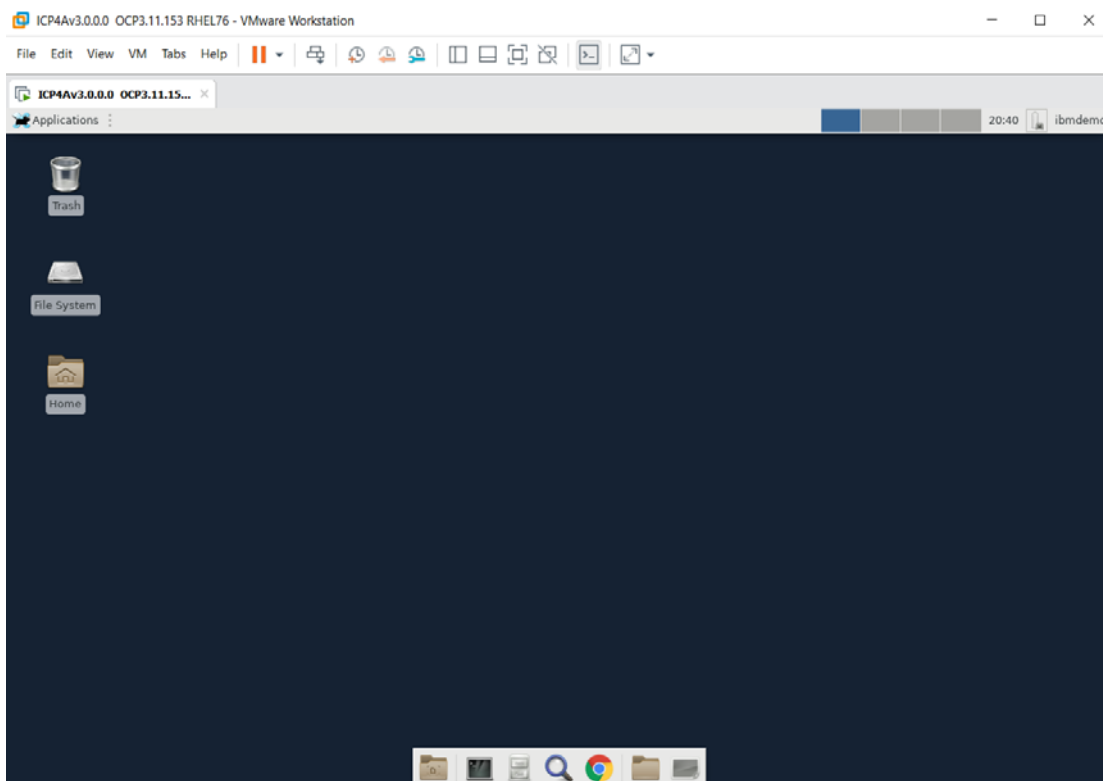
2. After the VM is running, click its icon to access the VM's desktop.



- ___3. After the VM machine powers on, log with the [ibmdemo](#) user using the password [passw0rd](#)



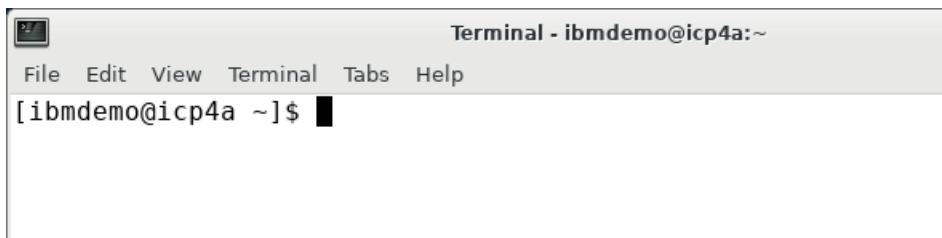
The [ICP4Av3.0.0.0 OCP3.11.153 RHEL76](#) virtual machine running and its Desktop is displayed in a web browser window.




- __4. Click the **Terminal** icon from the bottom of the desktop to open a command line terminal.




You'll be running in the terminal as the user **ibmdemo**



	Note: Please note that if needed root access can be obtained with sudo su -
--	---

- __5. Type **cd student/lab1** to access the Lab1 material.

	Note: Refer to the Appendix in this lab guide for details for using Copy / Paste between the lab guide and the lab environment.
---	--

- __6. Examine the Dockerfile which will be build a Liberty Docker image by typing **cat Dockerfile**

```
[ibmdemo@icp4a lab1]$ cat Dockerfile
# (C) Copyright IBM Corporation 2015,2018.
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
```

```
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# If on a slow network uncomment the line below to use image with Liberty server and feature(s)
#FROM lab1:latest

# If on slow network comment the line below
FROM ibmcom/websphere-liberty:19.0.0.6-kernel-ubi-min
COPY server.xml /config/
COPY ServletApp.war /config/apps/

USER root
RUN chown default:root -R /opt/ibm/wlp/usr/servers/defaultServer
USER 1001

RUN configure.sh
```

This file builds a Liberty Docker image with an application with the following actions:

- __a. Pulls the websphere-liberty-19.0.0.6-kernel UBI (Universal Base Image) image from Docker Hub
- __b. Copies a Liberty server.xml file from the current directory to the /config directory in the image
- __c. Copies an application war file from the current directory to the /config/apps directory
- __d. As the root USER specifies file permissions so that the non-root user that the container runs as can access the Liberty configuration, binaries and application binaries (In Docker 17.05 and above the chown command can be included in the Docker COPY command(s))
- __e. Runs the Liberty Docker image configure.sh script which prepares the Liberty environment in the image and parses the Liberty server.xml feature manager list adds the needed feature(s)

- ___7. Examine the Liberty server.xml for this lab by typing `cat server.xml`

```
[ibmdemo@icp4a lab1]$ cat server.xml

<server description="default servlet engine">

  <!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
  </featureManager>


  <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080" httpsPort="-1"/>

  <webApplication contextRoot="/" location="ServletApp.war" />

</server>
```

Notice that the server.xml file:

- ___a. Specifies that only the servlet-3.1 API feature is added to Liberty for this application

	<p>Note: If no API features were listed Liberty would introspect the application binaries and automatically load the API features required by the application.</p>
---	---

- ___b. Exposes port 9080 for http(s) traffic

- ___c. Defines the contextRoot for the application in the application in the ServletApp.war file

- ___8. Build a Liberty docker image named simpleapp by typing `docker build -t simpleapp .` (note the “.” at the end of the command, which will build an image named `simpleapp` using the Dockerfile in the local directory “.”)

```
[ibmdemo@icp4a lab1]$ docker build -t simpleapp .

Sending build context to Docker daemon 11.78 kB
Step 1/7 : FROM docker.io/ibmcom/websphere-liberty:19.0.0.6-kernel-ubi-min
--> 7810d7fa4666
Step 2/7 : COPY server.xml /config/
--> Using cache
--> 6edb30c0af77
Step 3/7 : COPY ServletApp.war /config/apps/
--> a3723ec150d9
Removing intermediate container 07c7632f0288
Step 4/7 : USER root
--> Running in e13bdf4044d8
--> 187f8d0995ed
Removing intermediate container e13bdf4044d8
Step 5/7 : RUN chown default:root -R /opt/ibm/wlp/usr/servers/defaultServer
--> Running in 94ee6f6f54b4

--> 44b1984ac0c4
Removing intermediate container 94ee6f6f54b4
Step 6/7 : USER 1001
--> Running in b8131fa83b3e
--> 78589551606b
Removing intermediate container b8131fa83b3e
```



```

Step 7/7 : RUN configure.sh
---> Running in 5cb8b3f4a436

+ WLP_INSTALL_DIR=/opt/ibm/wlp
+ SHARED_CONFIG_DIR=/opt/ibm/wlp/usr/shared/config
+ SHARED_RESOURCE_DIR=/opt/ibm/wlp/usr/shared/resources
+ SNIPPETS_SOURCE=/opt/ibm/helpers/build/configuration_snippets
+ SNIPPETS_TARGET=/config/configDropins/overrides
+ mkdir -p /config/configDropins/overrides
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ '[' '' == client ']'
+ '[' '' == embedded ']'
+ '[' '' == true ']'
+ '[' '' == true ']'
+ installUtility install --acceptLicense defaultServer
Checking for missing features required by the server ...
The server requires the following additional features: servlet-3.1. Installing features from the repository ...
Establishing a connection to the configured repositories . . .
This process might take several minutes to complete.

Successfully connected to all configured repositories.

Preparing assets for installation. This process might take several minutes to complete.

Additional Liberty features must be installed for this server.

To install the additional features, review and accept the feature license agreement:
The --acceptLicense argument was found. This indicates that you have
accepted the terms of the license agreement.

Step 1 of 4: Downloading servlet-3.1 ...
Step 2 of 4: Installing servlet-3.1 ...
Step 3 of 4: Validating installed fixes ...
Step 4 of 4: Cleaning up temporary files ...

All assets were successfully installed.

Start product validation...
Product validation completed successfully.
+ find /opt/ibm/fixes -type f -name '*.jar' -print0
+ sort -z
+ xargs -0 -n 1 -r -I '{}' java -jar '{}' --installLocation /opt/ibm/wlp
+ find /opt/ibm/wlp -perm -g=w -print0
+ xargs -0 -r chmod -R g+rw
+ /opt/ibm/wlp/bin/server start

Starting server defaultServer.
Server defaultServer started with process ID 103.
+ /opt/ibm/wlp/bin/server stop

Stopping server defaultServer.
Server defaultServer stopped.
+ rm -rf /output/resources/security/ /output/messaging /logs/console.log /logs/messages.log
/logs/messages_19.12.04_20.44.34.0.log /opt/ibm/wlp/output/.classCache

```

```
+ chmod -R g+rwX /opt/ibm/wlp/output/defaultServer
+ find /opt/ibm/wlp -type d -perm -g=x -print0
+ xargs -0 -r chmod -R g+rwX
---> 59258a04abcf
Removing intermediate container 5cb8b3f4a436
Successfully built 59258a04abcf
```

- ___9. You can obtain a list of all images in the local Docker registry by typing `docker images` and the scroll up and down in the shell. (Note most of the images are in the Docker Registry are used for RHOCP and ICP4Apps)
- ___10. You can also use the “grep” command to filter the images listed by typing `docker images | grep simpleapp`

```
[ibmdemo@icp4a lab1]$ docker images | grep simpleapp

simpleapp                                latest                bdd14974c316        4 minutes ago
365 MB

[ibmdemo@icp4a lab1]$
```

- ___11. Run the command `docker images | grep websphere-liberty` to display the liberty image that was used to build the simpleapp image (in some cases an additional intermediate image used to build a container may be listed).

```
[ibmdemo@icp4a lab1]$ docker images | grep websphere-liberty

docker.io/ibmcom/websphere-liberty      19.0.0.6-kernel-ubi-min  4b96d9ab9f54        4
days ago                               335 MB
```

- ___12. Before running containers with the image that you just built, run the following command:

```
docker pull ibmcom/websphere-liberty:19.0.0.9-kernel-ubi-min
```

NOTE If this command takes more than 1-2 minutes to complete, open another command shell, run `cd ~/student/lab1` and proceed with the remaining instructions, you can return to this command shell later to review the results

```
[ibmdemo@icp4a lab1]$ docker pull ibmcom/websphere-liberty:19.0.0.9-kernel-ubi-min

Trying to pull repository registry.redhat.io/ibmcom/websphere-liberty ...
Trying to pull repository docker.io/ibmcom/websphere-liberty ...
19.0.0.9-kernel-ubi-min: Pulling from docker.io/ibmcom/websphere-liberty
37b1dccc0828: Pull complete
70cf862fdf8a: Pull complete
f10a7057449d: Pull complete
0812c1c2516a: Pull complete
25d0a2624e59: Pull complete
35a02580b5d5: Pull complete
1259cde7bf9a: Pull complete
3274a00873e8: Pull complete
d8b7966867d2: Pull complete
1f5b215ea922: Pull complete
Digest: sha256:edae94995aa32e20f7e7c90da5b00ea76f151b45d0011a316cc406878d3fd1b8
Status: Downloaded newer image for docker.io/ibmcom/websphere-liberty:19.0.0.9-kernel-ubi-min
```

As you can see some of the layers of the image binaries already exist, they were pulled when the websphere-liberty:19.0.0.6 kernel-ubi-min was pulled. Those existing layers, also known as “intermediate images” will be reused when building another image which employs the same layers.

__13. Run the command: `docker images | grep websphere-liberty`

```
[ibmdemo@icp4a lab1]$ docker images | grep websphere-liberty
```

docker.io/ibmcom/websphere-liberty	19.0.0.6-kernel-ubi-min	4b96d9ab9f54	4
days ago 335 MB			
docker.io/ibmcom/websphere-liberty	19.0.0.9-kernel-ubi-min	a68a678ba2e5	
4 days ago 335 MB			

You can see that there are now **two images** for websphere-liberty:

- the 19.0.0.6 kernel ; the 2019 June release
- the 19.0.0.9 kernel; the 2019 September release (in some cases an additional intermediate image used to build a container may be listed).

__14. Now that you have an image let's run it in a container by typing the following command:

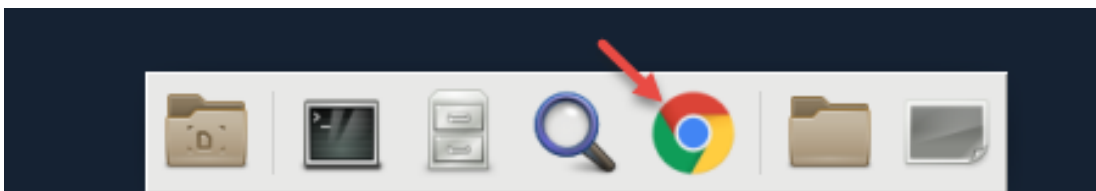
```
docker run --name=simpleC1 --hostname=localhost -d -p 8081:9080 simpleapp
```

The command will start a container named “simpleC1” and map port 9080 in the container to port 8081 on the localhost using the image “simpleapp” which you just created.

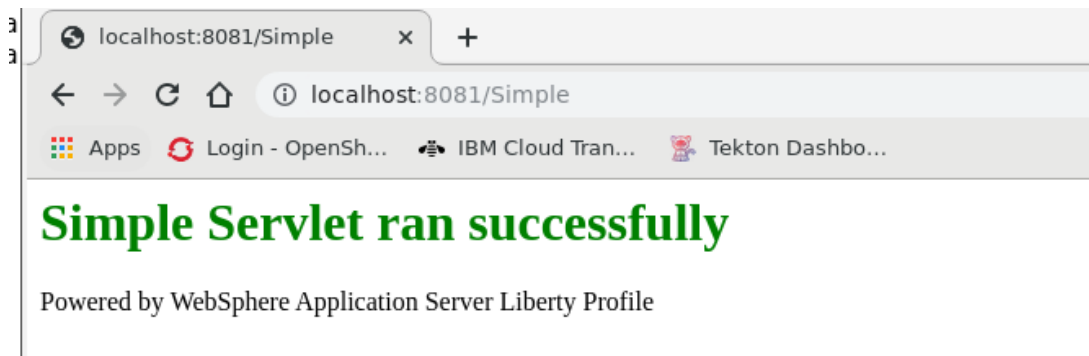
```
[ibmdemo@icp4a lab1]:$ docker run --name=simpleC1 --hostname=localhost -d -p 8081:9080 simpleapp
```

```
ba9a790061d60fd9e77bbab4a5b98cd671d2ca4dc4c08ce216a7aae8c16cd275
```

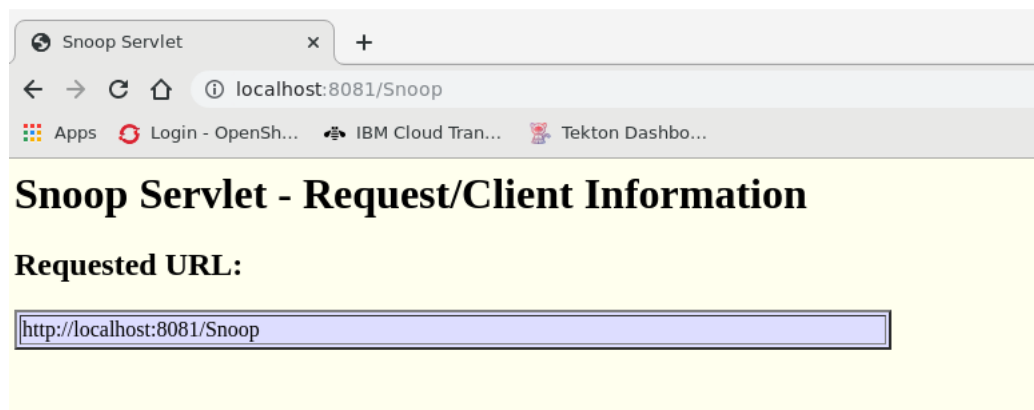
__15. Launch the [Chrome](#) browser by clicking on the icon at the bottom of the desktop



__16. Enter the address <http://localhost:8081/Simple>



__17. Change the address bar to <http://localhost:8081/Snoop>



- ___18. Scroll down the browser page note the **Request Information** specifically the **addresses** and **ports** which reflect the container environment

Request Information:

Request method	GET
Request URI	/Snoop
Request protocol	HTTP/1.1
Servlet path	/Snoop
Path info	<none>
Path translated	<none>
Character encoding	<none>
Query string	<none>
Content length	<none>
Content type	<none>
Server name	localhost
Server port	8081
Remote user	<none>
Remote address	172.17.0.1
Remote host	172.17.0.1
Remote port	53818
Local address	172.17.0.3
Local host	localhost
Local port	9080
Authorization scheme	<none>
Preferred Client Locale	en_US
All Client Locales	en_US
All Client Locales	en

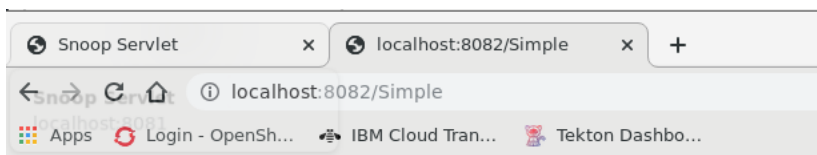
- ___19. Now return to the command line shell and start another container by typing the following:

```
docker run --name=simpleC2 --hostname=localhost -d -p 8082:9080 simpleapp
```

The command will start a container named “**simpleC2**” and map port **9080** in the container to port **8082** on the localhost using the same image (note the use of port 8082 to avoid port conflicts)

```
[ibmdemo@icp4a lab1]$ docker run --name=simpleC2 --hostname=localhost -d -p 8082:9080 simpleapp
f73eaf2b20d766fc95cc876875517224bfcbb7030f4f0036c17b58d7f52edb68
```

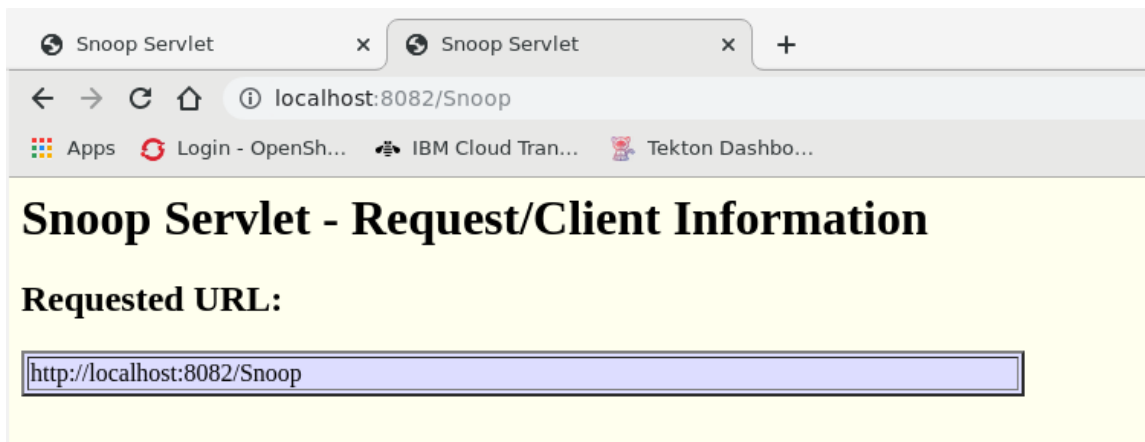
- ___20. Open a browser tab and enter the address <http://localhost:8082/Simple>



Simple Servlet ran successfully

Powered by WebSphere Application Server Liberty Profile

- __21. Change the address bar to <http://localhost:8082/Snoop>



- __22. Again, scroll down to the Request Information and note the addresses and ports which reflect the container environment, noted that while the Liberty is using port **9080** to serve the requests in each container, each container has different a different local address and remote port

Request Information:

Request method	GET
Request URI	/Snoop
Request protocol	HTTP/1.1
Servlet path	/Snoop
Path info	<none>
Path translated	<none>
Character encoding	<none>
Query string	<none>
Content length	<none>
Content type	<none>
Server name	localhost
Server port	8082
Remote user	<none>
Remote address	172.17.0.1
Remote host	172.17.0.1
Remote port	51572
Local address	172.17.0.4
Local host	localhost
Local port	9080
Authorization scheme	<none>
Preferred Client Locale	en-US

- __23. Now return to the terminal window and type: `docker ps | grep simple` which will show the docker container processes running for this lab (without limiting the results with “grep simple” all the docker processes would have been listed including those for the containers in this lab)

```
[ibmdemo@icp4a lab1]$ docker ps | grep simple

f73eaf2b20d      simpleapp
"/opt/ibm/helpers/..." 2 minutes ago    Up 2 minutes      9443/tcp, 0.0.0.0:8082->9080/tcp
simpleC2

4b3e50826176     simpleapp
"/opt/ibm/helpers/..." 5 minutes ago    Up 5 minutes      9443/tcp, 0.0.0.0:8081->9080/tcp
simpleC1
```

You now have some familiarity with docker images and containers. As you can see it's easy to create images and run containers, at least on a small scale, but an enterprise needs to run containers at scale, route traffic to those contains and assign QOS values the containers which is where Kubernetes and OCP come in.

- __24. In order to avoid port conflicts in later labs, stop the containers you started by running these two commands:

```
docker stop simpleC1
```

```
docker stop simpleC2
```

```
[ibmdemo@icp4a lab1]$ docker stop simpleC1
simpleC1
[ibmdemo@icp4a lab1]$ docker stop simpleC2
simpleC2
```

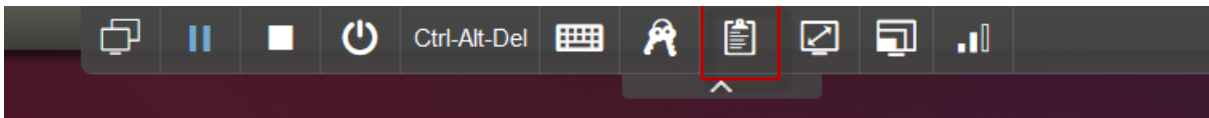
End of Lab 01 – Getting Started with Docker

Appendix: SkyTap Tips for labs

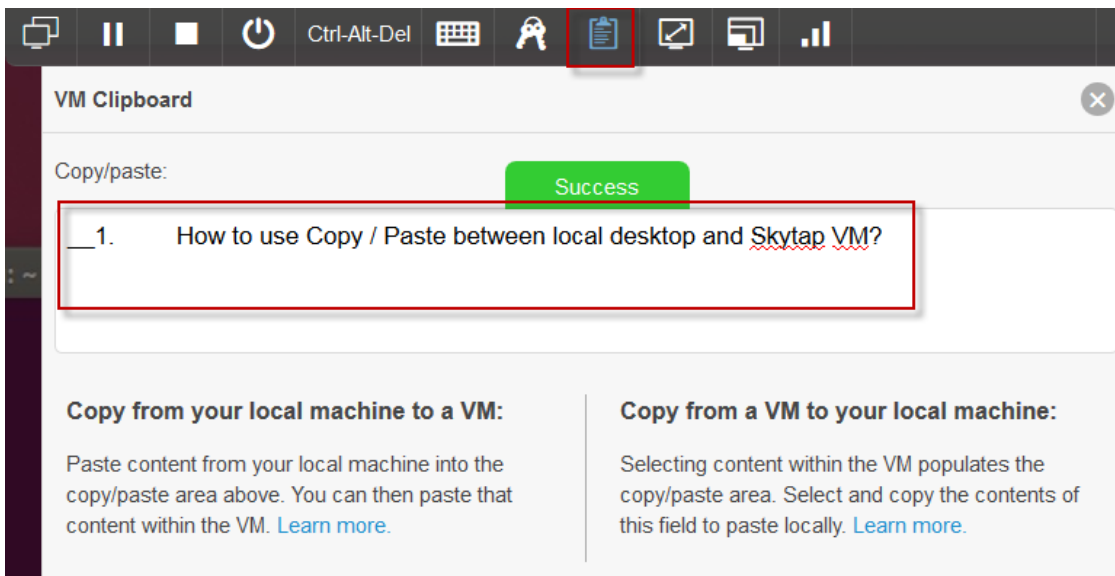
1.1 How to use Copy / Paste between local desktop and Skytap VM?

Using copy / Paste capabilities between the lab document (PDF) on your local workstation to the VM is a good approach to more efficiently work through a lab, while reducing the typing errors that often occur when manually entering data.

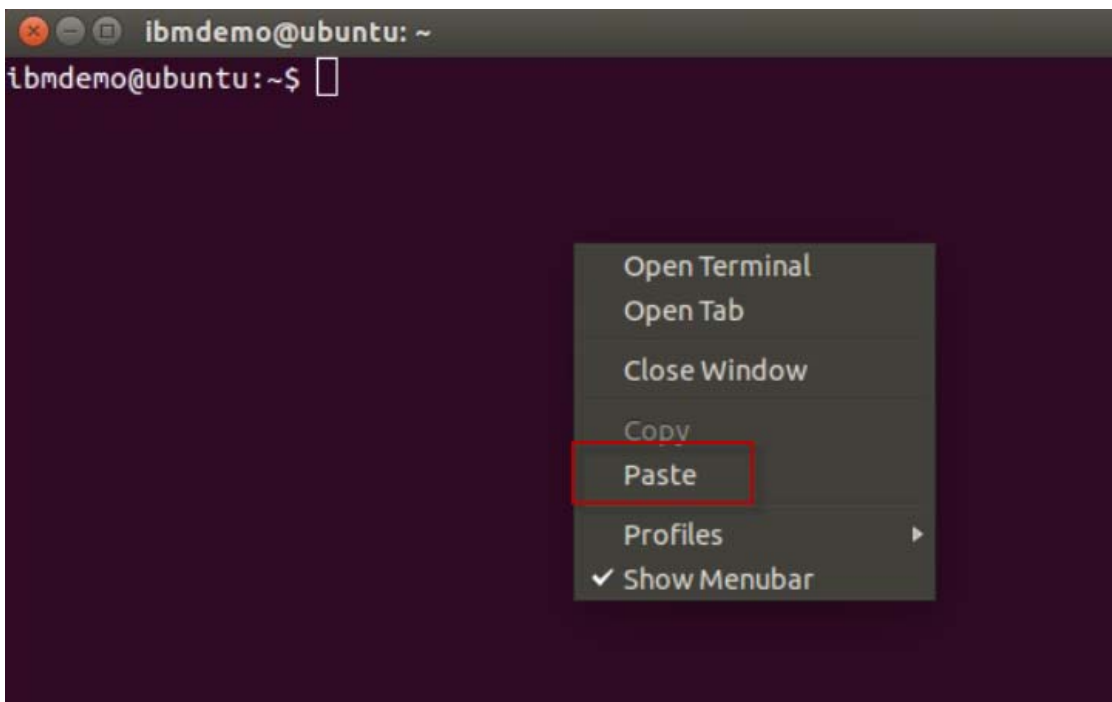
- ___1. In SkyTap, you will find that any text copied to the clipboard on your local workstation is not available to be pasted into the VM on SkyTap. So how can you easily accomplish this?
 - ___a. First copy the text you intend to paste, from the lab document, to the clipboard on your local workstation, as you always have (CTRL-C)
 - ___b. Return to the SkyTap environment and click on the Clipboard at the top of the SkyTap session window.



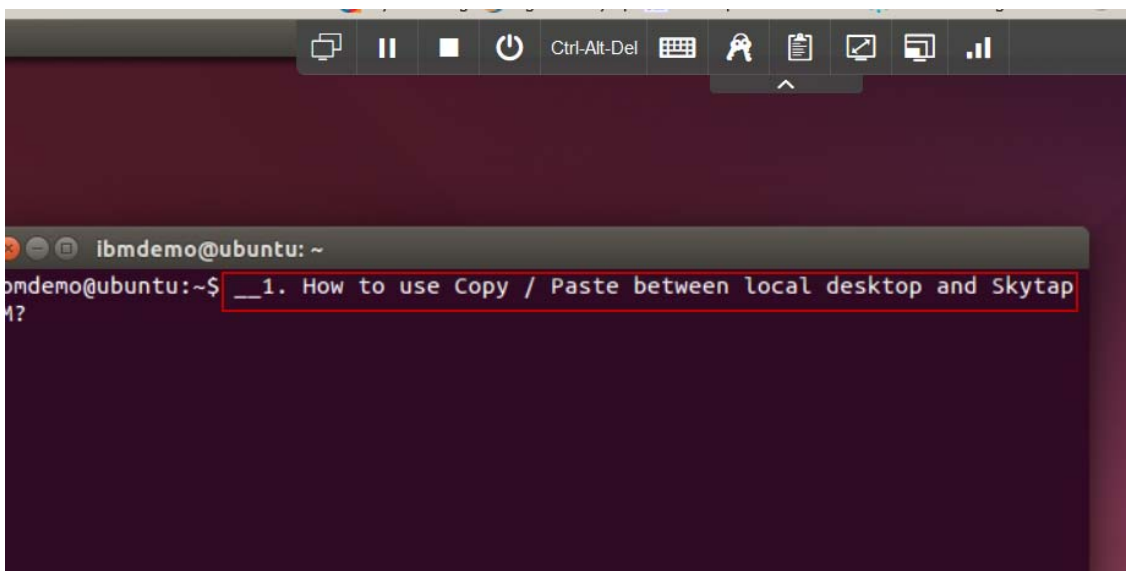
- ___c. Use **CTRL-V** to paste the content into the Copy/paste VM clipboard. Or use the **paste** menu item that is available in the dialog, when you right mouse click in the clipboard text area.



- ___d. Once the text is pasted, just navigate away to the VM window where you want to paste the content. Then, use **CTRL-C**, or right mouse click & use the **paste menu item** to paste the content.



__e. The text is pasted into the VM



Note: The very first time you do this, if the text does not paste, you may have to paste the contents into the Skytap clipboard twice. This is a known Skytap issue. It only happens on the 1st attempt to copy / paste into Skytap.