Table of Contents

Introduction 1.1
Acknowledgements 1.2
Introduction 1.3
Setup 1.4
Lab 1 - Docker Refresh (optional) 1.5
Lab 2 - OpenShift Command Line Interface (CLI) 1.6
Lab 3 - Analyzing a microservices application 1.7
Lab 4 - Changing the microservices application using data services 1.8
Lab 4 part II - Changing the microservices application using data services with security 1.9
Resources 1.10

L104372 - Exposing data as services in a microservices architecture on OpenShift documentation

In this hands-on lab, you'll learn how to expose data as services in a Microservices Architecture running on Red Hat OpenShift Container Platform (OCP). See how to transform a microservices application to use data services instead of connecting directly to the underlying datasource. In this lab, you'll learn how to use to:

- Quickly develop a basic container-based application
- Reuse container images from the Red Hat container registry
- Migrate a simple microservices application to a containerized version using data services to connect to the underlying data sources
- Get a feel for Red Hat OpenShift Container Platform (OCP)

You'll also learn what tools to consider when implementing a containerized, microservices architecture.

Audience/Intro/Prerequisites:

This lab is geared towards developers who are interested in learning how to containerize their data applications. Attendees, during this session, will explore Microservice architecture, Red Hat OpenShift Container Platform, Red Hat JBoss Data Virtualization and Red Hat JBoss Data Grid. To accomplish this, one need a little background or experience in Linux.

Here you can find the L104372 - Exposing data as services in a microservices architecture on OpenShift book.

Labs

Lab 1	Docker refresh (optional)
Lab 2	OpenShift Command Line Interface (CLI)
Lab 3	Analyzing a Microservices Application
Lab 4	Changing the microservices application using data services
Lab 4 part II	Changing the microservices application using data services with security

Generate html/pdf/epub/mobi

You may locally create rendered forms of the documentation. To do this install gitbook and ebook-convert, then execute the following commands from the checkout location:

```
$ gitbook build ./ L104372
$ gitbook pdf ./L104372 L104372.pdf
$ gitbook epub ./L104372 L104372.epub
$ gitbook mobi ./L104372 L104372.mobi
```

Once above commands executes successfully, the `L104372` folder, `L104372.pdf` , `L104372.epub` , and `L104372.mobi` will be generated.

Acknowlegdement

Red Hat Summit	2017
Version Number	1.0
Final as of	May 5th, 2017
Author(s)	Bill Kemp Cojan van Ballegooijen Madou Coulibaly Tariq Islam

Introduction

Before we jump into the demo and see how OpenShift can support a polyglot microservice system based on many different services, let’s spend some time talking about the architecture of the demo application and the components the demo is using.

Red Hat OpenShift Container Platform

Red Hat OpenShift Container Platform (OCP) v3 is a layered system designed to expose underlying Docker-formatted container image and Kubernetes concepts as accurately as possible, with a focus on easy composition of applications by a developer. For example, install Ruby, push code, and add MySQL.

The concept of an application as a separate object is removed in favor of more flexible composition of "services", allowing two web containers to reuse a database or expose a database directly to the edge of the network.

The Docker service provides the abstraction for packaging and creating Linux-based, lightweight container images. Kubernetes provides the cluster management and orchestrates containers on multiple hosts. OCP adds:

- Source code management, builds, and deployments for developers
- Managing and promoting images at scale as they flow through your system
- Application management at scale
- Team and user tracking for organizing a large developer organization

The following topics provide high-level, architectural information on core concepts and objects you will encounter when using OCP. Many of these objects come from Kubernetes, which is extended by OCP to provide a more feature-rich development lifecycle platform.

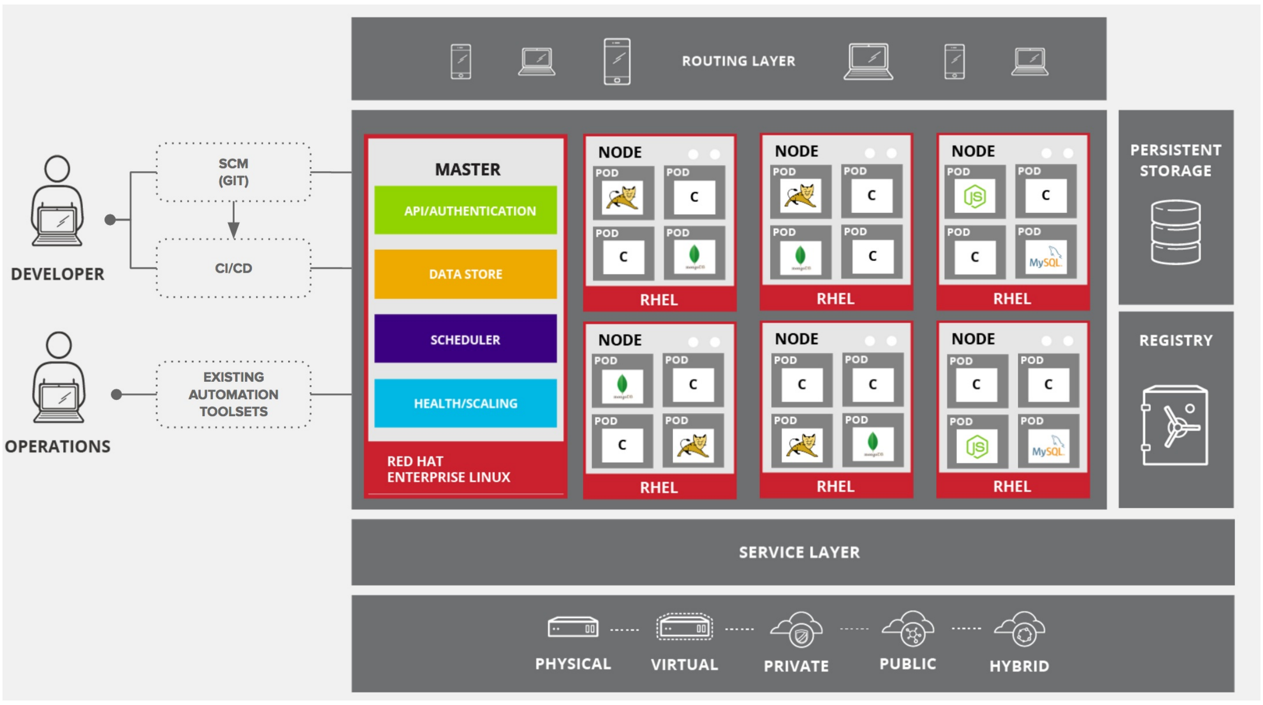
- Containers and images are the building blocks for deploying your applications.
- Pods and services allow for containers to communicate with each other and proxy connections.

- Projects and users provide the space and means for communities to organize and manage their content together.
- Builds and image streams allow you to build working images and react to new images.
- Deployments add expanded support for the software development and deployment lifecycle.
- Routes announce your service to the world.
- Templates allow for many objects to be created at once based on customized parameters.

What Is the Red Hat OpenShift Container Platform (OCP) Architecture?

OCP has a microservices-based architecture of smaller, decoupled units that work together. It can run on top of (or alongside) a Kubernetes cluster, with data about the objects stored in etcd, a reliable clustered key-value store.

The figure below depicts the architectural overview of Red Hat OpenShift Container Platform:



Those services are broken down by function:

- REST APIs, which expose each of the core objects.
- Controllers, which read those APIs, apply changes to other objects, and report status or write back to the object.

Users make calls to the REST API to change the state of the system. Controllers use the REST API to read the user’s desired state, and then try to bring the other parts of the system into sync. For example, when a user requests a build they create a "build" object. The build controller sees that a new build has been created, and runs a process on the cluster to perform that build. When the build completes, the controller updates the build object via the REST API and the user sees that their build is complete.

The controller pattern means that much of the functionality in OCP is extensible. The way that builds are run and launched can be customized independently of how images are managed, or how deployments happen. The controllers are performing the "business logic" of the system, taking user actions and transforming them into reality. By customizing those controllers or replacing them with your own logic, different behaviors can be implemented. From a system administration perspective, this also means the API can be used to script common administrative actions on a repeating schedule. Those scripts are also controllers that watch for changes and take action. OCP makes the ability to customize the cluster in this way a first-class behavior.

To make this possible, controllers leverage a reliable stream of changes to the system to sync their view of the system with what users are doing. This event stream pushes changes from etcd to the REST API and then to the controllers as soon as changes occur, so changes can ripple out through the system very quickly and efficiently. However, since failures can occur at any time, the controllers must also be able to get the latest state of the system at startup, and confirm that everything is in the right state. This resynchronization is important, because it means that even if something goes wrong, then the operator can restart the affected components, and the system double checks everything before continuing. The system should eventually converge to the user’s intent, since the controllers can always bring the system into sync.

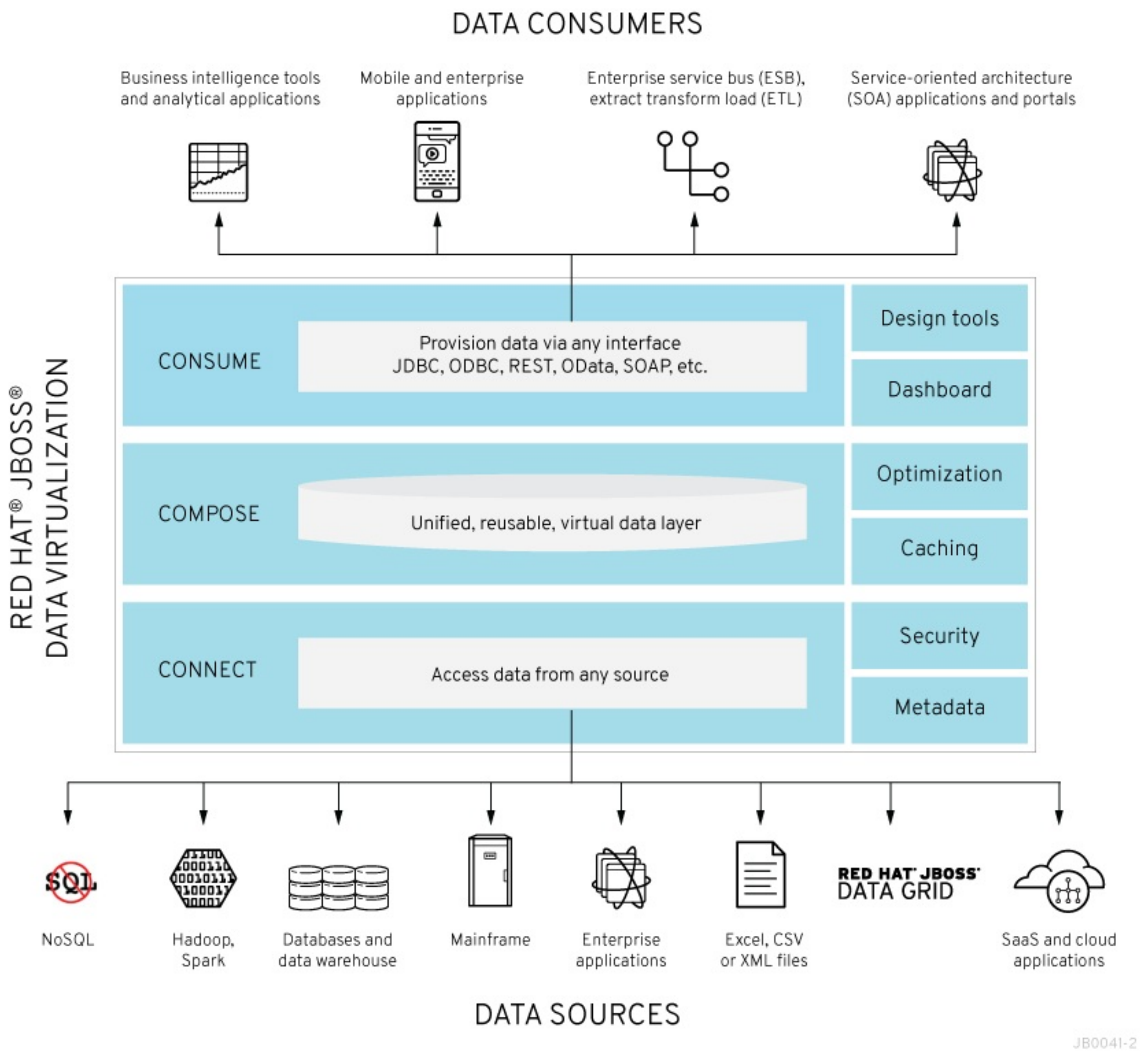
Red Hat JBoss Data Virtualization

RED HAT® JBOSS®
DATA VIRTUALIZATION

Red Hat JBoss Data Virtualization (JDV) is a complete data provisioning, federation, integration and management solution that enables organizations to gain actionable and unified information. Red Hat JBoss Data Virtualization enables agile data utilization in three steps:

1. **Connect**: Access data from multiple, heterogeneous data sources.
2. **Compose**: Create reusable, business-friendly logical data models and views by combining and transforming data.

3. **Consume**: Make unified data easily consumable through open standard interfaces.


DATA CONSUMERS
Business intelligence tools and analytical applications
Mobile and enterprise applications
Enterprise service bus (ESB), extract transform load (ETL)
Service-oriented architecture (SOA) applications and portals
RED HAT® JBOSS® DATA VIRTUALIZATION
CONSUME
Provision data via any interface JDBC, ODBC, REST, OData, SOAP, etc.
COMPOSE
Unified, reusable, virtual data layer
CONNECT
Access data from any source
Design tools
Dashboard
Optimization
Caching
Security
Metadata
NoSQL
Hadoop, Spark
Databases and data warehouse
Mainframe
Enterprise applications
Excel, CSV or XML files
RED HAT JBOSS DATA GRID
SaaS and cloud applications
DATA SOURCES
JB0041-2

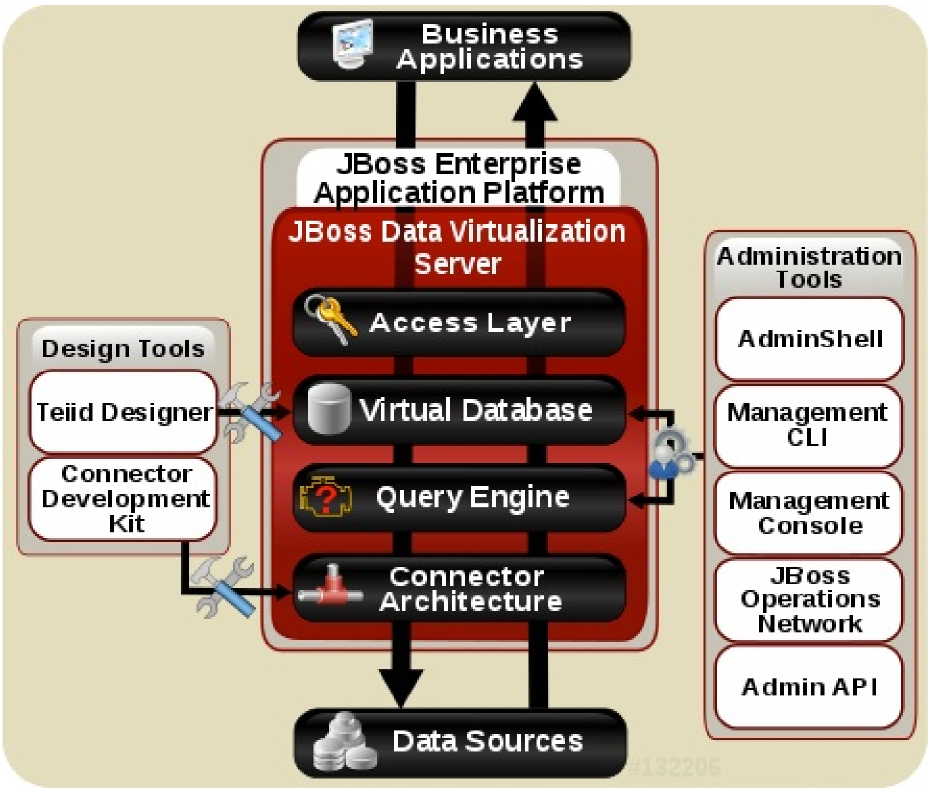
JDV includes:

- Tools for creating data views that are accessible through standard protocols. For those who are already familiar with JDV one of the tools is the Teiid Designer plug-in for Red Hat JBoss Developer Studio (JBDS).
- A robust runtime environment that provides enterprise-class performance, data integrity, and security (the JDV Server, which executes as a process within the Red Hat JBoss Enterprise Application Platform (EAP)).
- A repository for storing metadata (ModeShape)

JDV is based on the following community projects:

- Teiid (http://www.jboss.org/teiid)
- Teiid Designer (http://www.jboss.org/teiiddesigner)
- ModeShape (http://www.jboss.org/modeshape)

The figure below depicts the architectural overview of JDV:



JDV Components	Description
Query Engine	The heart of JDV Server is a high-performance query engine that processes relational, XML, XQuery and procedural queries from federated data sources. Features include support for homogeneous schemas, heterogeneous schemas, transactions, and user defined
Embedded	An easy-to-use JDBC Driver that can embed the Query Engine in any Java application.
Server	An enterprise ready, scalable, manageable, runtime for the Query Engine that runs inside JBoss EAP that provides additional security, fault-tolerance, and administrative features.
Connectors	JDV Server includes a rich set of Translators and Resource Adapters that enable access to a variety of sources, including most relational databases, web services, text files, and ldap. Need data from a different source? Custom translators and resource adaptors can easily be developed.
Tools	JDV Server includes development and administration tools to * Create - Use Teiid Designer to define virtual databases containing views, procedures or even dynamic XML documents. * Monitor & Manage - Use the Management Console with JBoss EAP or use the JDV JBoss Operations Network (JON) plugin to control any number of servers. * Script - Use the AdminShell to automate administrative and testing tasks.

The Red Hat JBoss Data Virtualization for OpenShift image is based on Red Hat JBoss Data Virtualization 6.3. In addition, the JDV for OpenShift image is built on the EAP for OpenShift image. As a result, the same differences exist for the JDV for OpenShift image. To get started with the JDV for OpenShift image please check out https://access.redhat.com/documentation/en/red-hat-xpaas/0/paged/red-hat-xpaas-jdv-for-openshift-image/chapter-3-get-started

Setup

Demo Context

The demo application we have here is a microservices application built using various technologies like Spring, AngularJS and PostgreSQL providing information of food and wine. The food and wine data is served through the food-service and wine-service respectively. The data connection from both the food- and wine-service are tightly coupled to the PostgreSQL database. We would like to decouple the food- and wine-service using data services provided by Red Hat JBoss Data Virtualization.

Get Lab Materials

For the convenience of users of the lab, we created a script and installed it on the machine in front of you. If you are in the lab environment please check if */home/student/summit-2017-dataservices* exits:

```
[student@localhost ~]$ cd ~/summit-2017-dataservices
```

if not please run the following:

```
[student@localhost ~]$ getlab
Cloning into 'summit-2017-dataservices'...
```

For those of you following along at home or own pc, just git clone the github repo from https://github.com/cvanball/summit-2017-dataservices.git.

Lab structure explained

The lab structure we are going to use during this lab is depicted and described below.

```
[student@localhost summit-2017-dataservices]$ tree -d -L 3
└── labs
        ├── lab1
        ├── lab3
        │   └── projects
        ├── lab3_ocp
        │   └── templates
        ├── lab4
        │   └── projects
        ├── lab4_ocp
        │   ├── extensions
        │   ├── templates
        │   └── vdb
        └── lab4_secure_ocp
        │   ├── extensions
        │   ├── templates
        │   └── vdb
```

Directory	Description
lab1	Sample Dockerfile
lab3	Food/wine microservice app project artifacts like sources and binaries
lab3_ocp	Artifacts for lab3, like OCP template
lab4	Food/wine microservice app project artifacts like sources, binaries and use of JDV
lab4_ocp	Artifacts for lab4 with JDV, like OCP template, JDV extensions, sql script
lab4_secure_ocp	Artifacts for lab4 with JDV, like OCP template, JDV extensions, sql script to showcase security capabilities of JDV

Lab 1 - Docker Refresh (optional)

In this lab we will explore the docker environment within Red Hat OpenShift Container Platform. If you are familiar with docker this may function as a brief refresher or proceed with Lab 2. If you are new to docker this will serve as an introduction to docker basics. Don't worry, we will progress rapidly. To get through this lab, we are going to focus on the environment itself as well as walk through some exercises with a couple of Docker images / containers to tell a complete story and point out some things that you might have to consider when containerizing your application.

This lab should be performed on **the machine in front** of you unless otherwise instructed.

The machine should have been brought up in lab. You can access that machine using username *student* and password *student*

Expected completion: 15 minutes

Topics:

- Review Docker and systemd
- Review Docker help
- Explore a Dockerfile
- Build an image
- Launch a container
- Inspect a container

Docker and systemd

Check out the systemd unit file that starts Docker on Red Hat OpenShift Container Platform and notice that it includes 3 Environment Files. These files tell Docker how the Docker daemon, storage and networking should be set up and configured. Take a look at those files too. Specifically, in the /etc/sysconfig/docker file check out the registry settings. You may find it interesting that you can ADD_REGISTRY and BLOCK_REGISTRY. Think about the different use cases for that.

Perform the following commands as root unless instructed otherwise.

```
[student@localhost ~]$ cat /usr/lib/systemd/system/docker.service
[student@localhost ~]$ cat /usr/lib/systemd/system/docker-storage-setup.service
[student@localhost ~]$ cat /etc/sysconfig/docker
[student@localhost ~]$ cat /etc/sysconfig/docker-storage
[student@localhost ~]$ cat /etc/sysconfig/docker-network
```

Now check the status of docker and make sure it is running before moving forward. It should have been brought up automatically for us by the OCP environment.

```
[student@localhost ~]$ sudo systemctl status docker
docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: dis
abled)
   Active: active (running) since Wed 2017-04-12 04:00:12 EDT; 49min ago
        Docs: http://docs.docker.com
 Main PID: 1155 (dockerd-current)
   Memory: 53.1M
   CGroup: /system.slice/docker.service
        ├─ 1155 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/li...
        ├─ 1208 /usr/bin/docker-containerd-current -l unix:///var/run/dock...
        ├─12583 /usr/bin/docker-containerd-shim-current 333a60f20aa4656a83...
        ├─12872 /usr/bin/docker-containerd-shim-current 7db3ba876d6b184019...
        ├─12932 /usr/bin/docker-containerd-shim-current 04672958b1e3c86d47...
        ├─13043 /usr/bin/docker-containerd-shim-current 5e1c65898c640522f6...
        └─13109 /usr/bin/docker-containerd-shim-current ee864ca54c2c2e7d9d...

Apr 12 04:46:12 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:12 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:12 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:15 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:16 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:16 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:18 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:18 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:19 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Apr 12 04:46:20 localhost.localdomain dockerd-current[1155]: time="2017-04-12...
Hint: Some lines were ellipsized, use -l to show in full.
```

Docker Help

Now that we see how the Docker startup process works, we should make sure we know how to get help when we need it. Run the following commands to get familiar with what is included in the Docker package as well as what is provided in the man pages. Spend some time exploring here. When you run docker info check out the storage configuration. The CDK automatically sets up storage for us by creating an LVM thin pool for use as a device mapper direct docker storage backend.

Check out the executables provided:

```
[student@localhost ~]$ rpm -ql docker | grep bin
/usr/bin/docker-containerd-current
/usr/bin/docker-containerd-shim-current
/usr/bin/docker-ctr-current
/usr/bin/docker-storage-setup
/usr/bin/dockerd-current
```

Check out the configuration files that are provided:

```
[student@localhost ~]$ rpm -qc docker
/etc/sysconfig/docker-network
/etc/sysconfig/docker-storage
/etc/sysconfig/docker-storage-setup
```

Check out the documentation that is provided:

```
[student@localhost ~]$ rpm -qd docker
/usr/share/doc/docker-1.12.6/AUTHORS
/usr/share/doc/docker-1.12.6/CHANGELOG.md

[student@localhost ~]$ docker --help
Usage: docker [OPTIONS] COMMAND [arg...]
        docker [ --help | -v | --version ]
A self-sufficient runtime for containers……
[student@localhost ~]$ docker info
Containers: 6
 Running: 5
 Paused: 0
 Stopped: 1
Images: 20
Server Version: 1.12.6

OSType: linux
Architecture: x86_64
Number of Docker Hooks: 2
CPUs: 2
Total Memory: 11.3 GiB
Name: localhost.localdomain
ID: LCWG:G2DM:GTYE:XQXP:TGFH:KEZA:BAXC:YASG:3PJ2:AJ4D:QSLD:OOUM
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://registry.access.redhat.com/v1/
Insecure Registries:
 172.30.0.0/16
 127.0.0.0/8
Registries: registry.access.redhat.com (secure), docker.io (secure)
```

Take a look at the Docker images on the system. You should see some Openshift images that are cached in the OCP environment so you can build the containers without having to wait for the container images to download from the internet.

```
[student@localhost ~]$ docker images
REPOSITORY                                                                     TAG
                IMAGE ID                CREATED                 SIZE
registry.access.redhat.com/openshift3/ose-sti-builder                          v3.4.1
.12             06af71a951dd            13 days ago             726.6 MB
registry.access.redhat.com/openshift3/ose-haproxy-router                       v3.4.1
.12             0e5da1bc1bd6            13 days ago             745.3 MB
registry.access.redhat.com/openshift3/ose-deployer                             v3.4.1
.12             77323ab89f5c            13 days ago             726.6 MB
registry.access.redhat.com/openshift3/ose-docker-registry                      v3.4.1
.12             08aaa1c313ef            13 days ago             806.5 MB
registry.access.redhat.com/openshift3/ose                                      v3.4.1
.12             14a5d3344278            13 days ago             726.6 MB
registry.access.redhat.com/openshift3/ose-pod                                  v3.4.1
.12             310eda5cf7fd            13 days ago             205 MB
registry.access.redhat.com/jboss-eap-7/eap70-openshift                         latest
                f6ca7f01844e            3 weeks ago             1.042 GB
registry.access.redhat.com/jboss-datavirt-6/datavirt63-openshift               latest
                837aa4172c2c            4 weeks ago             972.6 MB
```

Lets explore a Dockerfile

As a part of the Red Hat Software Collections offering, Red Hat provides a number of container images, which are based on the corresponding Software Collections. These include application, daemon, and database images. Here you can see in the FROM command that we are pulling a Apache Web Server base image based on RHEL 7.3 that we are going to use in this example. Containers that are being built inherit the subscriptions of the host they are running on, so you only need to register the host system. Here we are just going to explore a simple Dockerfile. The purpose for this is to have a look at some of the basic commands that are used to construct a Docker image. For this lab, we will explore a basic Apache Dockerfile and then confirm functionality.

As the student user, change directory to ~/summit-2017-dataservices/labs/lab1/ and cat out the Dockerfile

```
[student@localhost ~]$ cd ~/summit-2017-dataservices/labs/lab1
[student@localhost lab1]$ cat Dockerfile
# Pull the rhel image from the local repository
FROM registry.access.redhat.com/rhscl/httpd-24-rhel7
MAINTAINER Student <student@foo.io>

USER root

EXPOSE 80
```

After gaining access to a repository, we EXPOSE port 80, which allows traffic into the container, and then set the container to start.

Build an Image

Now that we have taken a look at the Dockerfile, lets build this image. Since it was already built previously the image is retrieved from the cache.

```
[student@localhost lab1]$ docker build -t redhat/apache .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM registry.access.redhat.com/rhscl/httpd-24-rhel7
 ---> 533e496998ca
Step 2 : MAINTAINER Student <student@foo.io>
 ---> Using cache
 ---> 2421ced729fb
Step 3 : USER root
 ---> Using cache
 ---> 0fd493ddbb4a
Step 4 : EXPOSE 80
 ---> Using cache
 ---> 3ce031e2bbc5
Successfully built 3ce031e2bbc5
```

Run the Container

Next, lets run the image and make sure it started.

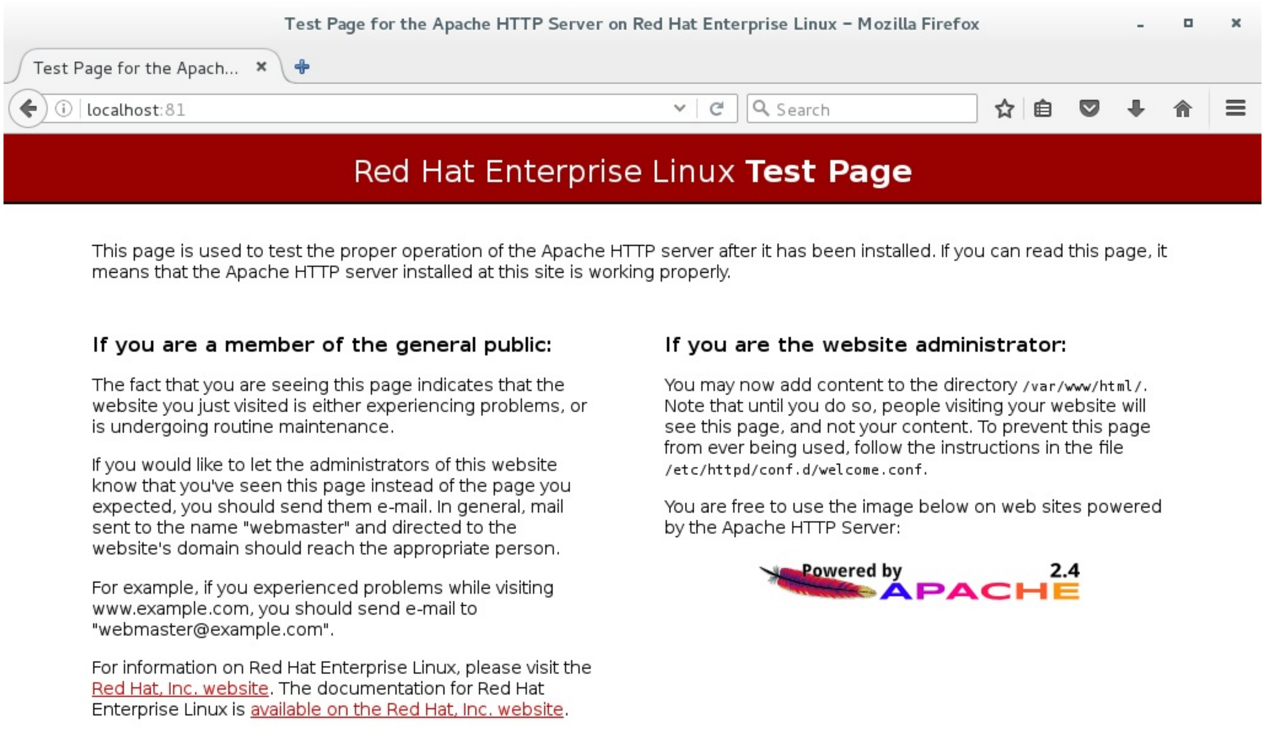
```
[student@localhost lab1]$ docker run -dt -p 81:80 --name apache redhat/apache
e9e06e014a73c7250f3c3c23d8be902fbf47db2e110d4d531c8fcadaa51a771c

[student@localhost lab1]$ docker ps
CONTAINER ID            IMAGE
        COMMAND                 CREATED                 STATUS                  PORTS
                                                NAMES
e9e06e014a73            redhat/apache
        "/usr/local/bin/run-h"   21 seconds ago          Up 18 seconds           443/tc
p, 8080/tcp, 8443/tcp, 0.0.0.0:81->80/tcp   apache
ee864ca54c2c            registry.access.redhat.com/openshift3/ose-docker-registry:v3.4
.1.12 "/bin/sh -c DOCKER_R"   19 minutes ago     Up 19 minutes
                                k8s_registry.8a800f10_docker-registry-1-j6jhx_default_
49d05df7-1ef3-11e7-90e8-5254006bc4cb_65305227
5e1c65898c64            registry.access.redhat.com/openshift3/ose-haproxy-router:v3.4.
1.12    "/usr/bin/openshift-r"   19 minutes ago          Up 19 minutes
                                        k8s_router.6a91aafa_router-1-qcf69_default_49d
f9473-1ef3-11e7-90e8-5254006bc4cb_d29bf1f4
04672958b1e3            registry.access.redhat.com/openshift3/ose-pod:v3.4.1.12
        "/pod"                  19 minutes ago          Up 19 minutes
                                        k8s_POD.b6fc0873_docker-registry-1-j6jhx_defau
lt_49d05df7-1ef3-11e7-90e8-5254006bc4cb_f5a20da2
7db3ba876d6b            registry.access.redhat.com/openshift3/ose-pod:v3.4.1.12
        "/pod"                  19 minutes ago          Up 19 minutes
                                        k8s_POD.8f3ae681_router-1-qcf69_default_49df94
73-1ef3-11e7-90e8-5254006bc4cb_7697cd22
333a60f20aa4            registry.access.redhat.com/openshift3/ose:v3.4.1.12
        "/usr/bin/openshift s"   20 minutes ago          Up 20 minutes
```

Here we are using a few switches to configure the running container the way we want it. We are running a -dt to run in detached mode with a pseudo TTY. Next we are mapping a port from the host to the container. We are being explicit here. We have told Docker to map port 81 on the host to port 80 in the container. Now, we could have let Docker handle the host side port mapping dynamically by passing a -P or -p 80, in which case Docker would have randomly assigned a port to the container. Finally, we passed in the name of the image that we built earlier.

Okay, lets make sure we can access the web server.

```
[student@localhost lab1]$ curl http://localhost:81
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xh
tml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
        <head>
                <title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linu
x</title>
.....
```

Start Firefox Web Browser using Applications→Favorites→Firefox Web Browser and point the URL to http://localhost:81 and should see a similar screen as depicted below


Test Page for the Apache HTTP Server on Red Hat Enterprise Linux – Mozilla Firefox
Test Page for the Apach...
localhost:81
Search

Red Hat Enterprise Linux **Test Page**

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.

If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting www.example.com, you should send e-mail to "webmaster@example.com".

For information on Red Hat Enterprise Linux, please visit the Red Hat, Inc. website. The documentation for Red Hat Enterprise Linux is available on the Red Hat, Inc. website.

If you are the website administrator:

You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page, and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

You are free to use the image below on web sites powered by the Apache HTTP Server:

Now that we have built an image, launched a container and confirmed that it is running, let's do some further inspection of the container. We should take a look at the container IP address. Let's use docker inspect to do that.

Time to Inspect

```
[student@localhost lab1]$ docker inspect apache
```

We can see that this gives us quite a bit of information in json format. We can scroll around and find the IP address, it will be towards the bottom.

```
        "Networks": {
                "bridge": {
                        "IPAMConfig": null,
                        "Links": null,
                        "Aliases": null,
                        "NetworkID": "4c6c77ea7038a36ca39f11d4cfb80cb0e502d975f87d33ba
47bccccd0c6c168d",
                        "EndpointID": "251efeefa42411516842d8d4ca230759d8a63ef6c670a15
bc4f4e0ef3faa95ce",
                        "Gateway": "172.17.0.1",
                        "IPAddress": "172.17.0.3",
                        "IPPrefixLen": 16,
                        "IPv6Gateway": "",
                        "GlobalIPv6Address": "",
                        "GlobalIPv6PrefixLen": 0,
                        "MacAddress": "02:42:ac:11:00:03"
                }
                }
```

Let's be more explicit with our docker inspect

```
[student@localhost lab1]$ docker inspect --format '\{\{ .NetworkSettings.IPAddress \}\
}' apache
172.17.0.3
```

You should see the IP address that was assigned to the container.

We can apply the same filter to any value in the json output. Try a few different ones.

Now lets look inside the container and see what that environment looks like. We first need to get the PID of the container so we can attach to the PID namespace with nsenter. After we have the PID, go ahead and enter the namespaces of the container substituting the PID on your container for the one listed below. Take a look at the man page to understand all the flags we are passing to nsenter.

```
[student@localhost lab1]$ docker inspect --format '\{\{ .State.Pid \}\}' apache
15860

[student@localhost lab1]$ man nsenter
NAME
        nsenter - run program with namespaces of other processes
…...

[student@localhost lab1]$ sudo nsenter -m -u -n -i -p -t 15860
[sudo] password for student:
[root@e9e06e014a73 /]#
```

Now run some commands and explore the environment. Remember, we are in a slimmed down container at this point - this is by design. You may find yourself restricted.

```
[root@e9e06e014a73 /]# ps aux
USER     PID %CPU %MEM    VSZ    RSS TTY    STAT START    TIME COMMAND
root            1  0.0   0.0 258144   7508 ?      Ss+  09:05    0:00 httpd -DFOREGROUND
apache          20  0.0   0.0 266472   4712 ?     Sl+  09:05    0:00 httpd -DFOREGROUND
apache          21  0.0   0.0 266472   4196 ?     Sl+  09:05    0:00 httpd -DFOREGROUND
apache          22  0.0   0.0 266472   4200 ?     Sl+  09:05    0:00 httpd -DFOREGROUND
apache          23  0.0   0.0 266472   4712 ?     Sl+  09:05    0:00 httpd -DFOREGROUND
apache          26  0.0   0.0 266472   4196 ?     Sl+  09:05    0:00 httpd -DFOREGROUND
apache          30  0.0   0.0 266472   4196 ?     Sl+  09:10    0:00 httpd -DFOREGROUND
root            32  0.0   0.0  13368   2020 ?     S       09:18    0:00 -bash
root            46  0.0   0.0  49040   1836 ?     R+   09:18    0:00 ps aux


[root@e9e06e014a73 /]# ls /bin
[                       findmnt                 msgconv                 sim_client
a2p                     find-repos-of-install  msgen                   size
aclocal                 fipscheck               msgexec                 skill

[root@e9e06e014a73 /]# cat /etc/hosts
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
fe00::0    ip6-localnet
ff00::0    ip6-mcastprefix
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters
172.17.0.3    e9e06e014a73

[root@e9e06e014a73 /]# ip addr
-bash: ip: command not found
```

Well, what can we do? You can install software into this container.

```
[root@e9e06e014a73 /]# yum -y install iproute
[root@e9e06e014a73 /]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
16: eth0@if17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
        link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 172.17.0.3/16 scope global eth0
        valid_lft forever preferred_lft forever
        inet6 fe80::42:acff:fe11:3/64 scope link
        valid_lft forever preferred_lft forever
```

Exit the container namespace with CTRL+d or exit.

In addition to using nsenter to enter the namespace of your container, you can also execute commands in that namespace with docker exec.

```
$ docker exec <container-name OR container-id> <cmd>
[student@localhost lab1]$ docker exec apache pwd
/opt/app-root/src
```

Whew, so we do have some options. Now, remember that this lab is all about containerizing your existing apps. You will need some of the tools listed above to go through the process of containerizing your apps. Troubleshooting problems when you are in a container is going to be something that you get very familiar with.

Before we move on to the next section let's clean up the apache container so we don't have it hanging around.

```
[student@localhost lab1]$ docker rm -f apache
Apache
[student@localhost lab1]$ cd $HOME
[student@localhost ~]$
```

Congratulations!!!!! You have completed this lab.

Lab 2 - OpenShift Command Line Interface (CLI)

RED HAT® OPENSHIFT Container Platform

OpenShift Container Platform ships with a feature rich web console as well as a command line interface tool (CLI) to provide users with a nice interface to work with applications deployed to the platform. The OpenShift CLI is a single executable written in the Go programming language and is available for the following operating systems:

- Microsoft Windows
- macOS
- Linux

This lab should be performed on **the machine in front of you** unless otherwise instructed.

Expected completion: 15 minutes

Topics:

- Downloading OpenShift CLI
- Extracting the OpenShift CLI
- Verify OpenShift CLI installation
- Useful OpenShift CLI commands

Tip	First three topics are already performed on **the machine in front of you**. If you are using this machine you can proceed with paragraph Useful OpenShift CLI commands.

Downloading OpenShift CLI

During this lab, we are going describe how to setup the OpenShift CLI tool and add them to our operating system PATH environment variables so the executable is accessible from any directory on the command line. For your convenience this is explained and for your reference but OpenShift CLI tool is already installed in the OCP environment in front of you.

Tip	This is for your reference. The environment in front of you is already setup with the oc command. You can proceed with paragraph Useful CLI commands

The first thing we want to do is download the correct executable for your operating system as linked below: OpenShift CLI can be downloaded here from the Red Hat's Customer Portal: https://access.redhat.com/downloads/content/290

Once the file has been downloaded, you will need to extract the contents as it is a compressed archive. I would suggest saving this file to the following directories:

Windows

```
C:\> cd C:\OpenShift
```

macOS

```
$ cd ~/bin
```

Linux

```
$ cd ~/bin
```

Extracting the OpenShift CLI

Once you have the tools downloaded, you will need to extract the contents:

Windows

In order to extract a zip archive on windows, you will need a zip utility installed on your system. With newer versions of windows (greater than XP), this is provided by the operating system. Just right click on the downloaded file using file explorer and select to extract the contents.

macOS

Open up a terminal window and change to the directory where you downloaded the file. Once you are in the directory, enter in the following command:

```
$ tar zxvf oc-3.4.1.12-macosx.tar.gz
```

Linux

Open up a terminal window and change to the directory where you downloaded the file. Once you are in the directory, enter in the following command:

```
$ tar zxvf oc-3.4.1.12-linux.tar.gz
```

Adding oc to your PATH

Windows

Because changing your PATH on windows varies by version of the operating system, we will not list each operating system here. However, the general workflow is right click on your computer name inside of the file explorer. Select Advanced system settings. I guess changing your PATH is considered an advanced task? :) Click on the advanced tab, and then finally click on Environment variables. Once the new dialog opens, select the Path variable and add ";C:\OpenShift" at the end. For an easy way out, you could always just copy it to C:\Windows or a directory you know is already on your path.

macOS

```
$ export PATH=$PATH:~/bin
```

Linux

```
$ export PATH=$PATH:~/bin
```

Verify

At this point, we should have the oc tool available for use. Let's test this out by printing the version of the oc command:

```
[student@localhost ~]$ whereis oc
oc: /usr/bin/oc /usr/share/man/man1/oc.1.gz
[student@localhost ~]$ oc version
oc v3.4.1.12
kubernetes v1.4.0+776c994
features: Basic-Auth GSSAPI Kerberos SPNEGO

Server https://192.168.122.45:8443
openshift v3.4.1.12
kubernetes v1.4.0+776c994
```

If you get an error message, the PATH is not updated correctly. If you need help, raise your hand and the instructor will assist.

Tip	The OCP environment is automatically started as a system daemon service called oc-cluster. In other words no need to stop or start the OCP environment. this is for your reference.

You can start the OCP environment using the following command:

```
[student@localhost ~]$ sudo systemctl start oc-cluster
```

You can stop the OCP environment using the following command:

```
[student@localhost ~]$ sudo systemctl stop oc-cluster
```

Check the status of the OCP environment using the following command:

```
[student@localhost ~]$ sudo systemctl status oc-cluster
oc-cluster.service - OpenShift Cluster Service
   Loaded: loaded (/etc/systemd/system/oc-cluster.service; enabled; vendor preset: dis
abled)
   Active: active (exited) since Wed 2017-04-12 04:01:12 EDT; 39min ago
  Process: 2558 ExecStart=/usr/local/bin/oc-cluster-up.sh (code=exited, status=0/SUCCE
SS)
 Main PID: 2558 (code=exited, status=0/SUCCESS)
   Memory: 0B
   CGroup: /system.slice/oc-cluster.service

Apr 12 04:01:11 localhost.localdomain oc-cluster-up.sh[2558]: Waiting for API...
Apr 12 04:01:11 localhost.localdomain oc-cluster-up.sh[2558]: OpenShift serve...
Apr 12 04:01:12 localhost.localdomain oc-cluster-up.sh[2558]: -- Removing tem...
Apr 12 04:01:12 localhost.localdomain oc-cluster-up.sh[2558]: -- Server Infor...
Apr 12 04:01:12 localhost.localdomain oc-cluster-up.sh[2558]: OpenShift serve...
Apr 12 04:01:12 localhost.localdomain oc-cluster-up.sh[2558]: The server is a...
Apr 12 04:01:12 localhost.localdomain oc-cluster-up.sh[2558]: https://192.168...
Apr 12 04:01:12 localhost.localdomain oc-cluster-up.sh[2558]: To login as adm...
Apr 12 04:01:12 localhost.localdomain oc-cluster-up.sh[2558]: oc login -u sys...
Apr 12 04:01:12 localhost.localdomain systemd[1]: Started OpenShift Cluster S...
Hint: Some lines were ellipsized, use -l to show in full.
```

Useful OpenShift CLI commands

The Openshift CLI allows interaction with the various objects that are managed by OpenShift Container Platform. Many common oc operations are invoked using the following syntax:

```
[student@localhost ~]$ oc <action> <object_type> <object_name>
```

Where

- An <action> to perform, such as get or describe.
- The <object_type> to perform the action on, such as service or the abbreviated svc.
- The <object_name> of the specified <object_type>.

The student user is sudoer. They can execute commands with '--as=system:admin'.

Now, lets work with the OCP environment to showcase some useful CLI commands:

Openshift client help:

```
[student@localhost ~]$ oc help
```

Log in to the OCP server as *admin* user:

```
[student@localhost ~]$ oc login -u system:admin
Logged into "https://192.168.122.45:8443" as "system:admin" using existing credentials
.

You have access to the following projects and can switch between them with 'oc project
 <projectname>':

        default
        kube-system
        * myproject
        openshift
        openshift-infra

Using project "myproject".
```

Check who is logged in:

```
[student@localhost ~]$ oc whoami
system:admin
```

Display one or many resources using:

```
[student@localhost ~]$ oc get
[(-o|--output=)json|yaml|wide|custom-columns=...|custom-columns-file=...|go-template=.
..|go-template-file=...|jsonpath=...|jsonpath-file=...]
(TYPE [NAME | -l label] | TYPE/NAME ...) [flags] [options]
```

Possible resources include builds, buildConfigs, services, pods, etc. To see a list of common resources, use 'oc get'. Some resources may omit advanced details that you can see with '-o wide'. If you want an even more detailed view, use 'oc describe'.

List all pods in ps output format

```
[student@localhost ~]$ oc get pods
```

List all pods and show more details about them

```
[student@localhost ~]$ oc get -o wide pods
```

List a single pod in JSON output format.

```
[student@localhost ~]$ oc get -o json pod apache
```

List a single replication controller with specified ID in ps output format.

```
[student@localhost ~]$ oc get rc apache
```

List build config with specified ID in ps output format.

```
[student@localhost ~]$ oc get bc apache
```

List deployment config with specified ID in ps output format.

```
[student@localhost ~]$ oc get dc apache
```

End the current session.

```
[student@localhost ~]$ oc logout
```

Log in in OCP as developer user.

```
[student@localhost ~]$ oc login -u developer -p developer
Login successful.

You have one project on this server: "myproject"

Using project "myproject".
[student@localhost ~]$ oc get projects
NAME           DISPLAY NAME   STATUS
myproject   My Project   Active
```

Check who is logged in.

```
[student@localhost ~]$ oc whoami
Developer
```

Create new project.

```
[student@localhost ~]$ oc new-project <project-name>
```

Switch to another project.

```
[student@localhost ~]$ oc project <project-name>
```

Get current status of OCP environment.

```
[student@localhost ~]$ sudo systemctl status oc-cluster
```

Start the OCP environment.

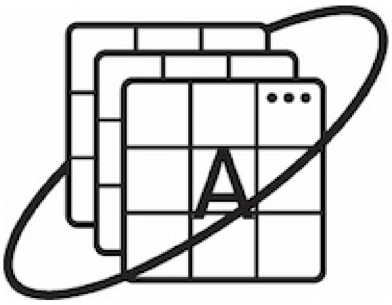
```
[student@localhost ~]$ sudo systemctl start oc-cluster
```

Stop the OCP environment.

```
[student@localhost ~]$ sudo systemctl stop oc-cluster
```

Congratulations!!!!! You have completed this lab.

Lab 3 - Analyzing a microservices application



Typically, it is best to break down services into the simplest components and then containerize each of them independently. However, when initially migrating an application it is not always easy to break it up into little pieces but you can start with big containers and work towards breaking them into smaller pieces.

In this lab we will create a project which contains multiple container images comprised of multiple services since our application is already a microservices application. In lab 3 we will describe and run the microservices application which is already split up into more manageable pieces.

This lab should be performed on **the machine in front of you** unless otherwise instructed.

Expected completion: 20-30 minutes

Topics:

- Prerequisites
- Overview microservices application
- Exploring OpenShift template
- Setup OCP environment based on pre-built OpenShift template
- Exploring the running containers
- Connecting to the application

Prerequisites

To check if OpenShift Container Platform (OCP) is running execute:

source,bash]

```
[student@localhost ~]$ sudo systemctl status oc-cluster
```

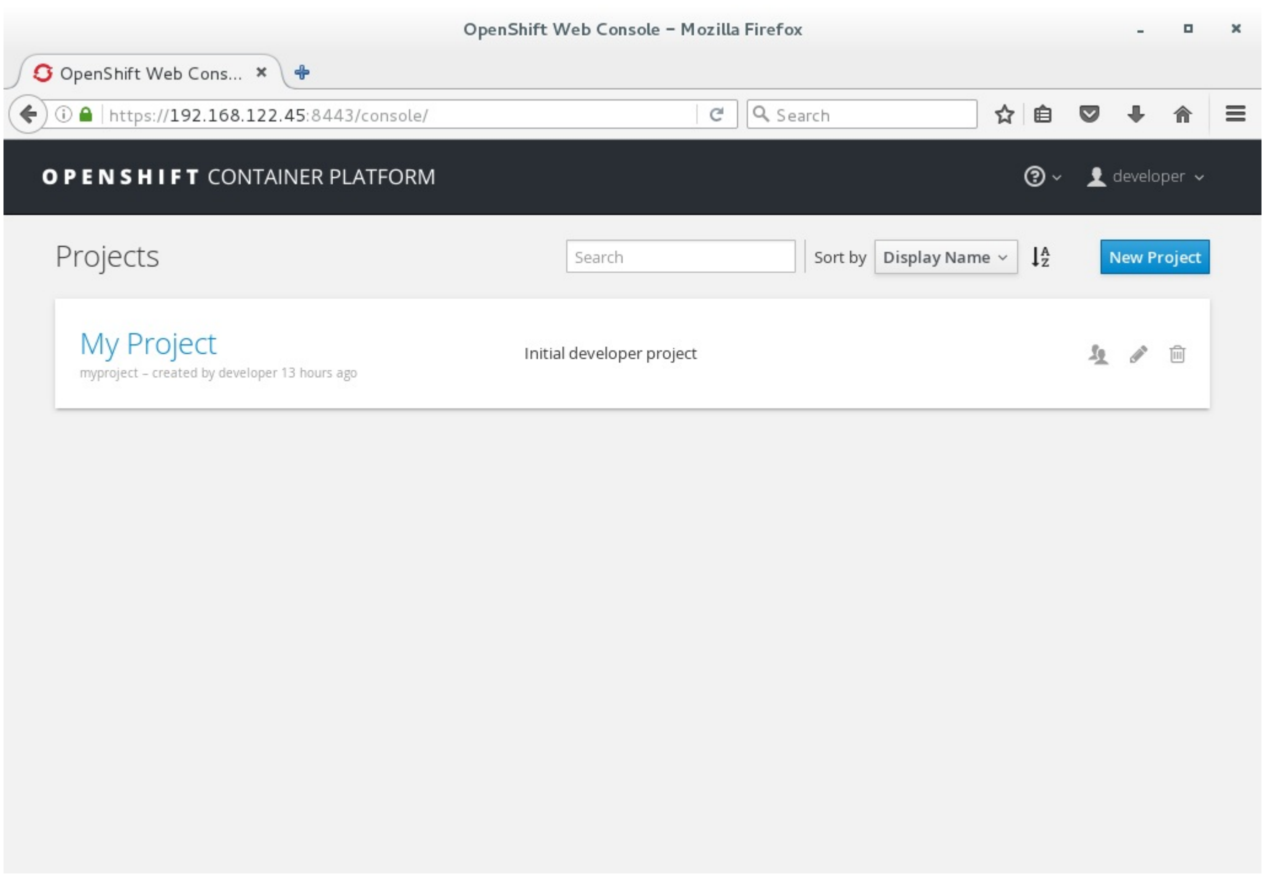
If you get no cluster running start ODP with

```
[student@localhost ~]$ sudo systemctl start oc-cluster
```

Now log in to OpenShift with username *developer* and password *developer* :

```
[student@localhost ~]$ oc login -u developer -p developer
Login successful.

You have one project on this server: "myproject"

Using project "myproject".
```

You are now logged in to OpenShift and are using the *myproject* project. You can also view the OpenShift web console by using the same credentials to log in using Firefox Web Browser as depicted below.


OpenShift Web Console – Mozilla Firefox
OpenShift Web Cons...
https://192.168.122.45:8443/console/
Search
OPENSHIFT CONTAINER PLATFORM
developer
Projects
Search
Sort by
Display Name
New Project
My Project
myproject – created by developer 13 hours ago
Initial developer project

First we are going to create a new project called *lab3*.

```
[student@localhost ~]$ oc new-project lab3
Now using project "lab3" on server "https://192.168.122.45:8443".

You can add applications to this project with the 'new-app' command. For example, try:

oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.
```

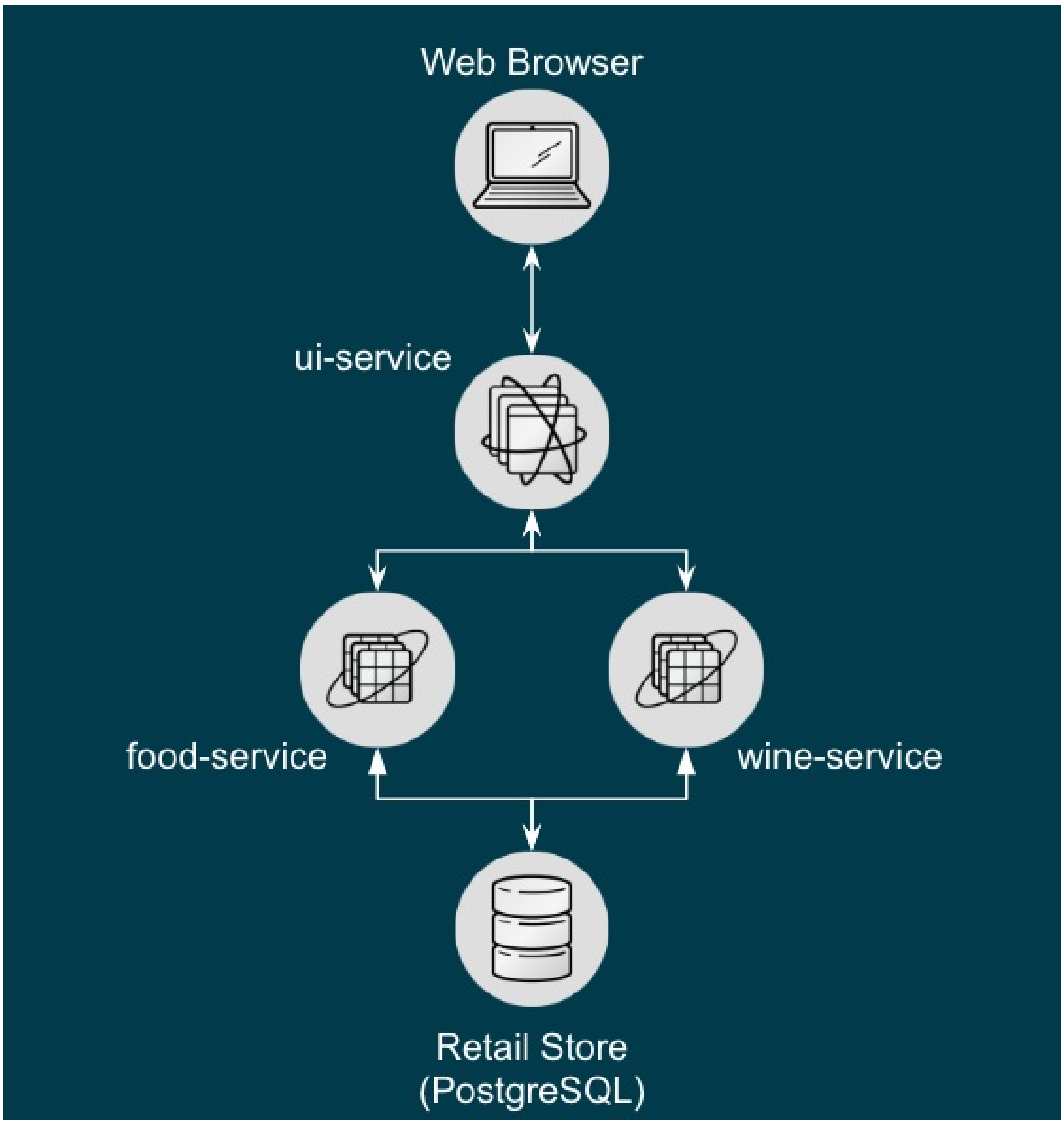
Overview microservices application

The microservices application we are going to use in lab 3 is a simple AngularJS (https://angularjs.org/) and Spring (http://spring.io/) application using food and wine datas.

Our initial OpenShift template will be using the following four services:

- ui-service: AngularJS and Spring application
- food-service: Backend service providing food data
- wine-service: Backend service providing wine data
- PostgreSQL service serving the food and wine data

The food- and wine-service are connecting directly to the PostgreSQL environment. The illustration below depicts the starting point of our microservices application.



In the next steps, we are going to setup and run the above environment in OCP. Exploring Microservices Application Familiarize yourself with the initial Microservices Application by opening a code editor using the following steps:

```
[student@localhost ~]$ cd ~/summit-2017-dataservices/labs/lab3/projects
[student@localhost ~] code .
```

Dive into the code

Caution	In the ui-service, food-services and wine-services directory you'll find all maven based java projects. To limit the use of wifi during this Summit lab all the projects are prebuilt and the WAR files containing the ui and business logic are already available for your convenience, see target directory. I.e. /home/student/summit-2017-dataservices/labs/lab3/projects/ui-service/target/ROOT.war /home/student/summit-2017-dataservices/labs/lab3/projects/food-service/target/ROOT.war /home/student/summit-2017-dataservices/labs/lab3/projects/wine-service/target/ROOT.war Same applies for the projects used in lab 4.

Exploring OpenShift template

An OpenShift template describes a set of objects that can be parameterized and processed to produce a list of objects for creation by OpenShift Container Platform. The objects to create can include anything that users have permission to create within a project, for example services, build configurations, and deployment configurations. A template may also define a set of labels to apply to every object defined in the template. See the template guide for details about creating and using templates. Check out directory *~/summit-2017-dataservices/labs/lab3_ocp/templates*.

```
[student@localhost ~]$ cd ~/summit-2017-dataservices/labs/lab3_ocp
```

In this directory you will find the OpenShift template containing the configuration needed to setup a complete OCP environment for lab3. Take a look at lab3-template.json using a pre-installed code editor.

```
[student@localhost lab3_ocp]$ code templates/lab3-template.json
```

Setup OCP environment based on pre-built OpenShift template

If you have a JSON or YAML file that defines a template, for example in our case we are using lab3-template.json, you can upload the template to projects using the CLI. This saves the template to the project for repeated use by any user with appropriate access to that project. See for more instructions on writing your own templates. Furthermore you can use the CLI to process templates and use the configuration that is generated to create objects as shown below.

```
[student@localhost lab3_ocp]$ oc process -f templates/lab3-template.json | oc create -f
 -
service "wineapp-wine-service" created
service "wineapp-food-service" created
service "wineapp-postgresql" created
service "wineapp-ui" created
route "wineapp-route" created
imagestream "wineapp-ui" created
imagestream "wineapp-food-service" created
imagestream "wineapp-wine-service" created
buildconfig "wineapp-food-service" created
buildconfig "wineapp-ui" created
buildconfig "wineapp-wine-service" created
deploymentconfig "wineapp-food-service" created
deploymentconfig "wineapp-postgresql" created
deploymentconfig "wineapp-ui" created
deploymentconfig "wineapp-wine-service" created
```

As mentioned earlier we would like to minimize the use of wifi during this Summit lab. Typically the template will build the pod downloading the source code from a github repository. Since we have already built our projects using maven (mvn clean package -DskipTests), we can use binary deployment with following command:

```
[student@localhost lab3_ocp]$ cd ~/summit-2017-dataservices/labs/lab3
[student@localhost lab3]$ oc start-build <build config> <options>
```

For more information How Builds works, see the OpenShift Cotainer Platform documentation: https://docs.openshift.com/container-platform/3.4/dev_guide/builds/index.html

Get all available build configs

```
[student@localhost lab3_ocp]$ oc get bc
NAME TYPE FROM LATEST
wineapp-food-service Source     Binary  0
wineapp-ui Source       Binary  0
wineapp-wine-service Source     Binary  0
```

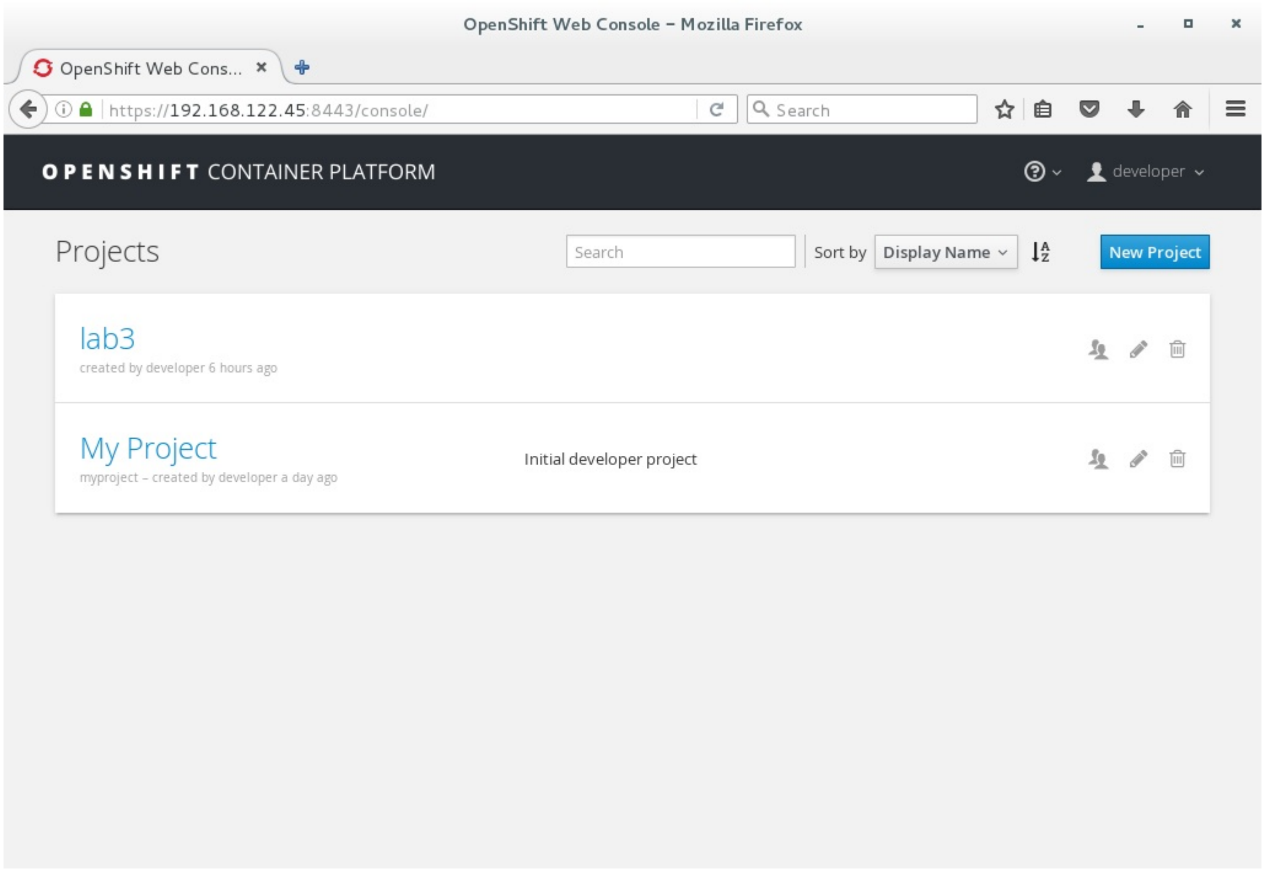
Now start the binary builds using the following commands:

```
[student@localhost lab3]$ cd ~/summit-2017-dataservices/labs/lab3/projects

[student@localhost projects]$ oc start-build wineapp-food-service --from-dir=food-serv
ice/deployments
Uploading directory "food-service/deployments" as binary input for the build ...
build "wineapp-food-service-1" started

[student@localhost projects]$ oc start-build wineapp-wine-service --from-dir=wine-serv
ice/deployments
Uploading directory "wine-service/deployments" as binary input for the build ...
build "wineapp-wine-service-1" started

[student@localhost projects]$ oc start-build wineapp-ui --from-dir=ui-service/deployme
nts
Uploading directory "ui-service/deployments" as binary input for the build ...
build "wineapp-ui-1" started
```

Now the containers will be built and deployed. Let’s see how it looks like in the OpenShift Web Console.

Exploring the running the containers

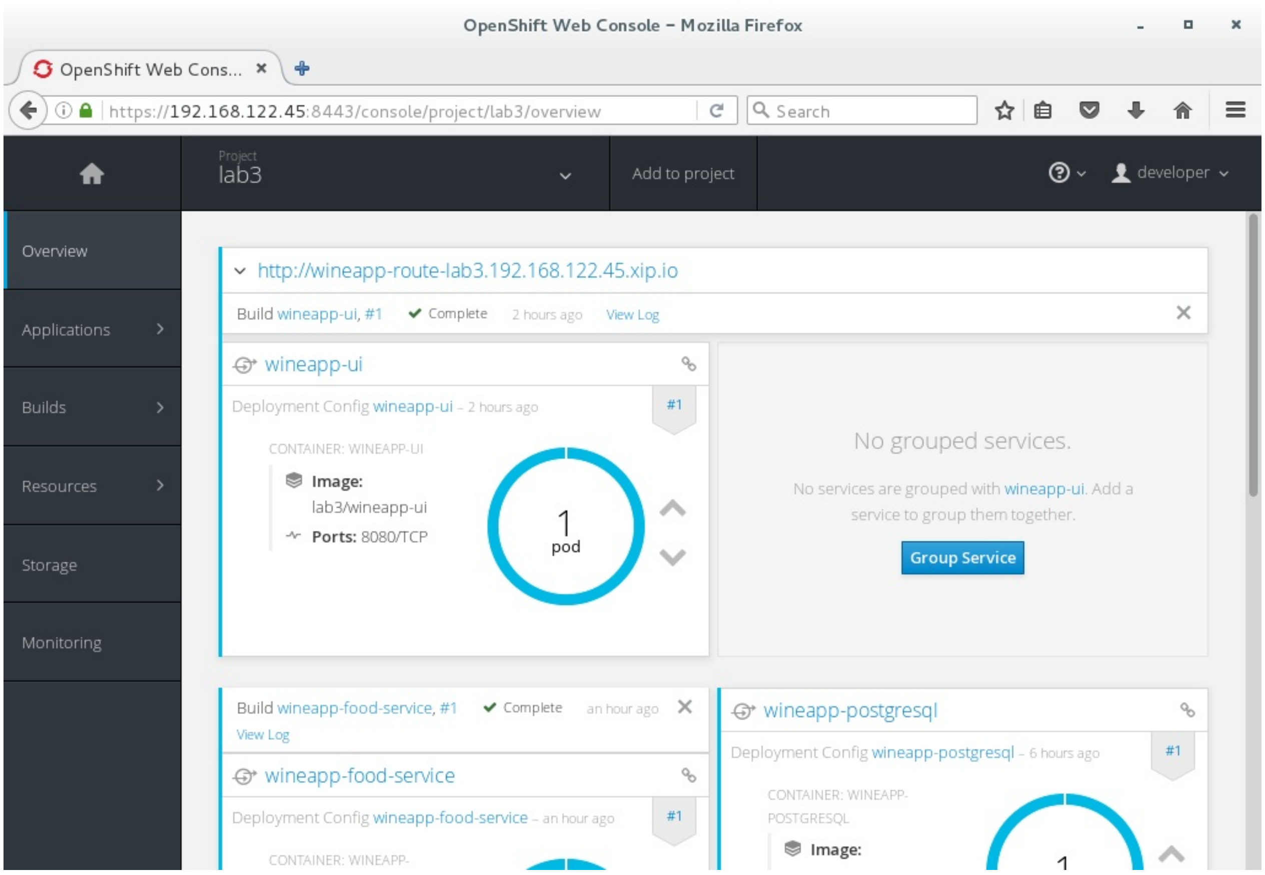
Login into the OpenShift Web Console and login with username developer


OpenShift Web Console – Mozilla Firefox
OpenShift Web Cons...
https://192.168.122.45:8443/console/
Search
OPENSHIFT CONTAINER PLATFORM
developer
Projects
Search
Sort by
Display Name
New Project

lab3
created by developer 6 hours ago

My Project
myproject – created by developer a day ago
Initial developer project

Click on project *lab3* and the lab3 overview page should appear as depicted below.



No services are grouped with wineapp-ui. Add a service to group them together.

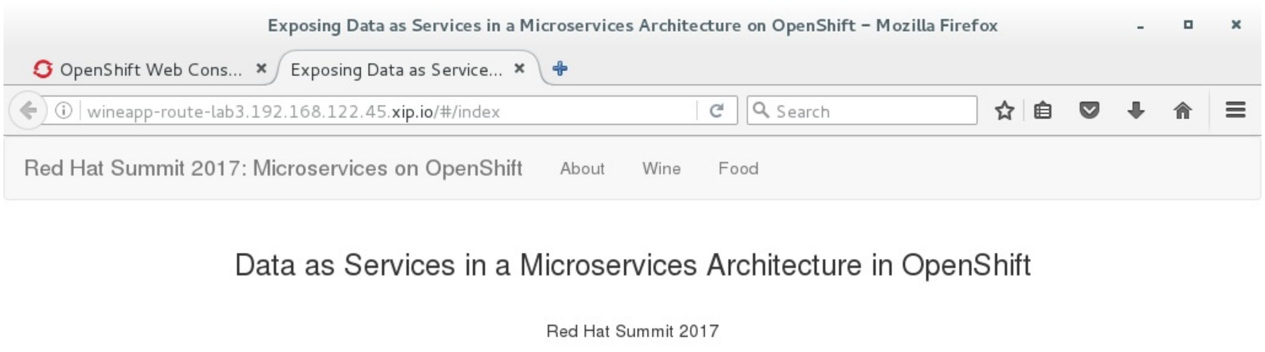
Scroll down and use the menu options to familiarize with the OpenShift lab3 containers.

Connecting to the application

An OpenShift Container Platform route exposes a service at a host name, like *www.example.com*, so that external clients can reach it by name.

In the example depicted in screenshots before we can see a route defined in *lab3* project which expose the webui of our food and wine microservices application at url: http://wineapp-route-lab3.192.168.122.45.xip.io

Another way to get the routes is to navigate to the Browse → Routes page. Click on the URL of the route and you should be redirected to the food and wine microservices application as depicted below.


Exposing Data as Services in a Microservices Architecture on OpenShift – Mozilla Firefox
OpenShift Web Cons...
Exposing Data as Service...
wineapp-route-lab3.192.168.122.45.xip.io/#/index
Search
Red Hat Summit 2017: Microservices on OpenShift
About
Wine
Food

Data as Services in a Microservices Architecture in OpenShift

Red Hat Summit 2017

The wineapp microservices application showcases a web application providing create and read functionality. Click on Wine and/or Food and see if existing data is retrieved from the PostgreSQL database. Furthermore try to add your favorite wine and food using the application.

Cleanup lab 3

Delete project using OpenShift CLI

```
[student@localhost projects]$ oc delete project lab3
```

Remove the docker images To remove the created docker images during this lab you can do

```
[student@localhost projects]$ docker images | grep wineapp
REPOSITORY TAG IMAGE ID CREATED SIZE
172.30.1.1:5000/lab3/wineapp-food-service latest 1af952bac3a7 About an hour ago 877.8
MB
172.30.1.1:5000/lab3/wineapp-wine-service latest d934bcff78c4 About an hour ago 873 MB
172.30.1.1:5000/lab3/wineapp-ui latest 3db40e59a493 About an hour ago 775.9 MB
```

You can remove the image one by one using:

```
[student@localhost projects]$ docker rmi <image id>
```

For you convenience we have a script called *rmlab3* available which removes all images with wineapp in the name:

```
[student@localhost projects]$ rmlab3
```

Congratulations!!!!! You have completed this lab.

Lab 4 - Changing the microservices application using data services

In this lab you will change the microservices application to use data services provided by Red Hat JBoss Data Virtualization. In this process we explore the changes we need to make in order to utilize the capabilities of Red Hat JBoss Data Virtualization.

This lab should be performed on **the machine in front of you** unless otherwise instructed.

Expected completion: 20-30 minutes

Topics:

- Prerequisites
- Overview Microservices Application using data services
- Exploring OpenShift template
- Setup environment based on pre-built OpenShift template
- Exploring the running containers
- Connecting to the application

Prerequisites

Log in to OpenShift with username *developer* and password *developer* :

```
[student@localhost ~]$ oc login -u developer -p developer
Login successful.

You have one project on this server: "myproject"
```

You are now logged in to OpenShift and are using the *myproject* project. First we are going to create a new project called *lab4*.

```
[student@localhost ~]$ oc new-project lab4
Now using project "lab4" on server "https://192.168.122.45:8443".

You can add applications to this project with the 'new-app' command. For example, try:

oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.
```

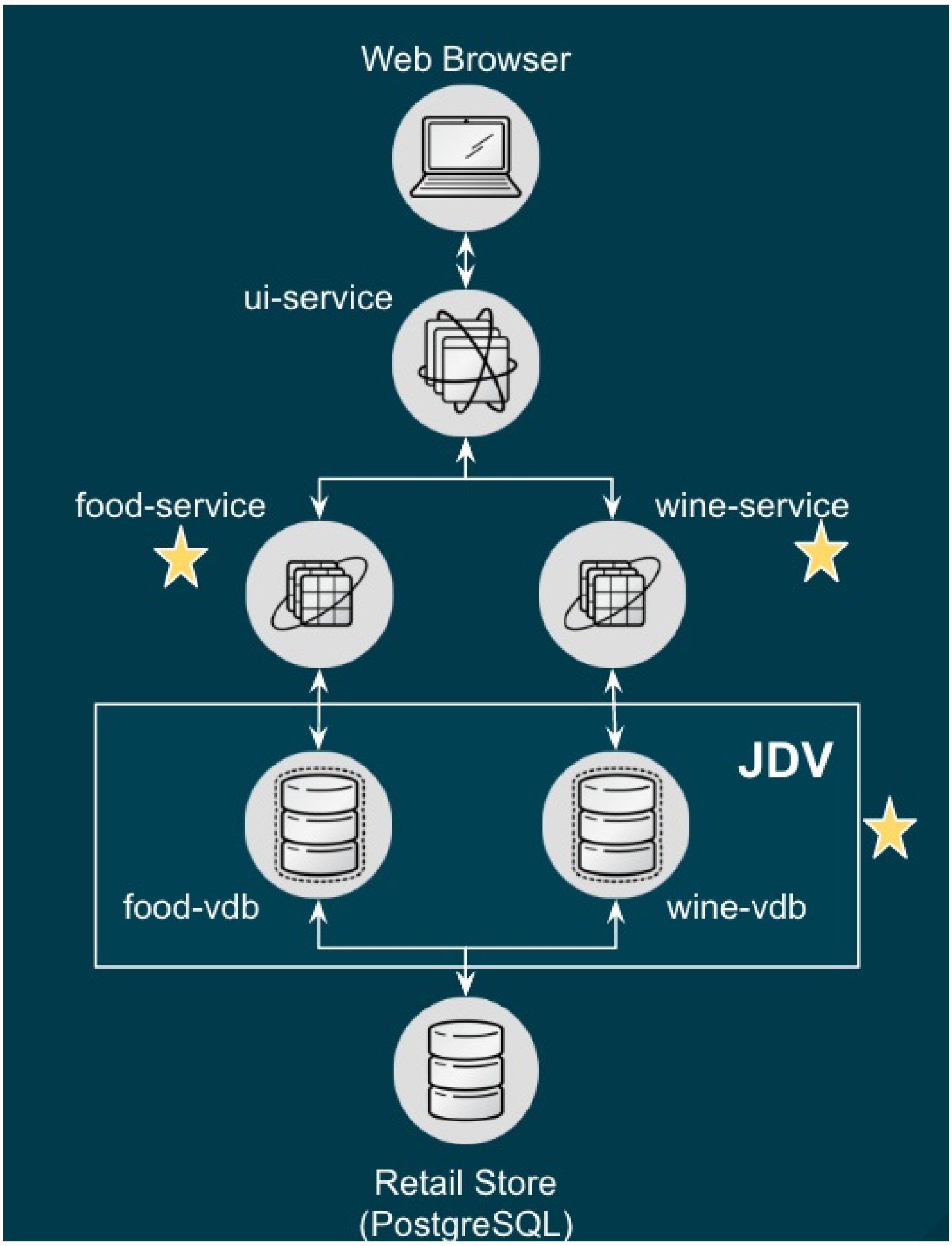
Overview Microservices Application using data services

The microservices application we are going to use in this lab is a simple AngularJS (https://angularjs.org/) and Spring (http://spring.io/) application.

The OpenShift template will be using the following five services:

- ui-service: AngularJS and Spring application
- food-service: Backend service providing food data
- wine-service: Backend service providing wine data
- jdv-service: Red Hat JBoss Data Virtualization (JDV) providing an abstraction layer between the food-service, wine-service and the underlying PostgreSQL database
- PostgreSQL service serving the food and wine data

The food- and wine-service are connecting to the JDV environment and the JDV environment will connect to the PostgreSQL service. The illustration below depicts the end result of our microservices application using JDV running on OCP.



Exploring OpenShift template

Check out directory ~/summit-2017-dataservices/labs/lab4_ocp/templates.

```
[student@localhost lab4_ocp]$ cd ~/summit-2017-dataservices/labs/lab4_ocp
```

In this directory you will find the OpenShift template containing the configuration needed to setup the complete lab4 environment. Take a look at lab4-template.json using a pre-installed code editor.

```
[student@localhost lab4_ocp]$ code templates/lab4-template.json
```

Compare lab4-template.json against lab3-template.json and see the differences.

```
[student@localhost lab4_ocp]$ code -d templates/lab4-template.json ~/summit-2017-datas
ervices/labs/lab3_ocp/templates/lab3-template.json
```

There might be more changes than your initial thoughts, take a look at the Route, Services, BuildConfig and DeploymentConfig sections especially the sections with jdv and jdv-ext in there.

Setup environment based on pre-built OpenShift template

Files for runtime artifacts are passed to the JDV for OpenShift image using the OpenShift secret mechanism. This includes the environment files for the data sources and resource adapters, as well as any additional data files. These files need to be present locally so we have to create secrets for them.

```
[student@localhost lab4_ocp]$ oc create -f extensions/datavirt-app-secret.yaml
[student@localhost lab4_ocp]$ oc secrets new datavirt-app-config extensions/datasource
s.properties
[student@localhost lab4_ocp]$ oc adm policy add-role-to-user view system:serviceaccoun
t:lab4:datavirt-service-account
```

If you have a JSON or YAML file that defines a template, for example in our case we are using lab4-template.json, you can upload the template to projects using the CLI. This saves the template to the project for repeated use by any user with appropriate access to that project. See for more instructions on writing your own templates. Furthermore you can use the CLI to process templates and use the configuration that is generated to create objects as shown below.

```
[student@localhost lab4_ocp]$ oc process -f templates/lab4-template.json | oc create -f
 -
buildconfig "rhapp-food-service" created
buildconfig "rhapp-jdv" created
buildconfig "rhapp-jdv-ext" created
buildconfig "rhapp-ui" created
buildconfig "rhapp-wine-service" created
imagestream "rhapp-food-service" created
imagestream "rhapp-jdv" created
imagestream "rhapp-jdv-ext" created
imagestream "rhapp-ui" created
imagestream "rhapp-wine-service" created
deploymentconfig "rhapp-food-service" created
deploymentconfig "rhapp-jdv" created
deploymentconfig "rhapp-postgresql" created
deploymentconfig "rhapp-ui" created
deploymentconfig "rhapp-wine-service" created
route "jdbc-rhapp-jdv" created
route "secure-rhapp-jdv" created
route "rhapp-jdv" created
route "rhapp-ui" created
service "rhapp-food-service" created
service "rhapp-jdv" created
service "rhapp-postgresql" created
service "rhapp-ui" created
service "rhapp-wine-service" created
```

As mentioned earlier we would like to minimize the use of wifi during this Summit lab. Typically the template will be built the pod downloading the source code from a github repository. Since we already have built our projects using maven (mvn clean package -DskipTests) we can do binary deployment with following command:

```
[student@localhost projects]$ oc start-build <build config>
```

For more information on How OpenShift Builds works, see the OpenShift Cotainer Platform documentation: https://docs.openshift.com/container-platform/3.4/dev_guide/builds/index.html

Get all available build configs.

```
[student@localhost lab4_ocp]$ oc get bc
NAME TYPE FROM LATEST
rhapp-food-service Source Binary 0
rhapp-jdv Source Binary 0
rhapp-jdv-ext Docker Binary 1
rhapp-ui Source Binary 0
rhapp-wine-service Source Binary 0
```

Now start the binary builds using the following commands:

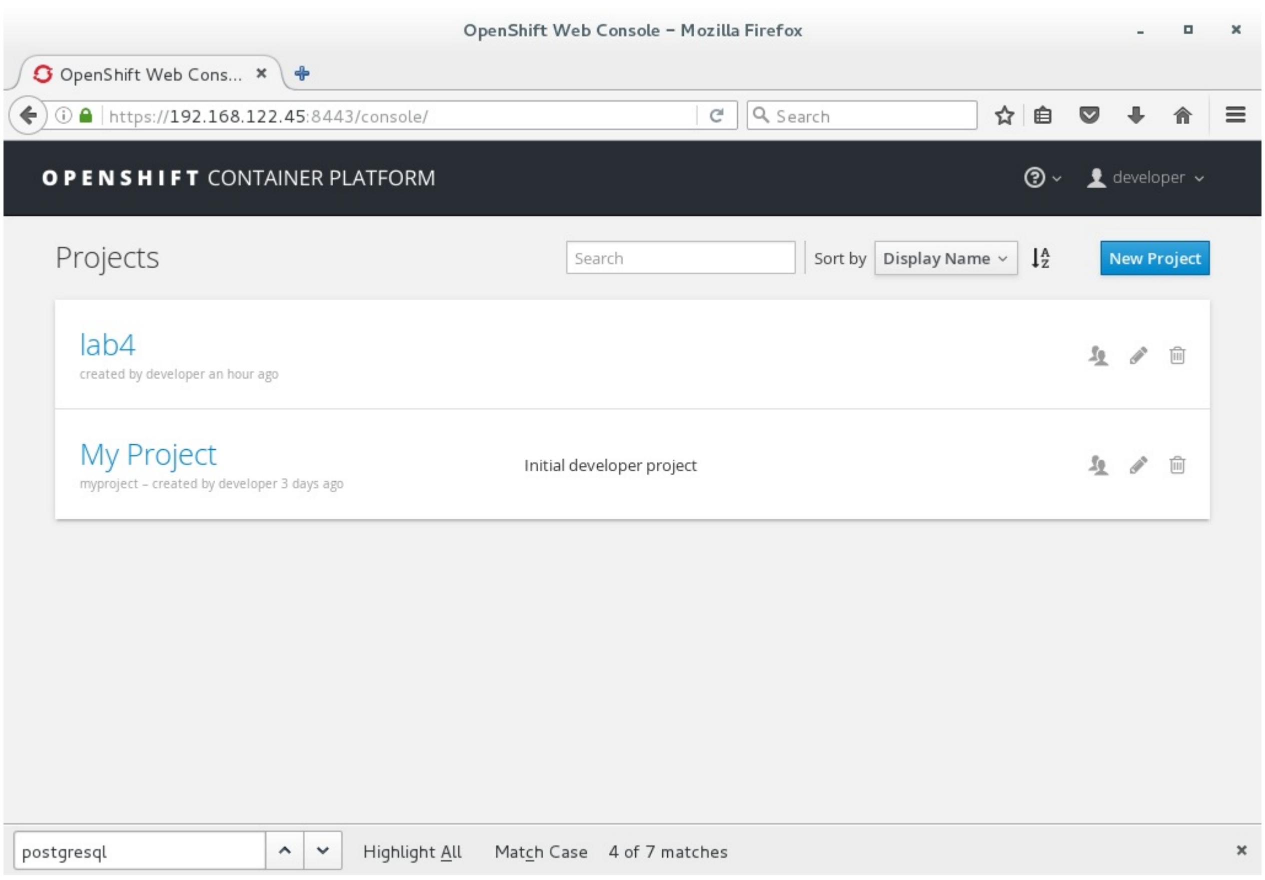
```
[student@localhost lab4_ocp]$ oc start-build rhapp-jdv-ext --from-dir=extensions
Uploading directory "extensions" as binary input for the build ...
build "rhapp-jdv-ext-2" started
```

Before going to the next step make sure the rhapp-jdv-ext build is completed.

```
[student@localhost lab4_ocp]$ oc start-build rhapp-jdv --from-dir=vdb
Uploading directory "vdb" as binary input for the build ...
build "rhapp-jdv-1" started

[student@localhost lab4_ocp]$ cd ~/summit-2017-dataservices/labs/lab4/projects
[student@localhost projects]$ oc start-build rhapp-food-service --from-dir=food-servic
e/deployments
Uploading directory "food-service/deployments" as binary input for the build ...
build "rhapp-food-service-1" started

[student@localhost projects]$ oc start-build rhapp-wine-service --from-dir=wine-servic
e/deployments
Uploading directory "wine-service/deployments" as binary input for the build ...
build "rhapp-wine-service-1" started

[student@localhost projects]$ oc start-build rhapp-ui --from-dir=ui-service/deployment
s
Uploading directory "ui-service/deployments" as binary input for the build ...
build "rhapp-ui-1" started
```

Now the containers will be built and deployed. Let’s see how it looks like in the OpenShift Web Console. Login into the OpenShift Web Console and login with username developer

Exploring the Running Containers



OpenShift Web Console – Mozilla Firefox

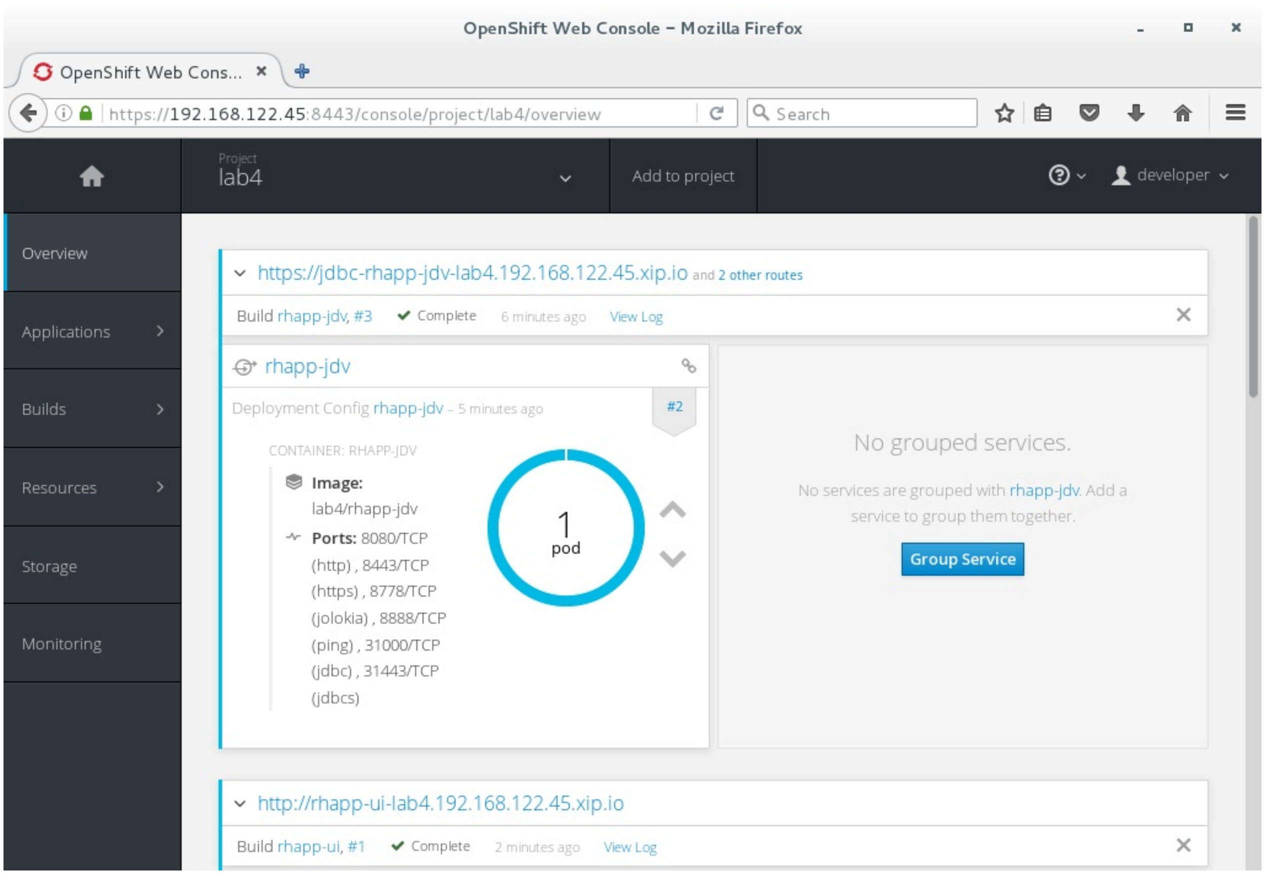
https://192.168.122.45:8443/console/

Projects

lab4	created by developer an hour ago	
My Project	myproject – created by developer 3 days ago	Initial developer project

postgresql — Highlight All — Match Case — 4 of 7 matches

Click project lab4 and the lab4 overview page should appear as depicted below.



OpenShift Web Console – Mozilla Firefox

https://192.168.122.45:8443/console/project/lab4/overview

Project lab4

https://jdbc-rhapp-jdv-lab4.192.168.122.45.xip.io and 2 other routes

Build rhapp-jdv, #3 ✓ Complete 6 minutes ago View Log

rhapp-jdv

Deployment Config rhapp-jdv – 5 minutes ago #2

CONTAINER: RHAPP-JDV

Image: lab4/rhapp-jdv

Ports: 8080/TCP (http) , 8443/TCP (https) , 8778/TCP (jolokia) , 8888/TCP (ping) , 31000/TCP (jdbc) , 31443/TCP (jdbcs)

1 pod

No grouped services.

No services are grouped with rhapp-jdv. Add a service to group them together.

http://rhapp-ui-lab4.192.168.122.45.xip.io

Build rhapp-ui, #1 ✓ Complete 2 minutes ago View Log

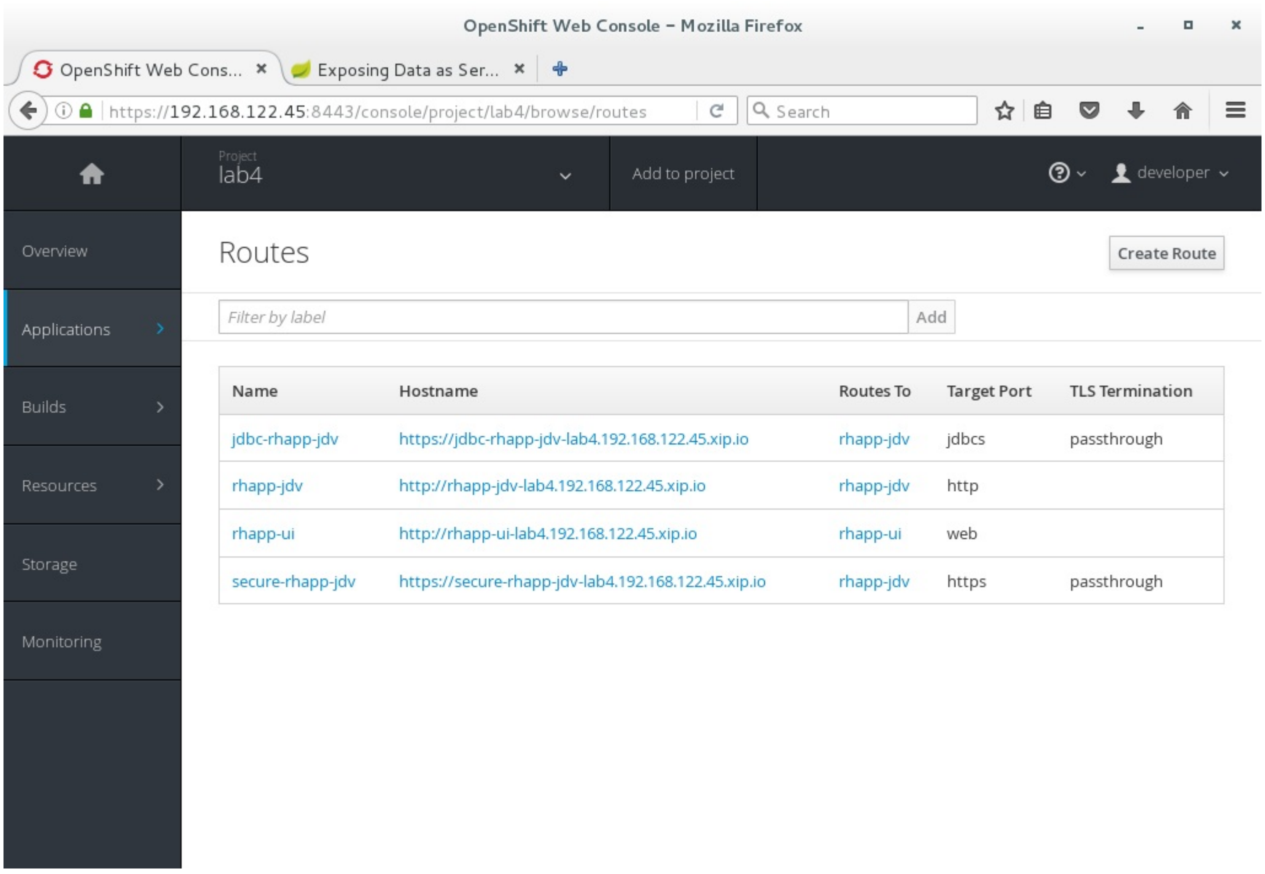
Scroll down and use the menu options to familiarize with the OpenShift lab4 project.

Connecting to the application

An OpenShift Container Platform route exposes a service at a host name, like *www.example.com*, so that external clients can reach it by name.

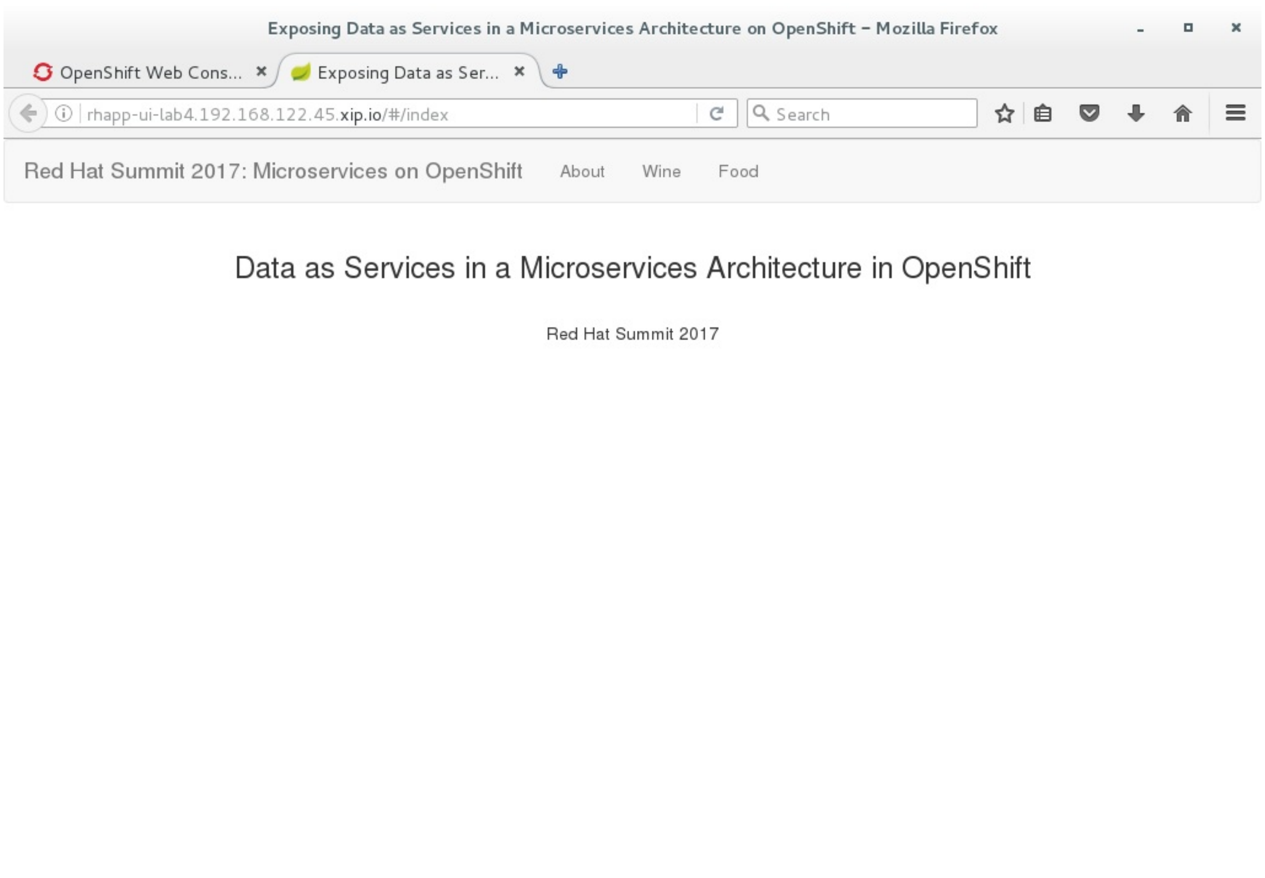
In the example depicted in screenshots before we can see routes defined in lab4 project which expose the webui of our food and wine microservices application at url: http://rhapp-ui-lab4.192.168.122.45.xip.io

Another way to get the routes is to navigate to the Applications→Routes page. Click on the URL of the ui route and you should be redirected to the food and wine microservices application as depicted below.



Routes

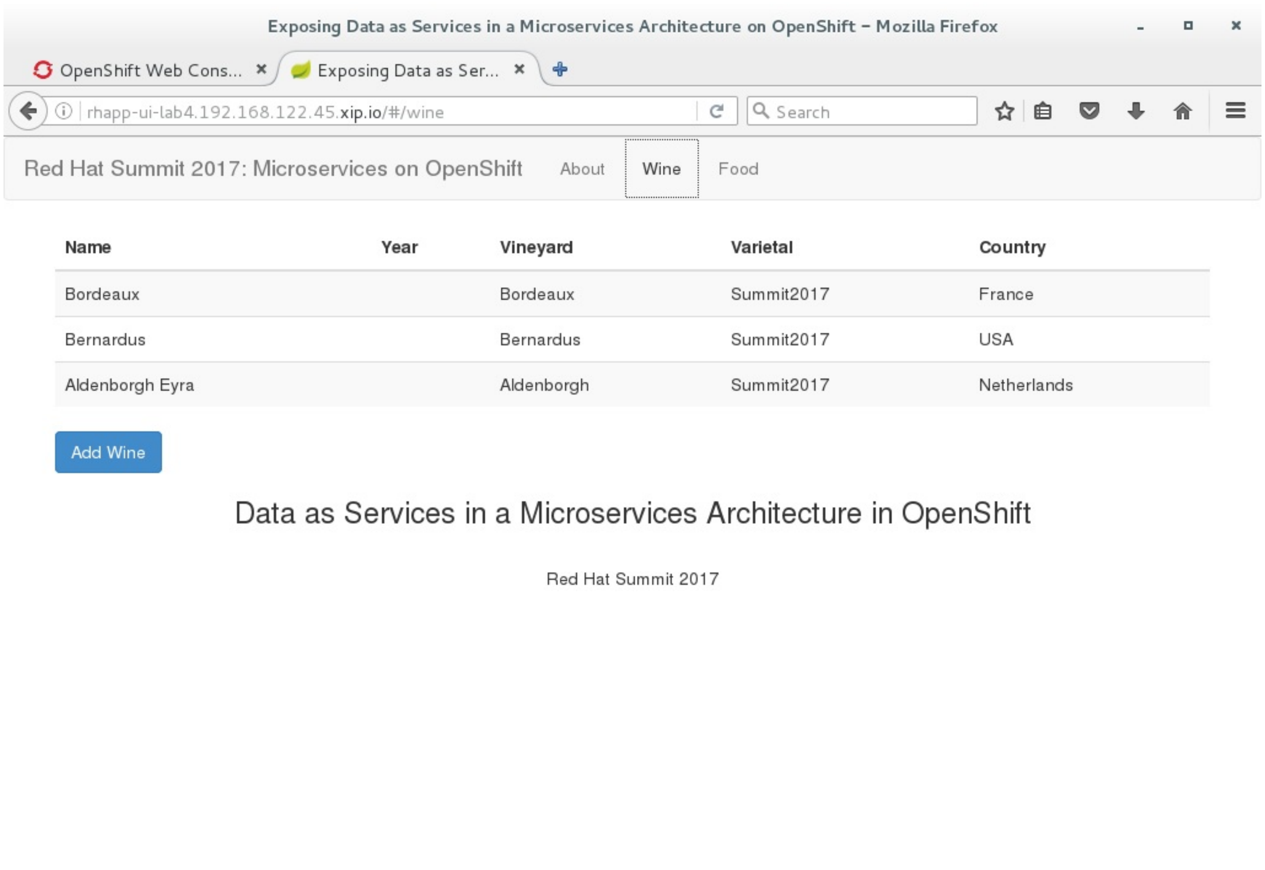
Name	Hostname	Routes To	Target Port	TLS Termination
jdbc-rhapp-jdv	https://jdbc-rhapp-jdv-lab4.192.168.122.45.xip.io	rhapp-jdv	jdbcs	passthrough
rhapp-jdv	http://rhapp-jdv-lab4.192.168.122.45.xip.io	rhapp-jdv	http	
rhapp-ui	http://rhapp-ui-lab4.192.168.122.45.xip.io	rhapp-ui	web	
secure-rhapp-jdv	https://secure-rhapp-jdv-lab4.192.168.122.45.xip.io	rhapp-jdv	https	passthrough


Exposing Data as Services in a Microservices Architecture on OpenShift – Mozilla Firefox
OpenShift Web Cons...
Exposing Data as Ser...
rhapp-ui-lab4.192.168.122.45.xip.io/#/index
Search
Red Hat Summit 2017: Microservices on OpenShift
About
Wine
Food

Data as Services in a Microservices Architecture in OpenShift

Red Hat Summit 2017

Click on menu option *wine* and you should see similar data as depicted below

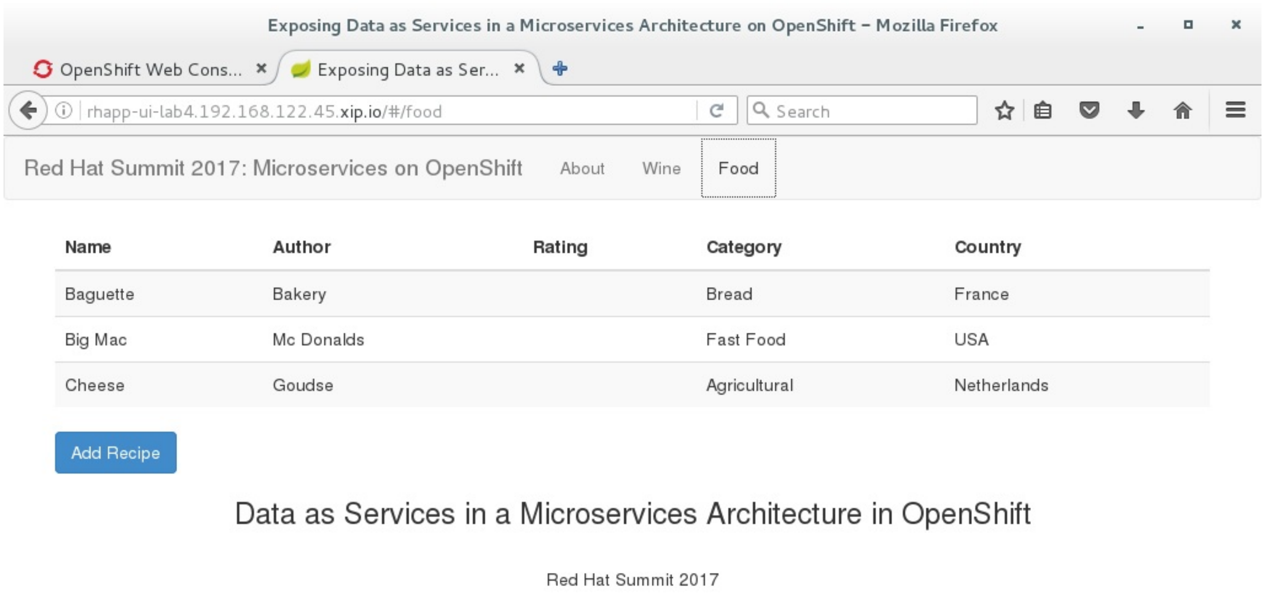

Exposing Data as Services in a Microservices Architecture on OpenShift – Mozilla Firefox
OpenShift Web Cons...
Exposing Data as Ser...
rhapp-ui-lab4.192.168.122.45.xip.io/#/wine
Search
Red Hat Summit 2017: Microservices on OpenShift
About
Wine
Food
Add Wine

Name	Year	Vineyard	Varietal	Country
Bordeaux		Bordeaux	Summit2017	France
Bernardus		Bernardus	Summit2017	USA
Aldenborgh Eyra		Aldenborgh	Summit2017	Netherlands

Data as Services in a Microservices Architecture in OpenShift

Red Hat Summit 2017

Click on menu option food and you should see similar data as depicted below


Exposing Data as Services in a Microservices Architecture on OpenShift – Mozilla Firefox
OpenShift Web Cons...
Exposing Data as Ser...
rhapp-ui-lab4.192.168.122.45.xip.io/#/food
Search
Red Hat Summit 2017: Microservices on OpenShift
About
Wine
Food
Add Recipe

Name	Author	Rating	Category	Country
Baguette	Bakery		Bread	France
Big Mac	Mc Donalds		Fast Food	USA
Cheese	Goudse		Agricultural	Netherlands

Data as Services in a Microservices Architecture in OpenShift

Red Hat Summit 2017

Cleanup lab 4

Delete project using OpenShift CLI

```
[student@localhost projects]$ oc delete project lab4
```

Remove the docker images To remove the created docker images during this lab you can do

```
[student@localhost projects]$ docker images | grep rhapp
REPOSITORY TAG IMAGE ID CREATED SIZE
172.30.1.1:5000/lab3/rhapp-ui latest e4b265ec1c0a 42 minutes ago 727.1 MB
172.30.1.1:5000/lab3/rhapp-wine-service latest eaba6f6ce6d9 42 minutes ago 796.3 MB
172.30.1.1:5000/lab3/rhapp-food-service latest 0e9a01a2f132 43 minutes ago 799.2 MB
172.30.1.1:5000/lab4/rhapp-jdv latest 5b3603a285c6 46 minutes ago 972.6 MB
```

You can remove the image one by one using:

```
[student@localhost projects]$ docker rmi <image id>
```

For you convenience we have a script called *rmlab4* available which removes all images with rhapp in the name:

```
[student@localhost projects]$ rmlab4
```

Congratulations!!!!! You have completed this lab.

Lab 4 part II - Changing the microservices application using data services with security

Data roles, also called entitlements, are sets of permissions defined per VDB that dictate data access (create, read, update, delete). Data roles use a fine-grained permission system that JDV will enforce at runtime and provide audit log entries for access violations.

Before applying Role Based Access Control (RBAC), note/consider the following: A VDB deployed without any defined data roles is open for access to any authenticated user Restrict source system access by modeling the VDB such that Imported metadata is narrowed to what will be used (directly or indirectly) by the view models At a more granular level, source table columns that will not be used or required are removed or the source table marked as non-updatable.

Let’s see this in action.

First we are going to create a new project called *lab4-secure*.

```
[student@localhost ~]$ oc new-project lab4-secure
Now using project "lab4-secure" on server "https://192.168.122.45:8443".

You can add applications to this project with the 'new-app' command. For example, try:

        oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.
```

Check out directory ~/summit-2017-dataservices/labs/lab4_secure_ocp/templates.

```
[student@localhost ~]$ cd ~/summit-2017-dataservices/labs/lab4_secure_ocp
```

In this directory you will find the OpenShift template containing the configuration needed to setup the complete lab4-secure environment. The template is exactly the same as in the lab4 environment. Take a look at food-vdb.xml and win-vdb.xml using a pre-installed code editor.

```
[student@localhost lab4_secure_ocp]$ code vdb/secure-food-vdb.xml
[student@localhost lab4_secure_ocp]$ code vdb/secure-wine-vdb.xml
```

Compare secure-food-vdb.xml and secure-wine-vdb.xml against the ones we were using during lab4 and see the differences. In example:

```
[student@localhost lab4_secure_ocp]$ code -d vdb/secure-wine-vdb.xml ~/summit-2017-dat
aservices/labs/lab4_ocp/vdb/wine-vdb.xml
[student@localhost lab4_secure_ocp]$ code -d vdb/secure-food-vdb.xml ~/summit-2017-dat
aservices/labs/lab4_ocp/vdb/food-vdb.xml
```

Note the differences, especially in the data-role section. ;) Setup environment based on pre-built OpenShift template Files for runtime artifacts are passed to the JDV for OpenShift image using the OpenShift secret mechanism. This includes the environment files for the data sources and resource adapters, as well as any additional data files. These files need to be present locally so we have to create secrets for them.

```
[student@localhost lab4_secure_ocp]$ oc create -f extensions/datavirt-app-secret.yaml
[student@localhost lab4_secure_ocp]$ oc secrets new datavirt-app-config extensions/dat
asources.properties
[student@localhost lab4_ocp]$ oc adm policy add-role-to-user view system:serviceaccoun
t:lab4-secure:datavirt-service-account
```

If you have a JSON or YAML file that defines a template, for example in our case we are using lab4-secure-template.json, you can upload the template to projects using the CLI. This saves the template to the project for repeated use by any user with appropriate access to that project. See for more instructions on writing your own templates. Furthermore you can use the CLI to process templates and use the configuration that is generated to create objects as shown below.

```
[student@localhost lab4_ocp]$ oc process -f templates/lab4-secure-template.json | oc c
reate -f -
buildconfig "rhapp-sec-food-service" created
buildconfig "rhapp-sec-jdv" created
buildconfig "rhapp-sec-jdv-ext" created
buildconfig "rhapp-sec-ui" created
buildconfig "rhapp-sec-wine-service" created
imagestream "rhapp-sec-food-service" created
imagestream "rhapp-sec-jdv" created
imagestream "rhapp-sec-jdv-ext" created
imagestream "rhapp-sec-ui" created
imagestream "rhapp-sec-wine-service" created
deploymentconfig "rhapp-sec-food-service" created
deploymentconfig "rhapp-sec-jdv" created
deploymentconfig "rhapp-sec-postgresql" created
deploymentconfig "rhapp-sec-ui" created
deploymentconfig "rhapp-sec-wine-service" created
route "jdbc-rhapp-sec-jdv" created
route "secure-rhapp-sec-jdv" created
route "rhapp-sec-jdv" created
route "rhapp-sec-ui" created
service "rhapp-sec-food-service" created
service "rhapp-sec-jdv" created
service "rhapp-sec-postgresql" created
service "rhapp-sec-ui" created
service "rhapp-sec-wine-service" created
```

Get all available build configs

```
[student@localhost lab4_ocp]$ oc get bc
NAME                     TYPE      FROM      LATEST
rhapp-sec-food-service   Source    Binary    0
rhapp-sec-jdv            Source    Binary    0
rhapp-sec-jdv-ext        Docker    Binary    1
rhapp-sec-ui             Source    Binary    0
rhapp-sec-wine-service   Source    Binary    0
```

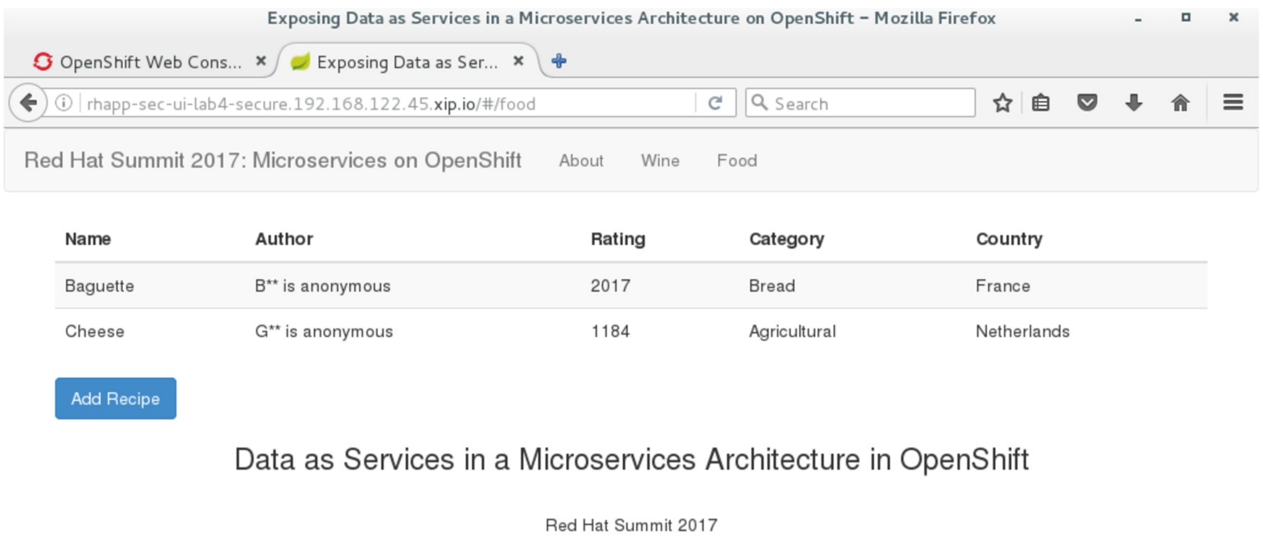
Now start the binary builds using the following commands:

```
[student@localhost lab4_secure_ocp]$ oc start-build rhapp-sec-jdv-ext --from-dir=exten
sions
Uploading directory "extensions" as binary input for the build ...
build "rhapp-sec-jdv-ext-2" started
```

Go to the OpenShift Web console and navigate to the Builds→Builds page. Make sure the build rhapp-secure-jdv-ext is completed and proceed with the next step.

```
[student@localhost lab4_secure_ocp]$ oc start-build rhapp-sec-jdv --from-dir=vdb
Uploading directory "vdb" as binary input for the build ...
build "rhapp-sec-jdv-2" started

[student@localhost lab4_secure_ocp]$ cd ~/summit-2017-dataservices/labs/lab4/projects

[student@localhost projects]$ oc start-build rhapp-sec-food-service --from-dir=food-se
rvice/deployments
Uploading directory "food-service/deployments" as binary input for the build ...
build "rhapp-sec-food-service-1" started

[student@localhost projects]$ oc start-build rhapp-sec-wine-service --from-dir=wine-se
rvice/deployments
Uploading directory "wine-service/deployments" as binary input for the build ...
build "rhapp-sec-wine-service-1" started

[student@localhost projects]$ oc start-build rhapp-sec-ui --from-dir=ui-service/deploy
ments
Uploading directory "ui-service/deployments" as binary input for the build ...
build "rhapp-sec-ui-1" started
```

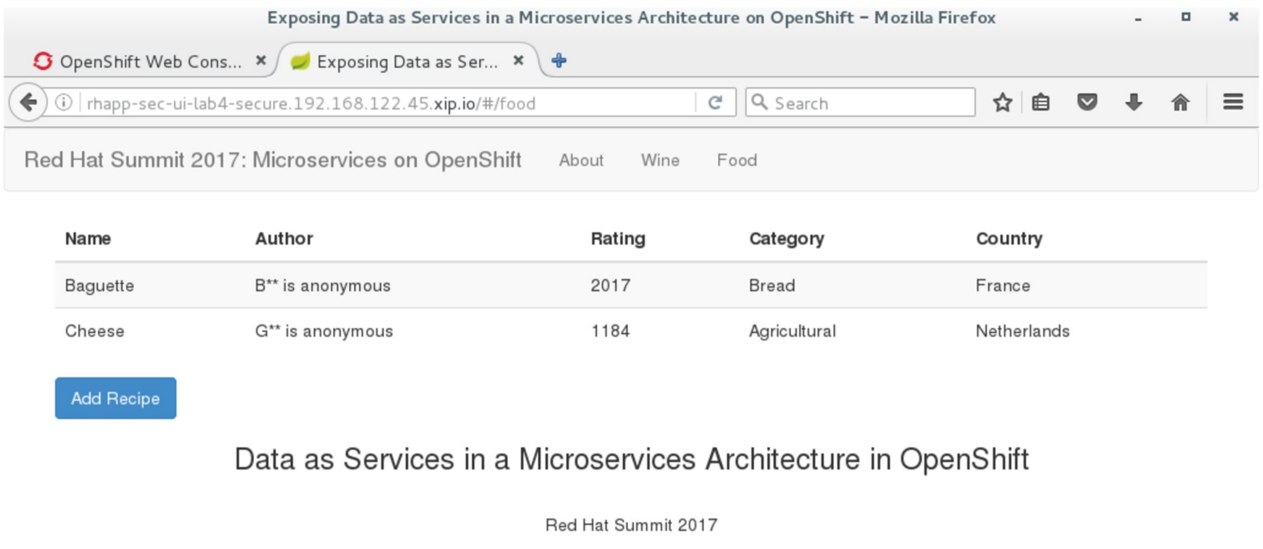
Now the containers will be built and deployed. Let’s see how our web application and in particular the wine and food data looks like when clicking on Wine and Food menu options.


Exposing Data as Services in a Microservices Architecture on OpenShift – Mozilla Firefox
OpenShift Web Cons...
Exposing Data as Ser...
rhapp-sec-ui-lab4-secure.192.168.122.45.xip.io/#/food
Search
Red Hat Summit 2017: Microservices on OpenShift
About
Wine
Food
Add Recipe

Name	Author	Rating	Category	Country
Baguette	B** is anonymous	2017	Bread	France
Cheese	G** is anonymous	1184	Agricultural	Netherlands

Data as Services in a Microservices Architecture in OpenShift

Red Hat Summit 2017


Exposing Data as Services in a Microservices Architecture on OpenShift – Mozilla Firefox
OpenShift Web Cons...
Exposing Data as Ser...
rhapp-sec-ui-lab4-secure.192.168.122.45.xip.io/#/food
Search
Red Hat Summit 2017: Microservices on OpenShift
About
Wine
Food
Add Recipe

Name	Author	Rating	Category	Country
Baguette	B** is anonymous	2017	Bread	France
Cheese	G** is anonymous	1184	Agricultural	Netherlands

Data as Services in a Microservices Architecture in OpenShift

Red Hat Summit 2017

Do you see any difference in the wine/food data compared to the previous lab 4 without security? Explain why….

Tip	check secure-wine-vdb.xml and secure-food-vdb.xml again.

Cleanup lab4_secure

Delete project using OpenShift CLI

```
[student@localhost projects]$ oc delete project lab4-secure
```

Remove the docker images To remove the created docker images during this lab you can do

```
[student@localhost projects]$ docker images | grep rhapp-sec
REPOSITORY                                                                    TAG
                IMAGE ID                CREATED                 SIZE
172.30.1.1:5000/lab4-secure/rhapp-sec-wine-service                         latest
           6b3908837766        About an hour ago   884.7 MB
172.30.1.1:5000/lab4-secure/rhapp-sec-food-service                         latest
           0a774c1c21ed        About an hour ago   889.6 MB
172.30.1.1:5000/lab4-secure/rhapp-sec-ui                                   latest
           09ec00b02d1c        About an hour ago   775.6 MB
172.30.1.1:5000/lab4-secure/rhapp-sec-jdv                                  latest
           d0c414c5c4cf        About an hour ago   972.7 MB
172.30.1.1:5000/lab4-secure/rhapp-sec-jdv                                  <none>
           1f9933bb9eb8        About an hour ago   972.7 MB
172.30.1.1:5000/lab4-secure/rhapp-sec-jdv-ext                              latest
           ff5776835b2b        About an hour ago   972.7 MB
```

You can remove the image one by one using:

```
[student@localhost projects]$ docker rmi <image id>
```

For you convenience we have a script called rmlab4secure available which removes all images with rhapp in the name:

```
[student@localhost projects]$ rmlab4secure
```

Congratulations, you’ve finished all labs!!!!!

In this year’s Summit lab you have learnt how to expose data as services in a microservice architecture using Red Hat JBoss Data Virtualization running on Red Hat OpenShift Container Platform. Got exited, see below for a list of useful resource to get even more excited. Enjoy your further stay at Red Hat Summit 2017.

Resources

Description	Website
OpenShift	http://www.openshift.com and https://developers.redhat.com/products/openshift/
OpenShift CLI tool download	https://access.redhat.com/downloads/content/290
Red Hat Container Catalog	https://access.redhat.com/containers
OpenShift Container Tested Integrations	https://access.redhat.com/articles/2176281
Red Hat JBoss Data Virtualization (JDV)	http://developers.redhat.com/products/datavirt
Red Hat JDV 6.x Supported Configurations	https://access.redhat.com/articles/703663
Red Hat JBoss Data Grid (JDG)	http://developers.redhat.com/products/datagrid
Red Hat JDG 6.x Supported Configurations	https://access.redhat.com/articles/115883
Red Hat JDG 7.x Supported Configurations	https://access.redhat.com/articles/2435931
Red Hat Middleware images for OpenShift Documentation	https://access.redhat.com/documentation/en/red-hat-xpaas/0/paged/red-hat-xpaas-jdv-for-openshift-image/