DO180 - Red Hat OpenShift I: Containers & Kubernetes 4.5

An Introduction

Course Info

Send your Redhat Ids: https://forms.gle/WhLTjJZhcpzLzrmf7
(Check your emails for the course invitations)

Course URL : https://rha.ole.redhat.com/rha/app/courses/do180-4.6/67500f64-8484-4e48-b612-76ba6efa0047/pages/ch01

Grades distribution

Project : 30

3 phases , upto 2 Students in a Team

Midterm : 10

Final Exam : 60

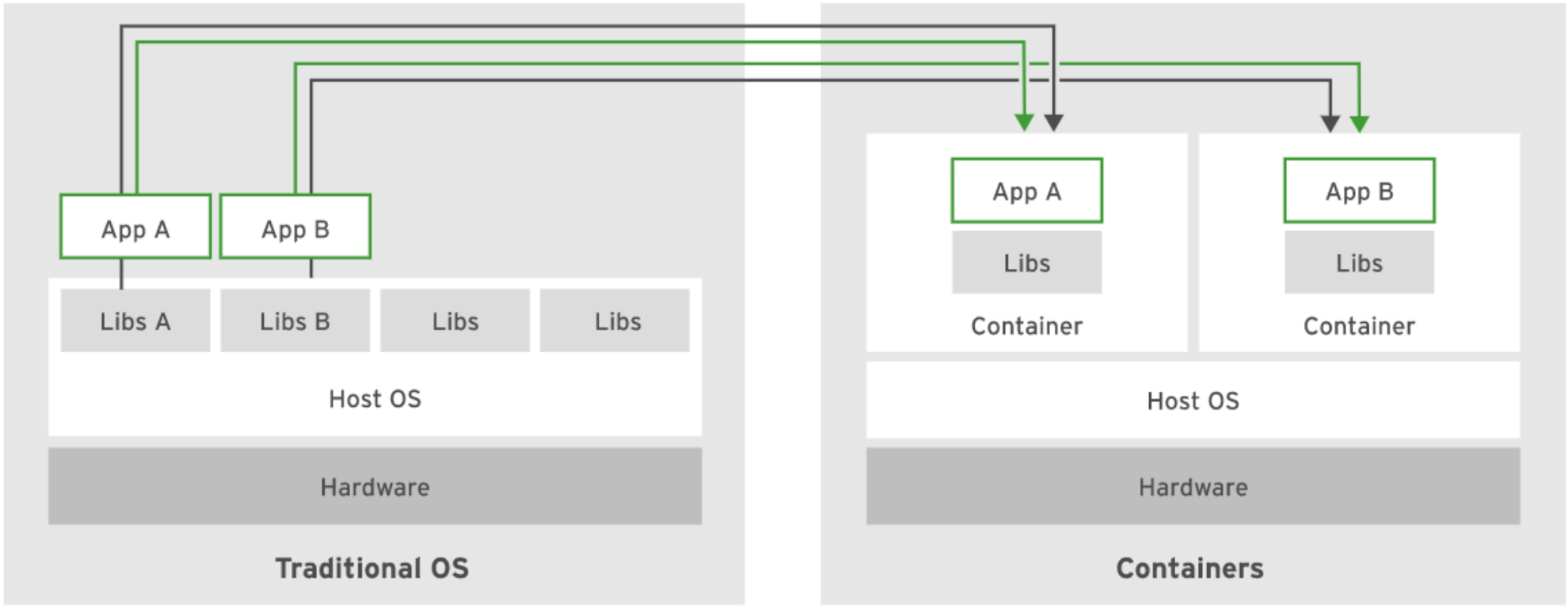
Required accounts

1. Redhat account to access the course material
2. Create github account (check Appendix B)
3. Create Quay account (check Appendix C)

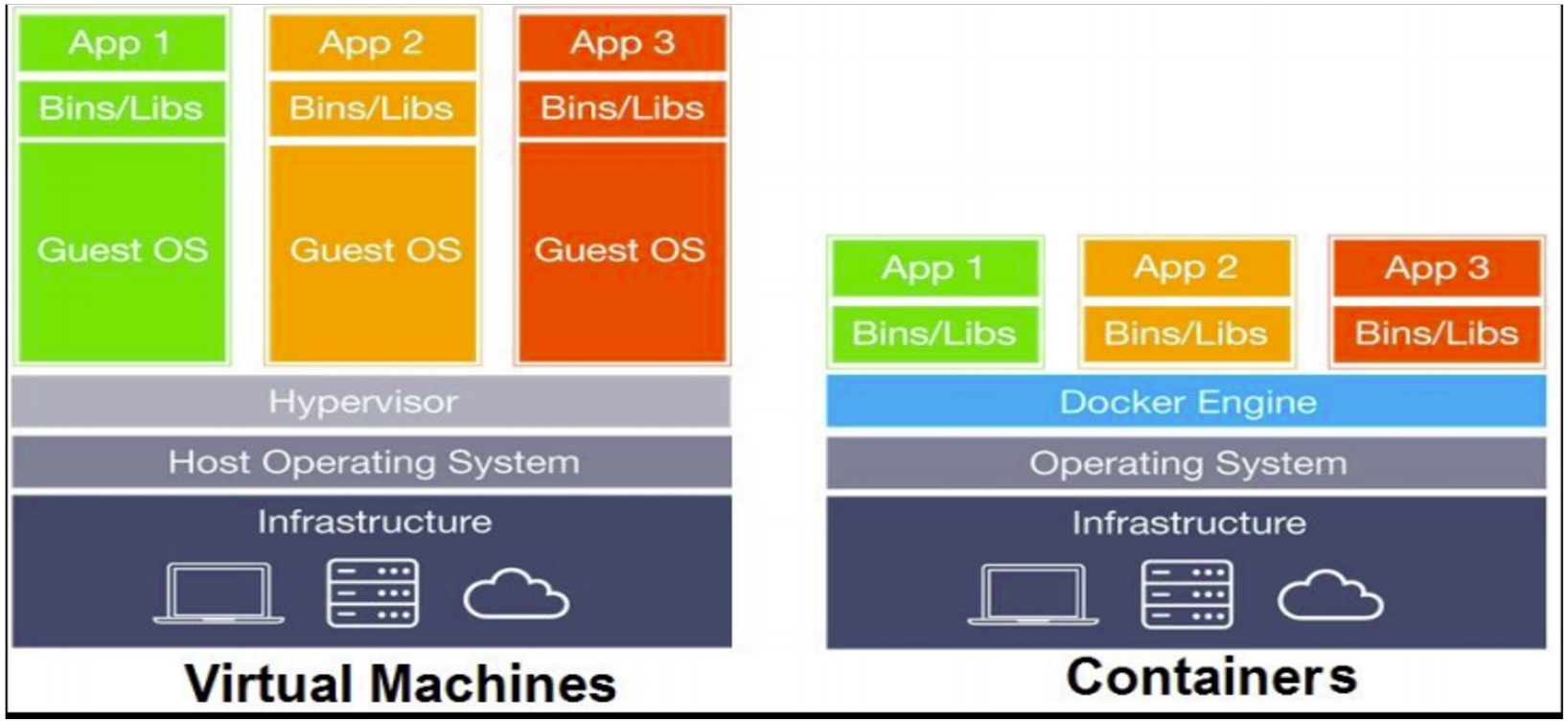
Course content summary

- Container and OpenShift architecture
- Creating containerized services
- Managing containers and container images
- Creating custom container images
- Deploying containerized applications on OpenShift
- Deploying multi-container applications

Traditional OS vs Containers



Virtualization vs Containerization



Virtualization vs Containerization

- ***Virtualization*** − Virtualization is the technology that can simulate your physical hardware (such as CPU cores, memory, disk) and represent it as a separate machine. It has its own Guest OS, Kernel, process, drivers, etc. Therefore, it is ***hardware-level virtualization***. Most common technology is "VMware" and "Virtual Box".
- ***Containerization*** − Containerization is "***OS-level virtualization***". It doesn't simulate the entire physical machine. It just simulates the OS of your machine. Therefore, multiple applications can share the same OS kernel. Containers play similar roles as virtual machine but without hardware virtualization. Most common container technology is "*Docker*".

Virtualization vs Containerization

Key Factor	Virtualization	Containerization
Technology	One physical machine has multiple OSs residing on it and appears as multiple machines.	The application developed in a host environment with same OS and the same machine executes flawlessly on multiple different environments.
Start-up Time	Higher than containers	Less
Speed of working	VMs being a virtual copy of the host server on its own operating system, VMs are resource-heavy, hence slower.	Containers are faster.
Size	Larger	Smaller
Component that Virtualizes and the one being virtualized	Hypervisors virtualize the underlying hardware (use of the same hardware).	Containers virtualize the operating system (use of the same OS).
Cost of implementation	Higher	Lower
Benefits for	IT enterprise businesses	Software developers and in turn IT businesses

Overview of Container Architecture

- From the Linux kernel perspective, a container is a process with restrictions. However, instead of running a single binary file, a container runs an image.
- An image is a file-system bundle that contains all dependencies required to execute a process: files in the file system, installed packages, available resources, running processes, and kernel modules.
- Like executable files are the foundation for running processes, *images* are the foundation for running containers.
- Container images need to be locally available for the container runtime to execute them, but the images are usually stored and maintained in an image repository. An image repository is just a service where images can be stored, searched and retrieved.
- There are many different image repositories available, each one offering different features: Docker Hub, Redhat Quay, and others.

Pros of Containers

- Low hardware footprint
- Environment Isolation
- Quick Deployment
- Reusability

Limitations of containers

- Containers provide an easy way to package and run services.
- As the number of containers managed by an organization grows, the work of manually starting them rises exponentially along with the need to quickly respond to external demands.
- And here comes the need for “*orchestration*”

Kubernates and Openshift

- **Kubernetes** is an orchestration service that simplifies the deployment, management, and scaling of containerized applications.
  - The smallest unit manageable in Kubernetes is a **pod**.
  - A pod consists of one or more containers with its storage resources and IP address that represent a single application.
- **Red Hat OpenShift** Container Platform (RHOCP) is a set of modular components and services built on top of a Kubernetes container infrastructure.
  - RHOCP adds the capabilities to provide a production PaaS platform such as remote management, multitenancy, increased security, monitoring and auditing, application life-cycle management, and self-service interfaces for developers.

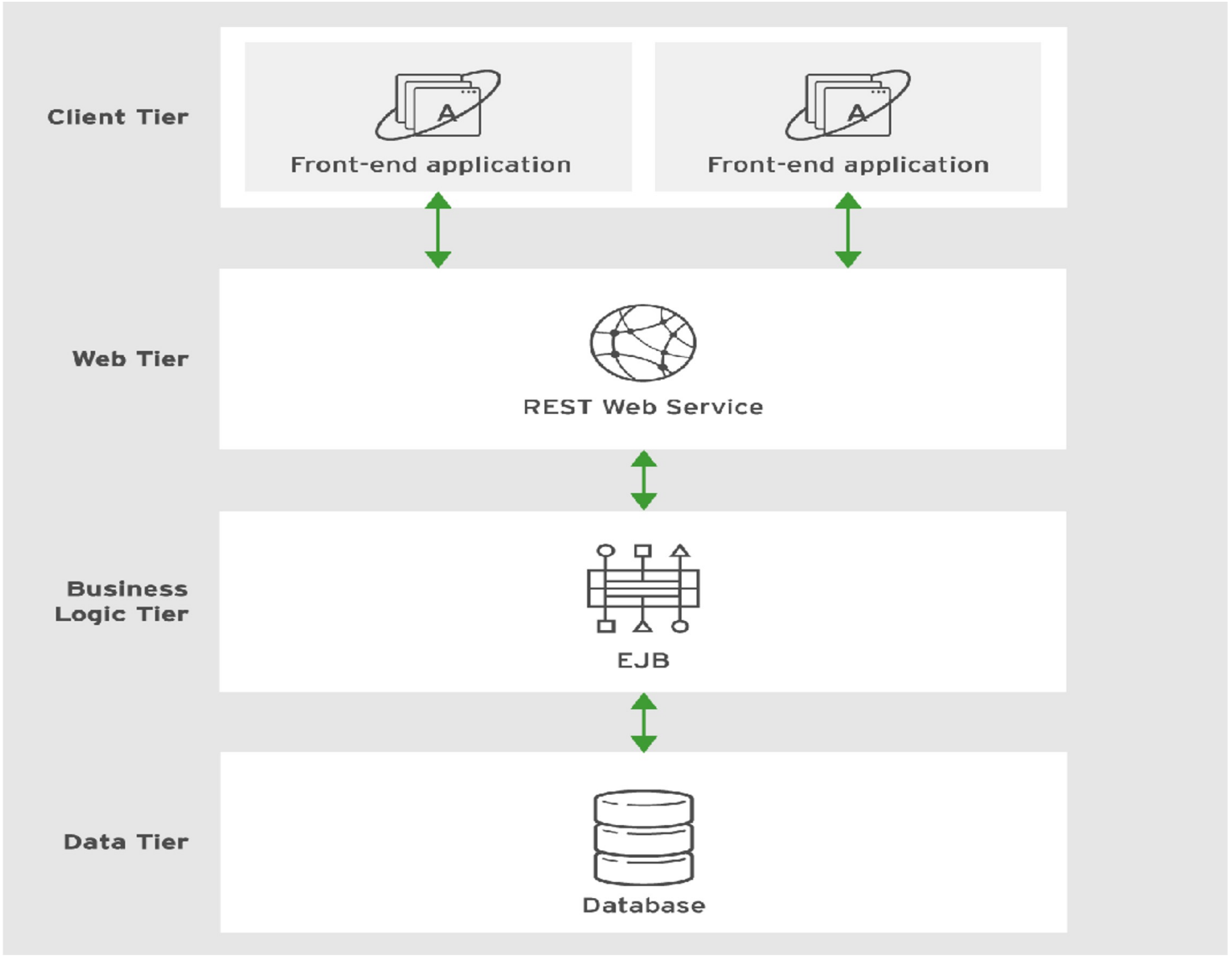
Guided Exercise: Configuring the Classroom Environment

- Next Lab

Revision

- Multi-tiered Applications
- Creating REST Services

Multi-tiered Applications



RESTful Web Services with JAX-RS

JAX-RS is the Java API used to create RESTful web services.

By implementing a web service layer, developers can abstract the front-end layer and create an application comprised of many loosely coupled components. This type of architecture is known as a **client-server** architecture, and it is a requirement for REST web services.

Example: Call and Response

API call:

curl localhost:8080/todo/api/items/1

[student@workstation todojee]$ **curl localhost:8080/todo/api/items/1**

Json response:

{"id":1,"description":"Pick up newspaper","done":false,"user":null}

Creating RESTful Web Services - step 1

```
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/api")
public class Service extends Application {
 //Can be left empty
}
```

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/
xml/ns/javaee/web-app_3_0.xsd">
  <servlet>
      <servlet-name>javax.ws.rs.core.Application</servlet-name>
  </servlet>
  <servlet-mapping>
      <servlet-name>javax.ws.rs.core.Application</servlet-name>
      <url-pattern>/api/*</url-pattern>
  </servlet-mapping>
```

Creating RESTful Web Services - step 2

Basic pojo class - Before

```
public class HelloWorld {

  public String hello() {
     return "Hello World!";
  }
}
```

RESTful web service - After

```
@Stateless
@Path("hello")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public class HelloWorld {

  @GET
  public String hello() {
     return "Hello World!";
  }
}
```

Annotation	Description
`@ApplicationPath`	The **`@ApplicationPath`** annotation is applied to the subclass of the **`javax.ws.rs.core.Application`** class and defines the base URI for the web service.
`@Path`	The **`@Path`** annotation defines the base URI for either the entire root class or for an individual method. The path can contain either an explicit static path, such as **`hello`**, or it can contain a variable to be passed in on the request. This value is referenced using the **`@PathParam`** annotation.
`@Consumes`	The **`@Consumes`** annotation defines the type of the request's content that is accepted by the service class or method. If an incompatible type is sent to the service, the server returns HTTP error 415, "Unsupported Media Type." Acceptable parameters include **`application/json`**, **`application/xml`**, **`text/html`**, or any other MIME type.

	response's content that is returned by the service class or method. Acceptable parameters include **`application/json`**, **`application/xml`**, **`text/html`**, or any other MIME type.
`@GET`	The **`@GET`** annotation is applied to a method to create an endpoint for the HTTP GET request type, commonly used to retrieve data.
`@POST`	The **`@POST`** annotation is applied to a method to create an endpoint for the HTTP POST request type, commonly used to save or create data.
`@DELETE`	The **`@DELETE`** annotation is applied to a method to create an endpoint for the HTTP DELETE request type, commonly used to delete data.
`@PUT`	The **`@PUT`** annotation is applied to a method to create an endpoint for the HTTP PUT request type, commonly used to update existing data.
`@PathParam`	The **`@PathParam`** annotation is used to retrieve a parameter passed in through the URI, such as **`http://localhost:8080/hello-web/api/hello/1`**.

Http method

- GET: The GET method retrieves data.
- POST: The POST method creates a new entity.
- DELETE: The DELETE method removes an entity.
- PUT: The PUT method updates an entity.

HTTP Status Code	Description
200	"OK." The request is successful.
400	"Bad Request." The request is malformed or it is pointing to the wrong endpoint.
403	"Forbidden." That client did not provide the correct credentials.
404	"Not Found." The path or endpoint is not found or the resource does not exist.
405	"Method Not Allowed." The client attempted to use an HTTP method on an endpoint that does not support it.
409	"Conflict." The requested object cannot be created because it already exists.
500	"Internal Server Error." The server failed to process the request. Contact the owner of the REST service to investigate the reason.