DO180 - Red Hat OpenShift I: Containers & Kubernetes 4.5

Proposed grades distribution

Project : 40

- 3 phases , upto 3 Students in a Team
- Phase 1: 10
- Phase 2: 15
- Phase 3: 15

Final Exam: Practical (60)

Required accounts

1- Create github account (check Appendix B)

2- 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

The Challenge

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Development VM

QA server

Customer Data Center

Public Cloud

Disaster recovery

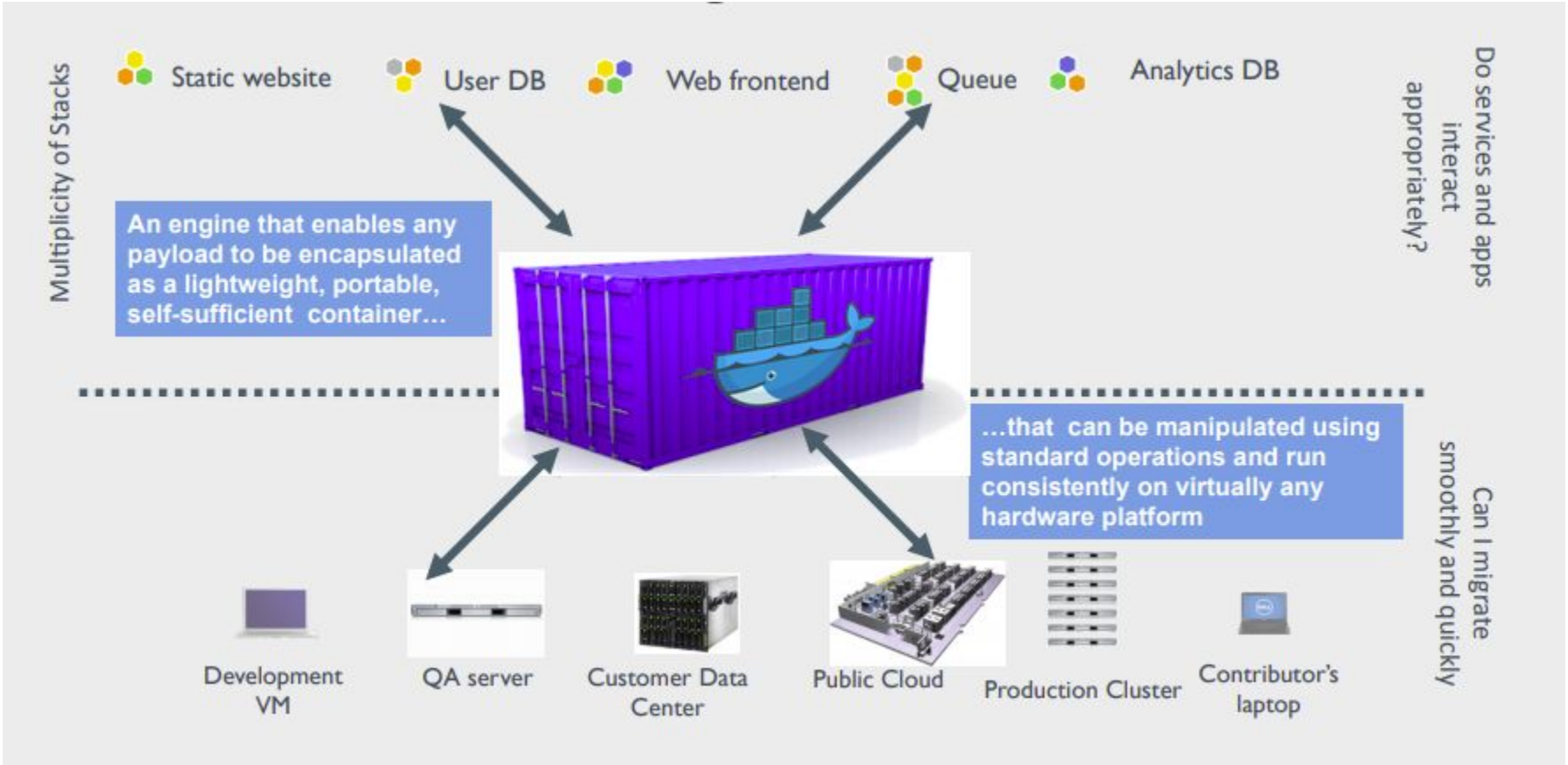
Production Servers

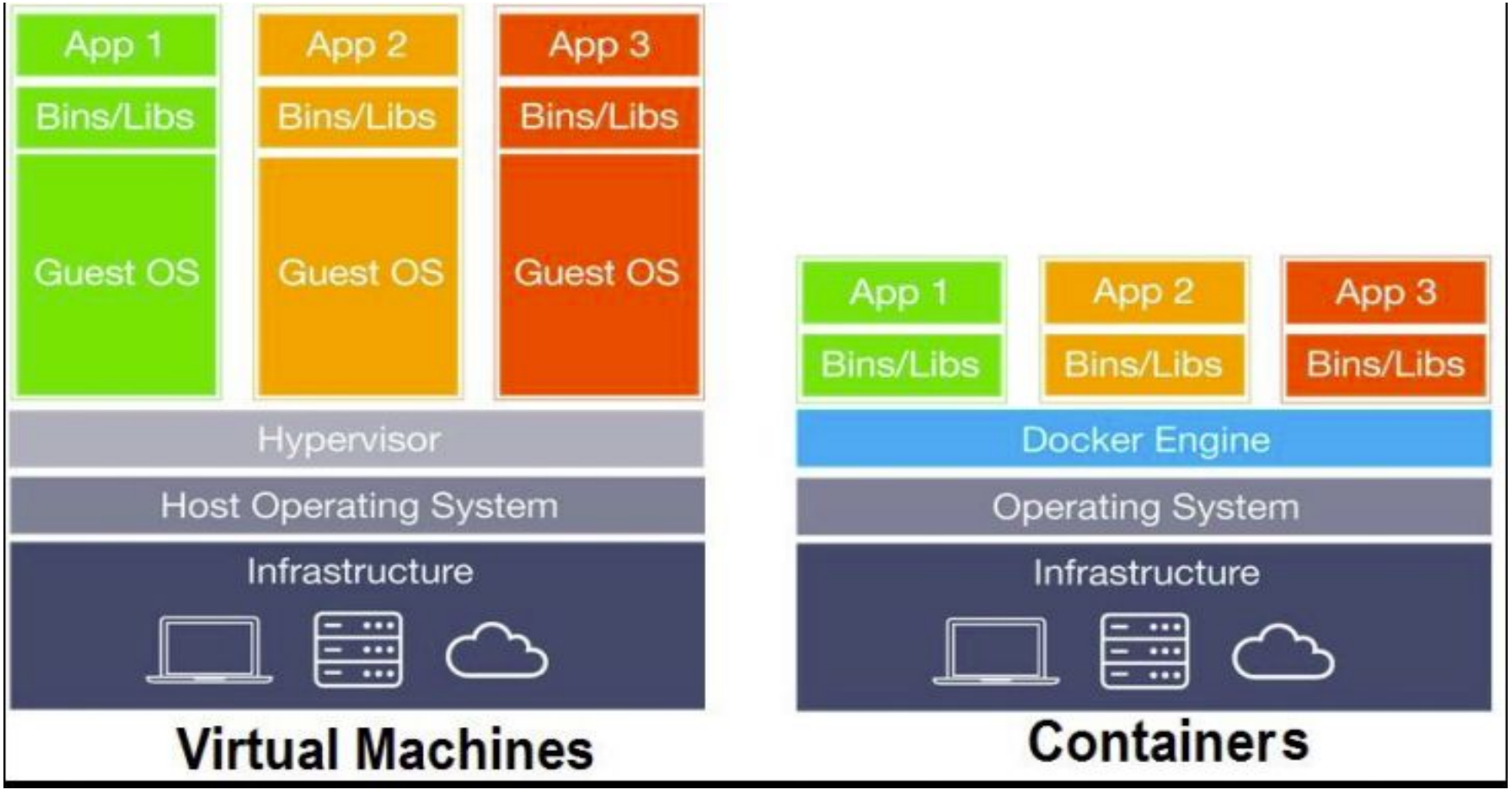
Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly?

Shipping container for code




App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure
Virtual Machines
App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Engine
Operating System
Infrastructure
Containers

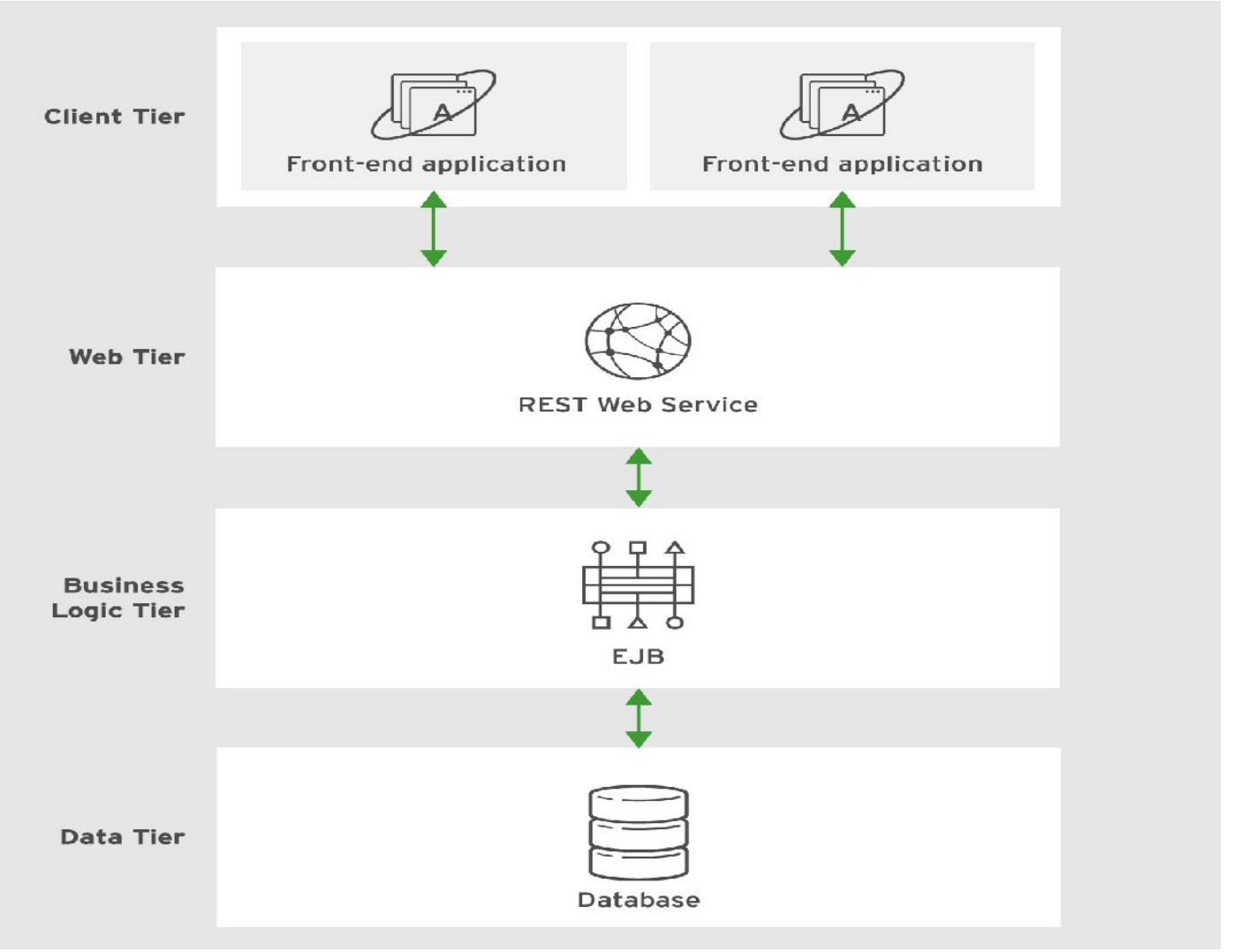
Virtualization vs containerization

Virtualization enables you to run multiple operating systems on the hardware of a single physical server, while **containerization** enables you to deploy multiple applications using the same operating system on a single virtual machine or server.

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.

Project Description

- You are required to develop a ToDo list application using MEAN Stack framework.
- MEAN stands for MongoDB, Express.js, Angular and NodeJs.
- Todo list frontend code use Angular and use a backend api developed in NodeJs and use Express as a web server and MongoDB as a database

Project Description (Cnt)

- Phase-1
  - Follow the following tutorial from here
  - Reuse the following github repo from here.
  - Extend the app to support registration and login modules
  - A running local version of the application is expected after Phase-1 where you use your local machine to include the required tools and stacks to deploy the application locally.

Old Material !

Project Description (Cnt)

- Phase-2
  - You required to extend the Phase-1 delivery to use a kubernetes and docker container to deploy the application inside a 3 containers
    - Frontend Container
    - Backend Api Container
    - MongoDB Container.

Project Description (Cnt)

- Phase-3
  - Based on Phase-2 delivery, You required to deploy the containers into openshift using the free account as a containerized services.
  - You should utilize all what you have learned during the course to provide a production level scalable deployment of your application.
- Bonus :)
  - Deliver a load test report against your deployed application where your are able to serve 1000 request concurrently, you