Table of Contents

* [Homework Assignment Lab](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_homework_assignment_lab)
* [1. Business Use Case](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_business_use_case)
* [2. POC Requirements](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_poc_requirements)
  + [2.1. CI/CD Infrastructure Setup](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_ci_cd_infrastructure_setup)
  + [2.2. OpenShift Setup](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_openshift_setup)
  + [2.3. Development Pipeline](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_development_pipeline)
  + [2.4. Deployment Pipeline](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_deployment_pipeline)
* [3. Instructions](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_instructions)
* [4. Homework Submission](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_homework_submission)
  + [4.1. OPENTLC Grading Jenkins](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_opentlc_grading_jenkins)
  + [4.2. Hints](https://www.opentlc.com/labs/ocp_advanced_development/07_1_Assignment_Lab.html#_hints)

Homework Assignment Lab

**Goal**

* Assess hands-on proficiency with Red Hat OpenShift Advanced Development topics

**Criteria**

* Expect assignments to take approximately 30-40 hours to complete
* Assignments are an individual effort—complete assignments without collaboration
* Assignments simulate a challenge typically encountered in a Red Hat consulting engagement
* Assignments are graded automatically via the grading pipeline

**Grading**

* Assignments must pass the automated grading pipeline

**Submission**

* All code must be in a public Git-compatible repository.
* You submit your homework via one of the following:
  + [Red Hat LMS](https://docs.google.com/document/d/1nxlvAOlSdNs3-y8AkmDjnc8vtCH9rJdI5zbN9deCK50/edit) if you are a Red Hat employee
  + [Red Hat Connect](https://partner.redhat.com/) if you are a business partner
* Submit your documentation as you would submit a client engagement journal. The documentation must include a link to the public Git repository.

**Environment**

* You may choose where to develop your solution:
  + If you are also completing the OpenShift Advanced Deployment accreditation, you can use the same environment for this assignment.
  + You can develop the entire solution on your laptop using the Container Development Kit (Minishift) and then move it to the **OpenShift AdvDev Homework** environment.
  + If you do not have your own environment or cannot set up the Container Development Kit, you can use the **OpenShift AdvDev Homework** environment.

**OpenShift AdvDev Homework Environment**

* + - You can use a shared cluster to build/submit your homework assignment.
    - As usual, you can request access in the [OPENTLC lab portal](https://labs.opentlc.com/)
    - You need the following two items:
      * **OpenShift AppDev Homework**: This item gives you access to the [OPENTLC OpenShift 3.9 Shared Access](https://master.na39.openshift.opentlc.com/)environment. Once you provision this item, you can create projects and you are provided with sufficient quota to complete the homework assignment.
      * **OpenShift 3.9 - Client VM**: Use the instructions in the email that you receive from Red Hat to access this VM.
        + The client VM is optional. You can also use the **oc** client from your laptop if you prefer.

1. Business Use Case

You are a consultant assigned to MitziCom, a telecommunications company that provides hosting and cloud services to a variety of clients, from medium-sized companies to enterprise giants.

MitziCom has asked you to lead a 30- to 40-hour proof-of-concept (POC) using Red Hat OpenShift Container Platform. The purpose of the POC is to determine the feasibility of using OpenShift as a target for an existing Java-based microservices workload.

MitziCom has asked you to demonstrate a fully integrated CI/CD pipeline using Nexus as the artifact repository and container registry and SonarQube for source code analysis. The application needs to be deployed to production in a blue-green strategy orchestrated by Jenkins.

The application consists of three microservices—two back-end services and one front-end service calling the back-end services. The repository also contains build and deployment information that you need. Make sure to understand how the application works before starting the project.

You can find the source code as well as skeleton shell scripts and pipelines for the solution in this [GitHub repository](https://github.com/wkulhanek/advdev_homework_template.git). Read the notes and comments in the repository carefully.

The automatic grading pipeline can be found [here](https://github.com/wkulhanek/advdev_homework_grading).

2. POC Requirements

MitziCom management requires that you include all of the items listed in these subsections in your POC.

2.1. CI/CD Infrastructure Setup

* Create a public Git repository with your homework code.
* Set up the following CI/CD components in individual projects—all with persistent storage:
  + Jenkins
  + Nexus
  + SonarQube
* Create liveness and readiness probes for all infrastructure components.
* For each infrastructure component, create a template or Ansible Playbook in order to easily reuse your work.
  + Verify that you can create the components using the templates or Ansible Playbooks by implementing the provided skeleton shell scripts.
  + Make sure the templates or Ansible Playbooks have adequate parameters to customize the application.

**Grading Guide:**

* Jenkins available with persistent storage
* Fully configured Nexus with persistent storage
* SonarQube with persistent storage
* Liveness and readiness probes for infrastructure components
* Templates or Ansible Playbooks for all infrastructure components

2.2. OpenShift Setup

* Create two OpenShift projects for development (including test) and production.
  + To conserve resources, you need to test on the development images of the application, rather than setting up a separate test/QA project.
* Set up the correct permissions for Jenkins to manipulate objects in the development and production projects.
* Set up the correct permissions for the production project to deploy images from the development project.
* Set up a MongoDB database in the development project.
* Set up a *replicated* MongoDB database (StatefulSet) with at least three replicas in the production project.
* Create build configurations in the development project.
* Create deployment configurations in both the development and production projects.
* Configure the applications using ConfigMaps.

**Grading Guide:**

* Two projects available with correct permissions
* Build configurations set up in the development project
* Deployment configurations set up in the development project
* Deployment configurations set up in the production project
* MongoDB database in the development project
* Replicated MongoDB database in the production project
* ConfigMaps used to configure applications (MongoDB databases, Nationalparks, MLBParks)

2.3. Development Pipeline

* Create a Jenkins pipeline for each of the three services and store them in the source code repository.
* Make sure that the development pipelines do the following:
  + Build the source code, using Nexus as a Maven proxy cache
  + Execute the following tests in parallel and present the results in the build log:
    - Unit tests
    - Code coverage tests
  + Tag the image with the version and build number
  + Upload the generated artifact to the artifact repository
  + Run an integration test if the service built was a back-end service
  + Upload the tested container image to another (Nexus) Docker registry

**Grading Guide:**

* Three Jenkins pipelines in the source code repository
* Each Jenkins pipeline builds the application using Nexus
* Each Jenkins pipeline executes unit tests
* Each Jenkins pipeline executes code coverage tests
* Each Jenkins pipeline tags the image correctly
* Each Jenkins pipeline uploads the generated artifact to the artifact repository
* Each Jenkins pipeline executes integration tests if a back-end service is built
* Each Jenkins pipeline uploads the finished Docker image to another Docker registry

2.4. Deployment Pipeline

* Continue your pipeline development with the following:
  + Tags the image as **version** for production deployment
  + Deploys only the newly built microservice using a blue-green strategy
  + Automatically executes the blue-green go-live switch

**Grading Guide:**

* Jenkins pipeline tags the image as **version**
* Jenkins pipeline only deploys the newly built microservice in a blue-green fashion
* Jenkins pipeline switches to the newly built microservice

3. Instructions

1. Clone the shell source code repository (<https://github.com/wkulhanek/advdev_homework_template.git>) to your hard drive.
2. Create a public Git repository under your account.
3. Populate the public Git repository with the contents of the shell source code repository.
4. Implement the shell scripts under **Infrastructure/bin** where each script calls a template with the correct parameters rather than executing shell script commands.
   * **setup\_projects.sh**: Does not need to change
   * **delete\_projects.sh**: Does not need to change
   * **setup\_jenkins.sh**: This script needs to do the following in the **$GUID-jenkins** project:
     + Create a Jenkins instance with persistent storage and sufficient resources
     + Create a build configuration to build the custom Maven slave pod to include Skopeo
     + Set up three build configurations with pointers to the pipelines in the source code project. Each build configuration needs to point to the source code repository and the respective **contextDir**. The build configurations also need the following environment variables:
       - **GUID**: The common GUID for all projects
       - **CLUSTER**: The cluster base URL—for example, **na39.openshift.opentlc.com**
   * **setup\_nexus.sh**: This script needs to do the following in the **$GUID-nexus** project:
     + Create a new Nexus instance from **docker.io/sonatype/nexus3:latest**
     + Configure Nexus appropriately for resources, deployment strategy, persistent volumes, and readiness and liveness probes
     + When Nexus is running, populate Nexus with the correct repositories
     + Expose the container registry
   * **setup\_sonar.sh**: This script needs to do the following in the **$GUID-sonarqube** project:
     + Create a new PostgreSQL database
     + Create a new SonarQube instance from **docker.io/wkulhanek/sonarqube:6.7.4**
     + Configure SonarQube appropriately for resources, deployment strategy, persistent volumes, and readiness and liveness probes
   * **setup\_dev.sh**: This script needs to do the following in the **$GUID-parks-dev** project:
     + Grant the correct permissions to the Jenkins service account
     + Create a MongoDB database
     + Create binary build configurations for the pipelines to use for each microservice
     + Create ConfigMaps for configuration of the applications
       - Set **APPNAME** to the following values—the grading pipeline checks for these exact strings:
         * **MLB Parks (Dev)**
         * **National Parks (Dev)**
         * **ParksMap (Dev)**
     + Set up placeholder deployment configurations for the three microservices
     + Configure the deployment configurations using the ConfigMaps
     + Set deployment hooks to populate the database for the back end services
     + Set up liveness and readiness probes
     + Expose and label the services properly (**parksmap-backend**)
   * **setup\_prod.sh**: This script needs to do the following in the **$GUID-parks-prod** project:
     + Grant the correct permissions to the Jenkins service account
     + Grant the correct permissions to pull images from the development project
     + Grant the correct permissions for the **ParksMap** application to read back-end services (see the associated README file)
     + Set up a replicated MongoDB database via **StatefulSet** with at least three replicas
     + Set up blue and green instances for each of the three microservices
     + Use ConfigMaps to configure them
       - Set **APPNAME** to the following values—the grading pipeline checks for these exact strings:
         * **MLB Parks (Green)**
         * **MLB Parks (Blue)**
         * **National Parks (Green)**
         * **National Parks (Blue)**
         * **ParksMap (Green)**
         * **ParksMap (Blue)**
     + Make the **Green** service active initially to guarantee a **Blue** rollout upon the first pipeline run
   * **reset\_prod.sh**: This script needs to do the following in the **$GUID-parks-prod** project:
     + Reset the three microservices to the **Green** version to guarantee a **Blue** rollout upon the first pipeline run
5. Implement the pipelines for each microservice (just like in class). The homework skeleton project has a shell for you to fill in.
6. Execute the pipelines via pipeline build configurations in your Jenkins project.
   * Once all pipelines succeed, you are ready to test the environment.

4. Homework Submission

It is a good idea to make sure that the automatic grading pipeline succeeds **before** submitting your homework assignment.

1. Create another Jenkins project in your cluster, such as **grading-jenkins**.
2. Allow the **jenkins** service account to create projects:

oc adm policy add-cluster-role-to-user self-provisioner system:serviceaccount:grading-jenkins:jenkins

1. Create a persistent Jenkins with 2Gi memory in that project.
2. Create a new item in Jenkins of type **Pipeline**.
3. Use the following settings in the item:
   1. Check the **This build is parameterized** box.
   2. Create the following six parameters:

| **Parameter** | **Type** | **Description** |
| --- | --- | --- |
| **GUID** | String | * + - GUID to prefix all projects |
| **USER** | String | * + - OPENTLC user ID     - Example: **wkulhane-redhat.com** set as **admin** on all created projects |
| **REPO** | String | * + - Full URL to the public Homework repository     - Gogs or GitHub |
| **CLUSTER** | String | * + - Cluster base URL     - Example: **na39.openshift.opentlc.com** |
| **SETUP** | Boolean | * + - Default: **true**     - If **true**, creates all necessary projects     - If **false**, assumes that projects are already there and only pipelines need to be executed. Also executes **reset\_prod.sh** to reset the production project to green service to ensure the first run is a green to blue deployment. |
| **DELETE** | Boolean | * + - Default: **true**     - If **true**, deletes all created projects after a successful run |

1. Use **https://github.com/wkulhanek/advdev\_homework\_grading** as the Git repository and **Jenkinsfile** as the Jenkinsfile.
2. Run the pipeline and provide the appropriate parameters for your environment.

4.1. OPENTLC Grading Jenkins

If you are on the Shared cluster, you cannot grant **self-provisioner** permissions because you need **cluster-admin** permissions to do so. You can use the official OPENTLC Shared Grading Jenkins, instead.

When you order the **OpenShift AdvDev Homework** environment, you also get permission to view the grading Jenkins. This means that you can go to Jenkins and kick off a grading pipeline yourself.

1. Access the OPENTLC Jenkins [here](http://bit.ly/gpte-jenkins).
2. Click **Grade Advanced Development Homework**, select **Build with Parameters**, fill in your specific values, and build.
   * The created projects will have the user ID that you specify as the **admin** user so you can examine what is going on during the build.

**Grading Pipeline**

The pipeline does the following:

* Checks out your source code repository
* Calls your shell scripts to do the following:
  + Create projects
  + Set up Nexus
  + Set up SonarQube
  + Set up Jenkins
  + Set up the development project
  + Set up the production project
* Executes the pipelines for your three microservices for the initial (green) deployment.
* Checks that the services return the correct application name
* Executes the pipelines again to execute the blue deployment
* Checks again that the services return the correct application name
* Deletes the projects

If the pipeline completes successfully you have passed the homework assignment.

4.2. Hints

* Once your infrastructure is set up correctly, you may set the **SETUP** variable to false for subsequent runs. This causes the pipeline to skip the initial project setup and call **reset\_prod.sh** instead to reset to a defined state for the two pipeline runs.
* It is usually a good idea to leave **DELETE** set to **false** while you are testing your pipelines.
* The created projects are annotated with your ${user} - which means they will be deleted when the homework environment gets deprovisioned.

Build Version: 98391d492c70bf60a6900485d3134768c5a5bfd7 : Last updated 2018-07-25 03:37:02 EDT