Stone Door Group OpenShift Technical Workshop

Migrating dotNet Applications to OpenShift

28 January 2020
Heist Brewery, Charlotte, North Carolina

Exercise 1: Verifying Access to the Environment

Log into the web console

1. Open your web browser and connect to the following URL.

   https://console-openshift-console.apps.opendemo.ocp-1.stonedoor.io

2. Select the **dotNet Lab** identity provider on the login page.
3. Enter the user name and password provided by your instructor.
4. Create a project named **userXX-test** (where **XX** is your user number) to verify your ability to create projects.

   Note: The **userXX** notation will be used throughout this document. Always replace the **XX** with your unique user number.

Log into the bastion host

5. Using your SSH client, connect to the following host and provide the same user name and password as in step 3 above.

   bastion.ocp-1.stonedoor.io

Connect to the OpenShift cluster using the ‘oc’ command from the bastion host

To connect to the OpenShift cluster on the command line, you will supply the URL of the OpenShift API which may be different than the web console URL.

6. Use the following commands to log into the OpenShift cluster and confirm that you can see the project created in step 4 above.

   Note: The backslash “\” is used to escape (ignore) the carriage return so that all the text below is considered to be on the same line.

   ```
   oc login -u userXX --insecure-skip-tls-verify=true \
      https://api.opendemo.ocp-1.stonedoor.io:6443

   oc whoami
   oc get projects
   ```

Exercise 2: Deploying a dotNet Core Application using an OpenShift Template

Create a New Project for your Application

1. Use the following command from the bastion host to create your new project.

   oc new-project userXX-dotnet-template

2. Confirm that the project was created by running the following command.

   oc get projects

3. You can also confirm that the project was created by looking at the **Projects** page in the web console.

   https://console-openshift-console.apps.opendemo.ocp-1.stonedoor.io/k8s/cluster/projects

Deploy the Application

4. Navigate to the application catalog for your project (change the XX in the URL to your student number).

   https://console-openshift-console.apps.opendemo.ocp-1.stonedoor.io/catalog/ns/userXX-dotnet-template

5. Select the **.NET Core** template
6. Review the information about the application to be deployed and click **Create Application.**
7. Change the builder image version to **2.1**, click **Try Sample** in the Git Repo URL section to select the sample application, and then click **Create**.
8. On the left menu, click the **Developer** item and select **Administrator**.
9. Select **Workloads > Pods**. You should see a pod named **dotnet-1-build**.
10. Click the **dotnet-1-build** pod and select the **Logs** tab. This will show you the log output of the application build process.
11. When the build has completed, you should see a message similar to the following.

    ```
    Successfully pushed
    ```

```
image-registry.openshift-image-registry.svc:5000/userXX-dotnet-template/dotnet@sha25
6:324fdcca34ddca1efe14c9b97edf9b4eb675722da5c5defd93cfd54e37a7aedf
Push successful
```

12. When you see the above message, click **Workloads > Pods** again and confirm that you have a running application pod. It will have the same first two terms of the build pod (dotnet-1) but will then have a five character random string. It will look similar to the following.

    dotnet-1-bk9z5

13. Connect to the route for the application and confirm that the web site is reachable from a browser.

    http://dotnet-userXX-dotnet-template.apps.opendemo.ocp-1.stonedoor.io/

Exercise 3: Working with the Deployed Application

Gathering Information

1. From the bastion host command line, run the following command to see some of the resources created by the application template in exercise 2.

   oc get all

   You should see the following resources:
   - Pod
   - Service
   - Route
   - ImageStream
   - BuildConfig
   - Build
   - DeploymentConfig

2. In addition to the above resources, you can see other resources and information with the following commands. Run these commands to learn more about these resources.

   ```
   oc get serviceaccounts
   oc get secrets
   oc get events
   oc describe pod <name of pod>
   oc describe service dotnet
   ```

oc describe deploymentconfig dotnet
oc describe buildconfig dotnet

Viewing Logs

3. In exercise 2, step 10, you viewed the logs for the build pod. Navigate to the pod in the web console and view its logs.
4. Run the following command from the bastion host and compare the log output from the CLI to the log output in the web console.

   oc logs dotnet-1-RRRRR (where RRRRR is your 5 character random string)

Interacting with Containers

5. Run the following commands to log into your running container

   oc rsh dotnet-1-RRRRR

6. Switch to the **bash** shell.

   exec bash

7. Note the running container's UID and GID.

   id

   Notice that the UID is a non-privileged user and that the GID is root. This is the standard configuration for containers running in OpenShift.

8. Note the running processes.

   ps -ef

9. Exit the pod and delete it. Watch the replication controller spin up a new pod.

   exit
   oc delete pod dotnet-1-RRRRR
   oc get pods

Making Changes at Run Time

10. Run the following command to edit the DeploymentConfig.

Note: The default editor in this configuration will be **vi**. If you are not familiar with vi, you may skip this step.

```
oc edit dc dotnet
```

11. On line 27, change the number of replicas to **2**. Save and quit, then observe the second pod spinning up automatically with the following command.

    ```
    oc get pods -w
    ```

    The **-w** switch means watch. Updates to the status of pods will be shown on the command line. When you seen the new pod running, you can break out of the command with **Ctrl-C** and get the current state of the pods with **oc get pods** without the **-w** switch.

12. Run the following command to change the number of running pods without editing the DeploymentConfig.

    ```
    oc scale --replicas=3 dc dotnet
    oc get pods
    ```

Exercise 4: Cloning an Application into Your Own Git Repository

Create a User Account

If you already have an account on github.com, gitlab.com, or another public git repository, you may use that account for these lab exercises. If you do not already have an account, you can sign up for one at gitlab.com at the following URL.

https://gitlab.com/users/sign_up

Create Your Repository

1. Create a new repository called **dotnet-sdg-heist**. This should be a public repository so that you do not need to configure authentication for these exercises. Select **Initialize this Repository with a README** (or similar for your git server).
2. On the bastion host, clone the repo. You can get the clone URL from the repository itself by clicking the **Clone** button and selecting **Clone with HTTPS**.

```
cd $HOME
git clone <URL>
```

You should have a directory in your bastion host home directory named **dotnet-sdg-heist**

Clone the Original Repository to the Bastion Host

3. Run the following command from your home directory on the bastion host to clone the example application.

   ```
   git clone https://github.com/redhat-developer/s2i-dotnetcore-ex.git
   ```

   You should now have a directory named **s2i-dotnetcore-ex** in your bastion host home directory.

Push the Code to Your Repository

4. Copy the code from the **s2i-dotnetcore-ex** directory to the **dotnet-sdg-heist** directory.

   ```
   cp -r ~/s2i-dotnetcore-ex/* ~/dotnet-sdg-heist/
   ```

5. Upload the code to your git server.

   ```
   cd ~/dotnet-sdg-heist/
   ```

   Update the git config with your information (must match your git login)

   ```
   git config --global user.email "you@example.com"
   git config --global user.name "Your Name"
   git config --global push.default simple
   ```

   Add code, commit, and push.

   ```
   git add *
   git commit -m “initial upload”
   git push
   ```

   Enter your username and password when prompted.

Exercise 5: Using the ‘oc new-app’ Command to Deploy an Application

Create a New Project for this version of the Application

1. From the bastion host command line, run the following command to create a new project.

```
oc new-project userXX-new-app
```

Deploy the Application

2. Run the following command to deploy the dotNet code from the git repository you created in exercise 4.

```
oc new-app  dotnet:2.1~https://<repo_url> --context-dir=/app
```

Note: include the **.git** extension the repo_url.

Expose the Service

The template that was deployed in exercise 2 included a route for the application. The **oc new-app** command will create several of the resources to support an application on OpenShift but it does not create a route. You will have to create a route by exposing the service with the following command.

```
oc expose svc dotnet-sdg-heist
```

Verify the Application Deployment

3. Get the route for the application and connect to it in your browser. Do you see the application?

Exercise 6: Rebuilding an Application after Code Change (manually)

Update Code and Push it to Repository

1. Edit the home page of the application and change the **Overview** heading to **My Overview.** If you are not familiar with **vi** or other GNU/Linux editors, **pico** is relatively easy to use.

   ```
   cd ~/dotnet-sdg-heist
   pico app/Views/Home/Index.cshtml
   ```

   ctrl-w (opens search box)
   overview (search for overview)

   change to **My Overview**

   ctrl-x (exit)
   Y (confirm save)

2. Add, commit, and push the update.

   ```
   git add app/Views/Home/Index.cshtml
   git commit -m "updated Overview heading"
   git push
   ```

Use ‘oc start-build’ to Re-build the Application

3. Run the following command to start a new build with the updated code.

   ```
   oc start-build buildconfig.build.openshift.io/dotnet-sdg-heist
   ```

4. You can run the following command to follow the build progress.

   ```
   oc logs -f build/dotnet-sdg-heist-2
   ```

Verify the Change

5. Connect to the route for the application after the new pods are launched and confirm the update to the web page.

Exercise 7: Deploying Jenkins to Support a CI/CD Pipeline

Clone the container-pipelines Repository

1. Run the following commands to clone the container-pipelines repository.

   ```
   cd $HOME
   git clone https://github.com/redhat-cop/container-pipelines.git
   ```

Deploy the Jenkins Pipeline

2. Create projects for each stage in development.

   ```
   oc new-project userXX-dotnet-jenkins-build
   oc new-project userXX-dotnet-jenkins-dev
   oc new-project userXX-dotnet-jenkins-stage
   oc new-project userXX-dotnet-jenkins-prod
   ```

3. Deploy the **jenkins-ephemeral** template to the build project.

   ```
   oc process openshift//jenkins-ephemeral \
   | oc apply -f - -n userXX-dotnet-jenkins-build
   ```

4. Instantiate the pipeline by applying the deployment template to the dev, stage, and prod projects.

   ```
   cd ~/container-pipelines/basic-dotnet-core/
   oc process -f .openshift/templates/deployment.yml \
       -p APPLICATION_NAME=userXX-dotnet-jenkins \
       -p NAMESPACE=userXX-dotnet-jenkins-dev \
       -p SA_NAMESPACE=userXX-dotnet-jenkins-build \
       -p READINESS_PATH="/" \
       -p READINESS_RESPONSE="status.:.UP" \
       | oc apply -f -

   oc process -f .openshift/templates/deployment.yml \
       -p APPLICATION_NAME=userXX-dotnet-jenkins \
       -p NAMESPACE=userXX-dotnet-jenkins-stage \
       -p SA_NAMESPACE=userXX-dotnet-jenkins-build \
   ```

```
    -p READINESS_PATH="/" \
    -p READINESS_RESPONSE="status.:.UP" \
    | oc apply -f -

oc process -f .openshift/templates/deployment.yml \
    -p APPLICATION_NAME=userXX-dotnet-jenkins \
    -p NAMESPACE=userXX-dotnet-jenkins-prod \
    -p SA_NAMESPACE=userXX-dotnet-jenkins-build \
    -p READINESS_PATH="/" \
    -p READINESS_RESPONSE="status.:.UP" \
    | oc apply -f -
```

5. Deploy the BuildConfigs to build the application.

```
oc process -f .openshift/templates/build.yml \
    -p APPLICATION_NAME=userXX-dotnet-jenkins \
    -p NAMESPACE=userXX-dotnet-jenkins-build \
    -p
SOURCE_REPOSITORY_URL="https://github.com/redhat-cop/container-pipelines.
git" \
    -p
APPLICATION_SOURCE_REPO="https://github.com/redhat-developer/s2i-dotnetc
ore-ex.git" \
    | oc apply -f -
```

Verify Pipeline Deployment

6. Log into Jenkins at the following URL (select **Log in with OpenShift**, select **Allow selected permissions**, and supply your OpenShift credentials).

   https://jenkins-userXX-dotnet-jenkins-build.apps.opendemo.ocp-1.stonedoor.io

7. Click the following link to monitor the progress of the pipeline.

   userXX-dotnet-jenkins-build >
   userXX-dotnet-jenkins-build/userXX-dotnet-jenkins-pipeline

8. You can also monitor the progress by accessing the OpenShift web console and clicking **Builds > Builds** in the **Administrator** view. On the Builds page, click **userXX-dotnet-jenkins-pipeline-1** and then click the **View Logs** link to follow the progress.
9. Confirm that the pipeline deploys the application into the “prod” project by accessing the route at the completion of the pipeline.

Exercise 8: Manually Building a dotNet Core Container Image

Prepare the dotNet Core Application for Containerization

1. Restore the dependencies for the s2i application.

```
cd $HOME/s2i-dotnetcore-ex/app
dotnet restore -r rhel.7-x64
```

2. Publish the application.

```
dotnet publish \
    -f netcoreapp2.1 \
    -c Release \
    -r rhel.7-x64 \
    --self-contained false \
    /p:PublishWithAspNetCoreTargetManifest=false
```

Build the dotNet Core Container Image

3. Create the Dockerfile

```
cat > Dockerfile <<EOF
FROM registry.access.redhat.com/dotnet/dotnet-21-runtime-rhel7:latest
ADD app/bin/Release/netcoreapp2.1/rhel.7-x64/publish/. .
CMD [ "dotnet", "app.dll" ]
EOF
```

4. Create an OpenShift Project for Deploying the Pre-built Container Image

```
oc new-project userXX-dotnet-image
```

5. Build the docker image.

```
docker build -t
image-registry-openshift-image-registry.apps.opendemo.ocp-1.stonedoor.io/userX
X-dotnet-image/dotnet-ex:latest .
```

Note: There is a period (.) at the end of the above line indicating to the docker command that it should find the Dockerfile in the current directory.

Exercise 9: Deploying a Pre-built dotNet Core Container Image

Push the dotNet Core Image into the OpenShift Container Registry

1. Get your login token with the following command.

   ```
   oc whoami -t
   ```

2. Log into the image registry.

   ```
   docker login -u userXX -p <your token>
   image-registry-openshift-image-registry.apps.opendemo.ocp-1.stonedoor.io
   ```

3. Push the dotNet Core Container Image into OpenShift

   ```
   docker push
   image-registry-openshift-image-registry.apps.opendemo.ocp-1.stonedoor.io/userX
   X-dotnet-image/dotnet-ex:latest
   ```

Deploy the Container Image as an Application

4. Run the following command to deploy the application.

   ```
   oc new-app --image-stream=userXX-dotnet-jenkins
   ```

5. Verify that the application deployed properly.