Lab 2: Set-up Toolchain for Catalog

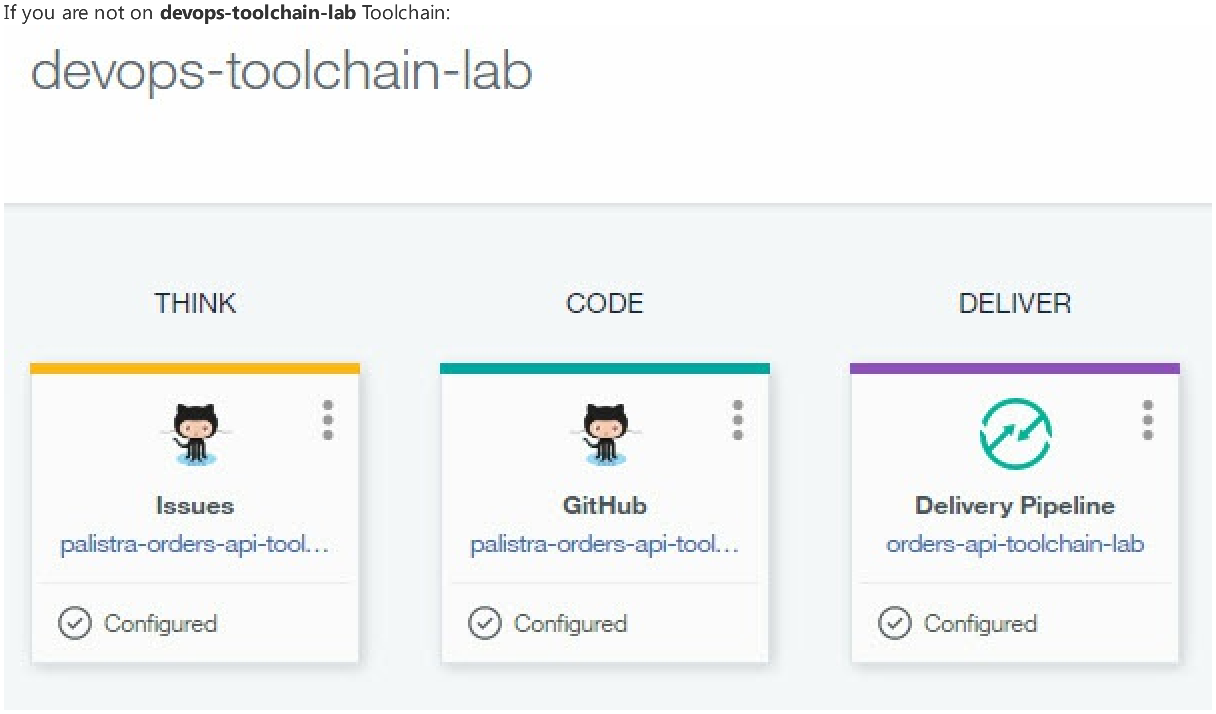
Objective

This lab adds the Catalog application to the Toolchain. And now that you have experience working with Toolchains and Pipelines, the next lab does not go into the same level of detail. You may want to refer to the prior lab if you need additional details as you will be creating the same set of artifacts for the Catalog application.

Tasks:

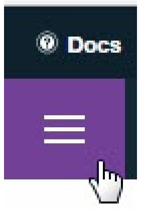
Task 1: Go to devops-toolchain

1. If you are not on **devops-toolchain-lab** Toolchain:



perform the following steps:

1. Click on the Bluemix Menu Bar



2. Click **Services**
3. Select DevOps to display all the Toolchains.
4. Click on the **devops-toolchain-lab** tile.

Task 2: Add and Configure GitHub Integration for Catalog

The code for the Catalog microservice already exists in a GitHub repository (https://github.com/open-toolchain/Microservices_CatalogAPI). We will clone this repository and link to the clone.

1. Click on the **+** plus icon on the right side of the screen to add a Tool Integration.
2. Click on **GitHub** to add integration with GitHub to the Toolchain.
3. Select 'Clone' as the Repository type.
4. Enter "*githubuserid*/catalog-api-toolchain-lab.git" for the New Repository Name.
   where *githubuserid* is your GitHub userid.
5. Enter "https://github.com/open-toolchain/Microservices_CatalogAPI" for the Source repository URL.
6. Click **Create Integration**.

Task 3: Add Catalog Delivery Pipeline

Now that you have a Git repository clone of the code, we will add a Delivery Pipeline to deploy it and test it.

1. Click on the **+** plus icon on the right side of the screen to add a Tool Integration.
2. Click on **Delivery Pipeline** to create a new Delivery Pipeline (we will add tool integrations to this).
3. Under 'Pipeline name:', enter "catalog-api-toolchain-lab" and select the 'Show apps in the VIEW APP menu' checkbox.
4. Click **Create Integration**.
5. The catalog-toolchain-lab delivery pipeline is displayed.

Task 4: Configure Catalog Delivery Pipeline

1. Now to configure the catalog-api-toolchain-lab delivery pipeline. Four stages will be added: Build, Dev, Test and Prod.
   - The **Build** stage has one job, performing the initial build of the code from the GitHub Repository.
   - The **Dev** stage has two jobs, taking the output from the Build stage and deploying on Bluemix into the *dev* space, then performing automated functional tests.
   - The **Test** stage has two jobs, taking the output from the Dev stage and deploying on Bluemix into the *qa* space, then performing automated tests.
   - The **Prod** stage has one job, taking the output from the Test stage and deploying on Bluemix into the *prod* space. This stage will also check to see there is an earlier instance of this application running and if it is, keep it around in case the deploy of the new version of the app has problems. If the new version deploys successfully, the old version is deleted. If not, the new version is deleted and the old version continues to run.

   Click on the **Delivery Pipeline** tile for the catalog-api-toolchain-lab delivery pipeline.

2. Add the **Build** stage and jobs.
   1. Click on **ADD STAGE**.
   2. On the **INPUT** tab, enter "Build" for Stage Name. Note that:
      - 'Input Type' is set to a SCM Repository, in this case, Git.
      - 'Git Repository' is set to the name of the Git Repository we just cloned.
      - 'Git URL' is set to the URL of the Git Repository we just cloned.
      - 'Branch' is set to "Master".
      - 'Stage Trigger' is set to "Run jobs whenever a change is pushed to Git", resulting in the Build stage running continuously when Git is updated.
   3. Click the **Jobs** tab.
   4. Click **ADD JOB**.
   5. Click the **+** and select **Build** for the JOB TYPE.
   6. On the Job configuration panel, note that:
      - 'Builder Type' is set to "Simple" (other options are available on the pull-down).
      - 'Run Conditions' is set to "Stop running this stage if this job fails" to prevent any other jobs in this stage from running and to make the stage failed is this Job fails.
   7. Click **Save** to save the **Build** stage.
   8. The **Delivery Pipeline** displays the **Build** stage. This stage has not been run. Click on the **Run Stage** icon to run the build.
      The JOBS section shows the Build was successful.

      The **Build** stage has been successfully added and executed.

3. Add the **Dev** stage and jobs (remember, two jobs, taking the output from the Build stage and deploying on Bluemix into the *dev* space, then performing automated functional tests).
   1. Click on **ADD STAGE**.
   2. On the **INPUT** tab, enter "Dev" for Stage Name. Note that:
      - 'Input Type' is set to Build Artifacts (from the **Build** stage).
      - 'Stage' and 'Job' are both 'Build'.
      - 'Stage Trigger' is set to "Run jobs when the previous stage is completed", resulting in the Dev stage running when the **Build** stage successfully completes.
   3. Click the **Jobs** tab.
   4. Click **ADD JOB**.
   5. Click the **+** and select **Deploy** for the JOB TYPE.
   6. On the Job configuration panel, note that:
      - 'Deployer Type' is set to "Cloud Foundry" (other options are available on the pull-down).
      - 'Target' is set to "US South - https://api.ng/bluemix.net" as this is where the code will be deployed.
      - 'Space' is set to "dev" (or Create a new space called **dev** if not on the dropdown).
      - 'Application Name' is "catalog-api-toolchain-lab".
      - Type the following into the "Deploy Script" section.

        ```
        #!/bin/bash
        #get user name
        a=$(cf services | grep @)
        b=${a%@*}
        c=($b)
        len=${#c[@]}
        user_name=${c[len-1]}
        #add Cloudant service
        cf create-service cloudantNoSQLDB Shared myMicroservicesCloudant
        # Push app
        export CF_APP_NAME="$user_name-dev-$CF_APP"
        cf push "${CF_APP_NAME}"
        echo "Pushed App Name: ${CF_APP_NAME}."
        export APP_URL=http://$(cf app $CF_APP_NAME | grep urls: | awk '{print $2}')
        # View logs
        #cf logs "${CF_APP_NAME}" --recent
        ```

        This will create and deploy the cloudantNoSQLDB service, update the APP_URL environment variable for use by the functional test job and deploy the Catalog application.

      - 'Run Conditions' is set to "Stop running this stage if this job fails" to prevent any other jobs in this stage from running and to make the stage failed is this Job fails.
   7. The bash script just entered into the Deploy Script references both the *CF_APP_NAME* and *APP_URL* environment variable ($CF_APP is provided by default). These need to be added to the environment variables as Text. Click the **ENVIRONMENT PROPERTIES** tab.
   8. Click **ADD PROPERTY** and select **Text Property**.
   9. Enter "CF_APP_NAME" as the 'Name'. Do not enter anything for the 'Value'.
   10. Click **ADD PROPERTY** and select **Text Property**.
   11. Enter "APP_URL" as the 'Name'. Do not enter anything for the 'Value'.
   12. Click the **Jobs** tab.
   13. Click **ADD JOB**.
   14. Click the **+** and select **Test** for the JOB TYPE.
   15. On the Job configuration panel, note that:
       - Job name **Functional Tests**.
       - 'Tester Type' is **Simple**. Enter the following code to the **Test Command**.

         ```
         #!/bin/bash
         export CATALOG_API_TEST_SERVER=$APP_URL
         GRUNTFILE="tests/Gruntfile.js"
         ```

```
if [ -f $GRUNTFILE ]; then
  npm install -g npm@3.7.2 ### work around default npm 2.1.1 instability
  npm install
  grunt dev-fvt --no-color -f --gruntfile $GRUNTFILE --base .
 else
   echo "$GRUNTFILE not found."
fi
```

16. Click **Save** to save the **Dev** stage.
17. The **Delivery Pipeline** displays the **Build** and **Dev** stages. The **Dev** stage has not been run. Click on the **Run Stage** icon to run the **Dev** stage to deploy Catalog application and run the functional tests.
    The JOBS section shows the Stage was successful. Click on "View logs and history" to the Job log.
18. LAST EXECUTION RESULT displays the url to the successfully deployed application (*user_name*-dev-catalog-api-toolchain-lab.mybluemix.net) as well as a link to the runtime log.
    Click on "*user_name*-dev-catalog-api-toolchain-lab.mybluemix.net" to access the running application.
    The **Dev** stage has been successfully added and executed.

- Add the **Test** stage (remember, two jobs, one to deploy to the *test* space and another to perform an automated test). We will clone the **Dev** stage and make some modifications.

1. Ensure the catalog-api-toolchain-lab **Delivery Pipeline** is displayed.
2. On the **Dev** stage, click the **Stage Configuration** and select "Clone Stage".
3. Rename the cloned stage to **Test** (from **Dev [copy]**).
4. On the **Jobs** tab, for the **Deploy** job, change the space to **qa** (from **dev**) (or Create a new space called **qa** if not on the dropdown) and change the deploy script to change CF_APP_NAME to "$user_name-test-$CF_APP" from "$user_name-dev-$CF_APP".
5. On the **Jobs** tab, for the **Functional Test** job, change the Functional Test name to simply **Test**. Enter the following code to the **Test Command**.

```
#!/bin/bash
# invoke tests here
echo "Testing of App Name ${CF_APP_NAME} was successful"
```

6. Click **Save** to save the **Test** stage.
7. The **Delivery Pipeline** displays the **Build** and **Dev** stages. The **Test** stage has not been run. Click on the **Run Stage** icon to run the **Test** stage and deploy the order API to the *test* space.
8. As before for the **Dev** stage, the JOBS section shows the Deploy and Test Jobs were successful. Click **Test** to display the log for the **Test** job.
   Click on "test-catalog-toolchain-lab.mybluemix.net" to access the running application.
   The **Test** stage has been successfully added and executed.

- Add the **Prod** stage (remember, one job, to deploy to the *prod* space). This stage will also check to see there is an earlier instance of this application running and if it is, keep it around in case the deploy of the new version of the app has problems. If the new version deploys successfully, the old version is deleted. If not, the new version is deleted and the old version continues to run.
We will clone the **Dev** stage and make some modifications.

1. Ensure the catalog-api-toolchain-lab **Delivery Pipeline** is displayed.
2. On the **Dev** stage, click the **Stage Configuration** and select "Clone Stage".
3. Rename the cloned stage to **Prod** (from **Dev [copy]**).
4. On the **Jobs** tab, change the Job name to 'Blue/Green Deploy', change the space from **dev** to **prod** (or Create a new space called **prod** if not on the dropdown) and change the deploy script to the following:

```
#!/bin/bash
#get user name
a=$(cf services | grep @)
b=${a%@*}
c=($b)
len=${#c[@]}
user_name=${c[len-1]}
export CF_APP_NAME="$user_name-prod-$CF_APP"
#add Cloudant service
cf create-service cloudantNoSQLDB Shared myMicroservicesCloudant
if ! cf app $CF_APP_NAME; then
```

```
  cf push $CF_APP_NAME
else
  OLD_CF_APP=${CF_APP_NAME}-OLD-$(date +"%s")
  rollback() {
    set +e
    if cf app $OLD_CF_APP; then
      cf logs $CF_APP_NAME --recent
      cf delete $CF_APP_NAME -f
      cf rename $OLD_CF_APP $CF_APP_NAME
    fi
    exit 1
  }
  set -e
  trap rollback ERR
  cf rename $CF_APP_NAME $OLD_CF_APP
  cf push $CF_APP_NAME
  cf delete $OLD_CF_APP -f
fi
```

5. Click **Save** to save the **Prod** stage.
6. Click on **Run Stage** to run the **Prod** stage and deploy the order API to the *prod* space.
7. The JOBS section shows the Deploy was successful. Inspect the Job log.
   Click on the blue arrow to display the Delivery Pipeline. Click on "*user_name*-catalog-api-toolchain-lab.mybluemix.net" to access the running application.
   The **Prod** stage has been successfully added and executed. The Catalog application has been deployed to production.