Lab 3: Create Catalog Toolchain by Hand

Objective

This lab manually creates a simple toolchain for the Catalog API microservice and then configures it. It assumes that a Bluemix Organization and *dev*, *qa* and *prod* Spaces are already created.

Tasks:

- Task 1: Create Toolchain
- Task 2: Add and Configure GitHub Integration
- Task 3: Add Eclipse Orion Web IDE to Toolchain
- Task 4: Add Catalog Delivery Pipeline
- Task 5: Add Build stage to Catalog Delivery Pipeline
- Task 6: Add Dev stage to Catalog Delivery Pipeline
- Task 7: Add Test stage to Catalog Delivery Pipeline
- Task 8: Add Prod stage to Catalog Delivery Pipeline

Task 1: Create Toolchain

Throughout the lab, the phrase *timestamp* is used to indicate the same timestamp string that was appended to *simple-order-toolchain*. While a timestamp string is not required, it does help make the name of the created objects unique.
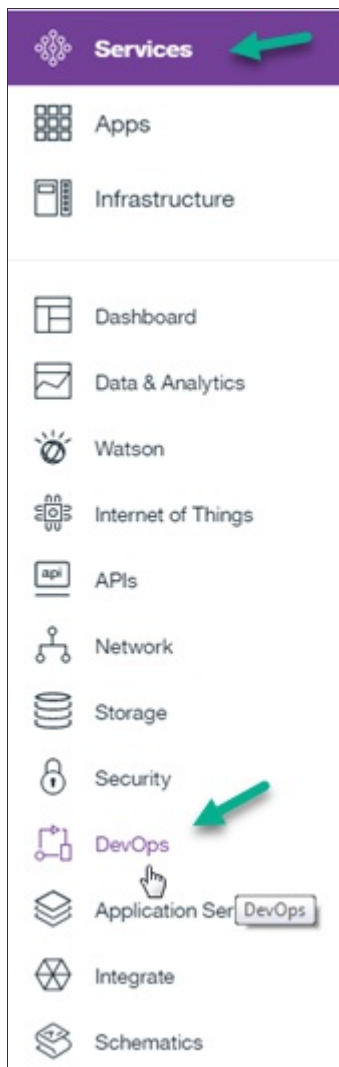
1. If you are not already logged into IBM Bluemix, log into IBM Bluemix (https://www.ibm.com/cloud-computing/bluemix/).
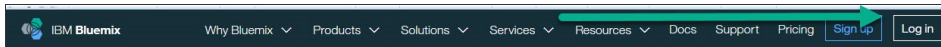
2. If you don't see a button called *Create a Toolchain*, you need to get to DevOps Services. Click on the **Bluemix menu bar** in the upper left corner.
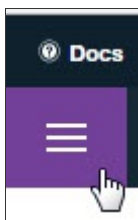
    and click on **Services** then **DevOps**

and click on **Toolchains**.

3.   Click **Create a Toolchain**.

4.   Click on **Build your own toolchain**.

5.   Change the *Toolchain Name* from *empty-toolchain-timestamp* to **catalog-toolchain-*timestamp***.

6.   Click **Create** to create the Toolchain.

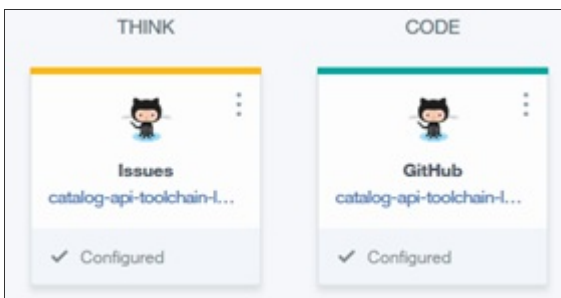7.   The (empty) *catalog-toolchain-timestamp* is displayed.

Task 2: Add and Configure GitHub Integration

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 **Add a Tool** 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 **catalog-api-toolchain-lab-*timestamp*** for the *New Repository Name*.
5. Enter *https://github.com/open-toolchain/Microservices_CatalogAPI* for the Source repository URL.

Repository type:

Clone

Clone the repository that is specified in the Source repository URL field.

New repository name:

catalog-api-toolchain-lab-1489593492738

Source repository URL:

https://github.com/open-toolchain/Microservices_CatalogAPI

☑ Enable GitHub Issues

6. Click on **Create Integration**. The integration is created.

THINK	CODE
Issues	GitHub
catalog-api-toolchain-l...	catalog-api-toolchain-l...
✓ Configured	✓ Configured

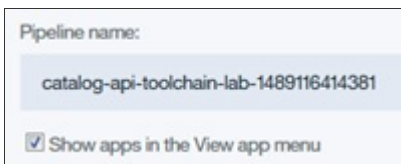## Task 3: Add Eclipse Orion Web IDE to Toolchain

If we want to modify the application, one convenient way is to use the Eclipse Orion Web IDE.

1. On the *catalog-toolchain-timestamp* toolchain's Tool Integrations page, click **Add a Tool**
2. Click **Eclipse Orion Web IDE**.
3. No configuration is needed, so click **Create Integration**.

Task 4: Add Catalog Delivery Pipeline

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

1. Click on **Add a Tool** 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 Stages and Jobs to this).
3. For 'Pipeline name:', enter "**catalog-api-toolchain-lab-*timestamp***" and ensure 'Show apps in the VIEW APP menu' checkbox is checked.

4. Click **Create Integration**.
5. The *catalog-api-toolchain-lab-timestamp* Delivery Pipeline is created.

## Task 5: Add Build stage to Catalog Delivery Pipeline

Now to configure the *catalog-api-toolchain-lab-timestamp* Delivery Pipeline. We will make this pipeline a little more complex. We will add four stages: Build, Dev, Test and Prod.

- The **Build** stage has two jobs, performing the initial build of the code from the GitHub Repository then some unit tests.
- 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 perform a Blue-Green deployment, checking 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.

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

2. Click **Add Stage**.

3. This is the *INPUT* portion of the stage. Note that the *Input Type* is set to Git Repository_ and the *Git Repository*, *Git URL* and (Git) *Branch* are pre-filled. Also, *Stage Trigger* is set to "Run jobs whenever a change is pushed to Git", resulting in this stage running when Git is updated.

4. Rename the stage from *MyStage* to **Build**.

5. Click the *JOBS* tab so we can add some jobs.
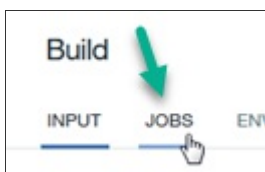
6. Click the **+** and select **Build** for the JOB TYPE.
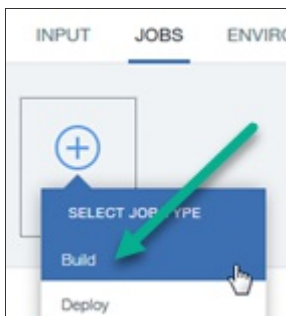
On the Build configuration panel, note that:

○ The job name is *Build* (just like the stage name.)

- ○ *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 mark the stage failed if this Job fails.

7.   Click **ADD JOB**, this time selecting **Test** for the JOB TYPE.

8.   Rename the job from *Test* to *Unit Tests*.

9.   Enter the following for the *Test Command*. *Note:* You can enter the following URL into another browser tab to display the code for easy copy and pasting: http://ibm.biz/CatalogAPIDevUnitTest

```
#!/bin/bash
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-test-cov --no-color -f --gruntfile $GRUNTFILE --base .
else
  echo "$GRUNTFILE not found."
fi
```

This script checks to see if the file *tests/Gruntfile.js* exists. If it does, we install a version of npm then run an automated Grunt tests. If the file *tests/Gruntfile.js* does not exist, simple echo a line into the log file.

10.   Click **Save** to save the *Build* stage.

11.   The *Delivery Pipeline* displays the **Build** stage. This stage has not been run. Click on the **Run Stage** icon to run the build.

12.   The JOBS section shows the Build (job) progress. After a few moments, the **Build** stage has been successfully executed.
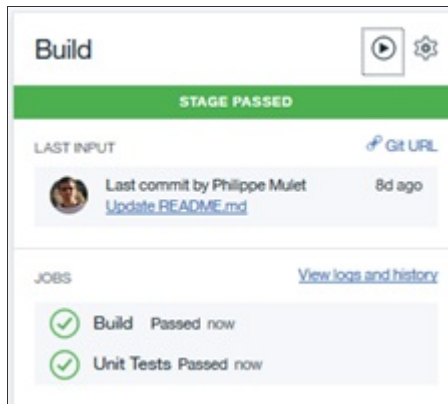
13.   Click *View logs and history* for the jobs to examine the logs for each job. When done, return to the Delivery Pipeline.

Task 6: Add Dev stage to Catalog Delivery Pipeline

Now we add the *Dev* stage and jobs. The *Dev* stage has two jobs. The first job deploys the just built Catalog API microservice and deploys it into the *dev* space on Bluemix and the second job performs some automated tests on the deployed microservice.

1.   Click on **ADD STAGE**.
2.   Name the stage **Dev**. 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 and add a new job of type **Deploy**. 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".
- *Application Name* is "catalog-api-toolchain-lab-*timestamp*".

3.   Type the following into the *Deploy Script* section. This script first creates the cloudantNoSQLDB (remember, if it already exists, the script simply continues). Then the variable *CF_APP_NAME* is set to the application name (*catalog-api-toolchain-lab-timestamp*) has the space name *dev* added to the front of the name. This keeps the name unique as we will deploy this application to the *qa* and *prod* space later. *Note:* You can enter the following URL into another browser tab to display the code for easy copy and pasting: http://ibm.biz/CatalogAPIDevDeploy

```
cf create-service cloudantNoSQLDB Lite myMicroservicesCloudant
# Push app
export CF_APP_NAME="dev-$CF_APP"
cf push "${CF_APP_NAME}"
export APP_URL=http://$(cf app $CF_APP_NAME | grep urls: | awk '{print $2}')
```

Deploy

Deploy Configuration

Deployer Type

Cloud Foundry

Target

US South - https://api.ng.bluemix.net

Organization

DevOpsLab

Space

dev

Application Name

catalog-api-toolchain-lab-1489116414381

Deploy Script

```
cf create-service cloudantNoSQLDB Lite myMicroservicesCloudant
# Push app
export CF_APP_NAME="dev-$CF_APP"
cf push "${CF_APP_NAME}"
export APP_URL=http://$(cf app $CF_APP_NAME | grep urls: | awk '{print $2}')
```

4. The bash script just entered into the Deploy Script references both the *CF_APP_NAME* and *APP_URL* environment variables (remember, *CF_APP* is provided by default). These two environment variables are used to pass information between jobs in this stage and need to be added to the environment variables as Text.

5. Click the **ENVIRONMENT PROPERTIES** tab.

6. Click **ADD PROPERTY** and select **Text Property**.

7. Enter **CF_APP_NAME** as the 'Name'. Do not enter anything for the 'Value'.

8. Click **ADD PROPERTY** and select **Text Property**.

9. Enter **APP_URL** as the 'Name'. Do not enter anything for the 'Value'.

Deploy

INPUT   JOBS   ENVIRONMENT PROPERTIES

+ ADD PROPERTY

| CF_APP_NAME | Value |
| APP_URL | Value |

10. Click the **JOBS** tab and add a new job of type **Test**.

11. Change the jobs name from *Tests* to **Functional Tests**.

12. Note that the *Tester Type* is *Simple*.

13. Enter the following code to the *Test Command* section. *Note:* You can enter the following URL into another browser tab to

display the code for easy copy and pasting: http://ibm.biz/CatalogAPIDevFVT

```bash
#!/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 --gruntfile $GRUNTFILE --base .
else
  echo "$GRUNTFILE not found."
fi
```

This bash shell runs the same functional test scripts on the catalog service but this time against the deployed application in the *dev* space.


14. Click **Save** to save the *Dev* stage.
15. The *Delivery Pipeline* displays the *Build* and *Dev* stages. The *Dev* stage has not been run. Click on the **Run Stage** icon (the right arrow in the *Dev* stage) to run the *Dev* stage, deploying the Catalog application to the *dev* space and executing the functional tests.
16. The *JOBS* section shows the Stage was successful. Click on "View logs and history" to the Stage history.
17. Stage History displays the execution history of the stage in reverse chronological order (so the most recent on top and the oldest at the bottom). Within the history of a stage execution, the job history is displayed in the order in which the job was attempted. For example, the following screen shot shows that this stage was executed twice. Within the most recent execution (*9*), the *Deploy* job was attempted (and passed) followed by the *Functional Tests* job (which also passed). Your screen will probably just have **1** attempt.
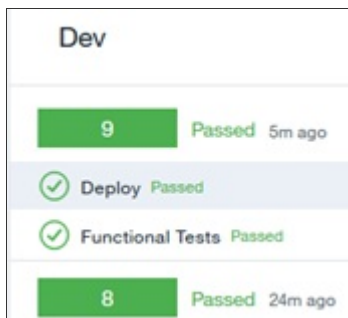
18. This display shows that the *Dev* stage ran both jobs and they both passed. Initially, the log for the *Deploy* job is displayed. Scrolling to the bottom and you see the application was deployed as *catalog-api-toolchain-lab-timestamp* into the *dev* space.

19. Scroll back to the top and click the **Test** job to display the log for it. Scroll to the bottom.
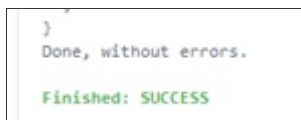

20. Return to the Delivery Pipeline.

## Task 7: Add Test stage to Catalog Delivery Pipeline

Now we add the *Test* stage and associated jobs. The *Test* stage has two jobs. The first job deploys the just built Catalog API microservice and deploys it into the *qa* space on Bluemix and the second job performs some automated tests on the deployed microservice. This time, we will clone the *Dev* stage and make some modifications.

1. On the *Dev* stage, click the *Configure Stage* gear icon and select **Clone Stage**.
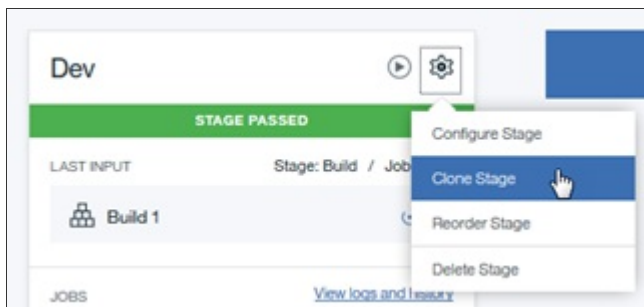
2. Rename the cloned stage to **Test** from *Dev [copy]*.
3. On the *Jobs* tab, for the *Deploy* job, change the space to **qa** from *dev*..

4. Change the *Deploy* deploy script so CF_APP_NAME gets set to **"test-$CF_APP"** from *"dev-$CF_APP"*.
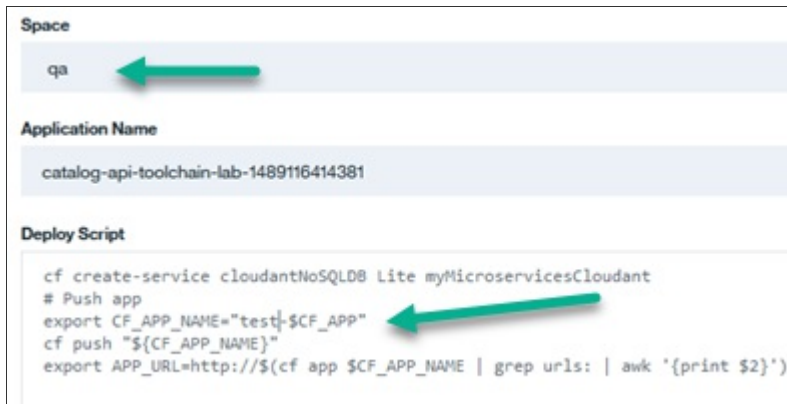
5. Switch to the *Functional Test* job.
6. Change the *Test Command* to:

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

This 'test' script just echos the app name to the console log. In a real environment, we would execute automated test tools and scripts to validate the deployed service still worked.

7. Click **Save** to save the *Test* stage.
8. The Delivery Pipeline displays the *Build*, *Dev* and *Test* 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.
9. As before for the *Dev* stage, the **JOBS** section of the *Test* stage shows the *Deploy* and *Functional Tests* jobs were successful. Click **Functional Tests** to display the log for the *Functional Tests* job. Notice the "Testing of App Name" message was echoed to the log.
10. Return to the Delivery Pipeline. Click on the application URL (http://test-catalog-api-toolchain-lab-*timestamp*.mybluemix.net/) to access the running application. Note that *test-* was added to the start of the application name.
11. Close the application browser window.
12. On the Delivery Pipeline display, click on the **View runtime log** link to examine the log for this runtime. Return to the Delivery Pipeline.
13. The **Test** stage has been successfully added and executed.

Task 8: Add Prod stage to Catalog Delivery Pipeline

Now we will add the final stage to the Delivery Pipeline, the *Prod* stage. This stage has one job, which performs a *Blue-green* deployment to the *prod* space. As you may remember from the simple Order lab, a blue-green deployment is a release technique reducing downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic.

So during deployment, this stage will 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. To do this, we will clone the *Dev* stage and make some modifications.

1. Ensure the catalog-api-toolchain-lab *catalog-api-toolchain-lab-timestamp* Delivery Pipeline is displayed.
2. Clone the *Dev* stage.

3. Rename the cloned stage to **Prod** (from *Dev [copy]*).
4. On the *Jobs* tab, change the Deploy Job name to **Blue/Green Deploy** and change the space from *dev* to **prod**
5. Change the deploy script to the following *HINT*: It is very similar to the script we used for the Order Pipeline lab, perhaps you configure that Job to copy and paste the deploy script *or* you can enter the following URL into another browser tab to display the code for easy copy and pasting: http://ibm.biz/CatalogAPIProdBlueGreenDeploy

```
#!/bin/bash
echo "Attempting to create cloudantNoSQLDB Lite myMicroservicesCloudant for use by the
microservices. It is not a problem if it already exists, we simply continue."
cf create-service cloudantNoSQLDB Lite myMicroservicesCloudant
export CF_APP_NAME="prod-$CF_APP"
# Push app
echo "If the $CF_APP_NAME does not exist, push the app."
if ! cf app $CF_APP_NAME; then
  cf push $CF_APP_NAME
else
  OLD_CF_APP_NAME=${CF_APP_NAME}-OLD-$(date +"%s")
  rollback() {
    set +e
    if cf app $OLD_CF_APP_NAME; then
      cf logs $CF_APP_NAME --recent
      cf delete $CF_APP_NAME -f
      cf rename $OLD_CF_APP_NAME $CF_APP_NAME
    fi
    exit 1
  }
  set -e
  trap rollback ERR
  echo "If the $CF_APP_NAME does exist, rename it."
  cf rename $CF_APP_NAME $OLD_CF_APP_NAME
  echo "And push out the new version."
  cf push $CF_APP_NAME
  echo "If the push is successful, delete the old app."
  cf delete $OLD_CF_APP_NAME -f
fi
# Export app name and URL for use in later Pipeline jobs
# export CF_APP_NAME="$CF_APP"
export APP_URL=http://$(cf app $CF_APP_NAME | grep urls: | awk '{print $2}')
# View logs
#cf logs "${CF_APP}" --recent
```

6. Click **Save** to save the *Prod* stage.
7. Click on **Run Stage** to run the *Prod* stage and deploy the Catalog API app to *prod* space.
8. The JOBS section of the *Blue/Green Deploy* shows the Deploy was successful. Inspect the *Blue/Green Deploy* Job log to see where the app was deployed and the *Functional Tests* Job log to ensure the tests were successful.
9. Return to the Delivery Pipeline.
10. Click the application URL in the *Prod* stage to access the running application. Note that *prod-* was added to the start of the application name. Where was that changed?
11. Close the application browser window. The **Prod** stage has been successfully added and executed, deploying the application to the *prod* space.
12. Click on the left arrow to the left of *Toolchain* to return to the _catalog-toolchain-*timestamp* page.
13. Click on the left arrow to the left of *Toolchains* to return to the _Toolchains page.