Lab 1: Set-up Toolchain for Order

Objective

This lab creates and configures the Toolchain for the Order application.

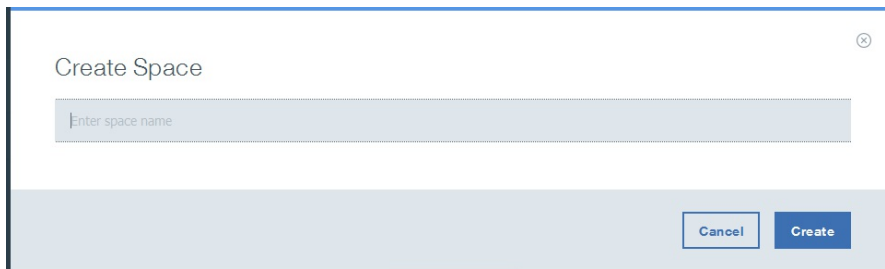**Tasks**:

Task 1: Log into IBM Bluemix and create spaces

1. If you are not already logged into IBM Bluemix, log into IBM Bluemix (https://www.ibm.com/cloud-computing/bluemix/).

2. The toolchain will be deploying into three Bluemix spaces that are named `dev, qa, and prod` . Before creating the toolchain, first create these spaces in your Bluemix organization. Froom the Bluemix console, click your account information at the top. This displays the Account, Region, Organization, and space that you are currently logged into and displaying.
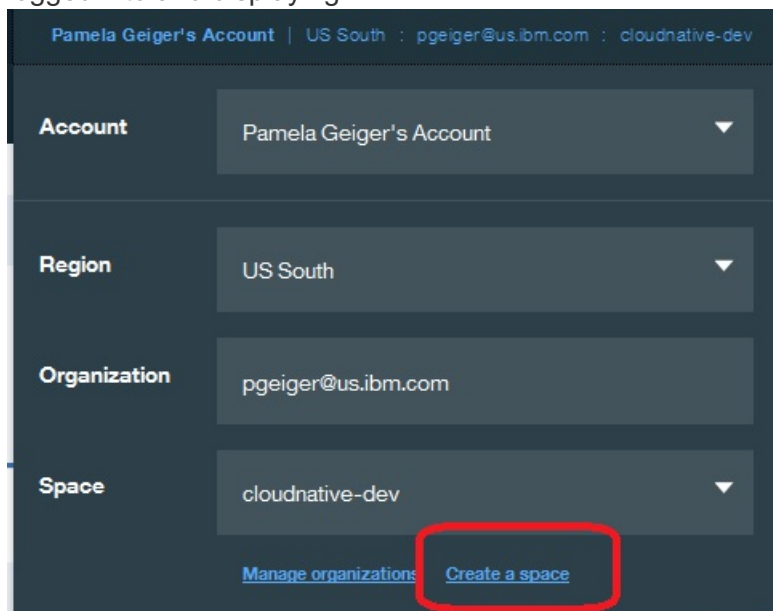
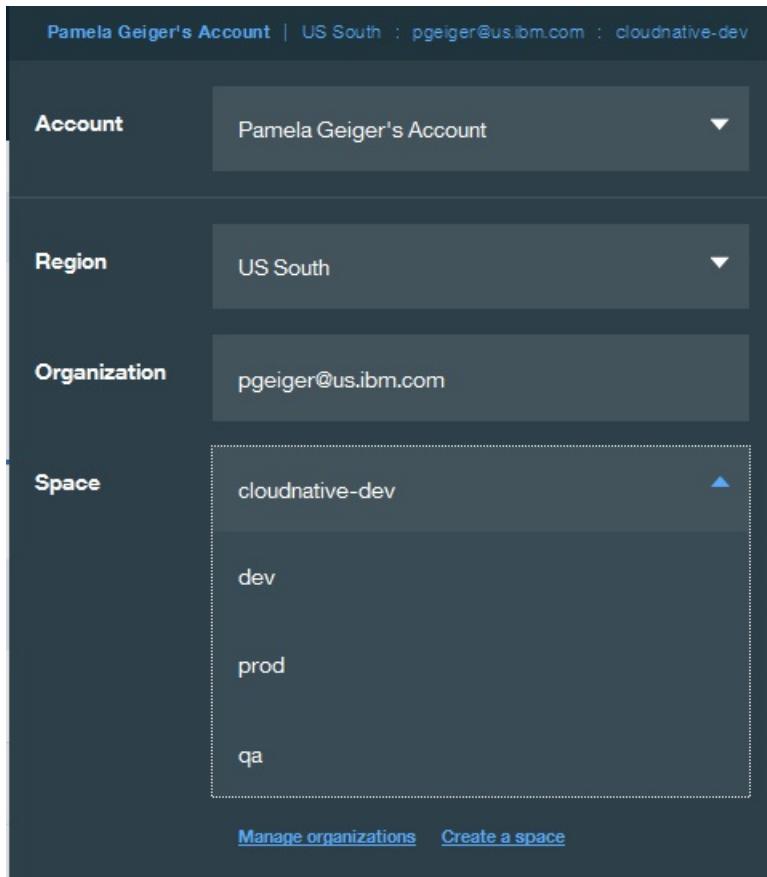3. Click `Create a Space` to create a new space in the current org. Enter **dev** for the space name and click **Create**.

4. Repeat the previous step to create spaces that are named **test** and **prod**. The spaces are listed in the account information.
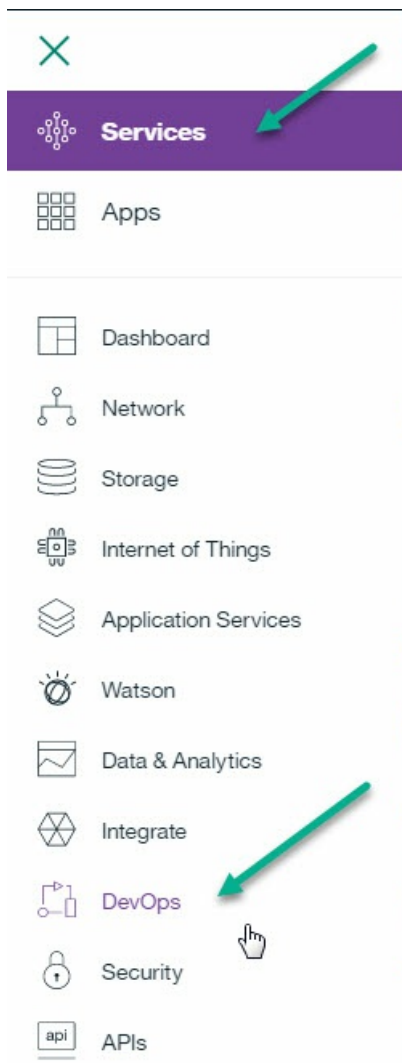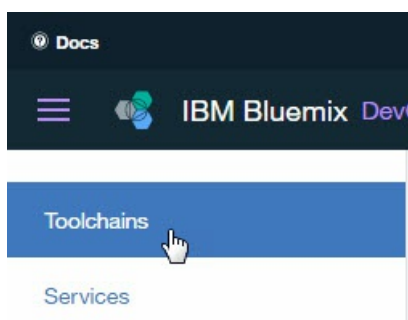

Task 2: Create Toolchain

1. We need to get to the DevOps Services. Click on the **Bluemix menu bar**
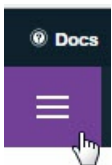
and click on **Services** then **DevOps**.

2. Click on **Toolchains**.

3. Click on **Create a Toolchain** on the right side of the screen.

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

Other Templates

Build your own toolchain
For advanced users, create
your toolchain from scratch.

5. Under 'Toolchain Settings', enter the name "devops-toolchain-lab" and click **Create**.

Toolchain Settings

Name:

devops-toolchain-lab

Your Toolchain is created and you are redirected to the Toolchain panel.

Task 3: Add and Configure GitHub Integration for Order

The code for the Order microservice already exists in a GitHub repository (https://github.com/open-toolchain/Microservices_OrdersAPI). 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.

GitHub
Store and manage code in a social way.
Third Party
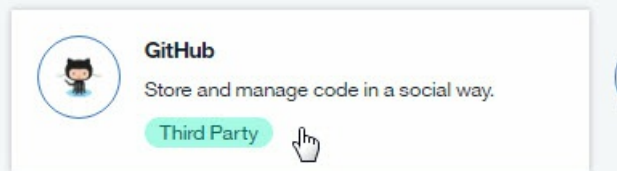
- Select 'Clone' as the Repository type.
- Enter "*githubuserid*/orders-api-toolchain-lab.git" for the New Repository Name.
  where *githubuserid* is your GitHub userid.
- Enter "https://github.com/open-toolchain/Microservices_OrdersAPI" for the Source
  repository URL.
- Ensure the 'Enable GitHub Issues' checkbox is selected.

Repository type:

Clone

New repository name:

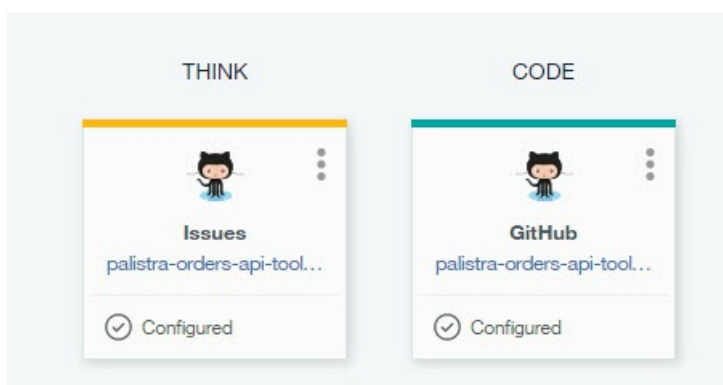palistra/orders-api-toolchain-lab.git

Source repository URL:

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

☑ Enable GitHub Issues

- Click **Create Integration**.

The devops-toolchain-lab tool integrations is displayed.

THINK   CODE

Issues   **GitHub**
palistra-orders-api-tool...   palistra-orders-api-tool...

⊘ Configured   ⊘ Configured

Task 4: Add Order 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 **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 tool integrations to this).
3. Under 'Pipeline name:', enter "orders-api-toolchain-lab".

Pipeline name:

orders-api-toolchain-lab

☐ Show apps in the View app menu

4. Click **Create Integration**.
5. The devops-toolchain-lab delivery pipeline is displayed.

```
![CreateDeliveryPipelineResult](screenshots/CreateDeliveryPipelineResult.jpg)
```

Task 5: Configure Order Delivery Pipeline

1. Now to configure the **orders-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 one job, taking the output from the Build stage and deploying on Bluemix into the *dev* space.
   - 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.
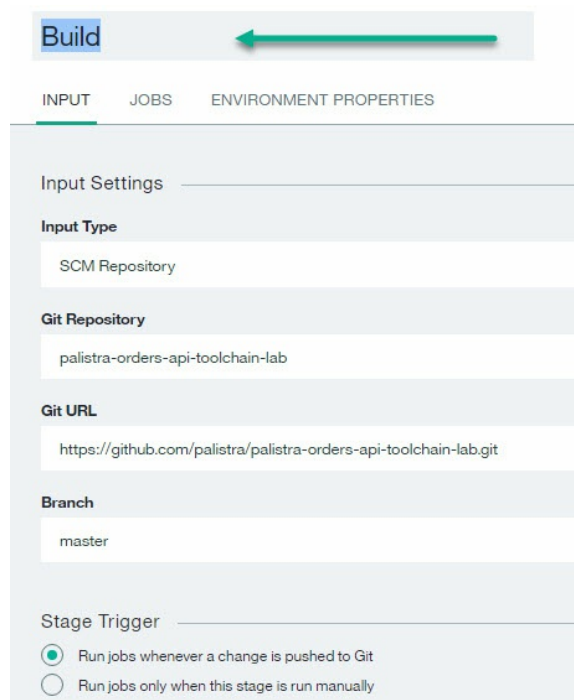
   Click on the **Delivery Pipeline** tile for the **orders-api-toolchain-lab** pipeline.
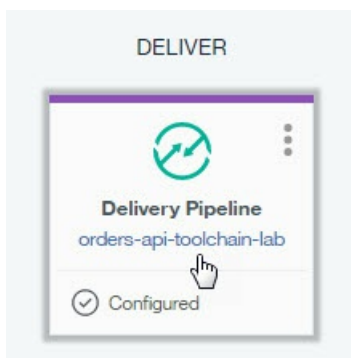

2. Add the **Build** stage and jobs.
   1. Click on **ADD STAGE**.
   2. On the **INPUT** tab, change "MyStage" to "Build" for Stage Name. Note that:
      - 'Input Type' is set to a SCM Repository, in this case, Git.
      - 'Git Repository' is set to the name of the Git Repository we just cloned. Make sure to select the "order" repo from the dropdown list.
      - '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.
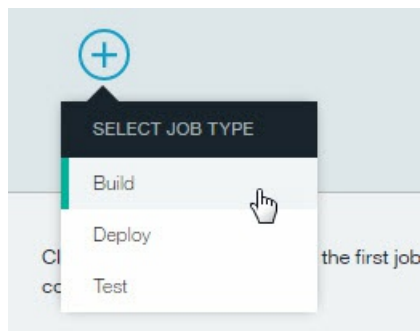

6. On the Job configuration panel, note that:
    - 'Builder Type' is set to "Simple" (other options are available on the pull-down).
    - 'Run Conditions' is set to "Stop running this stage if this job fails" to prevent any other jobs in this stage from running and to make the stage failed if this 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.
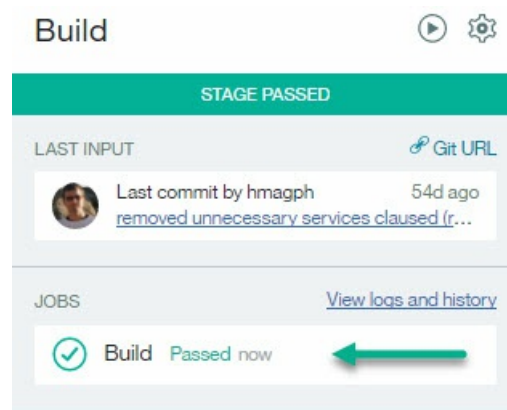
The **Build** stage has been successfully added and executed.

3. Add the **Dev** stage and jobs (remember, just one job, deploying to the *dev* space).
   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 "orders-api-toolchain-lab".
   - Type the following into the "Deploy Script" section.

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

   When an app is pushed to a space, its name is used as part of its route name. Routes need to be unique inside Bluemix. When there are multiple people doing the lab, they will all have the same routes, such as dev-orders-api-toolchain-lab.mybluemix.org. The first part of the bash script adds the user name to the app name to keep it unique.

   - '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.

IBM

Space

dev

Application Name

orders-api-toolchain-lab

Deploy Script

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

7. The bash script just entered into the Deploy Script references the
    CF_APP_NAME environment variable (*CF_APP* is provided by
    default). CF_APP_NAME needs to be added to the environment
    variables.
8. Click the **ENVIRONMENT PROPERTIES** tab.
9. Click **ADD PROPERTY** and select **Text Property**.
10. Enter "CF_APP_NAME" as the 'Name'. Do not enter anything for the
    'Value'.
11. Click **Save** to save the **Dev** stage.
12. 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 and deploy the order API.

The JOBS section shows the Deploy job Passed. Click on the
"Deploy" Jobs to to view the log for that Job.

13. Scroll to the bottom of the log to see the result.

14. Click on the navigation arrow to return to the orders-api-toolchain-lab Delivery Pipeline.

15. LAST EXECUTION RESULT displays the url to the successfully deployed application (*user_name*-dev-orders-api-toolchain-lab.mybluemix.net) as well as a link to the runtime log.
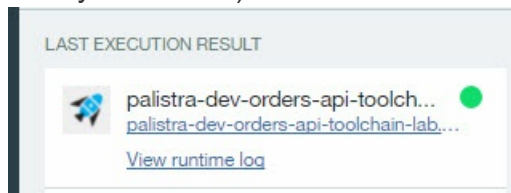
16. Click on "*user_name*-dev-orders-api-toolchain-lab.mybluemix.net" to access the running application.

17. CLose the application window.

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

4. Add the **Test** stage (remember, two jobs, one to deploy to the *qa* space and another to perform an automated test). We will clone the **Dev** stage and make some modifications.
   1. Ensure the orders-api-toolchain-lab **Delivery Pipeline** is displayed.
   2. On the **Dev** stage, click the **Stage Configuration** gear and select "Clone Stage".
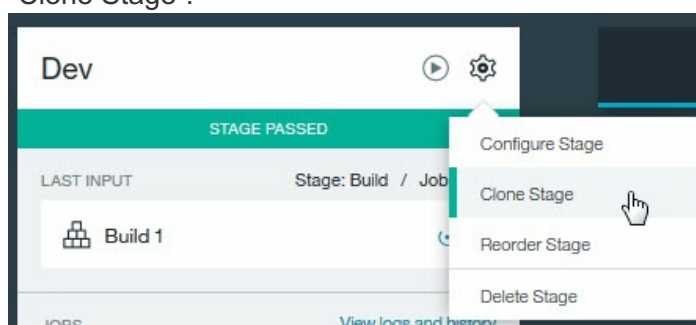
   3. Rename the cloned stage to **Test** (from **Dev [copy]**).
   4. On the **Jobs** tab, change the space to **qa** (from **dev**) (or Create a new space called **qa** if not on the dropdown) and change the deploy

IBM

script to change CF_APP_NAME to "$user_name-test-$CF_APP"
(from "$user_name-dev-$CF_APP").

Space

    qa

Application Name

    orders-api-toolchain-lab

Deploy Script

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

5. Add a new Job of type Test called **Test**. There are a number of different Testers available. For this exercise, we will select the default Simple Tester. Enter the following code to the **Test Command**.

```
#!/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.

6. Click the **ENVIRONMENT PROPERTIES** tab. Note the environment variable CF_APP_NAME is already present.

7. Click **Save** to save the **Test** stage.

8. 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.

9. 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.

JOBS   View logs and history

  ⊘ Deploy Passed 1h ago

  ⊘ Test   Passed 1h ago

10. The **Test** job was successful.

```
Downloading artifacts...DOWNLOAD SUCCESSFUL
Testing of App Name palistra-test-orders-api-toolchain-lab was successful

Finished: SUCCESS
```
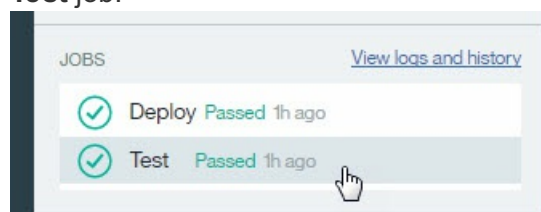
Click on "*user_name*-test-orders-api-toolchain-lab.mybluemix.net" to access the running application.

The **Test** stage has been successfully added and executed. Close the browser tab with the running application.

● 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. Click on the navigation arrow to return to the orders-api-toolchain-lab Delivery Pipeline.
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**' (from **Deploy**), change the space to **prod** (from **dev**) (or Create a new space called **prod** if not on the dropdown) and change the deploy script to the following:

```bash
#!/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 Lite 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
```

prod

Application Name

orders-api-toolchain-lab

Deploy Script

```
#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 the **ENVIRONMENT PROPERTIES** tab. Note that the environment variable CF_APP_NAME is already present.

6. Click **Save** to save the **Prod** stage.

7. Click on **Run Stage** to run the **Prod** stage and deploy the order API to the *prod* space.

8. The JOBS section shows the Deploy was successful. Inspect the Job log.

```
     state      since                    cpu    memory        disk         details
#0   running    2016-10-24 01:51:07 AM   0.0%   78.4M of 96M  107.6M of 1G
Sending deployment success of palistra-prod-orders-api-toolchain-lab to IBM DevOps Services...
IBM DevOps Services notified successfully.

Finished: SUCCESS
```

9. Click **Redeploy** to redeploy the stage.

10. The JOBS section shows the Redeploy was successful. Inspect the Job log. Note the application was renamed, replace and the old version deleted.

```
     state      since                    cpu    memory        disk         details
#0   running    2016-10-24 01:59:10 AM   0.1%   78.3M of 96M  107.6M of 1G
Sending deployment success of palistra-prod-orders-api-toolchain-lab to IBM DevOps Services...
IBM DevOps Services notified successfully.
Deleting app palistra-prod-orders-api-toolchain-lab-OLD-1477274247 in org palistra@us.ibm.com / space prod as palistra@us.ibm.com...
OK
Sending deployment success of palistra-prod-orders-api-toolchain-lab-OLD-1477274247 to IBM DevOps Services...
IBM DevOps Services notified successfully.

Finished: SUCCESS
```

11. Click on the blue arrow (upper left) to display the Delivery Pipeline.

12. Click on "*user_name*-prod-orders-api-toolchain-lab.mybluemix.net" to access the running application.

The **Prod** stage has been successfully added and executed. The Orders application has been deployed to production.

13. Close the application window.

14. Click on the blue arrow (upper left) to return to the devops-toolchain-lab Toolchain.