Creating Open Toolchains for IBM Bluemix

Objective

This series of labs shows how to set up a productive Continuous Delivery toolchain with a sample that consists of three microservices. After you finish this part of the series, you will be familiar with a toolchain that demonstrates practices from the IBM® Bluemix® Garage Method. ***Note:*** Toolchains are currently available in the US South region only and the instructions in this lab are written for the US South region.

To create this toolchain, we will use a sample to create an online store that consists of three microservices: a Catalog API, an Orders API, and a UI that calls both of the APIs. The toolchain is pre-configured for continuous delivery, source control, blue-green deployment, functional testing, issue tracking, online editing, and alert notification. We will explore the various integrations.

Online Store sample

The online store consists of three microservices:

1. Catalog API: A back-end RESTful API that tracks all of the items in the store.
2. Orders API: A back-end RESTful API that tracks all store orders.
3. UI: A simple UI that displays all of the items in the store catalog, and that can create orders. This PHP UI calls both of the REST APIs.

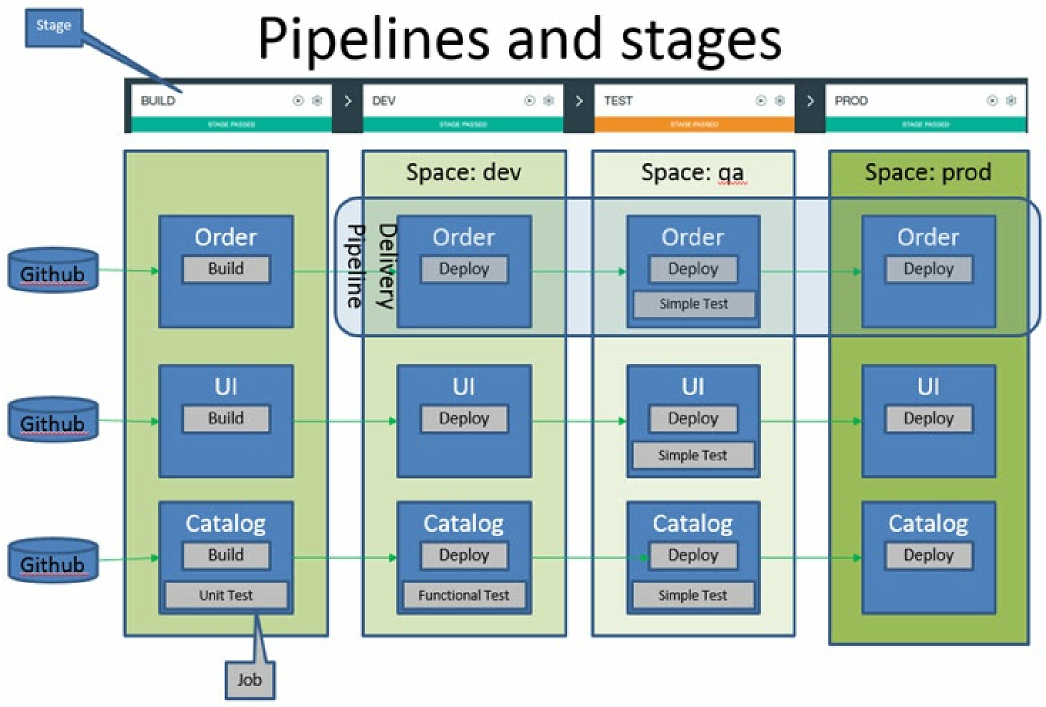
The Catalog and Orders API are backed by a Cloudant store. As part of deploying this application a no cost Cloudant service instance will be created.

Pipelines, stages and deployment environments - oh my!

In the real world, many enterprises have a process for developing, testing and deploying code to production. The lab scenario we will go through shows how Bluemix Continuous Delivery toolchains can be used to to automate that process.

- The code for the online store is already in three different GitHub repositories, one per microservice. As part of creating the Continuous Delivery toolchain, you will clone the repositories to your own GitHub account.
- Three delivery pipelines will also be created, again one per microservice. Each pipeline can be run in parallel.
- Each delivery pipeline will consists of a number of stages (Build, Dev, Test and Prod).
- Each stage will consist of one or more jobs that perform a task such a build the code, deploy the code, or testing the codes.
- As part of the respective delivery pipeline, each microservice is deployed to three environments: development, test, and production. You end up with nine deployed applications.
- While we could edit locally (using Eclipse for example) and push the code up to Bluemix, we will instead use the Eclipse Orion Web IDE, which you can use to edit your code and deploy it with the pipeline from a web browser.
- We want our application to scale as needed so we will be deploying one of the microservices into Bluemix containers

Conceptually, the process looks like:



Teaming

Software development is a team activity. The lab scenario also shows how Bluemix Continuous Delivery tool integrations can be used to alerts teams when activities occur (such as builds or deployments) as well as when events happen (such as a build failing or an application outage).

- Slack is configured to alert the team when activities occur
- [PagerDuty | IBM Alert Notification] is configured to alert the team when events happen
- Bluemix Availability Monitoring is configured to monitor the application in production and alert the team when outages occur

It sounds like a lot ... and it is! Thankfully, it is all handled by a Bluemix Continuous Delivery toolchain. And even better, we will use an existing template to give a great starting point.

Prerequisites

Prior to running these labs, you must have a Bluemix account, a GitHub account and access to a lab laptop. Follow the steps in Lab 0 to create one or both of those accounts.

Labs

Lab 0 Create Bluemix and GitHub accounts

Objective

This lab sets up the prerequisite accounts for the remaining labs.

Prerequisites

Prior to running these labs, you must have a Bluemix account, a GitHub account and access to a lab laptop. Follow the steps in this lab to create one or both of those accounts. Note you will need access to your eMail account to confirm the account setup activity.

Tasks:

- Task 1: Create Bluemix trial account
- Task 2: Create GitHub account

Task 1: Create Bluemix trial account

1. If you already have an active Bluemix account, you can skip this task.
2. Open a web browser and enter the following URL: https://console.ng.bluemix.net/
3. Click on the **Sign Up** button.
4. Follow the directions to fill out the form. Note you will need access to an eMail account to confirm the account setup activity. Make note of the password you specify.
5. Click **Create Account**. This will cause Bluemix to send an email to the eMail account you specified.
6. Login into the eMail account you specified. Open the eMail with the subject: *Action Required: Confirm your Bluemix account*.
7. Click on the **Confirm Account** button.
8. You now have an active Bluemix trial account.

Task 2: Create GitHub account

1. If you already have a GitHub account, skip this task.
2. In a web browser, enter the following URL: https://github.com/
3. Follow the directions to fill out the form. Note you will need access to an eMail account to confirm the account setup activity. Make note of the password you specify.
4. Click on the **Sign up for GitHub** button. This will cause GitHub to send an email to the eMail account you specified.
5. Login into the eMail account you specified. Open the eMail from GitHub with the subject: *Please verify your email address*.
6. Click on the **Verify email address** link.
7. You now have an active GitHub account.

Lab 1 Create Toolchain for Sample Application

Objective

This lab takes you through the process of creating the Continuous Delivery toolchain for the sample online application.

Tasks:

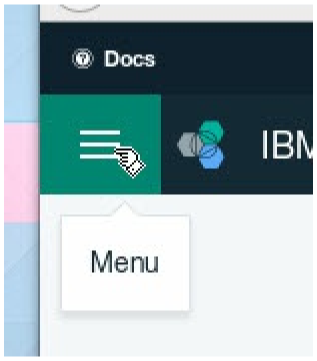
- Task 1: Log in to Bluemix
- Task 2: Display Microservices Toolchain panel
- Task 3: Understand Microservices Toolchain panel
- Task 4: Create Microservices Toolchain

Task 1: Log in to Bluemix

1. In a web browser, go to the Bluemix login page: https://console.ng.bluemix.net/
2. Login into Bluemix by entering your Bluemix account and password.

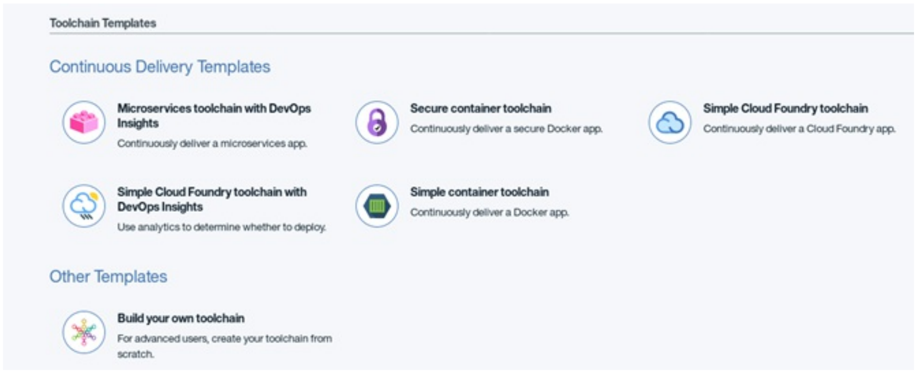
Task 2: Display Microservices Toolchain panel

1. We need to get to the DevOps Services. Click on the hamburger menu.



2. Click on **Services** then **DevOps**.
3. Click on **Toolchains**.

4. Click on **Create a Toolchain**.


Toolchain Templates
Continuous Delivery Templates
Microservices toolchain with DevOps Insights
Continuously deliver a microservices app.
Secure container toolchain
Continuously deliver a secure Docker app.
Simple Cloud Foundry toolchain
Continuously deliver a Cloud Foundry app.
Simple Cloud Foundry toolchain with DevOps Insights
Use analytics to determine whether to deploy.
Simple container toolchain
Continuously deliver a Docker app.
Other Templates
Build your own toolchain
For advanced users, create your toolchain from scratch.

Task 3: Understand Microservices Toolchain panel

1. There are a number of ways to create a Toolchain. For this exercise, we will start with an existing template, the Microservices toolchain. Click **Microservices toolchain with DevOps Insights**.
2. For an explanation of this toolchain, read the paragraphs on the left. This toolchain integrates various tools, some of which we will configure:

- GitHub
- Delivery pipeline

Some of which we will not configure:

- PagerDuty
- Sauce Labs
- Slack

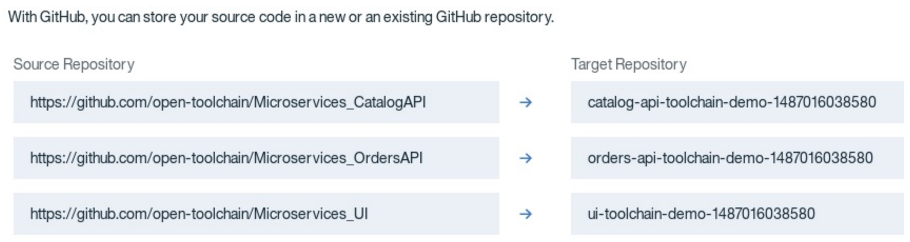
And some of which do not require configuration.

- DevOps Insights
- Eclipse Orion Web IDE

3. The **Organization** field is the name of the Bluemix Organization in which this toolchain will be created.

4. The **Toolchain Name** names is a generated name for this Toolchain. Rename it to something memorable (as you would in real life) or leave it at the default generated name (which is what the screenshots show).

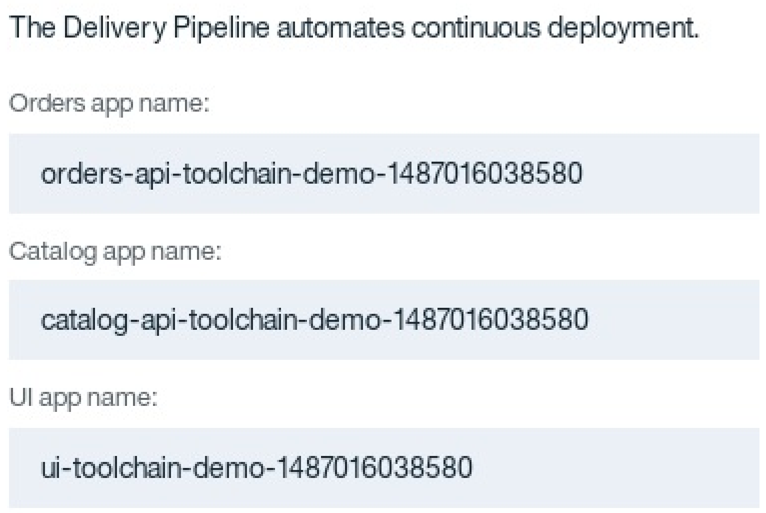
5. Click on **GitHub**. GitHub is where the source of the application is stored, one GitHub repo per application (so three GitHub repos). We will set up our Toolchain to create a clone of each repo for use in this lab.
6. If you haven't authorized Bluemix to access GitHub, you need to:
    1. Click **Authorize** to go to the GitHub website.
    2. Enter your GitHub username and password.
    3. Click **Sign in**.
    4. Click **Authorize application**.
7. Once authorized, you see the three Source Repositories (one for each of Catalog, Orders and UI) where the code is stored and three corresponding Target Repositories, where the Source Repositories will be cloned. The Target Repository name is generated and just like Toolchain Name, you can leave the default generated name or make it something more memorable.



With GitHub, you can store your source code in a new or an existing GitHub repository.

Source Repository		Target Repository
https://github.com/open-toolchain/Microservices_CatalogAPI	→	catalog-api-toolchain-demo-1487016038580
https://github.com/open-toolchain/Microservices_OrdersAPI	→	orders-api-toolchain-demo-1487016038580
https://github.com/open-toolchain/Microservices_UI	→	ui-toolchain-demo-1487016038580

8. Click on **Delivery Pipeline**. We will be creating three delivery pipelines, one for each microservices. This is where the application name for each microservice will be specified, as well as the Bluemix Region, Organization and Space where the microservices will be deployed.
9. The application names for the three microservices must be unique in the Bluemix environment so it is best to leave them as generated.



The Delivery Pipeline automates continuous deployment.

Orders app name:

orders-api-toolchain-demo-1487016038580

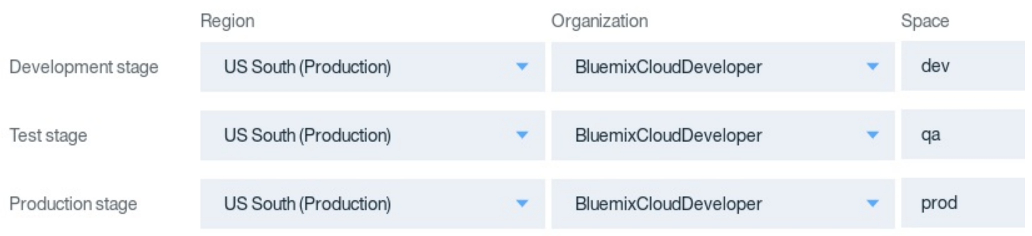
Catalog app name:

catalog-api-toolchain-demo-1487016038580

UI app name:

ui-toolchain-demo-1487016038580

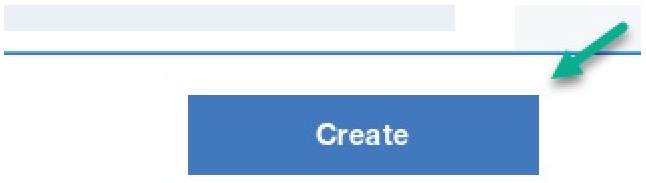
10. Pipelines can only be created in the US South region so to keep things simple we will deploy to only the US South region and in the Organization we are logged into.
11. We have three spaces for our environment corresponding to the lifecycle we are using.
    1. Development (**dev**) where code development takes place
    2. Testing or Quality Assurance (**qa**) where testing takes place
    3. Production (**prod**) where the application is available to end users (in our lab scenario, we do not restrict access to the dev or qa applications but in real life you would).



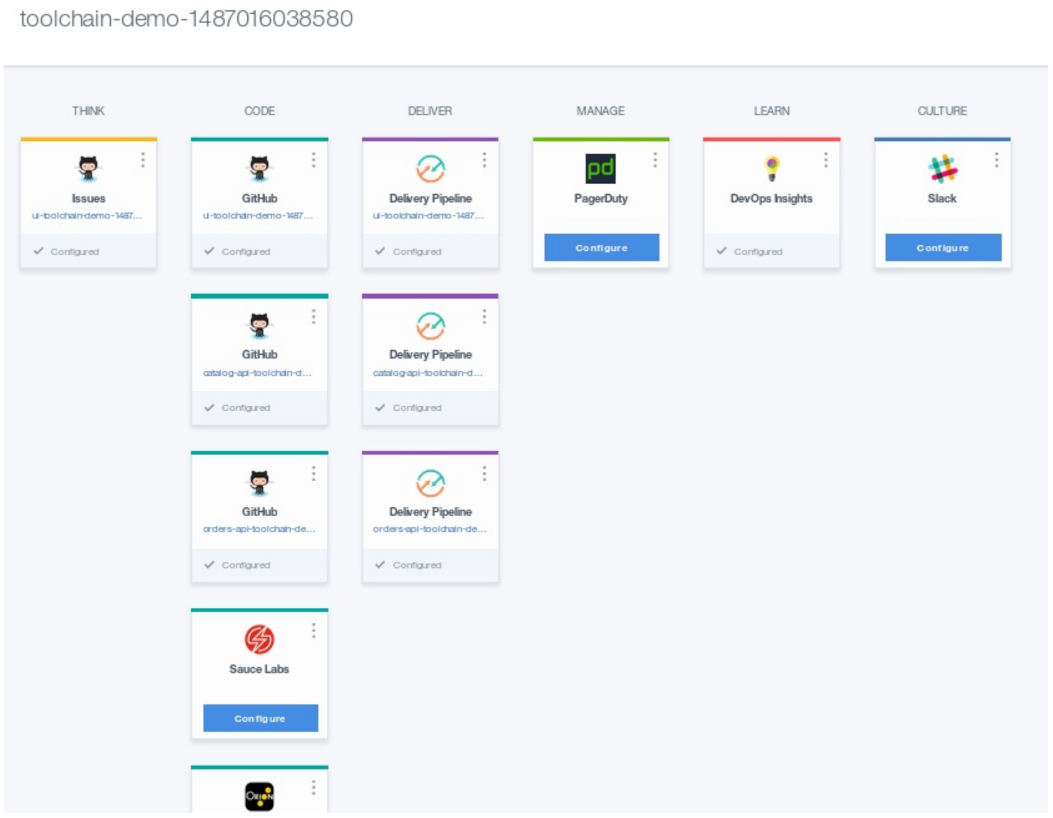
	Region	Organization	Space
Development stage	US South (Production)	BluemixCloudDeveloper	dev
Test stage	US South (Production)	BluemixCloudDeveloper	qa
Production stage	US South (Production)	BluemixCloudDeveloper	prod

Task 4: Create Microservices Toolchain

1. Click **Create** to create the toolchain.



2. The Microservices toolchain is created. If it take more then a few moments for the various tiles to display either Configured or Configure, refresh the browser.



Lab 2 Build and deploy to dev space

Objective

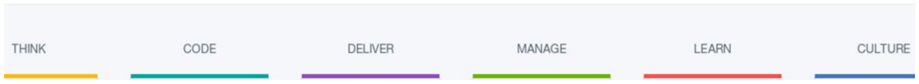
This lab will build the all three microservices and deploy them to the Development space.

Tasks:

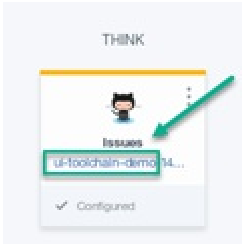
- Task 1: Explore the Microservices toolchain
- Task 2: Examine the delivery pipelines

Task 1: Explore the Microservices toolchain

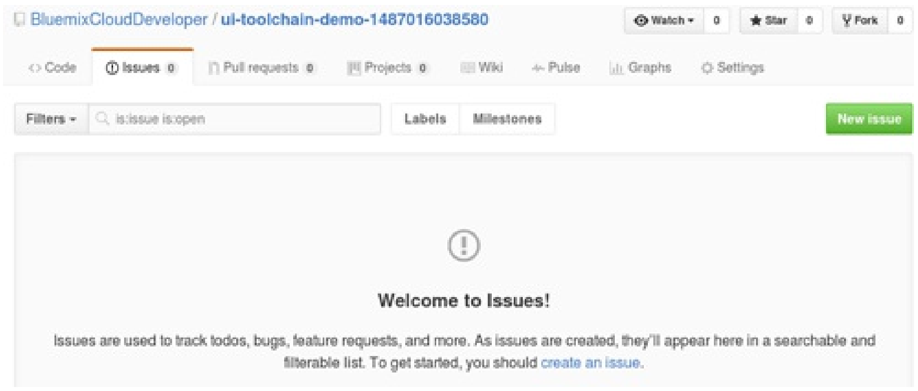
1. IBM Bluemix created the Continuous Delivery Toolchain based on the Microservices template. At the top you see the pieces of the Bluemix Garage Method and where each tool integration fits.



2. **Think** is where the GitHub Issues database is listed. Click on the **ui-toolchain-demo** link (or right-mouse button click and select **Open Link in New Tab**, then select the new tab to save time later on).



3. This displays the **GitHub Issues** page. Issues are used to track todos, bugs, feature requests, and more. Each GitHub repository (*repo* for short) can include issues. The Microservices template we used only included issues for the UI repo.



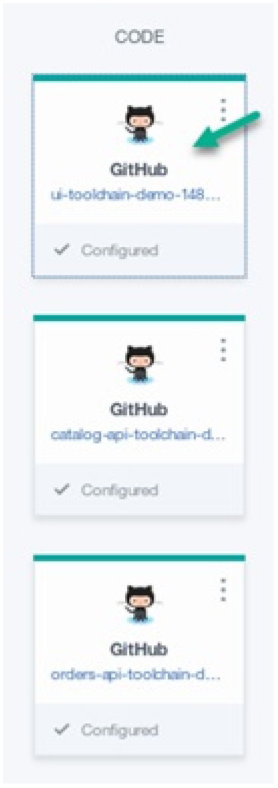
Welcome to Issues!

Issues are used to track todos, bugs, feature requests, and more. As issues are created, they'll appear here in a searchable and filterable list. To get started, you should create an issue.

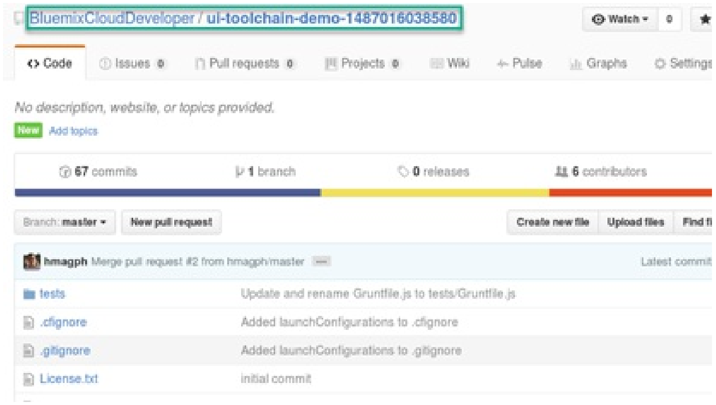
Return to the Microservices toolchain by either clicking on the **Go back one page** arrow on the browser or, if you clicked the right-mouse button to open a new tab, close the GitHub Issues tab. (Note that the remainder of these lab instructions will not go into this level of detail on opening and closing pages and tabs - pick the method that is best for you.)

4. **Code** is where GitHub code repos, Sauce Labs and Eclipse Orion Web IDE are integrated.

    1. Clicking on one of the three repos

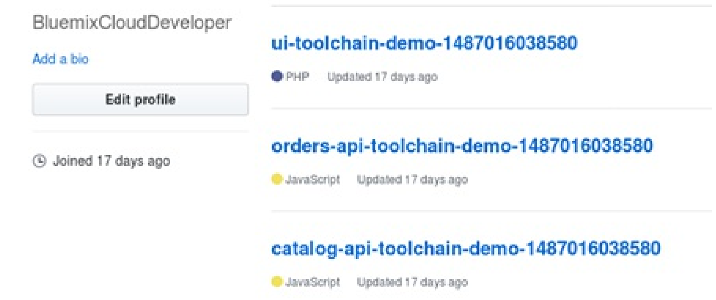


will display the respective (cloned) repo



tests	Update and rename Gruntfile.js to tests/Gruntfile.js
.cfignore	Added launchConfigurations to .cfignore
.gitignore	Added launchConfigurations to .gitignore
License.txt	initial commit

The creation of the Toolchain did clone three repos in GitHub.

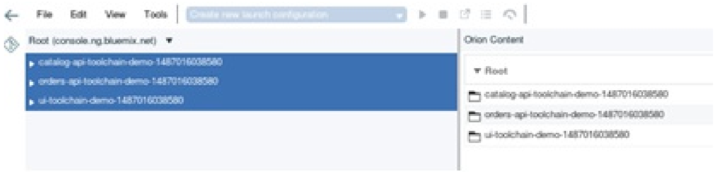


ui-toolchain-demo-1487016038580
PHP Updated 17 days ago

orders-api-toolchain-demo-1487016038580
JavaScript Updated 17 days ago

catalog-api-toolchain-demo-1487016038580
JavaScript Updated 17 days ago

2. Clicking on the **Eclipse Orion Web IDE** will display the Web editor.


File
Edit
View
Tools
Create new launch configuration
Root (console.ng.bluemix.net)
Orion Content
Root

catalog-api-toolchain-demo-1487016038580
orders-api-toolchain-demo-1487016038580
ui-toolchain-demo-1487016038580

3. We do not have a Sauce Labs account, so we really don't need the Sauce Labs integration. We will leave it alone for now.

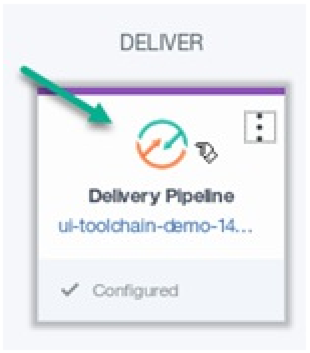
5. **Deliver** is where the code gets built, tested and deployed through the integrations of build pipelines, one per microservice. We explore build pipelines later.

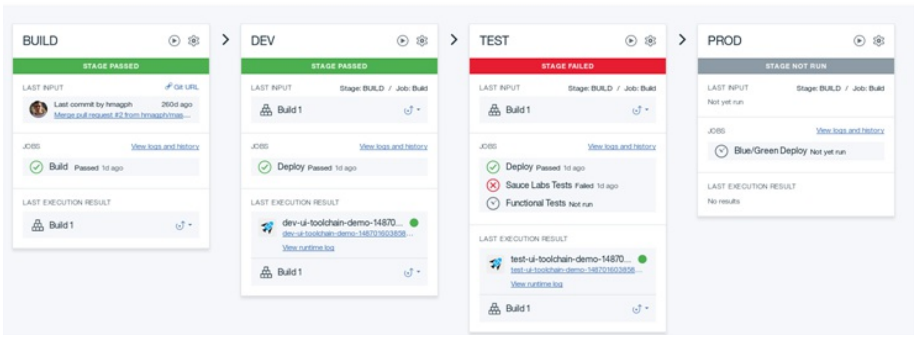
6. **Manage** is where management tools, such as Pager Duty, get integrated.

7. **Learn** is where tools helping to drive application insight, such as DevOps Insights, get integrated.

8. **Culture** is where tools helping teams collaborate more effectively, such as Slack, get integrated.

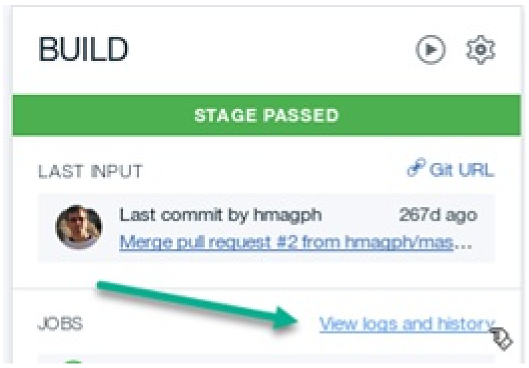
Task 2: Examine the delivery pipelines

1. Click on the circle in the center of the *ui-toolchain-demo* Delivery Pipeline tile


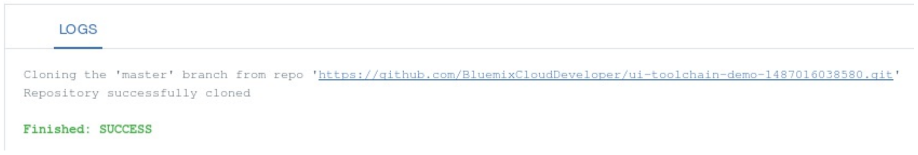
to display the UI delivery pipeline.
ui-toolchain-demo-1487016038580 | Delivery Pipeline


2. While we were busy exploring the toolchain, the various pipelines (remember, we have 3) started the build process. The UI delivery pipeline displays the status of each stage in the UI pipeline. The **Build** and **Dev** stage passed, while the **Test** stage failed. The **Prod** stage was not even attempted. If you look at the other 2 pipelines (catalog and orders) you would see similar results.

3. In the Build stage, click **View logs and history**

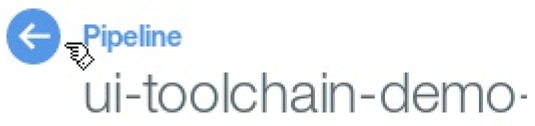


to display the commands and results of the Build stage, in this case simply cloning the repo.



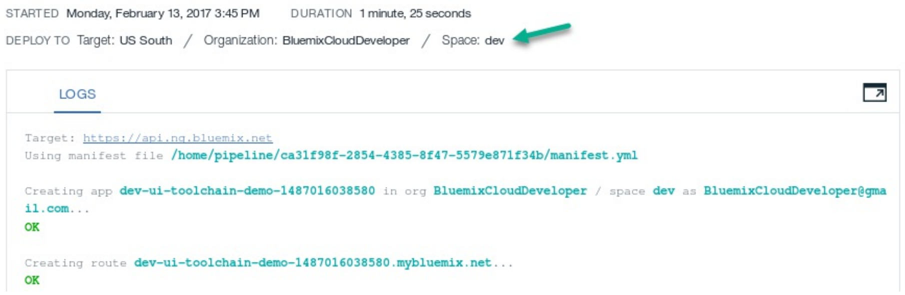
```
Cloning the 'master' branch from repo 'https://github.com/BluemixCloudDeveloper/ui-toolchain-demo-1487016038580.git'
Repository successfully cloned

Finished: SUCCESS
```

Click the arrow to the left of Pipeline



to return to the delivery pipeline.

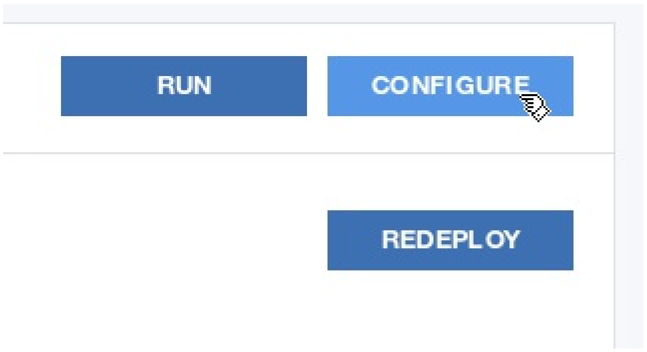
4. In the **Dev** stage, click **View logs and history** to display the commands and results of the Dev stage. This stage deployed the UI microservice to the dev space.



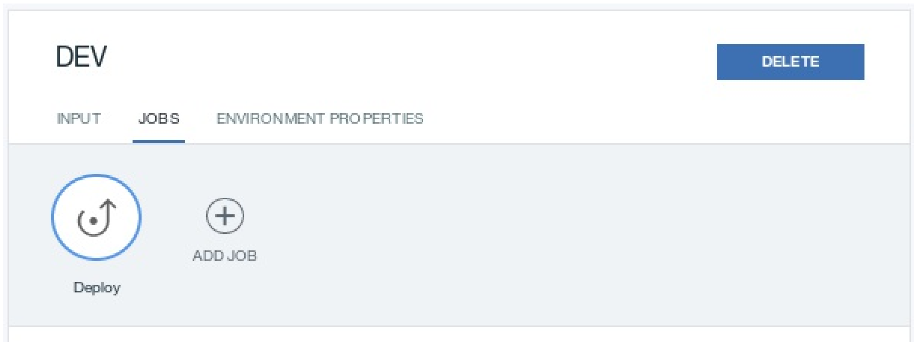
STARTED Monday, February 13, 2017 3:45 PM DURATION 1 minute, 25 seconds
DEPLOY TO Target: US South / Organization: BluemixCloudDeveloper / Space: dev

```
Target: https://api.ng.bluemix.net
Using manifest file /home/pipeline/ca31f98f-2854-4385-8f47-5579e871f34b/manifest.yml

Creating app dev-ui-toolchain-demo-1487016038580 in org BluemixCloudDeveloper / space dev as BluemixCloudDeveloper@gma
il.com...
OK

Creating route dev-ui-toolchain-demo-1487016038580.mybluemix.net...
OK
```

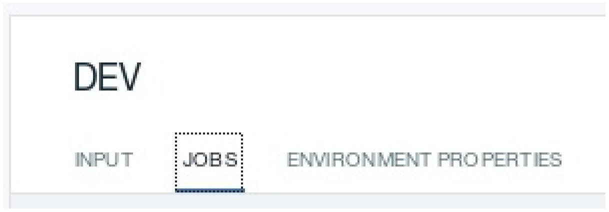
5. If you scroll through the log, you can see all the details of that job.
6. Click **Configure** in the upper right hand corner of the display



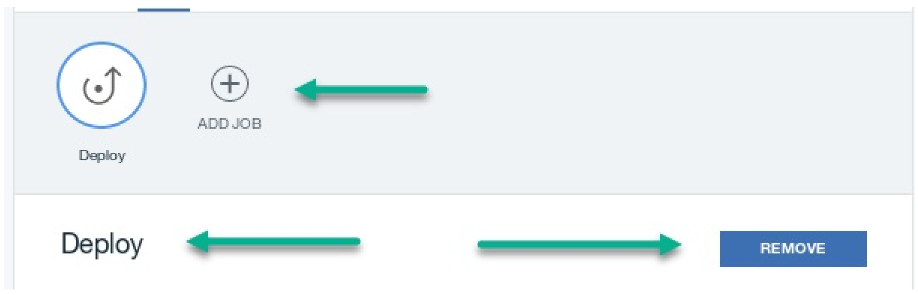
to display the Stage Configuration screen.



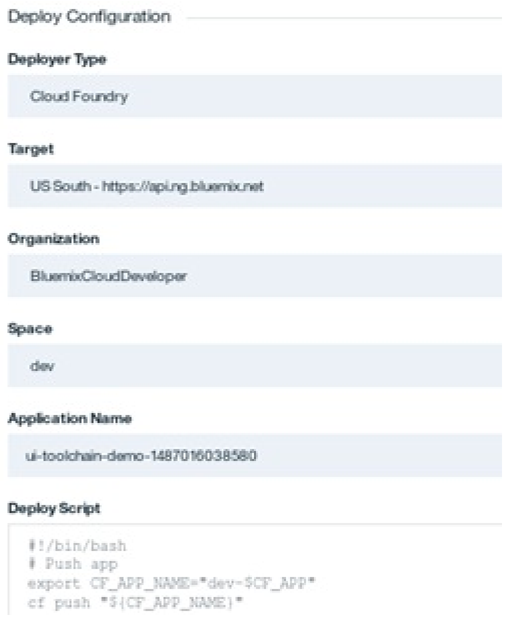
7. The Stage Configuration displays details about the stage. The *INPUT* tab displays the input to the stage, the *JOBS* tab displays the discrete jobs of the stage, and the *ENVIRONMENT PROPERTIES* tab displays variables used by the jobs in the stage.



8. The UI Dev stage has just one job, the *Deploy* job. The job name can be changed by simply typing over the name. The job can be removed by clicking the **Remove** button and a new job can be added by clicking the **Add Job** button.



9. Details of each job are displayed when the job is selected.



Deploy Configuration

Deployer Type

Cloud Foundry

Target

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

Organization

BluemixCloudDeveloper

Space

dev

Application Name

ui-toolchain-demo-1487016038580

Deploy Script

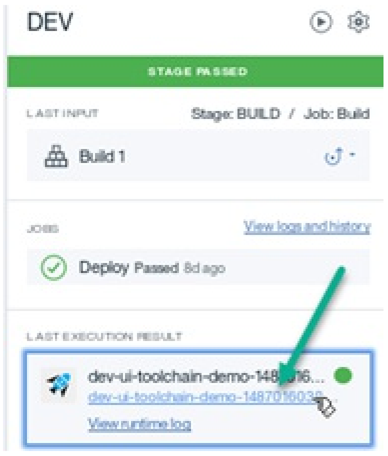
```
#!/bin/bash
# Push app
export CF_APP_NAME="dev-$CF_APP"
cf push "${CF_APP_NAME}"
```

The lines in the Deploy Script:

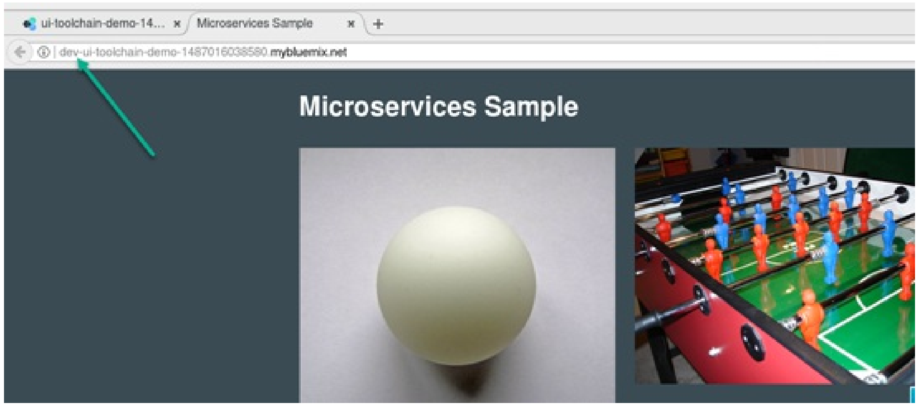
```
export CF_APP_NAME="dev-$CF_APP"
cf push "${CF_APP_NAME}"
```

set the name of the application to deploy as the application name prefaced with *dev* and the issues the Cloud Foundry command to deploy it.

10. Return to the delivery pipeline. Click on the application link to display the application.

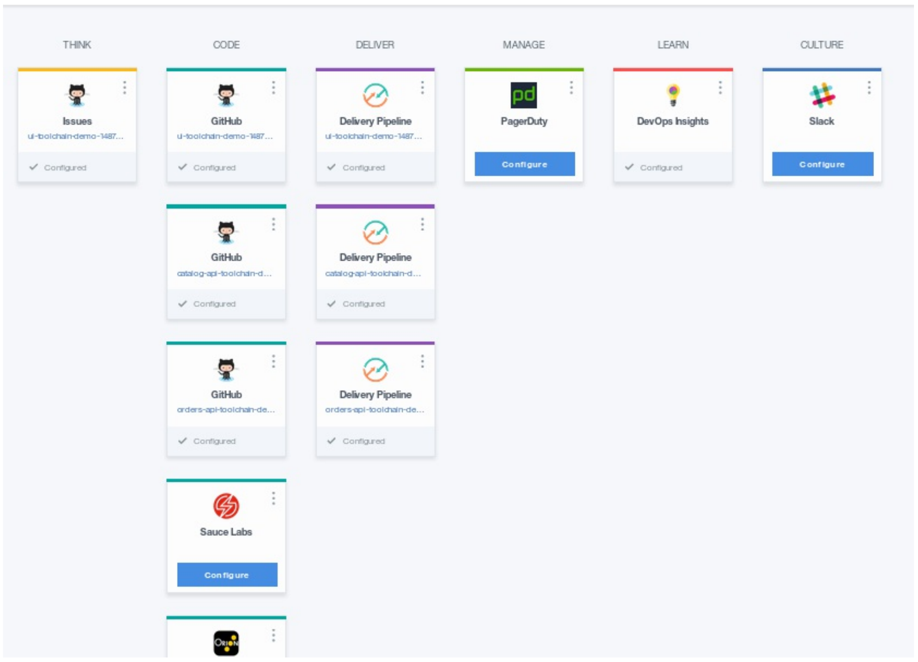


11. The application is displayed. We deployed all the microservices to the *dev* space and prefaced each deployed app with *dev*.



12. Close the application tab to return to the delivery pipeline, then return to the Pipeline and finally return to the Toolchain.

toolchain-demo-1487016038580



Lab 3 Customize Toolchain to add Slack Integration

Objective

This lab will integrate Slack into the Continuous Delivery Toolchain. Slack is a cloud-based team collaboration tool. We will integrate Slack into our Toolchain so team members get notified when development events, such as builds, occur.

Tasks:

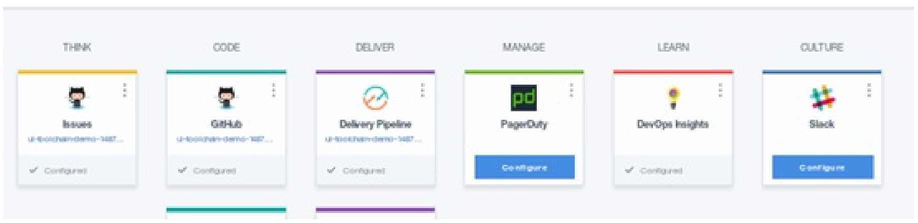
- Task 1: Integrate Slack
- Task 2: Work with Slack
- Task 3: Modify Toolchain for Sauce Labs test job
- Task 4: Modify Toolchain for Functional Tests job
- Task 5: Examine PROD Stage

Task 1: Integrate Slack

1. If we needed to add Slack to a Toolchain, we would click **Add a Tool** on the Toolchain display and select **Slack** from the available integrations. We don't have to do this as the Microservices template already included Slack in the Toolchain but we did not configure it. We also have a Slack user ID already created (*bluemix_interconnect*).

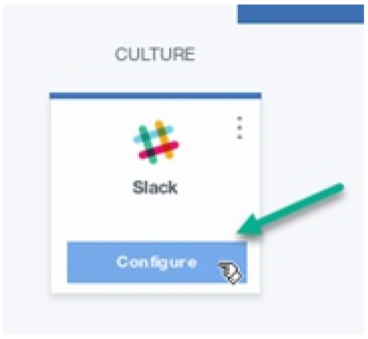
2. You should be displaying the Toolchain.

   toolchain-demo-1487016038580



   If not, click on the hamburger menu, then click on **Services**. Click on **DevOps** and then click on **Toolchains**. And finally click on the toolchain you created.

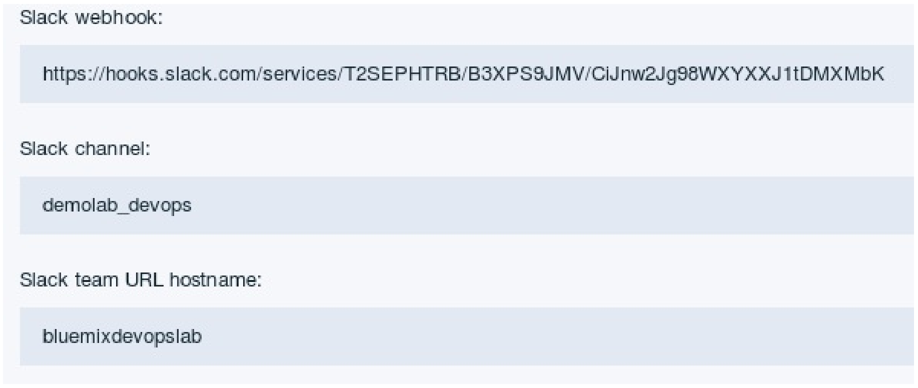
3. Click on **Configure** to configure the connection between Bluemix and Slack.



4. Enter the following for Slack webhook (all one string with no blanks or spaces):

```
https://hooks.slack.com/services/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
```

5. Enter the following for Slack channel: **demolab_devops**
6. Enter the following for Slack team URL Host name: **bluemixdevopslab**



Slack webhook:

https://hooks.slack.com/services/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

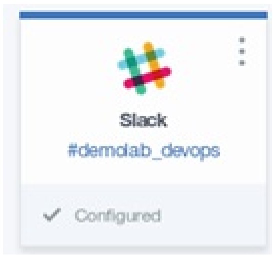
Slack channel:

demolab_devops

Slack team URL hostname:

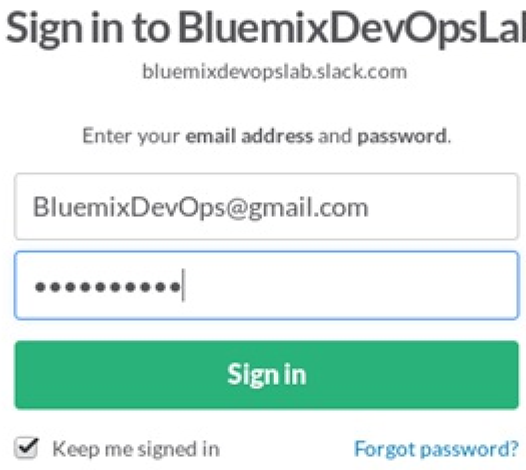
bluemixdevopslab

5. Click **Save Integration** to save the information.
6. Slack is now Configured.



Task 2: Work with Slack

1. In the browser, open a new tab and go to the following URL to go to the (already created) Slack team. https://bluemixdevopslab.slack.com
2. Enter the following information:
    1. Email address: **BluemixDevOps@gmail.com**
    2. Password: **bluemix4me**



Sign in to BluemixDevOpsLab

bluemixdevopslab.slack.com

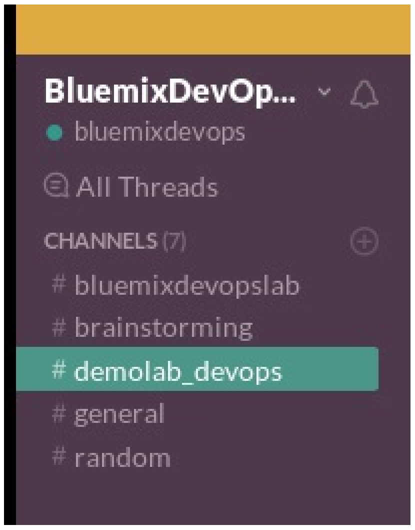
Enter your **email address** and **password**.

BluemixDevOps@gmail.com

•••••••••••

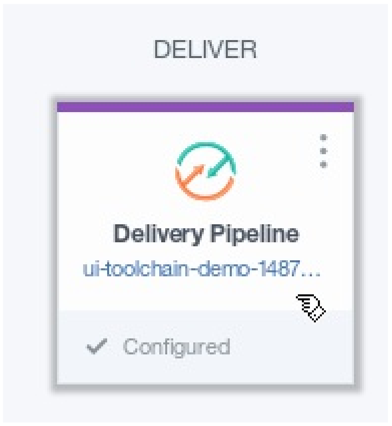
and click **Sign In**.

3. Click on the **demolab_devops** channel to show the messages for that channel.

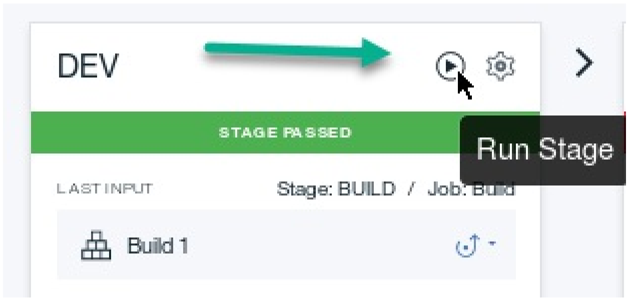


This channel will show all the messages the Toolchain sends to it.

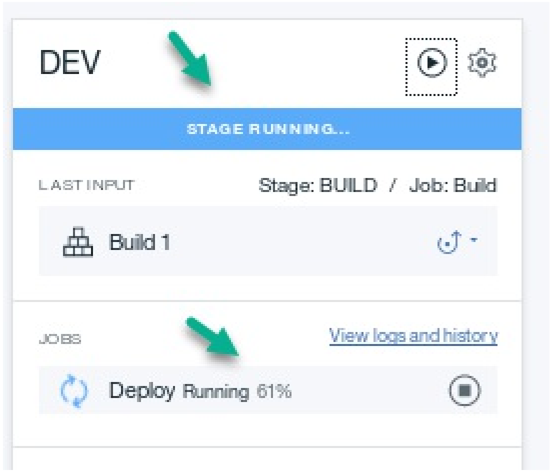
4. Leaving the Slack browser tab open, switch over to the Toolchain browser tab.
5. Click on the UI Delivery Pipeline tile to display the UI Delivery Pipeline.



6. Click on the arrow in the DEV stage to run the jobs in the DEV stage.

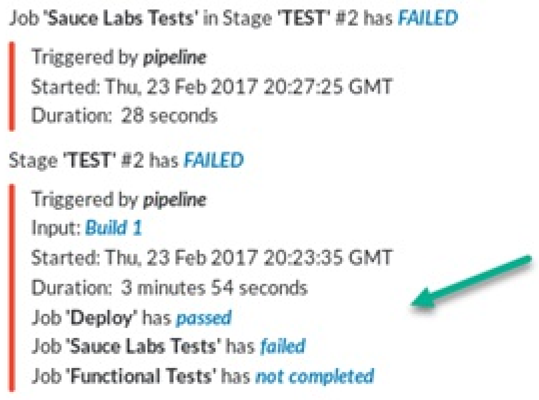


7. The DEV stage runs the jobs and you get a visual indication of the progress of the stage.



8. Switch to the Slack browser tab. Here you also get a visual indication of the progress of the DEV stage process. This is useful for notifying team members when events occur without them being logged into Bluemix

the DEV stage passed but the TEST stage failed. The *Deploy* job was successful, but the *Sauce Labs Test* job failed and the *Functional Tests* job was not attempted.



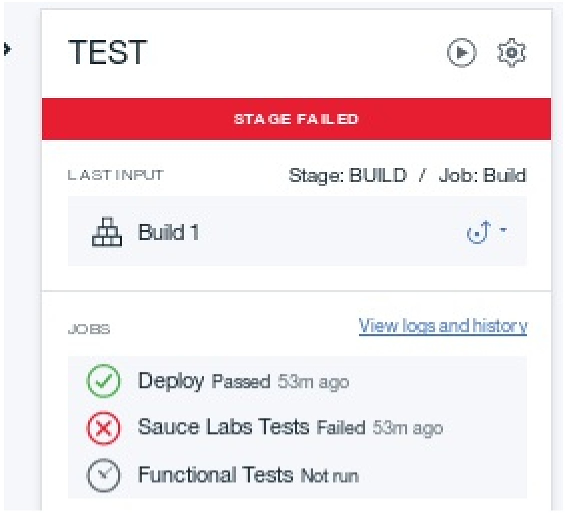
Job 'Sauce Labs Tests' in Stage 'TEST' #2 has FAILED

Triggered by pipeline
Started: Thu, 23 Feb 2017 20:27:25 GMT
Duration: 28 seconds

Stage 'TEST' #2 has FAILED

Triggered by pipeline
Input: Build 1
Started: Thu, 23 Feb 2017 20:23:35 GMT
Duration: 3 minutes 54 seconds
Job 'Deploy' has passed
Job 'Sauce Labs Tests' has failed
Job 'Functional Tests' has not completed

9. Switch back to the Bluemix browser tab. The Toolchain indicates the same results. Something is wrong with the Toolchain.



TEST

STAGE FAILED

LAST INPUT Stage: BUILD / Job: Build

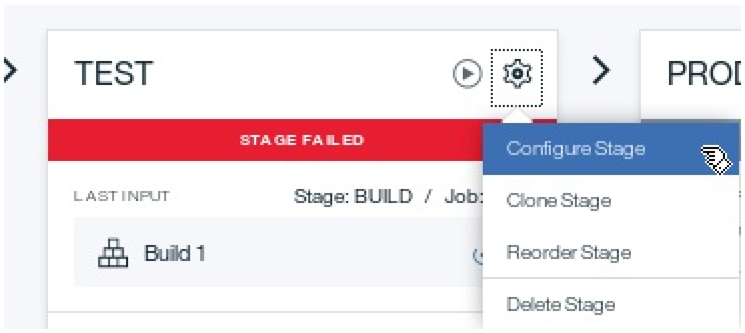
Build 1

JOBS View logs and history

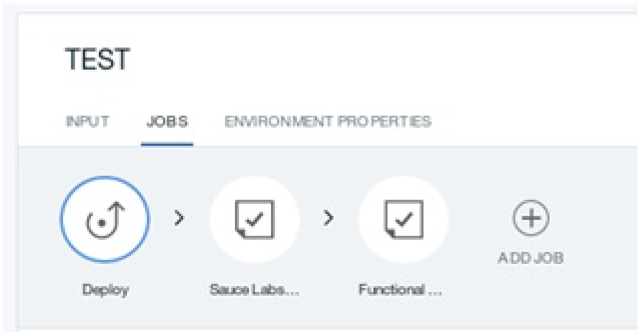
Deploy Passed 53m ago
Sauce Labs Tests Failed 53m ago
Functional Tests Not run

Task 3: Modify Toolchain for Sauce Labs test job

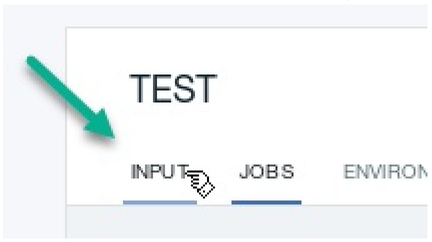
1. Remember we ran the DEV stage. Why did the TEST stage start? In the TEST stage tile, click on the gear and then click on **Configure Stage**.



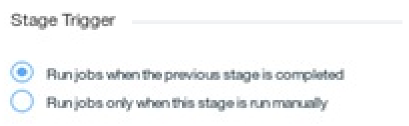
2. The TEST stage has three jobs, *Deploy*, *Sauce Labs Test*, and *Functional Tests*.



3. The *Deploy* job is highlighted (the blue circle around the icon), so details are the *Deploy* job are displayed.
4. Click **INPUT** to display the input settings for the TEST stage.



5. The Stage Trigger for the TEST stage indicates that this stage will run when the prior stage is complete. So in this case, when the DEV stage is complete, the Toolchain started the TEST stage.

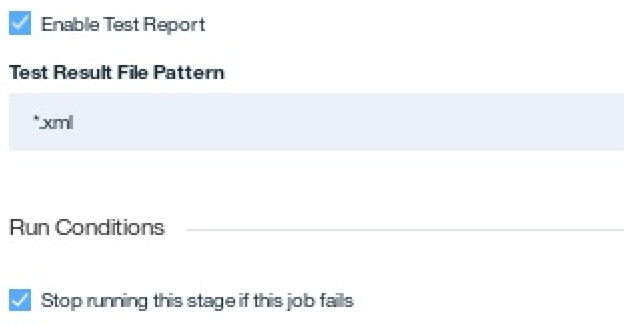


Stage Trigger

[✓ Run jobs when the previous stage is completed]
Run jobs only when this stage is run manually

6. Click on **JOBS** to display the jobs for the TEST stage.
7. Click on the **Sauce Labs Test** icon to display the details for the *Sauce Labs Test* job.

8. At the bottom of the details, under *Run Conditions*, the option to stop the stage if this job (the *Sauce Labs Test* job) fails. That explains why this stage stopped and why the *Functional Tests* job did not run.



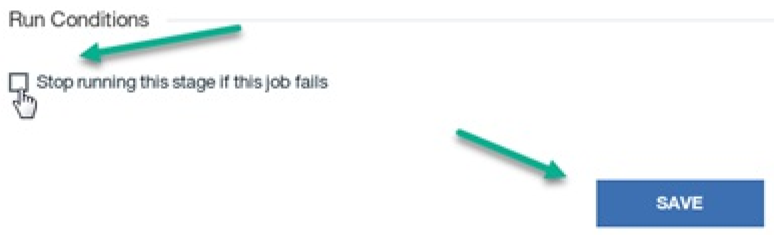
☑ Enable Test Report

Test Result File Pattern: *.xml

Run Conditions

☑ Stop running this stage if this job fails

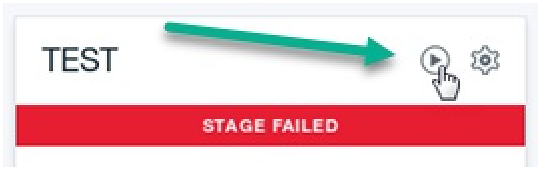
9. Why did the *Sauce Labs Test* job fail? As you may recall, we never configured that tool integration after we created the Toolchain. Sauce Labs requires a valid userid and password and we do not have those. If we did, we would configure the Sauce Labs integration with those details. So we have two choices, either remove the *Sauce Labs Test* job from the TEST stage or, assuming we may one day get a Sauce Labs account, allow the *Sauce Labs Test* job to fail but continue the stage. We will choose this option.
10. Deselect the option to stop the stage if this job fails and click **SAVE**.



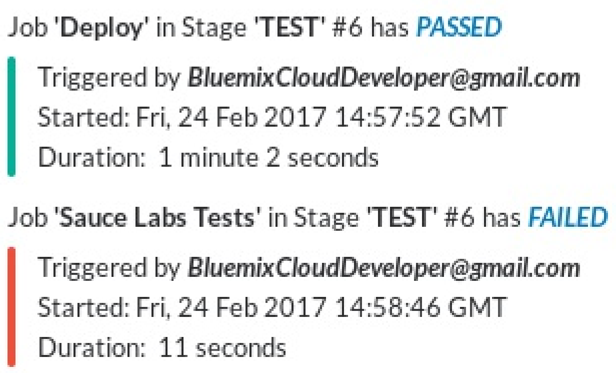
Run Conditions

☐ Stop running this stage if this job fails

11. Click on the arrow in the TEST stage to run the jobs in the TEST stage.



12. Both Bluemix and Slack show the TEST stage executing, even though the *Sauce Labs Test* job still fails.



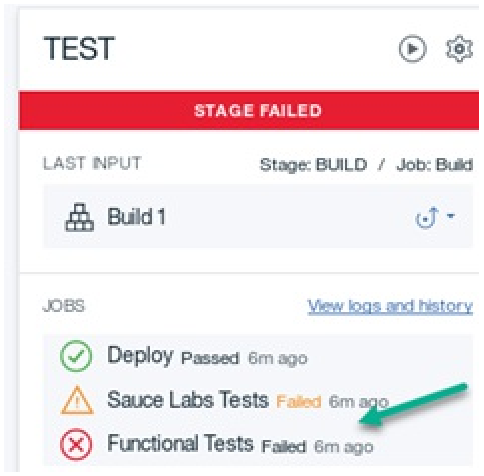
Job '**Deploy**' in Stage '**TEST**' #6 has *PASSED*

Triggered by *BluemixCloudDeveloper@gmail.com*
Started: Fri, 24 Feb 2017 14:57:52 GMT
Duration: 1 minute 2 seconds

Job '**Sauce Labs Tests**' in Stage '**TEST**' #6 has *FAILED*

Triggered by *BluemixCloudDeveloper@gmail.com*
Started: Fri, 24 Feb 2017 14:58:46 GMT
Duration: 11 seconds

13. After a few moments (or minutes), the *Functional Tests* job also fails. This is the first time we executed it. There is still a problem.


TEST
STAGE FAILED
LAST INPUT
Stage: BUILD / Job: Build
Build 1
JOBS
View logs and history
Deploy Passed 6m ago
Sauce Labs Tests Failed 6m ago
Functional Tests Failed 6m ago

Lab 4 Customize Toolchain to allow full deployment

Objective

This lab first updates the UI Delivery Pipeline so the UI application will fully deploy to the PROD stage, then updates the Orders and Catalog Delivery Pipelines as well.

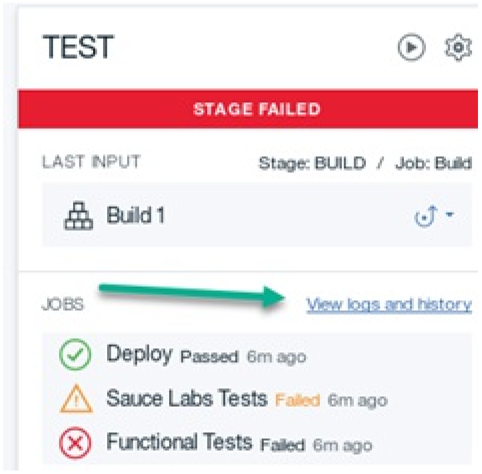
Tasks:

- Task 1: Modify Toolchain for Functional Tests job
- Task 2: Examine PROD Stage

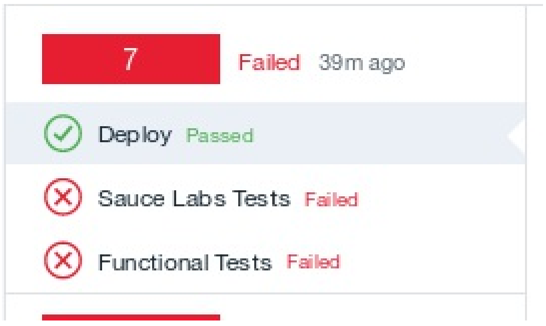
- Task 3: Update Catalog and Orders Delivery Pipelines

Task 1: Modify Toolchain for Functional Tests job

1. Click **View logs and history**.

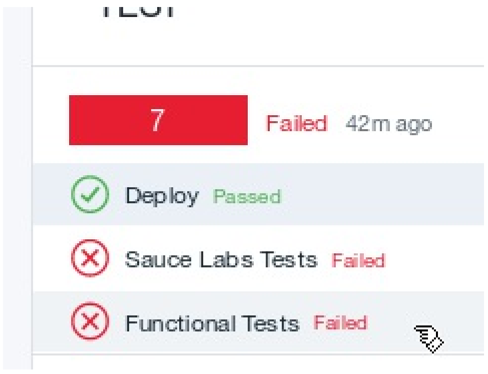

TEST
STAGE FAILED
LAST INPUT
Stage: BUILD / Job: Build
Build 1
JOBS
View logs and history
Deploy Passed 6m ago
Sauce Labs Tests Failed 6m ago
Functional Tests Failed 6m ago

2. The stage log history displays the number of times the stage has been executed and the results, showing the individual jobs.

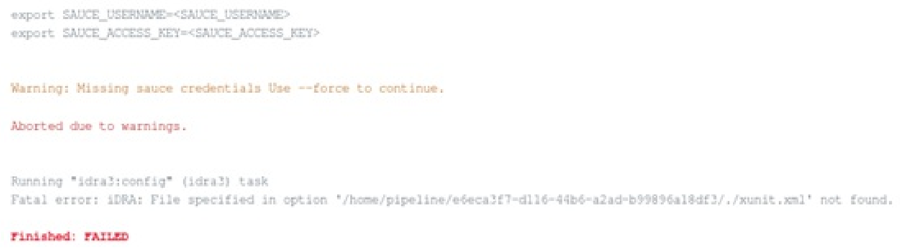


The *Deploy* job passed and the *Sauce Labs Test* job and the *Functional Tests* job both failed.

3. Click **Functional Tests** to display the log for that job.

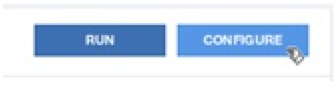


4. Scroll to the bottom of the log file. The message indicates that the missing Sauce Labs credentials are again making the job fail.

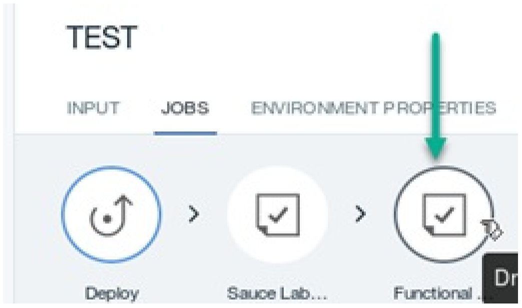


```
export SAUCE_USERNAME=<SAUCE_USERNAME>
export SAUCE_ACCESS_KEY=<SAUCE_ACCESS_KEY>


Warning: Missing sauce credentials Use --force to continue.

Aborted due to warnings.


Running "idra3:config" (idra3) task
Fatal error: iDRA: File specified in option '/home/pipeline/e6eca3f7-d116-44b6-a2ad-b99896a18df3/./xunit.xml' not found.

Finished: FAILED
```

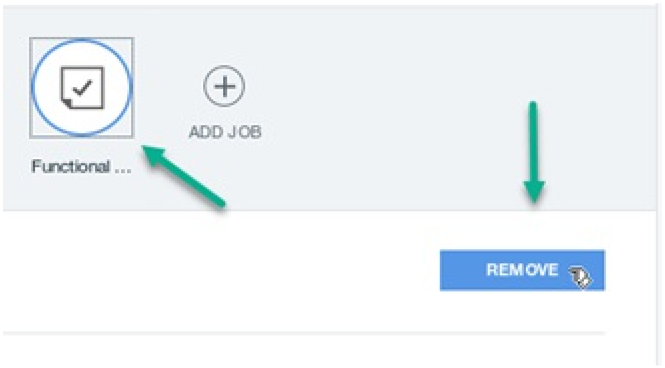
5. We could modify the *Functional Tests* job options to allow the TEST stage to continue despite the failure of the *Functional Tests* job. Instead, we will remove the job from the stage.
6. Scroll back to the top of the browser tab. Click **CONFIGURE** to configure the TEST stage.



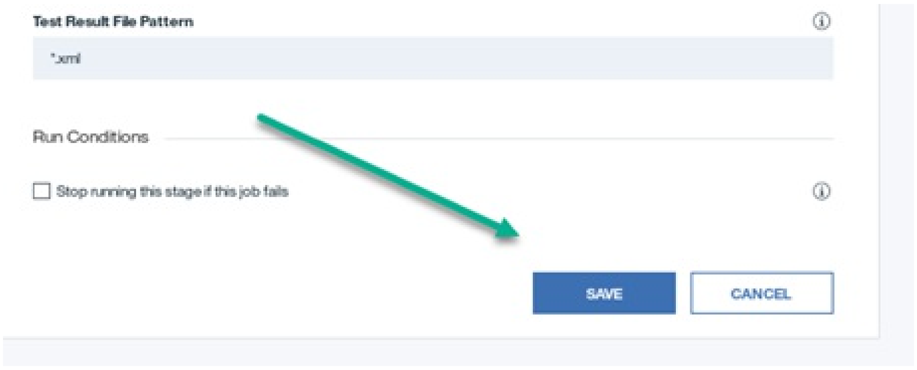
7. Click the **Functional Tests** job to select it.



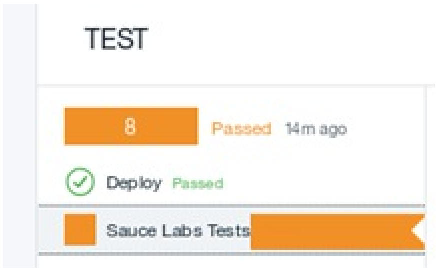
8. Click **REMOVE** to delete the *Functional Tests* job from the TEST stage. Click **OK** to confirm.



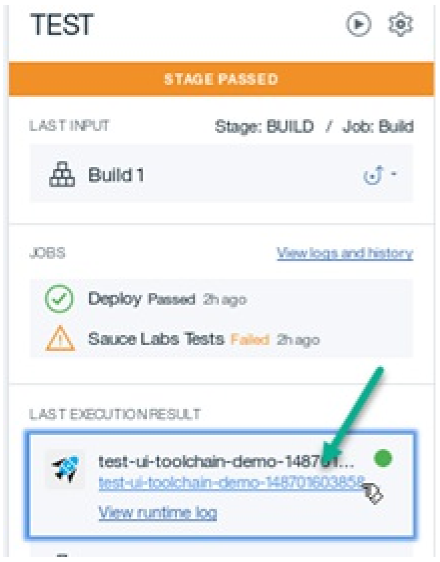
9. Scroll to the bottom of the page and click **SAVE** to save the stage changes. If you don't do this, the changes we just made will be lost.



10. The Delivery Pipeline is displayed. Run the *TEST* stage by clicking on the arrow in the TEST stage.
11. Look at both the Bluemix console and the Slack channel. The *Deploy* job runs (and passes), the *Sauce Labs Test* job fails, and the *TEST* stage completes.



12. Click on the application link to display the application running in the *TEST* space.


TEST

STAGE PASSED

LAST INPUT Stage: BUILD / Job: Build

Build 1

JOBS View logs and history

Deploy Passed 2h ago

Sauce Labs Tests Failed 2h ago

LAST EXECUTION RESULT

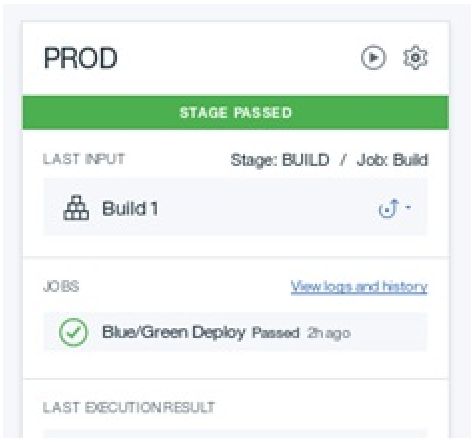
test-ui-toolchain-demo-14870...

test-ui-toolchain-demo-148701603858

View runtime log

Task 2: Examine PROD Stage

1. Since the *TEST* stage completed, the next stage, *PROD*, executed. The *PROD* stage successfully executed.


PROD

STAGE PASSED

LAST INPUT Stage: BUILD / Job: Build

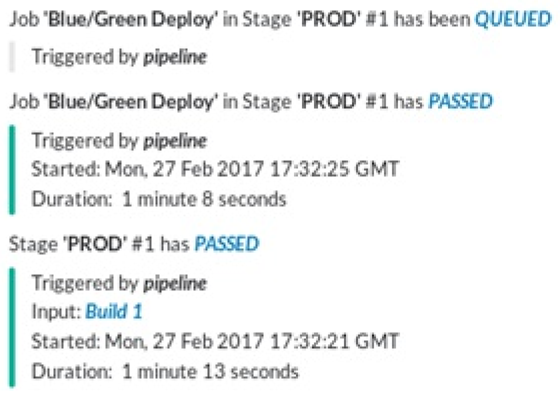
Build 1

JOBS View logs and history

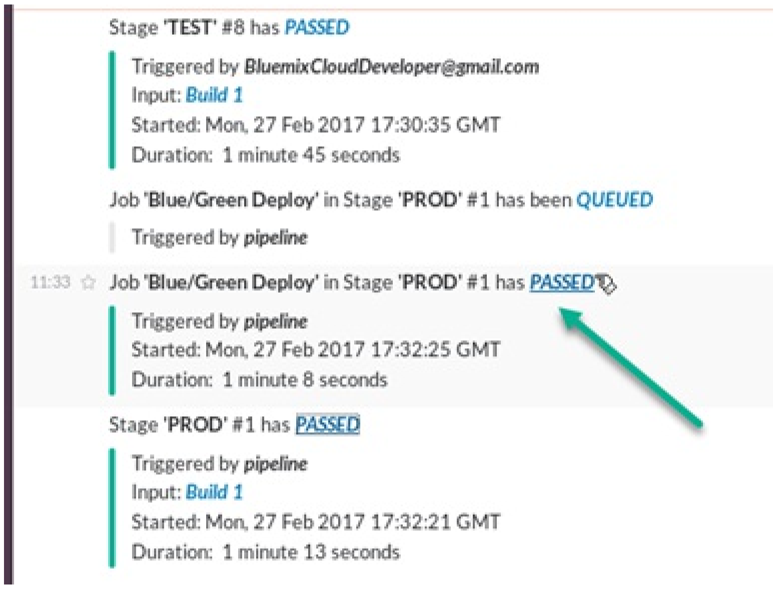
Blue/Green Deploy Passed 2h ago

LAST EXECUTION RESULT

2. Display the Slack browser tab. This also shows the successful execution of the *PROD* stage.

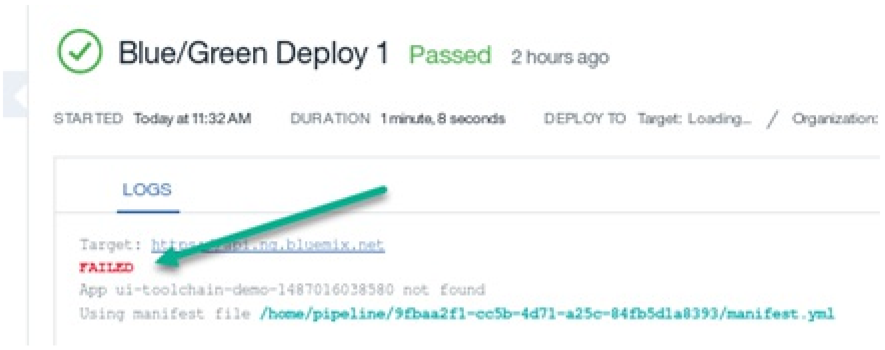

Job 'Blue/Green Deploy' in Stage 'PROD' #1 has been QUEUED
Triggered by pipeline
Job 'Blue/Green Deploy' in Stage 'PROD' #1 has PASSED
Triggered by pipeline
Started: Mon, 27 Feb 2017 17:32:25 GMT
Duration: 1 minute 8 seconds
Stage 'PROD' #1 has PASSED
Triggered by pipeline
Input: Build 1
Started: Mon, 27 Feb 2017 17:32:21 GMT
Duration: 1 minute 13 seconds

3. Click on the **PASSED** link at the end of the line *Job **'Blue/Green Deploy'** in Stage **'PROD'** #1 has passed*.


Stage 'TEST' #8 has PASSED
Triggered by BluemixCloudDeveloper@gmail.com
Input: Build 1
Started: Mon, 27 Feb 2017 17:30:35 GMT
Duration: 1 minute 45 seconds
Job 'Blue/Green Deploy' in Stage 'PROD' #1 has been QUEUED
Triggered by pipeline
11:33 Job 'Blue/Green Deploy' in Stage 'PROD' #1 has PASSED
Triggered by pipeline
Started: Mon, 27 Feb 2017 17:32:25 GMT
Duration: 1 minute 8 seconds
Stage 'PROD' #1 has PASSED
Triggered by pipeline
Input: Build 1
Started: Mon, 27 Feb 2017 17:32:21 GMT
Duration: 1 minute 13 seconds

This displays the job log for the *Blue/Green Deploy* job in the *PROD* stage. We could have gotten to this display from the Bluemix console, but clicking on the link in Slack accesses the log directly from Slack without the need to go through Bluemix. ***Note***: You may need to wait a moment or two or click this link multiple times depending on how quickly the application is deployed.

4. The log shows the first step of the job failing.


Blue/Green Deploy 1 Passed 2 hours ago
STARTED Today at 11:32 AM
DURATION 1 minute, 8 seconds
DEPLOY TO Target: Loading... / Organization:
LOGS

```
Target: https://api.ng.bluemix.net
FAILED
App ui-toolchain-demo-1487016038580 not found
Using manifest file /home/pipeline/9fbaa2f1-cc5b-4d71-a25c-84fb5d1a8393/manifest.yml
```

5. A look at the script used in this job (by clicking on **CONFIGURE** in the upper right hand corner) shows the following:

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

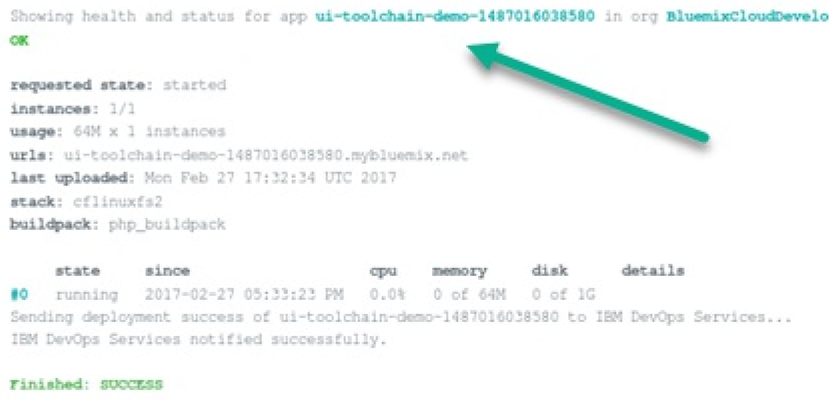
This script does a *Blue-green* deployment. A blue-green deployment is a release technique reducing downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic.

The script first issues the Cloud Foundry (cf) app command for the UI application to be deployed. If the command returns without finding the application, then the script pushes (deploys) the UI application.

```
#!/bin/bash
    if ! cf app $CF_APP; then
      cf push $CF_APP
```

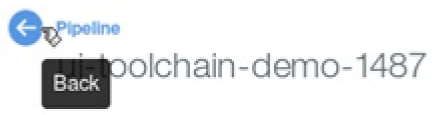
This is what happens the first time this job and script are run, as the application has never been deployed.

6. The end of the log shows the successful deployment. All that failed was finding a running UI application.

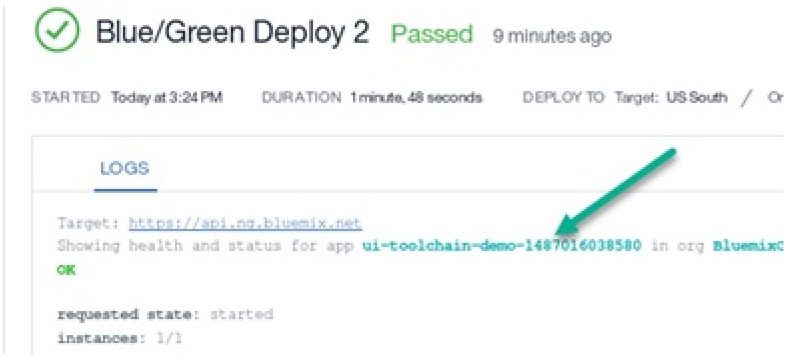


```
Showing health and status for app ui-toolchain-demo-1487016038580 in org BluemixCloudDevelo
OK

requested state: started
instances: 1/1
usage: 64M x 1 instances
urls: ui-toolchain-demo-1487016038580.mybluemix.net
last uploaded: Mon Feb 27 17:32:34 UTC 2017
stack: cflinuxfs2
buildpack: php_buildpack

     state     since                    cpu    memory    disk      details
#0   running   2017-02-27 05:33:23 PM   0.0%   0 of 64M   0 of 1G
Sending deployment success of ui-toolchain-demo-1487016038580 to IBM DevOps Services...
IBM DevOps Services notified successfully.

Finished: SUCCESS
```

7. Return to the delivery pipeline (by clicking on the left arrow next to *Pipeline*).

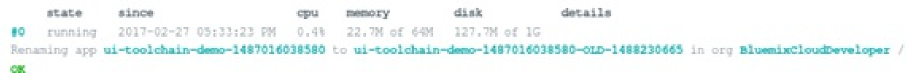


8. Run the *PROD* stage again by clicking on the arrow next to *PROD*.
9. Wait for the *PROD* stage to complete, remembering you can watch either the Bluemix console or the Slack channel.
10. Once the *PROD* stage completes, look at the job log for the deployment. It shows the first step of the job finding a running UI application.


Blue/Green Deploy 2 Passed 9 minutes ago
STARTED Today at 3:24 PM
DURATION 1 minute, 48 seconds
DEPLOY TO Target: US South / Or
LOGS

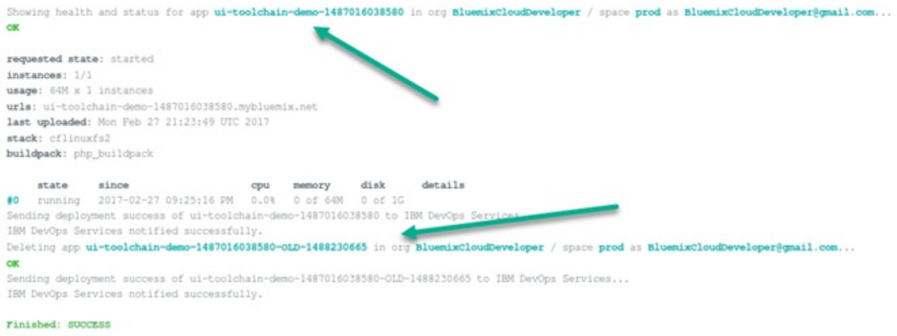
```
Target: https://api.ng.bluemix.net
Showing health and status for app ui-toolchain-demo-1487016038580 in org BluemixC
OK

requested state: started
instances: 1/1
```

11. Scrolling down the log further, we see the old UI application being renamed and the new one being deployed.



```
     state     since                    cpu    memory        disk          details
#0   running   2017-02-27 05:33:23 PM   0.4%   22.7M of 64M  127.7M of 1G
Renaming app ui-toolchain-demo-1487016038580 to ui-toolchain-demo-1487016038580-OLD-1488230665 in org BluemixCloudDeveloper /
OK
```

12. Finally, once the new UI application is successfully deployed, the old one is deleted.



```
Showing health and status for app ui-toolchain-demo-1487016038580 in org BluemixCloudDeveloper / space prod as BluemixCloudDeveloper@gmail.com...
OK

requested state: started
instances: 1/1
usage: 64M x 1 instances
urls: ui-toolchain-demo-1487016038580.mybluemix.net
last uploaded: Mon Feb 27 21:23:49 UTC 2017
stack: cflinuxfs2
buildpack: php_buildpack

     state     since                    cpu    memory    disk      details
#0   running   2017-02-27 09:25:16 PM   0.0%   0 of 64M  0 of 1G
Sending deployment success of ui-toolchain-demo-1487016038580 to IBM DevOps Services...
IBM DevOps Services notified successfully.
Deleting app ui-toolchain-demo-1487016038580-OLD-1488230665 in org BluemixCloudDeveloper / space prod as BluemixCloudDeveloper@gmail.com...
OK
Sending deployment success of ui-toolchain-demo-1487016038580-OLD-1488230665 to IBM DevOps Services...
IBM DevOps Services notified successfully.

Finished: SUCCESS
```

13. Return to the Pipeline by clicking on the arrow to the left of *Pipeline* in the upper left corner.
14. Return to the Toolchain by clicking on the arrow to the left of *Toolchain* in the upper left corner.

Task 3: Update Catalog and Orders Delivery Pipelines

1. The Catalog and Orders delivery pipeline have the same problems as the UI delivery pipeline. Modify both to remove the *Sauce Labs Tests* and the *Functional Tests* jobs (or make failure of those jobs non-blocking) from the *TEST* stage of those Delivery Pipelines. If you are uncertain, take a look at the steps used for removing the *Functional*

Test job from the *TEST* stage of the UI delivery pipeline.

Lab 5 Customize Toolchain to add Bluemix Availability Monitoring

Objective

This lab ...

Tasks:

- Task 1: ...
- Task 2: ...
- Task 3: ...

Lab 6 Add Bluemix IBM Alert Notification

Objective

This lab ...

Tasks:

- Task 1: ...
- Task 2: ...
- Task 3: ...

Lab 7 Modify Pipeline for Catalog to deploy Catalog to Containers

Objective

This lab ...

Tasks:

- Task 1: ...
- Task 2: ...
- Task 3: ...

Lab 8 Add auto-scaling support to Catalog

Objective

This lab ...

Tasks:

- Task 1: ...
- Task 2: ...
- Task 3: ...