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.

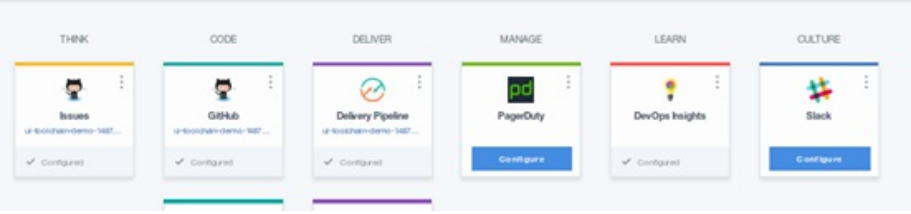
Tasks:

- 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.

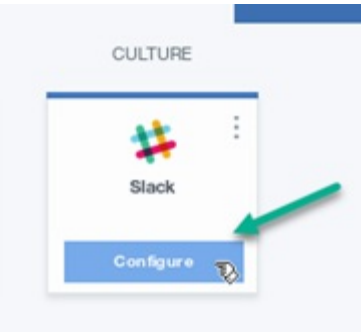
   toolchain-demo-1487016038580

   If not, click on the hamburger menu, then **Menu**. Then click on **Services** then **DevOps**. Then click on **Toolchains**. And finally click on

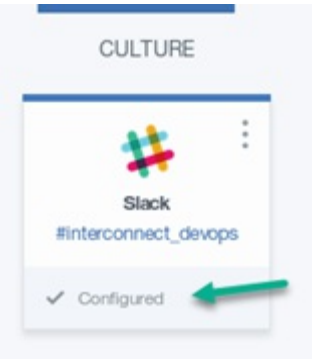
the toolchain you created.

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



4. Enter the following information:
    1. Slack webhook (all one string): **https://hooks.slack.com/services/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**
    2. Slack channel: **interconnect_devops**
    3. Slack team URL Host name: **bluemixdevopslab**

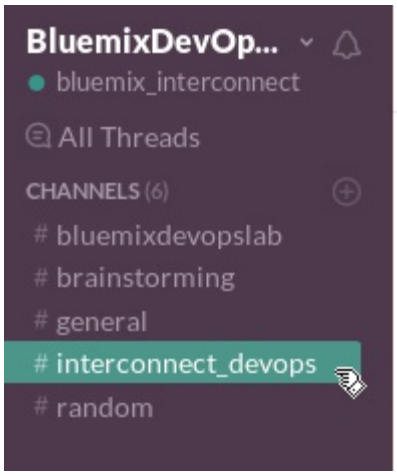
5. Click **Save Integration** to save the information.
6. Slack should now be Configured



Task 2: Work with Slack

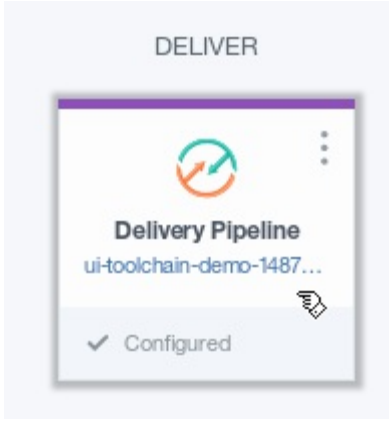
1. In the browser, open a new tab and go to the following URL to go to the (already created) Slack team. https://bluemixdevopslab.slack.com
2. Enter the following information:
    1. Email address: **BluemixInterConnectDevOps@gmail.com**
    2. Password: **devops4me**

    and click **Sign In**.
3. Click on the **interconnect_devops** channel to show the messages for that channel.

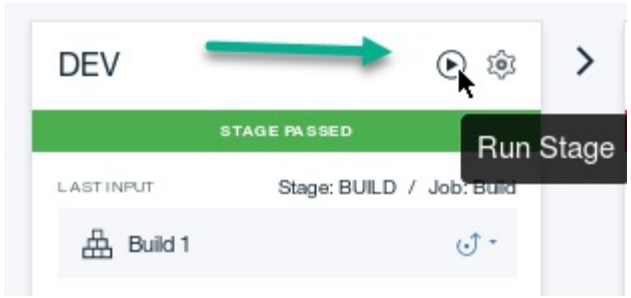


This channel will show all the messages the Toolchain sends to it.

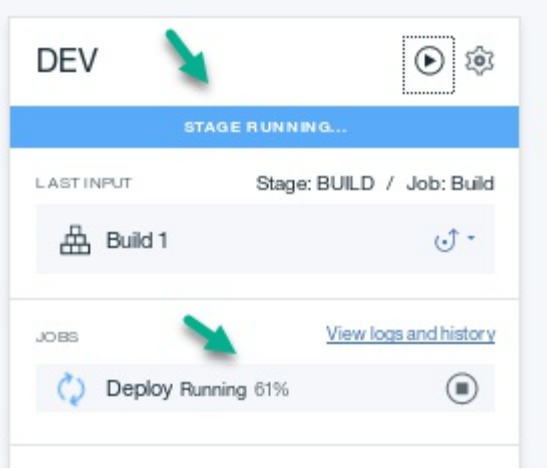
4. Leaving the Slack browser tab open, switch over to the Toolchain browser tab.
5. Click on the UI Delivery Pipeline tile to display the UI Delivery Pipeline.



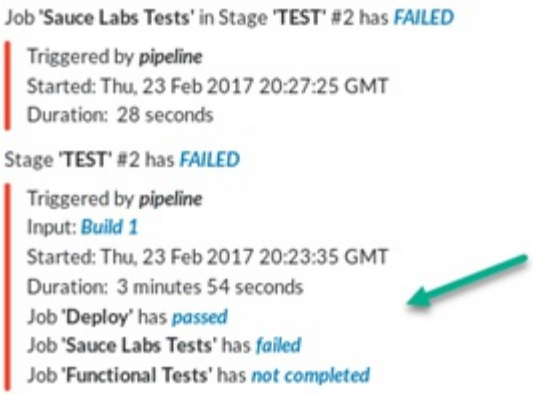
6. Click on the arrow in the DEV stage to run the jobs in the DEV stage.



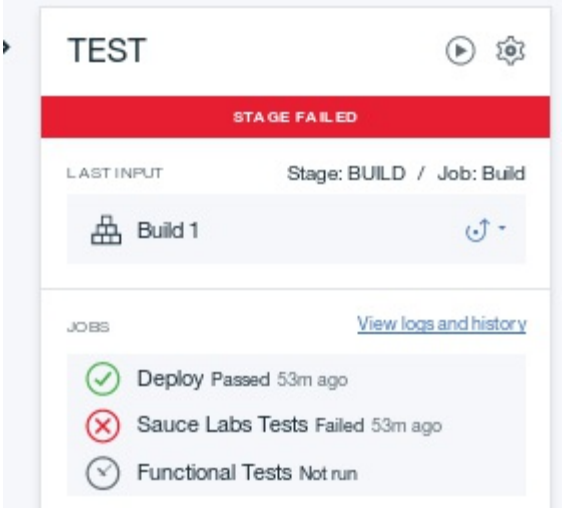
7. The DEV stage runs the jobs and you get a visual indication of the progress of the stage.



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.

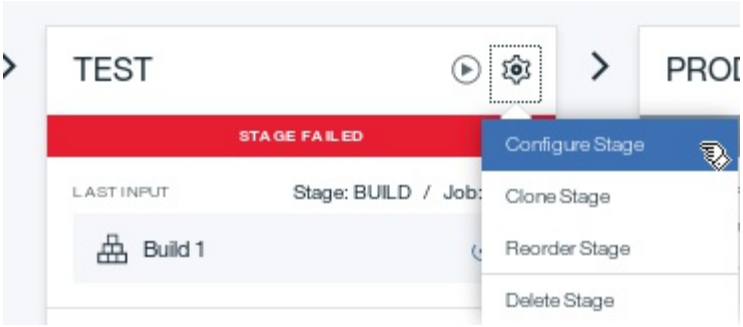


9. Switch back to the Bluemix browser tab. The Toolchain indicates the same results. Something is wrong with the Toolchain.

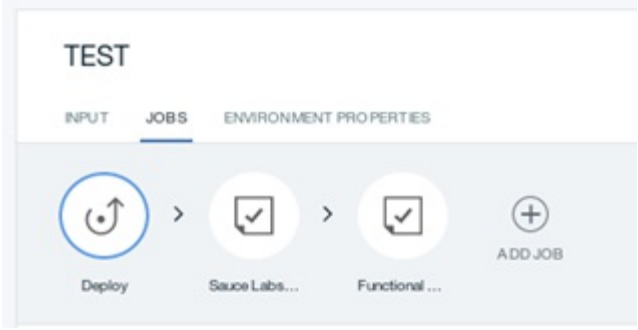


Task 3: Modify Toolchain for Sauce Labs test job

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**.

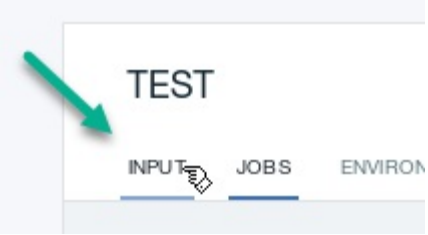


2. The TEST stage has three jobs, *Deploy*, *Sauce Labs Test*, and *Functional Tests*.



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.

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.

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**.

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.

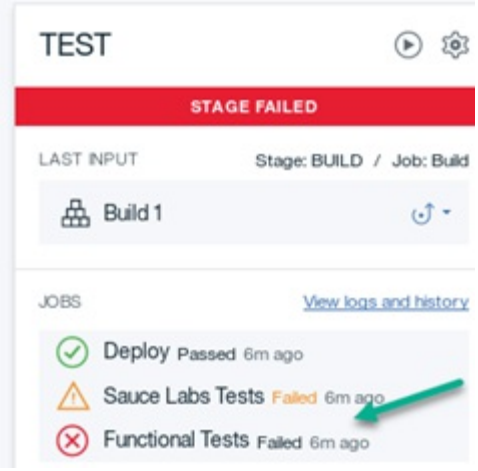
Job **'Deploy'** in Stage **'TEST'** #6 has ***PASSED***

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.



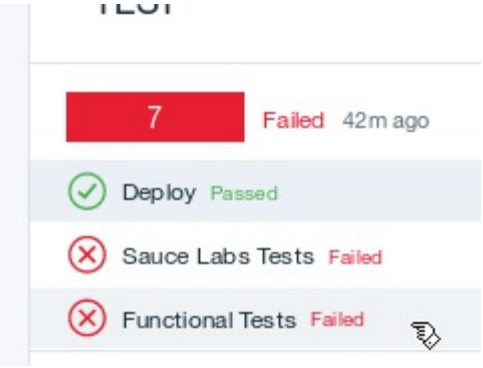
Task 4: Modify Toolchain for Functional Tests job

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

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.

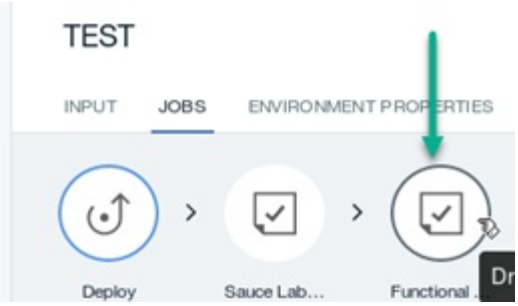
3. Click **Functional Tests** to display the log for that job.



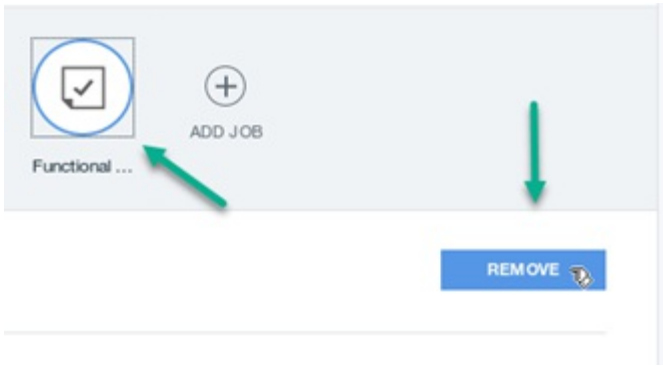
4. Scroll to the bottom of the log file. The message indicates that the missing Sauce Labs credentials are again making the job fail.

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.

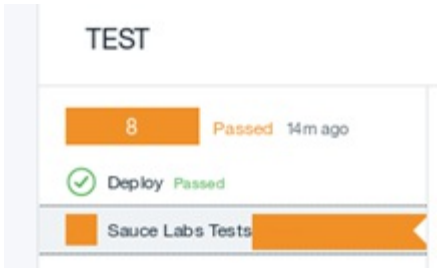
7. Click the **Functional Tests** job to select it.



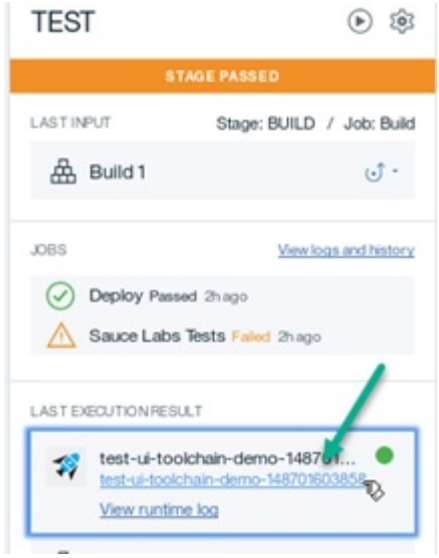
8. Click **REMOVE** to delete the *Functional Tests* job from the TEST stage. Click **OK** to confirm.



9. Scroll to the bottom of the page and click **SAVE** to save the stage changes.
10. The Deivery 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.

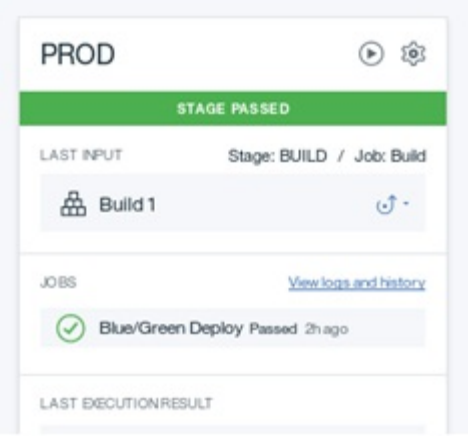


12. Click on the application link to display the application running in the *TEST* space.

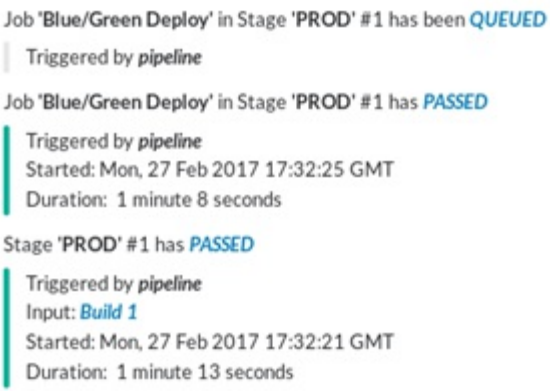


Task 5: Examine PROD Stage

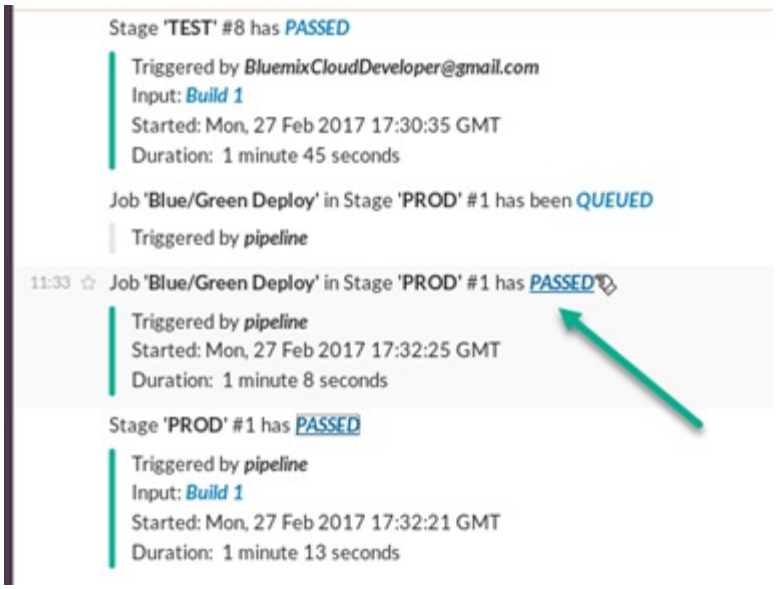
1. Since the *TEST* stage completed, the next stage, *PROD*, executed. The *PROD* stage successfully executed.



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



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



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.

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

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.

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.

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

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

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.