Getting Started with Jenkins X

Class Labs : Revision 1.1 - 9/8/19

Brent Laster

Important Note: Prior to starting with this document, you should have the ova file for the class (the virtual machine already loaded into Virtual Box and have ensured that it is startable. See the setup doc jxi-setup.pdf in http://github.com/brentlaster/safaridocs for instructions and other things to be aware of. You should also have a GitHub account.

Lab 1: Creating a Cluster

Purpose: In this lab, we'll see how to create a Kubernetes cluster starting without one on our minikube system.

1. If not already up and running, start up your VM session.

2. Click on the Terminal Emulator icon to open a new terminal session.

3. Let's create our initial cluster using jx. Enter the line below. (Note that we add the –skip-intallation argument here because we just want to create the cluster and not install jx on it yet.)

   sudo jx create cluster minikube --skip-installation=true

4. At the prompt for the amount of memory, enter **12000**.

5. At the prompt for the number of cpu cores, enter **5**.

6. At the prompt for the disk size, enter **35GB**.

7. At the prompt for the driver, use the down arrows and select "**none**" (or hit the "**n**" key) and **then Enter**.

8. This should then create the cluster. **It will take a few minutes to run.** After this is done, we still need to run a command to "fix" a couple of the DNS pods in the cluster. Execute the command below. (You can look at the script to see what it is doing if interested.)

   ./fixdns.sh

9. Now, let's start up the Kubernetes dashboard and have a look around at our cluster. Do this by running the command below.

   minikube dashboard &

10. If you want, you can explore the cluster by looking at the various elements selectable via the dashboard.

Lab 2: Installing Jenkins X on the Cluster

Purpose: In this lab, we'll see how to install Jenkins X (jx) on the cluster we created in lab 1.

1. To install Jenkins X on the cluster, we'll use the jx install command. Like the jx create cluster command, we have some options to select on this. Start by invoking the command below. (The --default-admin-password option is to avoid having a long, difficult system-generated password for our Jenkins instance). (If you are not back at the prompt in your terminal session, you can just Enter before putting in the command below.)

   jx install --default-admin-password=admin

2. For this initial example, we'll leverage the "**Static Jenkins Server and Jenkinsfile**" implementation. At the "Select Jenkins Installation Type" prompt, arrow down and select that option and hit Enter. (This may run for a bit as it creates the jx namespace and sets the default namespace context to it.)

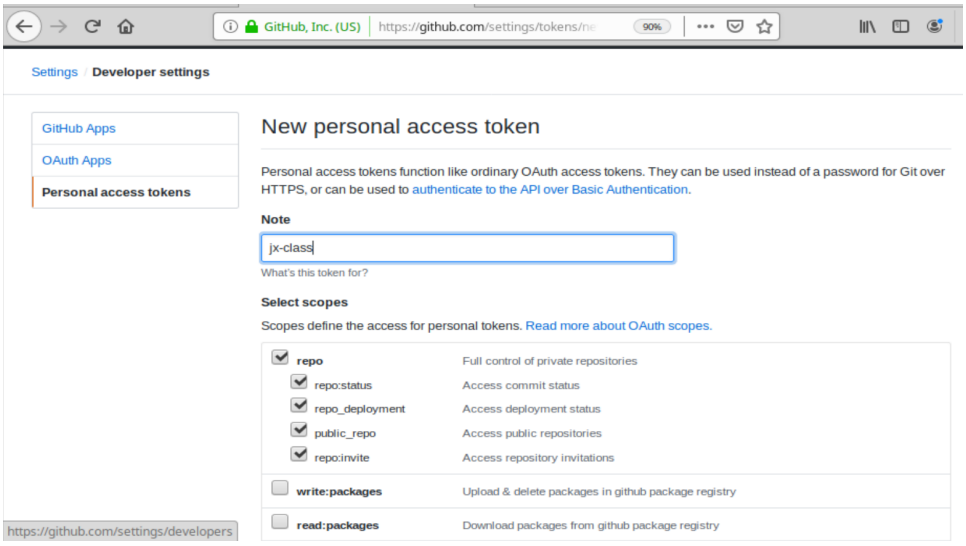
3. At the prompt for "Please enter the name you wish to use with git:", enter your name – usually in the form First **Last**

4. At the prompt for "Please enter the email address you wish to use with git:", enter your email address.

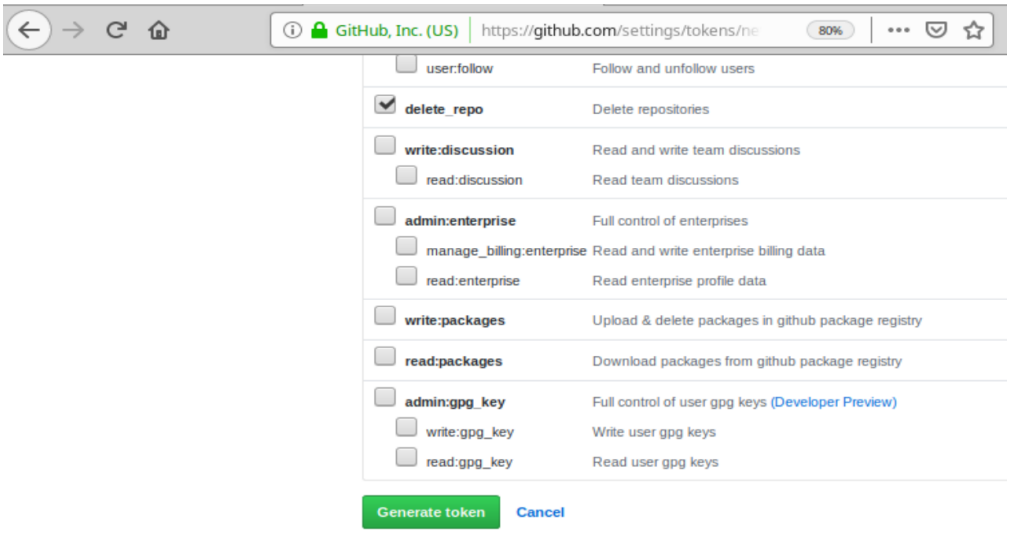
5. At the prompt for "GitHub username:", enter your GitHub username.

6. You'll then need to supply a GitHub API token to have access to GitHub. jx will generate a URL for you. Right click on that URL and select "Open Link". (Don't worry if you see an error in the background from Firefox.)

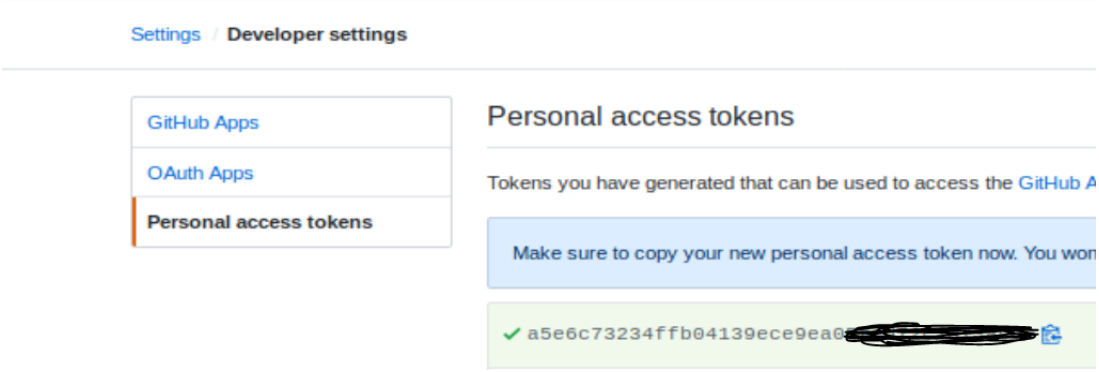
7. Log in to GitHub and you'll be on the right screen to generate a token for access. In the Note field, you can just enter any kind of identifier/name, such as "jx-class".



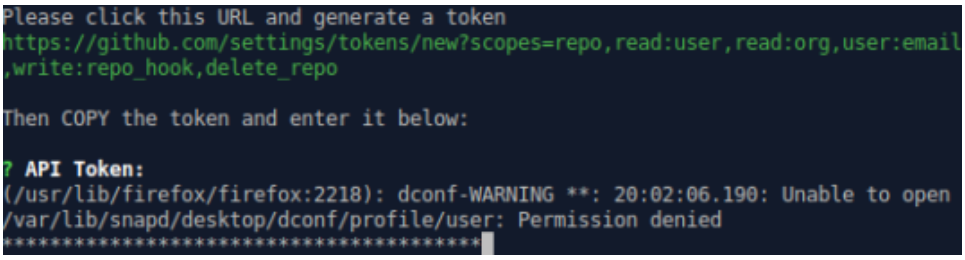
8. Scroll down to the bottom and click on the green “Generate token” button.



9. You’ll then see your new personal access token displayed. Select it and copy it (or just click the icon on the end to copy).



10. **Paste the copied token** (right-click and select Paste) back at the “API Token” prompt in the terminal session and **hit Enter**. See screenshot below. Paste it under the "Permission denied" line.



(Note: You may want to save the token string in a file somewhere or email it to yourself as you won't be able to get to the actual string again.)

11. At the prompt for “Do you wish to use GitHub as the pipelines Git server: (Y/n), just hit Enter to select the default Yes answer.

12. You’ll be prompted again for a GitHub API Token for a Git user for the pipeline processes (the jx bot). We can use the same token as before, so just paste it again and hit Enter.

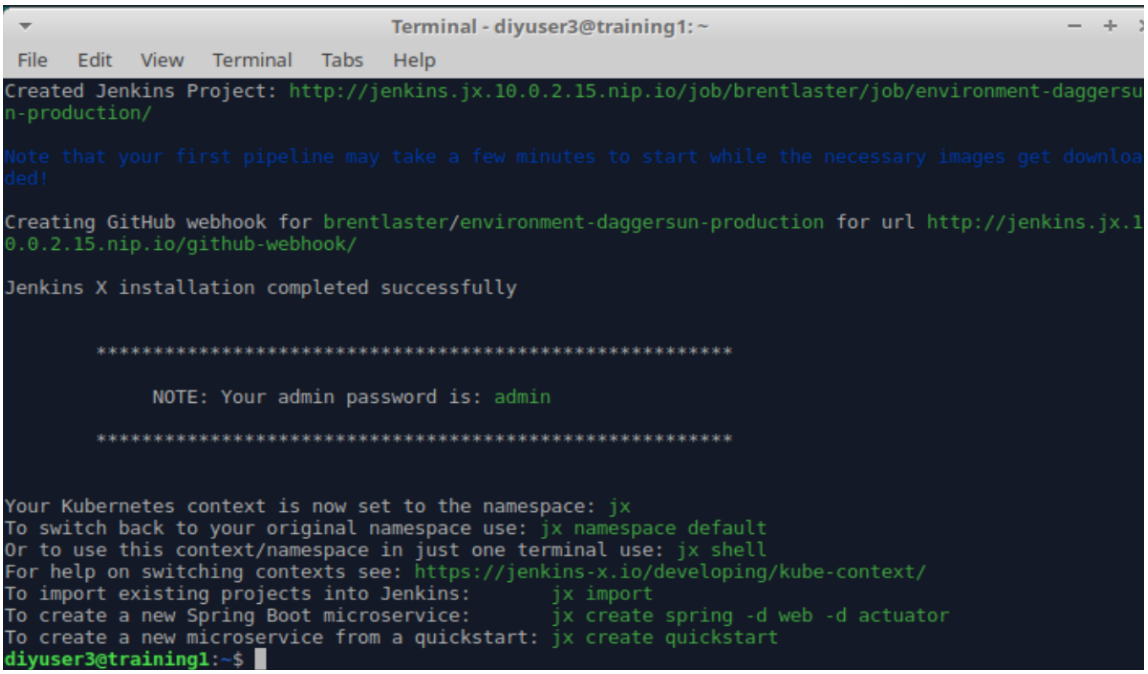
13. This will take a while. You can leave it running while we cover the next section. (You can also ignore the WARNING about vault.)

Lab 3: Creating a Quickstart Project

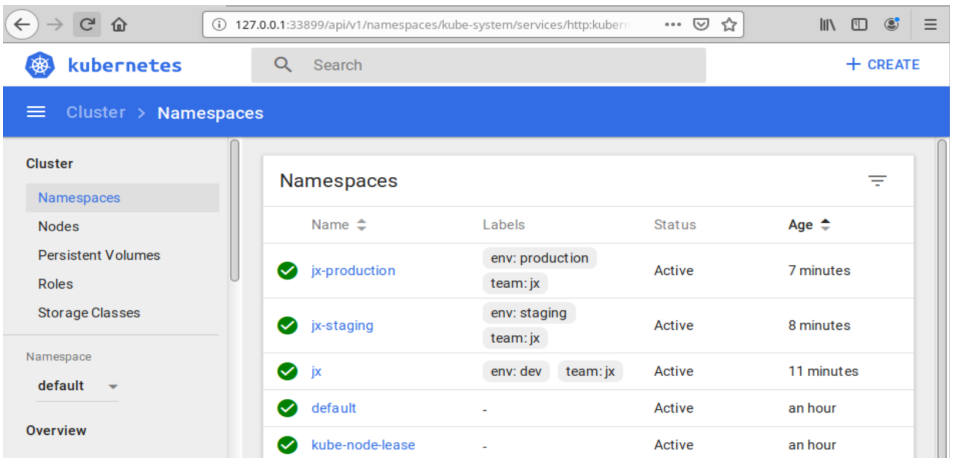
Purpose: In this lab, we’ll finish setting up our jx install and then create a Quickstart project with it.

1. At this point, your install will probably be at the point of asking you to “Select the organization where you want to create the environment repository:”. It should also be displaying your GitHub userid as the suggested response. (If you have multiple organizations, choose the one you want to use.) Go ahead and hit Enter to accept that.

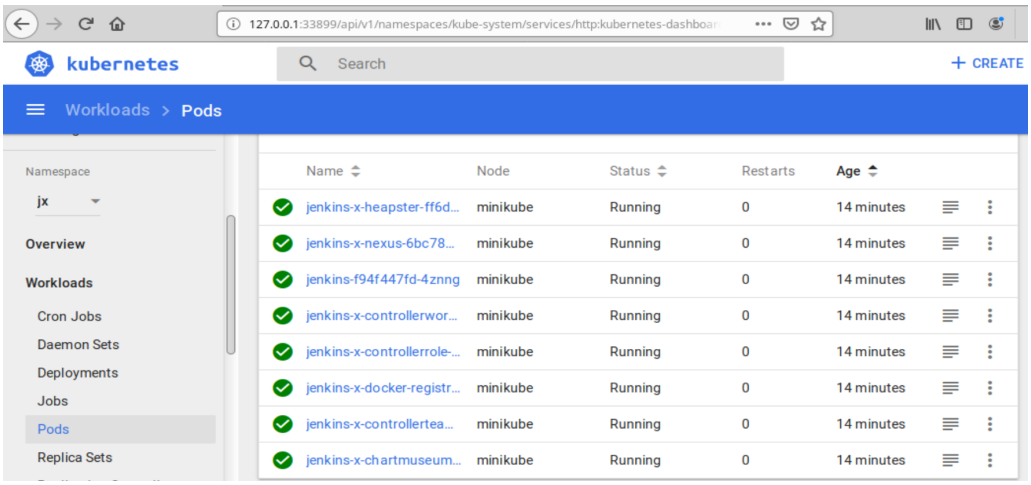
2. After a few moments, you should see a completion message saying “Jenkins X installation completed successfully. It will also show your admin password and other information.



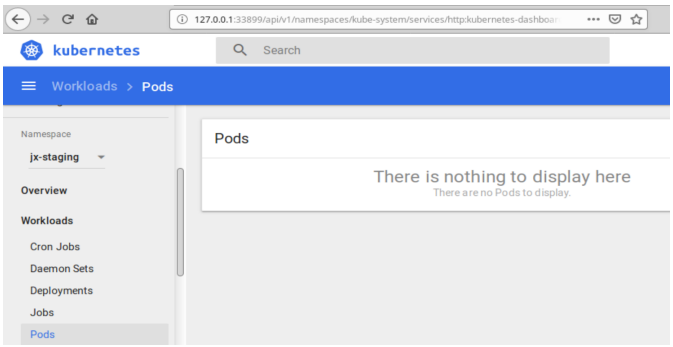
3. Now, let’s take a look around and see what Jenkins created for us. Switch back to the Kubernetes dashboard in your browser. Click on the “Namespaces” link under the “Cluster” link in the left menu. Notice that you now have “jx”, “jx-staging” and “jx-production” namespaces.



4. Let's look at all the different pods that were created for the different applications in the jx namespace. In the left menu, just below the horizontal bar, under "Namespace", click the down arrow next to "default" and change it to "jx". Then under "Workloads", click the "Pods" link to see the various applications running in pods in this workspace. (You can expand your browser if you can't see the whole name or hover over it.



5. Set the "Namespace" to the "jx-staging" one and notice that there are no pods here yet. (If you don't see it, you may need to force a refresh by hitting F5.)



6. One of the things this process did was create a staging GitHub repository and a production GitHub repository for our use. We'll look at the production one first. In the terminal window, run the command below.

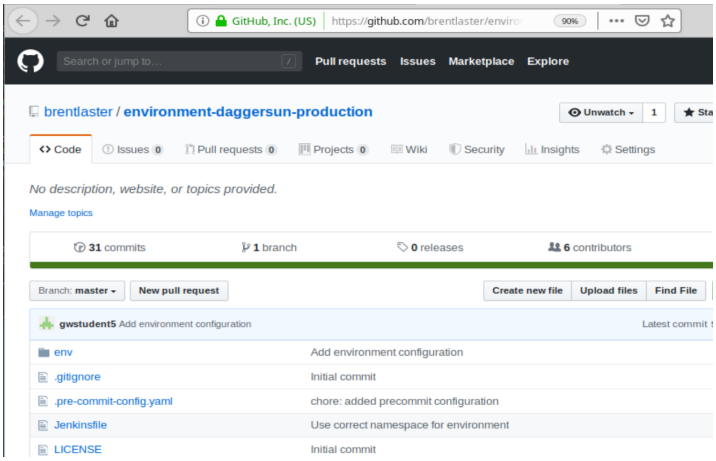
   jx get env

7. Hover over the link for the "production" one, right click and select "Open Link".

```
diyuser3@training1:~$ jx get env
NAME       LABEL       KIND        PROMOTE NAMESPACE     ORDER CLUSTER SOURCE
                                   REF PR
dev        Development Development Never   jx            0

staging    Staging     Permanent   Auto    jx-staging    100           https://github.com/brentlaste
r/environment-daggersun-staging.git
production Production  Permanent   Manual  jx-production 200           https://github.com/brentlaste
r/environment-daggersun-production.git
```

8. You should be taken to the GitHub location for your production repository.

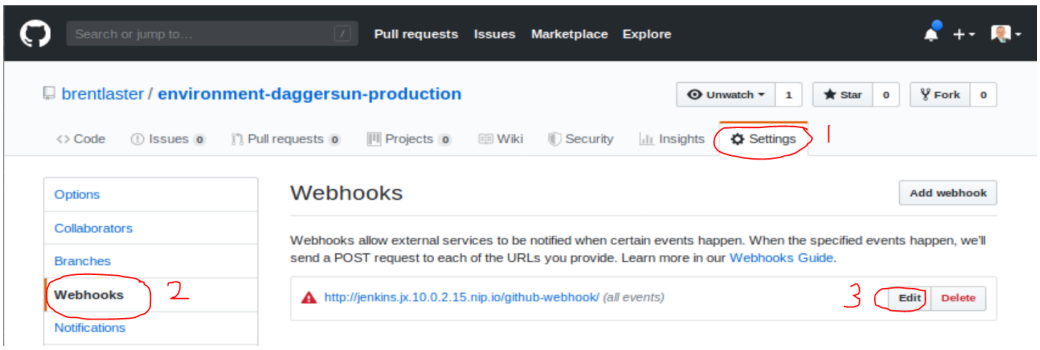


9. While we're here, let's look at the basic Jenkinsfile that was created for Jenkins to run our project. Click on the "Jenkinsfile" link. Take a look at what the Jenkinsfile is doing.

35 lines (34 sloc) | 564 Bytes

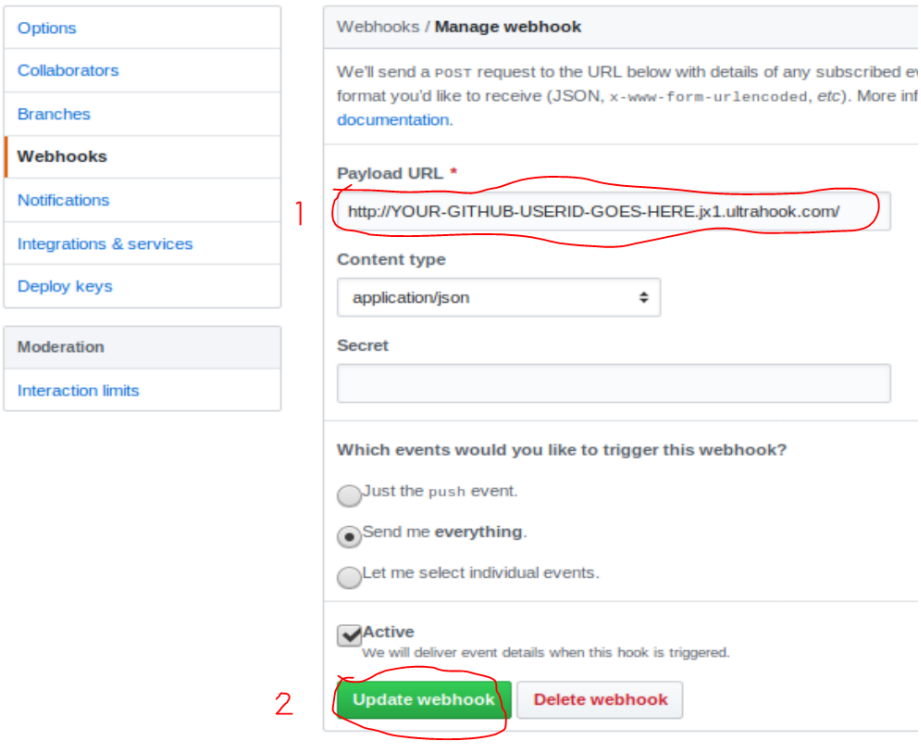
```
pipeline {
  options {
    disableConcurrentBuilds()
  }
  agent {
    label "jenkins-maven"
  }
  environment {
    DEPLOY_NAMESPACE = "jx-production"
  }
  stages {
    stage('Validate Environment') {
      steps {
        container('maven') {
          dir('env') {
            sh 'jx step helm build'
          }
        }
      }
    }
    stage('Update Environment') {
      when {
        branch 'master'
      }
      steps {
        container('maven') {
          dir('env') {
            sh 'jx step helm apply'
          }
        }
      }
    }
  }
}
```

10. While we are here, we need to modify the webhook that was setup for us so it will work with our local instance. Go back one page in the browser. Near the top of the screen, in the line of tabs that starts with "<> Code", click on the **Settings** tab (at the end). Then in the left menu, click on **Webhooks**. Then click on the **Edit** button on the right.

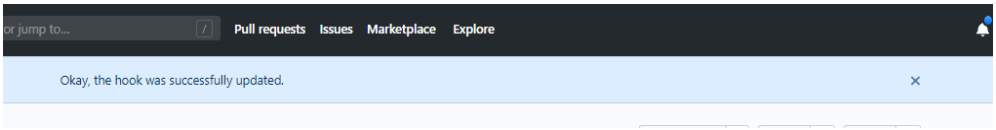


11. In the **Webhooks/Manage** webhook screen, change the **Payload URL** field to be the link below (substituting your github userid for "<your github userid>"). That's the only thing you need to change. After you make the change, click on the **Update webhook** button. (You may then need to enter your password.)

    http://**<your github userid>**.jx1.ultrahook.com/



12. If all went well, you should see a message at the top that tells you the hook was successfully updated.



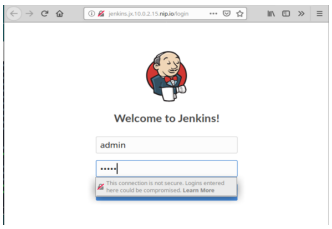
13. **Make the same webhook change for the staging environment**. You can get that link back in the terminal window with the list of environments. After you open up that link, repeat steps 10-12 above for the -staging repository.

```
diyuser3@training1:~$ jx get env
NAME       LABEL       KIND        PROMOTE NAMESPACE     ORDER CLUSTER SOURCE
                                       REF PR
dev        Development Development Never   jx            0
staging    Staging     Permanent   Auto    jx-staging    100           https://github.com/brentlaste
r/environment-daggersun-staging.git
production Production  Permanent   Manual  jx-production 200           https://github.com/brentlaste
r/environment-daggersun-production.git
```

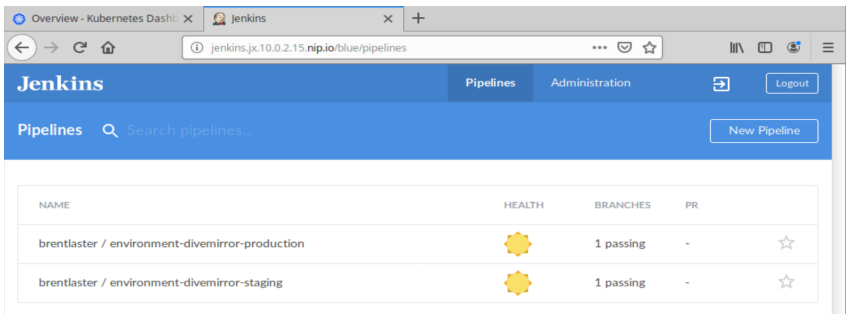
14. Next let's look at the Jenkins project that was setup for us. Connect to the static Jenkins instance by typing

    jx console

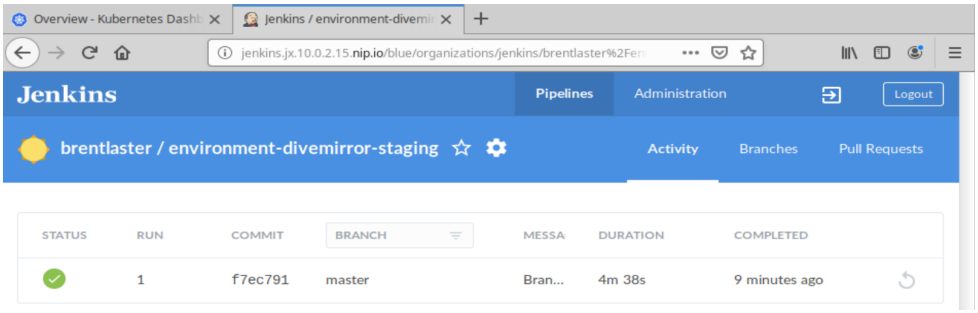
15. This should take you to the sign-in screen for the Jenkins instance that is running in the jx namespace in the cluster. Login with userid of "admin" and password of "admin".



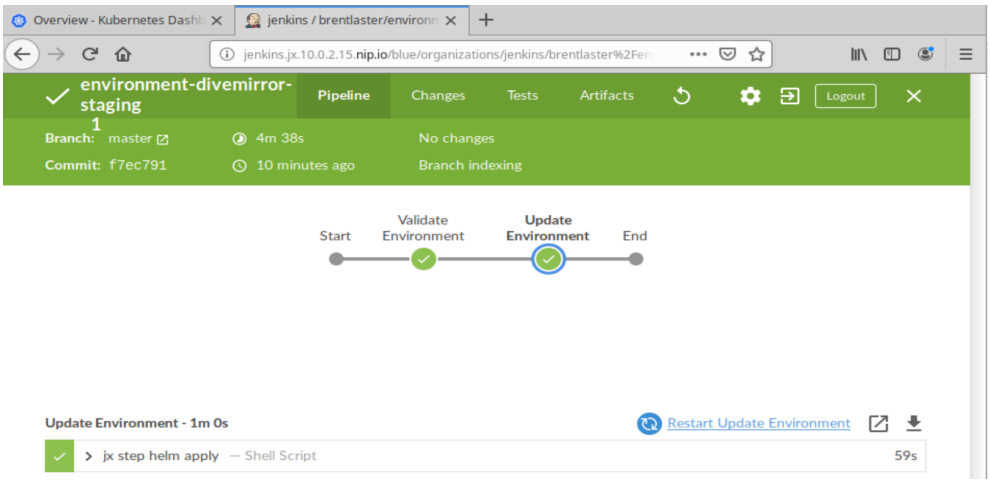
16. This will put you on the Blue Ocean interface screen for the pipelines associated with the production and staging environments.



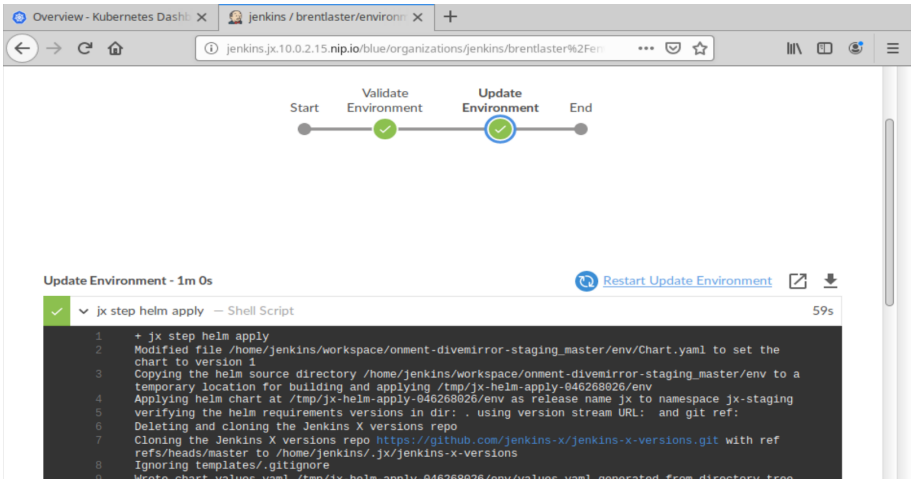
17. Select one of the jobs and click on the name. These are multi-branch pipelines which means it creates jobs in Jenkins based on the branches in the GitHub repository that have Jenkinsfiles. For right now, this is only for master.



18. Click on the "master" branch and you can see the Blue Ocean output. Notice that the stages here correspond to the stages that were defined in the Jenkinsfile.



19. If you want to see more details on the build, you can click on the steps at the bottom to expand them. Note that the steps will vary depending on which circle you have selected.



20. We need to fix up one more thing to avoid an error later in our build. We need to change Docker to use the insecure-registry that is running in our jx namespace. In the original terminal session, run the command below.

    ./fixregistry.sh

It will take a little bit to complete, but you can leave it running while we continue. It will also interrupt your Jenkins session and restart it but that's ok. (You may see a small box in your browser pop up that says "Connection lost: waiting" - that's ok. It will reload eventually.)

Lab 4: Creating a Quickstart project

Purpose: In this lab, we'll see how easy it is to create a simple Go project via a quickstart and see it go through our pipeline.

1. We're going to use an application called Ultrahook to allow the jx webhook functionality to be used with our minikube instance. Start that by opening a separate terminal session and running the following command, passing your GitHub userid as the one argument.

   ./runhook.sh <your GitHub userid>

   (For example, mine would be ./runhook.sh brentlaster)

2. In your original terminal session, issue the command to start creating the quickstart.

   jx create quickstart

3. At the prompt about the Git user name, just hit Enter to use the GitHub username we've been using.

4. If prompted about the GitHub organization, just select the one that matches the one you selected before – probably your GitHub username.

5. At the prompt for the new repository name, enter a name such as "go-proj1". (If you want to use a different name, you can – just use that name wherever we use "go-proj1" in the rest of the labs.)

6. At the prompt about selecting the quickstart you wish to create, use the arrow keys to select the "golang-http" project type and hit Enter.

7. At the prompt about "initialize git now", just hit Enter to take the default yes option.

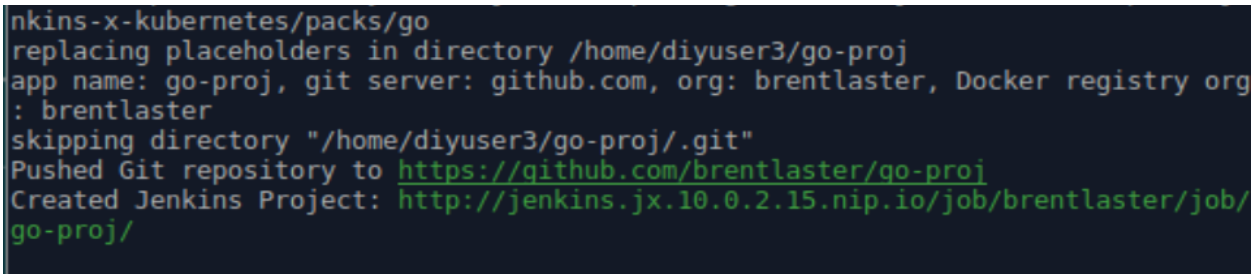
8. At the commit message prompt, you can type a new commit message or just hit Enter to accept the default one.

9. After a few moments, you should see the messages indicating that the project has been created. Notice that it has pushed it up to a GitHub repository and created a new Jenkins project for it.

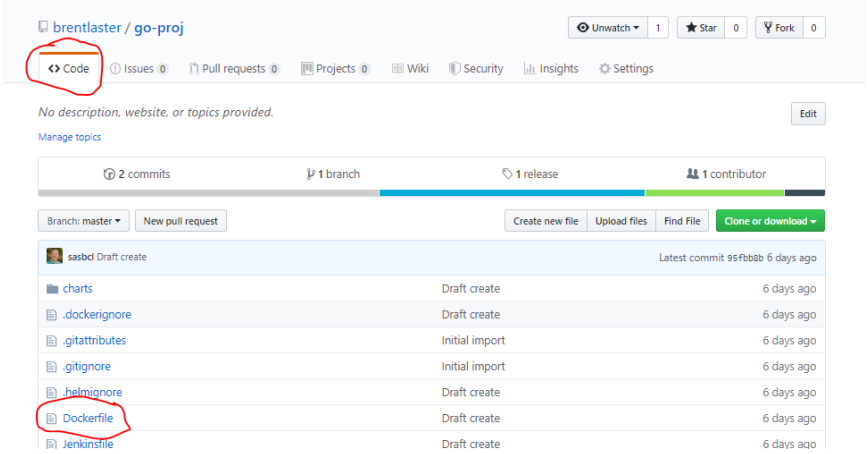
10. Now let's update the webhook on this project so it can send notifications to our local system. For this one, Jenkins X provides a command line way to do this. Run the command below (substituting in your github userid where noted).

    jx update webhooks --endpoint http://<your github userid>.jx1.ultrahook.com/

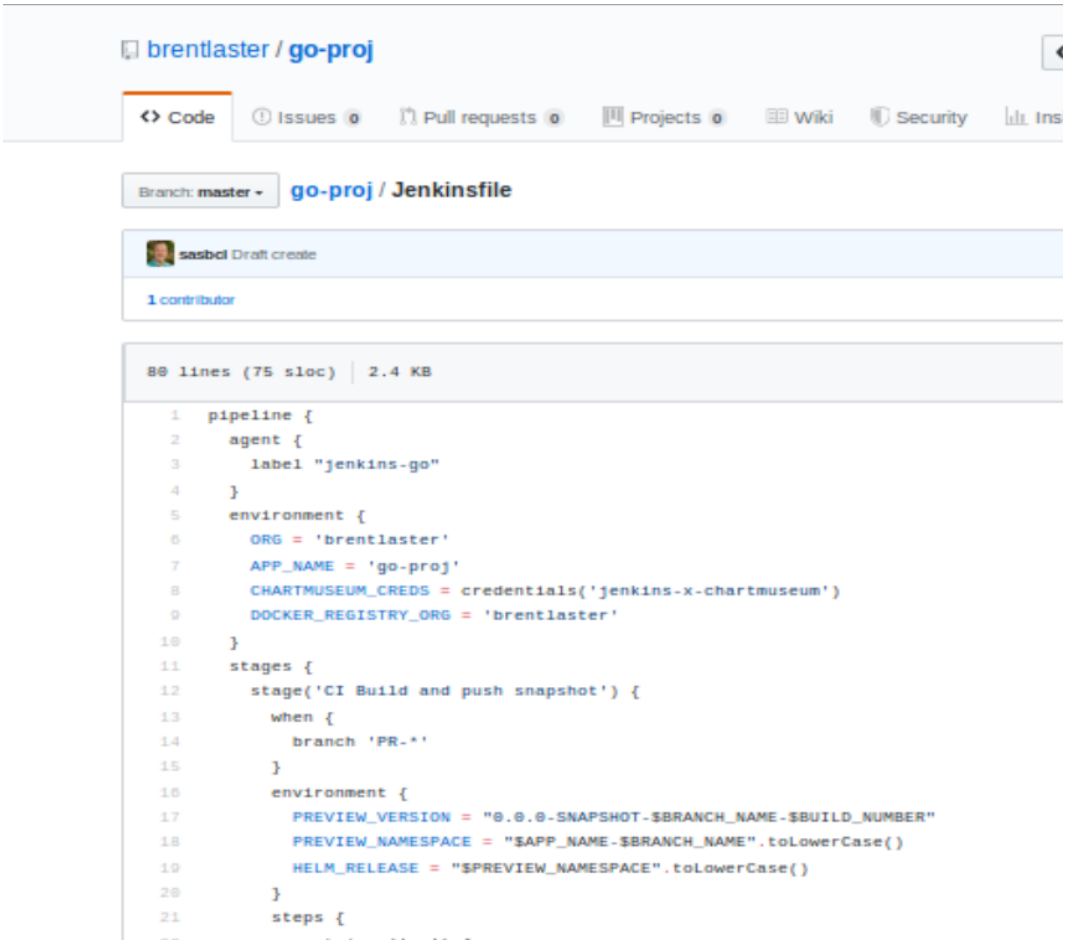
11. Find the link in the output for the GitHub project and then open it up and take a look.



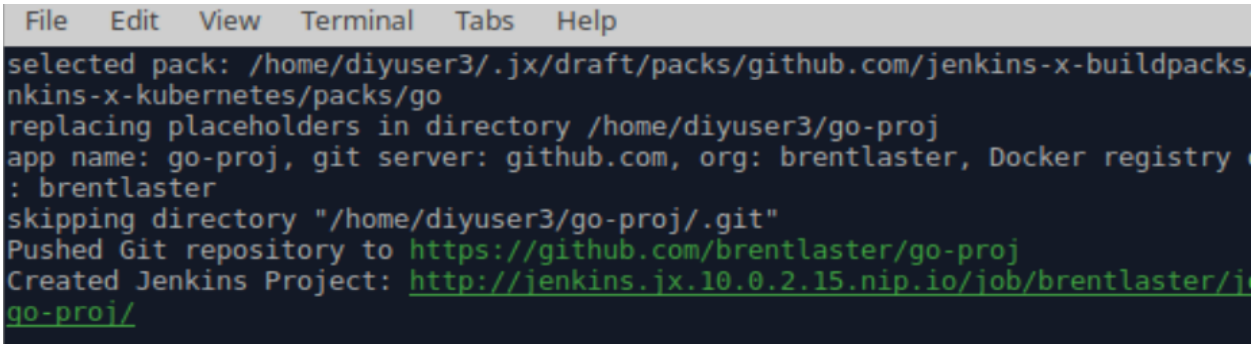
12. Let's take a look at how the project is put together. Go back to the Code section by clicking on the **Code** tab on the far left. Then open up the Dockerfile to see how the container is constructed.



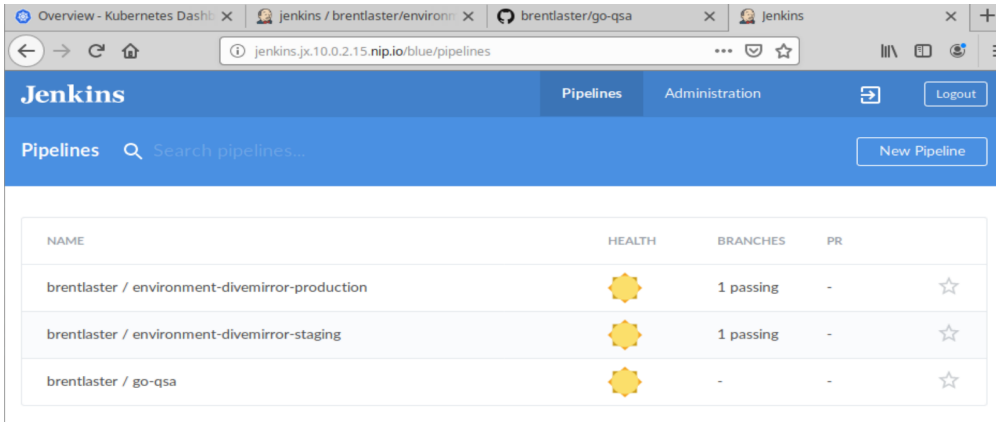
13. Next, open up the “Jenkinsfile” (below the Dockerfile in the Code list) and look at the kind of processing that it’s doing.



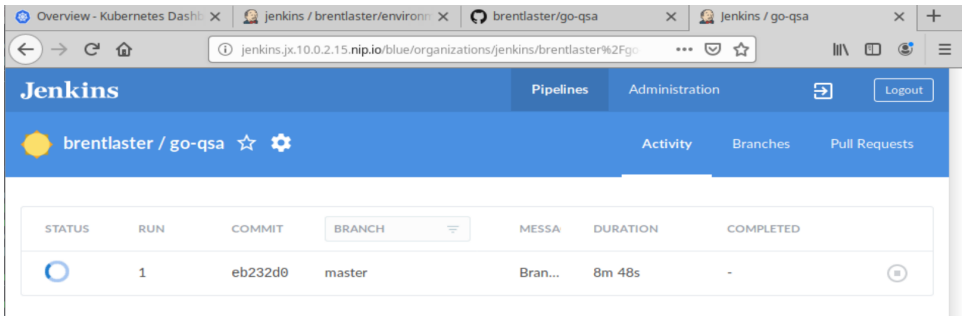
14. Now switch back to the terminal, find the link in the output for the Jenkins project that was created and open it up in Jenkins to look at the project.



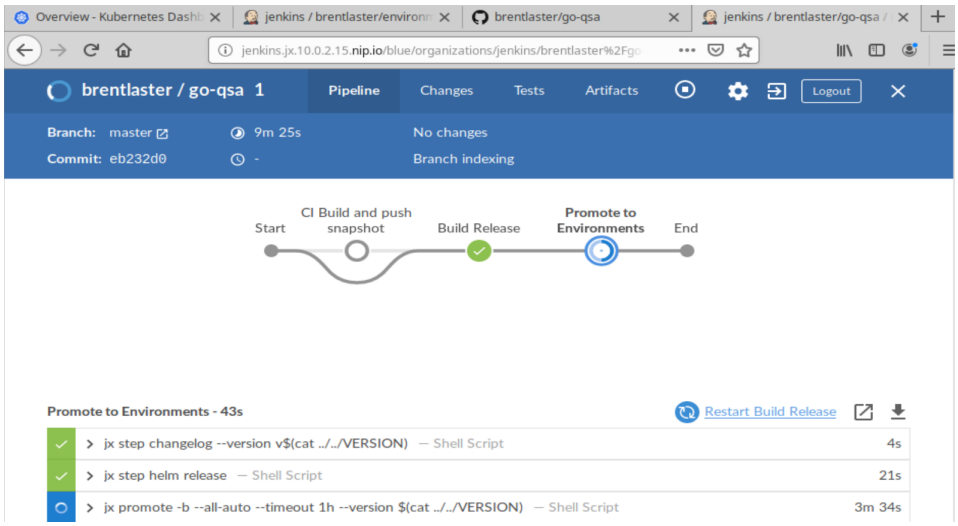
15. Login again with the same userid and password. If you are not seeing all the pipelines on the screen, click on the word "Pipelines" in the blue bar at the top. You should then see something like this.



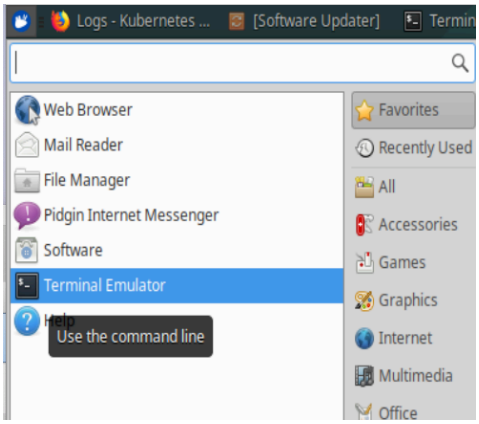
16. From here, click on the row for the name of the quickstart project you created.



17. And then click on the entry for the "master" branch to see the running pipeline. You can expand any of the steps you want to see the output and what was actually done.



18. At the end of the output from the create quickstart command, you should see a list of jx commands you can run to see things about the pipeline. We'll use the watch option to watch the pipeline activity. Open up a third terminal window to use for watching the activity. To do this, click on the mouse head icon in the upper left of the VM window, then select "Terminal Emulator" from the list of apps.



19. In this third terminal window, enter the following command:

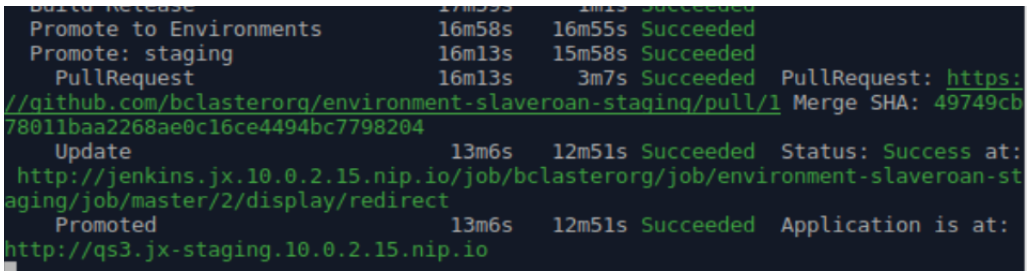
    jx get activity -f <project name> -w

    You can leave this running while class continues.

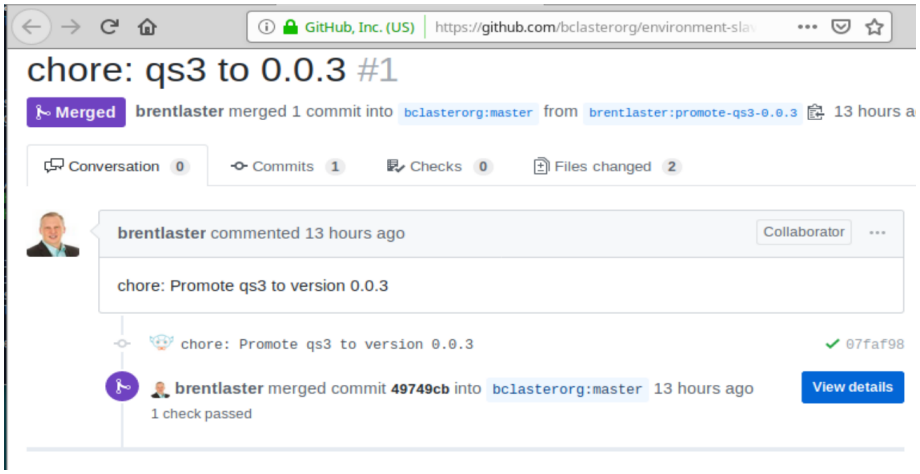
Lab 5: Creating a Preview environment

Purpose: In this lab, we'll see how to make a change to our app, push the changes, create a Pull Request, merge it, and have Jenkins spin up a Preview Environment for us.

1. At this point, your new go quickstart should have gone through the CI/CD pipeline that Jenkins X created and hopefully completed that process successfully. One of the items that was created for this was a Pull Request in your GitHub project. Let's take a look at that. In the terminal window that has the output from the watch (the last command we did in the previous lab), find the line (a few from the bottom) that starts with "PullRequest". Hover over the link in that line that starts with "https", right-click and "Open Link".

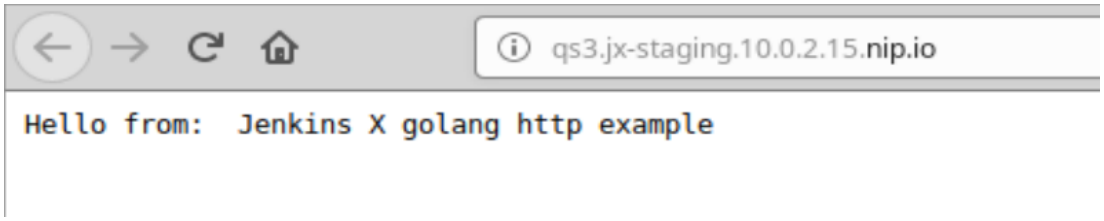


2. Find the “View details” around the middle of the page (below the green checkmark). Click on the button to see what occurred automatically as part of the pipeline processing.



3. This tells us that there was a merge check run for the pull request automatically that passed. The job may no longer be in Jenkins since the Pull Request has been merged successfully (as indicated by the “Merged” status in the upper left).

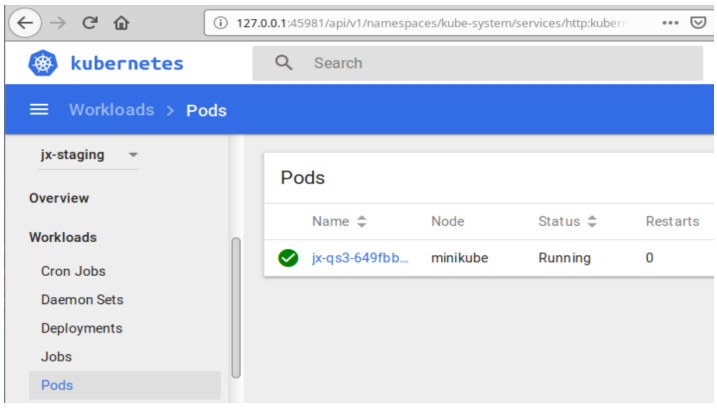
4. Now let’s look at the actual app that got created as it is running. In the terminal session with the watch running, find the line (usually the last one) that starts with “Promoted” and has the phrase "Application is at:" followed by a link to the URL for the staging version of our running application. Open that link and you should see the running instance of your app in the staging environment.



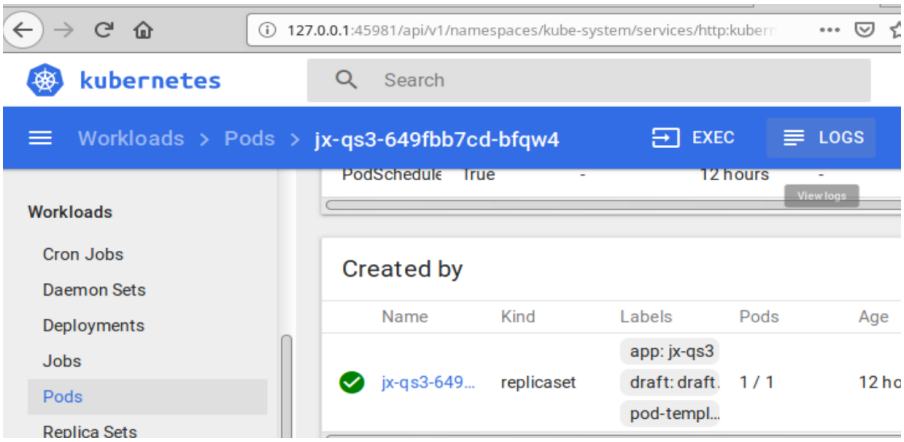
5. This app is running in a container that is being run in a Kubernetes pod in our staging namespace within the cluster. Let’s drill down to find that pod and look at the logs from it. If you no longer have the Kubernetes dashboard up, start it again with:

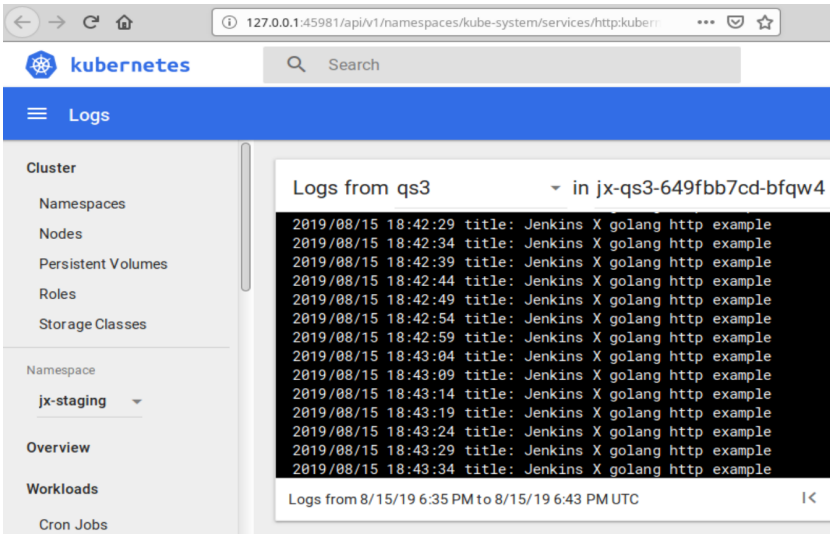
minikube dashboard &

6. In the browser tab for the dashboard, under the line in the middle of the left column, under “Namespace” select **jx-staging**. Then under Workloads, scroll down and select **Pods**.



7. In the Pods section on the right, click on the pod name for your project (will be different than what’s shown). Then, in the blue bar across the top, click on **LOGS**. After that, you should see similar log output from the app running inside the container as you saw in the web page for the app. (Again, names will be different.)





8. Now let's see how Jenkins X can automatically create a Preview Environment for a Pull Request that we create.

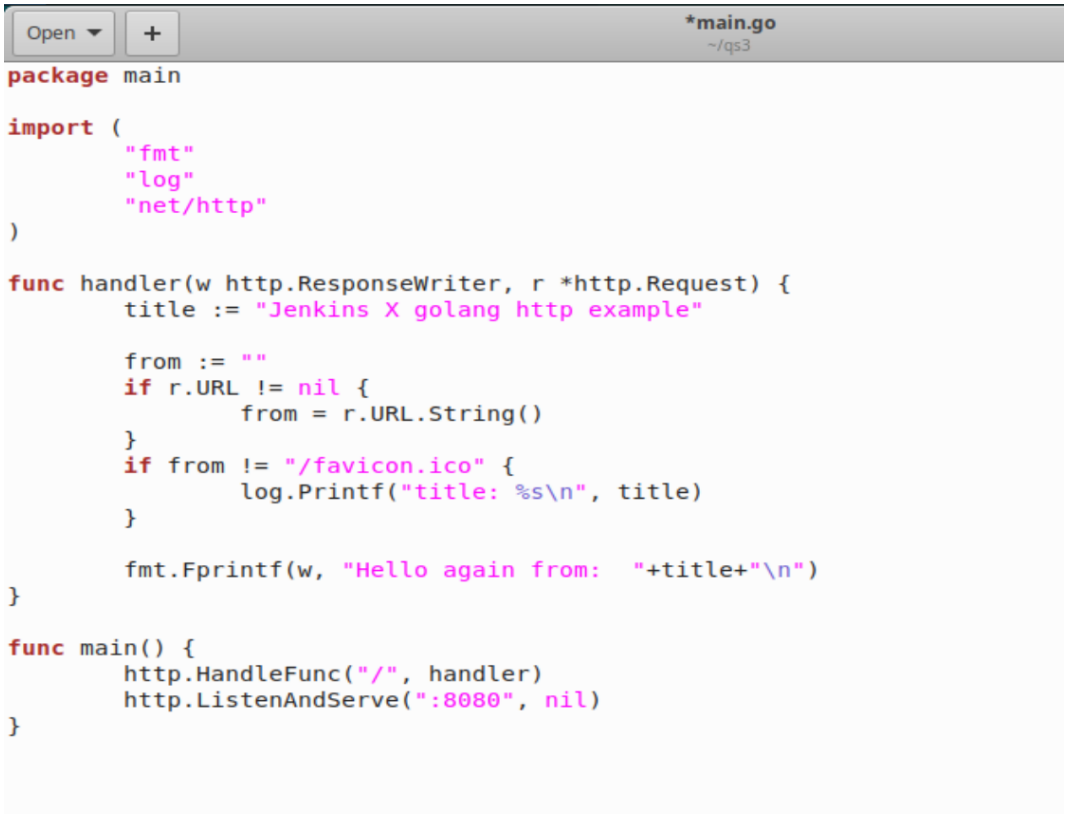
9. Now, in your original terminal window, change into the directory for your project. (Hit Enter if you don't see a prompt.). Then create a new branch and modify the main.go file to add some other words in the message that is displayed in the browser.

   cd ~/<directory for project>

   git checkout -b update

   gedit main.go

   Then, modify the Fprintf statement in the editor as shown below (6th line from the bottom, change "Hello again" to be "Hello again from" - or any text you want). Ignore any errors in the terminal from gedit.



10. Save your changes and exit the editor. Then commit and push your change into GitHub.

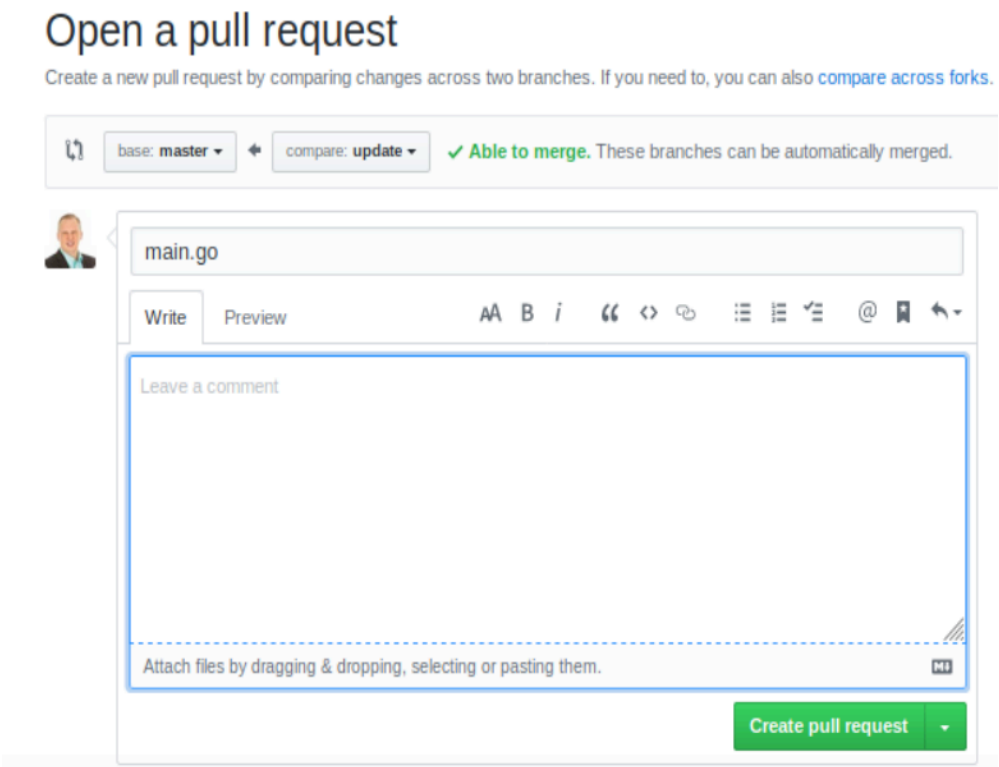
    git commit -am main.go

    git push origin update:update

11. When this completes, you should see a message in your output that tells you how to create a pull request. Go ahead and open the link that's provided for this.

```
diyuser3@training1:~/go-proj7$ git push origin update:update
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 293 bytes | 293.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'update' on GitHub by visiting:
remote:      https://github.com/brentlaster/go-proj7/pull/new/update
remote:
To https://github.com/brentlaster/go-proj7.git
 * [new branch]      update -> update
```

12. If you need to, login to your GitHub account. Then you'll be at the screen to create the pull request. Enter some text if you want in the "Update" box if you want and then click the green button to "Create pull request".



13. After you do this, Jenkins X should automatically create a preview environment for you. In the window where you have the watch activity running, you should see output shortly indicating that the preview environment was created and other activities around it are starting to take place.

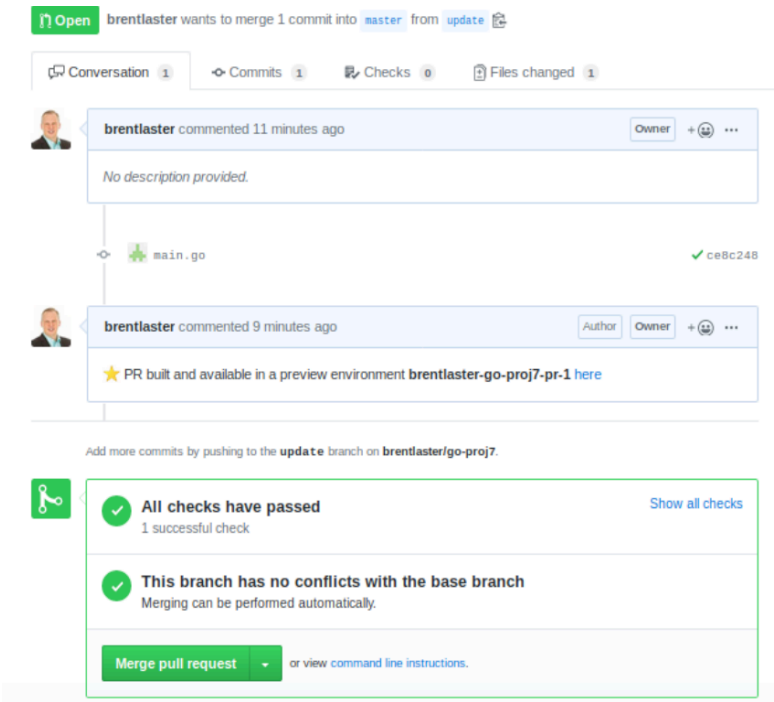
```
  CI Build and push snapshot          51s          Pending
  Preview                              0s           https://github.com/brentlas
ter/go-proj7/pull/1
    Preview Application                0s           http://go-proj7.jx-brentlas
ter-go-proj7-pr-1.10.0.2.15.nip.io
brentlaster/go-proj7/PR-1 #1         1m20s         Running
  Checkout Source                     57s       4s Succeeded
  CI Build and push snapshot          53s      52s Succeeded
  Preview                              2s           https://github.com/brentlas
ter/go-proj7/pull/1
    Preview Application                2s           http://go-proj7.jx-brentlas
ter-go-proj7-pr-1.10.0.2.15.nip.io
```

14. In the activity watch window, wait until you see the line about "Preview Application" (there may be others under it) . Then, in the original terminal window, run the jx get preview command to see the preview environment that was setup for this and where the application running in the preview environment can be accessed.

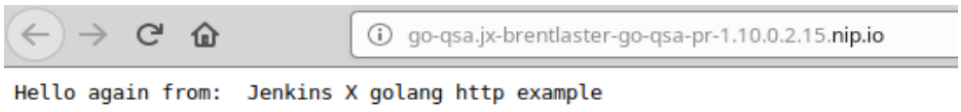
jx get preview

```
diyuser3@training1:~$ jx get preview
PULL REQUEST                                    NAMESPACE
APPLICATION
https://github.com/brentlaster/go-proj7/pull/1 jx-brentlaster-go-proj7-pr-1
http://go-proj7.jx-brentlaster-go-proj7-pr-1.10.0.2.15.nip.io
```

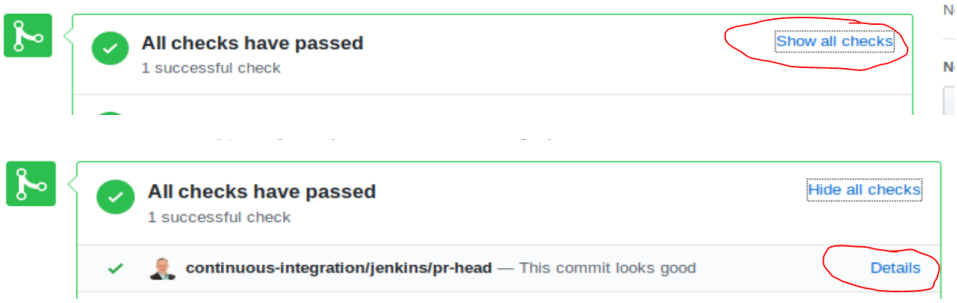
15. Open the link under PULL REQUEST (the first one in the line) to open up the GitHub page for the Pull Request. You should see a screen that indicates the PR was built and is available in a preview environment.



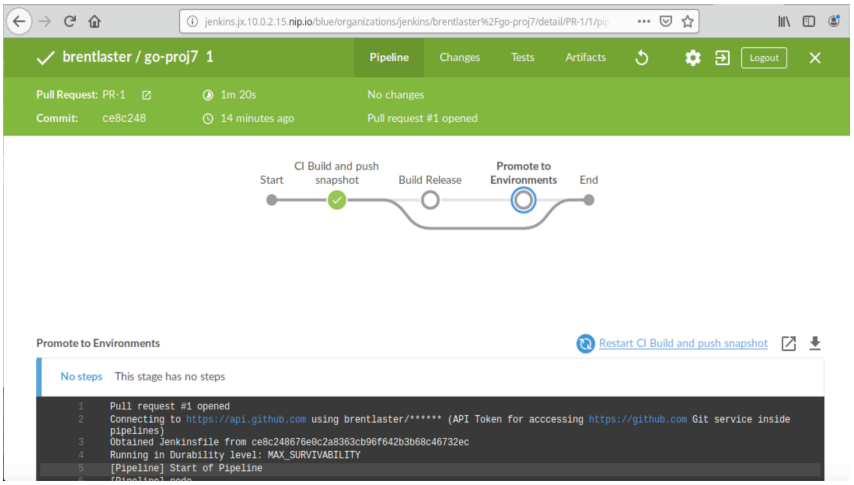
16. Right-click on the link that says “here” at the end of the line that has the star at the left and select “Open Link in New Tab”. This will take you to the running instance of your application in the preview environment. (You could have also gotten to it by clicking the second link in the output from the "jx get preview" command.)



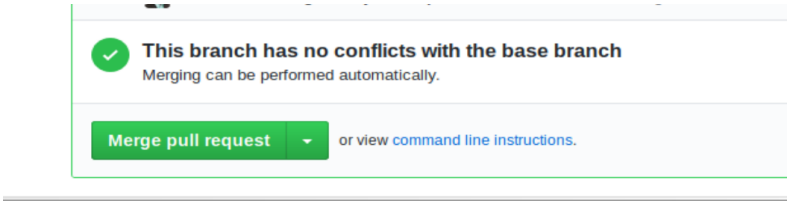
17. Now go back to the previous page for the pull request and click on the “Show all checks” link in the window that says “All checks have passed”. When that expands, click on the “Details” link that comes up and then the “Open Link in New Tab” entry.



18. Log into Jenkins again if needed with user “admin” and password “admin”. You should now see the build for the preview environment.



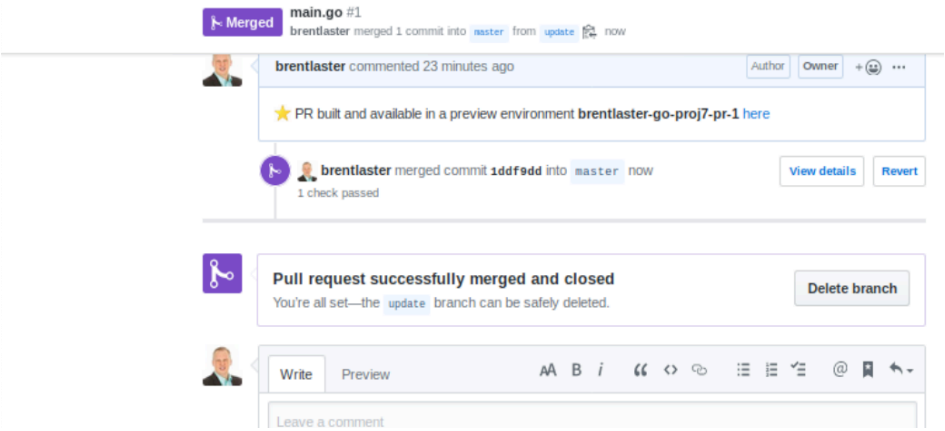
19. In the GitHub page for your PR that we were looking at in the last lab, go ahead and click on the big green button to “Merge pull request” and then confirm the merge.



Lab 6: Promoting to Prod

In this lab, we'll take the version of our application that is in the Preview Environment and promote it to production.

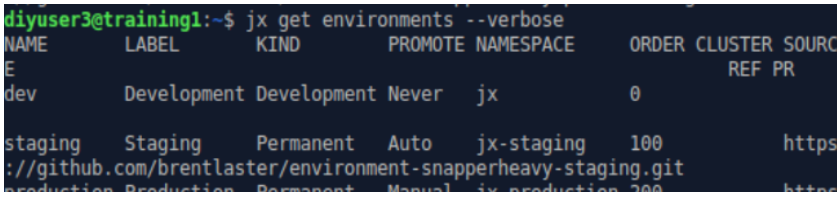
1. Looking at your pull request in GitHub, you should now see that it has been successfully merged and closed.



2. At this point, the app should have been promoted and built in staging. In the activity watch terminal, look for the line that starts with "Promoted" and shows "Succeeded" to know it has gotten this far. (If not there yet, wait for it.) In the original window, take a look at the environments you have again with the command below.

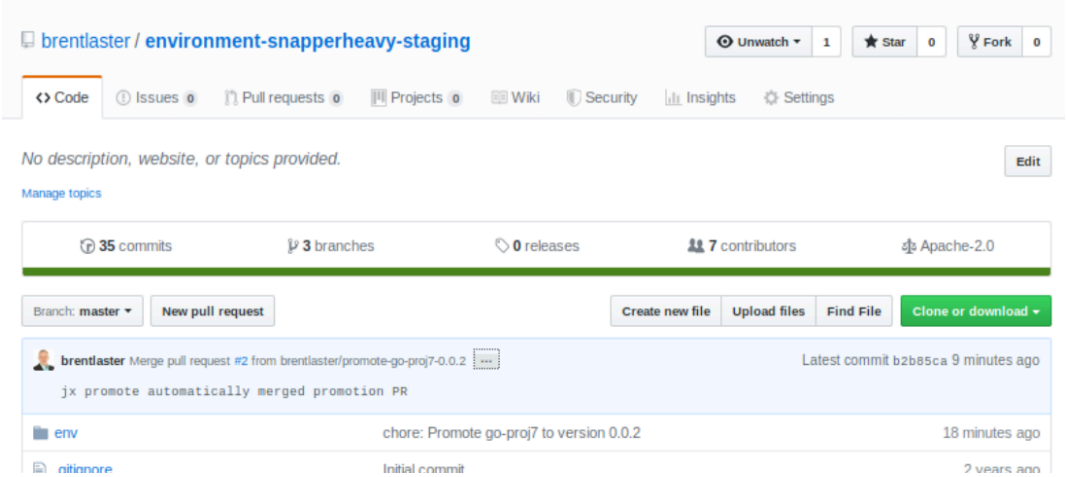
   jx get environments --verbose

3. Look at the PROMOTE column. Notice that the Staging environment is set to "Auto" and the Production environments is set to "Manual". Open the output link for the GitHub staging area (the link in the line that starts with "staging") and look at the changes that happened in there.



4. In the line in the blue box that starts with "<github userid> Merge pull request #1…" notice the version number at the end. "-0.0.2". Where did that version number come from?

   Click on the three dots. "…" after the text on that line. Notice the message about "jx promote automatically merged promotion PR". What PR is it talking about?



5. Take a look at the current version. In an available terminal window, type:

   jx get version

```
diyuser3@training1:~$ jx get version
APPLICATION STAGING PODS URL
go-proj7    0.0.2   1/1  http://go-proj7.jx-staging.10.0.2.15.nip.io
```

   Take note of the version showing here - you will need it further down.

6. There are other ways to get info about, and see, your application. Try the commands below.

   jx get applications
   jx open <application-name> -e staging

7. As we can see, our app was automatically promoted to staging. But as we saw when we looked at the promotion policies on the environments, we have to manually promote it to production.

8. To do this, in the original window, make sure you are in the app's directory and on the master branch. Then run the promote command to promote your app to production.

   cd ~/<app-name> (if not already there)
   git checkout master

   jx promote --version <version number from step 5> -e production

   (Don't worry about the WARNING: Failed to query here - it will resolve after a bit.)
   You may be asked about using your github userid as the user name to comment on issues. Just accept the default Yes answer.

9. Take a look at the current version.

   jx get version

10. You should be able to see your app in the staging environment by doing

    jx open <app-name> -e production

11. Finally, you can go back to the minikube dashboard and look at the different namespaces, including the one for jx-<github userikd>-<project name>-pr-1 for the preview environments.

THE END - THANKS!