Microsoft Cloud Workshop

OSS DevOps

Hands-on lab step-by-step

February 2018

Contents

OSS DevOps hands-on lab step-by-step 1

Abstract and learning objectives 1

Overview 1

Solution architecture 1

Requirements 1

Help References 2

Before the hands-on lab 3

Task 1: Install prerequisites 3

Exercise 1: Configure an Azure Web App 4

Task 1: Create the Azure Web App 4

Task 2: Create an Azure Storage Account 6

Task 3: Configure FTP Deployment Credentials 9

Task 4: Update and review the Web App Settings 10

Task 5: Configure a staging slot 12

Summary 15

Exercise 2: Configure local Git repository 16

Task 1: Fork a GitHub repository locally 16

Summary 16

Exercise 3: Configure Git and Jenkins for continuous integration, delivery and deployment 17

Task 1: Deploy a Jenkins Server in Azure 17

Task 2: Post-Deployment Configuration of Jenkins Server 20

Task 3: Configure Jenkins staging deployment 26

Task 4: Configure your GitHub repo to notify Jenkins of changes 34

Task 5: Check in a change to trigger Jenkins job 36

Task 6: Update Jenkins Project to account for Dropbox content 40

Task 7: Manually Deploy to Production 48

Summary 50

After the hands-on lab 51

Task 1: Delete Resources 51

OSS DevOps hands-on lab step-by-step

Abstract and learning objectives

Migrate an online health food supplier from a hosted environment to Azure and fully embrace modern DevOps tools, investigate PaaS Services and leverage their deep knowledge of Eclipse development tools and its integration with Azure.

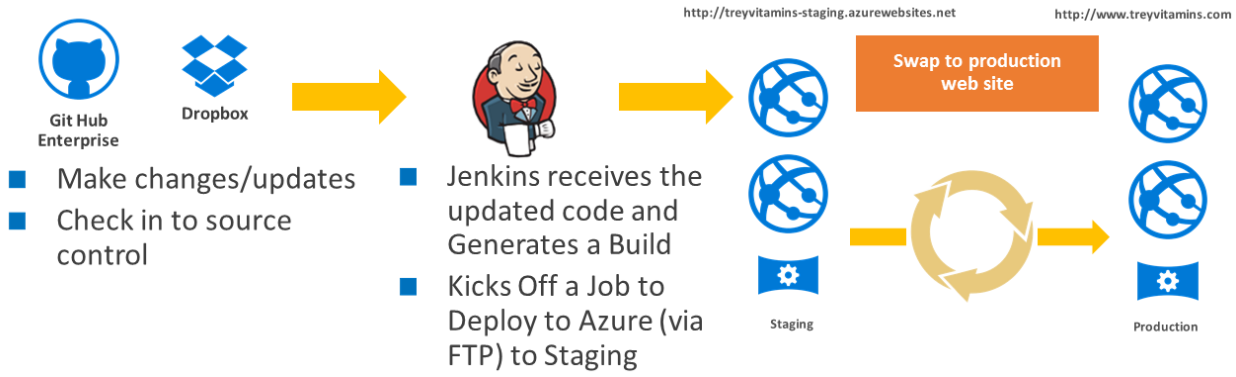
Attendees will be better able to deploy complex OSS workloads into Azure PaaS using Azure App Services as well as the following tasks:

- Deployment and integration of Azure Marketplace products from Partners such as Bitnami for Jenkins
- Update Azure App Services using providers such as GitHub and Dropbox

Overview

The scenario will challenge you to setup continuous integration and delivery of an application using open source tools such as Jenkins and GitHub along with automated deployments to Azure App Services. You will learn about continuous deployment and the benefits of staged publishing.

Solution architecture



Requirements

1. An Azure Subscription
2. A Dropbox account
3. A GitHub account

Help References

Description	Links
Jenkins Documentation	https://jenkins.io/doc/
GitHub Documentation	https://help.github.com/
Dropbox Documentation	https://www.dropbox.com/developers/documentation
Azure Web Apps Documentation	https://azure.microsoft.com/en-us/services/app-service/web/

Before the hands-on lab

Duration: 30 minutes

In this exercise, you will setup everything you need to complete this lab. Note: This lab can be completed on Windows, Mac, or Linux.

Task 1: Install prerequisites

1. On a virtual machine in Azure (Standard_DS2_V2 is recommended) or your own computer install the following:
    a. Visual Studio Code - https://code.visualstudio.com/docs/setup/setup-overview
    b. Install the GitHub client - http://desktop.github.com
    c. Install the DropBox client - https://www.dropbox.com/install
    d. If using Windows, disable IE enhanced security mode
        i. https://blogs.technet.microsoft.com/chenley/2011/03/10/how-to-turn-off-internet-explorer-enhanced-security-configuration/
    e. SSH Clients for Windows
        i. Linux Subsystem for Windows - https://msdn.microsoft.com/en-us/commandline/wsl/install_guide
        ii. PuTTY - http://www.putty.org/

You should follow all steps provided *before* attending the hands-on lab.

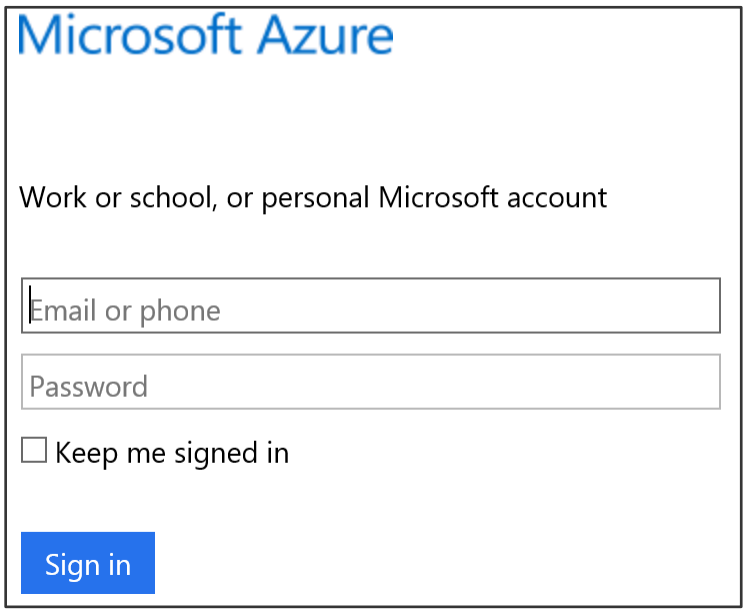
Exercise 1: Configure an Azure Web App

Duration: 30 minutes

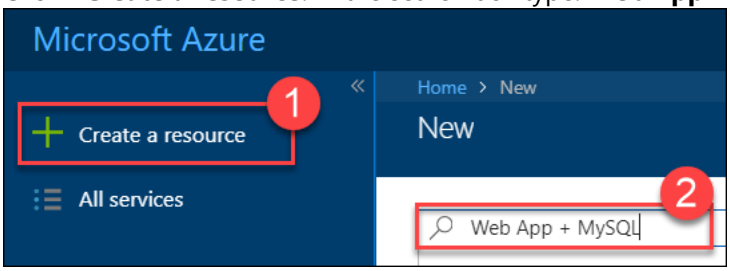
You will use several Azure Platform as a Service (PaaS) components to configure a Web Application. This Web App will use Azure Storage as well as MySQL for data. The actual deployment of the Web Application will happen in the next exercises.

Task 1: Create the Azure Web App

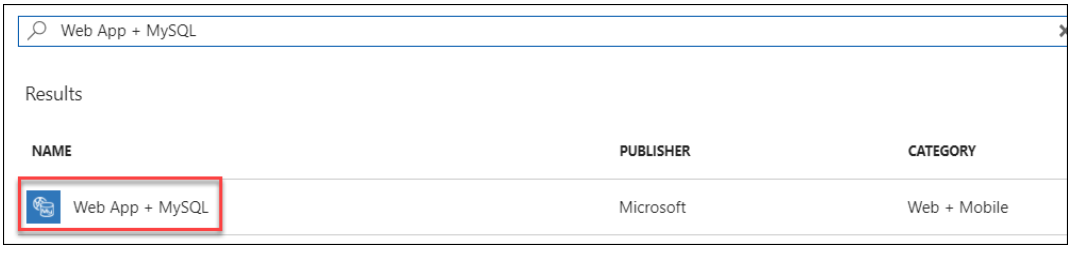
1. If you are leveraging a virtual machine in Azure as your lab machine, first connect to it via RDP. Otherwise, browse to https://portal.azure.com, and authenticate with your Organization or Microsoft Account.



2. Click **+Create a resource**. In the search box type: **Web App + MySQL**, and press **Enter**

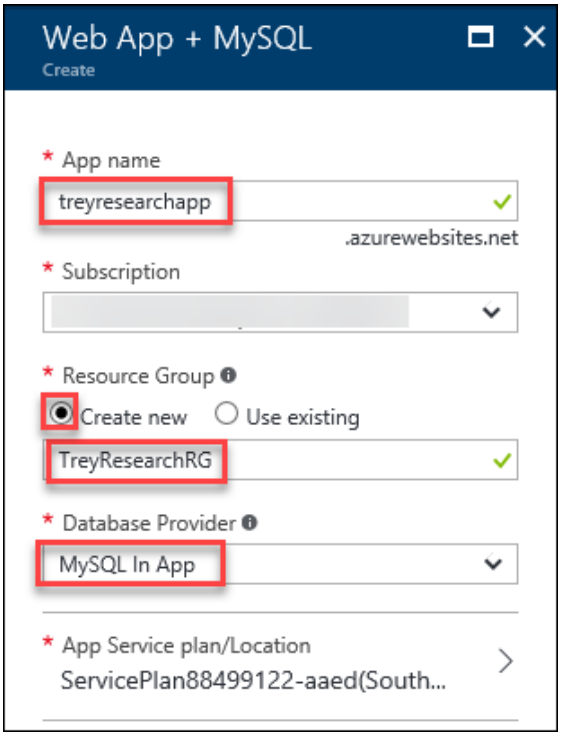


3. Select the **Web App + MySQL** result, and then click **Create** on the next blade

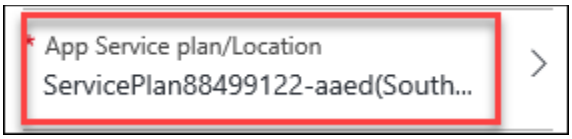


4. Specify the following on the **Web App + MySQL** blade:

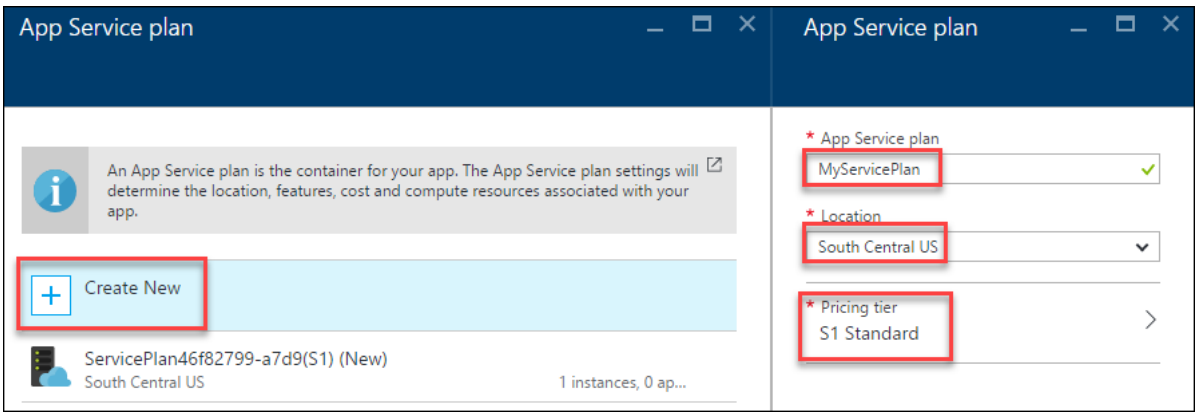
    - App name: **<unique name *ex: treyresearchapp*>**
    - Resource group: **TreyResearchRG**
    - Database Provider: **MySQL In App**



5. Click the **App Service plan/Location** tile

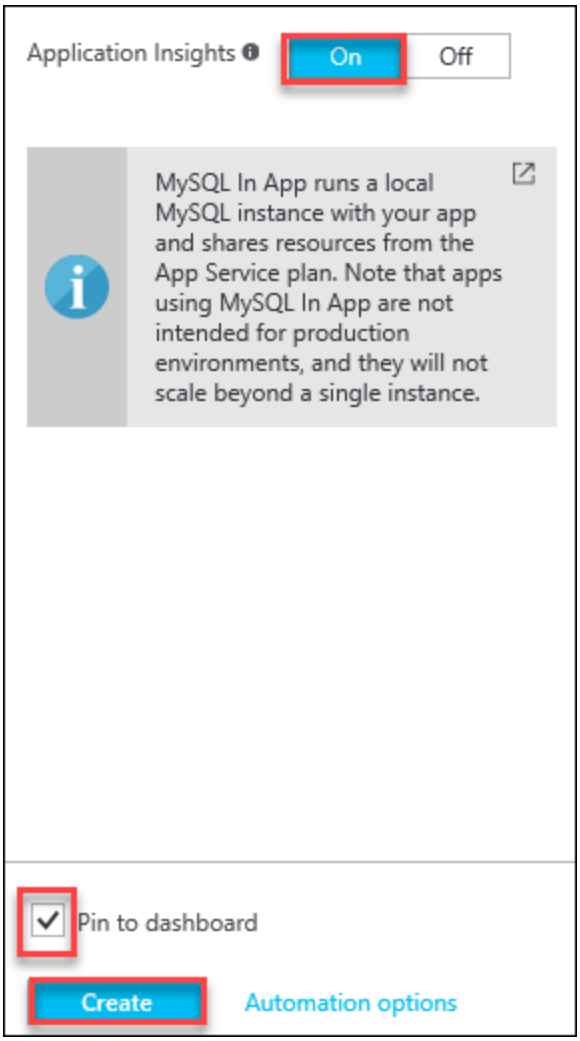


6. Click **+Create New**, and specify the App Service Plan as **MyServicePlan**, choose the region closest to you, and click **OK**



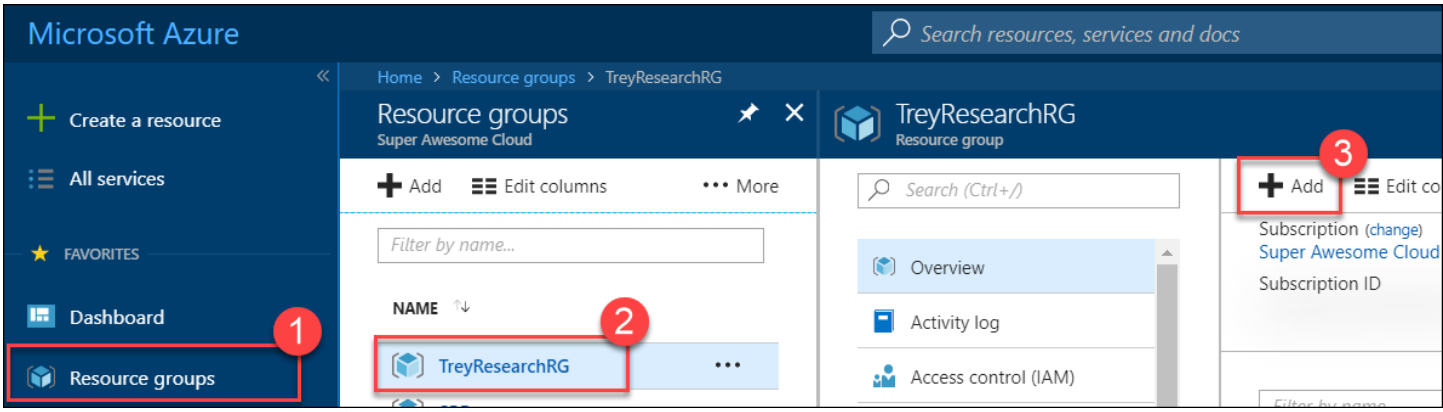
Note: If you receive an error the RG failed to deploy because there is no credit card found associated with the account, choose the FREE option in the Pricing Tier.

7. Set App Insights to **On,** check the **Pin to dashboard** checkbox, and click **Create**

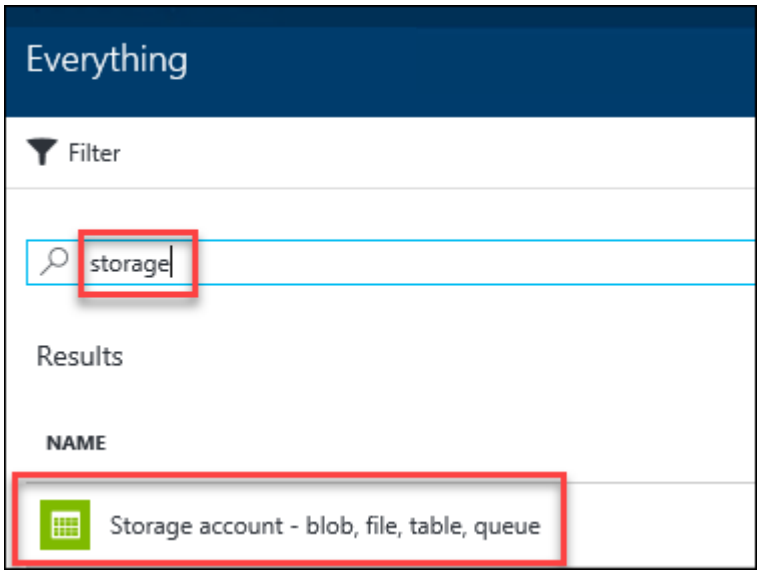


Task 2: Create an Azure Storage Account

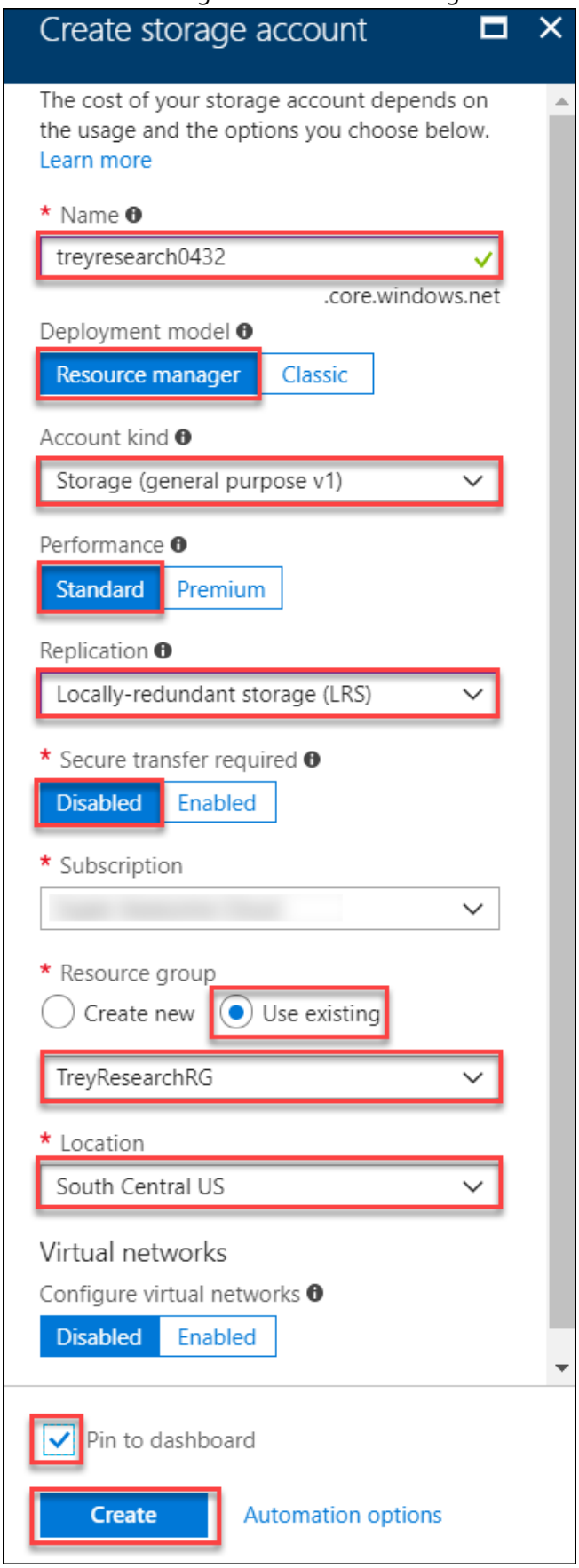
1. Click **Resource groups > TreyResearchRG > +Add**



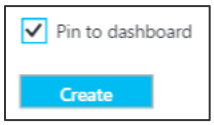
2. Type **storage** in the search bar, hit **Enter**, choose **Storage account**, and click **Create**



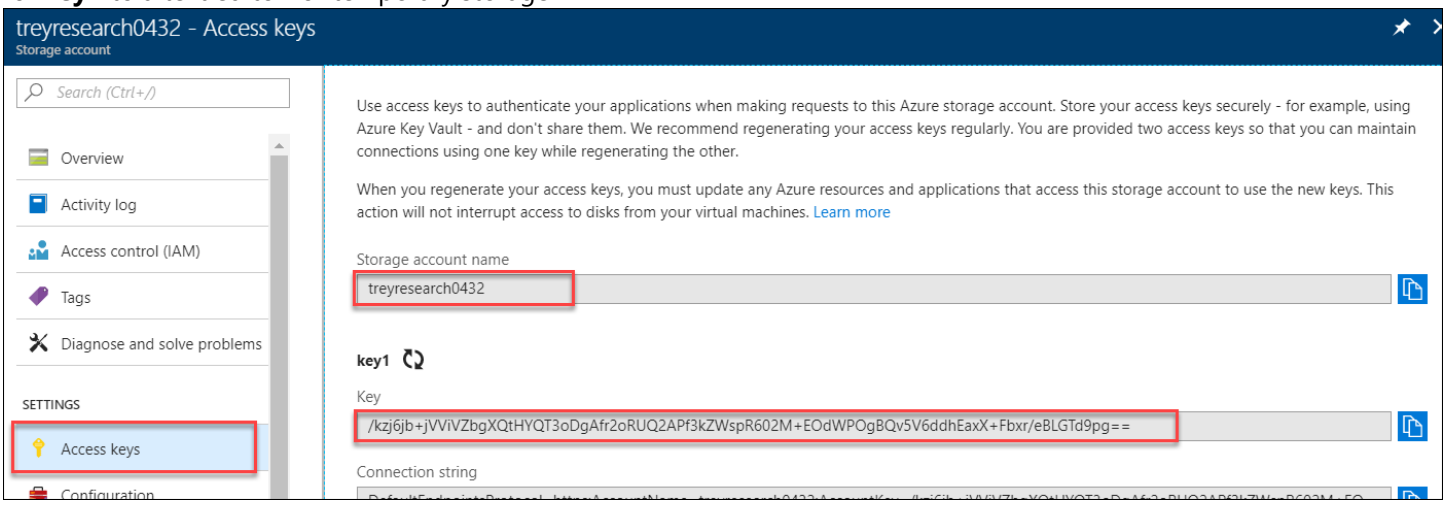
3. Specify a unique name for the storage account. Ensure the green checkmark is displayed



4. Check the check box next to **Pin to dashboard**, and click **Create**



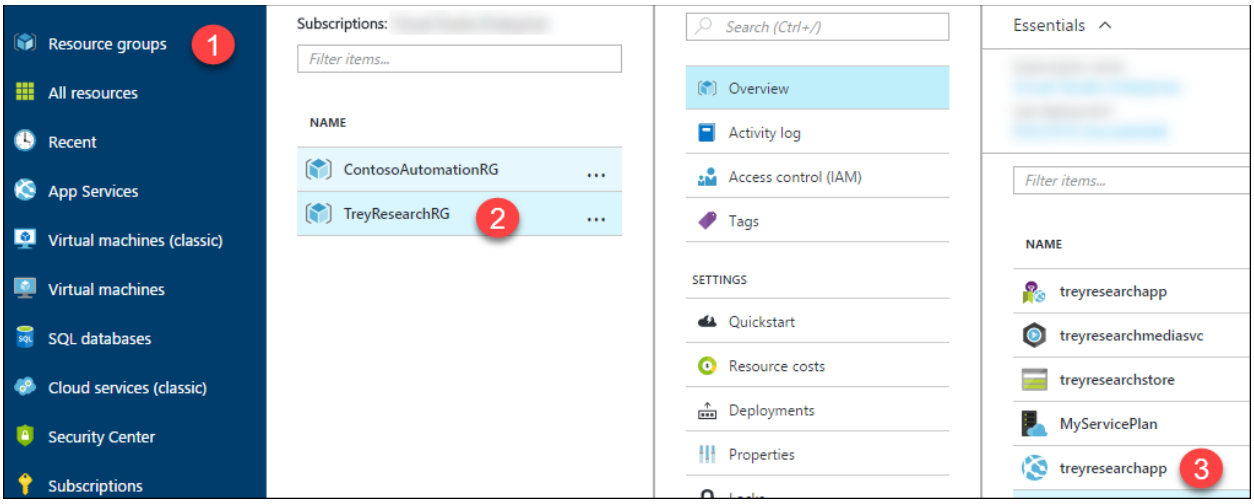
5. After the storage account is provisioned, click **Access keys**, copy the **Storage account name** as well as the value for **key1** to a text editor for temporary storage



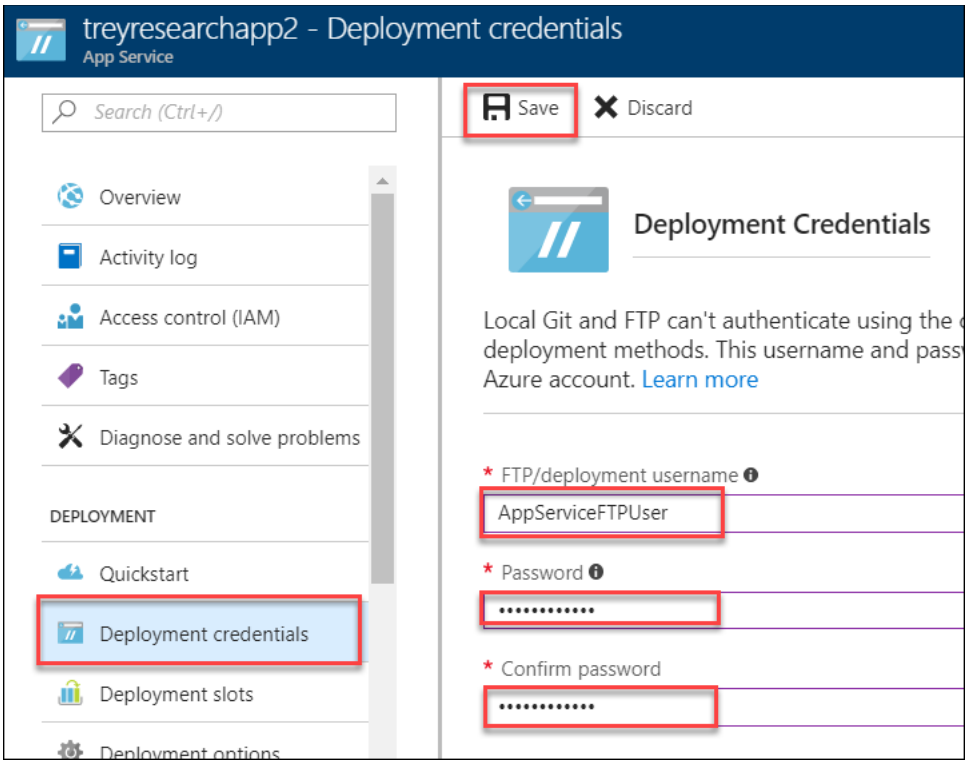
Task 3: Configure FTP Deployment Credentials

Since you will be leveraging Jenkins to deploy the source code, you must first update the credentials that are used for an FTP deployment.

1. In the Azure portal, click **Resource groups > TreyResearchRG > treresearchapp** to open the settings of the App Service



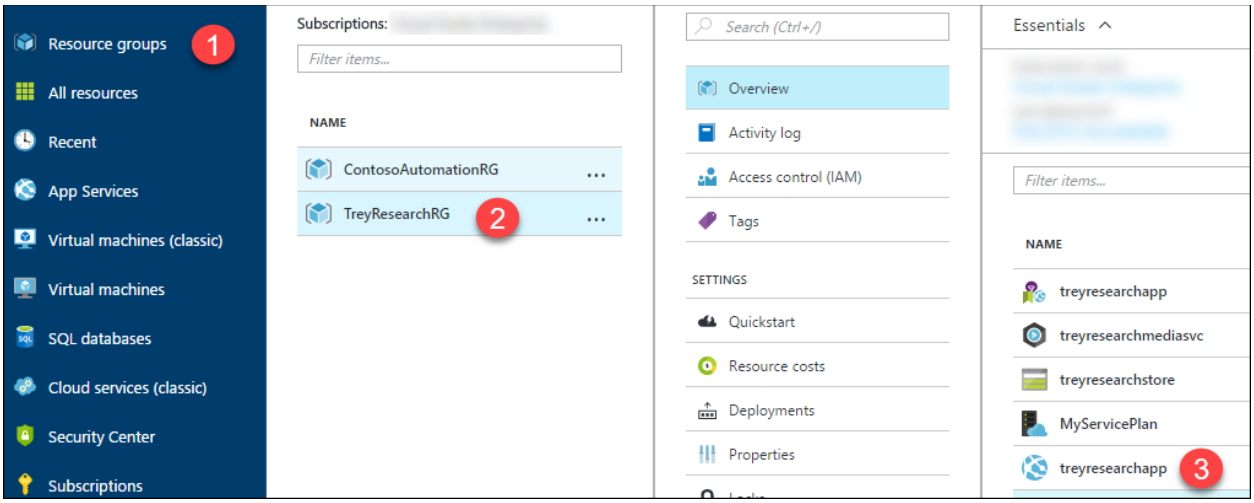
2. Click Deployment credentials, specify a username (**AppServiceFTPUser**) / password (**Demo@pass123**), and click **Save**



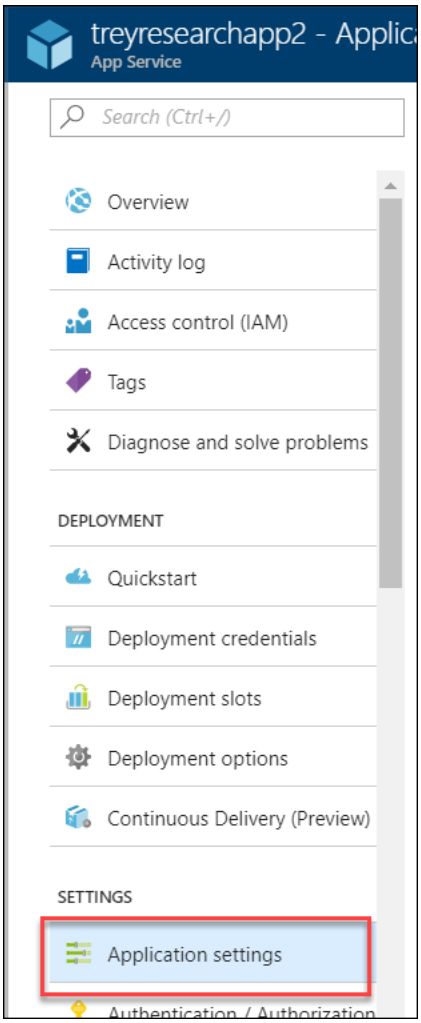
Note: The username must be globally unique and follow the rules on the blade.

Task 4: Update and review the Web App Settings

1. In the Azure portal (https://portal.azure.com), click **Resource groups > TreyResearchRG > treyresearchapp**

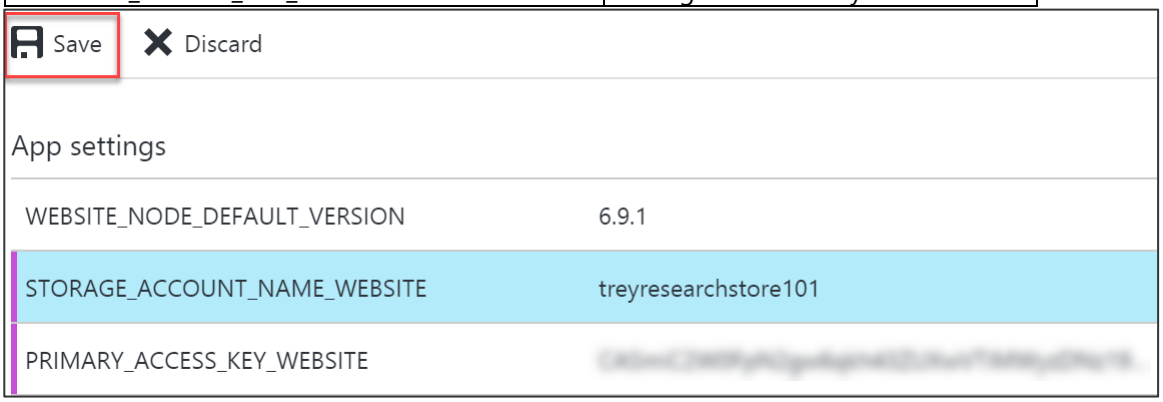


2. Click the **Application Settings** link



3. Within App settings, add 2 new entries using the app settings names below along with the storage account name

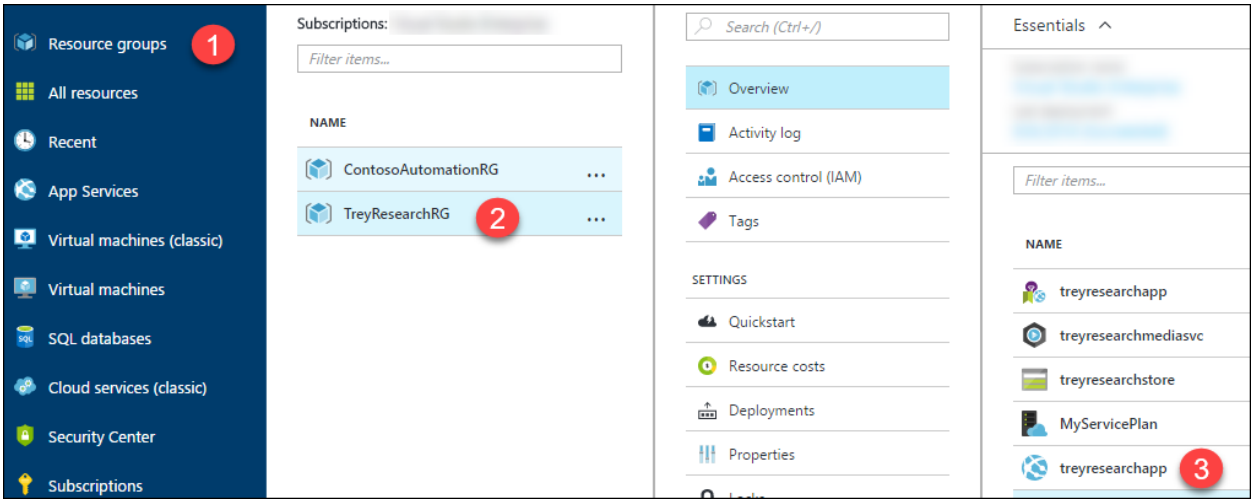
STORAGE_ACCOUNT_NAME_WEBSITE	Storage Account Name
PRIMARY_ACCESS_KEY_WEBSITE	Storage Account Key



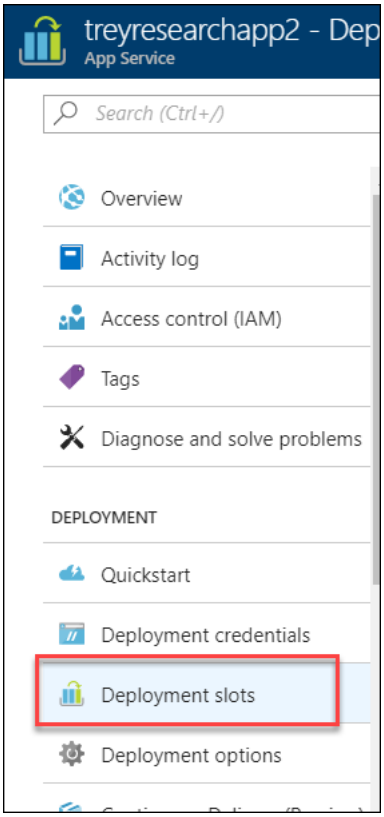
4. Click **Save** on the toolbar

Task 5: Configure a staging slot

1. In the Azure portal, click **Resource groups > TreyResearchRG > treresearchapp** to open the settings of the App Service



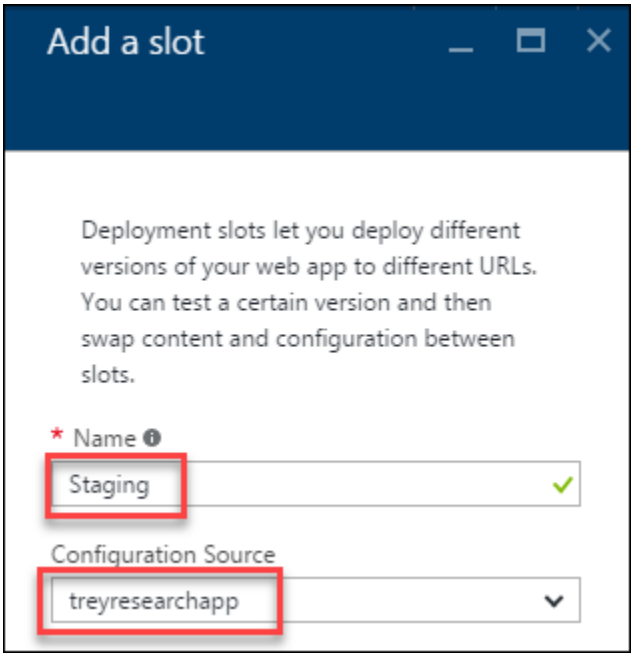
2. Click on **Deployment slots** under the **DEPLOYMENT** category



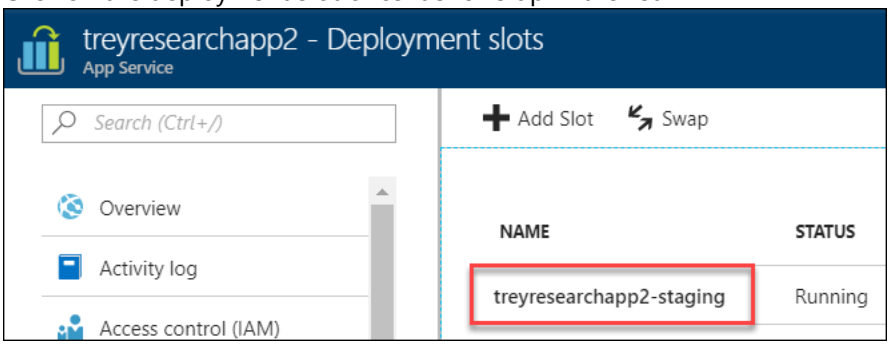
Note: If you deployed with the Free plan, you can now choose the S1 Standard plan, and it will upgrade to allow for Deployment Slots to be created and utilized.

3. Click **Add Slot**

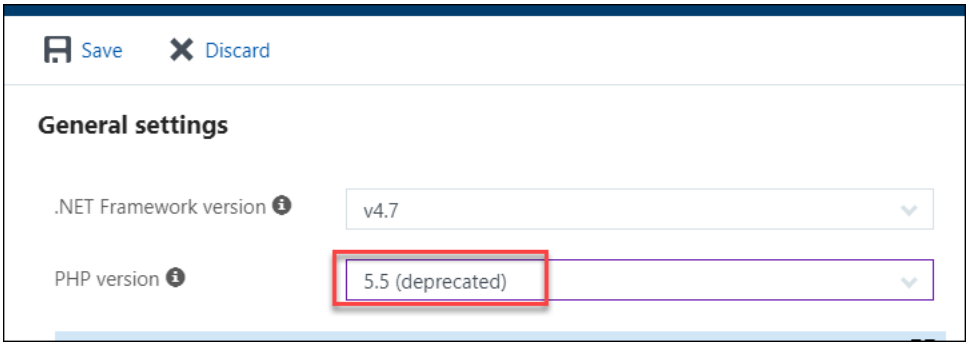
4. Name the slot **Staging**, specify the primary site as the **Configuration Source** (this will copy over the 6 variables and their values we defined previously as well as the connection string), and click **OK** to create the deployment slot



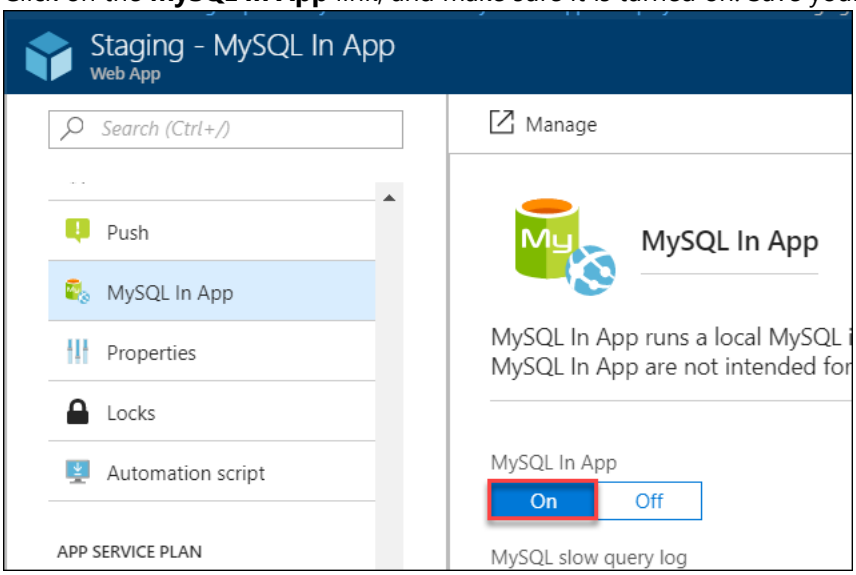
5. Click on the deployment slot once it shows up in the list



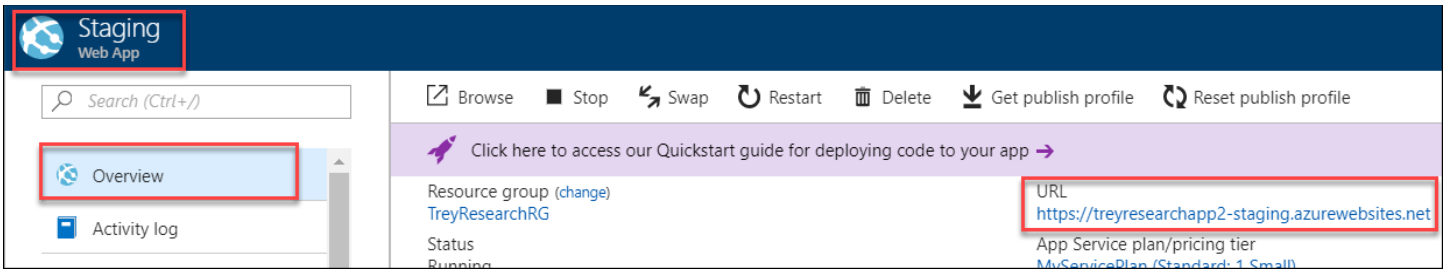
6. Click on the **Application settings** link, verify the same storage settings you created in Task 4, and set the PHP version to 5.5



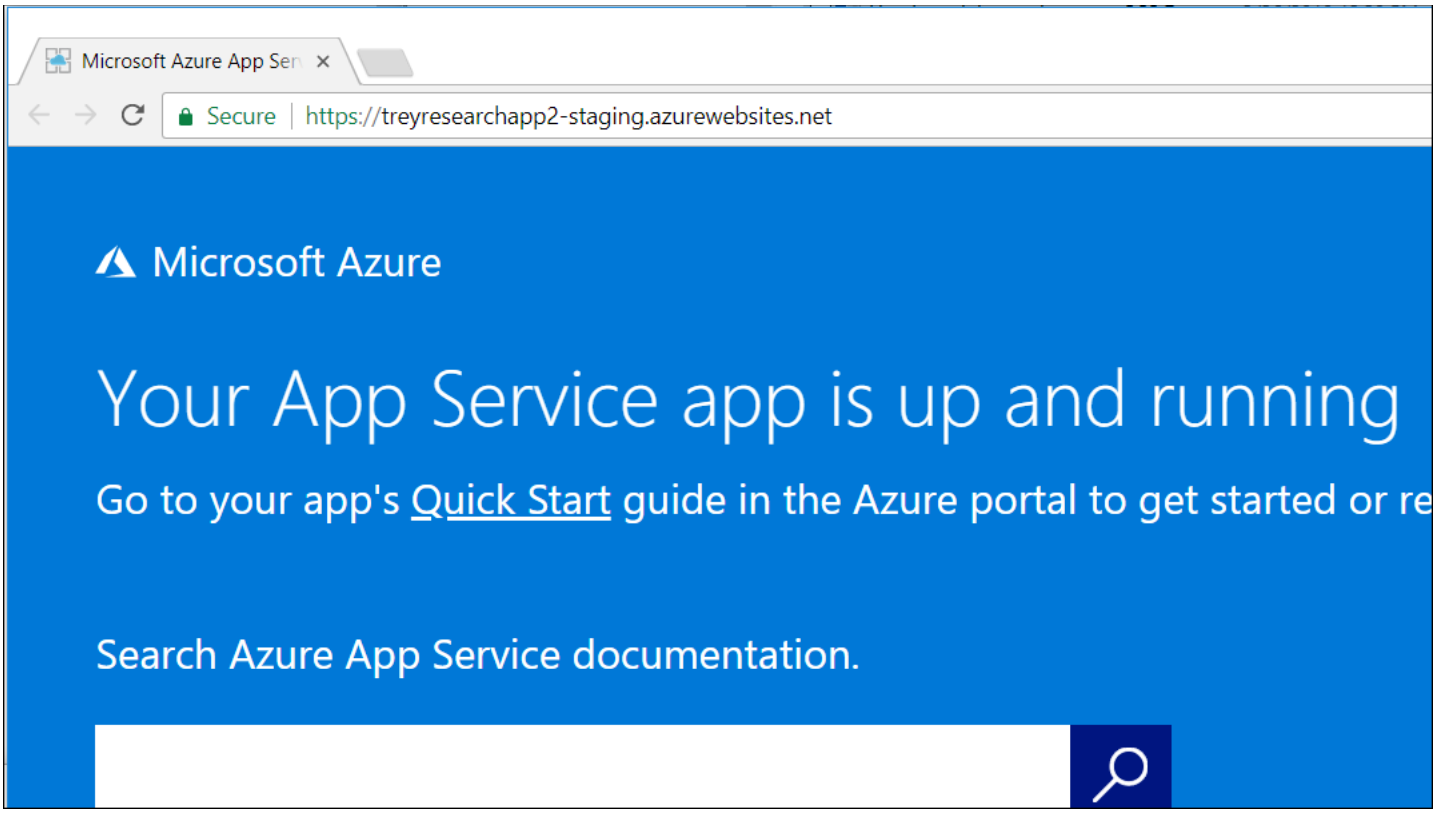
7. Click on the **MySQL In App** link, and make sure it is turned on. Save your changes



8. Once the staging slot has been created, click its name. On the **Overview** link, click on the **URL** from the staging slot's essentials pane



9. At this time, no code has been deployed to either production or the staging slot we just created. Both URLs will have the default website like the one below. You will be leveraging Jenkins and GitHub in the exercises that follow to deploy the website.



Summary

In this exercise, you used several Azure Platform as a Service (PaaS) components to configure a Web Application. The Web App will use Azure Storage as well as MySQL for data. You also configured a staging slot for the Web Application with duplicate settings. The actual deployment of the Web Application will happen in the next exercises.

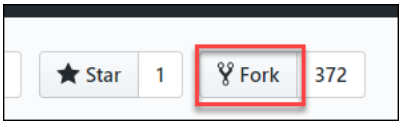
Exercise 2: Configure local Git repository

Duration: 10 minutes

In this exercise, you will fork a GitHub repository and clone it locally so that you can configure your web app.

Task 1: Fork a GitHub repository locally

1. Browse to https://github.com/, and login with your GitHub credentials
2. Navigate to https://github.com/opsgility/php-da-sample, and click the Fork button. This will create a copy of the 'php-da-sample' in a web directory with your GitHub username.



3. After the fork is complete, clone the site locally on your computer for future changes by executing the commands and starting a console/terminal session containing the Git client

```
mkdir repos
cd repos
git clone https://github.com/[YOUR_GITHUB_USERNAME]/php-da-sample
```

Summary

In this exercise, you forked a GitHub repository and cloned it locally, so you can configure your web app.

Exercise 3: Configure Git and Jenkins for continuous integration, delivery and deployment

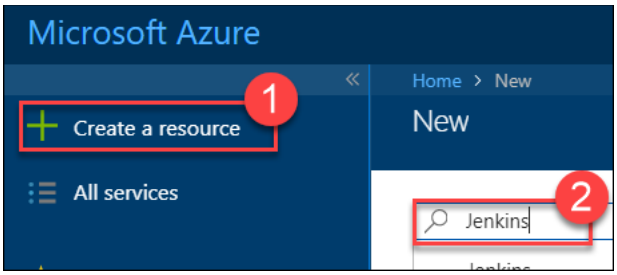
Duration: 90 minutes

In this exercise, you will configure a Jenkins server in Azure and leverage it along with Git to setup continuous integration & delivery of your Web Application. You will be pulling source code from a GitHub repository and configuring Jenkins to build and deploy the code to your Staging slot before it is pushed to production (manually).

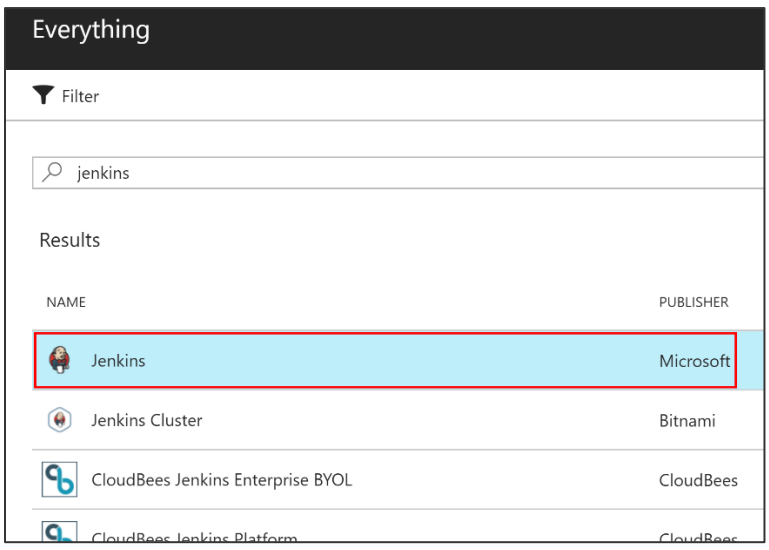
Task 1: Deploy a Jenkins Server in Azure

Jenkins is an open source continuous integration tool written in Java. It provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. In this exercise, you will deploy a Jenkins Server in Azure leveraging a prebuilt virtual machine image from the Azure marketplace.

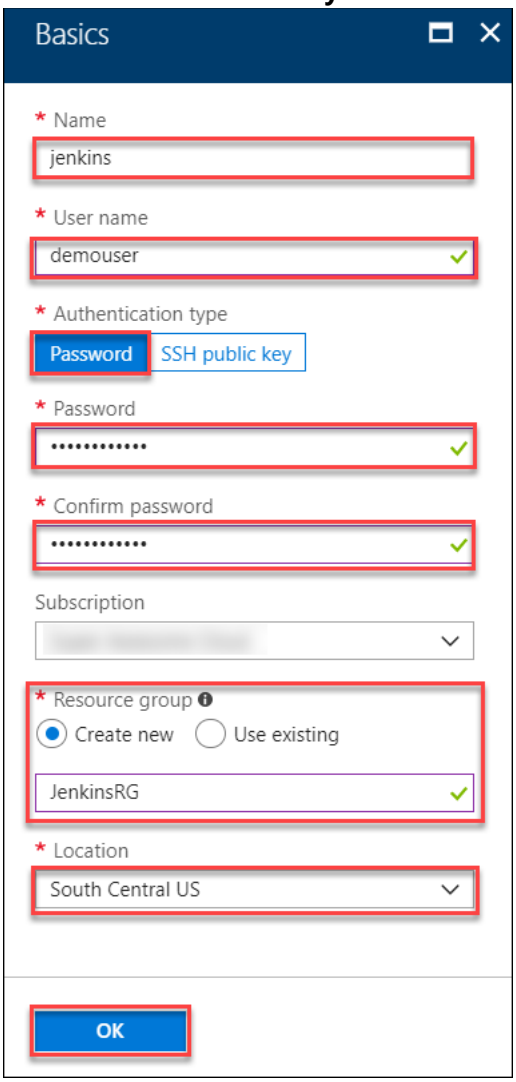
1. If you are leveraging a virtual machine in Azure as your lab machine, first connect to it via RDP Otherwise to deploy a Jenkins Server instance in Azure, browse to https://portal.azure.com. Click **+Create a resource**, type **Jenkins** in the search box, and hit **Enter**.



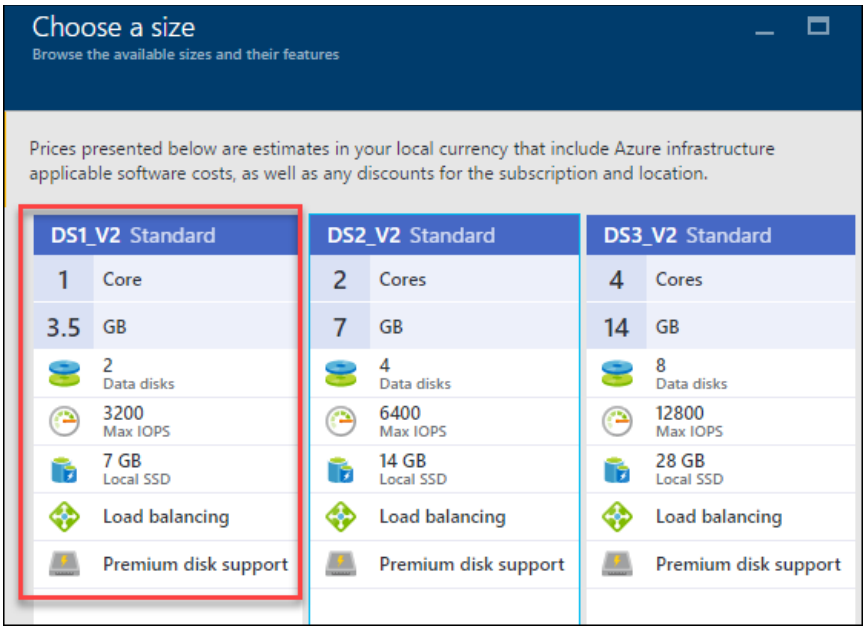
2. You will notice that there are numerous preconfigured Jenkins servers available in the marketplace, for the purposes of the labs, choose the one submitted by **Microsoft** and click **Create**



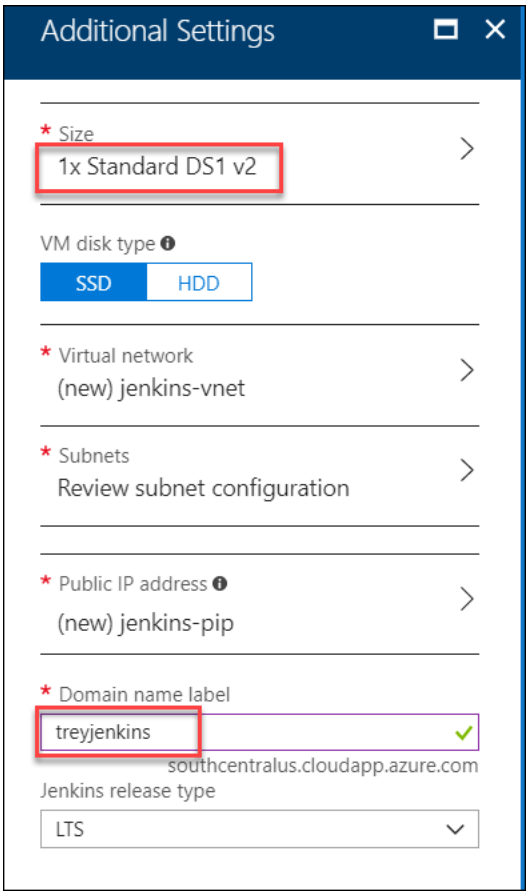
3. Specify the following on the **Basics** blade, and click **OK:**
    a) Name: **jenkins**
    b) User name: **demouser**
    c) Password: **Demo@pass123**
    d) Resource group: **JenkinsRG**
    e) Location: **location nearest you**



4. This will present the **Settings** blade. Click the **Size** option, choose the **DS1_V2 Standard** size for the VM, and click **Select**

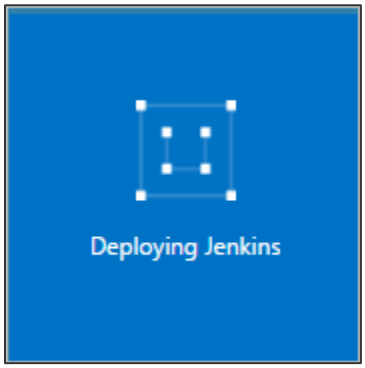


5. Enter a unique name in the Domain name label field, and click **OK** on the **Settings** blade



6. On the **Jenkins Integration Settings** blade, leave the default values, and click **OK**

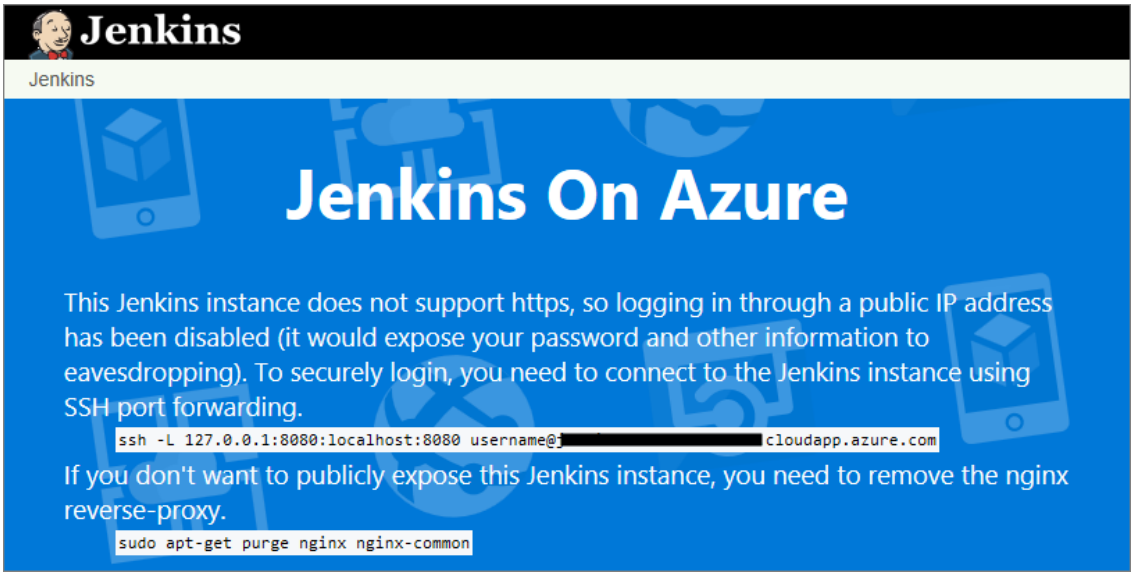
7. Click **OK** on the **Summary** blade, and **Create** on the **Buy** blade to start the provisioning of your Jenkins server



Task 2: Post-Deployment Configuration of Jenkins Server

Navigate to your virtual machine (for example, http://treyjenkins.southcentralus.cloudapp.azure.com/) in your web browser. The Jenkins console is inaccessible through unsecured HTTP so instructions are provided on the page to access the Jenkins console securely from your computer using an SSH tunnel. After that, you will update the OS and Jenkins to the latest as well as install other tools needed for our scenario.

1. Using the FQDN you defined in the previous Exercise, browse to your Jenkins portal. Click the **Log In** link.



2. Set up the tunnel using the `ssh` command on the page from the command line replacing `username` with the name of the virtual machine admin user chosen earlier

```
ssh -L 127.0.0.1:8080:localhost:8080
demouser@treyjenkins.southcentralus.cloudapp.azure.com
```

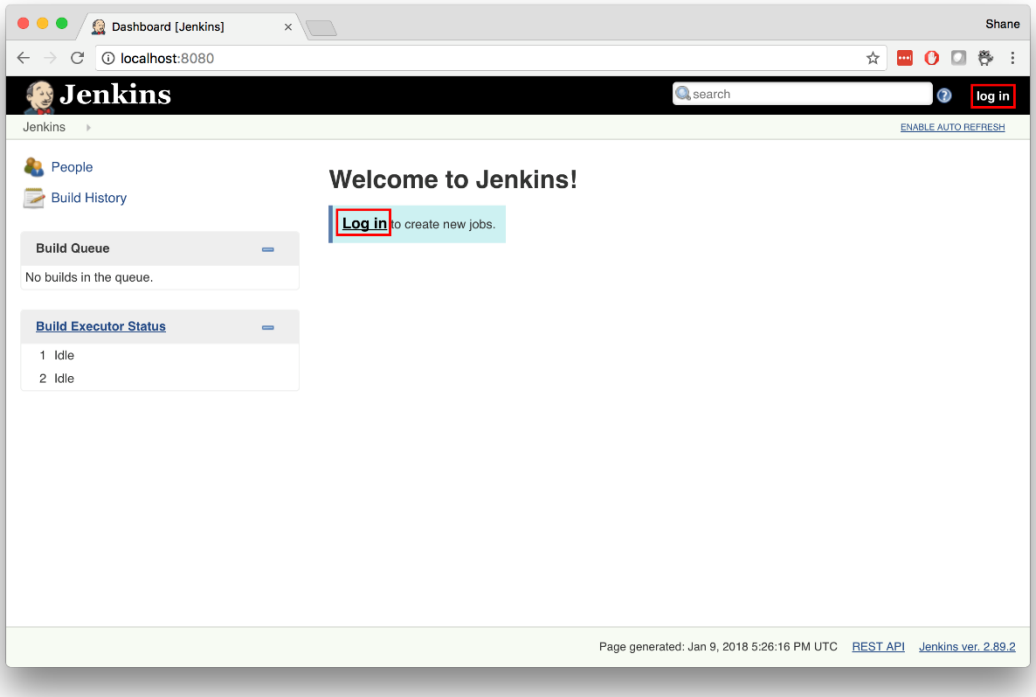
3. When prompted, enter the password you previously chose. Keep the command line window open

4. Get the initial password by running the following command in the command line while still connected through SSH:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

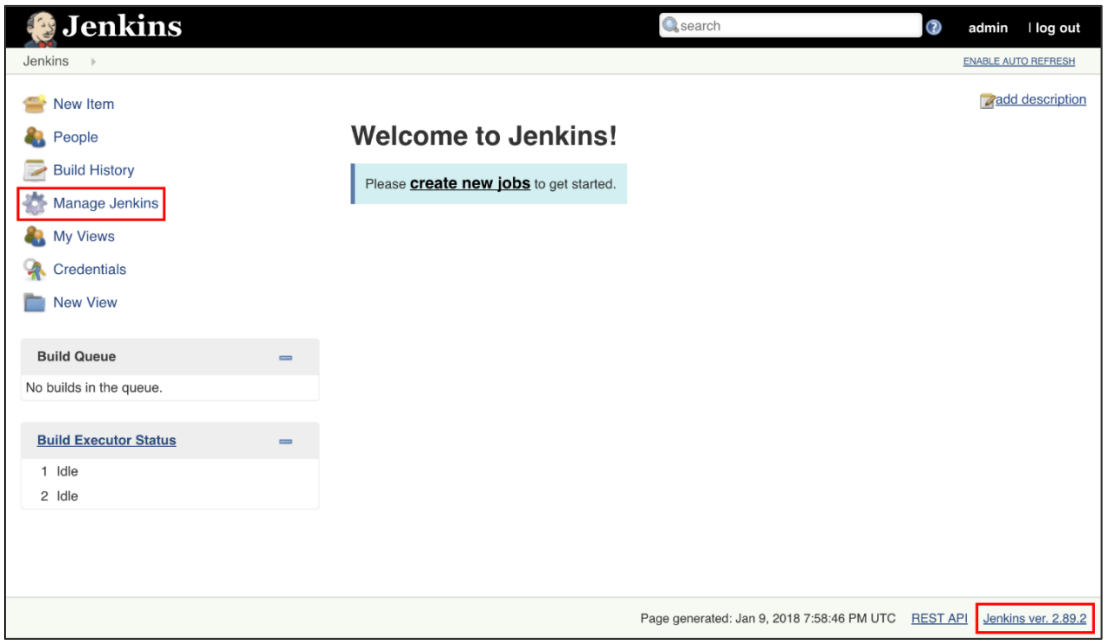
5. Copy the password shown to your clipboard, as you will need it to initially login to the Jenkins interface

6. Open a web browser, and navigate to http://localhost:8080/ on your local machine. Click either of the **Log in** links.



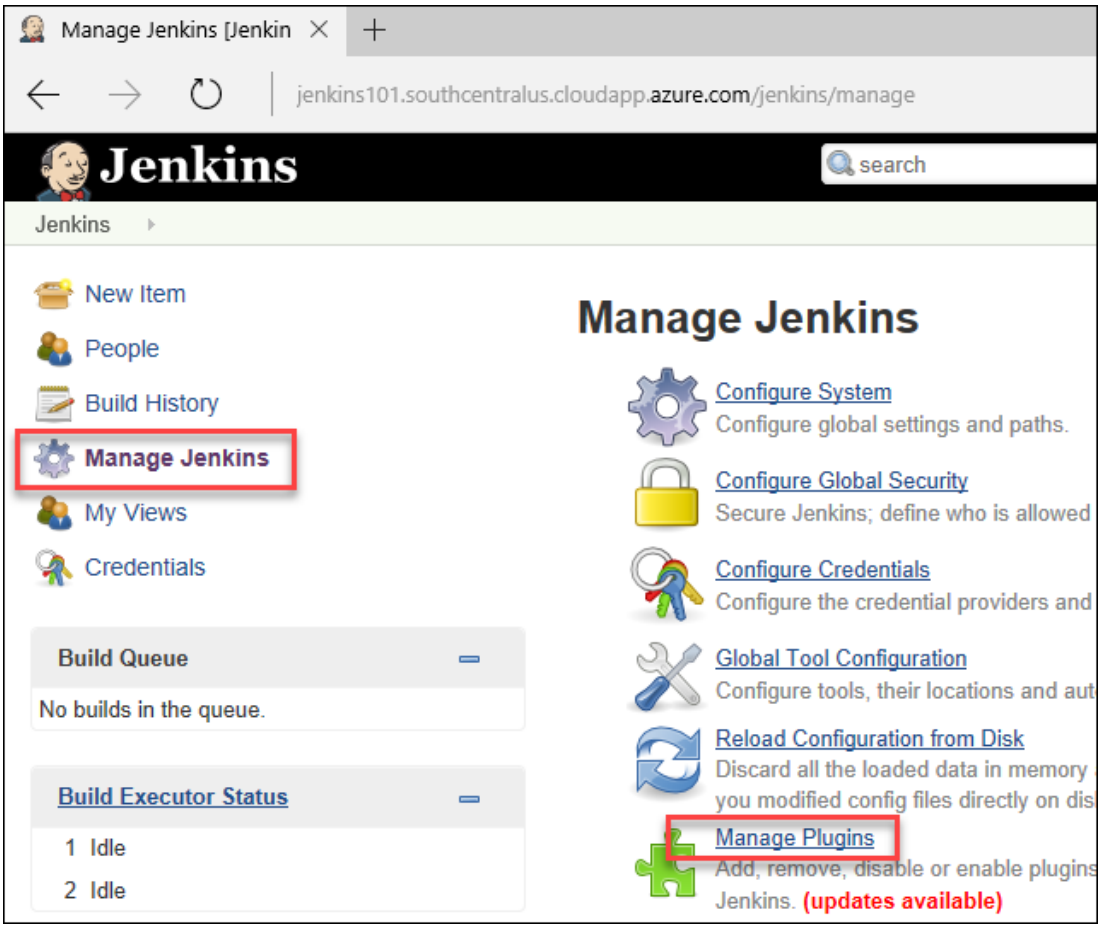
7. The default username is **admin**, and the password is the one you copied a few steps ago. Enter both, and click **log in**

8. Once logged into the portal, notice the version of Jenkins (**2.89.2** as of the writing of this lab). Click **Manage Jenkins** for more details

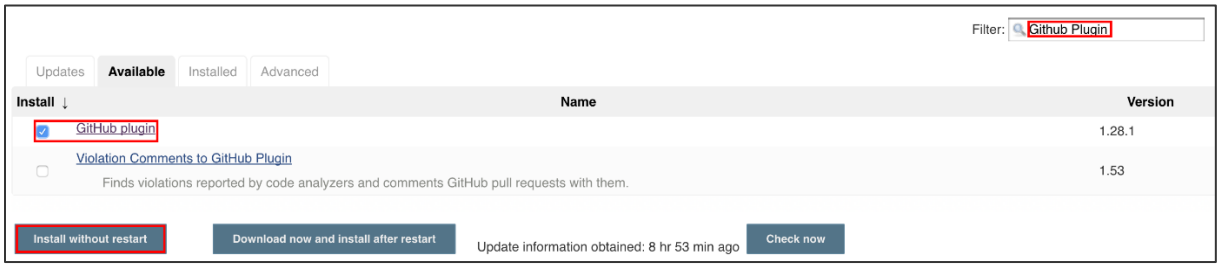


9. Here, you will get more information on any update that may be available for the server instance itself or for Plugins

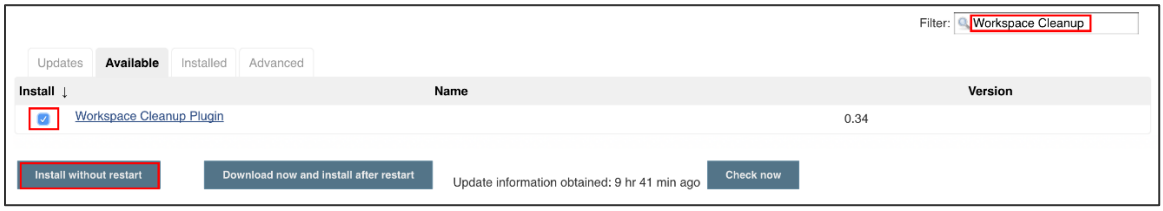
10. Click **Manage Jenkins** on the left-hand side followed by **Manage Plugins**



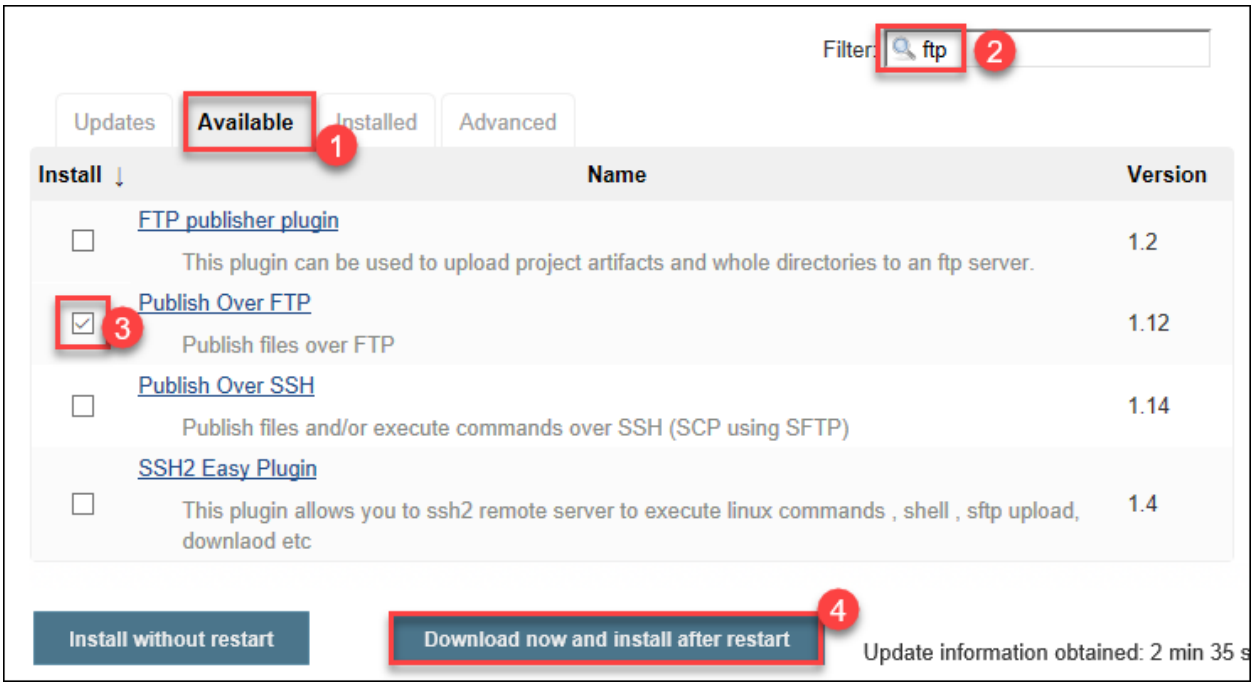
11. There are three Plug-Ins you need to install for the next exercises. Click the Available tab, and type **Github Plugin**. Click the check box for **Github plugin**, and click **Install without restart**.



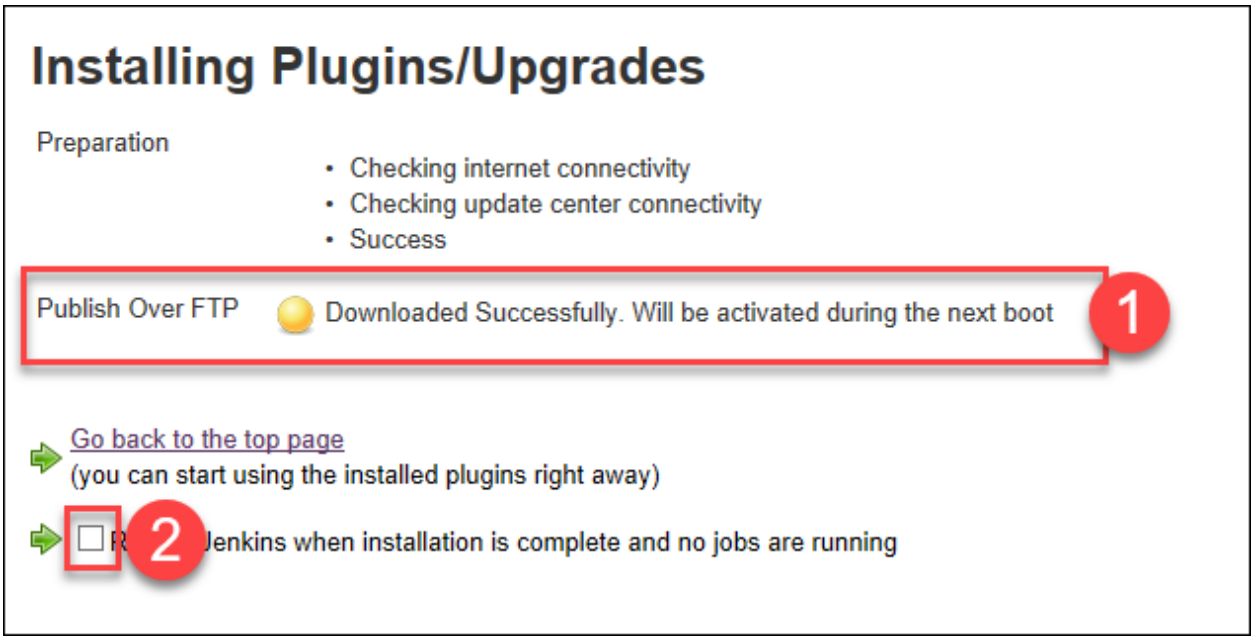
12. Next, click on the Available tab and type **Workspace Cleanup**. Click the check box for **Workspace Cleanup Plugin** and click **Install without restart**.



13. Next, click the Available tab, and type **ftp**. Click the check box for **Publish Over FTP**, and click **Download now and install after restart**



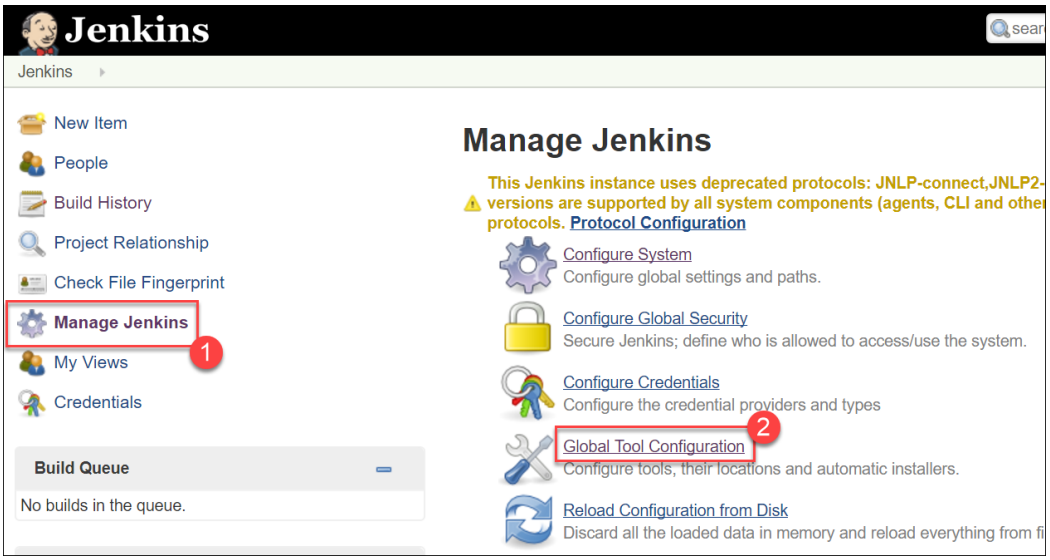
14. Check the box to Restart Jenkins



15. Once the restart has completed, you will be redirected to the login page once again. (*You may need to refresh the browser page to update the install status.*)

16. Log back into Jenkins portal with the **admin** account

17. Navigate to **Manage Jenkins** followed by **Global Tool Configuration**



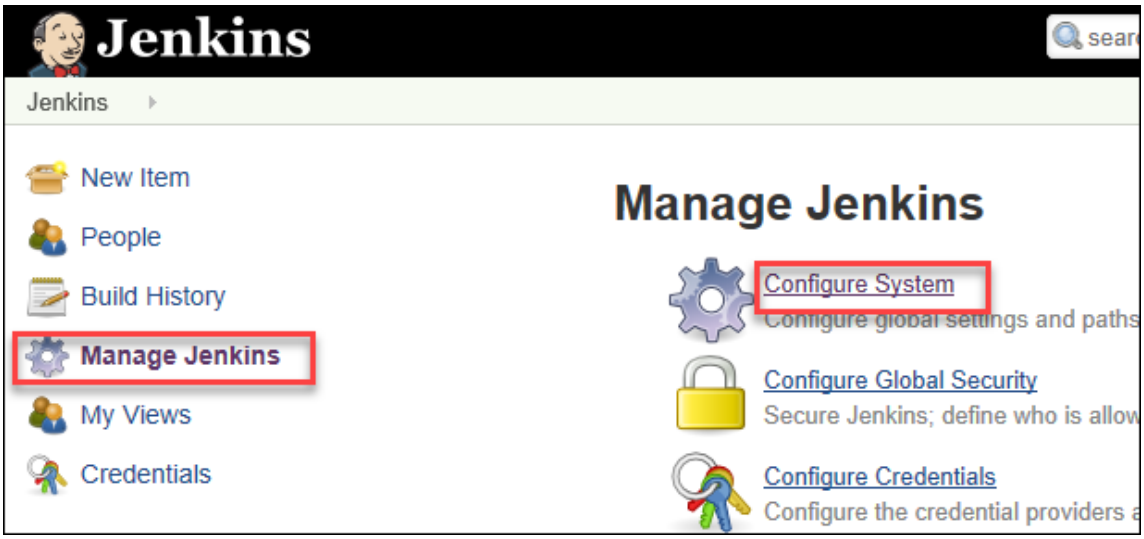
18. Under **Git**, ensure the **Name** is set to ***Default***, and the **Path to Git executable** is set to ***git***

19. Click **Save**

Task 3: Configure Jenkins staging deployment

You are now ready to define your staging deployment job.

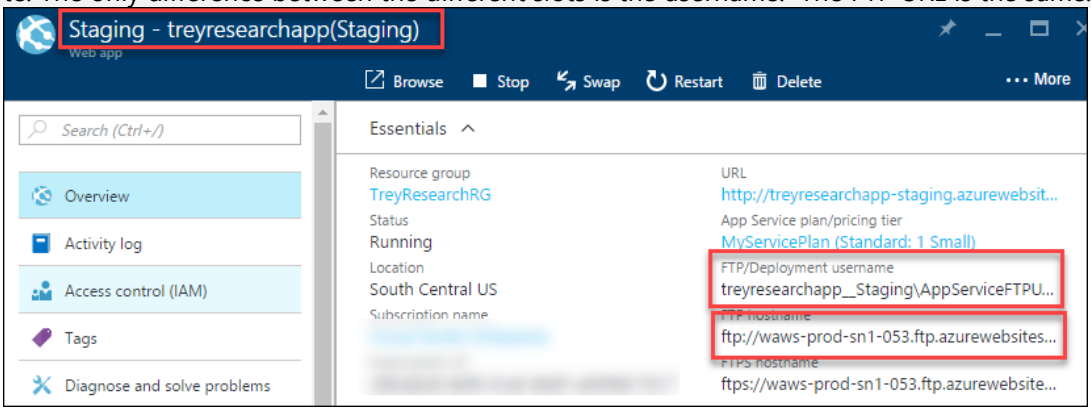
1. Login to your Jenkins portal with the **admin** account
2. You will first configure your FTP Plugin with the information from your Azure App Service. Click **Manage Jenkins** followed by **Configure System**



3. Scroll down to the section titled **Publish over FTP**, and click **Add**

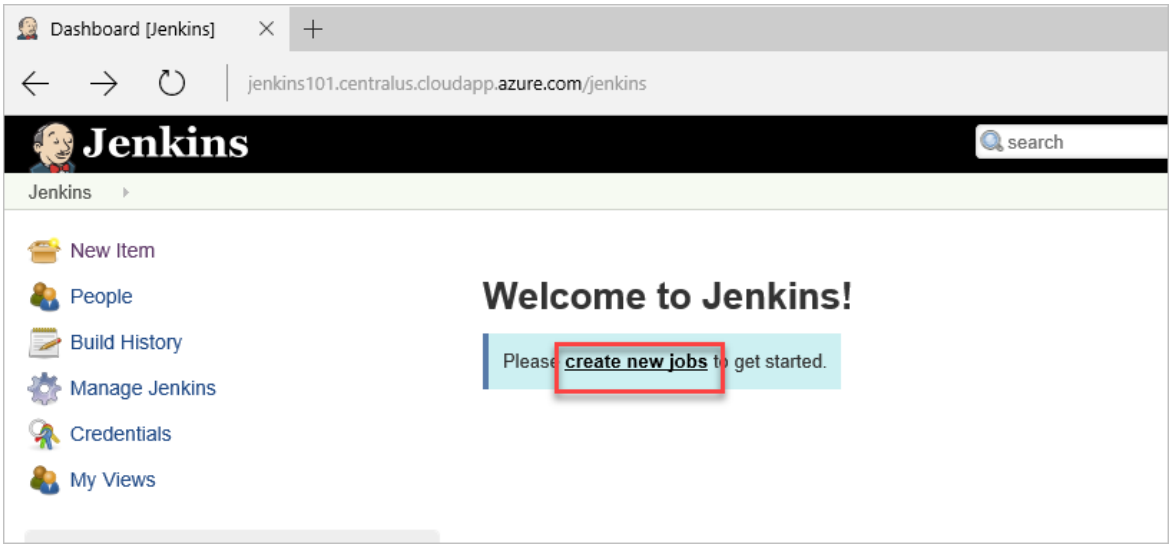
4. The information that is needed here can be obtained from the settings of the App Service in Azure. Since you will be deploying to the Staging slot, be sure to get the information from the staging slot settings.

Note: The only difference between the different slots is the username. The FTP URL is the same.

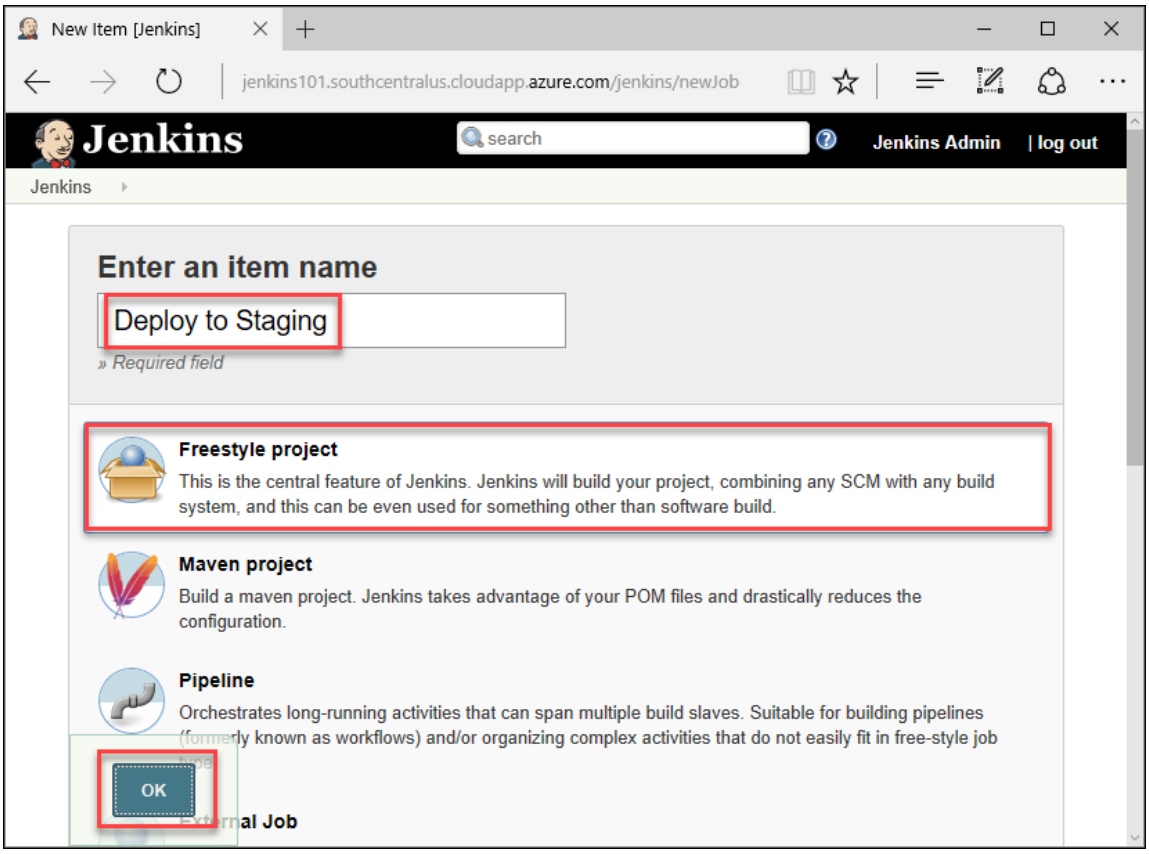


5. Update the Plug-In settings with the following information, and click **Save** (after confirming the connection is successful by clicking **Test Configuration**):
    a. Name: **Staging Slot for Web App**
    b. Hostname: <**ftp hostname from web app staging slot settings**>
    c. Username: <**username from web app staging slot settings**>
    d. Password: **Demo@pass123**

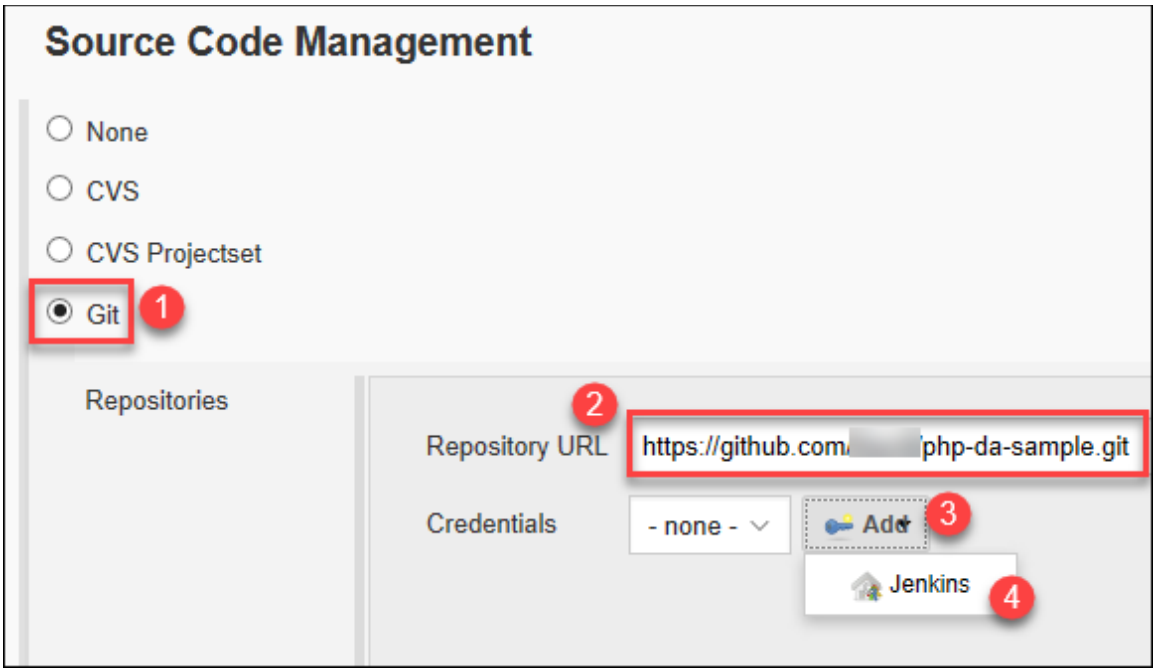
6. On the Welcome page, click the **create new jobs** link

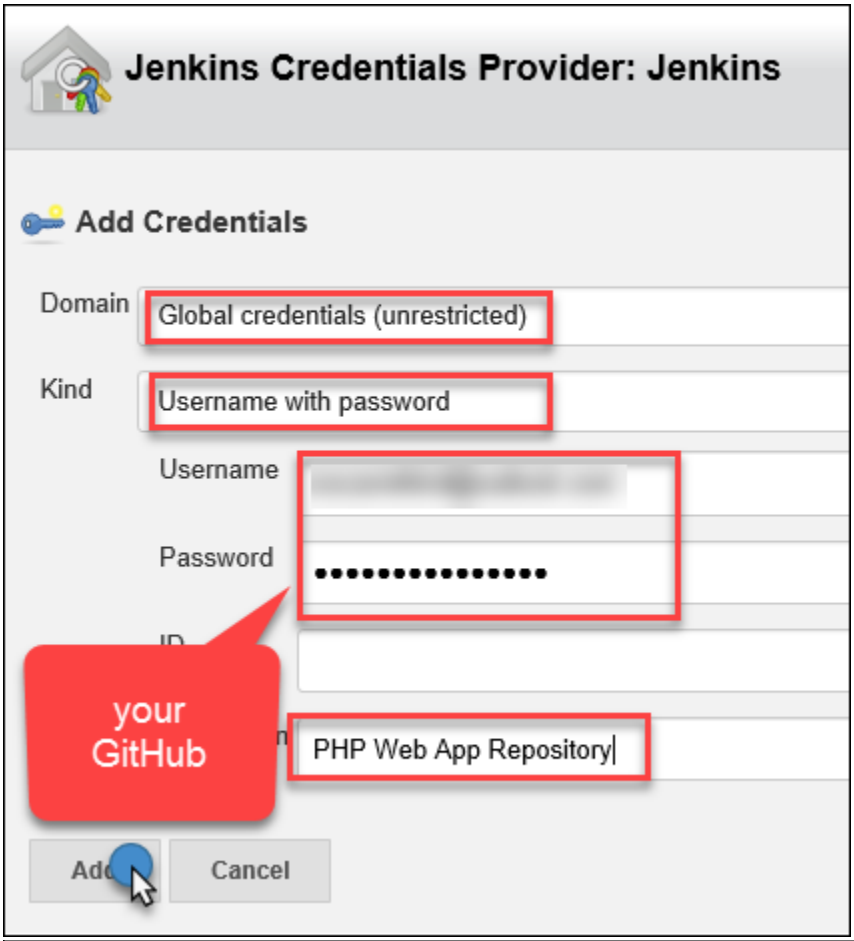


7. Choose Freestyle project and name the project **Deploy to Staging**, and click **OK**

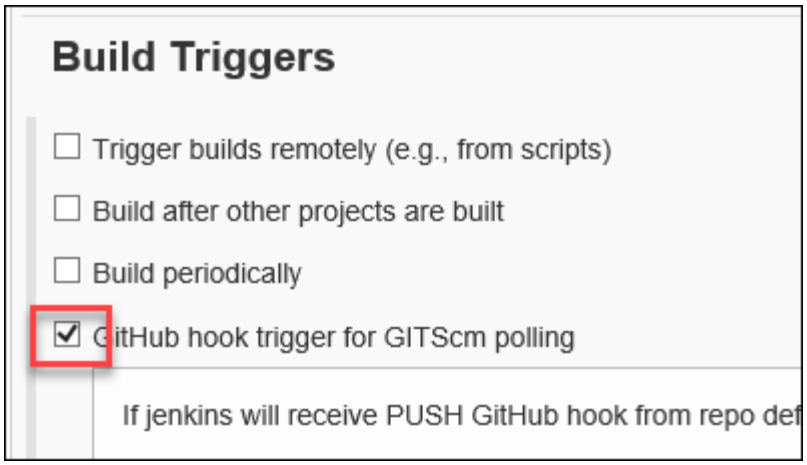


8. In the **Source Code Management** section, choose Git and specify your GitHub repository URL. Click **Add** to configure your credentials.

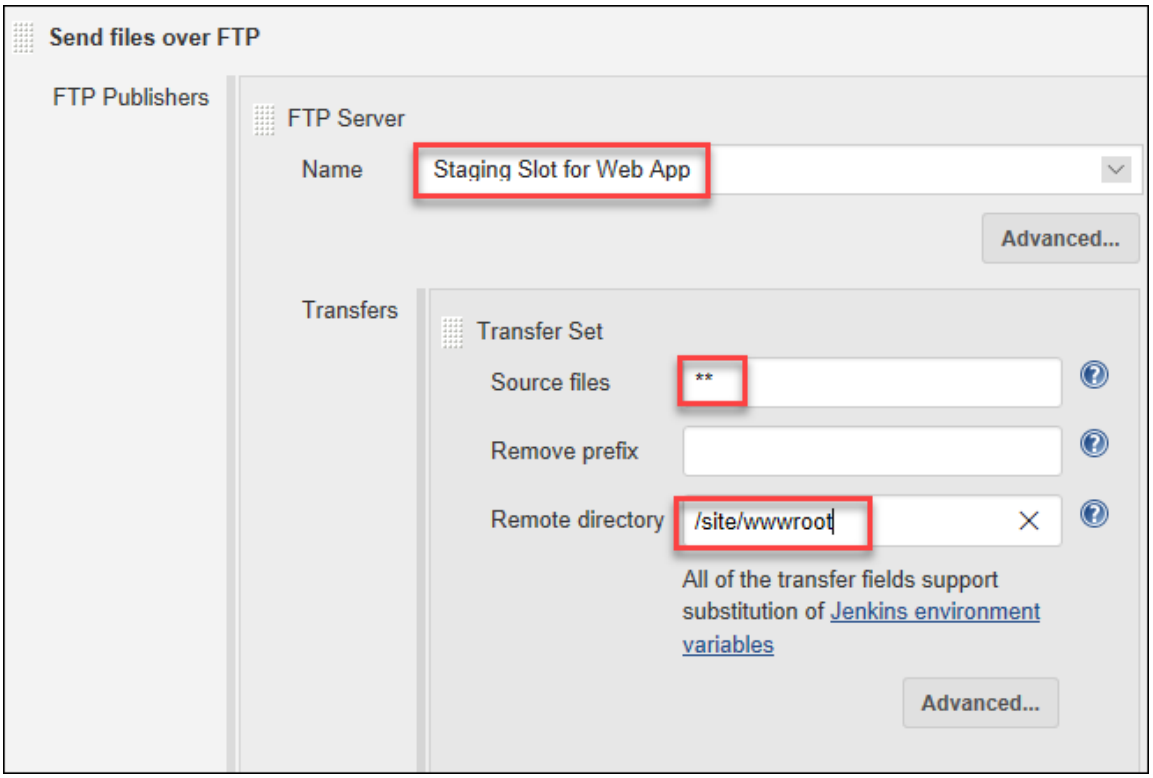




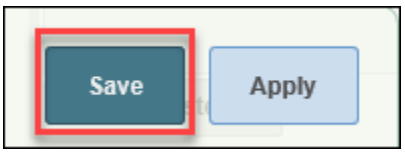
9. In Build Triggers, check **GitHub hook trigger for GITScm polling**



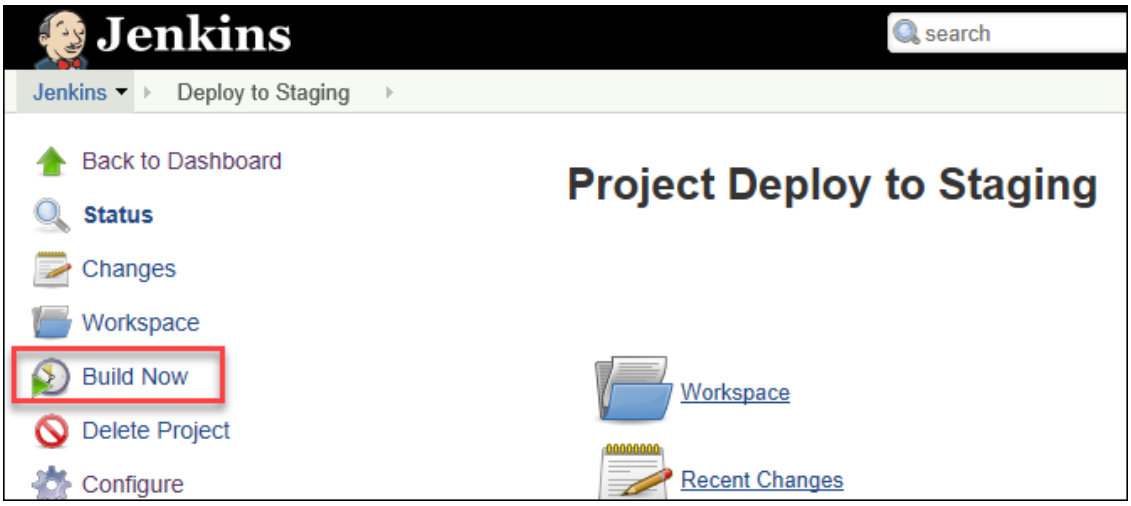
10. In the Build section, from the Add build step dropdown list, choose **Send files over FTP**. Be sure to choose the FTP you defined earlier, specify ****** for the **Source files** and **/site/wwwroot** for the **Remote directory**.



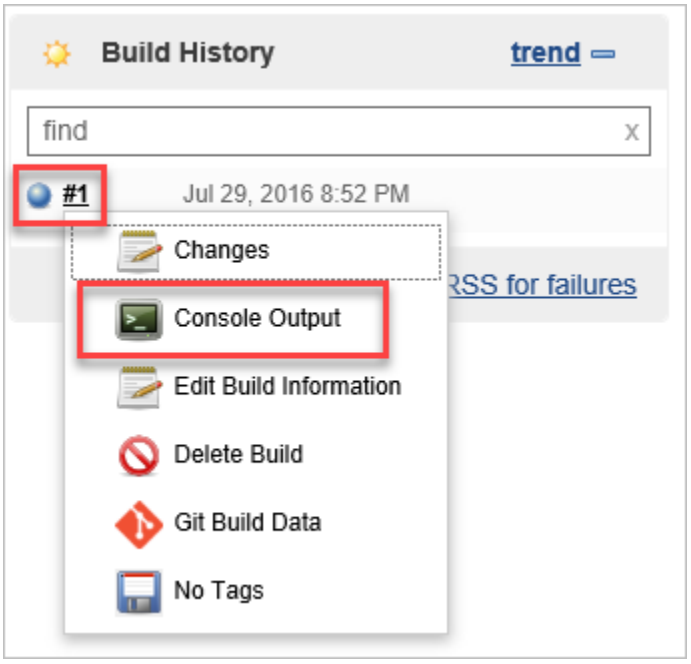
11. Click **Save** to save your changes



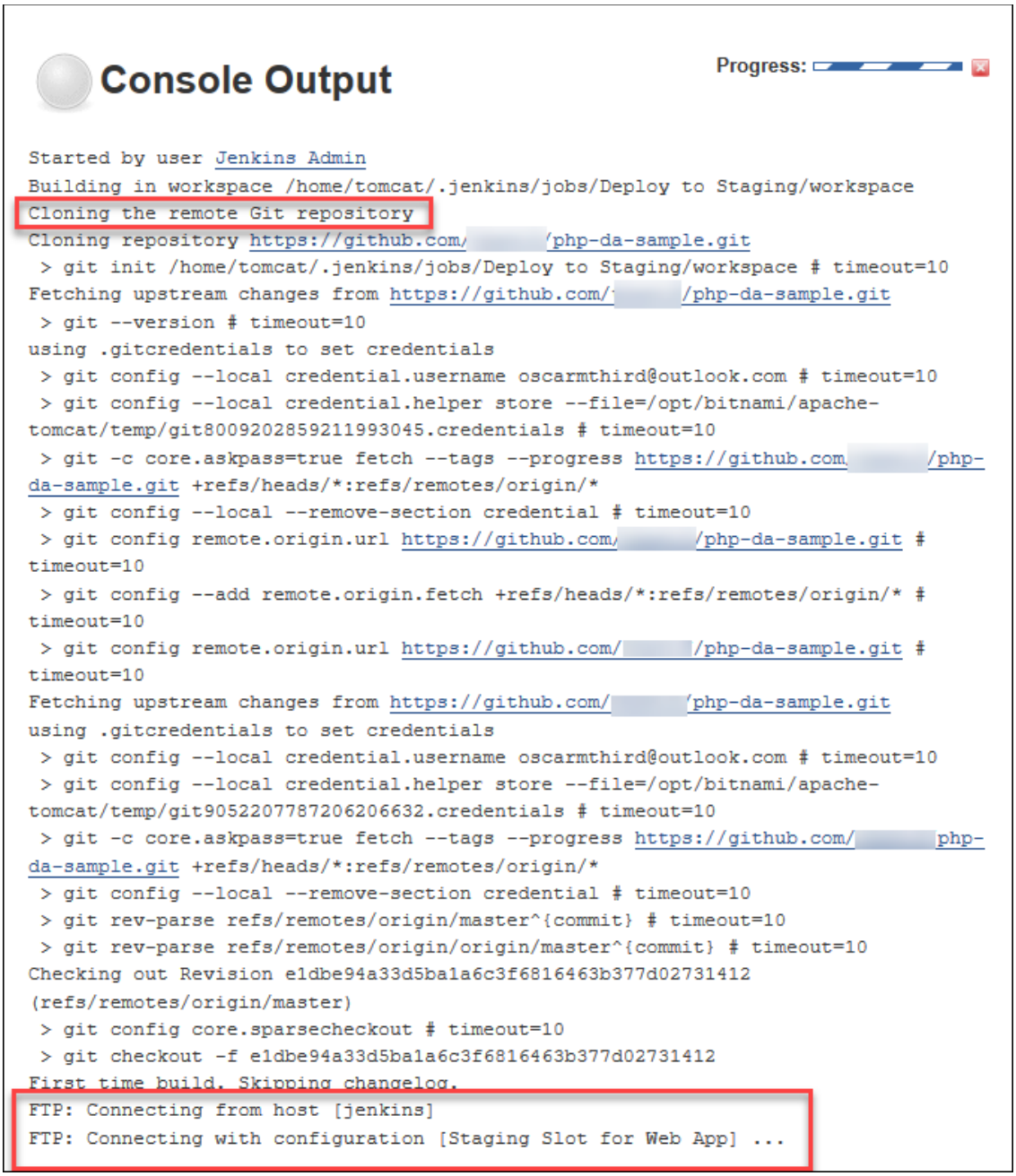
12. Although you specified to run this project whenever a check-in was done in GitHub, you can force the job to run. Click **Build Now** on project page.



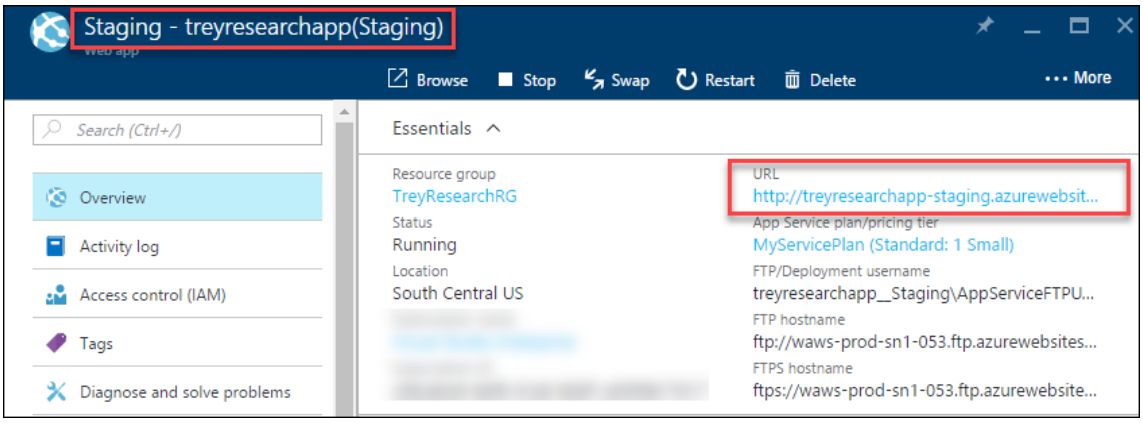
13. You will see an entry in Build History with your build number. Hover over the number to get a dropdown list of options. Choose **Console Output**.



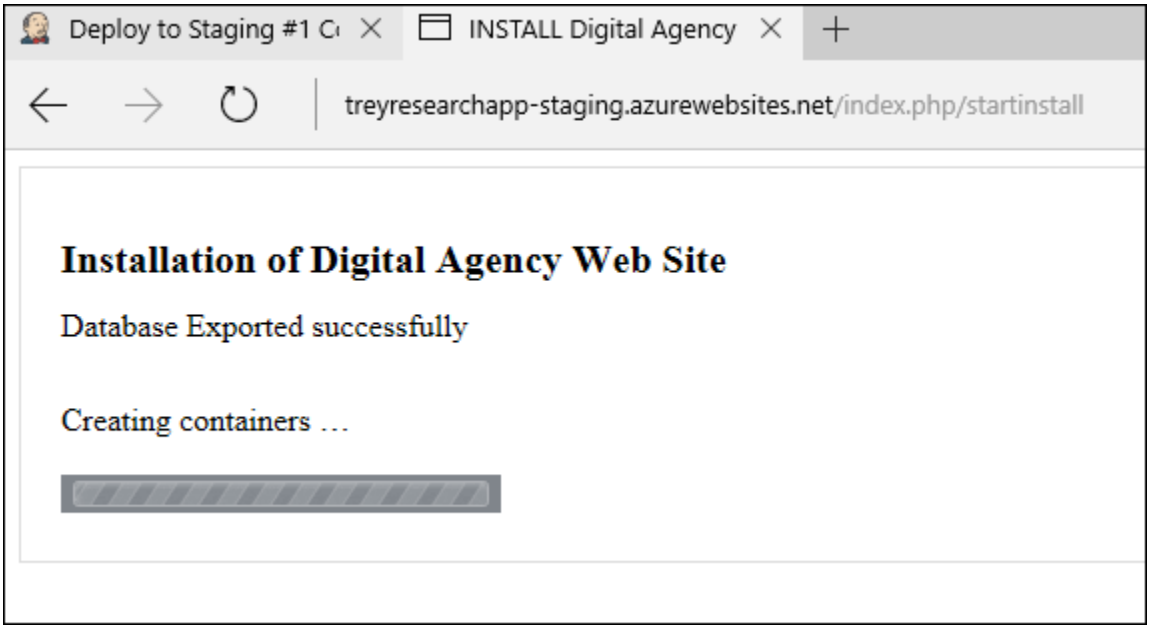
14. Review the Console Output for any errors etc. You can see in the Console Output where the call to your GitHub repo is made and where it is synced on your Jenkins server. You can also see the commands specified and their output.



15. Once the job has completed, you can verify the deployment by browsing to the URL of the staging slot

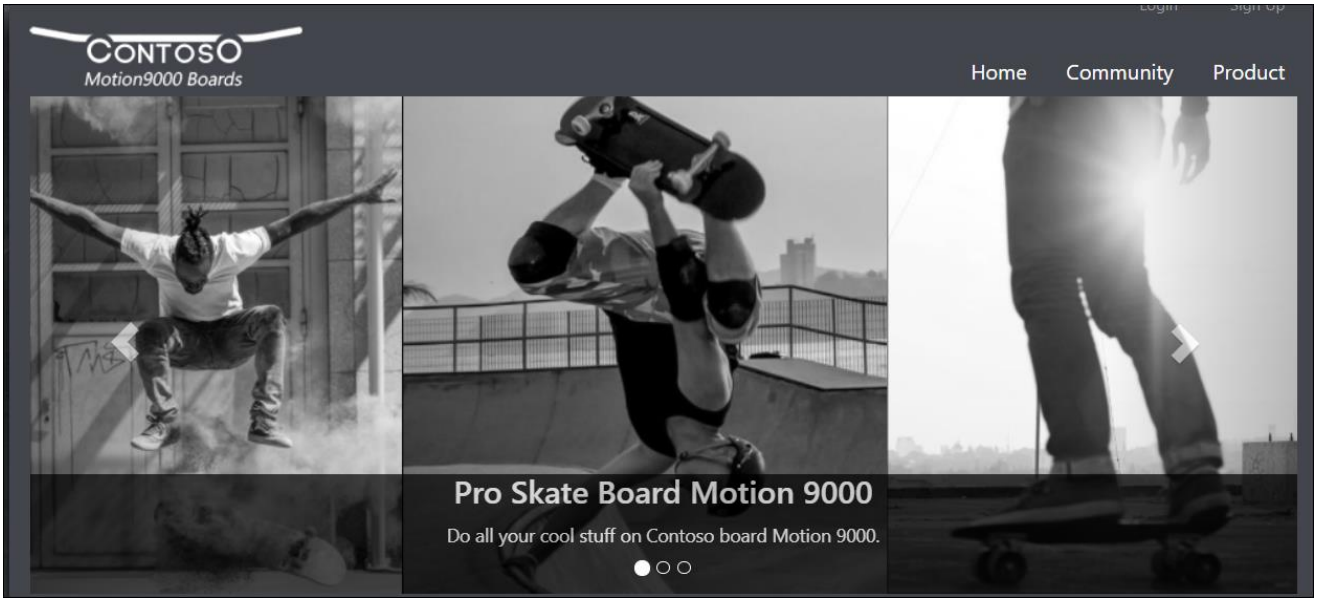


16. You will notice the site has been deployed, and it is now beginning to provision itself



17. Click the **click here** link to continue to the completed site

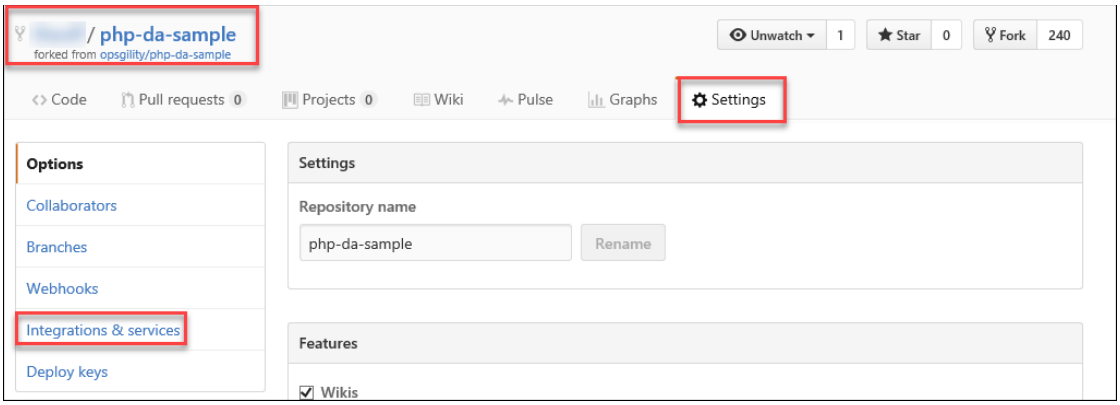
18. You should see the home page load



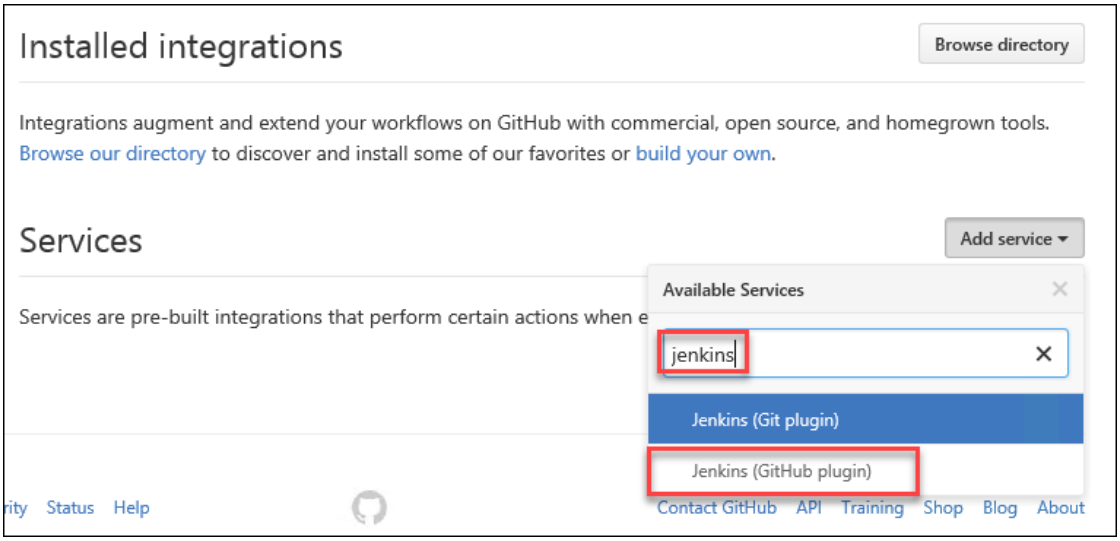
Task 4: Configure your GitHub repo to notify Jenkins of changes

You will now configure your GitHub repository to notify your Jenkins server when a change has occurred, so the Jenkins Job is kicked off automatically.

1. Log into your GitHub repo (https://github.com/<username>/php-da-sample), and click on **Settings** followed by **Integration & services**

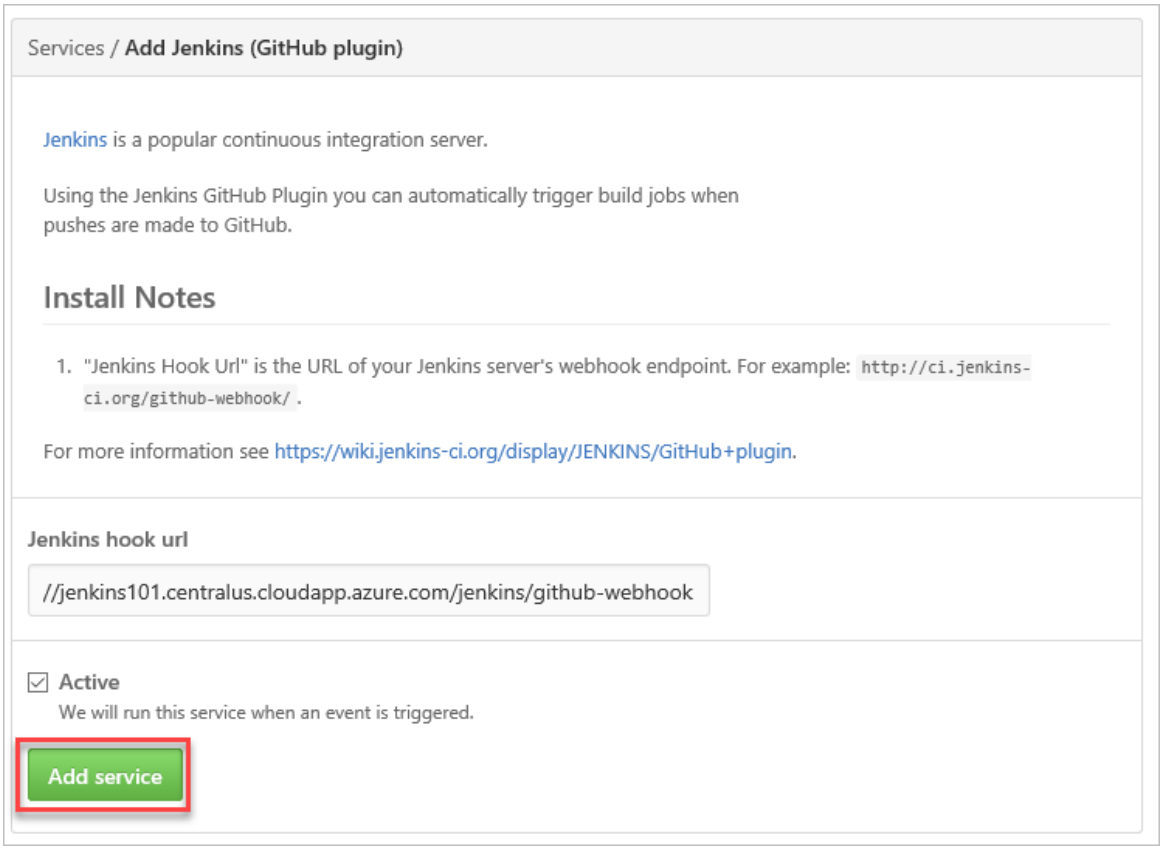


2. Click **Add service**, and choose **Jenkins (GitHub plugin)**



3. For the Jenkins hook URL, enter the following (after updating the string with your servers FQDN). **Note the trailing slash (/)** – make sure it is included in your URL:

```
http://<your-jenkins-vmname>.<your-azure-region>.cloudapp.azure.com/github-webhook/
```



Task 5: Check in a change to trigger Jenkins job

You will now check in a change to your Web Application code that will trigger your Jenkins job by editing the file that updates the home page.

1. Open the following file in an editor such as Visual Studio Code **php-da-sample\application\views\pages\home.php**
2. Click Ctrl+F, and find the following HTML code:

```
<h3>Pro Skate Board Motion 9000</h3>
<p> Do all your cool stuff on Contoso board Motion 9000.</p>
```

3. Make a modification to the text such as changing the model number, and click File→Save

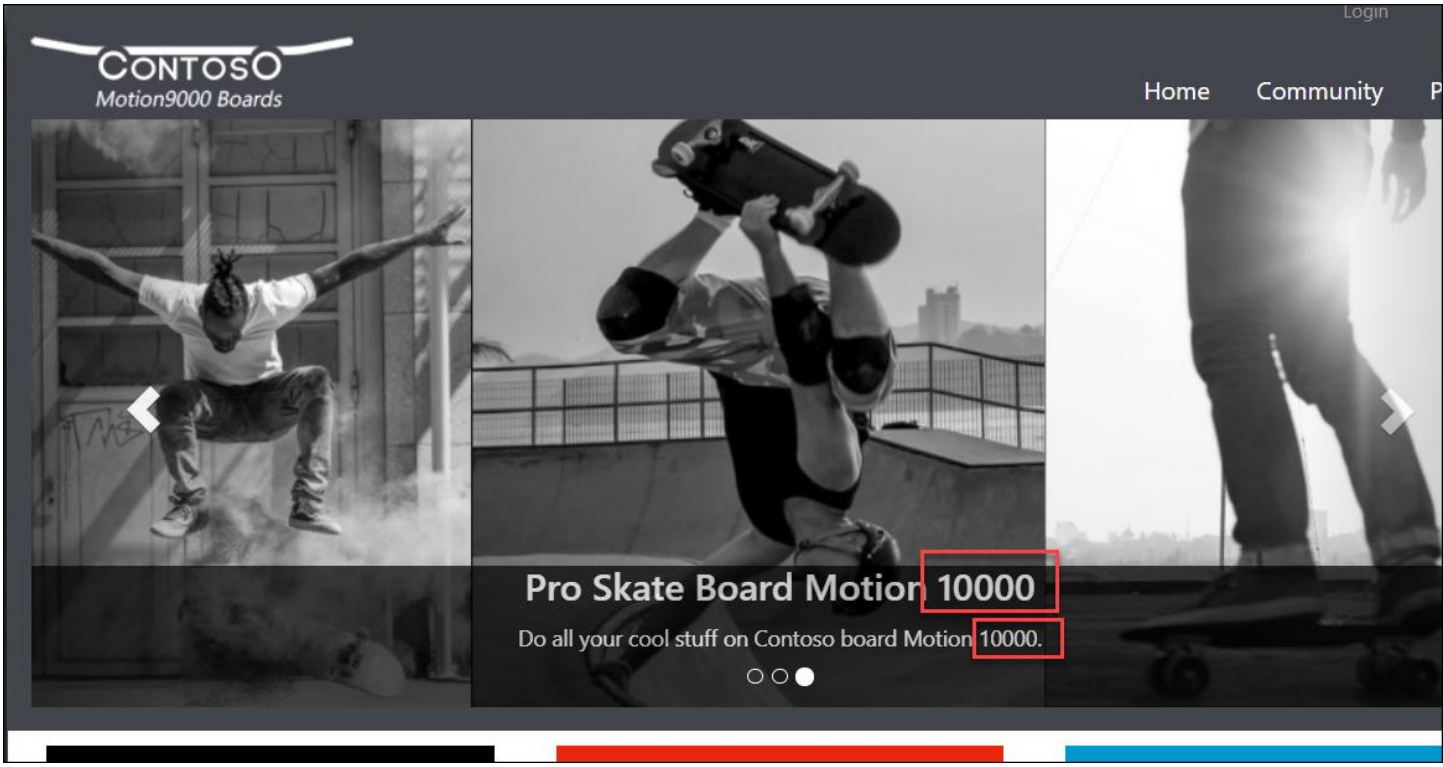
```
<h3>Pro Skate Board Motion 10000</h3>
<p> Do all your cool stuff on Contoso board Motion 10000.</p>
```

4. Move to a **Git Shell**, and execute the following git commands from the directory where the repo resides to push the update to your repository in GitHub

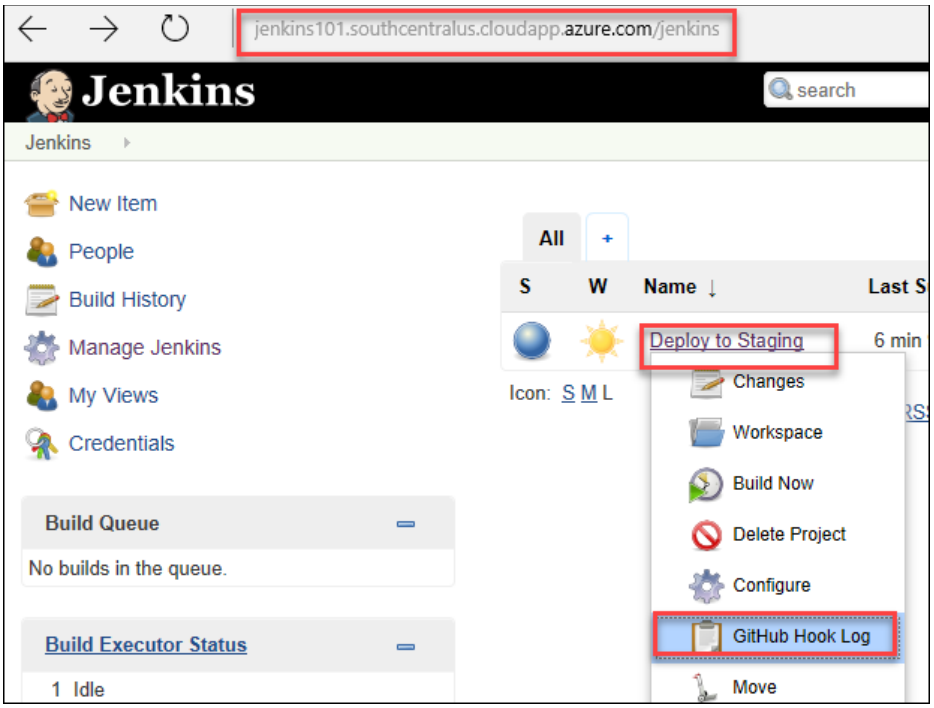
```
git config user.name "Your Name"
git config user.email "your@email.com"
git add -A
git commit -m "updated model"
git push
```

Note: You may be required to authenticate using your github.com username and password

5. Now that a change has been checked in, there are various places to check that the process has worked. Start by checking that the staging website has been updated.



6. You can also check logs in Jenkins. From the home page of your Jenkins portal, click the drop-down of your Jenkins job, and choose **GitHub Hook Log**.

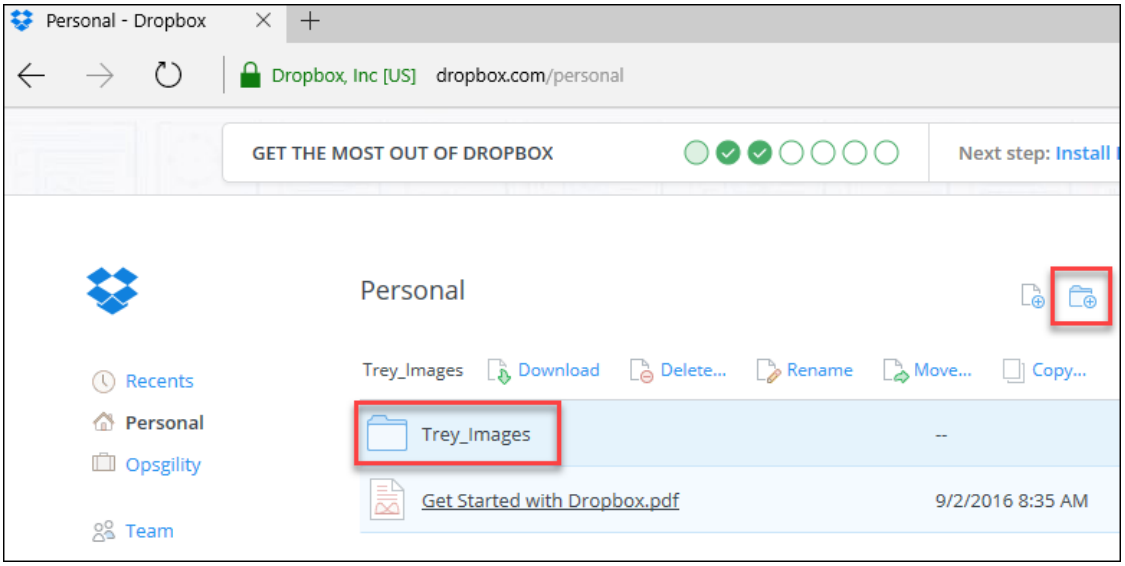


7. You can also check the Build History log and Console Output of the Jenkins project.

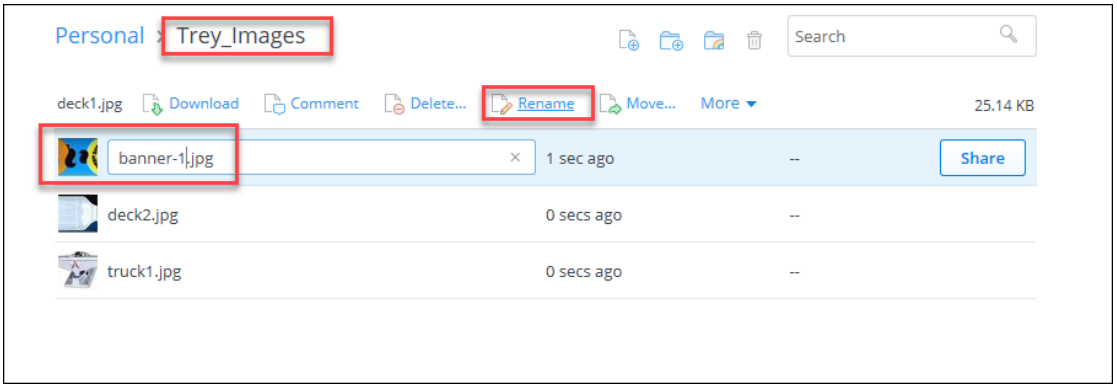
Task 6: Update Jenkins Project to account for Dropbox content

Up to this point, you have configured a Web Application in Azure where the source is deployed via FTP and Jenkins. You will extend your Jenkins deployment project to account for your Dropbox content. This is a useful scenario where one team or department, typically not technical in nature, submits content for a website. We can separate these activities from those the development team might perform to update a website structure or features. In this scenario, a marketing department might update a folder of images to keep the look of a website fresh.

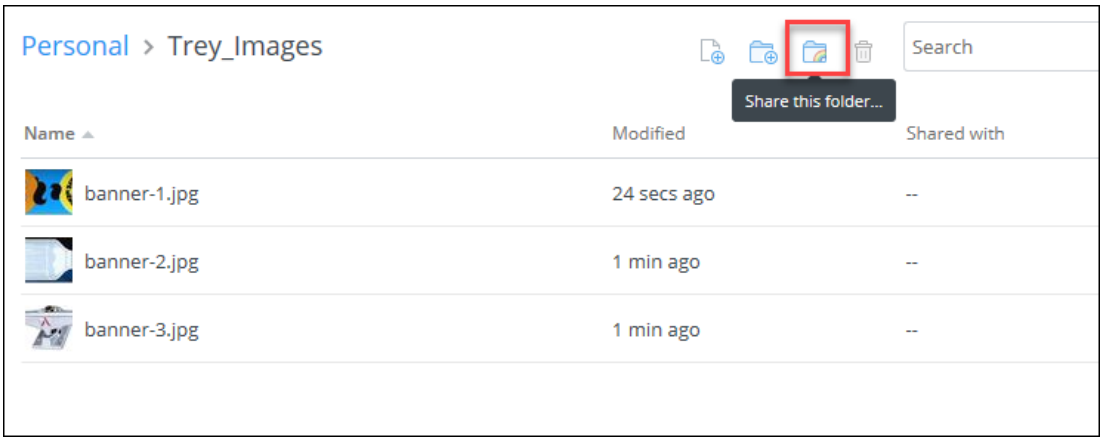
1. Log into your Dropbox account at https://www.dropbox.com, and create a new folder name **Trey_Images**



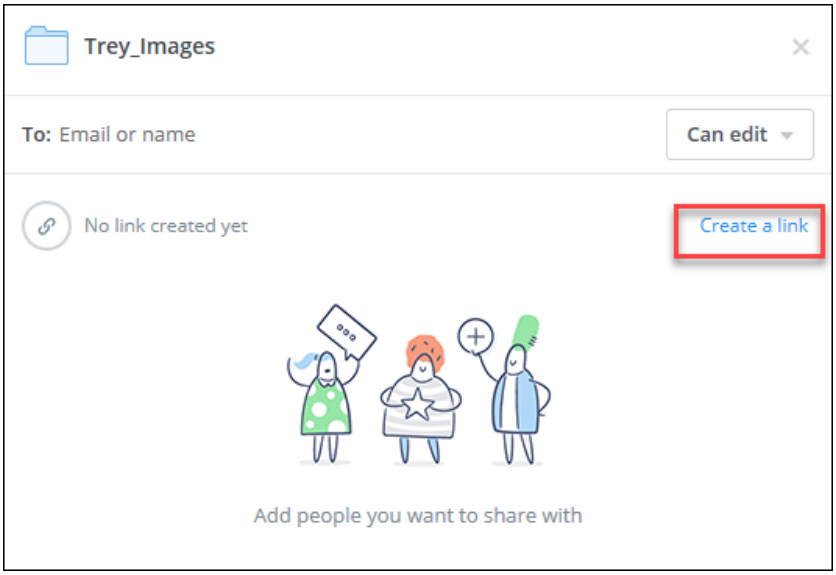
2. Upload the following three images from your local git repository. You are going to replace the three banner images on the homepage with three product images from **php-da-sample\assets\images\products\large**. Upload the following three images and rename them accordingly:
   a. deck1.jpg –> banner-1.jpg
   b. deck2.jpg –> banner-2.jpg
   c. truck1.jpg –> banner-3.jpg



3. To create a share link of this folder, click the **Share this folder... icon**

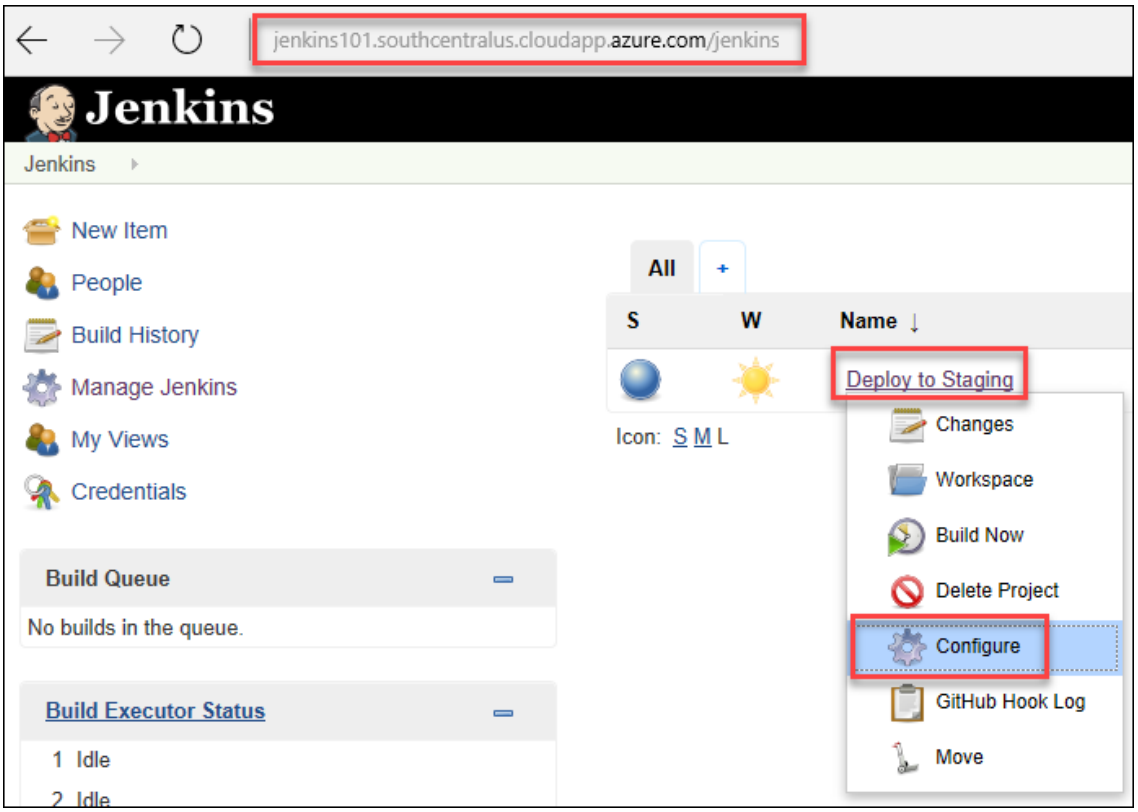


4. Click **Create a link**

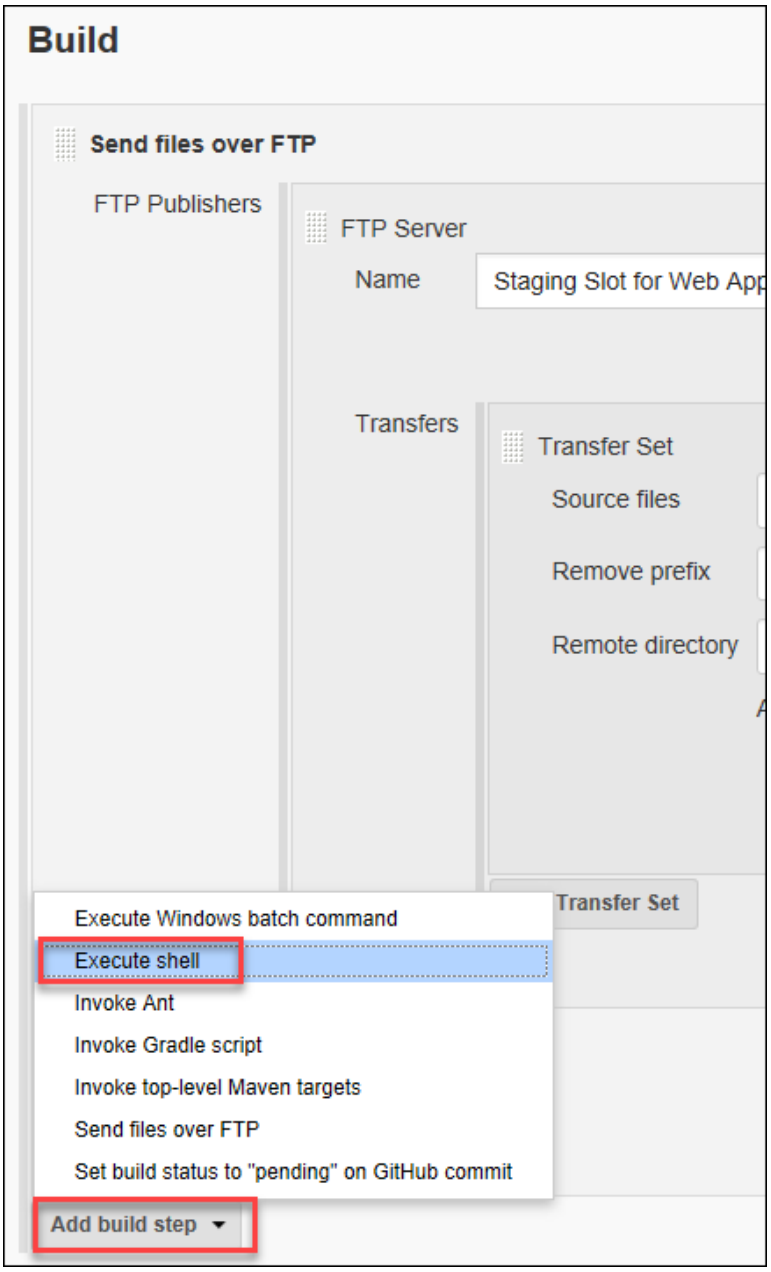


5. Once the link is created, click **Copy link**, and save to a file for use later

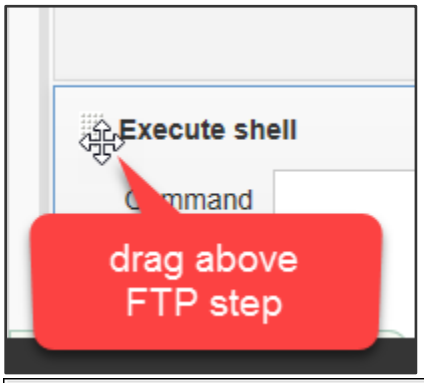
6. Log back into your Jenkins portal, hover over your previously defined job (**Deploy to Staging**) to get the dropdown list, and choose **Configure**

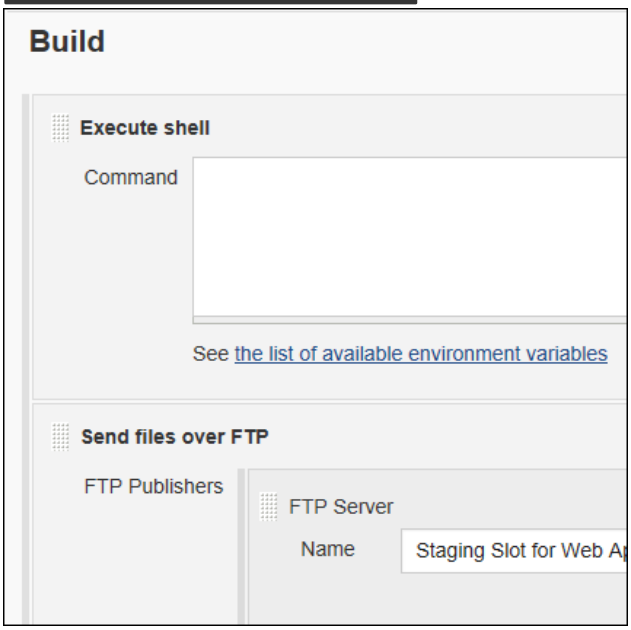


7. You will now add another Build step (**Execute shell**) that should execute before the FTP copy



8. Drag the newly created **Execute shell** step above the **Send files over FTP** step

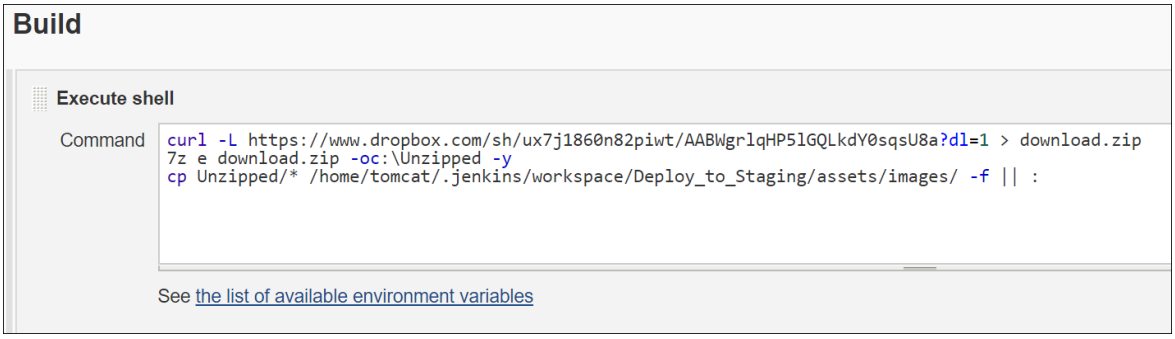




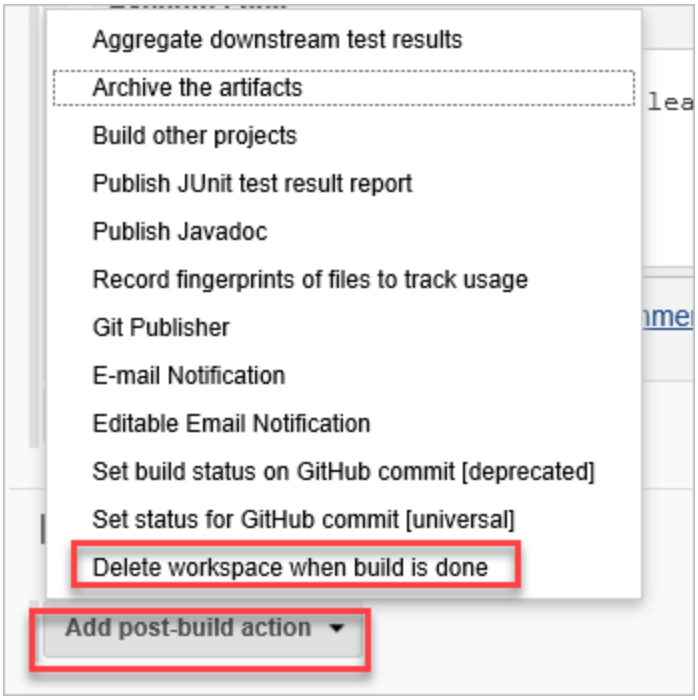
9. In the **Command** window, add the following commands:

Note: The URL should end in dl=1 instead of dl=0. Also, the bolded command below is case sensitive (be sure the name of your job has the correct casing based on what you named your project – **Deploy to Staging** in the case of this lab guide).

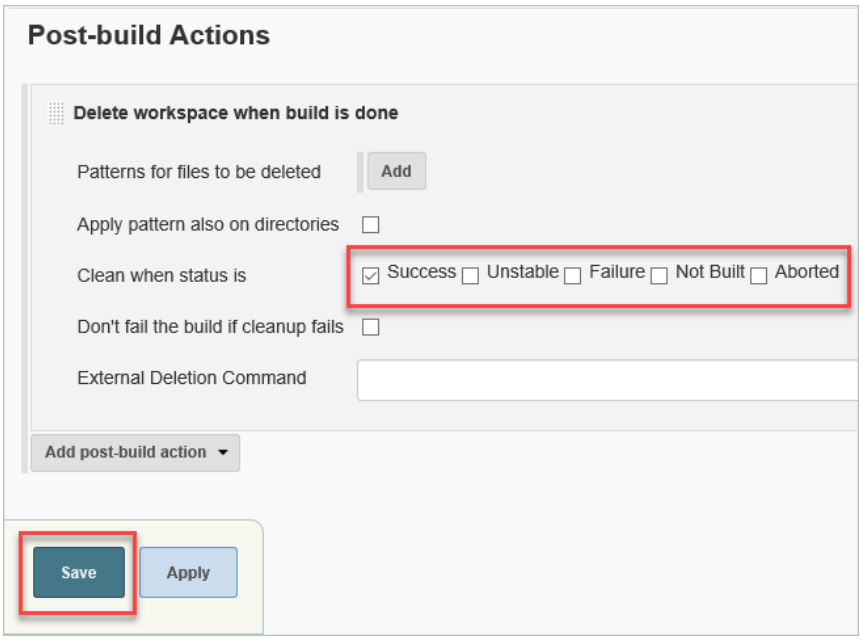
```
curl -L https://<dropbox_folder_share_url_dl=1 > download.zip
7z e download.zip -oc:\Unzipped -y
cp Unzipped/*
/home/tomcat/.jenkins/workspace/Deploy_to_Staging/assets/images/ -f || :
```



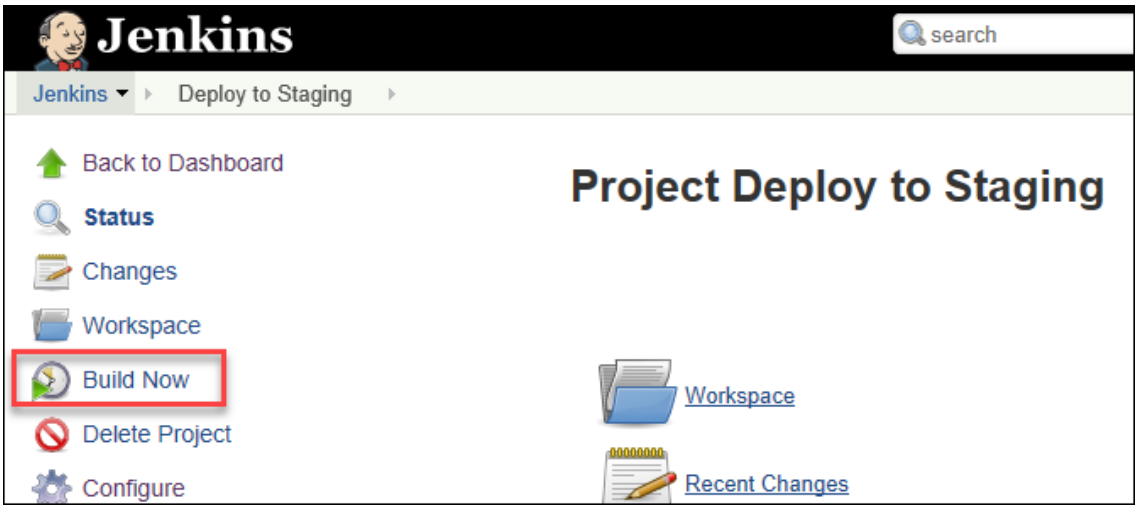
10. In the **post-build Actions** section, from the **Add** post-build action dropdown list, choose **Delete workspace when build is done**



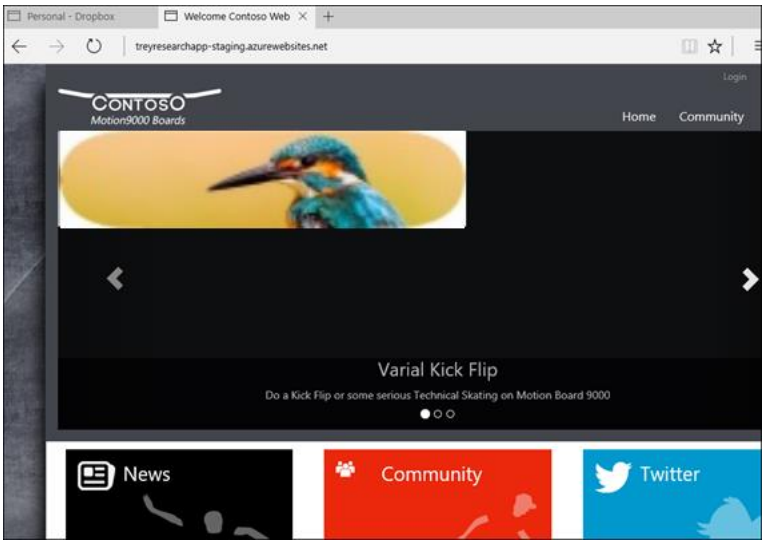
11. Click **Advanced**, make sure only **Success** is checked, and click **Save** to commit your changes



12. Click **Build Now** on project page to force a build and deploy to staging.



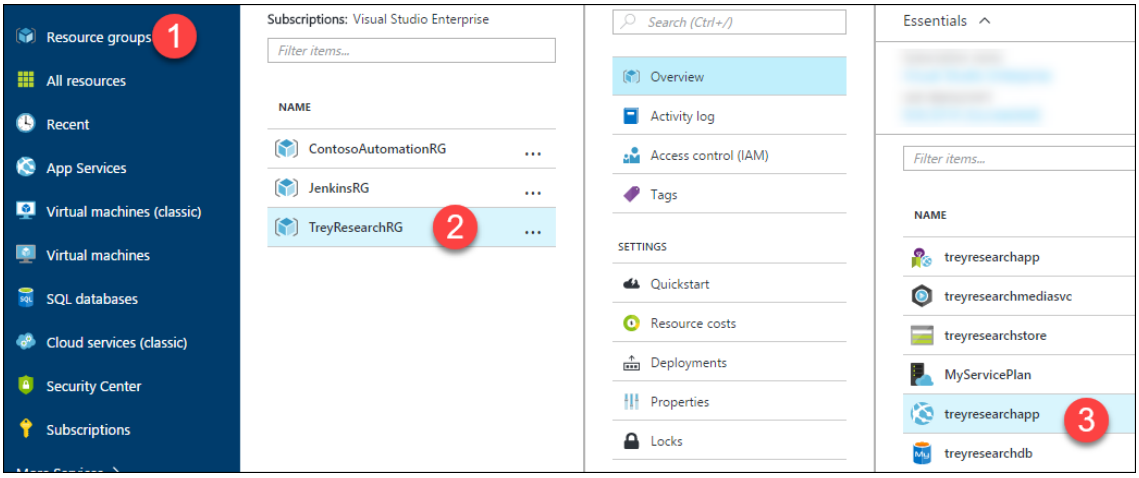
13. Once the build completes, browse to the staging slot of your Web Application, and you should see the updated images (that came from Dropbox) reflected



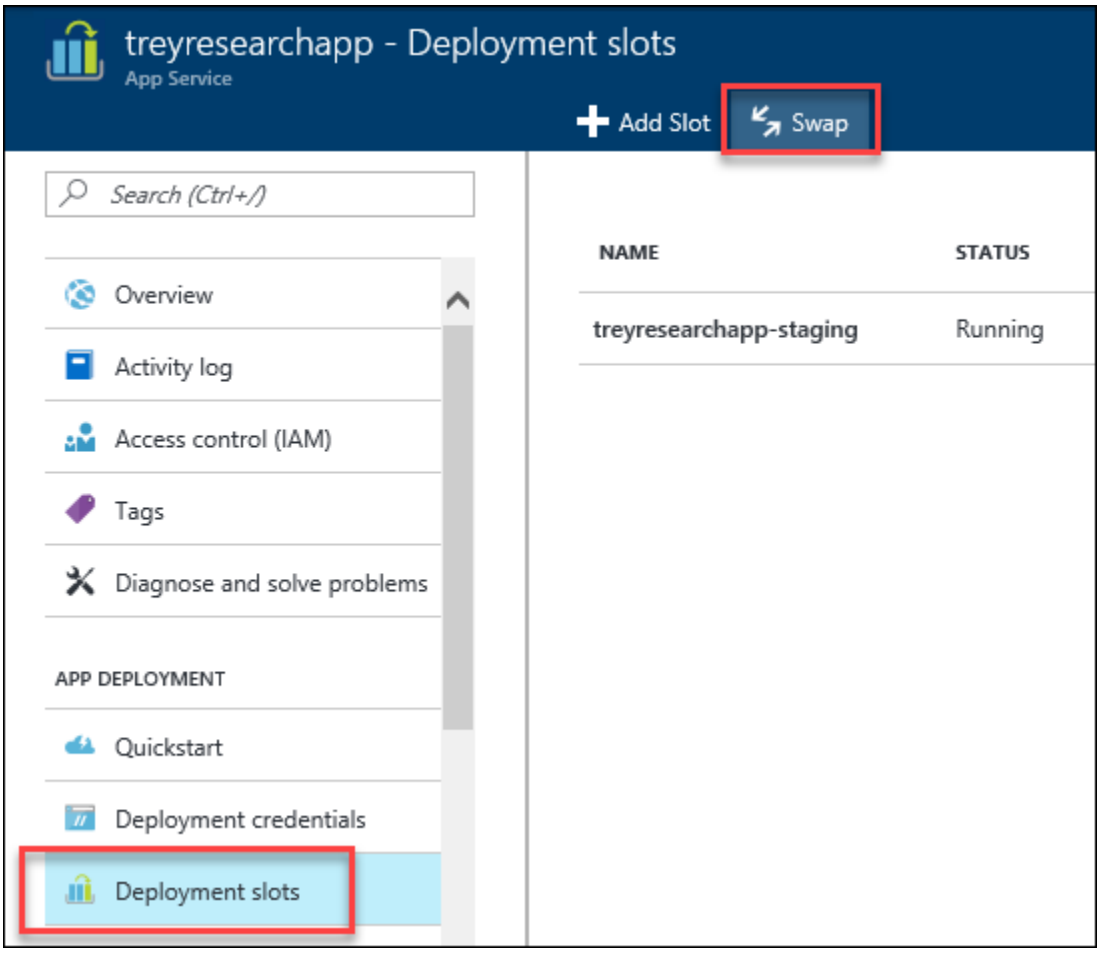
Task 7: Manually Deploy to Production

Up to this point, you have automated the integration and delivery to your staging slot. You will now move those changes into the production slot *manually*. Leveraging the tools you have configured so far, you could ultimately automate this last step to get continuous deployment.

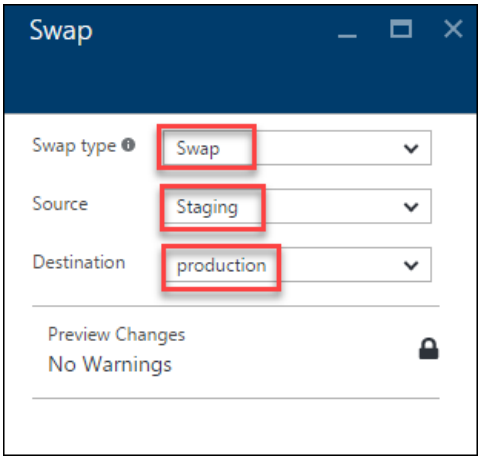
1. To push the changes from your staging slot to production, in the Azure portal (https://portal.azure.com), click **Resource groups > TreyResearchRG > treyresearchapp**



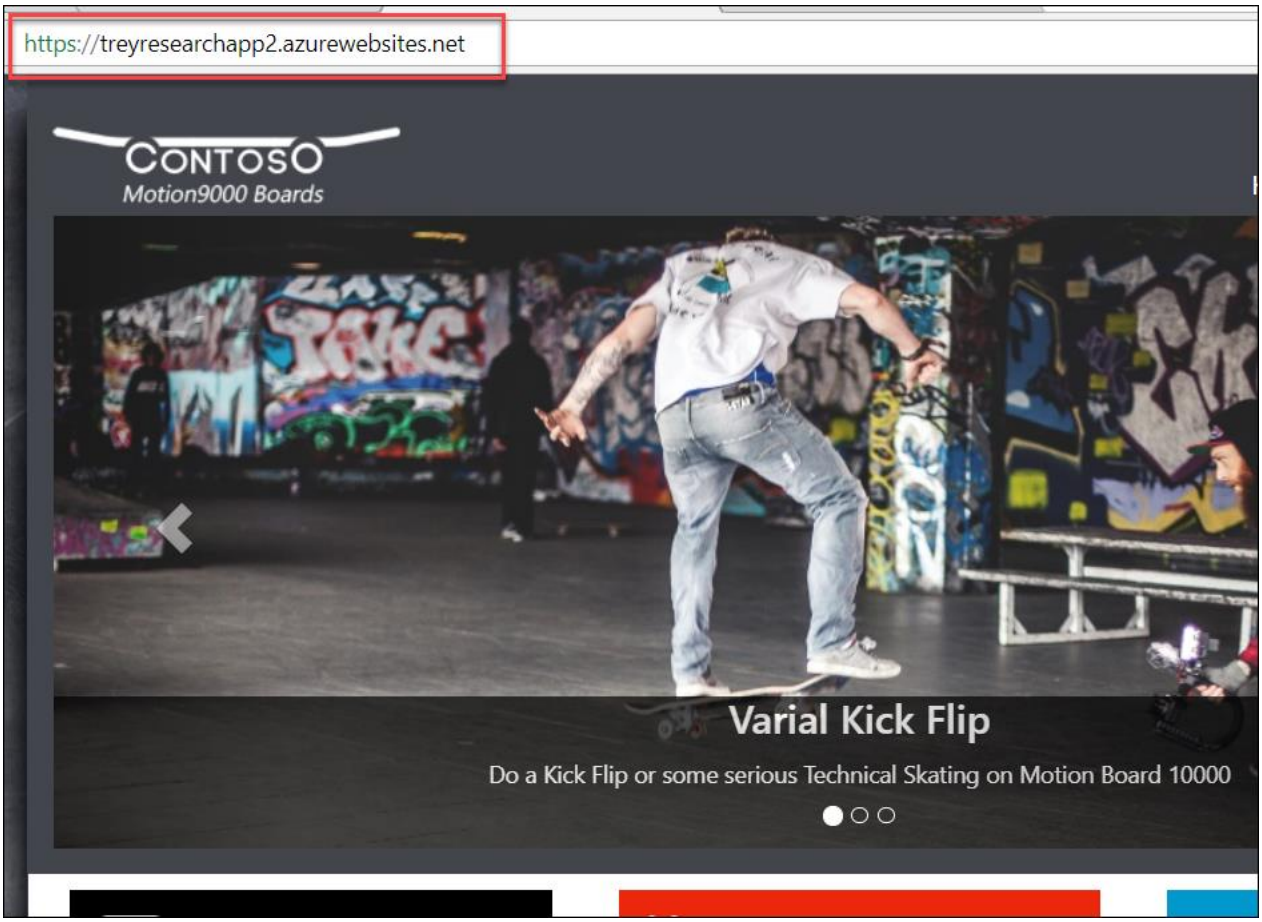
2. Click **Deployment slots > Swap**



3. Be sure that **Staging** is listed as the **Source** and **production** as the **Destination**, and click **OK**



4. Once the Swap has completed, verify the changes have been pushed to your production slot by browsing to your production URL



Summary

In this exercise, you leveraged Azure, Jenkins, GitHub & Dropbox to setup continuous integration, delivery, and deployment for your web site. You built a scenario where your code changes were automatically pushed out to a staging slot after collecting assets from GitHub as well as Dropbox.

After the hands-on lab

Duration: 10 minutes

Task 1: Delete Resources

1. Now that the Hands-on lab is complete, go ahead and delete all the Resource Groups that were created for this lab. You will no longer need those resources and it will be beneficial to clean up your Azure Subscription.

You should follow all steps provided *after* attending the Hands-on lab