## Microsoft Cloud Workshops

OSS DevOps Hackathon

Lab Guide

February 2017

© 2017 Microsoft Corporation. All rights reserved. This document is confidential and proprietary to Microsoft. Internal use only. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples are for illustration only and are fictitious. No real association is intended or inferred.

### Contents

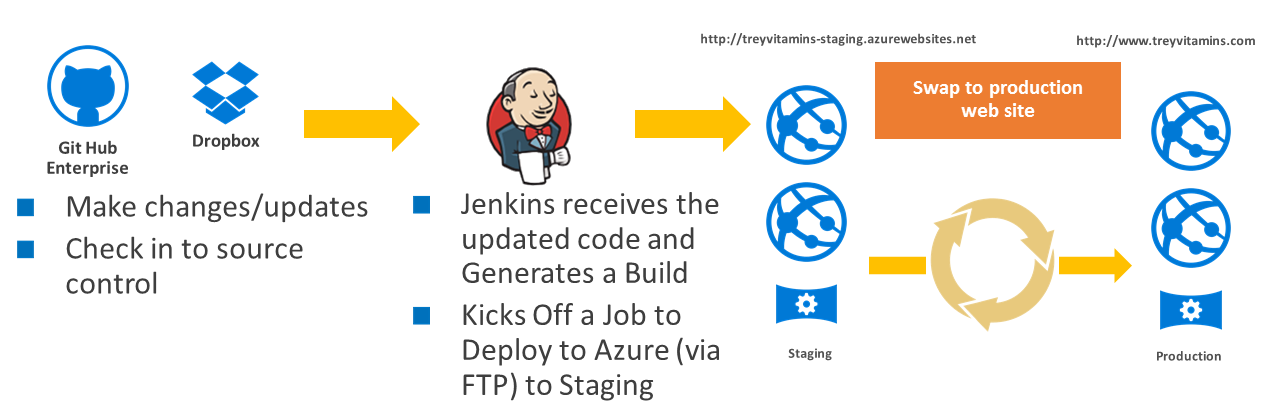
[OSS DevOps Hackathon 1](#_Toc474140919)

[OSS DevOps Hackathon Answers 6](#_Toc474140920)

# OSS DevOps Hackathon

## Overview

The OSS DevOps Hackathon is a hands-on exercise that will challenge you to implement an end-to-end scenario using a supplied sample that is based on Microsoft Azure App Services and related services. The scenario will challenge you to setup continuous integration and delivery of the application. The Hackathon can be implemented on your own, but it is highly recommended to pair up with other members at the Hackathon to model a real-world experience much closer and to allow each member to share their expertise for the overall solution.



## Requirements

* Microsoft Azure Subscription
* GitHub Account
* Dropbox Account

## Lab Structure

This lab has two sets of instructions. The first is a high-level set of instructions that is designed for students that have previous experience with Azure and Open Source tools for deployment (GitHub, Dropbox, and Jenkins). The second is a traditional hands-on lab guide that is designed for users that are new to Azure and Open Source tools for deployment (GitHub, Dropbox, and Jenkins).

## Help References

|  |  |
| --- | --- |
| 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/> |

## Exercise 1: Environment Setup

### Task 1: Install & Configure the necessary prerequisites

#### Tasks to complete

* Install GitHub Desktop
* Create a Dropbox account (if you don’t already have one)
* If leveraging a Windows VM on Azure:
  + Create a VM on Azure
  + Install GitHub Desktop
  + Download PuTTY
  + Create a Dropbox account (if you don’t already have one)

#### Exit criteria

All prerequisites have been accounted for before continuing.

## Exercise 2: Configure an Azure Web App

### Task 1: Create the Azure Web App

#### Tasks to complete

* Create a **Web App + MySQL** web application in Azure
* Place it in a Resource Group named **TreyResearchRG**
* Specify **MySQL In App (Preview)** as the Database provider

#### Exit criteria

Web App + MySQL application successfully deployed in Azure.

### Task 2: Create an Azure Storage Account

#### Tasks to complete

* Create a storage account in the **TreyResearchRG** Resource Group
* Make it **Standard** with **Locally-redundant storage (LRS)**
* Make note of the account name and Access key

#### Exit criteria

Storage account created with account name and key saved off for later use.

### Task 3: Create an Azure Media Service Instance

#### Tasks to complete

* Create a Media Services account in the **TreyResearchRG** Resource Group
* Specify the storage account created in previous task
* Make note of the account name and Account key
* Set Media reserved units to 2

#### Exit criteria

Media Services account created with account name and key saved off for later use.

### Task 4: Update and Review the Web App Settings

#### Tasks to complete

* Set the following variables in the Application settings of the Web App

|  |  |
| --- | --- |
| STORAGE\_ACCOUNT\_NAME\_WEBSITE | Storage Account Name |
| PRIMARY\_ACCESS\_KEY\_WEBSITE | Storage Account Key |
| STORAGE\_ACCOUNT\_NAME\_WEBJOBS | Storage Account Name |
| PRIMARY\_ACCESS\_KEY\_WEBJOBS | Storage Account Key |
| MEDIA\_SVC\_ACCOUNT\_NAME | Media Service Account |
| MEDIA\_SVC\_ACCESS\_KEY | Media Service Key |

* Set the PHP version to 5.5
* Specify a connection string named **defaultConnection** with the connection string for the local instance of MySQL

#### Exit criteria

Application settings updated on Production Slot.

### Task 5: Configure FTP Deployment Credentials

#### Tasks to complete

* Set FTP credentials on the Web Application to Demo@pass123

#### Exit criteria

Credentials set on the Web Application for use in later exercise.

### Task 6: Configure a Staging Slot

#### Tasks to complete

* Create a Staging Slot that duplicates the settings of the main web application

#### Exit criteria

Staging Slot created

## Exercise 3: Configure Local Git Repository

### Task 1: Fork a GitHub repository locally

#### Tasks to complete

* Fork the following GitHub repository and sync it locally (<https://github.com/opsgility/php-da-sample>)

#### Exit criteria

Local Git Repository of PHP Web Application source code

## Exercise 4: Configure Git and Jenkins for Continuous Integration, Delivery and Deployment

### Task 1: Deploy a Jenkins Server in Azure

#### Tasks to complete

* Deploy a Jenkins server on Azure (use the Bitnami version on the Azure Marketplace)
* Specify a DS1\_V2 sized virtual machine
* Create a fully qualified domain name for the public ip address of the virtual machine

#### Exit criteria

Jenkins deployed in Azure

### Task 2: Post-Deployment Configuration of Jenkins Server

#### Tasks to complete

* Patch the virtual machine hosting Jenkins and install the latest version of Jenkins
* Install 7zip on the virtual machine hosting Jenkins
* Create jenkinsadmin user account
* Install FTP publisher plugin

#### Exit criteria

Jenkins patched with 7zip and FTP publisher plugin installed.

### Task 3: Configure Jenkins Staging Deployment

#### Tasks to complete

* Create a Jenkins project that is connected to your GitHub instance for source
* Leverage the FTP publisher plugin to push code to the staging slot of your web application

#### Exit criteria

A functioning version of the website on the Staging Slot.

### Task 4: Configure your GitHub Repo to Notify Jenkins of Changes

#### Tasks to complete

* From GitHub, configure the Webhook back to your Jenkins instance

#### Exit criteria

GitHub notifies Jenkins when any code has been checked in.

### Task 5: Check in a Change to Trigger Jenkins Job

#### Tasks to complete

* Update the source for the home page
* Commit the change to Git and push to GitHub

#### Exit criteria

Change should automatically be pushed to Staging Slot of the web application

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

#### Tasks to complete

* Create a folder in Dropbox and copy three images to it (named banner-1.jpg, banner-2.jpg, banner-3.jpg)
* Share out the folder by creating a link in Dropbox

#### Exit criteria

Shared folder with three images that can be accessed by a URL.

### Task 7: Manually Deploy to Production

#### Tasks to complete

* Within the Staging Slot of the application, Swap to production

#### Exit criteria

Production website with latest code from Staging

# OSS DevOps Hackathon Answers

## Overview

This portion of the lab is designed to help you if you are blocked or have limited experience with Azure Resource Manager.

## Exercise 1: Environment Setup

### Overview

In this Exercise, you will configure a lab machine on a Mac or a Windows Virtual Machine in Azure that will be used to complete the exercises. The option to configure a Virtual Machine in Azure is useful in cases where you are using a computer you are not allowed to alter or install software on.

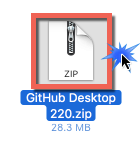
### Prerequisites

* Mac OSX 10.9 or later, 64-bit processor (*Mac only*)
* Microsoft Azure Subscription: <http://azure.microsoft.com/en-us/pricing/free-trial/>
* Visual Studio Code
* GitHub Account and Client
* Dropbox Account

### Option 1: Configure a Mac as your lab machine

#### Task 1: Install GitHub Desktop

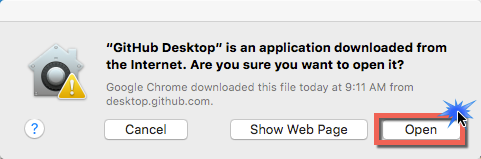
1. Open a web browser and navigate to <http://desktop.github.com>
2. Click on the button to Download GitHub Desktop for OSX 10.9 or Later.
3. This will download a file called GitHub Desktop 220.zip to your **Downloads** folder (later versions may change the number in the file name). Double-click this file.

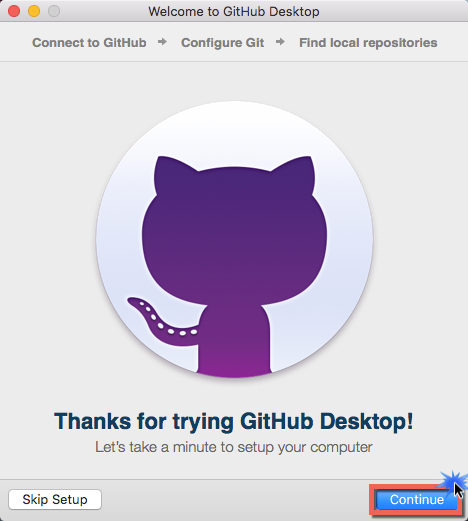


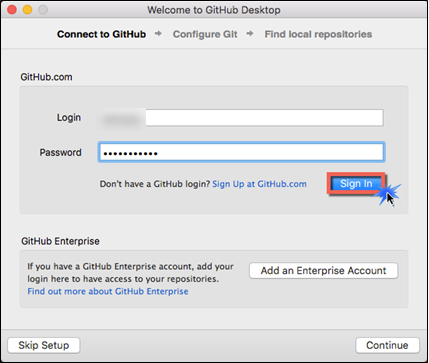
1. This will unzip the GitHub Desktop Application. Drag the profile to your **Applications** Folder.



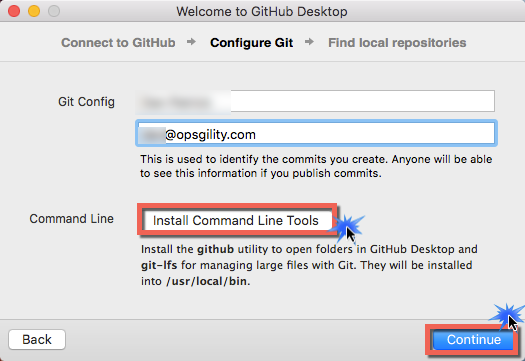
1. Navigate to the **Applications** on your Mac and then Double-click the GitHub Desktop icon.
2. You will receive a warning: “GitHub Desktop is an application downloaded from the Internet. Are you sure you want to open it? Click **Open**.



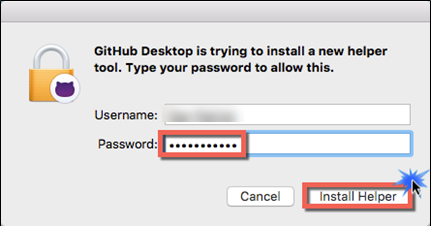
1. The Welcome screen will appear for GitHub Desktop. Click **Continue.**
2. On the next screen, you need to connect to your GitHub Account. If you don’t have an account, it will need to be created at: <https://github.com/join?source=header-home>



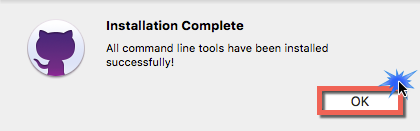
1. Next configure Git with your name and email address. Then click on the **Install Command Line Tools** button.Next click **Continue**.



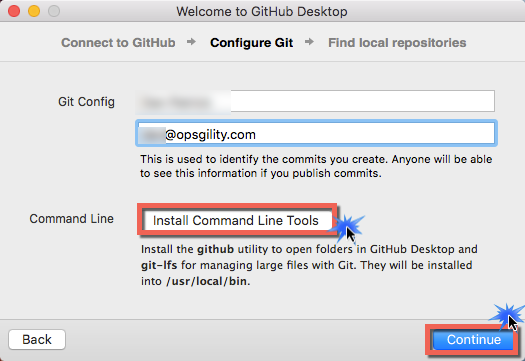
1. You will be prompted by OSX to authenticate. Enter the user name and password of an administrator account on your Mac.



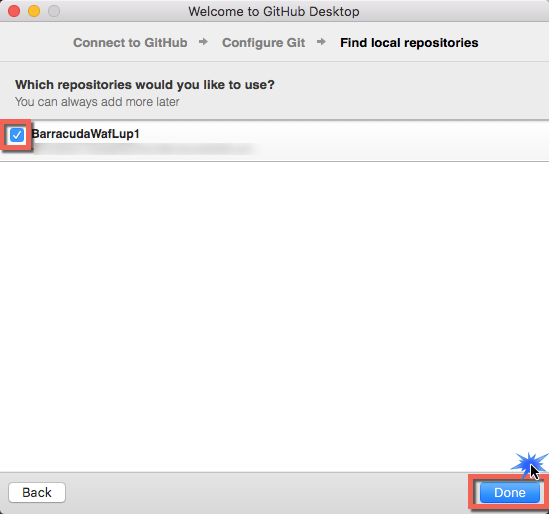
1. An installation complete message will pop-up on the screen click **Ok**.



1. You will again need to click Continue on this window.



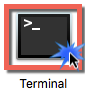
1. Next you will be asked if you want to use any local repositories that you may have had on your machine already. Click any that you wish to use with GitHub Desktop. Click **Done** after you have made your selections.



1. The GitHub Desktop will load into a tutorial. Complete the tutorial, so that you understand how to use GitHub Desktop.



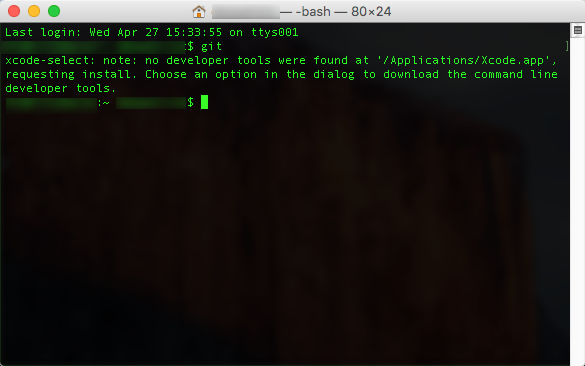
1. Open a Terminal on your Mac Applications🡪Utilities🡪Terminal.

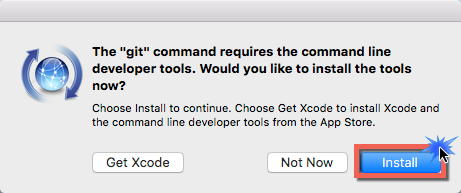


1. In the command window type the following command

git <enter>

1. This will give a warning no developer tools were founds on the machine as seen below and will prompt for an installation of the “git” command. Click i**nstall**.

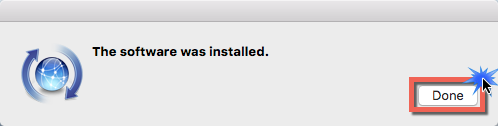




1. Click Agree and then your Mac will download and Install the software.

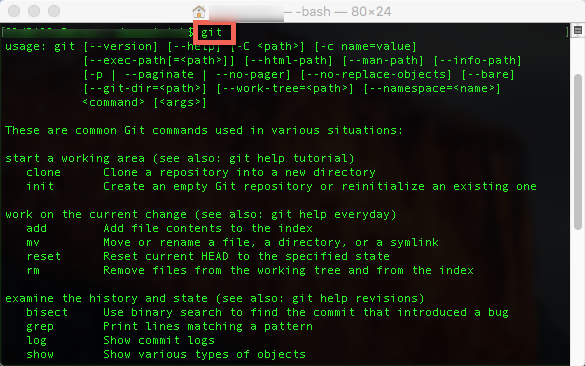


1. A message will then be providing that the Software was Installed. Click Done.



1. In the command window type the following command

git <enter>



#### Task 2: Create a Dropbox account (if you don’t already have one)

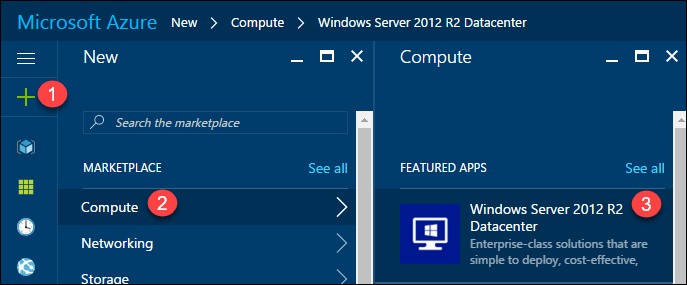
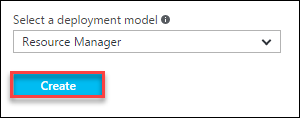
1. If you don’t already have a Dropbox account, create one at the following URL <https://www.dropbox.com/team/join/new_pair> - it will be leveraged in one of the exercises that follow.

### Option 2: Create/Configure a Windows VM as your lab machine

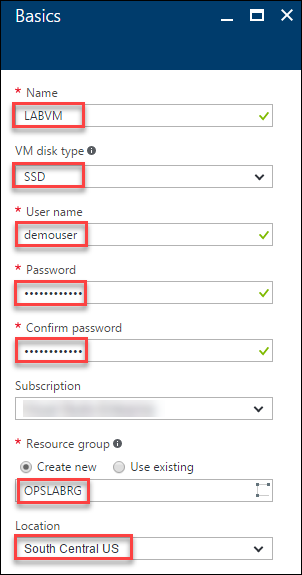
#### Task 1: Create Windows Virtual Machine in Azure

1. Launch a browser and navigate to <https://portal.azure.com>. Once prompted, login with your Microsoft Azure credentials. If prompted, choose whether your account is an organization account or just a Microsoft Account.

**Note:** You may need to launch an "in-private" session in your browser if you have multiple Microsoft Accounts.

1. Click on **+NEW > Compute > Windows Server 2012 R2 Datacenter.** Select **Resource Manager** as the deployment model and click **Create**.   
     
   
2. Set the following configuration on the **Basics** tab and click **OK**.

* Name: **LABVM**
* VM disk type: **SSD**
* User name: **demouser**
* Password: **Demo@pass123**
* Subscription: If you have multiple subscriptions choose the subscription to execute your labs in.
* Resource Group: **OPSLABRG**
* Location: Choose the closest Azure region to you.



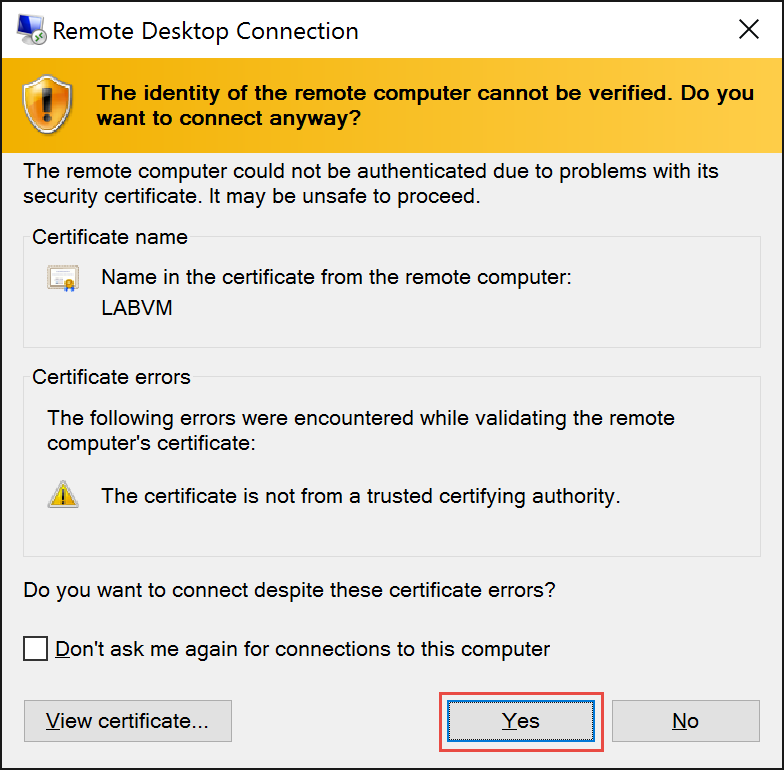
Choose the **DS1\_V2 Standard** instance size on the Size blade. If DS1 Standard is not avaialable choose **D1\_V2 Standard**.

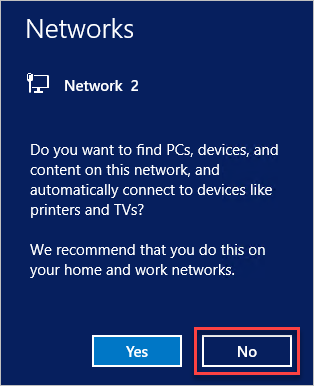
Note: You may have to click the View All link to see the instance sizes.



**Note:** We could use one of the recommended configurations, but if you are using a trial Azure subscription there is a restriction of 4 cores per region. Hence we are choosing a lower configuration. **Make sure to not chose a VM from a different family. Use the DS1\_V2 as directed.**

1. Accept the default values on the **Settings** blade and continue clicking **OK**, and then **OK** on the **Summary** blade.
2. It may take 10+ minutes for the virtual machine to complete provisioning.  
   
3. After the virtual machine is created, click the Connect button on the toolbar.   
   
4. Depending on your remote desktop protocol client and browser configuration you will either be prompted to open an RDP file or you will need to download it and then open it separately to connect.
5. Login with the credentials specified during creation:
   1. User: **demouser**
   2. Password: **Demo@pass123**
6. You will be presented with a Remote Desktop Connection warning because of a certificate trust issue. Click **Yes** to continue with the connection.



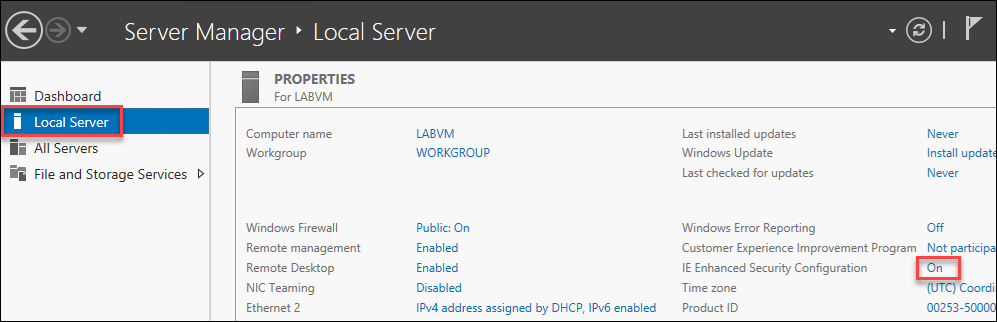
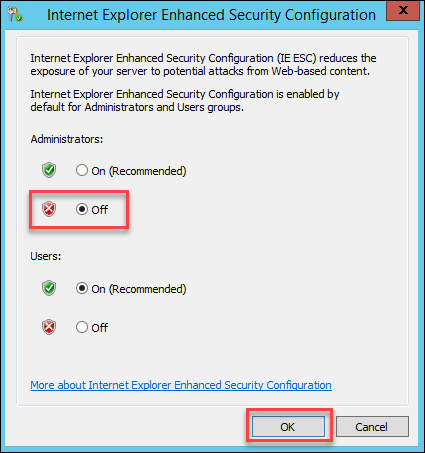
1. If presented with a network prompt, click **No**

#### Task 2: Disable IE Enhanced Security

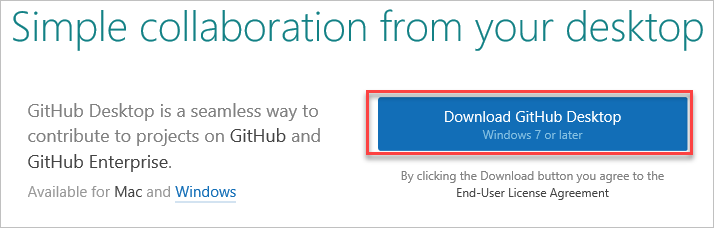
1. If the Server Manager does not launch on its own, click the Server Manager icon on the toolbar.

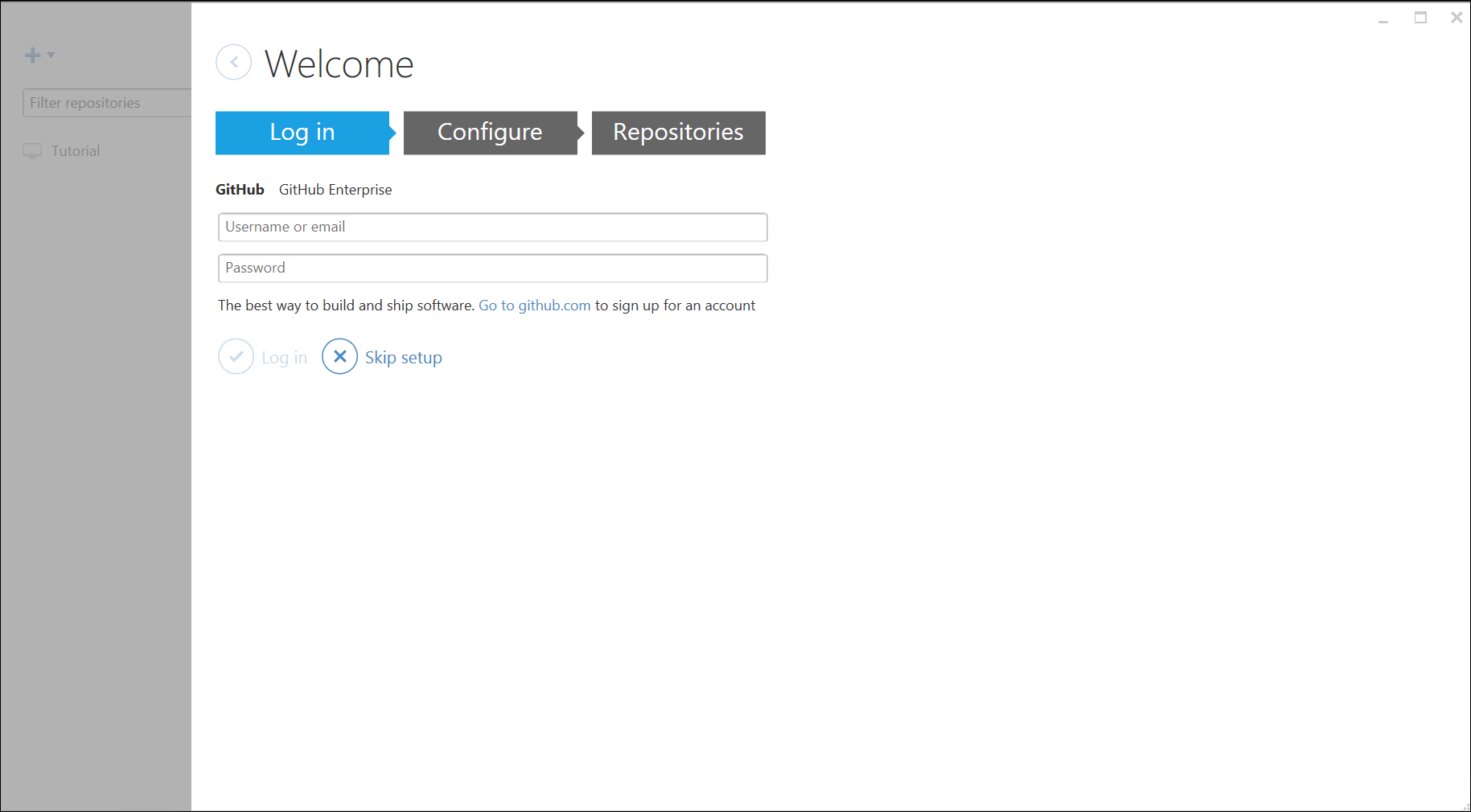


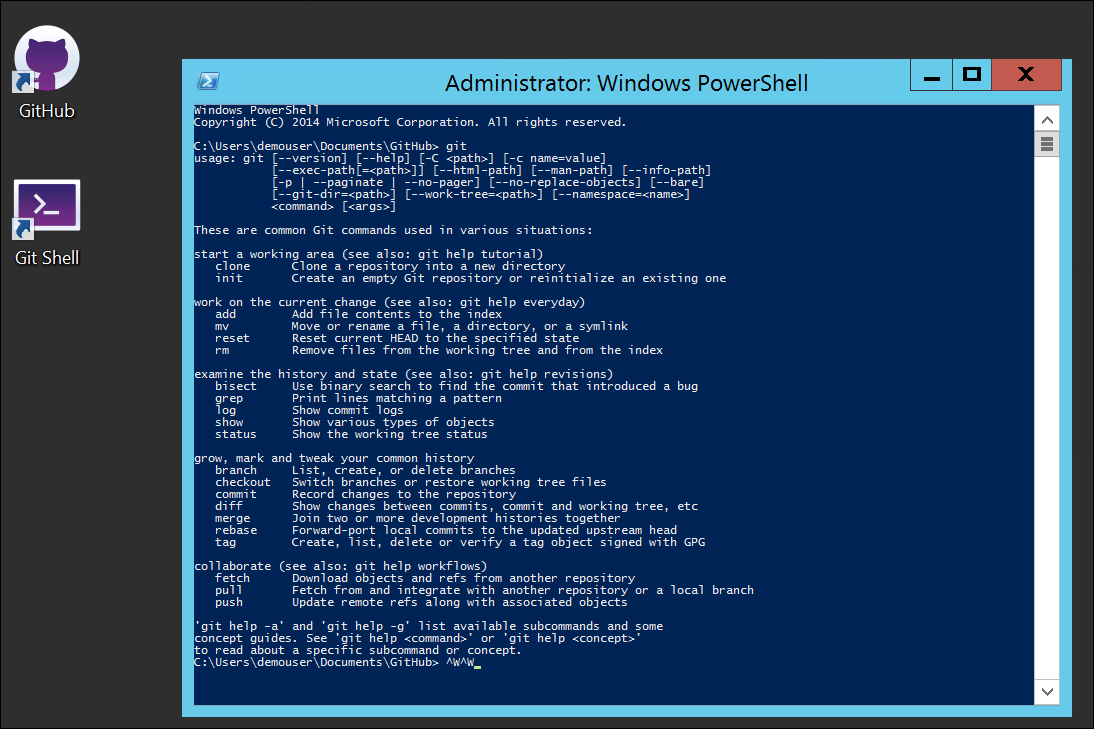
1. Click **Local Server** on the left menu area, and then click the “**On**” link next to **IE Enhanced Security Configuration**. On the subsequent pop-up, choose **Off** for Administrators and click **OK**.

#### Task 3: Install GitHub Desktop

1. Launch a browser and navigate to <http://desktop.github.com>
2. Click on the **Download GitHub Desktop** button and follow the prompts for install  
   
3. Once the installation is complete, you will need to configure the GitHub Desktop. If you do not have an account, you will need to create one. A personal account is free and can be created here: <https://github.com/join>.
4. Enter your login information in the configuration screen.

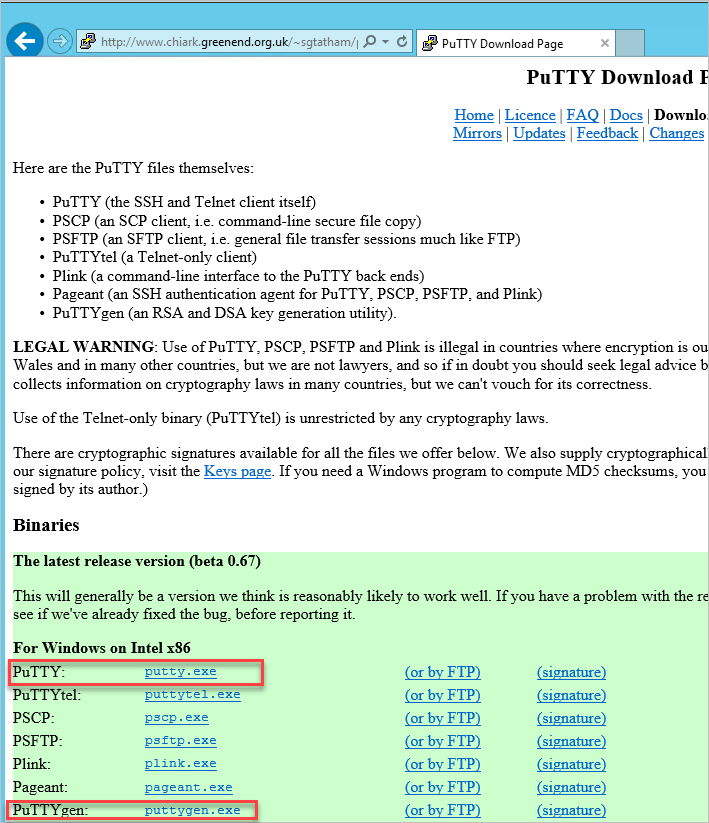


1. After logging in, you can close out this screen.
2. You will see the GitHub and the Git Shell shortcuts on the desktop of your LABVM. Launch the **Git Shell** and type **git** to make sure the commands are available. The output should be like the screen shot below.  
   

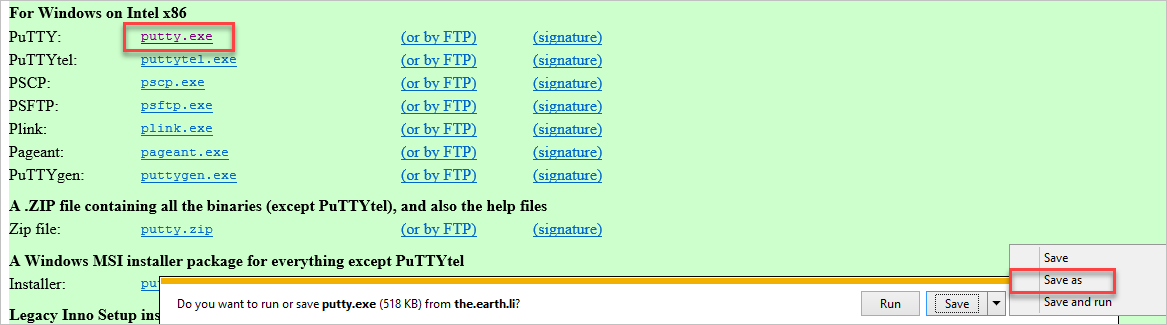
#### Task 4: Download Putty and PuttyGen

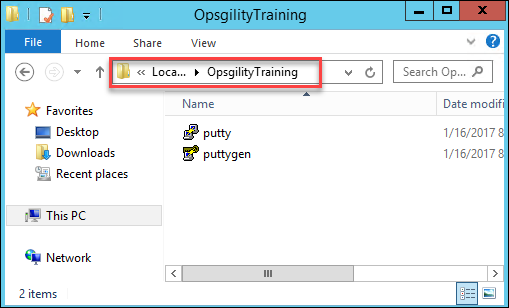
In this exercise, you will download two tools needed for SSH connections in the labs that follow.

1. Create a folder at the root of your **C:\** drive named **OpsgilityTrianing**
2. Navigate to <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
3. From this page, you will download **putty.exe** and **puttygen.exe** to your **c:\OpsgilityTraining folder**



1. For PuTTY, click the link for **putty.exe** and chose Save as and browse to **c:\OpsgilityTraining**



1. Do the same for the **puttygen.exe** link.

#### Task 5: Create a Dropbox account (if you don’t already have one)

1. If you don’t already have a Dropbox account, create one at the following URL <https://www.dropbox.com/team/join/new_pair> - it will be leveraged in one of the exercises that follow.

### Summary

In this lab, you configured a machine (Windows or Mac) that will be used to complete the exercises in this course.

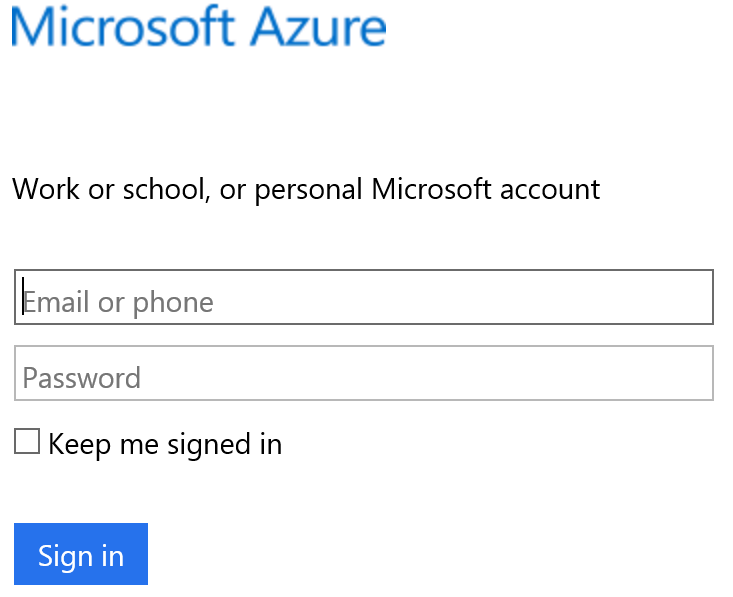
## Exercise 2: Configure an Azure Web App

### Overview

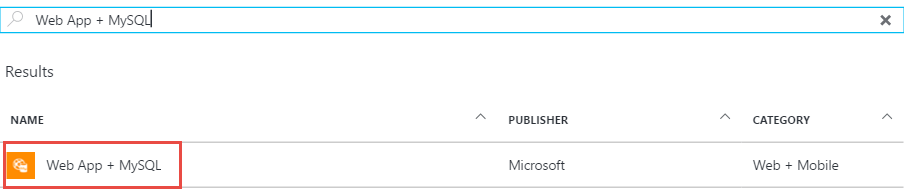
You will use several Azure Platform as a Service (PaaS) components to configure a web application. This web app will use Azure Media Services and Storage as well as MySQL for data. The actual deployment of the web application will happen in the next exercises.

### Task 1: Create the Azure Web App

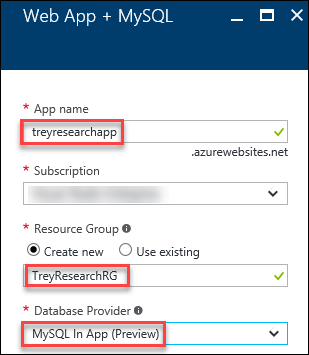
1. If you are leveraging a virtual machine in Azure as your lab machine, first connect to it via RDP, otherwise, browse to <https://portal.azure.com> and authenticate with your Organization or Microsoft Account.



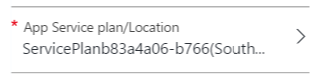
1. Click **New**, and in the search box type: **Web App + MySQL** and press **enter.** 
2. Select the **Web App + MySQL** result, and then click **Create** on the next blade.



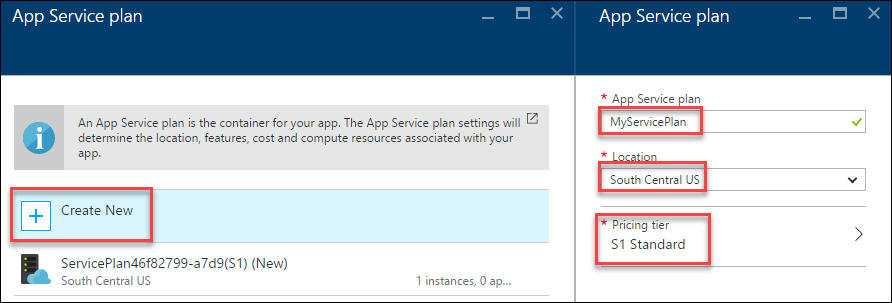
1. Specify the following on the **Web App + MySQL** blade
   * App name: **<unique name *ex: treyresearchapp*>**
   * Resource group: **TreyResearchRG**
   * Database Provider: **MySQL In App (Preview)**



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

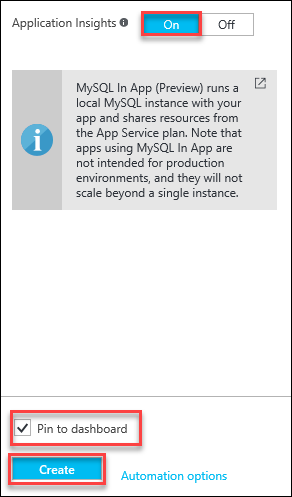


1. Click **+Create New**, and specify the App Service Plan as **MyServicePlan** and choose the region closest to you, and then click **OK**.



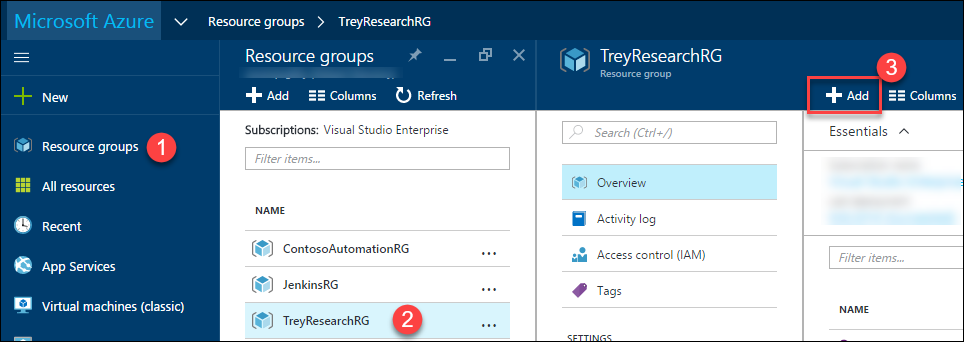
NOTE: If you receive an error that the RG failed to deploy because there is no credit card found associated with the account, choose the FREE option in the Pricing Tier.

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

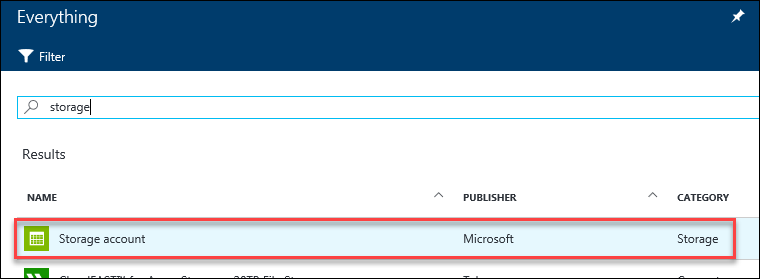


### Task 2: Create an Azure Storage Account

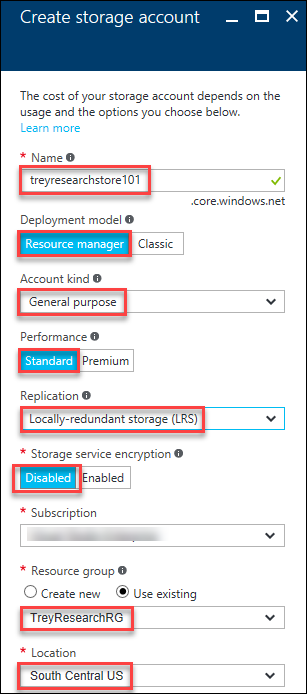
1. Click **Resource groups** **> TreyResearchRG** and then click **+Add**.



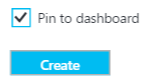
1. Type **storage** in the search bar and hit **enter** and choose **Storage account** and click **Create**.



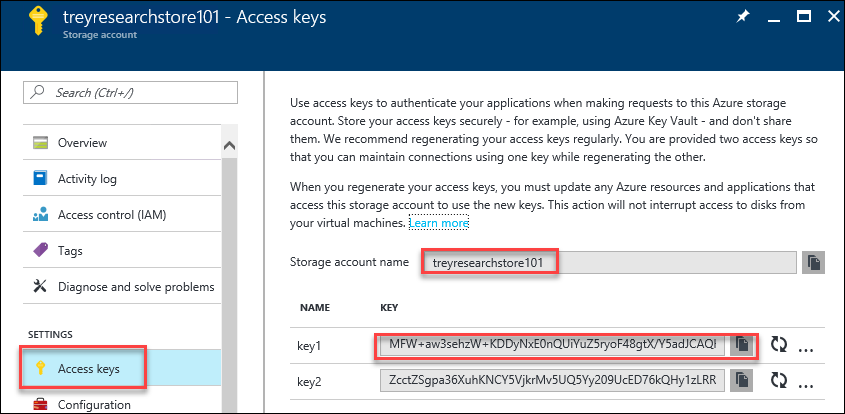
1. Specify a unique name for the storage account. Ensure the green checkmark is displayed.



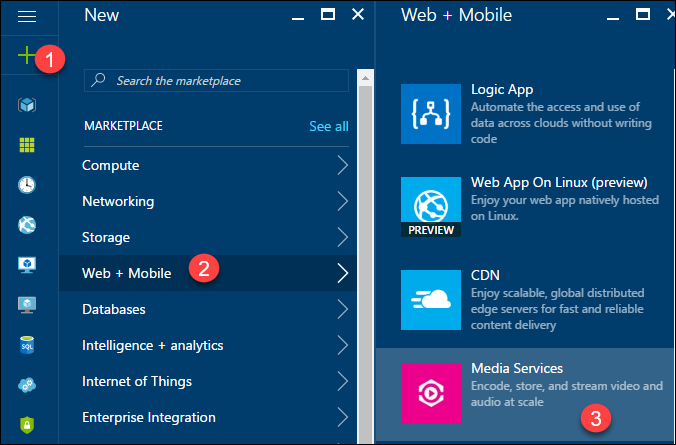
1. Check the check box next to **Pin to dashboard** and click **Create**.

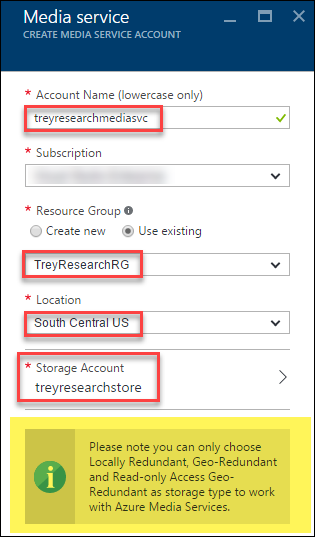


1. After the storage account has been provisioned, click **Access keys** and then copy the **Storage account name**, and the value for **key1** to a text editor for temporary storage.

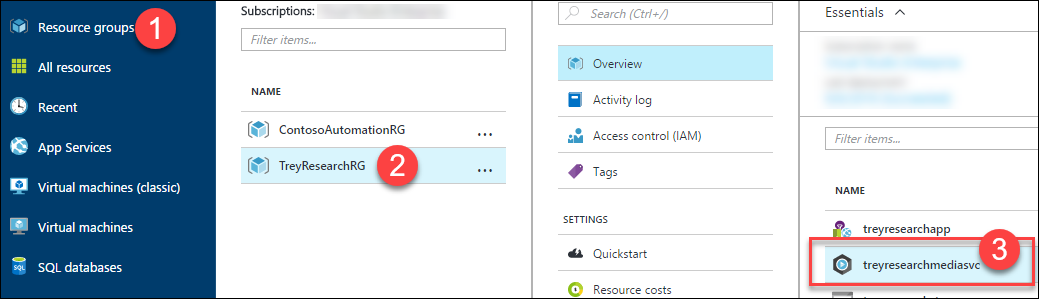


### Task 3: Create an Azure Media Service Instance

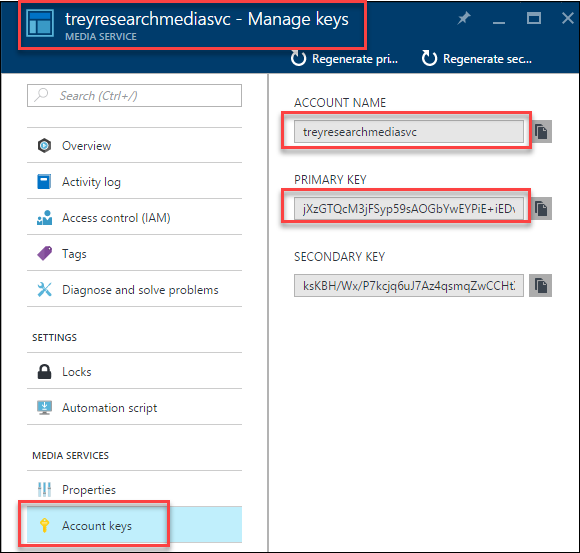
1. Open your browser and navigate to <https://portal.azure.com>. Click +**New** > **Web + Mobile**, and then select **Media Services**. 
2. Specify the following in the **Media service** blade and click **Create**.
   * Account Name: **<uniquename *ex: treyresearchmediasvc*>**
   * Resource group: **TreyResearchRG**
   * Location: **<location nearest you>**
   * Storage Account: **treyresearchstore (or whatever name you used in earlier)**



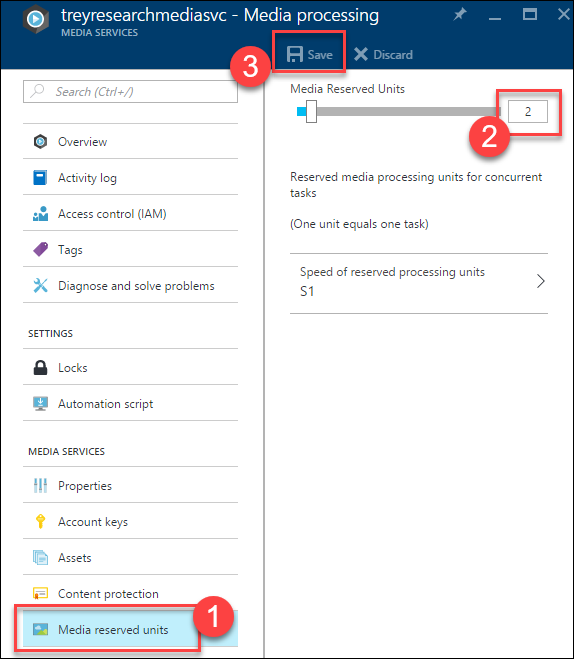
1. After the Media service is created, you need to obtain the **Account Keys** of the service. Click **Resource groups > TreyResearchRG > treyresearchmediasvc**



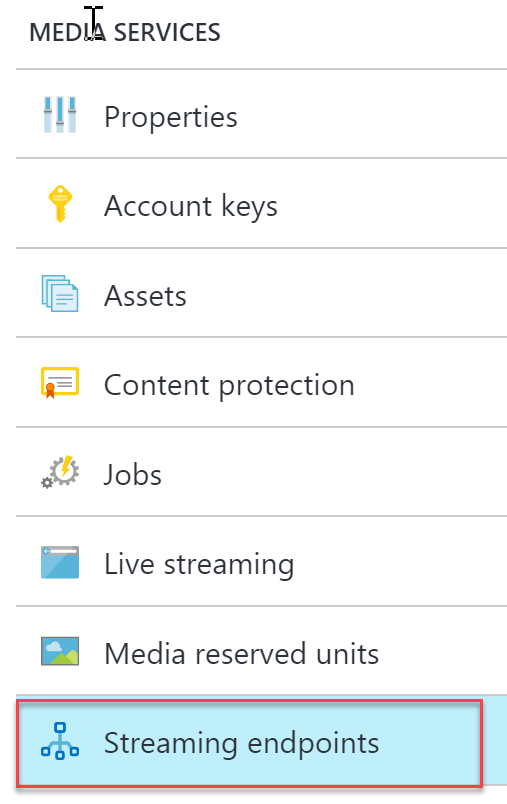
1. Copy the name of the media service account and the access key and save them in a text editor for future reference.



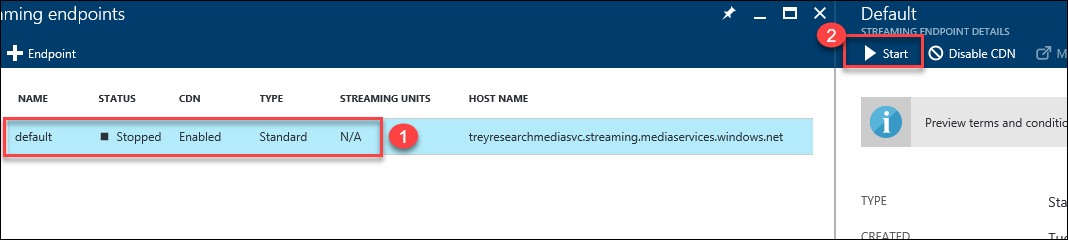
1. Click **Media reserved units** on the Settings blade, change the value to **2** and click **Save**.

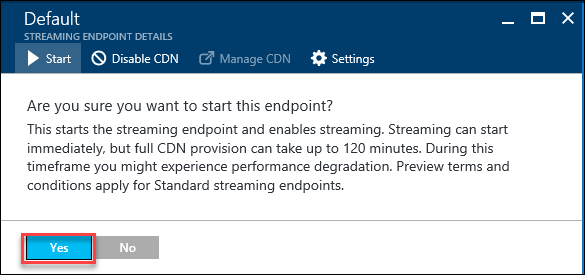


1. Next you will start the Streaming Endpoint to the Media Service instance. Streaming Endpoints allow streaming video from Azure Media Services. Click on **Streaming endpoints** on the left navigation.



1. Click the **default** Endpoint that has been created, and then click **Start**.



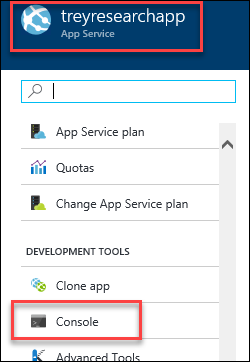


Note: It make take up to two hours for the streaming endpoint to start and until then any uploaded videos will not render. Rendering videos is not a required step in this lab.

### Task 4: Obtain Connection String of Local DB

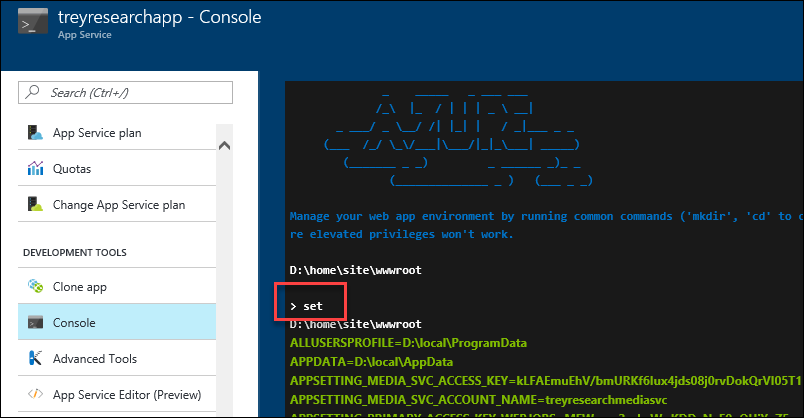
Since you leveraged the In-App version of MySQL, you need to get the connection string so that we can set it as one of the application settings.

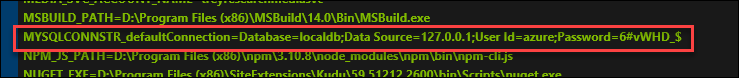
1. Scroll down and click Console to launch an Azure command line console.



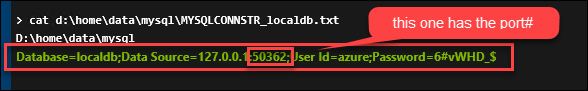
1. Type “set” and hit enter to see the list of environment variables. The variable you will need is **MYSQLCONNSTR\_defaultConnection**.

NOTE: In-App MySQL is still in preview and the behavior is still changing. At the time of the writing of this lab, the environment variable did not contain the port number needed in the connection string. If you run into the same behavior (environment variable with no port defined), follow the instructions in step 3.





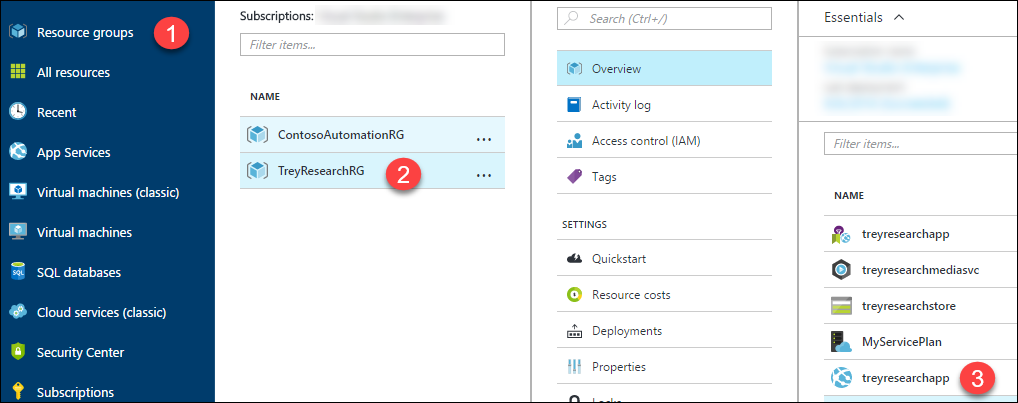
1. As mentioned in the note above, the **MYSQLCONNSTR\_defaultConnection** should contain a port number, if it does not, you will obtain it by opening the **MYSQLCONNSTR\_localdb.txt** text file.

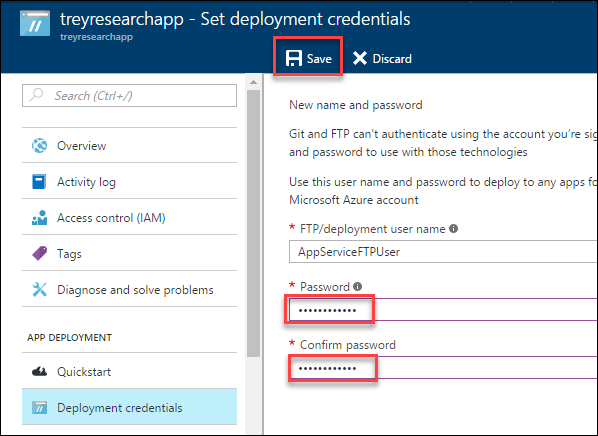


1. Save off the connection string into a text editor for reference in an upcoming Task.

### Task 5: Configure FTP Deployment Credentials

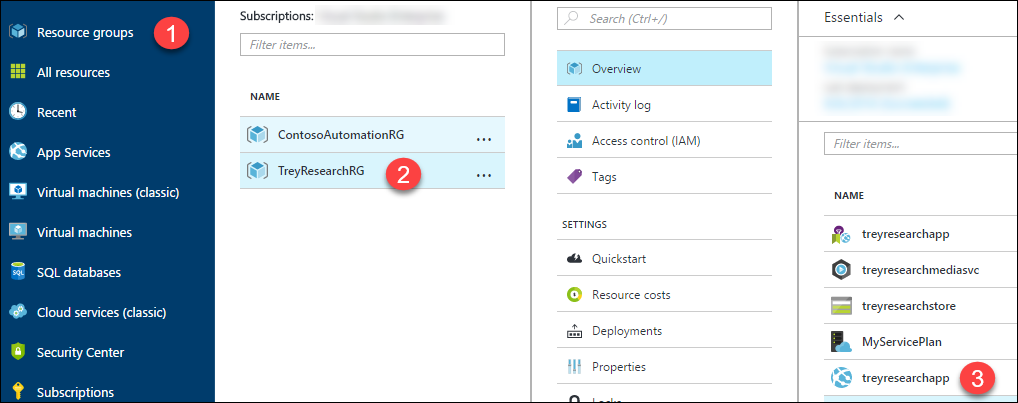
Since you will be leveraging Jenkins to deploy the source code, you must first update the credentials that are used for an FTP deployment.

1. In the Azure portal, click **Resource groups > TreyResearchRG > treresearchapp** to open the settings of the App Service
2. Click Deployment credentials, specify a password (**Demo@pass123**) and click **Save**.

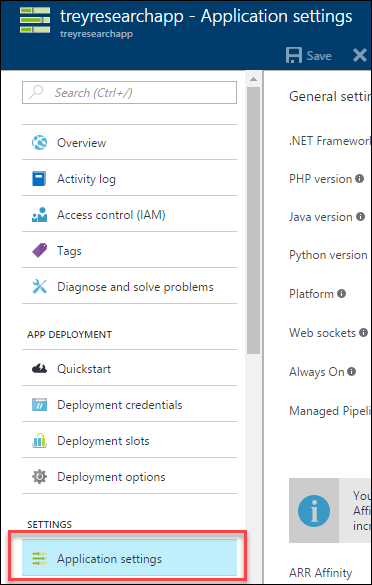


### Task 6: Update and Review the Web App Settings

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

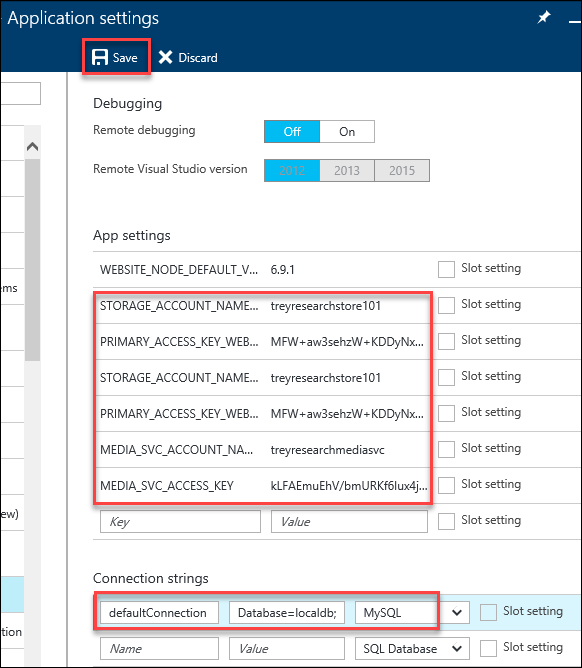


1. Click the **Application Settings** link.



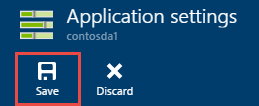
1. Within App settings, add six new entries using the app settings names below along with the storage account name, key, and media services account and key. For the **Connection strings**, add an entry with the Name of **defaultConnection**, Value of the connection string that was in MYHSQLCONNSTR\_localdb.txt or environment variable (***obtained in the previous task***)and then click **Save**.

|  |  |
| --- | --- |
| STORAGE\_ACCOUNT\_NAME\_WEBSITE | Storage Account Name |
| PRIMARY\_ACCESS\_KEY\_WEBSITE | Storage Account Key |
| STORAGE\_ACCOUNT\_NAME\_WEBJOBS | Storage Account Name |
| PRIMARY\_ACCESS\_KEY\_WEBJOBS | Storage Account Key |
| MEDIA\_SVC\_ACCOUNT\_NAME | Media Service Account |
| MEDIA\_SVC\_ACCESS\_KEY | Media Service Key |



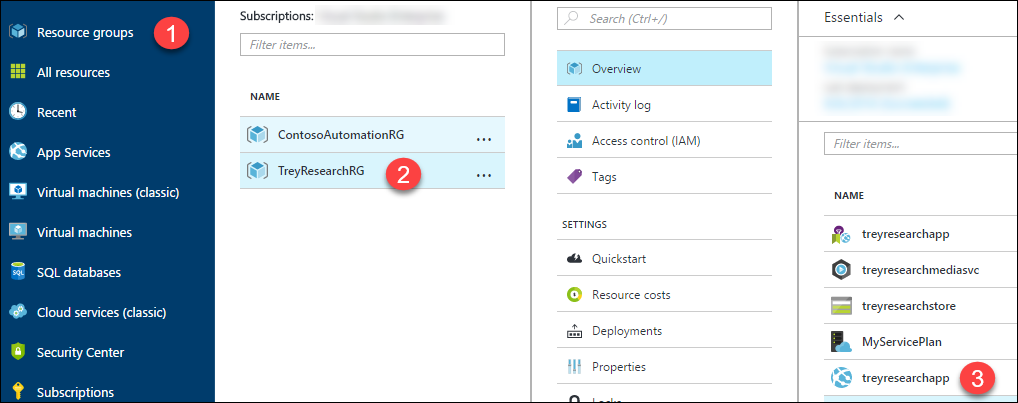
1. On the same blade, change the **PHP version to 5.5**, and click **SAVE** on the toolbar.



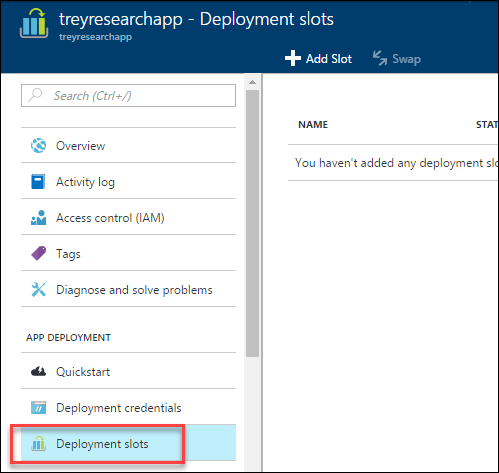


### Task 7: Configure a Staging Slot

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



1. Click **Deployment slots** under the **APP DEPLOYMENT** category.

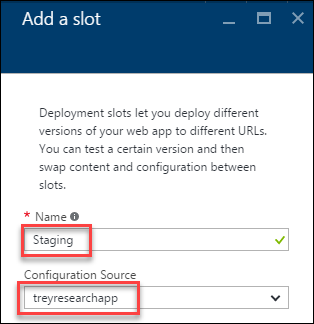


NOTE: If you deployed with the Free plan, you can now choose the S1 Standard plan and it will upgrade to allow for Deployment Slots to be created and utilized.

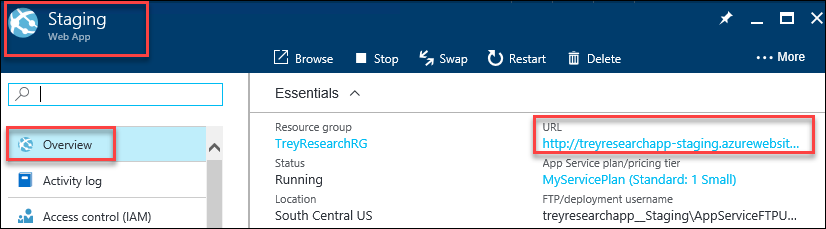
1. Click **Add Slot**.



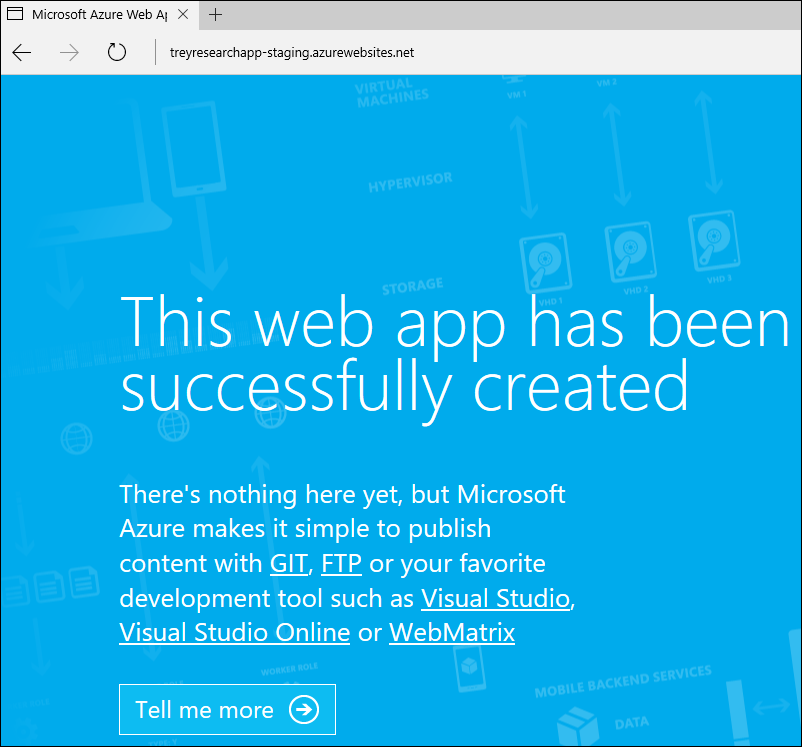
1. Name the slot **Staging**, and specify the primary site as the **Configuration Source** (this will copy over the 6 variables and their values that we defined previously as well as the connection string), click **OK** to create the deployment slot.



1. Once the staging slot has been created, click its name. Click the **URL** from the staging slot’s essentials pane.



1. At this time, no code has been deployed to either production or the staging slot that we just created. Both URLs will have the default web site 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 Media Services and 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 3: Configure Local Git Repository

### Overview

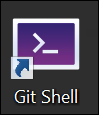
In this exercise, you will fork a github repository and clone it locally so that you can configure your web app.

### 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.



1. After the fork is complete, clone the site locally on your computer for future changes by executing the commands below within Git Shell (installed at the beginning of the labs)

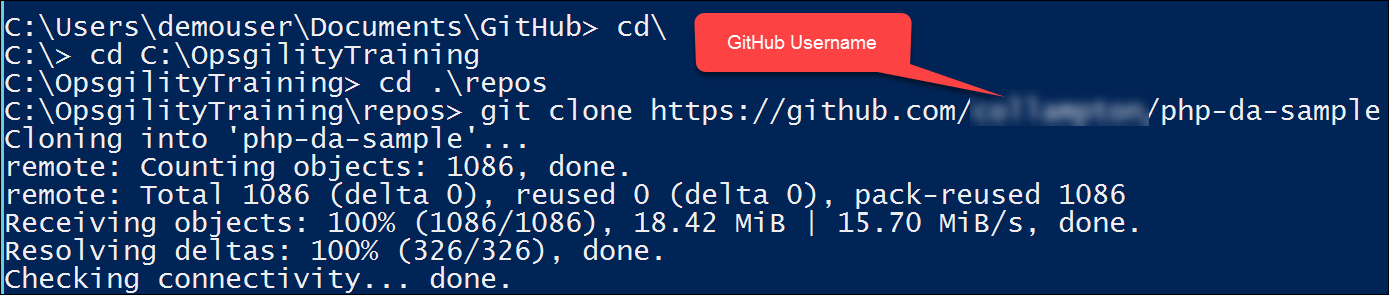


cd c:\OpsgilityTraining

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 that you can configure your web app.

## Exercise 4: Configure Git and Jenkins for Continuous Integration, Delivery and Deployment

### Overview

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).

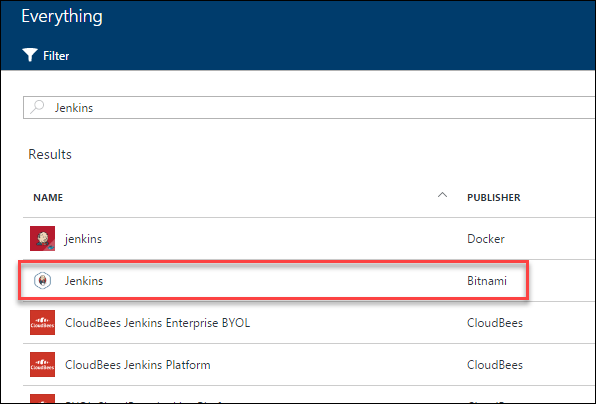
### Task 1: Deploy a Jenkins Server in Azure

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.

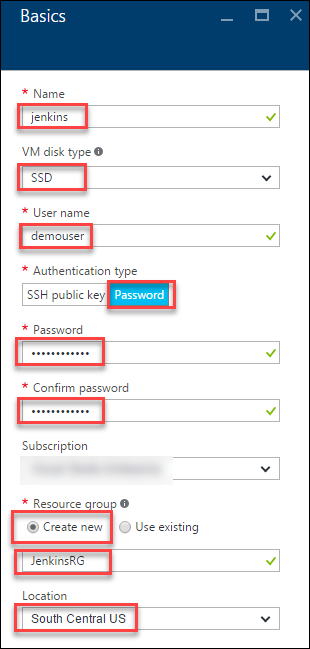
1. If you are leveraging a virtual machine in Azure as your lab machine, first connect to it via RDP, otherwise to deploy a Jenkins Server instance in Azure browse to <https://portal.azure.com>. Click **+New** and type **Jenkins** in the search box and hit enter



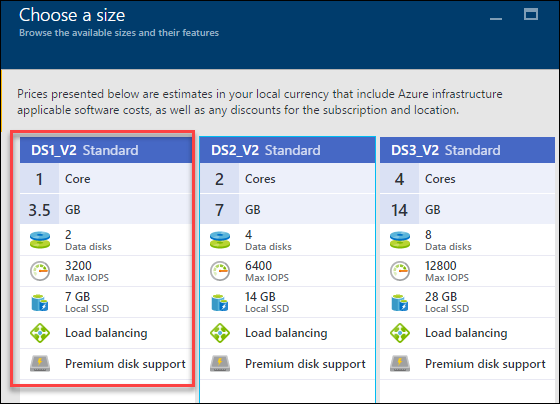
1. You will notice that there are numerous preconfigured Jenkins servers available in the marketplace, for the purposes of the labs, choose the one submitted by **Bitnami** and click **Create.**



1. Specify the following on the **Basics** blade and click OK
   * Name: **jenkins**
   * VM disk type: **SSD**
   * User name: **demouser**
   * Password: **Demo@pass123**
   * Resource group: **JenkinsRG**
   * Location: **location nearest you**



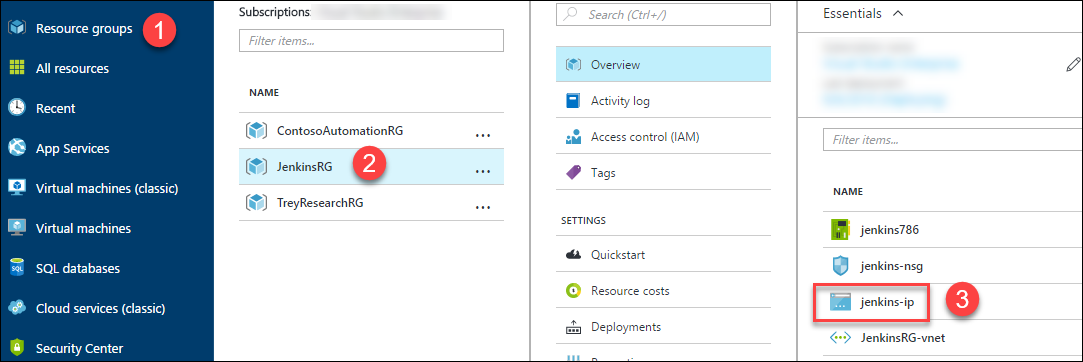
1. Choose a size for your virtual machine (**DS1\_V2 Standard** if available in your region – choose a comparable one if **DS1\_V2** is not available) and click **Select**



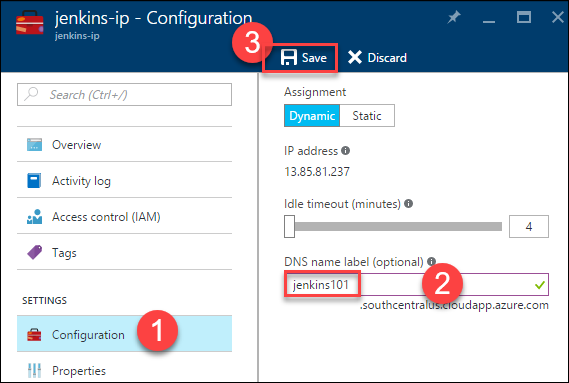
1. Leave the default settings and click **OK** on the **Settings** blade.
2. Click **OK** on the **Summary** blade and **Purchase** on the **Purchase** blade to start the provisioning of your Jenkins server



1. Once the deployment has completed, configure a fully qualified domain name for your Jenkins server by clicking **Resource groups > JenkinsRG > Jenkins-ip** (Public IP address)



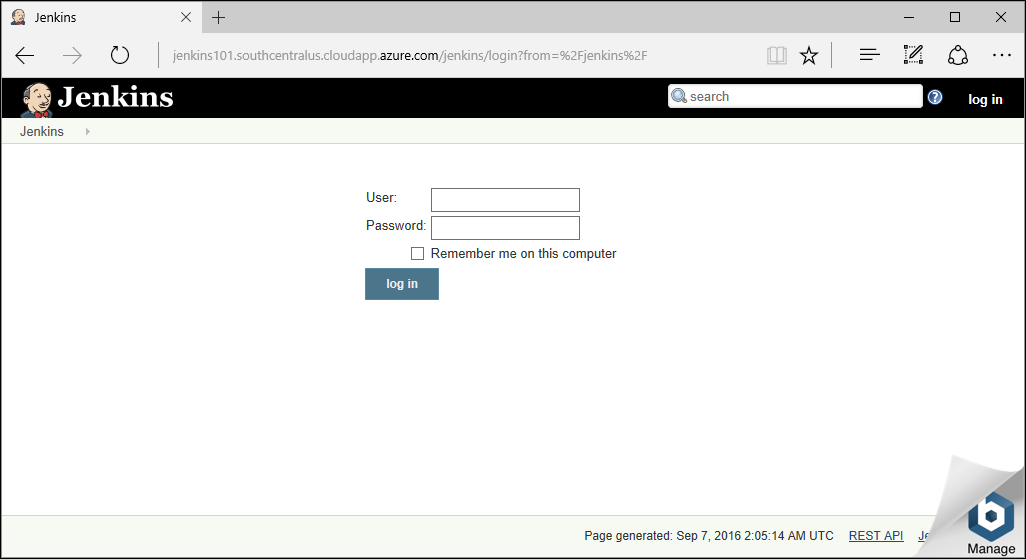
1. Click **Configuration** on the **Settings** tab of the Public IP address and specify a **unique DNS name** for your server and click **Save**



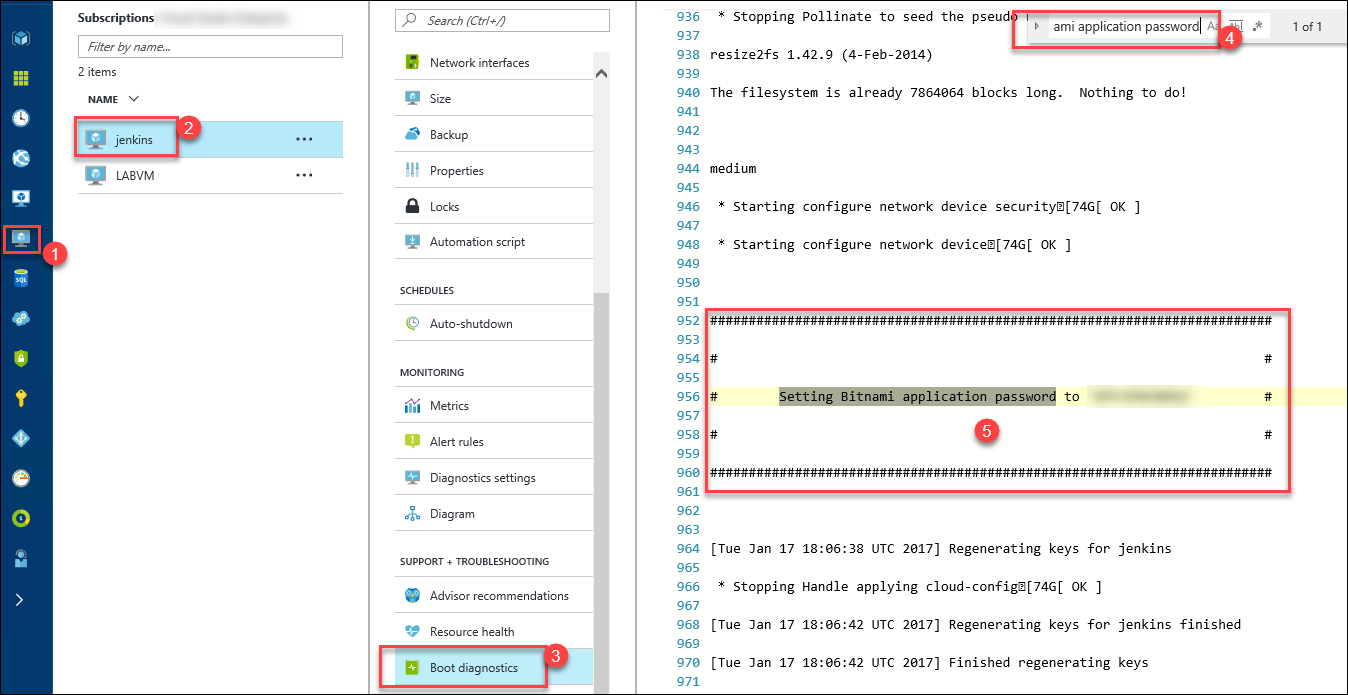
### Task 2: Post-Deployment Configuration of Jenkins Server

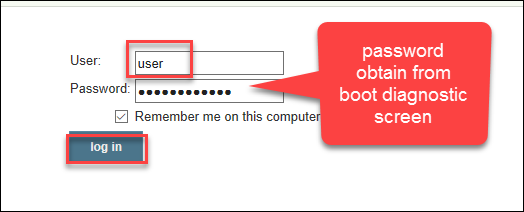
Log into the Jenkins portal to confirm it was successfully deployed. After that, you will update the OS and Jenkins to the latest as well as install other tools needed for our scenario.

1. Browse to your Jenkins portal, using the FQDN you defined in the previous Exercise.

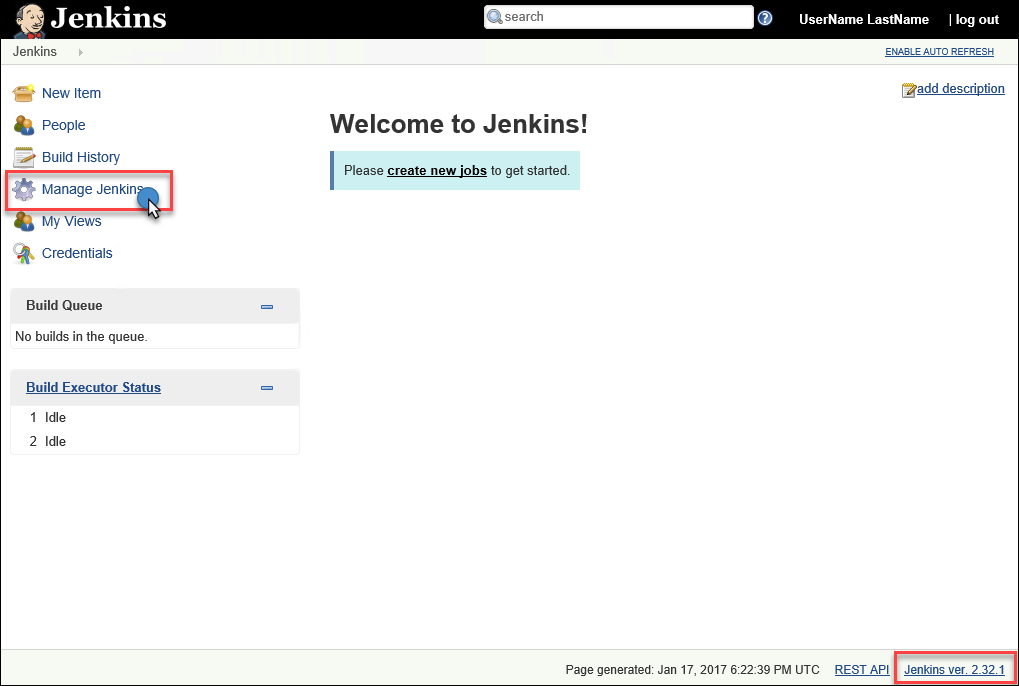


1. The default username is **user** and the password has to be retrieved by looking at the Boot Diagnostics of the Jenkins virtual machine. Click **Virtual machine > Jenkins > Boot diagnostics** and search for the string **Setting Bitnami application password**





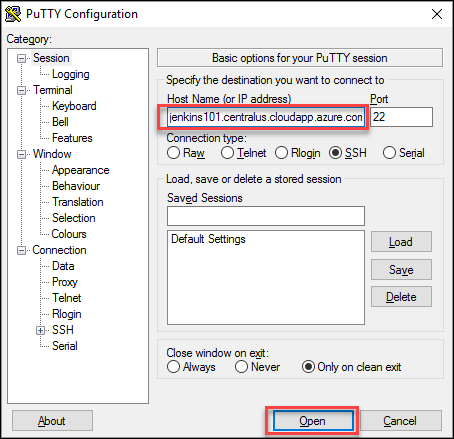
1. Once logged into the portal, notice the version of Jenkins (**2.32.1** as of the writing of this lab). Click **Manage Jenkins** for more details

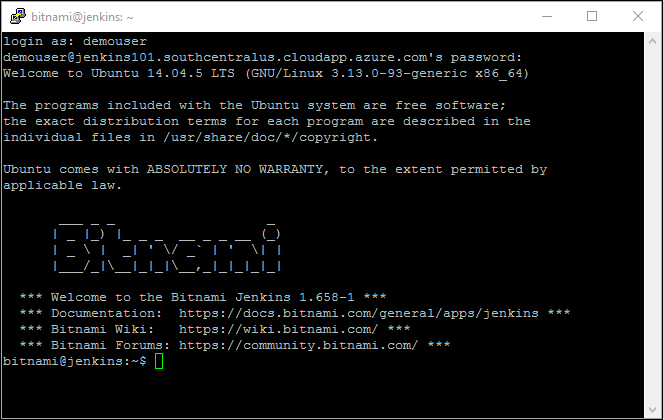


1. Here you will get more information on any update that may be available for the server instance itself or for Plugins.
2. To update Jenkins to the latest version, you will need to connect to Jenkins via SSH. On a Mac, you can leverage Terminal and the following command.

ssh [demouser@jenkins101.centralus.cloudapp.azure.com](mailto:demouser@jenkins101.centralus.cloudapp.azure.com)

For Windows users, you can leverage PuTTY. Authenticate with **demouser** and **Demo@pass123** after establising a connection.





1. Run the following command to elevate your user and update the packages of the OS.

sudo su

apt-get update

sudo apt-get install p7zip-full

1. Run the following command to download the latest version of Jenkins

wget <https://updates.jenkins-ci.org/latest/jenkins.war>

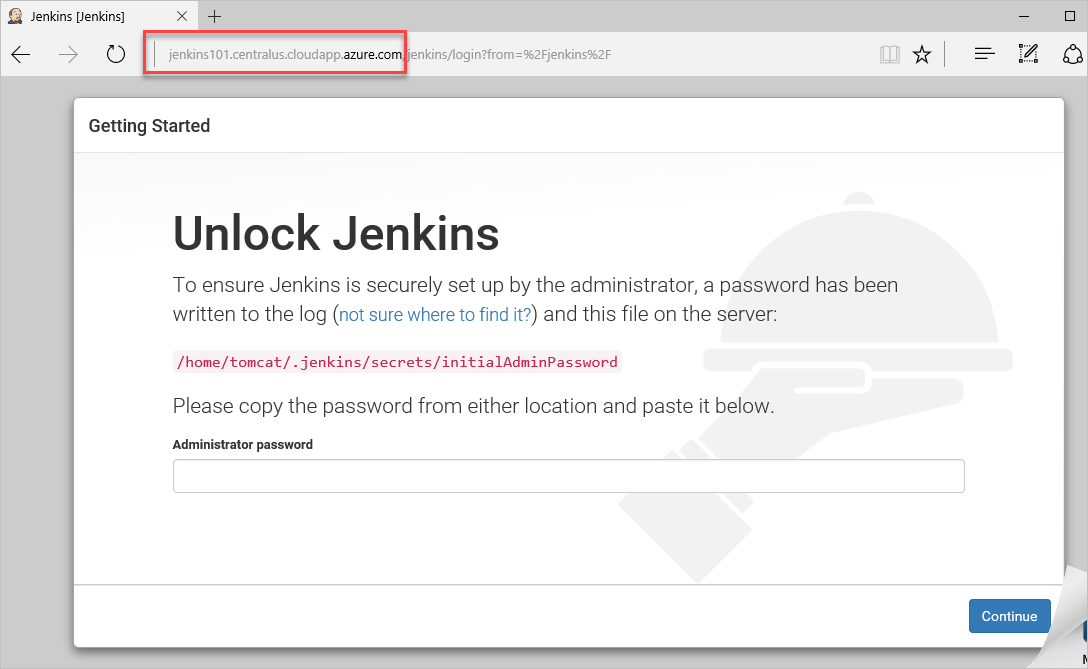
1. The following commands will clean up a folder and file before copying the latest version of **jenkins.war** to the directory it needs to reside in.

rm -r /opt/bitnami/apache-tomcat/webapps/jenkins

rm -r /opt/bitnami/apache-tomcat/webapps/jenkins.war

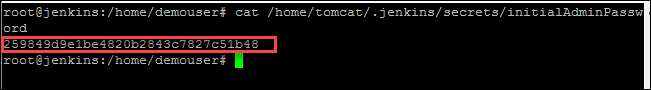
mv jenkins.war /opt/bitnami/apache-tomcat/webapps/

1. Once again browse to the URL of your Jenkins portal (if you already have a browser connection to the portal, close it and re-browse to it). Because of the updates you made during the SSH tasks, you will be prompted for the initialAdminPassword

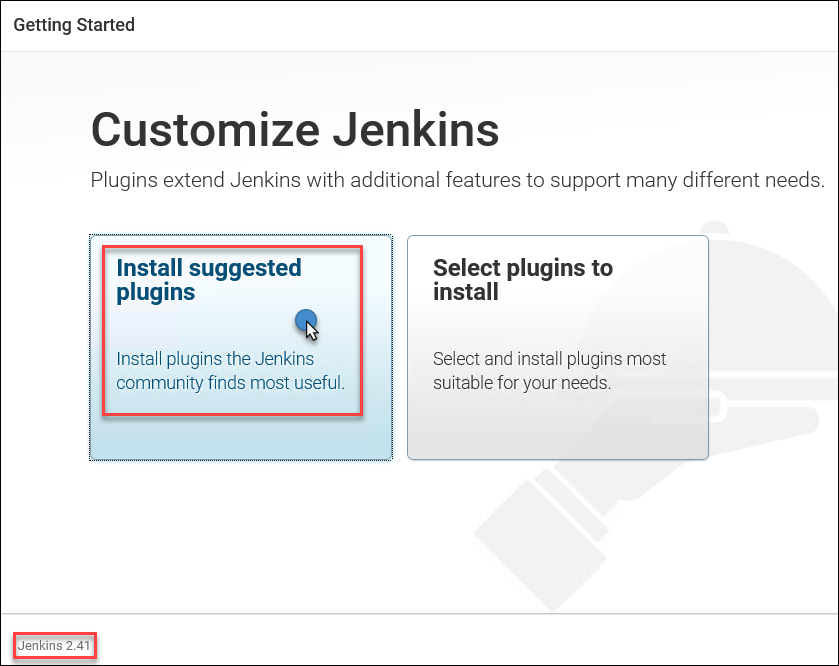


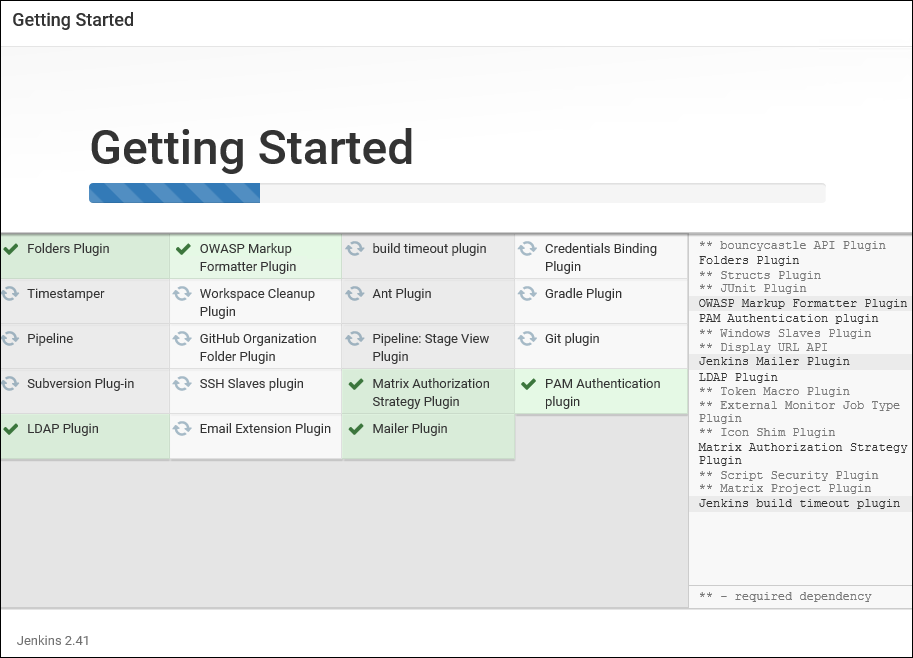
1. To get the necessary value, run the following command in your SSH session and paste the value into your browser and click **Continue**

cat /home/tomcat/.jenkins/secrets/initialAdminPassword



1. Click **Install suggested plugins** after getting past the **Unlock** page. (NOTE: The version of Jenkins has also been updated)

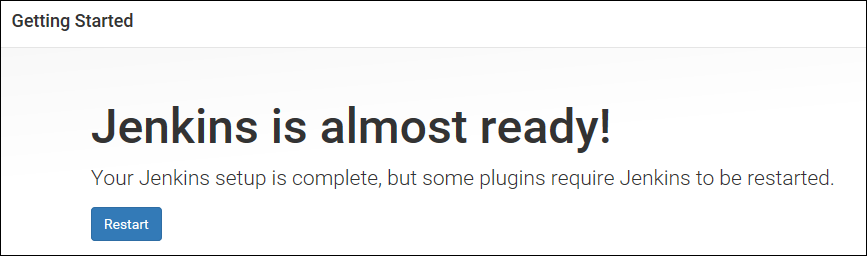




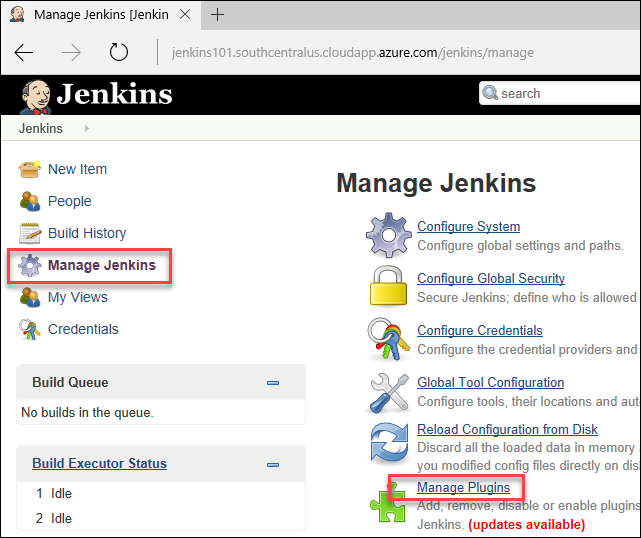
1. You may need to refresh the browser page to update the install status.
2. On the **Getting Started** page, specify the following
   * Username: **jenkinsadmin**
   * Password: **Demo@pass123**
   * Full name: **Jenkins Admin**
   * E-mail address: [**jenkins@admin.com**](mailto:jenkins@admin.com)



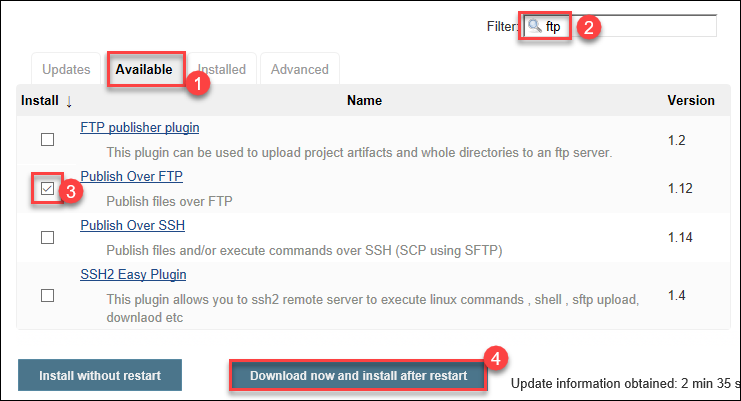
1. **If prompted**, click the **Restart** button.



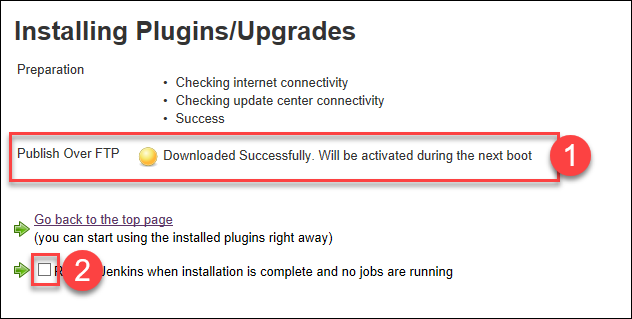
1. Once the Jenkins server comes back online, log in with the newly created **jenkinsadmin** account. Click **Manage Jenkins** on the left-hand side and then **click Manage Plugins**.



1. There is one Plug-In that you need to install for the next exercises. Click the Available tab and type **ftp**. Click the check box for **Publish Over FTP** and click **Download now and install after restart**



1. Check the box to Restart Jenkins.

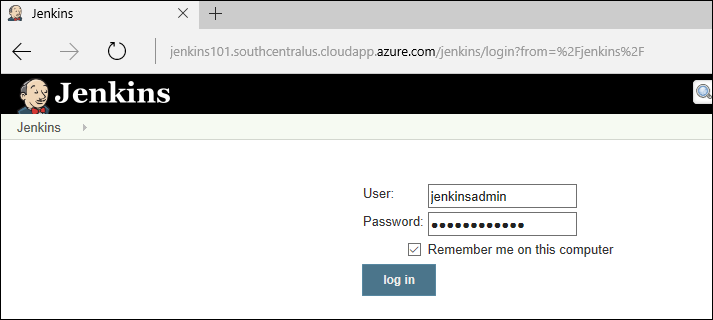


1. Once the restart has completed, you will be redirected to the login page once again. (*You may need to refresh the browser page to update the install status.*)

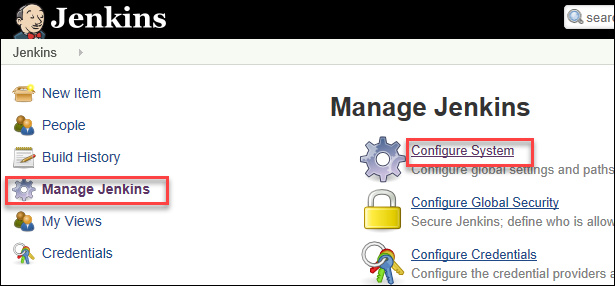
### Task 3: Configure Jenkins Staging Deployment

You are now ready to define your staging deployment job.

1. Login to your Jenkins portal with the **jenkinsamdin** account.



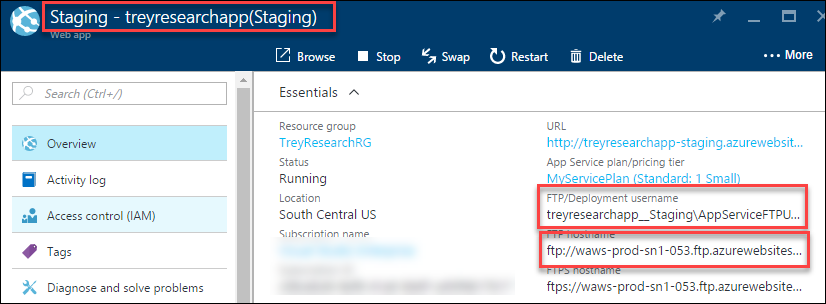
1. You will first configure your FTP Plugin with the information from your Azure App Service. Click **Manage Jenkins** and then **Configure System**.



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



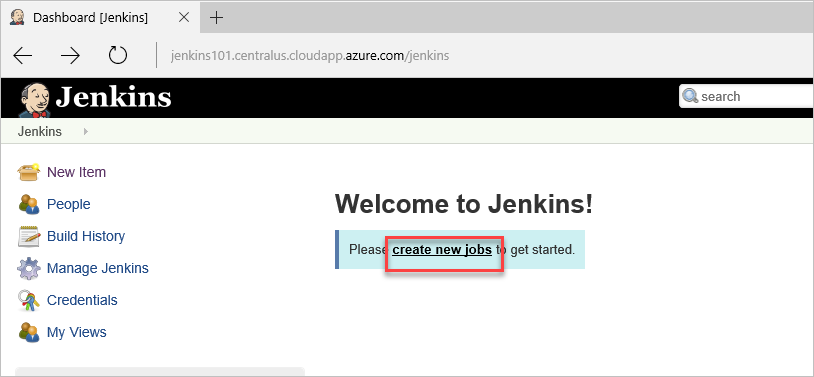
1. The information that is needed here, can be obtained from the settings of the App Service in Azure. Since you will be deploying to the Staging slot, be sure to get the information from the Staging Slot settings. (**NOTE**: The only difference between the different slots is the username, the ftp url is the same).



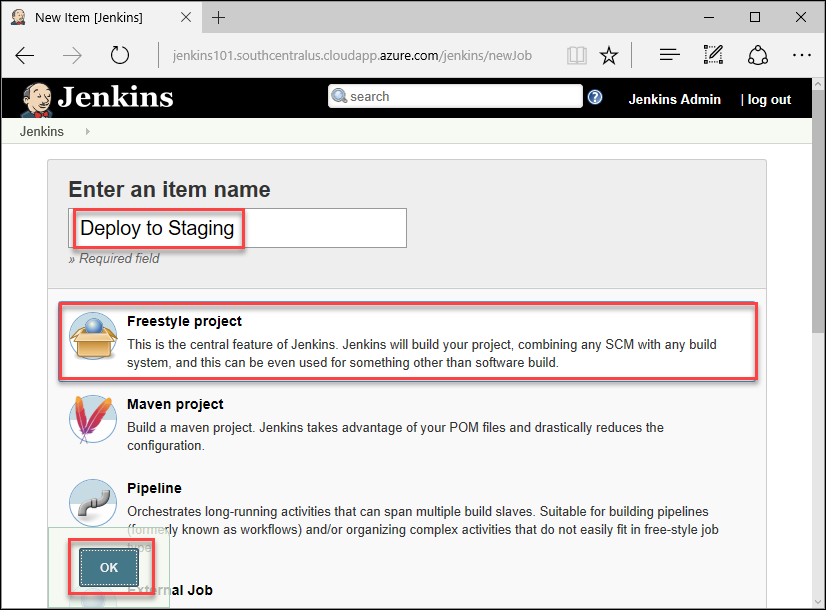
1. Update the Plug-In settings with the following information and click **Save** (after confirming the connection is successful by clicking **Test Configuration**).
   * Name: **Staging Slot for Web App**
   * Hostname: <**ftp hostname from web app staging slot settings**>
   * Username: <**username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



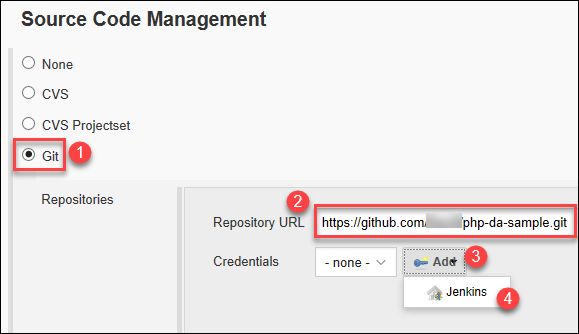
1. On the Welcome page, click the **create new jobs** link

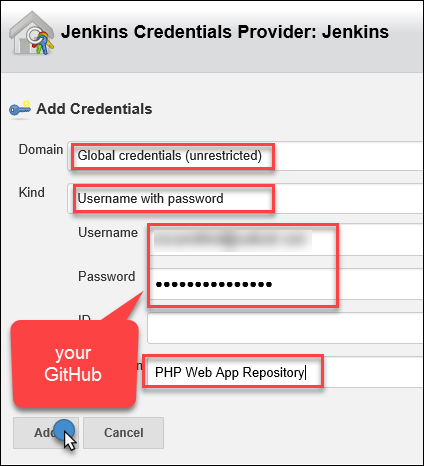


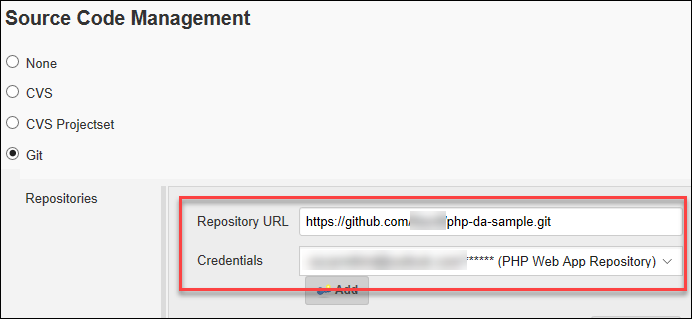
1. Choose Freestyle project and name the project **Deploy to Staging** andClick **OK**



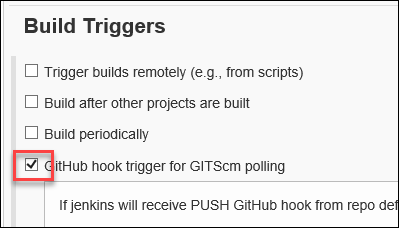
1. In the **Source Code Management** section, choose Git and specify your GitHub repository URL and click Add to configure your credentials



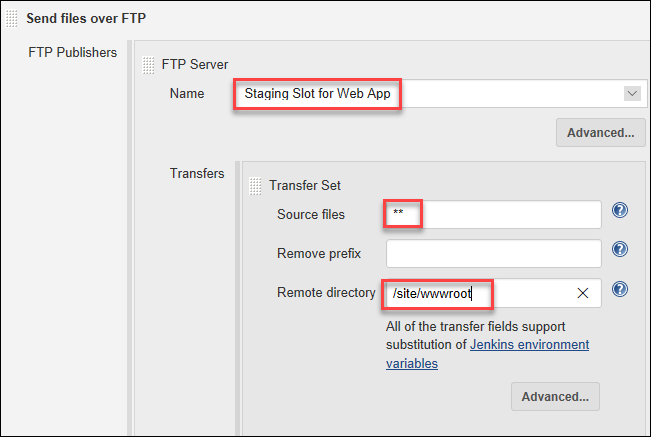




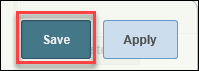
1. In Build Triggers, check **GitHub hook trigger for GITScm polling**



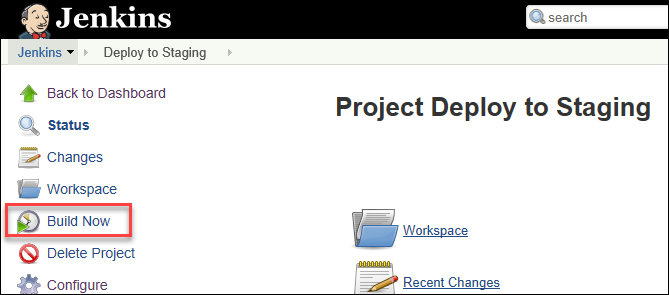
1. In the Build section, from the Add build step dropdown list, choose **Send files over FTP**. Be sure to choose the FTP that you defined earlier and specify **\*\*** for the **Source files** and **/site/wwwroot** for the **Remote directory**.



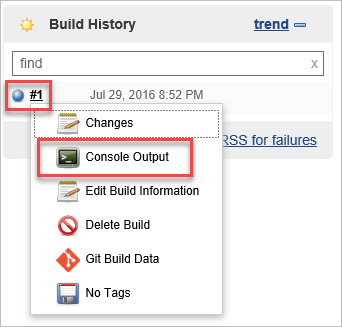
1. Click Save to Save your changes.



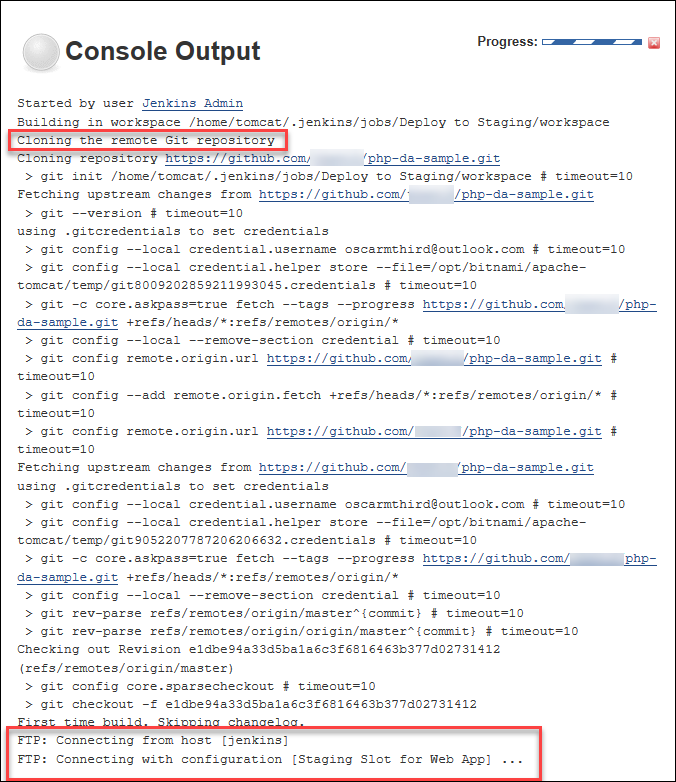
1. Although you specified to run this project whenever a check-in was done in GitHub, you can force the job to run. Click **Build Now** on project page



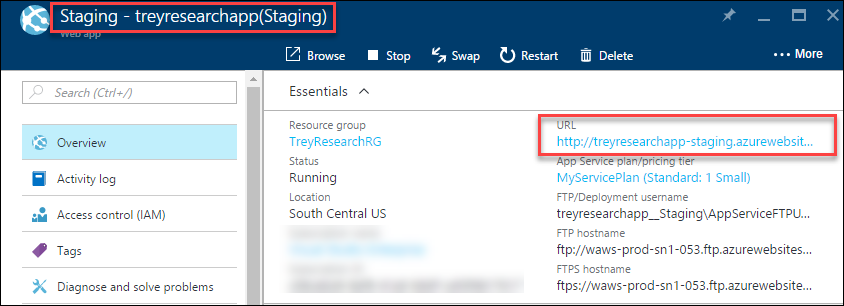
1. You will see an entry in Build History with your build number. Hover over the number to get a dropdown list of options. Choose Console Output



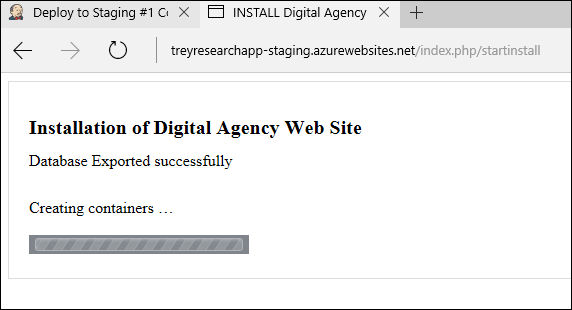
1. Review the Console Output for any errors etc. You can see in the Console Output where the call to your GitHub repo is made and where it is synced on your Jenkins server. You can also see the commands specified and their output.



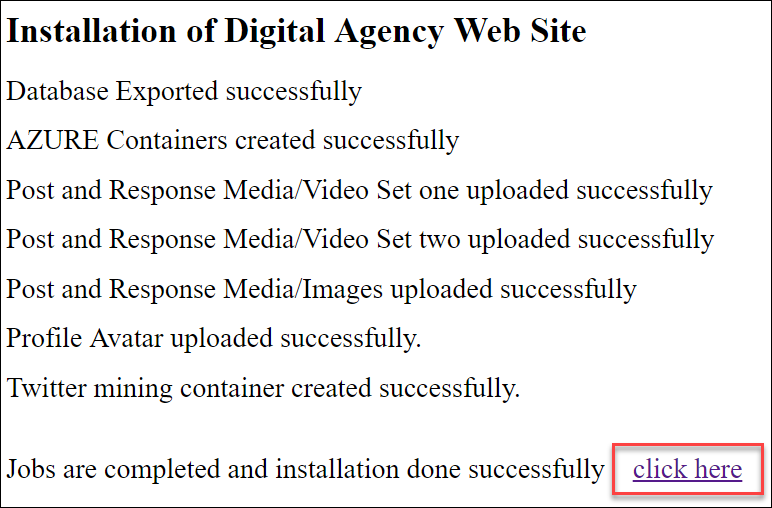
1. Once the job has completed, you can verify the deployment by browsing to the URL of the Staging Slot.



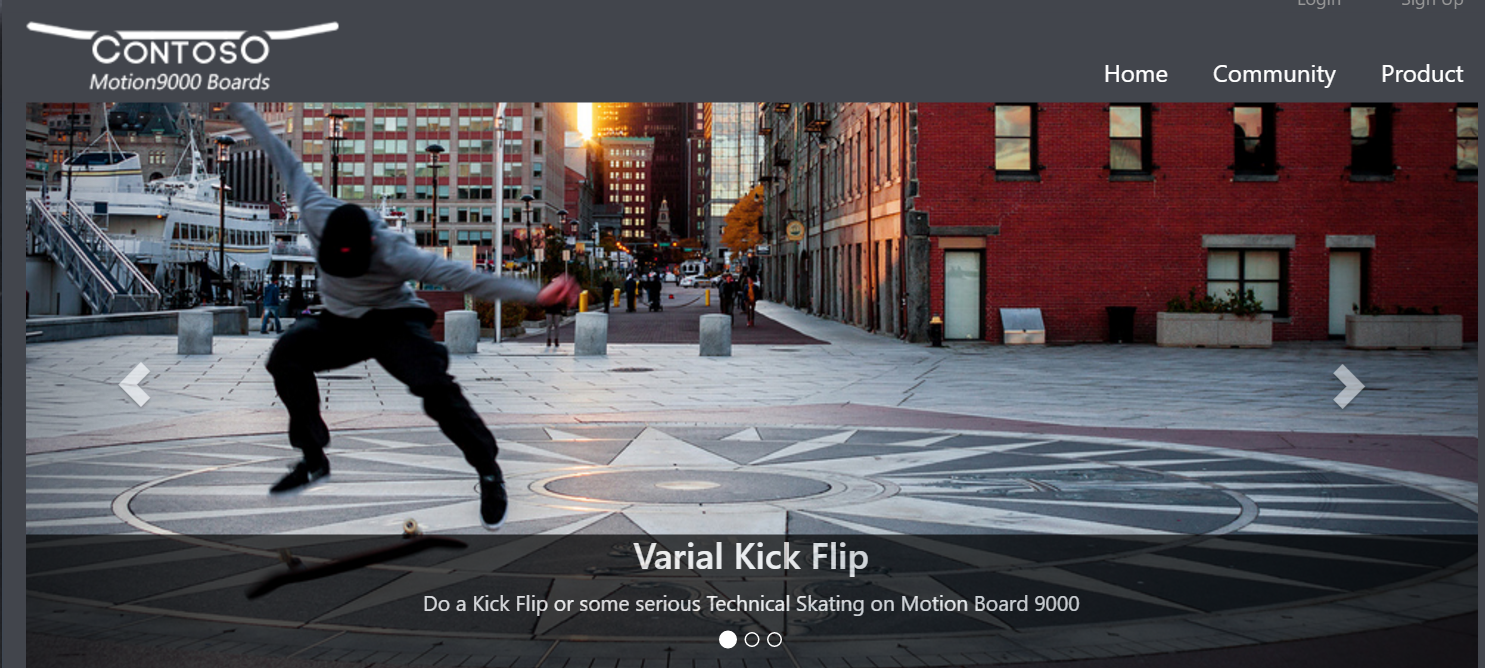
1. You will notice that the site has been deployed and is now beginning to provision itself



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



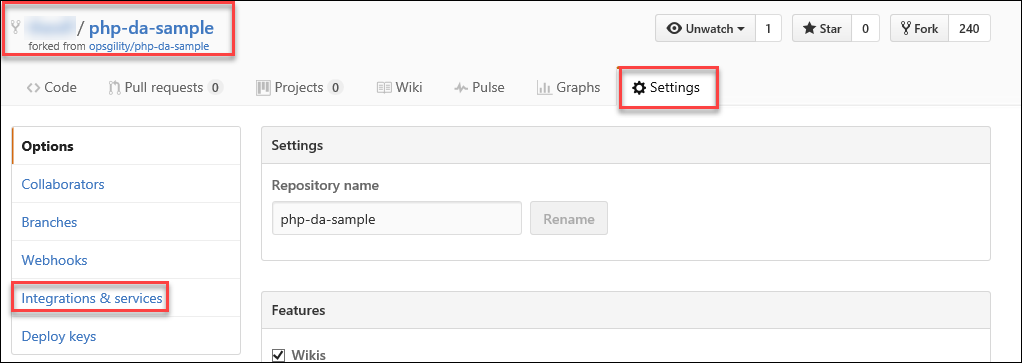
1. You should see the home page load.



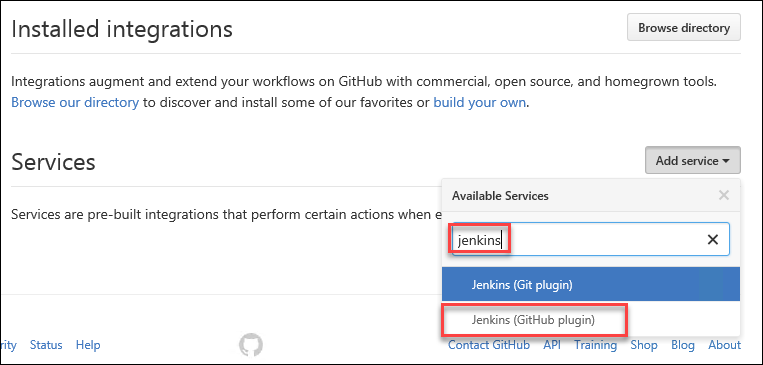
### Task 4: Configure your GitHub Repo to Notify Jenkins of Changes

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

1. Log into your GitHub repo ([https://github.com/<username>/php-da-sample](https://github.com/%3cusername%3e/php-da-sample)) and click on **Settings** and then **Integration & services**

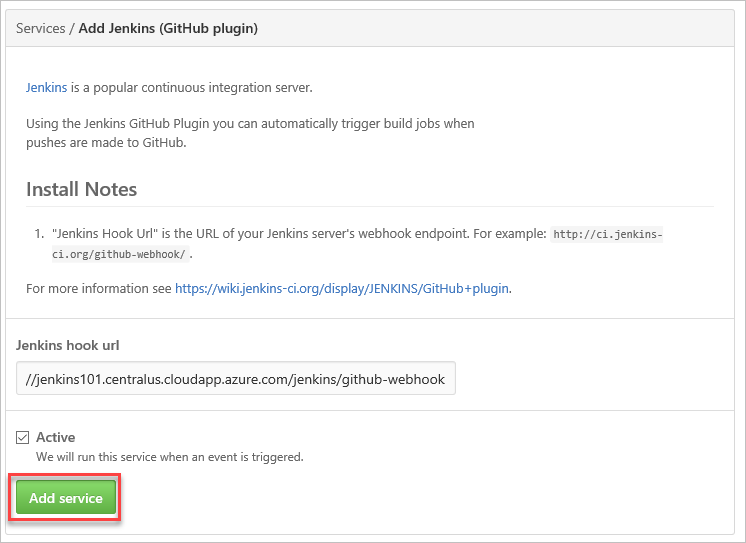


1. Click **Add service** and choose **Jenkins (GitHub plugin)**



1. 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.

<https://jenkins101.centralus.cloudapp.azure.com/jenkins/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 a text editor **C:\OpsgilityTraining\repos\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>

1. Make a modification to the text such as changing the model number and then click File🡪Save.

<h3>Pro Skate Board Motion 10000</h3>

<p> Do all your cool stuff on Contoso board Motion 10000.</p>

1. 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 (**c:\OpsgilityTraining\repos\php-da-sample**).

git config user.name "Your Name"

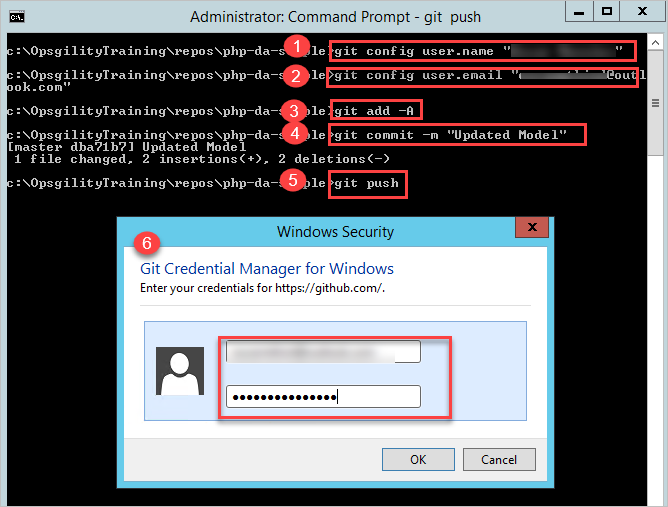
git config user.email "your@email.com"

git add -A

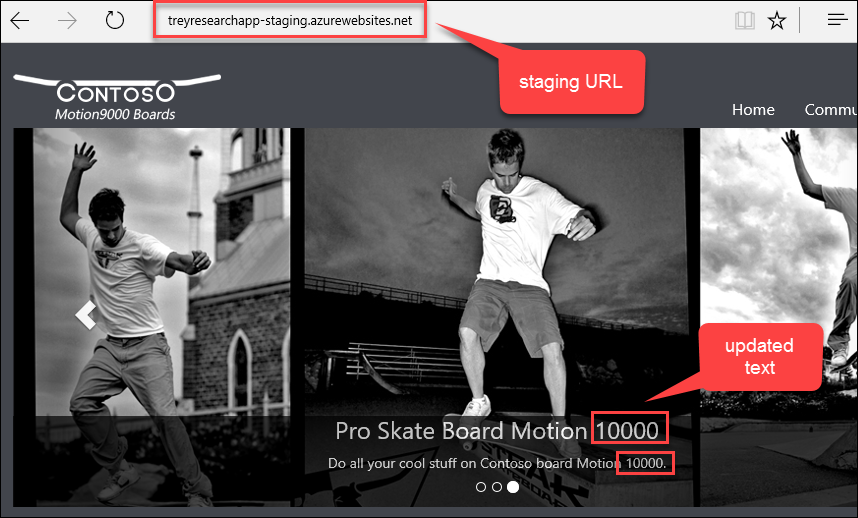
git commit -m 'updated model'

git push

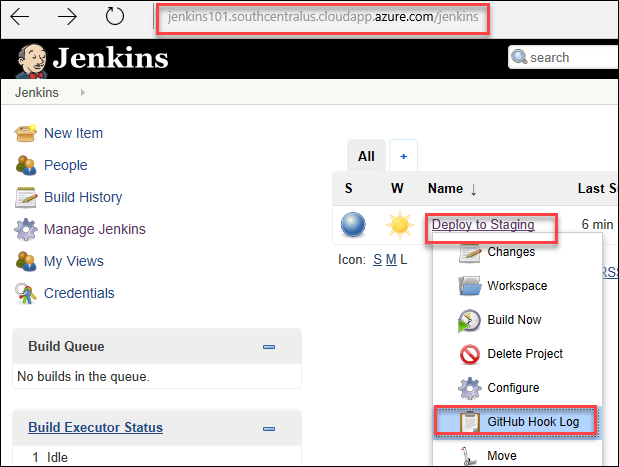
Note: You may be required to authenticate using your github.com username and password.



1. 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.



1. 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**



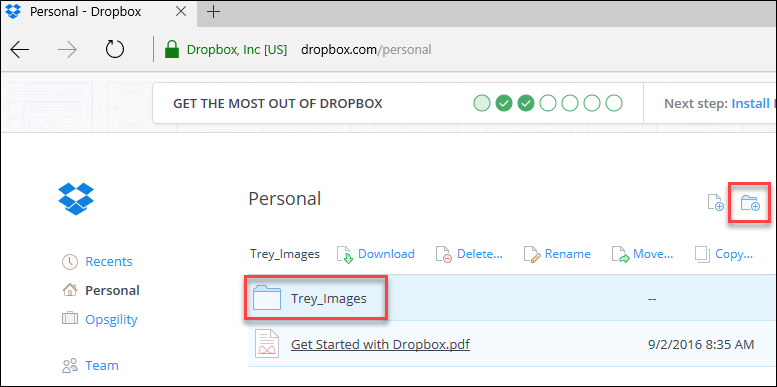
1. You can also check the Build History log and Console Output of the Jenkins project.



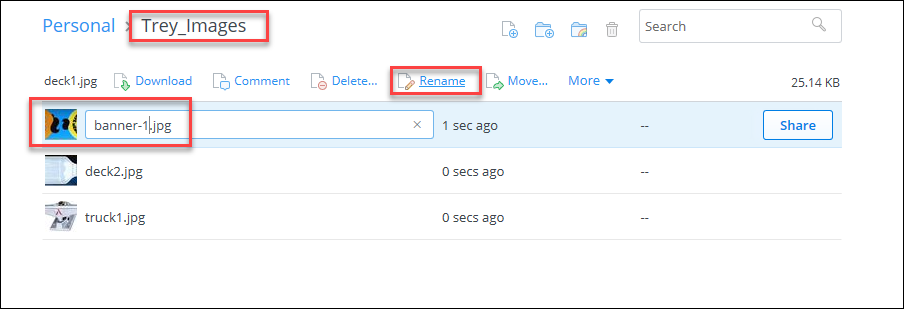
### Task 6: Update Jenkins Project to account for Dropbox content

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 scenario where Jenkins goes above and beyond what can be done with Azure alone.

1. Log into your Dropbox account at <https://www.dropbox.com> and create a new folder name **Trey\_Images**

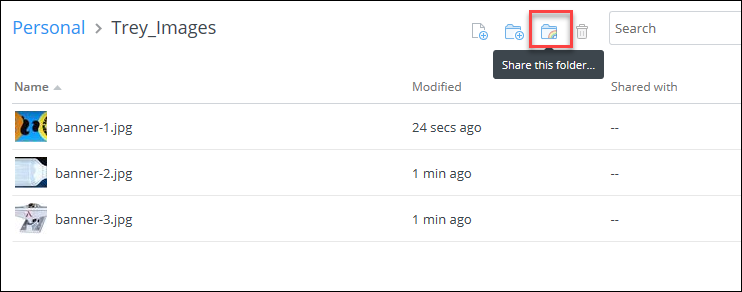


1. Upload the following three images from your local git repository. You are going to replace the three banner images on the homepage, with three product images from **repos\php-da-sample\assets\images\products\large**. Upload the following three images and rename them accordingly
   * deck1.jpg – banner-1.jpg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

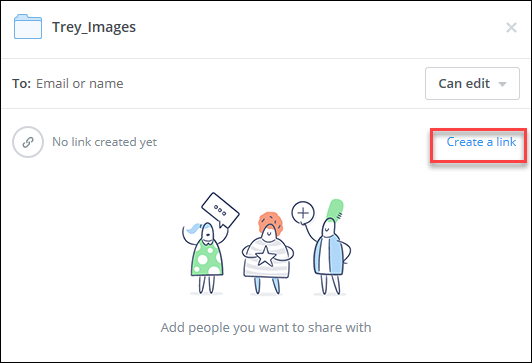




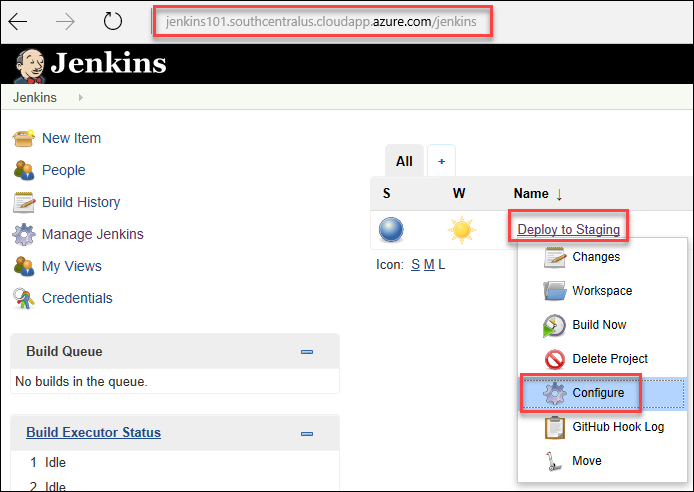
1. To create a share link of this folder, click the **Share this folder… icon**



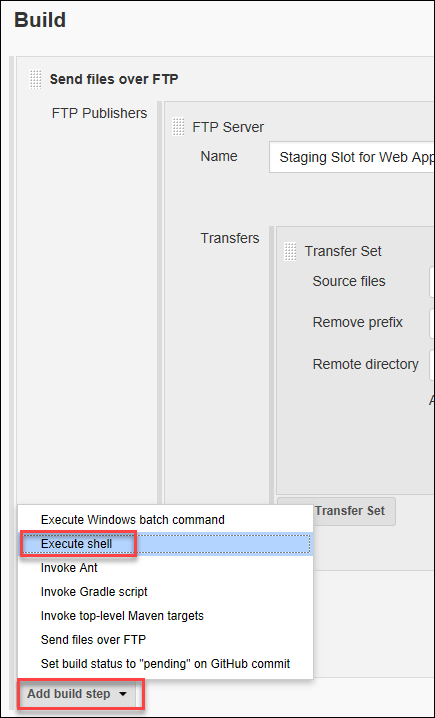
1. Click **Create a link**



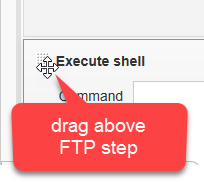
1. Once the link is created, click **Copy link** and save to a file for use later. The link will look something like the following: <https://www.dropbox.com/sh/lm8cyar53f23pc6/AAAlxTGrH5j8pyaNZvmfzewRa?dl=0>
2. Log back into your Jenkins portal, hover over your previously defined job (**Deploy to Staging**) to get the dropdown list and choose **Configure.**

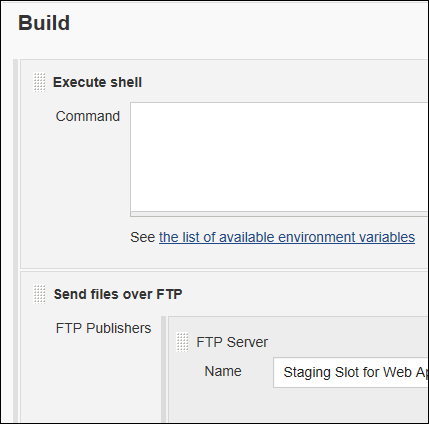


1. You will now add another Build step (**Execute shell**) that should execute before the FTP copy.



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





1. In the **Command** window add the following commands.

**NOTE** that 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).

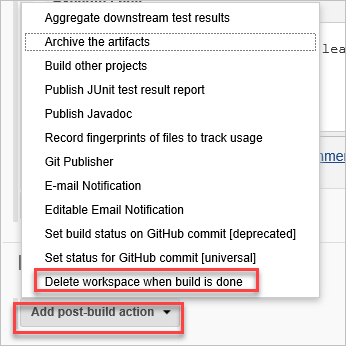
curl -L https://<drobox\_folder\_share\_url\_**dl=1**> > download.zip

7z e download.zip -oc:\Unzipped -y

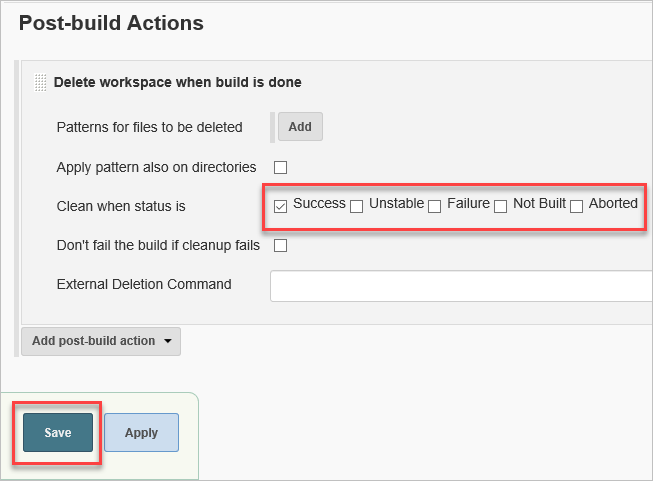
**cp Unzipped/\* "/home/tomcat/.jenkins/workspace/Deploy to Staging/assets/images/" -f || :**



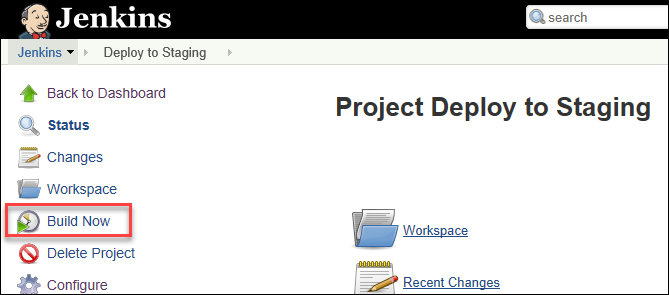
1. In the Post-build Actions section, from the Add post-build action dropdown list, choose Delete workspace when build is done



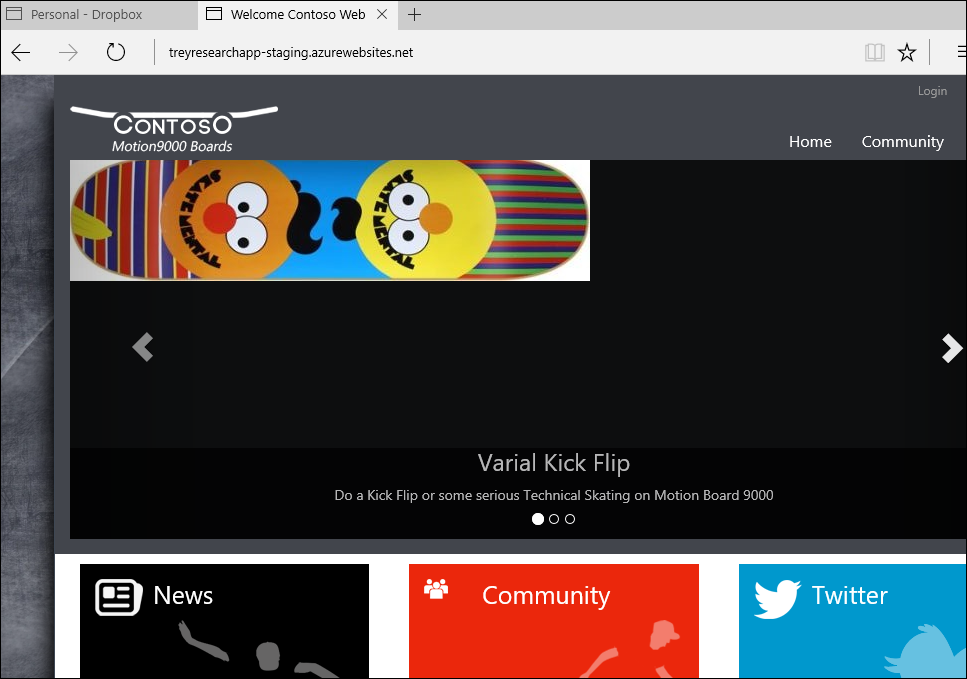
1. Click Advanced and make sure only **Success** is checked and click **Save** to commit your changes



1. Click **Build Now** on project page to force a build and deploy to Staging.



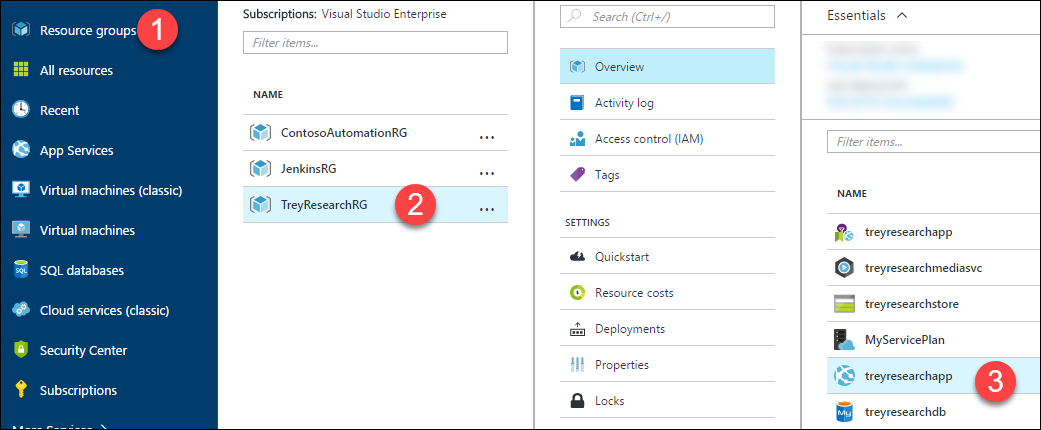
1. Once the build completes, browse to the Staging slot of your Web Application and you should see the updated images (that came from Dropbox) reflected.



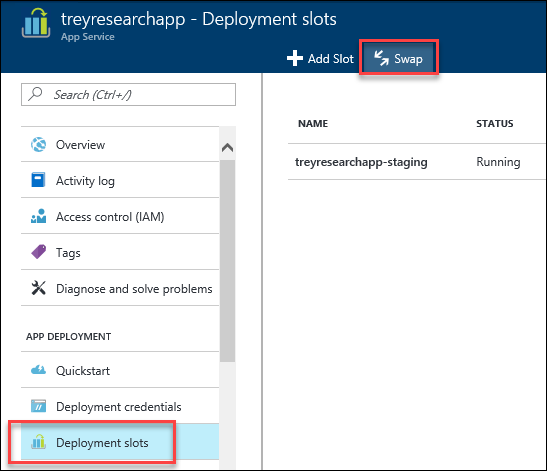
### Task 7: Manually Deploy to Production

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.

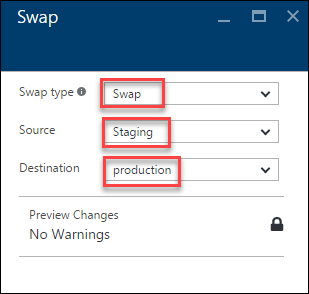
1. To push the changes from your Staging Slot to production, in the Azure portal (<https://portal.azure.com>) click **Resource groups > TreyResearchRG > treyresearchapp**



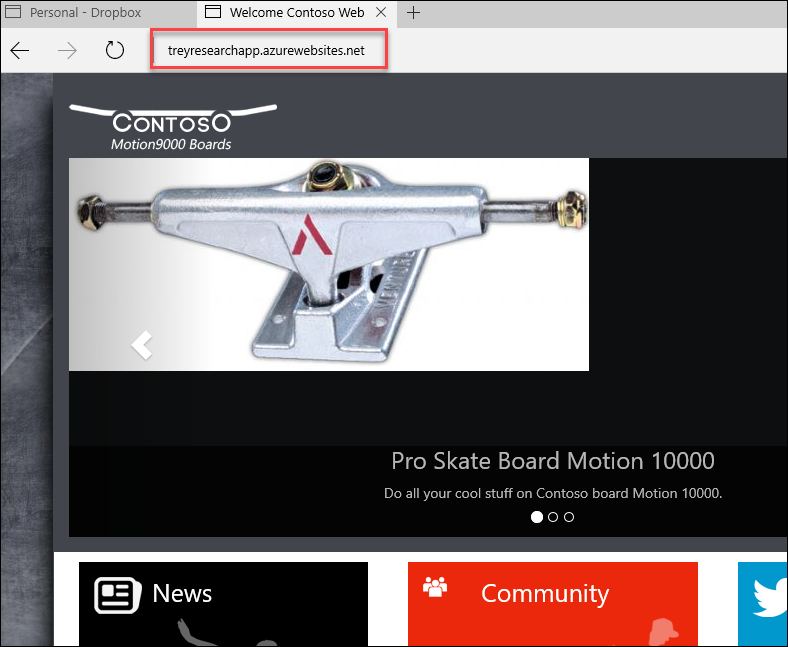
1. Click **Deployment slots > Swap**



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



1. Once the Swap has completed, verify that 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.