



Microsoft Cloud Workshop

OSS PaaS and DevOps

Hands-on lab step-by-step

April 2018

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third-party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2018 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/Usage/General.aspx> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Contents

OSS PaaS and DevOps hands-on lab step-by-step	1
Abstract and learning objectives.....	1
Overview	1
Solution architecture.....	2
Requirements	3
Before the hands-on lab.....	4
Task 1: Provision a resource group	4
Task 2: Create a development virtual machine	4
Task 3: Provision a Jenkins server.....	6
Task 4: Create GitHub account	11
Task 5: Fork the starter app	13
Exercise 1: Run starter application	15
Task 1: Connect to your Lab VM.....	15
Task 2: Grant permissions to Docker	16
Task 3: Integrate GitHub into VS Code	17
Task 4: Clone the starter application	20
Task 5: Launch the starter application.....	22
Exercise 2: Migrate the database to Cosmos DB.....	25
Task 1: Provision Cosmos DB using the MongoDB API.....	25
Task 2: Update database connection string.....	26
Task 3: Pre-create and scale collections	29
Task 4: Import data to the API for MongoDB using mongoimport.....	31
Task 5: Install Azure Cosmos DB extension for VS Code	34
Task 6: Decrease collection throughput	37
Exercise 3: Containerize the app.....	38
Task 1: Create an Azure Container Registry	38
Task 2: Install Docker extension in VS Code	39
Task 3: Create Docker image and run the app	40
Task 4: Run the containerized App.....	43
Task 5: Push image to Azure Container Registry	44
Exercise 4: Set up Web App for Containers.....	47
Task 1: Provision Web App for Containers	47
Task 2: Navigate to the deployed app	48
Exercise 5: Configure CI/CD pipeline.....	50
Task 1: Prepare GitHub account for service integrations.....	50
Task 2: Configure Continuous Integration (CI) with Jenkins	56
Task 3: Trigger CI build	64

Task 4: Create Free Visual Studio Team Services Account.....	66
Task 5: Create a VSTS personal access token	68
Task 6: Configure Jenkins for Team Services integration.....	71
Task 7: Create a Jenkins service endpoint is VSTS	73
Task 8: Create a Team Services release definition.....	75
Task 9: Trigger CI/CD pipeline	82
Exercise 6: Create Azure Function for order processing.....	86
Task 1: Provision a Function App.....	86
Task 2: Configure storage queues.....	87
Task 3: Create Cosmos DB trigger function.....	89
Task 4: Create Queue function	99
Exercise 7: Create Logic App for sending SMS notifications.....	108
Task 1: Create Free Twilio account.....	108
Task 2: Create Logic App	112
After the hands-on lab	126
Task 1: Delete Azure resource groups.....	126
Task 2: Delete WebHooks and Service Integrations.....	126
Appendix A: Lab VM setup.....	127
Task 1: Create VM config script.....	127
Task 2: Create a Linux virtual machine	127

OSS PaaS and DevOps

hands-on lab step-by-step

Abstract and learning objectives

This workshop is designed to help students gain a better understanding of how to integrate and deploy complex Open Source Software (OSS) workloads into Azure PaaS. Attendees will migrate an existing MERN (MongoDB, Express.js, React.js, Node.js) stack application from a hosted environment into Azure PaaS services, and fully embrace modern DevOps tools.

Attendees will learn how to:

- Provision Web App for Containers for hosting OSS applications
- Migrate a MongoDB instance into Cosmos DB
- Implement serverless technologies, such as Logic Apps and Azure Functions, to enhance OSS app functionality
- Provision an Azure Container Registry
- Build Docker images and push them into the Azure Container Registry
- Enable continuous deployment with Jenkins and Visual Studio Team Services (VSTS)

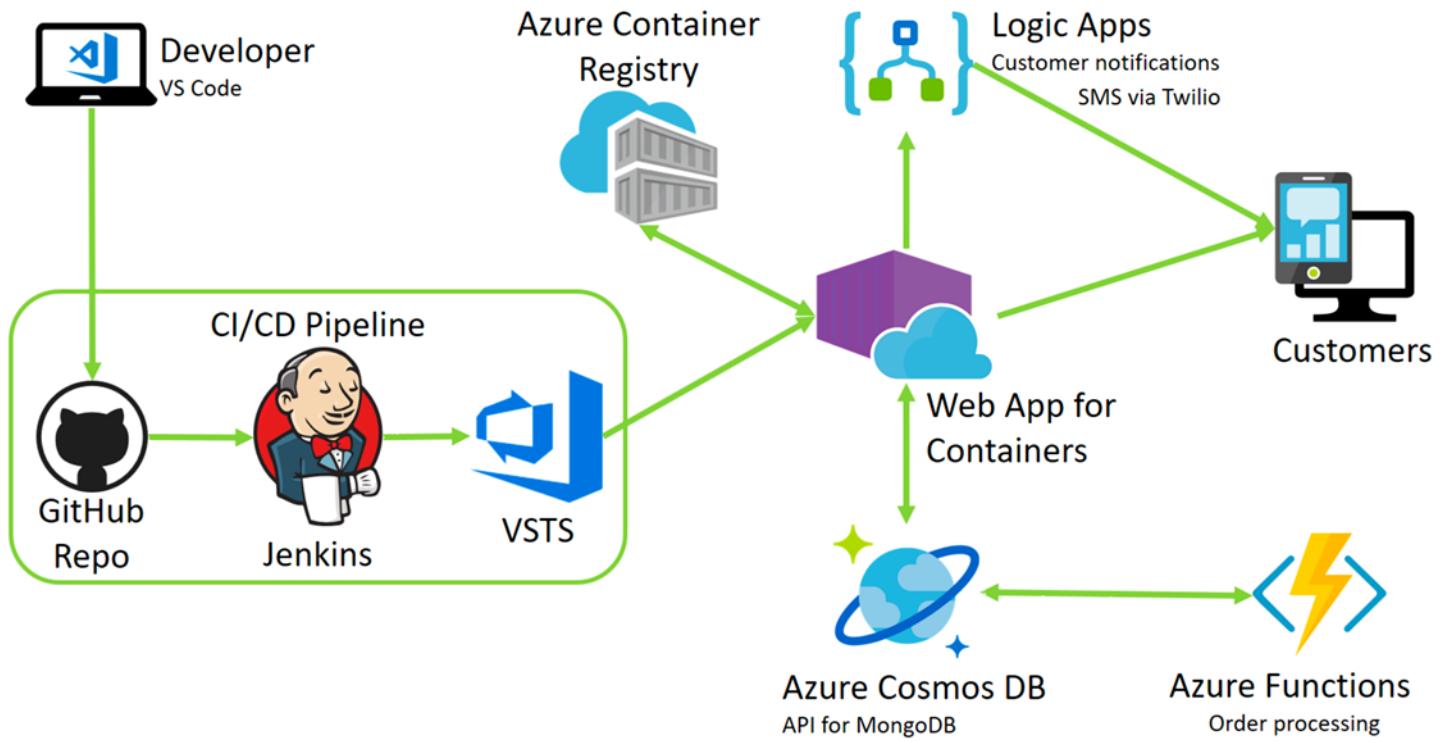
Overview

Best For You Organics Company is one of the leading health food suppliers in North America, serving customers in Canada, Mexico, and the United States. They have a MERN stack web application which they host on-premises and are looking to migrate their OSS application into Azure. They will be creating a custom Docker image for their application and using a Jenkins and Visual Studio Team Services (VSTS) continuous integration/continuous delivery (CI/CD) pipeline to deploy the application into a Web App for Containers instance. In addition, they will be setting up Azure Cosmos DB, and using the MongoDB APIs, so they don't have to make application code changes to leverage the power of Cosmos DB.

In this hands-on lab, you will assist with completing the OSS application and database migrations into Azure. You will create a custom Docker image, provision an Azure Container Registry, push the image to the registry, and configure the CI/CD pipeline to deploy the application to a Web App for Containers instance. You will also help them implement functionality enhancements using serverless architecture.

Solution architecture

Below is a diagram of the solution architecture you will build in this lab. Please study this carefully, so you understand the whole of the solution as you are working on the various components.



The solution begins with developers using Visual Studio Code (VS Code) as their code editor, so they can leverage its rich integration with GitHub, Docker, and Azure. From a high level, developers will package the entire OSS application inside a custom Docker container using the Docker extension in VS Code. The image will be pushed to an Azure Container Registry as part of a continuous integration/continuous delivery (CI/CD) pipeline using GitHub, Jenkins, and Visual Studio Team Services (VSTS). This Docker image will then be deployed to a Web App for Containers instance, as part of their continuous delivery process using Release Management in VSTS.

The MongoDB database will be imported into Azure Cosmos DB, using mongoimport.exe, and access the database from the application will continue to use the MongoDB APIs. The database connection string in the application will be updated to point to the new Cosmos DB.

Serverless architecture will be applied to order processing and customer notifications. Azure Functions will be used to automate the processing of orders. Logic Apps will be applied to send SMS notifications, via a Twilio connector, to customers informing them that their order has been processed and shipped.

Requirements

1. Microsoft Azure subscription must be pay-as-you-go or MSDN
 - a. Trial subscriptions will *not* work
2. Linux virtual machine configured with:
 - a. Visual Studio Code
 - b. Azure CLI
 - c. Docker
 - d. Node.js and npm
 - e. MongoDB Community Edition

Before the hands-on lab

Duration: 30 minutes

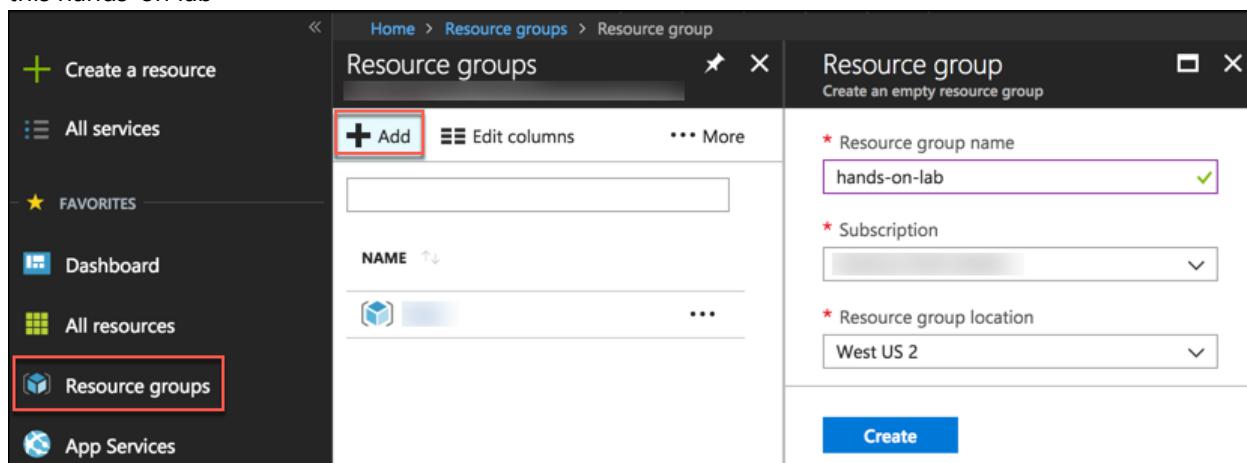
In this exercise, you will set up your environment for use in the rest of the hands-on lab. You should follow all steps provided *before* attending the hands-on lab.

IMPORTANT: Most Azure resources require unique names. Throughout this lab you will see the word "SUFFIX" as part of resource names. You should replace this with your Microsoft alias, initials, or another value to ensure the resource is uniquely named.

Task 1: Provision a resource group

In this task, you will create an Azure resource group for the resources used throughout this lab.

1. In the [Azure Portal](#), select **Resource groups**, select **+Add**, then enter the following in the Create an empty resource group blade:
 - a. **Name:** Enter hands-on-lab-SUFFIX
 - b. **Subscription:** Select the subscription you are using for this hands-on lab
 - c. **Resource group location:** Select either **East US, West US 2, West Europe, or Southeast Asia**, as these are currently the only regions which offer Dv3 and Ev3 VMs. Remember this location for other resources in this hands-on lab



2. Select **Create**.

Task 2: Create a development virtual machine

In this task, you will provision a Linux virtual machine (VM) running Ubuntu Server 16.04 LTS, which will be used as your development machine throughout this lab. The VM will be created using an Azure Resource Manager (ARM) template from a GitHub repository. The ARM template includes a custom extension script which installs Docker, Visual Studio Code (VS Code), MongoDB, and other required software on the VM. The ARM template also adds an inbound port rule that opens port 3389 on the network security group for the VM to allow RDP connections. To review the steps to manually provision the VM and installed software, see [Appendix A](#).

1. To open a custom deployment screen in the Azure portal, click the following link:
<https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fraw.githubusercontent.com%2FZoinerTejada%2Fmcw-oss-paas-devops%2Fmaster%2FLabVM%2Fazure-deploy.json>

2. On the custom deployment screen in the Azure portal, enter the following:
 - a. **Subscription:** Select the subscription you are using for this hands-on lab
 - b. **Resource group:** Select Use existing, and select the hands-on-lab-SUFFIX resource group
 - c. **Location:** Select the location you used for the hands-on-lab-SUFFIX resource group
 - d. **Virtual Machine Name:** Accept the default value, LabVM
 - e. **Admin Username:** Accept the default value, demouser
 - f. **Admin Password:** Accept the default value, Password.1!!
 - g. Check the box to agree to the Azure Marketplace terms and conditions.
 - h. Select **Purchase**.

The screenshot shows the 'Custom deployment' dialog box from the Azure portal. It has sections for TEMPLATE, BASICS, SETTINGS, TERMS AND CONDITIONS, and a footer.

TEMPLATE: Shows a 'Customized template' with 5 resources. Buttons for 'Edit template', 'Edit parameters', and 'Learn more' are available.

BASICS: Fields for Subscription (selected), Resource group (set to 'Use existing' with 'hands-on-lab'), and Location (set to 'West US 2').

SETTINGS: Fields for Virtual Machine Name (LabVM), Admin Username (demouser), and Admin Password (redacted).

TERMS AND CONDITIONS: A section containing the 'Azure Marketplace Terms' and 'Azure Marketplace' links, a detailed description of the terms of service, and a note about Microsoft's responsibility for third-party templates. It includes a checked checkbox for agreeing to the terms and conditions, and an unchecked checkbox for pinning to the dashboard.

Footer: A 'Purchase' button.

3. It takes about 20 minutes to deploy the Lab VM. Move on to the next task while the VM is deploying.

Task 3: Provision a Jenkins server

In this task, you will provision an Azure Linux VM, which will serve as your Jenkins server for this hands-on lab.

1. In the Azure portal, select **+Create a resource**, enter “Jenkins” into the **Search the Marketplace** box, then select the **Jenkins** compute item from the results.

The screenshot shows the Azure Marketplace search interface. A red box highlights the search bar containing the text "jenkins". Below the search bar, the results table has three columns: NAME, PUBLISHER, and CATEGORY. The first result, "Jenkins" by Microsoft, is highlighted with a red box. The second result, "Jenkins Cluster" by Bitnami, is also visible.

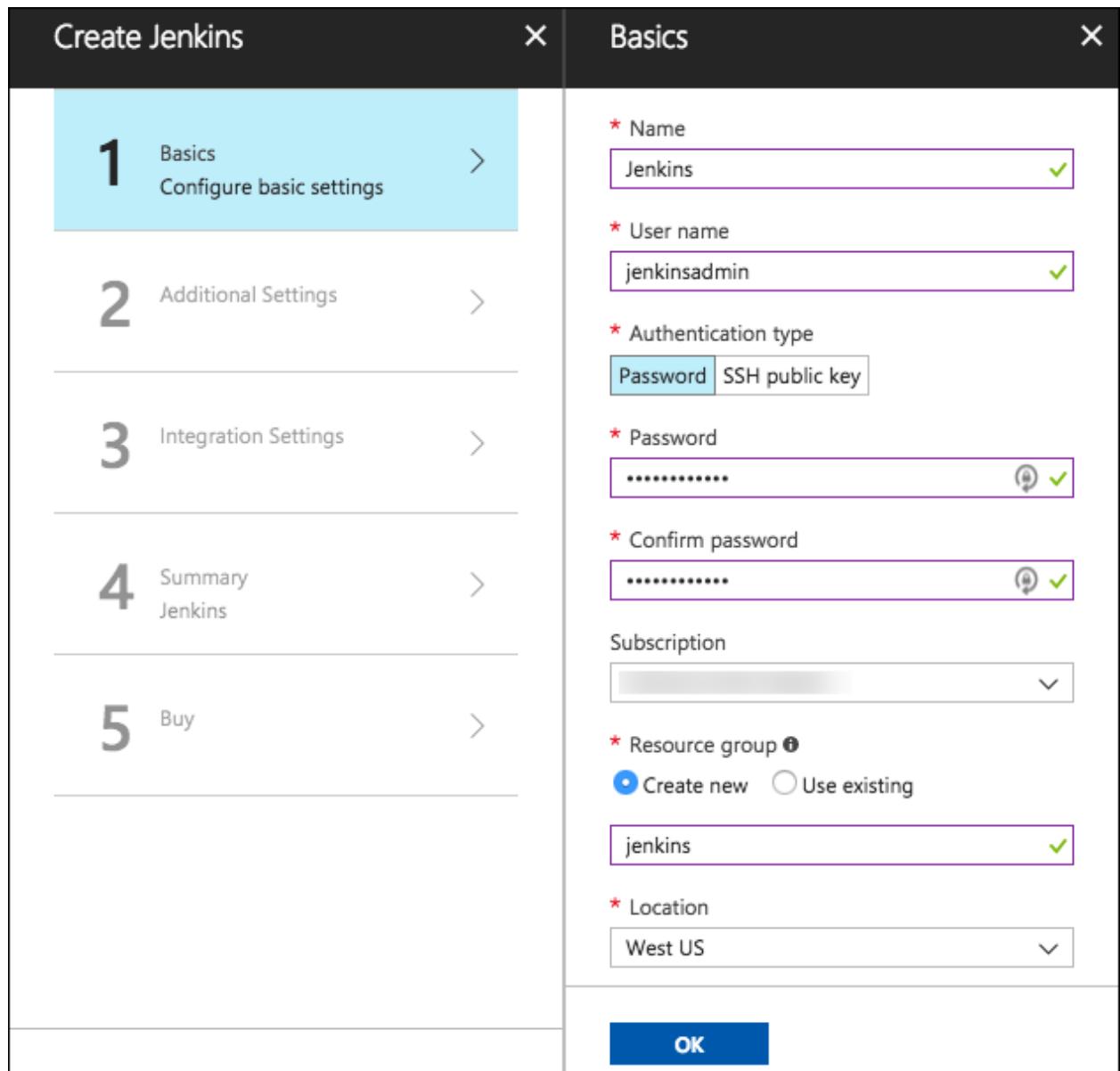
NAME	PUBLISHER	CATEGORY
Jenkins	Microsoft	Compute
Jenkins Cluster	Bitnami	Compute

2. On the **Jenkins** blade, select **Create** to configure the Jenkins server.

This screenshot shows the Jenkins blade in the Azure Marketplace. It includes the following sections:

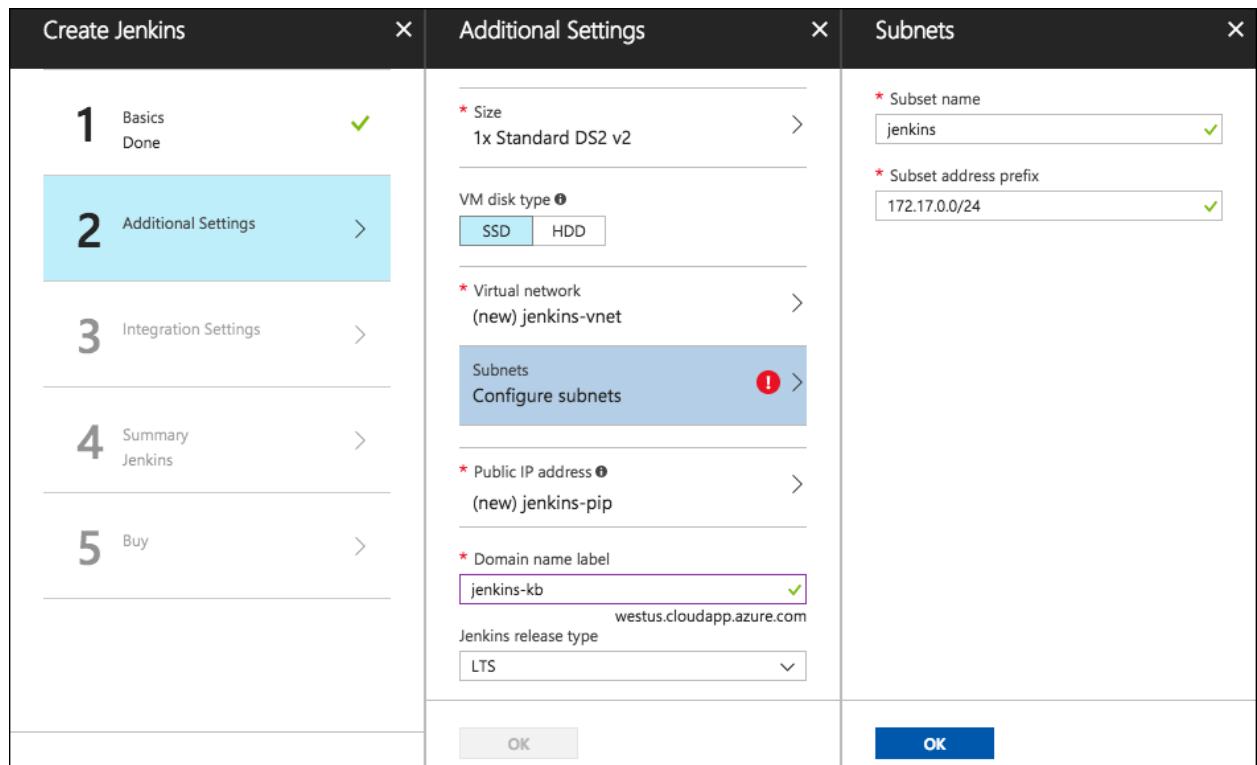
- Header:** Shows the Jenkins logo and Microsoft branding.
- Text:** A message about bringing support for Jenkins to Azure.
- Note:** Instructions for connecting to the Jenkins instance once deployed.
- Description:** Details about the solution template, mentioning Ubuntu 14.04 LTS and supported tools.
- List:** A bulleted list of included tools and plugins.
- Link:** A link to a blog post for a detailed walkthrough.
- Diagram:** An illustration showing Jenkins integrated with VM Agents, Storage, Docker, and Kubernetes.
- Deployment Model:** A dropdown menu set to "Resource Manager".
- Create Button:** A prominent blue "Create" button at the bottom.

3. On the **Create Jenkins Basics** blade, enter the following:
- Name:** Enter "Jenkins"
 - User name:** Enter "jenkinsadmin"
 - Authentication type:** Select **Password**
 - Password:** Enter "Password.1!!"
 - Subscription:** Select the subscription you are using for this hands-on lab
 - Resource group:** Select **Create new**, and enter "jenkins-SUFFIX" (Note: this will use a different resource group than the other resources in this lab)
 - Location:** Select the location you are using for resources in this hands-on lab
 - Select **OK** to proceed to the **Settings** blade.

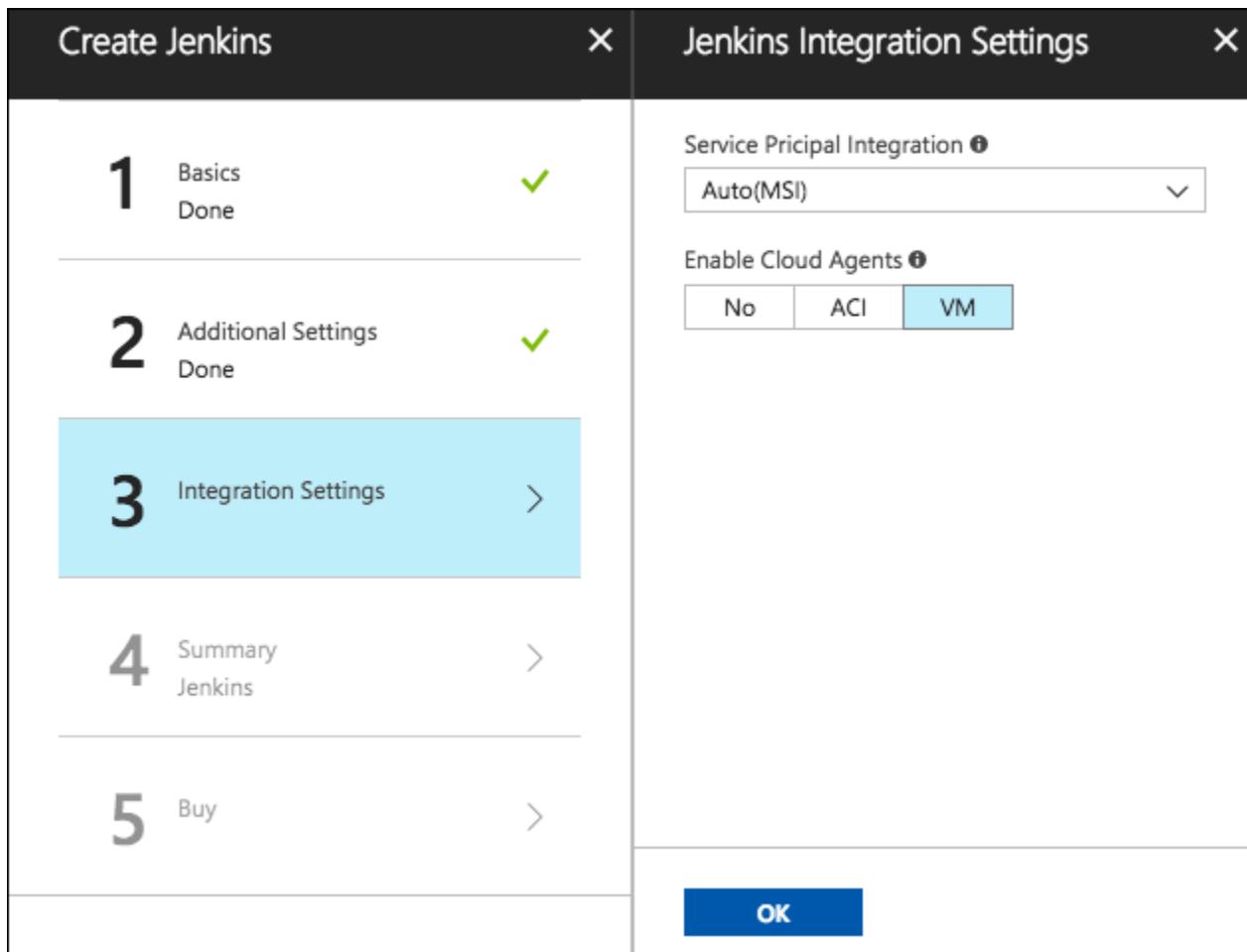


4. On the **Additional Settings** blade:
- Select **Configure subnets**, and then select **OK** on the **Subnets** blade to accept the defaults.
 - Enter a unique **Domain name label**, such as "Jenkins-SUFFIX."
 - Ensure the **Jenkins release type** is set to LTS.

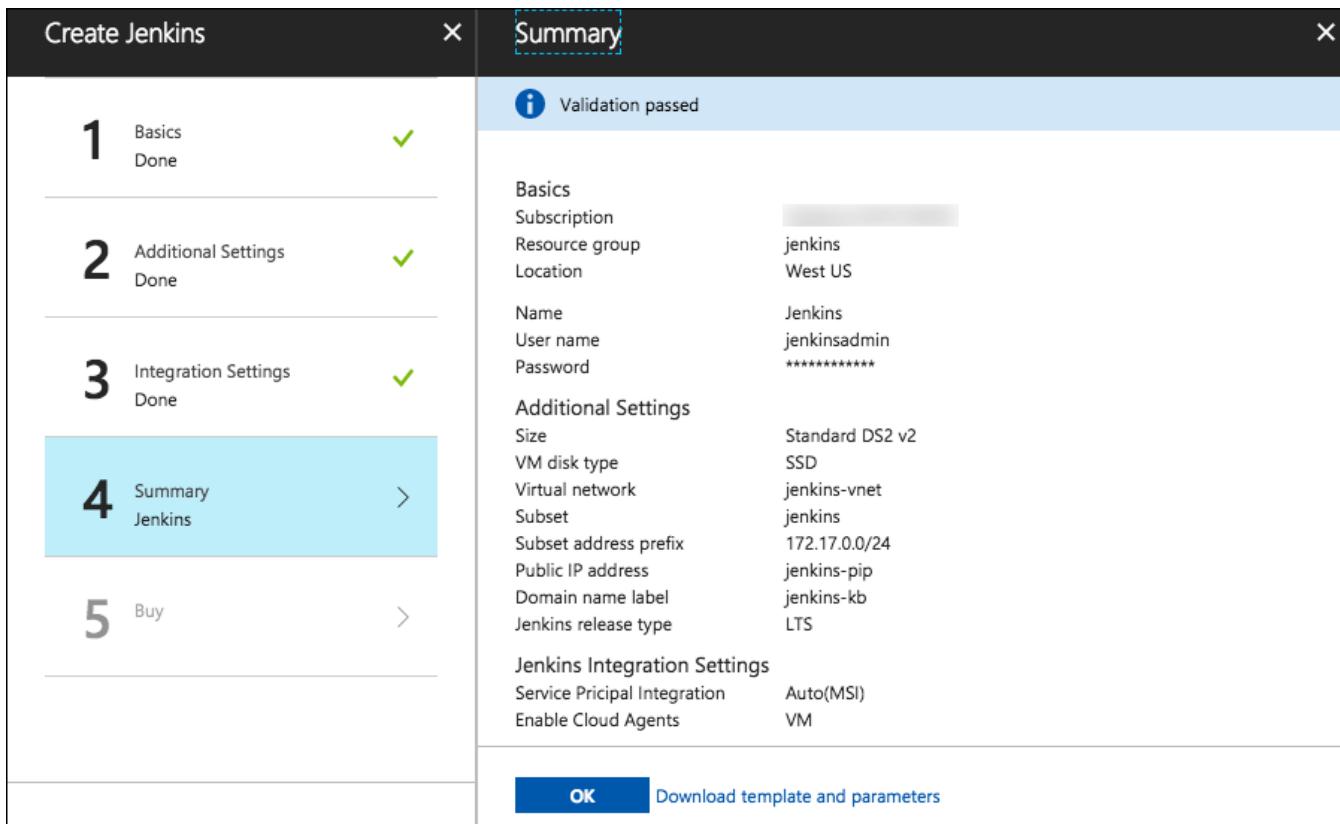
- d. Select **OK** to proceed to the Integration Settings screen.



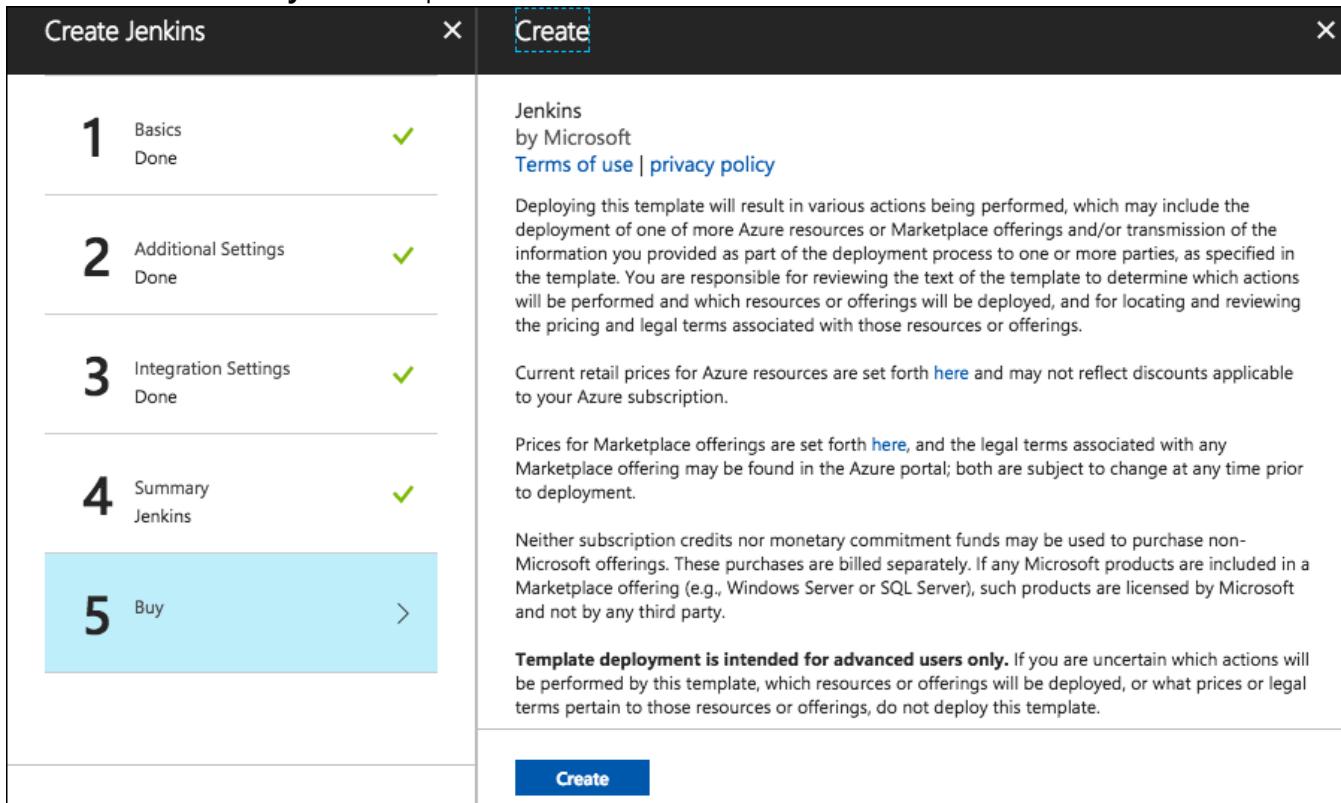
5. On the **Integration Settings** blade, select **OK**.



6. On the **Summary** blade, ensure validation passed, and select **OK**.



- Select **Create** on the **Buy** screen to provision the Jenkins server.



- It can take 10+ minutes for the VM to provision. You can move on to the next task while you wait.

Task 4: Create GitHub account

In this task, you will sign up for a free GitHub account, which will be used for hosting a copy of the sample application used throughout this lab. This account will be integrated into the CI/CD workflow for pushing updates to the application into Azure. If you already have a GitHub account, and wish to use that account, you can skip to the [next task](#).

1. Open a browser and navigate to <https://github.com>.
2. In the form on the page, enter a **username**, your **email** address, and a **password**, then select **Sign up for GitHub**.

The image shows a screenshot of the GitHub sign-up form. It consists of three input fields: 'Username' (placeholder: 'Pick a username'), 'Email' (placeholder: 'you@example.com'), and 'Password' (placeholder: 'Create a password'). Below the password field is a note: 'Use at least one letter, one numeral, and seven characters.' At the bottom is a large green button labeled 'Sign up for GitHub'. A note below the button states: 'By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails.'

Username

Pick a username

Email

you@example.com

Password

Create a password

Use at least one letter, one numeral, and seven characters.

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails.

3. On the Welcome to GitHub screen, select **Unlimited public repositories free** under **Choose your personal plan**, and select **Continue**.

Welcome to GitHub

You've taken your first step into a larger world, @ [REDACTED].

Completed Set up a personal account	Step 2: Choose your plan	Step 3: Tailor your experience
--	-----------------------------	-----------------------------------

Choose your personal plan

Unlimited public repositories for free.

Unlimited private repositories for \$7/month.

Don't worry, you can cancel or upgrade at any time.

Help me set up an organization next
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.
[Learn more about organizations](#)

Send me updates on GitHub news, offers, and events
Unsubscribe anytime in your email preferences. [Learn more](#)

Continue

Both plans include:

- ✓ Collaborative code review
- ✓ Issue tracking
- ✓ Open source community
- ✓ Unlimited public repositories
- ✓ Join any organization

4. On the next screen, you can select options to tailor your experience and select **Submit**, or select **skip this step**, next to **Submit**, to complete your registration.

Welcome to GitHub

You'll find endless opportunities to learn, code, and create, @ [REDACTED].

Completed Set up a personal account	Step 2: Choose your plan	Step 3: Tailor your experience
--	-----------------------------	-----------------------------------

How would you describe your level of programming experience?

Very experienced Somewhat experienced Totally new to programming

What do you plan to use GitHub for? (check all that apply)

School projects Project Management Research
 Design Development Other (please specify)

Which is closest to how you would describe yourself?

I'm a student I'm a hobbyist I'm a professional
 Other (please specify)

What are you interested in?

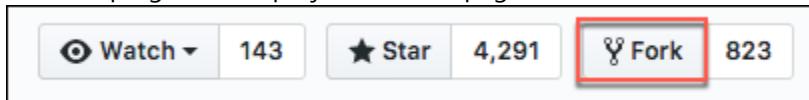
[REDACTED]
e.g. tutorials, android, ruby, web-development, machine-learning, open-source

Submit [skip this step](#)

Task 5: Fork the starter app

In this task, you will fork the starter application to create a copy in your GitHub account.

1. Log into your GitHub account, then navigate to the GitHub repository for the mcw-oss-paas-devops starter app, [here](#).
2. At the top right of the projects GitHub page, select **Fork**.



3. If you have more than one GitHub account, select the account to which the project should be forked.
4. This will start the process of making a copy of the starter application into your GitHub account, in a repository named **mcw-oss-paas-devops**.
5. Once completed the project page will open.

No description, website, or topics provided. [Edit](#)

Add topics

18 commits 1 branch 0 releases 1 contributor

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

This branch is even with :master. [Pull request](#) [Compare](#)

File	Description	Time Ago
AzureFunctions	Added Azure function code files to the project. Modified the user rou...	3 days ago
bin	First commit	11 days ago
data	Updated readme, and order process to redirect to a thank you page. Ad...	2 days ago
models	Model changes	4 days ago
public	Added Azure Function code	4 days ago
routes	Added Azure function code files to the project. Modified the user rou...	3 days ago
src	Initial commit	14 minutes ago
.gitignore	First commit	11 days ago
README.md	Updated readme, and order process to redirect to a thank you page. Ad...	2 days ago
app.js	Initial commit	14 minutes ago
package-lock.json	First commit	11 days ago
package.json	Added seed.js to populate the database with seed data.	6 days ago

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

Exercise 1: Run starter application

Duration: 30 minutes

In this exercise, you will create a local copy of the starter application on your Lab VM, add some sample data to the local MongoDB database, and run the application.

Task 1: Connect to your Lab VM

In this task, you will create an RDP connection to your Lab VM. If you are already connected, skip to [Task 2](#).

1. Navigate to the Azure portal and select **Resource groups** from the left-hand menu, then select the **hands-on-lab-SUFFIX resource group** from the list. If there are too many, enter "hands-on-lab" into the filter box to reduce the resource groups displayed in the list.

The screenshot shows the Azure Resource Groups blade. On the left, there's a sidebar with options like 'Create a resource', 'All services', 'FAVORITES', 'Dashboard', 'All resources', 'Resource groups' (which is highlighted with a red box), and 'App Services'. The main area is titled 'Resource groups' and shows a list of items under 'Subscriptions: hands-on-lab'. A single item, 'hands-on-lab', is listed with its name and a small icon. The entire row for 'hands-on-lab' is also highlighted with a red box.

2. Next, select **LabVM** from the list of available resources.

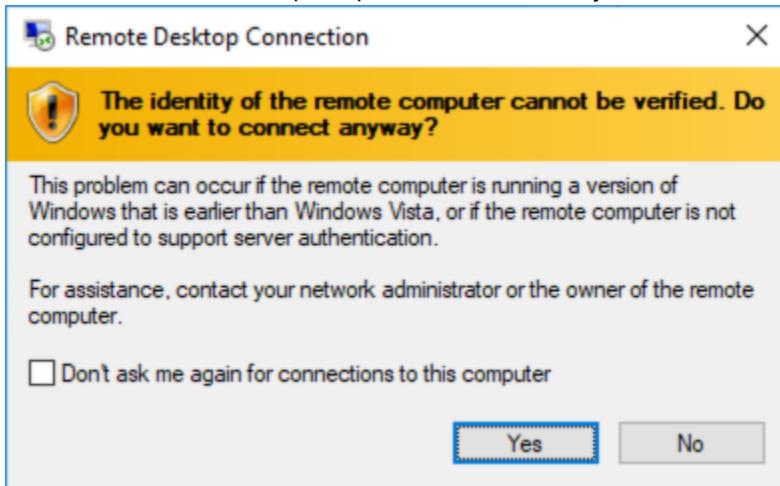
NAME	TYPE	LOCATION	...
hands-on-labs-vnet	Virtual network	West US	...
LabVM	Virtual machine	West US	...
LabVM_OsDisk_1_540ee19fd3454989b79d583ce2fec339	Disk	West US	...
labvm459	Network interface	West US	...
LabVM-ip	Public IP address	West US	...
LabVM-nsg	Network security group	West US	...

3. On the **LabVM** blade, copy the Public IP address from the Essentials area on the Overview screen.

The screenshot shows the 'LabVM' blade in the Azure portal. At the top, there are several actions: 'Connect', 'Start', 'Restart', 'Stop', 'Capture', 'Move', 'Delete', and 'Refresh'. Below these, there's a summary section with details about the resource group ('hands-on-lab'), status ('Running'), location ('West US 2'), and subscription ('Subscription (change)'). To the right of this summary, there's a table with columns for 'Computer name' (LabVM), 'Operating system' (Linux), 'Size' (Standard D2s v3 (2 vcpus, 8 GB memory)), and 'Public IP address' (40.125.66.242). The 'Public IP address' field is highlighted with a red box.

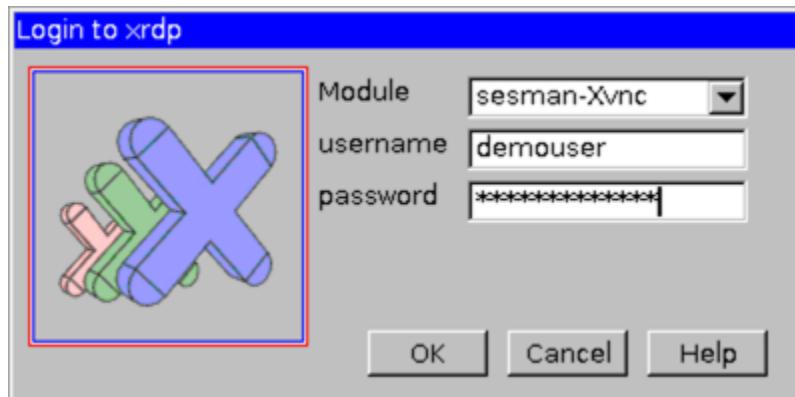
4. Open a Remote Desktop Client (RDP) application and enter or paste the Public IP address of your Lab VM into the computer name field.
5. Select **Connect** on the Remote Desktop Connection dialog.

6. Select **Yes** to connect, if prompted that the identity of the remote computer cannot be verified.



7. Enter the following credentials (or the non-default credentials if you changed them):

- User name:** demouser
- Password:** Password.1!!



8. Select **OK** to log into the Lab VM.

Task 2: Grant permissions to Docker

In this task, you will grant permissions to the demouser account to access the Unix socket needed to communicate with the Docker engine.

1. On your Lab VM, open a bash shell.
2. At the command prompt, enter the following command:

```
sudo usermod -a -G docker $USER
```

3. After running the command, you will need **completely log out of the Lab VM** and log back in (if in doubt, reboot).
4. After logging back in, run the following command to test that the demouser account has proper permissions:

```
docker run hello-world
```

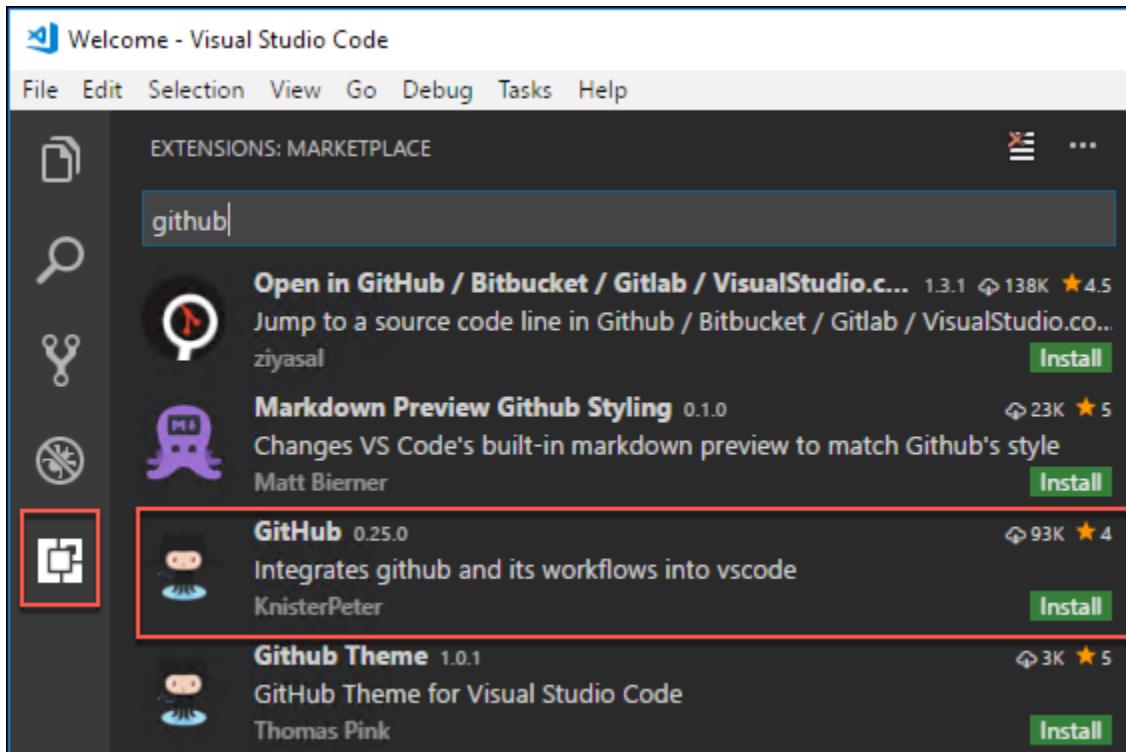
```
demouser@LabVM:~$ sudo usermod -a -G docker $USER
sent invalidate(passwd) request, exiting
sent invalidate(group) request, exiting
demouser@LabVM:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:97ce6fa4b6cdc0790cda65fe7290b74cfecd9fa0c9b8c38e979330d547d22ce1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

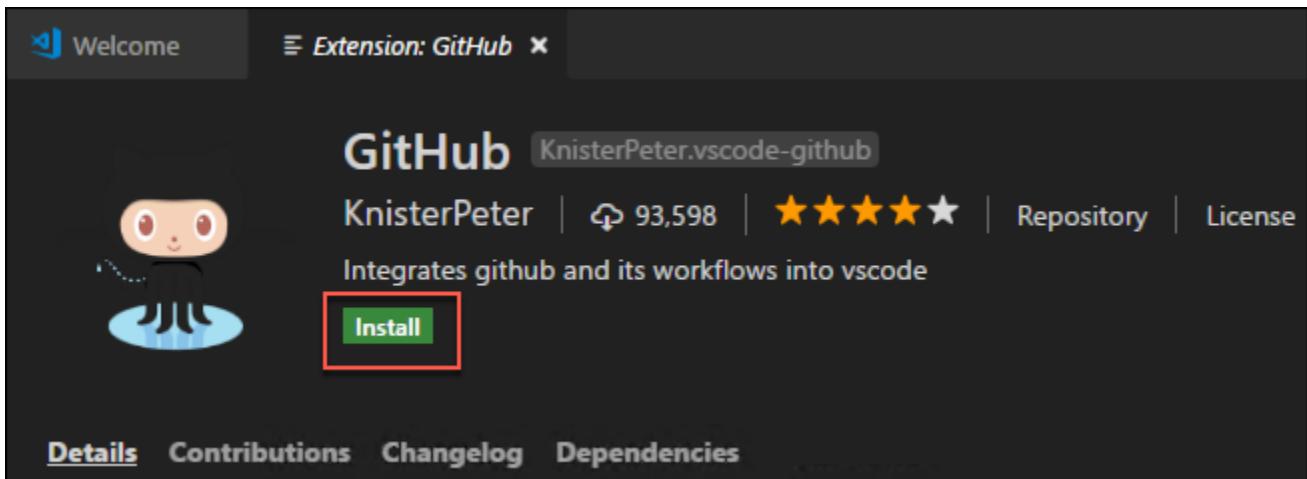
Task 3: Integrate GitHub into VS Code

In this task, you will install the GitHub extension in VS Code, and configure a service integration with your GitHub account. This integration will allow you to push your code changes to GitHub directly from VS Code.

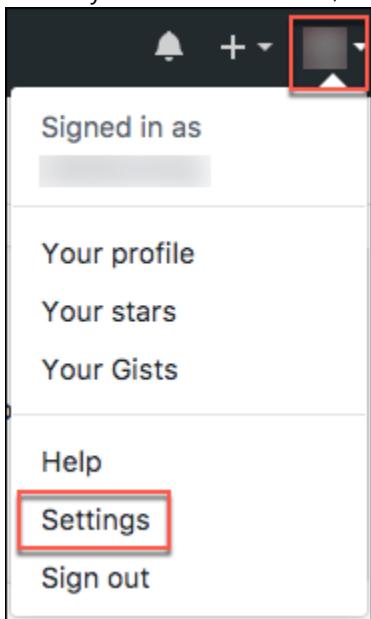
1. On your Lab VM, open **VS Code**.
2. In VS Code, select the **Extensions** icon from the left-hand menu, enter "github" into the **Extensions** search box, and select the **GitHub** extension.



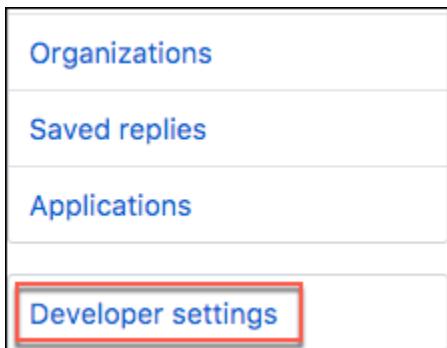
3. Select **Install** in the Extension: GitHub window.



4. Close VS Code. Note: VS Code must be restarted to enable the extension.
5. To connect VS Code with your GitHub account, you need to generate a Personal access token.
6. Open a browser window and navigate to your GitHub account (<https://github.com>).
7. Within your GitHub account, select **your user profile icon** in the top right, then select **Settings** from the menu.



8. On the Settings screen, select **Developer settings** at the bottom of the Personal settings menu on the left-hand side of the screen.



9. On the Developer settings page, select **Personal access tokens** from the left-hand menu.

A screenshot of the GitHub Developer settings sidebar. It shows three options: 'OAuth Apps', 'GitHub Apps', and 'Personal access tokens'. The 'Personal access tokens' option is highlighted with a red border.

10. Select **Generate new token**.

A screenshot of the 'Personal access tokens' generation screen. It has a title 'Personal access tokens' and two buttons: 'Generate new token' (highlighted with a red border) and 'Revoke all'.

Tokens you have generated that can be used to access the [GitHub API](#).

11. Enter a token description, such as "VS Code Integration", and check the box next to **repo** under **Select scopes**. This will select all the boxes under it.

A screenshot of the 'New personal access token' creation screen. It includes a description of what personal access tokens are, a 'Token description' input field containing 'VS Code Integration', a 'What's this token for?' section, and a 'Select scopes' section.

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

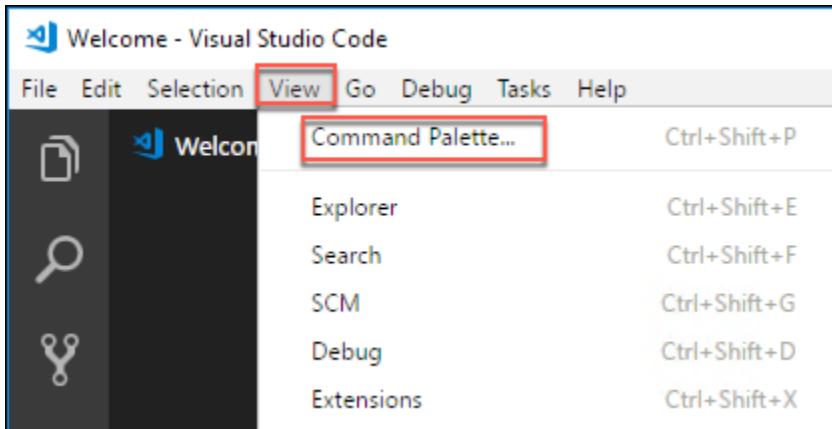
<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations

12. Select **Generate token** near the bottom of the screen.

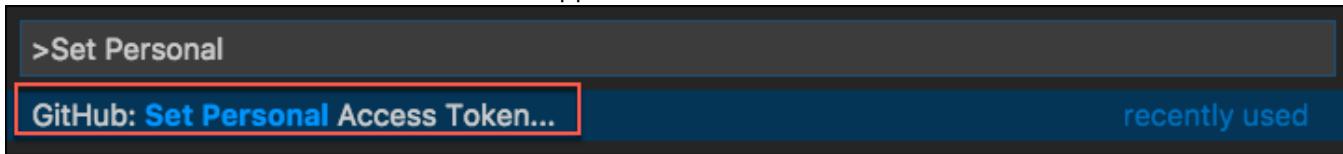
A screenshot of the generated token list. It shows a single token: 'dabe08e1952a0e30f060a25247328bfde643621a' with a copy icon (highlighted with a red border), and 'Edit' and 'Delete' buttons.

14. Open **VS Code** on your Lab VM.

15. Select the **View** menu, then select **Command Palette...** from the menu.



16. In the box that appears at the top center of the VS Code window, enter "Set Personal Access Token," then select **GitHub: Set Personal Access Token**, when it appears.



17. Paste the Personal access token you copied from GitHub into the box, and press **Enter**.



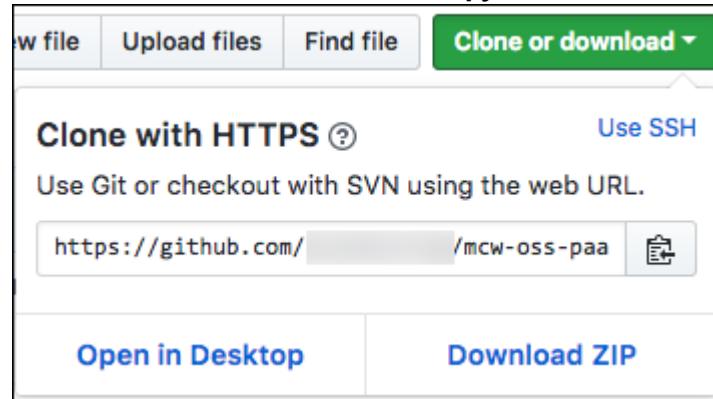
18. VS Code is now connected to your GitHub account.

19. Close VS Code.

Task 4: Clone the starter application

In this task, you will clone the starter application, creating a local copy on your Lab VM.

1. On your Lab VM, open a browser, and navigate to your GitHub account. (<https://github.com>)
2. Within your GitHub account, navigate to the forked copy of the *mcw-oss-paas-devops* application page, select **Clone or download**, then select the **copy** link next to the web URL.

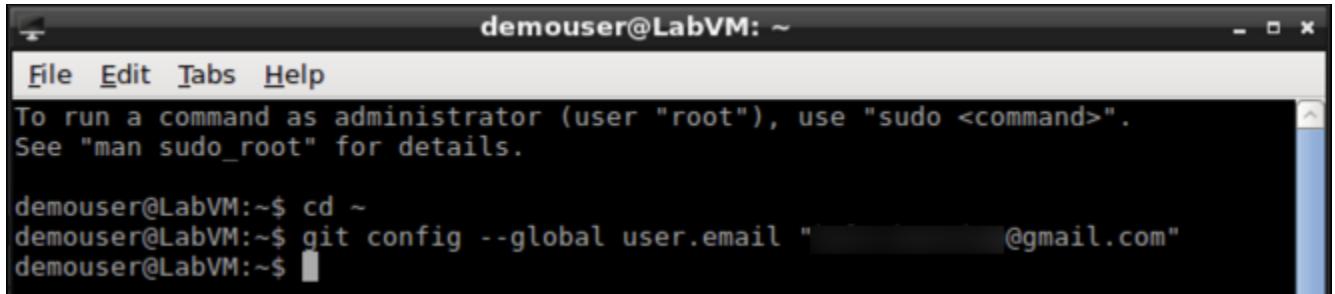


3. Open a new bash shell, and enter the following command:

```
cd ~
```

4. Next, enter the following command, replacing [EMAIL] with the email address you used when creating your GitHub account. This will associate your git username with the commits made from the Lab VM.

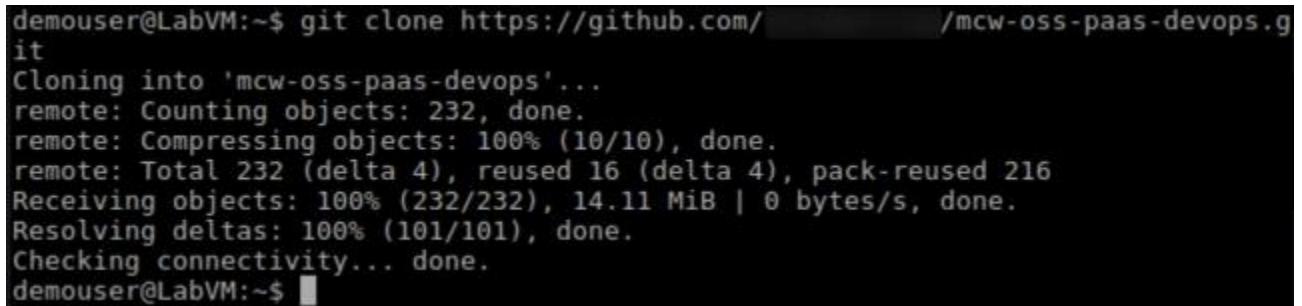
```
git config --global user.email "[EMAIL]"
```



A screenshot of a terminal window titled "demouser@LabVM: ~". The window shows the command "git config --global user.email" being run, followed by a redacted email address and "@gmail.com". A message at the top of the terminal indicates that to run a command as administrator (user "root"), use "sudo <command>".

5. At the prompt, enter the following command, replacing [CLONE_URL] with URL you copied from GitHub in step 2 above:

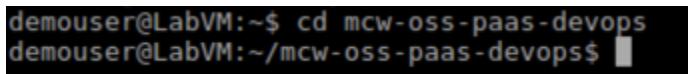
```
git clone [CLONE_URL]
```



A screenshot of a terminal window titled "demouser@LabVM: ~\$". The window shows the command "git clone https://github.com/" followed by a redacted URL and "/mcw-oss-paas-devops.git". The output of the command shows the cloning process, including counting objects, compressing objects, receiving objects, and resolving deltas, all completed successfully.

6. Now, change the directory to the cloned project by entering the following at the prompt:

```
cd mcw-oss-paas-devops
```

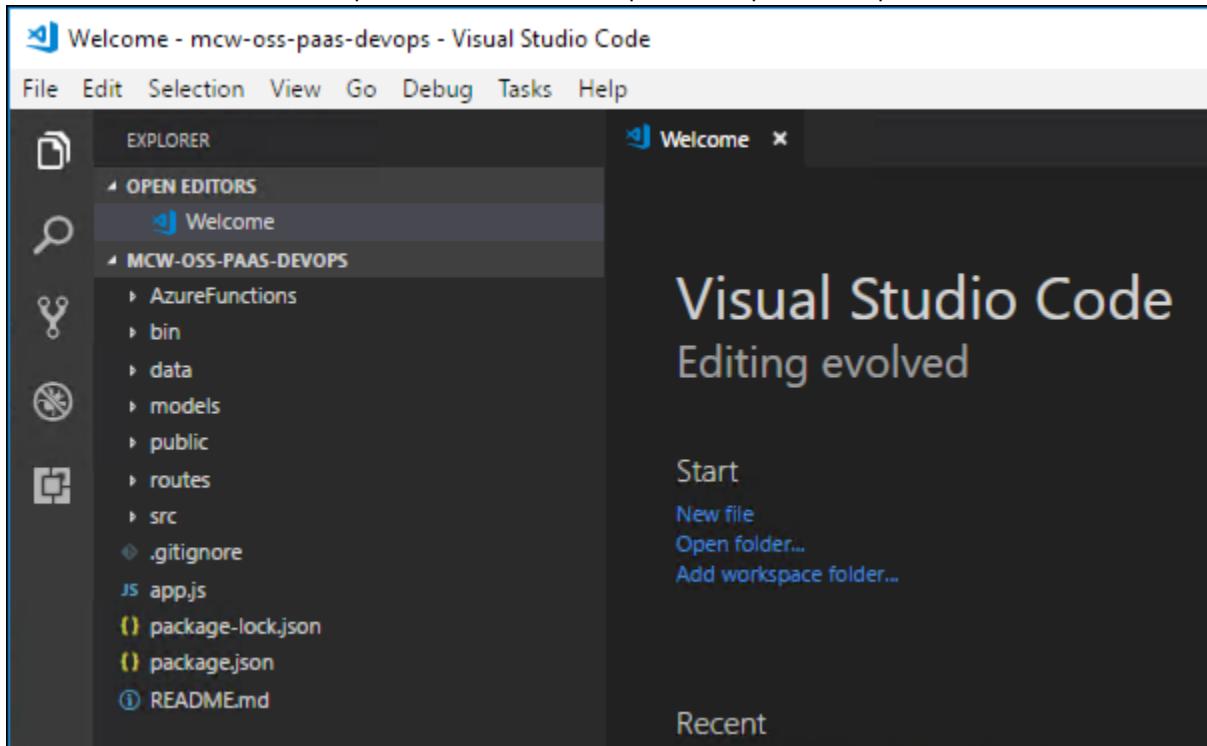


A screenshot of a terminal window titled "demouser@LabVM: ~\$ cd mcw-oss-paas-devops". The window shows the command "cd mcw-oss-paas-devops" being run, followed by a redacted path and the final prompt "demouser@LabVM: ~/mcw-oss-paas-devops\$".

7. Finally, issue a command to open the starter project in VS Code by typing:

```
code .
```

8. A new VS Code window will open, with the mcw-oss-paas-devops folder opened.

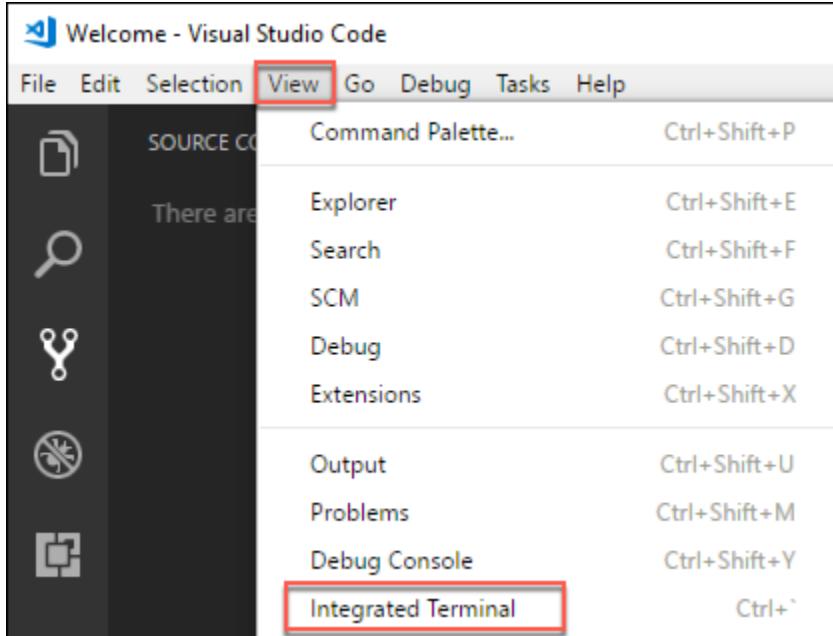


9. You are now ready to begin working with the project in VS Code.

Task 5: Launch the starter application

In this task, you will seed the MongoDB with sample data, then run the application locally, connected to your MongoDB instance. This task is to verify the connection to MongoDB and that it contains the seeded plan data, before we migrate the application and data to Azure Cosmos DB.

1. Return to VS Code, select **View** from the menu, and select **Integrated Terminal**.



2. This will open a new bash terminal window at the bottom of the VS Code dialog.

3. At the bash prompt, enter:

```
npm install
```

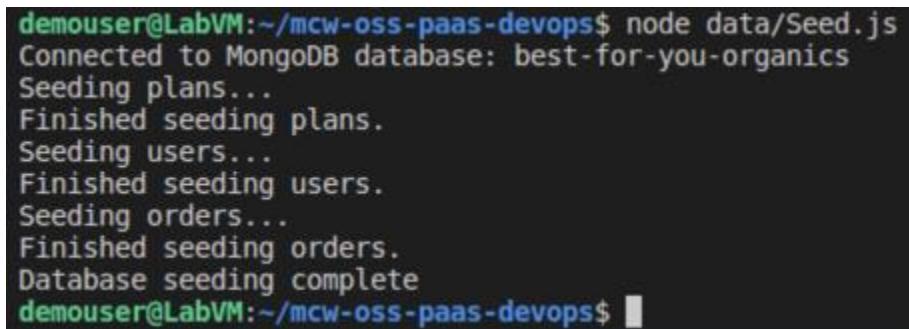


To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo root" for details.

```
demouser@LabVM:~/mcw-oss-paas-devops$ npm install
> uglifyjs-webpack-plugin@0.4.6 postinstall /home/demouser/mcw-oss-paas-devops/node_modules/uglifyjs-webpack-plugin
> node lib/post install.js
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.2 (node modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
added 1212 packages in 27.762s
```

4. Next, enter the following to seed the local MongoDB database with plans, user accounts, and orders.

```
node data/Seed.js
```



```
demouser@LabVM:~/mcw-oss-paas-devops$ node data/Seed.js
Connected to MongoDB database: best-for-you-organics
Seeding plans...
Finished seeding plans.
Seeding users...
Finished seeding users.
Seeding orders...
Finished seeding orders.
Database seeding complete
demouser@LabVM:~/mcw-oss-paas-devops$
```

5. Next, build the application using the following:

```
npm run build
```

6. Finally, enter the following to launch the application.

```
npm start
```

7. Open a browser and navigate to <http://localhost:3000> to view the landing page of the starter application. You will see three plans listed on the application home page, which are pulled from the local MongoDB database.

The screenshot shows a web application interface for meal delivery plans. At the top, there is a navigation bar with a logo, a "Home" link, and a "Sign In" button. Below the navigation bar, there are three separate boxes, each representing a meal plan:

- Two Person Plan**:
 - 1-2 Person**
 - 3 Unique meals per week
 - Our basic plan, delivering 3 meals per week, which will feed 1-2 people.
 - \$72/Week**
 - Select this plan**
- Four Person Plan**:
 - 3-4 Person**
 - 3 Unique meals per week
 - Our family plan, delivering 3 meals per week, which will feed 3-4 people.
 - \$87/Week**
 - Select this plan**
- High-Pro Plan**:
 - 1-2 Person**
 - 3 High protein meals per week
 - Specialty formulated for athletes and active individuals, delivering 3 meals per week, for 1-2 people.
 - \$80/Week**
 - Select this plan**

8. Return to the VS Code integrated terminal window, and press **CTRL+C** to stop the application.

Exercise 2: Migrate the database to Cosmos DB

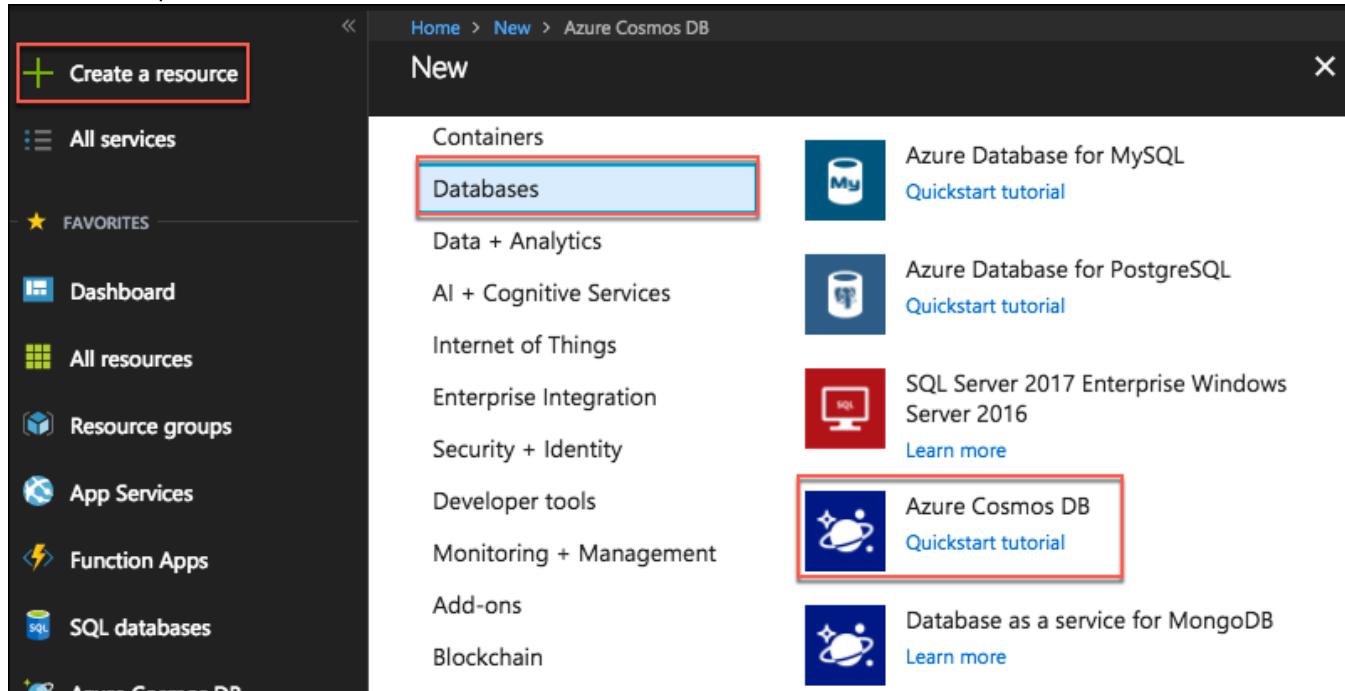
Duration: 30 minutes

In this exercise, you will provision an Azure Cosmos DB account, and then update the starter application's database connection string to point to your new Azure Cosmos DB account. You will then, use *mongoimport.exe* to migrate the data in your MongoDB database into Cosmos DB collections, and verify with the application that you are connected to your Cosmos DB database.

Task 1: Provision Cosmos DB using the MongoDB API

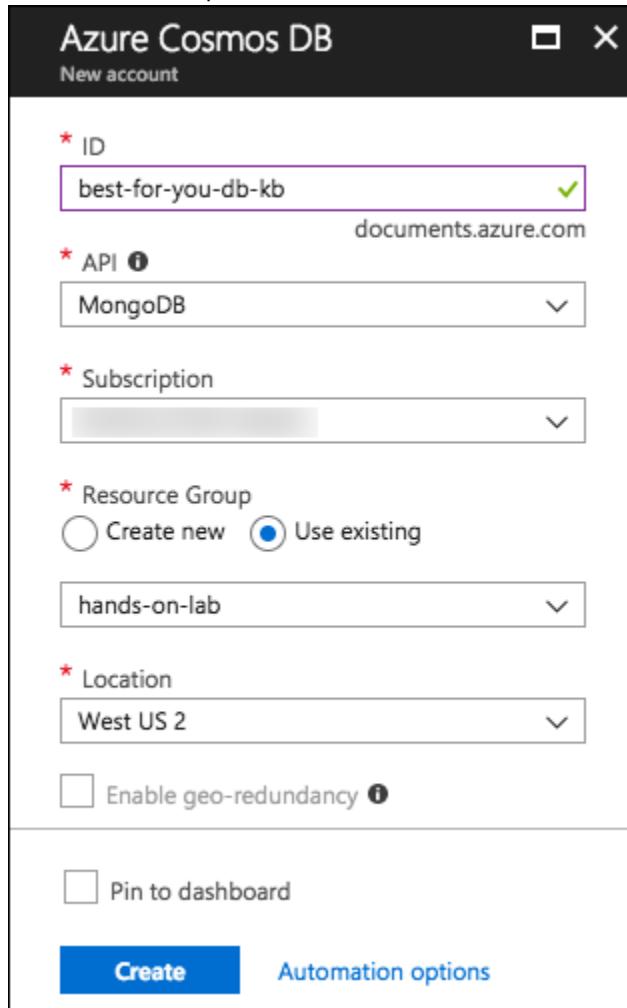
In this task, you will provision a new Azure Cosmos DB account using the MongoDB API.

1. In the Azure portal, select **+Create a resource, Databases**, the select **Azure Cosmos DB**.



2. On the **Azure Cosmos DB** blade, enter the following:
 - a. **ID:** Enter "best-for-you-db-SUFFIX," where SUFFIX is your Microsoft alias, initials, or another value to ensure the name is unique (indicated by a green check mark).
 - b. **API:** Select **MongoDB**.
 - c. **Subscription:** Select the subscription you are using for this hands-on lab.
 - d. **Resource Group:** Select **Use existing** and choose the **hands-on-lab-SUFFIX resource group** you created previously.
 - e. **Location:** Select a location near you from the list (Note: not all locations are available for Cosmos DB)
 - f. **Enable geo-redundancy:** Unchecked

- g. Select **Create** to provision the new Azure Cosmos DB account



Task 2: Update database connection string

In this task, you will retrieve the connection string for your Azure Cosmos DB database and update the starter application's database connection string.

1. When your Cosmos DB account is provisioned, navigate to it in the Azure portal.

2. On the **Azure Cosmos DB account** blade, select **Connection String** under **SETTINGS** in the left-hand menu, and copy the **PRIMARY connection string**.

The screenshot shows the 'best-for-you-db - Connection String' blade in the Azure portal. The left sidebar has 'SETTINGS' selected, and 'Connection String' is highlighted. The main area shows connection details: HOST (best-for-you-db.documents.azure.com), PORT (10255), USERNAME (best-for-you-db), PRIMARY PASSWORD (redacted), SECONDARY PASSWORD (redacted). The 'PRIMARY CONNECTION STRING' field contains the value 'mongodb://best-for-you-db:P1aUUP7hlAwjppfeUHGTS0HaTJyTq8f646eMkPOUZkeX8s1YMpq7yOimIZUTd7pfPkGL9D9k0ueE4oe2tyjGg=='. This field is also highlighted with a red box.

3. Return to VS Code.
4. Open app.js from the root directory of the application and locate the line that starts with var **databaseUrl1**.

The screenshot shows the VS Code interface with the 'app.js' file open in the editor. The file content is:

```

1  var express = require('express');
2  var path = require('path');
3  var favicon = require('serve-favicon');
4  var logger = require('morgan');
5  var bodyParser = require('body-parser');
6  var session = require('express-session');
7  var mongoStore = require('connect-mongo')(session);
8
9  var order = require('./routes/order');
10 var plan = require('./routes/plan');
11 var user = require('./routes/user');
12 var userSession = require('./routes/session');
13
14 var app = express();
15
16 var databaseUrl1 = 'mongodb://localhost:27017/best-for-you-organics';
17
18 var mongoose = require('mongoose');
19 mongoose.Promise = require('bluebird');
20 mongoose.connect(databaseUrl1, { useMongoClient: true, promiseLibrary: require('bluebird') })
21   .then(() => console.log('connection succesful'))
22   .catch((err) => console.error(err));
23 var db = mongoose.connection;
24

```

The line 'var databaseUrl1 = 'mongodb://localhost:27017/best-for-you-organics';' is highlighted with a red box.

5. Replace the value of the **databaseUrl1** variable with the Cosmos DB connection string you copied from the Azure portal.

```

14  var app = express();
15
16  var databaseUrl1 = 'mongodb://best-for-you:miZiDmNr8TnSAufBvTQsghbYPiQOY69hIHgFhSn7Gf10cvbRLXvqxaherSKY6vQTDrVHH';
17

```

6. Scroll to the end of the value you just pasted in and locate **10255/?ssl=** in the string.

```
15  
16 |  @best-for-you.documents.azure.com 10255/?ssl=true&replicaSet=globaldb'  
17
```

7. Between the "/" and the "?" insert **best-for-you-organics** to specify the database name.

```
15  
16 |  @best-for-you.documents.azure.com:10255/best-for-you-organics?ssl=true&replicaSet=globaldb'  
17
```

8. Save **app.js**.

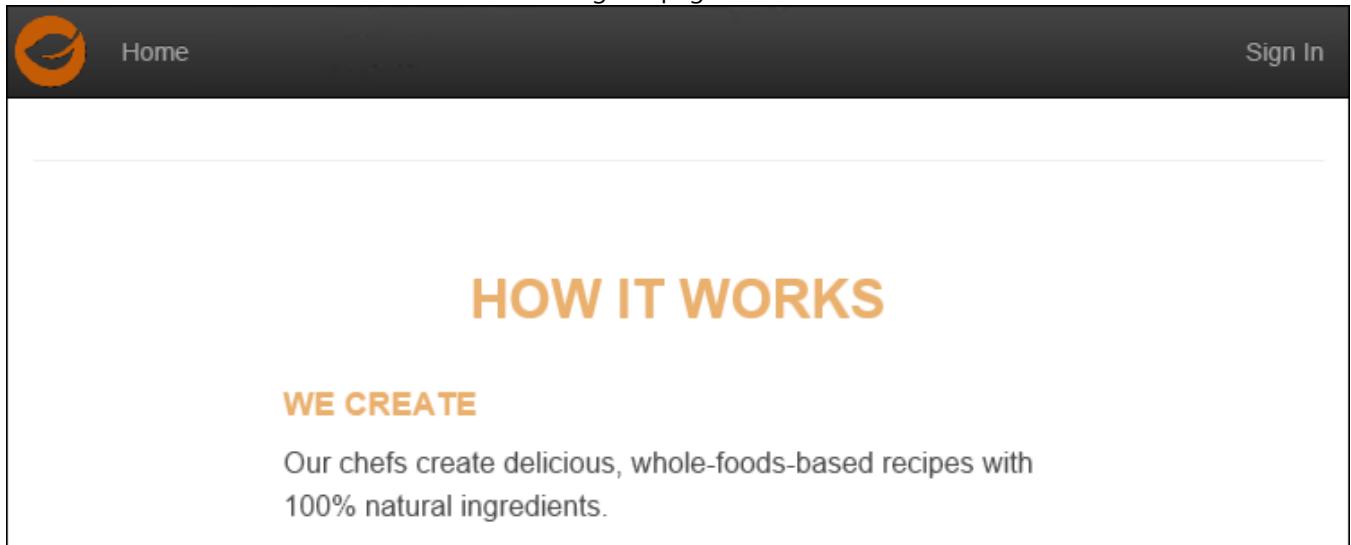
9. In the VS Code integrated terminal, enter the following command to rebuild the application.

```
npm run build
```

10. When the build completes, start the application by typing the following:

```
npm start
```

11. Return to your browser and refresh the application page. Note: You may need to press **CTRL+F5** in your browser window to clear the browser cache while refreshing the page.



12. Notice the three plans that were displayed on the page previously are no longer there. This is because the application is now pointed to your Azure Cosmos DB, and there is no plan data in that database yet.

13. Let's move on to copying the data from the local MongoDB instance into Cosmos DB.

Task 3: Pre-create and scale collections

In this task, you will create the collections needed for your database migration and increase each collections' throughput from the default 1,000 RUs to 2,500 RUs. This is done to avoid throttling during the migration, and reduce the time required to import data.

1. In the Azure portal, navigate to your Azure Cosmos DB account, select **Browse** from the left-hand menu, under **COLLECTIONS**, and select **+Add Collection**.

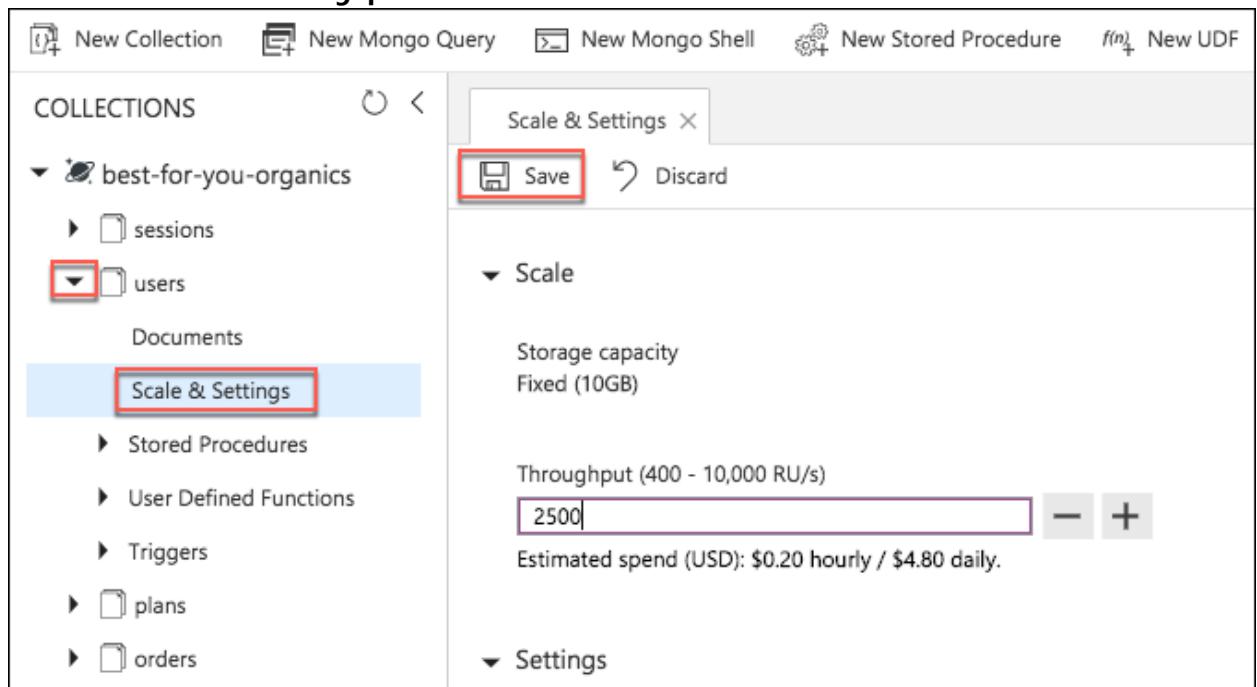
The screenshot shows the 'best-for-you - Browse' page in the Azure portal. On the left, there's a sidebar with 'Search (Ctrl+)', 'Locks', 'Automation script', 'COLLECTIONS' (with 'Browse' highlighted), 'Scale', 'MONITORING', and 'Metrics'. The main area has a header with '+ Add Collection', '+ Add Database', 'Refresh', 'Delete Collection', and 'Delete Database'. Below this is a dropdown for 'Database' set to 'best-for-you-organics'. The main table lists three collections: 'sessions', 'users', and 'plans', all provisioned at 1000 RU/s. A 'TOTAL' row shows 3000 RU/s. At the bottom, a note states: '* This is NOT your bill. This is an estimated hourly rate for the currently provisioned throughput, and the size of the data stored. The estimate DOES NOT take into account any of the discounts you may be eligible for.'

COLLECTION ID	DATABASE	THROUGHPUT (RU/S)	STORAGE	EST. HOURLY COST (USD)*
sessions	best-for-you-organics	1000	0 kB	0.08
users	best-for-you-organics	1000	0 kB	0.08
plans	best-for-you-organics	1000	0 kB	0.08
TOTAL		3000	0 kB	0.24

2. In the **Add Collection** dialog, enter the following:
 - Database id:** Select **Create new**, and enter "best-for-you-organics."
 - Collection Id:** Enter "orders."
 - Storage capacity:** Select **Fixed (10 GB)**.
 - Throughput:** Enter "2500."
 - Select **OK** to create the collection.

The 'Add Collection' dialog box is shown. It has fields for 'Database id' (radio buttons for 'Create new' selected, value 'best-for-you-organics'), 'Collection Id' (value 'orders'), 'Storage capacity' (radio buttons for 'Fixed (10 GB)' selected, 'Unlimited' option available), and 'Throughput (400 - 10,000 RU/s)' (value '2500', with minus and plus buttons for adjustment). A note at the bottom says 'Estimated spend (USD): \$0.20 hourly / \$4.80 daily.' and 'Choose unlimited storage capacity for more than 10,000 RU/s.'

3. If they don't already exist, repeat steps 1 and 2 to create collections for:
 - a. users
 - b. plans
4. If the users and plans collections already exist from your application connecting, edit each collection to increase their throughput using the following steps.
 - a. Expand the existing collection and select **Scale & Settings**.
 - b. Enter "2500" into the **Throughput** box and select **Save**.



Task 4: Import data to the API for MongoDB using mongoimport

In this task, you will use **mongoimport.exe** to import data to your Cosmos DB account. There is a shell script located in the *mcw-oss-paas-devops* solution which handles exporting each collection to a JSON file.

1. Next, you will execute a script file, included in the project files you downloaded, to handle exporting the data out of your MongoDB into .JSON files on the local file system. These files will be used for the import into Cosmos DB.
2. On your Lab VM, open a new integrated bash prompt in VS Code by selecting the + next to the shell dropdown in the integrated terminal pane.



The screenshot shows the VS Code interface with the 'TERMINAL' tab selected. A new terminal window titled '2: bash' is open, indicated by a red box around its title bar. The terminal output shows:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo root" for details.

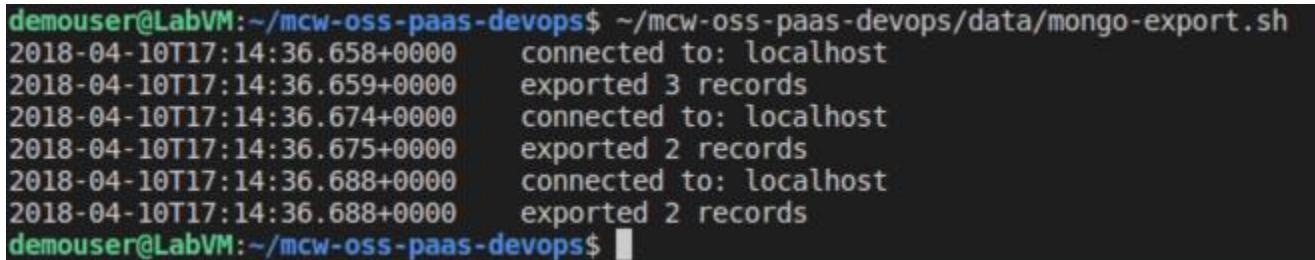
demouser@LabVM:~/mcw-oss-paas-devops$
```

3. At the prompt, enter the following command to grant execute permissions on the export script:

```
chmod +x ~/mcw-oss-paas-devops/data/mongo-export.sh
```

4. Next, run the script by entering the following command:

```
~/mcw-oss-paas-devops/data/mongo-export.sh
```



```
demouser@LabVM:~/mcw-oss-paas-devops$ ~/mcw-oss-paas-devops/data/mongo-export.sh
2018-04-10T17:14:36.658+0000      connected to: localhost
2018-04-10T17:14:36.659+0000      exported 3 records
2018-04-10T17:14:36.674+0000      connected to: localhost
2018-04-10T17:14:36.675+0000      exported 2 records
2018-04-10T17:14:36.688+0000      connected to: localhost
2018-04-10T17:14:36.688+0000      exported 2 records
demouser@LabVM:~/mcw-oss-paas-devops$
```

5. The script creates the folder ~MongoExport, and exports each of the collections in your MongoDB to JSON files.
6. You are now ready to import the data into Azure Cosmos DB using mongoimport.exe.
7. Use the following command template for executing the data import:

```
mongoimport --host <your_hostname>:10255 -u <your_username> -p <your_password> --db <your_database> --collection <your_collection> --ssl --sslAllowInvalidCertificates --type json --file ~/MongoExport/<your_collection>.json
```

8. To get the values needed for the template command above, return to the **Connection string** blade for your Azure Cosmos DB account in the Azure portal. Leave this window up, as will need the **HOST**, **USERNAME**, and

PRIMARY PASSWORD values from this page to import data to your Cosmos DB account.

The screenshot shows the 'best-for-you-db - Connection String' blade in the Azure portal. The left sidebar lists navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Data Explorer, and SETTINGS. Under SETTINGS, 'Connection String' is selected and highlighted in blue. The main content area displays connection information:

- HOST:** best-for-you-db.documents.azure.com
- PORT:** 10255
- USERNAME:** best-for-you-db
- PRIMARY PASSWORD:** P1aUUP7hlAwjppfeUHGTSD0HaTJyTq8f646eMkPOUZkeX8s1YMpq7yOimIZUTd7pfPkGL9D9k0ueE4oe2tyJGg==
- SECONDARY PASSWORD:** ZoadTbw5GfG2UMVaewJa7QQbnMJLPKh7EijwHdW8iE96qh3RoExqwlibeHLG9iSPLbeMwPFWbHbipA9LHu6xQ==
- PRIMARY CONNECTION STRING:** mongodb://best-for-you-db:P1aUUP7hlAwjppfeUHGTSD0HaTJyTq8f646eMkPOUZkeX8s1YMpq7yOimIZUTd7pfPkGL9D9k0u...
- SECONDARY CONNECTION STRING:** mongodb://best-for-you-db:ZoadTbw5GfG2UMVaewJa7QQbnMJLPKh7EijwHdW8iE96qh3RoExqwlibeHLG9iSPLbeMwPFW...
- SSL:** true

A note at the bottom states: "Azure Cosmos DB has strict security requirements and standards. Azure Cosmos DB accounts require authentication and secure communication via SSL."

9. Replace the values as follows in the template command above:

- <your_hostname>: Copy and paste the **Host** value from your **Cosmos DB Connection String** blade.
- <your_username>: Copy and paste the **Username** value from your **Cosmos DB Connection String** blade.
- <your_password>: Copy and paste the **Primary Password** value from your **Cosmos DB Connection String** blade.
- <your_database>: Enter "best-for-you-organics."
- <your_collection>: Enter "plans" (Note there are two instances of <your-collection> in the template command).

10. Your final command should look something like:

```
mongoimport --host best-for-you-db.documents.azure.com:10255 -u best-for-you-db -p
miZiDmNrn8TnSAufBvTQsghbYPiQ0Y69hIHgFhSn7Gf10cvbRLXvqxaherSKY6vQTDrvHHqYyICP40cLncqWew== --
--db best-for-you-organics --collection plans --ssl --sslAllowInvalidCertificates --type json
--file ~/MongoExport/plans.json
```

11. Copy and paste the final command at the command prompt to import the plans collection into Azure Cosmos DB.

```
demouser@LabVM:~/mcw-oss-paas-devops$ mongoimport --host best-for-you-db-kb.documents.azure.com:10255 -u best-for-you-db-kb -p GVclmUF
NTsuCUVvZsqYxhfBvmvnFxMh3xh3liQQY5ln5lpE3BPGYhRrgymUeyGdbd9RBMYwPC8PzStfEKVqzHBA== --db best-for-you-organics --collection plans --ssl
--sslAllowInvalidCertificates --type json --file ~/MongoExport/plans.json
2018-04-10T17:32:58.006+0000      connected to: best-for-you-db-kb.documents.azure.com:10255
2018-04-10T17:32:58.035+0000      imported 3 documents
```

12. You will see a message indicating the number of documents imported, which should be 3 for plans.

13. Verify the import by selecting **Data Explorer** in your Cosmos DB account in the Azure portal, expanding plans, and selecting **Documents**. You will see the three documents imported listed.

The screenshot shows the Azure Cosmos DB Data Explorer interface. On the left, the navigation menu includes 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Quick start', and 'Data Explorer' (which is highlighted with a red box). Below these are 'SETTINGS' and 'Connection String'. The main area has tabs for 'New Collection', 'New Mongo Query', 'New Mongo Shell', 'New Stored Procedure', 'New UDF', 'New Trigger', and 'Delete Collection'. Under 'COLLECTIONS', 'best-for-you-organics' is expanded, showing 'sessions', 'users', and 'plans'. 'plans' is also expanded, showing 'Documents' (which is highlighted with a red box), 'Scale & Settings', 'Stored Procedures', 'User Defined Functions', 'Triggers', and 'orders'. In the 'Documents' section, a list of three documents is shown, each with a preview and edit options. The first document's preview is displayed in a code editor-like pane:

```

1 {
2     "_id" : ObjectId("5a64c2cc29339c0540e68d1b"),
3     "name" : "two_person",
4     "friendlyName" : "Two Person Plan",
5     "portionSize" : "1-2 Person",
6     "mealsPerWeek" : "3 Unique meals per week",
7     "price" : 72,
8     "description" : "Our basic plan, delivering 3 meals per week",
9     "__v" : 0
10 }
```

14. Repeat step 8 for the users and orders collections, replacing the <your_collection> values with:

- users

```
mongoimport --host best-for-you-db.documents.azure.com:10255 -u best-for-you-db -p
miZiDmNrn8TnSAufBvTQsghbYPiQ0Y69hIHgFhSn7Gf10cvbRLXvqxaherSKY6vQTDrvHHqYyICP40cLncqWew== --
--db best-for-you-organics --collection users --ssl --sslAllowInvalidCertificates --type
json --file ~/MongoExport/users.json
```

- orders

```
mongoimport --host best-for-you-db.documents.azure.com:10255 -u best-for-you-db -p
miZiDmNrn8TnSAufBvTQsghbYPiQ0Y69hIHgFhSn7Gf10cvbRLXvqxaherSKY6vQTDrvHHqYyICP40cLncqWew== --
--db best-for-you-organics --collection orders --ssl --sslAllowInvalidCertificates --
--type json --file ~/MongoExport/orders.json
```

15. To verify the starter application is now pulling properly from Azure Cosmos DB, return to your browser running the starter application (<http://localhost:3000>), and refresh the page. You should now see the three plans appear

again on the home page. These were pulled from your Azure Cosmos DB database.

The screenshot shows a website interface for meal delivery plans. At the top, there is a navigation bar with a logo, a 'Home' link, and a 'Sign In' button. Below the navigation bar, there are three main sections, each representing a different meal plan:

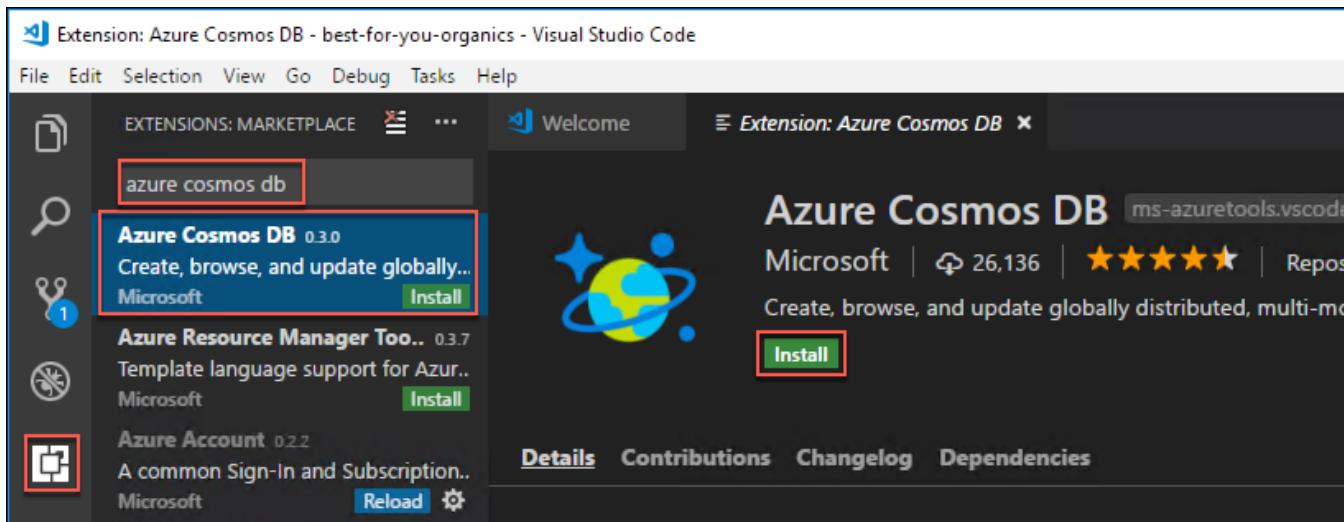
- Two Person Plan**:
 - Icon:** A small orange circular icon with a white leaf-like symbol.
 - Name:** 1-2 Person
 - Description:** 3 Unique meals per week. Our basic plan, delivering 3 meals per week, which will feed 1-2 people.
 - Price:** \$72 /Week
 - Action:** A yellow 'Select this plan' button.
- High-Pro Plan**:
 - Icon:** A small orange circular icon with a white leaf-like symbol.
 - Name:** 1-2 Person
 - Description:** 3 High protein meals per week. Specially formulated for athletes and active individuals, delivering 3 meals per week, for 1-2 people.
 - Price:** \$80 /Week
 - Action:** A yellow 'Select this plan' button.
- Four Person Plan**:
 - Icon:** A small orange circular icon with a white leaf-like symbol.
 - Name:** 3-4 Person
 - Description:** 3 Unique meals per week. Our family plan, delivering 3 meals per week, which will feed 3-4 people.
 - Price:** \$87 /Week
 - Action:** A yellow 'Select this plan' button.

16. You have successfully migrated the application and data to use Azure Cosmos DB with MongoDB APIs.
17. Return to the Integrated terminal window of VS Code which is running the application, and press **CTRL+C** to stop the application.

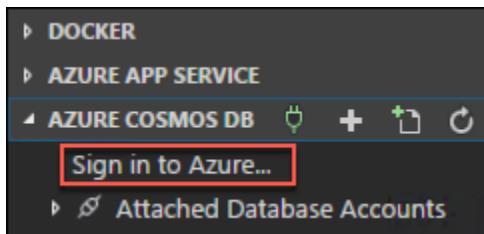
Task 5: Install Azure Cosmos DB extension for VS Code

In this task, you will install the Azure Cosmos DB extension for VS Code, to take advantage of the integration with Azure Cosmos DB. This extension allows you to view and interact with your Cosmos DB databases, collections, and documents directly from VS Code.

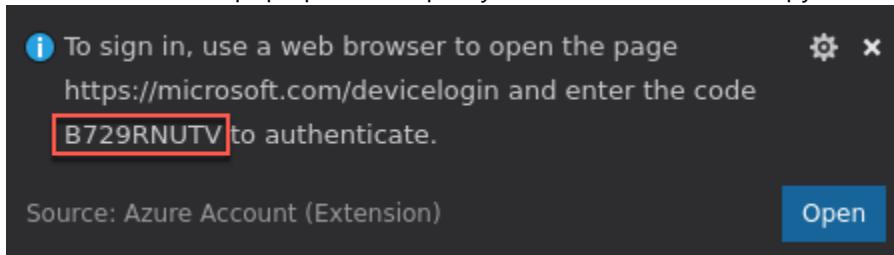
1. Select the **Extensions** icon, enter “cosmos” into the search box, select the **Azure Cosmos DB** extension, and then select **Install** in the Extension: Azure Cosmos DB window.



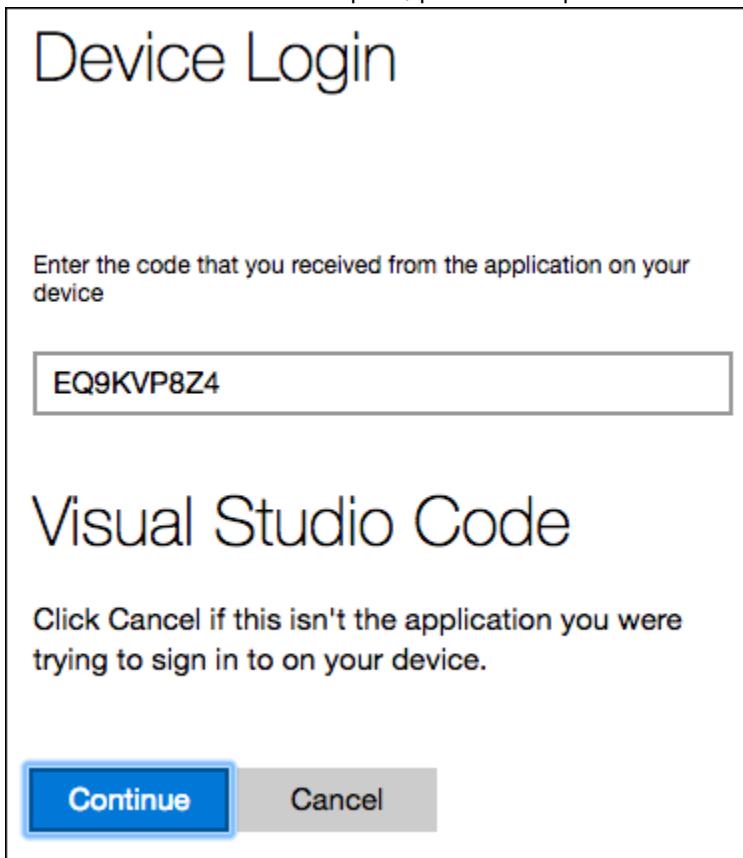
2. Once the extension installation completes, restart VS Code, and reopen the **mcw-oss-paas-devops** project folder.
3. At the bottom left-hand corner of VS Code, you will now see **AZURE COSMOS DB**. Expand that, and select **Sign in to Azure...**



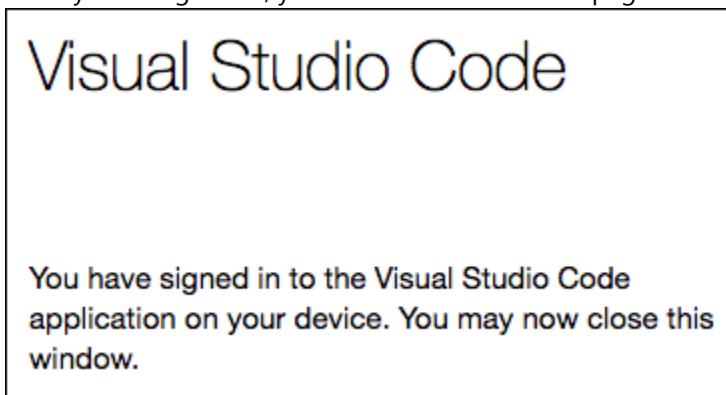
4. An info banner will pop up at the top of your VS Code window. Copy the code provided and select **Open**.



5. In the browser window that opens, paste the copied code into the **Code** box, and select **Continue**.



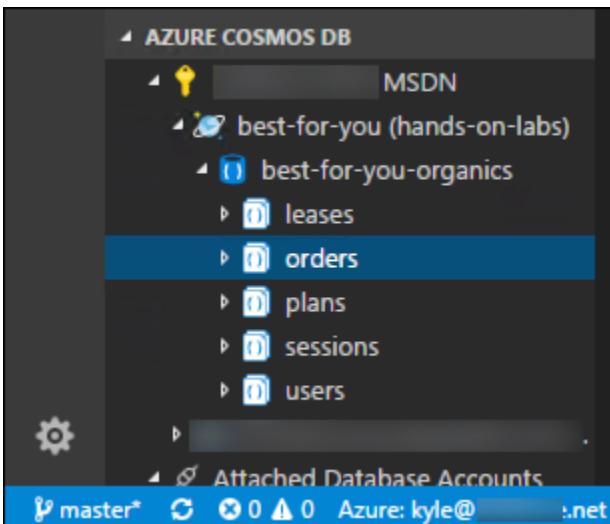
6. On the following screens, log in with your Azure account credentials.
7. Once you've signed in, you will be taken to a new page in the browser. You can close the browser window.



8. If presented with a prompt to enter a password for a new keyring, enter "**Password.1!!**" as the password, and select **Continue**.



9. Back in VS Code, you should now see your Azure account listed under Azure Cosmos DB, along with your Azure account email listed in the status bar of VS Code.



10. From here, you can view your databases, collections, and documents, as well as edit documents directly in VS Code, and push the updated documents back into your database.

Task 6: Decrease collection throughput

In this task, you will decrease the throughput on your collections. Azure Cosmos DB uses an hourly billing rate, so reducing the throughput after the data migration will help save costs.

1. In the Azure portal, navigate to your Azure Cosmos DB account, select **Scale** from the left-hand menu, under **COLLECTIONS**.
2. Expand the **users** collection and select **Scale & Settings**.
3. Change the **Throughput** value to "500," and select **Save**.

The screenshot shows the Azure portal's 'best-for-you - Scale' page for an Azure Cosmos DB account. On the left, there's a navigation menu with 'Automation script', 'Browse', and 'Scale' (which is highlighted with a red box). Below that are 'MONITORING' sections for 'Metrics', 'Alert rules', and 'Diagnostics logs'. The main area shows 'COLLECTIONS' for the 'best-for-you-organics' database. Under 'best-for-you-organics', the 'users' collection is expanded, and its 'Scale & Settings' tab is selected (also highlighted with a red box). On the right, there's a 'Scale & Settings' panel with a 'Save' button (highlighted with a red box) and a 'Discard' button. The 'Scale' section shows 'Storage capacity Fixed (10GB)'. The 'Throughput (400 - 10,000 RU/s)' field contains the value '500' (highlighted with a red box), and below it, the text 'Estimated spend (USD): \$0.040 hourly / \$0.96 daily.' The 'Settings' section is partially visible at the bottom.

4. Repeat steps 2 & 3 for the **plans** and **orders** collections.

Exercise 3: Containerize the app

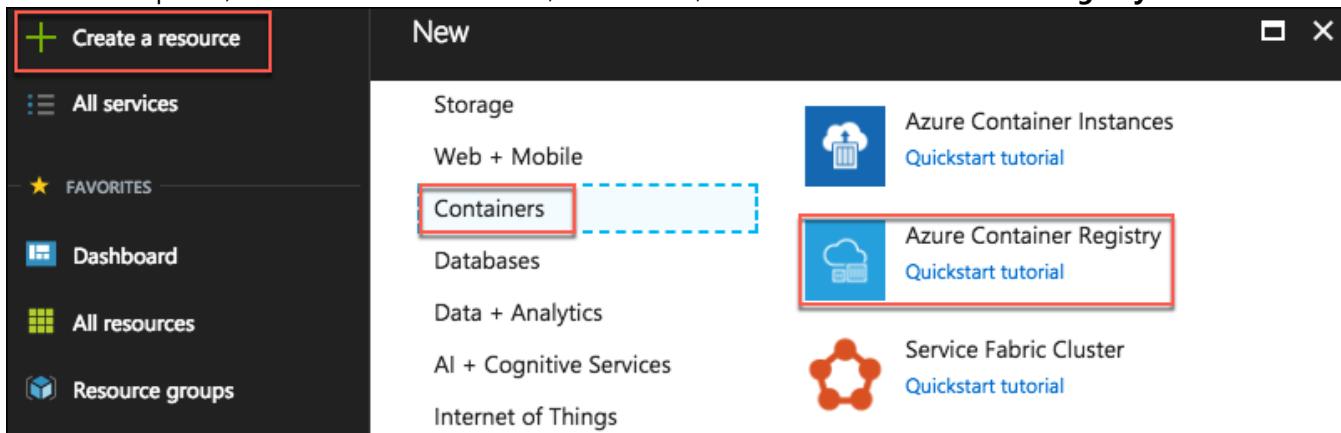
Duration: 30 minutes

This exercise walks you through containerizing an existing MERN application using Docker, pushing the image to an **Azure Container Registry**, then deploying the image to **Web App for Containers** directly from VS Code.

Task 1: Create an Azure Container Registry

In this task, you will be creating a private Docker registry in the Azure portal, so you have a place to store the custom Docker image you will create in later steps.

1. In the Azure portal, select **+Create a resource, Containers**, and select **Azure Container Registry**.



2. On the **Create container registry** blade, enter the following:

- a. **Registry name:** Enter "bestforyouregistrySUFFIX," where SUFFIX is your Microsoft alias, initials, or another value to ensure the name is unique (indicated by a green check mark).
- b. **Subscription:** Select the subscription you are using for this hands-on lab.
- c. **Resource group:** Select **Use existing** and select the **hands-on-lab-SUFFIX** resource group created previously.
- d. **Location:** Select the location you are using for resources in this hands-on lab.
- e. **Admin user:** Select **Enable**.
- f. **SKU:** Select **Basic**.

- g. Select **Create** to provision the new Azure Container Registry

The screenshot shows the 'Create container registry' dialog box. It contains the following fields:

- Registry name:** bestforyouregistry.azurecr.io
- Subscription:** A dropdown menu.
- Resource group:** hands-on-labs
- Location:** West US
- Admin user:** Enable (selected)
- SKU:** Basic
- Pin to dashboard:** An unchecked checkbox.

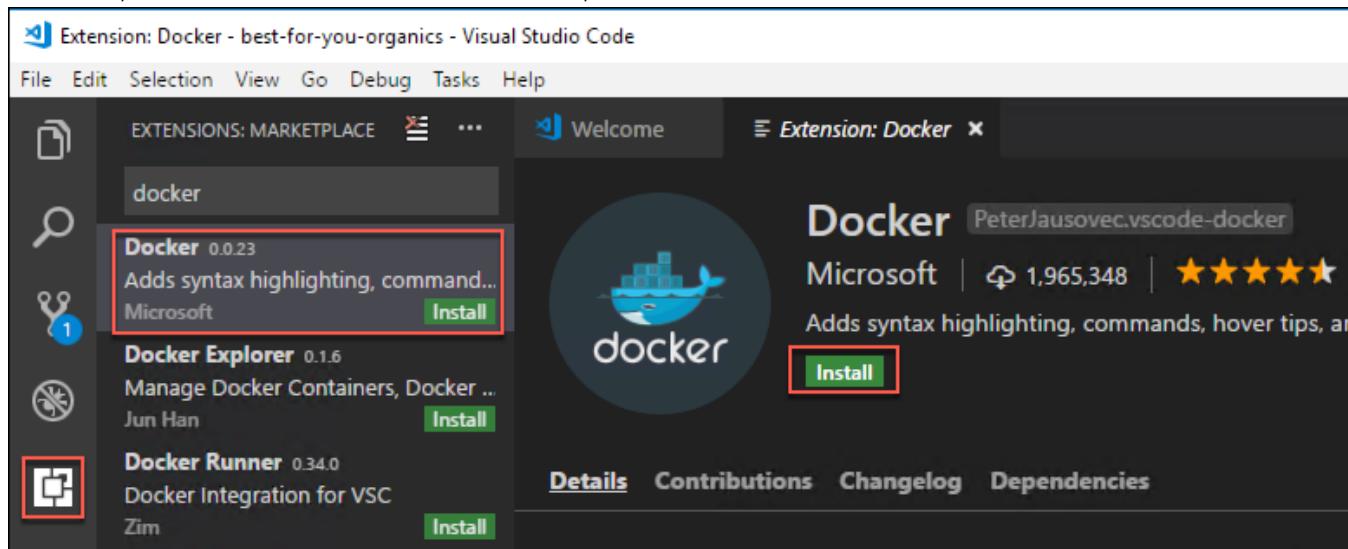
At the bottom are two buttons: **Create** (highlighted in blue) and **Automation options**.

Task 2: Install Docker extension in VS Code

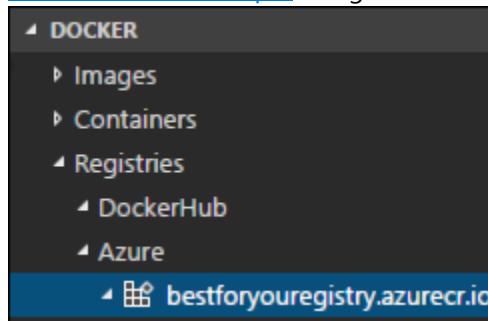
The Docker extension for VS Code is used to simplify the management of local Docker images and commands, as well as the deployment of a built app image to Azure.

1. On your Lab VM, return to VS Code, and the open starter project.

2. Select the **Extensions icon** from the left-hand menu, enter “Docker” into the search box, select the **Docker** extension, and in the **Extension: Docker** window, select **Install**.



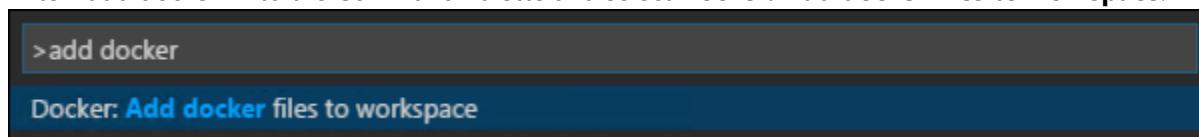
3. Close and reopen VS Code, and the **mcw-oss-paas-devops** project.
 4. Expand the **DOCKER** extension block, then expand **Registries** and **Azure**, and you should see your Azure Container Registry listed. You should already be logged into your Azure subscription, but if not, follow the steps in [Exercise 2, Task 5 Step 3](#) to sign in.



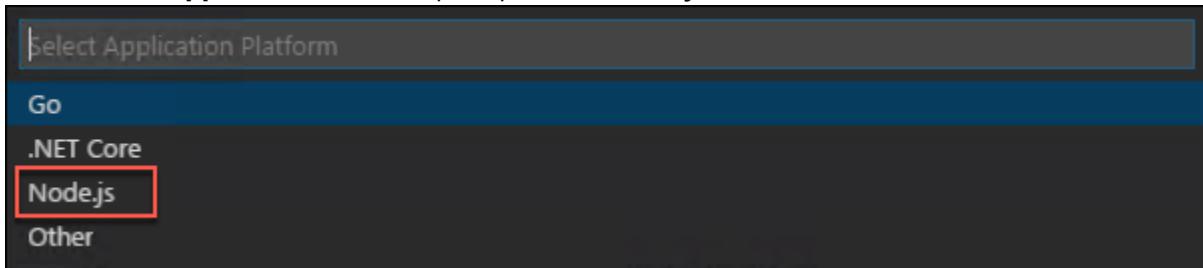
Task 3: Create Docker image and run the app

In this task, you will use VS Code, and the Docker extension, to add the necessary files to the project to create a custom Docker image for the **mcw-oss-paas-devops** app.

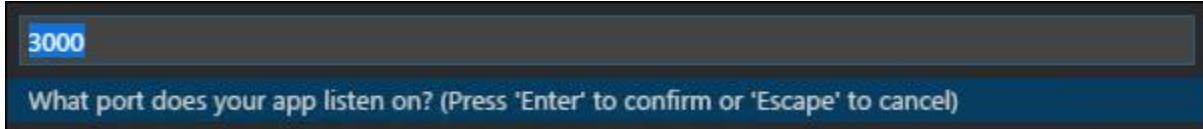
1. On your Lab VM, return to VS Code, and the mcw-oss-paas-devops project.
2. Open the VS Code Command Palette, by selecting **View** from the menu, then **Command Palette**.
3. Enter “add docker” into the Command Palette and select **Docker: Add docker files to workspace**.



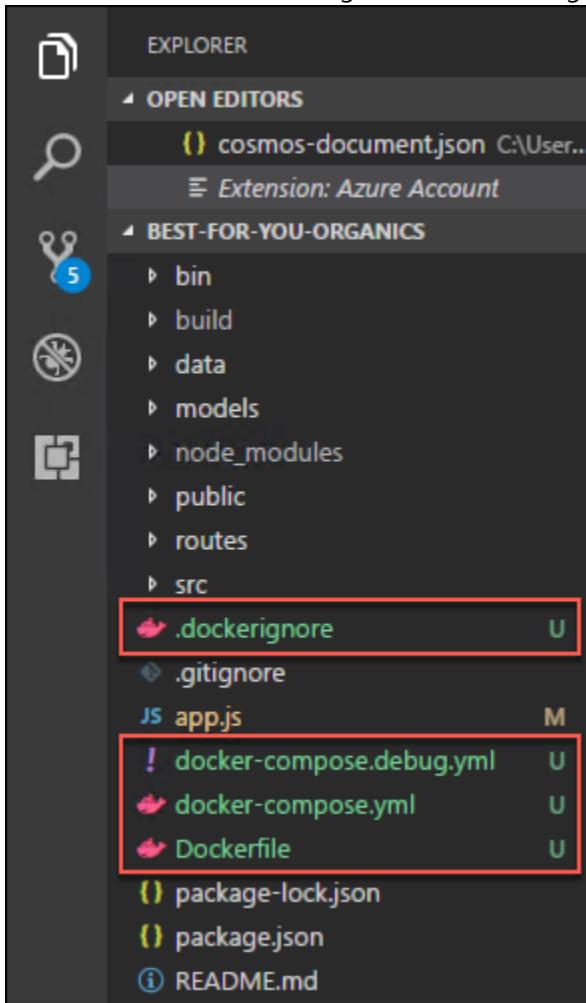
4. At the **Select Application Platform** prompt, select **Node.js**.



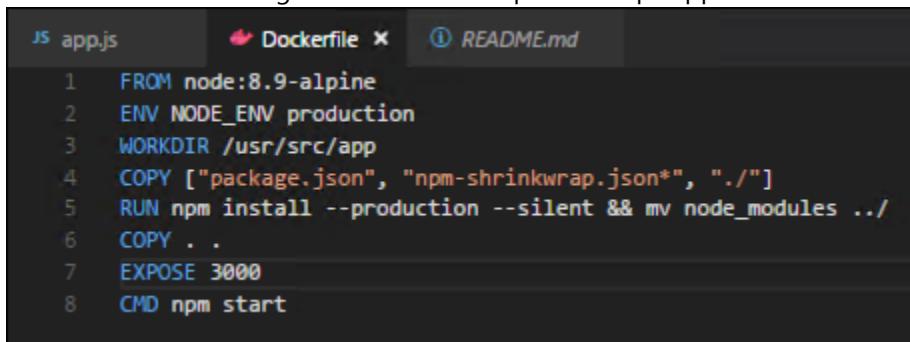
5. Ensure port "3000" is entered on the next screen, and press **Enter**.



6. This will add Dockerfile, along with several configuration files for Docker compose to the project.



7. Select **Dockerfile** from the file navigator and observe the contents. This file provides the commands required to assemble a Docker image for the mcw-oss-paas-devops application.

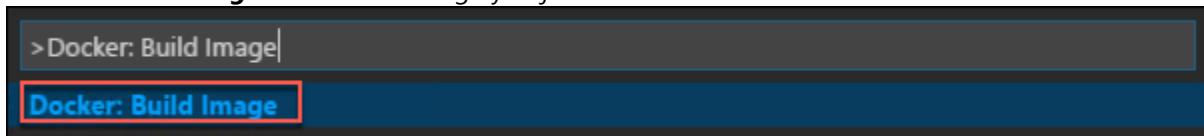


```

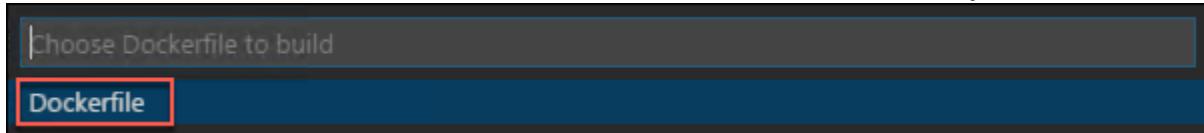
JS app.js      Dockerfile ✘ README.md
1 FROM node:8.9-alpine
2 ENV NODE_ENV production
3 WORKDIR /usr/src/app
4 COPY ["package.json", "npm-shrinkwrap.json*", "./"]
5 RUN npm install --production --silent && mv node_modules ../
6 COPY . .
7 EXPOSE 3000
8 CMD npm start

```

8. Next, you will tell Docker to build and image for your app. Open the **VS Code Command Palette** again and run **Docker: Build Image** to build the image you just created.



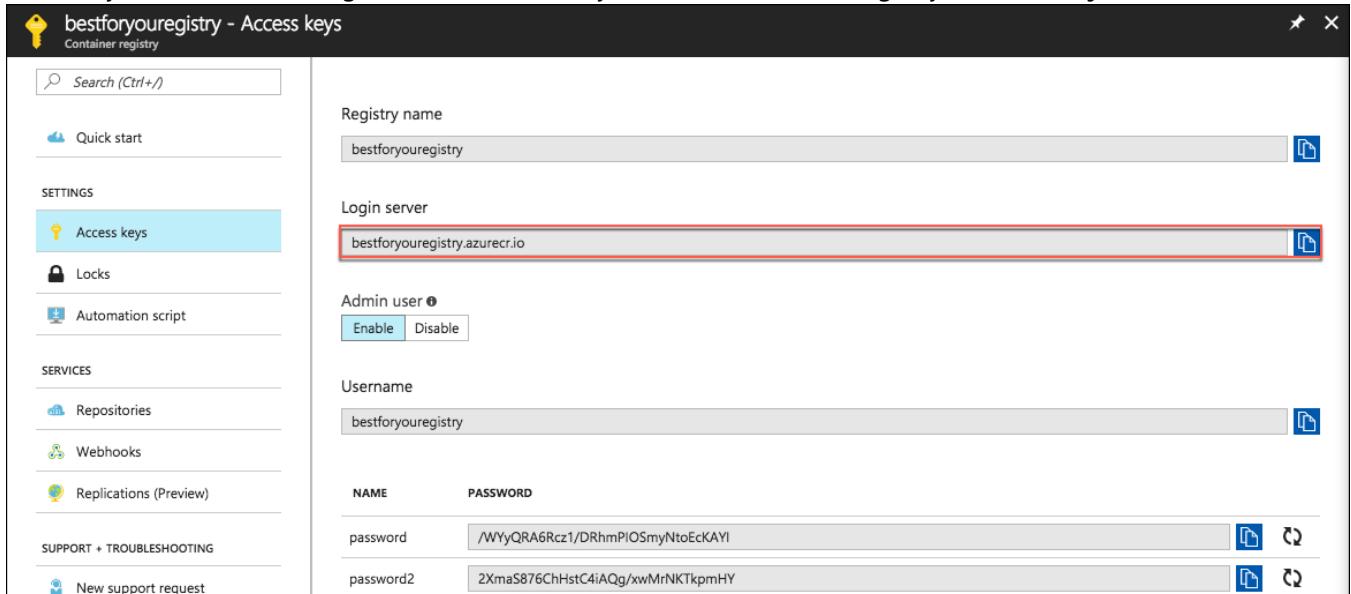
9. In the **Choose Dockerfile to build** box, select **Dockerfile** (the Dockerfile that was just created).



10. In the next box, you will provide a **registry**, **image name**, and **tag**, using the following format. This format will allow the image to be pushed to your container registry.

[registry]/[image name]:[tag]

11. For this, you will need the Login server value from your Azure Container registry's **Access keys** blade.



NAME	PASSWORD
password	/WYyQRA6Rcz1/DRhmPIOSmyNtoEckAYI
password2	2XmaS876ChHstC4iAQg/xwMrNKTkpmHY

12. Entry the following into the **Tag image as...** box, where [Login server] is the Login server value from Azure:

[Login server]/best-for-you-organics:latest

13. For example:

```
bestforyouregistry.azurecr.io/best-for-you-organics:latest
```

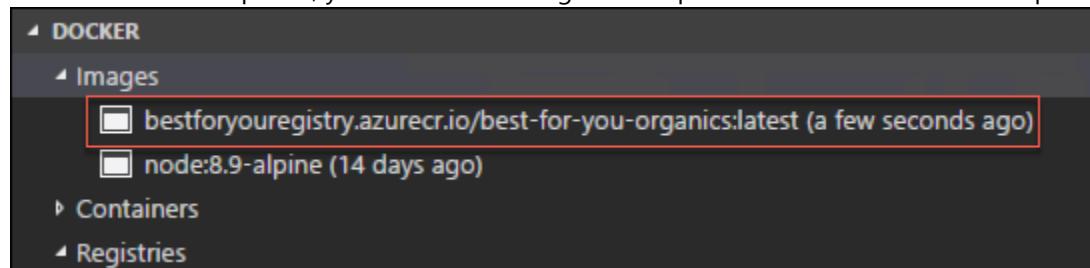
14. Enter your value, and press **Enter**, which will trigger the build of the image.

```
bestforyouregistry.azurecr.io/best-for-you-organicslatest  
Tag image as... (Press 'Enter' to confirm or 'Escape' to cancel)
```

15. In the terminal window, you will see the following docker build commands execute:

```
Successfully built be0c884360aa  
Successfully tagged bestforyouregistrykb.azurecr.io/best-for-you-organics:latest  
demouser@LabVM:~/mcw-oss-paas-devops$
```

16. Once the build completes, you will see the image show up in the **DOCKER** extension explorer, under **Images**.

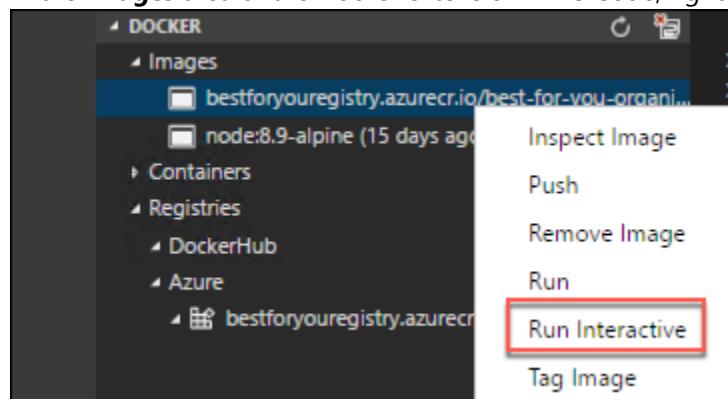


17. You can also use the docker `images` command in the Integrated terminal to list the images.

Task 4: Run the containerized App

In this task, you will run the app from the container you built in the previous task.

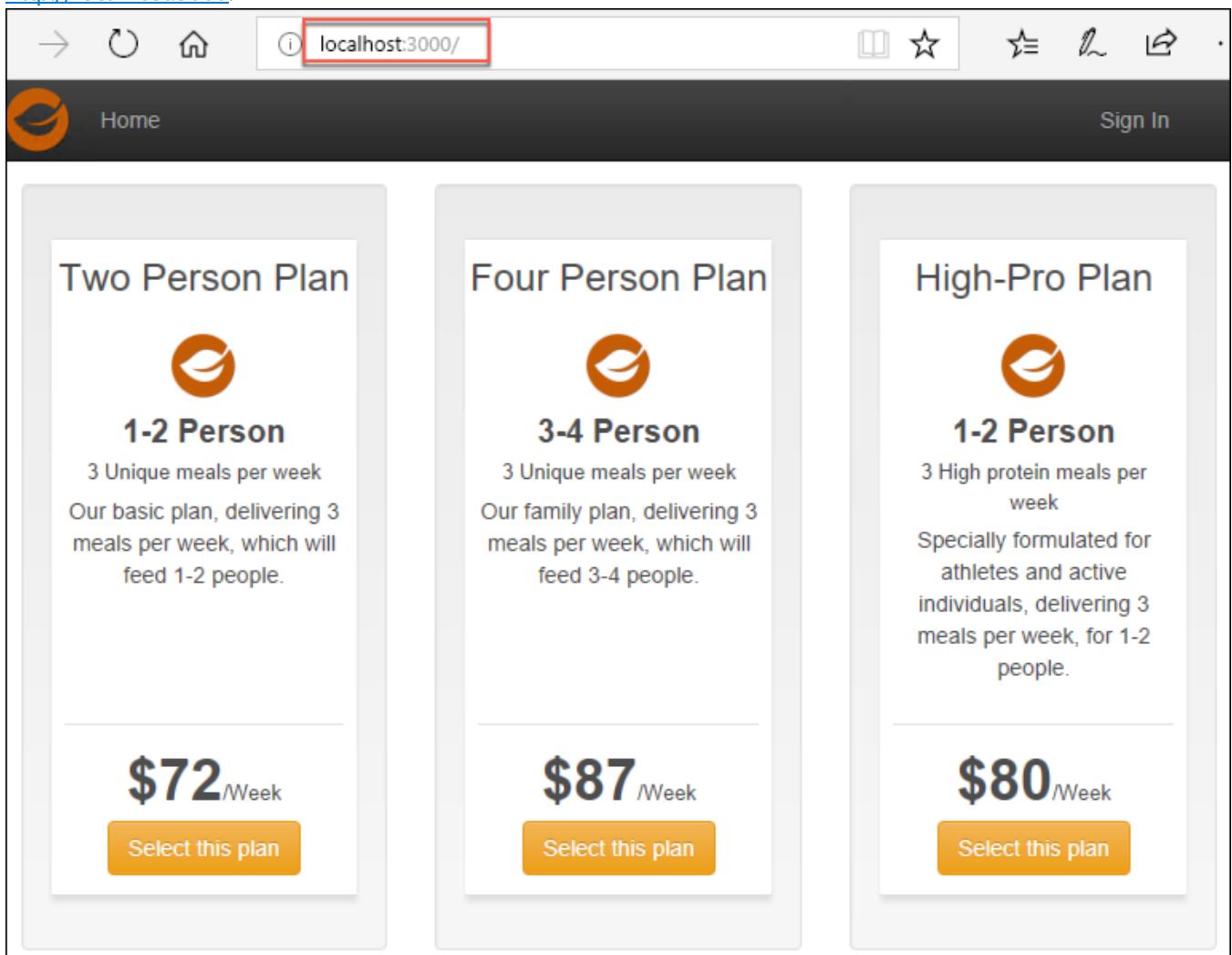
1. In the **Images** area of the Docker extension in VS Code, right-click your image, and select **Run Interactive**.



2. Notice in the Interactive terminal that a docker run command is issued. Using the VS Code Docker extension, you can issue some docker commands, without needing to work from the command line.

```
demouser@LabVM MINGW64 ~/best-for-you-organics (master)  
$ docker run --rm -it -p 3000:3000 bestforyouregistry.azurecr.io/best-for-you-organics:latest  
  
> best-for-you-organics@0.1.0 start /usr/src/app  
> node ./bin/www  
  
connection successful
```

- Verify the web app and container are functioning by opening a browser window and navigating to <http://localhost:3000>.



- In the Integrated terminal of VS Code, for the interactive session, press **CTRL+C** to stop the container.

Task 5: Push image to Azure Container Registry

In this task, you are going to push the image to your Azure Container Registry.

- In the Azure portal, navigate to the hands-on-lab-SUFFIX resource group, and select the **bestforyouregistrySUFFIX** Container registry from the list of resources.

	NAME ↑↓	TYPE ↑↓	LOCATION ↑↓
<input type="checkbox"/>	bestforyouorders	Storage account	West US
<input type="checkbox"/>	bestforyouorders	App Service	West US
<input type="checkbox"/>	bestforyouregistry	Container registry	West US

2. On the **bestforyouregistrySUFFIX** blade, select **Access keys** under settings in the left-hand menu, and leave this page up as you will be referencing the **Login server**, **Username**, and **password** in the next task.

The screenshot shows the 'Access keys' blade for a container registry named 'bestforyouregistry'. The 'SETTINGS' sidebar has 'Access keys' selected. The main area shows the 'Login server' as 'bestforyouregistry.azurecr.io', 'Username' as 'bestforyouregistry', and two password fields: 'password' containing '/WYyQRA6Rcz1/DRhmPIOSmyNtoEcKAYI' and 'password2' containing '2XmaS876ChHstC4iAQg/xwMrNKTkpmHY'. The 'password' and 'password2' fields are both highlighted with a red border.

3. Return to the Integrated terminal window in VS Code and enter the following command to log in to your Azure Container registry, replacing the bracketed values with those from the container registry access keys page.

```
docker login [Login Server] -u [Username]
```

4. For example:

```
docker login bestforyouregistry.azurecr.io -u bestforyouregistry
```

5. Enter the password when prompted to complete the login process.

```
demouser@LabVM MINGW64 ~/best-for-you-organics (master)
$ docker login bestforyouregistry.azurecr.io -u bestforyouregistry
Password:
Login Succeeded
```

6. Once you're successfully logged in, find your image in the Docker extension section of the VS Code, right-click your image, and select **Push**.

The screenshot shows the VS Code Docker extension interface. The 'Images' section lists an image named 'bestforyouregistry.azurecr.io/best-for-you-organics'. A context menu is open over this image, with the 'Push' option highlighted and surrounded by a red box. Other options in the menu include 'Remove Image', 'Run', 'Run Interactive', and 'Tag Image'.

7. Note the “docker push” command issued in the terminal window.

```
demouser@LabVM MINGW64 ~/best-for-you-organics (master)
$ docker push bestforyouregistry.azurecr.io/best-for-you-organics:latest
The push refers to repository [bestforyouregistry.azurecr.io/best-for-you-organics]
3f1e7c3b498f: Pushed
db893b47741e: Pushed
1edec2fe8db7: Pushed
10d5bf15bc20: Pushed
f3d6daebf694: Pushed
ef6ea5eda60b: Pushed
9dfa40a0da3b: Pushed
latest: digest: sha256:3e13d26ffff2bb7ad84e0c8a3adeb2e847dec0966e63a52d4f9f8c4f8af7bde28 size: 1788
```

8. To verify the push, return to the **bestforyouregistry** blade in the Azure portal, and select **Repositories** under **SERVICES** on the left-hand side, and note the **best-for-you-organics** repository.

The screenshot shows the Azure portal interface for the 'bestforyouregistry' blade. On the left, there's a sidebar with several options: 'Locks', 'Automation script', and 'SERVICES'. Under 'SERVICES', the 'Repositories' option is selected and highlighted with a red box. The main content area is titled 'bestforyouregistry - Repositories' and contains a 'Container registry' header. It features a search bar ('Search (Ctrl+/)') and a 'Refresh' button. Below these are sections for 'REPOSITORIES' and 'REGISTRY'. The 'best-for-you-organics' repository is listed in the 'REPOSITORIES' section, also highlighted with a red box.

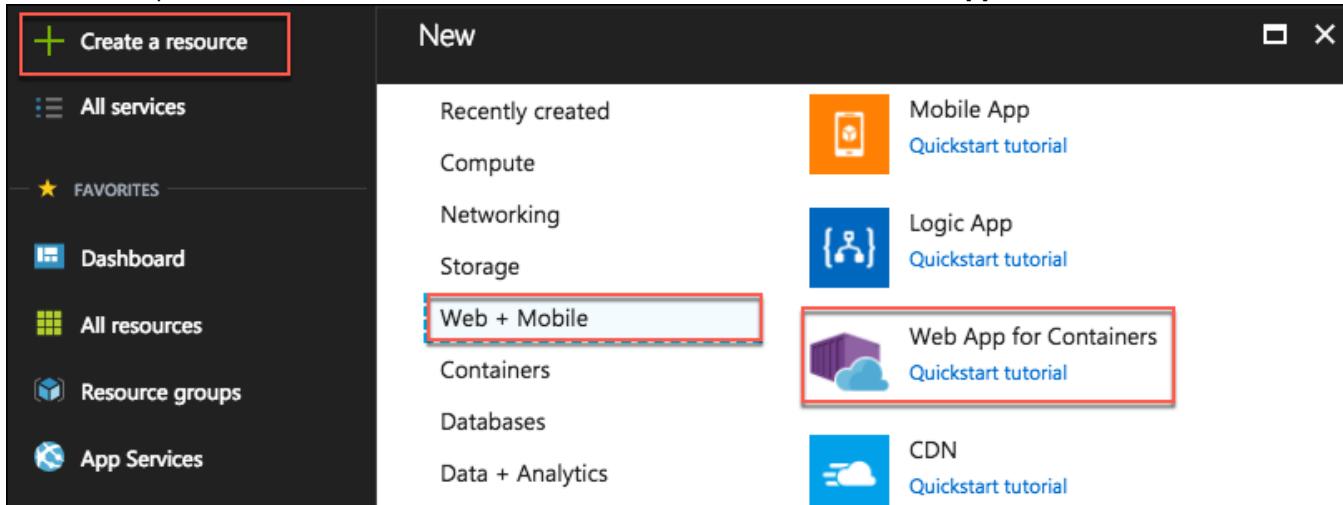
Exercise 4: Set up Web App for Containers

Duration: 10 minutes

In this exercise, you will deploy the containerized app to a Web App for Containers instance from the image stored in your Azure Container Registry.

Task 1: Provision Web App for Containers

1. In the Azure portal, select **+Create a resource**, **Web + Mobile**, and select **Web App for Containers**.



2. On the **Create** blade, enter the following:
 - a. **App name:** Enter "best-for-you-app-SUFFIX" (the name must be unique).
 - b. **Subscription:** Select the subscription you are using for this lab.
 - c. **Resource group:** Select **Use existing** and choose the hands-on-lab-SUFFIX resource group.
 - d. **App Service plan/Location:** Accept the default assigned value, which will create a new App Service plan.
 - e. Select **Configure container**, and enter the following:
 - i. **Image source:** Select **Azure Container Registry**.
 - ii. **Registry:** Select **bestforyouregistrySUFFIX**.
 - iii. **Image:** Select **best-for-you-organics**.
 - iv. **Tag:** Select **latest**.
 - v. **Startup File:** Leave blank.
 - vi. Select **OK**.

f. Select **Create**.

The screenshot shows two overlapping windows in the Azure portal. On the left, the 'Web App for Containers' creation window is visible, with the 'Create' tab selected. It contains fields for 'App name' (best-for-you-app), 'Subscription' (Soliance MVP MSDN), 'Resource Group' (hands-on-labs), 'App Service plan/Location' (ServicePlan61aaafa1-ad26(West US)), and 'Configure container'. At the bottom are 'Create' and 'Automation options' buttons. On the right, the 'Docker Container' configuration window is partially visible, showing fields for 'Image source' (Azure Container Registry selected), 'Registry' (bestforyouregistry), 'Image' (best-for-you-organics), 'Tag' (latest), and 'Startup File'. A large 'OK' button is at the bottom.

Task 2: Navigate to the deployed app

In this task, you will navigate to the deployed app, and log in to verify it is functioning correctly.

- When you receive the notification that the Web App for Containers deployment has completed, navigate to the Web App by selecting the **notifications icon**, and selecting **Go to resource**.

The screenshot shows the 'Notifications' blade in the Azure portal. It displays a single notification: 'Deployment succeeded' at 10:52 PM, stating that a deployment to resource group 'hands-on-labs' was successful. Below the message are two buttons: 'Go to resource' (which is highlighted with a red box) and 'Pin to dashboard'.

- On the **Overview** blade of **App Service**, select the URL for the App Service.

The screenshot shows the 'best-for-you-app' App Service overview blade. It includes a search bar, navigation links for 'Overview' (which is highlighted with a blue box) and 'Activity log', and various management actions like 'Browse', 'Stop', 'Restart', 'Delete', 'Get publish profile', and 'Reset publish profile'. On the right, the 'URL' field is highlighted with a red box, showing the deployed app's endpoint: <https://best-for-you-app.azurewebsites.net>. Below the URL, it shows the 'App Service plan/pricing tier' as 'ServicePlan61aaafa1-ad26 (Standard: 1 Small)'.

3. A new browser window or tab will open, and you should see the *mcw-oss-paas-devops* application's home page displayed.
4. Sign in to the site with the following credentials to verify everything is working as expected.
 - a. demouser@bfyo.com
 - b. Password.1!!

The screenshot shows a web application interface with a dark header bar. On the left is a circular logo with a stylized orange leaf or swirl design. To its right is the word "Home". Further to the right, it says "Welcome Demo" and has a "Logout" button. Below the header are three large, light-gray rectangular boxes, each representing a different meal plan:

- Two Person Plan**: Features a small orange icon of a leaf inside a circle. Below it, the text "1-2 Person" is bolded. It says "3 Unique meals per week" and "Our basic plan, delivering 3 meals per week, which will feed 1-2 people."
- Four Person Plan**: Features a small orange icon of a leaf inside a circle. Below it, the text "3-4 Person" is bolded. It says "3 Unique meals per week" and "Our family plan, delivering 3 meals per week, which will feed 3-4 people."
- High-Pro Plan**: Features a small orange icon of a leaf inside a circle. Below it, the text "1-2 Person" is bolded. It says "3 High protein meals per week" and "Specially formulated for athletes and active individuals, delivering 3 meals per week, for 1-2 people."

Exercise 5: Configure CI/CD pipeline

Duration: 60 minutes

In this exercise, you are going to add continuous delivery (CD) to a Jenkins continuous integration (CI) using Release Management in VSTS.

Task 1: Prepare GitHub account for service integrations

In this task, you will be adding a Jenkins service integration into your GitHub account. This integration will enable a Jenkins continuous integration build job to be triggered when code is checked in to your GitHub repository.

1. On your Lab VM, navigate to your Jenkins VM in the [Azure portal](#).
 - a. Select **Resource groups** from the left-hand menu, then enter "Jenkins" into the search box, and select your **Jenkins-SUFFIX** resource group from the list.

NAME
jenkins

- b. On the Jenkins-SUFFIX Resource group blade, select your **Jenkins** virtual machine.

NAME	TYPE	LOCATION
Jenkins	Virtual machine	West US
Jenkins-disk0	Disk	West US
Jenkins-nic	Network interface	West US

2. On the **Overview** blade of your Jenkins virtual machine, locate the **DNS name**, and copy the value.

The screenshot shows the 'Overview' blade for a Jenkins VM. It displays various configuration details:

- Resource group: jenkins
- Status: Running
- Location: West US
- Subscription: (change)
- Computer name: Jenkins
- Operating system: Linux
- Size: Standard DS2 v2 (2 vcpus, 7 GB memory)
- Public IP address: 13.88.30.34
- Virtual network/subnet: JenkinsVNET/JenkinsSubnet
- DNS name: jenkins-kb.westus.cloudapp.azure.com

3. Return to your forked **mcw-oss-paas-devops** application page in your GitHub account, select **Settings**, then select **Integrations & services** from the left-hand menu.

The screenshot shows the GitHub Settings page. The left sidebar has the following options:

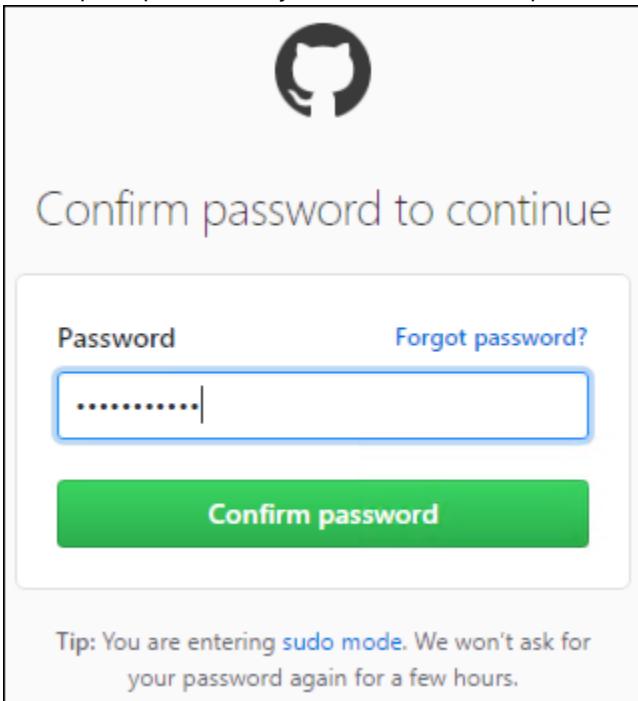
- Code
- Issues 0
- Pull requests 0
- Projects 0
- Wiki
- Insights
- Settings** (highlighted with a red box)

The main area is titled "Installed GitHub Apps" and contains sections for "Services" and "Webhooks". The "Integrations & services" link in the sidebar is also highlighted with a red box.

4. Select **Add service**, enter "Jenkins" into the search filter, and select **Jenkins (GitHub plugin)**.

The screenshot shows the GitHub Services page. The "Add service" button is highlighted with a red box. A search bar contains the text "jenkins (git)". Below it, the "Available Services" dropdown shows "jenkins (github)" and "Jenkins (GitHub plugin)", with the latter also highlighted with a red box.

5. When prompted, enter your GitHub account password to continue.



6. In the Jenkins hook URL text box, enter "http://[YOUR_JENKINS_URL]/github-webhook/" replacing [YOUR_JENKINS_URL] with the Jenkins DNS name you copied from the Azure portal, and select **Add service**.

A screenshot of the Azure portal's "Services / Add Jenkins (GitHub plugin)" configuration dialog. The title bar says "Services / Add Jenkins (GitHub plugin)".

Jenkins is a popular continuous integration server.
Using the Jenkins GitHub Plugin you can automatically trigger build jobs when pushes are made to GitHub.

Install Notes

- "Jenkins Hook Url" is the URL of your Jenkins server's webhook endpoint. For example: `http://ci.jenkins-ci.org/github-webhook/`.

For more information see <https://wiki.jenkins-ci.org/display/JENKINS/GitHub+plugin>.

Jenkins hook url

`http://jenkins-kb.westus.cloudapp.azure.com/github-webhook/`

Active
We will run this service when an event is triggered.

Add service

7. You will see the Jenkins (GitHub plugin) under the list of services.

The screenshot shows the 'Installed GitHub Apps' section of the GitHub interface. A red box highlights the entry for 'Jenkins (GitHub plugin)'. To the right of this entry are 'Edit' and 'Delete' buttons. The overall heading is 'Services' with an 'Add service' button.

8. Next, you need to grant the Jenkins user access to your GitHub repository by adding a deploy key in the GitHub settings.
9. Return to your Jenkins virtual machine page in the Azure portal, select **Connect**, and copy the SSH command.

The screenshot shows the 'Connect' dialog for a Jenkins VM. The 'Connect' button is highlighted with a red box. Below it, the text 'To connect to your Linux virtual machine using SSH, use the following command:' is followed by the SSH command 'ssh jenkinsadmin@137.117.14.92', which is also highlighted with a red box. At the bottom is an 'OK' button.

10. Open a new bash shell and paste the SSH command you copied above at the prompt. Enter "yes" if prompted about continuing to connect, and enter the jenkinsadmin password, "Password.1!!," when prompted.

```
demouser@LabVM MINGW64 ~
$ ssh jenkinsadmin@13.91.105.165
The authenticity of host '13.91.105.165 (13.91.105.165)' can't be established.
ECDSA key fingerprint is SHA256:9GG01sDiRT6uh5a07DJubpPgakg00bWz7u7km5D46/E.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '13.91.105.165' (ECDSA) to the list of known hosts.
jenkinsadmin@13.91.105.165's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.13.0-1005-azure x86_64)
```

11. At the jenkinsadmin@Jenkins prompt, enter:

```
ssh-keygen
```

12. Press **Enter** to accept the default file in which to save the key.
13. Press **Enter** to use an empty passphrase, and re-enter it to confirm. (Note: This is done only for simplicity in this hands-on lab, and is not recommended for actual environments.)

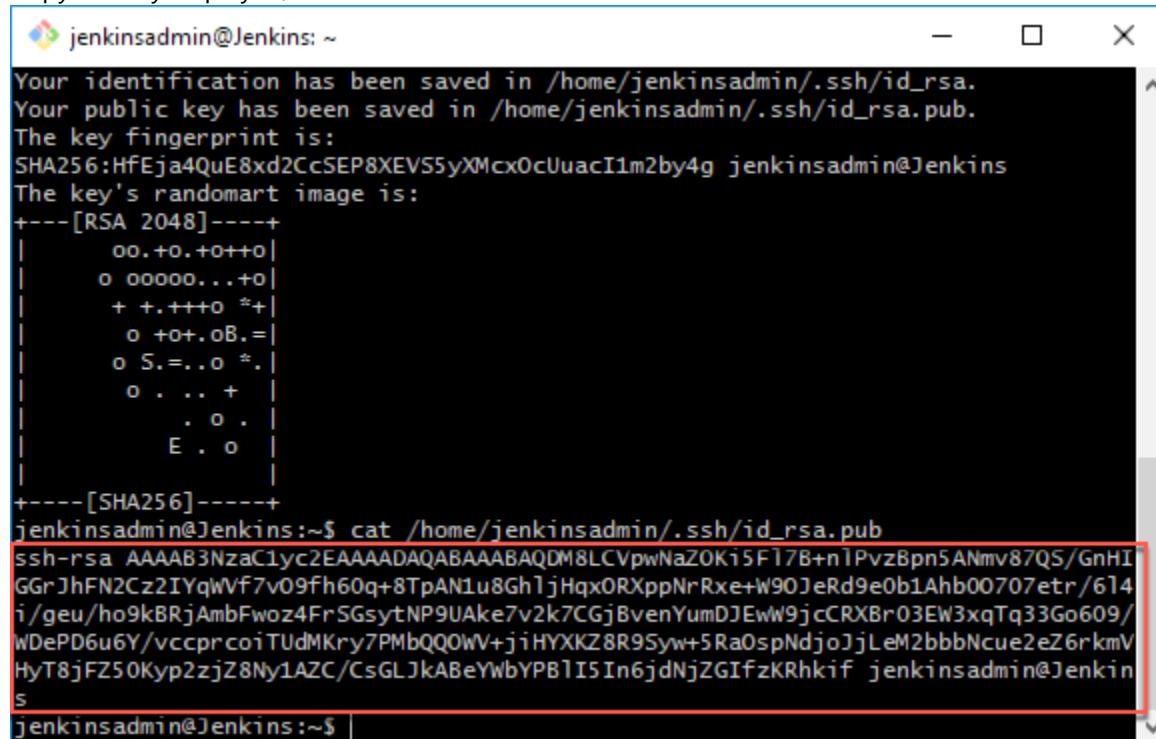
14. Copy the location into which your public key has been saved.

```
jenkinsadmin@Jenkins:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jenkinsadmin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jenkinsadmin/.ssh/id_rsa.
Your public key has been saved in /home/jenkinsadmin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:HfEja4QuE8xd2CcSEP8XEV55yXMcx0cUuacI1m2by4g jenkinsadmin@Jenkins
The key's randomart image is:
+---[RSA 2048]---
|   oo.+o.+o++o|
|   o ooooo...+o|
|   + .++o *+
|   o +o+.oB.=|
|   o S.=..o *.
|   o . . +
|   . o .
|   E . o
+---[SHA256]---+
jenkinsadmin@Jenkins:~$
```

15. Show the public key using the following command, replacing [KEY_PATH] with the location of your public key.

```
cat [KEY_PATH]
```

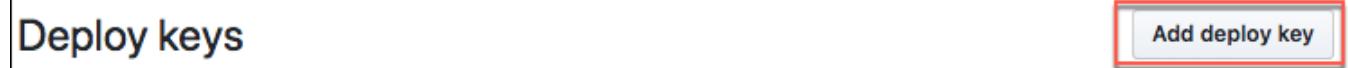
16. Copy the key displayed, so it can be added to GitHub.



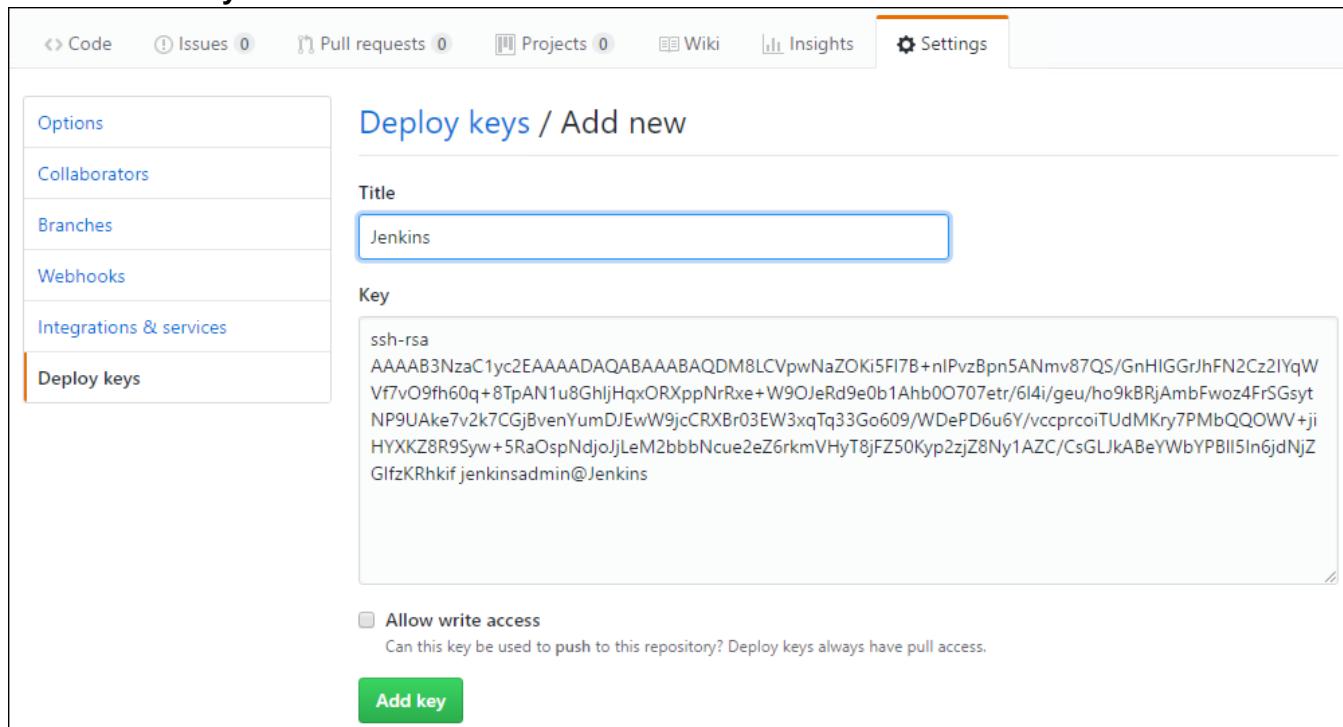
The terminal window shows the output of the ssh-keygen command. It includes the identification and public key locations, the key fingerprint, the randomart image, and the SHA256 hash. Below this, the command cat /home/jenkinsadmin/.ssh/id_rsa.pub is run, displaying the full content of the public key file. The public key itself is highlighted with a red rectangle.

```
jenkinsadmin@Jenkins: ~
Your identification has been saved in /home/jenkinsadmin/.ssh/id_rsa.
Your public key has been saved in /home/jenkinsadmin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:HfEja4QuE8xd2CcSEP8XEV55yXMcx0cUuacI1m2by4g jenkinsadmin@Jenkins
The key's randomart image is:
+---[RSA 2048]---
|   oo.+o.+o++o|
|   o ooooo...+o|
|   + .++o *+
|   o +o+.oB.=|
|   o S.=..o *.
|   o . . +
|   . o .
|   E . o
+---[SHA256]---+
jenkinsadmin@Jenkins:~$ cat /home/jenkinsadmin/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDM8LCVpwNaZOKi5F17B+n1PvzBpn5ANmv87QS/GnHI
GGrJhFN2Cz2IYqWVf7v09fh60q+8TpAN1u8Gh1jHqx0RXppNrRxe+W90JeRd9e0b1Ahb00707etr/614
i/geu/h09kBRjAmbFwoz4FrSGsytNP9UAke7v2k7CGjBvenYumDJEwW9jcCRXBr03EW3xqTq33Go609/
WDePD6u6Y/vccprcoiTudMKry7PMbQQ0WV+jiHYXKZ8R95yw+5RaOspNdjoJjLeM2bbbNcue2eZ6rkmV
HyT8jFZ50Kyp2zjZ8Ny1AZC/CsGLjkABeYWbYPBlISIn6jdNjZGIfzKRhkif jenkinsadmin@Jenkin
s
jenkinsadmin@Jenkins:~$
```

17. Return to your GitHub account in the browser, select the **Deploy keys** option from the left-hand menu, and select **Add deploy key**.



18. Enter "Jenkins" for the title, paste the SSH key you copied above into the Key field, removing any trailing spaces, and select **Add key**.



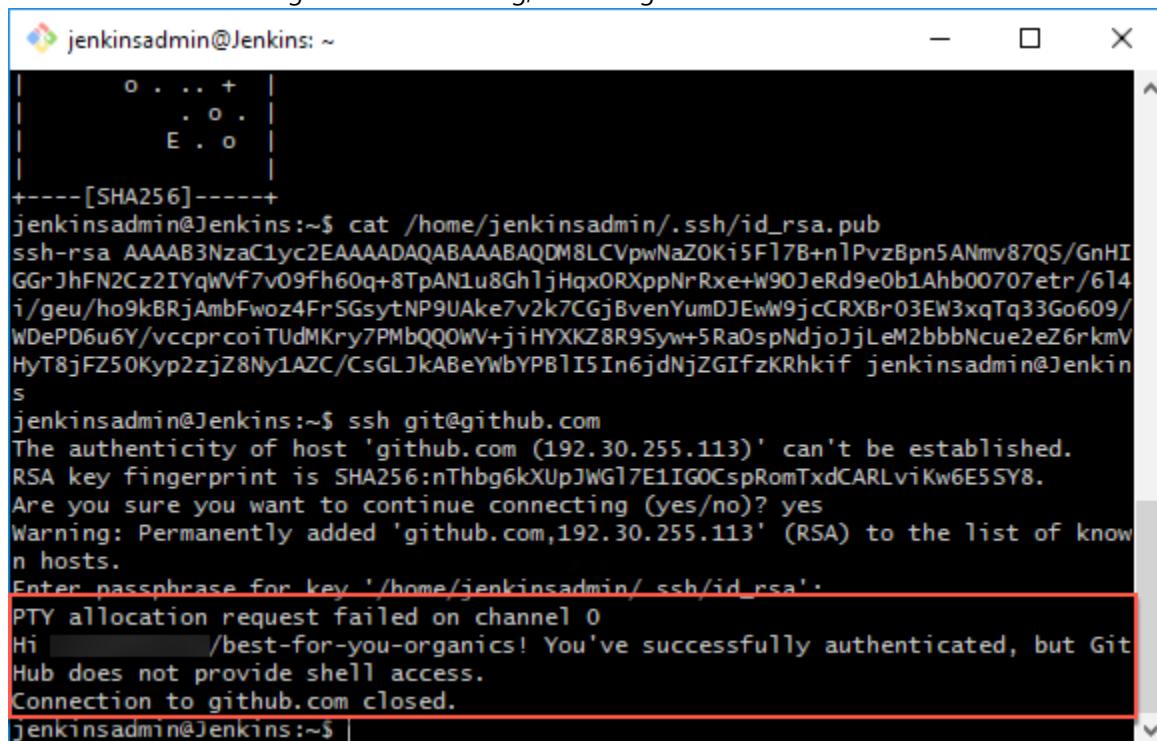
The screenshot shows the GitHub 'Deploy keys / Add new' page. On the left, there's a sidebar with options like 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. The 'Settings' tab is active. Below it, a vertical list includes 'Options', 'Collaborators', 'Branches', 'Webhooks', 'Integrations & services', and 'Deploy keys', with 'Deploy keys' being the current selection. The main area has a title 'Deploy keys / Add new' and a 'Title' input field containing 'Jenkins'. A large text area labeled 'Key' contains a long SSH RSA key. Below the key area is a checkbox for 'Allow write access' with a note: 'Can this key be used to push to this repository? Deploy keys always have pull access.' At the bottom is a green 'Add key' button.

19. To ensure that everything is working, return to the Jenkins' bash shell, and enter the below command which will check the connection to GitHub.

```
ssh git@github.com
```

20. Enter "yes" when prompted about continuing.

21. You should see a message like the following, indicating a successful connection.



The screenshot shows a terminal window titled 'jenkinsadmin@Jenkins: ~'. The terminal output shows the user running 'ssh git@github.com'. The system prompts for a passphrase, which is redacted. It then displays a message from GitHub stating: 'PTY allocation request failed on channel 0', 'Hi [REDACTED] /best-for-you-organics!', 'You've successfully authenticated, but GitHub does not provide shell access.', and 'Connection to github.com closed.' The entire terminal session is framed by a red border.

22. The GitHub side of the integration with Jenkins is complete. Next, you will configure Jenkins as part of your CI/CD pipeline.

Task 2: Configure Continuous Integration (CI) with Jenkins

In this task, you will set up a simple Jenkins CI pipeline, which will build the **mcw-oss-paas-devops** application with every code commit into GitHub.

1. Return to your **Jenkins** VM blade in the Azure portal.
2. On the **Overview** blade of your Jenkins VM, locate the **DNS name**, and copy the value.

Resource group (change) jenkins	Computer name Jenkins
Status Running	Operating system Linux
Location West US	Size Standard DS2 v2 (2 vcpus, 7 GB memory)
Subscription (change)	Public IP address 13.88.30.34
Subscription ID [REDACTED]	Virtual network/subnet JenkinsVNET/JenkinsSubnet
	DNS name jenkins-kb.westus.cloudapp.azure.com

3. Open a new browser window or tab and paste the copied DNS name into the browser's address bar to navigate to your Jenkins server.
4. On the Jenkins on Azure screen, you will see a message that this Jenkins instance does not support https, so logging in through a public IP address has been disabled. You will need to create an SSH tunnel to securely connect to the Jenkins instance.
5. To set up an SSH tunnel to Jenkins, copy the ssh command provided in the Jenkins on Azure window, as highlighted in the screen shot below.

This Jenkins instance does not support https, so logging in through a public IP address has been disabled (it would expose your password and other information to eavesdropping). To securely login, you need to connect to the Jenkins instance using SSH port forwarding.

```
ssh -L 127.0.0.1:8080:localhost:8080 username@jenkins-kb.westus.cloudapp.azure.com
```

If you don't want to publicly expose this Jenkins instance, you need to remove the nginx reverse-proxy.

```
sudo apt-get purge nginx nginx-common
```

6. Open a new bash shell, and at the command prompt paste the copied ssh command, replacing **username** with **jenkinsadmin**. The command will resemble the following:

```
ssh -L 127.0.0.1:8080:localhost:8080 jenkinsadmin@jenkins-kb.westus.cloudapp.azure.com
```

7. If prompted that authenticity of the Jenkins host cannot be established, enter "yes" to continue.

8. Enter the **jenkinsadmin** password, "Password.1!!".

```
Kyle@VS2017 MINGW64 ~
$ ssh -L 127.0.0.1:8080:localhost:8080 jenkinsadmin@jenkins-kb.westus.cloudapp.azure.com
The authenticity of host 'jenkins-kb.westus.cloudapp.azure.com (13.88.30.34)' can't be established.
ECDSA key fingerprint is SHA256:J1FsBXBnvpndoq3kdHnF6B0MCXGxP+XQ41j0yujfWhI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'jenkins-kb.westus.cloudapp.azure.com,13.88.30.34' (EDSA) to the list of known hosts.
jenkinsadmin@jenkins-kb.westus.cloudapp.azure.com's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.11.0-1016-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 Get cloud support with Ubuntu Advantage Cloud Guest:
  http://www.ubuntu.com/business/services/cloud

12 packages can be updated.
5 updates are security updates.
```

9. After you have started the SSH tunnel, open a new browser tab or window, and navigate to <http://localhost:8080/>.

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

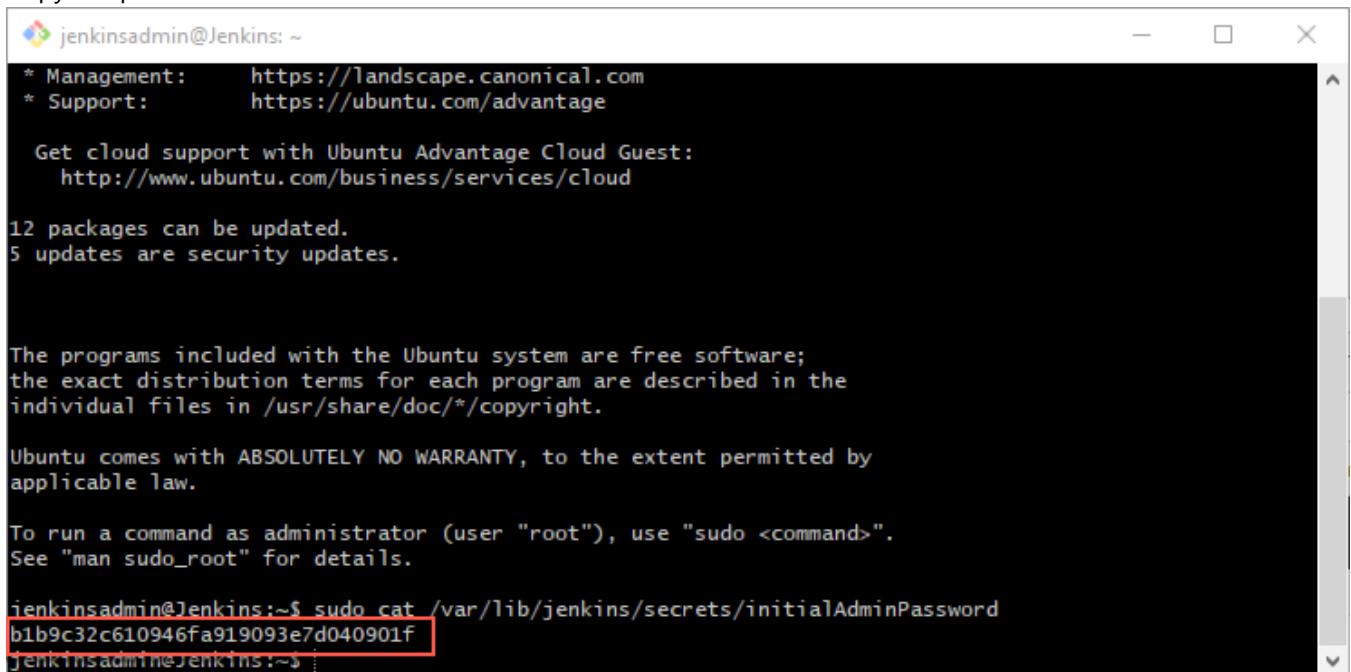
Administrator password

Continue

10. To get the initial password, copy the path provided, return to the SSH tunnel bash window, and run the following command:

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

11. Copy the password returned.



```
jenkinsadmin@Jenkins: ~
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

12 packages can be updated.
5 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

jenkinsadmin@Jenkins:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
b1b9c32c610946fa919093e7d040901f
jenkinsadmin@Jenkins:~$
```

12. Return to the Getting Started screen in your browser, paste the password into the **Administrator password** box, and select **Continue**.

Please copy the password from either location and paste it below.

Administrator password

.....

Continue

13. On the Customize Jenkins screen, select **Install suggested plugins**.

Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.107.1

14. On the Create First Admin User screen, enter the following:

- Username:** Enter a username, such as your first name
- Password:** Password.1!!
- Confirm Password:** Password.1!!
- Full name:** Enter your first name
- E-mail address:** Enter your email address
- Select **Save and Finish**

Getting Started

Create First Admin User

Username:	kyle	
Password:	
Confirm password:	
Full name:	Kyle	
E-mail address:	@gmail.com	

Jenkins 2.107.1

[Continue as admin](#) [Save and Finish](#)

15. Select **Start using Jenkins** on the Jenkins is ready screen.

The screenshot shows a 'Getting Started' section with a large heading 'Jenkins is ready!' and the subtext 'Your Jenkins setup is complete.' Below this is a blue button labeled 'Start using Jenkins'. At the bottom of the page, it says 'Jenkins 2.107.1'.

16. You will be redirected to the Jenkins dashboard.

The dashboard features a navigation sidebar on the left with links: 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', 'Credentials', and 'New View'. The main area has a large 'Welcome to Jenkins!' message and a call-to-action 'Please create new jobs to get started.'

17. Select **Manage Jenkins** from the left-hand menu and select **Manage Plugins**.

The screenshot shows the Jenkins Manage Jenkins interface. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is highlighted with a red box), 'My Views', 'Credentials', and 'New View'. Below the sidebar are three collapsed sections: 'Cloud Statistics', 'Build Queue' (showing 'No builds in the queue.'), and 'Build Executor Status'. The main area is titled 'Manage Jenkins' and contains several configuration links: 'Configure System', 'Configure Global Security', 'Configure Credentials', 'Global Tool Configuration', 'Reload Configuration from Disk', 'Manage Plugins' (which is also highlighted with a red box), and 'System Information'. The 'Manage Plugins' link has a tooltip: 'Add, remove, disable or enable plugins that can extend the functionality of Jenkins.'

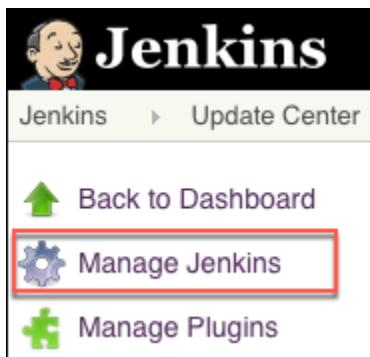
18. To install the VS Team Services Continuous Deployment plug-in, enter **vs team services continuous deployment** into the Filter box, select the **VS Team Services Continuous Deployment** plug-in, and select **Install without restart**.

This screenshot shows the Jenkins Manage Plugins page with a filter set to 'vs team'. A specific plugin, 'VS Team Services Continuous Deployment', is selected and highlighted with a red box. Below the list, there are three buttons: 'Install without restart' (which is highlighted with a red box), 'Download now and install after restart', and 'Check now'. A status message at the bottom right says 'Update information obtained: 22 hr ago'.

19. Now, change the Filter text to **nodejs**, select the **NodeJS** plug-in, and select **Install without restart**.

This screenshot shows the Jenkins Manage Plugins page with a filter set to 'nodejs'. A specific plugin, 'NodeJS', is selected and highlighted with a red box. Below the list, there are three buttons: 'Install without restart' (which is highlighted with a red box), 'Download now and install after restart', and 'Check now'. A status message at the bottom right says 'Update information obtained: 22 hr ago'.

20. Scroll up to the top of the screen, select **Manage Jenkins** from the left-hand menu.



21. Select **Global Tool Configuration**.

A screenshot of the 'Manage Jenkins' page. It contains several configuration sections: 'Configure System' (gear icon), 'Configure Global Security' (padlock icon), 'Configure Credentials' (key icon), and 'Global Tool Configuration' (wrench and screwdriver icon). The 'Global Tool Configuration' section is highlighted with a red box.

22. Find **NodeJS** and select **Add NodeJS** next to NodeJS installations.

23. Enter a name, ensure **Install automatically** is checked, and accept the default (latest) version of nodejs.

A screenshot of the 'Add NodeJS' configuration page. It shows a 'Name' field containing 'bestforyounode' (with a red 'Required' indicator), an 'Install automatically' checkbox checked, and an 'Install from nodejs.org' section. The 'Version' dropdown is set to 'NodeJS 9.4.0'. There is also a note about specifying global npm packages to install and a setting for global npm packages refresh hours (set to 72).

24. Select **Save**.



25. Return to the **Jenkins** dashboard, and select **New Item** from the left-hand menu.
26. Enter "best-for-you-build" as the name, select **Freestyle project**, and select **OK**.

Enter an item name

best-for-you-build
» Required field

Freestyle project
 This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
 Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
 Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
 Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline
 Creates a set of Pipeline projects according to detected branches in one SCM repository.

OK

27. On the **General** tab of the project page, select **GitHub project**, and enter the URL for your forked copy of the best-for-you-organics project page in your GitHub account.

General Source Code Management Build Triggers Build Environment Build Post-build Actions

>>

Project name: best-for-you-build

Description:

[Plain text] [Preview](#)

Enable project-based security

Discard old builds

GitHub project

Project url: <https://github.com/mcw-oss-paas-devops/>

[Advanced...](#)

28. Next, scroll down to the **Source Code Management** section, select **Git**, and enter the URL to your project, including the ".git" extension.

Source Code Management

None Git

Repositories

Repository URL: https://github.com/mcw-oss-paas-devops.git

Credentials: - none -

Advanced...

Branches to build

Branch Specifier (blank for 'any'): */master

Add Branch

Repository browser: (Auto)

Additional Behaviours

29. Scroll down to the **Build Triggers** section and select **GitHub hook trigger for GITScm polling**.

Build Triggers

Trigger builds remotely (e.g., from scripts)

Build after other projects are built

Build periodically

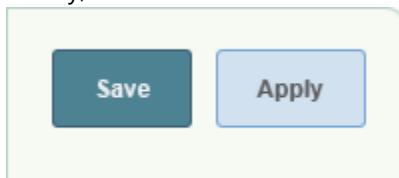
Build when a change is pushed to TFS/Team Services

Build when a change is pushed to a TFS pull request

GitHub hook trigger for GITScm polling

Poll SCM

30. Finally, select **Save**.



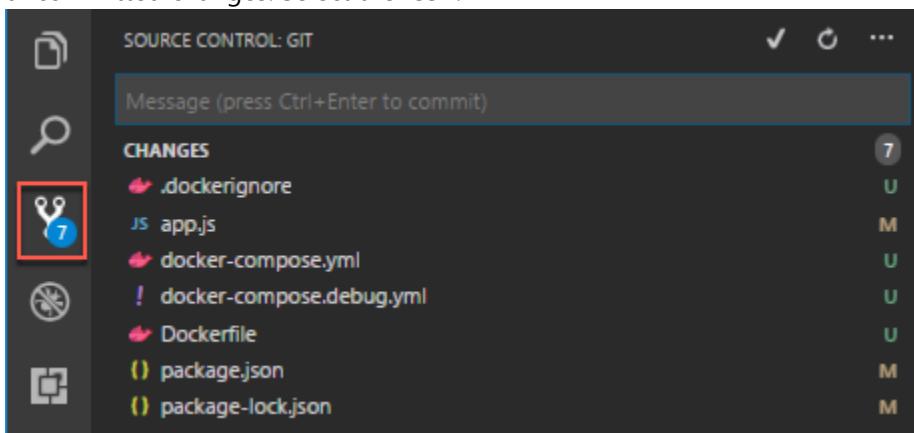
31. Your Jenkins CI build job should now be triggered whenever a push is made to your repository.

Task 3: Trigger CI build

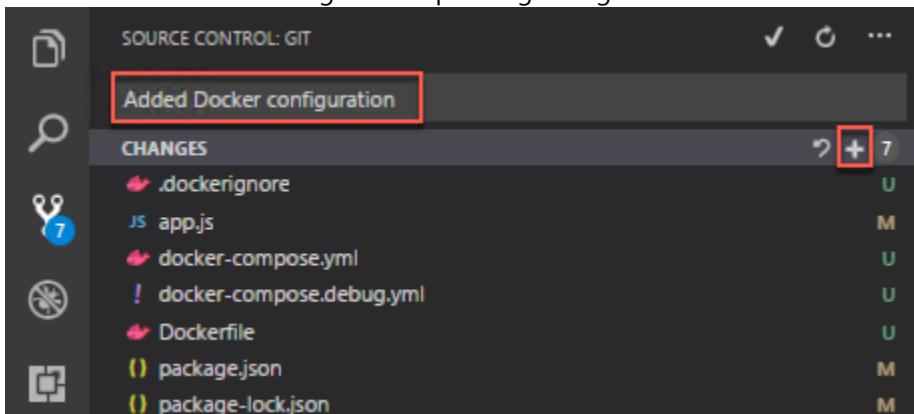
In this task you will commit your pending changes in VS Code to your GitHub repo, and trigger the Jenkins CI build job.

1. Return to VS Code on your Lab VM, and open the mcw-oss-paas-devops project.

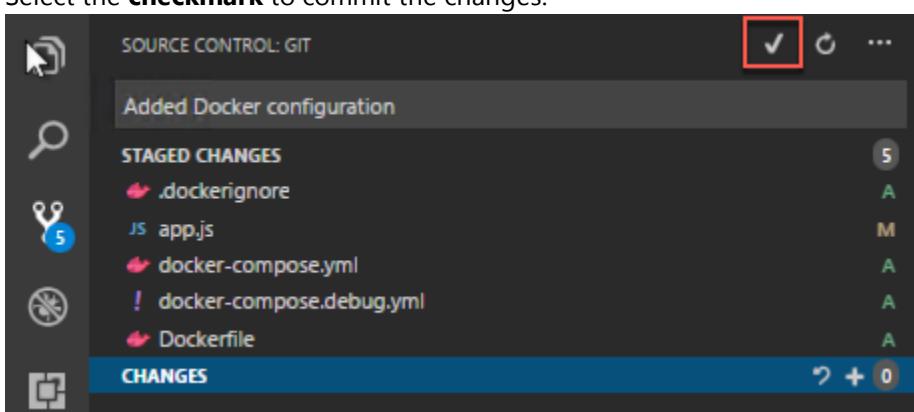
2. Observe that the **source control icon** on the left-hand navigation bar has a badge indicating you have uncommitted changes. Select the **icon**.



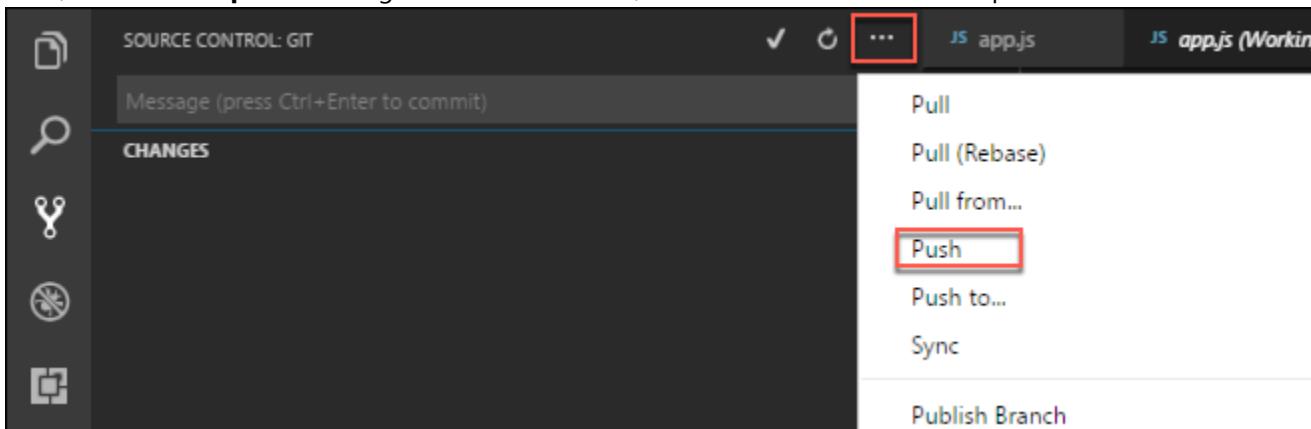
3. In the **SOURCE CONTROL: GIT** pane, enter a commit message, such as "Added Docker configuration," and select **+ next to CHANGES** to stage all the pending changes.



4. Select the **checkmark** to commit the changes.



5. Next, select the **ellipsis** to the right of the check mark, and select **Push** from the dropdown.



6. If prompted, enter your GitHub account credentials to log into your GitHub account.
7. Return to your best-for-you-build job in Jenkins, and locate the **Build History** block on the left-hand side. Select **#1** to view the details of the build job, caused by your GitHub commit.

The screenshot shows the Jenkins 'Build History' page. It displays a single build entry for '#1' which was triggered on 'Apr 11, 2018 2:59 PM'. Below the entry are two RSS feed links: 'RSS for all' and 'RSS for failures'.

8. On the build page, you can see the changes you committed.

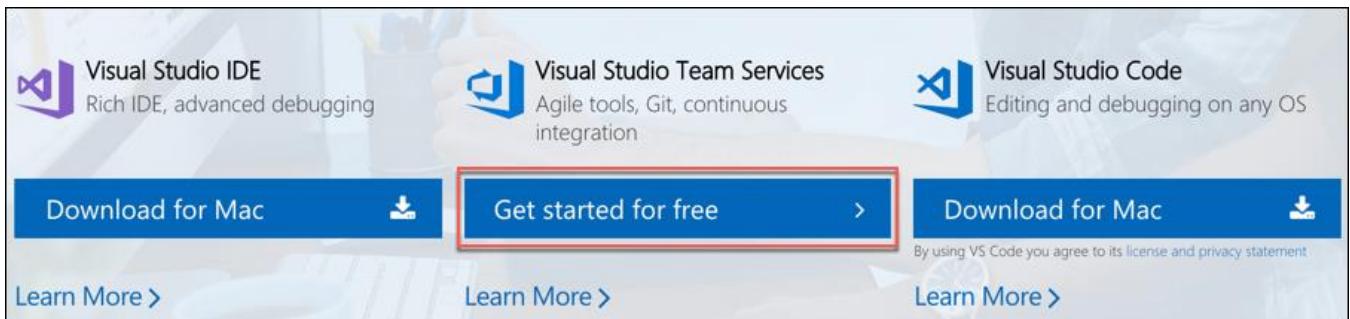
The screenshot shows the Jenkins build page for job #1. It displays the commit history under the 'Changes' section, showing three commits: 'Added Docker configuration (commit: 1a9fd22) (detail / githubweb)', 'test (commit: ca9f6bd) (detail / githubweb)', and 'test (commit: 0a61acc) (detail / githubweb)'. Below the changes, it shows the 'Started by GitHub push by [redacted]'. Under the 'git' icon, it shows the 'Revision: 8262b32626e658ef013ad9e8fcc262615a58c10f' and the 'refs/remotes/origin/master' ref.

9. You have successfully set up your CI pipeline.

Task 4: Create Free Visual Studio Team Services Account

In this task, you will create a free Visual Studio Team Services (VSTS) account. If you already have a VSTS account, you can skip this task.

1. In a browser, navigate to <https://www.visualstudio.com/>, and select **Get started for free** under **Visual Studio Team Services**.



2. Sign into your Microsoft account when prompted.
3. At the **Host my project at:** page, enter a unique name, and select **Continue** to create your VSTS account.

Host my projects at:

Pick a memorable name **.visualstudio.com**

Manage code using:

Git

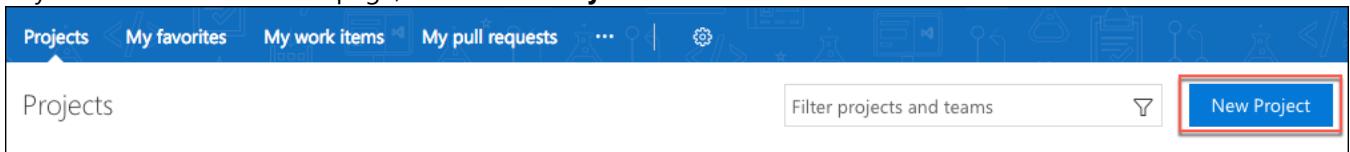
Team Foundation Version Control

We will host your projects in **Central US** region.
You can share work with other [redacted] users.

Change details

Continue

4. In your VSTS account home page, select **New Project**.



5. On the **Create new project** screen, enter “BestForYouOrganics” for the Project name, and select **Create**.

Create new project

Projects contain your source code, work items, automated builds and more.

Project name *

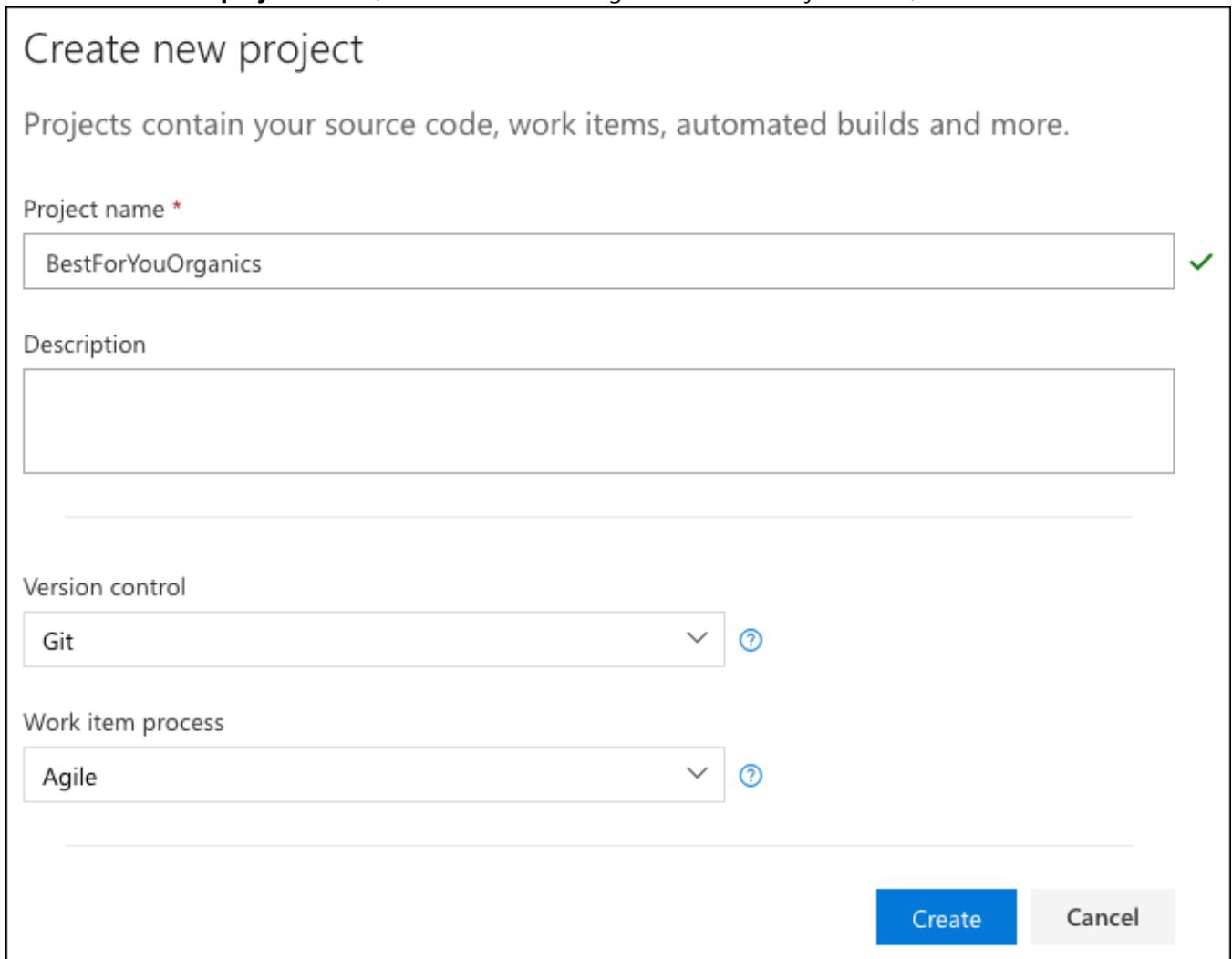
 ✓

Description

Version control

Work item process

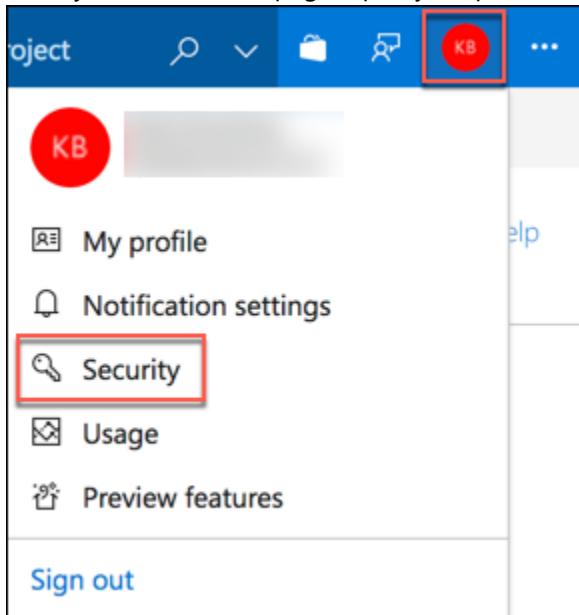
Create Cancel



Task 5: Create a VSTS personal access token

In this task, you will create a personal access token with the Release (read, write, execute, and manage) permission in VSTS, which will allow Jenkins to communicate with Team Services account.

1. From your VSTS home page, open your profile, and select **Security**.



2. On the **Security** page, select **Personal access tokens**, then select **Add**.

A screenshot of the 'Personal access tokens' page. It shows a list of token types: 'Personal access tokens' (highlighted with a red box), 'Alternate authentication credentials', 'OAuth authorizations', and 'SSH public keys'. To the right, there is a summary section and a control bar with 'Add' (highlighted with a red box), 'Revoke All', and 'Show Revoked Tokens' buttons. Below the control bar, it says 'You have not created any personal access tokens.'

3. On the **Create a personal access token screen**, enter a **description**, such as "best-for-you-organics," select an **expiration period** for the token, and select your **account**.

A screenshot of the 'Create a personal access token' dialog. It has a heading 'Create a personal access token' and a note about browser-based access. There are three input fields: 'Description' containing 'best-for-you-organics', 'Expires In' set to '90 days', and 'Accounts' which is currently empty. Each field has a dropdown arrow to its right.

4. Under **Authorized Scopes**, choose **Selected scopes**, and select **Release (read, write, execute and manage)**.

Authorized Scopes

All scopes
 Selected scopes

<input type="checkbox"/> Agent Pools (read)	<input type="checkbox"/> Agent Pools (read, manage)	<input type="checkbox"/> Build (read and execute)
<input type="checkbox"/> Build (read)	<input type="checkbox"/> Code (full)	<input type="checkbox"/> Code (read and write)
<input type="checkbox"/> Code (read)	<input type="checkbox"/> Code (read, write, and manage)	<input type="checkbox"/> Code (status)
<input type="checkbox"/> Code search (read)	<input type="checkbox"/> Connected Server	<input type="checkbox"/> Deployment group (read, manage)
<input type="checkbox"/> Entitlements (Read)	<input type="checkbox"/> Extension data (read and write)	<input type="checkbox"/> Extension data (read)
<input type="checkbox"/> Extensions (read and manage)	<input type="checkbox"/> Extensions (read)	<input type="checkbox"/> Identity (manage)
<input type="checkbox"/> Identity (read)	<input type="checkbox"/> Load test (read and write)	<input type="checkbox"/> Load test (read)
<input type="checkbox"/> Marketplace	<input type="checkbox"/> Marketplace (acquire)	<input type="checkbox"/> Marketplace (manage)
<input type="checkbox"/> Marketplace (publish)	<input type="checkbox"/> Notifications (manage)	<input type="checkbox"/> Notifications (read)
<input type="checkbox"/> Notifications (write)	<input type="checkbox"/> Packaging (read and write)	<input type="checkbox"/> Packaging (read)
<input type="checkbox"/> Packaging (read, write, and manage)	<input type="checkbox"/> Project and team (read and write)	<input type="checkbox"/> Project and team (read)
<input type="checkbox"/> Project and team (read, write and manage)	<input type="checkbox"/> Release (read)	<input type="checkbox"/> Release (read, write and execute)
<input checked="" type="checkbox"/> Release (read, write, execute and manage)	<input type="checkbox"/> Security (manage)	<input type="checkbox"/> Service Endpoints (read and query)
<input type="checkbox"/> Service Endpoints (read)	<input type="checkbox"/> Service Endpoints (read, query and manage)	<input type="checkbox"/> Symbols (read and write)
<input type="checkbox"/> Symbols (read)	<input type="checkbox"/> Symbols (read, write and manage)	<input type="checkbox"/> Task Groups (read)
<input type="checkbox"/> Task Groups (read, create and manage)	<input type="checkbox"/> Task Groups (read, create)	<input type="checkbox"/> Team dashboards (manage)
<input type="checkbox"/> Team dashboards (read)	<input type="checkbox"/> Test management (read and write)	<input type="checkbox"/> Test management (read)
<input type="checkbox"/> User profile (read)	<input type="checkbox"/> User profile (write)	<input type="checkbox"/> Wiki (read and write)
<input type="checkbox"/> Wiki (read)	<input type="checkbox"/> Work item search (read)	<input type="checkbox"/> Work items (full)
<input type="checkbox"/> Work items (read and write)	<input type="checkbox"/> Work items (read)	

Create Token **Cancel**

5. Select **Create Token**.

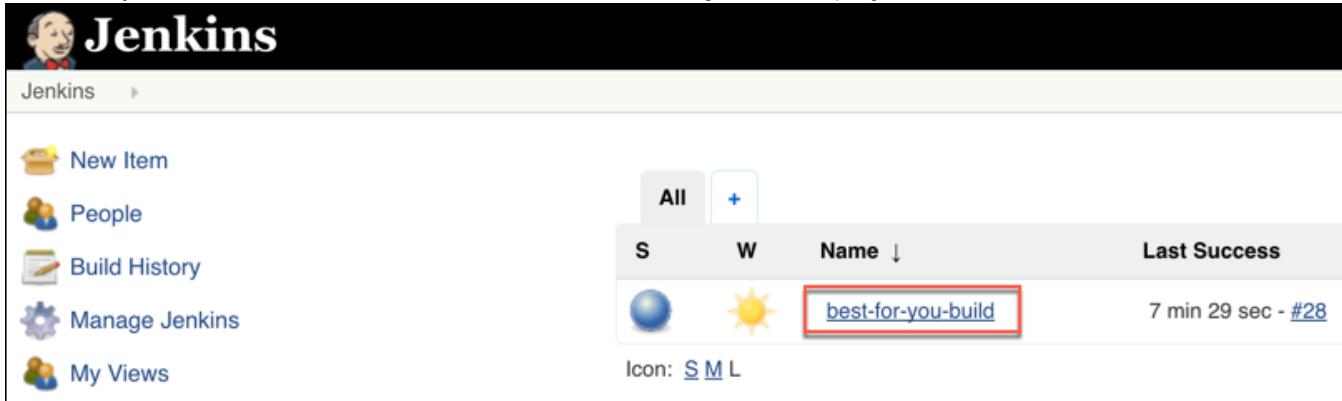
6. Copy the generated token, and paste it into a text editor, such as Notepad, for later use. Make sure you copy the token and save it in a file, as it will no longer be accessible once you navigate away from the screen.

Description	Expiration	Status ↑	Actions
best-for-you-organics	4/25/2018 2:23:12 PM	Active	 Make sure you copy the token now. We don't store it anywhere else. <code>ki75vilgfsaztlx7hbput3roqx5wvjykb23ahtif4fxqfnhscqa</code>

Task 6: Configure Jenkins for Team Services integration

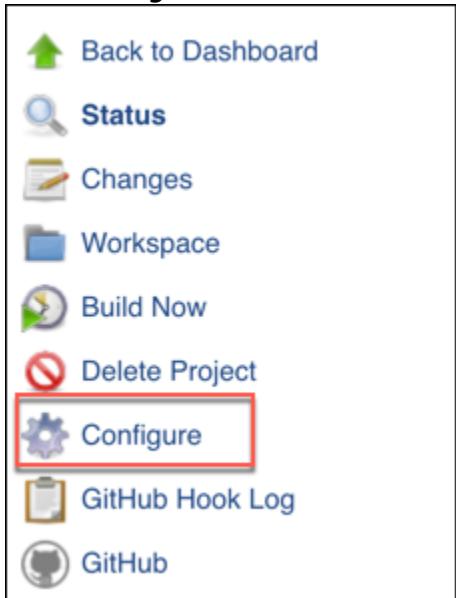
In this task, you will add the necessary configuration information to allow Jenkins to send build artifacts to a VSTS release.

1. Return to your **Jenkins** dashboard, and select the **best-for-you-build** project.



The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links: New Item, People, Build History, Manage Jenkins, and My Views. The main area displays a list of projects. The project 'best-for-you-build' is highlighted with a red box. The table columns are All, S, W, Name (sorted by Name), and Last Success. The 'best-for-you-build' row shows an icon, a sun icon, the name 'best-for-you-build' (which is also highlighted with a red box), and the status '7 min 29 sec - #28'. Below the table, it says 'Icon: S M L'.

2. Select **Configure** from the left-hand menu.



The screenshot shows the 'Configure' menu on the Jenkins dashboard. The menu items are: Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Project, Configure (which is highlighted with a red box), GitHub Hook Log, and GitHub.

3. Select the **Post-build Actions** tab, then select **Add post-build action**, and select **Archive the artifacts** from the list.

The screenshot shows the 'Post-build Actions' tab selected in the top navigation bar. A dropdown menu is open, listing several actions. The 'Archive the artifacts' option is highlighted with a blue selection bar and has a blue border around it. At the bottom of the dropdown menu is a red-bordered button labeled 'Add post-build action ▾'. Below the dropdown, there are 'Save' and 'Apply' buttons.

4. Enter “**/*” in the Files to archive box.

The screenshot shows the 'Archive the artifacts' configuration block. It includes a 'Files to archive' input field containing '**/*' and an 'Advanced...' button.

5. Select **Add post-build action** again and select **VS Team Services Continuous Deployment**.

The screenshot shows the 'Add post-build action' dropdown menu. The 'VS Team Services Continuous Deployment' option is highlighted with a blue selection bar and has a blue border around it. At the bottom of the dropdown menu is a red-bordered 'Add post-build action ▾' button.

6. In the **VS Team Services Continuous Deployment** block, enter the following:

- Collection url:** Enter the URL to your VSTS account, such as [https://\[your-account-name\].visualstudio.com](https://[your-account-name].visualstudio.com)
- Team project:** Enter "BestForYouOrganics."
- Release definition:** Enter "BestForYouOrganics CD."

- d. **Username:** Enter "admin."
- e. **Password:** Paste the personal access token you copied previously from your VSTS account.

Trigger release in TFS/Team Services

Collection url: https://.visualstudio.com

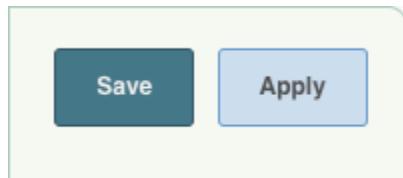
Team project: BestForYouOrganics

Release definition: BestForYouOrganics CD

Username: admin

Password or PAT:

7. Select **Save**.



Task 7: Create a Jenkins service endpoint in VSTS

In this task, you will configure a service endpoint for your Jenkins server in VSTS.

1. Open the **Services** page from your **BestForYouOrganics** project home page in VSTS, and select **New Service Endpoint**, and select **Jenkins**.

BestForYouOrganics / B... Overview Work Security Version Control Policies Agent Queues Notifications Service Hooks Test

Endpoints XAML Build Services

+ New Service Endpoint

Azure Classic

Azure Resource Manager

Service Fabric

Azure Service Bus

Chef

Docker Registry

Docker Host

Kubernetes

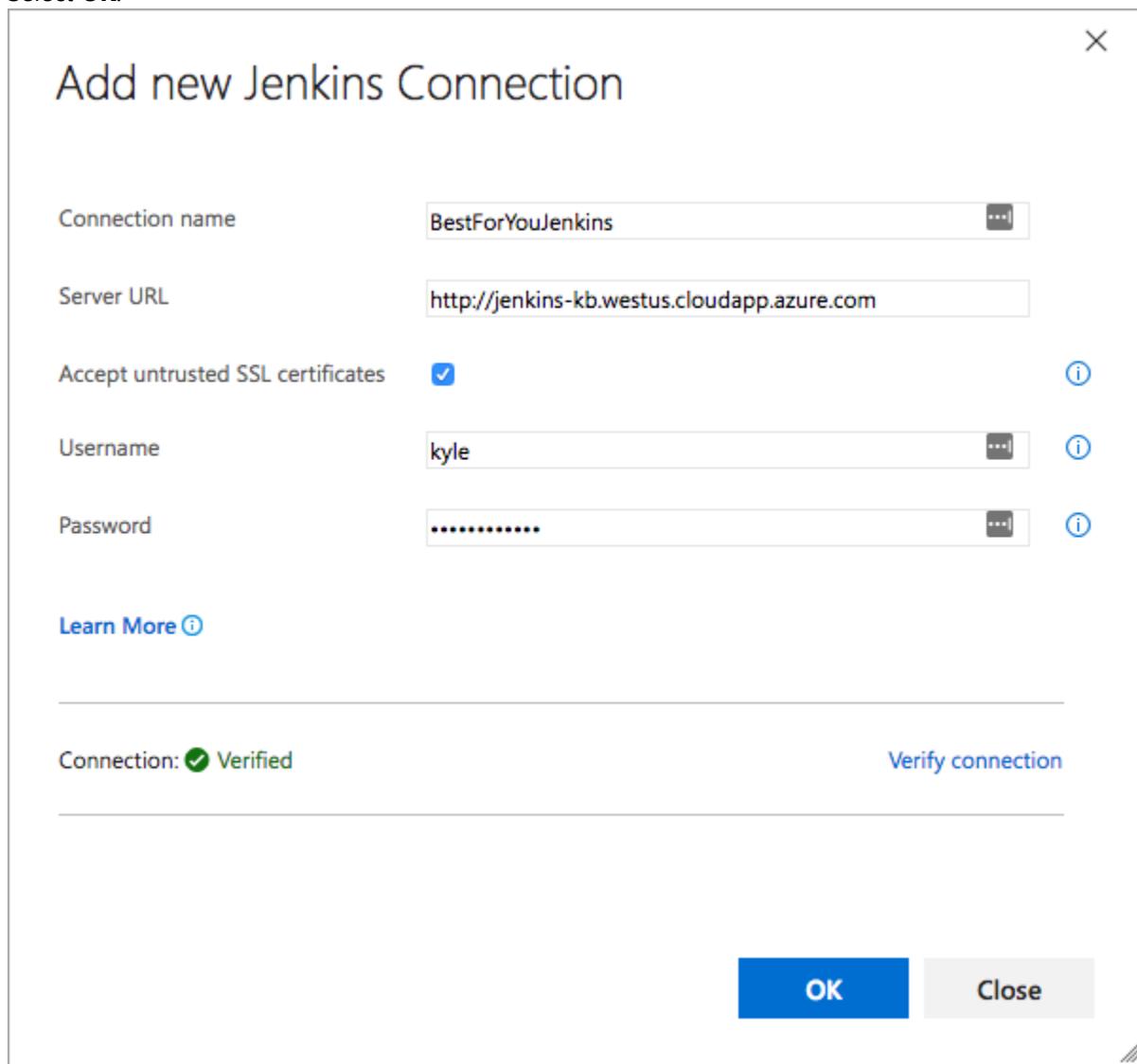
Jenkins

Visual Studio App Center

Services

No Services found.

2. In the **Add new Jenkins Connection** dialog, enter the following:
 - a. **Connection name:** Enter "BestForYouJenkins."
 - b. **Server URL:** Enter the URL of your Jenkins server, such as "http://{YourJenkinsUrl}.westus.cloudapp.azure.com."
 - i. You can obtain this by navigating to your Jenkins VM in the Azure portal and copying the *DNS name* on the VM's overview blade.
 - c. **Accept untrusted SSL certificates:** Check this.
 - d. **Username:** Enter the username you created in [Exercise 5, Task 2, Step 14](#).
 - e. **Password:** Enter "Password.1!!".
 - f. Select **Verify connection**, to check that the information is correct.
 - g. Select **OK**.



Task 8: Create a Team Services release definition

In this task, you will create the release definition in VSTS, which will take the build artifacts from Jenkins, and push them to the Azure Container Registry associated with your Web App for Containers.

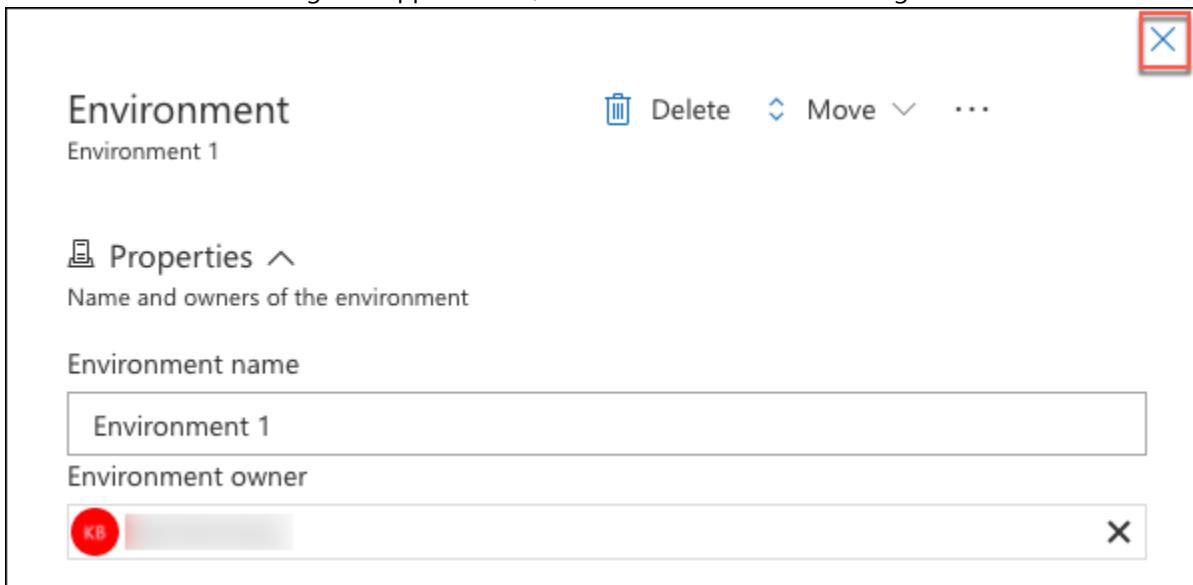
1. In VSTS, select **Build and Release, Releases**, and select **+New definition**.

The screenshot shows the VSTS interface with the 'Build and Release' tab selected. Below it, the 'Releases' tab is also selected and highlighted with a red box. On the left, there's a cloud icon with a rocket ship. To the right, a text block explains Release Management and provides a link to '+ New definition'. This link is also highlighted with a red box.

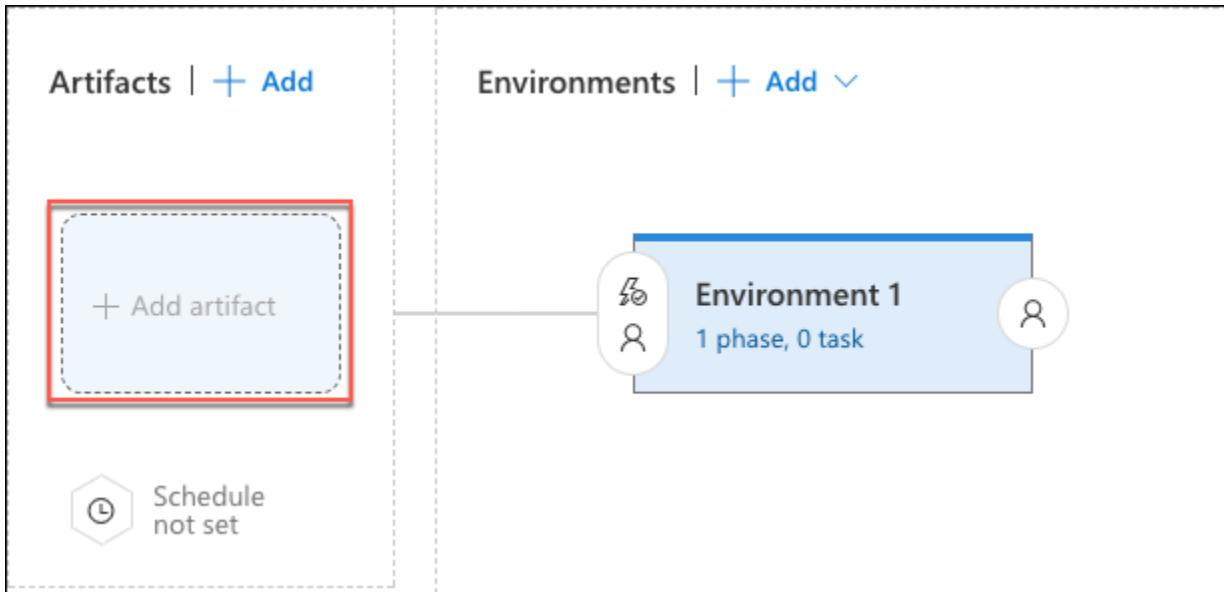
2. In the **Select a Template** dialog that appears, select **start with an Empty process**.

The screenshot shows the 'Select a Template' dialog. It has a search bar at the top right. Below it, a section labeled 'Or start with an' contains a button labeled 'Empty process', which is highlighted with a red box. Below this, there's a 'Featured' section with two items: 'Azure App Service Deployment' and 'Deploy Node.js App to Azure App Service', each with its own icon and brief description.

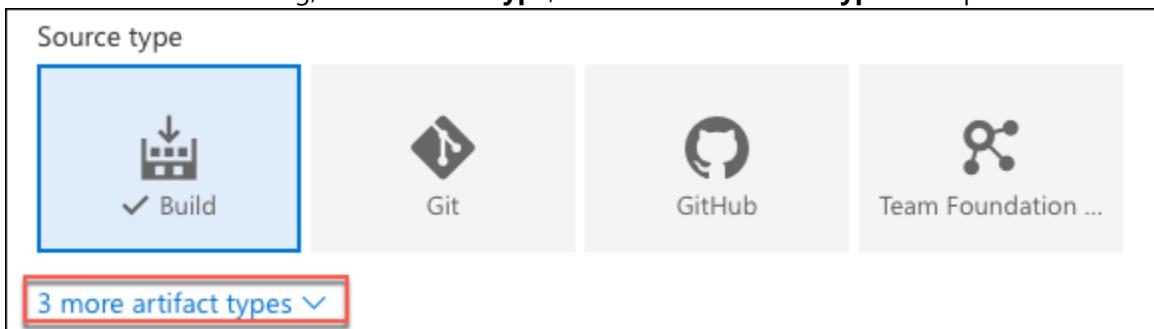
3. In the **Environment** dialog that appears next, select the **X** to close the dialog.



4. Next, select **+Add Artifact**.



5. In the **Add artifact** dialog, under **Source type**, select **3 more artifact types** to expand the entire list.



6. Select **Jenkins**.

Source type

Build

Git

GitHub

Team Foundatio...

Azure Container ...

Docker Hub

✓ Jenkins

Show less ^

7. In the Add artifact form, set the following:

- a. **Service Endpoint:** Select the **BestForYouJenkins** endpoint
- b. **Jenkins Job:** Select best-for-you-build
- c. **Default Version:** Leave Specify at the time of release creation selected
- d. **Source alias:** Enter "best-for-you-build"
- e. Select **Add**.

Service Endpoint * | Manage ↗

BestForYouJenkins

Jenkins Job *

best-for-you-build

Download artifacts from Azure storage ⓘ

Default version *

Specify at the time of release creation

Source alias ⓘ

best-for-you-build

ⓘ The artifacts published by each version will be available for deployment in Release Management. Select the version when you create a release. For automatically triggered releases, the latest version will be chosen.

Add

8. Select **Tasks** from the menu.

Pipeline Tasks | + Add Variables Retention Options History

Artifacts | + Add

best-for-you-build

Schedule not set

Environments | + Add

Environment 1
1 phase, 0 task

9. Select **Agent phase** and change the **Agent queue** to **Hosted Linux Preview**.

Pipeline Tasks | + Add Variables Retention Options History

Environment 1
Deployment process

Agent phase | Run on agent

Agent phase | +

Display name *

Agent phase

Agent selection ^

Agent queue | Manage

Hosted Linux Preview

10. Next, select the + in **Agent phase**, enter "Docker" in the Search box, and select **Add** next to the **Docker** task.

Pipeline Tasks | + Add Variables Retention Options History

Environment 1
Deployment process

Agent phase | Run on agent

+

Add tasks

Docker

Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure Container registry.

by Microsoft Corporation

Add

11. Repeat the previous step to add a second Docker task to the phase.

12. Now, change the text in the **Add tasks** search box to **Azure app service**, and select **Add** for the **Azure App Service Deploy** task.

Pipeline Tasks Variables Retention Options History

Environment 1 Deployment process

Agent phase Run on agent

+ Build an image Docker

+ Build an image Docker

Add tasks

Azure app service X

Don't see what you need? Check out our Marketplace. ↗

Azure App Service Deploy

Update Azure WebApp Services On Windows, Web App On Linux with built-in images or docker containers, ASP.NET, .NET Core, PHP, Python or Node based Web applications, Function Apps, Mobile Apps, API applications, Web Jobs using Web Deploy / Kudu REST APIs

by Microsoft Corporation Learn more ↗

Add

13. Your Environment should now look like the following.

Environment 1 Deployment process

Agent phase Run on agent

+ Build an image Docker

+ Build an image Docker

Azure App Service Deploy: Some settings need attention

14. Select the first Docker **Build an image** task under **Agent phase**, and set the following:

- Container Registry Type: Azure Container Registry**
- Azure subscription:** Select your subscription. You may need to select the **Authorize** button and enter your credentials first.
- Azure Container Registry:** Select **bestforyouregistrySUFFIX**.
- Action:** Leave as Build an image.
- Image name:** Enter **[Your Registry Name].azurecr.io/best-for-you-organics:v1**, for example, **bestforyouregistry.azurecr.io/best-for-you-organics:v1**.
- Include Latest Tag:** Check this.

- g. Leave all other fields set to their defaults.

The screenshot shows the Azure DevOps Pipeline Editor. On the left, there's a tree view with 'Environment 1' expanded, showing 'Deployment process', 'Agent phase' (selected), and 'Azure App Service Deploy: best-for-you-app'. Under 'Agent phase', there are two tasks: 'Build an image' (highlighted with a red box) and 'Push an image'. The 'Build an image' task is configured for Docker, version 0.*. It has a display name 'Build an image', uses 'Azure Container Registry' as the container registry type, and is set to build an image from '**/Dockerfile'. The 'Image Name' field contains 'bestforyouregistry.azurecr.io/best-for-you-organics:v1'. The 'Action' dropdown is set to 'Build an image'. The 'Docker File' field is '**/Dockerfile'. The 'Build Arguments' field is empty. Under 'Image Name', the 'Use Default Build Context' checkbox is checked. Under 'Additional Image Tags', the 'Include Latest Tag' checkbox is checked.

15. Select the second **Docker Build an image** task under Agent phase, and set the following:

- a. **Container Registry Type: Azure Container Registry**
- b. **Azure subscription:** Select your subscription.
- c. **Azure Container Registry:** Select **bestforyouregistrySUFFIX**.
- d. **Action:** Select **Push an image** (this will change the display name to Push an image as well).
- e. **Image name:** Enter [Your Registry Name].azurecr.io/best-for-you-organics:v1; for example, **bestforyouregistry.azurecr.io/best-for-you-organics:v1**.
- f. **Include Latest Tag:** Check this.

- g. Leave all other fields set to their defaults

The screenshot shows the Azure DevOps pipeline editor. On the left, there's a list of tasks: 'Build an image', 'Push an image' (which is highlighted with a red box), and 'Azure App Service Deploy: best-for-you-app'. On the right, the configuration for the 'Push an image' task is displayed. It includes fields for 'Docker' version (0.*), 'Display name' (Push an image), 'Container Registry Type' (Azure Container Registry), 'Azure subscription' (selected), 'Azure Container Registry' (bestforyouregistry), 'Action' (Push an image), 'Image Name' (bestforyouregistry.azurecr.io/best-for-you-organics:v1), and checkboxes for 'Qualify Image Name' (unchecked) and 'Include Latest Tag' (checked). There are also sections for 'Additional Image Tags' and 'Include Source Tags'.

16. Select the **Azure App Service Deploy**: task, and, enter the following:

- Azure Subscription:** Select your subscription. Select **Authorize**, and enter your Azure account credentials, if prompted.
- App type:** Select **Linux App**.
- App service name:** Enter "best-for-you-app-SUFFIX" (you may have to manually enter this, if it doesn't show up in the list).
- Image source: Container Registry.**
- Registry or Namespace:** Enter "[Your Registry Name].azurecr.io," for example, bestforyouregistry.azurecr.io.
- Image:** Enter "best-for-you-organics."

g. **Tag:** Enter v1

Azure App Service Deploy ①

Version 3.*

Display name * Azure App Service Deploy: best-for-you-app

Azure subscription * Manage ②

App type * ① Linux Web App

App Service name * ① best-for-you-app

Deploy to slot ①

Image Source ① Container Registry

Registry or Namespace * ① bestforyouregistry.azurecr.io

Image * ① best-for-you-organics

Tag ① v1

17. Finally, enter "BestForYouOrganics CD" for the release definition name, and select **Save**. (Note: the release definition name must match the release definition name you entered in your Jenkins build definition.)

Builds Releases Library Task Groups Deployment Groups*

All definitions > BestForYouOrganics CD

Save

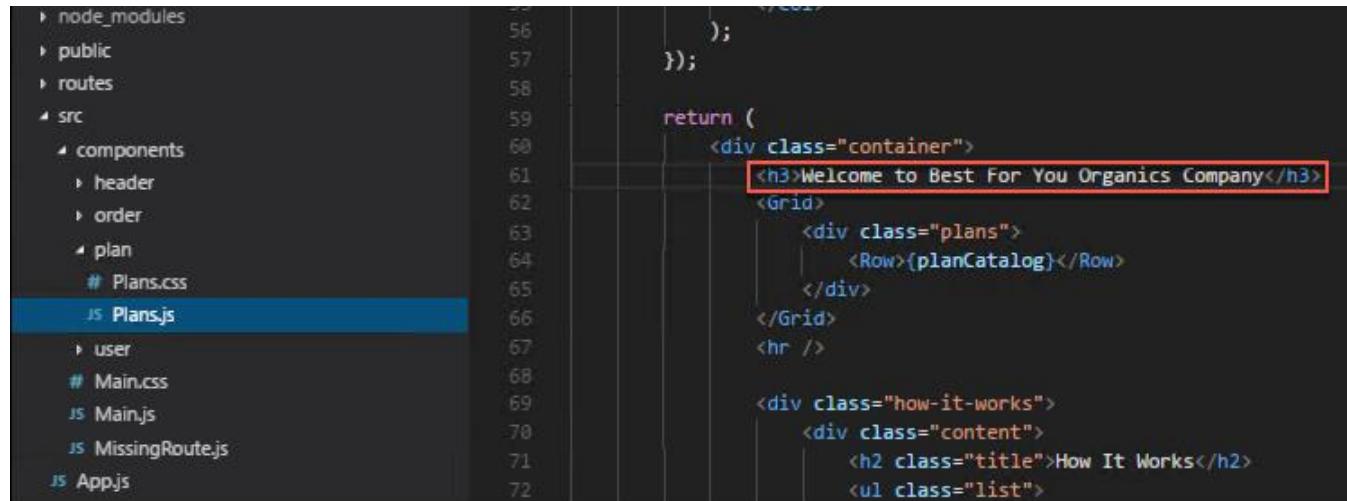
Pipeline Tasks Variables Retention Options History

Task 9: Trigger CI/CD pipeline

In this task, you will commit a change to the mcw-oss-paas-devops starter application and trigger the full CI/CD pipeline through Jenkins and VSTS.

1. Return to VS Code on your Lab VM, open the src/components/plan/Plans.js file, and insert the following markup between <div class="container"> and <Grid>.

```
<h3>Welcome to Best for You Organics Company</h3>
```



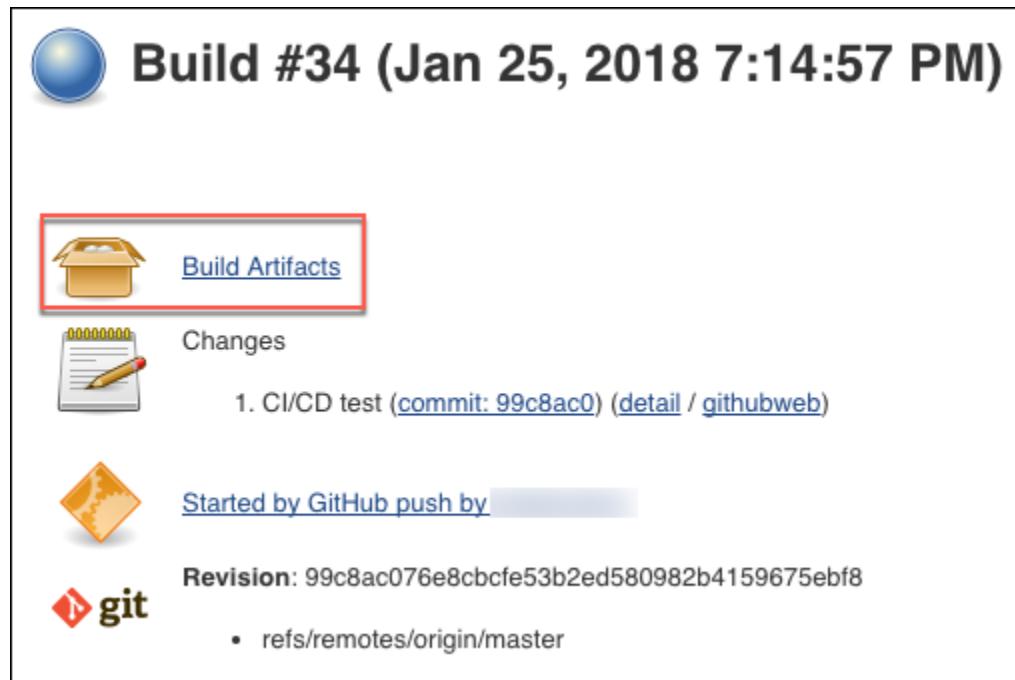
```
node_modules
public
routes
src
  components
    header
    order
    plan
      # Plans.css
      JS Plans.js
    user
      # Main.css
      JS Main.js
      JS MissingRoute.js
    JS App.js
```

```
56      );
57    });
58
59    return (
60      <div class="container">
61        <h3>Welcome to Best For You Organics Company</h3>
62        <Grid>
63          <div class="plans">
64            <Row>{planCatalog}</Row>
65          </div>
66        </Grid>
67        <hr />
68
69        <div class="how-it-works">
70          <div class="content">
71            <h2 class="title">How It Works</h2>
72            <ul class="list">
```

2. Save the file.
3. Run the following command in the Integrated terminal.

```
npm run build
```

4. As you did in [Task 3](#), above, select the **Source Control icon** from the left-hand menu, enter a commit comment, select + to stage the change, and select the **checkmark** to commit the change, and push to GitHub. This will trigger the CI/CD pipeline, starting with the Jenkins build.
5. Return to your Jenkins dashboard, and select the **best-for-you-build project**, select the **latest build**, and note the build number.



Build #34 (Jan 25, 2018 7:14:57 PM)

 [Build Artifacts](#)

 Changes

1. CI/CD test (commit: 99c8ac0) ([detail](#) / [githubweb](#))

 Started by GitHub push by

Revision: 99c8ac076e8cbcfe53b2ed580982b4159675ebf8

• refs/remotes/origin/master

6. Next, go to the **Releases** page in your VSTS account, and you will see an active release, with the same build number as you noted in Jenkins.

Lock	⚡ Title	Release Definition	Environments	Build
🔒	Release-3	... BestForYouOrganics CD	▶	#34 (Jenkins)
🔒	Release-2	BestForYouOrganics CD	✓	#33 (Jenkins)
🔒	Release-1	BestForYouOrganics CD	✗	

7. Select the **ellipsis** next to the active release and select **Open**.

- ...
- ↗ Open (highlighted)
- ↗ Open in new tab
- Start
- 🔒 Retain indefinitely
- Abandon
- Delete

8. On the summary tab, you can see that the release is in progress. Select the **Logs** tabs to see that real-time progress of the release.

Step	Action	Agent queue: Hosted VS2017 Agent: Hosted Agent	Start Time: 1/25
Environment 1	...	Downloaded 'node_modules/es5-ext/number/maxSafeInteger.js' to 'd:\a\1\a\best-for-you-build\node_modules\es5-ext\index.js'	
Pre-deployment approval	...	Downloading 'node_modules/es5-ext/number/minSafeInteger/implement.js' to 'd:\a\1\a\best-for-you-build\node_modules\es5-ext\index.js'	
Run on agent	...	Downloading 'node_modules/es5-ext/number/minSafeInteger/index.js' to 'd:\a\1\a\best-for-you-build\node_modules\es5-ext\index.js'	
Initialize Agent	...	Downloading 'node_modules/es5-ext/number/toInteger.js' to 'd:\a\1\a\best-for-you-build\node_modules\es5-ext\index.js'	
Initialize Job	...	Downloading 'node_modules/es5-ext/number/minSafeInteger/isImplemented.js' to 'd:\a\1\a\best-for-you-build\node_modules\es5-ext\index.js'	
Download artifact - best-for-you-build	...	Downloading 'node_modules/es5-ext/number/toPosInteger.js' to 'd:\a\1\a\best-for-you-build\node_modules\es5-ext\index.js'	
Deploy Azure App Service	...	Downloading 'node_modules/es5-ext/number/toUInt32.js' to 'd:\a\1\a\best-for-you-build\node_modules\es5-ext\index.js'	

9. When the deployment is complete, check change in by going to Azure reloading the web page for your App Service in the browser. The deployment can take 30+ minutes, so move on to the next exercise, and come back and check the deployment progress periodically.

10. When the deployment is complete you should see the home page, with a new header above the three plans on the page.

The screenshot shows a web application interface for a meal delivery service. At the top, there is a dark navigation bar with a logo icon, the word "Home", and a "Sign In" button. Below this, a large red rectangular box highlights the main heading "Welcome to Best for You Organics Company!". The page features three distinct sections, each representing a meal plan:

- Two Person Plan**: \$72/Week. This plan is described as a "basic plan, delivering 3 meals per week, which will feed 1-2 people." It includes a small orange circular logo with a white stylized leaf or swirl design. A yellow "Select this plan" button is located at the bottom of this section.
- Four Person Plan**: \$87/Week. This plan is described as a "family plan, delivering 3 meals per week, which will feed 3-4 people." It includes a similar orange circular logo. A yellow "Select this plan" button is located at the bottom of this section.
- High-Pro Plan**: \$80/Week. This plan is described as being "specially formulated for athletes and active individuals, delivering 3 meals per week, for 1-2 people." It includes a similar orange circular logo. A yellow "Select this plan" button is located at the bottom of this section.

Exercise 6: Create Azure Function for order processing

Duration: 45 minutes

In this task, you will create an Azure Function that will be triggered by orders being added to the Orders collection in Azure Cosmos DB. This Function will trigger whenever a document in the orders collection is inserted or updated. The function checks the processed field on the order document, ensuring only unprocessed orders are sent to the processing queue.

Task 1: Provision a Function App

In this task, you will create a Function App in Azure, which will host your Functions.

1. In the Azure portal, select **+Create a resource**, enter “function app” in to the **Search the marketplace** box, and select **Function App** from the results.

NAME	PUBLISHER	CATEGORY
Function App	Microsoft	Web + Mobile
Functions Bot	Microsoft	AI + Cognitive Services

2. On the **Function App** blade, select **Create**.
3. On the **Create Function App** blade, enter the following:
 - a. **App name:** Enter a unique name, such as “bestforyouordersSUFFIX.”
 - b. **Subscription:** Select the subscription you are using for this hands-on lab.
 - c. **Resource group:** Select **Use existing** and choose the **hands-on-lab-SUFFIX** resource group.
 - d. **OS:** Select Windows.
 - e. **Hosting Plan:** Choose Consumption Plan.
 - f. **Location:** Select the location you’ve been using for resources in this hands-on lab.
 - g. **Storage:** Select **Create new** and enter “bestforyouorders” for the name.

- h. Select **Create** to provision the new Function App.

The screenshot shows the 'Function App' creation dialog box. It includes fields for App name ('bestforyouorders'), Subscription ('Solliance MVP MSDN'), Resource Group ('hands-on-labs'), OS ('Windows'), Hosting Plan ('Consumption Plan'), Location ('West US'), Storage ('bestforyouorders'), Application Insights ('Off'), and a 'Pin to dashboard' checkbox. At the bottom are 'Create' and 'Automation options' buttons.

Function App

Create

* App name
bestforyouorders .azurewebsites.net

* Subscription
Solliance MVP MSDN

* Resource Group ⓘ
Create new Use existing
hands-on-labs

* OS Windows Linux (Preview)

* Hosting Plan ⓘ
Consumption Plan

* Location
West US

* Storage ⓘ
Create new Use existing
bestforyouorders

Application Insights ⓘ On Off

Pin to dashboard

Create Automation options

Task 2: Configure storage queues

In this task, you will add two storage queues to the storage account provisioned when you created your Function App. These queues will be used to store orders and notifications needing to be processed.

- In the Azure portal, navigate to the new **bestforyouorders storage account** that was created when you provisioned your Function App, by selecting **Resource groups** from the left-hand menu, selecting your **hands-on-lab-SUFFIX** resource group from the list, and then selecting the **bestforyouorders storage account**.

The screenshot shows the Azure portal's Resource groups blade. On the left, under 'Resource groups', there is a search bar with 'hands' typed in, and a list of items with one item named 'hands-on-labs'. On the right, the 'hands-on-labs' resource group is selected, showing its Overview page. In the 'Overview' section, there is a list of resources, including a storage account named 'bestforyouorders' which is highlighted with a red box.

- Select **Queues** from the **Services** area of the **Overview** blade.

The screenshot shows the 'Services' area of the Azure portal's Overview blade. It lists four services: Blobs, Files, Tables, and Queues. The 'Queues' service is highlighted with a red box. Each service has a description and links to 'View metrics', 'Configure CORS rules', and 'Setup custom domain'.

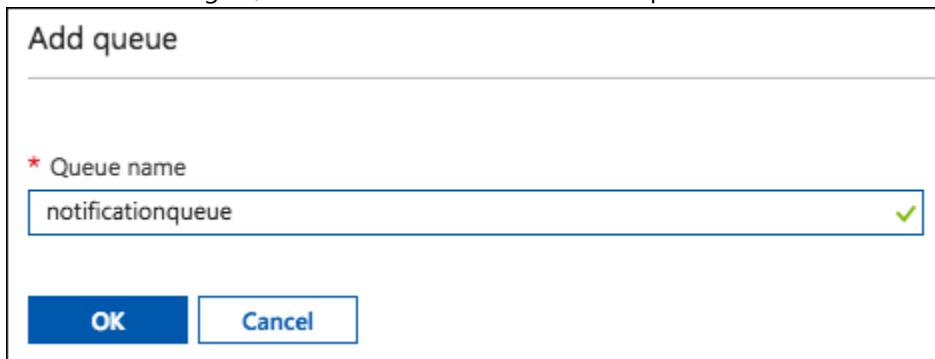
- On the **Queue service** blade, select **+Queue** to add a new queue.

The screenshot shows the 'Queue service' blade for the 'bestforyouorders' storage account. At the top, there are buttons for 'Refresh', '+ Queue' (which is highlighted with a red box), and 'Delete queues'. Below the buttons, it shows the storage account name 'bestforyouorders'.

- In the **Add queue** dialog, enter **orderqueue** for the **Queue name**, and select **OK**.

The screenshot shows the 'Add queue' dialog. It has a text input field labeled 'Queue name' with the value 'orderqueue'. At the bottom, there are two buttons: 'OK' (highlighted with a red box) and 'Cancel'.

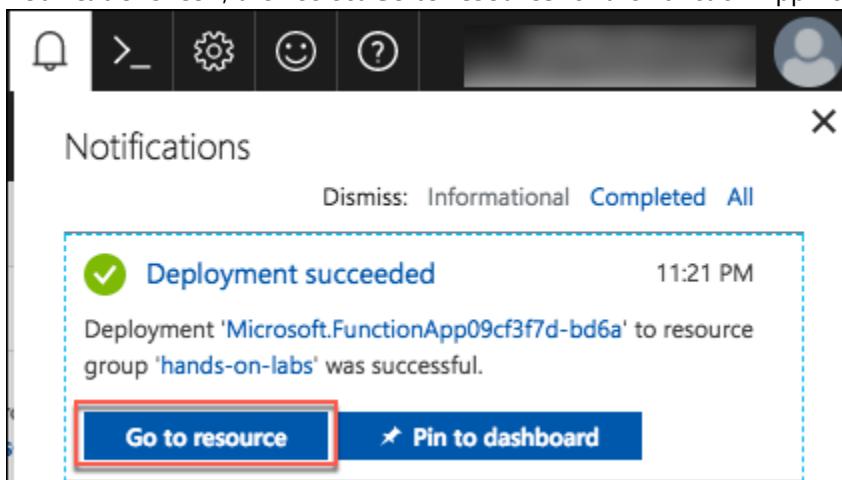
5. Select **+Queue** again, and this time enter “notificationqueue” for the **Queue name**.



Task 3: Create Cosmos DB trigger function

In this task, you will create a function that will be triggered whenever a document is inserted into the orders collection in your Azure Cosmos DB. This function sends all new orders to a queue for processing and shipping. This function will use a Cosmos DB trigger and an output binding to an Azure Storage Queue.

1. When the Function App has deployed, navigate to the new Function App in the Azure portal, by selecting the **notifications icon**, then select **Go to resource** for the Function App notification.



2. From the left-hand menu on your **Function Apps** blade, select **Functions**, then select **+New function**.

The screenshot shows the 'bestforyouorders' Function Apps blade. On the left, there's a sidebar with a search bar ('bestforyouorders'), a dropdown for 'All subscriptions', and a tree view of resources: 'Function Apps' (selected), 'bestforyouorders' (selected), 'Functions' (highlighted with a red box), 'Proxies', and 'Slots (preview)'. On the right, under the 'Functions' heading, there's a search bar ('Search functions'), a table header ('NAME ▾ STATUS ▾'), and a message 'No results'.

3. In the trigger search box, enter "cosmos," and select the **Cosmos DB trigger**.

The screenshot shows a search interface with a search bar containing 'cosmos' (highlighted with a red box), a 'Language' dropdown set to 'All', and a 'Scenario' dropdown set to 'All'. Below the search bar, a card for the 'Cosmos DB trigger' is displayed, featuring a preview image, the name 'Cosmos DB trigger', a description 'A function that will be run whenever documents change in a document collection', and language options 'C# JavaScript'.

4. In the **Cosmos DB trigger** dialog, enter the following:

- Language:** Select **JavaScript**.
- Name:** Enter "OrdersCosmosTrigger."
- Azure Cosmos DB account connection:** Select **new**, then select the **best-for-you-db DocumentDB Account**.
- Collection name:** Enter orders (use all lowercase, as case matters).
- Create lease collection if it does not exist:** Leave this checked.
- Database name:** Enter "best-for-you-organics."

- g. **Collection name for leases:** Leave set to leases.
- h. Select **Create**.

Cosmos DB trigger

New Function

Language:

JavaScript

Name:

OrdersCosmosTrigger

Azure Cosmos DB trigger

Azure Cosmos DB account connection i new show value
best-for-you_DOCUMENTDB

Collection name i
orders

Create lease collection if it does not exist i

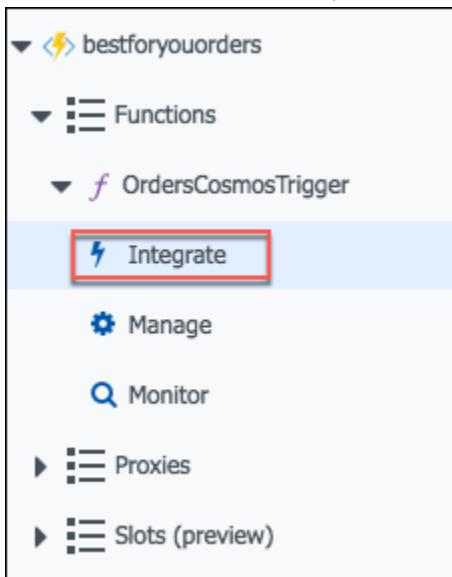
Database name i
best-for-you-organics

Collection name for leases i
leases

Create

Cancel

5. After the function is created, select **Integrate** under the new function.



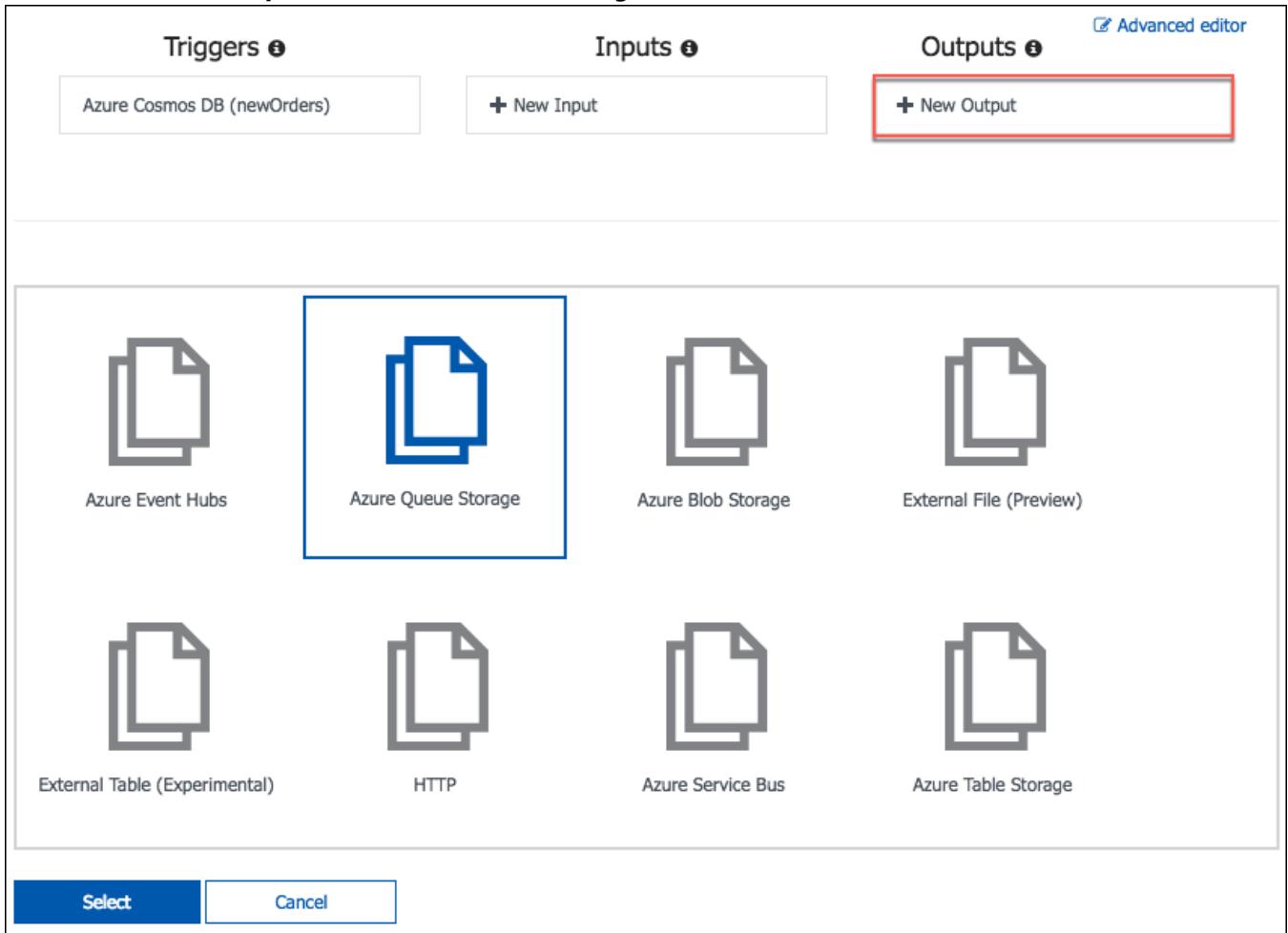
6. Change the Document collection parameter name to "newOrders" for the **Azure Cosmos DB trigger** and select **Save**.

A screenshot of the 'Azure Cosmos DB trigger' configuration dialog. It has three main sections: Triggers, Inputs, and Outputs. The 'Triggers' section shows 'Azure Cosmos DB (input)'. The 'Inputs' section has a 'New Input' button. The 'Outputs' section has a 'New Output' button. Below these sections is a summary of the trigger settings:

Document collection parameter name	newOrders
Azure Cosmos DB account connection	best-for-you_DOCUMENTDB
Database name	best-for-you-organics
Collection name	orders
Collection name for leases	leases
Create lease collection if it does not exist	<input checked="" type="checkbox"/>

At the bottom are 'Save' and 'Cancel' buttons.

7. Next, select **+New Output**, select **Azure Queue Storage**, and select **Select**.



8. For the **Azure Queue Storage output**, enter the following:

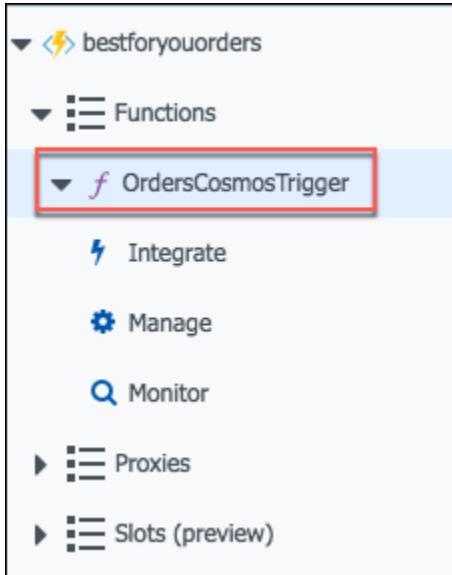
- Message parameter name:** outputQueue.
- Queue name:** orderqueue (all lowercase, as casing matters).
- Storage account collection:** Select **AzureWebJobsStorage** from the list (this is the best for you orders storage account you created when you provisioned your Function App).
- Select **Save**.

The screenshot shows the 'Azure Queue Storage output' configuration dialog. It contains the following fields:

- Message parameter name:** A text input field containing 'outputQueue'.
- Queue name:** A text input field containing 'orderqueue'.
- Storage account connection:** A dropdown menu set to 'AzureWebJobsStorage'.

At the bottom of the dialog are two buttons: 'Save' (highlighted with a blue border) and 'Cancel'.

9. Now, select the **OrdersCosmosTrigger** function in the left-hand menu.



10. To get the code for the OrdersCosmosTrigger function, go into the project in VS Code, expand the AzureFunctions folder, select **OrdersCosmosTrigger.js**, and copy the code, as highlighted in the screen shot below.

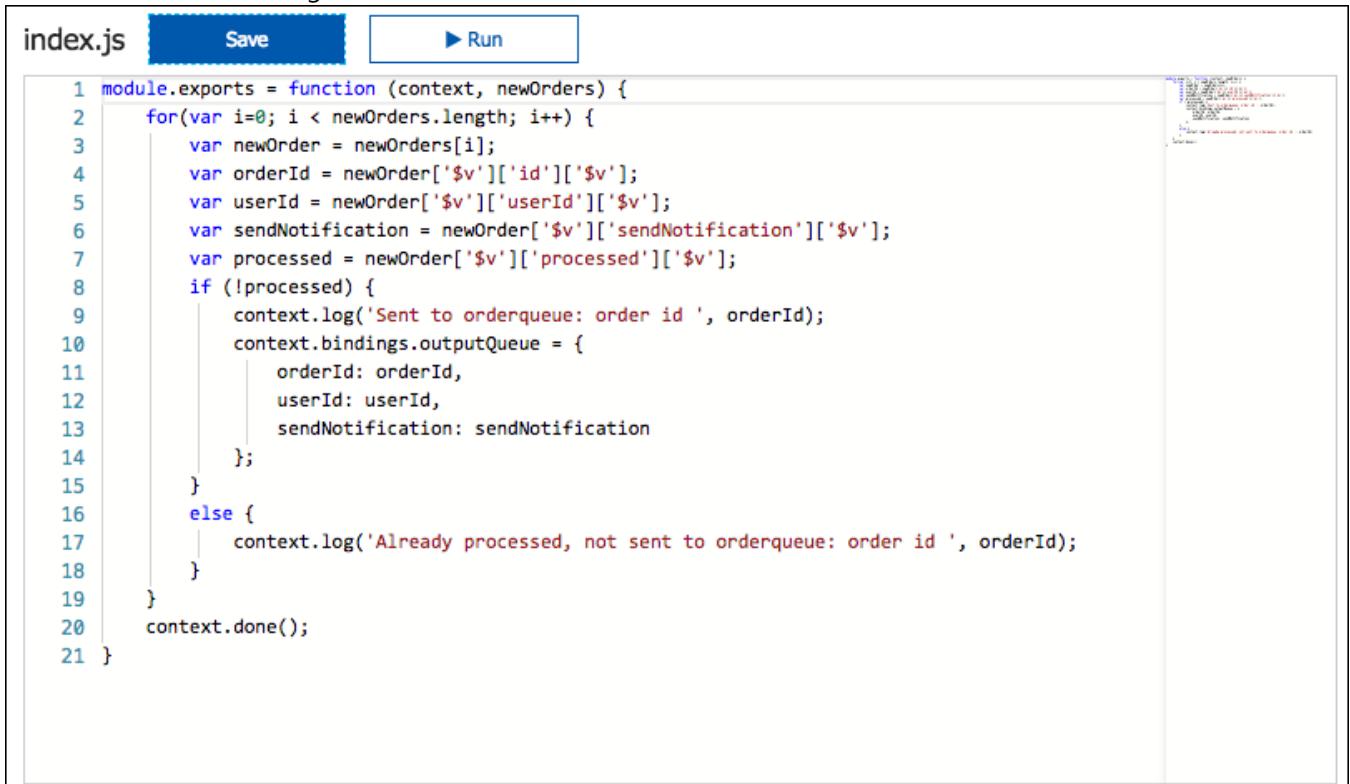
The screenshot shows the VS Code interface with the following tabs:

- EXPLORER
- Plans.js
- Submit.js
- OrdersCosmosTrigger.js (highlighted with a red box)
- order.js
- Order.js
- JS

The 'OrdersCosmosTrigger.js' file contains the following code:

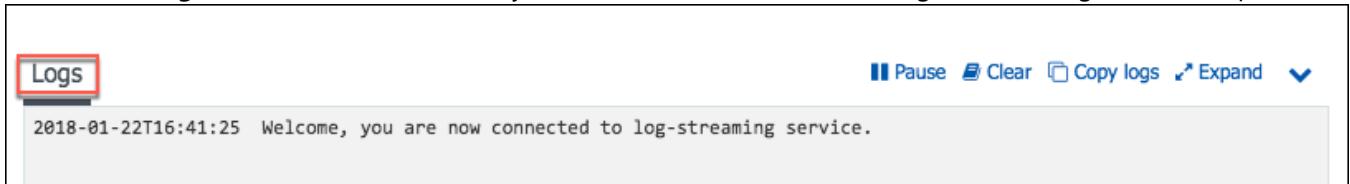
```
/*
 * OrdersCosmosTrigger function code
 * Trigger: Azure Cosmos DB on newOrders into the orders collection,
 *           * Output: Azure Storage Queue - orderqueue
 *
 * Triggers whenever a document in the orders collection is inserted or updated.
 * By checking processed on the document, we only send unprocessed orders to the queue,
 * so when the next Function updates the processed value to true, it won't be requeued.
 */
module.exports = function (context, newOrders) {
    for(var i=0; i < newOrders.length; i++) {
        var newOrder = newOrders[i];
        var orderId = newOrder['$v']['id']['$v'];
        var userId = newOrder['$v']['userId']['$v'];
        var sendNotification = newOrder['$v']['sendNotification']['$v'];
        var processed = newOrder['$v']['processed']['$v'];
        if (!processed) {
            context.log('Sent to orderqueue: order id ', orderId);
            context.bindings.outputQueue = {
                orderId: orderId,
                userId: userId,
                sendNotification: sendNotification
            };
        } else {
            context.log('Already processed, not sent to orderqueue: order id ', orderId);
        }
    }
    context.done();
}
```

11. Paste the code into the **index.js** block, overwriting all the existing code, and select **Save**. Your index.js file should now look like the following:



```
index.js Save ▶ Run
1 module.exports = function (context, newOrders) {
2     for(var i=0; i < newOrders.length; i++) {
3         var newOrder = newOrders[i];
4         var orderId = newOrder['$v']['id']['$v'];
5         var userId = newOrder['$v']['userId']['$v'];
6         var sendNotification = newOrder['$v']['sendNotification']['$v'];
7         var processed = newOrder['$v']['processed']['$v'];
8         if (!processed) {
9             context.log('Sent to orderqueue: order id ', orderId);
10            context.bindings.outputQueue = {
11                orderId: orderId,
12                userId: userId,
13                sendNotification: sendNotification
14            };
15        } else {
16            context.log('Already processed, not sent to orderqueue: order id ', orderId);
17        }
18    }
19 }
20 context.done();
21 }
```

12. Next, select **Logs** below the code block, so you can observe the Function being called during the next steps.



Logs Pause Clear Copy logs Expand

```
2018-01-22T16:41:25 Welcome, you are now connected to log-streaming service.
```

13. To trigger the function, return to the starter application in your browser window, and select **Sign In**.

The screenshot shows a meal delivery service interface. At the top right is a "Sign In" button. Below it are three meal plan options:

- Two Person Plan**: \$72/Week. Description: "Our basic plan, delivering 3 meals per week, which will feed 1-2 people." Includes a "Select this plan" button.
- High-Pro Plan**: \$80/Week. Description: "Specially formulated for athletes and active individuals, delivering 3 meals per week, for 1-2 people." Includes a "Select this plan" button.
- Four Person Plan**: \$87/Week. Description: "Our family plan, delivering 3 meals per week, which will feed 3-4 people." Includes a "Select this plan" button.

14. On the Login screen, enter the following credentials, and select **Login**.

- Email address:** demouser@bfyo.com.
- Password:** Password.1!!

The screenshot shows a login form with the following fields and buttons:

- Email Address:** demouser@bfyo.com
- Password:** (Redacted)
- Login** button
- Not registered yet? Register now!** link
- Register** button

15. After logging in, you will be returned to the home page. Select **Select this plan** for any of the plans.

Two Person Plan	High-Pro Plan	Four Person Plan
1-2 Person	1-2 Person	3-4 Person
3 Unique meals per week Our basic plan, delivering 3 meals per week, which will feed 1-2 people.	3 High protein meals per week Specially formulated for athletes and active individuals, delivering 3 meals per week, for 1-2 people.	3 Unique meals per week Our family plan, delivering 3 meals per week, which will feed 3-4 people.
\$72 /Week	\$80 /Week	\$87 /Week
Select this plan	Select this plan	Select this plan

16. On the **Place Order** screen, select **Place Order**. This will create a new order, which will fire the Azure Cosmos DB trigger in your function, and then send the order on to the ordersqueue for processing.

Place Order for the Four Person Plan

[Back to all Plans](#)

Selected Plan:
Four Person Plan

Credit Card Number:
4111111111111111

CVV:
...

Place Order

17. Next, you will update an order in the Azure portal, to set the processed value to true. This will be a change that should not be sent into the orderqueue for processing.

18. Navigate to your Cosmos DB account in the Azure portal, select **Data Explorer**, expand the **orders** collection, then select **Documents**.

The screenshot shows the Azure Data Explorer interface for the 'best-for-you' database. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, and Data Explorer. The Data Explorer link is highlighted with a red box. Below it is a SETTINGS section with a Connection String link. The main area is titled 'COLLECTIONS' and shows a tree view of the database. Under 'best-for-you-organics', the 'sessions', 'users', 'plans', and 'orders' collections are listed. The 'orders' collection is expanded, and its 'Documents' sub-section is selected and highlighted with a red box. Other options under 'orders' include Scale & Settings, Stored Procedures, User Defined Functions, Triggers, and leases.

19. Select any order document, and change the processed value to "true," then select **Update**.

The screenshot shows the Azure Data Explorer document editor for an order document. At the top, there are buttons for New Document, Update (which is highlighted with a red box), Discard, and Delete. Below that, there are filters: 'No filter applied' and 'Edit Filter'. The document list shows several order IDs, and one specific document is selected and highlighted with a blue box. The document details pane on the right shows the JSON structure of the document. The 'processed' field is currently set to 'false', which is highlighted with a red box. A cursor is positioned over the 'true' value, indicating it is being edited. The JSON code is as follows:

```
1 {  
2     "_id" : ObjectId("5a660b3e51a250ee340a1691"),  
3     "id" : "5a660b3e51a250ee340a1692",  
4     "userId" : "5a64c2cc29339c0540e68d1e",  
5     "planId" : "5a64c2cc29339c0540e68d1c",  
6     "processed" : true, // This line is being edited  
7     "notificationSent" : false,  
8     "sendNotification" : false,  
9     "orderDate" : {  
10         "$date" : 1516636990845  
11     },  
12     "__v" : 0  
13 }
```

20. Return to the logs pane of your function and observe that the orders have been processed though the Function, and that the new order was sent to the orderqueue, while the updated order was not.

```

Logs
Pause Clear Copy logs Expand ▾

2018-01-22T16:18:59 Welcome, you are now connected to log-streaming service.
2018-01-22T16:19:11.438 Function started (Id=4bde9e86-d7dc-4482-9f26-63b2a281205b)
2018-01-22T16:19:11.641 Sent to orderqueue: order id 5a660efc51a250ee340a169a
2018-01-22T16:19:11.750 Function completed (Success, Id=4bde9e86-d7dc-4482-9f26-63b2a281205b, Duration=311ms)
2018-01-22T16:20:01.907 Function started (Id=435df55b-9521-4ceb-9ca4-6b190c318022)
2018-01-22T16:20:01.907 Sent to orderqueue: order id 5a660f2d51a250ee340a169a
2018-01-22T16:20:01.942 Function completed (Success, Id=435df55b-9521-4ceb-9ca4-6b190c318022, Duration=29ms)
2018-01-22T16:21:57.254 Function started (Id=edb34f46-cb19-4027-97e3-09f5ccc289d6)
2018-01-22T16:21:57.254 Already processed, not sent to orderqueue: order id 5a660b3e51a250ee340a1692
2018-01-22T16:21:57.254 Function completed (Success, Id=edb34f46-cb19-4027-97e3-09f5ccc289d6, Duration=2ms)

```

21. Finally, verify items are being written to the order queue, by going to the queue in the Azure Storage account, and observing that items have been added to the queue.

ID	MESSAGE TEXT	I...	EX...	DEQUEUE C...
5071c3eb...	{"orderId": "5a660efc51a250ee340a1698", "userId": "5a64c2cc29339c0540e68d1e", "sendNotification": true}	...	M...	0
a830a396...	{"orderId": "5a660f2d51a250ee340a169a", "userId": "5a64c2cc29339c0540e68d1e", "sendNotification": false}	...	M...	0

Task 4: Create Queue function

In this task, you will create a second function which will be triggered the output of the OrdersCosmosTrigger function. This will simulate the order processing and will add items to the notificationqueue if the order processing is complete and sendNotifications is true for the order.

This will use an Azure Storage Queue trigger, and an input dataset from Cosmos DB, pulling in customers. Output dataset will be Azure Cosmos DB orders table, and an update to set processed = true, and the processedDate to today.

- Select **Integrate** under the OrdersCosmosTrigger function, then select **Azure Queue Storage (outputQueue)** under **Outputs**.

Triggers	Inputs	Outputs
Azure Cosmos DB (newOrders)	+ New Input	Azure Queue Storage (outputQueue)

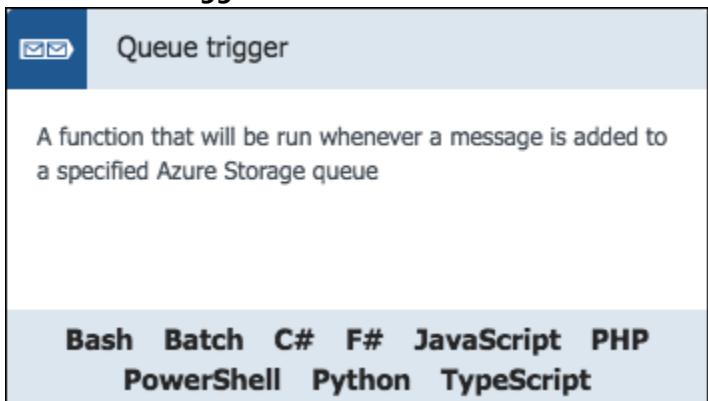
- Under **Actions** for the output, select **Go** next to **Create a new function triggered by this output**.

Actions

Create a new function triggered by this output

Go

3. Select **Queue trigger** from the list.

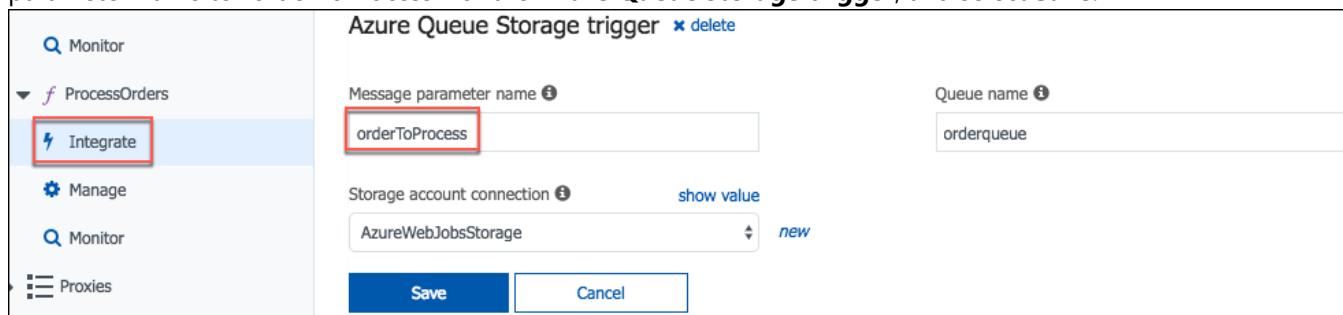


4. On the **Queue trigger New Function** dialog, enter the following:

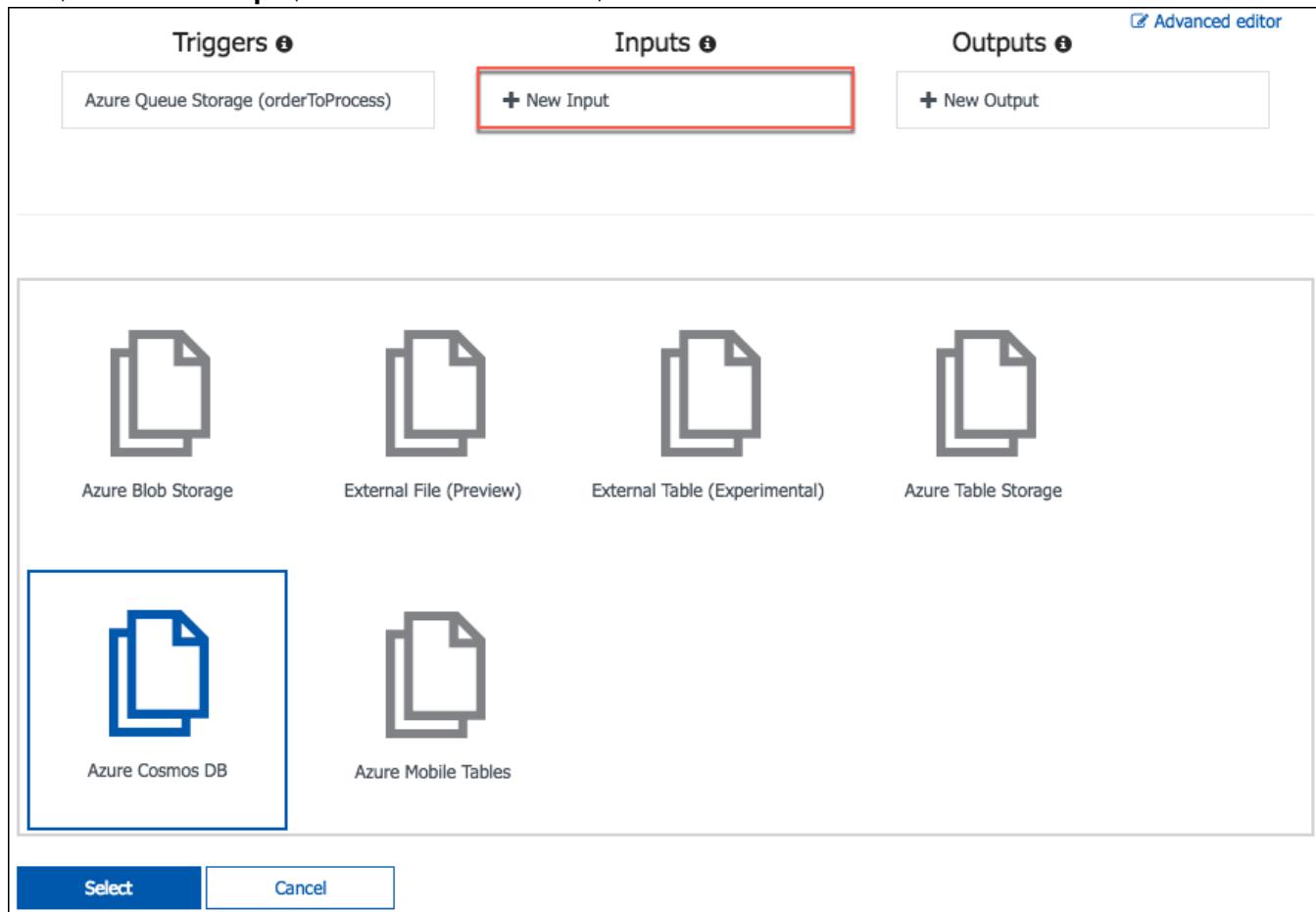
- Language:** Select **JavaScript**.
- Name:** Enter **ProcessOrders**.
- Queue name:** **orderqueue**.
- Storage account connection:** Select **AzureWebJobsStorage**.
- Select **Create**.

A screenshot of the "New Function" dialog for a "Queue trigger". The "Language" dropdown is set to "JavaScript". The "Name" field contains "ProcessOrders". Under the "Azure Queue Storage trigger" section, the "Queue name" dropdown is set to "orderqueue". The "Storage account connection" dropdown is set to "AzureWebJobsStorage". At the bottom are "Create" and "Cancel" buttons.

5. When the function has been created, select **Integrate** under the **ProcessOrders** function, change the Message parameter name to "orderToProcess" for the **Azure Queue storage trigger**, and select **Save**.



6. Now, select **+New Input**, select **Azure Cosmos DB**, and select **Select**.



7. On the **Azure Cosmos DB** input screen, enter the following:

- Document parameter name:** Enter "users."
- Database name:** Enter "best-for-you-organics."
- Collection name:** Enter "users" (all lowercase, as case matters).
- Azure Cosmos DB account connection:** Select **best-for-you_DOCUMENTDB**.

- e. Select **Save**.

Azure Cosmos DB input

Document parameter name <small>i</small>	Database name <small>i</small>
users	best-for-you-organics
Collection Name <small>i</small>	Document ID (optional) <small>i</small>
users	Document ID (optional)
Partition key (optional) <small>i</small>	SQL Query (optional) <small>i</small>
Partition key (optional)	SQL Query (optional)
Azure Cosmos DB account connection <small>i</small>	show value
best-for-you_DOCUMENTDB	<small>new</small>
Save Cancel	

8. Next, select **+New Output**, select **Azure Queue Storage**, and select **Select**.

Triggers i

Azure Queue Storage (orderToProcess)

Inputs i

Azure Cosmos DB (users)

Outputs i

Advanced editor

+ New Output

Outputs

 Azure Event Hubs

 Azure Queue Storage

 Azure Blob Storage

 External File (Preview)

 External Table (Experimental)

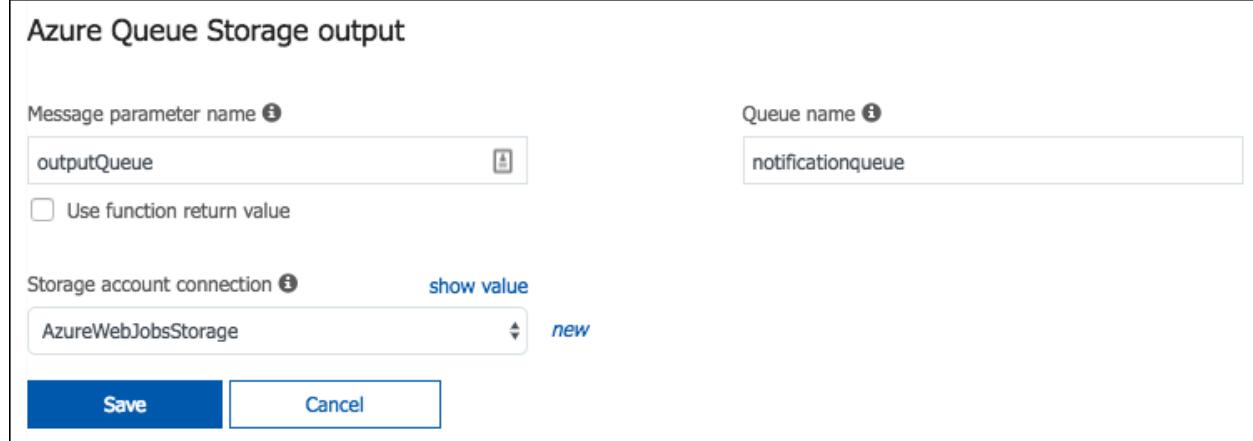
 HTTP

 Azure Service Bus

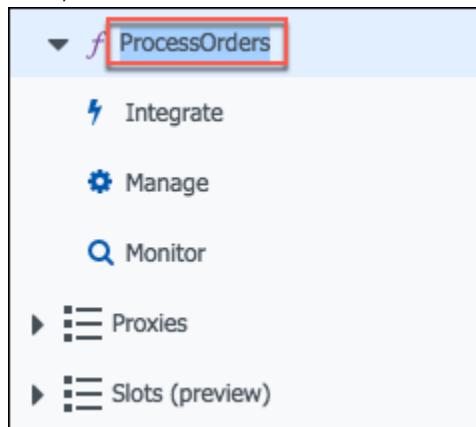
 Azure Table Storage

Select **Cancel**

9. For the **Azure Queue Storage output**, enter the following:
- Message parameter name:** "outputQueue."
 - Queue name:** "notificationqueue" (all lowercase, as casing matters).
 - Storage account collection:** Select **AzureWebJobsStorage** from the list
 - Select **Save**.



10. Now, select the **ProcessOrders** function in the left-hand menu.



11. To get the code for the **ProcessOrders** function, go into the project in VS Code, expand the **AzureFunctions** folder, select **ProcessOrders.js**, and copy the code, as highlighted in the screen shot below.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar:
 - OPEN EDITORS: app.js, Plans.js, ProcessOrders.js
 - BEST-FOR-YOU-ORGANICS: AzureFunctions, OrdersCosmosTrigger.js, ProcessOrders.js (highlighted with a red box)
 - DATA: bin, build, static, asset-manifest.json, favicon.ico, index.html, manifest.json, service-worker.js
 - MODELS: data, models, node_modules, public, routes
 - SRC: components (header, order, plan), # Plans.css, Plans.js
 - AZURE COSMOS DB: Docker, Images (bestforyouregistry.azurecr.io/best-for-yo...), node:8.9-alpine (16 days ago)
 - CONTAINERS
 - REGISTRIES
- ProcessOrders.js** editor tab:

```
/*
 * ProcessOrders function code
 * Trigger: Azure Storage Queue - orderqueue
 * Input: Azure Cosmos DB - orders collection
 * Input: Azure Cosmos DB - users collection
 * Output: Azure Storage Queue - notificationqueue

 Fires when items are added to the orderqueue. Pulls in the associated order and customer
 to see if a notification should be sent. If so, adds the order and customer ids to the
 notificationqueue.

*/
module.exports = function (context, orderToProcess) {
    // Do order processing stuff...

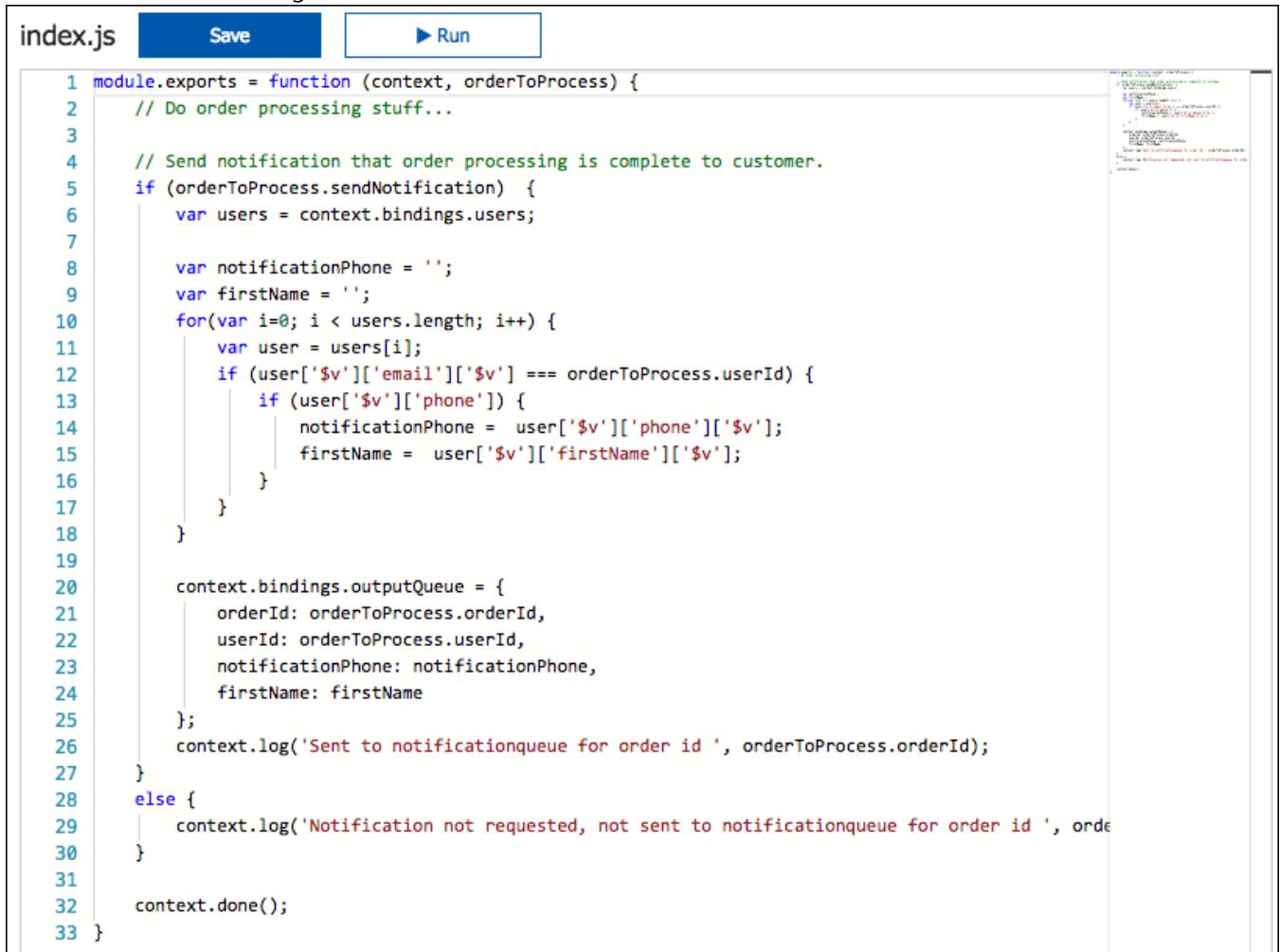
    // Send notification that order processing is complete to customer.
    if (orderToProcess.sendNotification) {
        var users = context.bindings.users;

        var notificationPhone = '';
        var firstName = '';
        for(var i=0; i < users.length; i++) {
            var user = users[i];
            if (user['$v']['email'][ '$v' ] === orderToProcess.userId) {
                if (user['$v']['phone']) {
                    notificationPhone = user['$v']['phone'][ '$v' ];
                    firstName = user['$v']['firstName'][ '$v' ];
                }
            }
        }

        context.bindings.outputQueue = {
            orderId: orderToProcess.orderId,
            userId: orderToProcess.userId,
            notificationPhone: notificationPhone,
            firstName: firstName
        };
        context.log('Sent to notificationqueue for order id ', orderToProcess.orderId);
    } else {
        context.log('Notification not requested, not sent to notificationqueue for order id ', orderToProcess.orderId);
    }

    context.done();
}
```

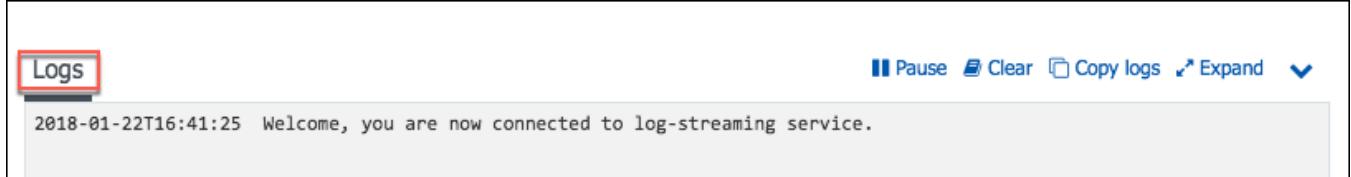
12. Paste the code into the **index.js** block, overwriting all the existing code, and select **Save**. Your index.js file should now look like the following:



The screenshot shows the Azure Functions developer tools interface. On the left, there's a code editor window titled "index.js". The code itself is a function that processes an order, sends notifications to customers, and logs the output queue details. The code is color-coded for syntax highlighting. At the top of the editor, there are three buttons: "Save" (highlighted in blue), "Run" (with a play icon), and "Logs" (disabled). To the right of the editor, there's a preview pane showing a browser-based interface for managing orders.

```
1 module.exports = function (context, orderToProcess) {
2     // Do order processing stuff...
3
4     // Send notification that order processing is complete to customer.
5     if (orderToProcess.sendNotification) {
6         var users = context.bindings.users;
7
8         var notificationPhone = '';
9         var firstName = '';
10        for(var i=0; i < users.length; i++) {
11            var user = users[i];
12            if (user['$v']['email']['$v'] === orderToProcess.userId) {
13                if (user['$v']['phone']) {
14                    notificationPhone = user['$v']['phone']['$v'];
15                    firstName = user['$v']['firstName']['$v'];
16                }
17            }
18        }
19
20        context.bindings.outputQueue = {
21            orderId: orderToProcess.orderId,
22            userId: orderToProcess.userId,
23            notificationPhone: notificationPhone,
24            firstName: firstName
25        };
26        context.log('Sent to notificationqueue for order id ', orderToProcess.orderId);
27    }
28    else {
29        context.log('Notification not requested, not sent to notificationqueue for order id ', orderToProcess.orderId);
30    }
31
32    context.done();
33 }
```

13. Next, select **Logs** below the code block, so you can observe the Function being called during the next steps.



The screenshot shows the Azure Functions log viewer. A red box highlights the "Logs" tab at the top left. Below the tabs, there's a log message: "2018-01-22T16:41:25 Welcome, you are now connected to log-streaming service." At the top right, there are several buttons: "Pause" (disabled), "Clear" (disabled), "Copy logs" (disabled), "Expand" (disabled), and a dropdown menu.

14. To trigger the function, return to the starter application in your browser window, select **Sign In**, and on the **Sign In** screen, select **Register**.

The screenshot shows a web-based application interface. At the top, there's a dark header with a logo on the left, a 'Home' link in the center, and a 'Sign In' button on the right, which is highlighted with a red border. Below the header is a light gray 'Login' section. Inside this section, there are two input fields: 'Email Address:' and 'Password:', each with a small gray box containing three dots on its right side. Below the password field is a 'Login' button. At the bottom of the 'Login' section is a callout box with a thin gray border. Inside the box, the text 'Not registered yet? Register now!' is displayed above a 'Register' button. Both the callout box and the 'Register' button are outlined with a thick red border.

15. Complete the registration form, being sure to include a cell phone number in the Phone field that you can receive text messages on. (If you opt not to enter a phone number, you can still complete the next Exercise, but will not receive the text notifications that your order has been processed.)
- You only need to enter data into the **First name**, **Last name**, **email address**, and **phone** fields. All other fields have been pre-populated to save time.
 - The password has been set to Password.1!! If you choose to enter a different password, note that when you log into the account.
16. After registering, select **Sign In** from the Home page, enter your email address and password (Password.1!!) on the login screen, and select **Login**.

17. Select the **Select this plan** button for any plan on the home page, and on the Order screen, select **Place Order**.

Place Order for the Four Person Plan

[Back to all Plans](#)

Selected Plan:
Four Person Plan

Credit Card Number:
4111111111111111

CVV:
...

Place Order

18. Return to your **ProcessOrders Function** page in the Azure portal and observe the logs.

Logs

Pause Clear Copy logs Expand ▾

```
2018-01-25T04:38:35 Welcome, you are now connected to log-streaming service.
2018-01-25T04:39:35 No new trace in the past 1 min(s).
2018-01-25T04:40:35 No new trace in the past 2 min(s).
2018-01-25T04:41:35 No new trace in the past 3 min(s).
2018-01-25T04:42:35 No new trace in the past 4 min(s).
2018-01-25T04:43:21.906 Function started (Id=288d08d5-a9ac-4fcd-9920-29a905eb7c2e)
2018-01-25T04:43:22.156 Sent to notificationqueue for order id 5a696067a9b5607c920ad82a
2018-01-25T04:43:22.171 Function completed (Success, Id=288d08d5-a9ac-4fcd-9920-29a905eb7c2e,
```

19. The order you placed has been sent to the notificationqueue and is pending the notification being sent to your cell phone via SMS text message.

Exercise 7: Create Logic App for sending SMS notifications

Duration: 30 minutes

In this exercise, you will create Logic App which will trigger when an item is added to the notificationqueue Azure Storage Queue. The Logic App will use a Twilio connection to send an SMS message to the phone number included in the notificationqueue message.

Task 1: Create Free Twilio account

In this task, you will use a free Twilio account to send SMS notifications to customers, informing them that their order has been processed, and is on its way.

1. If you do not have a Twilio account, sign up for one for free at by going to <https://www.twilio.com/try-twilio>.
2. On the **Sign up for free** page:
 - a. Enter your personal info, email address, and a 14+ character password
 - b. Select SMS under **Which product do you plan to use first?**
 - c. Select **Order Notifications** under **What are you building?**
 - d. Select **JavaScript** under **Choose your language**
 - e. Select **Not a Production App** under **Potential monthly interactions**
 - f. Check the box next to **I'm not a robot**

- g. Select **Get Started**.

Sign up for free

First Name Last Name

Company Name (optional)

Email

Choose a password Password, again

WHICH PRODUCT DO YOU PLAN TO USE FIRST?

SMS

WHAT ARE YOU BUILDING?

Order Notifications

CHOOSE YOUR LANGUAGE

JavaScript

POTENTIAL MONTHLY INTERACTIONS (OVER SMS, CHAT, VOICE, & VIDEO)

Not a Production App

I'm not a robot 
reCAPTCHA
Privacy - Terms

Get Started By clicking the button, you agree to our [legal policies](#).

3. Enter your **cell phone number** on the We need to verify you're a human screen, check the box if you do not wish to be contacted at the number you enter, and select **Verify** via SMS.

We need to verify you're a human.

We will send a verification code via **SMS** to number above
Or, we call you instead.

The phone number you provide will be used for authentication when you login to Twilio Console. A Twilio onboarding specialist may also use this number to reach out with free onboarding support. If you do not want to be contacted at this phone number, please check this box.

4. Enter the verification code received via text into the box and select **Submit**.

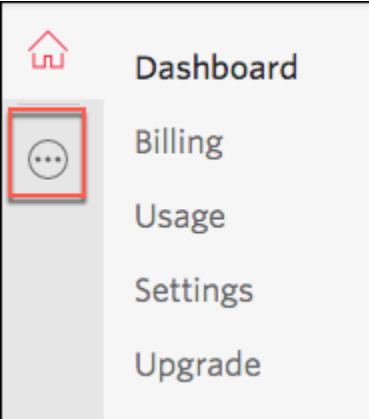
We need to verify you're a human

Please enter the verification code we sent to your phone. If you didn't receive a code, you can try again

0534

Submit

5. From your account dashboard, select the **All Products & Services icon**.



6. Select **#Phone Numbers** under **Super Network**.

SUPER NETWORK

- # Phone Numbers
- SIP Elastic SIP Trunking
- Programmable Wireless
- Interconnect
- Lookup

7. Select **Get Started**.

Phone Numbers Dashboard

Instantly provision local, national, mobile, and toll-free phone numbers in ne...

Get Started Tutorial Docs ↗ Learn More

8. Select **Get your first Twilio phone number**.

Get Started with Phone Numbers

Getting started with Twilio's phone numbers is easy! Search for local, toll-free, or mobile numbers by capability, country, or prefix.

Get your first Twilio phone number

Looking for a short-code? [Apply for a short-code here ↗](#).

9. Select **Choose this Number** (or search for a different number if you want something different).

Your first Twilio Phone Number

(302) 337- [REDACTED]

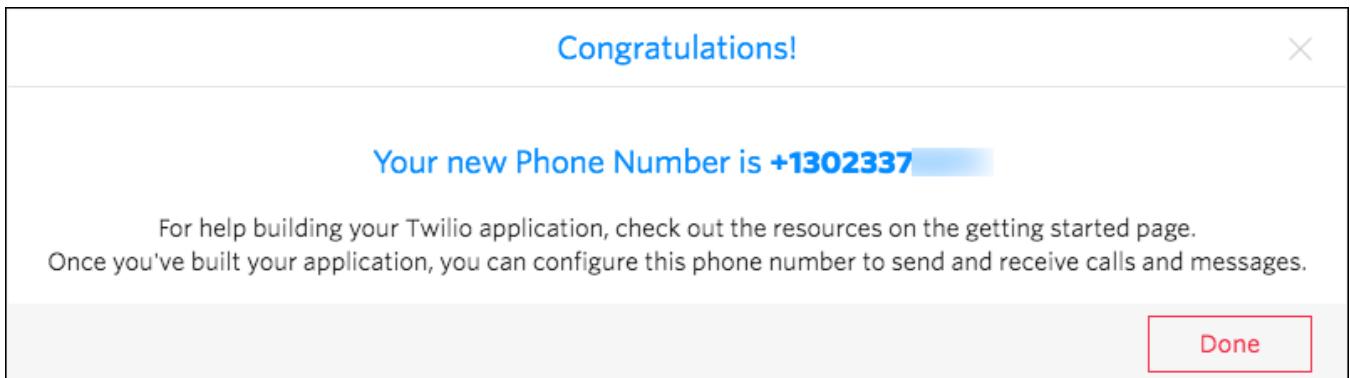
Don't like this one? [Search for a different number](#)

This United States phone number has the following capabilities:

- Voice:** This number can receive incoming calls and make outgoing calls.
- SMS:** This number can send and receive text messages to and from mobile numbers.
- MMS:** This number can send and receive multi media messages to and from mobile numbers.

Cancel **Choose this Number**

10. Select **Done** on the Congratulations dialog.

11. Select **Home**

on your **Account Dashboard**, and leave this page up, as you will be referencing the **Account SID** and **Auth Token** in the next task to configure the Twilio Connector.

The screenshot shows the Twilio Account Dashboard. On the left is a sidebar with links: Dashboard, Billing, Usage, Settings, and Upgrade. The main area shows 'Project Info' with 'ACCOUNT SID' set to 'AC4e3e411b607bc6d' and 'AUTH TOKEN' as a series of dots. It also shows 'USERS' 1. A 'Project Settings' link is at the bottom. A red-bordered 'Done' button is in the bottom right corner of the dashboard area.

Task 2: Create Logic App

In this task, you will create a new Logic App, which will use the Twilio connector to send SMS notifications to users, informing them that their weekly order has processed and shipped.

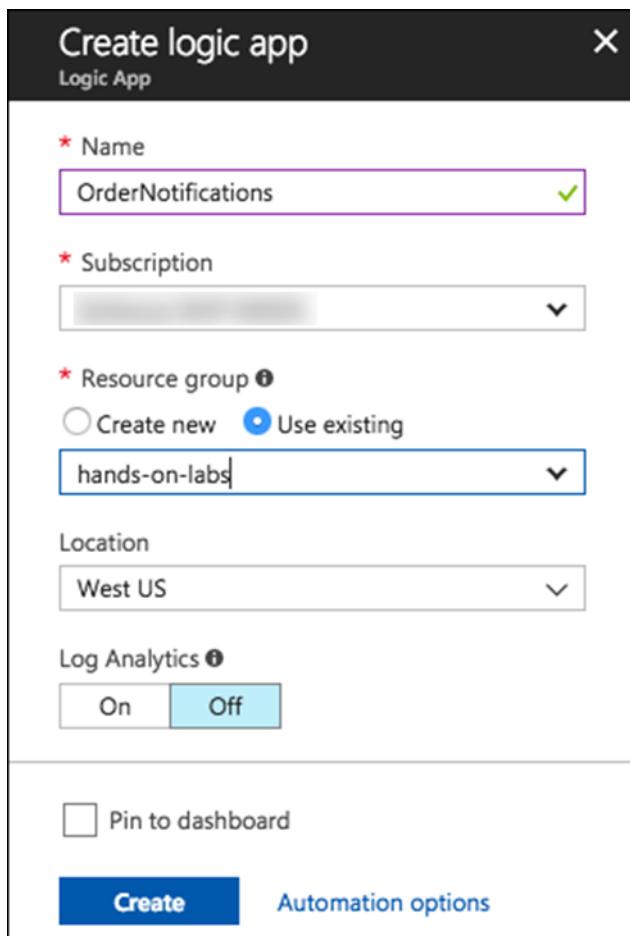
1. In the Azure portal, select **+Create a resource**, select **Web + Mobile**, and select **Logic App**.

The screenshot shows the 'New' blade in the Azure portal. On the left is a sidebar with 'Create a resource', 'All services', 'FAVORITES' (empty), 'Dashboard', 'All resources', and 'Resource groups'. The main area shows 'Recently created' and a list of categories: 'Compute', 'Networking', 'Storage', 'Web + Mobile' (which is highlighted with a red dashed box), and 'Containers'. To the right are icons for 'Mobile App' (Quickstart tutorial), 'Logic App' (Quickstart tutorial, highlighted with a red box), and 'Web App for Containers' (Quickstart tutorial). A red-bordered 'Done' button is in the bottom right corner of the blade.

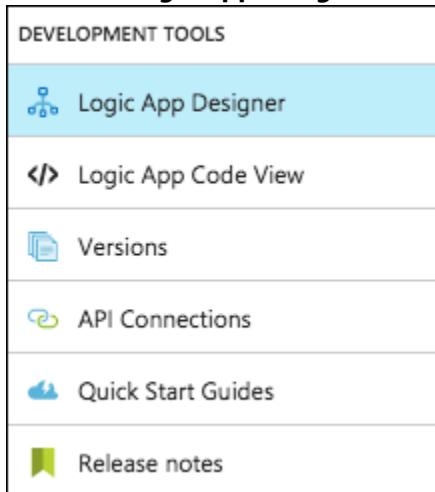
2. In the **Create logic app** blade, enter the following:

- a. **Name:** Enter "OrderNotifications."
- b. **Subscription:** Select the subscription you are using for this hands-on lab.
- c. **Resource group:** Select **Use existing** and choose the **hands-on-lab-SUFFIX** resource group.
- d. **Location:** Select the location you've been using for resources in this hands-on lab.

- e. Select **Create** to provision the new Logic App.



3. Navigate to your newly created Logic App in the Azure portal.
4. Select the **Logic App Designer** under **Development Tools** on the left-hand menu.



5. In the Logic App Designer, select **Blank Logic App** under **Templates**.

The screenshot shows the 'Templates' section of the Logic App Designer. At the top, there is a search bar with placeholder text 'Choose a template below to create your Logic App.' Below it are two dropdown menus: 'Category : All' and 'Sort by : Popularity'. A red box highlights the first template card, which is labeled 'Blank Logic App' and features a large black plus sign icon.

Blank Logic App +	Correlated in-order delivery using service bus sessions	Delete old Azure blobs
HTTP Request-Response	Peek-lock receive a Service Bus message and complete it	Receive an X12 EDI document over AS2 and transform it to

6. Select **Azure Queues** under **Connectors**.

The screenshot shows the 'Connectors' section of the Logic App Designer. At the top, there is a search bar with placeholder text 'Search all connectors and triggers'. Below it is a grid of connector icons. One icon, 'Azure Queues', is highlighted with a red box.

Request	Schedule	Service Bus	Twitter	Office 365 Outlook	SharePoint	FTP
Dynamics 365	SFTP	Salesforce	RSS	OneDrive	Azure Event Grid	Azure Queues

7. Select **Azure Queues** – When there are messages in a queue.

Triggers (2) Actions (5)

Azure Queues - When there are messages in a queue

Azure Queues - When a specified number of messages are in a given queue

TELL US WHAT YOU NEED

Help us decide which connectors and triggers to add next with [UserVoice](#)

8. On the When there are messages in a queue dialog, enter **bestforyouorders** for the, **Connection Name** select the bestforyouorders **Storage account** from the list and select **Create**.

When there are messages in a queue ...

* Connection Name bestforyouorders

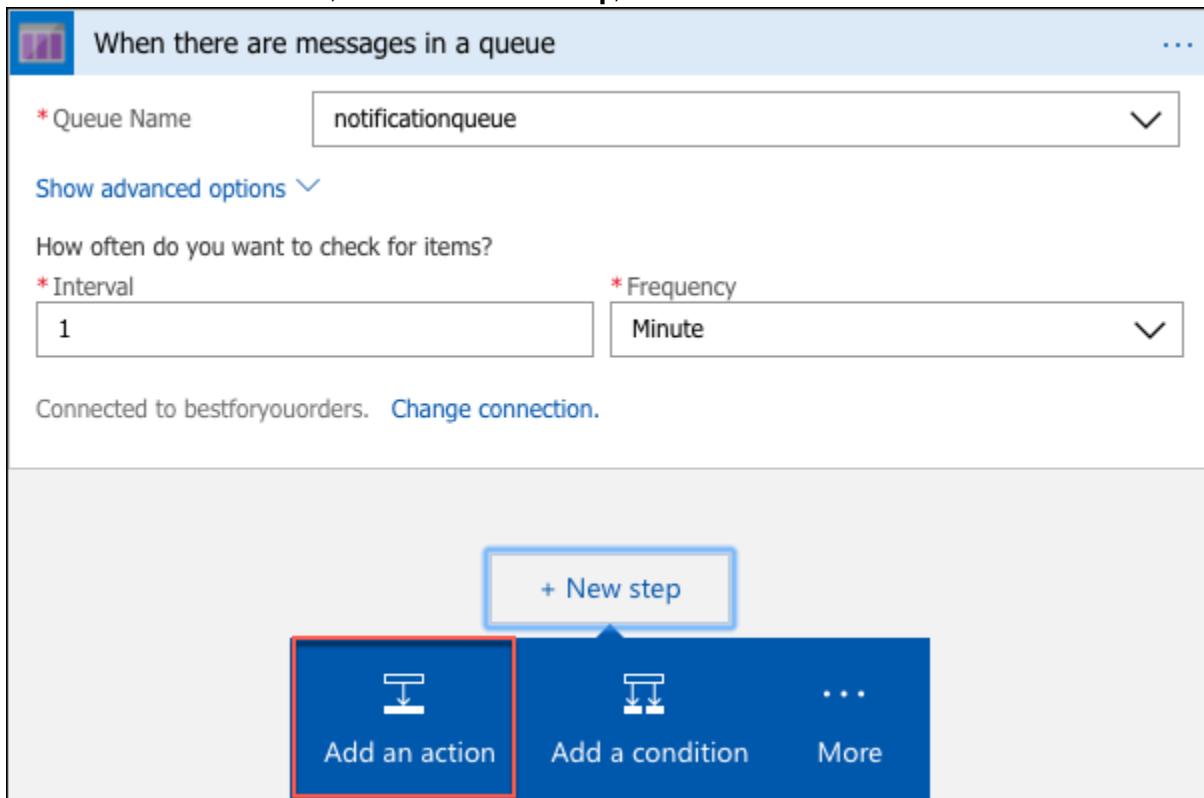
* Storage Account

bestforyouorders hands-on-labs westus

Create

Manually enter connection information

9. In the next When there are messages in a queue dialog, select **notificationqueue** from the **Queue Name** list, and set the interval to **1 minute**, then select **+New step**, and **Add an action**.



10. In the **Choose an action box**, enter "Parse," and select **Data Operations – Parse JSON** from the list.

When there are messages in a queue

Choose an action

Parse

Connectors

See more

Data Operations Docparser Parserr

Triggers (2) Actions (4)

See more

Data Operations - Parse JSON

Docparser - Upload a document to Docparser Preview

Docparser - Fetch document from a URL Preview

Parserr - List inboxes Preview

TELL US WHAT YOU NEED

Help us decide which connectors and triggers to add next with UserVoice

Cancel

11. Select the **Content** box, select **Add dynamic content +**, then select **Message Text** from the input parameters list that appears.

The screenshot shows the 'Parse JSON' dialog. On the left, under 'Content', there is a 'Message Text' input field. To its right is a button labeled 'Add dynamic content +'. Below this is the 'Schema' section, which displays a JSON schema:

```
{  
  "type": "object",  
  "properties": {  
    "orderId": {  
      "type": "string"  
    },  
    "userId": {  
      "type": "string"  
    }  
}
```

At the bottom of the schema section is a link 'Use sample payload to generate schema'. On the right side of the dialog, there is a sidebar titled 'Dynamic content' with a list of options:

Dynamic content	Expression
Search dynamic content	
When there are messages in a queue	
Body	
Expiration Time	The time the message will expire from the queue.
Insertion Time	The time the message was inserted into the queue
Message ID	The unique identifier of the message.
Message Text	The text of the message.

12. Next, select **Use sample payload to generate schema** below the **Schema** box.

The screenshot shows the 'Parse JSON' dialog again. The 'Content' section contains a 'Message Text' input field. The 'Schema' section is currently empty. At the bottom of the dialog, there is a link 'Use sample payload to generate schema'.

13. In the dialog that appears, paste the following JSON into the sample JSON payload dialog that appears, then select **Done**.

```
{"orderId": "5a6748c5d0d3199cfa076ed3", "userId": "demouser@bfyo.com", "notificationPhone": "317551212", "firstName": "Demo"}
```



14. You will now see the Schema for messages coming from the notification queue in the Schema box. Select **+New step** and select **Add an action**.

A screenshot of the Microsoft Logic App builder interface. A "Parse JSON" step is selected. The "Content" field is set to "Message Text". The "Schema" field displays the following JSON schema:

```
{
  "type": "object",
  "properties": {
    "orderId": {
      "type": "string"
    },
    "userId": {
      "type": "string"
    }
  }
}
```

Below the schema, there is a link "Use sample payload to generate schema". At the bottom of the step's configuration panel, there are three buttons: "+ New step" (highlighted with a red box), "Add an action" (also highlighted with a red box), "Add a condition", and "More".

15. In the **Choose an action** box, enter "Twilio," and select **Twilio – Send Text Message (SMS)** under Actions.

Choose an action

twilio

Connectors

Twilio

Triggers (0) Actions (3)

Twilio - Get Message

Twilio - List Messages

Twilio - Send Text Message (SMS)

TELL US WHAT YOU NEED

Help us decide which connectors and triggers to add next with [UserVoice](#)

Cancel

16. In the **Twilio – Send Text Message (SMS)** dialog, enter the following (You will need the details from Project Info block on the dashboard of your Twilio account for this step):

- Connection Name:** Twilio
- Twilio Account Id:** Enter your Twilio account SID.
- Twilio Access Token:** Enter your Twilio auth token.
- Select **Create**.

Twilio - Send Text Message (SMS)

* Connection Name: Twilio

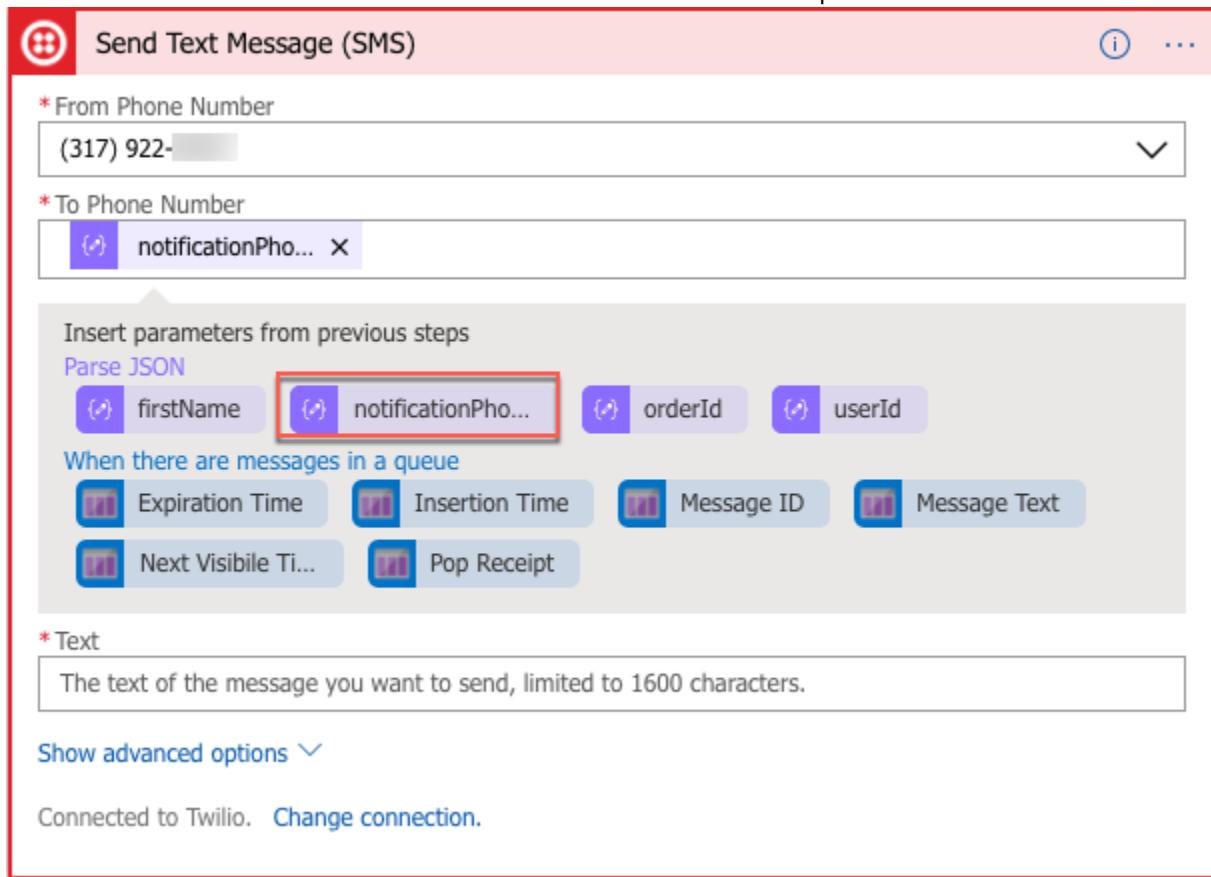
* Twilio Account Id:

* Twilio Access Token:

Create

17. On the next **Send Text Message (SMS)** dialog, enter the following:

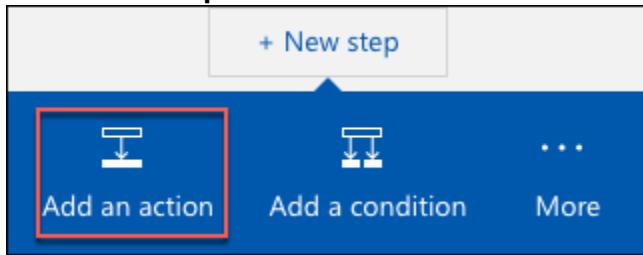
- From Phone Number:** Select your Twilio phone number from the drop down.
- To Phone Number:** Select **notificationPhone** from the **Parse JSON** parameters.



- c. **Text:** Enter a message, such as "Hello [firstName], your Best for You Organics weekly order has shipped!" For [firstName], select the **firstName** parameter from the **Parse JSON** items.

The screenshot shows the configuration for a 'Send Text Message (SMS)' step. The 'From Phone Number' field contains '(317) 922-'. The 'To Phone Number' field contains a placeholder 'notificationPho...'. The 'Text' field contains the message 'Hello [firstName] , your Best for You Organics weekly order has shipped!'. Below the form, there is a section titled 'Insert parameters from previous steps' containing a 'Parse JSON' button and several parameter options: 'firstName' (highlighted with a red box), 'notificationPho...', 'orderId', and 'userId'. Another section titled 'When there are messages in a queue' lists 'Expiration Time', 'Insertion Time', 'Message ID', 'Message Text', 'Next Visible Ti...', and 'Pop Receipt'. At the bottom, there is a 'Show advanced options' link and a note that the step is connected to Twilio with a 'Change connection' link.

18. Select **+New step** and **Add an action**.



19. In the **Choose an action** dialog, enter “queue” in to the search box, and select **Azure Queues – Delete message**.

Choose an action

queue

Connectors

Service Bus Azure Queues Dynamics 365 MQ Salesforce Visual Stu... Team

See more

Triggers (11) Actions (34)

See more

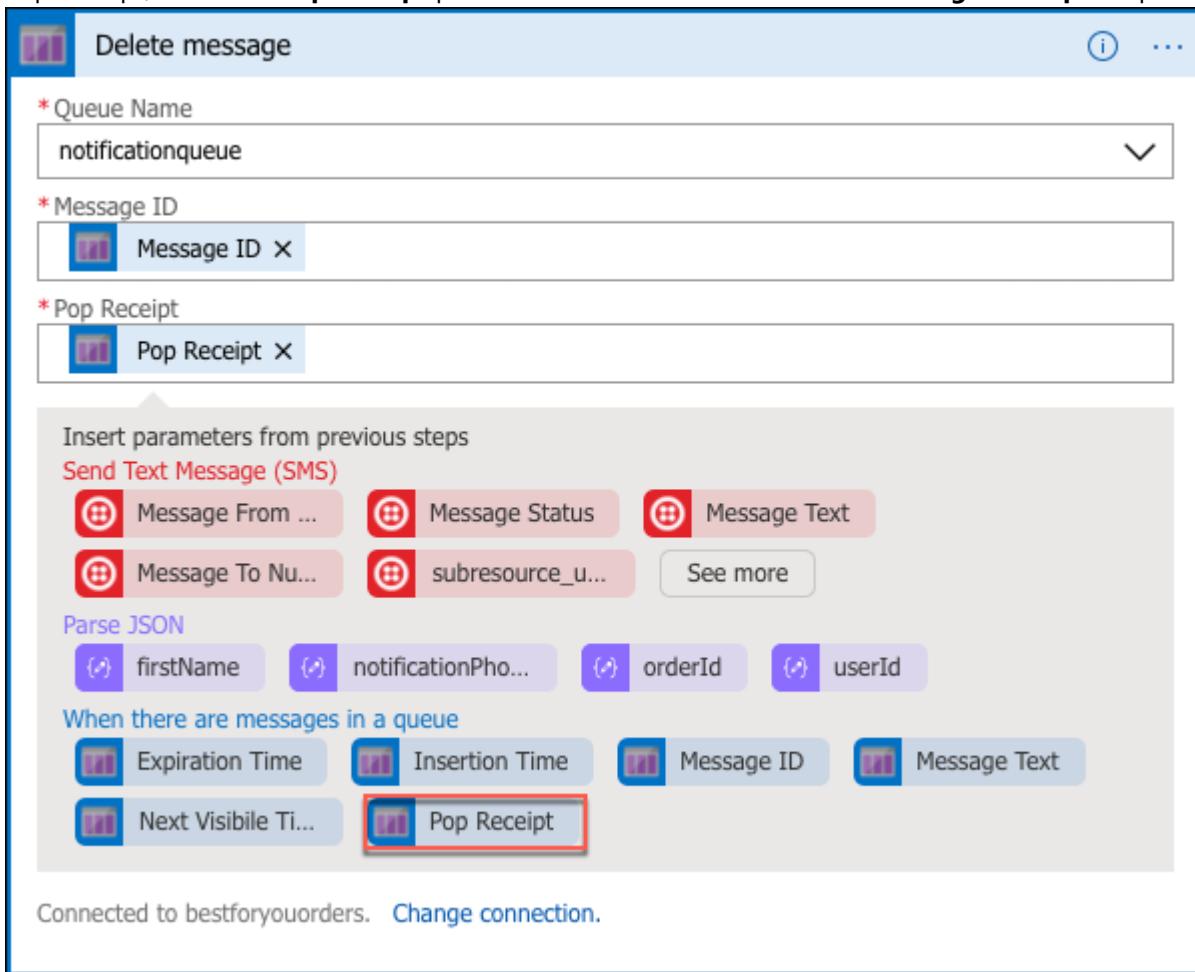
- Azure Queues - Get messages
- Azure Queues - Put a message on a queue
- Azure Queues - Create a new queue
- Azure Queues - Delete message**
- Azure Queues - List queues

20. Select **notificationqueue** for the Queue Name.

21. For Message ID, select the **Message ID** parameter from the **When there are messages in the queue** parameter list.

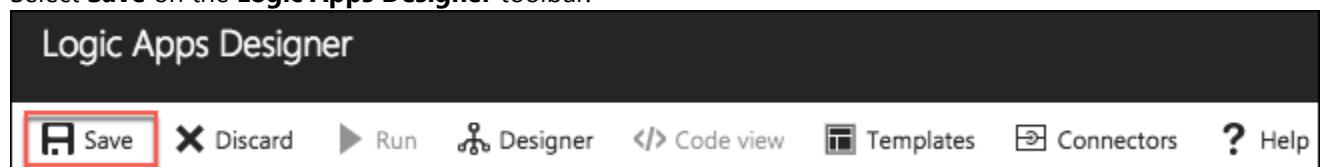
The screenshot shows the 'Delete message' step in Microsoft Flow. At the top, it asks for the 'Queue Name' (set to 'notificationqueue') and 'Message ID'. Below this, there's a section titled 'Insert parameters from previous steps' containing 'Send Text Message (SMS)' and 'Parse JSON' options. Under 'When there are messages in a queue', the 'Message ID' parameter is highlighted with a red box. A note below says 'A valid pop receipt value returned from an earlier call to the Get Messages.' At the bottom, it says 'Connected to bestforyouorders. Change connection.'

22. For Pop Receipt, select the **Pop Receipt** parameter from the **When there are messages in a queue** parameter



list.

23. Select **Save** on the **Logic Apps Designer** toolbar.



24. The Logic App will begin running immediately, so if you entered your cell phone number when you registered your account in the Best for You Organics starter app, and placed an order, you should receive a text message on your phone within a minute or two of selecting Save.

After the hands-on lab

Duration: 10 minutes

In this exercise, you will deprovision all Azure resources that were created in support of this hands-on lab.

Task 1: Delete Azure resource groups

1. In the Azure portal, select **Resource groups** from the left-hand menu, and locate and delete the following resource groups.
 - a. hands-on-lab-SUFFIX
 - b. jenkins-SUFFIX

Task 2: Delete WebHooks and Service Integrations

1. In your GitHub account
 - a. Delete the Jenkins service integration
 - b. Delete the Personal Access Token you created for integration with VS Code.
2. In your VSTS account
 - a. Delete the Personal Access Token you created for integration with Jenkins.
 - b. Delete the Jenkins service endpoint you added
 - c. Delete the BestForYouOrganics CD release

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

Appendix A: Lab VM setup

Appendix A provides steps for manually provisioning and setting up the Lab VM used as a development machine for this lab.

Task 1: Create VM config script

In this task, you will create a script file that will be used as a custom extension for configuring your Linux virtual machine in Azure. This script contains commands to install all the required software and configure a desktop on the Linux VM. These commands could also be run from an SSH shell manually.

1. Open a web browser, and navigate to <https://raw.githubusercontent.com/ZoinerTejada/mcw-oss-paas-devops/master/LabVM/labvmconfig.sh>.
2. Copy the contents displayed in the browser into a text editor, such as Notepad, and save the file as **labvmconfig.sh**. Note the location you saved the file, as you will be referencing it in the next task.

Task 2: Create a Linux virtual machine

In this task, you will provision a Linux virtual machine (VM) running Ubuntu Server 16.04 LTS.

1. In the [Azure Portal](#), select **+Create a resource**, then enter “ubuntu” into the search bar, and select **Ubuntu Server 16.04 LTS** from the results.

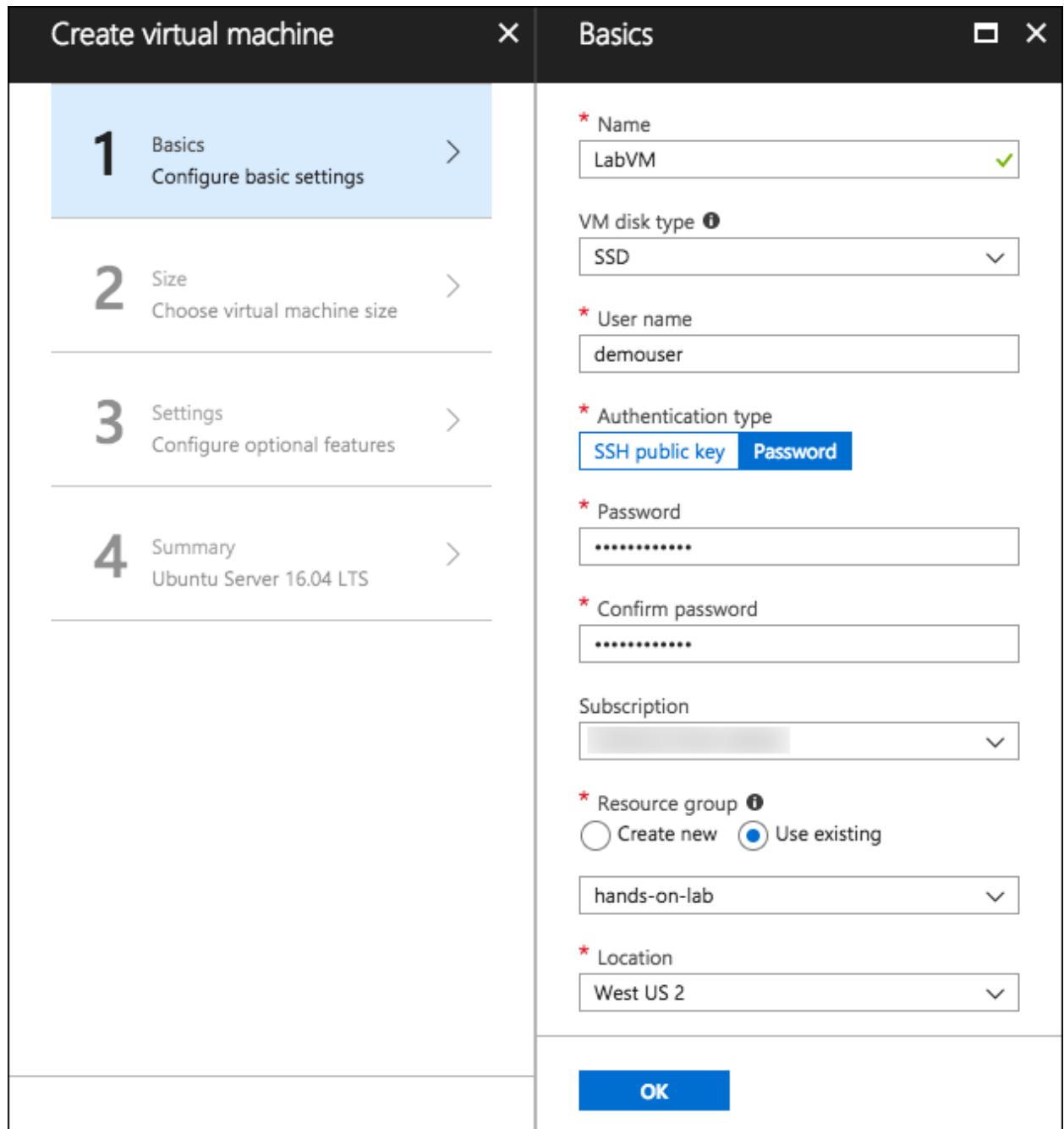
NAME	PUBLISHER	CATEGORY
Ubuntu Server 16.04 LTS	Canonical	Compute
Ubuntu Server 17.10	Canonical	Compute

2. On the **Ubuntu Server 16.04 LTS** blade, select **Create**.

3. Set the following configuration on the **Basics** tab.

- Name:** Enter LabVM.
- VM disk type:** Select **SSD**.
- User name:** Enter demouser
- Authentication Type:** Select **Password**
- Password:** Enter Password.1!!
- Subscription:** Select the subscription you are using for this hands-on lab
- Resource Group:** Select **Create new**, and enter “hands-on-lab-(SUFFIX)” as the name of the new resource group

- h. **Location:** Select either **East US**, **West US 2**, **West Europe**, or **Southeast Asia**, as these are currently the only regions which offer Dv3 and Ev3 VMs. Remember this location for other resources in this hands-on lab



4. Select **OK** to move to the next step.
5. On the **Choose a size** blade, select **View all**. This machine will be doing nested virtualization, so it needs to be in either the Dv3 or Ev3 series, so selecting **D2S_V3 Standard** is a good baseline option. If that size is not available in

the region you selected, go back to the **Basics** blade, and try one of the other regions listed above.

The screenshot shows the 'Create virtual machine' wizard at step 2: 'Choose a size'. The left sidebar lists steps 1 through 4. Step 1 is 'Basics' (Done). Step 2 is 'Size' (Choose virtual machine size). Step 3 is 'Settings' (Configure optional features). Step 4 is 'Summary' (Ubuntu Server 16.04 LTS). The main area is titled 'Choose a size' with the sub-instruction 'Browse the available sizes and their features'. It includes a note about prices being estimates in local currency. Below this are filters for 'Supported disk type' (SSD), 'Minimum vCPUs' (1), and 'Minimum memory (GiB)' (0). A 'View all' button is highlighted with a red box. The table lists several VM sizes:

VM Size	Processor	Memory (GB)	Storage (Data disks)	IOPS (Max)	Local SSD (GB)	Premium Disk Support	Load Balancing	Price (USD/Month)	Notes
D2S_V3 Standard	2 vCPUs	8 GB	4 Data disks	4000 Max IOPS	16 GB Local SSD	Premium disk support	Load balancing	71.42	USD/MONTH (ESTIMATED)
E2S_V3 Standard	2 vCPUs	16 GB	4 Data disks	4000 Max IOPS	32 GB Local SSD	Premium disk support	Load balancing	98.95	USD/MONTH (ESTIMATED)
E4-2S_V3 Standard	2 vCPUs	32 GB	8 Data disks	8000 Max IOPS	64 GB Local SSD	Premium disk support	Load balancing	197.90	USD/MONTH (ESTIMATED)
E8-2S_V3 Standard	2 vCPUs	64 GB	16 Data disks	16000 Max IOPS	128 GB Local SSD	Premium disk support	Load balancing	395.80	USD/MONTH (ESTIMATED)
DS1_V2 Standard	1 vCPU	2 GB	2 Data disks	2000 Max IOPS	4 GB Local SSD	Premium disk support	Load balancing	24.90	USD/MONTH (ESTIMATED)
DS2_V2 Standard	2 vCPUs	4 GB	4 Data disks	4000 Max IOPS	8 GB Local SSD	Premium disk support	Load balancing	49.80	USD/MONTH (ESTIMATED)

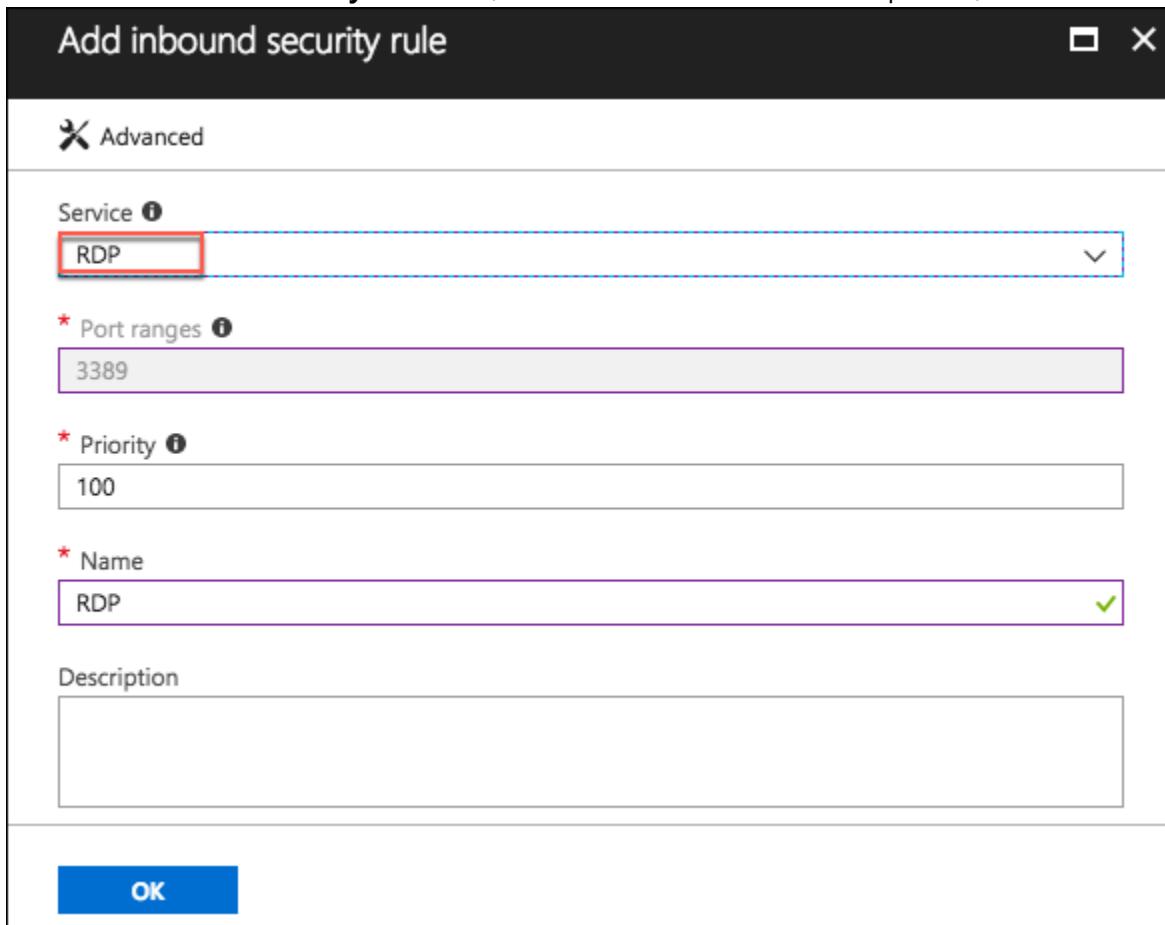
A 'Select' button is at the bottom of the table.

6. Click **Select** to move on to the **Settings** blade.

7. On the **Settings** blade, select **Network security group (firewall)**, then select **+Add an inbound rule** in the Create network security group blade.

The screenshot illustrates the process of creating a virtual machine and configuring its network security settings. The 'Create virtual machine' wizard has completed steps 1, 2, and 3. In step 3 'Settings', the 'Network' section is selected, and the 'Network security group (firewall)' option is highlighted with a red box. This leads to the 'Settings' blade where the same 'Network security group (firewall)' option is also highlighted. The 'Choose network security group' blade shows a list of available network security groups. Finally, the 'Create network security group' blade is open, showing an inbound rule for SSH (TCP/22) being added, with the '+ Add an inbound rule' button highlighted.

8. On the **Add inbound security rule** blade, select **RDP** from the **Service** drop down, then select **OK**.



9. Select **OK** on the Create network security group blade.
10. Next, select **Extensions** on the **Settings** blade, and select **Add extension**.

The screenshot shows three blades overlaid. The leftmost blade is 'Create virtual machine' with steps 1-4. Step 3 is active, showing 'Configure optional features'. The middle blade is 'Settings' showing network and public IP configurations. The rightmost blade is 'Extensions' with the heading 'Add new features, like configuration management or antivirus protection, to your virtual machine using extensions.' It shows 'No extensions' and a blue 'Add extension' button, which is highlighted with a red box.

11. On the **New resource** blade, select **Custom Script for Linux**, then select **Create** on the **Custom Script for Linux** blade. By using a custom script, you can install software and configure the VM as part of the provisioning process.

The screenshot shows the 'New resource' blade with the following details:

- Publisher:** Microsoft Corp.
- Extension Name:** Custom Script For Linux
- Description:** CustomScript Extension is a tool to execute your VM customization tasks. This Extension is added to a Virtual Machine, it can download customer's storage or public storage, and execute the scripts on the VM. CustomScript can be automated using the Azure PowerShell cmdlets and Azure Cross-Platform Interface (xPlat CLI).
- Legal Terms:** By clicking the Create button, I acknowledge that I am getting this software and that the [legal terms](#) of Microsoft Corp. apply to it. Microsoft does not sell this software. Also see the [privacy statement](#) from Microsoft Corp..
- Share:** Social sharing icons for Twitter, Facebook, LinkedIn, YouTube, Google+, and Email.
- Publisher:** Microsoft Corp.
- Useful Links:** Documentation
- Create:** A large blue button.

12. On the **Install extension** blade:

- Script files:** Select the **labvmconfig.sh** file you saved in the previous task
- Command:** Enter "bash labvmconfig.sh"
- Select **OK**

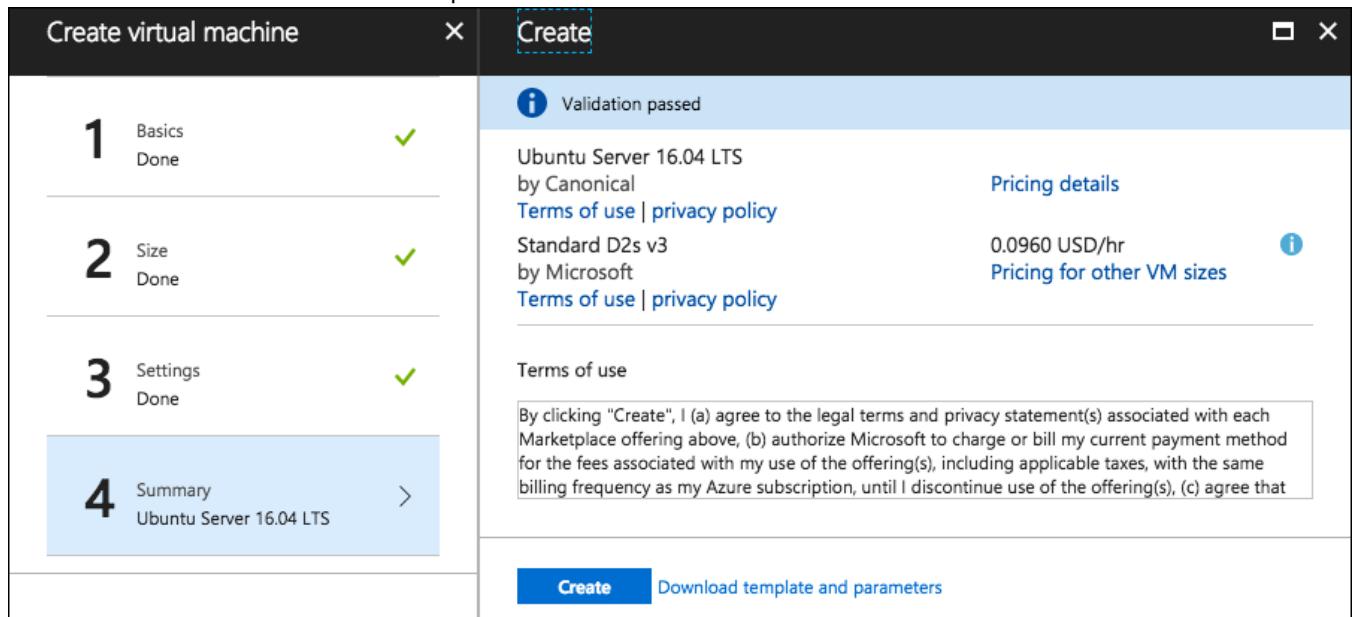
The screenshot shows the 'Install extension' blade with the following configuration:

- Script files:** labvmconfig.sh
- Command:** bash labvmconfig.sh

A large blue **OK** button is at the bottom.

13. Select **OK** on the Extension blade.
14. Select **OK** on the Settings blade.

15. Select **Create** on the **Create** blade to provision the virtual machine.



16. It may take 10+ minutes for the virtual machine to complete provisioning. You can move on to the next task while you wait for this to complete.