Microsoft Cloud Workshop

Containers and DevOps

Hands-on lab step-by-step

January 2018

Contents

Containers and DevOps hands-on lab step-by-step 1

Abstract and learning objectives 1

Overview 1

Requirements 1

Before the hands-on lab 3

- Task 1: Resource Group 3
- Task 2: Create a Windows 10 Development VM 4
- Task 3: Install WSL (Bash on Ubuntu on Windows) 6
- Task 4: Create an SSH key 6
- Task 5: Create a build agent VM 7
- Task 6: Connect securely to the build agent 11
- Task 7: Complete the build agent setup 12
- Task 8: Create an Azure Container Registry 14
- Task 9: Create a Service Principal 16
- Task 10: Create an Azure Container Service cluster 17
- Task 11: Install Azure CLI 20
- Task 12: Install Kubernetes CLI 21

Exercise 1: Environment setup 22

- Task 1: Download the FabMedical starter files 22

Exercise 2: Create and run a Docker application 22

- Task 1: Test the application 23
- Task 2: Enable browsing to the web application 23
- Task 3: Create a Dockerfile 26
- Task 4: Create Docker images 28
- Task 5: Run a containerized application 29
- Task 6: Setup environment variables 30
- Task 7: Push images to Azure Container Registry 31

Exercise 3: Deploy the solution to Azure Container Service 35

- Task 1: Tunnel into the Azure Container Service cluster 35
- Task 2: Deploy a service using the Kubernetes management dashboard 37
- Task 3: Deploy a service using Kubernetes REST API 42
- Task 4: Explore service instance logs and resolve an issue 47
- Task 5: Test the application in a browser 53

Exercise 4: Scale the application and test HA 56

- Task 1: Increase service instances from the Kubernetes dashboard 56
- Task 2: Increase service instances beyond available resources 58
- Task 3: Restart containers and test HA 59

Exercise 5: Setup load balancing and service discovery 65

- Task 1: Create a public load balancer for a service 65
- Task 2: Scale a service without port constraints 67
- Task 3: Update a service to support dynamic service discovery without a load balancer 69
- Task 4: Update an external service to support dynamic discovery with a load balancer 70
- Task 5: Adjust CPU constraints to improve scale 72
- Task 6: Perform a rolling update 73

After the hands-on lab.. 75

Containers and DevOps hands-on lab step-by-step

Abstract and learning objectives

Build a PoC to deliver a multi-tenant web app hosting solution leveraging Azure Container Service with Kubernetes, Docker containers, and Linux nodes.

Attendees will be better able to deploy Docker-based applications and scale them with Azure Container Service and Kubernetes orchestration. In addition,

- Create & run a Docker Application
- Deploy to the Azure Container Service with Kubernetes orchestration
- Implement load balancing and service discovery
- Scale the application and test availability

Overview

Fabrikam Medical Conferences (FabMedical) provides conference website services tailored to the medical community. They are refactoring their application code, based on node.js, so that it can run as a Docker application, and want to implement a POC that will help them get familiar with the development process, lifecycle of deployment, and critical aspects of the hosting environment. They will be deploying their applications to Azure Container Service based on Kubernetes and want to learn how to deploy containers in a dynamically load-balanced manner, discover containers, and scale them on demand.

In this hands-on lab, you will assist with completing this POC with a subset of the application code base. You will create a build agent based on Linux, and an Azure Container Service cluster for running deployed applications. You will be helping them to complete the Docker setup for their application, test locally, push to an image repository, deploy to the cluster, and test load-balancing and scale.

IMPORTANT: Most Azure resources require unique names. Throughout these steps you will see the word "SUFFIX" as part of resource names. You should replace this with your Microsoft email prefix to ensure the resource is uniquely named.

Requirements

1. Microsoft Azure subscription must be pay-as-you-go or MSDN.
    a. Trial subscriptions will *not* work.
    b. You must have rights to create a service principal as discussed in Task 9: Create a Service Principal — and this typically requires a subscription owner to log in. You may have to ask another subscription owner to login to the portal and execute that step ahead of time if you do not have the rights.
    c. You must have enough cores available in your subscription to create the build agent and Azure Container Service cluster in Task 5: Create a build agent VM and Task 10: Create an Azure Container Service cluster. You'll need eight cores if following the exact instructions in the lab, more if you choose additional

agents or larger VM sizes. If you execute the steps required before the lab, you will be able to see if you need to request more cores in your sub.

2. Local machine or a virtual machine configured with:
    a. A browser, preferably Chrome for consistency with the lab implementation tests
    b. Command prompt
        i. On Windows, you will be using Bash on Ubuntu on Windows, hereon referred to as WSL.
        ii. On Mac, all instructions should be executed using bash in Terminal.
3. You will be asked to install other tools throughout the exercises.

VERY IMPORTANT: You should be typing all of the commands as they appear in the guide, except where explicitly stated in this document. Do not try to copy and paste from Word to your command windows or other documents where you are instructed to enter information shown in this document. There can be issues with Copy and Paste from Word that result in errors, execution of instructions, or creation of file content.

Before the hands-on lab

Duration: 1 hour

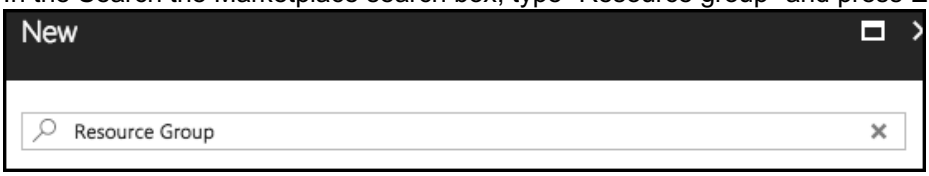
You should follow all of the steps provided in this section *before* taking part in the hands-on lab ahead of time as some of these steps take time.

Task 1: Resource Group

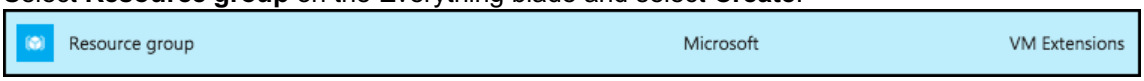
You will create an Azure Resource Group to hold most of the resources that you create in this hands-on lab. This approach will make it easier to clean up later. You will be instructed to create new resources in this Resource Group during the remaining exercises.

1. In your browser, navigate to the **Azure Portal** (https://portal.azure.com).
2. Select **+ New** in the navigation bar at the left.

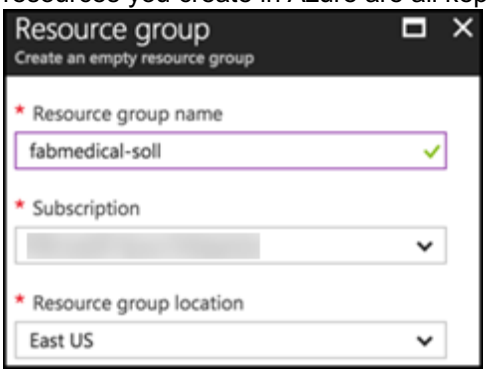
3. In the Search the Marketplace search box, type "Resource group" and press Enter.



4. Select **Resource group** on the Everything blade and select **Create**.

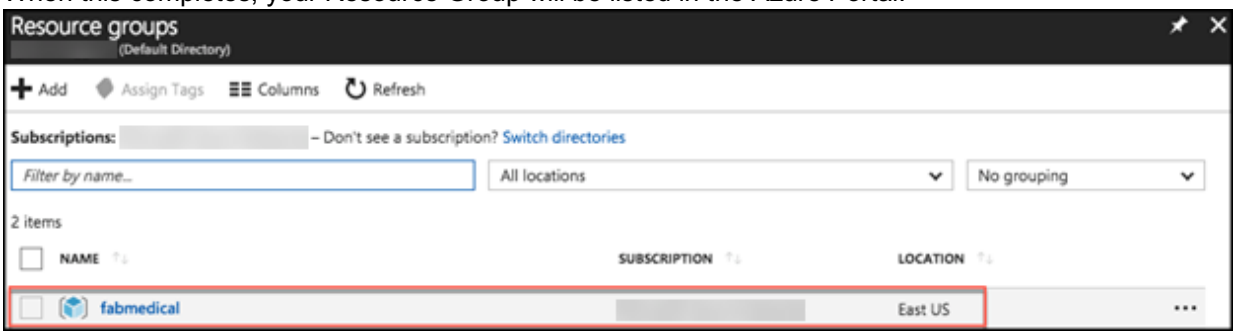


5. On the new Resource group blade, set the following:
   a. Resource group name: Enter something like “fabmedical-SUFFIX”, as shown in the following screenshot.
   b. Subscription: Select the subscription you will use for all the steps during the lab.
   c. Resource group location: Choose a region where all Azure Container Registry SKUs are available, which is currently East US, West Central US, or West Europe, and remember this for future steps so that the resources you create in Azure are all kept within the same region.



   d. Select **Create**.

6. When this completes, your Resource Group will be listed in the Azure Portal.



Task 2: Create a Windows 10 Development VM

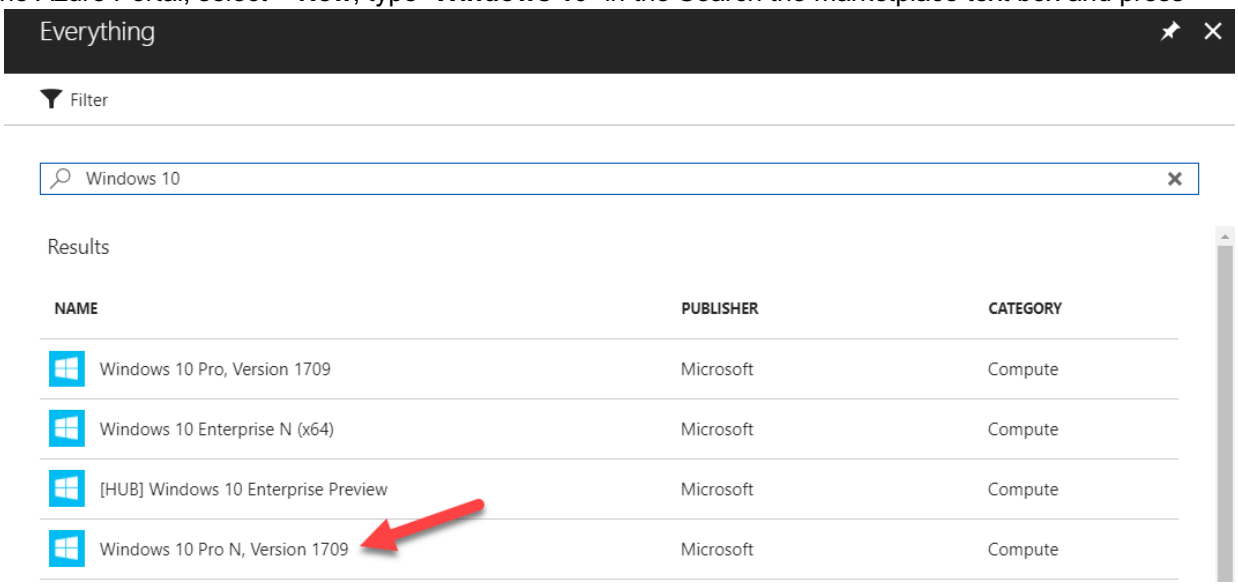
You will follow these steps to create a development VM (machine) for the following reasons:

- If your operating system is earlier than Windows 10 Anniversary Update, you will need it to work with WSL as instructed in the lab.
- If you are not sure if you set up WSL correctly, given there are a few ways to do this, it may be easier to create the development machine for a predictable experience.

NOTE: Setting up the development machine is optional for Mac OS since you will use Terminal for commands. Setting up the development machine is also optional if you are certain you have a working installation of WSL on your current Windows 10 VM.

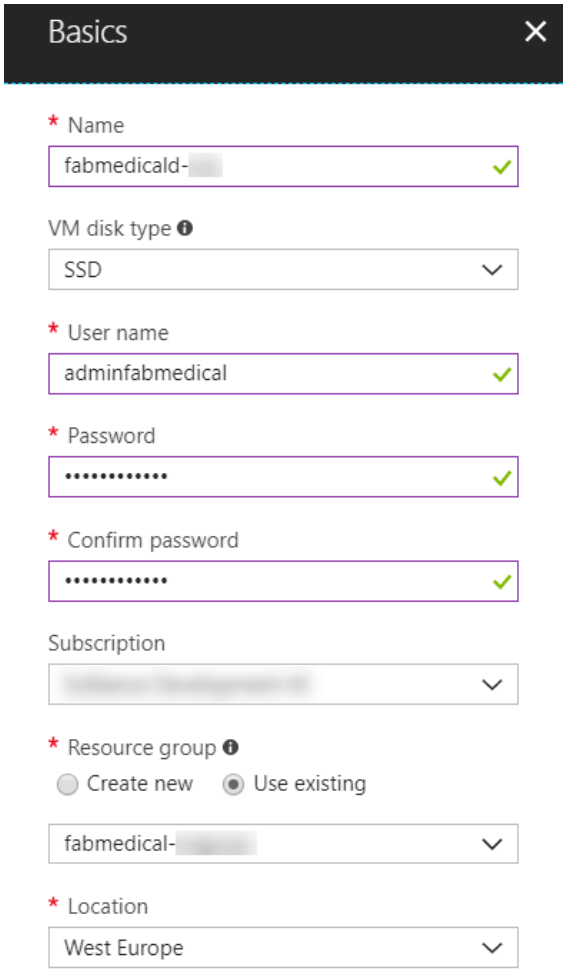
In this section, you will create a Windows 10 VM to act as your development machine. You will install the required components to complete the lab using this machine. You will use this machine instead of your local machine to carry out the instructions during the lab.

1. From the Azure Portal, select **+ New**, type "**Windows 10**" in the Search the marketplace text box and press **Enter**.

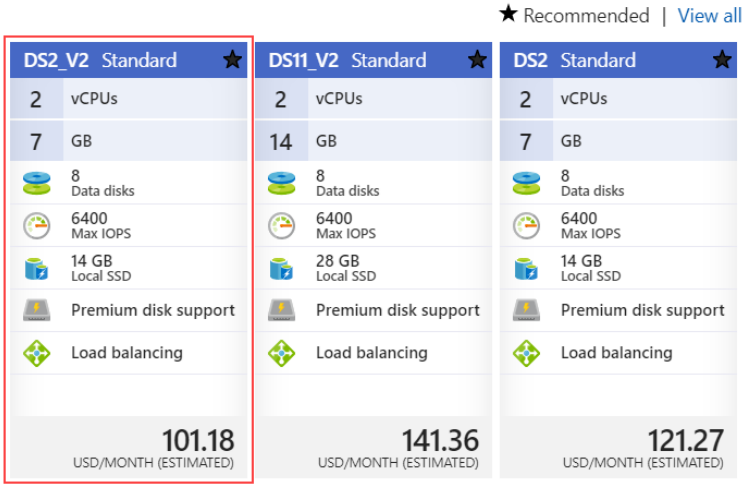


2. Select **Windows 10 Pro N, Version 1709** and select **Create**.
3. On the Basics blade of the Virtual Machine setup, set the following:
    a. **Name**: Provide a unique name, such as "fabmedicald-SUFFIX" as shown in the following screenshot.
    b. **VM disk type**: Leave as SSD.
    c. **User name**: Provide a user name, such as "adminfabmedical".
    d. **Password**: Provide a password, such as "Password$123".
    e. **Confirm password**: Confirm the previously entered password.
    f. **Subscription**: Choose the same subscription you are using for all your work.
    g. **Resource group**: Choose Use existing and select the resource group you created previously.

h. **Location**: Choose the same region that you did before.
i. Select **OK** to complete the Basics blade.

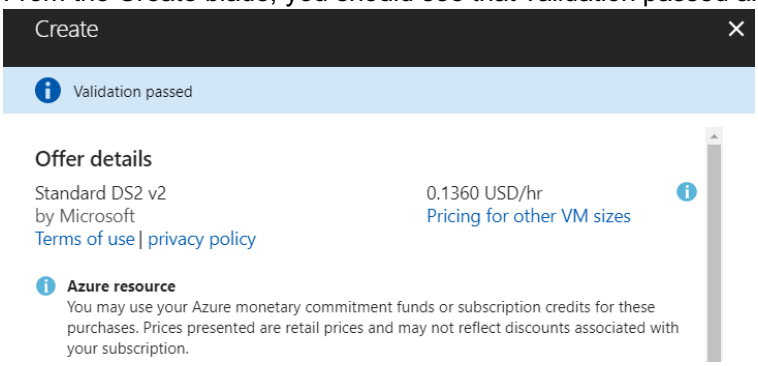


4. From the Size blade, choose **D2S_V2 Standard** and **Select**.



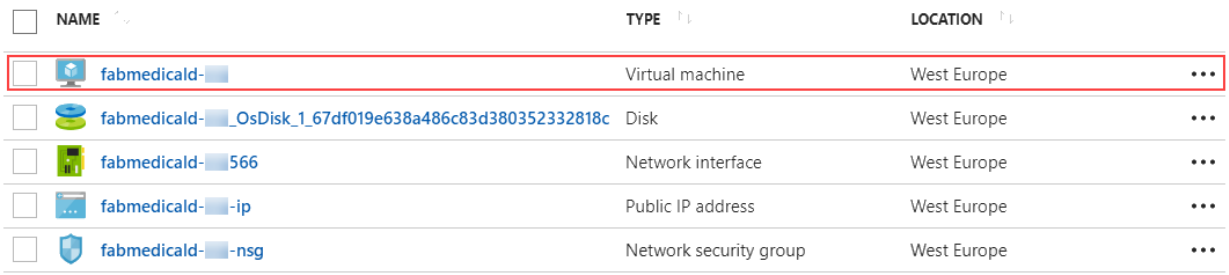
5. From the Settings blade, accept the default values for all settings and select **OK**.

6. From the Create blade, you should see that validation passed and select **Create**.

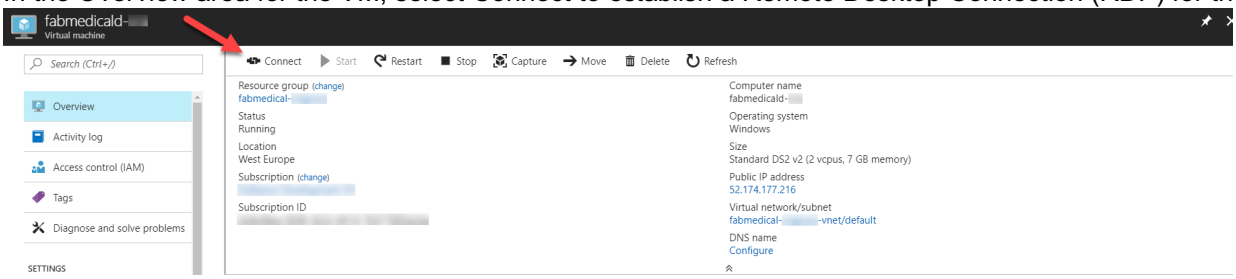


7. The VM will begin deployment to your Azure subscription.

8. Once provisioned, you will see the VM in your list of resources belonging to the resource group you created previously, and select the new VM.



9. In the Overview area for the VM, select Connect to establish a Remote Desktop Connection (RDP) for the VM.



10. Complete the steps to establish the RDP session and ensure that you are connected to the new VM.

Task 3: Install WSL (Bash on Ubuntu on Windows)

NOTE: If you are using a Windows 10 development machine, follow these steps. For Mac OS you can ignore this step since you will be using Terminal for all commands.

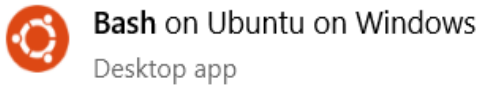
You will need WSL to complete various steps. A complete list of instructions for supported Windows 10 versions is available on this page:

https://docs.microsoft.com/en-us/windows/wsl/install-win10

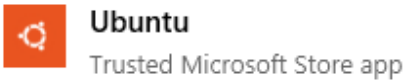
Task 4: Create an SSH key

In this section, you will create an SSH key to securely access the VMs you create during the upcoming exercises.

1. Open a WSL command window.



or



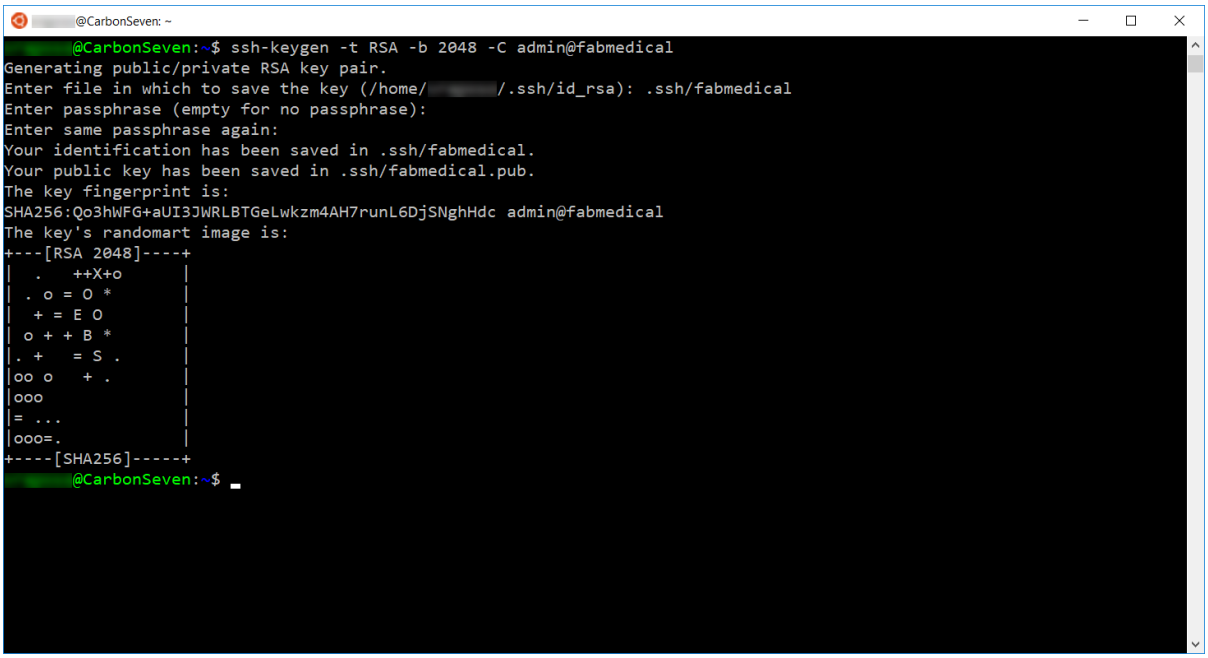
2. From the command line, enter the following command to ensure that a directory for the SSH keys is created. You can ignore any errors you see in the output.

```
mkdir .ssh
```

3. From the command line, enter the following command to generate an SSH key pair. You can replace “admin” with your preferred name or handle.

```
ssh-keygen -t RSA -b 2048 -C admin@fabmedical
```

4. You will be asked to save the generated key to a file. Enter ".ssh/fabmedical" for the name.
5. Enter a passphrase when prompted, and don’t forget it!
6. The file will be generated in your user folder, where WSL opens by default.
7. Keep this WSL window open and remain in the default directory for Task 5: Create a build agent VM.

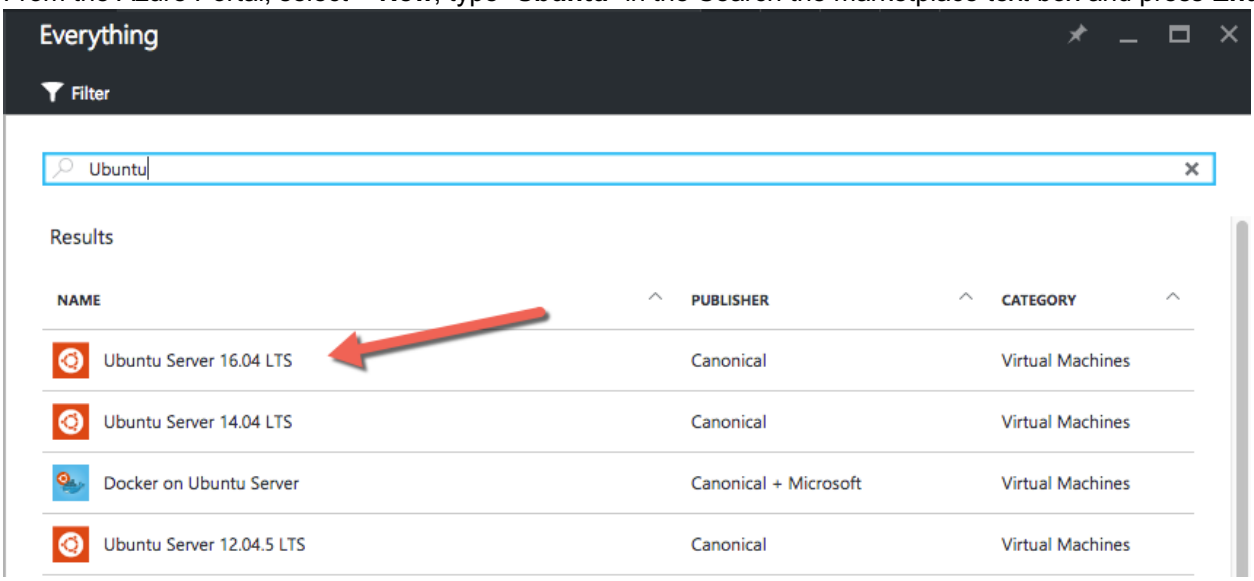


Task 5: Create a build agent VM

In this section, you will create a Linux VM to act as your build agent. You will install Docker to this VM once it is set up, and you will use this VM during the lab to develop and deploy.

NOTE: You can set up your local machine with Docker however the setup varies for different versions of Windows. For this lab, the build agent approach simply allows for predictable setup.

11. From the Azure Portal, select **+ New**, type "**Ubuntu**" in the Search the marketplace text box and press **Enter**.

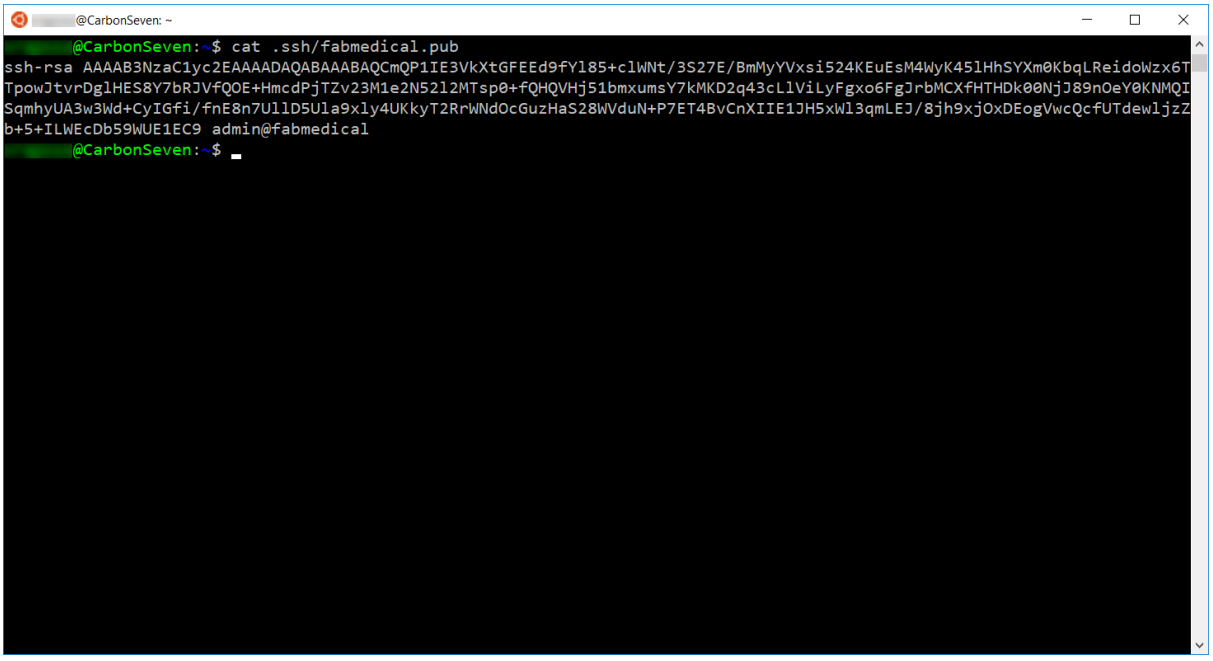


12. Select **Ubuntu Server 16.04 LTS** and select **Create**.
13. On the Basics blade of the Virtual Machine setup, set the following:
    a. **Name**: Provide a unique name, such as "fabmedical-SUFFIX" as shown in the following screenshot.
    b. **VM disk type**: Leave as SSD.
    c. **User name**: Provide a user name, such as "adminfabmedical".
    d. **Authentication type**: Leave as SSH public key.
    e. **SSH public key**: From your local machine, copy the public key portion of the SSH key pair you created previously, to the clipboard.
        i. From WSL, verify you are in your user directory shown as "**~**". This command will take you there:

```
cd ~
```

        ii. Type the following command at the prompt to display the public key that you generated.

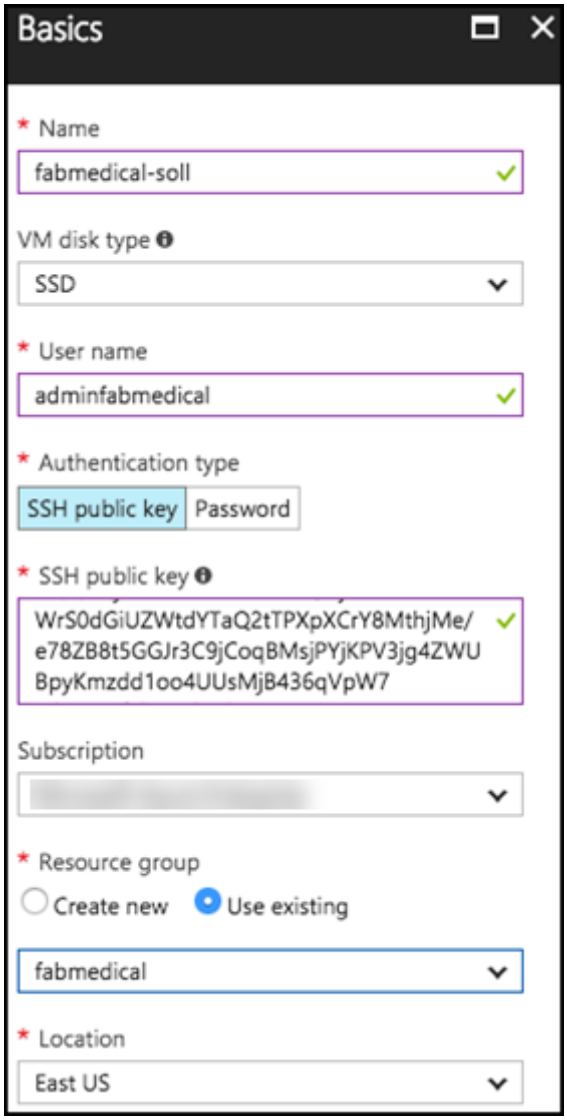
```
cat .ssh/fabmedical.pub
```



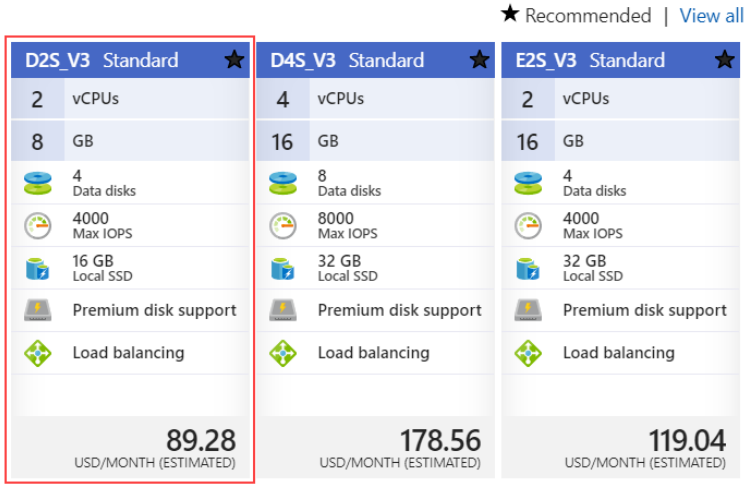
        iii. Copy the entire contents of the file to the clipboard.

iv. Paste this value in the SSH public key textbox of the blade.

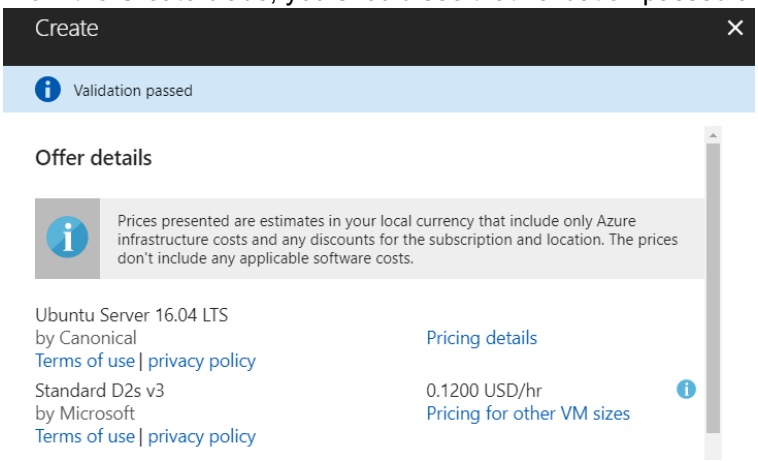
f. **Subscription**: Choose the same subscription you are using for all your work.
g. **Resource group**: Choose Use existing and select the resource group you created previously.
h. **Location**: Choose the same region that you did before.
i. Select **OK** to complete the Basics blade.



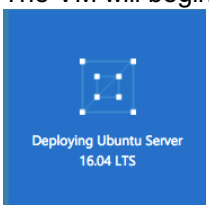
14. From the Size blade, choose **D2S_V3 Standard** and **Select**.



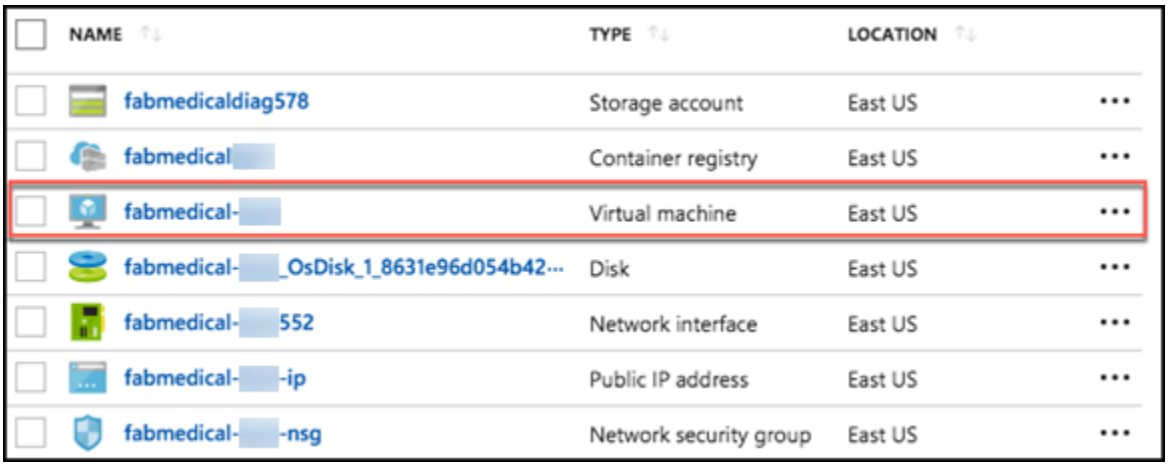
15. From the Settings blade, accept the default values for all settings and select **OK**.
16. From the Create blade, you should see that validation passed and select **Create**.



17. The VM will begin deployment to your Azure subscription.



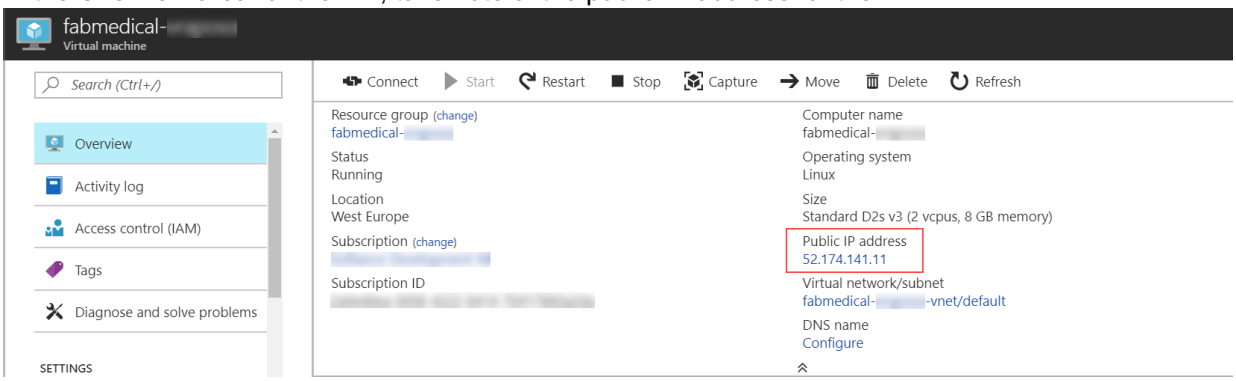
18. Once provisioned, you will see the VM in your list of resources belonging to the resource group you created previously.



Task 6: Connect securely to the build agent

In this section, you will validate that you can connect to the new build agent VM.

1. From the Azure portal, navigate to the Resource Group you created previously and select the new VM, fabmedical-SUFFIX.
2. In the Overview area for the VM, take note of the public IP address for the VM.



3. From your local machine, return to your open WSL window and make sure you are in your user directory ~ where the key pair was previously created. This command will take you there:

```
cd ~
```

4. Connect to the new VM you created by typing the following command.

```
ssh -i [PRIVATEKEYNAME] [BUILDAGENTUSERNAME]@[BUILDAGENTIP]
```

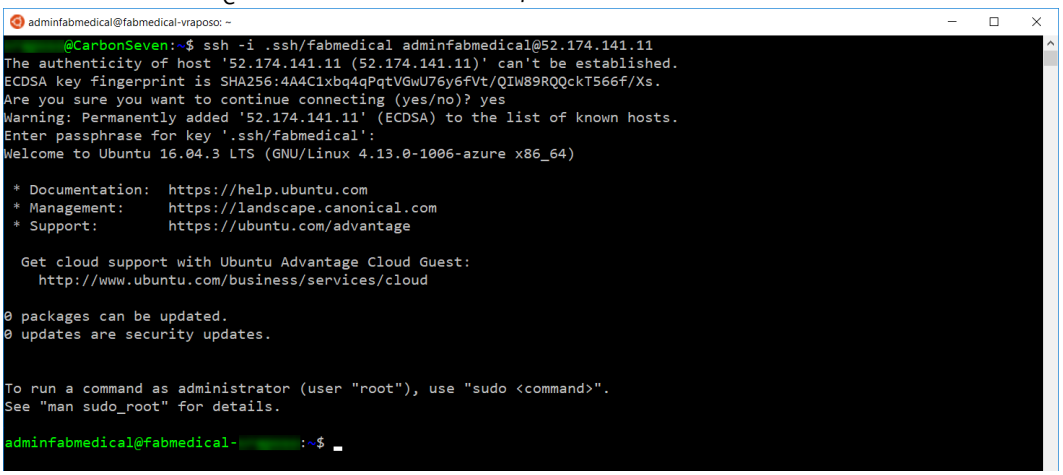
Replace the bracketed values in the command as follows:

- [PRIVATEKEYNAME]: Use the private key name ".ssh/fabmedical," created above.
- [BUILDAGENTUSERNAME]: Use the username for the VM, such as adminfabmedical.
- [BUILDAGENTIP]: The IP address for the build agent VM, retrieved from the VM Overview blade in the Azure Portal.

```
ssh -i .ssh/fabmedical adminfabmedical@52.174.141.11
```

5. When asked to confirm if you want to connect, as the authenticity of the connection cannot be validated, type "yes".
6. When asked for the passphrase for the private key you created previously, enter this value.
7. You will connect to the VM with a command prompt such as the following. Keep this command prompt open for the next step.

adminfabmedical@fabmedical-SUFFIX:~$



NOTE: If you have issues connecting, you may have pasted the SSH public key incorrectly. Unfortunately, if this is the case, you will have to recreate the VM and try again.

Task 7: Complete the build agent setup

In this task, you will update the packages and install Docker engine.

1. Go to the WSL window that has the SSH connection open to the build agent VM.
2. Update the Ubuntu packages and install curl and support for repositories over HTTPS in a single step by typing the following in a single line command. When asked if you would like to proceed, respond by typing “y” and pressing enter.

```
sudo apt-get update && sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

3. Add Docker’s official GPG key by typing the following in a single line command.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

4. Add Docker’s stable repository to Ubuntu packages list by typing the following in a single line command.

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

5. Update the Ubuntu packages and install Docker engine, node.js and the node package manager in a single step by typing the following in a single line command. When asked if you would like to proceed, respond by typing “y” and pressing enter.

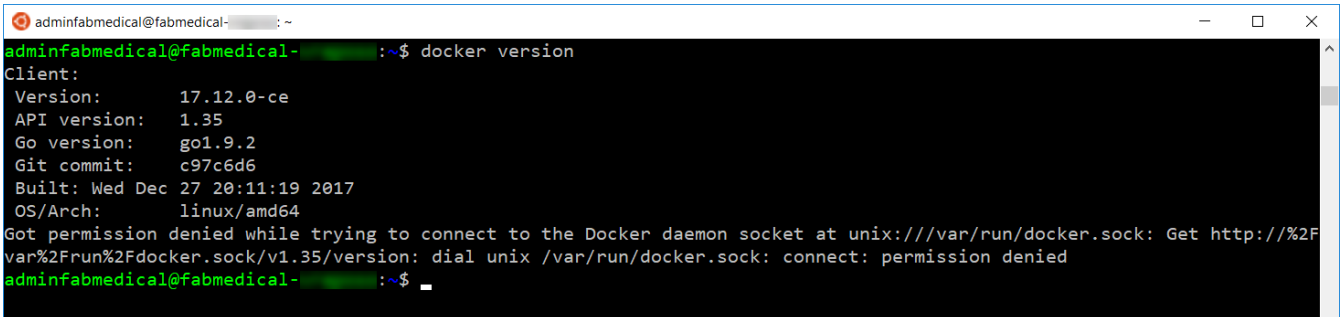
```
sudo apt-get update && sudo apt install docker-ce nodejs npm
```

6. Now, upgrade the Ubuntu packages to the latest version by typing the following in a single line command. When asked if you would like to proceed, respond by typing “y” and pressing enter.

```
sudo apt-get upgrade
```

7. When the command has completed, check the Docker version installed by executing this command. The output may look something like that shown in the following screen shot. Note that the server version is not shown yet, because you didn’t run the command with elevated privileges (to be addressed shortly).

```
docker version
```

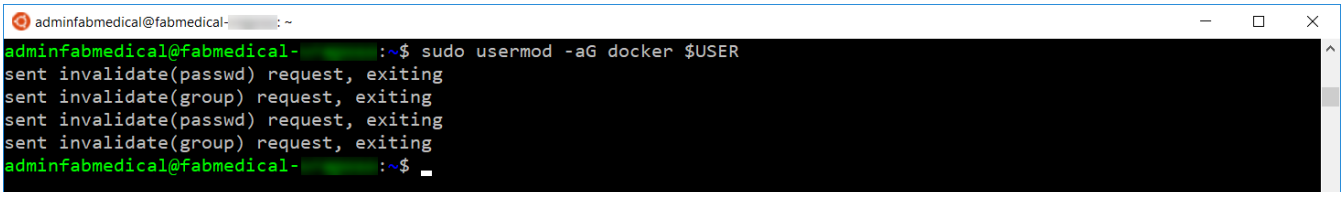


8. You may check the versions of node.js and npm as well, just for information purposes, using these commands.

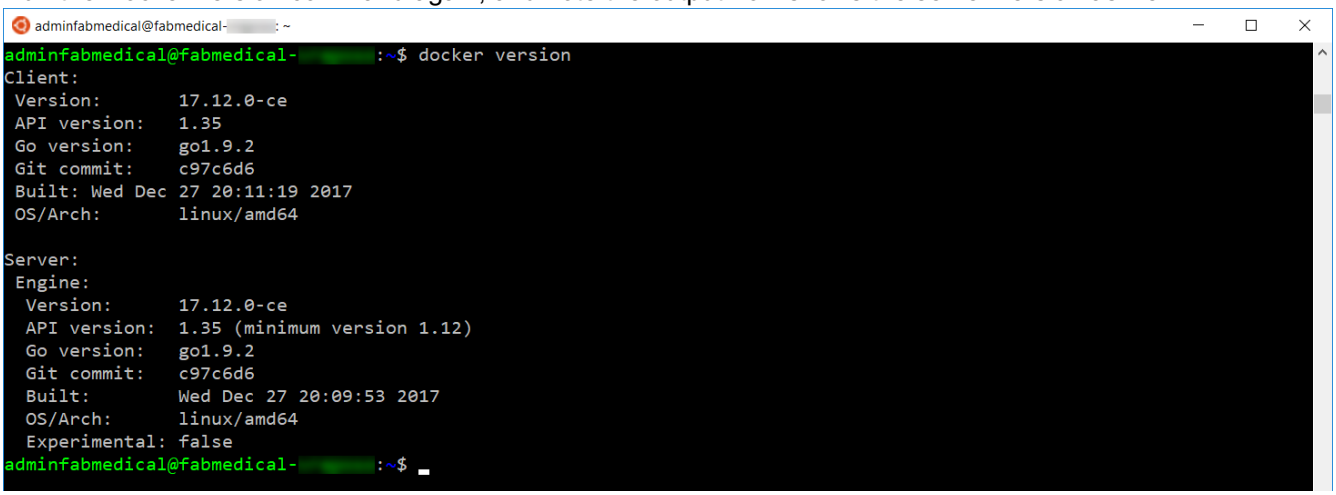
```
nodejs --version
npm -version
```

9. Add your user to the Docker group so that you do not have to elevate privileges with sudo for every command. You can ignore any errors you see in the output.

```
sudo usermod -aG docker $USER
```



10. In order for the user permission changes to take effect, exit the SSH session by typing ‘exit’, then press <Enter>. Repeat the commands in Task 6: Connect securely to the build agent from step 4 to establish the SSH session again.
11. Run the Docker version command again, and note the output now shows the server version as well.



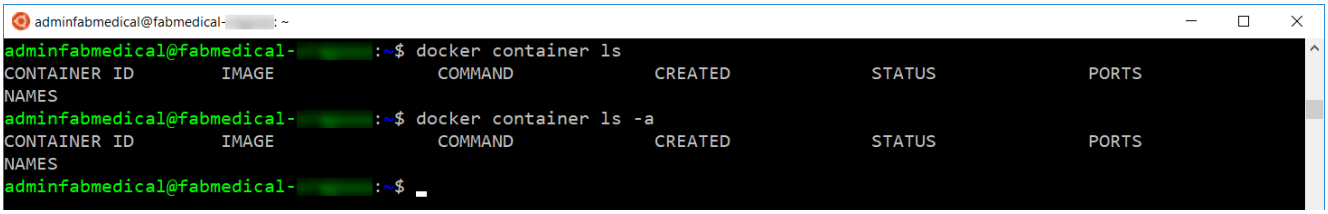
12. Run a few Docker commands.
    a. One to see if there are any containers presently running

    ```
    docker container ls
    ```

    b. One to see if any containers exist whether running or not

    ```
    docker container ls -a
    ```

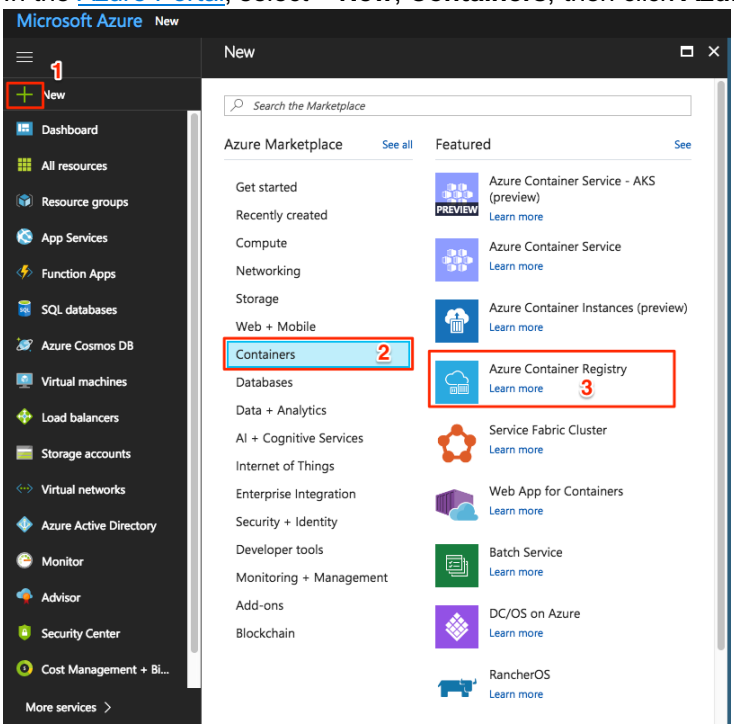
13. In both cases, you will have an empty list but no errors running the command. Your build agent is ready with Docker engine running properly.



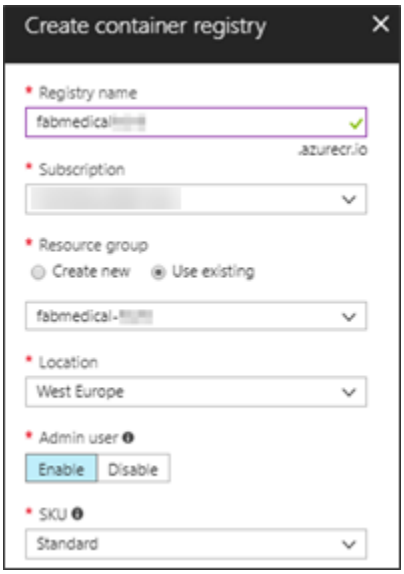
Task 8: Create an Azure Container Registry

You deploy Docker images from a registry. To complete the hands-on lab, you will need access to a registry that is accessible to the Azure Container Service cluster you are creating. In this task, you will create an Azure Container Registry (ACR) for this purpose, where you push images for deployment.

1. In the Azure Portal, select **+ New**, **Containers**, then click **Azure Container Registry**.

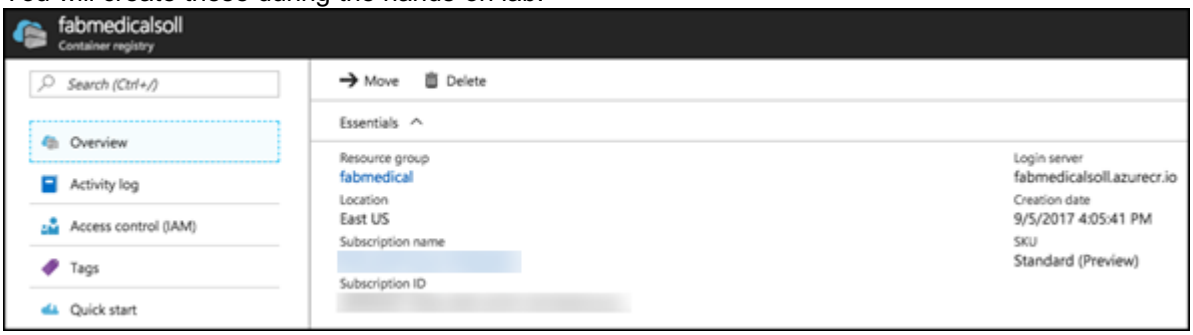


2. On the Create container registry blade, enter the following:
   a. Registry name: Enter a name, such as "fabmedicalSUFFIX," as shown in the following screenshot.
   b. Subscription: Choose the same subscription you are using for all your work.
   c. Resource group: Choose Use existing and select the resource group you created previously.
   d. Location: Choose the same region that you did before.
   e. Admin user: Select Enable.
   f. SKU: Select Standard.



3. Select **Create**.

4. Navigate to your ACR account in the Azure Portal. As this is a new account, you will not see any repositories yet. You will create these during the hands-on lab.

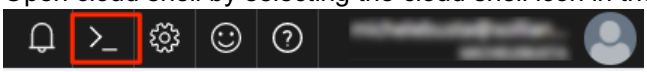


Task 9: Create a Service Principal

In Azure Container Service, a Kubernetes cluster requires an Azure Active Directory service principal to interact with Azure APIs. The service principal is needed to dynamically manage resources such as user-defined routes and the Layer 4 Azure Load Balancer. The easiest way to set up the service principal is using the Azure cloud shell.

NOTE: By default, creating a service principal in Azure AD requires account owner permission. You may have trouble creating a service principal if you are not the account owner.

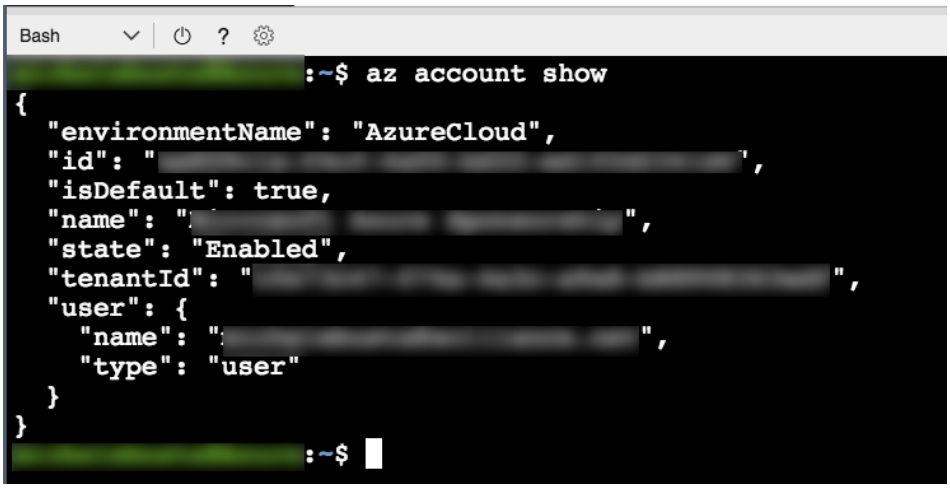
1. Open cloud shell by selecting the cloud shell icon in the menu bar.



2. The cloud shell will open in the browser window. Choose “Bash (Linux)” if prompted, or use the left hand dropdown on the shell menu bar to choose “Bash” (as shown).

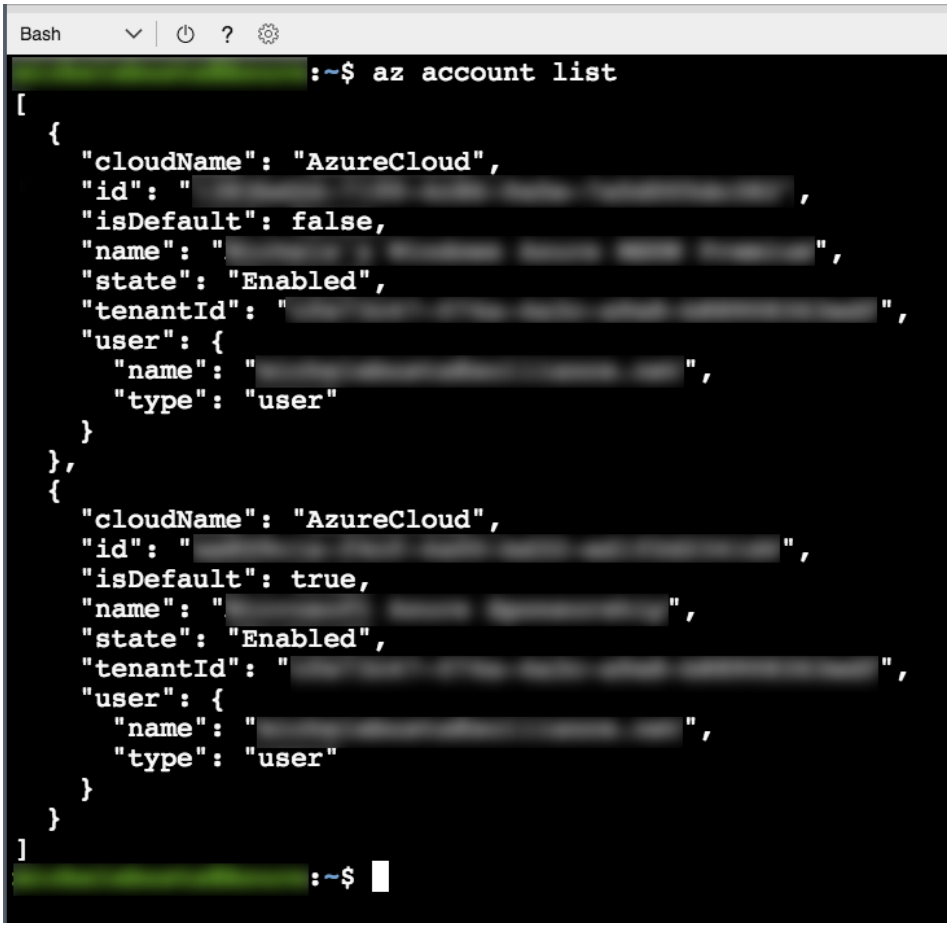
3. Before completing the steps to create the service principal, you should make sure to set your default subscription correctly. To view your current subscription type:

```
az account show
```



4. To list all of your subscriptions, type:

```
az account list
```



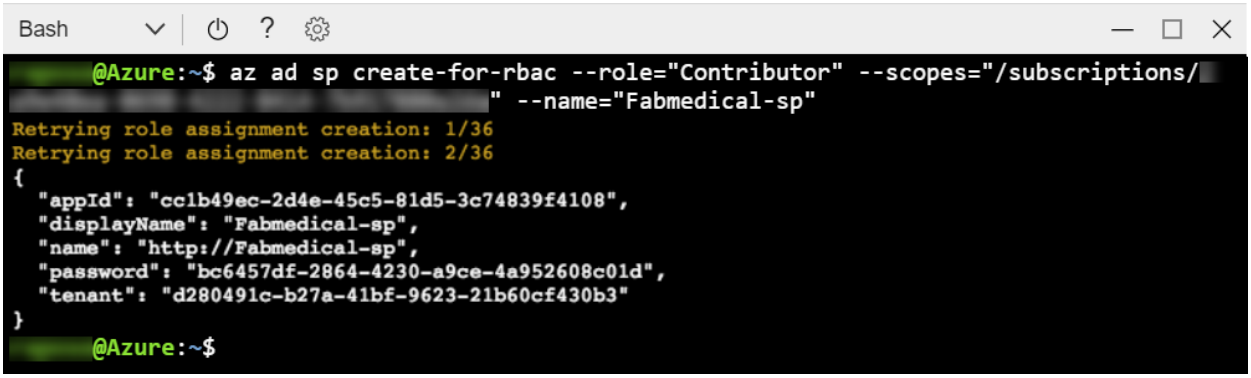
5. To set your default subscription to something other than the current selection, type the following, replacing {id} with the desired subscription id value:

```
az account set --subscription {id}
```

6. To create a service principal, type the following command, replacing {id} with your subscription identifier.

```
az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/{id}" --
name="Fabmedical-sp"
```

7. The service principal command will produce output like this. Copy this information; you will need it later.



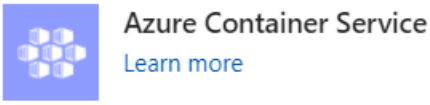
Task 10: Create an Azure Container Service cluster

In this task, you will create your Azure Container Service cluster based on Kubernetes. You will use the same SSH key you created previously to connect to this cluster in the next task.

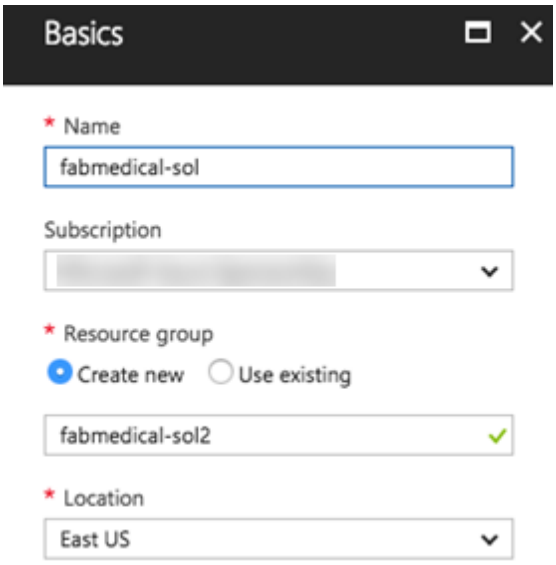
NOTE: In this step, you will be asked to create a second Resource Group as part of the template for creating the Azure Container Service cluster. This is due to the current template not providing the option to select an *existing* Resource Group, and this may change in the future.

NOTE: "Azure Container Service – AKS" is currently in preview and is not used in this lab. Make sure you choose the correct option "Azure Container Service".

1. From the Azure Portal, select **+ New**, **Containers** and select **Azure Container Service**.

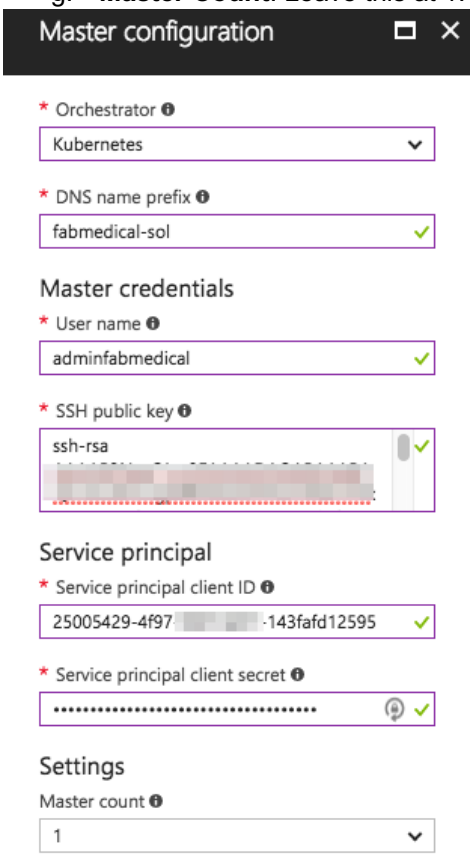


2. In the Basics blade provide the information shown in the screenshot that follows:
   a. **Name**: Enter fabmedical-SUFFIX.
   b. **Subscription**: Choose your subscription.
   c. **Resource group**: Create new and provide a unique name. Since the template does not support using an existing Resource Group, provide a new name such adding a "2" to the suffix, such as fabmedical-SUFFIX2.
   d. **Location**: Choose a location.

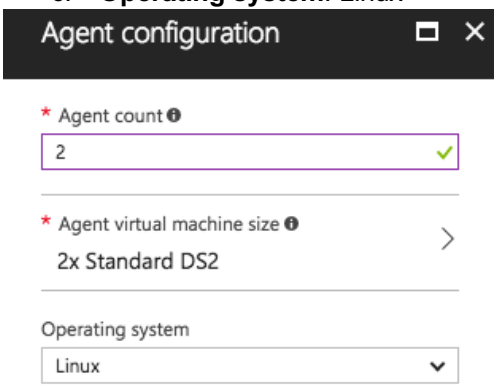


3. Select **OK**.
4. On the Master configuration blade provide:
   a. **Orchestrator**: Kubernetes
   b. **DNS Name Prefix**: Set the DNS prefix to something like "fabmedical-SUFFIX".
   c. **User name**: Enter a username such as "adminfabmedical".
   d. **SSH Public Key**: Paste the same SSH public key you used for the agent VM previously.
   e. **Service principal client ID**: Use the service principal "appId" from the previous step.
   f. **Service principal client secret**: Use the service principal "password" from the previous step.

g. **Master Count**: Leave this at 1.

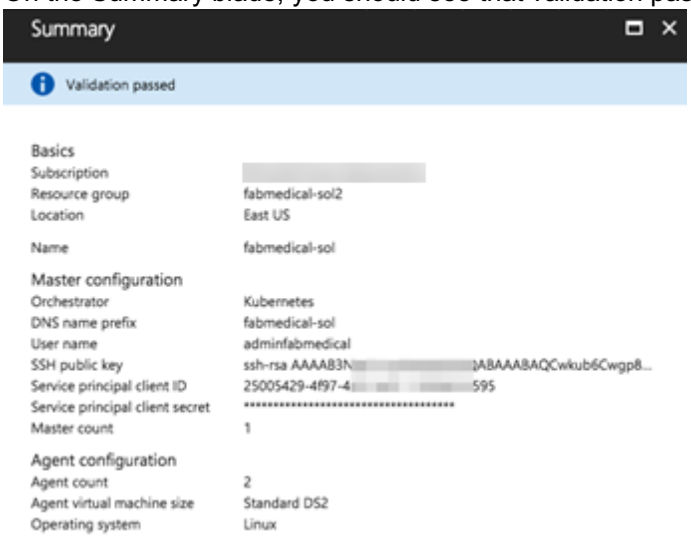


5. Select **OK**.
6. On the Agent configuration blade, set the following settings:
   a. **Agent count**: 2
   b. **Agent virtual size**: Standard DS2
   c. **Operating system**: Linux

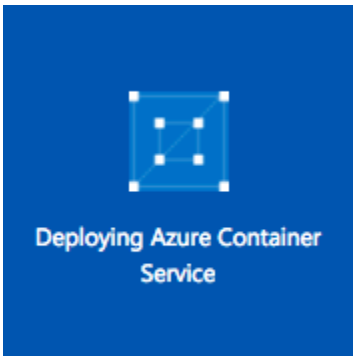


7. Select **OK**.

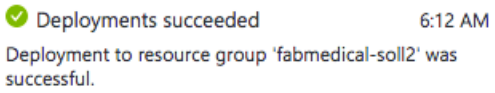
8. On the Summary blade, you should see that validation passed; select **OK**.



9. The Azure Container Service cluster will begin deployment to your Azure subscription.



10. You should see a successful deployment notification when the cluster is ready. It can take up to 10 minutes before your Azure Container Service cluster is listed in the Azure Portal. You can proceed to the next step while waiting for this to complete, then return to view the success of the deployment.



NOTE: If you experience errors related to lack of available cores, you may have to delete some other compute resources or request additional cores to your subscription and then try this again.

Task 11: Install Azure CLI

In later exercises, you will need the Azure CLI 2.0 to connect to your Kubernetes cluster and run commands from your local machine. A complete list of instructions for supported platforms is available on this page:

https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest

1. For MacOS – use homebrew

```
brew update
brew install azure-cli
```

2. For Windows – using WSL

```
echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-cli/ wheezy main" | sudo
tee /etc/apt/sources.list.d/azure-cli.list
```

```
sudo apt-key adv --keyserver packages.microsoft.com --recv-keys
52E16F86FEE04B979B07E28DB02C46DF417A0893

sudo apt-get install apt-transport-https

sudo apt-get update && sudo apt-get install azure-cli
```

Task 12: Install Kubernetes CLI

In later exercises, you will need the Kubernetes CLI (kubectl) to deploy to your Kubernetes cluster and run commands from your local machine.

1. Regardless of the operating system, considering the usage of WSL on Windows, install the Kubernetes client using Azure CLI

```
az login

sudo az acs kubernetes install-cli --install-location /usr/local/bin/kubectl
```

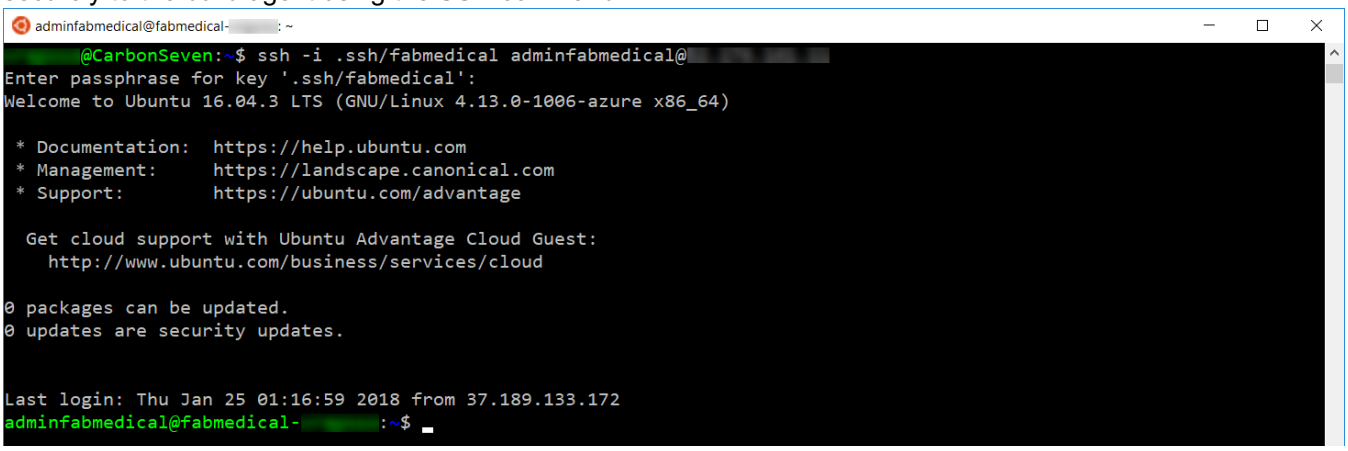
Exercise 1: Environment setup

Duration: 10 minutes

FabMedical has provided starter files for you. They have taken a copy of one of their websites, for their customer Contoso Neuro, and refactored it from a single node.js site into a website with a content API that serves up the speakers and sessions. This is a starting point to validate the containerization of their websites. They have asked you to use this to help them complete a POC that validates the development workflow for running the website and API as Docker containers and managing them within the Azure Container Service Kubernetes environment.

Task 1: Download the FabMedical starter files

1. From WSL, connect to the build agent VM as you did previously in Before the hands-on lab - Task 6: Connect securely to the build agent using the SSH command.



2. Download the starter files to the build agent by typing the following curl instruction (case sensitive):

```
curl -L -o FabMedical.tgz http://bit.ly/2ALnIYI
```

3. Create a new directory named FabMedical by typing in the following command:

```
mkdir FabMedical
```

4. Unpack the archive with the following command. This command will extract the files from the archive to the FabMedical directory you created. The directory is case sensitive when you navigate to it.

```
tar -C FabMedical -xzf FabMedical.tgz
```

5. Navigate to FabMedical folder and list the contents.

```
cd FabMedical
ll
```

6. You'll see the listing includes two folders, one for the web site and another for the content API.

```
content-api/
content-web/
```

NOTE: Keep this WSL window open as your build agent SSH connection. You will later open new WSL sessions to other machines.

Exercise 2: Create and run a Docker application

Duration: 40 minutes

In this exercise, you will take the starter files and run the node.js application as a Docker application. You will create a Dockerfile, build Docker images, and run containers to execute the application.

NOTE: Complete these tasks from the WSL window with the build agent session.

Task 1: Test the application

The purpose of this task is to make sure you can run the application successfully before applying changes to run it as a Docker application.

1. From the WSL window, navigate to the content-api directory and run npm install.

```
cd content-api
npm install
```

2. Start the API as a background process.

```
nodejs ./server.js &
```

```
adminfabmedical@fabmedical-[redacted]:~/FabMedical/content-api$ nodejs ./server.js &
[1] 73302
adminfabmedical@fabmedical-[redacted]:~/FabMedical/content-api$ Listening on port 3001
```

3. Press ENTER again to get to a command prompt for the next step.
4. Test the API using curl. You will request the speakers content, and this will return a JSON result.

```
curl http://localhost:3001/speakers
```

5. Navigate to the web application directory, run npm install, and then run the application as a background process as well. Ignore any warnings you see in the output; this will not affect running the application.

```
cd ..
cd content-web
npm install
nodejs ./server.js &
```

```
adminfabmedical@fabmedical-[redacted]:~/FabMedical/content-web$ nodejs ./server.js &
[2] 73785
adminfabmedical@fabmedical-[redacted]:~/FabMedical/content-web$ === Using development environment ===
Express server listening on port 3000
```

6. Press ENTER again to get a command prompt for the next step.
7. Test the web application using curl. You will see HTML output returned, without errors.

```
curl http://localhost:3000
```

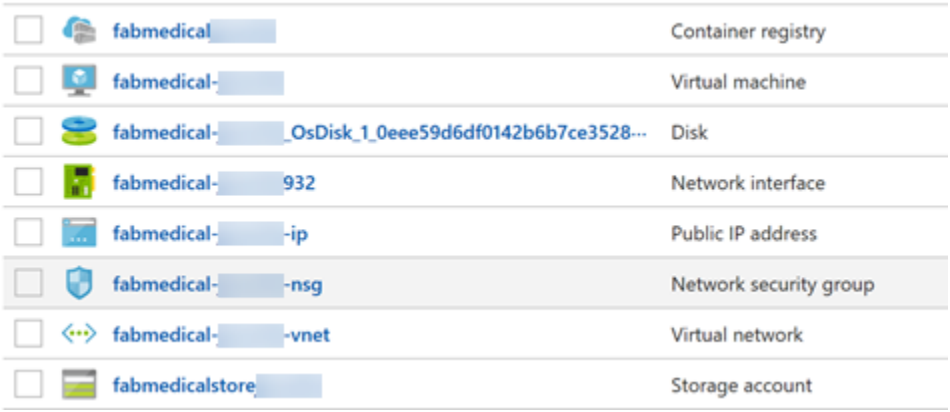
8. Leave the application running for the next task.
9. If you received a JSON response to the /speakers content request and an HTML response from the web application, your environment is working as expected.

Task 2: Enable browsing to the web application

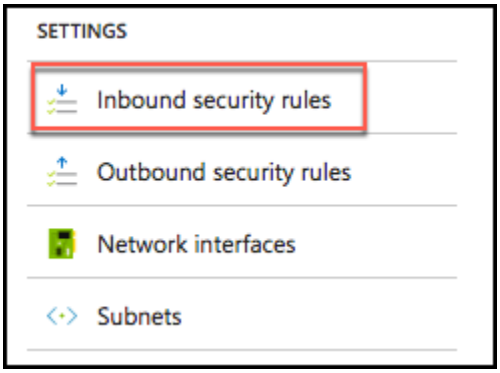
In this task, you will open a port range on the agent VM so that you can browse to the web application for testing.

1. From the Azure portal select the resource group you created named fabmedical-SUFFIX.

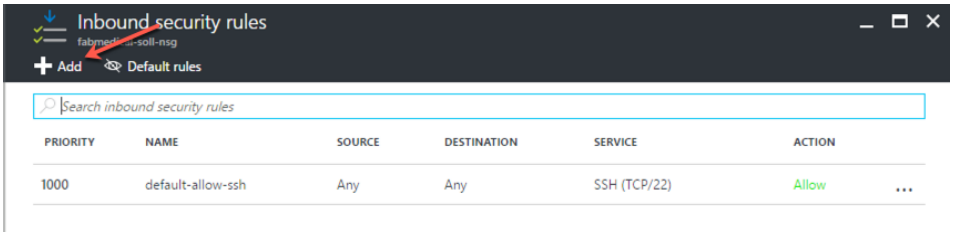
2. Select the Network Security Group associated with the build agent from your list of available resources.



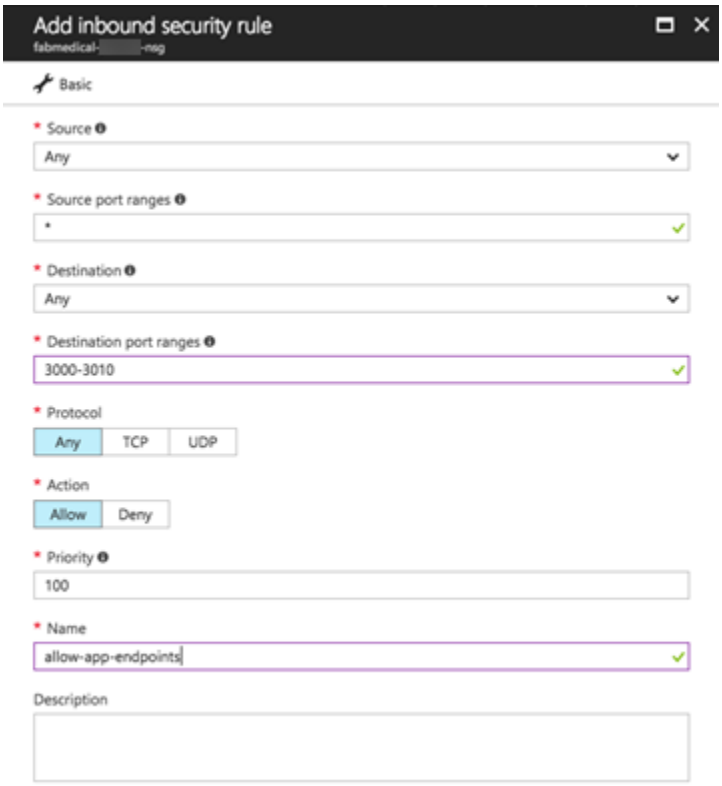
3. From the Network interface essentials blade, select **Inbound security rules**.



4. Select **Add** to add a new rule.



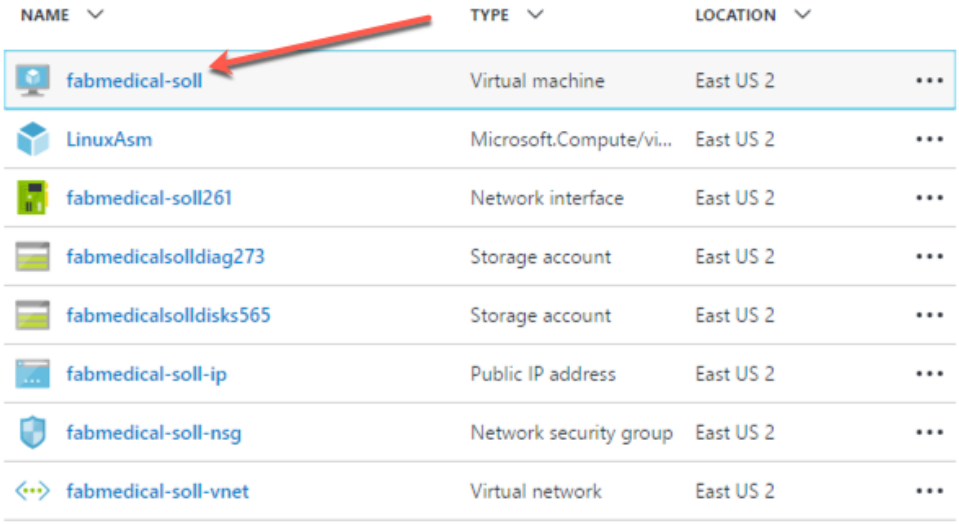
5. From the Add inbound security rule blade, enter the values as shown in the screenshot below:
    a. **Source**: Any
    b. **Source port ranges**: *
    c. **Destination**: Any
    d. **Destination Port Ranges**: 3000-3010
    e. **Protocol**: Any
    f. **Action**: Allow
    g. **Priority**: Leave at the default priority setting
    h. **Name**: Enter “allow-app-endpoints”



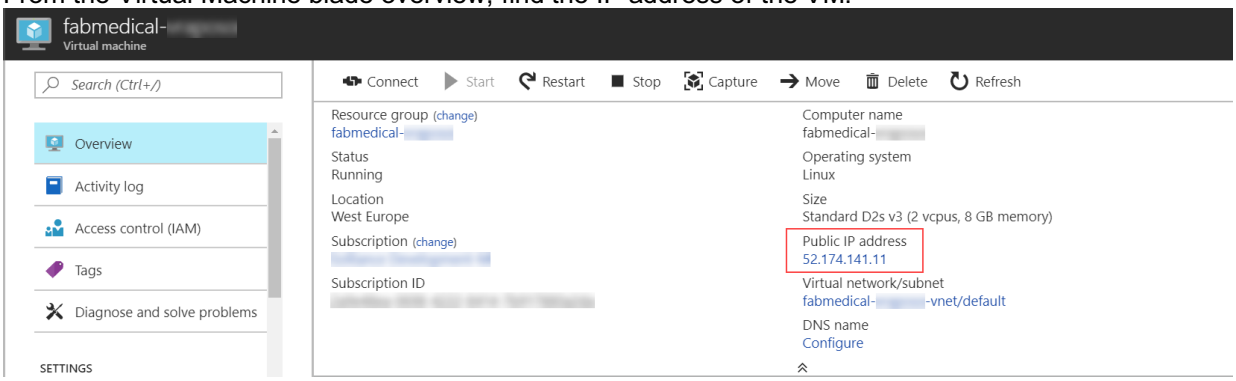
6. Select **OK** to save the new rule.

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
100	allow-app-endpoints	3000-3010	Any	Any	Any	Allow
1000	default-allow-ssh	22	TCP	Any	Any	Allow

7. From the resource list shown in step 2, select the build agent VM named fabmedical-SUFFIX.



8. From the Virtual Machine blade overview, find the IP address of the VM.



9. Test the web application from a browser. Navigate to the web application using your build agent IP address at port 3000.

```
http://[BUILDAGENTIP]:3000
EXAMPLE: http://13.68.113.176:3000
```

10. Select the Speakers and Sessions links in the header. You will see the pages display the HTML version of the JSON content you curled previously.
11. Once you have verified the application is accessible through a browser, go to your WSL window and stop the running node processes.

```
killall nodejs
```

Task 3: Create a Dockerfile

In this task, you will create a new Dockerfile that will be used to run the API application as a containerized application.

NOTE: You will be working in a Linux VM without friendly editor tools. You must follow the steps very carefully to work with Vim for a few editing exercises, if you are not already familiar with Vim.

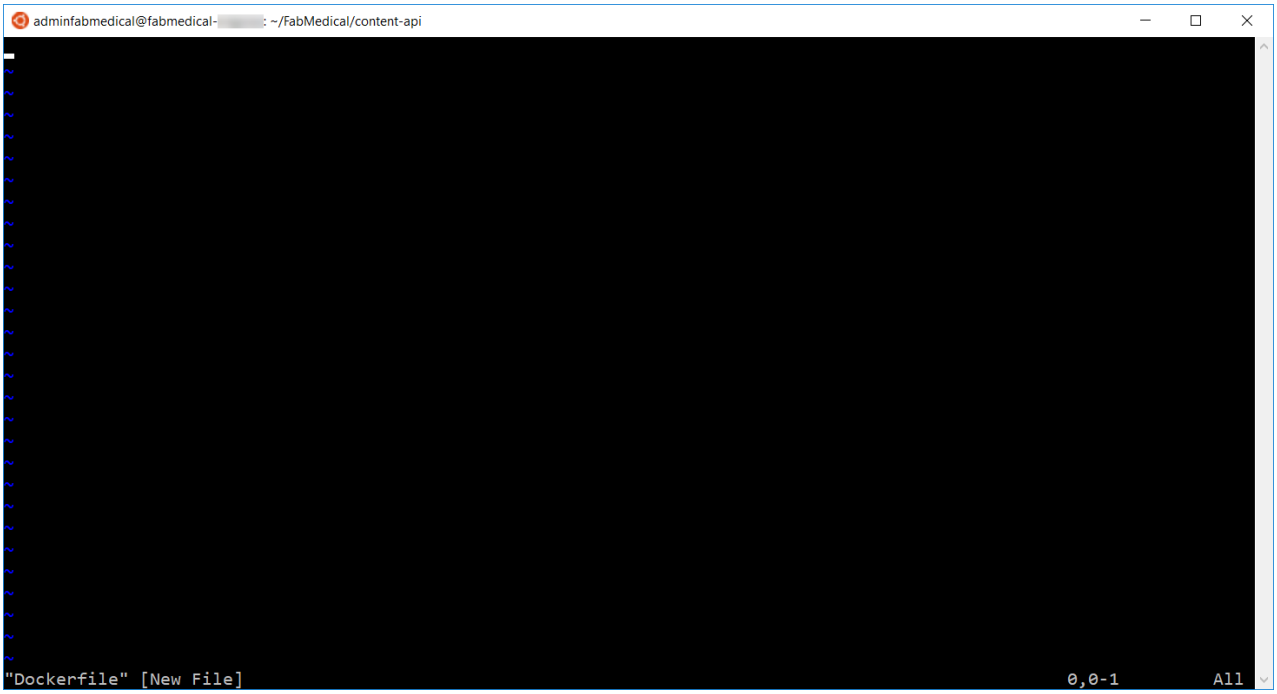
1. From WSL, navigate to the content-api folder. List the files in the folder with this command. The output should look like the screenshot below.

```
cd ..
cd content-api
ll
```

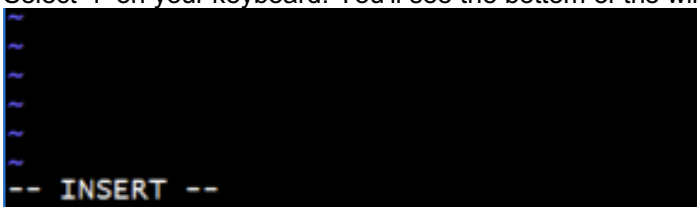
```
adminfabmedical@fabmedical-hackathon:~/FabMedical/FabMedical/content-api$ ll
total 72
drwxr-xr-x  3 adminfabmedical adminfabmedical  4096 Jun  8 19:41 ./
drwxr-xr-x  4 adminfabmedical adminfabmedical  4096 Sep 20  2016 ../
-rwxr-xr-x  1 adminfabmedical adminfabmedical   222 Sep 20  2016 ._Dockerfile-finish*
-rwxr-xr-x  1 adminfabmedical adminfabmedical   239 Sep 20  2016 Dockerfile-finish*
drwxrwxr-x 49 adminfabmedical adminfabmedical  4096 Jun  8 19:41 node_modules/
-rwxr-xr-x  1 adminfabmedical adminfabmedical   222 Sep 20  2016 ._package.json*
-rwxr-xr-x  1 adminfabmedical adminfabmedical   224 Sep 20  2016 package.json*
-rwxr-xr-x  1 adminfabmedical adminfabmedical   222 Sep 20  2016 ._routes.js*
-rwxr-xr-x  1 adminfabmedical adminfabmedical  1016 Sep 20  2016 routes.js*
-rwxr-xr-x  1 adminfabmedical adminfabmedical   222 Sep 20  2016 ._server.js*
-rwxr-xr-x  1 adminfabmedical adminfabmedical   960 Sep 20  2016 server.js*
-rwxr-xr-x  1 adminfabmedical adminfabmedical   222 Sep 20  2016 ._sessions.json*
-rwxr-xr-x  1 adminfabmedical adminfabmedical 11008 Sep 20  2016 sessions.json*
-rwxr-xr-x  1 adminfabmedical adminfabmedical   222 Sep 20  2016 ._speakers.json*
-rwxr-xr-x  1 adminfabmedical adminfabmedical  5448 Sep 20  2016 speakers.json*
```

2. Create a new file named “Dockerfile” and note the casing in the name. Use the following Vim command to create a new file. The WSL window should look as shown in the following screenshot.

```
vi Dockerfile
```



3. Select “i” on your keyboard. You’ll see the bottom of the window showing INSERT mode.



4. Type the following into the file. These statements produce a Dockerfile that describes the following:
    a. Creates a new Docker image from the base image node:argon. This base image has node.js on it and is compatible with the Ubuntu distribution of Linux you are deploying to.
    b. Creates a directory on the image where the application files can be copied.
    c. Copies the source files for the application over to the image.
    d. Runs npm install to initialize the node application environment.
    e. Exposes application port 3001 to the container environment so that the application can be reached at port 3001.
    f. Indicates the command to start the node application when the container is run.

NOTE: Type the following into the editor, as you may have errors with copying and pasting.

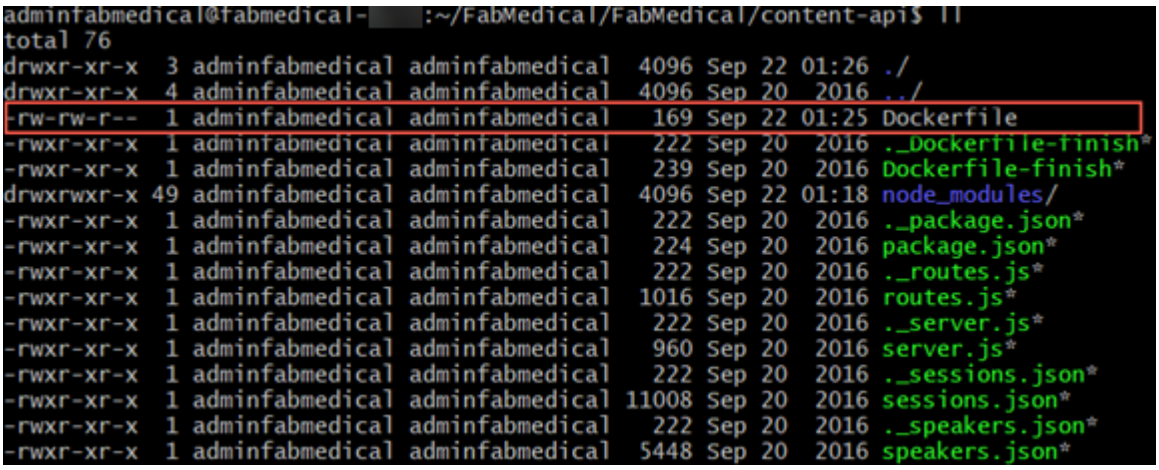
```
FROM node:argon

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY package.json /usr/src/app/
RUN npm install
COPY . /usr/src/app

EXPOSE 3001
CMD [ "npm", "start" ]
```

5. When you are finished typing, hit the Esc key and type “:wq” and hit the Enter key to save the changes and close the file.


```
<Esc>
:wq
<Enter>
```

6. List the contents of the folder again, to verify the new Dockerfile has been created.

```
ll
```



7. Verify the file contents, to ensure it was saved as expected. Type the following command to see the output of the Dockerfile in the command window.

```
cat Dockerfile
```

Task 4: Create Docker images

In this task, you will create Docker images for the application — one for the API application and another for the web application. Each image will be created via Docker commands that rely on a Dockerfile.

1. From WSL, type the following command to view any Docker images on the VM. The list will be empty.

```
docker images
```

2. From the content-api folder containing the API application files and the new Dockerfile you created, type the following command to create a Docker image for the API application. This command does the following:
   a. Executes the Docker build command to produce the image.
   b. Tags the resulting image with the name content-api (-t).
   c. The final dot (“.”) indicates to use the Dockerfile in this current directory context. By default, this file is expected to have the name “Dockerfile” (case sensitive).

```
docker build -t content-api .
```

3. Once the image is successfully built, run the Docker images command again. You will see two images: the node image and your container image.

```
docker images
```

4. Navigate to the content-web folder again and list the files. Note that this folder already has a Dockerfile.

```
cd ..
cd content-web
ll
```

5. View the Dockerfile contents – which are similar to the file you created previously in the API folder. Type the following command:

```
cat Dockerfile
```

6. Type the following command to create a Docker image for the web application.

```
docker build -t content-web .
```

7. When complete, you will see three images now exist when you run the Docker images command.

```
docker images
```

```
adminfabmedical@fabmedical:~/FabMedical/FabMedical/content-web$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
content-web         latest              c25f7830cfc1        5 seconds ago       724.5 MB
content-api         latest              52adf58220d4        About a minute ago  661.9 MB
node                argon               cfe7aced3467        8 days ago          654.6 MB
adminfabmedical@fabmedical:~/FabMedical/FabMedical/content-web$
```

Task 5: Run a containerized application

The web application container will be calling endpoints exposed by the API application container. In this exercise, you will create a bridge network so that containers can communicate with one another, and then launch the images you created as containers in that network.

1. From WSL, type the following command to create a Docker network named “fabmedical”.

```
docker network create fabmedical
```

2. Create and start the API application container with the following command. The command does the following:
   a. Creates a container from the specified image, by its tag, such as content-api.
   b. Names the container “api” for later reference with Docker commands.
   c. Issues the Docker run command indicating it should run the new container as a daemon.
   d. Instructs the Docker engine to use port 3001 and map that to the internal container port 3001.
   e. Instructs the Docker engine to use the “fabmedical” network.

```
docker run -d -p 3001:3001 --name api --net fabmedical content-api
```

3. Enter the command to show running containers. You’ll observe that the “api” container is in the list.

```
docker container ls
```

```
adminfabmedical@fabmedical-[redacted]:~/FabMedical/content-web$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
548d25a1449f        content-api         "npm start"         8 seconds ago       Up 6 seconds        0.0.0.0:3001->3001/tcp   api
adminfabmedical@fabmedical-[redacted]:~/FabMedical/content-web$
```

4. Test the API by curling the URL. You will see JSON output as you did when testing previously.

```
curl http://localhost:3001/speakers
```

5. Create and start the web application container with a similar Docker run command – instruct the docker engine to use any port with the -P command:

```
docker run -d -P --name web --net fabmedical content-web
```

6. Enter the command to show running containers again and you’ll observe that both the API and web containers are in the list. The web container shows a dynamically assigned port mapping to its internal container port 3000.

```
docker ps
```

```
adminfabmedical@fabmedical-[redacted]:~/FabMedical/content-web$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                     NAMES
b540dd250fc2        content-web         "npm start"         16 seconds ago      Up 14 seconds       0.0.0.0:32768->3000/tcp   web
548d25a1449f        content-api         "npm start"         3 minutes ago       Up 3 minutes        0.0.0.0:3001->3001/tcp    api
adminfabmedical@fabmedical-[redacted]:~/FabMedical/content-web$
```

7. Test the web application by curling the URL. For the port, use the dynamically assigned port, which you can find in the output from the previous command. You will see HTML output, as you did when testing previously.


```
curl http://localhost:[PORT]/speakers.html
```

Task 6: Setup environment variables

In this task, you will configure the “web” container to communicate with the API container using an environment variable. You will modify the web application to read the URL from the environment variable, rebuild the Docker image, and then run the container again to test connectivity.

1. From WSL, stop and remove the web container using the following commands.

```
docker stop web
docker rm web
```

2. Validate that the web container is no longer running or present by using the -a flag as shown in this command. You will see that the “web” container is no longer listed.

```
docker container ls -a
```

3. Navigate to the content-web\data-access directory. From there, open the index.js file for editing using Vim and press the “i" key to go into edit mode.

```
cd data-access
vi index.js
<i>
```

4. Locate the following TODO item and modify the code to comment the first line and uncomment the second. The result is that the contentApiUrl variable will be set to an environment variable.

```
//TODO: Exercise 2 - Task 6 - Step 4
//var contentApiUrl = “http://localhost:3001”;
var contentApiUrl = process.env.CONTENT_API_URL;
```

5. Press the Escape key and type “:wq” and then press the Enter key to save and close the file.

```
<Esc>
:wq
<Enter>
```

6. Navigate to the content-web directory. From there open the Dockerfile for editing using Vim and press the “i" key to go into edit mode.

```
cd ..
vi Dockerfile
<i>
```

7. Locate the EXPOSE line shown below and add a line above it that sets the environment variable as shown in the screenshot.

```
ENV CONTENT_API_URL http://localhost:3001
```

```
FROM node:argon

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

ENV CONTENT_API_URL http://localhost:3001

EXPOSE 3000
CMD [ "npm", "start" ]
~
~
```

8. Press the Escape key and type ":wq" and then press the Enter key to save and close the file.

```
<Esc>
:wq
<Enter>
```

9. Rebuild the web application Docker image using the same command as you did previously.

```
docker build -t content-web .
```

10. Create and start the image passing the correct URI to the API container as an environment variable. This variable will address the API application using its container name over the Docker network you created. After running the container, check to see the container is running and note the dynamic port assignment for the next step.

```
docker run -d -P --name web --net fabmedical -e CONTENT_API_URL=http://api:3001 content-web

docker container ls
```

11. Curl the speakers path again, using the port assigned to the web container. This time you will see HTML returned.

```
curl http://localhost:[PORT]/speakers.html
```

12. You will not be able to browse to the web application unless the agent VM exposes the port. Now you will stop the web container and restart it using port 3000 to test in the browser. To do this, type the following commands to stop the container, remove it, and run it again using explicit settings for the port.

```
docker stop web
docker rm web
docker run -d -p 3000:3000 --name web --net fabmedical -e CONTENT_API_URL=http://api:3001
content-web
```

13. Curl the speaker path again, using port 3000. You will see the same HTML returned.

```
curl http://localhost:3000/speakers.html
```

14. You can now use a web browser to navigate to the website and successfully view the application at port 3000. Replace [BUILDAGENTIP] with the IP address you used previously.

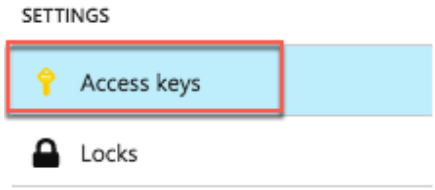
```
http://[BUILDAGENTIP]:3000
EXAMPLE: http://13.68.113.176:3000
```

Task 7: Push images to Azure Container Registry

To run containers in a remote environment, you will typically push images to a Docker registry, where you can store and distribute images. Each service will have a repository that can be pushed to and pulled from with Docker commands. Azure Container Registry (ACR) is a managed private Docker registry service based on Docker Registry v2.

In this task, you will push images to your ACR account, version images with tagging, and run containers from an image in your ACR.

1. In the Azure Portal, navigate to the ACR you created in Before the hands-on lab.
2. Select Access keys under Settings on the left-hand menu.



3. The Access keys blade displays the Login server, username, and password that will be required for the next step. Keep this handy as you perform actions on the build VM.

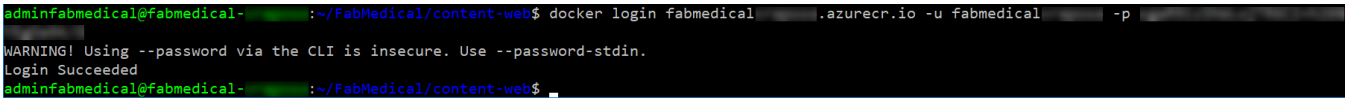
   NOTE: If the username and password do not appear, select Enable on the Admin user option.

4. From the WSL session connected to your build VM, login to your ACR account by typing the following command. Follow the instructions to complete the login.

```
docker login [LOGINSERVER] -u [USERNAME] -p [PASSWORD]
```

   For example:

```
docker login fabmedicalsoll.azurecr.io -u fabmedicalsoll -p
+W/j=l+Fcze=n07SchxvGSlvsLRh/7ga
```



   Tip: Make sure to specify the fully qualified registry login server (all lowercase).

5. Run the following commands to properly tag your images to match your ACR account name. This also specifies the "fabmedical" namespace to avoid clutter in the root of the registry.

```
docker tag content-web [LOGINSERVER]/fabmedical/content-web
docker tag content-api [LOGINSERVER]/fabmedical/content-api
```

6. List your docker images and look at the repository and tag. Note that the repository is prefixed with your ACR account name, and fabmedical namespace, such as the sample shown in the screenshot below.

```
docker images
```

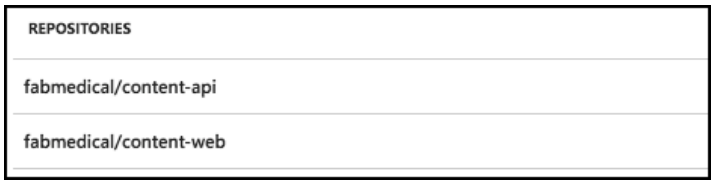
```
adminfabmedical@fabmedical:~/FabMedical/FabMedical/content-web$ docker images
REPOSITORY                                          TAG                 IMAGE ID            CREATED             SIZE
content-web                                         latest              fdd59f1cafc3        53 minutes ago      724.5 MB
fabmedicalsoll.azurecr.io/fabmedical/content-web    latest              fdd59f1cafc3        53 minutes ago      724.5 MB
<none>                                              <none>              c25f7830cfc1        About an hour ago   724.5 MB
fabmedicalsoll.azurecr.io/fabmedical/content-api    latest              52adf58220d4        About an hour ago   661.9 MB
content-api                                         latest              52adf58220d4        About an hour ago   661.9 MB
node                                                argon               cfe7aced3467        8 days ago          654.6 MB
adminfabmedical@fabmedical:~/FabMedical/FabMedical/content-web$
```

7. Push the images to your ACR account with the following command.

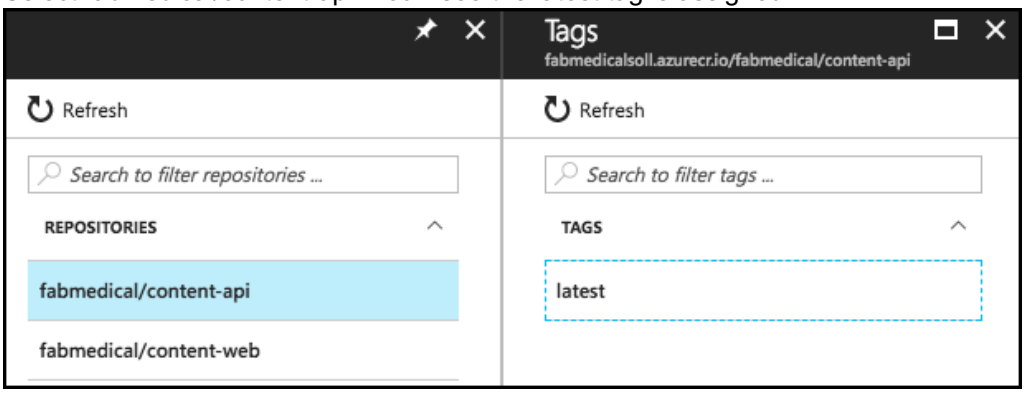
```
docker push [LOGINSERVER]/fabmedical/content-web
docker push [LOGINSERVER]/fabmedical/content-api
```

```
adminfabmedical@fabmedical-      :~/FabMedical/content-web$ docker push fabmedical      .azurecr.io/fabmedical/content-web
The push refers to repository [fabmedical      .azurecr.io/fabmedical/content-web]
6ba4c178398f: Pushing [==================================================> ]  27.72MB/28.03MB
3102f625fb5f: Pushing [=========================>                         ]  22.63MB/42.33MB
78df18d4aa61: Pushed
85a41a83aef5: Pushed
15c8638d8d77: Pushed
fcf07eec7d39: Pushing [=======>                                           ]  5.513MB/36.8MB
50599c766115: Pushing [==================================================>]  135.2kB
d4141af68ac4: Pushing [==================================================>]  352.8kB
8fe6d5dcea45: Waiting
06b8d020c11b: Waiting
b9914afd042f: Waiting
4bcdffd70da2: Waiting
```

8. In the Azure Portal, navigate to your ACR account, and select Repositories under Services on the left-hand menu. You will now see two, one for each image.



9. Select fabmedical/content-api. You'll see the latest tag is assigned.

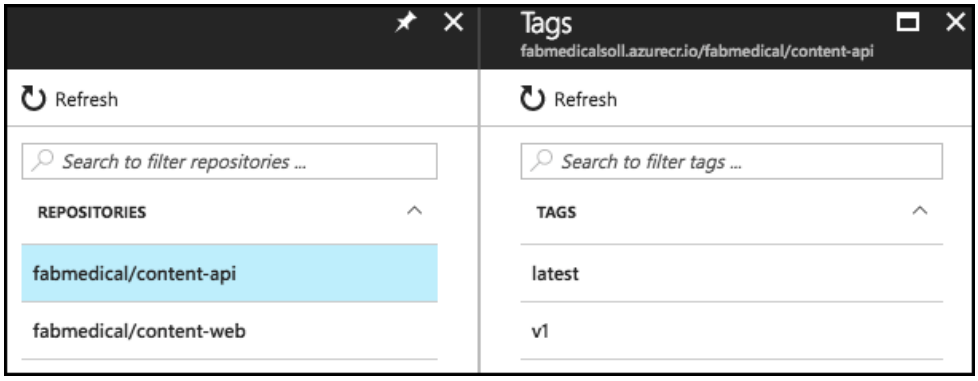


10. From WSL, assign the v1 tag to each image with the following commands. Then, list the Docker images to note that there are now two entries for each image, showing the latest tag and the v1 tag. Also note that the image ID is the same for the two entries, as there is only one copy of the image.

```
docker tag [LOGINSERVER]/fabmedical/content-web:latest [LOGINSERVER]/fabmedical/content-web:v1

docker tag [LOGINSERVER]/fabmedical/content-api:latest [LOGINSERVER]/fabmedical/content-api:v1

docker images
```

```
adminfabmedical@fabmedical:~/FabMedical/FabMedical/content-web$ docker images
REPOSITORY                                          TAG                 IMAGE ID            CREATED             SIZE
content-web                                         latest              fdd59f1cafc3        About an hour ago   724.5 MB
fabmedicalsoll.azurecr.io/fabmedical/content-web    latest              fdd59f1cafc3        About an hour ago   724.5 MB
fabmedicalsoll.azurecr.io/fabmedical/content-web    v1                  fdd59f1cafc3        About an hour ago   724.5 MB
<none>                                              <none>              c25f7830cfc1        About an hour ago   724.5 MB
content-api                                         latest              52adf58220d4        About an hour ago   661.9 MB
fabmedicalsoll.azurecr.io/fabmedical/content-api    latest              52adf58220d4        About an hour ago   661.9 MB
fabmedicalsoll.azurecr.io/fabmedical/content-api    v1                  52adf58220d4        About an hour ago   661.9 MB
node                                                argon               cfe7aced3467        8 days ago          654.6 MB
adminfabmedical@fabmedical:~/FabMedical/FabMedical/content-web$
```

11. Repeat Step 7 to push the images to ACR again, so that the newly tagged v1 images are pushed. Then, refresh one of the repositories to see the two versions of the image now appear.



12. Run the following commands to pull an image from the repository. Note that the default behavior is to pull images tagged with “latest.” You can pull a specific version using the version tag. Also, note that since the images already exist on the build agent, nothing is downloaded.

```
docker pull [LOGINSERVER]/fabmedical/content-web
docker pull [LOGINSERVER]/fabmedical/content-web:v1
```

```
adminfabmedical@fabmedical-      :~/FabMedical/content-web$ docker pull fabmedical      .azurecr.io/fabmedical/content-web
Using default tag: latest
latest: Pulling from fabmedical/content-web
Digest: sha256:743d8c4bc06a2baa635034fc54eda12d795273e65cd9e57da9348e7a3acdb6f6
Status: Image is up to date for fabmedical      .azurecr.io/fabmedical/content-web:latest
adminfabmedical@fabmedical-      :~/FabMedical/content-web$ docker pull fabmedical      .azurecr.io/fabmedical/content-web:v1
v1: Pulling from fabmedical/content-web
Digest: sha256:743d8c4bc06a2baa635034fc54eda12d795273e65cd9e57da9348e7a3acdb6f6
Status: Image is up to date for fabmedical      .azurecr.io/fabmedical/content-web:v1
adminfabmedical@fabmedical-      :~/FabMedical/content-web$
```

Exercise 3: Deploy the solution to Azure Container Service

Duration: 30 minutes

In this exercise, you will connect to the Azure Container Service cluster you created before the hands-on lab and deploy the Docker application to the cluster using Kubernetes.

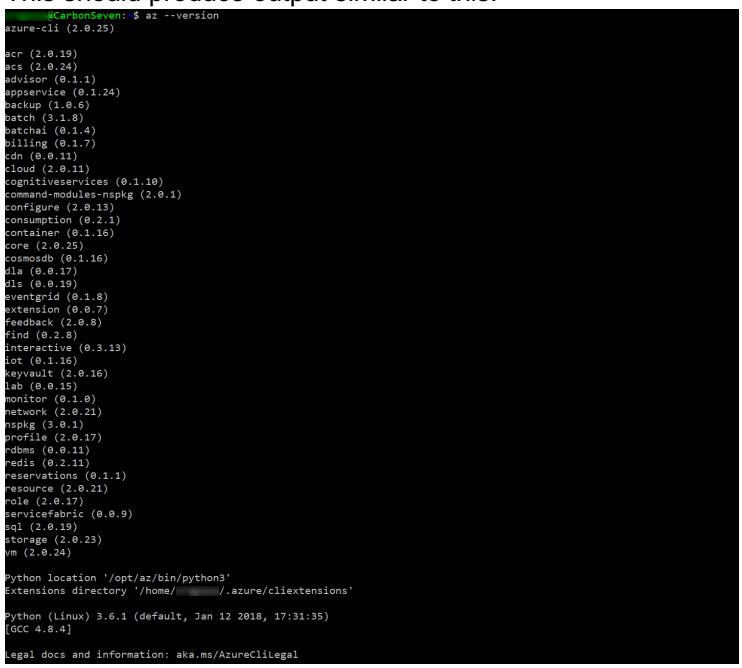
Task 1: Tunnel into the Azure Container Service cluster

In this task, you will gather the information you need about your Azure Container Service cluster to connect to the cluster and execute commands to connect to the Kubernetes management dashboard from your local machine.

1. Open your WSL console. From this WSL console, ensure that you installed the Azure CLI correctly by running the following command:

```
az --version
```

   a. This should produce output similar to this:

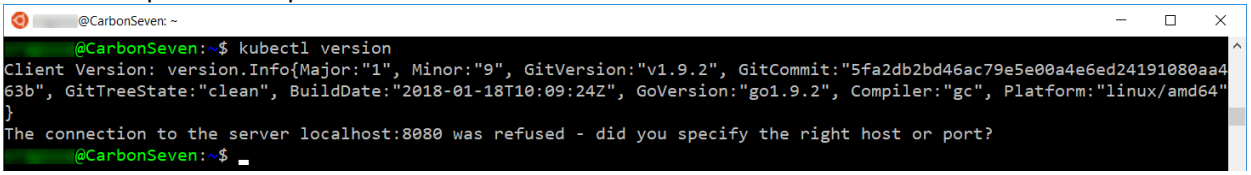


   b. If the output is not correct, review your steps from the instructions in Task 11: Install Azure CLI from the instructions before the lab exercises.

2. Also, check the installation of the Kubernetes CLI (kubectl) by running the following command:

```
kubectl version
```

   a. This should produce output similar to this:



   b. If the output is not correct, review the steps from the instructions in Task 12: Install Kubernetes CLI from the instructions before the lab exercises.

3. Once you have installed and verified Azure CLI and Kubernetes CLI, login with the following command, following the instructions to complete your login as presented:

```
az login
```


4. Verify that you are connected to the correct subscription with the following command to show your default subscription:

```
az account show
```

   a. If you are not connected to the correct subscription, list your subscriptions and then set the subscription by its id with the following commands (similar to what you did in cloud shell before the lab):

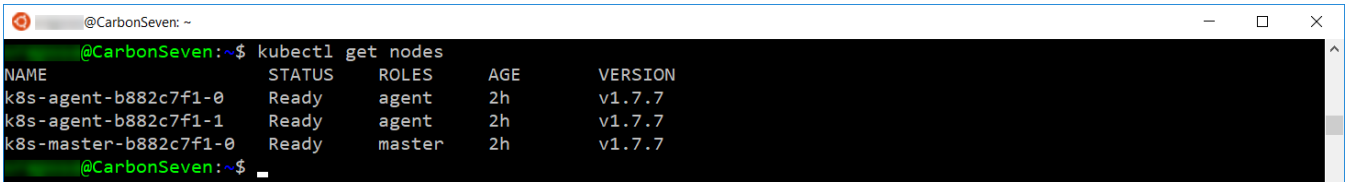
```
az account list
az account set --subscription {id}
```

5. Configure kubectl to connect to the Kubernetes cluster. You must specify your fabmedical ssh key.

```
az acs kubernetes get-credentials --resource-group=fabmedical-SUFFIX2 --name=fabmedical-SUFFIX --ssh-key-file=.ssh/fabmedical
```

6. Test that the configuration is correct by running a simple kubectl command to produce a list of nodes:

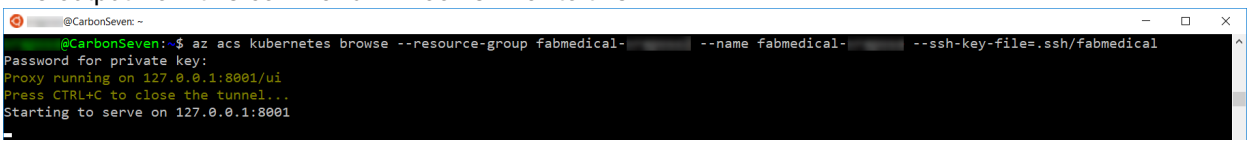
```
kubectl get nodes
```



7. Create an SSH tunnel linking a local port (8001) on your machine to port 80 on the management node of the cluster. Execute the command below replacing the values as follows:
   a. Replace [PRIVATEKEY] with the location of your SSH key file, such as .ssh/fabmedical
   b. Replace [RESOURCEGROUP] with the resource group used when creating the ACS cluster, such as fabmedical-sol2
   c. Replace [CONTAINERINSTANCE] with the container service instance name used when creating the ACS cluster, such as fabmedical-sol

```
az acs kubernetes browse --resource-group [RESOURCEGROUP] --name [CONTAINERINSTANCE] --ssh-key-file=[PRIVATEKEY]
```

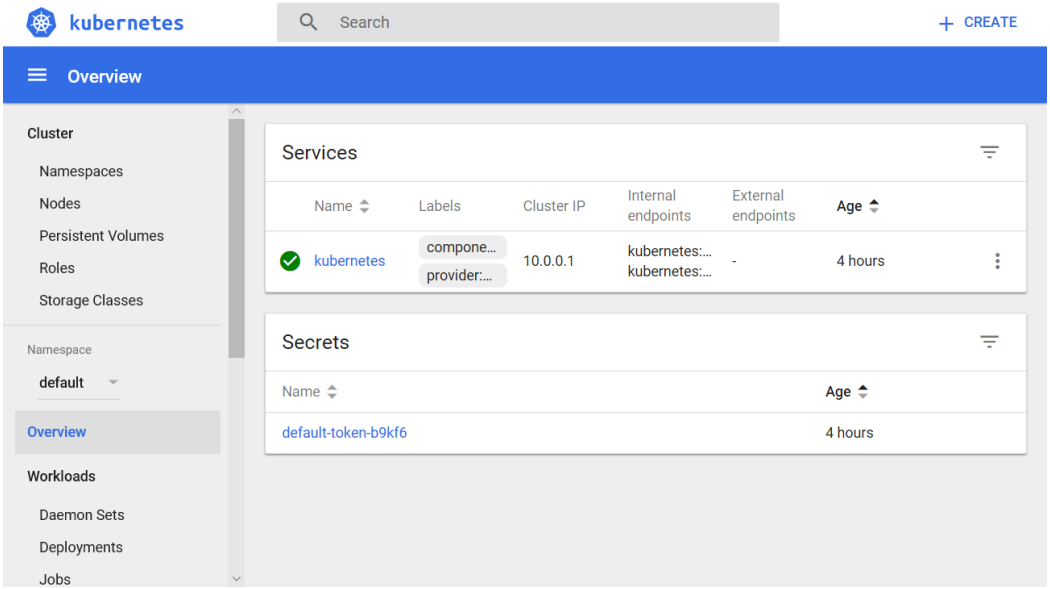
   d. The output from this command will look similar to this:



8. Open a browser window and access the Kubernetes management dashboard at the Services view. To reach the dashboard, you must access the following address:

```
http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/kubernetes-dashboard/proxy/#
```

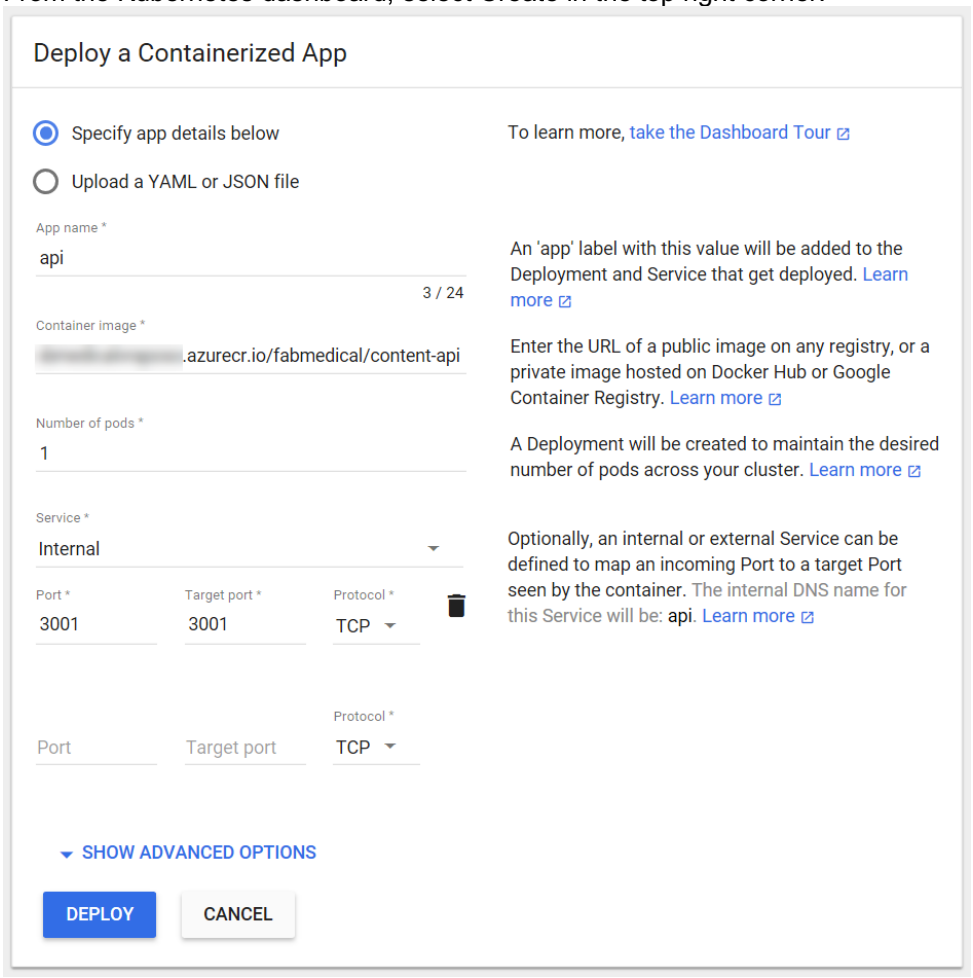
9. If the tunnel is successful, you will see the Kubernetes management dashboard.



Task 2: Deploy a service using the Kubernetes management dashboard

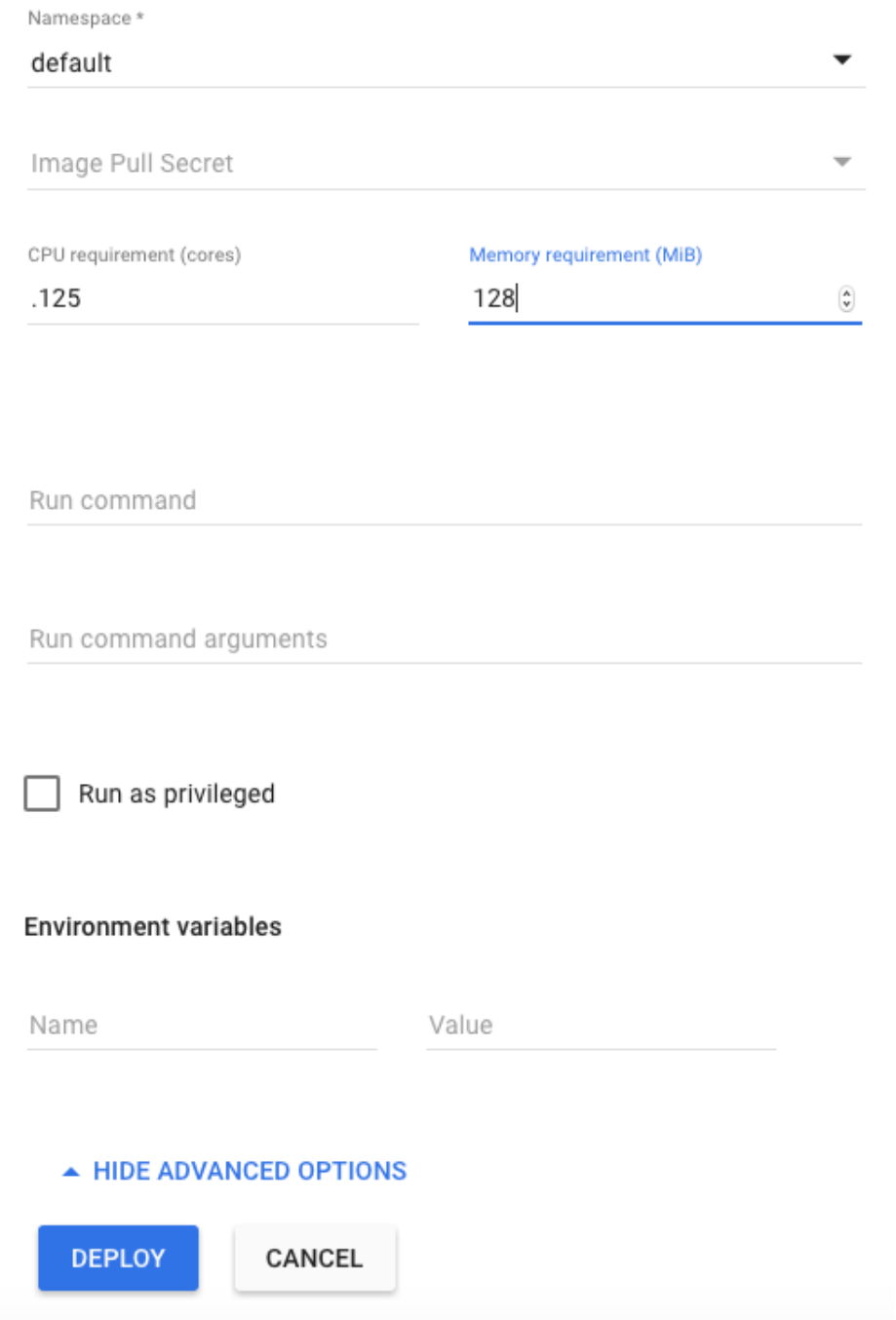
In this task, you will deploy the API application to the Azure Container Service cluster using the Kubernetes dashboard.

1. From the Kubernetes dashboard, select Create in the top right corner.

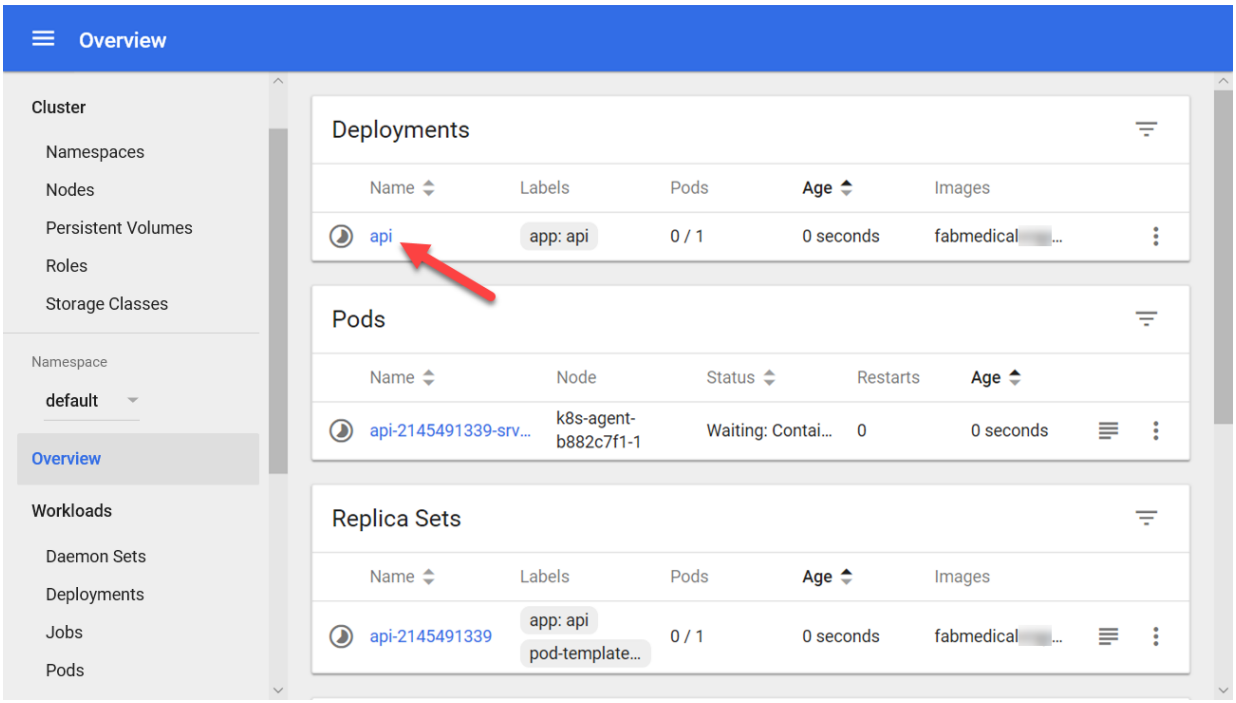


   a. Enter “api” for the App name.

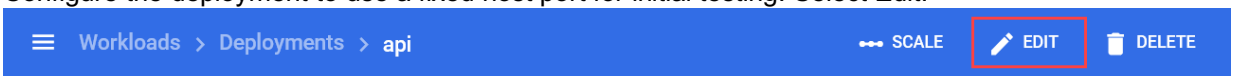
   b. Enter [LOGINSERVER]/fabmedical/content-api for the Container Image, replacing [LOGINSERVER] with your ACR login server, such as fabmedicalsol.azurecr.io
   c. Set Number of pods to 1.
   d. Set Service to “Internal”.
   e. Use 3001 for Port and 3001 for Target port.
2. Select SHOW ADVANCED OPTIONS.
   a. Enter 0.125 for the CPU requirement.
   b. Enter 128 for the Memory requirement.



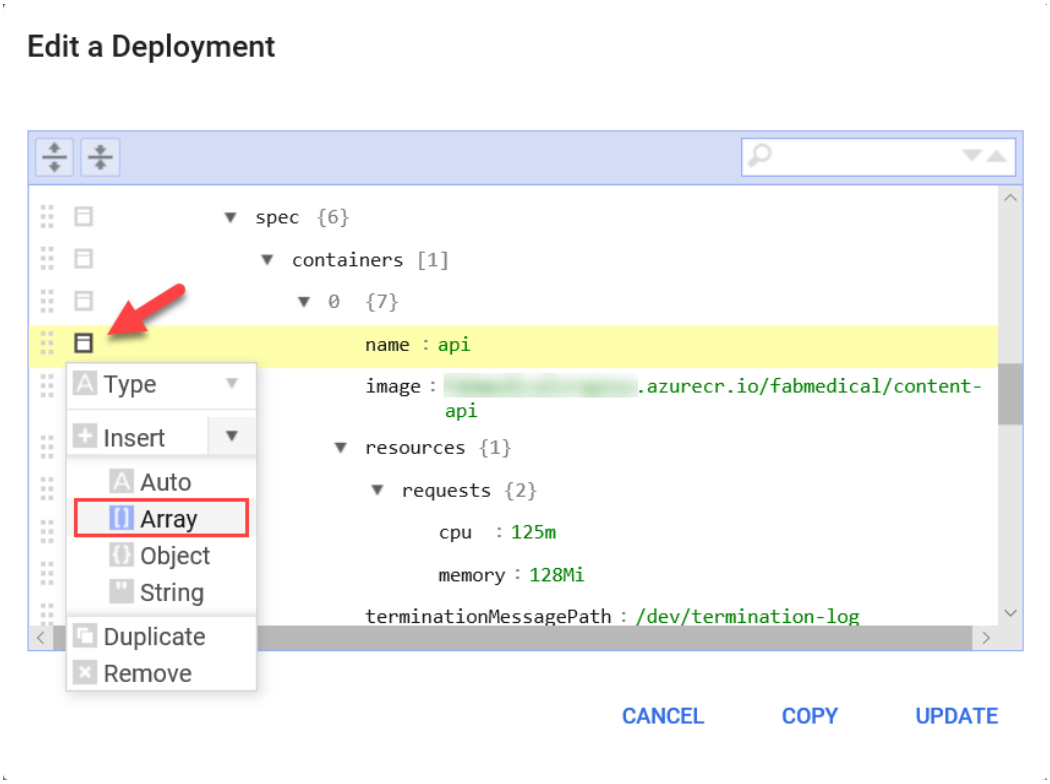
3. Select Deploy to initiate the service deployment based on the image. This can take a few minutes. In the meantime, you will be redirected to the Overview dashboard. Select the API deployment from the Overview dashboard to see the deployment in progress.



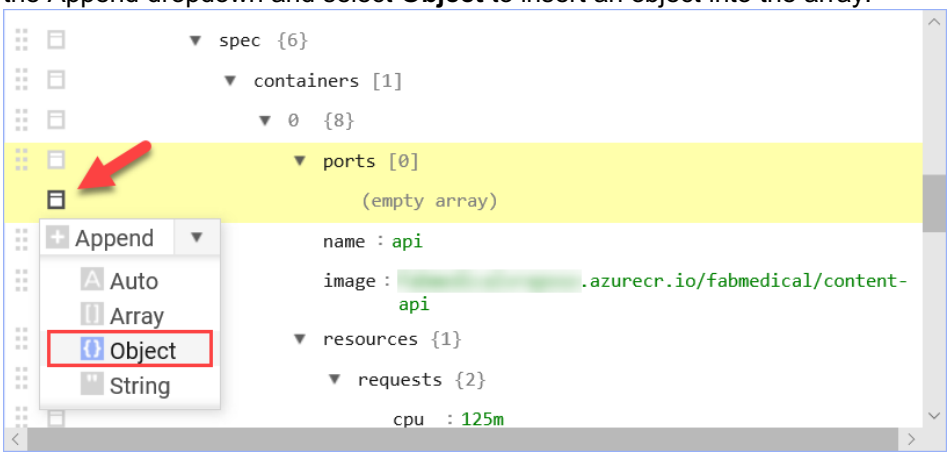
4. Configure the deployment to use a fixed host port for initial testing. Select Edit.



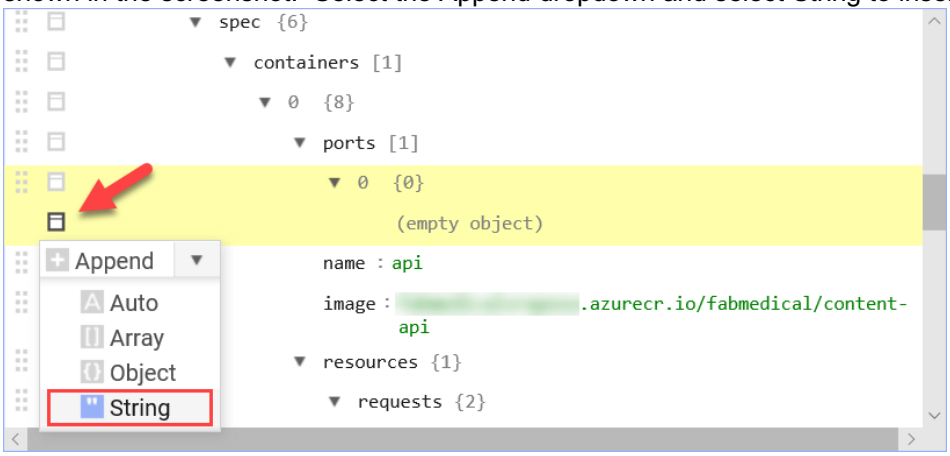
5. In the Edit a Deployment dialog you will see a list of settings shown in an expanded tree format. If you scroll down half way you will see a spec node as shown in the screenshot below. The containers spec has a single entry for the API container at the moment, shown under the first container array entry shown as 0 in the screenshot. Beneath this, you'll see the name of the container API — this is how you know you are looking at the correct container spec.
    a. Select the menu icon on the name node and you'll see the menu shown in the screenshot.
    b. From this menu, select the Insert dropdown.
    c. Select **Array** to insert an array inside the 0 node at the same level as the name node.



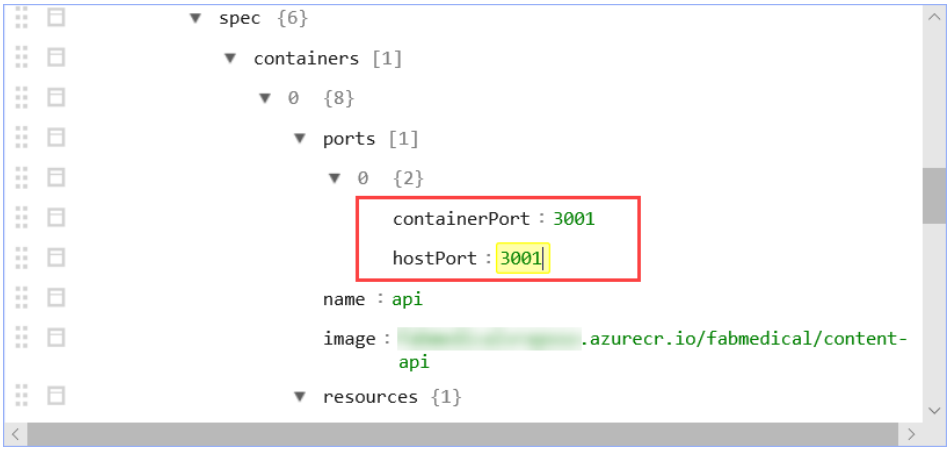
6. Name the new array “ports” and select the menu icon on the empty array row as shown in the screenshot. Select the Append dropdown and select **Object** to insert an object into the array.



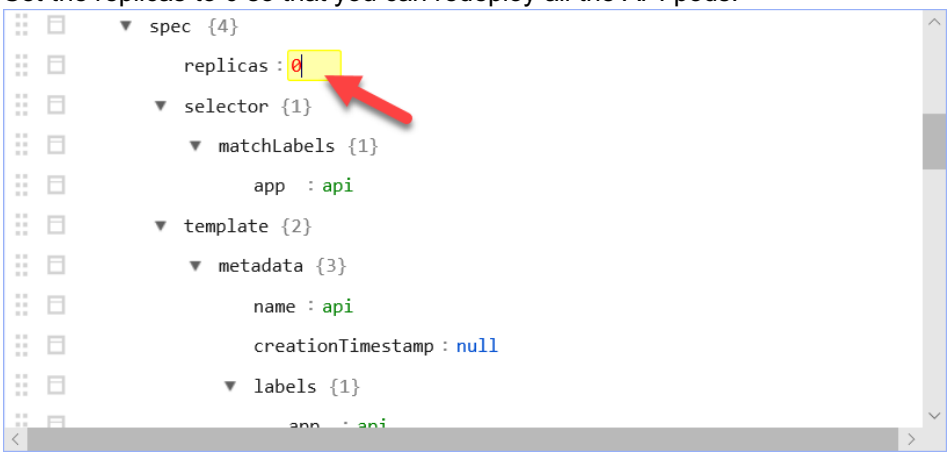
7. Use the menu to add two properties to the port array entry 0. Select the menu icon for the empty object row as shown in the screenshot. Select the Append dropdown and select String to insert a string property into the object:



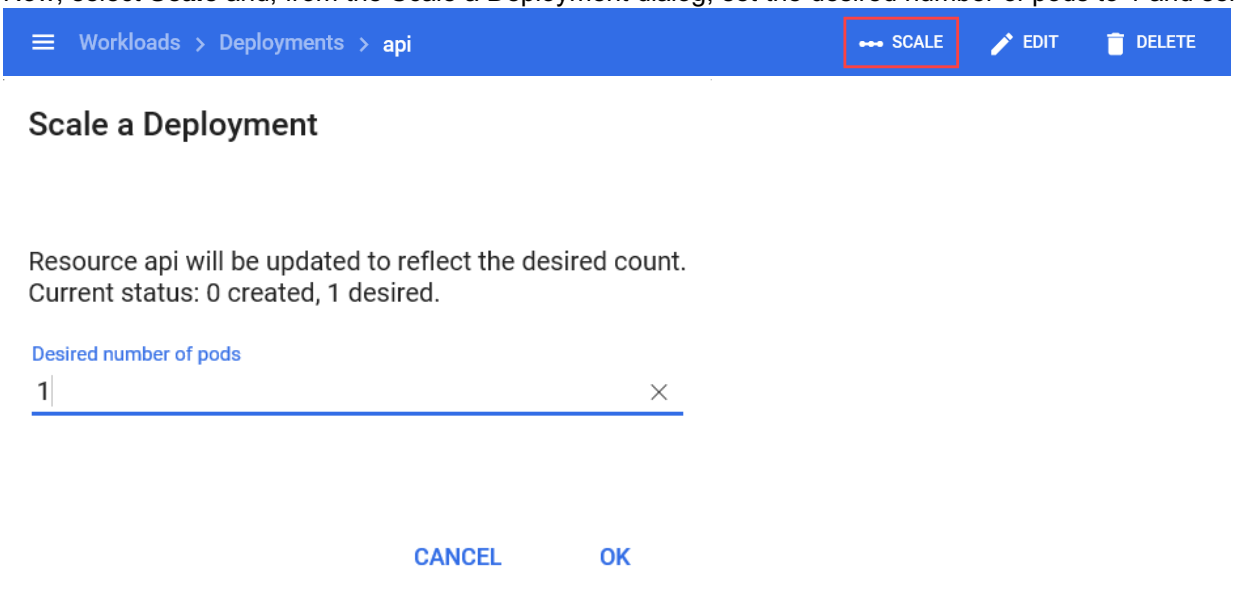
8. Set the following values (repeat the append string procedure to create hostPort):
    a. **containerPort**: 3001
    b. **hostPort**:3001



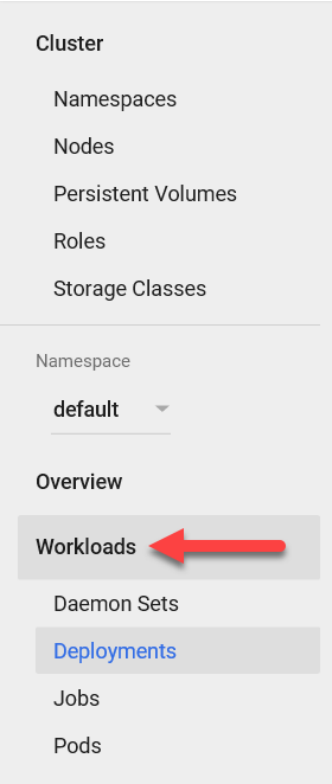
9. Set the replicas to 0 so that you can redeploy all the API pods.



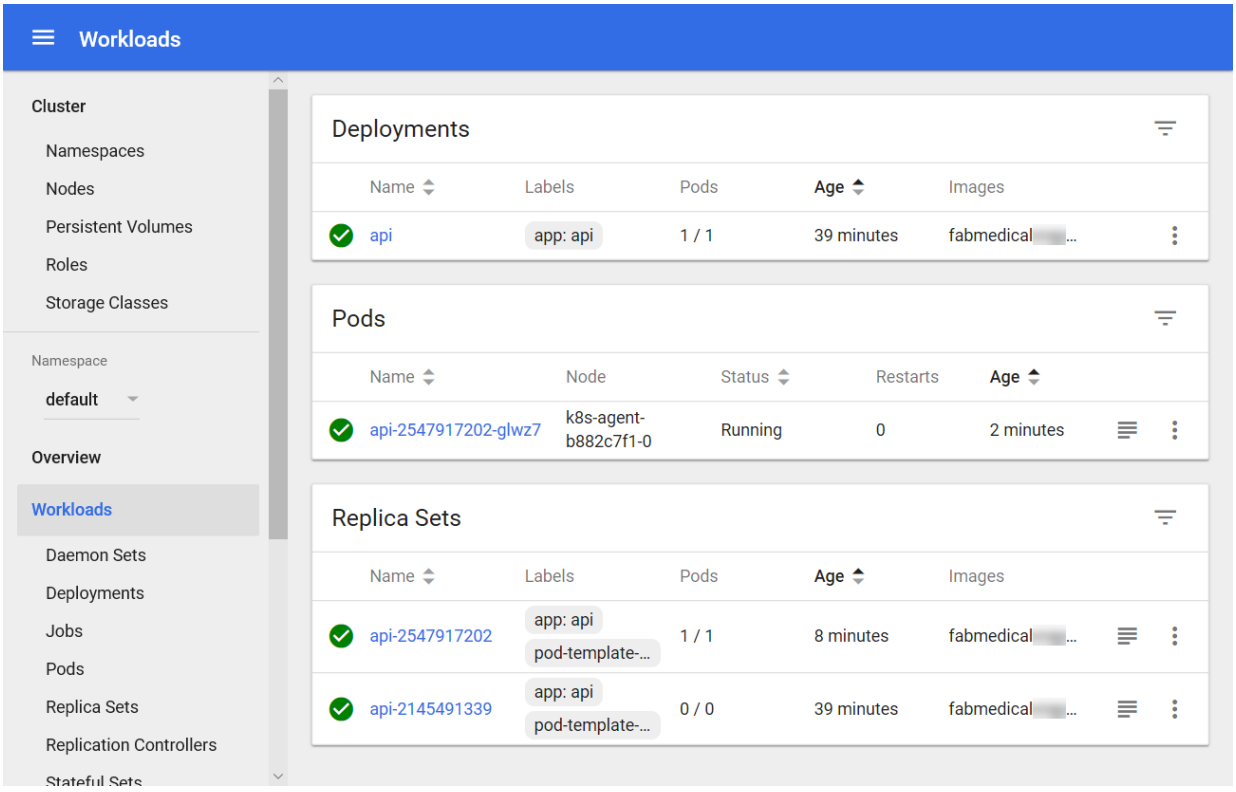
10. Select **Update** to save the changes.
11. Now, select **Scale** and, from the Scale a Deployment dialog, set the desired number of pods to 1 and select **OK**.



12. From the navigation menu, select the **Workloads** view.



13. From the Workloads view, you can see the deployed objects and the replica sets that correspond to your two API deployments After the deployment is successful, it will transition to a Running state as shown in the following screenshot.



Task 3: Deploy a service using Kubernetes REST API

In this task, you will make HTTP requests using kubectl. These requests will deploy the web application using the REST API.

1. Open a new WSL console.
2. Create a text file in this directory called kubernetes-web.yaml using Vim and press the “i" key to go into edit mode.

```
vi kubernetes-web.yaml
<i>
```

3. Copy and paste the following text into the editor.
NOTE: be sure to copy and paste only the contents of the code block carefully to avoid introducing any special characters.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: web
  name: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: web
      name: web
    spec:
      containers:
      - image: [LOGINSERVER].azurecr.io/fabmedical/content-web
        env:
        - name: CONTENT_API_URL
          value: http://api:3001
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 30
          periodSeconds: 20
          timeoutSeconds: 10
          failureThreshold: 3
        imagePullPolicy: Always
        name: web
        ports:
        - containerPort: 80
          hostPort: 80
          protocol: TCP
        resources:
          requests:
            cpu: 1000m
            memory: 128Mi
```



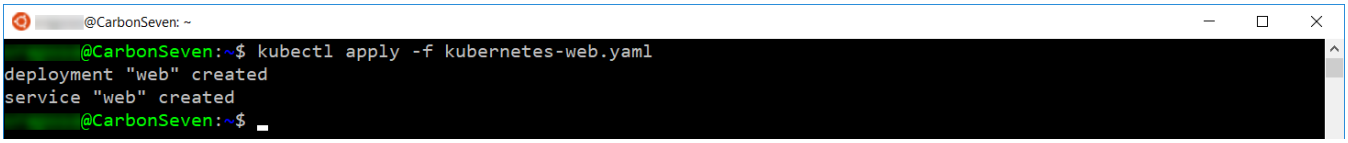
```
        securityContext:
          privileged: false
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: web
  name: web
spec:
  ports:
  - name: web-traffic
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
```

4. Edit this file and update the [LOGINSERVER] entry to match the name of your ACR login server.
5. Press the Escape key and type ":wq" and then press the Enter key to save and close the file.

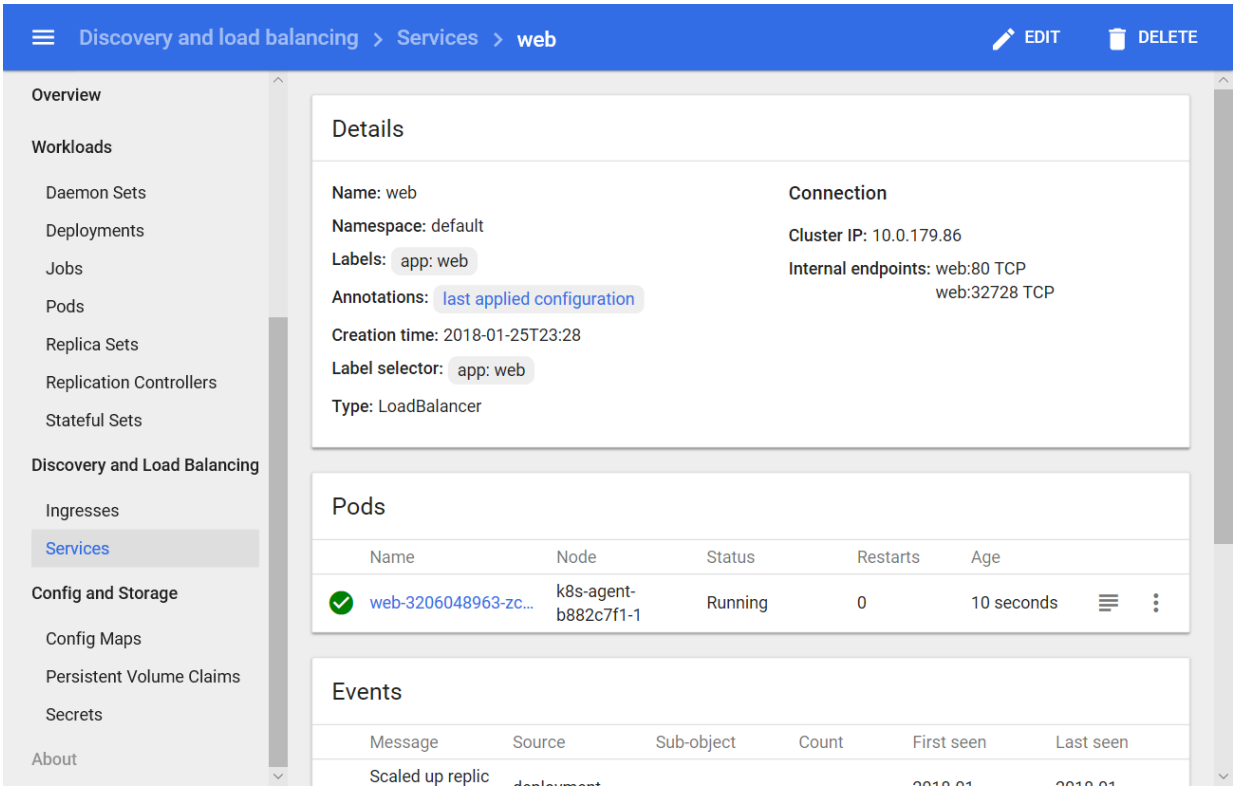
```
<Esc>
:wq
<Enter>
```

6. Type the following command to deploy the application described by the YAML file. You will receive a message indicating the items kubectl has created — a web deployment and a web service.

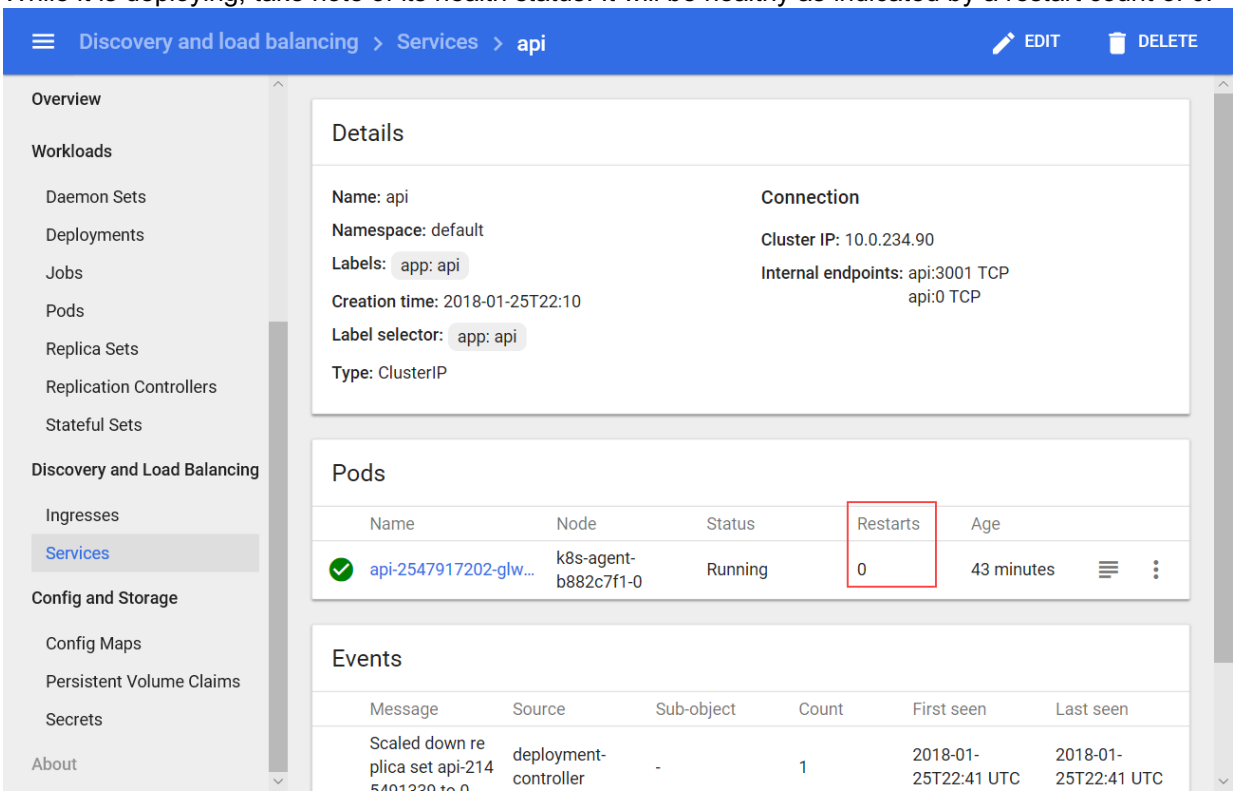
```
kubectl apply -f kubernetes-web.yaml
```



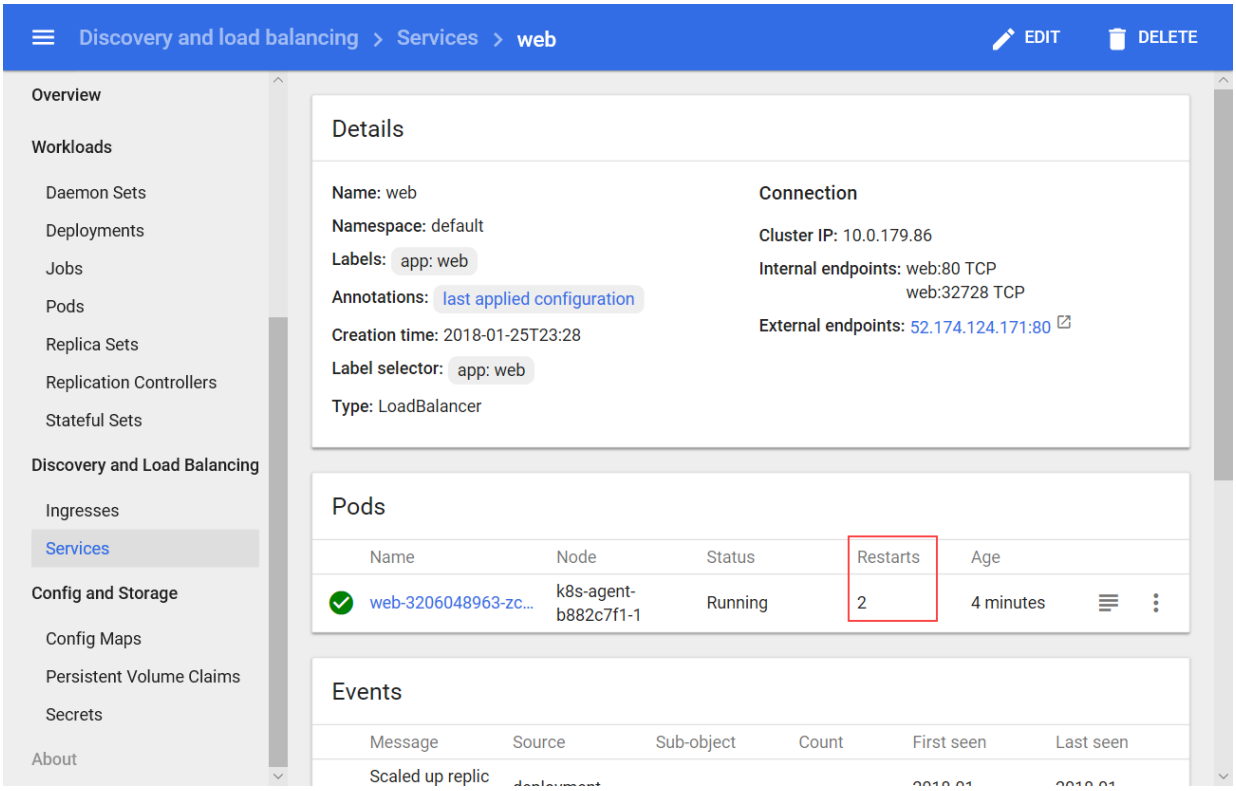
7. Return to the browser where you have the Kubernetes management dashboard open. From the navigation menu, select Services view under Discovery and Load Balancing. From the Services view, select the web service and, from this view, you will see the web service deploying. This deployment can take a few minutes. In the meantime, continue to the next step.



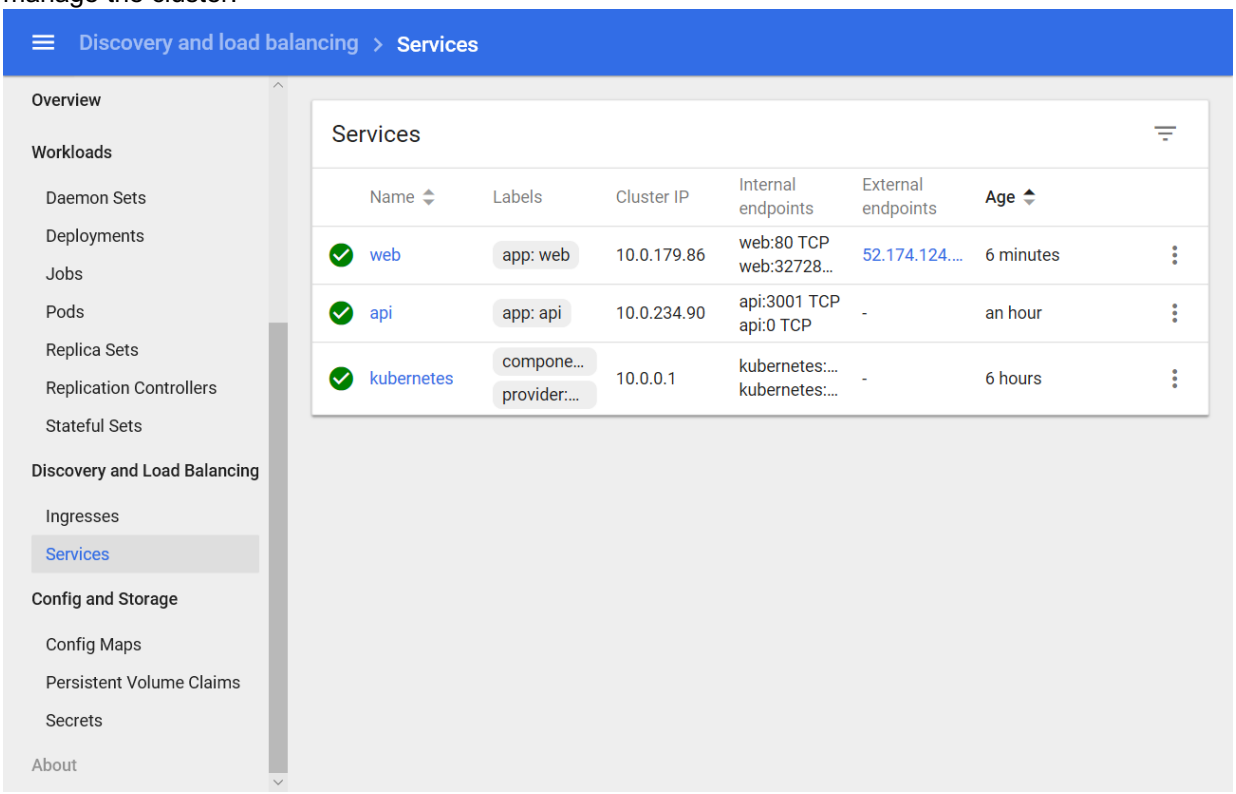
8. From the navigation menu, select Services view under Discovery and Load Balancing. Select the API service. While it is deploying, take note of its health status. It will be healthy as indicated by a restart count of 0.



9. From the navigation menu, select Services view under Discovery and Load Balancing. Select the web service. At some point, if you wait long enough, you will see it is in an unhealthy state (indicated by restarts in the screenshot below). This will only happen after you fail to deploy three consecutive times, so you should not wait for this state. In the meantime, it remains in the Deploying state. You will resolve this in the next task.



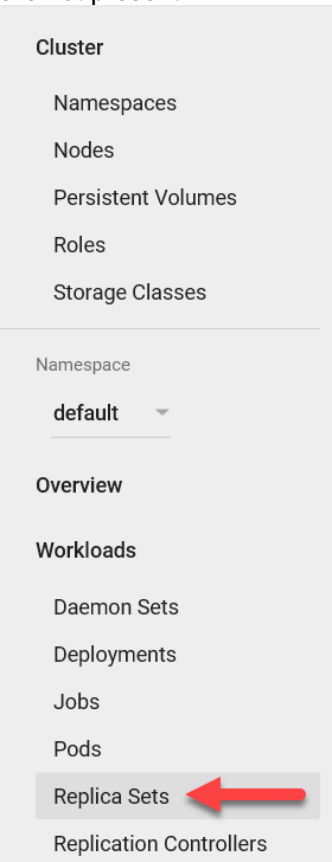
10. From the navigation menu, select Services view under Discovery and Load Balancing. On the Services view, you will see two services deployed. Note that the “kubernetes” API service is always running by default to allow you to manage the cluster.

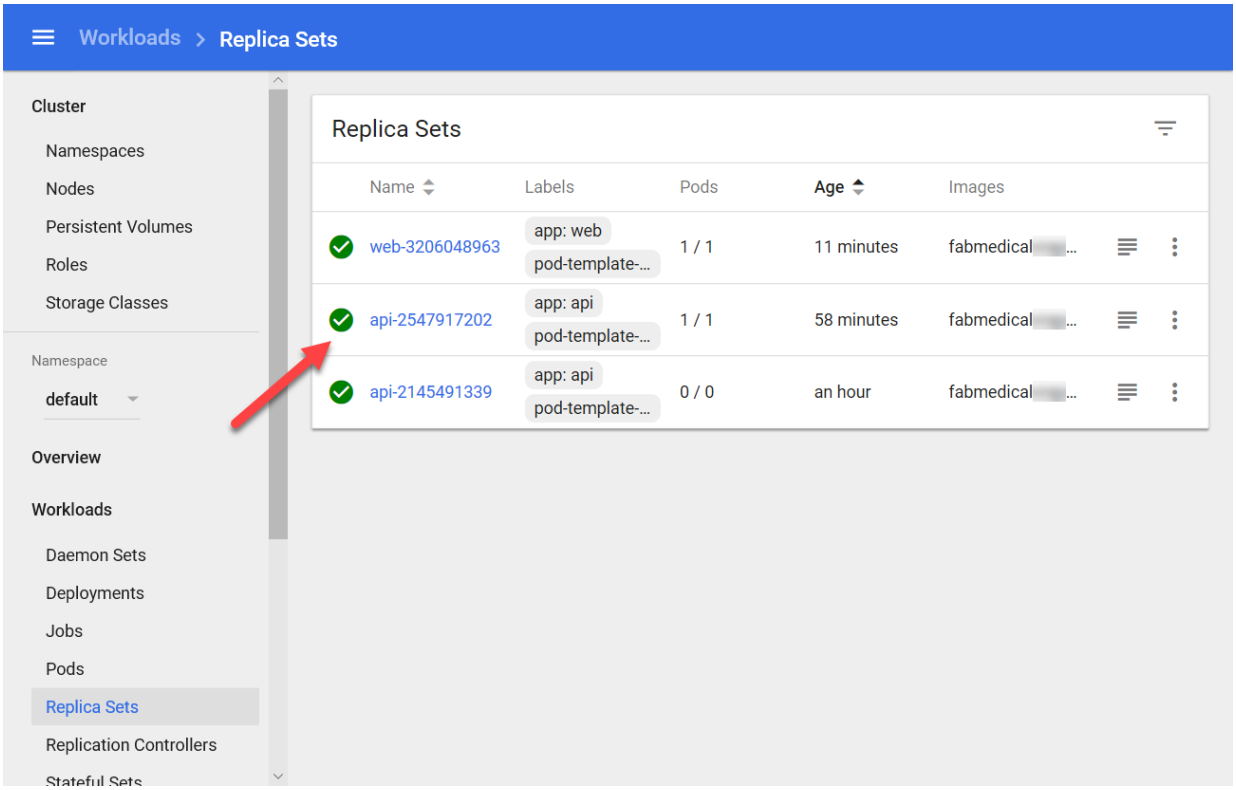


Task 4: Explore service instance logs and resolve an issue

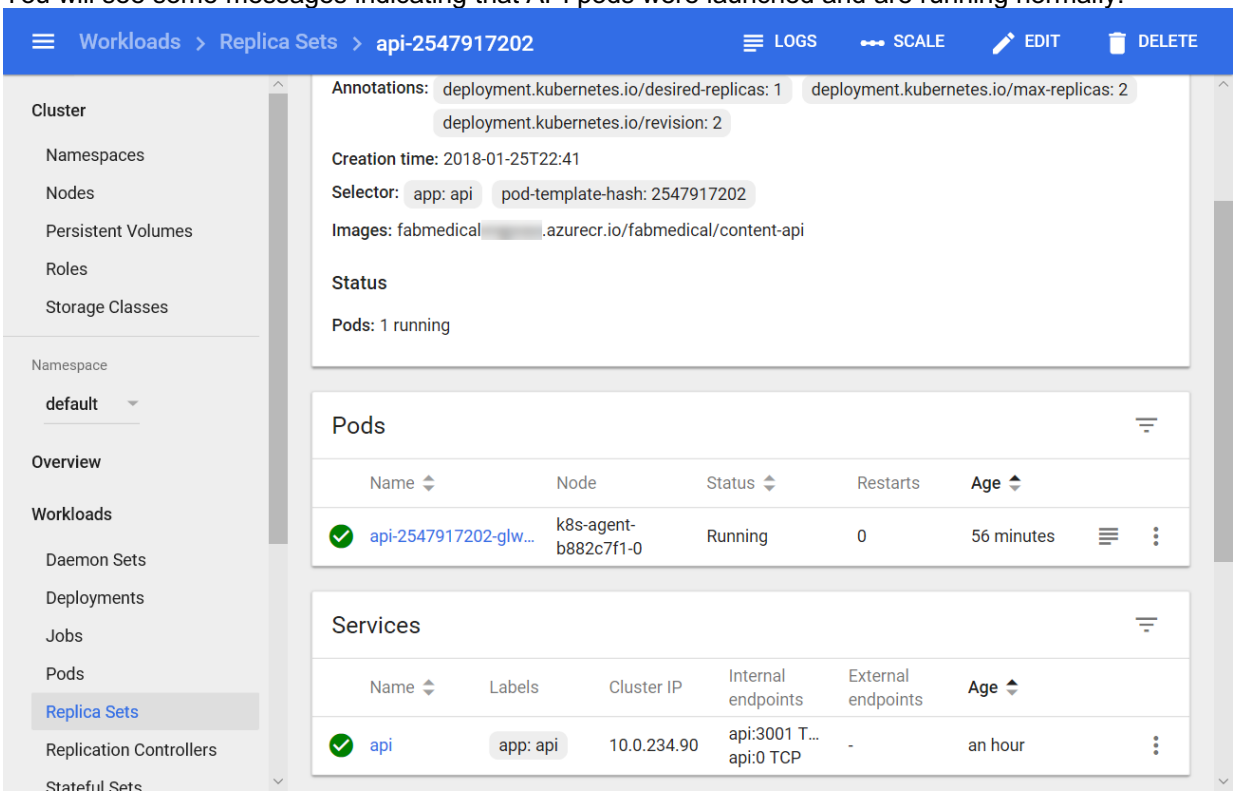
In this task, you will determine why the web service deployment is unhealthy and learn how to explore logs in Kubernetes to troubleshoot issues. You will then fix the problem and redeploy the service.

1. From the navigation menu, select the Replica Sets view under Workloads. From the Replica Sets view, select the api replica set from the list in the Replica Sets section of the view.
   NOTE: the CPU usage and Memory usage graphs may not appear as it depends on the default configuration of the specific version of Kubernetes used by the template when you create the cluster. You can ignore if the graphs are not present.

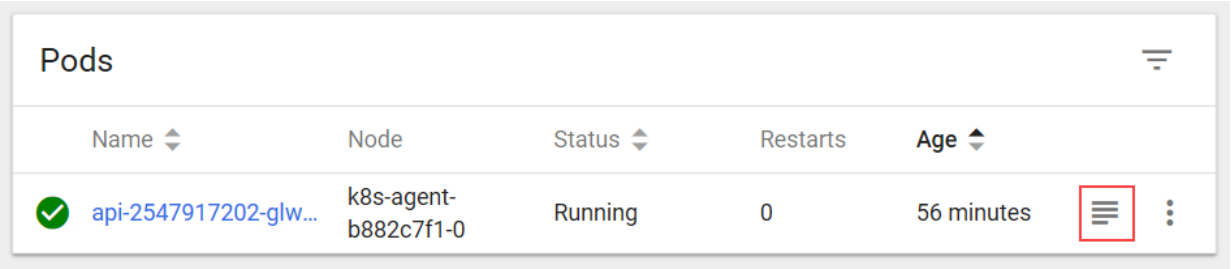




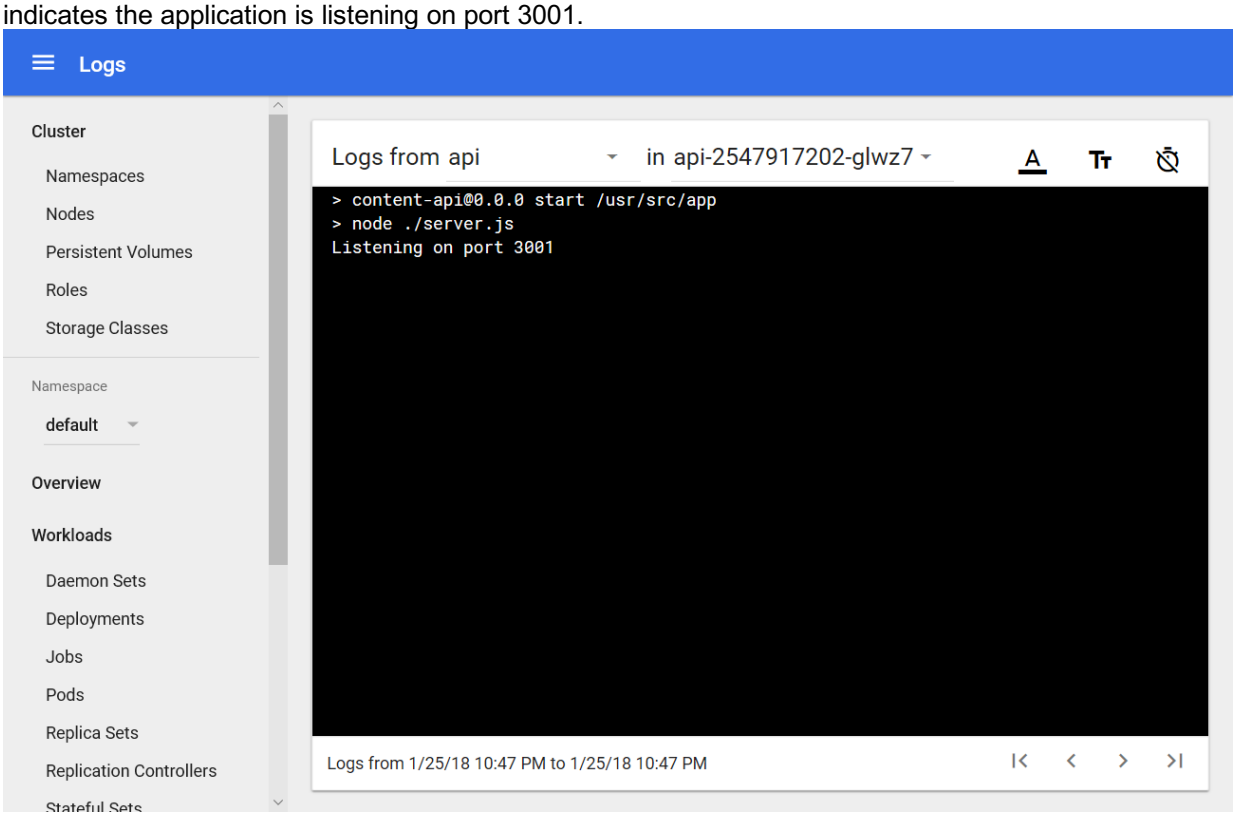
2. You will see some messages indicating that API pods were launched and are running normally.



3. Select the logs icon to the right of the pod.

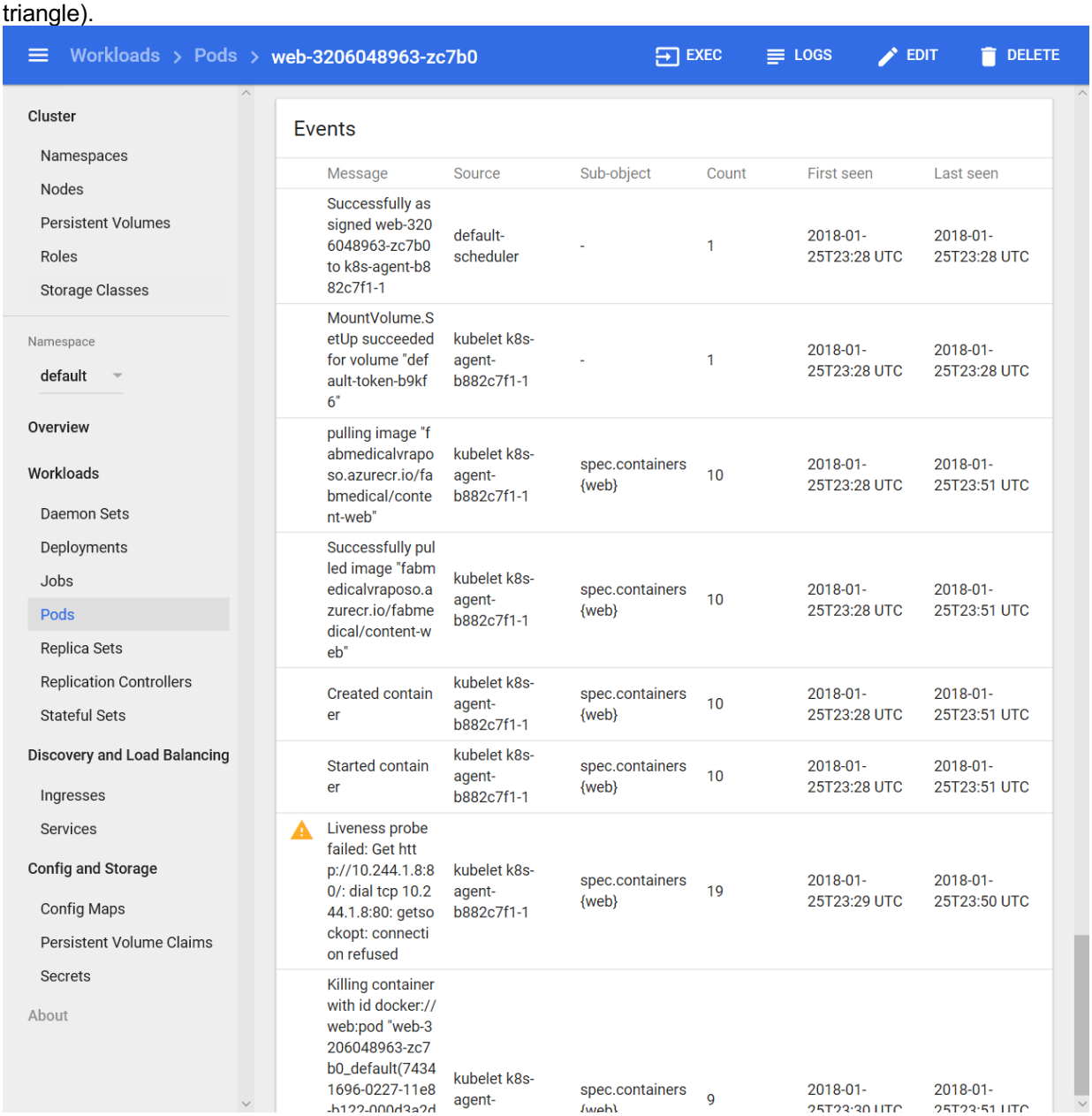


4. You'll see the console output (stdout) from the initialization of the API application container. The end of the output indicates the application is listening on port 3001.



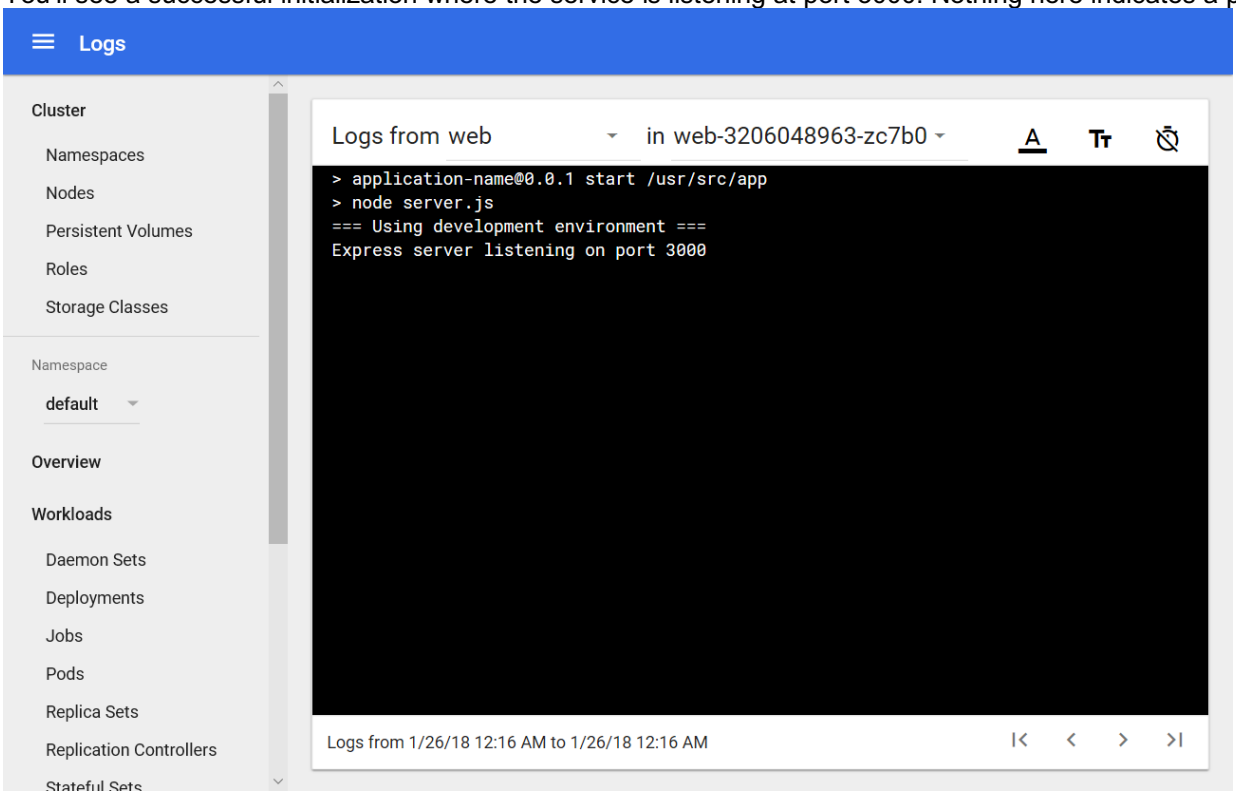
5. From the navigation menu, select Services view under Discovery and Load Balancing. Select the web service. You should see a task in an unhealthy state (indicated by the number of restarts).

6. From the navigation menu, select the Replica Sets view under Workloads. Select the web Replica Set, then select the web Pod. In the Events list, you will see warnings regarding failed health checks (indicated by orange warning triangle).

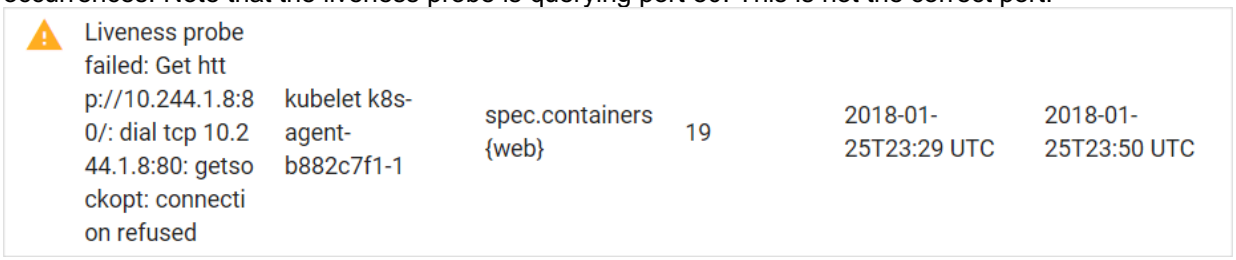


7. In the Pods section, select the logs icon for the pod to show the service's console output.

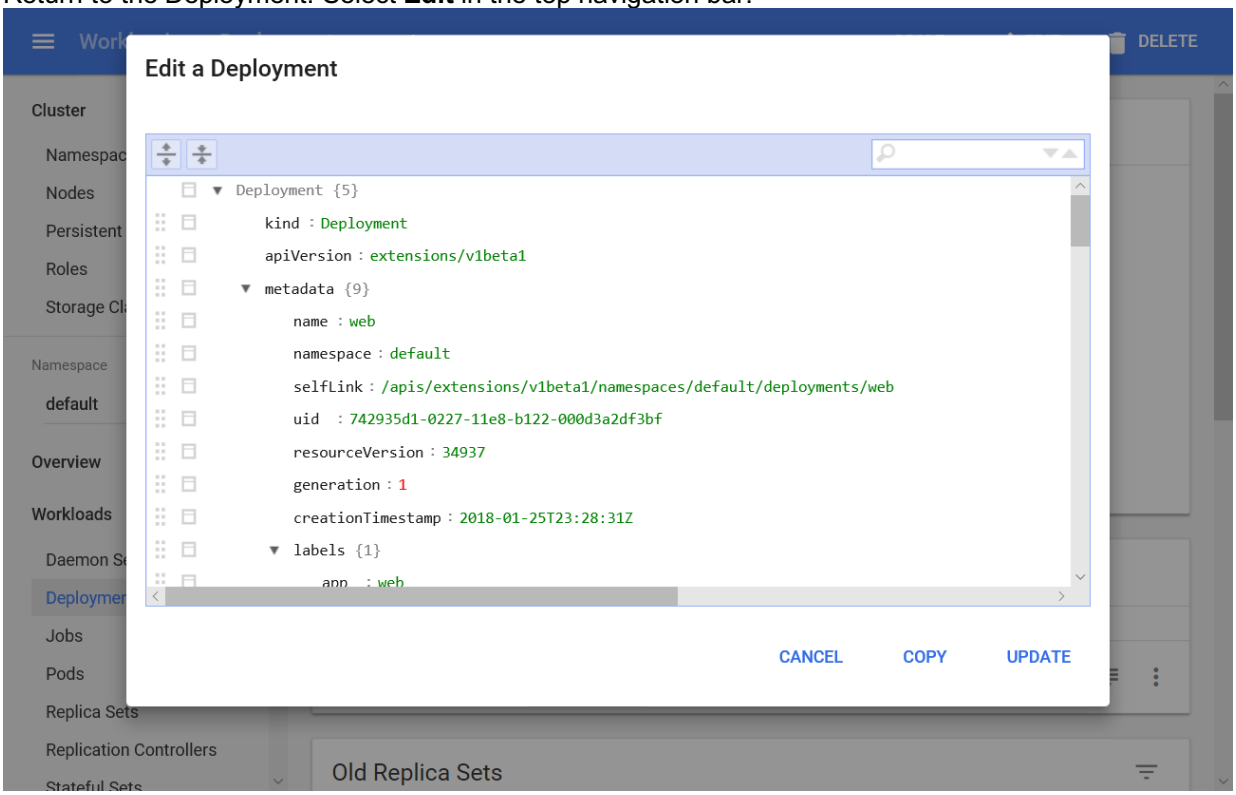
8. You'll see a successful initialization where the service is listening at port 3000. Nothing here indicates a problem.



9. Return to the web pod view to reexamine the events shown. Note the high number of failed health check occurrences. Note that the liveness probe is querying port 80. This is not the correct port.

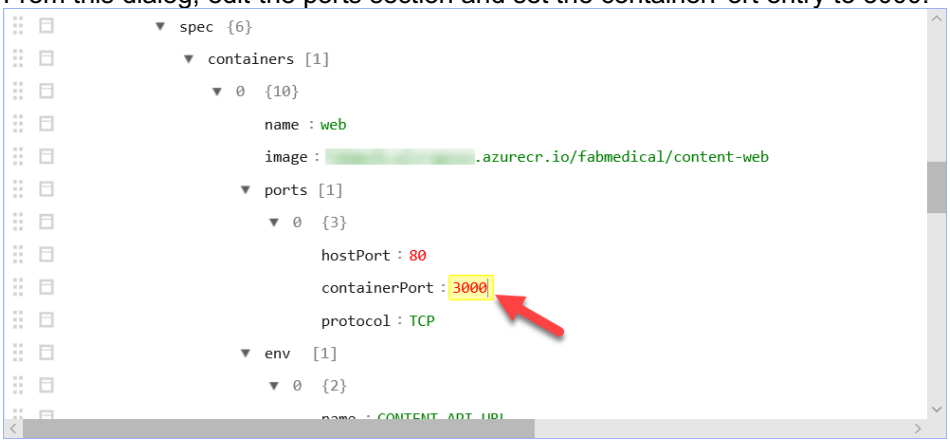


10. Return to the Deployment. Select **Edit** in the top navigation bar.

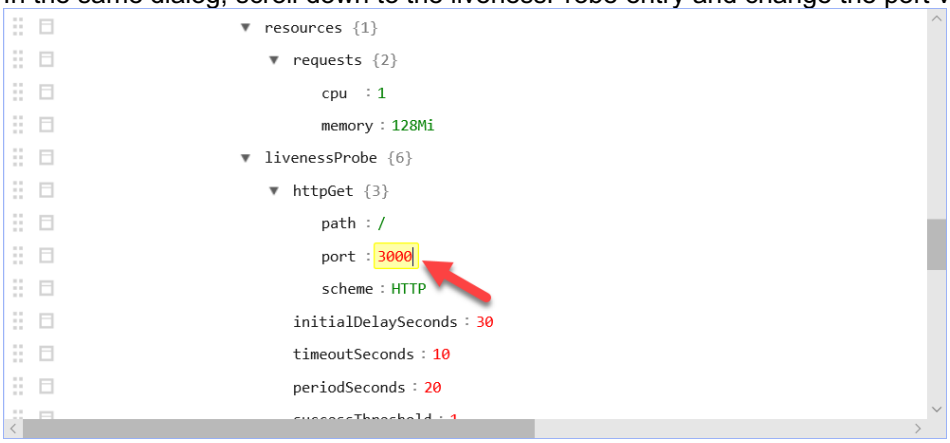


11. Scroll down to view the Port Definitions used when you published the web application. Note the container port is incorrectly set to 80, it should be 3000. Therefore, the health check is failing. The public host port is set correctly to port 80.

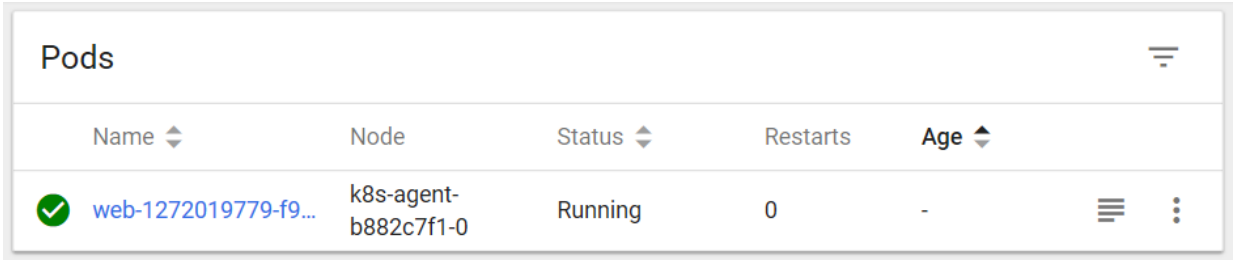
12. From this dialog, edit the ports section and set the containerPort entry to 3000.



13. In the same dialog, scroll down to the livenessProbe entry and change the port value to 3000.



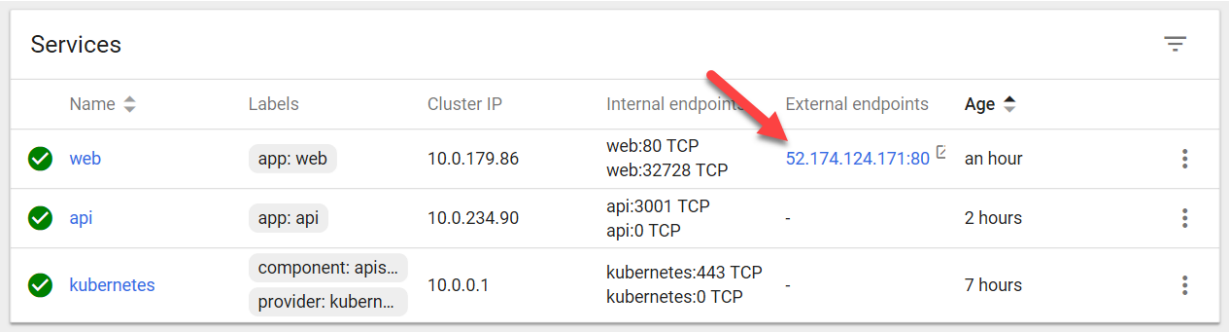
14. Select **Update** to save the changes.
15. From the navigation menu, select Workloads >Replica Sets. You'll see that the web pod is replaced. When it is fully initialized, the pod should run normally with no restarts.
16. Select the web Replica Set. From the web Replica Set view, ensure the controller shows one pod and has 0 restarts.



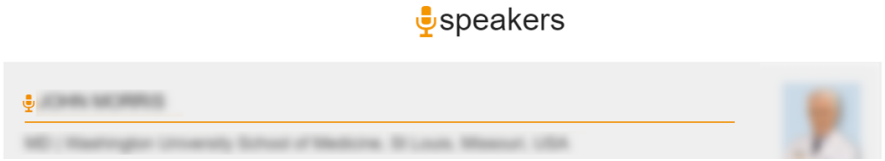
Task 5: Test the application in a browser

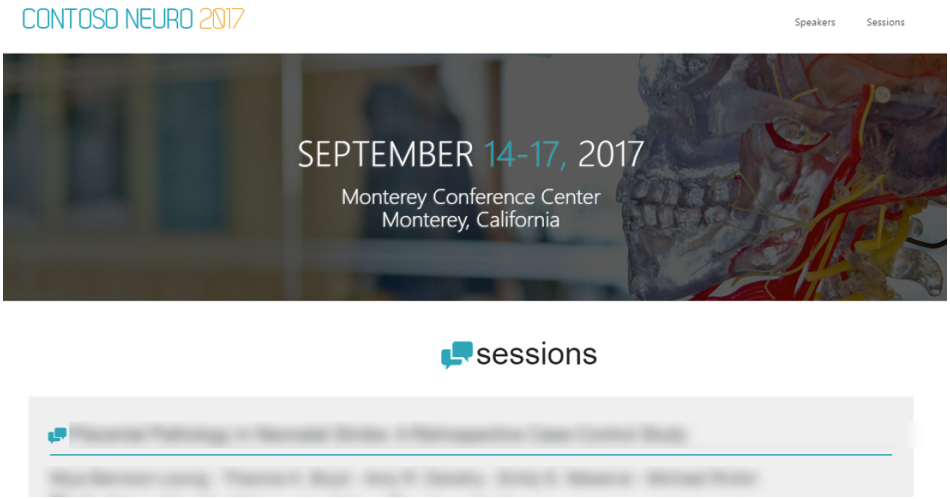
In this task, you will verify that you can browse to the web service you have deployed and view the speaker and content information exposed by the API service.

1. From the Kubernetes management dashboard, in the navigation menu, select the Services view under Discovery and Load Balancing.
2. In the list of services, locate the external endpoint for the web service and select this hyperlink to launch the application.



3. You will see the web application in your browser and be able to select the Speakers and Sessions links to view those pages without errors. The lack of errors means that the web application is correctly calling the API service to show the details on each of those pages.





Exercise 4: Scale the application and test HA

Duration: 20 minutes

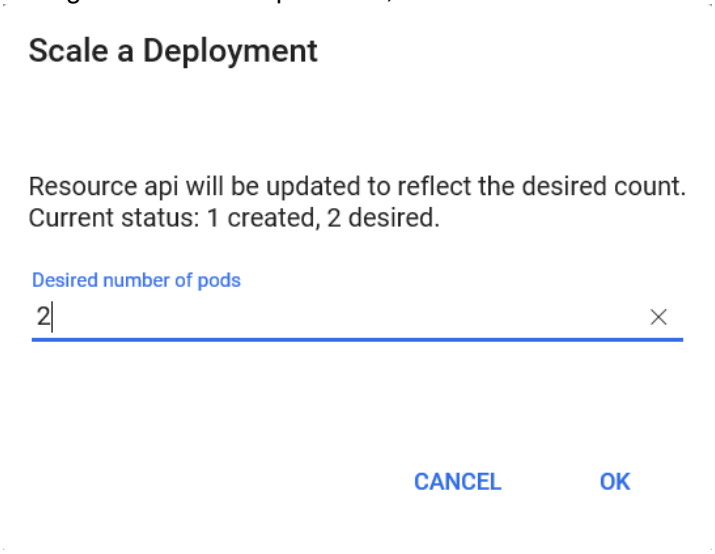
At this point you have deployed a single instance of the web and API service containers. In this exercise, you will increase the number of container instances for the web service and scale the front end on the existing cluster.

Task 1: Increase service instances from the Kubernetes dashboard

In this task, you will increase the number of instances for the API deployment in the Kubernetes management dashboard. While it is deploying, you will observe the changing status.

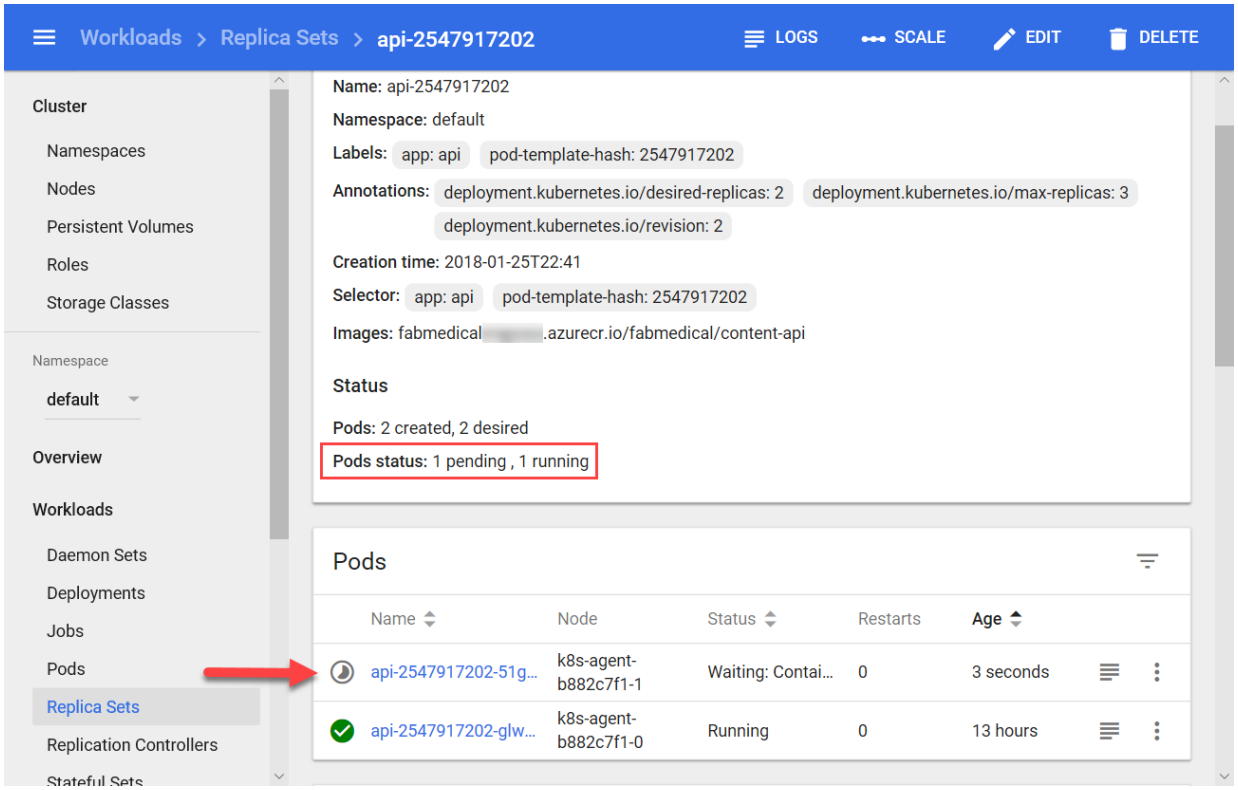
1. From the navigation menu, select Workloads>Deployments and select the API deployment.
2. Select SCALE.

3. Change the number of pods to 2, and then select **OK**.

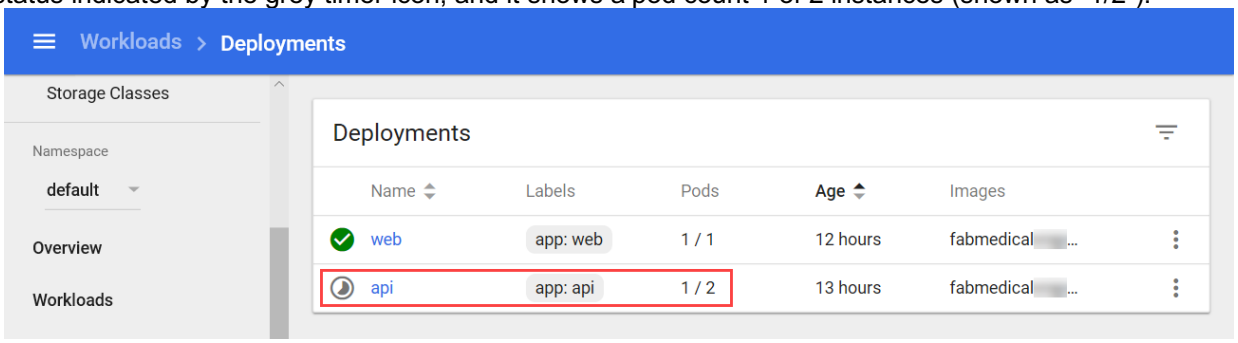


NOTE: If the deployment completes quickly, you may not see the deployment Waiting states in the dashboard as described in the following steps.

4. From the Replica Set view for the API, you'll see it is now deploying and that there is one healthy instance and one pending instance.



5. From the navigation menu, select Deployments and from the list and note that the api service has a pending status indicated by the grey timer icon, and it shows a pod count 1 of 2 instances (shown as “1/2”).



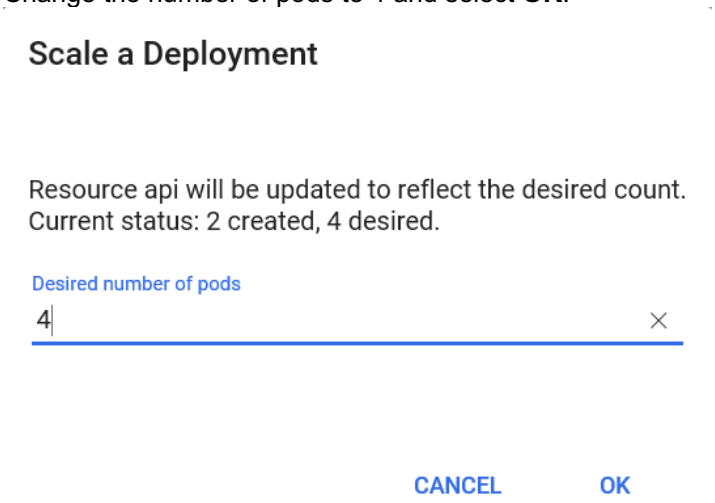
6. From the Navigation menu, select Workloads. From this view note that the health overview in the right panel of this view. You’ll see the following:
    a. One deployment and one replica set are each healthy for the api service.
    b. One replica set is healthy for the web service.
    c. Three pods are healthy.
7. Navigate to the web application from the browser again. The application should still work without errors as you navigate to Speakers and Sessions pages.
    a. Navigate to the /stats.html page. You’ll see information about the environment including:
        i. webTaskId: the task identifier for the web service instance
        ii. taskId: the task identifier for the API service instance
        iii. hostName: the hostname identifier for the API service instance
        iv. pid: the process id for the API service instance
        v. mem: some memory indicators returned from the API service instance
        vi. counters: counters for the service itself, as returned by the API service instance
        vii. uptime: the up time for the API service
    b. Refresh the page in the browser, and you can see the hostName change between the two API service instances. The letters after “api-{number}-” in the hostname will change.

Task 2: Increase service instances beyond available resources

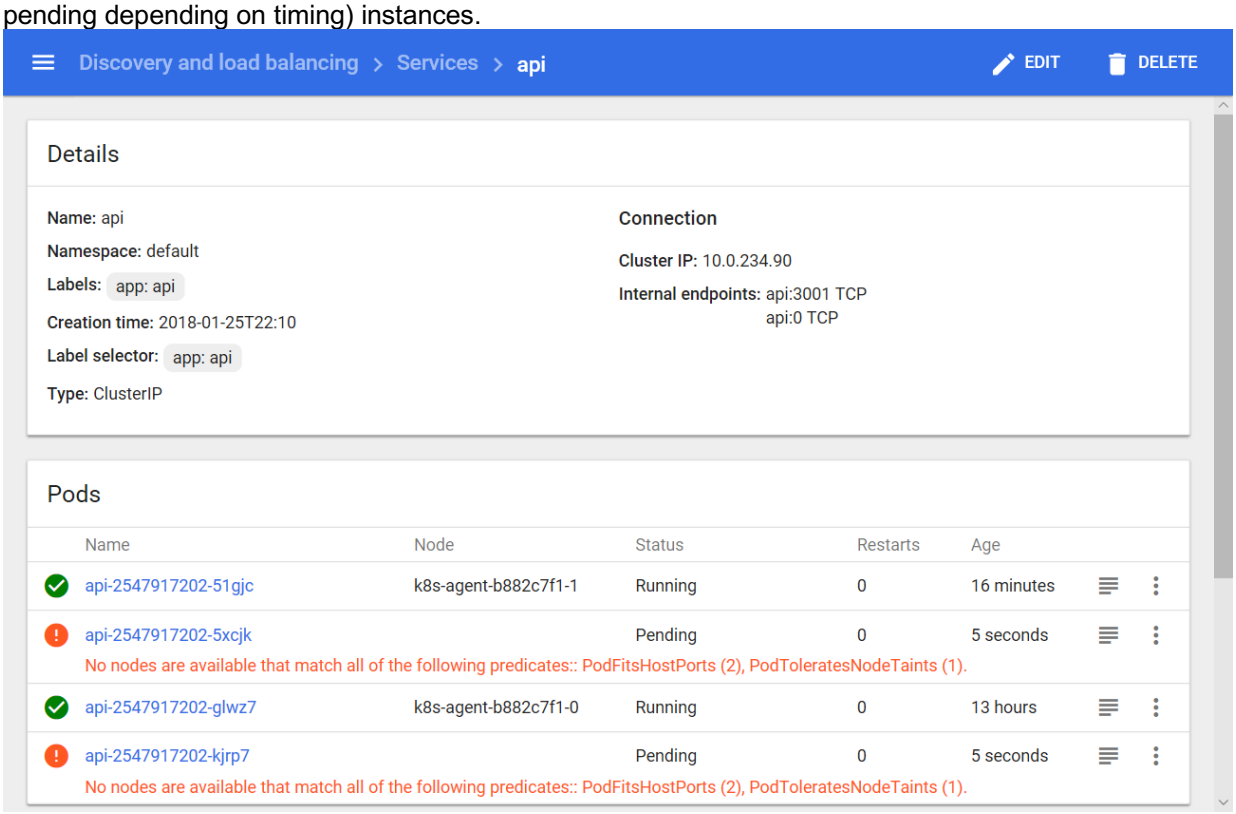
In this task, you will try to increase the number of instances for the API service container beyond available resources in the cluster. You will observe how Kubernetes handles this condition and correct the problem.

1. From the navigation menu, select Deployments. From this view, select the API deployment.
2. From the API deployment view, select **Scale**.

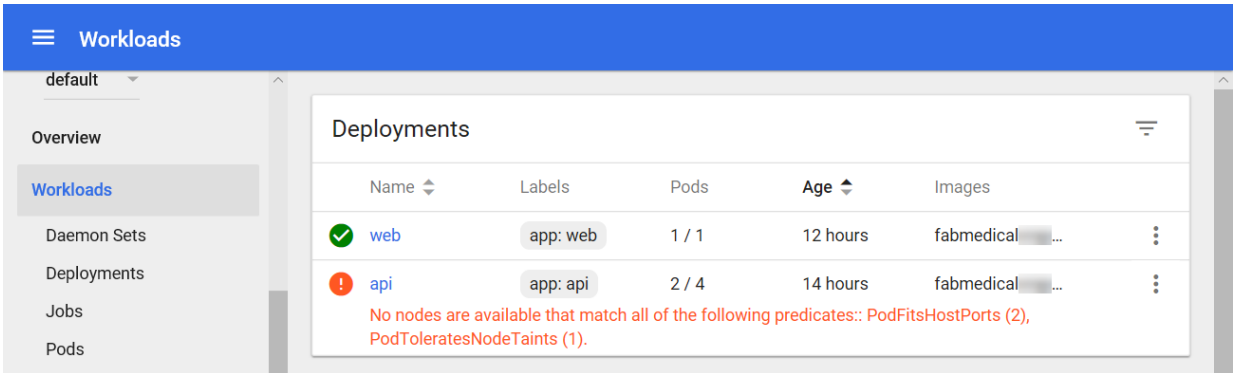
3. Change the number of pods to 4 and select **OK**.



4. From the navigation menu, select Services view under Discovery and Load Balancing. Select the api service from the Services list. From the api service view, you'll see it has two healthy instances and two unhealthy (or possibly pending depending on timing) instances.

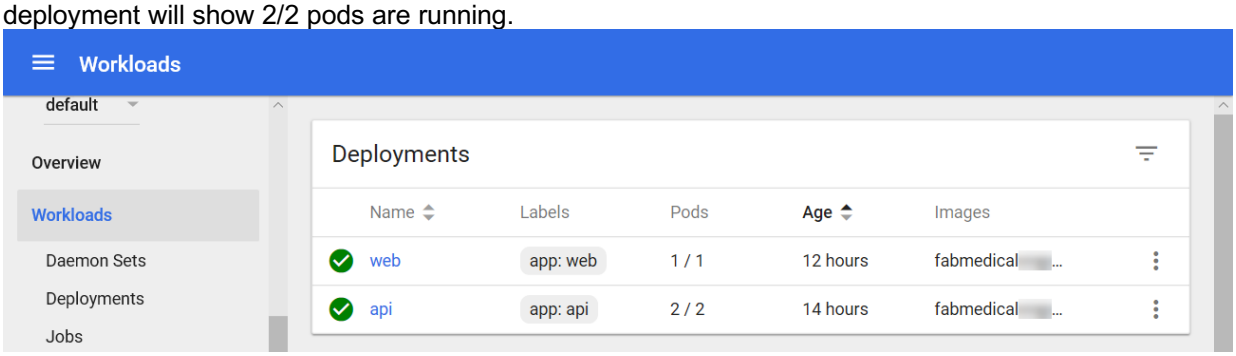


5. After a few minutes, select Workloads from the navigation menu. From this view, you should see an alert reported for the api deployment.



NOTE: This message indicates that there weren't enough available resources to match the requirements for a new pod instance. In this case, this is because the instance requires port 3001, and since there are only 2 nodes available in the cluster, only two api instances can be scheduled. The third and fourth pod instances will wait for a new node to be available that can run another instance using that port.

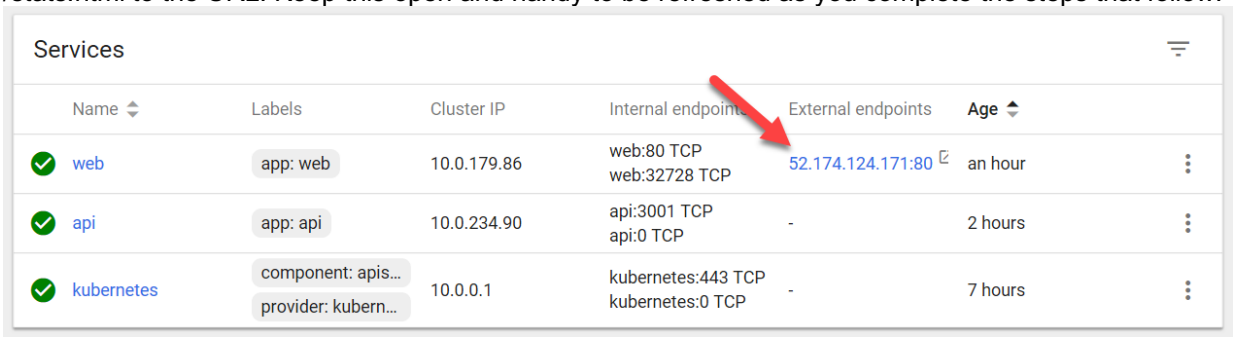
6. Reduce the number of requested pods to 2 using the Scale button.
7. Almost immediately, the warning message from the Workloads dashboard should disappear, and the API deployment will show 2/2 pods are running.



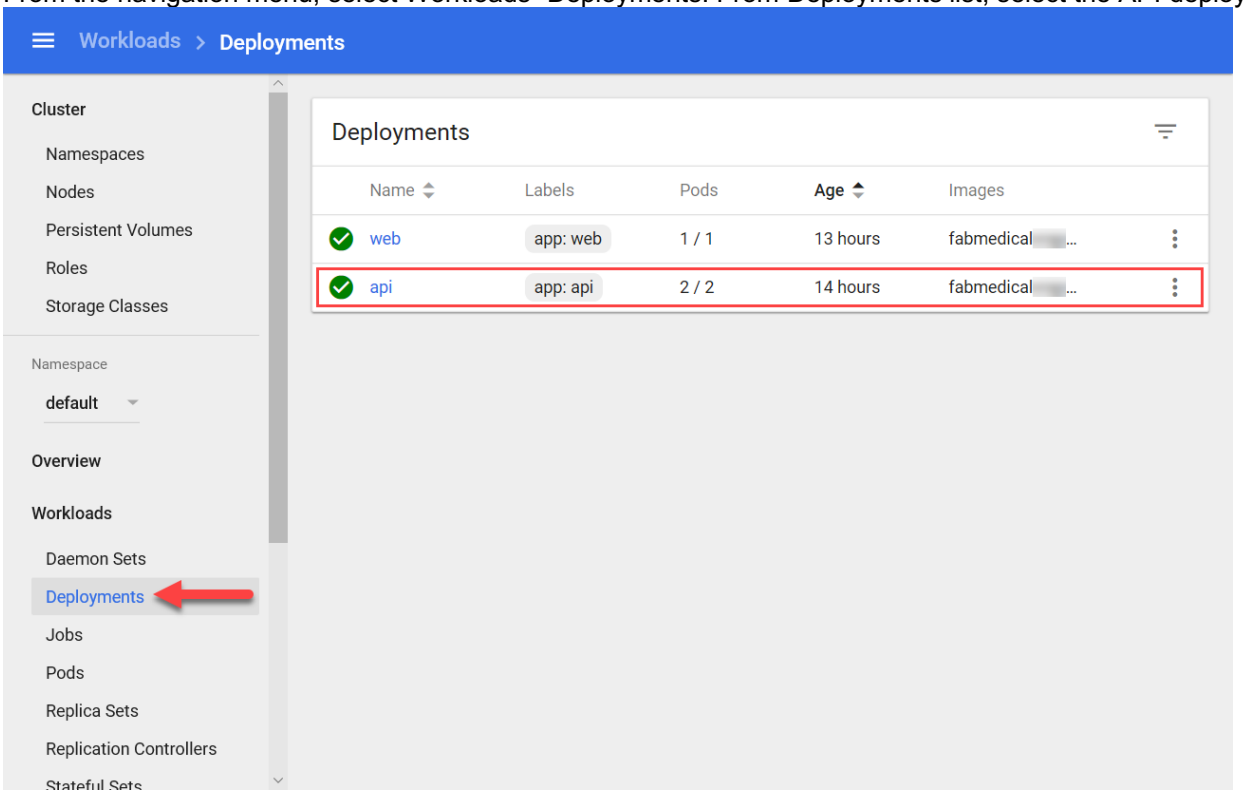
Task 3: Restart containers and test HA

In this task, you will restart containers and validate that the restart does not impact the running service.

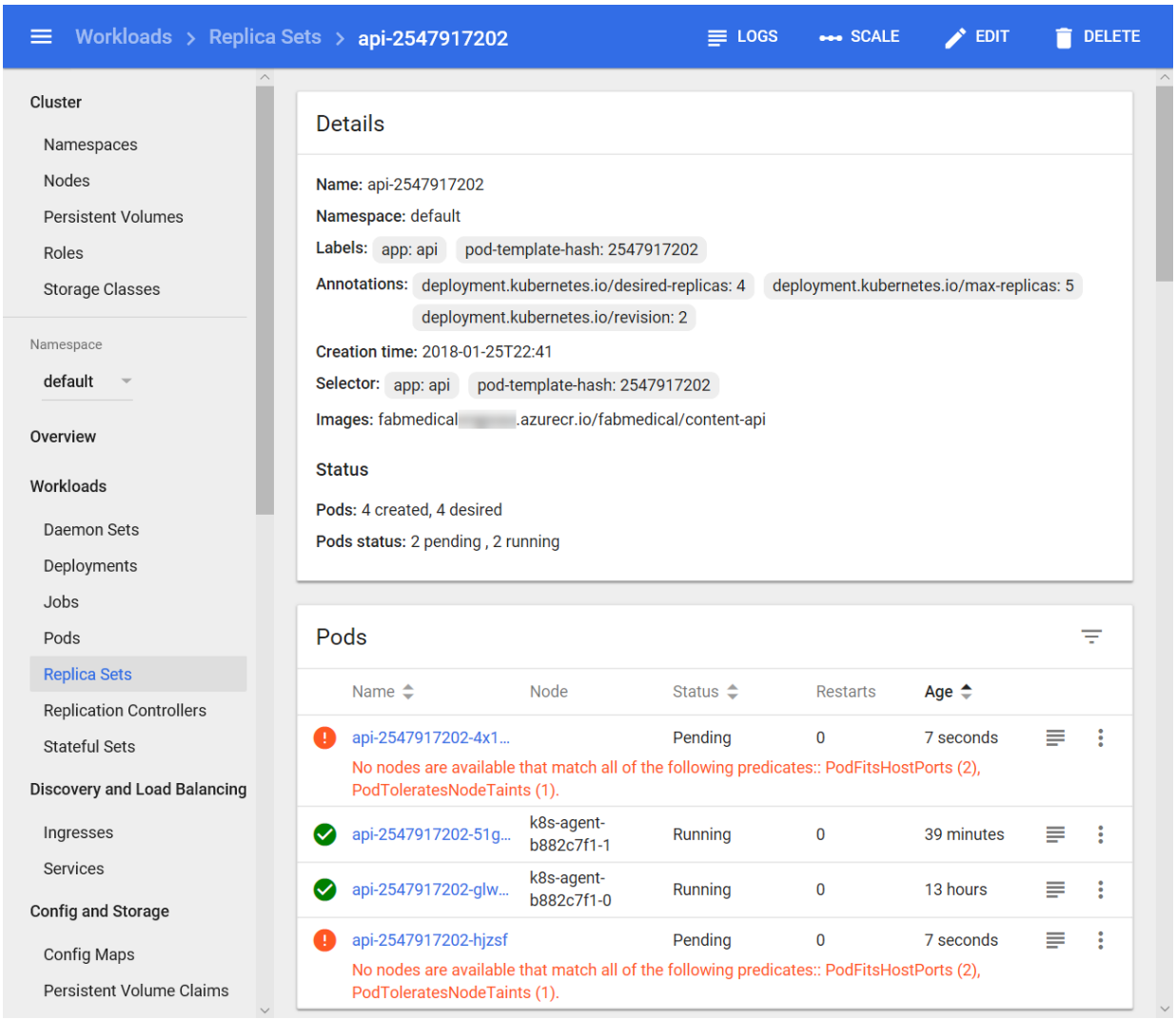
1. From the navigation menu on the left, select Services view under Discovery and Load Balancing. From the Services list, select the external endpoint hyperlink for the web service and visit the stats page by adding /stats.html to the URL. Keep this open and handy to be refreshed as you complete the steps that follow.



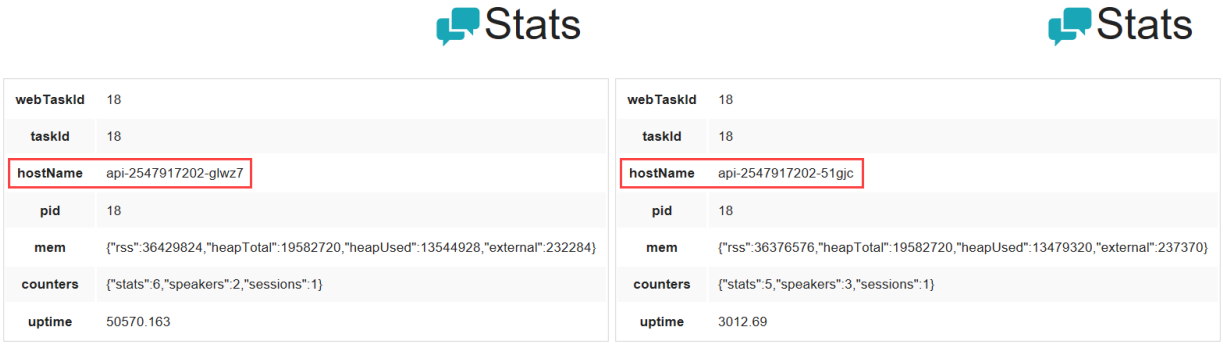
2. From the navigation menu, select Workloads>Deployments. From Deployments list, select the API deployment.



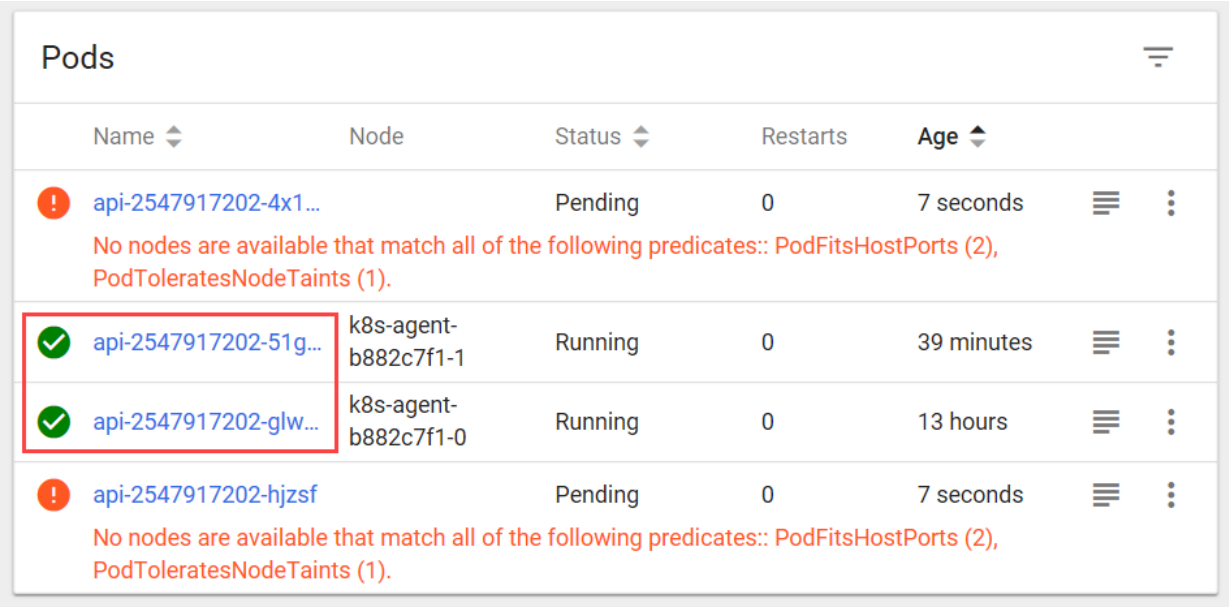
3. From the API deployment view, select **Scale** and from the dialog presented, enter 4 for the desired number of pods. Select **OK**.
4. From the navigation menu, select Workloads>Replica Sets. Select the api replica set and, from the Replica Set view, you will see that two pods cannot deploy.



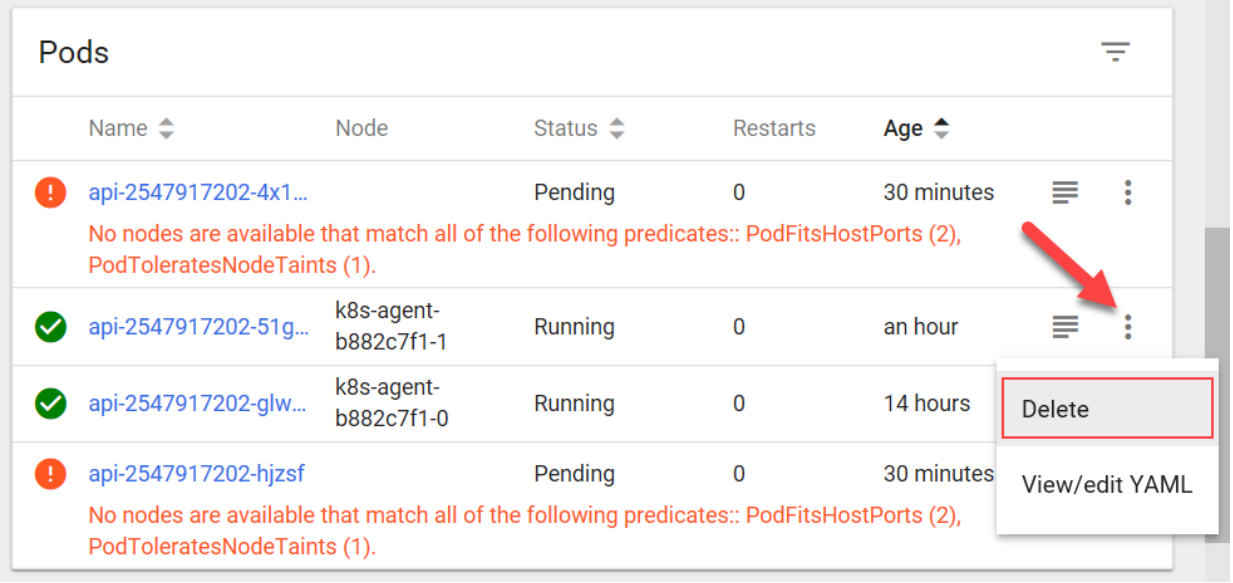
5. Return to the browser tab with the web application stats page loaded. Refresh the page over and over. You will not see any errors, but you will see the api host name change between the two api pod instances periodically. The task id and pid might also change between the two api pod instances.



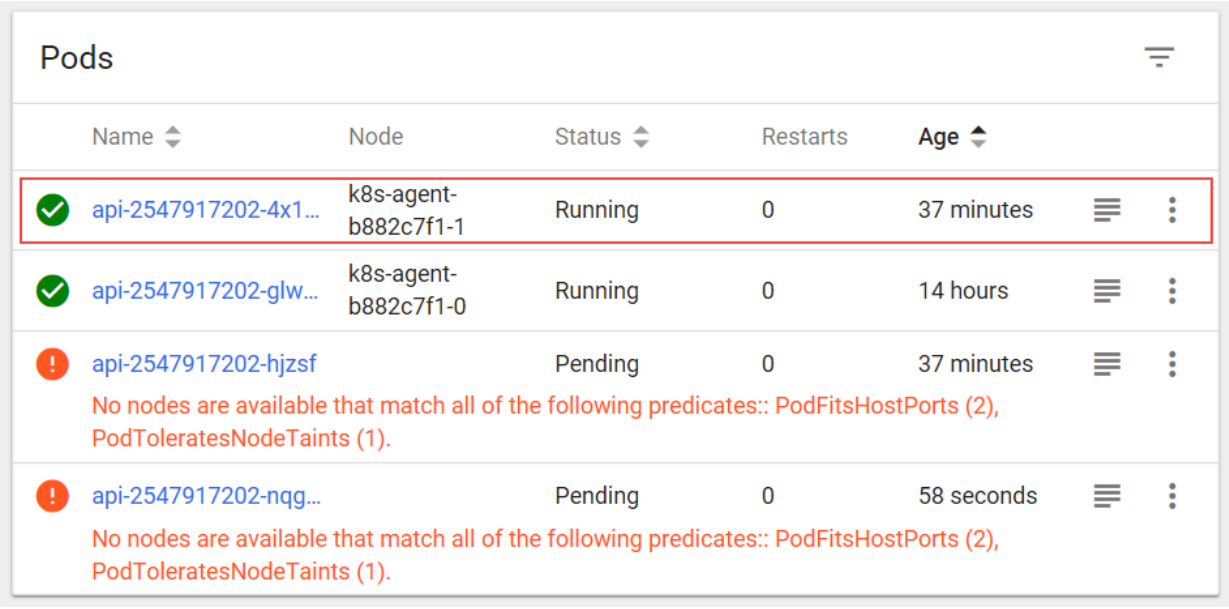
6. After refreshing enough times to see that the hostName value is changing, and the service remains healthy, return to the Replica Sets view for the API. From the navigation menu, select Replica Sets under Workloads and select the API replica set.
7. From this view, take note that the hostName value shown in the web application stats page matches the pod names for the pods that are running.



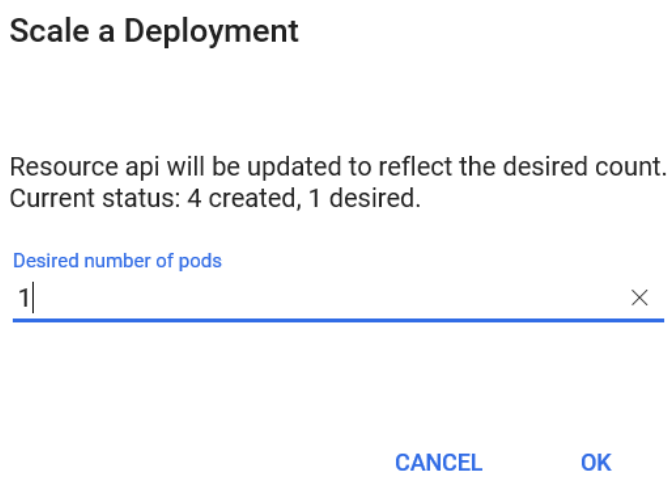
8. Note the remaining pods are still pending, since there are not enough port resources available to launch another instance. Make some room by deleting a running instance. Select the context menu and choose Delete for one of the healthy pods.



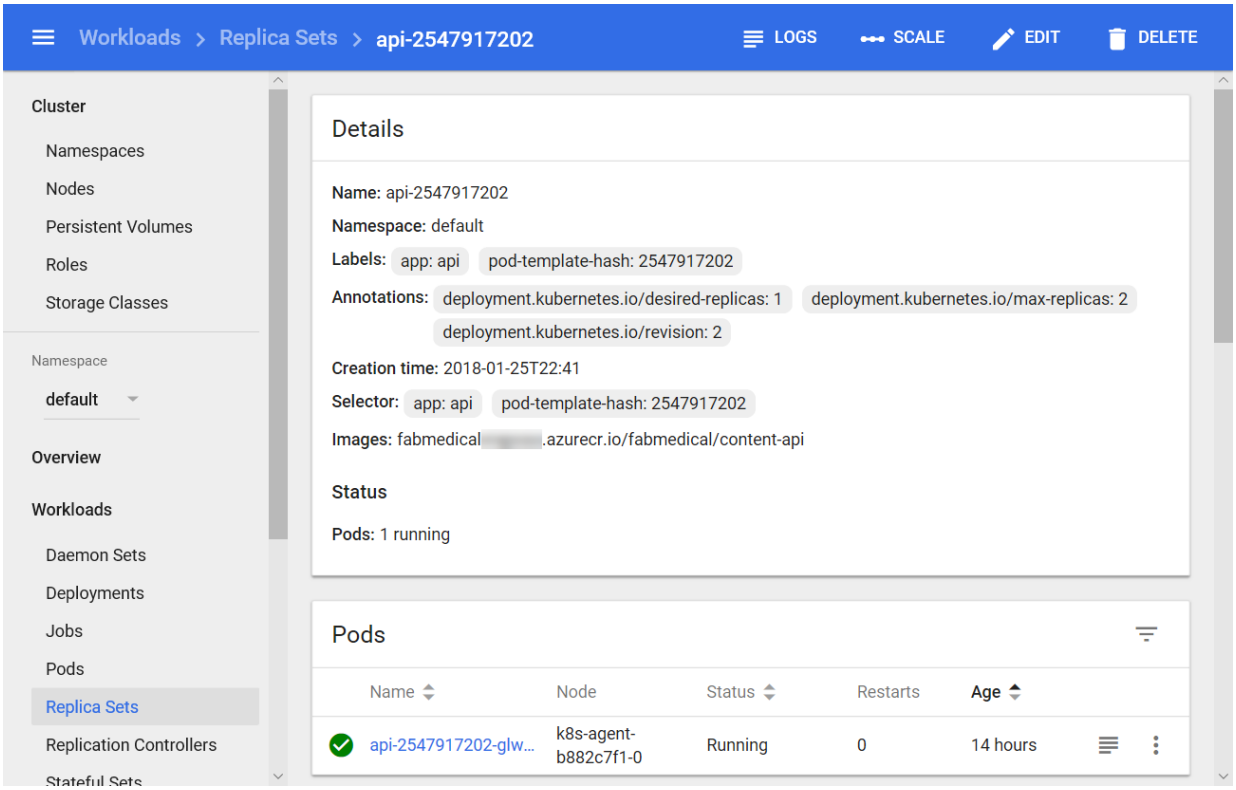
9. Once the running instance is gone, Kubernetes will be able to launch one of the pending instances. However, because you set the desired size of the deploy to 4, Kubernetes will add a new pending instance. Removing a running instance allowed a pending instance to start, but in the end, the number of pending and running instances is unchanged.



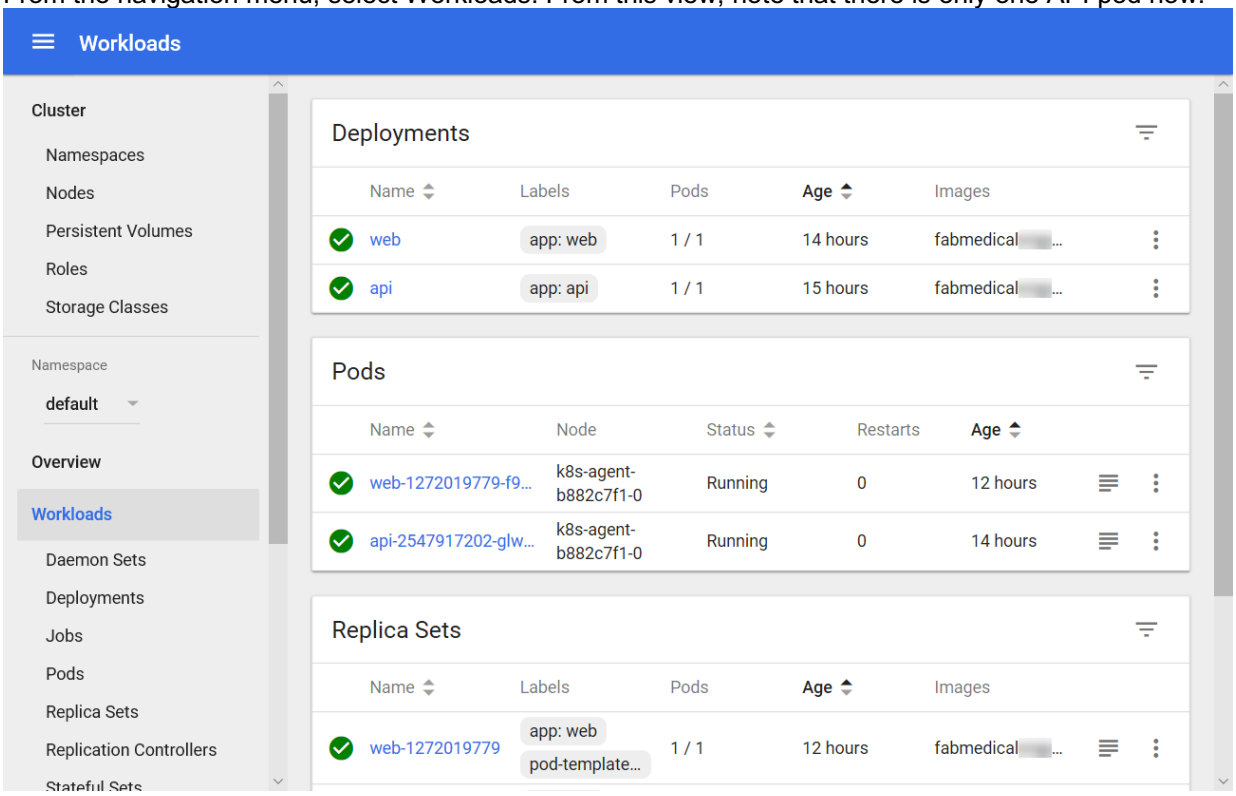
10. From the navigation menu, select Deployments under Workloads. From the view's Deployments list select the API deployment.
11. From the API Deployment view, select Scale and enter 1 as the desired number of pods. Select OK.



12. Return to the web site's stats.html page in the browser, and refresh while this is scaling down. You'll notice that only one API host name shows up, even though you may still see several running pods in the API replica set view. Even though several pods are running, Kubernetes will no longer send traffic to the pods it has selected to scale down. In a few moments, only one pod will show in the API replica set view.



13. From the navigation menu, select Workloads. From this view, note that there is only one API pod now.



Exercise 5: Setup load balancing and service discovery

Duration: 45 minutes

Until now there have been some restrictions to the scale properties of the service. In this exercise, you will configure a replica set to create pods that use dynamic port mappings to eliminate the port resource constraint during scale activities.

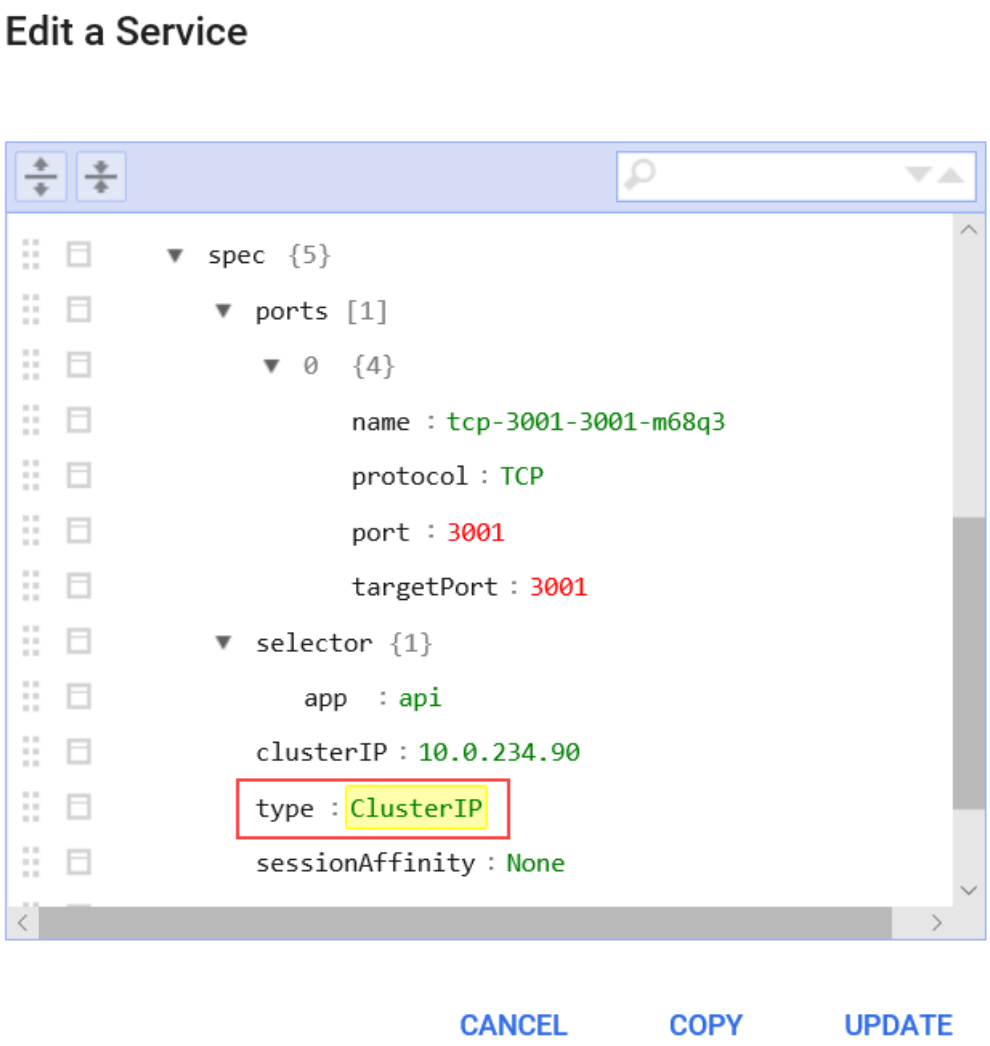
Kubernetes services can discover the ports assigned to each pod, allowing you to run multiple instances of the pod on the same agent node — something that is not possible when you configure a specific static port (such as 3001 for the API service).

Task 1: Create a public load balancer for a service

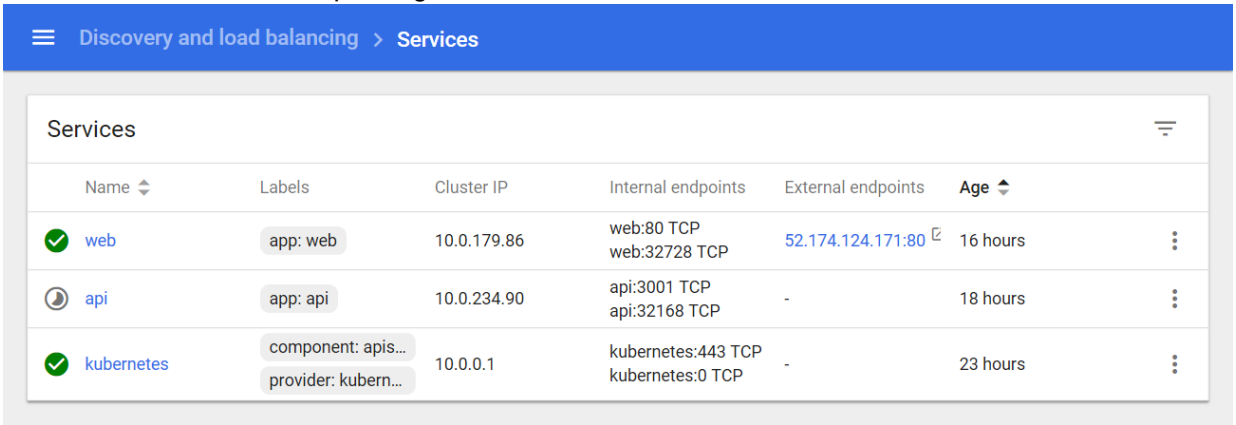
Any external access to Kubernetes pods requires a load balancer. Because of this, you defined the web service with the type LoadBalancer in the YAML description, and you can access the web application using the public IP. In this task, you will change the API service type to LoadBalancer. When you change the type of the service to LoadBalancer, the ACS will create a public-facing load balancer with a public IP address.

1. From the navigation menu, select Services under Discovery and Load Balancing. From the view's Services list, select the API service.
2. Select Edit.

3. From the Edit a Service dialog, scroll down to the type setting and change it from ClusterIP to LoadBalancer. Select Update to save the changes.

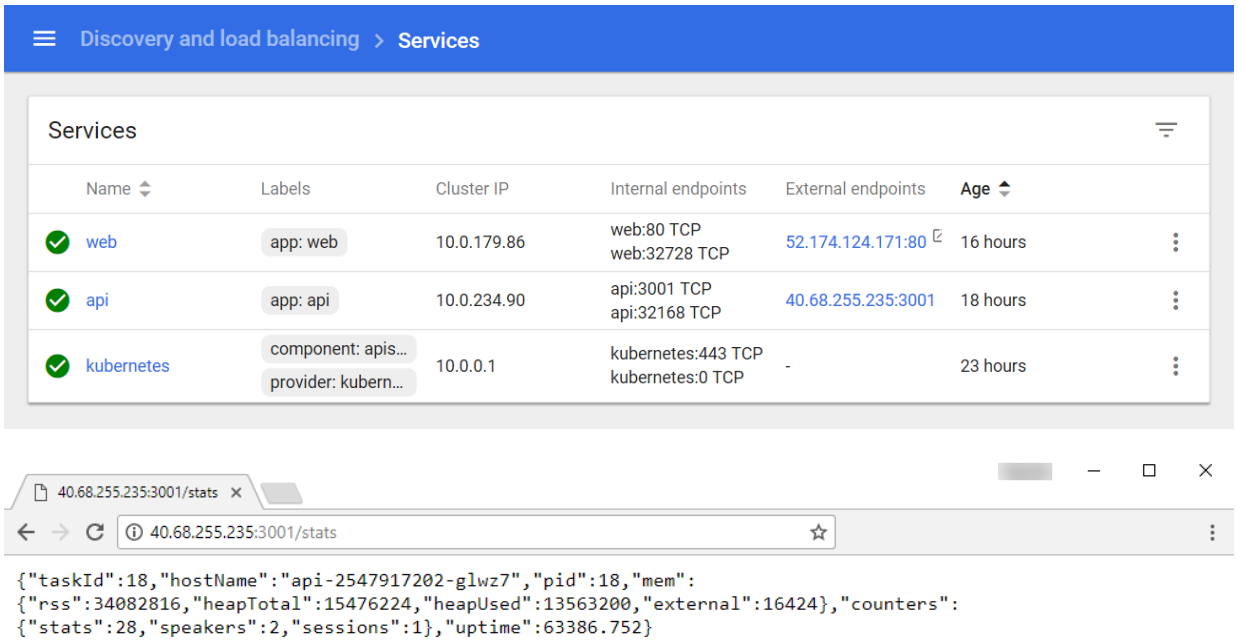


4. The API service will enter a pending state.



5. When the API service is ready again, it will have an external endpoint. You can select the hyperlink and navigate to the http://<external IP>:3001/stats path to see a JSON response. For example, considering the screenshot, the following URL would be used:

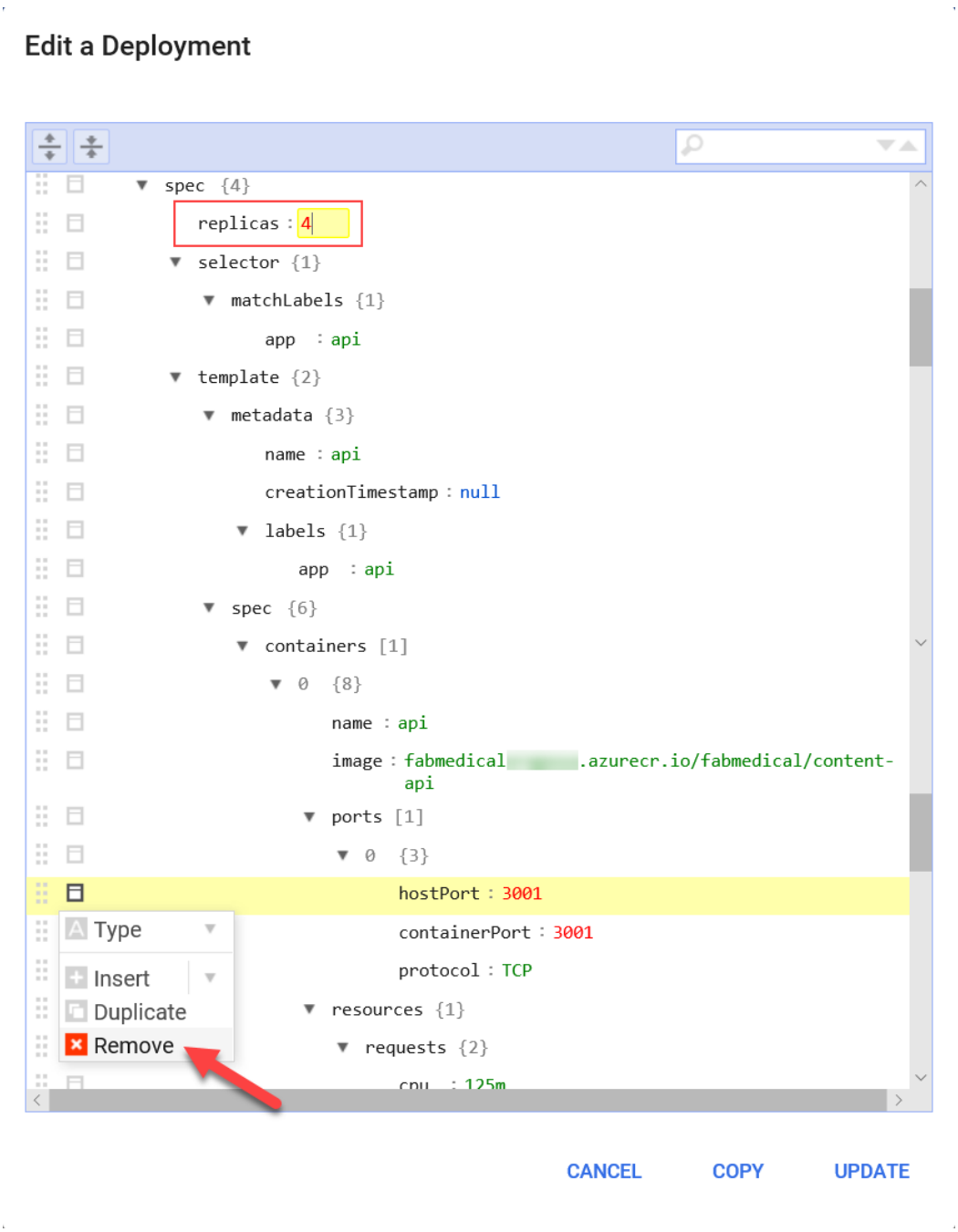
```
http://40.68.255.235:3001/stats
```



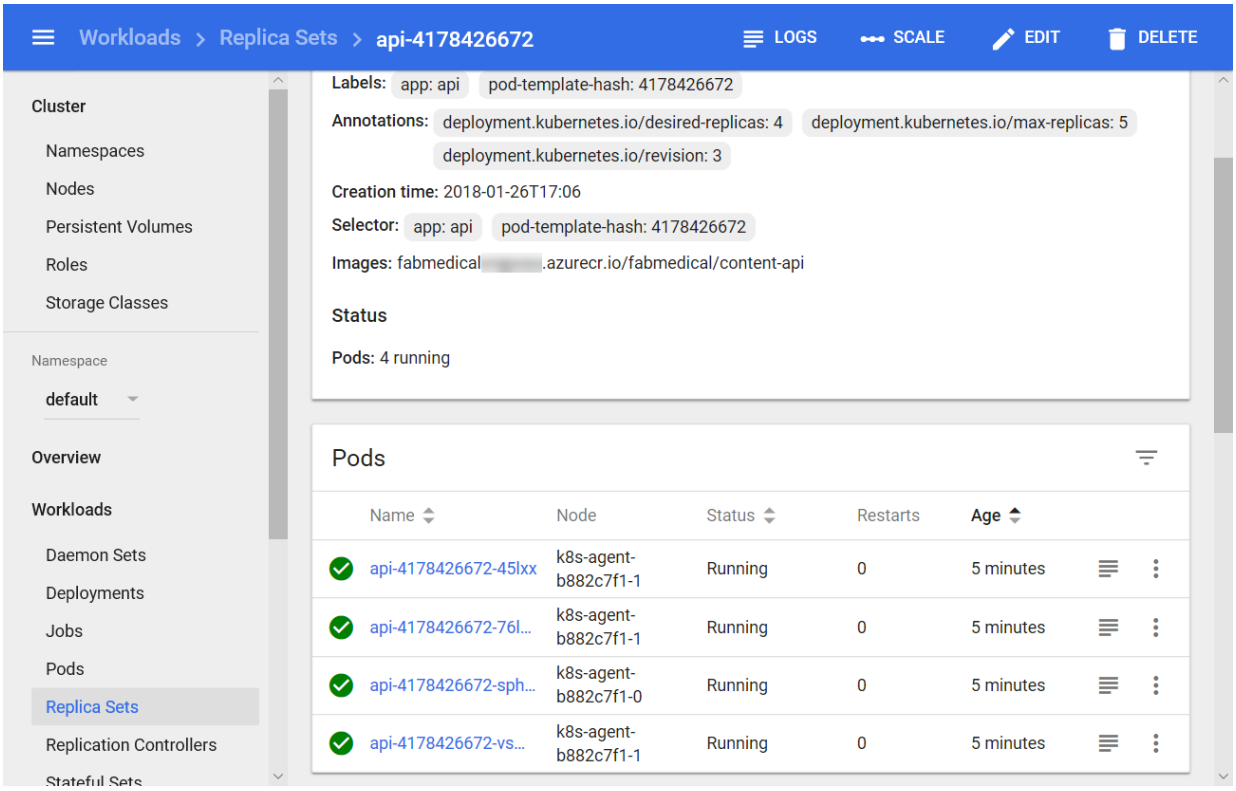
Task 2: Scale a service without port constraints

In this task, we will reconfigure the API deployment so that it will produce pods that choose a dynamic hostPort for routing from the Azure load balancer.

1. From the navigation menu select Deployments under Workloads. From the view's Deployments list select the API deployment.
2. Select Edit.
3. From the Edit a Deployment dialog, do the following:
    a. Scroll to the first spec node that describes replicas as shown in the screenshot. Set the value for replicas to 4.
    b. Within the replicas spec, beneath the template node, find the API containers spec as shown in the screenshot. Remove the hostPort entry for the API container's ports array by selecting the menu for the hostPort node and selecting Remove.



4. Select **Update**. New pods will now choose a dynamic port.
5. The API service can now scale to 4 pods since it is no longer constrained to an instance per node – a previous limitation while using port 3001.



6. Return to the browser and refresh the API stats endpoint previously loaded. Now open a second browser and visit the API stats endpoint, in the second browser you should see a different hostName in the JSON produced by the stats endpoint.

Task 3: Update a service to support dynamic service discovery without a load balancer

The API service does not need a public endpoint to take advantage of the dynamic port discovery. Earlier in this lab you added a public load balancer with an external endpoint to directly observe dynamic discovery to learn about constraints. Now you will restore the API service to its original configuration in preparation for the following tasks that show off how seamlessly Kubernetes supports scaling services for internal endpoints as well.

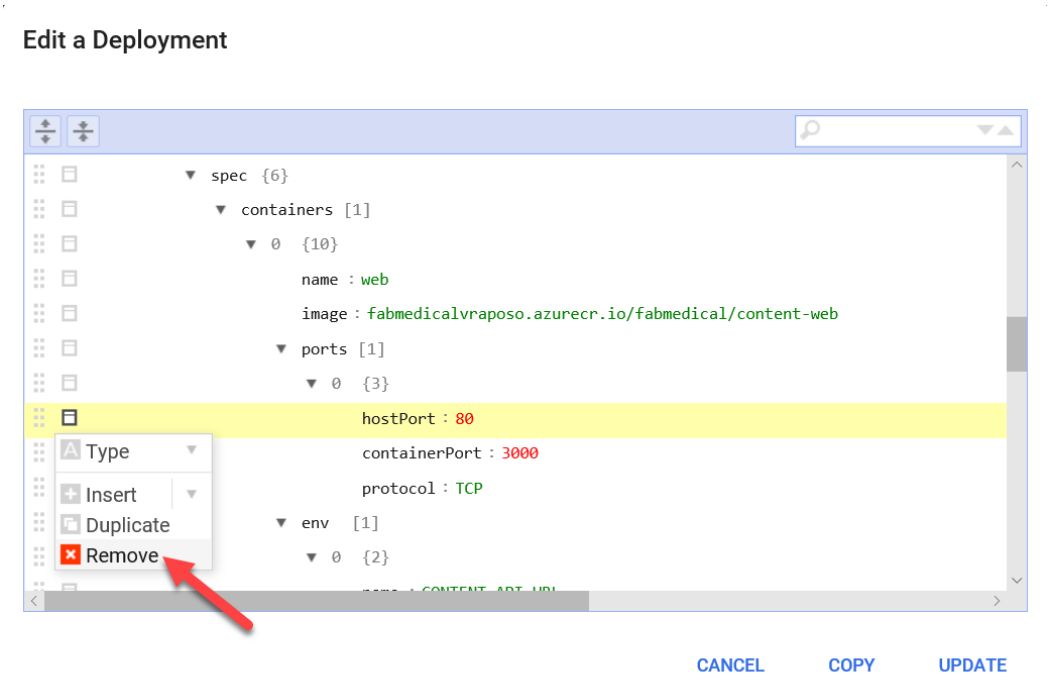
1. From the navigation menu, select Services under Discovery and Load Balancing. From the view's Services list select the API service.
2. Select **Edit**.
3. From the Edit a Service dialog, do the following:
    a. Scroll down to the *type* setting and change it from LoadBalancer to ClusterIP.
    b. Remove the *nodePort* setting.
    c. Select **Update**.

When the deployment update is complete, and the API service is ready, you will no longer be able to access /stats from the public endpoint. In the next task, you will update the web service to support dynamic discovery, but you will still be able to access the web application path at /stats.html as before and observe hostName and taskId changes to reflect the four API pods being called as you refresh.

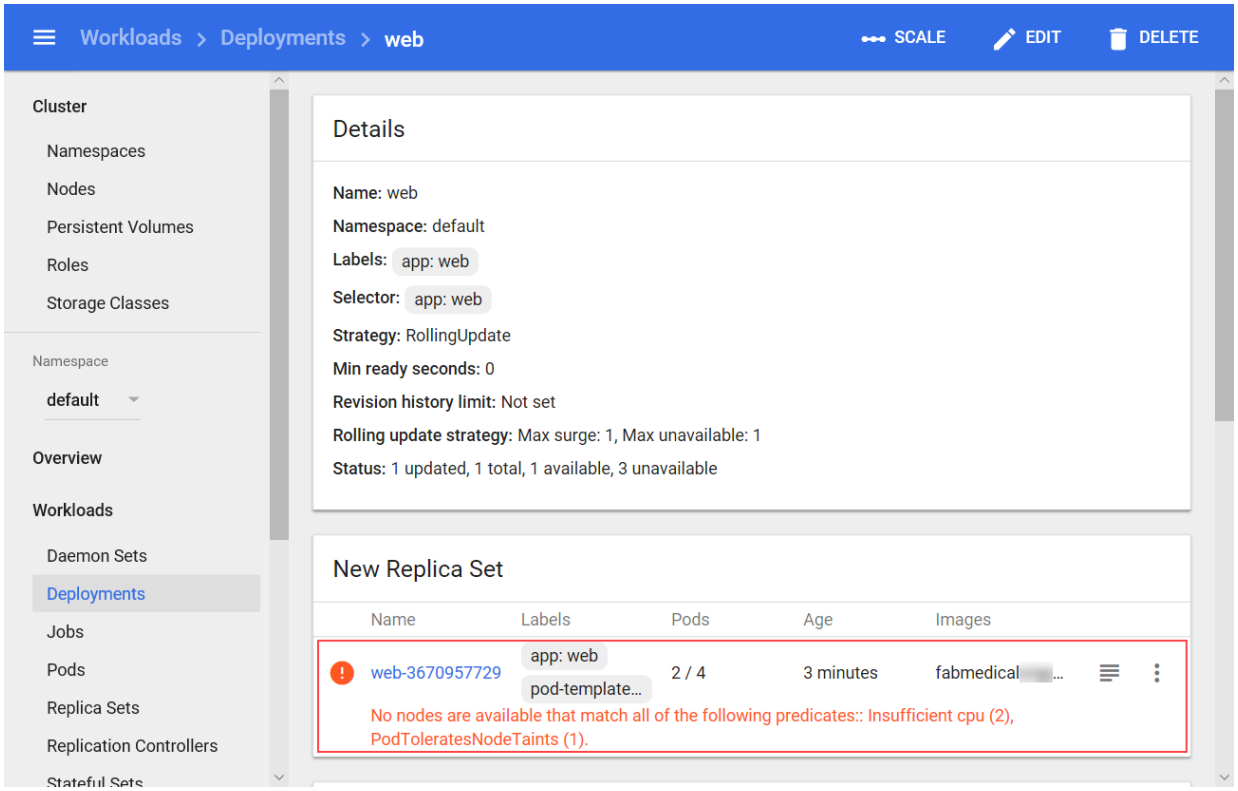
Task 4: Update an external service to support dynamic discovery with a load balancer

In this task, you will update the web service so that it supports dynamic discovery through the Azure load balancer.

1. From the navigation menu, select Deployments under Workloads. From the view's Deployments list select the web deployment.
2. Select **Edit**.
3. From the Edit a Deployment dialog, scroll to the web containers spec as shown in the screenshot. Remove the hostPort entry for the web container's ports array by selecting the menu for the hostPort node and selecting Remove.



4. Select **Update**.
5. From the web Deployments view, select **Scale**. From the dialog presented enter 4 as the desired number of pods and select **OK**.
6. Check the status of the scale out by refreshing the web deployments view. From the navigation menu, select Deployments from under Workloads. Select the web deployment and from this view you should see an error like that shown in the following screenshot.



Like the API deployment, the web deployment used a fixed *hostPort*, and your ability to scale was limited by the number of available agent nodes. However, after resolving this issue for the web service by removing the

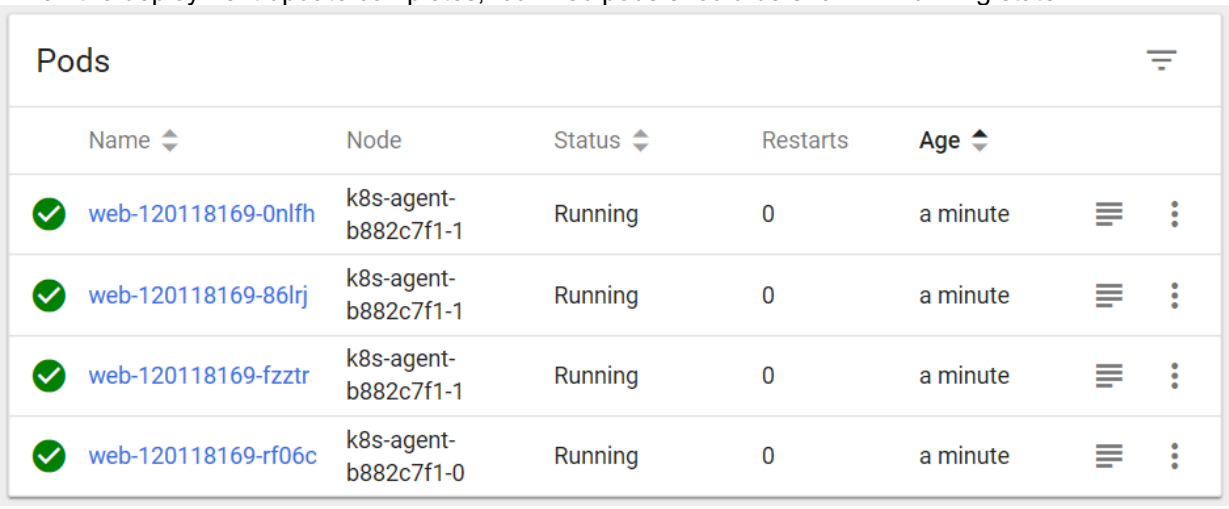
hostPort setting, the web deployment is still unable to scale past two pods due to CPU constraints. The deployment is requesting more CPU than the web application needs, so you will fix this constraint in the next task.

Task 5: Adjust CPU constraints to improve scale

In this task, you will modify the CPU requirements for the web service so that it can scale out to more instances.

1. From the navigation menu, select Deployments under Workloads. From the view's Deployments list select the web deployment.
2. Select **Edit**.
3. From the Edit a Deployment dialog, find the *cpu* resource requirements for the web container. Change this value to "125m".

4. Select **Update** to save the changes and update the deployment.
5. From the navigation menu, select Replica Sets under Workloads. From the view's Replica Sets list select the web replica set.
6. When the deployment update completes, four web pods should be shown in running state.



7. Return to the browser tab with the web application loaded and refresh the stats page at /stats.html to watch the display update to reflect the different api pods by observing the host name refresh.

Task 6: Perform a rolling update

In this task, you will edit the web application source code and update the Docker image used by the deployment. Then you will perform a rolling update to demonstrate how to deploy a code change in a production scenario.

1. Connect to your build agent VM using ssh as you did in Task 6: Connect securely to the build agent before the hands-on lab.
2. From the command line, navigate to the FabMedical/content-web directory.
3. Open the server.js file using VI:

```
vi server.js
```

4. Enter insert mode by pressing <i>.
5. Scroll to the stats endpoint definition:

```
app.get('/api/stats', function (req, res) {
  dataAccess.stats(function (error, data) {
    if (error) {
      return res.status(500).send(error);
    }
    else {
      data.webTaskId = process.pid;
      return res.status(200).send(data);
    }
  });
});
```

6. Change the line that sets data.webTaskId to the following:

```
data.webTaskId = process.env.HOSTNAME;
```

7. Press the Escape key and type ":wq" and then press the Enter key to save and close the file.

```
<Esc>
:wq
<Enter>
```

8. Your current docker image for the web application should be "v1". Verify this by running the command to view your docker images and look for two content-web images with "latest" and "v1" tags.

```
docker images
```

9. Build a new version of the docker image and then tag it with "v2":

```
docker build -t [LOGINSERVER]/fabmedical/content-web .

docker tag [LOGINSERVER]/fabmedical/content-web:latest
[LOGINSERVER]/fabmedical/content-web:v2
```

10. Push these two new tagged images:

```
docker push [LOGINSERVER]/fabmedical/content-web:latest
docker push [LOGINSERVER]/fabmedical/content-web:v2
```

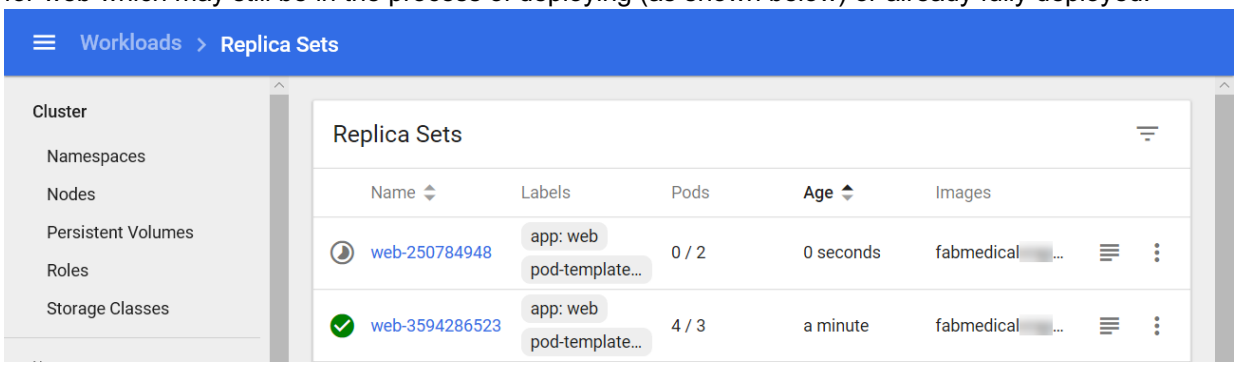
11. Now that you have finished updating the docker image, you can exit the build agent.

```
exit
```

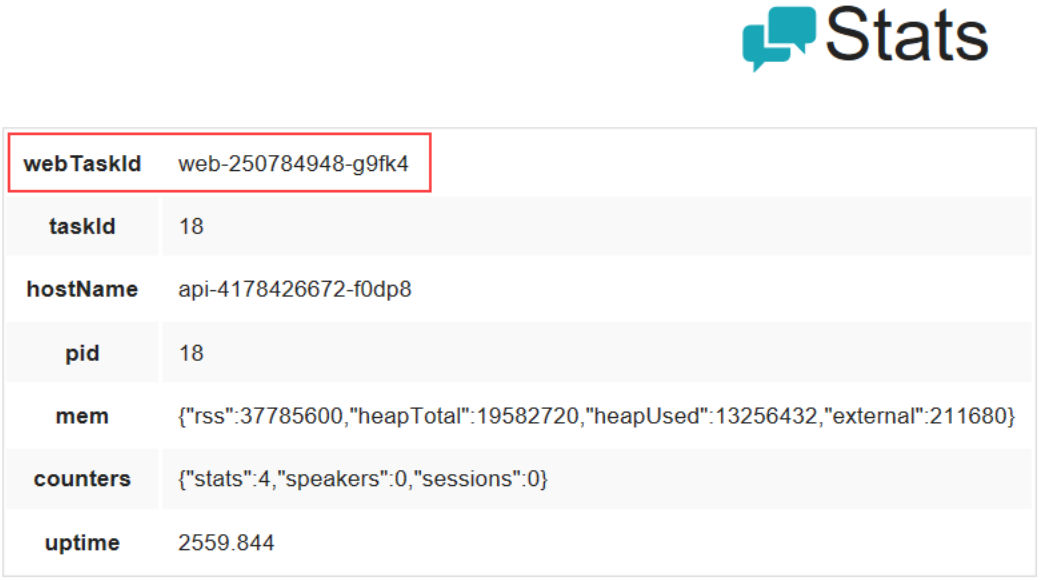
12. From WSL, request a rolling update using this kubectl command:


```
kubectl set image deployment/web web=[LOGINSERVER]/fabmedical/content-web:v2
```

13. While this update runs, return the Kubernetes management dashboard in the browser.
14. From the navigation menu select Replica Sets under Workloads. From this view you will see a new replica set for web which may still be in the process of deploying (as shown below) or already fully deployed.



15. While the deployment is in progress, you can navigate to the web application and visit the stats page at /stats.html. Refresh the page as the rolling update executes. Once the update completes, you will see the webTaskId begins to update correctly to show the hostName of the web pod, as shown here.



16. Open a second browser to view the web application stats page and note that the webTaskId is different for each browser as requests are routed among the available web pods. Notice also that the hostName of the API pod changes from time to time as well when the browser is refreshed.

After the hands-on lab

Duration: 10 mins

In this exercise, you will de-provision any Azure resources created in support of this lab.

1. Delete both of the Resource Groups in which you placed all of your Azure resources.
    a. From the Portal, navigate to the blade of your Resource Group and select Delete in the command bar at the top.
    b. Confirm the deletion by re-typing the resource group name and selecting Delete.
2. Delete the Service Principal created on Task 9: Create a Service Principal before the hands-on lab.

```
az ad sp delete --id "Fabmedical-sp"
```