



Microsoft Cloud Workshop

Containers and DevOps

Hands-on lab step-by-step

March 2018

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

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

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

© 2018 Microsoft Corporation. All rights reserved.

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

Contents

Containers and DevOps hands-on lab step-by-step.....	1
Abstract and learning objectives.....	1
Overview	1
Solution architecture.....	1
Requirements	2
Before the hands-on lab.....	4
Task 1: Resource Group	4
Task 2: Create a Windows 10 Development VM.....	5
Task 3: Install WSL (Bash on Ubuntu on Windows)	7
Task 4: Create an SSH key	8
Task 5: Create a build agent VM.....	9
Task 6: Connect securely to the build agent.....	12
Task 7: Complete the build agent setup.....	14
Task 8: Create an Azure Container Registry.....	15
Task 9: Create a Service Principal.....	17
Task 10: Create an Azure Container Service (AKS) cluster	19
Task 11: Install Azure CLI	22
Task 12: Install Kubernetes CLI	22
Task 13: Download the FabMedical starter files	22
Exercise 1: Create and run a Docker application	23
Task 1: Test the application	24
Task 2: Enable browsing to the web application	24
Task 3: Create a Dockerfile.....	27
Task 4: Create Docker images.....	30
Task 5: Run a containerized application.....	30
Task 6: Setup environment variables.....	31
Task 7: Push images to Azure Container Registry	33
Exercise 2: Deploy the solution to Azure Container Service	37
Task 1: Tunnel into the Azure Container Service cluster	37
Task 2: Deploy a service using the Kubernetes management dashboard.....	39
Task 3: Deploy a service using Kubernetes REST API.....	48
Task 4: Explore service instance logs and resolve an issue	52
Task 5: Test the application in a browser.....	60
Exercise 3: Scale the application and test HA.....	62
Task 1: Increase service instances from the Kubernetes dashboard	62
Task 2: Increase service instances beyond available resources.....	64
Task 3: Restart containers and test HA	66
Exercise 4: Setup load balancing and service discovery.....	72

Task 1: Create a public load balancer for a service	72
Task 2: Scale a service without port constraints	73
Task 3: Update a service to support dynamic service discovery without a load balancer	75
Task 4: Update an external service to support dynamic discovery with a load balancer	76
Task 5: Adjust CPU constraints to improve scale.....	78
Task 6: Perform a rolling update.....	80
After the hands-on lab	82

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 (AKS), 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 and run a Docker Application
- Deploy to the Azure Container Service (AKS)
- 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 (AKS) and want to learn how to deploy containers in a dynamically load-balanced manner, discover containers, and scale them on demand.

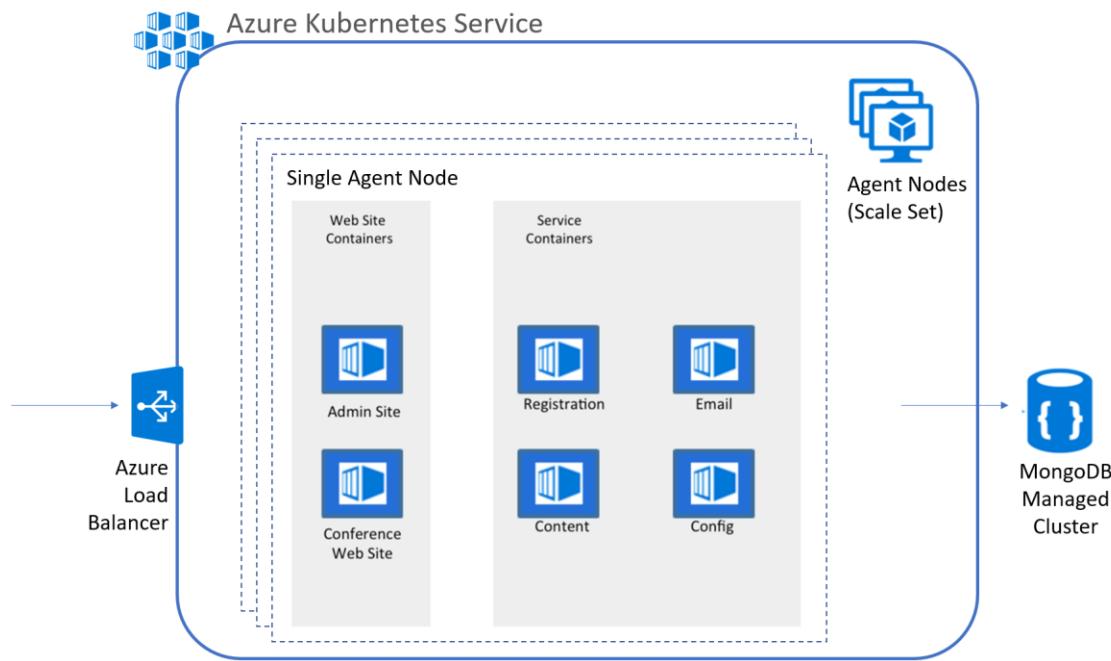
In this hands-on lab, you will assist with completing this POC with a subset of the application code base. You will create a build agent based on Linux, and an Azure Container Service cluster for running deployed applications. You will be helping them to complete the Docker setup for their application,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

Solution architecture

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

The solution will use Azure Kubernetes Service (AKS) which means that the container cluster topology is provisioned according to the number of requested nodes. The proposed containers deployed to the cluster are illustrated below, with MongoDB remaining as a managed service:



Each tenant will have the following containers:

- **Conference Web site:** the SPA application that will use configuration settings to handle custom styles for the tenant
- **Admin Web site:** the SPA application that conference owners use to manage conference configuration details, manage attendee registrations, manage campaigns and communicate with attendees
- **Registration service:** the API that handles all registration activities creating new conference registrations with the appropriate package selections and associated cost
- **Email service:** the API that handles email notifications to conference attendees during registration, or when the conference owners choose to engage the attendees through their admin site
- **Config service:** the API that handles conference configuration settings such as dates, locations, pricing tables, early bird specials, countdowns and related
- **Content service:** the API that handles content for the conference such as speakers, sessions, workshops and sponsors

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 (AKS) 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.

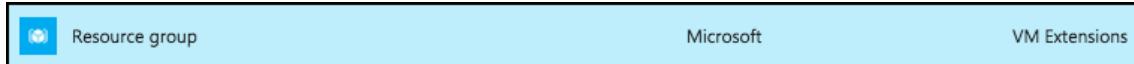
1. In your browser, navigate to the **Azure Portal** (<https://portal.azure.com>).
2. Select **+ Create a resource** in the navigation bar at the left.



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

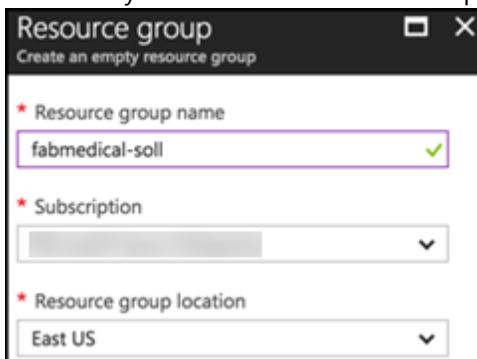


4. Select **Resource group** on the Everything blade and select **Create**.



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**.

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

The screenshot shows the Azure Resource Groups portal. At the top, there are buttons for '+ Add', 'Assign Tags', 'Columns', and 'Refresh'. Below that, a search bar says 'Filter by name...' and dropdown menus for 'All locations' and 'No grouping'. A message says 'Subscriptions: - Don't see a subscription? Switch directories'. The main area shows '2 items' with columns for 'NAME', 'SUBSCRIPTION', and 'LOCATION'. The row for 'fabmedical' has its entire row highlighted with a red box. The 'LOCATION' column shows 'East US'.

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.

- From the Azure Portal, select **+ Create a resource**, type "**Windows 10**" in the Search the marketplace text box and press **Enter**.

The screenshot shows the Azure Marketplace search results for 'Windows 10'. At the top, there's a search bar with 'Windows 10' and a 'Filter' button. Below the search bar is a 'Results' section with a table. The table has columns for 'NAME', 'PUBLISHER', and 'CATEGORY'. There are four items listed: 'Windows 10 Pro, Version 1709' (Publisher: Microsoft, Category: Compute), 'Windows 10 Enterprise N (x64)' (Publisher: Microsoft, Category: Compute), '[HUB] Windows 10 Enterprise Preview' (Publisher: Microsoft, Category: Compute), and 'Windows 10 Pro N, Version 1709' (Publisher: Microsoft, Category: Compute). A red arrow points to the last item, 'Windows 10 Pro N, Version 1709'.

- Select **Windows 10 Pro N, Version 1709** and select **Create**.
- On the Basics blade of the Virtual Machine setup, set the following:
 - Name:** Provide a unique name, such as "fabmedical-SUFFIX" as shown in the following screenshot.
 - VM disk type:** Leave as SSD.
 - 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.

Basics X

* Name
fabmedicald- ✓

VM disk type ?
SSD ▼

* User name
adminfabmedical ✓

* Password
***** ✓

* Confirm password
***** ✓

Subscription
 ▼

* Resource group ?
 Create new Use existing
fabmedical- ▼

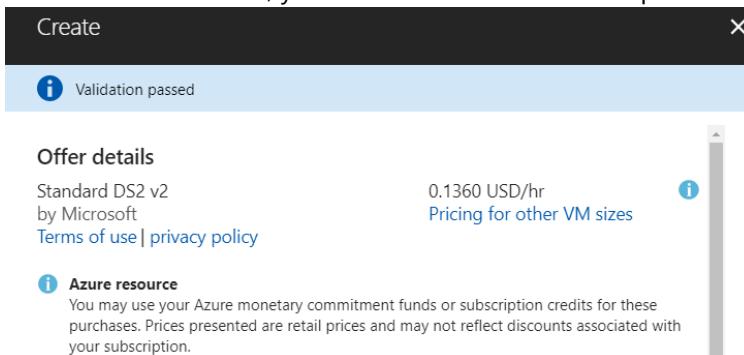
* Location
West Europe ▼

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

★ Recommended | [View all](#)

DS2_V2 Standard ★	DS11_V2 Standard ★	DS2 Standard ★
2 vCPUs	2 vCPUs	2 vCPUs
7 GB	14 GB	7 GB
 8 Data disks	 8 Data disks	 8 Data disks
 6400 Max IOPS	 6400 Max IOPS	 6400 Max IOPS
 14 GB Local SSD	 28 GB Local SSD	 14 GB Local SSD
 Premium disk support	 Premium disk support	 Premium disk support
 Load balancing	 Load balancing	 Load balancing
101.18 USD/MONTH (ESTIMATED)	141.36 USD/MONTH (ESTIMATED)	121.27 USD/MONTH (ESTIMATED)

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.

	NAME	TYPE	LOCATION	
<input type="checkbox"/>	fabmedicald-1	Virtual machine	West Europe	...
<input type="checkbox"/>	fabmedicald-1_OsDisk_1_67df019e638a486c83d380352332818c	Disk	West Europe	...
<input type="checkbox"/>	fabmedicald-1_566	Network interface	West Europe	...
<input type="checkbox"/>	fabmedicald-1-ip	Public IP address	West Europe	...
<input type="checkbox"/>	fabmedicald-1-nsg	Network security group	West Europe	...

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

The screenshot shows the 'Overview' page for a virtual machine named 'fabmedicald-1'. On the left, there's a sidebar with links like 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. On the right, there's a main panel with a toolbar at the top containing 'Connect', 'Start', 'Restart', 'Stop', 'Capture', 'Move', 'Delete', and 'Refresh'. Below the toolbar, the VM details are listed: Resource group (change) to 'fabmedicald-1', Status 'Running', Location 'West Europe', Subscription (change), and Subscription ID. To the right of the details, there's a large list of properties including Computer name, Operating system, Size, Public IP address, Virtual network/subnet, and DNS name.

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. Because you entered ".ssh/fabmedical", the file will be generated in the ".ssh" folder 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.

```
@CarbonSeven:~$ ssh-keygen -t RSA -b 2048 -C admin@fabmedical
Generating public/private RSA key pair.
Enter file in which to save the key (/home/____/.ssh/id_rsa): .ssh/fabmedical
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in .ssh/fabmedical.
Your public key has been saved in .ssh/fabmedical.pub.
The key fingerprint is:
SHA256:Qo3hwFG+aUI3JWRlBTGeLwkzm4AH7runL6DjSNghHdc admin@fabmedical
The key's randomart image is:
+---[RSA 2048]---+
| . ++X+o |
| . o = O * |
| + = E O |
| o + + B * |
| .. + = S . |
|oo o + . |
|oo|
|= ... |
|ooo=|
+---[SHA256]---+
@CarbonSeven:~$
```

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.

- From the Azure Portal, select **+ Create a resource**, type “**Ubuntu**” in the Search the marketplace text box and press **Enter**.

The screenshot shows the Azure Marketplace search results for "Ubuntu". The search bar at the top contains "Ubuntu". Below the search bar, there is a "Results" section with a table. The columns are labeled "NAME", "PUBLISHER", and "CATEGORY". There are four items listed:

NAME	PUBLISHER	CATEGORY
Ubuntu Server 16.04 LTS	Canonical	Virtual Machines
Ubuntu Server 14.04 LTS	Canonical	Virtual Machines
Docker on Ubuntu Server	Canonical + Microsoft	Virtual Machines
Ubuntu Server 12.04.5 LTS	Canonical	Virtual Machines

- Select **Ubuntu Server 16.04 LTS** and select **Create**.

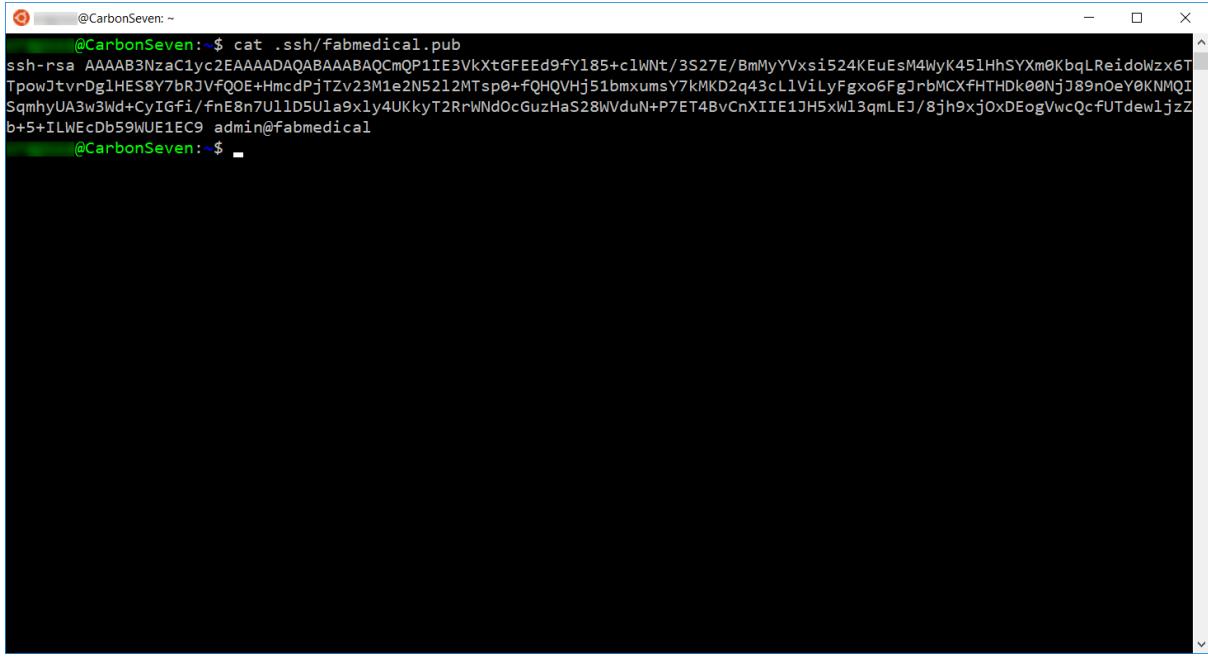
- On the Basics blade of the Virtual Machine setup, set the following:

- Name:** Provide a unique name, such as “fabmedical-SUFFIX” as shown in the following screenshot.
- VM disk type:** Leave as SSD.
- User name:** Provide a user name, such as “adminfabmedical”.
- Authentication type:** Leave as SSH public key.
- SSH public key:** From your local machine, copy the public key portion of the SSH key pair you created previously, to the clipboard.
 - From WSL, verify you are in your user directory shown as “~”. This command will take you there:

```
cd ~
```

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

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



```
@CarbonSeven:~$ cat .ssh/fabmedical.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQmQP1IE3VKXtGFEEd9fyI85+c1wNt/3S27E/BmMyVVxsi524KEuEsM4WyK451HsSYxm0KbqLReidoWzx6T
TpowJtvrDg1HE58Y7brJVfQOE+HmcDPjTZv23M1e2N5212MTsp0+fHQVhj51bxumsY7kMKD2q43cLlViLyFgxo6FgJrbMCXFHTHDk00Njj89nOeY0KNMQI
SqmyUA3w3Wd+CyIGfi/fnE8n7UllDSUla9x1y4UKkyT2RrWNdoGuzHaS28WVduN+P7ET4BvCnXIE1JH5xWl3qmLEJ/8jh9xj0xDegVwcQcfUTdewljzz
b+5+ILWeDb59NUe1Ec9 admin@fabmedical
@CarbonSeven:~$
```

- 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.

Basics

* Name: fabmedical-soll

VM disk type: SSD

* User name: adminfabmedical

* Authentication type: SSH public key

* SSH public key: WrS0dGiUZWtdYTaQ2tTPXpXCrY8MthjMe/ e78ZB8t5GGJr3C9jCoqBMsjPYjKPV3jg4ZWU BpyKmzdd1oo4UUsMjB436qVpW7

Subscription: [dropdown]

* Resource group: Use existing: fabmedical

* Location: East US

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

★ Recommended | [View all](#)

D2S_V3 Standard ★	D4S_V3 Standard ★	E2S_V3 Standard ★
2 vCPUs	4 vCPUs	2 vCPUs
8 GB	16 GB	16 GB
4 Data disks 4000 Max IOPS 16 GB Local SSD 	8 Data disks 8000 Max IOPS 32 GB Local SSD 	4 Data disks 4000 Max IOPS 32 GB Local SSD
89.28 USD/MONTH (ESTIMATED)	178.56 USD/MONTH (ESTIMATED)	119.04 USD/MONTH (ESTIMATED)

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**.

The screenshot shows the 'Create' blade in the Azure portal. At the top, a blue bar indicates 'Validation passed'. Below it, the 'Offer details' section for 'Ubuntu Server 16.04 LTS by Canonical' is displayed. It includes links for 'Terms of use | privacy policy' and 'Pricing details'. The price listed is '0.1200 USD/hr' for 'Standard D2s v3 by Microsoft'. There is also a link for 'Pricing for other VM sizes'.

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.

	NAME	TYPE	LOCATION	
	fabmedicaldiag578	Storage account	East US	...
	fabmedical	Container registry	East US	...
	fabmedical-[REDACTED]	Virtual machine	East US	...
	fabmedical-[REDACTED]_OsDisk_1_8631e96d054b42...	Disk	East US	...
	fabmedical-[REDACTED]_552	Network interface	East US	...
	fabmedical-[REDACTED]-ip	Public IP address	East US	...
	fabmedical-[REDACTED]-nsg	Network security group	East US	...

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.

The screenshot shows the Azure portal's VM overview blade for a resource group named 'fabmedical-'. The 'Overview' tab is selected. Key details shown include:

- Resource group: fabmedical-
- Status: Running
- Location: West Europe
- Subscription: Standard D2s v3 (2 vcpus, 8 GB memory)
- Public IP address: 52.174.141.11 (highlighted in red)
- Virtual network/subnet: fabmedical- -vnet/default
- DNS name: Configure

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:~\$

```
adminfabmedical@fabmedical-vraposo: ~
@CarbonSeven:~$ ssh -i .ssh/fabmedical adminfabmedical@52.174.141.11
The authenticity of host '52.174.141.11 (52.174.141.11)' can't be established.
ECDSA key fingerprint is SHA256:4AAC1xbq4qPqtGwU7y6fVt/QIW89RQQckT566f/Xs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.174.141.11' (ECDSA) to the list of known hosts.
Enter passphrase for key '.ssh/fabmedical':
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.13.0-1006-azure x86_64)

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

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

0 packages can be updated.
0 updates are security updates.

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

adminfabmedical@fabmedical:~$
```

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
```

```
adminfabmedical@fabmedical-:~$ docker version
Client:
 Version:      17.12.0-ce
 API version:  1.35
 Go version:   go1.9.2
 Git commit:   c97c6d6
 Built:        Wed Dec 27 20:11:19 2017
 OS/Arch:      linux/amd64
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.35/version: dial unix /var/run/docker.sock: connect: permission denied
adminfabmedical@fabmedical-:~$
```

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
```

```
adminfabmedical@fabmedical-:~$ sudo usermod -aG docker $USER
sent invalidate(passwd) request, exiting
sent invalidate(group) request, exiting
sent invalidate(passwd) request, exiting
sent invalidate(group) request, exiting
adminfabmedical@fabmedical-:~$
```

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.

```
adminfabmedical@fabmedical-:~$ docker version
Client:
Version: 17.12.0-ce
API version: 1.35
Go version: go1.9.2
Git commit: c97c6d6
Built: Wed Dec 27 20:11:19 2017
OS/Arch: linux/amd64

Server:
Engine:
Version: 17.12.0-ce
API version: 1.35 (minimum version 1.12)
Go version: go1.9.2
Git commit: c97c6d6
Built: Wed Dec 27 20:09:53 2017
OS/Arch: linux/amd64
Experimental: false
adminfabmedical@fabmedical-:~$
```

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`

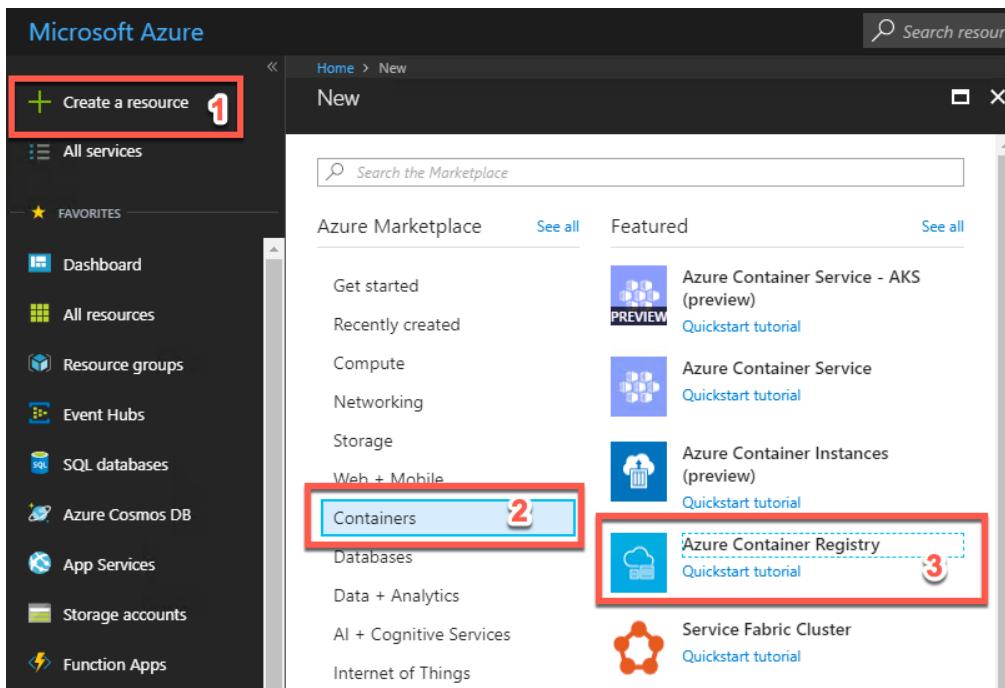
13. In both cases, you will have an empty list but no errors running the command. Your build agent is ready with Docker engine running properly.

```
adminfabmedical@fabmedical-:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS
NAMES
adminfabmedical@fabmedical-:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS
NAMES
adminfabmedical@fabmedical-:~$
```

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 + **Create a resource**, **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.

The screenshot shows the 'Create container registry' blade. It includes fields for Registry name (fabmedicalsol), Subscription (Microsoft Azure Sponsorship), Resource group (Create new, fabmedical-sol), Location (East US), Admin user (Enable), and SKU (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.

The screenshot shows the Azure Portal interface for a Container registry named 'fabmedicalsoil'. The left sidebar has a 'Overview' tab selected, which is highlighted with a dashed blue border. Other tabs include 'Activity log', 'Access control (IAM)', 'Tags', and 'Quick start'. The main content area displays the following details:

- Resource group:** fabmedical
- Location:** East US
- Subscription name:** [REDACTED]
- Subscription ID:** [REDACTED]
- Login server:** fabmedicalsoil.azurecr.io
- Creation date:** 9/5/2017 4:05:41 PM
- SKU:** Standard (Preview)

Task 9: Create a Service Principal

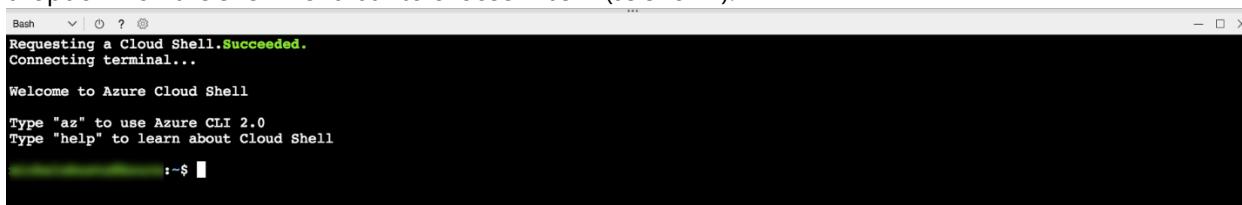
Azure Container Service (AKS) requires an Azure Active Directory service principal to interact with Azure APIs. The service principal is needed to dynamically manage resources such as user-defined routes and the Layer 4 Azure Load Balancer. The easiest way to set up the service principal is using the 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:

The terminal window shows the output of the 'az account show' command:

```
az account show
{
  "environmentName": "AzureCloud",
  "id": "REDACTED",
  "isDefault": true,
  "name": "REDACTED",
  "state": "Enabled",
  "tenantId": "REDACTED",
  "user": {
    "name": "REDACTED",
    "type": "user"
  }
}:~$
```

4. To list all of your subscriptions, type:

```
az account list
```

A screenshot of a terminal window titled "Bash". The command `az account list` is run, and the output is displayed as a JSON array of subscription objects. The output is heavily redacted, showing only the structure and some placeholder values for cloudName, id, name, state, tenantId, and user details.

```
[{"cloudName": "AzureCloud", "id": "REDACTED", "isDefault": false, "name": "REDACTED", "state": "Enabled", "tenantId": "REDACTED", "user": {"name": "REDACTED", "type": "user"}}, {"cloudName": "AzureCloud", "id": "REDACTED", "isDefault": true, "name": "REDACTED", "state": "Enabled", "tenantId": "REDACTED", "user": {"name": "REDACTED", "type": "user"}}]
```

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, and replacing suffix with your chosen suffix to make the name unique.

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

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

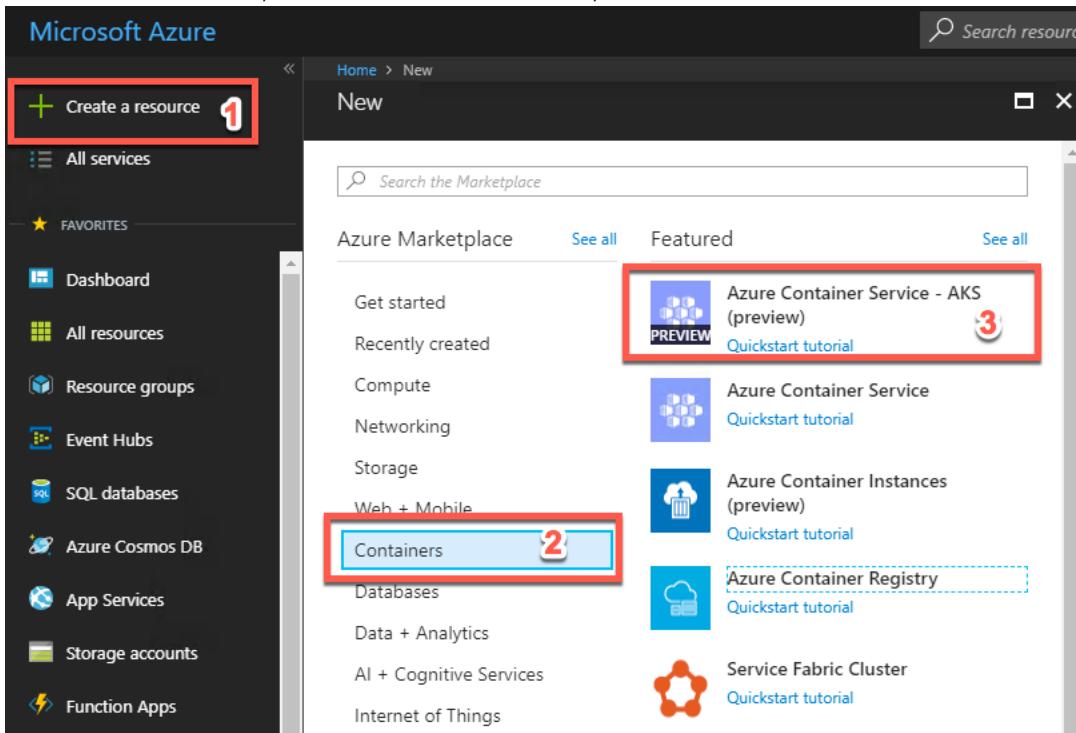
```
@Azure:~$ az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/*" --name="Fabmedical-sp-sol"
{
  "appId": "REDACTED",
  "displayName": "Fabmedical-sp-sol",
  "name": "http://Fabmedical-sp-sol",
  "password": "REDACTED",
  "tenant": "REDACTED"
}
@Azure:~$
```

Task 10: Create an Azure Container Service (AKS) cluster

In this task, you will create your Azure Container Service (AKS) cluster. 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.

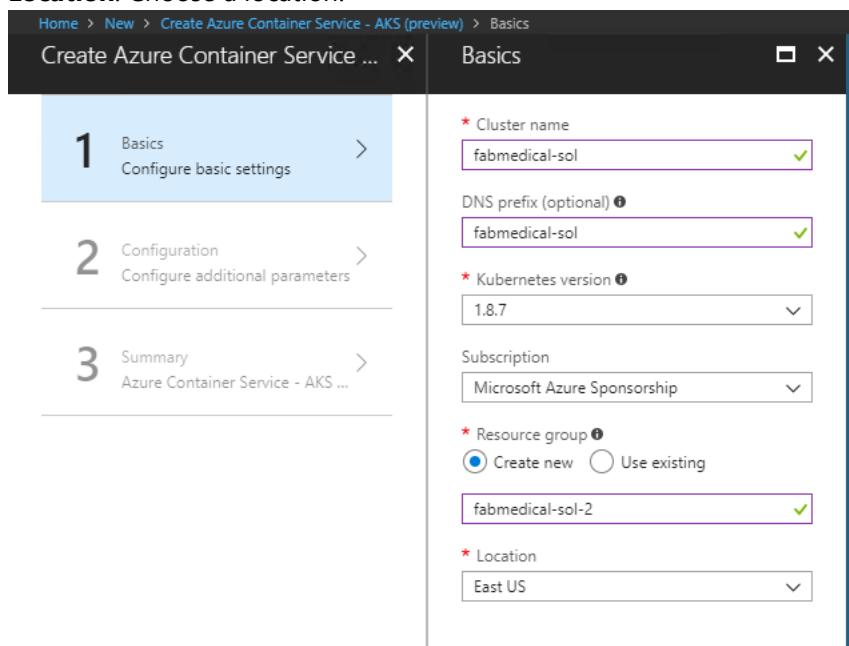
1. From the Azure Portal, select **+ Create a resource**, **Containers** and select **Azure Container Service - AKS**.



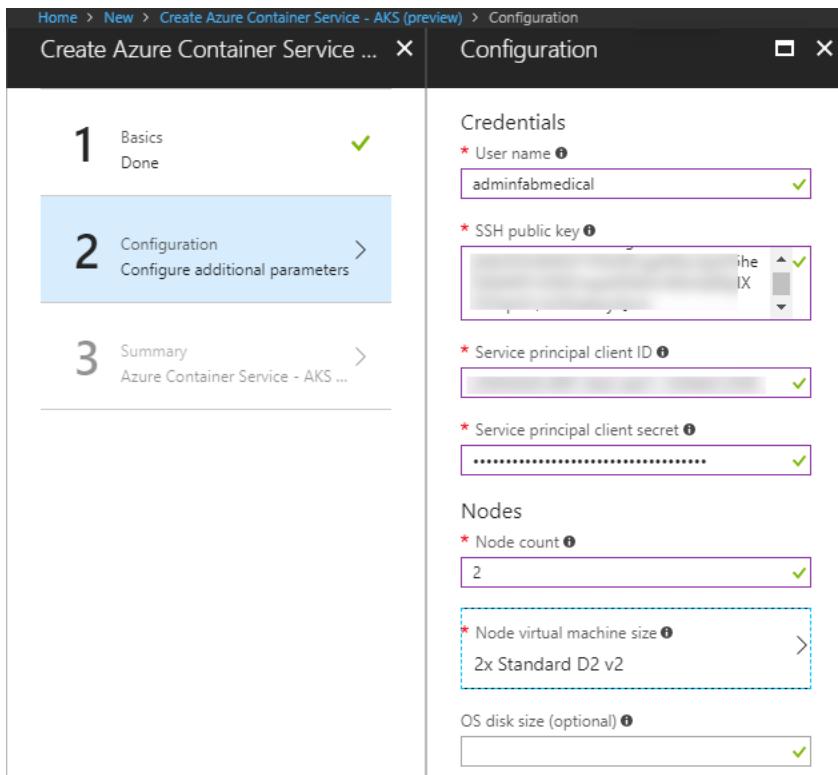
2. In the Basics blade provide the information shown in the screenshot that follows:

- Name:** Enter fabmedical-SUFFIX.
- DNS Prefix:** Enter fabmedical-SUFFIX.
- Kubernetes version:** keep default.
- Subscription:** Choose your subscription which you have been using throughout the lab.

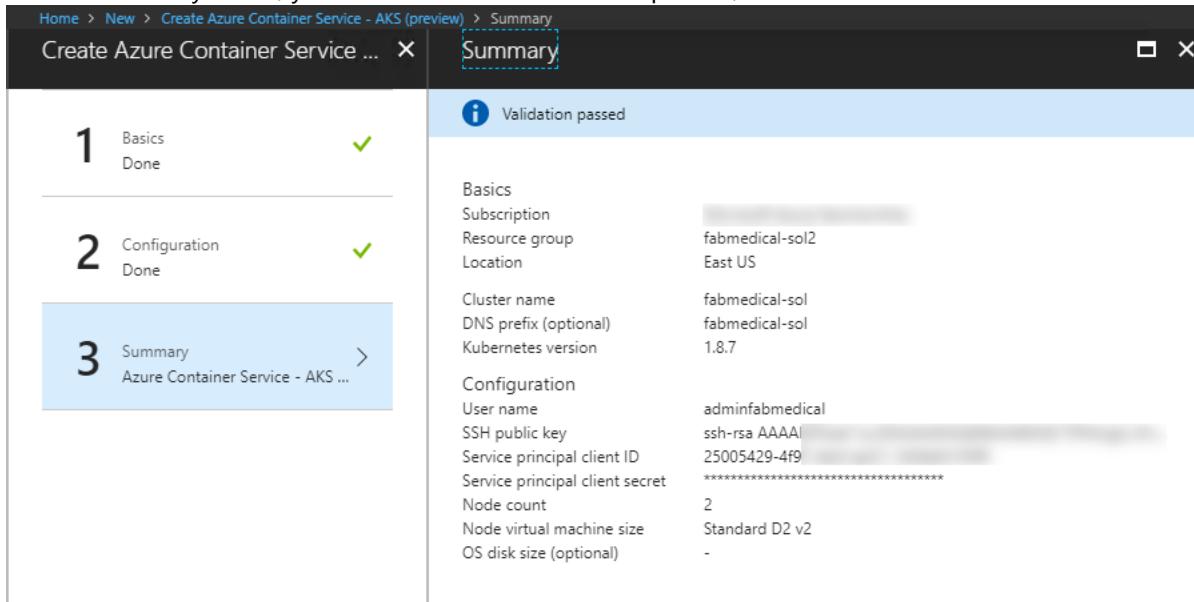
- e. **Resource group:** Create new and provide a unique name. Since the template does not support using an existing Resource Group, provide a new name such adding a "2" to the suffix, such as fabmedical-SUFFIX-2.
- f. **Location:** Choose a location.



3. Select **OK**.
4. On the Configuration blade provide:
 - a. **User name:** Enter adminfabmedical
 - b. **SSH Public Key:** Paste the same SSH public key (fabmedical.pub) you used for the agent VM previously.
 - c. **Service principal client ID:** Use the service principal "appId" from the previous step.
 - d. **Service principal client secret:** Use the service principal "password" from the previous step.
 - a. **Node count:** 2
 - b. **Node virtual machine size:** Standard DS2_V2
 - c. **OS Disk Size:** leave empty



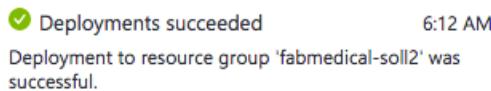
5. Select **OK**.
6. On the Summary blade, you should see that validation passed; select **OK**.



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



8. 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.



✓ Deployments succeeded 6:12 AM
Deployment to resource group 'fabmedical-soll2' was successful.

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
```

Task 13: Download the FabMedical starter files

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 (AKS) environment.

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.

```
@CarbonSeven:~$ ssh -i .ssh/fabmedical adminfabmedical@[REDACTED]
Enter passphrase for key '.ssh/fabmedical':
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.13.0-1006-azure x86_64)

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

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

0 packages can be updated.
0 updates are security updates.

Last login: Thu Jan 25 01:16:59 2018 from 37.189.133.172
adminfabmedical@fabmedical-[REDACTED]:~$
```

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/2IyjeYl
```

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
11
```

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 1: 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 &
```

```
admin@fabmedical-[REDACTED]:~/FabMedical/content-api$ nodejs ./server.js &  
[1] 73302  
admin@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 &
```

```
admin@fabmedical-[REDACTED]:~/FabMedical/content-web$ nodejs ./server.js &  
[2] 73785  
admin@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

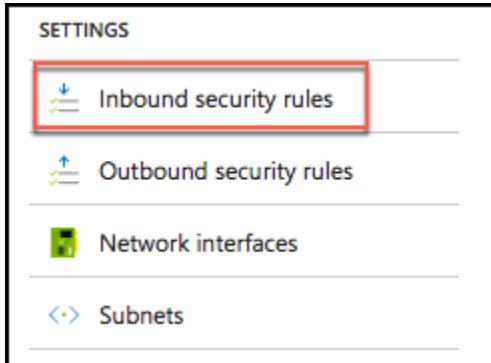
In this task, you will open a port range on the agent VM so that you can browse to the web application for testing.

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.

<input type="checkbox"/>	fabmedical	Container registry
<input type="checkbox"/>	fabmedical-	Virtual machine
<input type="checkbox"/>	fabmedical- OsDisk_1_Oeee59d6df0142b6b7ce3528...	Disk
<input type="checkbox"/>	fabmedical- 932	Network interface
<input type="checkbox"/>	fabmedical- -ip	Public IP address
<input type="checkbox"/>	fabmedical- -nsg	Network security group
<input type="checkbox"/>	fabmedical- -vnet	Virtual network
<input type="checkbox"/>	fabmedicalstore	Storage account

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



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

The screenshot shows the 'Inbound security rules' blade. At the top, there is a list of existing rules: 'fabmedical-nsg' (selected) and 'Default rules'. Below this is a search bar labeled 'Search inbound security rules'. A red arrow points to the '+ Add' button. The main table lists one rule: 'default-allow-ssh' with priority 1000, source Any, destination Any, service SSH (TCP/22), and action Allow.

PRIORITY	NAME	SOURCE	DESTINATION	SERVICE	ACTION
1000	default-allow-ssh	Any	Any	SSH (TCP/22)	Allow

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"

Add inbound security rule
fabmedical-soll-nsg

Basic

* Source: Any

* Source port ranges: *

* Destination: Any

* Destination port ranges: 3000-3010

* Protocol: Any (selected)

* Action: Allow (selected)

* Priority: 100

* Name: allow-app-endpoints

Description:

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

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
100	allow-app-endpoints	3000-3010	Any	Any	Any	<input checked="" type="checkbox"/> Allow
1000	default-allow-ssh	22	TCP	Any	Any	<input checked="" type="checkbox"/> Allow

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

NAME	TYPE	LOCATION	...
 fabmedical-soll	Virtual machine	East US 2	...
 LinuxAsm	Microsoft.Compute/vi...	East US 2	...
 fabmedical-soll261	Network interface	East US 2	...
 fabmedicalsolldiag273	Storage account	East US 2	...
 fabmedicalsolldisks565	Storage account	East US 2	...
 fabmedical-soll-ip	Public IP address	East US 2	...
 fabmedical-soll-nsg	Network security group	East US 2	...
 fabmedical-soll-vnet	Virtual network	East US 2	...

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

The screenshot shows the Azure portal's Virtual Machine blade for a VM named "fabmedical-*". The "Overview" tab is selected. Key details shown include:

- Resource group: fabmedical-*
- Status: Running
- Location: West Europe
- Subscription: Standard D2s v3 (2 vcpus, 8 GB memory)
- Public IP address: 52.174.141.11 (highlighted in red)
- Virtual network/subnet: fabmedical-*-vnet/default
- DNS name: Configure

- 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
```

- 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.
- 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.

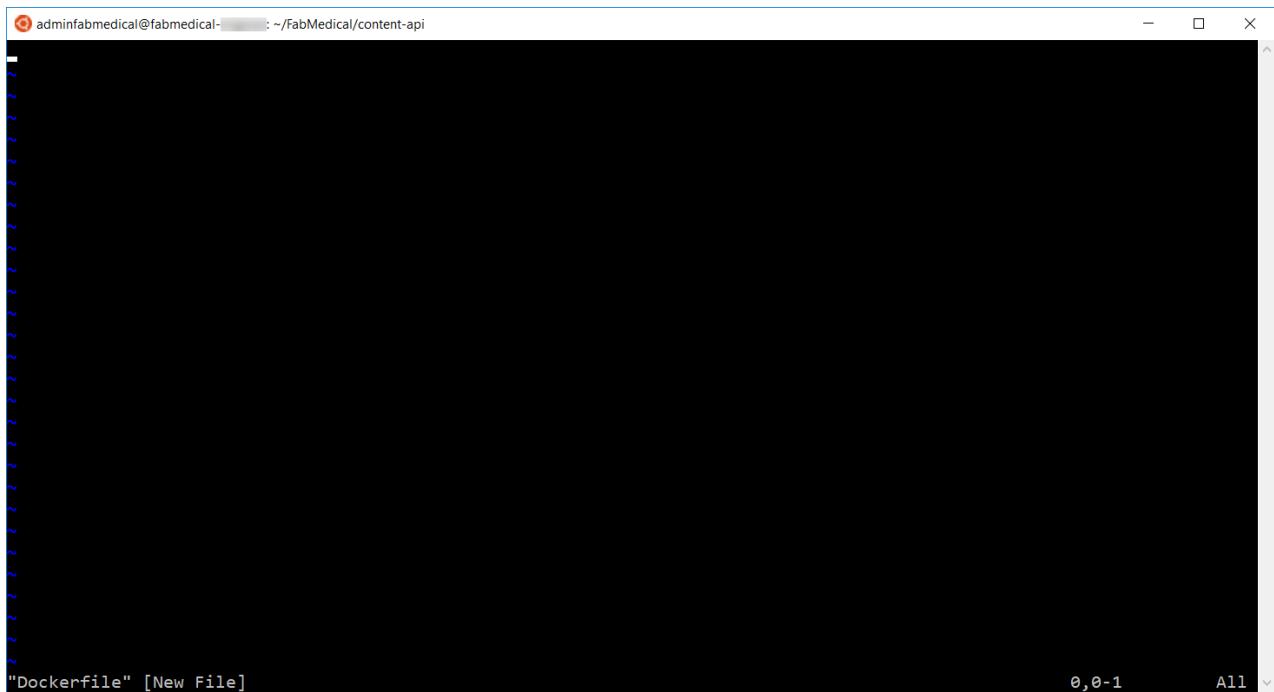
- From WSL, navigate to the content-api folder. List the files in the folder with this command. The output should look like the screenshot below.

```
cd ..  
cd content-api  
11
```

```
adminfabmedical@fabmedical-sol:~/FabMedical/content-api$ ll
total 64
drwxrwxr-x 3 adminfabmedical adminfabmedical 4096 Mar 27 03:01 .
drwxrwxr-x 4 adminfabmedical adminfabmedical 4096 Mar 27 02:48 ..
-rwxrwxr-x 1 adminfabmedical adminfabmedical 239 Mar 27 02:32 Dockerfile-finish*
drwxrwxr-x 50 adminfabmedical adminfabmedical 4096 Mar 27 02:49 node_modules/
-rwxrwxr-x 1 adminfabmedical adminfabmedical 224 Mar 27 02:32 package.json*
-rwxrwxr-x 1 adminfabmedical adminfabmedical 13078 Mar 27 02:32 package-lock.json*
-rwxrwxr-x 1 adminfabmedical adminfabmedical 903 Mar 27 02:32 routes.js*
-rwxrwxr-x 1 adminfabmedical adminfabmedical 960 Mar 27 02:32 server.js*
-rwxrwxr-x 1 adminfabmedical adminfabmedical 11008 Mar 27 02:32 sessions.json*
-rwxrwxr-x 1 adminfabmedical adminfabmedical 5448 Mar 27 02:32 speakers.json*
adminfabmedical@fabmedical-sol:~/FabMedical/content-api$
```

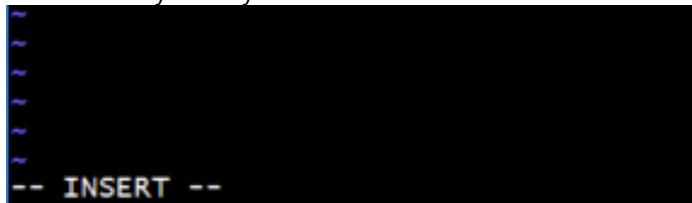
2. Create a new file named "Dockerfile" and note the casing in the name. Use the following Vim command to create a new file. The WSL window should look as shown in the following screenshot.

```
vi Dockerfile
```



The screenshot shows a WSL terminal window with the command 'vi Dockerfile' entered. The file 'Dockerfile' is a new file, indicated by the '(New File)' suffix at the bottom of the screen. The Vim interface is visible, with the status bar showing '0,0-1' and 'All'. The bottom status bar also displays '(New File)'.

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.

```
11
```

```
adminfabmedical@fabmedical-sol:~/FabMedical/content-api
adminfabmedical@fabmedical-sol:~/FabMedical/content-api$ ll
total 68
drwxrwxr-x 3 adminfabmedical adminfabmedical 4096 Mar 27 03:05 .
drwxrwxr-x 4 adminfabmedical adminfabmedical 4096 Mar 27 02:48 /
-rw-rw-r-x 1 adminfabmedical adminfabmedical 239 Mar 27 02:32 Dockerfile*
-rw-rw-r-x 1 adminfabmedical adminfabmedical 239 Mar 27 03:05 Dockerfile-trimish*
drwxrwxr-x 50 adminfabmedical adminfabmedical 4096 Mar 27 02:49 node_modules/
-rw-rw-r-x 1 adminfabmedical adminfabmedical 224 Mar 27 02:32 package.json*
-rw-rw-r-x 1 adminfabmedical adminfabmedical 13078 Mar 27 02:32 package-lock.json*
-rw-rw-r-x 1 adminfabmedical adminfabmedical 903 Mar 27 02:32 routes.js*
-rw-rw-r-x 1 adminfabmedical adminfabmedical 960 Mar 27 02:32 server.js*
-rw-rw-r-x 1 adminfabmedical adminfabmedical 11008 Mar 27 02:32 sessions.json*
-rw-rw-r-x 1 adminfabmedical adminfabmedical 5448 Mar 27 02:32 speakers.json*
adminfabmedical@fabmedical-sol:~/FabMedical/content-api$ -
```

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.

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

```
docker images
```

- 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 .
```

- 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
```

```
Successfully built 52adf58220d4
adminfabmedical@fabmedical:~/FabMedical/FabMedical/content-api$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
content-api        latest   52adf58220d4  2 seconds ago  661.9 MB
node                argon    cfe7aced3467  8 days ago   654.6 MB
adminfabmedical@fabmedical:~/FabMedical/FabMedical/content-api$ |
```

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

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

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

```
cat Dockerfile
```

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

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

- 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   c25f7830cf1   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.

- 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
```

```
admin@fabmedical- [~]:~/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
admin@fabmedical- [~]:~/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
```

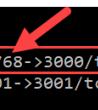
5. Create and start the web application container with a similar Docker run command – instruct the docker engine to use any port with the -P command:

```
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
```

```
admin@fabmedical- [~]:~/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
admin@fabmedical- [~]:~/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>
```

- 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;
```

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

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

- 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>
```

- 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" ]  
=
```

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

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

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

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

- 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 c 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.

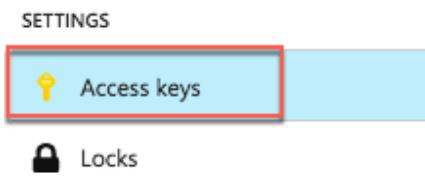
```
http://[BUILDAGENTIP]:3000
EXAMPLE: http://13.68.113.176:3000
```

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 fabmedical.soll.azurecr.io -u fabmedical -p
+W/j=1+Fcze=n07SchxvGSlvsLRh/7ga
```

```
admin@fabmedical-:~/FabMedical/content-web$ docker login fabmedical .azurecr.io -u fabmedical -p
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
admin@fabmedical-:~/FabMedical/content-web$
```

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

- 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
```

- 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
```

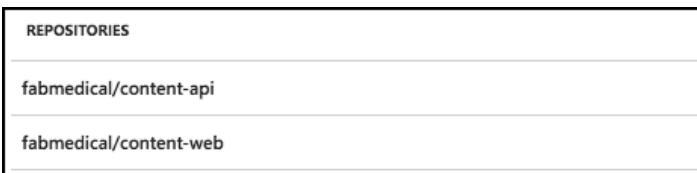
```
admin@fabmedical:~/FabMedical/FabMedical/content-web$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
content-web         latest   fdd59f1cafc3  53 minutes ago  724.5 MB
fabmedical.soll.azurecr.io/fabmedical/content-web  latest   fdd59f1cafc3  53 minutes ago  724.5 MB
<none>              <none>   c25f7830cf1   About an hour ago  724.5 MB
fabmedical.soll.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
admin@fabmedical:~/FabMedical/FabMedical/content-web$
```

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

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

```
admin@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
b9914af042f: Waiting
4bcdffd70da2: Waiting
```

- 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.

The screenshot shows the Azure Container Registry interface. On the left, under 'REPOSITORIES', the 'fabmedical/content-api' repository is selected and highlighted in blue. On the right, under 'TAGS', the 'latest' tag is selected and highlighted with a dashed blue border. Both sections have a search bar at the top.

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
```

```
admin@fabmedical:~/FabMedical/FabMedical/content-web$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
content-web          latest   fd5d9f1caf3c  About an hour ago  724.5 MB
fabmedicals01.azurecr.io/fabmedical/content-web    latest   fd5d9f1caf3c  About an hour ago  724.5 MB
fabmedicals01.azurecr.io/fabmedical/content-web    v1      fd5d9f1caf3c  About an hour ago  724.5 MB
<none>              <none>   c25f7830cf1c  About an hour ago  724.5 MB
content-api          latest   52adf58220d4  About an hour ago  661.9 MB
fabmedicals01.azurecr.io/fabmedical/content-api    latest   52adf58220d4  About an hour ago  661.9 MB
fabmedicals01.azurecr.io/fabmedical/content-api    v1      52adf58220d4  About an hour ago  661.9 MB
node                argon    cfe7aced3467  8 days ago    654.6 MB
admin@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.

The screenshot shows the Azure Container Registry interface. On the left, under 'REPOSITORIES', the 'fabmedical/content-api' repository is selected and highlighted in blue. On the right, under 'TAGS', both the 'latest' and 'v1' tags are listed. Both sections have a search bar at the top.

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
```

```
admin@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
admin@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
admin@fabmedical-:~/FabMedical/content-web$ -
```

Exercise 2: 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:

```
:~$ az --version
azure-cli (2.0.29)

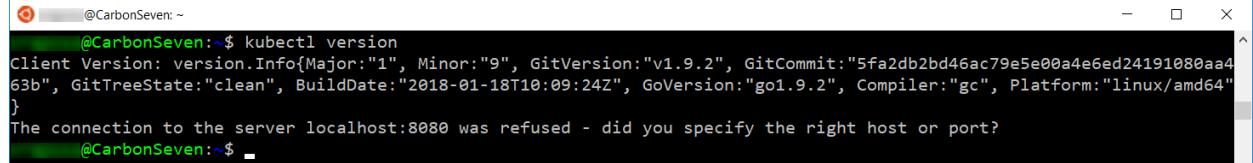
acr (2.0.22)
acs (2.0.28)
advisor (0.5.0)
appservice (0.1.29)
backup (1.0.7)
batch (3.1.11)
batchai (0.1.6)
billing (0.1.7)
cdn (0.0.13)
cloud (2.0.12)
cognitiveservices (0.1.11)
command-modules-nspkg (2.0.1)
configure (2.0.14)
consumption (0.2.2)
container (0.1.19)
core (2.0.29)
cosmosdb (0.1.19)
dla (0.0.18)
dls (0.0.19)
eventgrid (0.1.11)
eventhubs (0.1.0)
extension (0.0.10)
feedback (2.1.0)
find (0.2.8)
interactive (0.3.17)
iot (0.1.18)
keyvault (2.0.20)
```

- 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:



```
@CarbonSeven:~$ kubectl version
Client Version: version.Info{Major:"1", Minor:"9", GitVersion:"v1.9.2", GitCommit:"5fa2db2bd46ac79e5e00a4e6ed24191080aa463b", GitTreeState:"clean", BuildDate:"2018-01-18T10:09:24Z", GoVersion:"go1.9.2", Compiler:"gc", Platform:"linux/amd64"}
The connection to the server localhost:8080 was refused - did you specify the right host or port?
@CarbonSeven:~$
```

- 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
```

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

```
az account show
```

- 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):

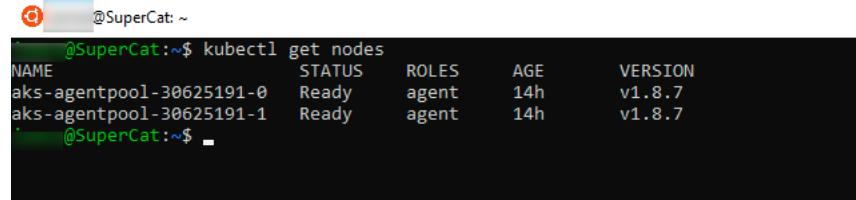
```
az account list  
az account set --subscription {id}
```

- Configure kubectl to connect to the Kubernetes cluster.

```
az aks get-credentials --name fabmedical-SUFFIX --resource-group fabmedical-SUFFIX-2
```

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

```
kubectl get nodes
```

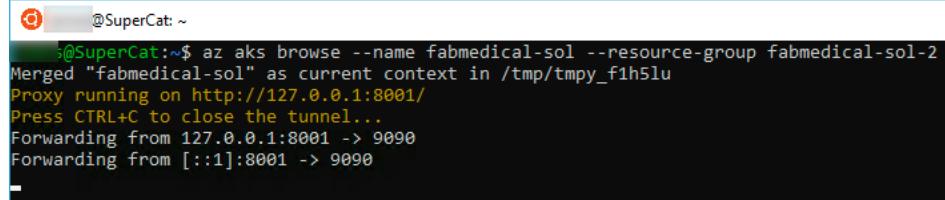


```
@SuperCat:~$ kubectl get nodes  
NAME           STATUS   ROLES      AGE     VERSION  
aks-agentpool-30625191-0   Ready    agent      14h    v1.8.7  
aks-agentpool-30625191-1   Ready    agent      14h    v1.8.7  
@SuperCat:~$ -
```

- 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:

NOTE: After you run this command, it may work at first and later lose its connection, so you may have to run this again to reestablish the connection. If the Kubernetes dashboard becomes unresponsive in the browser this is an indication to return here and check your tunnel or rerun the command.

```
az aks browse --name fabmedical-SUFFIX --resource-group fabmedical-SUFFIX-2
```



```
@SuperCat:~$ az aks browse --name fabmedical-sol --resource-group fabmedical-sol-2  
Merged "fabmedical-sol" as current context in /tmp/tmpy_f1h5lu  
Proxy running on http://127.0.0.1:8001/  
Press CTRL+C to close the tunnel...  
Forwarding from 127.0.0.1:8001 -> 9090  
Forwarding from [::1]:8001 -> 9090  
-
```

- 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://localhost:8001
```

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

The screenshot shows the Kubernetes Management Dashboard with the 'Overview' tab selected. On the left sidebar, under 'Cluster', the 'Namespaces' section shows 'default'. Under 'Workloads', it lists 'Cron Jobs', 'Daemon Sets', and 'Deployment'. The main area has two tabs: 'Discovery and Load Balancing' and 'Config and Storage'. The 'Services' tab is active, displaying a table with one row for the 'kubernetes' service. The 'Secrets' tab is also visible below it. The 'kubernetes' service row in the 'Services' table contains the following data:

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
kubernetes	component: kube-apiserver, provider: k8s	10.0.0.1	kubernetes:443 kubernetes:0	-	14 hours

The 'Secrets' table shows one entry:

Name	Type	Age
default-token-s6kmc	kubernetes.io/service-account	14 hours

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.
2. From the Resource creation view, select Create an App

The screenshot shows the Kubernetes Resource creation interface. On the left sidebar, under the Cluster section, the 'Namespaces' option is selected, showing the 'default' namespace. Under the Workloads section, several options like Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, and Stateful Sets are listed.

The main area displays the 'CREATE AN APP' tab, which includes the following fields:

- App name ***: api
- Container image ***: fabmedicalsol.azurecr.io/fabmedical/content-api
- Number of pods ***: 1
- Service ***: Internal
- Port ***: 3001
- Target port ***: 3001
- Protocol ***: TCP

Below these fields, there is a link to **SHOW ADVANCED OPTIONS**. At the bottom of the form are two buttons: **DEPLOY** (highlighted in blue) and **CANCEL**.

- 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.
3. Select SHOW ADVANCED OPTIONS.
- a. Enter 0.125 for the CPU requirement.
 - b. Enter 128 for the Memory requirement.

Description

The description will be added as an annotation to the Deployment and displayed in the application's details.

Labels

Key	Value
k8s-app	api

3 / 253

0 / 253

Namespace *

default

Namespaces let you partition resources into logically named groups. [Learn more](#)

Image Pull Secret

The specified image could require a pull secret credential if it is private. You may choose an existing secret or create a new one. [Learn more](#)

CPU requirement (core)

.125

Memory requirement (MiB)

128

You can specify minimum CPU and memory requirements for the container. [Learn more](#)

Run command

By default, your containers run the selected image's default entrypoint command. You can use the command options to override the default. [Learn more](#)

4. 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.

Cluster

Namespaces

Nodes

Persistent Volumes

Roles

Storage Classes

Namespace

default

Overview

Workloads

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

Workloads Statuses

Deployments

100.00%

Pods

100.00%

Replica Sets

100.00%

Deployments

Name	Labels	Pods	Age	Images
api	k8s-app: api	0 / 1	0 seconds	fabmedicalsol.a...

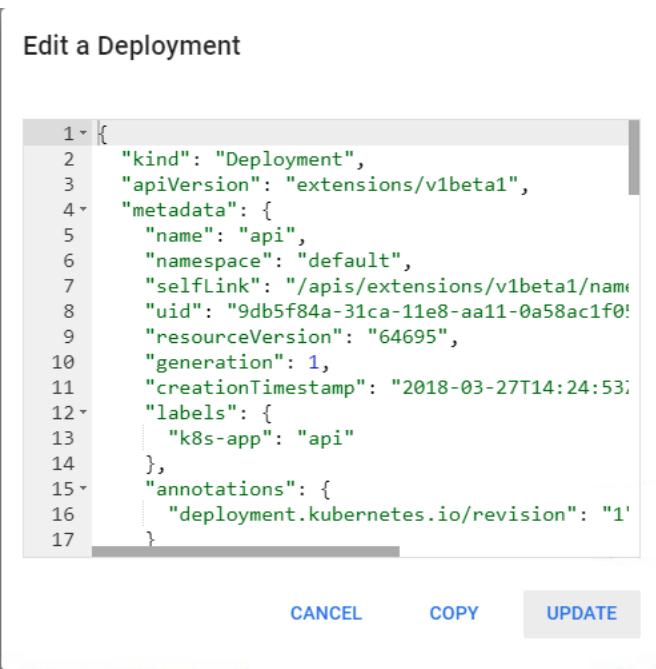
Pods

Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
api-5dddfdf	aks-agentpool-30625191-1	Waiting: Con	0	0 seconds	-	-

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

☰ Workloads > Deployments > api ⚙ SCALE 🖊 EDIT 🗑 DELETE

- In the Edit a Deployment dialog you will see a list of settings shown in JSON format. Use the copy button to copy the text to your clipboard.

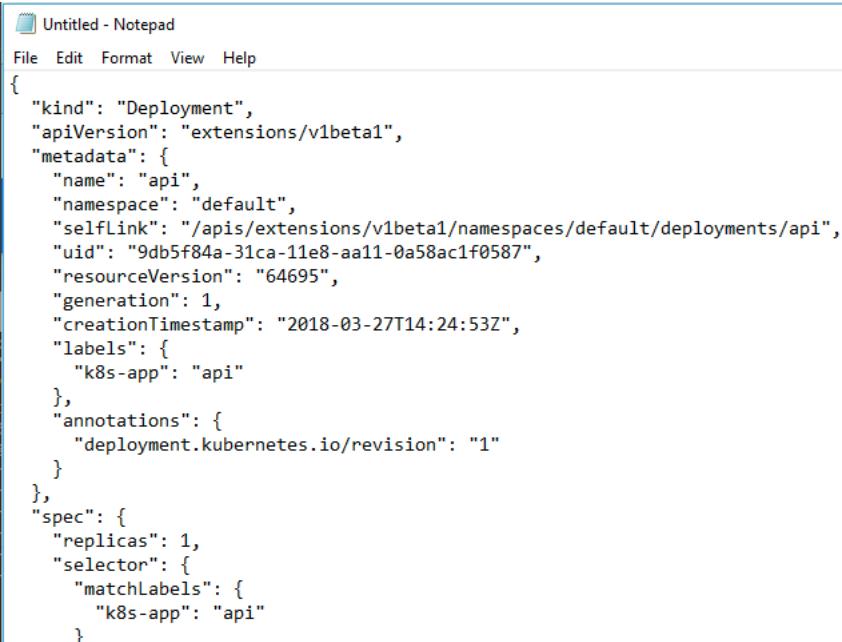


A screenshot of a web-based Kubernetes deployment editor. The main area displays a JSON configuration for a deployment named 'api'. The JSON includes fields like kind, apiVersion, metadata (name, namespace, uid, resourceVersion, generation, creationTimestamp, labels, annotations), and spec (replicas, selector). Below the JSON is a toolbar with three buttons: CANCEL, COPY, and UPDATE.

```
1 [
2   "kind": "Deployment",
3   "apiVersion": "extensions/v1beta1",
4   "metadata": {
5     "name": "api",
6     "namespace": "default",
7     "selfLink": "/apis/extensions/v1beta1/namespaces/default/deployments/api",
8     "uid": "9db5f84a-31ca-11e8-aa11-0a58ac1f0587",
9     "resourceVersion": "64695",
10    "generation": 1,
11    "creationTimestamp": "2018-03-27T14:24:53Z",
12    "labels": {
13      "k8s-app": "api"
14    },
15    "annotations": {
16      "deployment.kubernetes.io/revision": "1"
17    }
18 }
```

CANCEL COPY UPDATE

7. Paste the contents into the text editor of your choice (notepad is shown here, MacOS users can useTextEdit).



A screenshot of a Windows Notepad window titled 'Untitled - Notepad'. The window contains the same JSON configuration for a deployment named 'api' as shown in the previous screenshot. The file menu includes options like File, Edit, Format, View, Help.

```
{ "kind": "Deployment", "apiVersion": "extensions/v1beta1", "metadata": { "name": "api", "namespace": "default", "selfLink": "/apis/extensions/v1beta1/namespaces/default/deployments/api", "uid": "9db5f84a-31ca-11e8-aa11-0a58ac1f0587", "resourceVersion": "64695", "generation": 1, "creationTimestamp": "2018-03-27T14:24:53Z", "labels": { "k8s-app": "api" }, "annotations": { "deployment.kubernetes.io/revision": "1" } }, "spec": { "replicas": 1, "selector": { "matchLabels": { "k8s-app": "api" } } }
```

8. Scroll down about half way to find the node "\$.spec.template.spec.containers[0]" as shown in the screenshot below.

```
{  
  "kind": "Deployment",  
  "apiVersion": "extensions/v1beta1",  
  "metadata": {  
    "name": "api",  
    "namespace": "default",  
    "selfLink": "/apis/extensions/v1beta1/namespaces/default/deployments/api",  
    "uid": "9db5f84a-31ca-11e8-aa11-0a58ac1f0587",  
    "resourceVersion": "64695",  
    "generation": 1,  
    "creationTimestamp": "2018-03-27T14:24:53Z",  
    "labels": {  
      "k8s-app": "api"  
    },  
    "annotations": {  
      "deployment.kubernetes.io/revision": "1"  
    }  
  },  
  "spec": {  
    "replicas": 1,  
    "selector": {  
      "matchLabels": {  
        "k8s-app": "api"  
      }  
    },  
    "template": {  
      "metadata": {  
        "name": "api",  
        "creationTimestamp": null,  
        "labels": {  
          "k8s-app": "api"  
        }  
      },  
      "spec": {  
        "containers": [  
          {  
            "name": "api",  
            "image": "fabmedicalsol.azurecr.io/fabmedical/content-api",  
            "resources": {  
              "requests": {  
                "cpu": "125m",  
                "memory": "128Mi"  
              }  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

9. The containers spec has a single entry for the API container at the moment, you'll see the name of the container is "api" — this is how you know you are looking at the correct container spec.
 - a. Add the following JSON snippet below the "name" property in the container spec.

```
"ports": [  
  {  
    "containerPort": 3001,  
    "hostPort": 3001  
  }  
,
```

- b. Your container spec should now look like this:

```

"spec": {
  "containers": [
    {}
    {
      "name": "api",
      "ports": [
        {
          "containerPort": 3001,
          "hostPort": 3001
        }
      ],
      "image": "fabmedical.azurecr.io/fabmedical/content-api",
      "resources": {
        "requests": {
          "cpu": "125m",
          "memory": "128Mi"
        }
      },
      "terminationMessagePath": "/dev/termination-log",
      "terminationMessagePolicy": "File",
      "imagePullPolicy": "Always",
      "securityContext": {
        "privileged": false
      }
    }
  ],
  "restartPolicy": "Always",
  "terminationGracePeriodSeconds": 30
}

```

10. Set `".spec.replicas"` to 0 so that you can redeploy all the API pods. (This is the outer "deployment" spec).

```

{
  "kind": "Deployment",
  "apiVersion": "extensions/v1beta1",
  "metadata": {
    "name": "api",
    "namespace": "default",
    "selfLink": "/apis/extensions/v1beta1/namespaces/default/deployments/api",
    "uid": "9db5f84a-31ca-11e8-aa11-0a58ac1f0587",
    "resourceVersion": "64695",
    "generation": 1,
    "creationTimestamp": "2018-03-27T14:24:53Z",
    "labels": {
      "k8s-app": "api"
    },
    "annotations": {
      "deployment.kubernetes.io/revision": "1"
    }
  },
  "spec": {
    "replicas": 0, ←
    "selector": {
      "matchLabels": {
        "k8s-app": "api"
      }
    },
    "template": {
      "metadata": {
        "labels": {
          "k8s-app": "api"
        }
      }
    }
  }
}

```

11. Copy the updated JSON document from notepad into the clipboard. Return to the Kubernetes dashboard, which should still be viewing the "api" deployment.

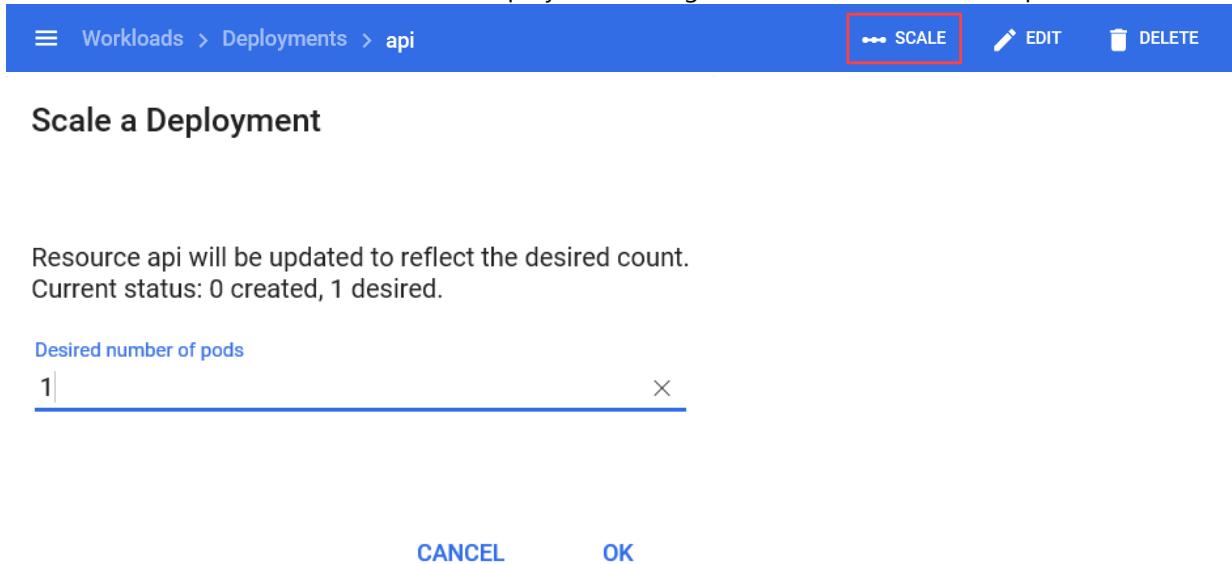
- a. Click Edit.



- b. Paste the updated JSON document.
c. Click Update.



12. Now, select **Scale** and, from the Scale a Deployment dialog, set the desired number of pods to 1 and select **OK**.



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

The screenshot shows the Kubernetes dashboard interface. On the left, a sidebar lists 'Cluster' options: Namespaces, Nodes, Persistent Volumes, Roles, and Storage Classes. Below that, under 'Namespaces', 'default' is selected. The main area is titled 'Overview' and contains a 'Workloads' tab, which is highlighted with a red arrow. Other tabs include Daemon Sets, Deployments (which is currently selected), Jobs, and Pods.

14. 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.

The screenshot shows the 'Workloads' view in the Kubernetes dashboard. The sidebar on the left shows 'Cluster' options: Namespaces, Nodes, Persistent Volumes, Roles, and Storage Classes. Under 'Namespaces', 'default' is selected. The main area has three sections: 'Deployments', 'Pods', and 'Replica Sets'. The 'Deployments' section shows one deployment named 'api' with 1/1 pods, created 39 minutes ago, and using the image 'fabmedical/app:api'. The 'Pods' section shows one pod named 'api-2547917202-glwz7' running on node 'k8s-agent-b882c7f1-0' for 2 minutes. The 'Replica Sets' section shows two replica sets: 'api-2547917202' and 'api-2145491339', both with 1/1 pods, created 8 and 39 minutes ago respectively, and using the image 'fabmedical/app:api'.

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
```

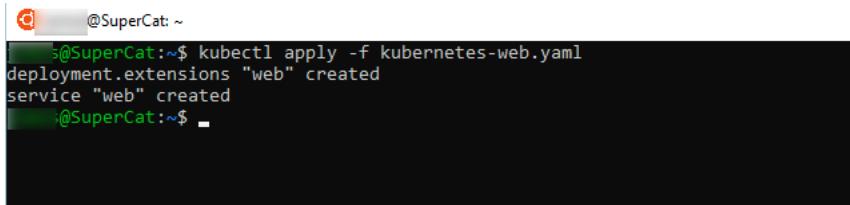
```
        hostPort: 80
        protocol: TCP
      resources:
        requests:
          cpu: 1000m
          memory: 128Mi
      securityContext:
        privileged: false
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    restartPolicy: 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
```



```
@SuperCat: ~
$@SuperCat:~$ kubectl apply -f kubernetes-web.yaml
deployment.extensions "web" created
service "web" created
@SuperCat:~$ -
```

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.

The screenshot shows the Azure Cloud Shell interface with the 'Discovery and load balancing' section selected. Under 'Services', the 'web' service is selected. The 'Details' panel shows the following information:

Name: web	Connection
Namespace: default	Cluster IP: 10.0.127.157
Labels: app: web	Internal endpoints: web:80 TCP web:31297 TCP
Annotations: kubectl.kubernetes.io/last-applied-configuration	
Creation Time: 2018-03-27T20:19 UTC	
Label selector: app: web	
Type: LoadBalancer	
Session Affinity: None	

The 'Endpoints' panel shows the following table:

Host	Ports (Name, Port, Protocol)	Node	Ready
10.244.0.6	web-traffic, 3000, TCP	aks-agentpool-30625191-1	true

The 'Pods' panel shows the following table:

Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
web-86c5891	aks-agentpool-	Running	1	2 minutes	0	70,664,880

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

Name	Node	Status	Restarts	Age
api-2547917202-glw...	k8s-agent-b882c7f1-0	Running	0	43 minutes

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.

Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
web-86c589l	aks-agentpool-30625191-1	Waiting: Create	2	12 minutes	0	68.836 Mi

Liveness probe failed: Get http://10.244.0.6:80/: dial tcp 10.244.0.6:80: getsockopt: connection refused
Failed to pull image "fabmedicalsol.azurecr.io/fabmedical/content-web": rpc error: code = Unknown desc = Error response from daemon: Get https://fabmedicalsol.azurecr.io/v2/fabmedical/content-web/manifests/latest: unauthorized: authentication required
Error syncing pod
Back-off restarting failed container

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.

The screenshot shows the Microsoft Cloud Workshop interface. At the top, there's a blue header bar with the text "Discovery and load balancing > Services". Below this is a sidebar with a tree-like navigation menu. The "Services" option under "Discovery and Load Balancing" is highlighted with a grey background. The main content area is titled "Services" and contains a table with three rows. The columns are: Name, Labels, Cluster IP, Internal endpoints, External endpoints, and Age. The first row has a green checkmark icon next to "web", with labels "app: web" and internal endpoints "web:80 TCP" and "web:32728...". Its external endpoint is "52.174.124..." and it's 6 minutes old. The second row has a green checkmark icon next to "api", with labels "app: api" and internal endpoints "api:3001 TCP" and "api:0 TCP". Its external endpoint is "-" and it's an hour old. The third row has a green checkmark icon next to "kubernetes", with labels "compone..." and "provider:...". Its internal endpoint is "10.0.0.1" and its external endpoint is "-". It's 6 hours old. Each row has a three-dot menu icon on the far right.

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
web	app: web	10.0.179.86	web:80 TCP web:32728...	52.174.124... -	6 minutes an hour
api	app: api	10.0.234.90	api:3001 TCP api:0 TCP	-	6 hours
kubernetes	compone... provider:...	10.0.0.1	kubernetes:... kubernetes:...	-	6 hours

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.

- 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.

The screenshot shows the AKS portal interface. On the left, a sidebar menu lists various cluster components: Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, Namespace (set to default), Overview, Workloads (selected), Daemon Sets, Deployments, Jobs, Pods, Replica Sets (highlighted with a red arrow), and Replication Controllers.

The main content area is titled "Workloads > Replica Sets". It displays a table of Replica Sets:

Name	Labels	Pods	Age	Images
web-3206048963	app: web pod-template-...	1 / 1	11 minutes	fabmedical ...
api-2547917202	app: api pod-template-...	1 / 1	58 minutes	fabmedical ...
api-2145491339	app: api pod-template-...	0 / 0	an hour	fabmedical ...

A red arrow points from the sidebar's "Replica Sets" link to the corresponding table row in the main content area.

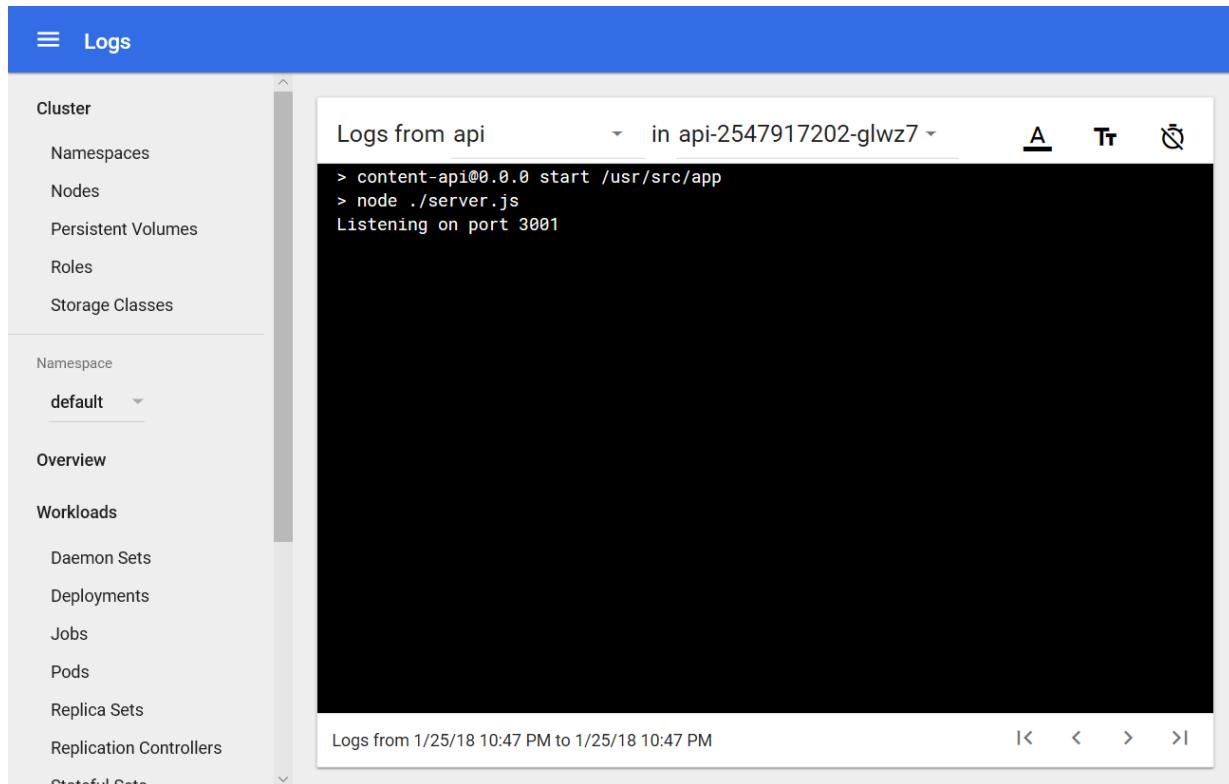
2. You will see some messages indicating that API pods were launched and are running normally.

The screenshot shows the AKS portal interface. On the left, a sidebar menu is open under the 'Workloads' section, with 'Replica Sets' selected. The main content area displays details for a Replica Set named 'api-2547917202'. It includes annotations, creation time (2018-01-25T22:41), selector (app: api, pod-template-hash: 2547917202), images (fabmedical.azurecr.io/fabmedical/content-api), and status (1 running pod). Below this, two tables are shown: 'Pods' and 'Services'. The 'Pods' table lists one pod named 'api-2547917202-qlw...' running on node 'k8s-agent-b882c7f1-0' for 56 minutes. The 'Services' table lists one service named 'api' with internal endpoint 'api:3001' and external endpoint 'api:0 TCP'.

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

This screenshot shows the 'Pods' table from the AKS portal. It lists the same pod as the previous screenshot: 'api-2547917202-qlw...' on node 'k8s-agent-b882c7f1-0'. The 'Logs' icon, which is a grid icon, is highlighted with a red box to indicate it should be selected.

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).

Cluster		Events					
		Message	Source	Sub-object	Count	First seen	Last seen
Namespaces		Successfully assigned web-3206048963-zc7b0 to k8s-agent-b82c7f1-1	default-scheduler	-	1	2018-01-25T23:28 UTC	2018-01-25T23:28 UTC
Nodes							
Persistent Volumes							
Roles							
Storage Classes							
Namespace							
default	▼	MountVolume.S etUp succeeded for volume "default-token-b9kf6"	kubelet k8s-agent-b882c7f1-1	-	1	2018-01-25T23:28 UTC	2018-01-25T23:28 UTC
Overview							
Workloads							
Daemon Sets							
Deployments							
Jobs							
Pods	▼	pulling image "fabmedicalvrapposo.azurecr.io/fabmedicalvrapposo.azurecr.io/fabmedical/content-web"	kubelet k8s-agent-b882c7f1-1	spec.containers {web}	10	2018-01-25T23:28 UTC	2018-01-25T23:51 UTC
Replica Sets							
Replication Controllers							
Stateful Sets							
Discovery and Load Balancing							
Ingresses							
Services							
Config and Storage		⚠ Liveness probe failed: Get http://10.244.1.8:80/: dial tcp 10.244.1.8:80: getsockopt: connection refused	kubelet k8s-agent-b882c7f1-1	spec.containers {web}	19	2018-01-25T23:29 UTC	2018-01-25T23:50 UTC
Config Maps							
Persistent Volume Claims							
Secrets							
About		Killing container with id docker://web:pod 'web-3206048963-zc7b0_default(74341696-0227-11e8-b123-000d223d'	kubelet k8s-agent-	spec.containers {web}	9	2018-01-25T23:29 UTC	2018-01-25T23:51 UTC

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

☰ Workloads > Pods > web-3206048963-zc7b0 EXEC LOGS EDIT DELETE

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

The screenshot shows the AKS Logs interface. On the left, there's a sidebar with navigation links for Cluster (Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes), Namespace (default), Overview, Workloads (Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets), and Logs. The main area is titled "Logs from web" and shows log entries for a pod named "web-3206048963-zc7b0". The logs output are:

```
> application-name@0.0.1 start /usr/src/app
> node server.js
== Using development environment ==
Express server listening on port 3000
```

Below the logs, it says "Logs from 1/26/18 12:16 AM to 1/26/18 12:16 AM". At the bottom right are navigation icons for back, forward, and search.

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.

	Liveness probe failed: Get http://10.244.1.8:80/: dial tcp 10.244.1.8:80: connect refused	kubelet k8s-agents-b882c7f1-1	spec.containers {web}	19	2018-01-25T23:29 UTC	2018-01-25T23:50 UTC
--	---	-------------------------------	-----------------------	----	----------------------	----------------------

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

The screenshot shows the Azure portal interface for managing workloads. In the top navigation bar, 'Workloads > Deployments > web'. On the left sidebar, under 'Deployments', the 'web' deployment is selected. A modal window titled 'Edit a Deployment' is open, showing the deployment's configuration in JSON format. The JSON code includes details like the deployment's kind, metadata, labels, annotations, and its containers. The 'containers' section defines a single container named 'web' with an image from 'fabmedicalsol.azurecr.io'. The 'ports' section is highlighted with a red box around the 'containerPort' entry, which is set to 80. Below the JSON, there are 'CANCEL', 'COPY', and 'UPDATE' buttons.

```

1  {
2    "kind": "Deployment",
3    "apiVersion": "extensions/v1beta1",
4    "metadata": {
5      "name": "web",
6      "namespace": "default",
7      "selfLink": "/apis/extensions/v1beta1/namespaces/default/deployments/web",
8      "uid": "1d818551-31fc-11e8-aa11-0a58ac1f01",
9      "resourceVersion": "95485",
10     "generation": 1,
11     "creationTimestamp": "2018-03-27T20:19:13Z",
12     "labels": {
13       "app": "web"
14     },
15     "annotations": {
16       "deployment.kubernetes.io/revision": "1"
17       "kubectl.kubernetes.io/last-applied-com-

```

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.

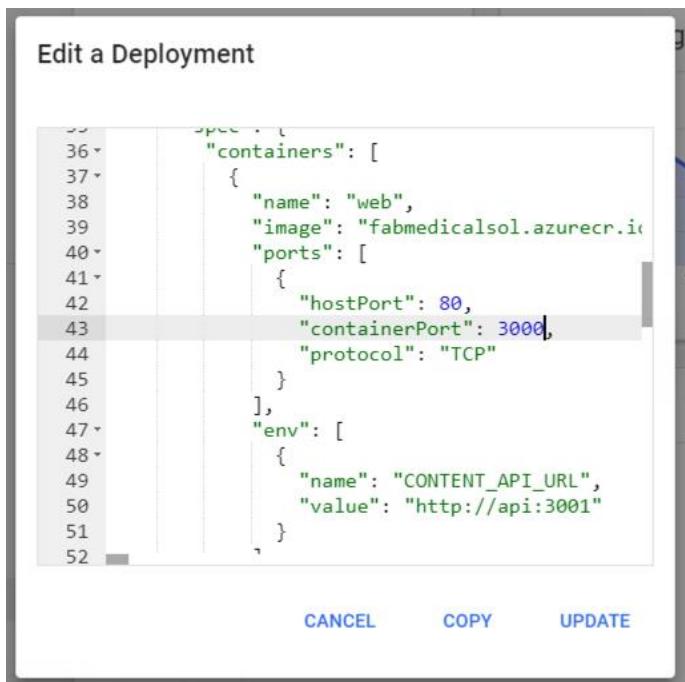
This screenshot shows the 'Edit a Deployment' dialog with the JSON configuration for the 'web' deployment. The 'ports' section is highlighted with a red box around the 'containerPort' entry, which is currently set to 80. The JSON code also shows other configurations like the container's name ('web'), image ('fabmedicalsol.azurecr.io'), and environment variables ('CONTENT_API_URL'). Below the JSON, there are 'CANCEL', 'COPY', and 'UPDATE' buttons.

```

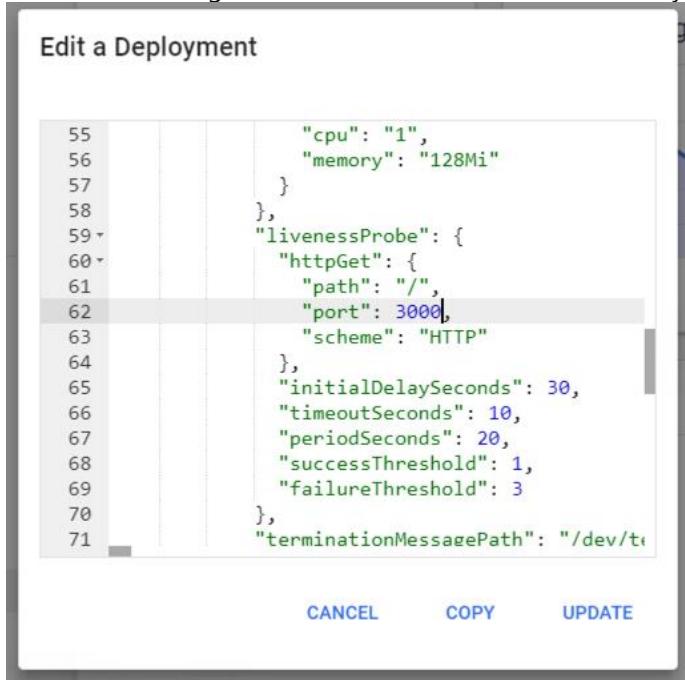
36  {
37    "spec": {
38      "containers": [
39        {
40          "name": "web",
41          "image": "fabmedicalsol.azurecr.io/web:1.0",
42          "ports": [
43            {
44              "hostPort": 80,
45              "containerPort": 80,
46              "protocol": "TCP"
47            }
48          ],
49          "env": [
50            {
51              "name": "CONTENT_API_URL",
52              "value": "http://api:3001"
53            }
54          ]
55        }
56      ]
57    }
58  }

```

12. 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 replica set 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.

Pods					
Name	Node	Status	Restarts	Age	
web-1272019779-f9...	k8s-agent-b882c7f1-0	Running	0	-	

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.

Services						
Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age	
web	app: web	10.0.179.86	web:80 TCP web:32728 TCP	52.174.124.171:80	an hour	
api	app: api	10.0.234.90	api:3001 TCP api:0 TCP	-	2 hours	
kubernetes	component: apis... provider: kubern...	10.0.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	7 hours	

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.



CONTOSO NEURO 2017

Speakers Sessions

SEPTEMBER 14-17, 2017
Monterey Conference Center
Monterey, California

speakers

[View recordings](#)

UC San Francisco University School of Medicine, Monterey, California

CONTOSO NEURO 2017

Speakers Sessions

SEPTEMBER 14-17, 2017
Monterey Conference Center
Monterey, California

sessions

[View recordings](#)

UC San Francisco University School of Medicine, Monterey, California

Exercise 3: 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**.

Scale a Deployment

Resource api will be updated to reflect the desired count.
Current status: 1 created, 2 desired.

Desired number of pods

 X

CANCEL 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.

Workloads > Replica Sets > api-2547917202

Cluster

- Namespaces
- Nodes
- Persistent Volumes
- Roles
- Storage Classes

Namespace

default

Overview

Workloads

- Daemon Sets
- Deployments
- Jobs
- Pods** (highlighted)
- Replica Sets** (highlighted)
- Replication Controllers
- Stateful Sets

Pods

Name	Node	Status	Restarts	Age
api-2547917202-51g...	k8s-agent-b882c7f1-1	Waiting: Contai...	0	3 seconds
api-2547917202-glw...	k8s-agent-b882c7f1-0	Running	0	13 hours

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").

Workloads > Deployments

Storage Classes

Namespace

default

Overview

Workloads

Deployments

Name	Labels	Pods	Age	Images
web	app: web	1 / 1	12 hours	fabmedical ...
api	app: api	1 / 2	13 hours	fabmedical ...

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:
- One deployment and one replica set are each healthy for the api service.
 - One replica set is healthy for the web service.
 - 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.
- Navigate to the /stats.html page. You'll see information about the environment including:
 - webTaskId: the task identifier for the web service instance
 - taskId: the task identifier for the API service instance
 - hostName: the hostname identifier for the API service instance
 - pid: the process id for the API service instance
 - mem: some memory indicators returned from the API service instance
 - counters: counters for the service itself, as returned by the API service instance
 - 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**.

Scale a Deployment

Resource api will be updated to reflect the desired count.
Current status: 2 created, 4 desired.

Desired number of pods

 X

CANCEL 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.

The screenshot shows the 'Discovery and load balancing > Services > api' view. In the 'Details' section, it lists the service name as 'api', namespace as 'default', labels as 'app: api', creation time as '2018-01-25T22:10', and type as 'ClusterIP'. The 'Connection' section shows the cluster IP as 10.0.234.90 and internal endpoints as 'api:3001 TCP' and 'api:0 TCP'. Below this, the 'Pods' section displays four pods: 'api-2547917202-51gjc' (Running, Node k8s-agent-b882c7f1-1), 'api-2547917202-5xcjk' (Pending, reason: 'No nodes are available that match all of the following predicates: PodFitsHostPorts (2), PodToleratesNodeTaints (1)'), 'api-2547917202-glwz7' (Running, Node k8s-agent-b882c7f1-0), and 'api-2547917202-kjrp7' (Pending, reason: 'No nodes are available that match all of the following predicates: PodFitsHostPorts (2), PodToleratesNodeTaints (1)').

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

The screenshot shows the 'Workloads' dashboard for the 'default' namespace. The 'Workloads' section is selected, showing options for Daemon Sets, Deployments, and Jobs. The 'Deployments' section shows two entries: 'web' (Running, 1/1 pods) and 'api' (Pending, 2/4 pods). A warning message for the 'api' deployment states: 'No nodes are available that match all of the following predicates: PodFitsHostPorts (2), PodToleratesNodeTaints (1)'.

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.

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

The screenshot shows the 'Workloads' dashboard for the 'default' namespace. The 'Workloads' section is selected, showing options for Daemon Sets, Deployments, and Jobs. The 'Deployments' section shows two entries: 'web' (Running, 1/1 pods) and 'api' (Running, 2/2 pods). The warning message for the 'api' deployment has disappeared.

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.

- 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.

Services						
Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age	⋮
web	app: web	10.0.179.86	web:80 TCP web:32728 TCP	52.174.124.171:80	an hour	⋮
api	app: api	10.0.234.90	api:3001 TCP api:0 TCP	-	2 hours	⋮
kubernetes	component: apis... provider: kubern...	10.0.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	7 hours	⋮

CONTOSO NEURO 2017

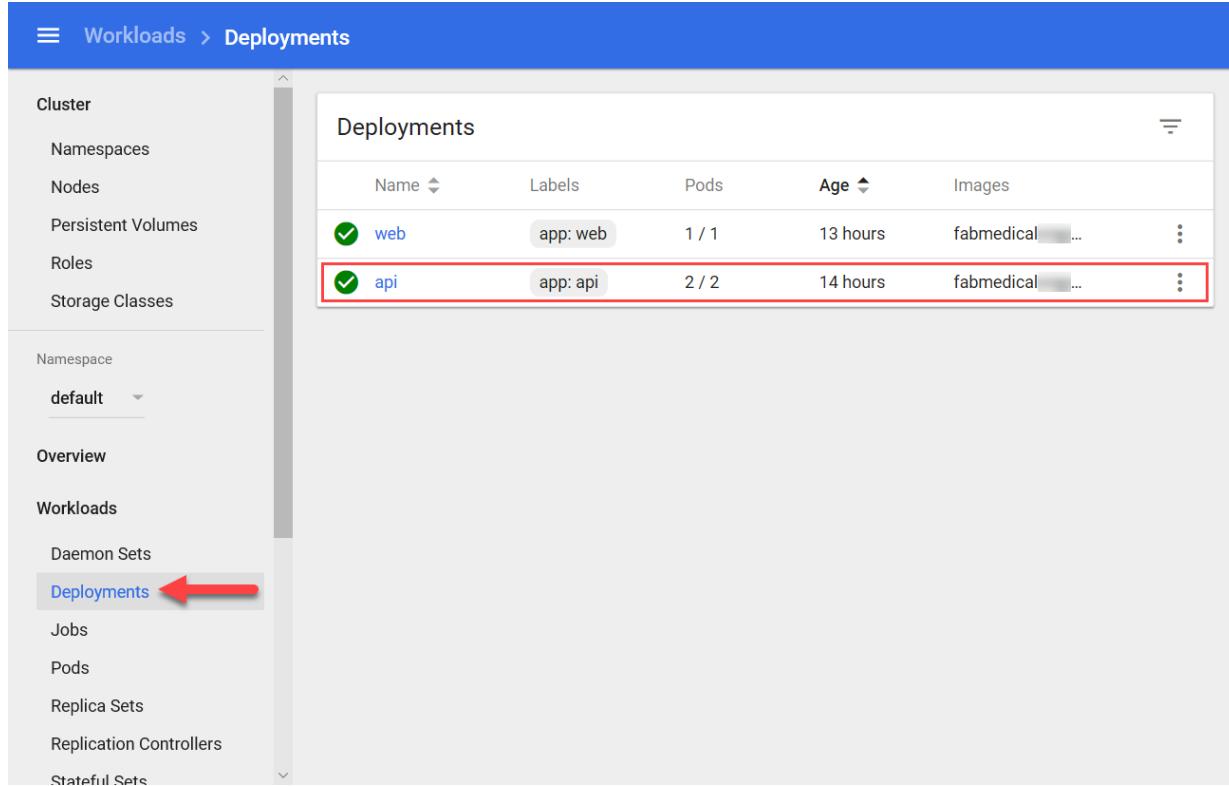
Speakers Sessions



Stats

webTaskId	18
taskId	18
hostName	api-2547917202-glwz7
pid	18
mem	{"rss":36429824,"heapTotal":19582720,"heapUsed":13460680,"external":232284}
counters	{"stats":3,"speakers":2,"sessions":1}
uptime	49422.159

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



The screenshot shows the 'Workloads > Deployments' view. On the left, a navigation sidebar lists 'Cluster' (Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes), 'Namespace' (default), 'Overview', 'Workloads' (Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets), and a 'More' section. A red arrow points to the 'Deployments' link under 'Workloads'. The main area displays a table titled 'Deployments' with columns: Name, Labels, Pods, Age, and Images. It shows two entries: 'web' (1 pod, 13 hours old, fabmedical) and 'api' (2 pods, 14 hours old, fabmedical). The 'api' row is highlighted with a red border.

Name	Labels	Pods	Age	Images
web	app: web	1 / 1	13 hours	fabmedical ...
api	app: api	2 / 2	14 hours	fabmedical ...

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.

Details

- Name: api-2547917202
- Namespace: default
- Labels: app: api, pod-template-hash: 2547917202
- Annotations: deployment.kubernetes.io/desired-replicas: 4, deployment.kubernetes.io/max-replicas: 5, deployment.kubernetes.io/revision: 2
- Creation time: 2018-01-25T22:41
- Selector: app: api, pod-template-hash: 2547917202
- Images: fabmedical.azurecr.io/fabmedical/content-api

Status

- Pods: 4 created, 4 desired
- Pods status: 2 pending, 2 running

Name	Node	Status	Restarts	Age
api-2547917202-4x1...		Pending	0	7 seconds
api-2547917202-51g...	k8s-agent-b882c7f1-1	Running	0	39 minutes
api-2547917202-glw...	k8s-agent-b882c7f1-0	Running	0	13 hours
api-2547917202-hjzsf		Pending	0	7 seconds

- 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.

webTaskId	18
taskId	18
hostName	api-2547917202-glwz7
pid	18
mem	{"rss":36429824,"heapTotal":19582720,"heapUsed":13544928,"external":232284}
counters	{"stats":6,"speakers":2,"sessions":1}
uptime	50570.163

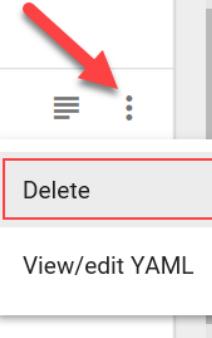
webTaskId	18
taskId	18
hostName	api-2547917202-51gjc
pid	18
mem	{"rss":36376576,"heapTotal":19582720,"heapUsed":13479320,"external":237370}
counters	{"stats":5,"speakers":3,"sessions":1}
uptime	3012.69

- 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.
- 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.

Pods					
Name	Node	Status	Restarts	Age	
! api-2547917202-4x1...		Pending	0	7 seconds	⋮
		No nodes are available that match all of the following predicates:: PodFitsHostPorts (2), PodToleratesNodeTaints (1).			
✓ api-2547917202-51g...	k8s-agent-b882c7f1-1	Running	0	39 minutes	⋮
✓ api-2547917202-glw...	k8s-agent-b882c7f1-0	Running	0	13 hours	⋮
! api-2547917202-hjzs...		Pending	0	7 seconds	⋮
		No nodes are available that match all of the following predicates:: PodFitsHostPorts (2), PodToleratesNodeTaints (1).			

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.

Pods					
Name	Node	Status	Restarts	Age	
! api-2547917202-4x1...		Pending	0	30 minutes	⋮
		No nodes are available that match all of the following predicates:: PodFitsHostPorts (2), PodToleratesNodeTaints (1).			
✓ api-2547917202-51g...	k8s-agent-b882c7f1-1	Running	0	an hour	⋮
✓ api-2547917202-glw...	k8s-agent-b882c7f1-0	Running	0	14 hours	⋮
! api-2547917202-hjzs...		Pending	0	30 minutes	⋮
		No nodes are available that match all of the following predicates:: PodFitsHostPorts (2), PodToleratesNodeTaints (1).			



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.

Pods						
Name	Node	Status	Restarts	Age		
api-2547917202-4x1...	k8s-agent-b882c7f1-1	Running	0	37 minutes		
api-2547917202-glw...	k8s-agent-b882c7f1-0	Running	0	14 hours		
api-2547917202-hjzsf		Pending	0	37 minutes		
	No nodes are available that match all of the following predicates:: PodFitsHostPorts (2), PodToleratesNodeTaints (1).					
api-2547917202-nqq...		Pending	0	58 seconds		
	No nodes are available that match all of the following predicates:: PodFitsHostPorts (2), PodToleratesNodeTaints (1).					

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.

Scale a Deployment

Resource api will be updated to reflect the desired count.
Current status: 4 created, 1 desired.

Desired number of pods

×

[CANCEL](#)
[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.

Cluster
Namespaces
Nodes
Persistent Volumes
Roles
Storage Classes

Namespace
default

Overview

Workloads
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets

Details

Name: api-2547917202
Namespace: default
Labels: app: api, pod-template-hash: 2547917202
Annotations: deployment.kubernetes.io/desired-replicas: 1, deployment.kubernetes.io/max-replicas: 2, deployment.kubernetes.io/revision: 2
Creation time: 2018-01-25T22:41
Selector: app: api, pod-template-hash: 2547917202
Images: fabmedical.azurecr.io/fabmedical/content-api

Status

Pods: 1 running

Pods

Name	Node	Status	Restarts	Age
api-2547917202-glw...	k8s-agent-b882c7f1-0	Running	0	14 hours

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

Cluster
Namespaces
Nodes
Persistent Volumes
Roles
Storage Classes

Namespace
default

Overview

Workloads
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets

Deployments

Name	Labels	Pods	Age	Images
web	app: web	1 / 1	14 hours	fabmedical...
api	app: api	1 / 1	15 hours	fabmedical...

Pods

Name	Node	Status	Restarts	Age
web-1272019779-f9...	k8s-agent-b882c7f1-0	Running	0	12 hours
api-2547917202-glw...	k8s-agent-b882c7f1-0	Running	0	14 hours

Replica Sets

Name	Labels	Pods	Age	Images
web-1272019779	app: web, pod-template...	1 / 1	12 hours	fabmedical...

Exercise 4: 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 AKS 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.

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
web	app: web	10.0.179.86	web:80 TCP web:32728 TCP	52.174.124.171:80	16 hours
api	app: api	10.0.234.90	api:3001 TCP api:32168 TCP	-	18 hours
kubernetes	component: apis... provider: kubern...	10.0.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	23 hours

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
```

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
web	app: web	10.0.179.86	web:80 TCP web:32728 TCP	52.174.124.171:80	16 hours
api	app: api	10.0.234.90	api:3001 TCP api:32168 TCP	40.68.255.235:3001	18 hours
kubernetes	component: apis... provider: kubern...	10.0.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	23 hours



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.

- From the navigation menu select Deployments under Workloads. From the view's Deployments list select the API deployment.
- Select Edit.
- From the Edit a Deployment dialog, do the following:
 - Scroll to the first spec node that describes replicas as shown in the screenshot. Set the value for replicas to 4.
 - 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 port mapping.

The screenshot shows the 'Edit a Deployment' interface with the following YAML code:

```
19+ "spec": {
20+   "replicas": 4,
21+   "selector": {
22+     "matchLabels": {
23+       "k8s-app": "api"
24+     }
25+   },
26+   "template": {
27+     "metadata": {
28+       "name": "api",
29+       "creationTimestamp": null,
30+       "labels": {
31+         "k8s-app": "api"
32+       }
33+     },
34+     "spec": {
35+       "containers": [
36+         {
37+           "name": "api",
38+           "image": "fabmedical.azurecr.io/fabmedical/content-ap
39+           "ports": [
40+             {
41+               "containerPort": 3001,
42+               "protocol": "TCP"
43+             }
44+           ],
45+           "resources": {
```

Two sections of the code are highlighted with red boxes:

- The first red box highlights the `"replicas": 4` line.
- The second red box highlights the `"ports": [...]` section under the container definition.

At the bottom of the interface, there are three buttons: CANCEL, COPY, and UPDATE.

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.

The screenshot shows the AKS portal interface. On the left, there's a navigation sidebar with sections for Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, and a Namespace dropdown set to 'default'. Below these are sections for Overview, Workloads, and a list of workloads: Daemon Sets, Deployments, Jobs, Pods, **Replica Sets** (which is selected), Replication Controllers, and Stateful Sets. The main content area is titled 'Workloads > Replica Sets > api-4178426672'. It shows detailed information for the replica set: Labels (app: api, pod-template-hash: 4178426672), Annotations (deployment.kubernetes.io/desired-relicas: 4, deployment.kubernetes.io/max-relicas: 5, deployment.kubernetes.io/revision: 3), Creation time (2018-01-26T17:06), Selector (app: api, pod-template-hash: 4178426672), Images (fabmedical.azurecr.io/fabmedical/content-api), and Status (4 running pods). Below this is a table titled 'Pods' with columns: Name, Node, Status, Restarts, and Age. The table lists four pods, each with a green checkmark icon: api-4178426672-45lxx, api-4178426672-76l..., api-4178426672-sph..., and api-4178426672-vs... All pods are listed as Running with 0 restarts and 5 minutes age.

Name	Node	Status	Restarts	Age	⋮
api-4178426672-45lxx	k8s-agent-b882c7f1-1	Running	0	5 minutes	⋮
api-4178426672-76l...	k8s-agent-b882c7f1-1	Running	0	5 minutes	⋮
api-4178426672-sph...	k8s-agent-b882c7f1-0	Running	0	5 minutes	⋮
api-4178426672-vs...	k8s-agent-b882c7f1-1	Running	0	5 minutes	⋮

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 from the service port mapping.
 - c. Select **Update**.

```
8   "uid": "9db90469-31ca-11e8-aa11-0a58ac1f0587",
9   "resourceVersion": "99006",
10  "creationTimestamp": "2018-03-27T14:24:53Z",
11  "labels": {
12    "k8s-app": "api"
13  }
14 },
15 "spec": {
16   "ports": [
17     {
18       "name": "tcp-3001-3001-d4w9n",
19       "protocol": "TCP",
20       "port": 3001,
21       "targetPort": 3001
22     }
23   ],
24   "selector": {
25     "k8s-app": "api"
26   },
27   "clusterIP": "10.0.163.7",
28   "type": "ClusterIP",
29   "sessionAffinity": "None",
30   "externalTrafficPolicy": "Cluster"
31 },
32 "status": {
33   "loadBalancer": {
34     "ingress": [
35   ]
36   }
37 }
```

CANCEL COPY 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 port mapping.



```
app . web
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
```

```
        }
      },
      "template": {
        "metadata": {
          "name": "web",
          "creationTimestamp": null,
          "labels": {
            "app": "web"
          }
        },
        "spec": {
          "containers": [
            {
              "name": "web",
              "image": "Tabmedicalsol.azurecr.io/tbmedical/content-we
              "ports": [
                {
                  "containerPort": 3000,
                  "protocol": "TCP"
                }
              ],
              "env": [
                {
                  "name": "CONTENT_API_URL",
                  "value": "http://api:3001"
                }
              ]
            }
          ]
        }
      }
    }
```

CANCEL COPY UPDATE

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.

The screenshot shows the 'Workloads > Deployments > web' view. The left sidebar under 'Workloads' has 'Deployments' selected. The main area shows the 'Details' section for the 'web' deployment, which includes fields like Name: web, Namespace: default, Labels: app: web, Selector: app: web, Strategy: RollingUpdate, Min ready seconds: 0, Revision history limit: Not set, Rolling update strategy: Max surge: 1, Max unavailable: 1, and Status: 1 updated, 1 total, 1 available, 3 unavailable. Below this is the 'New Replica Set' section, which lists a single pod named 'web-3670957729' with labels app: web and pod-template... It shows 2 / 4 pods, an age of 3 minutes, and an image of fabmedical. A red box highlights an error message: 'No nodes are available that match all of the following predicates:: Insufficient cpu (2), PodToleratesNodeTaints (1)'.

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".

```

32      "app": "web"
33    },
34  },
35  "spec": {
36    "containers": [
37      {
38        "name": "web",
39        "image": "fabmedicalsol.azurecr.io/fabmedical/content-w
40        "ports": [
41          {
42            "containerPort": 3000,
43            "protocol": "TCP"
44          }
45        ],
46        "env": [
47          {
48            "name": "CONTENT_API_URL",
49            "value": "http://api:3001"
50          }
51        ],
52        "resources": {
53          "requests": {
54            "cpu": "125m",
55            "memory": "128Mi"
56          }
57        },
58        "livenessProbe": {
59          ...
        }
      }
    ]
  }
}

```

CANCEL COPY UPDATE

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.

Pods						
Name	Node	Status	Restarts	Age		
web-120118169-0nlfh	k8s-agent-b882c7f1-1	Running	0	a minute		
web-120118169-86lrj	k8s-agent-b882c7f1-1	Running	0	a minute		
web-120118169-fzztr	k8s-agent-b882c7f1-1	Running	0	a minute		
web-120118169-rf06c	k8s-agent-b882c7f1-0	Running	0	a minute		

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.

Name	Labels	Pods	Age	Images
web-250784948	app: web pod-template...	0 / 2	0 seconds	fabmedical ...
web-3594286523	app: web pod-template...	4 / 3	a minute	fabmedical ...

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.



webTaskId	web-250784948-g9fk4
taskId	18
hostName	api-4178426672-f0dp8
pid	18
mem	{"rss":37785600,"heapTotal":19582720,"heapUsed":13256432,"external":211680}
counters	{"stats":4,"speakers":0,"sessions":0}
uptime	2559.844

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"
```