Two-tier Nodejs App migration on Azure Container: Lab Guide

Overview

This guide will help you in migrating On-premises two-tier Nodejs App with MongoDB to Container and PaaS database on Azure.

Content

Lab 1: Getting Started with Azure

Lab Overview

In this lab, you will be deploying pre-requisite infrastructure which is simulation of on-premises two-tier Nodejs app with MongoDB database.

Prerequisites

- Windows or a Mac machine with HTML5 supported browser such as Microsoft Edge, Internet Explorer, Chrome or Firefox.
- Putty.

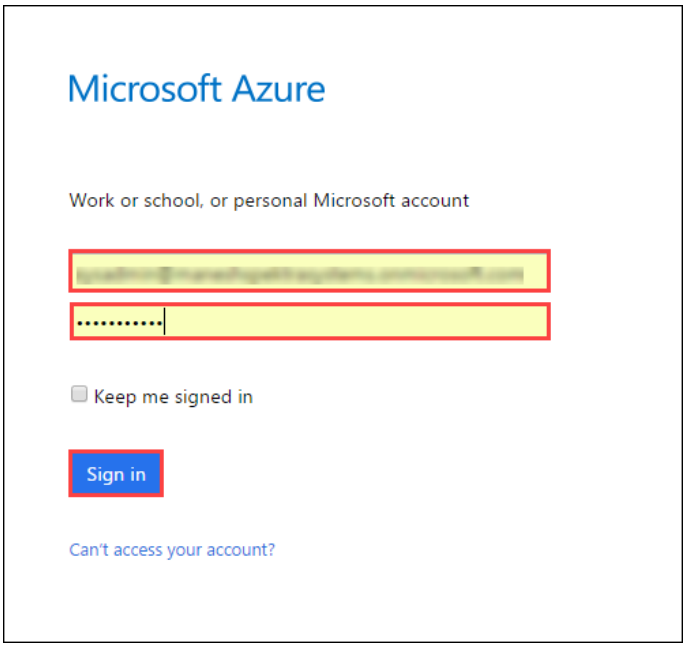
Time Estimate

20 minutes

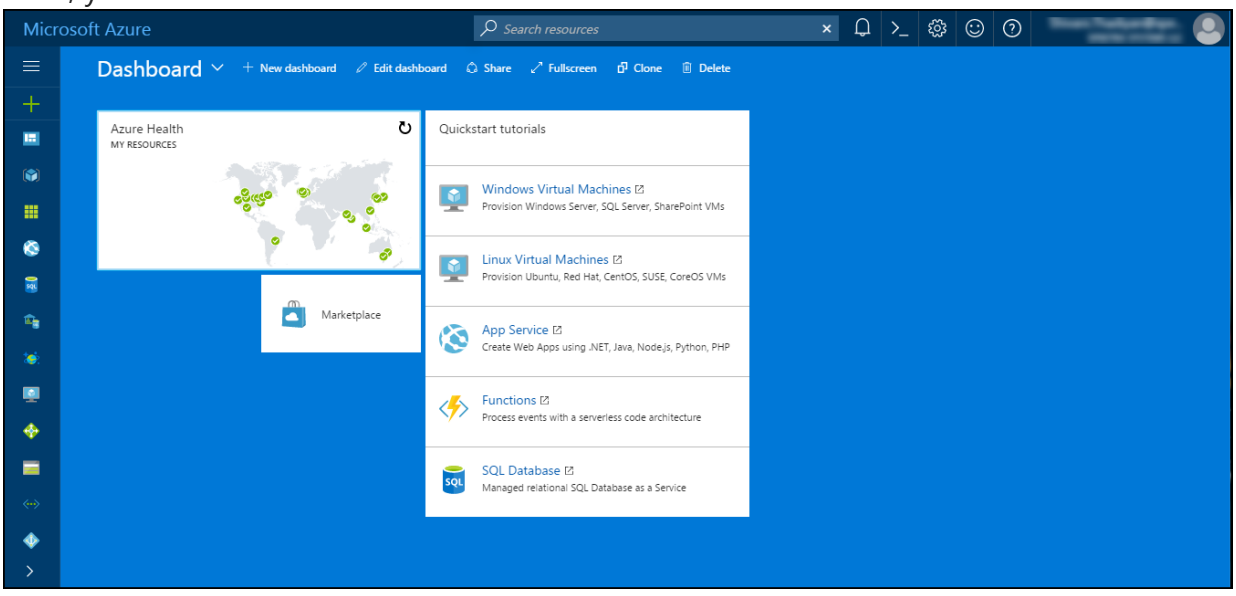
Exercise 1: Deploy Pre-requisite On-premises template

In this exercise, you will log into the Azure Portal using your Azure credentials.

1. **Launch** a browser and **Navigate** to **https://portal.azure.com**. Provide your Azure login credentials and click on **Sign In**.



2. Now, you will be directed to the Azure Dashboard.



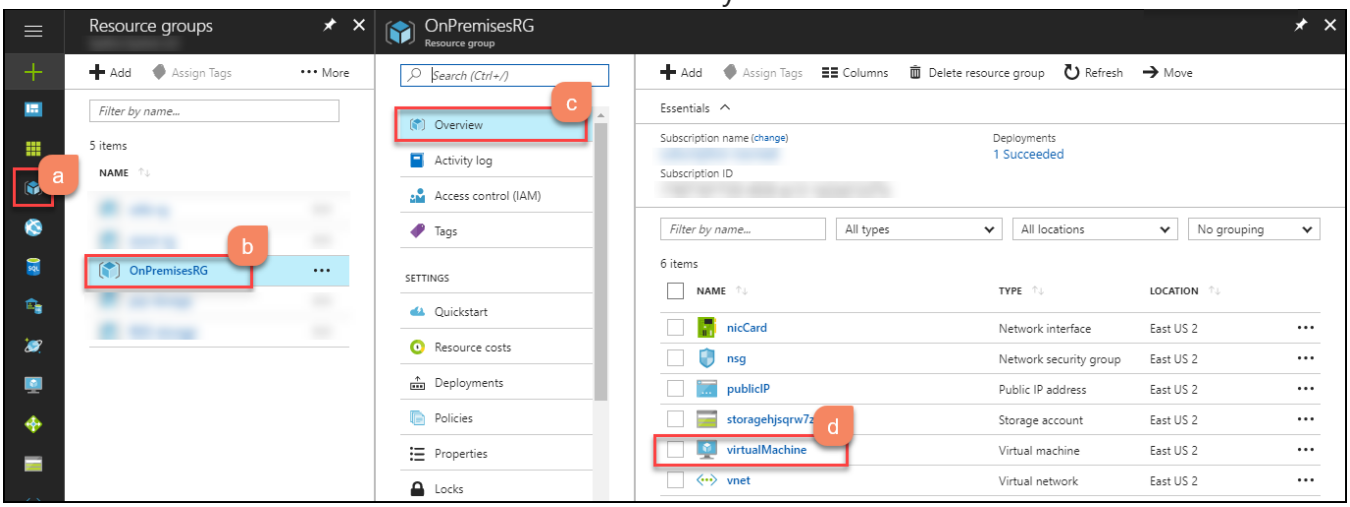
3. Open github URL (https://github.com/SpektraSystems/2-Tier-nodejsapp-migration-to-containers-on-Azure) and click on deploy to azure to create pre-requisite simulation of on-premises two tier infrastructures.
    a. *ResourceGroup:* **OnPremisesRG (**any valid name**)**
    b. *Location:* **East US**(any location)
    c. *Admin User:* **demouser** (any username of your choice)
    d. *Admin Password:* *< Valid Password>*

    Accepts terms and conditions and click on **Purchase** button.

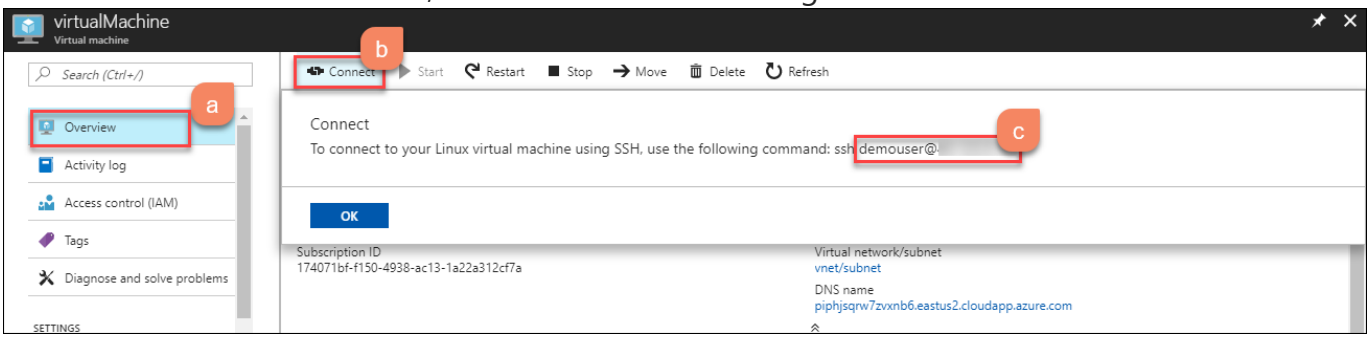
Exercise 2: Verify the DB and WebApp

In this exercise, we will login to VM which is pre-deployed as a part of lab. Login to VM with the credentials provided in mail.

1. In Azure portal, click on Resource Group which contains the pre-deployed on-premises infrastructure then click on **Overview** tab and finally on VM.

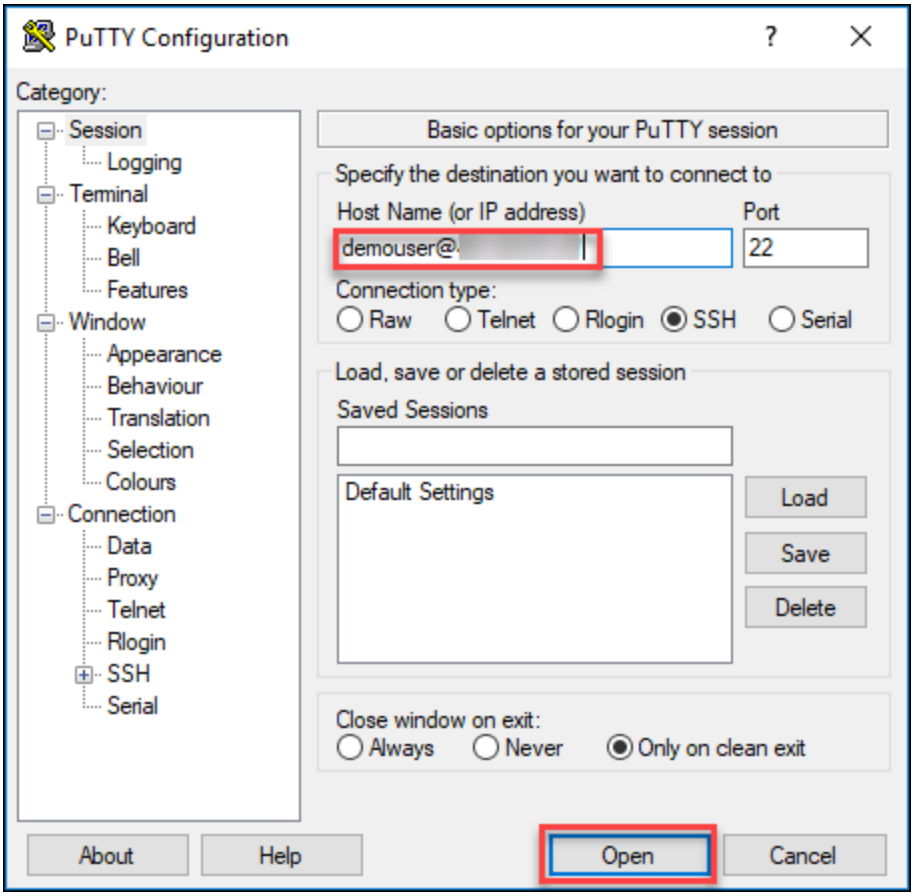


2. In overview section, click on **Connect** button. It will show the username with IP address. Copy that username with IP address, we will use it for connectig the VM.

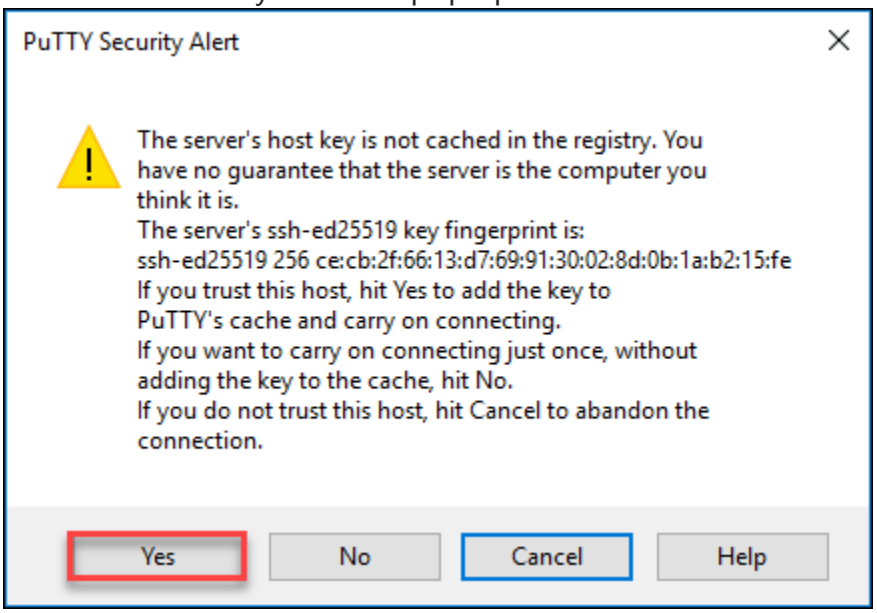


3. Now run putty.exe from your PC.

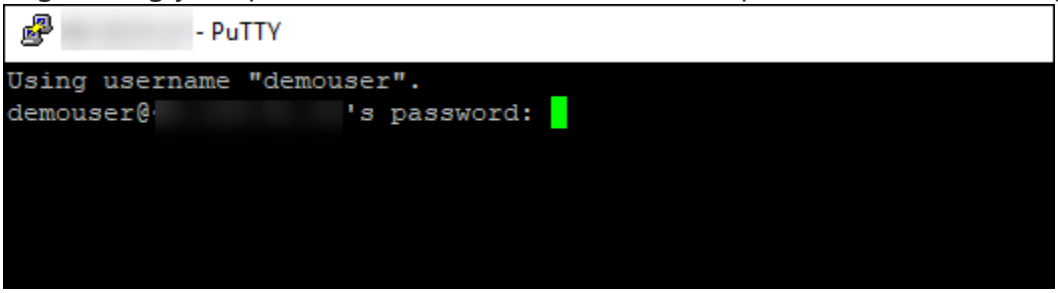
4. This is the application window that pops up when you run putty.exe. Paste the Username with Public IP address of the VM that you copied in step 2 to the Host Name (or IP address) box of the putty. Port will be 22 by default. Click on **Open**.



5. The PuTTY Security Alert will pop up. Click on **Yes**.

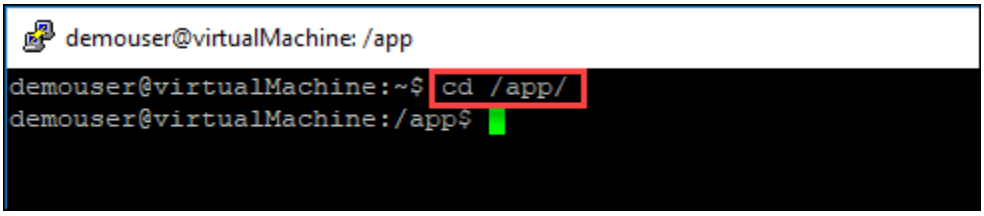


6. Login using your password for the **Virtual Machine** provided while creating virtual machine.



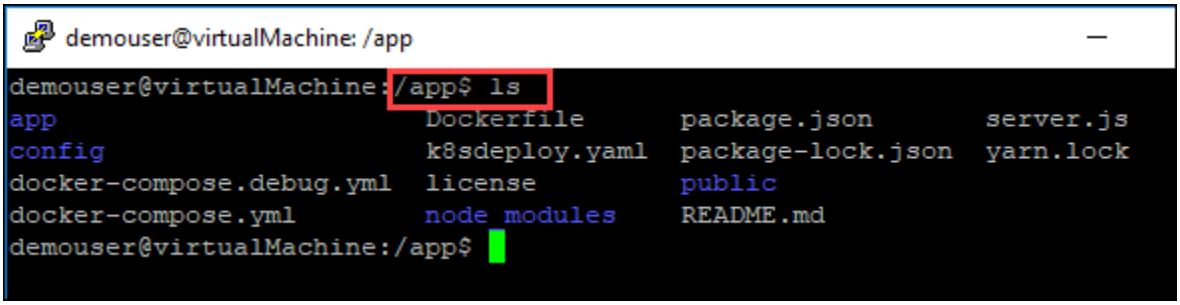
7. Run the following command for changing directory to application folder where we have Nodejs application.

```
cd /app/
```



8. List the files on app folder using following command.

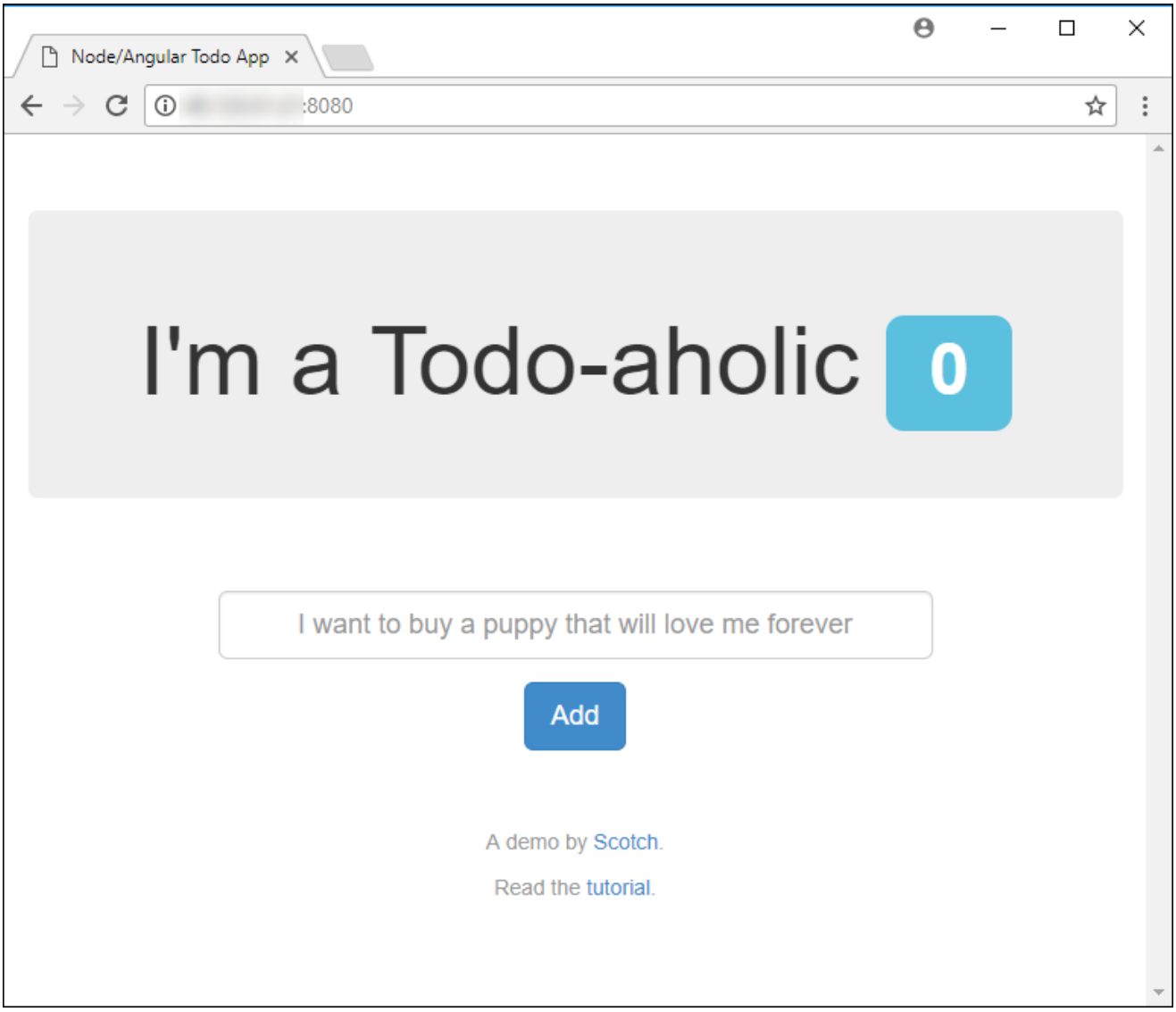
```
ls
```



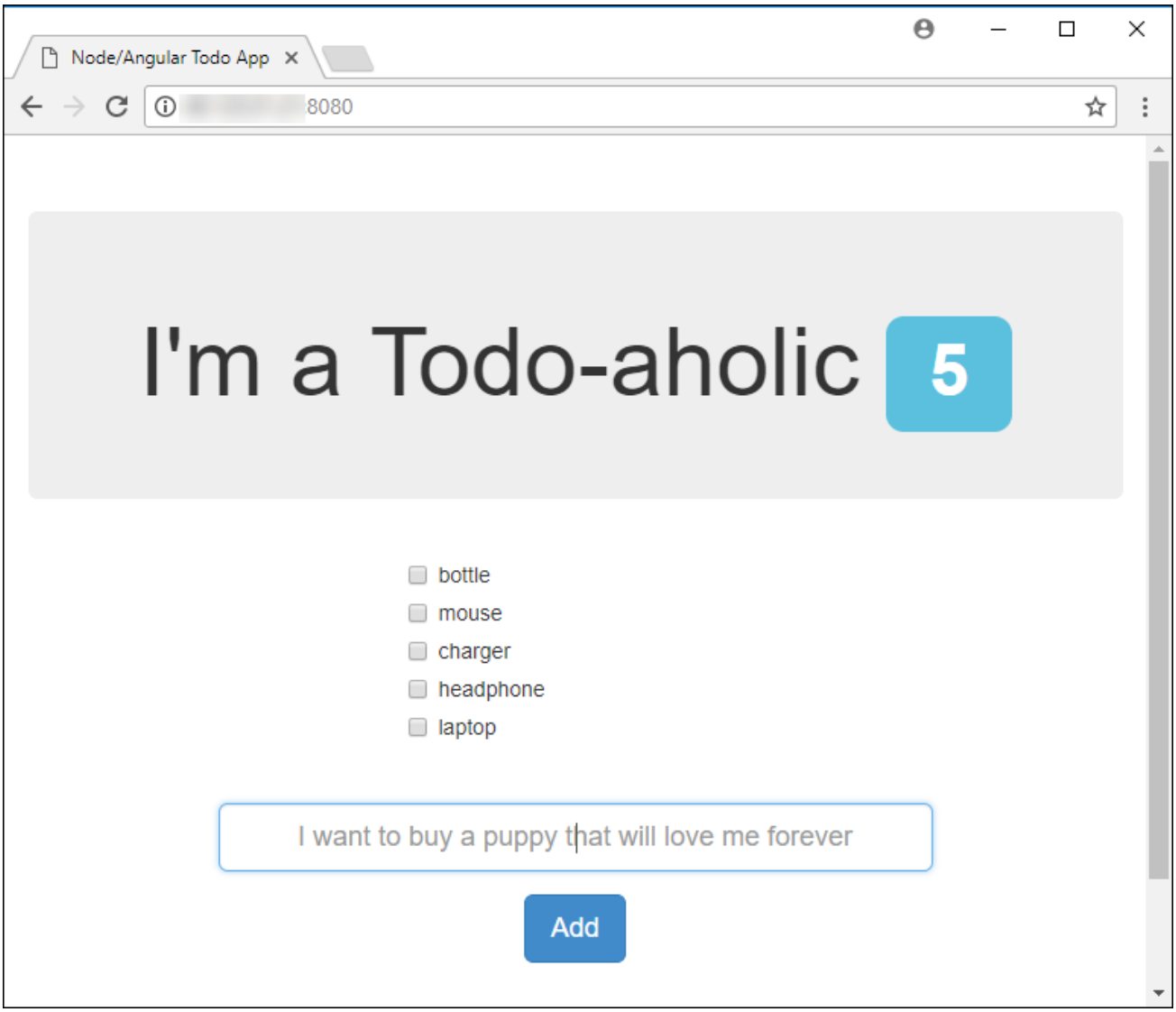
9. Run the commad for starting the Nodejs application using below command.

```
npm start
```

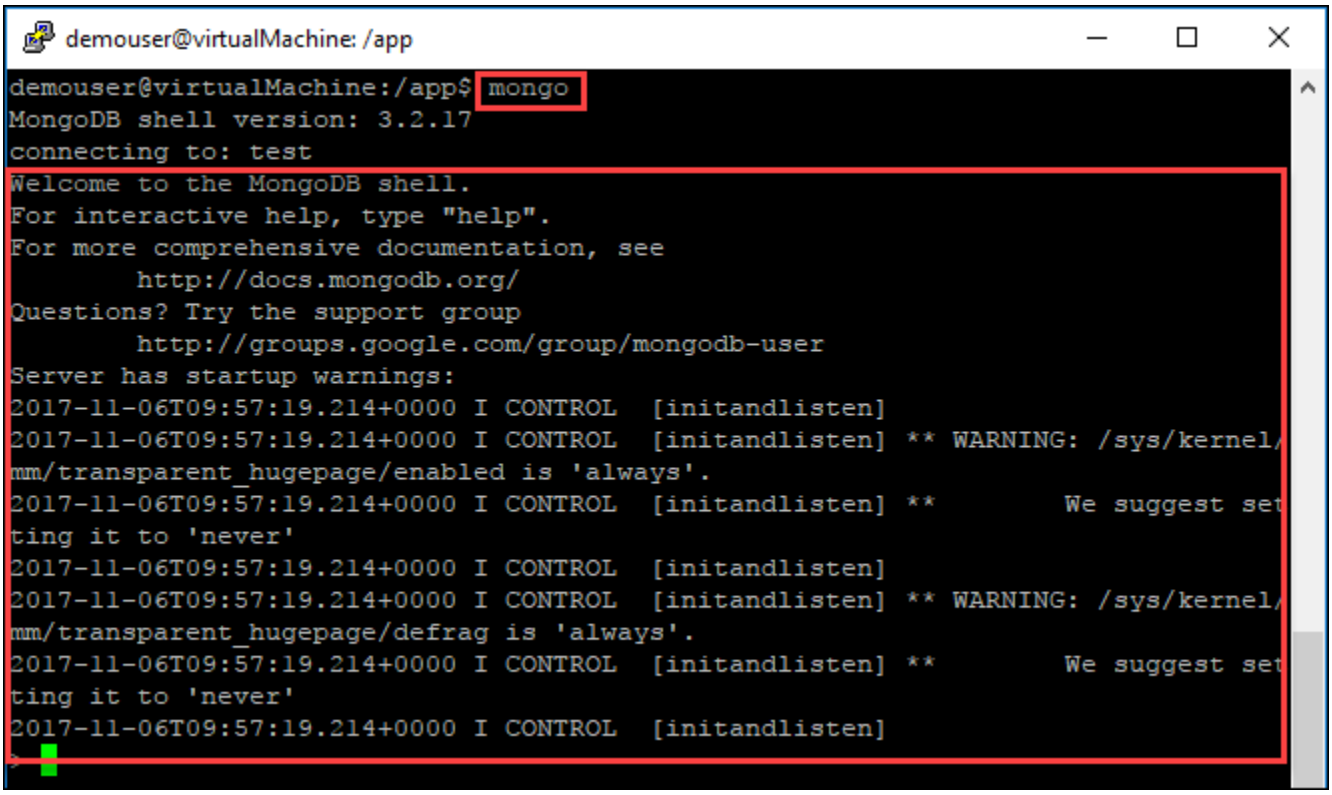
10. Open the browser and browse the Public IP address with port 8080 which will show the Node.js application is running.



11. Create some record in that application as shown below.



12. You can verify the mongoDB database while running the **mongo** as shown below:



13. You can also verify the records in mongoDB database while running the following commands. Copy the **db_name(**meanstacktutorials**)** and **collection_name(**todos**)** for future use.

```
show dbs
use <db_name>
show collections
db.<collection_name>.find()
```

Lab 2: Two-tier Nodejs App migration

Lab Overview

In this lab, you will:

- Create Azure Cosmos DB (MongoDB)
- Export on-premises database and import to Azure Cosmos DB (MongoDB)
- Create Docker Host on Azure
- Build and Run Nodejs application on container

Prerequisites

- Windows or a Mac machine with HTML5 supported browser such as Microsoft Edge, Internet Explorer, Chrome or Firefox.
- Putty client.
- Lab 1 must be completed.

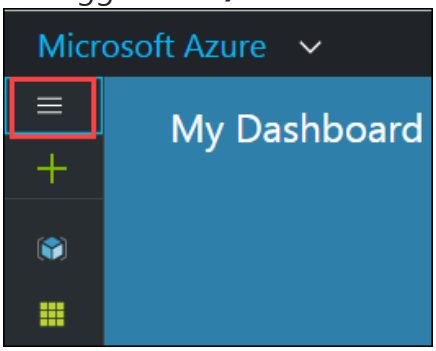
Time Estimate

60 minutes

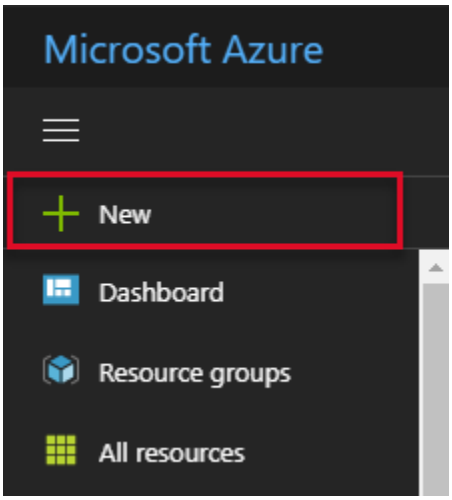
Exercise 1: Deploy Azure Cosmos DB for migration

In this exercise, you will deploy Azure Cosmos database which is required for migration of on-premises database to Azure PaaS DB.

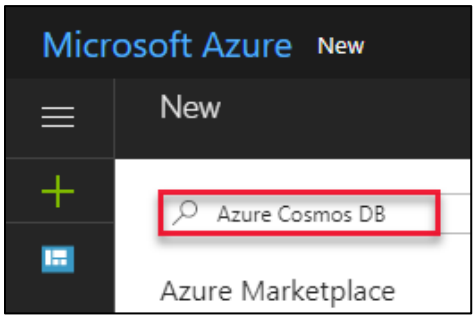
1. **Launch** a browser and **Navigate** to https://portal.Azure .com. **Login** with your Microsoft Azure credentials.
2. To toggle **show/hide** the Portal menu options with icon, **Click** on the **Show Menu** button.



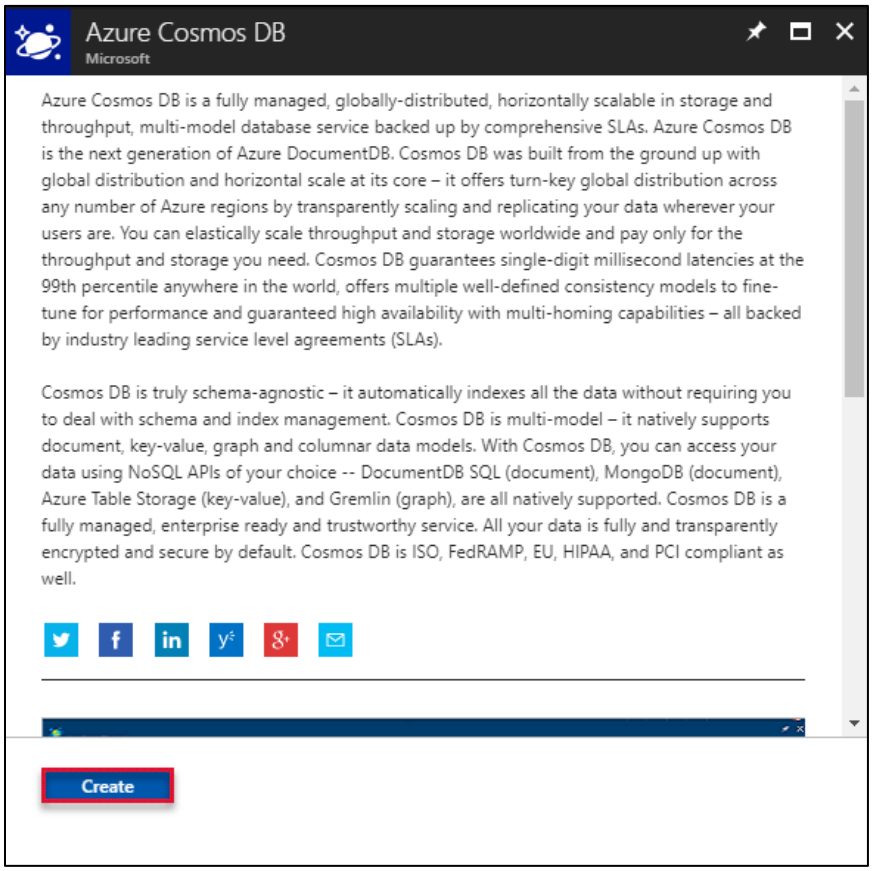
3. **Click** on **+New** button.



4. Search for **Azure Cosmos DB** and click on the same.



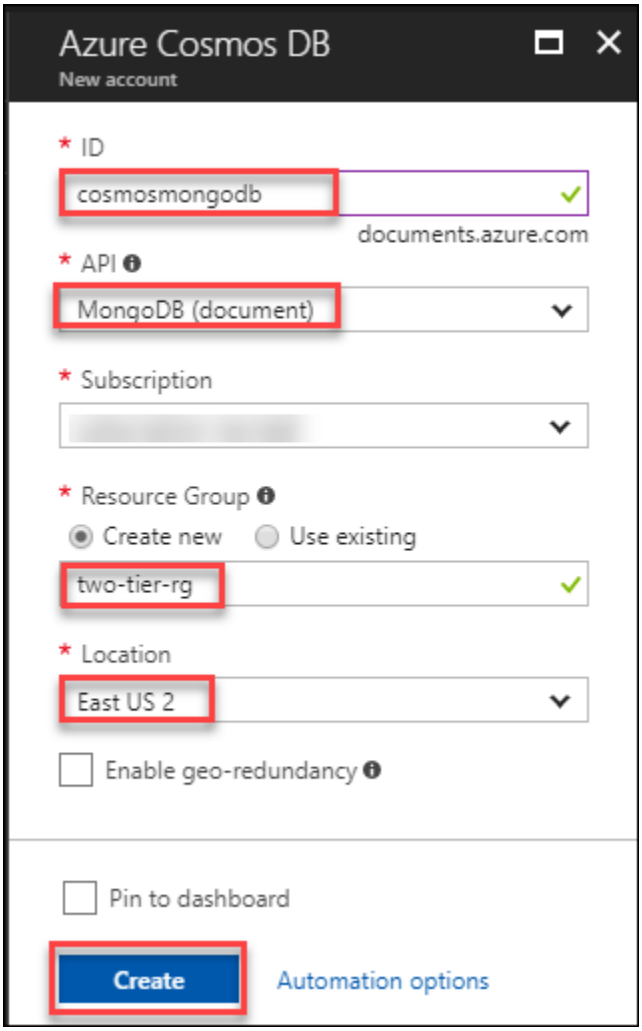
5. Click on **Create** button.



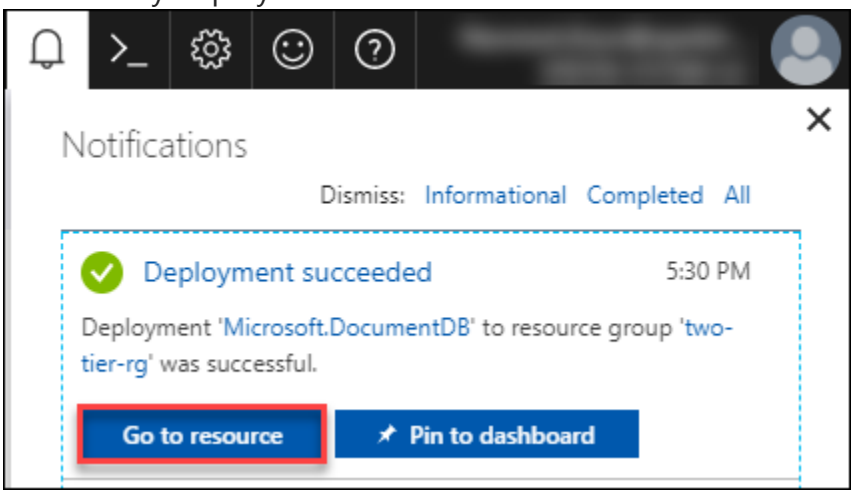
6. Populate the below parameters as shown below.

- *ID:* **cosmomongodb(**any valid name**)**
- *API:* select **Mongo DB** from the dropdown

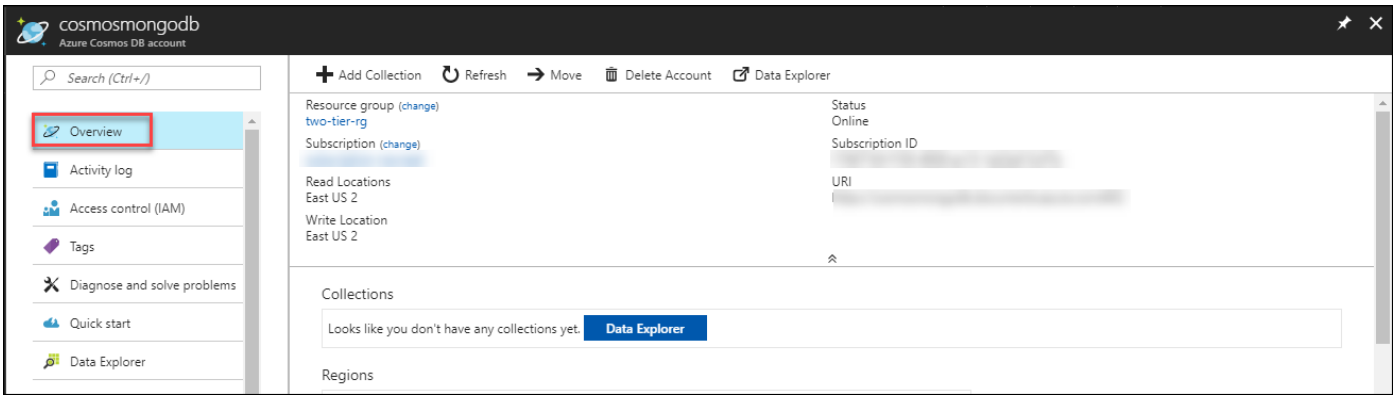
- *Resource Group:* Choose **Create new** and provide any valid name



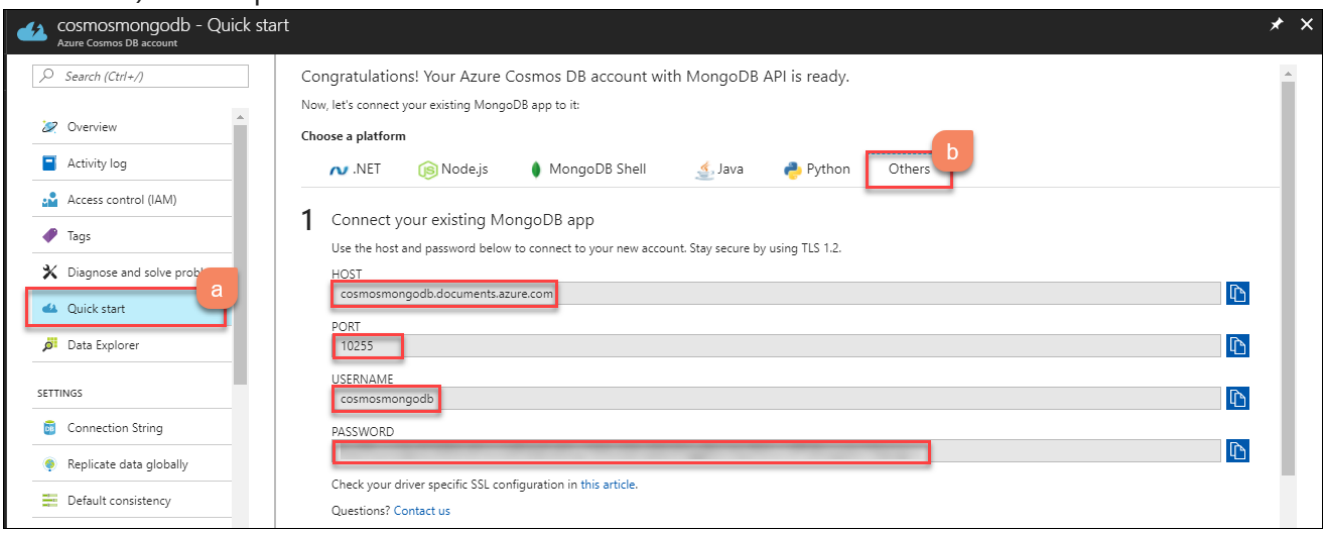
7. After deployment gets completed, click on **Go to resource** to verify that resource is successfully deployed.



8. After that you can view that **cosmosmongodb** is created. Click on **Overview**.

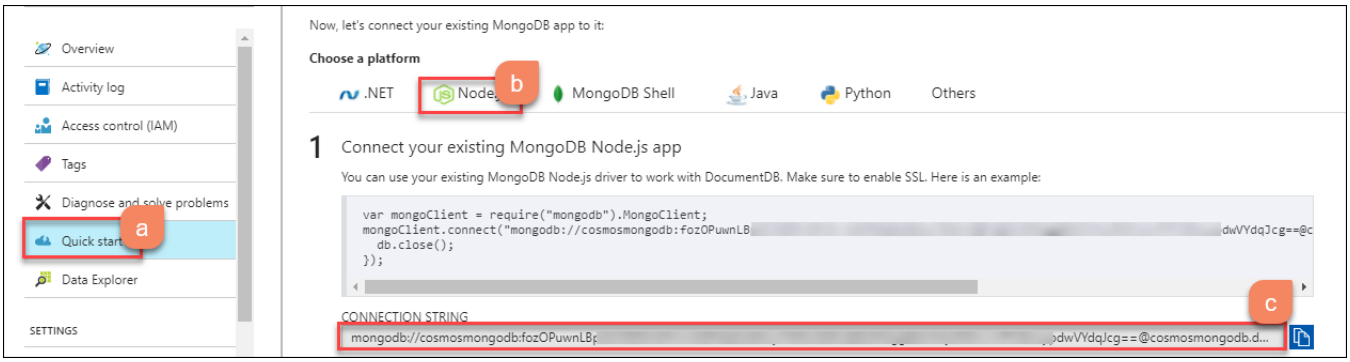


9. Go to **Quick start** and click on **Others**. Copy all the parameters (Host, Port, Username, Password) in notepad for future use.



10. Click on **Node.js** and copy the **CONNECTION STRING** to be used in future.

```
mongodb://cosmosmongodb:<password>@cosmosmongodb.documents.azure.com:10255/tododb/?ssl=true
```



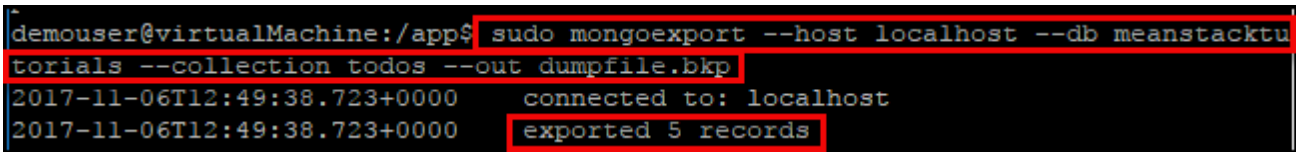
Exercise 2: Migrate MongoDB On Azure

In this exercise, we will login to **On-Premises VM** which we created in Lab 1→ Exercise 1 and migrate MongoDB on azure.

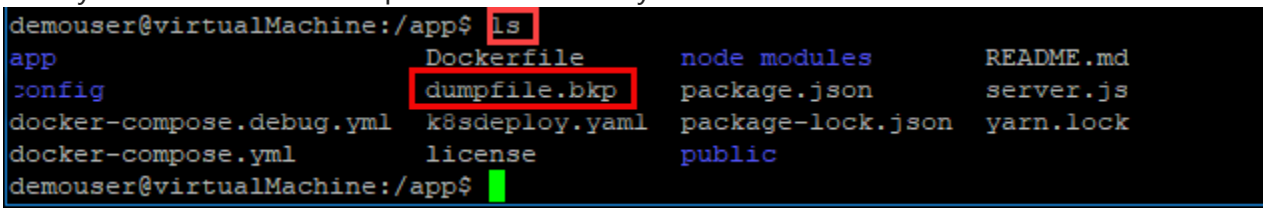
1. Once, you get connected to **On-premises VM** which you deployed in Lab 1→ Exercise 1. Use the below command to export the data from on-premises VM.

- <*db_name*>: **meanstacktutorials** (any database name to be used by application post migration, you can keep it same or change as per requirement of your application)
- <*Collection_name*>: **todos** (as per app requirement)

```
sudo mongoexport --host localhost --db <db_name> --collection
<Collection_name> --out dumpfile.bkp
```

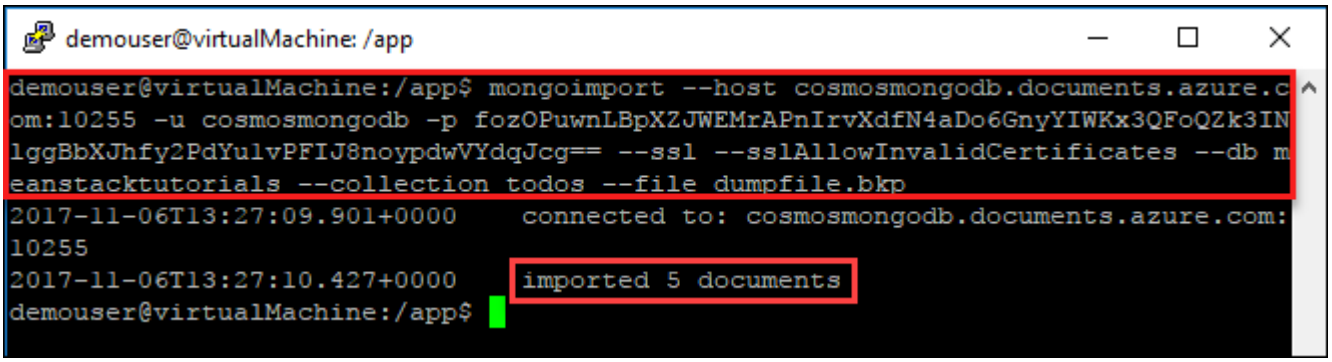


2. Now you can see data is exported successfully.



3. Now, we will import this MongoDB on Azure Cosmos DB (MongoDB) and replace the HOST, PORT, USERNAME and PASSWORD with the parameters in below command with values you copied in LAB 2→ Exercise 1→ Step 9.

```
mongoimport --host <HOST>:<PORT> -u <USERNAME> -p <PASSWORD> --ssl --
sslAllowInvalidCertificates --db <db_name> --collection todos --file
<BackupFileName>
```

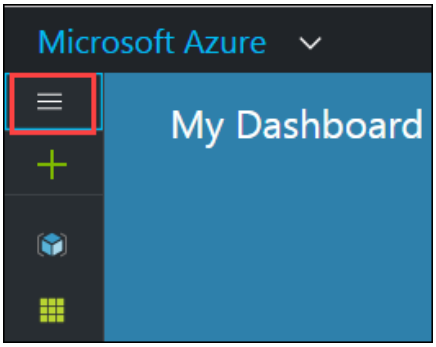


4. You can see the message for imported documents as highlighted in above screenshot.

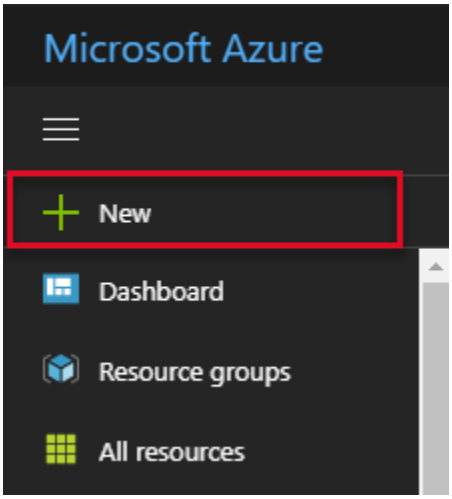
Exercise 3: Deploy Ubuntu with Docker using template

In this exercise, we will deploy the template for docker on ubuntu server to migrate Nodejs on container.

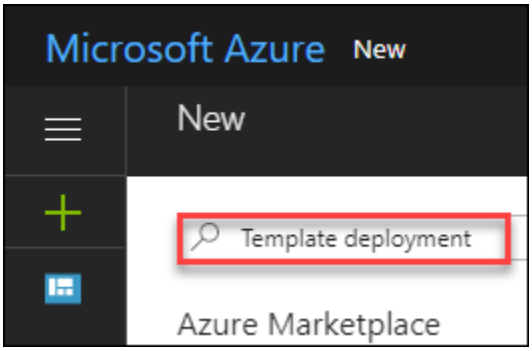
1. **Launch** a browser and **Navigate** to https://portal.Azure .com. **Login** with your Microsoft Azure credentials.
2. To toggle **show/hide** the Portal menu options with icon, **Click** on the **Show Menu** button.



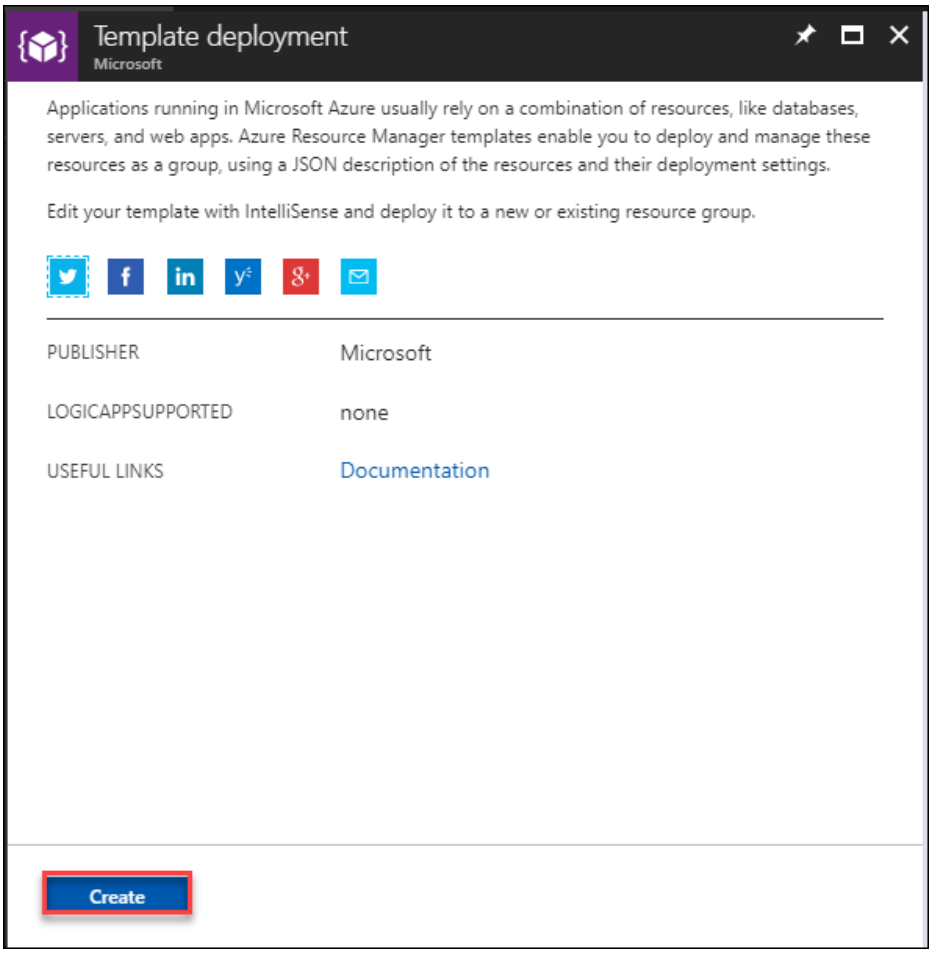
3. **Click** on **+New** button.



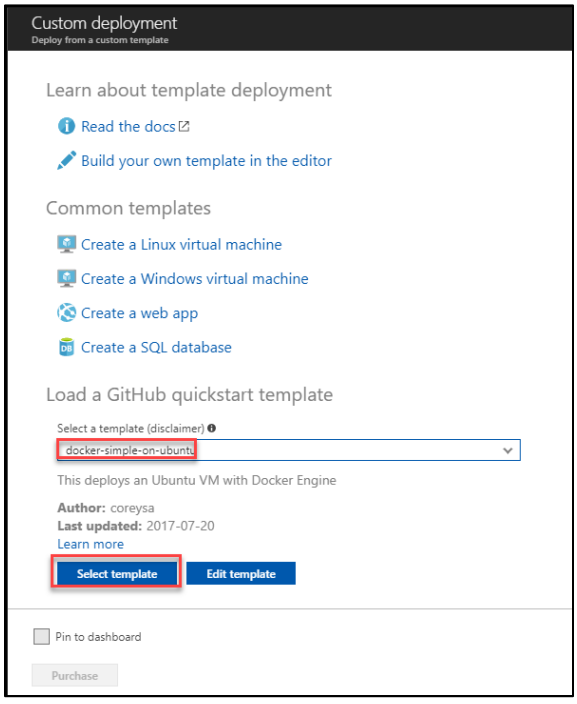
4. Search for **Template Deployment**.



5. Click on **Create** Button.



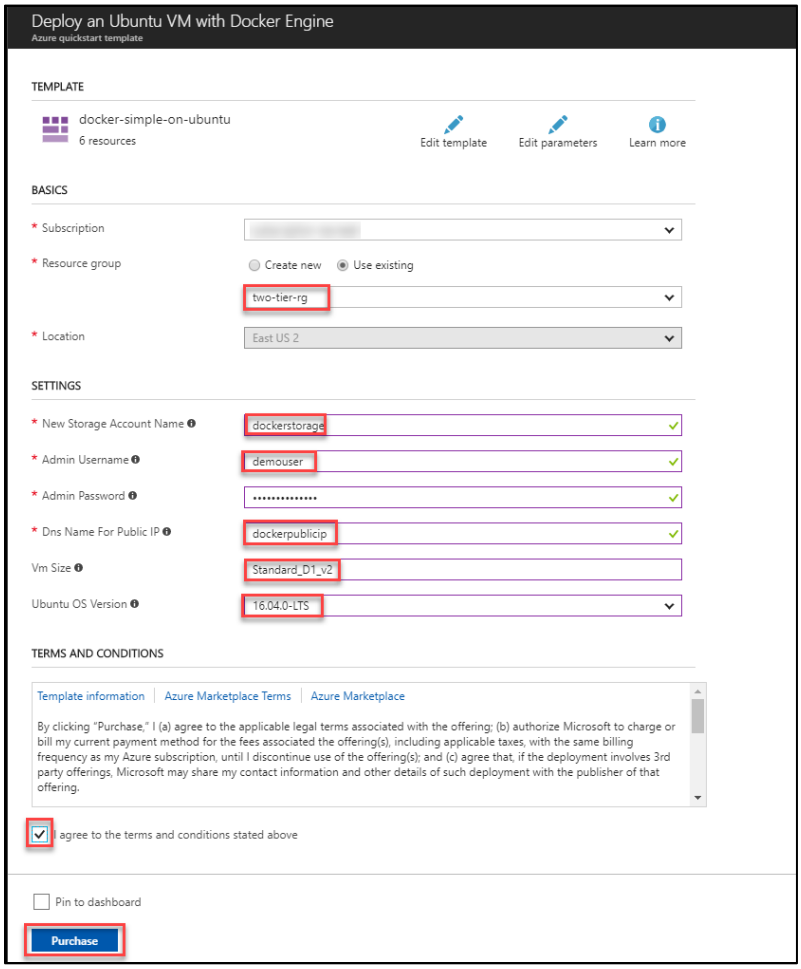
6. Select **docker-simple-on-ubuntu template** and Click on **Select** template.



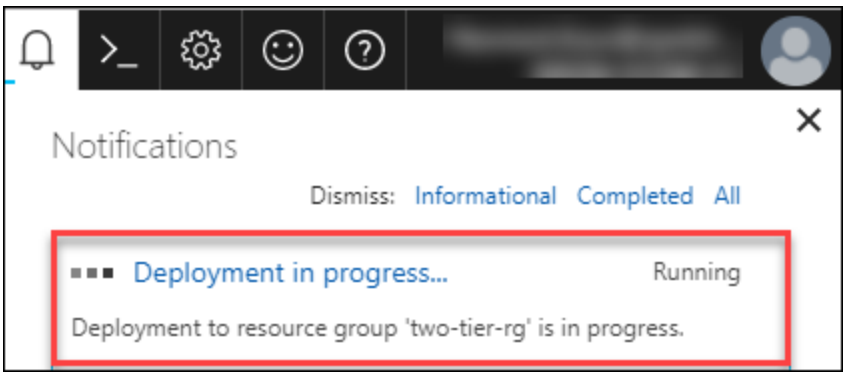
7. Provide the parameters as shown below.
    a. *Resource Group:* **two-tier-rg** (use existing resource group)
    b. *New Storage Account Name:* **dockerstorage12** (any valid name)

c. *Admin Username:* **demouser** (as per your choice)
d. *Admin Password:* <valid password>
e. *Dns Name for Public IP:* **dockerpublicip12** (any valid name)
f. *Vm Size:* **Standard_D1_v2**
g. *Ubuntu OS Version:* **16.04.0-LTS**

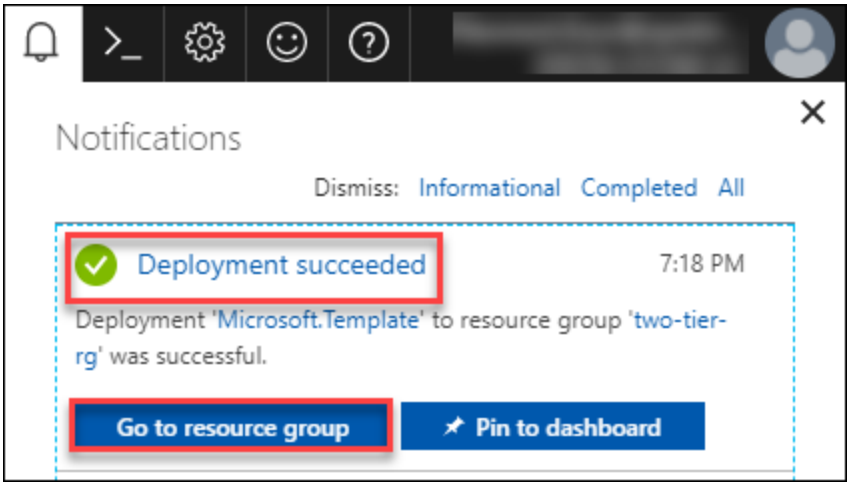
Accepts terms and conditions and click on **Purchase** button.



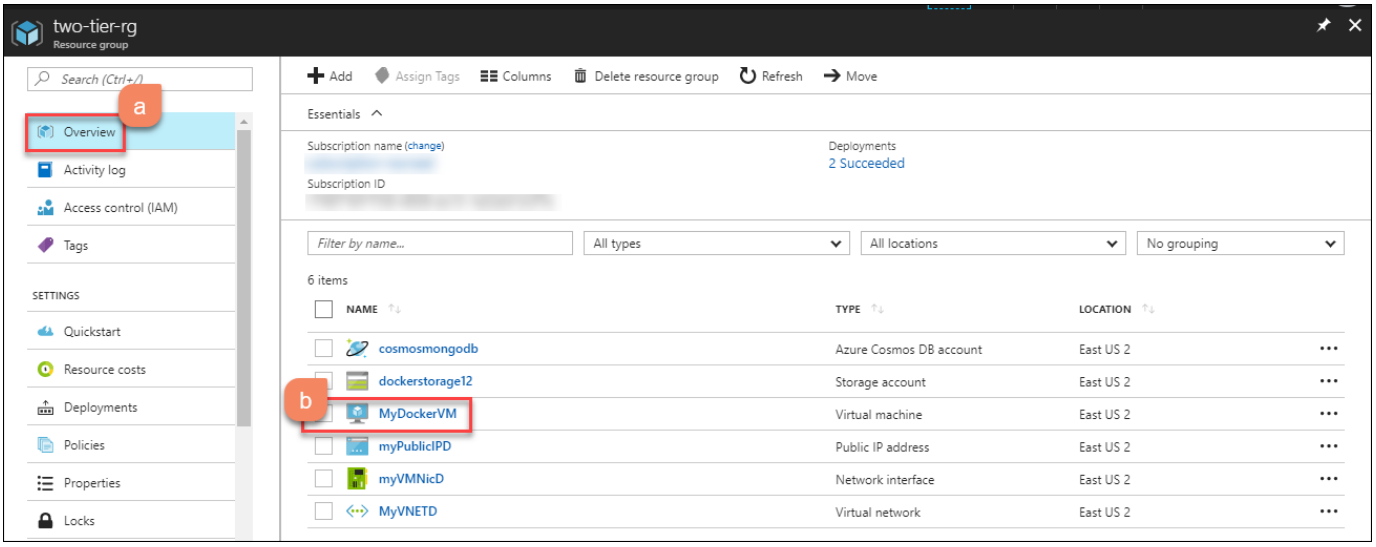
8. It will be validated and then deployment will start.



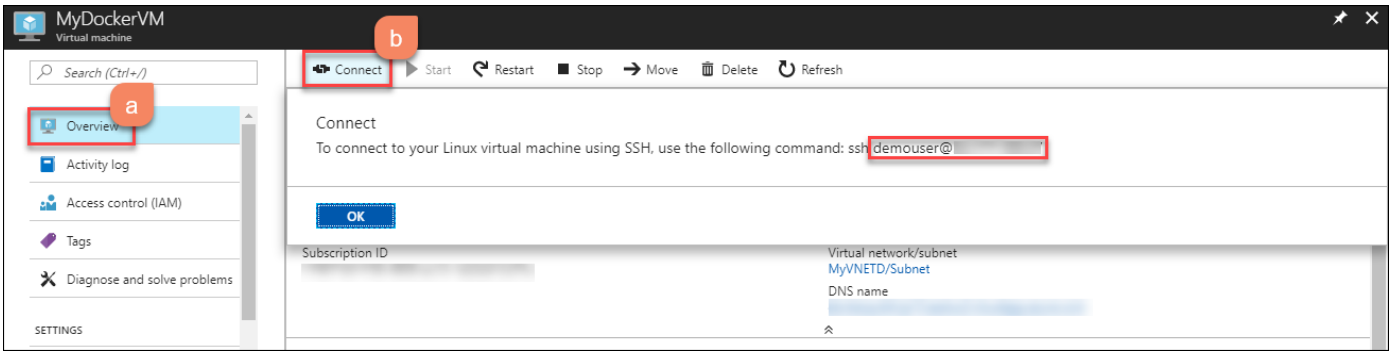
9. After deployment gets completed, click on **Go to resource** to verify that resource is successfully deployed.



10. Click on Overview and select the **MyDockerVM.**

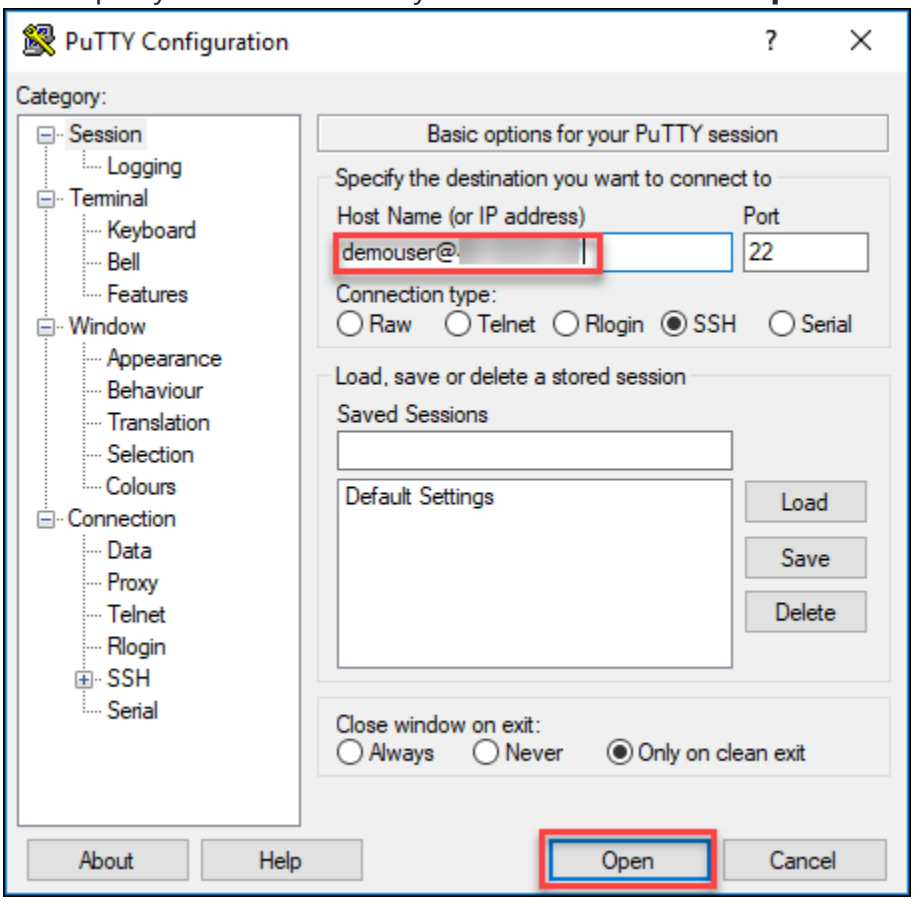


11. In overview section, click on **Connect** button. It will show the username with IP address. Copy that username with IP address will you it for connectig the VM.

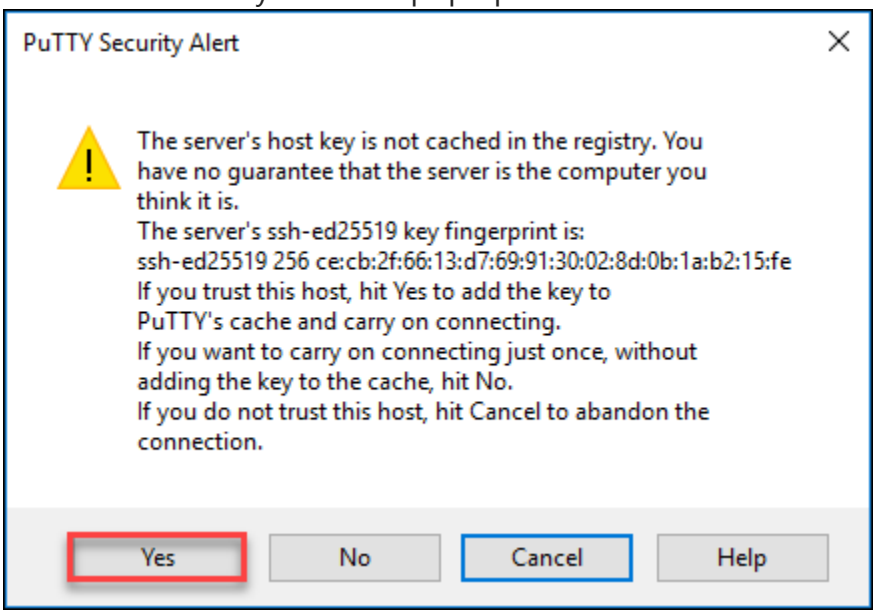


12. Now run putty.exe from your PC.

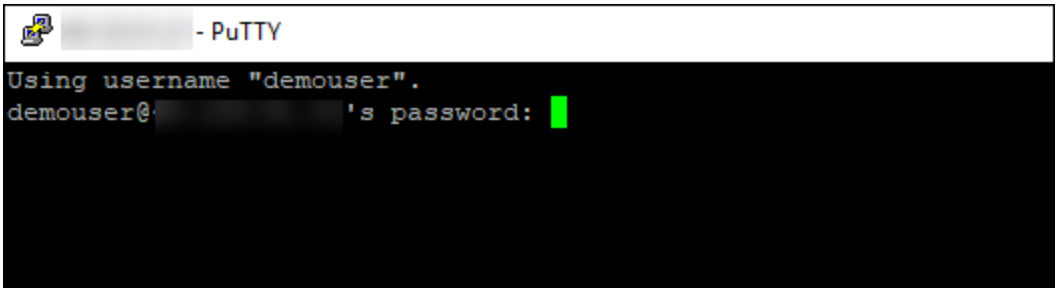
13. This is the application window that pops up when you run **putty**. Paste the Username with Public IP address of the VM that you copied from above to the Host Name (or IP address) box of the putty. Port will be 22 by default. Then Click on **open**.



14. The PuTTY Security Alert will pop up. Click on **Yes**.



15. Login with Virtual Machine **password** that you provided while creating the same.

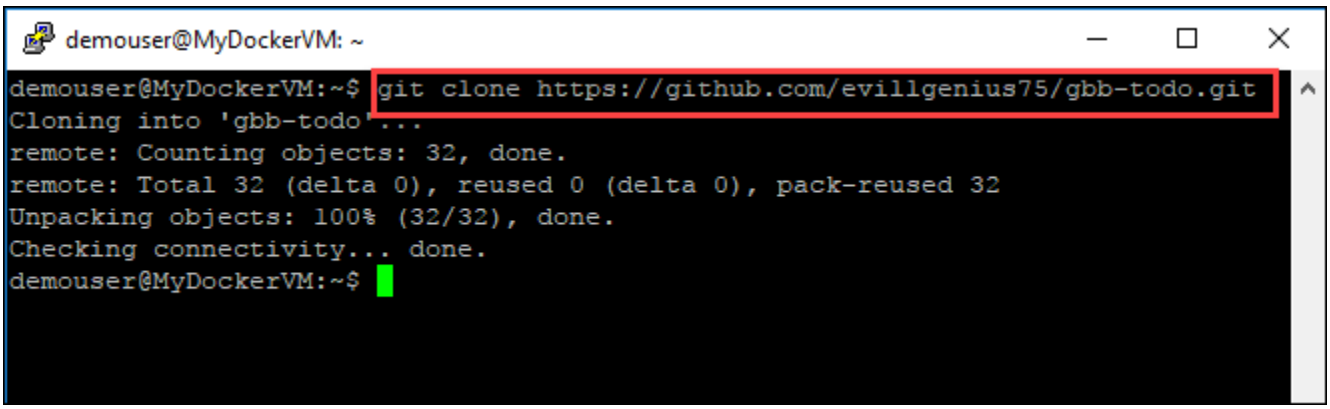


Exercise 4: Migrate NodeJs App to Container

In this exercise, we will connect to Docker VM if you are not already connected, follow the step 11 to 15 in Lab2 → Exercise3 and migrate NodeJS app to container.

1. Once you get connected to Docker VM, clone the app from git URL (https://github.com/evillgenius75/gbb-todo.git) by using following command.

```
Git clone https://github.com/evillgenius75/gbb-todo.git
```

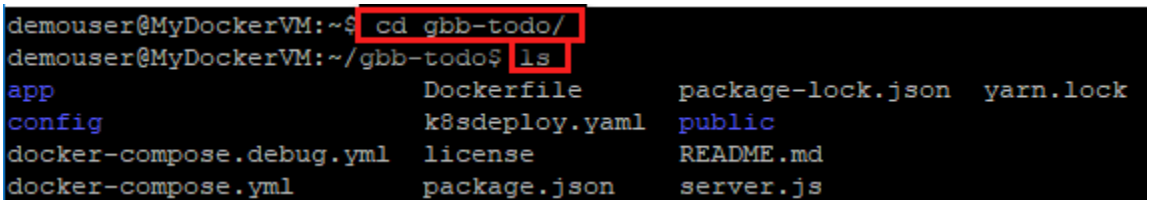


2. Run below command to see the app is cloned successfully.

```
ls
```

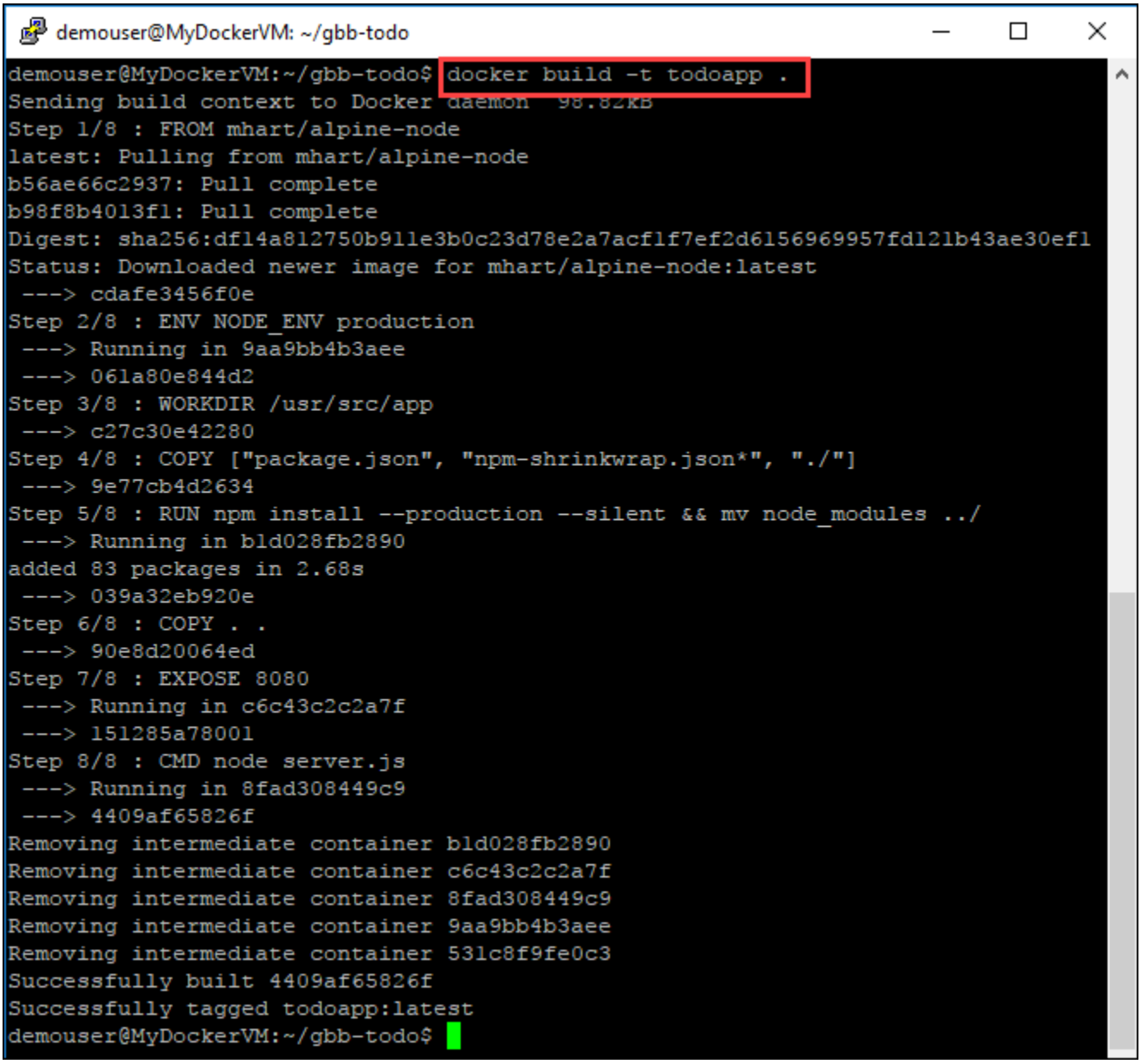
3. Run below command to change directory to application folder and list the files inside application folder

```
cd gbb-todo/
ls
```



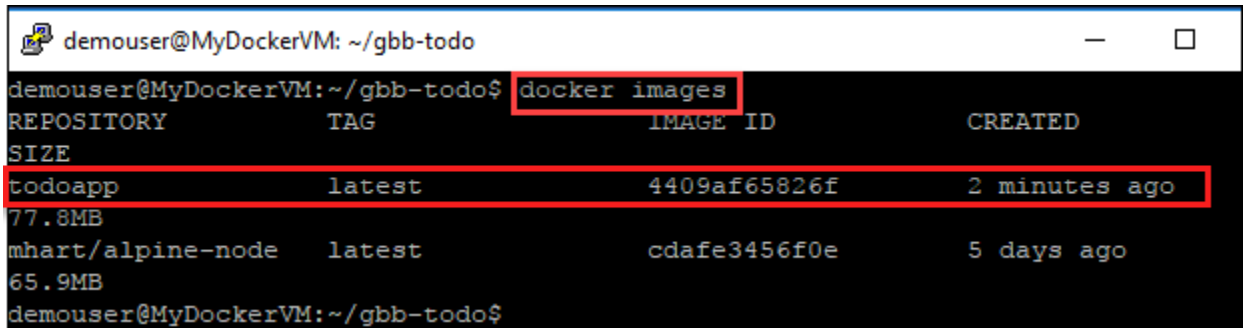
4. You can see there is a Dockerfile which is used to **create** docker images. We will run the below command to create an image for nodejs ToDo app.

```
docker build -t <image_name> .
```



5. Run below command to list the docker images. We will see that todoapp image is now listed.

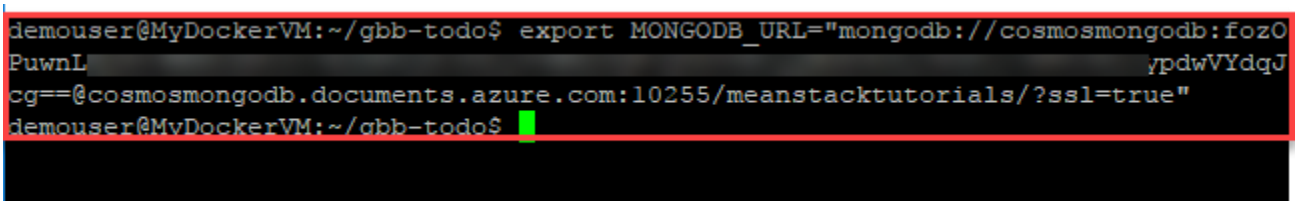
```
docker images
```



6. Define the environment variable for MonogDB to be used in container using below command. We are referring to connection string copied in Lab 2→ Exercise 1 → Step 10. We need to

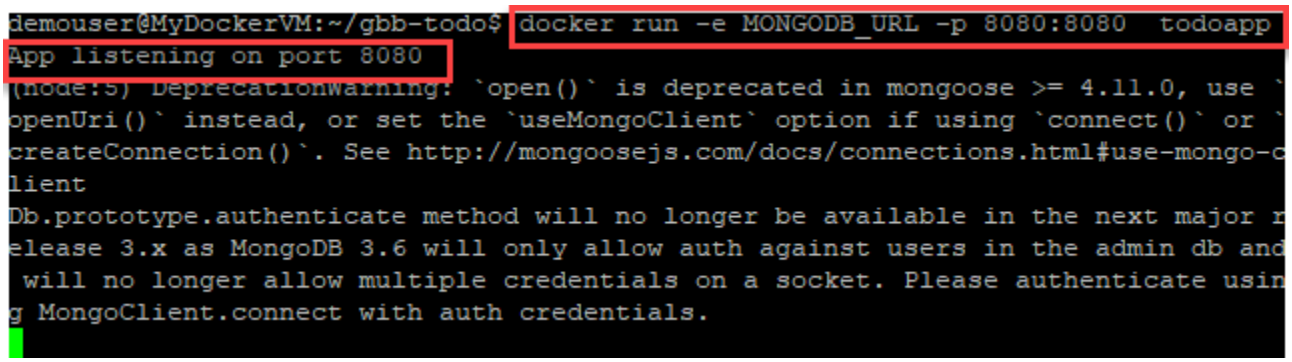
append the dbname in the connection string as well. **dbname** refers to name defiend in Lab2→ Exercise1→ Step 1.

```
export MONGODB_URL="mongodb://cosmosmongodb:<password>@cosmosmongodb.documents.azure.com:10255/tododb/<dbname>/?ssl=true"
```

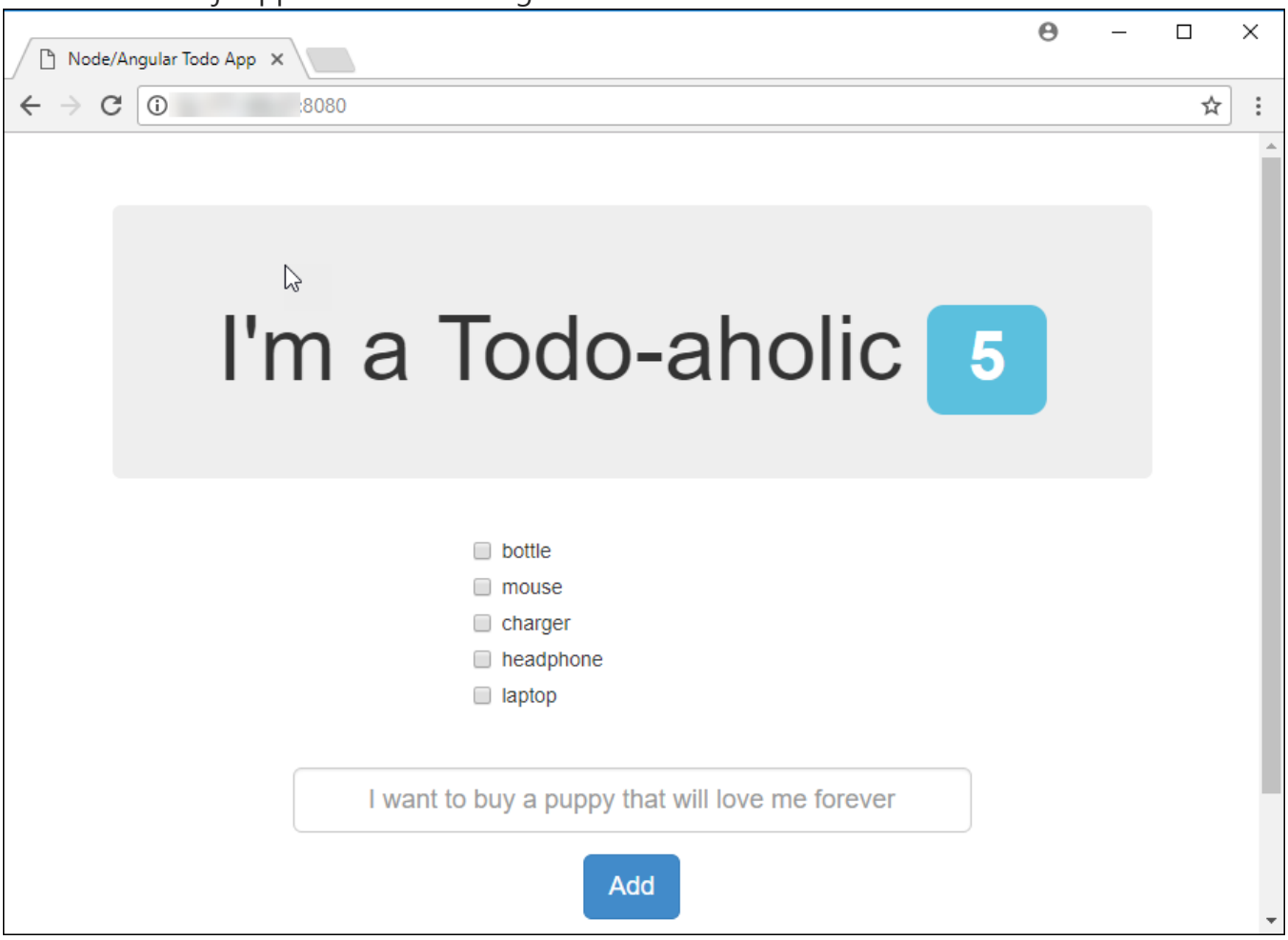


7. Run container using newly created image with below command:

```
docker run -e  MONGODB_URL -p 8080:8080 <image_name>
```



8. Open the browser and browse the Public IP address of docker VM with port 8080 which will show the Node.js application is running.



*** This ends the lab.***