LAB GUIDE

Deploying spring boot application in Azure Kubernetes cluster .

|  |
| --- |
| Author :Victor Dey |

Overview

In this lab, an AKS cluster is deployed using the Azure CLI. A multi-container application consisting of web front end (spring boot) instance is then run on the cluster. Once completed, the application is accessible over the internet.

Learning Objectives

* Understand the fundamentals of creating a Kubernetes Cluster using Azure Kubernetes Service (AKS)
* Understand how to deploy a simple application to the cluster

Exercise 1: Login to the Azure Management Portal

Overview

In this exercise, you will login to the Azure Portal with your lab credentials and install all prerequisites ( Maven,Docker etc)

Time Estimate

* 30 minutes

Task 1: Login to Portal

1. Launch a separate browser on your computer using Incognito or In-Private mode and navigate to **https://portal.azure.com**.
2. Use your username and password to login to the Azure portal.

Task 2: Install AZURE CLI (you can skip the step if you have already installed)

* 1. Download Azure CLI for windows and install the **azure-cli-x.x.msi**
  2. Login using your azure portal credential

Task 3: Install Maven

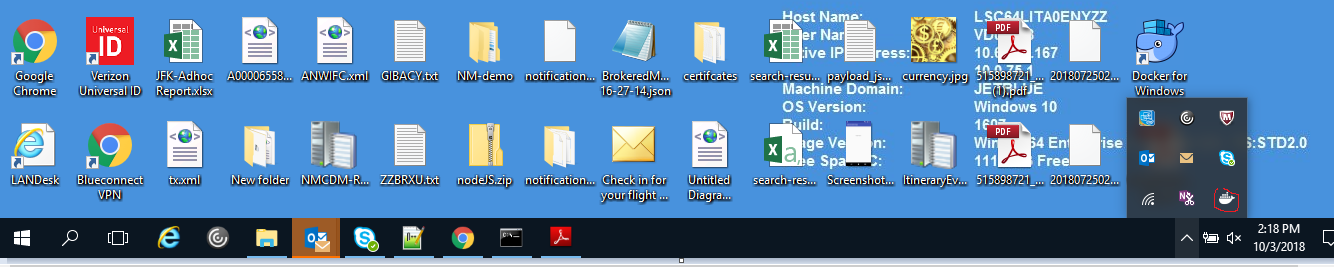
Install Maven in your computer in case it’s not installed

1. Download the Maven <https://maven.apache.org/download.cgi> Binary zip archive
2. Create System/env variable M2\_HOME c:\<yourpath>\apache-maven-3.5.4
3. Add the variable in PATH %M2\_HOME%\bin;

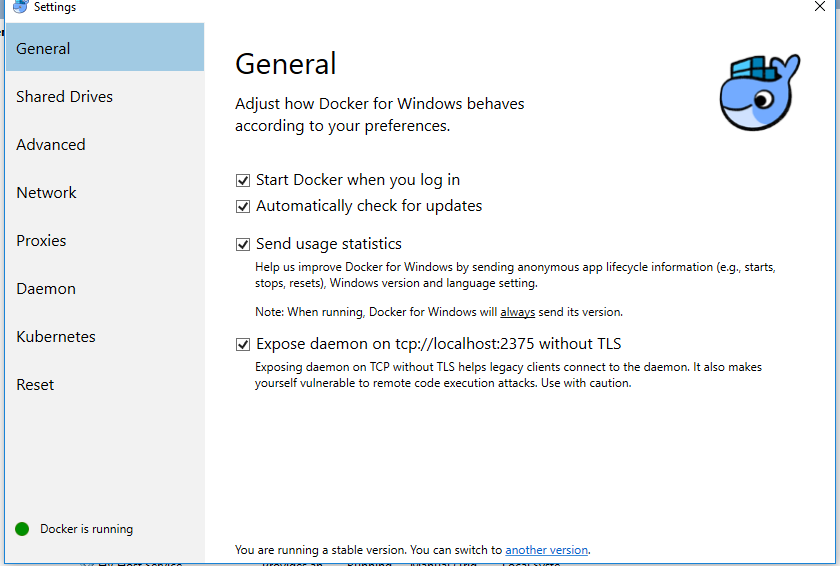
Task 4: Install Docker

Install docker in your windows link:

1. Download and install <https://docs.docker.com/docker-for-windows/install/>
2. After installation go to service.msc and confirm the service “Docker for Windows Service” is running
3. On the task bar right click on the docker icon and click on setting



1. Check the Expose the daemon option



Summary

In this exercise, you successfully created all the prerequisite for the excercise2

Exercise 2: Build one docker image and deployed in kubernetes cluster

Overview

In this exercise, we will create docker image of a spring boot application and then deploy the image in kubernetes cluster from Container Registry

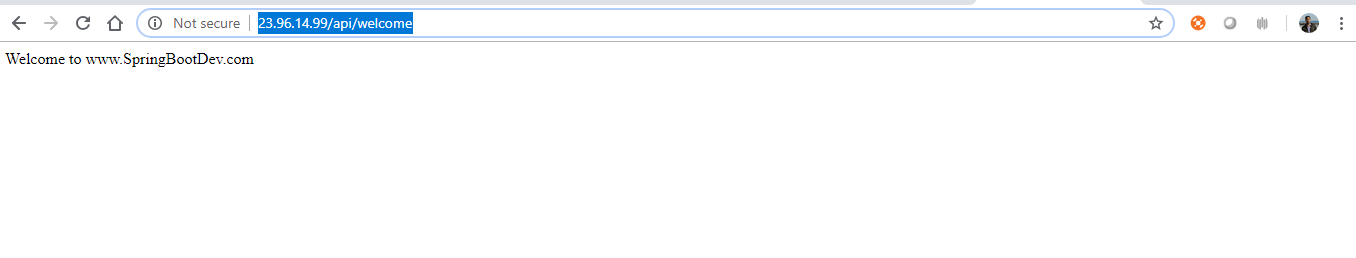
Time Estimate

* 30 minutes

Exercise 2: Deploy an Azure Kubernetes Service (AKS) cluster

In this exercise, an AKS cluster is deployed using the Azure CLI. A multi-container application consisting of web front end and a Redis instance is then run on the cluster. Once completed, the application is accessible over the internet.

Image



Task 1: Create a resource group

1. Create a resource group with the [az group create][az-group-create] command. An Azure resource group is a logical group in which Azure resources are deployed and managed. When you create a resource group, you are asked to specify a location. This location is where your resources run in Azure.

The following example creates a resource group named *myAKSCluster* in the *eastus* location.

1. az login
2. az account set -s <YourSubscriptionID>
3. az group create --name myAKSCluster --location eastus

Output:

{

"id": "/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myAKSCluster",

"location": "eastus",

"managedBy": null,

"name": "myAKSCluster",

"properties": {

"provisioningState": "Succeeded"

},

"tags": null

}

Task 2: Create a Azure container registry

Create a Create a private Azure container registry in the resource group. The tutorial pushes the sample app as a Docker image to this registry in later steps. Replace wingtiptoysregistry with a unique name for your registry.

1. az acr create --admin-enabled --resource-group wingtiptoys-kubernetes --location eastus --name wingtiptoysregistry --sku Basic

Task 3: Build Docker image

Download the project from https://github.com/victordey2007/springBootApp and navigate to the completed project directory for your Spring Boot application (for example, "C:\SpringBoot\intial" and open the pom.xml file with a text editor.

Create Azure Container private repository using the following command .

* 1. az acr create --admin-enabled --resource-group wingtiptoys-kubernetes --location eastus --name wingtiptoysregistry --sku Basic

note : Please change the --name parameter and give some unique name .

## Navigate to the root folder where you download the spring project and update the pom.xml with yellow highlighted portion with –-name parameter given in previous command

1.

<properties>

<docker.image.prefix>wingtiptoysregistry.azurecr.io</docker.image.prefix>

<java.version>1.8</java.version>

</properties>

2.<plugin>

<groupId>com.spotify</groupId>

<artifactId>docker-maven-plugin</artifactId>

<version>0.4.11</version>

<configuration>

<imageName>${docker.image.prefix}/${project.artifactId}</imageName>

<buildArgs>

<JAR\_FILE>target/${project.build.finalName}.jar</JAR\_FILE>

</buildArgs>

<baseImage>java</baseImage>

<entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]</entryPoint>

<resources>

<resource>

<targetPath>/</targetPath>

<directory>${project.build.directory}</directory>

<include>${project.build.finalName}.jar</include>

</resource>

</resources>

<serverId>wingtiptoysregistry</serverId>

<registryUrl>https://wingtiptoysregistry.azurecr.io</registryUrl>

</configuration>

</plugin>

In CLI Navigate to the completed project directory for your Spring Boot application and run the following command to build the Docker container .

* 1. mvn package docker:build
  2. docker images //it will show you the image created recently in console .

Task 4: Push your image to the container registry

Retrieve the password for your container registry from the Azure CLI and push the image with the credential. Please replace the registry name with your own registry name .

1. az acr credential show --name wingtiptoysregistry --query passwords[0]

Output:

{

"name": "password",

"value": "AbCdEfGhIjKlMnOpQrStUvWxYz"

}

Now login with your registry name and password , then push the image .

1. az docker login -u wingtiptoysregistry -p "AbCdEfGhIjKlMnOpQrStUvWxYz" wingtiptoysregistry.azurecr.io
2. docker push wingtiptoysregistry.azurecr.io/gs-spring-boot-docker

Task 5: Create a Kubernetes Cluster on AKS using the Azure CLI

Create a Kubernetes cluster in Azure Kubernetes Service. The following command creates a kubernetes cluster in the wingtiptoys-kubernetes resource group, with wingtiptoys-akscluster as the cluster name, and wingtiptoys-kubernetes as the DNS prefix:.

1. az aks create --resource-group=wingtiptoys-kubernetes --name=wingtiptoys-akscluster --dns-name-prefix=wingtiptoys-kubernetes --generate-ssh-keys
2. az aks get-credentials --resource-group=wingtiptoys-kubernetes --name=wingtiptoys-akscluster // it will get the credential and merger it in your local config file

Now run the container in Kubernetes cluster using command firs , then we will do the same with Yaml file .

Using command

1. kubectl run gs-spring-boot-docker --image=wingtiptoysregistry.azurecr.io/gs-spring-boot-docker:latest
2. kubectl expose deployment gs-spring-boot-docker --type=LoadBalancer --port=80 --target-port=8080
3. kubectl get service gs-spring-boot-docker --watch

notes:

In this command:run

* 1. The container name gs-spring-boot-docker is specified immediately after the run command
  2. The --image parameter specifies the combined login server and image name as wingtiptoysregistry.azurecr.io/gs-spring-boot-docker:latest

In this command:expose

The container name gs-spring-boot-docker is specified immediately after the expose deployment command

* 1. The --type parameter specifies that the cluster uses load balance
  2. The --port parameter specifies the public-facing TCP port of 80. You access the app on this port.
  3. The --target-port parameter specifies the internal TCP port of 8080 because the code running on Tomcat server and by default its using 8080. The load balancer forwards requests to your app on this port.

In this comman : get will give the external IP .Copy the external ip and open the url in browser <http://yourip:80/api/welcome>

**Now we will execute the same using yaml**

1. kubectl apply -f docker-compose.yaml
2. kubectl get service gs-spring-boot-docker --watch

Once you get the external ip open it in browser.

Note : in yaml file we have define the container with few configuration and define one service with type LoadBalancer and map the port 80 to target port 8080 as our container having tomcat server running on 8080.