Ahmad Mughrabi ID # 012233817

Amughr1@wgu.edu

Abstract

This document will explain our choice in cloud provider, alongside with the deployment process.

THE CADE

Deployment Doc

# Choice in Cloud Provider

I opted to go with Microsoft Azure as our cloud provider as I believe it is the best in class for simple maintainability and longevity combined. Microsoft Azure as a service provider has been dealing with customers for years on end. Over time they have built up the best practices and understand first and foremost what is best for the user. This is evident as we see time and time again from their ever so popular Windows OS. Their products might start complex early into their lifecycle but gradually get shaven down and off to show quite the straightforward and immediate usefulness in their user interfaces and user experience. This translates fully over to their Microsoft Azure cloud services in regards to maintainability and longevity.

## Maintainability

Azure’s UI is extremely simple and straight to the point for all products. You click a service and you know what you are getting. See the following example

1. New Resouce
2. Container App

You know on from the following two steps that you are creating a new resource, and it’s a containerized app. On something like AWS you’ll be met with decision paralysis unless you know what it is that you want. You can host the same app multiple ways-

1. Elastic Beanstalk
2. Elastic Container Service
3. Elastic Kubernetes Service
4. Elastic Compute Cloud

## Longevity

I could have gone with a VPS on something like Linode, or Render for the simplicity, but I chose one of the big 3 cloud providers. Although Linode has been around for ~20ish years, the issue with Linode is that unless you know the in’s and out’s of a Linux Distro, its hard to maintain this app after I’m gone. Render would’ve been the next best option, but Render is on the newer side of things. There is no guarantee they will be around in the next 5-10 years. These household names like AWS, GCP, and Azure will not likely have this problem.

# Deployment Process

To get an app like this in a scalable state involves two steps. Containerization and Deployment. Some names will be different for you, depending on project name, image name, repository name etc.

Containerization is the process of taking your bundled application and putting it in a image, which then gets ran as a container. An image is a template for running a container, and a container is a running process. The most popular software for containerization is Docker, they have a Docker app, and a whole slew of Docker Images on Docker Hub, their online repository for finding Docker Images. You’ll need to create an account here and upload your docker image. When that’s done you’ll need to create a Microsoft Azure account, create the required resources in the New Resource, Create Container App section, link your Docker image from Docker Hub, and you’ll be set!

## Containerization

1. You’ll need to download Docker and Docker Desktop, you’ll also need to create an account with Docker Hub for your new repository.
   1. Create a repository on Docker Hub for your application, for demonstration purposes this will be a public repo.
2. After installing Docker, you’ll open your IDE in the root of the project folder. This will set the Terminal that opens to the same folder.
3. Create a Dockerfile, (a set of instructions used to build your Container Image)
   1. FROM eclipse-temurin:21-jdk-alpine
      1. This is the OS environment used on the inside of your container.
      2. Alpine is always good for a smaller container size
   2. MAINTAINER baeldung.com
      1. This is metadata showing who currently maintains the above OS
   3. COPY target/demo-0.0.1-SNAPSHOT.jar app.jar
      1. This copies the to be created jar file to the new container’s directory
   4. ENTRYPOINT ["java","-jar","/app.jar"]
      1. This is the first command your container runs at startup
      2. This is like saying “java -jar /app.jar”
   5. EXPOSE 8080
      1. This opens up port 8080 on the new container, letting us be able to communicate with it on the outside looking in.
4. Run the following commands
   1. mvn package
      1. This bundles your Java Application into one executable jar file.
   2. docker image build -t imagename:latest .
      1. This builds an image using all of your application files.
      2. Verify that your container works with Docker Desktop
   3. docker login
      1. Authorize your computer to be able to push up to your docker repo
   4. Docker tag imagename:latest yourdockerhubusername/dockerhubreponame:latest
   5. Docker push yourdockerhubusername/dockerhubreponame:latest
5. Your image is now on Docker Hub!

## Deployment

1. For this step you’ll need an Azure Account.
2. On the main landing page, click + New Resource.
3. Search for Container App, click it, then click create.
4. Basics
   1. Project Details
      1. Create a subscription, create a new resource group
      2. Name your container.
      3. Deployment source – container image
   2. Container Apps Environment
      1. Region = Pick one where you want this to be hosted
      2. Create new container apps environment, leave default settings on new page, hit Create.
5. Container
   1. Container Details
      1. Name will be the same as container name on previous page
      2. Image Source = Docker Hub
      3. Image Type = Public
      4. Image and Tag = your entire dockerhub repo name + tag
   2. Ingress
      1. Ingress = Checked
      2. Ingress Traffic = Accepting Traffic from Anywhere
      3. Target Port = The same as the EXPOSE command in your dockerfile.
   3. Leave the defaults for the next two pages and go to the end, hit Create.
6. After a few minutes, your app will now be deployed!