Publishing ASP.NET Core Web App to AWS

Cheat Sheet

INET2005 Web App Programming I
Sean Morrow

Publishing ASP.NET Core web apps is complicated. No longer can you just FTP files to a server - Instead, most modern web apps require heavy configuration on multiple servers. Cloud based solutions (AWS, Azure, Heroku, Digital Ocean) have made this a little easier, but it is still a challenge.

In this course, we will be exploring Amazon Web Services (AWS) for publishing. AWS is a collection of remote computing services, or web services, offered by Amazon to handle tasks related to cloud computing, storage, databases, etc. The setup is complicated – but luckily, we have a secret weapon…Docker!

With Docker, we can spin up all required containers (and thus servers) on an AWS EC2 instance without having to spend time on configuration. Note the approach taken below is very manual (for educational purposes) as there are other more automated approaches.

This cheat sheet will walk you through the process.

Step 1:
Login to AWS EC2 Dashboard

- Each student has already received an invite from AWS Academy
- Login to AWS Academy and open the “AWS Academy Learner Lab” course
- Click the “Learner Lab - Foundational Services” link
- Click the “Start Lab” play button to start up your AWS lab – it may take some time to startup. Note that you only have a certain amount of time to play with AWS before your session expires. Your data is not lost when the session runs out, but it will shut down all your EC2 instances.
- Once the lab is started you can click the “AWS” link to go to the “AWS Management Console”

Step 2:
Setup EC2 (Amazon Elastic Compute Cloud)

- EC2 provides a computing service for handling computational tasks. In our case, we’ll be using EC2 to run the servers which will host our web application. In other words, EC2 is a virtual machine running your server, etc. (spins it up)
- AWS refers to these virtual machines as "instances"
- Click "Launch Instance " from EC2 Dashboard

Step 3:
Select an AMI (Amazon Machine image) to base the EC2 instance on

- While there are prebuilt .NET Core stack AMIs, it is good practice to set one up from scratch using our docker containers. Select “Amazon Linux 2” (free tier eligible) from the Quick Start menu
- click the launch button – this will require key/pair setup

Step 4:
Create your Key/Pair

- to launch your instance you need to create an EC2 Key Pair
- Basically, key pairs enable you to login to EC2 instances without a username / password. Also known as an SSH key
- select from the drop down "Create a new key pair"
- give it a name like "INET2005-AWS-Key"
- Click the download button to download .pem file with the same name. Don’t lose this file! It’s only generated once and you’ll need it to access your EC2 instance.
- the EC2 instance will now be launched – wait for it to start running before continuing

Step 5:
Connecting to the EC2 Instance with SSH and key pair

- click on the running EC2 instance and click the connect button for detailed instructions on how to connect to your server via SSH
- open a terminal (or windows bash / or putty) in same folder as the pem key pair file and run commands:
```
chmod 400 INET2005-AWS-Key.pem
ssh -i "INET2005-AWS-Key.pem" ec2-user@[public DNS of EC2 instance]
```
- the SSH command above can be found by clicking link for EC2 instance and clicking the connect button. Look for SSH client and copy the provided example
- access should be granted and you will see a flashing cursor – you can now run commands on your instance on AWS!

Step 6:
Thanks to docker, we don’t need to manually install all the dependencies for our app, but we do need to install Docker and Git:

- Will use YUM which is a packaging tool much like NPM – but first make sure it is updated. Run command:
```
sudo yum update -y
```
- Install docker with commands:
```
sudo amazon-linux-extras install docker

sudo usermod -aG docker $USER

newgrp docker
```
- Install docker-compose with commands:
```
sudo curl -L
https://github.com/docker/compose/releases/latest/download/docker-
compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

- Install git:
  ```
  sudo yum install git
  ```

Step 7:
There is some configuration that needs to be done to the web app before deploying to the EC2 instance. Luckily, with docker this can all be tested locally.

- Currently only one container is spun up for the MySQL server. We need to spin up a second server to host the ASP.NET Core web app itself. This second container will run the kestrel server internally. Copy over the provided Dockerfile and docker-compose.yml files into the root of the project folder (overwrite any existing files)
- In ./Dockerfile, on the last line, change the name of the dll file to be the same as your project folder:
  ```
  ENTRYPOINT ["dotnet", "[Name of your Project Folder].dll"]
  ```
  This Dockerfile will create a docker container image complete with all required dependencies and a compiled production version of your web app
- Change all DB connection strings in code to target “mysql” as the server instead of “localhost”:
  ```
  private const string CONNECTION_STRING =
  "Server=mysql;port=3306;Database=[name of
  database];Uid=[username];Pwd=[password];SslMode=none;";
  ```
- Test the web app locally by building the images and running the containers and hitting http://localhost

Step 8:
Copy project folder to EC2 instance

- Create a *private* repo of your web app on Github and push your web app to that repo
- In order to clone this webapp in the EC2 instance you need to have a Personal Access Token (PAT). Github no longer supports passwords for cloning private repos
- Login to github.com and go to Settings / Developer Settings / Personal Access Tokens / and press “Generate New Token” button. Save token string in a text file for safe keeping
- From SSH terminal, clone your web app into EC2 instance with command:
  ```
  git clone <HTTPS of git Repo from GitHub>
  ```
  You will have to enter in your Username / PAT (when it asks for password)
  If any changes are made later on, use “git pull” to update this project folder
- Move into the project folder with cd

Step 9:
Start up docker service with command:
```
sudo service docker start
```
Setup docker to always fire up when the instance starts with command:
```
sudo chkconfig docker on
```
Build the production docker image of the web app with command:
```
docker-compose build
```
This will take some time!

Step 9:
Setup Security Group and Startup Express Server (Node.js)

- because our express server uses port 8080 and the client side also runs on it we need to open up the port
- in AWS EC2 dashboard - click on instance / security tab – click on the link to go directly to security group settings
- click Edit Inbound Rules / Add Rule buttons to with the following settings:
  `HTTP / 80 / 0.0.0.0/0`
- save rules

Step 10:
Run the docker containers with command:
`docker-compose up`

Step 11:
Hit the web app with a browser

- Use the public DNS URL outlined in EC2 instance dashboard – for example:
  `http://ec2-18-207-204-246.compute-1.amazonaws.com`

Other Notes:

- Stopping your EC2 instance in the AWS dashboard will shut it all down but will also assign a different public DNS (URL) to it when you restart the instance
- Any changes made to your web app requires:
  - Shut down docker container(s) with CTRL+C
  - Commit to Git locally and pushed to Github
  - Pull from Github to project folder in EC2 instance
  - Re-build of docker image (will be faster than first build)
  - Restart the docker containers

References:
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html
https://gist.github.com/npearce/6f3c7826c7499587f00957fee62f8ee9