Shipping and running the Docker container in the cloud

We will rely on AWS for our cloud requirements. We will use two of AWS's free offerings for our purpose:

- **Elastic Container Registry** (**ECR**): Here, we will store our Docker image.
- **EC2**: Here, we will create a Linux system to run our API Docker image.

For this section, let's focus only on the ECR part of it. A high-level overview of the steps we will follow to push the Docker image to the cloud is as follows:

1. Configure AWS on the local machine.
2. Create a Docker repository on AWS ECR and push the `fmnist:latest` image.
3. Create an EC2 instance.
4. Install dependencies on the EC2 instance.
5. Create and run the Docker image on the EC2 instance.

The code in the following sections is also summarized as a video walkthrough here: `https://tinyurl.com/MCVP-FastAPI2AWS`.

Let's implement the preceding steps, starting with configuring AWS in the next section.

Configuring AWS

We are going to log in to AWS from Command Prompt and push our Docker image. Let's do it step by step:

1. Create an AWS account at https://aws.amazon.com/ and log in.
2. Install the AWS CLI on your local machine (which contains the Docker image).

The AWS CLI is a command-line interface application for all Amazon services. It should be installed from the official website for your operating system first. Visit `https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html` for more details.

3. Verify that it is installed by running `aws --version` in your local terminal.
4. Configure the AWS CLI. Get the following tokens from https://aws.amazon.com/:
    - `aws_account_id`
    - Access key ID
    - Secret access key

- Region

We can find all the preceding variables in the **Identity and Access Management** (**IAM**) section in AWS. Run `aws configure` in the terminal and give the appropriate credentials when asked:

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key
[None]:wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: region
Default output format [None]:json
```

We have now logged in to Amazon's services from our computer. We can, in principle, access any of their services directly from the terminal. In the next section, let's connect to ECR and push the Docker image.

Creating a Docker repository on AWS ECR and pushing the image

Now, we will create the Docker repository, as follows:

1. After configuring, log in to AWS ECR using the following command (the following code is all one line), providing the preceding region and account ID details at the places that are given in bold in the following code:

```
$ aws ecr get-login-password --region region | docker login --
username AWS --password-stdin
aws_account_id.dkr.ecr.region.amazonaws.com
```

The preceding line of code creates and connects you to your own Docker registry in the Amazon cloud. Much like the Docker registry in the local system, this is where the images are going to reside, but instead, it will be in the cloud.

2. Create a repository from the CLI by running the following:

```
$ aws ecr create-repository --repository-name fmnist_app
```

With the preceding code, a location is now created in the cloud that can hold your Docker images.

3. Tag your local image by running the following command so that when you push the image, it will be pushed to the tagged repository. Remember to give your own `aws_accound_id` and `region` values in the bolded part of the following code:

```
$ docker tag fmnist:latest
aws_account_id.dkr.ecr.region.amazonaws.com/fmnist_app
```

4. Run the following command to push the local Docker image to the AWS repository in the cloud:

```
$ docker push
aws_account_id.dkr.ecr.region.amazonaws.com/fmnist_app
```

We have successfully created a location in the cloud for our API and pushed the Docker image to this location. As you are now aware, this image already has all the components to run the API. The only remaining aspect is to create a Docker container out of it in the cloud, and we will have successfully moved our application to production!

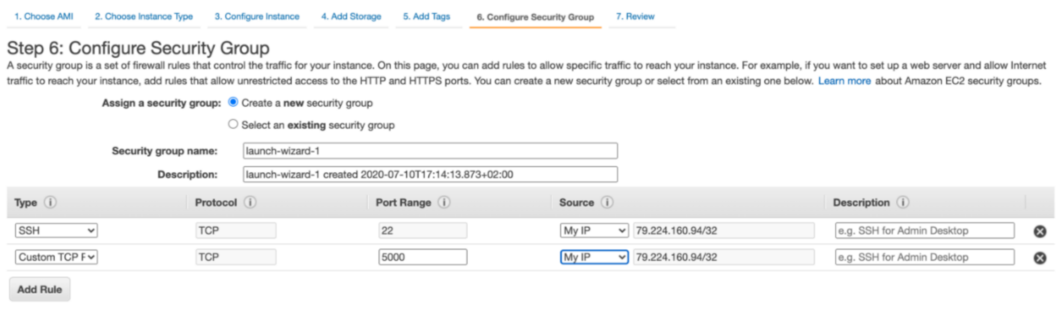
Creating an EC2 instance

Pushing the Docker image to AWS ECR is like pushing code to a GitHub repository. It just resides in one place and we still need to build the application out of it.

For this, you have to create an Amazon EC2 instance that can serve your web application:

1. Go to the search bar of the AWS Management Console and search for `EC2`.
2. Select Launch Instance.
3. You will be given a list of the instances that are available. AWS offers many instances in the free tier. We chose the Amazon Linux 2 AMI - t2.micro instance with 20 GB of space here (you can use other instances as well but remember to change the configuration accordingly).

4. While configuring the instance creation, in the Configure Security Group section, add a rule with Custom TCP set, and set Port Range as `5000` (as we have exposed port `5000` in the Docker image), as shown here:



Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

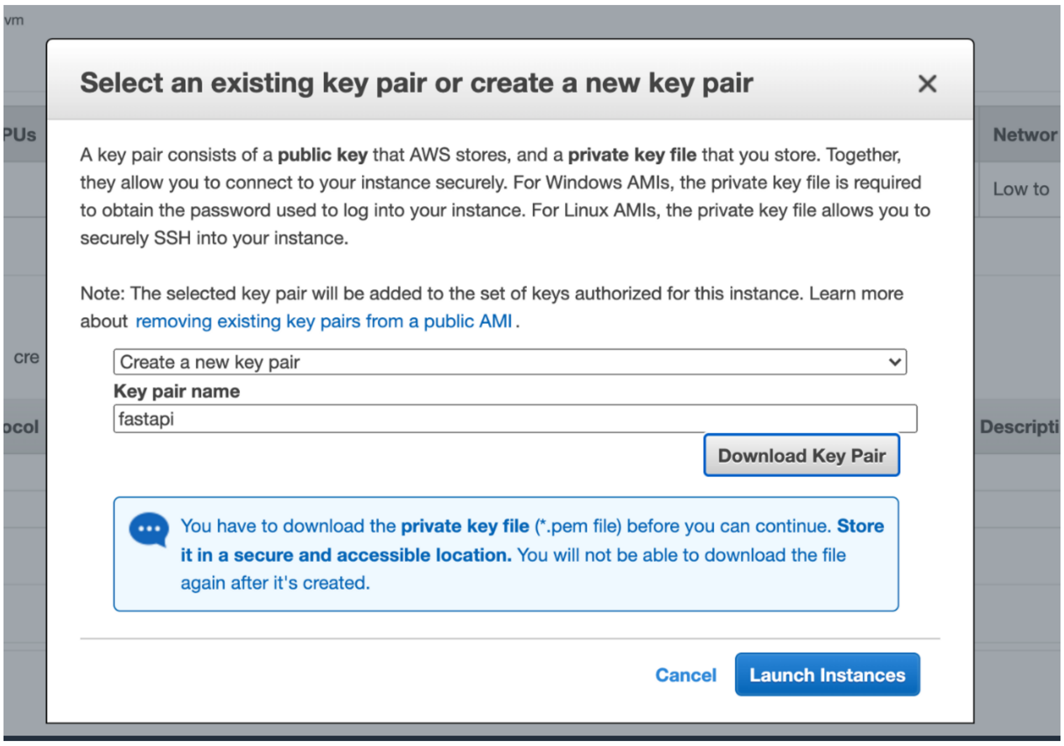
Assign a security group: [✓ Create a **new** security group] Select an **existing** security group

Security group name: launch-wizard-1

Description: launch-wizard-1 created 2020-07-10T17:14:13.873+02:00

Type	Protocol	Port Range	Source		Description
SSH	TCP	22	My IP	79.224.160.94/32	e.g. SSH for Admin Desktop
Custom TCP F	TCP	5000	My IP	79.224.160.94/32	e.g. SSH for Admin Desktop

5. In the **Launch Instances** popup (see the following screenshot), which is the last step, create a new key pair (this will download a `.pem` file, which is needed for logging into the instance). This is as good as a password, so do not lose this file:


Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI.

Create a new key pair

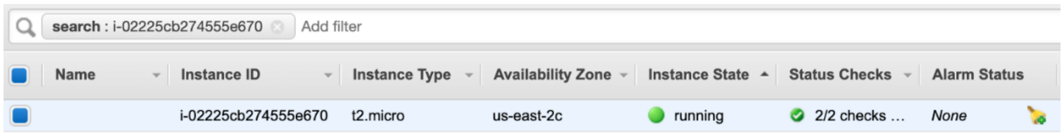
Key pair name

fastapi

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

6. Move the `.pem` file to a secure location and change its permissions by running `chmod 400 fastapi.pem`.

You should see an instance running in your EC2 dashboard at this point:


search : i-02225cb274555e670 Add filter

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
	i-02225cb274555e670	t2.micro	us-east-2c	● running	✓ 2/2 checks ...	None

7. Copy the EC2 instance name that looks like this:

```
ec2-18-221-11-226.us-east-2.compute.amazonaws.com
```

8. Log in to the EC2 instance by using the following command in your local terminal:

```
$ ssh -i fastapi.pem ec2-user@ec2-18-221-11-226.us-east-
2.compute.amazonaws.com
```

We have created an EC2 instance with the necessary space and operating system. Furthermore, we were able to expose port `8000` from the machine and could also take note of the public URL for this machine (this URL will be used by the client for sending `POST` requests). Finally, we were able to log in to it by successfully using the downloaded `.pem` file, treating the EC2 machine like any other machine that can install software.

Pulling the image and building the Docker container

Let's install the dependencies for running the Docker image on the EC2 machine, and then we'll be ready to run the API. The following commands are all needed to be run in the EC2 console that we have logged in to in the previous section (*step 8* of the previous section):

1. Install and configure the Docker image on a Linux machine:

```
$ sudo yum install -y docker
$ sudo groupadd docker
$ sudo gpasswd -a ${USER} docker
$ sudo service docker restart
```

`groupadd` and `gpasswd` ensure that Docker has all the permissions required to run.

2. Configure AWS in an EC2 instance, as you did earlier, and reboot the machine:

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key
[None]:wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]:json
$ reboot
```

3. Log in again to the instance from the local terminal using the following command:

```
$ ssh -i fastapi.pem ec2-user@ec2-18-221-11-226.us-east-
2.compute.amazonaws.com
```

4. Now, from the EC2 logged-in console (which has Docker installed), log in to AWS ECR (change the region that is present in bold in the following code):

```
$ aws ecr get-login --region region --no-include-email
```

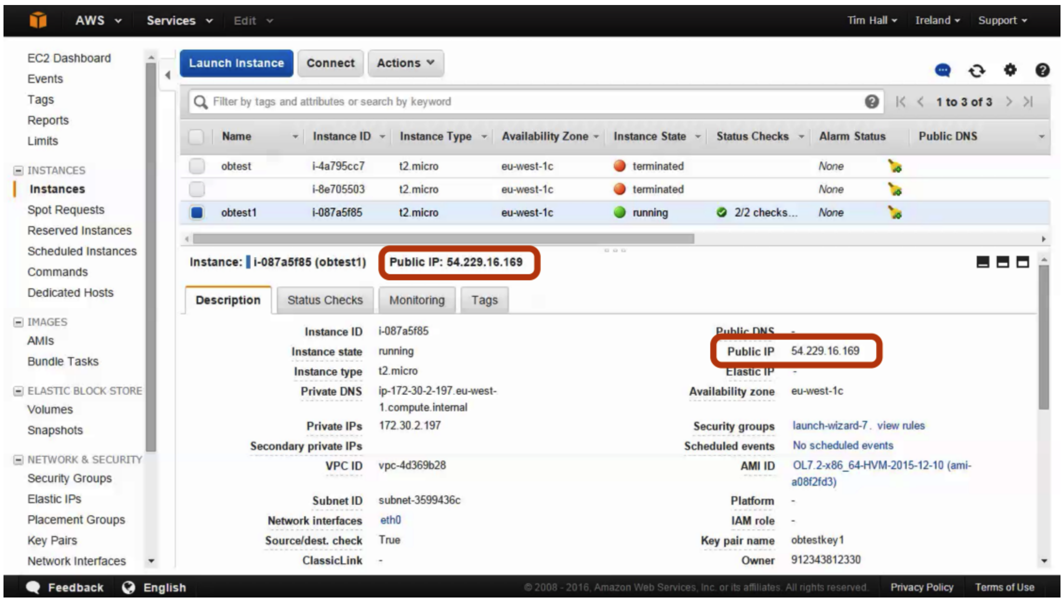
5. Copy the output from the preceding code, then paste and run it in the command line. Once you are successfully logged in to AWS ECR, you will see **Login Succeeded** in the console.
6. Pull the Docker image from AWS ECR:

```
$ docker pull
aws_account_id.dkr.ecr.region.amazonaws.com/fmnist_app:latest
```

7. Finally, run the pulled Docker image in the EC2 machine:

```
docker run -p 5000:5000
aws_account_id.dkr.ecr.region.amazonaws.com/fmnist_app
```

We have our API running on EC2. All we have to do is get the public IP address for the machine and run the `curl` request with this address in place of `127.0.0.1`. You can find this address on the EC2 dashboard at the right of the page:



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
obtest	i-4a795cc7	t2.micro	eu-west-1c	terminated		None	
	i-8e705503	t2.micro	eu-west-1c	terminated		None	
obtest1	i-087a5f85	t2.micro	eu-west-1c	running	2/2 checks...	None	

Instance: i-087a5f85 (obtest1) Public IP: 54.229.16.169

Instance ID	i-087a5f85	Public DNS	-
Instance state	running	Public IP	54.229.16.169
Instance type	t2.micro	Elastic IP	-
Private DNS	ip-172-30-2-197.eu-west-1.compute.internal	Availability zone	eu-west-1c
Private IPs	172.30.2.197	Security groups	launch-wizard-7. view rules
Secondary private IPs		Scheduled events	No scheduled events
VPC ID	vpc-4d369b28	AMI ID	OL7.2-x86_64-HVM-2015-12-10 (ami-a08f2fd3)
Subnet ID	subnet-3599436c	Platform	-
Network interfaces	eth0	IAM role	-
Source/dest. check	True	Key pair name	obtestkey1
ClassicLink	-	Owner	912343812330

8. You can now call a `POST` request from any computer, and the EC2 instance will respond to it, giving us predictions for what type of clothing image we have uploaded:

```
$ curl -X POST "http://54.229.16.169:5000/predict" -H "accept:
application/json" -H "Content-Type: multipart/form-data" -F
"file=@/home/me/Pictures/shirt.png;type=image/png"
```

The preceding code results in the following output:

```
{"class":"Coat","confidence":"0.6488"}
```

In this section, we were able to install the dependencies for EC2, pull the Docker image, and run the Docker container, to enable any user with the URL to make predictions on a new image.