Using AWS EC2 Container Service (ECS) with AWS CLI

In this exercise you will complete the following tasks :

- Install AWS CLI
- Install Docker
- Use AWS CLI to create an AWS ECS Cluster
- Launch ECS AMI into your ECS cluster
- Create a Docker image of a sample PHP application and upload to Docker Hub
- Define an ECS task to use the Docker image
- Use AWS to run the ECS task

See instructions for Installing the AWS Command Line Interface on your own OS at

http://docs.aws.amazon.com/cli/latest/userguide/installing.html

On a Linux OS the basic commands using python pip are

```
curl -O https://bootstrap.pypa.io/get-pip.py
sudo python27 get-pip.py
sudo pip install awscli
```

Confirm that the CLI is installed correctly by viewing the help file. Open a terminal, shell or command prompt, enter aws help and press **Enter**:
Next you should configure the AWS Command Line Interface
For example, the following command includes user input, replaceable text, and output:

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: eu-west-1
Default output format [None]: ENTER
```

For more details on configuration see
http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

Some basic commands for working with EC2 e.g. launch an EC2 instance are :

```
$ aws ec2 create-security-group --group-name devenv-sg --description "security group for development environment in EC2"
{
    "GroupId": "sg-b018ced5"
}
$ aws ec2 authorize-security-group-ingress --group-name devenv-sg --protocol tcp --port 22 --cidr 0.0.0.0/0

$ aws ec2 create-key-pair --key-name devenv-key --query 'KeyMaterial' --output text > devenv-key.pem
```

$ **aws ec2 run-instances --image-id** `ami-d65dfbaf --security-group-ids` *sg-b018ced5* **--count 1 --instance-type** `t2.micro` **--key-name devenv-key --query 'Instances[0].InstanceId'**

Note : The AMI `ami-d65dfbaf` is the Amazon optimized AMI for the eu-west-1 region. You can find the AMI for your region here - http://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-optimized_AMI.html

$ **aws ec2 describe-instances --instance-ids** *i-ec3e1e2k* **--query 'Reservations[0].Instances[0].PublicIpAddress'**

```
"54.183.22.255"
```

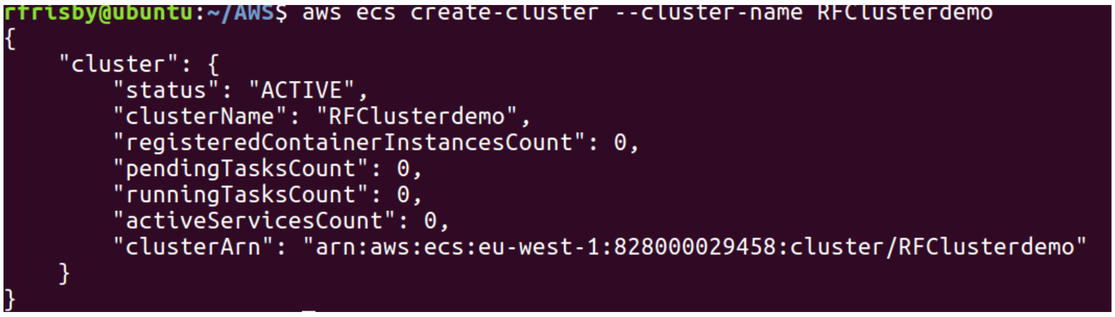
```
$ ssh -i devenv-key.pem ec2-user@54.183.22.255
```

Note you may need to chmod 400 the permissions on your pem key.

Install docker on your local machine – see instructions for your particular OS at https://www.docker.com/products/overview

Once you are a little familiar with the AWS CLI you can interact with the EC2 Container Service to create an ECS Cluster. For detailed instructions see

http://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_AWSCLI.html



```
rfrisby@ubuntu:~/AWS$ aws ecs create-cluster --cluster-name RFClusterdemo
{
    "cluster": {
        "status": "ACTIVE",
        "clusterName": "RFClusterdemo",
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:eu-west-1:828000029458:cluster/RFClusterdemo"
    }
}
```

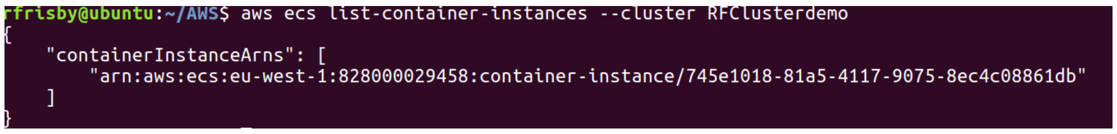
Note you may need to grant IAM permissions to allow your user access the ECS resources.

Next you must launch an ECS container instance into your cluster before running tasks on it – you can do this via the AWS console or try it from the AWS CLI ! **Note:** You must select the **ecsinstancerole** for this ECS AMI. If you don't already have such a role you will need to create this role and attach the Policy AmazonEC2ContainerServiceforEC2Role

By default, your container instance launches into your default cluster. If you want to launch into your own cluster instead of the default, choose the Advanced Details list and paste the following script into the User data field, replacing your_cluster_name with the name of your cluster. Note if you have already created the instance above you will need to stop the instance and under Instance settings you can View/Change User Data to add the details in below.

```
#!/bin/bash
echo ECS_CLUSTER=RFClusterdemo >> /etc/ecs/ecs.config
```

You can launch into your default VPC but obviously you could place this in a private subnet in a custom VPC if this was for example a MongoDB service you wanted to secure.



```
rfrisby@ubuntu:~/AWS$ aws ecs list-container-instances --cluster RFClusterdemo
{
    "containerInstanceArns": [
        "arn:aws:ecs:eu-west-1:828000029458:container-instance/745e1018-81a5-4117-9075-8ec4c08861db"
    ]
}
```

Now we want to create a Docker image for a sample PHP app. You will need to create a Docker Hub account. (see details at http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html)

git clone https://github.com/awslabs/ecs-demo-php-simple-app

cd ecs-demo-php-simple-app

Examine the Dockerfile in this folder. A Dockerfile is a manifest that describes the base image to use for your Docker image and what you want installed and running on it. For more information about Dockerfiles, go to the Dockerfile Reference.

docker build -t *my-dockerhub-username*/amazon-ecs-sample .

Run **docker images** to verify that the image was created correctly and that the image name contains a repository that you can push to (in this example, your Docker Hub user name).
Run the newly built image. The -p 80:80 option maps the exposed port 80 on the container to port 80 on the host system.

docker run -p 80:80 **my-dockerhub-username/amazon-ecs-sample**

```
apache2: Could not reliably determine the server's fully
qualified domain name, using 172.17.0.2 for ServerName
```

Output from the Apache web server is displayed in the terminal window. You can ignore the "Could not reliably determine the server's fully qualified domain name" message.

If you are running Docker locally on a Linux computer, point your browser to `http://localhost/`.

You now need to login to your docker hub account using **docker login** command and then you must push your docker image to your docker hub account using the **docker push rfrisby/amazon-ecs-sample** command.

This will take a few minutes.

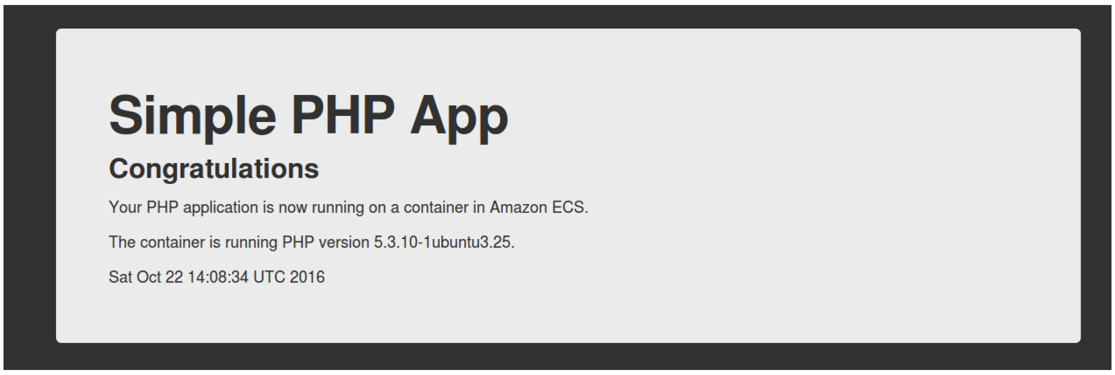
Now to use your Docker Hub copy of the sample app in the ECS task definition edit the simple-app-task-def.json file replacing amazon with your docker hub account. You should also reduce the amount of memory allocated to the containers from 500 to 200 (there are 2 entries in this task definition file). The reason for this is that the basic t2 micro instance that is running these containers only has 1GB of RAM.

aws ecs register-task-definition --cli-input-json file://simple-app-task-def.json

The **register-task-definition** returns a description of the task definition after it completes its registration. You can list the task definitions for your account at any time with the **list-task-definitions** command.

After you have registered a task for your account and have launched a container instance that is registered to your cluster, you can run the registered task in your cluster.
aws ecs run-task --cluster RFClusterdemo --task-definition console-sample-app:1 --count 1

You can check that the sample application is running successfully in your container on your ECS cluster by browsing to the IP address of your ECS AMI. You should see a screen like the one below :



Simple PHP App

Congratulations

Your PHP application is now running on a container in Amazon ECS.

The container is running PHP version 5.3.10-1ubuntu3.25.

Sat Oct 22 14:08:34 UTC 2016

To clean up you should stop your tasks, stop your EC2 instance. You can do this from either the amazon dashboard or the AWS CLI.