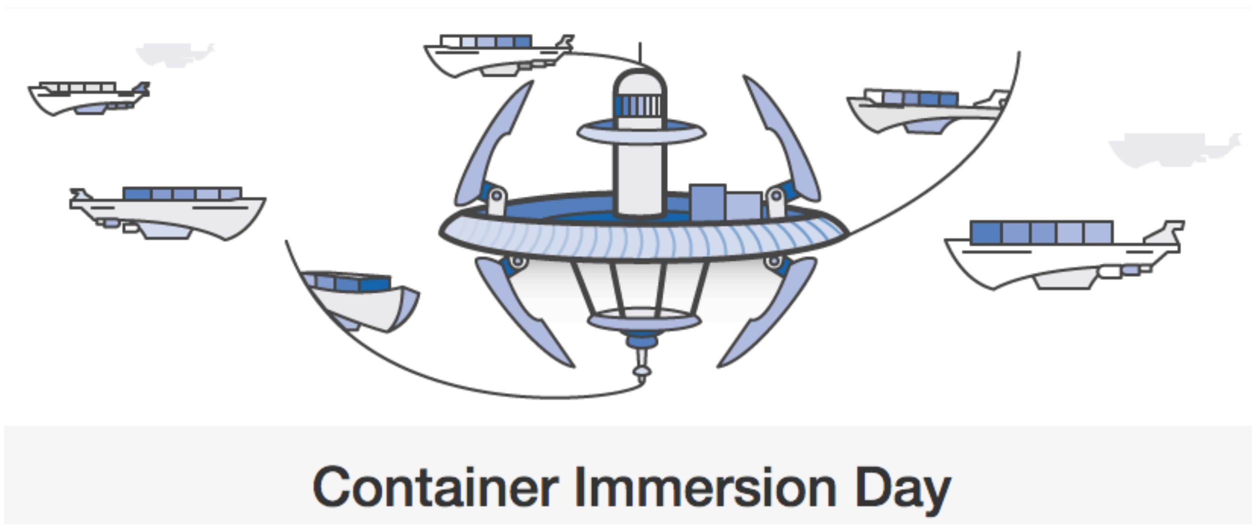
AWS Container Immersion Day: Lab 1



1. From your Cloud9 IDE prompt, verify docker is configured correctly:

```
$ docker info

Containers: 1

 Running: 0

 Paused: 0

 Stopped: 1

Images: 15

Server Version: 18.06.1-ce

Storage Driver: overlay2

 Backing Filesystem: extfs

 Supports d_type: true

 Native Overlay Diff: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

 Volume: local

 Network: bridge host macvlan null overlay

 Log: awslogs fluentd gcplogs gelf journald json-file
logentries splunk syslog

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init
```

```
containerd version:
468a545b9edcd5932818eb9de8e72413e616e86e

runc version: 69663f0bd4b60df09991c08812a60108003fa340

init version: fec3683

Security Options:

 seccomp

  Profile: default

Kernel Version: 4.14.121-85.96.amzn1.x86_64

Operating System: Amazon Linux AMI 2018.03

OSType: linux

Architecture: x86_64

CPUs: 1

Total Memory: 985.8MiB

Name: ip-172-31-93-202

ID:
VSBK:S673:ZSWY:6MNN:CBIO:54NR:GWQK:LEIF:EBQH:JXND:FX3M:
HRCW

Docker Root Dir: /var/lib/docker

Debug Mode (client): false

Debug Mode (server): false

Registry: https://index.docker.io/v1/

Labels:

Experimental: false

Insecure Registries:

 127.0.0.0/8
```

```
Live Restore Enabled: false…
```

2. Prepping the Docker images

At this point, we're going to pretend that we're the developers of both the `web` and `api` microservices, and we will get the latest from our source repo. In this case we will just be using the plain old `curl`, but just pretend you're using `git:`

```
$ curl -O https://s3-us-west-2.amazonaws.com/apn-
bootcamps/microservice-ecs-2017/ecs-lab-code-
20170524.tar.gz

$ tar -xvf ecs-lab-code-20170524.tar.gz
```

Our first step is to build and test our containers locally. If you've never worked with Docker before, there are a few basic commands that we'll use in this workshop, but you can find a more thorough list in the Docker "Getting Started" documentation.

To build your first container, go to the `web` directory. This folder contains our `web` Python Flask microservice:

```
$ cd /home/ec2-user/environment/aws-microservices-ecs-bootcamp-
v2/web
```

To build the container:

```
$ docker build -t ecs-lab/web .
```

This should output steps that look something like this:

```
Sending build context to Docker daemon 4.096 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:latest
 ---> 6aa0b6d7eb90
Step 1 : MAINTAINER widha@amazon.com
```

```
 ---> Using cache
 ---> 3f2b91d4e7a9
```

If the container builds successfully, the output should end with something like this:

```
Removing intermediate container d2cd523c946a
Successfully built ec59b8b825de
```

To view the image that was just built:

```
 $ docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE

ecs-lab/web         latest              0ba3cebe670c        42 seconds ago      435MB

utilities           latest              3a74030b141d        13 minutes ago      289MB

ubuntu              latest              7698f282e524        2 weeks ago         69.9MB

lambci/lambda       python2.7           f81d6f78b2ff        2 weeks ago         682MB

lambci/lambda       nodejs8.10          546cfb23364f        2 weeks ago         742MB

lambci/lambda       python3.6           62cc52d59225        2 weeks ago         814MB

lambci/lambda       nodejs4.3           3f2ef3c2a27c        2 weeks ago         712MB

lambci/lambda       nodejs6.10          84e9cc454dd9        2 weeks ago         727MB

alpine              3.9.4               055936d39205        3 weeks ago         5.53MB
```

To run your container:

```
$ docker run -d -p 3000:3000 ecs-lab/web
```

This command runs the image in daemon mode and maps the docker container port 3000 with the host (in this case our workstation) port 3000. We're doing this so that we can run both microservices on a single host without port conflicts.

To check if your container is running:

```
 $ docker ps
```

This should return a list of all the currently running containers. In this example, it should just return a single container, the one that we just started:

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
7b0d04f4502c        ecs-lab/web         "python app.py"     9 seconds ago       Up 9 seconds        0.0.0.0:3000->3000/tcp   eloquent_noether
```

To test the actual container output:

```
 $ curl localhost:3000/web
```

This should return:

```
 <html><head>...</head><body>hi!  i'm served via Python
+ Flask.  i'm a web endpoint. ...</body></html>
```

Repeat the same steps with the api microservice. Change directory to `/api` and repeat the same steps above:

```
$ cd /home/ec2-user/environment/aws-microservices-ecs-
bootcamp-v2/api
$ docker build -t ecs-lab/api .
$ docker images (should see the api image now)
$ docker run -d -p 8000:8000 ecs-lab/api
$ docker ps (should see both containers running)
$ curl localhost:8000/api
```

The API container should return:

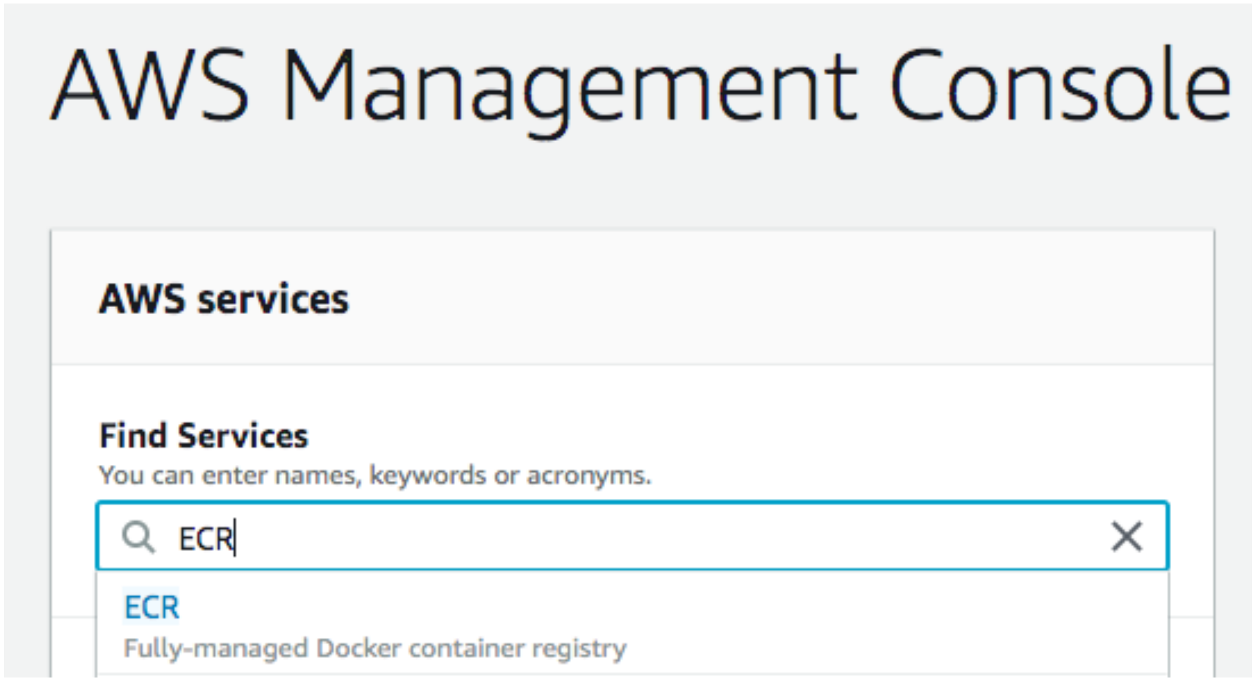
```
{ "response" : "hi!  i'm ALSO served via Python +
Flask.  i'm an API." }
```

We now have two working microservice containers.

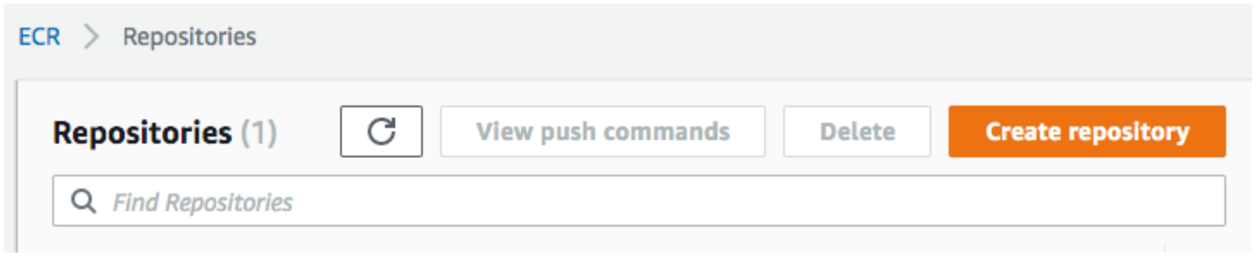
3. Creating container registries with ECR

Once images are built, it's useful to share them and this is done by pushing the images to a container registry. Let's create two repositories in Amazon EC2 Container Registry (ECR).

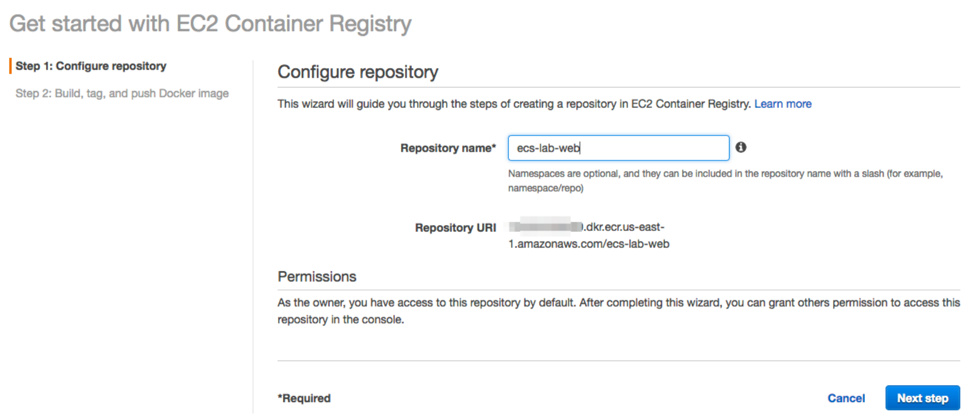
Navigate to the ECS console and Select ECR from AWS Services


AWS Management Console
AWS services
Find Services
You can enter names, keywords or acronyms.
ECR
ECR
Fully-managed Docker container registry

Next **Repositories** and choose **Create repository**.



Name your first repository **ecs-lab-web**:


Get started with EC2 Container Registry

Step 1: Configure repository

Step 2: Build, tag, and push Docker image

Configure repository

This wizard will guide you through the steps of creating a repository in EC2 Container Registry. Learn more

Repository name* ecs-lab-web

Namespaces are optional, and they can be included in the repository name with a slash (for example, namespace/repo)

Repository URI [redacted].dkr.ecr.us-east-1.amazonaws.com/ecs-lab-web

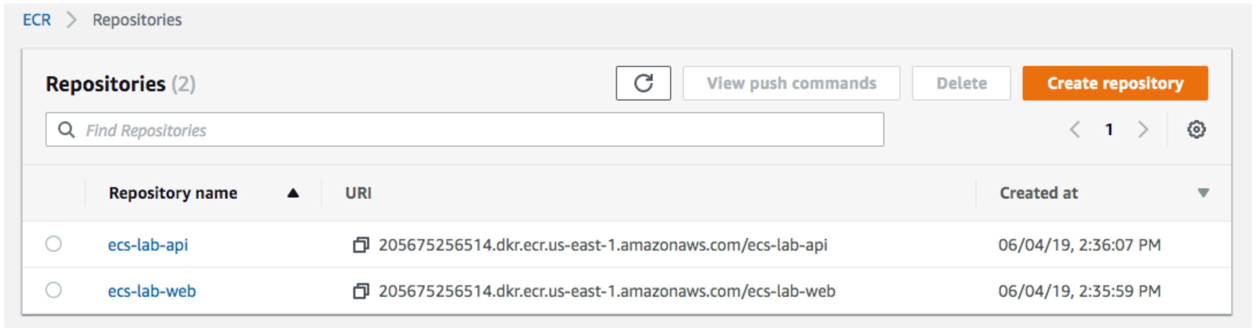
Permissions

As the owner, you have access to this repository by default. After completing this wizard, you can grant others permission to access this repository in the console.

*Required

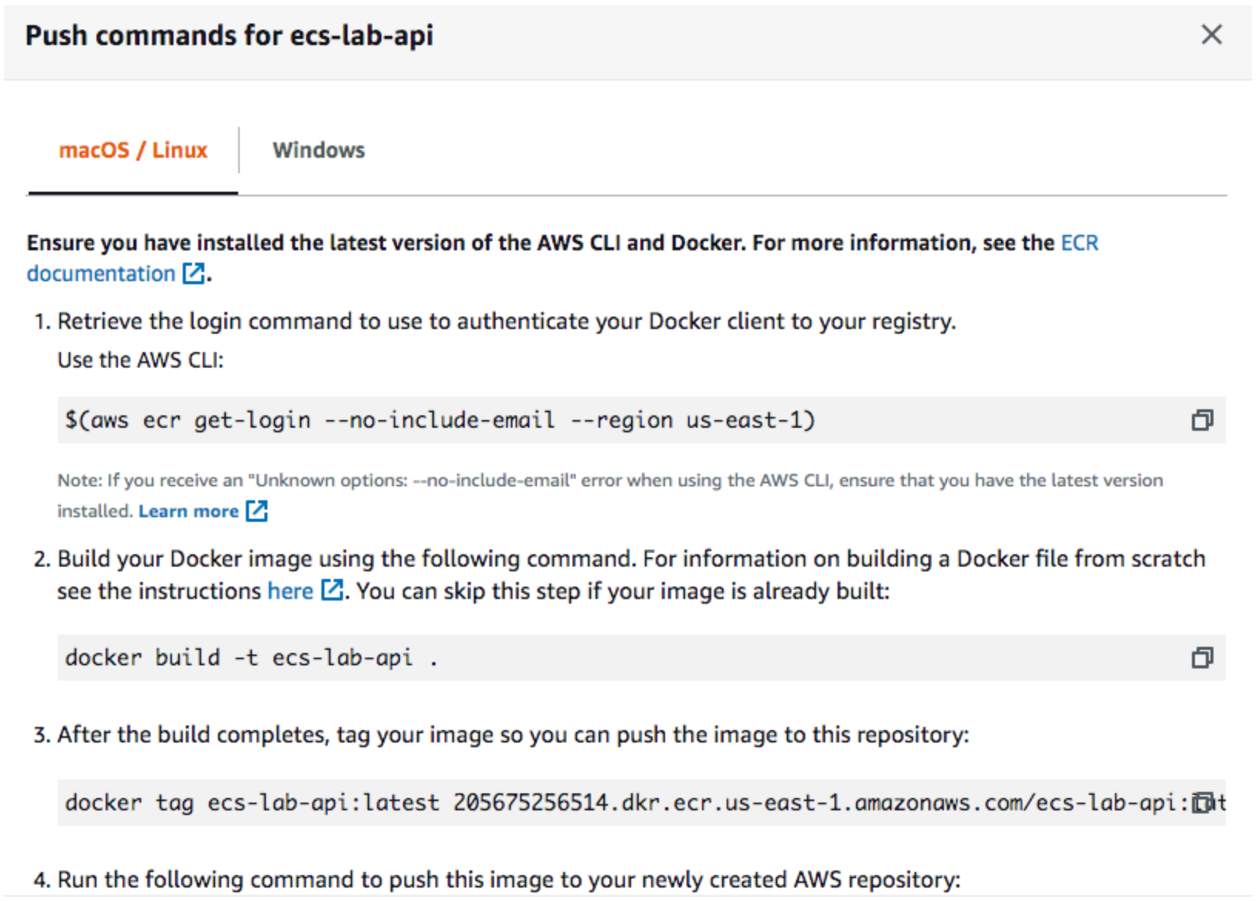
Once you've created the ecs-lab-web repository, repeat the process for the **ecs-lab-api** repository. Take note of the push commands for this second repository. Push commands are unique per repository.

Once you've created the repository, select it and then click View Push Commands



Repositories (2)

	Repository name	URI	Created at
○	ecs-lab-api	205675256514.dkr.ecr.us-east-1.amazonaws.com/ecs-lab-api	06/04/19, 2:36:07 PM
○	ecs-lab-web	205675256514.dkr.ecr.us-east-1.amazonaws.com/ecs-lab-web	06/04/19, 2:35:59 PM



Push commands for ecs-lab-api

macOS / Linux | Windows

Ensure you have installed the latest version of the AWS CLI and Docker. For more information, see the ECR documentation.

1. Retrieve the login command to use to authenticate your Docker client to your registry.
   Use the AWS CLI:

```
$(aws ecr get-login --no-include-email --region us-east-1)
```

Note: If you receive an "Unknown options: --no-include-email" error when using the AWS CLI, ensure that you have the latest version installed. **Learn more**

2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here. You can skip this step if your image is already built:

```
docker build -t ecs-lab-api .
```

3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag ecs-lab-api:latest 205675256514.dkr.ecr.us-east-1.amazonaws.com/ecs-lab-api:lat
```

4. Run the following command to push this image to your newly created AWS repository:

4. Configuring the AWS CLI

On our workstation, we will use the AWS CLI to push images to ECR. Let's configure the CLI by running:

```
$ aws configure
```

This should drop you into a set of prompts. Since our workstation is an EC2 instance pre-configured in an IAM role, the only information required is your preferred region:

```
$ aws configure
AWS Access Key ID: <leave empty>
AWS Secret Access Key: <leave empty>
Default region name [us-east-1]: us-east-1
Default output format [json]: <leave empty>
```

You can confirm that your CLI is setup correctly by running the command to obtain an ECR authentication token.

```
$ aws ecr get-login --no-include-email --region us-east-1
```

This should output something like:

```
docker login -u AWS -p
AQECAHhwm0YaISJeRtJm5n1G6uqeekXuoXXPe5UFce9Rq8/14wAAAy0
wggMpBgkqhkiG9w0BBwagggMaMIIDFgIBADCCAw8GCSqGSIb3DQEHAT
AeBglghkgBZQMEAS4wEQQM+76slnFaYrrZwLJyAgEQgIIC4LJKIDmvE
DtJyr7jO661//6sX6cb2jeD/RP0IA03wh62YxFKqwRMk8gjOAc89ICx
lNxQ6+cvwjewi+8/W+9xbv5+PPWfwGSAXQJSHx3IWfrbca4WSLXQf2B
Dq0CTtDc0+payiDdsXdR8gzvyM7YWIcKzgcRVjOjjoLJpXemQ9liPWe
4HKp+D57zCcBvgUk131xCiwPzbmGTZ+xtE1GPK0tgNH3t9N5+XA2BYY
hXQzkTGISVGGL6Wo1tiERz+WA2aRKE+Sb+FQ7YDDRDtOGj4MwZ3/uMn
OZDcwu3uUfrURXdJVddTEdS3jfo3d7yVWhmXPet+3qwkISstIxG+V6I
IzQyhtq3BXW/I7pwZB9ln/mDNlJVRh9Ps2jqoXUXg/j/shZxBPm33LV
+MvUqiEBhkXa9cz3AaqIpc2gXyXYN3xgJUV7OupLVq2wrGQZWPVoBvH
Pwrt/DKsNs28oJ67L4kTiRoufye1KjZQAi3FIPtMLcUGjFf+ytxzEPu
TvUk4Xfoc4A29qp9v2j98090Qx0CHD4ZKyj7bIL53jSpeeFDh9EXube
qp6idIwG9SpIL9AJfKxY7essZdk/0i/e4C+481XIM/IjiVkh/ZsJzuA
PDIpa8fPRa5Gc8i9h0bioSHgYIpMlRkVmaAqH/Fmk+K00yG8USOAYtP
6BmsFUvkBqmRtCJ/Sj+MHs+BrSP7VqPbO1ppTWZ6avl43DM0blG6W9u
IxKC9SKBAqvPwr/CKz2LrOhyqn1WgtTXzaLFEd3ybilqhrcNtS16I5S
FVI2ihmNbP3RRjmBeA6/QbreQsewQOfSk1u35YmwFxloqH3w/lPQrY1
OD+kySrlGvXA3wupq6qlphGLEWeMC6CEQQKSiWbbQnLdFJazuwRUjSQ
lRvHDbe7XQTXdMzBZoBcC1Y99Kk4/nKprty2IeBvxPg+NRzg+1e0lkk
qUu31oZ/AgdUcD8Db3qFjhXz4QhIZMGFogiJcmo= -e none
https://<account_id>.dkr.ecr.us-east-1.amazonaws.com
```

To register ECR as your Docker repository, copy and paste that output or run:
`$ aws ecr get-login --region us-east-1`

Your shell will execute the output of that command and respond:

```
Login Succeeded
```

If you are unable to login to ECR, check your IAM permissions.

5. Pushing our tested images to ECR

Now that we've tested our images locally, we need to tag and push them to ECR. This will allow us to use them in Task Definitions that can be deployed to an ECS cluster.

You'll need your push commands that you saw during registry creation. You can find them again by going back to the repository (**ECS Console** > **Repositories** > Select the Repository you want to see the commands for > **View Push Commands)**.

To tag and push to the web repository (if you're using a shared account, use your name in the tag: `fred-ecs-lab:latest`):

```
$ docker tag ecs-lab/web:latest <account_id>.dkr.ecr.us-east-1.amazonaws.com/ecs-lab-web:latest
$ docker push <account_id>.dkr.ecr.us-east-1.amazonaws.com/ecs-lab-web:latest
```

This should return something like this:

```
The push refers to a repository [<account_id>.ecr.us-
east-1.amazonaws.com/ecs-lab-web] (len: 1)
ec59b8b825de: Image already exists
5158f10ac216: Image successfully pushed
860a4e60cdf8: Image successfully pushed
6fb890c93921: Image successfully pushed

aa78cde6a49b: Image successfully pushed
```

```
Digest:
sha256:fa0601417fff4c3f3e067daa7e533fbed479c95e40ee96a2
4b3d63b24938cba8
```

To tag and push to the api repository:

```
$ docker tag ecs-lab/api:latest <account_id>.dkr.ecr.us-east-1.amazonaws.com/ecs-lab-api:latest
$ docker push <account_id>.dkr.ecr.us-east-1.amazonaws.com/ecs-lab-api:latest
```

Note: why `:latest`? This is the actual image tag. In most production environments, you'd tag images for different schemes, for example, you might tag the most up-to-date image with `:latest`, and all other versions of the same container with a commit SHA from a CI job. If you push an image without a specific tag, it will default to `:latest`, and untag the previous image with that tag. For more information on Docker tags, see the Docker documentation.

You can see your pushed images by viewing the repository in the ECS Console. Alternatively, you can use the CLI:

```
$ aws ecr list-images --repository-name=ecs-lab-api
{
    "imageIds": [        {
            "imageTag": "latest",
            "imageDigest": "sha256:f0819d27f73c7fa6329644efe8110644e23c248f2f3a9445cbbb6c84a01e108f"
        }
    ]
}
```

You have successfully completed Lab 1.. Keep all the infrastructure you have built running. You will be building on this in Lab 2