# Lab. ECS + deploy tools



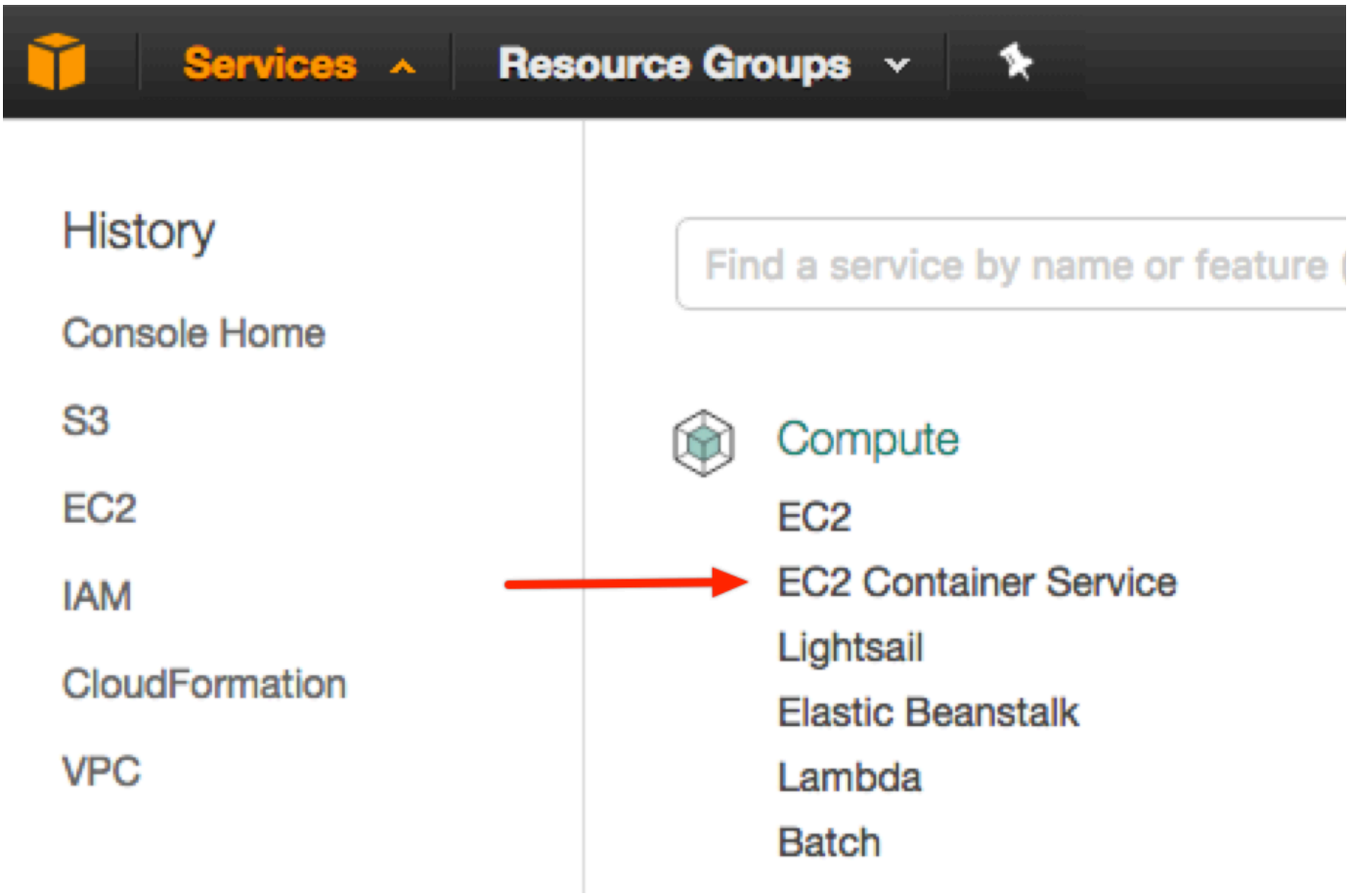
## Introduction

In this Lab Step you will use Amazon ECR to create your own fully-managed Docker container registry within Amazon ECS. ECR hosts your images in a highly available and scalable architecture, and features the same security features (secure transmission, encryption) as other Amazon services. The ECR also integrates with the Docker CLI and ECS to allow for easy and convenient push, pull, and deployment operations.

## Creating the EC2 Container Repository

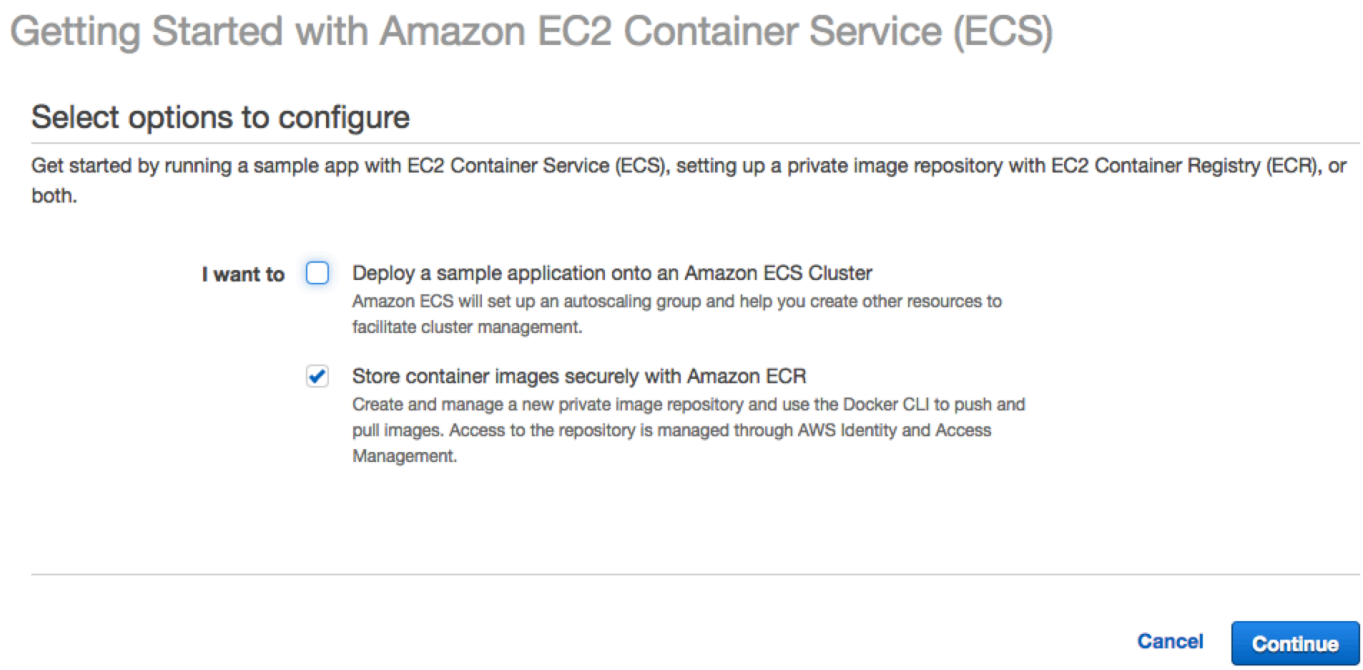
## Instructions

1. Navigate to Services > Compute > EC2 Container Service, or search for *ECS* in the service search box:



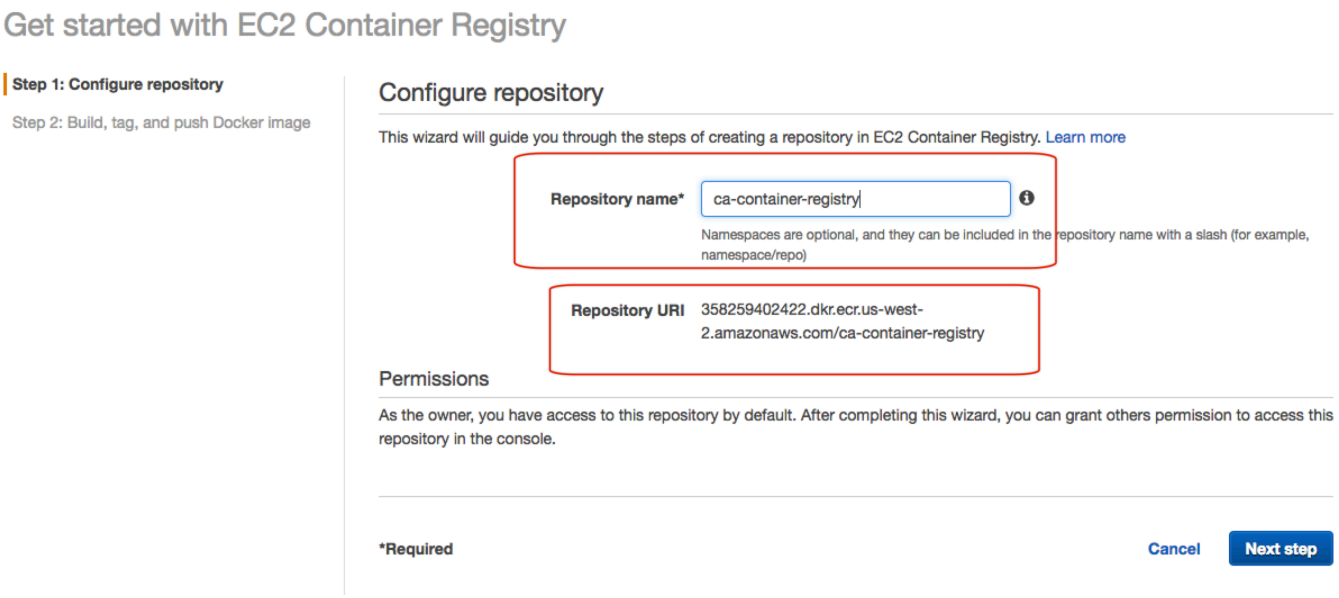

2. Click the blue Get started button.

3. Uncheck the box next to Deploy a sample application onto an Amazon ECS Cluster. Leave the box next to Store container images securely with Amazon ECR checked. Click Continue:




4. In the Configure repository section, enter *ca-container-registry* as the Repository name.

5. Copy and paste the Repository URI (Uniform Resource Identifier). You will need this URI to reference your Docker images in a later Lab Step. Click Next step:




6. Although not used in this Lab, Step 2: Build, tag, and push Docker image provides you with a convenient list of Docker commands configured to reference your new repository. You can reference these commands whenever you need by revisiting your repository and clicking the View Push Commands button. Click Done:

## Build, tag, and push Docker image

Now that your repository exists, you can push a Docker image by following these steps:

**Successfully created repository**
358259402422.dkr.ecr.us-west-2.amazonaws.com/ca-container-registry

To install the AWS CLI and Docker and for more information on the steps below, visit the ECR documentation page.

1) Retrieve the `docker login` command that you can use to authenticate your Docker client to your registry:
Note:
If you receive an "Unknown options: --no-include-email" error, install the latest version of the AWS CLI. Learn more

```
aws ecr get-login --no-include-email --region us-west-2
```

2) Run the `docker login` command that was returned in the previous step.
Note:
If you are using Windows PowerShell, run the following command instead.

```
Invoke-Expression -Command (aws ecr get-login --no-include-email --region us-west-2)
```

3) 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 ca-container-registry .
```

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

```
docker tag ca-container-registry:latest 358259402422.dkr.ecr.us-west-2.amazonaws.com/ca-container-registry:latest
```

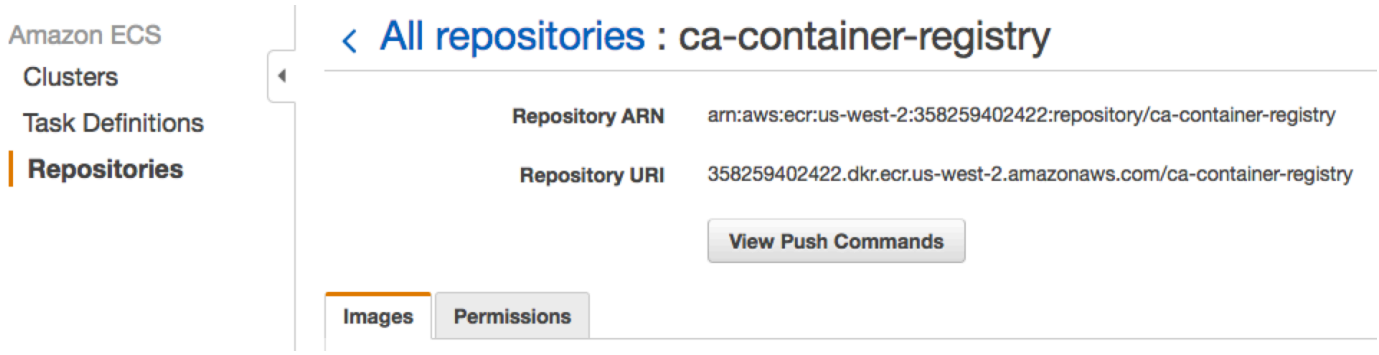
5) Run the following command to push this image to your newly created AWS repository:

```
docker push 358259402422.dkr.ecr.us-west-2.amazonaws.com/ca-container-registry:latest
```

*Required

Done

7. Click the Repositories link and view your new, currently-empty repository. This location is a convenient place to view your repository URI, push commands, and list of images whenever required:




## Summary

In this Lab Step you created a repository for Docker images within AWS. This allows you to integrate your Docker images more closely with ECS and other Amazon services, such as CodeBuild.

# Building the Docker Images

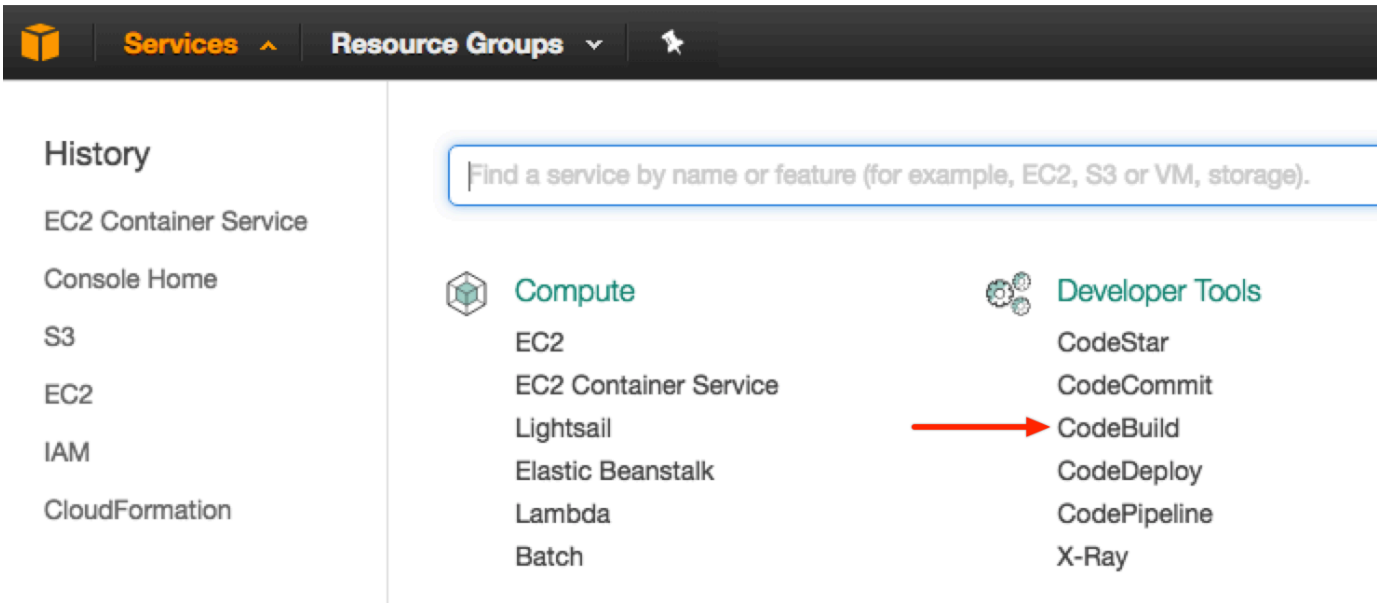
CodeBuild is a fully-managed build service. CodeBuild can compile code, run tests, and produce packages you can deploy on any compatible resource. Using CodeBuild in this Lab eliminates the need for you to create a build server or install any software on your local machine. In this Lab Step you will use CodeBuild to build the Docker images containing the two applications you need to complete the Lab.

**Prepare files:**

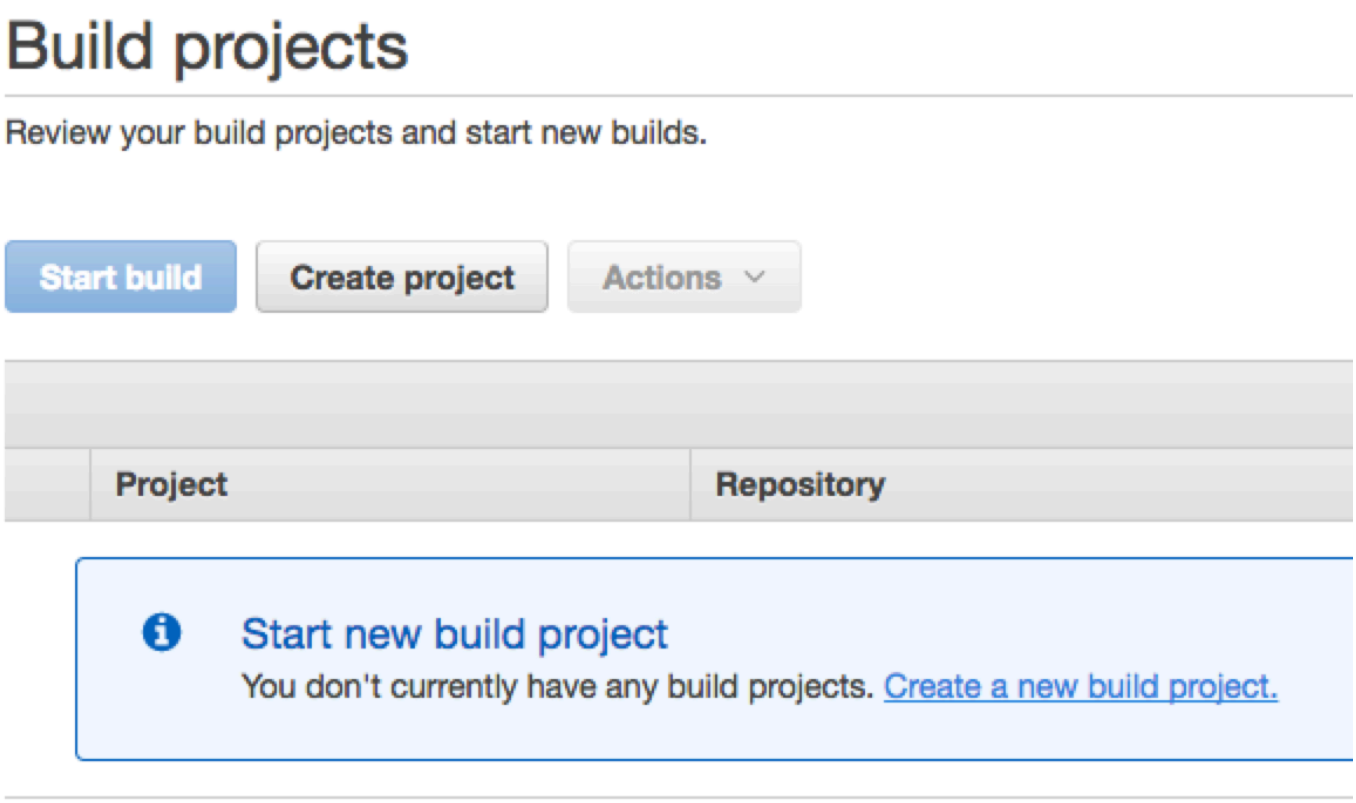
In your Lab folder ecs_lab.zip you have two more folders: ecslabgreen and ecslabblue. In each of them exists a file server.js. Replace ? in line app.use('/devops**?**/api', router); with your number. Put all files in each directory in zip archives ecslabgreen.zip and ecslabblue.zip.

## Instructions

1. Navigate to Services > Developer Tools > CodeBuild, or search for *CodeBuild* in the service search box:




2. Click Get started. If you arrive at the Build Projects page, click Create project:

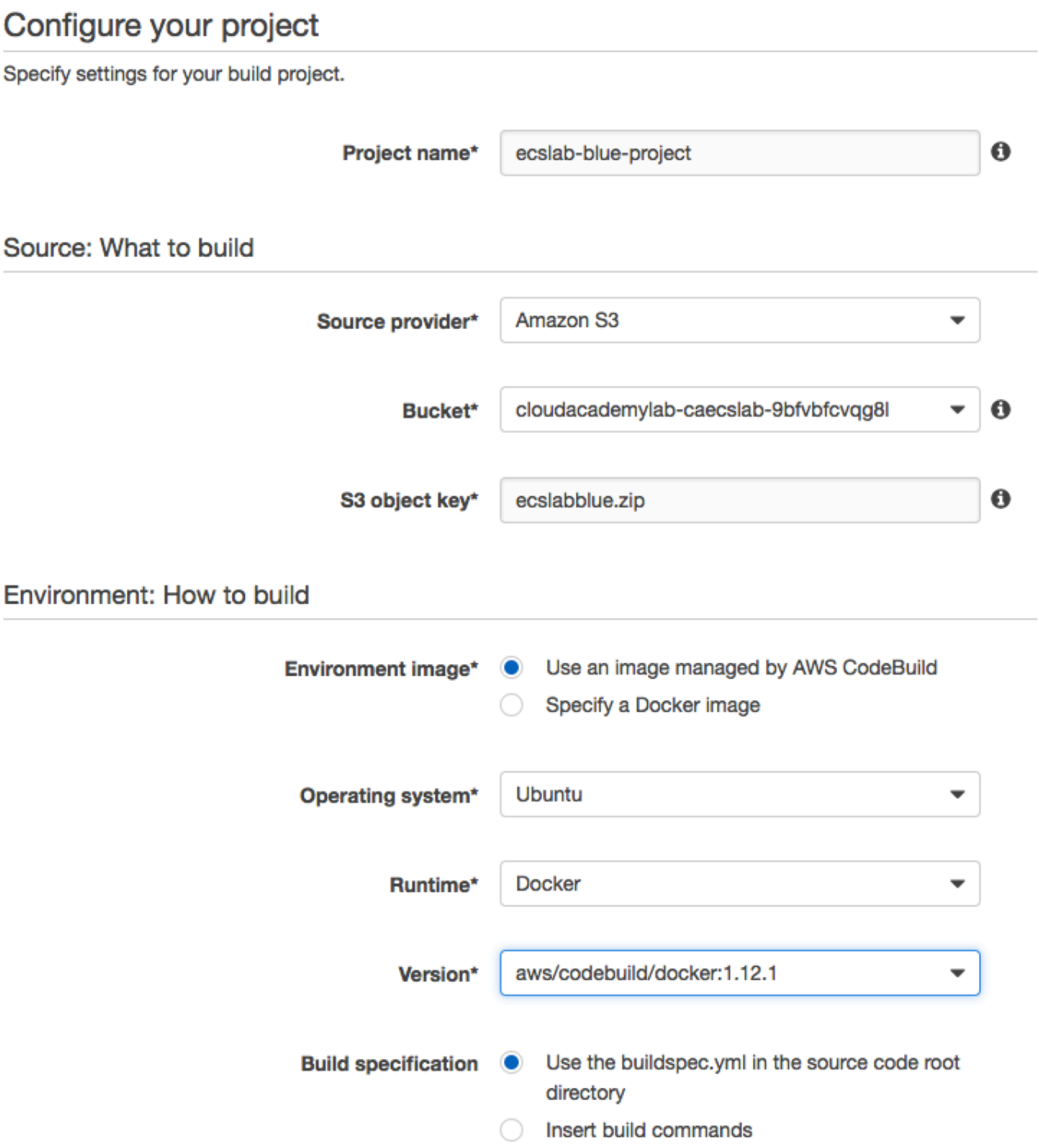



3. In the Step 1: Configure project area specify the following details:

- Project name: *ecslab-blue-project*
- Source provider: Amazon S3
- Bucket: devops?
- S3 object key: *ecslabblue.zip*
- Environment image: Use an image managed by AWS CodeBuild
- Operating system: Ubuntu
- Runtime: Docker
- Version: aws/codebuild/docker:1.12.1

*Note*: The Docker version listed is the latest at lab creation. In the future, the version will update. Choose the latest version if the version listed is unavailable.

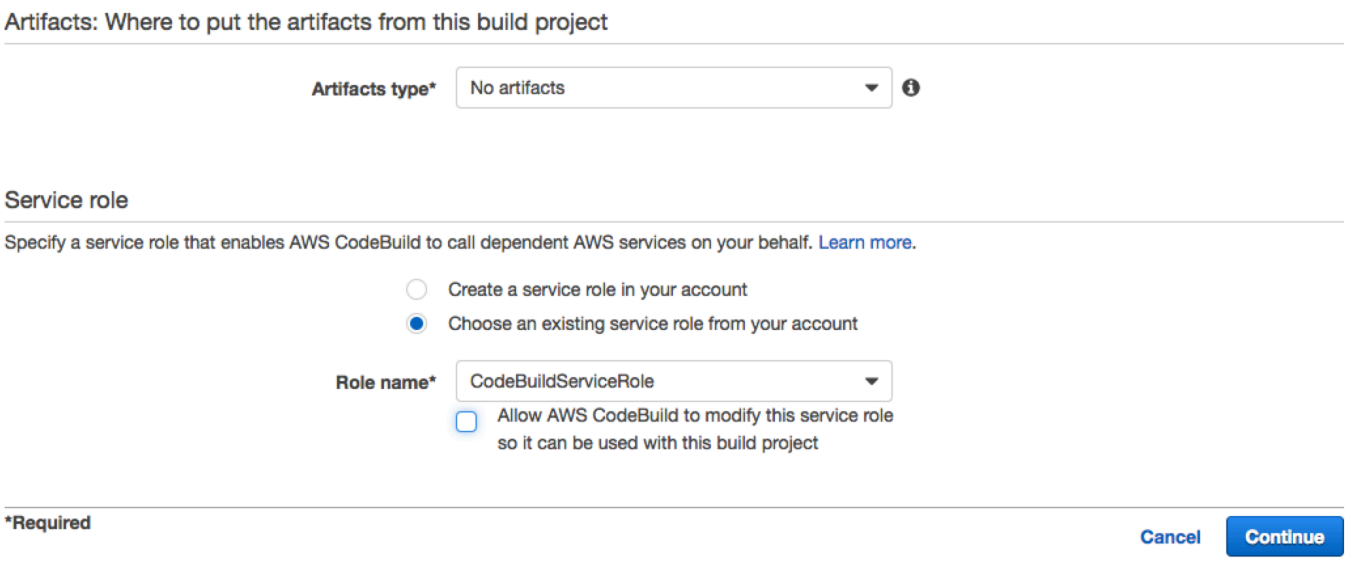
- Build specification: Use the buildspec.yml in the source code root directory




The project source files and Amazon S3 bucket are created for you by the Cloud Academy Lab environment. The project uses Ubuntu Linux with Docker installed as the build environment to create the image. The build spec is a collection of build commands and settings that provide instructions on how to build the Docker image and where to push it once complete. For the purposes of differentiating the images and the applications they contain in this Lab, the build spec also tags the images with distinct names. CodeBuild will tag your blue application image with *testblue* and your green application image with *testgreen* due to instructions in the source files.

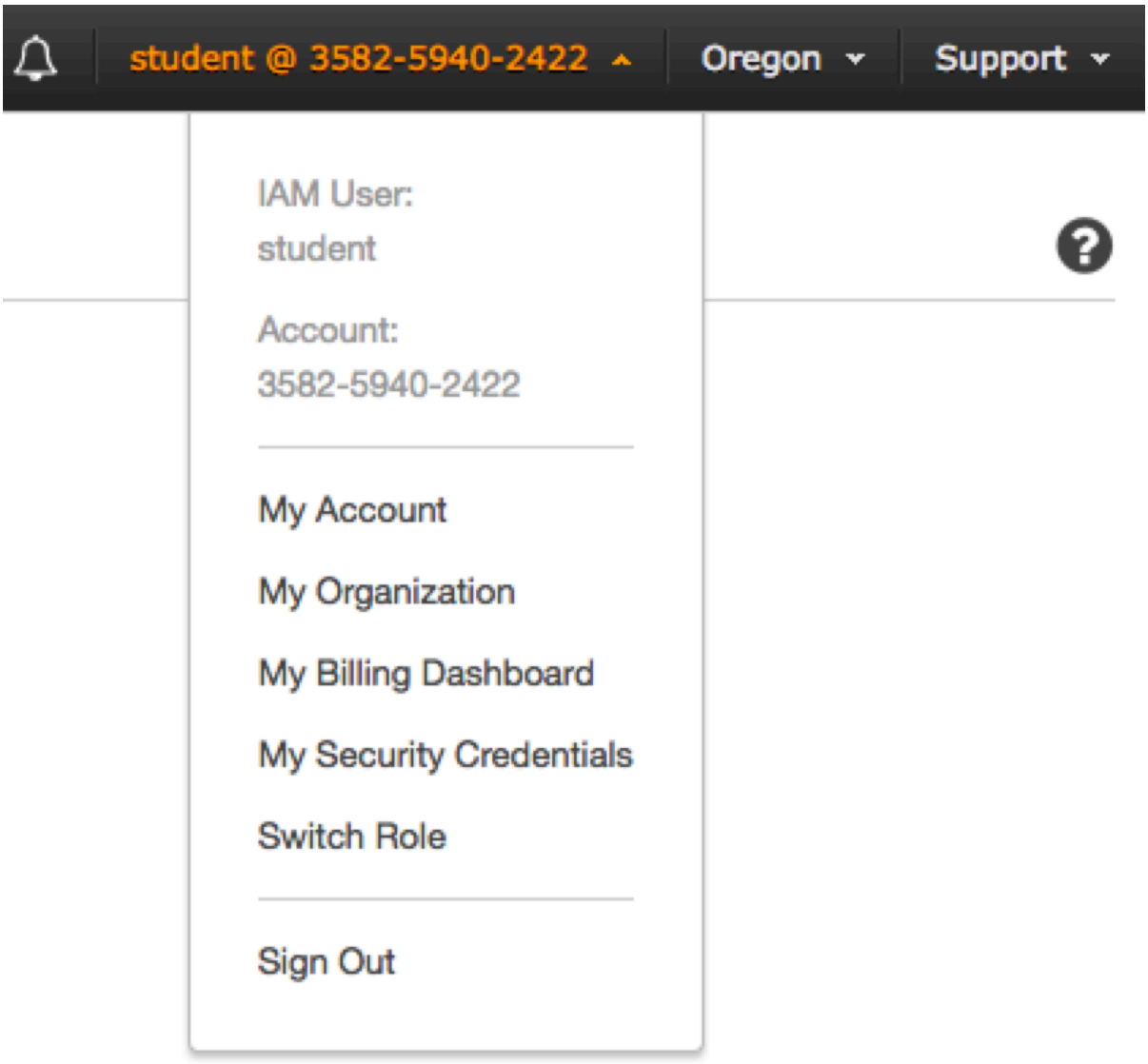
4. Scroll down the page to continue entering the final project details. Click Continue:

- Artifacts type: No artifacts
- Service role: Choose an existing service role from your account
- Role name: CodeBuildServiceRole (Remove the check from Allow AWS CodeBuild to modify this service role...)



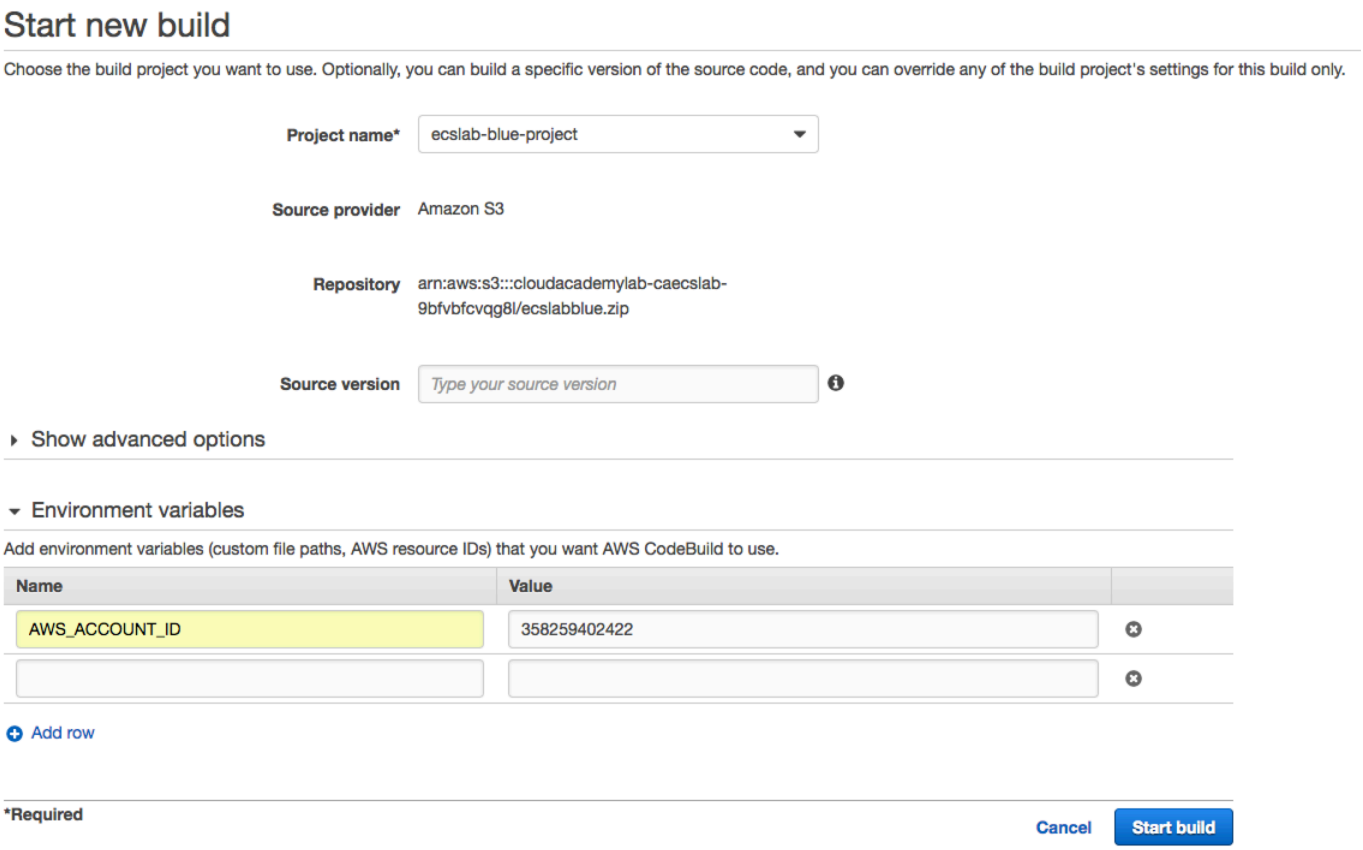

5. Review the project parameters in Step 2: Review. Click Save and Build.

6. Expand the student account information in the upper-right corner of the screen and record the Account number:




7. Expand the Environment variables area in the Start new build section. In the Name field enter *AWS_ACCOUNT_ID*. In the Value field enter the 12-digit student account ID. Remove the hyphens as shown in the following image. Also you will need *AWS_REGION* set to your region.

Click Start build:




The sample application uses the AWS account variable to locate the Docker image repository you created earlier. Once CodeBuild finishes building the Docker image, it executes a Docker push command that stores the image in the repository.

8. Observe the Phase Details section as each step of the build process proceeds and completes. Wait approximately one minute until the Status fi eld shows Succeeded. CodeBuild is building a Docker container image containing one of the custom applications (blue) made for this Lab:

## ecslab-blue-project:e5c6ee2a-b25f-4aca-89d1-1d1030be6d59 Succeeded

Review your build details as it progresses.

### Build

| | |
|---|---|
| **Build ARN** | arn:aws:codebuild:us-west-2:358259402422:build/ecslab-blue-project:e5c6ee2a-b25f-4aca-89d1-1d1030be6d59 |
| **Build project** | ecslab-blue-project |
| **Source provider** | Amazon S3 |
| **Repository** | arn:aws:s3:::cloudacademylab-caecslab-9bfvbfcvqg8l/ecslabblue.zip |
| **Start time** | 1 minute ago |
| **End time** | 1 second ago |
| **Status** | Succeeded |
| **Initiator** | student |

▸ Build details

▸ Environment variables

### Phase details

| | Name | Status | Duration | Completed |
|---|---|---|---|---|
| ▸ | SUBMITTED | Succeeded | | 1 minute ago |
| ▸ | PROVISIONING | Succeeded | 21 secs | 48 seconds ago |
| ▸ | DOWNLOAD_SOURCE | Succeeded | 4 secs | 44 seconds ago |
| ▸ | INSTALL | Succeeded | | 44 seconds ago |
| ▸ | PRE_BUILD | Succeeded | 4 secs | 39 seconds ago |
| ▸ | BUILD | Succeeded | 18 secs | 21 seconds ago |
| ▸ | POST_BUILD | Succeeded | 9 secs | 12 seconds ago |
| ▸ | UPLOAD_ARTIFACTS | Succeeded | | 12 seconds ago |
| ▸ | FINALIZING | Succeeded | 5 secs | 6 seconds ago |

9. Scroll down to the Build logs section to view information about the build process. This screen only shows the last 20 lines of the build log. The lines in the following image show part of the POST_BUILD stage where the layers of the image are pushed to the repository before finalizing the build:

# Build logs

Showing the last 20 lines of build log below. View entire log

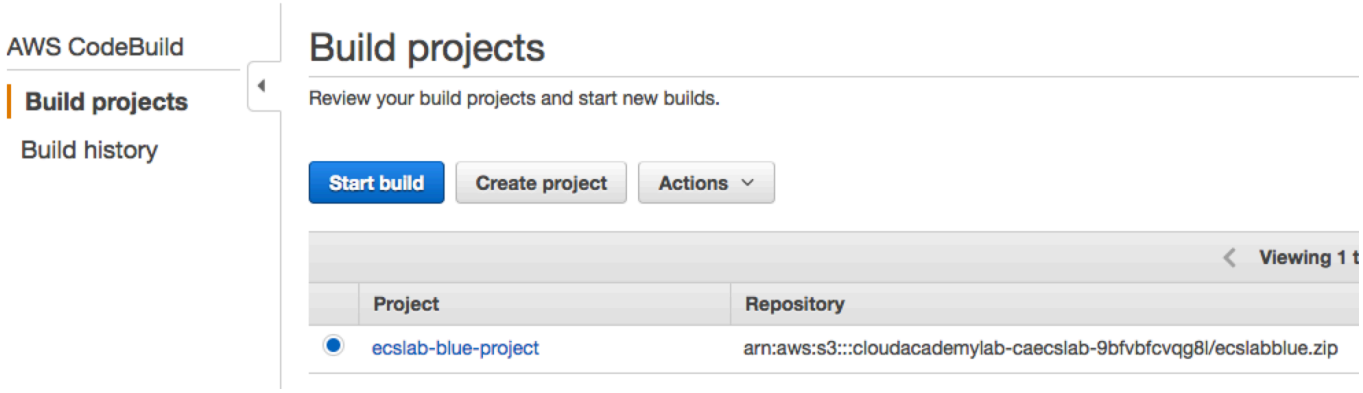
```
5bef08742407: Preparing
3e89bd0d5acf: Waiting
f7e883283ebc: Waiting
0a19bde117a5: Waiting
5bef08742407: Waiting
2531a8d82cf7: Pushed
e64991fc8bf5: Pushed
4d9359363687: Pushed
8253865cc65c: Pushed
26c56125406d: Pushed
3e89bd0d5acf: Pushed
f7e883283ebc: Pushed
5bef08742407: Pushed
0a19bde117a5: Pushed
testblue: digest: sha256:bfefadb6e7ebaeea21d2859d2c1ee8bae0b31db35d2dfa789d673a8aa0039fc1 size: 2200

[Container] 2017/07/21 20:17:17 Phase complete: POST_BUILD Success: true
[Container] 2017/07/21 20:17:17 Phase context status code: Message:
[Container] 2017/07/21 20:17:17 Preparing to copy artifacts
[Container] 2017/07/21 20:17:17 No artifact files specified
```

10. Click View entire log to go to CloudWatch logs and view detailed information about every step in the build process. Examining and retaining these logs can be useful for troubleshooting purposes:

| Time (UTC +00:00) | Message |
|---|---|
| 2017-07-21 | |
| 20:17:09 | Step 10 : CMD npm start |
| 20:17:09 | ---> Running in bb4ff8fbf96d |
| 20:17:09 | ---> 9e5ff243296d |
| 20:17:09 | Removing intermediate container bb4ff8fbf96d |
| 20:17:09 | Successfully built 9e5ff243296d |
| 20:17:09 | |
| 20:17:09 | [Container] 2017/07/21 20:17:08 Running command docker tag ca-container-registry:testblue $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/ca-container-registry:testblue |
| [Container] 2017/07/21 20:17:08 Running command docker tag ca-container-registry:testblue $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/ca-container-registry:testblue | |
| 20:17:09 | |
| 20:17:09 | [Container] 2017/07/21 20:17:08 Phase complete: BUILD Success: true |
| [Container] 2017/07/21 20:17:08 Phase complete: BUILD Success: true | |
| 20:17:09 | [Container] 2017/07/21 20:17:08 Phase context status code: Message: |
| [Container] 2017/07/21 20:17:08 Phase context status code: Message: | |
| 20:17:09 | [Container] 2017/07/21 20:17:08 Entering phase POST_BUILD |
| [Container] 2017/07/21 20:17:08 Entering phase POST_BUILD | |
| 20:17:09 | [Container] 2017/07/21 20:17:08 Running command echo Build completed on `date` |
| [Container] 2017/07/21 20:17:08 Running command echo Build completed on `date` | |
| 20:17:09 | Build completed on Fri Jul 21 20:17:08 UTC 2017 |
| 20:17:09 | |
| 20:17:09 | [Container] 2017/07/21 20:17:08 Running command echo Pushing the Docker image... |
| 20:17:09 | Pushing the Docker image... |
| 20:17:09 | |
| 20:17:09 | [Container] 2017/07/21 20:17:08 Running command docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/ca-container-registry:testblue |
| [Container] 2017/07/21 20:17:08 Running command docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/ca-container-registry:testblue | |

11. Return to CodeBuild > Build projects and click Create project:




For convenience's sake and for the purposes of this lab demonstration, you will create both the blue and green application container images at the same time. In a real environment your releases may be weeks or months apart, or several times a day, depending on your development and deployment approach.

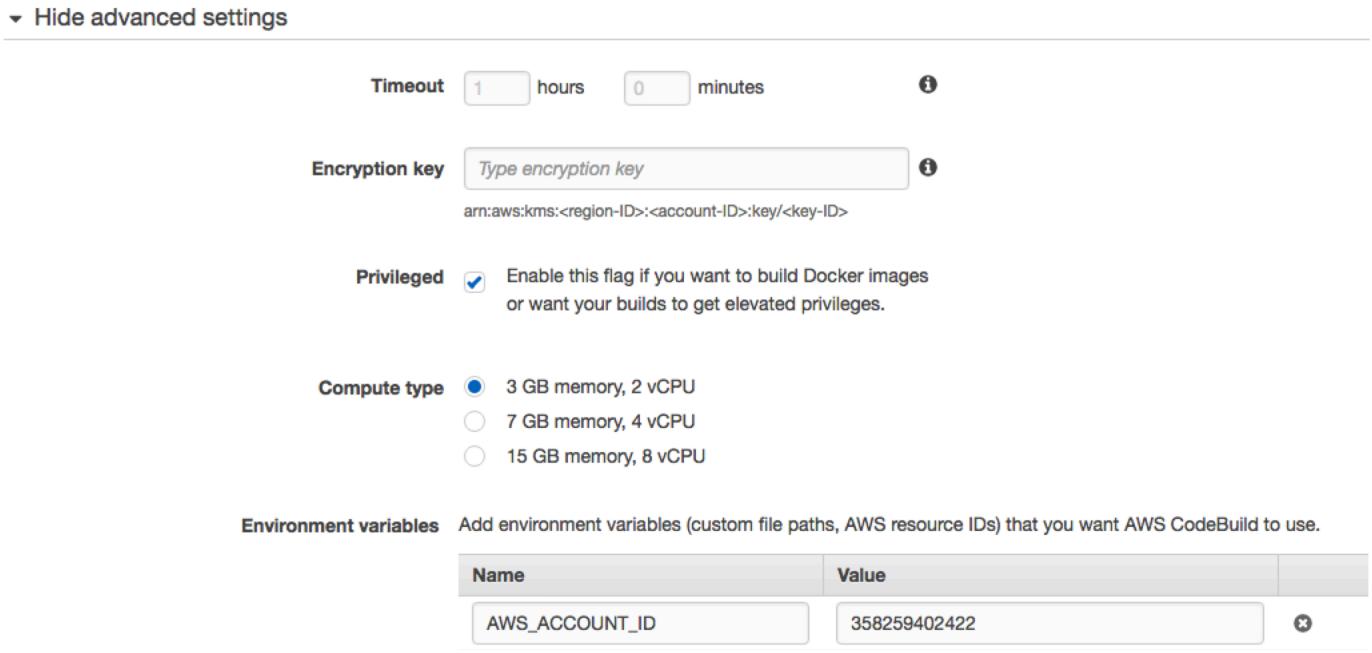
12. The steps and details for creating the green project are almost identical to those for the blue project. Enter the following details in the Configure your project section:

- Project name: *ecslab-green-project*
- Source provider: Amazon S3
- Bucket: cloudacademylab-caecslab-XXXXXXXXXX (The final values are randomized to prevent name conflicts in Amazon S3.)
- S3 object key: *ecslabgreen.zip*
- Environment image: Use an image managed by AWS CodeBuild
- Operating system: Ubuntu
- Runtime: Docker
- Version: aws/codebuild/docker:1.12.1
- Privileged: Yes

*Note*: The Docker version listed is the latest at lab creation. In the future, the version will update. Choose the latest version if the version listed is unavailable.

- Build specification: Use the buildspec.yml in the source code root directory
- Artifacts type: No artifacts
- Service role: Choose an existing service role from your account
- Role name: CodeBuildServiceRole (Remove the check from Allow AWS CodeBuild to modify this service role...)

13. One important difference for your second CodeBuild project is the ability to enter advanced settings in the initial project creation screen. Expand the Show advanced settings section. Add the *AWS_ACCOUNT_ID* in the Environment variables section. Remember to delete the hyphens from the AWS account number. Click Continue:

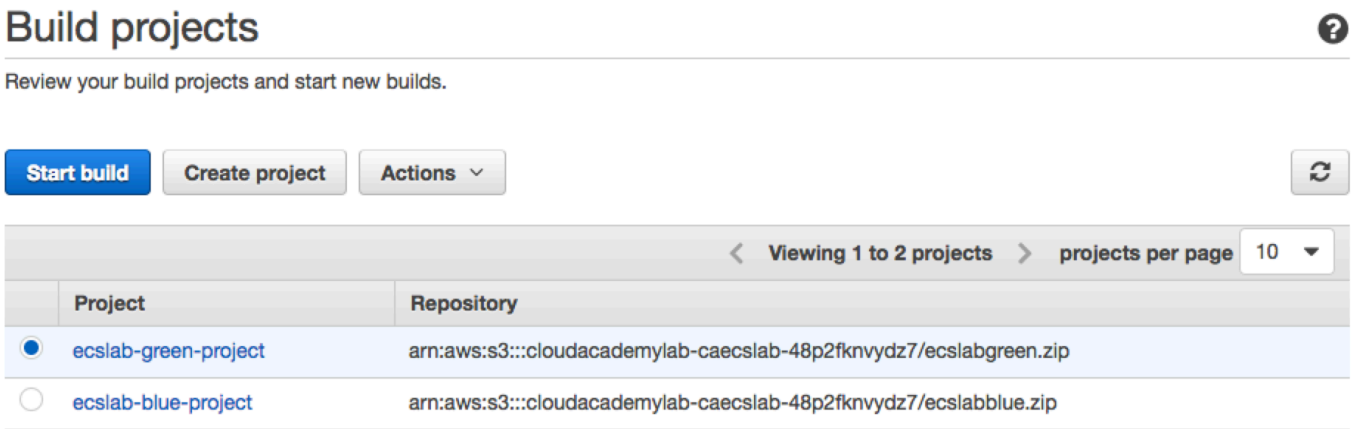



14. Review the project details in the Review section and click Save and Build.

15. Click Start build.

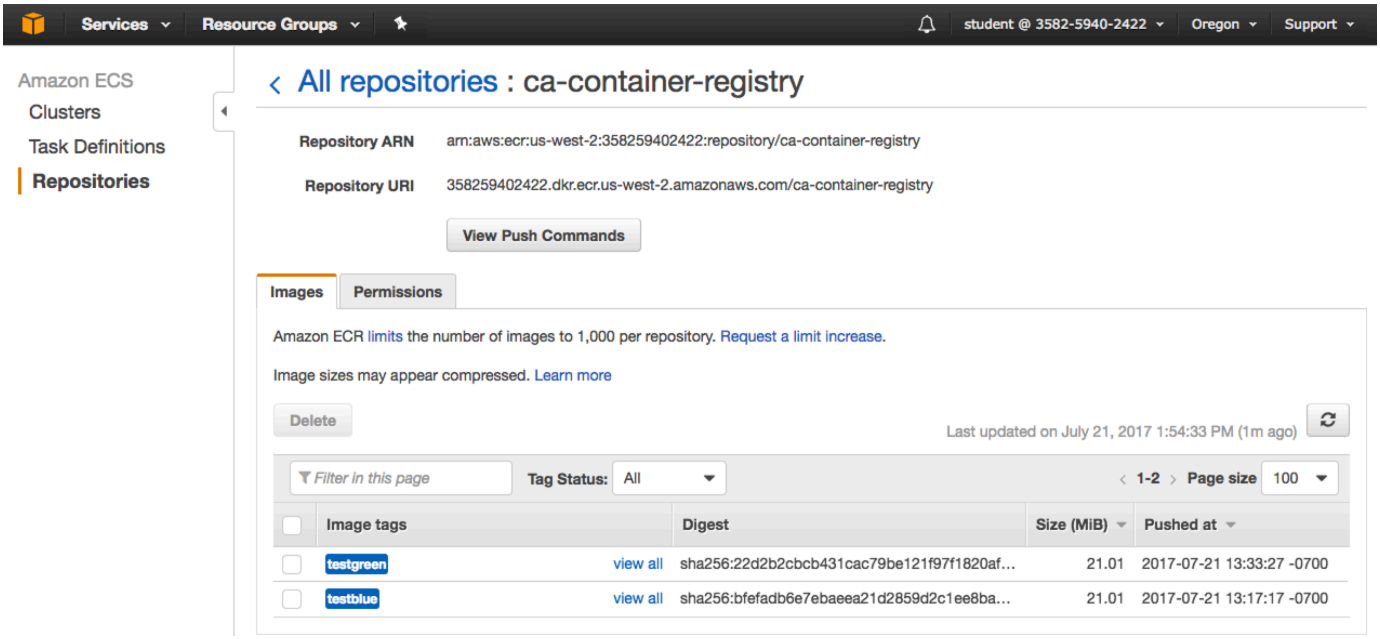
16. Wait approximately one minute for the Status field in the Build section to show Succeeded. Examine the build log and details, if you wish. All of the build process information is very similar to the blue project, but the new green container has different unique identifiers.

17. Return to CodeBuild > Build projects and view your two projects. You can initiate a new build and create new containers at any time by selecting a project and clicking Start build. This is not necessary for this Lab, but a useful tool in case you upload a new version of the application to one of your source locations:




So where do your container images go? You are building Docker container images, but where can you view the output of your build operations?

18. Return to Services > EC2 Container Service and click Repositories. Click on ca-container-registry to view your Docker container images:




The application buildspec.yml files tag the container images with testgreen or testblue, depending on the application they contain.

## Summary

In this Lab Step you used CodeBuild to create two Docker container images with two different applications. You modified some variables and examined the build phases and logs. You also viewed your Docker images and details in your ECR repository.

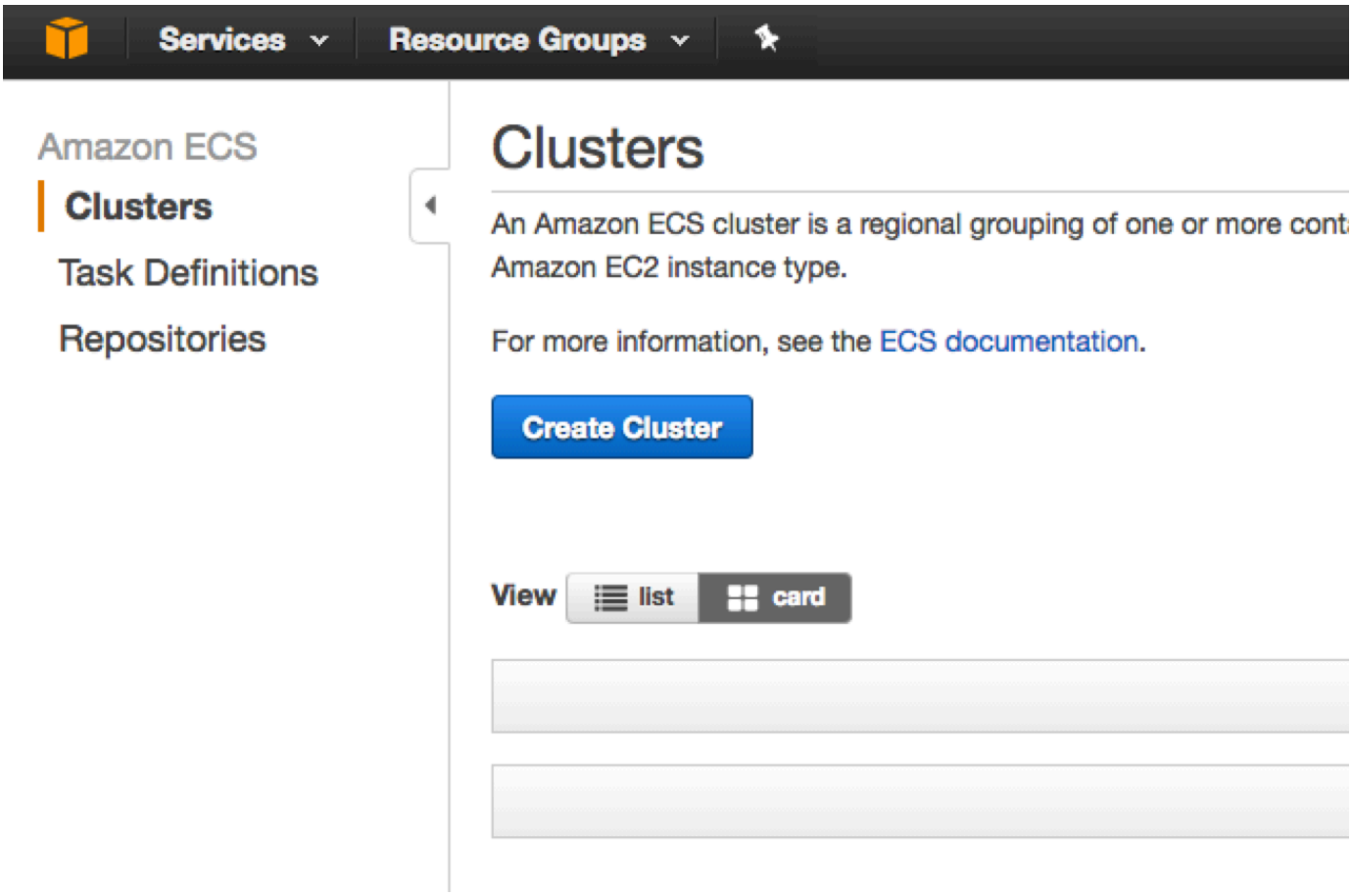
# Creating the ECS Cluster

## Introduction

An ECS cluster is a logical grouping of container instances where you can define task definitions to execute tasks. A task is the instantiation of a task definition on a container instance within an ECS cluster. You can customize several features of a cluster such as EC2 instance type, storage allocation, and networking specifics. In this Lab Step you will create an ECS cluster using a single t2.micro EC2 instance and some pre-configured networking resources.

## Instructions

1. Click on Clusters and click Create Cluster:




2. In the Create Cluster section enter the following details. Leave other settings as the defaults. Click Create:

- Cluster name: *ecslab-cluster*
- EC2 instance type: t2.micro

*Warning!*: The lab environment is restricted to only allow the t2.micro instance type. You must select a t2.micro or the cluster will not launch.

- VPC: Choose the VPC with a name ending in cloudacademylab. The Lab environment creates this VPC specifically for this Lab.
- Subnets: Select all available subnets.
- Security group: Select the Security Group (SG) whose name contains cloudacademylab-UsingECSLabSecurity...
- Container instance IAM role: ecsInstanceRole

# Create Cluster

When you run tasks using Amazon ECS, you place them on a cluster, which is a logical grouping of EC2 instances. This wizard will guide you through the process to cr
container instances that your tasks can be placed on, the security group for your container instances to use, and the IAM role to associate with your container instance

**Cluster name*** ecslab-cluster

Create an empty cluster

## Instance configuration

**Provisioning Model**

On-Demand Instance

With On-Demand Instances, you pay for compute capacity by the hour, with no long-term commitments or upfront payments.

Spot

Amazon EC2 Spot Instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price. Learn more

**EC2 instance type*** t2.micro

**Number of instances*** 1

**EC2 Ami Id*** amzn-ami-2017.03.d-amazon-ecs-optimized [ami-57d9cd2e]

**EBS storage (GiB)*** 22

**Key pair** None - unable to SSH

You will not be able to SSH into your EC2 instances without a key pair. You can create a new key pair in the EC2 console.

## Networking

Configure the VPC for your container instances to use. A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You can choose an existing VPC, or create a new one with this wizard.

**VPC** vpc-9f0fd9f9 (10.10.10.0/24) | cloudacademylab

Check the structure for vpc-9f0fd9f9 in the Amazon EC2 console.

**Subnets**

subnet-b47393d2 (10.10.10.0/25) - us-west-2a

subnet-9fd9e3d6 (10.10.10.128/25) - us-west-2b

Select a subnet...

**Security group** sg-bc2e9cc6 ( cloudacademylab-UsingECSLabSecurity...

Rules for sg-bc2e9cc6 in the EC2 Console.

## Container instance IAM role

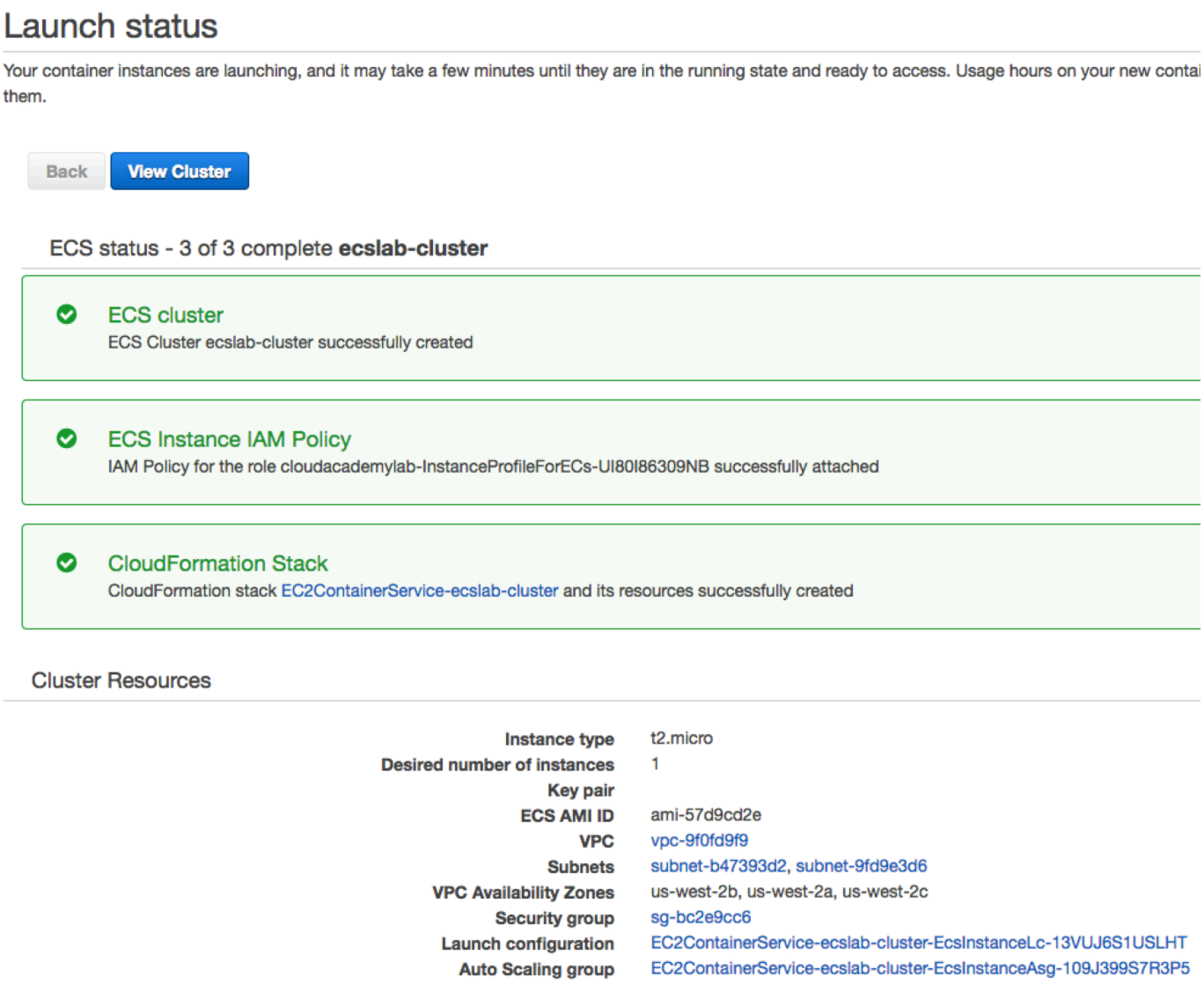
The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so container instances that run the agent require the ecsInstanceRole IAM policy and role for the service to know that the agent belongs to you. If you do not have the ecsInstanceRole already, we can create one for you.

**Container instance IAM role** ecsInstanceRole

*Required

Cancel Create

3. Observe the cluster creation process on the Launch status screen. Wait until all three steps complete and click View Cluster:




The ECS cluster creation process is a fairly complex operation that utilizes the following Amazon services:

- Elastic Compute Cloud (EC2) - The cluster is powered by EC2 instances and networking features.
- Identity and Access Management - The EC2 instances require roles with defined policies to securely interact with other services.
- CloudFormation - CloudFormation is the mechanism that deploys and manages your cluster as configured.
- Auto Scaling - Though not used in this Lab, it is important to note you can configure Auto Scaling rules and conditions for your cluster.

### Summary

In this Lab Step you created an ECS cluster. Currently, the cluster does nothing, but it is a vital logical grouping of resources for the tasks and services you will soon create. This step serves as a bridge to connect the services involved with the source code compilation and those responsible for the final application deployment. In the next Lab Step you will create task definitions for your cluster.

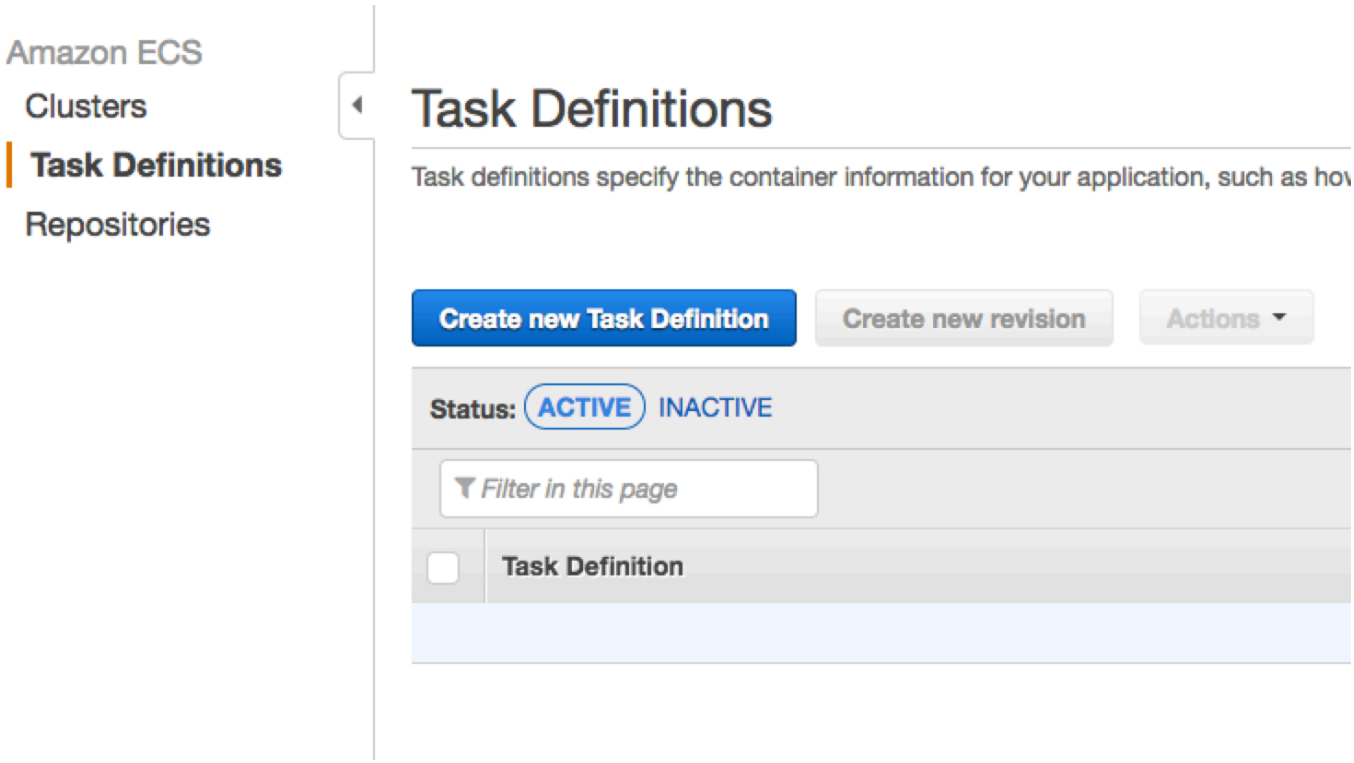
## Creating the Task Definitions

### Introduction

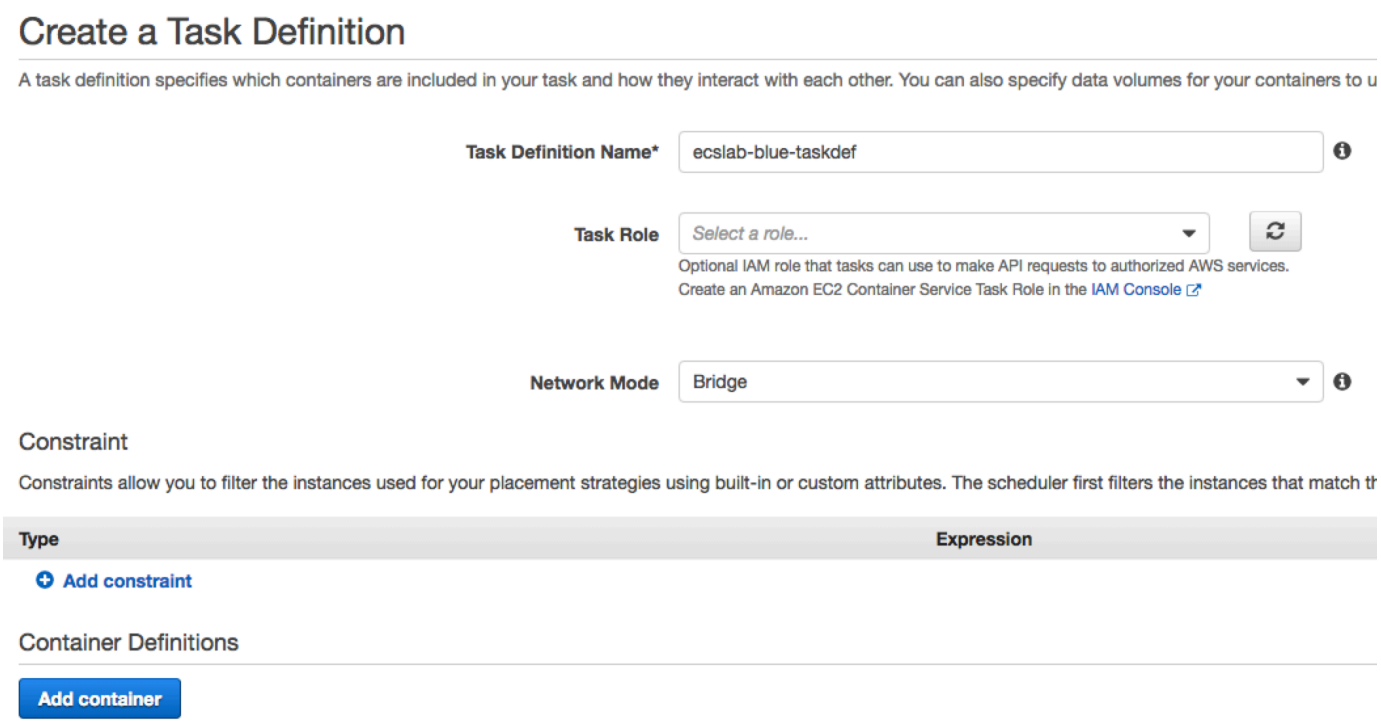
Task definitions are required to run Docker containers in ECS. Task definitions tell the services which Docker images to use for the container instances, what kind of resources to allocate, network specifics, and other details. In this Lab Step you will create two task definitions, one for your blue application and one for your green application. This is a somewhat simplified demonstration. In reality, you can define multiple containers and data volumes in a task definition, use variables, modify source files, or a number of other approaches to maintaining an application.

## Instructions

1. Select Task Definitions. Click Create new Task Definition:



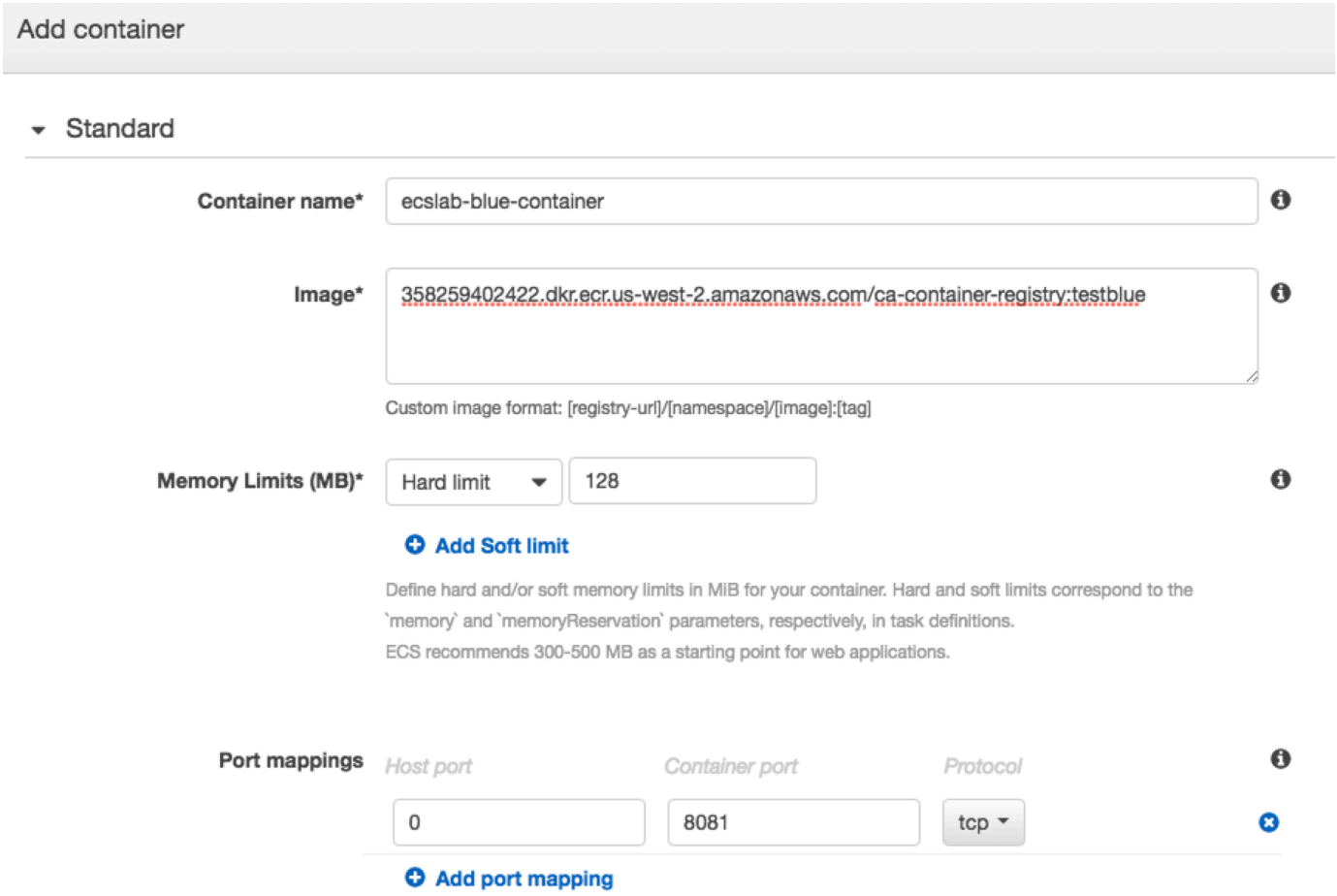

2. Enter a Task Definition Name of *ecslab-blue-taskdef*. Leave the other settings as default. Click Add container:




3. Enter the following information in the Standard section of the Add container page. Leave other settings as default.

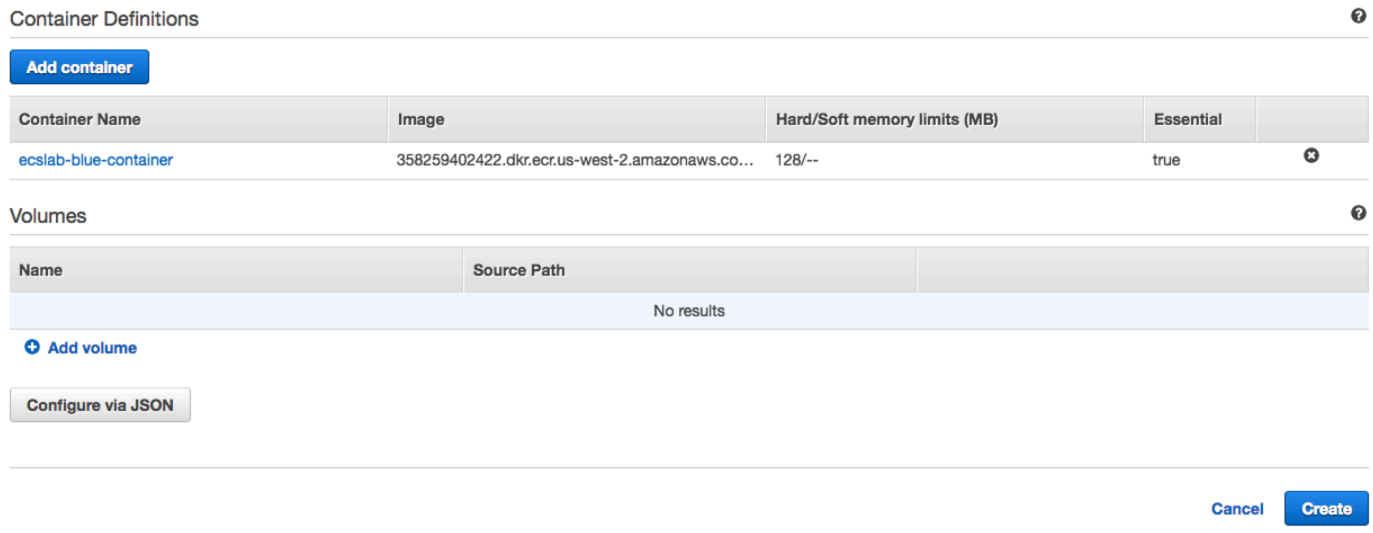
- Container name: *ecslab-blue-container*
- Image: <*YOURREPOSITORYURI*>*:testblue*, for example - 339748811106.dkr.ecr.us-west-2.amazonaws.com/ca-container-registry:testblue
- Memory Limits (MB): Hard limit *128*
- Port mappings: Host port = *0*, Container port = *8081*

Scroll down and click Add:



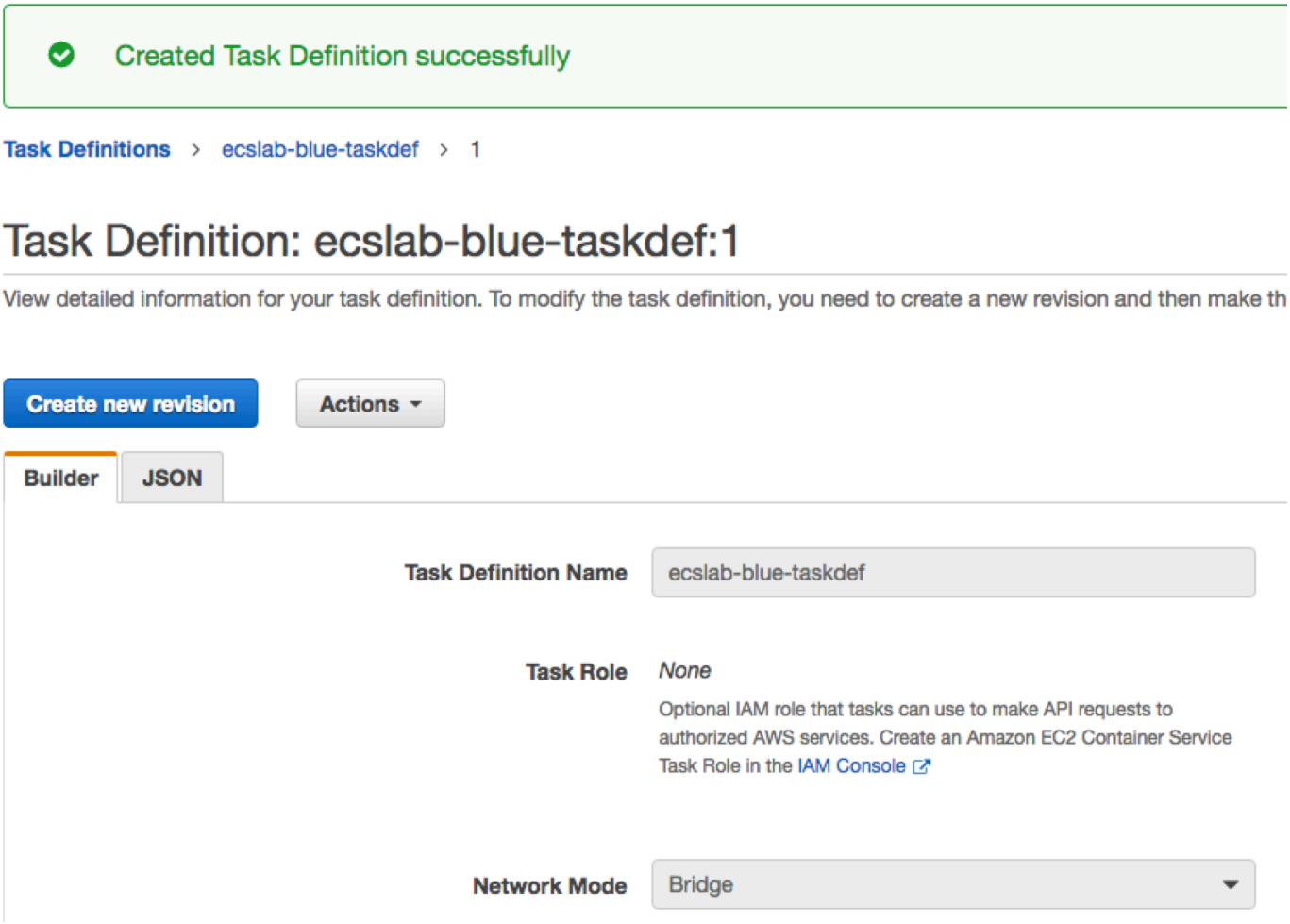

The Host port 0, Container port 8081 tells the EC2 instance hosting the containers to look for traffic on any port an application is using. Port 8081 is the traffic port exposed by the Docker container.

4. When you return to the Create a Task Definition page, note that you now have an entry in the Container Definitions section. Click Create:




Container definitions are a set of parameters passed to the Docker daemon on a container instance. In this case you have specified the name, image, and memory limit parameters.

5. Verify you see the Created Task Definition successfully message and can now view details about the ecslab-blue-taskdef task definition:




6. Return to Task Definitions and click Create new Task Definition again. As with the Docker container images, you must create another Task Definition for your green application. Enter a Task Definition Name of *ecslab-green-taskdef*. Leave the other settings as default. Click Add container.

7. Enter the following information in the Standard section of the Add container page. Leave other settings as default:

- Container name: *ecslab-green-container*
- Image: <*YOURREPOSITORYURI*>*:testgreen*, for example - 339748811106.dkr.ecr.us-west-2.amazonaws.com/ca-container-registry:testgreen
- Memory Limits (MB): Hard limit *128*
- Port mappings: Host port = *0*, Container port = *8081*

Click Add.

8. Verify you see your ecslab-green-container in the Container Definitions section. Click Create.

9. Verify you see the Created Task Definition successfully message and can now view details about the ecslab-green-taskdef task definition.

## Summary

In this Lab Step you created two task definitions for your applications. You specified information about the Docker container images, resources, and networking details to use.

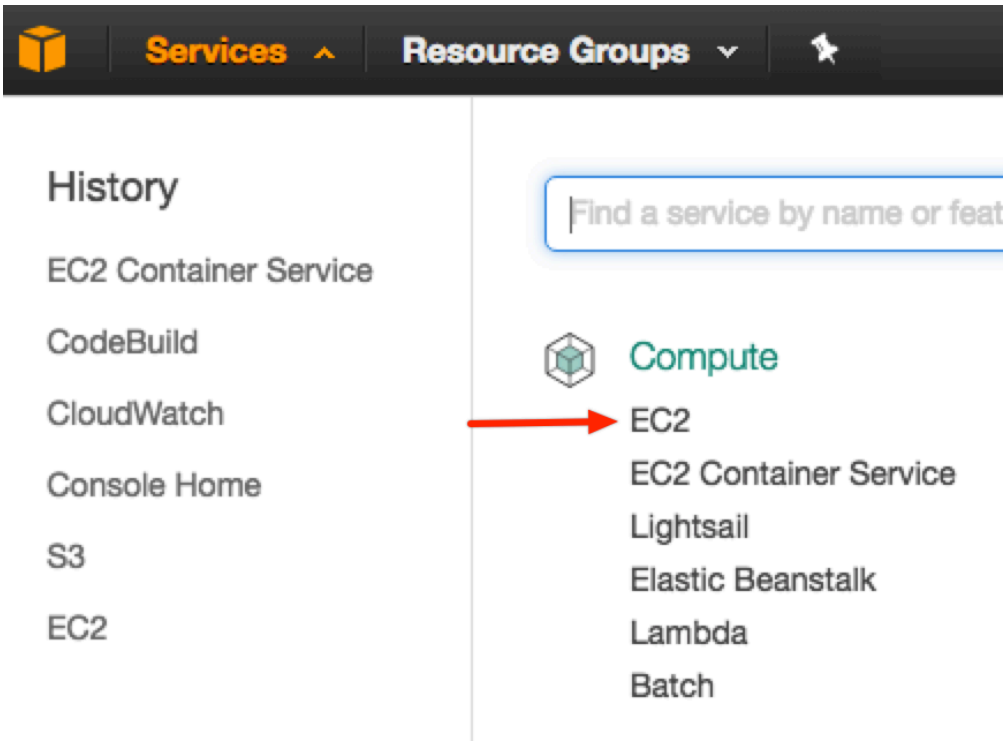
# Creating the Target Group and Load Balancer

## Introduction

Using multiple containers is a good practice to avoid a single point of failure and maintain availability. However, making the front-end access point of an application reliant on fixed references to dynamic and scalable cloud resources is not the best idea. In this Lab Step you will create a target group and load balancer to consolidate access to your resources in one place with one Domain Name Server (DNS) entry.

## Instructions

1. Navigate to Services > Compute > EC2, or search for *EC2* in the service search box:

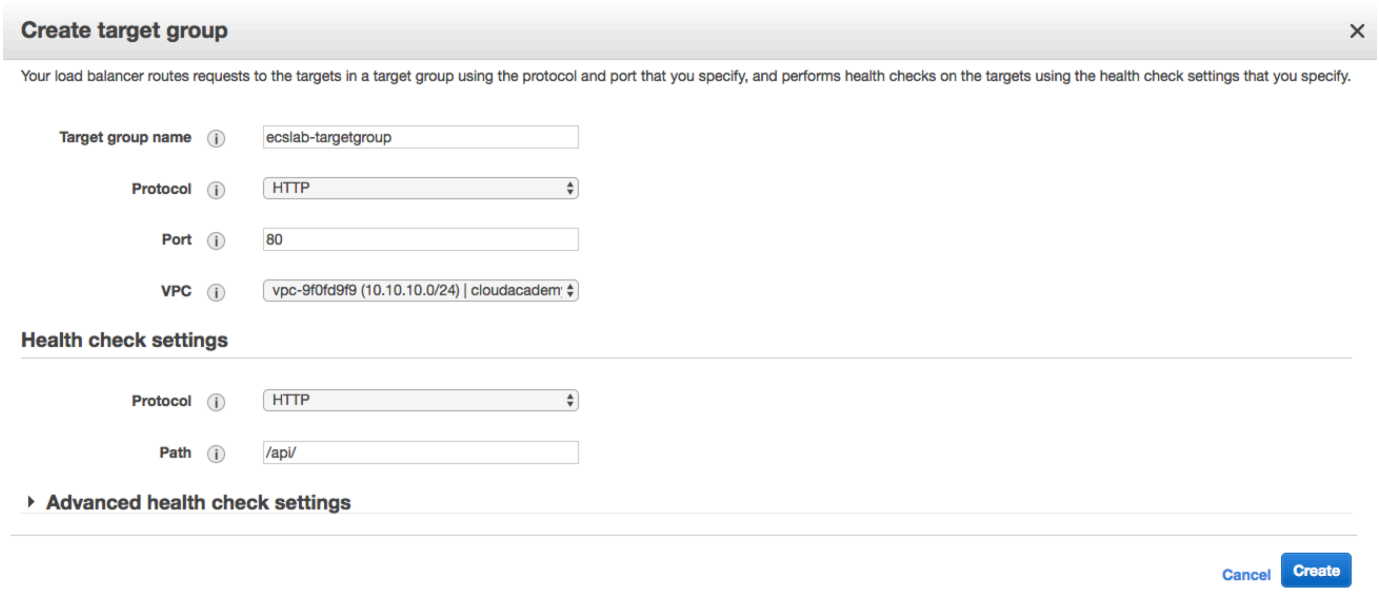



2. Select LOAD BALANCING > Target Groups from the EC2 Dashboard column.

3. Click Create target group. A target group is a group of resources, such as container or EC2 instances, registered and grouped together for reference by an Application Load Balancer (ALB).

4. Enter the following information in the Create target group window. (Channge ? to your number) Leave other settings as default. Click Create:

- Target group name: *devops?-targetgroup*
- VPC: Select the VPC with cloudacademylab in the name.
- Path: /devops?*/api/*




The /api/ path reference is not an essential part of creating every load balancer. It is a specific requirement for your Lab example application.

5. Verify you receive a Successfully created target group message and click Close.

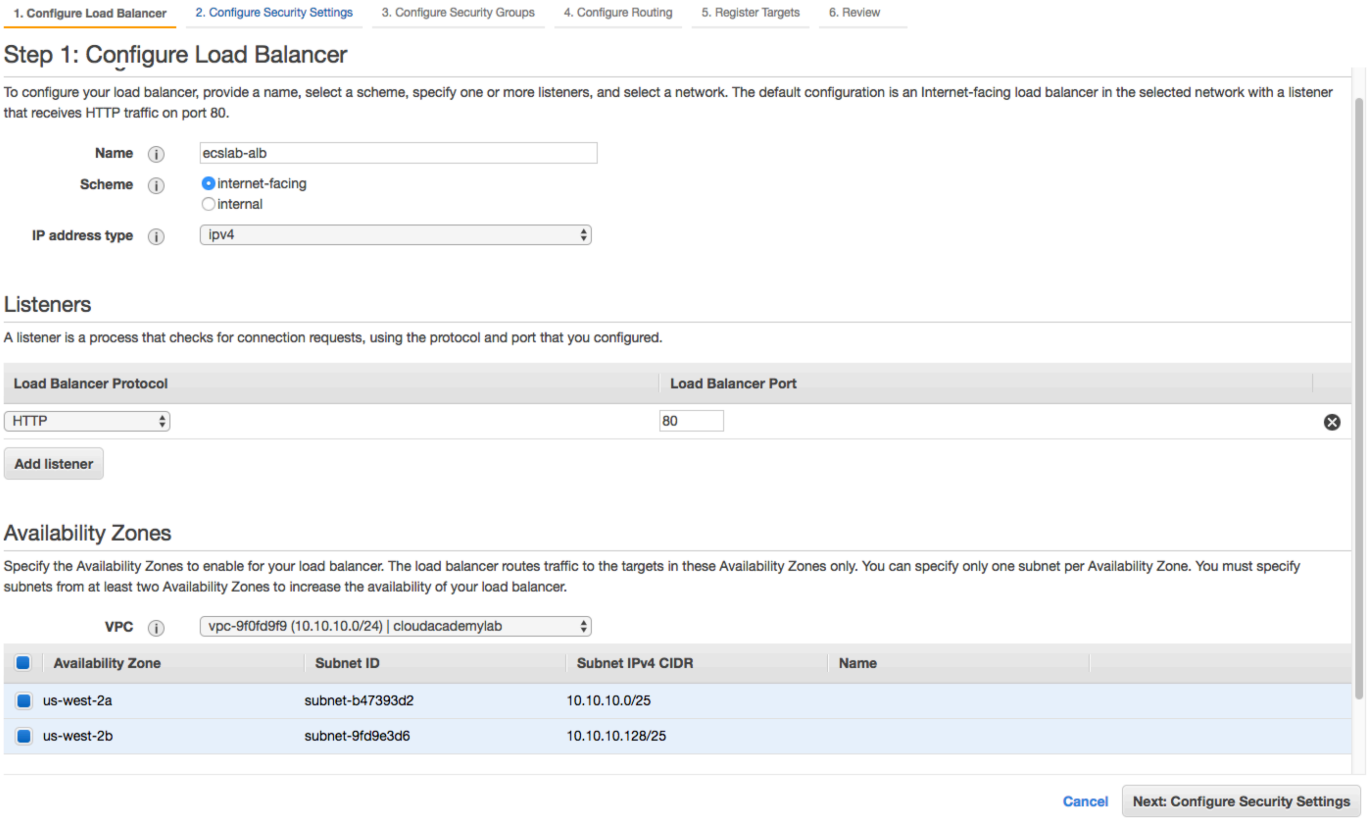
6. Select LOAD BALANCING > Load Balancers from the EC2 Dashboard column.

7. Click Create Load Balancer.

8. Leave Application Load Balancer selected and click Continue. An ALB is a type of load balancer that functions at the application layer, the seventh layer of the Open Systems Interconnection (OSI) model. It is similar to a classic Elastic Load Balancer (ELB) in that it distributes traffic among targets using a routing algorithm. However, ALBs have the ability to examine content and route traffic accordingly. The essential ALB feature for this Lab is support for containerized applications. ALBs have the ability to select an unused port when scheduling a task and registering a task with a target group using this port.

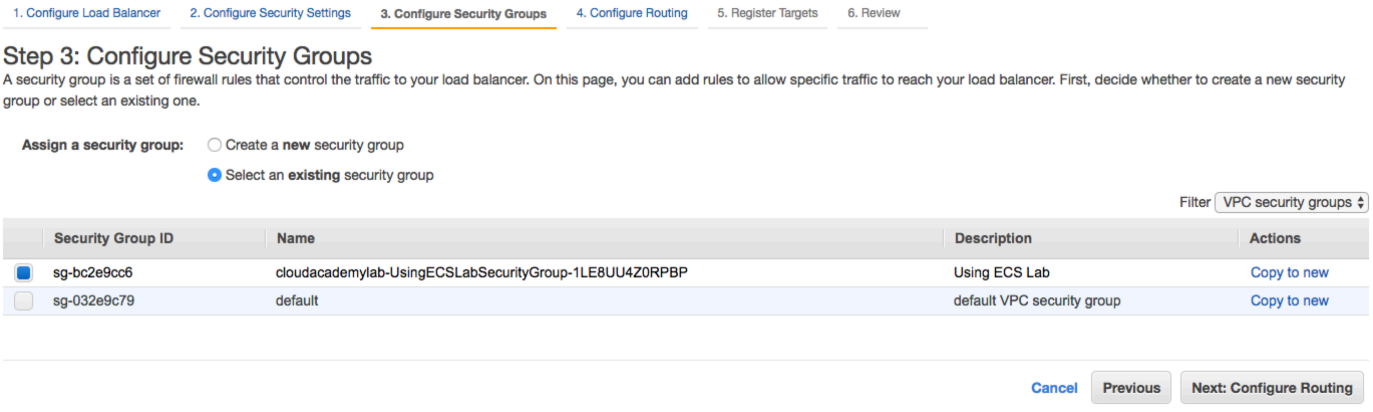
9. On the Step 1: Configure Load Balancer page, enter the following information. Leave the other settings as default. Click Next: Configure Security Settings:

- Name: ecslab-alb
- VPC: Select the VPC with cloudacademylab in the name. Select all available Availability Zones.



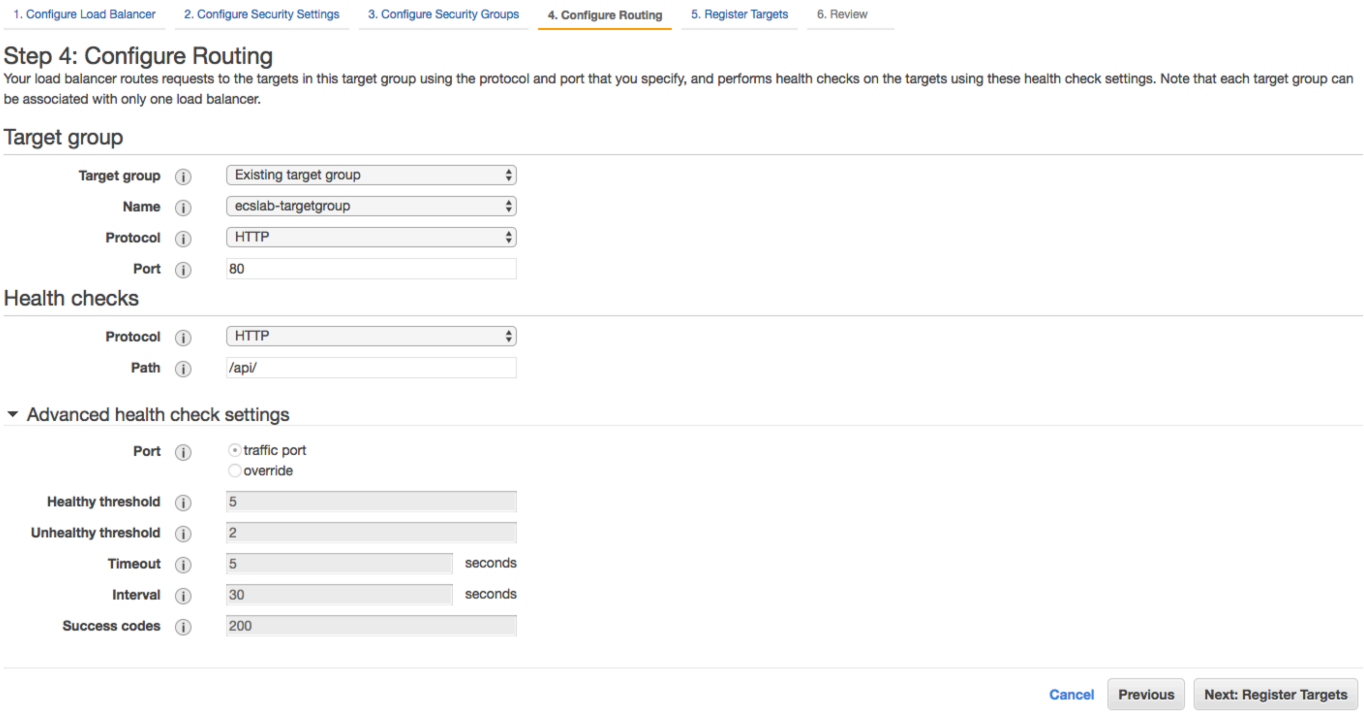

10. Ignore the warning about a secure listener. It is not important for this lab exercise. Click Next: Configure Security Groups.

11. Choose Select an existing security group and select the SG with cloudacademylab in the name. Click Next: Configure Routing:




12. Enter the following information in the Target group and Health checks sections. Leave other settings as default. Click Next: Register Targets:

- Target group: Existing target group
- Name: ecslab-targetgroup
- Path: */api/*




This Lab shows the Advanced health check settings for informational purposes. You can specify a port for routing purposes and control the health check thresholds. The thresholds determine how quickly a target registers as healthy or unhealthy.

13. There are no targets available in Step 5: Register Targets. Click Next: Review. As your ECS services launch tasks and container instances, they will dynamically register with the target group and load balancer.

14. Review the ALB information in Step 6: Review and click Create.

15. Verify you receive a Successfully created load balancer message. Click Close.

## Summary

In this Lab Step you created and configured a target group and ALB. You learned about ALBs and how their support for containerized applications makes them essential for providing a single point of access for dynamic resources.

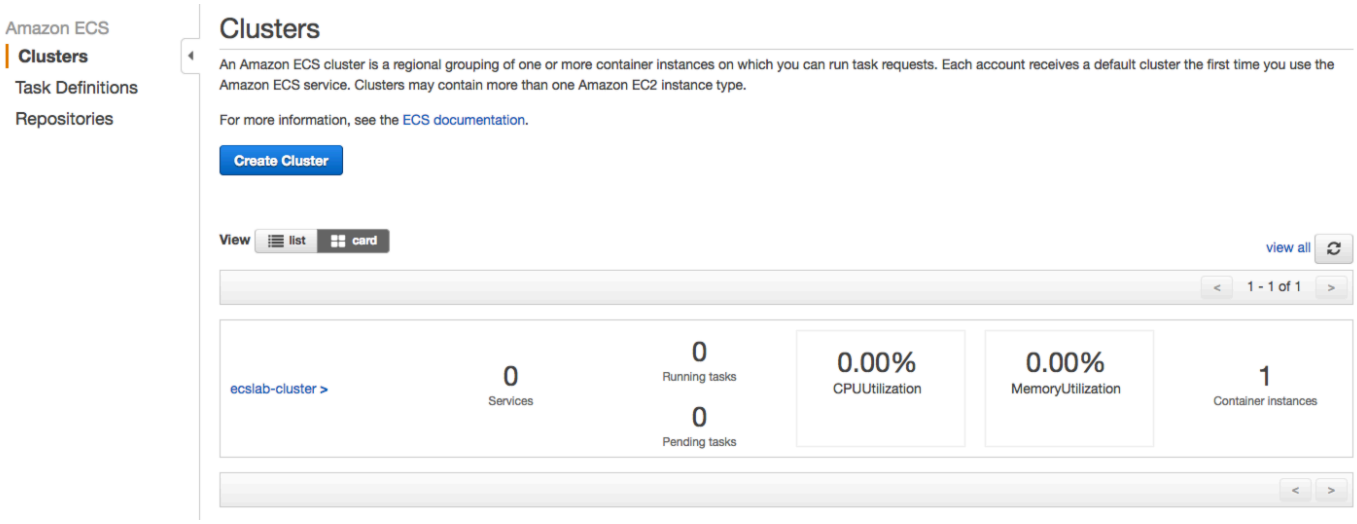
# Creating the ECS Services

## Introduction

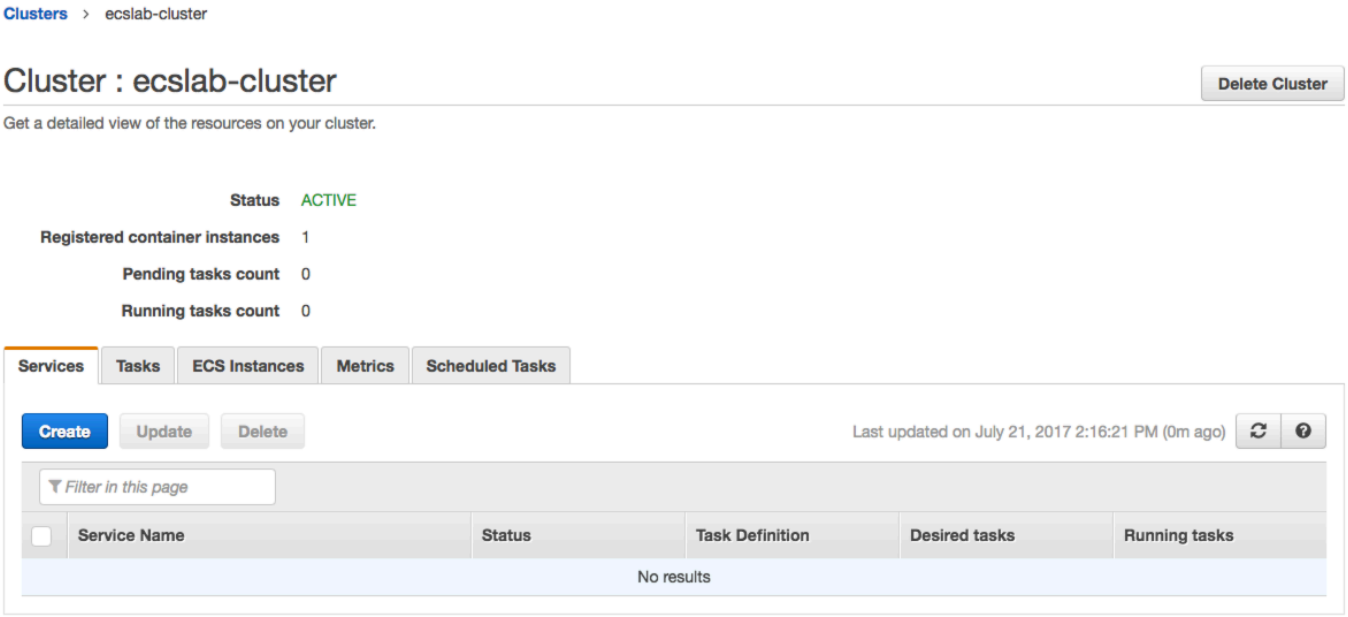
An ECS service is a mechanism that allows ECS to run and maintain a specified number of instances of a task definition. If any tasks or container instances should fail or stop, the ECS service scheduler launches another instance to replace it. This is similar to Auto Scaling in that it maintains a desired number of instances, but it does not scale instances up or down based on CloudWatch alarms or other Auto Scaling mechanisms. Services behind a load balancer provide a relatively seamless way to maintain a certain amount of resources while keeping a single application reference point. In this Lab Step you will create two services, one for your blue application and one for the green application.

## Instructions

1. Return to Services > EC2 Container Service > Clusters and click on the ecslab-cluster:

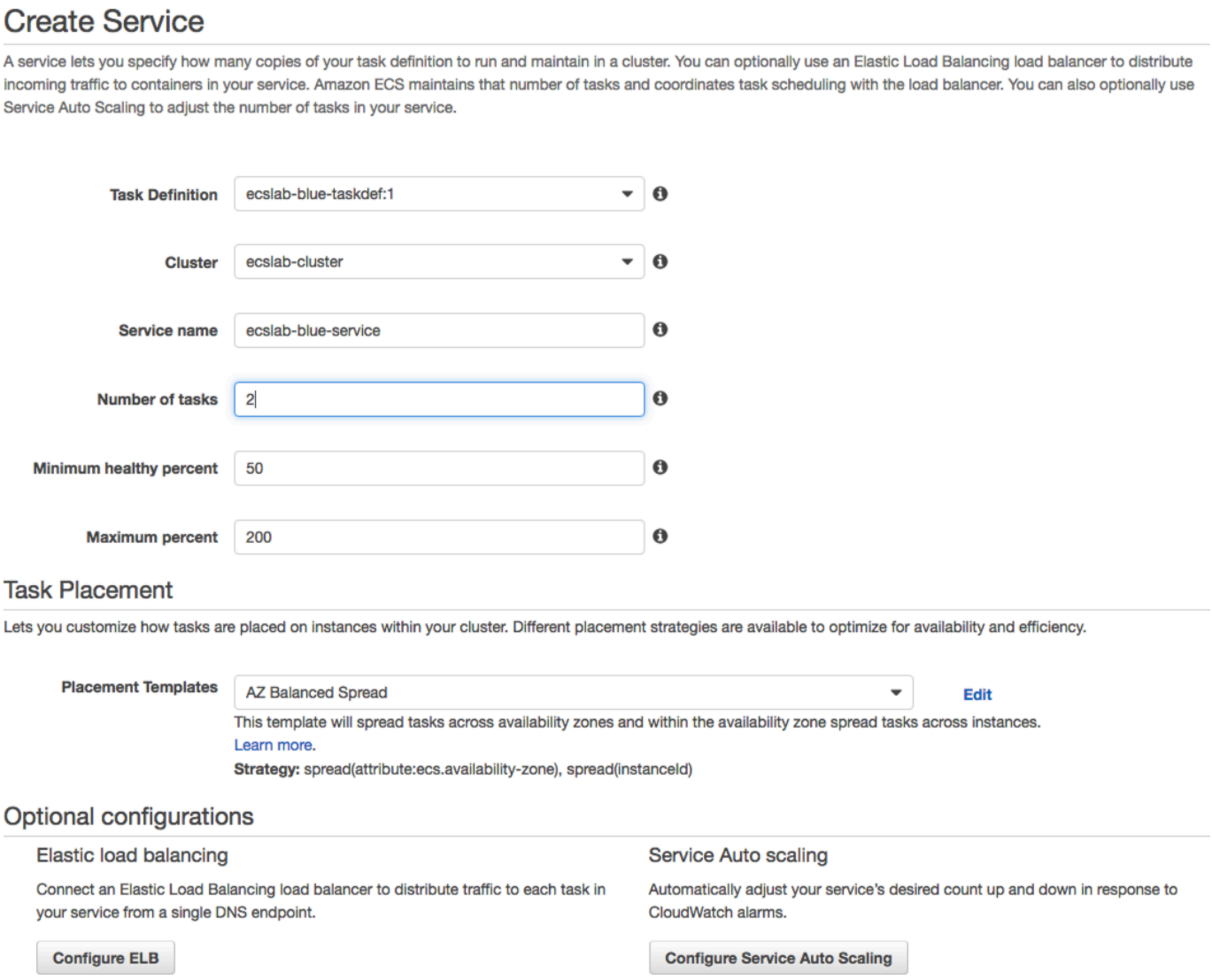



2. Click the Create button on the Services tab:




3. Enter the following information on the Create Service page. Leave the other settings as default:

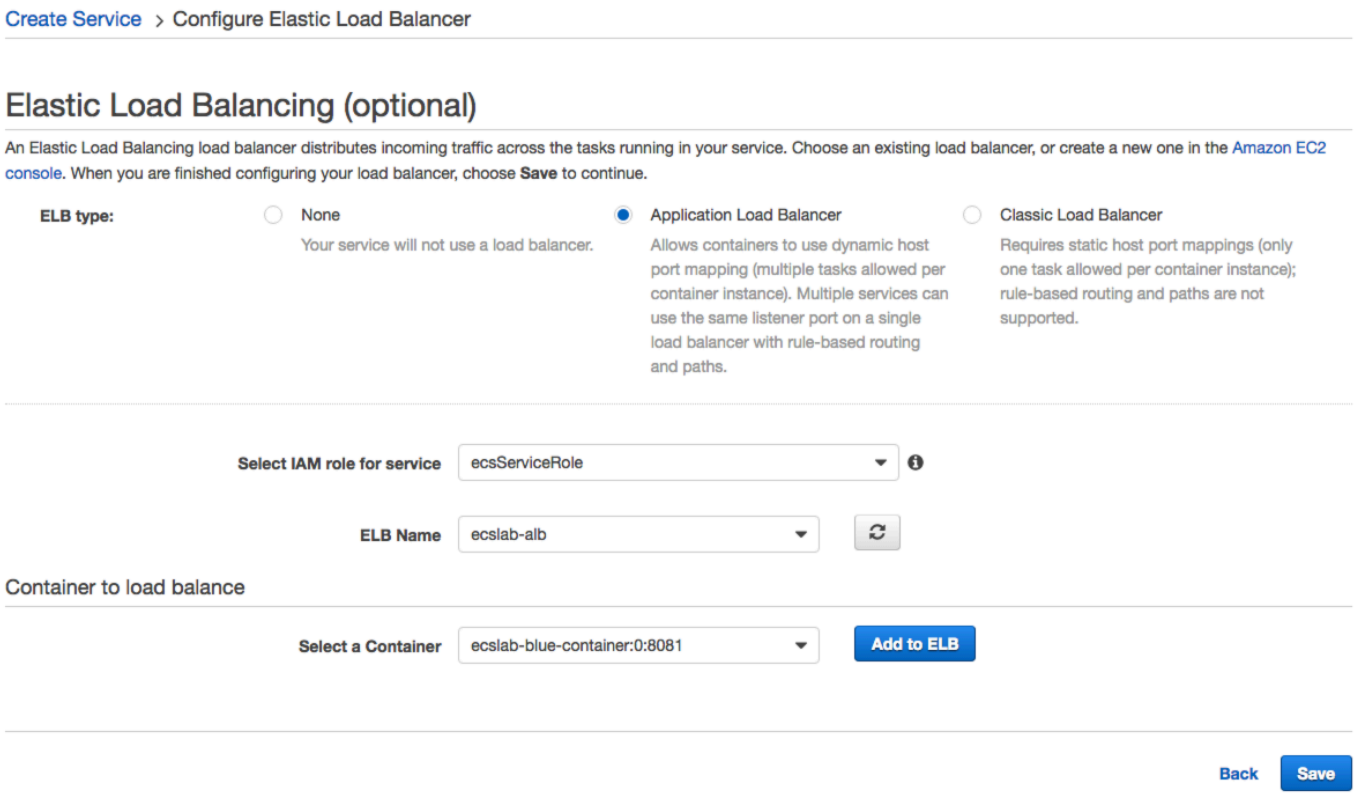
- Task Definition: ecslab-blue-taskdef:1
- Service name: *ecslab-blue-service*
- Number of tasks: *2*



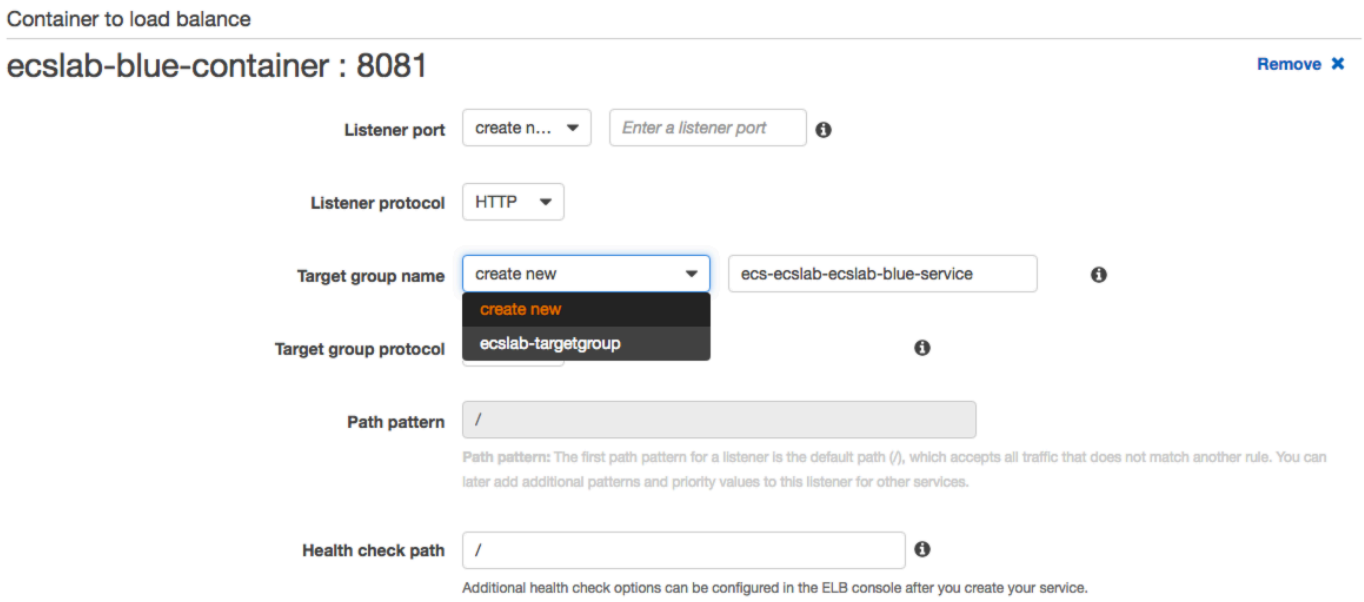

The Number of tasks determines the number of container instances the service launches and maintains. In this case you specify two tasks to launch two container instances. Even though for the sake of simplicity and to minimize costs this Lab uses a single EC2 host instance, you can alleviate the risk of having a single point of failure in the number of container instances you launch. The Minimum healthy percent and Maximum percent parameters help determine deployment strategies. The maximum parameter enables you to define the deployment batch size. With the value of 200%, the scheduler may start two new tasks before stopping the two older tasks. This can help you maintain consistent active capacity, but you may want to reduce the value to control costs if your environment is large and expensive to run. The minimum healthy parameter controls the lower limit of your service's tasks and is a way to maintain minimum capacity.

4. Click Configure ELB in the Optional configurations section. Choose the following options in the Elastic Load Balancing (optional) section. Click Add to ELB:

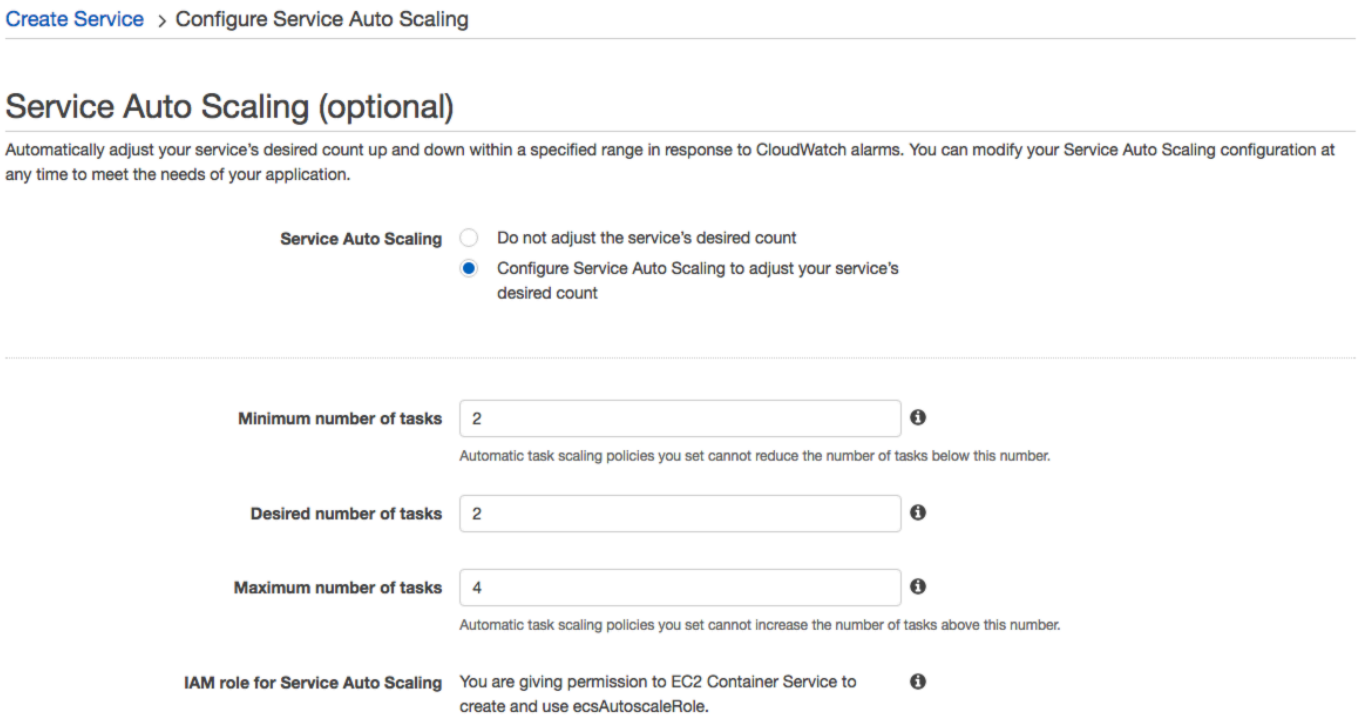
- ELB type: Application Load Balancer
- Select IAM role for service: ecsServiceRole
- ELB Name: ecslab-alb
- Container to load balance: ecslab-blue-container:0:8081



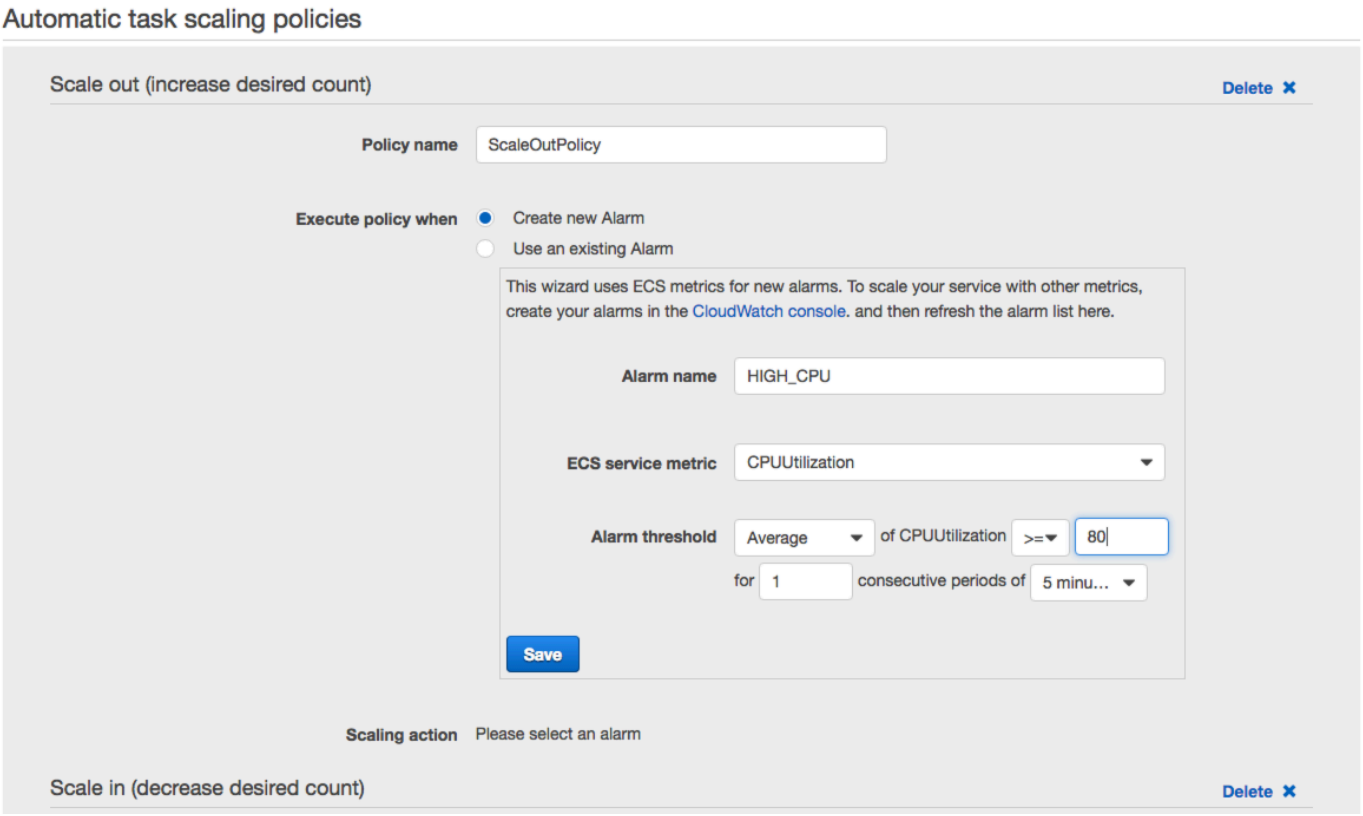

5. In the Container to load balance section, select ecslab-targetgroup from the Target group name drop-down list. This automatically populates the other settings with the target group information you entered in an earlier Lab Step. Click Save:




6. Click Configure Service Auto Scaling. Though not used in this Lab, it is useful to examine the Auto Scaling options and understand how you can use Auto Scaling to maintain a target service level in your cluster. You can specify a minimum, maximum, and desired number of tasks:



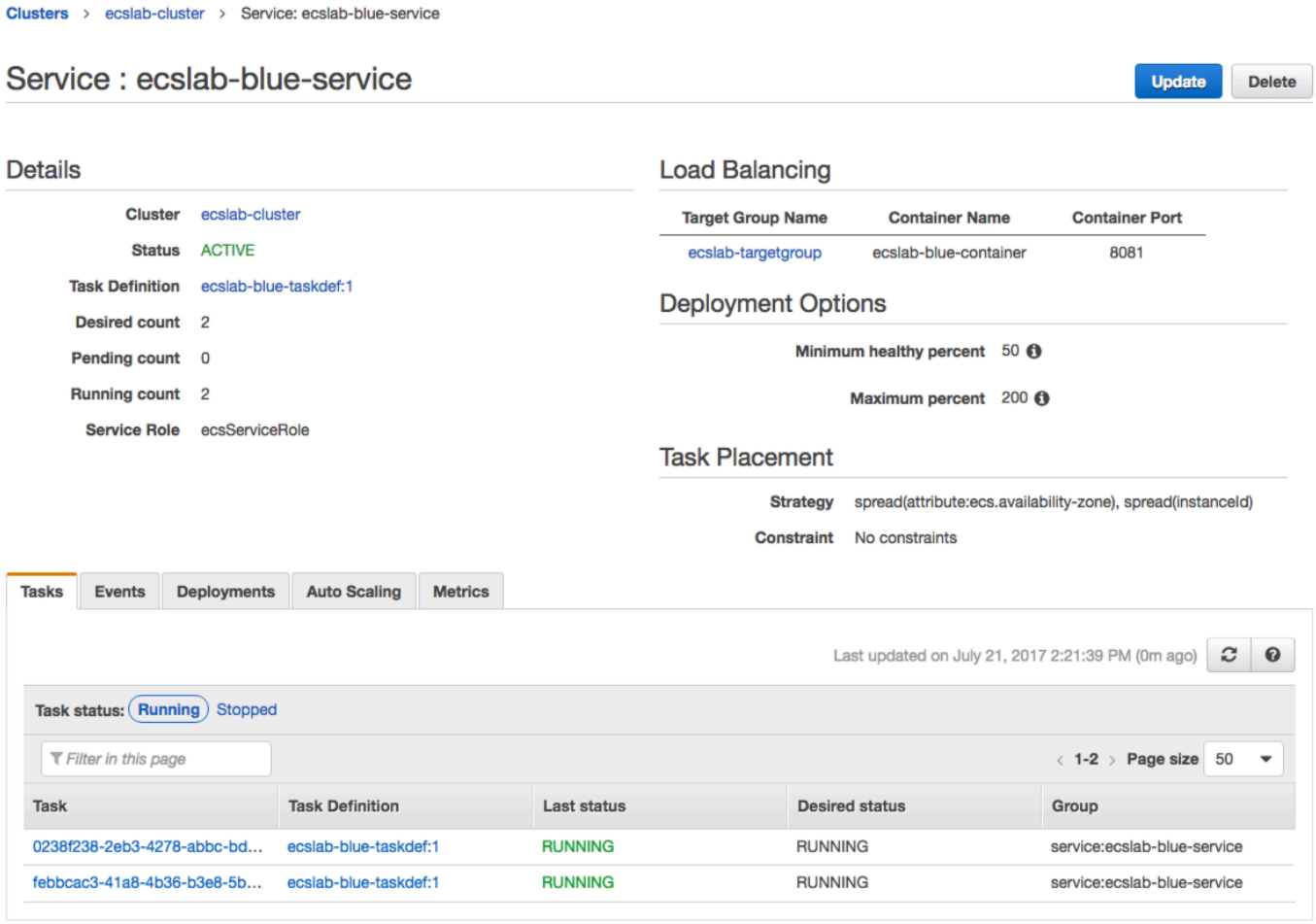

You can also set task scaling policies based on CloudWatch alarms. This allows your cluster to respond to increases or decreases in activity or traffic.



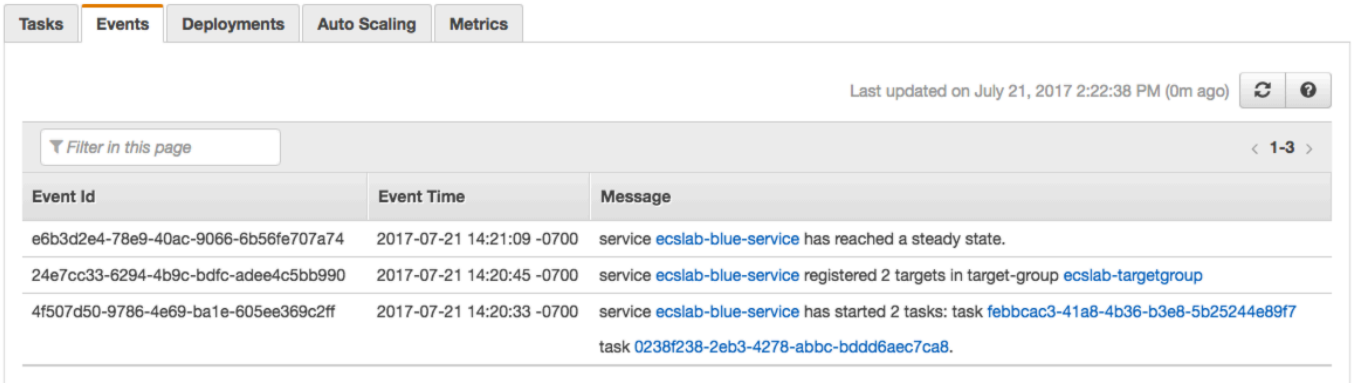

7. After exploring the scaling options, click Cancel to return to the Create Service page. Click Create Service.

8. After the Create Load Balancer and Create Service tasks complete, click View Service.

9. The ecslab-blue-service starts two tasks and begins the process of deploying them to the ECS cluster. Within approximately two minutes the service Status will change to ACTIVE and the Tasks tab will show two running tasks, as shown in the following image:




10. Click the Events tab to view the service events:




Note how the service registered two targets with the application load balancer target group. Earlier you created a load balancer but did not register any target instances. Instead, the service instantiates container instances based on the number of tasks you specify. You added the load balancer and target group to the service, so any new container instances the service creates automatically register as a target in the target group.

11. Return to Clusters > ecslab-cluster. Click the Create button on the Services tab. You are going to create the green service now, but you will not give it any tasks at this moment.

12. Enter the following information on the Create Service page. Leave the other settings as default.

- Task Definition: ecslab-green-taskdef:1
- Service name: *ecslab-green-service*
- Number of tasks: *0*

*Note*: You can create services with zero tasks. They will not launch any container instances upon creation. This allows you to prepare for future operations before your deployment is fully ready.

13. Click Configure ELB in the Optional configurations section. Choose the following options in the Elastic Load Balancing (optional) section. Click Add to ELB:

- ELB type: Application Load Balancer
- Select IAM role for service: ecsServiceRole
- ELB Name: ecslab-alb
- Container to load balance: ecslab-green-container:0:8081

14. In the Container to load balance section, select ecslab-targetgroup from the Target group name drop-down list. Click Save.

15. Click Create Service. After the Create Load Balancer and Create Service tasks complete, click View Service.

### Summary

In this Lab Step you created two services for your blue and green applications. You learned how these services control the desired capacity and how you can configure them to scale, if necessary. You also learned more about how they interact with the target group and load balancer you created earlier. You started two tasks for the blue application upon service creation. These tasks launched and registered two container instances.

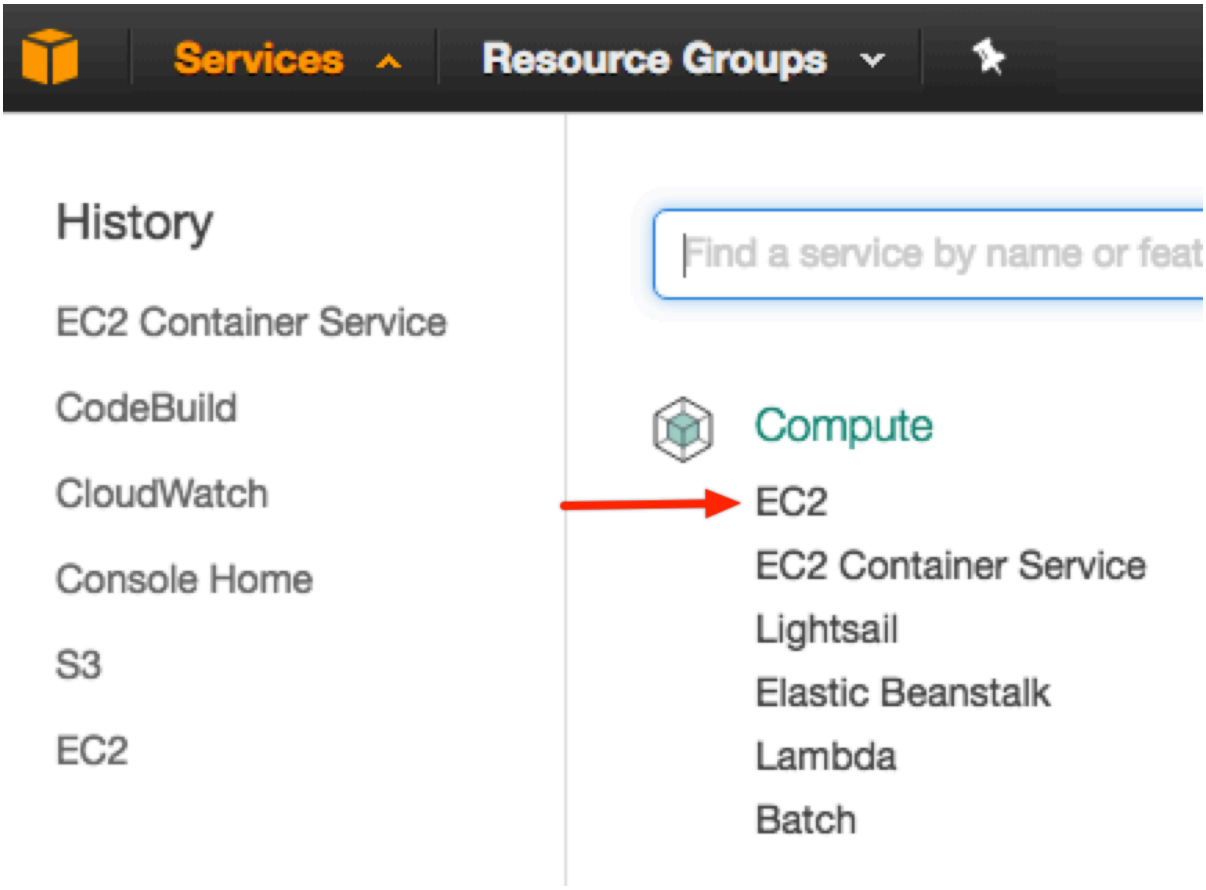
# Viewing Instances and Application Message

### Introduction

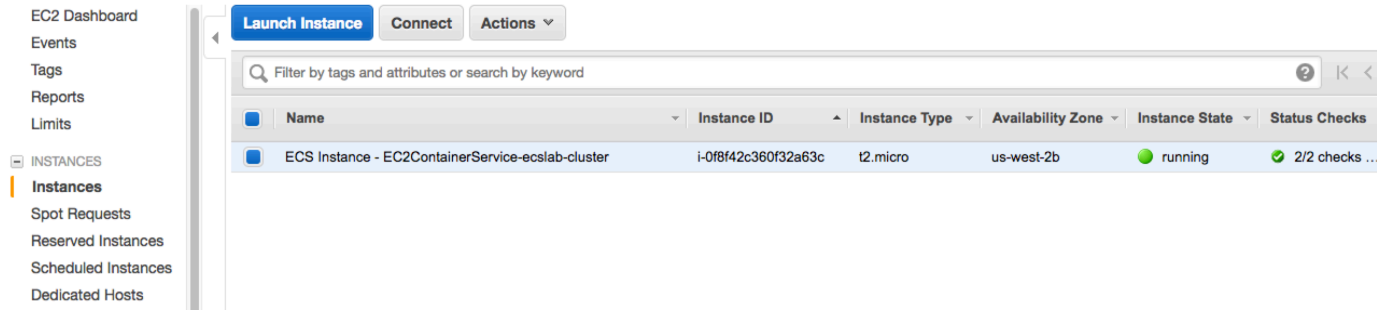
Now that you have put all of the pieces in place to build and store Docker images, dynamically register container instances, and maintain a target capacity, it is time to learn more about the interdependent service actions and see some results. In this Lab Step you will view the resources launched by ECS and access the blue application launched when you created the blue service.

### Instructions

1. Navigate to Services > Compute > EC2:

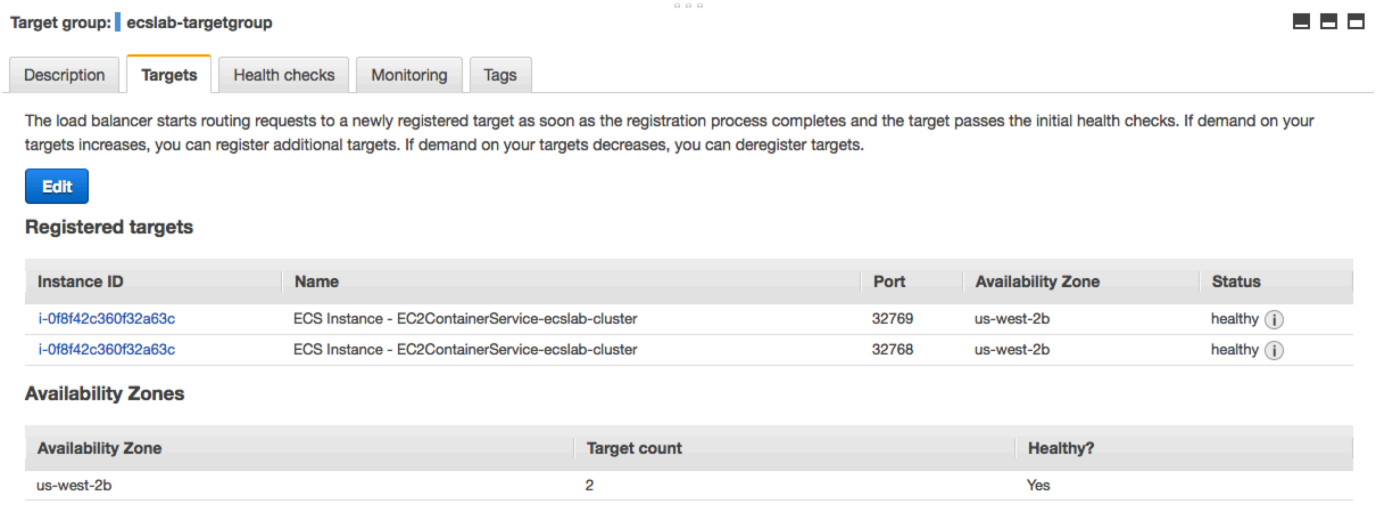



2. Select INSTANCES > Instances from the EC2 Dashboard column. The ECS blue service automatically launched an EC2 instance to host the ECS container instances running your tasks:




3. Select LOAD BALANCING > Target Groups from the EC2 Dashboard column.

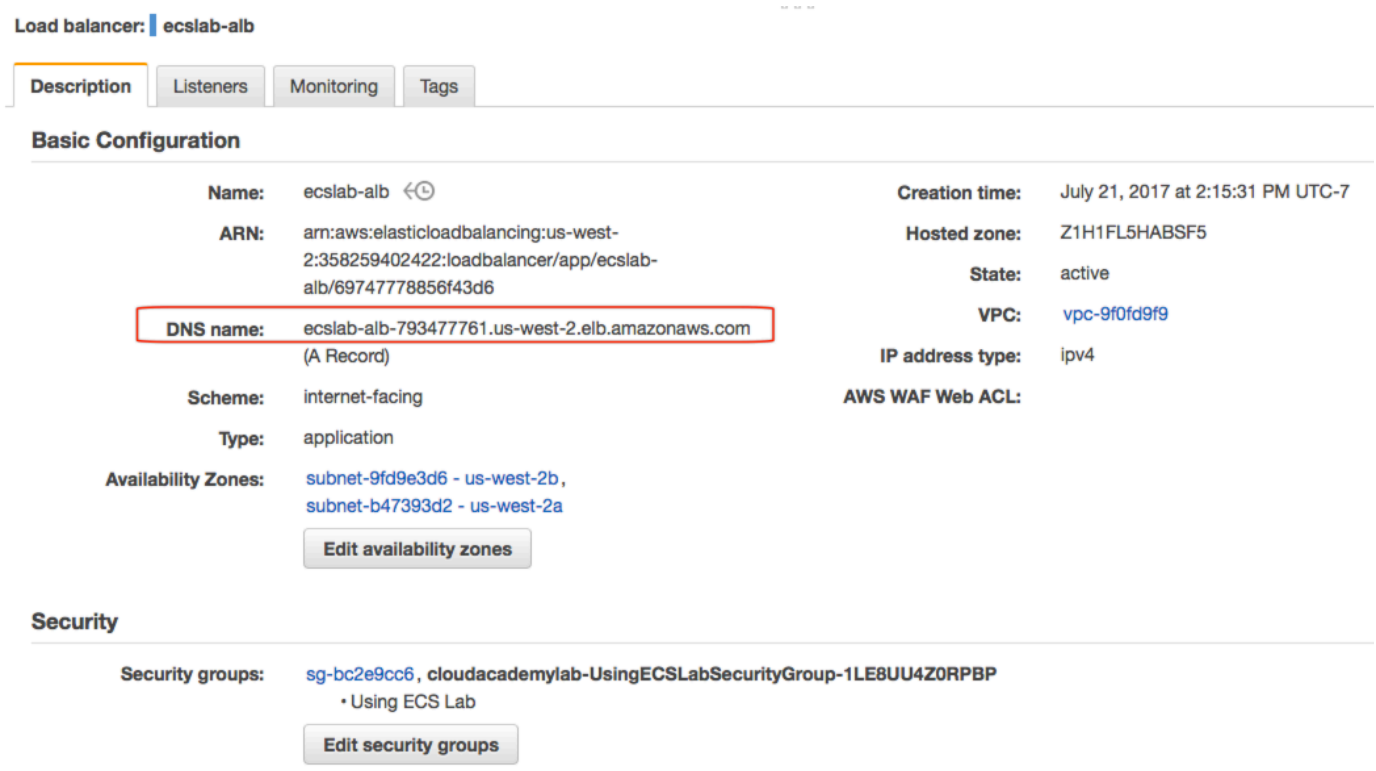
4. Click the Targets tab and view the Registered targets section. Notice that you have two healthy targets despite having only one running EC2 instance. ECS deployed two container instances on the EC2 host instance. Each container instance dynamically registered with the target group on an assigned port in the ephemeral port range:




5. Select LOAD BALANCING > Load Balancers from the EC2 Dashboard column.

6. Copy the DNS name from the Description tab and paste it into a browser window or tab. Append */api/* to the DNS value to form a Uniform Resource Locator (URL) that resembles this example:

*http://ecslab-alb-793477761.us-west-2.elb.amazonaws.com/api/*




The load balancer DNS name provides a single point of access for your users/application for all instances behind the load balancer, even if those instances fail or are replaced.

7. Your browser will display the following message, though the format or appearance may vary slightly depending on the browser you use. This is a simple JavaScript Object Notation (JSON) message delivered by the application running on the container instances:

```
{
    "message": "Hello - I'm BLUE"
}
```


### Summary

In this Lab Step you looked at the resources created by your ECS services and tasks. You also looked at the end result of your application, a JSON message on a website.

## Updating the Services and Application Deployments

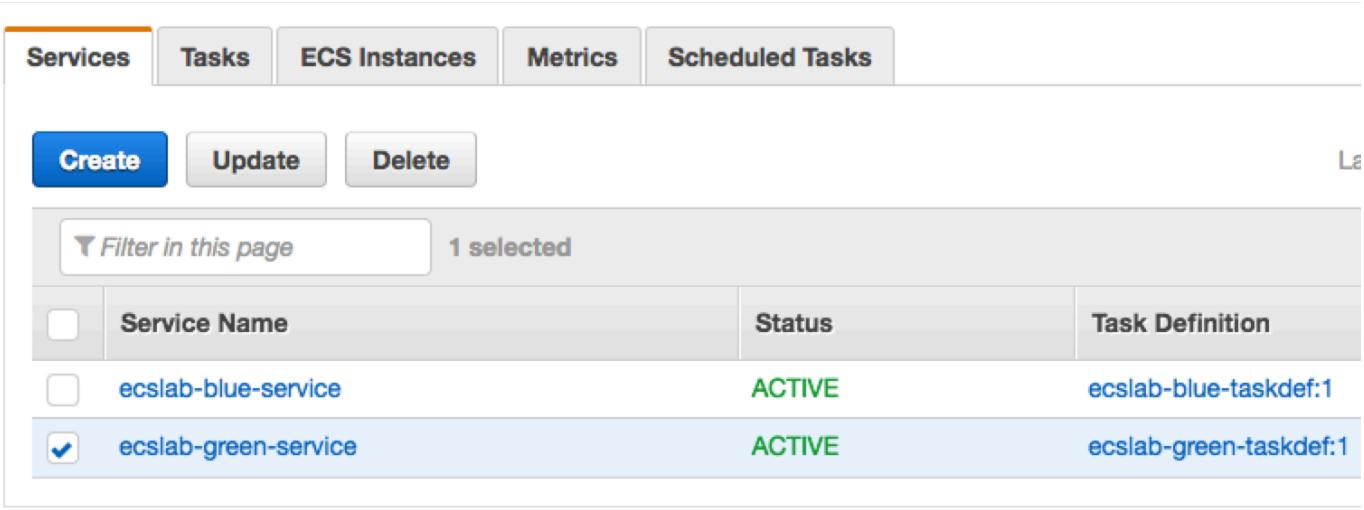
### Introduction

Now that you know how the services interact and how to launch tasks into your ECS cluster, it is time to switch from one application to the next. A blue-green deployment is a technique that uses two identical production environments to reduce downtime. There are different cutover strategies, but generally only one environment should be serving production traffic. In this Lab Step you will launch both applications with all of the container instances you need, including a brief overlap period, to demonstrate the load balancer's round-robin distribution and container resource efficiency.

### Instructions

1. Return to Services > EC2 Container Service > Clusters and click on the ecslab-cluster.

2. Select the ecslab-green-service on the Services tab and click Update:

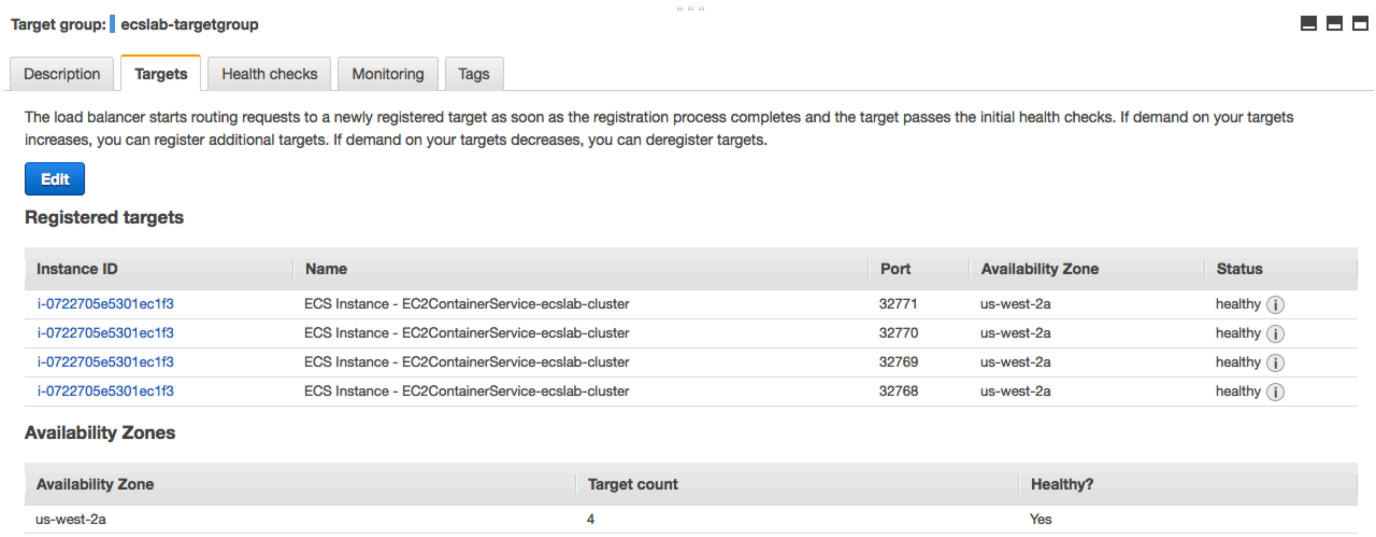



3. Change the Number of tasks value from *0* to *2*. Click Update Service.

4. Wait for the Service updated successfully message and click View Service.

5. Wait approximately two minutes for the new green tasks to launch. You can get service information on the service Events tab or wait for the Ru nning count to reach 2.

6. Return to Services > Compute > EC2 > LOAD BALANCING > Target Groups and verify you have four healthy targets representing the four running container instances:




The reduced resource requirements for a Docker container allow you to run multiple container instances on one EC2 instance. This works well for applications that require little resources, such as this simple message application.

7. Refresh your browser tab with the "Hello - I'm BLUE" message. It may take more than one refresh, but you will see the message alternate between "Hello - I'm BLUE" and "Hello - I'm GREEN" as you refresh the page and cycle through the container instances. Right now you have both applications running in their own pair of container instances.

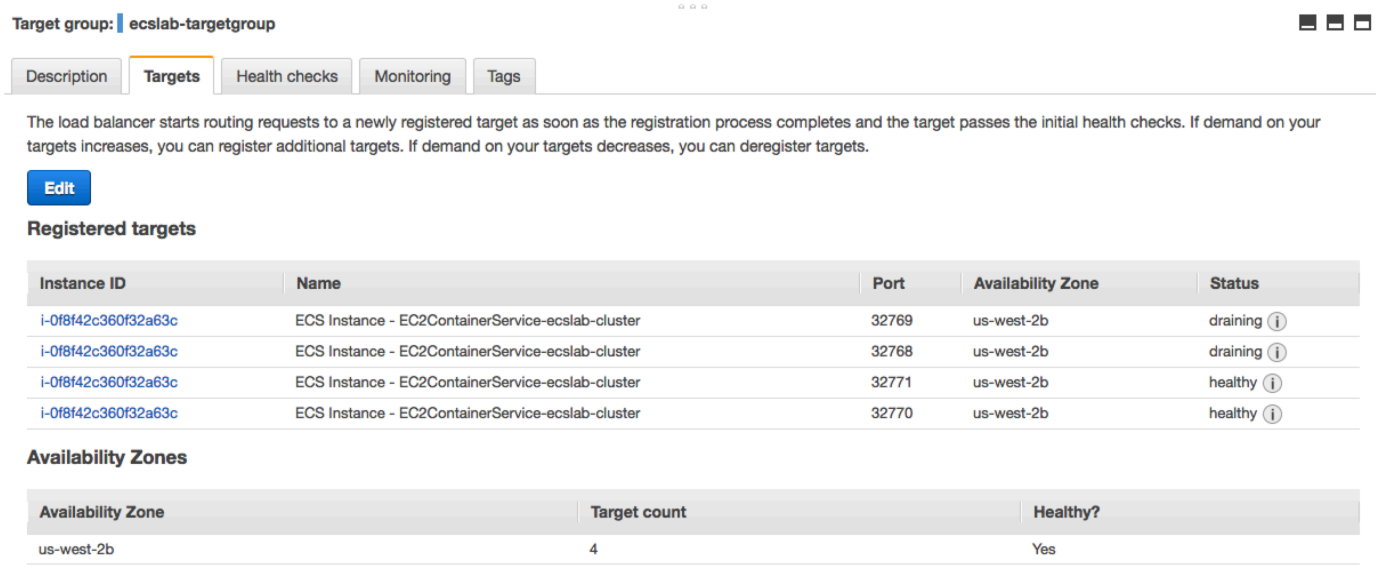
8. Return to Services > EC2 Container Service > Clusters and click on the ecslab-cluster.

9. Select the ecslab-blue-service on the Services tab and click Update.

10. Change the Number of tasks value from *2* to *0*. Click Update Service.

11. Wait for the Service updated successfully message and click View Service.

12. Return to Services > Compute > EC2 > LOAD BALANCING > Target Groups. You can watch two of the container instance targets draining as ECS stops the tasks:




13. Refresh the browser until the only message visible is "Hello - I'm GREEN". You have successfully transitioned from one application to another with no downtime or launching any new EC2 instances.

*Note*: There are different approaches to transitioning between one application or version to another. In this Lab Step you bring up both applications and leave them running simultaneously and externally accessible for a short time. Another approach might be to schedule a maintenance window and set the number of running tasks for the blue application to 0. Wait for the blue targets to successfully drain. Then bring up the green application if you need to avoid overlap. Total downtime remains only minutes.

## Summary

In this Lab Step you launched the green application alongside the blue application in the ECS cluster. You watched as the round-robin load balancer distribution served messages from both applications running on one EC2 host instance. You then set the number of tasks for the blue application to 0 and watched as ECS drained the tasks and stopped serving the application's message.

# Deleting the ECS Laboratory Resources

## Introduction

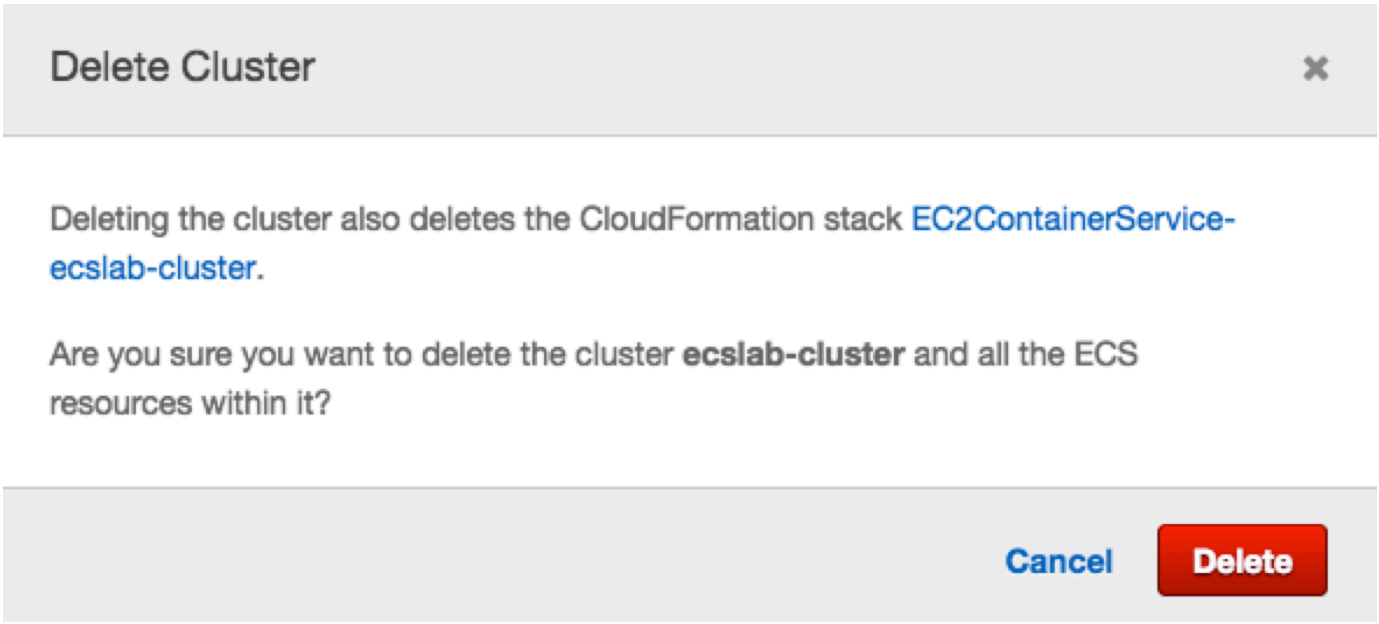
One of the great features of ECS is it helps make your application robust and available. If you manually terminate the EC2 instance running your ECS tasks, ECS starts another instance and relaunches the tasks. If you try to delete a service with running tasks, ECS will not allow it. ECS does an admirable job safeguarding running services and tasks inside a cluster. To stop an application you must either bring services down one at a time or delete the entire cluster.

## Instructions

1. Return to Services > EC2 Container Service > Clusters and click on the ecslab-cluster.

2. Click the Delete Cluster button in the upper-right.

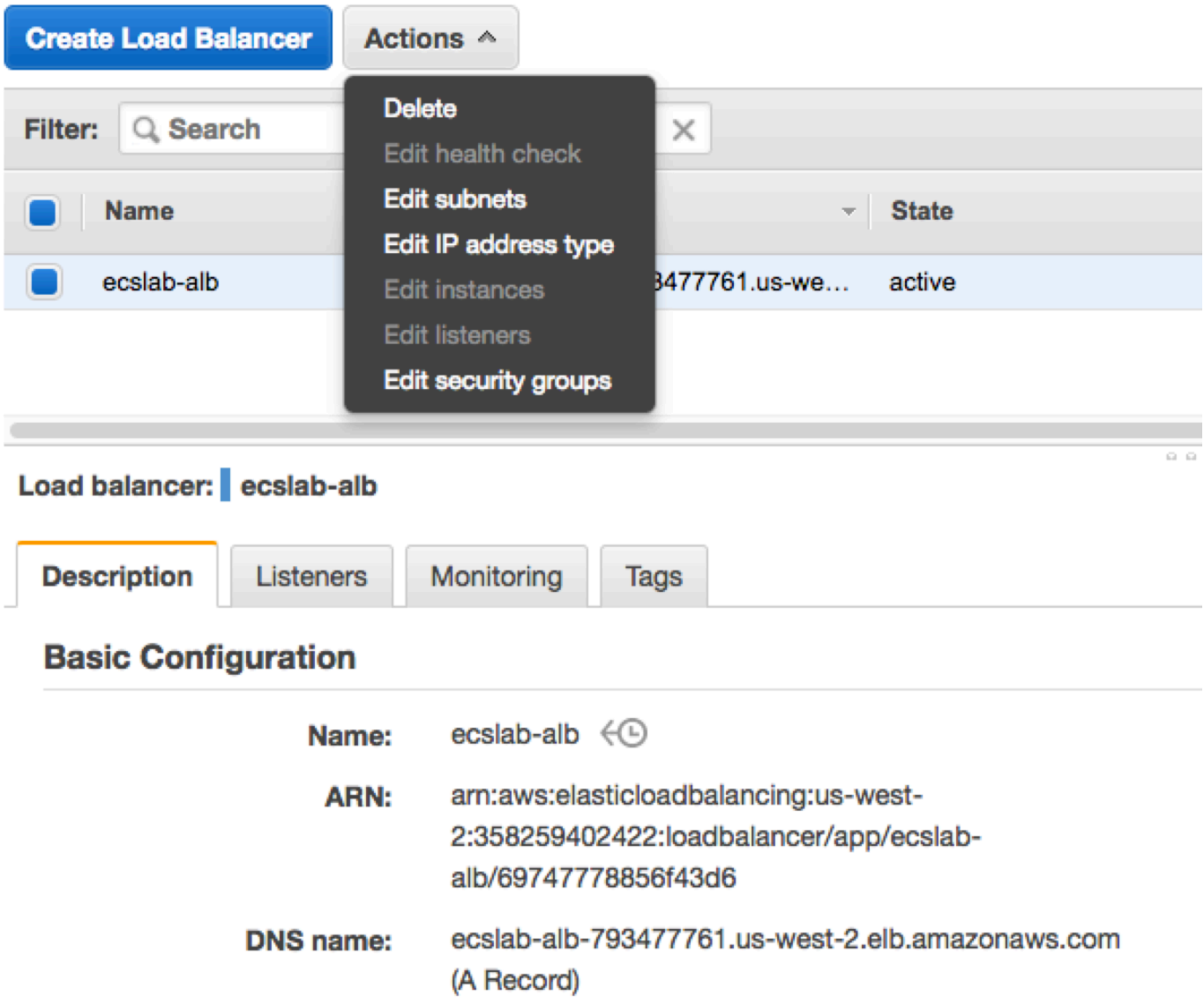
3. Note the warning about deleting a CloudFormation stack. CloudFormation powers much of the ECS functionality and allows you to delete all of the resources in a cluster in an orderly fashion. Click Delete:




4. Wait for the Delete Cluster processes to complete. After approximately one minute you will see the Deleted cluster ecslab-cluster successfully message.

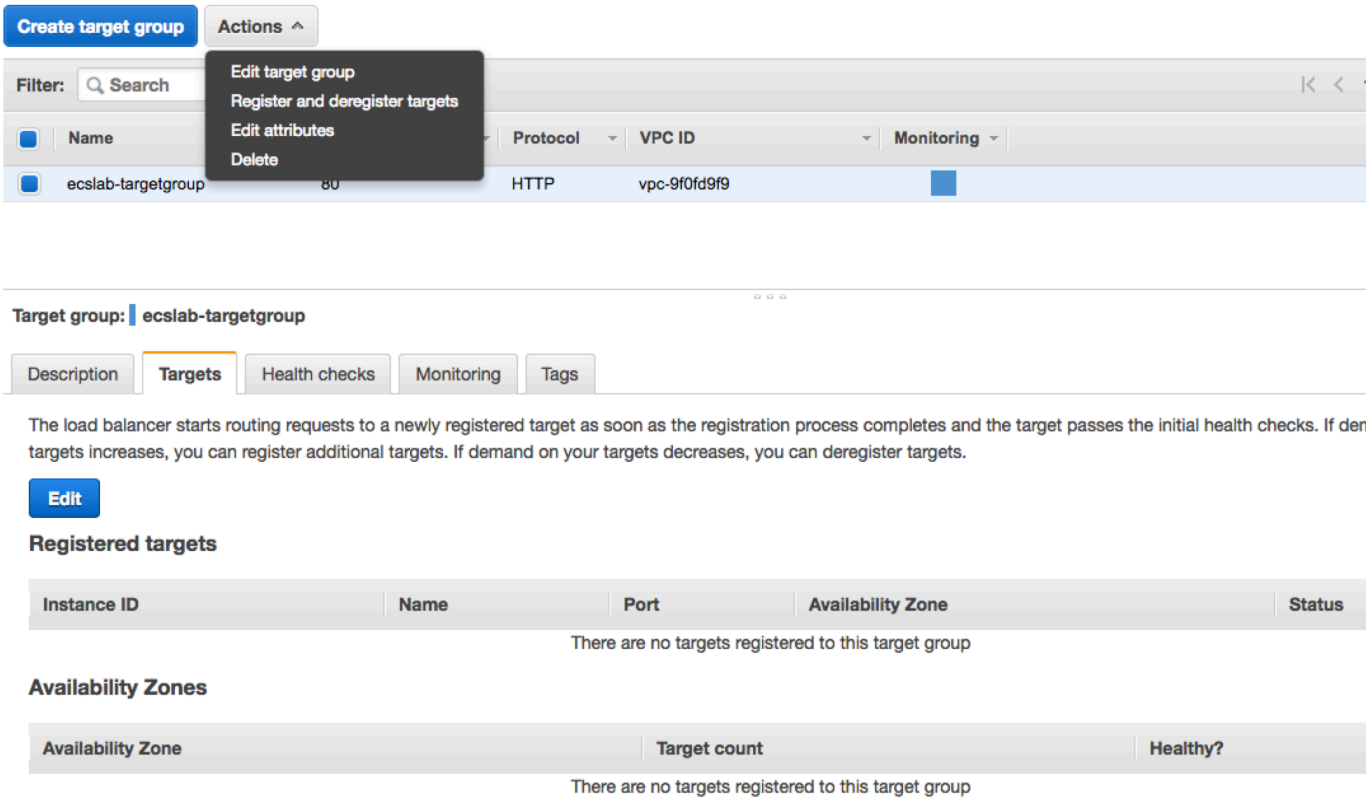
5. Click Repositories in the Amazon ECS column. Select the ca-container-registry repository and click Delete repository. Click Delete at the Delete repository warning indicating your images will also be deleted.

6. Return to Services > Compute > EC2 > LOAD BALANCING > Load Balancers from the EC2 Dashboard column. Select ecslab-alb and select A ctions > Delete:




7. Click Yes, Delete at the Delete Load Balancer message.

8. Select LOAD BALANCING > Target Groups from the EC2 Dashboard column. Select ecslab-targetgroup and select Actions > Delete:




9. Click Yes at the Delete target group message.

## Summary

In this Lab Step you deleted the key components of your ECS application including the cluster, image repository, target group, and load balancer. Other, ancillary resources remain such as the S3 source files, VPC, and CodeBuild projects. The Cloud Academy lab environment will delete these resources for you. In a real environment you may want to keep your source files and network infrastructure in case you want to update or redeploy an application later. These deletion steps stop the most expensive items from incurring additional charges. For many resource types, Amazon will charge a full hour if the resource is used during even one minute of that hour, so it is a good idea to delete resources when you are done using them.