Deploying a Node-Express Container Application on Amazon ECS

Table of Contents

Introduction

Amazon Elastic Container Service (Amazon ECS) is a fully managed service for deploying, managing, and scaling containerized applications. In this article, we will walk through the steps to deploy a simple Express web application on Amazon ECS. Express is a Node.js web application framework that provides features for building web and mobile applications.

Tasks

In this article, we will be using a simple calculator node-express application available here. First, we will create a docker image and store the image in a secure and reliable container registry. We will be utilizing Amazon Elastic Container Registry (Amazon ECR) for this purpose. The following are the steps that we will cover in this article:

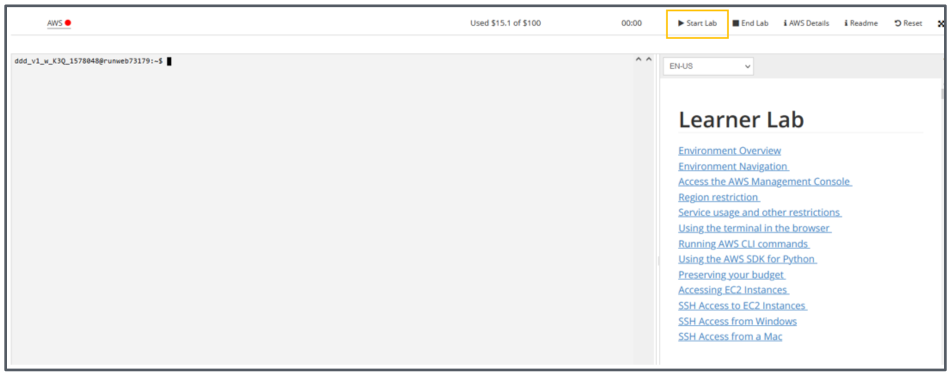
- Create an Amazon ECR repository
- Download the application code, build docker images, and push the images to ECR repository.
- Create an Application Load Balancer.
- Create an Amazon ECS Cluster.
- Create a task definition.
- Create an ECS service.

- Test the application.

Prerequisites

Before continuing with the tasks listed above, you will need:

- **Access to an AWS Academy Learner (LL) course**: From the Student View of the LL, click the Start Lab button as shown in the screenshot below:

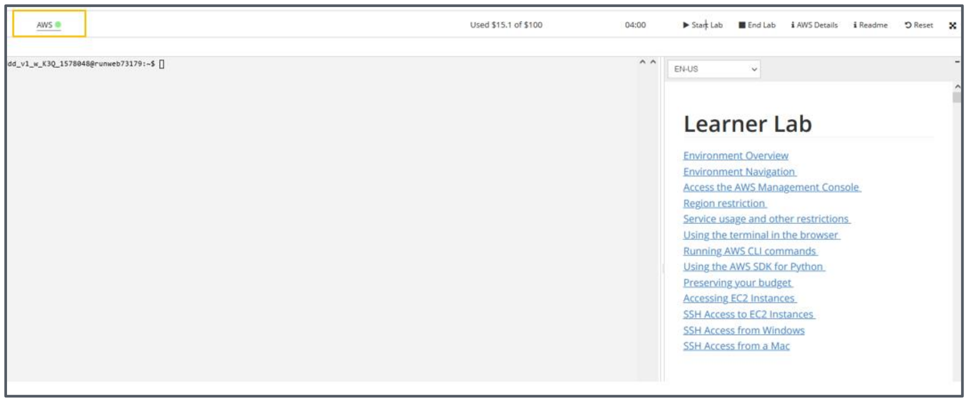


```
ddd_v1_w_K3Q_1578048@runweb73179:~$
```

Learner Lab

Environment Overview
Environment Navigation
Access the AWS Management Console
Region restriction
Service usage and other restrictions
Using the terminal in the browser
Running AWS CLI commands
Using the AWS SDK for Python
Preserving your budget
Accessing EC2 Instances
SSH Access to EC2 Instances
SSH Access from Windows
SSH Access from a Mac

- After starting the lab, wait until the AWS icon turns green as shown below:



```
dd_v1_w_K3Q_1578048@runweb73179:~$
```

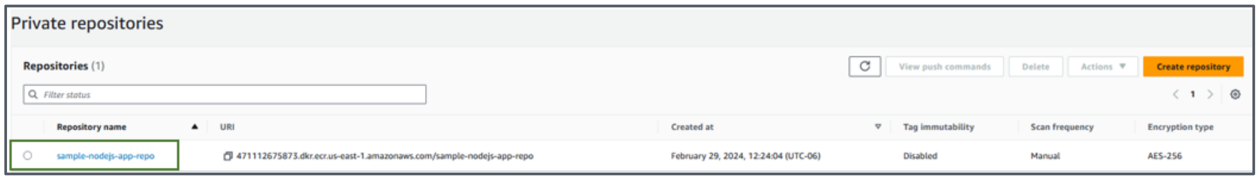
Learner Lab

Environment Overview
Environment Navigation
Access the AWS Management Console
Region restriction
Service usage and other restrictions
Using the terminal in the browser
Running AWS CLI commands
Using the AWS SDK for Python
Preserving your budget
Accessing EC2 Instances
SSH Access to EC2 Instances
SSH Access from Windows
SSH Access from a Mac

Implementation

Create an Amazon ECR repository

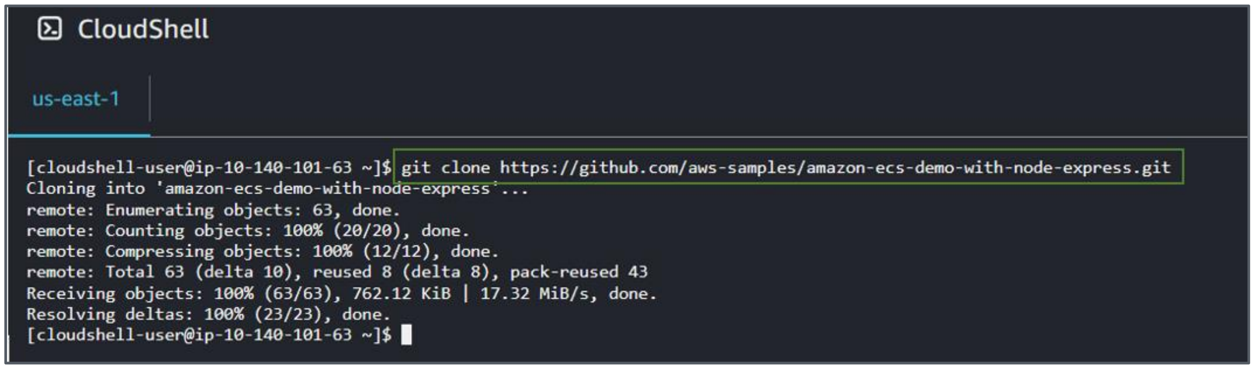
Begin by clicking on the **AWS icon** from the above screenshot to open the AWS Management Console in a new browser tab. Search for Amazon ECR and create a new **private** repository. You can name your repo **sample-nodejs-app-repo.** Click **Create repository** while keeping all other settings as default.



Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
sample-nodejs-app-repo	471112675873.dkr.ecr.us-east-1.amazonaws.com/sample-nodejs-app-repo	February 29, 2024, 12:24:04 (UTC-06)	Disabled	Manual	AES-256

Download the application code, build docker images, and push the images to ECR repository.

We will use CloudShell as our terminal to interact with our ECR repository. Search for CloudShell from the Console and clone the application code from here. The screenshot below shows how to clone an application code. Locate the URL in the **git** command from the application code repository linked in the first sentence.

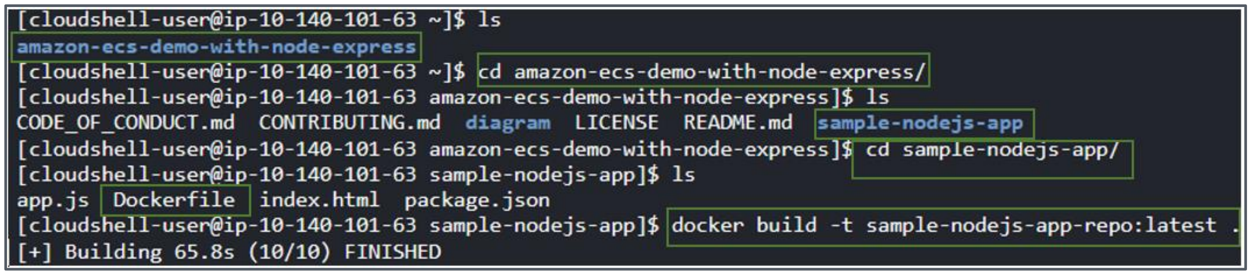


```
[cloudshell-user@ip-10-140-101-63 ~]$ git clone https://github.com/aws-samples/amazon-ecs-demo-with-node-express.git
Cloning into 'amazon-ecs-demo-with-node-express'...
remote: Enumerating objects: 63, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 63 (delta 10), reused 8 (delta 8), pack-reused 43
Receiving objects: 100% (63/63), 762.12 KiB | 17.32 MiB/s, done.
Resolving deltas: 100% (23/23), done.
[cloudshell-user@ip-10-140-101-63 ~]$
```

Then change your directory to the folder that contains the **Dockerfile** and run the below command to build your docker image.

```
docker build -t sample-nodejs-app-repo:latest .
```

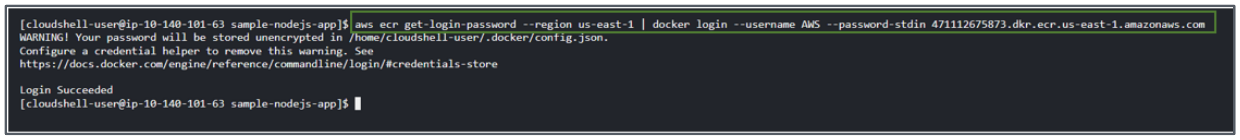
The screenshot below shows how to locate the Dockerfile folder.



```
[cloudshell-user@ip-10-140-101-63 ~]$ ls
amazon-ecs-demo-with-node-express
[cloudshell-user@ip-10-140-101-63 ~]$ cd amazon-ecs-demo-with-node-express/
[cloudshell-user@ip-10-140-101-63 amazon-ecs-demo-with-node-express]$ ls
CODE_OF_CONDUCT.md  CONTRIBUTING.md  diagram  LICENSE  README.md  sample-nodejs-app
[cloudshell-user@ip-10-140-101-63 amazon-ecs-demo-with-node-express]$ cd sample-nodejs-app/
[cloudshell-user@ip-10-140-101-63 sample-nodejs-app]$ ls
app.js  Dockerfile  index.html  package.json
[cloudshell-user@ip-10-140-101-63 sample-nodejs-app]$ docker build -t sample-nodejs-app-repo:latest .
[+] Building 65.8s (10/10) FINISHED
```

After successfully building your docker image, login to your ECR repo using the provided command from your repo. You can access this command and the other two commands for *tagging* and *pushing* your image to the ECR repo from your sample-nodejs-app-repo **View push commands** button.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 471112675873.dkr.ecr.us-east-1.amazonaws.com
```

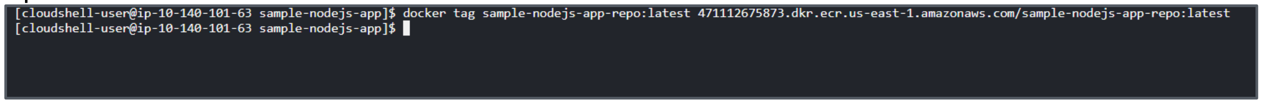


```
[cloudshell-user@ip-10-140-101-63 sample-nodejs-app]$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 471112675873.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/cloudshell-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[cloudshell-user@ip-10-140-101-63 sample-nodejs-app]$
```

Next, tag the Docker image with the ECR repository image:

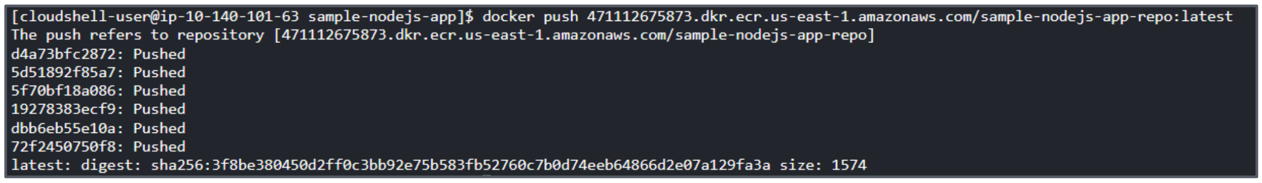
docker tag sample-nodejs-app-repo:latest 471112675873.dkr.ecr.us-east-1.amazonaws.com/sample-nodejs-app-repo:latest



```
[cloudshell-user@ip-10-140-101-63 sample-nodejs-app]$ docker tag sample-nodejs-app-repo:latest 471112675873.dkr.ecr.us-east-1.amazonaws.com/sample-nodejs-app-repo:latest
[cloudshell-user@ip-10-140-101-63 sample-nodejs-app]$
```

Finally, push the image to the ECR repository by running the command below:

docker push 471112675873.dkr.ecr.us-east-1.amazonaws.com/sample-nodejs-app-repo:latest



```
[cloudshell-user@ip-10-140-101-63 sample-nodejs-app]$ docker push 471112675873.dkr.ecr.us-east-1.amazonaws.com/sample-nodejs-app-repo:latest
The push refers to repository [471112675873.dkr.ecr.us-east-1.amazonaws.com/sample-nodejs-app-repo]
d4a73bfc2872: Pushed
5d51892f85a7: Pushed
5f70bf18a086: Pushed
19278383ecf9: Pushed
dbb6eb55e10a: Pushed
72f2450750f8: Pushed
latest: digest: sha256:3f8be380450d2ff0c3bb92e75b583fb52760c7b0d74eeb64866d2e07a129fa3a size: 1574
```

Once the image is successfully pushed to the ECR repo, your repo should contain an image as shown below:

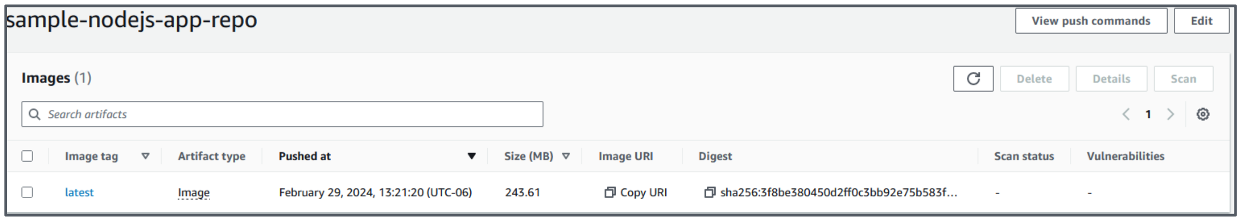


Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Scan status	Vulnerabilities
latest	Image	February 29, 2024, 13:21:20 (UTC-06)	243.61	Copy URI	sha256:3f8be380450d2ff0c3bb92e75b583f...	-	-

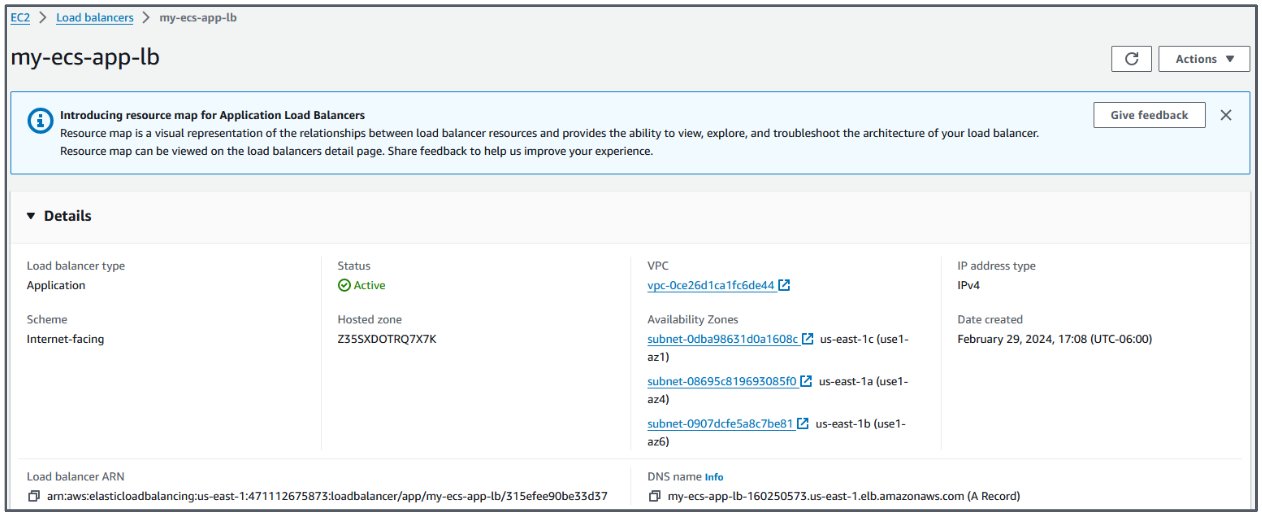
Create an Application Load Balancer

In the Console, search for Load Balancers and create an application load balancer with the following configurations:

- **Name**: my-ecs-app-lb
- **Scheme**: Internet facing
- **IP address type**: IPV4
- Select a VPC (for example: default VPC) and subnet mapping where your load balancer has to route the traffic. Also, select a security group (allow the traffic on port 80 or 443 from internet).
- Keep the listener on HTTP Port 80 and create a target-group (where the application will be running).
- **Creating a target group**:
  - **Target type**: IP addresses
  - **Target group name**: my-alb-target-grp
  - **Protocol**: HTTP on PORT: 3000
  - **IP type**: IPv4
  - Select VPC. Keep all other settings as their default value and click Next.
  - In Register targets step, don't do anything as there is no targets created as of now.
  - Create target group

- Select the created target group in load balancer screen
- Create load balancer

Once the load balancer becomes active, note the DNS name; you will need it later to access the application.



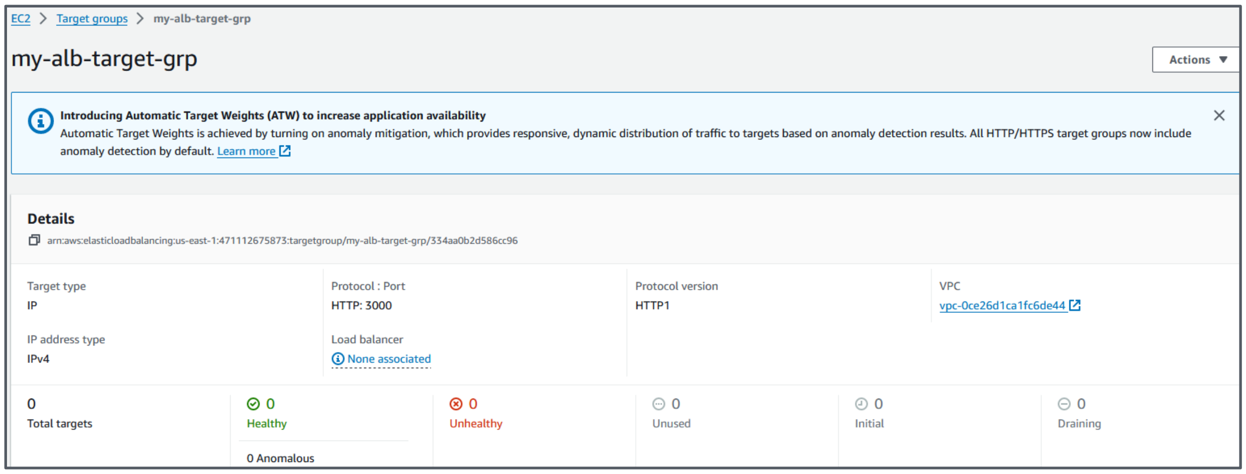
EC2 > Load balancers > my-ecs-app-lb

my-ecs-app-lb

Introducing resource map for Application Load Balancers
Resource map is a visual representation of the relationships between load balancer resources and provides the ability to view, explore, and troubleshoot the architecture of your load balancer. Resource map can be viewed on the load balancers detail page. Share feedback to help us improve your experience.

Details

Load balancer type: Application
Status: Active
VPC: vpc-0ce26d1ca1fc6de44
IP address type: IPv4
Scheme: Internet-facing
Hosted zone: Z35SXDOTRQ7X7K
Availability Zones: subnet-0dba98631d0a1608c us-east-1c (use1-az1); subnet-08695c819693085f0 us-east-1a (use1-az4); subnet-0907dcfe5a8c7be81 us-east-1b (use1-az6)
Date created: February 29, 2024, 17:08 (UTC-06:00)
Load balancer ARN: arn:aws:elasticloadbalancing:us-east-1:471112675873:loadbalancer/app/my-ecs-app-lb/315efee90be33d37
DNS name: my-ecs-app-lb-160250573.us-east-1.elb.amazonaws.com (A Record)



EC2 > Target groups > my-alb-target-grp

my-alb-target-grp

Introducing Automatic Target Weights (ATW) to increase application availability
Automatic Target Weights is achieved by turning on anomaly mitigation, which provides responsive, dynamic distribution of traffic to targets based on anomaly detection results. All HTTP/HTTPS target groups now include anomaly detection by default. Learn more

Details
arn:aws:elasticloadbalancing:us-east-1:471112675873:targetgroup/my-alb-target-grp/334aa0b2d586cc96

Target type: IP
Protocol : Port: HTTP: 3000
Protocol version: HTTP1
VPC: vpc-0ce26d1ca1fc6de44
IP address type: IPv4
Load balancer: None associated

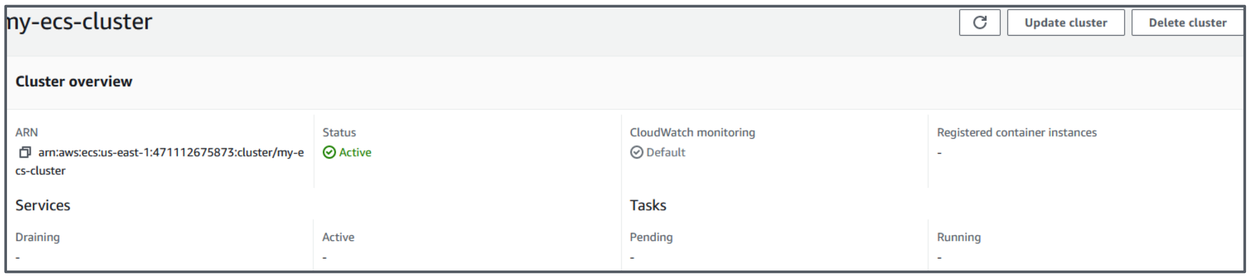
0 Total targets; 0 Healthy; 0 Anomalous; 0 Unhealthy; 0 Unused; 0 Initial; 0 Draining

Create an Amazon ECS Cluster

In the Console, search for ECS and create a new ECS cluster with the following configurations:

- **Name**: my-ecs-cluster
- **Infrastructure**: AWS Fargate (serverless)

Keep the rest of the settings as they are and click Create.



my-ecs-cluster

Cluster overview

ARN	Status	CloudWatch monitoring	Registered container instances
arn:aws:ecs:us-east-1:471112675873:cluster/my-ecs-cluster	Active	Default	-

Services

Draining	Active
-	-

Tasks

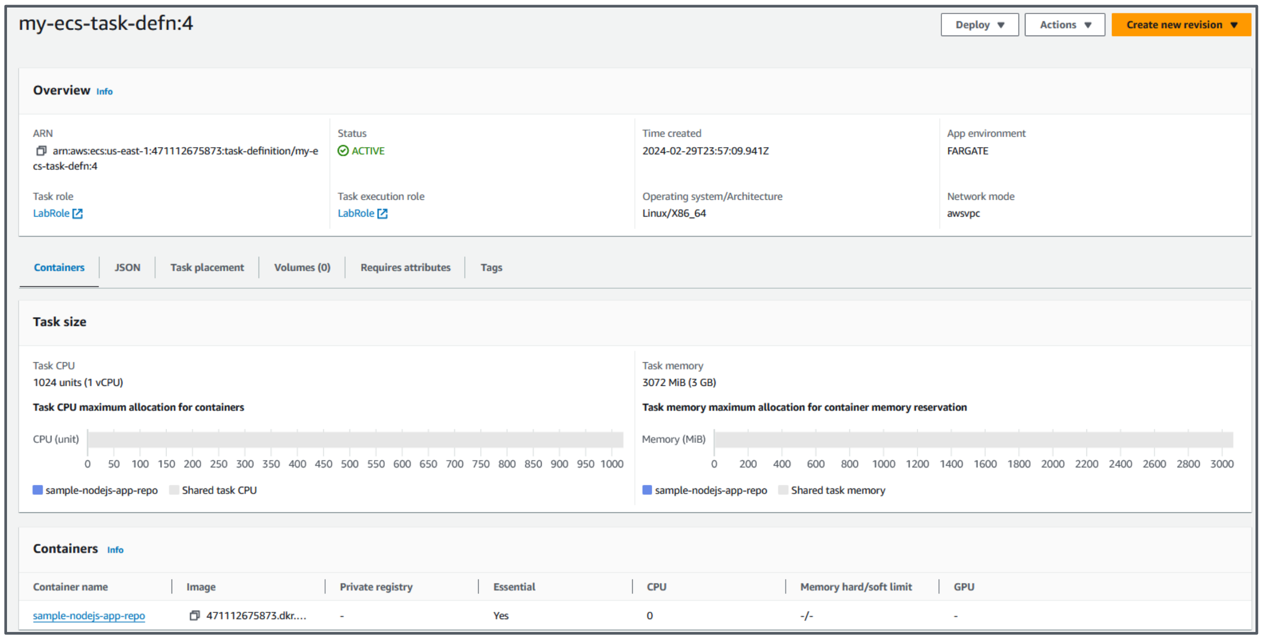
Pending	Running
-	-

Create a task definition

With the ECS cluster ready, create a task definition that describes the parameters and the containers that form the application. From the ECS page, click on Task definitions from the left panel and create a new task definition with the following details:

- **Name**: my-ecs-task-defn
- **Launch type**: AWS Fargate
- **Operating System/Architecture**: Linux/x86_64
- **Task size:** 1 vCPU and 3GB Memory
- **Task role:** Since we don't have permissions to create our own IAM role in Learner Lab, select LabRole
- **Task execution role:** Also, select LabRole
- **Under container – 1**: Enter name **sample-nodejs-app-repo** which is an ECR repo name and in image URL enter ECR repo image url - *471112675873.dkr.ecr.us-east-1.amazonaws.com/sample-nodejs-app-repo:latest*
- Container port should be *3000* because the application is configured to run on this port.

Keep the other settings as they are and click Create.



my-ecs-task-defn:4

Overview Info

ARN
arn:aws:ecs:us-east-1:471112675873:task-definition/my-ecs-task-defn:4

Status
ACTIVE

Time created
2024-02-29T23:57:09.941Z

App environment
FARGATE

Task role
LabRole

Task execution role
LabRole

Operating system/Architecture
Linux/X86_64

Network mode
awsvpc

Task size

Task CPU
1024 units (1 vCPU)

Task CPU maximum allocation for containers

■ sample-nodejs-app-repo ▢ Shared task CPU

Task memory
3072 MiB (3 GB)

Task memory maximum allocation for container memory reservation

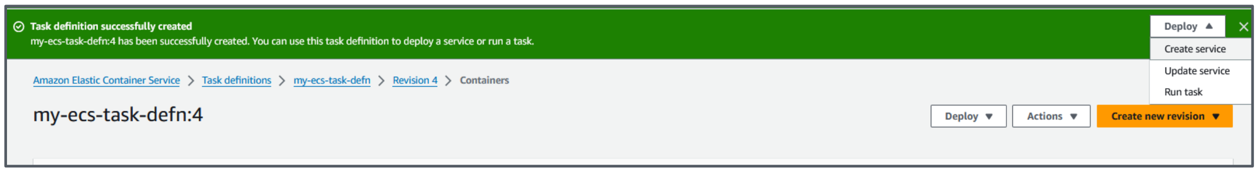
■ sample-nodejs-app-repo ▢ Shared task memory

Containers Info

Container name	Image	Private registry	Essential	CPU	Memory hard/soft limit	GPU
sample-nodejs-app-repo	471112675873.dkr....	-	Yes	0	-/-	-

Create an ECS service

Since the task has been created, select Create service from the Deploy button on the task as shown below:



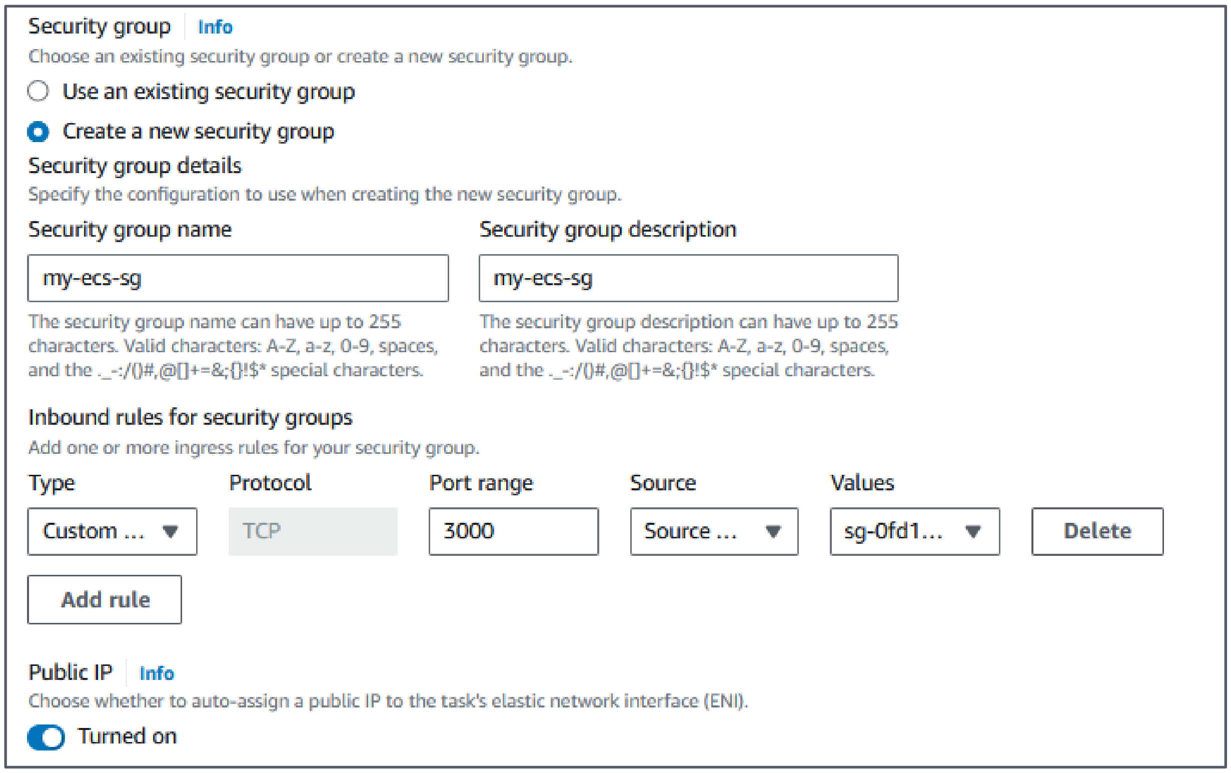
Task definition successfully created
my-ecs-task-defn:4 has been successfully created. You can use this task definition to deploy a service or run a task.

Amazon Elastic Container Service > Task definitions > my-ecs-task-defn > Revision 4 > Containers

my-ecs-task-defn:4

On the Create service page, add the following specifications for your service.

- **Compute options**: Launch type
- **Launch type**: Fargate and platform version is LATEST
- **Application type**: Service
- Under Task definition, if your task definition is not selected by default, select it with the latest version
- **Service name**: my-ecs-service
- **Service type**: Replica
- **Desired tasks**: 1
- Keep deployment options as it is
- Under networking, select the same VPC as your load balancer and subnets where you want to run your containers. Make sure that subnets you are selecting here are the same as the ones you selected in target group.
- Create a new security group which will use port 3000 of this service and is accessible only by the load balancer security group.
- Turn on Public IP


Security group Info
Choose an existing security group or create a new security group.

( ) Use an existing security group
(●) Create a new security group

Security group details
Specify the configuration to use when creating the new security group.

Security group name: my-ecs-sg
The security group name can have up to 255 characters. Valid characters: A-Z, a-z, 0-9, spaces, and the ._-:/()#,@[]+=&;{}!$* special characters.

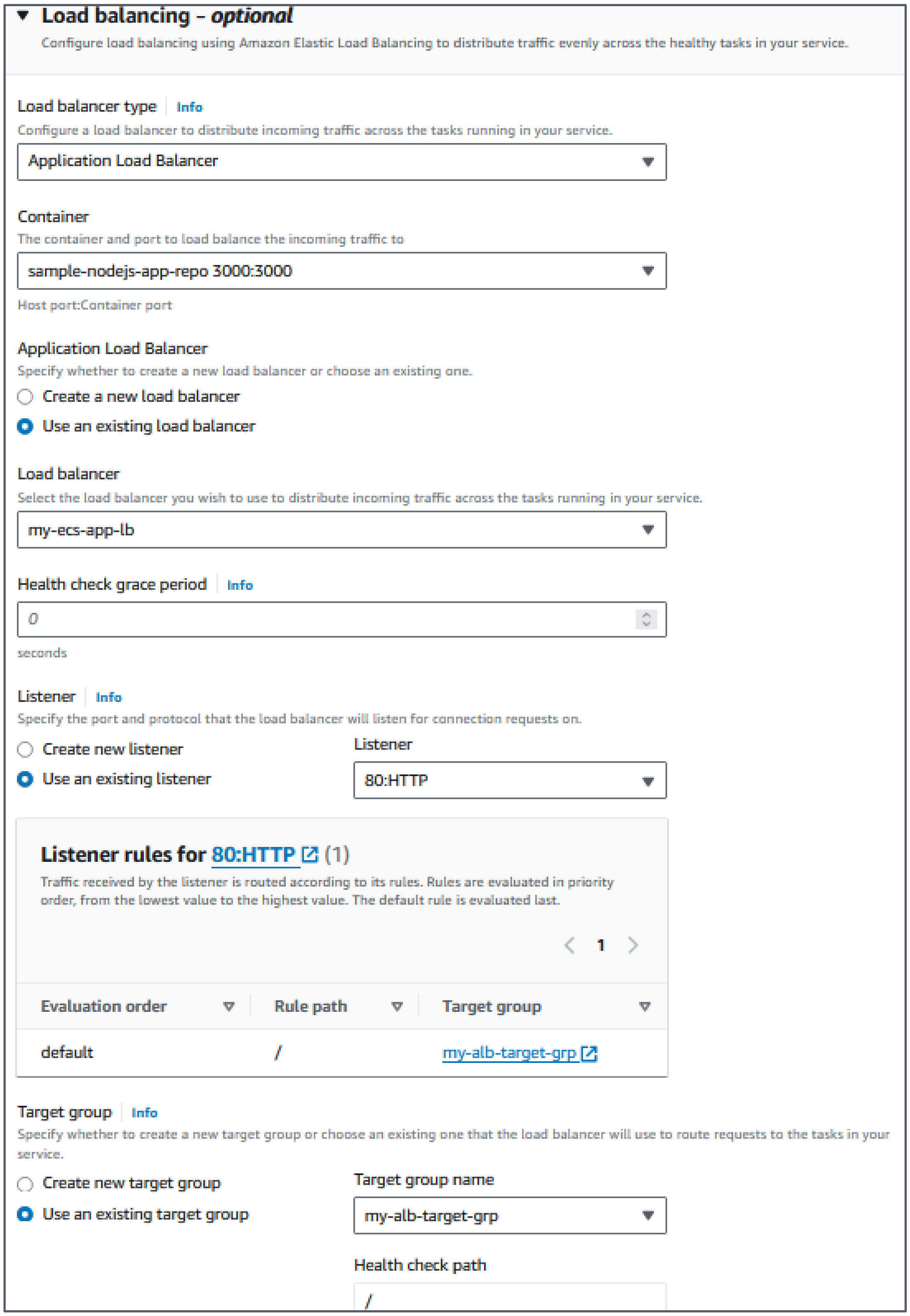
Security group description: my-ecs-sg
The security group description can have up to 255 characters. Valid characters: A-Z, a-z, 0-9, spaces, and the ._-:/()#,@[]+=&;{}!$* special characters.

Inbound rules for security groups
Add one or more ingress rules for your security group.

Type	Protocol	Port range	Source	Values	
Custom ...	TCP	3000	Source ...	sg-0fd1...	Delete

Public IP Info
Choose whether to auto-assign a public IP to the task's elastic network interface (ENI).
Turned on

- Under load balancer, select Application Load Balancer, select your already created application load balancer, listener, and target group.



Load balancing - *optional*

Configure load balancing using Amazon Elastic Load Balancing to distribute traffic evenly across the healthy tasks in your service.

Load balancer type | Info

Configure a load balancer to distribute incoming traffic across the tasks running in your service.

Application Load Balancer

Container

The container and port to load balance the incoming traffic to

sample-nodejs-app-repo 3000:3000

Host port:Container port

Application Load Balancer

Specify whether to create a new load balancer or choose an existing one.

- ○ Create a new load balancer
- ● Use an existing load balancer

Load balancer

Select the load balancer you wish to use to distribute incoming traffic across the tasks running in your service.

my-ecs-app-lb

Health check grace period | Info

0

seconds

Listener | Info

Specify the port and protocol that the load balancer will listen for connection requests on.

- ○ Create new listener
- ● Use an existing listener

Listener: 80:HTTP

Listener rules for 80:HTTP (1)

Traffic received by the listener is routed according to its rules. Rules are evaluated in priority order, from the lowest value to the highest value. The default rule is evaluated last.

Evaluation order	Rule path	Target group
default	/	my-alb-target-grp

Target group | Info

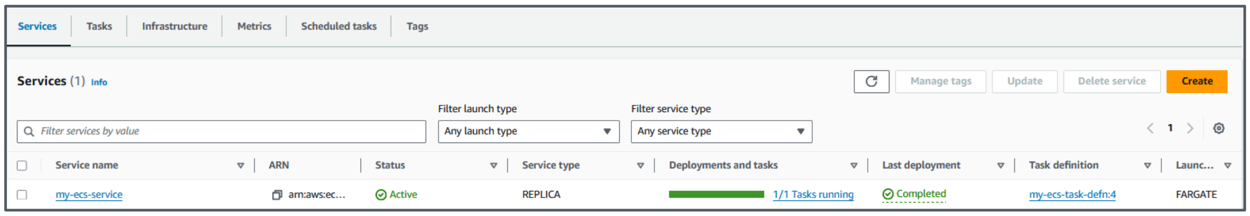
Specify whether to create a new target group or choose an existing one that the load balancer will use to route requests to the tasks in your service.

- ○ Create new target group
- ● Use an existing target group

Target group name: my-alb-target-grp

Health check path: /

Keep the rest of the settings as their default value, click create and wait till the status for service deployment becomes green as shown below:

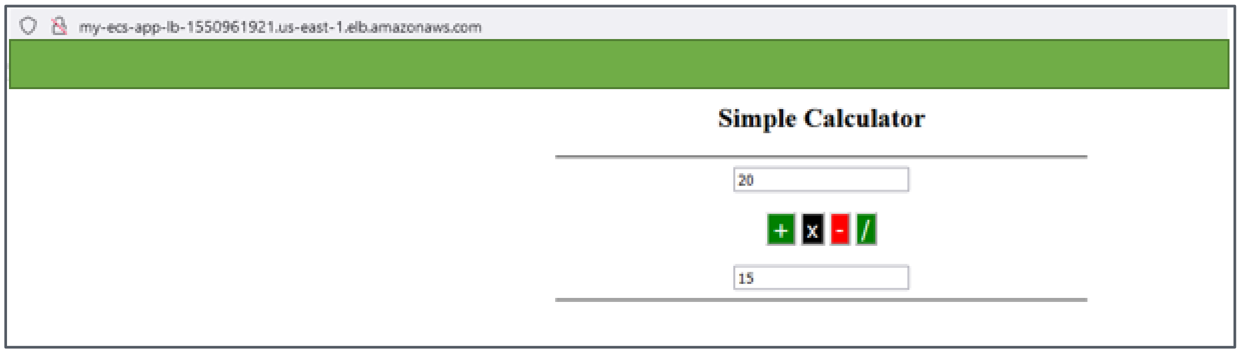


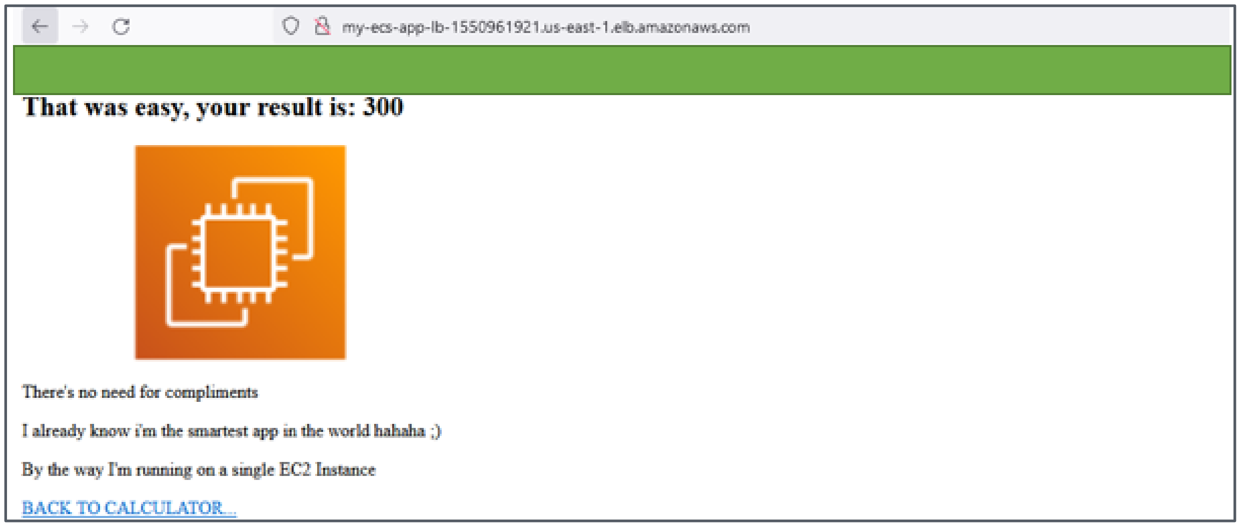
Services (1) Info

	Service name	ARN	Status	Service type	Deployments and tasks	Last deployment	Task definition	Launc...
	my-ecs-service	arn:aws:ec...	Active	REPLICA	1/1 Tasks running	Completed	my-ecs-task-defn:4	FARGATE

Test the application

Open the application load balancer, copy the DNS name, and paste in a new browser tab to load the application:





my-ecs-app-lb-1550961921.us-east-1.elb.amazonaws.com

That was easy, your result is: 300

There's no need for compliments

I already know i'm the smartest app in the world hahaha ;)

By the way I'm running on a single EC2 Instance

BACK TO CALCULATOR

NB - Don't forget to clean your resources to prevent any unexpected charge.