



AWS Project : Container Deployment with CICD

Documentation



By : Abouchiba Mohamed Yassir

To see the files used in this project go to this link:

https://github.com/M-Yassir/AWSProjects/tree/main/Container_deployment_with_CICD

Introduction

Project Description

This project demonstrates the development of a fully automated CI/CD pipeline for deploying containerized applications using Amazon Web Services (AWS). The system is designed to streamline the process of building, testing, and deploying applications, ensuring efficient and reliable delivery to end users.

The project is highly beneficial for organizations aiming to adopt modern DevOps practices, as it significantly reduces manual intervention, minimizes deployment errors, and accelerates the delivery of new features and updates. By leveraging AWS services such as CodePipeline, CodeBuild, ECR, ECS, and ALB, the pipeline ensures a seamless, scalable, and cost-effective deployment process. This approach not only enhances operational efficiency but also improves the overall reliability and performance of the application, providing a better experience for end users.

Through this project, I gained hands-on experience in CI/CD automation, containerization, cloud infrastructure, and AWS service integration, making it a valuable exploration of modern DevOps practices and scalable application deployment.

Key Features

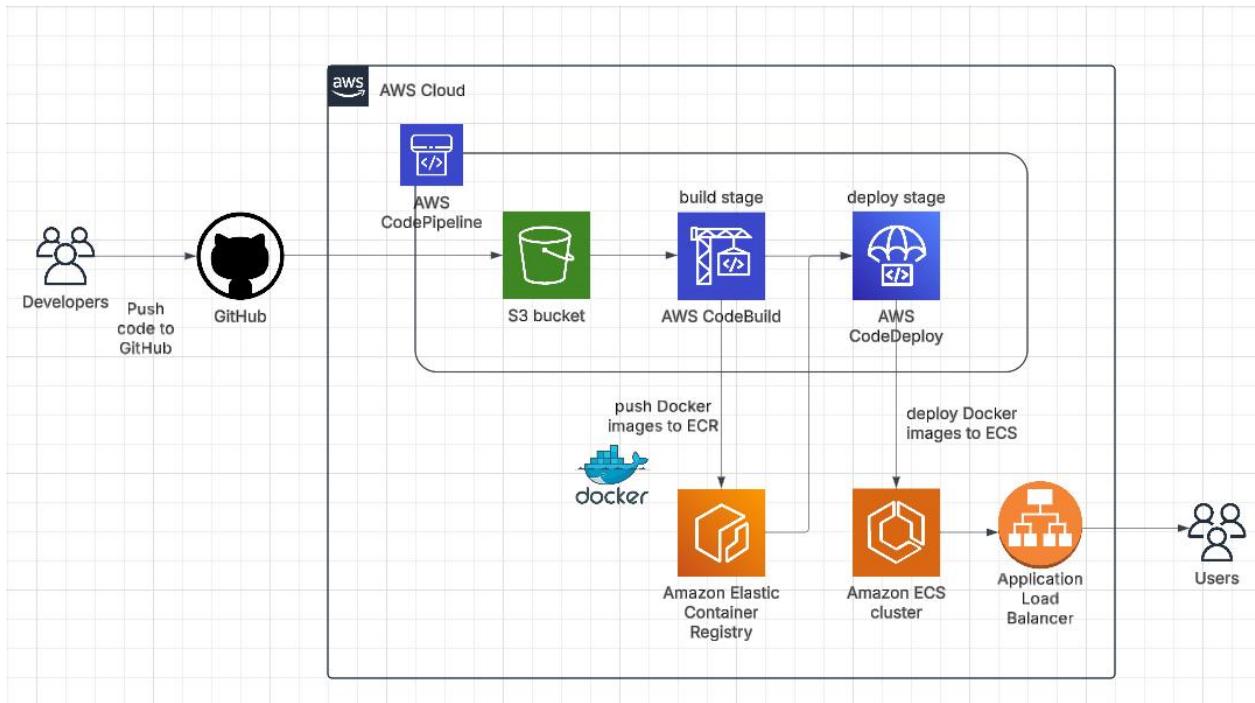
-Automated CI/CD Pipeline: The pipeline is orchestrated using AWS CodePipeline, which integrates with GitHub to detect code changes, triggers CodeBuild for Docker image creation, and uses CodeDeploy to update the ECS service. This ensures a fully automated and efficient deployment process.

-Docker Image Management: AWS CodeBuild pulls the latest code from GitHub, builds the Docker image using the provided Dockerfile, and pushes it to Amazon ECR. ECS then pulls the image from ECR to run the application in a scalable and managed environment.

-Scalable Application Deployment: The application runs on Amazon ECS, which manages the deployment of Docker containers using either EC2 instances or AWS Fargate. This ensures high availability, scalability, and efficient resource utilization.

-Traffic Routing with ALB: The Application Load Balancer (ALB) routes incoming traffic to the ECS tasks, ensuring even distribution and high availability for end users.

AWS architecture diagram



- Application workflow:

This application automates the CI/CD process for a Node.js app using AWS services and GitHub. The workflow begins when developers push code changes, such as the Node.js application code, Dockerfile, or configuration files, to the GitHub repository. This triggers AWS CodePipeline, which orchestrates the pipeline. In the Source Stage, CodePipeline pulls the latest code from GitHub, preparing it for the build process.

Next, in the Build Stage, AWS CodeBuild compiles the Node.js app and builds a Docker image using the provided Dockerfile. Once the image is created, it is pushed to Amazon Elastic Container Registry (ECR) for secure storage and versioning.

In the Deploy Stage, AWS CodeDeploy updates the Amazon Elastic Container Service (ECS) with the new Docker image from ECR. ECS runs the containers, ensuring the Node.js app is operational, scalable, and ready to handle requests. Traffic is routed through the Application Load Balancer (ALB), which distributes incoming requests evenly across the ECS tasks, providing high availability and reliability for end users.

Step-by-Step implementation

- NB:** - All the necessary files are on the repository given above.
- The creation of the app files isn't covered here as it isn't our topic.
 - Make sure that you have the latest version of the AWS CLI and Docker installed.
 - AWS CLI must be configured.

Step1: Create an ECR repository to store docker images:

Amazon ECR is a fully managed Docker container registry that stores the Docker image securely. It allows versioning of images, ensuring that the correct image is used during deployment.

We go to Amazon ECR service and we click on **create repository**, then we fill in the necessary informations as it is shown below

The screenshot shows the 'Create private repository' wizard in the AWS ECR service. The steps completed are:

- General settings:** Repository name is set to ".amazonaws.com/weather-app". Tag mutability is set to "Mutable".
- Encryption:** Encryption configuration is set to "AES-256".
- Image scanning settings - deprecated:** This section is collapsed.

At the bottom right, there are "Cancel" and "Create" buttons. The "Create" button is highlighted in orange.

We click on 'create'. Note that we are creating a private registry. We should then go to the CLI and use the following steps to authenticate and push an image to our repository:

1. Retrieve an authentication token and authenticate Docker client to our registry. Use the AWS CLI command:

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

2. Build a Docker image using the following command:

docker build -t weather-app .

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

docker tag weather-app:latest

{YOUR_ACCOUNT_ID}.dkr.ecr.{YOUR_REGION}.amazonaws.com/weather-app:latest

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

docker push

{YOUR_ACCOUNT_ID}.dkr.ecr.{YOUR_REGION}.amazonaws.com/weather-app:latest

After these 4 steps we should see that the image is been pushing on the repository like this:

```
0738bb988510: Pushed
bc4e4e530d16: Pushed
c2771b4263da: Pushed
0319727acbcd: Pushed
0d5f5a015e5d: Pushed
3c777d951de2: Pushed
f8a91dd5fc84: Pushing [=====] 62.35MB/100.4MB
cb81227abde5: Pushed
e01a454893a9: Pushing [=> 15.92MB/509.5MB
c45600adde37: Pushing [=====] 41.02MB/145.6MB
fe0fb3ab4a0f: Pushed
f1186e5061f2: Pushed
b2dba7477754: Pushing [===> 11.98MB/114.1MB
[]
```

We should wait till the upload is complete, then, on the console we should see the image uploaded to the repository:

The screenshot shows the Amazon ECR console interface. On the left, there's a navigation sidebar with sections for Private registry (Repositories, Images, Permissions, Lifecycle Policy, Repository tags, Features & Settings) and Public registry (Repositories, Settings). The main area is titled 'Images (1)' and shows a table with one row of data. The table columns are: Image tag, Artifact type, Pushed at, Size (MB), Image URI, Digest, and Last recorded pull time. The data in the table is as follows:

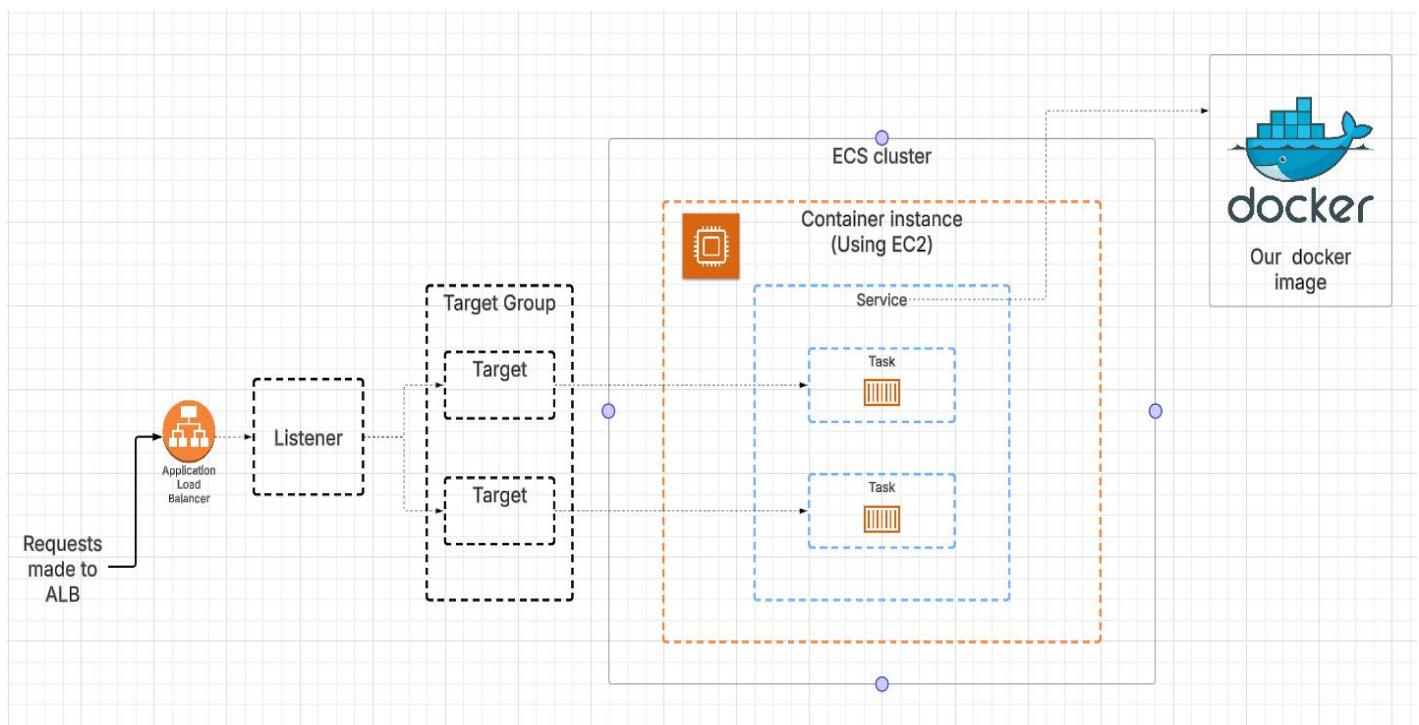
Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Last recorded pull time
-	Image	March 12, 2025, 19:10:27 (UTC-00)	357.14	Copy URI	sha256:a1698d2bd78cde...	March 12, 2025, 19:42:11 (UTC-00)

At the top right of the main area, there are buttons for 'Delete', 'Details', 'Scan', and 'View push commands'. There are also navigation controls (back, forward, search) and a refresh icon.

Step 2: Create an ECS Task, ECS Cluster, ECS service and configure the load balancer:

The Application Load Balancer (ALB) and Amazon ECS work together to ensure our containerized application, which uses a Docker image, is scalable, highly available, and accessible to users. The ALB acts as the entry point for incoming traffic, using listeners to accept requests on specific ports (e.g., HTTP on port 80 or HTTPS on port 443) and forwarding them to a target group. The target group is associated with our ECS service, which manages the tasks (containers) running our Dockerized application. ECS automatically registers the tasks with the target group, allowing the ALB to distribute traffic evenly across them. The ALB also performs health checks to ensure only healthy tasks receive traffic. If a task fails, ECS replaces it, maintaining the desired number of running tasks. This seamless integration ensures our application is always available, scalable, and responsive to user requests.

The diagram below illustrates the functioning of these services.



First, we create 2 security groups one for the application load balancer and one for securing the traffic from the application load balancer to ECS, we go to the EC2 service, choose security groups, click on **create security group** and we fill in the necessary informations:

Create security group Info

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name Info

Name cannot be edited after creation.

Description Info

VPC Info

Inbound rules Info

Type Info

Protocol Info

Port range Info

Source Info

Description - optional Info



Outbound rules Info

Type Info

Protocol Info

Port range Info

Destination Info

Description - optional Info



Tags - optional

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

No tags associated with the resource.

You can add up to 50 more tags

[Cancel](#)[Create security group](#)

Create security group Info

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name Info

Name cannot be edited after creation.

Description Info

VPC Info

Inbound rules Info

Type Info

Protocol Info

Port range Info

Source Info

Description - optional Info



Outbound rules Info

Type Info

Protocol Info

Port range Info

Destination Info

Description - optional Info



Rules with destination of 0.0.0.0/0 or ::/0 allow your instances to send traffic to any IPv4 or IPv6 address. We recommend setting security group rules to be more restrictive and to only allow traffic to specific known IP addresses.

Note that the **weather-app-alb-sg** is on the source of this security group

Tags - optional

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

No tags associated with the resource.

You can add up to 50 more tags

[Cancel](#)[Create security group](#)

We'll create now an ALB, we go to EC2 on the loadBalancers tab, click on create load Balancer

The screenshot shows the AWS EC2 console with the 'Load balancers' tab selected. The left sidebar has sections for AMI Catalog, Elastic Block Store (Volumes, Snapshots, Lifecycle Manager), Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces), Load Balancing (Load Balancers, Target Groups, Trust Stores), Auto Scaling (Auto Scaling Groups), and Settings. The main area is titled 'Load balancers' and says 'No load balancers'. It includes a search bar 'Filter load balancers', columns for Name, DNS name, State, VPC ID, Availability Zones, Type, and Date created, and a 'Create load balancer' button.

We select **Application Load Balancer** and we fill in the necessary informations:

The screenshot shows the 'Create Application Load Balancer' wizard. Step 1: Basic configuration. It includes fields for Load balancer name (Weather-app-lb), Scheme (Internet-facing selected), Load balancer IP address type (IPv4 selected), and a note about public IPv4 IP Address Management (IPAM). A summary at the bottom indicates 1 step remaining.

Network mapping [Info](#)

The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.

VPC [Info](#)

The load balancer will exist and scale within the selected VPC. The selected VPC is also where the load balancer targets must be hosted unless routing to Lambda or on-premises targets, or if using VPC peering. To confirm the VPC for your targets, view [target groups](#). For a new VPC, [create a VPC](#).

vpc-0bb0d666ceb0a46d6	Copy
IPv4 VPC CIDR: 172.31.0.0/16	Edit

IP pools - new [Info](#)

You can optionally choose to configure an IPAM pool as the preferred source for your load balancers IP addresses. Create or view [Pools](#) in [Amazon VPC IP Address Manager console](#).

[Use IPAM pool for public IPv4 addresses](#)
The IPAM pool you choose will be the preferred source of public IPv4 addresses. If the pool is depleted IPv4 addresses will be assigned by AWS.

Availability Zones and subnets [Info](#)

Select at least two Availability Zones and a subnet for each zone. A load balancer node will be placed in each selected zone and will automatically scale in response to traffic. The load balancer routes traffic to targets in the selected Availability Zones only.

us-east-1a (use1-az2)
Subnet
Only CIDR blocks corresponding to the load balancer IP address type are used. At least 8 available IP addresses are required for your load balancer to scale efficiently.
subnet-0950ea98be2c1626d
IPv4 subnet CIDR: 172.31.80.0/20

us-east-1b (use1-az4)
Subnet
Only CIDR blocks corresponding to the load balancer IP address type are used. At least 8 available IP addresses are required for your load balancer to scale efficiently.
subnet-018da8ab505c1ddf4
IPv4 subnet CIDR: 172.31.16.0/20

us-east-1c (use1-az6)
Subnet
Only CIDR blocks corresponding to the load balancer IP address type are used. At least 8 available IP addresses are required for your load balancer to scale efficiently.
subnet-0d72a6195b6ac6f7a
IPv4 subnet CIDR: 172.31.32.0/20

us-east-1d (use1-az1)

us-east-1e (use1-az3)

us-east-1f (use1-az5)

Listeners and routing [Info](#)

A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.

Listener HTTP:80 [Remove](#)

Protocol	Port	Default action Info
HTTP	: 80	Forward to Select a target group Create target group
Listener tags - optional Consider adding tags to your listener. Tags enable you to categorize your AWS resources so you can more easily manage them.		
Add listener tag You can add up to 50 more tags.		
Add listener		

Load balancer tags - optional
Consider adding tags to your load balancer. Tags enable you to categorize your AWS resources so you can more easily manage them. The 'Key' is required, but 'Value' is optional. For example, you can have Key = production-webserver, or Key = webserver, and Value = production.

We click on **Create target group**

Another page will appear to create the target group and we fill in the necessary informations:

Register targets**Basic configuration**

Settings in this section can't be changed after the target group is created.

Choose a target type Instances

- Supports load balancing to instances within a specific VPC.
- Facilitates the use of [Amazon EC2 Auto Scaling](#) to manage and scale your EC2 capacity.

 IP addresses

- Supports load balancing to VPC and on-premises resources.
- Facilitates routing to multiple IP addresses and network interfaces on the same instance.
- Offers flexibility with microservice based architectures, simplifying inter-application communication.
- Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.

 Lambda function

- Facilitates routing to a single Lambda function.
- Accessible to Application Load Balancers only.

 Application Load Balancer

- Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
- Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.

Target group name

Weather-App-TG

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Protocol : Port

Choose a protocol for your target group that corresponds to the Load Balancer type that will route traffic to it. Some protocols now include anomaly detection for the targets and you can set mitigation options once your target group is created. This choice cannot be changed after creation

HTTP

80

1-65535

IP address type

Only targets with the indicated IP address type can be registered to this target group.

 IPv4

Each instance has a default network interface (eth0) that is assigned the primary private IPv4 address. The instance's primary private IPv4 address is the one that will be applied to the target.

 IPv6Each instance register must have an assigned primary IPv6 address. This is configured on the instance's default network interface (eth0). [Learn more](#)**VPC**

Select the VPC with the instances that you want to include in the target group. Only VPCs that support the IP address type selected above are available in this list.

-
vpc-0bb0d666ceb0a46d6
IPv4 VPC CIDR: 172.31.0.0/16**Protocol version** HTTP1

Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.0 or HTTP/2.

 HTTP2

Send requests to targets using HTTP/2. Supported when the request protocol is HTTP/2 or gRPC, but gRPC-specific features are not available.

 gRPC

Send requests to targets using gRPC. Supported when the request protocol is gRPC.

Health checks

The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.

Health check protocol

HTTP

Health check path

Use the default path of "/" to perform health checks on the root, or specify a custom path if preferred.

/

Up to 1024 characters allowed.

► Advanced health check settings**Attributes** Certain default attributes will be applied to your target group. You can view and edit them after creating the target group.**► Tags - optional**

Consider adding tags to your target group. Tags enable you to categorize your AWS resources so you can more easily manage them.

Cancel

Next

We click on 'Next'

EC2 > Target groups > Create target group

Step 1 Specify group details
Step 2 Register targets
Register targets

This is an optional step to create a target group. However, to ensure that your load balancer routes traffic to this target group you must register your targets.

Available instances (0)

No instances

Ports for the selected instances

80
1-65535 (separate multiple ports with commas)
Include as pending below

Review targets

Targets (0)

No instances added yet
Specify instances above, or leave the group empty if you prefer to add targets later.

0 pending

Cancel Previous Create target group

We click on Create 'target group'
 Back to our ALB creation, we refresh and select our target group created earlier

Listeners and routing Info

A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.

Listener HTTP:80

Protocol	Port
HTTP	: 80 1-65535

Default action | Info

Forward to Weather-App-TG
Target type: Instance, IPv4
HTTP

Create target group

Listener tags - optional

Consider adding tags to your listener. Tags enable you to categorize your AWS resources so you can more easily manage them.

Add listener tag

You can add up to 50 more tags.

Add listener

Load balancer tags - optional

Consider adding tags to your load balancer. Tags enable you to categorize your AWS resources so you can more easily manage them. The 'Key' is required, but 'Value' is optional. For example, you can have Key = production-webserver, or Key = webserver, and Value = production.

Optimize with service integrations - optional Info

Optimize your load balancing architecture by integrating AWS services with this load balancer at launch. You can also add these and other services after your load balancer is created by reviewing the load balancer's "Integrations" tab.

We leave all the other things as they are and we click on 'create load balancer', we should get this:

EC2 > Load balancers > Weather-app-lb

Weather-app-lb

Details

Load balancer type	Status	VPC	Load balancer IP address type
Application	Provisioning	vpc-0bbb0d666ceb0a46d6	IPv4
Scheme	Hosted zone	Availability Zones	Date created
Internet-facing	Z35SXDOTRQ7X7K	subnet-018da8ab305c1dd1f4 us-east-1b (use1-az2) subnet-0950ea98be2c1626d1 us-east-1a (use1-az2) subnet-0d72a6195b6ac6f7a us-east-1c (use1-az6)	March 22, 2025, 12:49 (UTC+00:00)

Load balancer ARN: arn:aws:elasticloadbalancing:us-east-1:577638379285:loadbalancer/app/Weather-app-lb/895e102aee0839a0

DNS name info: Weather-app-lb-1650679389.us-east-1.elb.amazonaws.com (A Record)

Listeners and rules (1) Info

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate	mTLS	Trust store
HTTP:80	Forward to target group	1 rule	ARN	Not applicable	Not applicable	Not applicable	Not applicable
	• Weather-App-TG	1 (100%)					
	• Target group stickiness: Off						

Listeners and rules Manage rules Manage listener Add listener

Filter listeners

Protocol:Port Default action Rules ARN Security policy Default SSL/TLS certificate mTLS Trust store

HTTP:80 Forward to target group 1 rule ARN Not applicable Not applicable Not applicable Not applicable

Weather-App-TG 1 (100%) Target group stickiness: Off

Now, we're going to create an ECS task, which is a blueprint for our application. It includes details such as the Docker image to use (from ECR), CPU and memory requirements, networking settings, and environment variables. This step ensures that ECS knows how to run our Node.js application container.

We go to ECS, we click on the task definitions tab

We click on **create a new task definition**, we fill in the informations like it is shown below:

Create new task definition Info

Task definition configuration

Task definition family Info
Specify a unique task definition family name.

Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

▼ Infrastructure requirements

Specify the infrastructure requirements for the task definition.

Launch type Info
Selection of the launch type will change task definition parameters.
 AWS Fargate
Serverless compute for containers.
 Amazon EC2 instances
Self-managed infrastructure using Amazon EC2 instances.

OS, Architecture, Network mode
Network mode is used for tasks and is dependent on the compute type selected.

Operating system/Architecture <small>Info</small>	Network mode <small>Info</small>
<input type="text" value="Linux/X86_64"/>	<input type="text" value="default"/>

Task size Info
Specify the amount of CPU and memory to reserve for your task.

CPU <input type="text" value=".3 vCPU"/>	Memory <input type="text" value=".3 GB"/>
--	---

▼ Task roles - conditional

Task role Info
A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the [IAM console](#).

Task execution role Info
A task execution IAM role is used by the container agent to make AWS API requests on your behalf. If you don't already have a task execution IAM role created, we can create one for you.

Task placement - optional

Constraint | Info
Task placement constraints allow you to filter the container instances used for the placement of your tasks using built-in or custom attributes. The service scheduler first filters the container instances that match the constraints and then applies the placement strategy to place the task.

Add placement constraint
You can add 10 more placement constraints.

Fault injection - optional

Container - 1 Info

Container details
Specify a name, container image, and whether the container should be marked as essential. Each task definition must have at least one essential container.

Name weather-app-container

Image URI repository-uri/image:tag

Essential container Yes

Private registry | Info
Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.

Private registry authentication

Port mappings | Info
Add port mappings to allow the container to access ports on the host to send or receive traffic. For port name, a default will be assigned if left blank.

Host port	Container port	Protocol	Port name	App protocol
0	3000	TCP	weather-app-port	HTTP

Add port mapping

Note: The bridge network mode is selected. By default, dynamic host port mappings are used for each container. To configure a static host port mapping, specify a host port for each container port.

Read only root file system | Info
When this parameter is turned on, the container is given read-only access to its root file system.

Read only

Note: In the Image URI field, copy and paste the one created in the ECR repository during step1 and add “:latest” to it.

We leave all the other configurations as they are and we click on **create**.

Next up, we’re gonna create an ECS cluster which is a logical grouping of resources (EC2 instances or Fargate tasks) where your containers will run. The cluster provides the infrastructure needed to deploy and manage our Node.js application. We can choose to use EC2 instances for more control or AWS Fargate for a serverless approach, here we’ll choose EC2.

Amazon Elastic Container Service > Clusters

Clusters (0) Info

Last updated March 22, 2025 at 12:04 (UTC) **Create cluster**

Clusters (0)

Search clusters

Cluster	Services	Tasks	Container instances	CloudWatch monitoring
No clusters				

Install AWS Copilot

Amazon ECR

AWS Batch

Documentation

Discover products

Subscriptions

Tell us what you think

We go to ECS, click on ‘Create cluster’

Note: make sure you chose the **weather-app-alb-to-ecs** security group we created before

Amazon Elastic Container Service

Clusters

- Namespaces
- Task definitions
- Account settings

[Install AWS Copilot](#)

Amazon ECR

[Repositories](#)

AWS Batch

Documentation

[Discover products](#)

[Subscriptions](#)

[Tell us what you think](#)

Cluster name
weather-app-cluster
Cluster name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Default namespace - optional
Select the namespace to specify a group of services that make up your application. You can overwrite this value at the service level.

▼ Infrastructure Info
Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances.

AWS Fargate (serverless)
Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

Amazon EC2 instances
Manual configurations. Use for large workloads with consistent resource demands.

Auto Scaling group (ASG) | Info
Use Auto Scaling groups to scale the Amazon EC2 instances in the cluster.
[Create new ASG](#)

Provisioning model
Select a provisioning model for your instances

On-demand
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 let you take advantage of unused EC2 capacity in the AWS cloud. Spot instances are available at up to a 90% discount compared to on-demand prices.

Container instance Amazon Machine Image (AMI)
Choose the Amazon ECS-optimized AMI for your instance.
[Amazon Linux 2 \(kernel 5.10\)](#)

EC2 instance type
Choose based on the workloads you plan to run on this cluster.

t2.micro	Free tier eligible
i3.8xlarge	
1 vCPU	1 GiB Memory

EC2 instance role
An instance role is used by Amazon EC2 instances to make AWS API requests. If you don't already have an instance IAM role created, we can create one for you.
[Create new role](#)

Desired capacity
Specify the number of instances to launch in your cluster.

Minimum	Maximum
2	3

SSH Key pair
If you do not specify a key pair, you can't connect to the instances via SSH unless you choose an AMI that is configured to allow users another way to log in.
[None - unable to SSH](#)

Root EBS volume size
You can increase the size of the root EBS volume to allow for greater image and container storage.

A min of 30 GiB and a max of 16,384 GiB is allowed.

External instances using ECS Anywhere can be registered after cluster creation is complete.

▼ Network settings for Amazon EC2 instances Info
By default Amazon EC2 instances are launched in the default subnets for your default VPC. To use the non-default VPC, specify the VPC and subnets.

VPC
Select a VPC to use for your Amazon ECS resources.
[vpc-0bb0d666ceb0a46d6](#)

Subnets
Select the subnets where your instances are launched and your tasks run. We recommend that you use three subnets for production.
[Choose subnets](#)
[subnet-018da8ab305c1ddf4](#) us-east-1b 172.31.16.0/20
[subnet-0950ea98be2c1626d](#) us-east-1a 172.31.80.0/20
[subnet-0d72a6195b6ac6f7a](#) us-east-1c 172.31.32.0/20

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 name
Choose an existing security group.
[Choose security groups](#)
[sg-0bb13f984e5c403bc](#) weather-app-alb-to-ecs-sg

Auto-assign public IP Info
Choose whether to auto-assign a public IP to the Amazon EC2 instances
[Use subnet setting](#)

▼ Monitoring - optional Info
CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices.

▼ Encryption - optional
Choose the KMS keys used by tasks running in this cluster to encrypt your storage.

▼ Tags - optional Info
Tags help you to identify and organize your clusters.

[Cancel](#)[Create](#)

After filling all the informations, click on ‘create’

We'll now walk through the process of creating an ECS Service, a core feature of an Amazon ECS cluster that helps us manage and maintain our containerized applications. The main goal of an ECS Service is to ensure that a specific number of tasks (containers) are always running and healthy. It provides us with scalability, high availability, and the ability to seamlessly update our application, making it easier to deploy and manage workloads in a production environment. Let's dive into the steps to create an ECS Service.

So we click on the **weather-app-cluster** created before, go to services tab, click on ‘create’ and fill in the necessary informations:

Amazon Elastic Container Service > Clusters > weather-app-cluster > Create service

Create [Info](#)

Environment

Existing cluster
weather-app-cluster

▼ Compute configuration (advanced)

Compute options [Info](#)
To ensure task distribution across your compute types, use appropriate compute options.

Capacity provider strategy
Specify a launch strategy to distribute your tasks across one or more capacity providers.

Launch type
Launch tasks directly without the use of a capacity provider strategy.

Launch type [Info](#)
Select either managed capacity (Fargate), or custom capacity (EC2 or user-managed, External instances). External instances are registered to your cluster using the ECS Anywhere capability.

EC2

Deployment configuration

Application type [Info](#)
Specify what type of application you want to run.

Service
Launch a group of tasks handling a long-running computing work that can be stopped and restarted. For example, a web application.

Task
Launch a standalone task that runs and terminates. For example, a batch job.

Task definition
Select an existing task definition. To create a new task definition, go to [Task definitions](#).

Specify the revision manually
Manually input the revision instead of choosing from the 100 most recent revisions for the selected task definition family.

Family [Info](#)
weather-app-task

Revision [Info](#)
5 (LATEST)

Service type [Info](#)
Specify the service type that the service scheduler will follow.

Replica
Place and maintain a desired number of tasks across your cluster.

Daemon
Place and maintain one copy of your task on each container instance.

Desired tasks
Specify the number of tasks to launch.
2

Availability Zone rebalancing [Info](#)
 Turn on Availability Zone rebalancing
Amazon ECS automatically detects Availability Zone imbalances in task distributions across an ECS service, and evenly redistributes ECS service tasks across Availability Zones.

Health check grace period [Info](#)
0 seconds

► Deployment options

► Deployment failure detection [Info](#)

▼ Service Connect - optional [Info](#)
Service Connect allows for service-to-service communications with automatic discovery using short names and standard ports.

Use Service Connect
Configure the namespaces, and the services to interconnect.

► Service discovery - optional
Service discovery uses Amazon Route 53 to create a namespace for your service, which allows it to be discoverable via DNS.

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

Use load balancing

▼ Load balancing - optional

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

Use load balancing

VPC

Select the VPC for your load balancing resources.

vpc-0bb0d666ceb0a46d6
default

[Create a new VPC](#)

Load balancer type | Info

Specify the load balancer type to distribute incoming traffic across the tasks running in your service.

Application Load Balancer
An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports.

Network Load Balancer
A Network Load Balancer makes routing decisions at the transport layer (TCP/UDP).

Container

The container and port to load balance the incoming traffic to

weather-app-container 80:3000

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

Choose an existing load balancer to distribute traffic. View existing load balancers and create new one in [EC2 Console](#).

Weather-app-lb
Weather-app-lb-1650679389.us-east-1.elb.amazonaws.com

internet-facing

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.

< 1 >

Evaluation order	Rule path	Target group
default	/	Weather-App-TG

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

Weather-App-TG

Health check path

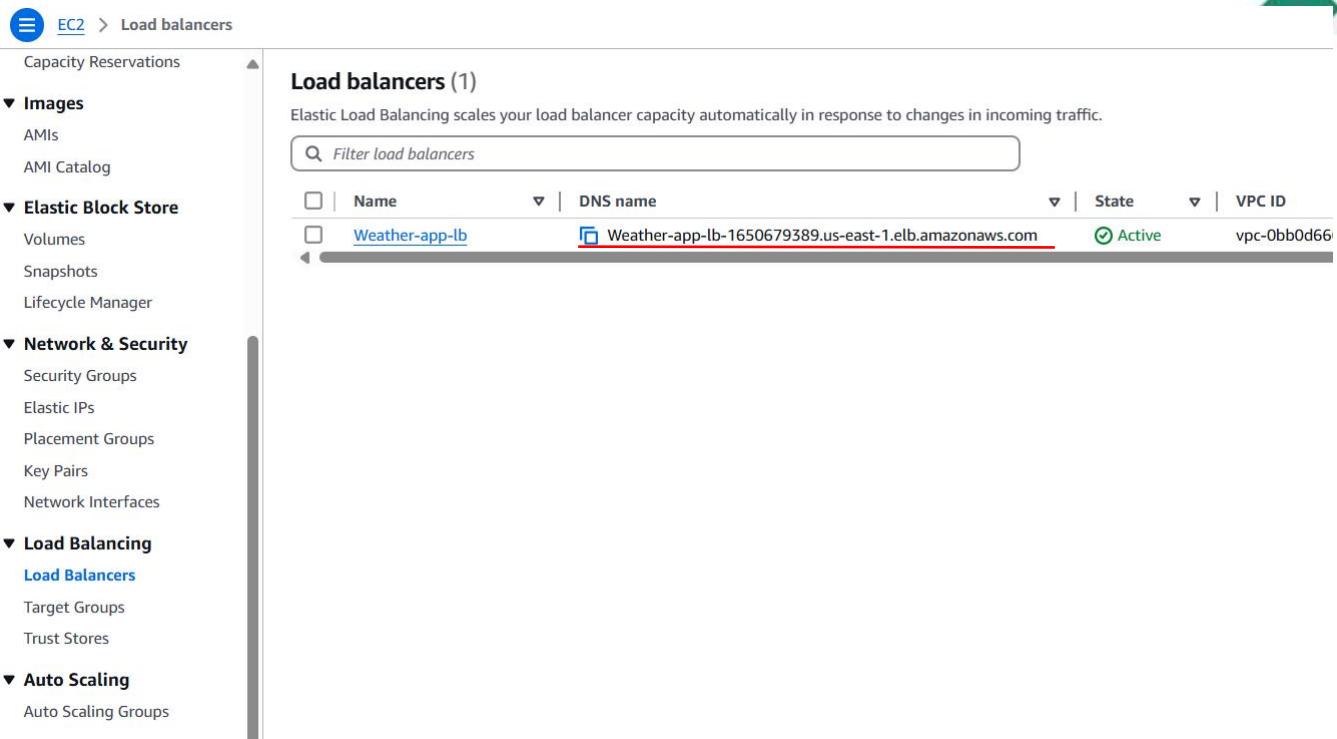
/

Health check protocol

HTTP

We leave all the other configurations as they are and we click on **create**

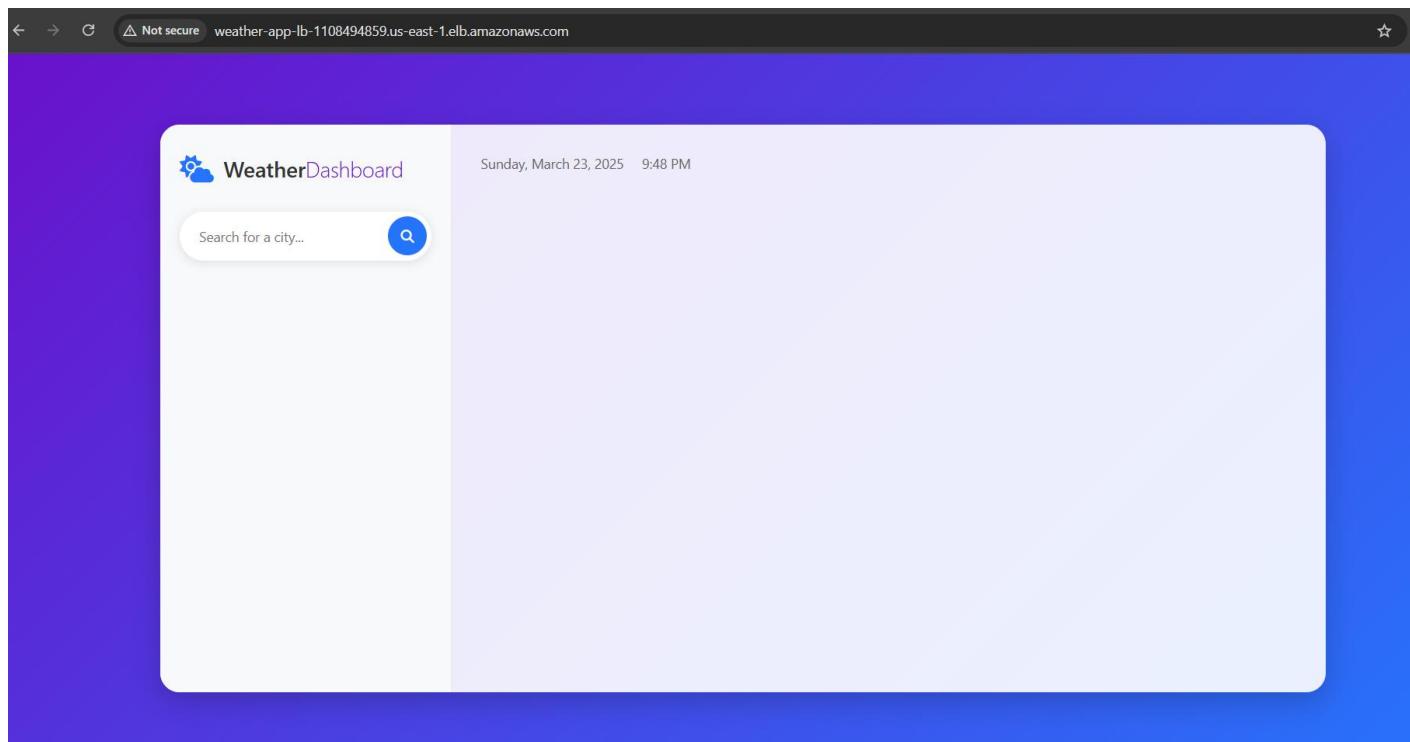
Once it has launched after some waiting time, it's time to put it to the test. We go to the load balancer, copy its DNS name:



The screenshot shows the AWS EC2 console with the navigation bar at the top. Under the 'Load balancers' section, there is a table with one row. The table columns are 'Name', 'DNS name', 'State', and 'VPC ID'. The 'Name' column contains 'Weather-app-lb'. The 'DNS name' column contains 'Weather-app-lb-1650679389.us-east-1.elb.amazonaws.com', which is highlighted with a red border. The 'State' column shows 'Active' with a green checkmark. The 'VPC ID' column shows 'vpc-0bb0d66'. A search bar labeled 'Filter load balancers' is located above the table.

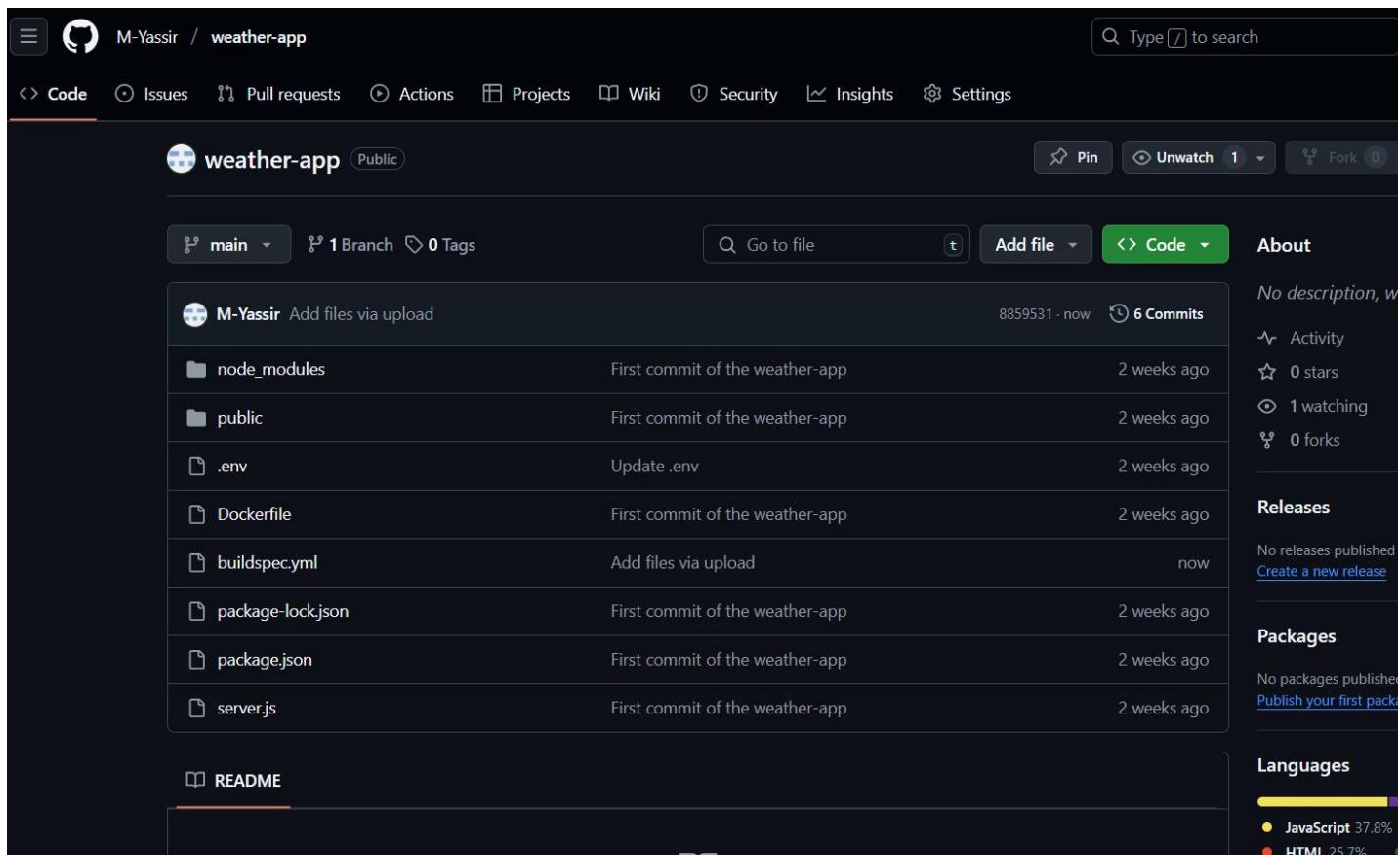
Name	DNS name	State	VPC ID
Weather-app-lb	Weather-app-lb-1650679389.us-east-1.elb.amazonaws.com	Active	vpc-0bb0d66

Paste it on the browser and we should get this:



Step3: Configure the CICD pipeline:

First, we'll upload the application files to GitHub and include a **buildspec.yml** file which defines the build process for our application in AWS CodeBuild, specifying steps like installing dependencies, building the Docker image, and pushing it to Amazon ECR. By adding it to our repository, CodeBuild automatically executes these steps when changes are pushed, streamlining our CI/CD pipeline and ensuring consistent builds, when the files are uploaded, the repo should look like this:



The screenshot shows a GitHub repository named "weather-app". The repository is public and has 6 commits. The commits are as follows:

- M-Yassir Add files via upload (8859531 · now)
- node_modules (First commit of the weather-app · 2 weeks ago)
- public (First commit of the weather-app · 2 weeks ago)
- .env (Update .env · 2 weeks ago)
- Dockerfile (First commit of the weather-app · 2 weeks ago)
- buildspec.yml (Add files via upload · now)
- package-lock.json (First commit of the weather-app · 2 weeks ago)
- package.json (First commit of the weather-app · 2 weeks ago)
- server.js (First commit of the weather-app · 2 weeks ago)

The repository also contains a README file.

On the right side of the repository page, there are sections for About, Activity, Releases, Packages, and Languages. The Languages section shows that the code is primarily written in JavaScript (37.8%) and HTML (25.7%).

Then, we'll create a build project in code build, we go to codebuild, click on 'create project' and fill the necessary informations

Note: make sure you connect your github account to aws, you can do it when creating the build project by clicking on [Manage account credentials](#)

Create build project

Project configuration

Project name
A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special characters - and _.

Project type
Select what type of project you would like to create. [Info](#)

Default project
Create a custom CodeBuild project.

Runner project
Create a CodeBuild managed runner for workflows in GitHub Actions, GitHub Enterprise Actions, GitLab, or Buildkite.

► Additional configuration
Description, public build access, build badge, concurrent build limit, tags

Source

Add source

Source 1 - Primary

Source provider

Credential
 Your account is successfully connected by using an AWS managed GitHub App. [Manage account credentials](#).

Use override credentials for this project only

Repository
 Repository in my GitHub account
 Public repository
 GitHub scoped webhook

Repository
 [X](#) [C](#)

Source version - *optional* [Info](#)
Enter a pull request, branch, commit ID, tag, or reference and a commit ID.

► Additional configuration
Git clone depth, Git submodules, Build status config

Primary source webhook events [Info](#)

Webhook - *optional* [Info](#)
 Rebuild every time a code change is pushed to this repository

Environment

Provisioning model [Info](#)

On-demand
Automatically provision build infrastructure in response to new builds.

Reserved capacity
Use a dedicated fleet of instances for builds. A fleet's compute and environment type will be used for the project.

Environment image
 Managed image
Use an image managed by AWS CodeBuild

Custom image
Specify a Docker image

Compute
 EC2
Optimized for flexibility during action runs

Lambda
Optimized for speed and minimizes the start up time of workflow actions

Running mode

Container
 Running on Docker container

Instance
 Running on EC2 instance directly

Operating system

Ubuntu

Runtime(s)

Standard

Image

aws/codebuild/standard:7.0

Image version

Always use the latest image for this runtime version

Use GPU-enhanced compute

Service role

New service role
 Create a service role in your account

Existing service role
 Choose an existing service role from your account

Role name

code-build-weather-app-role

Type your service role name

▼ Additional configuration

Timeout, privileged, certificate, VPC, compute type, environment variables, file systems, auto-retry, registry credential

Auto-retry limit

CodeBuild will automatically call retry build using the project's service role up to the auto-retry limit

0

Timeout

Default timeout is 1 hour

Hours	Minutes
1	0

Timeout must be between 5 minutes and 36 hours

Queued timeout

Default time in build queue is 8 hours

Hours	Minutes
8	0

Timeout must be between 5 minutes and 8 hours

Privileged

Enable this flag if you want to build Docker images or want your builds to get elevated privileges

Report auto-discover [Info](#)

Disable report auto-discover

Auto-discover directory - optional

**/*

CodeBuild will search for supported report file types in this directory. **/* by default

Certificate

If you have a self-signed certificate or a certificate signed by a certification authority, choose the option to install it from your S3 bucket.

Do not install any certificate

Install certificate from your S3 bucket

VPC

Select a VPC that your AWS CodeBuild project will access.

Compute

3 GB memory, 2 vCPUs

7 GB memory, 4 vCPUs

15 GB memory, 8 vCPUs

- 70 GB memory, 36 vCPUs
- 145 GB memory, 72 vCPUs

Registry credential - *optional*

You can supply a secret name or ARN from AWS Secrets Manager to authenticate to an external registry. The credentials can be used for image overrides from self-hosted runners.

Environment variables

Name	Value	Type	
<input type="text"/>	<input type="text"/>	Plaintext	<input type="button" value="Remove"/>

File systems

Identifier	ID	
<input type="text"/>	<input type="text"/>	<input type="button" value="Remove"/>

Directory path - *optional*

Mount point

Mount options - *optional*

▼ Buildspec

Build specifications

Build specifications

- Insert build commands

Store build commands as build project configuration

- Use a buildspec file

Store build commands in a YAML-formatted buildspec file

Buildspec name - *optional*

By default, CodeBuild looks for a file named buildspec.yml in the source code root directory. If your buildspec file uses a different name or location, enter its path from the source root here (for example, buildspec-two.yml or configuration/buildspec.yml).

buildspec.yml

▼ Batch configuration

You can run a group of builds as a single execution. Batch configuration is also available in advanced option when starting build.

Define batch configuration - *optional*

You can also define or override batch configuration when starting a build batch.

▼ Artifacts

Artifact 1 - Primary

Type

No artifacts



You might choose no artifacts if you are running tests or pushing a Docker image to Amazon ECR.

► Additional configuration

Cache, encryption key

Logs

CloudWatch

CloudWatch logs - optional
Checking this option will upload build output logs to CloudWatch.

Group name - optional
`aws/codebuild/weather-app-project`

The group name of the logs in CloudWatch Logs. The log group name will be `/aws/codebuild/<project-name>` by default.

Stream name prefix - optional

The prefix of the stream name of the CloudWatch Logs.

S3

S3 logs - optional
Checking this option will upload build output logs to S3.

Create build project

After creating the build project, you must add an **AmazonEC2ContainerRegistryFullAccess** permission to the policy given to the code-build-weather-app-role like that:

code-build-weather-app-role Info Delete

Summary Edit

Creation date April 02, 2025, 08:46 (UTC)	ARN <code>arn:aws:iam::577638379285:role/service-role/code-build-weather-app-role</code>
Last activity -	Maximum session duration 1 hour

Permissions Trust relationships Tags Last Accessed Revoke sessions

Permissions policies (3) Info Add permissions ▾

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
<input type="checkbox"/> AmazonEC2ContainerRegistryFullAccess	AWS managed	1
<input type="checkbox"/> CodeBuildBasePolicy-code-build-weather-app-role-us-east-1	Customer managed	1
<input type="checkbox"/> CodeBuildCodeConnectionsSourceCredentialsPolicy-weather...	Customer managed	1

Finally, configuring the pipeline, we go to codepipeline, click on **create pipeline** and fill in the necessary informations:

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Step 1
Choose creation option

Step 2
Choose pipeline settings

Step 3
Add source stage

Step 4
Add build stage

Step 5
Add test stage

Step 6
Add deploy stage

Step 7
Review

Choose creation option

Step 1 of 7

Category

Deployment Continuous Integration Automation

Build custom pipeline

Cancel **Next**

Step 1
Choose creation option

Step 2
Choose pipeline settings

Step 3
Add source stage

Step 4
Add build stage

Step 5
Add test stage

Step 6
Add deploy stage

Step 7
Review

Choose pipeline settings

Step 2 of 7

Pipeline settings

Pipeline name
Enter the pipeline name. You cannot edit the pipeline name after it is created.

No more than 100 characters

Execution mode Info
Choose the execution mode for your pipeline. This determines how the pipeline is run.
 Superseded
 Queued
 Parallel

Service role

New service role
Create a service role in your account

Existing service role
Choose an existing service role from your account

Role name

Type your service role name

Allow AWS CodePipeline to create a service role so it can be used with this new pipeline

Advanced settings
Configure artifact store location, encryption settings, and pipeline variables for your pipeline.

Cancel **Previous** **Next**

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Add source stage Info

Step 1 Choose creation option
Step 2 Choose pipeline settings
Step 3 Add source stage **Add source stage**
Step 4 Add build stage
Step 5 Add test stage
Step 6 Add deploy stage
Step 7 Review

Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

GitHub (via GitHub App)

Connection
Choose an existing connection that you have already configured, or create a new one and then return to this task.

or

Repository name
Choose a repository in your GitHub account.

M-Yassir/weather-app

⚠ Cannot read properties of undefined (reading 'OwnerId') ([Click here to retry](#))

You can type or paste the group path to any project that the provided credentials can access. Use the format 'group/subgroup/project'.

Default branch
Default branch will be used only when pipeline execution starts from a different source or manually started.

main

Choose the output artifact format.

CodePipeline default
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include Git metadata about the repository.

Full clone
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only supported for AWS CodeBuild actions. [Learn more](#)

Enable automatic retry on stage failure

Webhook events

Webhook - *optional*

Start your pipeline on push and pull request events.

▶ Webhook event filters - *optional*
Starts your pipeline on a specific event.

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Step 1 Choose creation option
Step 2 Choose pipeline settings
Step 3 Add source stage
Step 4 Add build stage
Step 5 Add test stage
Step 6 Add deploy stage
Step 7 Review

Add build stage Info

Step 4 of 7

Build - *optional*

Build provider
Choose the tool you want to use to run build commands and specify artifacts for your build action.

Commands Other build providers

AWS CodeBuild

Project name
Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.

weather-app-project or

Define buildspec override - *optional*
Buildspec file or definition that overrides the latest one defined in the build project, for this build only.

Environment variables - *optional*
Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. [Learn more](#)

Name	Value
AWS_DEFAULT_REGION	your default region

Type: Plaintext

Name	Value
AWS_ACCOUNT_ID	your account id

Type: Plaintext

Name	Value
REPOSITORY_NAME	weather-app

Type: Plaintext

Name	Value
CONTAINER_NAME	weather-app-container

Type: Plaintext

Build type

Single build
Triggers a single build.

Batch build
Triggers multiple builds as a single execution.

Region
 United States (N. Virginia)

Input artifacts
Choose an input artifact for this action. [Learn more](#)

Defined by: Source

Enable automatic retry on stage failure

Developer Tools > CodePipeline > Pipelines > Create new pipeline

Add test stage Info

Step 5 of 7

Test - optional

Test provider
Choose how you want to test your application or content. Choose the provider, and then provide the configuration details for that provider.

Enable automatic retry on stage failure

Cancel Previous Skip test stage Next

Step 6 of 7

Deploy - optional

Deploy provider
Choose how you want to deploy your application or content. Choose the provider, and then provide the configuration details for that provider.

Amazon ECS

Region
United States (N. Virginia)

Input artifacts
Choose an input artifact for this action. [Learn more](#)

BuildArtifact X
Defined by: Build

No more than 100 characters

Cluster name
Choose a cluster that you have already created in the Amazon ECS console. Or create a cluster in the Amazon ECS console and then return to this task.

weather-app-cluster X

Service name
Choose a service that you have already created in the Amazon ECS console for your cluster. Or create a new service in the Amazon ECS console and then return to this task.

weather-app-service X

Image definitions file - optional
Enter the JSON file that describes your service's container name and the image and tag.

Myfilename.json

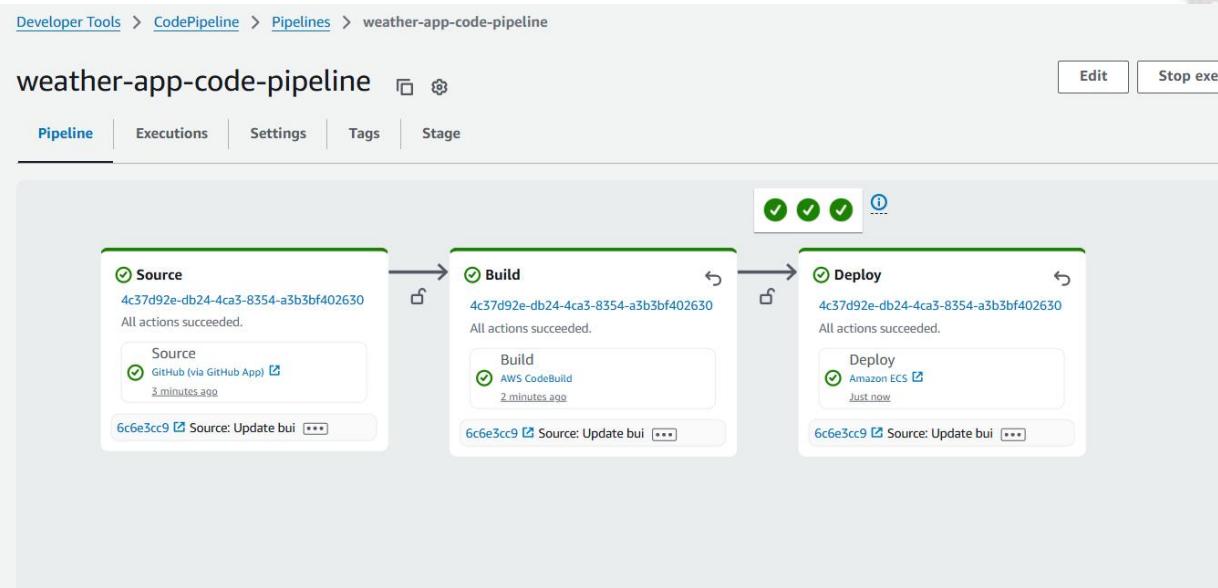
Deployment timeout - optional
Enter the timeout in minutes for the deployment action.

Configure automatic rollback on stage failure

Enable automatic retry on stage failure

Cancel Previous Skip deploy stage Next

On the review page (step 7), we click on create pipeline.
 After waiting a little bit, the pipeline should be deployed successfully, we should get something like this:



When we modify the app code and rerun the pipeline, it will automatically rebuild the Docker image, push it to ECR, and deploy the updated version to ECS. The ALB seamlessly shifts traffic to the new containers once they pass health checks, ensuring a smooth rollout without downtime. This end-to-end automation keeps our app updated with every code change.

To see a demo of this, go watch the video in the repo and don't forget to delete all of these resources (ECS cluster, ALB, task definitions, pipeline...) when you finish.

Conclusion

This initiative represented a comprehensive exploration of containerized CI/CD implementation using AWS CodePipeline. Through strategic integration of ECS, ALB, and AWS developer tools, I engineered a resilient deployment pipeline that delivered scalability while operating within rigorous Free Tier limitations.

Key challenges & technical solutions:

1) Resource Allocation Optimization

The project presented unique constraints in resource management. Initial configurations using 1 vCPU/1GB task definitions proved unsustainable, while undersized 0.1 vCPU allocations resulted in performance degradation.

Solution: Conducted methodical capacity testing to identify the optimal 0.3 vCPU/0.3GB configuration, demonstrating that precise resource calibration can achieve stability even under tight constraints.

2) Port Mapping and Load Balancer Configuration

Architecting efficient traffic distribution required solving multiple integration challenges. The initial approach using static port mapping (hostPort: 80) created scaling limitations, while ALB target group misconfigurations disrupted service health monitoring.

Solution: Implemented dynamic port allocation (hostPort: 0) and refined path-based routing rules, establishing a foundation for elastic workload distribution.

3) Pipeline Automation and Permission Management

The deployment process uncovered critical automation gaps, particularly the missing imagedefinitions.json artifact, while permission constraints hindered CodeBuild operations.

Solution: Restructured the build specification to ensure proper artifact generation and implemented granular IAM policies, achieving both functionality and security compliance.

Professional development outcomes:

This engagement yielded significant technical competencies:

- ✓ Designed and validated cost-efficient ECS architectures
- ✓ Engineered auto-scaling solutions through dynamic load balancing
- ✓ Gained practical experience with AWS IAM policy troubleshooting

- 
- ✓ Learned container optimization techniques for resource-constrained environments
 - ✓ Developed systematic debugging skills for CI/CD pipelines
 - ✓ Developed problem-solving skills for cloud deployment challenges

Strategic value & future applications:

The project not only demonstrated the effectiveness of AWS CI/CD services but also established a framework for infrastructure optimization. These learnings create opportunities to advance deployment methodologies through:

- Blue/green deployment patterns
- Infrastructure-as-Code adoption
- Kubernetes ecosystem integration

