

PHASE 4

CODEDEPLOY SETUP

STEP 1: LAUNCH EC2 WITH CLOUDFORMATION

Definition:

What is AWS CodeDeploy?

AWS CodeDeploy is a **continuous deployment** service. This means CodeDeploy...

- **Automates deployments:** Eliminates error-prone manual steps - no more manually copying files and running commands to deploy the application.
- **Enables consistent rollouts:** The application deploys the same way every time.
- **Minimizes downtime:** Can deploy in ways that keep the application available.
- **Handles failures:** Can automatically roll back if something goes wrong.
- Go to CloudFormation console in AWS

What is AWS CloudFormation?

Think of CloudFormation is AWS' **infrastructure as code** tool. Instead of clicking around the AWS console to set up resources (which gets tedious fast!), a single template file is written that describes everything needed - like EC2 instances, security groups, databases, and more. Then, CloudFormation reads this file and builds the entire environment that is needed, exactly the same way every time.

- Click on Create Stack → With new resources (from drop down)

What is a CloudFormation stack?

When a CloudFormation template is deployed , a **stack is** created - think of it as a project folder that holds all the connected resources. The cool thing is that CloudFormation treats this stack as a single unit, so we can create, update, or delete all those resources together with one command.

- Select Choose an existing template → Under Specify Template select "Upload a template" and upload the YAML file (gamerhub-cicd-cloudformation-template) as cloud formation template → Click next

The template's structure:

Resource Type	Status	Comments
VPC, Subnet, IGW, Route Table	✓	Creates public networking for EC2
Security Group	✓	Allows HTTP access from your given IP
EC2 Instance	✓	Tagged, has instance profile and public IP
IAM Role for EC2	✓	SSM & S3 read — suitable for connecting + pulling artifact
Output (URL)	✓	Prints public URL to check Flask app later

Why are we deploying networking resources too?

By defining these networking resources in the template, we're not just launching an EC2 instance, but creating a complete, secure, and configurable infrastructure that can be easily replicated or modified. This is an especially good idea for EC2 instances that are hosting web apps, because they have more complex needs like connecting with multiple databases and controlling both public and private network traffic.

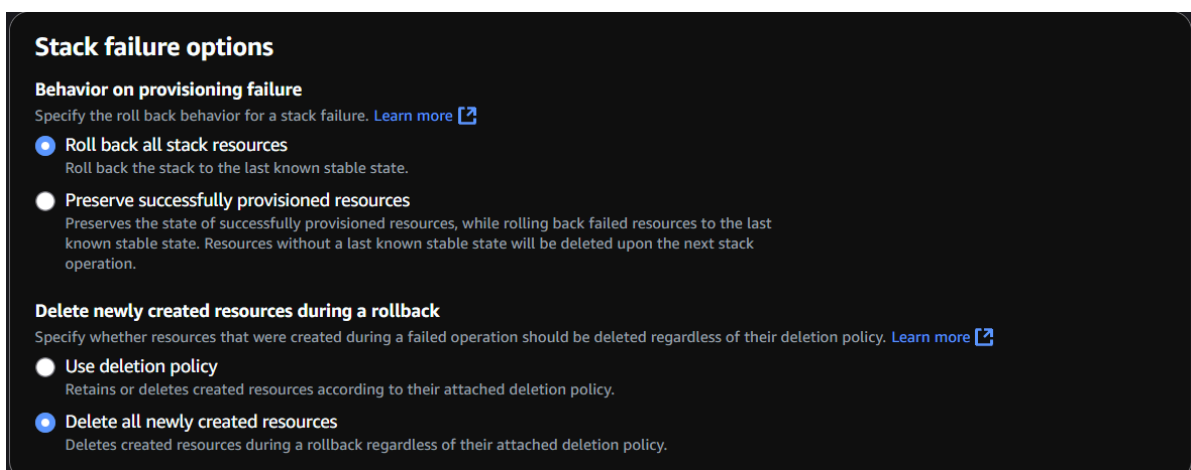
- Give a stack name like : GamerHubCodeDeployEC2Stack
- Paste the IP adress in MyIP field . Check IP using : <https://checkip.amazonaws.com/>
- Paste the IP like : 123.45.67.89/32 Note: /32 is important → Click Next

Why do we add /32 to the IP address?

Adding /32 to your IP address is like telling AWS "I only want this exact address to have access, not any others." The /32 is CIDR notation that specifies exactly how many IP addresses are included in your rule.

With /32, it's just one - your specific IP. If we used /24 instead, we'd be allowing 256 different IP addresses! For security, we're being as specific as possible to minimize who can access our EC2 instance.

- Under Stack Failure options choose Roll back all stack resources and Delete all newly created resources .



The screenshot shows the 'Stack failure options' section in the AWS CloudFormation console. It is divided into two main parts: 'Behavior on provisioning failure' and 'Delete newly created resources during a rollback'. In the first part, 'Roll back all stack resources' is selected. In the second part, 'Delete all newly created resources' is selected.

Stack failure options

Behavior on provisioning failure
Specify the roll back behavior for a stack failure. [Learn more](#)

- ☒ **Roll back all stack resources**
Roll back the stack to the last known stable state.
- ☐ **Preserve successfully provisioned resources**
Preserves the state of successfully provisioned resources, while rolling back failed resources to the last known stable state. Resources without a last known stable state will be deleted upon the next stack operation.

Delete newly created resources during a rollback
Specify whether resources that were created during a failed operation should be deleted regardless of their deletion policy. [Learn more](#)

- ☐ **Use deletion policy**
Retains or deletes created resources according to their attached deletion policy.
- ☒ **Delete all newly created resources**
Deletes created resources during a rollback regardless of their attached deletion policy.

What are Stack failure options?

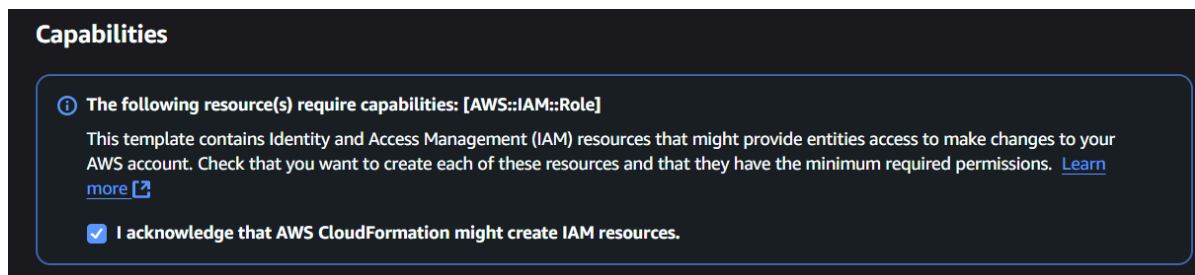
Stack failure options are your safety net when things don't go as planned. They determine what CloudFormation should do if it runs into an error while creating your resources:

- **Roll back all stack resources:** This is like having an "undo" button for your entire deployment. If anything fails, CloudFormation will automatically revert everything

back to how it was before you started. This prevents you from ending up with a half-built environment that might not work or cost you money unnecessarily.

- **Delete all newly created resources:** This makes sure CloudFormation cleans up after itself during a rollback. No resources left behind to surprise you on your bill next month!

- Scroll down and check this option under Capabilities:

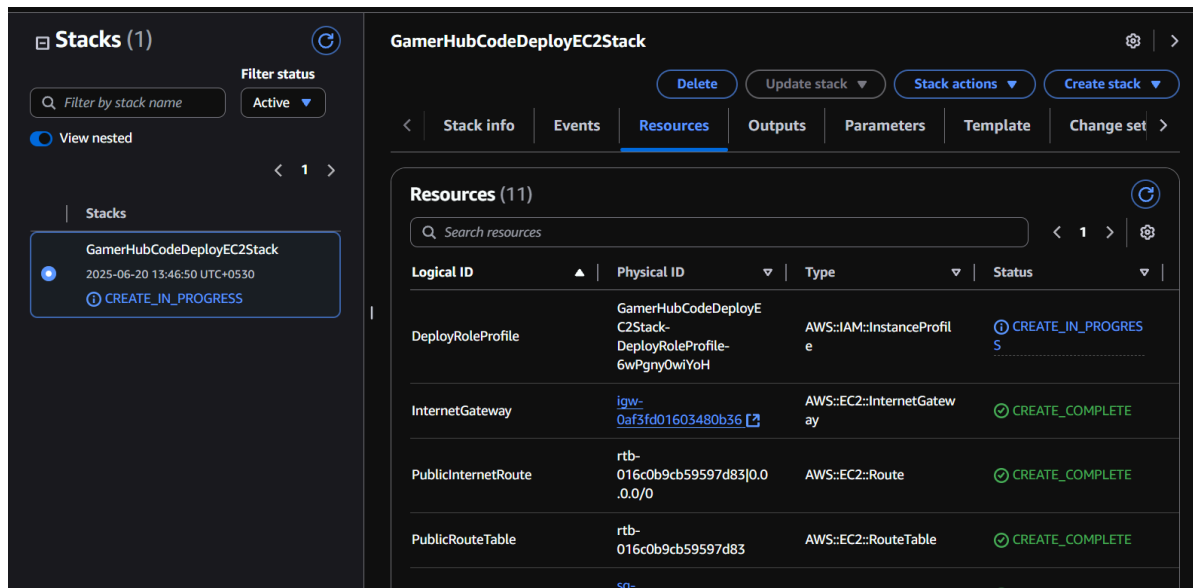


Why would CloudFormation create an IAM role?

IAM roles are like special visitor passes that AWS services can "wear" to temporarily access other services. In our case, our deployment EC2 instance will use a role to access files from S3.

Why do you think it'll need access to S3 (have you created an S3 bucket anywhere)? Because the deployment environment will need to use the build artifacts stored in your S3 bucket

- Click Next → Review details and Click Submit
- Go to resources tab to see the list of resources that are being created



- Go to Events tab:

What is a CloudFormation stack event?

Every time CloudFormation creates, updates, or deletes something, it records an **event** - "Starting to create EC2 instance," "Security group created successfully," "There isn't enough capacity to create a new VPC." These events give you visibility into exactly what's happening behind the scenes, which is super helpful for troubleshooting if something goes wrong.

- Wait for the stack's status to become CREATE_COMPLETE.
- If it shows : ROLLBACK_IN_PROGRESS or ROLLBACK_COMPLETE it means CloudFormation encountered an error while creating your resources. This is a common issue.

To fix this:

- Check the detailed error messages in the **Events** tab
- Make any needed corrections to template parameters or IAM permissions

- Try creating the stack again

STEP 2 : PREPARE DEPLOYMENT SCRIPTS AND APPSEC

NOTE: REFER GITHUB FOR THE SCRIPTS

- Go back to VSCode . Time to add some new files
- Create Scripts folder
- In Scripts/install_dependencies add the dependencies to be installed

`install_dependencies.sh` Explanation (Line by Line)

1. `#!/bin/bash`

This tells the system to run the script using the Bash shell interpreter.

2. `sudo yum update -y`

Updates all installed packages on the EC2 instance to their latest available versions using `yum` .

3. `sudo yum install -y python3 unzip`

Installs Python 3 and the `unzip` utility needed to unpack the zipped application artifact.

4. `cd /home/ec2-user`

Changes the working directory to the EC2 default user's home directory — where the zip file (`flask-app.zip`) is expected to be.

5. `unzip flask-app.zip`

Extracts the contents of the deployment zip file into the current directory. This typically includes your Python files, templates, static files, etc.

6. `pip3 install -r requirements.txt`

Installs all required Python packages listed in `requirements.txt` to run the Flask app.

systemd Service Block for Flask + Gunicorn

This part creates a Linux service using systemd so that your Flask app can run as a background service and restart automatically if it crashes.

1. `sudo cat << EOF > /etc/systemd/system/gamerhub.service`

Starts writing a new file at the given path — this will become the service definition for your Flask app.

2. `[Unit]`

Marks the beginning of the service's metadata section.

3. `Description=GamerHub Flask App`

A description for the service. This is shown when checking its status.

4. `After=network.target`

Ensures the service starts only after network connectivity is established (which is needed for a web app).

5. `[Service]`

Starts the section defining how the actual application should be run.

6. `User=ec2-user`

Specifies that the service should run under the `ec2-user` account, not as root.

7. `WorkingDirectory=/home/ec2-user`

Sets the directory from which the app will be run — where the app files were unzipped.

8. `ExecStart=/usr/local/bin/gunicorn -w 3 -b 0.0.0.0:80 app:app`

The command that starts the app:

- `gunicorn` is the production WSGI server.
- `w 3` launches 3 worker processes.
- `b 0.0.0.0:80` binds to all IP addresses on port 80.
- `app:app` means: from `app.py`, use the Flask app object named `app`.

9. `Restart=always`

Ensures the service will automatically restart if it crashes.

10. `[Install]`

Marks the start of the section that defines when to start this service.

11. `WantedBy=multi-user.target`

Configures the service to start during the standard boot process.

12. `EOF`

Ends the block writing to the service file.

Final systemd Setup

1. `sudo systemctl daemon-reexec`

Ensures any low-level changes to systemd itself are reloaded.

2. `sudo systemctl daemon-reload`

Reloads the systemd configuration to register the new `gamerhub` service.

3. `sudo systemctl enable gamerhub`

Enables the `gamerhub` service so it starts automatically when the EC2 instance boots.

4. `sudo systemctl start gamerhub`

Starts the Flask app right now as a systemd service.

- Inside the Scripts directory, create `start_server.sh`

`start_server.sh` – Explanation

1. `#!/bin/bash`

Tells the system to execute the script using the Bash shell.

2. `sudo systemctl daemon-reexec`

Reinitializes the systemd process itself — useful when making major system changes, though optional here.

3. `sudo systemctl daemon-reload`

Reloads the systemd configuration so that it recognizes any new or changed service files (like `gamerhub.service`).

4. `sudo systemctl start gamerhub`

Starts the `gamerhub` service immediately — this runs your Flask app using Gunicorn.

5. `sudo systemctl enable gamerhub`

Ensures the `gamerhub` service automatically starts on boot in future.

- Inside the Scripts directory , create stop_server.sh

`stop_server.sh` – Explanation

1. The script starts with the Bash shell.
2. It checks if a Gunicorn process (your Flask app server) is running using `pgrep -f gunicorn` .
3. If such a process is found (`if [[-n "$isExistApp"]]`), that means the service is running.
4. It prints `"Stopping GamerHub service..."` to the terminal.
5. Then, it gracefully stops the `gamerhub` systemd service using `sudo systemctl stop gamerhub` .
6. If the Gunicorn process is not found, it prints `"GamerHub service not running."` — indicating nothing needed to be stopped.

- (OPTIONAL CHECK): Flask logs check after deployment :

```
ssh ec2-user@<your-ec2-ip>
sudo journalctl -u gamerhub -f
```

- Create appspec.yml in root directory

`appspec.yml` – Explanation

version: 0.0

Specifies the version of the AppSpec file format. `0.0` is used for EC2/on-premise deployments.

os: linux

Defines the operating system of your deployment target. In this case, it's Linux (Amazon Linux 2).

files: section

- **source: flask-app.zip**

Tells CodeDeploy to take the file named `flask-app.zip` from the root of the build artifact (S3 uploaded by CodeBuild).

destination: /home/ec2-user/

Instructs CodeDeploy to copy `flask-app.zip` into the EC2 instance's `/home/ec2-user/` directory.

hooks: section

Hooks tell CodeDeploy what scripts to run during different phases of deployment.

BeforeInstall:

Runs *before* CodeDeploy installs your application files.

- **location: scripts/install_dependencies.sh**

Runs your script to install Python, unzip, and setup `gamerhub.service`.

ApplicationStop:

Runs if there's a previous version of your app already running. Stops it cleanly.

- **location: scripts/stop_server.sh**

Stops the Gunicorn (Flask) service if running.

ApplicationStart:

Runs *after* the new application files are copied to the instance.

- **location: scripts/start_server.sh**

Starts and enables the Gunicorn-based Flask service using systemd.

- Edit buildspec.yml :

```
artifacts:
  files:
    - flask-app.zip
    - appspec.yml
    - scripts/**/*
  discard-paths: no
```

Why edit?

The artifacts section we just edited is particularly important - it's telling CodeBuild: "After you build the app, make sure to include these additional files in the final package." We're adding our appspec.yml and scripts folder because CodeDeploy will need them to properly deploy the application. Without them, CodeDeploy wouldn't know what to do with our compiled code.

- Commit and push changes to GitHub repo

When git add. is done this might happen if doing in local windows environment:

```
PS D:\Projects\boardgame> git add .
```

```
warning: in the working copy of 'scripts/install_dependencies.sh', LF will be
replaced by CRLF the next time Git touches it
```

```
warning: in the working copy of 'scripts/start_server.sh', LF will be replaced
by CRLF the next time Git touches it
```

```
warning: in the working copy of 'scripts/stop_server.sh', LF will be
```

rwarning: in the working copy of 'scripts/start_server.sh', LF will be replaced by CRLF the next time Git touches it

Fix:

- Check in terminal with : `git config --get core.autocrlf`
- `true` → **CRLF on checkout, LF on commit** (default on Windows — causes the warning you're seeing)
- `input` → LF stays LF (ideal for cross-platform shell scripts!)
- `false` → Git does no auto conversion (risky if team has mixed OSes)
- Terminal output:

```
PS D:\Projects\boardgame> git config --get core.autocrlf
true
PS D:\Projects\boardgame>
```

Fix:

- To avoid Git warning about this **and prevent any CRLF sneak-ins:**
`git config core.autocrlf input`
This will:
 - Keep LF endings as LF
 - Stop Git from threatening to convert it in the future
- Then force Git to "re-check" your files:
`git add --renormalize .`

STEP 3 : SET UP CODEDEPLOY

- Go to CodeDeploy console on AWS
- Go to Applications → Create application

What is a CodeDeploy application?

A CodeDeploy application is like the main folder for your deployment project. It doesn't do much on its own, but it helps you organize everything related to deploying one application.

In more technical, AWS terms, a CodeDeploy application is a namespace or container that groups deployment configurations, deployment groups, and revisions for a specific application. Having separate CodeDeploy applications helps you manage multiple applications without mixing up their deployment resources.

- Enter application name like : gamerhub-cicd-application
- Choose EC2/On-premises on Compute platform

What are CodeDeploy compute platforms?

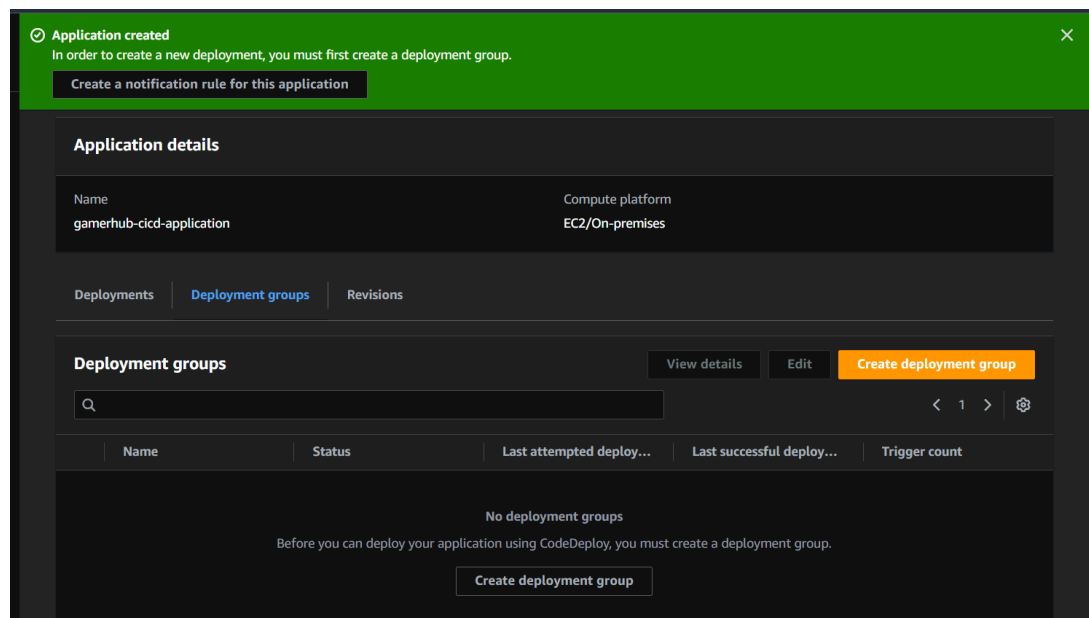
CodeDeploy's compute platforms are basically the different types of environments where your application can live:

- **EC2/On-premises:** This is for traditional server-based applications - like what we're doing in this project. Your app runs on actual servers (either in AWS or in your own data center).

- **AWS Lambda:** This is for serverless applications where you don't manage any servers. Your code just runs when triggered, and AWS handles all the infrastructure.
- **Amazon ECS:** This is for containerized applications running in Docker containers managed by Amazon's Elastic Container Service.

Choosing the right platform depends on how your application is built. Each has its own advantages for different types of applications.

- Select Create application



(As the message says. Deployment group is to be created next)

- Select Create deployment group

What is a CodeDeploy deployment group?

A deployment group is a collection of EC2 instances that are grouped to deploy something together.

It's also where you plan out exactly **where** and **how** your application gets deployed on this group of instances. In other words, it's where you tell CodeDeploy "let's deploy this app to these specific servers, using this deployment pattern, with these load balancing settings, and handle failures this way."

The power of deployment groups is that you can have multiple groups within the same application - maybe one for your test environment, another for staging, and a third for production. Each can have different settings and target different sets of servers, but they all deploy the same core application. This adds to the (many) reasons why CI/CD tools are so powerful - in this case, CodeDeploy saves you the time it'd take to manually deploy the same app to each instance in each environment.

- Give a name like : gamerhub-cicd-deploymentgroup
- Need to create a new service role

Why does CodeDeploy need IAM roles?

CodeDeploy needs IAM roles to get permissions to access and manage AWS resources on your behalf. These permissions let CodeDeploy do things like:

- Accessing EC2 instances to deploy applications.
 - Reading application artifacts from S3 buckets.
 - Updating Auto Scaling groups.
 - Write CloudWatch logs about what it's doing
- Go to IAM console → Roles → Create roll → Choose AWS Services as Trusted entity type → Select CodeDeploy for Service or use case → Click Next :

Trusted entity type

- ☒ **AWS service**
Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- ☐ **AWS account**
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- ☐ **Web identity**
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- ☐ **SAML 2.0 federation**
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- ☐ **Custom trust policy**
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case

CodeDeploy

Choose a use case for the specified service.

Use case

- ☒ **CodeDeploy**
Allows CodeDeploy to call AWS services such as Auto Scaling on your behalf.
- ☐ **CodeDeploy for Lambda**
Allows CodeDeploy to route traffic to a new version of an AWS Lambda function version on your behalf.
- ☐ **CodeDeploy - ECS**
Allows CodeDeploy to read S3 objects, invoke Lambda functions, publish to SNS topics, and update ECS services on your behalf.

- When the dropdown is clicked to inspect the policy , a lot of permissions are listed. When looking at EC2's permissions within the policy:

What permissions does AWSCodeDeployRole policy include?

The AWSCodeDeployRole policy lets CodeDeploy work with your EC2 instances. It includes permissions for:

- **Auto Scaling**, so CodeDeploy can handle deployments when you're automatically scaling instances up or down

- **EC2**, so CodeDeploy can interact with your instances - tag them, query them, and deploy to them
- **Elastic Load Balancing**, so CodeDeploy can temporarily remove instances from load balancers during deployment
- **CloudWatch**, so CodeDeploy can send logs and metrics so you can monitor what's happening
- **S3**, so CodeDeploy can access the build artifacts stored in S3 buckets

Not that these permissions are carefully scoped to only what CodeDeploy actually needs to do its job, instead of everything in your AWS account. This is a best practice in security called the "principle of least privilege."

- Give a name like CodeDeploy-Role and create the role
- Head back to CodeDeploy's deployment group creation tab and refresh it. Now enter group name again and now the option to select Service role is available. Select the newly created CodeDeploy-Role
- Choose In-place as Deployment type

What is the difference between In-place and Blue/Green deployments?

- **In-place deployment:** Updates the application on existing instances. During the deployment, the application on the instances is stopped, the new version is installed, and then the application is restarted. This can cause a brief downtime during deployment.

- **Blue/green deployment:** Creates a new, separate environment (the "green" environment) to deploy the new application version. Once the new version is verified in the green environment, traffic is switched from the old environment (the "blue" environment) to the new one. This minimizes downtime and allows for quick rollbacks.

For most production applications, blue/green is preferred because of the zero downtime and easy rollback, but for this learning project, in-place is simpler and more cost-effective

- Under Environment configuration → Select Amazon EC2 instances
- Select role as the key and webserver as the value in Tag group 1
- NOTE:

Can't find any matching instances?


If CodeDeploy isn't finding your EC2 instance, The issue is usually related to the tags we're using to identify the instance.

When you see "0 instances found" instead of "1 unique matched instance," here's what to check:

- Verify that you entered the exact tag key role and value webserver
- Double-check for any typos or extra spaces
- Check your CloudFormation outputs to confirm the EC2 instance was tagged correctly

- Select [Click here for details](#) to view the matched instance.
- You'll see an EC2 instance in new window. That's the EC2 instance we launched from our CloudFormation template. This confirms to us that the web app will be deployed onto that instance.
- Head back to deployment group tab → For Agent configuration , leave it as it is

Agent configuration with AWS Systems Manager [Info](#)


Complete the required prerequisites before AWS Systems Manager can install the CodeDeploy Agent.
 Make sure the AWS Systems Manager Agent is installed on all instances and attach the required IAM policies to them. [Learn more](#)

Install AWS CodeDeploy Agent

☐ Never
 ☐ Only once
 ☒ Now and schedule updates

Basic scheduler

Cron expression

14

Days ▼

What is CodeDeploy Agent?

The CodeDeploy Agent is a software that lets your EC2 instance communicate with CodeDeploy.

Whenever you initiate a deployment, it's the CodeDeploy Agent that receives the deployment instructions (i.e. bash scripts) from CodeDeploy and carries them out on the EC2 instance.

Setting it to update every 14 days simply makes sure your agent software is always up to date to the latest version that AWS released.

- For deployment settings , leave it as `CodeDeployDefault.AllAtOnce`

What are deployment settings?

Deployment settings help you control how quickly you'd like to roll out your application.

CodeDeployDefault.AllAtOnce deploys the application to all instances in the deployment group at the same time. It's the fastest option, but also the riskiest - if something goes wrong, all your instances could be affected at once.

What deployment setting to choose?

For production environments, you might choose more conservative options like `OneAtATime` (update one instance, make sure it works, then move to the next) or `HalfAtATime` (update 50% of instances, verify, then do the rest). These slower approaches reduce risk by limiting the blast radius of any potential issues.

For production systems with hundreds of instances, these settings become crucial. Imagine updating your banking app on all servers simultaneously vs. trying it on 10% first to make sure customers can still access their accounts! But since there is only one instance in the project, the cautious approach doesn't offer much benefit.

Using `AllAtOnce` in this project because there is only one instance .

- **Deselect** Enable load balancing

What is load balancing?

Load balancing directs visitors to whichever server is least busy at the moment. Instead of everyone lining up at one server (which could get overwhelmed), traffic is distributed across multiple servers. When a visitor tries to access your website, they actually connect to the load balancer first. The load balancer then decides which server should handle this particular request - typically choosing the least busy one.

Not using load balancing in this project because there is only one server - so there's nothing to balance between! But in a production environment, there is need to almost always use it to improve the availability and performance of the application.

- Select Create deployment group

The screenshot shows the AWS CodeDeploy console interface. At the top, a green banner indicates 'Success: Deployment group created'. Below this, the breadcrumb navigation shows the path: Developer Tools > CodeDeploy > Applications > gamerhub-cicd-application > gamerhub-cicd-deploymentgroup. The main heading is 'gamerhub-cicd-deploymentgroup', with 'Edit', 'Delete', and 'Create deployment' buttons to its right. The 'Deployment group details' section contains a table with the following information:

Deployment group name	Application name	Compute platform
gamerhub-cicd-deploymentgroup	gamerhub-cicd-application	EC2/On-premises
Deployment type	Service role ARN	Deployment configuration
In-place	arn:aws:iam::975050195505:role/CodeDeploy-Role	CodeDeployDefault.AllAtOnce
Rollback enabled	Agent update scheduler	
False	Learn to schedule update in AWS Systems Manager 🔗	

Below the details table, the 'Environment configuration: Amazon EC2 instances' section is visible, showing a table with 'Key' and 'Value' columns.

STEP 4 : CREATE AND VERIFY DEPLOYMENT

- In the deployment group details page that was created in previous step, click Create deployment

What is a CodeDeploy deployment?

A CodeDeploy **deployment** represents a single update to your application, with its own unique ID and history. When you create a deployment, you're telling CodeDeploy:

- Which version of the application to deploy (the revision)
- Where to deploy it (the deployment group)
- How to deploy it (the deployment settings)

CodeDeploy then orchestrates the entire process - stopping services, copying files, running scripts, starting services - and keeps track of whether it succeeds or fails. You can monitor it happening in real-time and see a detailed log of each step.

- Some options are auto selected. Go to S3 → Copy the artifact's S3 URI → Paste the URI into the Revision location field in CodeDeploy → Select .zip as file type

What is a revision location? Why did we use our WAR/zip file?

The **revision location** is the place where CodeDeploy looks to find the application's build artifacts. In this project, the S3 bucket stores the ZIP file,

so CodeDeploy knows where to find the latest version of the web app it's deploying to the deployment EC2 instances

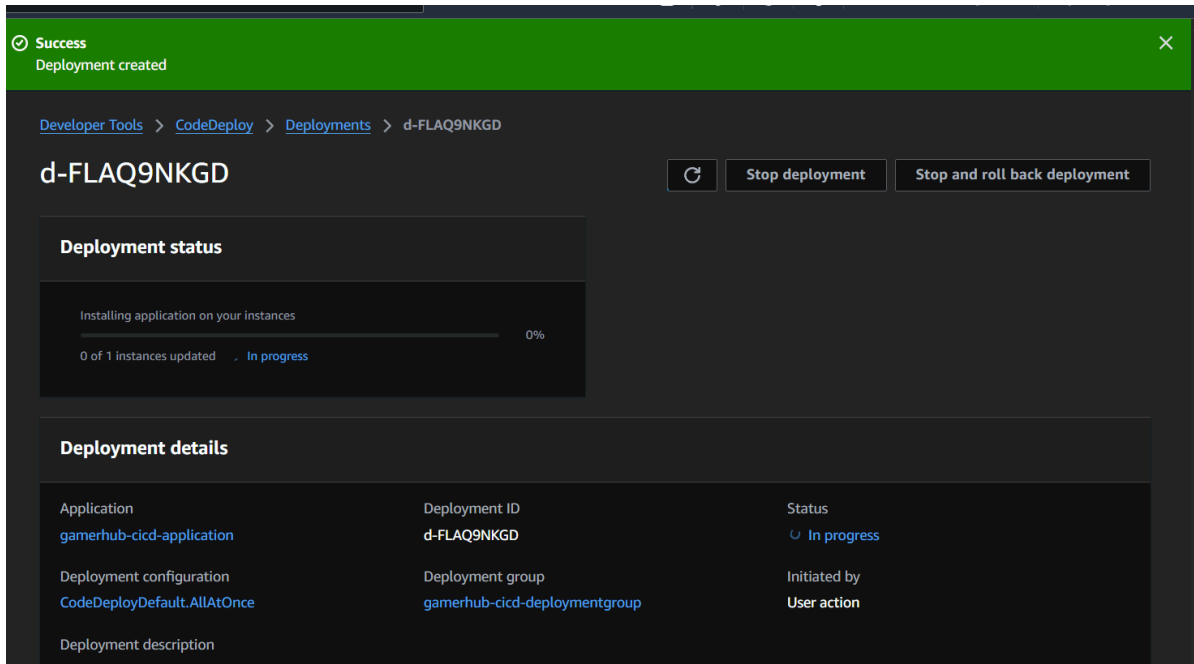
- Leave Additional deployment behaviors settings and Deployment group overrides as default

What are Additional deployment behaviors settings and Deployment group overrides?

These settings give extra flexibility when in need to handle special cases without changing the standard deployment settings:

- **Additional deployment behaviors settings** let to configure options like how to handle file permissions during deployment or whether to immediately allow traffic to new instances.
- **Deployment group overrides** let to override deployment group settings for a specific deployment - user may normally deploy one instance at a time (safe but slow), but for a critical security fix, there is a need to override that and deploy to all instances at once (faster but riskier).

- Click on Create deployment



- Scroll down to Deployment lifecycle events and click on View events to see the lifecycle events progressing, such as BeforeInstall, ApplicationStart, etc. These are the events defined in appspec.yml

Event	Duration	Status	Error code	Start time	End time
ApplicationStop	0 seconds	✔ Succeeded	-	Jun 20, 2025 3:40 PM (UTC+5:30)	Jun 20, 2025 3:40 PM (UTC+5:30)
DownloadBundle	1 second	✔ Succeeded	-	Jun 20, 2025 3:40 PM (UTC+5:30)	Jun 20, 2025 3:40 PM (UTC+5:30)
BeforeInstall	-	⌚ Pending	-	-	-
Install	-	⌚ Pending	-	-	-
AfterInstall	-	⌚ Pending	-	-	-
ApplicationStart	-	⌚ Pending	-	-	-
ValidateService	-	⌚ Pending	-	-	-

- Notice build failure:

FLAQ9NKGD					
Event	Duration	Status	Error code	Start time	End time
ApplicationStop	0 seconds	✔ Succeeded	-	Jun 20, 2025 3:40 PM (UTC+5:30)	Jun 20, 2025 3:40 PM (UTC+5:30)
DownloadBundle	1 second	✔ Succeeded	-	Jun 20, 2025 3:40 PM (UTC+5:30)	Jun 20, 2025 3:40 PM (UTC+5:30)
BeforeInstall	0 seconds	✘ Failed	UnknownError	Jun 20, 2025 3:45 PM (UTC+5:30)	Jun 20, 2025 3:45 PM (UTC+5:30)
Install	-	ⓘ Skipped	-	-	-
AfterInstall	-	ⓘ Skipped	-	-	-
ApplicationStart	-	ⓘ Skipped	-	-	-
ValidateService	-	ⓘ Skipped	-	-	-

- CodeDeploy is deploying your web app by grabbing the latest **build artifact** from your S3 bucket.
- The last time running a build was **before** appspec and the deployment scripts were added.
- Because of this, your deployment instance isn't getting **any** of the scripts that were written - causing the error now.
- Head back to your CodeBuild build project, and **rebuild** the project.
- Once second build is a success, return to CodeDeploy and retry the deployment.

And NOW!

Developer Tools > CodeDeploy > Deployments > d-QH30YDKGD

d-QH30YDKGD
↻
Copy deployment
Retry deployment

Deployment status

Installing application on your instances

100%

1 of 1 instances updated ✔ Succeeded

Deployment details

Application gamerhub-cicd-application	Deployment ID d-QH30YDKGD	Status ✔ Succeeded
Deployment configuration CodeDeployDefault.AllAtOnce	Deployment group gamerhub-cicd-deploymentgroup	Initiated by User action
Deployment description -		

NOTE: THE STEPS ARE MENTIONED CLEARLY. IF THEY DON'T SEEM TO WORK
REFER THE Errors and Fixes DOCUMENT

_____PHASE 4
COMPLETED_____