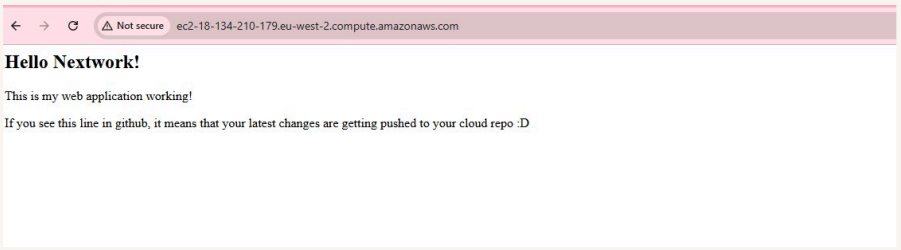
nextwork.org

Deploy a Web App with CodeDeploy

MA Marzena Pugo



MA
Marzena Pugo
NextWork Student
nextwork.org

Introducing Today's Project!

In this project I will demonstrate how to use CodeDeploy to deploy a web app. I am doing this project to learn how deployment works, and how it can be automated using a combination of CodeDeploy and deployment scripts. By the end of this project I will see a live website.

Key tools and concepts

Services I used were CodeBuild, CodeDeploy, CodeArtifact, CloudFormation, IAM , EC2, S3, CodeConnection,Vs Code and Github. Key concepts I learnt include deployment, deployments groups, deployment scripts, appspec.yml and the importance of rebuilding my project before II deploy.

Project reflection

This project took me approximately three hours including troubleshooting. The most challlenging part was understanding the separation of the deployment instance (launching a deployment environment using AWS CloudFormation) from the development instance. It was most rewarding to see the deployed app.

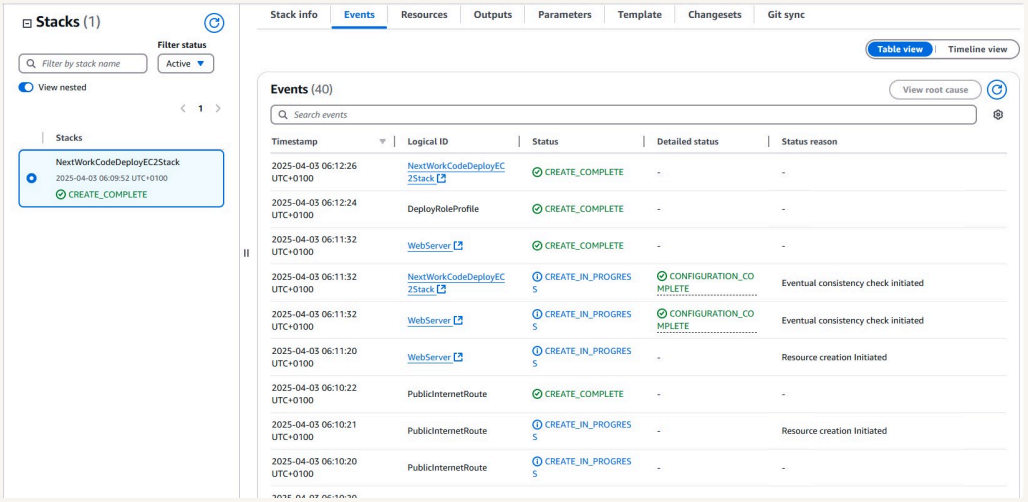
This project is part five of a series of DevOps projects where I'm building a CI/CD pipeline! I'll be working on the next project.

Deployment Environment

To set up for CodeDeploy, I launched an EC2 instance and VPC because they will make up my production environment. I need to seperate my development environment (where I am writing the code)with the production environment where i am deploying the web app, so that the code that I am writing as developers don't get shown to users until I am ready to push them into production.

Instead of launching these resources manually, I used Cloud Formation. When I need to delete these resources created by a CloudFormation template we can simply delete the CloudFormation task. This will automatically delete all the resources that are inside that stack.

Other resources created in this template include networking resources like VPCs, Internet gateways, route tables and subnets. They're also in the template because production environment has specific requirements arround traffic that it should enable and block out. Setting up all of the networking resources that will help the deployment work is a common practice.


Stacks (1)
Filter status
Filter by stack name
Active
View nested
1
Stacks
NextWorkCodeDeployEC2Stack
2025-04-03 06:09:52 UTC+0100
CREATE_COMPLETE
Stack info
Events
Resources
Outputs
Parameters
Template
Changesets
Git sync
Table view
Timeline view
Events (40)
View root cause
Search events
Timestamp
Logical ID
Status
Detailed status
Status reason
2025-04-03 06:12:26 UTC+0100
NextWorkCodeDeployEC2Stack
CREATE_COMPLETE
-
-
2025-04-03 06:12:24 UTC+0100
DeployRoleProfile
CREATE_COMPLETE
-
-
2025-04-03 06:11:32 UTC+0100
WebServer
CREATE_COMPLETE
-
-
2025-04-03 06:11:32 UTC+0100
NextWorkCodeDeployEC2Stack
CREATE_IN_PROGRESS
CONFIGURATION_COMPLETE
Eventual consistency check initiated
2025-04-03 06:11:32 UTC+0100
WebServer
CREATE_IN_PROGRESS
CONFIGURATION_COMPLETE
Eventual consistency check initiated
2025-04-03 06:11:20 UTC+0100
WebServer
CREATE_IN_PROGRESS
-
Resource creation Initiated
2025-04-03 06:10:22 UTC+0100
PublicInternetRoute
CREATE_COMPLETE
-
-
2025-04-03 06:10:21 UTC+0100
PublicInternetRoute
CREATE_IN_PROGRESS
-
Resource creation Initiated
2025-04-03 06:10:20 UTC+0100
PublicInternetRoute
CREATE_IN_PROGRESS
-
-

Deployment Scripts

Scripts are mini programs that help me automate the writing of commands. To set up CodeDeploy, I also wrote scripts to automate deployment commands (these are other commands that my deployment EC2 instance needs to run in order to host my web app).

install_dependencies.sh will help my EC2 instance install all dependencies they need to host the web app like Tomcat js web server.

start_server.sh is another script that's all about starting up two servers that I'm using in my EC2 instance. One it starts Tomcat (responsible for running my Java app) and two it starts Apache (the web server responsible for handling web traffic and passing requests to Tomcat).

stop_server.sh is a script that stops my Apache and Tomcat servers when they are not longer needed to serve the web app to a user.

appspec.yml

Then, I wrote an appspec.yml file to give CodeDeploy the instructions to deploy my web app. The key sections in appspec.yml are BeforeInstall (use the install _dependencies.sh script to install dependencies like Tomcat), ApplicationStart (use the start_server.sh script) and ApplicationStop (use stop_server script).

I also updated buildspec.yml because it contains step-by-step instructions that tell CodeBuild exactly how to turn your raw code into a deployable package. In my project, it tells CodeBuild how to compile the Java application, run any tests, and then package up everything needed for deployment. The artifacts section I 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." I'm adding my 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 my compiled code!

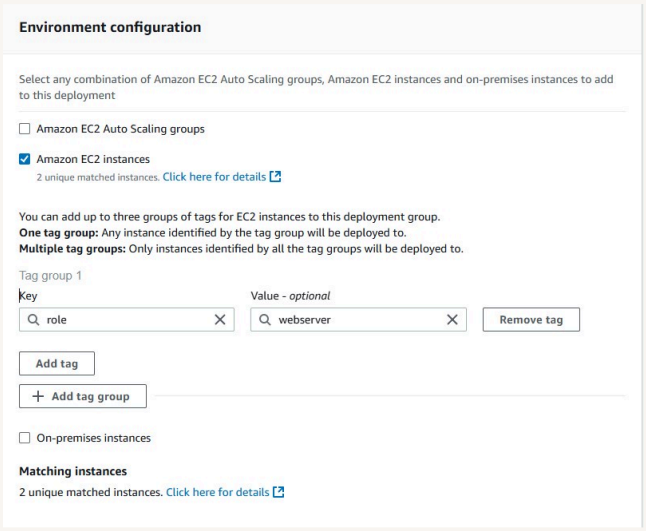

stop_server.sh U
appspec.yml U
appspec.yml
version: 0.0
os: linux
files:
- source: /target/nextwork-web-project.war
destination: /usr/share/tomcat/webapps/
hooks:
BeforeInstall:
- location: scripts/install_dependencies.sh
timeout: 300
runas: root
ApplicationStart:
- location: scripts/start_server.sh
timeout: 300
runas: root
ApplicationStop:
- location: scripts/stop_server.sh
timeout: 300
runas: root

Setting Up CodeDeploy

A deployment group is a group of EC2 instances that you can deploy to. A CodeDeploy application is a collection of settings that determine how you want to deploy your web app. A CodeDeploy application is simply a folder that holds together all the deployment groups for the same app.

To set up a deployment group, you also need to create an IAM role to give CodeDeploy the permission to access the EC2 instances that it needs to coordinate. Otherwise CodeDeploy does not have access to EC2 = not being able to give your deployment instances the instructions for deploying the web app.

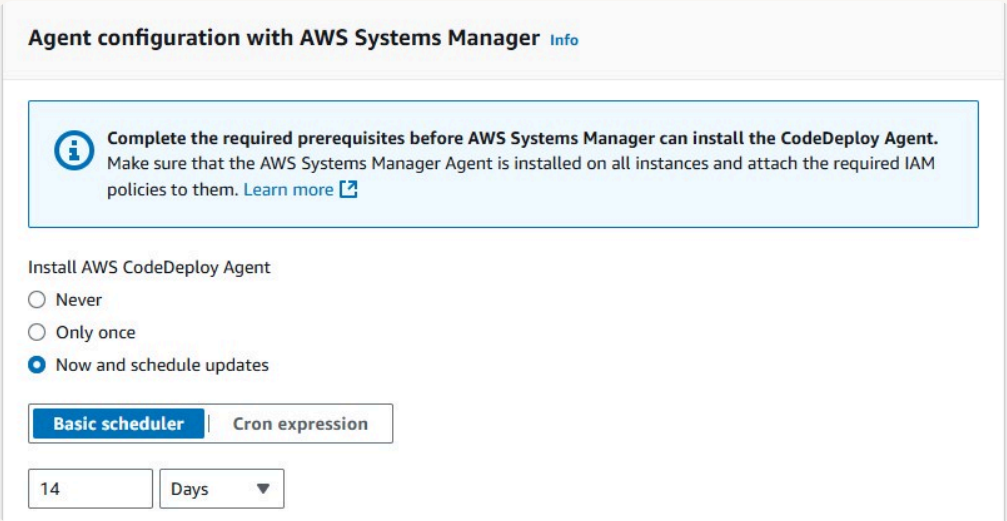
Tags are helpful for identifying instances that will be deploying the web app. I used the tag "role" "webserver" to automatically much the EC2 instance I deploy with my CloudFormation template. Now in the future when I have other EC2 instances I want to also be in this deployment group, I can simply tag that new instance with the same "role" "webserver" tag. They will be automatically added to the same deployment group (a good time server).


Environment configuration
Select any combination of Amazon EC2 Auto Scaling groups, Amazon EC2 instances and on-premises instances to add to this deployment
Amazon EC2 Auto Scaling groups
Amazon EC2 instances
2 unique matched instances. Click here for details
You can add up to three groups of tags for EC2 instances to this deployment group.
One tag group: Any instance identified by the tag group will be deployed to.
Multiple tag groups: Only instances identified by all the tag groups will be deployed to.
Tag group 1
Key
role
Value - optional
webserver
Remove tag
Add tag
Add tag group
On-premises instances
Matching instances
2 unique matched instances. Click here for details

Deployment configurations

Another key settings is the deployment configuration, which affects the production environment. I used CodeDeployDefault. AllAtOnce in this project because I only have one instance and I'm learning - speed is more important than caution here! 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.

I order to connect the deployment instance with CodeDeploy agent is also set up to receive instructions from CodeDeploy and make sure that the commands in appsec.yml are run.

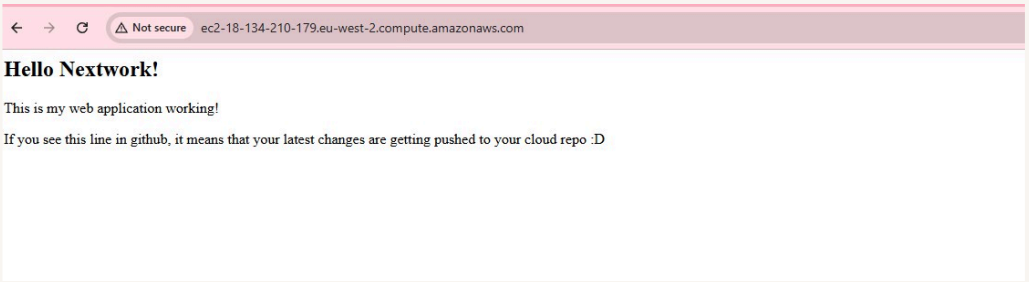


Success!

A CodeDeploy deployment is a specific update to my application that I am deploying to my users. The different between a deployment and a deployment group is that the group is like a settings file and a deploymment itself is like a specific update I roll out using that settings file.

I had to configure a revision location, which means when my web app's WAR file (the compressed file ready to be deployed) lives. My revision location is the S3 bucket that I've created and linked with CodeBuild (build artifacts automatically go into that bucket).

To check that the deployment was a success, I visited the IPv4 DNS of my EC2 instance from the EC2 console. In a web browser and I saw my application to become fully accessible after deployment.

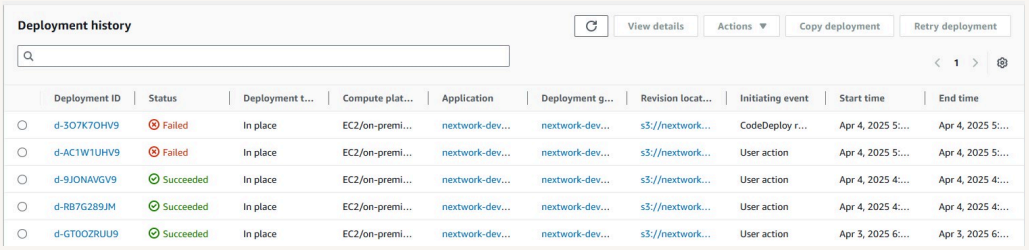

Not secure
ec2-18-134-210-179.eu-west-2.compute.amazonaws.com
Hello Nextwork!
This is my web application working!
If you see this line in github, it means that your latest changes are getting pushed to your cloud repo :D

Disaster Recovery

In a project extension, I decided to Intentionally create a broken deployment. The intentional error I created was scripts folder and open the stop_server.sh file. I rewrote the entire stop_server.sh file. I introduced an error by deleting everything, and having only one line that uses systemctll instead of systemctl. # Your intentional error sudo systemctll stop httpd.service # Misspelled command will now cause script to exit with non-zero status # To be extra sure, we can also add an explicit exit code exit 1 This will cause the deployment to fail because of a command not found error.

I also enabled rollbacks with this deployment. A rollback is like having an "undo" button for your software updates. When you deploy a new version of your application and something goes wrong, a rollback lets you quickly go back to the previous version that was working fine. For websites and apps, rollbacks are super important because they help minimize downtime when things break. Instead of spending hours trying to fix a broken update while users are frustrated, you can rollback to the working version in minutes, then fix the problems at your own pace.

There were actually two failed deployments that failed. The deployment at the top of the list is especially interesting - its initiating event is CodeDeploy rollback. This means even the automatic rollback failed In production environments, true automated rollbacks are often implemented with more advanced tools. AWS CodePipeline can be configured to automatically roll back to the last successful deployment when a failure is detected, which eliminates the need for manual intervention.


Deployment history
View details
Actions
Copy deployment
Retry deployment
Deployment ID
Status
Deployment t...
Compute plat...
Application
Deployment g...
Revision locat...
Initiating event
Start time
End time
d-3O7K7OHV9
Failed
In place
EC2/on-premi...
nextwork-dev...
nextwork-dev...
s3://nextwork...
CodeDeploy r...
Apr 4, 2025 5:...
Apr 4, 2025 5:...
d-AC1W1UHV9
Failed
In place
EC2/on-premi...
nextwork-dev...
nextwork-dev...
s3://nextwork...
User action
Apr 4, 2025 5:...
Apr 4, 2025 5:...
d-9JONAVGV9
Succeeded
In place
EC2/on-premi...
nextwork-dev...
nextwork-dev...
s3://nextwork...
User action
Apr 4, 2025 4:...
Apr 4, 2025 4:...
d-RB7G289JM
Succeeded
In place
EC2/on-premi...
nextwork-dev...
nextwork-dev...
s3://nextwork...
User action
Apr 4, 2025 4:...
Apr 4, 2025 4:...
d-GT0OZRUU9
Succeeded
In place
EC2/on-premi...
nextwork-dev...
nextwork-dev...
s3://nextwork...
User action
Apr 3, 2025 6:...
Apr 3, 2025 6:...

The place to learn & showcase your skills

Check out nextwork.org for more projects