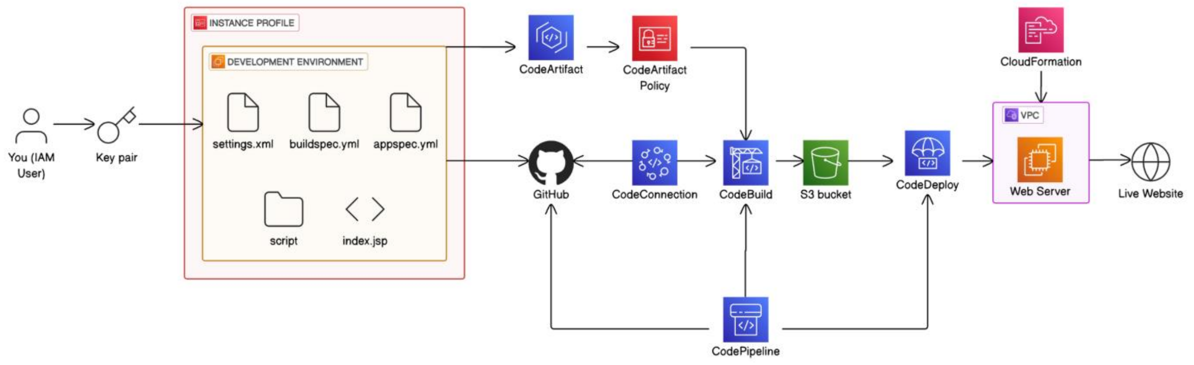
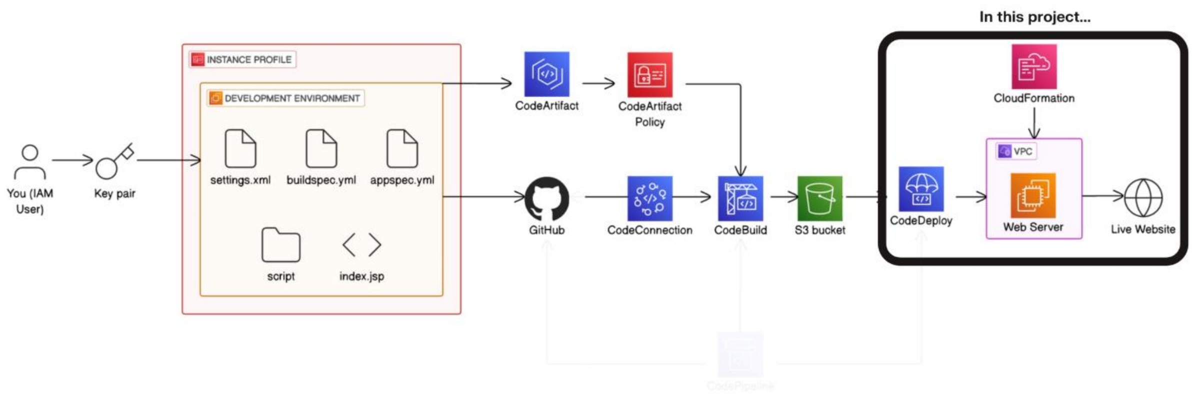
Deploy a Web App with CodeDeploy



Introducing Today's Project!

In this project, I will demonstrate deploying a web app using AWS CloudFormation, automation scripts, and CodeDeploy. I'm doing this to learn automated deployments, infrastructure as code, and how to roll back safely during failures.



Key tools and concepts

Services I used were CloudFormation, EC2, VPC, Subnet, IGW, Route Table, Security Group, IAM, and SSM. Key concepts I learnt include IaC, public subnet setup, secure access via SG, dynamic AMI with SSM, and EC2 role-based permissions.

Project reflection

This project took me approximately 1 day to complete. The most challenging part was configuring the VPC networking and routing correctly for internet access. It was most rewarding to see the EC2 instance launch successfully with secure HTTP access.

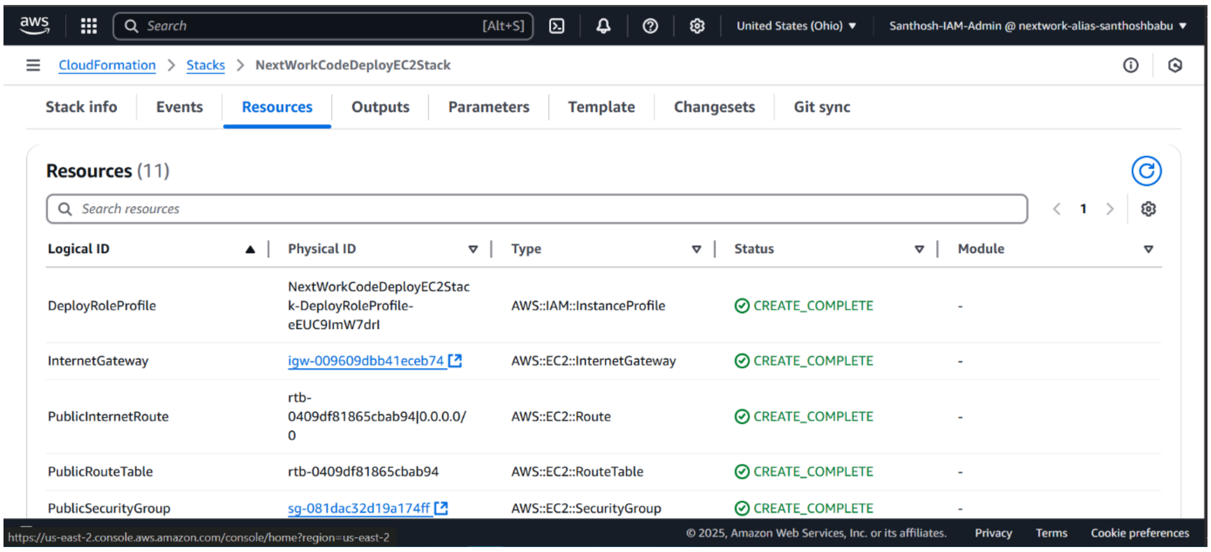
This project is part five of a series of DevOps projects where I'm building a CI/CD pipeline! I'll be working on the next project soon, focusing on automating deployments with Jenkins or AWS CodePipeline for continuous integration and delivery.

Deployment Environment

To set up for CodeDeploy, I launched an EC2 instance and VPC because the EC2 instance serves as the deployment target for my app, and the VPC provides the necessary networking environment to securely manage and control access to the instance.

Instead of launching these resources manually, I used AWS CloudFormation to deploy my EC2 instance and VPC. When I need to delete these resources, I can simply delete the CloudFormation stack, which automatically removes all associated components.

Other resources created in this template include a Security Group, IAM Role, Instance Profile, and Key Pair. They're also in the template because they provide secure access, necessary permissions, and enable integration with CodeDeploy.


aws
Search [Alt+S]
United States (Ohio)
Santhosh-IAM-Admin @ nextwork-alias-santhoshbabu
CloudFormation > Stacks > NextWorkCodeDeployEC2Stack
Stack info
Events
Resources
Outputs
Parameters
Template
Changesets
Git sync
Search resources
1
https://us-east-2.console.aws.amazon.com/console/home?region=us-east-2
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Resources (11)

Logical ID	Physical ID	Type	Status	Module
DeployRoleProfile	NextWorkCodeDeployEC2Stack-DeployRoleProfile-eEUC9ImW7drI	AWS::IAM::InstanceProfile	CREATE_COMPLETE	-
InternetGateway	igw-009609dbb41eceb74	AWS::EC2::InternetGateway	CREATE_COMPLETE	-
PublicInternetRoute	rtb-0409df81865cbab94|0.0.0.0/0	AWS::EC2::Route	CREATE_COMPLETE	-
PublicRouteTable	rtb-0409df81865cbab94	AWS::EC2::RouteTable	CREATE_COMPLETE	-
PublicSecurityGroup	sg-081dac32d19a174ff	AWS::EC2::SecurityGroup	CREATE_COMPLETE	-

Deployment Scripts

Scripts are executable files that automate tasks. To set up CodeDeploy, I also wrote scripts to install dependencies, configure the environment, and manage the start and stop of the web application, ensuring a smooth deployment process.

install_dependencies.sh will install required packages like Tomcat and Apache HTTP Server, set up necessary configurations, and ensure the server is ready for deployment. It automates the installation of dependencies before the application runs.

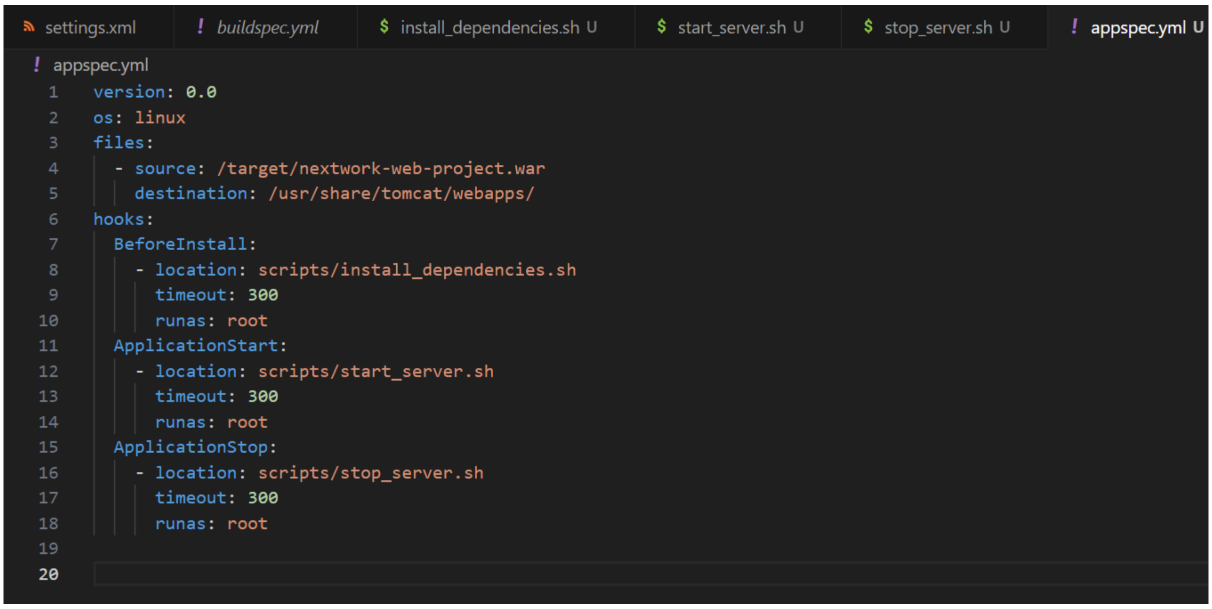
start_server.sh will start the Tomcat and Apache HTTP Server services, ensuring both are running. It also enables these services to start automatically on system boot, ensuring the application is accessible and the server is properly configured.

stop_server.sh will check if the Apache HTTP Server (httpd) and Tomcat services are running. If either service is running, it stops them using systemctl stop. This ensures that both services are properly stopped before any updates or changes are made.

appspec.yml

Then, I wrote an appspec.yml file to define the deployment process in AWS CodeDeploy. The key sections are files (specifying source and destination) and hooks (defining lifecycle events like installing dependencies and managing server start/stop).

I also updated buildspec.yml because it is used to define the build and packaging process for CodeDeploy. I configured it to include the necessary steps to package deployment files (like the WAR file) into the build artifact for deployment.



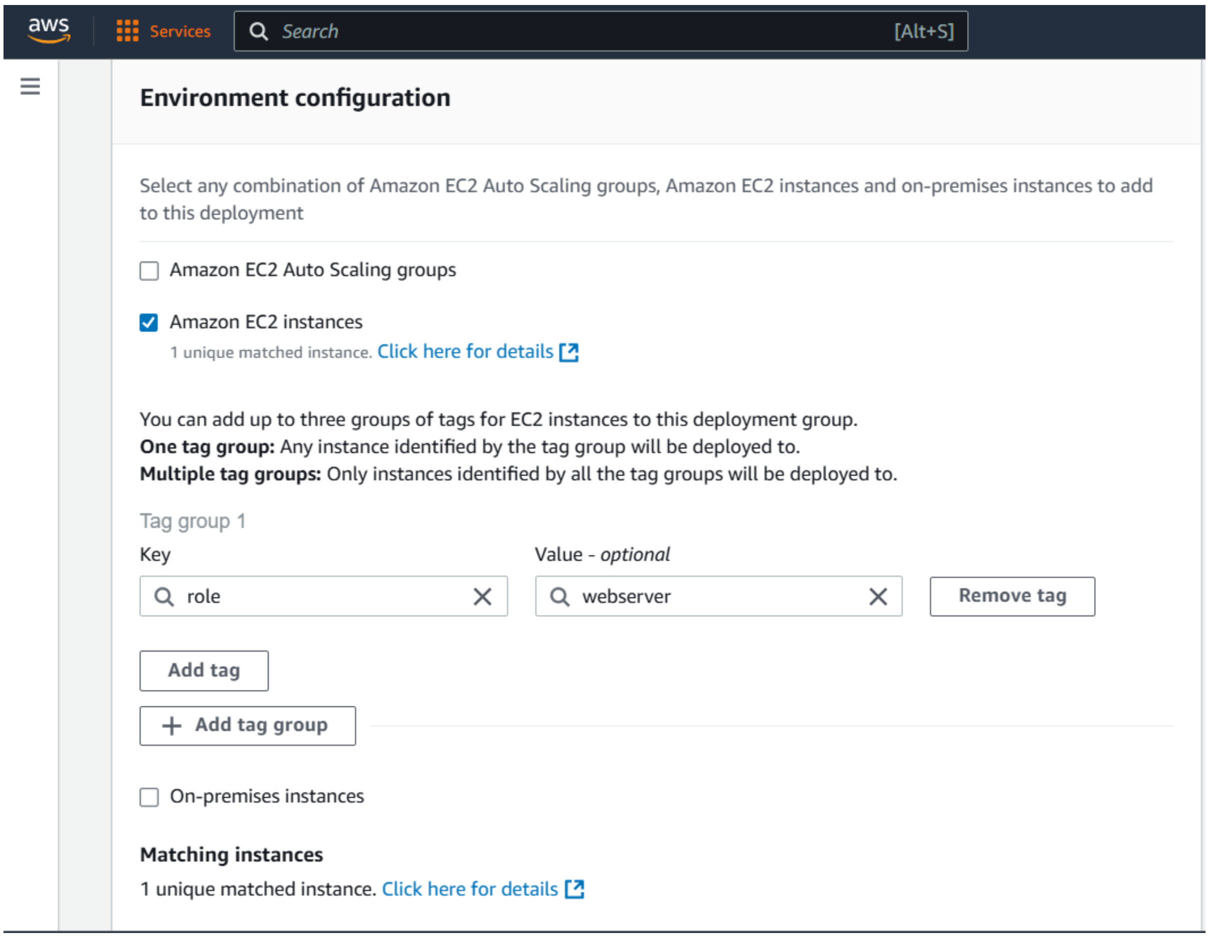
```yaml
version: 0.0
os: linux
files:
  - source: /target/nextwork-web-project.war
    destination: /usr/share/tomcat/webapps/
hooks:
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_server.sh
    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

```

Setting Up CodeDeploy

A deployment group is a set of target instances and deployment settings for a specific environment, like staging or production. A CodeDeploy application is the overall unit that represents what you're deploying, such as a web app or service.

To set up a deployment group, you also need to create an IAM role to give CodeDeploy the necessary permissions to access EC2 instances, pull artifacts, run scripts, and manage the deployment process securely on your behalf.

Tags are helpful for organizing and identifying AWS resources. I used the tag `CodeDeployEC2` to target specific EC2 instances for deployment, allowing CodeDeploy to easily find and deploy the application only to the intended instances.



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

1 unique matched instance. 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	Value - *optional*
role	webserver

☐ On-premises instances

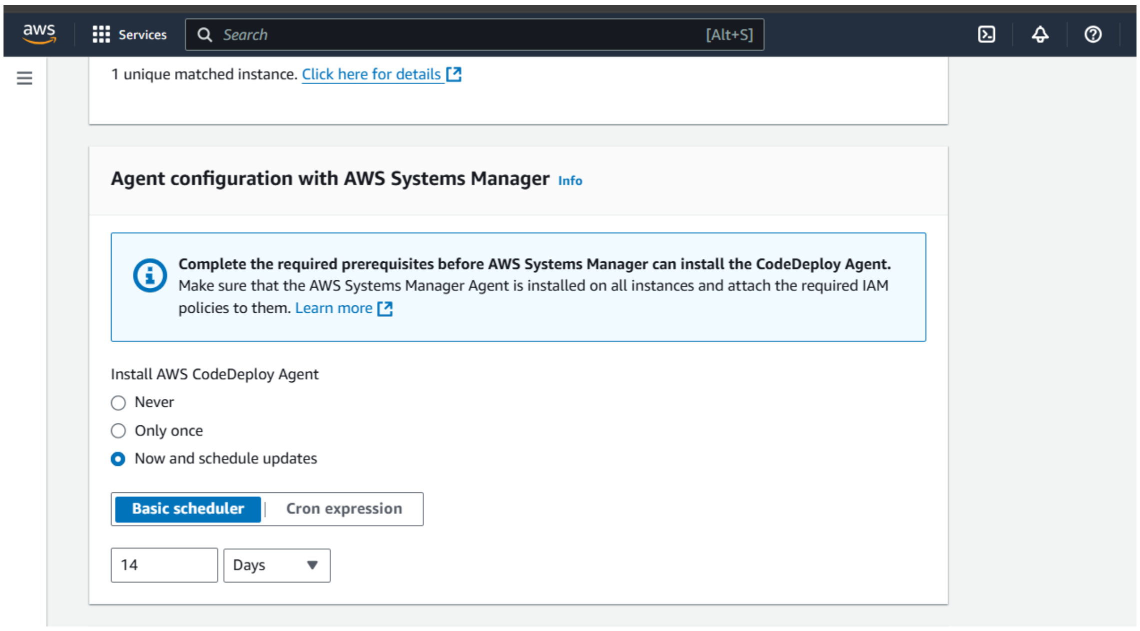
Matching instances

1 unique matched instance. Click here for details

Deployment configurations

Another key setting is the deployment configuration, which affects how updates are rolled out. I used CodeDeployDefault.AllAtOnce, so all target instances are updated at the same time, allowing faster deployment but with a higher risk of failure.

To connect EC2 instances with CodeDeploy, a CodeDeploy Agent is also set up to listen for deployment commands, pull the application revision, and run the scripts defined in `appspec.yml` to manage the deployment lifecycle on the instance.



1 unique matched instance. Click here for details

Agent configuration with AWS Systems Manager Info

Complete the required prerequisites before AWS Systems Manager can install the CodeDeploy Agent.
Make sure that 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

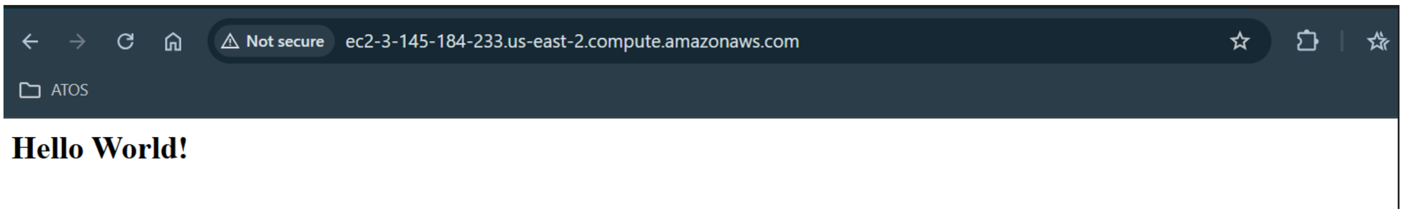
14 Days

Success!

A CodeDeploy deployment is the process of delivering an app version to target instances. The difference from a deployment group is that the group defines the settings and targets, while the deployment is the actual release execution to those instance.

I had to configure a revision location, which means specifying where the application revision is stored so CodeDeploy can access it. My revision location was an Amazon S3 bucket, where the WAR file and other deployment artifacts were stored.

To check that the deployment was a success, I visited the web application URL. I saw the updated version of the application running, confirming that the deployment was successful, and the changes were applied correctly.



Hello World!

You've learned how to:

- 👏 Launch a deployment architecture using CloudFormation.
- ✍️ Prepare deployment scripts and an appspec.yml file for CodeDeploy.
- ⚙️ Configure a CodeDeploy application and deployment group.
- 🚀 Deploy your web application using CodeDeploy (woah!).
- 💎 Implement disaster recovery and **roll back** a deployment!