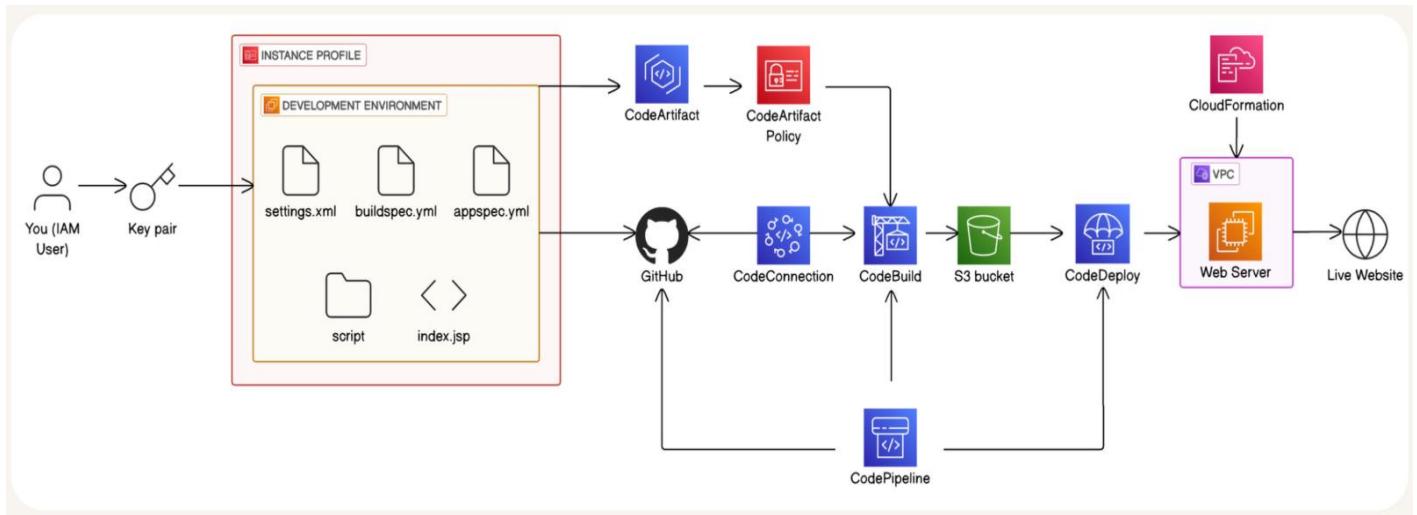


Automated CI/CD Pipeline with AWS



This project demonstrates how to **automate the deployment of a Java web application to EC2 using AWS CodeDeploy**, as part of a full CI/CD pipeline.

The pipeline follows a real-world DevOps flow:

- **Source:** GitHub
- **Build:** AWS CodeBuild (WAR artifact)
- **Artifact Storage:** Amazon S3
- **Deployment:** AWS CodeDeploy
- **Infrastructure:** AWS CloudFormation (IaC)

The goal was to eliminate manual deployments and replace them with a **repeatable, automated, and reliable deployment process**.

Why This Project Matters

In production environments, manually copying files to servers and restarting services is risky and error-prone. This project shows how to:

- Deploy consistently every time
- Reduce human error
- Enable rollbacks when deployments fail
- Treat infrastructure and deployments as code

Architecture Overview

End-to-end flow:

1. Developer pushes code to GitHub
2. CodeBuild compiles the app and produces a WAR file
3. Build artifacts (WAR + scripts + appspec.yml) are stored in S3
4. CodeDeploy pulls the artifact from S3
5. CodeDeploy runs lifecycle scripts on EC2
6. Application is deployed and made live

Step-by-Step Implementation

Step 1: Create IAM Access and Key Pair

- Log in using an IAM Admin user.
- Ensure permissions for EC2, S3, IAM, CodeArtifact, CodeBuild, CodeDeploy, CloudFormation, and CodeConnections.
- Create or use an EC2 key pair for SSH access.

Step 2: Launch Development EC2 Instance

This EC2 instance is used to write code and manage Git operations.

- Launch an EC2 instance
- Allow SSH access from your IP.
- Attach an IAM **instance profile** so the instance can securely access AWS services.
- Connect via SSH

On this instance, you manage:

- Application source code
- settings.xml
- buildspec.yml
- appspec.yml
- Deployment scripts

Step 3: Set Up Secure Dependency Management (CodeArtifact)

CodeArtifact is used to securely store and fetch Maven dependencies during the build process.

- Create a CodeArtifact **domain**
- Create a CodeArtifact **repository** (format: Maven).
- Enable Maven Central as an upstream repository.
- Grant permissions to the development EC2 role and CodeBuild role.

Configure Maven

- Update settings.xml to authenticate Maven with CodeArtifact.
- During builds, Maven pulls dependencies from CodeArtifact instead of the public internet.

Why CodeArtifact:

- Secure dependency sourcing
- Consistent builds
- Enterprise-grade supply chain security

Developer Tools > CodeArtifact > Repositories > maven-central-store

maven-central-store Info

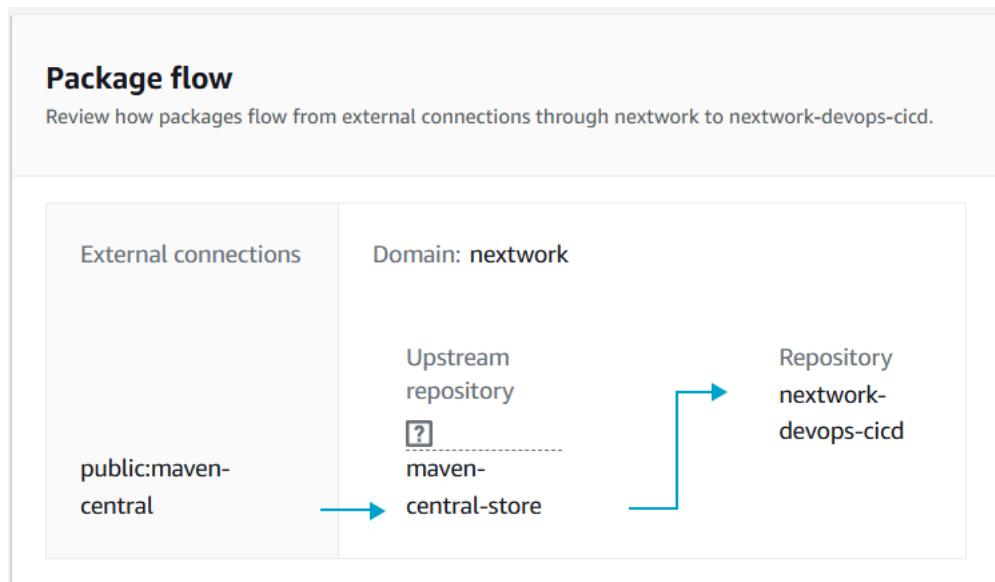
Repository Connected to public repository Provides Maven artifacts from Maven Central Repository.

Details Domain, policy, tags, ARN, and external connection.

Packages Info

Filter by package name prefix, format, namespace prefix, and origin controls

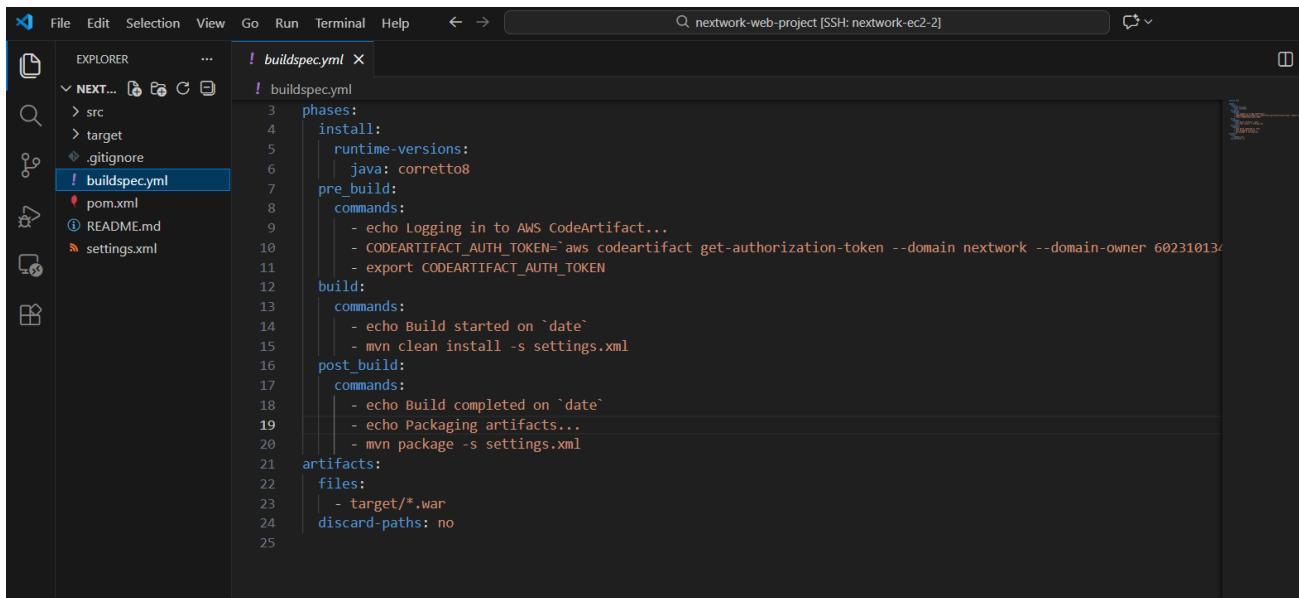
Package name	Namespace	Format	Latest version	Latest publish date	Publish	Upstream
backport-util-concurrent	backport-util-concurrent	maven	3.1	2 minutes ago	Block	Allow
classworlds	classworlds	maven	1.1	3 minutes ago	Block	Allow
google	com.google	maven	1	2 minutes ago	Block	Allow
jsr305	com.google.code.findbugs	maven	2.0.1	2 minutes ago	Block	Allow
google-collections	com.google.collections	maven	1.0	2 minutes ago	Block	Allow



Step 4: Push Source Code to GitHub

- Create a GitHub repository.
- Push application code and pipeline files:
 - Application source files
 - settings.xml
 - buildspec.yml
 - appspec.yml
 - scripts/ directory

GitHub acts as the single source of truth.



```

File Edit Selection View Go Run Terminal Help < > Q nextwork-web-project [SSH: nextwork-ec2-2]
EXPLORER ... ! buildspec.yml
NEXT... pom.xml .gitignore pom.xml
src target buildspec.yml
buildspec.yml
phases:
  install:
    runtime-versions:
      java: corretto8
  pre_build:
    commands:
      - echo Logging in to AWS CodeArtifact...
      - CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --domain nextwork --domain-owner 60231013...
      - export CODEARTIFACT_AUTH_TOKEN
  build:
    commands:
      - echo Build started on `date`
      - mvn clean install -s settings.xml
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Packaging artifacts...
      - mvn package -s settings.xml
  artifacts:
    files:
      - target/*.war
  discard-paths: no

```

Step 5: Connect GitHub to AWS (CodeConnection)

- Create an AWS CodeConnection connection to GitHub.
- Authorize access to your GitHub account and repository.
- This connection allows AWS services to pull source code securely.

Step 6: Configure CodeBuild (Continuous Integration)

CodeBuild compiles the application and produces deployable artifacts.

Setup

- Create an S3 bucket to store build artifacts.
- Create a CodeBuild project:
 - Source: GitHub (via CodeConnection)
 - Environment: Amazon Linux

IAM Permissions

Grant CodeBuild permissions to:

- Access CodeArtifact
- Write artifacts to S3

buildspec.yml Responsibilities

- Install dependencies
- Compile application
- Generate WAR file
- Package:
 - WAR file
 - appspec.yml
 - scripts/

Run the build and confirm it succeeds.

Step 7: Artifact Storage (S3)

- CodeBuild uploads the final artifact as a .zip file to S3.
- This S3 object becomes the **revision source** for CodeDeploy.

Step 8: Provision Production Infrastructure (CloudFormation)

CloudFormation is used to create the production environment.

Resources Created

- VPC
- Subnet
- Internet Gateway
- Route Tables
- Security Group (HTTP + SSH)
- Production EC2 instance

Important Tag

The EC2 instance is tagged as:

`role = webserver`

This tag allows CodeDeploy to identify deployment targets.

You have successfully created the template. Next, you can migrate to CDK or import to a stack.

Template details

Template generation status: Complete

Template ID: arn:aws:cloudformation:us-east-2:602310134577:generatedTemplate/56951718-0415-4952-96ed-9c6e95a5e6a4

Configurations

Deletion policy: DELETE

Update replace policy: DELETE

Creation time: 2026-01-25 15:33:57 UTC-0600

Updated time: 2026-01-25 15:34:04 UTC-0600

Built off existing stack: -

Catch errors before deployment: CloudFormation now validates your stack configuration changes upfront, preventing common deployment failures. [Get started](#)

Notifications: 0 | **Actions**: Delete, Update stack, Stack actions, Create stack

Resources (3)

Logical ID	Physical ID	Type	Status	Module
InternetGateway	igw-00a97f1b3820b2c02	AWS::EC2::InternetGateway	CREATE_IN_PROGRESS	-
ServerRole	NextWorkCodeDeployEC2Stack-ServerRole-j40IS6zEIV4t	AWS::IAM::Role	CREATE_IN_PROGRESS	-
VPC	vpc-0cf2475ee987bf5e5	AWS::EC2::VPC	CREATE_IN_PROGRESS	-

Step 9: Create CodeDeploy Application

- Create a CodeDeploy application.
- Select **EC2/On-premises** as the compute platform.

Step 10: Create CodeDeploy Deployment Group

- Create a deployment group.
- Select deployment type: **In-place**.
- Target EC2 instances using tag:
 - Key: role
 - Value: webserver

IAM Role

- Create a service role with AWSCodeDeployRole policy.
- Allows CodeDeploy to access EC2, S3, and CloudWatch.

Deployment Settings

- Deployment configuration: CodeDeployDefault.AllAtOnce
- Load balancer: Disabled
- CodeDeploy Agent updates enabled

Step 11: Prepare Deployment Scripts

Create a scripts folder with the following:

install_dependencies.sh

- Installs Tomcat and Apache
- Configures Apache as a reverse proxy

start_server.sh

- Starts Tomcat and Apache
- Enables services on reboot

stop_server.sh

- Safely stops services
- Checks if services are running before stopping

Scripts ensure automation and consistency.

Step 12: appspec.yml (Deployment Blueprint)

The appspec.yml file controls the deployment lifecycle.

Responsibilities

- Defines file copy locations
- Controls lifecycle hooks

Lifecycle Flow

- BeforeInstall → install dependencies
- ApplicationStop → stop services
- ApplicationStart → start services

CodeDeploy reads this file to execute deployments correctly.

Step 13: Create and Run Deployment

- Create a deployment in CodeDeploy.
- Select S3 as the revision source.
- Provide the S3 URI of the artifact .zip file.
- Start deployment.

CodeDeploy will:

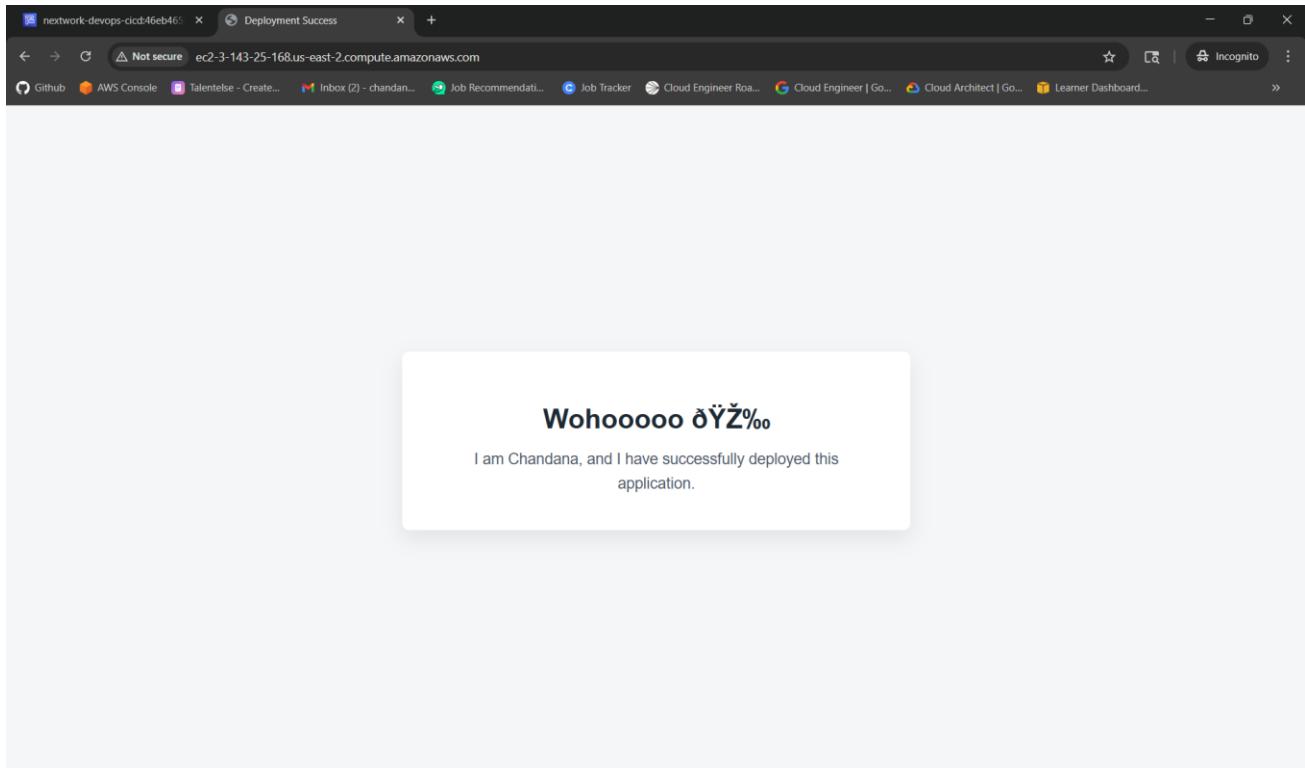
- Pull artifacts from S3
- Execute lifecycle scripts
- Deploy WAR to Tomcat

Step 14: Verify Live Application

- Retrieve the production EC2 public DNS.
- Open in a browser using:

`http://<public-dns>`

The application should be live.



Step 15: Disaster Recovery and Rollback

- Intentionally deploy a broken or outdated artifact.
- Observe deployment failure.
- Rebuild the application using CodeBuild.
- Redeploy using CodeDeploy.

This validates rollback and recovery capability.

Key Services Used

- Amazon EC2
 - Amazon S3
 - AWS CodeArtifact
 - AWS CodeBuild
 - AWS CodeDeploy
 - AWS CloudFormation
 - AWS IAM
-

Key Concepts Learned

- CI/CD pipelines
- Infrastructure as Code (IaC)
- Secure dependency management
- Automated deployments
- Deployment lifecycle hooks
- Rollback and disaster recovery
- IAM roles and least privilege