AWS CI/CD Workshop With Java Application

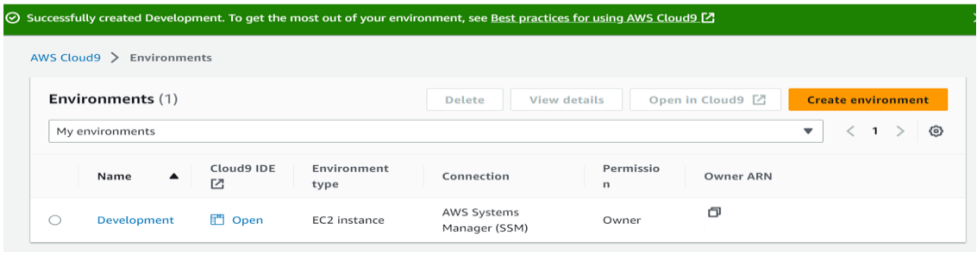
- CI/CD with Java Application workshop illustrates how we can use the AWS developer tools to implement Continuous Integration (CI) and Continuous Deployment (CD) for our application.
- First, I begin by creating cloud9 for cloud-based integrated development environment (IDE) for code editing and terminal use, and next proceed to deploy Cloud Development Kit(CDK) in cloud9 to quickly complete many tasks on AWS.
- And then start building a pipeline by creating a source code repository using CodeCommit to store the Java web application for testing.
- And CodeBuild to build the Java application source code into a deployable web archive.
- CodeDeploy to deploy the application and, finally, CodePipeline to automate the release process from source code all the way to deployment.

Before creating the cloud9 environment, just an intro of AWS Cloud9

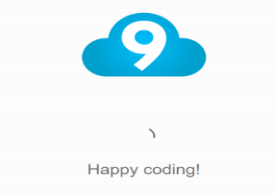
AWS Cloud9 IDE :

- AWS Cloud9 IDE offers a code-editing experience with support for several programming languages and runtime debuggers, as well as a built-in terminal.
- It contains a collection of tools that we use to code, build, run, test, and help to release software to the cloud. We access the AWS Cloud9 IDE through a web browser.
- The role of cloud9 in this workshop is that it is used as a git remote helper that makes it easier to interact with AWS CodeCommit like a local git terminal.
- This tool is installed in cloud9 by default and cloud9 works with AWS CDK to deploy AWS infrastructure components as code. For that, we have to instruct AWS Cloud9 to create an Amazon EC2 instance.
- This type of setup is called an EC2 environment and it stores our project's files locally, for instance.
- In Aws management console in services under developer tools we can find cloud9 in that I have created a cloud9 environment under name development and environment type EC2.

- AWS Cloud9 EC2 development environment running on Amazon Linux.
- The AWS Cloud9 IDE opens in a web browser.



Above it shows successfully created Development after we open cloud IDE it shows happy coding and Once ready, IDE will open to a welcome screen. We can see a terminal window

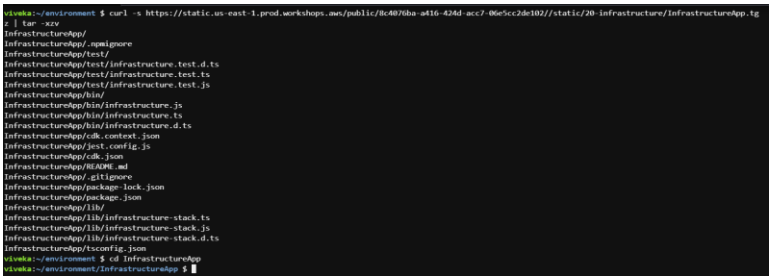


Now we need to create an infrastructure application stack using AWS CDK in the cloud9

- The AWS CDK is an open-source software development framework for defining cloud infrastructure in code and provisioning it through AWS CloudFormation.
- CDK Use AWS CloudFormation to do infrastructure deployments consistently and repeatedly, including error rollback.
- In this workshop we use a typescript programming language to provision the AWS resource.
- CloudFormation stack contains multiple AWS resources.
- In this workshop, we will simply execute the prebuilt provided CDK application.
- To prepare a CDK Application from our cloud9 environment in a terminal window.
- The below command downloads the CDK application source and extracts the application to our development environment.

curl -s https://static.us-east-1.prod.workshops.aws/public/8c4076ba-a416-424d-acc7-06e5cc2de102//static/20-infrastructure/InfrastructureApp.tgz | tar -xzv

cd InfrastructureApp



Next Install CDK application dependencies and build the CDK project using

npm install

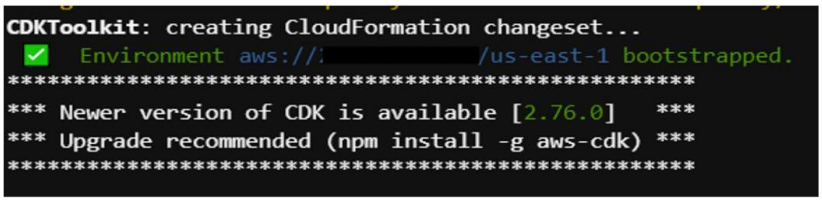
npm run build

- npm install cmd will install the AWS CDK package globally on our system which then allows us to initialize our first project and start building and deploying code in our AWS account using AWS CDK.
- npm run build cmd Use npm to run the TypeScript compiler to check for coding errors and then enable the AWS CDK to execute the project's file. The preceding command runs the TypeScript compiler.

CDK Bootstrap

- The first time we deploy an AWS CDK app into an environment for a specific AWS account and AWS Region combination, we must install a bootstrap stack.
- This stack includes various resources that the AWS CDK needs to complete its various operations. this stack includes an Amazon S3 bucket that the AWS CDK uses to store templates and assets during its deployment processes.
- To install the bootstrap stack, run the CDK Bootstrap cmd in our terminal window.

After we should see messages from CDK indicating that it bootstrapped our account and region

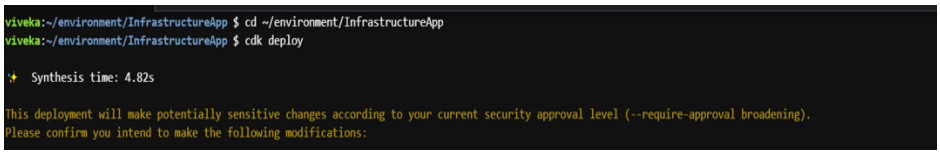


CDK Deploy

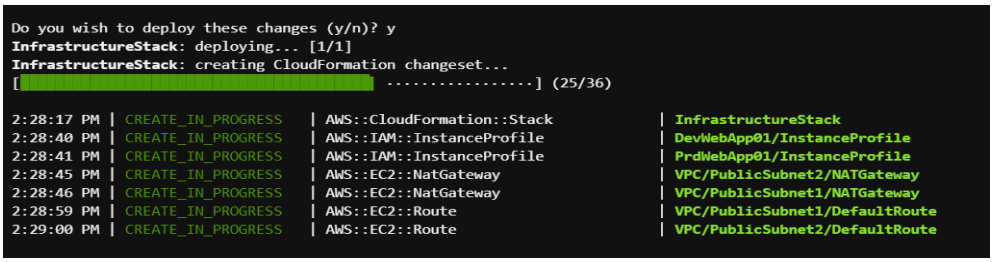
Next,we need to Deploy the CDK application to our account and region using the below cmd

cd ~/environment/InfrastructureApp

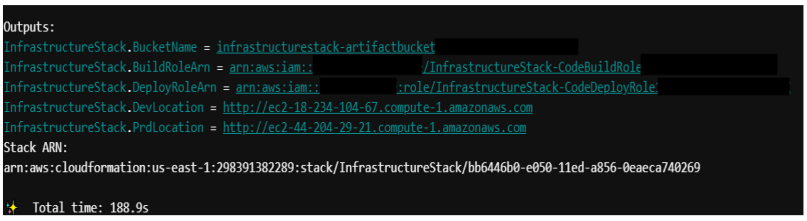
cdk deploy



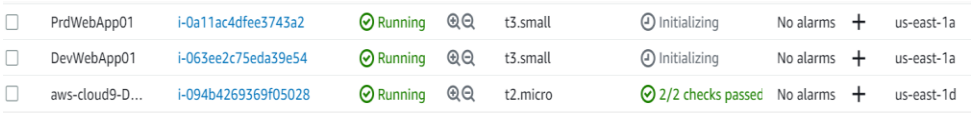
AWS CDK runs the AWS CloudFormation stack template to deploy the stack.



Upon successful deployment, we will see a list of output values in our terminal



which creates the 2 Amazon EC2 instances and s3bucket and code build and code deploy role and DNS for our 2 EC2 server names prdWebApp01 and DevWebApp01.



Source stage

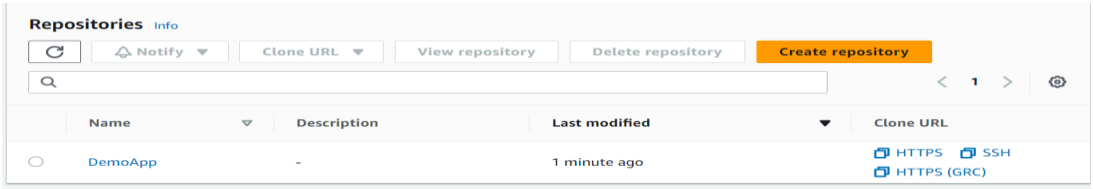
- we can now start building the components required for our CI/CD pipeline. Let's start with creating the source code repository using AWS CodeCommit and adding files to the repository.

AWS CodeCommit

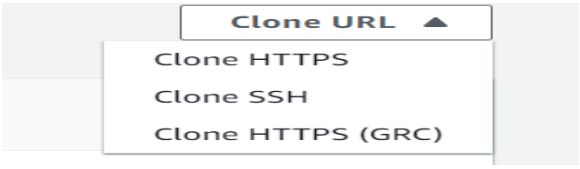
- CodeCommit is one of the AWS continuous integration tools it is similar to Github, CodeCommit is a fully-managed source control service that hosts secure Git-based repositories.
- we can create a code repository using Git commands to interact with our repository we can easily commit, branch and merge our code.
- We can store and version any kind of file, and source code.
- In this workshop, we are using cloud9 as a local git terminal, AWS Cloud9 contains a collection of in-built tools that we can use to write code and build, run, test, and debug.
- We use AWS Cloud9 to create a new repository in a CodeCommit. We create repositories, Clone existing repositories in codecommit, and commit and push code changes to a repository from our AWS Cloud9 EC2 development environment.
- The AWS Cloud9 EC2 development environment is generally preconfigured with the AWS CLI, an Amazon EC2 role, and Git, so in most cases, we can run a few simple commands and start interacting with our central repository.

Create Repository

In the AWS management console in services under developer tools we can see CodeCommit in that I have created a repository under the name DemoApp



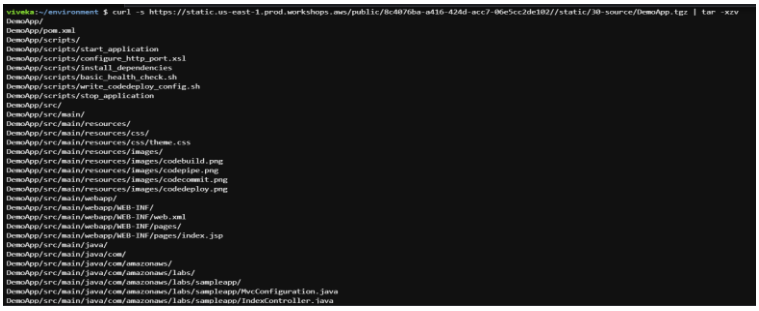
After creating the repository we have to Clone the repository URL by selecting the clone URL button and then selecting clone HTTPS



- Before cloning the repository URL in the cloud9 terminal have to change the directory to the environment directory using cd ~/environment cmd
- And clone our Git repository, using the clone URL we just copied earlier
- Using git clone <our repository URL>
- A clone is a full copy of a repository, including all logging and versions of files

```
viveka:~/environment $ cd ~/environment
viveka:~/environment $ git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/DemoApp
Cloning into 'DemoApp'...
warning: You appear to have cloned an empty repository.
viveka:~/environment $
```

- It shows msg like we have cloned an empty repository so we have added files to our repository
- I have populated my new repository using an already existing public access repository with a demo java application.
- Using this cmd curl -s https://static.us-east-1.prod.workshops.aws/public/8c4076ba-a416-424d-acc7-06e5cc2de102//static/30-source/DemoApp.tgz | tar -xzv
- cd DemoApp



- Before making the initial commit, we have to ConfigureGit with my name and email address. This is important for version control systems, as each Git commit uses this information. Using the below cmd

git config --global user.email "email"

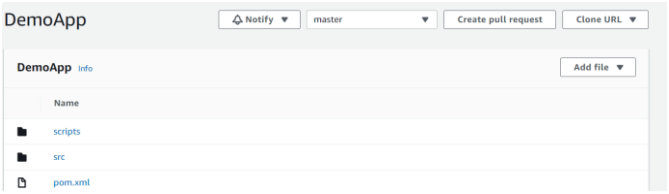
git config --global user.name "Name"

```
viveka:~/environment/DemoApp (master) $ git config --global user.email "vivekamano2001@gmail.com"
viveka:~/environment/DemoApp (master) $ git config --global user.email "Viveka"
viveka:~/environment/DemoApp (master) $
```

- Next, we commit our changes to the local Git repository and push our changes to the CodeCommit-hosted repository.
- Git add . cmd adds a change in the working directory to the staging area. It tells Git that we want to include updates to a particular file in the next commit.
- git commit -m ”Initial commit ”. This cmd will save all staged changes along with msg
- git push cmd is used to upload local repository content(cloud 9)to a codecomit repository.

```
viveka:~/environment/DemoApp (master) $ git commit -m "Initial commit"
[master (root-commit)  Initial commit
 16 files changed, 726 insertions(+)
 create mode 100644 pom.xml
 create mode 100644 scripts/basic_health_check.sh
 create mode 100644 scripts/configure_http_port.xsl
 create mode 100644 scripts/install_dependencies
 create mode 100644 scripts/start_application
 create mode 100644 scripts/stop_application
 create mode 100644 scripts/write_codedeploy_config.sh
 create mode 100644 src/main/java/com/amazonaws/labs/sampleapp/IndexController.java
 create mode 100644 src/main/java/com/amazonaws/labs/sampleapp/MvcConfiguration.java
 create mode 100644 src/main/resources/css/theme.css
 create mode 100644 src/main/resources/images/codebuild.png
 create mode 100644 src/main/resources/images/codecommit.png
 create mode 100644 src/main/resources/images/codedeploy.png
 create mode 100644 src/main/resources/images/codepipe.png
 create mode 100644 src/main/webapp/WEB-INF/pages/index.jsp
 create mode 100644 src/main/webapp/WEB-INF/web.xml
viveka:~/environment/DemoApp (master) $ git push
Enumerating objects: 32, done.
Counting objects: 100% (32/32), done.
Compressing objects: 100% (24/24), done.
Writing objects: 100% (32/32), 120.27 KiB | 5.23 MiB/s, done.
Total 32 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/DemoApp
 * [new branch]      master -> master
```

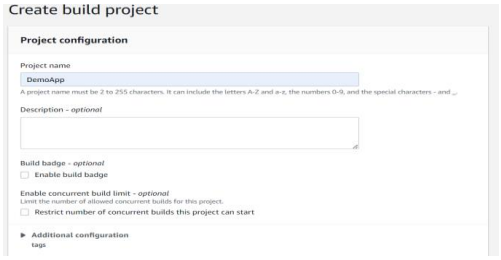
- After git push , in our codecommit repository we can see the files after populating, pushing it to local git(cloud9) to remote repository (code commit)



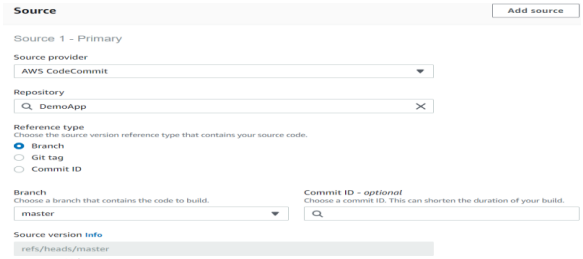
- Source stage has been completed. Now we have to move on to the build stage, which will test code we build our java application into a deployable artifact. For a Java web app, that artifact is a WAR file.

Create Build Project

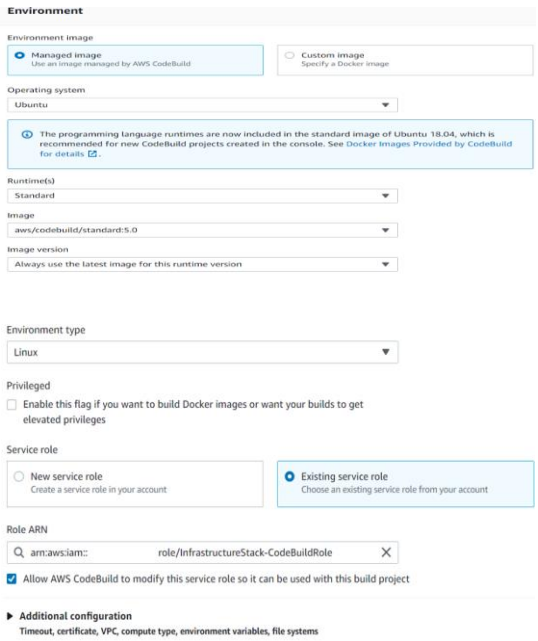
- We create build projects using AWS CodeBuild.
- AWS CodeBuild is a fully managed build service in the cloud. It is a continuous integration service .
- CodeBuild compiles source code, runs unit tests, and produces artifacts that are ready to deploy.
- A build project is used to define how CodeBuild will run a build. It includes information such as where to get the source code, which build environment to use, the build commands to run, and where to store the build output.
- A build environment is the combination of an operating system, programming language runtime, and tools used by CodeBuild to run a build.
- To create code built on the aws management console in services under developer tool tools, we can see code built there.
- In the project configuration panel, I entered DemoApp as the project name.



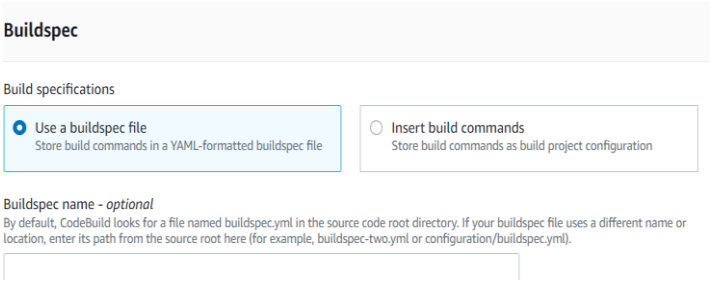
Next, in the source stage, I chose codecommit as a source provider and configured it.



In the Environment panel, a build environment is the combination of an operating system, programming language runtime, and tools used by CodeBuild to run a build. I configured it.

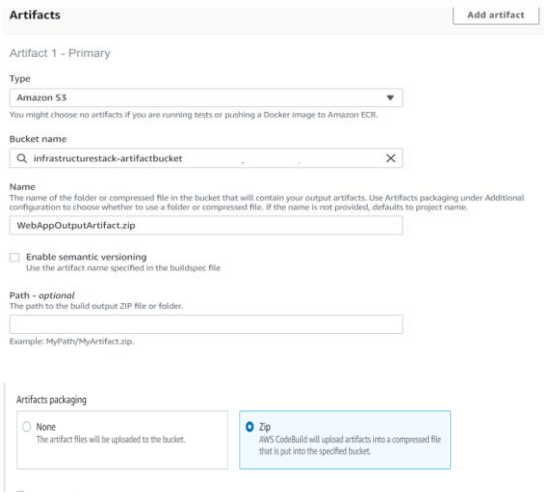


Buildspec panel: Build Specification – a YAML file that describes the collection of commands and settings for CodeBuild to run a build.

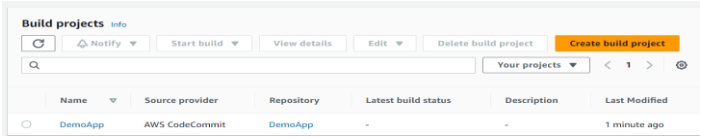


Artifacts panel:

- Build artifacts are files produced by a build. Typically, these include distribution packages, WAR files.
- When creating a build configuration, we specify the paths to the artifacts of our build on the Configuring General Settings page.
- AWS CodeBuild builds our code and stores the artifacts into an Amazon S3 bucket .



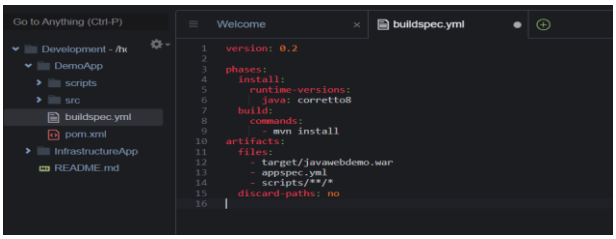
After the overall configuration, The build project was created in CodeBuild



Build Specification file

- In this step, we create a build specification (build spec) file. A buildspec is a collection of build commands and related settings, in YAML format, that CodeBuild uses to run a build. Without a build spec, CodeBuild cannot successfully convert our build input into build output or locate the build output artifact in the build environment to upload to our output bucket.

- In the Cloud9 environment, in the DemoApp repository, create this file, name it buildspec.yml, and then save it.



build spec declaration:

Version: 0.2 : version represents the version of the build spec standard being used. This build spec declaration uses the latest version, 0.2.

Phases : represents the build phases during which we can instruct CodeBuild to run commands. During the build phase, CodeBuild runs the mvn install command.

- This command instructs Apache Maven to compile, test, and package the compiled Java class files into a built output artifact.
- For each build phase, CodeBuild runs each specified command, one at a time, in the order listed, from beginning to end.

Artifacts:

- Represent the set of build output artifacts that CodeBuild uploads to the output bucket. Files represent the files to include in the build output.
- CodeBuild uploads the javawebdemo.war file found in the target relative directory in the build environment.
- The file javawebdemo.war and the directory name target are based on the way Apache Maven creates and stores and builds output artifacts for this.

After adding new files to our local git(cloud9) we have to commit the changes in the local Git repository and push the changes to the CodeCommit-hosted repository.

Using this command

cd ~/environment/DemoApp

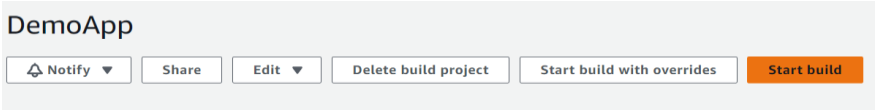
git add .

git commit -m "Add buildspec"

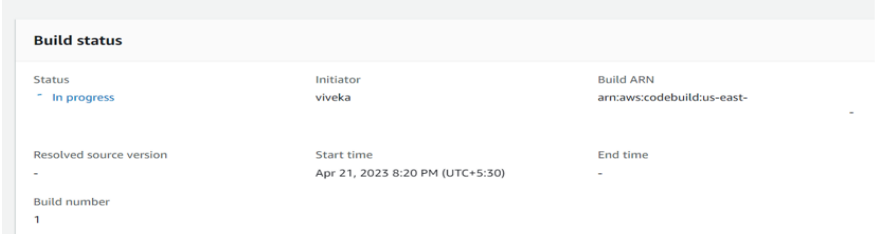
git push

Verify Build

- In this step, we instruct AWS CodeBuild to run the build with the settings in the build project. In the build projects, choose demoApp, and then choose Start build. The build starts immediately.



We can see the build has been started. It is in progress in less than a minute. It will change its status from in progress to succeed state.



Deploy

Now we have a WAR file, we can deploy the application to a server running on an EC2 instance in our test network.

AWS CodeDeploy

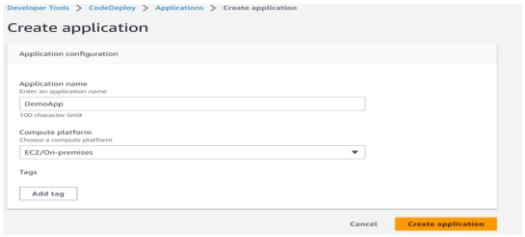
- CodeDeploy is a deployment service that automates application deployments to Amazon EC2 instances.
- AWS CodeDeploy makes it easier for us to rapidly release new features, helps to avoid downtime during deployment, and handles the complexity of updating our applications.

  Create an application and deployment group using CodeDeploy.

- On the AWS Management console in services under developer tools, we can find CodeDeploy.

Application configuration panel

- An Application is a name that uniquely identifies the application we want to deploy. I created DemoApp has application name.
- The Compute platform is the platform on which CodeDeploy deploys an application. Here I configured EC2/On-premises as compute platform.

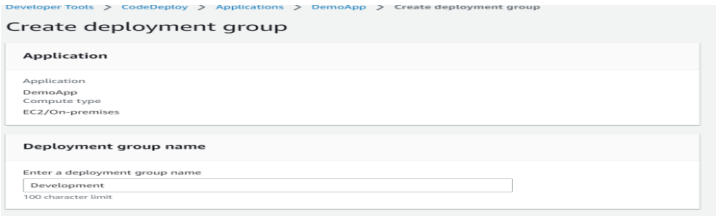


Deployment Group

Deployment groups include individually tagged instances. In an EC2/On-Premises deployment, a deployment group is a set of individual instances targeted for a deployment. We can add instances to a deployment group by specifying a tag.

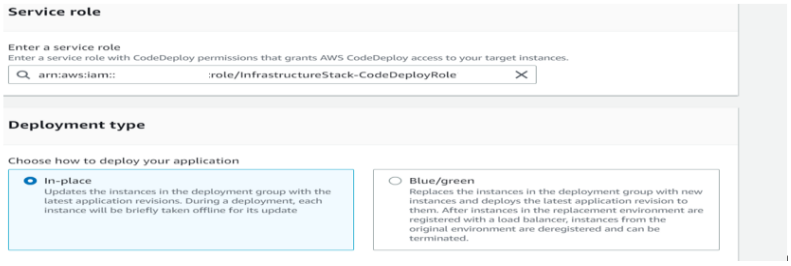
In the Deployment Group Name panel:

Configured development as deployment name



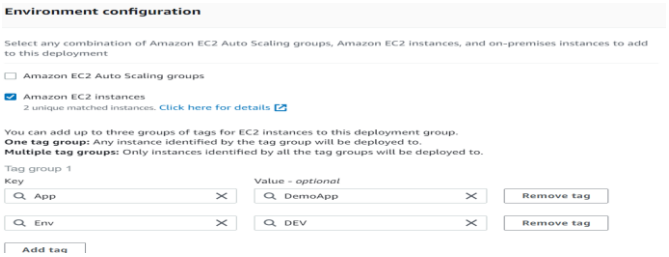
In the Service Role panel:

- Service roles are used to grant permission to an AWS service so it can access AWS resources. The policies that we attach to the service role determine which resources the service can access and what it can do with those resources.
- For EC2/On-Premises deployments, we should attach the AWSCodeDeployRole policy.
- In an in-place deployment, the instances in the deployment group are updated with the latest application revision.



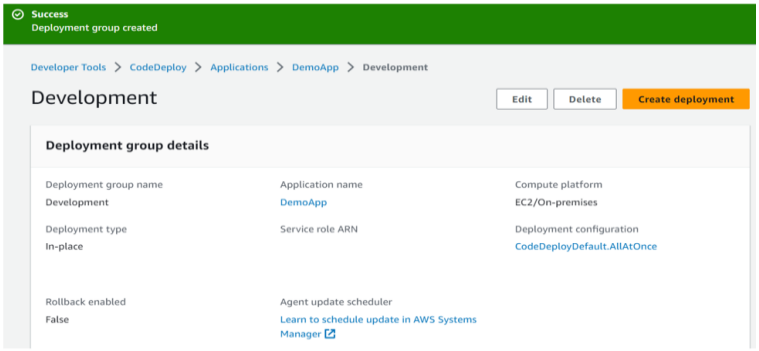
In the Environment Configuration panel

- It is a set of instances associated with an application that we target for deployment. We can add instances to a deployment group by specifying a tag.

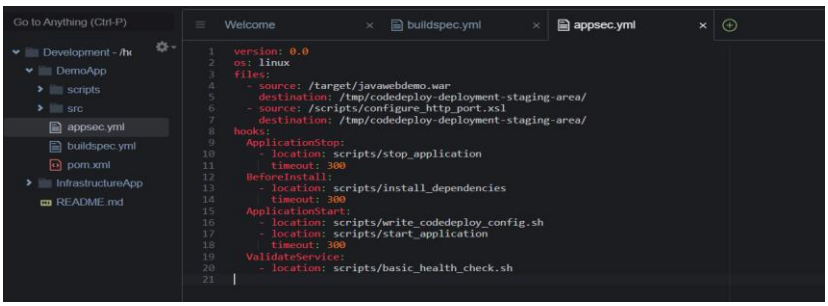


In the Deployment Settings panel:

- The Deployment configuration default and unchecked load balancer and created deployment group



- And successfully created a deployment group in code deploy
- Next is to Add Application Specification file, In our Cloud9 environment, we have to create an appspec.yml file in the DemoApp repository with the following content:



- The application specification file (AppSpec file) is a YAML file used by CodeDeploy to manage a deployment.
- The AppSpec file defines the deployment actions we want AWS CodeDeploy to execute, and consists of two sections. The files section specifies the source files in our revision to be copied and the destination folder in each instance.
- The hooks section specifies the location of the scripts to run during each phase of the deployment. Each phase of a deployment is called a deployment lifecycle event.

Appsecfile declaration

- Version :This section specifies the version of the AppSpec file. The only allowed value is 0.0. It is reserved by CodeDeploy for future use.
- OS: This section specifies the operating system value of the instance to which we deploy. Linux – for instance, is Amazon Linux
- Files :This section specifies the names of files that should be copied to the instance during the deployment's Install event.
- Source : instructions identify a file or directory from our revision to copy to the instance .The destination instruction identifies the location of the instance where the files should be copied.
- Hooks This section specifies scripts to run at specific deployment lifecycle events during the deployment.

ApplicationStop – This deployment lifecycle event occurs even before the application revision is downloaded. We can specify scripts for this event to gracefully stop the application or remove currently installed packages in preparation for a deployment.

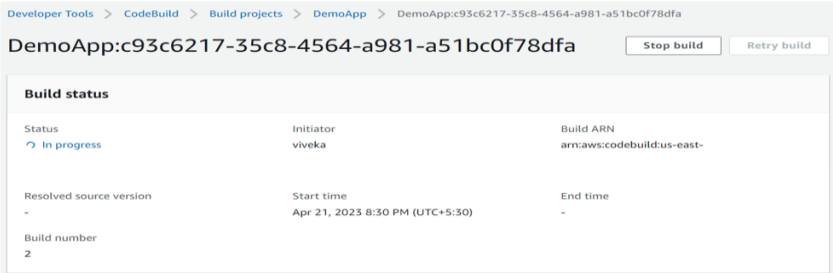
BeforeInstall – we can use this deployment lifecycle event for preinstall tasks, such as decrypting files and creating a backup of the current version.

ApplicationStart –we typically use this deployment lifecycle event to restart services that were stopped during ApplicationStop.

ValidateService – This is the last deployment lifecycle event. It is used to verify the deployment was completed successfully.

After adding the appsec file, we have to commit our changes to the local Git repository and push our changes to the CodeCommit-hosted repository.

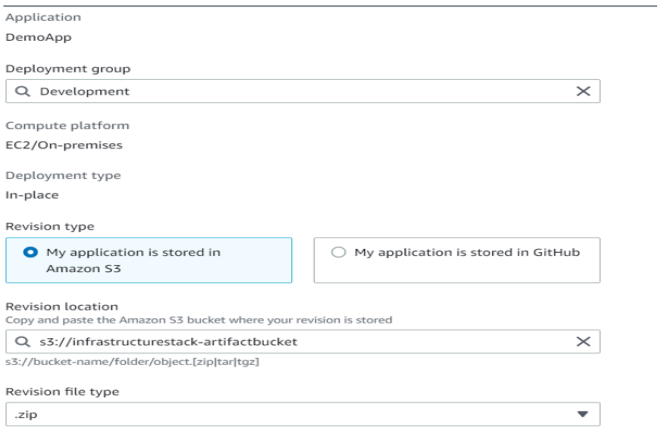
- Trigger a new build for our application so that the deployment artifact stored in S3 will contain our appspec.yml file.
- In the code build for our demoapp project, the selected start build we can see below. Here the build has started it is our second build.



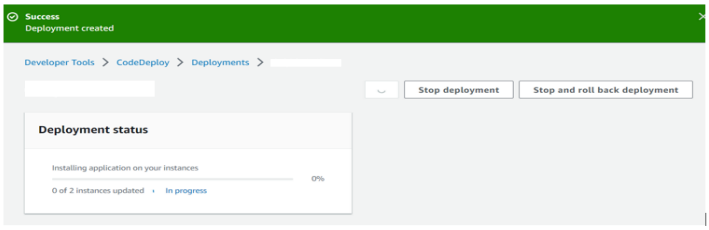
Verify Deployment:

Verify our application and specification by initiating a deployment using the CodeDeploy console.

- In this step, we will use the CodeDeploy console to verify the success of the deployment. We will use our web browser to view the web page that was deployed to the instance we created .
- In codedeploy in my demoapp application, I selected to create deployment
- In the deployment setting panel, enter s3 bucket name in the revision location field. We will find our bucket name in the output of the CDK application deployment with the key InfrastructureStack.BucketName.

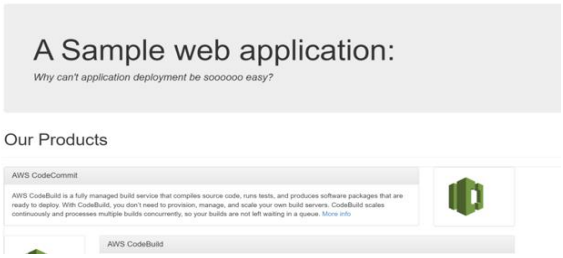


Review the default select option and created deployment



After deployment, status changes from in progress to succeeding the state.

- To check the output of the stack which is created in cdk output, if we copy the InfrastructureStack.DevLocation Ip address and paste it into our browser, it will show the image below in our web browser.



Pipeline

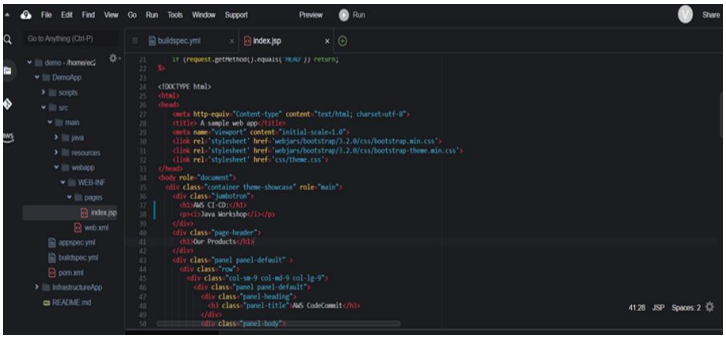
- Now that we have built and verified the build project and deployment, we are ready to construct the pipeline that moves changes from the source repository to our test server.

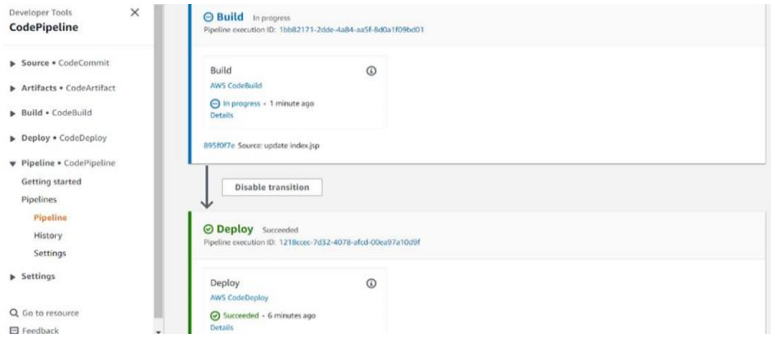
AWS CodePipeline

- AWS CodePipeline is a continuous integration and continuous delivery service for fast and reliable application and infrastructure updates. We can use CodePipeline to fully model and automate our software release processes.
- On an AWS management console under developer tools, we can see codepipeline
- In that, a pipeline created as a demoapp
- And choose codecommit as source provider
- And in the build panel, configure aws code build as build provider
- In deploy stage, choose code deploy as deploy provider
- Reviewed and created pipeline
- After creating a pipeline, verify our deployment by loading the web application into our browser. We will find our development server location in the output of the CDK application deployment with the key InfrastructureStack.DevLocation.

Experiment with changing the web application and triggering new pipeline executions

- In our Cloud9 IDE and open DemoApp/src/main/webapp/WEB-INF/pages/index.jsp in our editor. changed the title to CI-CD and saved the file.
- And commit our changes to the local Git repository and push our changes to the CodeCommit-hosted repository.
- After that, our pipeline detects the new revision and proceeds through the pipeline stages.





Once deployment succeeds, reload our browser and check our updated title instead of the IP address. It will be changed to a CI-CD .

Conclusion

We built a CI/CD pipeline using CodePipeline. Our pipeline began with code in CodeCommit, built WAR using CodeBuild and deployed the code to a test network we built with CDK.