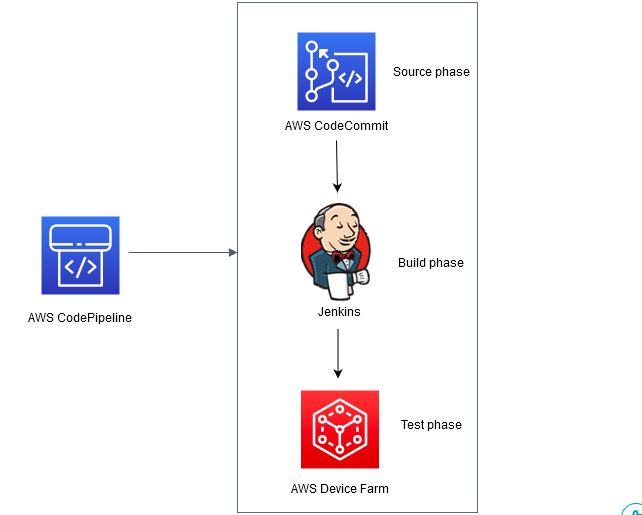
**Jenkins-POC**

**Architecture:**

A new commit to the source repository triggers the pipeline. The build is done on a Jenkins server, and the build artifact from Jenkins is passed to the test phase, which is configured with Aws Device Farm to test the application on real devices. Aws Code Pipeline provides the orchestration and helps automate the build and test phases.



**Prerequisites:**

Ensure you have the following prerequisites set up before beginning:

1. Apple developer account (Paid or Provisioning Profile and Certifcate)
2. Build server (macOS)
3. Xcode Version 11.3 + (installed on the build server and setup)
4. Jenkins (installed on the build server)
   * Aws CodePipeline Jenkins Plugin installed
5. AWS CLI installed and configure on the workstation

**Source:**

For this demo, I am using Simple iOS notes app which is hosted in CodeCommit Repository. Here is the URL of the repo:

<https://github.com/aws-samples/aws-mobile-ios-notes-tutorial>

**Launch EC2 Mac-Server Instance:**

For this POC we have to require a mac-server. In this case, I am using the ec2-mac instance. Let’s check how to configure and provision EC2 Mac.

To launch an Amazon EC2 Mac instance, you must first allocate a dedicated host. Here is a cfn template to deploy a dedicated host, security group, and ec2 mac instance.

Note: Change parameters of resources according to your requirement

|  |
| --- |
| AWSTemplateFormatVersion: "2010-09-09" Description: Template to deploy EC2 mac instance onto a dedicated host Parameters:  project:  Type: String  Default: weom  Description: Project Name  app:  Type: String  Default: ios  Description: App name  env:  Type: String  Default: poc  AllowedValues:  - dev  - poc  - qa  - uat  - prod   Description: Environment name  imageId:  Type: String  Default: ami-062fd989fe6709f19  Description: The ID of the AMI  keyName:  Type: String  Default: jenkins-mac2  Description: The Name of the key pair  autoPlacement:  Type: String  Default: "off"  Description: The AutoPlacement for the dedicated host  availabilityZone:  Type: String  Default: "us-east-2b"  Description: The Availability Zone of the instance  instanceType:  Type: String  Default: "mac2.metal"  Description: The Instance type  hostRecovery:  Type: String  Default: "off"  Description: The Host Recovery for the host  vpcId:  Type: String  Default: "vpc-0a5016aaca24644d1"  Description: The Vpc ID  subnetId:  Type: String  Default: subnet-02583e38b5949ee2c  Description: The Subnet ID  ingressIpProtocol:  Type: String  Default: tcp  Description: The IP protocol name  cidrIp:  Type: String  Default: "0.0.0.0/0"  Description: The IPv4 address range  sshPort:  Type: String  Default: 22  Description: SSH Port  httpPort:  Type: String  Default: 80  Description: HTTP Port  httpsPort:  Type: String  Default: 443  Description: HTTPS Port  jenkinsPort:  Type: String  Default: 8080  Description: Jenkins Port  associatePublicIp:  Type: String  Default: "true"  Description: Associate Public IP  deleteOnTermination:  Type: String  Default: "true"  Description: Delete on termination  deviceIndex:  Type: String  Default: "0"  Description: Device index Resources:  dedicatedHost:  Type: 'AWS::EC2::Host'  Properties:  AutoPlacement: !Ref autoPlacement  AvailabilityZone: !Ref availabilityZone  InstanceType: !Ref instanceType  HostRecovery: !Ref hostRecovery  instanceSecurityGroup:  Type: 'AWS::EC2::SecurityGroup'  Properties:  GroupDescription: Enable SSH, HTTP and HTTPS , Jenkins access to the instance  VpcId: !Ref vpcId  SecurityGroupIngress:  - IpProtocol: !Ref ingressIpProtocol  FromPort: !Ref sshPort  ToPort: !Ref sshPort  CidrIp: !Ref cidrIp  - IpProtocol: !Ref ingressIpProtocol  FromPort: !Ref httpPort  ToPort: !Ref httpPort  CidrIp: !Ref cidrIp  - IpProtocol: !Ref ingressIpProtocol  FromPort: !Ref httpsPort  ToPort: !Ref httpsPort  CidrIp: !Ref cidrIp  - IpProtocol: !Ref ingressIpProtocol  FromPort: !Ref jenkinsPort  ToPort: !Ref jenkinsPort  CidrIp: !Ref cidrIp  ec2Instance:  Type: 'AWS::EC2::Instance'  Properties:  InstanceType: !Ref instanceType  AvailabilityZone: !Ref availabilityZone  HostId: !Ref dedicatedHost  ImageId: !Ref imageId  KeyName: !Ref keyName  NetworkInterfaces:  - AssociatePublicIpAddress: !Ref associatePublicIp  DeviceIndex: !Ref deviceIndex  DeleteOnTermination: !Ref deleteOnTermination  GroupSet:  - !Ref instanceSecurityGroup  SubnetId: !Ref subnetId Outputs:  ec2Instance:  Description: Ec2 Instance Id  Value: !Ref ec2Instance  Export:  Name: !Sub "${project}-${app}-${env}-ec2Instances"  instanceSecurityGroup:  Description: Security Group Id  Value: !Ref instanceSecurityGroup  Export:  Name: !Sub "${project}-${app}-${env}-securityGroup"  dedicatedHost:  Description: Dedicated Host Id  Value: !Ref dedicatedHost  Export:  Name: !Sub "${project}-${app}-${env}-dedicatedHosts" |

**Enable Graphical Remote Desktop (VNC):**

In this module, we will enable MacOS's graphical screen-sharing software VNC. Firstly, we install VNC software to enable MacOS's graphical screen. Here is the URL to install VNC: <https://www.realvnc.com/en/connect/download/viewer/>

To establish a graphical VNC session:

* Connect to your EC2 Instance using **SSH Client** onto your local Machine.
* Run the following command to install and start VNC (macOS screen sharing SSH) from the macOS instance:

1. sudo defaults write /var/db/launchd.db/com.apple.launchd/overrides.plist

com.apple.screensharing -dict Disabled -bool false

2. sudo launchctl load -w

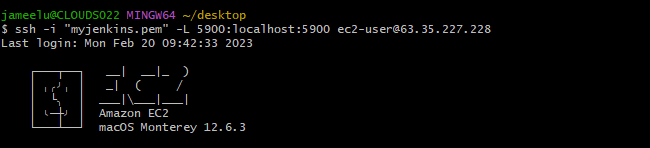
/System/Library/LaunchDaemons/com.apple.screensharing.plist

* Run the following command to set a **password** for ec2-user:

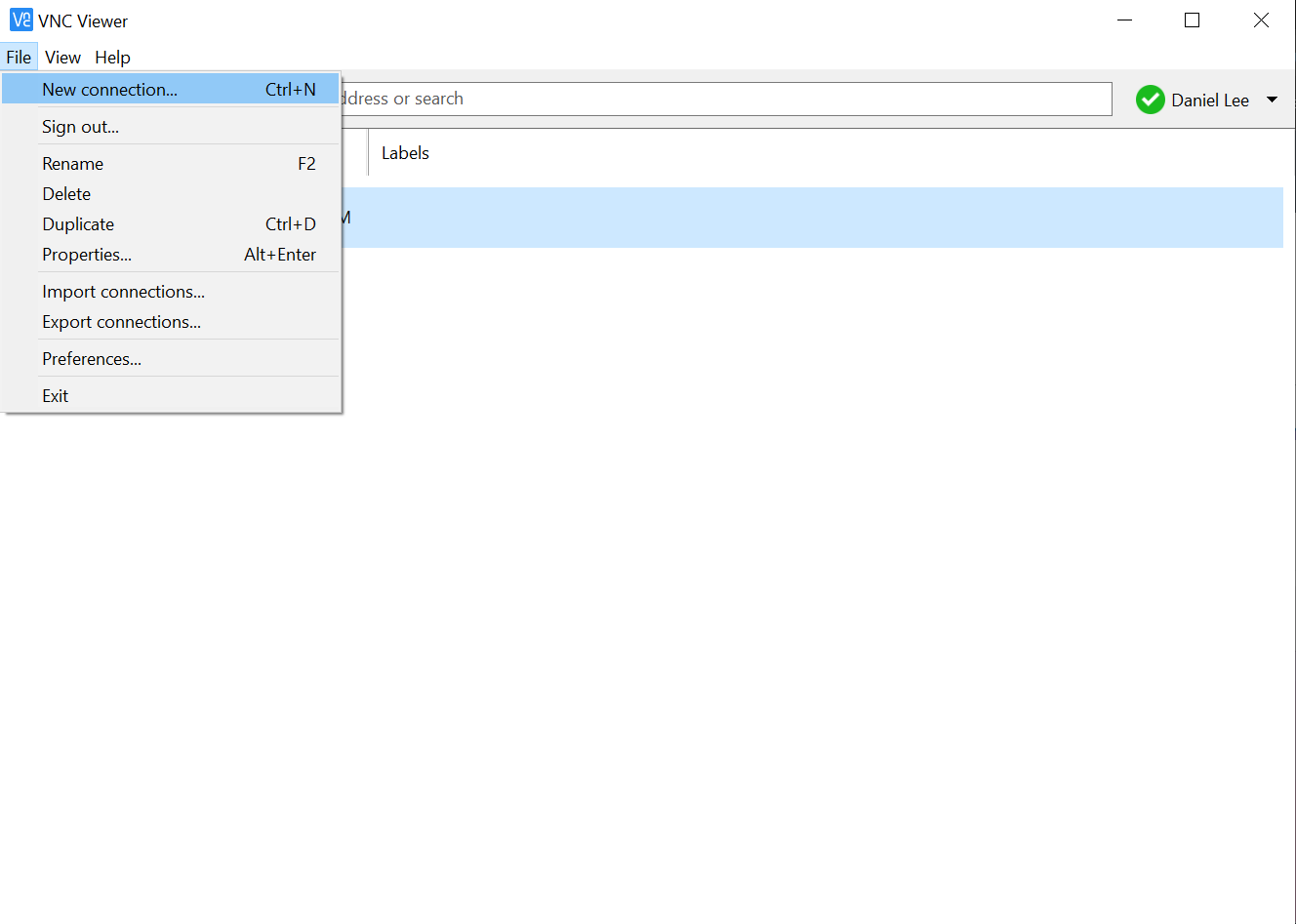
3. sudo /usr/bin/dscl . -passwd /Users/ec2-user

* Create an SSH tunnel to the VNC port. In the following command, replace keypair\_file with your SSH key path, and replace 192.0.2.0 with your instance's IP address or DNS name:

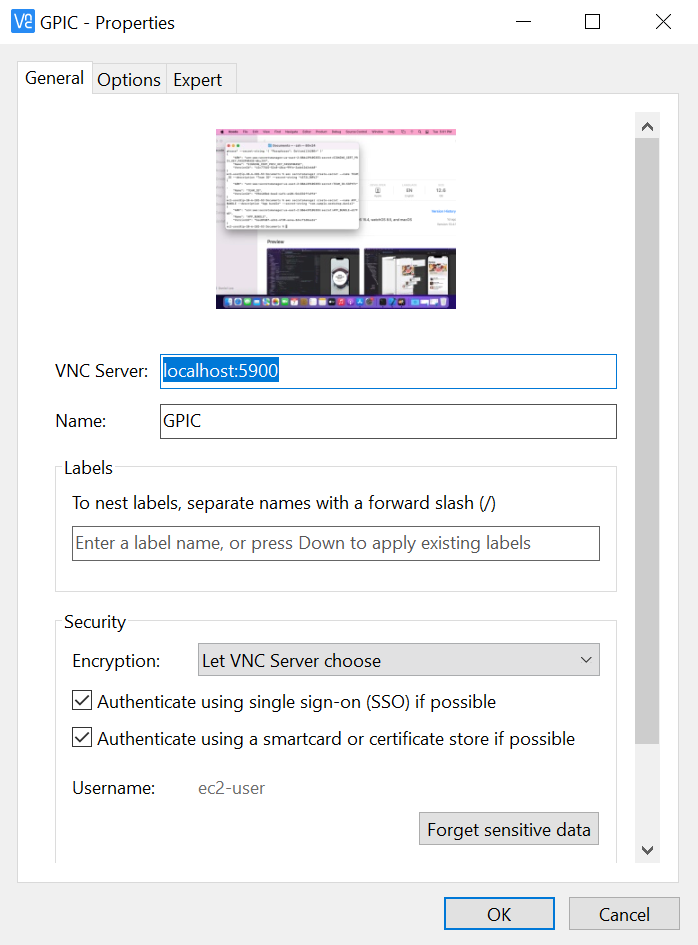
4. ssh -i “keypair\_file.pem” -L 5900:localhost:5900 ec2-user@192.0.2.0



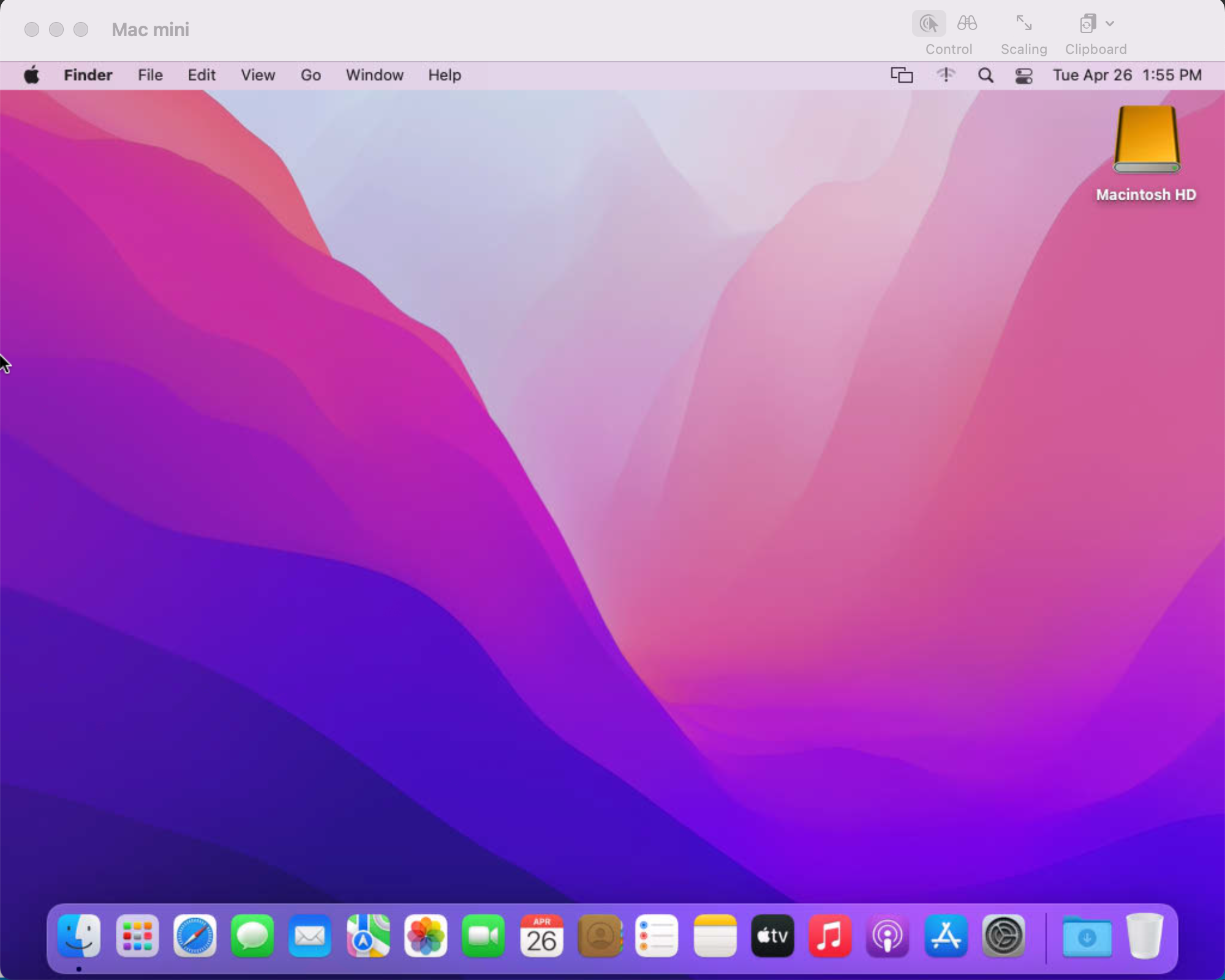
* After that go to your VNC Client and do sign up and sign in.
* Click "File" -> "New Connection" as shown in screenshot.



* Type localhost:5900 in VNC server and Name



* Type localhost:5900 in VNC server and Name
* Click Ok
* Connect.
* At this point, you should see the following screen in your VNC client:



Here we are done with our Mac-Server GUI.

**Install XCode and accept the license:**

To install XCode using the homebrew package manager, you need to install homebrew first.

* First of all we install the homebrew package manager with the following command:

1. [/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"](https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh))

* After Installation it will give you two command which is compulsory to run in order to complete the installation process for homebrew.
* After that run the following command to install ‘**mas’** which is compulsory to download the xcode application using the app store :

1. brew install mass

* Now run the following command to download the xcode application:

1. mas install 497799835

* The above command downloads the Xcode app from the Mac App Store using the product ID 497799835. This is the product ID of Xcode as listed on the Mac App Store.
* Please note that you must have signed in to the Mac App Store with your Apple ID in order to use the mas command to download Xcode.
* Run the following command to accept license”

1. sudo xcodebuild -license accept

**Provisioning Profile on Xcode:**

A provisioning profile is a critical component of the app development process in Xcode, as it provides the necessary security credentials, permissions, and entitlements required for code signing, device testing, and app distribution.

This is the necessary module to build code through a third-party tool like Jenkins.

Here is the step you can follow for provisioning profiles on Xcode.

* Put all your **.cer** and **.p12** files in the zip folder with the name of **profiles.zip.**
* Now Upload the Zip file in the S3 bucket
* To upload zip file from the s3 bucket to your machine run the following command:

1. aws s3 cp s3://<your-bucket-name>/profiles.zip .
2. unzip profiles.zip

* Now, for uploading the certificate and .p12 files in the keychain login run the following commands. (Note: change login keychain path, .cer and .p12 files name with your path and names)

1. sudo security add-certificates -k /Users/ec2-user/Library/Keychains/login.keychain-db ios\_development.cer
2. For .p12 you have to convert into .pem then upload it into keychain. Run the following commond to do this.
3. openssl pkcs12 -in CertificatesDev.p12 -out CertificatesDev.pem -passin pass:123456
4. security import CertificatesDev.pem -k /Users/ec2-user/Library/Keychains/login.keychain-db -P 123456 -T /usr/bin/codesign -A

**Jenkins Installation:**

Jenkins can be installed on macOS using a homebrew package manager for macOS with the following commands:

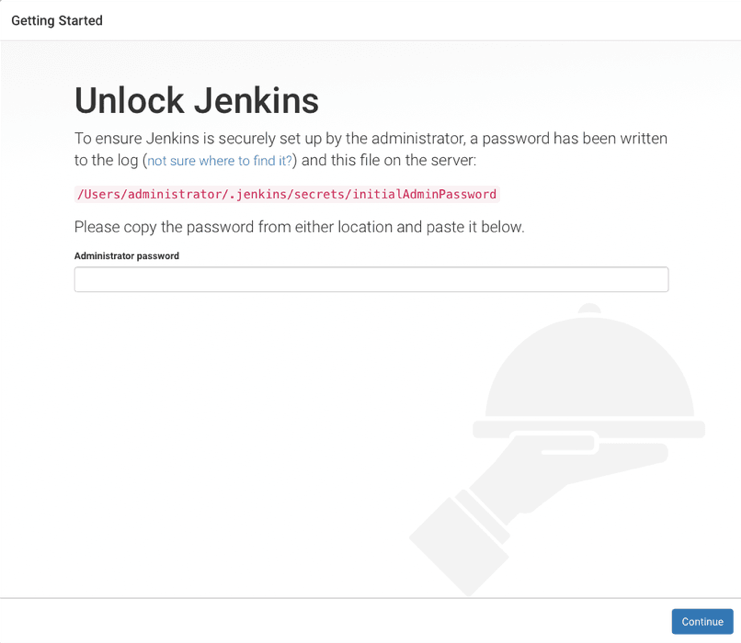
* First of all we install the homebrew package manager with the following command:

1. [/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"](https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh))

* After Installation it will give you two command which is compulsory to run in order to complete the installation process for homebrew.
* Now we are able to install jenkins on macOS by running following commands:

1. brew install jenkins-lts
2. brew services start jenkins-lts

* After running these two commands Jenkins can install successfully on your localhost.
* On a browser on your local machine visit <http://localhost:8080/>. You should see the setup screen shown in the following screenshot.



**Jenkins Configuration:**

* First of all, we have to make some changes in config file to ensure jenkins is accessible from anywhere, not just the local machine.

1. Open the config file by running following command: **nano /usr/ec2-user/local/Cellar/jenkins-lts/2.xxx/homebrew.mxcl.jenkins.plist**

2. Find the following line <string>--httpListenAddress=127.0.0.1</string>

3. Change it to the : <string>--httpListenAddress=0.0.0.0</string>

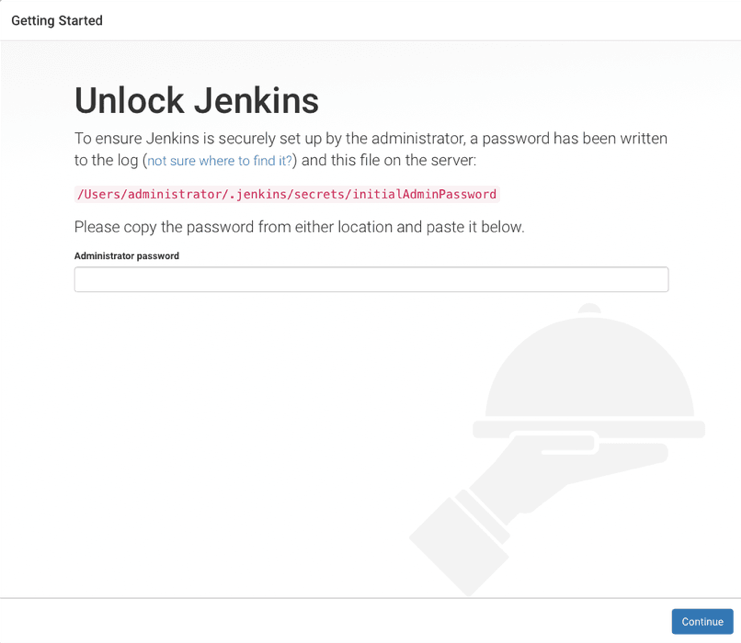
4. Run the following command to change listen address

**sed -i '' 's/--httpListenAddress=127.0.0.1/--httpListenAddress=0.0.0.0/g' homebrew.mxcl.jenkins-lts.plist**

* After this restart the jenkins server by using the following command:

1. brew services restart jenkins-lts

* Now you are able to reach jenkins through internet by writing <macOS server Public ip>:<Jenkins-port> For Example: 1.2.3.4:8080
* When you visit this link <Public\_ip>:8080 You see the following screen:



* Now to unlock the Jenkins grab the initial admin password from the terminal by typing:

1. cat /Users/administrator/.jenkins/secrets/initialAdminPassword

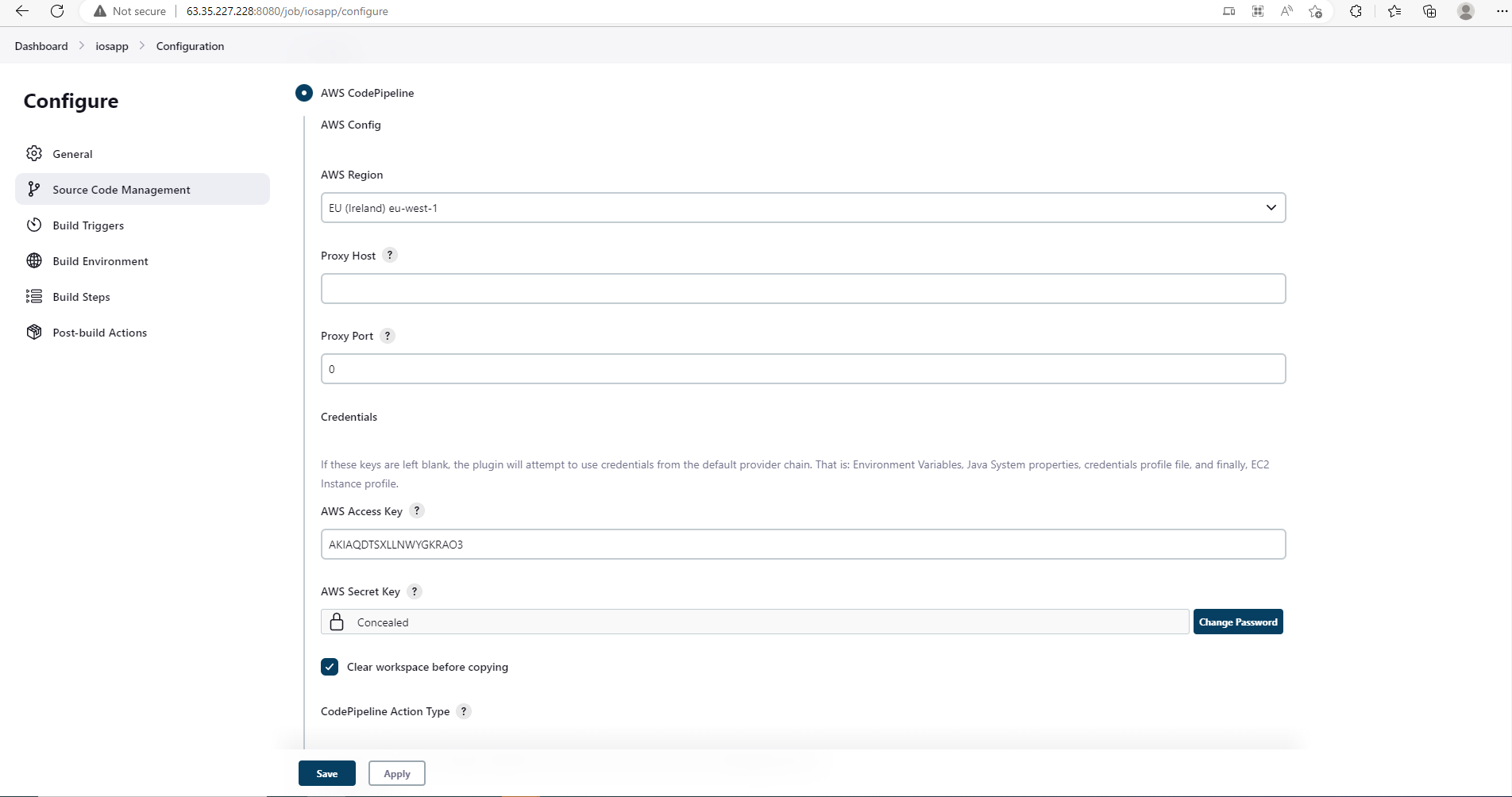
* After this, follow the on screen instruction to complete the setup. This includes creating a first admin user, installing the initial plugin, etc.
* Now, install the AWS CodePipeline Plugin.

1. Sign in to Jenkins using the username and password you created. Choose **Manage Jenkins**, then **Manage Plugins**.
2. Switch to the **Available** tab and start typing CodePipeline into the filter until AWS CodePipeline Plugin appears. Select the plugin, then select **Install without restart**.
3. Select **Restart Jenkins when installation is complete** and no jobs are running.

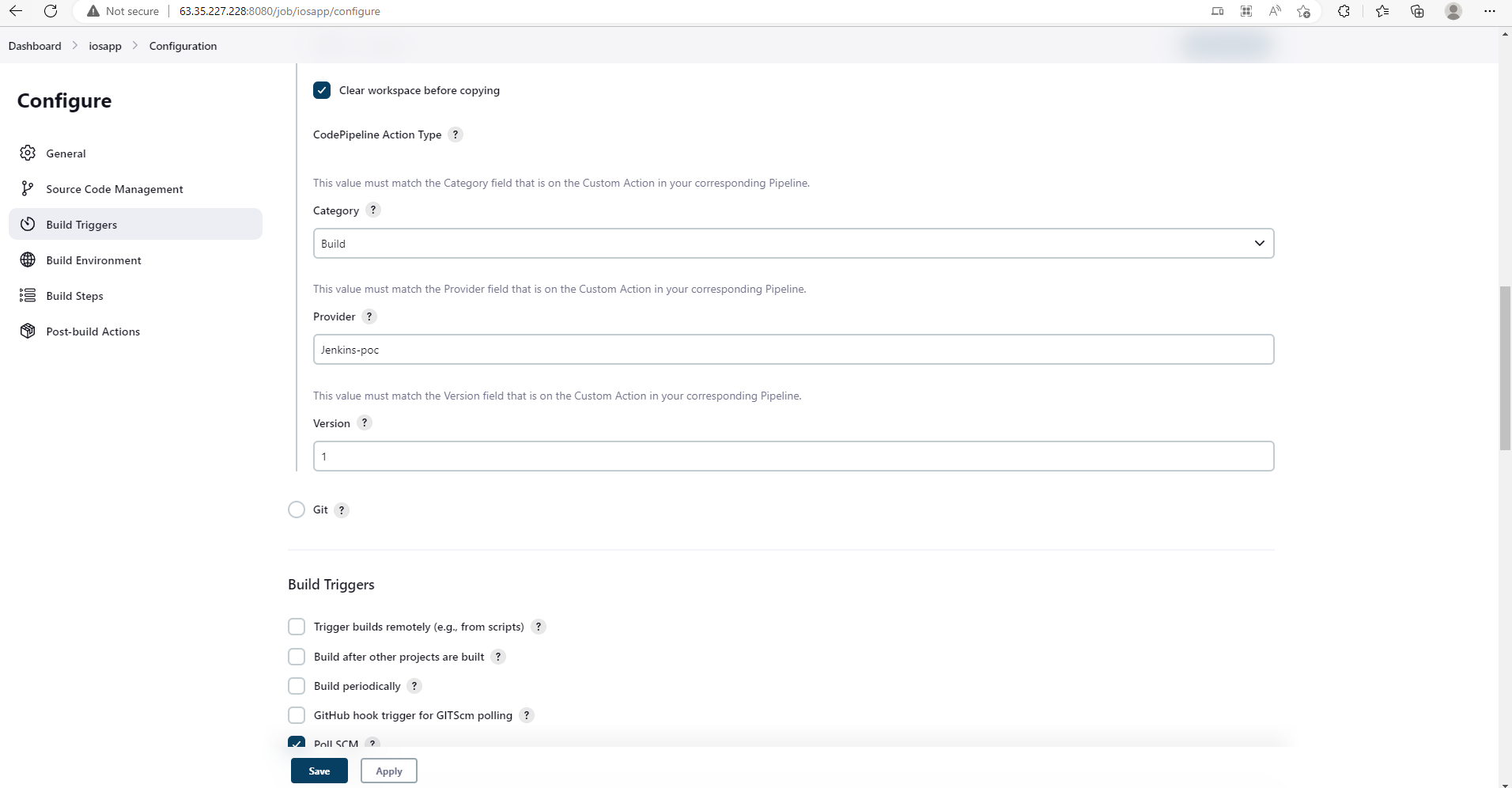
**Project Creation For iOS Build in Jenkins:**

Create a Project:

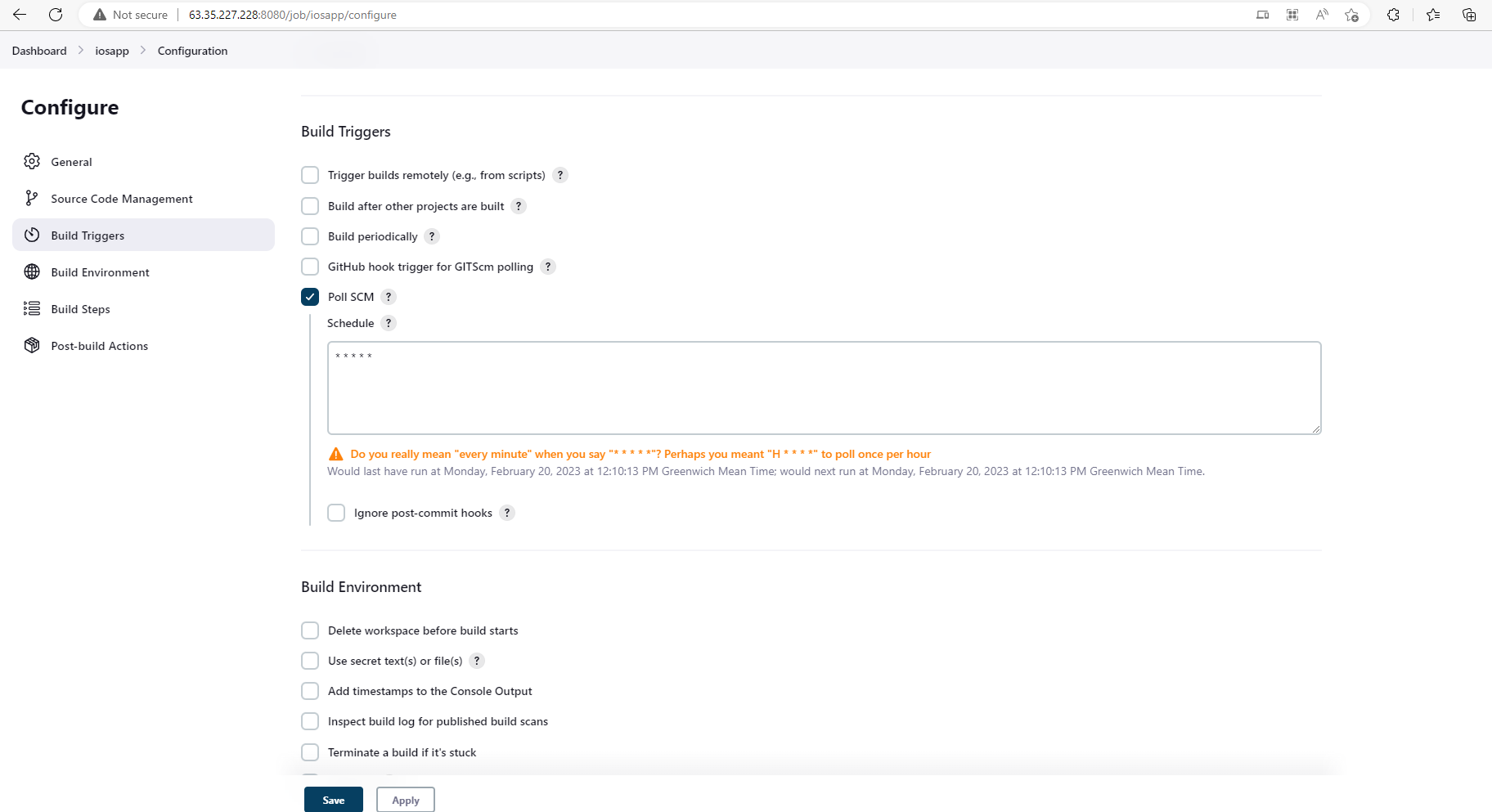
* Choose **New Item**, then **Freestyle Project**. Enter a descriptive name. This POC I uses iosapp as the item name.
* In the **Source Code Management** section, select **AWS CodePipeline** and configure the plugin as shown in the following screenshot
* **AWS region**: The region in which you want to create the CI/CD pipeline.
* **AWS access key and AWS secret key:** Create a special [IAM](https://aws.amazon.com/iam/) user and apply the AWSCodePipelineCustomActionAccess managed policy to that user. Use the access credentials for that user to configure this section.



* **Category:** Choose to **Build**. This is also used in the pipeline configuration.
* **Provider:** This example uses the name, Jenkins. It can be renamed, but take note of the name specified here.
* **Version:** Enter 1 here. This value is used in the pipeline configuration.

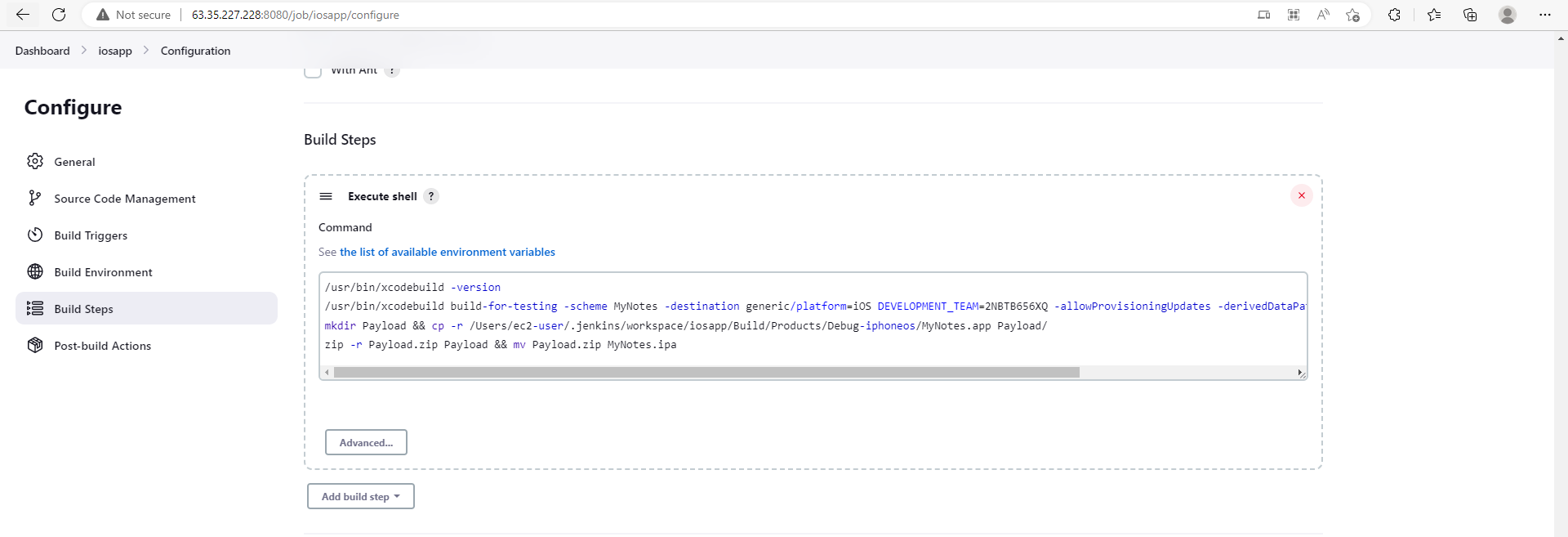


* Under **Build Triggers**, select **Poll SCM**. Enter the schedule \* \* \* \* \* separated by spaces, as shown in the following screenshot

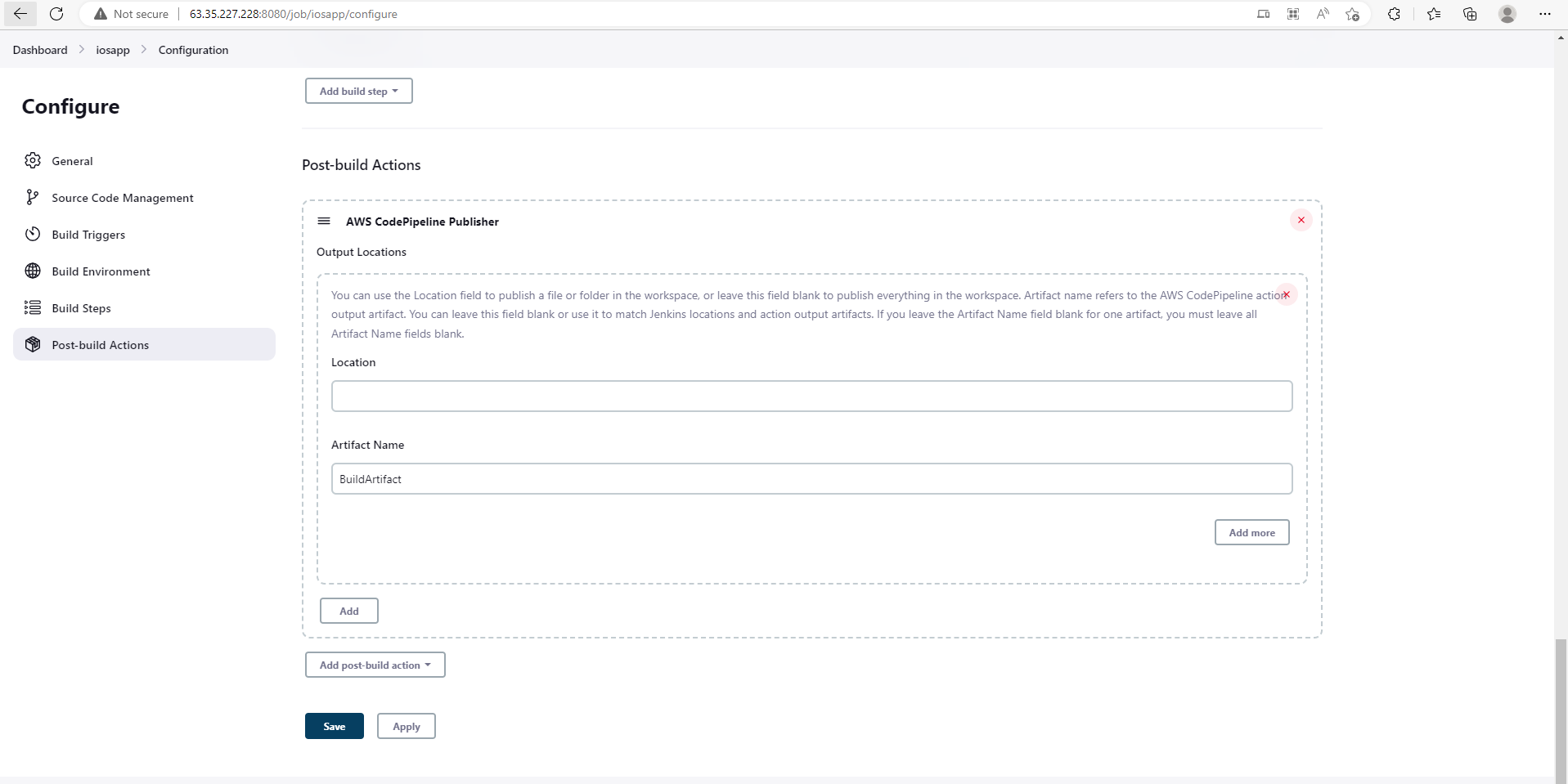


* Under **Build**, select Add **build step**, then **Execute shell**. Enter the following commands, inserting your development team ID which you noted earlier at Xcode profile configuration section. You can copy the text by clicking on the following text box.





* Under **Post-build** Actions, select **Add post-build action**, then **AWS CodePipeline Publisher**. Fill in the fields as shown in the following screenshot:



* Save the configuration.

**Create and Configure AWS Device Farm:**

In this module, you configure Device Farm to test the sample iOS app on real-world devices. We use the cloud formation template to make device farm projects and device pool. Here is the template which you can use for your project.

Note: Change parameters according to your requirement.

|  |
| --- |
| --- AWSTemplateFormatVersion: "2010-09-09" Description: Template to deploy device farm Parameters:  project:  Type: String  Default: weom  Description: Project Name  app:  Type: String  Default: ios  Description: App name  env:  Type: String  Default: poc  AllowedValues:  - dev  - poc  - qa  - uat  - prod   Description: Environment name  projectName:  Type: String  Default: iosdevices  Description: The name of the Device Farm project to Create  devicePoolName:  Type: String  Default: iosdevicespool  Description: The name of the Curated Device Pool to Create  maxDevices:  Type: String  Default: 3  Description: The Number of Devices in which you want to test your project  attribute:  Type: String  Default: "PLATFORM"  Description: The rule's stringified attribute  operator:  Type: String  Default: "EQUALS"  Description: Specifies how Device Farm compares the rule's attribute to the value  value:  Type: String  Default: '"IOS"'  Description: The rule's value Resources:  DeviceFarmProject:  Type: AWS::DeviceFarm::Project  Properties:  Name: !Ref projectName  DevicePool:  Type: AWS::DeviceFarm::DevicePool  Properties:  Name: !Ref devicePoolName  ProjectArn: !GetAtt DeviceFarmProject.Arn  Rules:  - Attribute: !Ref attribute  Operator: EQUALS  Value: !Ref value  MaxDevices: !Ref maxDevices  Description: A curated device pool for iOS app testing |

**Create and Configure CodePipeline:**

Since you have a Jenkins stage, which is considered a custom action and use the AWS cloudformation template to create your pipeline. Here is the template which you can use to deploy your pipeline.

|  |
| --- |
| --- AWSTemplateFormatVersion: "2010-09-09" Description: Template to deploy Code Pipeline  Parameters:  project:  Type: String  Default: weom  Description: Project Name  app:  Type: String  Default: ios  Description: App name  env:  Type: String  Default: poc  AllowedValues:  - dev  - poc  - qa  - uat  - prod  Description: Environment name  repositoryName:  Type: String  Default: jenkins-poc-repo  Description: The name of the repository where source changes are to be detected  branchName:  Type: String  Default: master  Description: The name of the branch where source changes are to be detected  runOrder:  Type: String  Default: "1"  Description: The order in which actions are run  version:  Type: String  Default: '1'  Description: A string that describes the action version  sourceProvider:  Type: String  Default: CodeCommit  Description: The provider of the service being called by the action  buildProvider:  Type: String  Default: Jenkins  Description: The provider of the service being called by the action  sourceCategory:  Type: String  Default: Source  Description: A category defines what kind of action can be taken in the stage  testProvider:  Type: String  Default: DeviceFarm  Description: The provider of the service being called by the action  testCategory:  Type: String  Default: Test  Description: A category defines what kind of action can be taken in the stage  testOwner:  Type: String  Default: AWS  Description: The creator of the action being called  buildCategory:  Type: String  Default: Build  Description: A category defines what kind of action can be taken in the stage  owner:  Type: String  Default: AWS  Description: The creator of the action being called  artifactStoreType:  Type: String  Default: S3  Description: The type of the artifact store  artifactStoreLocation:  Type: String  Default: jenkins-pipeline-bucket-123  Description: The S3 bucket used for storing the artifacts for a pipeline  codePipelineName:  Type: String  Default: "iosapp"  Description: Deployment Group Name  buildOwner:  Type: String  Default: Custom  Description: The creator of the action being called  jenkinsProjectName:  Type: String  Default: "iosapp"  Description: The jenkins Project Name  deviceFarmProjectId:  Type: String  Default: "0ee67ec3-8f93-49b6-baed-cb9a200bed6f"  Description: The Device Farm Project Id  devicePoolArn:  Type: String  Default: "arn:aws:devicefarm:us-west-2:007756798683:devicepool:0ee67ec3-8f93-49b6-baed-cb9a200bed6f/c05bffb1-918a-4205-bc10-fffeff108dd8"  Description: The Device Pool Arn  appName:  Type: String  Default: MyNotes.ipa  Description: App Name  appType:  Type: String  Default: iOS  Description: The app type   testType:  Type: String  Default: BUILTIN\_FUZZ  Description: The Test Type for the app Resources:  PipelineRole:  Type: "AWS::IAM::Role"  Properties:  AssumeRolePolicyDocument:  Version: '2012-10-17'  Statement:  - Effect: Allow  Principal:  Service:  - codepipeline.amazonaws.com  Action:  - sts:AssumeRole  Path: '/'  Policies:  - PolicyName: CodePipelinePolicy  PolicyDocument:  Version: '2012-10-17'  Statement:  - Action:  - 'iam:PassRole'  Resource: '\*'  Effect: Allow  Condition:  StringEqualsIfExists:  'iam:PassedToService':  - cloudformation.amazonaws.com  - elasticbeanstalk.amazonaws.com  - ec2.amazonaws.com  - ecs-tasks.amazonaws.com  - Action:  - 'codecommit:CancelUploadArchive'  - 'codecommit:GetBranch'  - 'codecommit:GetCommit'  - 'codecommit:GetRepository'  - 'codecommit:GetUploadArchiveStatus'  - 'codecommit:UploadArchive'  Resource: '\*'  Effect: Allow  - Action:  - 'codestar-connections:UseConnection'  Resource: '\*'  Effect: Allow  - Action:  - 'elasticbeanstalk:\*'  - 'ec2:\*'  - 'elasticloadbalancing:\*'  - 'autoscaling:\*'  - 'cloudwatch:\*'  - 's3:\*'  - 'sns:\*'  - 'cloudformation:\*'  - 'rds:\*'  - 'sqs:\*'  - 'ecs:\*'  Resource: '\*'  Effect: Allow  - Action:  - 'cloudformation:CreateStack'  - 'cloudformation:DeleteStack'  - 'cloudformation:DescribeStacks'  - 'cloudformation:UpdateStack'  - 'cloudformation:CreateChangeSet'  - 'cloudformation:DeleteChangeSet'  - 'cloudformation:DescribeChangeSet'  - 'cloudformation:ExecuteChangeSet'  - 'cloudformation:SetStackPolicy'  - 'cloudformation:ValidateTemplate'  Resource: '\*'  Effect: Allow  - Action:  - 'codebuild:BatchGetBuilds'  - 'codebuild:StartBuild'  - 'codebuild:BatchGetBuildBatches'  - 'codebuild:StartBuildBatch'  Resource: '\*'  Effect: Allow  - Effect: Allow  Action:  - 'devicefarm:ListProjects'  - 'devicefarm:ListDevicePools'  - 'devicefarm:GetRun'  - 'devicefarm:GetUpload'  - 'devicefarm:CreateUpload'  - 'devicefarm:ScheduleRun'  Resource: '\*'  - Effect: Allow  Action:  - 'servicecatalog:ListProvisioningArtifacts'  - 'servicecatalog:CreateProvisioningArtifact'  - 'servicecatalog:DescribeProvisioningArtifact'  - 'servicecatalog:DeleteProvisioningArtifact'  - 'servicecatalog:UpdateProduct'  Resource: '\*'  - Effect: Allow  Action:  - 'cloudformation:ValidateTemplate'  Resource: '\*'  - Effect: Allow  Action:  - 'ecr:DescribeImages'  Resource: '\*'  - Effect: Allow  Action:  - 'states:DescribeExecution'  - 'states:DescribeStateMachine'  - 'states:StartExecution'  Resource: '\*'  iosAppPipeline:  Type: "AWS::CodePipeline::Pipeline"  Properties:  ArtifactStore:  Location: !Ref artifactStoreLocation  Type: !Ref artifactStoreType  Name: !Ref codePipelineName  RoleArn: !GetAtt PipelineRole.Arn  Stages:  - Name: Source  Actions:  - Name: SourceAction  ActionTypeId:  Category: !Ref sourceCategory  Owner: !Ref owner  Provider: !Ref sourceProvider  Version: !Ref version  OutputArtifacts:  - Name: SourceArtifact  Configuration:  RepositoryName: !Ref repositoryName  BranchName: !Ref branchName  RunOrder: !Ref runOrder  - Name: Build  Actions:  - Name: JenkinsBuildAction  ActionTypeId:  Category: !Ref buildCategory  Owner: !Ref buildOwner  Provider: !Ref buildProvider  Version: !Ref version  InputArtifacts:  - Name: SourceArtifact  OutputArtifacts:  - Name: BuildArtifact  Configuration:  ProjectName: !Ref jenkinsProjectName  RunOrder: !Ref runOrder  - Name: Test  Actions:  - Name: testiosapp  ActionTypeId:  Category: !Ref testCategory  Owner: !Ref testOwner  Version: !Ref version  Provider: !Ref testProvider  Configuration:  ProjectId: !Ref deviceFarmProjectId  DevicePoolArn: !Ref devicePoolArn  AppType: !Ref appType  App: !Ref appName  TestType: !Ref testType  InputArtifacts:  - Name: BuildArtifact  RunOrder: !Ref runOrder Outputs:  CodePipeline:  Description: Code Pipeline Id  Value: !Ref iosAppPipeline  Export:  Name: !Sub "${project}-${app}-${env}-codePipelines" |

**Visit the AWS Device Farm For Testing:**

You can visit the AWS Device Farm to check your iOS App either it is compatible for multiple iOS device or not.

