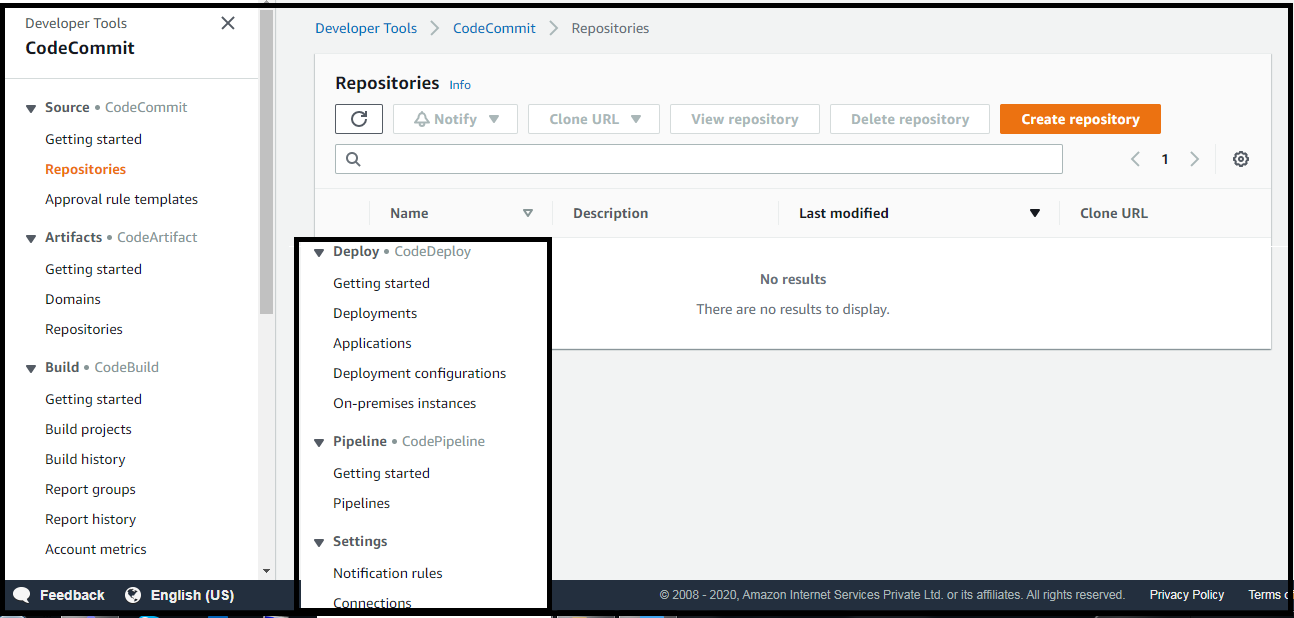
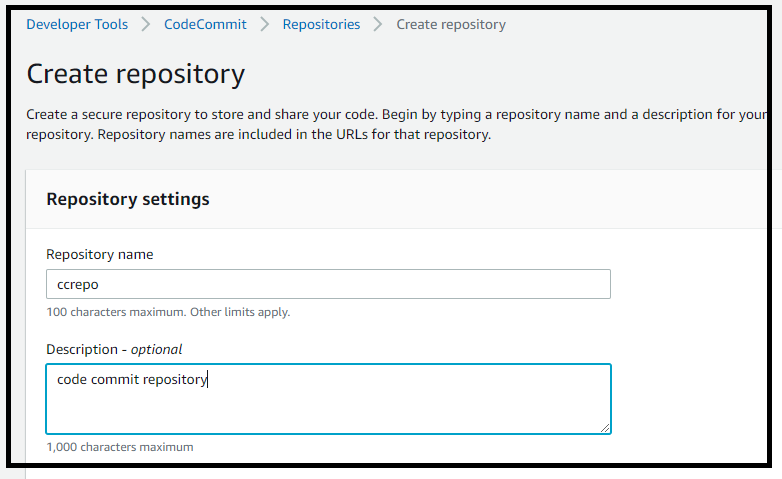
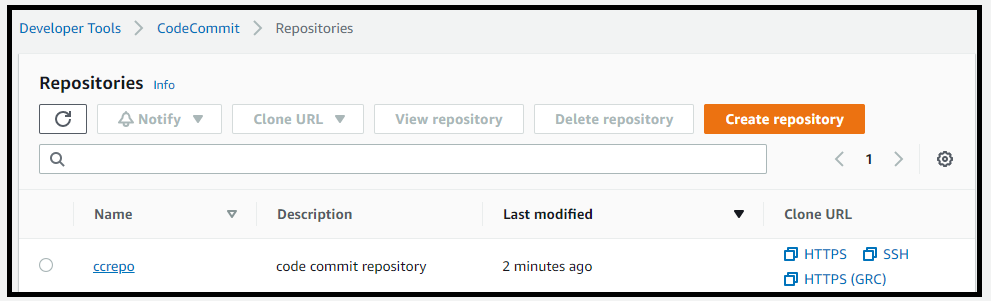
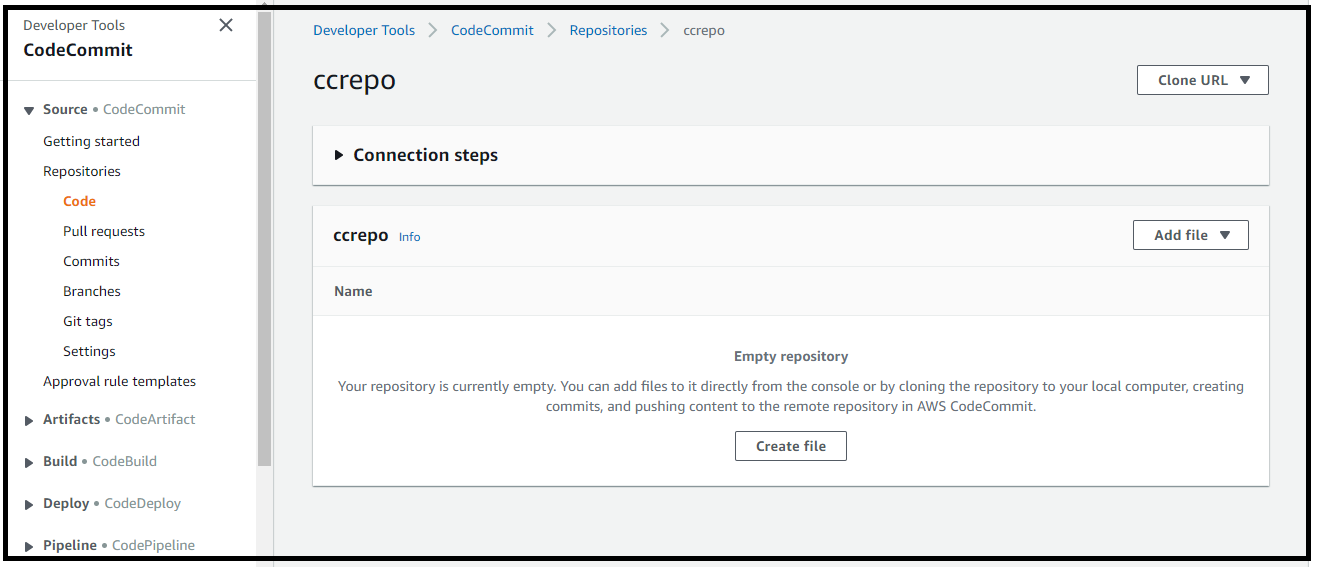
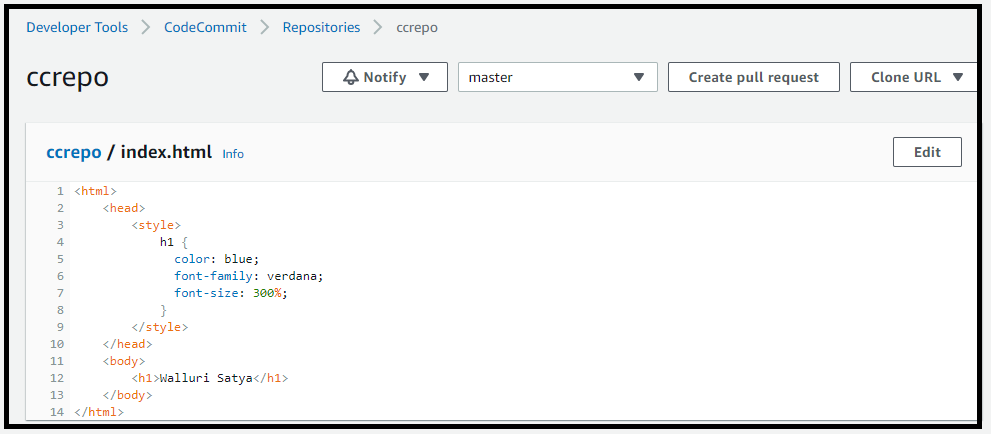
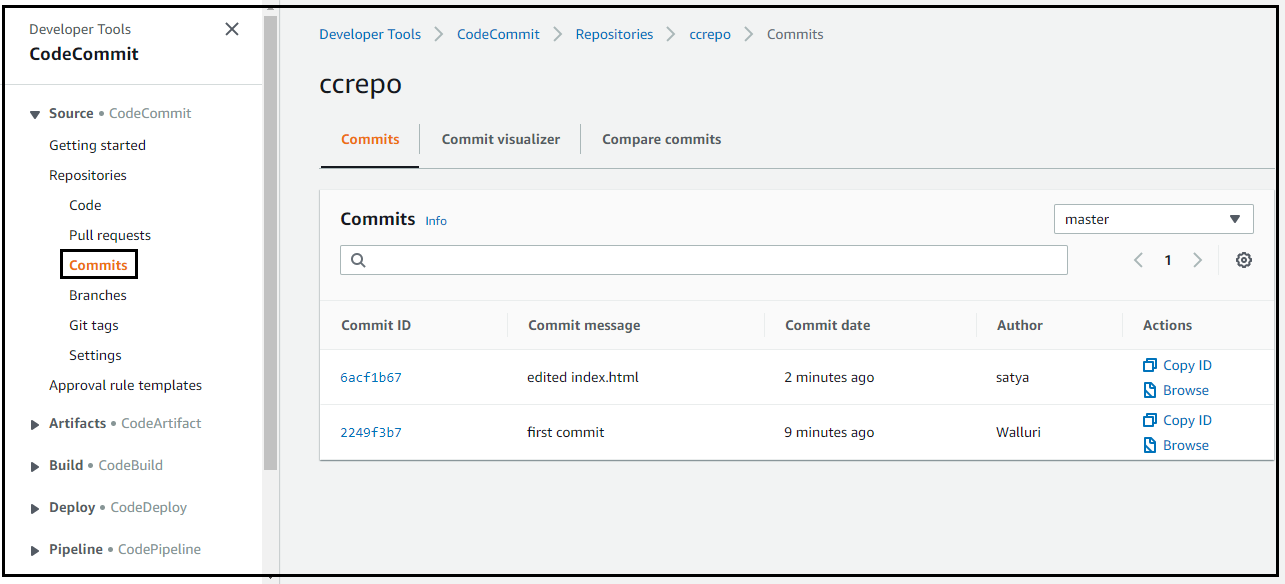
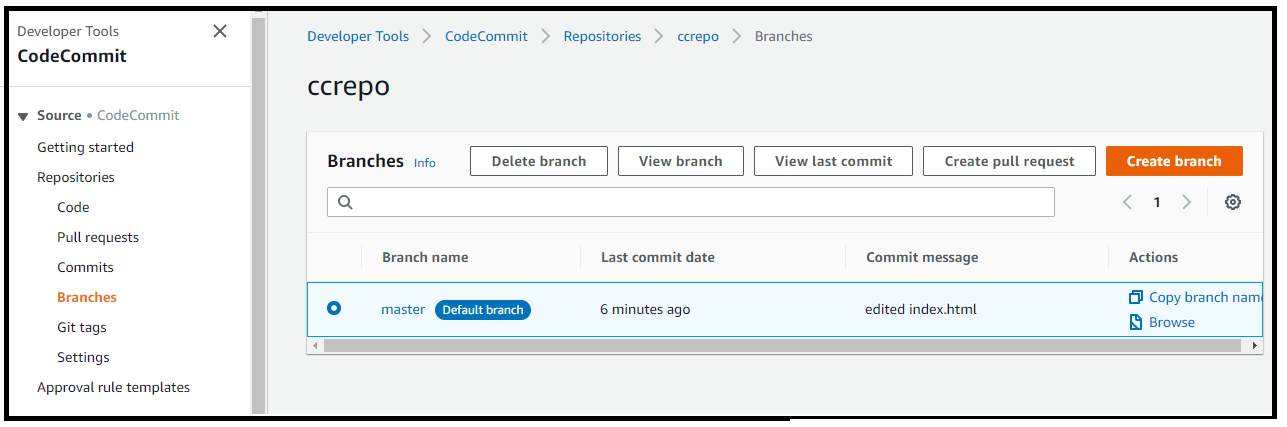
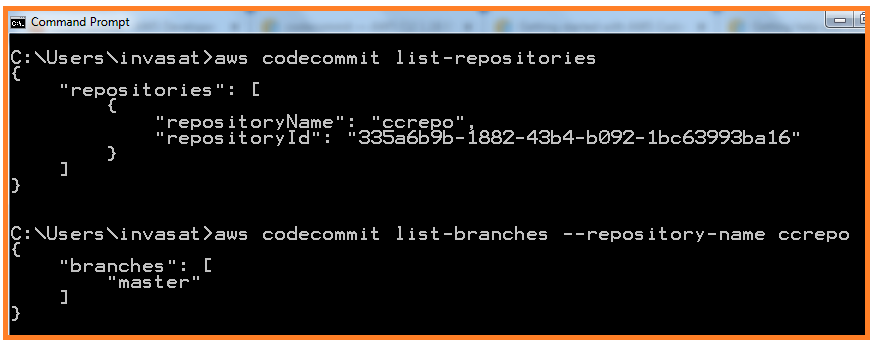
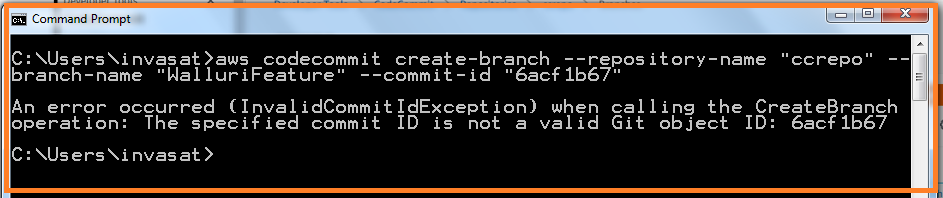
  
1. CODE COMMIT LAB.  
  
Step 1. Code commit dashboard  
  
  
Step 2. Create Repository.  


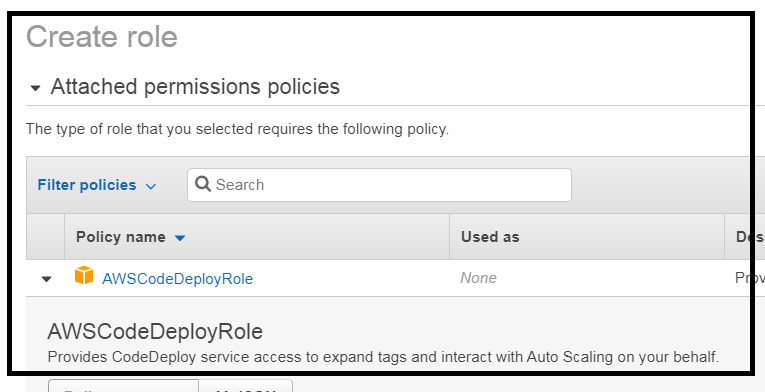
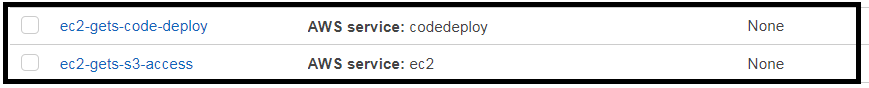
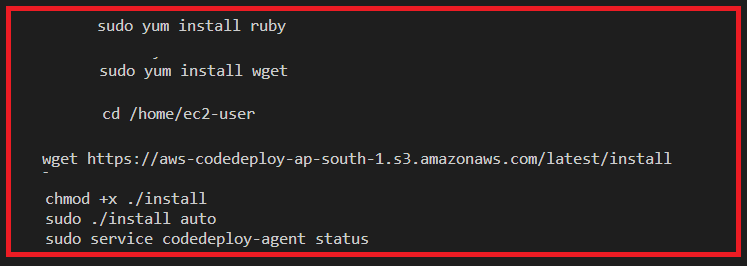
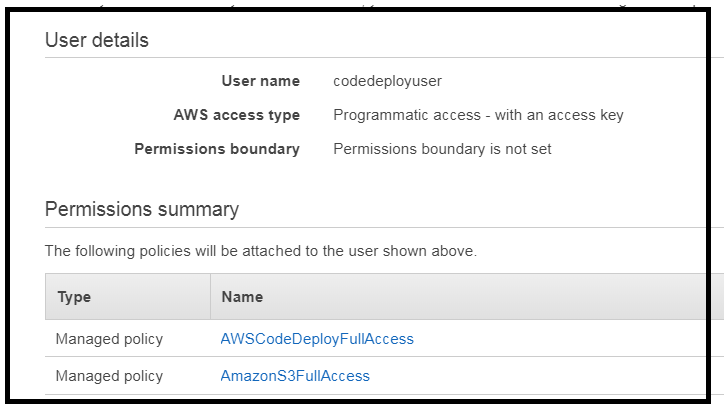
Step 3 : Dashboard after creating the repository.  


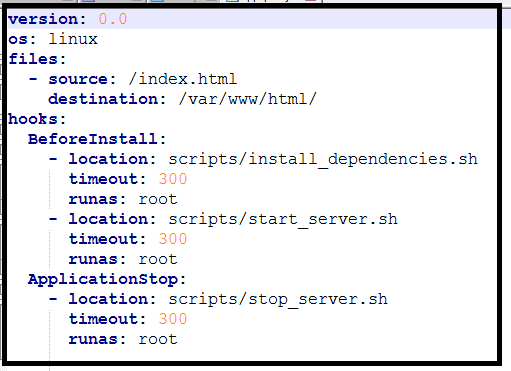
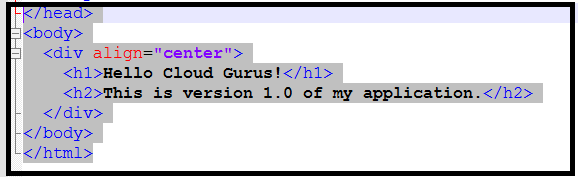
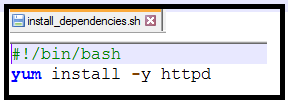
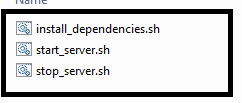
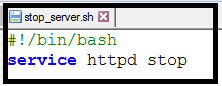
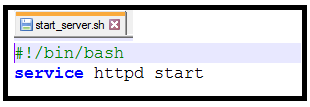
Step 4 : Dashboard of repository.  


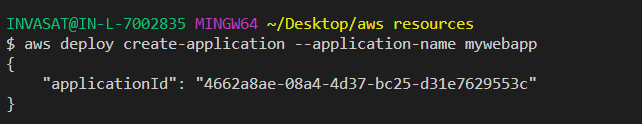
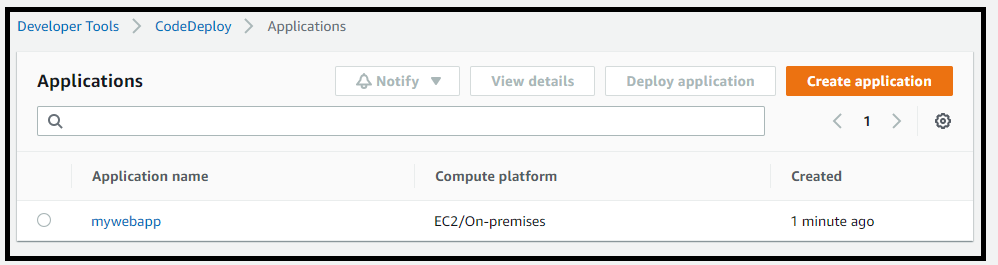
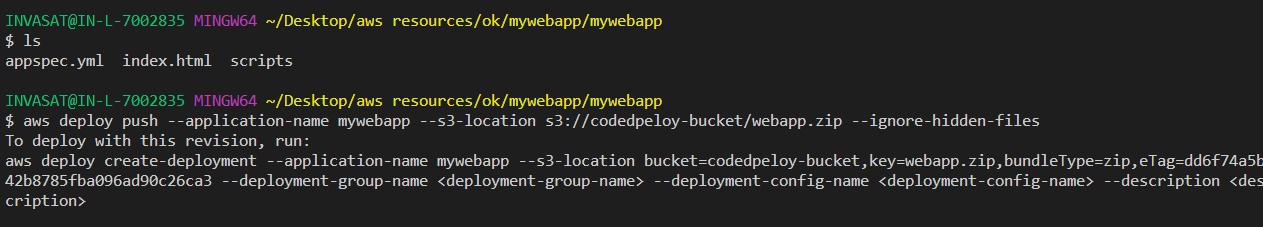
Step 5 : Create a file and make some changes.  
Note that the file is in the master branch.  
  
  
  
Step 6 : Branching.  
We would not want to make any change in the master copy of the data.  
We would create our own branch and update changes in our own branch.  
  
List Repositories and their branches.  


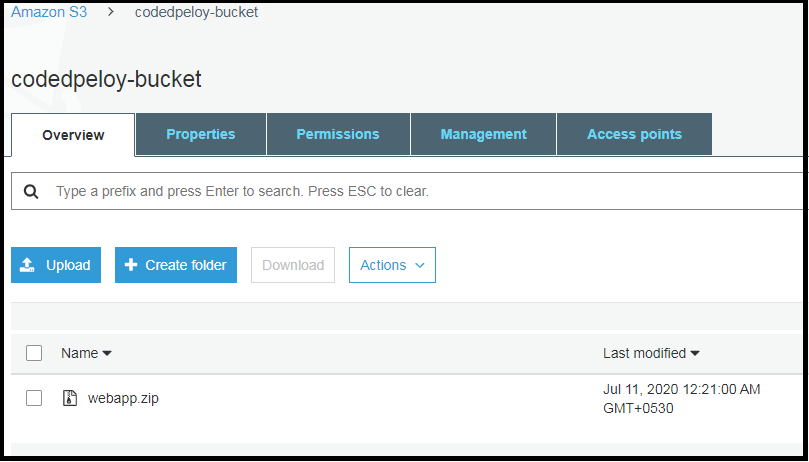
Trying to create a branch in a repo based on the commit id instead of just the branch name.

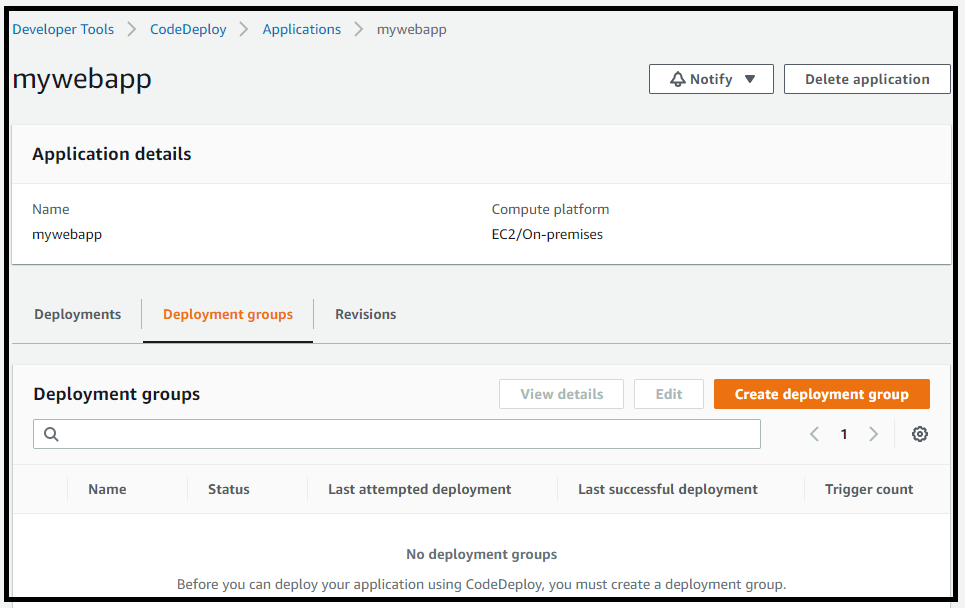
  
  
  
The same error was seen during the creation of branch via console.  
- CREATE ANOTHER BRANCH.  
- EDIT A FILE + COMMIT DETAILS  
- BRANCHES WILL LIST BOTH THE BRANCHES SO FAR WE HAVE, MASTER AND OUR BRANCH  
- GET CODE REVIEWED,RUN AUTO/MAN TESTS ON THIS BRANCH  
- CREATE PULL REQUEST – TO MERGE THE CHANGE INTO MASTER BRANCH.  
- MERGE CHANGES AND DELETE SOURCE BRANCH.  
- MASTER BRANCH WILL HAVE CHANGES.  
- WE CAN CONFIGURE NOTIFICATIONS WHEN CERTAIN EVENTS HAPPEN IN THE SOURCE CODE.NOTIFICATIONS VIA SNS AND CLOUDWATCH.

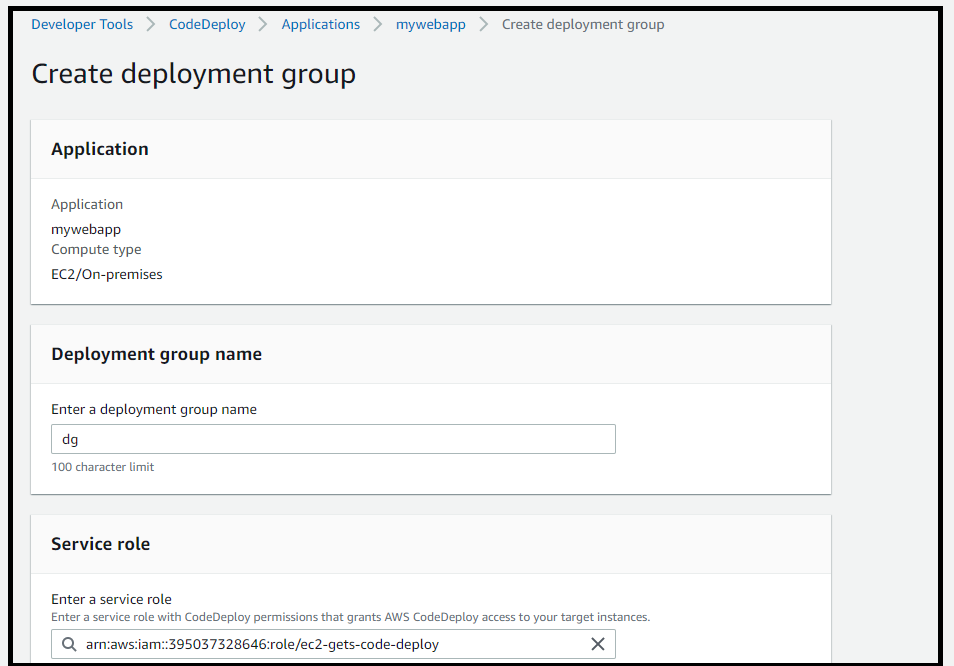
2. CODE DEPLOY LAB  
  
2.1   
First we shall create EC2 instance which has access to S3 bucket, where our code is going to be stored.  
We shall install the code-deploy agent into the EC2 instance.  
And then we will use code-deploy to deploy a simple web application.  
  
2.2   
Configure a service role for EC2 instance with S3 access, So that ec2 instance can access S3 bucket.  
Configure a service role for codedeploy as well. Code deploy makes api calls on your behalf. We are going to give code-deploy permissions to install applications on your EC2 instances.  
  
   
There is already a policy for codedeploy + EC instance.  
  
  
2.3 Create an instance with the S3 role that we just created.  
Add a tag appname with value mywebapp. Code deploy is going to use these tags to identify the instances that you are going to deploy the application to.  
Install the code-deploy agent also.  
  
  
  
Check if the code deploy agent is running.  
  
  
2.4 We need to configure our local machine as a developer machine which is going to connect into code deploy. It is going to package up our code and upload it to S3 for code-deploy to use.  
  
First we need to have a IAM user with all permissions to do that.[Create a user and attach policies (code deploy full access policy and s3 full access policy) directly]  
  
  
  
Then run aws configure with the relevant user credentials, in the local machine.

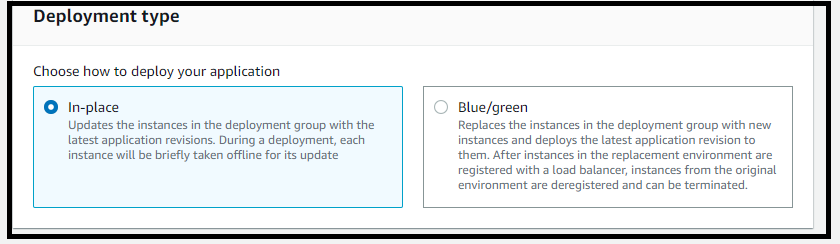
Appspec.yml : It is the configuration file used by Code Deploy.  
 I defines all the parameters needed for ‘code deploy’ deployment.  
  
os : The operating system this applies to.  
files : Any files that we want to copy to the instance. The file name and the destination in the instance.  
hooks : Hooks section defines the commands we want to run during the deployment.   
 This section defines the commands we want to run during the deployment   
 and when we want to run them.  
beforeinstall: There is a script which installs all the dependencies.  
  
HTML FILE.  
  
  
  
SCRIPTS:  
  
  
  
  
These are all the files needed to build our deployment.  
  
These files need to be in the S3 bucket.  
  
Bundle up our application files.  
Create our application in code deploy.  
Upload application code bundle into S3 bucket.  
  
Run AWS Configure in local machine.

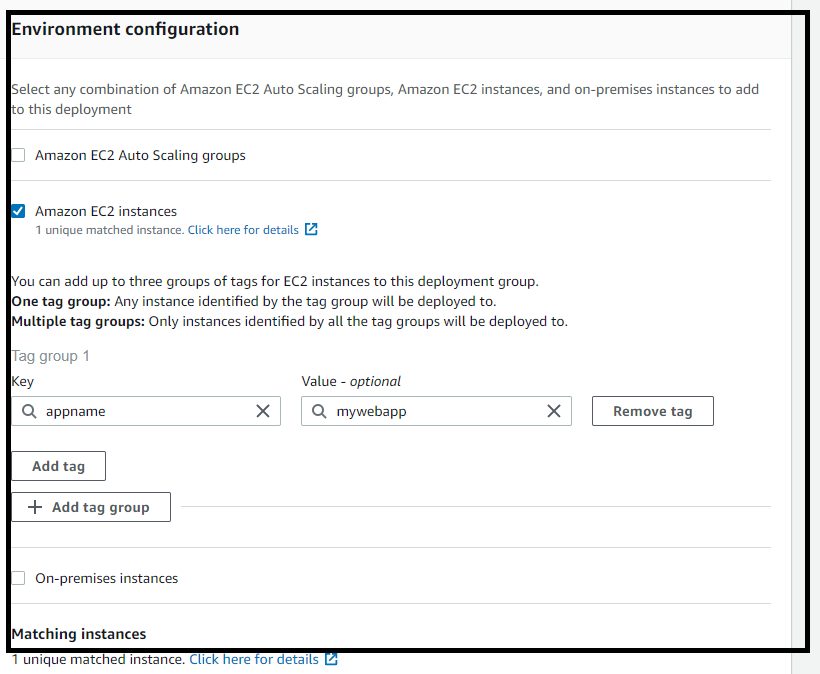
Create the application within code deploy.  
  
  
  
  
Push our application file into the S3 bucket.  


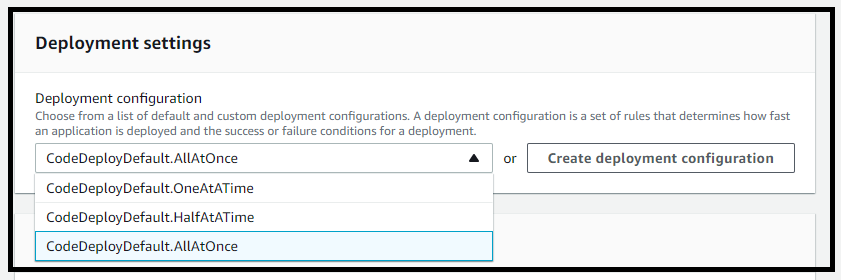
  
This would have created our application within code deploy and pushed it in to our s3 bucket.

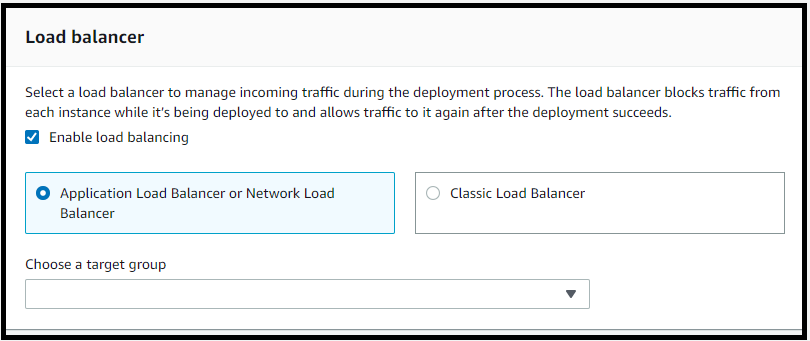
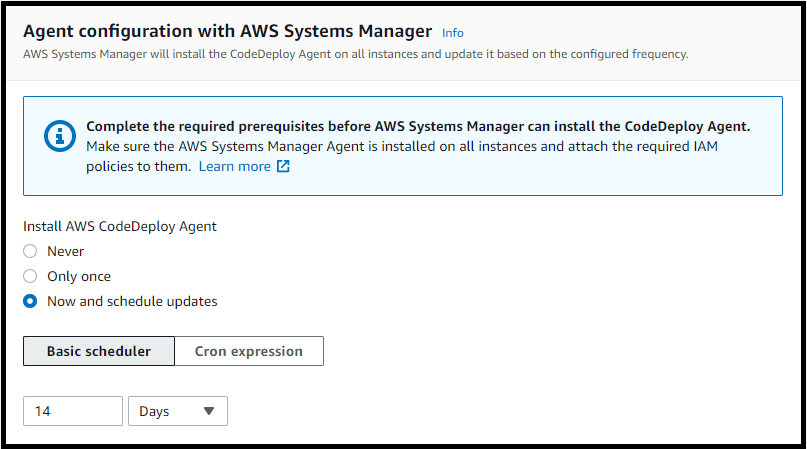
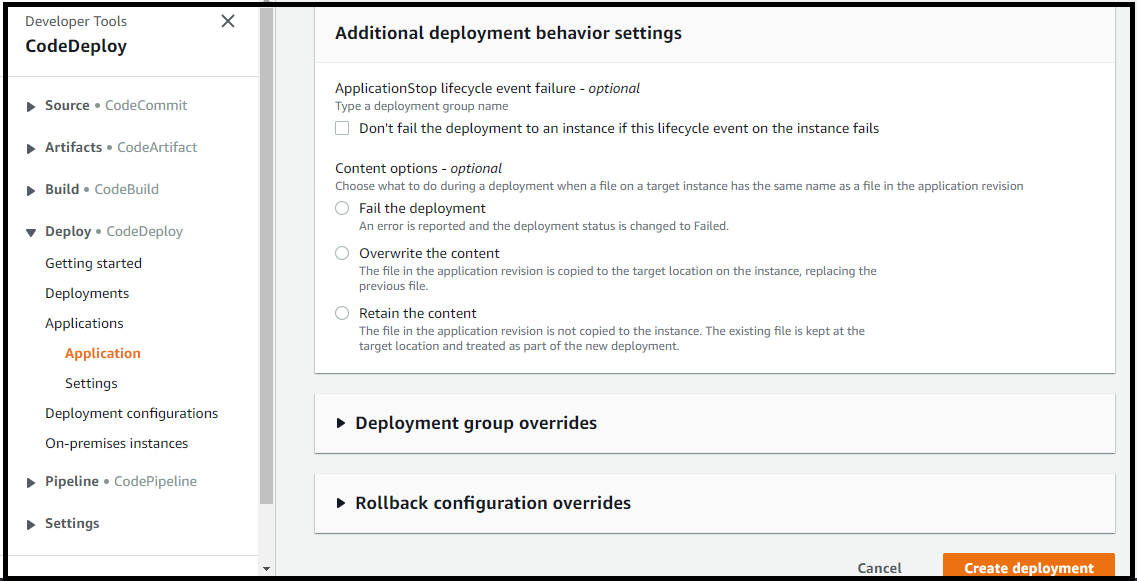
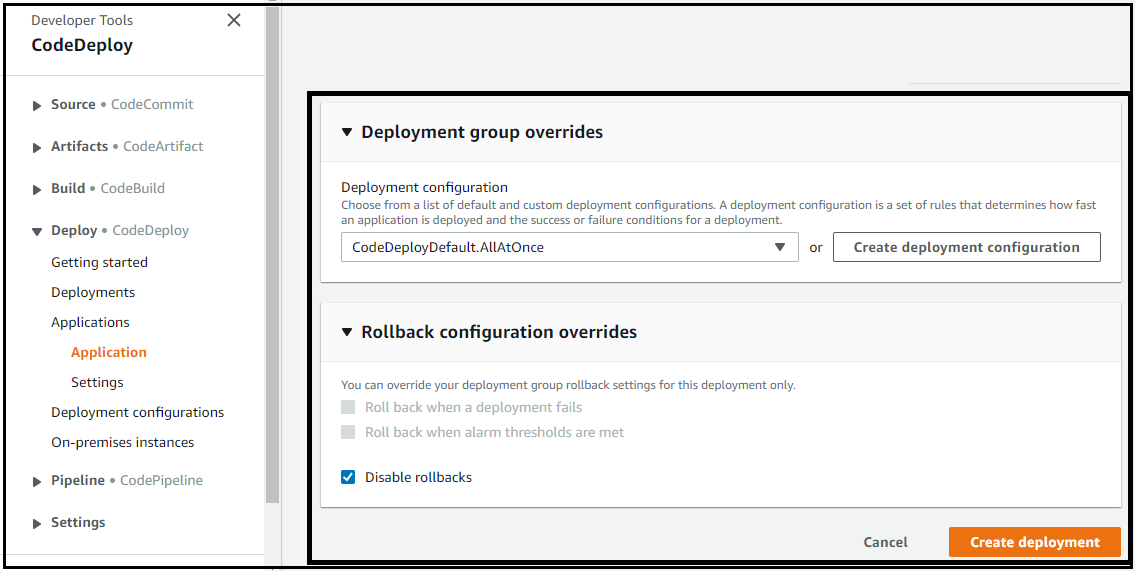
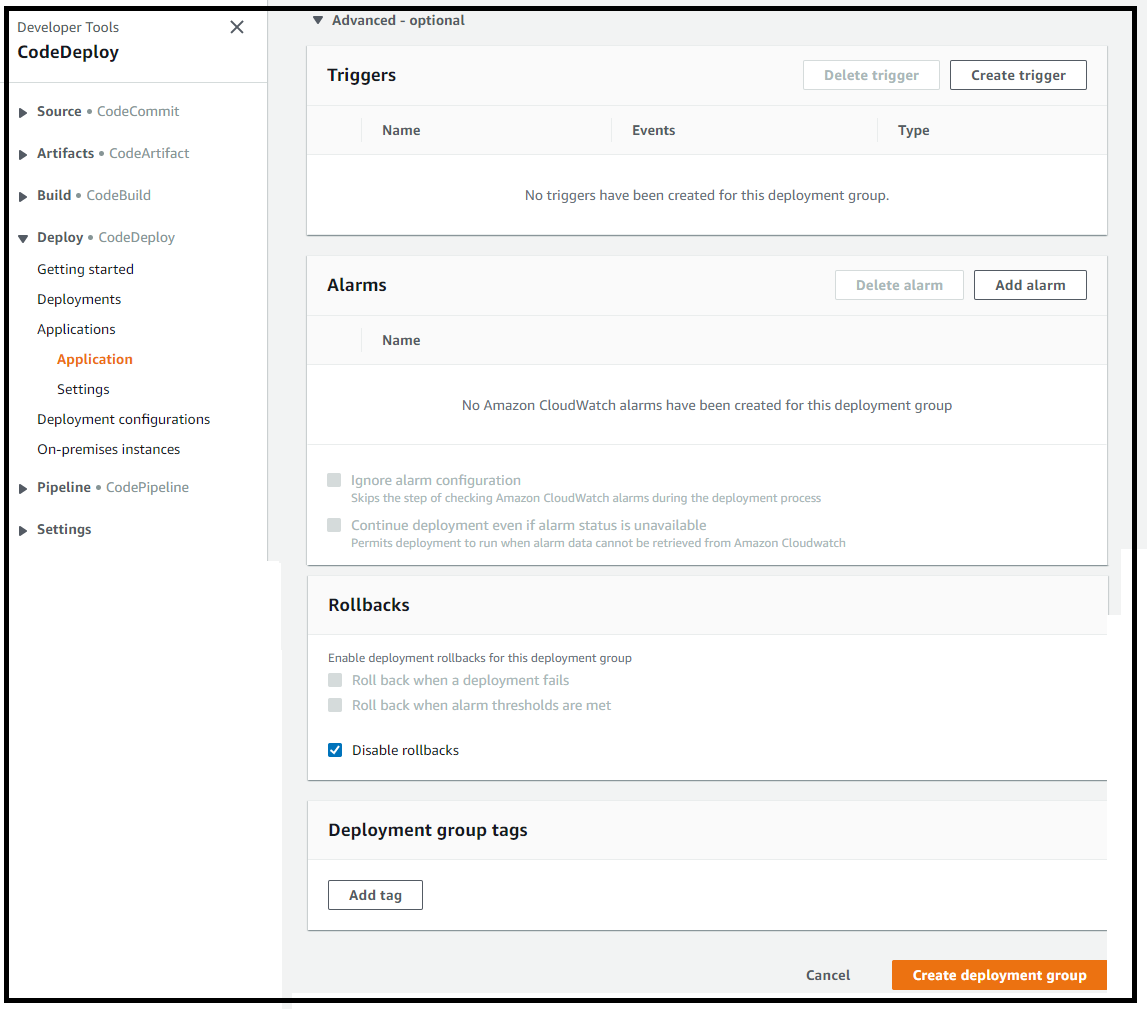
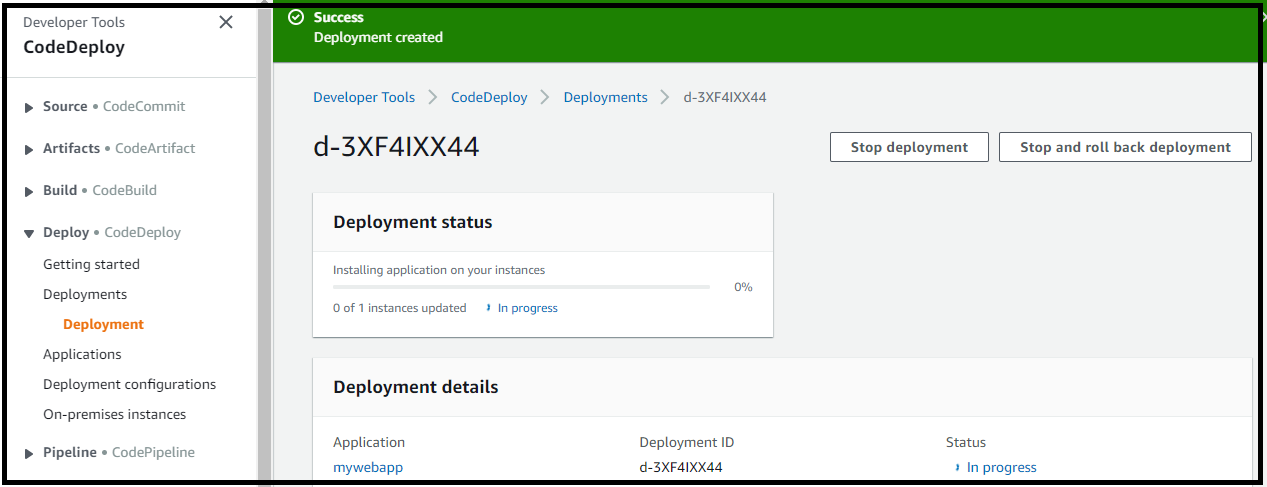
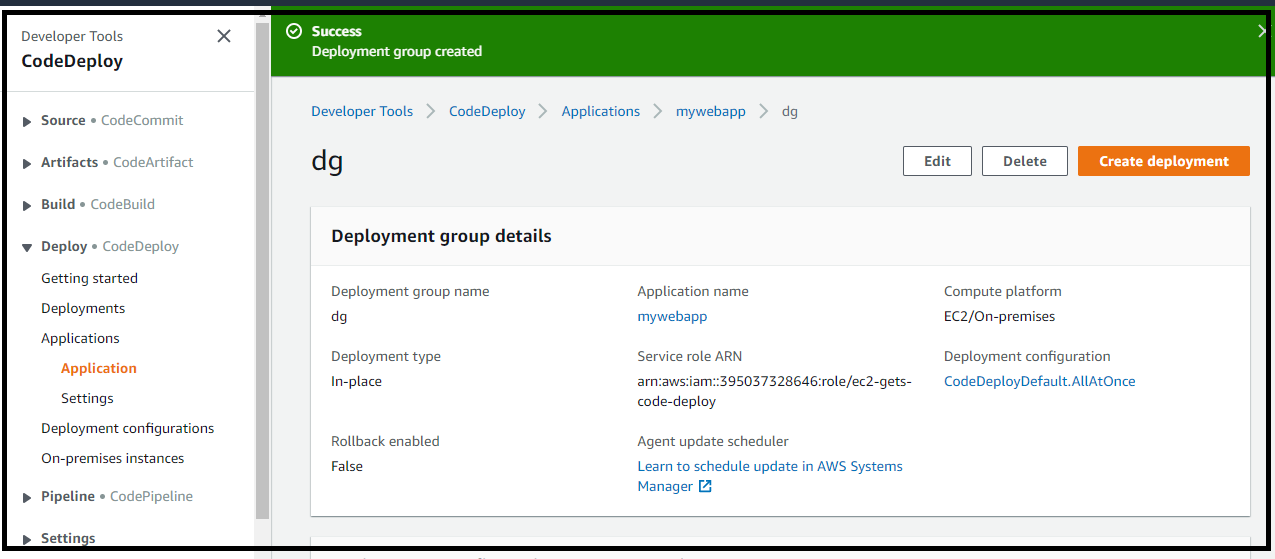
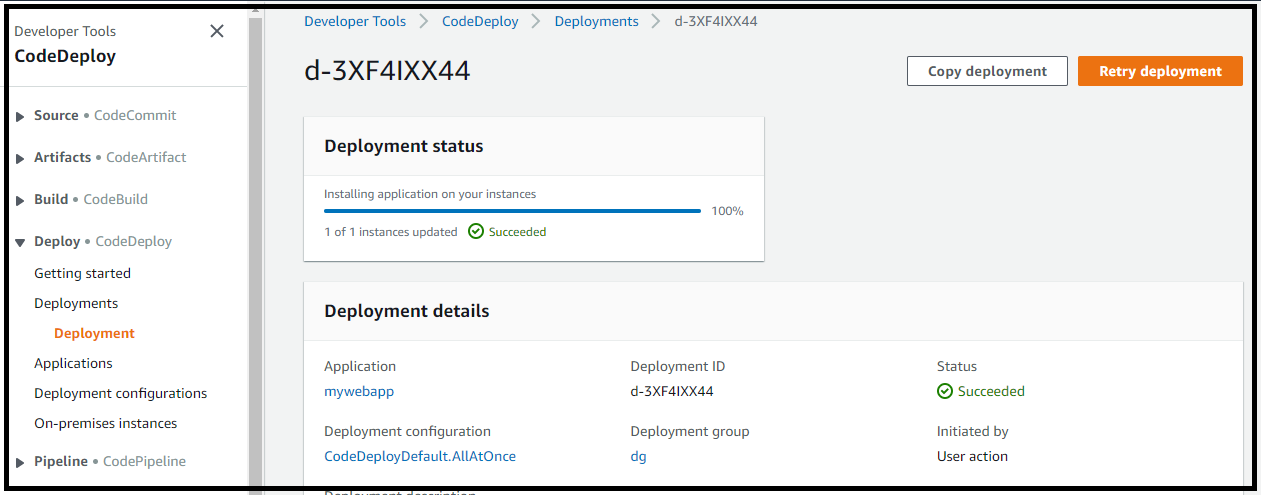
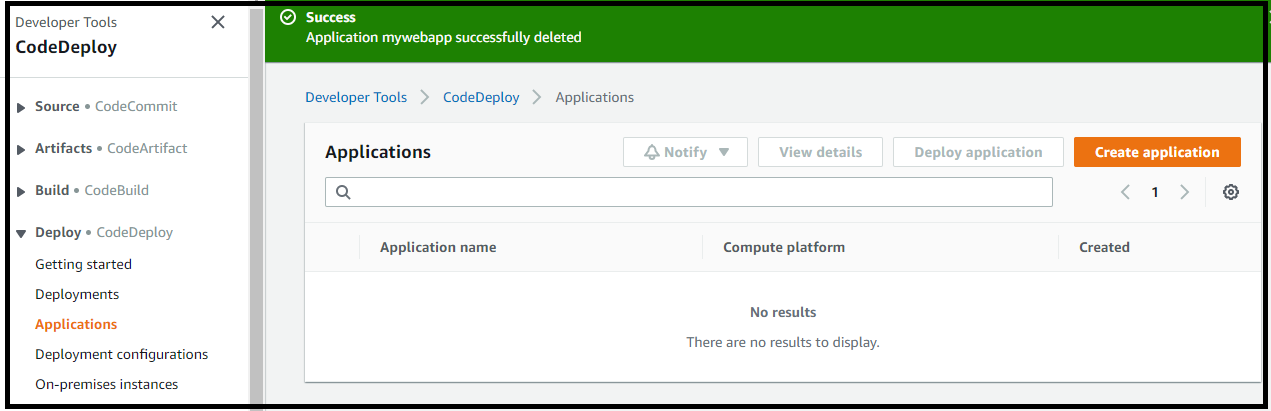
Now we are ready to deploy our application using code deploy.  


Deployment group : the group of hosts that we are going to deploy to.  
We need to use the service role that code deploy is going to use to access our target instances.  


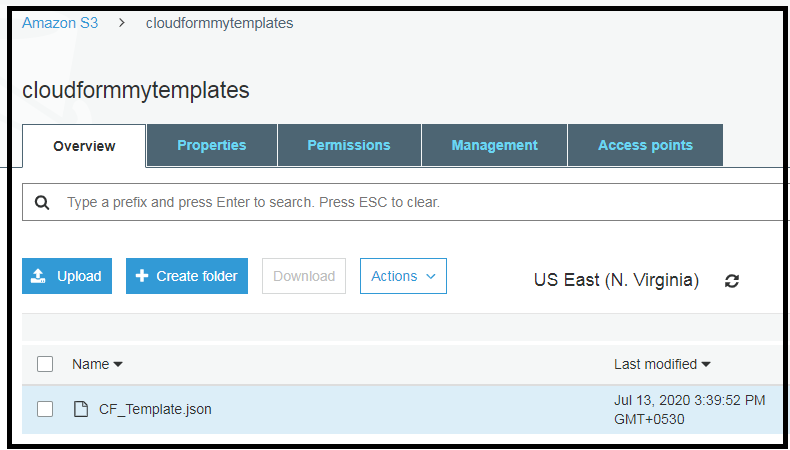
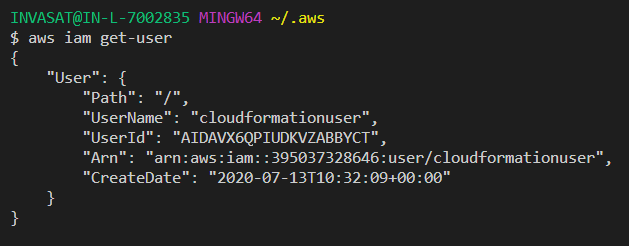
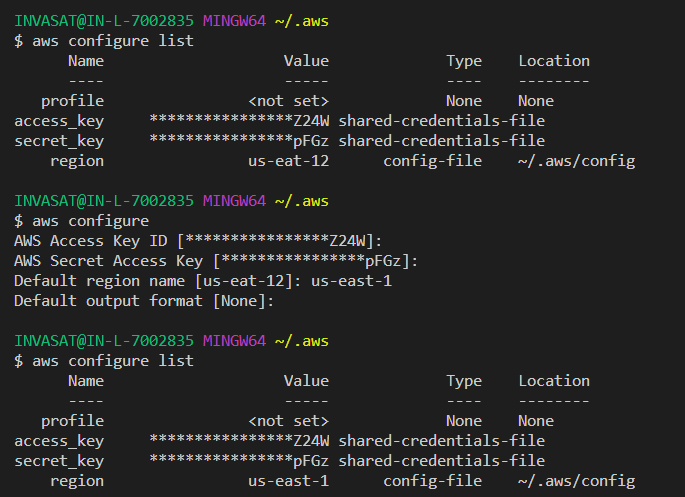
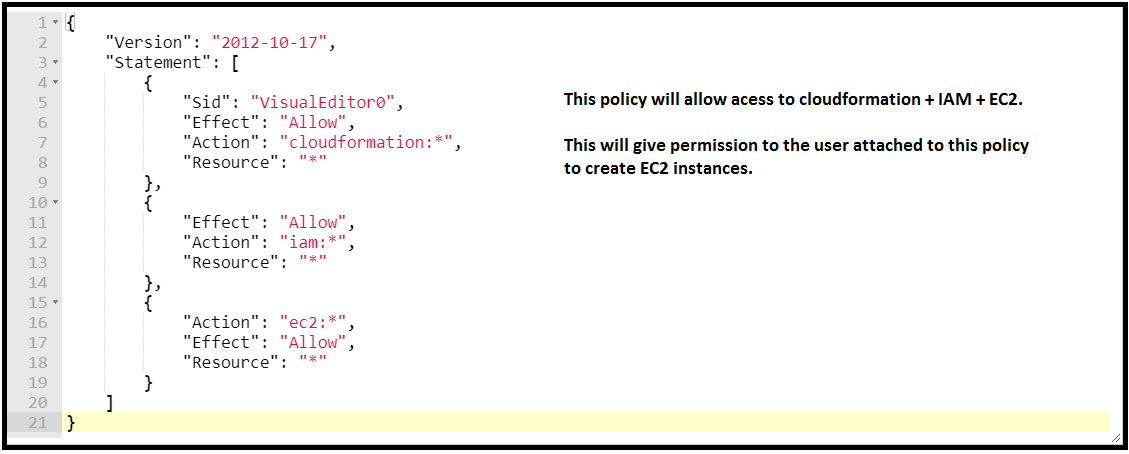
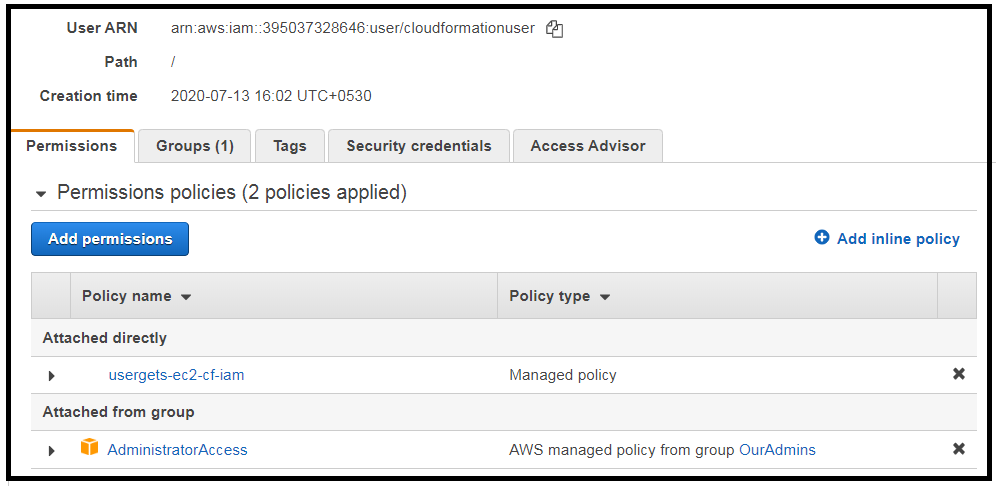
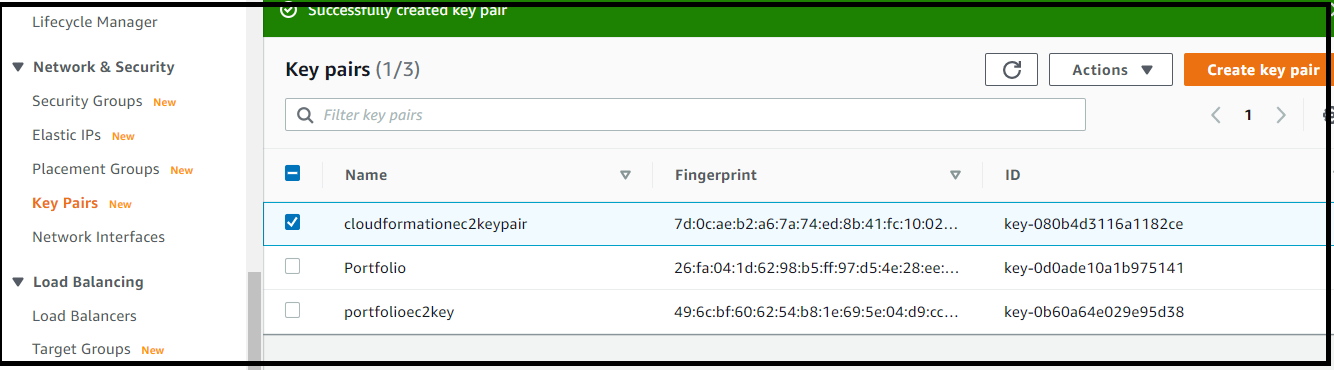


Which instances you want to deploy using the instance tags.  
  
  
We can choose to deploy our application to all the instances at once / one at a time / half at a time.



We can use the load balancer to manage the incoming traffic and manage the load across new and the original deployment.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
3. Code deploy Theory  
  
3.1 Code Deploy Deployment Approaches  
In-Place : Application is stopped in each instance and the new release is installed. It is also knows as rolling update.  
Rollback : if we decide to go back to the previous version, it is not a easy as we need to re deploy the previous version which is time consuming.  
  
Blue / Green deployment : New instances are provisioned and the new release is installed on the new instances. Blue represents the active deployment and green is the new release.  
Rollback : Code deploy provisions new instances which are independent from the old/production environment. The new revision is installed in the new instances and they are registered with the elastic load balancer and traffic is routed away from the environment. When you want to rollback in a blue green deployment, we just set the load balancer to direct the traffic to the old environment. By just re-registering the old environment with the load balancer + de-register the green.  
  
3.2 Code Deploy appspec.yml file.  
It is a configuration file which defines the parameters which are going to be used during a code deploy deployment.  
With deployments to EC2 instances and on premises systems the app spec should be written in yml only.  
With a Lambda based deployment either a yml or json are supported.  
  
3.3 appspec yml file structure.  
version : allowed value is 0.0. And is reserved for future use.  
os : The type of operating system that you are deploying to.  
files : Relates to configuration files or packages which are going to be used during the deployment.  
we use this section to define the location of files which need to be copied and where there should be copied to during the deployment.  
hooks : Often refereed as lifecycle event hooks. These are scripts which needs to run at set points during the deployment life cycle. They have a specific run order. Example : You might want to provide a script which will uzip your application file prior to the deployment. You might want to run some functional tests on a newly deployed application. You might want to run some scripts for dealing with load balancer such as registering and de-registering of instances with a load balancer.And for all of these scripts we want be in control of exactly when code deploy runs them. We want to run them at specific points in the deployment life cycle , with the help of life cycle event hooks.  
  
3.4 folder setup.  
Root of the folder : appspec yml file. [Deployment will fail if this file is not present here + Relative paths of other folders must be mentioned wrt to the appspec file location]  
Our scripts, Config, source contents are organized into their own folders.  
  
3.5 Code Deploy Life Cycle Event Hooks.  
  
The life cycle event hooks define scripts that we want to run + the order that we want to run them in during our deployment.  
  
The life cycle event hooks – They run in a specific order, called the **run order**.  
  
Life cycle event hooks for an in-place deployment is discussed below.  
  
PHASE1 : De-register instances from a load balancer.  
PHASE2 : All the activity that is needed to deploy the application itself.  
PHASE3 : Re-Registering our instances with load balancer.  
  
Life Cycle Hooks.  
  
BeforeBlockTraffic : This hooks covers all the tasks that you want to run on instances, before they are de-registered from the load balancer.  
BlockTraffic : This is where we de-register our instances from the load balancer, by including the scripts related to that.  
AfterBlockTraffic : Any tasks you want to run on your instances after they are de-registered from load balancer.  
  
ApplicationStop : This is all about gracefully stopping the application by running certain scripts.[With an in-place deployment you would want to shut down the app before doing anything]  
DownloadBundle : This is where the code deploy agent copies the application revision files/ or the new version of you application to a temporary location.  
BeforeInstall : You would want to execute any pre-installation scripts that you want to run. Example take backup of files or decrypt files from the download bundle.  
Install : These have scripts which install you application. Like copying the download files from temporary location to the final location.  
AfterInstall : You might want to run any post installation scripts. You might want to change file permissions or update configuration files. i.e any scripts you want to run after the installation has been done.  
ApplicationStart :At this point we start up any services which were stopped during application stop.  
ValidateService : Any tests you want to run to confirm that the application is working as expected.  
  
BeforeAllowTraffic : Any tasks that you want to run on your instances, before they get registered with the load balancer.  
AllowTraffic : This is where we register our instances with the load balancer.  
AfterAllowTraffic : Any tasks that you want to run on your instances after they have been registered with the load balancer.

4. CODE PIPELINE  
  
Code-pipeline is a fully managed CI/ CD service, which orchestrates the build test and deployment stages of your software release process.  
  
we can configure code-pipeline to trigger the pipeline when ever there is a change to your code.  
Ex : We can use code-commit to trigger the pipeline.  
  
It also integrates with other 3rd party tools like github , Jenkins etc.  
  
Code-pipeline also integrates with elastic beanstalk, cloud formation, Lambda and Elastic Container Service.  
  
Code pipeline workflow :  
- define workflow  
- When change is detected in code repository, code build comes in and compiles the source code and runs any tests and produces packages to deploy your software.  
- Code Deploy comes in and then the newly built application is then deployed into staging/production environment.  
  
5. Code pipeline lab.  
stage 1 : we shall use cloud formation to create an EC2 instance. We shall upload version 1 of our application into S3 bucket. And then we will deploy our application into EC2 instance using code deploy.  
  
Stage 2 : We are going to build a code pipeline and manually trigger code deploy to update our application to version 2.  
  
Stage 3 : Finally we are going to configure an automated pipeline which is going to be triggered when we add a new version of our code to the S3 bucket. This will automatically trigger a cloud watch event which will trigger code pipeline to re-deploy our code to EC2 instance using code deploy.

5.1 Create EC2 instance with code deploy agent installed using Cloud formation template.  
 - To use a cloud formation template we need to store it an S3 Bucket.  
Cloud formation always sources its templates from S3.  
  
  
  
Create a IAM user ‘**cloudformationuser’** and add him to a group which has AdministratorAccess policy attached.  
This is the user we have used to configure the aws command line with.  
  
  
Change to the region which you have picked.  
  
  
  
The command line is set up with this user, An inorder for this user to use cloud formation to launches EC2 instances and also create security groups – we also need to add additional permissions for our user.  
  
Create a policy and give it a name ‘usergets-ec2-cf-iam’  
  
  
Attach that policy to the current user.  
  
  
Create a Key pair from the EC2 dashboard, and download the pem file.  
  
  
  
Now Run the below command to create an EC2 instance using the Cloud formation template which is in S3.  
Command in AWS console.  
aws cloudformation create-stack --stack-name CodeDeployDemoStack \

--template-url https://cloudformmytemplates.s3.amazonaws.com/CF\_Template.json \

--parameters ParameterKey=InstanceCount,ParameterValue=1 \

ParameterKey=InstanceType,ParameterValue=t2.micro \

ParameterKey=KeyPairName,ParameterValue=cloudformationec2keypair \

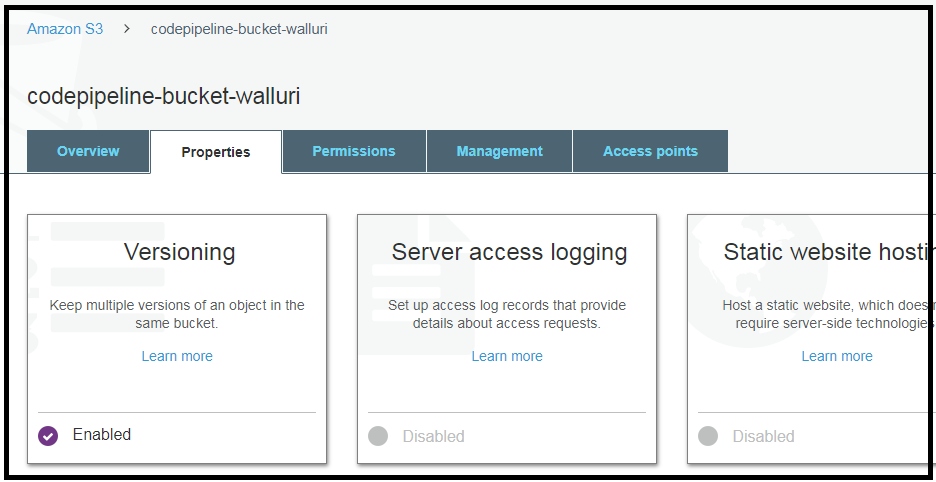
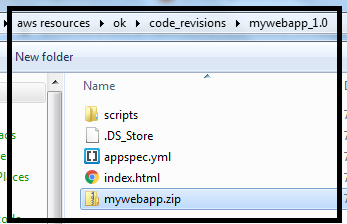
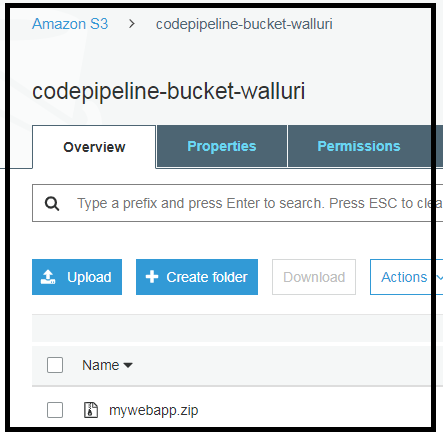
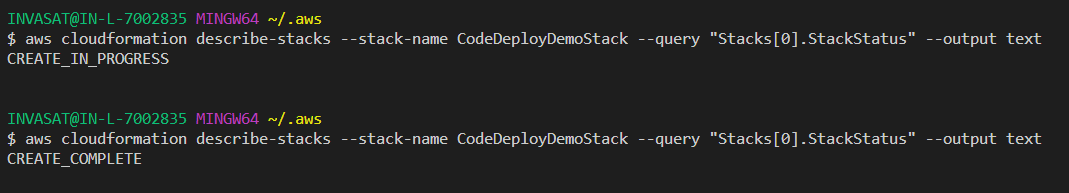
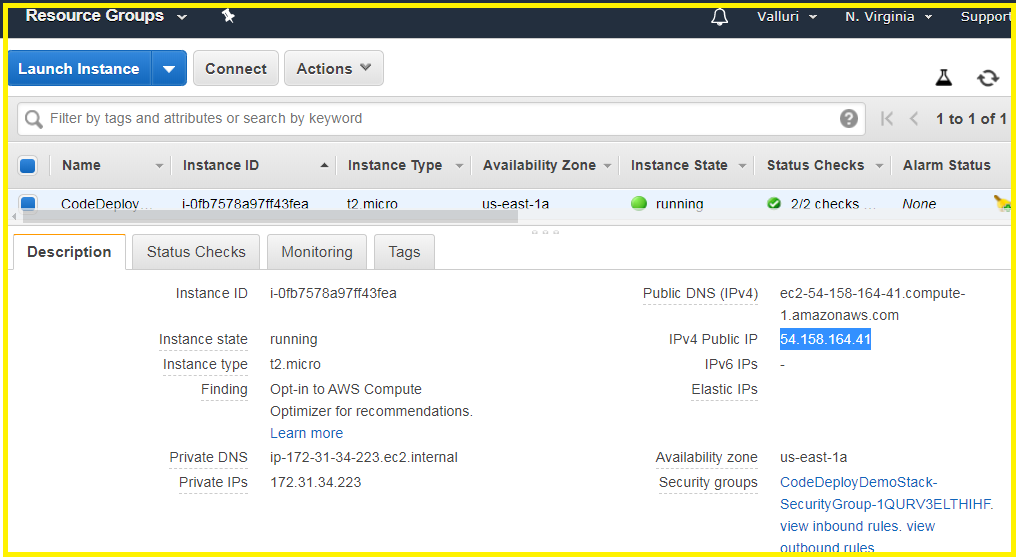
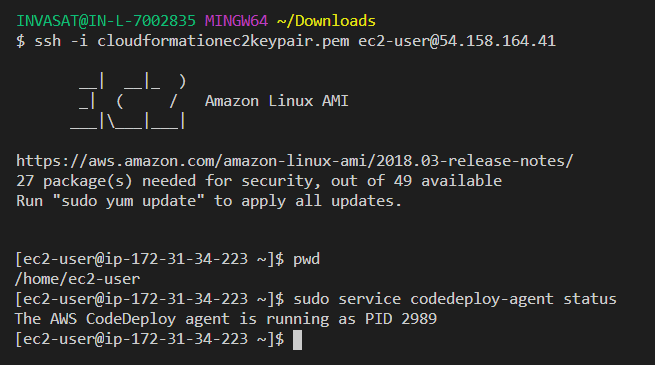
ParameterKey=OperatingSystem,ParameterValue=Linux \

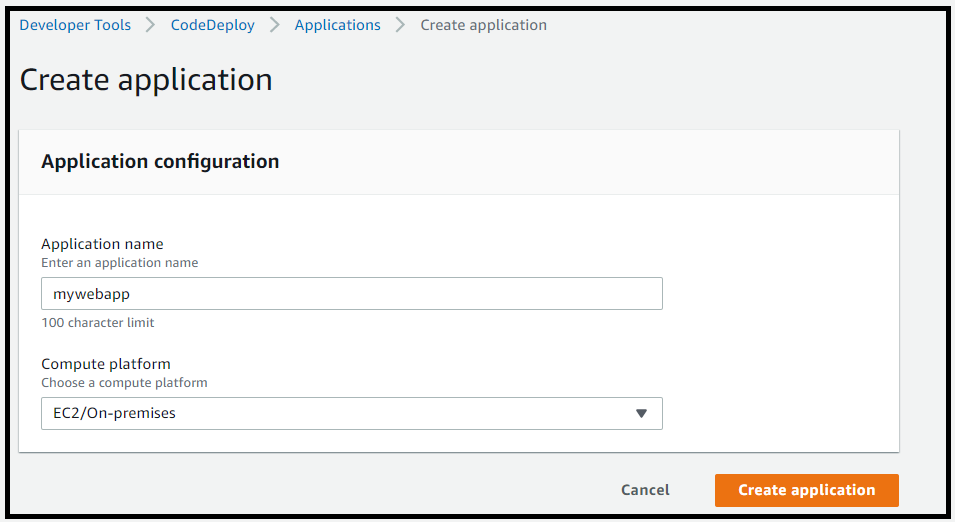
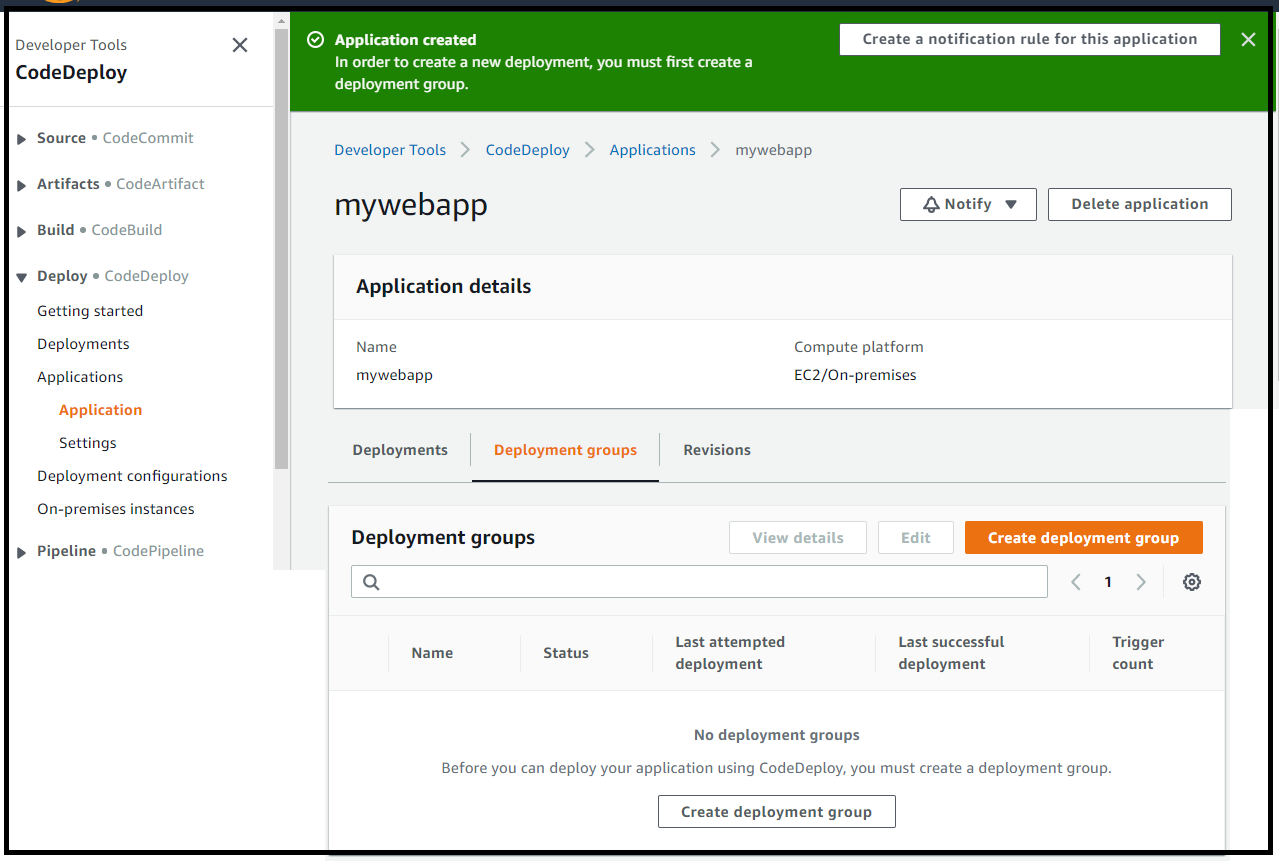
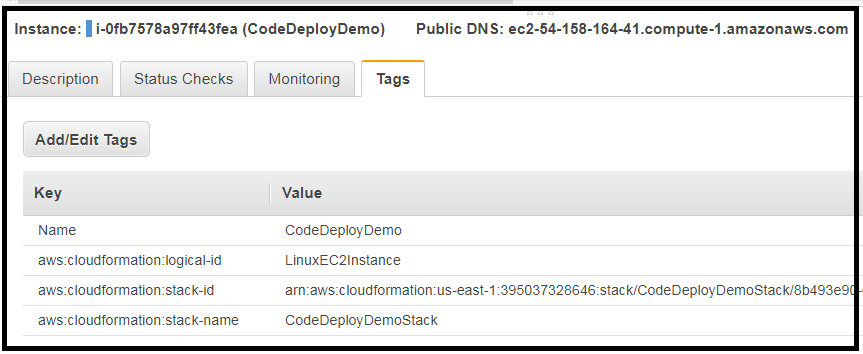
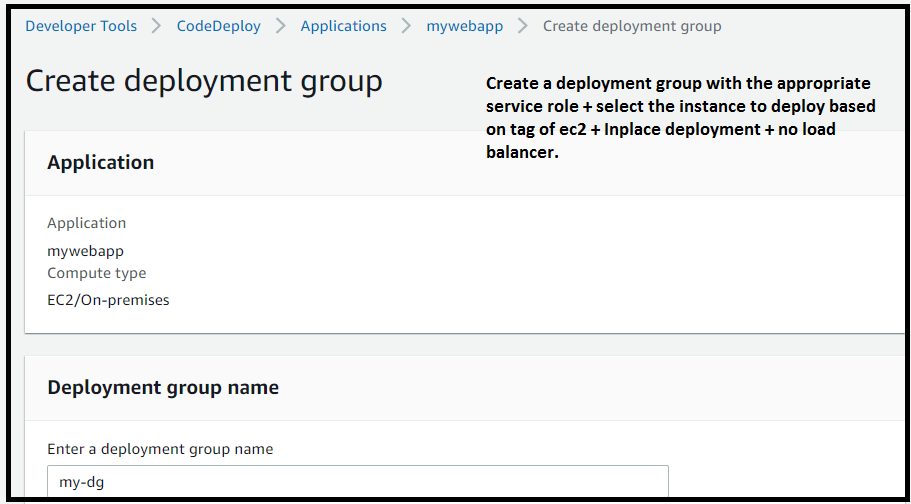
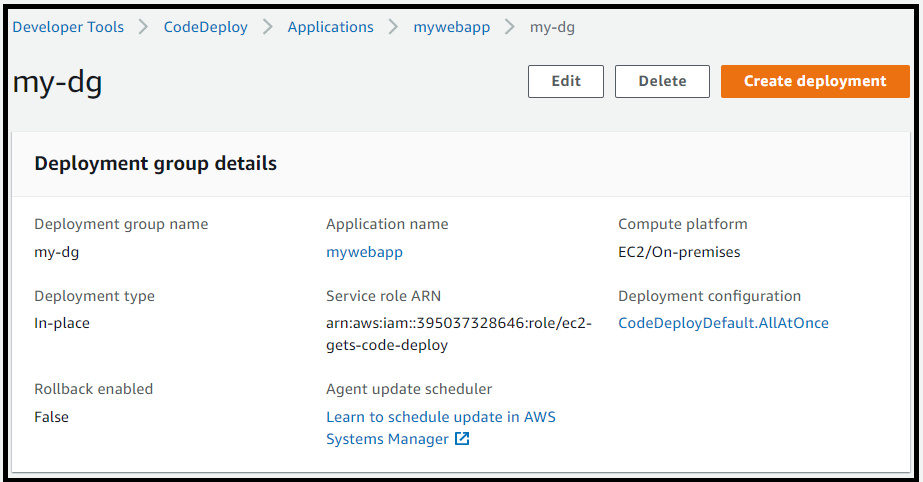
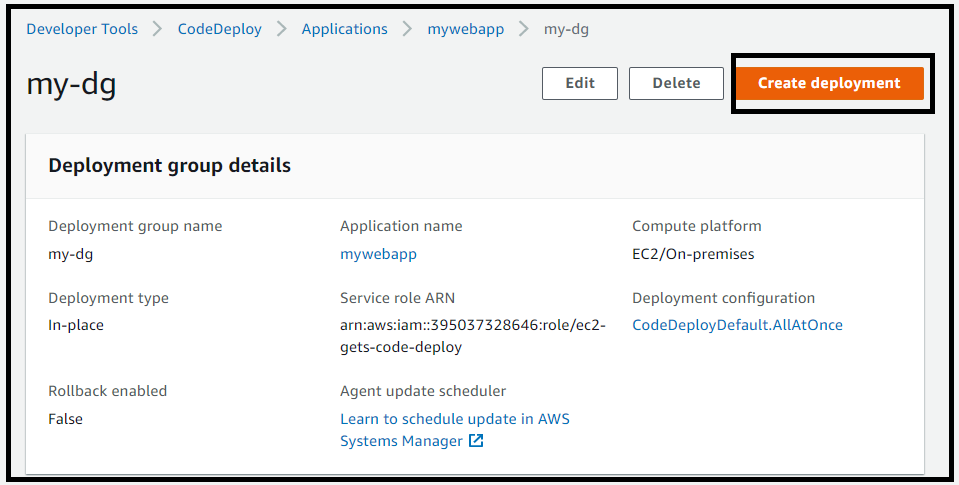
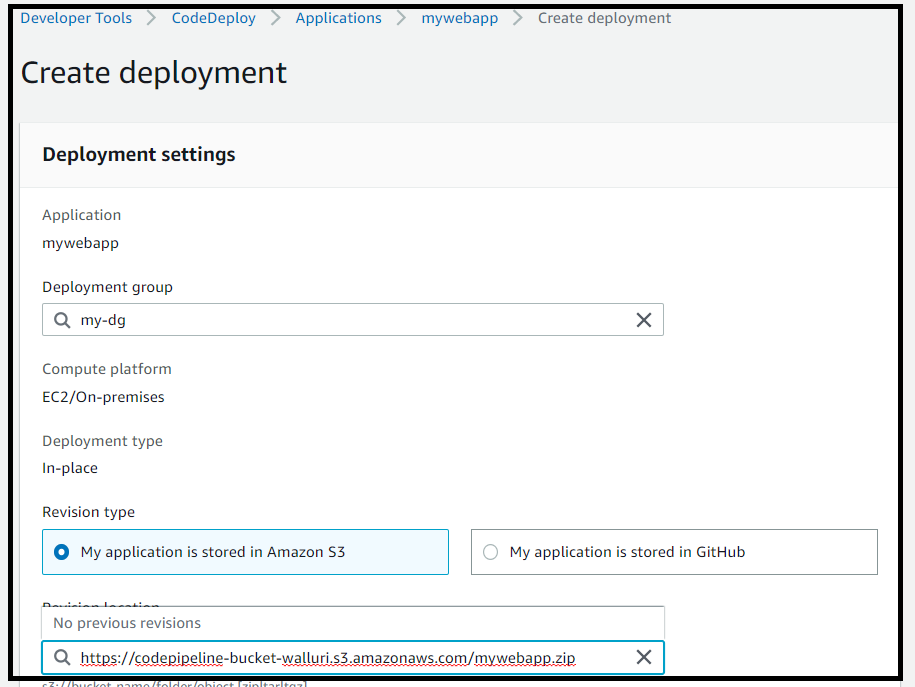
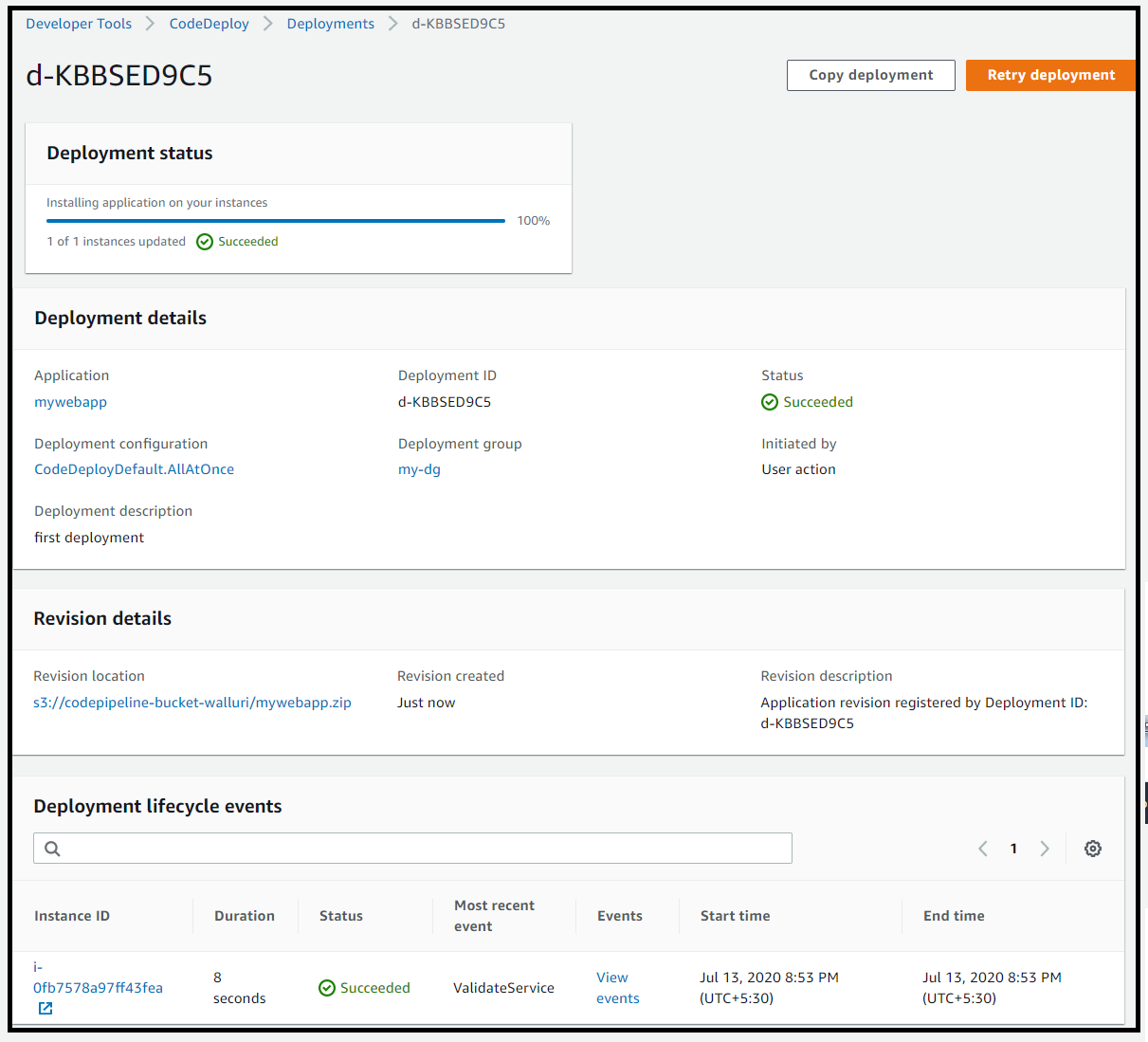
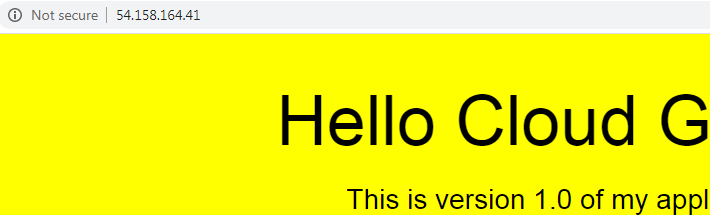
ParameterKey=SSHLocation,ParameterValue=0.0.0.0/0 \

ParameterKey=TagKey,ParameterValue=Name \

ParameterKey=TagValue,ParameterValue=CodeDeployDemo \

--capabilities CAPABILITY\_IAM  
  


Create another S3 bucket to upload source code and enable versioning on it . Because we want code-pipeline to find out when a new version of our source code appears in the bucket.  
  
  
Upload the first Version of the source code.   
   
  
Now confirm if the Cloud formation stack is completed creating our EC2 instance with Code-deploy agent.  
  
  
  
Now login to the instance and confirm if the code-deploy agent is installed successfully.  
  
  
  
  
  
Now we have to configure code-deploy to deploy our application using the code that we uploaded to the S3 bucket.  
  
Create a code deploy application.

  
  
  
  
  
Create a deployment group – note that the tag of EC2 instance is used to identify it for deploying.  
  
  
  
  
  
  
  
  
s3://codepipeline-bucket-walluri/mywebapp.zip [copy the underlying link not the url text..]  
  
After creating the deployment the dashboard looks as below.  
  
  
Access the EC2 instance as shown below.  
  
  
We shall Now use code-pipeline to manually deploy