

Deployment Automation using AWS Codedeploy, Jenkins and Code Commit

Continuous Integration: is a software development practice where continuous changes and updates in code base are integrated and verified by an automated build scripts using various tools.

Continuous Deployment: is also a software development practice whose role is to automatically deploy the code to the application folder of specified server

Scenario: Suppose we have set up an architecture of www.xyz.com application the server is setup on Amazon Web Services. As a part of the architecture , Our server is featured with AWS Auto Scaling service which is used to help scale our servers depending on the metrics and policies we specified. So every time a new feature will develop, we have to manually run the test cases before the code was integrated and deployed, later pull the latest code to all the environment servers.

Challenges :

1. Centralised repository for pulling and pushing code for deployment
2. Manually work to run test cases and pull the latest code on all the servers.
3. How to deploy code on new instance which are configured in AWS autoscaling
4. Since the servers were auto scaled we had to pull the latest code on one server, take image of that server, re configure it with Auto Scaling.
5. How to automatically deploy build on instances timely manner
6. How to Revert back to previous build

The above challenges requires lots of time and human resources. So we have to find a technique that must save high amount of time and make our life easy with automating all the process from CI to CD.

We are going to use **Jenkins** as CI tool and **AWS Code Deploy** as CD tool and **AWS Code Commit** as Application Repo.

Let's just walk through the flow, how it's going to work and what are the advantages before we implement it all.

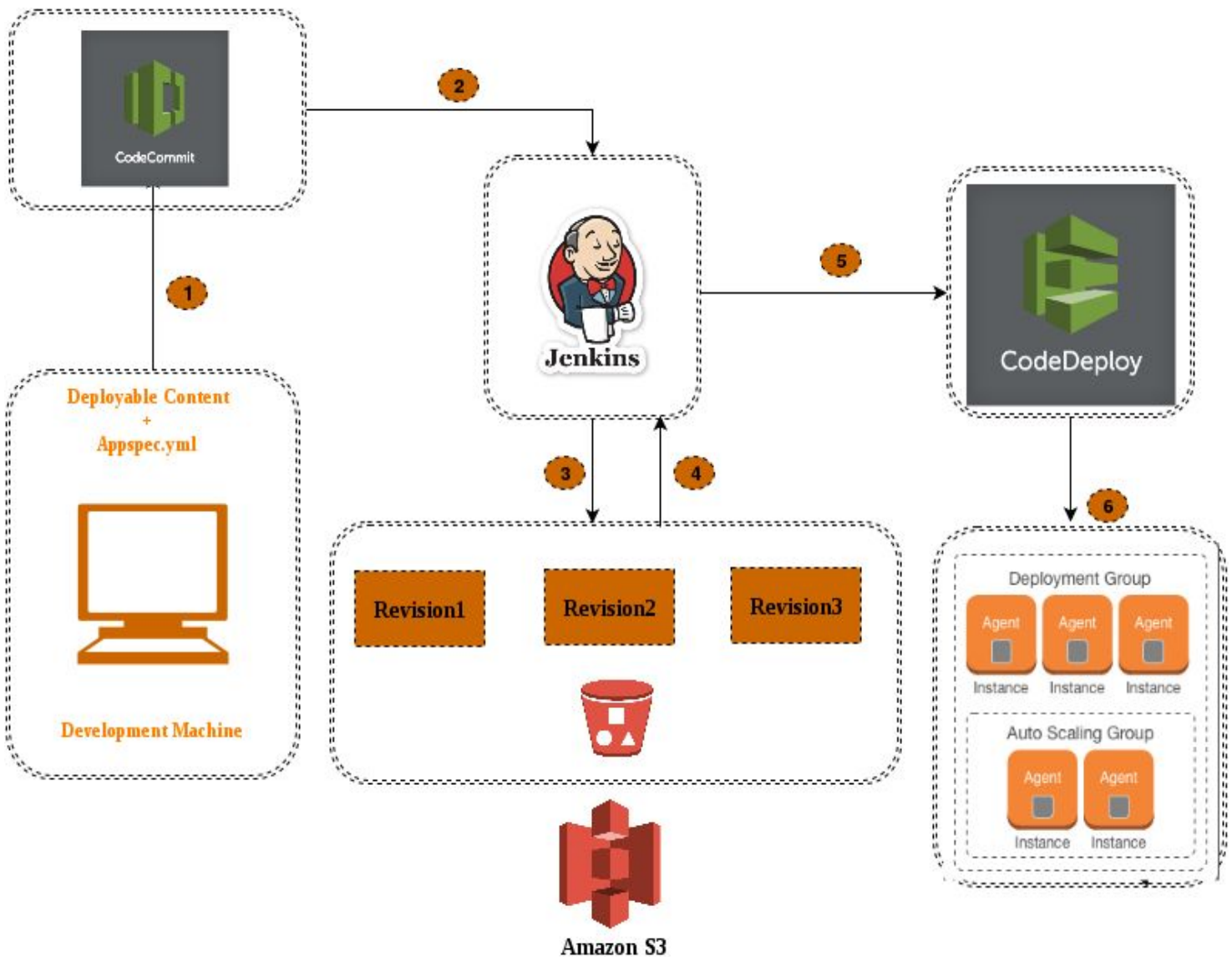
When a new code is pushed to a particular GIT repo/AWS Code Commit branch.

1. Jenkins will run the test cases, (Jenkins listening to a particular branch through git web hooks)
2. If the test cases fail. It will notify us and stop the further after build actions.
3. If the test cases are successful, it will go to post build action and trigger aws code deploy.
4. Jenkins will push the latest code in the zip file format to AWS S3 on the account we specify.
5. AWS Code Deploy will pull the zip file in all the Auto Scaled servers that have been mentioned.
6. For autoscaling server we can choose that ami which by default have the AWS Codedeploy Agent running on it cause these ami launch faster and pull the latest revision automatically.
7. Once the latest code is copied to the application folder , it will once again run the test cases.
8. If the test cases fail it will roll back the deployment to previous successful revision.
9. If it is successful , it will run post deployment build commands on server and ensure that latest deployment does not fail.
10. If we want to go back to previous revision then also we can roll back easily

So this way of automation makes the Deployment of application smooth, error tolerant, and faster.

Deployment of Application via CodeDeploy using Jenkins and CodeCommit (Implementation Steps)

Architecture :



Workflow:

Below are the workflow steps of the above architecture

1. The application code with the Appspec.yml file will be pushed to the AWS code Commit. The Appspec.yml file includes the necessary scripts path and command which will help the AWS code deploy to run the application successfully
2. As the Application and Appspec.yml file will get committed in the AWS codeCommit, jenkins will automatically will get triggered by poll SCM function.
3. Now Jenkins will pull the code from CodeCommit into its workspace (Path in Jenkins where all the artifacts is placed) and archive it and push it to the AWS S3 bucket. So this is considered as Job1. If the Job 1 will execute successfully then email will be sent to the Admin with the console output. And if the Job 1 get fail then email will be triggered with message of Job Failure and at that point only Jenkins stops working rather than executing the 2nd Job.
4. If Job 1 will execute successfully then it will trigger the Job2 which is responsible to pull the successful build version of code from S3 bucket and then trigger the Job3. If Job 2 will get fail, then again email will be triggered with message of Job Failure.
5. When Job 3 get triggered, the archive file (Application code along with appspec.yml) will be pushed to aws codedeploy.
6. Now AWS codedeploy is the deployment service which will run the Codedeploy agent in the instance and run the Appspec.yml file which will help the application to get up and running.
7. If at any place the Job will get fail then the application will be deployed will the previous build.

Step1. Setting Up CodeCommit in Development Environment.

Create an AWS CodeCommit Repository

- Open the AWS CodeCommit console at <https://console.aws.amazon.com/codecommit>.
- On the welcome page, choose Get Started Now. (If a Dashboard page appears instead of the welcome page, choose Create new repository.)



Dashboard

Share and manage your code in the cloud with AWS CodeCommit. Create, edit, and view details about your co

[Create new repository](#)

Filter by repository name 1 to 2 of 2 Repositories

Name	Description
JenkinsPipelineTestREPO	This repo is for the testing purpose for the jenkins pipeline
Intv-Test	Intv-Test

1 to 2 of 2 Repositories

- On the Create new repository page, in the Repository name box, type **shadi.com**
- In the Description box, type **Application repository of www.shadi.com**



Create new repository



Create a secure repository to store and share your code. Begin by typing a repository name and a description for your repository. Repository names are included in the URLs for that repository.



Access to the repository

Users connecting to an AWS CodeCommit repository for the first time must complete setup steps before they can use it. [Learn more](#)

Repository name*

shadi.com

Description

Application repository for www.shadi.com

Dev
Autl

*Required

Cancel

Create repository

- e. Choose Create repository to create an empty AWS CodeCommit repository named shadi.com

Create a Local Repo

In this step, we will set up a local repo on our local machine to connect to our repository. To do this, we will select a directory on our local machine that will represent the local repo. we will use Git to clone and initialize a copy of our empty AWS CodeCommit repository inside of that directory. Then we will specify the username and email address that will be used to annotate your commits.

- a. Generate ssh-keys in your local machine *#ssh-keygen* without any passphrase.

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/...):
Enter passphrase (empty for no passphrase):
Enter same passphrase again: <Type the passp...

Your identification has been saved in /home/...
Your public key has been saved in /home/...
The key fingerprint is:
45:63:d5:99:0e:99:73:50:5e:d4:b3:2d:86:4a:2c:...
The key's randomart image is:
+--[ RSA 2048 ]-----+
|      E . + . o * . + + |
|      . o . = . = o . |
|      . . . * . + |
|      . . o . + . . |
|      So . . . . |
|      . |
+-----+

```

- b. Cat **id_rsa.pub** and paste it into the IAM User->Security Credentials-> Upload SSH Keys Box. And Note Down the **SSH-KeyID**

```
$ cat ~/.ssh/id_rsa.pub
```

Copy this value. It will look similar to the following:

```
ssh-rsa EXAMPLE-AfICCQD6m7oRw0uX0jANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWV6b24xZDASBgNVBAsTC0lBTSDb25zbn2x1MRlWcEAYDVQQDEwLUZXR0dGx1YWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvaW5jb20wHhcNMTEwNDI1MjA0NTIwHhcNMTEwNDI1MjA0NTIwWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWV6b24xZDAS=EXAMPLE user-name@ip-192-0-2-137
```

SSH keys for AWS CodeCommit			
Use SSH public keys to authenticate to AWS CodeCommit repositories. Learn more about SSH keys.			
Upload SSH public key			
SSH Key ID	Uploaded	Status	Actions
APKAEIBAERIR2EXAMPLE	2015-07-21 16:32 PDT	Active	Make Inactive Show SSH Key Delete

- c. Click on Create Access keys and Download the Credentials having Access Key and Secret Key.
- d. Set the Environment Variables in BASHRC File at the end.

```
# vi /etc/bashrc
```

```
export AWS_ACCESS_KEY_ID=AKIAINTxxxxxxxxxxxxSAQ
export AWS_SECRET_ACCESS_KEY=9oqM2L2YbxxxxxxxxxxxxzSDFVA
```

- e. Set the config file inside .ssh folder

```
# vi ~/.ssh/config
```

```
Host git-codecommit.us-east-1.amazonaws.com
  User APKAXXXXXXXXXXT5RDFGV
  IdentityFile ~/.ssh/id_rsa          ---> Private Key
# chmod 400 config
```

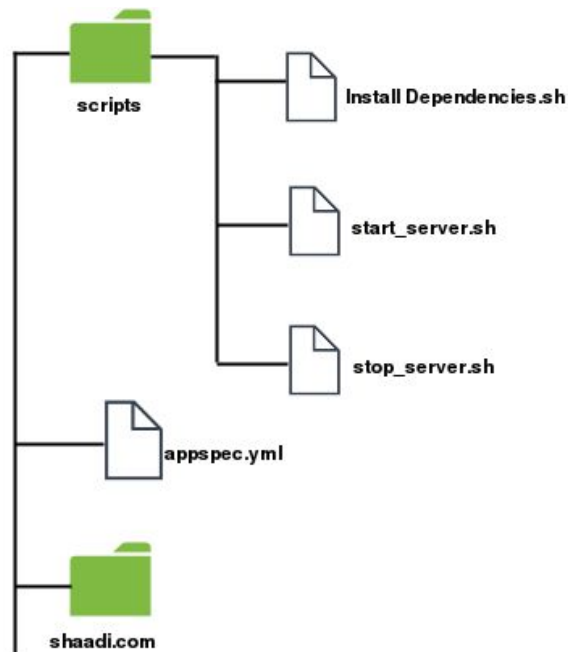
- f. Configure the Global Email and Username

```
#git config --global user.name "username"
#git config --global user.email "emailID"
```

- G. Copy the SSH URL to use when connecting to the repository and clone it

```
#git clone ssh://git-codecommit.us-east-1.amazonaws.com/shadi.com
```

Now Put the Application/Code inside the cloned directory and also write the appspec.yml file and you are ready to push it.



Install_dependencies.sh includes.

```
#!/bin/bash
yum groupinstall -y "PHP Support"
yum install -y php-mysql
yum install -y httpd
yum install -y php-fpm
```

Start_server.sh includes

```
#!/bin/bash
service httpd start
service php-fpm start
```

Stop_server.sh includes

```
#!/bin/bash
isExistApp=`pgrep httpd`
if [[ -n \${isExistApp} ]]; then
    service httpd stop
fi
isExistApp=`pgrep php-fpm`
if [[ -n \${isExistApp} ]]; then
    service php-fpm stop
Fi
```

Appspec.yml includes

```
version: 0.0
os: linux
files:
  - source: /
    destination: /var/www/shadi.com
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
```


H. Now push the code to the CodeCommit

```
# git add .  
# git commit -m "1st push"  
# git push
```

Now we can see that the code will be pushed to the codecommit .

Step2. Setting Up Jenkins Server in EC2-Instance.

Launch the EC2 instance (CentOS7/RHEL7) and perform the following operations

```
# yum update -y  
# yum install java-1.8.0-openjdk
```

Verify the java

```
# java -version
```

```
# wget -O /etc/yum.repos.d/jenkins.repo  
http://pkg.jenkins-ci.org/redhat/jenkins.repo  
# rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
```

Install Jenkins:

```
# yum install Jenkins
```

Add Jenkins to system boot:

```
# chkconfig jenkins on
```

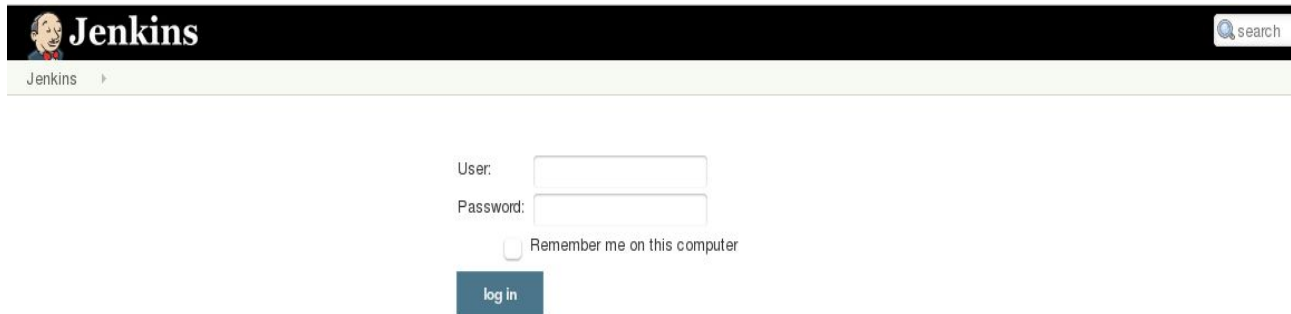
Start Jenkins:

```
# service jenkins start
```

By default Jenkins will start on Port 8080, this can be verified via

```
# netstat -tnlp | grep 8080
```

Go to browser and navigate to **http://<Elastic/Public-IP>:8080**. You will see jenkins dashboard.

The image shows the Jenkins login dashboard. At the top, there is a black header bar with the Jenkins logo (a cartoon character) and the word "Jenkins" in white. To the right of the header is a search bar with a magnifying glass icon and the word "search". Below the header, there is a light green navigation bar with the word "Jenkins" and a right-pointing arrow. In the center of the page, there is a login form. It consists of two input fields: "User:" and "Password:". Below the "Password:" field is a checkbox labeled "Remember me on this computer". At the bottom of the form is a blue button with the text "log in" in white.

Configure The Jenkins Username and password and Install the Aws and Git related Plugins.

Setup a Jenkins Pipeline Job.

In Source Control Management click on GIT.

Pass the git ssh URL and In credentials, Click on ADD and then in kind option click SSH username with PrivateKey. Username will be same as mentioned in the config file of development machine where repo was initiated and we have to cat the private key of development machine and paste it here.

Source Code Management

☐ None
☐ AWS CodePipeline
☐ Gerrit Repo
☒ Git

Repositories

Repository URL:

Credentials:

Branches to build

Branch Specifier (blank for 'any'):


Repository browser

Additional Behaviours

In Build Trigger click on Poll SCM and mention the time whenever you want to start the build.

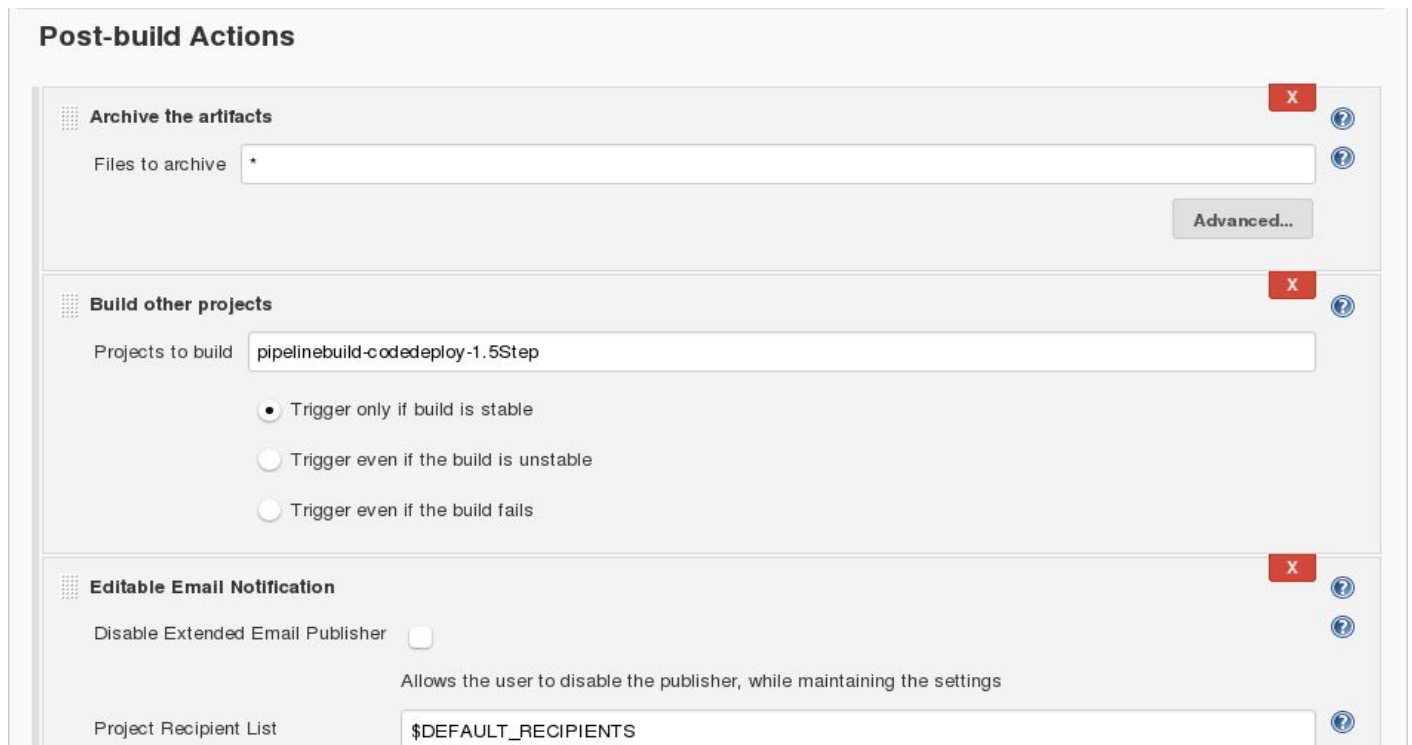
☒ Poll SCM

Schedule

 Do you really mean "every minute" when you say "*****"? Perhaps you meant "H*****" to poll once per hour

Would last have run at Friday, September 30, 2016 12:38:45 PM UTC; would next run at Friday, September 30, 2016 12:38:45 PM UTC.

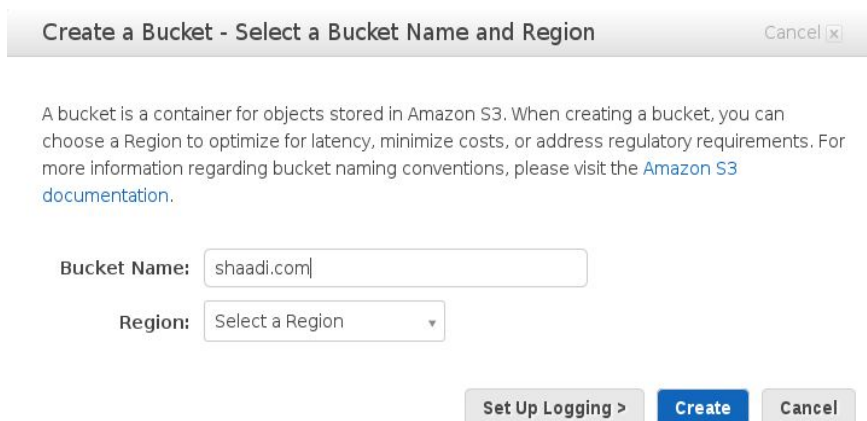
For the Post Build Action we have to archive the files and provide the name of job 2, if the job 1 will get successful build after then it should trigger the email.



The screenshot shows the 'Post-build Actions' configuration page in Jenkins. It contains three sections: 'Archive the artifacts', 'Build other projects', and 'Editable Email Notification'. The 'Archive the artifacts' section has a 'Files to archive' field with an asterisk and an 'Advanced...' button. The 'Build other projects' section has a 'Projects to build' field with the value 'pipelinebuild-codedeploy-1.5Step' and three radio buttons for triggering conditions: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. The 'Editable Email Notification' section has a 'Disable Extended Email Publisher' checkbox, a description 'Allows the user to disable the publisher, while maintaining the settings', and a 'Project Recipient List' field with the value '\$DEFAULT_RECIPIENTS'.

Now For the time being we can start Building the Job and we have to verify that when the code is committed, the Jenkins should start building automatically and weather it is able to pull the code into its workspace folder but before that we have to create S3 bucket and pass credentials (Access key and Secret key) in jenkins so that when the jenkins pull code from codecommit and after archiving it can push build in the s3 bucket

Step3. Create S3 Bucket.



The screenshot shows the 'Create a Bucket' dialog in AWS S3. It has a title bar 'Create a Bucket - Select a Bucket Name and Region' and a 'Cancel' button. Below the title bar is a paragraph of text explaining what a bucket is and how to choose a region. Below the text are two input fields: 'Bucket Name' with the value 'shaadi.com' and 'Region' with a dropdown menu showing 'Select a Region'. At the bottom are three buttons: 'Set Up Logging >', 'Create', and 'Cancel'.

After creating S3 bucket, provide the details into jenkins with the AWS credentials.

Amazon S3 profiles

S3 profiles

Profile name

Use IAM Role ☐

Access key

Check passed!

Secret key

Advanced...

Delete

Add

Profiles for publishing to S3 buckets

Now when we run job 1 of Jenkins it will pull the code from code commit and after archiving, it will keep into the workspace folder of job1.

Console Output

```
Started by user Nikit Swaraj
Building in workspace /var/lib/jenkins/workspace/pipelinebuild-codedeploy-1stStep
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/JenkinsPipe
Fetching upstream changes from ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/JenkinsPipe
> git --version # timeout=10
using GIT_SSH to set credentials
> git fetch --tags --progress ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/JenkinsPipe:
/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 3c7e516141f8a95e5bd1cd3e39f0bd7aa994dfde (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 3c7e516141f8a95e5bd1cd3e39f0bd7aa994dfde
> git rev-list 3c7e516141f8a95e5bd1cd3e39f0bd7aa994dfde # timeout=10
Checking for pre-build
Executing pre-build step
Checking if email needs to be generated
No emails were triggered.
Archiving artifacts
Checking for post-build
Performing post-build step
Checking if email needs to be generated
Email was triggered for: Always
Sending email for trigger: Always
messageContentType = text/plain; charset=UTF-8
Request made to attach build log
Adding recipients from project recipient list
Adding recipients from trigger recipient list
Successfully created MimeMessage
Sending email to: nikit.swaraj@minjar.com

downstream builds to be triggered
Triggering a new build of pipelinebuild-codedeploy-1.5Step
Finished: SUCCESS
```

From the above Console output we can see that it is pulling the code from codecommit and after archiving, it is triggering the email and after then it calls for the next job 2

Job 2 is responsible for pushing the archive code to S3 bucket and then after successful push it will trigger the email with console output and call job3.

Console Output

```
Started by upstream project "pipelinebuild-codedeploy-1stStep" build number 44
originally caused by:
  Started by user Nikit Swaraj
Building in workspace /var/lib/jenkins/workspace/pipelinebuild-codedeploy-1.5Step
Copied 10 artifacts from "pipelinebuild-codedeploy-1stStep" build number 44
Publish artifacts to S3 Bucket Build is still running
Publish artifacts to S3 Bucket Using S3 profile: jenkinspipelinecodepackage
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=Always-37.txt region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=Always-38.txt region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=Always-39.txt region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=Always-40.txt region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=Always-41.txt region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=Always-42.txt region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=Always-43.txt region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=appspec.yml region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=hello.sh region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Publish artifacts to S3 Bucket bucket=jenkinspipelinecodepackage/32, file=index.html region=sa-east-1, will be uploaded from
slave=false managed=false , server encryption false
Warning: you have no plugins providing access control for builds, so falling back to legacy behavior of permitting any
downstream builds to be triggered
Triggering a new build of pipelinebuild-codedeploy-2ndStep
Finished: SUCCESS
```

The above image shows that after building Job2, the Job3 will also get triggered. Now before triggering Job3, we need to setup AWS codedeploy environment.

Step4. Now launch the AWS codedeploy application.

Creating IAM Roles

Create an iam instance profile and attach AmazonEC2FullAccess policy and also attach the following inline policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Dashboard

Search IAM

Details

Groups

Users

Roles

Policies

Identity Providers

Account Settings

Credential Report

Encryption Keys

Creation Time: 2016-03-01 16:04 UTC+0530

Permissions | Trust Relationships | Access Advisor

Managed Policies

The following managed policies are attached to this role. You can attach up to 10 managed policies.

[Attach Policy](#)

Policy Name	Actions
AmazonEC2FullAccess	Show Policy Detach Policy Simulate Policy

Inline Policies

This view shows all inline policies that are embedded in this role.

[Create Role Policy](#)

Policy Name	Actions
CodeDeploy-EC2Inline	Show Policy Edit Policy Remove Policy Simulate Policy

Create a service role `CodeDeployServiceRole`. Select Role type AWS CodeDeploy. Attach the Policy `AWSCodeDeployRole` as shown in the below screenshots:

The first screenshot shows the 'Select Role Type' step of the 'Create Role' wizard. The 'AWS Service Roles' section is expanded, showing several roles. The 'AWS CodeDeploy' role is selected, which allows CodeDeploy to access other AWS services such as Auto Scaling on your behalf. The 'Role for Cross-Account Access' and 'Role for Identity Provider Access' sections are also visible.

The second screenshot shows the 'Attach Policy' step of the 'Create Role' wizard. It displays a table of policies to attach. The 'AWSCodeDeployRole' policy is selected, and the table shows its details.

Policy Name	Attached Entities	Creation Time	Edited Time
AWSCodeDeployRole	0	2015-05-04 23:35 UTC+0530	2016-02-17 22:05 UTC+0530

Create an autoscaling group for a scalable environment. Steps below:

Choose an ami and select an instance type for it and Attach the iam instance profile which we created in the earlier step

1. Choose AMI 2. Choose Instance Type 3. Configure details 4. Add Storage 5. Configure Security Group 6. Review

Create Launch Configuration

Name

Purchasing option ☐ Request Spot Instances

IAM role

Monitoring

► Advanced Details

Later, if you want to use a different launch configuration, you can create a new one and apply it to any Auto Scaling group. Existing launch configurations cannot be edited.

Cancel Previous Skip to review Next: Add Storage

Now go to Advanced Settings and type the following commands in “User Data” field to install codedeploy agent on your machine (if it’s not already installed on your ami)

```
#!/bin/bash
yum -y update
yum install -y ruby
yum install -y aws-cli
sudo su -
aws s3 cp s3://aws-codedeploy-us-east-1/latest/install . --region
us-east-1
chmod +x ./install
./install auto
```

Select Security Group in the next step and create the launch configuration for the autoscaling group. Now using the launch configuration created in the above step, create an Autoscaling group.

Now after creating Autoscaling group its time to create Deployment Group

Click on AWS codedeploy and click on create application and Mention the application name and Deployment Group Name

The screenshot shows the 'Create New Application' page in the AWS CodeDeploy console. At the top, there's a header with the AWS CodeDeploy logo and a dropdown arrow. Below the header, the title 'Create New Application' is followed by a help icon. A subtitle reads: 'Create a new application; specify the instances to deploy it to; and specify the conditions for a successful deployment.' There are two input fields: 'Application Name*' with a '100 character limit' hint, and 'Deployment Group Name*' also with a '100 character limit' hint. Below these is a section titled 'Add Instances' with a subtitle: 'Locate and add existing instances to this deployment group by searching for their tags or Auto Scaling group names.' A light blue information box contains a list of requirements for instances, each with a 'Learn More' link.

Create New Application ?

Create a new application; specify the instances to deploy it to; and specify the conditions for a successful deployment.

Application Name* 100 character limit

Deployment Group Name* 100 character limit

Add Instances

Locate and add existing instances to this deployment group by searching for their tags or Auto Scaling group names.

Info AWS CodeDeploy requires the following for each instance that it deploys to.

1. Each Amazon EC2 instance must launch with the correct IAM instance profile attached. [Learn More](#)
2. Each Amazon EC2 instance must have identifying Amazon EC2 tags ([Learn More](#)) or be in an Auto Scaling group. [Learn More](#)
3. Each on-premises instance must have an associated IAM user, identifying on-premises instance tags, and a special configuration file. [Learn More](#)
4. The AWS CodeDeploy agent must be installed and running on each instance. [Learn More](#)

In tag type click on either EC2 instance or AWS AutoScale Group. And mention the Name of ec2 instance or AWS autoscale Group.

The screenshot shows the 'Search by Tags' section of the AWS CodeDeploy console. It features a table with columns: 'Tag Type', 'Key', 'Value', and 'Instances'. The first row shows '1' in the 'Instances' column. A dropdown menu is open under the 'Tag Type' column, showing options: 'Auto Scaling Group', 'Amazon EC2', 'Auto Scaling Group', and 'On-Premises Instance'. Below the table, there's a 'Total Matches' label and a horizontal line. Further down is the 'Deployment Configuration' section with a subtitle: 'Choose from a list of default and custom deployment configurations. A deployment configuration is a set of rules that determines how fast an application will be deployed and the success or failure conditions for a deployment.' It includes a 'Deployment Config*' dropdown menu set to 'CodeDeployDefault.OneAtATime' with a help icon. At the very bottom, a small note says: 'Deploys to one instance at a time. Succeeds if all instances'.

Search by Tags ?

	Tag Type	Key	Value	Instances
1	Auto Scaling Group			

Total Matches

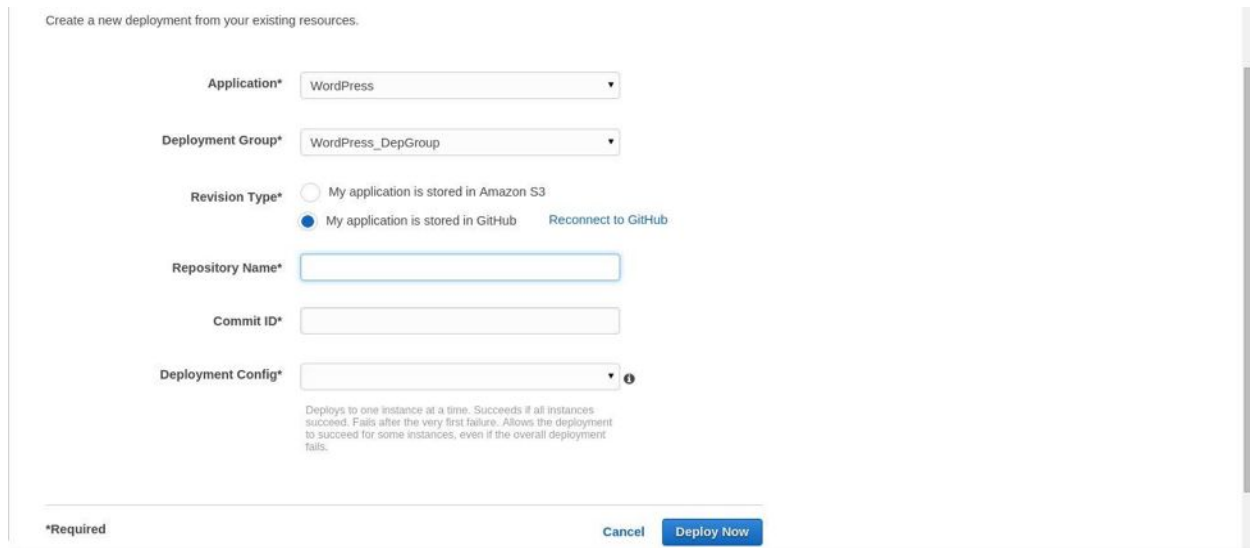
Deployment Configuration

Choose from a list of default and custom deployment configurations. A deployment configuration is a set of rules that determines how fast an application will be deployed and the success or failure conditions for a deployment.

Deployment Config* CodeDeployDefault.OneAtATime ?

Deploys to one instance at a time. Succeeds if all instances

Select ServiceRoleARN for the service role which we created in the “Creating IAM Roles” section of this post. Go to Deployments and choose Create New Deployment. Select Application and Deployment Group and select the revision type for your source code



The screenshot shows the 'Create a new deployment from your existing resources' form in the AWS CodeDeploy console. The form includes the following fields and options:

- Application***: A dropdown menu with 'WordPress' selected.
- Deployment Group***: A dropdown menu with 'WordPress_DepGroup' selected.
- Revision Type***: Two radio buttons. 'My application is stored in Amazon S3' is unselected, and 'My application is stored in GitHub' is selected. A 'Reconnect to GitHub' link is visible next to the selected option.
- Repository Name***: An empty text input field.
- Commit ID***: An empty text input field.
- Deployment Config***: A dropdown menu with an information icon (i) to its right.

Below the 'Deployment Config*' field, there is a small text description: 'Deploys to one instance at a time. Succeeds if all instances succeed. Fails after the very first failure. Allows the deployment to succeed for some instances, even if the overall deployment fails.'

At the bottom of the form, there is a legend for '*Required', a 'Cancel' button, and a blue 'Deploy Now' button.

Note: The IAM role associated with the instance or autoscale group should be same as codedeploy and the arn name must have the codedeploy policy associated with it.

Step5. Fill Codedeploy Info in jenkins and Build it

Now go back to jenkins job 3 and click on “Add PostBuild Action” and select “Deploy the application using AWS codedeploy. Fill the details of

AWS CodeDeploy Application Name , AWS CodeDeploy Deployment Group, AWS CodeDeploy Deployment Config, AWS Region S3 Bucket, Include Files ** and click on Access/secret to fill the Keys for the Authentication. Click on save and Build the project. After few min or sec the application will be deployed on the Autoscale instances.

Deploy an application to AWS CodeDeploy

X

AWS CodeDeploy Application Name

ShadiApp

AWS CodeDeploy Deployment Group

ShadiGN

AWS CodeDeploy Deployment Config

CodeDeployDefault.OneAtATime

AWS Region

SA_EAST_1

S3 Bucket

shadi.com

S3 Prefix

Subdirectory

Include Files

**

Exclude Files

Proxy Host

Proxy Port

0

Version File

Appspec.yml per Deployment Group

☐

☐ Register Revision

☐ Deploy Revision

☒ Use Access/Secret keys

If these keys are left blank, the plugin will attempt to use credentials from the default provider chain.
That is: Environment Variables, Java System properties, credentials profile file, and finally, EC2
Instance profile.

AKIAINTX2I4PI5G67OHQ

Save

Apply

When this Job3 will get build successfully then we will get the console output as below

Console Output

Started by upstream project "[pipelinebuild-codedeploy-1.5Step](#)" build number [32](#)

originally caused by:

Started by upstream project "[pipelinebuild-codedeploy-1stStep](#)" build number [44](#)

originally caused by:

Started by user [Nikit Swaraj](#)

Building in workspace /var/lib/jenkins/workspace/pipelinebuild-codedeploy-2ndStep

[pipelinebuild-codedeploy-2ndStep] \$ /bin/sh -xe /tmp/hudson6241934088690401524.sh

Ziping files into /tmp/pipelinebuild-codedeploy-2ndStep-3044597032762813881.zip

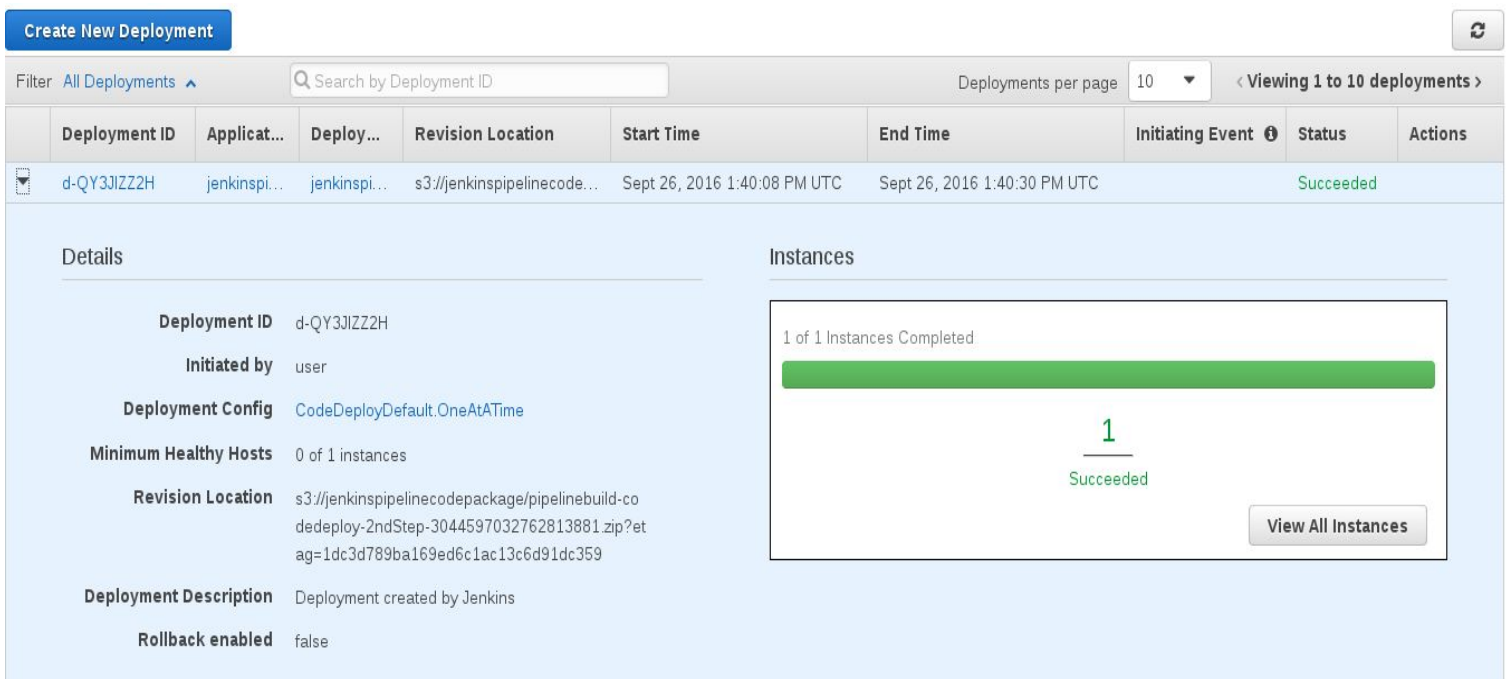
Uploading zip to s3://jenkinspipelinecodepackage/pipelinebuild-codedeploy-2ndStep-3044597032762813881.zip

Registering revision for application 'jenkinspipelinetestApp'

Creating deployment with revision at {RevisionType: S3,S3Location: {Bucket: jenkinspipelinecodepackage,Key: pipelinebuild-codedeploy-2ndStep-3044597032762813881.zip,BundleType: zip,ETag: 1dc3d789ba169ed6c1ac13c6d91dc359},}

Finished: SUCCESS

After this Build, there will be changes takes place in AWS codedeploy group



The screenshot displays the AWS CodeDeploy console interface. At the top, there is a blue button labeled "Create New Deployment" and a refresh icon. Below this is a filter section with "All Deployments" selected and a search bar. The main table lists deployments, with the first one highlighted: "d-QY3JIZZ2H" for application "jenkinspi...", initiated by "jenkinspi...", at location "s3://jenkinspipelinecode...", starting on "Sept 26, 2016 1:40:08 PM UTC" and ending on "Sept 26, 2016 1:40:30 PM UTC", with a status of "Succeeded".

Below the table, the "Details" section for the selected deployment is shown. It includes the following information:

- Deployment ID:** d-QY3JIZZ2H
- Initiated by:** user
- Deployment Config:** CodeDeployDefault.OneAtATime
- Minimum Healthy Hosts:** 0 of 1 instances
- Revision Location:** s3://jenkinspipelinecodepackage/pipelinebuild-codedeploy-2ndStep-3044597032762813881.zip?etag=1dc3d789ba169ed6c1ac13c6d91dc359
- Deployment Description:** Deployment created by Jenkins
- Rollback enabled:** false

The "Instances" section shows a progress bar indicating "1 of 1 Instances Completed". A large green number "1" is displayed, with the word "Succeeded" below it. A button labeled "View All Instances" is located at the bottom right of the instances section.

After then when you will hit the dns of the instance you will get your Application up and running.

Build Pipeline

Jenkins pipeline (previously workflow) refers to the job flow in a specific manner. Building Pipeline means breaking the big Job into small individual jobs, relying on which, if first job get failed then it will trigger the email to the admin and stop the building process at that step only and will not move to the second job.

To achieve the pipeline, one should need to install the pipeline plugin in Jenkins.

According to the above scenario, the Jobs will be broken into three individual Jobs

Job 1: When the code code commit, the Job 1 will run and it will pull the latest code from the CodeCommit repository, and it will archive the artifact and email about the status of Job1, weather it got successful build or got failed altogether with the console output. If the Job1 got build successfully then it will trigger to Job 2.

Job2: This Job will get run only when the Job 1 will be stable and run successfully. In Job2, the artifacts from Job1 will be copied to workspace 2 and will be pushed to AWS S3 bucket. Post to that if the artifacts will be send to S3 bucket, the email will be send to the admin. And then it will trigger the Job3.

Job3: This Job is responsible to invoke the AWS codedeploy and pull the code from S3 and push it either running ec2 instance or AWS auto scaling instances. When it will be done

The below image shows the structure of pipeline.

pipelinebuild-coddeploy-1stStep [Jenkins] - Mozilla Firefox

52.67.104.169:8080/view/pipelinebuild-coddeploy-1stStep/

Jenkins

pipelinebuild-coddeploy-1stStep

Build Pipeline

Run History Configure Add Step Delete Manage

Pipeline #44

- #44 pipelinebuild-coddeploy-1stStep
Sep 26, 2016 1:39:48 PM
2.5 sec
nikit
- #32 pipelinebuild-coddeploy-1.5Step
Sep 26, 2016 1:39:57 PM
0.22 sec
- #38 pipelinebuild-coddeploy-2ndStep
Sep 26, 2016 1:40:07 PM
1.1 sec

Pipeline #43

- #43 pipelinebuild-coddeploy-1stStep
Sep 26, 2016 1:12:14 PM
2.5 sec
nikit
- #31 pipelinebuild-coddeploy-1.5Step
Sep 26, 2016 1:12:27 PM
0.22 sec
- #37 pipelinebuild-coddeploy-2ndStep
Sep 26, 2016 1:12:37 PM
1.3 sec

Page generated: Sep 26, 2016 2:21:54 PM UTC REST API Jenkins ver. 2.7.4

pipelinebuild-coddeploy-1stStep root@ip-10-0-0-138:/var/lib/jenkins 07:51 PM