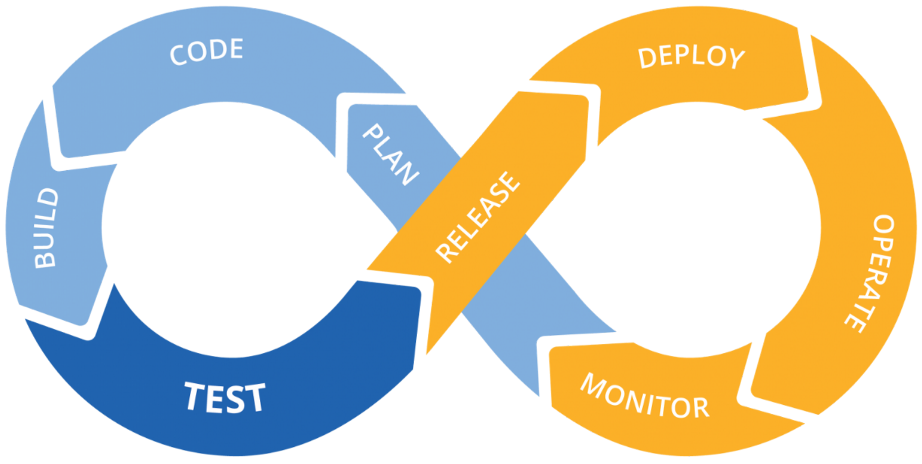
Continuous Integration - Continuous Delivery

What is CI/CD?

An assembly line in a factory produces consumer goods from raw materials in a fast, automated, reproducible manner. Similarly, a software delivery pipeline produces releases from source code in a fast, automated, and reproducible manner. The overall design for how this is done is called "continuous delivery." The process that kicks off the assembly line is referred to as "continuous integration." The process that ensures quality is called "continuous testing" and the process that makes the end product available to users is called "continuous deployment." This is a high level overview of what CI/CD is.

From a high level, a CI/CD pipeline usually consists of the following discrete steps:

1. Commit. When a developer finishes a change to an application, he or she commits it to a central source code repository.
2. Build. The change is checked out from the repository and the software is built so that it can be run by a computer. This steps depends a lot on what language is used and for interpreted languages this step can even be absent.
3. Automated tests. This is where the meat of the CI/CD pipeline is. The change is tested from multiple angles to ensure it works and that it doesn't break anything else.
4. Deploy. The built version is deployed to production.



What does "continuous" mean?
Continuous is used to describe many different processes that follow the practices I describe here. It doesn't mean "always running." It does mean "always ready to run." In the context of creating software, it also includes several core concepts/best practices. These are:

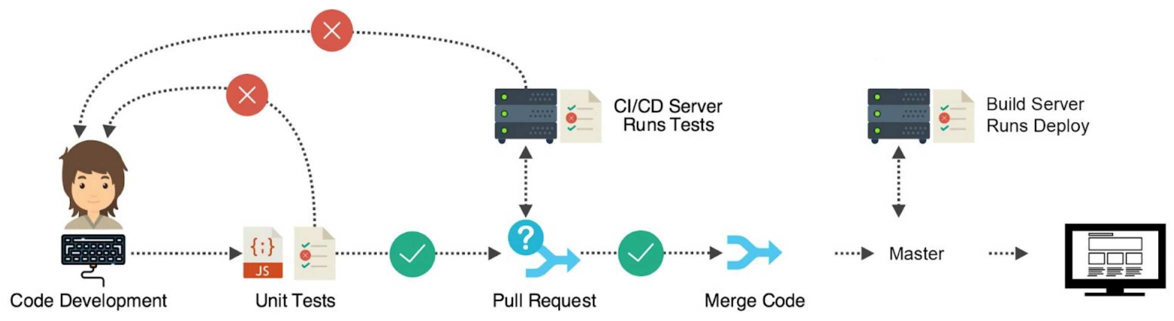
1. Frequent releases: The goal behind continuous practices is to enable delivery of quality software at frequent intervals. Frequency here is variable and can be defined by the team or company. For some products, once a quarter, month, week, or day may be frequent enough.

For others, multiple times a day may be desired and doable. Continuous can also take on an "occasional, as-needed" aspect. The end goal is the same: Deliver software updates of high quality to end users in a repeatable, reliable process. Often this may be done with little to no interaction or even knowledge of the users (think device updates).

2. Automated processes: A key part of enabling this frequency is having automated processes to handle nearly all aspects of software production. This includes building, testing, analysis, versioning, and, in some cases, deployment.

3. Repeatable: If we are using automated processes that always have the same behavior given the same inputs, then processing should be repeatable. That is, if we go back and enter the same version of code as an input, we should get the same set of deliverables. This also assumes we have the same versions of external dependencies (i.e., other deliverables we don't create that our code uses). Ideally, this also means that the processes in our pipelines can be versioned and re-created (see the DevOps discussion later on).

4. Fast processing: "Fast" is a relative term here, but regardless of the frequency of software updates/releases, continuous processes are expected to process changes from source code to deliverables in an efficient manner. Automation takes care of much of this, but automated processes may still be slow. For example, integrated testing across all aspects of a product that takes most of the day may be too slow for product updates that have a new candidate release multiple times per day.



Continuous integration
Developers practicing continuous integration merge their changes back to the main branch as often as possible. The developer's changes are validated by creating a build and running automated tests against the build. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes into the release branch.

Continuous integration puts a great emphasis on testing automation to check that the application is not broken whenever new commits are integrated into the main branch.

Continuous Testing

Continuous testing refers to the practice of running automated tests of broadening scope as code goes through the CD pipeline.
The goal of continuous testing in a delivery pipeline is always the same: to prove by successive levels of testing that the code is of a quality that it can be used in the release that's in progress. Building on the continuous principle of being fast, a secondary goal is to find problems quickly and alert the development team.

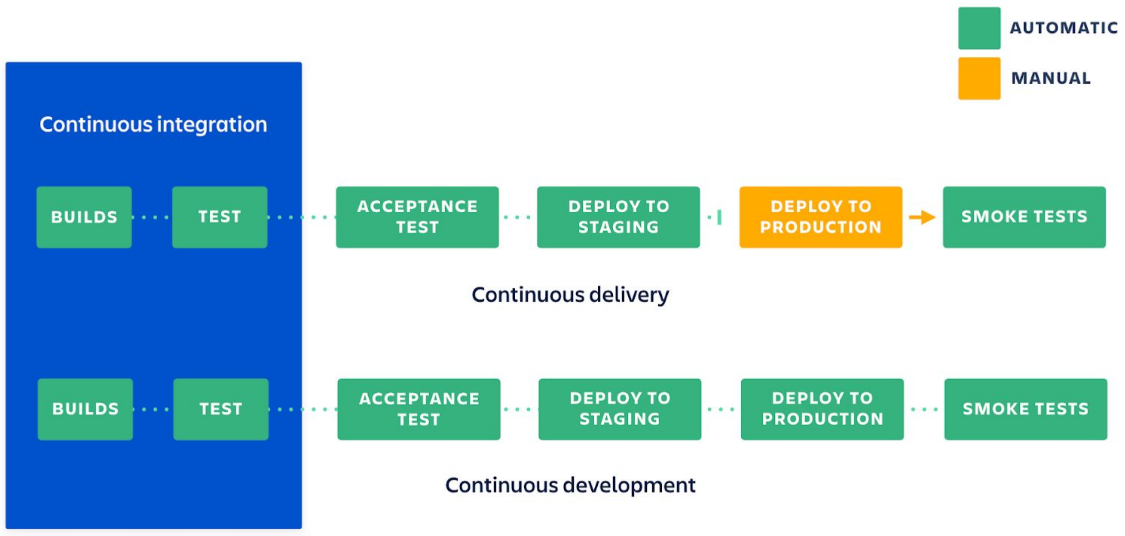
Continuous delivery
Continuous delivery is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button.

In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem.

Continuous deployment
Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Continuous deployment is an excellent way to accelerate the feedback loop with your customers and take pressure off the team as there isn't a Release Day anymore. Developers can focus on building software, and they see their work go live minutes after they've finished working on it.

And continuous deployment is like continuous delivery, except that releases happen automatically.



Advantages of CI/CD:

Faster Software Builds
As we know “time is money,” therefore, integrating CI and CD will result in faster builds and deliver quicker results. With deployments running in continuous cycles, this will enable you to track the project and provide feedback in real-time, as well as fix shortcomings with your team whenever necessary. With consistent reviews, the product will see more refinement, and be more in tandem with the end-users’ expectations.

Time-to-Market

By deploying your app to the market in time, this would not only engage your customers better, but would also assure profits, support pricing, and advance market goals as well. Therefore, focusing on time-to-market would not only make you well equipped to adapt to the changes in the market but would scale up your mobile app too, which would positively affect your ROI (Return of investment)!

Improvements to Code Quality
This will help build a high-quality app. CI allows developers to integrate their code into a common repository. With the help of this repository, developers can share their builds multiple times a day rather than working in isolation. In turn, this will help reduce integration costs, as developers can share their builds in a more frequent manner. For example, a developer may encounter conflicts between new and existing code while integrating. If these conflicts are addressed in real-time, then it will be easier to carry out the decided upon resolution.

There are tools to achieve this process of Continuous Integration and Continuous Deployment, and some of them are :

- Jenkins
- Circle CI
- Travis CI
- TeamCity
- CodeShip
- AWS CodePipeline

CircleCI 2.0 :

CircleCI integrates with a VCS and automatically runs a series of steps every time that it detects a change to your repository.
A CircleCI build consists of a series of steps. Generally, they're:
1. Dependencies
2. Testing
3. Deployment
The code is built in a container or a VM as per your wish.

Problem Statement:

Create a sample nodejs application, write certain API tests for it using Postman/Newman, and create a pipeline for build,test and deploy using CircleCI and AWS.

Step 1: Making the sample application : TodoList

Made a sample NodeJS application ~ Todo List using Rest APIs.

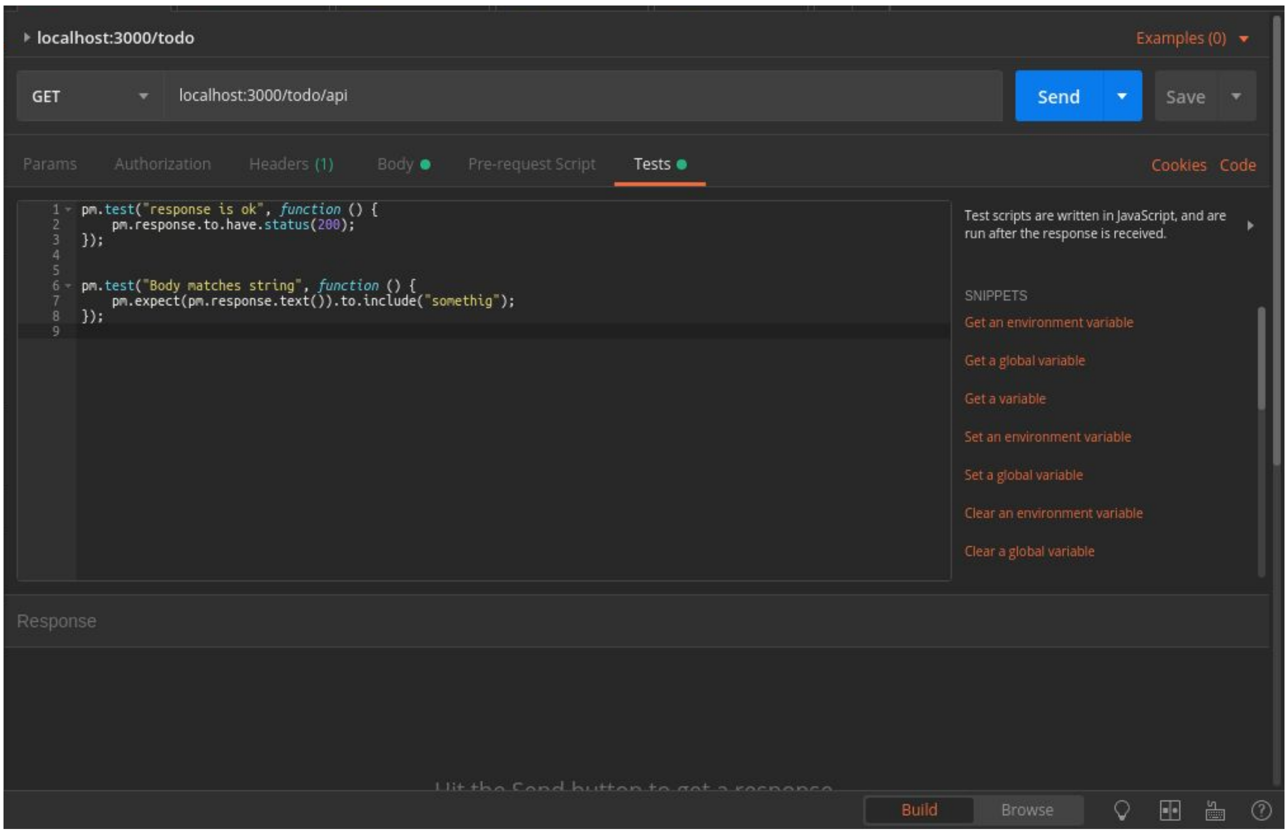
Followed the below link, for making the sample application:
https://www.youtube.com/watch?v=w-7RQ46RgxU&list=PL4cUxeGkcC9gcy9lrvMJ75z9maRw4byYp

It's a simple ToDo List, where the user can add an entity and on clicking on "Submit", the entity gets added to the list and the entire list is displayed in the same page.
On clicking any item in the list, it gets deleted.

Step 2: API Testing using Postman/Newman:

Follow the below documentation link for any doubts regarding postman and newman:
https://learning.getpostman.com/docs/postman/launching_postman/installation_and_updates/

Once you are done with step 1, it's time to write tests to ensure that your application doesn't have any bugs.
As for testing sake, I've written two tests in one API request in a collection and exported it as a json file, so that it can be run using newman cli.
Install "forever" command in your local machine.
With this command, you can connect to your localhost server in the background using the command :
forever start app.js
And then run your testcases using the command:
Newman run <collection-file-name.json>



```
pm.test("response is ok", function () {
    pm.response.to.have.status(200);
});


pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("somethig");
});
```

Test scripts are written in JavaScript, and are run after the response is received.

Note: Don't use the Google chrome extension for Postman, instead use the native version which can be downloaded here : https://www.getpostman.com/apps

We are done with the sample application part and it's testing. Now let us see about CircleCI.

Step 3: Setup CircleCI:

Once the sample application is done, let us see how we integrate it with CircleCI:

Create a CircleCI account:

1. Create an account on CircleCI by going to https://circleci.com/ , and by signing up with GitHub.
2. Once you are signed up, login into your CircleCI account.

You can find the list of your GitHub repositories, and can add any repo to CircleCI building process, provided it should contain a “config.yml” file.

Follow this link to get started with circleCI :
https://circleci.com/docs/2.0/getting-started/
You can also refer to the documentation page of CircleCI.
Now, to integrate any project with CircleCI, it should contain a “config.yml” file inside a “.circleci” folder.

Done. CircleCI is now set up. I will explain the config.yml later. I have also added comments in the file to help understand its working.

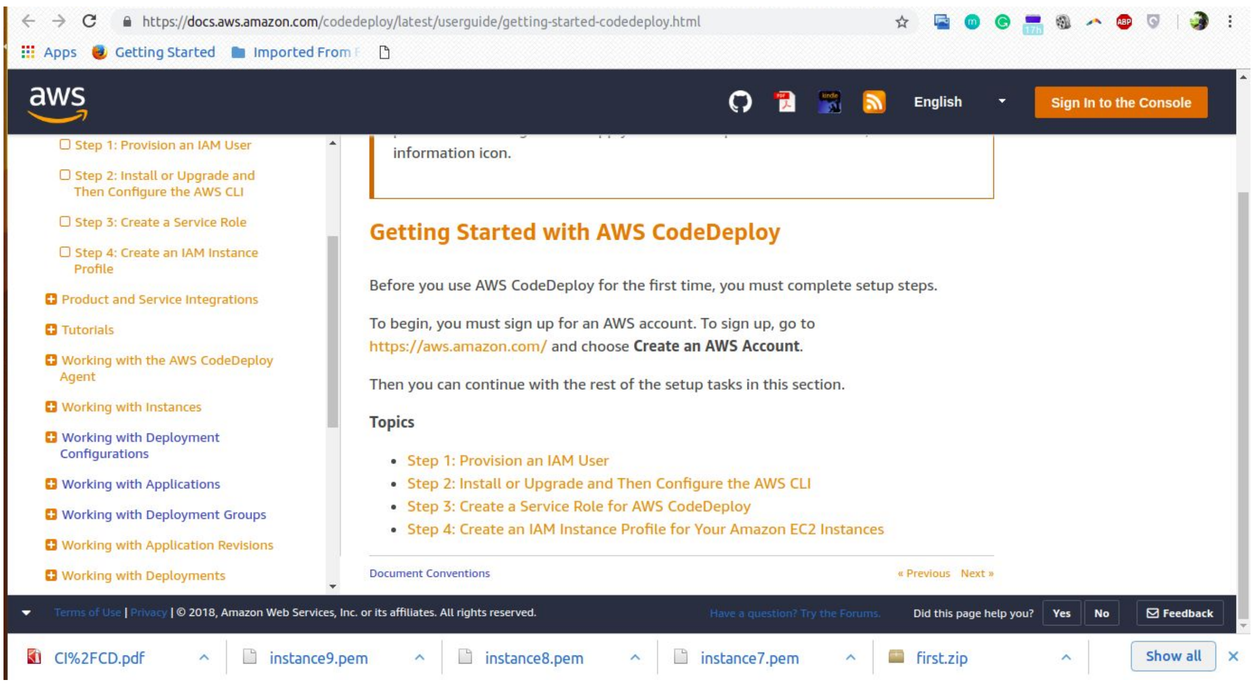
Step 4: Setup an AWS account and get acquainted with CodeDeploy

Create an account on AWS, if you don’t have one.

Visit the below link to know more about AWS CodeDeploy
https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html

Step 5 : Getting started with AWS CodeDeploy

I have used the below link for Step 5.
https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-codedeploy.html


https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-codedeploy.html
Apps
Getting Started
Imported From
aws
English
Sign In to the Console
Step 1: Provision an IAM User
Step 2: Install or Upgrade and Then Configure the AWS CLI
Step 3: Create a Service Role
Step 4: Create an IAM Instance Profile
Product and Service Integrations
Tutorials
Working with the AWS CodeDeploy Agent
Working with Instances
Working with Deployment Configurations
Working with Applications
Working with Deployment Groups
Working with Application Revisions
Working with Deployments

information icon.

Getting Started with AWS CodeDeploy

Before you use AWS CodeDeploy for the first time, you must complete setup steps.

To begin, you must sign up for an AWS account. To sign up, go to https://aws.amazon.com/ and choose **Create an AWS Account**.

Then you can continue with the rest of the setup tasks in this section.

Topics

- Step 1: Provision an IAM User
- Step 2: Install or Upgrade and Then Configure the AWS CLI
- Step 3: Create a Service Role for AWS CodeDeploy
- Step 4: Create an IAM Instance Profile for Your Amazon EC2 Instances

Document Conventions
« Previous Next »
Terms of Use | Privacy | © 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Have a question? Try the Forums.
Did this page help you? Yes No
Feedback
CI%2FCD.pdf
instance9.pem
instance8.pem
instance7.pem
first.zip
Show all

1. Follow the steps exactly as given in “Step 1: Provision an IAM user” which you will find in the below link: https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-provision-user.html
2. Step 2 shows how you can install or upgrade on aws cli on your local machine. Though this step is not mandatory (as we run aws cli commands in a docker environment and not on our local machine), I recommend you to follow this step as you can get acquainted with some basic aws commands, and it also helps for testing purposes from your local machine. https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-configure-cli.html
3. Create a Service Role by following “step 3 : create a service role” exactly as given in the link : https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-create-service-role.html
4. Create an instance profile by following the steps given here: https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-create-iam-instance-profile.html

Step 6: Create a GitHub repo and add your project in it:
Create a github repository.
Push your project to it.
We have not yet intgrated circleCI with it.

Step 7: config.yml file and add environment variables on circleCI project:

Refer to this link to know about config.yml file:
https://circleci.com/docs/2.0/configuration-reference/
Check this link to know more about caching dependencies:

https://circleci.com/docs/2.0/caching/

As of now create a simple “config.yml” file in “.circleci” folder. You can use the one which I am using, but you should remove all the steps from “Setup aws configure file variables onwards”.

We are doing this so that we can add environment variables in our circleCI project before we can actually do the deployment part.

In the CircleCI application, go to your project's settings by clicking the gear icon next to your project. In the Build Settings section, click on Environment Variables. Import variables from another project by clicking the Import Variable(s) button. Add new variables by clicking the Add Variable button.

Here are the variables that you gonna add:

1. **AWS_ACCESS_KEY_ID** : Access key ID of your IAM user
2. **AWS_SECRET_ACCESS_KEY** : Secret Access key of your IAM user
3. **AWS_REGION : ** The AWS Region which you are using for the entire process. I have used “us-east-1” (N. Virginia)

I have used the default output format as “json”.

Step 8: Provision an EC2 Instance:

Follow this link:
https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials-github-provision-instance.html
I have worked with ubuntu 16.04 AMI (free tier)

Step 9: Create an application and deployment group

Follow this link:
https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials-github-create-application.html

Step 10: Appspec.yml file

Go through the below link to know about it
https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file.html

And add the file to your repository.
Appspec.yml file basically tells AWS CodeDeploy on how to go with the deployment process to the EC2 instance.

Step 10: Deployment part of the config.yml file

Add the steps for deployment in the config.yml file.

Step 11: Check your security groups of your EC2 instance

1. Go to the "Network & Security" -> Security Group settings in the left hand navigation
2. Find the Security Group that your instance is apart of
3. Click on Inbound Rules
4. If you are running your localhost on a port say “3000” , add a “custom TCP rule” and mention the port there. Change source to “Anywhere”.
5. Click on Apply

Step 12: Final Step!!

That’s it! We are done with the configuration part. Just push your changes to GitHub after adding the config.yml and appspec.yml file. CircleCI starts building, you can view it in your circleCI dashboard. And you can check your deployment, by connecting to the iPv4 server