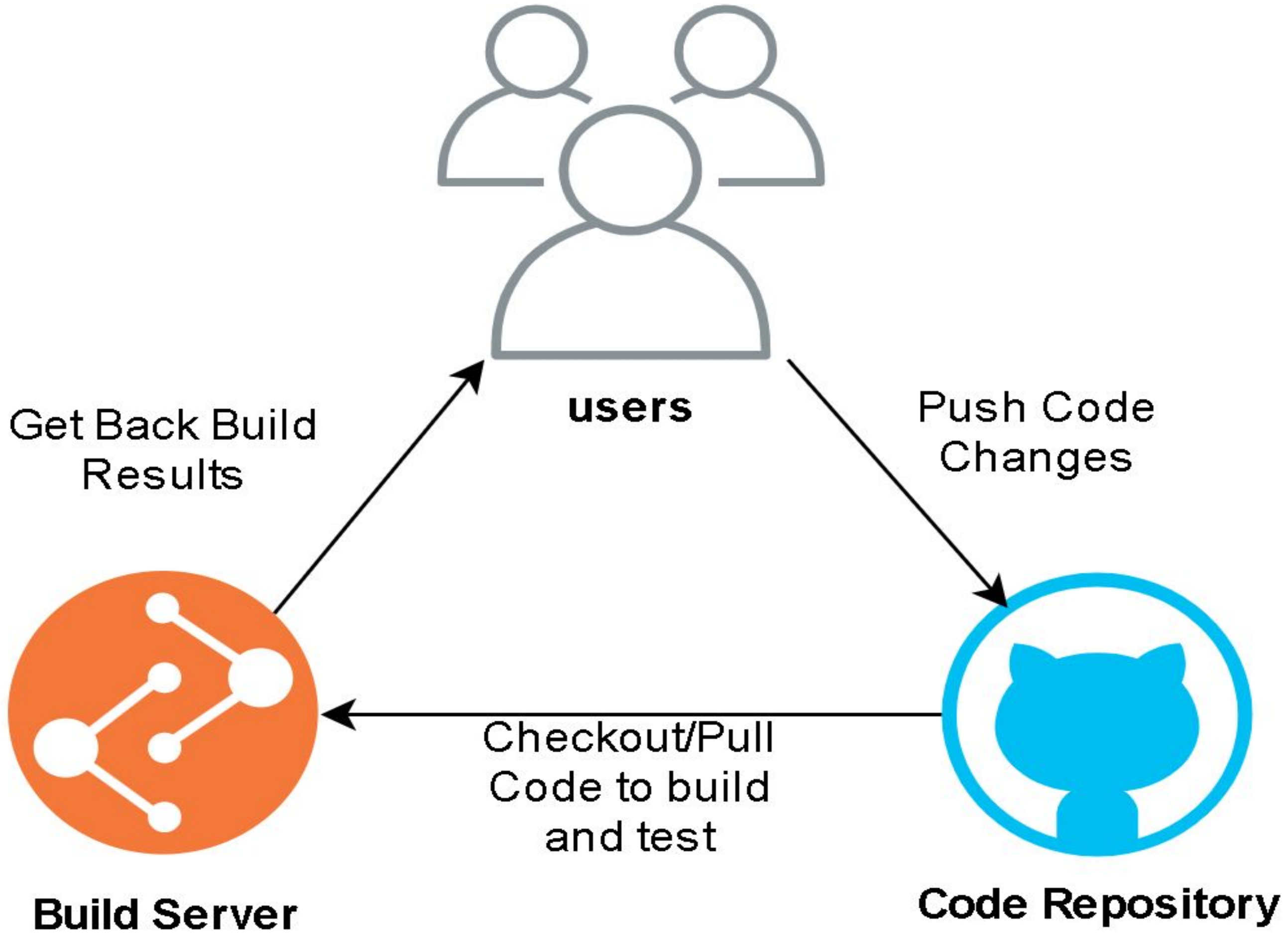
Continuous Integration?

- Developers push the code to a **code repository** often (**GitHub / CodeCommit / Bitbucket / Gitlab etc**) - all of these DVCS (git commands are same)
- A **testing / build server** checks/pulls the code from *Code Repo* as soon as it's pushed (**CodeBuild / Jenkins CI , Travis CI , Circle CI** etc)
- The developer gets feedback about the tests and checks that have passed / failed.
- Deliver faster as the code is tested
- Deploy often
- Find bugs early, fix bugs

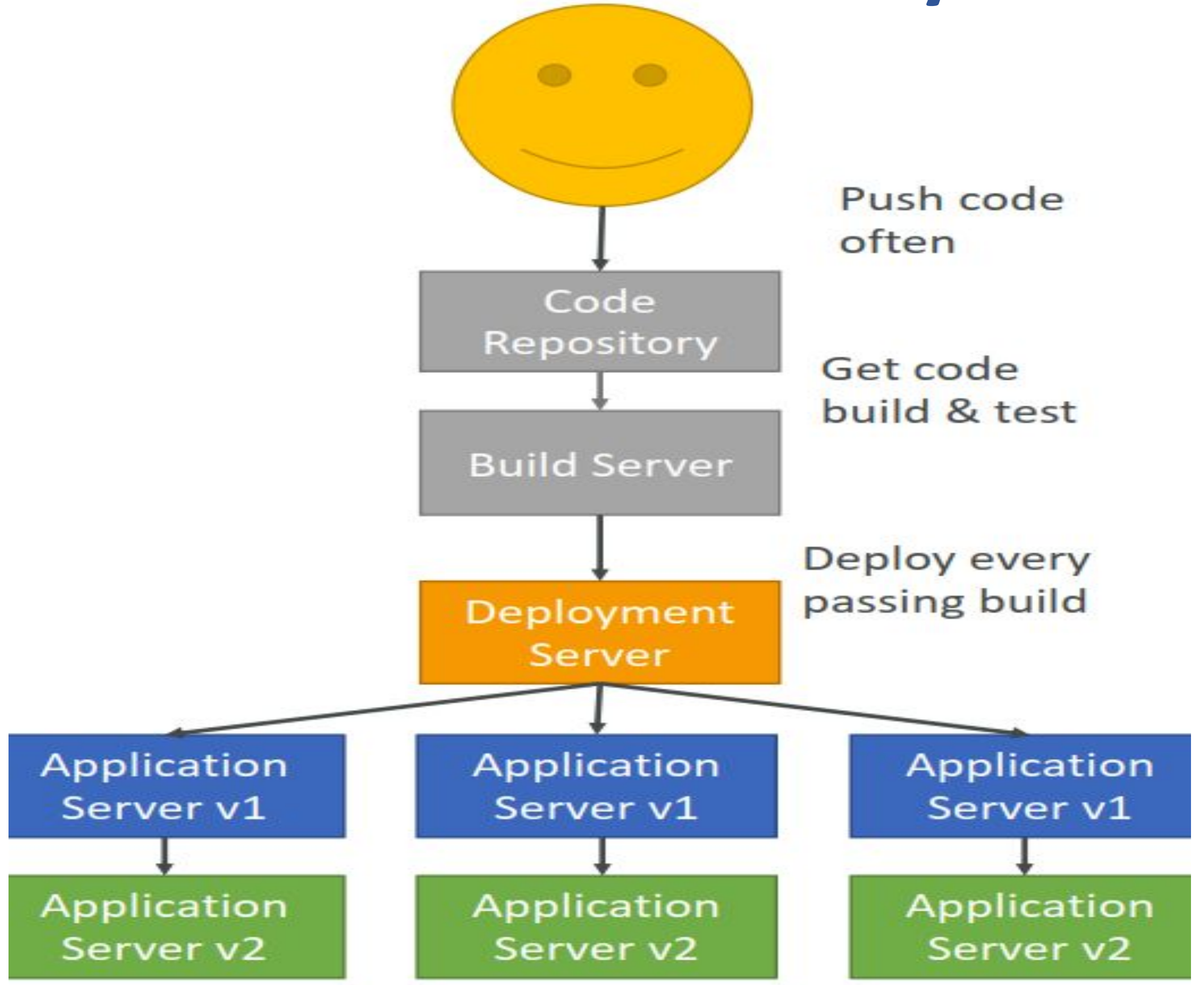
Continuous Integration



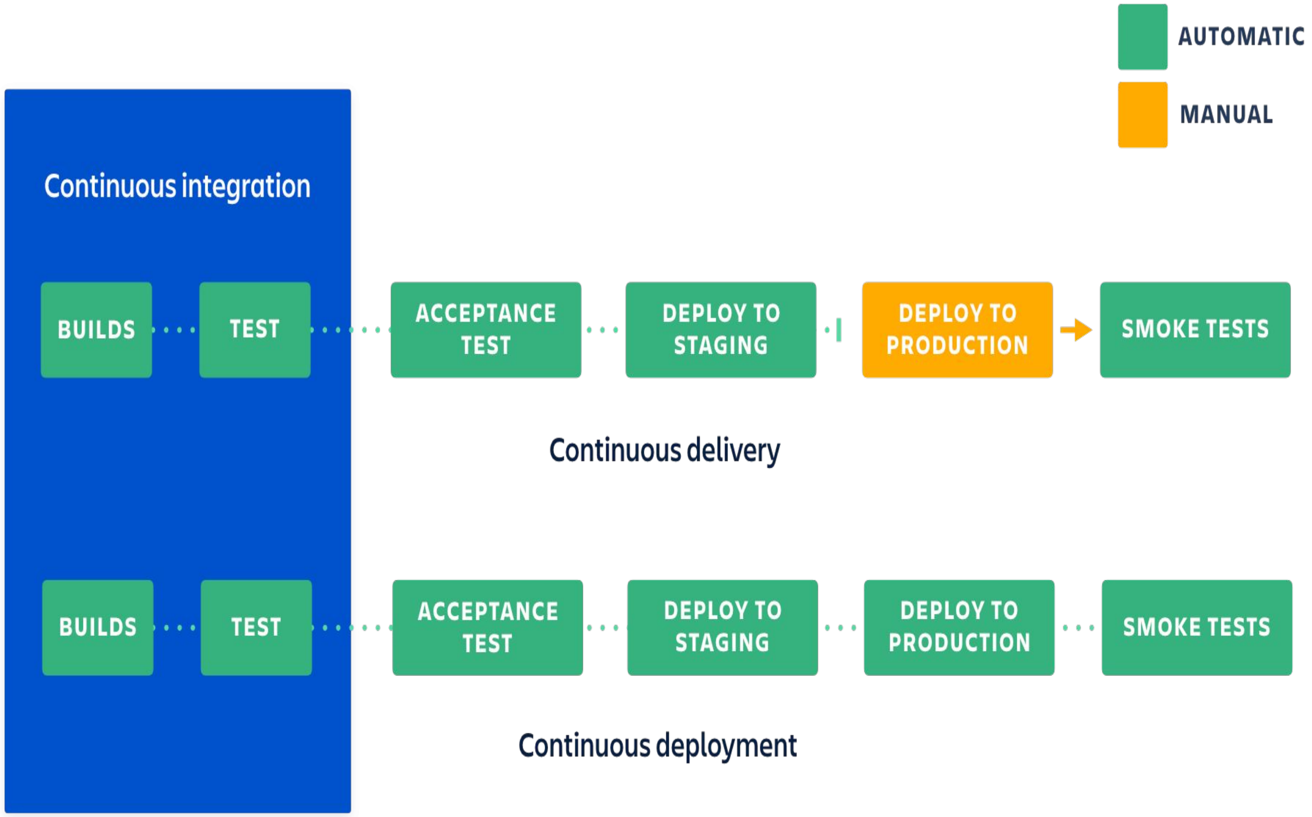
Continuous Delivery?

- Ensure that the software can be released reliably whenever needed
- Quick Deployments
- Shift from “one release every 3 months” to ”5 releases a day”
- That usually means automated deployment
  - **CodeDeploy**
  - **Jenkins CD**

Continuous Delivery?



Continuous Delivery & Continuous Deployment



Continuous Delivery:
Ability to deploy often using automation

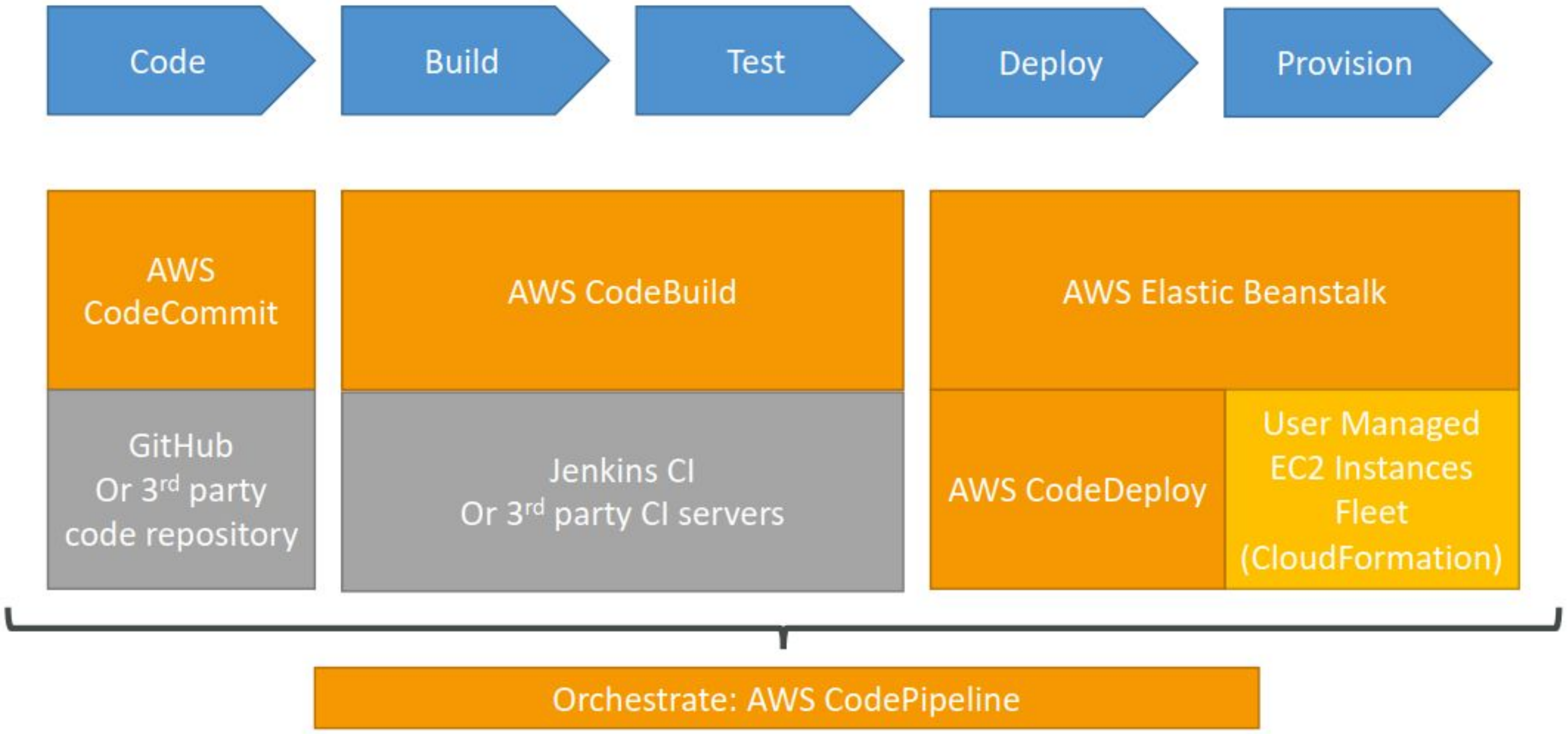
May involve a manual step to "**approve**" a deployment

The deployment itself is still automated and repeated!

Continuous Deployment:
Full automation, every code change is deployed all the way to production

No manual intervention of approvals

Technology Stack for CICD


Code
Build
Test
Deploy
Provision
AWS CodeCommit
AWS CodeBuild
AWS Elastic Beanstalk
GitHub Or 3rd party code repository
Jenkins CI Or 3rd party CI servers
AWS CodeDeploy
User Managed EC2 Instances Fleet (CloudFormation)
Orchestrate: AWS CodePipeline

AWS CodeDeploy

- AWS CodeDeploy is a **fully managed deployment service** that automates the deployment of the application to Amazon **EC2 instance/on-premise instances**.

- **<u>Components of AWS CodeDeploy</u>**
- **Application**: It represent the application name that needs to get deployed. This name also ensures the correct combination of the deployment group, configuration, and revision.
- **Deployment group**: It represents the individual **tagged instances** or an instance in auto scaling groups where the deployment needs to be done.
- **Deployment configuration**: It consists of a set of deployment rules and some condition of deployment success and failure used by CodeDeploy at the time of deployment.

- **Deployment type:** The deployment strategy used to make the latest application available on instances or auto scaling after the deployment. AWS CodeDeploy provides two different deployment types:

  In-place deployment

  Blue-green deployment
- **Revision**: It is basically an archive file, which contains **source code**, **deployment scripts**, and the **appspec.yml** file. It is stored in the **S3 bucket** or **GitHub/CodeCommit repository.**
- **Service role**: In case of AWS CodeDeploy, a service role generally has the following permissions:

 - Reading the tags of instances and auto scaling groups to identify the instances in which an application can get deployed.

 - Performing operations on instances, auto scaling groups, and ELB.

 - Accessing SNS for publishing information & CloudWatch for alarm monitoring

How CodeDeploy Works

1. Each EC2 Machine (or On Premise machine) must be running the **CodeDeploy Agent.**
2. The **CodeDeploy Agent** is *continuously polling* AWS CodeDeploy Service for work to do.
3. CodeDeploy sends **appspec.yml** file (should be present on root level directory of your revision file (app.zip))
4. Application is pulled from **GitHub or S3**.
5. CodeDeploy Agent on EC2 will run the deployment instructions which are specified in the appspec.yml
6. CodeDeploy Agent will report of **success / failure** of deployment on the instance.

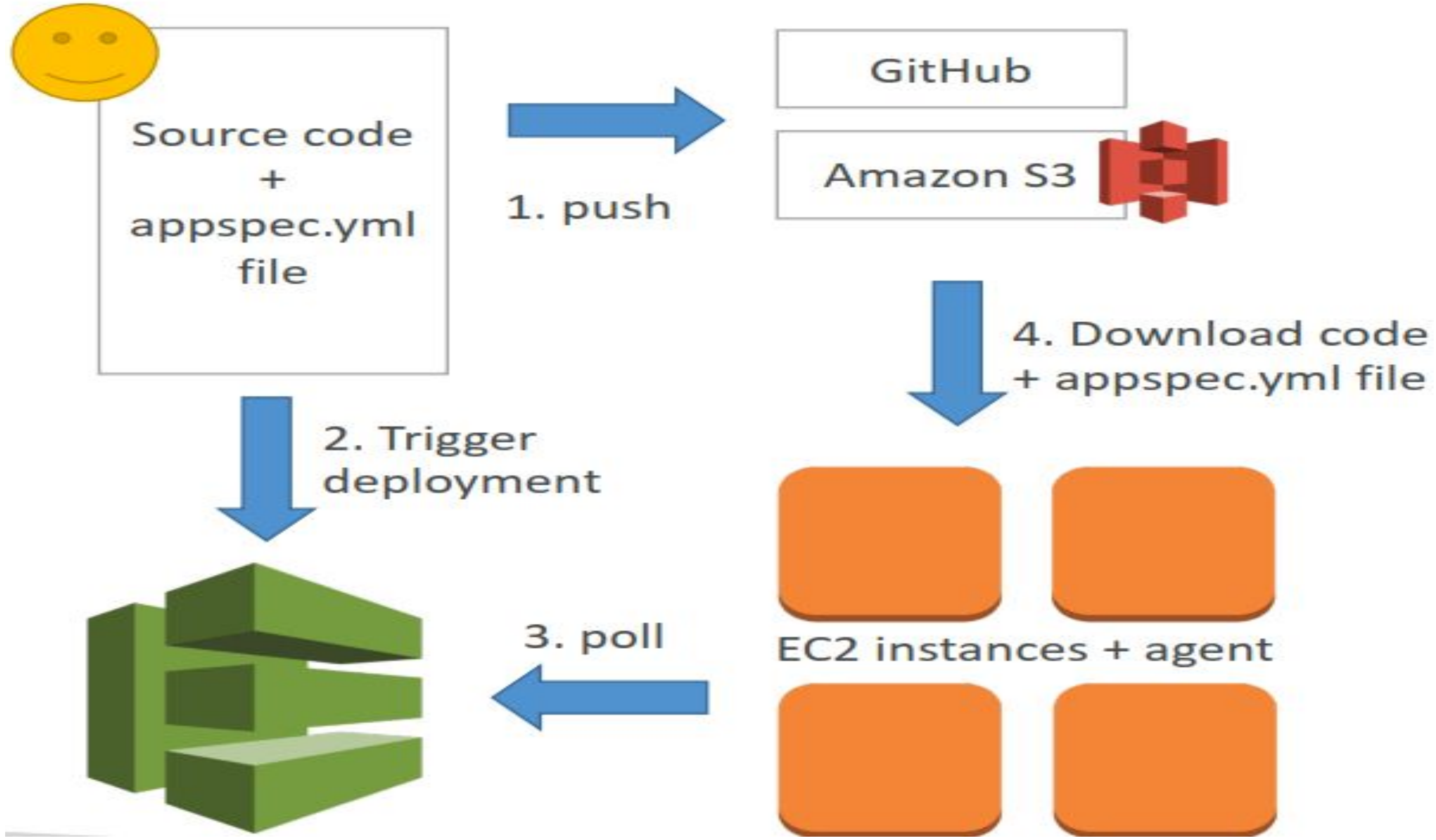
How CodeDeploy Works

- EC2 instances are grouped by deployment group (**dev / test / prod**)
- Lots of flexibility to define any kind of deployments.
- CodeDeploy can be chained into CodePipeline and use artifacts from there.
- CodeDeploy can re-use existing setup tools, works with any application, auto scaling integration.
- Note: Blue / Green only works with EC2 instances (not on premise)
- **Note : CodeDeploy does not provision resources.**

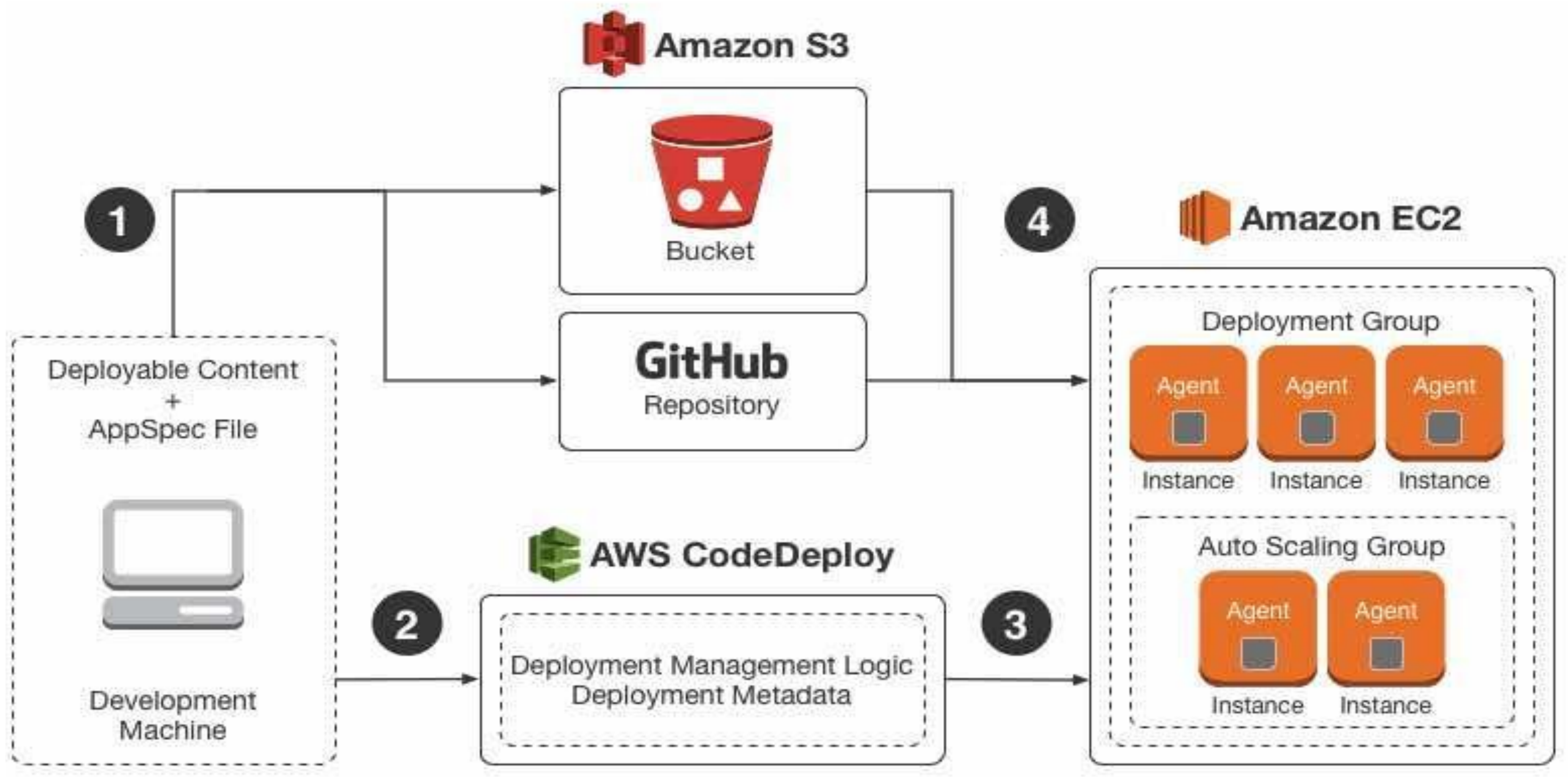
Application Specification Files

- The AppSpec file is a YAML-formatted file that is used to manage each deployment as a series of lifecycle event hooks.
- For EC2/On-Premises compute platform, **appspec.yml** file must be placed in the root of the directory structure of an application's source code. This file is used to:
  - Map the source files in your application revision to their destinations on the instance.
  - Specify custom permissions for deployed files.
  - Specify scripts to be run on each instance at various stages of the deployment process.

How CodeDeploy Works



How CodeDeploy Works



CodeDeploy Features

- AWS CodeDeploy is the deployment service provided by AWS that automates the deployment of the application to Amazon EC2 instance, Elastic Beanstalk, and on-premise instances.
- CodeDeploy can deploy application files stored in Amazon S3 buckets, GitHub/Bitbucket repositories.
- CodeDeploy can be chained into CodePipeline and use artifacts from there.
- Support for AWS Lambda deployments, EC2/On-premises

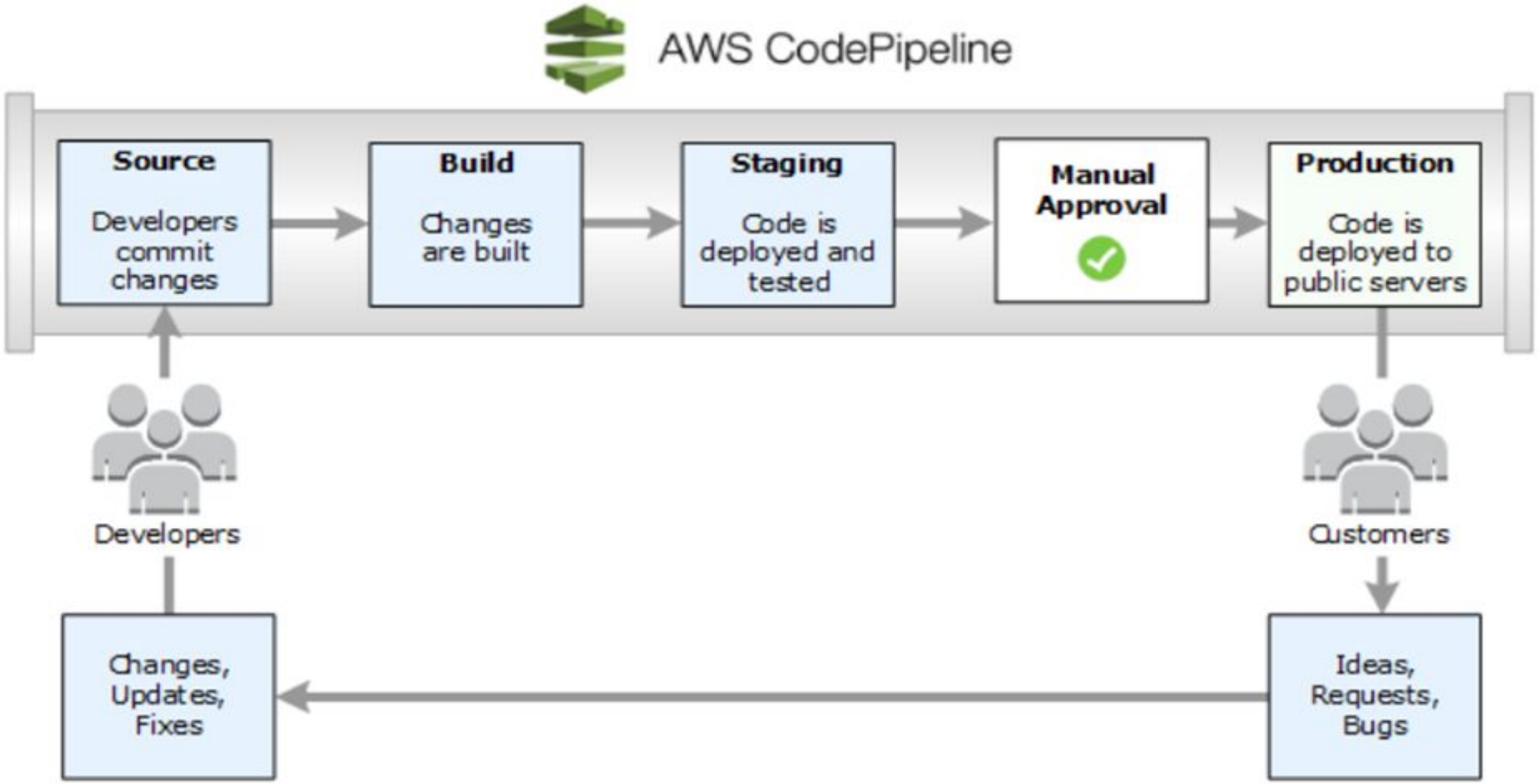
CodeDeploy Pricing

- There is no additional charge for code deployments to **Amazon EC2 or AWS Lambda** through AWS CodeDeploy.
- For more info visit CodeDeploy Pricing

AWS CodePipeline

- AWS CodePipeline is a continuous delivery service to model, visualize, and automate the steps required to release your software.
- CodePipeline trigger can be a full deployment pipeline, when developer pushes code to your CodeCommit repository, which will then start the series of pipeline stages:
  - Pull source code from the CodeCommit repository.
  - Run the CodeBuild project to build and test the artifact file. Upload the artifact to AWS S3 bucket.
  - Trigger CodeDeploy deployment to fetch the artifact and deploy it to your instances.

AWS CodePipeline



AWS CodePipeline

A pipeline defines your release process workflow, and describes how a new code change progresses through your release process.

<u>**Components of AWS CodePipeline**</u>

- **Stage** : A stage is a group of one or more actions. A pipeline can have two or more stages.
- **Action** : An action is a task performed on a revision. Pipeline actions occur in a specified order. Actions can be as :
  - Source
  - Build
  - Test
  - Deploy
  - Approval
- When an action runs, it acts upon a file or set of files called artifacts.

AWS CodePipeline

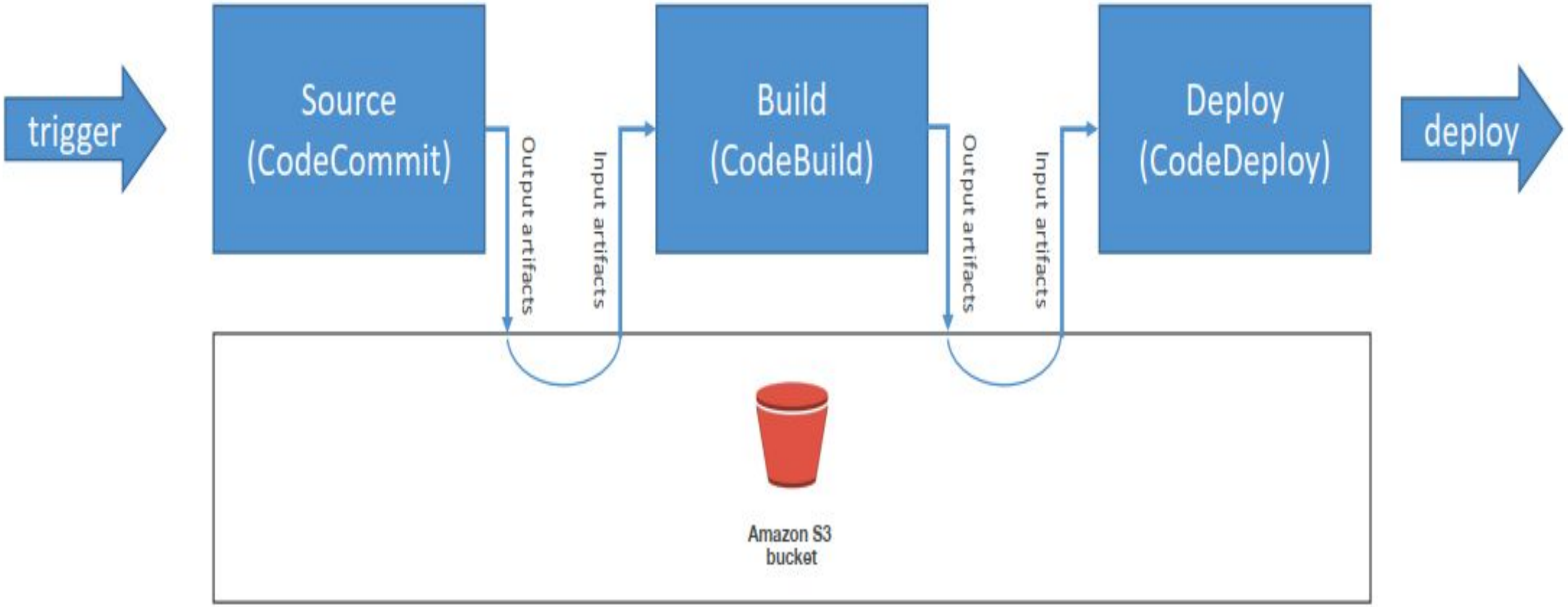
- **Transition :** The stages in a pipeline are connected by transitions.
  - Transitions can be disabled or enabled between stages.
- An **approval** action prevents a pipeline from transitioning to the next action until permission is granted (for example, through manual approval from an authorized IAM user). This is useful when you are performing code reviews before code is deployed to the next stage

Note: *A stage with an approval action is locked until the approval action is approved or rejected or has timed out. A timed-out approval action is processed in the same way as a failed action.*

Input and output artifacts

- CodePipeline integrates with development tools to check for code changes and then build and deploy through all of the stages of the continuous delivery process.
- Stages use input and output artifacts that are stored in the Amazon S3 artifact bucket you chose when you created the pipeline.
- CodePipeline zips and transfers the files for input or output artifacts as appropriate for the action type in the stage.

AWS CodePipeline Artifacts



Pricing

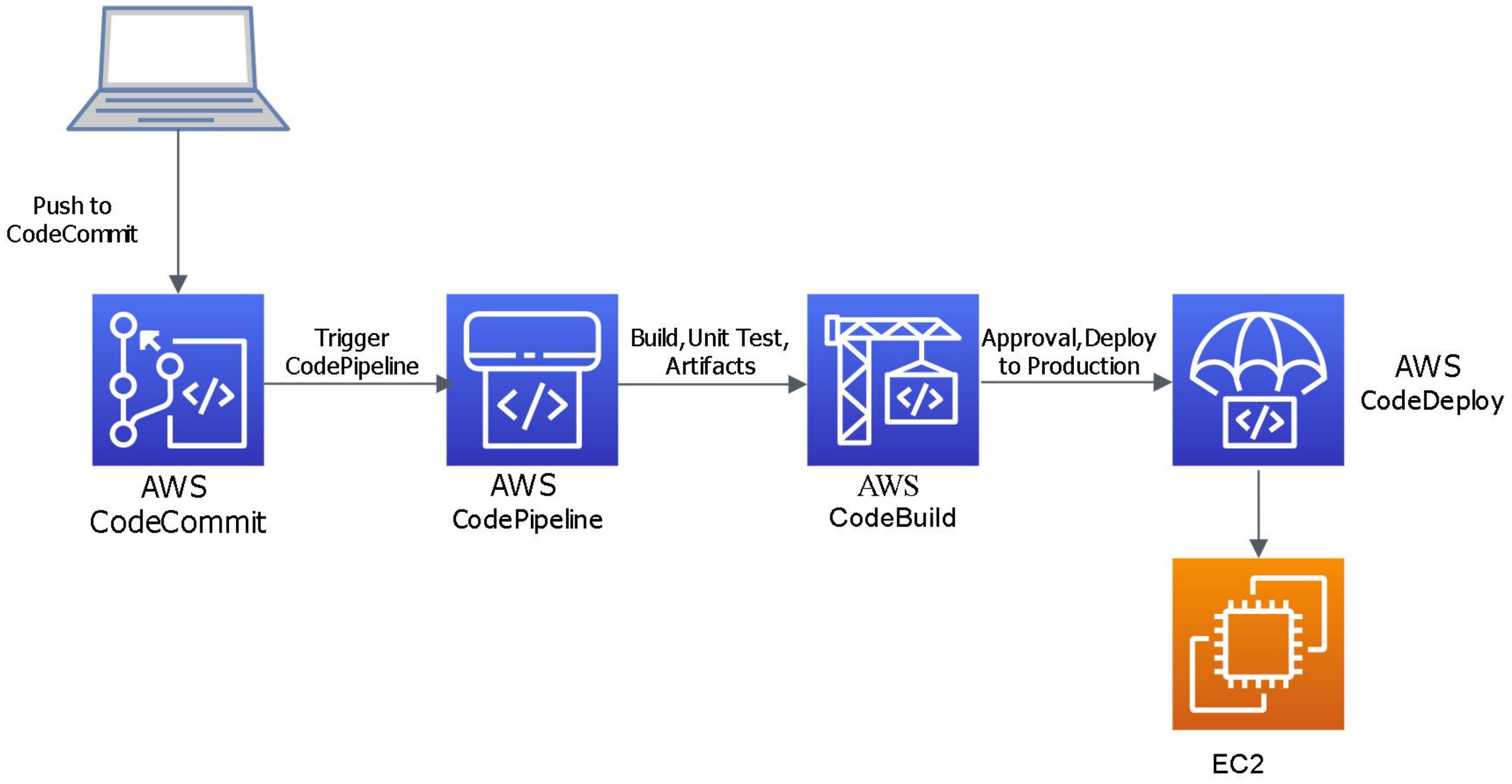
CodeDeploy Pricing

- There is no additional charge for code deployments to **Amazon EC2 or AWS Lambda** through AWS CodeDeploy.
- For more info visit CodeDeploy Pricing

CodePipeline Pricing

- Pipelines are free for the first 30 days after creation.
- AWS CodePipeline costs $1.00 per active pipeline* per month.
- For more info visit CodePipeline Pricing

AWS CICD Flow



AWS CICD Flow