
Linux Academy
Amazon Web Services
What is CodePipeline?

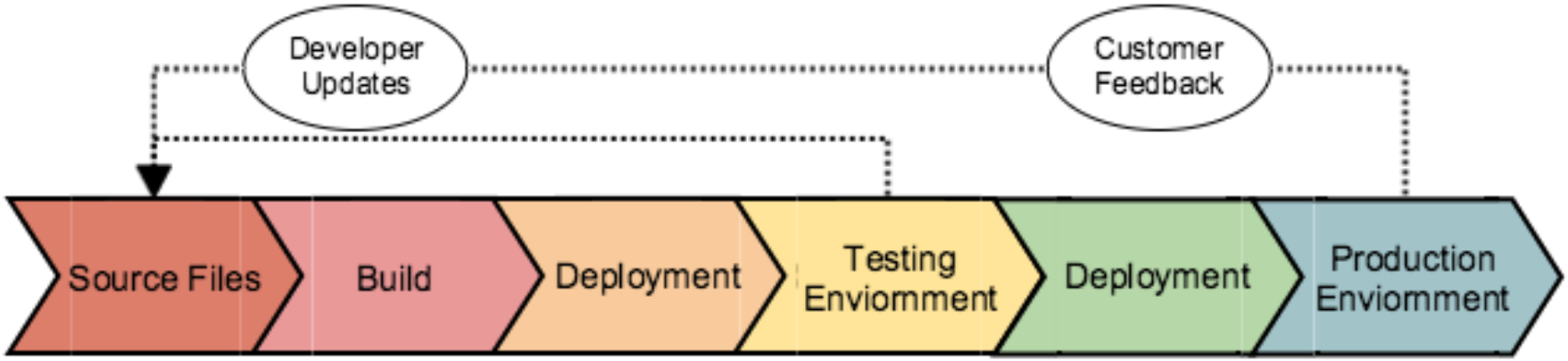
What is CodePipeline?

- CodePipeline is a continuous delivery service. It provides the tools to model, visualize, and automate the many steps that are required as part of the software release process.

- ***Automation:***
  - You can easily automate the entire release process, from the source repository all the way to production servers.
- ***Consistency:***
  - Create and repeat a consistent set of steps each time you want to release or update your software.
- ***Speed up the delivery process:***
  - Speed up the release process through automation.
- ***Use existing tools:***
  - CodePipeline integrates with many other AWS services (CodeCommit, CodeDeploy, S3, OpsWorks), as well as other major developer and DevOps platforms such as Jenkins and GitHub.
- ***Easy to visualize and view:***
  - See each stage of the release process, their status, and position.

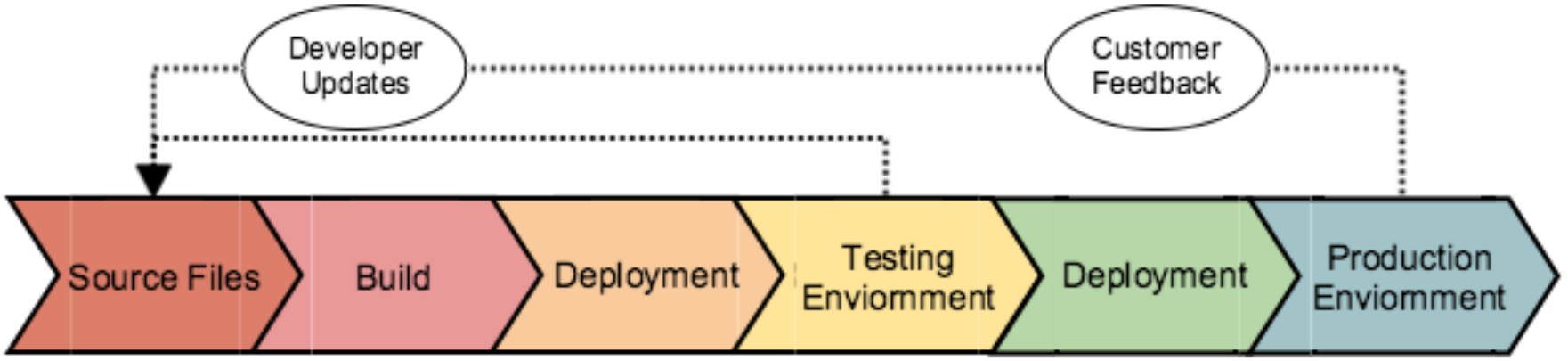
Linux Academy

Common Software Release Process:



Linux Academy

Common Software Release Process:



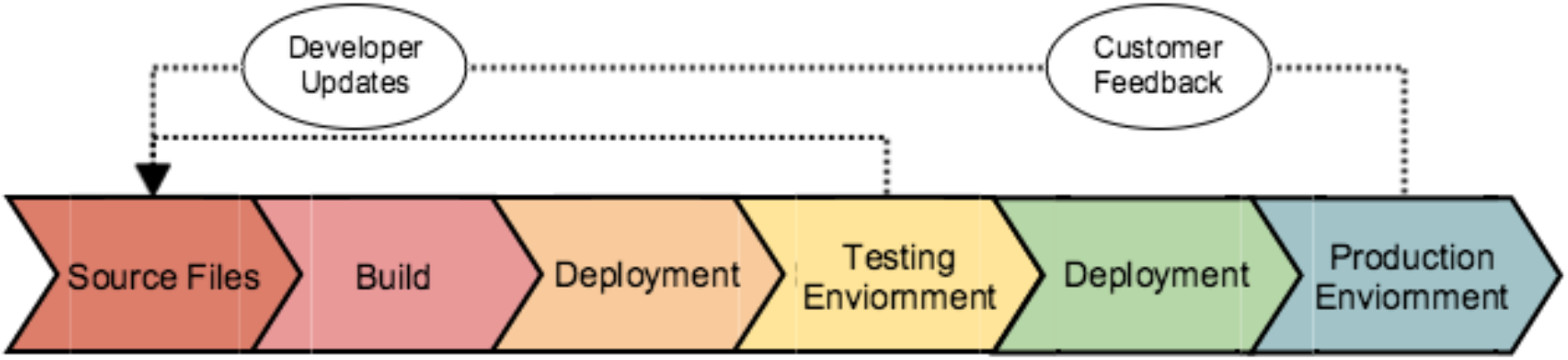
CodeCommit

S3

Github

Linux Academy

Common Software Release Process:



CodeCommit

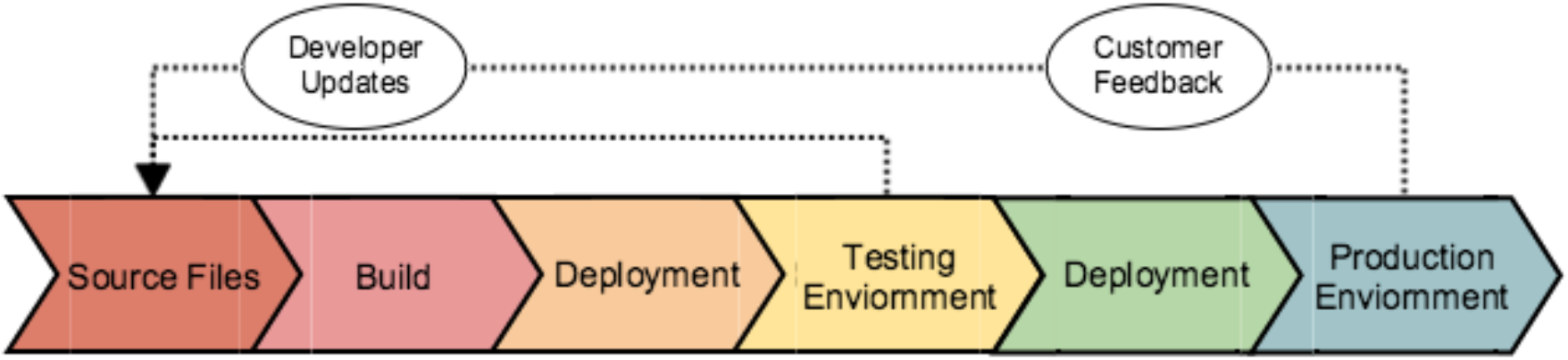
S3

Github

Jenkins

Linux Academy

Common Software Release Process:



CodeCommit

S3

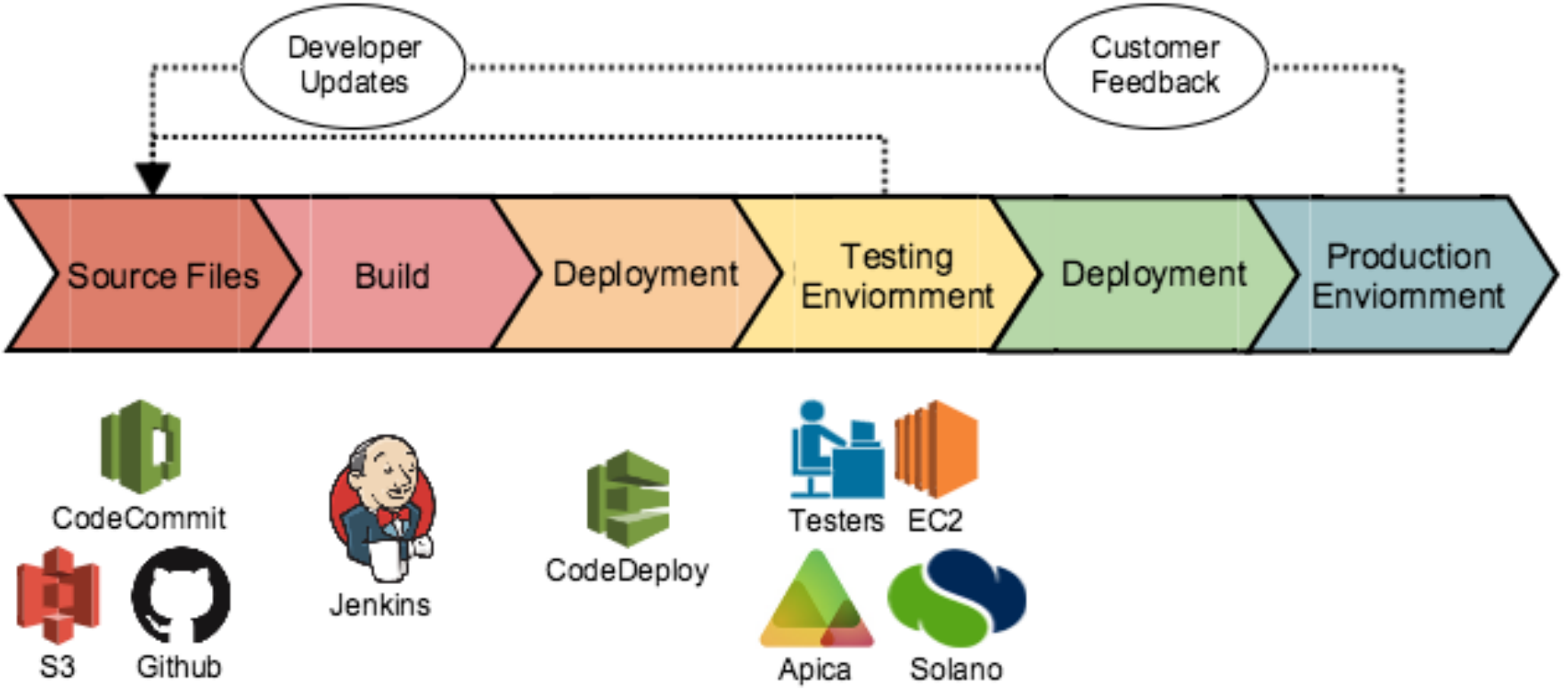
Github

Jenkins

CodeDeploy

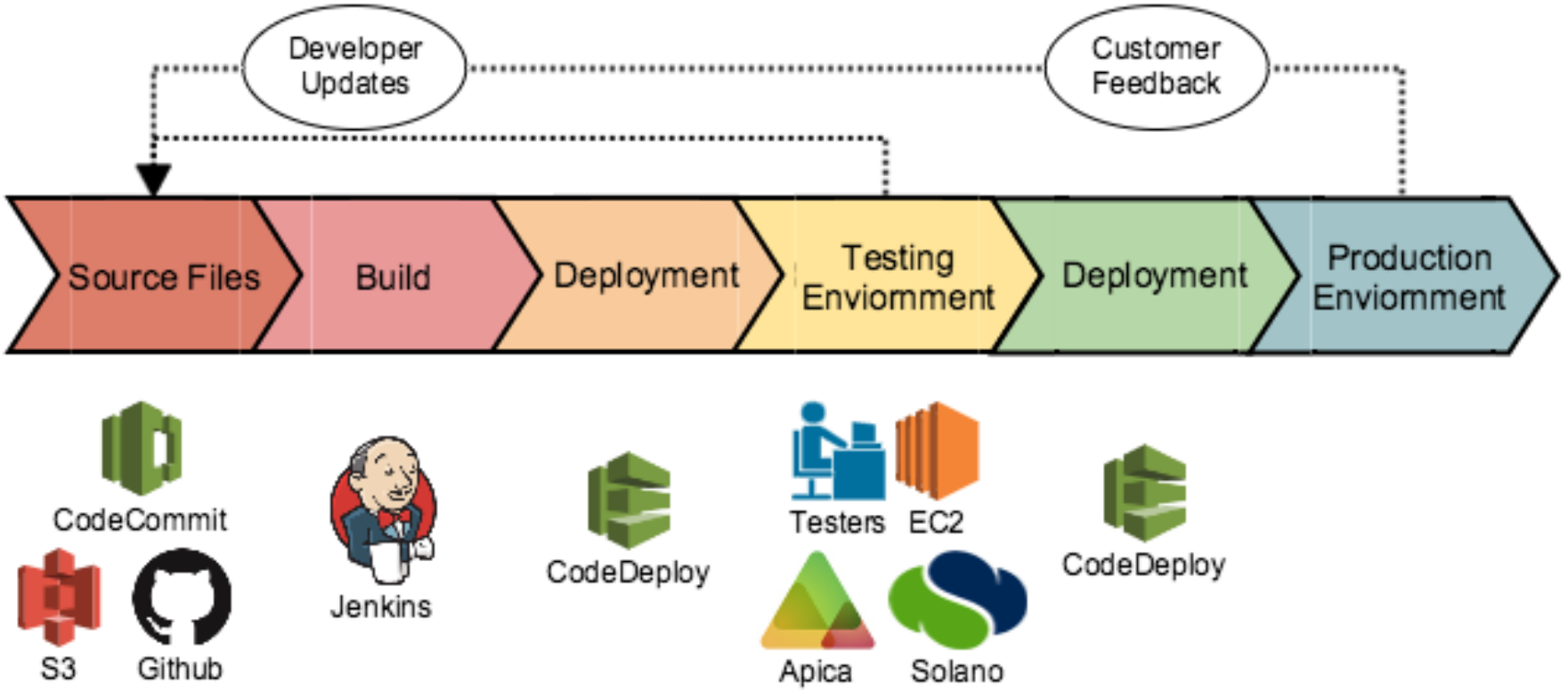
Linux Academy

Common Software Release Process:



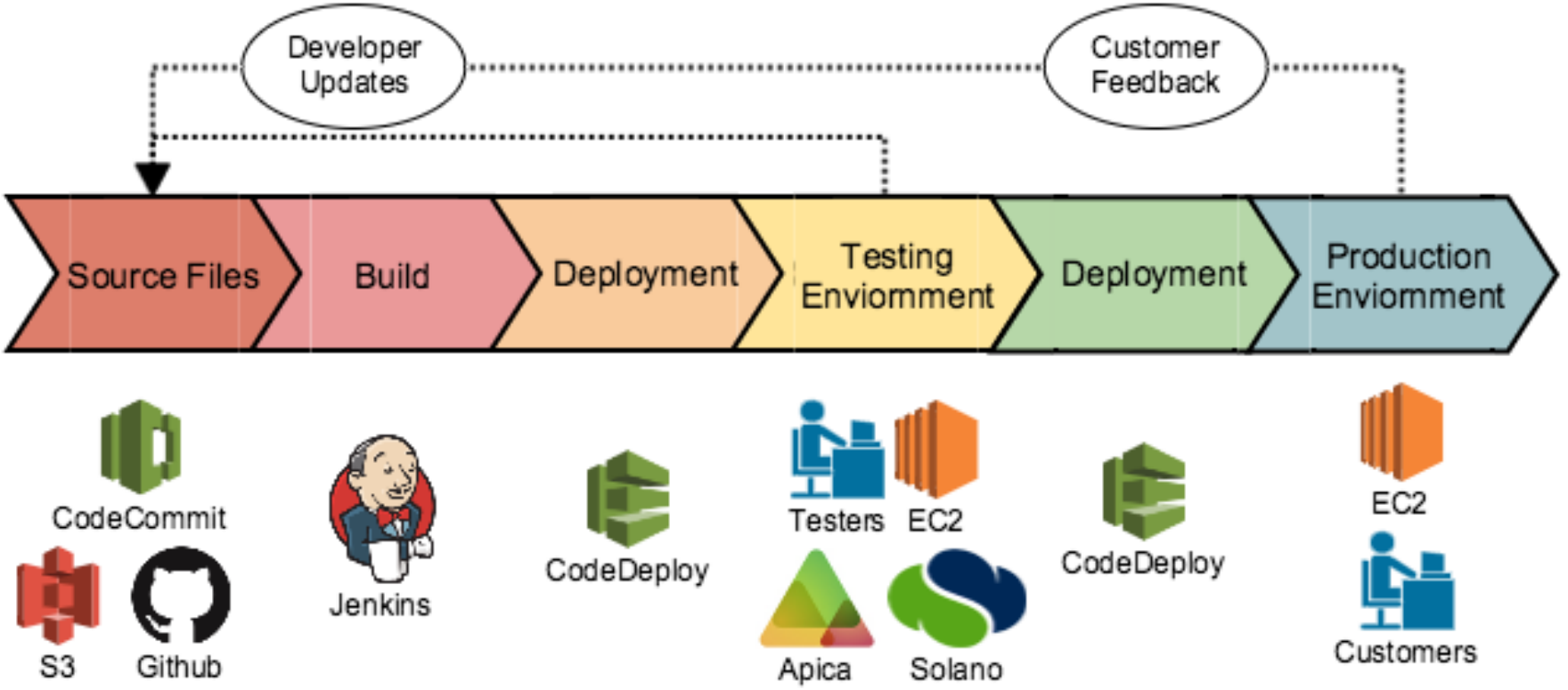
Linux Academy

Common Software Release Process:



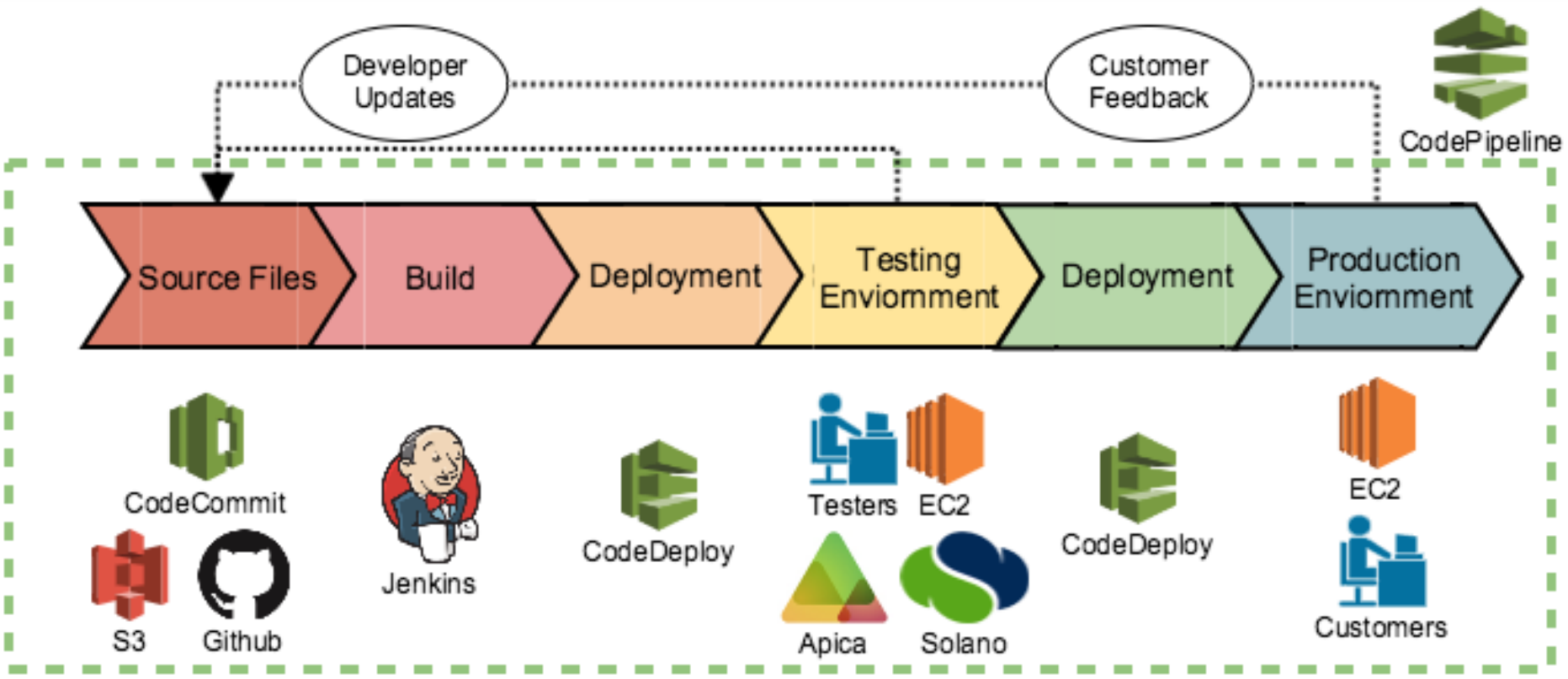
Linux Academy

Common Software Release Process:



Linux Academy

Common Software Release Process:




Linux Academy
Amazon Web Services
What is CodeDeploy?
Thank you for watching!


Linux Academy
Amazon Web Services
CodePipeline: Concepts & Terminolgy

Continuous Delivery:

- A software engineering approach where teams produce software in short cycles, ensuring that the software can be reliably released at any time. It aims for building, testing, and releasing software faster and more frequently. The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production. ***A straightforward and repeatable deployment process is important for continuous delivery.***

Continuous Integration:

- The practice of merging all developer working copies to a shared mainline (such as a master branch), at a continuous pace (often several times a day). Each addition (change) is built and tested as quickly as possible.

AWS CodePipeline Concepts & Terminology:

Pipeline:

- A workflow framework that helps you create and manage the release process. (blueprint)
- It is how you specify, build, coordinate, monitor, and execute your specific release process.
- Consists of:
  - **Stages** (which consist of **Actions**)
  - **Transitions** (between each Stage)
- The first time you create a pipeline (AWS Console) an S3 bucket is automatically created that will store the pipeline's **artifacts**:
  - Created in the same region as your pipeline
  - The bucket is named "codepipeline-<REGION>-<RANDOM_10_DIGIT_#>"
- You can have up to 20 pipelines per AWS account
- Pipelines must be in one of the following regions:
  - us-east-1
  - us-west-2
  - eu-west-1

AWS CodePipeline Concepts & Terminology:

Stages:

- Each pipeline is broken up into broad sections called stages.
- Stages are used to categorize, execute, and monitor **actions**.
- Stages are completed in order – as configured in the pipeline.
- Every stage must have at least one action.
- AWS default stages include:
  - Source
  - Build
  - Beta
- A stage can only process one **revision** at a time.
- Stages are connection through **transitions**.

AWS CodePipeline Concepts & Terminology:

Actions:

- An action is a task performed on the **artifact**.
- Action tasks include:
  - **Source**: Monitors the "source" location (i.e. CodeCommit, S3, GitHub) for new commits or uploaded revision – and starts the release process if found
  - **Build**: Code is built (i.e. Jenkins)
  - **Test:** Run the code through a test provider (i.e. BlazeMeter, Apica, etc.)
  - **Deploy**: Install the application files onto a fleet of instances
  - **Invoke**: Trigger a Lambda function
  - **Approve**: Require human approval before moving to the next stage

Note: Services, such as CodeCommit, Github, Jenkins, Apica, CodeDeploy, etc – are referred to as action "providers"

AWS CodePipeline Concepts & Terminology:

Revisions, Artifacts & Transitions:

- **Revision** is the general term used to describe the code "update" or version that is currently running through the pipeline.
- **Artifacts** refer to the actual set of files (objects) that are being passed through the pipeline, and is categorized by ***input*** or ***output*** artifacts.
- **For example:** The un-built source files being passed into the "build" stage are the "input artifacts." The built files (after running through Jenkins), are the "output artifacts."
- The "output artifacts" of a stage are then passed to the next stage via transitions.
- Artifacts are stored in the S3 bucket that was created or designated when you create a pipeline.
- **Transitions** tell the artifact what stage to go to next, and act as a delivery system between them.
- Transitions can be enabled or disabled to allow or prevent upcoming stages to be run.


Linux Academy
Amazon Web Services
What is CodeDeploy?
Thank you for watching!

Amazon Web Services

CodePipeline: Setup & Configuration

CodePipeline Setup & Configuration:

1) IAM Policy:
   - *Attach the **"AWSCodePipelineFullAccess"** IAM policy to any users or roles that you want to grant CodePipeline access.*

2) Install and configure the AWS CLI:
   - *If you have followed this course from the beginning (CodeCommit section) then you should already have the CLI setup and configured.*

   - *If you skipped ahead to this section (CodePipeline), you will need to set up and configure the AWS CLI. Please refer to the following lesson (located in the CodeCommit section of this course) depending on your operating system:*
     - *For Linux/OSX users:*
       (7) OSX/Linux: AWS CLI Installation
     - *For Windows users:*
       (2) Windows: GIT & AWS CLI Installation

Linux Academy

Amazon Web Services

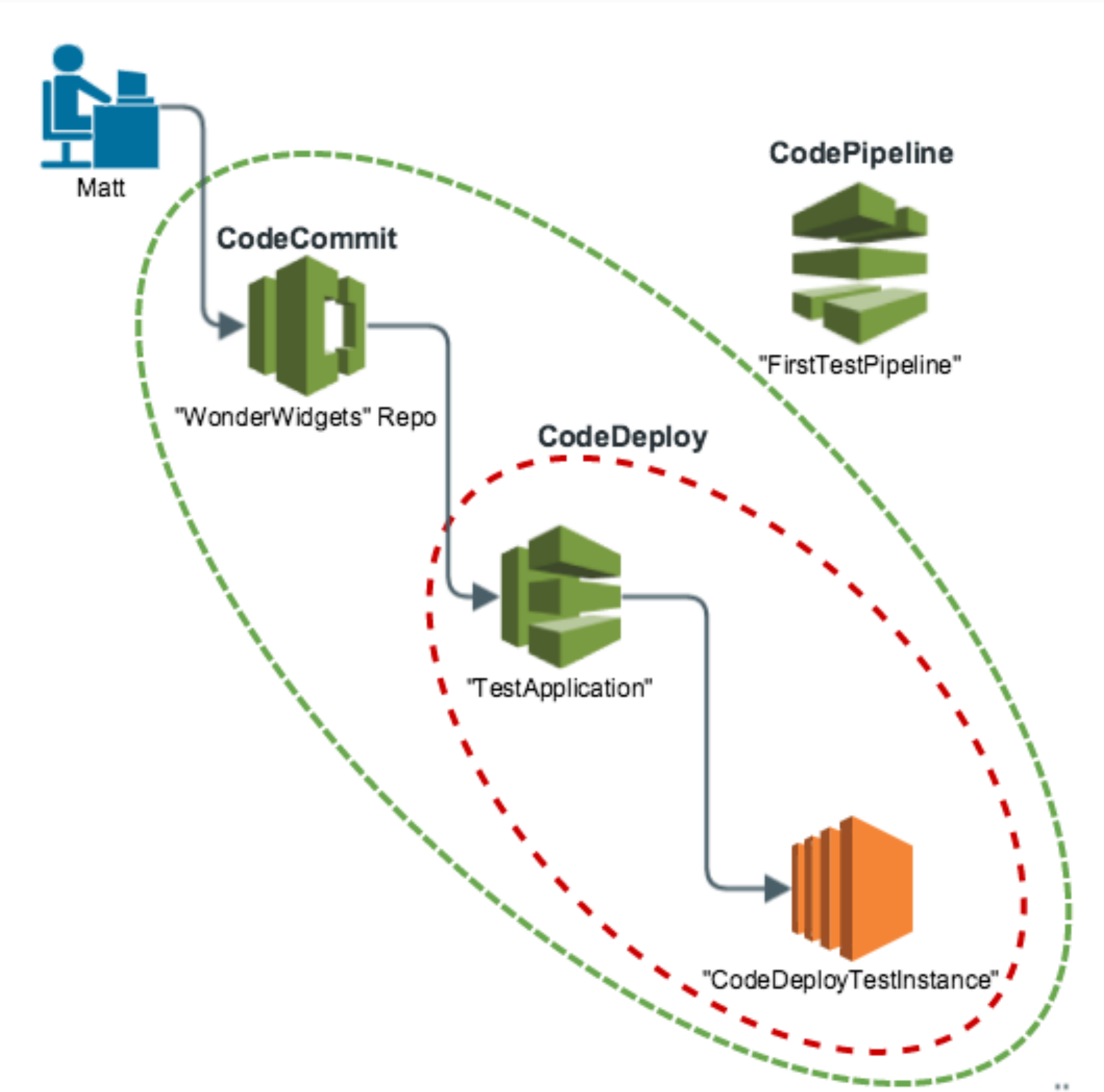
CodePipeline: Setup & Configuration

Thank you for watching!

Amazon Web Services

CodePipeline: Creating a Pipeline

Our First Pipeline:



A Few Notes:

1) *During the "Automating Deployments from S3 Using Lambda" lesson, I moved the AppSpec file from inside the "local-wonderwidgets" repository to the "home" directory of our developer Matt.*

2) *Before this lesson, I moved the AppSpec file back into the "local-wonderwidgets" directory – which is where it needs to be located for standard CodeDeploy execution protocol.*

3) *I have added a new file to the "local-wonderwidgets" directory called "Pipeline.txt" – which we will use to verify that our files have deployed correctly once we have created and executed the new pipeline.*

Creating a Pipeline (AWS Console):

1) *Navigate to CodePipeline and click on “Get Started” or “Create a Pipeline”.*

2) *Give the pipeline a name.*

3) *Select a Source (CodeCommit).*

4) *Select a Build Provider (optional).*

5) *Select a Deployment Provider (CodeDeploy).*

6) *Create/Select an AWS Service Role (permissions).*

7) *Review the Pipeline & create it.*

8) *View the results.*

Creating a Pipeline (AWS CLI):

1) *Generate a CodePipeline JSON Template:*
   - *The easiest way to do this is to generate the template from an existing pipeline (created in the AWS Console), because you then get:*
     - *Template outline (all sections/formatting)*
     - *Prepopulated fields (such as):*
       - *Service role ARN*
       - *S3 bucket location for artifacts*
   - *To get the template, run the command:*
     aws codepipeline get-pipeline –name <PIPELINE_NAME>

2) *Create a .json file and copy/paste the JSON template into it.*

3) *Edit the JSON template to the specifications you want for the new pipeline.*

4) *Generate the new pipeline by uploading it to AWS using the command:*
   aws codepipeline create-pipeline --cli-input-json file://<FILE-NAME>.json

Linux Academy

Amazon Web Services

CodePipeline: Creating a Pipeline

Thank you for watching!

Amazon Web Services

CodePipeline: Managing a Pipeline from the AWS Console

Items We can Manage in the AWS Console:

1) View pipelines and detailed pipeline information
2) Edit pipelines
3) Disable or enable transitions between stages
4) Retry failed actions
5) Delete pipelines

Viewing Pipelines:

1) Navigate to CodePipeline

2) View all available pipelines by name

3) Click on a pipeline for detailed information*:*
   - *Stage name*
   - *Action name/type*
   - *Actions most recent status*
   - *Source's type*
   - *Source location (repo & branch)*
   - *Source artifact (output)*
   - *Source's most recent commit*
   - *Deployment provider*
   - *Deployment group name*
   - *Deployment application name*
   - *Deployment artifact (input)*

Editing/Adding to Pipelines:

1) Click on “Edit”
   - *Edit or delete and existing stage*
   - *Edit or delete and existing action*
   - *Add a stage & new action*

******Any time that you edit a pipeline and “Save pipeline changes” changes, the pipeline will run all stages again.***

Disable/Enable Transitions:

*If currently **enabled**:*

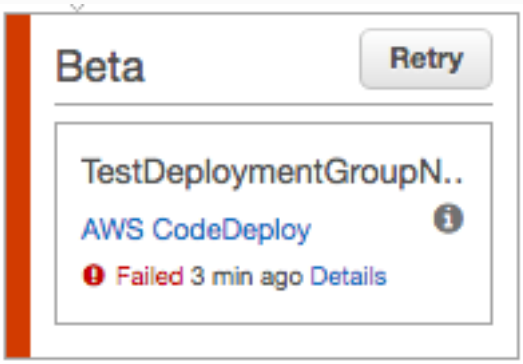
1) Click on the “down arrow” transition icon between stages
2) Enter a reason why you are disabling the transition
3) Click “disable”

*If currently **disabled**:*

1) Click on the “down arrow” transition icon between stages
2) Click “enable”

Retry Failed Actions:

1) If an action fails, the action will look like this:

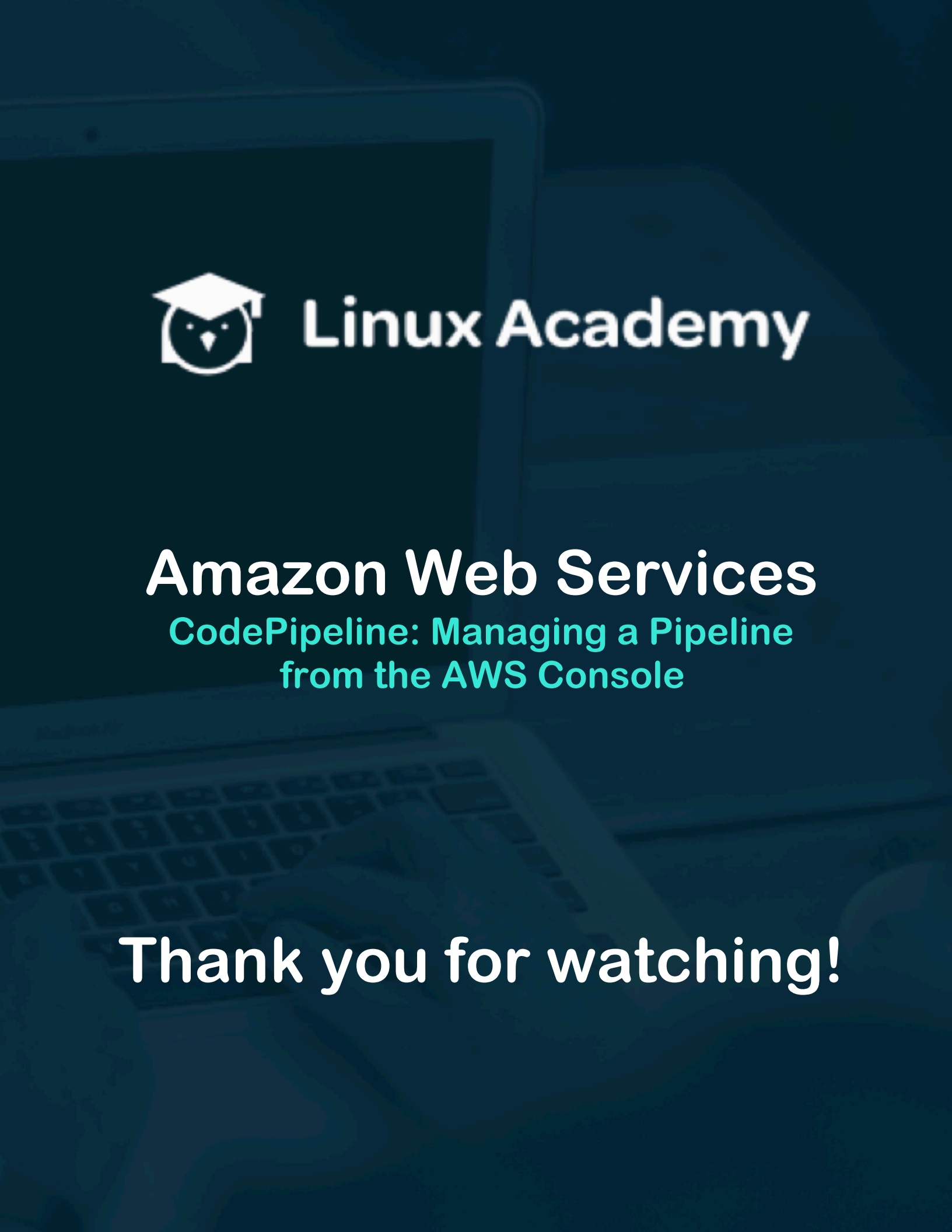


2) Troubleshoot and fix the issue
3) Click on “Retry” OR “Release change” (at the top):
   - *“Retry” if the error was isolated to that action*
   - *“Release change” if the error requires corrected items from previous stages/actions*

******Push a new commit (if the error was in the source files)***

Delete a Pipeline:

1) Click on “Edit”
2) Click on “Delete”
3) Enter pipeline name to confirm
4) Click “Delete”



Amazon Web Services

CodePipeline: Managing a Pipeline from the AWS Console

Thank you for watching!

Amazon Web Services

CodePipeline: Managing a Pipeline from the AWS CLI

Items we can Manage in the AWS CLI:

1) View pipelines and detailed pipeline information
2) Edit pipelines
3) Disable or enable transitions between stages
4) Retry failed actions
5) Delete pipelines

Viewing Pipelines:

1) View a list of all pipelines:
   aws codepipeline list-pipelines

2) View detailed information about a specific pipeline:
   aws codepipeline get-pipeline --name <PIPELINE_NAME>

   - *Stage names*
   - *Action names/type*
   - *Source type*
   - *Source location (repo & branch)*
   - *Source artifact (output)*
   - *Source's most recent commit*
   - *Deployment provider*
   - *Deployment group name*
   - *Deployment application name*
   - *Deployment artifact (input)*

****This command offers a more detailed information about the pipeline than viewing it in the AWS Console.*

Viewing Pipelines:

3) View the most recent “status” of each action of a pipeline:
aws codepipeline get-pipeline-state --name <PIPELINE_NAME>

- *Action status*
- *Revision URL*
- *Revision ID (most recent commit id #)*
- *ExternalExecutionID (deployment ID #)*

Editing/Adding to Pipelines:

1) Download the pipeline structure JSON file:
   aws codepipeline get-pipeline --name <PIPELINE_NAME> ><FILE_NAME>.json

   *****This command will create the .json file for you and populate it with the pipeline structure.***

2) Edit the contents of the .json file to meet specifications you want for your updated/edit Pipeline:
   - *Edit or delete and existing Stage*
   - *Edit or delete and existing Action*
   - *Add a stage & new ction*

3) Upload the edited .json structure file to AWS:
   codepipeline update-pipeline --cli-input-json file://<FILE_NAME>.json

Disable/Enable Transitions:

If currently ***enabled****:*

1) Run the command:

   aws codepipeline disable-stage-transition
   --pipeline-name <PIPELINE_NAME>
   --stage-name <STAGE_NAME>
   --transition-type Inbound
   --reason "<REASON>"

If currently ***disabled****:*

1) Run the command:

   aws codepipeline enable-stage-transition
   --pipeline-name <PIPELINE_NAME>
   --stage-name <STAGE_NAME>
   --transition-type Inbound

Retry Failed Actions:

1) View the most recent “status” of each action the pipeline, and identify the action that failed:

aws codepipeline get-pipeline-state --name <PIPELINE_NAME>

2) Identify and note the stage that failed & the “pipelineExecutionID”
3) Create a .json file and open it in a plain-text editor
4) Input the following lines into the .json file and fill in the appropriate information where required:

```
{
    "pipelineName": “<PIPELINE_NAME>“,
    "stageName": “<FAILED_STAGE_NAME>“,
    "pipelineExecutionId": “<ID #>“,
    "retryMode": "FAILED_ACTIONS”
}
```

5) Troubleshoot and fix the issue
6) Upload the .json file to re-run the failed action:

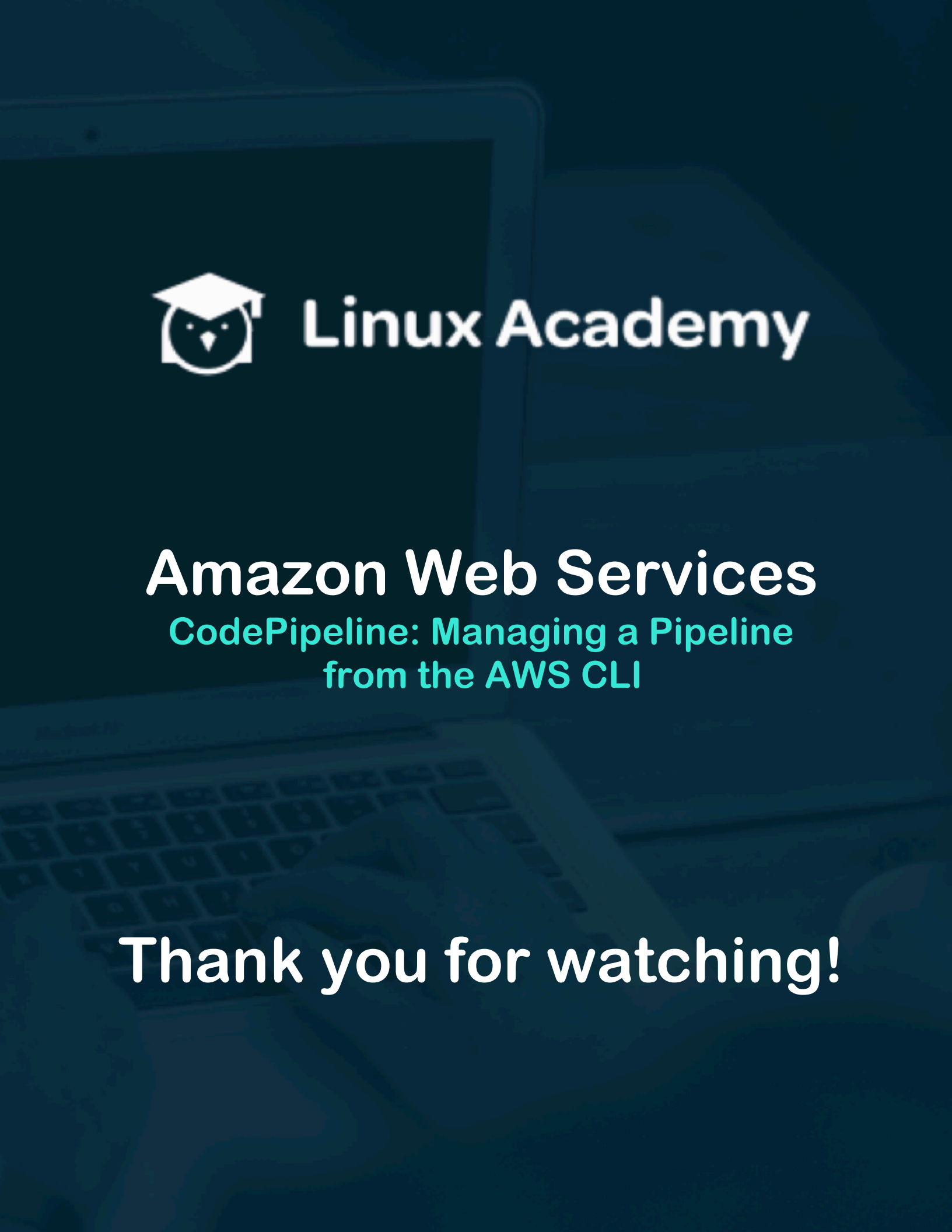
aws codepipeline retry-stage-execution --cli-input-json file://<RETRY_FILE_NAME>.json

- *“Retry” if the error was isolated to that action*
- *Push a new commit (if the error was in the source files)*

Delete a Pipeline:

1) Run the command:
   aws codepipeline delete-pipeline --name <PIPELINE_NAME>

**Note: Deleting a pipeline does not delete other AWS resources that have been provisioned or utilized. Such as CodeDeploy Applications, EC2 instances, or S3 buckets.*



Amazon Web Services

CodePipeline: Managing a Pipeline from the AWS CLI

Thank you for watching!

Amazon Web Services

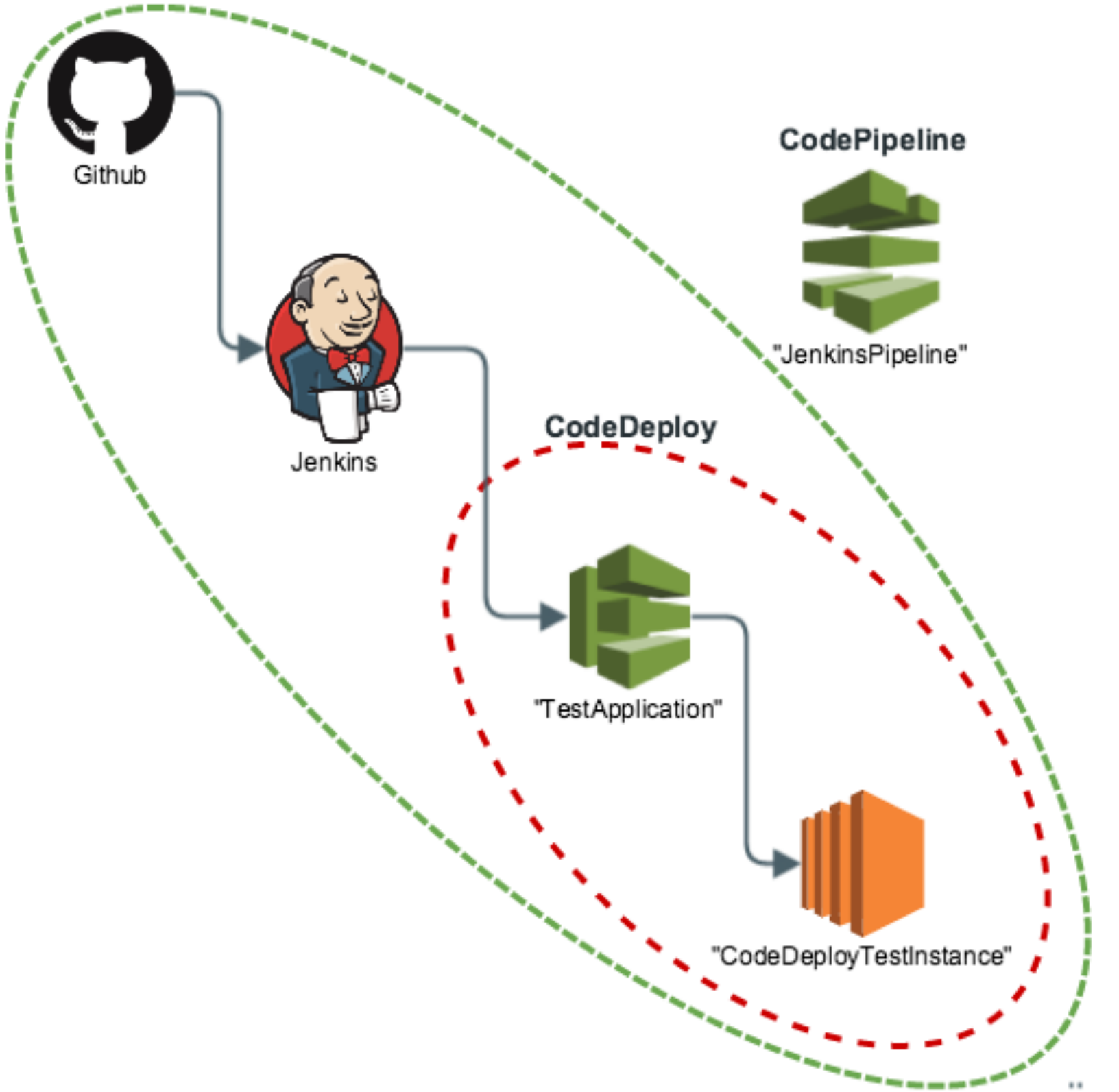
CodePipeline: Configuring a Pipeline with GitHub (source) & Jenkins (build)

Our Second Pipeline:

- This pipeline is going to use a code sample provided by AWS.
- The purpose of the Build Stage (Jenkins) in this example is to convert Haml code to HTML code.
- Once the pipeline has been created, configured, and run successfully, we will be able to access the target EC2 instance via a web browser and view a web page (the built and deployed HTML code).

- **Stage 1**
  - ***Name**: Source*
  - ***Provider**: Github*
  - ***Repository**: <account_name>/aws-codepipeline-jenkins-aws-codedeploy_linux*
  - ***Branch**: master*
- Stage 2
  - ***Name**: Build*
  - ***Provider**: Jenkins*
- Stage 3
  - ***Name**: Beta*
  - ***Provider**: CodeDeploy*
  - ***Application**: TestApplication*
  - ***Deployment Group**: TestDeploymentGroupName*

Our Second Pipeline:



Our Second Pipeline Setup & Configuration:

1) “Fork” the AWS GitHub Repository (containing the un-built source files) to our own GitHub Repository

2) Create an IAM role for the EC2 instance that Jenkins will be installed on

3) Launch a new EC2 Server

4) Install Jenkins on that EC2 server

5) Install Rake and HAML on that EC2 server

6) Configure Jenkins to access Rake on that EC2 server

7) Configure Jenkins on that EC2 server

8) Create a new Pipeline

9) Run Pipeline and view results

Step 1 - Getting the Source Files into GitHub:

1) Log into your GitHub account

2) Navigate to this URL:
   - ***Linux:*** *https://github.com/awslabs/aws-codepipeline-jenkins-aws-codedeploy_linux*

   - ***Windows:*** *https://github.com/awslabs/AWSCodePipeline-Jenkins-AWSCodeDeploy_windows*

3) Click on “Fork”
   - *This will copy this repository to your own GitHub account*

Step 2 - Creating an IAM Role for Jenkins:

1) Create a new role and attached the following policy to it: "***AWSCodePipelineCustomActionAccess***"

2) *Give the role a name that will make it obvious it is going to be used for a Jenkins server, like "JenkinsEC2Role"*

Linux Academy

Step 3 - Launch a new EC2 Instance:
Step 4 - Install Jenkins:
Step 5 - Install Rake & Haml:

1) Launch a new instance with the following settings:

- *AMI: Amazon Linux AMI*
- *Instance Type: t2.micro*
- *Set the IAM role to the "Jenkins" role you just created*
- *Under "Advanced Details" insert the following Bash Script:*

```
#! /bin/bash
yum update -y
wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
yum install jenkins -y
gem install rake
gem install haml
```

- *Default storage*
- *Give the instance a Name value*
- *Create a new Security Group, and have these ports open:*
  - *SSH (22)*
  - *HTTP (80)*
  - *Custom TCP (8080)*
- *Launch & select or create a key pair*

2) Wait for the instance to finish initializing before moving on

Step 6 - Configure Environment for Rake:

1) SSH into the EC2 server you just created

2) Switch to root user (command: **sudo su**)

3) Navigate to the directory **etc/init.d**

4) Open the file “jenkins” using nano (**nano jenkins**)

5) Insert the following (bottom) two lines of text after the (top) two lines of text in the “jenkins” file:

   # Source function library.
   . /etc/init.d/functions:

   # Custom Path
   export PATH="/usr/local/bin:/usr/bin:$PATH”

6) Save and exit

7) Restart Jenkins (run the command):

   service jenkins restart

Step 7 - Configure Jenkins:

1) Navigate to the directory /var/lib/jenkins/secrets

2) Cat the file “initialAdminPassword”

3) Record the password

4) Open a browser and navigate to the address:
   <EC2_IP_ADDRESS>:8080

5) When prompted, enter the password you just recorded from the Jenkins EC2 instance

6) Install suggested plugins

7) Create admin account credentials

8) Install Plugins (Manage Jenkins -> Manage Plugins -> Available) and search for “***AWS CodePipeline Plugin***” *and check the box next to the plugin*

9) Do the same but this time search for “Rake plugin”

Step 7 - Configure Jenkins (con't):

10) Click on "***Download now and install after restart***"

11) Click on "***Restart Jenkins when installation is complete and no jobs are running***". Jenkins will restart and install the plugins

12) Sign back into Jenkins

13) Dashboard -> Create new jobs

14) Enter a name for the Jenkins Project and select "***Freestyle project***", click "OK"

15) Under "***General***" check the box next to "***Execute concurrent builds if necessary***"

16) Under "***Source Code Management***", select "***AWS CodePipeline***"
    - Set "**AWS *Region***" to the region your CodePipeline S3 bucket and Pipeline are (or will be) located in
    - For "***Category***" select "***Build***"
    - Leave every other field blank or it's default entry

Step 7 - Configure Jenkins (con't):

17) Under "***Build Triggers***" check "***Poll SCM***"
    - In the "schedule" field, input "*** * * * ***" (that is five asterisks, each separated by a space)
    - This will have Jenkins poll AWS CodePipeline every minute

18) Under "***Build***" click on "***Add build step***"
    - Select "***Execute shell***"
    - In the "***Command***" field, type in "***rake***"

19) Under "***Post Build Actions***" click in "***Add post-build action***"
    - Select "***AWS CodePipeline Publisher***"
    - Under "***Output Locations***" click "***Add***"
    - Leave the "***Location***" field empty

20) Click "***Save***"

Step 8 - Create a new Pipeline:

1) Navigate CodePipeline in the AWS Console & click "***Create pipeline***"

2) Give the pipeline a name

3) Set the "Source Provider" to "GitHub"
   - Select "***Connect to Github***"
   - Authorize GitHub if prompted
   - For "***Repository***" select the AWS Repository we "forked" in step 1
   - For "***Branch***" select "***master***"

4) Set the "***Build provider***" to "***add Jenkins***"
   - "***Provider name***" must match what is in the "***Provider***" field in Jenkins ("Jenkins" is default)
   - For "***Server URL***" enter the IP Address of the EC2 server that Jenkins is installed on (http://IP_ADDRESS)
   - For "***Project name***" enter the name you gave the "***Project***" you just created in Jenkins

Step 8 - Create a new Pipeline (con't):

5) Set the "***Deployment provider***" to "***CodeDeploy***"
   - For "***Application name***" enter the name of the CodeDeploy Application you want to use
   - For "***Deployment group***" enter the name of the Deployment Group you want to use

6) Set the "***Role name***" to the AWS CodePipeline service role that you already have created (if you followed the earlier videos) OR choose to "Create Role" and follow the instructions

7) Review and Launch (click "***Create pipeline***")

Step 9 - Run & View Results:

1) Watch the Pipeline run
   - Hopefully there will not be any errors, and all stages will complete with a status of "***succeeded***"

2) Verify that everything has worked successfully:
   - In a web browser, navigate to:
     <EC2_IP_ADDRESS>/index.html

Linux Academy

Amazon Web Services

CodePipeline: Configuring a Pipeline with GitHub (source) & Jenkins (build)

Thank you for watching!

Amazon Web Services

CodePipeline: Approval Actions

Approval Actions:

- Approval actions allow for a pause in the pipeline, and require a human to manually “approve” before the pipeline will proceed to the next stage or action.

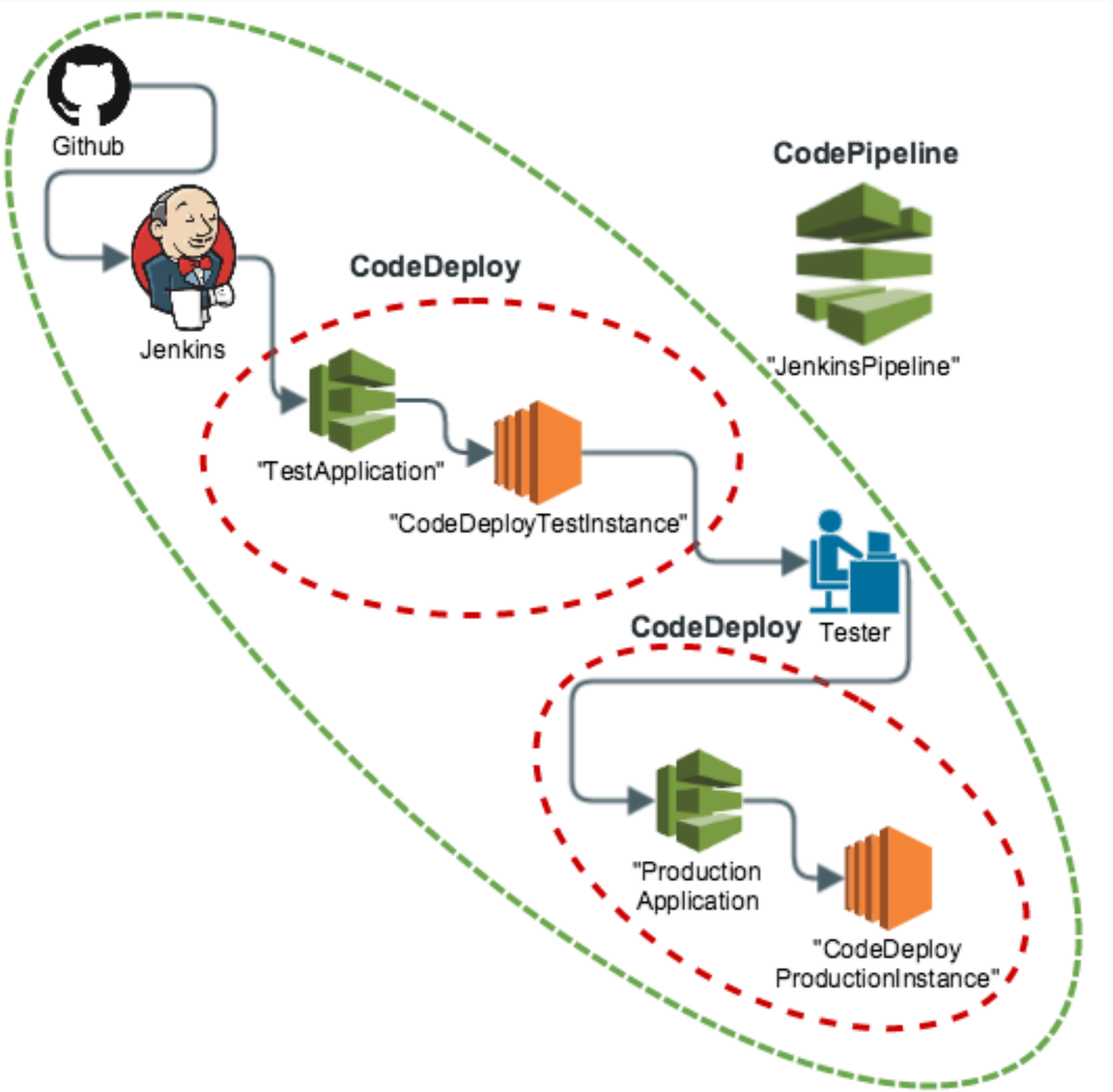
- Approval actions are generally used when you want a human to:
  - *Perform a code review or change management review*
  - *Test the application/perform quality assurance*

- *Approval actions can be implemented during any stage of a pipeline.*

- *However, they are generally added after the built application has been deployed onto test servers, and before the built application is deployed to production servers. (Beta -> Approval -> Production)*

- *Approval actions utilize AWS SNS Topics to notify the target human tester that whatever needs to be reviewed is ready for review*

Approval Action (5-stage) Pipeline:



Approval Action Prerequisites:

1) Permissions:
   - Make sure whoever is going to review and either “approve” or “disapprove” the approval action has the proper permissions to do so.

2) SNS Topic:
   - Have an SNS topic setup with the person who will be doing to review as a subscriber.

Adding an Approval Action:

1) Navigate to CodePipeline and select the pipeline you want to add the Approval action to.

2) Click “edit” to open the pipeline editing framework.

3) Add a new “stage” and give it a name

4) Add a new “action”:
   - For “***Action category***” select “***Approval***”
   - Give the action a name
   - For “***Approval type***” select “***Manual approval***”
   - For “***SNS topic ARN***” select the SNS Topic that your reviewer is subscribed to
   - For “***URL for review***” (optional) insert the location where the reviewer must to navigate to in order to review the code or application
   - For “***Comments***” (optional) provide a comment to the reviewer or leave it blank

5) Click “***Add action***”

Linux Academy

Amazon Web Services

CodePipeline: Approval Actions

Thank you for watching!

Amazon Web Services

CodePipeline: Creating a Custom Action

Custom Actions:

- Custom actions allow you to include activities in your pipeline which are outside the scope of the default set of six actions provided by AWS.

- Such actions may include utilizing an:
  - *Internally developed build process*
  - *Internally developed test software*

- Custom actions are generally defined into a:
  - *Build action*
  - *Deploy action*
  - *Test action*
  - *Invoke action*

- Custom actions must be accompanied with a "***job worker***"

Custom Actions:

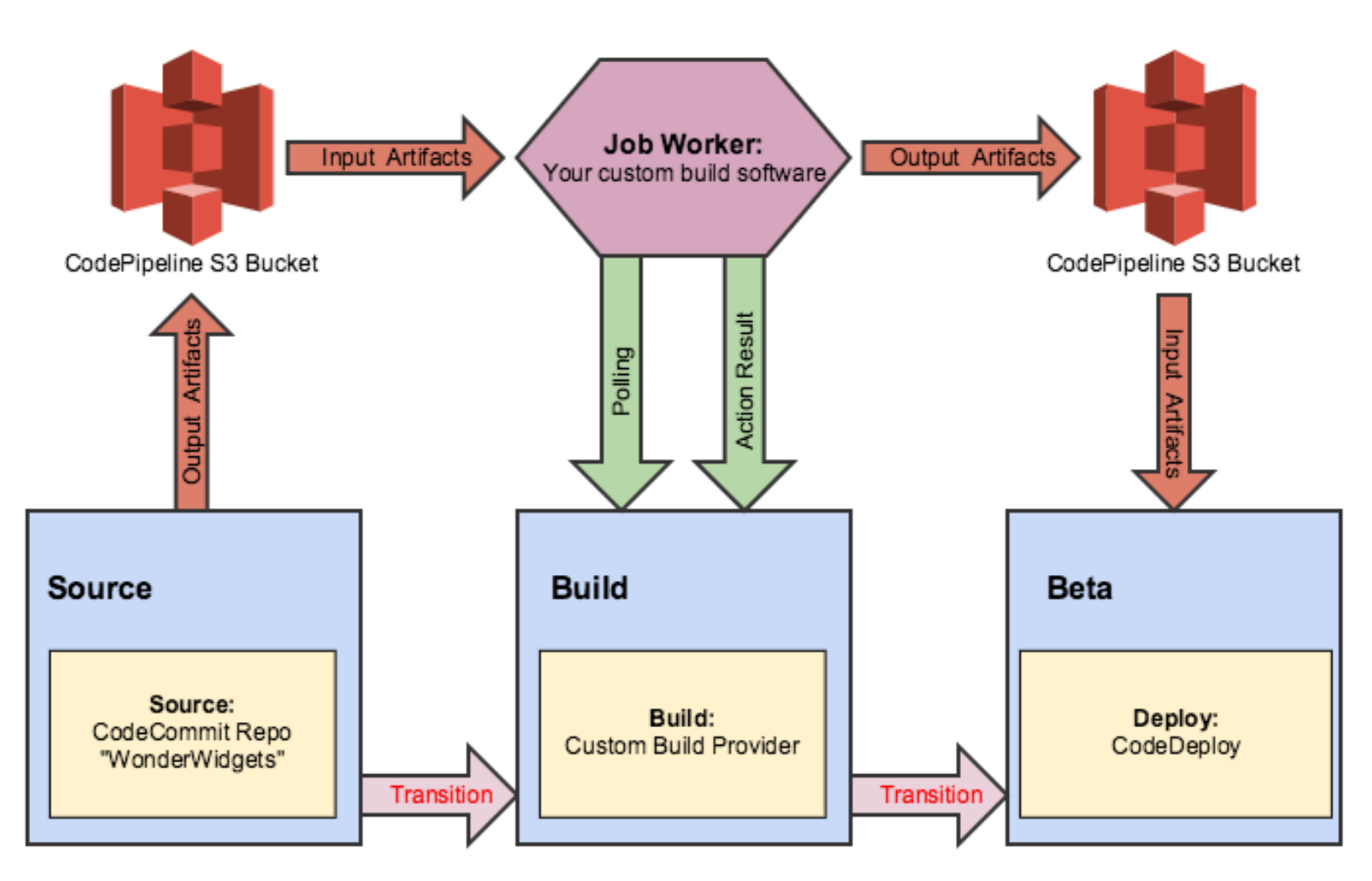
- A "***job worker"*** is the custom software that will be the *provider* for your custom action.

- The "***job worker***" is responsible for:
  - *Polling CodePipeline for "**job request**" (for the custom action)*
  - *Executing the job*
  - *Retuning the execution status to CodePipeline*

- "***Job request***" are created automatically by CodePipeline when a custom action is added to a pipeline.

- When a "***job worker***" detects a "***job request***" (through polling) it must be configured to complete the following:
  - *Acknowledge (to CodePipeline) that it has detected the job request*
  - *Pull the target (input) artifacts from the pipeline's S3 bucket*
  - *Execute the custom job (perform actions on the artifacts)*
  - *Push the (output) artifacts back to the pipeline's S3 bucket*
  - *Return the job results to CodePipeline (succeeded or failed)*

Custom Actions:



Creating a Custom Action & Job Worker:

- A custom action can only be created via the AWS CLI.
- A custom action, once created, can be added to a pipeline via either the AWS CLI or AWS Console.
- The job worker must be created, configured, and provided by you.

Creating a Custom Action (CLI):

1) Create a .json file, open it with nano, and paste in the contents of the "***custom action***" template.

2) Edit the JSON formatted "***custom action***" template.
   - This template is a sample provided by AWS and can be edited (and added to) to meet you needs.
   - **"category"**: Configure this field to one of the six categories provided by AWS (Approve, Build, Source, Invoke, Deploy, or Test).
   - "**provider**": This will be the display name under "build provider".
   - "**entityUrlTemplate**": This is a static link that when clicked on will take the user to a web page containing information about the build project (or test environment, etc).
   - "**executionUrlTemplate**": This is a dynamic link that when clicked will take the user to web page containing information specific to the more current run of the action.

Creating a Custom Action (CLI):

- **“name”**: This will be displayed as the prompt in the “Action Configuration” section when adding this action.
- “**description**”: This will be displayed under the entry field for “ProjectName” in the “Action Configuration” section.
- “**inputArtifactDetails**” & “**outputArtifactDetails**”: Setting the min/max number for each will set the size limit for the number of artifacts that can be used by the action.

3) Save and exit the file

4) Create the custom action by uploading the .json file to AWS (by using the command):

aws codepipeline create-custom-action-type --cli-input-json file://<FILE_NAME>.json

Creating a Job Worker (concepts):

- When a "***job worker***" detects a "***job request***" (through polling) it must be configured to complete the following:
  - *Acknowledge (to CodePipeline) that it has detected the job request*
  - *Pull the target (input) artifacts from the configured S3 bucket*
  - *Execute the custom job (perform actions on the artifacts)*
  - *Push the (output) artifacts back to the S3 bucket*
  - *Return the job results to CodePipeline (succeeded or failed)*

1) Configure Permissions for the job worker:
   - EC2 Instance with an IAM Instance role (easiest option)
   - Identity Federation (Active Directory, Amazon Cognito)

2) Configure AWS API calls:
   - For polling: ***PollForJobs***
   - *For job acknowledgement:* ***AcknowledgeJob***
   - For updates & succeeded status: ***PutJobSuccessResult***
     - *With continuation token allows you to update the action in the AWS console to include the* ***ExternalExecutionID*** *that will populate the link in "****executionUrlTemplate****"*
   - For failure: ***PutJobFailureResult:***

Creating a Job Worker (concepts):

3) Configure an Amazon S3 Client to retrieve and put artifacts
   - To ***download*** pipeline artifacts from S3, you must create a S3 Client that uses "***signature version 4 signing***" (Sig V4).

   - To ***upload*** pipeline artifacts to S3, you must configure the S3 ***PutObject*** request to use encryption.
   - Currently only SSE-KMS is supported for encryption.

Linux Academy

Amazon Web Services

CodePipeline: Creating a Custom Action

Thank you for watching!


Linux Academy
Amazon Web Services
AWS Developer Tools Course:
Recap & Next Steps
Congratulations &
Thank you for watching!