How do you implement environment promotion (Dev → Staging → Prod) using CodePipeline?

Goal:
Move the same artifact across multiple environments (Dev → Staging → Prod) in a controlled, secure, and repeatable way — without rebuilding.

Option 1: Single Multi-Stage
[Source (GitHub/CodeCommit)]
↓
[Build (CodeBuild)]
↓
[Deploy to Dev (CodeDeploy/ECS/Lambda)]
↓
[Manual Approval]
↓
[Deploy to Staging]
↓
[Manual Approval]
↓
[Deploy to Prod]

🧱 **Key Implementation Components**
1. Single Build → Reusable Artifact

- Only build once in Dev (via CodeBuild).
- Use that same artifact in all environments to ensure what was tested in Dev is what gets promoted to Staging/Prod.
- Set artifacts in buildspec.yml or pipeline config:

yaml
artifacts:
files:
- '/*'**
name: myAppArtifact
2. Separate Deploy Actions per Environment

- Define 3 separate deployment actions (Dev, Staging, Prod).
- Each uses the same output artifact from the Build stage.

3. Manual Approval Actions

- Place a ManualApproval action between:
  - Dev → Staging
  - Staging → Prod
- This allows for QA verification, business validation, or security reviews.

4. Environment-Specific Configurations

- Use parameterized configurations via:
  - Environment variables in CodeBuild
  - env.json files in S3 per environment
  - AWS SSM Parameter Store or Secrets Manager
- Pass configs dynamically using build or deploy hooks/scripts.

5. Separate IAM Roles per Environment

- Create IAM roles like:
  - CodeDeployServiceRole-Dev
  - CodeDeployServiceRole-Staging
  - CodeDeployServiceRole-Prod
- Grant least privilege access only to the relevant resources for that environment.

Option 2: Multiple Pipelines with Cross-Promotion
Use one pipeline per environment:
Pipeline 1: Dev → Build & Deploy
Pipeline 2: Staging ← Triggered via S3 artifact
Pipeline 3: Prod ← Triggered via Manual Approval or EventBridge

- Dev pipeline builds artifact and stores in S3.
- Staging and Prod pipelines pull same artifact using S3SourceAction.

Use this if:

- You want full isolation between environments.
- You need independent teams or schedules for Staging/Prod.

A CodePipeline deployment fails in production but works in staging — how do you debug?

- Check for **environment differences**: IAM roles, VPC, secrets, instance types.
- Review **CloudWatch Logs** and pipeline execution history.
- Compare **config files, env vars, deployment targets**.
- Validate **network access**, **resource limits**, or misconfigured parameters.
- Use **rollback and test in a cloned production environment**.

---

1. What are some common pre-built plugins available in CodePipeline?

*"AWS CodePipeline comes with a rich set of **pre-integrated actions** that cover the entire DevOps lifecycle. I regularly use:*

- **AWS CodeCommit / GitHub / Bitbucket** → For source control
- **AWS CodeBuild** → For compiling, unit testing, and packaging
- **AWS CodeDeploy / ECS / Lambda** → For deployments
- **Amazon S3** → For storing build artifacts
- **Manual Approval actions** → To gate production releases

- **CloudFormation** → For provisioning infrastructure as part of the pipeline

These plugins make it super easy to compose powerful CI/CD pipelines without writing custom glue code."

---

2. How can CodePipeline be adapted to support custom plugins or actions?

*"To support **custom actions**, I use the CodePipeline **Custom Action Integration** framework. Here's how I approached it:*

- First, I **registered a custom action type** using the CLI:

bash
aws codepipeline create-custom-action-type \
  --category Test --provider MySecurityScan \
  --version 1 --inputArtifactDetails maximumCount=1,minimumCount=1 \
  --outputArtifactDetails maximumCount=0,minimumCount=0

- Then, I implemented a webhook or used an **AWS Lambda** to run that custom logic.
- This allows me to plug in tools like **internal security scanners** or a custom compliance step before production.

The key is defining your input/output artifact schema and ensuring your custom tool can poll or react via webhook."

✅ **3. Explain how CodePipeline supports integration with custom build or deployment systems.**

*In one of my projects, we had a **legacy deployment tool** running on-prem. Instead of rewriting everything, I used a **Lambda function** inside a CodePipeline stage to trigger a webhook exposed by the legacy system.*

Here's an example snippet:

python
import requests

def lambda_handler(event, context):
    response = requests.post("https://legacy-deployer.local/deploy", json=event)
    return {"statusCode": 200, "body": "Triggered legacy deployment"}

This Lambda was placed after the build stage, and it effectively bridged AWS with our custom deployment environment."

. How would you customize a pipeline to integrate third-party tools like SonarQube or Snyk?

✅ **How It Works: buildspec.yml**

📝 1. Basic Structure of buildspec.yml

buildspec.yml is a YAML file that tells AWS CodeBuild what to do at each phase of the **build.**

Here's a sample with Snyk integration:

yaml
version: 0.2

phases:
  install:
    runtime-versions:

```
    nodejs: 18
  commands:
    - echo "Installing Snyk CLI"
    - npm install -g snyk

build:
  commands:
    - echo "Running Snyk vulnerability scan"
    - snyk test --severity-threshold=medium
```

🔍 What This Does:

- Install Phase: Installs Node.js and the Snyk CLI globally.
- Build Phase: Runs the Snyk test.
  - The --severity-threshold=medium flag tells Snyk to fail the build if it finds any vulnerabilities of medium severity or higher.

❗ If any critical/medium/major vulnerabilities are found, CodeBuild fails, and CodePipeline stops execution.

🔐 2. Using SonarQube with CodeBuild

SonarQube is used for static code analysis and checking code smells, bugs, coverage, and maintainability.

✅ **Steps to Integrate:**

a. Install SonarScanner CLI

In buildspec.yml:

yaml

```
phases:
  install:
    commands:
      - echo "Installing SonarQube Scanner CLI"
      - wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-5.0.1.3006-linux.zip
      - unzip sonar-scanner-cli-*.zip
      - export PATH=$PATH:$(pwd)/sonar-scanner-*/bin
```

b. Run Sonar Scan

You need the SonarQube projectKey, organization, and authentication token.

Store auth token securely in AWS Secrets Manager, and retrieve it in buildspec.yml:

yaml

```
  pre_build:
    commands:
      - echo "Retrieving SonarQube token from Secrets Manager"
      - export SONAR_TOKEN=$(aws secretsmanager get-secret-value --secret-id sonar-token --query SecretString --output text)
      - echo "SONAR_TOKEN=$SONAR_TOKEN" > sonar-project.properties
```

c. Define sonar-project.properties

You can also include this file directly in your repo or create it dynamically:

bash

```
echo "sonar.projectKey=myproject" >> sonar-project.properties
echo "sonar.host.url=https://sonarqube.myorg.com" >> sonar-project.properties
echo "sonar.login=$SONAR_TOKEN" >> sonar-project.properties
```

Then in build phase:

yaml

```
build:
  commands:
    - echo "Running SonarQube analysis"
    - sonar-scanner
```

Security Considerations

Concern	Best Practice
Token security	Use AWS Secrets Manager or SSM Parameter Store to store tokens securely
Build failure on scan	Let snyk test or sonar-scanner exit with a **non-zero code to halt pipeline**
Logs	Hide tokens from logs by not echoing secrets or masking with ***

Benefits of Integrating in CodeBuild

Benefit	Description
Shift-Left Security	Catches issues before deployment
CI-Driven Quality	Automatically checks every commit
Fail Fast	Fails pipeline immediately if critical issues found
Secret Management	Keeps sensitive tokens out of source code

✅ 6. Strategies to implement canary or blue/green deployments in CodePipeline

_"I use **CodeDeploy** with ECS or EC2-based apps for these advanced deployment strategies.

- For **Blue/Green**, CodeDeploy handles traffic shifting via **Application Load Balancer**.
- For **Canary**, I use deployment configs like:

```
json
"deploymentConfigName": "CodeDeployDefault.ECSCanary10Percent5Minutes"
```

This ensures 10% traffic goes to the new version initially, and only after 5 minutes of health check success, full traffic is routed."

✅ 7. How do you handle rollback or error handling in a multi-stage CodePipeline?

_"Error handling is built into CodePipeline:

- For **automated rollback**, I use **CodeDeploy** hooks and lifecycle events like BeforeInstall or AfterInstall. If something fails, rollback is triggered automatically.
- I also configure **SNS notifications + CloudWatch alarms** for custom rollback logic.
- On top of that, I use onFailure conditions to prevent further stages if a test or build fails."_

Declarative vs Imperative in CodePipeline

_"In **declarative** pipelines, you define the final state (e.g., JSON/YAML describing all stages). AWS CodePipeline follows this style. It's predictable, version-controlled, and reusable.

Imperative approaches, like scripting with Bash or Jenkins Groovy, focus on “how” to get things done step-by-step.

*I prefer declarative for CI/CD infrastructure because it's **cleaner, testable, and can be templatized easily with CloudFormation**."*

How do you manage version control and updates of pipeline configurations at scale?
_"I treat pipeline definitions as **infrastructure-as-code** using CloudFormation or CDK. For example, we stored the pipeline.json files in Git:

```
json
{
  "stages": [
    { "name": "Source", ... },
    { "name": "Build", ... }
  ]
}
```

*Any updates were done via Pull Requests. We used a **centralized repo with templates**, and each service could inherit or override pipeline logic. This ensured standardization and easy mass updates across 30+ services."*

Q: What is the goal of environment promotion in CodePipeline?
A. Build for each environment separately
B. Use different code versions per environment
C. Build once and reuse the artifact across all environments
D. Skip staging to save time

C. Build once and reuse the artifact across all environments ✅

Q: Why do we use separate IAM roles per environment?
A. Easier to delete pipelines
B. Faster builds
C. Enforce least privilege per environment
D. To avoid manual approvals

C. Enforce least privilege per environment ✅
