

@devopschallengehub

in

FOLLOW US ON
LINKEDIN

YouTube
SUBSCRIBE



WhatsApp Channel

Why CI/CD and Explain High Level AWS CI CD flow



1. Faster Delivery

- Automates the build, test, and deployment process.
- Reduces manual errors and speeds up time-to-market.
- Enables quick feedback and faster release cycles.



2. Early Bug Detection

- Code is tested with every commit (Continuous Integration).
- Problems are caught early in development, not in production.



3. Better Code Quality

- Encourages frequent, smaller commits and automated testing.
- Makes code more modular and easier to review.



4. Automation & Consistency

- Ensures consistent builds and deployments across environments.
- Eliminates “it works on my machine” problems.



5. Improved Collaboration

- Developers merge their code frequently.
- Encourages better teamwork and reduces integration conflicts.



6. Rapid Iteration & Feedback

- CI/CD enables continuous feedback from stakeholders and users.
- Quickly roll out features, A/B tests, or bug fixes.



7. Cloud & DevOps Friendly

- CI/CD is core to DevOps practices and cloud-native development.
- Scales with microservices, containers, and infrastructure as code.



Scenario 1: Simple Web App on EC2 (No Docker)



Flow:

[GitHub Push] → triggers → [CodePipeline]

└─ Source Stage: Pulls code from GitHub

- └─ Build Stage (optional): Run lint/tests using CodeBuild
- └─ Deploy Stage: CodeDeploy copies code to EC2 and executes scripts

🔧 **CodeBuild Add-ons:**

- Run **unit tests**
- Perform **static code analysis** (e.g., Flake8, ESLint)
- Send Slack/email notifications
- Archive artifacts (zip folder)

✅ **Scenario 2: Frontend + Backend, Dockerized, EC2 Deploy**

🔄 **Flow:**

mathematica

[GitHub Push] → [CodePipeline]

- └─ Source: GitHub
- └─ Build (CodeBuild):
 - └─ Build Docker images
 - └─ Run tests, scan code/images
 - └─ Optional: Push to ECR
- └─ Deploy (CodeDeploy EC2):
 - └─ SSH to EC2
 - └─ Run docker-compose up/down

🔧 **CodeBuild Add-ons:**

- Build/push multiple services (frontend/backend)
- Run **unit tests** (Jest, Pytest, etc.)
- Perform **Docker image vulnerability scanning** (e.g., Trivy)
- Linting (ESLint, Stylelint, etc.)
- Pre-deployment integration testing
- Update env vars/secrets from AWS SSM or Secrets Manager

✅ **Scenario 3: Frontend + Backend, Dockerized, ECS Deploy**

🔄 **Flow:**

mathematica

[GitHub Push] → [CodePipeline]

- └─ Source: GitHub
- └─ Build (CodeBuild):
 - └─ Build & push Docker images to ECR
 - └─ Scan Docker images (e.g., Trivy)
 - └─ Update ECS task definition file
- └─ Deploy (CodeDeploy ECS) or CLI:
 - └─ ECS service update via task definition

🔧 **CodeBuild Add-ons:**

- **Security scan** for container images
- Update ECS JSON task definitions dynamically
- Add environment validation steps
- Generate SBOMs (Software Bill of Materials)

✅ Scenario 4: Frontend + Backend, Dockerized, EKS Deploy

🔄 Flow:

vbnet

[GitHub Push] → [CodePipeline]

- |— Source: GitHub
- |— Build (CodeBuild):
 - |— Build & push Docker images to ECR
 - |— Update K8s manifests with new image tag
 - |— Apply changes using kubectl to EKS
 - |— Run unit tests, scan code/images
- |— Deploy: (Embedded in Build step via kubectl)

🔧 CodeBuild Add-ons:

- kubectl apply using kubeconfig/IRSA
- **Load testing** via tools like Locust or Artillery
- Validate manifests using tools like kubeval or kube-linter
- CI-quality gate integration (Snyk, Checkov)

✅ Scenario 5: Frontend + Backend, Dockerized, GitOps via ArgoCD

🔄 Flow:

pgsql

[GitHub Push to Source Repo] → [CodePipeline]

- |— Source: GitHub (App repo)
- |— Build (CodeBuild):
 - |— Build & push Docker images to ECR
 - |— Update image tag in GitOps repo (manifests)
 - |— Git commit & push to GitOps repo (infra repo)
- |— Deploy: Triggered automatically by ArgoCD
 - ArgoCD syncs updated GitOps repo and deploys to EKS

🔧 CodeBuild Add-ons:

- Image/tag promotion between environments (dev → staging → prod)
- Push manifest diff logs to Slack
- Run dry-run kubectl diff to preview changes
- Use cosign or notary to sign images before pushing

📦 Summary: Triggers and Connections Overview

Stage	Triggered By	Tool	Purpose/Action
Source	GitHub Push	CodePipeline	Pull latest code
Build	Source change	CodeBuild	Build app, Dockerize, run tests, scan, push
Deploy	Post-build	CodeDeploy / ArgoCD / CLI	Deploy to EC2/ECS/EKS

✓ Additional Things You Can Do in CodeBuild (Beyond Just Build)

Functionality	Description
✓ Unit Tests	Run tests using frameworks like Pytest, Mocha, Jest
✓ Static Code Analysis	Flake8, ESLint, SonarQube CLI
✓ Security Scans	Trivy (Docker), Snyk, Checkov, Bandit
✓ Docker Signing	Cosign, Notary for supply chain security
✓ Infra Linting	kube-linter, kubeval, tfsec
✓ Pre/Post Deploy Tests	Integration/load tests using curl, Locust, Artillery
✓ Update Git Repos	Commit manifest/image tag updates to GitOps repo
✓ Slack/Email Alerts	Notify on success/failure using webhook or SNS
✓ Tagging & Versioning	Auto-bump versions using Git tags or timestamps

Q1. In a simple EC2 deployment without Docker, what is the primary role of AWS CodeDeploy?

- A. Build Docker images
- B. Deploy code to ECS containers
- C. SSH into EC2 and launch containers
- D. Copy code to EC2 and execute deployment scripts

D. Copy code to EC2 and execute deployment scripts ✓

Q2. Which of the following is an optional stage in this CodePipeline setup?

- A. Source
- B. Build
- C. Deploy
- D. Notification

B. Build ✓

Q4. In this setup(Dockerized app), what command is typically used during the deploy stage on EC2?

- A. kubectl apply
- B. aws ecs update-service
- C. npm run start
- D. docker-compose up

D. docker-compose up ✓

Scenario Frontend + Backend, Dockerized, ECS Deploy

What is the deployment method used in this scenario?

- A. SCP to EC2
- B. ECS service update via task definition
- C. Jenkins pipeline
- D. Docker run on EC2 manually

B. ECS service update via task definition ✓

How is deployment typically triggered in an EKS pipeline?

- A. Via CodeDeploy agent on EC2
- B. Using kubectl apply within CodeBuild**
- C. Lambda function
- D. GitHub Actions

B. Using kubectl apply within CodeBuild ✓