# 1️⃣ What are CodeBuild, CodeDeploy, CodePipeline (in ONE line each)

### AWS CodeBuild

# 👉 *Builds your code*

# Compiles code

# Runs tests

# Creates artifacts (JAR, WAR, ZIP, Docker image)

# 🧠 Think: CI build server (like Jenkins build stage)

### AWS CodeDeploy

# 👉 *Deploys your code*

# Pushes code to EC2 / ECS / Lambda

# Handles rolling, blue-green deployments

# Manages deployment scripts

# 🧠 Think: Safe deployment engine

### AWS CodePipeline

# 👉 *Connects everything*

# Orchestrates steps: Source → Build → Test → Deploy

# Fully automated CI/CD pipeline

# 🧠 Think: Manager that runs the whole pipeline

## 🔁 Relationship (VERY IMPORTANT)

# CodePipeline

# ├── CodeBuild (build & test)

# └── CodeDeploy (deploy)

# Pipeline = Boss Build = Worker Deploy = Delivery guy

# 2️⃣ What is AWS Elastic Beanstalk?

### Elastic Beanstalk (EB)

# 👉 *Platform-as-a-Service (PaaS)*

# Upload code

# AWS handles:

# EC2

# Load balancer

# Auto scaling

# Deployments

# Minimal DevOps control

# 🧠 Think: “Just run my app” service

# 3️⃣ Code\* services vs Elastic Beanstalk (REAL comparison)

|  |  |  |
| --- | --- | --- |
| Feature | CodePipeline + Build + Deploy | Elastic Beanstalk |
| Control | 🔥 Full control | ⚠️ Limited |
| Learning DevOps | ✅ Yes | ❌ No |
| Custom pipelines | ✅ Yes | ❌ Limited |
| Production usage | ✅ Very common | ⚠️ Medium |
| Scaling control | ✅ Full | ⚠️ Abstracted |
| Best for | DevOps engineers | Beginners / quick apps |

# 👉 Industry truth

# DevOps roles → CodePipeline stack

# Startups / demos → Elastic Beanstalk

# 4️⃣ ONE PRACTICAL INDUSTRY USE CASE (Step-by-step)

### 🎯 Scenario:

# Company deploys a web app automatically when developer pushes code

## 🧱 Architecture (REAL industry setup)

# Developer

# ↓ git push

# GitHub

# ↓

# CodePipeline

# ↓

# CodeBuild (tests + build)

# ↓

# CodeDeploy

# ↓

# EC2 (Production)

## 🔧 What each service does (real action)

### 1️⃣ Developer pushes code to GitHub

# New feature / bug fix

### 2️⃣ CodePipeline detects change

# Starts pipeline automatically

# No human involvement

### 3️⃣ CodeBuild runs

# Installs dependencies

# Runs unit tests

# Builds application

# Creates artifact (ZIP)

# Example:

# npm install

# npm test

# npm run build

### 4️⃣ CodeDeploy takes over

# Deploys app to EC2

# Uses deployment strategy:

# Rolling

# Blue-Green

# Example:

# AfterInstall:

# - location: scripts/start\_app.sh

### 5️⃣ App is live 🚀

# Zero downtime

# Logs in CloudWatch

## 🔥 THIS is real DevOps

# No Elastic Beanstalk here.

# 5️⃣ Where Elastic Beanstalk fits (honest truth)

### EB Use Case

# College project

# POC (Proof of Concept)

# Simple app demo

# Example:

# Upload ZIP → App runs

# ⚠️ But:

# You don’t learn infra deeply

# Limited customization

# Not preferred for large-scale DevOps teams

# 6️⃣ When companies use BOTH (important)

# Some teams do:

# Elastic Beanstalk + CodePipeline

# Pipeline builds → EB deploys

# But: 👉 Advanced teams eventually move away from EB

# 7️⃣ How YOU should learn (perfect path)

# Since you want industry understanding:

# 1️⃣ Learn CodeBuild (buildspec.yml) 2️⃣ Learn CodeDeploy (apps + deployment groups) 3️⃣ Learn CodePipeline (end-to-end flow) 4️⃣ Then try Elastic Beanstalk once (to compare)

# 8️⃣ ONE HANDS-ON LAB IDEA (resume ready)

### 🧪 Lab: CI/CD for Node.js App

# Source: GitHub

# Build: CodeBuild

# Deploy: CodeDeploy

# Target: EC2

# Logs: CloudWatch

# 👉 This alone screams “DevOps-ready”

## 🧠 Interview Golden Line (remember this)

# *Elastic Beanstalk abstracts infrastructure, while CodePipeline with CodeBuild and CodeDeploy gives full CI/CD control suitable for production DevOps workflows.*

# **Week 2: AWS / Linux / DevOps Project**

### **Project 1: AWS CI/CD Pipeline (Full Guidance)**

**Project Name:** NodeJS-CICD-Pipeline

**Folder Structure:**

app/

├── index.js

├── package.json

├── buildspec.yml

├── appspec.yml

└── scripts/

├── install.sh

└── start.sh

# **PART 2️⃣ — COMPANY SCENARIO (REALISTIC)**

### **🏢 Company: TechNova Pvt Ltd**

* Product: Internal Web App
* Users: Employees
* Requirement:  
  + Automatic deployment
  + Zero manual work
  + Safe releases
  + Logs & monitoring

### **👤 You are the DevOps Engineer**

Your job:

“Whenever developer pushes code → app should update automatically on server”

# **PART 3️⃣ — SERVICES WE WILL USE (FINAL LIST)**

### **🔧 Core DevOps Services**

* **AWS CodePipeline** → CI/CD orchestration
* **AWS CodeBuild** → Build & test
* **AWS CodeDeploy** → Deploy to EC2

### **🧱 Infrastructure**

* **EC2 (t2.micro)** → App server
* **IAM** → Permissions
* **S3** → Store build artifacts

### **📊 Monitoring**

* **CloudWatch** → Logs & status

✅ All free-tier friendly.

# **PART 4️⃣ — FINAL ARCHITECTURE (INDUSTRY FLOW)**

Developer

↓ (git push)

GitHub

↓

CodePipeline

↓

CodeBuild

↓

S3 (artifact)

↓

CodeDeploy

↓

EC2 (App running)

↓

CloudWatch (logs)

This is **REAL DevOps**, not demo stuff.

# **PART 5️⃣ — STEP-BY-STEP (NO SKIPPING)**

## **STEP 1️⃣ — Application (Simple but realistic)**

We’ll use a **Node.js app** (industry common).

📁 App structure:

app/

├── index.js

├── package.json

├── buildspec.yml

├── appspec.yml

└── scripts/

├── install.sh

└── start.sh

## **STEP 2️⃣ — EC2 Setup (Server)**

1. Launch **EC2 (Amazon Linux 2, t2.micro)**
2. Enable:  
   * HTTP (80)
   * SSH (22)
3. Install Code-Deploy agent:

sudo yum update -y

sudo yum install ruby wget -y

wget https://aws-codedeploy-<region>.s3.amazonaws.com/latest/install

chmod +x install

sudo ./install auto

1. Install Node.js:

sudo yum install nodejs -y

## **STEP 3️⃣ — IAM Roles (CRITICAL)**

### **Create 3 roles:**

### **🔐 EC2 Role**

* Permissions:  
  + AmazonS3ReadOnlyAccess
  + CloudWatchLogsFullAccess

### **🔐 CodeBuild Role**

* Permissions:  
  + AmazonS3FullAccess
  + CloudWatchLogsFullAccess

### **🔐 CodeDeploy Role**

* Permissions:  
  + AWSCodeDeployRole

👉 **Industry rule:** least privilege, no root usage.

## **STEP 4️⃣ — CodeBuild (CI)**

### **buildspec.yml**

version: 0.2

phases:

install:

runtime-versions:

nodejs: 18

build:

commands:

- npm install

- npm test || echo "No tests"

artifacts:

files:

- '\*\*/\*'

What this does:

* Installs dependencies
* Runs tests
* Creates build artifact

## **STEP 5️⃣ — CodeDeploy (CD)**

### **appspec.yml**

version: 0.0

os: linux

files:

- source: /

destination: /home/ec2-user/app

hooks:

AfterInstall:

- location: scripts/start.sh

timeout: 300

### **scripts/start.sh**

#!/bin/bash

cd /home/ec2-user/app

npm install

node index.js > app.log 2>&1 &

👉 This is **how real deployments happen**.

## **STEP 6️⃣ — CodeDeploy Setup**

1. Create **Application**
2. Create **Deployment Group**
3. Choose:  
   * EC2
   * Tag-based deployment
4. Attach IAM role

## **STEP 7️⃣ — CodePipeline (Glue)**

Pipeline stages:

1. **Source** → GitHub
2. **Build** → CodeBuild
3. **Deploy** → CodeDeploy

Trigger:

* On every GitHub push

🎯 Result:

One git push = full deployment

# **PART 6️⃣ — HOW THIS LOOKS IN REAL COMPANY**

### **Daily workflow:**

* Developer pushes code
* Pipeline runs automatically
* Build logs in CloudWatch
* Deployment status visible
* App updates with zero SSH

### **Manager sees:**

* Faster releases
* No manual errors
* Audit logs

# **PART 7️⃣ — WHY COMPANIES LOVE THIS SETUP**

✅ Scalable  
 ✅ Secure  
 ✅ Auditable  
 ✅ Automatable  
 ✅ DevOps best practices

👉 This is **100% industry-grade**, not tutorial junk.

# **PART 8️⃣ — INTERVIEW-LEVEL EXPLANATION (SAVE THIS)**

*We used CodePipeline for orchestration, CodeBuild for CI, and CodeDeploy for controlled deployments to EC2. The setup ensures automated, repeatable, and safe releases with monitoring via CloudWatch, all within AWS Free Tier constraints.*

========================================================================

# **STEP 1️⃣ — Launch EC2 (Free Tier)**

* AMI: **Amazon Linux 2**
* Instance: **t2.micro**
* Storage: default (8 GB)
* Security Group:  
  + SSH (22) → Your IP
  + HTTP (80) → Anywhere
* IAM Role: *(attach later if not yet created)*

SSH into EC2:

ssh -i yourkey.pem ec2-user@<EC2\_PUBLIC\_IP>

# **STEP 2️⃣ — Prepare EC2 Environment**

Run these **once**:

sudo yum update -y

sudo yum install -y nodejs git

node -v

npm -v

👉 If Node isn’t installed:

curl -fsSL https://rpm.nodesource.com/setup\_18.x | sudo bash -

sudo yum install -y nodejs

# **STEP 3️⃣ — Create Project Structure (Exactly as you wrote)**

mkdir app

cd app

mkdir scripts

touch index.js package.json buildspec.yml appspec.yml

touch scripts/install.sh scripts/start.sh

Verify:

tree

# **STEP 4️⃣ — Paste CODE (VERY IMPORTANT PART)**

## **📄 index.js**

const http = require('http');

const PORT = 3000;

const server = http.createServer((req, res) => {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('Hello from CI/CD DevOps App 🚀\n');

});

server.listen(PORT, () => {

console.log(`App running on port ${PORT}`);

});

## **📄 package.json**

{

"name": "cicd-devops-app",

"version": "1.0.0",

"description": "Simple Node.js app for AWS CI/CD",

"main": "index.js",

"scripts": {

"start": "node index.js",

"test": "echo \"No tests yet\""

},

"dependencies": {}

}

## **📄 buildspec.yml (CodeBuild will use this, NOT EC2)**

version: 0.2

phases:

install:

runtime-versions:

nodejs: 18

build:

commands:

- echo "Installing dependencies"

- npm install

- echo "Running tests"

- npm test

artifacts:

files:

- '\*\*/\*'

## **📄 appspec.yml (CodeDeploy uses this)**

version: 0.0

os: linux

files:

- source: /

destination: /home/ec2-user/app

hooks:

AfterInstall:

- location: scripts/install.sh

timeout: 300

ApplicationStart:

- location: scripts/start.sh

timeout: 300

## **📄 scripts/install.sh**

#!/bin/bash

cd /home/ec2-user/app

npm install

## **📄 scripts/start.sh**

#!/bin/bash

cd /home/ec2-user/app

# Stop old app if running

pkill node || true

# Start new app

nohup node index.js > app.log 2>&1 &

# **STEP 5️⃣ — Permissions (CRITICAL, OFTEN MISSED)**

chmod +x scripts/install.sh

chmod +x scripts/start.sh

# **STEP 6️⃣ — Manual Test (Confidence Boost 💪)**

Run:

npm install

node index.js

In another terminal:

curl http://localhost:3000

Output:

Hello from CI/CD DevOps App 🚀

✅ Your app works.

Stop it:

CTRL + C

# **STEP 7️⃣ — IMPORTANT REALITY CHECK 🚨**

### **❌ What you will NOT do in real CI/CD:**

* SSH and run npm install
* Start app manually
* Edit files on EC2

### **✅ What WILL happen later:**

* Code lives in **GitHub**
* CodeBuild runs buildspec.yml
* CodeDeploy:  
  + Copies files to EC2
  + Runs install.sh
  + Runs start.sh
* EC2 becomes **immutable**

👉 EC2 = dumb server  
 👉 Pipeline = brain

# **STEP 8️⃣ — Why this step was still useful**

Now you:  
 ✔ Understand file roles  
 ✔ Know what CodeBuild executes  
 ✔ Know what CodeDeploy executes  
 ✔ Can debug failures confidently

This is **how industry DevOps engineers learn**.

========================================================================  
**🧠 FULL INDUSTRY EXPLANATION (VERY CLEAR)**

Let’s separate **WHAT lives WHERE**.

## **1️⃣ What lives in GitHub (SOURCE OF TRUTH)**

Your GitHub repo contains:

app/

├── index.js

├── package.json 👈 dependency LIST

├── buildspec.yml

├── appspec.yml

└── scripts/

├── install.sh

└── start.sh

❌ NOT in GitHub:

* node\_modules/
* Installed packages
* Running processes

📌 **package.json = instructions** 📌 **node\_modules = result**

## **2️⃣ Where dependencies are installed (2 PLACES)**

### **🔹 A) CodeBuild (BUILD TIME)**

📍 Happens in **CodeBuild container**

In buildspec.yml:

build:

commands:

- npm install

✅ What happens:

* CodeBuild downloads dependencies
* Runs tests
* Packages app as artifact
* Uploads to S3

📌 This is **CI (Continuous Integration)**

### **🔹 B) EC2 via CodeDeploy (RUNTIME)**

📍 Happens **on EC2**, automatically

In appspec.yml:

hooks:

AfterInstall:

- location: scripts/install.sh

scripts/install.sh:

npm install

✅ What happens:

* CodeDeploy copies files to EC2
* Runs install script
* Dependencies installed fresh on EC2

📌 This is **CD (Continuous Deployment)**

## **3️⃣ Why install dependencies TWICE? (Industry logic)**

This confuses everyone at first.

### **🧠 Reason:**

* **Build-time** → tests, linting, packaging
* **Run-time** → actual app execution

💡 In real companies:

* Sometimes **only build installs deps**
* Sometimes **only runtime installs deps**
* Sometimes **both**

For learning → **both is PERFECT**

## **4️⃣ What about node\_modules? (VERY IMPORTANT)**

### **❌ Do NOT commit this:**

node\_modules/

Add .gitignore:

node\_modules/

📌 Industry rule:

Rebuild dependencies every deploy → predictable → safe

## **5️⃣ Exact FLOW (End-to-End)**

### **1️⃣ You push code to GitHub**

git push origin main

### **2️⃣ CodePipeline triggers**

### **3️⃣ CodeBuild**

* Reads package.json
* Runs npm install
* Runs tests
* Creates artifact

### **4️⃣ CodeDeploy**

* Downloads artifact from S3
* Copies files to EC2
* Runs:  
  + install.sh → npm install
  + start.sh → node index.js

### **5️⃣ App runs 🎉**

NO SSH.  
 NO manual installs.

## **6️⃣ What YOU should do now (VERY IMPORTANT)**

Since you asked *“where to install dependencies”* — here’s your **exact action**:

### **✅ On EC2:**

* Install **Node.js only** (once)
* Install **CodeDeploy agent**

### **❌ On EC2:**

* Do NOT run npm install manually
* Do NOT start app manually

## **7️⃣ Interview-Perfect Explanation (MEMORIZE)**

*Dependencies are defined in package.json and installed automatically during the CI/CD process. CodeBuild installs them during build for testing, and CodeDeploy installs them on the EC2 instance at runtime using deployment hooks. The EC2 server remains immutable and is not manually modified.*

## **8️⃣ Next Logical Step (tell me one)**

1️⃣ Create .gitignore + push repo to GitHub  
 2️⃣ Set up **CodeBuild project** (console + IAM)  
 3️⃣ Install **CodeDeploy agent** on EC2  
 4️⃣ Wire everything using **CodePipeline**

You’re asking the **right DevOps questions** — keep going 💪

okay now i ahve my questions and answers now from the beginning every thing clear and step by step like i am a beginner guide me and lets do one project and remmember to give me git hub project a good title

Perfect 👍  
 From here on, I’ll treat this like **Day-1 onboarding of a junior DevOps engineer** in a company.  
 Slow, clean, **no assumptions**, and **one complete project** from zero → working CI/CD.

I’ll also **lock a GitHub project title** so you feel this is real.

# **🏷️ GitHub Project Title (FINAL)**

**aws-cicd-nodejs-devops-pipeline**

📌 Why this title is good:

* Mentions **AWS**
* Mentions **CI/CD**
* Mentions **DevOps**
* Mentions **Node.js** 👉 Recruiter + interviewer friendly

# **📘 PROJECT OVERVIEW (Beginner Friendly)**

### **🎯 What you will build**

A **simple Node.js web app** that:

* Automatically deploys to EC2
* On every GitHub push
* Using AWS CI/CD services

### **🧠 What you will learn**

* What each AWS DevOps service does
* Where code lives
* Where dependencies are installed
* How real companies deploy apps

# **🧱 SERVICES WE WILL USE (ALL FREE TIER SAFE)**

|  |  |
| --- | --- |
| **Purpose** | **AWS Service** |
| Source code | GitHub |
| CI/CD Orchestration | CodePipeline |
| Build & Test | CodeBuild |
| Deployment | CodeDeploy |
| Server | EC2 (t2.micro) |
| Artifacts | S3 |
| Access control | IAM |
| Logs | CloudWatch |

❌ No Load Balancer  
 ❌ No Elastic Beanstalk  
 ✅ Full DevOps-style setup

# **🏗️ FINAL ARCHITECTURE (KEEP THIS IN MIND)**

GitHub

↓

CodePipeline

↓

CodeBuild

↓

S3

↓

CodeDeploy

↓

EC2

# **🧑‍💻 STEP 0 — Mindset (IMPORTANT)**

Before touching AWS:

✅ Code lives in **GitHub** ✅ EC2 is **NOT for coding** ✅ EC2 is **only for running the app** ✅ CI/CD tools do the work

# **STEP 1️⃣ — Create GitHub Repository**

1. Go to GitHub
2. New repository

Name:  
  
 aws-cicd-nodejs-devops-pipeline

1. Public
2. Add README
3. Create repo

# **STEP 2️⃣ — Project Structure (THIS IS STANDARD)**

On your **local machine**:

aws-cicd-nodejs-devops-pipeline/

├── index.js

├── package.json

├── buildspec.yml

├── appspec.yml

├── .gitignore

└── scripts/

├── install.sh

└── start.sh

👉 This structure is **industry-acceptable**

# **STEP 3️⃣ — Application Code (VERY SIMPLE)**

## **index.js**

const http = require('http');

const PORT = 3000;

http.createServer((req, res) => {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('Hello from AWS CI/CD DevOps Project 🚀\n');

}).listen(PORT);

console.log(`Server running on port ${PORT}`);

## **package.json**

{

"name": "aws-cicd-nodejs-app",

"version": "1.0.0",

"description": "Node.js app deployed using AWS CI/CD",

"main": "index.js",

"scripts": {

"start": "node index.js",

"test": "echo \"No tests yet\""

}

}

## **.gitignore**

node\_modules/

# **STEP 4️⃣ — CodeBuild Configuration (CI)**

## **buildspec.yml**

version: 0.2

phases:

install:

runtime-versions:

nodejs: 18

build:

commands:

- echo "Installing dependencies"

- npm install

- echo "Running tests"

- npm test

artifacts:

files:

- '\*\*/\*'

🧠 Meaning:

* CodeBuild installs dependencies
* Runs tests
* Prepares files for deployment

# **STEP 5️⃣ — CodeDeploy Configuration (CD)**

## **appspec.yml**

version: 0.0

os: linux

files:

- source: /

destination: /home/ec2-user/app

hooks:

AfterInstall:

- location: scripts/install.sh

timeout: 300

ApplicationStart:

- location: scripts/start.sh

timeout: 300

## **scripts/install.sh**

#!/bin/bash

cd /home/ec2-user/app

npm install

## **scripts/start.sh**

#!/bin/bash

cd /home/ec2-user/app

pkill node || true

nohup node index.js > app.log 2>&1 &

Make executable:

chmod +x scripts/\*.sh

# **STEP 6️⃣ — Push Code to GitHub**

git init

git add .

git commit -m "Initial AWS CI/CD DevOps project"

git branch -M main

git remote add origin <your-repo-url>

git push -u origin main

✅ Code source ready

# **STEP 7️⃣ — Launch EC2 (Free Tier)**

* AMI: Amazon Linux 2
* Instance: t2.micro
* Security Group:  
  + SSH 22 (your IP)
  + HTTP 80 (0.0.0.0/0)

SSH into EC2:

sudo yum update -y

sudo yum install -y nodejs ruby wget

# **STEP 8️⃣ — Install CodeDeploy Agent (MANDATORY)**

cd /home/ec2-user

wget https://aws-codedeploy-<region>.s3.amazonaws.com/latest/install

chmod +x install

sudo ./install auto

sudo systemctl start codedeploy-agent

sudo systemctl status codedeploy-agent

# **STEP 9️⃣ — IAM Roles (SIMPLE VERSION)**

### **EC2 Role**

* AmazonS3ReadOnlyAccess
* CloudWatchLogsFullAccess

### **CodeBuild Role**

* AmazonS3FullAccess
* CloudWatchLogsFullAccess

### **CodeDeploy Role**

* AWSCodeDeployRole

Attach roles properly.

# **STEP 🔟 — Create AWS Services (ORDER MATTERS)**

1️⃣ Create **CodeDeploy Application** 2️⃣ Create **Deployment Group** (EC2 tags)  
 3️⃣ Create **CodeBuild Project** 4️⃣ Create **CodePipeline**

* Source: GitHub
* Build: CodeBuild
* Deploy: CodeDeploy

# **🎉 FINAL RESULT**

You push code →  
 Pipeline runs →  
 Build happens →  
 App deploys →  
 EC2 runs updated app automatically

Check:

curl http://<EC2\_PUBLIC\_IP>:3000

# **🧠 WHAT YOU JUST LEARNED (BIG DEAL)**

* Real CI/CD flow
* Industry DevOps tools
* Zero manual deployment
* AWS Free Tier safe project

========================================================================

# **✅ SECTION 1 — AWS CONSOLE CLICKS (EXACT UI STEPS)**

## **🔐 STEP 1: IAM ROLES (Do this FIRST)**

### **1️⃣ EC2 Role**

AWS Console → **IAM → Roles → Create role**

* Trusted entity: **AWS service**
* Use case: **EC2**
* Permissions:  
  + AmazonS3ReadOnlyAccess
  + CloudWatchLogsFullAccess
* Role name:

EC2-CodeDeploy-Role

Attach this role to your EC2 instance.

### **2️⃣ CodeDeploy Role**

IAM → Roles → Create role

* AWS service
* Use case: **CodeDeploy**
* Permissions:  
  + AWSCodeDeployRole
* Role name:

CodeDeploy-Service-Role

### **3️⃣ CodeBuild Role**

IAM → Roles → Create role

* AWS service
* Use case: **CodeBuild**
* Permissions:  
  + AmazonS3FullAccess
  + CloudWatchLogsFullAccess
* Role name:

CodeBuild-Service-Role

## **🖥️ STEP 2: EC2 SETUP (TARGET SERVER)**

EC2 → Launch instance

* AMI: **Amazon Linux 2**
* Instance type: **t2.micro**
* IAM role: EC2-CodeDeploy-Role
* Security Group:  
  + SSH (22) → Your IP
  + HTTP (80) → 0.0.0.0/0

SSH & install:

sudo yum update -y

sudo yum install -y nodejs ruby wget

### **Install CodeDeploy Agent**

cd /home/ec2-user

wget https://aws-codedeploy-<region>.s3.amazonaws.com/latest/install

chmod +x install

sudo ./install auto

sudo systemctl start codedeploy-agent

## **🚀 STEP 3: CodeDeploy**

### **Create Application**

CodeDeploy → Applications → Create application

* Name:

NodeJS-CICD-App

* Compute platform: **EC2/On-Premises**

### **Create Deployment Group**

* Deployment group name:

NodeJS-EC2-DeploymentGroup

* Service role: CodeDeploy-Service-Role
* Environment:  
  + EC2 instances
  + Tag-based (use your EC2 tag)
* Deployment type:  
  + **In-place**

## **🧪 STEP 4: CodeBuild**

CodeBuild → Create build project

* Project name:

NodeJS-CICD-Build

* Source: **GitHub**
* Environment:  
  + Managed image
  + Amazon Linux
  + Runtime: Node.js 18
* Service role:  
  + Existing role → CodeBuild-Service-Role
* Buildspec:  
  + **Use buildspec.yml from repo**

## **🔁 STEP 5: CodePipeline (FINAL GLUE)**

CodePipeline → Create pipeline

* Pipeline name:

NodeJS-CICD-Pipeline

* Service role: New role
* Source:  
  + GitHub
  + Repo: aws-cicd-nodejs-devops-pipeline
* Build:  
  + Provider: CodeBuild
  + Project: NodeJS-CICD-Build
* Deploy:  
  + Provider: CodeDeploy
  + App: NodeJS-CICD-App
  + Deployment group: NodeJS-EC2-DeploymentGroup

🎉 DONE  
 Now every git push = auto deployment.

# **✅ SECTION 2 — draw.io ARCHITECTURE (EXACT ICON NAMES)**

Open **draw.io → AWS Icons (2023 set)**

### **Use THESE icons (search exact names):**

1. **GitHub** (Simple Icons → GitHub)
2. **AWS CodePipeline**
3. **AWS CodeBuild**
4. **AWS CodeDeploy**
5. **Amazon S3**
6. **Amazon EC2**
7. **Amazon CloudWatch**

### **Layout (left → right)**

GitHub

↓

CodePipeline

↓

CodeBuild → S3

↓

CodeDeploy

↓

EC2

↓

CloudWatch

📌 Label arrows:

* “Source Trigger”
* “Build & Test”
* “Artifact Storage”
* “Automated Deployment”

# **✅ SECTION 3 — GitHub README.md (PROFESSIONAL)**

# AWS CI/CD Pipeline for Node.js Application

This project demonstrates a complete CI/CD pipeline using AWS DevOps services to automatically build, test, and deploy a Node.js application to an EC2 instance.

## 🚀 Technologies Used

- AWS CodePipeline

- AWS CodeBuild

- AWS CodeDeploy

- Amazon EC2 (Free Tier)

- Amazon S3

- AWS IAM

- Amazon CloudWatch

- GitHub

## 📐 Architecture

GitHub → CodePipeline → CodeBuild → S3 → CodeDeploy → EC2

## ⚙️ CI/CD Workflow

1. Developer pushes code to GitHub

2. CodePipeline triggers automatically

3. CodeBuild installs dependencies and runs tests

4. Build artifacts are stored in S3

5. CodeDeploy deploys the application to EC2

6. Application starts automatically with zero manual intervention

## 🎯 Key Learnings

- End-to-end CI/CD automation

- Infrastructure as Code principles

- Secure IAM role-based access

- Production-style deployment workflow

## 🧪 How to Test

```bash

curl http://<EC2\_PUBLIC\_IP>:3000

## **📌 Author**

Abdullah

---

# ✅ SECTION 4 — RESUME PROJECT (COPY-PASTE READY)

### AWS CI/CD DevOps Project

- Designed and implemented an end-to-end CI/CD pipeline using AWS CodePipeline, CodeBuild, and CodeDeploy

- Automated build, test, and deployment of a Node.js application to EC2

- Used IAM roles for secure, least-privilege access control

- Implemented artifact storage using Amazon S3

- Enabled monitoring and logs through Amazon CloudWatch

- Built using AWS Free Tier services following industry DevOps practices

---

# ✅ SECTION 5 — MEDIUM + BLOGGER ARTICLE (FULL ARTICLE)

### Title:

\*\*Building a Real-World CI/CD Pipeline on AWS Using CodePipeline, CodeBuild, and CodeDeploy\*\*

### Article Content:

```md

## Introduction

Modern software teams rely on CI/CD pipelines to deliver applications faster and more reliably. In this article, I demonstrate how to build a real-world CI/CD pipeline on AWS using native DevOps services.

## Problem Statement

Manual deployments are error-prone, slow, and not scalable. The goal was to automate the entire process from code commit to production deployment.

## Architecture Overview

The pipeline uses GitHub as the source repository, AWS CodePipeline for orchestration, CodeBuild for continuous integration, and CodeDeploy for automated deployments to EC2.

## Implementation

- CodeBuild installs dependencies and runs tests

- CodeDeploy handles in-place deployment on EC2

- CloudWatch provides logs and visibility

## Benefits

- Fully automated deployments

- Zero manual intervention

- Secure and scalable setup

- Free Tier friendly

## Conclusion

This project helped me understand how real DevOps teams implement CI/CD pipelines using AWS. The same pattern can be extended for microservices, containers, and production systems.

👉 Post **same content** on:

* **Medium**
* **Blogger**

# **🧠 FINAL TRUTH (IMPORTANT)**

What you built is:

* ✅ Industry-grade
* ✅ Interview-ready
* ✅ Resume-strong
* ✅ Beginner → Advanced bridge

Most people **never go this deep**.  
 You did.

## **NEXT LEVEL (when ready)**

* Add **Blue/Green deployment**
* Add **Docker**
* Add **Terraform**
* Add **Monitoring alarms**

Just say *“next”* and we keep going 🚀

and finally a linkedin post i will when do this project attach the screen shot with it just create a post with curious mind and practcal problem solving person

Nice — this is the **right finishing touch** 👌  
 I’ll write it so you sound **curious, hands-on, and problem-solving**, not braggy or cringe.  
 You’ll attach screenshots of the pipeline / EC2 output with it.

### **🔹 LinkedIn Post (Copy–Paste Ready)**

🚀 **From “how does deployment actually work?” to a real AWS CI/CD pipeline**

For a long time, CI/CD felt like a buzzword to me. I understood the theory, but I wanted to see **what really happens when code moves from GitHub to a running server**.

So I decided to build it properly.

🔹 Created a **Node.js application** 🔹 Set up **AWS CodePipeline** to orchestrate the flow  
 🔹 Used **CodeBuild** to install dependencies and run builds  
 🔹 Used **CodeDeploy** to deploy the app automatically to **EC2** 🔹 Implemented **IAM roles** for secure, least-privilege access  
 🔹 Monitored everything using **CloudWatch**

Now, every git push triggers an automated build and deployment — no manual SSH, no guesswork.

This project taught me that DevOps is less about tools and more about **solving practical problems**:

* How do teams reduce deployment errors?
* How do you make releases repeatable?
* How do you remove manual steps safely?

Still learning, still breaking things, still fixing them — but this hands-on approach is helping me understand how real production systems work.

📌 Tech used: AWS CodePipeline | CodeBuild | CodeDeploy | EC2 | S3 | IAM

#DevOps #AWS #CICD #CloudComputing #LearningByDoing #HandsOn #ProblemSolving

### **💡 Posting tips (important)**

* Attach **2–3 screenshots**:  
  + CodePipeline success screen
  + CodeDeploy deployment success
  + App running on EC2 (curl or browser)
* Post during **evening (7–10 PM local)** for better reach
* Don’t over-edit — this sounds **authentic**, which recruiters like

========================================================================

### **🔹 LinkedIn Post (Copy–Paste Ready)**

🚀 **From “how does deployment actually work?” to a real AWS CI/CD pipeline**

For a long time, CI/CD felt like a buzzword to me. I understood the theory, but I wanted to see **what really happens when code moves from GitHub to a running server**.

So I decided to build it properly.

🔹 Created a **Node.js application** 🔹 Set up **AWS CodePipeline** to orchestrate the flow  
 🔹 Used **CodeBuild** to install dependencies and run builds  
 🔹 Used **CodeDeploy** to deploy the app automatically to **EC2** 🔹 Implemented **IAM roles** for secure, least-privilege access  
 🔹 Monitored everything using **CloudWatch**

Now, every git push triggers an automated build and deployment — no manual SSH, no guesswork.

This project taught me that DevOps is less about tools and more about **solving practical problems**:

* How do teams reduce deployment errors?
* How do you make releases repeatable?
* How do you remove manual steps safely?

Still learning, still breaking things, still fixing them — but this hands-on approach is helping me understand how real production systems work.

📌 Tech used: AWS CodePipeline | CodeBuild | CodeDeploy | EC2 | S3 | IAM

#DevOps #AWS #CICD #CloudComputing #LearningByDoing #HandsOn #ProblemSolving

### **💡 Posting tips (important)**

* Attach **2–3 screenshots**:  
  + CodePipeline success screen
  + CodeDeploy deployment success
  + App running on EC2 (curl or browser)
* Post during **evening (7–10 PM local)** for better reach
* Don’t over-edit — this sounds **authentic**, which recruiters like

========================================================================

## **1️⃣ Is this project related to Linux & Bash?**

### **✅ YES — strongly related, and here’s *why* (this matters for LinkedIn):**

Even though the project uses AWS CI/CD, **everything actually runs on Linux**.

### **Linux + Bash used in this project:**

* **Amazon Linux 2** (production Linux OS)
* Bash scripts:  
  + install.sh
  + start.sh
* Linux commands:  
  + yum
  + systemctl
  + chmod
  + pkill
  + nohup
* Process management:  
  + Background services
* File permissions & ownership
* Logs (app.log, CloudWatch)

👉 **This is not a cloud-only project** 👉 It’s a **Linux automation project running in the cloud**

So yes — you can **confidently frame it as Linux + Bash + Automation**.

## **2️⃣ How you should POSITION this project (VERY IMPORTANT)**

### **❌ Don’t position it as:**

“I built an AWS CI/CD pipeline”

### **✅ Position it as:**

**“Automating Linux application deployment using Bash and CI/CD tools”**

That keeps you **Linux-first**, cloud second.

## **3️⃣ Your 2-Week LinkedIn Strategy (PERFECT for a beginner)**

You already have:

* 15,000+ impressions
* Momentum
* Beginner honesty (this is GOOD)

### **🎯 Goal:**

Stay **Linux-focused**, show **progress**, not perfection.

## **4️⃣ CONTENT FLOW YOU SHOULD FOLLOW**

### **🔹 Post Type 1 — Course Completion (Light, Honest)**

**Example framing:**

Completed my Linux fundamentals course and started applying the concepts in real labs.

Mention:

* File system
* Permissions
* Processes
* Bash basics

📌 Keep it short.

### **🔹 Post Type 2 — Lab Work (Hands-on proof)**

**Examples:**

* User & group management script
* Process monitoring script
* Log analyzer
* Backup automation

📌 These posts build **credibility**.

### **🔹 Post Type 3 — FINAL HANDS-ON PROJECT (BIG POST)**

👉 This is where **this project fits perfectly**.

Frame it like this:

I used Linux + Bash scripting to automate application deployment on a Linux server.

Then mention:

* EC2 (Linux server)
* Bash scripts
* Automation
* CI/CD tools as helpers

## **5️⃣ How THIS PROJECT CONNECTS ALL YOUR LEARNING**

You can honestly say:

“I combined Linux fundamentals, Bash scripting, and automation concepts to build a real deployment workflow.”

### **Skills you reinforce:**

* Linux administration
* Bash scripting
* Automation mindset
* DevOps basics

## **6️⃣ Suggested FINAL LINKEDIN POST (Linux-First Version)**

Use THIS when you finish the project 👇

🐧 **Learning Linux is not about commands — it’s about automation**

After completing my Linux fundamentals course, I wanted to apply what I learned in a practical way instead of stopping at theory.

So I built a **Linux-based deployment automation project**.

🔹 Used **Amazon Linux 2** as the server  
 🔹 Wrote **Bash scripts** to install dependencies and start applications  
 🔹 Managed Linux processes and permissions  
 🔹 Automated deployments so the server updates itself on every code change

This helped me understand how Linux is actually used in real environments — not just locally, but in production systems where automation matters.

I’m still a beginner, but building things and breaking them has been the fastest way for me to learn.

📌 Tools: Linux | Bash | Git | AWS (EC2, CI/CD)

#Linux #Bash #Automation #DevOps #LearningInPublic #HandsOn

## **7️⃣ Why this strategy WORKS**

* You don’t look fake
* You don’t look rushed
* You look **consistent**
* You look **teachable**
* You look **serious about Linux**

Recruiters LOVE this pattern.

## **8️⃣ My honest advice (important)**

👉 **Do Linux & Bash projects for 2 weeks** 👉 Then slowly add:

* Git
* CI/CD
* Docker

This project becomes your **bridge** from Linux → DevOps.

=======================================================================