Tell me about your experience with AWS CodeDeploy. What is it, and how have you used it?

In my last project, we relied heavily on **AWS CodeDeploy** to automate and standardize our deployment process across **EC2 instances**, **on-premises servers**, and **Lambda functions**. What I really appreciated is how it's **fully managed** — it saved us from writing custom scripts for orchestration.
It supports two main types of deployments, and I've worked with both:

- **In-place deployments**: The app is stopped on the same EC2 instance, updated, and restarted. It's cost-effective but can cause minor downtimes.
- **Blue/green deployments**: This one's a game-changer for production. We deploy the new version on a fresh environment, test it, and then shift traffic. It's ideal when you can't afford downtime.

How do you decide between in-place and **blue**/green?

We make that decision based on **risk tolerance and criticality**.

Feature	In-place	Blue/Green
Process	Stops and updates existing	Deploys to new env, then shifts traffic
Downtime	Possible	Near zero
Rollback	Manual/slow	Instant, just shift traffic back
Use Case	QA, staging, low-risk apps	Prod apps with high availability needs

For example, during a retail app launch, we used **blue/green deployments with Lambda** to avoid disruptions during peak sales.

What is experience deployment configurations? Any experience with Canary?

(**Canary deployment** is a software release strategy where a new version of an application is gradually **rolled out to a small subset of users before making it available to everyone.)**

Yes, I've used several **deployment configs**, especially with Lambda. One of my favorites is:

- CodeDeployDefault.**LambdaCanary10Percent5Minutes:** shifts 10% of traffic to the new version, waits 5 mins, then 100%. Perfect for catching issues early.

Other patterns we used:

- AllAtOnce: risky but fast.
- Linear10PercentEvery1Minute: good middle ground.

These configurations helped us **minimize risk** and still push updates confidently.

How does deployment differ for EC2 vs Lambda?

Here's what I've observed:

Feature	EC2 / On-Prem	Lambda
Target	Servers, VMs	Serverless Functions
Speed	Slower (stop/start app)	Fast (just shift traffic)
Traffic Control	Load balancer	Lambda aliases (canary/linear)
Rollback	Manual	Instant rollback via alias switch

In EC2, we use **AppSpec** files and lifecycle hooks. For Lambda, we just define traffic shifting configs and let AWS handle the rest.

Speaking of AppSpec, can you walk me through it?

The appspec.yml file is **central to EC2/On-Prem deployments**. It tells CodeDeploy:

- Which files to copy
- Which scripts to run and **when** to run them

Here's an example from our production app:

yaml

```
---
version: 0.0
os: linux
files:
  - source: /
    destination: /var/www/myapp

hooks:
  ApplicationStop:
    - location: scripts/application_stop.sh
  BeforeInstall:
    - location: scripts/before_install.sh
  Install:
    - location: scripts/install.sh
  AfterInstall:
    - location: scripts/after_install.sh
  ApplicationStart:
    - location: scripts/application_start.sh
  ValidateService:
    - location: scripts/validate_service.sh
```

Each hook handled specific steps — like stopping the app, installing dependencies, restarting the service, and running health checks.

How did you handle rollbacks if something failed?

We had both **manual and automatic rollback** strategies.

- For **critical pipelines**, we enabled **automatic rollback** on failure or failed health checks — CodeDeploy reverted to the last good revision automatically.
- For **manual rollback**, we redeployed the **last successful version** via **CLI** or the console.

This came in handy once when a bad config slipped into staging. The rollback kicked in and saved us from a production outage.

What about database migrations during deployments?

That's something we were very cautious about.
We usually ran DB migrations using the **AppSpec lifecycle hooks**, either in BeforeInstall or AfterInstall. For larger setups, we created a **dedicated CodeBuild stage** before CodeDeploy.
We ensured:

- **Locking mechanisms** were in place to prevent concurrent migrations
- Used tools like **Flyway** and **Liquibase**
- Had rollback scripts handy in case of failures

Any tips you'd give someone implementing CodeDeploy?

Yes:

- Always **test deployments in staging** using the same hooks and scripts
- Keep appspec.yml and scripts/ under version control
- Use **CloudWatch alarms** and **CodeDeploy events** for monitoring
- Enable **auto-rollback** for critical environments
- Practice deployments on **test EC2s or dummy Lambda functions** before going live

1. What are the two main types of deployments supported by AWS CodeDeploy?

A. Canary and Rolling
B. Blue/Green and In-Place
C. Immutable and Recreate
D. On-Demand and Scheduled

✅ **Answer:** B. Blue/Green and In-Place

---

2. Which deployment type is best for high-availability production environments with zero downtime?

A. In-place deployment
B. Rolling deployment
C. Blue/Green deployment
D. All-at-once deployment
✅ **Answer:** C. Blue/Green deployment

Which of the following deployment configurations gradually shifts 10% of traffic and waits 5 minutes before shifting the rest?

A. CodeDeployDefault.LambdaLinear10PercentEvery1Minute
B. AllAtOnce
C. CodeDeployDefault.LambdaCanary10Percent5Minutes
D. LinearAllAtOnce10MinDelay

✅ **Answer:** C. CodeDeployDefault.LambdaCanary10Percent5Minutes

---

What role does the AppSpec file play in an EC2 deployment via CodeDeploy?

A. Defines infrastructure
B. Contains IAM policies
C. Specifies deployment scripts and file locations
D. Sets up VPC networking

✅ **Answer:** C. Specifies deployment scripts and file locations

---

How did the team handle database migrations during deployment?

A. Manually using SSH
B. Using CodeBuild only
C. Using lifecycle hooks like BeforeInstall or AfterInstall
D. As part of the Lambda alias setup

✅ **Answer:** C. Using lifecycle hooks like BeforeInstall or AfterInstall

---