Question 5: How do you implement approval gates and manual interventions in CodePipeline?

In critical CI/CD workflows, especially when deploying to production or running sensitive operations like database changes or major releases, it's important to **introduce human oversight**. That's where AWS CodePipeline's **Manual Approval actions** come in handy.

What Are Manual Approvals in CodePipeline?

- A **Manual Approval** is a stage in CodePipeline that **pauses execution** and **waits for a human to approve or reject** the action before proceeding.
- It's a **built-in action type** under the **Approval category** in the pipeline.
- You can define:
    - Who gets notified (via **SNS**)
    - What conditions must be met (via **comments** or metadata)
    - Timeouts (pipeline fails if not approved in time)

Use Case: Finance Application Deployment

"We were working on a **financial transaction platform** where **accuracy and regulatory compliance** were top priorities."

Use Case Flow:

1. Developers push code → Pipeline runs unit/integration tests.
2. After successful deployment to **Staging**, the pipeline **paused**.
3. A **Manual Approval stage** was inserted **before Production deployment**.

Who Got Notified?

- An **SNS topic** was configured to send emails to the **Release Management team**.
- We also added **Slack integration** via webhook to notify instantly with the commit details and a direct link to approve.

Advanced Setup: Conditional Approvals

"Over time, we refined this process using **Lambda functions** for smart approvals based on context."

Automation vs Manual Rules:

Scenario	Approval Type	Reason
Minor UI tweak	**Auto-approved via Lambda** (during business hours)	Safe, reversible
DB Schema Change	**Manual Approval**	High-risk
Security Patch	**Manual Approval**	Needs validation
Feature Flag Toggle	**Auto** (if toggled on QA)	Controlled rollout
Major Release (v2.x)	**Manual Approval**	Involves infra + code change

Lambda Auto-Approval Logic:

- Time check: Only during **business hours**
- Change type: Based on tags/commit messages (e.g., #safe)
- Branch check: Only main or release/*

Best Practices We Followed

1. **Granular Approval Gates**
   - Added approval **before production**, but not for every environment.
   - Reduced bottlenecks while ensuring oversight where needed.
2. **Detailed Commit Metadata**
   - Slack messages included:
     - Git commit diff link
     - Author
     - JIRA ticket (if available)
3. **Timeout Management**
   - Approvals auto-failed after **2 hours** to avoid stale pipelines.
4. **Audit Trail**
   - All approvals/rejections logged in **CloudTrail** for compliance audits.

Takeaway

"Adding approval gates in CodePipeline gave us **flexibility + control**. For fast-moving deployments, we could automate approvals during low-risk periods, while still maintaining **strict manual review** for high-impact releases. It also helped align with both **security teams** and **release managers**."

Question 6: How do you handle rollbacks and failure scenarios in CodePipeline?

Handling failures proactively is one of the most critical aspects of DevOps. In my experience, especially during production deployments, we had to design pipelines that not only alert on failure but also support **fast and safe rollback mechanisms**.

What Are Rollbacks in CodePipeline?

Rollback means **restoring the application** to a known stable version when something goes wrong during or after deployment. In AWS CodePipeline, this can be handled:

Rollback Type	Trigger	Tools Used
Manual Rollback	Human-triggered	CodeDeploy, S3, ECR
Automated Rollback	Monitored via metrics	CloudWatch, Lambda, CodeDeploy

---

Real Use Case: Financial Web Application

🔧 **Stack Used:**

- EC2 for backend servers
- CodePipeline + CodeDeploy (with Blue/Green strategy)
- CloudWatch for error monitoring
- Lambda for alerting and rollback trigger

🚨 **Failure Scenario:**

We had a case where:

- A new release was deployed using **Blue/Green** strategy
- Within 2 minutes, **CloudWatch logged a spike** in 5xx errors
- Users started facing login issues

🔄 **Rollback Strategy We Implemented**

🧠 **1. Monitoring**

- CloudWatch monitored:
  - 5xx Error Rate
  - Latency
  - Custom app metrics from the backend (like login failure rate)

⚡ **2. Automatic Trigger**

- A **CloudWatch Alarm** was set with thresholds (e.g., >5% 5xx in 2 mins)
- The alarm triggered a **Lambda function**

🛠️ **3. Rollback via Lambda**

- The Lambda:
  - Identified the last healthy deployment revision
  - Called the **CodeDeploy rollback API** using AWS SDK
  - Notified DevOps team via **Slack and email** with rollback status

🔁 **4. Blue/Green Deployment Advantage**

- Since we used **Blue/Green**:
  - New version was deployed on a **separate set of instances**
  - Traffic could be shifted **back to the old "green" version** instantly

---

📉 **Outcome**

- **MTTR (Mean Time To Recovery)** dropped from **45 minutes to just 5 minutes**
- Less than 2 minutes of visible impact for end users
- Faster debugging using Lambda-generated rollback logs

💡 **Other Failure Handling Tips I Follow**

Practice	Why It Matters
Pipeline-level Error Handling	Add "catch" stages for Slack/email alerts

Practice	Why It Matters
Deploy with Approval Gates	Gives human buffer before prod
Tag Deployments in ECR/S3	Helps identify and rollback to stable versions
Retry logic in Lambda & alarms	Prevent false positives or noisy rollbacks
CloudWatch Dashboards	Real-time deployment health visualization

---

Takeaway
In critical production pipelines, **fast rollback = fast recovery**. By integrating CloudWatch, Lambda, and CodeDeploy together, we made our pipeline **smart, reactive, and safe**. We no longer feared failures — we embraced them as part of a resilient release process.

Question 7: How do you integrate third-party tools with CodePipeline?

Answer:

- **Prebuilt Plugins:**
  - Jenkins (plugin), GitHub (WebhookV2), Slack (Lambda)
- **Custom Integration (Lambda):**

javascript
```
exports.handler = async (event) => {
  const state = event.detail.state;
  const pipeline = event.detail.pipeline;
  if (state === 'SUCCEEDED') await sendSlackMessage(` ✅ ${pipeline} deployed successfully!`);
  else if (state === 'FAILED') await sendSlackMessage(` ❌ ${pipeline} failed. Check logs.`);
};
```

- **EC2 Custom Action:** Used CodePipeline agent for on-premise deployments
-

Which AWS service is typically used to notify stakeholders during a manual approval stage?
A. AWS CloudWatch
B. AWS CloudFormation
C. Amazon SNS
D. AWS Config

Correct Answer: C

What happens if a manual approval is not given within the timeout window?

A. The pipeline retries approval every 5 minutes
B. The pipeline proceeds automatically
C. The pipeline fails
D. An automatic rollback is triggered

Correct Answer: C

What is the benefit of using Lambda for auto-approvals in CodePipeline?

A. Manual oversight for all changes
B. Eliminate need for notifications
C. Enable dynamic approvals based on context (e.g., tags, time, branch)
D. Avoid timeout issues

Correct Answer: C