CI/CD Pipeline Video Walkthrough Script

# Slide 1: Title Slide

Hello and welcome. My name is Tatenda Manyepa, and this is part of my AWS Solutions Architect portfolio.  
In this presentation, I’ll walk you through a project called AWS CI/CD Pipeline with Security Best Practices: Modular and Reusable Architecture.  
This project demonstrates how to build a secure and efficient CI/CD pipeline using native AWS services, with reusability and flexibility at its core.

# Slide 2: Project Overview

The goal of this project was to implement a secure, automated CI/CD pipeline that ensures fast and reliable software delivery.  
I used AWS services such as CodePipeline, CodeBuild, and CodeDeploy, all secured through IAM and Secrets Manager.  
What makes this design powerful is its modularity, each part can be reused independently across different development workflows or adapted to suit a variety of deployment scenarios.

# Slide 3: Architecture Diagram

Here you can see the architecture diagram of the pipeline.  
The process begins when developers push code to GitHub. That commit event automatically triggers AWS CodePipeline, which coordinates the entire CI/CD process.  
CodeBuild pulls the source code, compiles it, runs tests, and stores the build artefacts in Amazon S3.  
Then, CodeDeploy takes over and handles deployment to targets such as EC2, ECS, or Lambda.  
Throughout this process, the architecture ensures security, efficiency, and high availability by leveraging native AWS tools and best practices.

# Slide 4: Architecture Flow Summary

This slide breaks down the flow of the pipeline:

* Code is pushed to GitHub, triggering the pipeline
* CodeBuild compiles and tests the application
* Artefacts are stored in S3
* CodeDeploy handles deployment to the target platform—whether EC2, ECS, or Lambda  
  Monitoring is handled through CloudWatch, CloudTrail, and AWS X-Ray, and scalability is built in via Auto Scaling Groups and AWS Fargate.

# Slide 5: Step-by-Step Implementation

To build this solution, I followed a secure and structured implementation plan:  
First, I configured IAM roles with least privilege access for each service.  
Then, I connected the source control repository, GitHub in this case, with CodePipeline.  
Next, I defined pipeline stages for build, test, and deploy.  
I used CodeBuild for secure, automated builds, and added automated testing to maintain code quality.  
Deployment was managed using CodeDeploy, and finally, I enabled real-time monitoring using CloudWatch and CloudTrail.

# Slide 6: Reusability and Integration

This pipeline is not limited to a single use case, it is highly reusable and integratable.  
It works across serverless deployments, containerised apps using ECS or EKS, and EC2-based environments.  
Real-world examples include:

* E-commerce platforms enabling fast deployments
* Financial services needing secure, compliant releases
* Healthcare systems deploying HIPAA-compliant updates
* SaaS products delivering features continuously

# Slide 7: Results and Impact

The impact of this solution is significant.  
It can reduce deployment time from hours to minutes, increase developer productivity, and improve software quality through automated testing.  
By using AWS-native services, you can improve scalability and security while accelerating the time-to-market for new features.

# Slide 8: Challenges and Solutions

Like any real-world project, there were a few challenges.  
For example, the CodeDeploy agent had connectivity issues, which I fixed by restarting the agent and reviewing IAM roles.  
I also encountered missing application files error, which I resolved by correcting the appspec.yml file and verifying artefact paths.  
SSH permission errors, missing Node.js installations, and deployment script failures were all resolved through debugging and configuration tuning.

# Slide 9: Pipeline Optimisation Techniques

To make the pipeline faster and more efficient, you can apply several optimisation techniques:

* Build caching in CodeBuild can be used to avoid reinstalling dependencies
* Combined commands can be used to reduce build steps
* Artefacts can be compressed and unnecessary files can be removed
* Docker containers can be used in CodeBuild for faster setup
* Parallel stages in CodePipeline can be enabled
* EC2 instances and pre-configured AMIs can be tuned for better performance  
  These steps collectively enhanced speed without compromising security.

# Slide 10: Conclusion

Thank you for taking the time to explore this project.  
This CI/CD pipeline architecture offers a secure and modular foundation for modern application delivery.  
It enables speed, security, and flexibility across a range of deployment models using AWS-native services.  
Feel free to visit cloudportfolio.co.uk to explore more of my work or connect with me directly.