MULESOFT ARCHITECTURE:

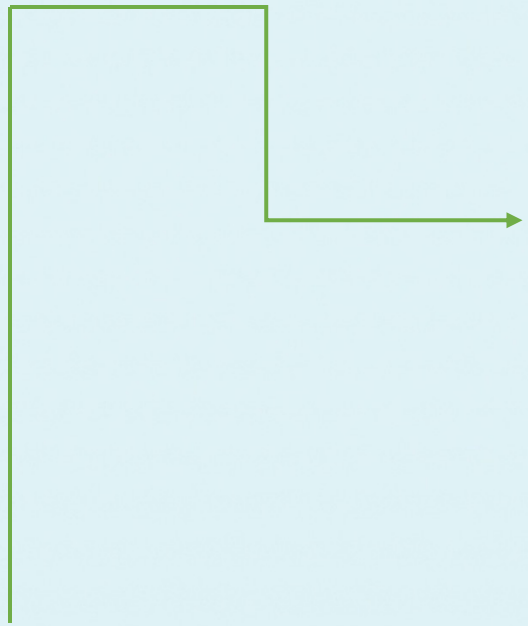
DevOps Part 2

PRACTIC LEARNING

DevOps Introduction

1. Automating deployments
2. CI/CD Pipeline Steps
3. Design Scenario

Automating Deployments



Systematic Deployments

- Maven is the key to systematic deployments
  - Use Maven commands from the command line to deploy
  - Reads configuration from pom.xml
  - Maven deployments use the Anypoint Platform APIs under the hood
  - Maven is the recommended way to automate deployments
    - Alternatively, can use the Anypoint Platform APIs, but would require custom code in order to deploy
- Authentication and authorization
  - 2 options
    - Connected app client ID and secret
    - Username and password of user (recommended to use service account if choosing this option)
  - Both methods allow for permissions governance in Anypoint Access Manager

Deploying to CloudHub

1. Set up authentication using a service account (connected apps or username and password)
2. Configure your pom.xml for cloudHubDeployment
3. Test locally using the mvn command line utility
4. Implement the mvn command in your CI/CD pipeline
    1. Ensure network connectivity is established from the CI/CD system to the internet for access to Anypoint Platform APIs

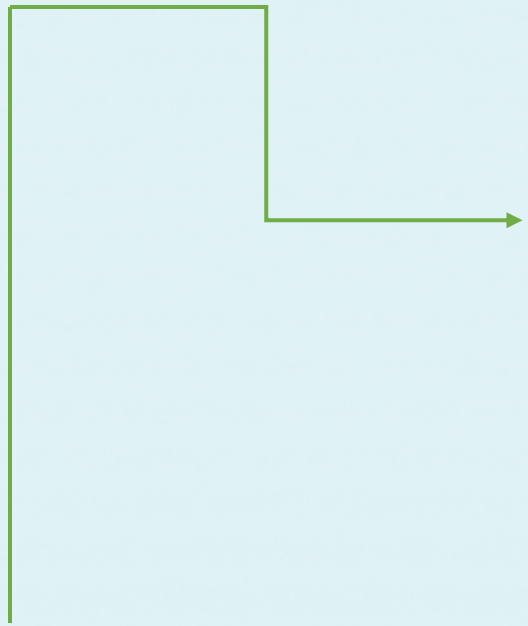
Deploying to Hybrid Environment

1. Set up authentication using a service account (connected apps or username and password)
2. Set up a server or cluster in Runtime Manager
3. Configure your pom.xml for armDeployment
4. Test locally using the mvn command line utility
5. Implement the mvn command in your CI/CD pipeline
    1. Ensure network connectivity is established from the CI/CD system to the internet for access to Anypoint Platform APIs

Technology Considerations

- There is not a single right tool to use for automating deployments using CI/CD
- Use the right tool for your organization
- High quality CI/CD tools include
  - AWS CodePipeline
  - Azure DevOps
  - Jenkins and CloudBees
  - CircleCI

CI/CD Pipeline



CI/CD Pipeline Steps

Build	Build the code and the output is a jar file ready to deploy
Run MUnit tests	Run MUnit tests and generate a testing report. Fail if MUnit tests fail
Code quality checks	Scan code using common coding standards
Dependency/security checks	Scan code and dependencies in your Mule application and fail if security risks are found
Artifact Repository	Upload jar to artifact repository such as Nexus
Manual approvals	Useful for Production deployments
Deployment	Deploy to the correct environment

Building and Unit Testing

- Building the MuleSoft application compiles the code as a Java application and outputs a jar file that can be used for deployment
- Running MUnit tests will output a report of successful and failed unit tests
  - Can choose to stop the CI/CD pipeline on any failed unit tests
- Execute the "*mvn clean package*" command to build and run unit tests

Code Quality Checks

- Code quality checks provide a useful way to automatically enforce coding best practices
  - Code review in the SDLC is very important, but prone to human mistakes
  - Automated code quality checks can fill in gaps from code review
- Tools for code quality checks
  - SonarQube
  - SonarCloud
- Execute the "*mvn sonar:sonar*" command to run SonarQube code quality checks
- Produces a report that the CI/CD pipeline can use to succeed or fail based on code quality

Artifact Repository

- Useful for a centralized location of project snapshots and releases
  - Can build code and later deploy the code from the pre-built jar in the repository
- Tools for artifact repository
  - Nexus
- Execute the "*mvn deploy*" command pointing to Nexus to upload the jar to Nexus

Manual Approvals

- For deploying to Production, recommended to include a manual approval step
  - Approved stakeholder or team member can approve deployments at specific times
  - Eliminates the possibility of an accidental commit being made to the master branch and deploying to Production accidentally
  - Run fully automated CI/CD in lower environments, but cut off continuous deployments in Production to ensure the highest quality Production deployments
    - Production deployments may require a change advisory board approval or coordinating support and development staff before deploying

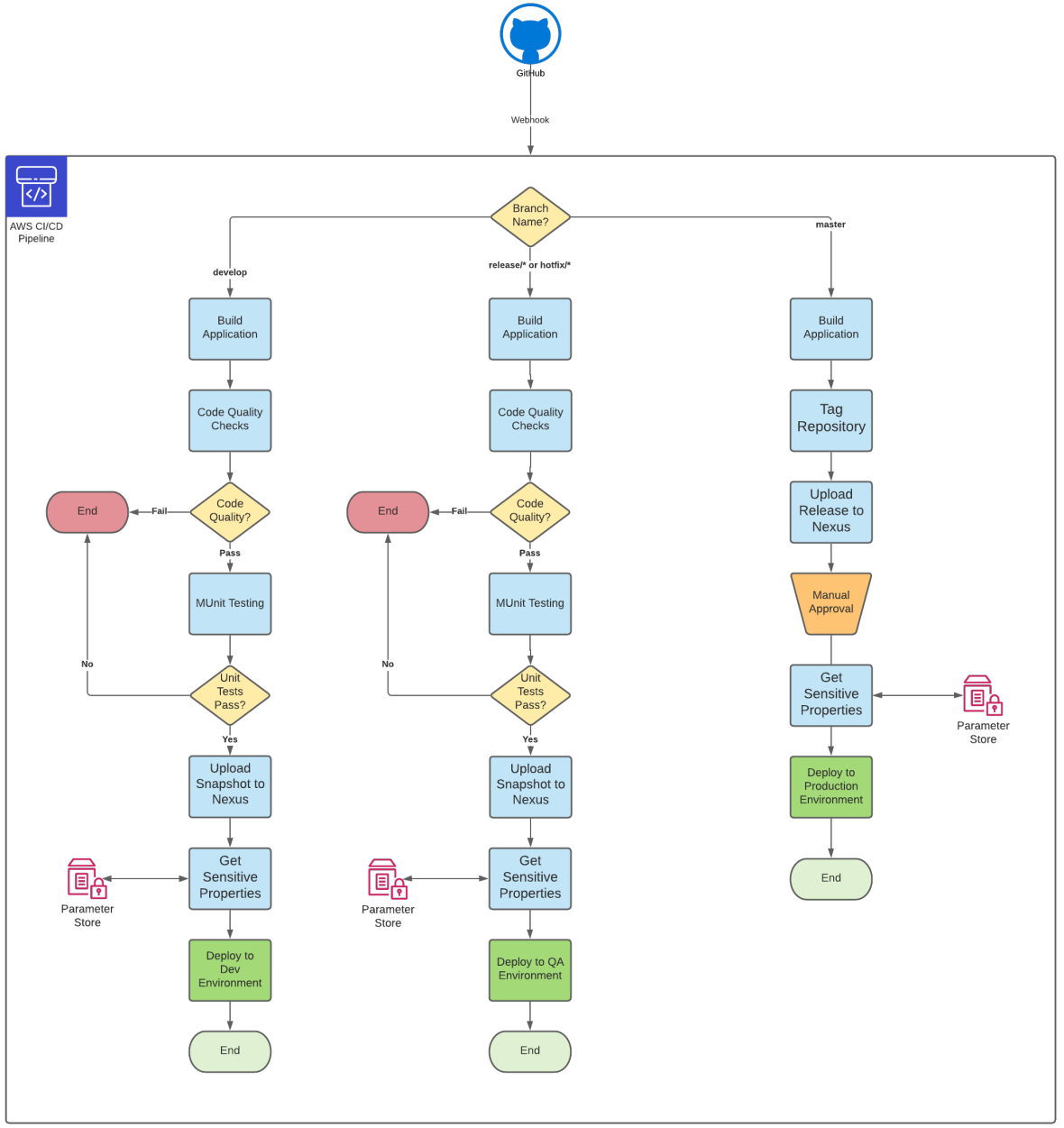
Deployments

- If all tests and checks pass, can automatically deploy to the correct environment using Maven
  - Use the branch name to enable deployments to the appropriate environment
  - Recommended to tag the repository upon Production deployment
- Inject sensitive properties at deploy time
  - Get sensitive properties from a secure store at deploy time and include them in the Maven deployment
- Execute the "*mvn deploy -DmuleDeploy*" command to deploy the MuleSoft application
  - If uploading jars to Nexus, can use the previously built jar in Nexus using the command "*mvn mule:deploy -Dmule.artifact=myProject/myArtifact.jar*"

CI/CD Scenario #1

An organization recognizes the need to automate builds and deployments of their MuleSoft applications. The organization has not set up CI/CD with MuleSoft yet and requires the MuleSoft architect to define the CI/CD process and tools to use. The organization already uses GitHub for source control and has an AWS account. They would like to leverage their existing GitHub and AWS accounts for the CI/CD implementation.

CI/CD Scenario #1 Architecture



DevOps Summary

- Automating Deployments
- Maven
- CI/CD Pipeline Steps

Additional Reading

- https://docs.mulesoft.com/mule-runtime/4.3/deploying
- https://docs.mulesoft.com/mule-runtime/4.3/mmp-concept#add-mmp
- https://docs.mulesoft.com/mule-runtime/4.3/deploy-to-cloudhub
- https://docs.mulesoft.com/mule-runtime/4.3/deploy-on-premises
- http://workshop.tools.mulesoft.com/modules/module7_lab4
- https://docs.mulesoft.com/munit/2.3/coverage-maven-concept
- https://vanchiv.com/how-to-use-mulesoft-sonarqube-plugin/
- https://support.sonatype.com/hc/en-us/articles/115006744008-How-can-I-programmatically-upload-an-artifact-into-Nexus-3-