MULESOFT ARCHITECTURE:
DevOps Part 1

PRACTIC LEARNING

DevOps Introduction

1. DevOps goal
2. DevOps in practice
3. DevOps and SDLC relationship
4. DevOps security

DevOps Overview

DevOps Mission

- DevOps aims to increase speed and efficiency for an organization
- DevOps should fit as seamlessly as possible into the organization’s fabric and the development team’s processes
- DevOps vs. DevSecOps
  - DevOps focuses on speed and efficiency
  - DevSecOps focuses on security, and may sacrifice speed and efficiency for highly secure code

DevOps Implementation Process

Convince	Convince stakeholders of the benefits to change the mindset and culture to focus on speed, efficiency, and innovation
Adopt	Adopt Agile methodology and best practices
Implement	Implement CI/CD
Use	Use distributed source control like GitHub
Integrate	Integrate code review into SDLC
Secure	Secure sensitive properties
Create	Create a reusable testing strategy
Set up	Set up monitoring tools
Establish	Establish a KPI baseline and measure success

DevOps In Practice

DevOps Mindset

- If implementing new DevOps processes, convince the business and stakeholders of the benefits of DevOps and Agile
- Benefits of DevOps include
  - Increased speed
  - Fewer mistakes
  - Easier and faster deployments
  - Enhanced innovation
- Align developers to DevOps best practices and standards
- Leverage Agile methodologies

Adopt Agile Methodology

- Agile focuses on iterative development, developing small pieces of code quickly across distributed teams
- Benefits of Agile
  - Aligns development team
  - Forms an organized structure for the entire development team which garners accountability
  - Allows people from all skill levels to contribute to consumable pieces of work
- As an architect, leverage Agile methodology to allow a development team to implement designs
  - Set up Agile ceremonies such as daily standup
  - Plan sprints with well-defined stories
  - Create a backlog with well-defined epics and stories

CI/CD Overview

- CI = Continuous Integration
  - The development lifecycle is continuously integrated with source control, builds, testing processes, etc
  - Continually iterating on an application using small features
- CD = Continuous Deployment
  - Automatically deploy to a desired environment as a team checks in code, checks pass, and testing passes
- Benefits
  - Speed up deployment
  - Reduce developer responsibility for testing
  - Eliminate responsibility of developers for deployments
  - Remove costly time developers spend deploying
  - Eliminate manual errors deploying

Source Control

- Source control is the basis for a good API lifecycle and SDLC
  - Required for a team to collaborate on code
- Common source control
  - GitHub
  - GitLab
  - AWS CodeCommit
  - Azure Repos

Source Control Branching Model and SDLC

- Best practice: follow GitFlow for branching
- Use source control branches to deploy to the appropriate environment
    - **develop** branch deploys to **Dev** environment
    - **release** branch deploys to **QA** environment
    - **master** branch deploys to **Production** environment
    - **hotfix** branch deploys to **QA** environment
- Allows a defined process to dictate deployments
    - Developers simply check in code and automated deployments occur in the background

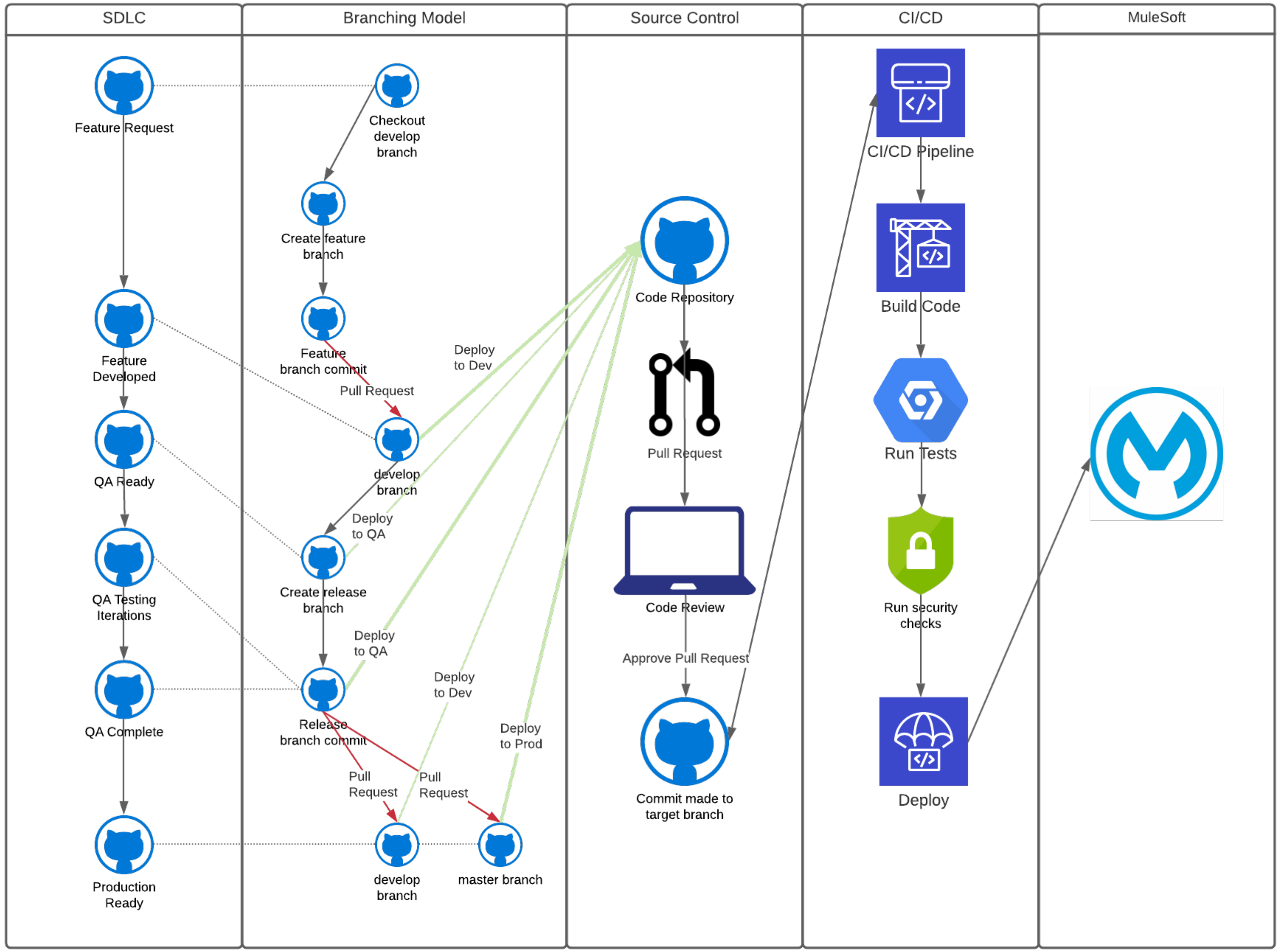
Code Reviews

- Use source control to perform code review on protected branches when developers finish development and testing
  - Integrated seamlessly into GitFlow and SDLC
  - Can be done with pull requests (PR) or merge requests (MR)
- Protected branches
  - develop
  - master
- When a PR or MR is opened on develop or master branch, use a code review checklist for approved reviewer(s) to review the code
  - Ensures top quality code
  - Helps eliminate coding errors

DevOps Scenario #1

An organization has not set up any formal DevOps, CI/CD, or source control standards and processes. As an architect, you are tasked with designing the end-to-end process including the source control branching model, source control code review process, CI/CD pipeline steps, and automated deployments. The process should follow standard SDLC processes.

DevOps Scenario #1 Architecture



Configuration Files

- Used to store global properties and environmental specific configurations
- Configuration (config) files live in /src/main/resources/{env}.yaml in a Mule app
  - Name yaml files after the environment they are used for. Examples:
    - dev.yaml
    - qa.yaml
    - prod.yaml
- How are sensitive properties stored securely?

Securing Sensitive Properties

- Keep your sensitive data secure!
  - Most stolen credentials come from plaintext sensitive properties
- Do not check sensitive properties into source control in plaintext
  - Attacks occur from stolen credentials that are stored in plaintext
  - GitHub has a feature that scans your repositories for plaintext sensitive data
- Develop a DevOps process to house your sensitive data securely

Strategies For Securing Sensitive Properties

Store in a Centralized External Location (Recommended)

- Store sensitive properties encrypted in a key/value store such as in Azure DevOps
  - Can be stored in your CI/CD pipeline at rest
  - Only accessible by systems/people with access
- When CI/CD initiates a build/deploy, it decrypts the sensitive properties and injects them into your application at deploy time
  - Upon deployment, safely hide the sensitive properties so they are not in plaintext
- This approach ensures no sensitive data is checked into source control
- To run the application locally, must use VM arguments

Encrypt Sensitive Properties in the MuleSoft Application

- Encrypt sensitive properties in the {env}.yaml config files using the MuleSoft's Secure Configuration Properties
- Ensure that the encryption key used is stored securely
  - Can be stored external to MuleSoft in a CI/CD pipeline's configuration
  - Can also be stored at rest in a secure vault
- This approach runs the risk of
  - Checking sensitive properties into source control in plaintext
  - Exposing the encryption key

Testing

- Recommended tests to execute
  - Regression testing (for iterative releases)
  - End to end testing
  - Security testing
  - Unit tests in MUnit
    - Encouraged to create a test-driven development (TDD) environment, but not always feasible
- CI/CD can automate unit testing
  - Run MUnit tests as a step in the CI/CD pipeline
    - MUnit tests help establish a baseline for future iterations of code saving time long term
    - Should provide good code coverage
    - MUnit tests should be run for any new commits to develop and master branch of source control
    - If MUnit tests fail in the CI/CD pipeline, fail the build and optionally alert

Monitoring

- For each application, ensure monitoring is in place after the initial deployment
  - Helps track KPIs over time
  - Establishes a baseline for normal behavior
  - Enables support and operations team to help support the deployed applications
    - Knowledge transfer and handoff may need to occur
- Anypoint Monitoring is extremely useful if enabled based on the MuleSoft subscription (recommended approach to monitoring MuleSoft applications)
  - Can also use an external system if Anypoint Monitoring capabilities don’t exist

Establish Key Performance Indicators (KPIs)

- As an organization, you can define KPIs to track across all applications as a part of C4E
  - There may be some applications where monitoring different KPIs makes sense
- Steps to establish KPIs
  1. Define KPIs to track and monitor
  2. Implement KPI tracking and dashboards
     - Can use 2 great pre-built tools to measure KPIs
       - MuleSoft Metrics Accelerator
         - MuleSoft accelerator built as a part of Catalyst containing well documented, common metrics of the Anypoint Platform
       - Big Compass KPIs for APIs Dashboard
         - Great dashboard with many KPIs built by Big Compass (MuleSoft partner), many focusing on ROI and executive-level KPIs such as reuse of APIs
  3. Record and document KPI baseline for tracking purposes over time
  4. Decide how often KPIs will be recorded for tracking purposes over time
  5. Update documentation periodically

DevOps Summary

- DevOps Overview
- DevSecOps Overview
- DevOps Best Practices
- How DevOps Integrates With and Drives the SDLC

Additional Reading

- https://www.mulesoft.com/resources/devops-integration
- https://docs.mulesoft.com/mule-runtime/4.3/configuring-properties
- https://docs.mulesoft.com/mule-runtime/4.3/secure-configuration-properties
- https://help.mulesoft.com/s/article/How-to-pass-additional-startup-arguments-to-Mule
- https://help.mulesoft.com/s/article/Studio-7-ignores-Default-global-VM-arguments
- https://nvie.com/posts/a-successful-git-branching-model/