Technical Design Document

Azure DevSecOps Pipeline

UMGC Capstone

11 October 2020

Contents

[Introduction 3](#_Toc53312765)

[Requirements of this Document 3](#_Toc53312766)

[Scope 3](#_Toc53312767)

[Document Reference 3](#_Toc53312768)

[Design 4](#_Toc53312769)

[Platform/Technology 4](#_Toc53312770)

[Requirements 4](#_Toc53312771)

[Assumptions and Constraints 4](#_Toc53312772)

[Strategy 5](#_Toc53312773)

[Bootstrap Environment 5](#_Toc53312774)

[Pipeline as Code & Sourced Controlled Pipelines 5](#_Toc53312775)

[Workflow 6](#_Toc53312776)

[Azure Pipeline Design 7](#_Toc53312777)

[Build Node 7](#_Toc53312778)

[Build 7](#_Toc53312779)

[Test 8](#_Toc53312780)

[Static Analysis 8](#_Toc53312781)

[Promote Artifacts 8](#_Toc53312782)

[Deploy 8](#_Toc53312783)

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Change Summary** |
| 10 October 2020 | Glenn Goodlett | 0.1 | Initial Draft |
| 11 October 2020 | Glenn Goodlett | 1.0 | Prerelease |
| 11 October 2020 | Dustin Emerson | 1.0 | Peer Review |

# Introduction

This document the UMGC Capstone Azure DevSecOp Pipeline system as designed. This document outlines the structure of the pipeline and how it integrates with various technologies. Upon completing the document, the reader shall be capable of developing, maintaining, and processing future feature requests to the Azure DevSecOp Pipeline system.

## Requirements of this Document

This technical document describes the various technologies within their nomenclature context. Readers shall be informed at a basic level of the following technologies and terminologies:

* Linux
* BASH
* Make
* git
* Java
* Maven
* Docker
* Azure & Azure DevOps
* Continuous Integration Continuous Deployment (CI/CD)
* SonarCloud

## Scope

This document is aimed at technical readers that will develop, maintain, or desire an understanding of the Azure DevSecOps Pipeline system.

## Document Reference

|  |  |  |
| --- | --- | --- |
| Ref # | Document name | URL |
| 1 | The Linux Documentation Project | <https://tldp.org/guides.html> |
| 2 | BASH Reference Manual | <https://www.gnu.org/software/bash/manual/bash.html> |
| 3 | GNU Make | <https://www.gnu.org/software/make/manual/make.html> |
| 4 | git | <https://git-scm.com/doc> |
| 5 | Java | <https://docs.oracle.com/en/java/javase/index.html> |
| 6 | Apache Maven Project | <http://maven.apache.org/guides/index.html> |
| 7 | Docker Docs | <https://docs.docker.com/> |
| 8 | Azure Documentation | <https://docs.microsoft.com/en-us/azure> |
| 9 | CI/CD Development Documentation | <https://docs.gitlab.com/ee/development/cicd> |
| 10 | SonarCloud documentation | <https://docs.sonarqube.org/latest/> |

# Design

The purpose of this design is to model a system that will provide a robust automated system for building, testing, and deploying UMGC Capstone applications.

## Platform/Technology

The platform and technology for the DevSecOps Capstone will leverage the Microsoft Azure cloud computing service. Microsoft Azure is commonly referred to as Azure and allows for building, testing, deploying, and managing applications and services via Microsoft-managed data centers. This design incorporates both the Azure portal technology and Azure DevOps for a complete CI/CD process.

## Requirements

|  |  |
| --- | --- |
| Ref # | Description |
| RE-1 | The system shall leverage the Azure Technology infrastructure. |
| RE-2 | The system shall be capable of building Java version 8-11 applications. |
| RE-3 | The system shall gate git Pull Request based on Azure pipeline triggers. |
| RE-4 | The system shall be capable of running Java version 8-11 tests. |
| RE-5 | The system shall publish test results via git pull request integrations. |
| RE-6 | The system shall be capable of creating Docker images. |
| RE-7 | The system shall promote and upload Docker images to docker.io. |
| RE-8 | The system shall be capable of deploy applications. |

## Assumptions and Constraints

|  |  |
| --- | --- |
| Ref # | Description |
| CO-1 | The system is constrained to Azure free or student subscription services. |
| AS-1 | The free or student subscription service provides all functionality outlined in the requirements. |

## Strategy

The DevSecOps Azure Pipeline system design strategy employs a robust system by bootstrapping environmental dependencies, pipeline as code concepts, and source-controlled pipelines.

### Bootstrap Environment

One of the most painful issues with complex automated systems for building applications is the environmental dependencies required for building an application. The DevSecOps Azure Pipeline system mitigates this concern by building an environment capable of building more complex applications with minimum system requirements; this process is known as bootstrapping and achieved via the Advance Development Factory (See the ADF System Design Document for specifies).

### Pipeline as Code & Sourced Controlled Pipelines

Another concept of managing a group's pipeline is the structure that drives the pipeline process. A common approach to pipeline setups leverages the pipeline technologies tool to outline the pipeline process, usually in a graphic user interface; this is problematic as the pipeline structure is not source controlled, exists ephemerally, and developers are unaware of the structure. The DevSecOps Azure Pipeline system mitigates this by utilizing the Azures YAML pipeline declarations. Azures YAML pipeline declaration enables teams to define a pipeline process via a YAML file stored in the applications source-controlled solution. This allows for seamless integration with git and Azure while enabling pipeline testing.

## Workflow

Figure 1 depicts a typical developer lifecycle to provide context for the DevSecOps Azure Pipeline system design.

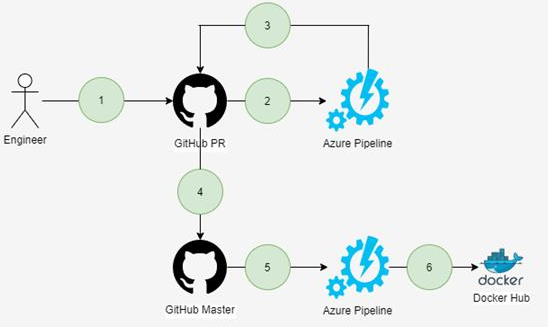


Figure 1. Developer Lifecycle

1. A developer makes a code change and creates a git pull request.
2. The DevSecOps Azure Pipeline system automatically detects the git PR and triggers a pipeline run that builds and tests the application.
3. The DevSecOps Azure Pipeline system notifies the git PR on the success or failure of the pipeline run.
4. Upon a successful pipeline run and other git PR requirements, the developer merges the PR into the master branch.
5. The DevSecOps Azure Pipeline system automatically detects a change in the master branch and runs the pipeline that builds and tests.
6. Because the change is to the master branch, the pipeline also promotes the applications Docker image to docker.io

# Azure Pipeline Design

Below outlines the high-level design of the DevSecOp Azure Pipeline system. This design is broken into the following section:

* Build Node – The Azure entity that drives the pipeline process
* Build – The pipeline stage that builds the UMGC Capstone application.
* Test – The pipeline stage that tests the UMGC Capstone application.
* Static Analysis – The pipeline stage that performs a static analysis on the UMGC Capstone application.
* Promotion – The pipeline stage that uploads build artifacts of the UMGC Capstone application.

## Build Node

The Azure build node is the entity that drives the pipeline process. The build node for the pipeline only requires the following environmental dependencies due to utilizing ADF, as outlined in the strategy bootstrap environment section.

* Linux
* Bash
* Make
* Docker

The DevSecOps Azure Pipeline system will use the provided VM image ubuntu-20.04 and is defined in the Azure YAML file.

*Sample: azure-pipeline.yml*

pool:  
 vmImage: 'ubuntu-20.04'

## Build

The build process of the pipeline is defined in the applications git repo's makefile. This process will build and unit tests the Java application producing a Java jar and construct the Docker image.

*Sample: azure-pipeline.yml*

steps:  
- task: CmdLine@2  
 inputs:  
 script: 'make all'

*Sample: Makefile*

all:  
 docker run -v $(PWD)/:/repo --entrypoint '/bin/bash' $(BUILD\_IMG) \  
 -c 'cd /repo && make target/$(VLOL\_JAR) VERSION=$(VERSION)'

target/$(VLOL\_JAR):  
 mvn $(MAVEN\_OPTS) package -f pom.xml

## Test

The testing of the application is conducted in the build stage; however, the test results' publication is performed in this stage. If the results of the test do not pass, the pipeline is considered a failure.

*Sample: azure-pipeline.yml*

- task: PublishTestResults@2  
 inputs:  
 testResultsFormat: 'JUnit'  
 testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml'  
 failTaskOnFailedTests: true

## Static Analysis

The static analysis of the application is conducted via SonarQube and is not conditionally measure against a pipeline run's success. This process runs a series of static analysis algorithms provided by SonarQube and publishes them to the SonarCloud.io for developer review.

*Sample: azure-pipeline.yml*

- task: CmdLine@2  
 inputs:  
 script: 'make sonar'

## Promote Artifacts

Provided previous successful pipeline stages and pipeline runs on the master branch, the promotion stage uploads the Docker image created in the build stage.

*Sample: azure-pipeline.yml*

- task: CmdLine@2  
 condition: and(succeeded(), eq(variables.isMain, true))  
 inputs:  
 script: 'make push'

## Deploy

The Deployment process for application will leverage Azure deployment templates and is outlined in the deployment and operations guide.