|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| UMGC Capstone | | | | |
|  |  | | |  |
| DevSecOps | | | | |
|  | | Programmers Guide |  | |

TABLE OF CONTENTS

[Introduction 3](#_Toc53837138)

[1. Getting started 4](#_Toc53837139)

[Accounts and Permissions: 4](#_Toc53837140)

[Building a Roster 5](#_Toc53837141)

[Getting Everyone on Microsoft Teams 6](#_Toc53837142)

[Getting Everyone on GitHub 7](#_Toc53837143)

[Setting up Repositories on GitHub 9](#_Toc53837144)

[2. Managing GitHub 12](#_Toc53837145)

[Process Overview 12](#_Toc53837146)

[Criteria for Merging to Master 13](#_Toc53837147)

[Cheat codes 13](#_Toc53837148)

[3. your Environment 14](#_Toc53837149)

[Setup your environment 14](#_Toc53837150)

[Testing your environment (and dipping your toes in ADF) 15](#_Toc53837151)

[4. The Automatic Deployment Factory (ADF) 16](#_Toc53837152)

[Maintaining 17](#_Toc53837153)

[Integrating ADF Into A Development Project 18](#_Toc53837154)

[Testing the ADF Integration on Your Local 26](#_Toc53837155)

# Introduction

This document outlines the best practices and processes that DevSecOps teams will adhere to in order to ensure success in the UMGC SWEN 670 course. To be successful in the course, the DevSecOps team will be required to take ownership of the build and deployment process. In addition to that, they will often be required to work through any issue’s development teams are having with GitHub and Azure. In this capacity, DevSecOps should serve as Experts in everything regarding what it takes to test and deploy software in a continuous fashion, while also being generally skilled in most software engineering practices.

This remainder of this document will operate within the context of the UMGC software engineering capstone project context, referring to the business as the class as a whole and project teams as development teams. The purpose of this document is to inform, and thus the remainder of the document will be focused on technical steps.  
  
This guide will focus on steps specific to DevSecOps for the SWEN 671 course. Instead of outlining walking through "How to install VirtualBox on Windows," a question that, if answered, would surely be rendered incorrect within a few releases, the guide will instead say "Use Virtual Box." A quick Google search will provide you with the most up to date information.

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reasons for change | Version |
| Dustin Emerson | 10/20/20 | Initial Draft | V 0.1 |
| Glenn Goodlett | 10/21/20 | Scrub for Accuracy | V 0.2 |
| Dustin Emerson | 20/22/20 | ADF section | V 0.3 |
| Glenn Goodlett | 20/23/20 | Integrating Tools | V 0.4 |
| Dustin Emerson | 20/24/20 | Review | V 0.5 |
| Glenn Goodlett | 20/25/20 | Review | V. 06 |
| Dustin Emerson | 20/26/20 | Finalize | V 1.0 |

# Getting started

Your first priority should be to ensure the entire class has access to the resources they need. This section will walk you through your priorities for the beginning of the course.

## Accounts and Permissions:

As DevSecOps, you will need to use the admin account umgc-capstone-bot to get a lot of your work done. You will need a User Names and passwords for the following services. Each of which will be provided by your professor or mentor:

* GitHub
* Gmail
* Azure
* One Drive

## Building a Roster

You are going to need to build a Roster with the following information. Keep in mind this roster could change over time:

1. Project Name
2. What technologies are they using
3. Who is on what Team
   1. Name
   2. School Email (students.umgc.com domain)
   3. Personal Email
   4. GitHub username / email
   5. Are they assigned to the correct team in GitHub?
   6. Are they on Microsoft Teams?
   7. Are they assigned to a channel for their team in MS Teams?

Example:

Project Tracker:

Java 12  
Spring  
MySQL

|GitHub Team |MS Teams |MS Teams Channel|

Blarx Mueno -> bmueno@student.umgc.edu | YES | YES | YES |

bmeuno\_cod4ever@gmail.com  
 GitHub:

Username: bmeuno  
 Email: bmeuno\_cod4ever@gmail.com

Kapran Smoke -> ksmoke@student.umuc.edu | NO | YES | YES |

usafmurzinski@gmail.com

## Getting Everyone on Microsoft Teams

The development teams are going to need a way to speak with each other easily. You will need a way to speak to them effortlessly. Microsoft Teams serves as the organization hub. Your mentor should add you to the UMGC SWEN Capstone organization. Once being added to MS Teams, you will need to do the following:

1. Create the following channels in MS teams
   1. One channel for each project
   2. A channel for general chat
   3. A Channel for DevSecOps to communicate internally
   4. A "DevSecOps – Help Desk" channel for teams to reach out to DevSecOps for help.
2. Ensure all students are invited to the UMGC SWEN Capstone organization.
3. Once a student accepts their invitation, add them to the following channels:
   1. General
   2. Their project team's channel
   3. DevSecOps Help Desk
4. The students should be "Members" of DevSecOps. UMGC Capstone Admin, and your mentor/professor as owners.

## Getting Everyone on GitHub

Everyone is going to have to use GitHub. You should have received the login and password information from the professor/mentor. Logging into GitHub for the first time with that account, and you will see **Your Teams** listed on the left navbar. Prior to starting the course, there should a team already created for your semester (example: umgc/fall-2020.)

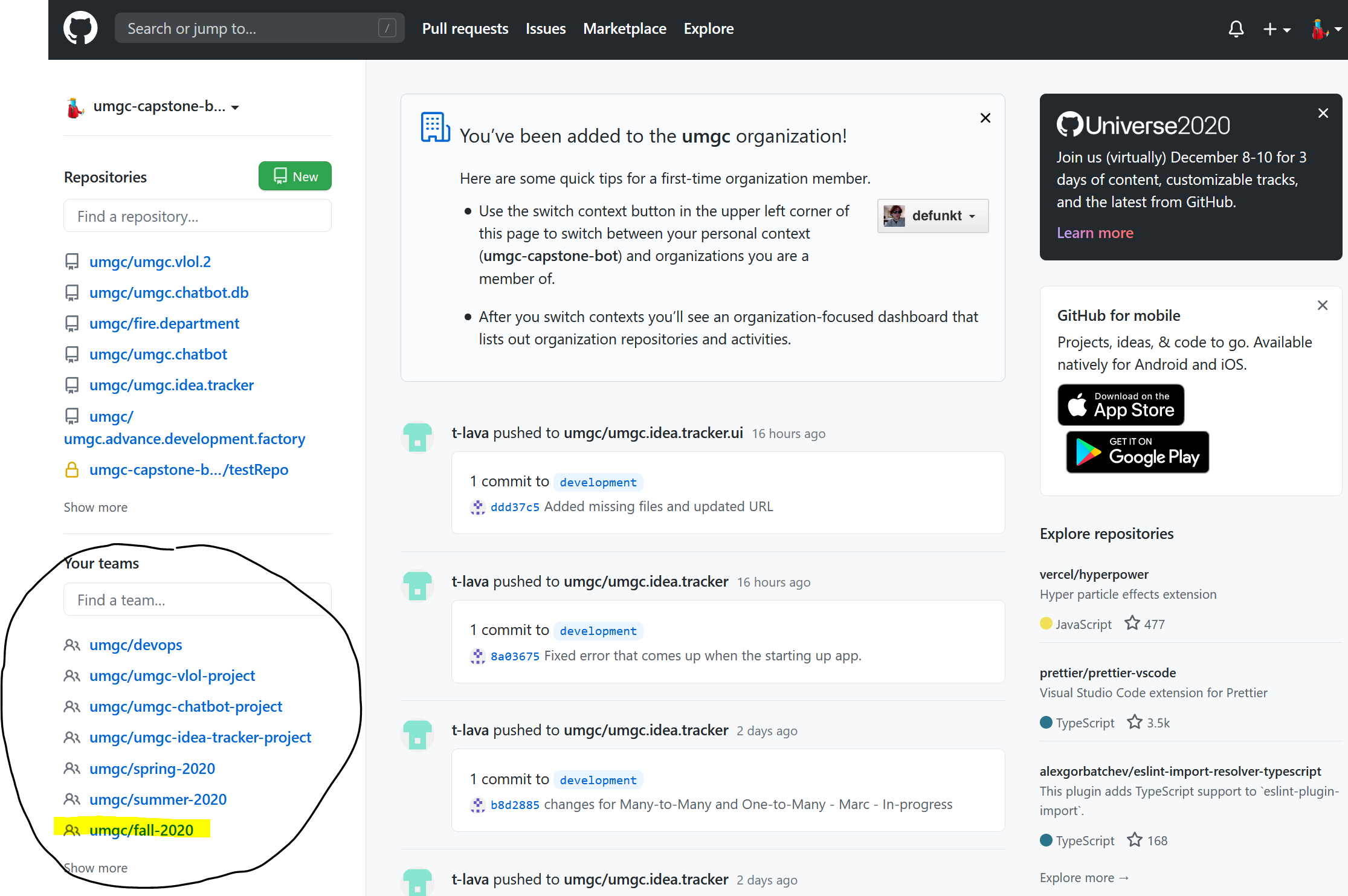


Figure: The GitHub dashboard for UMGC-capstone-bot

Click your semester's team to navigate to the team dashboard for your semester. You should have sub-teams for the development teams created before the semester started. If not, you can create any teams you need from here. Proceed by clicking on the development team you wish to add members to.

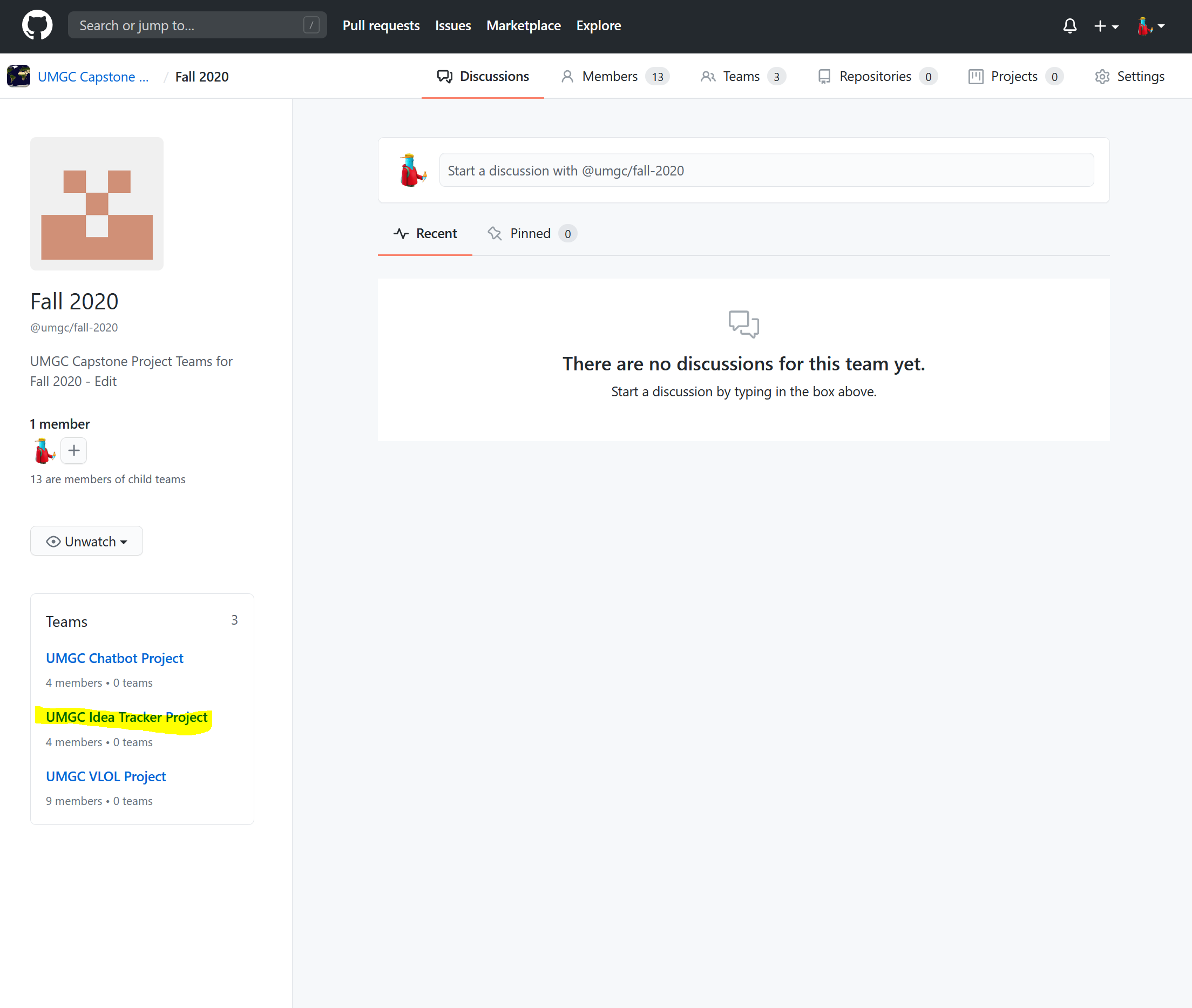


Figure: Semester Team Dashboard

You should now be on the Dashboard for the Development Team. On the left side of the navbar, you should see a list of members. Using the roster you created, add the correct members to the team. Members should be added with the following permissions:

1. UMGC-captsone-bot -> Owner
2. Members of DevSecOps -> Owner
3. Members of Development Team -> Maintainer

Repeat the above steps for all other teams.

## Setting up Repositories on GitHub

Development Teams will need a place to host their codebase. You will be in charge of setting up and managing these repositories.

Create a new Repository

Navigate to The UMGC Capstone Course Dashboard by clicking the GitHub Octocat Icon in the top left. From there, **new** from the repository sub-menu of the navigator pane.

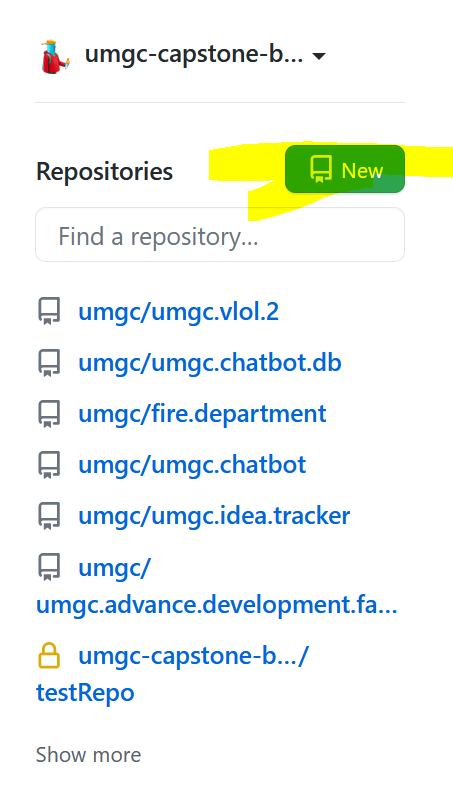


Figure: Repository List

This will bring you to the create new repository dialogue. Fill out the form as such and create.

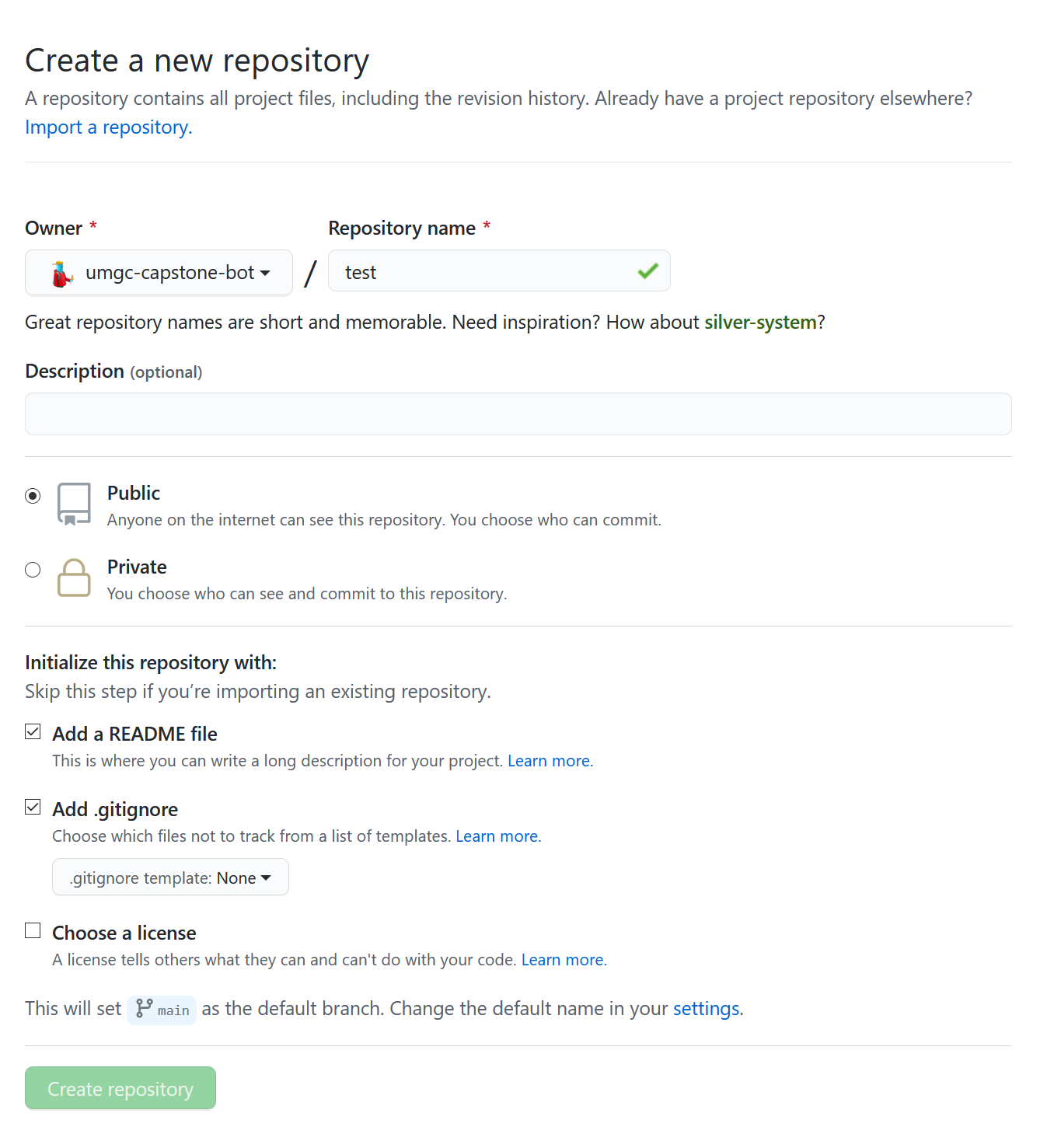


Figure: New Repository Dialogue

This will bring you to the Repository page. From here, create a new branch called "devevopment" by clicking the branch icon, typing "development," and clicking "Create Branch."

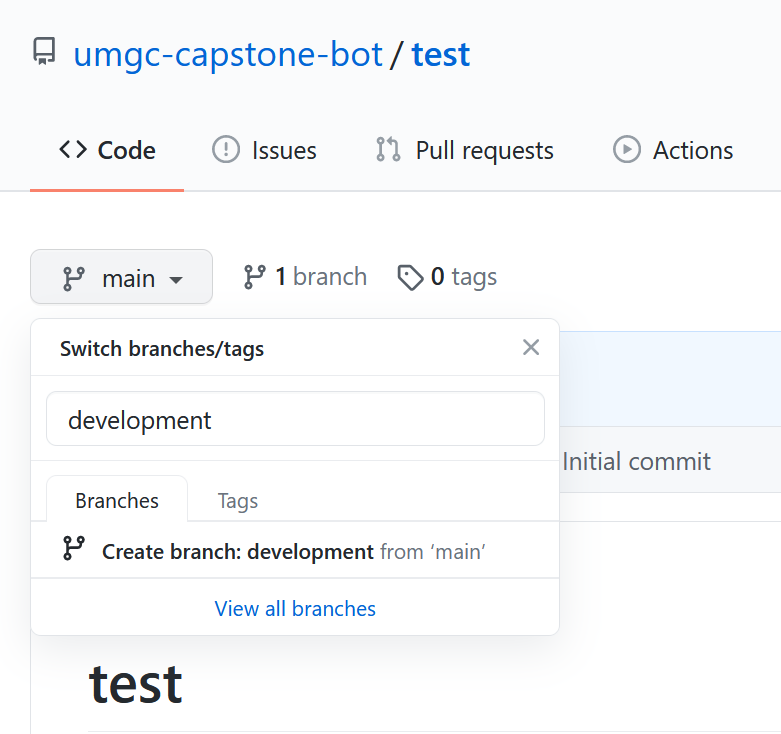


Figure: Creating a development branch

Once the development branch is created, it's time to configure the security preferences.

Configuring Preferences

An important concept in CI/CD is that the main branch should always build. In order to facilitate that e need to establish a set of rules to protect the main branch.  
  
From the Repository page from before, click **Settings**. From there, click **Branches** and select **Add Rule.**

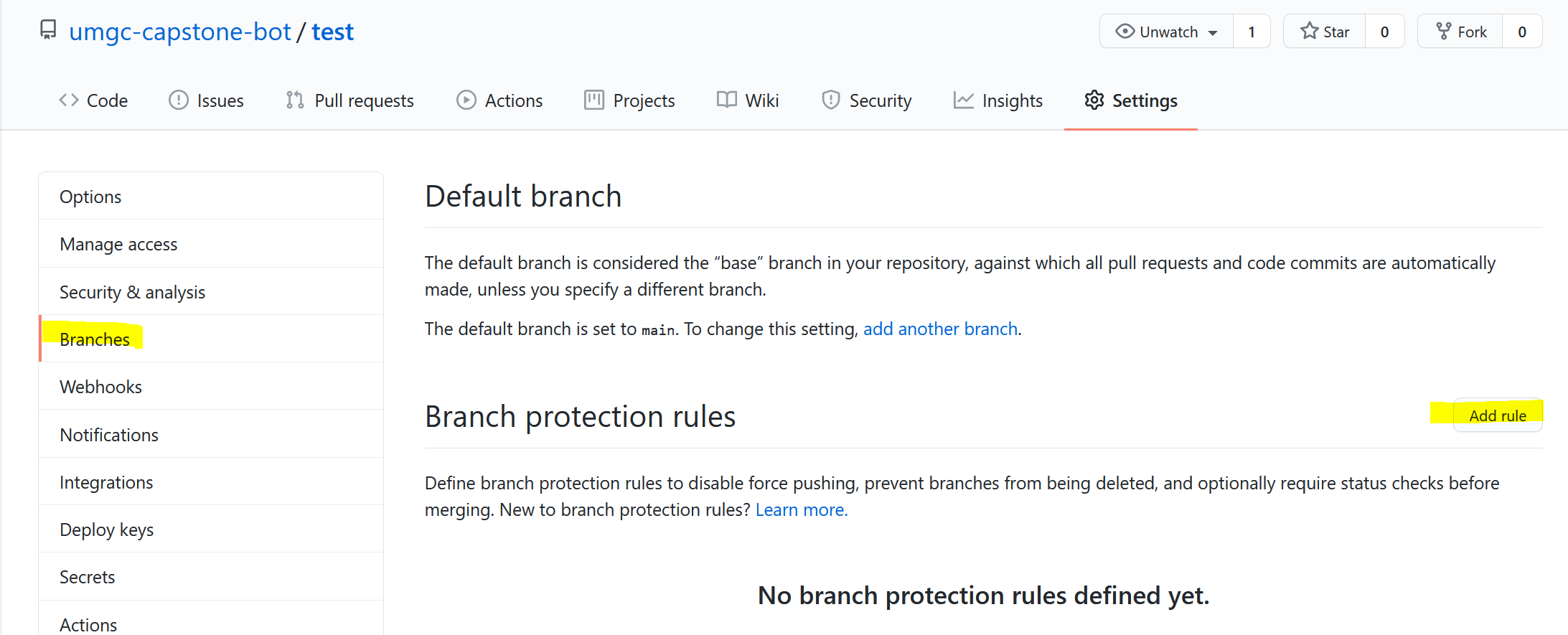


Figure: Repository Settings

Configure the following settings:

* Branch name pattern - The name of your main branch
* Require Pull Request Reviews before Merging - True
  + Require a review from Code Owners – True
* Require status checks to pass before merging – True
  + Require branches to be up to date before merging – True

Click create. Next, we need to add the repository to the team.

Adding the Repository to a Team

From the Development Team Dashboard, you will see a tab labeled **Repositories**, click that.

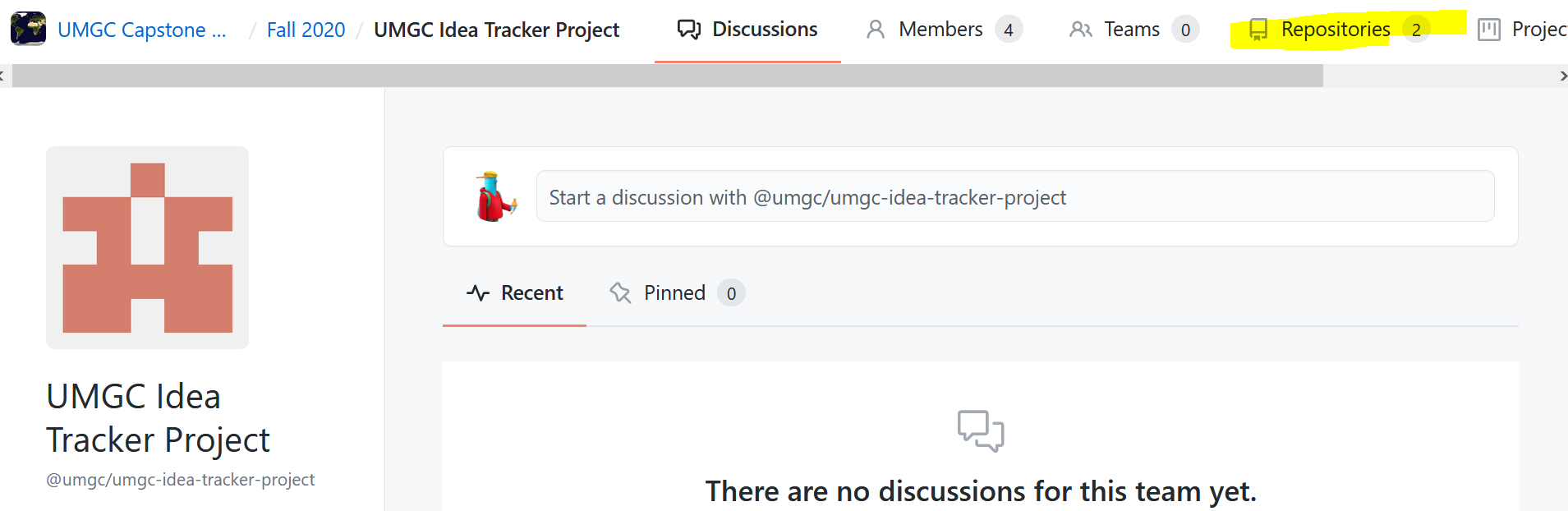


Figure: Development Team Dashboard

From there, you should be on the list of repositories. Click the green **Add repository** button.

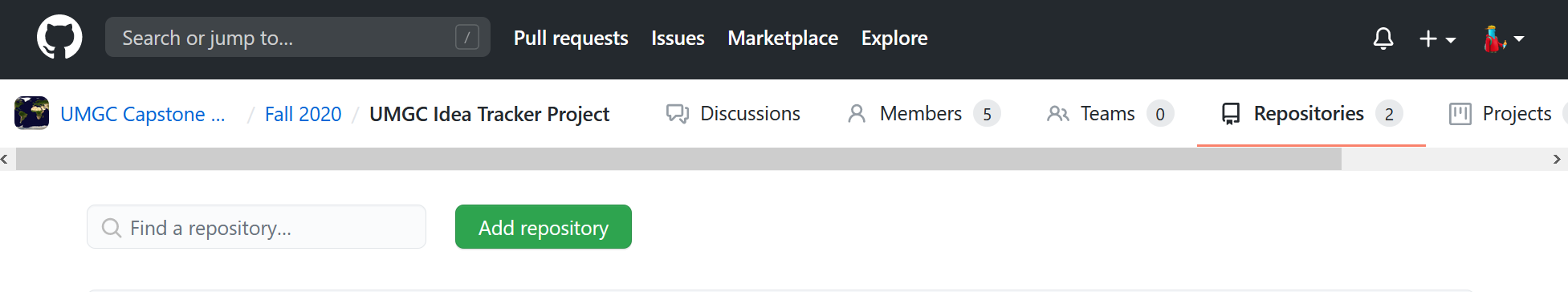


Figure: Add repository to Team

Type the name of the repository your repository and click **Add repository to the team**. This will add the repository to the team.

# Managing GitHub

Throughout the twelve weeks of this course, development teams will use git as their version control system. In addition to git, teams will utilize GitHub for hosting their work, tracking bugs, and facilitating code reviews.

## Process Overview

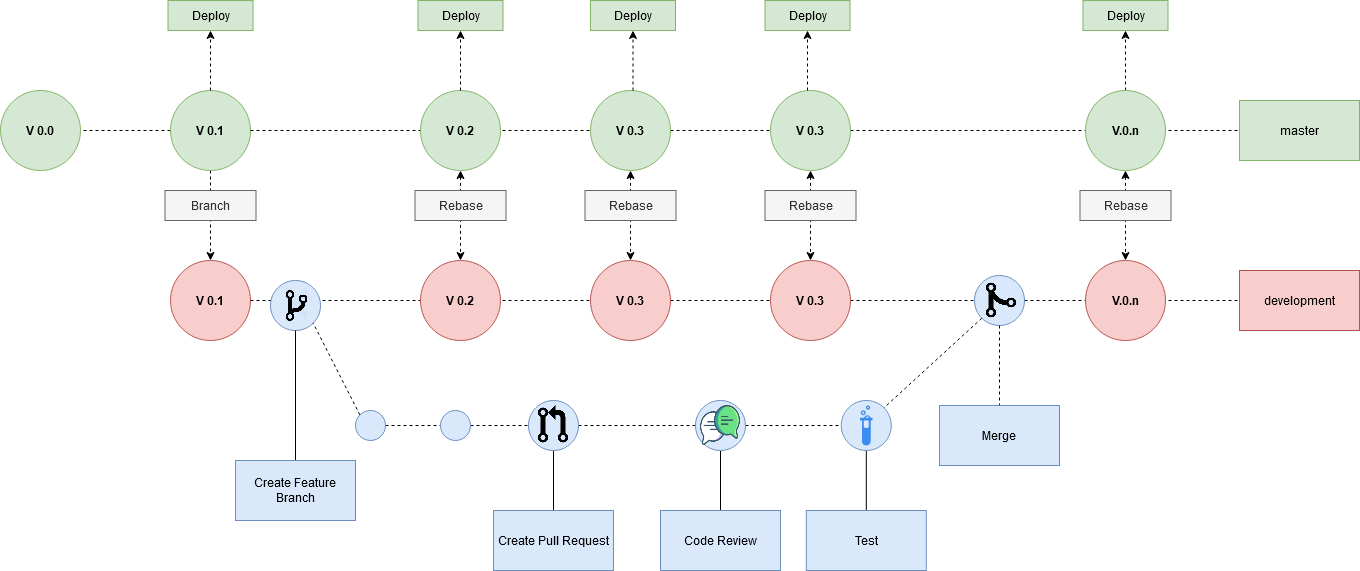


Figure: Version Control Process Overview

Projects repositories will be set up as such:

1. There is a mainline **master** branch that requires a member of DevSecOps to approve code being pushed to it. This branch will be pipelined as such where the project deploys after a build is successful.
2. There is a **development** branch forked off of master. This is the branch the dev teams will use when working. The team can decide how often they wish to move the development branch into master.

Development will be performed using a pull request model. Developers will checkout a new branch on their local machine off of the development branch when they begin developing a new feature. When the feature is ready to be presented for review, they will push their branch to GitHub and create a pull request for peer review. Once a member on their team and a member of DevSecOps has approved, the pull request will be tested. Finally, once passing tests, the branch will be merged into master.

## Criteria for Merging to Master

* The pull request must build.
* The code being added must pass a LINT evaluation.
* The code added or changed must be tested.
* The pull request cannot contain any confidential or personal information (passwords, social security, etc.)
* The pull request cannot contain binary files.
* 70%-80% Code Coverage in tests

## Cheat codes

1. Ensure the repository is cloned on your local machine:

git clone https://github.com/umgc/umgc.project.n.git

1. Checkout the development branch:

git checkout development

git pull origin development

1. Checkout a new branch for your feature

git checkout -b my-feature

1. Develop feature
2. Push branch to GitHub

git push origin **my-feature**

^ This is your branch name from earlier

1. Create a pull request on GitHub
   1. Follow [GitHub's documentation](https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/creating-a-pull-request) for creating a pull request.
2. Incorporate Feedback from team and DevSecOps
3. Once the pull request is accepted and tests pass, your work is ready to be merged into production.
   1. Follow [GitHub's documentation](https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/merging-a-pull-request) for merging a pull request.

# your Environment

## Setup your environment

As a member of the DevSecOps team, you will want to have an environment that is as close to the deployment environment as possible. For this, it is advised you get Linux on your machine. Setting up your environment early makes it easy to be successful the whole semester. The software you will need is outlined below.

**Required**:

* *git*– Source control (Note: every developer is required to use git)
* *Linux*– Our Operating system (OSX will also suffice)
  + Ways of getting Linux on Windows:
    - Windows Subsystem for Linux (WSL) – Requires Windows 10 professional edition. We recommend Core OS if you are going this route.
    - Oracle VM Virtual Box – Easy to use and free virtual machine to run Linux on.
* *Bash [version 4.2 or greater]*– Our shell
* *Make* [version 3.82 or greater] – Build automation tool
* *Docker [version 19.03 or greater]* – Containerization

**Suggested**:

* *Visual Studio Code*

Once you have set up your environment, you should create a directory on your local to hold everything for this class and clone all the repositories for each development team.

## Testing your environment (and dipping your toes in ADF)

Next, we will clone the **Advanced Deployment Factory (ADF)** repo. You can find the Advanced Deployment Factory at this address:  
<https://github.com/umgc/umgc.advance.development.factory>

This is a repository that is owned and operated by DevSecOps. The ADF sets the standard for developers to build, test, and deployment of a diverse set of applications under a common framework that mitigates clients operating environment dependencies. And every development project will depend on it. Once its clones, we can run some recipes in the makefile to ensure your environment is set up correctly:

# This builds the ADF Docker image.

~$ make build-env

# Expected Output:  
 # Successfully built d540a0f3160f

# Successfully tagged advance-development-factory:latest

# This starts the ADF docker image.

~$ make start-env

# you will now be in the docker container

[root@ba85cf5a7a50 repo]#

# you can type "exit" to exit the docker container

# The Automatic Deployment Factory (ADF)

Earlier, we cloned the Automatic Deployment Factory onto your local. The ADF provides all UMGC projects with a standard mechanism for building, testing, and deploying applications whilst limiting users' operating environment dependencies. ADF is able to accomplish a low operating environment dependency by building the environment in which users can build, test, and deploy applications. ADF is a bootstrap application context that leverages docker to isolate user's environments. Ensure you have the ADF project open whilst reading this guide.

The core features of the ADF provides are:

* An isolated operating environment
* A mechanism for building applications
* A Mechanism for testing applications
* A Mechanism for deploying applications

When ADF runs in the Azure pipeline, it pushed a docker image to docker.io. When a development project runs make all, the ADF build image will get pulled down and used as the container for the development project.

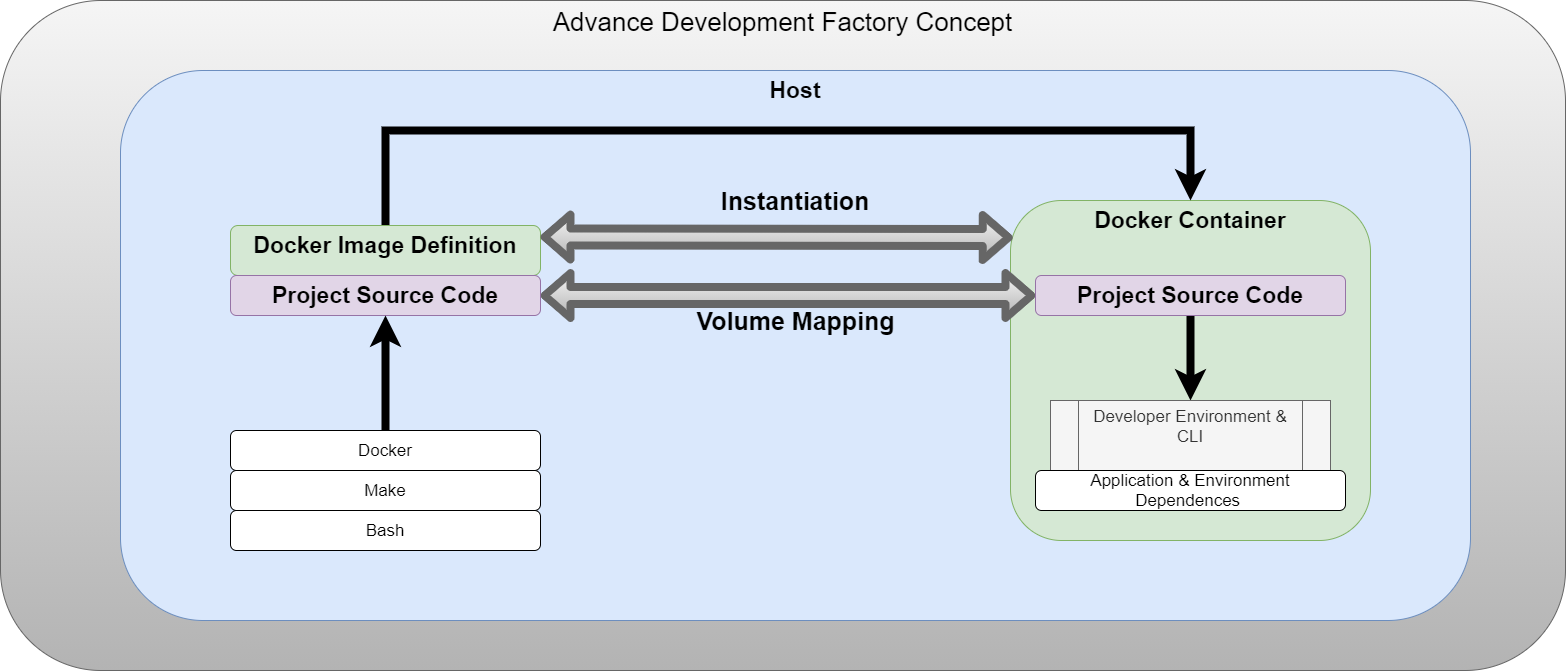


Figure: Advanced Development Factory

## Maintaining

The ADF could be thought of as three separate working components:

* Dockerfile
  + Configures the container, including all the dependencies required by other students.
  + If development teams have a dependency not covered by ADF, that dependency could be installed here using yum
* azure-pipelines.yml
  + Used by Azure. Defines the pipeline, which runs our make recipes.
  + ADF only needs three steps:
    - make build-env
    - Log in to docker.io
    - make push
* Make
  + We use make to define recipes for every step in the build process.

**Important Note:**

If you get acquainted with ADF now, once you start integrating the development teams' projects with ADF, you will notice that each of their projects also has the three mentioned above. The files are even in the same directory. This is because ADF could be thought of as a parent of the Development projects once configured properly. All Development projects shall conform to the structure of ADF with regards to where the Azure, Docker, and makefile are located.

## Integrating ADF Into A Development Project

As discussed above, the way ADF is structured is extremely similar to how development projects will be structured that use it. We will be creating the three files defined above in ADF, but this time in the development project itself.

Checkout a new branch to test your work

1. In the development project directory, checkout a new branch called "adf-integration."

Touch the pom file

1. The development team likely has their pom file configured to have their version be 0.0.1-SNAPSHOT. This needs to be changed to "1.0.0."
2. Note the Artifact ID of the application. For this guide, we will use ProjectApplicationName as the artifact ID
3. Add Azure version in pom:
   1. In the pom file, add the following to the properties

<properties>  
 <java.version>1.8</java.version> <!-- java version for project-->  
 <azure.version>2.3.5</azure.version>  
 <testcontainers.version>1.14.3</testcontainers.version>  
</properties>

* 1. If the team requires an azure starter, add it as a dependency:

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.microsoft.azure</groupId>  
 <artifactId>azure-spring-boot-starter</artifactId>  
 </dependency>

</dependencies>

<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>com.microsoft.azure</groupId>  
 <artifactId>azure-spring-boot-bom</artifactId>  
 <version>${azure.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
</dependencyManagement>

Create a MakeFile

1. In the root directory of the development project, create a file called "Makefile." This file will not only define recipes for building the project but also define variables to be used in maven, docker, possibly even the development application if you felt like it.
2. There will be some variation; the MakeFiles will roughly all look like this:

**Vars and properties:**

##############################################################  
# Makefile for building and testing  
##############################################################  
  
# Run make SKIP\_TEST=y  
SKIP\_TESTS:=  
  
# Version vars  
VERSION:=1.0.$(shell git rev-list HEAD | wc -l)  
# ^ A lot of teams will have their pom file say V0.0.1-SNAPSHOT  
# Their Pom needs to be changed to <version>1.0.0</version>  
PROJECT\_APPLICATION\_NAME\_JAR=ProjectApplicationName-1.0.0.jar  
# ^ The artifact name from the team's pom file  
  
# Docker vars  
APP\_NAME=project-application-name.app  
APP\_TAG=$(VERSION)  
APP\_IMG=$(APP\_NAME):$(APP\_TAG)  
REMOTE\_IMG:=docker.io/umgccaps/$(APP\_IMG)  
BUILD\_IMG=docker.io/umgccaps/advance-development-factory:latest  
BRANCH=$(shell git rev-parse --abbrev-ref HEAD)  
  
# Maven options  
MAVEN\_OPTS:=-Dversion=$(VERSION)  
  
# Unique ID used for devel Azure deployments  
UUID\_FILENAME:=user.uuid  
UUID:=$(shell cat $(UUID\_FILENAME) 2> /dev/null || (uuidgen | sed s/'-'/''/g | head -c 10 \  
 | tr A-Z a-z > $(UUID\_FILENAME) && cat $(UUID\_FILENAME)))  
  
# Skip test flag  
# make all SKIP\_TESTS=y <- doest not run unit tests  
ifdef SKIP\_TESTS  
 MAVEN\_OPTS:=$(MAVEN\_OPTS) -Dmaven.test.skip=true  
endif  
  
# PHONY  
.PHONY: all image start clean push

Recipies:

##############################################################  
# make image:  
# This recipe create the project-tracker Docker image  
#  
##############################################################  
image: target/$(PROJECT\_APPLICATION\_NAME\_JAR)  
 cp target/$(PROJECT\_APPLICATION\_NAME\_JAR) ./$(PROJECT\_APPLICATION\_NAME\_JAR)  
 docker build -f ./docker/Dockerfile --build-arg VERSION=$(VERSION) \  
 --build-arg PROJECT\_APPLICATION\_NAME\_APP=$(PROJECT\_APPLICATION\_NAME\_JAR) -t $(APP\_IMG) .  
 rm -rf ./$(PROJECT\_APPLICATION\_NAME\_JAR)

##############################################################  
# make start:  
# This recipe start the project-tracker Docker app  
#  
##############################################################  
start:  
 docker run --rm --name $(APP\_NAME) $(APP\_IMG)  
  
##############################################################  
# make clean:  
# This recipe cleans the user workspace  
#  
##############################################################  
clean:  
 @mvn $(MAVEN\_OPTS) clean -f pom.xml  
  
##############################################################  
# make push:  
# This recipe pushes the Docker vlol application to the  
# Azure container registry  
#  
##############################################################  
push:  
 docker tag $(APP\_IMG) $(REMOTE\_IMG)  
 docker push $(REMOTE\_IMG)

Create a Dockerfile

1. Create a new directory called "docker."
2. In the docker directory create a file called “docker-entrypoint.sh”

**#!/usr/bin/env sh**java -jar /usr/src/project-application-name/app.jar

1. In the docker, directory creates a file called "Dockerfile."

FROM ubuntu:20.10  
USER root  
ENV DEBIAN\_FRONTEND noninteractive  
RUN apt-get update && \  
 apt-get --no-install-recommends -y install default-jre openjdk-11-jdk  
   
  
ARG VERSION  
LABEL VERSION=$VERSION  
RUN mkdir /usr/src/project-application-name  
  
ARG PROJECT\_APPLICATION\_NAME\_APP  
COPY PROJECT\_APPLICATION\_NAME\_APP /usr/src/project-application-name/app.jar  
  
  
# configure the entry point  
ADD ./docker/docker-entrypoint.sh /docker-entrypoint.sh  
RUN chmod +x /docker-entrypoint.sh  
  
  
ENTRYPOINT ["/docker-entrypoint.sh"]

Create Azure Files

1. Create a new directory called Azure
2. Create a file called deploy-parameters.json and have it be as follows:

{  
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",  
 "contentVersion": "1.0.0.0",  
 "parameters": {  
 "containerName": {  
 "value": "project-application-name"  
 },  
 "location": {  
 "value": "eastus"  
 },  
 "imageType": {  
 "value": "Public"  
 },  
 "imageName": {  
 "value": "docker.io/umgccaps/project-application-name.app:latest"  
 },  
 "osType": {  
 "value": "Linux"  
 },  
 "numberCpuCores": {  
 "value": "2"  
 },  
 "memory": {  
 "value": "6"  
 },  
 "restartPolicy": {  
 "value": "OnFailure"  
 },  
 "environmentVariables": {  
 "value": [  
 {  
 "name": "Test",  
 "value": "test"  
 }  
 ]  
 },  
 "ipAddressType": {  
 "value": "Public"  
 },  
 "ports": {  
 "value": [  
 {  
 "port": "80",  
 "protocol": "TCP"  
 },  
 {  
 "port": "5000",  
 "protocol": "TCP"  
 }  
 ]  
 },  
 "dnsNameLabel": {  
 "value": "dev-project-application-name"  
 }  
 }  
}

1. Create a deploy-template.json file exactly as such:

{  
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
 "contentVersion": "1.0.0.0",  
 "parameters": {  
 "location": {  
 "type": "string"  
 },  
 "containerName": {  
 "type": "string"  
 },  
 "imageType": {  
 "type": "string",  
 "allowedValues": [  
 "Public",  
 "Private"  
 ]  
 },  
 "imageName": {  
 "type": "string"  
 },  
 "osType": {  
 "type": "string",  
 "allowedValues": [  
 "Linux",  
 "Windows"  
 ]  
 },  
 "numberCpuCores": {  
 "type": "string"  
 },  
 "memory": {  
 "type": "string"  
 },  
 "restartPolicy": {  
 "type": "string",  
 "allowedValues": [  
 "OnFailure",  
 "Always",  
 "Never"  
 ]  
 },  
 "environmentVariables": {  
 "type": "array"  
 },  
 "ipAddressType": {  
 "type": "string"  
 },  
 "ports": {  
 "type": "array"  
 },  
 "dnsNameLabel": {  
 "type": "string"  
 }  
 },

"resources": [  
 {  
 "location": "[parameters('location')]",  
 "name": "[parameters('containerName')]",  
 "type": "Microsoft.ContainerInstance/containerGroups",  
 "apiVersion": "2018-10-01",  
 "properties": {  
 "containers": [  
 {  
 "name": "[parameters('containerName')]",  
 "properties": {  
 "image": "[parameters('imageName')]",  
 "resources": {  
 "requests": {  
 "cpu": "[int(parameters('numberCpuCores'))]",  
 "memoryInGB": "[float(parameters('memory'))]"  
 }  
 },  
 "environmentVariables": "[parameters('environmentVariables')]",  
 "ports": "[parameters('ports')]"  
 }  
 }  
 ],  
 "restartPolicy": "[parameters('restartPolicy')]",  
 "osType": "[parameters('osType')]",  
 "ipAddress": {  
 "type": "[parameters('ipAddressType')]",  
 "ports": "[parameters('ports')]",  
 "dnsNameLabel": "[parameters('dnsNameLabel')]"  
 }  
 },  
 "tags": {}  
 }  
 ]  
}

1. Create a azure-pipelines.yml file as such

# Maven  
# Build your Java project and run tests with Apache Maven.  
# Add steps that analyze code, save build artifacts, deploy, and more:  
# https://docs.microsoft.com/azure/devops/pipelines/languages/java  
  
trigger:  
- master  
  
pool:  
 vmImage: 'ubuntu-20.04'  
  
variables:  
 isMain: $[eq(variables['Build.SourceBranch'], 'refs/heads/master')]  
  
steps:  
- task: CmdLine@2  
 inputs:  
 script: 'make all'   
  
- task: PublishTestResults@2  
 inputs:  
 testResultsFormat: 'JUnit'  
 testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml'  
 failTaskOnFailedTests: true  
  
- task: CmdLine@2  
 inputs:  
 script: 'make sonar'  
  
- task: CmdLine@2  
 inputs:  
 script: 'make image'  
  
- task: CmdLine@2  
 condition: and(succeeded(), eq(variables.isMain, true))  
 inputs:  
 script: 'docker login $(Host) -u $(Username) -p $(Password)'  
   
- task: CmdLine@2  
 condition: and(succeeded(), eq(variables.isMain, true))  
 inputs:  
 script: 'make push'

## Testing the ADF Integration on Your Local

In your terminal, perform the following steps to verify your work:

1. Run `make all`

[INFO] Replacing main artifact with repackaged archive  
[INFO] --------------------------------------------------------------------  
[INFO] BUILD SUCCESS  
[INFO] --------------------------------------------------------------------  
[INFO] Total time: 53.848 s  
[INFO] Finished at: 2020-10-17T21:16:48Z  
[INFO] --------------------------------------------------------------------

1. Run `make image`

Successfully built 2cf42ebd33ef  
Successfully tagged municipal-permit-chabot.app:1.0.19  
rm -rf ./MunicipalPermitChabot-1.0.0.jar

1. Run `make start`

Will run the application within the docker container

If you do not see the success message debugging will need to be done to determine if either there is an issue with your makefile, docker, or the pom file for the project.

## Pushing your Branch to GitHub for a Pull Request

1. Create a commit:
   1. git add .
   2. git commit -m 'integrate adf'
   3. git push origin adf-integration
2. This will give you a URL. Go to that URL and add relevant users to the pull request. Before merging, you will want to consult the **DevSecOps RunBook** and learn how to create and run a pipeline.
3. Once your pull request is approved, and you've configured Azure to run a pipeline on that repo, and the pipeline passes, you can merge your work into master

# 5. Integrating Tools

For every tool, you will want to create a branch off of master to integrate the tool with.

## Spotless

Spotless is a tool for verifying and fixing code syntax style issues in projects.

1. In the pom file for the project, add the following plugin:

<plugin>  
 <groupId>com.diffplug.spotless</groupId>  
 <artifactId>spotless-maven-plugin</artifactId>  
 <version>2.4.0</version>  
 <configuration>  
 <java>  
 <includes>  
 <include>src/main/java/\*\*/\*.java</include>  
 <include>src/test/java/\*\*/\*.java</include>  
 </includes>  
  
 <importOrder />  
  
 <removeUnusedImports />  
  
 <googleJavaFormat>  
 <version>1.9</version>  
 <style>GOOGLE</style>  
 </googleJavaFormat>  
 </java>  
 </configuration>  
</plugin>

1. Test it out by running
   1. mvn spotless:check
      1. Will report any style errors
   2. mvn spotless apply
      1. Will automatically fix any style errors

## SonarQube

The plan is for code quality and security will be enforced with [SonarQube](https://www.sonarqube.org/). SonarQube makes use of static and dynamic code analysis to verify the following:

* Code Reliability
* Code Security
* Code Maintainability

Our SonarQube implementation is hosted on sonarcloud.io at the following address:

<https://sonarcloud.io/organizations/umgc-capstone/projects>

Integrate SonarQube to a project

1. In the pom file, add the following:
   1. Properties

<properties>  
 <sonar.projectKey>umgccapstone\_umgc.chatbox.3</sonar.projectKey>  
 <sonar.organization>umgc-capstone</sonar.organization>  
 <sonar.host.url>https://sonarcloud.io</sonar.host.url>  
</properties>

* 1. Plugins

<pluginManagement>  
 <plugins>  
 <plugin>  
 <groupId>org.sonarsource.scanner.maven</groupId>  
 <artifactId>sonar-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</pluginManagement>

1. Add the following to the Makefile
   1. Add spotless to the list of .PHONY

# PHONY  
.PHONY: all image start clean push sonar

* 1. Add this recipe

sonar:  
 docker run -v $(PWD)/:/repo --entrypoint '/bin/bash' $(BUILD\_IMG) \  
 -c 'cd /repo && mvn verify sonar:sonar -Dsonar.branch.name=$(BRANCH)'

1. Test it out by running `make sonar` and navigating to the URL presented once it finished
2. Add it to the Azure Pipeline by adding the following to your azure-pipepline.yml

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