**MemorEz DevSecOps**

**Deployment and Operations Guide (Runbook)**

**Version 1.0**

DevSecOps Team

University of Maryland Global Campus

Spring 2022

SWEN 670

Prepared by Andrew Nicolette and Robert Wren

For Approval by Dr. Mir Assadullah

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author(s)** |
| 01/31/2022 | 1.0 | Initial Deployment and Operations Guide (Runbook) | Andrew Nicolette Robert Wren |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[**1** **Introduction** 4](#_Toc94856181)

[**1.1** **Purpose** 4](#_Toc94856182)

[**1.2** **System Components** 4](#_Toc94856183)

[**1.3** **Definitions** 4](#_Toc94856184)

[**1.4** **References** 4](#_Toc94856185)

[**2** **Development Team Workflow** 5](#_Toc94856186)

[**2.1** **Branch Organization Structure** 5](#_Toc94856187)

[**2.2** **GitHub Client Installation** 6](#_Toc94856188)

[**2.3** **Writing Code** 6](#_Toc94856189)

[**2.3.1** **Creating a Feature Branch** 7](#_Toc94856190)

[**2.3.2** **Committing Changes to a Feature Branch** 7](#_Toc94856191)

[**2.3.3** **Creating a Pull Request** 8](#_Toc94856192)

[**2.3.4** **Addressing a Pull Request** 9](#_Toc94856193)

[**2.3.5** **Submitting Merge Request to Master Branch** 10](#_Toc94856194)

[**2.3.6** **Flow Chart Example** 10](#_Toc94856195)

[**2.3.7** **Helpful Links** 11](#_Toc94856196)

[**3** **DevSecOps GitHub Setup** 11](#_Toc94856197)

[**3.1** **GitHub Teams** 11](#_Toc94856198)

[**3.1.1** **Prerequisites** 11](#_Toc94856199)

[**3.1.2** **Creating a GitHub Team** 11](#_Toc94856200)

[**3.1.3** **Adding Members to a Team** 12](#_Toc94856201)

[**3.2** **GitHub Repositories** 14](#_Toc94856202)

[**3.2.1** **Creating a GitHub Repository** 14](#_Toc94856203)

[**3.2.2** **Assign Repository to Team** 14](#_Toc94856204)

[**3.2.3** **Creating a CODEOWNERS, .gitattributes, and .gitignore files** 15](#_Toc94856205)

[**3.2.4** **Add Branch Protection Rules** 16](#_Toc94856206)

[**3.3** **GitHub Actions** 18](#_Toc94856207)

[**3.3.1** **Creating a Workflow** 18](#_Toc94856208)

[**4** **Troubleshooting** 19](#_Toc94856209)

# **Introduction**

## **Purpose**

This document serves as a compilation of procedures that will be routinely used by developers and maintainers of the projects used in this semester of SWEN 670 Capstone. It two sections. The first section is called Development Team Workflow, which outlines and describes procedures given to the development team for updating and utilizing the Version Control Repository. The second section is more utilized for the DevSecOps team which walks through the instantiation of the repository and any prerequisites needed to do so.

## **System Components**

This document utilizes GitHub exclusively to handle version control, repository management, and automatic pipelines. Version control and repository management is handled using the Git system. The pipelines are formed using GitHub Actions, which allows a developer to define a set of actions that should occur in response to a repository event.

## **Definitions**

* **Repository:** The entirety of a project and the project’s history. This is normally stored in a “.git/” folder on your local machine.
* **Branch:** A version of a repository that represents an independent line of development.
* **Remote:** The remote repository is a Git repository that’s hosted on the internet instead of on your local machine. It is through the remote that developers can share their code and collaborate.
* **Add:** Selects a file or multiple files and moves them to a staging area to be included in future commits.
* **Commit:** An action that saves staged changes to the local repository. Files can be removed from the staging area if they are not relevant to a commit. Each commit is a sort of snapshot of the state of the staged files that a developer can go back to at any time.
* **Push:** An action that copies the commit history on a local repository to the remote repository.
* **Pull Request:** A request for reviewers to look at the changes a developer has made and ensure that they are functional and understandable before merging the changes to a base branch.
* **Merge:** An action that combines the contents of one branch into another. In most cases, it will be combining changes from a feature branch into the developer branch, or changes from the developer branch into the main branch.

## **References**

Worku, A., Setiawan, M., Johnson, R. (2021, September 10). Captsone Project Management System (CaPPMS) & DevSecOps Infrastructure: Deployment and Operation Guide (Runbook). University of Maryland Global Campus.

Baker, J. (2021, March 31). *Feature branching using feature flags - dzone devops*. dzone.com. Retrieved January 13, 2022, from https://dzone.com/articles/feature-branching-using-feature-flags-1

# **Development Team Workflow**

## **Branch Organization Structure**

The GitHub repository will consist of 3 major types of branches to keep new developments and features organized while maintaining the functionality of each release.

**Main Branch**: This branch contains operational code that can be released as a working version of the project. It does not have to include all the features that have been planned, and the project lead can decide when the project is in a good state to be merged into this branch. Generally, if a demo of the project is requested, the latest version in the Main Branch will be shown. In our use case, the repository admins/DSO team are the only personnel that can merge from other branches into the Main Branch.

**Developer Branch**: This branch generally contains operational code and serves as a staging environment in which any new features will be tested to establish their effects before being allowed to merge into the Main Branch. Code merged into the Developer Branch should work as intended in isolation without errors. As such code merged into the Developer Branch should be reviewed beforehand. The reviewer ideally should be the lead developers of each team and should walk through the changes with the original developer before merging.

**Feature Branch**: This is a type of branch that developers will create to address a specific feature or set of features of the project. This can entail adding new functionality or fixing existing faulty functionality. Generally, the name of the Feature Branch will reflect the goal of the task. When the developer is finished making the desired changes, they will initiate a pull request, in which a lead developer of the project or team will review the code to be merged to enforce programming practices and limit the number of bugs introduced into the project. The DSO team is enforcing feature branches to have a common naming standard. The naming standard will always start with “feature/” and then a short description of what the feature is doing. For example, this would be a valid feature branch name – “feature/fixing-microphone”.

## **GitHub Client Installation**

Navigate to <https://desktop.github.com/> and download the desktop client for your operating system.

Once you’ve installed the desktop client, select “Clone a repository from the Internet”. If you’ve not yet signed in to your GitHub account through the client, do so first.

Graphical user interface, text, application

Description automatically generated

*Figure 1*. GitHub Client Sign-in Page

Once you’ve signed in, find the repository for your project under the “umgc” organization, then select it. The string in the “Local path” text field will be where your local repository is saved. If you are satisfied with this location, select the “Clone” button.

Graphical user interface, text, application, email

Description automatically generated

*Figure 2*. Clone Repository Page

## **Writing Code**

To get started with development, begin by creating a **Feature Branch**. This branch should adhere to the naming standards described above. Make sure the name is descriptive enough that you know what you were working on in this branch. This becomes especially helpful when you are working on multiple branches simultaneously and need to switch between them depending on changing priorities.

Graphical user interface, text, application

Description automatically generated  
*Figure 3*. Creating a Branch

### **Creating a Feature Branch**

Generally, you’ll want to create feature branches from based on the **Developer Branch**. This is because the **Developer Branch** is the staging environment in which any new features will be tested before being merged into the **Main Branch**, so the new feature should work with the **Developer Branch**. It should be set as the default branch, but if it is not, you’ll have to select it manually.

Graphical user interface, text, application

Description automatically generated

*Figure 4*. Using a Base Branch

After creating the **Feature Branch**, you’ll have to publish it to the **remote** before you can share it with the team. Once you do so, you can add your changes to the project and initiate a pull request, in which a maintainer of the repository will review your changes before merging it into the **Developer Branch.**

### **Committing Changes to a Feature Branch**

Once you make changes, you can **commit** them to the **Feature Branch**, essentially creating a snapshot of the state of the repository that you can reference in the future, should you make unwanted changes and wish to return to a previous state. Keep in mind that **committing** only saves changes to your local repository. It is important that you do not commit any executable files or build artifacts, as they should be able to be built using the code in the repository. In addition, ensure that there is no private information included, such as a private SSH key.

Graphical user interface, text, application, chat or text message

Description automatically generated

*Figure 5*. Committing Changes

In order to save your changes to the **remote**, you’ll have to **push** your **commits** to the origin remote. By doing so, your changes are shared with the rest of the team. In addition, if you happen to lose your data on your local computer, you can still access changes you’ve **pushed** to the **remote**.

Graphical user interface, text, application

Description automatically generated

*Figure 6*. Pushing Changes to the Remote

### **Creating a Pull Request**

Once you have merged the **Developer Branch** into your own branch with a **merge commit,** you can attempt to merge your **Feature Branch** into the **Developer Branch.**

Graphical user interface, text, application, email

Description automatically generated

*Figure 7*. Requesting a Merge into Developer Branch

Create a **pull request** and add any comments about the merge that the reviewer might want to know. For example, if you used a design pattern, it might help to let the reviewer know. Of course, you don’t need to leave a comment at all if you don’t think it is necessary. It is through this interface that your reviewers can interact with you.

Graphical user interface, text, application, email

Description automatically generated

*Figure 8*. Pull Request Form

### **Addressing a Pull Request**

Once a **pull request** has been initiated, reviewers or the lead developer on the team can see the changed files by selecting the “Files changed” tab. They may also comment on specific lines of code by selecting the “+” after each line number. When a **pull request** has been fully reviewed, the reviewer may select the “Review Changes” button and approve the request, or deny it, asking the developer to make suggested changes first.

Graphical user interface, text, application, email

Description automatically generated

*Figure 9*. Reviewing a Pull Request

Once the code has been adequately reviewed and approved by the reviewers, they may **merge** the **Feature Branch** into the base branch, which should be the **Developer Branch**. The method of **merging** is up to the repository maintainer, but assuming that the code has been rigorously tested and is in a good state, it is neater to do a “Squash and merge” so that there aren’t too many messages in the repository’s commit history.

Graphical user interface, text, application, email

Description automatically generated

*Figure 10*. Merging a Feature Branch into Developer Branch

With that, the new feature has been **merged** into the development branch.

### **Submitting Merge Request to Master Branch**

Once the lead developer on the team feels the development branch is in a good state, a **pull request** can be initiated to merge the code from the development branch to the master/main branch. The administrators of the repository will work with the lead developers to address if there are any conflicts and will validate that the CI/CD pipeline checks have all passed and been approved. Once validated and triaged, the administrator will approve the merge request and allow the merge to master.

### **Flow Chart Example**

Chart, line chart

Description automatically generated

*Figure 11*. Merging a Feature Branch into Developer Branch

Note. From Feature branching using feature flags [Photograph], by J. Baker, 2021, (https://dzone.com/articles/feature-branching-using-feature-flags-1).

### **Helpful Links**

<https://youtu.be/d-6ATlUn_zc> - This semester we created a video that will walk through common GitHub commands, what they mean, and how to use them. We also went through a workflow on how to create feature branches, submit merge requests, and triage conflicts.

<https://cbea.ms/git-commit/> - This article goes over how to write good git commit messages and why they are important.

# **DevSecOps GitHub Setup**

## **GitHub Teams**

Teams are an easy way to organize developers and perform actions on them collectively, such as giving access to a repository, or requiring certain permissions before performing a pull request. Teams are a subsection of an organization, which is a shared account that allows collaborators to work on many projects at once. It’s similar to a team, but on a larger scale.

### **Prerequisites**

Prior to creating teams in GitHub for the project, there are permissions and tasks you will need to accomplish first. Here are the associated tasks the team will need to do first before moving on to section 3.1.2.

* For each member of the DevSecOps team, have the mentor/organization owner add them to the organization UMGC Capstone Course.
* For each member of the DevSecOps team, have the mentor/organization owner change the account role in organization from Member to Owner. This will allow you to create the team in the organization and start inviting people.
* For each member of the DevSecOps team, have the mentor/organization owner add each member to the GitHub DevOps group within the organization.

### **Creating a GitHub Team**

To create a GitHub team, you’ll need to be given the administrative right to do so. Once you’ve been given administrative rights, navigate to your organization’s GitHub page, then select the “Teams” tab.

Graphical user interface, text, application

Description automatically generated

*Figure 12*. UMGC Capstone Course GitHub Overview Tab

From there, select the “New Team” button to initiate the creation of a team. It’s highly recommended to create the parent team first before creating the development teams. By design, the parent teams have been named the semester and year (Spring 2022).

Graphical user interface, text, application

Description automatically generated

*Figure 13*. UMGC Capstone Course GitHub Teams Page

Once the parent team is created, you can create the development teams. Similarly, you want to select “New Team” and then set the team’s name and add an optional description for the team. Now instead of leaving the parent team as its default value, you want to click the dropdown and click on the corresponding parent team you created above. This will allow for easier organization and represent each of the different teams within the class.

Graphical user interface, text, application, email

Description automatically generated

*Figure 14*. Create Team Form

### **Adding Members to a Team**

To add a people to the team, navigate to the team page, then select the “Members” tab. From there, select the “Add a member” button. This will invite each member to the organization and add them to the corresponding team.

Graphical user interface, application

Description automatically generated

*Figure 15*. GitHub Team Members List

You can type the desired person’s username, full name, or email address into the search bar to locate them, then select “Invite” and confirm the invite. It’s recommended that the product team leads should compile their developers GitHub username’s so the DevSecOps team can start adding them to the repository. Each member will receive an email to their account they setup GitHub with and need to click join organization/team.

Graphical user interface, text, application

Description automatically generated

*Figure 16*. Add Member to GitHub Team

## **GitHub Repositories**

A repository is a record of a project and the project’s history. Generally, the main location for a repository is on the remote, which is located on GitHub’s servers.

### **Creating a GitHub Repository**

To create a GitHub team, you’ll need to be given the administrative right to do so. Once you’ve been given administrative rights, navigate to your organization’s GitHub page, then select the “Repositories” tab. From there, select the “New repository” button to initiate the creation of a repository.

Graphical user interface, text, application, email

Description automatically generated

*Figure 17*. UMGC Capstone Course GitHub Repositories

Ensure that the owner is your organization, then enter a name for the repository, and optionally, a description. It is helpful to also set the visibility to Public for easy access, but you can always change this later. When you have finished the initial repository settings, select the “Create repository” button.

Graphical user interface, text, application, email

Description automatically generated

*Figure 18*. GitHub Create Repository

### **Assign Repository to Team**

To assign teams to a repository, you’ll need administrative rights to the repository, which you probably have, considering you just made the repository. Navigate to the repository’s GitHub page, then select the “Settings” tab. From there, select the “Manage access” menu item.

Graphical user interface, application, Teams

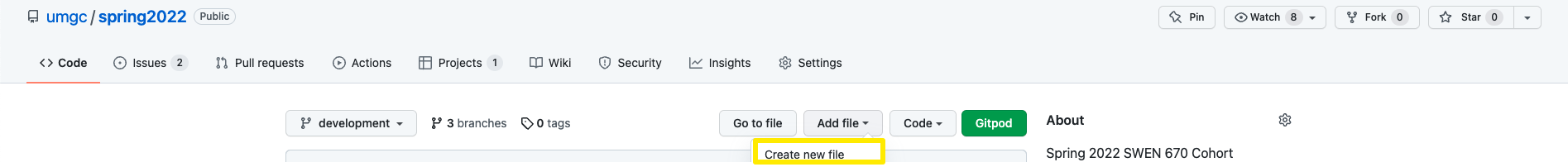
Description automatically generated

*Figure 19*. GitHub Add Repository Accesses

You can invite teams by selecting the “Add teams or people” button and searching by team name, then adding them to the repository. From the “Manage access” page, you can also change access controls for the teams on this repository. Generally, for development teams, the access level should be set to “Write”. A client team can be set to “Read”. The Development Security Operations team should be set to “Admin”.

### **Creating a CODEOWNERS, .gitattributes, and .gitignore files**

To create a new file, click “Add File” dropdown and select “Create a new file”. In the root directory of the project, it’s optional to add three files.



*Figure 20*. Create New File in Repository

**CODEOWNERS:** This file outlines who owns certain paths or files in a repository. Each branch can have different CODEOWNERS files and can be used in conjunction with branch protection rules to get approval from certain individuals or a team of individuals before merging code into that branch.For example, if there was a line in the CODEOWNERS file like this “/dso @drewnicolette @robertwren”, this would portray that drewnicolette and robertwren would need to approve any updates, inserts, or deletes within the /dso directory.

Graphical user interface, text, application

Description automatically generated

*Figure 21*. CODEOWNERS Sample File

**.gitignore:** This file tells Git which files to ignore when committing them to the remote repository. For example, you might want to ignore build artifacts being uploaded to Git because of their size. In that case, you can create a .gitignore file and add “build/” which will ignore everything inside the build directory.

Graphical user interface, application

Description automatically generated

*Figure 22*. Gitignore Sample File

**.gitattributes:** This file tells Git what attributes you want to provide to a given path name. For example, if you want the CODEOWNERS file being unique in different branches and/or have specific attributes when being merged you can specify “CODEOWNERS merge=ours” which will keep the current version of the file when a merge request has been instantiated. This file won’t be overwritten when merging between branches and will always stay unique to that branch.

Teams

Description automatically generated with low confidence

*Figure 23*. Gitattributes Sample File

### **Add Branch Protection Rules**

Adding branch protection rules will easily allow you as the admin to have more fine grain control over each branch within the repository. For the main and development branches, adding this can disable force pushing, prevent branches from being deleted, and optionally require status checks before merging. To create branch protection rules, you need to be an admin within the repository. If you have admin access already, go to the repository home page and click “Settings”. On the panel to left, click “Branches” and then click “Add rule”.

Graphical user interface, text, application, email

Description automatically generated

*Figure 24*. Creating Branch Protection Rules

You can also integrate the CODEOWNERS file mentioned in section 3.2.3, to schedule automatic approvals when a developer submits a merge request to the protected branch. To do so, specify the name pattern of the branch and then check “Require a pull request before merging”, “Require Approvals”, and “Require review from Code Owners” then click “Save”.

Graphical user interface, text, application, email

Description automatically generated

*Figure 25*. Configuring Branch Protection Rules

Depending on the name pattern of the branch, you might need to add another rule for another branch within your repository.

## **GitHub Actions**

GitHub Actions allows developers to automate their workflows in response to certain GitHub events, such as push or pull request.

### **Creating a Workflow**

To get started, navigate to the repository, and select the “Actions” tab, then select the “set up a workflow yourself” button.

Graphical user interface, text, application

Description automatically generated

*Figure 26*. UMGC Capstone Course GitHub Actions

GitHub will create a new YAML file in the .github/workflows directory with an explanation of common key words. For this project, we want a code analyzer, a project builder, and a unit test to be executed. These will be added to the “jobs:” section.

**Table 1**

*Notable Job Steps*

|  |  |
| --- | --- |
| **Job Step** | **Description** |
| actions/checkout@v2 | Checks out the repository so the workflow can access it |
| subosito/flutter-action@v1 | Sets up a Flutter environment |
| ValentinVignal/action-dart-analyze@v0.11 | Analyzes dart code using “dart analyze” and “dart format” |
| actions/upload-artifact@v2 | Creates and uploads an artifact to the specified location within the repository |
| maierj/fastlane-action@v1.4.0 | Creates and deploys build artifacts using lanes to publish to both Google Play and App Store |

An established working pipeline example can be seen in the “summer2021.amazing” and “fall2021” repository in the “umgc” organization.

# **Troubleshooting**

**Merge Conflicts:** When considering a pull request, sometimes two developers will make edits to the same file, or perhaps one developer has deleted a file that the other has made changes to. In these cases, you’ll have to attempt to “Resolve conflicts” by picking which of the two changes should remain in the branch.

Graphical user interface, text, application, email, Teams

Description automatically generated

*Figure 27*. Merge Conflicts in GitHub

The web interface has a way to easily make changes. The format for merge conflicts is generally as follows:  
<<<<<<<<<<< <first branch name>

<Conflicting code existing in first branch>

===========

<Conflicting code existing in second branch>

>>>>>>>>>>>> <second branch name>

You’ll have to delete all the markers and dividers, leaving behind only the desired code.

Graphical user interface, text, application, email

Description automatically generated

*Figure 28*. Resolving Merge Conflicts