# SWEN902 – Lab 3

## Introduction:

This lab is focuses on implementing a continuous integration pipeline that when code is committed to a version control system it will build, test, package and deploy the code. It will describe the use of a continuous integration tool and any supporting software required to aid this tool to meet the objectives. The chosen tool for the solution and one other tool will be contrasted in the conclusion. The tools used will be utilized in practice for the lab and the conclusion will be a high-level overview of the system with personal reflections on the value it delivers in terms of a DevOps solution as well as a reflection on continuous integration as part of the DevOps process.

## Aims/Objectives:

* Research and explore available solutions for continuous integration in a DevOps pipeline
* Choosing a suitable tool to achieve the objectives of this lab
* Demonstrate in practice how the tool achieves this
* Exploring the concept of continuous integration as a development tool and process in the DevOps culture.
* Gain a better understanding of continuous integration and the tools available and the part they play in delivering high quality software.
* Draw a conclusion of this lab and reflect on the value of the tools and any learnings achieved during the research
* Gain a better personal understanding of the tools and a deeper knowledge of how it integrates into the software development lifecycle

## Method:

* Researched continuous integration tools that can be used in a pipeline and how they can achieve the objectives.
* Decided on a language and framework that will be used to provide a solution that can be used to demonstrate the CICD pipeline.
* Chose one tool that will be implemented to demonstrate continuous integration and another tool that can be used as a comparison throughout the process.
* Created a repository to host the solution.
* Created a C# ASP.Net Web App using the default Visual Studio 2022 template targeting .Net 7
* Followed Microsoft Learn tutorial cited in conclusion to build up a working API to be used to demonstrate continuous integration.
* Added xUnit test project to solution to demonstrate unit testing in the app
* Pushed all code to GitHub so a workflow could be configured.
* Used the GitHub Action interface to create a .Net workflow to build and test the application.
* Further changes to the workflow were required to satisfy all the requirements for this lab so the default pipeline was modified accordingly to include deployment steps.
* The deployment required an endpoint to deploy to so Azure Portal was used to create a subscription, two resource groups and the required resources to deploy an ASP.Net Web App manually through UI.
* Two environments were created in GitHub for development and production
* Secrets were added to each one to allow the app to be published to Azure
* The pipeline code was updated to reflect these changes and add dependencies on the steps so they ran in order
* An approval gate was added on the production environment so this stage would require approval before this stage is run.
* The pipeline was tested and Postman’s desktop app was used to confirm API endpoints were live in both environments.

## Results:

* There are a wide variety of options for both free and paid continuous integration tools
* The most popular tools often offer a PAAS solution with free tiers available for limited use mostly aimed at individuals or open-source organisations and higher tiers for organisations who require more resources.
* Github offers unlimited build time on any public repository with Github Actions.
* Most tools are flexible enough to build any language and will provide tools to run custom scripts for bespoke requirements.
* Most tools integrate with other popular systems and toolsets across through built-in native support or through third party extensions.
* Configuring Github Actions as a CI tool is very easy as popular workflows are included out of the box in the UI.
* Github Actions uses yaml so it is pipeline as code and can be maintained in source control.
* There are multiple triggers available so you can configure a multitude of workflows based on commit type, branch, git event etc.
* Multiple workflows can be triggered in parallel using Github Actions.
* Dependencies on jobs and steps in Github Actions can be used to sequence events in a workflow.
* Environments and secrets can be used to pass in data to different steps in the workflow without exposing any sensitive information.
* Approval gates can be configured by using protection rules on an environment.
* Specific branches can be configured so that a deployment to production could only happen from a release branch that fits a naming pattern etc.
* There is a lot of help online in the form of official documentation and through community forums for Github Actions.

## Conclusion:

This lab was the most interesting as it brings together the previous two labs and extends them to create a more complete system that promotes continuous integration through the use of a CI tool demonstrated by building, testing, packaging and deploying code. To begin the lab, research was conducted on the available CI tools as well as the concept of continuous integration as it was important to have a good grasp on this before trying to implement a solution.   
  
According to the article Use Continuous Integration, (Microsoft, 2022), it describes continuous integration as the process of automatically building and testing code each time a commit to version control happens. It goes on to explain in greater detail the scenarios that can play out when continuous integration is not carried out by a team such as merge conflicts and other issues arising from changes to the main branch since initially branching off. This highlights a few key takeaways because it explains that the process of continuous integration itself is not just adding a tool in the build process but it is a best practice for teams so that they frequently integrate their code into the main branch and test that it works (or integrates) with existing code rather than branching away from the main codebase and working in isolation for extended periods of time. Regardless of the tools available if a developer does not merge their code frequently with the main branch, they will be liable to have to deal with overhead like duplicated workloads, incompatible changes due to other changes introduced since branching off and so on. This is extremely important to understand as the tools will only work when this best practice becomes a team norm.

In regards to the tools available to achieve continuous integration, there are many available tools to integrate across a number of systems. When researching a tool to use to implement CI in I was surprised to find a lot of the more popular software vendors such as Atlassian’s Continuous integration tool, (Rehkopf, 2022), didn’t mention Github Actions but did mention other popular tools such as Bitbucket Pipelines, Jenkins and Azure pipelines to name a few. My chosen source control was Github and I was aware of Github actions so using the official documentation on Github, (Github, 2022), I was able to determine that Github Actions would be a good fit for this particular task as it was freely available to the chosen version control system and would satisfy the requirements for this lab. It was interesting to see that the majority of tools listed could also be incorporated into Github as each one seemed to offer support for multiple code sources including Github as it is a very popular service.

Other considerations when choosing a suitable CI tool included determining which one could be used with the chosen language, which in this case was C# on .Net 7, and ensuring there were tasks available to make the CI as seamless as possible. Github actions provided an out of the box workflow for building .Net apps which was used to generate the initial pipeline and the workflow was triggered on each commit to the main branch so it would only run on main on either a direct commit or via a pull request from a feature branch. To build the .Net app the tutorial documentation for building minimal APIs was used initially, (Microsoft, 2022), but later on a more comprehensive tutorial was used to create a full web API project, (Microsoft, 2022), as issues with the deployment step were discovered that hindered verification of the deployed packages through Postman API testing.

This part of the implementation was interesting to me as it put me in the place of a developer who might be using the system to generate a CI pipeline to verify code being committed to the shared main branch would be built, unit tests ran, artifacts published into a deployable package, package deployed to development, potentially some integration tests ran on the dev environment and finally deploying to production. Using the official documentation on Github, (Github, 2022), it was easy to achieve the first steps for building and unit testing using the default yaml. After some very slight modifications to specify the solution folder the CI pipeline was triggering, building the code, running the unit tests and completing successfully.

The next part to accommodate the deployment of the app required an external service to host the deployed application which in this case was Azure Portal. Azure, being a Microsoft offering, seemed like the most sensible approach with the least overhead as the application was .Net and Github itself is owned by Microsoft so it is very well integrated as it is. The focus here was just to get something created quick so not a lot of time was spent creating infrastructure as code and most of the tasks were done via the UI or through the cloud shell offered in Azure. If there was more time to implement something like this it is possible to integrate these steps into a continuous integration/continuous delivery/continuous deployment pipeline but the idea here was to focus heavily on a continuous integration pipeline with some emphasis on delivery and deployment rather than a full production pipeline which might be used in an organisation. Github made this integration very easy as you can connect Github as a service from Azure using the built-in authorization and it links back secrets to your repo from the newly created resources, much like a service connection or personal access token in another system, and it even generates a workflow for you to use to build and deploy to that resource which appears in Github Actions. I deleted the default pipelines and secrets in the end because there was already a pipeline available and I wanted one pipeline to do build, test, package, deploy dev and deploy production. Using the reference documentation for Github secrets, (Github, 2022), and Github environments, (Gtihub, 2022), I was able to create an environment for both development and production that each contained a secret for the Azure Publish Profile that could be used in each stage of the pipeline to deploy the app. Using the environment, protection rules were able to be set on the environment so that if a deployment stage was triggered for production that it would request a manual approval before executing. This is a very nice feature as it gives some control to the stakeholders so not every change will immediately end up in production. This allows for more extensive manual testing or automated testing to be completed by the team on an environment before signing off on a release if necessary.   
  
On the subject of automated testing, there is potential here to add steps to run automated tests on the development environment after it has been successfully deployed and before moving on to the production deployment stage. It is omitted from this solution as the unit testing was used to satisfy the test ask although research was conducted on the availability of testing workflows for some of the more popular testing tools. According to the official Postman documentation for configuring the Postman CLI for Github Actions, (Postman, 2022), and the documentation on creating APIs in Postman, (Postman, 2022), if you have an upgraded (paid) account with Postman you can link your API testing to your application in Github and it will auto generate the code required to integration this into your Github Actions workflow. I spend a bit of time on this as I wanted to include it in the final solution but cost and time constraints meant it would not be feasible. It can be included though and also it could be implemented for free if you were to create a bespoke task to run the CLI tool as a step and feed in an export of the tests available from Postman desktop but this would require more effort again.

There is a lot of extensibilities available through the use of CI tools in a system and from personal experience and the learning gained in this lab it seems that the best tool to achieve continuous integration is the tool that fits into your current development pipeline or at least the one that will fit in the pipeline you intend to implement. One tool in particular that appeared sources cited earlier online, (Rehkopf, 2022), is CircleCI and it could also have been used here instead of Github Actions. CircleCI is available online and also as a self-hosted solution much the same as Github Actions but it’s pricing model differs slightly beginning with a free tier offering up to 6000 build minutes across a range of OSes and build agents with pricing going up depending on users and additional resources required. One major difference is that CircleCI will work with Git repos hosted in Github or Bitbucket while Github Actions will only work with Github. CircleCI also works off build credits rather than just minutes so there is some more complexity on what tier you might need for an organisation depending on the requirements for their builds. According to a comparison article on Techtarget, (Grasberger, 2022), there are some oddities on CircleCI which may not be apparent until the system is in use such as SSH access into failed builds where you can see the filesystem when errors occur, a very useful feature not in Github Actions, and also it seems to force users to use a single yaml pipeline which may affect reusability of build steps in teams that like to shared common tasks between pipelines as templates. CircleCI would have been an interesting option and seeing as it does integrate with Github it would have been a viable solution here but I felt it was too much overhead as Github Actions could satisfy the same task without the added complexity. If any of the additional features were required then CircleCI might be a better fit.

In summary of this lab, it was very insightful to see how important continuous integration is in achieving a more agile development process. Again, like the systems in lab 1 and 2 it provides the developer quicker feedback on potential issues with their code than more classical methodologies like waterfall where it was common for someone to take a copy of the code and work in isolation for extended periods of time before eventually merging multiple changes back in to the main shared branch. In my own experience, where I can see this still happening with teams using legacy tools and systems, it is plain to see that this is a much better and much more productive approach to development. It can potentially save hours of re-work by ensuring incremental changes can be integrated with the main codebase and builds are successful with tests passing. It can reduce the work required where one developer has removed redundant code from an area that might otherwise require special handling to implement a new feature or refactor an area of the code. The tools are great but mostly importantly for me personally is that the idea of continuous integration is something that is practiced by all members of a team and becomes the norm. I have witnessed cases of individual developers who may be responsible for a particular project, and be the sole developer on it, refuse to integrate their code on a continuous basis making arguments that the code isn’t changing etc. but when a solution like this can be provided to them and the benefits explained and illustrated through the use of these tools it makes it very difficult for further arguments to hold water and it normally is enough to encourage continuous integration as a practice to become the standard.

## References:

## **Microsoft (2022)** Use continuous integration. Available at: https://learn.microsoft.com/en-us/devops/develop/what-is-continuous-integration (Accessed 20 November 2022)

## **Rehkopf, M (2022)** Continuous integration tools. Available at: https://www.atlassian.com/continuous-delivery/continuous-integration/tools (Accessed 20 November 2022)

## **Github (2022)** About continuous integration. Available at: https://docs.github.com/en/actions/automating-builds-and-tests/about-continuous-integration (Accessed 20 November 2022)

## **Anderson, R and Dykstra, T (2022)** Tutorial: Create a minimal web API with ASP.NET Core. Available at: https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-7.0&tabs=visual-studio (Accessed 19 November 2022)

## **Github (2022)** Building and testing .NET. Available at: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-net (Accessed 20 November 2022)

## **Github (2022)** Encrypted Secrets. Available at: https://docs.github.com/en/actions/security-guides/encrypted-secrets (Accessed 20 November 2022)

## **Github (2022)** Using environments for deployment. Available at: https://docs.github.com/en/actions/deployment/targeting-different-environments/using-environments-for-deployment (Accessed 20 November 2022)

## **Postman (2022)** Creating an API. Available at: https://learning.postman.com/docs/designing-and-developing-your-api/creating-an-api/ (Accessed 20 November 2022)

## **Postman (2022)** Github Actions. Available at: https://learning.postman.com/docs/integrations/available-integrations/ci-integrations/github-actions/#configuring-the-postman-cli-for-github-actions (Accessed 20 November 2022)

## **Grasberger, M (2022)** CircleCI vs. GitHub Actions: CI/CD platform comparison Availble at: https://www.techtarget.com/searchsoftwarequality/tip/CircleCI-vs-GitHub-Actions-CI-CD-platform-comparison (Accessed 20 November 2022)

## Appendices:

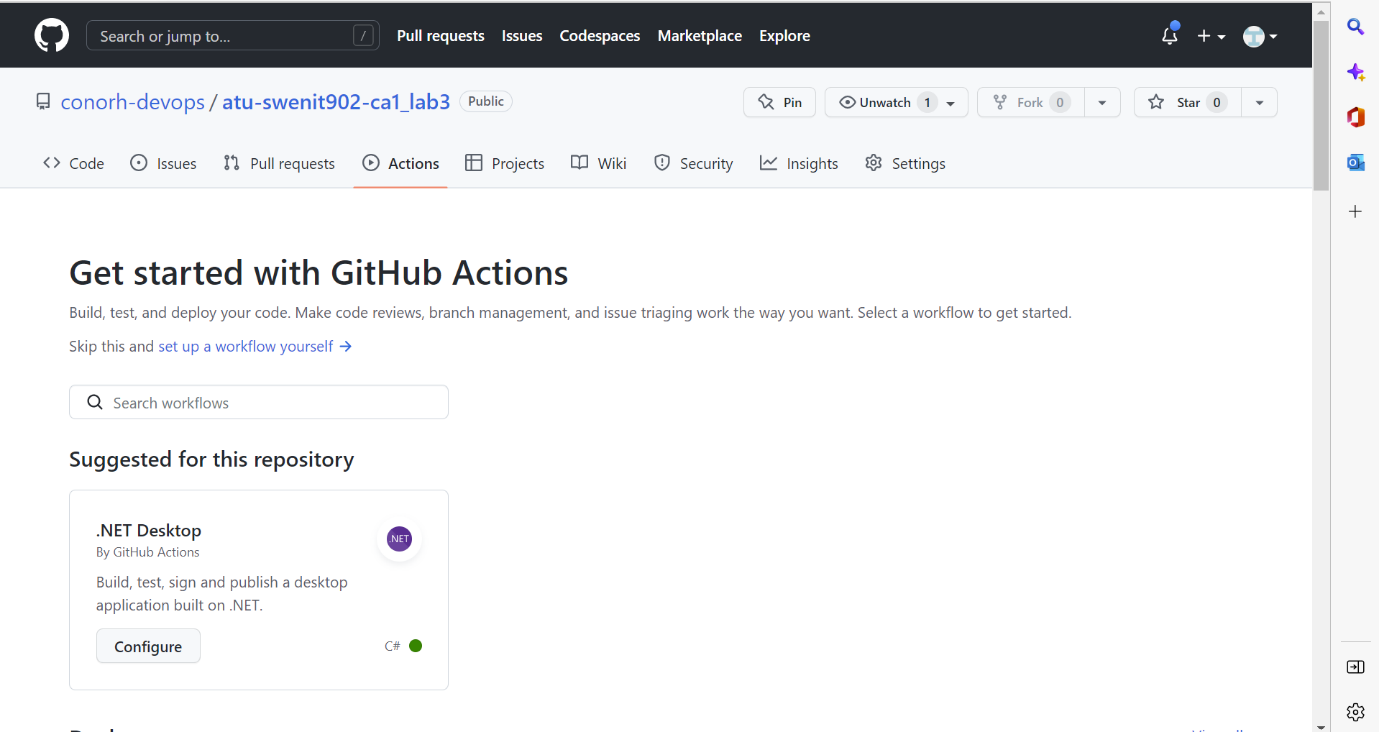


Figure Create Github Actions

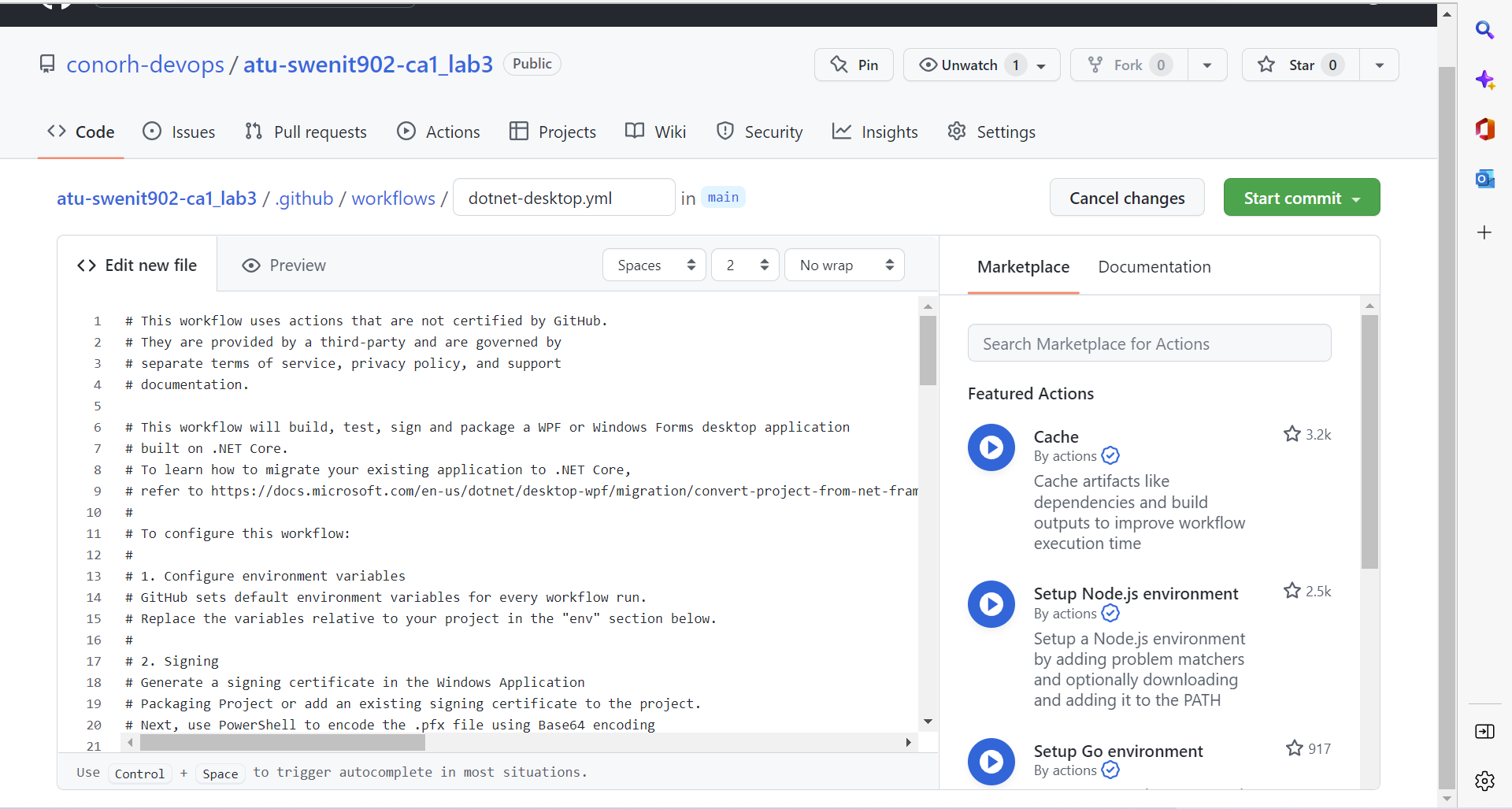


Figure Github Actions Auto Generated Pipeline

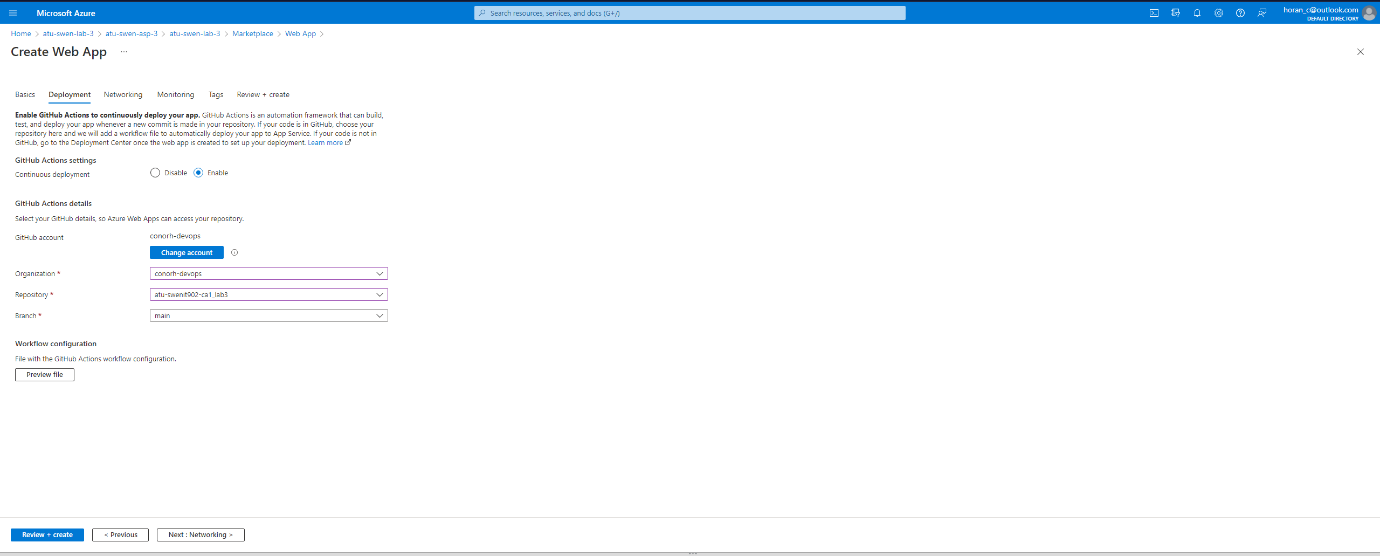


Figure Azure Creating Web App

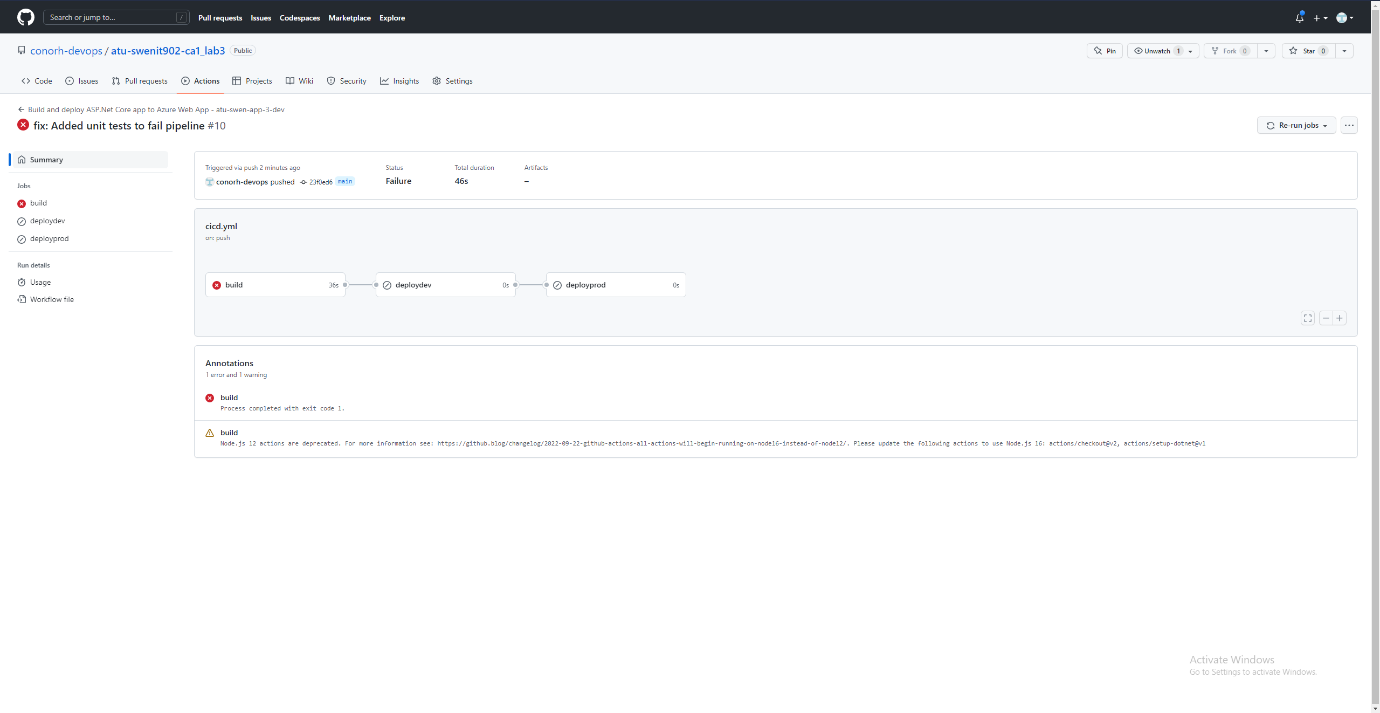


Figure Failed Unit Tests in Pipeline

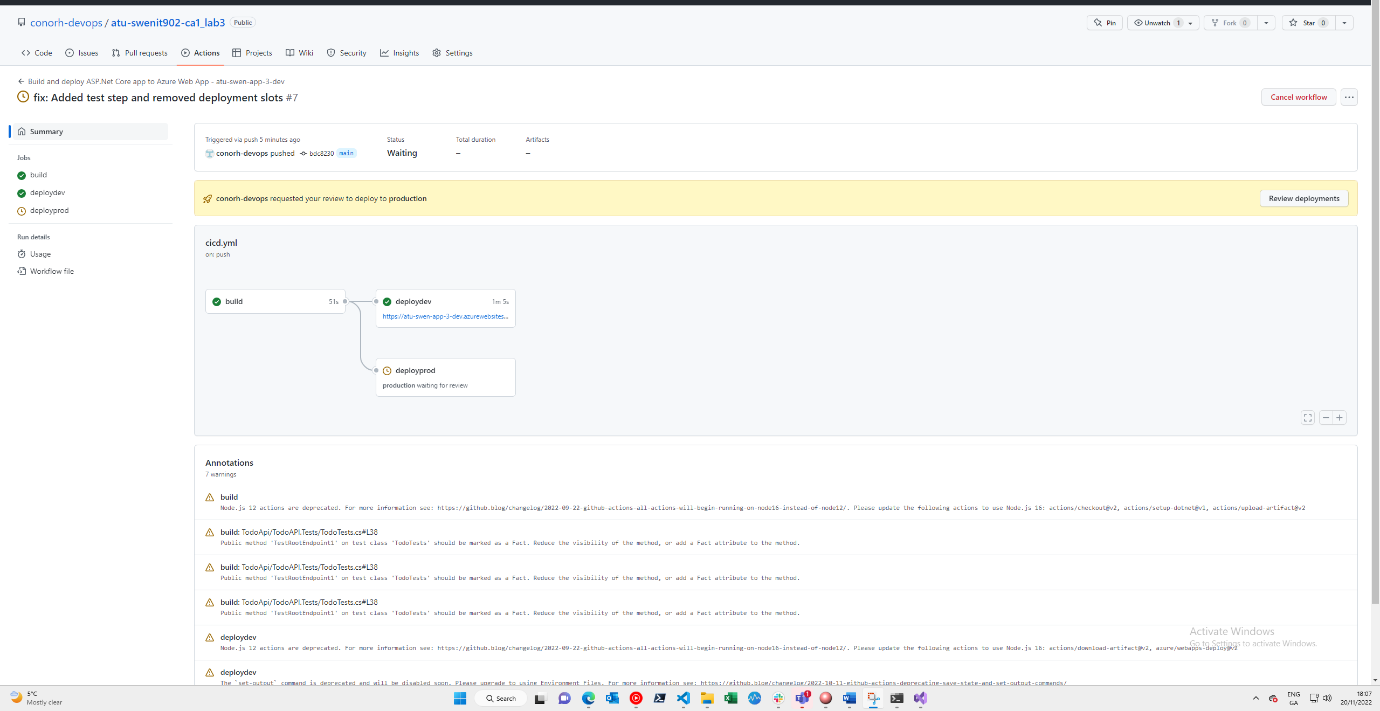


Figure Successful deployment to dev, waiting for approval production

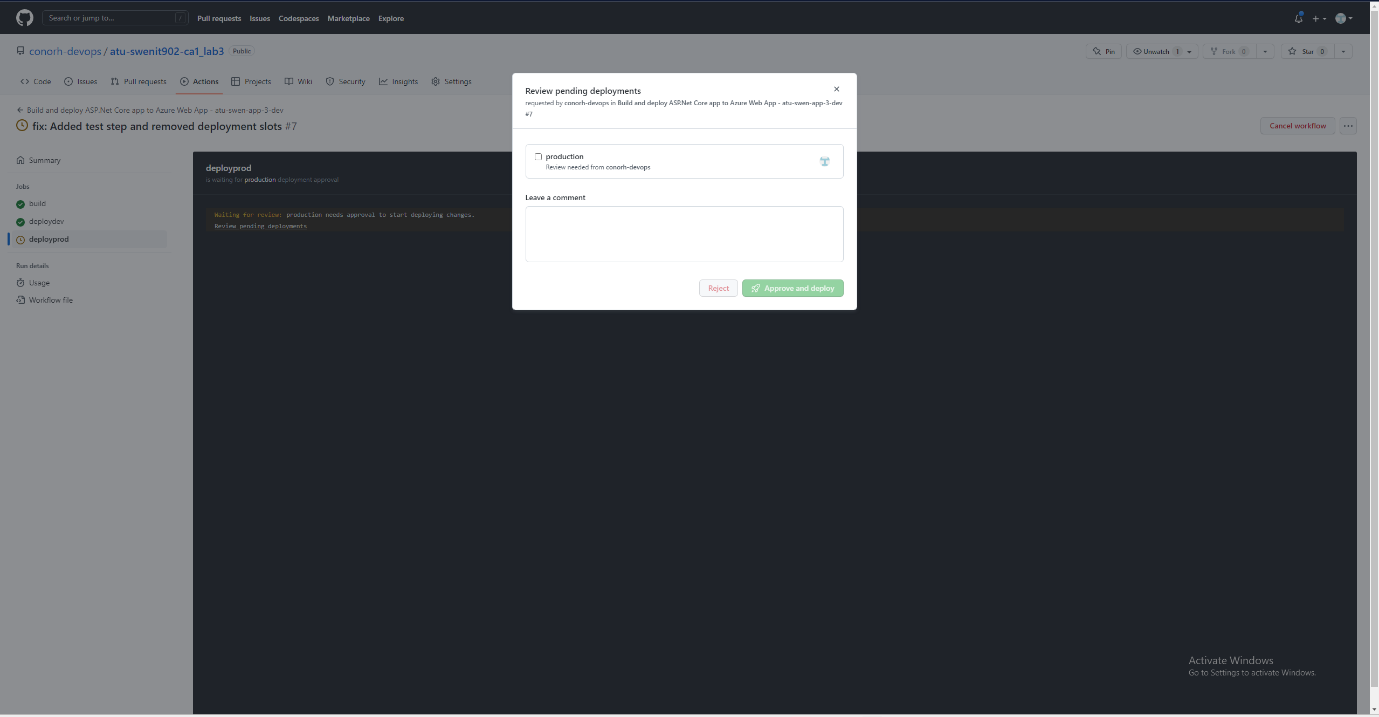


Figure Approving deployment to production

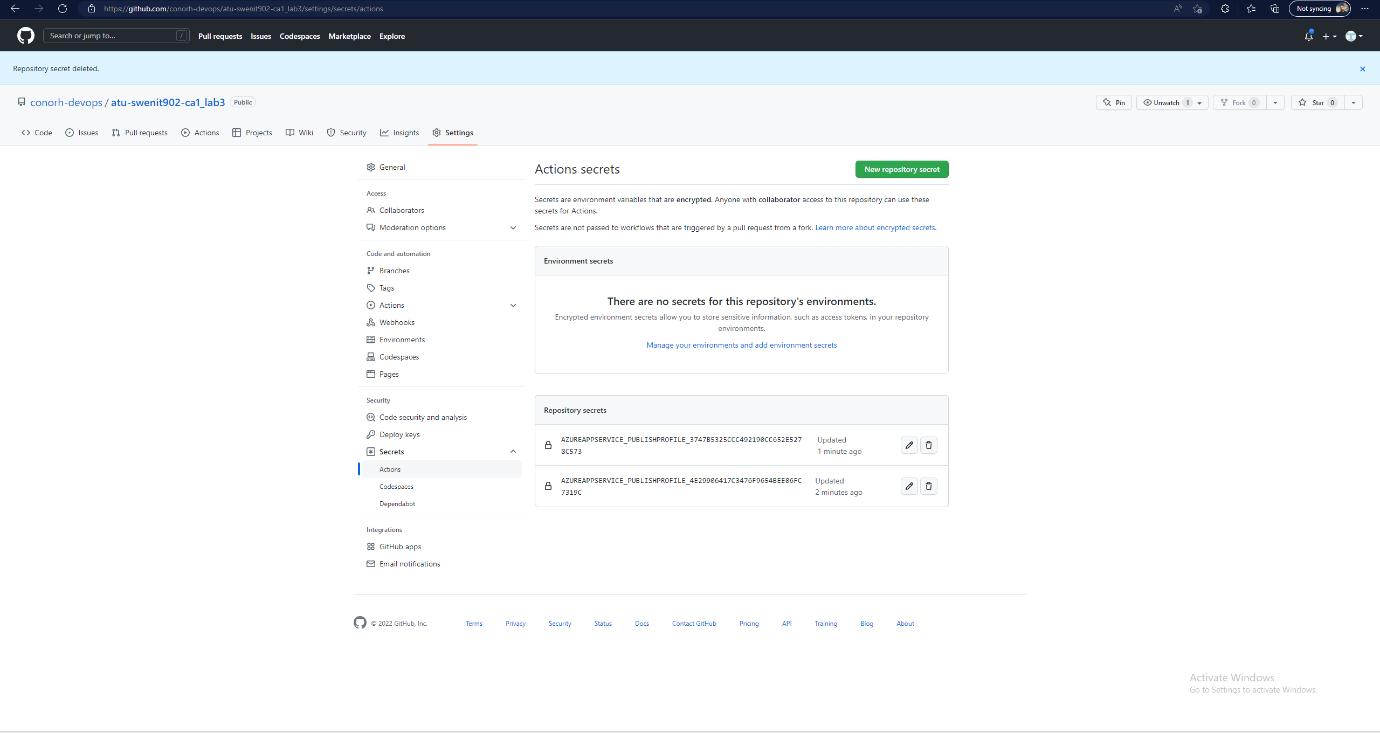


Figure Auto-created secrets from azure

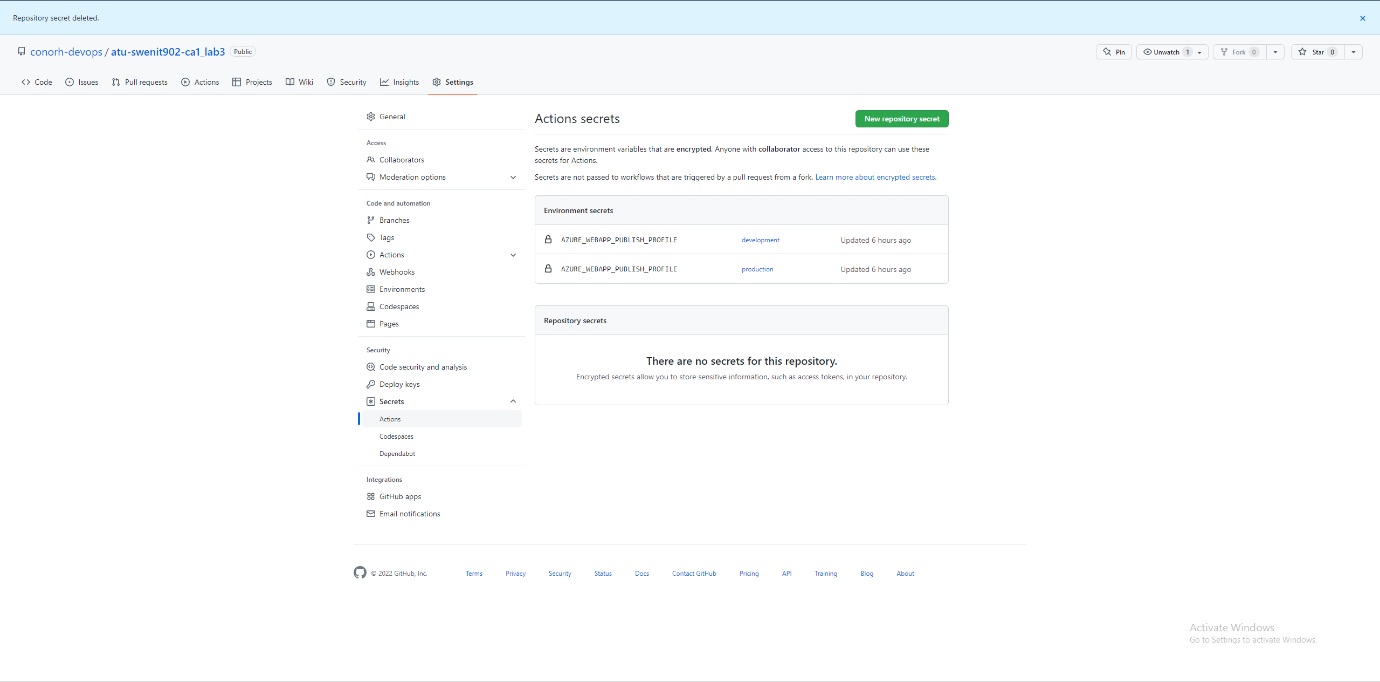


Figure Github Secrets

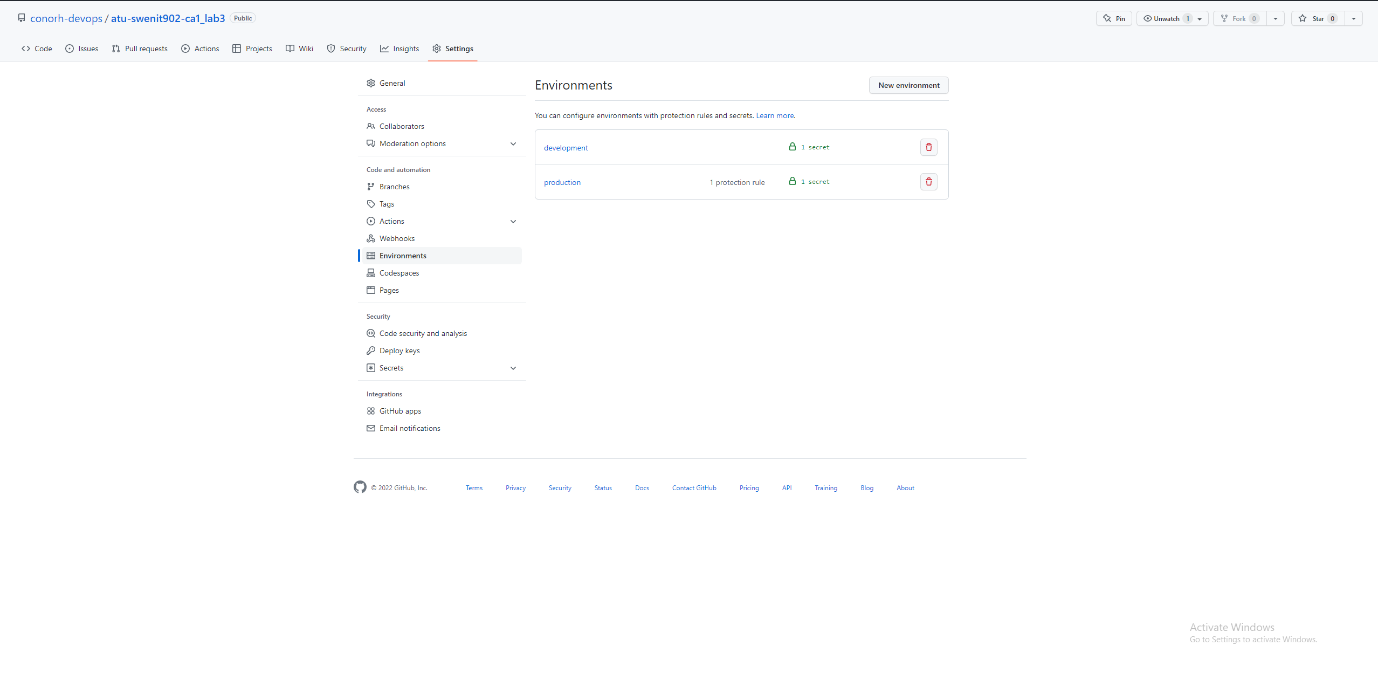


Figure Github Environments