Daniel Allison

danielallison77@gmail.com

n01465744

Assignment 1

CCGC 5502 Automation – Humber College

Table of Contents

[**Section 1: Setup** 3](#_Toc1)

[GitHub Repo 3](#_Toc2)

[Cocogitto 4](#_Toc3)

[backend.tf and providers.tf 5](#_Toc4)

[terraform init 6](#_Toc5)

Table of Figures

[SCREENSHOT 1: terraform validate shows valid configuration 3](#_Toc1)

[SCREENSHOT 2: terraform state list showing deployed resources 3](#_Toc2)

[SCREENSHOT 3: terraform output showing the output variables with resource group names 3](#_Toc3)

[SCREENSHOT 4: terraform state list shows no deployed resources after destruction 3](#_Toc4)

[SCREENSHOT 5: terraform validate shows configuration is valid 3](#_Toc5)

[SCREENSHOT 6: terraform state list | nl showing the number of deployed resources after apply 3](#_Toc6)

[SCREENSHOT 7: terraform output shows the output variables 4](#_Toc7)

[SCREENSHOT 8: terraform state list | nl shows no deployed resources after destruction 4](#_Toc8)

[SCREENSHOT 9: terraform validate shows configuration is valid after linux module added 4](#_Toc9)

[SCREENSHOT 10: terraform state list | nl shows deployed resources after linux module added 4](#_Toc10)

[SCREENSHOT 11: terraform output showing output variables 5](#_Toc11)

[SCREENSHOT 12: terraform state list | nl shows no deployed resources after deletion 5](#_Toc12)

[SCREENSHOT 13: terraform validate shows a valid configuration after windows module is added 5](#_Toc13)

[SCREENSHOT 14: terraform state list shows deployed resources after windows modules is added 6](#_Toc14)

[SCREENSHOT 15: terraform output shows output variables 6](#_Toc15)

[SCREENSHOT 16 terraform state list | nl shows no deployed resources after destruction 7](#_Toc16)

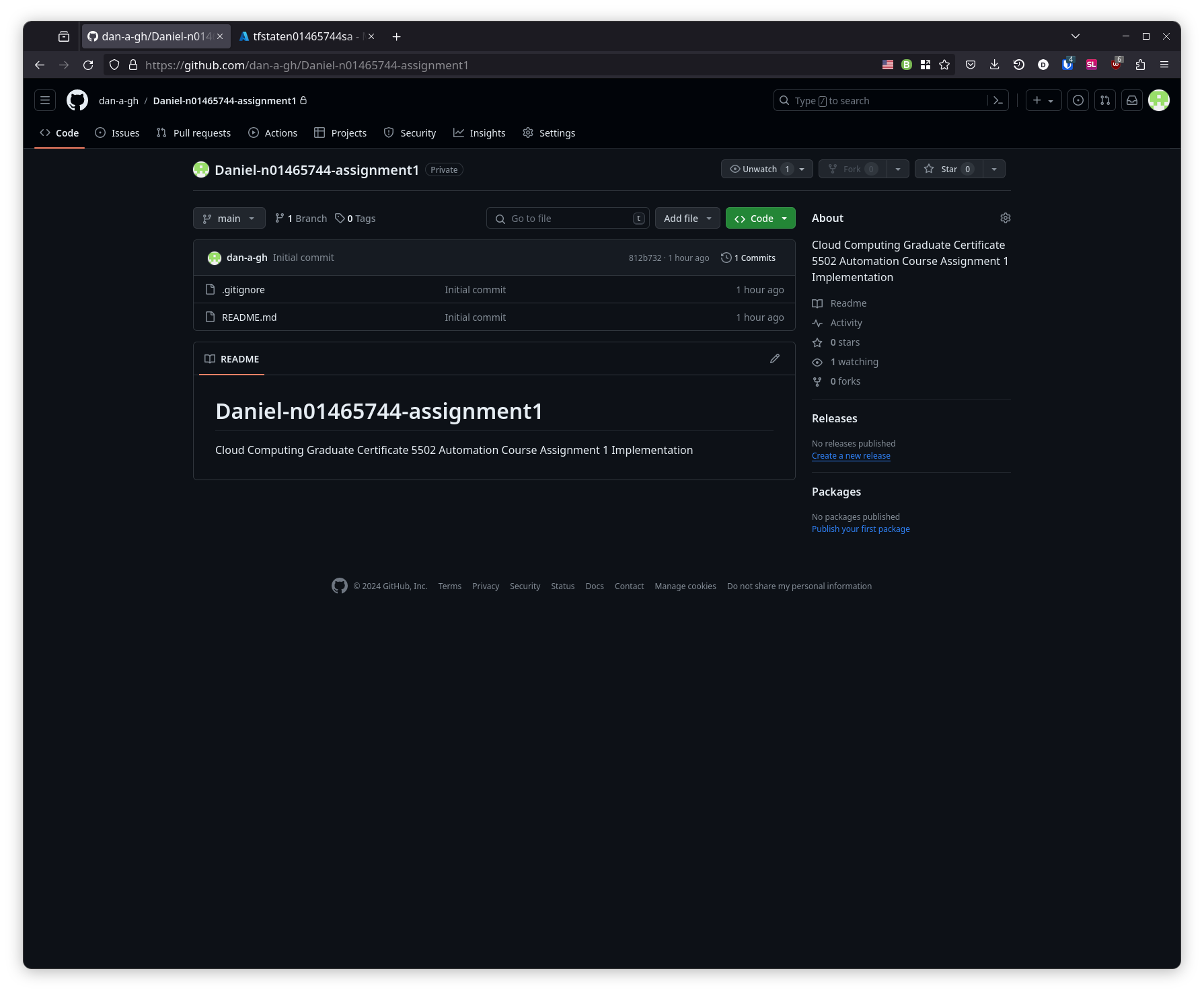
[SCREENSHOT 17: Operating system name and a breakdown of my terminal prompt settings 7](#_Toc17)

# **Section 1: Setup**

## GitHub Repo

The first step I took to start this assignment was to create a GitHub repository in which the code could be stored and shared. It was initialized with a Terraform specific .gitignore and a README.md file.

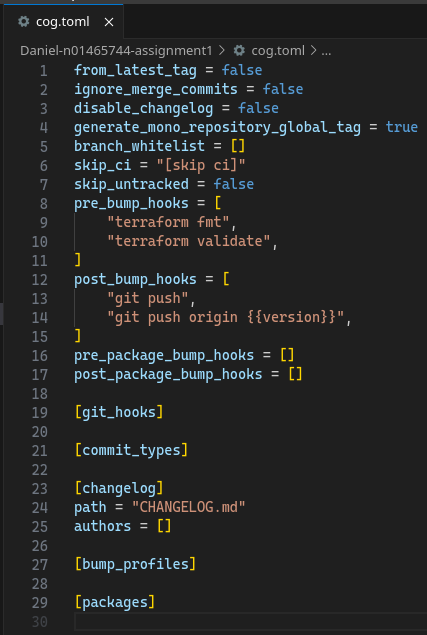
SCREENSHOT 1: The mostly empty initialized code repository in GitHub



## Cocogitto

Cocogitto is a tool to create conventional commits, which is a standard to help organize git commit messages. It can also manage versioning based on what type of work was identified in the commit messages since the last version was tagged. Since I already had the tool installed, setup involved running the command cog init, and filling out the cog.toml configuration file with commands that use git hooks to run before/after bumping the version number.

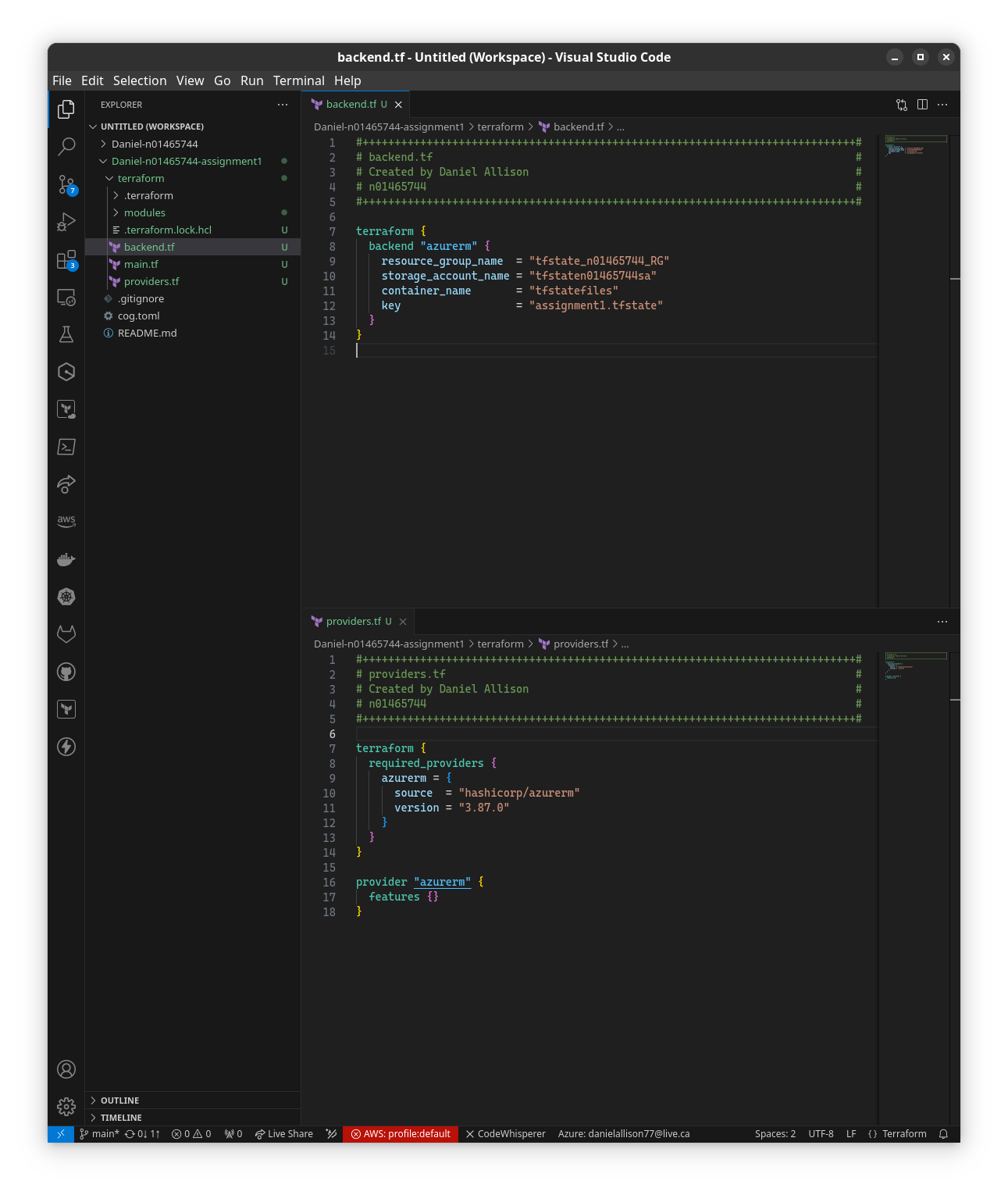
SCREENSHOT 2: cog.toml, the configuration file for cocogitto, a conventional commits tool



## backend.tf and providers.tf

The backend.tf and providers.tf files were next to be created. I drew heavy inspiration from work performed in lab 05. I changed the name of the key once I realized that it was going to overwrite tfstate from the labs if no changes were made.

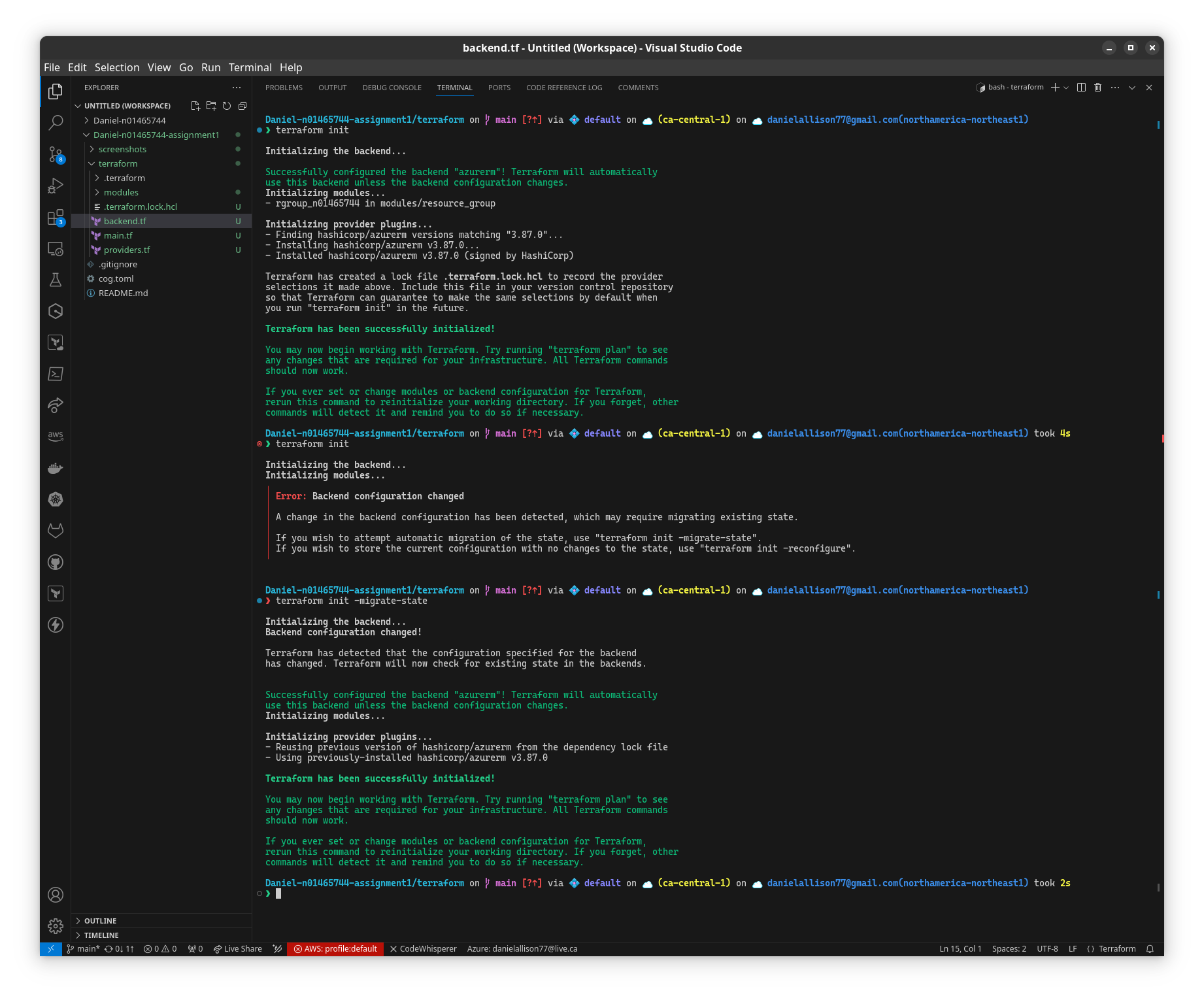
SCREENSHOT 3: backend.tf and providers.tf



## terraform init

With the bare minimum files in place to initialize, I went ahead and tested my code with terraform init. The first time was with the old key tfstate that would have overwritten the labs work, the second successful time with the migrate state option uses the new backend tfstate key.

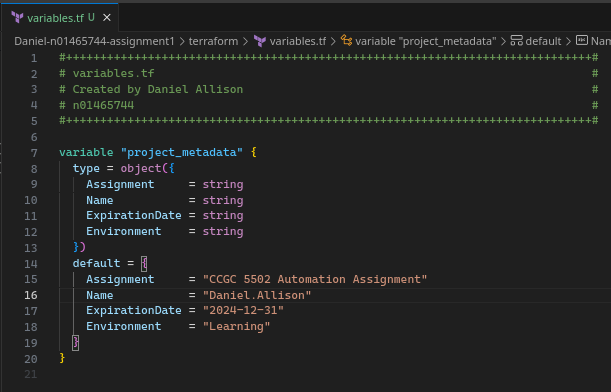
SCREENSHOT 4: terraform init and then again with migrate state to the new backend tfstate key



## Project Metadata

For the data used in tags common to all resources, I have created a variables.tf at the root level with one object variable called “project\_metadata”. I chose this approach over using locals because locals are meant to be used within a module, like a function’s temporary local variables in traditional programming languages.[[1]](#footnote-3)

SCREENSHOT 5: variables.tf in the root module showing the project\_metadata variable, used for tagging every resource

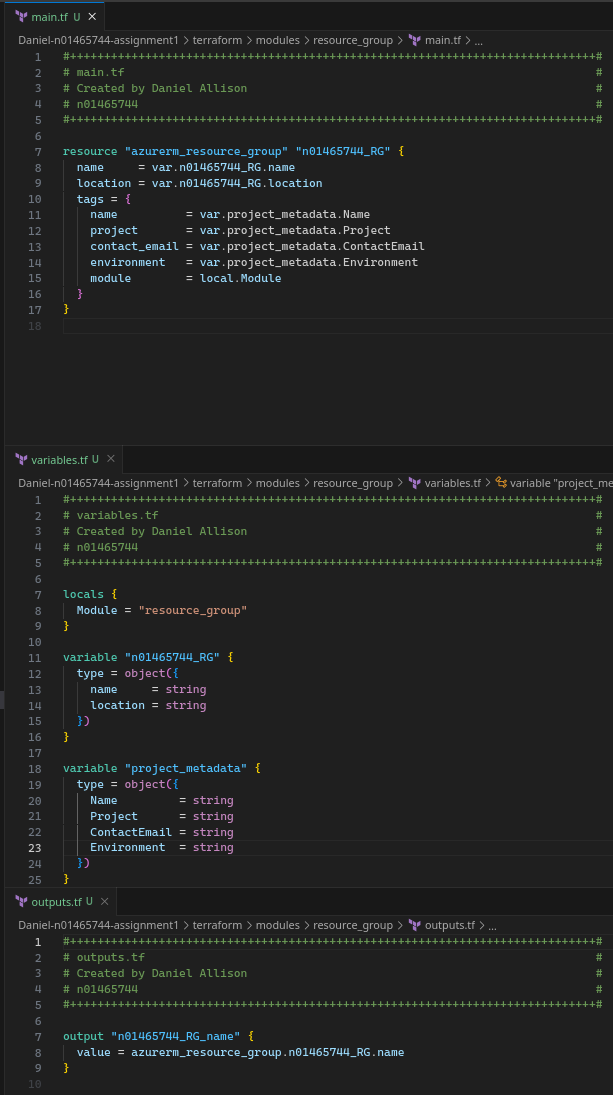


# **Section 2: Child Modules**

## Resource Group Child Module

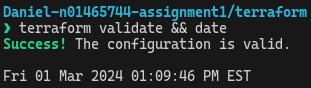
The resource group resource is defined in main.tf with parametrized fields, tagged with project metadata, corresponding variables are declared in variables.tf, and outputs.tf returns the resource group name.

SCREENSHOT 6: the resource group child module’s main.tf, variables.tf, and outputs.tf



This module was tested with terraform validate, plan, and deploy for release 0.1.0

SCREENSHOT 7: terraform validate shows valid configuration for release 0.1.0 with resource group

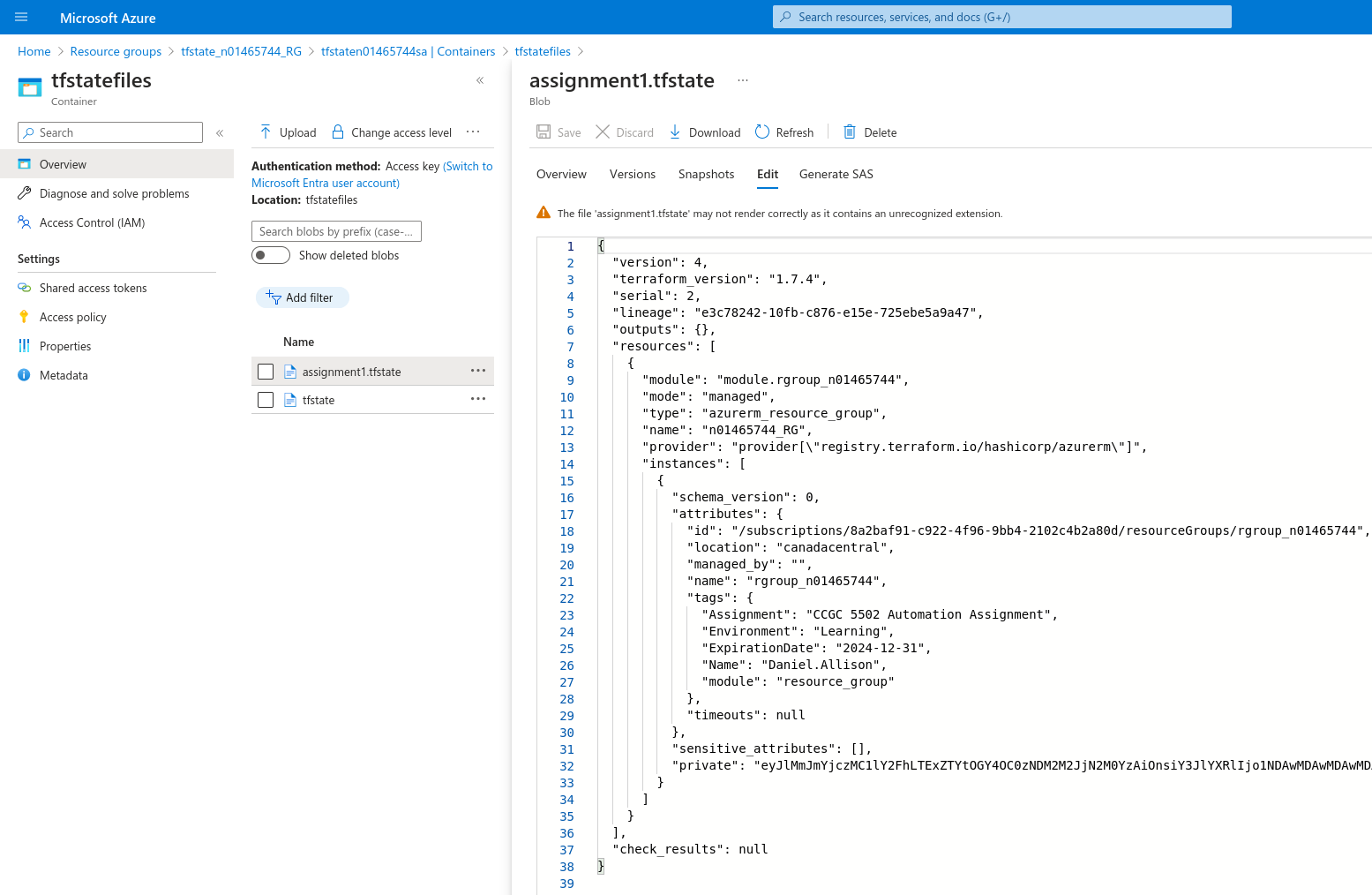


SCREENSHOT 8: terraform state list | nl shows the deployed resource group at release 0.1.0

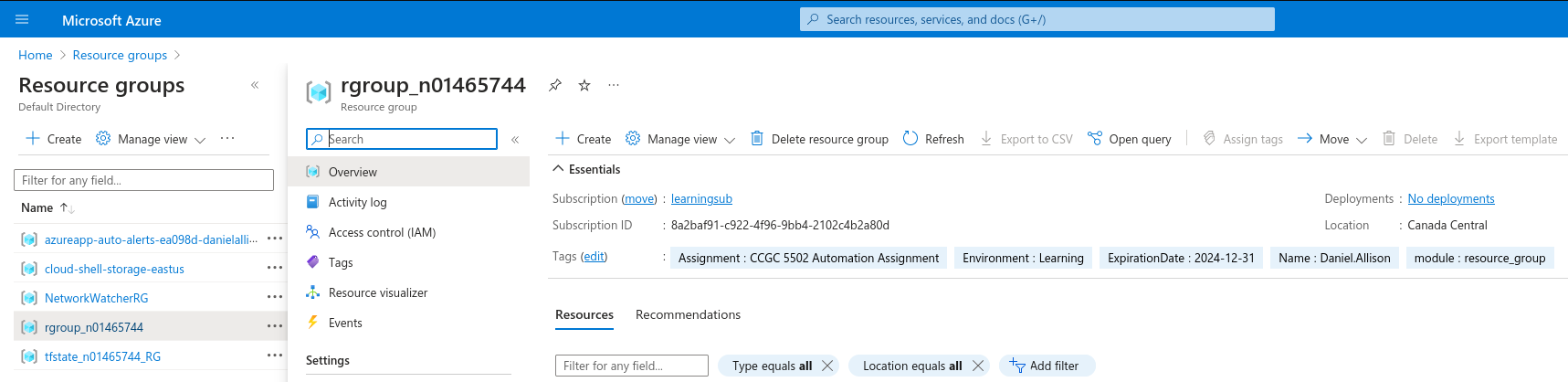


The resource group was verified as deployed in Azure Portal.

SCREENSHOT 9: Azure Portal shows the assignment1.tfstate for release 0.1.0 with resource group



SCREENSHOT 10: Azure Portal shows deployed resource group with tags



1. https://developer.hashicorp.com/terraform/language/values/locals [↑](#footnote-ref-3)