

Network as Code

A comprehensive guide from getting started to running in production

Sogode HOWTO Series

Network as Code

A comprehensive guide from getting started to running in production.

So that you, as an engineer, create better networks with more consistency; you can delegate certain work to colleagues; and you get to focus more on technology instead of troubleshooting.

Second Edition, July 2024

Author: Peet van de Sande

Contact details: peet@sogode.com

Web site: <https://sogode.com/>

Copyright © 2024 by Peet van de Sande, France.

About the book

This book is in the form of a HOWTO and meant for anyone who wishes to get started quickly.

The focus is on getting things done over explaining theory behind it. This keeps the pace up and the entry barrier down and is my personal preference.

We'll have a little theory in the introduction of each section before diving into the practicalities.

Although all companies will have their own software standards and regulations, this book provides working example using a software stack that is both corporate grade as well as freely available for home labs.

Let's get started with Network as Code (NaC).

Table of Contents

DAY 0	1
INTRODUCTION.....	1
TECHNOLOGY STACK	2
<i>Amazon Web Services</i>	3
<i>Code Editor</i>	5
<i>Source Control</i>	6
<i>Development Platform</i>	8
<i>Code Deployment</i>	9
<i>Set up Terraform Cloud</i>	10
<i>Other software</i>	13
DAY 1	14
INTRODUCTION.....	14
CUSTOMISING THE CODE.....	15
FIRST TFC RUN	18
PRE-AUTHORISE TFC RUNS	20
ACCEPT AWS TRANSIT GATEWAY PEERING REQUESTS.....	21
DEPLOY INFRASTRUCTURE STAGE 2	22
DAY 2	23
INTRODUCTION.....	23
UPDATE GIT SETTINGS	24
<i>Branch protection</i>	24
<i>Add collaborator</i>	26
CONFIGURE INFRASTRUCTURE STAGE 3.....	27
<i>Create new branch</i>	27
<i>Update code</i>	27
<i>Create Pull Request</i>	28
<i>Merge Pull Request</i>	31
<i>Verify Terraform Cloud</i>	33
REVIEW	34
SUMMARY	34

Day 0

Introduction

Day 0 is about preparation, design, and planning. Day 0 sets the foundation for the entire network lifecycle. Proper planning and execution during this phase lead to a robust and reliable network infrastructure.

Here are some key points:

1. Planning and Design:
 - a. We focus on designing the network architecture. This includes defining network topologies, selecting infrastructure, and planning for scalability and redundancy.
 - b. We also consider factors like security, compliance, and performance requirements.
2. Configuration and Provisioning:
 - a. We create initial configurations based on the design.
 - b. This involves setting up IP addresses, networks, routing protocols, and access control lists (ACLs).
3. Infrastructure Deployment:
 - a. We deploy the platforms and ensure everything works as intended.
 - b. We install, update and upgrade any relevant software.
4. Documentation and Inventory:
 - a. We create detailed documentation about the network setup. This includes diagrams, IP address assignments, and device inventory.
 - b. Proper documentation helps troubleshoot issues and maintain the network effectively.
5. Testing and Validation:
 - a. We conduct functional tests to validate configurations and connectivity.

Technology Stack

This chapter lists the software and tools making up the technology stack.

Cloud Platform	Amazon Web Services (AWS)
Code Editor	Visual Studio Code (VS Code)
Code Control	GitHub
Development Platform	Terraform
Code Deployment	Terraform Cloud (TFC)
Technical Drawings	Draw.io

Amazon Web Services

Deployment of the network will be done on AWS, so you'll need an account here as well.

Sign up by clicking the 'Sign in to the Console' button in the top right corner at:

<https://aws.amazon.com/>

Instruction video to create a completely new environment:

<https://sogode.com/howto/new-aws-account>

IAM Permissions, Groups, Accounts

Before adding users, you'll need a custom policy to permit Terraform Cloud to manage the Transit Gateways.

The policy is written in a JSON file in the code repository:

`aws/TGWAdministrator.policy.json`

In AWS IAM, select Policies from the left menu and create a new policy.

At the right, select JSON and replace the contents of the Policy editor with the contents of the file.

Hit Next and name the policy `TGWAdministrators`.

Create the below user groups and user accounts.

IAM Groups:

Group Name	Administrators
Permissions Policies	AdministratorAccess

Group Name	NetworkAdministrators
Permissions Policies	NetworkAdministrator TGWAdministrators

IAM Users

User name	netops
Provide user access to AWS Management Console	Yes
User group	Administrators

Day 0: Technology Stack

User name	tfc
Provide user access to AWS Management Console	No
User group	NetworkAdministrators

Command Line Interface

The AWS CLI version 2 is required to run terraform locally and is available from the below URL:

<https://aws.amazon.com/cli/>

Configuring the AWS CLI with an access key and default region:

```
$ aws configure
AWS Access Key ID: <paste key ID>
AWS Secret Access Key: <paste secret>
Default region name: us-east-2
Default output format: json
```


Code Editor

Visual Studio Code is one of the best all-round code editors with AI plugins and a wide range of extensions for all sorts of languages, including:

- AWS
- Azure
- GitHub
- Kubernetes
- Python
- Terraform

The software is freely downloadable and runs on Windows, Linux and Mac.

URL: <https://code.visualstudio.com/>


Follow the above URL and install the software.

Upon first start, VS Code will present you with a walkthrough to set up the environment.

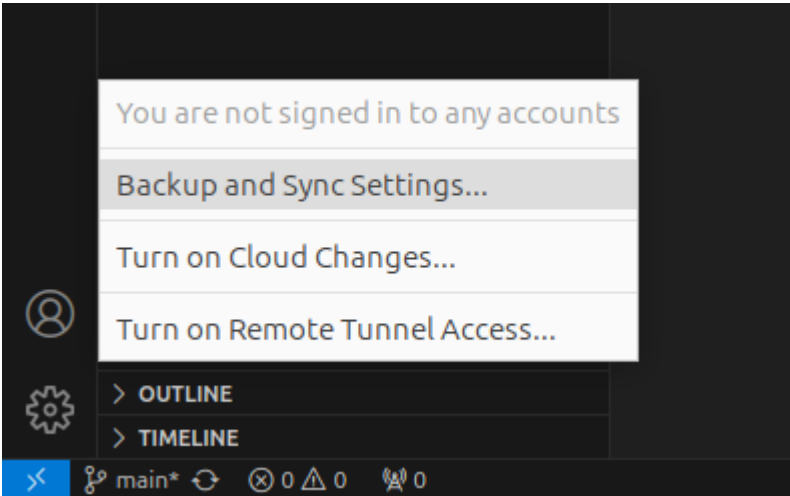
VS Code has an exceptionally good built-in mechanism of recommending extensions that add value such as code completion and syntax highlighting.

Recommended plugins for code completion, syntax checking and automatic suggestions:

- HashiCorp Terraform
- Python
- Pylance



Store your settings in your cloud account to ensure that you can log on from any computer – or your own after a fresh OS install – and return to your carefully curated and trusted environment.



The screenshot shows the 'Settings' menu in Visual Studio Code. The menu is open, displaying several options: 'You are not signed in to any accounts', 'Backup and Sync Settings...', 'Turn on Cloud Changes...', and 'Turn on Remote Tunnel Access...'. Below these options, there are icons for 'OUTLINE' and 'TIMELINE'. The background of the screenshot is dark, and the text is white and light blue.

Source Control

Source Control comes native to most major Cloud Service Providers (CSPs) however to maintain true to the tenet of service recoverability, we recommend GitHub.

This means that, when the normal cloud environment is compromised, you can still return to your code and deploy it elsewhere. A different account or a different region.

Part of the reasoning to NetDevOps is to be better prepared for eventualities, however improbable they may seem.

Online account

Head on over to <https://github.com/> and register for a set of free accounts.

For my demo environment, I use multiple accounts to show how different personas would work together. You can do everything under the same account if you want to.

- NetOps: this is the Network management role
- DevOps: This is the developer role who would use the provided code examples to set up their application environment

Client software

Download the software from the below URL:

<https://git-scm.com/download/>

Install the git client and configure the client in a command or terminal window:

```
$ git config --global user.name <Your Name>
$ git config --global user.email <you@email.address>
```

To check code in from your desktop, the easiest is to use SSH keys for authentication.

Open a PowerShell (Windows) or Terminal (Linux / Mac) and issue the following command:

```
$ ssh-keygen
```

Next, show the public key part to copy it to the clipboard:

```
$ cat .ssh/id_rsa.pub
```

Day 0: Technology Stack

The output will be something like this:

```
$ cat .ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGDQNBH/lxoAaxb9oTiskWDl6PY+2+/1daPWWf18TiVy52ia
+VPSD+i+IUaAZ0hO1EluViXRYudCSnmgi+U2CFG0ofOfotsHuzkmt1MDYbfscRcVM+j5ea7y4FTe
Uiavxfrlq9Lb43usp4vRRgklkqFtbTS0s8DImdvOqaL+Im9GDdlp08wyVlKomrSG4QVGUMapwPm
YfghZP+/27dfF/5KNsEnq2x3ljfPNr0Lkpvr2Opv4IZXTw/z0ompamwVpQeeyxjFgucDGhw6HJS
wBBreI+P1QMVs3bG/fsxv540hfd8JdyMajMnxGPBrGt0HTck+uBZ4EM1od/Lp5y/8ujevYchsZz
XJZtWmiWXw2Zewadlo+UROeZMRhT7q61hQr3CvO9QcvvBw80rp7+7FuDBiKhcn9KdGCRg+US9se
exI4kldVvbQFKmWXbuwkXw3IZjxNDCRFfhLtQ8pqHFm+nQCiYdjfPk9QkRLuBjpPxpIe5PX6Wh4
VI1Hq8gsRjLf6mhuak= sogodep@laptop
$
```

In my case, there is also a small comment “sogodep@laptop”, which is optional. The actual key ends with the “=” character.

Go into your GitHub NetOps account to add the SSH key:

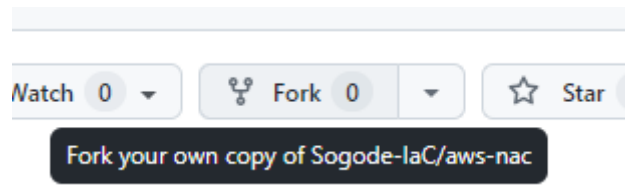
Account -> Settings, SSH & GPG Keys -> New SSH key

This will be an authentication key.

Forking the code

The next step is to copy the example repository into your own account by forking it. Follow the below URL and click the Fork button:

<https://github.com/Sogode-laC/aws-nac>



Next, on your desktop, clone the repository.

```
$ git clone git@github.com:<MyAccount>/aws-nac.git
```

Development Platform

The platform on which we will base our NaC is Terraform.

Terraform has recently undergone a change of license and ownership and as a result, an alternative has been developed. OpenTofu is interchangeable ¹ with Terraform, especially if you do not use any of the platform add-ons such as Vault or Packer.

They can even be installed on the same computer at the same time without interfering with each other.

Download the software from:

<https://developer.hashicorp.com/terraform/downloads>

Or

<https://opentofu.org/docs/intro/install/>

For Windows, see the following video to download, install and configure the software:

<https://sogode.com/howto/install-opentofu-terraform-on-windows/>



The only thing I found to date where Terraform and OpenTofu have incompatibility issues, is when the version is explicitly declared, i.e. when the backend.tf looks like in the below screenshot.

```
backend.tf x
backend.tf > terraform
1 terraform {
2   required_version = ">= 1.8.0"
3 }
```

My advice is to not explicitly declare the required version unless you know there actually is a requirement for a specific version.

Terraform uses the AWS credentials as configured in the AWS CLI client so there's nothing specific to configure here.

¹ Over time and with the expansion of features, it is expected there will be compatibility issues. At the time of writing and given the examples in this book, no issues were encountered.

Code Deployment

Code Deployment into AWS will be done via Terraform Cloud.

Follow the URL below to sign up:

<https://app.terraform.io/session>



Choose to use your GitHub account when registering so that you'll have less accounts to worry about and you're positive they work together.

In my example, I am setting up an organisation with a NetOps and a DevOps user to show how these roles would work together.

Set up Terraform Cloud

Log onto [TFC Portal](#) as the NetOps user and create a new project.

We'll name the project 'aws-nac' to keep it the same as the GitHub repository just because it's easier to remember for now.

You could name it anything you want, there are no direct links between the name of your project here and the name of the repository in GitHub but as always, we like to keep things simple.

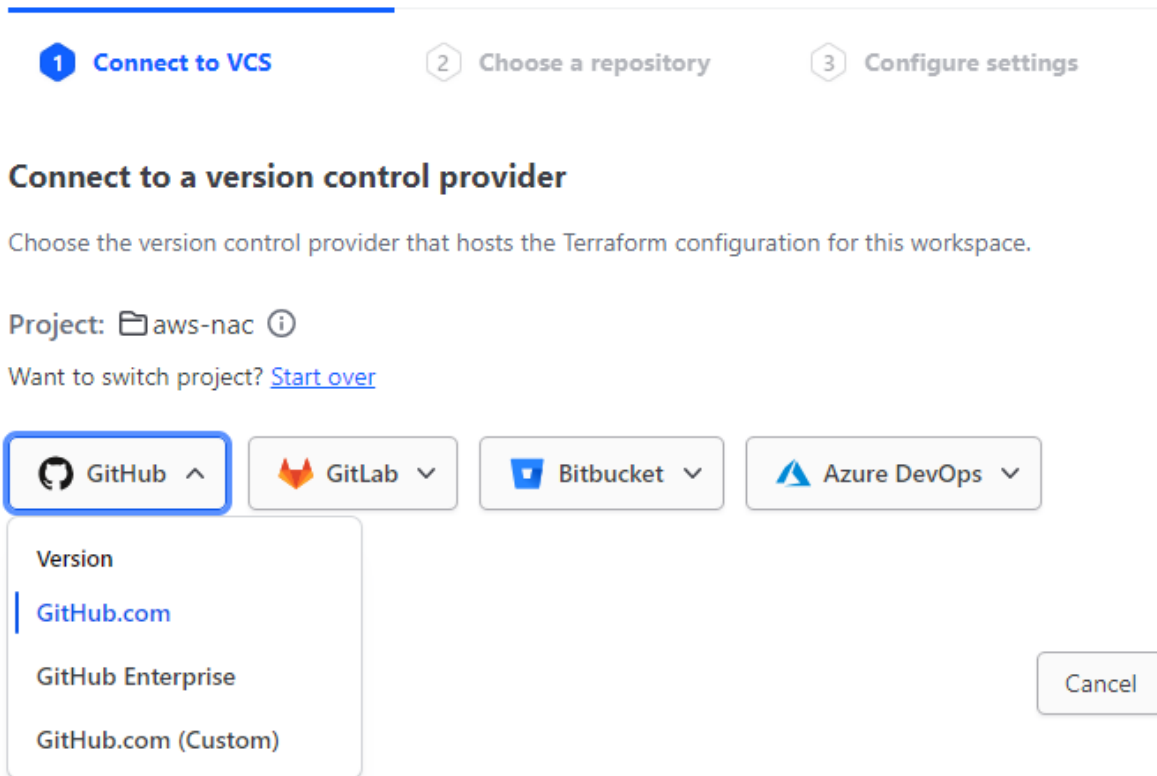
Next, you're invited to create a Workspace. We'll create a Version Control Workflow, so that whenever code is committed to the repository, a new run in Terraform is triggered.

Go ahead and create a new Workspace. In the next screen, choose GitHub:

Sogode-IaC / Workspaces / New Workspace

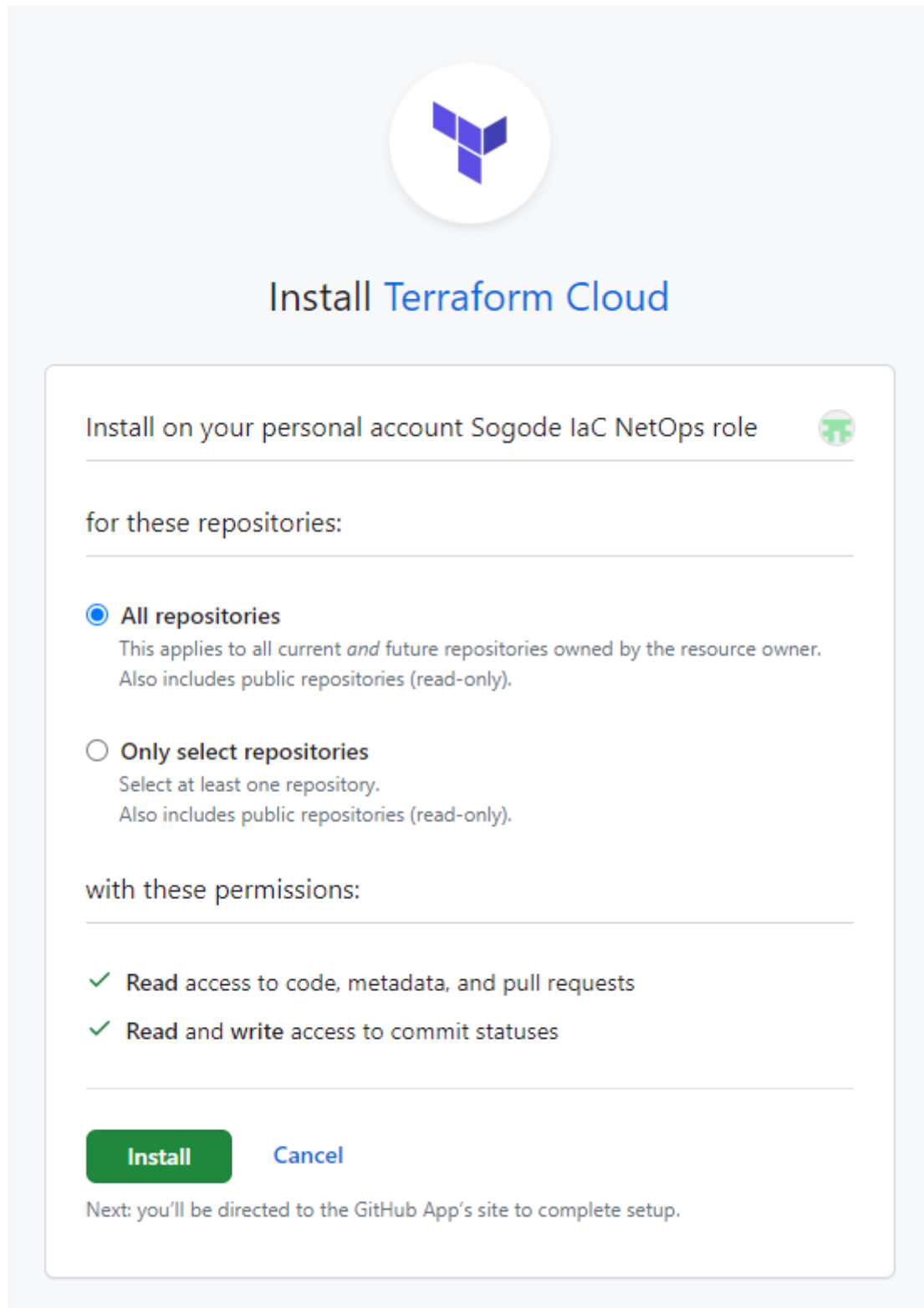
Create a new Workspace

HCP Terraform organizes your infrastructure resources by workspaces. A workspace contains infrastructure resources, variables, state data, and run history. [Learn more](#) about workspaces in HCP Terraform.



You'll be presented with a pop-up asking about permissions and to choose a repository.

For now, we can permit access to all repositories in our account but in production we'd restrict access.



Click install, choose the `aws-nac` repository and click Advanced options.

Scroll down to the VCS Triggers and enter `main` in the VCS branch to prevent it from running on any updates except to this branch. This becomes particularly important when

you advance in your Git skills and start creating development branches which you'd want to trigger new runs.

VCS Triggers

Automatic Run Triggering

Workspaces with no Terraform working directory will always trigger runs.

- Always trigger runs**
- Only trigger runs when files in specified paths change**
Supports either glob patterns or prefixes.
- Trigger runs when a git tag is published**
Git tags allow you to manage releases.

VCS branch

main

The branch from which to import new versions. This defaults to the value your version control provides as the default branch for this repository.

Go ahead and click 'Create'.

The next step is to add AWS credentials, so follow the link to the workspace overview to configure variables.

We will be adding the following variables:

Category	Key	Value	Sensitive
Environment variable	AWS_ACCESS_KEY_ID	<tfc access key>	Yes
Environment variable	AWS_SECRET_ACCESS_KEY	<tfc key secret>	Yes

Terraform Cloud is now configured to run the Terraform from our repository.

Other software

This section lists further software that is recommended.

Python 3

The python3 runtime is already installed on both Linux and MAC.

Download the software from the below URL:

<https://www.python.org/downloads/>

Vim

Vim is a versatile text editor that supports Regular Expressions and many other cool features.

<https://www.vim.org/download.php>

Note that we will be doing most text editing in Visual Studio Code, but it is useful to have a good, standalone text editor available.

Notepad++

Another great text editor, and a bit easier to get used to, is Notepad++.

<https://notepad-plus-plus.org/>

Postman

Postman is used to test APIs and other web-based functionality.

Sign up at the below URL, and optionally download the desktop client:

<https://www.postman.com/>

Bitwarden

You'll end up with a lot of accounts and passwords. I use Bitwarden to keep track of all my accounts and create great passwords.

<https://bitwarden.com/>

Day 1

Introduction

Day 1 sets the deployment in motion; this is where ideas become reality. It is about building out the designs and implementing them, making sure the environment is fit for production.

Typical Day 1 operations are:

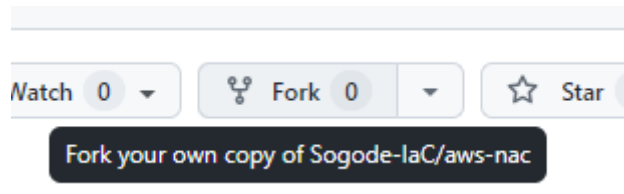
- **Deployment Phase:** We complete our work and commit changes.
- **Deployment Execution:** If the deployment pipeline isn't fully automated, Day 1 includes starting the deployment.
- **Rollback Handling:** In case of major problems, Day 1 also involves initiating rollbacks.

Customising the code

Forking the code

The first step is to copy the example repository into your own account by forking it. Follow the below URL and click the Fork button:

<https://github.com/Sogode-laC/aws-nac>



Next, on your desktop, clone the repository.

```
$ git clone git@github.com:<MyAccount>/aws-nac.git
```

Start VS Code to customise the code for your environment.

```
$ cd aws-nac  
$ code .
```

Select `backend.tf` to insert the correct HCP Cloud Organization ID.

The Organization ID can be found under “Organization Settings” in your [HCP dashboard](#).

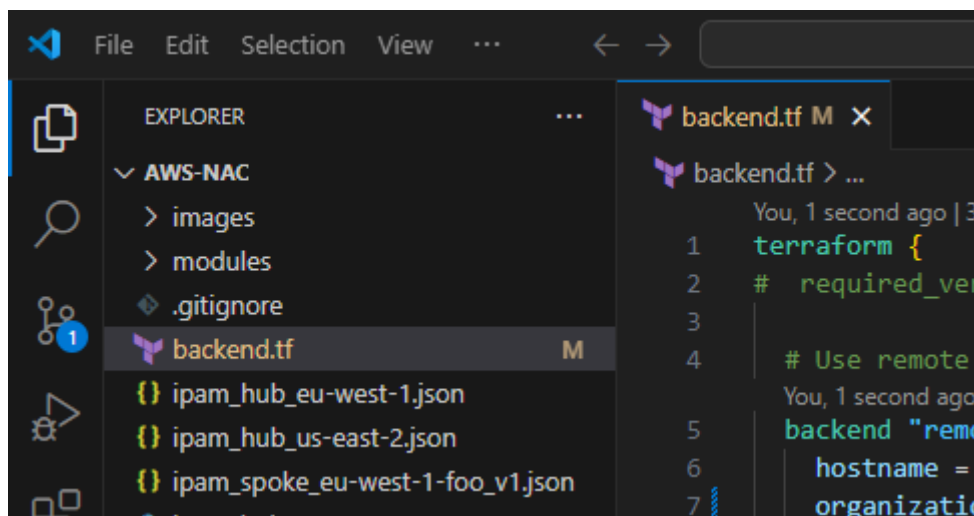
All being well, the file should look like this:

Day 1: Customising the code



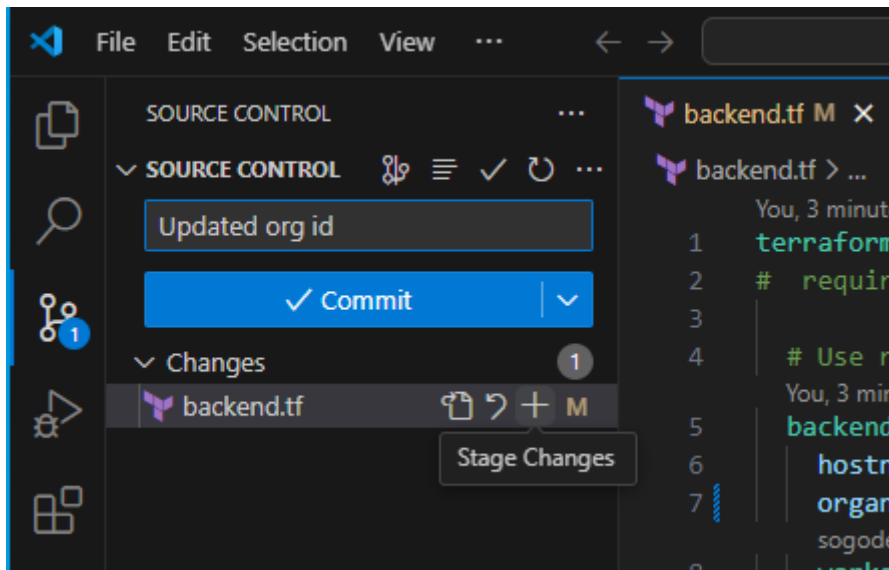
```
backend.tf
backend.tf > ...
...
1 terraform {
2   # required_version = ">= 1.8.0"
3
4   # Use remote backend for HCP Terraform (formerly Terraform Cloud)
5   ...
6   backend "remote" {
7     hostname = "app.terraform.io"
8     organization = "org-rba3Mx8AiLaXpjub"
9     ...
10
11    workspaces {
12      | name = "aws-nac" # Verify!
13    }
14  }
15 }
```

Save the file and check it into GitHub. In VS Code, you'll see the branch icon being overlaid with a 1, and backend.tf is now marked with an M (for Modified):



Click the branch icon to open the Git explorer, write a small comment about your changes (e.g. "Updated org id") and click the + next to the filename to add it to the commit:

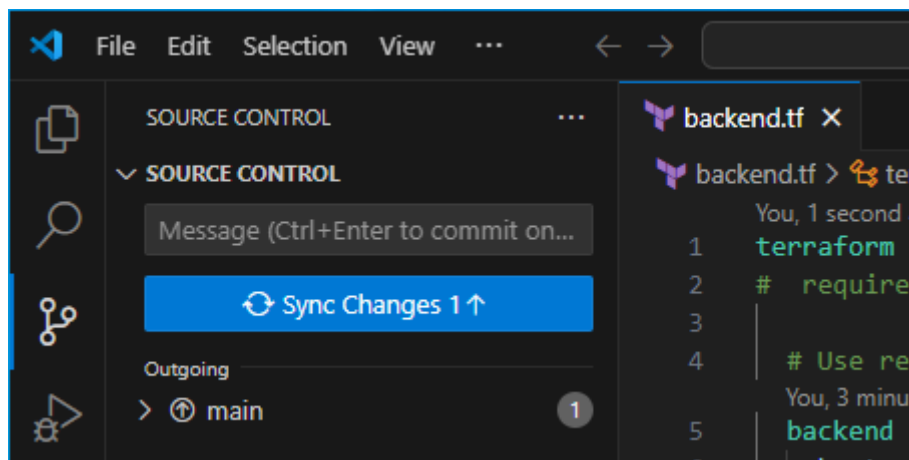
Day 1: Customising the code



Click the commit button to save the changes to the repository.

Now, the repository is updated on your desktop but not yet online. The blue Commit button is now changed to say “Sync Changes”.

If you had a lot of commits, you could add another message here. In our case, it’s just a single commit so we just click the blue button to sync send our changes to the online repository.




With this upstream syncing, an action is triggered in Terraform Cloud.

First TFC Run

Head back to the TFC portal and go to your `aws-nac` Workspace, then select Runs from the left sidebar.

If all is well, it should now be running a plan:

Current Run

 **Updated organization ID** CURRENT Planned
#run-NqES1LHiZ3tSHiaM | sogode-iac-netops triggered via GitHub | Branch `main` | `b458d3e`
in a few seconds

Click to see its details:

Sogode-iac / Workspaces / aws-nac / Runs / run-NqES1LHiZ3tSHiaM

aws-nac

[Force unlock](#) [+ New run](#)

ID: ws-Vp6DrtyVp89A42Fd [🔗](#)

[Add workspace description.](#)


[🔒](#) Running [📄](#) Resources 0 [🐦](#) Terraform `v1.8.4` [🕒](#) Updated in a few seconds


Updated organization ID

CURRENT Planned

Pending confirmation
2 minutes

Resources to be changed
`+35 -0 -0`


 **sogode-iac-netops** triggered a run from GitHub a minute ago Run Details

 **Plan finished** a minute ago Resources: 35 to add, 0 to change, 0 to destroy

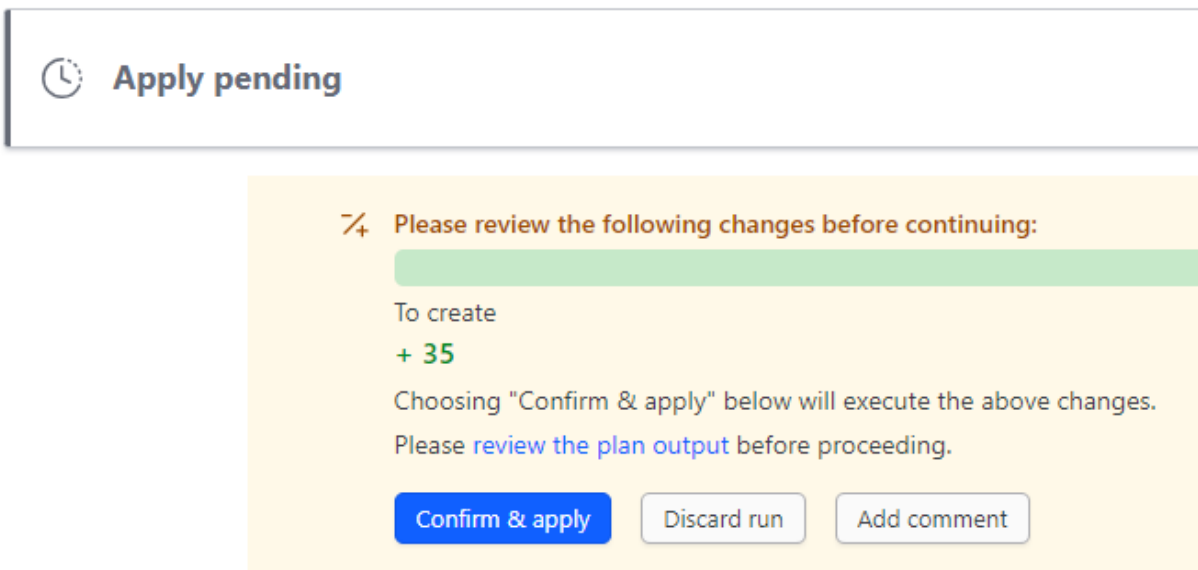
Started a minute ago > Finished a minute ago


+ 35 to create


Filter resources by address... Filter by action Terraform 1.8.4 [Download raw log](#)

>  `aws_ec2_transit_gateway_peering_attachment.tpa_ue2_x_ew1`

At the bottom of the page, there are some buttons to either apply or discard the run.



 **Apply pending**

 **Please review the following changes before continuing:**

To create
+ 35

Choosing "Confirm & apply" below will execute the above changes.
Please [review the plan output](#) before proceeding.

Go ahead and click the blue Confirm & apply button. This will ask you to add a comment (this is mandatory in TFC) before you can click to confirm the plan.

The page will now update as resources are being created and eventually display all the created resources and configured outputs.

Note that you can always stop and cancel the current run – if you should wish to do so.

Pre-authorise TFC Runs

Now that we've seen how TFC runs are triggered and we know they work okay, we can enable the automatic application of the plans.

In the TFC Portal, go to the aws-nac Workspace and edit its Settings.

On the General page, scroll down to see the Auto-apply heading and tick the box for the API, CLI & VCS runs:

Auto-apply

Sets this workspace to automatically apply changes

- Auto-apply API, CLI, & VCS runs ⓘ
- Auto-apply run triggers ⓘ

Scroll down to the bottom of the page to hit the purple Save Settings button.

Next time you check code into the main branch, this will trigger TFC to deploy any changes.

This means we now are effectively moving our authorization to Git. Whoever is permitted to commit to the main branch is permitted to make infrastructure changes.

Accept AWS Transit Gateway peering requests

One of the 'features' of AWS is that Transit Gateway Peering requests between Transit Gateways must be manually approved in the console so let's go ahead and do that.

Log on to the AWS Console as NetOps and head over to the `eu-west-1` (Ireland) region on the VPC Dashboard.

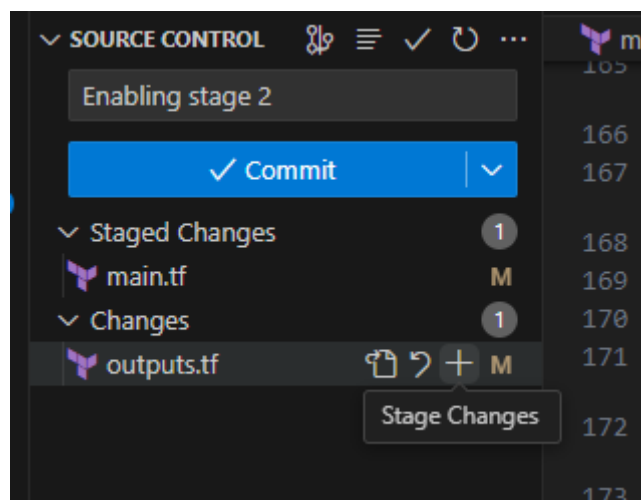
Scroll down to the Transit gateway attachments section and accept the pending request in the list.

Deploy Infrastructure Stage 2

We must now go back to VS Code to enable the deployment of Stage 2, which comprises of the Transit Gateways' backbone routing.

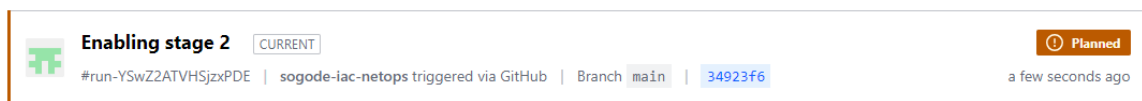
In `main.tf` and `outputs.tf`, uncomment the whole of the Stage 2 block (Stage 3 is clearly indicated and must remain commented out for now).

Save the files, then commit with a message and sync the repository.



Head back to the TFC Portal and watch the runs of the project. You'll see a new run automatically triggered:

Current Run



Day 2

Introduction

Day 2 operations refer to the ongoing management and optimization of networks after their initial deployment. They are about bringing the technology into the wider organisation for collaboration.

- Challenges in Day 2 Operations:
 - Day 2 operations involve manual processes that are time-consuming and prone to inconsistencies.
 - Focusing solely on device health overlooks application-specific network requirements.
 - Thousands of reported network issues often fall into a smaller set of recurring issue types.
- Strategic Approach:
 - Successful Day 2 network management considers the delivered results or “intents” of the network, not just individual device health.
 - Sharing knowledge across IT operations teams leads to more scalable troubleshooting processes.
- Automation Benefits:
 - Automation can significantly enhance Day 2 NetOps by providing scalable and repeatable processes.
 - It allows the NetOps team to manage networks continuously and efficiently.

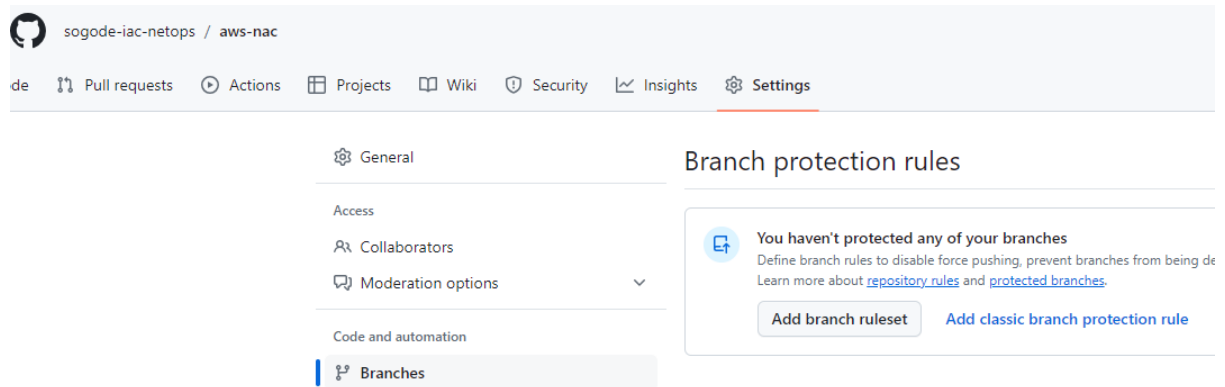
In our case, this is where the DevOps account comes into play to add on to our infrastructure. We’re effectively delegating work to the DevOps team and all we must do is verify that their code meets our standards. After our review, it’s automatically deployed to production.

Update Git settings

Branch protection

Before allowing everybody to update our code, we need to protect the main branch, as this is the one that triggers TFC to actually deploy the code.


In GitHub (as NetOps), go to the Settings tab of the repository and scroll down to the Branches section to add a **classic branch protection rule**.



Day 2: Update Git settings

The Branch name pattern is `main` and we tick the boxes to require a pull request before merging and at least 1 approval:

Branch protection rule

 **Protect your most important branches**

[Branch protection rules](#) define whether collaborators can delete or force push to the branch and set requirements for any pushes to the branch, such as passing status checks or a linear commit history.

[Your GitHub Free plan](#) can only enforce rules on its public repositories, like this one.

Branch name pattern *

Protect matching branches

Require a pull request before merging
When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

Require approvals
When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.

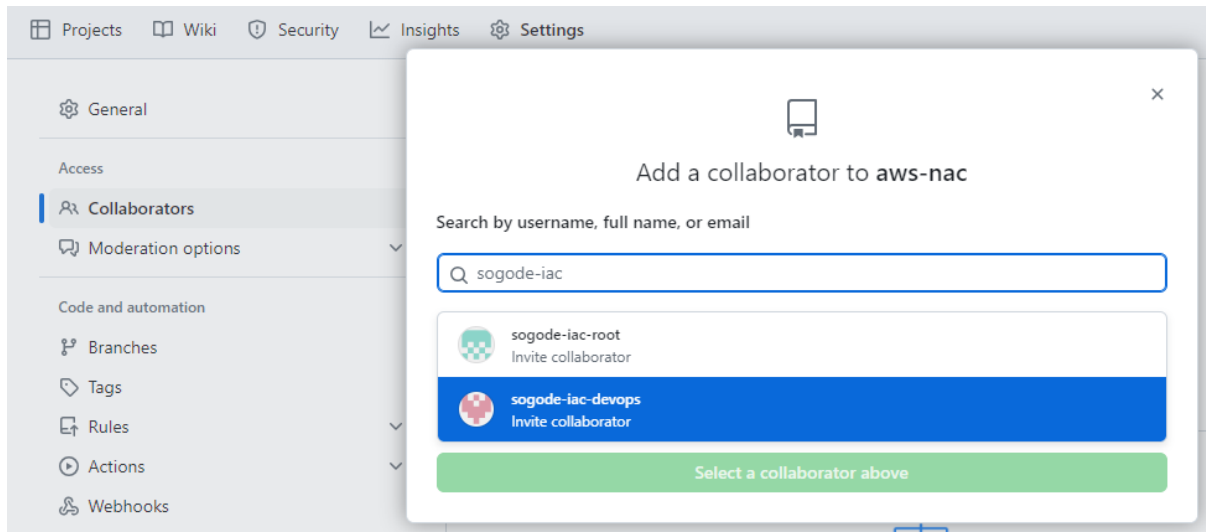
Required number of approvals before merging:

Dismiss stale pull request approvals when new commits are pushed
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

Scroll down to the bottom and hit the green Create button.

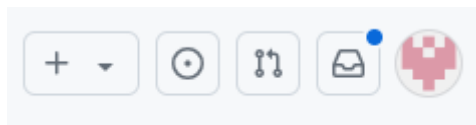
Add collaborator

Next we go to the Collaborators section on the settings page to add the DevOps account as a collaborator.



Add your DevOps account and hit the green button.

Now, log out of your GitHub account so that you can log back in as DevOps to accept the invitation to collaborate, which will be in the inbox at the top left corner.



Go to your inbox, open the message and accept the invite. This will take you to the repository, where we will be updating the code.

Configure Infrastructure stage 3

Because it's a bit much to change our VS Code to work with a new account, we will be making the next code changes inside GitHub, still logged in as DevOps.

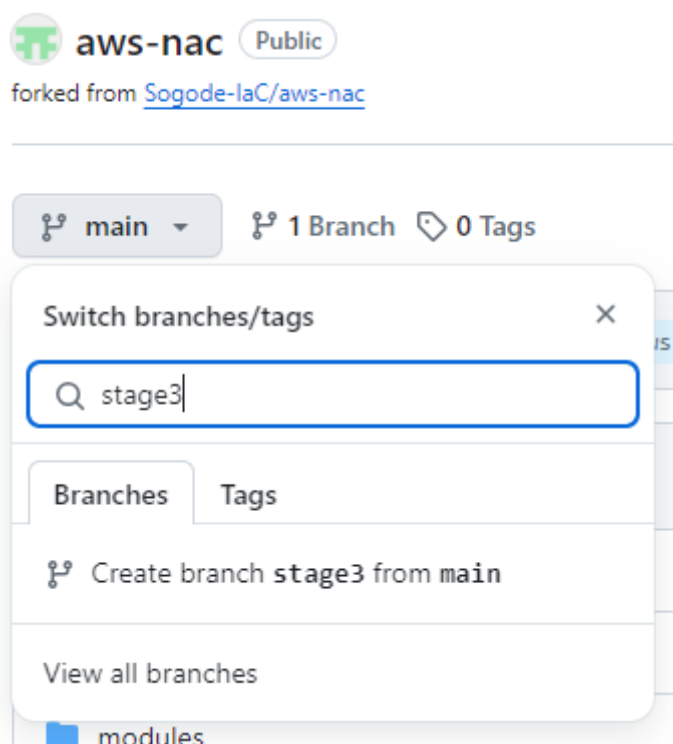
The third stage adds an application VPC, connects it to the nearest TGW and updates the routing.

Create new branch

A branch allows you to keep a personal version of the files to work with, without interfering with the production files.

When your code is ready for production, you create a Pull Request to merge the branch back in.

Click the main branch button, to unfold the option to create a new branch. Let's name it `stage3` and click the Create branch link.



Update code

Next, open `main.tf` and hit the pencil in the top right corner to edit it.

Scroll down to uncomment the Stage 3 section (i.e. remove the `/*` and `*/` lines around it).

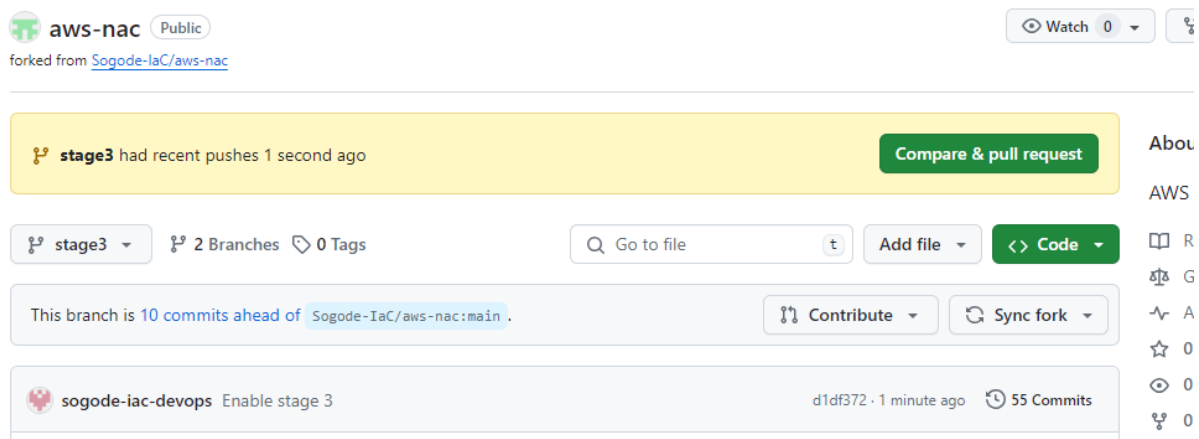
Day 2: Configure Infrastructure stage 3

Scroll back up to click the green Commit Changes button, which will bring up a form to add a commit message. Enter a message and hit the green button.

Do the same for `outputs.tf`.

Create Pull Request

When you scroll up to the top of the page you'll see a message about changes and suggesting to create a pull request.

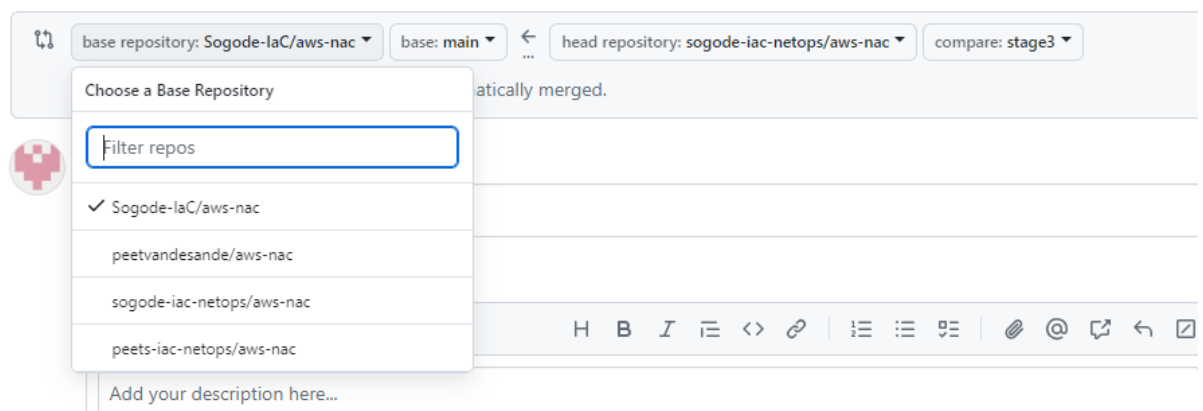


Go ahead to Compare & pull request. This brings up a new page that shows a message about whether the code can be merged and where you can add a message about your changes.

In our case, the comparison was made with the repository where the code was forked from instead of the current repository, so make sure to select your own repository.

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about](#)



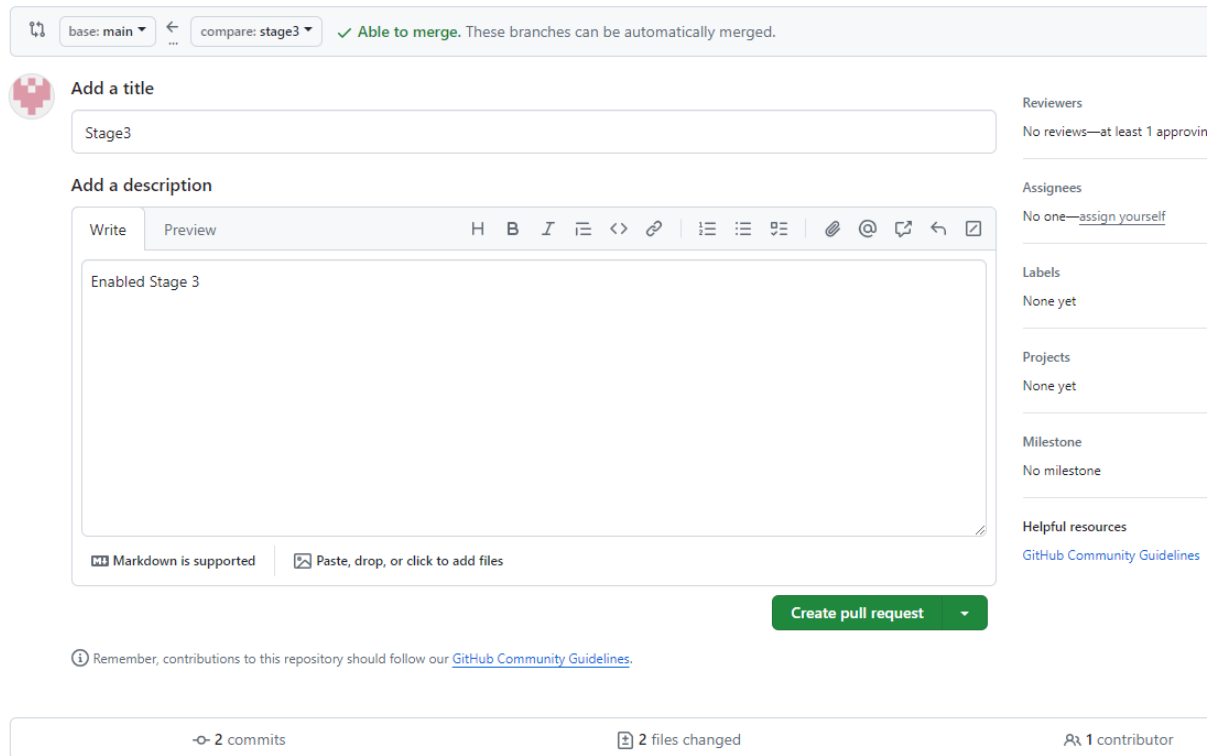
In my case, I have to select the `sogode-iac-netops/aws-nac` repository.

Add a little description and ensure all is as intended.

Day 2: Configure Infrastructure stage 3

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).



The screenshot shows the GitHub pull request creation interface. At the top, it indicates the base branch is 'main' and the compare branch is 'stage3'. A green checkmark and text state 'Able to merge. These branches can be automatically merged.' Below this, there are sections for 'Add a title' (with 'Stage3' entered), 'Add a description' (with 'Enabled Stage 3' entered), and a 'Create pull request' button. On the right side, there are sections for 'Reviewers', 'Assignees', 'Labels', 'Projects', 'Milestone', and 'Helpful resources'. At the bottom, it shows '2 commits', '2 files changed', and '1 contributor'.

Here, you can see the comparison is between 2 branches of the same repository, and at the bottom it shows there are 2 files changed.

Go ahead and hit the green button.

Day 2: Configure Infrastructure stage 3

Now, we're given error messages about not being able to merge, which means the branch protection works.

Stage3 #1

Open sogode-iac-devops wants to merge 2 commits into main from stage3

Conversation 0 Commits 2 Checks 0 Files changed 2

sogode-iac-devops commented 1 minute ago Collaborator

Enabled Stage 3

sogode-iac-devops added 2 commits 12 minutes ago

- Enable stage 3 Verified 4806389
- Enable stage 3 Verified d1df372

Add more commits by pushing to the `stage3` branch on `sogode-iac-netops/aws-nac`.

Review required
At least 1 approving review is required by reviewers with write access. [Learn more about pull request reviews.](#)

All checks have passed
1 successful check [Show all checks](#)

Merging is blocked
Merging can be performed automatically with 1 approving review.

Merge pull request You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Log out of GitHub so that we can log back in as NetOps and review the PR.

Merge Pull Request

Log into GitHub as Netops and go to your repository.

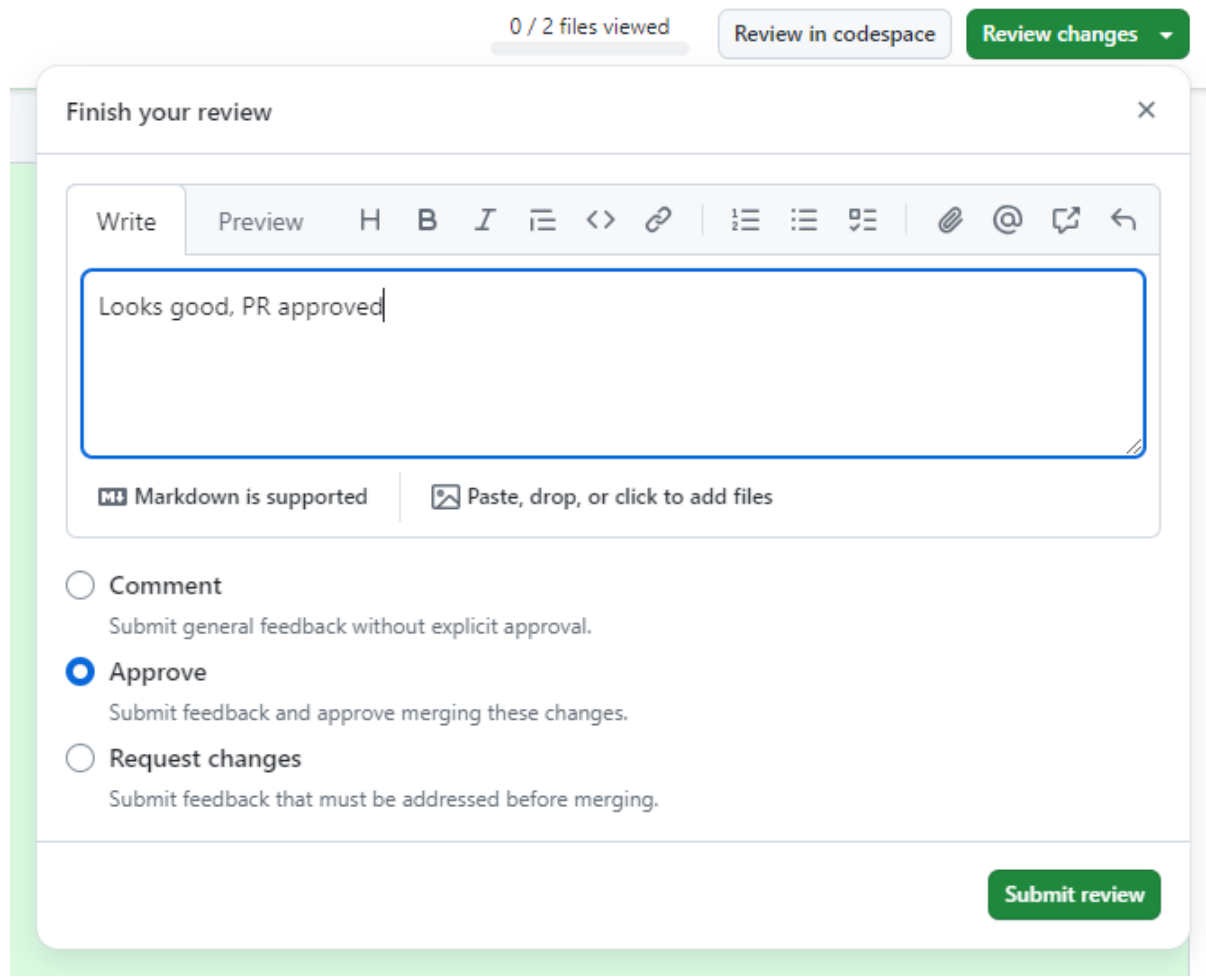
In the top bar, you'll see there is 1 Pull request. Go to the Pull request page and open the PR that was created by DevOps.

Select the link to add your review.

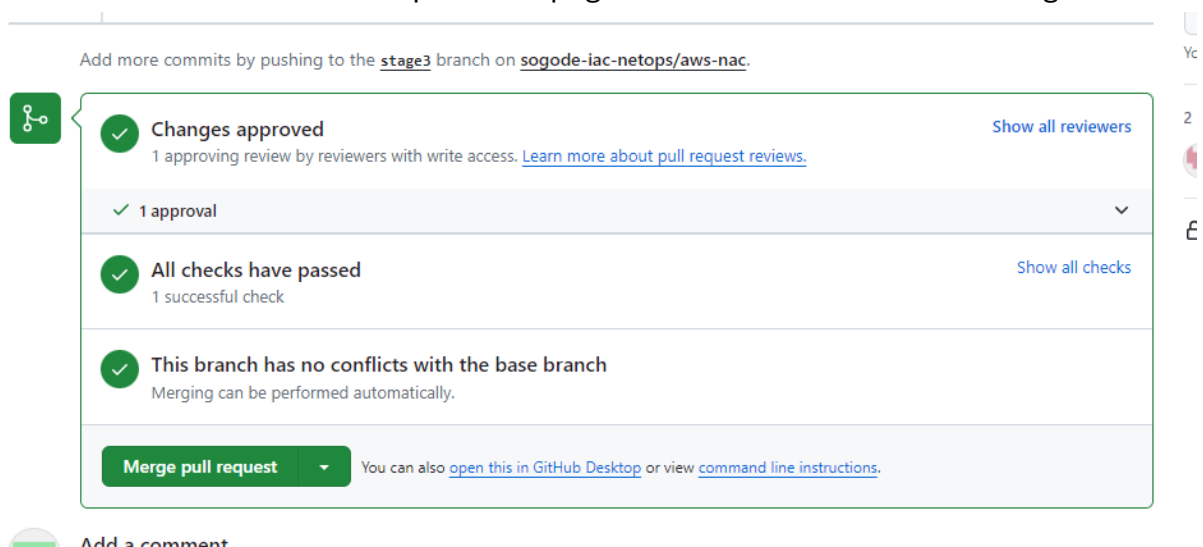
The screenshot shows a GitHub Pull Request page for 'Stage3 #1'. The repository is 'sogode-iac-devops' and the pull request is from the 'stage3' branch to the 'main' branch. The page header includes navigation links for Pull requests (1), Actions, Projects, Wiki, Security, Insights, and Settings. Below the header, there are tabs for Conversation (0), Commits (2), Checks (0), and Files changed (2). A comment from 'sogode-iac-devops' (a collaborator) states 'Enabled Stage 3'. Below the comment, a commit history shows two commits: 'Enable stage 3' (verified, 4806389) and 'Enable stage 3' (verified, d1df372). A message indicates that more commits can be added by pushing to the 'stage3' branch on the 'sogode-iac-netops/aws-nac' repository. At the bottom, there are two status boxes: 'Review required' (At least 1 approving review is required by reviewers with write access. [Learn more about pull request reviews.](#) [Add your review](#)) and 'All checks have passed' (1 successful check. [Show all checks](#)).

Day 2: Configure Infrastructure stage 3

When all looks good, hit the green button top right, enter a message and select Approve before hitting the Submit button.



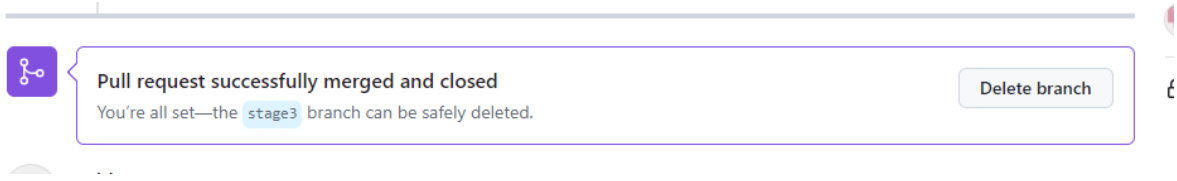
This takes us back to the previous page where we can hit the Merge button.



Hit Merge, add another message and confirm.

Day 2: Configure Infrastructure stage 3

You'll see confirmation the merge succeeded plus a suggestion to delete the branch. This is good practice to keep the code and repository tidy.



Delete the branch.

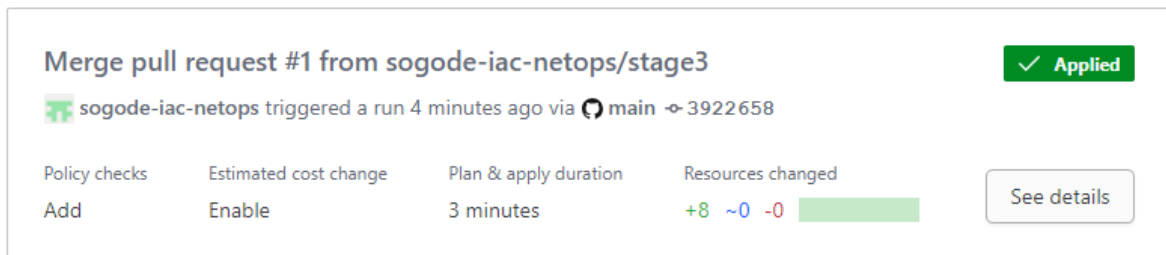
Verify Terraform Cloud

Next, we're logging back into Terraform Cloud to see if the merge triggered another run, and whether it ran successfully.

Go to the `aws-nac` workspace and check the merge was implemented.

Latest Run

[View all runs](#)

A screenshot of the Terraform Cloud interface showing the "Latest Run" section. The title is "Merge pull request #1 from sogode-iac-netops/stage3" with a green "Applied" status badge. Below the title, it says "sogode-iac-netops triggered a run 4 minutes ago via main -> 3922658". There are four columns of metrics: "Policy checks" (Add), "Estimated cost change" (Enable), "Plan & apply duration" (3 minutes), and "Resources changed" (+8 ~0 -0). A "See details" button is on the right.

Policy checks	Estimated cost change	Plan & apply duration	Resources changed
Add	Enable	3 minutes	+8 ~0 -0

Review

Summary

We have now achieved the following

1. Deployed network infrastructure as code
2. Collaborated between 2 teams
3. Used GitOps for Authorization
4. Deployed approved code to production without interaction

We have successfully implemented CI/CD for networking in AWS:

Continuous Integration refers to the automatic process of integrating code.

By adding collaborators, we are enabling colleagues/partners to add on to our infrastructure. They create Pull Requests which are reviewed by the Network Operators.

This process can be automated by configuring tests against coding standards, syntax linters and specific guard rails however that is beyond the scope of this book.

Also, in a real-world example, we would cut the infrastructure up into different projects such as a separate backbone and then publish the Terraform outputs to developers for them to easily access the required IDs of the resources they need for their infrastructure to connect to.

Continuous Deployment refers to the automatic deployment of the code.

Once the PR to the main branch is approved, Terraform is immediately and automatically triggered to deploy the changes into production.

There is much more to add, such as observability, guard rails, 'illegal' change monitoring etc. however this book provides a great start.