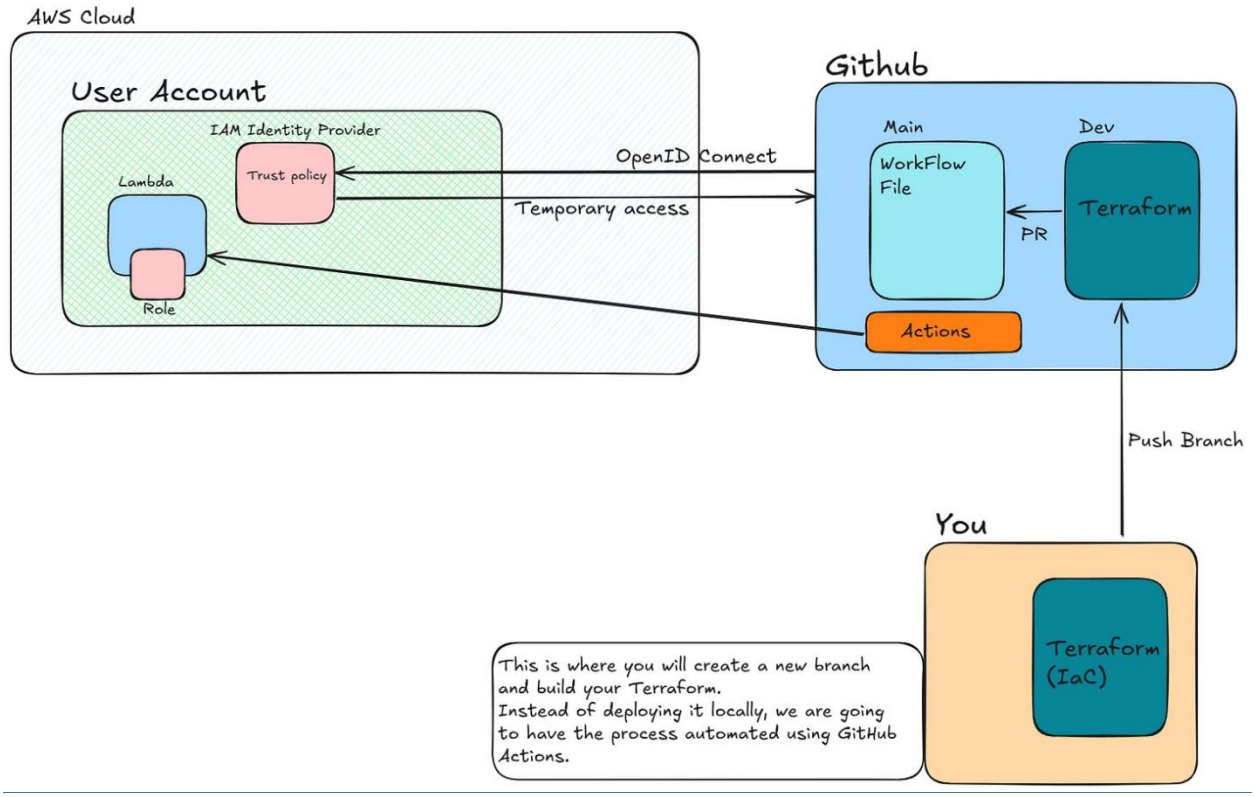
This project was my first step into Dev**Sec**Ops and it's a super easy one to get your head around.

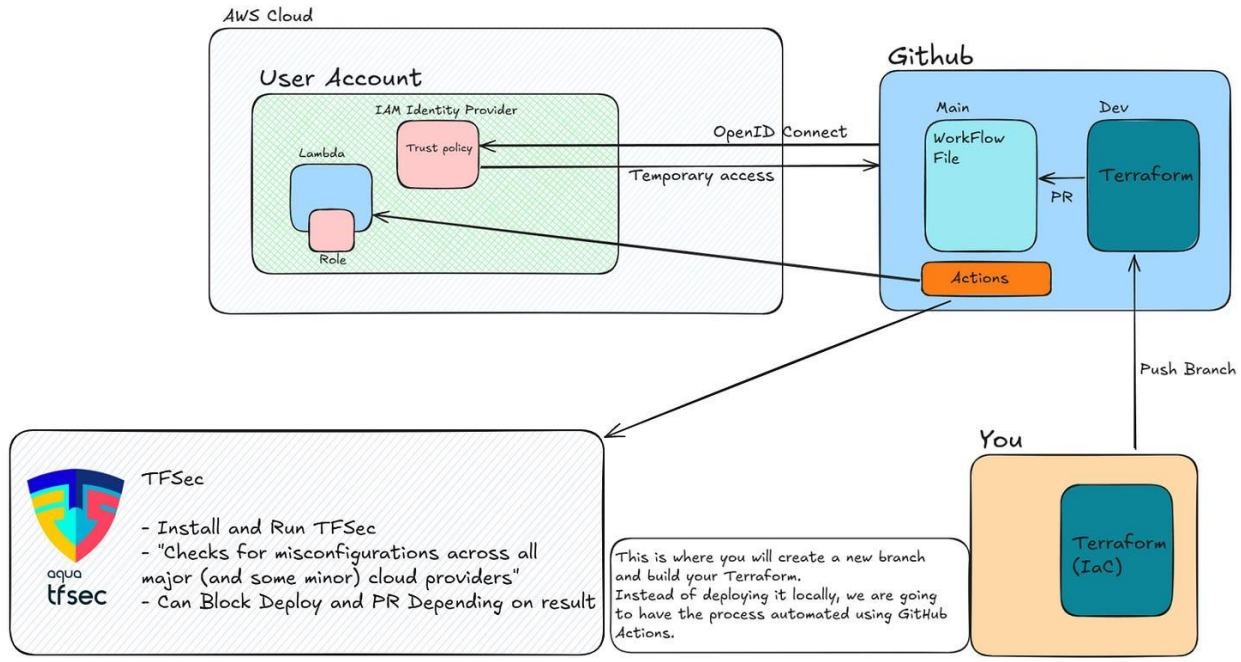
We are going to use Terraform to deploy some simple AWS Resources using a Github Actions as our Pipeline, with an additional security step to scan for misconfigurations in our IaC.

I've already build the basic foundations for this which you can find here: **LINK**



I really recommend following along with the above project before going ahead, it's going to walk you through how to get setup with deploying Terraform in your AWS Account

What we are doing here :



Let's begin.... ⬇️

I spend my evenings and weekends creating these projects to help you launch a career in Cloud Security. It's all made possible by your support, and it keeps me caffeinated with flat whites!

If you're searching for the perfect starting point in DevSecOps or cloud security, this is it. Consider becoming a paid subscriber to unlock this project and dozens more.

If you're reading this, thank you! Your support means the world 🚀

As you can see, we are still deploying Terraform through a Github Action but this time before we deploy we add a security step.

This step uses TFSec and:

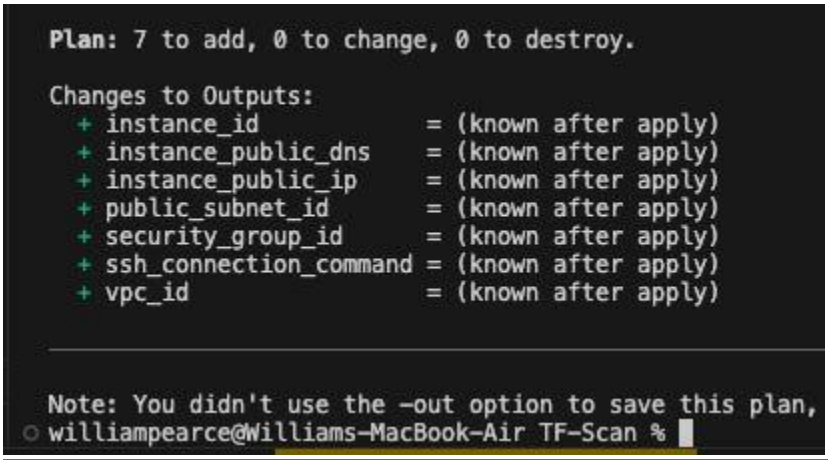
- ☁️ *Checks for misconfigurations across all major (and some minor) cloud providers*
- ❔ *Scans modules (local and remote)*
- ➕ *Evaluates HCL expressions as well as literal values*
- *And a ton more*

^ From the official GitHub Link Here ^

Basically, for our purposes, we want to quickly establish if the infrastructure we are about to deploy contains security misconfigurations.

Step One: Build

If you followed along with the first project, you should by now have a new repo with some Terraform ready to deploy into your AWS estate. You can really build what you like here, but let's run an init/plan just as **a** quick baseline check:



Here's what I'm building:

```
Network Infrastructure:

1 VPC (10.0.0.0/16)
1 Public subnet (10.0.1.0/24)
1 Internet Gateway
1 Route table (routes traffic to internet)

Security:

1 Security group with rules:

SSH (port 22) - from your IP only
HTTP (port 80) - from anywhere
HTTPS (port 443) - from anywhere
All outbound traffic allowed

Compute:

1 EC2 instance (t3.micro)
20GB encrypted root volume
Apache web server auto-installed
Publicly accessible via IP address
```

Step Two: Local Scan

To start off simple let's run TFSec locally, in a Docker image to get grips with the output.

Install Docker

and from the same directory you ran your Terraform plan run:

```
docker run --rm -it -v "$(pwd):/src" aquasec/tfsec /src
```

- **docker run** – Run a container.
- **--rm** – Automatically delete the container after it exits.
- **-it** – Run interactively with a terminal.
- **-v "$(pwd):/src"** – Mount your current directory into the container at /src.
- **aquasec/tfsec** – Use the tfsec image from Aqua Security (Terraform security scanner).
- **/src** – Tell tfsec to scan the mounted directory (your current folder).

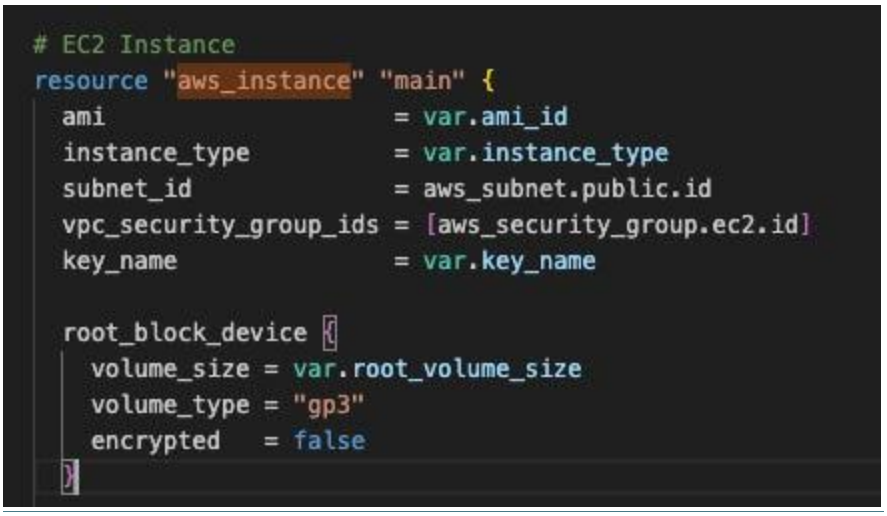
```
Result #7 MEDIUM VPC Flow Logs is not enabled for VPC

  main.tf:24-33

   24    resource "aws_vpc" "main" {
   25      cidr_block           = var.vpc_cidr
   26      enable_dns_hostnames = true
   27      enable_dns_support   = true
   28
   29      tags = {
   30        Name        = "${var.project_name}-vpc"
   31        Environment = var.environment
   32      }
   33    }

        ID aws-ec2-require-vpc-flow-logs-for-all-vpcs
    Impact Without VPC flow logs, you risk not having enough information about network traffic flow to investigate incidents or identify security issues.
Resolution Enable flow logs for VPC

More Information
- https://aquasecurity.github.io/tfsec/v1.28.14/checks/aws/ec2/require-vpc-flow-logs-for-all-vpcs/


timings
-----------------------------------------
disk i/o            750.916µs
parsing             673.999µs
adaptation          1.091958ms
checks              2.300833ms
total               4.817706ms

counts
-----------------------------------------
modules downloaded  0
modules processed   1
blocks processed    28
files read          3

results
-----------------------------------------
passed              7
ignored             0
critical            4
high                2
medium              1
low                 0
```

.....nice, some results. We can go through them one by one and address them before we ever push to our repo.

Let's take this for example, it's given me a warning about the root volume attached to my EC2 instance not being encrypted:

```
Result #6 HIGH Root block device is not encrypted.

  main.tf:138

  128    resource "aws_instance" "main" {
  ...
  138  [     encrypted   = false (false)
  ...
  147    }
```

See where it states: encrypted = false



```
# EC2 Instance
resource "aws_instance" "main" {
  ami                    = var.ami_id
  instance_type          = var.instance_type
  subnet_id              = aws_subnet.public.id
  vpc_security_group_ids = [aws_security_group.ec2.id]
  key_name               = var.key_name

  root_block_device {
    volume_size = var.root_volume_size
    volume_type = "gp3"
    encrypted   = false
  }
```

Hopefully you're starting to see what this tool can do and how useful it would be in deployments

---

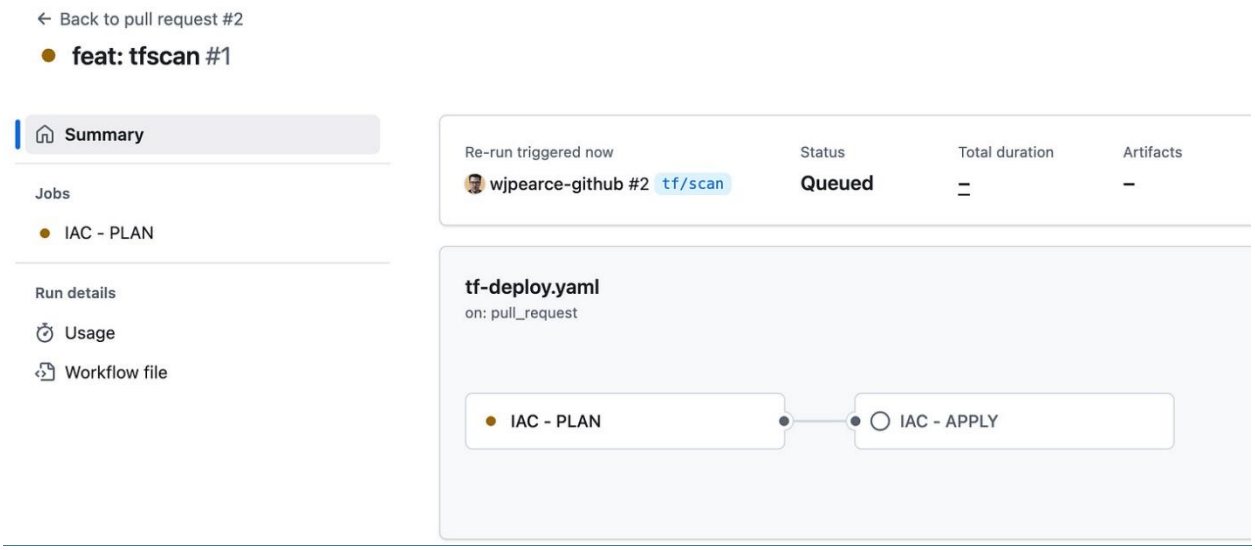
Step Three: Action Scan

Cool, this is what it does, but we don't want it to run Terraform locally, we want to apply it as part of a pipeline. It's very rare you would deploy Terraform locally in an org against production for reasons we can discuss another time.

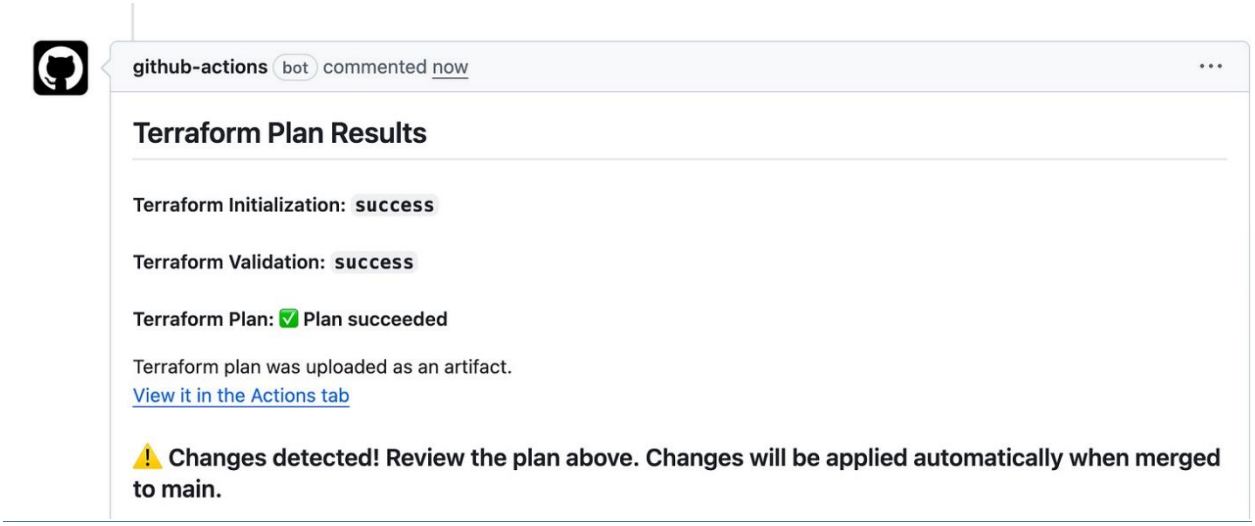
Let's build a GitHub Actions workflow. Again, following the project at the top of the article, you should be in a position where you have a GitHub Action ready to deploy Terraform into your AWS estate.

Here's a reminder of what that looks like:

- **Pull Request**: Run plan, post comment for review
- **Push to main** (merged PR): Run plan then apply to actually deploy



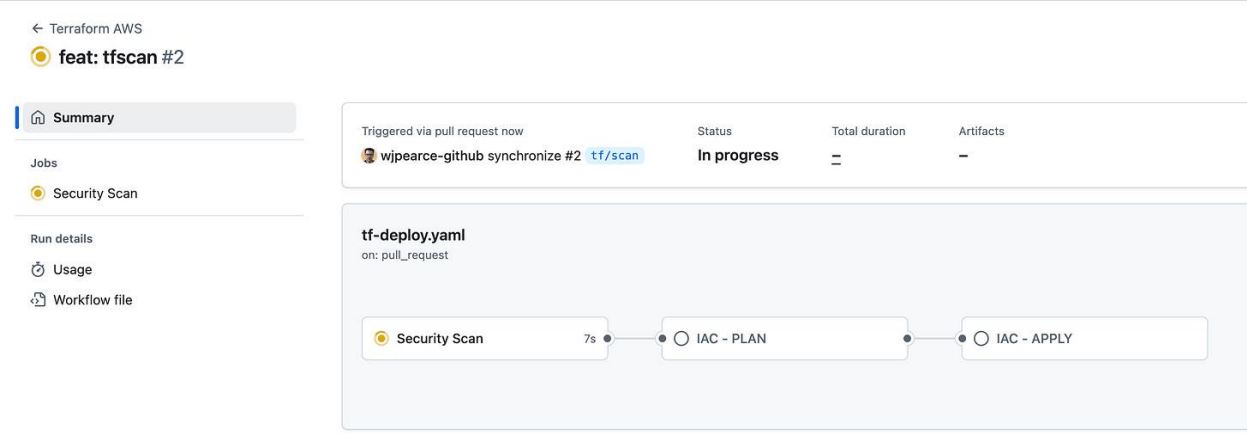
Nice, that works as expect, if I merge this to main it will deploy the infrastructure we've defined...



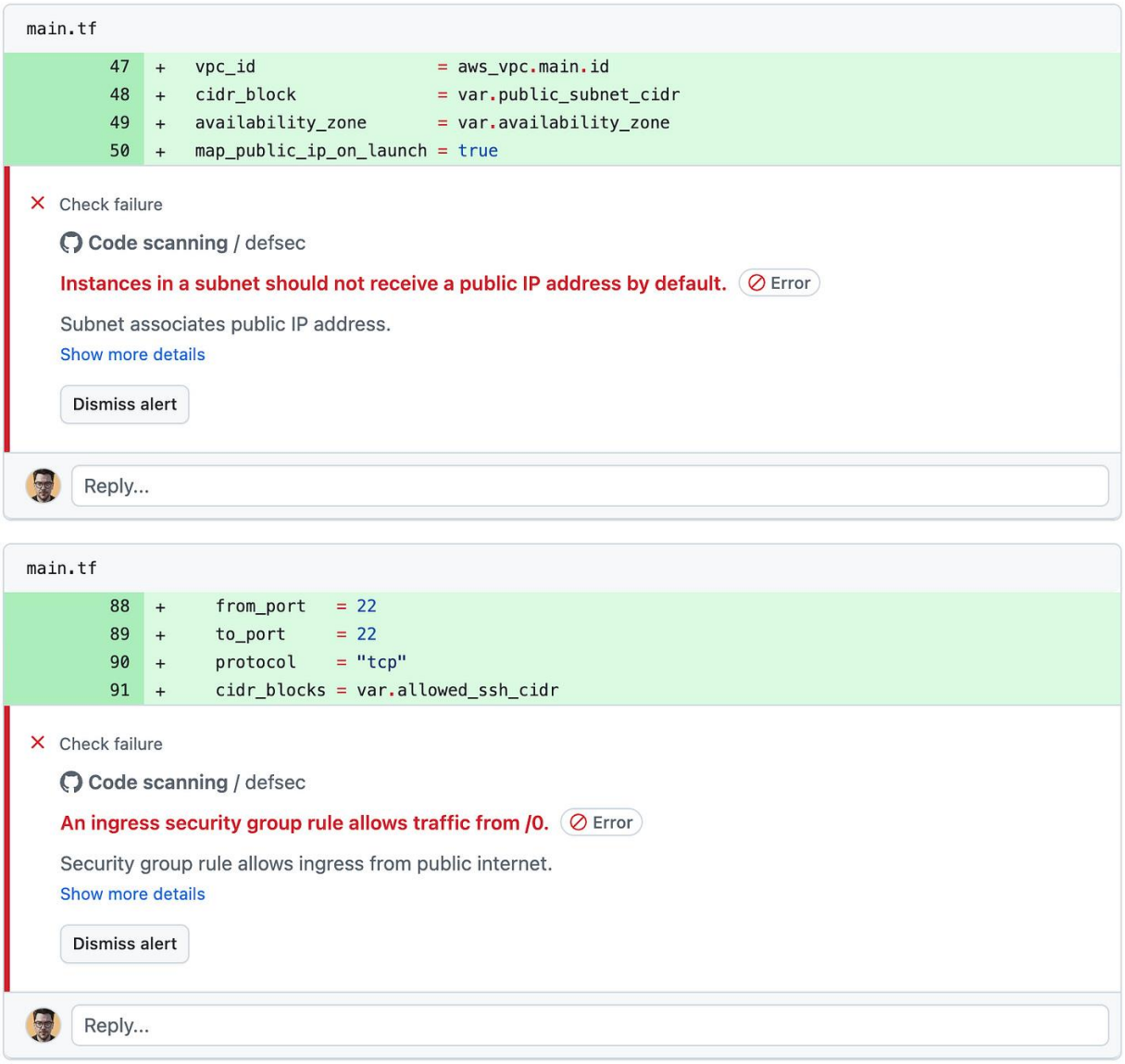
Let's go back and add that security step by adding this step to your GitHub Action

```yaml
steps:
  - name: Git Clone the Repo
    uses: actions/checkout@v4

  - name: Run tfsec
    uses: aquasecurity/tfsec-action@v1.0.3
    with:
      soft_fail: false
      format: sarif
      additional_args: --out tfsec-results.sarif

  - name: Upload tfsec results to GitHub Security
    uses: github/codeql-action/upload-sarif@v3
    if: always()
    with:
      sarif_file: tfsec-results.sarif

  - name: Run tfsec with standard output
    uses: aquasecurity/tfsec-action@v1.0.3
    with:
      soft_fail: false
```

As you can see it's running a new step, our security scan:



and right in the Pull Request, utilising the GitHub Code Scanning feature, because we uploaded our results in a SARIF format we have our security results:



Congrats! You've just added security to a deployment pipeline, pretty easy right?! 🎉

This really is the **Sec** in Dev**Sec**Ops and the **Security** in Cloud **Security**. If you're looking for a beginner project for either of those fields **this is it.**