The chosen structure is as follows:

```
css
Copy code
vannp_oblig02/
├── main.tf
├── variables.tf
├── outputs.tf
├── terraform.tfvars
├── README.md
├── report.pdf
├── modules/
│   ├── app_service/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── database/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── load_balancer/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── networking/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   └── storage/
│       ├── main.tf
│       ├── variables.tf
│       └── outputs.tf
└── .github/
    └── workflows/
        ├── deploy.yml
        └── devstageprodapp.yml
        ├── tflint.yml
        └── tfsec.yml
        ├── validate.yml
```

Explanation and Justification:

- **Root Level Files**:
  - `main.tf`, `variables.tf`, `outputs.tf`, and `terraform.tfvars` serve as the primary entry points for Terraform configurations. These files define the overarching infrastructure, variable definitions, and outputs required by different modules.
  - `README.md` and `report.pdf` provide documentation and project overview.
- **Modules Directory**:
  - The `modules/` directory is divided into specific resource types, such as `app_service`, `database`, `load_balancer`, `networking`, and `storage`.
  - Each module has its own `main.tf`, `variables.tf`, and `outputs.tf` files, which encapsulate the configuration for that specific resource. This modular approach enables reusability and isolation of resource definitions, allowing independent adjustments and easy integration across different environments.
- **.github Directory**:
  - This directory contains GitHub Actions workflows for CI/CD (`deploy.yml`, `devstageprodapp.yml`) and code quality checks (`tflint.yml`, `tfsec.yml`, `validate.yml`).
  - Automating tasks like validation, linting, and security checks ensures code consistency and enhances quality across development stages (dev, stage, prod).

This structure supports modularity and scalability, which are essential for large infrastructure projects where multiple resources are interdependent. It provides an organized and flexible foundation, making it easier to manage resources and workflows across different environments.

Challenges Faced

Backend Configuration Issues:

- Initially, there were challenges configuring the backend storage in Azure. Issues included missing or misconfigured `resource_group_name` and `storage_account_name`, resulting in `404` errors and failed state retrieval.
- I have reviewed and verified each backend parameter. Created necessary Azure resources manually when needed, allowing Terraform to recognize and configure the backend correctly.

Errors with CI/CD Workflow (devstageprodapp.yml):

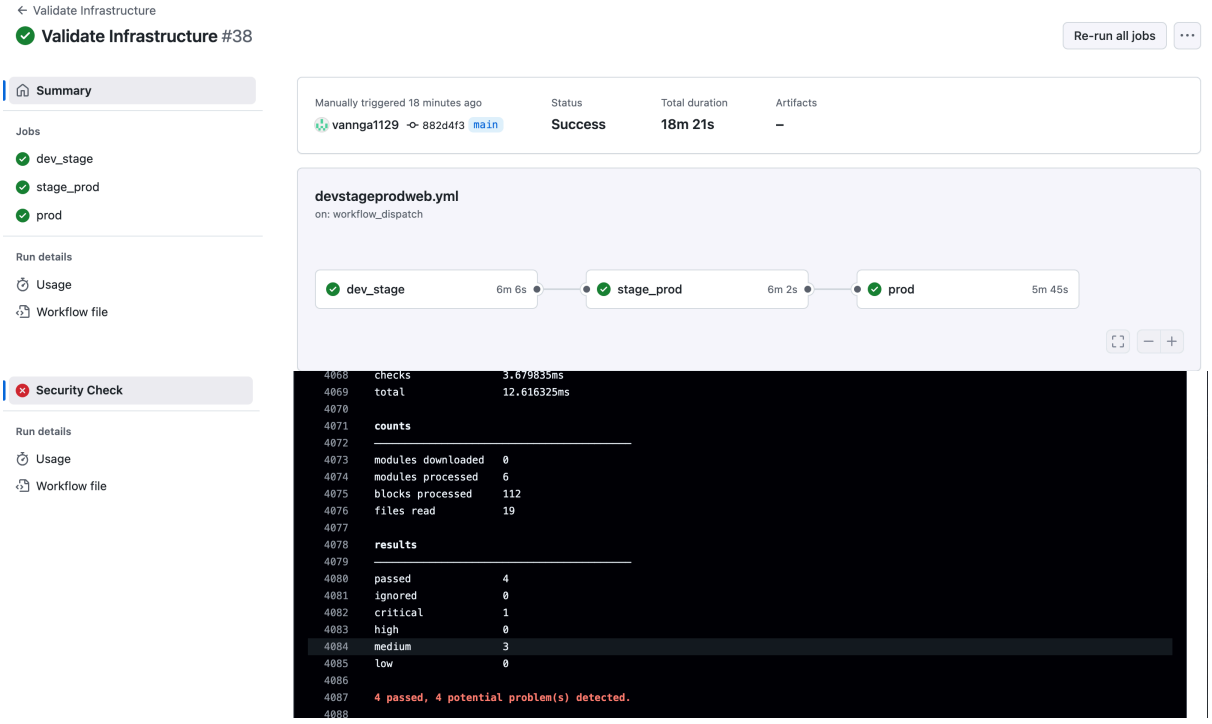
- The `devstageprodapp.yml` workflow file presented multiple errors due to variable misconfigurations and unsupported attributes. Debugging was complex, as errors were spread across different stages.
- I have reviewed the code, using GitHub Actions logs to identify specific errors. Refined variable declarations in the `variables.tf` and `terraform.tfvars` files and fixed unsupported attributes in the module outputs. Additionally, introduced modular testing to isolate errors in each workflow stage.

Code Quality and Security Checks:

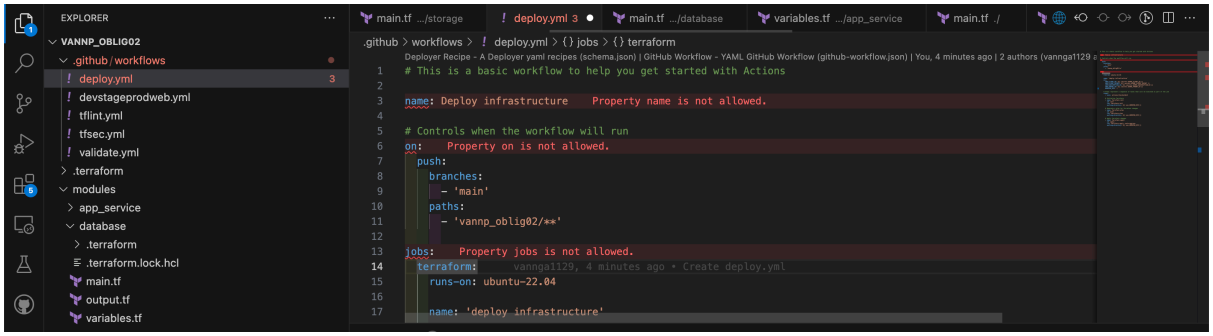
- Implementing `tflint` and `tfsec` checks initially yielded numerous warnings and errors, especially around deprecated interpolation syntax and missing required attributes.
- I have upgraded deprecated syntax, such as replacing interpolation-only expressions with direct variable references. Addressed all flagged security issues to ensure compliance with best practices, further improving code quality and maintainability.

Potential Improvements and Optimizations

1. Adding more specific error-handling workflows and enhanced logging would simplify debugging in CI/CD. Logging specific errors related to backend or workflow failures can speed up troubleshooting.
2. Using a remote state data source to retrieve values across environments (e.g., dev, stage, prod) could streamline dependencies and reduce the need for environment-specific hardcoding in the `devstageprodapp.yml` workflow.
3. Adjust resource configurations (e.g., downgrading `app_service` SKU for dev environments) to reduce costs in non-production environments, particularly useful when managing multiple deployments.



Besides that, in the end I have faced with this problem and I couldn't manage to fix it:


EXPLORER
VANNP_OBLIG02
.github / workflows
deploy.yml 3
devstageprodweb.yml
tflint.yml
tfsec.yml
validate.yml
.terraform
modules
app_service
database
.terraform
.terraform.lock.hcl
main.tf
output.tf
variables.tf
main.tf .../storage
deploy.yml 3
main.tf .../database
variables.tf .../app_service
main.tf ./
.github > workflows > deploy.yml > { } jobs > { } terraform
Deployer Recipe - A Deployer yaml recipes (schema.json) | GitHub Workflow - YAML GitHub Workflow (github-workflow.json) | You, 4 minutes ago | 2 authors (vannga1129
This is a basic workflow to help you get started with Actions
name: Deploy infrastructure Property name is not allowed.
Controls when the workflow will run
on: Property on is not allowed.
push:
branches:
- 'main'
paths:
- 'vannp_oblig02/**'
jobs: Property jobs is not allowed.
terraform: vannga1129, 4 minutes ago • Create deploy.yml
runs-on: ubuntu-22.04
name: 'deploy infrastructure'