Testing Terraform HCL Modules

How we got here…

- Platform Eng: *We need to deploy a load of resources and make sure it's deployed as we require it and written* ***in code****. Let's use Terraform*
- Mgmt: *That thing you're doing with deployments. Can we offer any of that to allow platform users to* ***re-use*** *it?*
- **Security**: *That thing you're offering to users, can we control certain attributes so users can't do anything bad?*
- Mgmt: *Can we make all of this* ***automated****?*
- Mgmt: *Can we allow users to provide some config to the automation and make it self-service?*
- **Compliance**: *Can we define what controls need to be applied and it be* ***enforced*** *at deployment*
- …

These are not the droids you're looking for

- Not Terraform application tests (Go Testing)
- Not validating the success of our deployments
- This is the presentation about unit testing logic HCL modules

Types of tests

- Unit
  - The atomic operation of a single piece of functionality
- Integration
  - Testing functionality that needs to talk to some other system or application
- Functional/Acceptance
  - When running as a more complete solution (module or app)

Why test?

- Testing is hard
- Testing often takes more time than the code

- Confidence
- Reliability
- Robustness
  - Code is easier to review
  - Small changes in code could have unexpected behaviour elsewhere

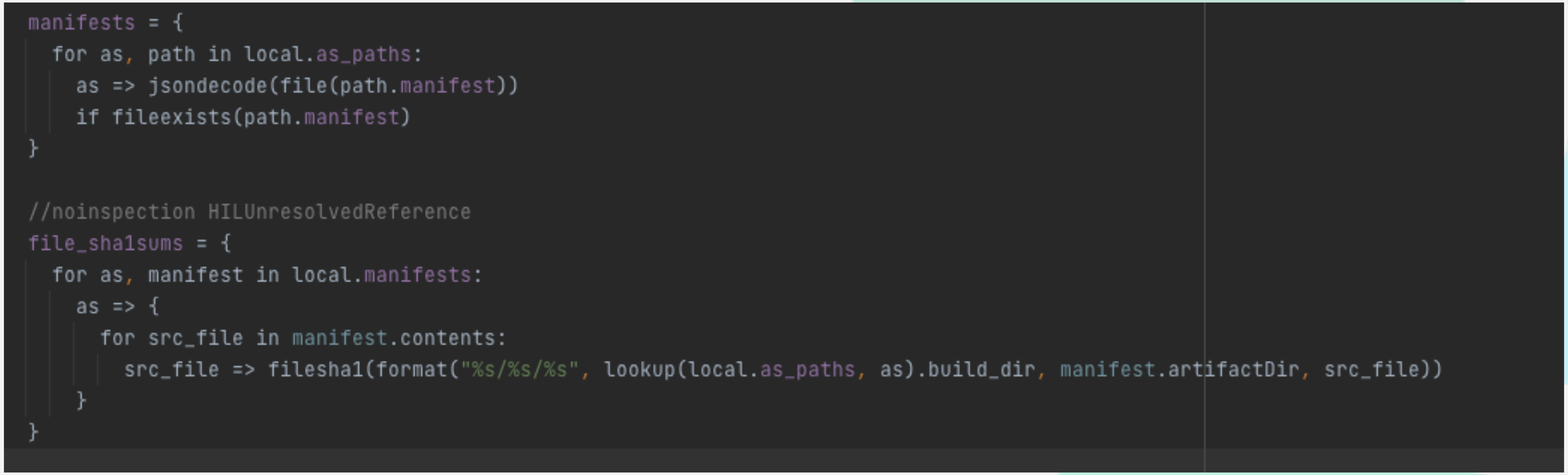
Automated testing

- With well written tests CI/CD is possible
- Reduces overall workload
- Reduces time to delivery
- Reduces incidents
- Increase TIB (Time In Bed)

Writing tests

- TDD (Test Driven Development)
- TDR (Test Driven Refactoring)
- The more flexible & feature rich the language the easier the testing (powerful frameworks)
- Writing a module in HCL does not have such a framework

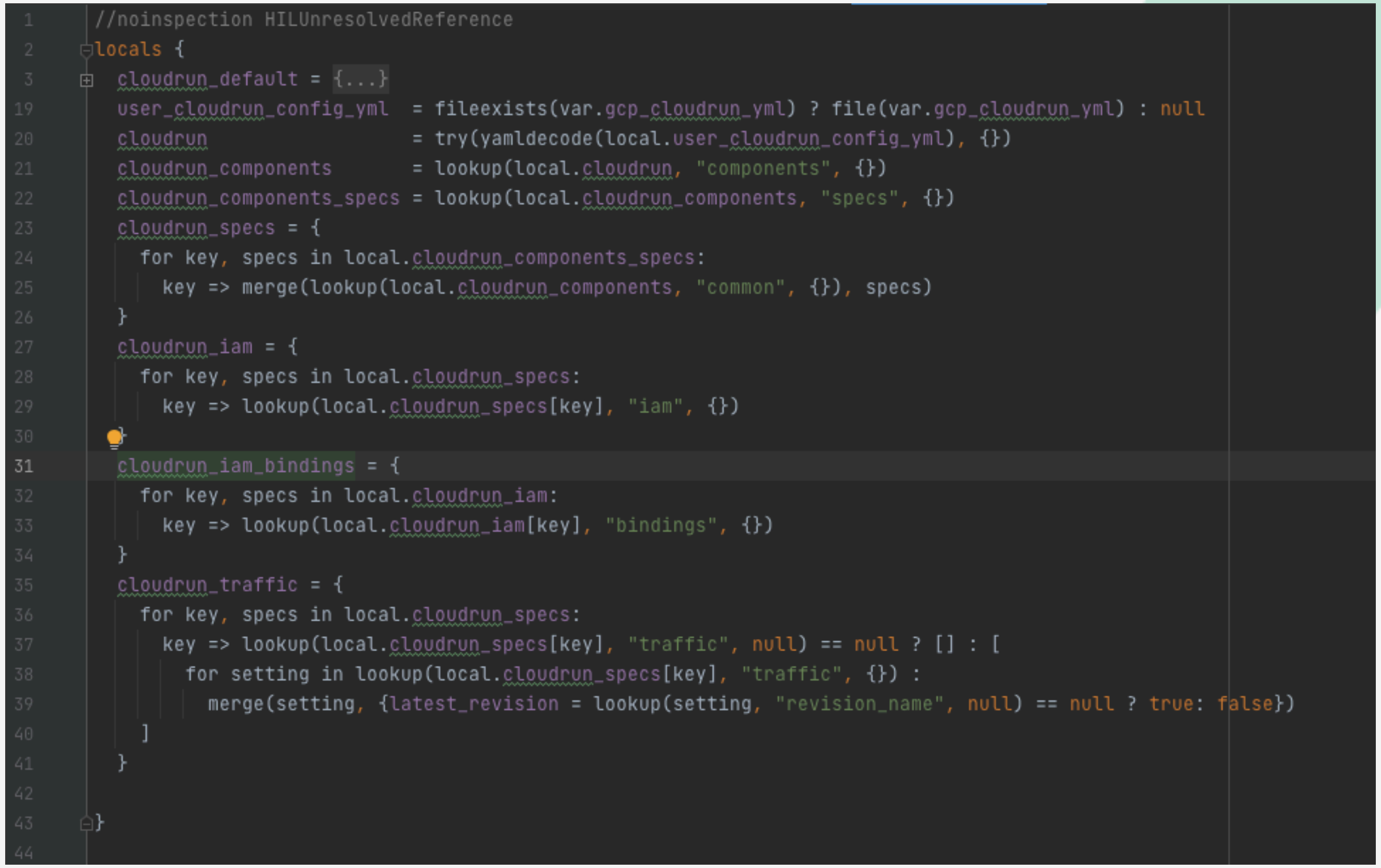
Why test our HCL?



```
manifests = {
  for as, path in local.as_paths:
    as => jsondecode(file(path.manifest))
    if fileexists(path.manifest)
}

//noinspection HILUnresolvedReference
file_sha1sums = {
  for as, manifest in local.manifests:
    as => {
      for src_file in manifest.contents:
        src_file => filesha1(format("%s/%s/%s", lookup(local.as_paths, as).build_dir, manifest.artifactDir, src_file))
    }
}
```

- Now assume that we have much more of this and we’re releasing these for others to consume

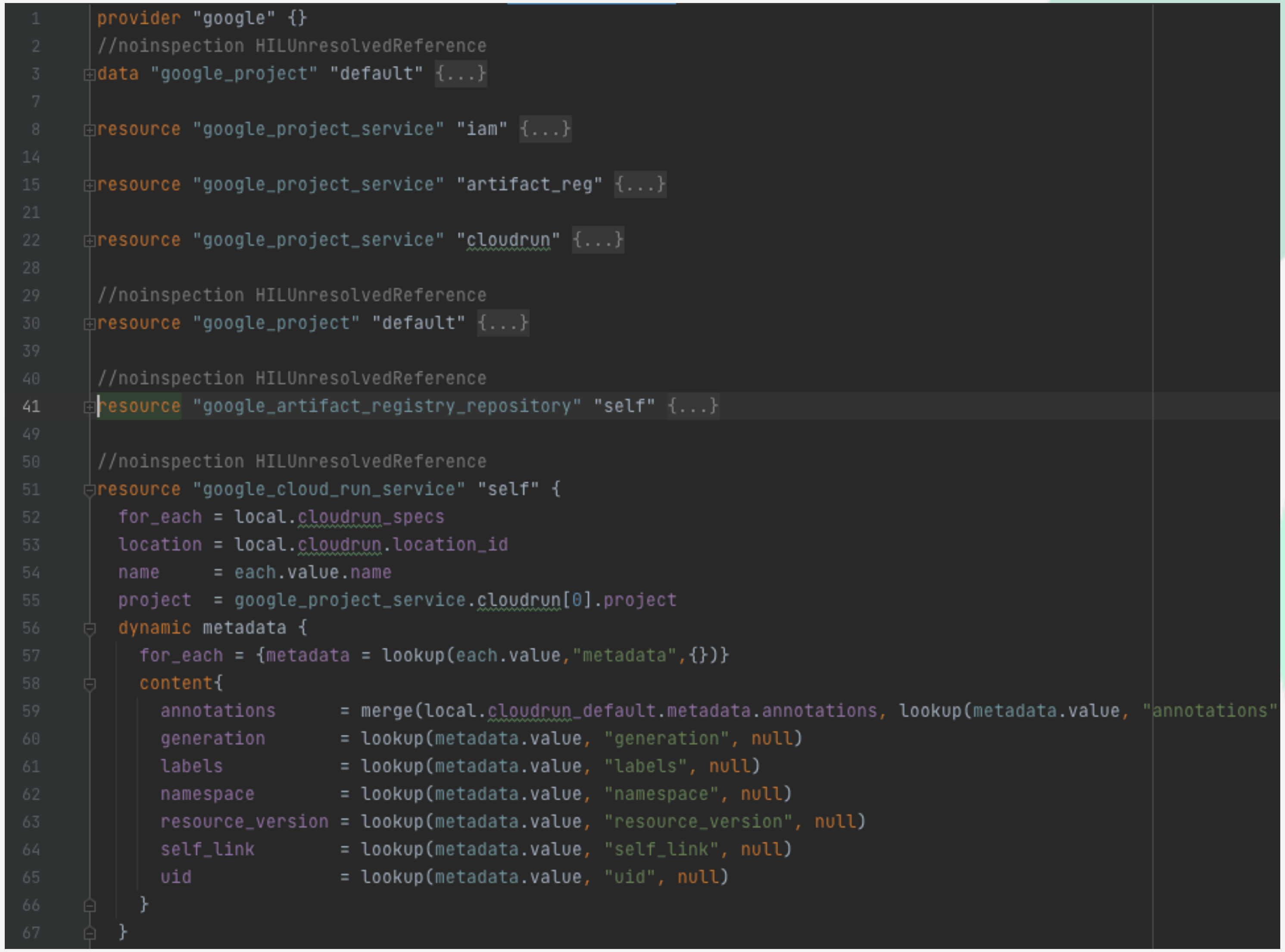
locals.tf



```
//noinspection HILUnresolvedReference
locals {
  cloudrun_default = {...}
  user_cloudrun_config_yml  = fileexists(var.gcp_cloudrun_yml) ? file(var.gcp_cloudrun_yml) : null
  cloudrun                  = try(yamldecode(local.user_cloudrun_config_yml), {})
  cloudrun_components       = lookup(local.cloudrun, "components", {})
  cloudrun_components_specs = lookup(local.cloudrun_components, "specs", {})
  cloudrun_specs = {
    for key, specs in local.cloudrun_components_specs:
      key => merge(lookup(local.cloudrun_components, "common", {}), specs)
  }
  cloudrun_iam = {
    for key, specs in local.cloudrun_specs:
      key => lookup(local.cloudrun_specs[key], "iam", {})
  }
  cloudrun_iam_bindings = {
    for key, specs in local.cloudrun_iam:
      key => lookup(local.cloudrun_iam[key], "bindings", {})
  }
  cloudrun_traffic = {
    for key, specs in local.cloudrun_specs:
      key => lookup(local.cloudrun_specs[key], "traffic", null) == null ? [] : [
        for setting in lookup(local.cloudrun_specs[key], "traffic", {}) :
          merge(setting, {latest_revision = lookup(setting, "revision_name", null) == null ? true: false})
      ]
  }

}
```

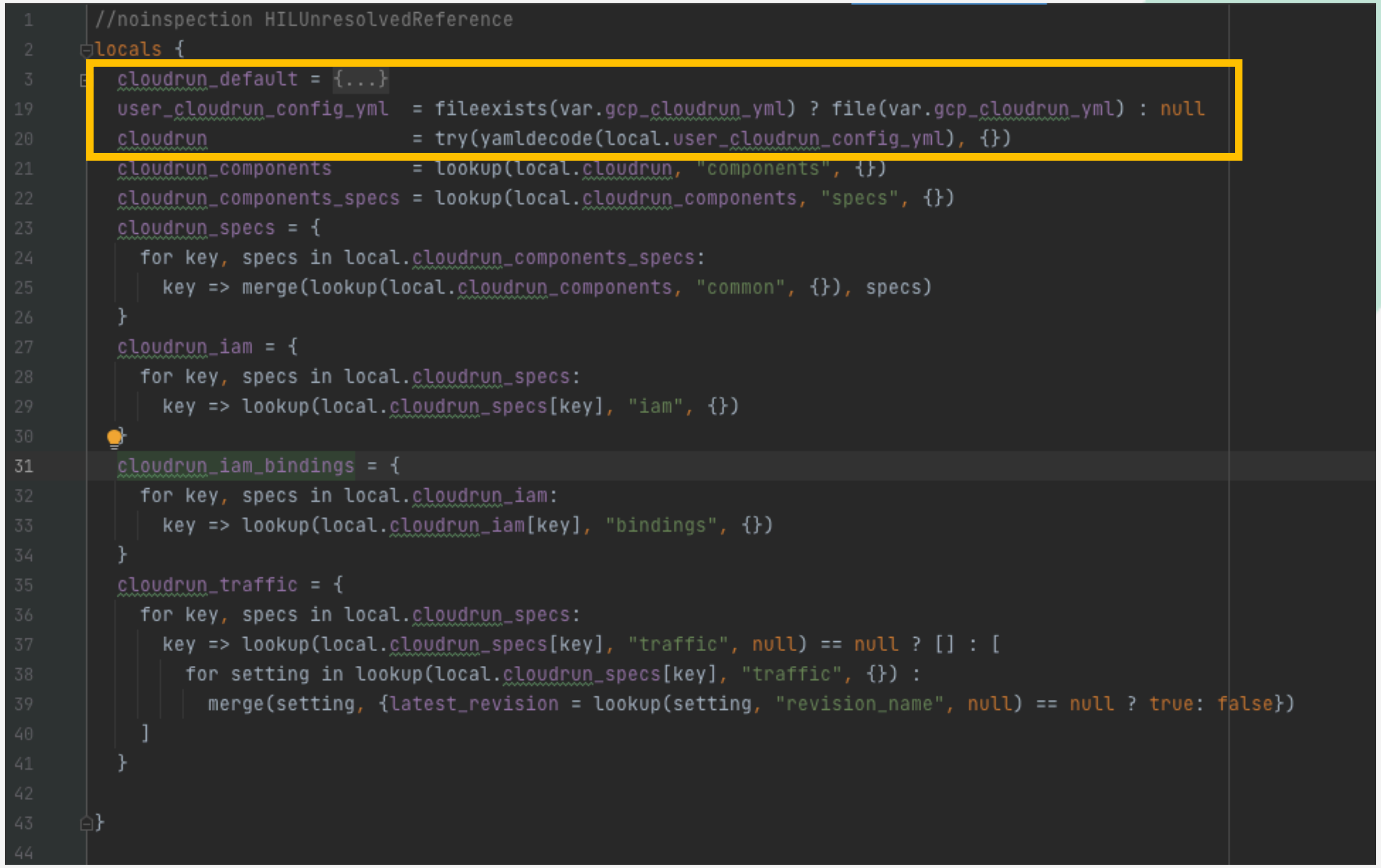
- Separate logic from resource definitions

main.tf



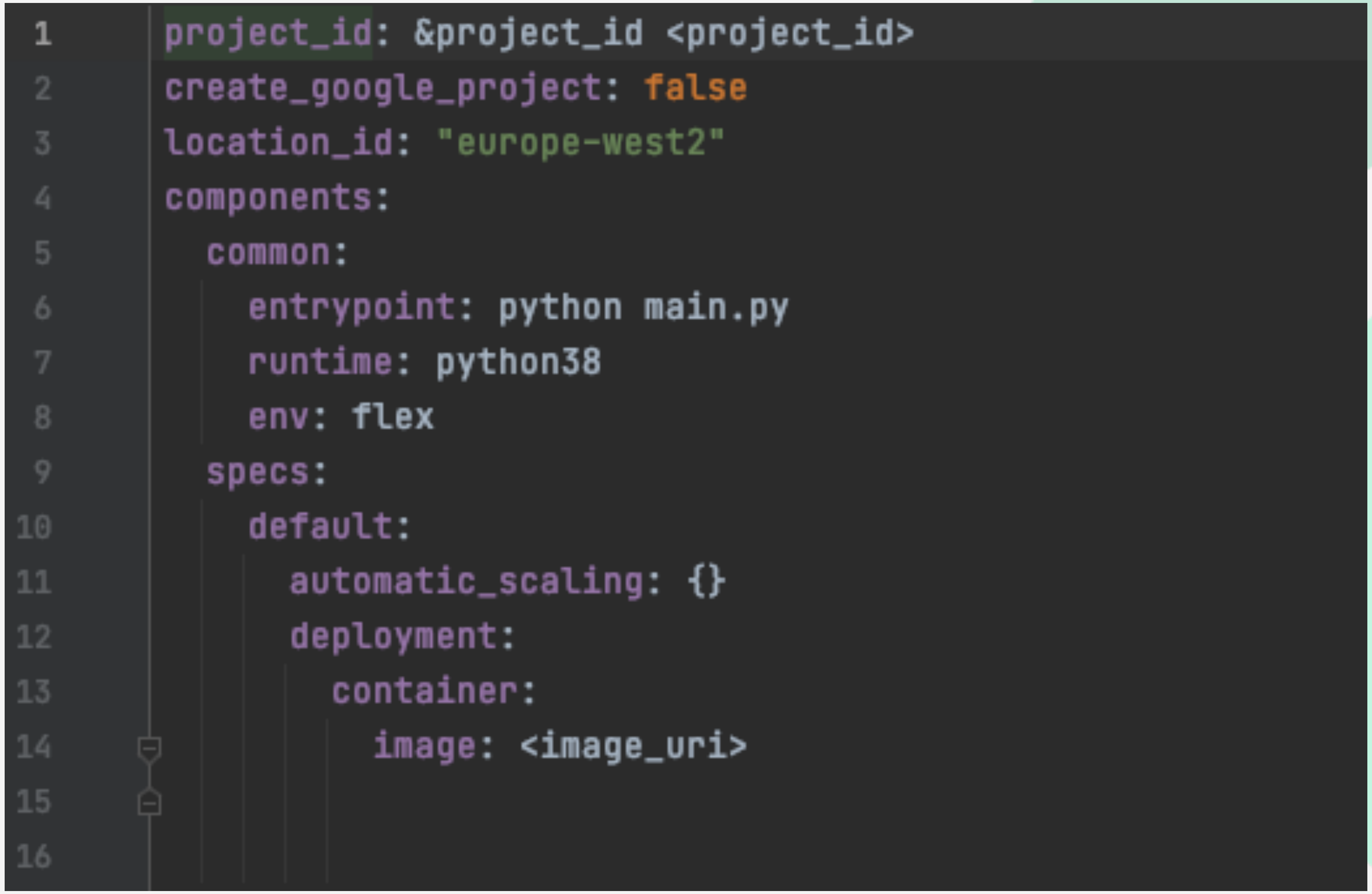
```
provider "google" {}
//noinspection HILUnresolvedReference
data "google_project" "default" {...}

resource "google_project_service" "iam" {...}

resource "google_project_service" "artifact_reg" {...}

resource "google_project_service" "cloudrun" {...}

//noinspection HILUnresolvedReference
resource "google_project" "default" {...}

//noinspection HILUnresolvedReference
resource "google_artifact_registry_repository" "self" {...}

//noinspection HILUnresolvedReference
resource "google_cloud_run_service" "self" {
  for_each = local.cloudrun_specs
  location = local.cloudrun.location_id
  name     = each.value.name
  project  = google_project_service.cloudrun[0].project
  dynamic metadata {
    for_each = {metadata = lookup(each.value,"metadata",{})}
    content{
      annotations      = merge(local.cloudrun_default.metadata.annotations, lookup(metadata.value, "annotations"
      generation       = lookup(metadata.value, "generation", null)
      labels           = lookup(metadata.value, "labels", null)
      namespace        = lookup(metadata.value, "namespace", null)
      resource_version = lookup(metadata.value, "resource_version", null)
      self_link        = lookup(metadata.value, "self_link", null)
      uid              = lookup(metadata.value, "uid", null)
    }
  }
```

locals.tf



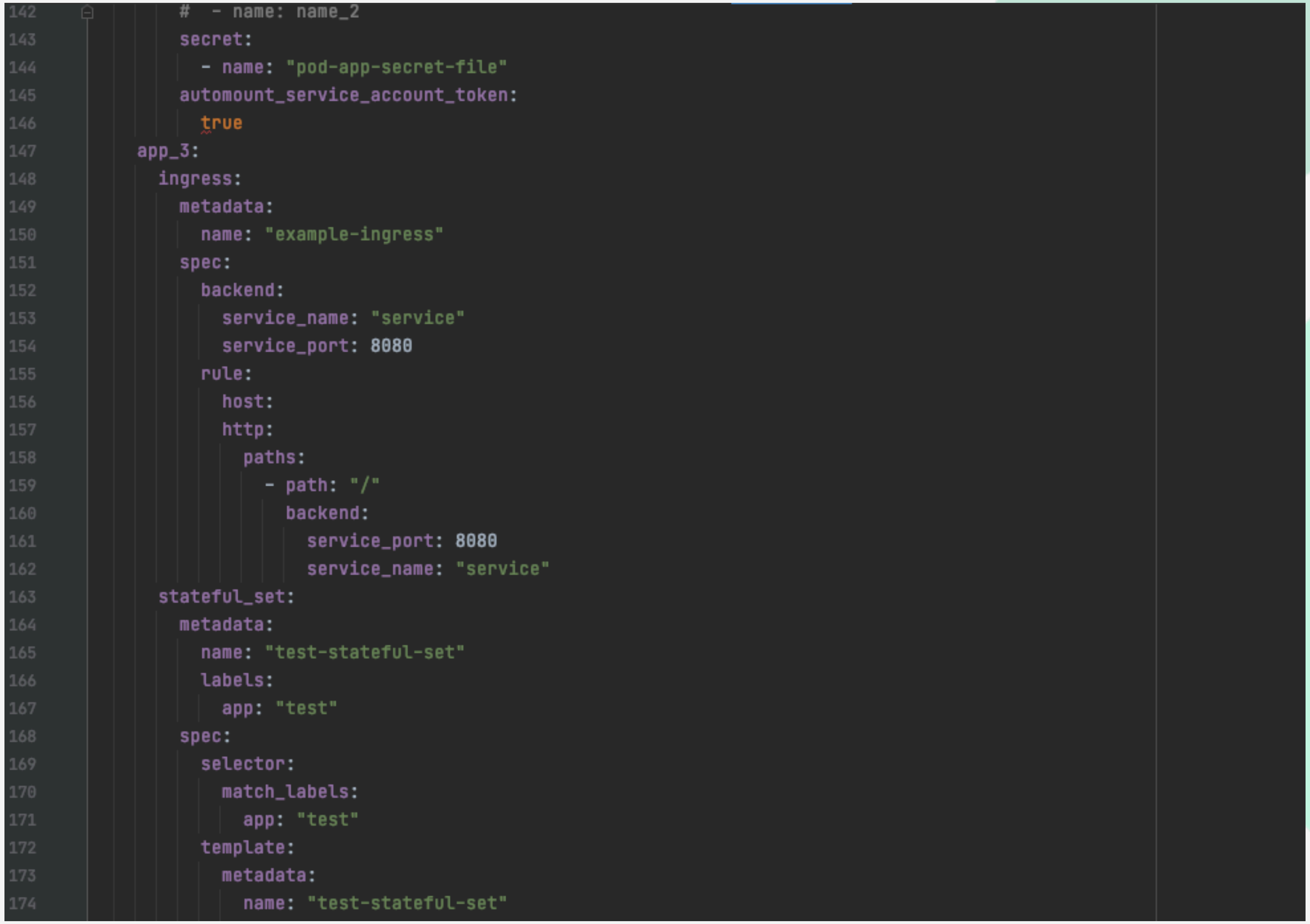
```
//noinspection HILUnresolvedReference
locals {
  cloudrun_default = {...}
  user_cloudrun_config_yml  = fileexists(var.gcp_cloudrun_yml) ? file(var.gcp_cloudrun_yml) : null
  cloudrun                  = try(yamldecode(local.user_cloudrun_config_yml), {})
  cloudrun_components       = lookup(local.cloudrun, "components", {})
  cloudrun_components_specs = lookup(local.cloudrun_components, "specs", {})
  cloudrun_specs = {
    for key, specs in local.cloudrun_components_specs:
      key => merge(lookup(local.cloudrun_components, "common", {}), specs)
  }
  cloudrun_iam = {
    for key, specs in local.cloudrun_specs:
      key => lookup(local.cloudrun_specs[key], "iam", {})
  }
  cloudrun_iam_bindings = {
    for key, specs in local.cloudrun_iam:
      key => lookup(local.cloudrun_iam[key], "bindings", {})
  }
  cloudrun_traffic = {
    for key, specs in local.cloudrun_specs:
      key => lookup(local.cloudrun_specs[key], "traffic", null) == null ? [] : [
        for setting in lookup(local.cloudrun_specs[key], "traffic", {}) :
          merge(setting, {latest_revision = lookup(setting, "revision_name", null) == null ? true: false})
      ]
  }

}

```

../gcp_ae.yml



```yaml
project_id: &project_id <project_id>
create_google_project: false
location_id: "europe-west2"
components:
  common:
    entrypoint: python main.py
    runtime: python38
    env: flex
  specs:
    default:
      automatic_scaling: {}
      deployment:
        container:
          image: <image_uri>


```

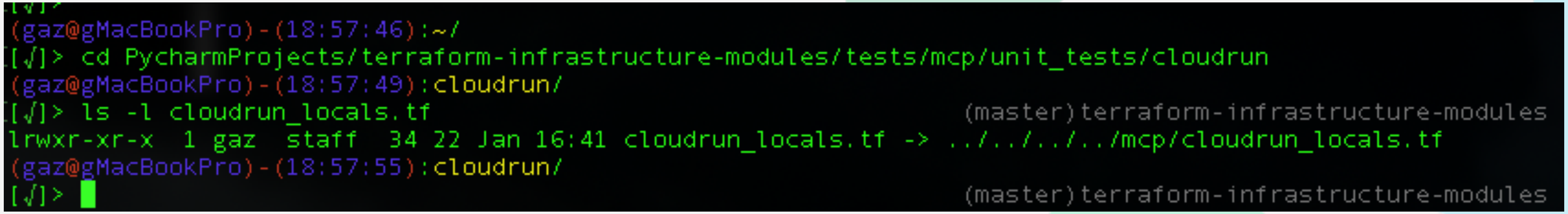
../k8s.yml



```yaml
   # - name: name_2
   secret:
    - name: "pod-app-secret-file"
   automount_service_account_token:
    true
 app_3:
  ingress:
   metadata:
    name: "example-ingress"
   spec:
    backend:
     service_name: "service"
     service_port: 8080
    rule:
     host:
     http:
      paths:
       - path: "/"
        backend:
         service_port: 8080
         service_name: "service"
  stateful_set:
   metadata:
    name: "test-stateful-set"
    labels:
     app: "test"
   spec:
    selector:
     match_labels:
      app: "test"
    template:
     metadata:
      name: "test-stateful-set"
```

Symlink -> ../tests

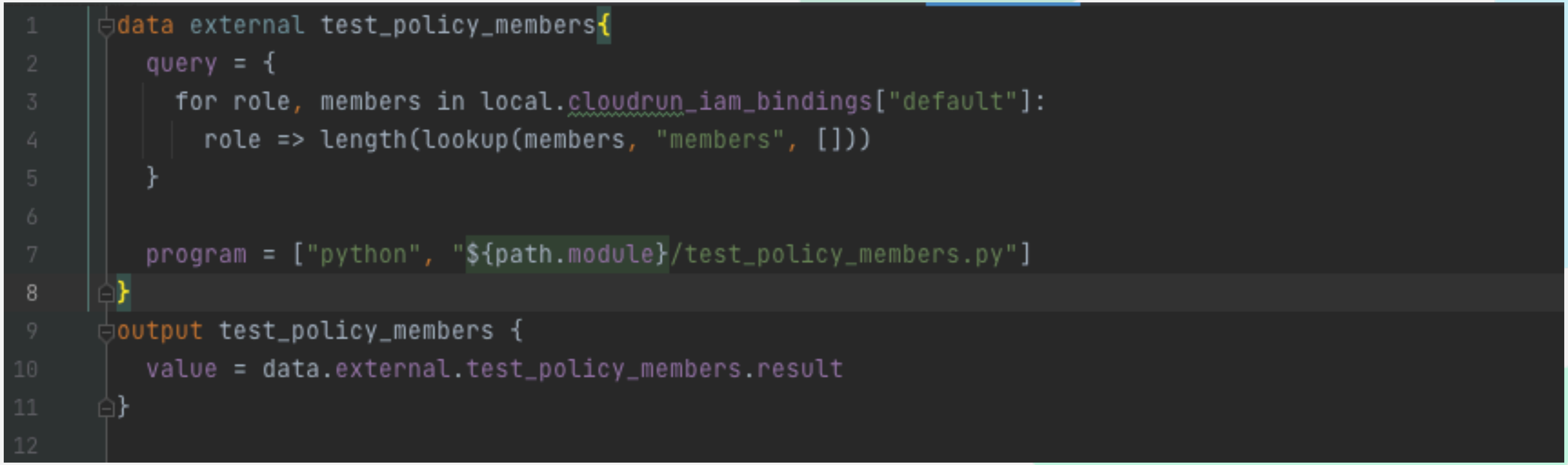
- Link to that file in the tests directory



```
(gaz@gMacBookPro)-(18:57:46):~/
[√]> cd PycharmProjects/terraform-infrastructure-modules/tests/mcp/unit_tests/cloudrun
(gaz@gMacBookPro)-(18:57:49):cloudrun/
[√]> ls -l cloudrun_locals.tf                                 (master)terraform-infrastructure-modules
lrwxr-xr-x  1 gaz  staff  34 22 Jan 16:41 cloudrun_locals.tf -> ../../../../mcp/cloudrun_locals.tf
(gaz@gMacBookPro)-(18:57:55):cloudrun/
[√]>                                                          (master)terraform-infrastructure-modules
```

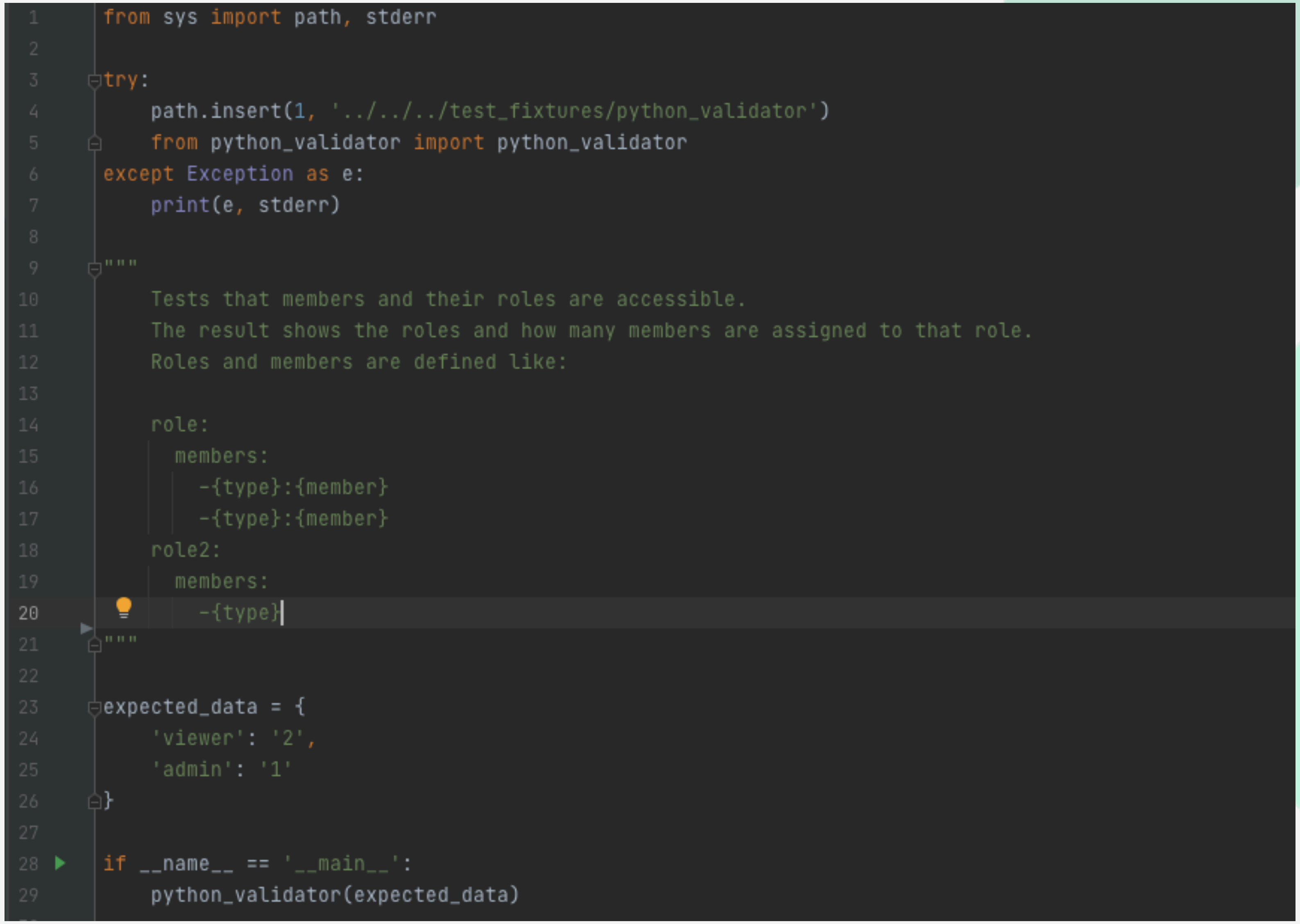
tests/main.tf

- Now we can access those local values in a 'test' main.tf



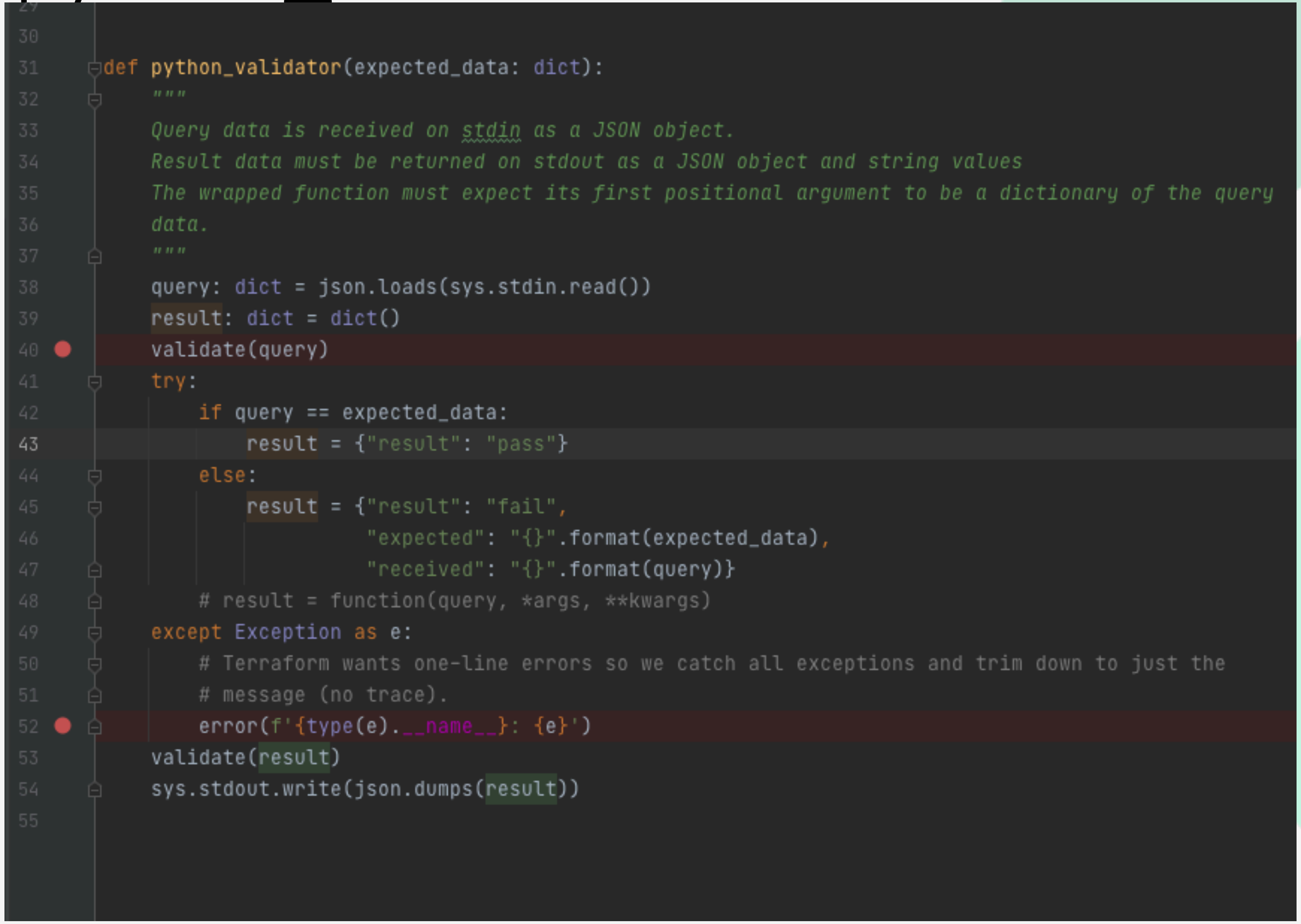
```
data external test_policy_members{
  query = {
    for role, members in local.cloudrun_iam_bindings["default"]:
      role => length(lookup(members, "members", []))
  }

  program = ["python", "${path.module}/test_policy_members.py"]
}
output test_policy_members {
  value = data.external.test_policy_members.result
}
```

Test case



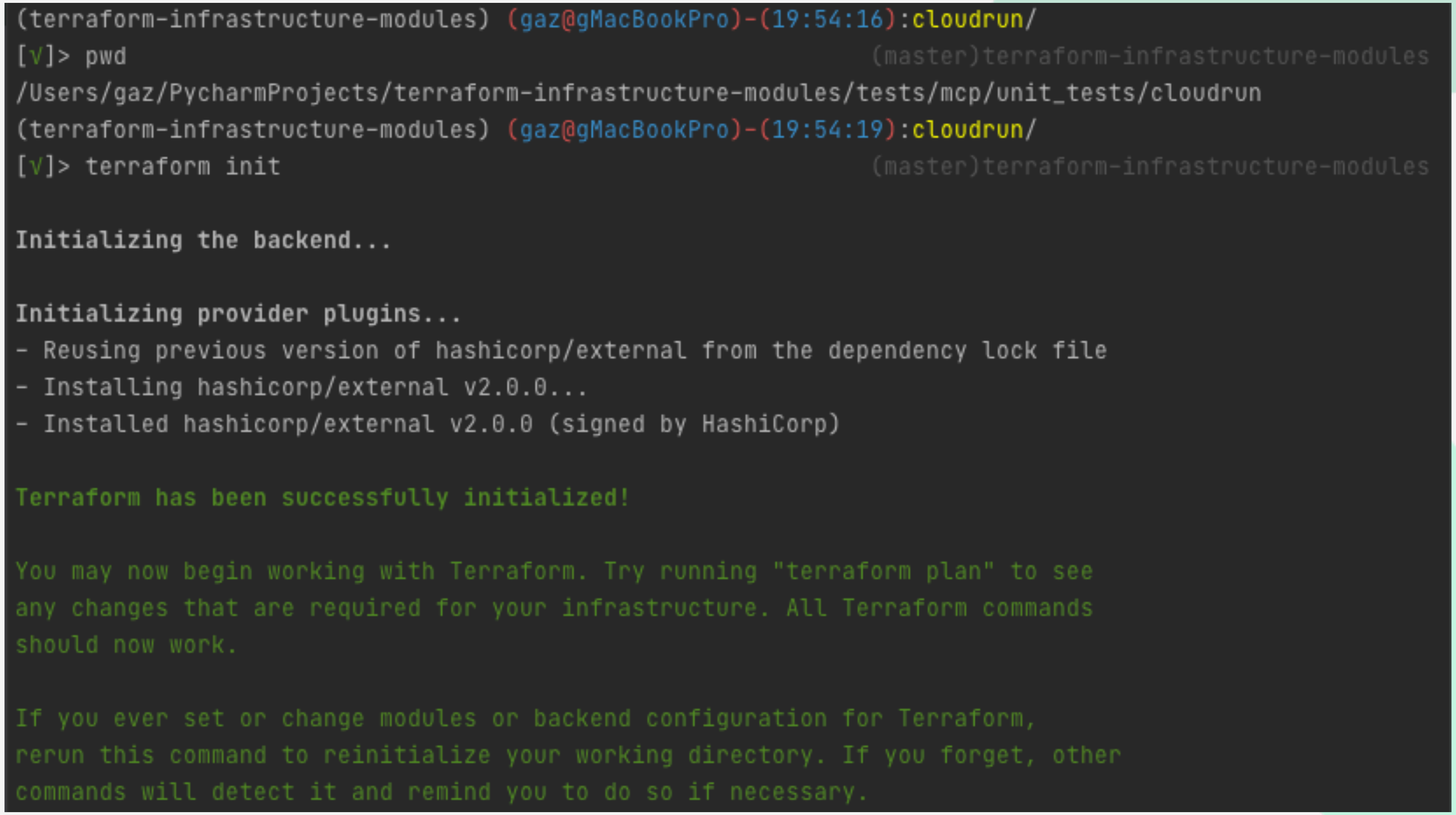
```python
from sys import path, stderr

try:
    path.insert(1, '../../../test_fixtures/python_validator')
    from python_validator import python_validator
except Exception as e:
    print(e, stderr)

"""
    Tests that members and their roles are accessible.
    The result shows the roles and how many members are assigned to that role.
    Roles and members are defined like:

    role:
      members:
        -{type}:{member}
        -{type}:{member}
    role2:
      members:
        -{type}
"""

expected_data = {
    'viewer': '2',
    'admin': '1'
}

if __name__ == '__main__':
    python_validator(expected_data)
```

python_validator



```python
def python_validator(expected_data: dict):
    """
    Query data is received on stdin as a JSON object.
    Result data must be returned on stdout as a JSON object and string values
    The wrapped function must expect its first positional argument to be a dictionary of the query
    data.
    """
    query: dict = json.loads(sys.stdin.read())
    result: dict = dict()
    validate(query)
    try:
        if query == expected_data:
            result = {"result": "pass"}
        else:
            result = {"result": "fail",
                      "expected": "{}".format(expected_data),
                      "received": "{}".format(query)}
        # result = function(query, *args, **kwargs)
    except Exception as e:
        # Terraform wants one-line errors so we catch all exceptions and trim down to just the
        # message (no trace).
        error(f'{type(e).__name__}: {e}')
    validate(result)
    sys.stdout.write(json.dumps(result))
```

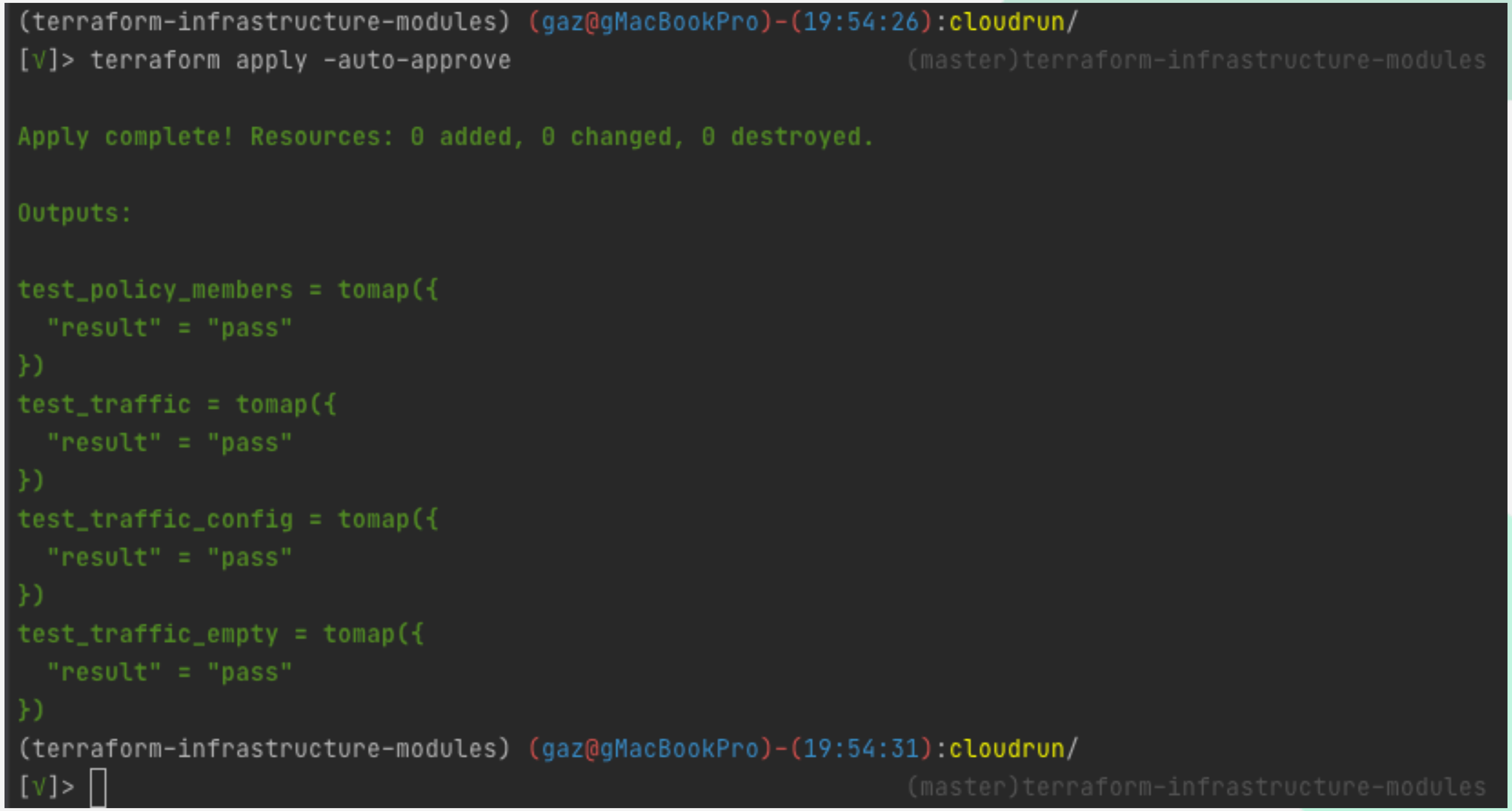
Running the tests from a command line



```
(terraform-infrastructure-modules) (gaz@gMacBookPro)-(19:54:16):cloudrun/
[v]> pwd                                                (master)terraform-infrastructure-modules
/Users/gaz/PycharmProjects/terraform-infrastructure-modules/tests/mcp/unit_tests/cloudrun
(terraform-infrastructure-modules) (gaz@gMacBookPro)-(19:54:19):cloudrun/
[v]> terraform init                                     (master)terraform-infrastructure-modules

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/external from the dependency lock file
- Installing hashicorp/external v2.0.0...
- Installed hashicorp/external v2.0.0 (signed by HashiCorp)

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

mesoform

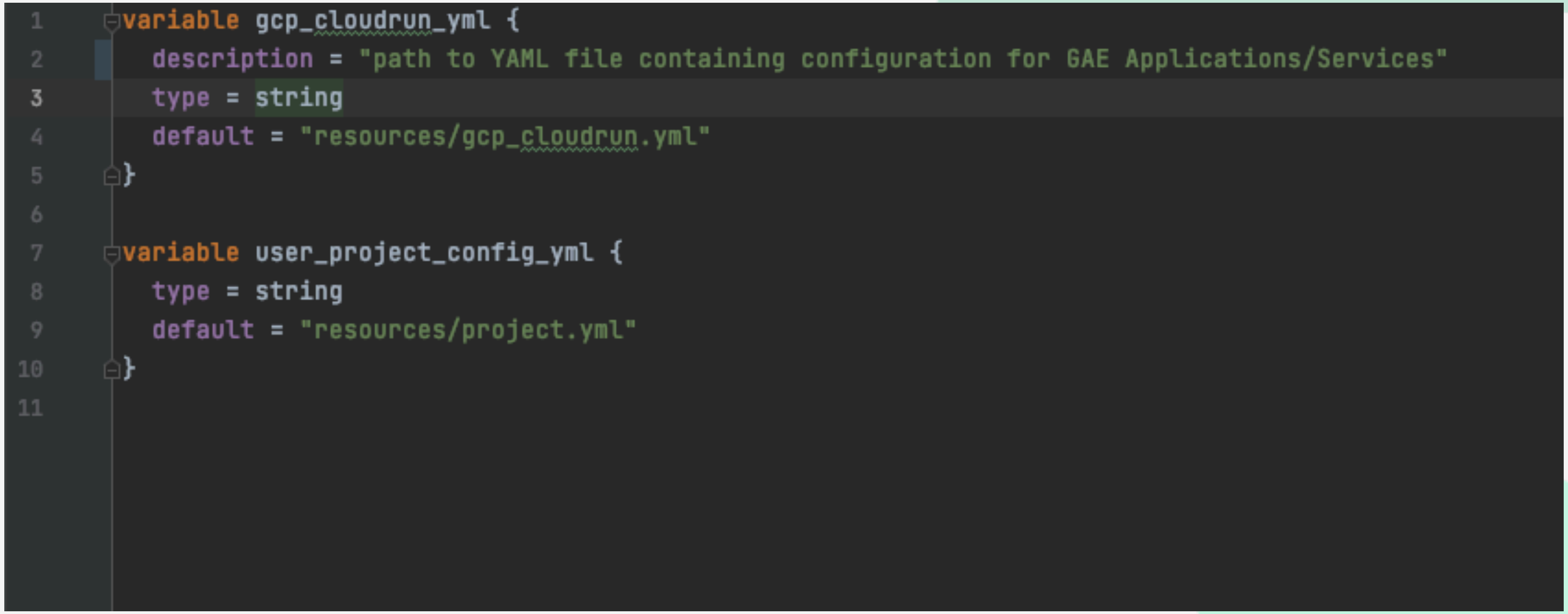
Running the tests from a command line



```
(terraform-infrastructure-modules) (gaz@gMacBookPro)-(19:54:26):cloudrun/
[√]> terraform apply -auto-approve                    (master)terraform-infrastructure-modules

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

test_policy_members = tomap({
  "result" = "pass"
})
test_traffic = tomap({
  "result" = "pass"
})
test_traffic_config = tomap({
  "result" = "pass"
})
test_traffic_empty = tomap({
  "result" = "pass"
})
(terraform-infrastructure-modules) (gaz@gMacBookPro)-(19:54:31):cloudrun/
[√]>                                                  (master)terraform-infrastructure-modules
```

HashiCorp

Test data



```
variable gcp_cloudrun_yml {
  description = "path to YAML file containing configuration for GAE Applications/Services"
  type = string
  default = "resources/gcp_cloudrun.yml"
}

variable user_project_config_yml {
  type = string
  default = "resources/project.yml"
}
```

Automating our tests

- https://github.com/mesoform/terraform-infrastructure-modules/blob/main/.github/workflows/unit_tests.yml
- https://github.com/mesoform/terraform-infrastructure-modules/blob/main/.github/workflows/deploy_mcp.yml

Questions?

- github.com/mesoform/terraform-infrastructure-modules/blob/master/tests/README.md
- github.com/operatingops (Python external data credits)
- linkedin.com/in/garethbrown1
- www.mesoform.com
- twitter.com/MesoformLtd
- hello@mesoform.com