

A screenshot of a computer

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with medium confidence

#!/bin/bash

# NOTE: install jq in the container first `apk add jq`

set -xeuo pipefail

for project\_id in $(cat projects.json | jq -rc '.[]')

do

terraform import -var project\_prefix= -var gcp\_project=dh-monitoring -var tfstate\_bucket=dh-monitoring-stackdriver-prod-tfstate google\_monitoring\_monitored\_project.project[\"$project\_id\"] dh-monitoring/$project\_id

done

project\_number\_to\_id.sh

#!/bin/bash

set -xeuo pipefail

# API only outputs project numbers (internal google ids)

gcloud alpha monitoring metrics-scopes describe dh-monitoring --format=json | \

jq '[.monitoredProjects[] | .name | split("/") | last]' \

> project\_numbers.json

# Use projects API to convert number to id

for project\_number in $(cat project\_numbers.json | jq -rc '.[]')

do

gcloud projects describe $project\_number --format=json | jq -r .projectId >> project\_ids

done

# Convert to json

jq --raw-input --slurp 'split("\n")' project\_ids > ../projects.json

update\_project\_labels.sh

#!/bin/bash

set -xeuo pipefail

all\_projects=( $(gcloud projects list --format="value(projectId)") )

projects\_monitoring\_enabled=( $(cat project\_ids) )

# https://stackoverflow.com/a/28161520

projects\_monitoring\_disabled=( $(echo ${all\_projects[@]} ${projects\_monitoring\_enabled[@]} | tr ' ' '\n' | sort | uniq -u ) )

echo "Projects with monitoring enabled"

echo ${projects\_monitoring\_enabled[@]}

echo "-----"

echo "Projects with monitoring disabled"

echo ${projects\_monitoring\_disabled[@]}

for i in "${projects\_monitoring\_enabled[@]}"

do

gcloud alpha projects update $i --update-labels=monitoring=true

done

for i in "${projects\_monitoring\_disabled[@]}"

do

gcloud alpha projects update $i --remove-labels=monitoring

done

test\_infra\_drift.py

import json

import os

import pytest

from google.cloud import monitoring\_metrics\_scope\_v1, resourcemanager\_v3

@pytest.fixture

def monitored\_projects\_in\_code(file\_name="monitored\_projects.json"):

monitored\_projects = {}

working\_directory\_inside\_container = os.path.dirname(os.path.realpath(\_\_file\_\_))

with open(f'{working\_directory\_inside\_container}/{file\_name}') as file:

data = json.load(file)

monitored\_projects = data

return monitored\_projects

def parse\_project\_number(monitored\_project\_resource\_name):

return str.split(monitored\_project\_resource\_name, '/')[-1]

def convert\_project\_numbers\_to\_ids(metrics\_scope\_response):

client = resourcemanager\_v3.ProjectsClient()

project\_ids = []

for monitored\_project in metrics\_scope\_response:

project\_number = parse\_project\_number(monitored\_project.name)

request = resourcemanager\_v3.GetProjectRequest(

name=f"projects/{project\_number}",

)

response = client.get\_project(request=request)

project\_ids.append(response.project\_id)

return project\_ids

def test\_terraform\_state\_and\_google\_api\_match(monitored\_projects\_in\_code):

client = monitoring\_metrics\_scope\_v1.MetricsScopesClient()

request = monitoring\_metrics\_scope\_v1.GetMetricsScopeRequest(

name="locations/global/metricsScopes/dh-monitoring",

)

response = client.get\_metrics\_scope(request=request)

monitored\_projects\_in\_api = convert\_project\_numbers\_to\_ids(response.monitored\_projects)

print(f"{len(monitored\_projects\_in\_api)} projects monitored in dh-monitoring")

print(f"{len(monitored\_projects\_in\_code)} projects monitored in code")

assert set(monitored\_projects\_in\_api) == set(monitored\_projects\_in\_code)

webhook\_report.py

import os

import requests

def send\_test\_report\_to\_webhook(test\_report):

print(test\_report)

requests.post(url = os.environ.get("WEBHOOK\_URL"), json = {

"title": "dh-google-cloud-monitoring drift detection",

"text": f'### \*[This message was generated from Gitlab CI]({os.environ.get("CI\_JOB\_URL")})\* \n\n # [Follow drift detection steps](https://dhgitlab.domain.co.uk/Cloud/dh-monitoring/dh-google-cloud-monitoring/-/blob/master/projects/README.md#drift-detection) to reconcile monitored projects \n\n {test\_report}'

}, timeout = 10)

Projects main.tf

data "google\_projects" "monitored\_projects" {

filter = "labels.monitoring:true lifecycleState:ACTIVE"

}

resource "google\_monitoring\_monitored\_project" "project" {

for\_each = { for p in data.google\_projects.monitored\_projects.projects : p.project\_id => p.project\_id }

metrics\_scope = "dh-monitoring"

name = each.value

}

Projects outputs.tf

output "monitored\_projects" {

value = [for p in google\_monitoring\_monitored\_project.project : reverse(split("/", p.id))[0]]

}

Projects provider.tf

provider "google" {

project = var.gcp\_project

}

Projects variables.tf

variable "gcp\_project" {

type = string

}

variable "project\_prefix" {

type = string

}

variable "tfstate\_bucket" {

type = string

}

Projects versions.tf

terraform {

required\_version = "~> 1.3.3"

backend "gcs" {}

required\_providers {

google = {

source = "hashicorp/google"

version = "~> 4.41.0"

}

}

}

# Projects README Monitored Projects

We use a centralised 'scoping project' dh-monitoring for pulling in metrics from other projects.

Projects connected to the 'scoping project' were previously defined in the UI [here](https://console.cloud.google.com/monitoring/settings/usage?project=dh-monitoring), but now defined by adding a monitoring:true project label instead.

**Historical context**

Another repo https://dhgitlab.domain.co.uk/Cloud/dh-google-cloud-project-inventory was previousy used for marking which projects should be used in monitoring. Some work was done to make this able to apply certain metadata as project labels. Ultimately this approach creates a dependency to that project, so we decided to keep the source of truth for monitored projects in this repo.

**Getting this into code**

First, the [project number to id script](/Cloud/dh-monitoring/dh-google-cloud-monitoring/-/blob/master/projects/scripts/project_number_to_id.sh) was used to pull the monitored projects that are already linked in dh-monitoring. For readability we convert project numbers to ids (alphabetical project names) rather than project numbers (used internally by Google).

Once the project id json (projects.json) was created, the [import script](/Cloud/dh-monitoring/dh-google-cloud-monitoring/-/blob/master/projects/scripts/project_number_to_id.sh) pulled the existing monitoring projects into Terraform state.

Then the [update project labels script](/Cloud/dh-monitoring/dh-google-cloud-monitoring/-/blob/master/projects/scripts/update_project_labels.sh) was used to apply the monitoring:true feature flag to each project already linked to dh-monitoring. This script was pulled from https://dhgitlab.domain.co.uk/Cloud/dh-google-cloud-project-inventory and modified to use the new feature flag label rather than the old support\_type:production\_runtime label.

See the comments in the scripts for more info.

**Maintaing this in future**

Any changes to monitored projects are done via the monitoring:true project label. Processes such as project commissioning and decommissioning should apply/remove that label and re-deploy this repo.

**Drift detection**

The [tests](/Cloud/dh-monitoring/dh-google-cloud-monitoring/-/blob/master/projects/test/test_infra_drift.py) should indicate whether or not projects are represented correctly in Terraform state.

Follow the steps below to reconcile any changes that may have caused drift between

**Project added via UI/API but not in Terraform state**

* Import the resource into Terraform state using a command similar to the one in [scripts](/Cloud/dh-monitoring/dh-google-cloud-monitoring/-/blob/master/projects/scripts/import.sh)
  + This must be done manually

**Project deleted via UI/API but not in Terraform state**

* Run a -refresh-only plan and apply to reconcile Terraform state with the real infrastructure:
  + Navigate to [run CI pipeline](https://dhgitlab.dunnhumby.co.uk/Cloud/dh-monitoring/dh-google-cloud-monitoring/-/pipelines/new) in Gitlab
  + Add a variable called PLAN\_REFRESH\_ONLY with the value true
  + Once the production-projects plan job finishes, apply it
  + See [MR #143](https://dhgitlab.dunnhumby.co.uk/Cloud/dh-monitoring/dh-google-cloud-monitoring/-/merge_requests/143) for a screenshot of this process

After performing these steps, drift detection should then pass. Re-run the pipeline with no CI variables to validate.

**Project added or deleted in Terraform state but not in UI/API**

This could be if an API error occured when Terraform was trying to deploy a new monitored project.

* Try running the pipeline again
* Add the monitored project via the UI or API and run a -refresh-only plan and apply to reconcile Terraform state

Run\_container.sh

#!/bin/bash

set -xeo pipefail

COMMAND="$1"

[[ -z "$COMMAND" ]] && { echo "ERROR: COMMAND (apply\_all, validate\_all, destroy\_all) must be defined"; exit 1; }

[[ -z "$ENVIRONMENT\_NAME" ]] && { echo "ERROR: ENVIRONMENT\_NAME must be defined"; exit 1; }

IMAGE="$(grep -m 1 image: .gitlab-ci.yml | awk '{print $2}')"

CREDENTIALS\_TYPE=""

CREDENTIALS\_PATH=""

if [[ -f $SERVICE\_ACCOUNT\_KEY\_PATH ]]

then

echo 'Using service account key'

CREDENTIALS\_TYPE="-e GOOGLE\_CREDENTIALS=$(pwd)/.credentials/key.json"

CREDENTIALS\_PATH="-v $SERVICE\_ACCOUNT\_KEY\_PATH:$(pwd)/.credentials/key.json:ro"

elif [[ -f ${HOME}/.config/gcloud/application\_default\_credentials.json ]]

then

echo 'Using ADC credentials'

CREDENTIALS\_TYPE="-e GOOGLE\_APPLICATION\_CREDENTIALS=$(pwd)/.credentials/application\_default\_credentials.json"

CREDENTIALS\_PATH="-v ${HOME}/.config/gcloud/application\_default\_credentials.json:$(pwd)/.credentials/application\_default\_credentials.json:ro"

else

exit "No valid auth method found"

fi

echo $CREDENTIALS\_TYPE

echo $CREDENTIALS\_PATH

docker run -it --rm \

-e TF\_LOG=JSON \

-e TF\_LOG\_PATH=tf.log \

-e "ENVIRONMENT\_NAME=$ENVIRONMENT\_NAME" \

-e "AUTO\_APPROVE"=$AUTO\_APPROVE \

-e PLAN\_REFRESH\_ONLY \

-e HTTP\_PROXY="http://securecomms.int.dh:3128" \

-e HTTPS\_PROXY="http://securecomms.int.dh:3128" \

$CREDENTIALS\_TYPE \

$CREDENTIALS\_PATH \

-v $(pwd):$(pwd) \

-w $(pwd) \

$IMAGE make $COMMAND

Gitlab-ci.yml

variables:

HTTPS\_PROXY: http://securecomms.int.dh:3128

NO\_PROXY:

AUTO\_APPROVE: -auto-approve

ENVIRONMENT\_NAME: $CI\_ENVIRONMENT\_NAME

TF\_IN\_AUTOMATION: "true"

TF\_LOG: JSON

TF\_LOG\_PATH: tf.log

default:

image: docker-dev.artifactory.domain.com/generic-prod/cloud-ops/dh-google-cloud-monitoring:3.5.1

tags:

- gke-shared-kubernetes

before\_script:

- echo $GOOGLE\_APPLICATION\_CREDENTIALS\_JSON > ~/google-credentials.json

- export GOOGLE\_APPLICATION\_CREDENTIALS=~/google-credentials.json

stages:

- test

- plan

- apply

.base:

parallel:

matrix:

- STACK: [projects, notification-channel, groups, alert-policies, dashboards]

environment: production

resource\_group: $STACK

setup-test:

stage: test

environment: production

script:

- make projects.init

- terraform -chdir=projects output -json monitored\_projects > monitored\_projects.json

rules:

- if: $PLAN\_REFRESH\_ONLY

when: never

- if: $CI\_COMMIT\_BRANCH == "master"

artifacts:

paths:

- monitored\_projects.json

test:

stage: test

needs: ["setup-test"]

image: docker-dev.artifactory.domain.com/generic-prod/cloud-ops/dh-google-cloud-monitoring-test:0.0.2

environment: production

script:

- mv monitored\_projects.json projects/test

- pytest --tb=no

rules:

- if: $PLAN\_REFRESH\_ONLY

when: never

- if: $CI\_COMMIT\_BRANCH == "master"

lint:

stage: test

environment: production

script:

- terraform fmt -recursive -check -diff || FAILED=true

- if [ $FAILED ]; then echo -e "\e[31mRun 'terraform fmt -recursive' to lint code" && exit 3; fi

allow\_failure: true

validate:

stage: test

parallel:

matrix:

- STACK: [notification-channel, projects, groups, alert-policies, dashboards]

environment: production

only:

- tags

- branches

script:

- make $STACK.validate

plan:

extends: .base

stage: plan

only:

- tags

- branches

script:

- make $STACK.plan

- terraform -chdir=$STACK show --json ${STACK}.tfplan | jq -r '([.resource\_changes[]?.change.actions?]|flatten)|{"create":(map(select(.=="create"))|length),"update":(map(select(.=="update"))|length),"delete":(map(select(.=="delete"))|length)}' > ${STACK}.plan.json

artifacts:

paths:

- tf.log

- $STACK/${STACK}.tfplan

reports:

terraform: ${STACK}.plan.json

apply:

extends: .base

stage: apply

needs: ["plan"]

allow\_failure: false

rules:

- if: $CI\_COMMIT\_BRANCH == "master"

when: manual

script:

- make $STACK.apply

artifacts:

paths:

- tf.log