

Preparing cloud staging/production Kubernetes cluster for a Microservices

Table of Contents

Agenda	1
Requirements	2
Why those requirements	3
Microservices Overview	3
Architecture	3
GCP Core Cloud setup	4
management	4
GCP Project setup	11
global	11
regional	32
TODO (fix old practice)	32
Prometheus	32
Kubernetes	37
SOPS with GCP KMS Example	42
What Next	45
Learning paths	45
Next workshops in the microservices series	45
Related Open Source project:	48
Cleanup	48
QnA	48

Solution which will be proposed during these workshops was created to simplify deployment process and infrastructure creation by automating it using DevOps methodology and tools.

Agenda

- Start 10:00
- Section 1:
 - Architecture
- Section 2:

- TODO
- Coffee break XX:XX - XX:XX
- Section 3:
 - TODO
- Lunch XX:XX - XX:XX
- Section 4:
 - TODO
- Coffee break XX:XX - XX:XX
- Section 5:
 - TODO
- Section 6:
 - What's Next
 - Cleanup
 - Q&A
- Ends between XX:XX and XX:XX - depends on Q&A session

Requirements

For this Workshop you will be working as Infrastructure engineer - and you need this software to be installed:

- Tools
 - kubectl (kubernetes cli)
 - kubectx (kubernetes cli extension)
 - helm (kubernetes package management)
 - k9s (kubernetes management tool for cli)
 - terraform (infrastructure as code tool)
 - gcloud (Google Cloud CLI)

You can find how to install this software here: <https://mobica-workshops.gitlab.io/documentation/requirements-guide/>

Windows users should install [VirtualBox](#) there will be a link to the Google Drive file given during the workshops, or use [WSL](#).



Please remember that Infrastructure Engineer can be just one of your roles in the projects.

In many projects one engineer can have many roles and sometimes one person is a Developer, QA, pipeline automator (devops automator), and an Infrastructure

Why those requirements

Like mentioned in this workshop you need all of this software installed to work as an Infrastructure engineer with our examples.

kubectl - Kubernetes CLI will be used to check if our cluster works

kubectx - CLI tool extending kubectl is used by our local deployment script to be sure that proper cluster and namespace is used.

helm - Package manager for Kubernetes used to deploy applications to our kubernetes cluster.

k9s - terminal based UI to interact with Kubernetes clusters

terraform - infrastructure as code tool we will use to deploy to the cluster and also which is used by our GCP infrastructure.

Microservices Overview

We will use our Microservices Examples as a core Infrastructure for our Deployments. If you were on our Backend and Frontend workshops this will be for you a very short recap. We will focus this time only on the Architecture as this training should be done after [development environment guide](#).

Architecture

Our target backend architecture will be looking like this:

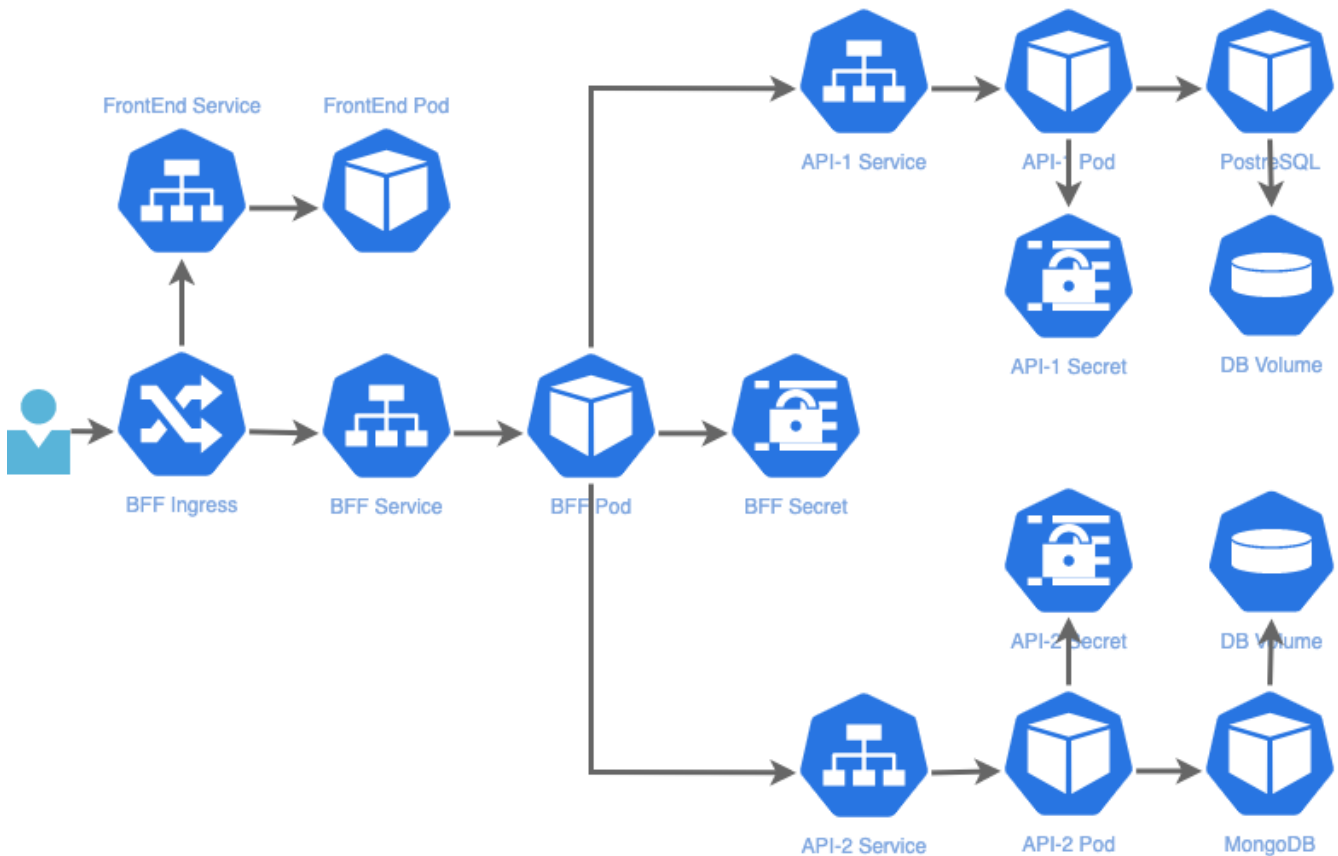


Figure 3: Target System Architecture

- Our FrontEnd Service will be simulated with [Book Frontend](#).

We have three backend microservices prepared which we will deploy to make our Frontend application working:

- Our BFF Service will be simulated with [Book API Gateway Service](#).
- Our API-1 Service will be simulated with [Book List API Service](#).
- Our API-2 Service will be simulated with [Book Admin API service](#)

In the upper target system which will be available on the staging and production environments our frontend is talking through the Ingress with our BFF Service which is operating in the fully working environment.

Locally we will be using K3s cluster, and we will resign from Ingress to not overcomplicate our helm chart with different Ingress setup. We will just use NodePort configuration to simply it.

GCP Core Cloud setup

In our case our core Cloud is GCP and domain used by the workshops is in this cloud

management

In case of the Cloud we will use cloud specific state. In case of the GCP state can be stored in the

GCS which already have locking mechanism, and we do not need any special service for this. When working with remote state like this is very important to choose a valid prefix convention. In our case prefix is exactly the same as our folder structure. We are storing terraform state in it but also using it with `data` when we need to access state of other module.

After creating cloud based terraform state we will focus on our domain and project management.



We will focus on explaining how this can be done, and we will assume you will prepare your GCP cloud in a similar way. The expectation is that basics of terraform also do not need to be explained and you learned more about terraform after our development infrastructure workshop.

Our Infrastructure repository is available here: <https://gitlab.com/devops-training-info/examples/terraform/infrastructure> and our modules repository is available here: <https://gitlab.com/devops-training-info/examples/terraform/modules>.

Expectation is that you will create own repository for and infrastructure repository with the same name, clone our modules repository and put them in the project folder. Later on I'll be just mentioning files like `infrastructure/gcp/management/global/tfstate/main.tf` expecting you working on project folder or just `gcp/management/global/tfstate/main.tf` when from the start mentioned is that you are working in the infrastructure repository.

First we need to log in to the Google Cloud. To do this in a way terraform can use it to access cloud please use instruction available [here](https://registry.terraform.io/providers/hashicorp/google/latest/docs/guides/getting_started)

Second if we are re-creating project from the scratch we need to prepare terraform state bucket. To do this modify `gcp/management/global/tfstate` in a way described [here](<https://cloud.google.com/docs/terraform/resource-management/store-state>). Short instruction is to first create bucket with a local state and then migrate it to created bucket. This step can be also achieved by using steps which are explained in the next sections.

Terraform State

Our first task is to create terraform state we will be using for our Cloud.

Please create `gcp/management/global/tfstate/versions.tf` file in the `infrastructure` repository looking like this:

```
terraform {
  required_version = "~> 1"
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 4"
    }
  }
}
```

In this file we will configure terraform and name of the file can be different to fit what you like.

Next, please create `gcp/management/global/tfstate/providers.tf` file in the `infrastructure` repository looking like this:

```
provider "google" {  
  region = "europe-central2"  
  project = "devops-training-workshop-admin"  
}
```

In this file we are configuring GCP provider. In most of the cases this file will be identical everywhere. What is important here is a region and project name. In case of the management we should use resources which are not region based and we are assuming that region put here we are treating as also a management region. In our case we have separate admin type of project which we created by hand.

Next, please create `gcp/management/global/tfstate/main.tf` file in the `infrastructure` repository looking like this:

```
data "google_project" "project" {  
}  
  
resource "random_id" "bucket_code" {  
  byte_length = 8  
}  
  
resource "google_storage_bucket" "tfstate" {  
  name = "${data.google_project.project.project_id}-${  
    ${random_id.bucket_code.hex}-tfstate"  
  }  
  force_destroy = false  
  location = var.location  
  uniform_bucket_level_access = true  
  storage_class = "STANDARD"  
  versioning {  
    enabled = true  
  }  
}
```

In this file we are creating bucket which will be used for our terraform state.

Next, please create `gcp/management/global/tfstate/outputs.tf` file in the `infrastructure` repository looking like this:

```
output "tfstate" {  
  value = google_storage_bucket.tfstate.name  
}
```

This will show the name of our bucket when created and store it in the outputs.

When bucket will be created please modify `gcp/management/global/tfstate/versions.tf` file in the `infrastructure` repository to look similar to this:

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate" ①
    prefix = "gcp/management/global/dns" ②
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 4"
    }
  }
}
```

- ① We will be using our bucket in all examples, but you will need to put here your bucket in this place when you will be copying code from our examples.
- ② We will let know where to store state for each section by changing prefix and in many cases this file will be different only here

With this we will migrate state from the local file to the cloud.

DNS

Please create `gcp/management/global/dns/versions.tf` file in the `infrastructure` repository looking like this:

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/management/global/dns" ①
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 4"
    }
  }
}
```

- ① Like mentioned before prefix should be different for each section

Like mentioned upper this file is in most of the cases almost identical in every section with a change in the prefix. Structure here should be the same like a folder structure as this simplifies all.

Next, please create `gcp/management/global/dns/providers.tf` file in the `infrastructure` repository looking like this:

```
provider "google" {
  region = "europe-central2"
  project = "devops-training-workshop-admin"
}
```

Next, please create `gcp/management/global/dns/main.tf` file in the `infrastructure` repository looking like this:

```
data "google_project" "project" {
}

resource "google_dns_managed_zone" "cloud_devops_training_info" {
  name          = "cloud-devops-training-info"
  dns_name      = "cloud.devops-training.info."
  description   = "CoE DevOps/Cloud main zone"
  project       = data.google_project.project.project_id
}

resource "google_dns_managed_zone" "gcp_cloud_devops_training_info" {
  name          = "gcp-cloud-devops-training-info"
  dns_name      = "gcp.cloud.devops-training.info."
  description   = "CoE DevOps/Cloud GCP zone"
  project       = data.google_project.project.project_id
}

resource "google_dns_record_set" "cloud_devops_training_info_gcp_NS" {
  name          = google_dns_managed_zone.gcp_cloud_devops_training_info.dns_name
  managed_zone  = google_dns_managed_zone.cloud_devops_training_info.name
  type          = "NS"
  ttl           = 3600

  rrdatas = google_dns_managed_zone.gcp_cloud_devops_training_info.name_servers
  project = data.google_project.project.project_id
}
```

In my case I have `devops-training.info`, you will need to modify this file to fit your domain

Next, please create `gcp/management/global/dns/outputs.tf` file in the `infrastructure` repository looking like this:

```
output "cloud_name_servers" {
  value = google_dns_managed_zone.cloud_devops_training_info.name_servers
}

output "gcp_cloud_name_servers" {
```

```

    value = google_dns_managed_zone.gcp_cloud_devops_training_info.name_servers
  }

  output "gcp_cloud_name" {
    value = google_dns_managed_zone.gcp_cloud_devops_training_info.name
  }

  output "gcp_cloud_dns_name" {
    value = google_dns_managed_zone.gcp_cloud_devops_training_info.dns_name
  }

```

I used `cloud_name_servers` to configure my domain `devops-training.info` and rest of the output will be used later.

In my case I planned to use my main domain for a blog and planning to create cloud type subdomains for GCP projects and other Clouds like AWS and Azure in the future.

Projects

Please create `gcp/management/global/projects/versions.tf` file in the `infrastructure` repository looking like this:

```

terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/management/global/projects"
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 4"
    }
  }
}

```

Please create `gcp/management/global/projects/providers.tf` file in the `infrastructure` repository looking like this:

```

provider "google" {
  region = "europe-central2"
  project = "devops-training-workshop-admin"
}

```

Please create `gcp/management/global/projects/variables.tf` file in the `infrastructure` repository looking like this:

```
variable "billing_account" {
  default = "01CDE3-215D6D-EB1CC5"
}

variable "folder_id" {
  default = "457640943932"
}
```

You will need to change those variables to fit your projects folder and billing account

Please create `gcp/management/global/projects/main.tf` file in the `infrastructure` repository looking like this:

```
data "google_project" "project" {}

resource "google_project" "devops_training_workshop_sbx" {
  name          = "devops-training-workshop-sbx"
  project_id     = "devops-training-workshop-sbx"
  billing_account = var.billing_account
  folder_id      = var.folder_id
}
```

Here I'm creating separate project which will be used by our microservice based examples and other workshops

Please create `gcp/management/global/projects/outputs.tf` file in the `infrastructure` repository looking like this:

```
output "management_project_id" {
  value = data.google_project.project.project_id
}

output "devops_training_workshop_sbx_project_id" {
  value = google_project.devops_training_workshop_sbx.project_id
}

output "devops_training_workshop_sbx_project_number" {
  value = google_project.devops_training_workshop_sbx.number
}

# This is something more like a variable for the other projects what needs to be
unique only internally
output "devops_training_workshop_sbx_project_short_name" {
  value = "sandbox"
}
```

Here we have outputs which will be used later related to our project which will be used for

examples.

GCP Project setup

Each project have a global resources and a regional one.



In case of examples used in this chapter our project was named **devops-training-workshop-sbx** and all code will be pitted in the infrastructure repository **gcp/devops-training-workshop-sbx** folder. I personally like this naming convention but you can choose your own.

global

In this chapter we will be focusing on the global resources. Global resources in our case sometimes are also regional but we are treating region mentioned here as a main region used by us.

Our plan here is to:

- create **artifact-registry** to store our docker images and helm charts
- create **dns** managed zone which will be used by our project
- create **secrets** resources which will be used for managing our secrets which sops
- create **secrets-manager** which will store secrets for our applications
- create **services** resources which are global
- create **workload-identity** federation for Gitlab CI

artifact-registry

Please create **gcp/devops-training-workshop-sbx/global/artifact-registry/versions.tf** file in the **infrastructure** repository looking like this:

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/artifact-registry"
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 7"
    }
  }
}
```

Next, please create **gcp/devops-training-workshop-sbx/global/artifact-registry/providers.tf** file in

the **infrastructure** repository looking like this:

```
provider "google" {
  region = "europe-central2"
  project = "devops-training-workshop-sbx"
}
```

Next, please create **gcp/devops-training-workshop-sbx/global/artifact-registry/main.tf** file in the **infrastructure** repository looking like this:

```
resource "google_artifact_registry_repository" "docker" {
  repository_id = "docker"
  description   = "our docker repository"
  format       = "DOCKER"
}

resource "google_artifact_registry_repository" "charts" {
  repository_id = "charts"
  description   = "Helm charts repository"
  format       = "DOCKER"
}
```

Here we are creating two registry repositories. First one is for a docker images and second one is for the helm charts

Next, please create **gcp/devops-training-workshop-sbx/global/artifact-registry/outputs.tf** file in the **infrastructure** repository looking like this:

```
output "docker_location" {
  value       = google_artifact_registry_repository.docker.location
  description = "Artifact registry docker - location"
}

output "docker_name" {
  value       = google_artifact_registry_repository.docker.name
  description = "Artifact registry docker - name"
}

output "charts_location" {
  value       = google_artifact_registry_repository.charts.location
  description = "Artifact registry charts - location"
}

output "charts_name" {
  value       = google_artifact_registry_repository.charts.name
  description = "Artifact registry charts - name"
}
```

We will be using those outputs later

dns

Please create `gcp/devops-training-workshop-sbx/global/dns/versions.tf` file in the `infrastructure` repository looking like this:

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/dns"
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 4"
    }
  }
}
```

Next, please create `gcp/devops-training-workshop-sbx/global/dns/providers.tf` file in the `infrastructure` repository looking like this:

```
provider "google" {
  region = "europe-central2"
  project = "devops-training-workshop-sbx"
}
```

Next, please create `gcp/devops-training-workshop-sbx/global/dns/data.tf` file in the `infrastructure` repository looking like this:

```
data "terraform_remote_state" "management_global_dns" {
  backend = "gcs"

  config = {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/management/global/dns"
  }
}

data "terraform_remote_state" "management_global_projects" {
  backend = "gcs"

  config = {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/management/global/projects"
  }
}
```

```
}
```

here we are accessing outputs from the other states which we will use as an input values.

Next, please create `gcp/devops-training-workshop-sbx/global/dns/main.tf` file in the `infrastructure` repository looking like this:

```
module "dns" {  
  # source = "../../../../../modules/gcp/environments/global/dns" ①  
  source = "git@gitlab.com:devops-training-  
info/examples/terraform/modules.git//gcp/environments/global/dns?ref=1.0.1" ②  
  
  management_project_id =  
data.terraform_remote_state.management_global_projects.outputs.management_project_id  
  project_short_name =  
data.terraform_remote_state.management_global_projects.outputs.devops_training_worksho  
p_sbx_project_short_name  
  gcp_cloud_dns_name =  
data.terraform_remote_state.management_global_dns.outputs.gcp_cloud_dns_name  
  gcp_cloud_name =  
data.terraform_remote_state.management_global_dns.outputs.gcp_cloud_name  
}
```

① we can use source targeting the module which is just the folder

② or we can use source targeting the module which is tagged version of our module and this is a preferred way

In upper example we are using module feature of the Terraform.

Next, please create `gcp/devops-training-workshop-sbx/global/dns/outputs.tf` file in the `infrastructure` repository looking like this:

```
output "project_cloud_name" {  
  value = module.dns.project_cloud_name  
}  
  
output "project_cloud_dns_name" {  
  value = module.dns.project_cloud_dns_name  
}
```

secrets

Please create `gcp/devops-training-workshop-sbx/global/secrets/versions.tf` file in the `infrastructure` repository looking like this:

```
terraform {  
  required_version = "~> 1"
```

```

backend "gcs" {
  bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
  prefix = "gcp/devops-training-workshop-sbx/global/secrets"
}
required_providers {
  google = {
    source = "hashicorp/google"
    version = "~> 4"
  }
}
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/secrets/providers.tf` file in the `infrastructure` repository looking like this:

```

provider "google" {
  region = "europe-central2"
  project = "devops-training-workshop-sbx"
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/secrets/main.tf` file in the `infrastructure` repository looking like this:

```

module "secrets" {
  # source = "../../../../../modules/gcp/environments/global/secrets"
  source = "git@gitlab.com:devops-training-
info/examples/terraform/modules.git//gcp/environments/global/secrets?ref=1.0.1"
}

```

with this we will create KMS crypto key and KMS key ring for our [SOPS](#).



You will need to modify your future actions using my KMS key to use your key instead.

secret-manager

Please create `gcp/devops-training-workshop-sbx/global/secret-manager/my-secrets.yml` file in the `infrastructure` repository looking like this:

```

---
devops-workshops:
  staging:
    postgresql:
      rootUser: root
      rootPassword: REDACTED
      bookListDatabase: book-list
      bookListUser: book-list

```

```

    bookListPassword: REDACTED
  mongodb:
    bookAdminDatabase: book-admin
    bookAdminUser: book-admin
    bookAdminPassword: REDACTED
  production:
    postgresql:
      rootUser: root
      rootPassword: REDACTED
      bookListDatabase: book-list
      bookListUser: book-list
      bookListPassword: REDACTED
    mongodb:
      bookAdminDatabase: book-admin
      bookAdminUser: book-admin
      bookAdminPassword: REDACTED

```

Encrypt it with SOPS and your key

```

sops --encrypt --gcp-kms projects/devops-training-workshop-
sbx/locations/europe/keyRings/sops/cryptoKeys/sops-key my-secrets.yml > my-
secrets.enc.yml

```



Please use YOUR key not My

Next, remove `gcp/devops-training-workshop-sbx/global/secret-manager/my-secrets.yml` as this file should NEVER be added to your repository.

Next, please create `gcp/devops-training-workshop-sbx/global/secret-manager/versions.tf` file in the `infrastructure` repository looking like this:

```

terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/secret-manager"
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 7"
    }
    sops = { ①
      source = "carlpett/sops"
      version = "~> 1"
    }
  }
}

```

```
}
```

① we are using here sops provider

Next, please create `gcp/devops-training-workshop-sbx/global/secret-manager/providers.tf` file in the `infrastructure` repository looking like this:

```
provider "google" {  
  region = "europe-central2"  
  project = "devops-training-workshop-sbx"  
}
```

Next, please create `gcp/devops-training-workshop-sbx/global/secret-manager/data.tf` file in the `infrastructure` repository looking like this:

```
data "google_project" "project" {}  
  
data "google_client_config" "this" {  
  provider = google  
}  
  
data "sops_file" "demo-secret" { ①  
  source_file = "my-secrets.enc.yml"  
  input_type  = "yaml"  
}
```

① SOPS provider allowing us to use this type of data source

Next, please create `gcp/devops-training-workshop-sbx/global/secret-manager/main.tf` file in the `infrastructure` repository looking like this:

```
module "staging" {  
  # source = "../../../../../modules/gcp/environments/global/secret-manager"  
  source = "git@gitlab.com:devops-training-  
info/examples/terraform/modules.git//gcp/environments/global/secret-manager?ref=1.1.0"  
  
  name = "staging"  
  root_user = data.sops_file.demo-secret.data["devops-  
workshops.staging.postgresql.rootUser"]  
  root_password = data.sops_file.demo-secret.data["devops-  
workshops.staging.postgresql.rootPassword"]  
  book_list_database = data.sops_file.demo-secret.data["devops-  
workshops.staging.postgresql.bookListDatabase"]  
  book_list_user = data.sops_file.demo-secret.data["devops-  
workshops.staging.postgresql.bookListUser"]  
  book_list_password = data.sops_file.demo-secret.data["devops-  
workshops.staging.postgresql.bookListPassword"]  
  book_admin_database = data.sops_file.demo-secret.data["devops-
```

```
workshops.staging.mongodb.bookAdminDatabase"]
  book_admin_user      = data.sops_file.demo-secret.data["devops-
workshops.staging.mongodb.bookAdminUser"]
  book_admin_password = data.sops_file.demo-secret.data["devops-
workshops.staging.mongodb.bookAdminPassword"]
}
```

This will store our secrets which we have in the sops file in the GCP secret manager

services

Please create `gcp/devops-training-workshop-sbx/global/services/versions.tf` file in the `infrastructure` repository looking like this:

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/services"
  }
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 7"
    }
    random = { ①
      source  = "hashicorp/random"
      version = "~> 3"
    }
  }
}
```

① Here we are using also the random provider

Next, please create `gcp/devops-training-workshop-sbx/global/services/providers.tf` file in the `infrastructure` repository looking like this:

```
provider "google" {
  region = "europe-central2"
  project = "devops-training-workshop-sbx"
}
```

Next, please create `gcp/devops-training-workshop-sbx/global/services/data.tf` file in the `infrastructure` repository looking like this:

```
data "google_project" "project" {}

data "google_client_config" "this" {
```

```

    provider = google
  }

  data "terraform_remote_state" "workload-identity" {
    backend = "gcs"

    config = {
      bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
      prefix = "gcp/devops-training-workshop-sbx/global/workload-identity"
    }
  }

  data "google_storage_project_service_account" "gcs_account" {}

```

Next, please create `gcp/devops-training-workshop-sbx/global/services/main.tf` file in the `infrastructure` repository looking like this:

```

module "staging" {
  source = "../../../../../modules/gcp/environments/global/services"
  # source = "git@gitlab.com:devops-training-
  info/examples/terraform/modules.git//gcp/environments/global/services?ref=1.0.1"

  environment          = "staging"
  location              = data.google_client_config.this.region
  project_id           = data.google_project.project.project_id
  deployer_service_account_email = data.terraform_remote_state.workload-
  identity.outputs.deployer_service_account_email
  gcs_account_email_address =
  data.google_storage_project_service_account.gcs_account.email_address
}

module "production" {
  source = "../../../../../modules/gcp/environments/global/services"
  # source = "git@gitlab.com:devops-training-
  info/examples/terraform/modules.git//gcp/environments/global/services?ref=1.0.1"

  environment          = "production"
  location              = data.google_client_config.this.region
  project_id           = data.google_project.project.project_id
  deployer_service_account_email = data.terraform_remote_state.workload-
  identity.outputs.deployer_service_account_email
  gcs_account_email_address =
  data.google_storage_project_service_account.gcs_account.email_address
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/services/outputs.tf` file in the `infrastructure` repository looking like this:

```

output "devops_workshops_staging_address" {
  value = module.staging.devops_workshops_address
}

output "staging_state_bucket_id" {
  value = module.staging.state_bucket_id
}

output "staging_services_kms_key" {
  value = module.staging.services_kms_key
}

output "devops_workshops_production_address" {
  value = module.production.devops_workshops_address
}

output "production_state_bucket_id" {
  value = module.production.state_bucket_id
}

output "production_services_kms_key" {
  value = module.production.services_kms_key
}

```

We are creating in this section all global resources we are planning to use in our staging and production environments

books

Please create `gcp/devops-training-workshop-sbx/global/services/versions.tf` file in the `infrastructure` repository looking like this:

```

terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/services/books"
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 7"
    }
  }
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/services/providers.tf` file in the `infrastructure` repository looking like this:

```

provider "google" {
  region = "europe-central2"
  project = "devops-training-workshop-sbx"
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/services/data.tf` file in the `infrastructure` repository looking like this:

```

data "terraform_remote_state" "sandbox_global_dns" {
  backend = "gcs"

  config = {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/dns"
  }
}

data "terraform_remote_state" "sandbox_global_services" {
  backend = "gcs"

  config = {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/services"
  }
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/services/main.tf` file in the `infrastructure` repository looking like this:

```

// https://cloud.google.com/certificate-manager/docs/deploy-google-managed-dns-auth#terraform_1

locals {
  domain =
    trimsuffix(data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_dns_name, ".")
}

resource "google_certificate_manager_dns_authorization" "dns_authorization" {
  name          = "staging-api-dns-auth"
  description    = "staging API dns authorization"
  domain        = local.domain
}

resource "google_certificate_manager_certificate" "root_cert" {
  name          = "sandbox-rootcert"
  description    = "The wildcard cert"
  managed {

```

```

domains = [local.domain, "*.${local.domain}"]
dns_authorizations = [
  google_certificate_manager_dns_authorization.dns_authorization.id
]
}
labels = {
  "terraform" : true
}
}

resource "google_certificate_manager_certificate_map" "certificate_map" {
  name          = "sandbox-certmap"
  description    = "${local.domain} certificate map"
  labels = {
    "terraform" : true
  }
}

resource "google_certificate_manager_certificate_map_entry" "first_entry" {
  name          = "sandbox-first-entry"
  description    = "certificate map entry for ${local.domain}"
  map           = google_certificate_manager_certificate_map.certificate_map.name
  labels = {
    "terraform" : true
  }
  certificates = [google_certificate_manager_certificate.root_cert.id]
  hostname     = local.domain
}

resource "google_certificate_manager_certificate_map_entry" "second_entry" {
  name          = "sandbox-second-entity"
  description    = "certificate map entry for *.${local.domain}"
  map           = google_certificate_manager_certificate_map.certificate_map.name
  labels = {
    "terraform" : true
  }
  certificates = [google_certificate_manager_certificate.root_cert.id]
  hostname     = "*.${local.domain}"
}

resource "google_dns_record_set" "cname" {
  name          =
  google_certificate_manager_dns_authorization.dns_authorization.dns_resource_record[0].
  name
  managed_zone =
  data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_name
  type          =
  google_certificate_manager_dns_authorization.dns_authorization.dns_resource_record[0].
  type
  ttl           = 300
  rrdatas       =

```

```

[google_certificate_manager_dns_authorization.dns_authorization.dns_resource_record[0]
.data]
}

resource "google_dns_record_set" "staging_api" {
  name          = "staging-books-
api.${data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_dns_name}"
  managed_zone =
data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_name
  type          = "A"
  ttl           = 300

  rrdatas =
[data.terraform_remote_state.sandbox_global_services.outputs.devops_workshops_staging_
address]
}

resource "google_dns_record_set" "staging_frontend" {
  name          = "staging-
books.${data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_dns_name}"
  managed_zone =
data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_name
  type          = "A"
  ttl           = 300

  rrdatas =
[data.terraform_remote_state.sandbox_global_services.outputs.devops_workshops_staging_
address]
}

resource "google_dns_record_set" "api" {
  name          = "books-
api.${data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_dns_name}"
  managed_zone =
data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_name
  type          = "A"
  ttl           = 300

  rrdatas =
[data.terraform_remote_state.sandbox_global_services.outputs.devops_workshops_producti
on_address]
}

resource "google_dns_record_set" "frontend" {
  name          =
"books.${data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_dns_name}"
  managed_zone =
data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_name
  type          = "A"

```

```

    ttl                = 300

    rrdatas =
[data.terraform_remote_state.sandbox_global_services.outputs.devops_workshops_producti
on_address]
}

# Keycloak

resource "google_dns_record_set" "staging_keycloak" {
  name          = "staging-
keycloak.${data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_dns_na
me}"
  managed_zone =
data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_name
  type          = "A"
  ttl           = 300

  rrdatas =
[data.terraform_remote_state.sandbox_global_services.outputs.devops_workshops_staging_
address]
}

resource "google_dns_record_set" "keycloak" {
  name          =
"keycloak.${data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_dns_n
ame}"
  managed_zone =
data.terraform_remote_state.sandbox_global_dns.outputs.project_cloud_name
  type          = "A"
  ttl           = 300

  rrdatas =
[data.terraform_remote_state.sandbox_global_services.outputs.devops_workshops_producti
on_address]
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/services/books/outputs.tf` file in the `infrastructure` repository looking like this:

```

output "certificate_id" {
  value = google_certificate_manager_certificate.root_cert.id
}

output "certificate_map_id" {
  value = google_certificate_manager_certificate_map.certificate_map.id
}

```

We are creating in this section all global resources we are planning to use in our staging and

production environments strictly for a books microservices

workload-identity

Please create `gcp/devops-training-workshop-sbx/global/workload-identity/versions.tf` file in the `infrastructure` repository looking like this:

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/workload-identity"
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 7"
    }
  }
}
```

Next, please create `gcp/devops-training-workshop-sbx/global/workload-identity/providers.tf` file in the `infrastructure` repository looking like this:

```
provider "google" {
  region = "europe-central2"
  project = "devops-training-workshop-sbx"
}

provider "google-beta" {
  region = "europe-central2"
  project = "devops-training-workshop-sbx"
}
```

Next, please create `gcp/devops-training-workshop-sbx/global/workload-identity/data.tf` file in the `infrastructure` repository looking like this:

```
data "google_project" "project" {
}

data "terraform_remote_state" "tfstate" {
  backend = "gcs"

  config = {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/management/global/tfstate"
  }
}
```

```

data "terraform_remote_state" "artifact_registry" {
  backend = "gcs"

  config = {
    bucket = "devops-training-workshop-admin-d39bcb104c3dc59c-tfstate"
    prefix = "gcp/devops-training-workshop-sbx/global/artifact-registry"
  }
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/workload-identity/main.tf` file in the `infrastructure` repository looking like this:

```

// https://cloud.google.com/blog/products/devops-sre/software-delivery-pipelines-with-
gitlab-cicd-and-cloud-deploy
// https://gitlab.com/google-gitlab-components/cloud-deploy#authorization

// Service Accounts
resource "google_service_account" "deployer" {
  account_id   = "gitlab-ci-deployer"
  display_name = "Gitlab CI Deployer Service"
}

resource "google_iam_workload_identity_pool" "gitlab_ci" {
  workload_identity_pool_id = "gitlab-ci"
  display_name               = "GitLab group ID 114676808"
}

// Workload Identity

resource "google_iam_workload_identity_pool_provider" "gitlab_ci" {
  workload_identity_pool_id =
google_iam_workload_identity_pool.gitlab_ci.workload_identity_pool_id
  workload_identity_pool_provider_id = "gitlab-ci"
  display_name                       = "GitLab group ID 114676808"
  attribute_mapping = {
    "attribute.guest_access"      = "assertion.guest_access"
    "attribute.planner_access"    = "assertion.planner_access"
    "attribute.reporter_access"   = "assertion.reporter_access"
    "attribute.developer_access"  = "assertion.developer_access"
    "attribute.maintainer_access" = "assertion.maintainer_access"
    "attribute.owner_access"      = "assertion.owner_access"
    "attribute.namespace_id"      = "assertion.namespace_id"
    "attribute.namespace_path"    = "assertion.namespace_path"
    "attribute.project_id"        = "assertion.project_id"
    "attribute.project_path"      = "assertion.project_path"
    "attribute.user_id"           = "assertion.user_id"
    "attribute.user_login"        = "assertion.user_login"
    "attribute.user_email"        = "assertion.user_email"
    "attribute.user_access_level" = "assertion.user_access_level"
  }
}

```

```

    "google.subject"                = "assertion.sub"
    # "attribute.my_project_maintainer" = "assertion.maintainer_access=="true" &&
assertion.project_path=="gitlab-org/devops-training-info\"
  }
  oidc {
    issuer_uri = "https://auth.gcp.gitlab.com/oidc/devops-training-info"
  }
}

// Bindings

resource "google_artifact_registry_repository_iam_binding" "docker_reader" {
  project      = data.google_project.project.project_id
  location     = data.terraform_remote_state.artifact_registry.outputs.docker_location
  repository   = data.terraform_remote_state.artifact_registry.outputs.docker_name
  role         = "roles/artifactregistry.reader"
  members = [

    "principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.guest_access/true"
  ]
}

resource "google_artifact_registry_repository_iam_binding" "docker_writer" {
  project      = data.google_project.project.project_id
  location     = data.terraform_remote_state.artifact_registry.outputs.docker_location
  repository   = data.terraform_remote_state.artifact_registry.outputs.docker_name
  role         = "roles/artifactregistry.writer"
  members = [

    "principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.developer_access/true"
    #
    "principalSet://iam.googleapis.com/projects/${google_iam_workload_identity_pool.gitlab
_ci.name}/attribute.my_project_maintainer/true"
  ]
}

resource "google_artifact_registry_repository_iam_binding" "charts_reader" {
  project      = data.google_project.project.project_id
  location     = data.terraform_remote_state.artifact_registry.outputs.charts_location
  repository   = data.terraform_remote_state.artifact_registry.outputs.charts_name
  role         = "roles/artifactregistry.reader"
  members = [

    "principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.guest_access/true",
  ]
}

resource "google_artifact_registry_repository_iam_binding" "charts_writer" {

```

```

project      = data.google_project.project.project_id
location     = data.terraform_remote_state.artifact_registry.outputs.charts_location
repository   = data.terraform_remote_state.artifact_registry.outputs.charts_name
role         = "roles/artifactregistry.writer"
members = [

"principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.developer_access/true"
    #
"principalSet://iam.googleapis.com/projects/${google_iam_workload_identity_pool.gitlab
_ci.name}/attribute.my_project_maintainer/true"
]
}

resource "google_project_iam_binding" "clouddeploy_releaser" {
  project = data.google_project.project.project_id
  role    = "roles/clouddeploy.releaser"

  members = [

"principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.developer_access/true",
  ]
}

resource "google_project_iam_binding" "storage_admin" {
  project = data.google_project.project.project_id
  role    = "roles/storage.admin"

  members = [

"principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.developer_access/true",
  ]
}

resource "google_project_iam_binding" "iam_service_account_user" {
  project = data.google_project.project.project_id
  role    = "roles/iam.serviceAccountUser"

  members = [

"principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.developer_access/true",
    # "serviceAccount:${google_service_account.cloud_deploy_cloud_run.email}",
    # "serviceAccount:${google_service_account.cloud_deploy_gke.email}",
  ]
}

resource "google_project_iam_binding" "clouddeploy_jobRunner" {

```

```

project = data.google_project.project.project_id
role    = "roles/clouddeploy.jobRunner"

members = [
  "serviceAccount:${google_service_account.deployer.email}",
]
}

resource "google_project_iam_binding" "logging_logWriter" {
  project = data.google_project.project.project_id
  role    = "roles/logging.logWriter"

  members = [
    "serviceAccount:${google_service_account.deployer.email}",
  ]
}

resource "google_project_iam_binding" "storage_object_viewer" {
  project = data.google_project.project.project_id
  role    = "roles/storage.objectViewer"

  members = [
    "serviceAccount:${google_service_account.deployer.email}",
  ]
}

resource "google_project_iam_binding" "storage_object_creator" {
  project = data.google_project.project.project_id
  role    = "roles/storage.objectCreator"

  members = [
    "serviceAccount:${google_service_account.deployer.email}",
  ]
}

# resource "google_project_iam_binding" "run_developer" {
#   project = data.google_project.project.project_id
#   role    = "roles/run.developer"
#
#   members = [
#     "principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}/attribute.developer_access/true",
#     "serviceAccount:${google_service_account.cloud_deploy_cloud_run.email}", // When
used by Cloud Deploy
#     "serviceAccount:${google_service_account.deployer.email}",                // When
used by the Gitlab CI
#   ]
# }

# To allow creating public google run services

```

```

resource "google_project_iam_binding" "run_admin" {
  project = data.google_project.project.project_id
  role    = "roles/run.admin"

  members = [

    "principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}/attribute.developer_access/true",
    # "serviceAccount:${google_service_account.cloud_deploy_cloud_run.email}", // When
    used by Cloud Deploy
    "serviceAccount:${google_service_account.deployer.email}", // When used by the
    Gitlab CI
  ]
}

resource "google_project_iam_binding" "container_developer" {
  project = data.google_project.project.project_id
  role    = "roles/container.developer"

  members = [

    "principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}/attribute.developer_access/true",
    # "serviceAccount:${google_service_account.cloud_deploy_gke.email}", // When used
    by Cloud Deploy
    "serviceAccount:${google_service_account.deployer.email}", // When used by the
    Gitlab CI
  ]
}

resource "google_storage_bucket_iam_binding" "tfstate_object_viewer" {
  bucket = data.terraform_remote_state.tfstate.outputs.tfstate
  role    = "roles/storage.objectViewer"

  members = [

    "principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}/attribute.developer_access/true",
    "serviceAccount:${google_service_account.deployer.email}",
  ]
}

resource "google_storage_bucket_iam_binding" "tfstate_bucket_viewer" {
  provider = google-beta
  bucket    = data.terraform_remote_state.tfstate.outputs.tfstate
  role      = "roles/storage.bucketViewer"

  members = [

    "principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}

```

```

/attribute.developer_access/true",
  "serviceAccount:${google_service_account.deployer.email}",
]
}

resource "google_project_iam_binding" "secret_manager_accessor" {
  project = data.google_project.project.project_id
  role     = "roles/secretmanager.secretAccessor"

  members = [

"principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.developer_access/true",
    # "serviceAccount:${google_service_account.cloud_deploy_gke.email}", // When used
by Cloud Deploy
    "serviceAccount:${google_service_account.deployer.email}", // When used by the
Gitlab CI
  ]
}

resource "google_project_iam_binding" "secret_manager_viewer" {
  project = data.google_project.project.project_id
  role     = "roles/secretmanager.viewer"

  members = [

"principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}
/attribute.developer_access/true",
    # "serviceAccount:${google_service_account.cloud_deploy_gke.email}", // When used
by Cloud Deploy
    "serviceAccount:${google_service_account.deployer.email}", // When used by the
Gitlab CI
  ]
}

```

Next, please create `gcp/devops-training-workshop-sbx/global/workload-identity/outputs.tf` file in the `infrastructure` repository looking like this:

```

output "gitlab_ci_pool_name" {
  value      = google_iam_workload_identity_pool.gitlab_ci.name
  description = "Gitlab CI pool name"
}

output "deployer_service_account_email" {
  value = google_service_account.deployer.email
}

output "developer_access_principal" {
  value =
"principalSet://iam.googleapis.com/${google_iam_workload_identity_pool.gitlab_ci.name}

```

```
/attribute.developer_access/true"  
}
```

regional

core

TODO

mongodb

TODO

redis

TODO

sql / Postgres

TODO

gke

TODO

TODO (fix old practice)

Prometheus

You used Helm and Terraform to deploy prometheus to our local K3s cluster. Now let us focus how the same is done in the real production cluster.

Because our example will be using kubernetes configuration later from the terraform state instead of `.kube/config`, we are accessing state of the `kubenretes` cluster here: `infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/prometheus/data.tf`

```
data "terraform_remote_state" "gke" {  
  backend = "gcs"  
  
  config = {  
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"  
    prefix = "gcp/coe-devops-cloud-sandbox/regional/europe-central2/core"  
  }  
}
```

We are also using `module` sections which are important as help us with reusing code we are using in

each environment. In our case we are just using source with folder path to the module but this is not a valid method of doing this. Proper one is to use or terraform module repository or just git where we can use branches, tags (you should use always tags), or git hash to have module versioning. In our case all resources for the `prometheus` deployment are stored in the `main.tf` file here: `infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/prometheus/main.tf`

```
module "prometheus" {
  source = "../../../../../modules/gcp/environments/regional/core/prometheus"
}
```

Helm provider requires kubernetes configuration which we are getting in the `data.tf` file mentioned at the start. You can see this in the `providers.tf` file: `infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/prometheus/providers.tf`

```
provider "helm" {
  kubernetes {
    host =
      "https://${data.terraform_remote_state.gke.outputs.kubernetes_cluster_host}"

    client_certificate =
      base64decode(data.terraform_remote_state.gke.outputs.kubernetes_cluster_client_certificate)
    client_key =
      base64decode(data.terraform_remote_state.gke.outputs.kubernetes_cluster_client_key)
    cluster_ca_certificate =
      base64decode(data.terraform_remote_state.gke.outputs.kubernetes_cluster_ca_certificate)
  }

  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    command = "gke-gcloud-auth-plugin"
  }
}
```

Of course, we can't forget to configure terraform also. I'm doing this in the `versions.tf` where I also configure `backend`. In many cases people setup backend in the separated file called `backend.tf` but in our case we are just using one file: `infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/prometheus/versions.tf`

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/prometheus"
  }
  required_providers {
```

```

helm = {
  source = "hashicorp/helm"
  version = "~> 2"
}
}

```

Now when we see how our IaC application is set for the prometheus we need to check how **module** we are using looks.

In the case of this particular module we are using helm provider release resource. You can see this at this file: **modules/gcp/environments/regional/core/prometheus/helm.tf**

```

resource "helm_release" "prometheus" {
  name          = "prometheus"
  repository    = "https://prometheus-community.github.io/helm-charts"
  chart         = "kube-prometheus-stack"
  version       = var.chart_version
}

```

We are setting the name of the release, repository where used chart is and name of the chart to use. We are also setting up a version of this chart to use but in our case this version isn't hardcoded, but it is a variable we will explain later. To see all configuration options of this resource you can check this resource type documentation here: <https://registry.terraform.io/providers/hashicorp/helm/latest/docs/resources/release>

Mentioned upper variable is set in the **variables.tf** file and also in our case this variable have a default value. You can see this here: **modules/gcp/environments/regional/core/prometheus/variables.tf**

```

variable "chart_version" {
  default = "48.3.3"
  description = "Helm chart version: https://github.com/prometheus-community/helm-charts/blob/main/charts/kube-prometheus-stack/Chart.yaml"
}

```

Modules should also have a **versions.tf** file showing which versions of terraform and used providers can be used with this module. You can see this here: **modules/gcp/environments/regional/core/prometheus/versions.tf**

```

terraform {
  required_version = "~> 1"
  required_providers {
    helm = {
      source = "hashicorp/helm"
      version = "~> 2"
    }
  }
}

```

```
}
```

We saw our prometheus module for

[infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/data.tf](#)

```
data "terraform_remote_state" "management_global_dns" {
  backend = "gcs"

  config = {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/management/global/dns"
  }
}

data "terraform_remote_state" "management_global_projects" {
  backend = "gcs"

  config = {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/management/global/projects"
  }
}
```

[infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/main.tf](#)

```
// gcloud container clusters get-credentials $(terraform output -raw
kubernetes_cluster_name) --region $(terraform output -raw region) --project
$(terraform output -raw project_id)
// gcloud container clusters update --disable-managed-prometheus $(terraform output
-raw kubernetes_cluster_name) --region $(terraform output -raw region) --project
$(terraform output -raw project_id)
module "core" {
  source = "../../../../../modules/gcp/environments/regional/core"
}
```

[infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/outputs.tf](#)

```
output "region" {
  value      = module.core.region
  description = "GCloud Region"
}

output "project_id" {
  value      = module.core.project_id
  description = "GCloud Project ID"
}
```

```

output "kubernetes_cluster_name" {
  value      = module.core.kubernetes_cluster_name
  description = "GKE Cluster Name"
}

output "kubernetes_cluster_host" {
  value      = module.core.kubernetes_cluster_host
  description = "GKE Cluster Host"
}

output "kubernetes_cluster_client_certificate" {
  value = module.core.kubernetes_cluster_client_certificate
  sensitive = true
}

output "kubernetes_cluster_client_key" {
  value = module.core.kubernetes_cluster_client_key
  sensitive = true
}

output "kubernetes_cluster_ca_certificate" {
  value = module.core.kubernetes_cluster_ca_certificate
  sensitive = true
}

```

[gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/providers.tf](#)

```

provider "google" {
  project = "coe-devops-cloud-sandbox"
  region  = "europe-central2"
  zone    = "europe-central2-b"
}

```

[infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/versions.tf](#)

```

terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/coe-devops-cloud-sandbox/regional/europe-central2/core"
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 4"
    }
  }
}

```

You can compare this real life solution with our example solution created for K3s now.

Kubernetes

We are creating our GKS Kubernetes cluster also with the use of our IaC application. Here you will have example of how this can be done. State generated by this code was used by our GCP prometheus deployment example upper.

Our code is also using remote state of other parts of our IaC application. We are data which were created for a management part of our Infrastructure. You can see also that we are having global and regional type of resources by reading this file: [infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/data.tf](#)

```
data "terraform_remote_state" "management_global_dns" {
  backend = "gcs"

  config = {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/management/global/dns"
  }
}

data "terraform_remote_state" "management_global_projects" {
  backend = "gcs"

  config = {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/management/global/projects"
  }
}
```

In our case we do not have separate clusters per environment but one cluster for all. This is just purely for economic reasons, but clusters we are creating at least are regional, and we are reusing prepared module for this. You can see this by checking this file: [infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/main.tf](#)

```
// gcloud container clusters get-credentials $(terraform output -raw
kubernetes_cluster_name) --region $(terraform output -raw region) --project
$(terraform output -raw project_id)
// gcloud container clusters update --disable-managed-prometheus $(terraform output
-raw kubernetes_cluster_name) --region $(terraform output -raw region) --project
$(terraform output -raw project_id)
module "core" {
  source = "../../../../../modules/gcp/environments/regional/core"
}
```

We are also creating outputs. Some of them were used in our prometheus example and some of them like you see are also **sensitive**. Terraform will do what it can to protect those values from showing but please remember that all values in the state are in the plain text, and You need to secure and encrypt this state by properly configuring remote state resource (in our case GCS bucket). You can see our outputs code here: [infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/outputs.tf](#)

```
output "region" {
  value      = module.core.region
  description = "GCloud Region"
}

output "project_id" {
  value      = module.core.project_id
  description = "GCloud Project ID"
}

output "kubernetes_cluster_name" {
  value      = module.core.kubernetes_cluster_name
  description = "GKE Cluster Name"
}

output "kubernetes_cluster_host" {
  value      = module.core.kubernetes_cluster_host
  description = "GKE Cluster Host"
}

output "kubernetes_cluster_client_certificate" {
  value = module.core.kubernetes_cluster_client_certificate
  sensitive = true
}

output "kubernetes_cluster_client_key" {
  value = module.core.kubernetes_cluster_client_key
  sensitive = true
}

output "kubernetes_cluster_ca_certificate" {
  value = module.core.kubernetes_cluster_ca_certificate
  sensitive = true
}
```

In this case we are using **google** provider and we are configuring it here: [gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/providers.tf](#)

```
provider "google" {
  project = "coe-devops-cloud-sandbox"
  region  = "europe-central2"
  zone    = "europe-central2-b"
```

```
}
```

We also need to configure terraform here: [infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/versions.tf](#)

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/coe-devops-cloud-sandbox/regional/europe-central2/core"
  }
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 4"
    }
  }
}
```

Now let us focus on the used module.

Our module is using other data types which are not related in this case to the state. You can see them here: [modules/gcp/environments/regional/core/data.tf](#)

```
data "google_project" "project" {}

data "google_client_config" "this" {
  provider = google
}

data "google_container_engine_versions" "gke_version" {
  version_prefix = "1.27."
}
```

We do not have separated IaC code for the VPC and just creating those resources in this file: [modules/gcp/environments/regional/core/vpc.tf](#)

```
# VPC
resource "google_compute_network" "vpc" {
  name = "${data.google_project.project.project_id}-vpc"
  auto_create_subnetworks = "false"
}

# Subnet
resource "google_compute_subnetwork" "subnet" {
  name = "${data.google_project.project.project_id}-subnet"
  network = google_compute_network.vpc.name
  ip_cidr_range = "10.10.0.0/24"
}
```

```
}
```

It is main reason I named this module **core**.

We are also expecting some variables to be used but we also in this example have a default variable. If you do not set default variable you should set type of the variable. Please look at this file: [modules/gcp/environments/regional/core/variables.tf](#)

```
variable "gke_num_nodes" {  
  default      = 1  
  description = "number of gke nodes"  
}
```

In our case by checking default value terraform will know that our type is **number**.

Now when we showed how our VPC is created let us look how our cluster is created here: [modules/gcp/environments/regional/core/gke.tf](#)

```
resource "google_container_cluster" "primary" {  
  name      = "${data.google_project.project.project_id}-gke"  
  location = data.google_client_config.this.region  
  
  # We can't create a cluster with no node pool defined, but we want to only use  
  # separately managed node pools. So we create the smallest possible default  
  # node pool and immediately delete it.  
  remove_default_node_pool = true  
  initial_node_count       = 1  
  
  network    = google_compute_network.vpc.name  
  subnetwork = google_compute_subnetwork.subnet.name  
  
  monitoring_config {  
    enable_components = ["SYSTEM_COMPONENTS"]  
    managed_prometheus {  
      enabled = false  
    }  
  }  
}  
  
maintenance_policy {  
  daily_maintenance_window {  
    start_time = "04:00"  
  }  
}  
}  
  
resource "google_container_node_pool" "primary_nodes" {  
  name      = google_container_cluster.primary.name  
  cluster   = google_container_cluster.primary.name  
  location  = data.google_client_config.this.region
```

```

version      =
data.google_container_engine_versions.gke_version.release_channel_latest_version["STABLE"]
node_count   = var.gke_num_nodes

node_config {
  oauth_scopes = [
    "https://www.googleapis.com/auth/compute",
    "https://www.googleapis.com/auth/devstorage.read_only",
    "https://www.googleapis.com/auth/logging.write",
    "https://www.googleapis.com/auth/monitoring",
  ]

  labels = {
    env = data.google_project.project.project_id
  }

  machine_type = "n1-standard-2"
  tags         = ["gke-node", "${data.google_project.project.project_id}-gke"]
  metadata     = {
    disable-legacy-endpoints = "true"
  }
}
}

```

As you can see to create GKS cluster you do not need a lot.

Outputs we are using later are set here: [modules/gcp/environments/regional/core/outputs.tf](#)

```

output "region" {
  value      = data.google_client_config.this.region
  description = "GCloud Region"
}

output "project_id" {
  value      = data.google_project.project.project_id
  description = "GCloud Project ID"
}

output "kubernetes_cluster_name" {
  value      = google_container_cluster.primary.name
  description = "GKE Cluster Name"
}

output "kubernetes_cluster_host" {
  value      = google_container_cluster.primary.endpoint
  description = "GKE Cluster Host"
}

```

```

output "kubernetes_cluster_client_certificate" {
  value = google_container_cluster.primary.master_auth.0.client_certificate
  sensitive = true
}

output "kubernetes_cluster_client_key" {
  value = google_container_cluster.primary.master_auth.0.client_key
  sensitive = true
}

output "kubernetes_cluster_ca_certificate" {
  value = google_container_cluster.primary.master_auth.0.cluster_ca_certificate
  sensitive = true
}

```

And we also need to set requirements for terraform and provider here: [modules/gcp/environments/regional/core/versions.tf](#)

```

terraform {
  required_version = "~> 1"
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 4"
    }
  }
}

```

And with help of this code and some other management resources we have our cluster!

SOPS with GCP KMS Example

One of very important topics still not mentioned is how to store secrets used by Terraform. In our case we will use SOPS tool for this configured for our GCP and using KMS. Let us use our postgresql database code.

Because we are using helm provider to deploy database inside our cluster we need the same as a prometheus data related to our cluster here: [infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/postgresql/data.tf](#)

```

data "terraform_remote_state" "gke" {
  backend = "gcs"

  config = {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/coe-devops-cloud-sandbox/regional/europe-central2/core"
  }
}

```

```
# Encrypt
# sops --encrypt --gcp-kms projects/coe-devops-cloud-
sandbox/locations/europe/keyRings/sops/cryptoKeys/sops-key my-secrets.yml > my-
secrets.enc.yml

data "sops_file" "demo-secret" {
  source_file = "my-secrets.enc.yml"
  input_type  = "yaml"
}
```

You see also that we have a new data type `sops_file` which is using `file` resource type in the `sops` provider.

File mentioned upper looks like this: `infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/postgresql/my-secrets.enc.yml`

```
mobica-workshops-staging-postgresql-rootPassword:
ENC[AES256_GCM,data:ftBQ7VratZkiQGdXC2z9zA==,iv:07ra3CWeW1JFrE+Ae2gUHD4sAw19zL0ZZ31tJ/A4VLc=,tag:ddxYp/CRl4t/07kTRM/Cw==,type:str]
mobica-workshops-staging-postgresql-bookListPassword:
ENC[AES256_GCM,data:3YbB1Hu1x4oib4fKCKNDaw==,iv:zYn0YeZzRYHrDun9Hz7/R71B0vkDPx+6XV2DzMOTprs=,tag:mY9Af1M9ELK997Ay4edJqA==,type:str]
mobica-workshops-production-postgresql-rootPassword:
ENC[AES256_GCM,data:4wCJsxgLdXo/rAAHoVAYSw==,iv:hG0Ds8S+BJe31cC9+Am9dWfqZxKm31R73eKcEEOWaiI=,tag:90J02XiJnf4vaVS8T0+Weg==,type:str]
mobica-workshops-production-postgresql-bookListPassword:
ENC[AES256_GCM,data:B5Z4uxTRX8p9fHkrgMgb+w==,iv:42YbqgQaXdkaNhGW5iog1dMuaK/KoFqKRkXH8ygmDfk=,tag:E5Wf6GXiuY3g9QG1bVx96Q==,type:str]
sops:
  kms: []
  gcp_kms:
    - resource_id: projects/coe-devops-cloud-
      sandbox/locations/europe/keyRings/sops/cryptoKeys/sops-key
      created_at: "2023-08-28T07:48:38Z"
  enc:
CiQA9sIYT8PlsbRmYp6glfIth2GNlEtOHtthR1PMjw0ZFmL2Fx8SSQAnRfr/w3pgUb1rAnNa4tzcdSMwX4Uk90P3+Zfdmt7Ct09cuogQrvHz9J0leGakfrxDDQS1LKXEX9WjT5EDIYN3U99y7WrU+E=
  azure_kv: []
  hc_vault: []
  age: []
  lastmodified: "2023-08-28T07:48:38Z"
  mac:
ENC[AES256_GCM,data:dv5y03L+zkCXvT/ngeoYTkjG+DCpM/2/rtrRRniaH63WrYf2g5dsLjRZb7N6k8yB7MDnvrQhm32WiQF824h3vDS6q/DVzHUaNOggAhr2xWQMm0JN8xoYnhK2/zJKjulepZNGltrDyjj1YuL6ALvSE16vGpddtIv5TbsdGJQ04Kc=,iv:/cSpXsIza0acbPJ8scoFNLfshB8UG9Ff7rIMpCUvZlk=,tag:pxak/Xhz1AzCu+0NWMS2Dg==,type:str]
  pgp: []
  unencrypted_suffix: _unencrypted
```

In our example we are not having separate folders for every environment as we are using just once cluster, but you can see that we are using exactly the same module for each environment here: [infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/postgresql/main.tf](#)

```
module "workshops-staging" {
  source      =
  "../../../../../modules/gcp/environments/regional/core/postgresql"
  namespace   = "workshops-staging"
  rootPassword = data.sops_file.demo-secret.data.mobica-workshops-staging-
postgresql-rootPassword
  bookListPassword = data.sops_file.demo-secret.data.mobica-workshops-staging-
postgresql-bookListPassword
}

module "workshops-production" {
  source      =
  "../../../../../modules/gcp/environments/regional/core/postgresql"
  namespace   = "workshops-production"
  rootPassword = data.sops_file.demo-secret.data.mobica-workshops-production-
postgresql-rootPassword
  bookListPassword = data.sops_file.demo-secret.data.mobica-workshops-production-
postgresql-bookListPassword
}
```

You should also see that for each database we have own variables which will be decoded by SOPS and given to the module.

Let us look to the one more file [versions.tf](#) which are showing how to set sops to be available for us here: [infrastructure/gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/postgresql/versions.tf](#)

```
terraform {
  required_version = "~> 1"
  backend "gcs" {
    bucket = "coe-devops-cloud-admin-ca18429efb894d39-tfstate"
    prefix = "gcp/coe-devops-cloud-sandbox/regional/europe-central2/core/postgresql"
  }
  required_providers {
    helm = {
      source = "hashicorp/helm"
      version = "~> 2"
    }
    sops = {
      source = "carlpett/sops"
    }
  }
}
```

```
}
```

With this example you can see how to secure data our terraform is using.

What Next

Learning paths

After this workshop you can take those learning paths:

- A Cloud Guru Containers and Kubernetes related trainings - where you can learn more about Docker, Kubernetes, and Helm
- A Cloud Guru DevOps automation related trainings - where you can learn more about Configuration Management tools like by example Ansible and many type of Pipelines

Next workshops in the microservices series

Backend

For a Backend developers we have this workshop in the series:

- Backend local development with Docker and K3s for project using microservices architecture

Current agenda for the third edition looks like this:

- Start 10:00
- Section 1:
 - Architecture
 - Automation
 - Configuration
 - Service Skeleton
 - Readiness Checklist
 - Documentation
 - Services
 - Mockups
- Coffee break 11:00 - 11:15
- Section 2:
 - Book List API (demo)
- Coffee break 12:30 - 12:45
- Section 3:

- Book Admin API (practice)
- Lunch 13:30 - 14:00
- Section 4:
 - Backend For Frontend (demo + practice)
- Coffee break 15:15 - 15:30
- Section 5:
 - Cleanup
 - What's Next
 - Q&A
- Ends between 16:00 and 17:00 - depends on Q&A session

Frontend

For a Frontend developers we have this workshop in the series:

- Frontend local development with Docker and K3s for project using microservices architecture

Current agenda for the first edition looks like this:

- Start 10:00
- Section 1:
 - Architecture
 - Automation
 - Configuration
 - Service Skeleton
 - Readiness Checklist
 - Documentation
 - Services
- Coffee break 11:00 - 11:15
- Section 2:
 - Book Frontend (demo + practice)
- Coffee break 12:30 - 12:45
- Section 3:
 - What's Next
 - Cleanup
 - Q&A
- Ends between 13:00 and 13:30 - depends on Q&A session

DevOps Automator and QA

For a DevOps Automator and QA engineers we plan to release:

- Continuous Integration, and Continuous Delivery/Deployment for project using microservices architecture with a help of the GITLAB platform. Delivery/Deployment will be targeting prepared K8s cluster.

Current agenda for the first edition looks like this:

- Start 10:00
- Section 1:
 - Architecture Overview
 - Gitlab Overview
 - Frontend Service Pipeline Overview
 - Backend Services Pipelines Overview
 - Gitlab Container Registry Overview
 - Gitlab Pages Overview
 - Gitlab CI Overview
- Coffee break 11:00 - 11:15
- Section 2:
 - Your first pipeline (practice + explanation)
 - A complex pipeline (practice + explanation)
- Coffee break 12:10 - 12:25
- Section 3:
 - DevOps pipeline for our frontend microservice (demo)
 - DevOps pipeline for our frontend microservice (explanation)
- Section 4:
 - DevOps pipeline for our backend microservices (demo)
 - DevOps pipeline for our backend microservices (explanation)
- Lunch 13:45 - 14:15
- Section 5:
 - Simplified pipeline for our frontend app (explanation)
 - Simplified pipeline for our frontend app (practice)
- Coffee break 15:45 - 16:00
- Section 6:
 - What's Next
 - Cleanup

- Q&A

- Ends between 16:15 and 16:30 - depends on Q&A session

Related Open Source project:

We have also a special OpenSource project MOB175:

- Preparing examples used in the workshops series connected to the microservices architecture.

Anyone who is interested can join in a free time and people who are currently without project can let us know if wanted to join this project. Please contact me to gain more details.

Cleanup

You can remove created K3d cluster created with this command:

```
k3d cluster delete bookCluster
```

You can clean up your docker engine with this command:

```
docker system prune -a --volumes
```

All tools installed with a help of the **brew** can be uninstalled with the command **brew uninstall** plus **tool name**.

```
brew uninstall tool-name
```

QnA

Waiting for Questions :)



This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/)