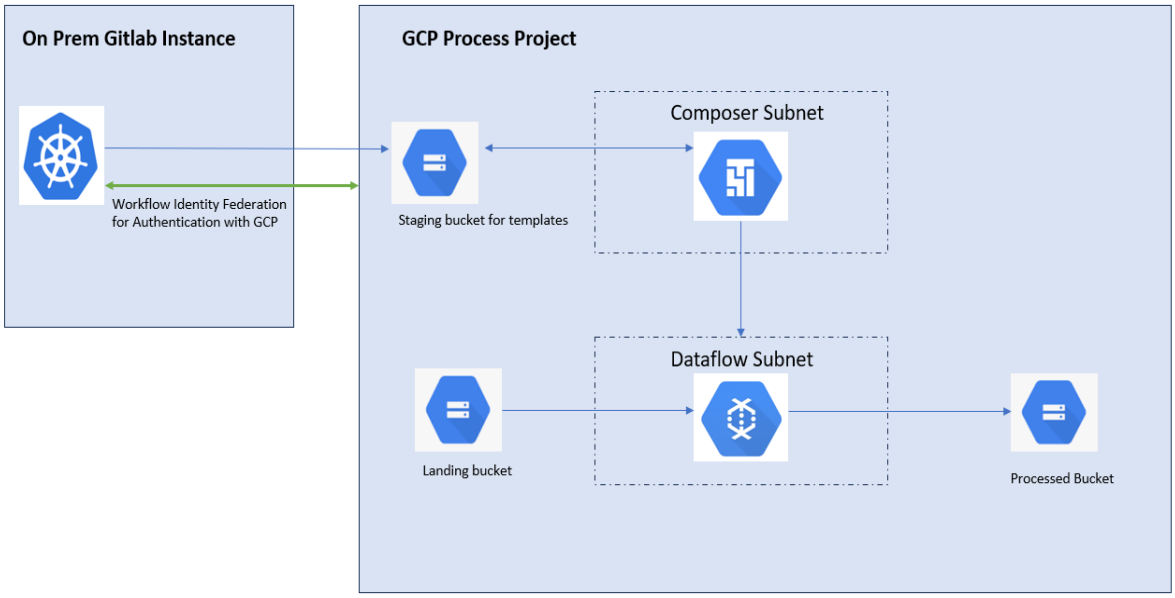
Design Document for Apache Beam Batch Dataflow Job Deployment on Cloud Composer

Date: 06/10/2024

Author: Ananth Pradeep Kumar

Version: v1.0.0

Architecture Overview



1. Create VPC, Subnets, Firewalls

Create VPC, Sperate subnetwork for dataflow and composer and provide the necessary firewall rules.

```
module "vpc" {
  source       = "terraform-google-modules/network/google"
  version      = "~> 9.0"
  project_id   = var.project_id
  network_name = "vpc-network"

  subnets = [
   {
    subnet_name   = "dataflow-subnetwork"
    subnet_ip     = "10.1.3.0/24"
    subnet_region = "europe-west2"
   },
          {
    subnet_name   = "composer-subnetwork"
    subnet_ip     = "17.23.3.0/24"
    subnet_region = "europe-west2"
   },
  ]
}
```

2. Create Service Accounts

Create a service account for terraform, dataflow and composer. Terraform service account should have necessary privilege to create/modify/destroy resources like cloud storage. Composer service account should have composer worker privilege and dataflow jobs create permissions. Similarly, dataflow service account should have dataflow worker privilege. These are required at minimum.

3. Create Buckets using Terraform

Create a bucket for storing input files in the landing bucket, a bucket for storing the output created by the dataflow job, and another bucket for storing artefacts and configurations. (Gcs buckets can also we be created using terraform)

4. Create the Composer environment

Create the composer environment with the subnet ranges mentioned in the composer subnet, provide worker, scheduler, configurations based on the load, choose the composer and airflow latest versions. Provide Pypi packages.

```
resource "google_composer_environment" "composer-env" {
  name   = "composer-env"
  region = "europe-west2"
 config {
    software_config {
     image_version = "composer-2-airflow-2"
    }
            node_config {
     network = "<vpc network created>"
     subnetwork = "<vpc subnet cerated>"
    }
     pypi_packages = {
       numpy = ""
       google-cloud-storage = "==2.18.2"
     }
  }
}
```

5. Deploying the artefacts file to GCS from Gitlab

a. Deploy the code into the Gitlab repository. (Assuming the Gitlab runners will be on On-prem Kubernetes pods). The Gitlab repositories needs to establish a connectivity to the GCP projects in different environments. This needs to be done through Workflow Identity Federation (WIF). This will enable to establish a connectivity to GCP without service account key file.

b. From Gitlab execute `mvn clean install` to create the jar file and copy this jar file into a gcs bucket created for storing the artefacts.

6. Create dataflow Using Terraform

In this step, the dataflow job can be created by using the dataflow terraform module. We can mention, job name, vpc, subnet, meachine type, and the jar file to be executed in templte_gcs_path etc.. This is a terraform configuration and when it is executed (terraform plan and terraform apply) from an external source (say for example composer) the dataflow-job will be executed.

```
module "dataflow-job" {
  source  = "terraform-google-modules/dataflow/google"
  version = "~> 2.0"

  project_id          = var.project_id
  name                = "batch-file-executor-df"
  on_delete           = "cancel"
  region              = var.region
  zone                = var.zone
  max_workers         = 10
  template_gcs_path     = "gs://dataflow-templates/dataflow-executor.jar"
  temp_gcs_location     = “gs://dataflpw-temp-bucket”
  service_account_email = var.service_account_email
  network_self_link     = module.vpc.network_self_link
  subnetwork_self_link  = module.vpc.subnets_self_links[0]
  machine_type          = "n1-standard-4"

  parameters = {
    inputFile = "gs://dataflow-input /shakespeare/kinglear.txt"
    output    = "gs dataflow-output/output/my_output"
  }
}
```

7. Run dataflow Using Composer

Once the terraform module for the dataflow job is created, it can be executed using an airflow DAG. A bash operator is used to execute terraform plan and apply commands. The above configuration needs to be accessible in composer. If there are terraform variables is used, use that as well while executing.
Check the following DAG to see how the terraform plan and apply can be executed.

Note: The Dataflow JAR file can be deployed to GCS and called using the DataflowTemplateOperator. With this operator, the template location (JAR file path), input file, and output file path can be passed and executed from the DAG. However, this is not covered in this document as it was not part of the requirement.

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

default_args = {
    'start_date': datetime(2024, 10, 6),
    'retries': 1,
}

with DAG(
    dag_id='trigger_terraform_dataflow_job',
    default_args=default_args,
    schedule_interval='@daily',
    catchup=False
) as dag:

    terraform_plan = BashOperator(
        task_id='terraform_plan',
        bash_command='cd /path/to/terraform && terraform plan -out=tfplan'
    )
    terraform_apply = BashOperator(
        task_id='terraform_apply',
        bash_command='cd /path/to/terraform && terraform apply -auto-approve tfplan'
    )
    terraform_plan >> terraform_apply
```