# Team Members

| | |
|---|---|
| Ali Taha | Developer |
| Ahmad Shata | DevOps Engineer |
| Andrew Anter | Solution Architect |
| Merit Mekhail | Scrum Master |
| Hadi Lotfy | Product Owner |

ITI - Devops Intensive Training - GCP Course

# Airflow DAG in Cloud Composer

# Agenda

- Product Owner
  - Product Backlog
- Scrum Master
  - Sprint backlog
- Solution Architect
  - What is composer?
  - Why composer?
- Devops Engineer
  - GCP
  - Terraform
- Developer
  - DAG

# Product Backlog

| PRODUCT BACKLOG | | | | | |
|---|---|---|---|---|---|
| User Story ID | User Story | Estimate (size) | Priority | Sprint | Estimated effort |
| US001 | **As** Admin<br>**I need to** take backup our database periodically<br>**So that** we can return to same state in case of failure | large | 1 | 1 | 24h |
| US002 | **As** Admin<br>**I need to** delete old backups<br>**So that** we save storage | small | 3 | 3 | 7h |
| US003 | **As** Admin<br>**I need to** see a list of all members and visitors<br>**So that** I monitor site visits | large | 2 | 2 | 24h |
| US004 | **As** Customer<br>**I need to** be able to login<br>**So that** I can post new entries | medium | 2 | 2 | 15h |

# Sprint Backlog

| SPRINT 1 BACKLOG | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID | User Story | Tasks | Owner | Status | Day 1 | Day 2 | Day 3 | Day 4 | Day 5 | Day 6 | Day 7 | Estimated effort |
| US001 | **As** Admin<br>**I need to** take backup our database<br>**So that** we can return to same state in case of failure | Create VPC and Subnet using Terraform | Shata | Done | 4 | | | | | | | 24 |
| | | Create Composer | Shata | Done | | 3 | 4 | | | | | |
| | | Composer Documentation | Shata | Done | | | | 3 | | | | |
| | | Create a DAG on the composer to Automate taking machine image Task | Ali | Done | | | | 4 | 3 | | | |
| | | DAG Documentation | Ali | Done | | | | | | 3 | | |
| | | | | **Total** | 4 | 3 | 4 | 7 | 3 | | | 24 |

Sprint Duration: One Week

# What is Composer?

## Apache Airflow

- A platform created by the community to programmatically author, schedule and monitor workflows.

## Google Cloud Composer

- A fully managed workflow orchestration service built on Apache Airflow.

# Why Composer?

## Benefits Of Using Cloud Composer

- Multi-cloud
- Open source
- Hybrid
- Integrated
- Python programming language
- Reliability
- Fully managed
- Networking and security

# Spinning up our Infrastructure

**We are going to use Terraform as our IaC utility rather than creating the infrastructure directly from the console to exploit the Terraform idempotency, modulation, and backend**

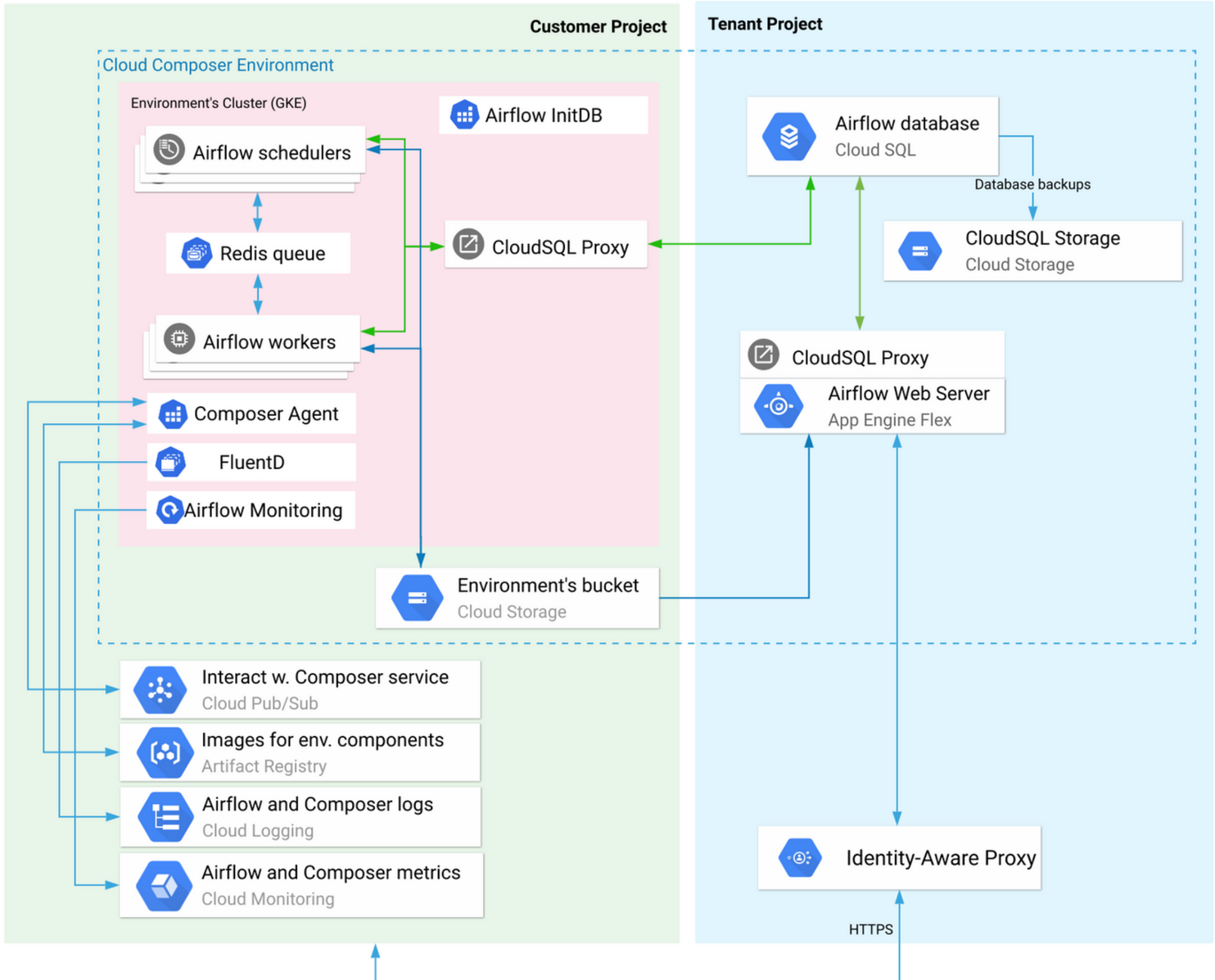
HashiCorp Terraform

# Network Module

```
resource "google_compute_network" "net" {
  project                         = "shata-387907"
  name                            = var.vpc-name
  auto_create_subnetworks         = false
}

resource "google_compute_subnetwork" "subnet" {
  name                            = var.subnet-name
  project                         = "shata-387907"
  private_ip_google_access        = var.private-google-access
  network                         = google_compute_network.net.id
  ip_cidr_range                   = var.subnet-cidr
  region                          = var.subnet-region
  secondary_ip_range {
    range_name                    = var.secondary-ip-range-1-name
    ip_cidr_range                 = var.secondary-ip-range-1-cidr
  }
  secondary_ip_range {
    range_name                    = var.secondary-ip-range-2-name
    ip_cidr_range                 = var.secondary-ip-range-2-cidr
  }
}
```

# Composer Structure Depiction



# Network Module Output

```
output "vpc-uri" {
  value = google_compute_network.net.self_link
}
output "subnet-uri" {
  value = google_compute_subnetwork.subnet.self_link
}
```

# Enabling Composer API

```
resource "google_project_service" "composer_api" {
  #Provider cannot be used as a variable
  provider                   =  google-beta
  project                    =  "shata-387907"
  service                    =  "composer.googleapis.com"
  disable_on_destroy         =  false
}
```

# Composer Module

```
# The composer env
resource "google_composer_environment" "test" {
  project                  =  var.project-id
  name                     =  var.composer-name
  region                   =  var.composer-region
  config {
    environment_size       =  var.environment-size

    node_config {

      network              =  var.vpc-uri
      subnetwork           =  var.subnet-uri

      service_account      =  "${google_service_account.test.email}"
    }

    software_config {
      image_version        =  var.composer-version
    }
  }

  depends_on = [
    google_project_iam_member.composer-worker,
    google_project_iam_member.composer-ext,
    google_service_account_iam_member.API_service_account
    ]
```

# Service Account

```
# Create a service account for the cluster nodes with minimum Privileges
resource "google_service_account" "test" {
  project                 =  var.project-id
  account_id              =  var.nodes-service-account-name
  display_name            =  var.nodes-service-acoount-description
}
```

# Least SA privilege

```
# Add a composer worker role to the service account
resource "google_project_iam_member" "composer-worker" {
  project                 = var.project-id
  role                    = "roles/composer.worker"
  member                  = "serviceAccount:${google_service_account.test.email}"
}
# Add a Cloud Composer v2 API Service Agent Extensio role to the service account (only required for composer V2)
resource "google_project_iam_member" "composer-ext" {
  project                 = var.project-id
  role                    = "roles/composer.ServiceAgentV2Ext"
  member                  = "serviceAccount:${google_service_account.test.email}"
}
```

# Custom IAM Role

```
resource "google_project_iam_custom_role" "my-composer-sa-role" {
  role_id     = "composer-sa-role"
  title       = "My Instance Role"
  description = "my custom iam role"
  permissions = [
    "compute.machineImages.create",
    "compute.disks.createSnapshot",
    "compute.instances.useReadOnly",
    "compute.instances.start",
    "compute.instances.stop"
  ]
}
```

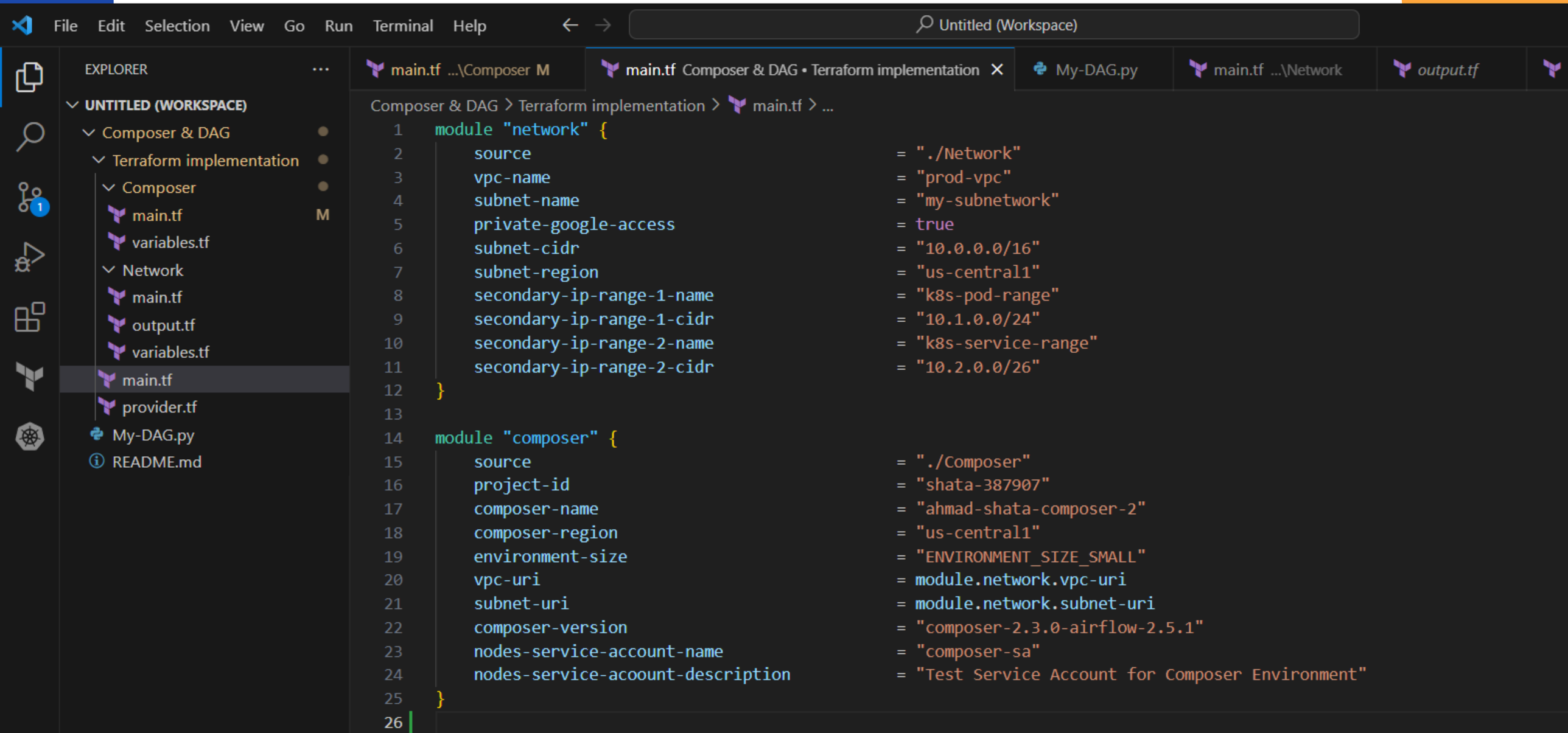
# Attaching the custom role

```
resource "google_project_iam_member" "composer-sa-custom-role" {
  project                 = var.project-id
  role                    = var.google_project_iam_custom_role.name
  member                  = "serviceAccount:${google_service_account.test.email}"
}
```

# Adjusting Service Agent

```
resource "google_service_account_iam_member" "API_service_account" {
  provider               = google-beta
  service_account_id     = google_service_account.test.id
  role                   = "roles/composer.ServiceAgentV2Ext"
  member                 = "serviceAccount:service-421061705201@cloudcomposer-accounts.iam.gserviceaccount.com"
}
```

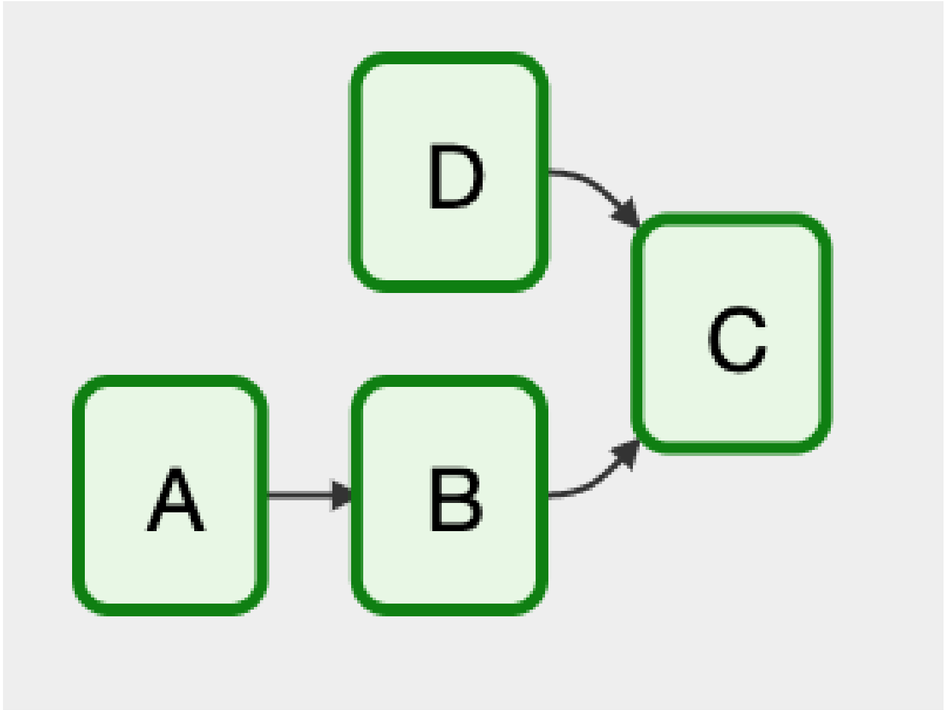
# Calling our Modules

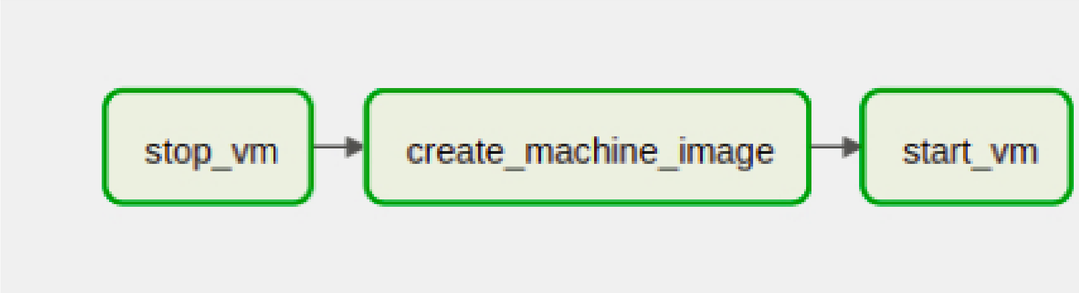


# BACKUP DAG FILE

## WHAT IS DAG

A DAG (Directed Acyclic Graph) is the core concept of Airflow, collecting Tasks together, organized with dependencies and relationships to say how they should run



# BACKUP DAG FILE

```
default_args = {
    'start_date': airflow.utils.dates.days_ago(0),
    'retries': 0
}
```

- The default_args dictionary contains default configuration parameters for the DAG.
- The 'start_date': airflow.utils.dates.days_ago(0) specifies the start date for the DAG as the current date.
- The 'retries': 0 specifies that the DAG should not be retried in case of failure. Change this value if you want to allow retries.

```
dag = DAG(
    'backup',
    default_args=default_args,
    description='backup',
    schedule_interval='0 3 * * 5',
    max_active_runs=2,
    catchup=False,
    dagrun_timeout=timedelta(minutes=10),
)
```

- The dag variable creates an instance of the DAG class.
- The 'backup' parameter is the DAG's identifier.
- The default_args=default_args sets the default configuration parameters for the DAG.
- The 'backup' string in the description parameter provides a brief description of the DAG.
- The '0 3 * * 5' schedule_interval parameter sets the schedule to run the DAG every Friday at 3 AM.
- The max_active_runs=2 parameter specifies that a maximum of two DAG runs can be active at the same time.
- The catchup=False parameter prevents Airflow from running any missed or overdue DAG runs when the DAG is first created.
- The dagrun_timeout=timedelta(minutes=10) parameter sets a timeout duration of 10 minutes for each DAG run.

# BACKUP DAG FILE

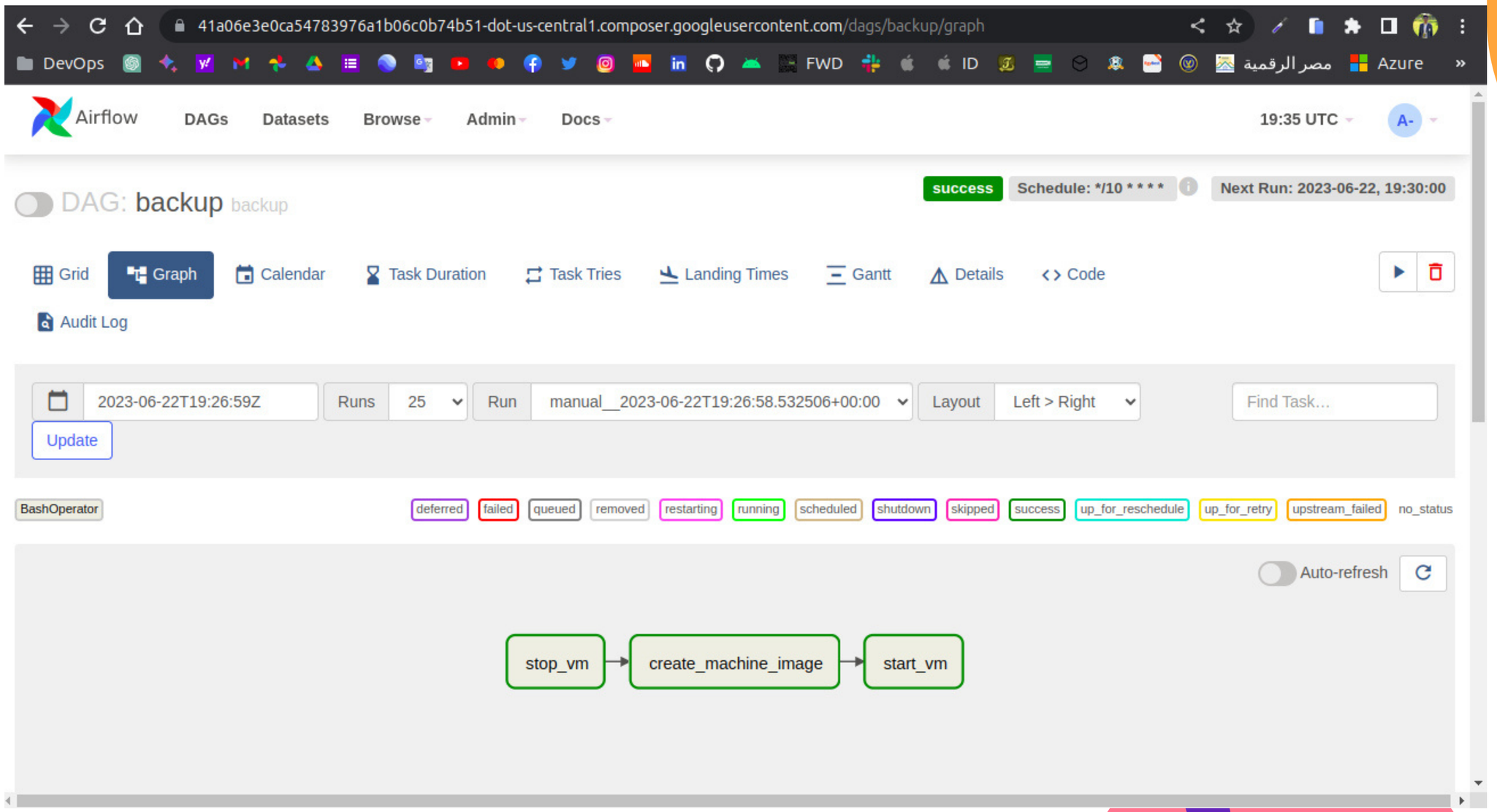
```
stop_vm = BashOperator(
    task_id='stop_vm',
    bash_command='gcloud compute instances stop instance --zone=us-central1-a',
    dag=dag,
    depends_on_past=False,
    priority_weight=2**31 - 1,
    do_xcom_push=False
)
```

- The stop_vm variable creates an instance of the BashOperator class to stop a virtual machine instance.
- The 'stop_vm' parameter sets the task's identifier.
- The 'gcloud compute instances stop instance --zone=us-central1-a' bash_command parameter specifies the Bash command to execute. It stops a virtual machine instance named 'instance' in the 'us-central1-a' zone using the gcloud command.
- The dag=dag parameter assigns the task to the defined DAG.
- The depends_on_past=False parameter indicates that the task does not depend on the past successful runs of other tasks.
- The priority_weight=2**31 - 1 parameter sets the priority weight of the task to the maximum value.
- The do_xcom_push=False parameter disables the pushing of task output to XCom.

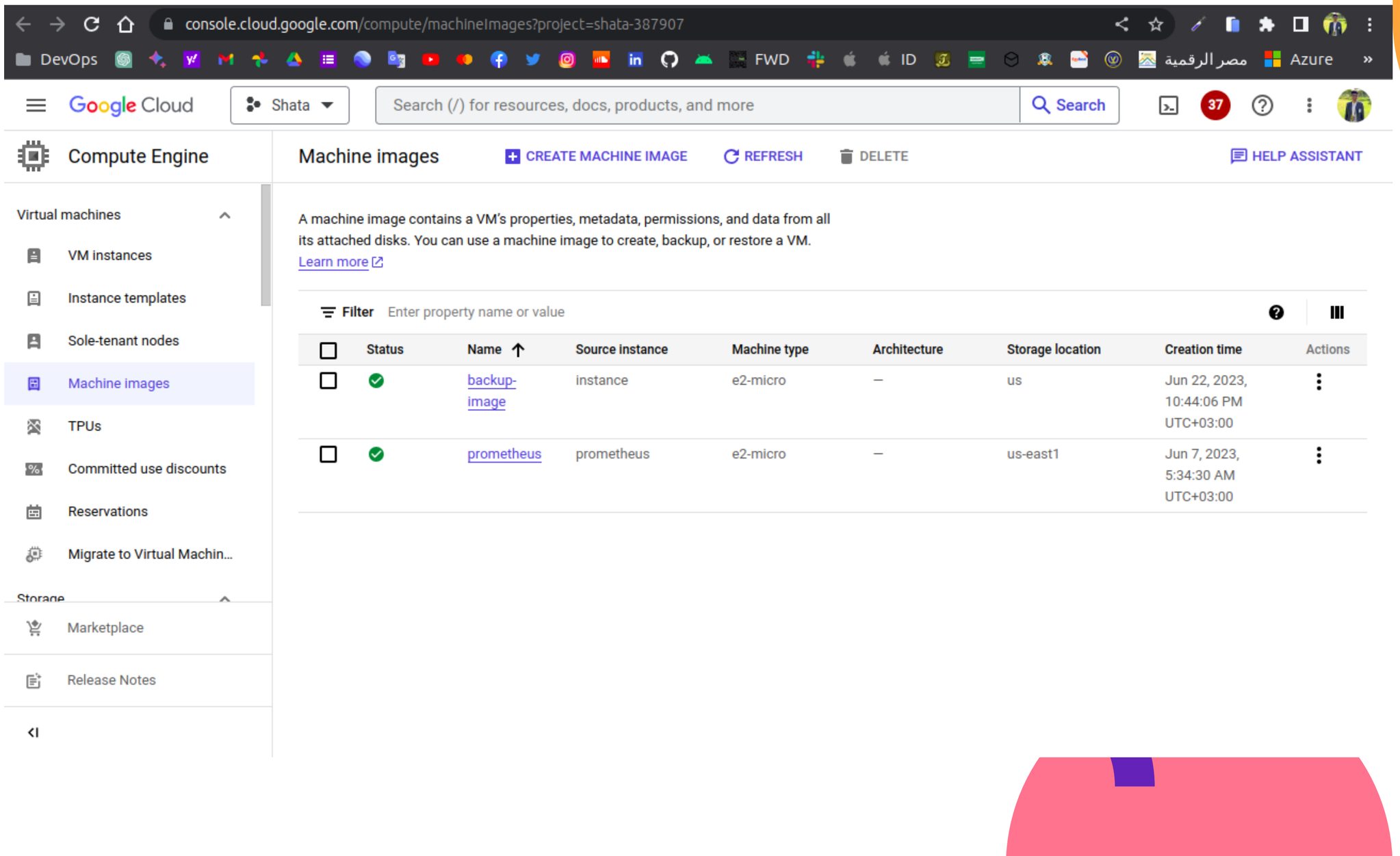
```
stop_vm >> create_machine_image >> start_vm
```

- The >> operator connects the tasks to define their dependencies. Here, the stop_vm task must complete successfully before the create_machine_image task can start, and the create_machine_image task must complete successfully before the start_vm task can start.

# BACKUP DAG FILE



# BACKUP DAG FILE



# Thank you!