תכנון שירותים מבוססי ענן – תרגיל בית 2

כרים גבארין - 211406343

אחמד גבארין - 314722307

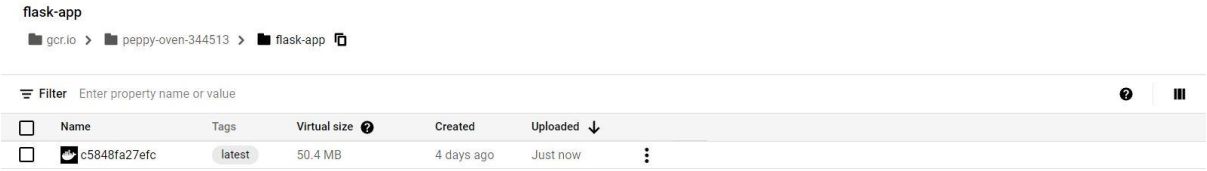
תימאא בסול - 208869222

חלק 1 (יצירת האפליקציה):

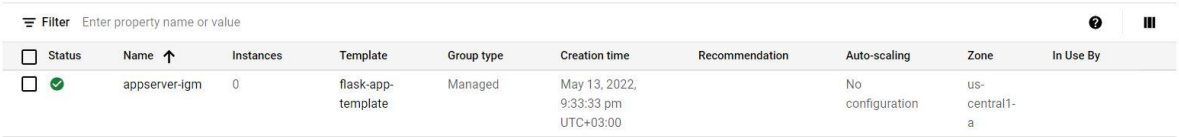
ביצענו את השלבים כפי שמתואר בלינק המצורף לחלק הזה. אחר כך, יצרנו את הקובץ Dockerfile שנמצא בתיקייה המצורפת לקובץ הזה. ואז הרצנו את השרת על ידי הפקודות שנמצאות בתום הדף שמצורף לחלק הזה ב- git bash. כדי להתחבר לשרת דרך הדפדפן, הכנסנו localhost:5000 או אפשר גם להכניס 127.0.0.1:5000 שזה אומר שאנחנו מתחברים לפורט מספר 5000 במכונה שמריצה את הקוד. ואז מה שקיבלנו היה דף עם רשימה ריקה, ובמסך של ה- git bash קיבלנו את התגובות של השרת עם קוד 200 שפירושו OK.

חלק 2 (פרישת האפליקציה):

כעת נעלה את ה- docker container שיצרנו ל- GCP Container Registry על ידי תיוג ה- container ואז דחיפתו על ידי שתי הפקודות: docker tag ו- docker push בעזרת git bash, לאחר ביצוע הפקודות האלו, בדקנו את הפרויקט דרך ה- UI של גוגל וקיבלנו את המסך הבא:



כדי ליצור את ה- Managed Instance Group, אנחנו צריכים קודם ליצור VM Template. ביצירת ה- template אנחנו בוחרים בסוג המכונה שתריץ את השרת ובמאפיינים שלה ושל ה- Boot Disk. בנוסף, אנחנו נבחר את ה- image שיצרנו והעלינו ל- GCP Container Registry, רק אחר כך נוכל ליצור את ה- MIG שלנו שיכיל בתוכו לפחות מכונה אחת, ויכיל לכל היותר 10 מכונות באזורים גיאוגרפיים שונים למען אחוז ה- Availability של התוכנית. להלן, צילום מסך ל- UI של ה- Managed Instance Group בפרויקט שלנו עד השלב הזה:



לאחר מכן, יצרנו את מסד הנתונים תחת הלשונית CloudSQL כפי שמתואר בלינק המצורף לחלק הזה.

בקוד של ה- Terraform יצרנו מודולים כדי להפריד בין שתי הסביבות (Dev & Production) והוספנו את קובץ ה- configurations. אחר כך יצרנו מפתח ב- UI של GCP והוספנו את קובץ ה- json לתיקיית הפרויקט, ואז הוספנו את ה- credentials לקוד ה- Terraform כך שימצא את המפתח שיצרנו והוספנו לתיקיית הפרויקט.
להלן צילום מסך לקוד יצירת סביבות העבודה ותוצאות שלו:

```
PS C:\Users\Karee\Documents\Cloud-Exercises\HW2> terraform workspace new Dev
Created and switched to workspace "Dev"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
PS C:\Users\Karee\Documents\Cloud-Exercises\HW2> terraform workspace new Production
Created and switched to workspace "Production"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
PS C:\Users\Karee\Documents\Cloud-Exercises\HW2>
```

להלן עוד צילום מסך של התמודדות עם סביבות העבודה השונות בפרויקט:

```
PS C:\Users\Karee\Documents\Cloud-Exercises\HW2> terraform workspace list
  default
  Dev
* Production

PS C:\Users\Karee\Documents\Cloud-Exercises\HW2> terraform workspace select Dev
Switched to workspace "Dev".
PS C:\Users\Karee\Documents\Cloud-Exercises\HW2> terraform workspace list
  default
* Dev
  Production

PS C:\Users\Karee\Documents\Cloud-Exercises\HW2> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_instance_group_manager.appserver will be created
  + resource "google_compute_instance_group_manager" "appserver" {
      + base_instance_name        = "appserver"
      + fingerprint               = (known after apply)
      + id                        = (known after apply)
      + instance_group            = (known after apply)
      + name                      = "appserver-igm"
      + operation                 = (known after apply)
      + project                   = (known after apply)
Git  TODO  Problems  Terminal  Endpoints  Python Packages  Python Console
```

לאחר סיום הקוד, הרצנו את הפקודה terraform init שתוריד את כל המשאבים של ה- backend שהקוד צריך, ואז הרצנו terraform plan שתציג את השינויים שיתבצעו על הפרויקט. ובסוף הרצנו את הפקודה terraform apply שביצעה את השינויים שלנו. ואז בדקנו ב- UI של GCP שהפרויקט התעדכן עם המשאבים החדשים.

צילומי המסך הבאים התקבלו מהרצת הפקודה terraform apply עם הקוד שכתבנו על פרויקט ריק:

```
PS C:\Users\Karee\Documents\Cloud-Exercises\HW2> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_instance_group_manager.appserver will be created
  + resource "google_compute_instance_group_manager" "appserver" {
      + base_instance_name        = "appserver"
      + fingerprint               = (known after apply)
      + id                        = (known after apply)
      + instance_group            = (known after apply)
      + name                      = "appserver-igm"
      + operation                 = (known after apply)
      + project                   = (known after apply)
      + self_link                 = (known after apply)
      + status                    = (known after apply)
      + target_size               = (known after apply)
      + wait_for_instances        = false
      + wait_for_instances_status = "STABLE"
      + zone                      = "us-central1-a"

      + update_policy {
          + max_surge_fixed         = (known after apply)
          + max_surge_percent       = (known after apply)
          + max_unavailable_fixed   = (known after apply)
          + max_unavailable_percent = (known after apply)
          + minimal_action          = (known after apply)
          + replacement_method      = (known after apply)
```

```
          + max_unavailable_percent = (known after apply)
          + minimal_action          = (known after apply)
          + replacement_method      = (known after apply)
          + type                    = (known after apply)
        }

      + version {
          + instance_template = (known after apply)

          + target_size {
              + fixed = 1
            }
        }
    }

  # google_compute_instance_template.flask-server-template will be created
  + resource "google_compute_instance_template" "flask-server-template" {
      + can_ip_forward       = false
      + description          = "This template is used for creating compute instances to run flask web app."
      + id                   = (known after apply)
      + labels               = {
          + "environment" = "dev"
        }
      + machine_type         = "c2-standard-4"
      + metadata_fingerprint = (known after apply)
      + name                 = "flask-app-template"
      + name_prefix          = (known after apply)
      + project              = (known after apply)
      + region               = (known after apply)
      + self_link            = (known after apply)
```

```
      + project              = (known after apply)
      + region               = (known after apply)
      + self_link            = (known after apply)
      + tags_fingerprint     = (known after apply)

      + confidential_instance_config {
          + enable_confidential_compute = (known after apply)
        }

      + disk {
          + auto_delete  = true
          + boot         = true
          + device_name  = (known after apply)
          + disk_size_gb = (known after apply)
          + disk_type    = (known after apply)
          + interface    = (known after apply)
          + mode         = (known after apply)
          + source_image = "debian-cloud/debian-9"
          + type         = (known after apply)
        }

        }
    }

Plan: 2 to add, 0 to change, 0 to destroy.


Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
PS C:\Users\Karee\Documents\Cloud-Exercises\HW2>
```

אחר כך, ביצענו כמה שינויים על הקובץ app.py, ואז כדי לעדכן את הקוד שבענן, נאלצנו למחוק את ה- image הישנה וליצור אחת חדשה, ואז בדומה למה שעשינו בהתחלה, נדחוף את ה- container ל- GCP Container Registry. רק הפעם במקום למחוק את כל המשאבים דרך ה- UI של גוגל שזה לוקח זמן ארוך ואז לבנות את ה- template ואת ה- MIG מהתחלה, את השינויים ייעשו דרך ה- Terraform על ידי הפקודה terraform plan שלוקחת ממש שניות לבצע את כל השינויים על הפרויקט, וזה מה שקיבלנו:

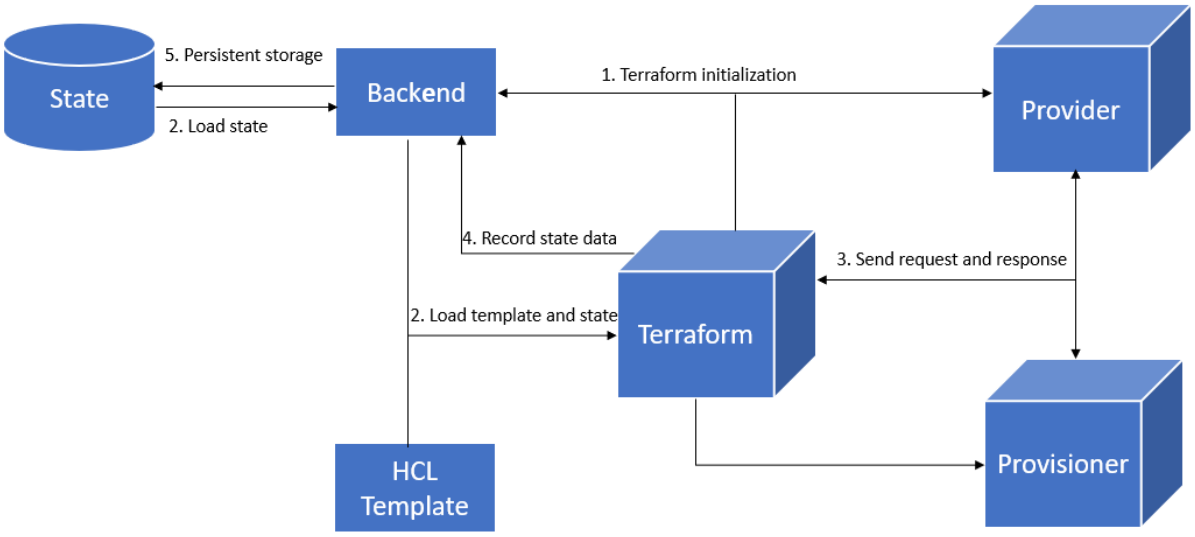
```
  # google_compute_instance_group_manager.appserver must be replaced
-/+ resource "google_compute_instance_group_manager" "appserver" {
      ~ fingerprint               = "xddpYhz9p-Q=" -> (known after apply)
      ~ id                        = "projects/peppy-oven-344513/zones/us-central1-a/instanceGroupManagers/appserver-igm" -> (known after apply)
      ~ instance_group            = "https://www.googleapis.com/compute/v1/projects/peppy-oven-344513/zones/us-central1-a/instanceGroups/appserver-igm" -> (known after apply)
        name                      = "appserver-igm"
      + operation                 = (known after apply)
      ~ project                   = "peppy-oven-344513" -> (known after apply)
      ~ self_link                 = "https://www.googleapis.com/compute/v1/projects/peppy-oven-344513/zones/us-central1-a/instanceGroupManagers/appserver-igm" -> (known after apply)
      ~ status                    = [
          - {
              - is_stable      = true
              - stateful       = [
                  - {
                      - has_stateful_config  = false
                      - per_instance_configs = [
                          - {
                              - all_effective = true
                            },
                        ]
                    },
                ]
              - version_target = [
                  - {
                      - is_reached = true
                    },
          ~ instance_template = "https://www.googleapis.com/compute/v1/projects/peppy-oven-344513/global/instanceTemplates/flask-app-template" -> "projects/peppy-oven-344513/global/ins
tanceTemplates/flask-app-template"
        }
```

בסוף העבודה, וכדי לא לאבד את ה- credits שלנו ב- GCP ביצענו את הפקודה terraform destroy שתמחק את כל המשאבים שיצרנו כולל גם את ה- MIG ואת ה- template כפי שמתואר בצילום הבא:

```
google_compute_instance_group_manager.appserver: Destroying... [id=projects/peppy-oven-344513/zones/us-central1-a/instanceGroupManagers/appserver-igm]
google_compute_instance_group_manager.appserver: Still destroying... [id=projects/peppy-oven-344513/zones/us-cen...-a/instanceGroupManagers/appserver-igm, 10s elapsed]
google_compute_instance_group_manager.appserver: Destruction complete after 11s
google_compute_instance_template.flask-server-template: Destroying... [id=projects/peppy-oven-344513/global/instanceTemplates/flask-app-template]
google_compute_instance_template.flask-server-template: Still destroying... [id=projects/peppy-oven-344513/global/instanceTemplates/flask-app-template, 10s elapsed]
google_compute_instance_template.flask-server-template: Destruction complete after 12s

Destroy complete! Resources: 2 destroyed.
PS C:\Users\Karee\Documents\Cloud-Exercises\HW2>
```

דיאגרמת ארכיטקטורה שמסבירה את האפליקציה:



כפי שאנחנו רואים, השלב הראשון בהרצת ה- Terraform זה לבצע terraform init, שזה ניגש ל- backend ומוריד את הכלים ש- Terraform צריך כדי להריץ את הקוד תלוי בקובץ ה- configurations.

כפי שאנחנו רואים בדיאגרמה, Terraform משתמש ב- state files, את הקבצים האלה אפשר למצוא למשל בתיקיית העבודה שלנו כשהרצנו את הפקודה terraform init. הקבצים האלה נוצרים בפעם הראשונה אנחנו מריצים את הפקודה terraform init, ואז בכל פעם אנחנו מריצים את הפקודה של terraform plan, הקבצים מתעדכנים עם ה- state החדש של הפרויקט.

באותו שלב, Terraform גם מקבל את ה- HCL (הקוד שכתבנו בקבצי tf.) ומבצע את השינויים על הפרויקט. לאחר מכן, Terraform מיצר קובץ state עדכני שישלח ל- Backend שהוא בתורו מעדכן את ה- state files במכונה שלנו שישתמש בהן בהרצה הבאה של Terraform.