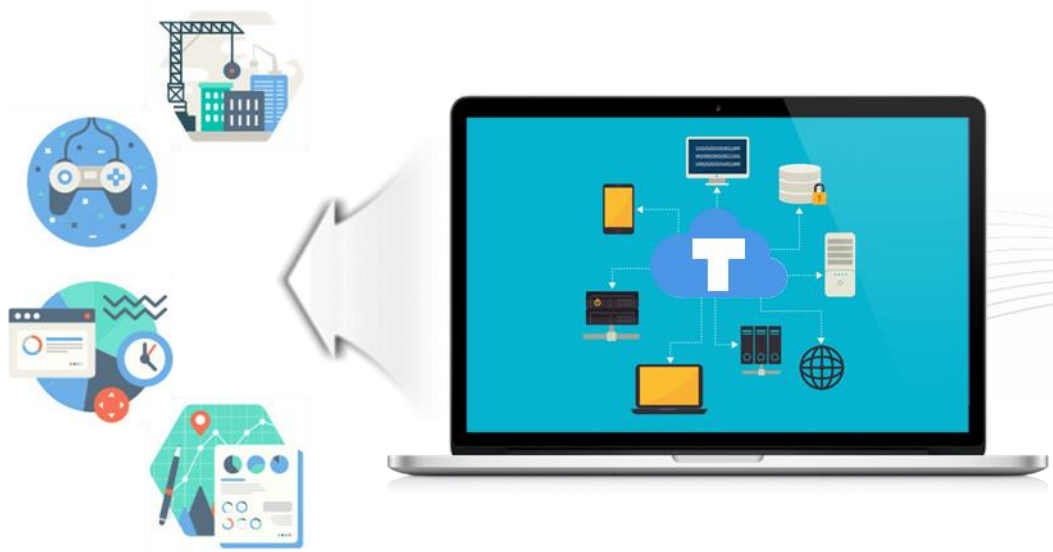


(실습) 7차수

Ⅰ GKE

Ⅱ GCP Cloud Build

Ⅲ CLI 활용 클러스터 생성 및 컨테이너 배포

Ⅳ IaC(Infrastructure as Code)

I GKE - 클러스터 생성

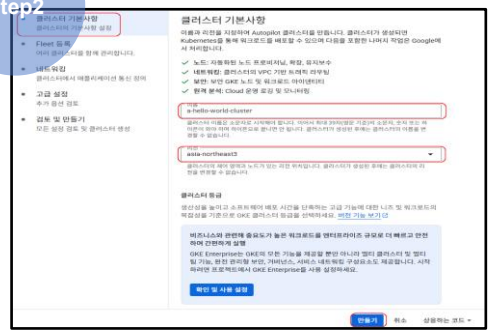
[GKE Autopilot 모드로 클러스터 만들기]

- Autopilot 모드에서 Google은 확장, 보안, 기타 사전 구성된 설정을 포함하여 클러스터 구성을 관리
- Autopilot 모드의 클러스터는 대부분의 프로덕션 워크로드를 실행하고 Kubernetes 매니페스트에 따라 컴퓨팅 리소스를 프로비저닝하도록 최적화 되어 있음

Step1

[그림 1] 클러스터 만들기

Step2



[그림 2] 클러스터 기본사항 설정

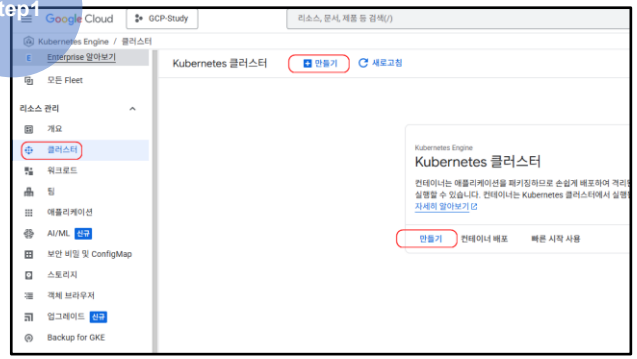
Step2

[그림 3] 클러스터 생성 결과

[GKE Standard 모드로 클러스터 만들기]

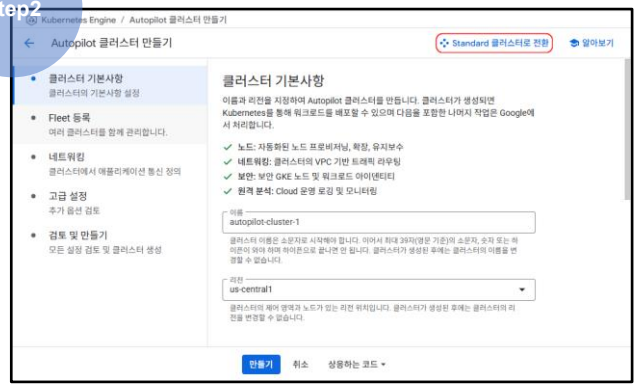
- Autopilot 모드에 비해 추가 기본설정(node 등) 적용 가능

Step1



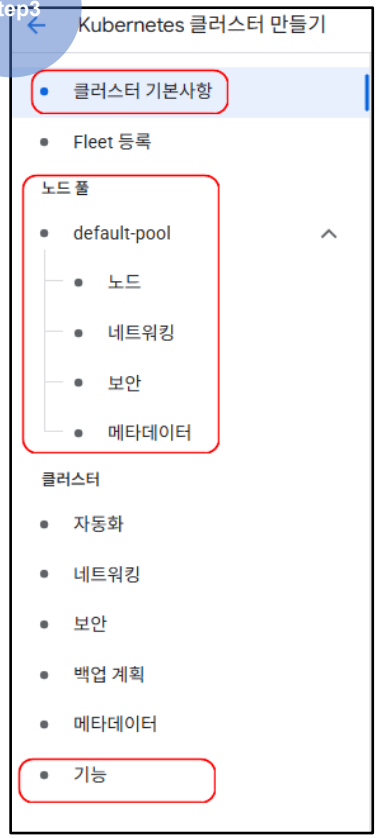
[그림 4] 클러스터 만들기

Step2



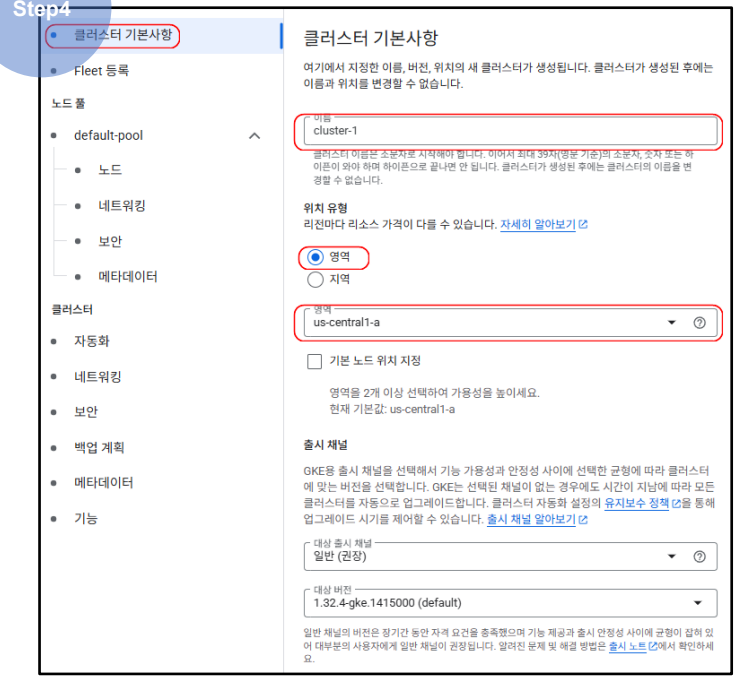
[그림 5] standard 모드 전환

Step3



[그림 6] standard 메뉴

Step4

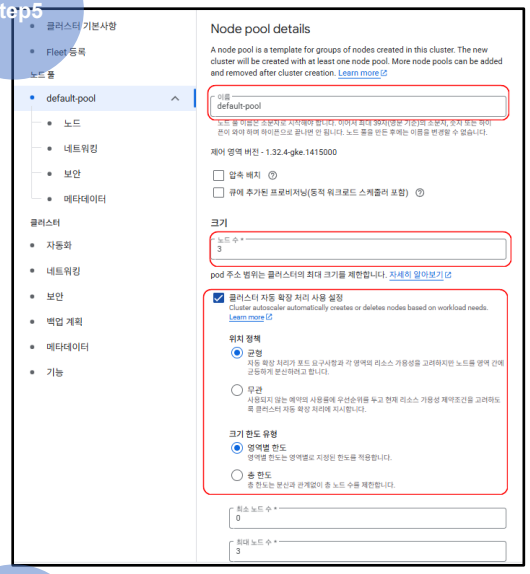


[그림 7] 클러스터 기본사항 설정

[GKE standard 모드로 클러스터 만들기]

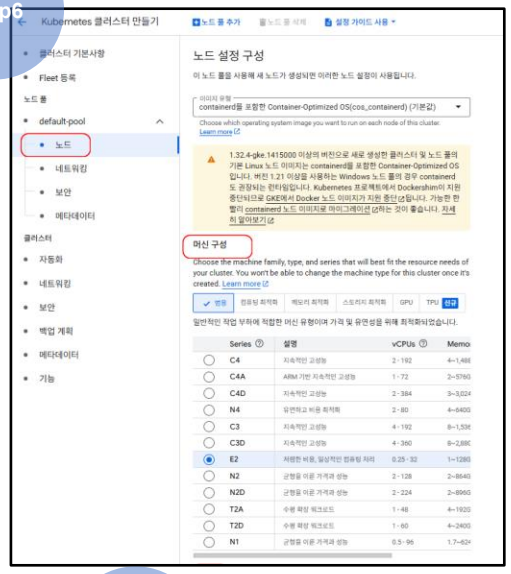
- Autopilot 모드에 비해 추가 기본설정 적용 가능

Step5



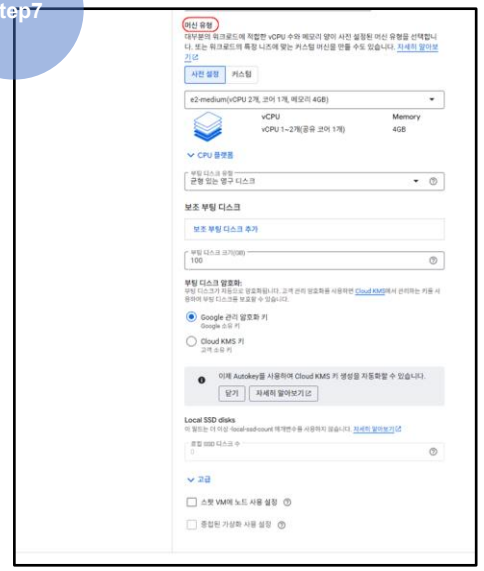
[그림 8] default pool 설정

Step6



[그림 9] 노드 설정

Step7

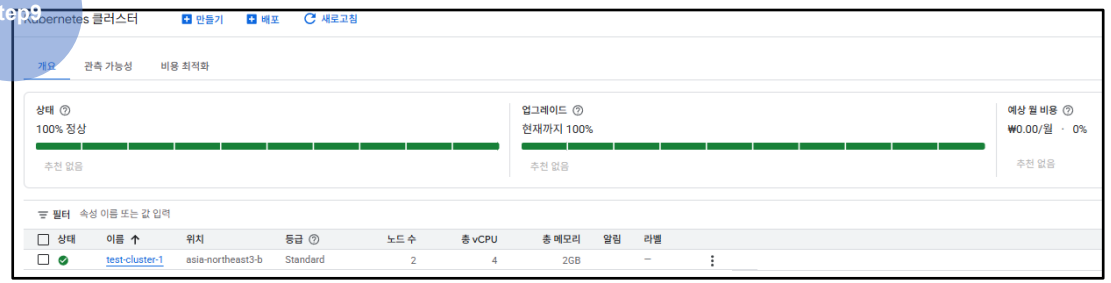


[그림 10] 머신유형 설정

Step8

[그림 11] 클러스터 생성

Step9

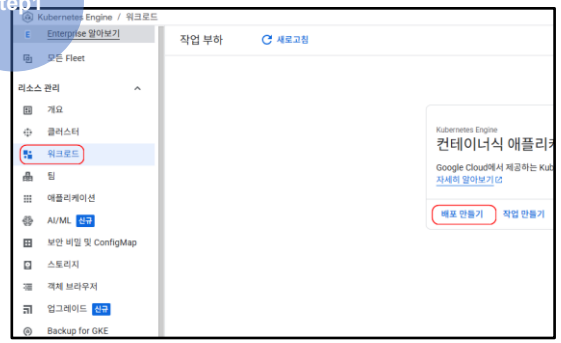


[그림 12] 클러스터 생성 확인

[클러스터에 샘플 앱 배포]

- Google에서 제공되고 Artifact Registry에 컨테이너로 저장되는 샘플 'hello world' 웹앱을 배포

Step1

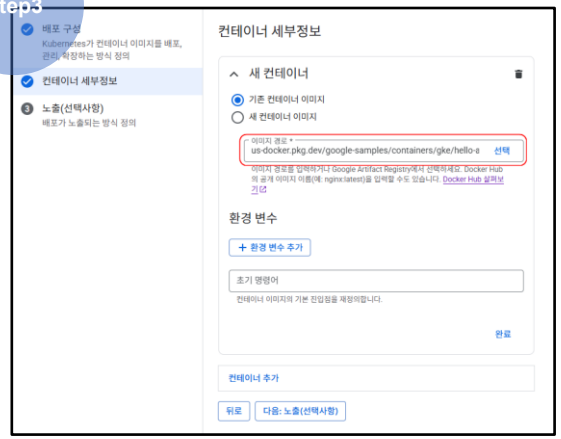


[그림 13] 워크로드 배포 만들기

Step2

[그림 14] 배포구성 설정

Step3



[그림 15] 컨테이너 세부정보 설정

Step4

[그림 16] 노출 설정

[클러스터에 샘플 앱 배포]

- 샘플 'hello world' 웹앱 yaml 설정 내용

```yaml
---
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  name: "hello-world-app"
  namespace: "default"
  labels:
    app: "hello-world-app"
spec:
  replicas: 3
  selector:
    matchLabels:
      app: "hello-world-app"
  template:
    metadata:
      labels:
        app: "hello-world-app"
    spec:
      containers:
      - name: "hello-app-1"
        image: "us-docker.pkg.dev/google-samples/containers/gke/hello-app:1.0"
```

[그림 17] hello-world-app-deployment.yml

```yaml
---
apiVersion: "autoscaling/v2"
kind: "HorizontalPodAutoscaler"
metadata:
  name: "hellow-world-app-hpa-dubi"
  namespace: "default"
  labels:
    app: "hello-world-app"
spec:
  scaleTargetRef:
    kind: "Deployment"
    name: "hello-world-app"
    apiVersion: "apps/v1"
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: "Resource"
    resource:
      name: "cpu"
      target:
        type: "Utilization"
        averageUtilization: 80
```

[그림 18] hello-world-app-hpa.yml

```yaml
---
apiVersion: "v1"
kind: "Service"
metadata:
  name: "hello-world-app-service"
  namespace: "default"
  labels:
    app: "hello-world-app"
spec:
  ports:
  - protocol: "TCP"
    port: 80
    targetPort: 8080
  selector:
    app: "hello-world-app"
  type: "LoadBalancer"
  loadBalancerIP: ""
```

[그림 19] hello-world-app-svc-lb.yml

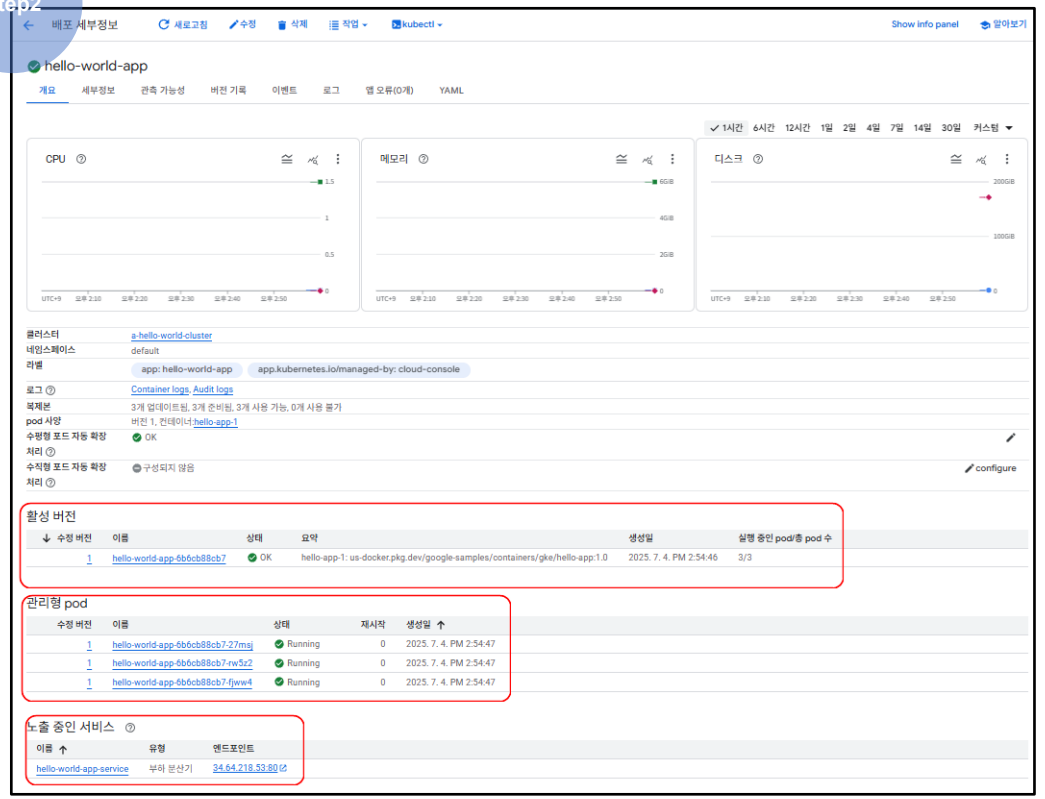
[클러스터에 샘플 앱 배포]

- 샘플 'hello world' 웹앱을 배포 완료 상태 확인

Step1

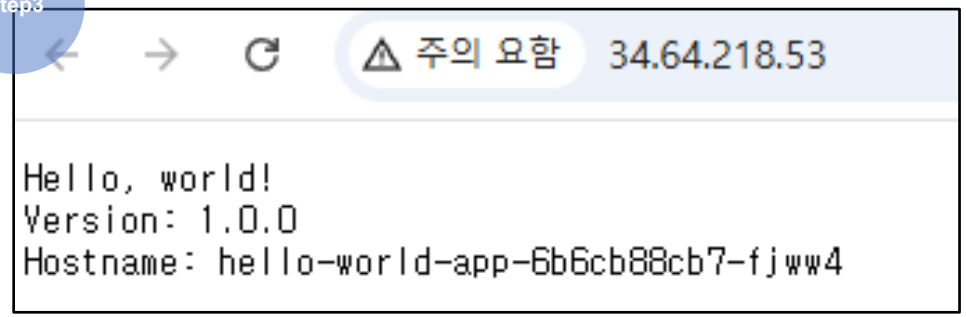
[그림 20] 배포 완료 상태

Step2



[그림 21] 배포 완료 상태(상세)

Step3

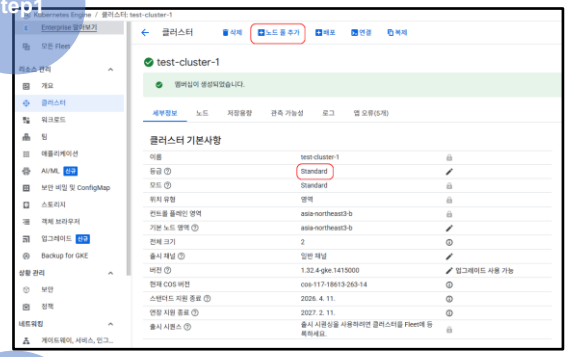


[그림 22] 배포 완료 상태(상세)

[노드 풀(Node Pool) 추가]

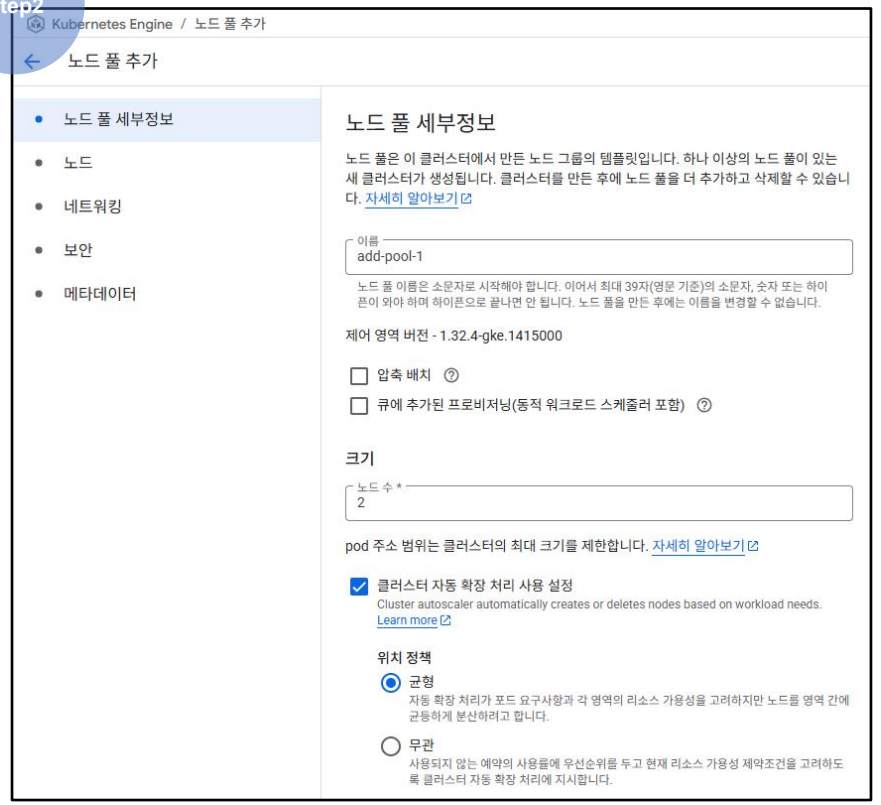
- 클러스터 > 대상 클러스터 이름 선택 > 노드 풀 추가
- 노드 풀(리스트 확인)

Step1



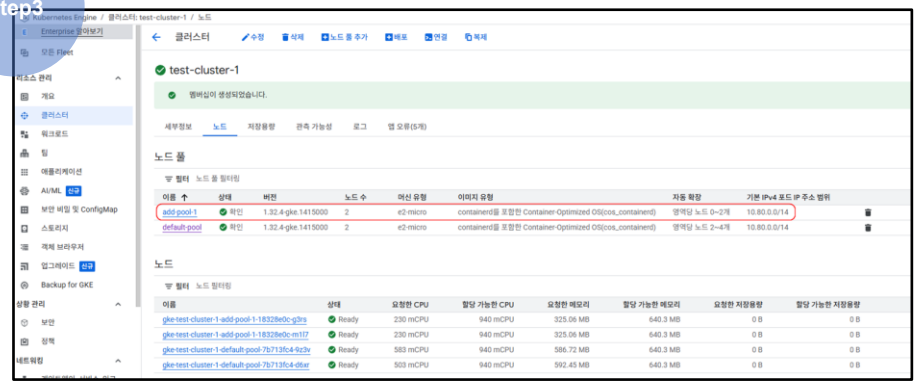
[그림 23] 노드풀 추가

Step2



[그림 24] 노드 풀 세부정보 설정

Step3

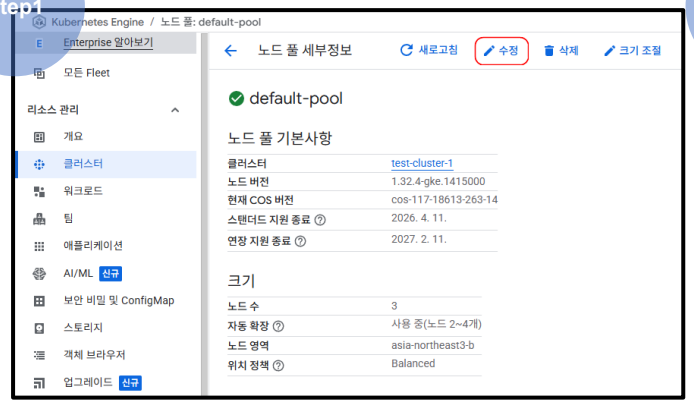


[그림 25] 노드 풀 추가 확인

[노드 풀(Node Pool) 수정]

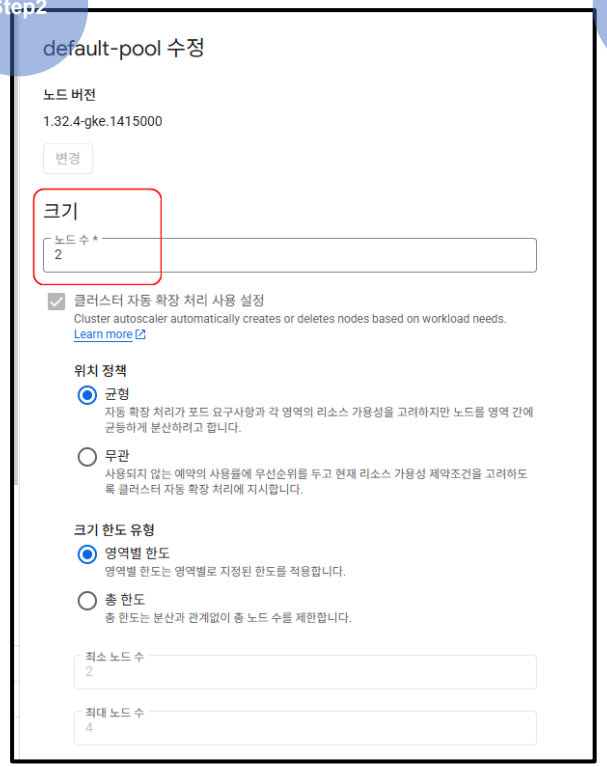
- 클러스터 > 대상 클러스터 이름 선택 > 노드(탭) > 삭제 대상 노트 풀(삭제)
- 노드 풀(리스트 확인)

Step1



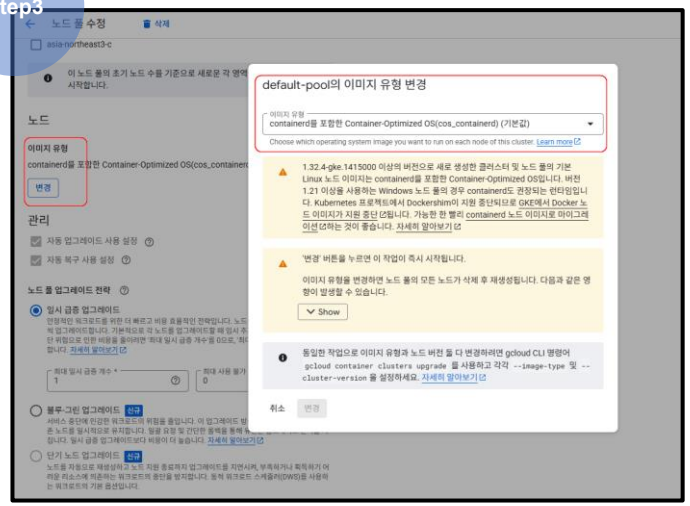
[그림 26] 노드 풀 수정

Step2



[그림 27] 노드 풀 수정

Step3

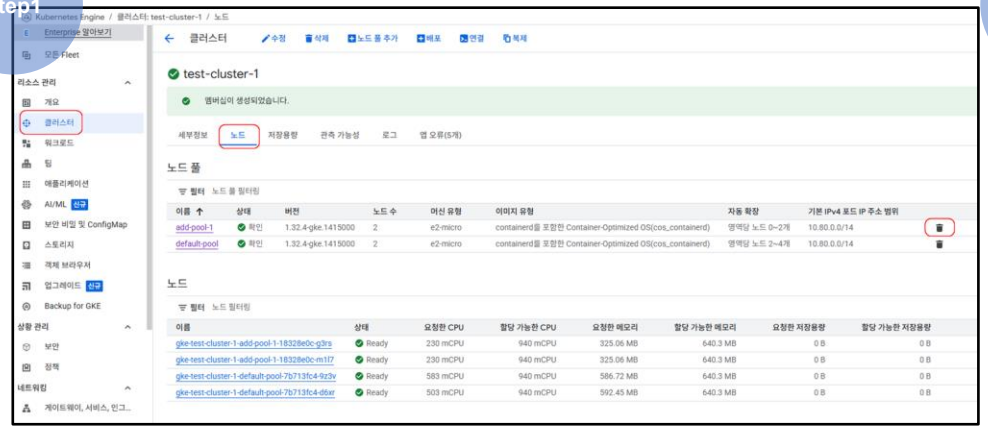


[그림 28] 노드 풀 수정

[노드 풀(Node Pool) 삭제]

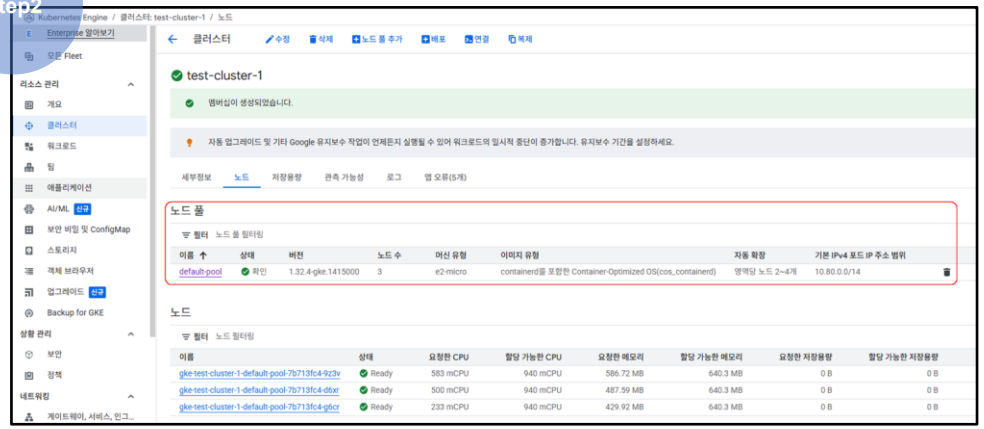
- 클러스터 > 대상 클러스터 이름 선택 > 노드(탭) > 삭제 대상 노트 풀(삭제)
- 노드 풀(리스트 확인)

Step1



[그림 29] 노드 풀 선택 및 삭제

Step2



[그림 30] 노드 풀 삭제 결과 확인

[AutoScaling-Pod]

- **Horizontal Pod Autoscaler(HPA)**

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  replicas: 1
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: k8s.gcr.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

[그림 31] deployment-php-apache.yml

```
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
 name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
   name: php-apache
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

[그림 32] hpa-php-apache.yml

[AutoScaling-Pod]

- **Vertical Pod Autoscaler(VPA)**

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vpa-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vpa-demo
  template:
    metadata:
      labels:
        app: vpa-demo
    spec:
      containers:
      - name: stress-container
        image: vish/stress
        resources:
          requests:
            cpu: "50m"
            memory: "50Mi"
          limits:
            cpu: "200m"
            memory: "200Mi"
        args:
        - -cpus
        - "2"
```

[그림 33] vpa-demo-deployment.yaml

```yaml
---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-demo
  namespace: default
  clusterName: gcp-edu-gke-standard-1
spec:
  targetRef:
    kind: Deployment
    name: vpa-demo
    apiVersion: apps/v1
  updatePolicy:
    updateMode: Auto
  resourcePolicy:
    containerPolicies:
      - containerName: '*'
        mode: Auto
        controlledResources:
          - cpu
          - memory
        minAllowed: {}
        maxAllowed: {}
```

[그림 34] vpa-deployment.yaml

```yaml
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name:
verticalpodautoscalers.autoscaling.k8s.io
spec:
  group: autoscaling.k8s.io
  names:
    kind: VerticalPodAutoscaler
    listKind: VerticalPodAutoscalerList
    plural: verticalpodautoscalers
    singular: verticalpodautoscaler
  scope: Namespaced
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
```

[그림 35] vpa-crd.yaml

[GCP Cloud Build 실습 목표]

- Cloud Build를 사용한 자동 빌드 & 배포
- Docker 이미지 자동 생성 및 GCR에 푸시
- GKE에 자동 배포 연결 (옵션)

[사전 준비]

- GCP 프로젝트 생성 및 Billing 활성화
- Cloud Build API 활성화
- GCR (Container Registry) 또는 Artifact Registry 활성화
- GitHub 또는 Cloud Source Repositories 연동

[예제 애플리케이션 구성]

- node.js 기반 간단한 웹 앱
- app.js 생성

```
const express = require('express');
const app = express();
const port = 8080;

app.get('/', (req, res) => res.send('Hello Cloud Build!'));

app.listen(port, () => console.log(`App listening on port ${port}`));
```

[그림 36] 노드 풀 삭제 결과 확인

[Dockerfile 생성]

```
FROM node:18

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

CMD ["node", "app.js"]
```

[그림 37] 노드 풀 삭제 결과 확인

[cloudbuild.yaml 작성]

```
steps:
  - name: 'gcr.io/cloud-builders/docker’
    args: ['build', '-t', 'gcr.io/$PROJECT_ID/demo-app', '.’]
  - name: 'gcr.io/cloud-builders/docker’
    args: ['push', 'gcr.io/$PROJECT_ID/demo-app’]

images:
  - 'gcr.io/$PROJECT_ID/demo-app’
```

[그림 38] 노드 풀 삭제 결과 확인

[Cloud Build 트리거 생성]

- Cloud Console > Cloud Build > 트리거 생성
- GitHub 또는 Cloud Source Repo 연결
- Branch: main 또는 특정 브랜치
- 빌드 구성: cloudbuild.yaml 선택

[Cloud shell에 Github 연동]

- ssh key 생성

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

[그림 39] ssh key 생성

```
[redacted]@cloudshell:~/.ssh$ ssh-keygen -t rsa -b 4096 -C "[redacted]"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jshan97/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jshan97/.ssh/id_rsa
Your public key has been saved in /home/jshan97/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Sx8HceLwrf2gUP1bZqL0OfuYLeJyPFawtsBtpuSjRCU jshan97@gmail.com
The key's randomart image is:
+---[RSA 4096]----+
|        . o .    |
|         + *     |
|        E * o    |
|         + +..   |
|        S.o.=oo +|
|       o ++=*=.B |
|        oooB.o*  |
|       .  = B .* |
|        .. *.o+oo|
+----[SHA256]-----+
```

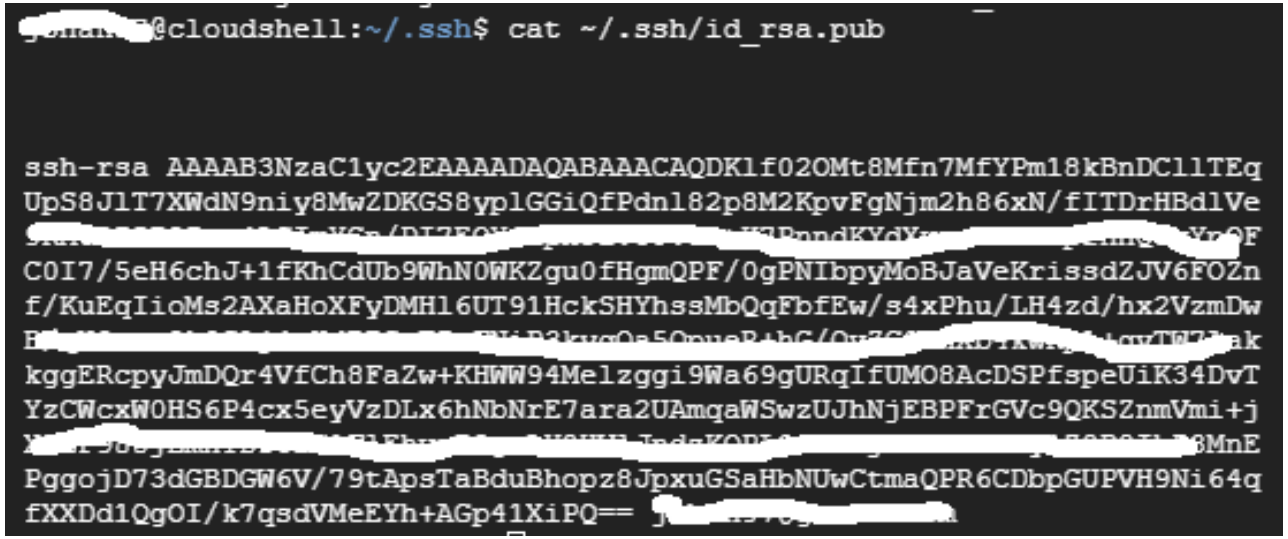
[그림 40] ssh key 생성 완료

[Cloud shell에 Github 연동]

- ssh key 생성 공개키 확인

```
cat ~/.ssh/your_key_name.pub
```

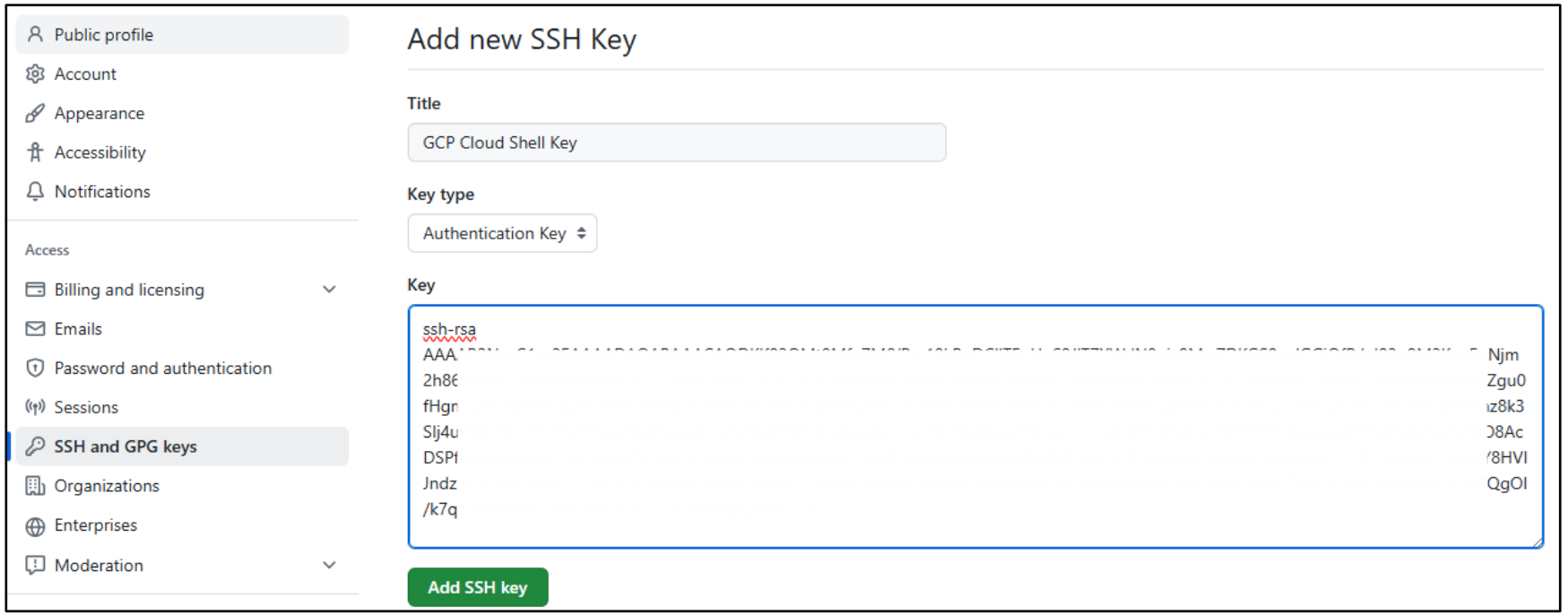
[그림 41] ssh 공개키 확인



[그림 42] ssh 공개키 확인

[Cloud shell에 Github 연동]

- Github에 ssh-key 설정



[그림 43] github ssh 공개키 등록

[Cloud shell에 Github 연동]

- Cloud shell ssh config 설정(~/.ssh/config)

```
Host github.com
  Hostname github.com
  IdentityFile ~/.ssh/your_key_name
```

```
Host github.com
  Hostname github.com
  IdentityFile ~/.ssh/id_rsa
```

[그림 44] ssh 사용 config 파일 설정

- Test the Connection

```
ssh -T git@github.com
```

```
[redacted]@cloudshell:~/.ssh$ ssh -T git@github.com
Hi [redacted]! You've successfully authenticated, but GitHub does not provide shell access.
```

[그림 45] ssh config 설정 확인

[Cloud shell에 git project repo 설정-1]

- git 초기화 / 사용자 이름 / 이메일주소 설정

```
git init
git config --global user.name "사용자이름"
git config --global user.email "이메일주소"
```

- 설정 확인

```
git config --list
```

```
[redacted]@cloudshell:~/gcp-cloud-build-sample-1$ git config --list
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
credential.https://source.developers.google.com.helper=gcloud.sh
user.email=[redacted]
user.name=[redacted]
core.repositoryformatversion=0
core.filemode=true
core.bare=false
pull.rebase=false
remote.origin.url=git@github.com:gcp-edu/gcp-cloud-bulid-sample.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
```

[그림 46] git 설정 확인

[Cloud shell에 git project repo 설정-2]

- git remote 설정

```
git remote add origin https://github.com/gcp-edu/gcp-cloud-bulid-sample.git
```

- 설정 확인

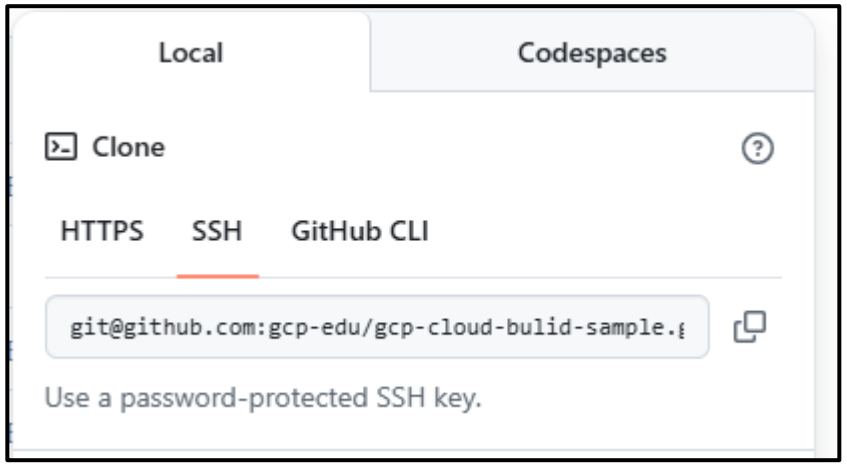
```
git remote -v
```

- Local branch 생성

```
git branch main
```

- 생성된 repo pull

```
git pull origin main
```

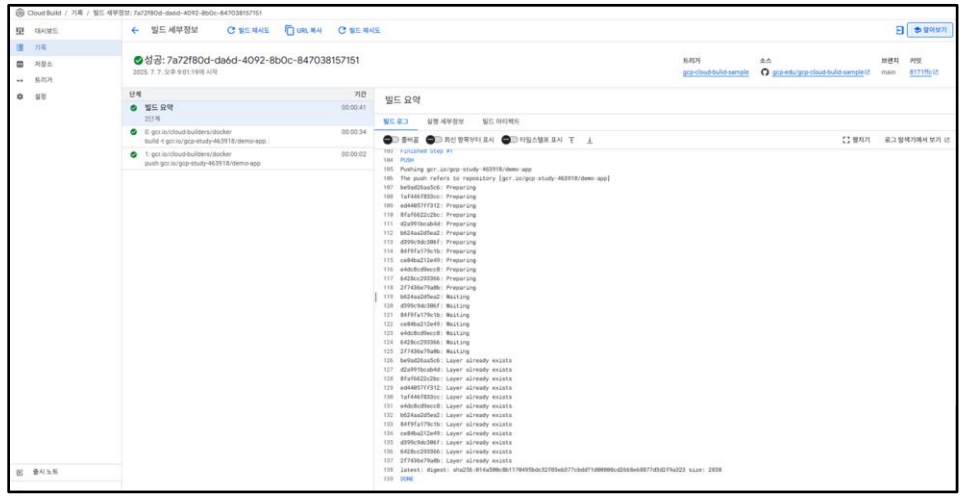


[그림 47] git repo ssh url 확인

[커밋 & 자동 실행 테스트]

- Cloud Build에서 자동으로 빌드 시작
- 이미지가 GCR로 푸시됨

```
git add .
git commit -m "Deploy with Cloud Build“
git push origin main
```



[그림 48] Cloud build 로그 확인

[그림 49] Artifact Registry 생성 확인

GKE에 Container 응용 배포하기

- GKE 클러스터 생성
- Docker 이미지 만들기
- Kubernetes YAML 작성
- GKE에 배포

사전 준비

- GCP 계정 필요
- 프로젝트 생성 (Billing enabled)
- Cloud Shell 또는 로컬 gcloud/kubectl 설치

1. GKE 클러스터 생성

- num-nodes=2
- region=us-central1
- machine-type=e2-medium
- disk-type=pd-standard
- disk-size=100

```
gcloud container clusters create demo-cluster \
  --num-nodes=2 \
  --region=us-central1 \
  --machine-type=e2-medium \
  --disk-type=pd-standard \
  --disk-size=100
```

```
$ gcloud container clusters create demo-cluster \
  --num-nodes=2 \
  --region=us-central1 \
  --machine-type=e2-medium \
  --disk-type=pd-standard \
  --disk-size=50
Note: The Kubelet readonly port (10255) is now deprecated. Please upd
ate your workloads to use the recommended alternatives. See https://c
loud.google.com/kubernetes-engine/docs/how-to/disable-kubelet-readonl
y-port for ways to check usage and for migration instructions.
Note: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate
at most 1008 node(s).
Creating cluster demo-cluster in us-central1... Cluster is being heal
th-checked (Kubernetes Control Plane is healthy)...done.


Created [https://container.googleapis.com/v1/projects/gcp-study-46391
8/zones/us-central1/clusters/demo-cluster].
To inspect the contents of your cluster, go to: https://console.cloud
.google.com/kubernetes/workload_/gcloud/us-central1/demo-cluster?proj
ect=gcp-study-463918
kubeconfig entry generated for demo-cluster.
NAME: demo-cluster
LOCATION: us-central1
MASTER_VERSION: 1.32.4-gke.1415000
MASTER_IP: 34.67.148.58
MACHINE_TYPE: e2-medium
NODE_VERSION: 1.32.4-gke.1415000
NUM_NODES: 6
STATUS: RUNNING
```

[그림 50] demo-cluster 생성 command

2. app.js 파일 만들기

```
const express = require('express');
const app = express();
const port = 8080;

app.get('/', (req, res) => {
  res.send('Hello from GKE!');
});

app.listen(port, () => {
  console.log(`App is running on port ${port}`);
});
```

2. package.json 만들기

```
{
  "name": "demo-app",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

3. Dockerfile 만들기 & Push

- Dokerfile 작성

```
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD ["node", "app.js"]
```

- 생성 파일 구조

```
project-root/
├── app.js
├── package.json
├── Dockerfile
```

- 빌드

  #> docker build -t gcr.io/[PROJECT_ID]/demo-app:v1 .

- GCR Push

  #> docker push gcr.io/[PROJECT_ID]/demo-app:v1

```
$ docker build -t gcr.io/gcp-study-463918/demo-app:v1 .
[+] Building 2.0s (10/10) FINISHED

                    docker:default
 => [internal] load build definition from Dockerfile

                                0.0s
 => => transferring dockerfile: 134B

                                0.0s
 => [internal] load metadata for docker.io/library/node:18

                                0.6s
 => [internal] load .dockerignore

                                0.0s
 => => transferring context: 2B

                                0.0s
 => [1/5] FROM docker.io/library/node:18@sha256:c6ae79e38498325db6719
3d391e6ec1d224d96c693a8a4d943498556716d3783
```

[그림 51] docker build command

```
$ docker push gcr.io/gcp-study-463918/demo-app:v1

The push refers to repository [gcr.io/gcp-study-463918/demo-app]
002762b6697c: Pushed
411422e32cb3: Pushed
4aab4bf90b47: Pushed
42d9c1bab012: Pushed
d2a991bcab4d: Layer already exists
b624aa2d5ea2: Layer already exists
d399c9dc306f: Layer already exists
84f9fa179c1b: Layer already exists
ce84ba212e49: Layer already exists
e4dc8cd9ecc8: Layer already exists
6428cc293366: Layer already exists
2f7436e79a0b: Layer already exists
v1: digest: sha256:4c2f37044c6d151a79688419e0267d3e54bec86b2016fec23d
f6387754d486e7 size: 2831
```

[그림 52] gcr push command

4. Kubernetes Deployment.yaml

- Deployment.yaml 작성

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: demo-app
  template:
    metadata:
      labels:
        app: demo-app
    spec:
      containers:
      - name: demo-app
        image: gcr.io/[PROJECT_ID]/demo-app:v1
        ports:
        - containerPort: 8080
```

4. Service.yaml (LoadBalancer)

- Service.yaml 작성

```
apiVersion: v1
kind: Service
metadata:
  name: demo-service
spec:
  type: LoadBalancer
  selector:
    app: demo-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

5. 적용 & 결과 확인

- kubectl apply -f Deployment.yaml

```
deployment.apps/demo-app created
```

- Kubectl apply –f Service.yaml

```
service/demo-service created
```

- kubectl get pods

```
NAME                        READY   STATUS             RESTARTS      AGE
demo-app-764798b7d8-422vk   0/1     Error              4 (44s ago)   116s
demo-app-764798b7d8-zr9dp   0/1     CrashLoopBackOff   3 (32s ago)   116s
```

- kubectl get svc

```
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)          AGE
demo-service   LoadBalancer   34.118.230.99   34.61.51.248   80:31572/TCP     2m25s
kubernetes     ClusterIP      34.118.224.1    <none>         443/TCP          32m
```

6. 환경 삭제

- gcloud container clusters delete demo-cluster --region us-central1

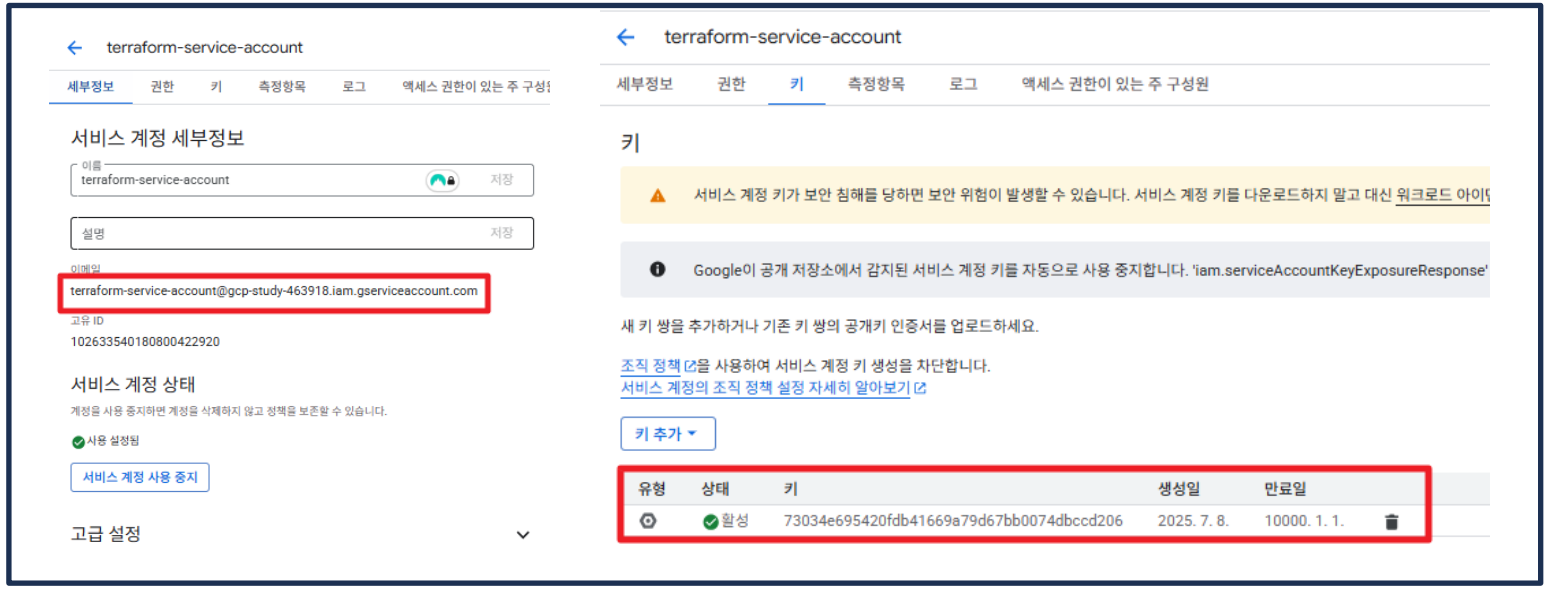
```
deployment.apps "demo-app" deleted
```

- kubectl delete deployments demo-app

```
$ gcloud container clusters delete demo-cluster --region us-central1
The following clusters will be deleted.
 - [demo-cluster] in [us-central1]

Do you want to continue (Y/n)?  y

Deleting cluster demo-cluster...done.
```

[Infrastructure as Code(IaC) 실습 목표]

- Terraform을 사용하여 VM 인스턴스를 배포
- IaC Tool : Terraform
- GCP Project 인증 정보 생성
  1. API 및 서비스 → 사용자 인증 정보 → 사용자 인증 정보 만들기 → 서비스 계정 → 역할 : 소유자
  2. 생성한 서비스 계정 클릭 → 키 → 키 추가 → 새 키 만들기 → 키 유형 : JSON → 인증키를 Terraform 폴더로 이동



[그림 53] terraform 서비스 계정 생성

Step 1. 인프라구조 정의(IaC Code 작성)

- main.tf 작성

```
# 1. Specify the Google Cloud Provider
# This block tells Terraform which cloud provider to use and how to
authenticate.
# Ensure your gcloud CLI is authenticated and configured for the
correct project.
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "~> 5.0" # Use a compatible version
    }
  }
}
```

[그림 54] main.tf (서비스 제공자 정의)

```
# 2. Configure the Google Cloud Project and Region/Zone
# You can define these as variables or hardcode them.
# Using variables makes your code more reusable.
variable "project_id" {
  description = "The Google Cloud Project ID."
  type        = string
  default     = "YOUR_PROJECT_ID" # <<< REMEMBER TO CHANGE THIS
}

variable "region" {
  description = "The Google Cloud region to deploy resources in."
  type        = string
  default     = "us-central1"
}

variable "zone" {
  description = "The Google Cloud zone to deploy the VM in."
  type        = string
  default     =  "us-central1-c"
}

provider "google" {
  project = var.project_id
  region  = var.region
  credentials = "${file("???.json")}"
}
```

[그림 55] main.tf (인프라 구성 위치 및 대상 프로젝트 정보 설정)

Step 1. 인프라구조 정의(IaC Code 작성)

- main.tf 작성

```
# 3. Define the Compute Engine VM Instance
resource "google_compute_instance" "example_vm" {
  name         = "my-terraform-vm"
  machine_type = "e2-medium" # A cost-effective machine type
  zone         = var.zone

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11" # Specify a public image
    }
  }

  network_interface {
    network = "default" # Use the default VPC network
    access_config {
      # This block creates an ephemeral external IP address
    }
  }
```

[그림 56] main.tf (VM 인스턴스 및 네트워크 정의)

```
  # Optional: Add a startup script to install Nginx
  metadata_startup_script = <<-EOF
    #!/bin/bash
    sudo apt-get update
    sudo apt-get install -y nginx
    echo "<h1>Hello from Terraform!</h1>" | sudo tee /var/www/html/index.nginx-debian.html
    sudo systemctl start nginx
    sudo systemctl enable nginx
  EOF

  tags = ["http-server"] # Apply tags for firewall rules
}
```

[그림 57] main.tf (install Nginx 정의)

Step 1. 인프라구조 정의(IaC Code 작성)

- main.tf 작성

```
# 4. Define a Firewall Rule to Allow HTTP Traffic
# This is crucial if you want to access the web server on the VM.
resource "google_compute_firewall" "allow_http" {
  name    = "allow-http-from-terraform"
  network = "default"

  allow {
    protocol = "tcp"
    ports    = ["80"]
  }

  source_ranges = ["0.0.0.0/0"] # Allow from anywhere (for
demonstration)
  target_tags   = ["http-server"] # Apply to VMs with this tag
}
```

[그림 58] main.tf (방화벽 규칙 설정)

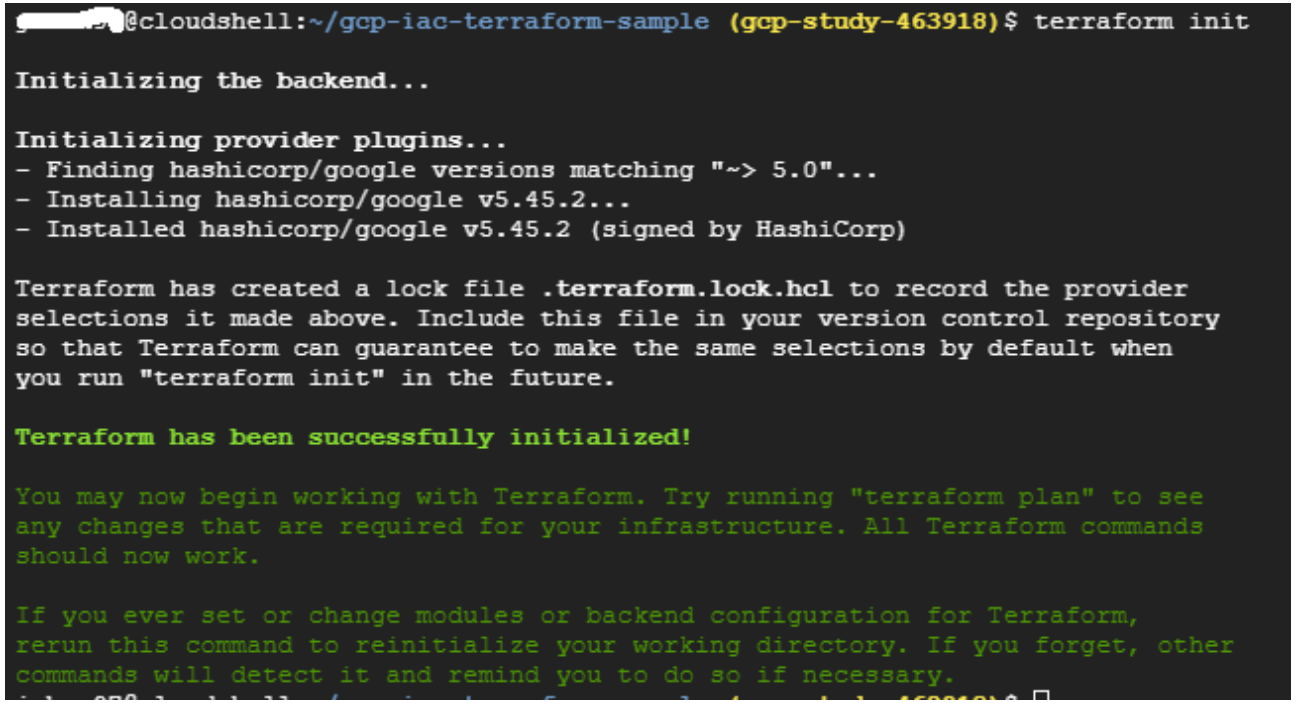
```
# 5. Output the VM's External IP Address
# This makes it easy to find the VM after deployment.
output "vm_external_ip" {
  value       =
google_compute_instance.example_vm.network_interface[0].access_config[0].nat_ip
  description = "The external IP address of the deployed VM."
}
```

[그림 59] main.tf (VM의 External IP 정의)

Step 2. 작업 디렉토리 초기화(terraform)

- Terraform 작업 디렉토리 초기화 명령

```
terraform init
```



[그림 60] terraform 초기화

Step 3. main.tf 적용 예정 내역 미리보기

terraform plan

```
[redacted]@cloudshell:~/gcp-iac-terraform-sample (gcp-study-463918)$ 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_firewall.allow_http will be created
  + resource "google_compute_firewall" "allow_http" {
      + creation_timestamp = (known after apply)
      + destination_ranges = (known after apply)
      + direction          = (known after apply)
      + enable_logging     = (known after apply)
      + id                 = (known after apply)
      + name               = "allow-http-from-terraform"
      + network            = "default"
      + priority           = 1000
      + project            = "gcp-study-463918"
      + self_link          = (known after apply)
      + source_ranges      = [
          + "0.0.0.0/0",
        ]
      + target_tags        = [
          + "http-server",
        ]

      + allow {
          + ports    = [
              + "80",
            ]
          + protocol = "tcp"
        }
    }

  # google_compute_instance.example_vm will be created
  + resource "google_compute_instance" "example_vm" {
      + can_ip_forward          = false
      + cpu_platform            = (known after apply)
      + current_status          = (known after apply)
      + deletion_protection     = false
      + effective_labels        = (known after apply)
      + guest_accelerator       = (known after apply)
      + id                      = (known after apply)
      + instance_id             = (known after apply)
      + label_fingerprint       = (known after apply)
      + machine_type            = "e2-medium"
      + metadata_fingerprint    = (known after apply)
      + metadata_startup_script = <<-EOT
            #!/bin/bash
            sudo apt-get update
            sudo apt-get install -y nginx
            echo "<h1>Hello from Terraform!</h1>" | sudo tee /var/www/html/index.nginx-debian.html
            sudo systemctl start nginx
            sudo systemctl enable nginx
```

[그림 61] plan 실행결과-1

```
            sudo systemctl enable nginx
        EOT
      + min_cpu_platform        = (known after apply)
      + name                    = "my-terraform-vm"
      + project                 = "gcp-study-463918"
      + self_link               = (known after apply)
      + tags                    = [
          + "http-server",
        ]
      + tags_fingerprint        = (known after apply)
      + terraform_labels        = (known after apply)
      + zone                    = "asia-northeast3-b"

      + boot_disk {
          + auto_delete                = true
          + device_name                = (known after apply)
          + disk_encryption_key_sha256 = (known after apply)
          + kms_key_self_link          = (known after apply)
          + mode                       = "READ_WRITE"
          + source                     = (known after apply)

          + initialize_params {
              + image                  = "debian-cloud/debian-11"
              + labels                 = (known after apply)
              + provisioned_iops       = (known after apply)
              + provisioned_throughput = (known after apply)
              + size                   = (known after apply)
              + type                   = (known after apply)
            }
        }

      + network_interface {
          + internal_ipv6_prefix_length = (known after apply)
          + ipv6_access_type            = (known after apply)
          + ipv6_address                = (known after apply)
          + name                        = (known after apply)
          + network                     = "default"
          + network_ip                  = (known after apply)
          + stack_type                  = (known after apply)
          + subnetwork                  = (known after apply)
          + subnetwork_project          = (known after apply)

          + access_config {
              + nat_ip       = (known after apply)
              + network_tier = (known after apply)
            }
        }
    }

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

Changes to Outputs:
  + vm_external_ip = (known after apply)
```

[그림 62] plan 실행결과-2

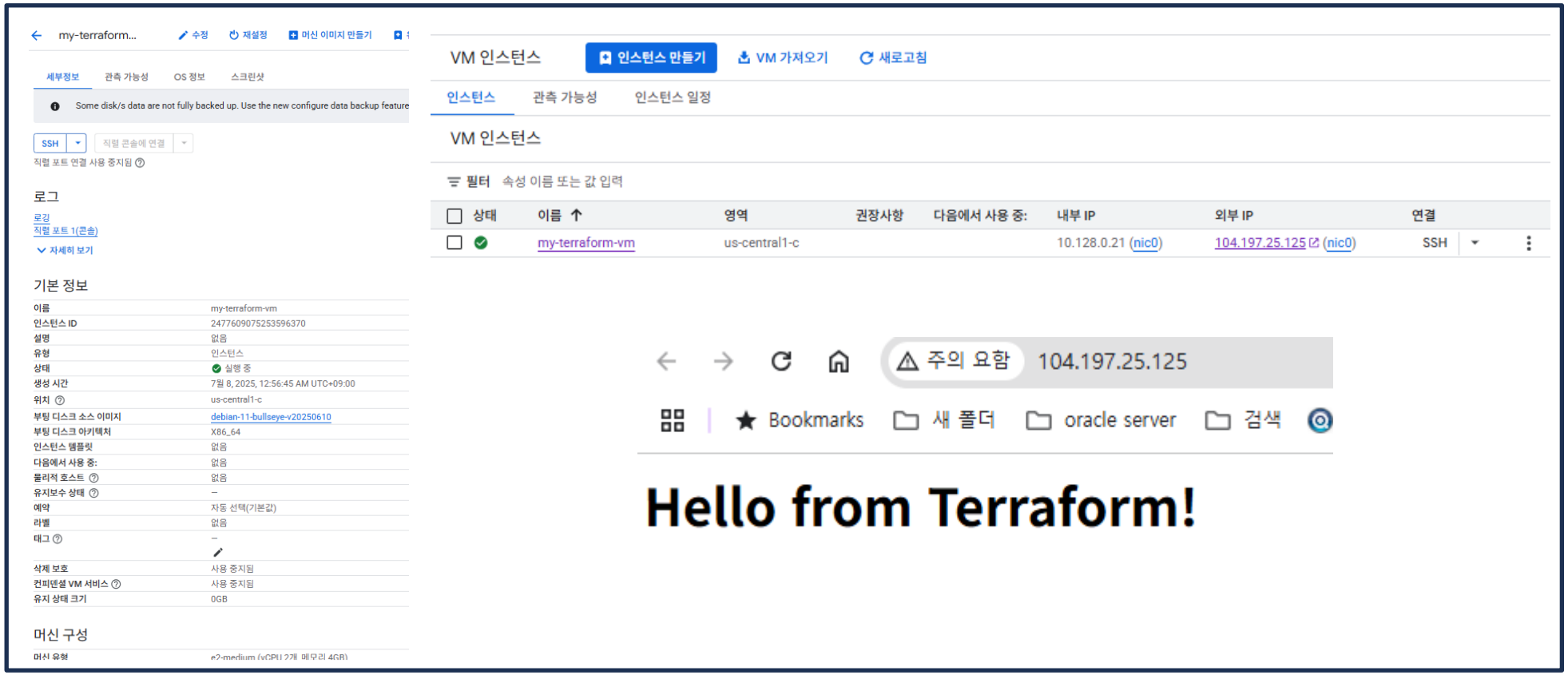
Step 4. main.tf 배포

terraform apply

```
Changes to Outputs:
  + vm_external_ip = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

google_compute_firewall.allow_http: Creating...
google_compute_instance.example_vm: Creating...
google_compute_firewall.allow_http: Still creating... [10s elapsed]
google_compute_instance.example_vm: Still creating... [10s elapsed]
google_compute_firewall.allow_http: Creation complete after 12s [id=projects/gcp-study-463918/global/firewalls/allow-http-from-terraform]
google_compute_instance.example_vm: Still creating... [20s elapsed]
google_compute_instance.example_vm: Creation complete after 28s [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

vm_external_ip = "104.197.25.125"
```

[그림 63] apply 실행결과

Step 4. main.tf 배포 결과



[그림 64] apply 배포결과 확인

Step 5. main.tf 구성 삭제

terraform destroy

```
      - scheduling {
          - automatic_restart   = true -> null
          - min_node_cpus       = 0 -> null
          - on_host_maintenance = "MIGRATE" -> null
          - preemptible         = false -> null
          - provisioning_model  = "STANDARD" -> null
        }

      - shielded_instance_config {
          - enable_integrity_monitoring = true -> null
          - enable_secure_boot          = false -> null
          - enable_vtpm                 = true -> null
        }
    }

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

Changes to Outputs:
  - vm_external_ip = "104.197.25.125" -> null

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

google_compute_firewall.allow_http: Destroying... [id=projects/gcp-study-463918/global/firewalls/allow-http-from-terraform]
google_compute_instance.example_vm: Destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm]
google_compute_firewall.allow_http: Still destroying... [id=projects/gcp-study-463918/global/firewalls/allow-http-from-terraform, 10s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 10s elapsed]
google_compute_firewall.allow_http: Destruction complete after 12s
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 20s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 30s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 40s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 50s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 1m0s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 1m10s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 1m20s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 1m30s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 1m40s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 1m50s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 2m0s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 2m10s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/gcp-study-463918/zones/us-central1-c/instances/my-terraform-vm, 2m20s elapsed]
google_compute_instance.example_vm: Destruction complete after 2m24s

Destroy complete! Resources: 2 destroyed.
```

[그림 65] destroy 결과 확인