# HANDS-ON GOOGLE CLOUD PLATFORM (GCP)

HTTPS://CLOUD.GOOGLE.COM/DOCS/OVERVIEW

# GOOGLE CLOUD PLATFORM (GCP)

- CGP is based on a set of distributed and interconnected Data Centers
- The DC are divided in five major geographic locations: North America, South America, Europe, Asia, and Australia, connected via high-speed networks and subsea cables
- Locations are currently divided into 40 regions and regions into 121 zones
- Region - A geographical region within the world (typically a city) in which at least 3 physical or logical data centers located in different zone. An example of a region is europe-west2, which is located in London, or europe-west8 in Milan
- Zone - A logical or physical data center (a building with computers in it) in which cloud resources physically reside. An example of a zone is europe-west2-c.

# REGIONS AND ZONES



Region

europe-west2

Zones

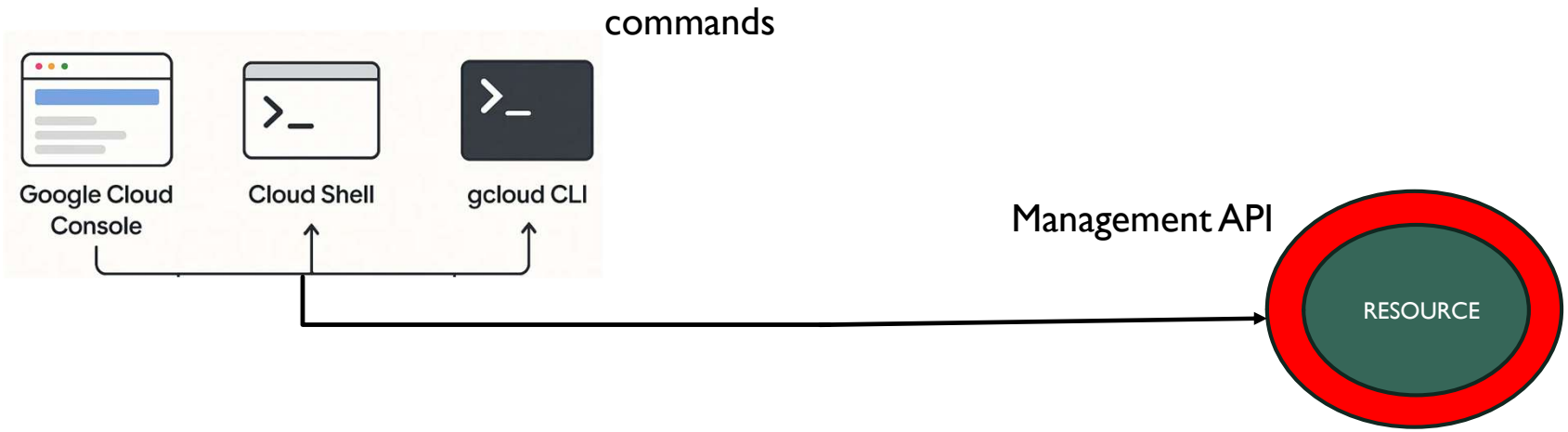
europe-west2-a

europe-west2-b

europe-west2-c

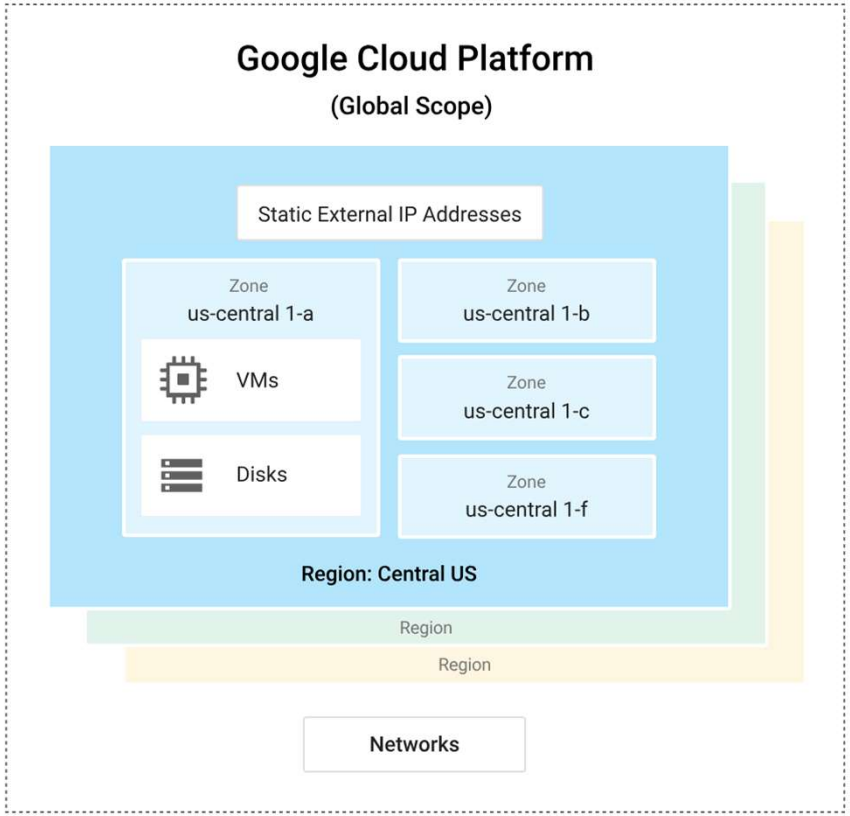
# RESOURCES

- A resource is any component or object that can be managed, configured, or used within the GCP, such as computers and hard disk drives, and virtual resources, such as virtual machines (VMs)
- Resources include organization, projects, and folders
- Resources are managed via API. These API must be enabled before the resource can be utilized



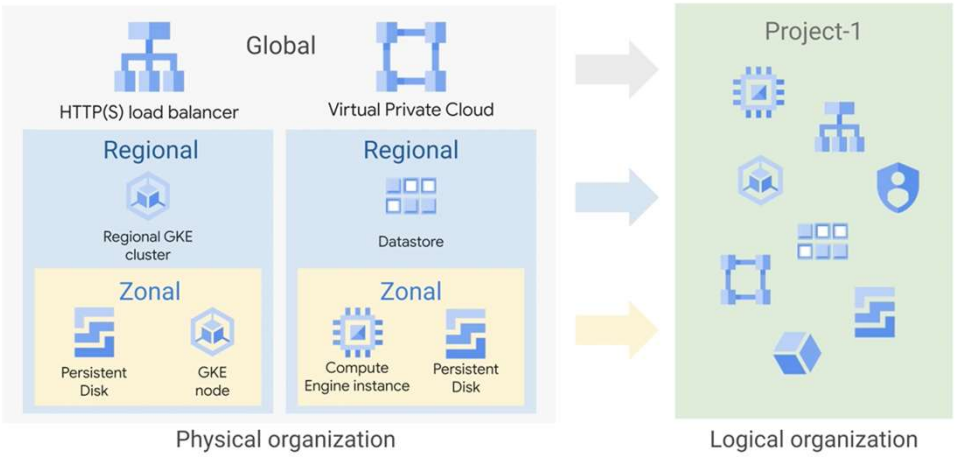
# GLOBAL, REGIONAL AND ZONAL RESOURCES

- **Zonal resources** an be accessed only by resources that are in the same zone.
  - VM instances can mount disks in the same zone.
- **Regional resources** an be accessed only by resources that are located in the same region.
  - Private subnet network, Artifact Registry
- **Global resources** can be accessed by any other resource, across regions and zones.
  - External static IP address, firewall rules, load balancer …



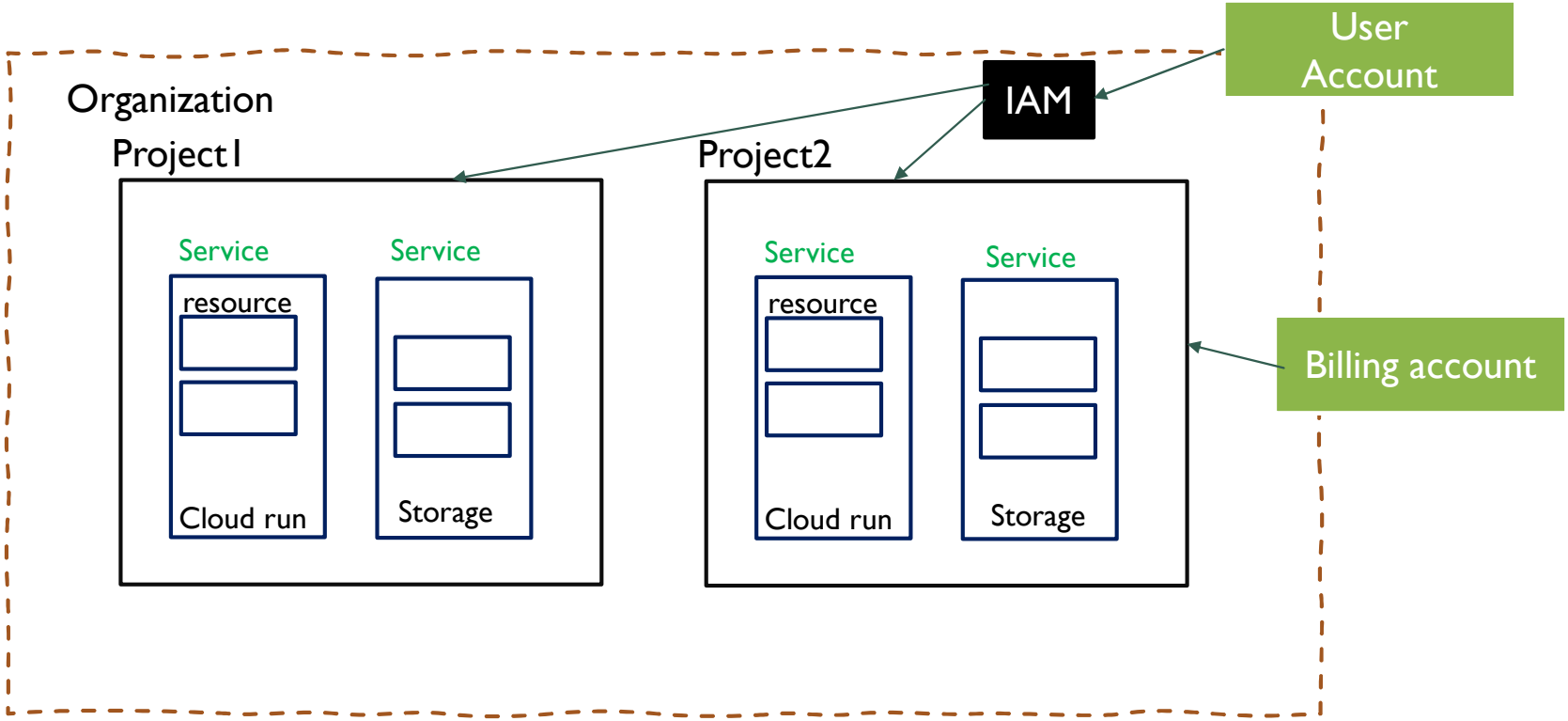
# PROJECTS

- Any Google Cloud resources must belong to a project.
- Each Google Cloud project has the following:
- A project name human readable name.
- A project ID, alpha-numeric string identifier for the project (can be changed).
- A project number, a long number that also identifies the project but is rarely used (it is an internal identifier). provided automatically.



# PROJECTS

- x


User Account
IAM
Organization
Project1
Project2
Service
resource
Cloud run
Service
Storage
Service
resource
Cloud run
Service
Storage
Billing account

# IDENTITY AND ACCESS MANAGEMENT

- IAM (Identity and Access Management) is Google Cloud's system to control who can do what on cloud resources.

1. Resource: project, bucket, VM, repository, etc.
2. Member: user, group, service account, domain
3. Permission: defines what actions can be performed on a resource
4. Role: collection of permissions
5. Policy: association of member + role + resource

# MEMBERS (OR PRINCIPALS)

- Individual users: user:alice@company.com
- Groups: group:dev-team@company.com
- Service accounts: serviceAccount:my-service-account@project.iam.gserviceaccount.com
- Domains: domain:company.com
- Public: allUsers, allAuthenticatedUsers (use with caution)

# TYPE OR ROLES

- Basic roles: Owner, Editor, Viewer
- Predefined roles: service-specific (Cloud Run Admin, Storage Object Viewer, Artifact Registry Reader)
- Custom roles: user-defined roles with specific permissions

# PERMISSIONS

- Permission defines what actions a member can perform on a resource.

- Examples:
- compute.instances.get → read VM information
- storage.objects.create → upload objects to a bucket
- artifactregistry.repositories.downloadArtifacts → pull Docker images

# ROLES

Roles combine multiple permissions (viewer, editor, owner,..)

Permissions inherit top-down

Roles can be assigned at different levels for granular control.

Organization

- Folder
  - Project
    - Resource (VM, bucket, repository...)

# POLICY

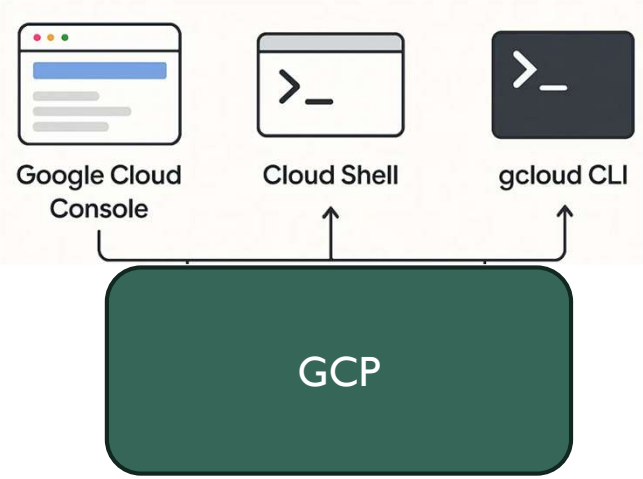
- Policy is a set of rules (bindings) rule stating which members have which roles on which resources

- Alice@company.com:
- roles/viewer on Organization → can view all resources
- roles/editor on project "test-ai" → can modify resources only in that project
- roles/storage.objectAdmin on bucket "data" → can read, write, delete objects

# 3 EXAMPLES

- VM create, start, stop a vm
- Docker container (rest-api with flask)
- Cloud function (simple function in python)

# GCP ACCESS




- Google cloud console : provides a web-based, graphical user interface to manage your Google Cloud projects and resources.
- The CLI allows to manage development workflow and Google Cloud resources in a terminal window. Any operation with point and click can be performed via CLI
- **gcloud** is a CLI provided that can be provided
  - web (into the web browser)
  - SDK that must be installed locally on a machine (requires authentication)
- Commands very useful as commands into a script

https://cloud.google.com/docs/get-started/interact-with-resources

# PROJECT CONFIGURATION

- # Show current configuration
- gcloud config list

- # Set active project
- gcloud config set project PROJECT_ID

- # Set default zone and region
- gcloud config set compute/zone ZONE
- gcloud config set compute/region REGION

- # Show available projects
- gcloud projects list

# VM MANAGEMENT

```
# List all VM instances
gcloud compute instances list

# Create a VM (Ubuntu 22.04)
gcloud compute instances create my-vm \
  --zone=europe-west1-b \
  --machine-type=e2-micro \
  --image-family=ubuntu-2204-lts \
  --image-project=ubuntu-os-cloud \
  --tags=http-server,https-server

# SSH into VM
gcloud compute ssh my-vm --zone=europe-west1-b

# Stop / Start VM
gcloud compute instances stop my-vm --zone=europe-west1-b
gcloud compute instances start my-vm --zone=europe-west1-b

# Delete VM
gcloud compute instances delete my-vm --zone=europe-west1-b
```

# FIREWALL RULES

```
# Create rule to allow HTTP
gcloud compute firewall-rules create allow-http \
  --allow tcp:80 --target-tags=http-server

# Create rule to allow HTTPS
gcloud compute firewall-rules create allow-https \
  --allow tcp:443 --target-tags=https-server

# Add tags to VM
gcloud compute instances add-tags my-vm \
  --tags=http-server,https-server \
  --zone=europe-west1-b
```

# APACHE COMMANDS

```
# Install Apache
sudo apt update
sudo apt install apache2 -y

# Check Apache status
sudo systemctl status apache2

# Start / Stop Apache
sudo systemctl start apache2
sudo systemctl stop apache2
sudo systemctl restart apache2

# Edit default web page
sudo nano /var/www/html/index.html
```

# USEFUL COMMANDS

- # Show help for any gcloud command
- gcloud help
- gcloud compute --help
- gcloud compute instances create --help

- # Tab completion
- gcloud [TAB][TAB]

# TIPS

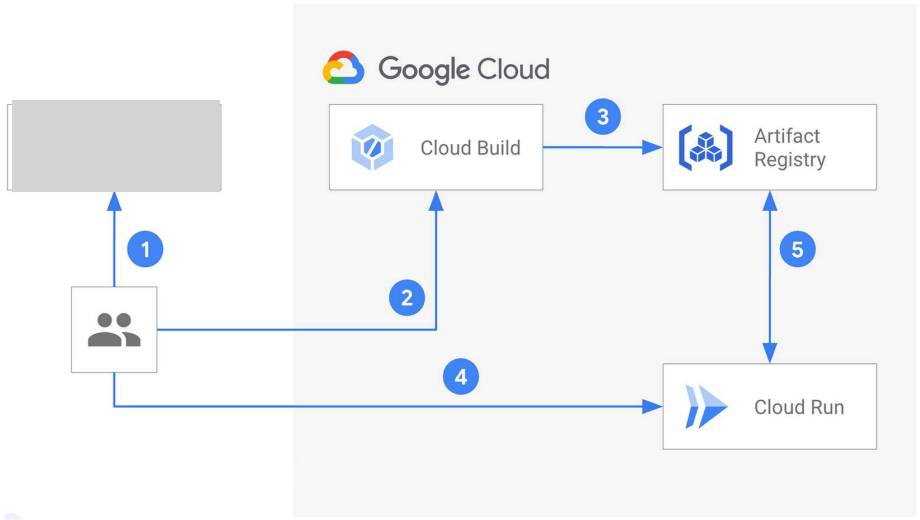
- #Always check the External IP of the VM to view Apache page
- gcloud compute instances list

- #clean up resources when done:
- gcloud compute instances delete my-vm --zone=ZONE

# SIMPLE MANUAL DEPLOY PIPELINE

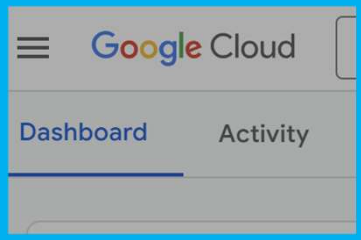
1. Write the code in a directory (cloud shell provides persistent data storage on a VM)
2. Build a docker image using cloud build
3. Submit (push) the image to artifact registry
4. Deploy an image on a cloud run service
5. Cloud Run deploys a container from an image

Pipeline



# STEP 1

- Create a new project from google cloud dashboard




- Set gcloud to use the new project (from CLI)
  - **gcloud projects list**
  - **gcloud config set project PROJECT_ID**
- Enable service api (this allows to use the service)
  - **gcloud services enable run.googleapis.com**
  - **gcloud services enable artifactregistry.googleapis.com**
  - **gcloud services enable cloudbuild.googleapis.com**

# DEVELOP API REST 1/2

- From cloud shell
  - **mkdir cloud-run-api**
  - **cd cloud-run-api**

# WRITE THE MAIN.PY CODE

```
import os
from flask import Flask, jsonify
app = Flask(__name__)
@app.route("/")
def hello_world():
    """Endpoint di esempio che restituisce un JSON."""
    return jsonify(message="Ciao dal mio servizio Cloud Run!")

if __name__ == "__main__":
    # PORT is an env variable from cloud RUN.
    port = int(os.environ.get("PORT", 8080))
    # host '0.0.0.0' allows to be reached from outside.
    app.run(debug=True, host="0.0.0.0", port=port)
```

# REQUIREMENT.TXT

- # requirements.txt
- Flask==2.3.3
- gunicorn==20.1.0

# DOCKER FILE

```
FROM python:3.9-slim # 1. Use an official Python base image
WORKDIR /app # 2. Set the working directory inside the container

COPY requirements.txt . # 3. Copy the dependency file and install them
RUN pip install --no-cache-dir -r requirements.txt
COPY . . # 4. Copy the rest of the application source code

ENV PORT 8080 # 5. Define the environment variable PORT expected by Cloud Run

EXPOSE 8080 # 6. Expose the port on which the container will listen
# 7. Command to start the application using Gunicorn    -w 4: number of workers (processes)    -b :8080: bind to host 0.0.0.0 on port 8080
#    main:app: file main.py, Flask app object
CMD ["gunicorn", "-w", "4", "-b", ":8080", "main:app"]
```

# BUILD AND DEPLOY USING GOOGLE CONTAINER REGISTRY

- **gcloud builds submit –tag gcr.io/PROJECT_ID/flask-api**
- **gcloud run deploy flask-api –image gcr.io/PROJECT_ID/flask-api –platform managed**