

Computing in the Cloud

Big Data and Cloud Computing (CC4093)

Eduardo R. B. Marques, DCC/FCUP

Computing in the cloud

We provide an overview of the main cloud computing models and example cloud service offerings, including:

- Virtual machines in more detail
- Brief reference to GPUs and AI accelerator hardware
- Container-based virtualisation
- Use of Google Compute Engine (for VMs) and Google Cloud Run (for containers)

Reference → the topics in these slides are covered in Chapters 4-6 of [Cloud Computing for Science and Engineering](#).

Virtual machines

Virtual machines (VMs) are virtual computers running on top a physical server machine such that:

- Each VM is a *guest* machine provided with virtual resources corresponding to actual physical resources like memory, CPUs, disks, and network interfaces. Each VM also runs its own operating system (OS) and applications. For all purposes, a VM behaves like a regular computer. The characteristics of a VM are described by **VM images**.
- The server machine, called the **host machine**, pools its **physical resources** to run several independent VMs. VMs are created, started, stopped or destroyed on the fly. Each VM is guaranteed to have an isolated execution environment, e.g., if it crashes, the host machine and other VMs on the same host do not. For all these purposes, the host machine runs a **hypervisor**.
- In a cloud data center, a system called the **fabric controller** manages the allocation of VMs to server machines.

Virtual machines

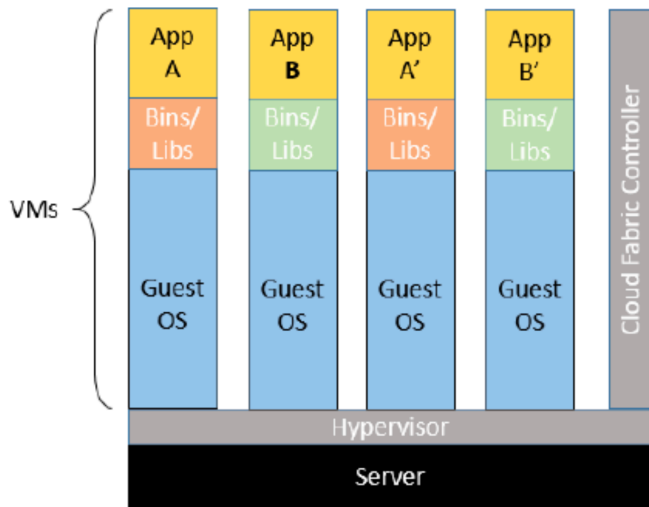





Image source: [Cloud Computing for Science and Engineering](#).


Google Compute Engine - VM creation


Name 


Region  **Zone** 

Machine type
Customize to select cores, memory and GPUs.

0.6 GB memory [Customize](#)

Container 
☐ Deploy a container image to this VM instance. [Learn more](#)

Boot disk 



New 10 GB standard persistent disk
Image
Debian GNU/Linux 9 (stretch)

Main parameters: **Name**, **Region** and **Zone** (defines the hosting data center), **Machine Type** (base CPU + memory configuration), and **Boot disk** (initial OS image + disk size). **Instance templates** can be used for the creation of VMs.

Google Compute Engine - billing

\$4.79 monthly estimate

That's about \$0.007 hourly

Pay for what you use: No upfront costs and per second billing

⌵ Details

- **Basic principle:** pay for what you use.
- VMs are charged according to the computing time they use (while they are turned on), their configuration in terms of CPUs, RAM, and storage, and the data center they run on.

Google Compute Engine - sole-tenant modes

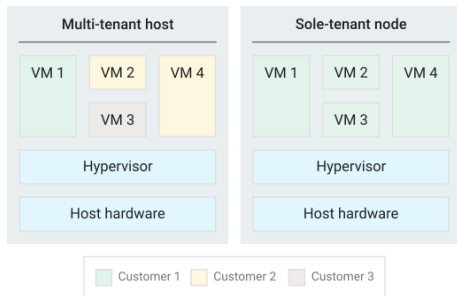


Image source: [Google Compute Engine documentation](#)

Sole tenancy → a client may reserve a server machine that is exclusively dedicated to running the client's VMs. Sole-tenant modes can be managed entirely by the client providing a more isolated environment, more stable performance, and efficient resource sharing between VMs.

Google Compute Engine - instance groups

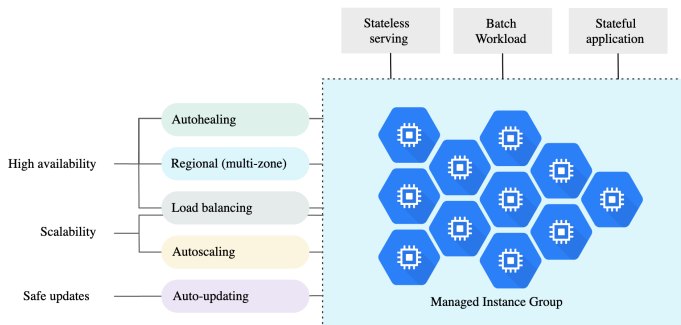


Image source: [Google Compute Engine documentation](#)

An **instance group** is a set of VMs created from the same instance template, whose size may grow or contract over time. **Typical use case:** instances in the group provide the same service (e.g. web content serving, queue-based computing workloads) in load-balanced manner.

Google Compute Engine - instance groups (cont.)


A few mechanisms associated to Compute Engine instance groups:

- **Auto scaling:** group size may contract or grow over time, according to configurable criteria.
- **Load balancing:** client requests may be balanced across instances in the group.
- **Auto healing:** health checks may be configured so that non-responding VMs are stopped and new ones are started.
- **Regional coverage:** instances may be allocated to different zones (data centers) in the same region (e.g. Europe).

Google Compute Engine - instance groups (cont.)

Example settings:

Autoscaling

Use autoscaling to allow automatic resizing of this instance group for periods of high and low load. [Autoscaling groups of instances](#) 

Autoscaling mode

Autoscale 

Autoscaling policy

Use metrics and schedules to determine when to autoscale the group.

[Autoscaling policy and target utilization](#) 

CPU utilization: 60% (default) 

Cool down period

Specify how long it takes for your app to initialize from boot time until it is ready to serve.

[Cool down period](#) 

60

seconds

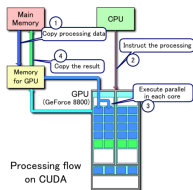
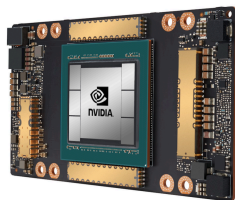
Minimum number of instances

1

Maximum number of instances

4

GPUs




A **Graphic Processing Unit (GPU)** is specialised hardware processor for graphics processing.

GPUs are not general-purpose processors, in fact they must be used in conjunction with CPUs, but provide **Single Instruction Multiple Data (SIMD)** (also called vectorised) instructions that perform an operation (e.g. vector addition, multiplication) on multiple data operands concurrently. Originally developed for high-performance graphics (e.g., in games), GPUs are also heavily used in AI/machine learning (e.g. deep neural networks) and other HPC applications.

Google Compute Engine and GPUs

Machine type

n1-standard-1 (1 vCPU, 3.75 GB memory)


	vCPU	Memory	GPUs
	1	3.75 GB	1 x NVIDIA Tesla K80


CPU platform ?
CPU platform configuration is permanent


Automatic


GPU type **Number of GPUs**

NVIDIA Tesla K80 1

 **NVIDIA RTX Virtual Workstation - Windows Server 2016**
NVIDIA
GPU-Accelerated Cloud Computing

 **NVIDIA Gaming PC - Ubuntu 18.04**
NVIDIA
GPU-Accelerated Cloud Gaming

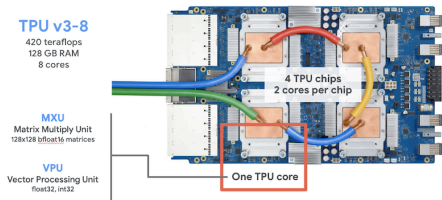
 **NVIDIA Image for AI using GPUs**
NVIDIA
Pre-Trained Models, AI SDKs and Optimized Containers from NVIDIA

 **NVIDIA GPU-Optimized Image for Deep Learning, ML & HPC**
NVIDIA
Maximize GPU performance for your AI applications

GPUs can be provisioned in association to virtual machines (and can be relatively expensive; you may [use a GPU within Google Colab](#) though)!

Through Google Cloud, Nvidia provides ready-to-use virtual machine images that include GPUs and dedicated software for virtual desktop environments, cloud gaming, and AI/deep learning. [More info here](#).

AI accelerator hardware



Beyond GPUs, specialised hardware has been developed for AI applications, known as **AI accelerators**. One example is that of **Tensor Processing Units (TPUs)** developed by Google, designed to accelerate AI applications, in particular those that make use of the [TensorFlow](#) software library. Google provides access to TPUs through GCP but also for free (with usage limitations) through [Google Colab](#) and [Kaggle](#).

Containers

Containers define a different paradigm of virtualisation.

- A container is layered on top of a **guest OS** rather than a full OS. The guest OS runs on top a host OS enable by a host machine (that may be a VM).
- A **container image** specifies the software packages to be available and programs that should be activated with each container instance, rather than the hardware characteristics and operating system.
- By sharing the same guest OS, containers can be created and destroyed very fast as needed, and use much less resources than virtual machines.
- Multiple instances of different container images may be running on the same machine. Usually containers in the same machine interact and may share resources like disk storage or network connections in the host machine.

Containers

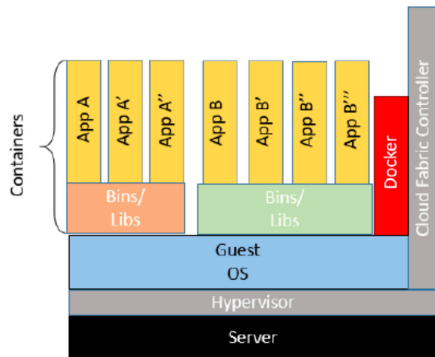


Image source: [Cloud Computing for Science and Engineering](#).

Containers - Docker

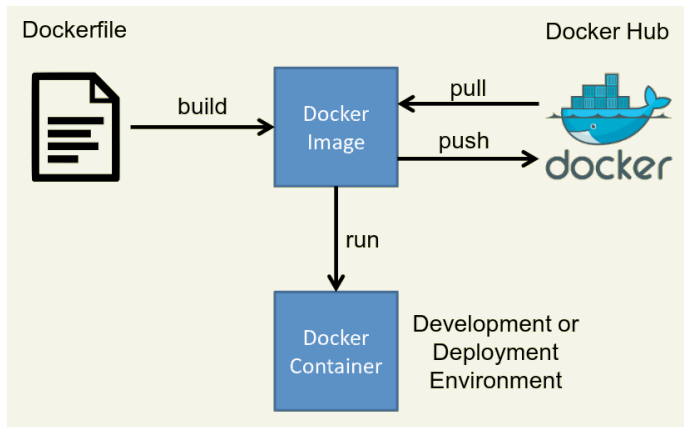


Image source: [Developing in Docker](#), Mark Buckler

Docker is the most popular container engine. Container images are defined using text files known as “**Dockerfiles**”. Images can be reused as base for other images. and can be made publicly available at [Docker Hub](#).

Docker example

Simple Dockerfile adapted from Docker Tutorials and Labs

```
# our base image
FROM alpine:3.5

# Install python and pip
RUN apk add --update py2-pip

# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/

RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 8080

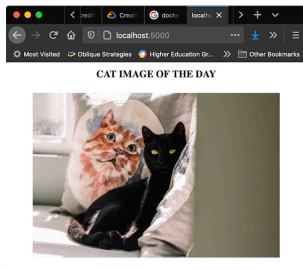
# run the application
CMD ["python", "/usr/src/app/app.py"]
```

Docker example (cont.)

Building a container image - this needs to be done just once.
Alternatively we can pull an image from Docker Hub.

```
$ ls
Dockerfile app.py requirements.txt templates
$ docker build . -t example
Step 1/9 : FROM alpine:3.5
3.5: Pulling from library/alpine
8cae0e1ac61c: Pull complete
Digest: sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed24
Status: Downloaded newer image for alpine:3.5
---> f80194ae2e0c
Step 2/9 : RUN apk add --update py2-pip
---> Running in 9f97589a16dd
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/main/x86_64/AP
...
Successfully built e383f0394693
Successfully tagged example:latest
```

Docker example (cont.)



Running the container:

```
$ docker run --rm -it -p 8080:8080 example
```

```
* Serving Flask app "app" (lazy loading)
```

```
* Environment: production
```

```
WARNING: Do not use the development server in a production
Use a production WSGI server instead.
```

```
* Debug mode: off
```

```
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Containers on Compute Engine VMs

Container ?

☒ Deploy a container image to this VM instance. [Learn more](#)

Container image ?

jupyter/datascience-notebook


⌵ [Advanced container options](#)

Boot disk ?



New 10 GB balanced persistent disk

Image

 Container-Optimized OS 85-13310.1...

Change

It is possible to deploy a single container in association to a single VM or a VM instance group. This is not flexible in the sense that there is a 1-to-1 relation between container and VM instances.

Google Cloud Container Registry

A more simplified approach is through [Container Registry](#) and [Cloud Run](#). Containers can be pushed to the Container Registry and then be deployed using Cloud Run.

Example (project_id: active project id)

```
# General config
$ gcloud services enable containerregistry.googleapis.com
$ gcloud auth configure-docker

# Tag image with gcr.io/PROJECT_NAME/CONTAINER_NAME
$ docker tag example gcr.io/project_id/example
$ docker push gcr.io/project_id/example

# Deploy container
$ gcloud run deploy example \
  --image gcr.io/project_id/example
```

Google Cloud Run configuration

General

Container image URL *

gcr.io/bdcc21/example

SELECT

E.g. us-docker.pkg.dev/cloudrun/container/hello

Should listen for HTTP requests on \$PORT and not rely on local state. [How to build a container?](#)

Container port

5000

Requests will be sent to the container on this port. We recommend listening on \$PORT instead of this specific number.

Google Cloud Run configuration (cont.)

Capacity

Memory allocated

256 MiB



Memory to allocate to each container instance.

CPU allocated

1



Number of vCPUs allocated to each container instance.

Request timeout

300

seconds

Time within which a response must be returned (maximum 3600 seconds).

Maximum requests per container

80

The maximum number of concurrent requests that can reach each container instance.

[What is concurrency?](#)

Autoscaling

Minimum number of instances *

0

Maximum number of instances

100

Container orchestration

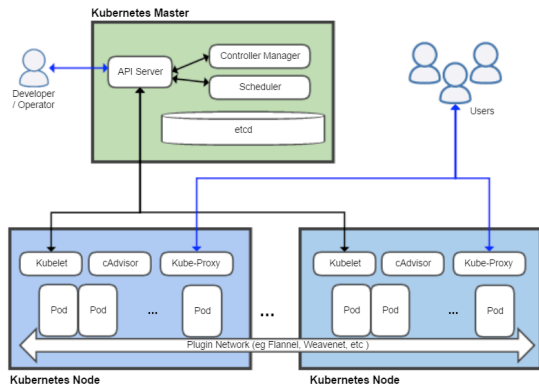


Image source: [Wikipedia - Kubernetes](#)

Kubernetes and **Docker in “swarm mode”** provide for the automated orchestration of containers, including features such as container recovery, auto-scaling, or load balancing. Kubernetes for instance allows groups of containers, called **pods**, to be deployed over a cluster of machines.

Google Cloud and the Kubernetes Engine

For more advanced deployments we can use the **Kubernetes Engine**. We won't go into details regarding the orchestrated use of containers, but the following example configurations illustrate a few aspects in the configuration of a Kubernetes cluster:

Size

Number of nodes *

Pod address range limits the maximum size of the cluster. [Learn more](#)

☒ Enable autoscaling 

Minimum number of nodes *


Maximum number of nodes *

Image type

Container-Optimized OS with Docker (cos) (default)  

Machine types for common workloads, optimized for cost and flexibility

Series

E2 

CPU platform selection based on availability

Machine type

e2-medium (2 vCPU, 4 GB memory) 