ACP / 1

Michael Glienecke, PhD

HAVE FUN

Introduction to ACP and overview

- Who is that man at the blackboard? A bit about my background…

- How is the course organized?
- Important dates

- What do I want to convey to you?
- Expectations
- Reading materials

Cloud computing core idea

- **Cloud computing is distributed system computing over the Internet**
- Common IT infrastructure
  - Runtime environment (pod, container, OCI)
  - Storage
  - Databases
  - Networking
- Services
  - IoT
  - ML, AI
  - …

Cloud computing promises

- No dedicated servers, no data centres
    - Reduced cost
    - High Availability
    - Horizontal and Vertical Scale on demand
- Absolute freedom where you want to have systems (and data)
    - Multiple geo-locations (GDRP-compliance, etc.)
    - Disaster recovery

What is "Cloud Programming"?

- Access to the provided services in a programmatic way
- It involves:
  - A client application
  - An API of some kind (often REST)
    - Sometimes a client library to make access simpler and encapsulated in i.e. objects
  - A service which recognizes the requests and serves them accordingly

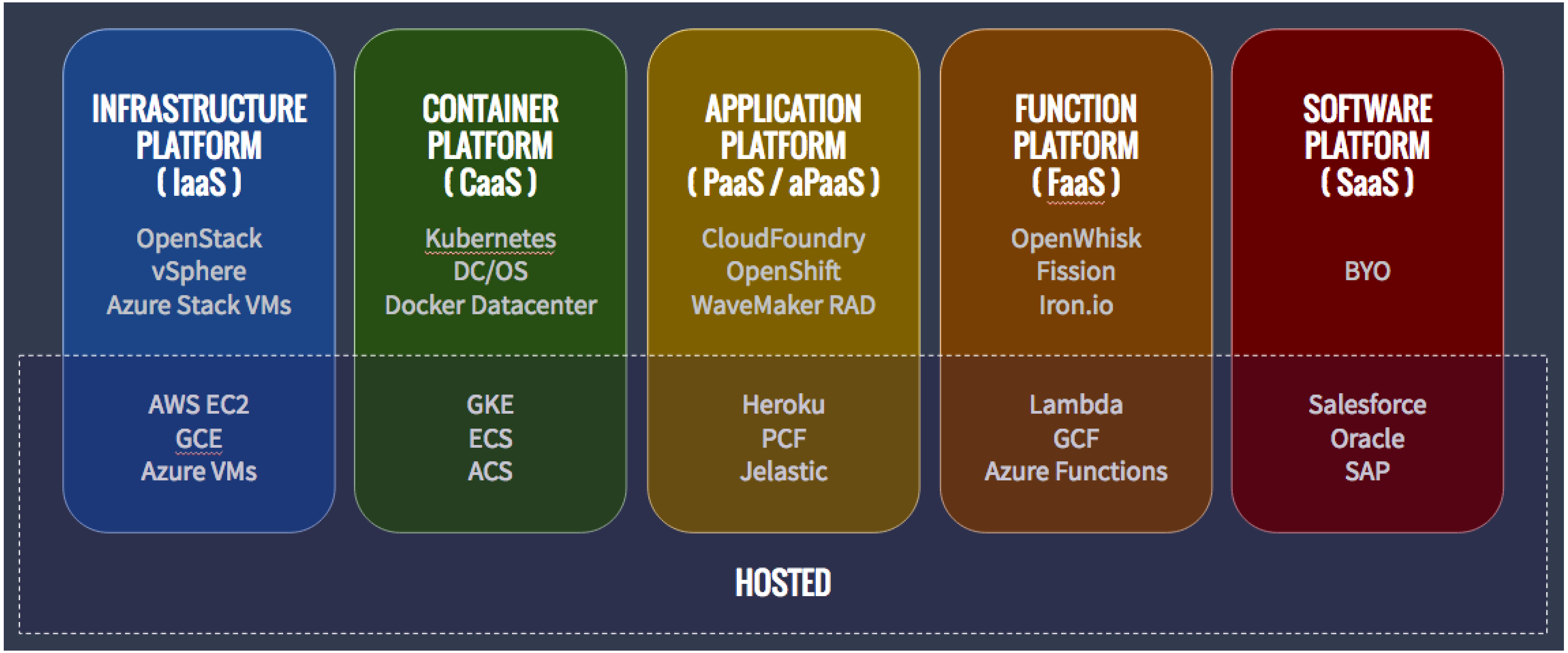
- Example -> Azure blob storage client (list blobs)
  - Java: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-java?tabs=powershell%2Cmanaged-identity%2Croles-azure-portal%2Csign-in-azure-cli

  - REST: https://learn.microsoft.com/en-us/rest/api/storageservices/list-containers2?tabs=microsoft-entra-id

Setting the stage

- XaaS (*here business people become excited…*)
- The big 3 providers
- Cloud-APIs
- Containerization (overview)

XaaS



https://medium.com/@nnilesh7756/what-are-cloud-computing-services-iaas-caas-paas-faas-saas-ac0f6022d36e

XaaS

- Mostly in use is SaaS, PaaS and CaaS
- SaaS
  - Oracle(People and Money used by UoE, etc.)
  - Salesforce
  - Here a discussion has been started, if this is relevant in the future with GenAI
  - …
- PaaS
  - CloudFoundry
- CaaS
  - docker
  - Kubernetes
  - Other runtimes for containers
- MaaS
  - Metal as a Service (marketing...)

IaC (Infrastructure as Code)

- Manage infrastructure dynamically via configuration files and (optionally) API-calls than through a graphical user interface
- Terraform is the market leader https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code

The big 3 (++) providers

- Microsoft (Azure) https://azure.microsoft.com/
- Amazon Web Service (AWS) https://aws.amazon.com/
- Google Cloud Services - https://cloud.google.com/

- IBM Cloud https://www.ibm.com/cloud
- …

THE UNIVERSITY *of* EDINBURGH
informatics

Get your free cloud account

- aws: https://aws.amazon.com/education/awseducate/
- Azure: https://azure.microsoft.com/en-gb/free/students
- Localstack: https://www.localstack.cloud/localstack-for-students
- Will be needed for many examples / testing

Services

- So, what is a service actually?
  - *Well, once upon a time…*

- A bit of history of services from the 1960s to 202x
  - Mainframe services (CICS, EHLLAPI, …)
  - Native communication between client - server
  - CORBA
  - SOAP
  - REST
  - gRPC, Avro

Cloud-APIs

- The API a "service" (running in the cloud) provides

- Azure as example
  https://azure.microsoft.com/en-us/products/
  https://aws.amazon.com/

Showcasing a super-simple service

- https://github.com/mglienecke/IlpTutorialRestService

Github

- Use it, please!

- Do regular commits – not just a big one at the end

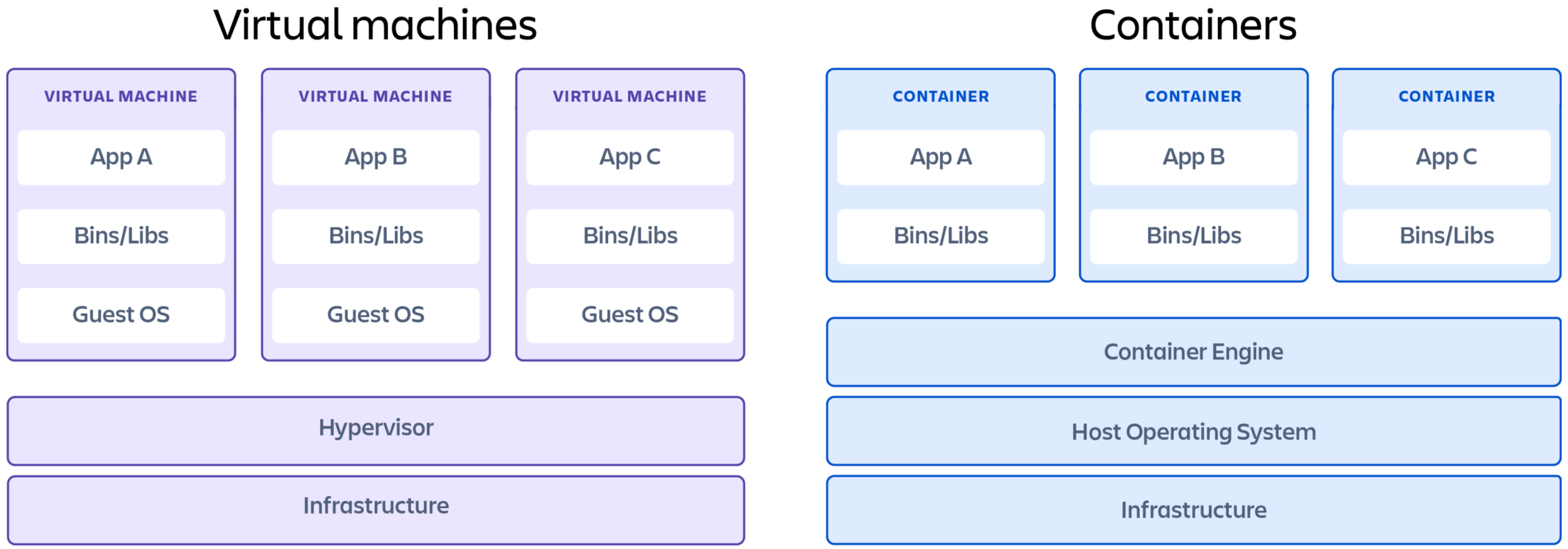
- If something happens to your submission this is the **only proof**, you have (commit history)!

Why and How to virtualize your system?

- The original idea was born in the late 1960s, then in 1972 IBM produced VM/370 which included virtual machines
  - More like current containers, though
- Mainly to run several things at the same time in segregated boxes
- Later we had "proper" virtual machines on Intel architecture (and others)

- Scaling, Failover and Load Management remained issues…
- Then came docker to make everything a bit lighter

VM vs Containers



https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms

Scaling your load

- If your computing power is not enough, you have 2 options:

- Vertical scaling
  - adding more CPUs, memory, etc. -> make the system bigger as such
  - Works only until the next limit is hit (again)

- Horizontal scaling
  - Adding more machines to distribute the load
  - Works almost unlimited

Horizontal scaling

- Can be done using metal
  - Is getting complex and expensive, soon
    - Especially if the load is not predictable
  - If you know and can guarantee that you have min 60% and max. 150% load, scaling is easier

- Can be done using containers
  - Systems can breathe – expand and collapse on demand

Issues during scaling

- Load balancing - How to make sure equal load is on each system
  - Sticky bit to keep sessions on systems
- SSL termination – that is rather low-level tech stuff…
  - Most systems use http (instead of https) internally in the cluster
- High availability
  - What happens if something happens – who takes over
  - Disaster recovery

Containerization - how

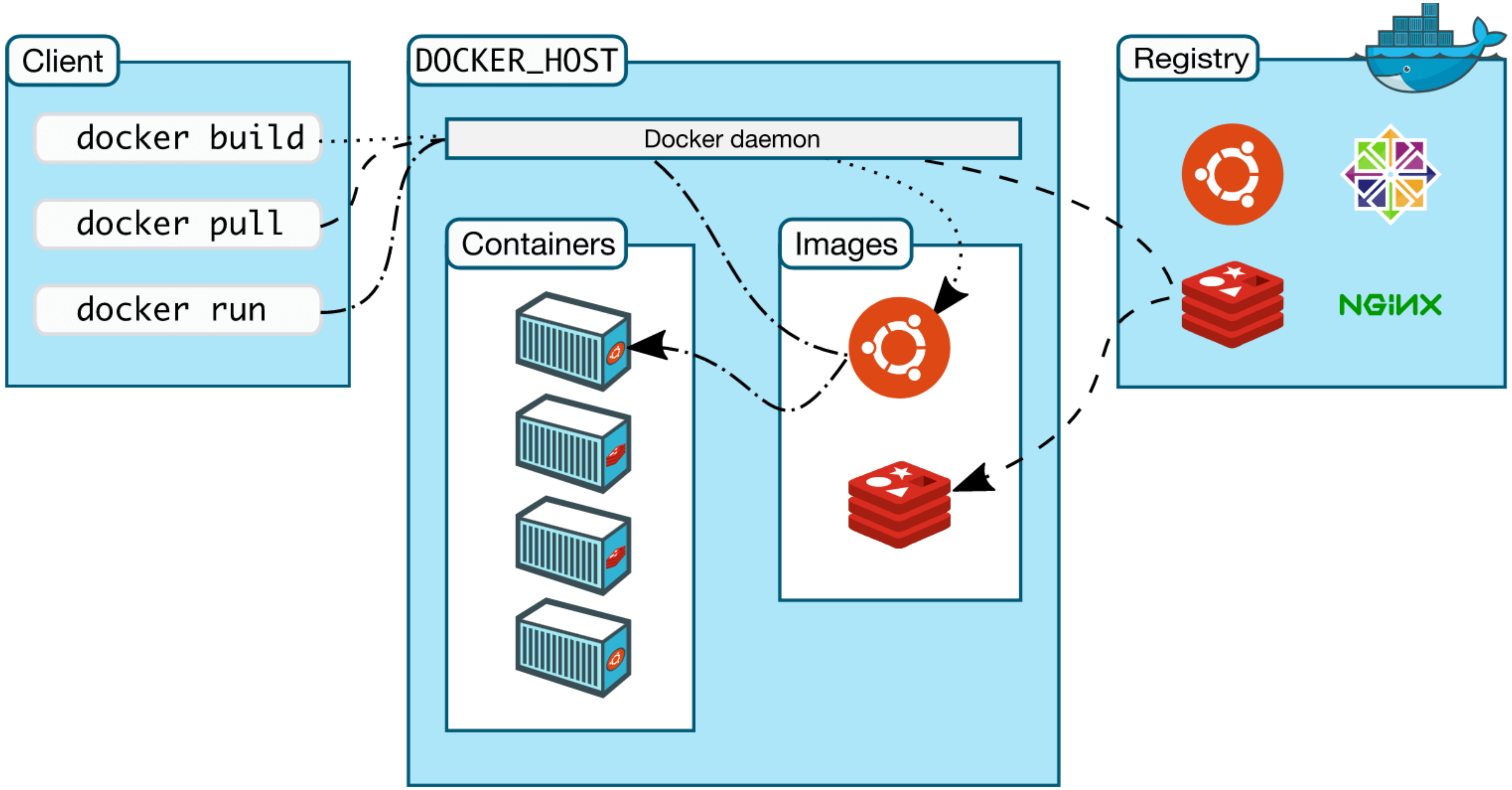
- OCI based
  - https://opencontainers.org/about/overview/
  - https://opencontainers.org/community/overview/

- Images form the runnable applications
- Are loaded on demand
- Executed when needed
- All networking is mapped
- File system access is mapped to volumes

Containerization - examples

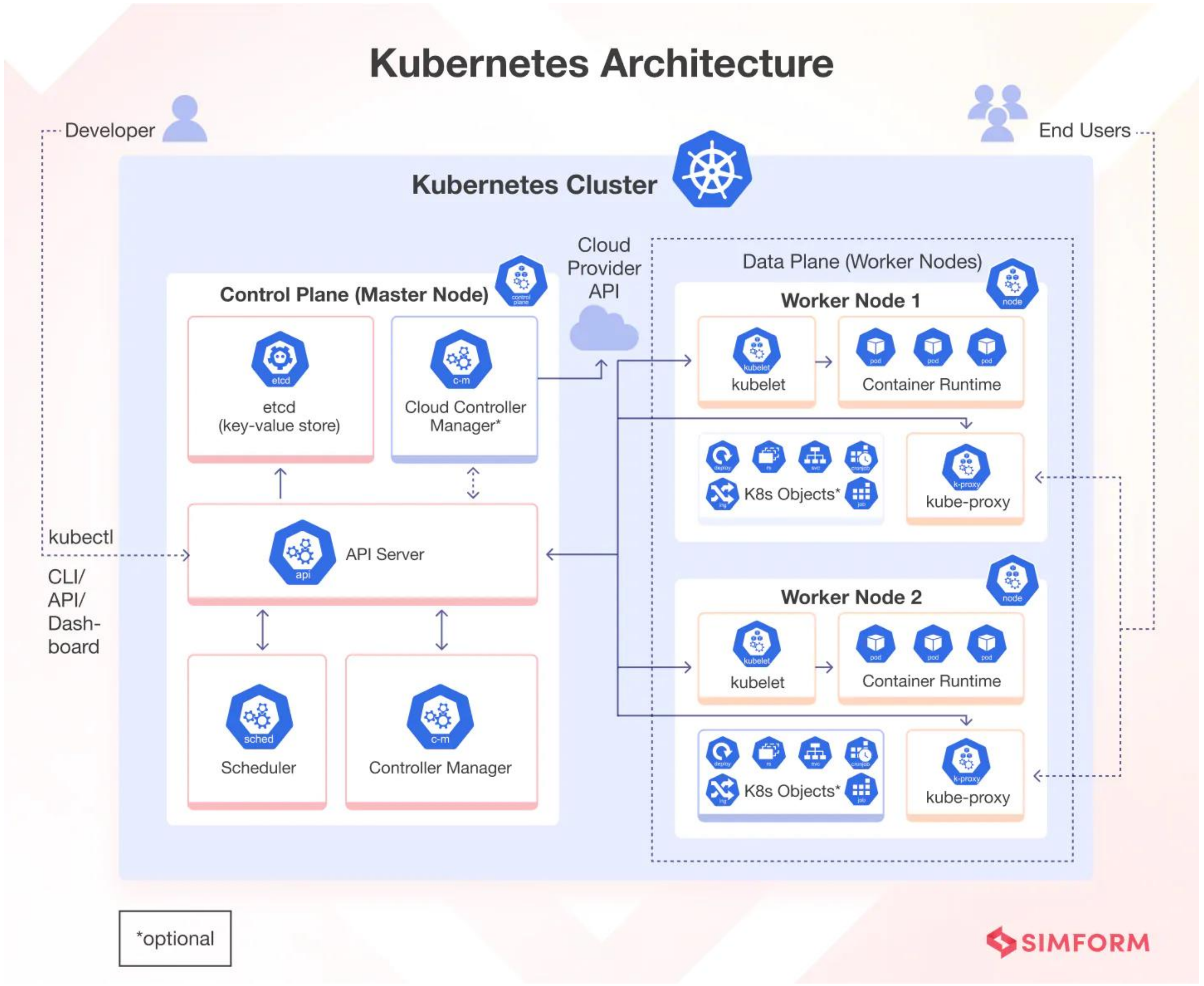
- docker
- docker swarm
- kubernetes

docker



https://www.cherryservers.com/blog/a-complete-overview-of-docker-architecture

Kubernetes



https://www.simform.com/blog/kubernetes-architecture/

Kubernetes local install

- docker desktop -> activate Kubernetes
- Rancher
  https://www.rancher.com/products/rancher-platform
- kind (kubernetes in docker)
- k3d
- Minikube
- MicroK8s

Things we will cover

- Containers
  - Using docker, docker compose
- local stack (aws)
- SQS + SNS, Kafka, RabbitMQ
  - Messaging infrastructure
- DynamoDB, PostgreSQL, S3
  - No-SQL + SQL
- Redis
  - caching