



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

Cloud Computing

Dr. Philipp Leitner



philipp.leitner@chalmers.se



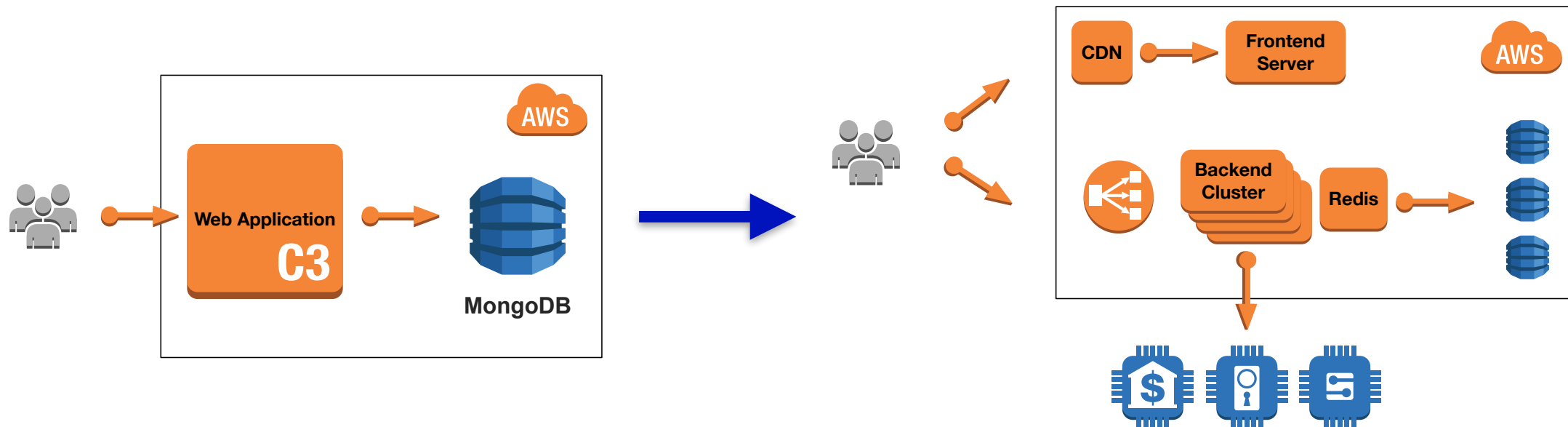
@xLeitix

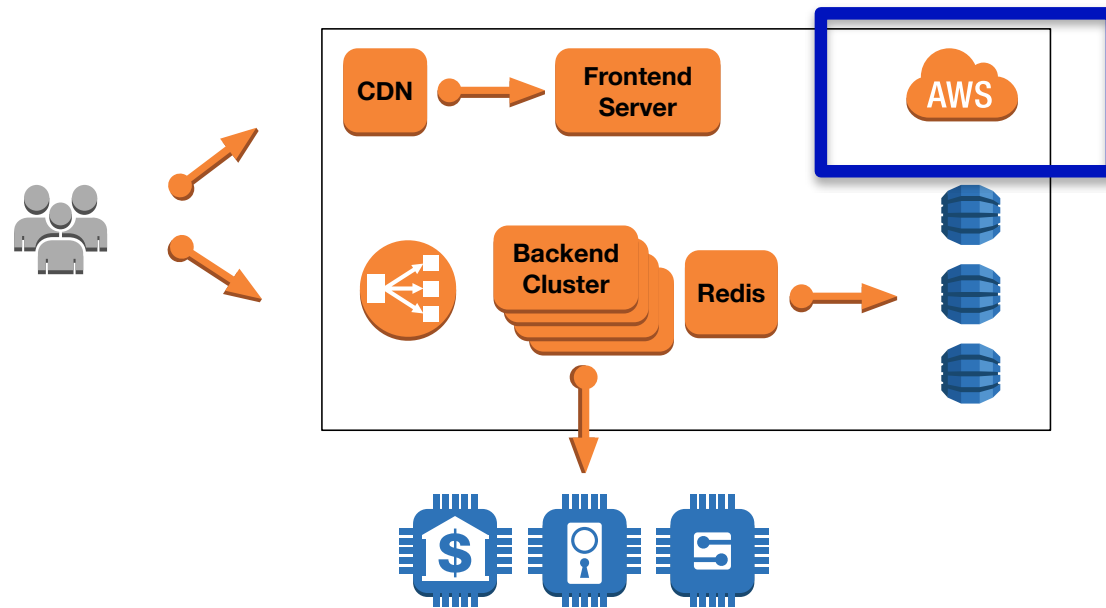
LECTURE 3

Covers ...

- Basics of Cloud Computing**
- Containerisation, Docker, and docker-compose**

Scalability and availability as driving the migration to scaled architectures





Cloud Computing

Most basic definition

“Have an external entity do your computing (provide computing resources) for you”

Examples:

- Dropbox / Google Photos (external entity provides storage)
- Github (external entity provides storage plus other services)
- Amazon EC2 (external entity provides virtual machines)
-

Unfortunately this is not a very useful definition ...

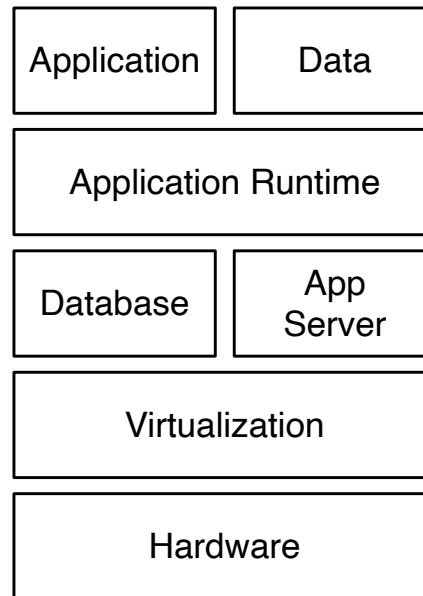
Larry Ellison (ex-CEO of Oracle Inc.)

“The interesting thing about cloud computing is that we’ve redefined cloud computing to include everything that we already do.”

Source:
<https://www.cbronline.com/cloud/he-said-what-5-things-larry-ellison-actually-said-about-cloud-4563323/>

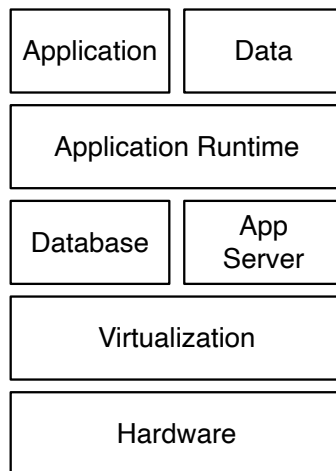


A Typical Application Stack

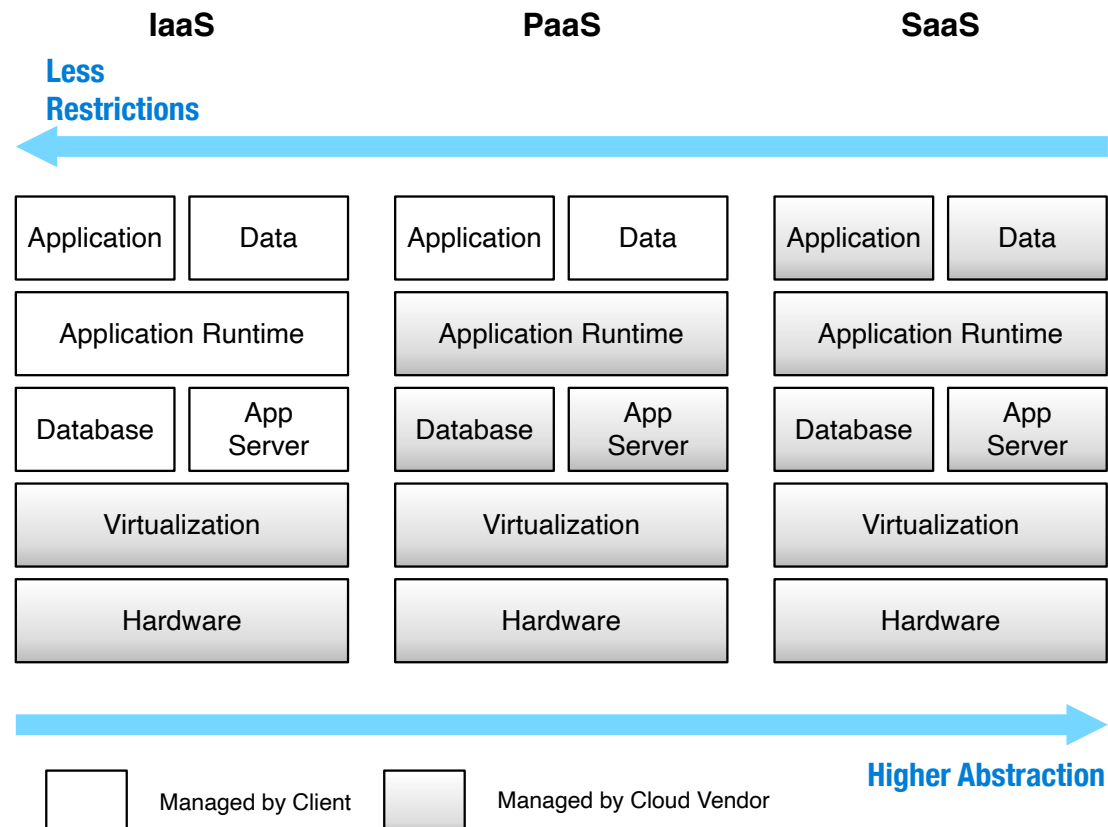


(For example: the ScalyShop backend)

On-Premise



Cloud

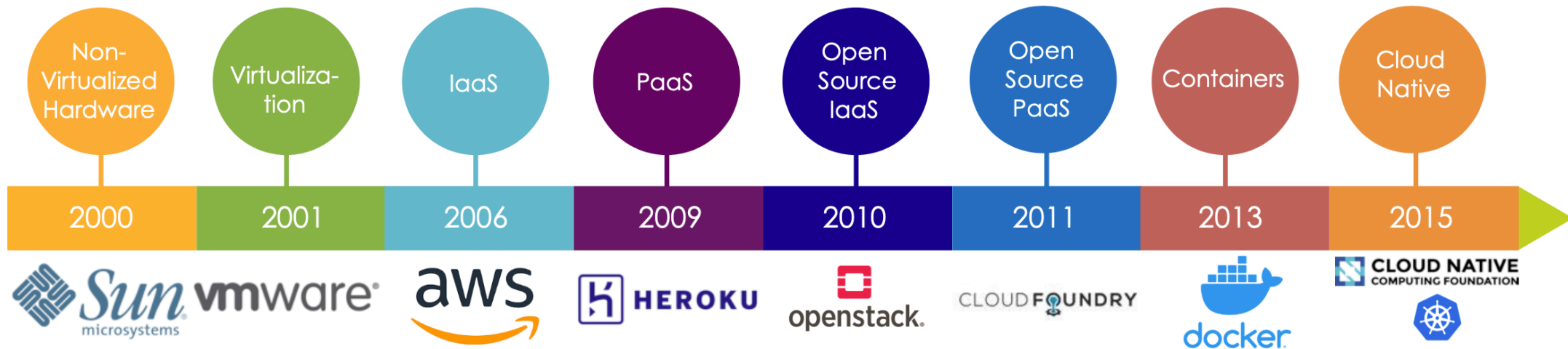


CLOUD NATIVE COMPUTING FOUNDATION



kubernetes

- Cloud native computing uses an open source software stack to:
 - segment applications into *microservices*,
 - package each part into its own *container*
 - and dynamically *orchestrate* those containers to optimize resource utilization



NIST Stack (XaaS Stack)

“Infrastructure-as-a-Service” (IaaS)

Developers get access to hosted VMs / containers, but manage all software above the operating system layer themselves

“Platform-as-a-Service” (PaaS)

Developers get access to hosted application servers, runtime environments, or databases, but need to program the actual application themselves

“Software-as-a-Service” (SaaS)

Developers get access to entire applications or services, which require / allow for only minimal customization

Vendor Lock-In

“Moving up” in the cloud stack is often compelling

- Cost advantages
- Increased business agility
- Reduced need for sysadmin / DevOps staff

But comes with:

- Increasing reliance on provider
- Vendor lock-in (difficult / impossible to change)
- Inability to “lift-and-shift” migrations

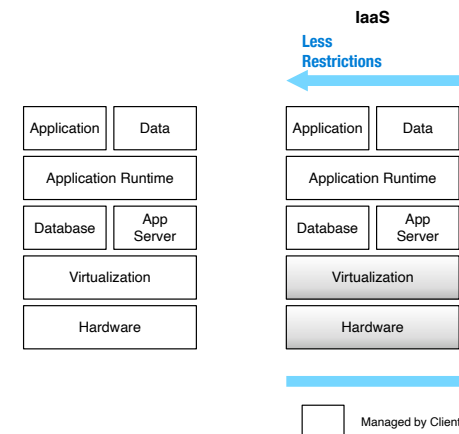
“Lift-and-shift” Migration

“Lift-and-shift” migrations are migrations that do not require code changes

- Simply install the application elsewhere and move customer data
- No need to do (non-trivial) code changes
- Only possible if:
 - Old and new provider share the exact same API
 - App does not use any provider-specific features or APIs

Often possible between IaaS services (or going from on-prem to IaaS),
rarely possible between PaaS services, **basically never feasible** for SaaS services.

On-Premise



Many Cloud Providers



IBM Bluemix™



**Suggested platform for
project**

Short Google Cloud Demo

Basic Cloud Computing Tenets

Pay-as-you-go pricing

Customers pay per usage time unit, rather than a fixed price

Resource virtualisation

Customers rent “virtual” computing rather than physical assets

Elasticity

Customers can quickly change how many resources they are leasing

Multi-tenancy

Physical hardware is shared between customers

Illusion of infinite resources

The cloud in practice does not “run out” of computing

Customers can scale up (virtually) indefinitely

Basic Cloud Computing Tenets

Pay-as-you-go pricing

Customers pay per usage time unit, rather than a fixed price

Resource virtualisation

Customers rent “virtual” computing rather than physical assets

Elasticity

Customers can quickly change how many resources they are leasing

Multi-tenancy

Physical hardware is shared between customers

Illusion of infinite resources

The cloud in practice does not “run out” of computing

Customers can scale up (virtually) indefinitely

Pay-as-you-go Pricing

Analogy:

Imagine renting / leasing a car vs. buying it

Advantage:

CapEx (capital expenses) -> OpEx (operational expenses)
(requires no up-front investment)

May be cheaper (especially if usage is not sustained)

Disadvantage:

Costs difficult to predict upfront

Possible “success disasters”, since costs scale with usage, not with business success

“In the past, if your application became popular and your systems or your infrastructure did not scale you became a victim of your own success. Conversely, if you invested heavily and did not get popular, you became a victim of your failure.” Jinesh Varia, Amazon

Basic Cloud Computing Tenets

Pay-as-you-go pricing

Customers pay per usage time unit, rather than a fixed price

Resource virtualisation

Customers rent “virtual” computing rather than physical assets

Elasticity

Customers can quickly change how many resources they are leasing

Multi-tenancy

Physical hardware is shared between customers

Illusion of infinite resources

The cloud in practice does not “run out” of computing

Customers can scale up (virtually) indefinitely

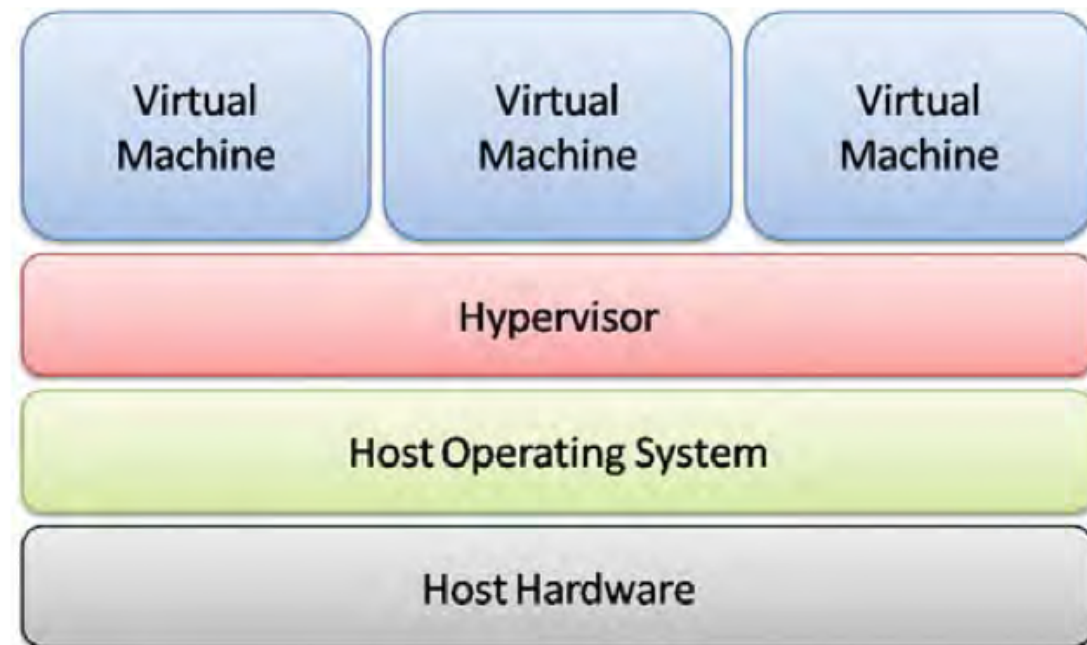
Resource Virtualization

Virtualization of:

Compute (virtual machines, containers)

Network (Software-Defined Networks)

Storage (Virtual disks, cloud storage)

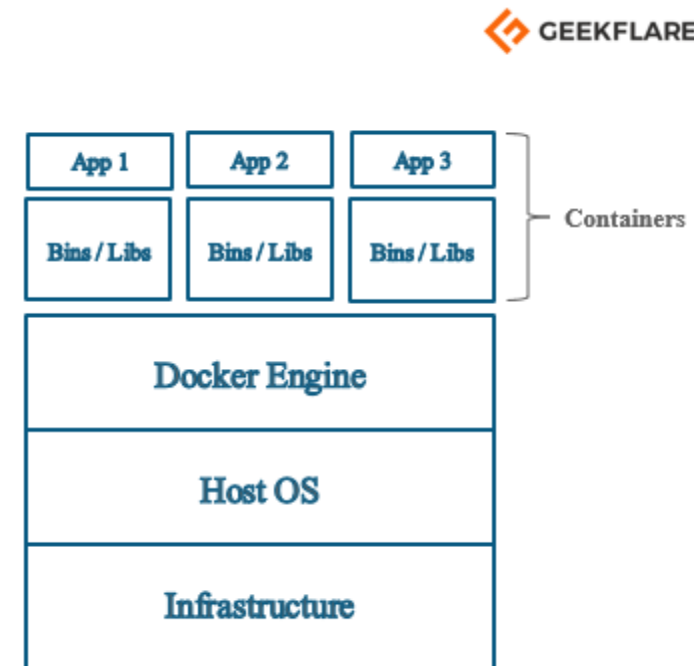
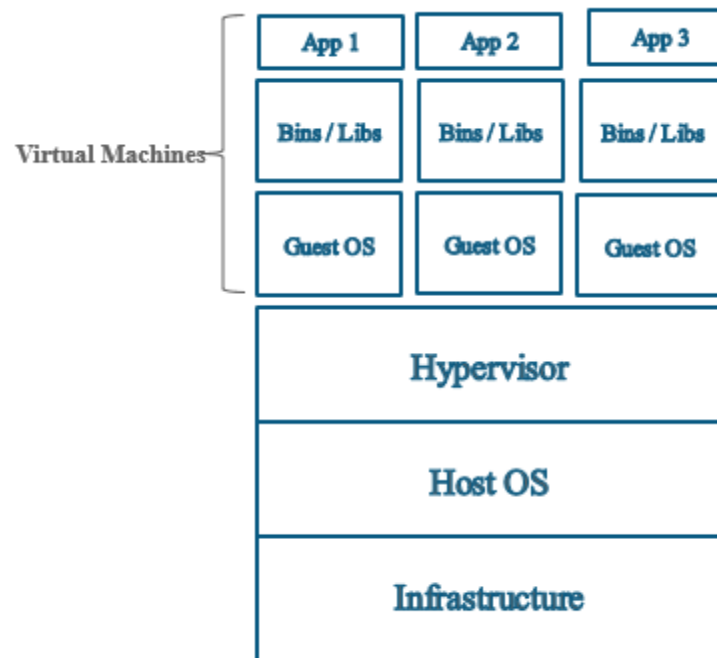


Containers

<https://geekflare.com/docker-vs-virtual-machine/>

Lightweight
virtualisation
approach

Trades off less
isolation for (much)
better performance



Docker

De-facto standard container technology

Docker **containers** are defined through **Dockerfiles** (scripts) to be executed by the Docker runtime environment against some base container when building the container

Dockerfile

```
1 FROM alpine
2 RUN apk add nodejs npm
3
4 WORKDIR /node_app
5 COPY ./ /node_app
6
7 RUN npm install express
8
9 EXPOSE 5000
10
11 CMD [ "node", "express.js" ]
```

Reference: <https://docs.docker.com/engine/reference/builder/>

Some Important Commands

FROM - defines the base image to derive the new image from
Should be the first command in the Dockerfile

COPY - copy files from the host computer to the image

VOLUME - mount directory from host computer to the image

WORKDIR - set the working directory of the image

RUN - execute a shell command (general-purpose “hammer”)

EXPOSE - declares that the image will listen at a certain port
Note: this **will not** automatically make the port externally available!

Some Important Commands

ENV - declare an environment variable in the container

(Remember how we said that ScalyShop uses env variables for configuration :) ?)

ARG - define that the container takes an argument at build time

Unlike ENV, these are **not** available in the final image, only at build time

CMD, ENTRYPOINT - what command to execute when starting the container

Every image needs at least one of the two

Should be the last command in the Dockerfile

See here for comparison: <https://docs.docker.com/engine/reference/builder/#entrypoint>

Building and Running

Building a docker container:

```
docker build -t dat490_1 example1
```

('dat490_1' is the name of the new docker image, example1 is a directory containing the dockerfile)

Show images in local registry:

```
docker images
```

Run container:

```
docker run dat490_1
```

Container Networking

Containers have their own private virtual network

You cannot “automatically” access ports exposed by a container from the host computer (or anywhere else)

Requires port mapping:

```
docker run -p 5000:5000 dat490_1
```

Container Networking

By default, each container runs in a separate network

If required to have multiple containers communicate, they need to be assigned an explicit virtual network at startup

```
docker run --name frontend --network scaly-net dat490_1
```

This container can then be addressed by the name “frontend” within the container network, but **not** from the host machine.

However, you **do not** need to use port mapping if you only want to access the container from other containers.

DockerHub

<https://hub.docker.com>



Central registry of pre-built docker containers

E.g., containers for web servers, databases, caches, middleware, etc.

DockerHub containers can be used as base images and extended (if necessary)

In the project we will not be using DockerHub (GitLab has its own registry)

Docker-Compose

Dockerfiles allow us to easily define individual services

However, many deployment architectures consist of multiple services that need to be connected (“service composition”)

Docker-compose is a simple way to define such an “**deployable architecture**” based on existing docker containers

- Comes pre-packaged with the docker runtime

Relatively simple YAML syntax

Docker-Compose - Basic Syntax

services:

<service1_name>:

image: <docker image name>

<image parameters ...>

<service2_name>:

image: <docker image name>

<image parameters ...>

volumes:

<volume definitions>

...

<https://docs.docker.com/compose/>

Building and Running a Composition

Starting all services in a composition

```
docker-compose up
```

(assumes a file `docker-compose.yaml` in the current working dir)

Stopping all services in a composition

```
docker-compose down
```

Dockerfile

```
1  FROM alpine
2  RUN apk add nodejs npm
3
4  WORKDIR /node_app
5  COPY ./ /node_app
6
7  RUN npm install express
8
9  EXPOSE 5000
10
11 CMD [ "node", "express.js" ]
```

Reference: <https://docs.docker.com/engine/reference/builder/>

Docker-Compose - Basic Syntax

services:

<service1_name>:

image: <docker image name>

<image parameters ...>

<service2_name>:

image: <docker image name>

<image parameters ...>

volumes:

<volume definitions>

...

<https://docs.docker.com/compose/>

Kubernetes

<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

Docker-compose allows you to easily start a composition **on a single computer**
... but that's not how we actually want to deploy our application in production

“Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.”



Basic Cloud Computing Tenets

Pay-as-you-go pricing

Customers pay per usage time unit, rather than a fixed price

Resource virtualisation

Customers rent “virtual” computing rather than physical assets

Elasticity

Customers can quickly change how many resources they are leasing

Multi-tenancy

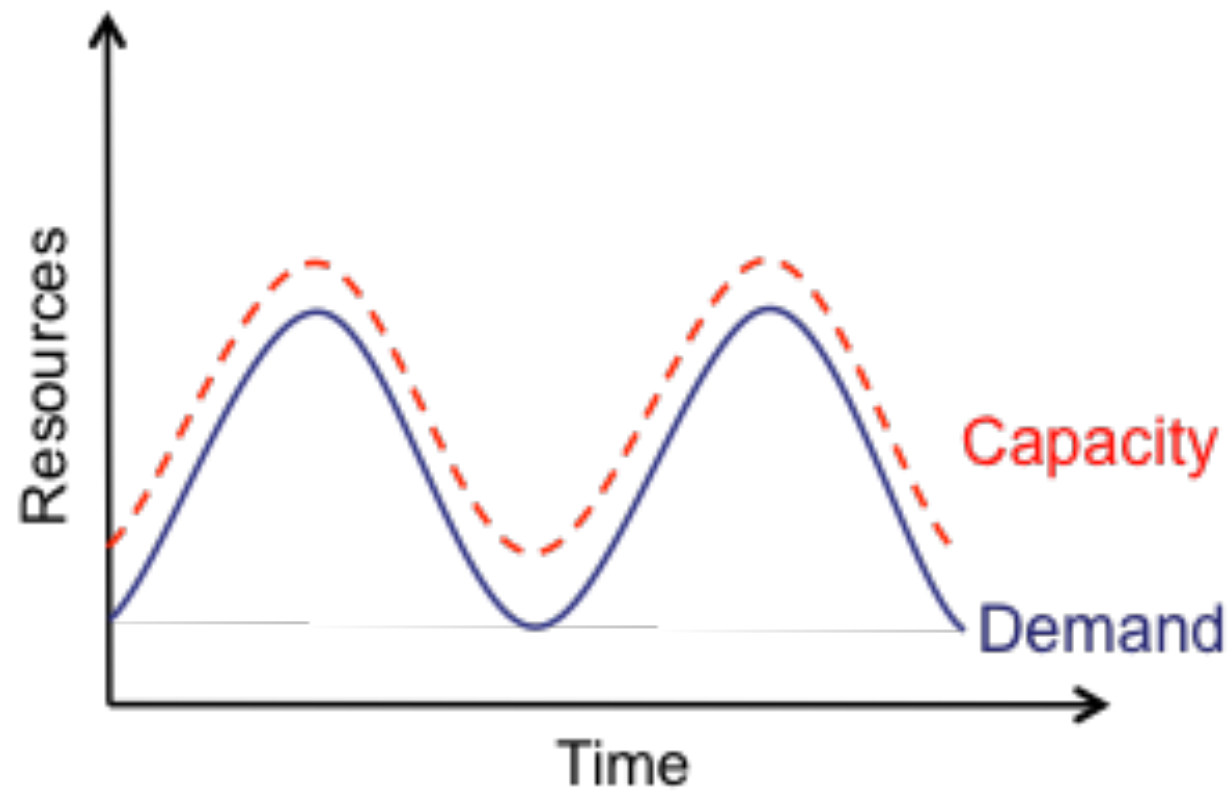
Physical hardware is shared between customers

Illusion of infinite resources

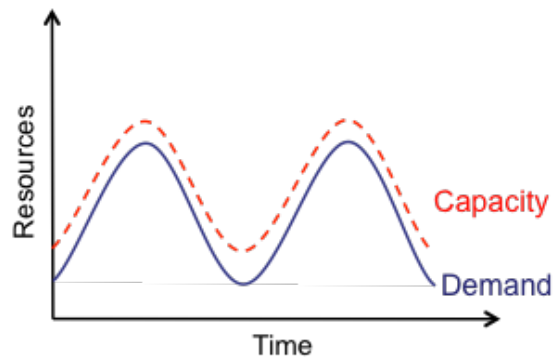
The cloud in practice does not “run out” of computing

Customers can scale up (virtually) indefinitely

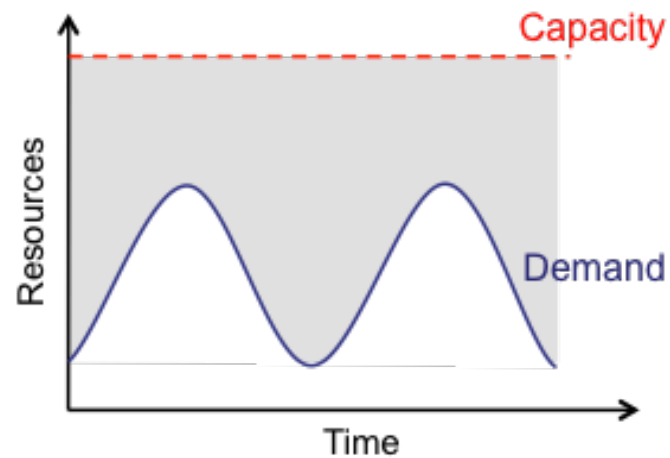
Elasticity




Elasticity

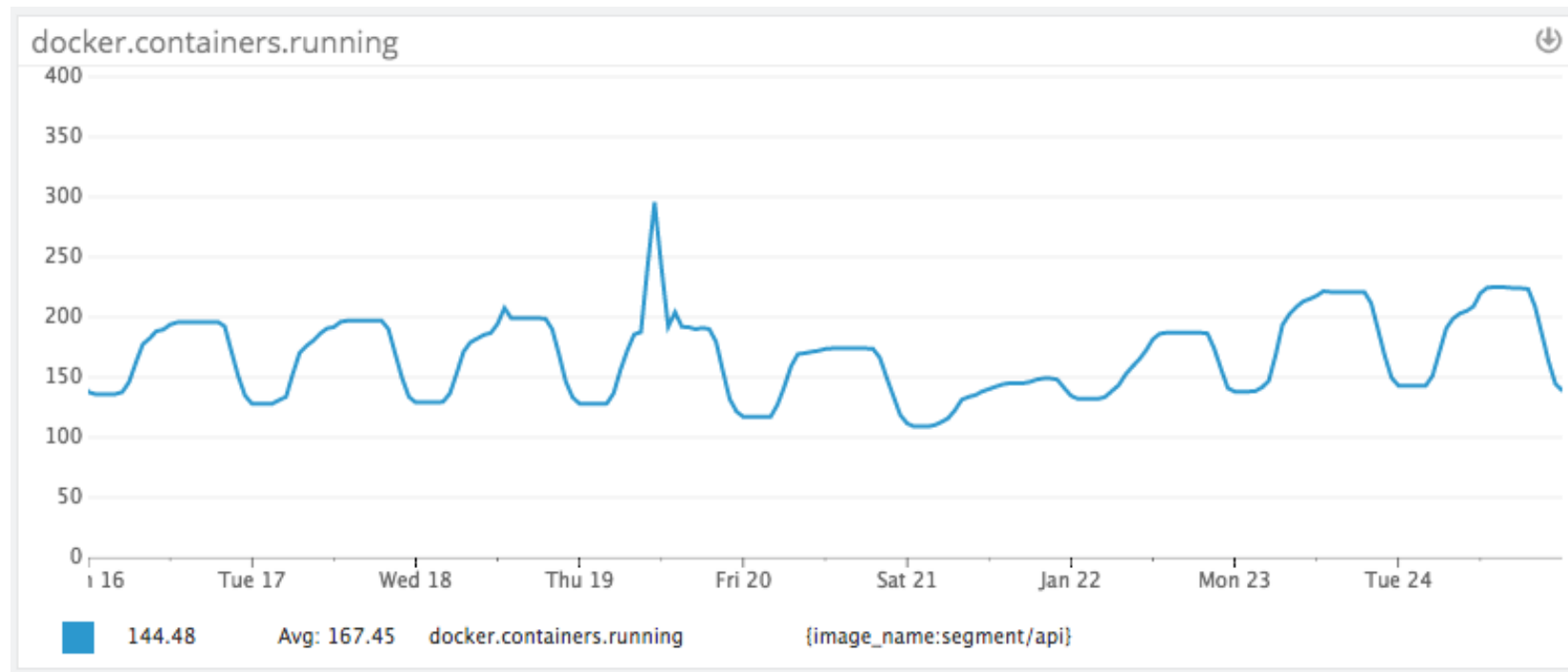


As opposed to:



 Unused resources

Real Example (<https://segment.com>)



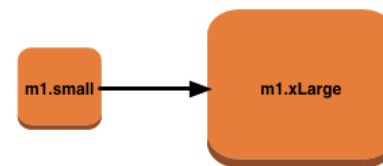
Source:
<https://segment.com/blog/the-million-dollar-eng-problem/>

Adjusting Resources

Elasticity assumes that we can “adjust” the amount of computing resources available to an application

Two different ways to do this:

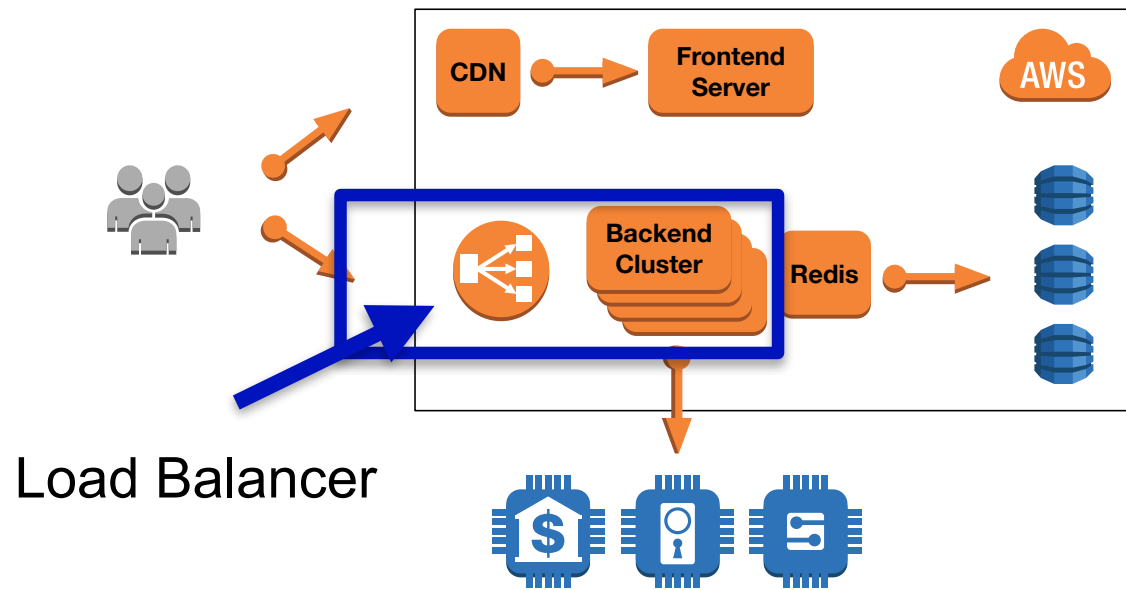
- **Scale up/down**



- **Scale out/in**



Load Balancing



Request Mapping and Scaling

Elastic load balancing requires to solve two problems:

1. How to select **which instance** to route each request to (scheduling, mapping, routing)
2. How to decide **when to increase / decrease** the number of available backend instances (scale out / in, auto-scaling)

Request Mapping

Common strategies for request mapping:

Round-robin

Random (~ the same as round-robin)

Load-based

+ client affinity (attempt to direct requests from the same customer to the same instances)

Auto-Scaling

Common strategies for auto-scaling:

None (fixed number of instances)

Threshold-based

(e.g., try to keep overall CPU utilisation within a certain interval)

Predictive

(estimate load in near future, increase / decrease in advance)

+ upper / lower bounds (very important in practice)

+ oscillation avoidance

Basic Cloud Computing Tenets

Pay-as-you-go pricing

Customers pay per usage time unit, rather than a fixed price

Resource virtualisation

Customers rent “virtual” computing rather than physical assets

Elasticity

Customers can quickly change how many resources they are leasing

Multi-tenancy

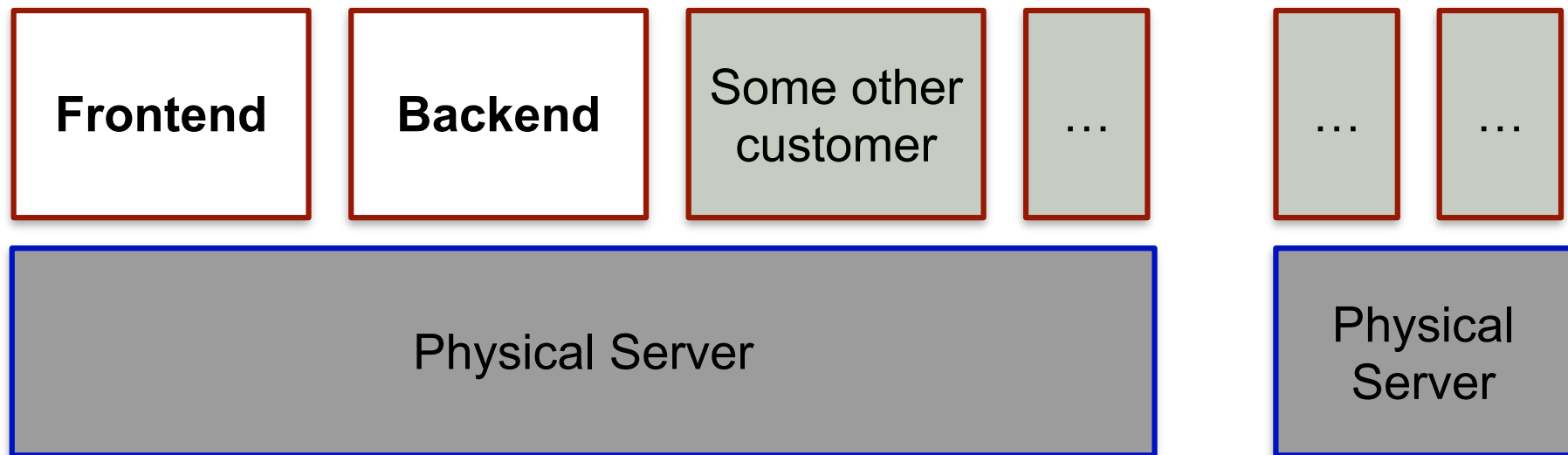
Physical hardware is shared between customers

Illusion of infinite resources

The cloud in practice does not “run out” of computing

Customers can scale up (virtually) indefinitely

Multi-Tenancy



Multi-Tenancy

- Physical resources are **shared** among many independent parties
 - *e.g., the physical host that runs your project also hosts virtual machines for Netflix*
- Fundamentally **how the cloud makes money**
 - Allows cloud to efficiently use its physical infrastructure and leverage **economies of scale**

Challenges of Multi-Tenancy

Noisy Neighbours

Other customers may be stealing your performance

Nosy Neighbours

Other customers may be stealing your data

(clearly cloud provider should prevent this, but side-channel attacks have been successful in the past)

Performance Variation

Instance Types Matrix

Instance Type	vCPU	Memory (GiB)	Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)	Intel® AES-NI	Intel® AVX†	Intel® Turbo	EBS OPT	Enhanced Networking
t2.micro	1	1	EBS Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
t2.small	1	2	EBS Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
t2.medium	2	4	EBS Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
m3.medium	1	3.75	1 x 4 SSD	Moderate	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	-	-
m3.large	2	7.5	1 x 32 SSD	Moderate	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	-	-
m3.xlarge	4	15	2 x 40 SSD	High	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	Yes	-
m3.2xlarge	8	30	2 x 80 SSD	High	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	Yes	-

Performance Variation

Instance Types Matrix

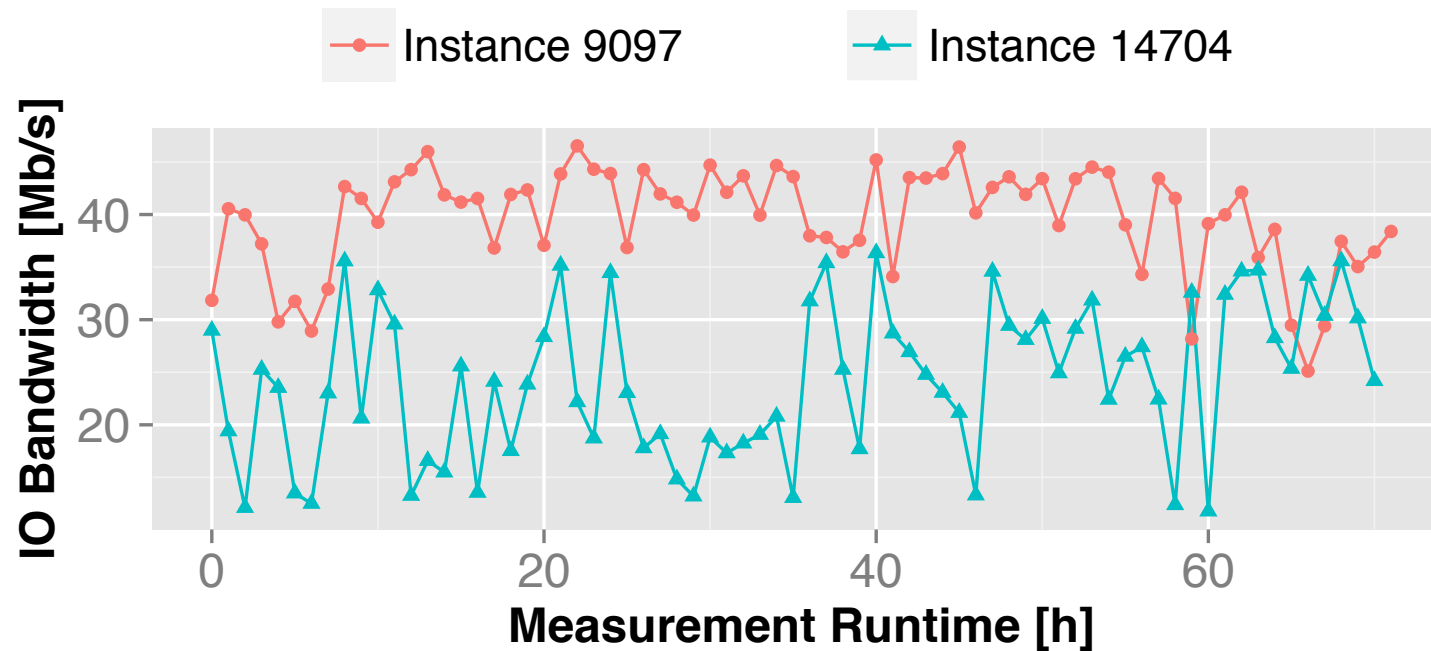
Instance Type	vCPU	Memory (GiB)	Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)	Intel® AES-NI	Intel® AVX†	Intel® Turbo	EBS OPT	Enhanced Networking
t2.micro	1	1	5 SSD Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
t2.small	1	2	5 SSD Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
t2.medium	2	4	5 SSD Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
m3.medium	1	3.75	1 x 4 SSD	Moderate	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	-	-
m3.large	2	7.5	1 x 32 SSD	Moderate	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	-	-
m3.xlarge	4	15	2 x 40 SSD	High	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	Yes	-
m3.2xlarge	8	30	2 x 80 SSD	High	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	Yes	-

Performance Variation

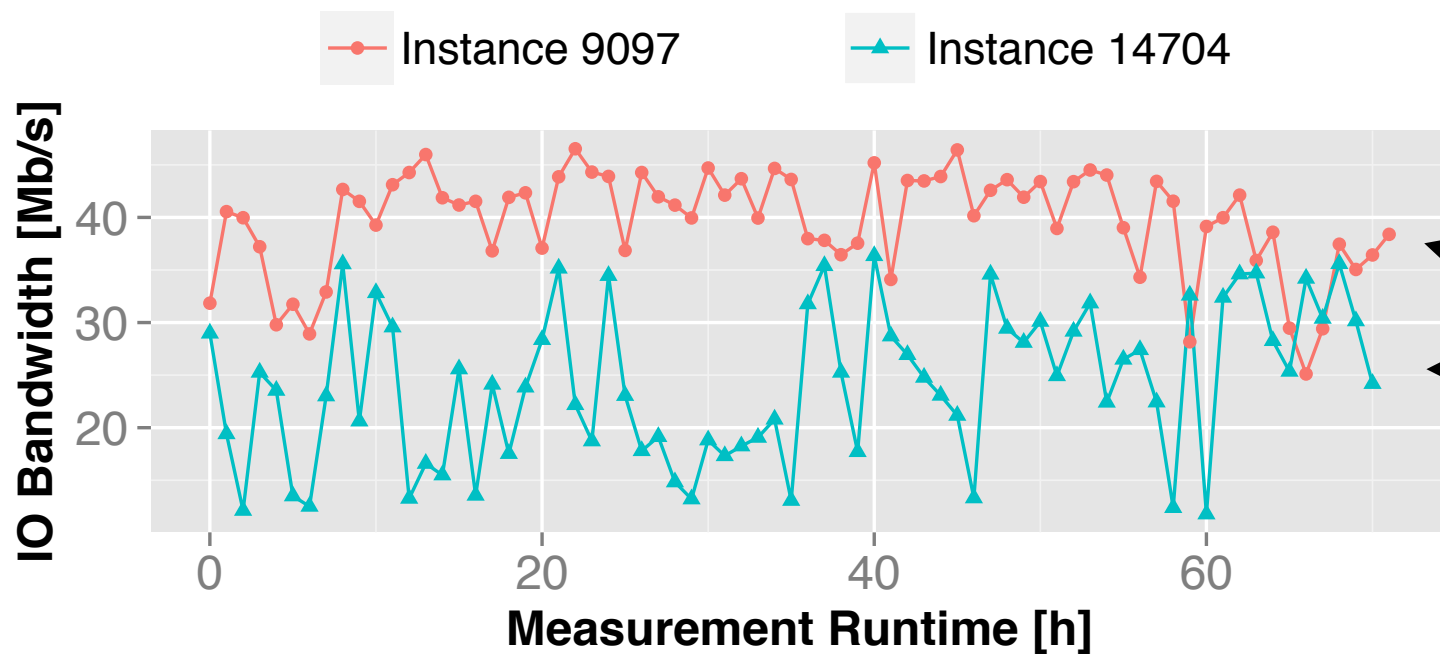
Instance Types Matrix

Instance Type	vCPU	Memory (GiB)	Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)	Intel® AES-NI	Intel® AVX†	Intel® Turbo	EBS OPT	Enhanced Networking
t2.micro	1	1	EBS Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
t2.small	1	2	EBS Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
t2.medium	2	4	EBS Only	Low to Moderate	Intel Xeon family	2.5	Yes	Yes	Yes	-	-
m3.medium	1	3.75	1 x 4 SSD	Moderate	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	-	-
m3.large	2	7.5	1 x 32 SSD	Moderate	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	-	-
m3.xlarge	4	15	2 x 40 SSD	High	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	Yes	-
m3.2xlarge	8	30	2 x 80 SSD	High	Intel Xeon E5-2670 v2*	2.5	Yes	Yes	Yes	Yes	-

Performance Variation

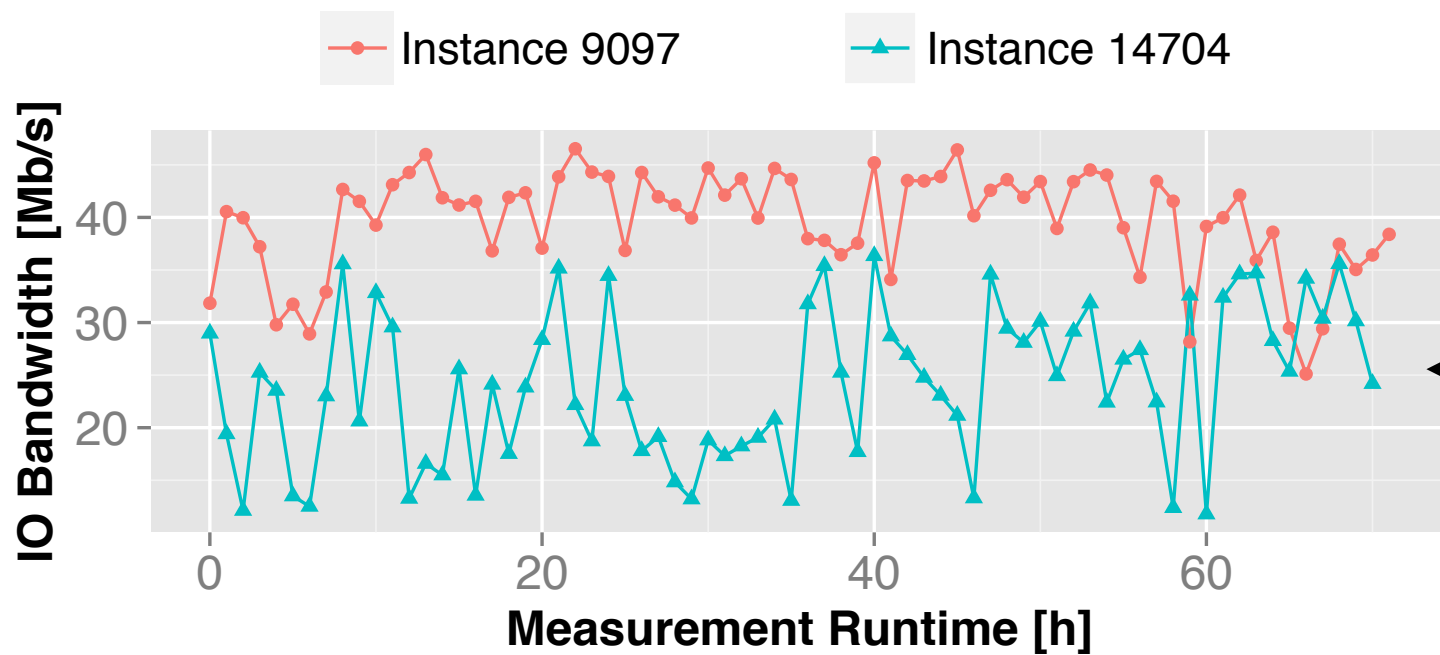


Performance Variation



Two identical instances
-
very different performance

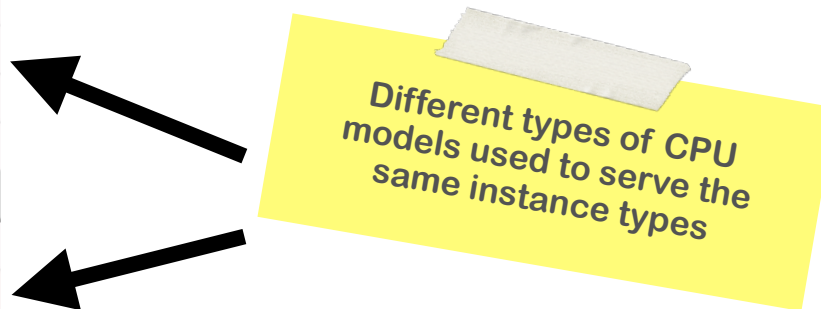
Performance Variation



Large differences over time

Hardware Heterogeneity

	Model	#
EC2	Intel E5-2650	1962
	Intel E5430	364
	Intel E5645	293
	Intel E5507	84
	Intel E5-2651	32
	AMD 2218 HE	9
Azure	AMD 4171 HE	782
	Intel E5-2673	595
	Intel E5-2660	189



Basic Cloud Computing Tenets

Pay-as-you-go pricing

Customers pay per usage time unit, rather than a fixed price

Resource virtualisation

Customers rent “virtual” computing rather than physical assets

Elasticity

Customers can quickly change how many resources they are leasing

Multi-tenancy

Physical hardware is shared between customers

Illusion of infinite resources

The cloud in practice does not “run out” of computing

Customers can scale up (virtually) indefinitely

Additional Reading

I started uploading some (non-mandatory) reading to Canvas:

For Lecture 1:

James Hamilton (Windows Live) - *On Designing and Deploying Internet-Scale Services*

For Lecture 3:

Jürgen Cito et al. - *The Making of Cloud Applications* (empirical research of how people design and build cloud apps)

Leitner and Cito - *Patterns in the Chaos* (lots of measurements of cloud variability)