ITI8801 · Cloud Architectures and DevOps

What cloud is, and why it exists

Siim Rebane · TalTech · Autumn 2026

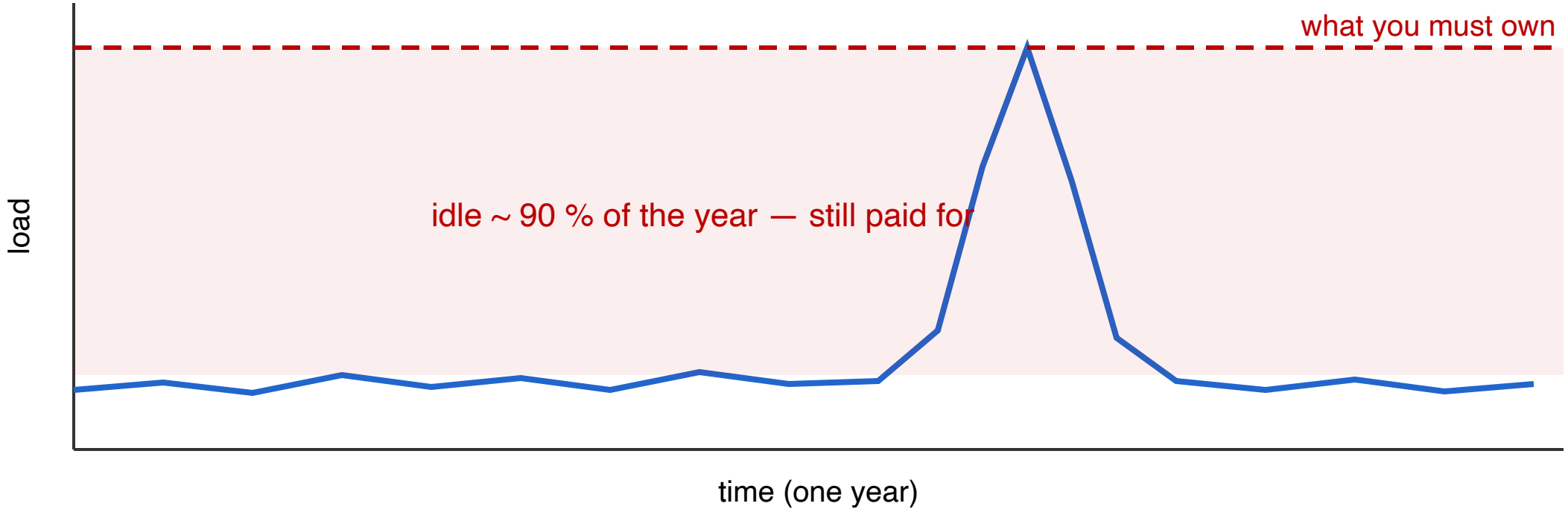
The baseline: you own the hardware

You have an application. Users use it. It runs on servers that **you** bought, racked, powered, cooled, patched, and replaced when a disk died.

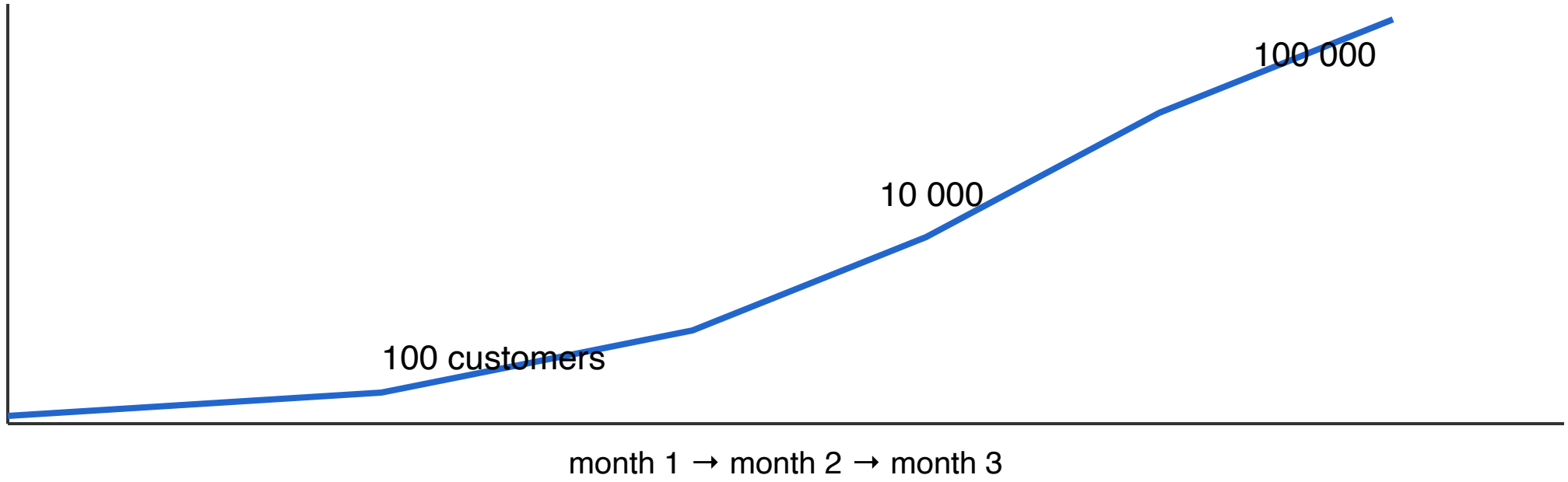
This worked for fifty years. It still works.

The question of this lecture: **what problem does it have, and what did we do about it?**

Problem 1: you have to buy for the peak



Problem 2: you have to become an expert in growth



Each step is a different job: new hardware, new network design, new database setup, new people who know how.

A company that needs this expertise **for one month a year** cannot keep it on staff.

The answer: someone who does this at scale sells it to you

Imagine a company whose *only* job is running hardware for a hundred thousand customers.

- They have the buying power — servers by the container
- They have the competency — the network and database people you can't hire
- They already run at a scale where your peak is noise

They sell it back to you **by the hour**, and you take exactly as much as you need, when you need it.

That is cloud.

Cloud, in one sentence

> Computing resources — machines, storage, networks, and services built on them — rented on demand, by the hour, through an API (application programming interface), from someone who runs them at scale.

Two words in that sentence do all the work:

- **on demand** — you add and remove capacity in minutes. Problems 1 and 2 disappear.
- **API** — you provision by code, not by a purchase order. This is what makes infrastructure *programmable* — infrastructure as code.

The formal version, so you know where the words come from

The US standards body NIST (National Institute of Standards and Technology) wrote the definition everyone quotes (SP 800-145, 2011 — two pages). Cloud is a service with five properties:

Property	Meaning for you
On-demand self-service	you click or call an API; no human on their side
Broad network access	reachable from anywhere over the internet
Resource pooling	your VM (virtual machine) shares a physical host with strangers
Rapid elasticity	capacity grows and shrinks with load — problem 1
Measured service	you pay per hour, per GB, per request — this is where your credits go

Same document also defines the three service models.

Service models are a scale of "what do I not want to manage"

Your hardware	IaaS (infrastructure as a service)	PaaS (platform as a service)	SaaS (software as a service)
you manage everything	they manage the machines	they manage the runtime	they manage the product

Not four boxes — a **spectrum**. Real systems sit at several points at once.

We'll walk along it, left to right, by asking one question each time:

"What is still my problem here?"

IaaS: infrastructure whenever I want it

Your hardware	**IaaS**	PaaS	SaaS

You get **virtual machines, disks, networks** — in minutes, by API.

What you gained: no purchasing, no racks, no peak planning.

What is still your problem: everything above the hardware. The OS (operating system). Its updates. The firewall rules. The database you install on it. Its backups.

This is where we start: today's assignment 1 is IaaS by hand — one VM on your own account. (Azure VM, EC2 (Amazon's virtual machines), OpenStack.)

What you're actually buying: a virtual machine

A **slice of a physical server**, carved out by a hypervisor, sold as a size (Azure *B2ats_v2*, AWS (Amazon Web Services) *t3.micro*):

You get	What it really is	Consequence
vCPU	a hardware thread on a shared host	you share the physical CPU (processor) with strangers — "noisy neighbours"
RAM (random-access memory)	dedicated memory	what you pay most for; the size ladder is mostly RAM
Disk	a network-attached volume	separate object, separate lifecycle: **stopping the VM doesn't stop paying for the disk**
NIC (network interface card)	a private IP address (Internet Protocol) in your network; public IP optional	the public IP is a separate, rented thing too

Two things about VMs that cost people money

1. Stopped ≠ free.

A deallocated VM stops the compute charge — but the **disk and the public IP keep billing**. Only *deleted* is free. This is why many assignments end with "delete the resource group", not "stop the VM".

2. It can vanish.

The provider can and will move, reboot, or retire your VM for maintenance — with notice if you're lucky. Design as if it could disappear tonight. That's why rebuilding it should be a single command — not a Saturday.

Burstable VMs: the peak problem again, inside one machine

The **B** in B2ats_v2 / the **t** in t3.micro = *burstable*: you pay for a **baseline** (say 10–20 % of a core), **earn credits** while idle, **spend them** on spikes and get the whole core.

Same trick as the whole lecture, one level down: pay for the average, borrow the peak.

Good for

- anything idle most of the time: web server, bastion, CI (continuous integration) runner between jobs
- **this course** — cheapest thing that works, free-tier on Azure

Bad for

- sustained load: credits run out → **throttled to baseline**, performance drops off a cliff
- databases, loaded Kubernetes nodes, benchmarks — "it got slow at 14:00 for no reason"

Try it: `stress` *your VM for 20 minutes and watch the CPU graph fall.*

But I didn't want a machine. I wanted a database.

You installed PostgreSQL on your IaaS VM. Now ask honestly:

- Does your **backup** actually restore? Did you test it? Will it still work in five years, after three OS upgrades?
- Who applies the **security patch** on a Saturday?
- Did you configure it **correctly** — given that running databases is *not your primary job*?

The provider runs a hundred thousand databases. Configuring them correctly *is* their primary job.

So: hand over the runtime, keep only your data and your code.

PaaS: I hand over the runtime and keep my code

Your hardware	IaaS	**PaaS**	SaaS

Managed databases, managed Kubernetes, app platforms. You give them **code and data**; they give you a running service with backups, patches, and scaling handled.

What you gained: a whole operations job you didn't want.

What is still your problem: your code, your data model, your costs — and you now pay more per month than the bare VM. That's the trade: **money for operational effort.**

Push further: I don't even want to think about size

Your hardware	IaaS	**PaaS → serverless**	SaaS

Still PaaS, but the last knob is gone: **no machine size, no instance count.**

"Here is what I need to run. Here is my credit card. Bill me for what it actually does."

- object storage + CDN (content delivery network): S3 (Amazon's object storage) + CloudFront, Azure Blob + Static Web Apps — *files, any amount*
- functions (Lambda, Azure Functions) — *code, any number of calls*
- serverless databases (DynamoDB, Cosmos DB) — *rows, any count*

Zero traffic → zero bill. A million requests → they scale, you pay.

Are you paying for provisioning, or for usage?

The same application, two ways to be billed:

Provisioned

a VM for the app, a VM for the database

You pay **per hour, for capacity** — whether anyone uses it or not. Idle at 03:00 costs the same as busy at 14:00.

Usage

static hosting + a serverless function + a serverless database

You pay **per request, per GB, per second of compute**. Nobody uses it → the bill is ~0.

Knowing which one you are on is the first question in every cloud architecture decision.

Cloud is not cheaper. It is elastic.

Compare a VM on **Hetzner** with the same VM on Azure or AWS, on price alone — the big cloud loses, badly. Same for a rack in your own basement.

What you pay the big cloud for is the **option to scale**: up in minutes, down to zero.

If you rent a fixed set of VMs and leave them running all year, you took none of that option — and you paid the premium anyway. That is the expensive way to use cloud, and it's the most common one.

SaaS: someone sells you the finished product

Your hardware	IaaS	PaaS	**SaaS**

GitHub, GitLab, Slack, Grafana Cloud. You don't run it, you don't deploy it — you log in.

Nothing to design here; you're a customer. It's on the scale only so you see the end of it.

Whose cloud: public, private, hybrid

Now that "cloud" means something, three ways to get one:

	Who runs the hardware	Example here	Trade
Public	a provider, shared with the world	AWS, Azure, GCP (Google Cloud Platform)	elastic, global, metered
Private	your hardware, **operated as a cloud** — self-service, API, pooling (plain servers don't count)	TalTech's OpenStack	your quota, your site, no invoice
Hybrid	public + private **used as one system**: networks connected, workloads placed by requirement — sensitive data private, spiky load public	—	owning both ≠ hybrid

A cloud is made of regions and zones

Region — a geographic cluster of data centres.
Availability zone (AZ) — one isolated data centre inside a region, with its own power, cooling, and network.

The big regions nearest to Tallinn:

Region	Where
Sweden Central	Stockholm
North Europe	Ireland
West Europe	Netherlands
Germany West Central	Frankfurt

Three reasons this matters:

Choosing a region is a technical, legal, and survival decision

Reason	Concretely
Latency	Tallinn → Stockholm ≈ 10 ms · Tallinn → Virginia ≈ 120 ms. Put it near your **users**, not near you.
Data residency	GDPR (EU data-protection law): some data may not leave the EU. The region is a legal choice.
Failure domains	an app in one zone dies with that zone. Two zones = first step of high availability. Two regions = second step, much more money.

Zones are a large part of what you pay a public cloud for.

Shared responsibility: the provider secures the cloud, you secure what's in it

Theirs: buildings, hardware, hypervisor, network fabric.
Yours: OS patches, firewall rules, identities, secrets, data.

The line moves with the scale from earlier — IaaS: most is yours. SaaS: almost none.

Three ways people actually get burned, every year, including students:

1. Firewall open to `0.0.0.0/0` on SSH (secure shell — remote login) or the database port
2. **Secrets committed to Git** → credits stolen by crypto-miners within hours
3. **Over-privileged identities** — "admin" for a CI job that only pushes an image

Tools change. Concepts don't.

Every provider has the same ideas under different names:

Concept	AWS	Azure	OpenStack
firewall per VM	security group	NSG (network security group — Azure's firewall rules)	security group
identity for a job	IAM (identity and access management) role	managed identity	application credential
public address	elastic IP	public IP	floating IP
virtual network	VPC (virtual private cloud)	VNet (virtual network)	network + router

Learn the concept, look up the button. The exam tests the concept.

How anything on the internet gets reached: address, port, name

Three things stand between a browser and your VM:

	What	In assignment 1
IP address	where the machine is. **Private** (the reserved ranges — next slide) = inside your network only. **Public** = routable from the internet — rented from the provider, like the machine	the VM gets one private IP + one public IP
Port	which program on that machine. 22 = SSH, 80 = HTTP (the web protocol), 443 = HTTPS (encrypted HTTP), 5432 = Postgres	you open **80 and 443** to the world, 22 to yourself
Name	humans don't type IPs. **DNS (Domain Name System)** maps a name to an IP	you create an **A record** — a name you control, pointing at your public IP

Everything else — load balancers, CDNs — is layered on top of these three.

Private vs public, in numbers

Three address ranges are **reserved for local networks** and never routed on the internet:

Range	Where you've seen it
`10.0.0.0 – 10.255.255.255`	offices, and most cloud networks by default
`172.16.0.0 – 172.31.255.255`	Docker, some clouds
`192.168.0.0 – 192.168.255.255`	your home router

- Your laptop has a private address; your home router hides it behind **one public IP** — that's **NAT (network address translation)**. Ten devices, one address the world sees.
- Your cloud VM is the same picture: a **private IP** inside your network (`10.0.1.4`) and, only if you ask for it, a **public IP** the provider rents you.
- Rule of thumb: anything starting `10.` , `172.16–31.` , `192.168.` is *inside*. Everything else is *the world*.

How a name becomes an address

- You type `hello.example.com` → your machine asks DNS → gets `20.31.4.7` → connects there on port 80.
- Names are **hierarchical and delegated**: `.com` → `example.com` → whoever owns it decides `hello.`
- Both clouds hand you a name for free, under *their* domain (Azure: `<label>.<region>.cloudapp.azure.com`, AWS: `ec2-….compute.amazonaws.com`) — but you have almost no control over it, and you don't want it for anything production-ready.
- A name of **your own** is just a DNS **A record** you create, pointing at that same IP — that's what the assignment uses.

HTTPS: encrypted, authenticated HTTP

HTTPS is HTTP inside **TLS (transport layer security)**. It gives you two things:

- **Encryption** — nobody between the browser and the server can read or alter the traffic (not the café Wi-Fi, not the ISP).
- **Identity** — the browser knows it is talking to the real `<name>`, not something pretending to be it.

To set it up you need three things:

1. a **name** — an A record pointing at your server (certificates are issued for names, never for bare IPs)
2. a **certificate** for that name from a **CA (certificate authority)** — **Let's Encrypt** issues them free; **certbot** automates it: proves to the CA that you control the name, installs the certificate into nginx, renews it every 90 days
3. port **443** open

How the browser decides to trust a certificate

Your OS and browser ship with a list of certificate authorities they trust. When you open `https://<name>`:

1. The server presents its certificate.
2. The browser checks: **signed by a trusted CA?** (the chain of trust) · **issued for this exact name?** · **not expired?**
3. All three pass → padlock. Any fail → the full-page warning.

That's the whole trick: trust in a name, delegated through a chain of signatures — the same delegation idea as DNS itself.

A **self-signed** certificate encrypts fine but has no trusted signature — browsers warn, users leave.

Assignment 1, in today's vocabulary

`GET https://<your-domain>/hello` → `{"student":"<your student code>"}`

You set up, **by hand**:

- a **VM** — compute, the burstable kind
- a **firewall** — ports 80 and 443 open to the world, 22 to yourself
- **nginx** answering `/hello` — the software on the machine is your problem: that's IaaS
- a **DNS A record you create** — a name of your own pointing at your public IP
- **HTTPS** — a certificate, bound to that name

2 points. Due 13.09.2026 23:59, on Moodle. Step-by-step slides are in the practice session.

Reading: nothing required. Optional primary sources are linked in the course group.