ITI8801 · Cloud Architectures and DevOps

The team project

Siim Rebane · TalTech · Autumn 2026

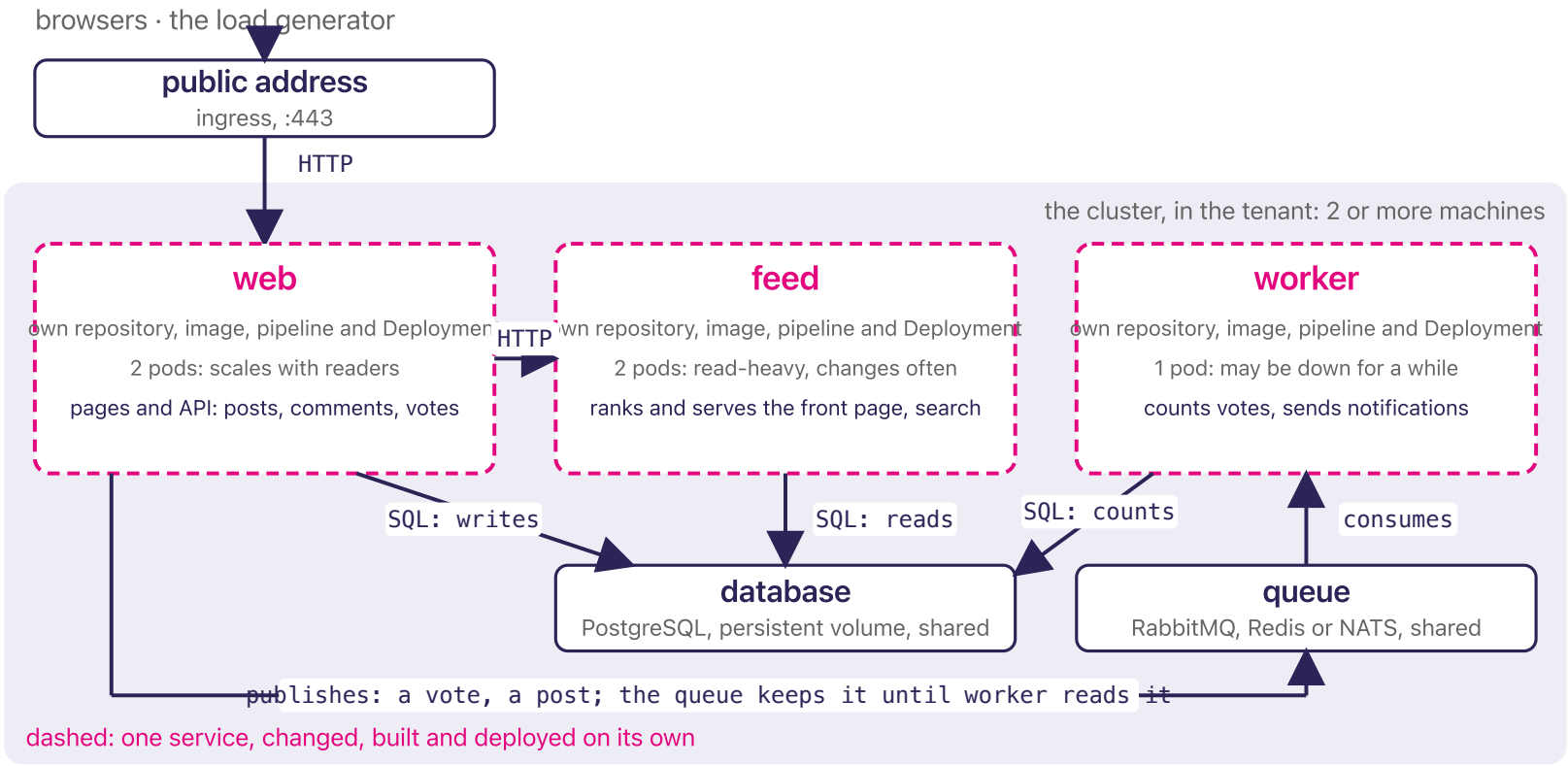
The team project

- Optional. 20 points on top of the 80 of the assignments and the exam
- Teams of 3–4, registered in Moodle. A registered team gets a tenant in Waldur (minu.etais.ee): 8 vCPU, 16 GB RAM, 160 GB disk, 2 public addresses. No other cloud
- The team develops and runs a web application of its own choice on TalTech's cloud until its demo, and shows it working under load, through a deploy, and while its machines go down

The application

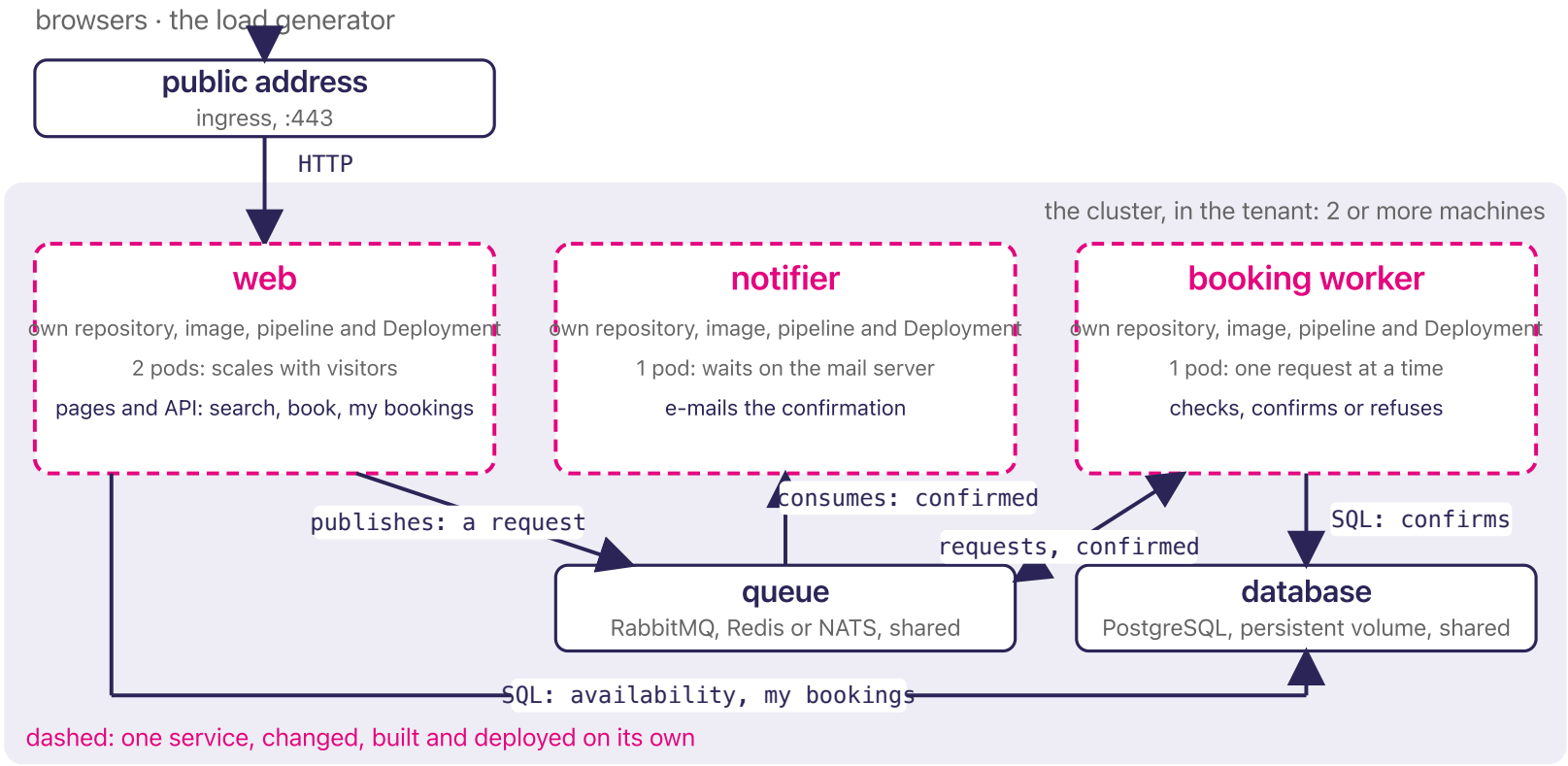
- Yours to choose: a forum like Reddit (posts, comments, votes), a shop, a booking system, a chat. A simple UI is fine; the points are not on it
- Split into **services**: separate programs, each with its own repository, pipeline and deployment, changed and deployed one at a time. At least three; the data is in a **database** on persistent storage. Two services talking through a **queue** instead of a direct call is preferred, not required; both samples do it. All on a Kubernetes cluster the team installs in its tenant; k3s is fine, the course cluster is not used
- A public HTTP API that takes writes. A write that was accepted is never lost and never applied twice
- A **load generator** of the team's own, run at the demo: it writes and reads at a rate the team states

Sample: a forum



- Why three: `web` must answer at once and scale with readers. `worker` does slow work later; while it is down the queue keeps the votes, and it counts each once when it is back. `feed` changes often and is deployed without touching `web`

Sample: a booking system



- Why three: `web` answers at once. `booking worker` takes one request at a time, so two people cannot book the same room; the queue keeps requests while it restarts. `notifier` waits on a mail server; its slowness never holds a booking up

Working together

- The work is shared. The team decides who does what and writes it down as it goes: a **work log** in the repository, one line per piece of work, who did it, when. The git history has to match it
- Any member may be asked about any part of the system at the demo. The questions go to members at random, not to the author
- No member is the only one who can do something: every deploy, restore and access change has been done by at least two different members before the demo. The work log shows it
- A member who cannot explain the system, or show their own work in the log and the git history, gets half of the team's points. LLM use is allowed; the questioning shows whether the team understands its own system

What the system has to do

- **Stays up**: it answers and takes writes all the time. Through a deploy, also one that changes the database schema. While any one of its machines is hard-rebooted. While its database restarts
- **Loses nothing**: a write that was accepted is never lost and never applied twice, through all of the above
- **Can be rebuilt**: the tenant, the machines, the cluster and every service come from the repositories, through pipelines, not by hand
- **Can be seen**: the team knows whether it works without being told. A dashboard per service with requests, errors, latency and pods; an SLO the team states, measured on it; an alert that reaches a human when writes stop. The logs of every service go to one place; one request is followed through all the services by its id; one alert comes from the logs, not from a metric
- **Survives losing its data**: a backup elsewhere on a schedule; a restore shown at the demo
- **Uses little**: it asks for only what it needs. Machines, disk and public addresses sized to the load the team states; each public address has a reason. One for the application is the usual case, a second for administration; more if the application needs it
- **Is closed**: nothing is reachable from the internet that need not be. SSH, the cluster API and Grafana only from TalTech, the VPN and the members' homes. Every member has their own access. No secret in git

How each is achieved is the team's choice. At the demo the team shows each one working and explains how.

The processes

Five short documents in the repository, each shown working at the demo:

Process	Says
developer path	clone, run a service locally, push, the pipeline, where the change lands first
deploy	how a change reaches production, who may, what is checked, how it is rolled back
on-call	who gets an alert, where, what they do first
restore	how the database is restored from the last backup, and how long it takes
access	how a new member gets into the tenant, the cluster and the repositories

The demo

Twenty minutes in a practice session, 18 December at the latest, slot booked in Moodle; the processes, post-mortem and cost report are in the repository before it. The code review of another team's repositories is in the week 15 practice. The system is running and the load generator is on; the parts are split among the members as the team chooses:

1. **What we built**: the architecture, what is public and why
2. **How a developer works**: clone, run locally, push, pipeline, where the change lands
3. **A deploy**, live, under load. The question: how does a change to the database schema go through?
4. **A machine hard-rebooted and the database restarted**, under load. The question: where could a write be lost, and why is it not?
5. **How we monitor it**: dashboard, SLO, the alerts that fired, one request followed through the services in the logs
6. **A restore** from the last backup
7. **What it cost**: asked for against used, the right-sizing
8. **How we worked**: the work log, who did what, how the work was shared

Then questions to any member on any part.



Questions