Mikroteenused & konteinerite halduse platvormid

Pelle Jakovits

April 2025, Tartu

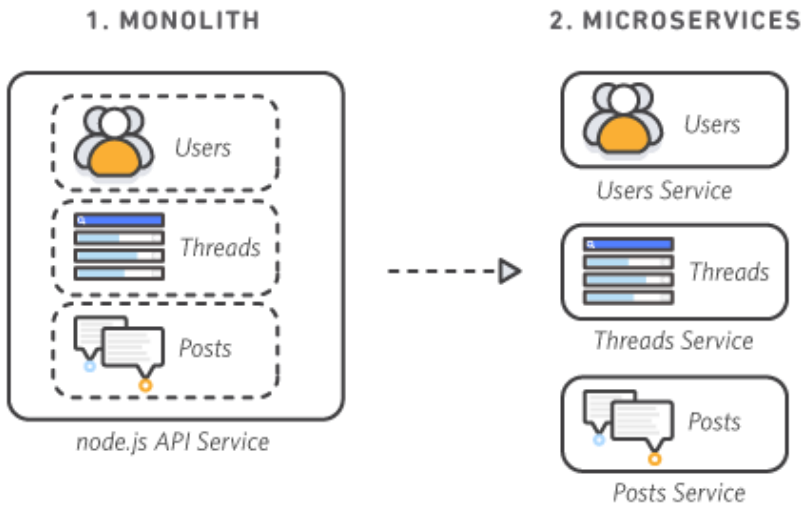
Sisukord

- Mikroteenused
  - Mikroteenuste omadused
- Konteinerite halduse platvormid
  - Konteinerite roll mikroteenuste loomisel
  - Docker & Docker Swarm

MIKROTEENUSED

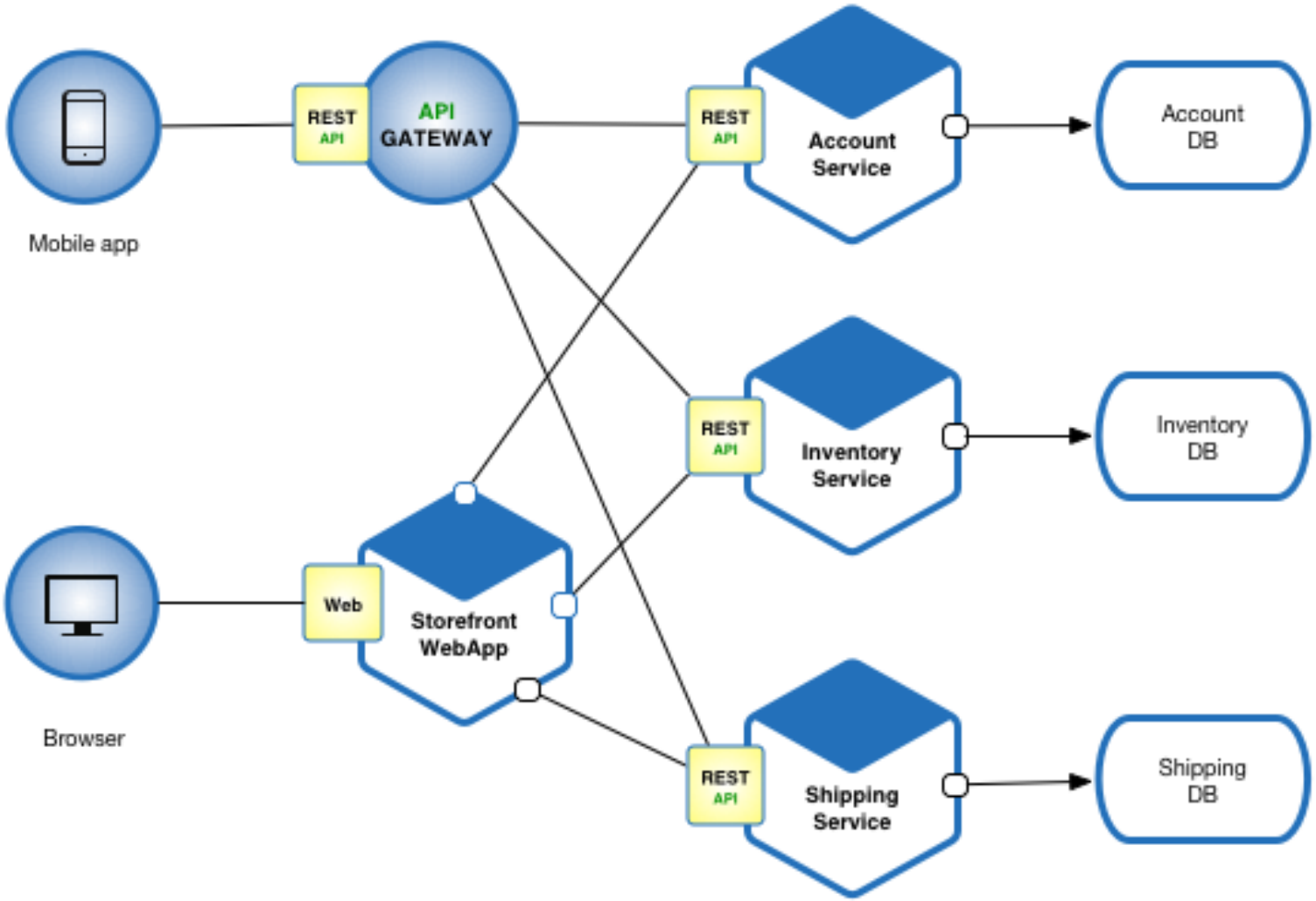
Mikroteenused

- Hajusüsteemide ahritektuuri muster
- Süsteem koosneb väikestest autonoomsetest ja spetsialiseerunud teenustest, mis omavahel suhtlevad standardsete API'de ja protokollide kaudu
- **Autonoomsus** - iga mikroteenus jookseb eraldi rakendusena, eraldi keskkonnas
- **Spetsialiseerimine –** iga mikroteenus haldab mingit kindlat süsteemi funktsionaalust (nt sisse logimist)
- **Standardite kasutamine**
  - HTTP, REST, SOAP, RPC, AMQP
- Hästi määratud APIde kasutamine
  - OpenAPI, WSDL, ..
- Teenusele orienteeritud arhitektuuri (SOA) variant

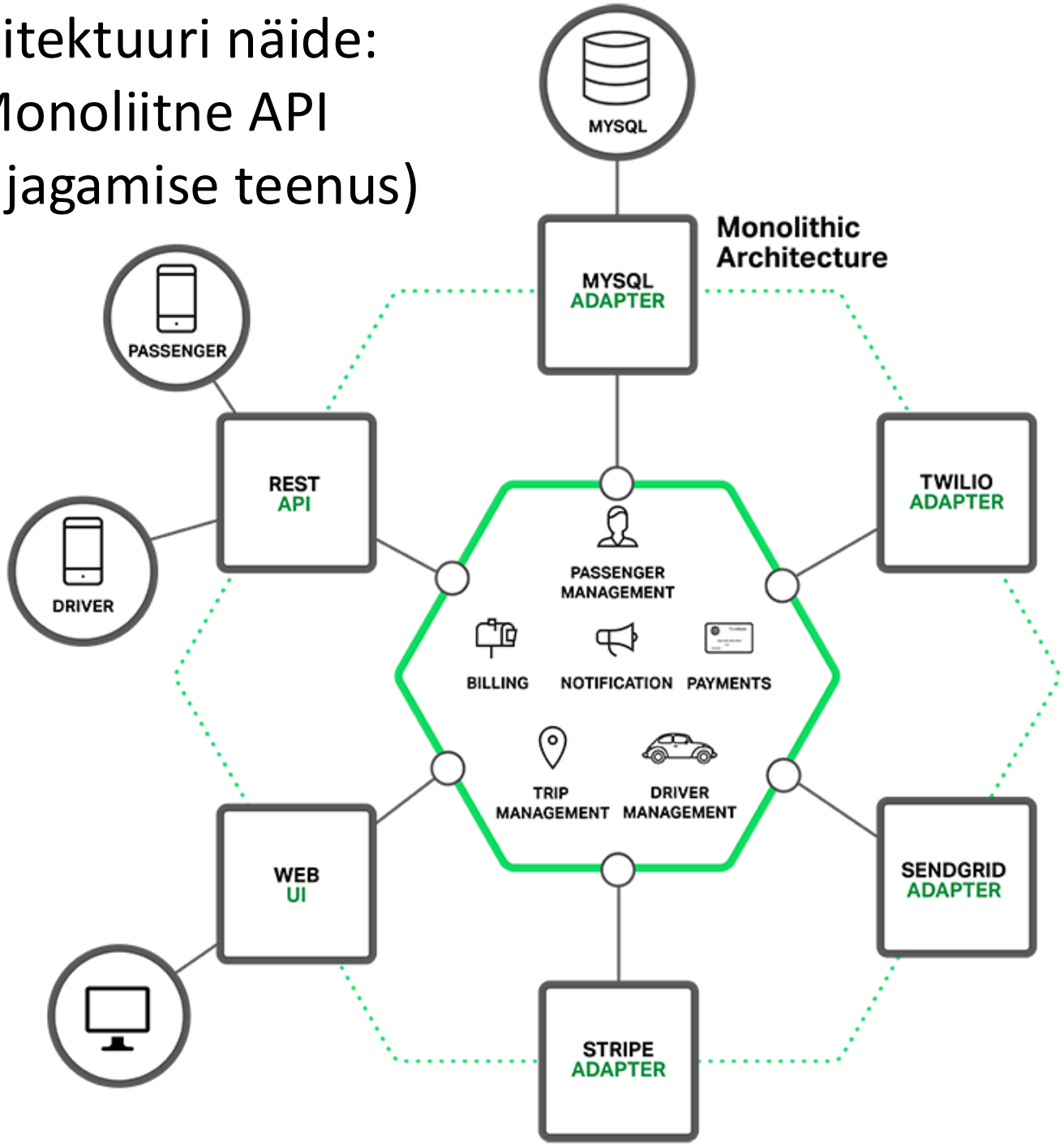


https://aws.amazon.com/microservices/

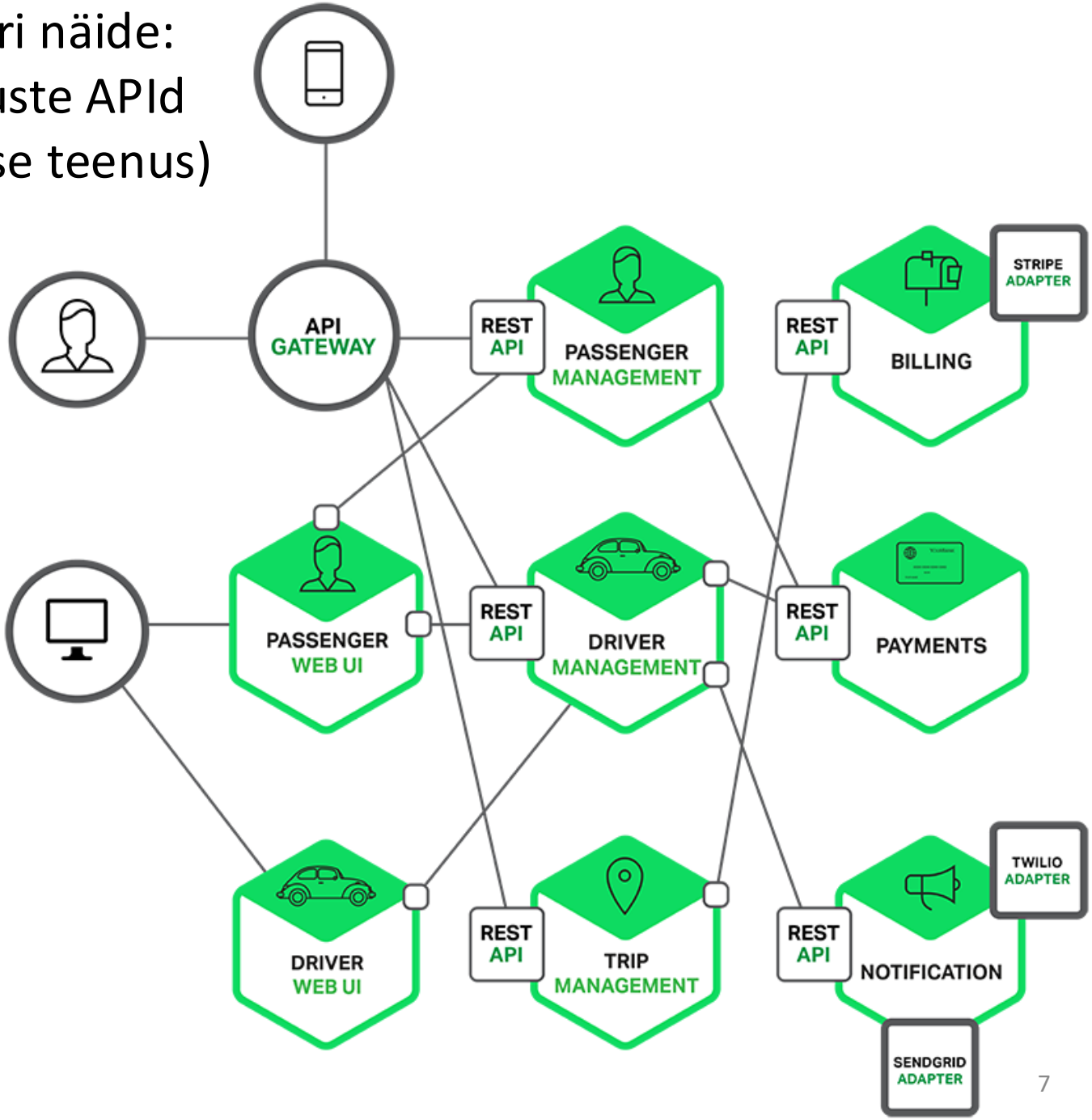
Mikroteenusete arhitektuuri näide: Hajutatud API'd



Arhitektuuri näide: Monoliitne API (Reisijagamise teenus)



Arhitektuuri näide: Mikroteenuste APId (Reisijagamise teenus)



Mikroteenuste omadused

- **Agiilsus** – Mikroteenuseid peab olema lihtne disainida ja implementeerida
  - Väikseid autonoomseid üksusi on lihtsam ümber disainida
- **Skaleeritavus** - Mikroteenust saab individuaalselt skaleerida
- **Lihtne juurutamine** – kiire üles seada ja uuendada
- **Tehnoloogiline vabadus** - Iga mikroteenus võib olla implementeeritud erinevas keeles, joosta erinevas keskkonas
- **Korduvkasutatav kood** – Funktsionaalsuste implementeerimine väikeste üksustena aitab vältida duplitseeritud koodi.
- **Vastupidavus** - ühe mikroteenuse tõrked ei mõjuta otseselt teiste teenuste jooksmist

- Konteinerid on üks peamisi aluseid mikroteenuse musteri edukuses

Kuna kasutada mikroteenuseid

- Alguses võib olla lihtsam luua monoliitne rakendus
  - Kui meeskonna suurus on väike
  - Alguses võib see arendust aeglustada
- Aga hiljem on seda tihti palju raskem edasi arendada ja skaleerida
  - Tugevad seosed erinevate moodulite vahel
- Väikestest iseseisvatest üksustest koosnevat süsteemi on lihne "tükkhaaval" edasi arendada, hajutada, skaleerida

Monoliitsete rakenduste jagamine mikroteenusteks

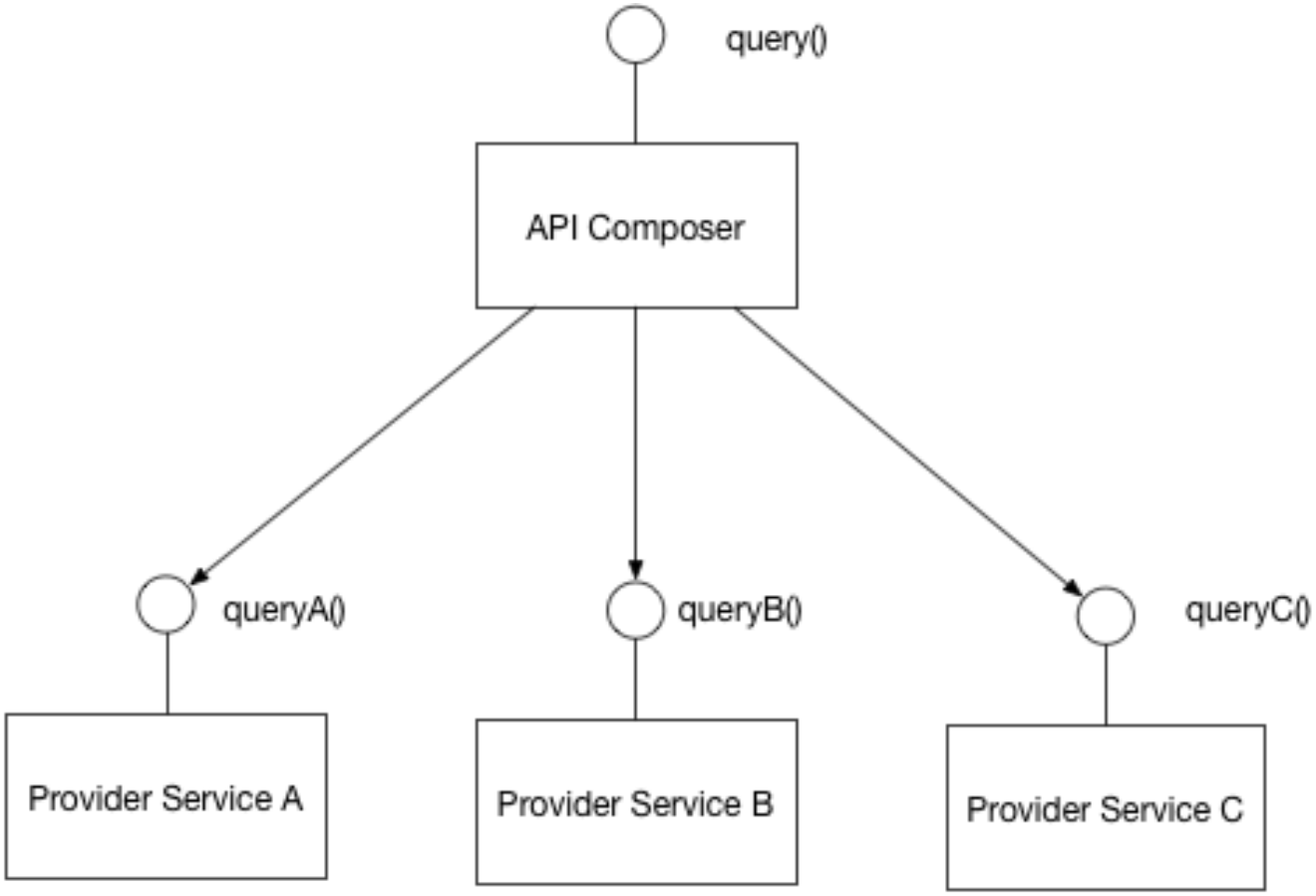
- Jagame äriloogika järgi alamteenusteks
- Jagame alamdomeeni järgi
- Implementeetrimine iga iseseisva teenuse mikroteenusena
- Jagame meeskondade vastutusala kaupa

- Ideaalselt võiks igal teenusel olla väike hulk vastutusala
  - Sarnaselt nagu Linuksis on iga käsurea käsk eraldi seisev väike programm (ls, grep, rm)

Mikroteenuste koostöö

- Kas kasutada mikroteenuste vahel otsesuhtlust või teadetejärjekordasid?
- Kuidas implementeerida kerulisemaid operatsioone, mis kasutavad erinevate mikroteenuste alamfunktsionaalsusi?
  - Komposeerida mikroteenuseid
- Kas kasutada üht keskset andmebaasi või teha igale teenusele erinev andmebaas?

API'de komposeerimine

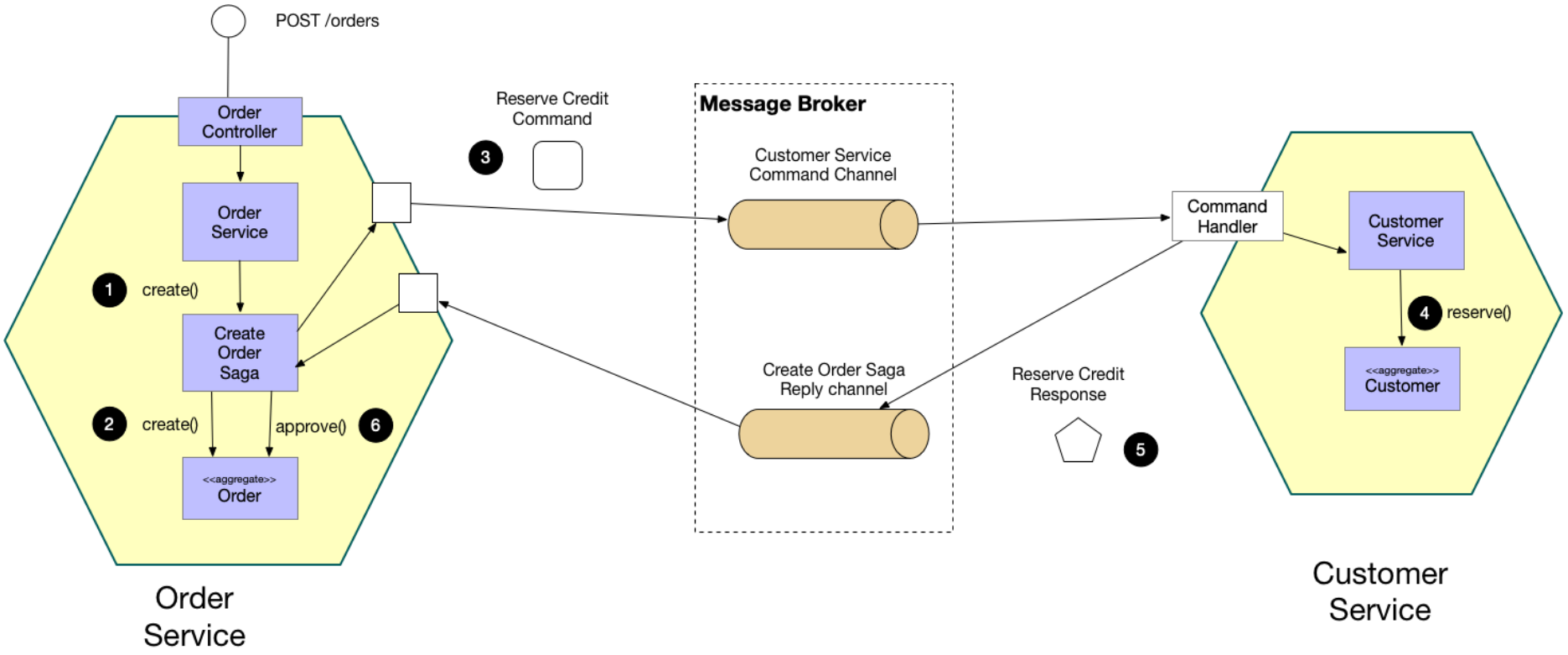


https://microservices.io/patterns/data/api-composition.html

Mikroteenuste andmebaasid

- Eraldi andmebaas
  - Tagab, et teenused on üksteisest sõltumatud
  - Muudatused ühe teenuse andmebaasis (nt sisemises struktuuris) ei mõjuta teise teenuse tööd
  - Iga teenus saab kasutada tema töö jaoks sobivat andmebaasi (SQL, Key-Value, Document, etc.)
- Jagatud andmebaas
  - Lihtsam implementeerida transaktsioone, mis peavad muutma mitme mikroteenuse halduses olevaid andmeid
  - Nõuab vähem suhtlust mikroteenuste vahel

Mikroteenuste koostöö näide



https://microservices.io/patterns/data/saga.html

Mikroteenuste kasutuselevõtu võimalikud kasud

- Lihtsasti **hooldatav ja testitav** – võimaldab sagedasemat ja pidevamat arendust ja juurutamist
- Lõdvalt ühendatud teiste teenustega – võimaldab meeskonnal töötada **oma teenuste** kallal iseseisvalt, ilma et teised teenused mõjutaks neid või nende muudatused mõjutaks teiste meeskonadade teenuseid
- **Iseseisvalt juurutatav** – võimaldab meeskonnal oma teenust uuendada ilma teiste meeskondadega kooskõlastamata
- Võimalik arendada v**äikese meeskonna** poolt – oluline kõrge tootlikkuse jaoks, vältides suuremat suhtluse jaoks vaja minevad aja kulu

Mikroteenuste eelised

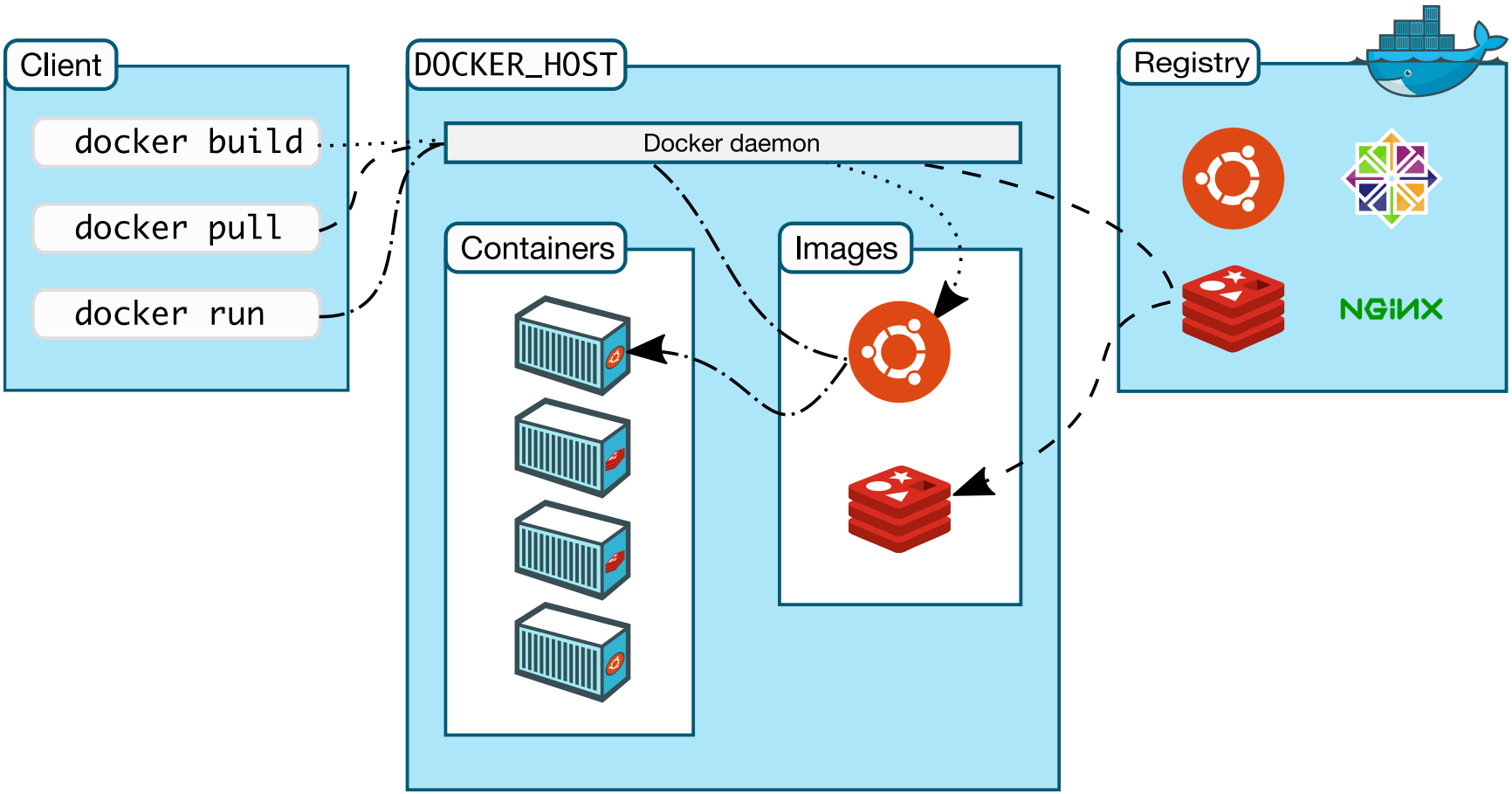
- Võimaldab suurte rakenduste pidevat tarnimist ja juurutamist (CI/CD)
  - Parem hooldatavus
  - Parem testitatavus
  - Kiirem juurutatavus
- Iga teenus on "piisavalt" väike
  - Lihtsam aru saada, uutele töötajatele tutvustada
  - IDE'sse ei pea importima ülisuuri projekte
  - Teenus alustab jooksmist kiiremini
- Parem isoleeritavus vigade korral
- Vähendab tehnoloogia võlga, sõltuvust tehnoloogia valikutest

Mikroteenuste puudused

- Raskem aru saada kogu süsteemi hajutatud arhitektuurist
  - Vaja tegeleda teenuste vahelise suhtlusega
  - Päringud, mis vajavad mitme teenuse välja kutsumist, või andmeid, on keerulisemad arendada ja testida
  - Teenuste vaheliste interaktsioonide testimine on keerulisem
- Kogu süsteemi korraga juurutada, üles seada on keerulisem
- Suurem resursside (eriti mälu) kasutus, kui igal teenusel on oma (mitte jagatud) keskkond

DOCKER'I ARHITEKTUUR

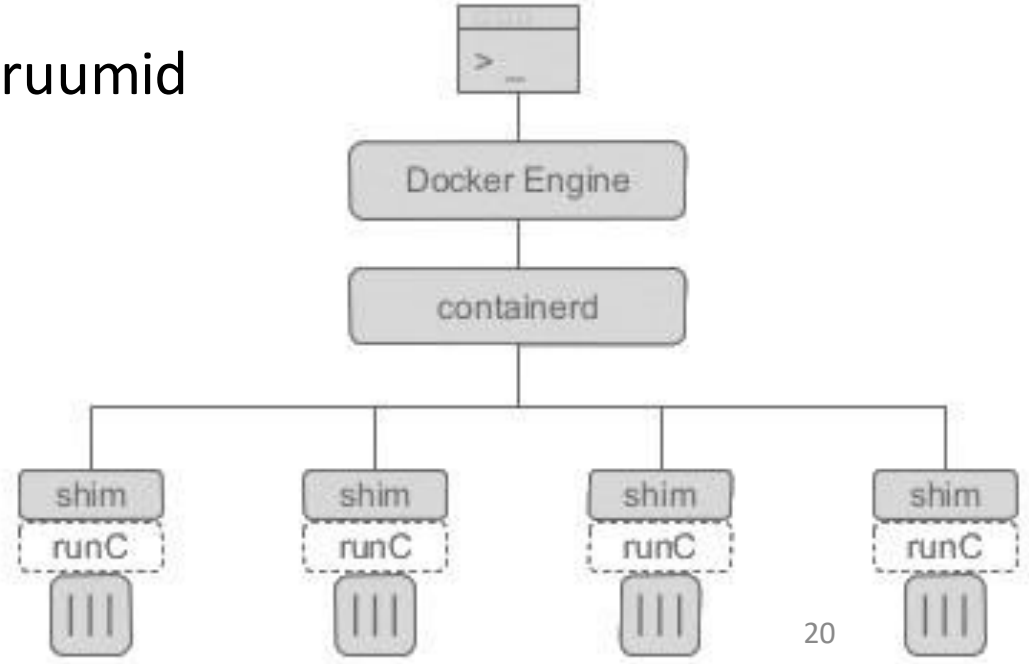
Docker sõlme arhitektuur



https://docs.docker.com/get-started/overview/

Dockeri Arhitektuur

- **Dockerd** – pakub liidest konteinerite halduseks
- **Containerd** - Võimaldab käivitada konteinereid, virtuaalseid võrke
- **Containerd-shim** - vahendab andmeid jooksva konteineri ja docker liidese vahel (logid, käskude käivitamine "konteineri sees", stderr, stdout)
- **RunC** – valmistab ette nimeruumid käivitab konteineri sees jooksvad protsessid
- **Konteiner** – Koosneb Linuksi protsessidest



Konteinerite omadused

- Muutumatus (Immutability)
- Ühekordseks kasutamiseks (Disposability)
  - Peaks toetama graatsilist väljalülitamist (graceful shutdown)
- Paralleelsus (Concurrency)
- Konteinerid peavad olema kohaliku mäluta (Stateless)
  - Ükski konteineris töötav rakendus ei tohiks sõltuda konteinerisse salvestatud andmetest
  - Konteinereid saab igal ajal migreerida, sulgeda, skaleerida ja välja vahetada

Konteinerite eelised

- Rakendused jooksevad operatsioonisüsteemist lahutatult (Decoupled)
- Kaasaskantavus (Portability)
  - Võimalus kasutusele võtta ja kasutada kõikjal
- Ressursitõhusus ja -tihedus
- Konteinerite isoleeritus
- Ressursside jagamine
- Kiirus
  - Konteinerite loomine, kopeerimine, käivitamine või hävitamine sekunditega
- Skaleeritavus
- Parem arendaja tootlikkus

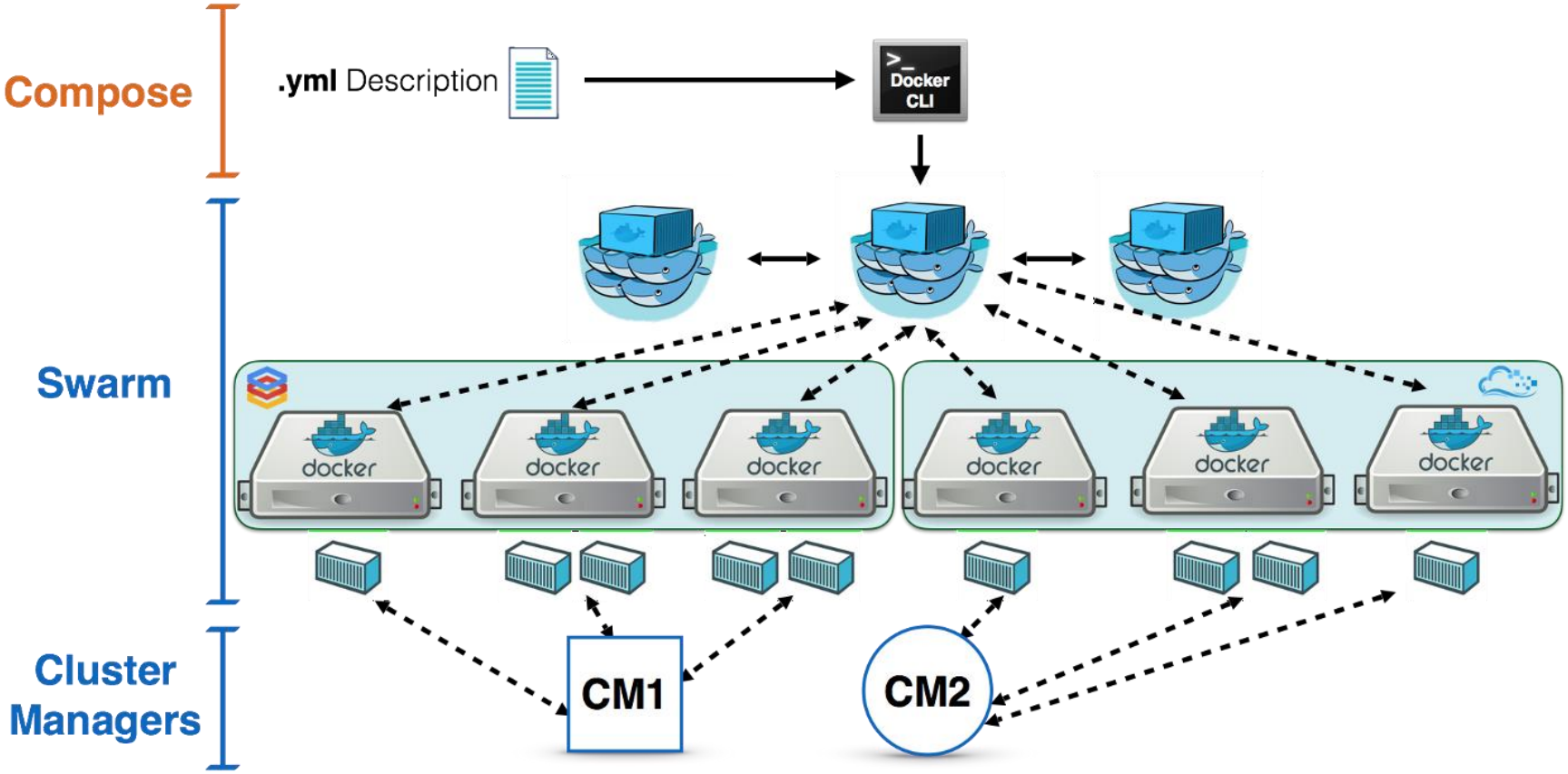
Docker Compose

- Tööriist mitme-konteineriliste rakenduste defineerimiseks ja käivitamiseks
  - Sobib lihtsate Mikroteenustepõhiste rakenduste jaoks
- Kasutab kõigi konteinerite konfigureerimisek s ühte YAML-faili
- Võimalik kõik konteinerid luua ja käivitada ühe käsuga

```
secrets:
  postgres_user:
    file: ./secrets/postgr_user.txt
  postgres_password:
    file: ./secrets/postgr_password.txt

services:
  postgres:
    restart: always
    image: postgres:latest
    volumes:
      - db-data:/var/lib/postgresql
    secrets:
      - postgres_user
      - postgres_password
    environment:
      POSTGRES_DB: db
      POSTGRES_USER_FILE: /run/secrets/postgr_user
      POSTGRES_PASSWORD_FILE: /run/secrets/postgr_password
    expose:
      - "5432"
  web:
    restart: always
    build: ./web
    links:
      - postgres:postgres
    secrets:
      - postgres_user
      - postgres_password
    environment:
      DEBUG: False
      POSTGRES_DB: db
      DATABASE_PORT: 5432
      POSTGRES_USER_FILE: /run/secrets/postgr_user
      POSTGRES_PASSWORD_FILE: /run/secrets/postgr_password
    expose:
      - "8000"
    depends_on:
      - postgres
    command: >
      sh -c "gunicorn -w 2 -b :8000 app:app"
```

Docker swarm



https://medium.com/tech-tajawal/create-cluster-using-docker-swarm-94d7b2a10c43

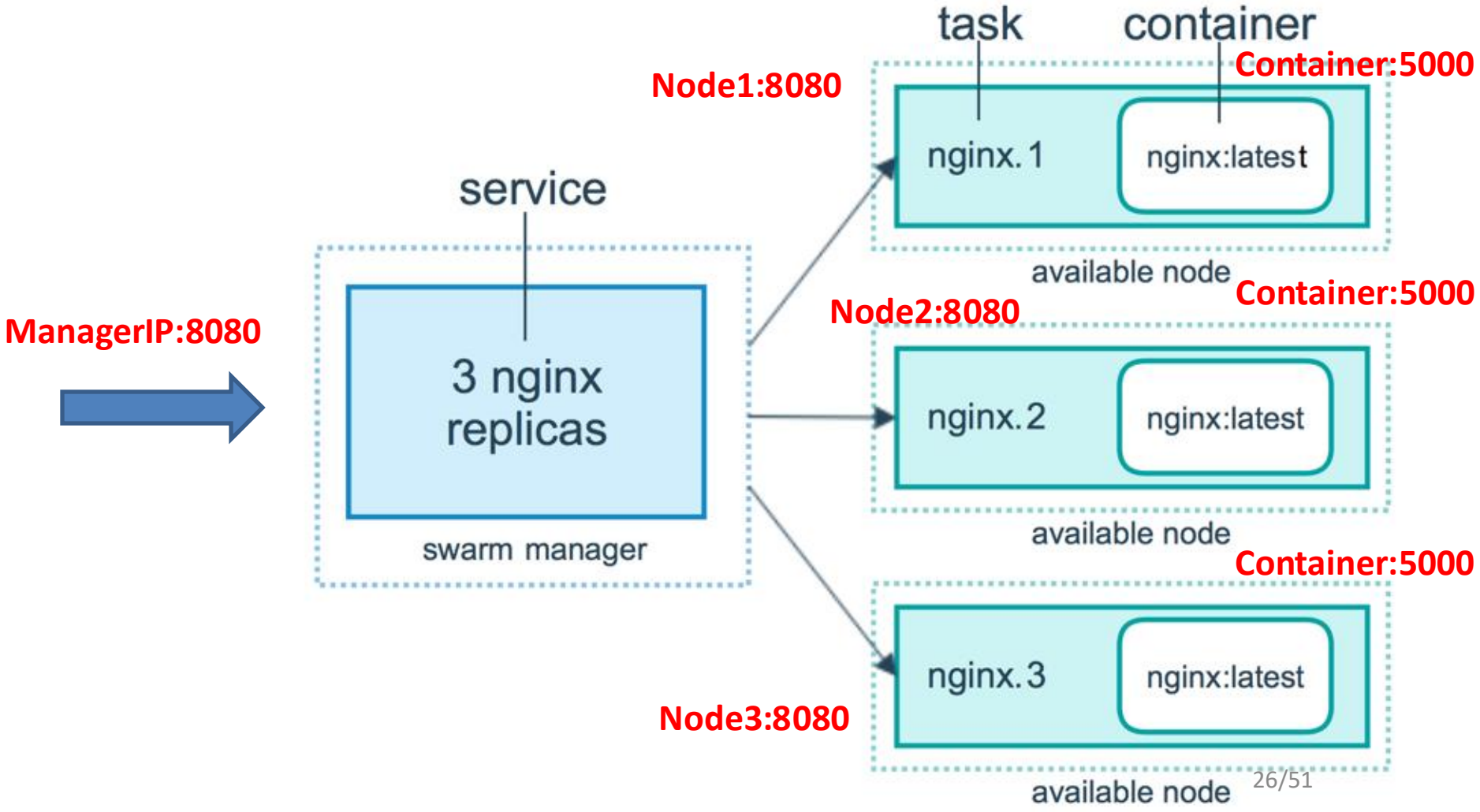
Docker Swarm teenuste käivitamine

- Docker teenuse loomine

```
docker service create --replicas 3 -p 8081:5000 --name web
shivupoojar/mywebv1.0
```

- Docker teenusele ligi pääsemine
  - http://MainNodeIP:8081
- Sisseehitatud koormuse tasakaalustaja (Load Balancer) jaotab liikluse teenuse siseste konteinerite vahel
  - Round-Robin algoritm

Teenuse kasutamine ja portide suunamine



https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

Docker teenuste skaleerimine

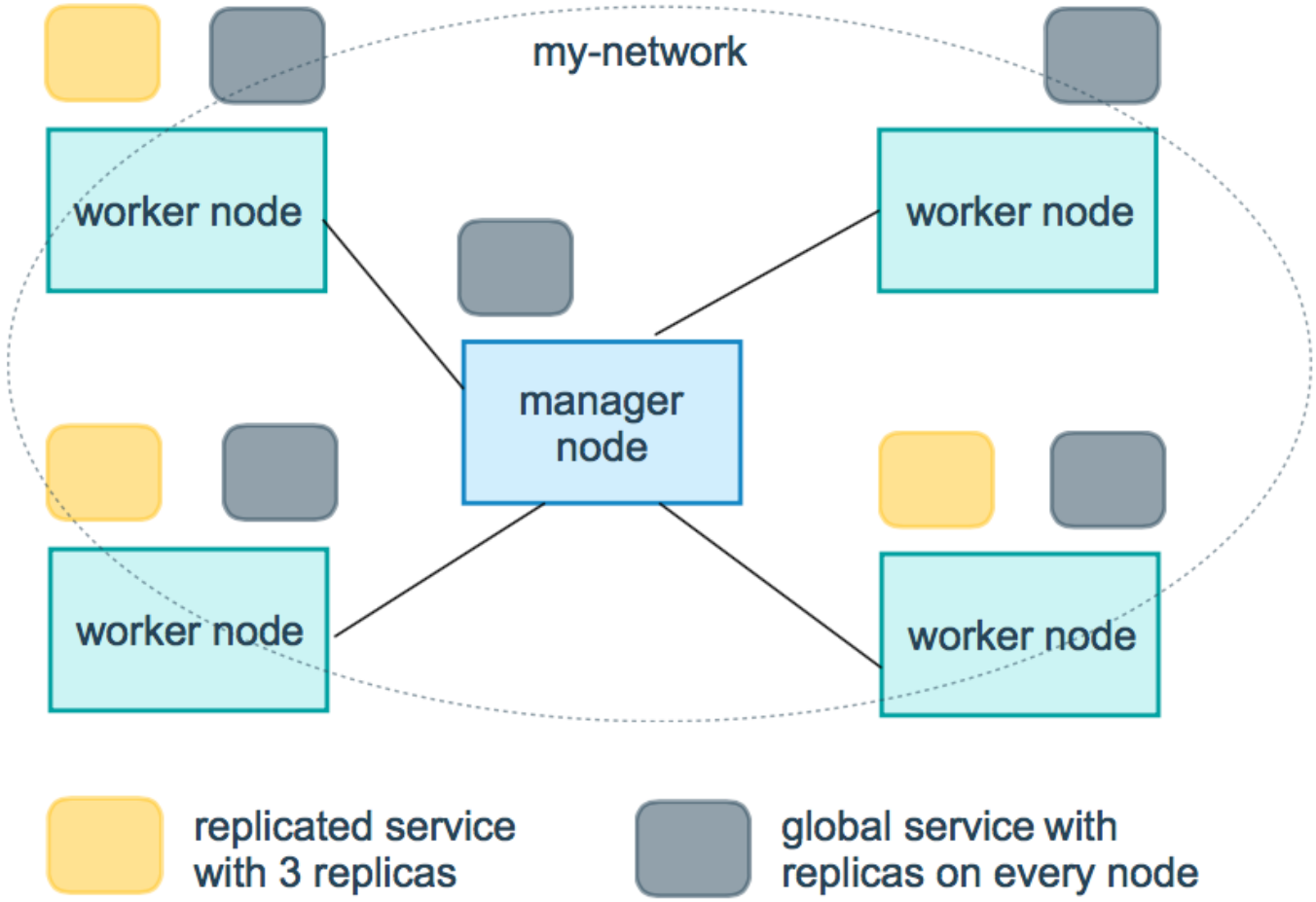
- Käsk teenuse replikaatide arvu muutmiseks:

```
docker service scale web=50
Web scaled to 50
```

- Alla skaleerimine

```
docker service scale web=1
Web scaled to 1
```

Teenuste replitseerimine



https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

Docker Swarm kasutamise probleemid

- Docker Swarm on lihtsam kasutada, aga suure hulga mikroteenuste haldamise korral tekivad probleemid
- Raskem hallata erinevaid alam keskkondasid (Test, production, developement)
- Klastri siseste ning mikroteenuste vaheliste võrkude haldamine osutub tihti keeruliseks
- Automaatset skaleerimist peab ise implementeerima
- Vähem stabiilne

Konteinerite väljakutsed

- Konteineritel on nõrgem isolatsioon
- Salvestusruumi haldamine võib muutuda problemaatiliseks
  - Stateless konteinerid, konteinerite migratsioonid, katkestused
- Haldamine on keerulisem, kuna komponentide suurus väheneb ja nende arv suureneb
- Oluline on tagada, et seisakuid ei oleks, rikete tagajärjed on minimaalsed
- Süsteemid peaksid skaleerima elastselt
- Suurte konteineripõhiste süsteemide haldamise lihtsustamiseks ja automatiseerimiseks on vaja konteinerite orkestreerimise lahendusi
  - Nt Docker Swarm, Kubernetes

Konteinerite orkestreerimine

- Konteinerite orkestreerimisplatvormid pakuvad lisafunktsioone, näiteks:
  - Klastrite haldamine ja replitseeritud klastriülesed teenused
    - Docker Swarm, Kubernetes
  - Teenuse leidmine ja koormuse tasakaalustamine
    - Docker Swarm, Kubernetes
  - Salvestusruumi orkestreerimine
    - Kubernetes
  - Automatiseeritud levitamine ja versioonide tagasi pööramine
    - Kubernetes
  - Eneseparanemine
    - Kubernetes, Docker Swarm (somewhat)
  - Saladuste ja konfiguratsioonihaldus
    - Kubernetes
  - Automaatne skaleerimine
    - Kubernetes

Kokkuvõte

- Mikroteenused lihtsustavad hajussüsteemide loomist, individuaalsete komponentde uuendamist ning skaleerimist
- Konteinerid on olnud ühed peamised Mikroteenuste edukuse võimaldajad
- Docker Swarm võimaldab luua Docker klastreid
  - Aga **Kubernetes** on palju võimsam ja palju rohkem automatiseeritud
  - Palju sobivam suuremate konteinerite klastrite ehitamiseks
- Mikroteenused teevad süsteemide arhitektuuri keerulisemaks
  - Tükke on lihtsam luua
  - Kogu süsteemi võib olla keerulisem hõlmata ning hallata
- Mikroteenused võivad olla ebaeffektiivsemad resursikasutuse osas
  - Nt. Lokaalse võrgu kasutamine teenuste vahel vs Jagatud mälualad

Järgmine loeng

- Nanoteenused (FaaS, Serverless)

- Tulevikus räägime:
  - Täpsemalt Kubernetesest
  - Mikroteenuste arhitektuuridest rohkem
  - Täielikult pilvepõhistest rakendustest

Praktikum

- Jagama oma API rakenduse kaheks mikroteenuseks
- Laiendame raamatute otsingu mikroteenust
- Paneme üles konteineritena