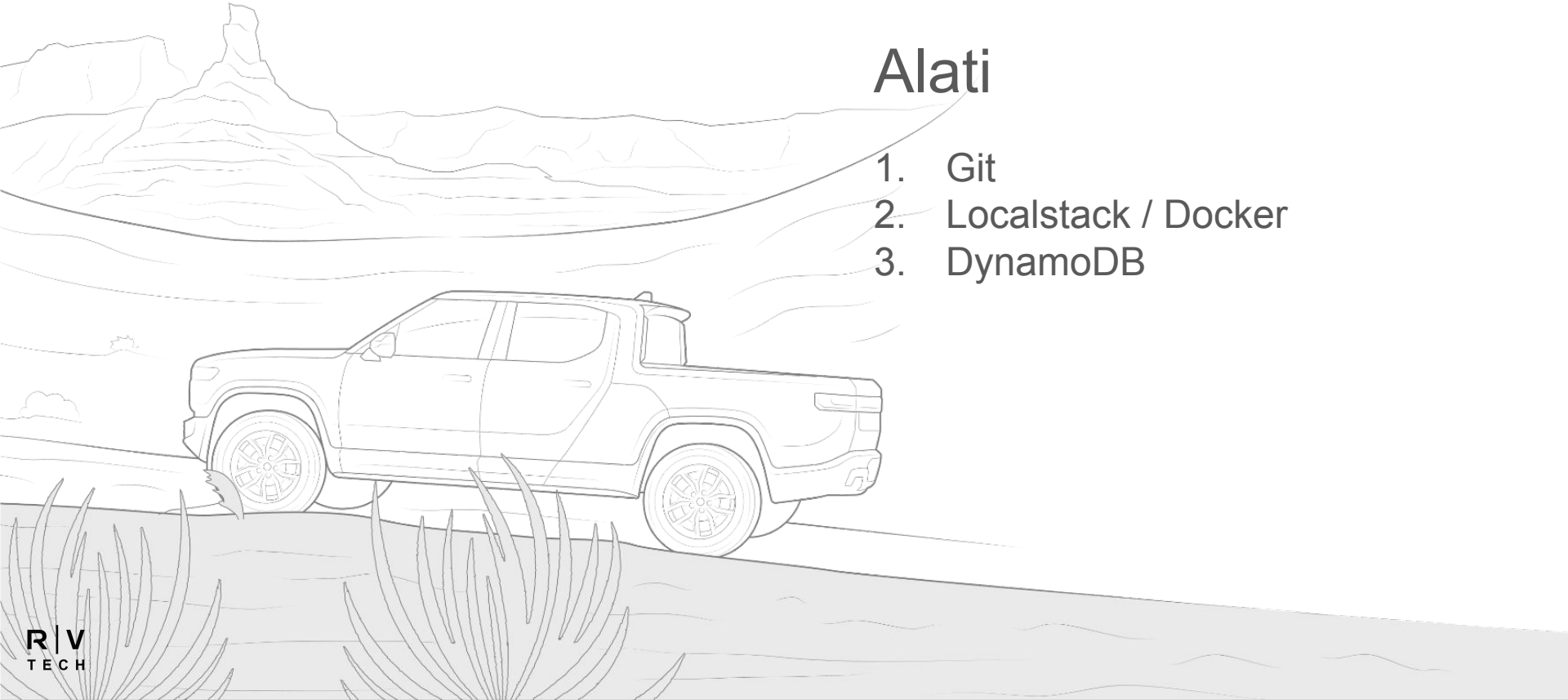


Alati

1. Git
2. Localstack / Docker
3. DynamoDB

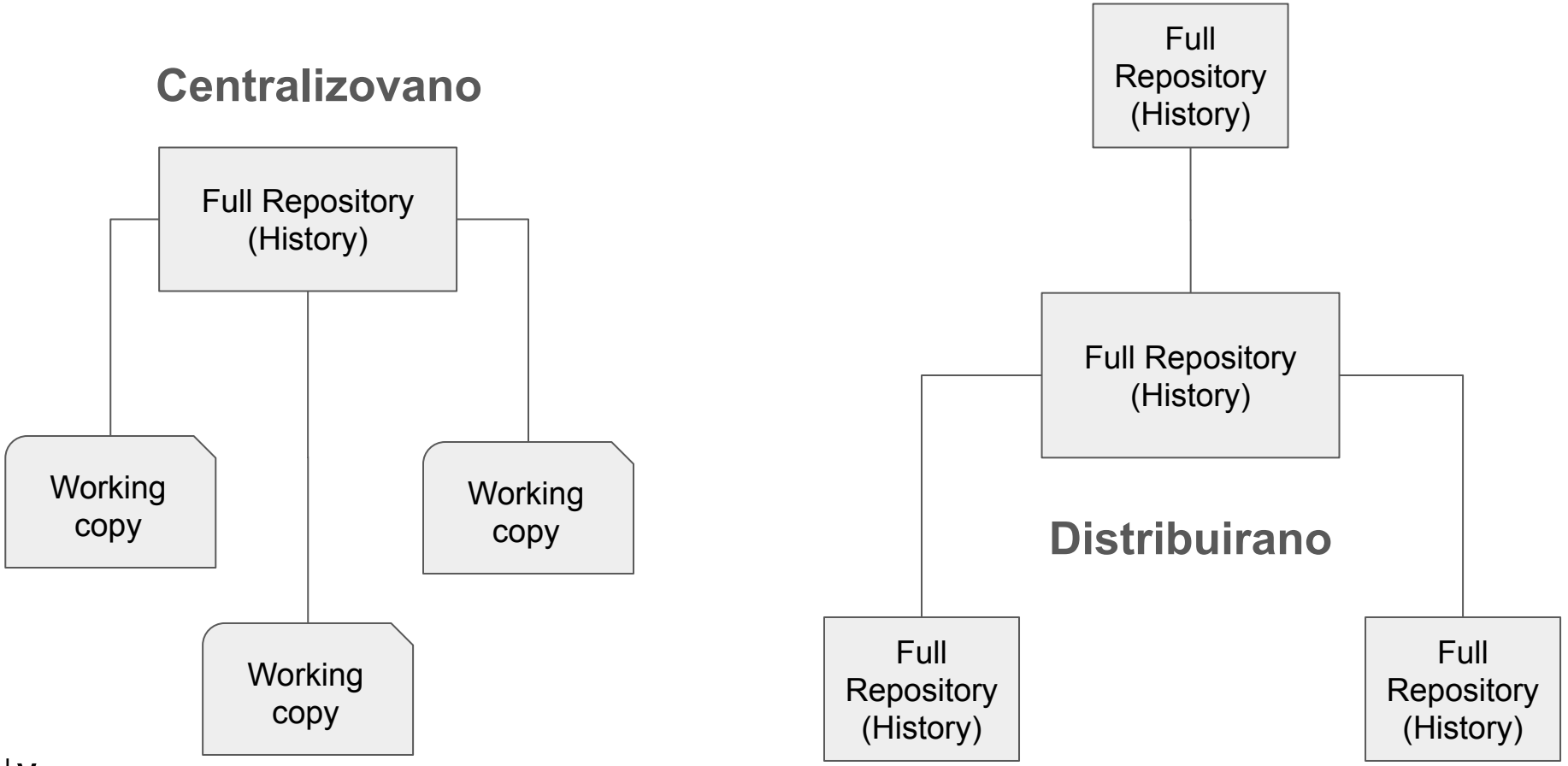
R|V
TECH

Kako nastaje softver koji menja svet

VS

BITKEEPER

1. Git


Centralizovano
Full Repository (History)
Working copy
Working copy
Working copy
Full Repository (History)
Full Repository (History)
Distribuirano
Full Repository (History)
Full Repository (History)

R|V TECH

1. Git | working directory, index, repo

```
import json

def extremely_complex_cloud_algorithm (data):
    # TODO: Connect this to AWS SageMaker later
    # TODO: Figure out why the AWS bill is $5, 000 ??

    print("DEBUG: Did I get here? Yes. ")
    print("DEBUG: Data is -> ", data)

    # Temporary logic until I read the documentation :
    return {
        "status": "It works... I think?",
        "probability_of_success": "50/50",
        "magic_number": 42
    }
```

> git status
On branch gentle_tweaks

Changes not staged for commit:
modified: mysickapp/mysickcode.py

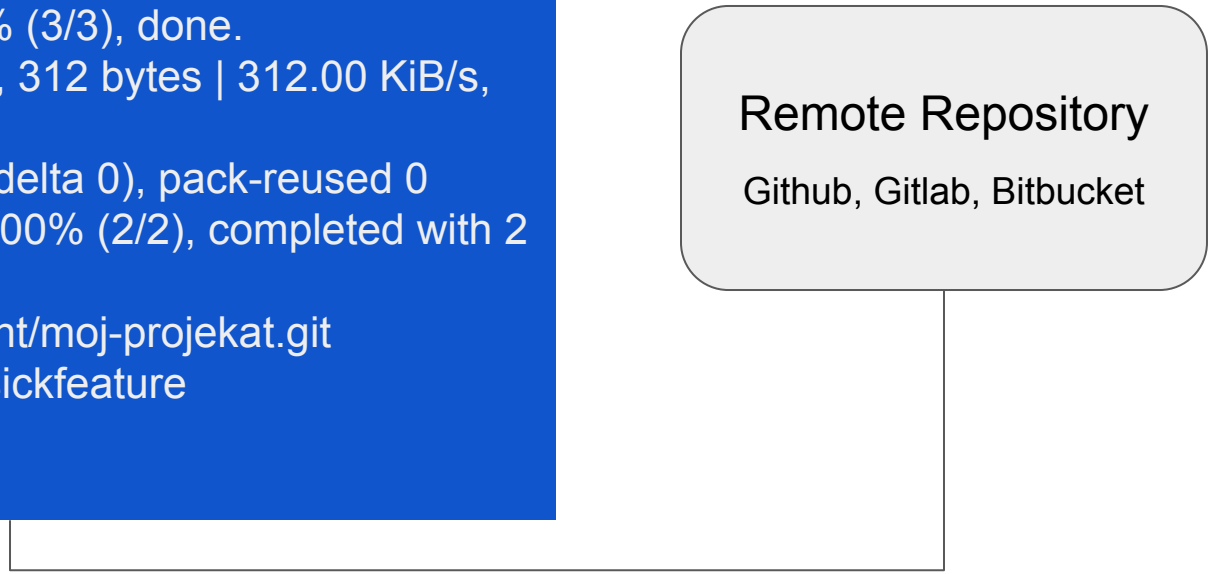
> git add .
Changes to be committed:
modified: mysickapp/mysickcode.py

> git commit -m “yolo”
[main 9a1b2c3] yolo
1 file changed, 331 insertions(+)

R|V TECH

1. Git | push

```
> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 312 bytes | 312.00 KiB/s,
done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2
local objects.
To https://github.com/student/moj-projekat.git
   2f3a4b..9c8d7e  feature/sickfeature
->feature/sickfeature
```

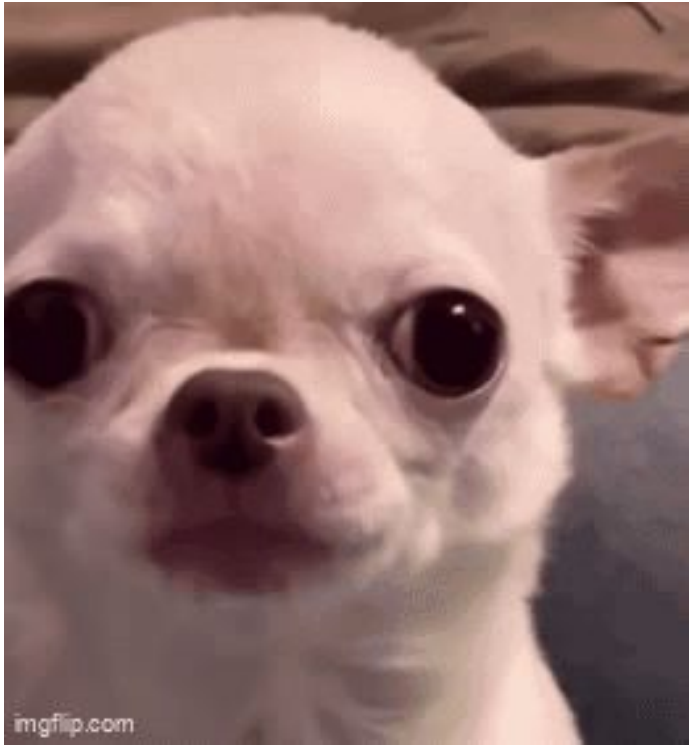


1. Git | kako rešavate konflikt?

R|V TECH

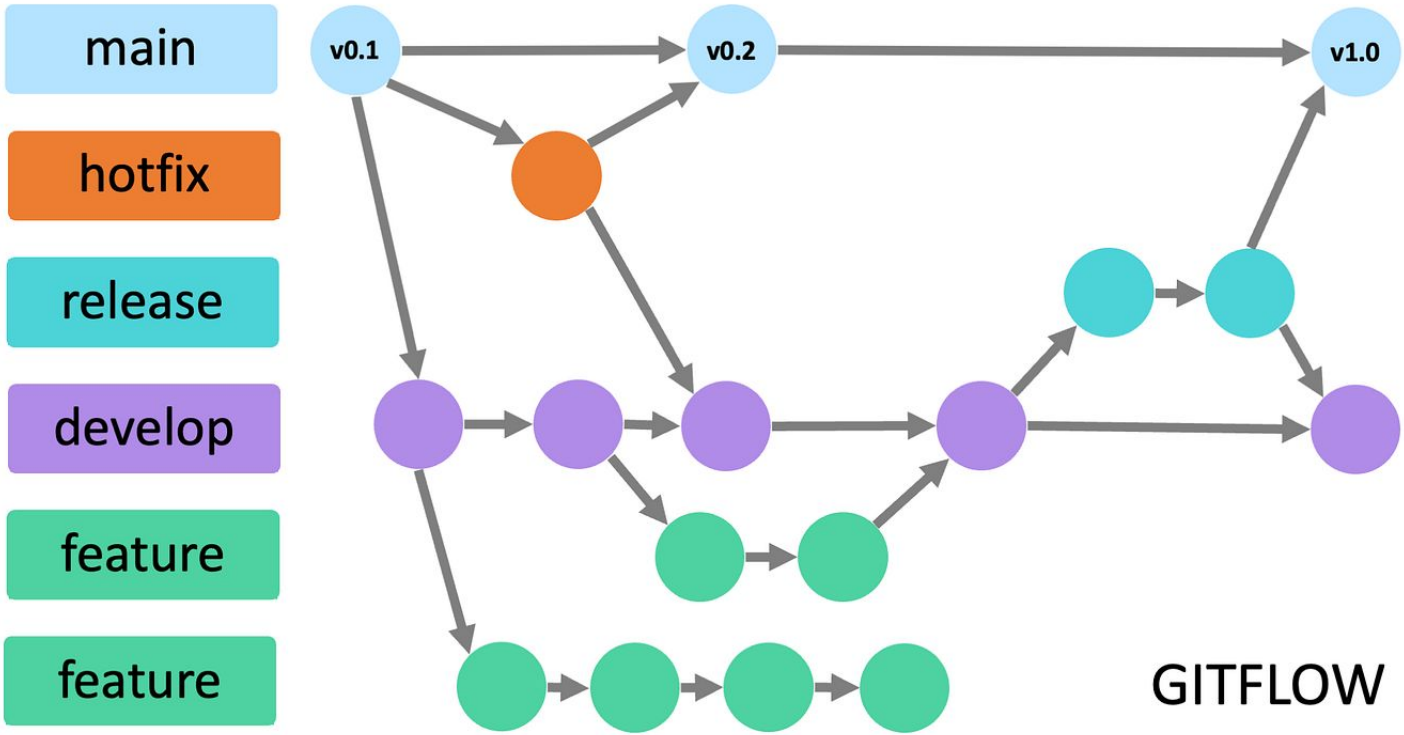
1. Git | Resavanje konflikta

git push –force



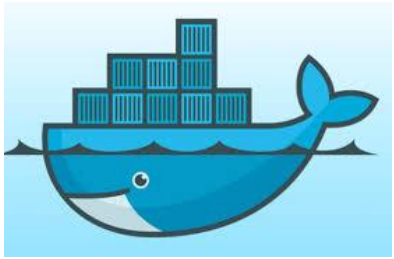
Rešiću ja kod mene pa guram

1. Git | git-flow branching strategija



2.1 Docker | Zašto?

- Rešava problem “Radi na mojoj mašini”
  - Razlicite verzije OS-a, biblioteka, Python/Node
- Koristi se za lokalno pokretanje projekta i u produkciji



2.1 Docker - Ključni Koncepti

1. Dockerfile

```
# 1. OSNOVA: Krećemo od čistog Ubuntu Linuxa
FROM ubuntu:22.04

# 2. PRIPREMA SISTEMA:
# Instaliramo curl i dodajemo Node.js repository
RUN apt-get update && apt-get install -y curl
RUN curl -fsSL https://deb.nodesource.com/setup_18.x | bash -

# 3. INSTALACIJA NODE.JS:
# Sada instaliramo sam Node.js
RUN apt-get install -y nodejs

# 4. LOKACIJA
WORKDIR /app

# 5. ZAVISNOSTI
COPY package.json .
RUN npm install

# 6. KOD
COPY . .

# 7. PORT I POKRETANJE
EXPOSE 3000
CMD ["node", "app.js"]
```

Dockerfile

R|V TECH

2.1 Docker - Ključni Koncepti

2. Docker image
   - Nastaje pokretanjem `docker build -t app:latest .`
   - Zamrznuti snapshot aplikacije i svih njenih zavisnosti

3. Docker container
   - Nastaje pokretanjem `docker run -p 3000:3000 app:latest`
   - Ziva instanca aplikacije

Jos bitnih komandi
- `docker ps` – izlistava sve aktivne container-e
- `docker image ls` – izlistava sve image koji postoje na masini
- `docker stop <container_id>` – zaustavlja kontainer

2.2 Docker Compose

- Alat koji služi za orkestraciju više kontejnera.
- Umesto ručnog pokretanja, piše se YAML fajl konfiguracija

```yaml
version: '3.8'  # Verzija compose sintakse

services:       # Ovde definišemo naše kontejnere

  backend-app:  # Ime prvog servisa
    build: .    # "Izbilduj image iz trenutnog foldera (koristi Dockerfile)"
    ports:
      - "3000:3000" # Otvori port 3000
    depends_on:
      - baza    # "Sačekaj da se baza pokrene pre mene"
    environment:
      - DB_HOST=baza # Kažemo aplikaciji gde se nalazi baza

  baza:         # Ime drugog servisa
    image: postgres:13  # Ne bildujemo, skidamo gotov image
    environment:
      POSTGRES_PASSWORD: tajna_sifra
    volumes:
      - pg-data:/var/lib/postgresql/data # Čuvaj podatke čak i ako obrišem kontejn

volumes:
  pg-data:      # Definicija prostora za čuvanje podataka
```

docker-compose.yml

- docker compose up
- docker compose down

2.3 Localstack | sta?

- **Lokalna imitacija AWS Okruženja**
  - Pruža potpuno funkcionalne emulatore za preko 60 AWS servisa (S3, Lambda, DynamoDB, SQS, itd.) direktno na vašoj mašini.
- **Brže testiranje i razvoj**
- **Nezavisnost od interneta i troškova**
- **Jednostavna konfiguracija**
  - Obično se pokreće kao jedan Docker kontejner, što ga čini lakim za integraciju u bilo koji razvojni setup.

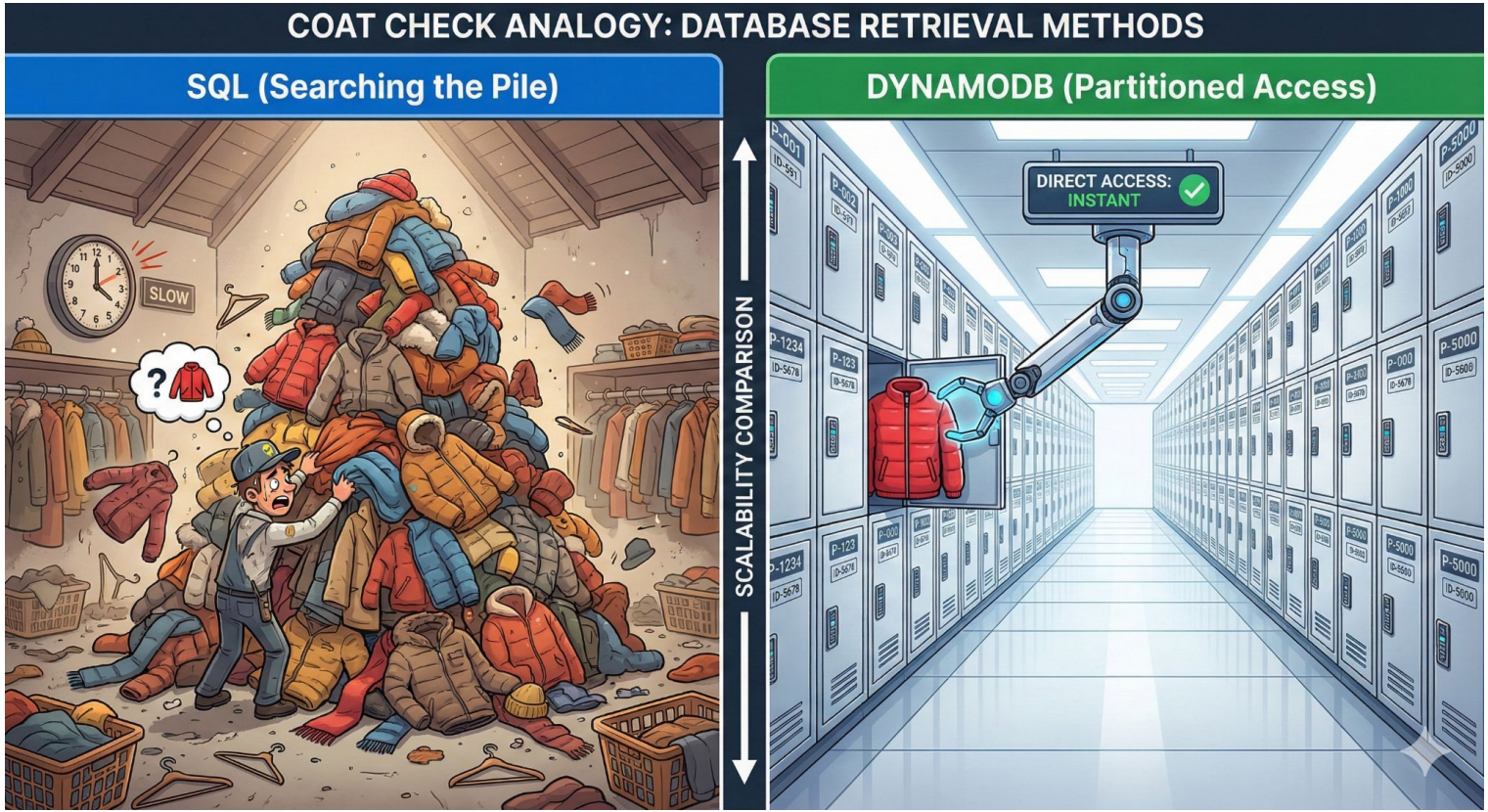
2.3 Localstack | kako?

- **Docker kontejnerizacija:** Lokalni servisi se pokreću kao lagani procesi unutar jednog Docker kontejnera za izolaciju i portabilnost.
- **Servis Endpoin-i:** Svaki emulirani AWS servis izlaže svoj lokalni endpoint (npr. `http://localhost:4566`) umesto pravog AWS region endpoint-a.
- **`awslocal` CLI Alat**
  - Specijalizovani CLI alat koji automatski usmerava komande na LocalStack endpoint-e, olakšavajući interakciju (alternativa standardnom `aws` CLI-ju).
- https://docs.localstack.cloud/aws/services/
- https://docs.localstack.cloud/aws/getting-started/installation/#docker-compose

3. DynamoDB | šta?

- **Fully Managed & Serverless:** Nema održavanja servera
- **Key-Value Store:** Optimizovano za brzi pristup po ključu
- **Poreklo:** Nastao da reši "Black Friday" problem

3. DynamoDB vs SQL

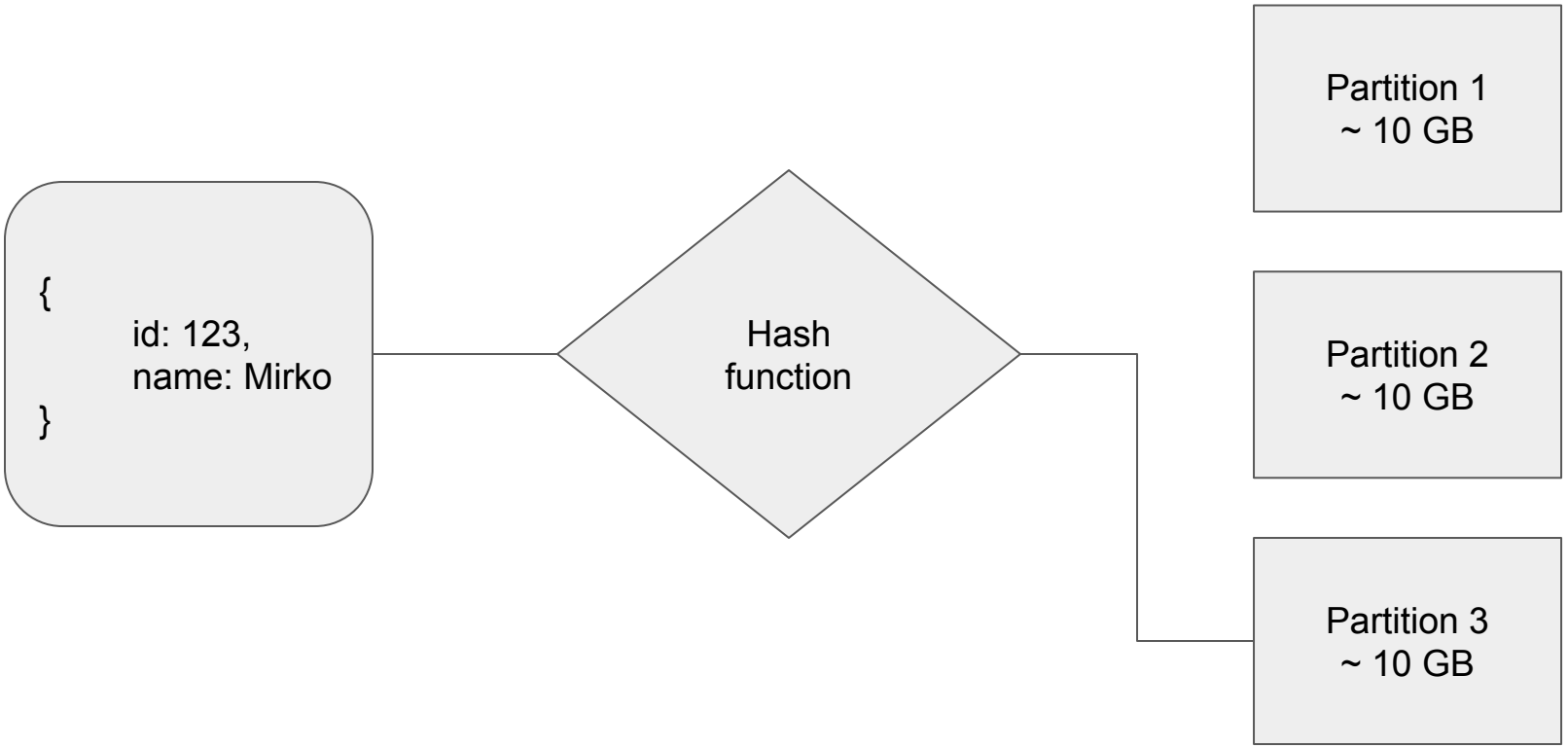


R|V TECH

3. DynamoDB | zašto - black friday problem

- **Automatsko skaliranje:** milioni zahteva u sekundi
- **Munjevit odziv:** jednocifrena milisekunda za citanje
- **On demand:** ako ne koristiš, ne plaćaš

3. DynamoDB | kako?



3. DynamoDB | ispod haube?

- **Partition Key:** Obavezan atribut koji određuje logičku particiju u kojoj će podaci biti uskladišteni (glavna tačka distribucije).
- **Sort Key:** Opcioni atribut koji, zajedno sa Particionim Ključem, čini Primarni Ključ i omogućava sortiranje i fleksibilnije upite unutar particije.
- **Globalni Sekundarni Indeksi (GSI):** Omogućavaju kreiranje alternativnih primarnih ključeva za upite, proširujući fleksibilnost pristupa podacima.

3. DynamoDB | query vs scan

Query

- Traži podatke na osnovu ključa
- Brz
- Efikasan

Scan

3. DynamoDB | database design

ACCESS PATTERNS FIRST

3. DynamoDB | primer

Tabela: `Users`

- **Primarni ključ:**
  - **Partition Key:** `UserID` (String)
- **Upotreba:** Brzo preuzimanje detalja korisnika (profil, ime, email) na osnovu njihovog ID-a.

Tabela: `Posts`

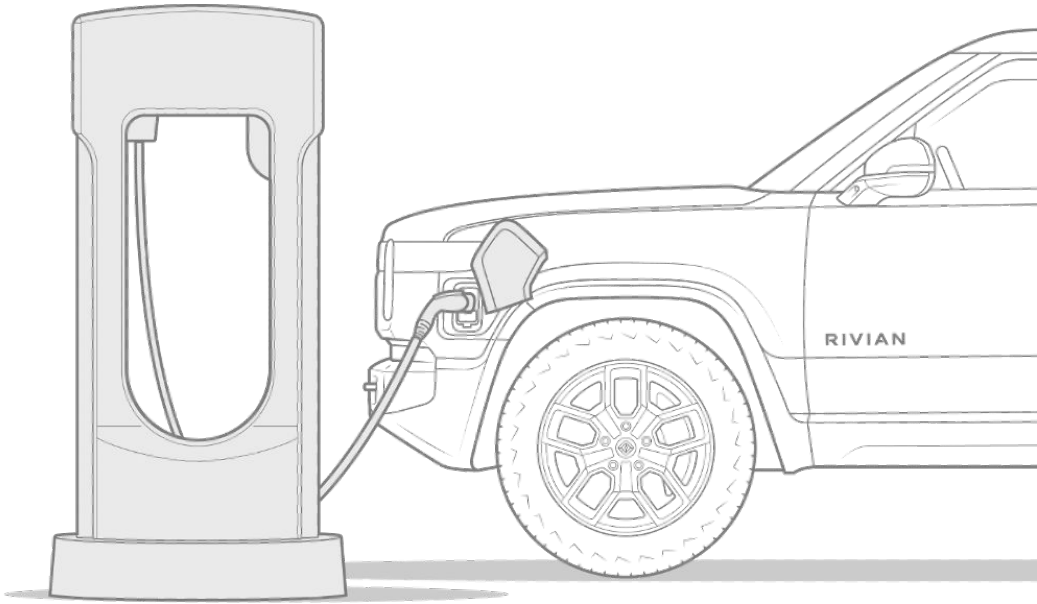
- **Primarni ključ:**
  - **Partition Key:** `UserID` (String)
  - **Sort Key:** `PostTimestamp` (Number/String)
- **Upotreba:** Preuzimanje svih postova određenog korisnika, automatski sortiranih po vremenu objave (najnoviji prvi).

  Korišćenje GSI-a: Možete kreirati GSI na atributu `Tags` (ako bi bio Lista) da biste pronašli sve postove sa određenim tagom, što ne bi bilo moguće koristeći Primarni Ključ.

Atribut	Tip Podataka	Vrednost
UserID (PK)	String	korisnik_123
Ime	String	Marko Markovic
Email	String	marko@primer.com
Status	String	Aktivan
Pretplate	Number	150
Adresa	Map (JSON)	{“grad”: “Beograd”, “drzava”: “Srbija}
Interesovanja	List	[“Programiranje”, “Sport”]

R | V TECH

PITANJA



R|V
TECH