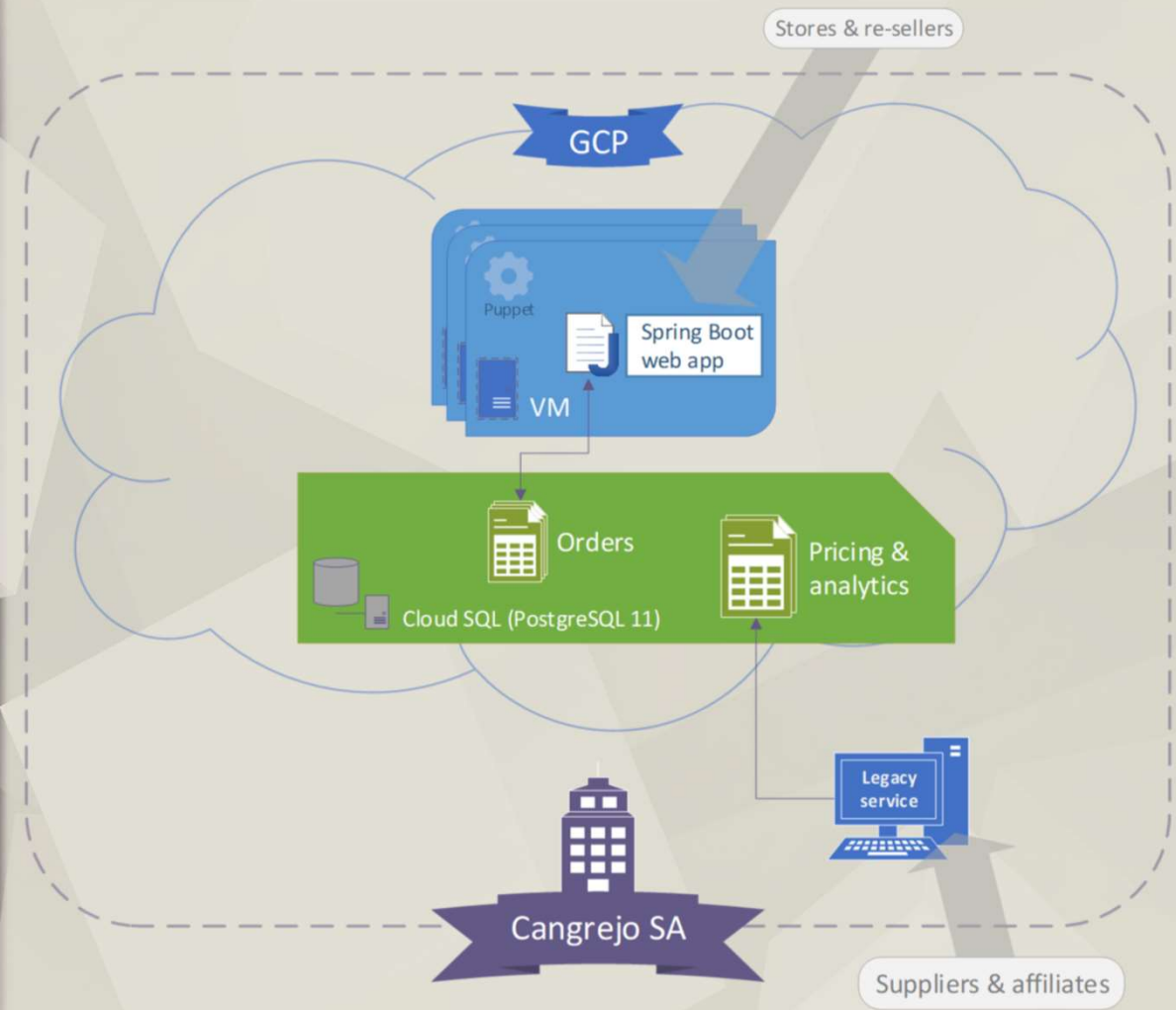
Läksyt: Bramante

A solution architecture plan for migrating a global retailer company from a dated GCP/on-prem infrastructure (centered on a PostgreSQL instance) onto services provided by Aiven.io

Current set-up

- Some infrastructure already in the cloud (GCP)
- Single relational data store (Cloud SQL for PostgreSQL 11)
- On-prem service aggregates data from suppliers and affiliates
- That pricing & analytics data is batch uploaded once a day

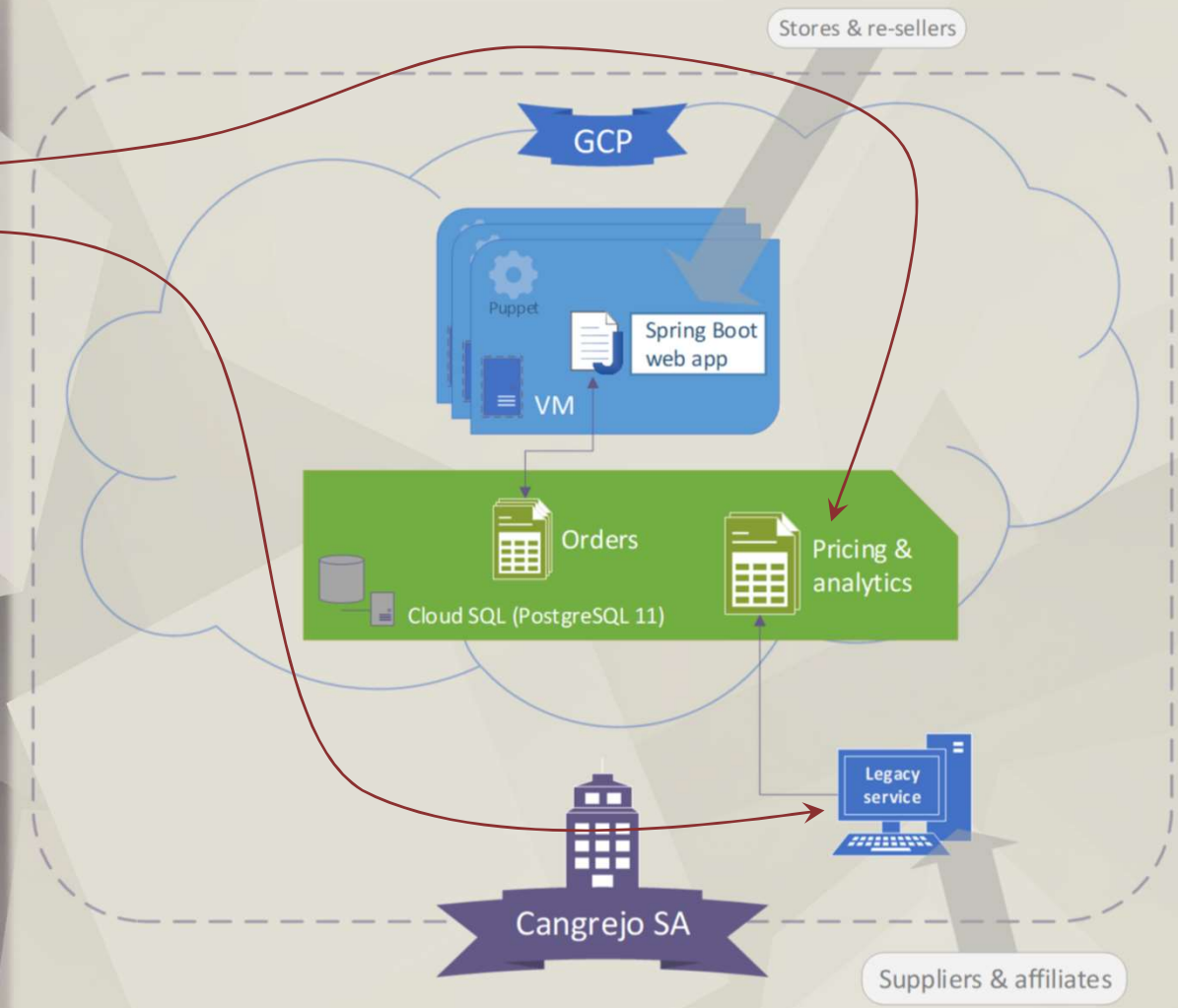


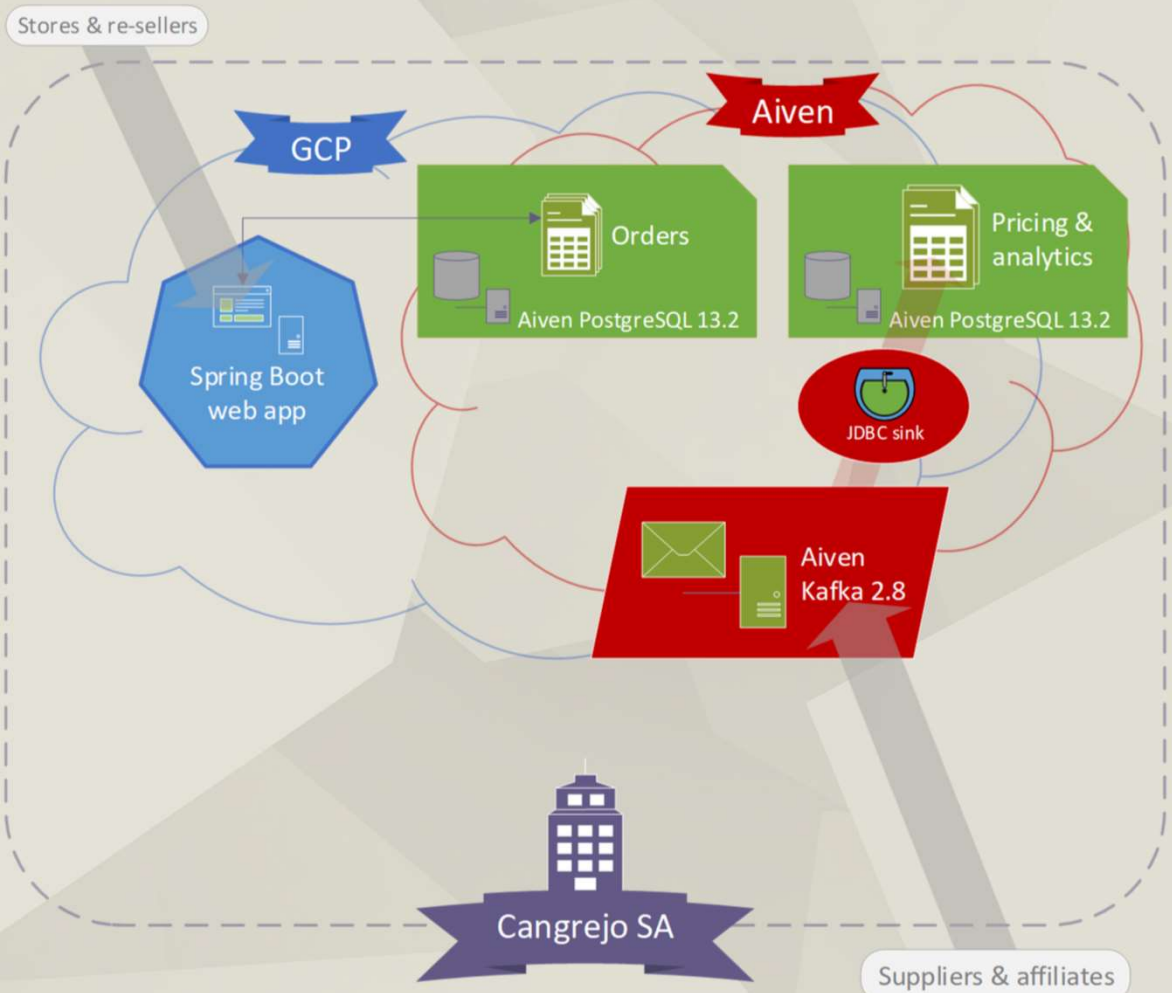
Pain points

- Slow analytical queries
- Inflexible batch ingestion
- Overall dated infrastructure

Requirements

- Faster analytical queries
- Modernized set-up in the cloud
- No vendor lock
- Data security & GDPR compliance
- Support for business expansion
- Painless integration with third parties
- Quicker reaction to incoming pricing & business data





Proposed set-up

- Split database by usage domains:
  - Independent scaling
  - Isolation of query workloads
- Migrate both databases to Aiven PostgreSQL 13.2:
  - Orders: "Business-4"+
  - Analytics: "Business-4"+
- Ingest pricing & analytics data via Aiven Kafka 2.8 and JDBC sink:
  - Kafka: start with "Business-4", scale as needed
  - Connector: start with "Startup-4", scale as needed

Roads not taken

NoSQL datastore

E.g., use a non-relational datastore for pricing & analytics data

- Nothing in the problem statement suggests the need for massive write scalability or time-series optimization
- At the same time, analytical queries usually require SQL
- A specialized datastore might end up costing substantially more for no discernable benefit

Elastic Stack

E.g., use Elastic products for pricing & analytics data

- The problem statement does not directly call for full-text search (or other Elastic-specific) capabilities
- The cost concern from above still applies

BigQuery/Redshift

E.g., use a cloud-specific analytical data warehouse

- While these products match the problem description, they are proprietary and introduce the risk of vendor lock
- PostgreSQL 13.2, while providing internal performance improvements, also allows partitioning and other optimizations

Redis

E.g., use an in-memory cache for read queries

- The in-memory cache is useful for repeated read queries against a sub-set of “hot” data — a scenario not explicitly evident in this case