SUPERBET

How Superbet Migrated from NoSQL to the Cloud to Simplify Expansion to New Markets

Sergej Jakovljev, Senior Site Reliability Engineer
Ivan Hrastinski, Engineering Manager

RoachFest24

Happening

Driven by ambition. Powered by technology.

The Tech Engine Behind Superbet’s Global Platforms

Consumer brands

SUPERBET

🇷🇴 🇷🇸 🇵🇱 🇧🇷

NAPOLEON

🇧🇪

LUCKY 7™
VENTURES

🇸🇪

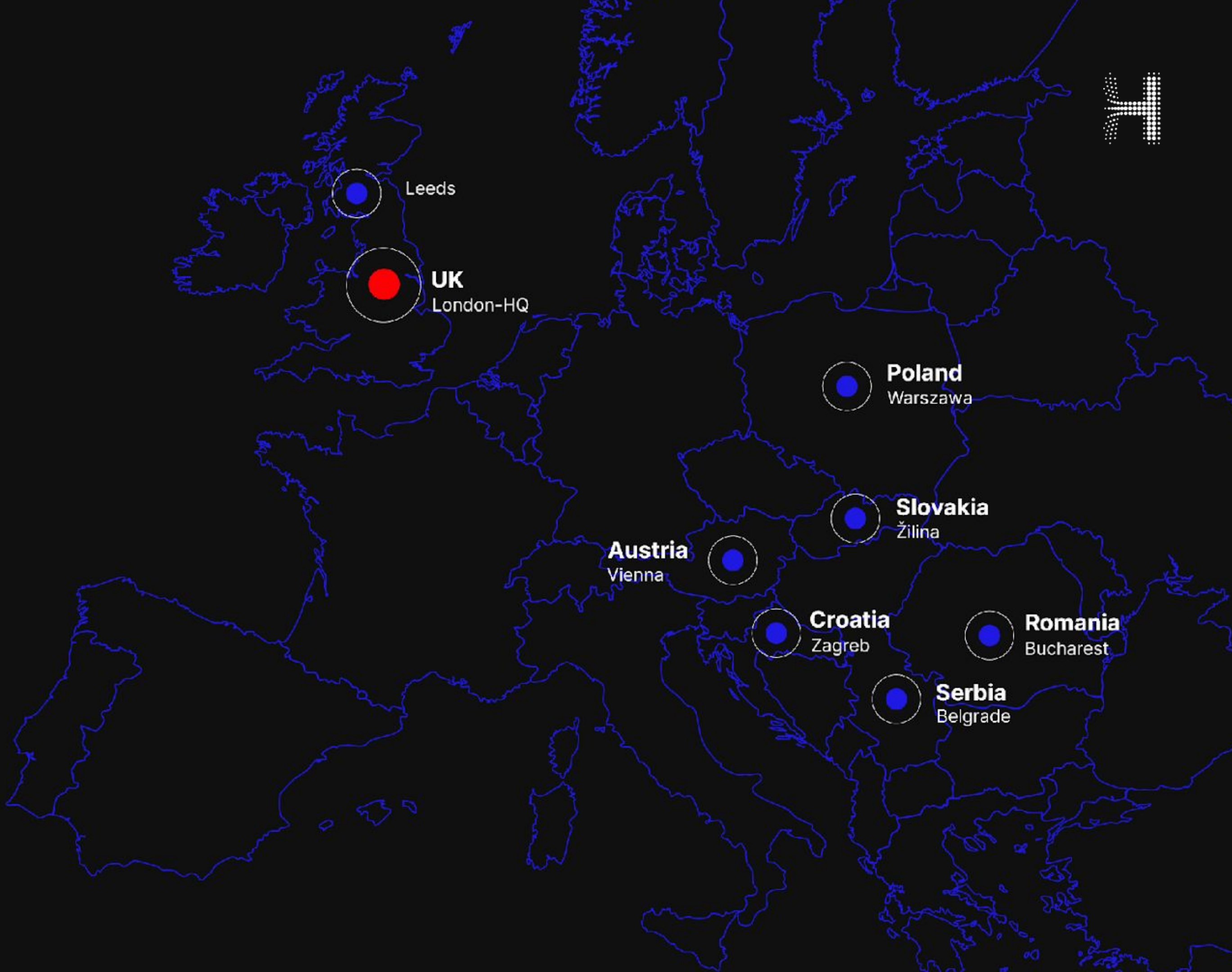
powered by

HAPPENING

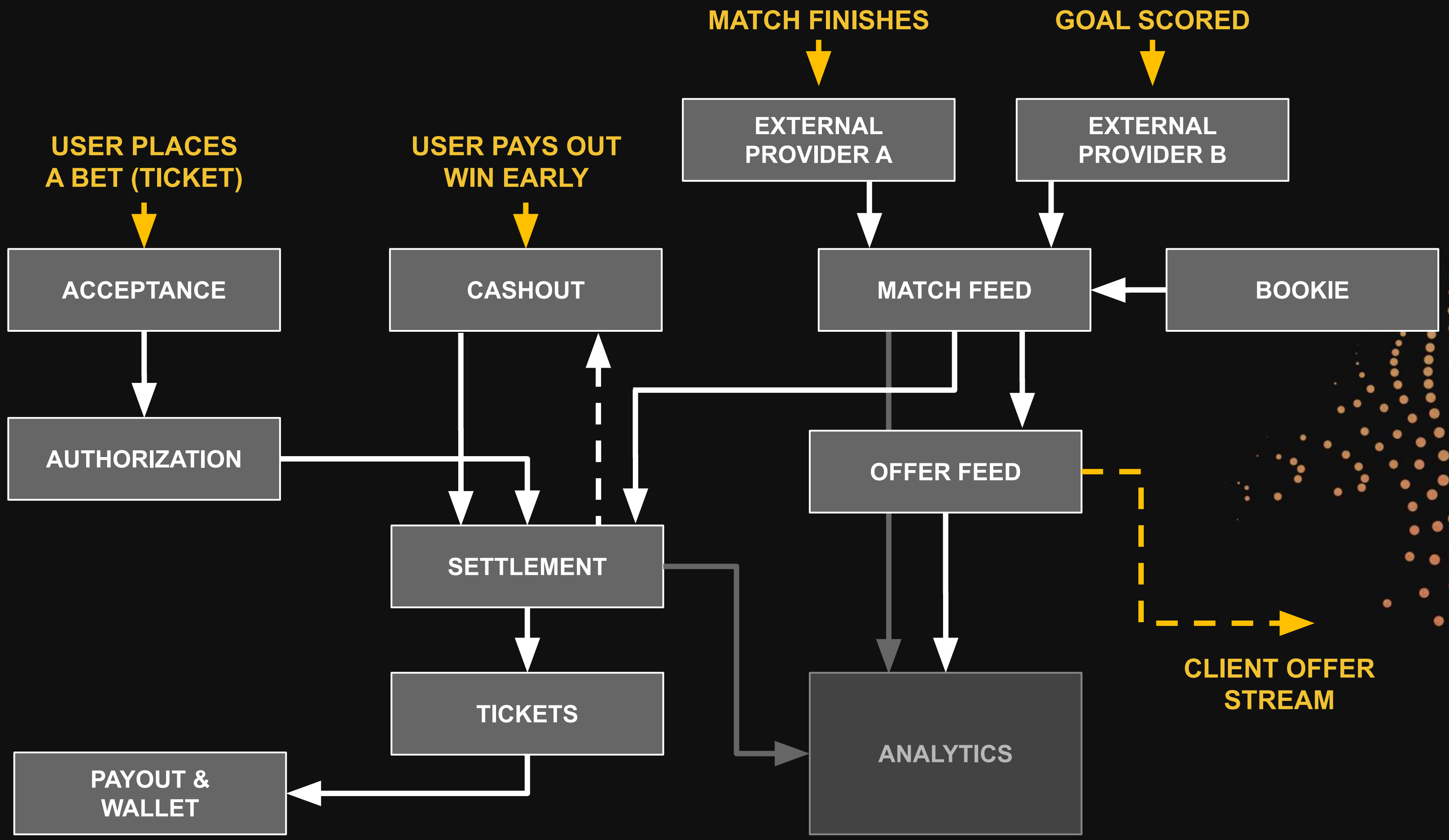
10+
Locations

30+
Product teams

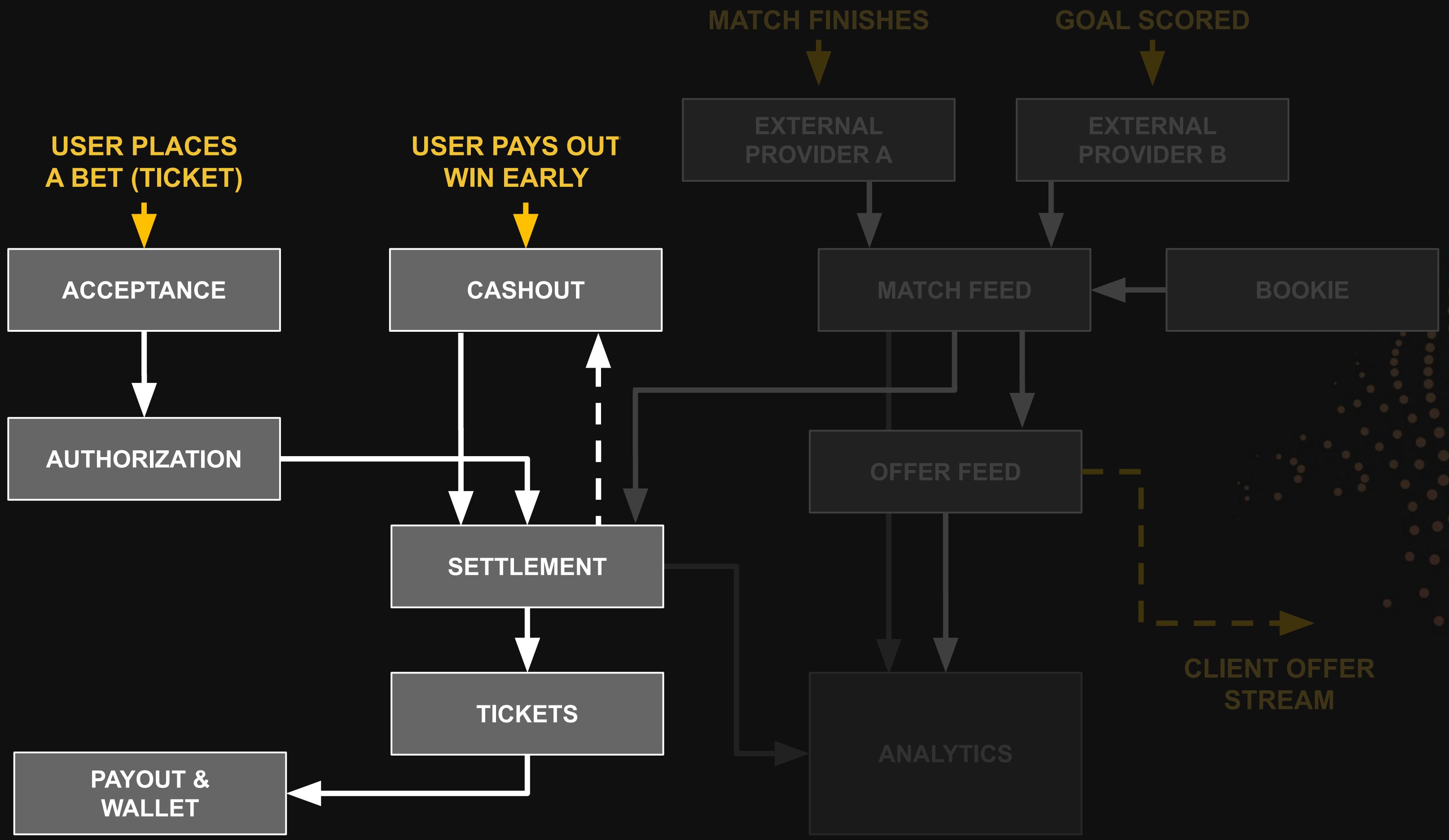
550+
People in tech

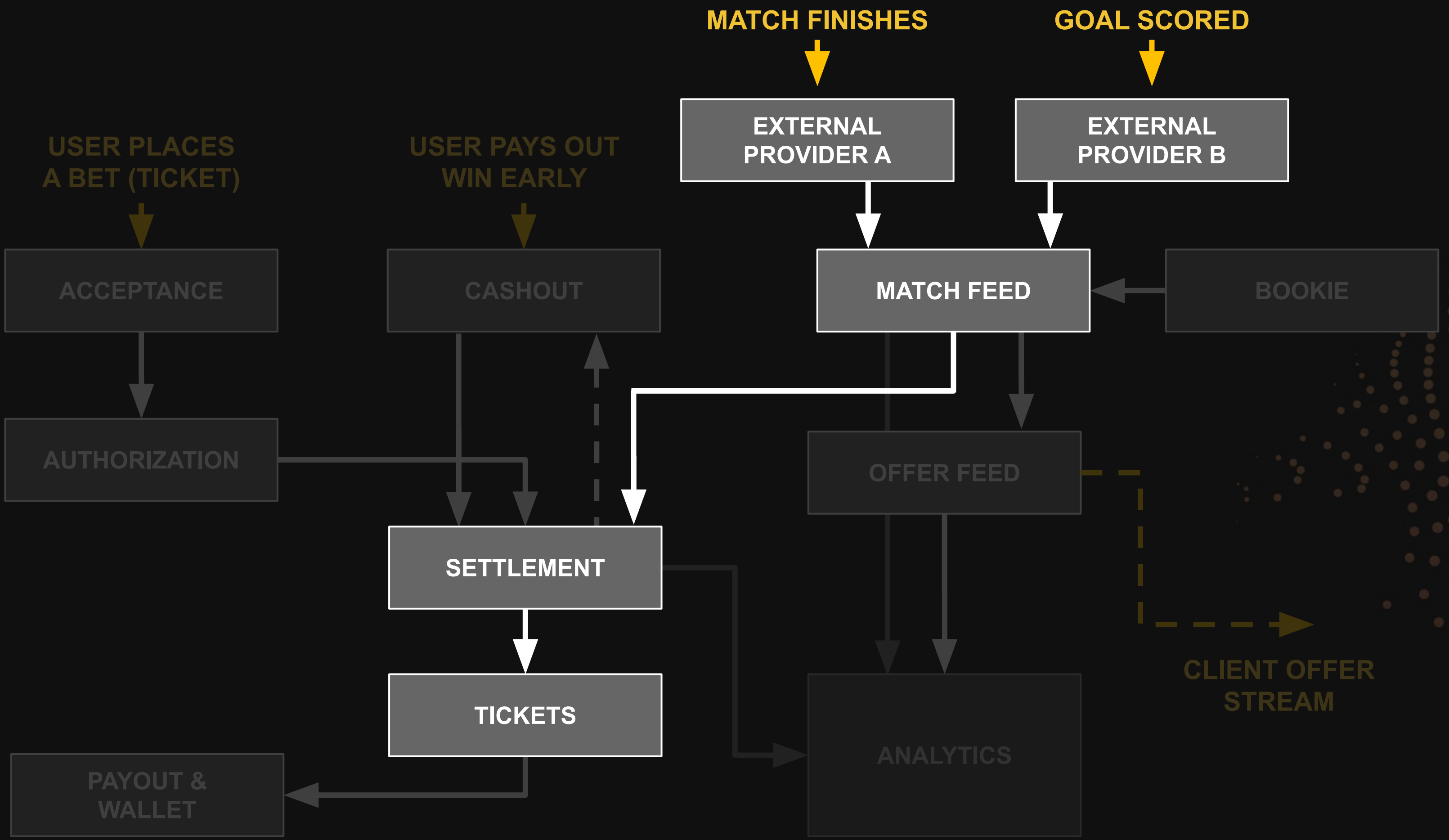


What is “betting” in Sports Betting?



MATCH FINISHES
GOAL SCORED
EXTERNAL PROVIDER A
EXTERNAL PROVIDER B
USER PLACES A BET (TICKET)
USER PAYS OUT WIN EARLY
ACCEPTANCE
CASHOUT
MATCH FEED
BOOKIE
AUTHORIZATION
OFFER FEED
SETTLEMENT
CLIENT OFFER STREAM
TICKETS
ANALYTICS
PAYOUT & WALLET


MATCH FINISHES
GOAL SCORED
EXTERNAL PROVIDER A
EXTERNAL PROVIDER B
USER PLACES A BET (TICKET)
USER PAYS OUT WIN EARLY
ACCEPTANCE
CASHOUT
MATCH FEED
BOOKIE
AUTHORIZATION
OFFER FEED
SETTLEMENT
TICKETS
CLIENT OFFER STREAM
ANALYTICS
PAYOUT & WALLET


MATCH FINISHES
GOAL SCORED
EXTERNAL PROVIDER A
EXTERNAL PROVIDER B
USER PLACES A BET (TICKET)
USER PAYS OUT WIN EARLY
ACCEPTANCE
CASHOUT
MATCH FEED
BOOKIE
AUTHORIZATION
OFFER FEED
SETTLEMENT
CLIENT OFFER STREAM
TICKETS
ANALYTICS
PAYOUT & WALLET

How we got started with CockroachDB

Tickets Service

- We need to store all the active (still not finally settled) and recent tickets (up to 30-90 days) so they are available quickly
- Part of wider pipeline architecture (based on Kafka), so data was already denormalized and in JSON format
- All querying is done by ticket ID and user ID
- Expected >10x growth over next 3 years
- Sports betting is strictly regulated industry
    - Data and processing had to reside in the licence country (Romania, Poland)
    - Data integrity is extremely important

We basically need a highly available, durable KV store with secondary indexes that can reliably run on-premises.

How does CockroachDB fit?

- We need to store all the active (still not finally settled) and recent tickets (up to 30-90 days) so they are available quickly
  - CockroachDB automatically shards and scales horizontally so even when having 100M+ active tickets, we can just add more hardware to scale it and spread the load
- Part of wider pipeline architecture (based on Kafka), so data was already denormalized and in JSON format
  - Use JSONB + create computed columns (STORED)

How does CockroachDB fit? (pt. 2)

- All the querying is done by ticket ID and user ID
  - Use ticket ID as primary, create secondary index for user ID, no need to manually create and (even harder) maintain the secondary index
- Expected >10x growth over next 3 years
  - Based on our tests and load patterns, it would nicely scale just by adding more nodes

How does CockroachDB fit? (pt. 3)

- Data and processing had to reside in the licence country
  - Since there is no AWS in Romania and Poland, being able to run it on the premises in both locations was very important factor
  - Even now with more relaxed rules in the cloud era, having a copy of data in country of presence is still legal requirement in all jurisdictions
- Data integrity is extremely important
  - Strong consistency matters a lot when losing data can mean losing a licence; database level resiliency also matters much more when running on premises where hardware reliability is not guaranteed to be as high as in cloud

Additional benefits we liked

- Online schema changes
  - These are quite useful when using the computed columns from JSONB fields, as it allows you to easily introduce new fields when needed
- PostgreSQL compatible
  - We already had applications and instrumented in-house library wrappers for PostgreSQL → easier adoption
- Locality awareness and topology based data spreading
  - In simplest scenario, when it becomes legal we can deploy to cloud, and get automatic, up to date copy in the country
  - Also could be a long term gateway to multitenancy

~~Why I love it?~~ DevOps Benefits

- Kubernetes native (not that common in 2019)
  - Easily deployed via helm, managed the same as the rest of infrastructure, not relying on hacks and magic workarounds
- Comes fully automated with best practices
  - Out of the box TLS setup
  - Automatic rebalancing (when adding nodes and in failures)
  - Zero downtime upgrades (no need to do failovers)
- Scheduled backups directly to S3
  - Doing SQL dump on multi TB DB is not viable (MVCC limit); binary backup is significantly faster and more compact
  - Additionally, incremental backups on Enterprise plans

It went really well.
But some helpful tips.

Tips (on-prem)

- NTP Clocks synchronization
  - CRDB heavily depends on clocks being in sync (500ms)
  - Be careful of NTP sources you use, default NTP pool might not implement leap second smearing (in general just use Google/AWS ones)
- Make sure that you implement anti-affinity both on VM and Kubernetes level
  - Most common source of our clock drift crashes was VMware moves of VM from one physical node to another
  - This is not a problem if single pod gets moved

Tips (on-prem) #2

- Run on dedicated Kubernetes nodes and enable the CPU pinning, this yielded us ~40% performance gains compared to baseline
  - Use `cpuManagerPolicy: static` to enable CPU pinning
  - Then configure requests and limits to be the same resources; you need to configure it on all the containers (beware of init container)
  - CFS is not an issue because CockroachDB adjusts GOMAXPROCS
- Provisioning additional hardware on-prem can take a long time, so overprovision especially when starting
  - To speed up recovery rebalance of failed node, at cost of BW and CPU, tweak `kv.snapshot_rebalance.max_rate`

Tips (DevOps)

- Configure automatic backups
  - Even the free version includes scheduled (full) backups, there is no excuse not to run them and have them offsite (i.e., to AWS S3)
- Setup monitoring with Prometheus
  - Despite DB Console already having all the metrics and much more, given data is stored in the cluster itself, if you ever end up with broken cluster, having metrics outside can be really helpful for debugging
  - It also allows you to configure alerting (CockroachDB has good example with common ones available on the GitHub)

Lessons from running CockroachDB on-prem

What we got wrong?

- Using large JSONB is quite inefficient as it requires rewriting the whole rows on even small updates
  - Single large JSONB meant that we also couldn’t leverage the column families and do “partial” row updates
  - This was even bigger problem due to our large rows (avg. 4kB), their update frequency and how LSM tree works
  - JSONB was significantly slower than serializing on client side
- Backups blew the edge router licence throughput limits
  - Easy to fix since v22.1 – just configure the cloud storage rate limiting, note it is per node (remember when adding nodes)
  - `cloudstorage.s3.write.node_rate_limit = 4MiB`

```sql
CREATE TABLE tickets (
    id UUID PRIMARY KEY,
    ticket_status STRING AS (settlement_payload->>'status') STORED,
    settlement_payload JSONB,
    acceptance_payload JSONB,
    payout_payload JSONB,

    FAMILY f1 (
        id,
        ticket_status,
        settlement_payload
    ),
    FAMILY f2 (
        acceptance_payload,
        payout_payload
    )
)
```

id	ticket_status	settlement_payload	payout_payload	acceptance_payload

id	ticket_status	settlement_payload	payout_payload	acceptance_payload
UUID	**STRING**	**JSONB**	**JSONB**	**JSONB**

id	ticket_status	settlement_payload	payout_payload	acceptance_payload
🟧	🟧	🟧	🟦	🟦
🟧	🟧	🟧	🟦	🟦
🟧	🟧	🟧	🟦	🟦
UUID	**STRING**	🟩 **JSONB**	**JSONB**	**JSONB**
🟧	🟧	🟧	🟦	🟦
🟧	🟧	🟧	🟦	🟦
🟧	🟧	🟧	🟦	🟦
🟧	🟧	🟧	🟦	🟦
🟧	🟧	🟧	🟦	🟦

id	ticket_status	settlement_payload	payout_payload	acceptance_payload
UUID	STRING	JSONB	JSONB	JSONB

Tips

- While it is PSQL compatible syntax, avoid thinking of it as your “typical” relational database (RDBMS)
  - At some point, JOIN(s) just do not scale – query can too easily end up spawning the whole cluster
  - Thinking of it as infinitely scalable KV store from the start will help you better denormalize the data which will better scale in the long run
  - While some queries will be slower due to nature of distributed database, with properly designed schemas, you gain the benefit of linearly scaling out performance by just adding more nodes

Tips (pt. 2)

- Secondary indexes can be even more performant if they need to retrieve a small amount of data
    - By using STORED indexes it is possible to inline the fields in the index itself, avoiding the primary index lookup
- Denormalizing data and using JSONB
    - It is a tradeoff between ease of extracting more fields by just adding computed columns and overhead of processing JSON on each write
    - When storing larger payloads, consider the write patterns – you can gain significant performance benefits by leveraging column families

Tips (pt. 3)

- Read the documentation
  - Seriously, it is probably one of the best documentations for databases out there
- There is no magic, so familiarize yourself with the core concepts
  - CockroachDB is MVCC, this means that data is not deleted from database immediately – rather it depends on `gc.ttlseconds` (default 4h), or on backup execution
  - Read about how Cockroach stores the data and uses LSM trees, which will make it clear why some writes get expensive
- DB Console is your best friend

Optimizations & Migration to Cloud

Taking Over The CRDB Project

- Unfamiliar With CRDB
  - Had experience with SQL Databases
  - CRDB is PostgreSQL wire compatible
- How to configure CRDB?
  - Dashboard
  - Documentation
- Easy transition
  - Makes it possible to scale the team

Scaling

- Distributed Database
  - Resilient
  - Started out small
  - Cluster grew with demand both in node size and count
- Smallest Cluster
  - 3 nodes (4 vCPU each → 12 vCPU total)
- Biggest Cluster
  - 24 nodes (16 vCPU each → 384 vCPU total)

Migration to AWS

- What Instance Type should we choose?
  - CRDB Benchmarking
- There is no static CPU policy (CPU pinning) on our AWS clusters
  - Dramatic performance difference compared to on-prem
- How to migrate the data?
  - Leveraging Kafka
  - Leveraging CRDB Performance

Migration to AWS

- Main source of truth is the CRDB database
- Actions/Changes are propagated through Kafka Topics

- Options for Cluster Migration:
  - Backup & Restore
  - CRDB Physical Cluster Replication
  - Custom Solution

Backup & Restore

Pros:

- Easy to do
- No Logic required
- Would have all the data

Cons:

- Long Downtime
  - Turn off system
  - Make backup
  - Transfer data
  - Restore
  - Switch system
- Big Bang Release
- Actions are not transferred
- Reverting would be tricky

CRDB Physical Cluster Replication

Pros:

- Would have all the data
- Continuous process

Cons:

- Downtime
- Actions are not transferred
- Stable connection to AWS

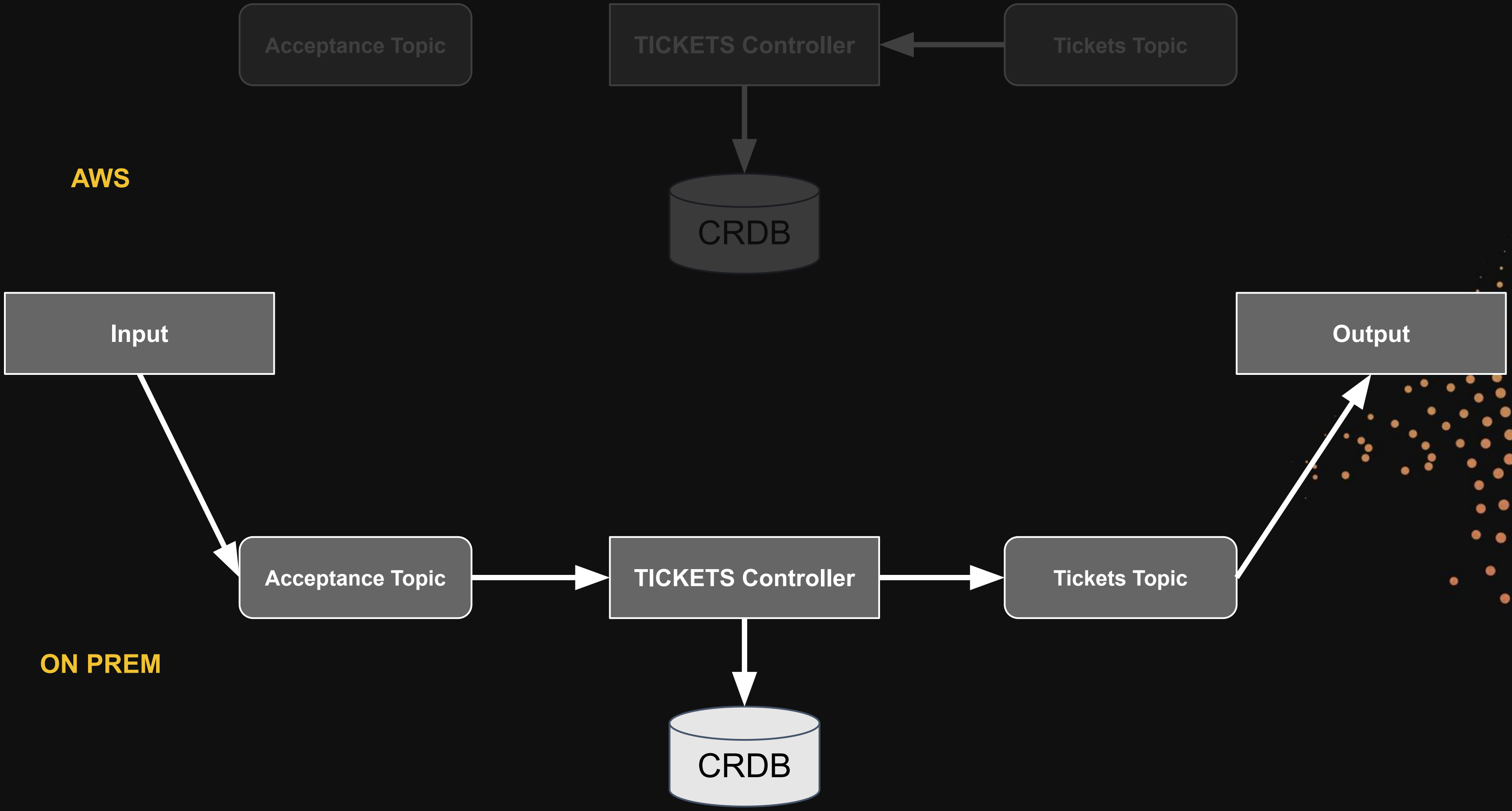
Custom Solution

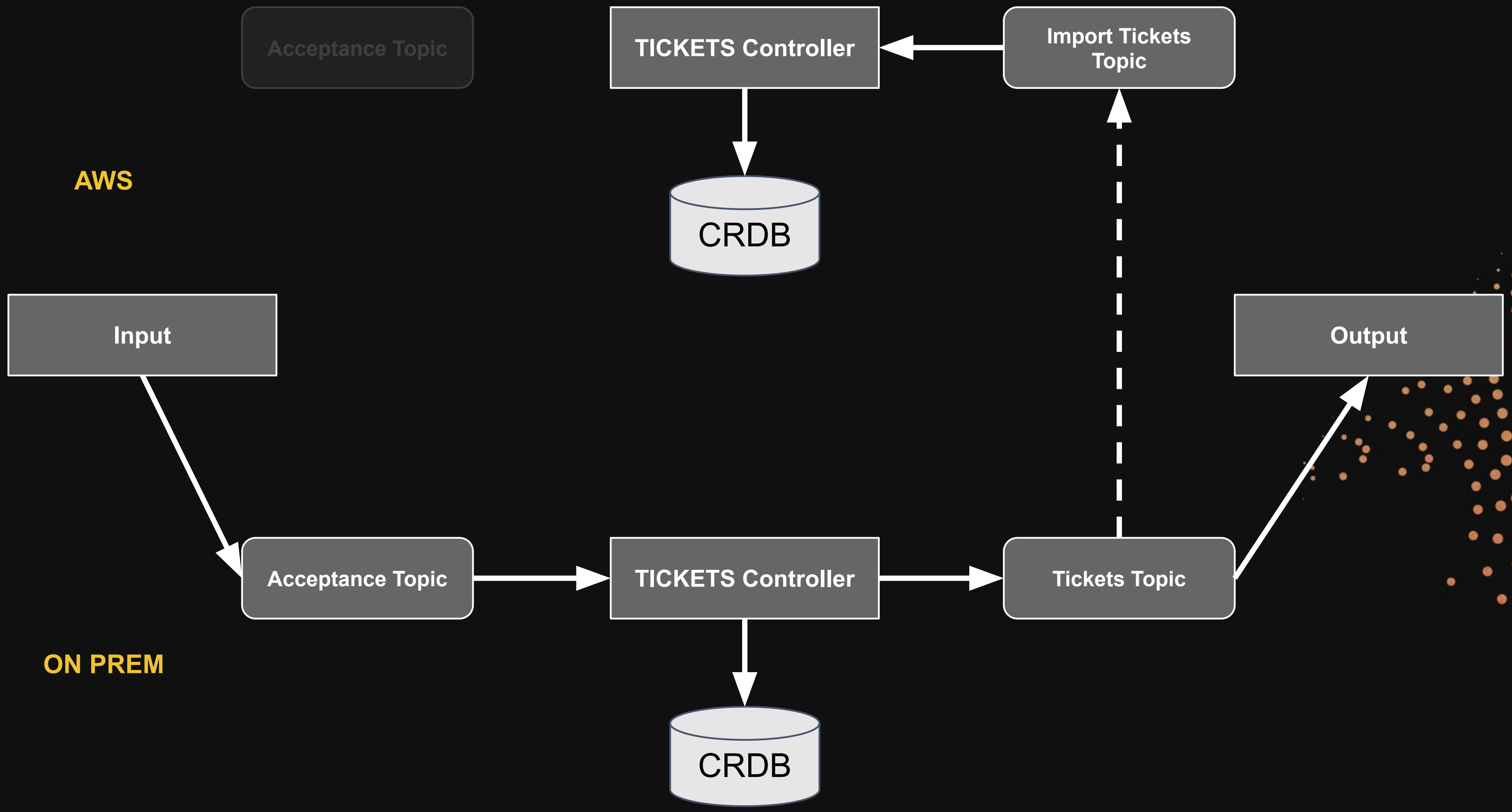
Pros:

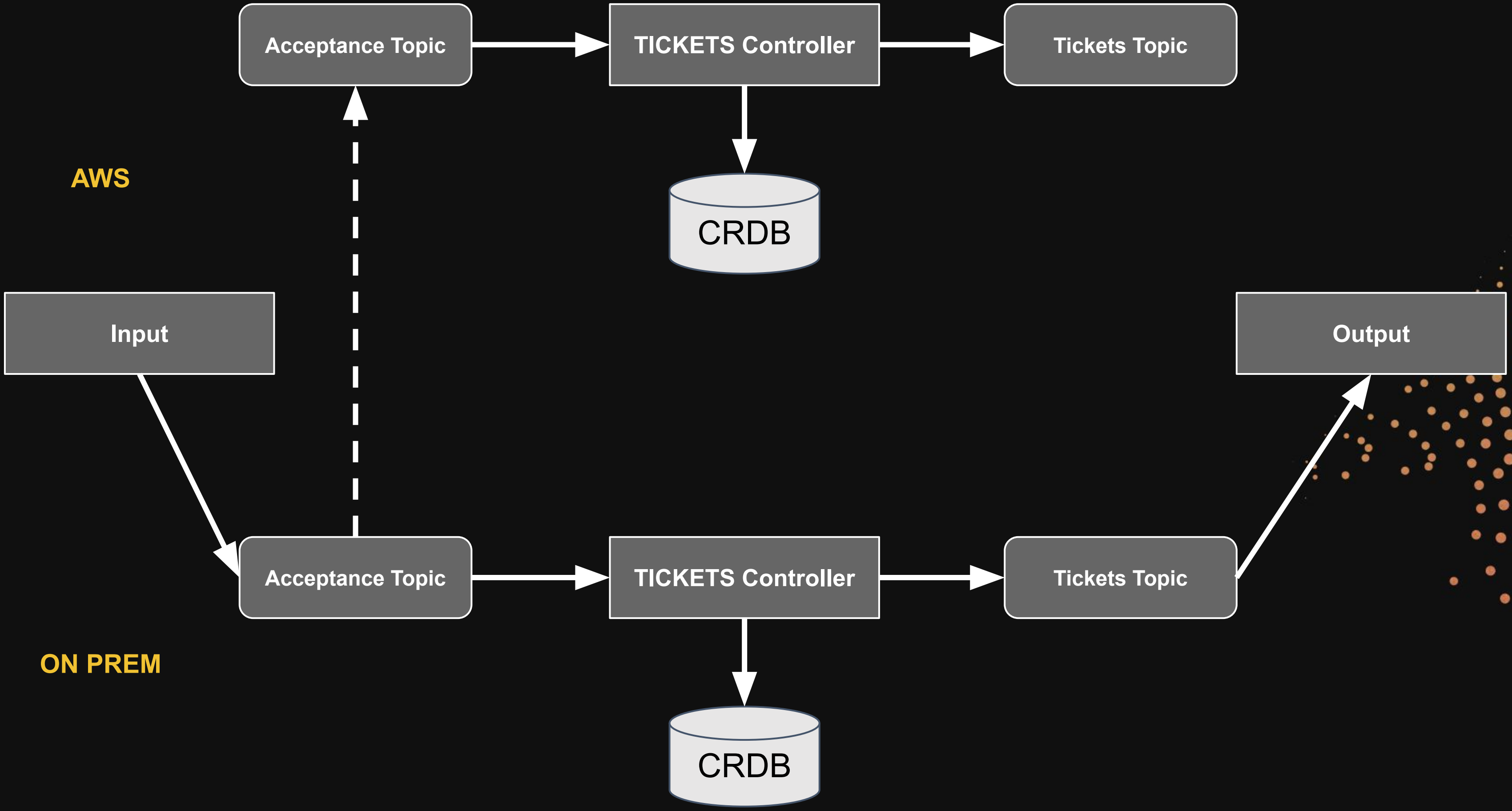
- No downtime
- Partial rollouts
- Action history would be copied over
- Rollback would be easy

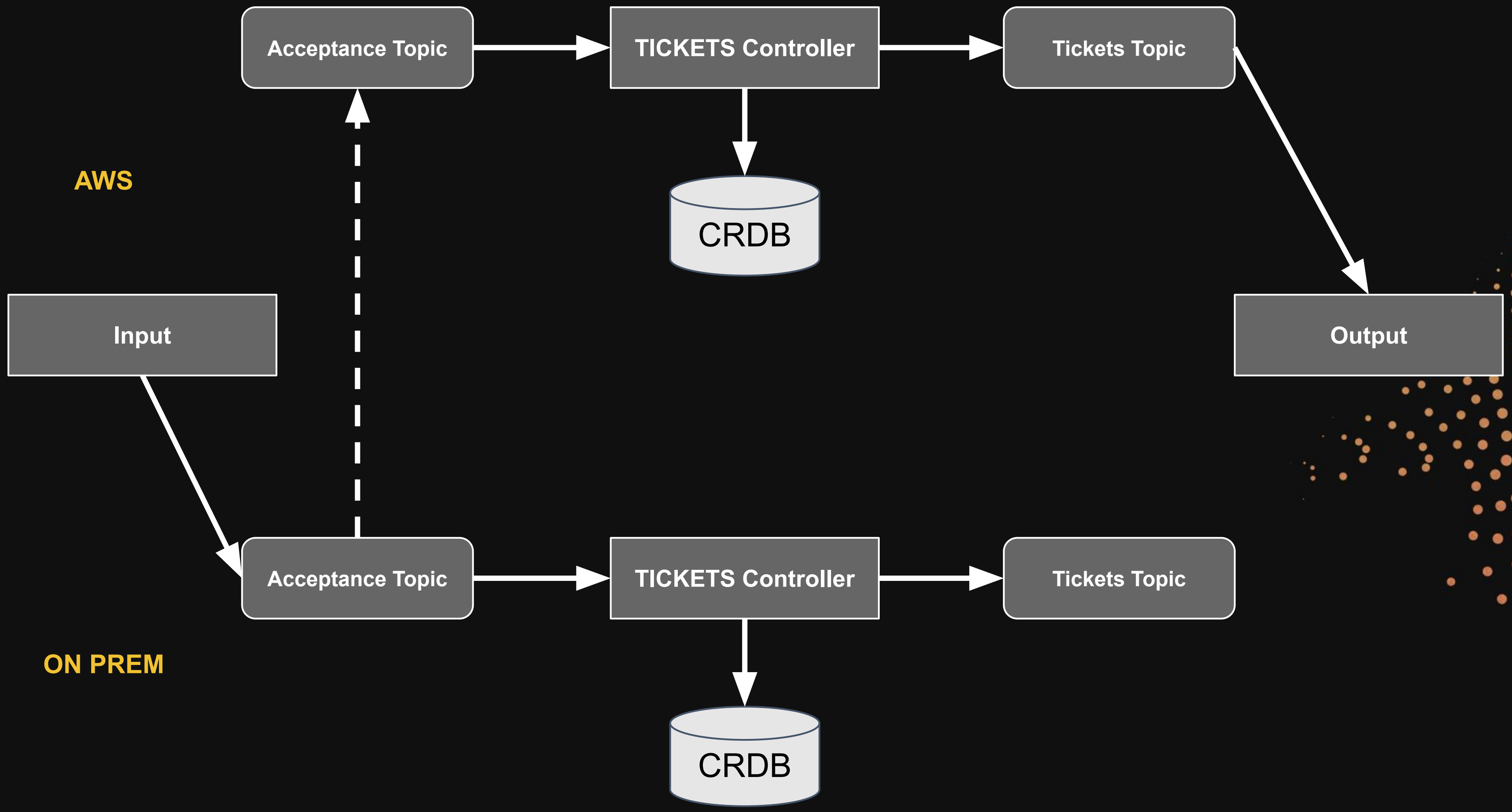
Cons:

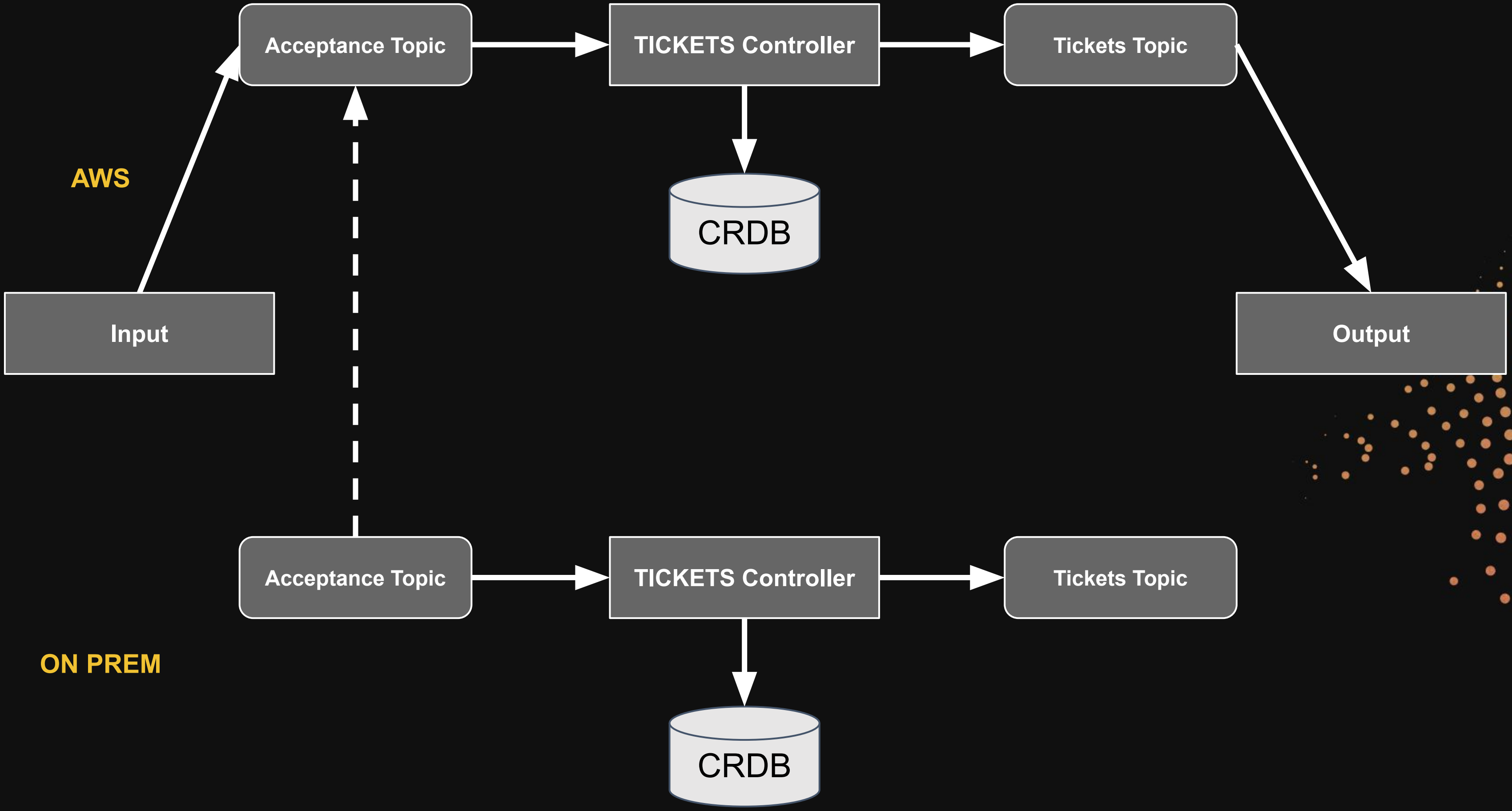
- Additional logic required
- Stable Internet connection

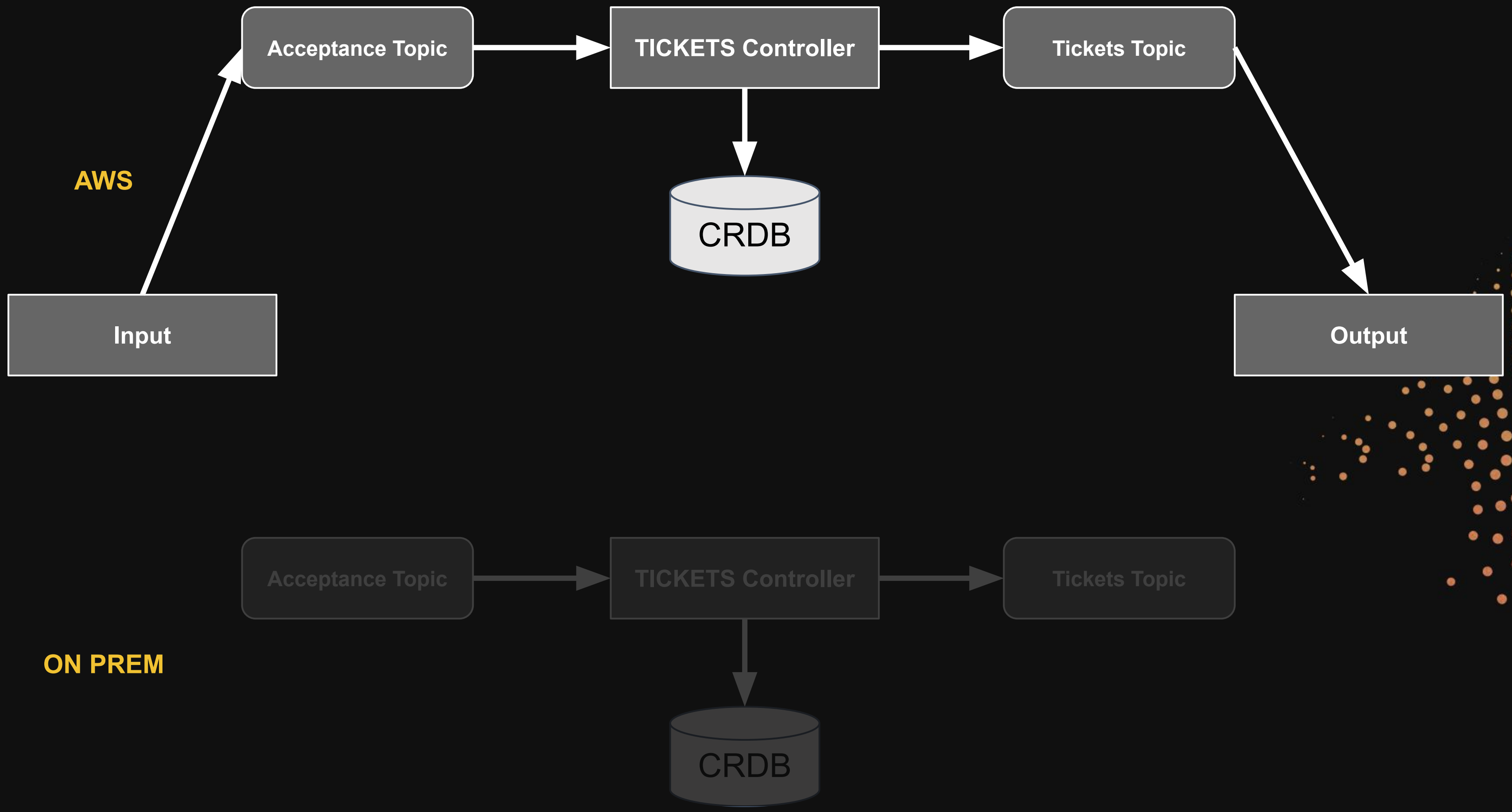

Acceptance Topic
TICKETS Controller
Tickets Topic
CRDB
AWS
Input
Output
Acceptance Topic
TICKETS Controller
Tickets Topic
ON PREM
CRDB


Acceptance Topic
TICKETS Controller
Import Tickets Topic
AWS
CRDB
Input
Output
Acceptance Topic
TICKETS Controller
Tickets Topic
ON PREM
CRDB


Acceptance Topic
TICKETS Controller
Tickets Topic
AWS
CRDB
Input
Output
Acceptance Topic
TICKETS Controller
Tickets Topic
ON PREM
CRDB


AWS
Acceptance Topic
TICKETS Controller
Tickets Topic
CRDB
Input
Output
Acceptance Topic
TICKETS Controller
Tickets Topic
CRDB
ON PREM


AWS
Input
Acceptance Topic
TICKETS Controller
CRDB
Tickets Topic
Output
ON PREM
Acceptance Topic
TICKETS Controller
CRDB
Tickets Topic

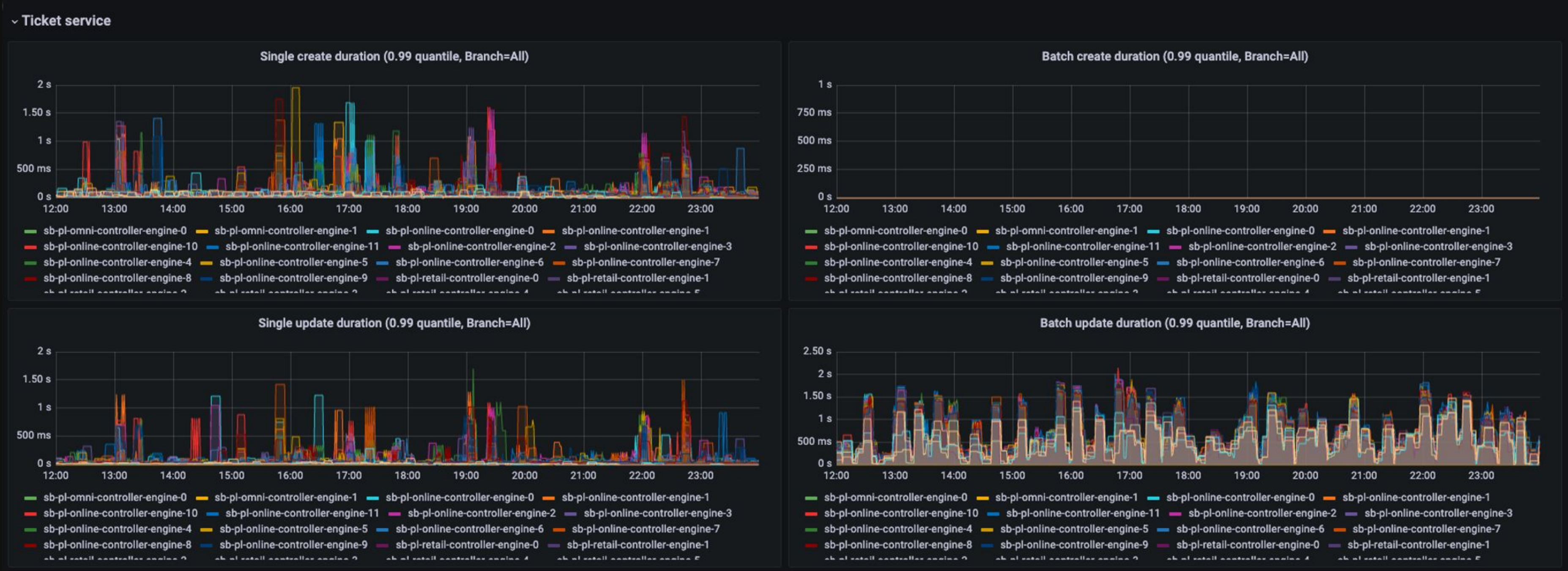

Acceptance Topic
TICKETS Controller
Tickets Topic
AWS
CRDB
Input
Output
Acceptance Topic
TICKETS Controller
Tickets Topic
ON PREM
CRDB

Schema optimisation

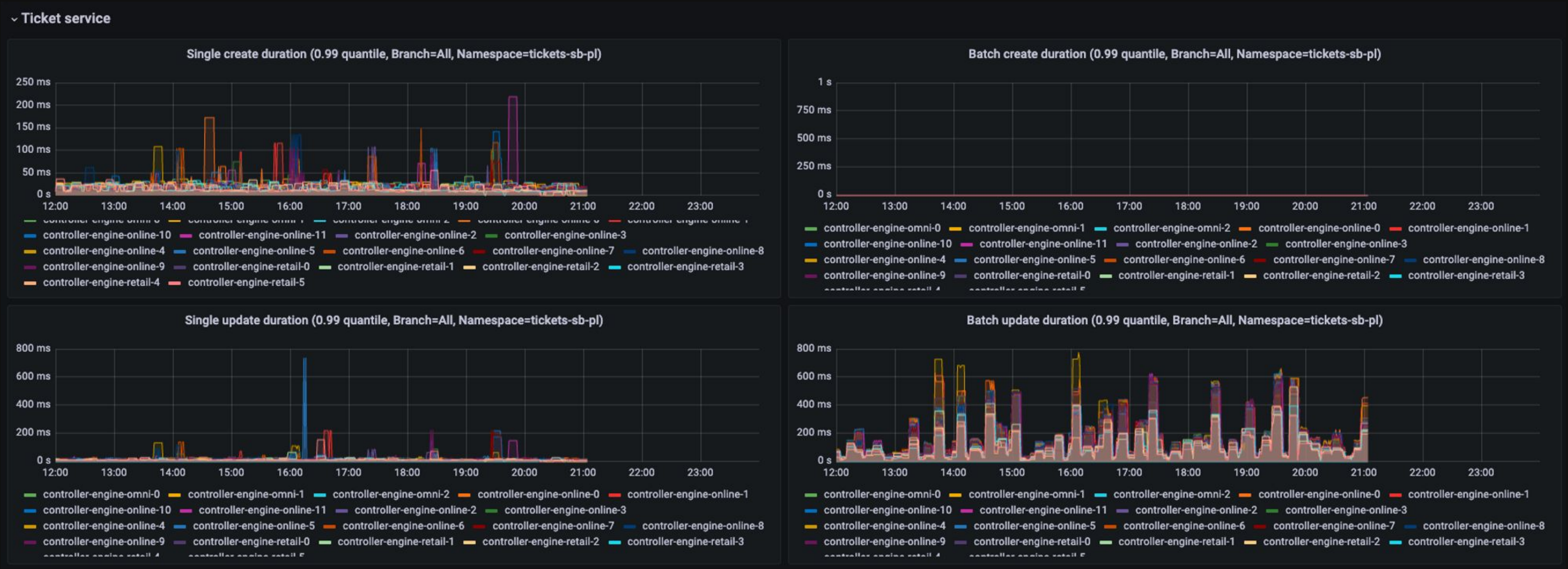
```sql
CREATE TABLE tickets (
     -- Computed fields
     id UUID PRIMARY KEY AS ((payload->>'id')::UUID) STORED,
     code STRING NOT NULL AS (payload->>'code') STORED,
     customer_id STRING AS (payload->>'customer'->>'id') STORED,
     customer_external_id INT AS ((payload->>'customer'->>'externalId')::INT) STORED,
     accept_time TIMESTAMP NOT NULL AS (parse_timestamp (payload->>'acceptTime')) STORED,
     modify_time TIMESTAMP NOT NULL AS (parse_timestamp (payload->>'modifyTime')) STORED,
     event_type STRING AS (payload->>'eventType') STORED,
     market_type STRING AS (payload->>'marketType') STORED,
     ticket_status STRING AS (payload->>'status') STORED,
     selection_count_type STRING AS (payload->>'selectionCountType') STORED,

     -- One big JSON payload is source of truth
     payload JSONB NOT NULL
);
```

```sql
CREATE TABLE tickets (
     id UUID PRIMARY KEY,
     modify_time TIMESTAMP NOT NULL,
     priority STRING NOT NULL,

     -- Manually extracted fields used for indexes and filtering
     code STRING NOT NULL,
     customer_external_id STRING,
     customer_id STRING,
     ...
     -- Binary fields to store Protobuf, faster serialization/deserialization and compact
     acceptance_payload BYTES,
     settlement_payload BYTES,
     selections_payload BYTES,
     payout_payload BYTES,

     -- Column families to reduce amount of whole row updates
     FAMILY main(id, modify_time, priority, ...,  branch),
     FAMILY acceptance(acceptance_payload),
     FAMILY settlement(settlement_payload),
     FAMILY selections(selections_payload),
     FAMILY payout(payout_payload)
);
```

Before


Ticket service
Single create duration (0.99 quantile, Branch=All)
2 s
1.50 s
1 s
500 ms
0 s
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
Batch create duration (0.99 quantile, Branch=All)
1 s
750 ms
500 ms
250 ms
0 s
Single update duration (0.99 quantile, Branch=All)
Batch update duration (0.99 quantile, Branch=All)
2.50 s
2 s
1.50 s
1 s
500 ms
0 s
sb-pl-omni-controller-engine-0
sb-pl-omni-controller-engine-1
sb-pl-online-controller-engine-0
sb-pl-online-controller-engine-1
sb-pl-online-controller-engine-10
sb-pl-online-controller-engine-11
sb-pl-online-controller-engine-2
sb-pl-online-controller-engine-3
sb-pl-online-controller-engine-4
sb-pl-online-controller-engine-5
sb-pl-online-controller-engine-6
sb-pl-online-controller-engine-7
sb-pl-online-controller-engine-8
sb-pl-online-controller-engine-9
sb-pl-retail-controller-engine-0
sb-pl-retail-controller-engine-1

After



Ticket service
Single create duration (0.99 quantile, Branch=All, Namespace=tickets-sb-pl)
250 ms
200 ms
150 ms
100 ms
50 ms
0 s
12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00
controller-engine-online-10
controller-engine-online-11
controller-engine-online-2
controller-engine-online-3
controller-engine-online-4
controller-engine-online-5
controller-engine-online-6
controller-engine-online-7
controller-engine-online-8
controller-engine-online-9
controller-engine-retail-0
controller-engine-retail-1
controller-engine-retail-2
controller-engine-retail-3
controller-engine-retail-4
controller-engine-retail-5
Batch create duration (0.99 quantile, Branch=All, Namespace=tickets-sb-pl)
1 s
750 ms
500 ms
250 ms
0 s
12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00
controller-engine-omni-0
controller-engine-omni-1
controller-engine-omni-2
controller-engine-online-0
controller-engine-online-1
controller-engine-online-10
controller-engine-online-11
controller-engine-online-2
controller-engine-online-3
controller-engine-online-4
controller-engine-online-5
controller-engine-online-6
controller-engine-online-7
controller-engine-online-8
controller-engine-online-9
controller-engine-retail-0
controller-engine-retail-1
controller-engine-retail-2
controller-engine-retail-3
Single update duration (0.99 quantile, Branch=All, Namespace=tickets-sb-pl)
800 ms
600 ms
400 ms
200 ms
0 s
12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00
controller-engine-omni-0
controller-engine-omni-1
controller-engine-omni-2
controller-engine-online-0
controller-engine-online-1
controller-engine-online-10
controller-engine-online-11
controller-engine-online-2
controller-engine-online-3
controller-engine-online-4
controller-engine-online-5
controller-engine-online-6
controller-engine-online-7
controller-engine-online-8
controller-engine-online-9
controller-engine-retail-0
controller-engine-retail-1
controller-engine-retail-2
controller-engine-retail-3
Batch update duration (0.99 quantile, Branch=All, Namespace=tickets-sb-pl)
800 ms
600 ms
400 ms
200 ms
0 s
12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00
controller-engine-omni-0
controller-engine-omni-1
controller-engine-omni-2
controller-engine-online-0
controller-engine-online-1
controller-engine-online-10
controller-engine-online-11
controller-engine-online-2
controller-engine-online-3
controller-engine-online-4
controller-engine-online-5
controller-engine-online-6
controller-engine-online-7
controller-engine-online-8
controller-engine-online-9
controller-engine-retail-0
controller-engine-retail-1
controller-engine-retail-2
controller-engine-retail-3

What is next?

- Multitenancy
  - Improved resource utilization
  - Faster time to new markets
  - Faster feature development

Questions?