#ASLI ENGINEERING

How Shopify balances the shard without downtime

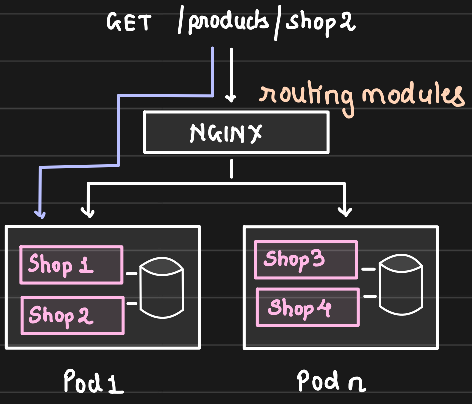
SWIPE

BY

ARPIT BHAYANI

How Shopify rebalances shards without any downtime

People can host their shops on Shopify and they use MySQL as their database.



Current Architecture

1. Shops are distributed across 'pods'
2. All shops in a pod share a database
3. Request come to NGINX proxy, and it routes it to the corresponding pod.

Every row in table has a column 'shop_id' that tells which shop it belongs to

Moving shop from one pod to another

↳ iterate through all the tables

↳ pick rows with specific shop_id

↳ move those rows to a db of another pod

Challenge: Do this without any downtime!!

Why do we need to move?

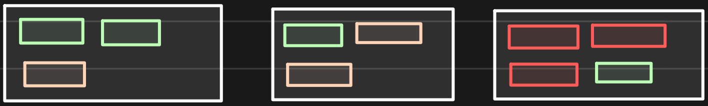
Resource intensive shops on the same shard may

1. risk failure due to over-utilization
2. inconsistent database utilization across shards

ARPIT BHAYANI

How to decide which shop lives in which shard ?

Distribution based on number of shops is not a good idea because we may end up having two 'heavy' shops on one shard.

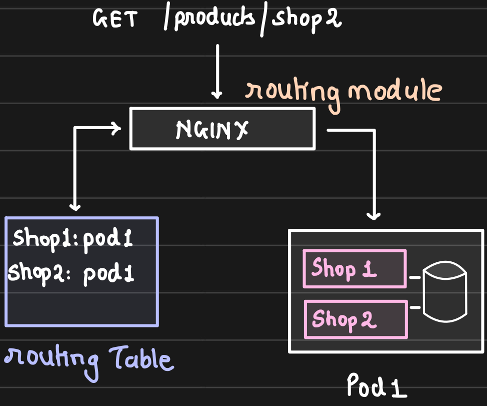


The way we decide depens on 'heuristics' we want to apply

1. historical database utilization
2. historical traffic on the shop
3. forecasting (private request)

Data Science Team decides the optimal distribution based on these factors

Moving the shops



Critical constraints

1. Shop must be entirely available
2. No data loss or corruption
3. No unnecessary strain on infra

Three high-level phases

1. Batch copy and Tail Binlog
2. Cutover
3. Update routing table

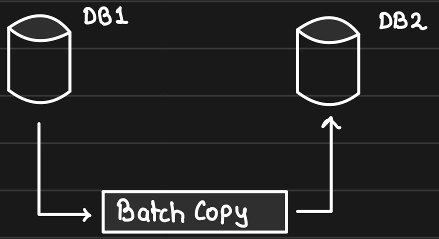
ARPIT BHAYANI

Phase 1: Batch copy and tail Binlog

Shopify uses an internal tool (also opensourced) named Ghostferry.

1. Go through tables and pick rows with the 'shop_id' and write them to another database.

in a transaction

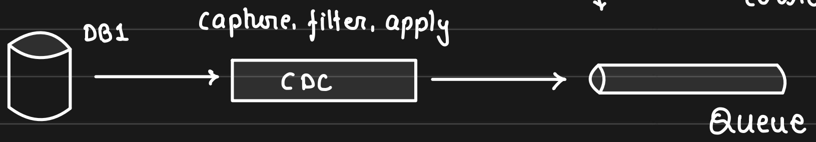


2. While batch copy is happening, keep track of newer changes happening on DB by consuming Binlog (write ahead log)

* we need to filter out enteries for shops that don't interest us.

{ Insert, orders, (...),
Update, orders, (....),
Insert, orders, (...),
}

or just note the binlog coordinates



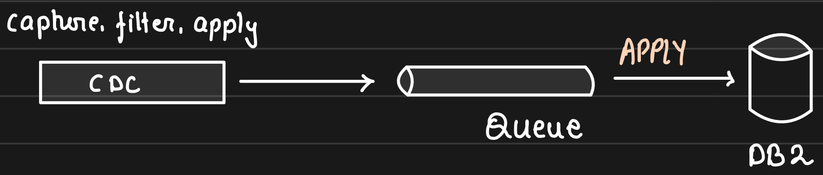
To speed up - Read multiple tables in parallel

While batch copy and tailing happens, our DB continues to serve requests.

ARPIT BHAYANI

Phase 2: Prepare for cutover

Once batch copy is complete, consume all the newer writes through Binlog



Wait until the 'lag' is down to seconds (near-realtime)

i.e. we are almost done consuming the queue and newer events are almost immediately consumed.

The writes to source DB is stopped !!

(very short duration ~ one/two seconds)

[application logic has retries]

The source DB's binlog coordinate are recorded and as soon as target DB reaches that we say replication done.

At this stage: 1. no new writes to source DB

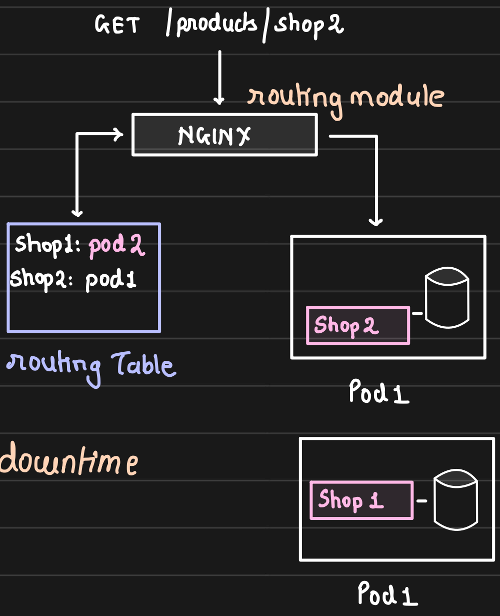
2. source db = target db

ARPIT BHAYANI

Phase 3: Update routing table

Once we have confidence of no dataloss, we update routing table and traffic is switched on.

New request thus flows to the new pod.



* Cutover is completed in a very short window to minimize downtime

Next Steps

1. Validate and verify the correctness
2. prune the data from old database

ARPIT BHAYANI