Local-first shopping list application built with CRDTs

- Requisites

- ❑ Users can create a new shopping list and share with other users
- ❑ List can be stored locally and be on cloud
- ❑ Users can concurrently change the lists and its items
- ❑ Data should be shared to distribute workload among different nodes

Local-first shopping list application built with CRDTs

- Client

- A client is able to create, edit and delete shopping lists through the use of different endpoints served by the cloud's API.
- As a local-first application, shopping lists are stored locally prior to attempts of synchronization with the cloud. Upon creation, an unique hash is assigned to them so that it can be used as an ID in order to perform operations on it.
- Synchronization with the server is attempted on every operation performed.
- A user also has the possibility of listing all lists that are being locally stored

- Load balancer

- For load balancing purposes, ***traefik*** (https://traefik.io) is used
- It employs the ***round robin algorithm*** for scheduling processes, effectively ensuring no one node is overworked
- Its decoupled nature makes it so that changing to another load balancer does not make hours of setup useless
- Matched up against other known balancers, such as ***nginx***, it suffers in terms of performance but provides in simplicity of use

Local-first shopping list application built with CRDTs

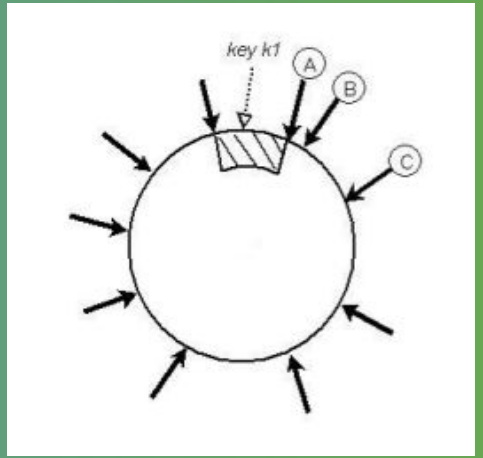
- Cloud

- ❑ Cloud-side architecture, much in the same way that the Amazon Dynamo is designed
- ❑ Lists are stored as key-value pairs

- ❑ Each node is identified by an hash
- ❑ A hash is applied on each list and becomes its unique identifier
- ❑ These hashes are used to determine the node that should
- ❑ became the owner of the list

- ❑ The owner node then replicates its lists to the next N (replication factor) nodes

- ❑ The node number are variable and the lists are reallocated when
- ❑ this number changes.



- CRDTs

- Conflict-Free Replicated Data Types are data structures that allow for concurrent changes without worries of data loss.
- For the sake of the platform, **three** distinct, **non-delta**, CRDTs were implemented.
- **CCounter**, or Causal Counter, which uses the concept of causal context to keep track of changes to the counter and update it accordingly.
- An **AWORMap**, or Add-Wins Observed Remove Map, which prioritizes the addition of keys over their removal, was implemented using a set to store the keys in use as well as a causal context for proper handling of key merging.
- While not a standard CRDT, the **DotKernel** CRDT is similar to an **AWORSet**, meant to be included in other CRDTs as a way to automate the process of handling causality, and is a key part of both the **CCounter** and the **AWORMap**.

- Solution Analysis

- ❏ Ideally, these components represent the architecture for the best possible implementation of a platform such as this.
- ❏ Unfortunately, linking these distinct components presents a real challenge due to the convoluted nature of each of them.
- ❏ Currently, due to these issues, CRDTs have issues in performing joins with other replicas in the cloud, and work similarly to a **last-writer-wins** protocol, making the assumption that the last version of the list written to the cloud is the also the most recent.