Key-Value Store

Participants:

Daniel Walder - 015153159
Krisztián Ádám - 015177287
Jun Ma - 015092540

GitHub: https://github.com/wald3r/ds_keyvaluestore
Video1: vid_updating_feature.mkv
Video2: vid_shared_state.mkv

Description:

We developed a key-value store, using MongoDB, NodeJS and Kafka. Multiple instances of the application node appear as they would share a database. For example: sharing user setting values between multiple instances of a service.

Design:

We have 2 types of Nodes: Kafka and the key-value store service. Running 2 instances of the latter one.

Key-value store service: Standard React + Node + MongoDB application (3 processes), with a simple interface to create, delete and modify key-value pairs.

Kafka: a simple Kafka process managed by a Zookeeper process. Distributes messages between the service instances.

Communication: Kafka service is used to keep the data in different nodes synchronized. Consequently 2 application nodes appear as they would share one database. On every database action (create, update, delete), the node sends a description of the action to the Kafka topic, then the other node(s) execute the same action.

Challenges:

- Using Kafka group names and topics properly
- Preventing duplicates on the node that creates the kv pair
- Settings up networking between VMs
- Virtualization took us some time till all nodes were able to communicate fully
- A problem still is that the view of the nodes doesn't get updated automatically. So if changes happen on the other node, the user would have to reload the page to see the changes.

Installation & running:

If you want to run all of them on the same machine, you can jump directly to “Run”.

Installation (replace <?> with 1 or 2 depending on which node you want to edit) :

Update the IP addresses (all of them can be localhost if running on the same machine):

- docker-compose-kafka.yml
  - KAFKA_ADVERTISED_HOST_NAME : IP of the node running Kafka
- docker-compose-node<?>.yml
  - KAFKA : IP of the node running Kafka
- frontend<?>/src/services/pair.js
  - basicUrl : IP of the node running the backend<?> for this frontend<?>

Run:

- docker-compose --file docker-compose-kafka.yml up -d
- docker-compose --file docker-compose-node1.yml up -d
- docker-compose --file docker-compose-node2.yml up -d

  OR ALTERNATIVELY to start all at once

- docker-compose --file docker-compose-all.yml up

Tests:

Minimum Payload: 10 bytes
Average Payload: 256 bytes
Maximum Payload: 512 bytes

	Random Payload	Min Payload	Avg Payload	Max Payload
Difference	13,12 ms	19,12 ms	13,28 ms	12,96 ms

We measured how long it takes kafka two update the second database when data gets stored in the first database. The time span is minimal and you can’t tell much of a difference among the various payloads. Minimum Payload is around 6ms higher than the others, which is most probably just a coincidence.

Discussion:

When it comes to reliability our architecture offers some advantages. There are 2 databases which keep each other up-to-date, which equals a backup. Even if kafka fails, both databases can keep working independently.
Another advantage is when a new node joins, the new node gets updated automatically by kafka. So it is easy extendable with more nodes or to recover nodes from a failure.

Although our system is not 100% fault tolerant, then when one database fails the frontend of it can’t fetch data anymore, until the database has been restored.

Overall our architecture can be used for any kind of application which needs a reliable distributed key-value database.