Deuxfleurs Association

`https://garagehq.deuxfleurs.fr/`
Matrix channel: `#garage:deuxfleurs.fr`

Our objective at Deuxfleurs

Promote self-hosting and small-scale hosting as an alternative to large cloud providers

Our objective at Deuxfleurs

Promote self-hosting and small-scale hosting as an alternative to large cloud providers

Why is it hard?

Our objective at Deuxfleurs

Promote self-hosting and small-scale hosting
as an alternative to large cloud providers

Why is it hard?

Resilience

(we want good uptime/availability with low supervision)

How to make a stable system

Enterprise-grade systems typically employ:

- RAID
- Redundant power grid + UPS
- Redundant Internet connections
- Low-latency links
- ...

→ it's costly and only worth it at DC scale

How to make a resilient system

Instead, we use:

- Commodity hardware (e.g. old desktop PCs)

How to make a resilient system

How to make a resilient system

How to make a resilient system

Instead, we use:

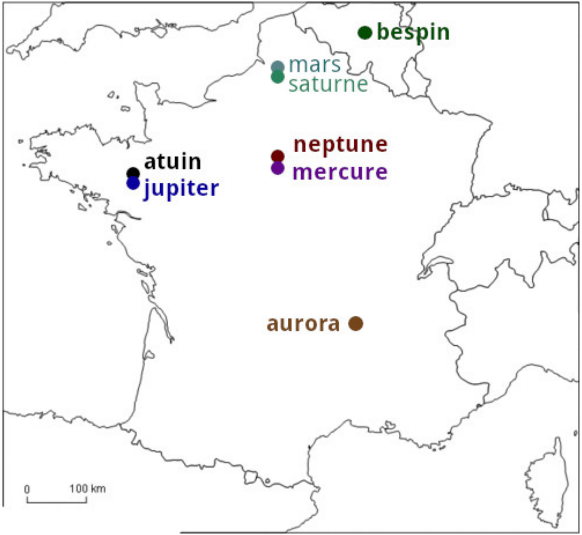
- Commodity hardware (e.g. old desktop PCs)
- Commodity Internet (e.g. FTTB, FTTH) and power grid

How to make a resilient system

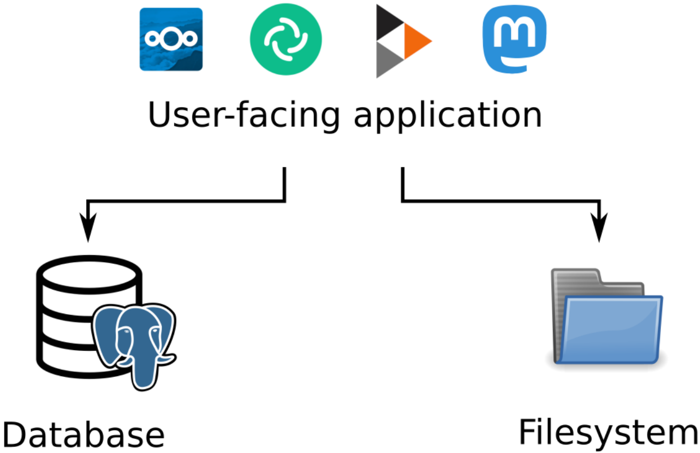
Instead, we use:

- Commodity hardware (e.g. old desktop PCs)
- Commodity Internet (e.g. FTTB, FTTH) and power grid
- **Geographical redundancy** (multi-site replication)

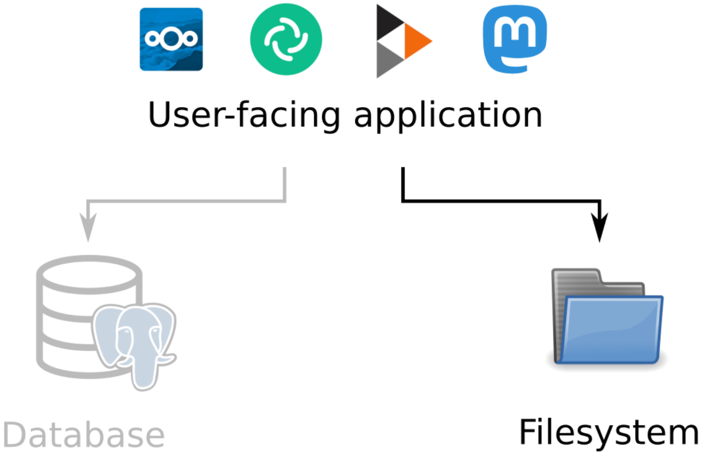
How to make a resilient system



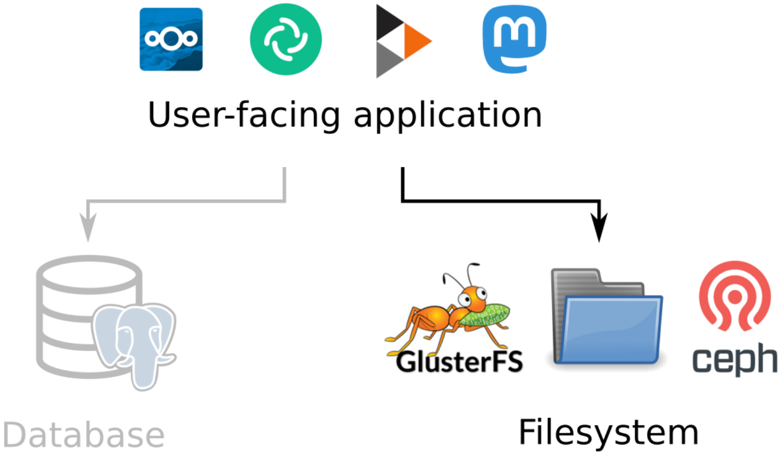
How to make this happen



How to make this happen



How to make this happen



Distributed file systems are slow

File systems are complex, for example:

- Concurrent modification by several processes
- Folder hierarchies
- Other requirements of the POSIX spec

Coordination in a distributed system is costly

Costs explode with commodity hardware / Internet connections
(we experienced this!)

A simpler solution: object storage

Only two operations:

- Put an object at a key
- Retrieve an object from its key

(and a few others)

Sufficient for many applications!

A simpler solution: object storage

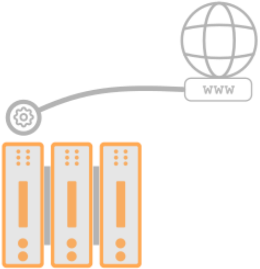
S3: a de-facto standard, many compatible applications

MinIO is self-hostable but not suited for geo-distributed deployments

But what is Garage, exactly?

Garage is a self-hosted drop-in replacement for the Amazon S3 object store

that implements resilience through geographical redundancy on commodity hardware

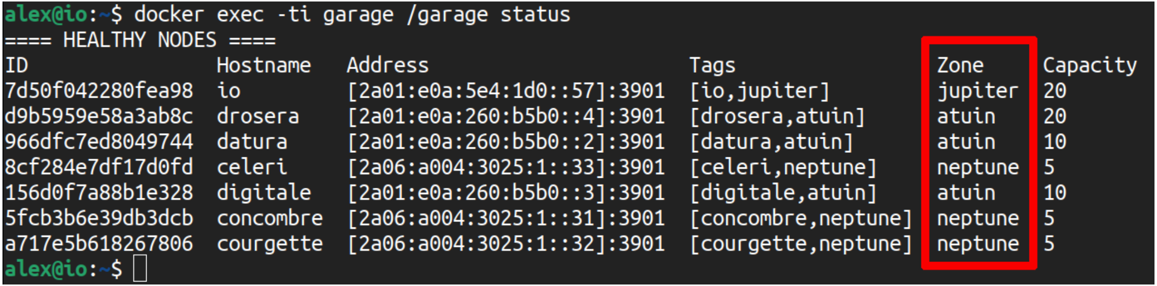


Host a website

Store Media

Backup Target

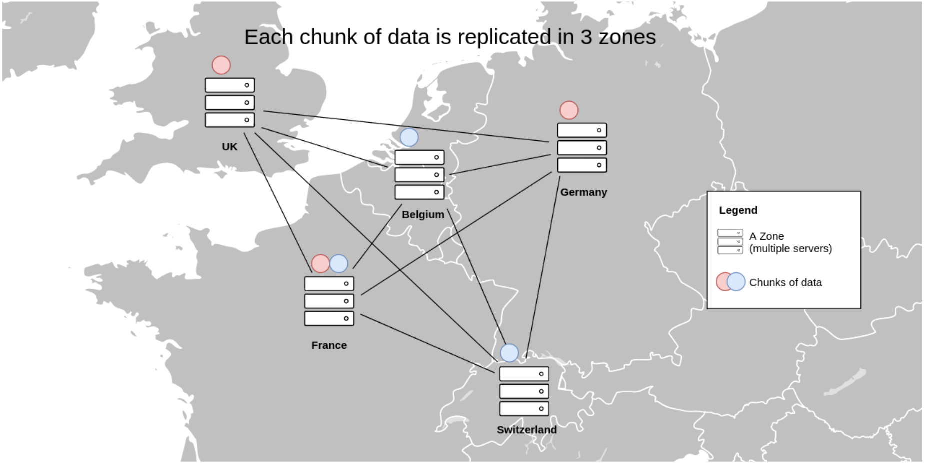
Garage is *location-aware*



```
alex@io:~$ docker exec -ti garage /garage status
==== HEALTHY NODES ====
ID                Hostname   Address                      Tags                 Zone     Capacity
7d50f042280fea98  io         [2a01:e0a:5e4:1d0::57]:3901  [io,jupiter]         jupiter  20
d9b5959e58a3ab8c  drosera    [2a01:e0a:260:b5b0::4]:3901  [drosera,atuin]      atuin    20
966dfc7ed8049744  datura     [2a01:e0a:260:b5b0::2]:3901  [datura,atuin]       atuin    10
8cf284e7df17d0fd  celeri     [2a06:a004:3025:1::33]:3901  [celeri,neptune]     neptune  5
156d0f7a88b1e328  digitale   [2a01:e0a:260:b5b0::3]:3901  [digitale,atuin]     atuin    10
5fcb3b6e39db3dcb  concombre  [2a06:a004:3025:1::31]:3901  [concombre,neptune]  neptune  5
a717e5b618267806  courgette  [2a06:a004:3025:1::32]:3901  [courgette,neptune]  neptune  5
alex@io:~$ 
```

Garage replicates data on different zones when possible

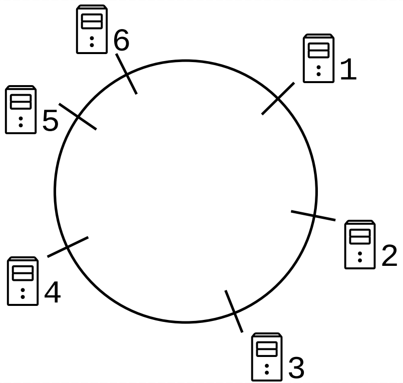
Garage is *location-aware*



Each chunk of data is replicated in 3 zones

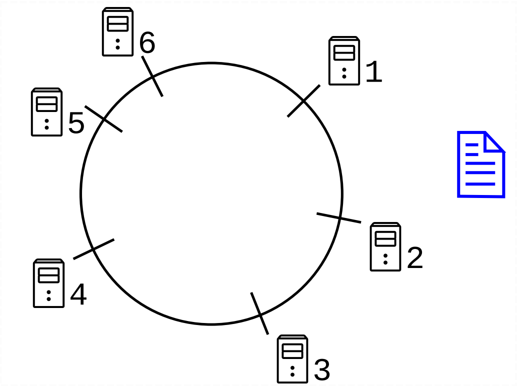
How to spread files over different cluster nodes?

Consistent hashing (DynamoDB):



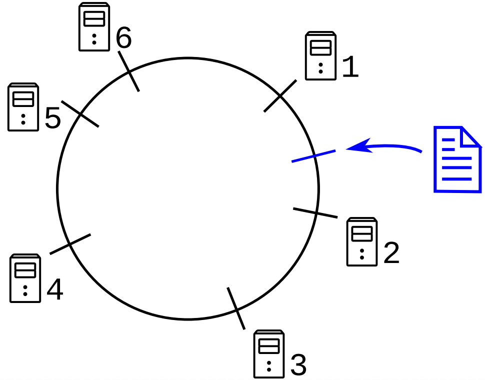
How to spread files over different cluster nodes?

Consistent hashing (DynamoDB):



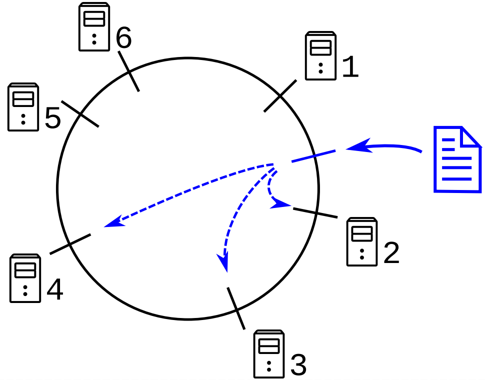
How to spread files over different cluster nodes?

Consistent hashing (DynamoDB):



How to spread files over different cluster nodes?

Consistent hashing (DynamoDB):



How to spread files over different cluster nodes?

Issues with consistent hashing:

- Doesn't dispatch data based on geographical location of nodes

How to spread files over different cluster nodes?

Issues with consistent hashing:

- Doesn't dispatch data based on geographical location of nodes
- Geographically aware adaptation, try 1:
  data quantities not well balanced between nodes

How to spread files over different cluster nodes?

Issues with consistent hashing:

- Doesn't dispatch data based on geographical location of nodes
- Geographically aware adaptation, try 1:
  data quantities not well balanced between nodes
- Geographically aware adaptation, try 2:
  too many reshuffles when adding/removing nodes

How to spread files over different cluster nodes?

Garage's method: build an index table

Realization: we can actually precompute an optimal solution

How to spread files over different cluster nodes?

Garage's method: build an index table

Realization: we can actually precompute an optimal solution

Partition	**Node 1**	**Node 2**	**Node 3**
Partition 0	Io (jupiter)	Drosera (atuin)	Courgette (neptune)
Partition 1	Datura (atuin)	Courgette (neptune)	Io (jupiter)
Partition 2	Io(jupiter)	Celeri (neptune)	Drosera (atuin)
⋮	⋮	⋮	⋮
Partition 255	Concombre (neptune)	Io (jupiter)	Drosera (atuin)

How to spread files over different cluster nodes?

Garage's method: build an index table

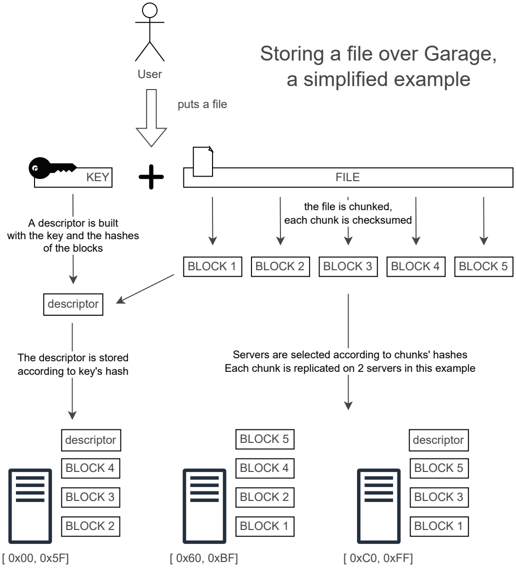
Realization: we can actually precompute an optimal solution

Partition	**Node 1**	**Node 2**	**Node 3**
Partition 0	Io (jupiter)	Drosera (atuin)	Courgette (neptune)
Partition 1	Datura (atuin)	Courgette (neptune)	Io (jupiter)
Partition 2	Io(jupiter)	Celeri (neptune)	Drosera (atuin)
⋮	⋮	⋮	⋮
Partition 255	Concombre (neptune)	Io (jupiter)	Drosera (atuin)

The index table is built centrally using an optimal* algorithm,
then propagated to all nodes

*not yet optimal but will be soon

Storing and retrieving files



Storing a file over Garage, a simplified example

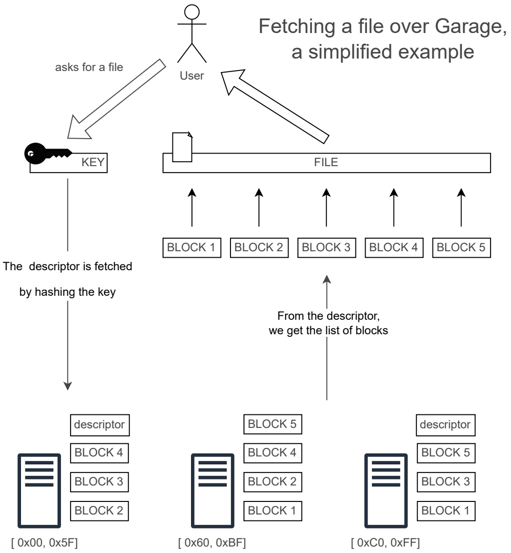
A descriptor is built with the key and the hashes of the blocks

the file is chunked, each chunk is checksumed

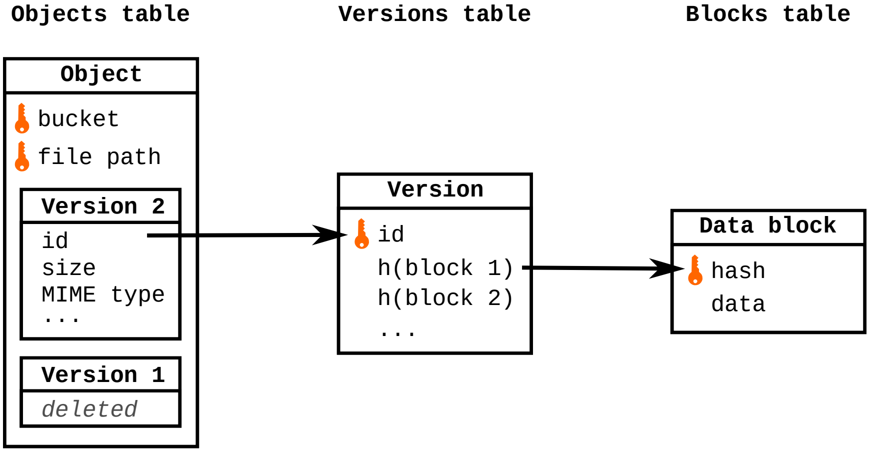
The descriptor is stored according to key's hash

Servers are selected according to chunks' hashes
Each chunk is replicated on 2 servers in this example

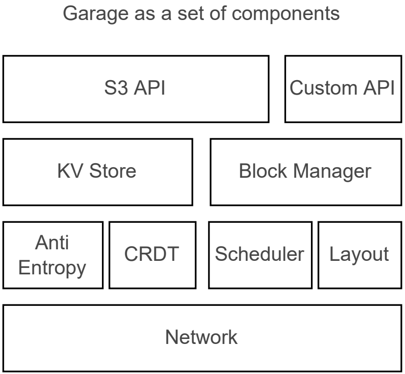
Storing and retrieving files



Garage's internal data structures



Garage's architecture

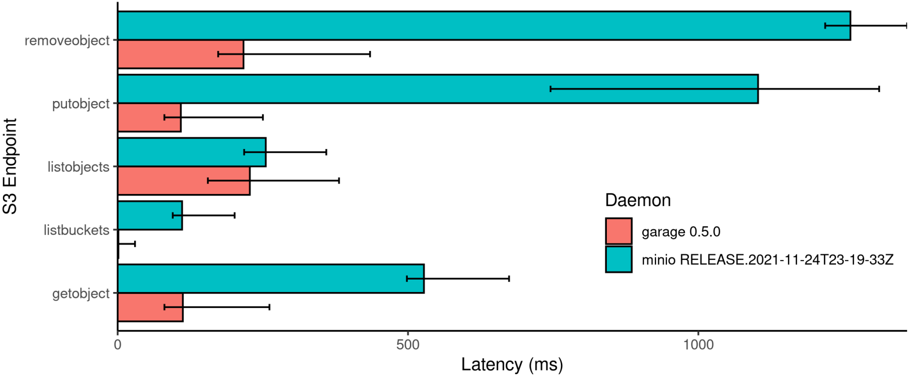


Garage is *coordination-free*:

- No Raft or Paxos
- Internal data types are CRDTs
- All nodes are equivalent (no master/leader/index node)

→ less sensitive to higher latencies between nodes

Impact on performances



S3 endpoint latency in a simulated geo-distributed cluster

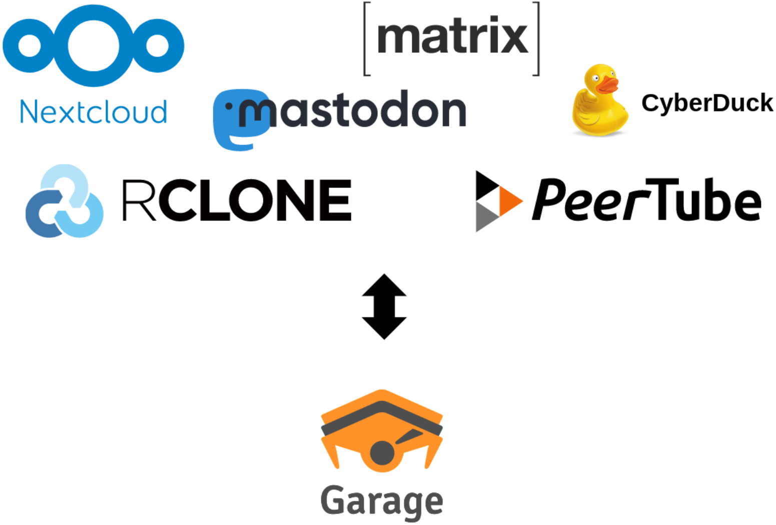
100 measurements, 6 nodes in 3 DC (2 nodes/DC), 100ms RTT + 20ms jitter between DC
no contention: latency is due to intra-cluster communications
colored bar = mean latency, error bar = min and max latency

Get the code to reproduce this graph at https://git.deuxfleurs.fr/quentin/benchmarks

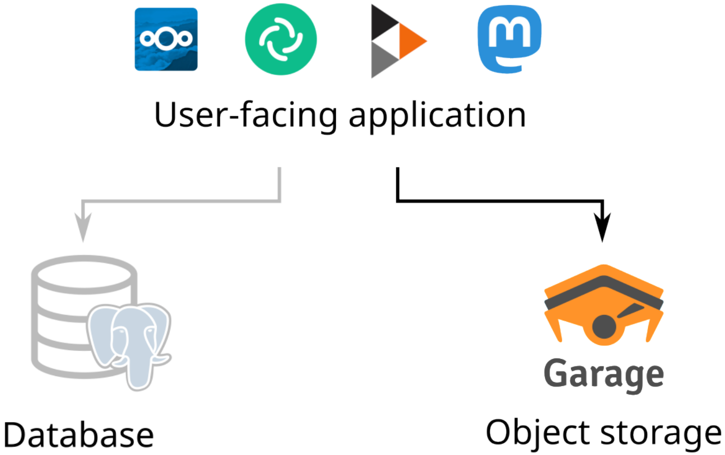
Consistency model

- Not ACID (not required by S3 spec) / not linearizable
- **Read-after-write consistency**
  (stronger than eventual consistency)

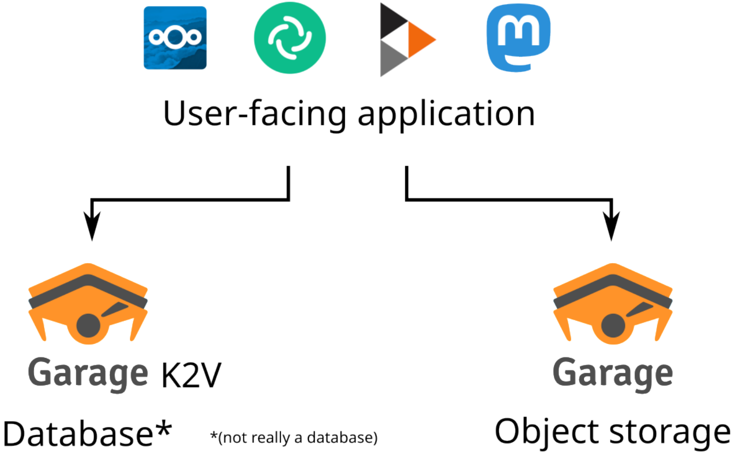
An ever-increasing compatibility list



Further plans for Garage



Further plans for Garage



User-facing application

Garage K2V

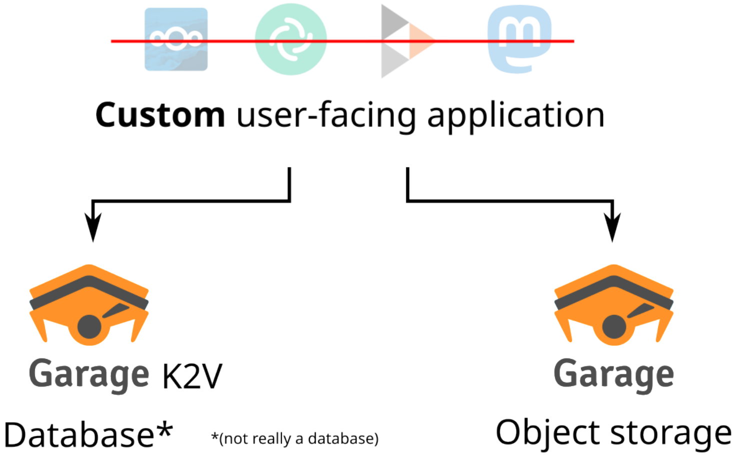
Database*

*(not really a database)

Garage

Object storage

Further plans for Garage



Custom user-facing application

Garage K2V

Database*

*(not really a database)

Garage

Object storage

K2V Design

- A new, custom, minimal API

K2V Design

- A new, custom, minimal API
- Exposes the partitoning mechanism of Garage
  K2V = partition key / sort key / value (like Dynamo)

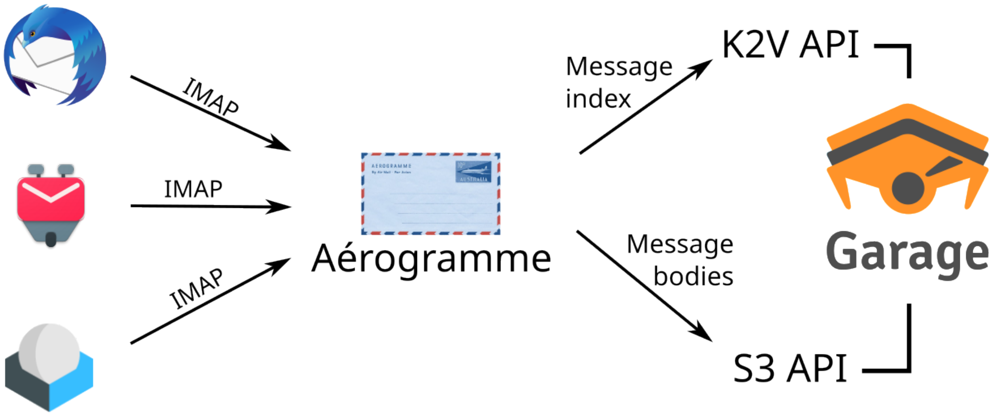
K2V Design

- A new, custom, minimal API
- Exposes the partitoning mechanism of Garage
  K2V = partition key / sort key / value (like Dynamo)
- Coordination-free, CRDT-friendly (inspired by Riak)

K2V Design

- A new, custom, minimal API
- Exposes the partitoning mechanism of Garage
  K2V = partition key / sort key / value (like Dynamo)
- Coordination-free, CRDT-friendly (inspired by Riak)
- Cryptography-friendly: values are binary blobs

Application: an e-mail storage server



Get Garage now!

`https://garagehq.deuxfleurs.fr/`

Matrix channel: `#garage:deuxfleurs.fr`