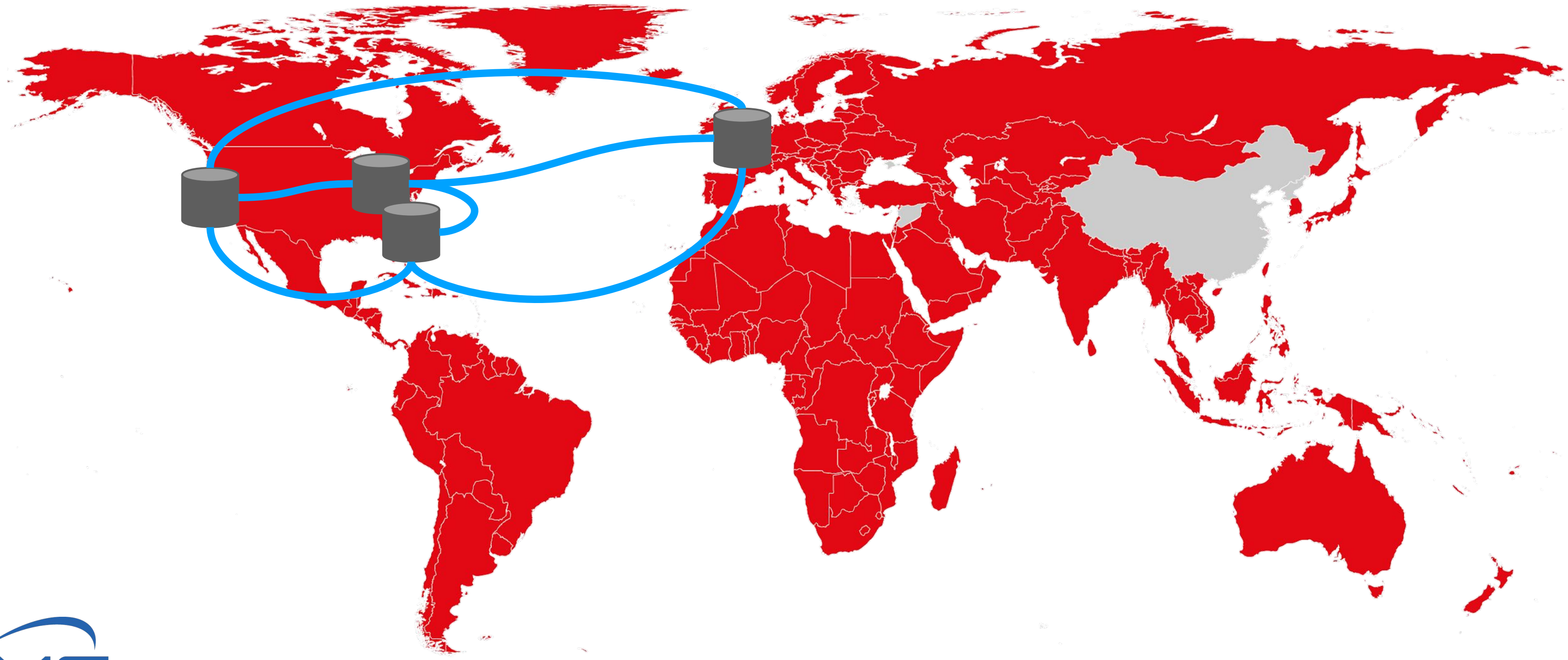
Decoupling Services from Storage Engines Through Data Abstractions at Netflix

Vidhya Arvind & Rajasekhar Ummadisetty
August 6th, 2024

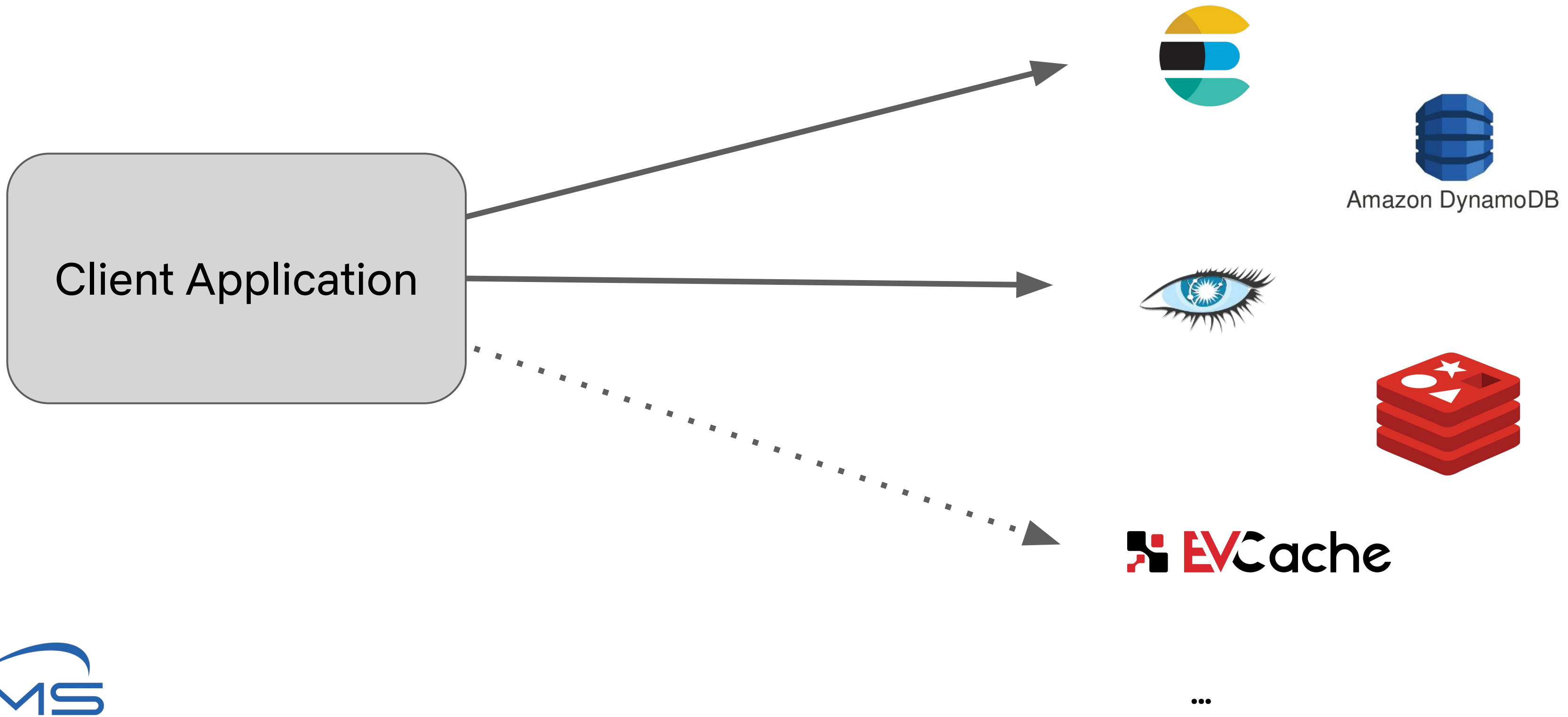
Vidhya Arvind
Staff Software Engineer
Data Platform @ Netflix

Rajasekhar Ummadisetty
Senior Software Engineer
Data Platform @ Netflix

Netflix Online Stateful Scale



Clients connecting to Storage engines directly



Challenges!

- ❑ Advanced knowledge to avoid antipatterns

Challenges!

- ❏ Advanced knowledge to avoid antipatterns
- ❏ Coordinated database migrations

Challenges!

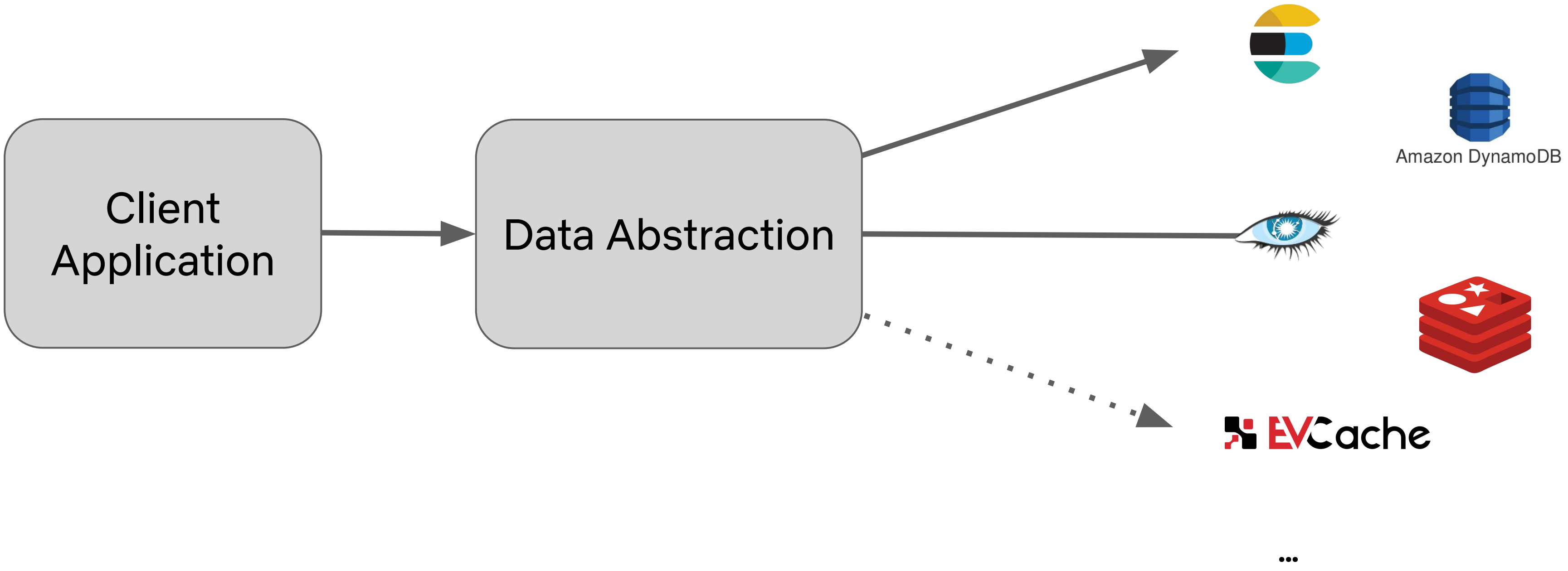
- ❑ Advanced knowledge to avoid antipatterns
- ❑ Coordinated database migrations
- ❑ Frequent Reimplementation of Patterns

Challenges!

- ❑ Advanced knowledge to avoid antipatterns
- ❑ Coordinated database migrations
- ❑ Frequent Reimplementation of Patterns
- ❑ Integrating with Internal core services

Solution

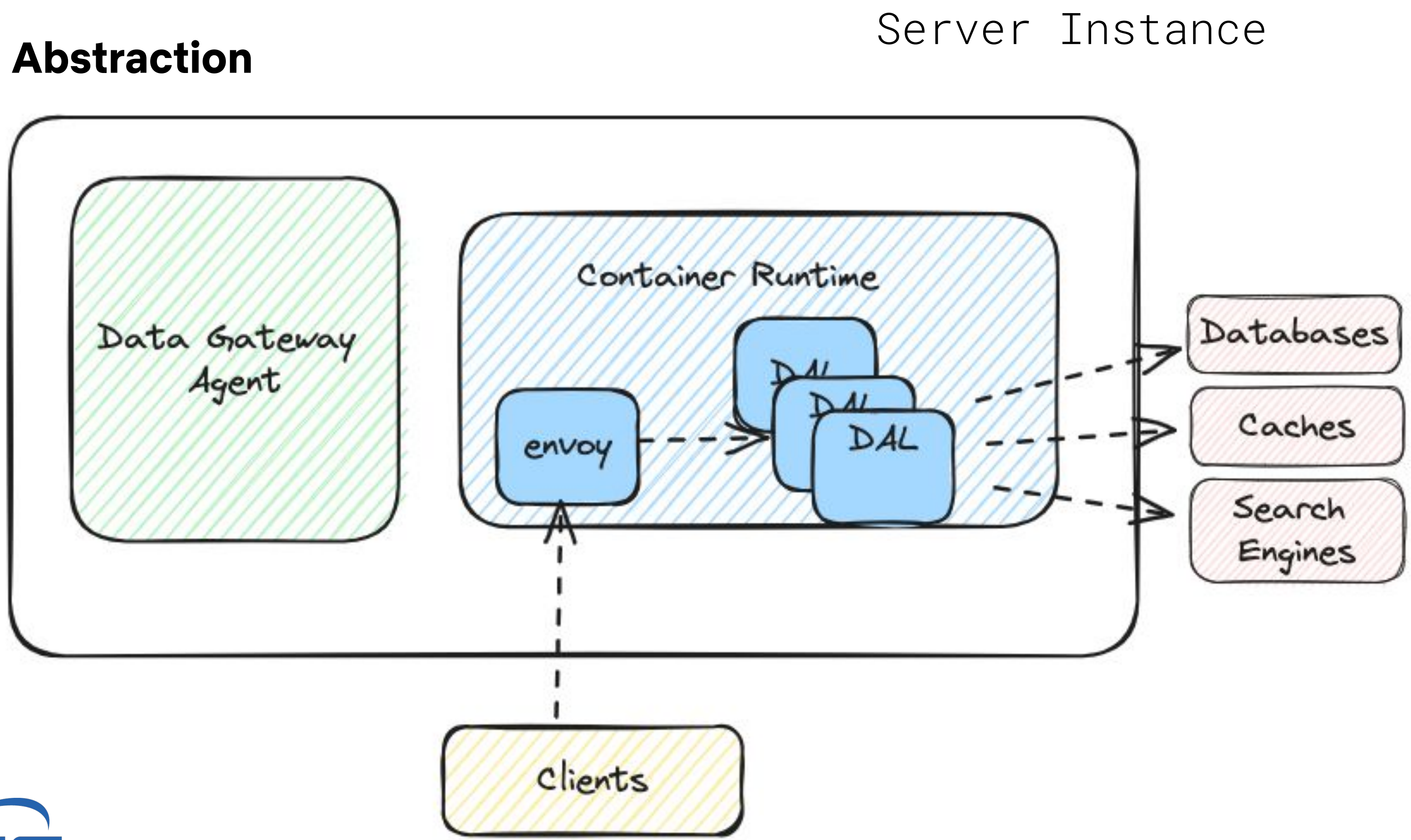
Data Abstraction Layers



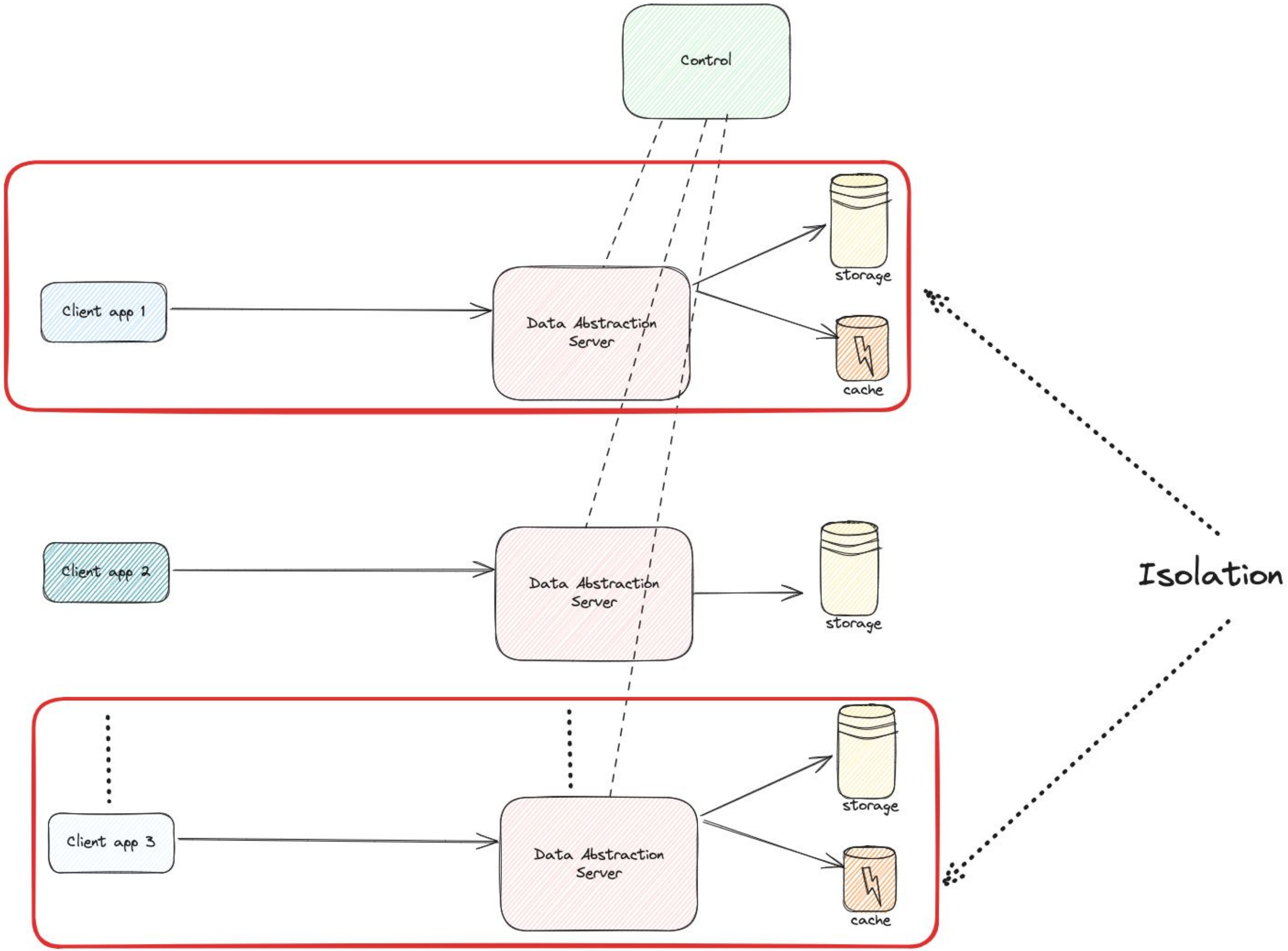
Data Abstraction Layers

- ❑ Simplifies ***Data Access***
- ❑ Enhances ***Security***
- ❑ Improves ***Reliability***
- ❑ Increases ***Scalability***
- ❑ ***Centralized*** Management
- ❑ Boosts ***Developer Productivity***

Data Abstraction

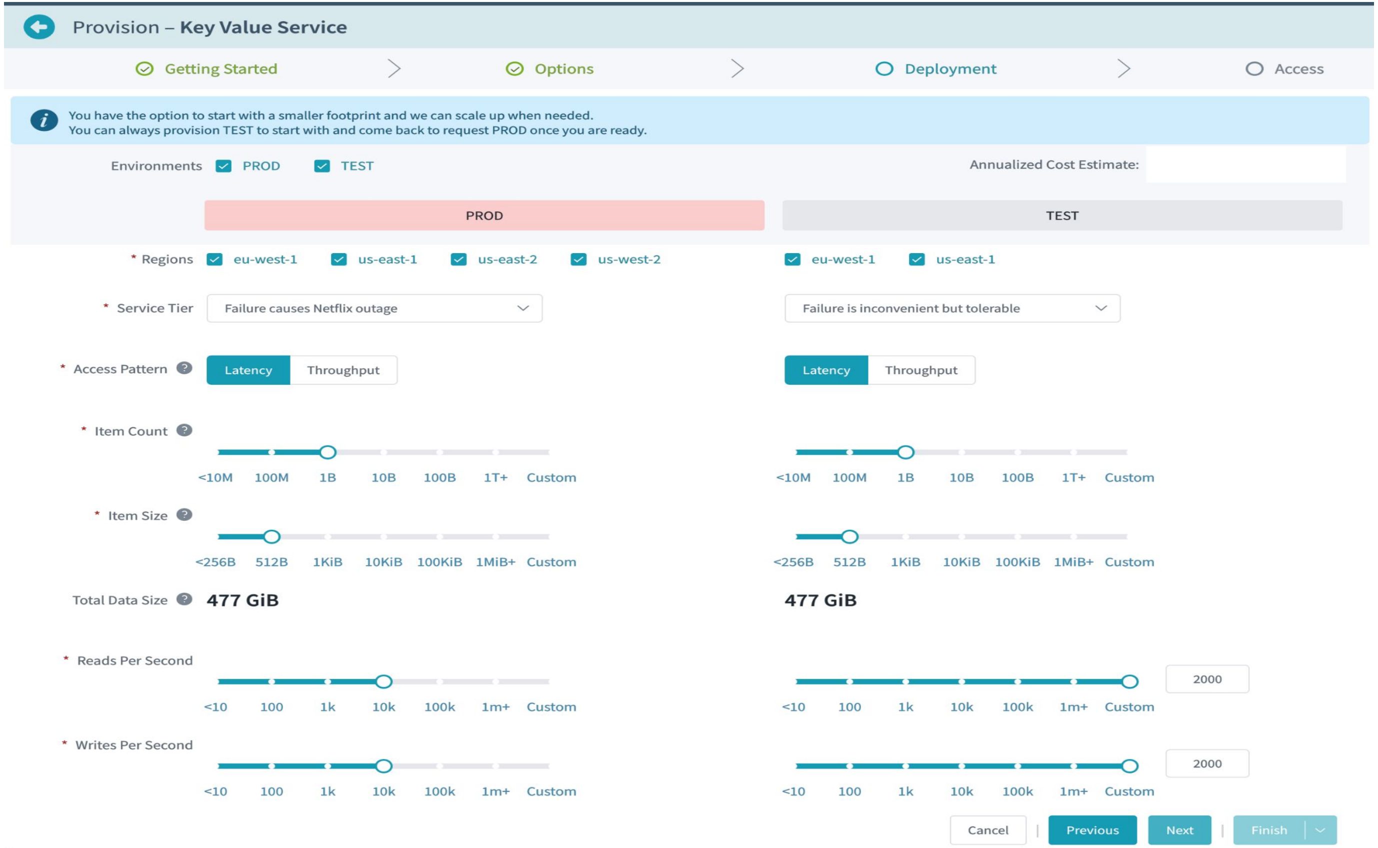


Sharding


Control
Client app 1
Data Abstraction Server
storage
cache
Client app 2
Data Abstraction Server
storage
Client app 3
Data Abstraction Server
storage
cache
Isolation

Self service

Self Service



Deployment Desire

```yaml
deploy_desires:
 # What are the access pattern and capacity
 capacity:
   model_name: org.netflix.key-value
   query_pattern:
     access_pattern: latency
     estimated_read_per_second:  {low: 2000, mid: 20000, high: 200000}
     estimated_write_per_second: {low: 2000, mid: 20000, high: 200000}
   data_shape:
     estimated_state_size_gib:   {low:   20, mid: 200,   high: 2000}
     reserved_instance_app_mem_gib: 20
 # How critical is this deployment to Netflix
 service_tier: 0
 # What version set of software should be deployed
 version_set:
     artifacts:
       dals/dgw-kv:  {kind: branch, value: main}
       # Runtime configuration is a container as well!
       configs/main: {kind: branch, sha: ${DGW_CONFIG_VERSION}}
```

Deployment Desire

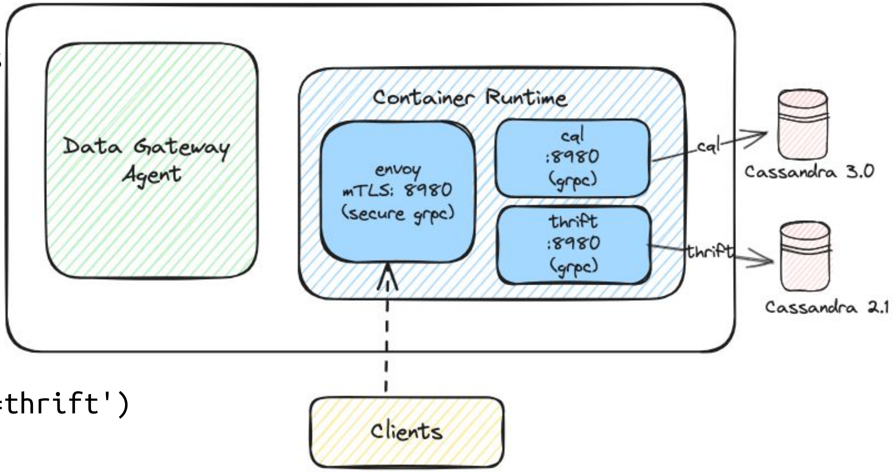
```yaml
# Where should we deploy to, including multiple clusters
locations:
  - account: prod
    regions: [us-east-2, us-east-1, eu-west-1, us-west-2]
  - account: prod
    regions: [us-east-1]
    stack: leader
# Who owns (is responsible for) this deployment
owners:
  - {type: google-group, value: our-cool-team@netflix.com}
  - {type: pager, value: our-cool-pagerduty-service}
# Who consumes (uses) this deployment, and what role?
consumers:
  - {type: account-app, value: prod-api, group: read-write}
  - {type: account-app, value: studio_prod-ui, group: read-only}
```

Deployment Configuration

Data Gateway Configuration

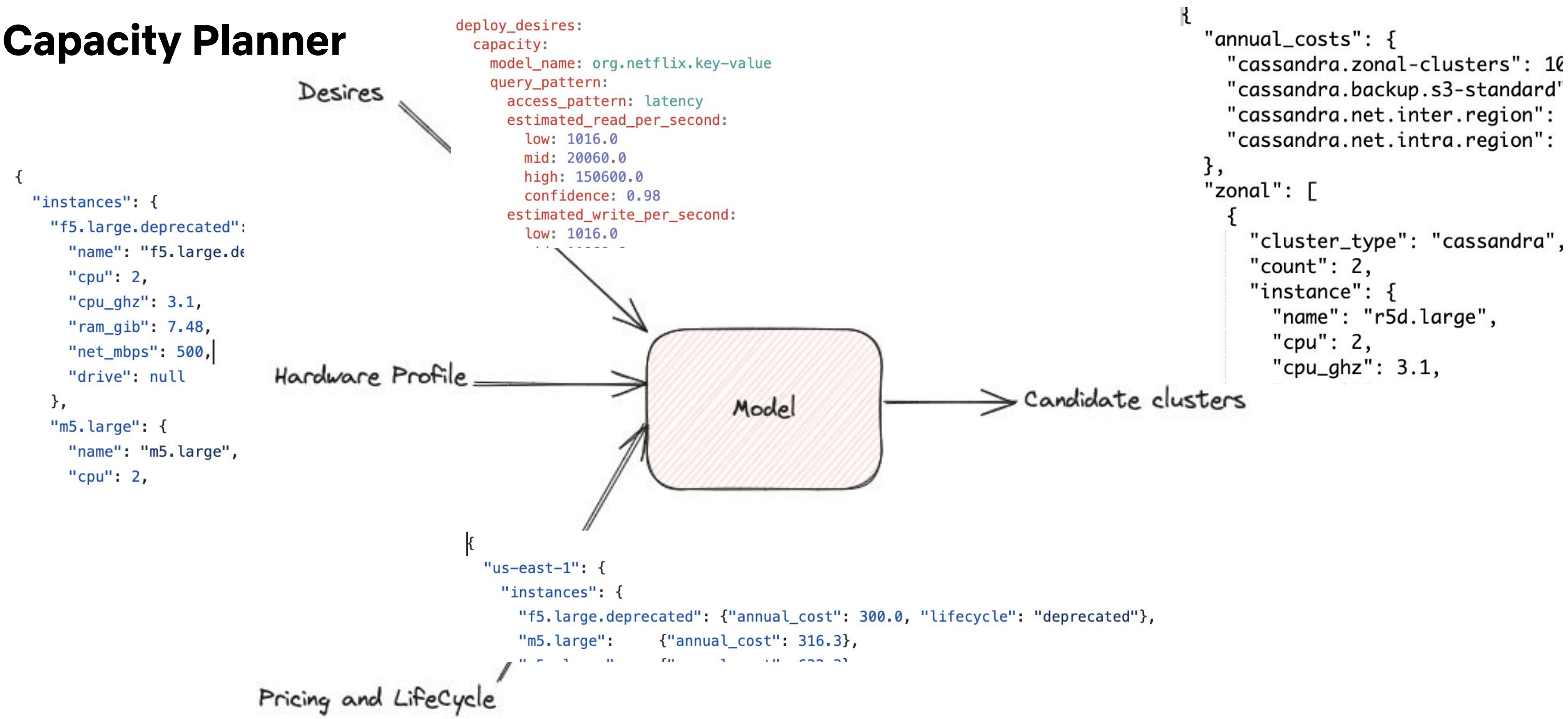
```
# Configure the proxy to accept protocols
proxy_config:
  public_listeners:
    secure_grpc: {mode: grpc, tls_creds: metatron,
                  authz: gandalf, path: 8980}
# Configure the DAL containers, implementing protocols
container_dals:
  cql:
    container_listeners: {secure_grpc: 8980}
    image: "dgw-kv"
  thrift:
    container_listeners: {secure_grpc: 8980}
    image: "dgw-kv"
    env:
      predicate.expression: scope.value.contains('dal=thrift')

# Configure advanced wiring of protocols
wiring:
  thrift: {mode: shadow, target: cql}
```



Capacity Planner

Capacity Planner



https://github.com/Netflix-Skunkworks/service-capacity-modeling

Namespace

Logical View of

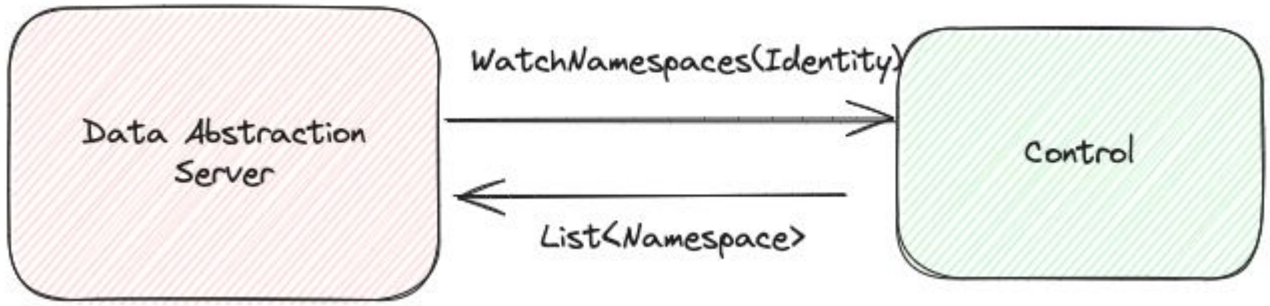
Physical Storages & **Configuration**

Unit of *data isolation* and *scaling*

Think "table", "database", "module", etc ...

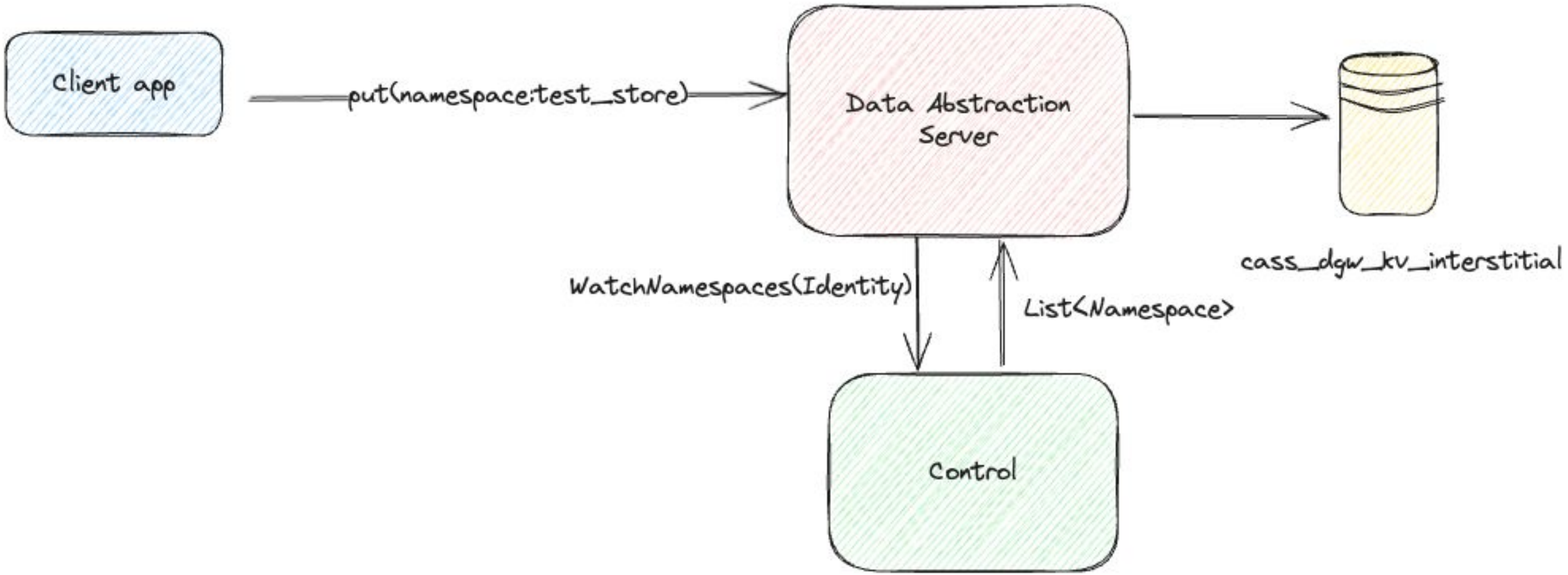
Namespace

Watch Namespace via Control Plane

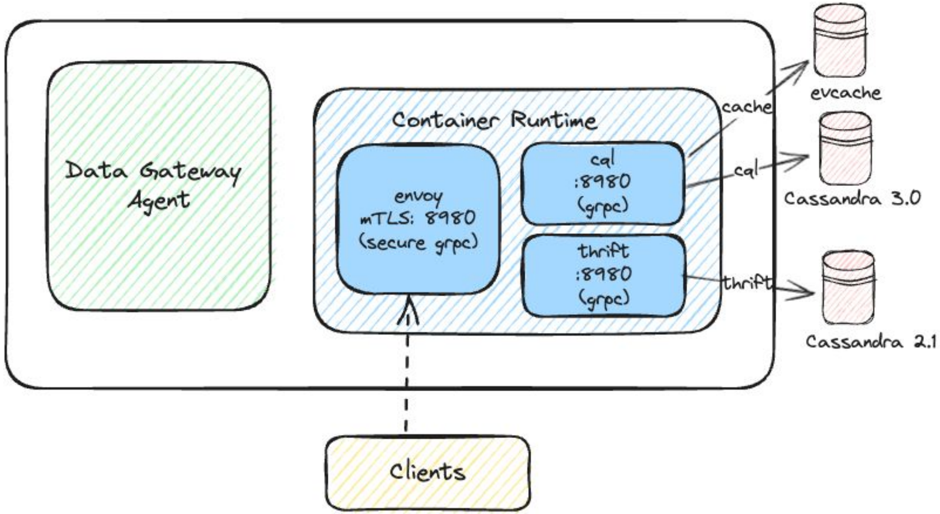


Namespace

Each Request contains Namespace



Namespace



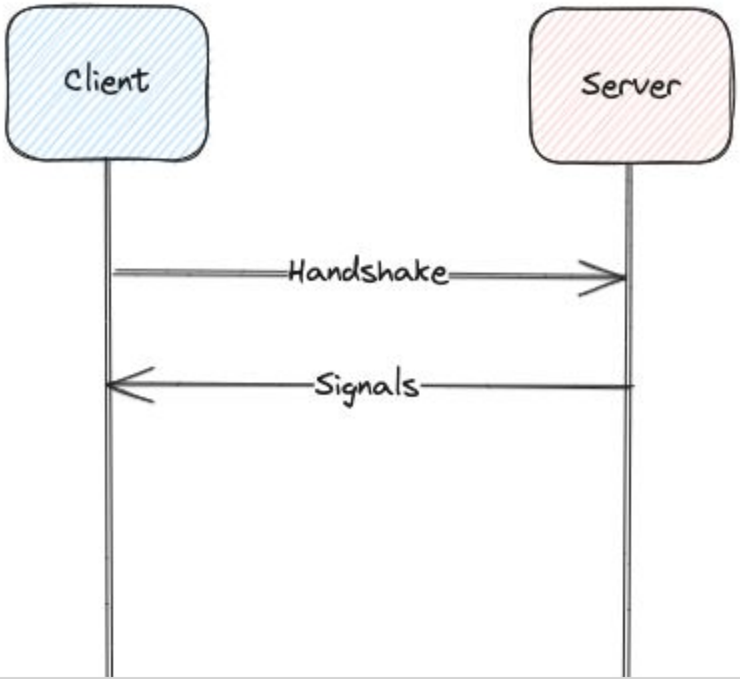

```
{
    "namespaceName":"ngevents",
    "persistenceConfigurations":{
       "persistenceConfiguration":[
          {
             "id":"PRIMARY_STORAGE",
             "version":1,
             "level":4,
             "scope":"dal=thrift",
             "physicalStorage":{...},
             "config":{...},
          {
             "id":"PRIMARY_STORAGE",
             "version":4,
             "level":4,
             "scope":"dal=cql",
             "physicalStorage":{...},
             "config":{...}
          },
          {
             "id":"CACHE",
             "version":1,
             "level":2,
             "scope":"dal=cql",
             "physicalStorage":{...},
             "config":{...}
          }
       ],
       "provenance":""
    },
    "capabilities":[...],
    "owners":[...],
    "lifecycleEvents":[...],
    "status":"ACTIVE",
    "createTs":"2023-07-07T20:39:06Z",
    "shardIdentity":[...]
}
```

Namespace

```
{
   "id":"PRIMARY_STORAGE",
   "version":4,
   "level":4,
   "scope":"dal=cql",
   "physicalStorage":{
      "type":"CASSANDRA",
      "cluster":"cass_dgw_kv_ngevents",
      "dataset":"ngevents",
      "table":"ngevents",
      "schemaId":"kv:cassandra",
      "regions":[
         "us-east-1"
      ]
   },
"config":{
      "chunked":{
         "chunk-after":128
      },
      "consistency_scope":"LOCAL",
      "consistency_target":"READ_YOUR_WRITES",
      "context":"Push notification events",
      "disable_adaptive_page_limit":false,
      "enable_slo_stop_predicate":true,
      "kv_scan_checkpointing_disabled":false,
      "slos":{
         "access":{
            "latency":{
               "max":"0.5s",
               "target":"0.03s"
            }
         },
         "mutate":{
            "latency":{
               "max":"0.5s",
               "target":"0.03s"
            }
         },
         "read":{
            "latency":{
               "max":"0.5s",
               "target":"0.03s"
            }
         }
      }
   }
}
```

Signals

Signals to client



- Signals are a medium for exchanging capabilities and configurations between the client and server
- Facilitate dynamic configuration

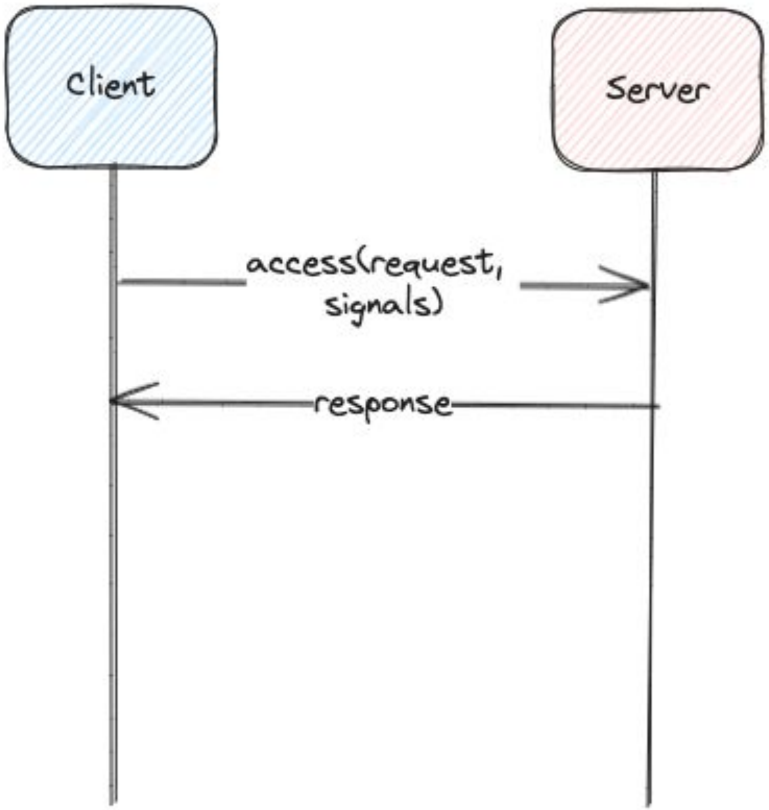
$
grpc -a dgwkv.napa -e prod -r us-east-1 **com.netflix.dgw.kv.v2.KeyValueServiceV2/Handshake** | jq .signals

Signal

Signals to client

```
{
    "ngevents":{
        "payload":{
            "chunked":{...},
            "client-cache-control":{...},
            "client-route-to-dal":"cql",
            "system_utilization":{...}
        },
        "compression":[...],
        "slo_by_endpoint":{...}
    }
}
```

Signals to server



Client capabilities

Signals to server


```
{
   "signals":{
      "accept-encoding":{
         "payload":{
            "chunked":true
         },
         "compression":[
            {
               "algorithm":"LZ4"
            },
            {
               "algorithm":"CUSTOM"
            }
         ]
      }
   }
}
```

Reliable Abstractions

Idempotency

- APIs are designed to be idempotent, ensuring safety during retries.
- Clients provide an idempotency token to achieve idempotency.
- Ensures operations can be retried without unintended side effects.

```
def put_with_retry(data):
 Idempotency_token =
        get_idempotency_token()
 result = put(idempotency_token, data)
 // safely retry
 if result.status != SUCCESS:
   result = put(idempotency_token, data)
```

```
message IdempotencyToken {
  google.protobuf.Timestamp
          generation_time = 1;

  string token = 2;
}
```

Idempotency

- Client-Generated Tokens:
  - Guaranteed to monotonically increase within a single client
  - Suitable for most operations
- Server-Generated Tokens:
  - Guaranteed to monotonically increase within a given region
  - Generated on the server
  - Client requests the server for a token before performing the operation
  - Suitable for performing isolated operations

Chunking

Small Payloads:

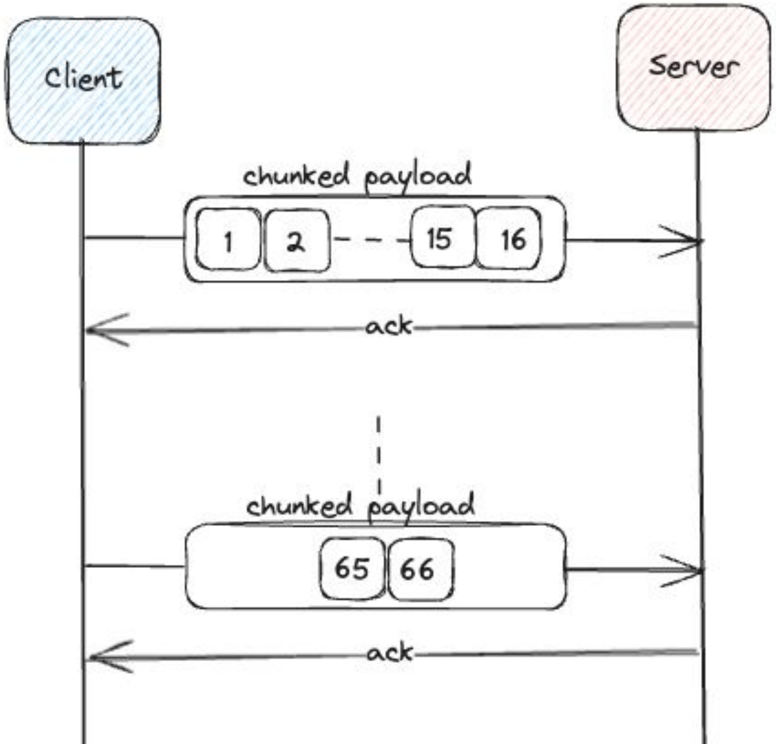
- ❑ Clients can send small payloads directly in a single request.
- ❑ Simple and efficient for small data sizes.

Large Payloads:

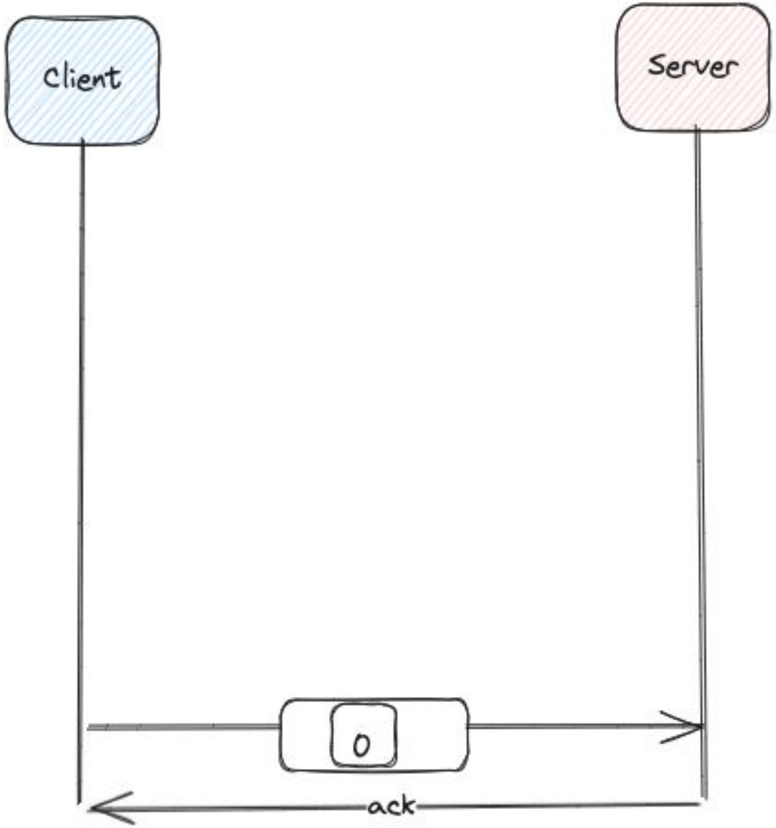
- ❑ For large payloads, clients can break them into smaller chunks.
- ❑ Helps avoid resending large payloads over the network in case of request failures.

Chunking

Chunked Payload Transmission

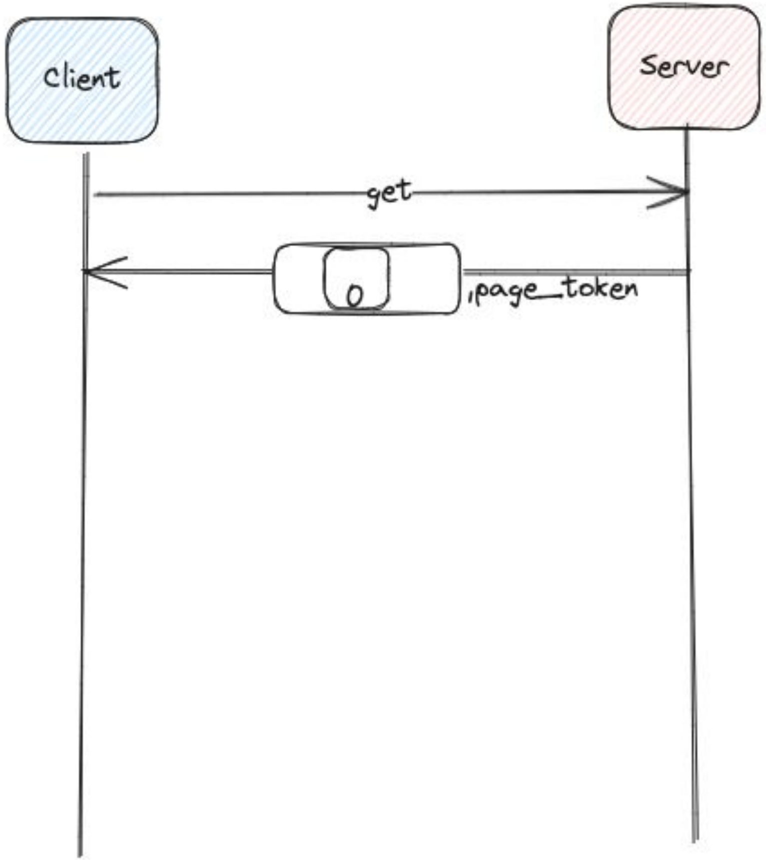



```
chunk-after-bytes = 1 Mib
chunk-size-bytes  = 64 Kib
```

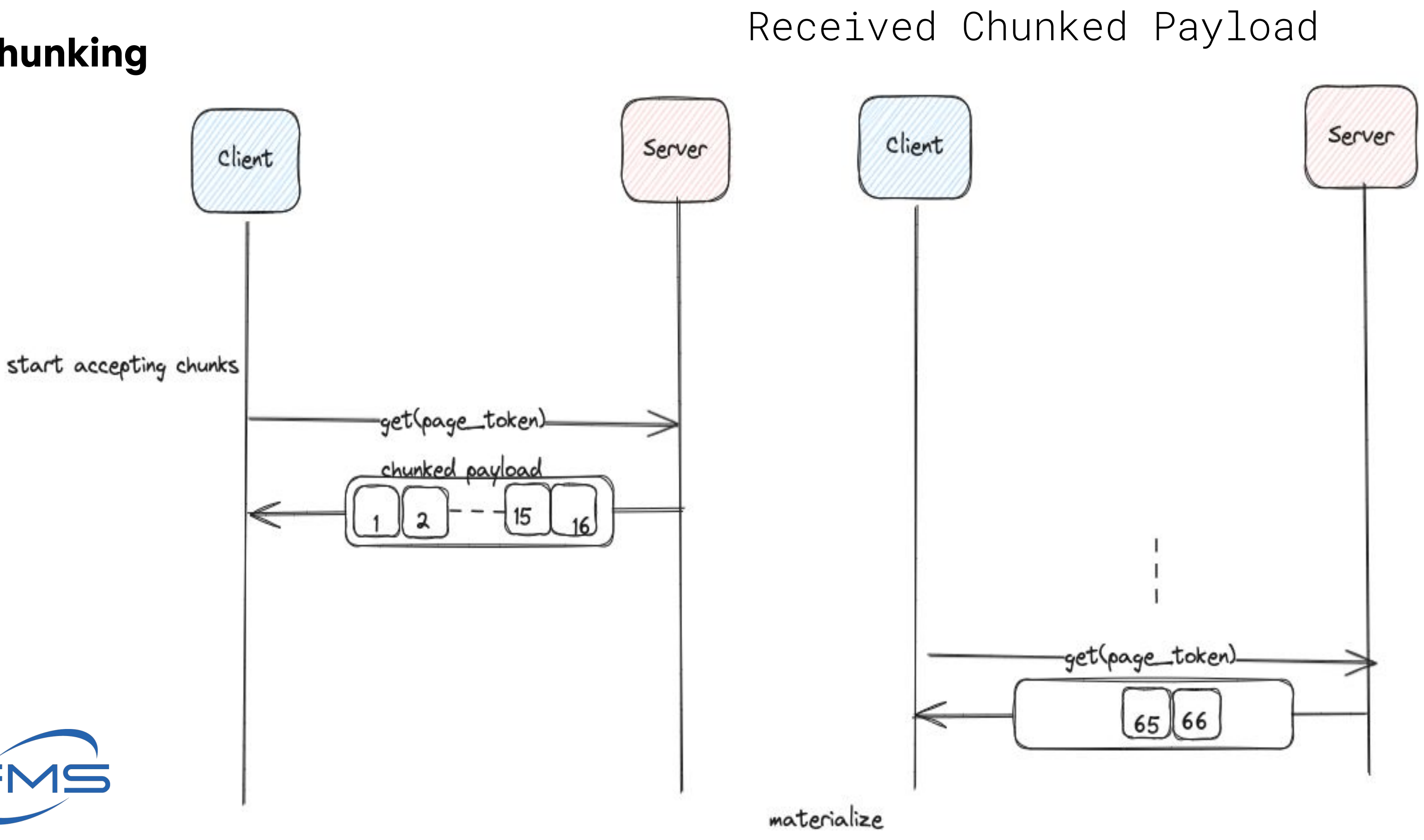


Chunking

Received Chunked Payload

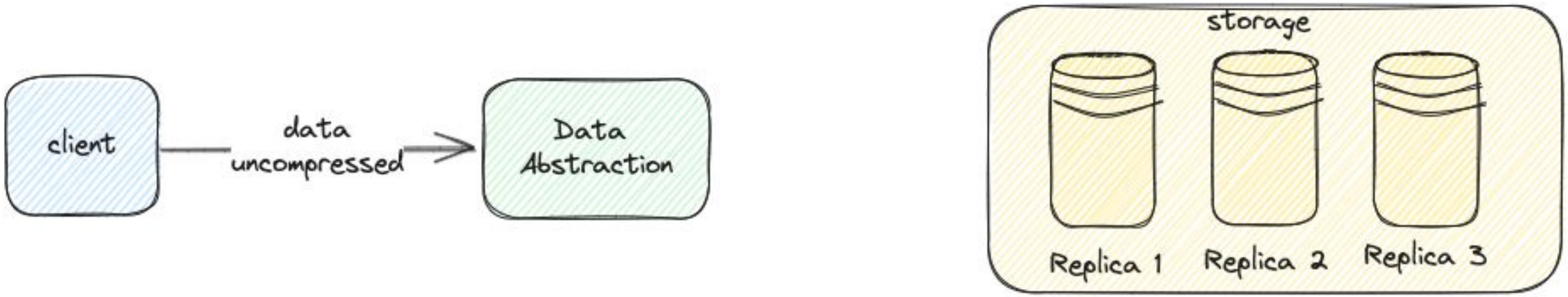


Chunking

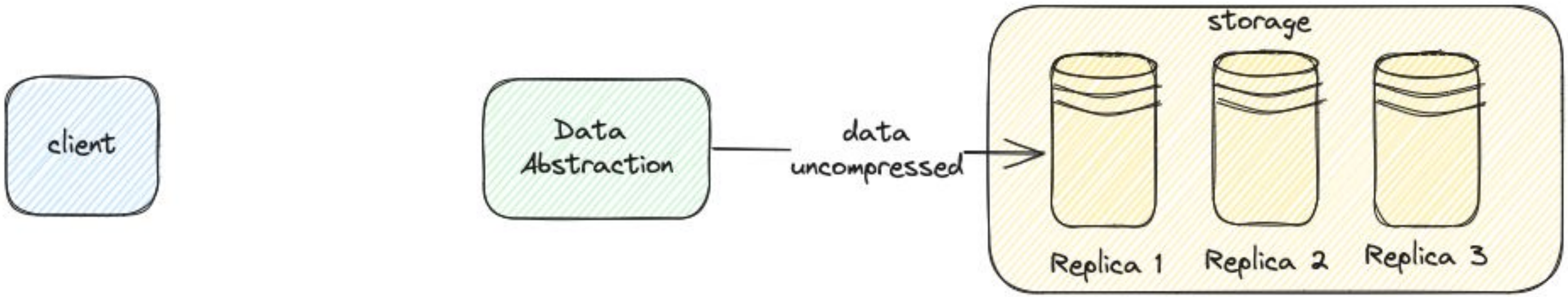


Received Chunked Payload
Client
Server
start accepting chunks
get(page_token)
chunked payload
1
2
15
16
Client
Server
get(page_token)
65
66
materialize

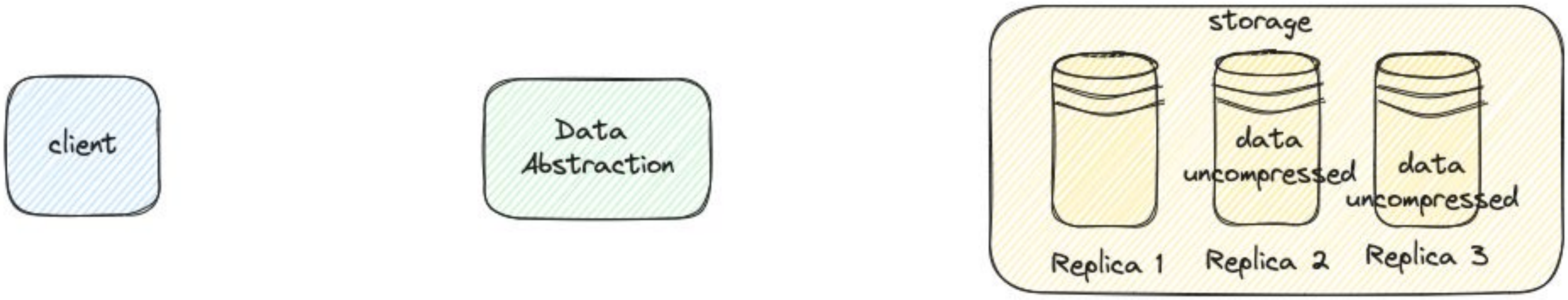
Compression



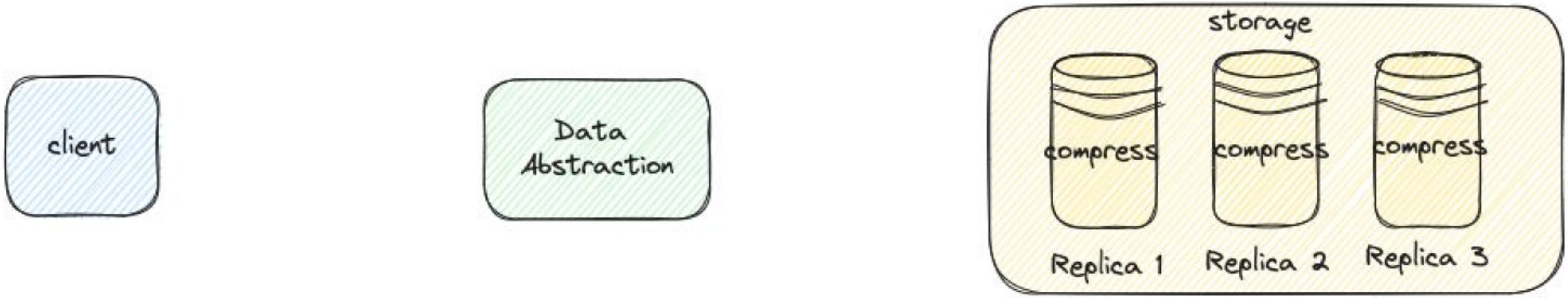
Compression



Compression

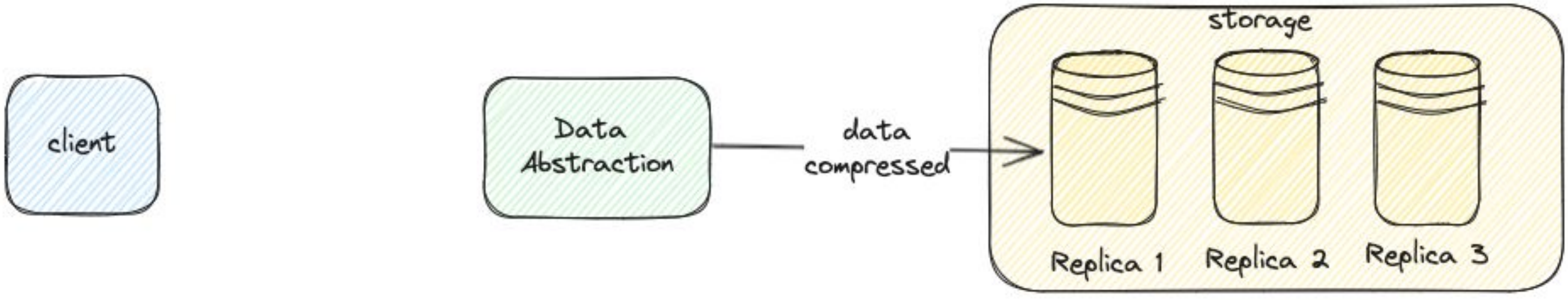


Compression



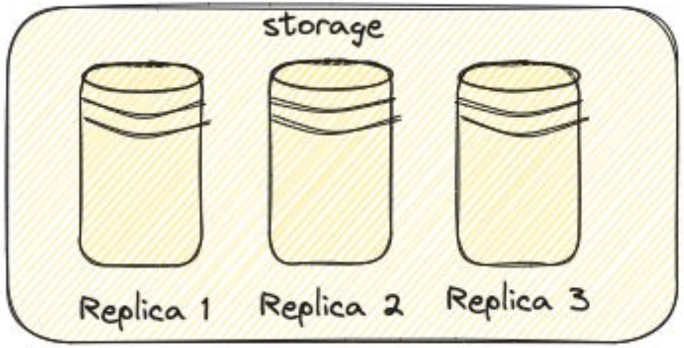
Compression

Same when decompressing the data. But instead if you compress in the abstraction layer



Compression

Or in the client



Storage still compresses, but saves:

- Commit Log
- Allocations
- Disk IO
- Network IO
- Overall ratio

Pagination

Why Paginate?

Accumulate pages of fixed size work (MiB), clients must ask for more

Storage engines almost always paginate by row count

Robust APIs paginate by *size not count*. Translation required.

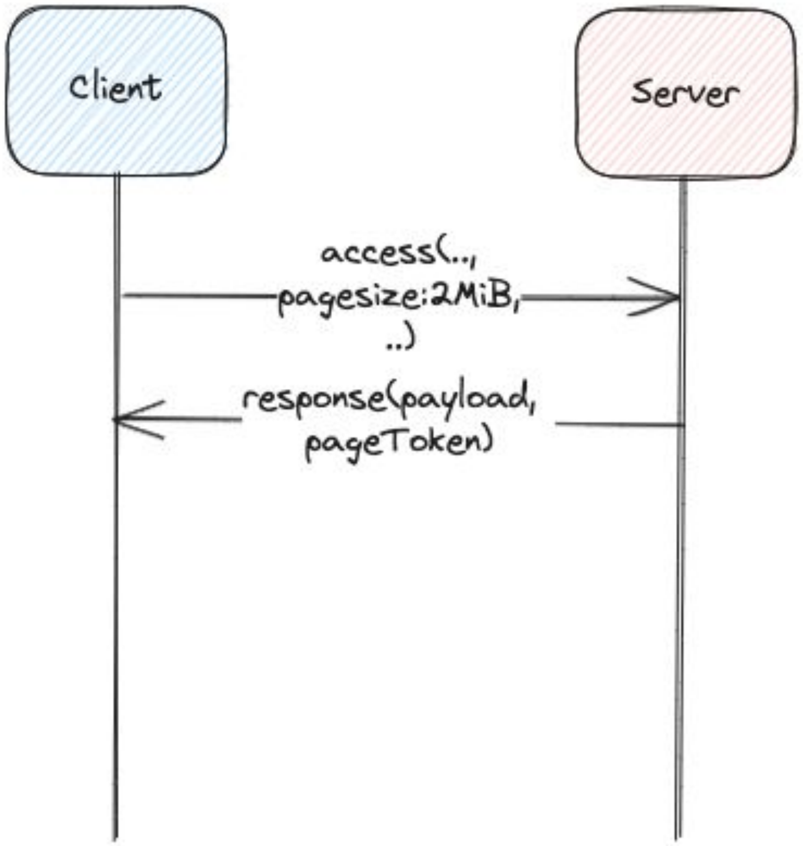
Pagination

Responses are paginated

Clients can specify the page size in bytes

Server sends back a response with:

- Payload size <= page size bytes
- Page token if more data is present



Pagination

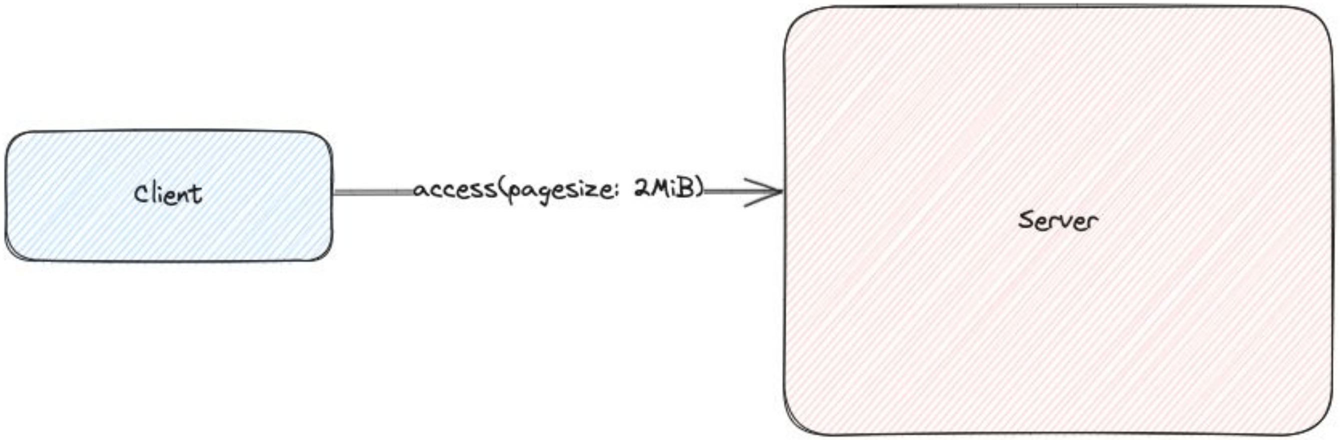
What should be the value for count, the server uses to retrieve data from db?

Large count value, will result in wasted resources if only part of the data is used to fill a page

Small count will result in read amplification and additional network round trips to the db

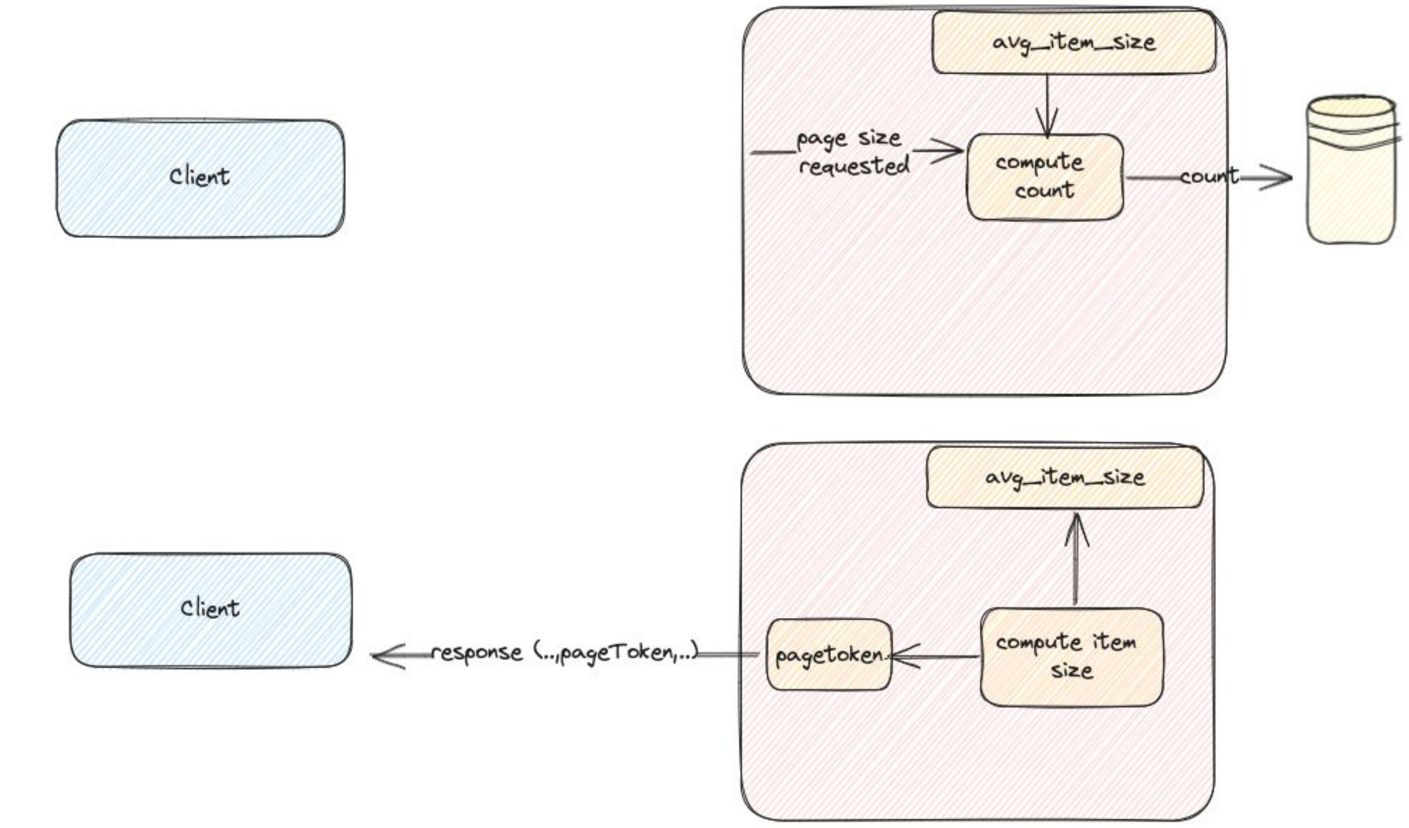
Pagination

Count is dynamically adjusted to an optimum value



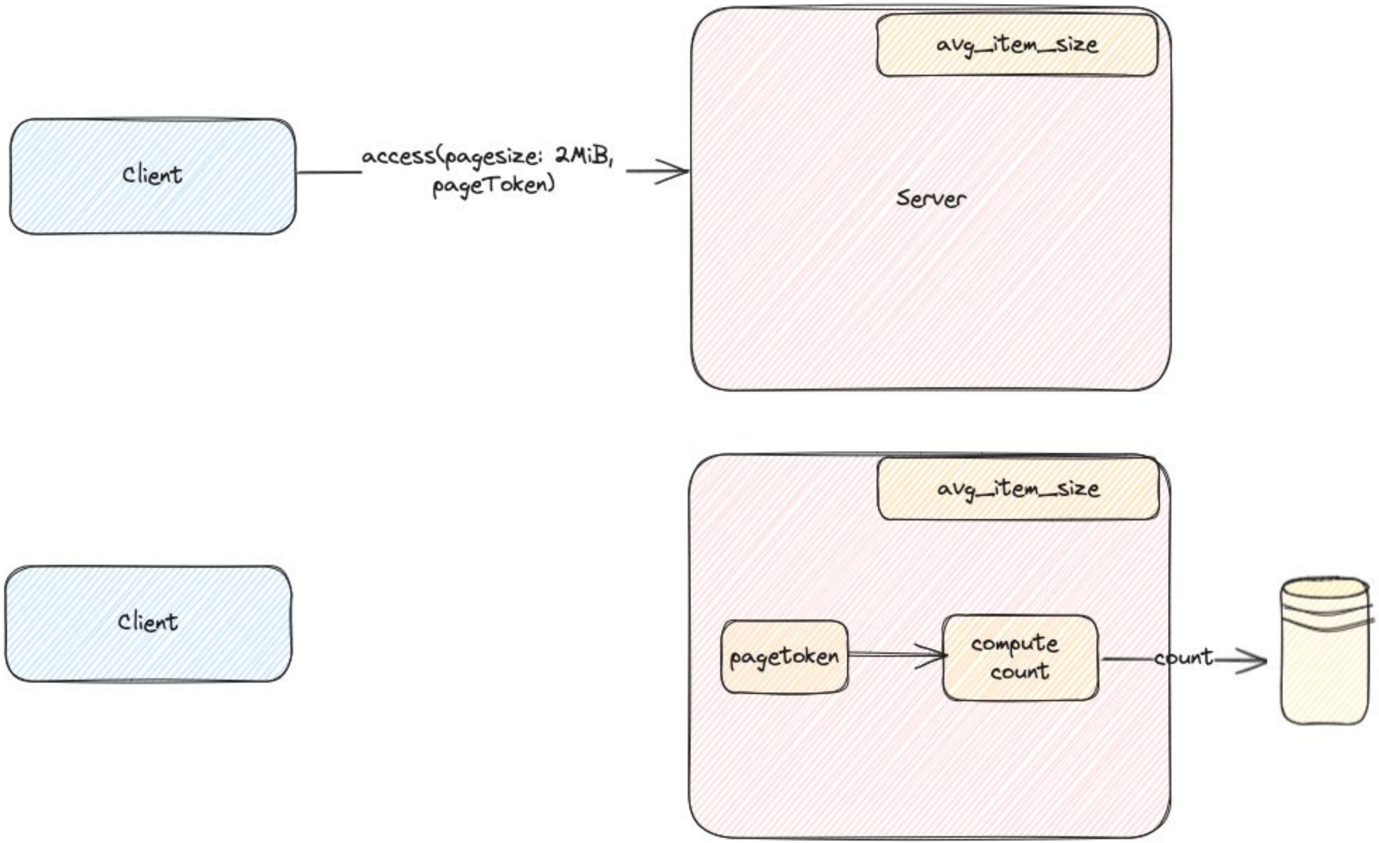
Pagination

Adaptive Pagination



Pagination

Adaptive Pagination



Adaptive Pagination

- ❑ In Addition to dynamic count we also have implemented SLO based pagination.
- ❑ If server is taking time to fill up a page and can potential violate SLO, server will stop and return early with pagination token.
- ❑ Ensures that requests are processed within the agreed upon SLO but the results may contain less than the page size

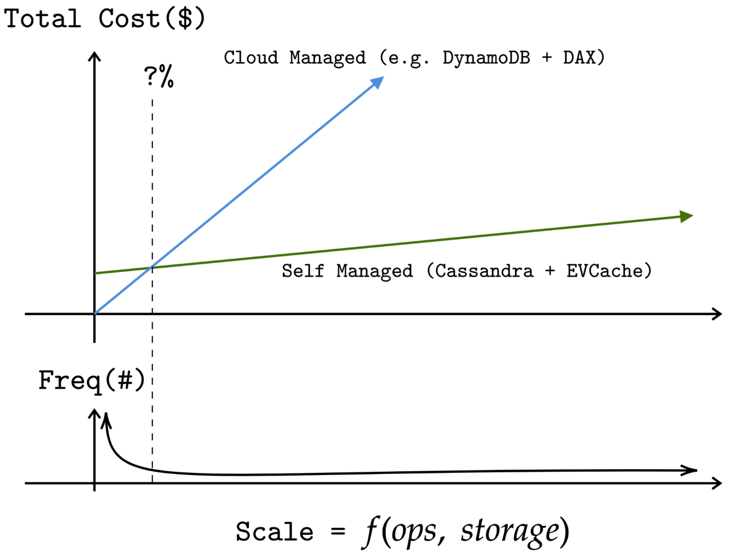
Migration

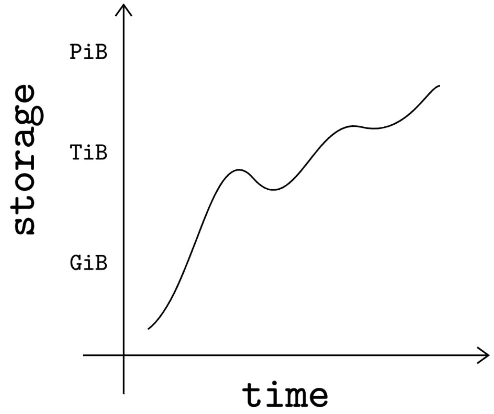
Problem

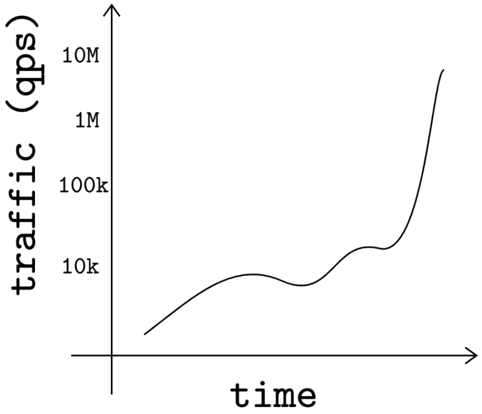
Access patterns change over time

Deprecate a db in favor of a different db

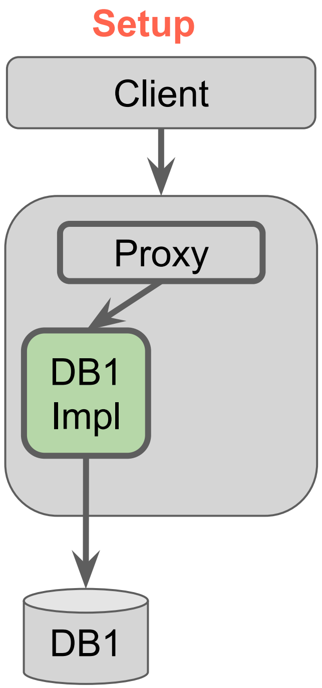
Backward incompatible DB upgrades



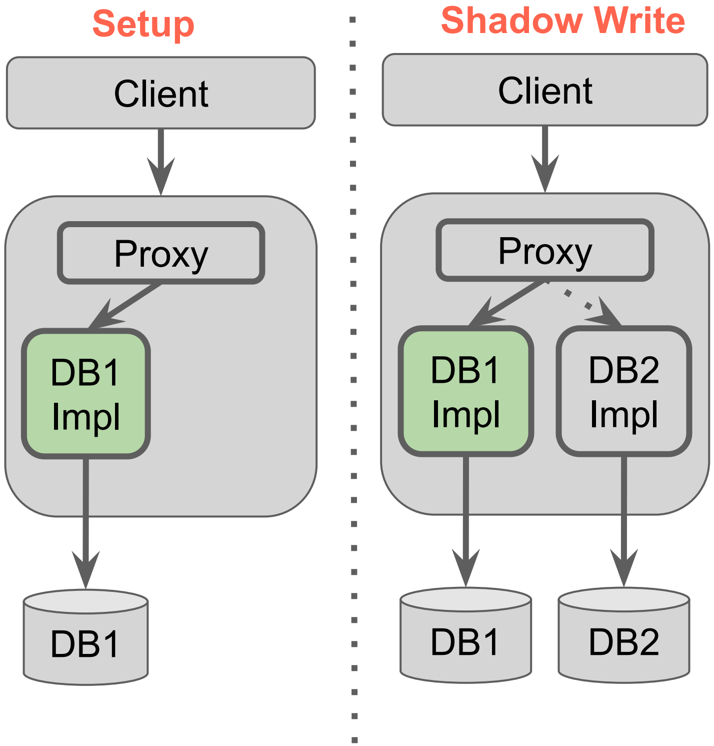




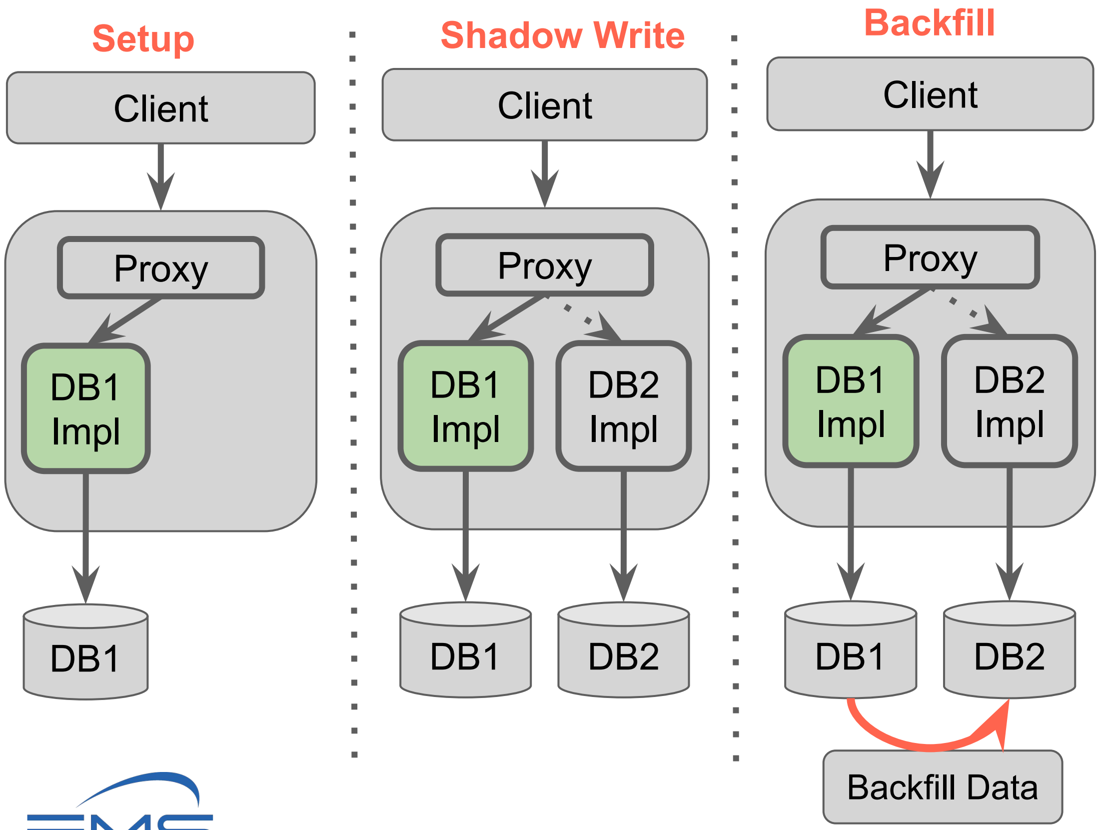
Migrations



Migrations

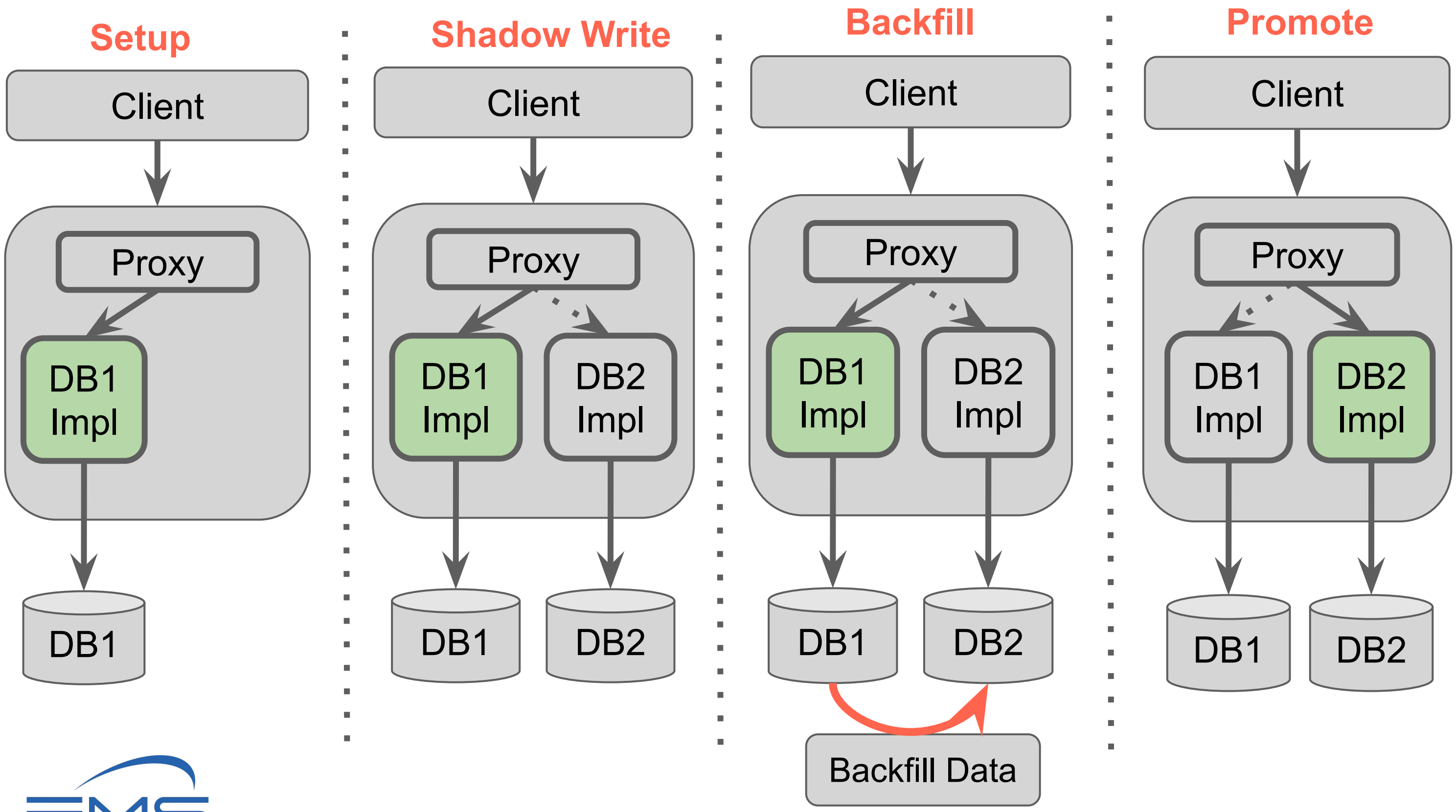

Setup
Client
Proxy
DB1
Impl
DB1
Shadow Write
Client
Proxy
DB1
Impl
DB2
Impl
DB1
DB2

Migrations

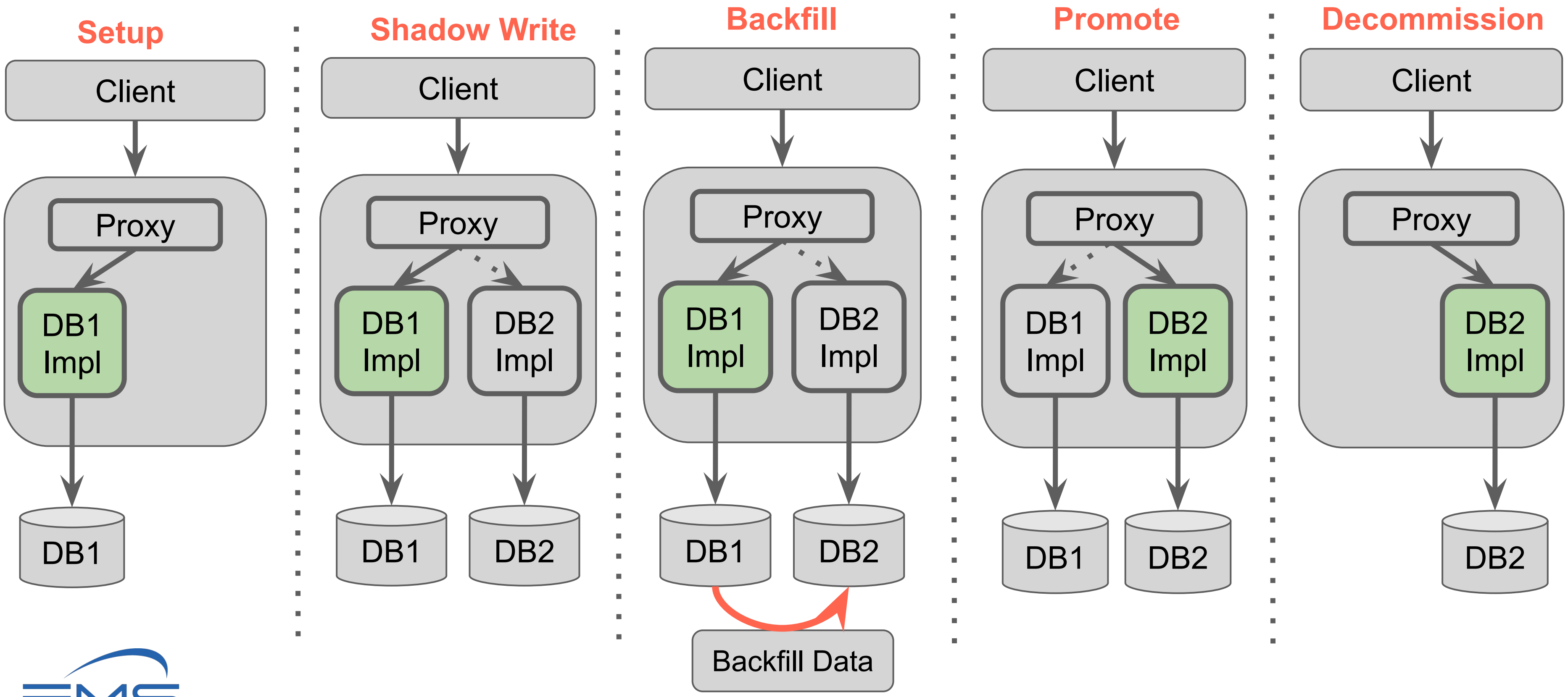


Setup
Client
Proxy
DB1
Impl
DB1
Shadow Write
Client
Proxy
DB1
Impl
DB2
Impl
DB1
DB2
Backfill
Client
Proxy
DB1
Impl
DB2
Impl
DB1
DB2
Backfill Data

Migrations


Setup
Client
Proxy
DB1 Impl
DB1
Shadow Write
Client
Proxy
DB1 Impl
DB2 Impl
DB1
DB2
Backfill
Client
Proxy
DB1 Impl
DB2 Impl
DB1
DB2
Backfill Data
Promote
Client
Proxy
DB1 Impl
DB2 Impl
DB1
DB2

Migrations



Setup
Client
Proxy
DB1 Impl
DB1
Shadow Write
Client
Proxy
DB1 Impl
DB2 Impl
DB1
DB2
Backfill
Client
Proxy
DB1 Impl
DB2 Impl
DB1
DB2
Backfill Data
Promote
Client
Proxy
DB1 Impl
DB2 Impl
DB1
DB2
Decommission
Client
Proxy
DB2 Impl
DB2

Abstractions

- Key Value
- Time Series
- Control
- Counter
- Identifier
- WAL
- Tree
- Graph

Key Value Abstraction

The KeyValue Data Abstraction offers a robust "HashMap as a service"

Key Value Abstraction

The KeyValue Data Abstraction offers a robust "HashMap as a service"

Namespace (table) can contain up to hundreds of billions of Records

Key Value Abstraction

The KeyValue Data Abstraction offers a robust "HashMap as a service"

Namespace (table) can contain up to hundreds of billions of Records

Each Record contains unique Items of key-value pairs.

Key Value Abstraction

The KeyValue Data Abstraction offers a robust "HashMap as a service"

Namespace (table) can contain up to hundreds of billions of Records

Each Record contains unique Items of key-value pairs.

Within a Record, the items are sorted either ascending (default) or descending (optional)

Key Value Abstraction

HashMap<String, SortedMap<ByteString, ByteString>>

Item

```
message Item {
    string id = 1;

    bytes key = 2;

    bytes value = 3;

    Metadata metadata = 4;
}
```

APIs - PutItems

```
// Write one or more items into a Record.
rpc PutItems (PutItemsRequest) returns (PutItemsResponse)
```

```
message PutItemsRequest {
  IdempotencyToken
          idempotency_token = 1;
  string namespace = 2;

  string id = 3;

  repeated Item items = 4;
}
```

```
message PutItemsResponse {

  Trilean durable = 1;

  Trilean visible = 2;

  map<string, Signal> signals =
                              3;
}
```

APIs - GetItems

```
// Read all keys, certain keys, or ranges of keys from a Record.
rpc GetItems (GetItemsRequest) returns (GetItemsResponse)
```

```
message GetItemsRequest {
    string namespace = 1;

    string id = 2;

    Predicate predicate = 3;

    Selection selection = 4;

    map<string, Signal> signals = 5;
}
```

```
message GetItemsResponse {

 repeated Item items = 1;

 string next_page_token = 2;
}
```

APIs - ScanItems

```
message ScanItemsRequest {
   string client_id = 1;
   string unique_scan_id = 2;
   string namespace = 3;
   Predicate predicate = 4;
   ScanPredicate scan_predicate = 5;
   Selection selection = 6;
   Duration target_scan_duration = 7;
   int32 scan_concurrency = 8;
   map<string, Signal> signals = 9;
}
```

APIs - ScanItems

```
// Retrieve all items across all Maps stored in this Namespace.
rpc ScanItems(ScanItemsRequest) returns(ScanItemsResponse) {}
```

```
message ScanItemsResponse {

 repeated ScanResult
        results = 1;

 repeated string
        next_page_tokens = 2;

}
```

```
message ScanResult {

 string id = 1;

 repeated Item items = 2;

 int32 scanPercentComplete =
                             3;
}
```

Future Work

- ❑ Summarization
- ❑ Secondary Indexes
- ❑ Lifecycle Management
- ❑ Resource Limiters
- ❑ Back Pressure handling
- ❑ Nearline Caching

Thank You.

Vidhya Arvind
varvind@netflix.com

Rajasekhar Ummadisetty
rummadisetty@netflix.com