Deploying MongoDB Sharded Clusters with OpenStack, Terraform, Kubernetes and Helm

Nico Döbele, Robin Fink, Faraan Choudhry and Jianbang Zhuang
Cloud & Big Data Technologies

Agenda

Introduction

Introduction

Goal: Production-ready, distributed MongoDB sharded cluster

Focus: Automated deployment, data distribution, high availability

Tools: MongoDB, Kubernetes, Helm, Terraform, OpenStack

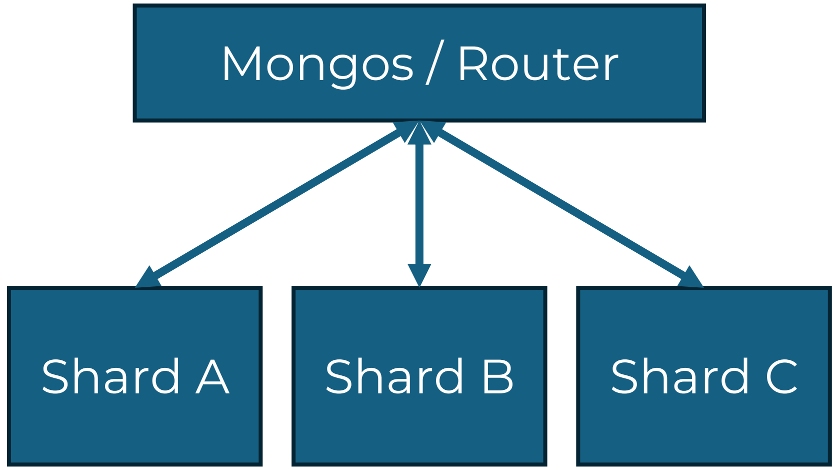
No pre-made Helm Chart: We want to create our own configuration

Example-App: „Twutter“ as a realistic test environment

Motivation

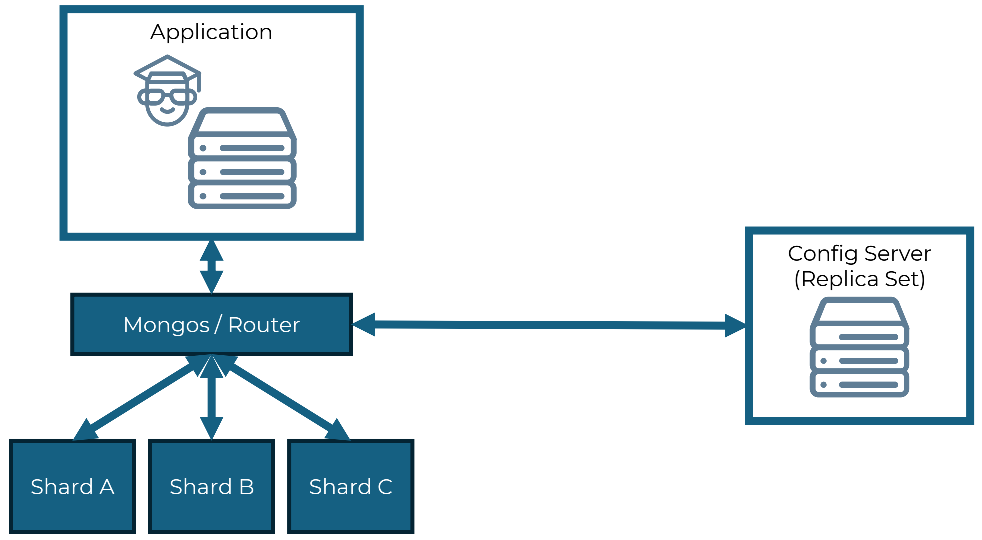
Motivation - Why a distributed database?

- **Data too large:** No longer fits on a single machine
- **Slow response times:** The database becomes a bottleneck
- **Scalability:** Scale horizontally instead of just upgrading hardware
- **Learning goal:** Understand sharding and failover in a cloud environment
- **Cloud deployment:** Use Terraform and Kubernetes to distribute data on CloudStack
- **Leverage CloudStack features:** Apply real infrastructure capabilities
- **Learn data distribution:** Understand how data is actually spread across shards
- **Kubernetes experience:** Explore and experiment as much as possible



Motivation - Cluster Architecture

- Distributed architecture with config servers, shards, and mongos
- Mongos router as the entry point for applications
- Data distributed across multiple shards (sharding)
- Each shard as a ReplicaSet (high availability)
- Managed and orchestrated via Kubernetes & Helm



Motivation - Why Own Helm-Chart?

Standard-Charts: Often to unflexible

Our Helm-Chart:

- Resources and replicas configurable per function
- Volumes defined per function
- Automatic sharding and replication
- Authentication via secrets
- Safe upgrades and idempotency
- Learning everything from scratch

MongoDB

MongoDB

NoSQL

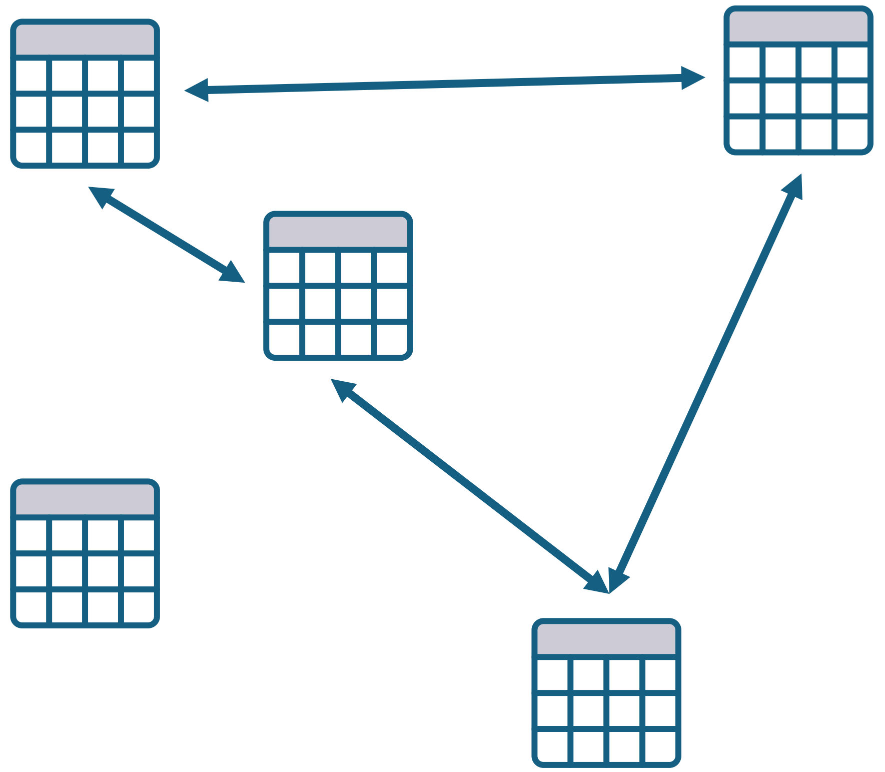
MongoDB

Not Only SQL

MongoDB

Nicht relationale Datenbank

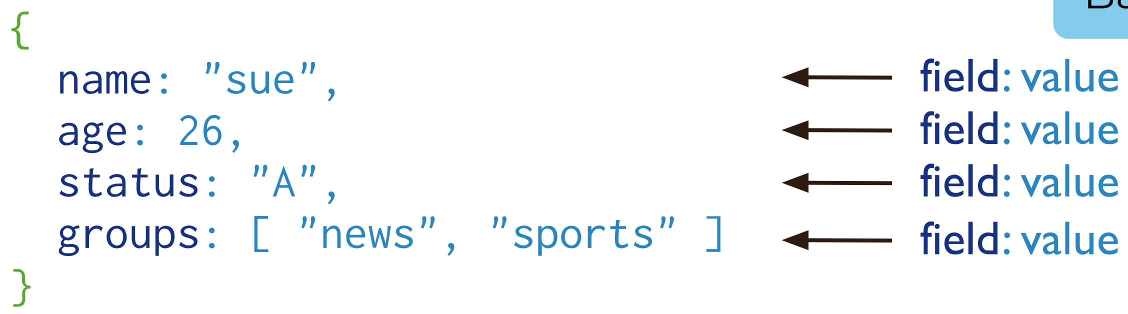
Relational Database



MongoDB – Document

A record in MongoDB is a document, which is a data structure composed of field and value pairs.

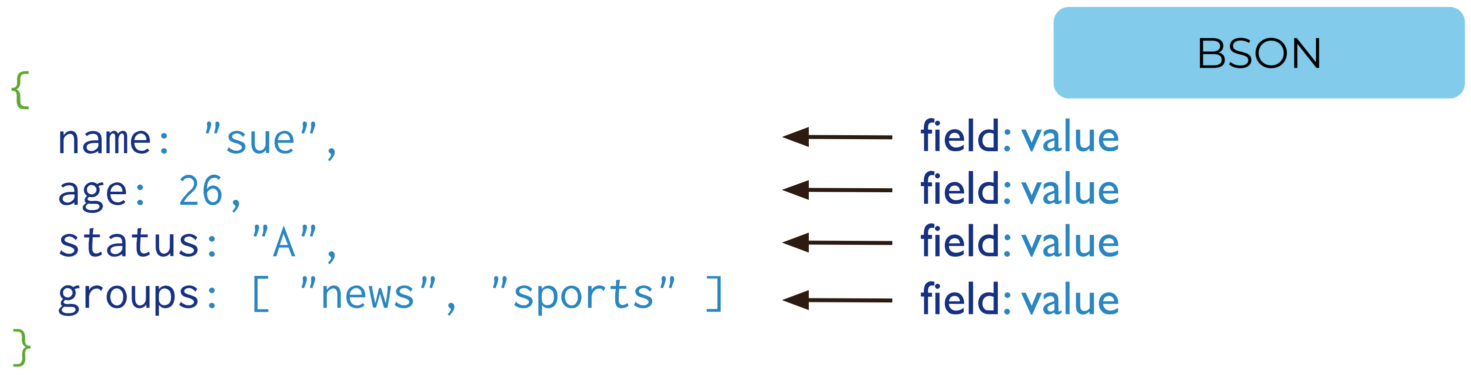
Basically a JSON



```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

MongoDB – Document

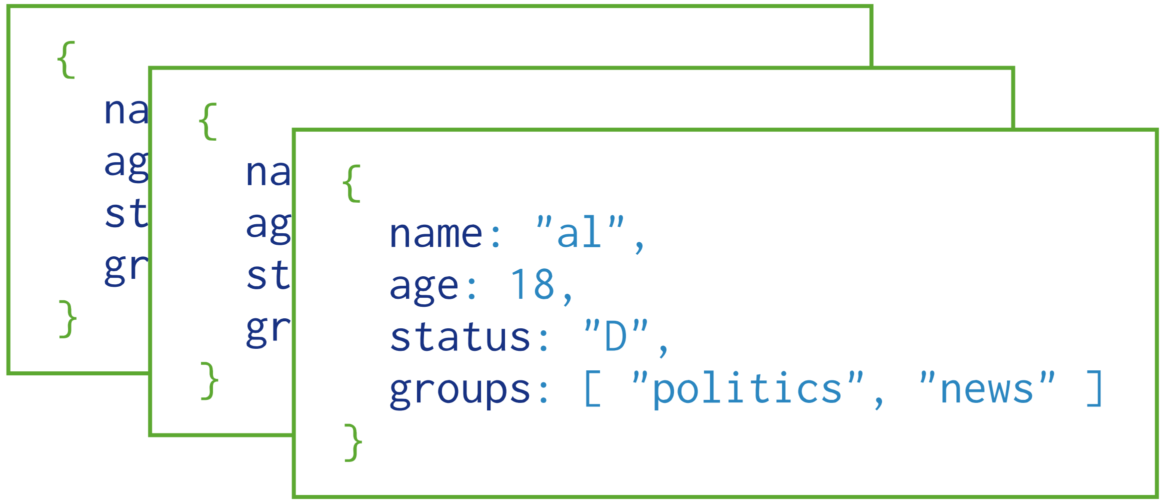
A record in MongoDB is a document, which is a data structure composed of field and value pairs.



```
{
   name: "sue",
   age: 26,
   status: "A",
   groups: [ "news", "sports" ]
}
```

MongoDB - Collections

Documents are stored in Collections

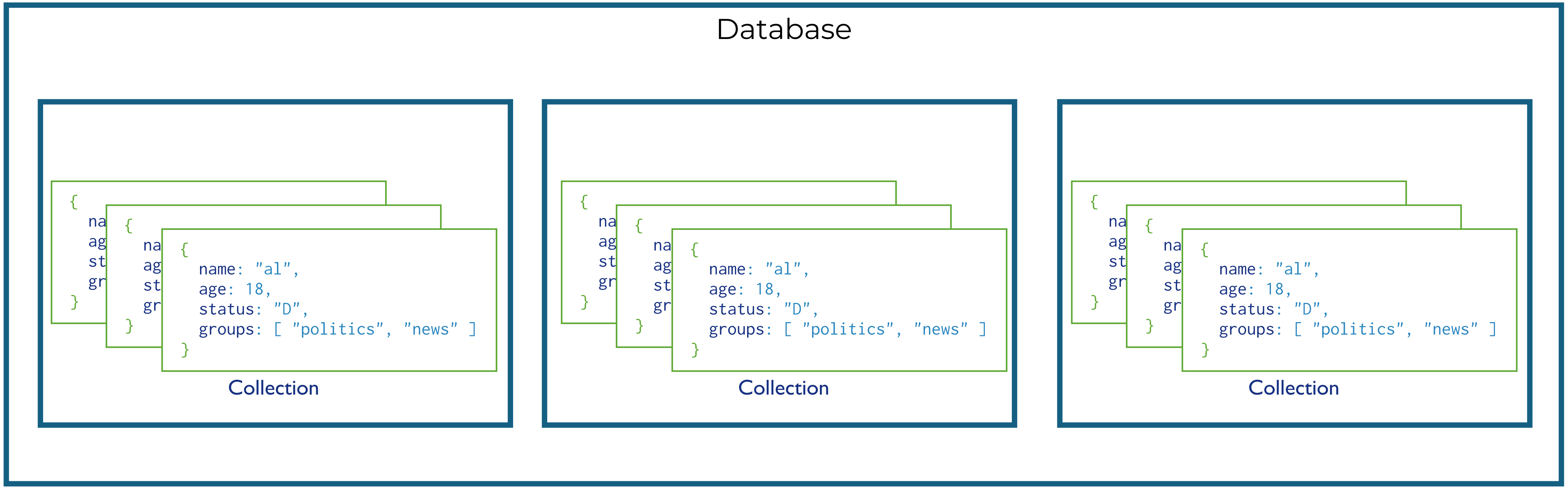


```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Collection

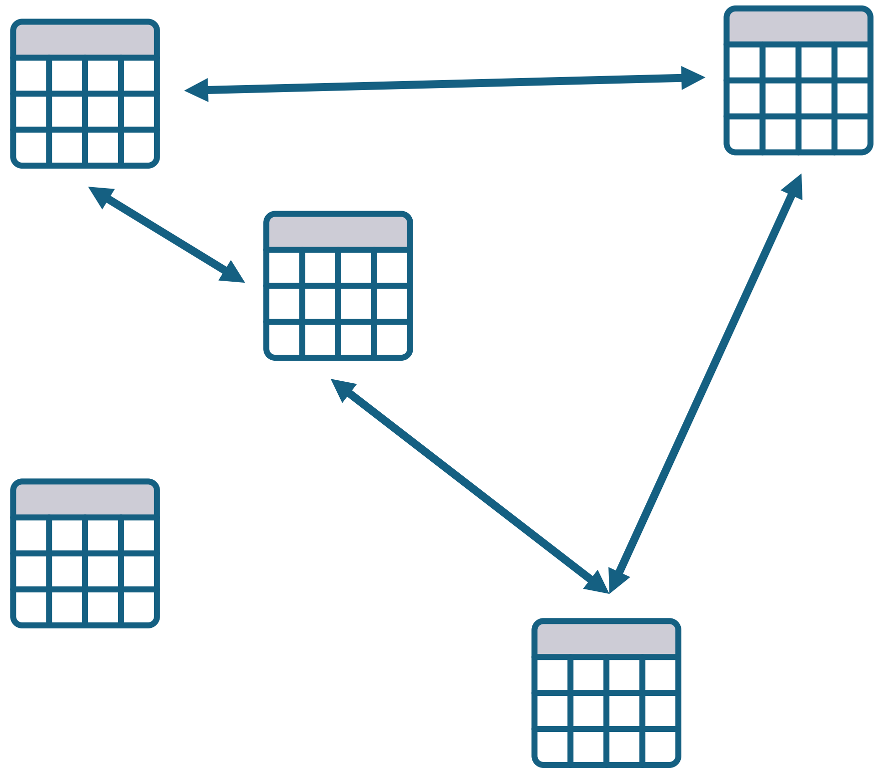
Comparable to tables in relational databases

MongoDB - Database



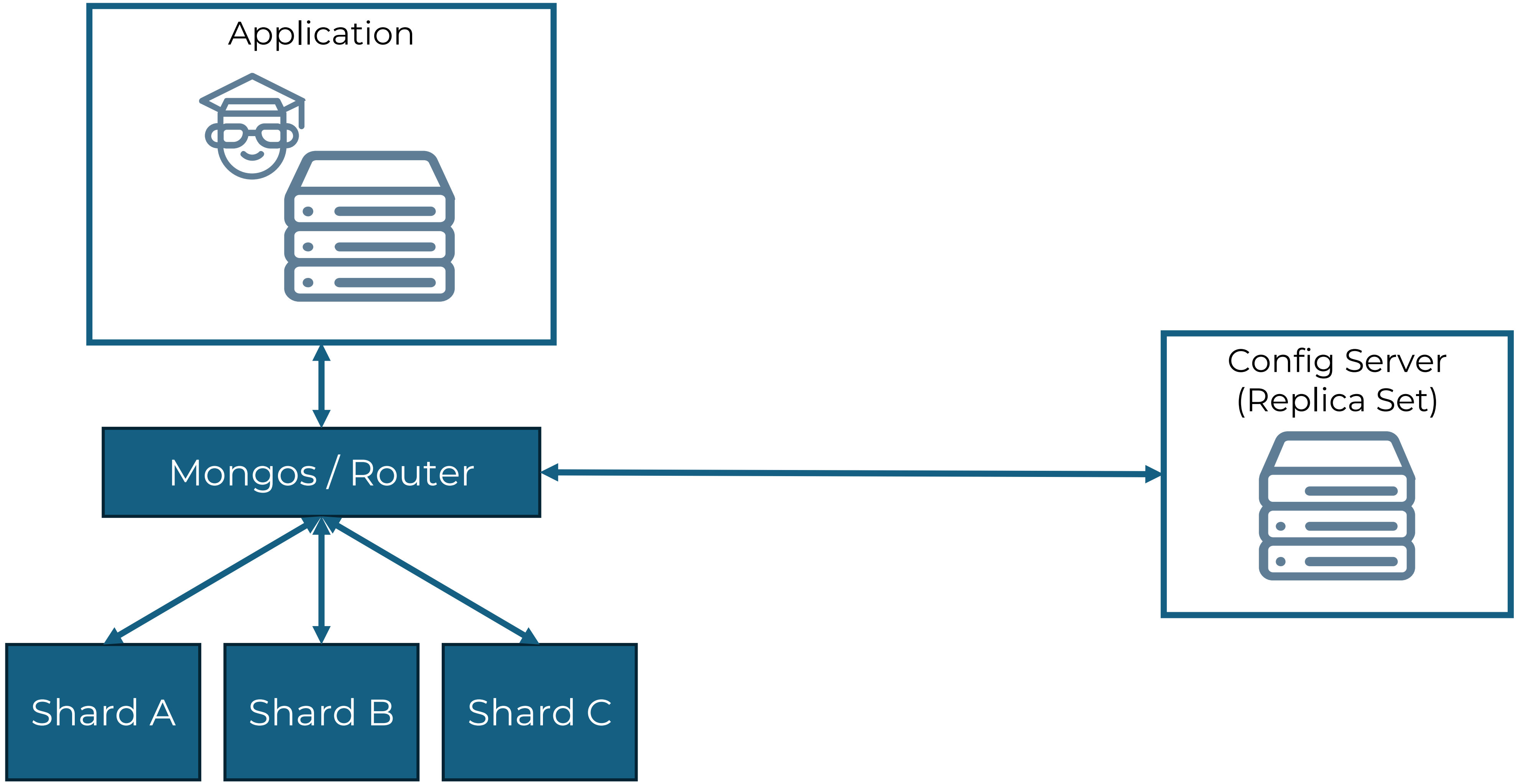
```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Relational vs No Relational DB

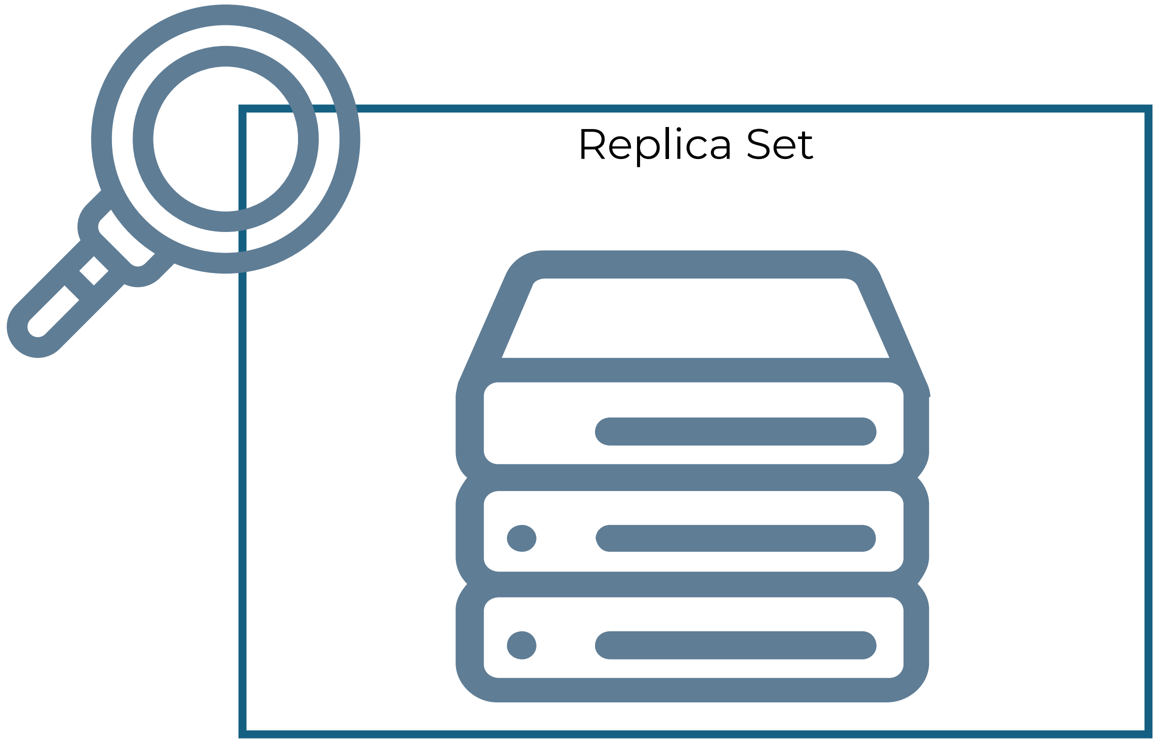


```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

MongoDB - Server Types

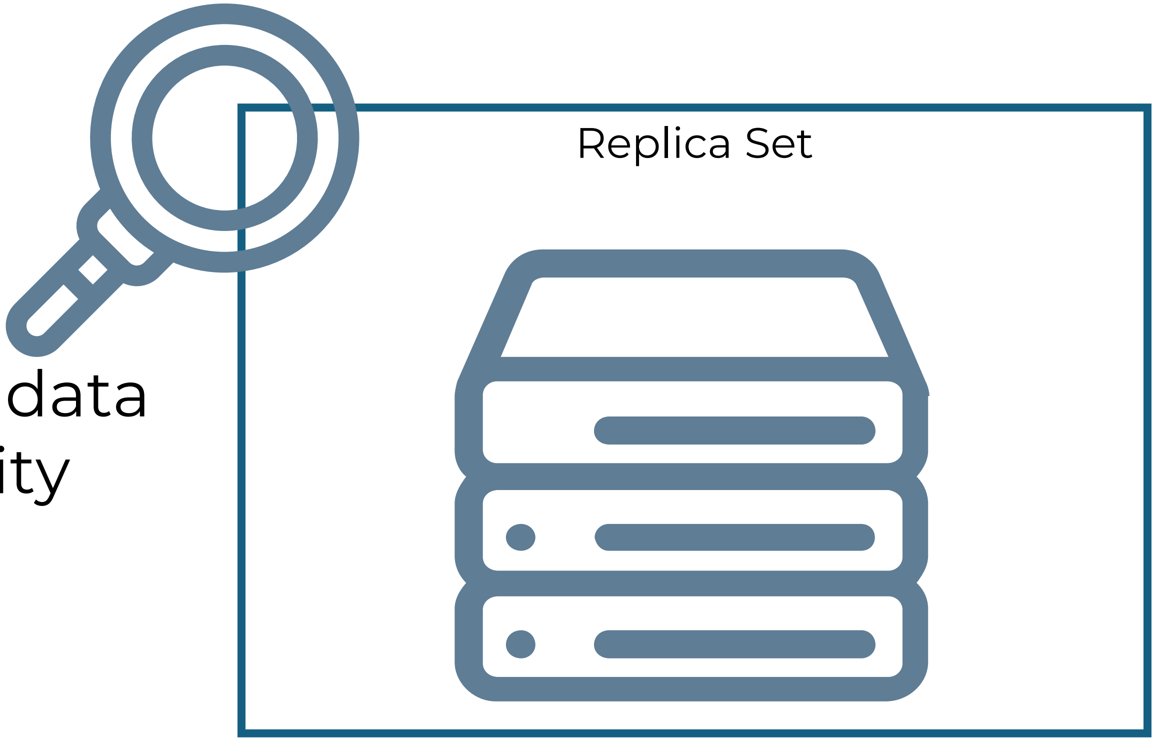


MongoDB – Replica Set



Group of servers that maintain the same data set

MongoDB – Replica Set

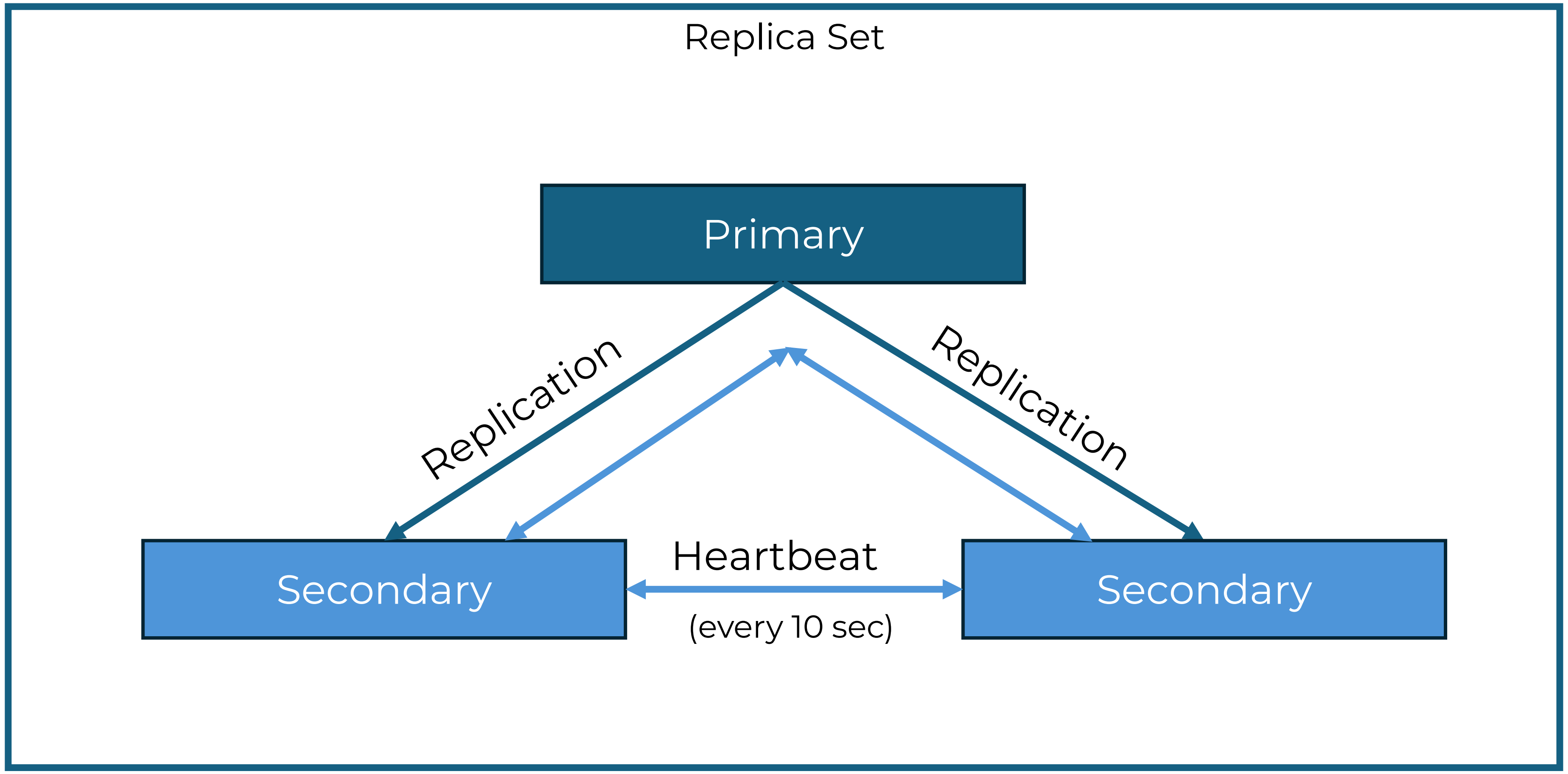


Increasing data availability

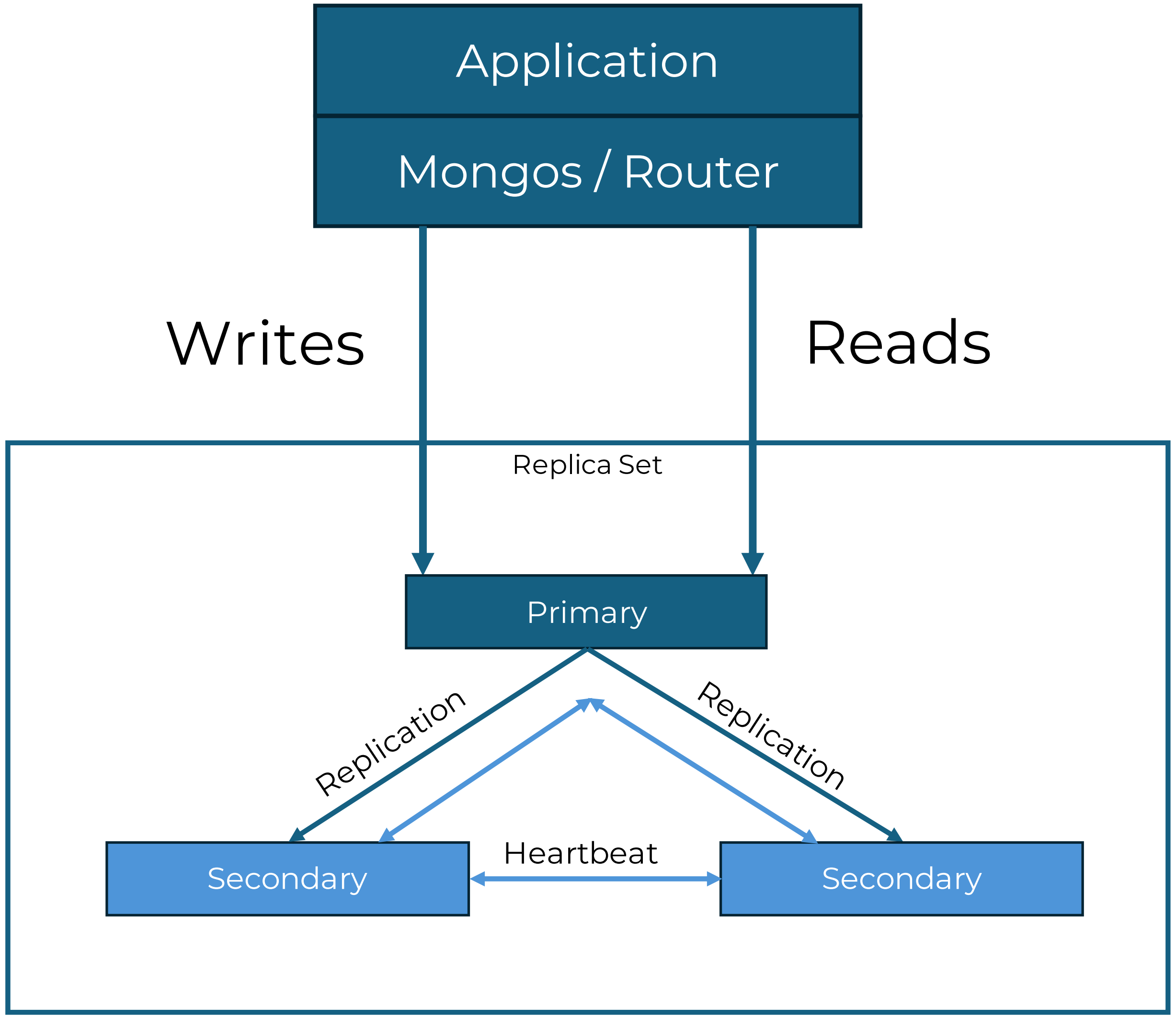
Providing redundancy (fault tolerance)

Group of servers that maintain the same data set

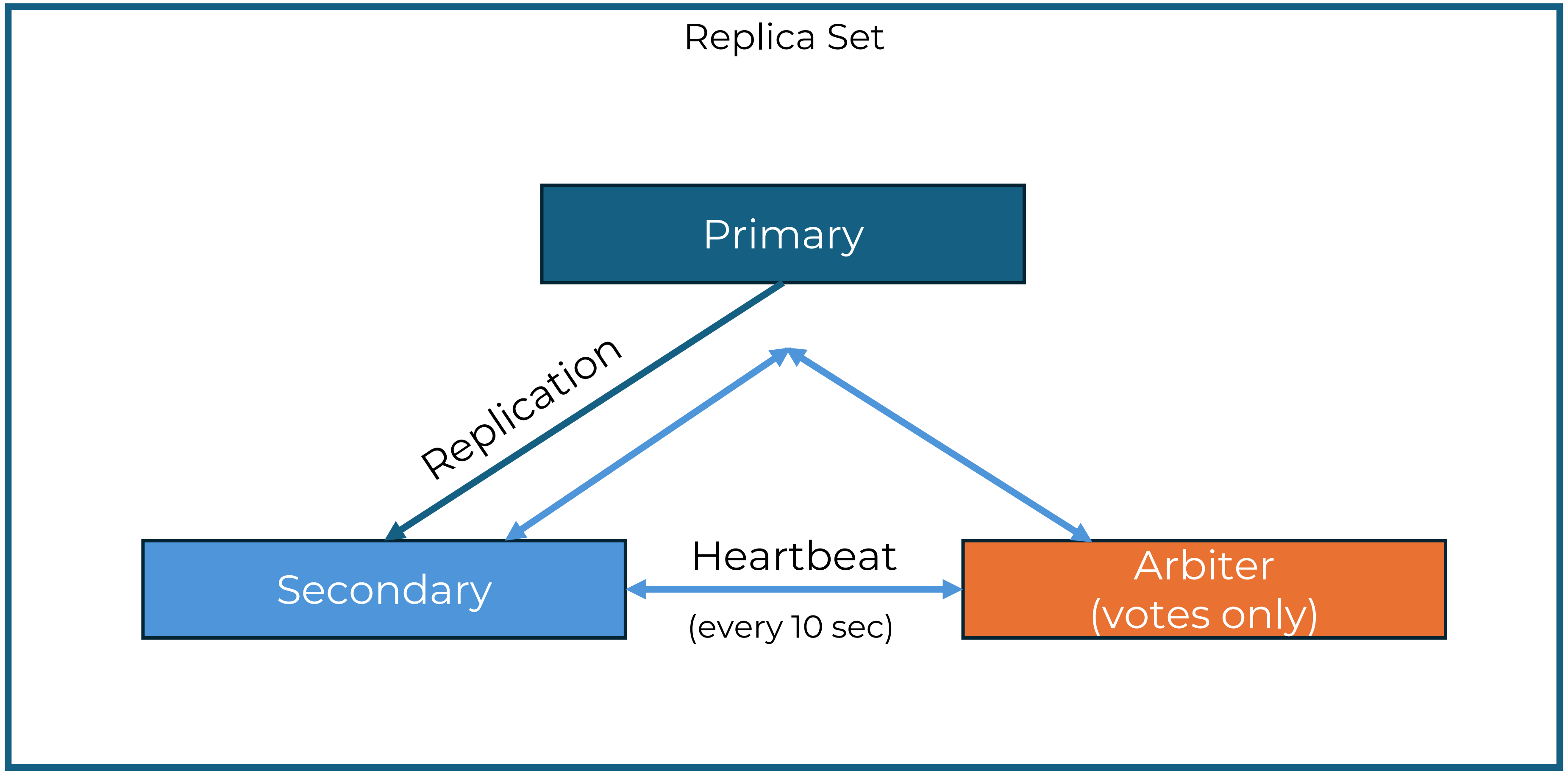
MongoDB – Replica Set



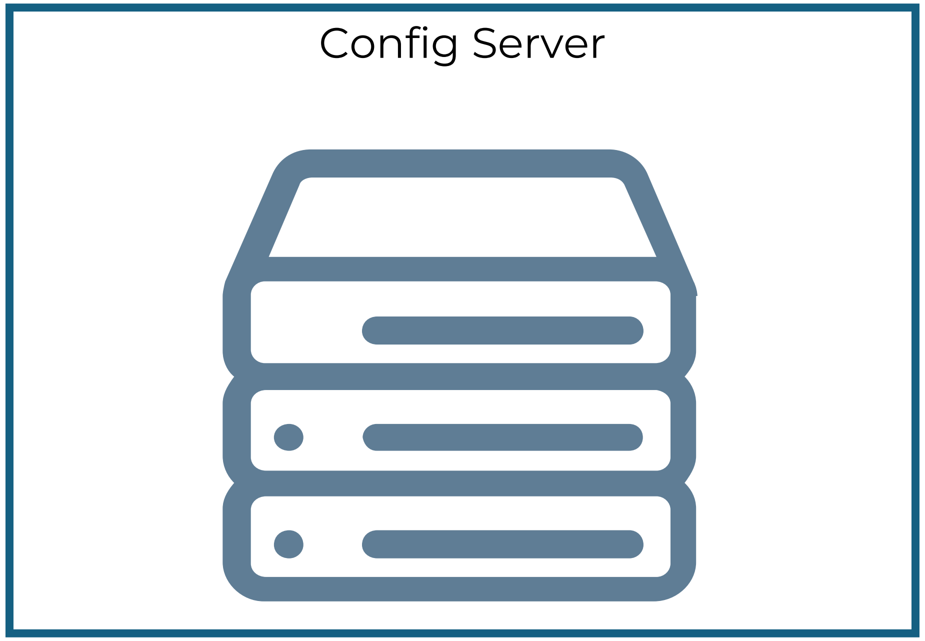
MongoDB – Replica Set



MongoDB – Replica Set (Arbiter)

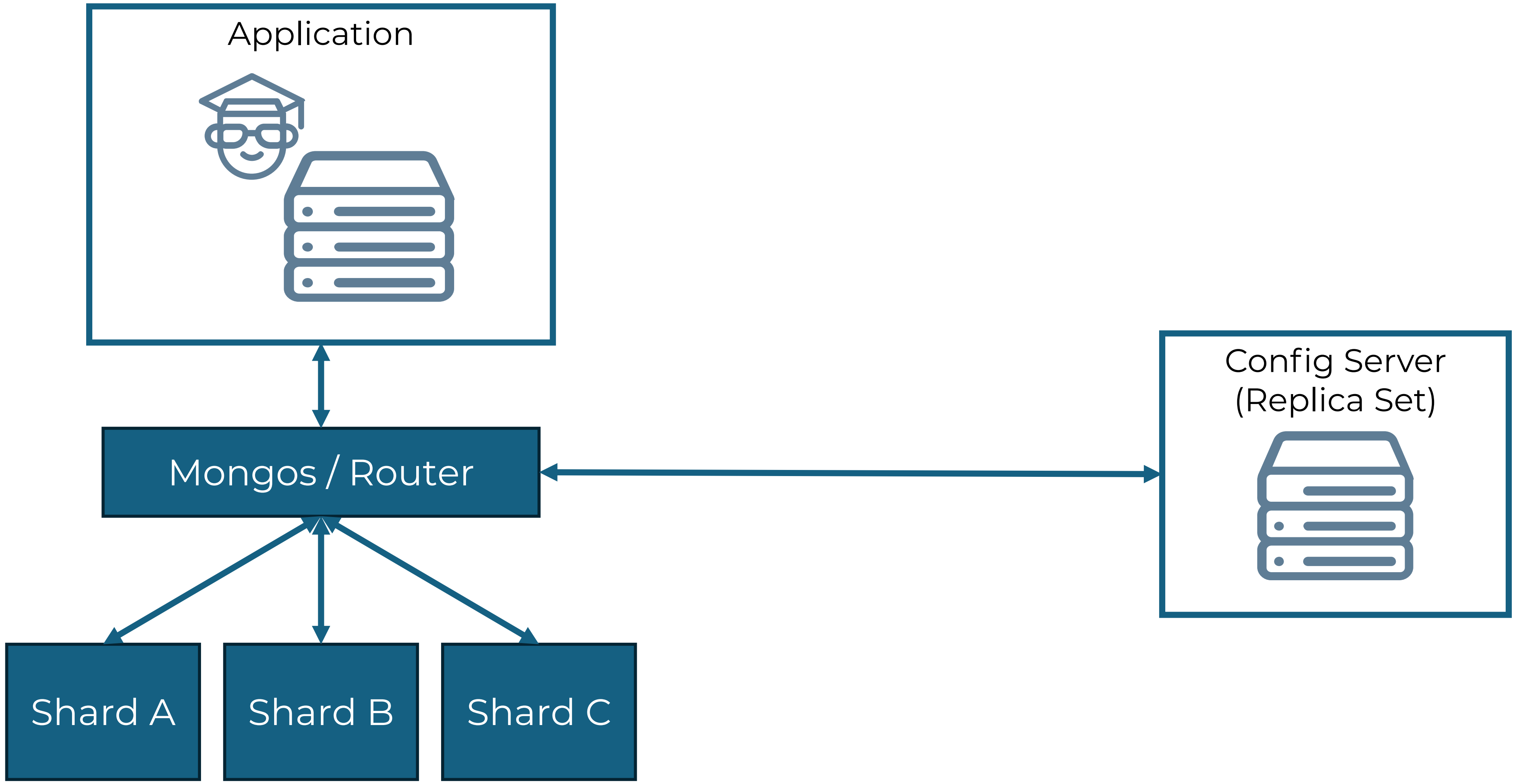


MongoDB – Config Server



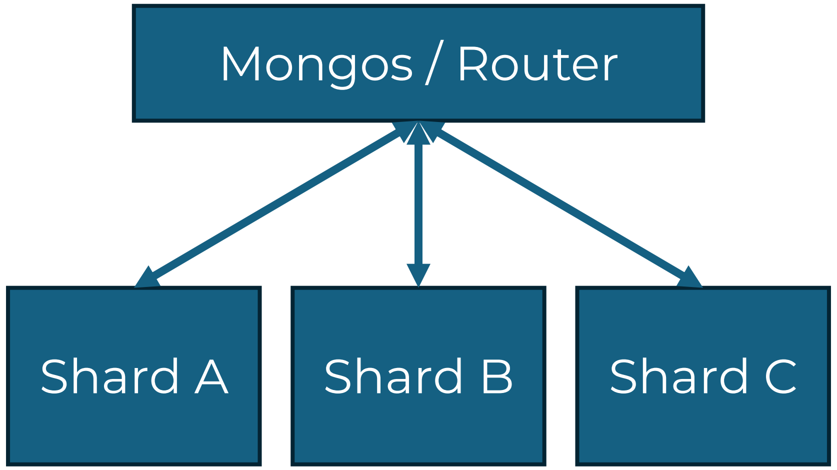
Stores Meta Data for shared clusters

MongoDB - Server Types



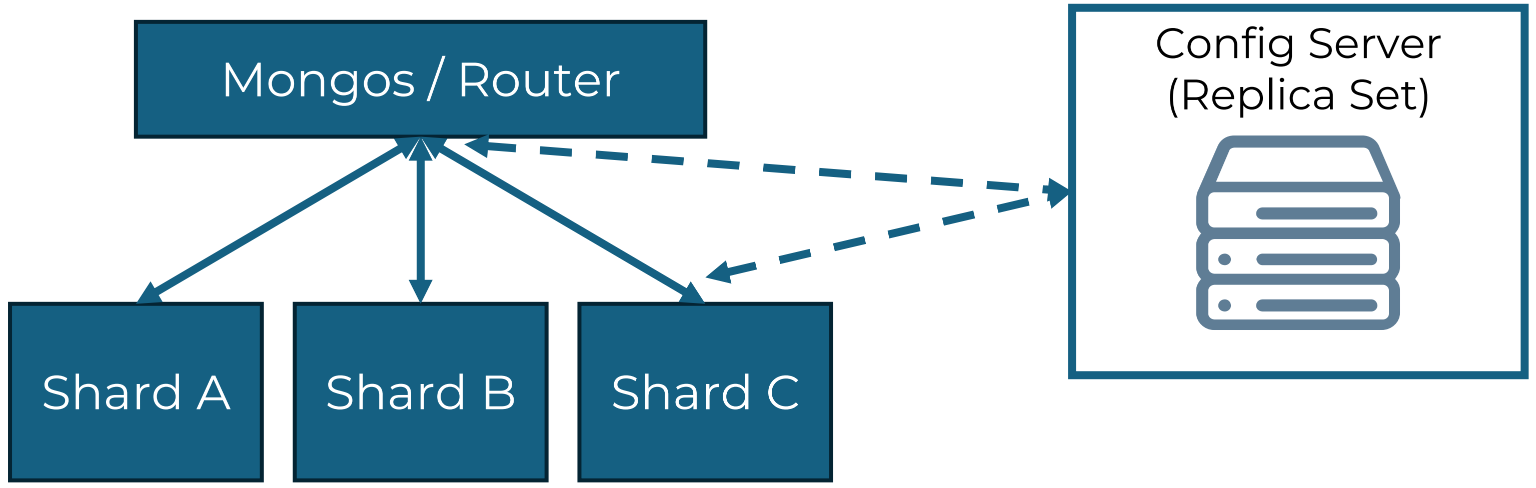
MongoDB - Sharding Strategies

Why do I need sharding?



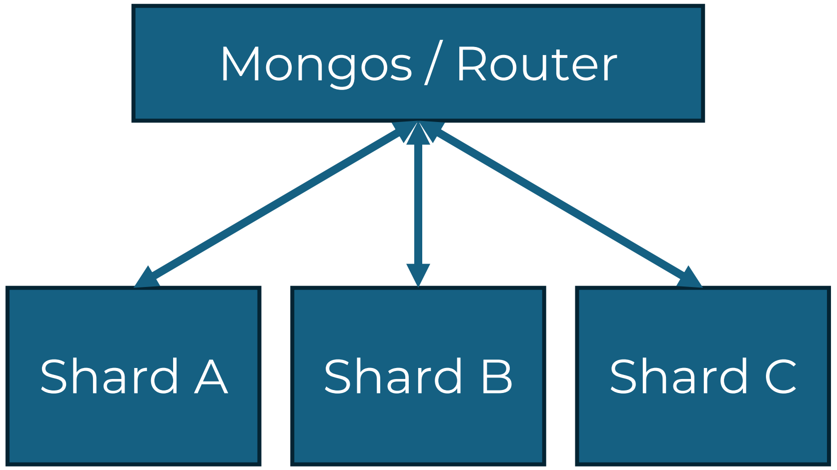
MongoDB - Sharding Strategies

Why do I need sharding?



MongoDB - Sharding Strategies

Why do I need sharding?



Data is bigger than one server can handle!

“…very large data sets and high throughput operations”

MongoDB - Sharding Strategies

Why do I need sharding?

Increase Storage capacity

- Additional disks increas a single clusters size

Linear scaling reads and writes increase the througput

- Parallel processing

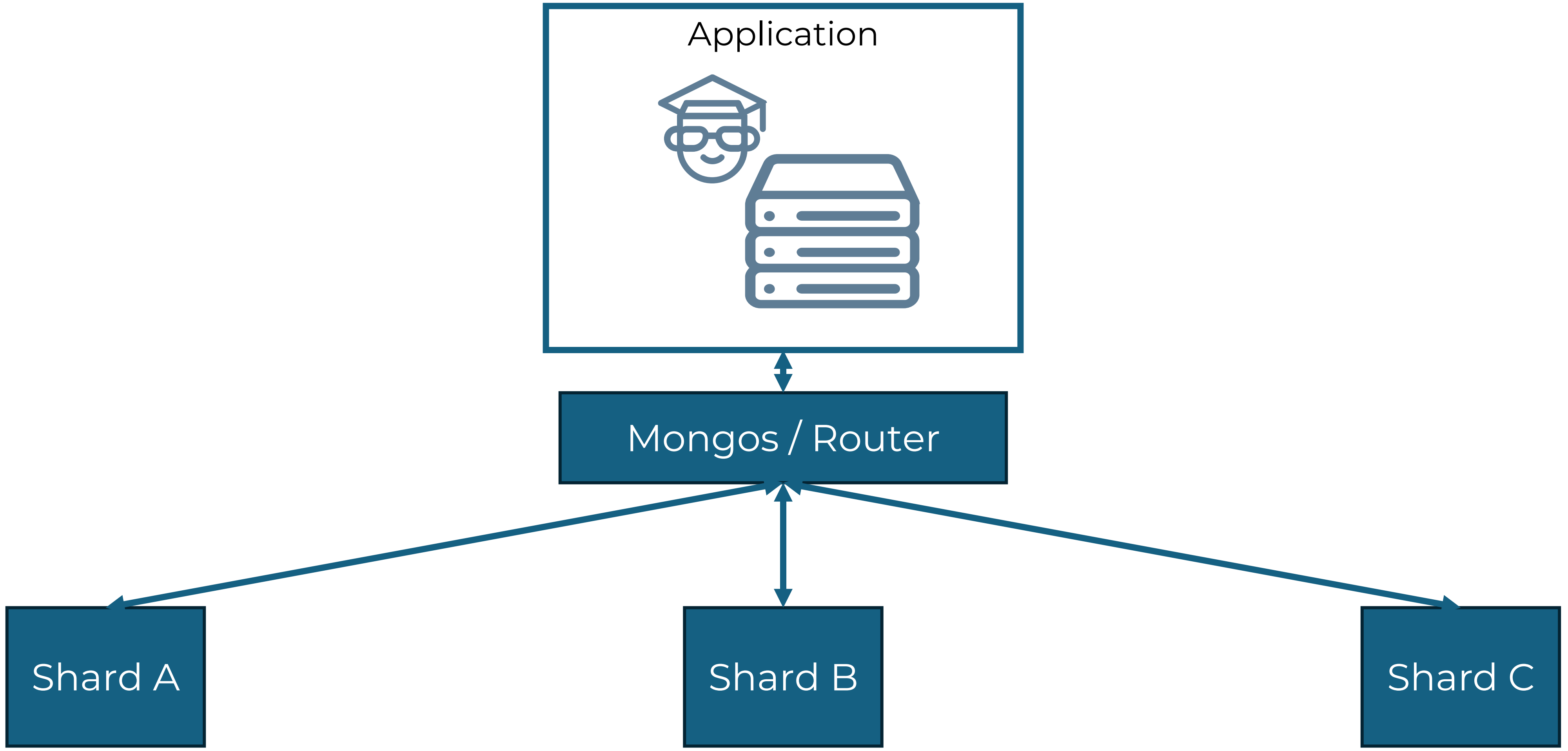
Latency decreases (for reads and writes)

- More RAM improves system performance

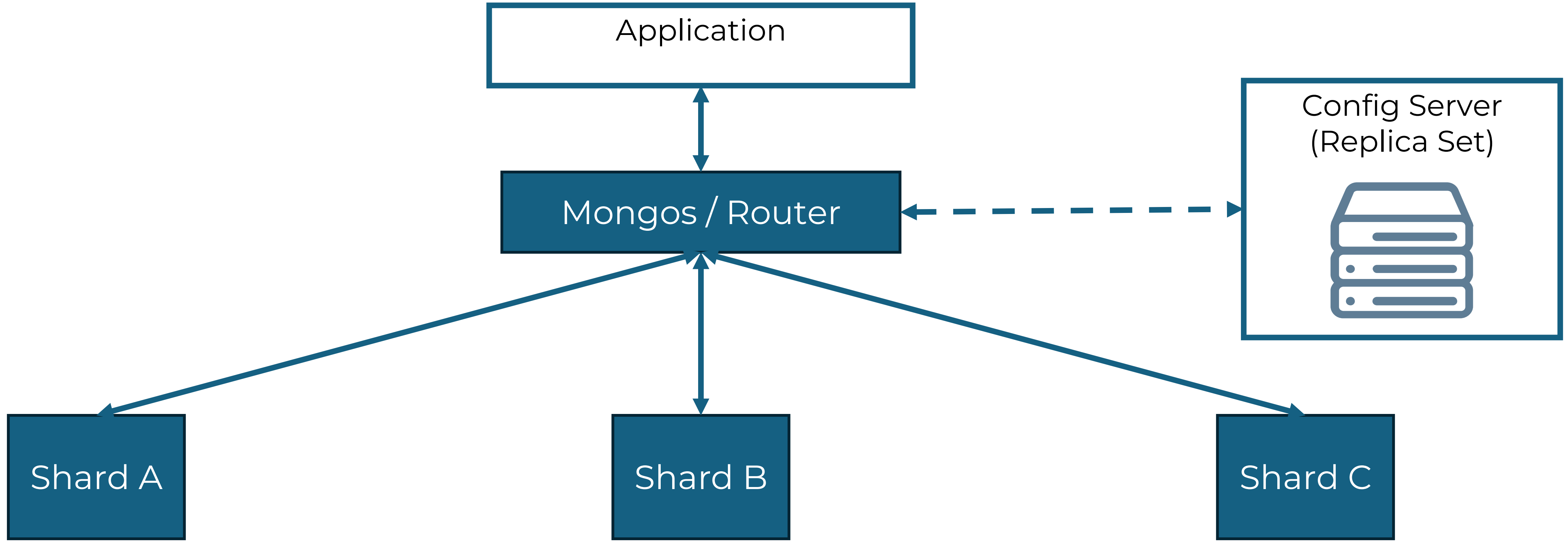
Disaster Recovery gets more efficent

- Restores can be parallelised

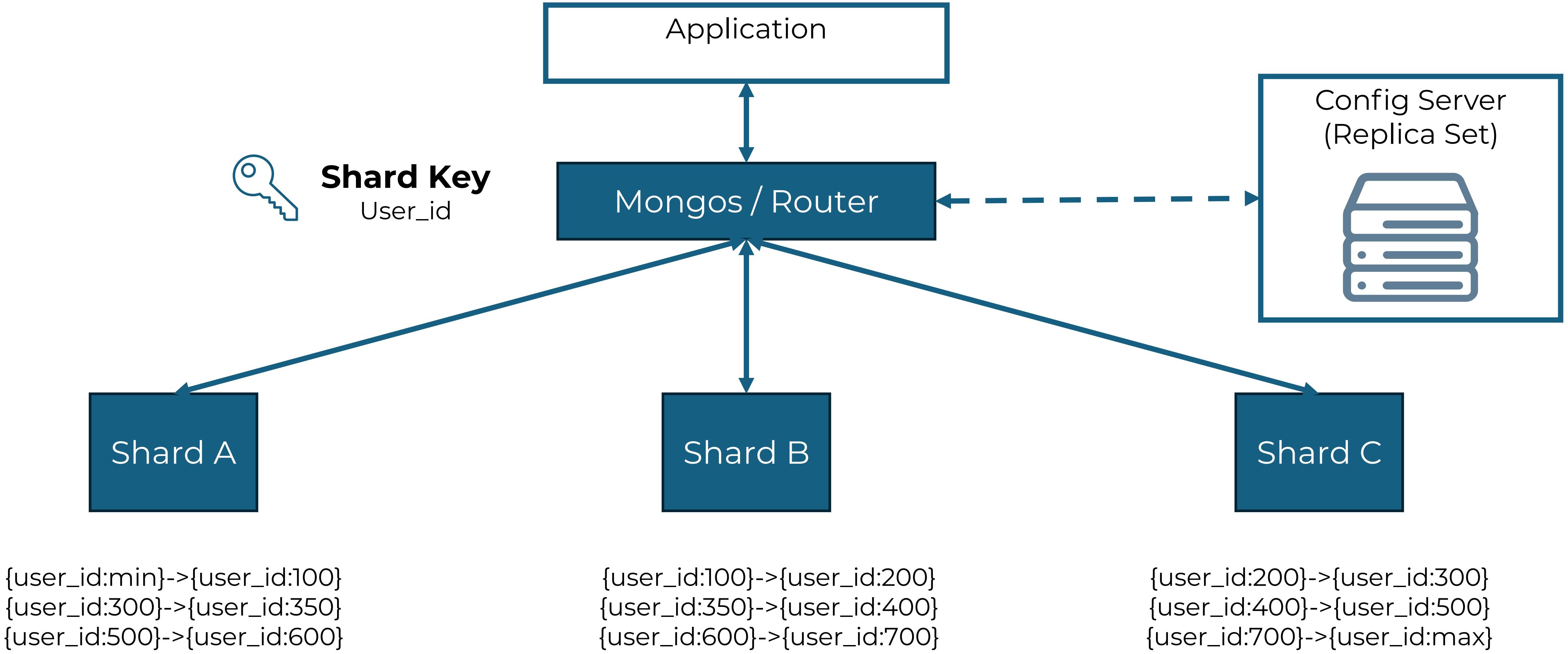
MongoDB – Sharding



MongoDB – Shard Key

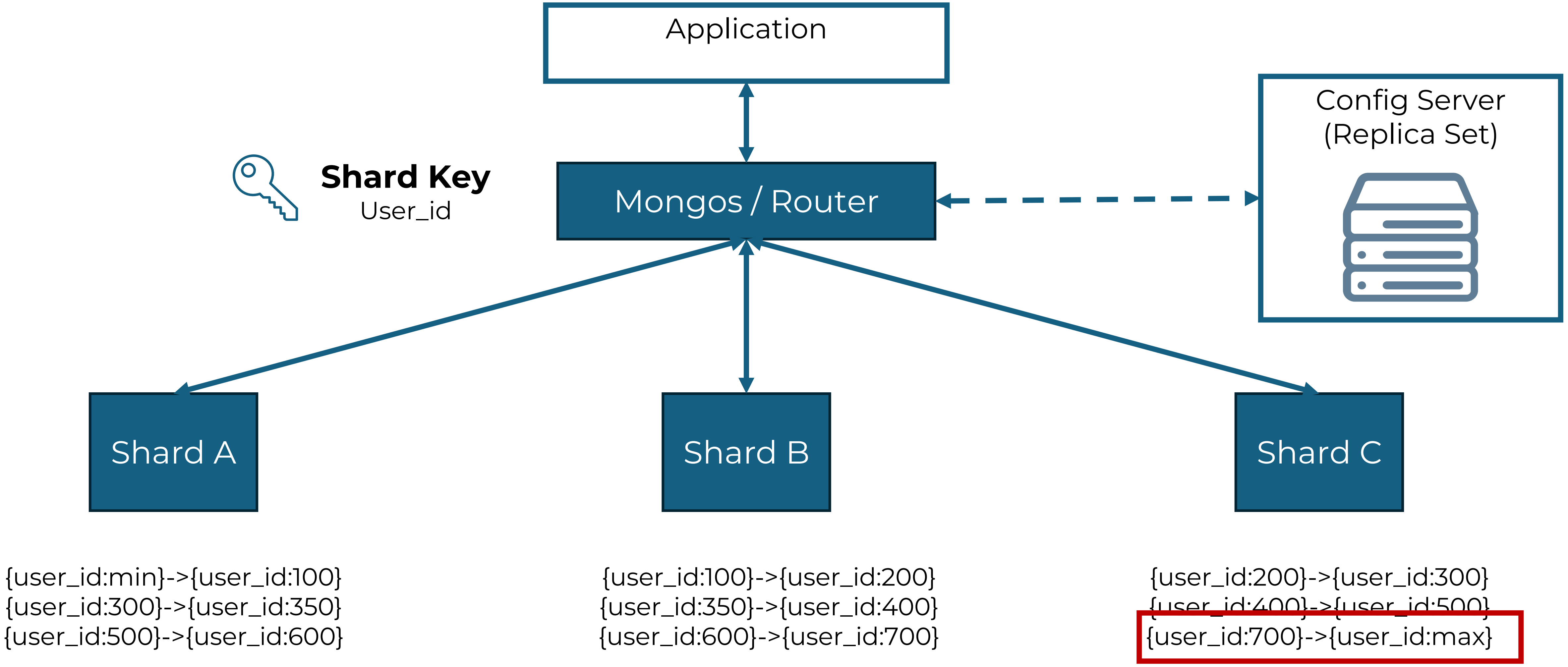


MongoDB – Shard Key



Application
Config Server
(Replica Set)
Shard Key
User_id
Mongos / Router
Shard A
Shard B
Shard C
{user_id:min}->{user_id:100}
{user_id:300}->{user_id:350}
{user_id:500}->{user_id:600}
{user_id:100}->{user_id:200}
{user_id:350}->{user_id:400}
{user_id:600}->{user_id:700}
{user_id:200}->{user_id:300}
{user_id:400}->{user_id:500}
{user_id:700}->{user_id:max}

MongoDB – Ranged Sharding

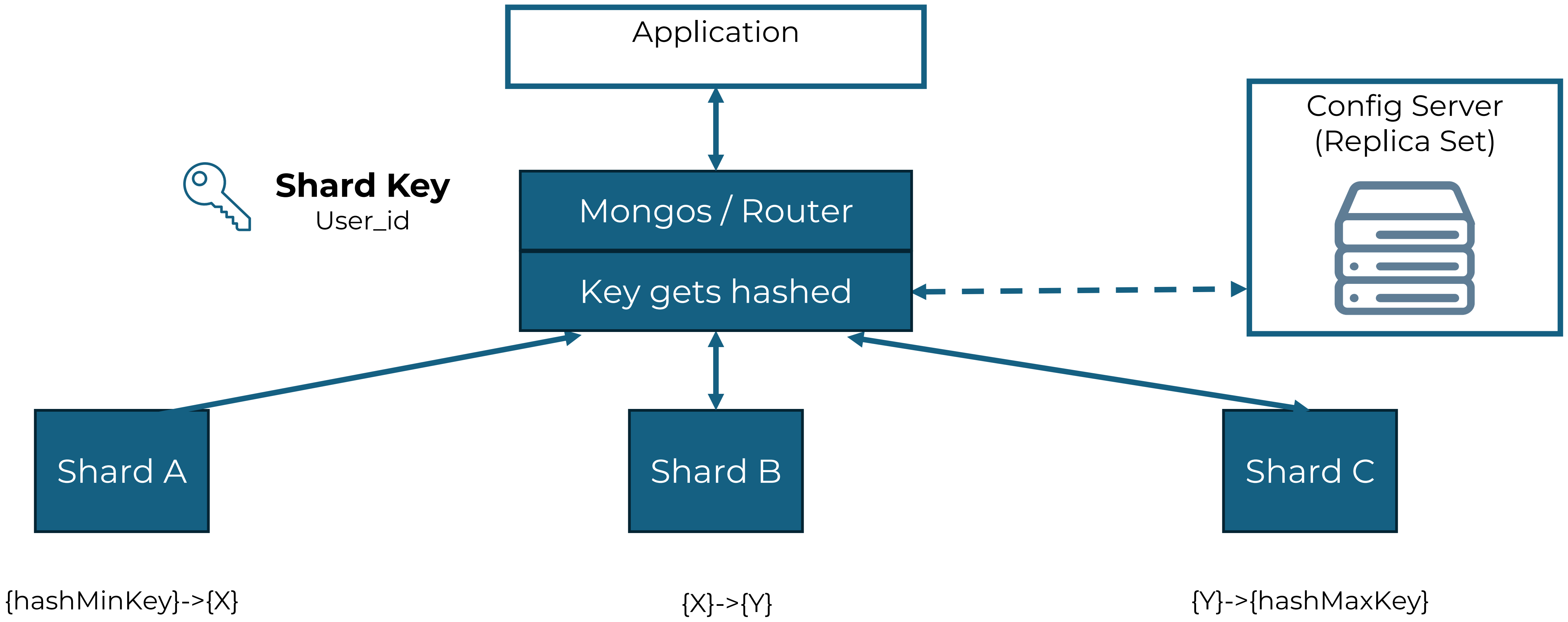


{user_id:min}->{user_id:100}
{user_id:300}->{user_id:350}
{user_id:500}->{user_id:600}

{user_id:100}->{user_id:200}
{user_id:350}->{user_id:400}
{user_id:600}->{user_id:700}

{user_id:200}->{user_id:300}
{user_id:400}->{user_id:500}
{user_id:700}->{user_id:max}

MongoDB – Hashed Sharding



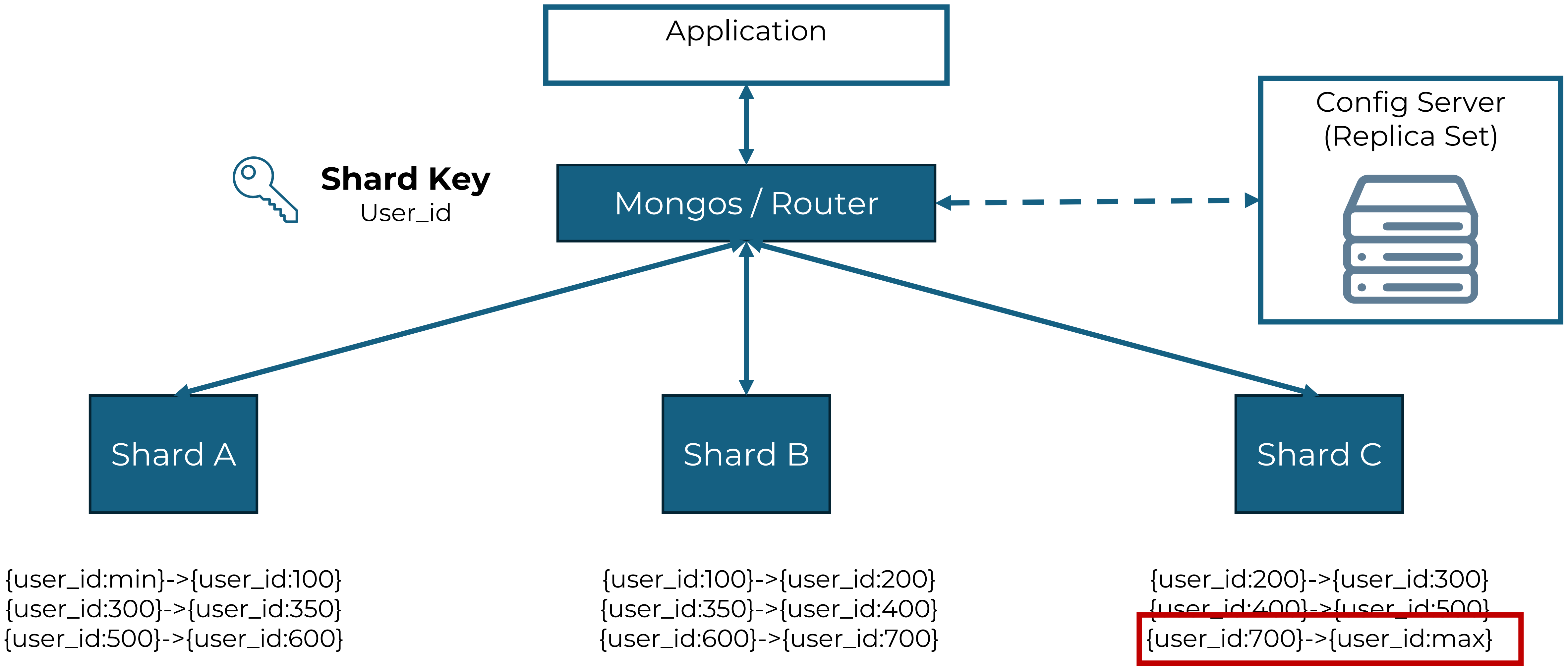
MongoDB – Hashed Sharding

Hashed sharding uses a hashed value as the shard key

- Even data distribution across shards
- Even read / write workload distribution

- Range queries can get very compute intense

MongoDB – Ranged Sharding



{user_id:min}->{user_id:100}
{user_id:300}->{user_id:350}
{user_id:500}->{user_id:600}

{user_id:100}->{user_id:200}
{user_id:350}->{user_id:400}
{user_id:600}->{user_id:700}

{user_id:200}->{user_id:300}
{user_id:400}->{user_id:500}
{user_id:700}->{user_id:max}

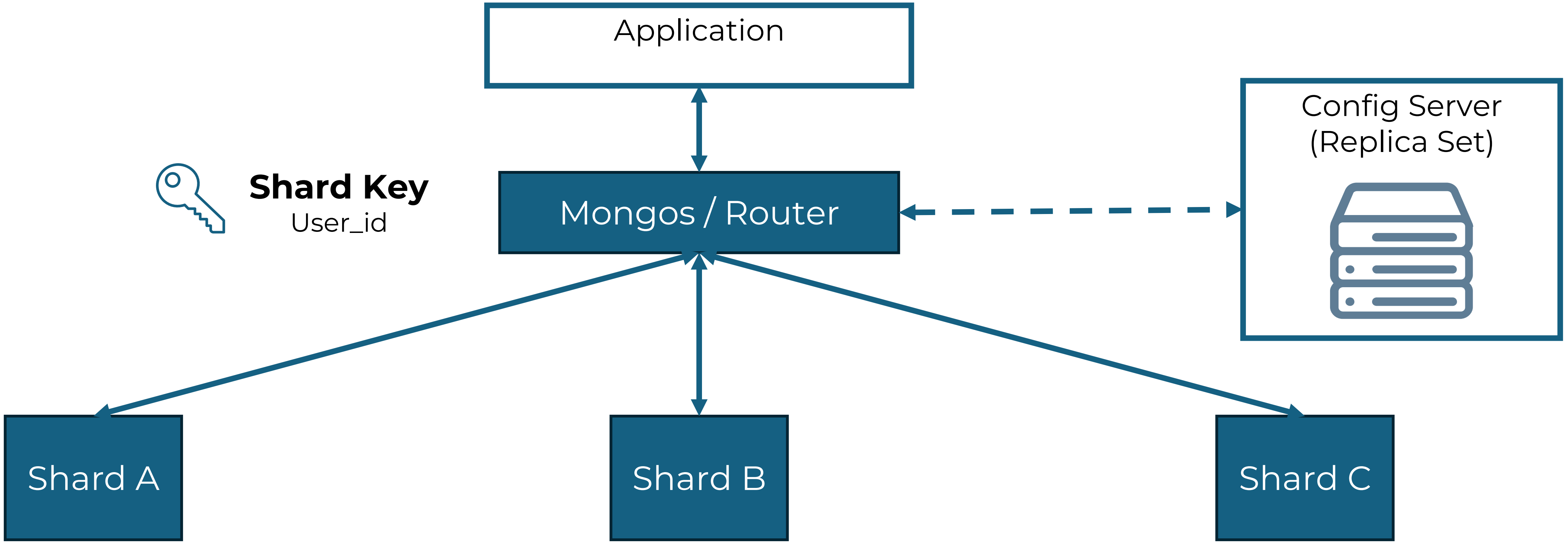
MongoDB – Ranged Sharding

Queries can read target documents within a contiguous range

- Ideal for targeted operations

- MaxKey shard receives majority of incoming writes
- Reduces advantage of distributed writes in a shared cluster

MongoDB – Ranged Sharding (Balancer)

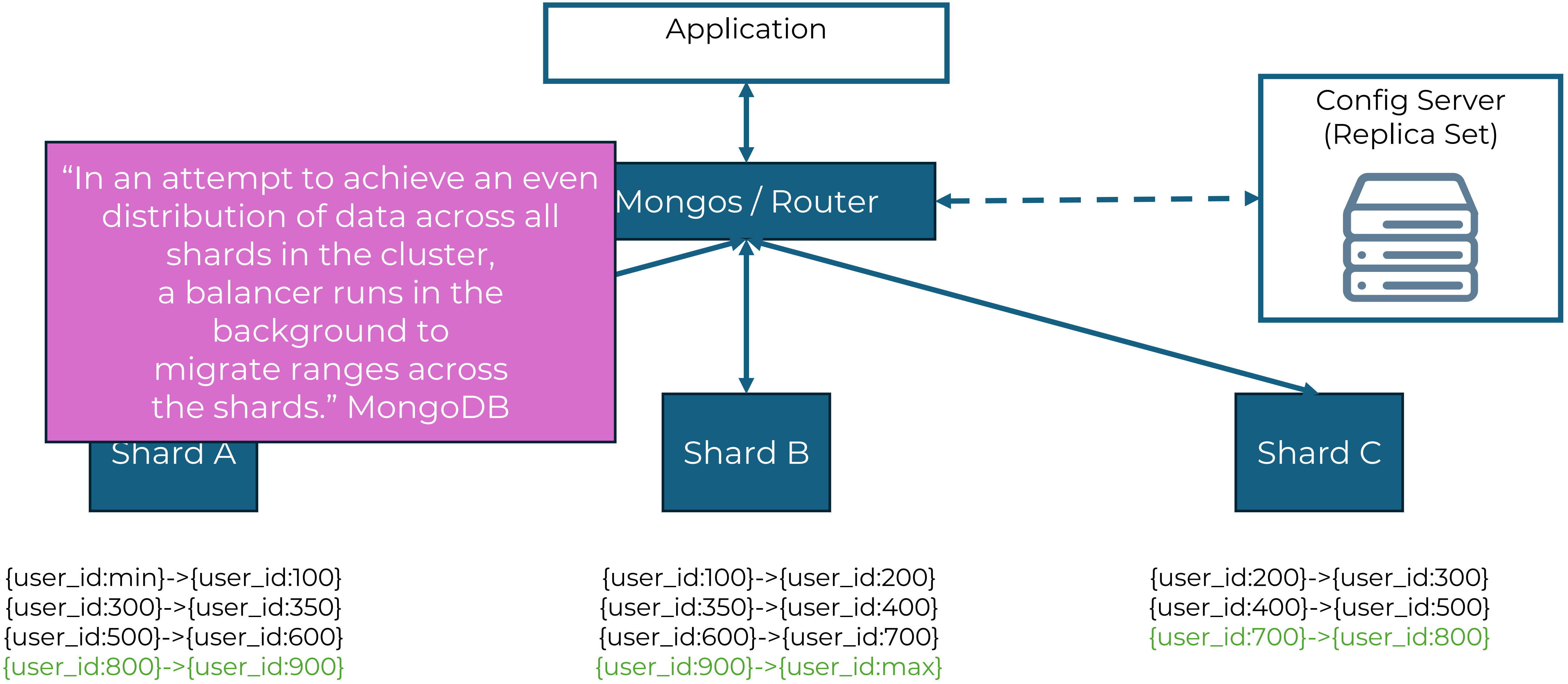


{user_id:min}->{user_id:100}
{user_id:300}->{user_id:350}
{user_id:500}->{user_id:600}
{user_id:800}->{user_id:900}

{user_id:100}->{user_id:200}
{user_id:350}->{user_id:400}
{user_id:600}->{user_id:700}
{user_id:900}->{user_id:max}

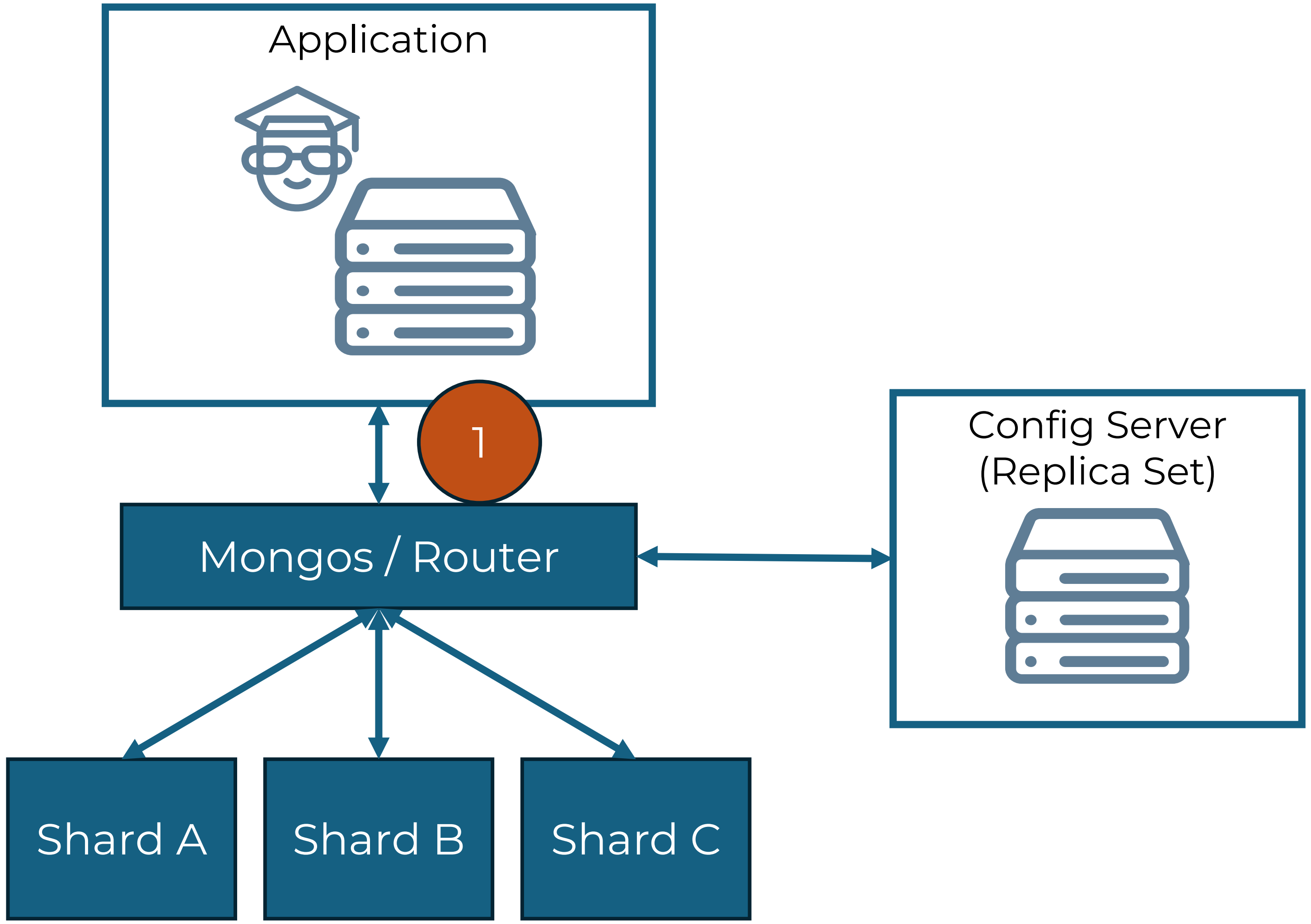
{user_id:200}->{user_id:300}
{user_id:400}->{user_id:500}
{user_id:700}->{user_id:800}

MongoDB – Ranged Sharding (Balancer)


Application
Mongos / Router
Config Server (Replica Set)
Shard A
Shard B
Shard C
{user_id:min}->{user_id:100}
{user_id:300}->{user_id:350}
{user_id:500}->{user_id:600}
{user_id:800}->{user_id:900}
{user_id:100}->{user_id:200}
{user_id:350}->{user_id:400}
{user_id:600}->{user_id:700}
{user_id:900}->{user_id:max}
{user_id:200}->{user_id:300}
{user_id:400}->{user_id:500}
{user_id:700}->{user_id:800}

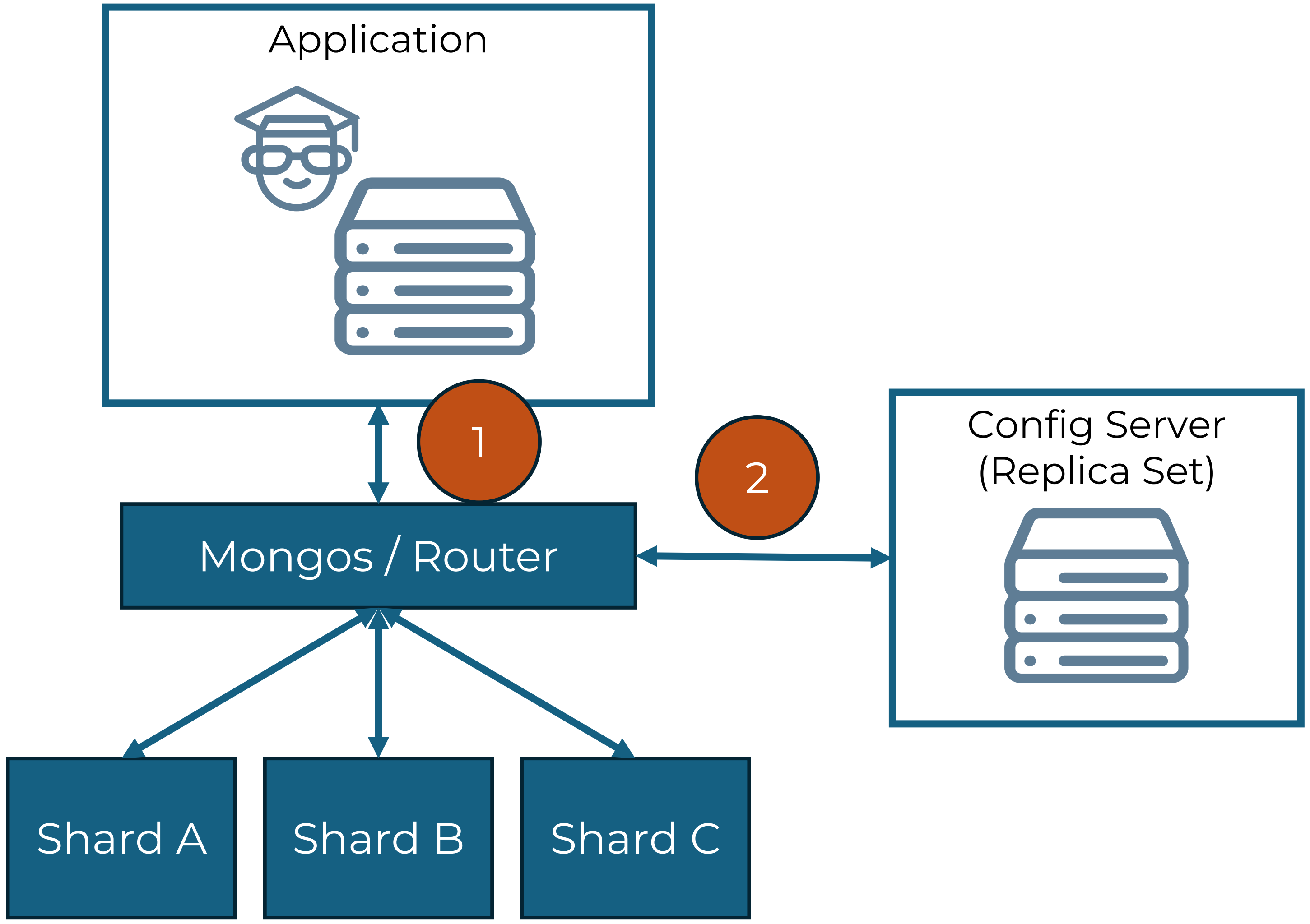
"In an attempt to achieve an even distribution of data across all shards in the cluster, a balancer runs in the background to migrate ranges across the shards." MongoDB

MongoDB – Query example



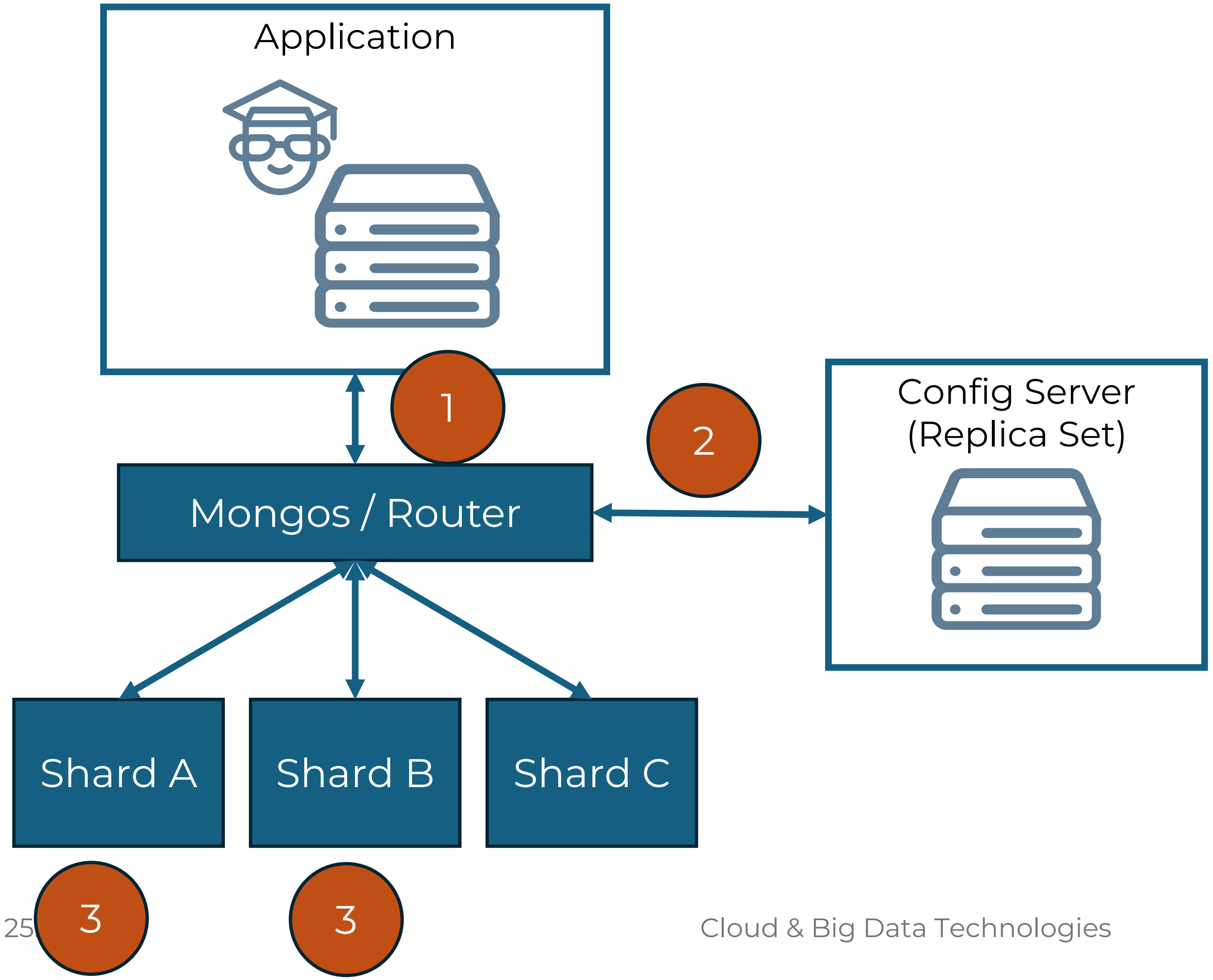
1	Client makes request

MongoDB – Query example



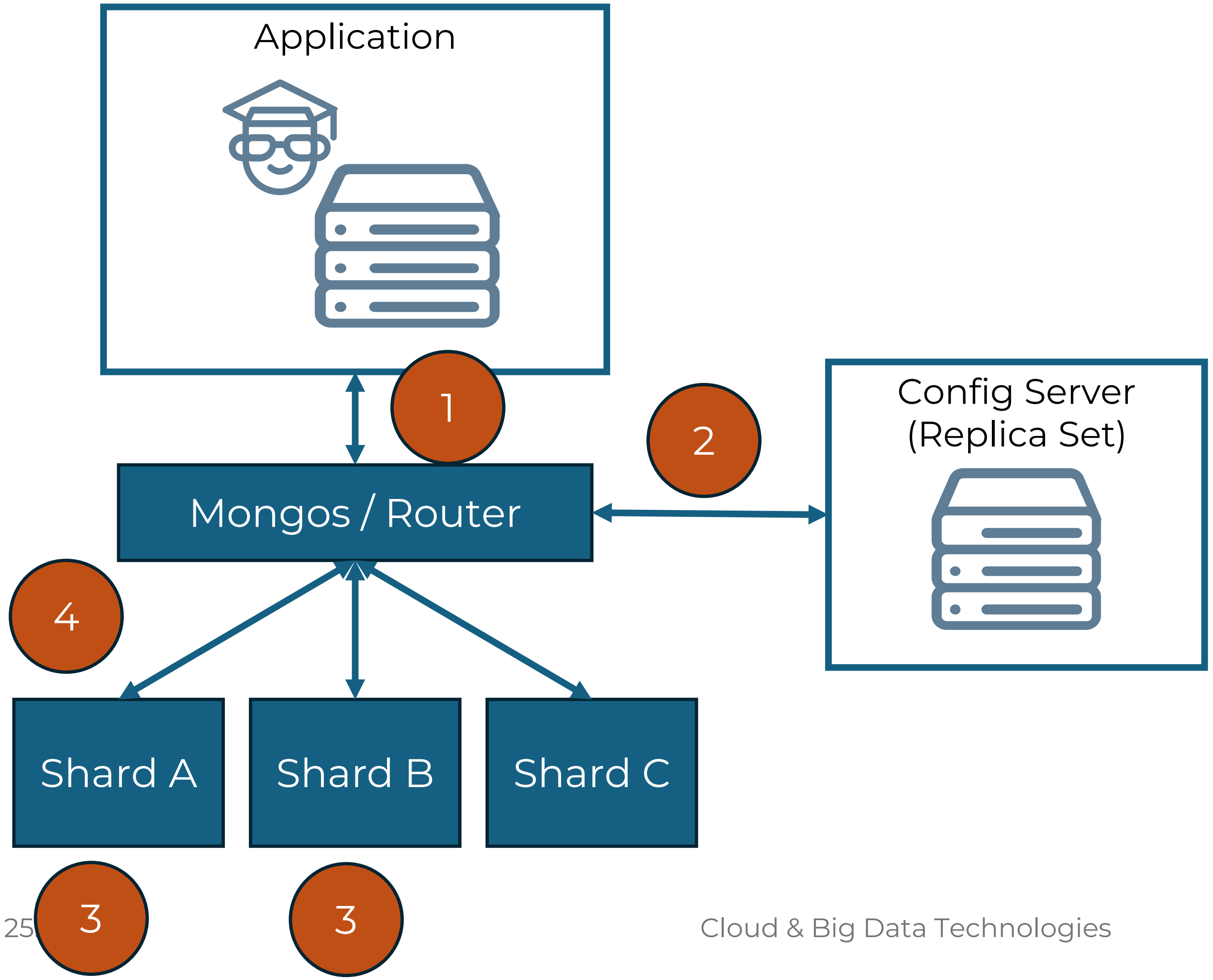
1	Client makes request
2	Mongos determines on which shards the data is available (A & B)

MongoDB – Query example



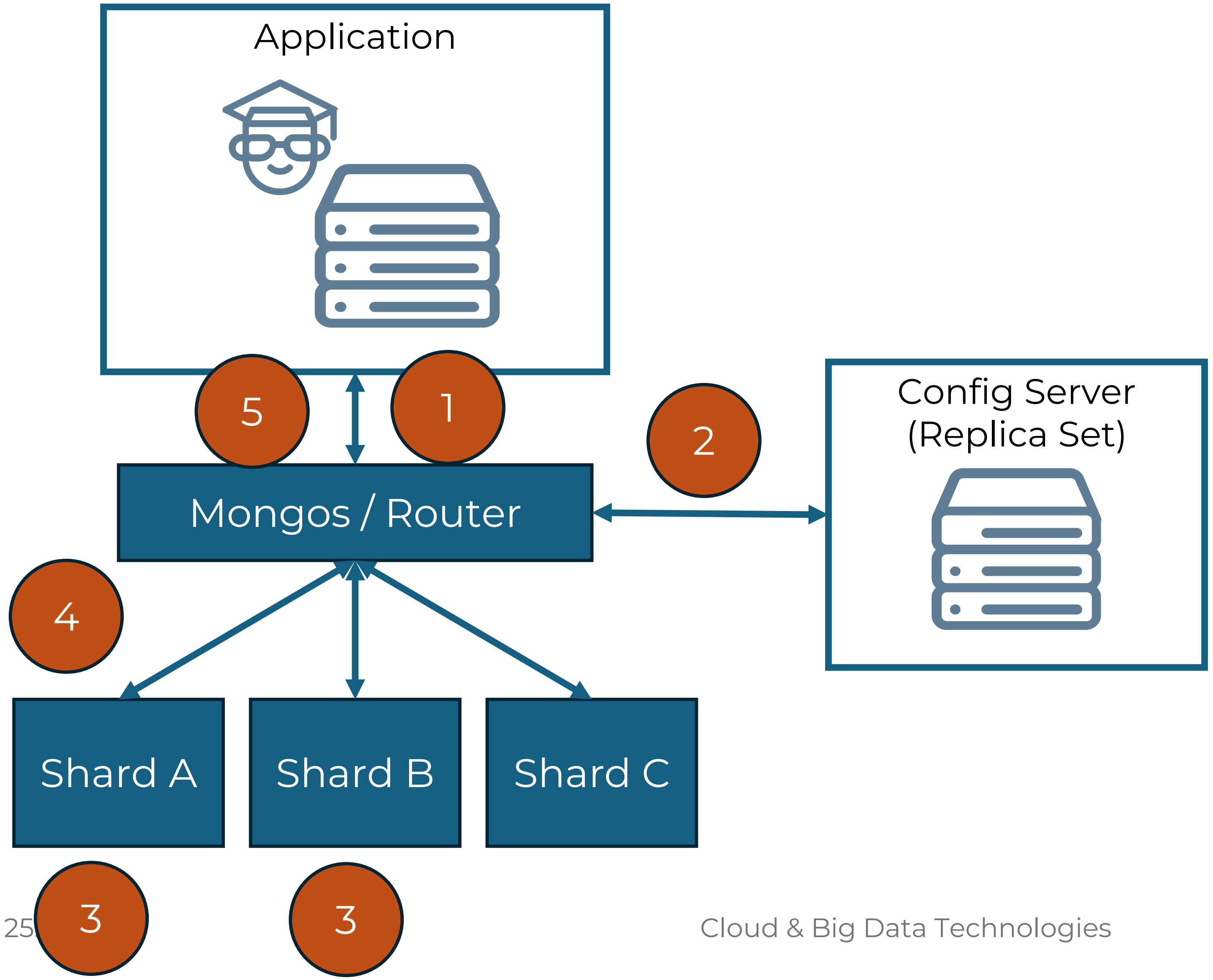
1	Client makes request
2	Mongos determines on which shards the data is available (A & B)
3	Mongos routes request to shard A and B

MongoDB – Query example



1	Client makes request
2	Mongos determines on which shards the data is available (A & B)
3	Mongos routes request to shard A and B
4	Mongos waits for response and merges the results

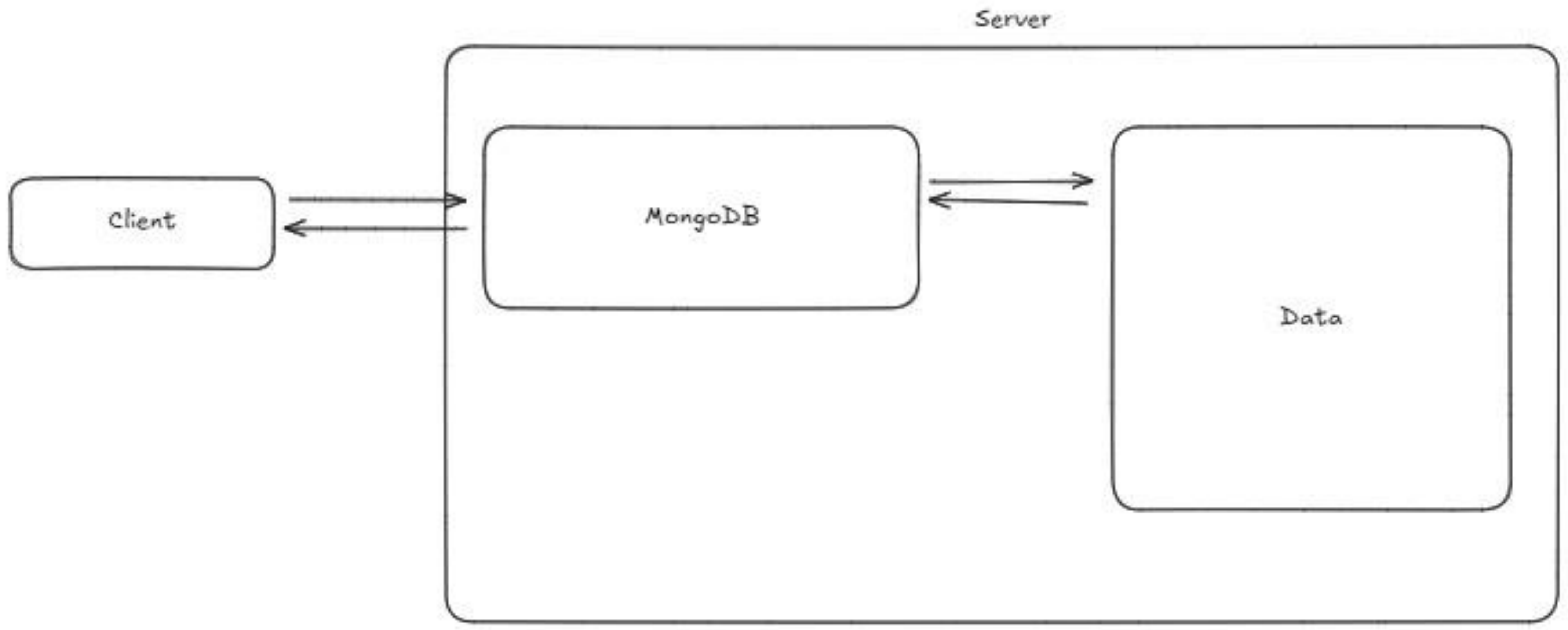
MongoDB – Query example



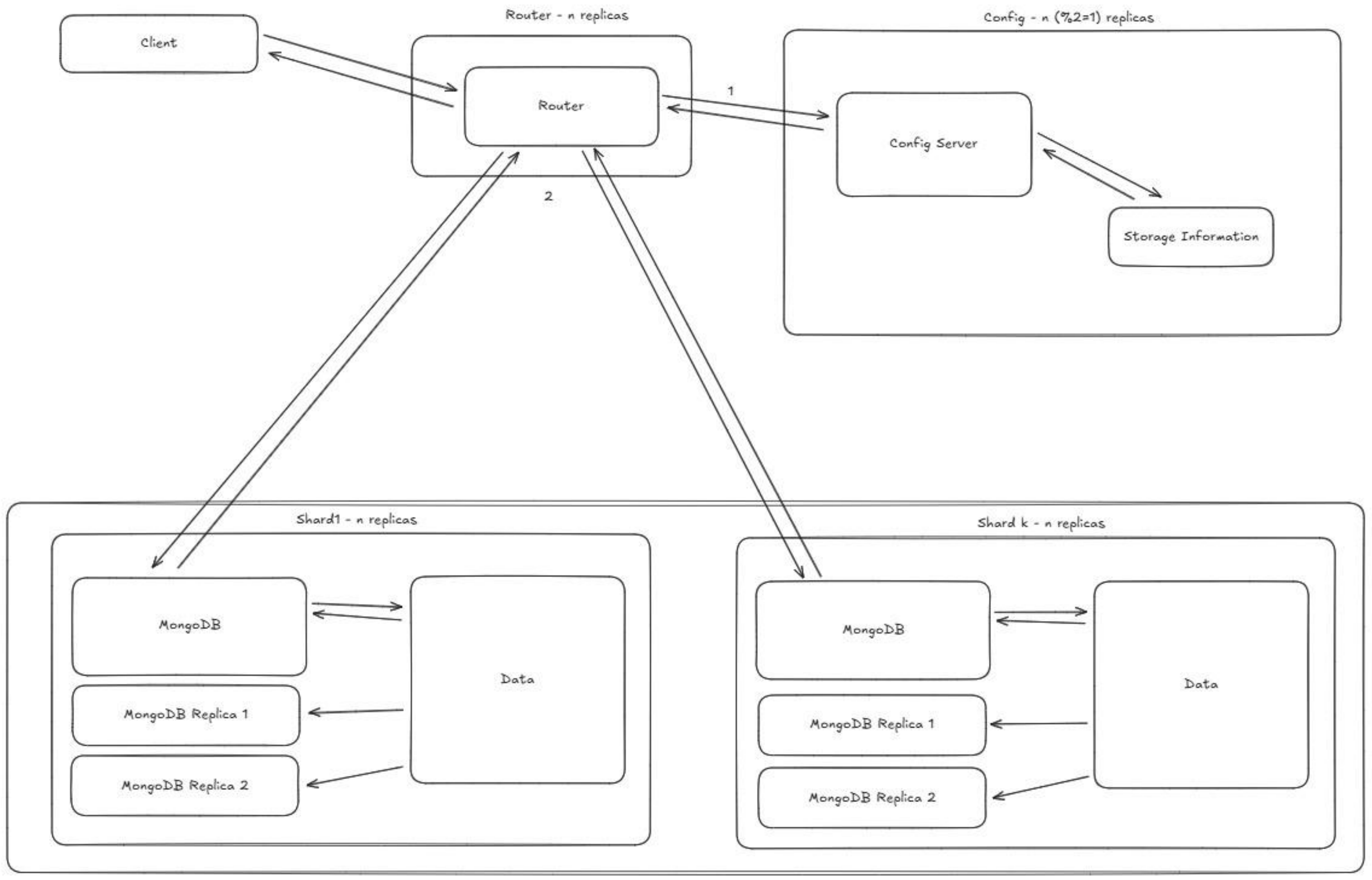
1	Client makes request
2	Mongos determines on which shards the data is available (A & B)
3	Mongos routes request to shard A and B
4	Mongos waits for response and merges the results
5	Merged and sorted result is sent to user

Initial Concept

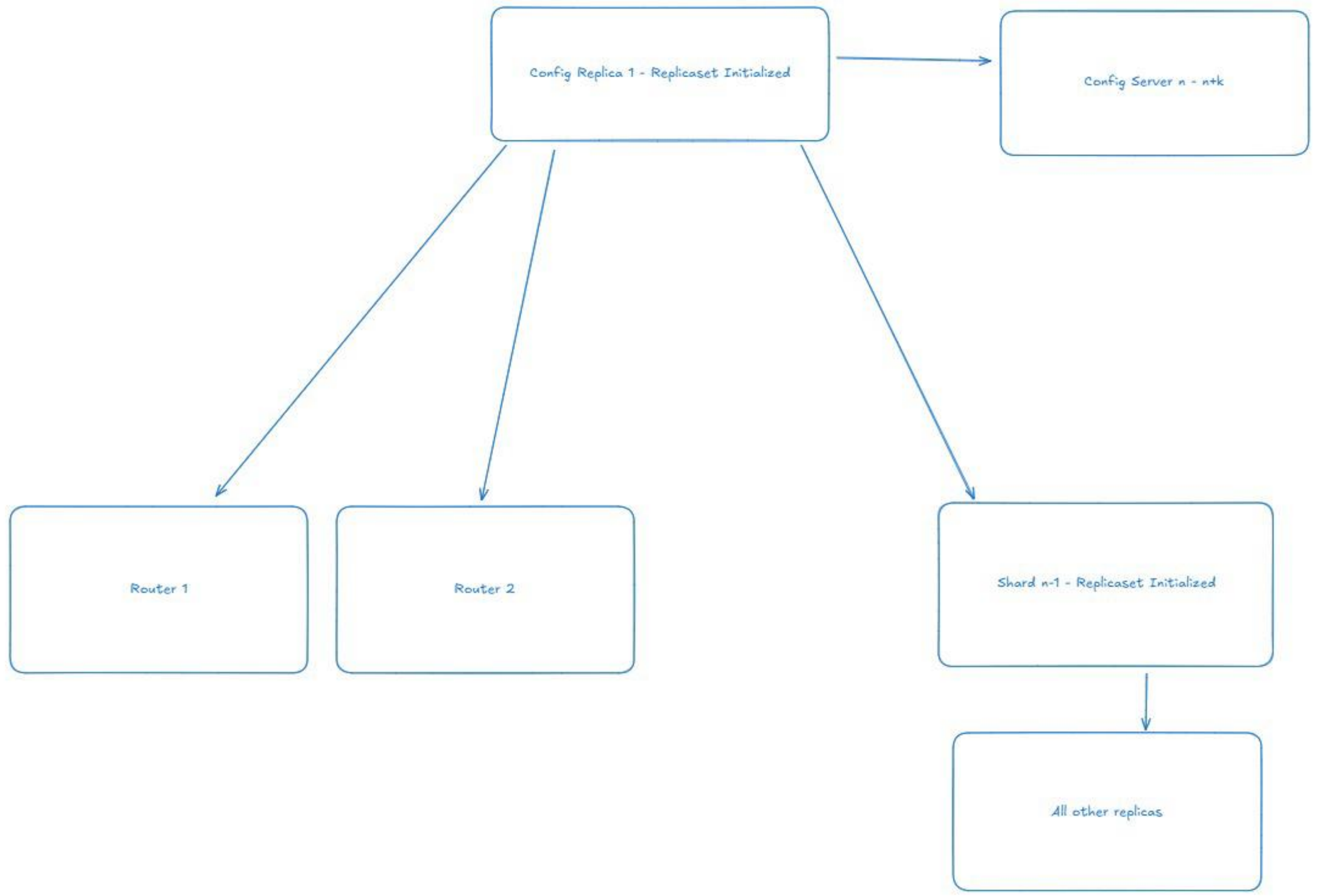
Initial Concept



Initial Concept



Initial Concept


Config Replica 1 - Replicaset Initialized
Config Server n - n+k
Router 1
Router 2
Shard n-1 - Replicaset Initialized
All other replicas

Implementation

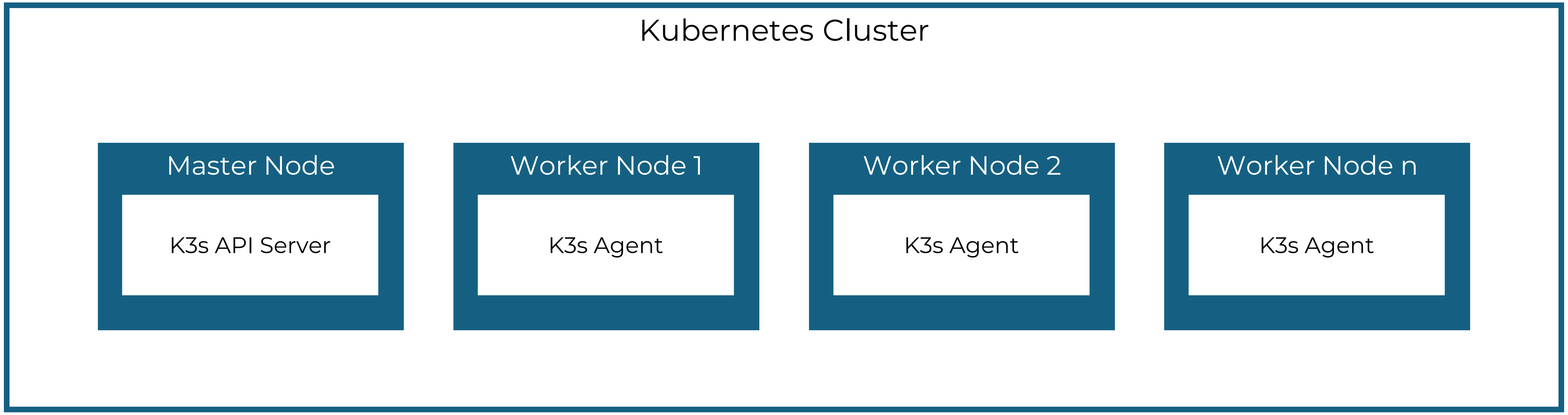
K3s Deployment

K3s Setup

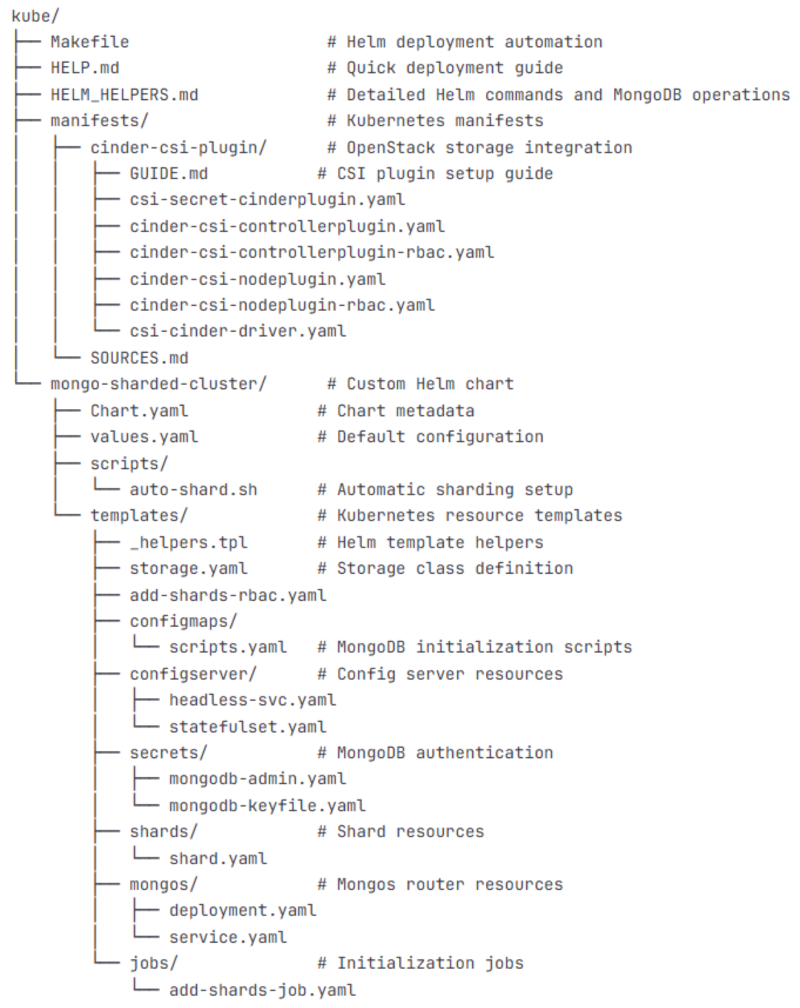
- 1 Master Node
- n Worker Nodes (2 used in this deployment)

Instances

- Rocky Linux 9.4
- m1.large (4 vCPUs, 8GB RAM)



OUR Helm Chart

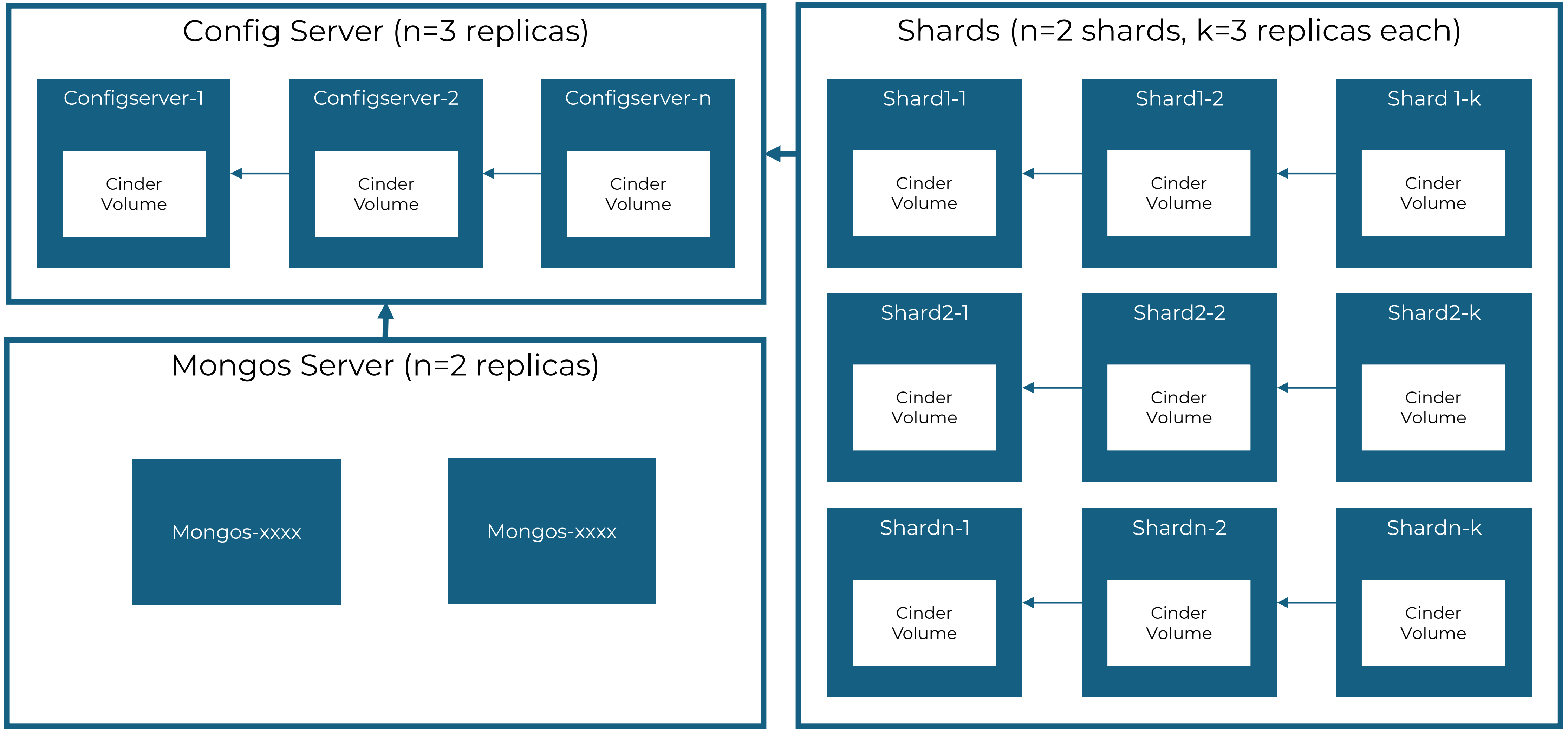


```
kube/
├── Makefile                    # Helm deployment automation
├── HELP.md                     # Quick deployment guide
├── HELM_HELPERS.md             # Detailed Helm commands and MongoDB operations
├── manifests/                  # Kubernetes manifests
│   ├── cinder-csi-plugin/      # OpenStack storage integration
│   │   ├── GUIDE.md           # CSI plugin setup guide
│   │   ├── csi-secret-cinderplugin.yaml
│   │   ├── cinder-csi-controllerplugin.yaml
│   │   ├── cinder-csi-controllerplugin-rbac.yaml
│   │   ├── cinder-csi-nodeplugin.yaml
│   │   ├── cinder-csi-nodeplugin-rbac.yaml
│   │   └── csi-cinder-driver.yaml
│   └── SOURCES.md
└── mongo-sharded-cluster/      # Custom Helm chart
    ├── Chart.yaml             # Chart metadata
    ├── values.yaml            # Default configuration
    ├── scripts/
    │   └── auto-shard.sh      # Automatic sharding setup
    └── templates/             # Kubernetes resource templates
        ├── _helpers.tpl       # Helm template helpers
        ├── storage.yaml       # Storage class definition
        ├── add-shards-rbac.yaml
        ├── configmaps/
        │   └── scripts.yaml   # MongoDB initialization scripts
        ├── configserver/      # Config server resources
        │   ├── headless-svc.yaml
        │   └── statefulset.yaml
        ├── secrets/           # MongoDB authentication
        │   ├── mongodb-admin.yaml
        │   └── mongodb-keyfile.yaml
        ├── shards/            # Shard resources
        │   └── shard.yaml
        ├── mongos/            # Mongos router resources
        │   ├── deployment.yaml
        │   └── service.yaml
        └── jobs/              # Initialization jobs
            └── add-shards-job.yaml
```

This custom helm chart includes

- Default configuration
- Chart metadata
- Helpers
- Fully customizable values including:
  - Pod and service metadata per function
  - Resource allocation per function
  - Volume allocation and settings per function
  - Cluster count
  - Replica counts per function
- Safe Roll-Overs
- Idempotence (can upgrade shard count, replicas…)
- Consistent storage
- Automatic sharding configuration and scripts
- Different headless services per function
- Authentication using secrets and private keys
- Usage of the Cinder CSI Plugin for volume mounts
- Fully dynamic setup and workflow scripts
- Healthiness probes
- Node port for reachability (load balancer alternative)
- and more…

MongoDB


Config Server (n=3 replicas)
Configserver-1
Cinder Volume
Configserver-2
Cinder Volume
Configserver-n
Cinder Volume
Shards (n=2 shards, k=3 replicas each)
Shard1-1
Cinder Volume
Shard1-2
Cinder Volume
Shard 1-k
Cinder Volume
Shard2-1
Cinder Volume
Shard2-2
Cinder Volume
Shard2-k
Cinder Volume
Shardn-1
Cinder Volume
Shardn-2
Cinder Volume
Shardn-k
Cinder Volume
Mongos Server (n=2 replicas)
Mongos-xxxx
Mongos-xxxx

MongoDB Deployment ‘Phases’

Phase 1:
Config Servers

1. Config stateful set starts
2. Init script runs on first instance
3. Replica set is formed
4. Master is elected
5. Admin user is created
6. Next config servers is created
7. Each pod is responsible for adding the new pod to the existing replica set via admin credentials

Phase 2:
Shards Init

1. Wait for first config server to run
2. Shard stateful sets start
3. Init script runs on the last replica
4. Last replica creates a replica set with all instances
5. Master is elected

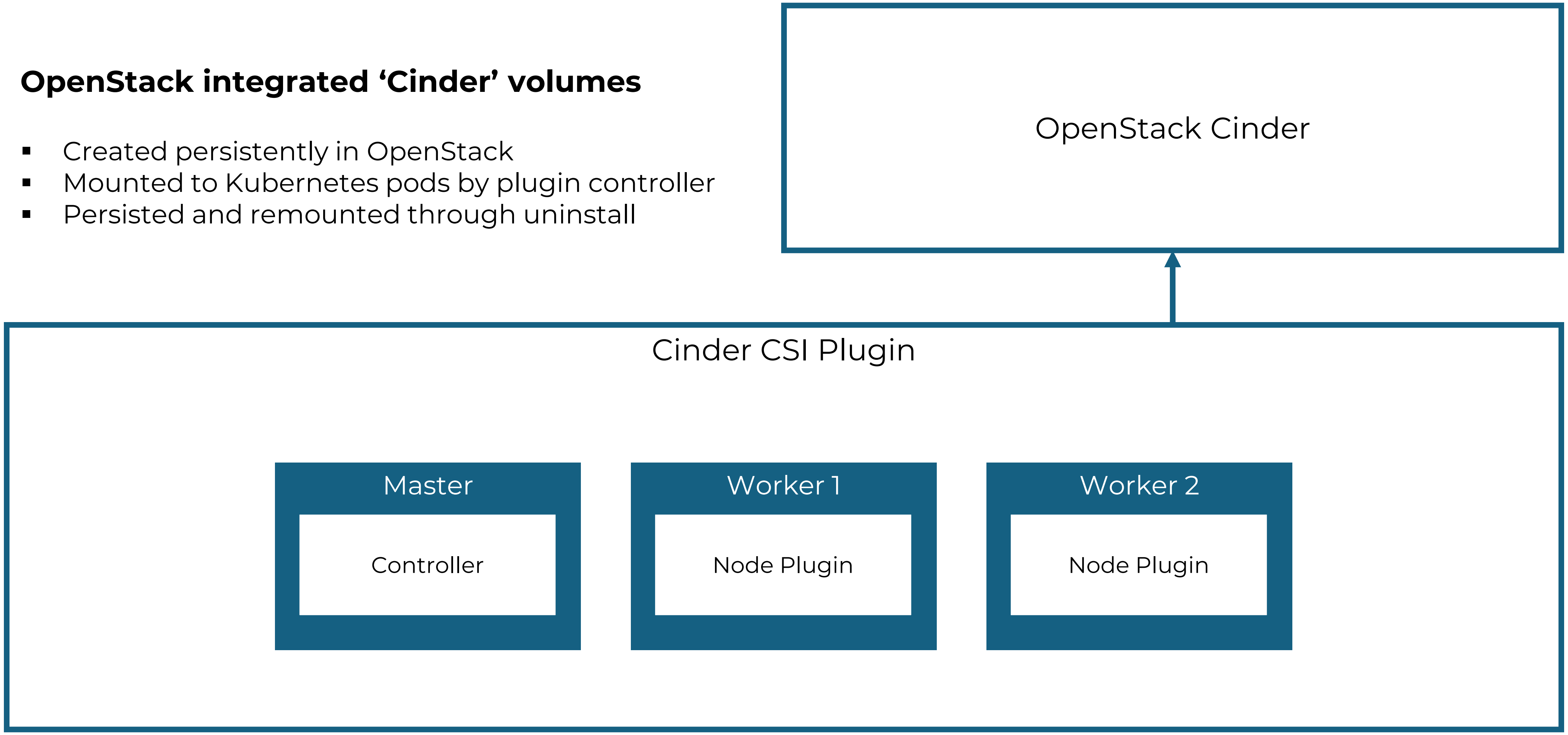
Phase 3:
Mongos & Sharding

1. Wait for first config server to run
2. Mongos deployment runs
3. The add-shards job is run
4. Each shard is added to the config servers via admin credentials
5. Optional: the auto-shard script is run

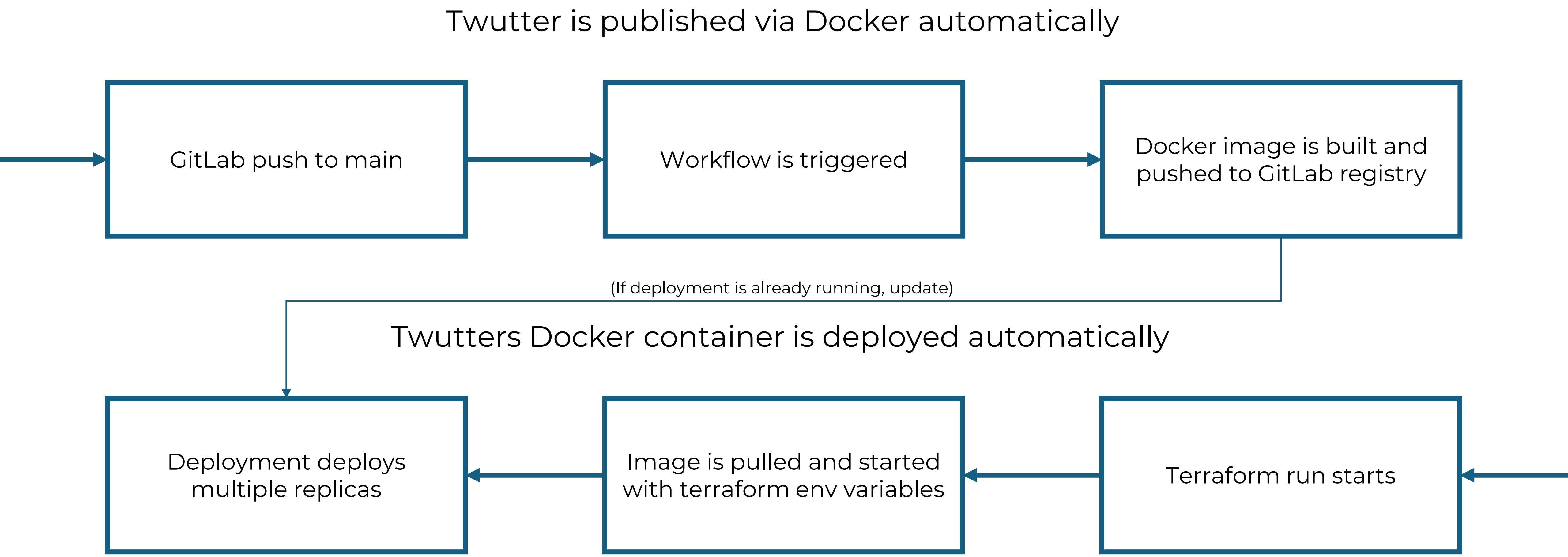
Storage Layer

OpenStack integrated ‘Cinder’ volumes

- Created persistently in OpenStack
- Mounted to Kubernetes pods by plugin controller
- Persisted and remounted through uninstall



Twutter Deployment


Twutter is published via Docker automatically
GitLab push to main
Workflow is triggered
Docker image is built and pushed to GitLab registry
(If deployment is already running, update)
Twutters Docker container is deployed automatically
Terraform run starts
Image is pulled and started with terraform env variables
Deployment deploys multiple replicas

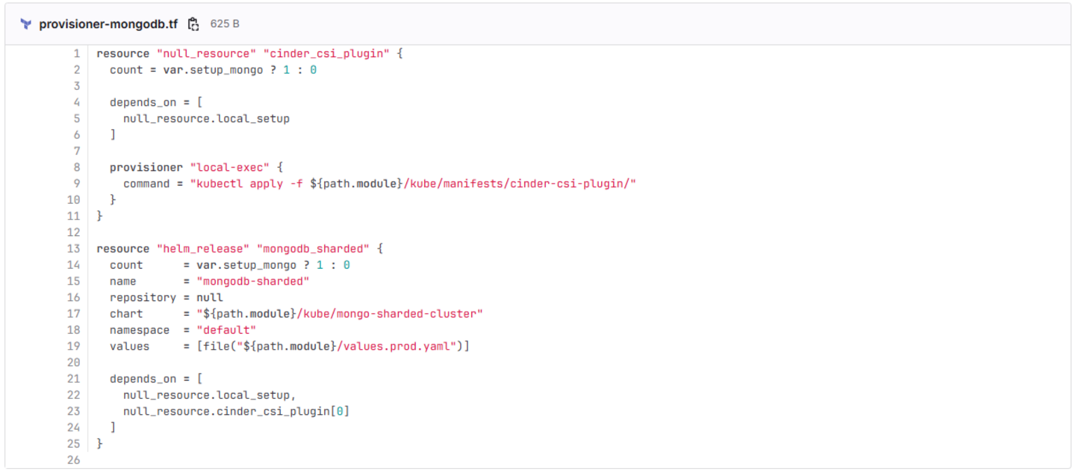
Terraform Additions

Automatic Helm and Plugin deployment

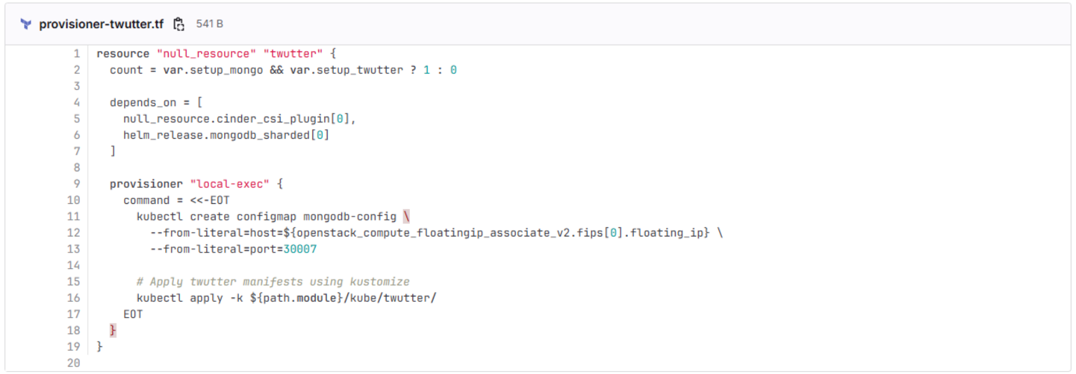
- Local exec starts the Cinder CSI Plugin
- Hashicorp helm provider starts the helm chart
- Local exec starts Twutter deployment

Automation can be disabled

- Variable ‘setup_mongo’ toggles mongo deployment
- Variable ‘setup_twutter’ toggles twutter deployment
- Makefile can be used to deploy manually


provisioner-mongodb.tf 625 B

```
resource "null_resource" "cinder_csi_plugin" {
  count = var.setup_mongo ? 1 : 0

  depends_on = [
    null_resource.local_setup
  ]

  provisioner "local-exec" {
    command = "kubectl apply -f ${path.module}/kube/manifests/cinder-csi-plugin/"
  }
}

resource "helm_release" "mongodb_sharded" {
  count      = var.setup_mongo ? 1 : 0
  name       = "mongodb-sharded"
  repository = null
  chart      = "${path.module}/kube/mongo-sharded-cluster"
  namespace  = "default"
  values     = [file("${path.module}/values.prod.yaml")]

  depends_on = [
    null_resource.local_setup,
    null_resource.cinder_csi_plugin[0]
  ]
}

```


provisioner-twutter.tf 541 B

```
resource "null_resource" "twutter" {
  count = var.setup_mongo && var.setup_twutter ? 1 : 0

  depends_on = [
    null_resource.cinder_csi_plugin[0],
    helm_release.mongodb_sharded[0]
  ]

  provisioner "local-exec" {
    command = <<-EOT
      kubectl create configmap mongodb-config \
        --from-literal=host=${openstack_compute_floatingip_associate_v2.fips[0].floating_ip} \
        --from-literal=port=30007

      # Apply twutter manifests using kustomize
      kubectl apply -k ${path.module}/kube/twutter/
    EOT
  }
}

```

(Scratched) Considerations

Create a GitLab pipeline

⇢ Databases handle important data so a pipelines automatically scaling this type of application would require *great* care, especially when removing shards. This process alone can take days on large datasets.
Also, we cannot create enough resources to enable staging, so testing would become very hard.

Lock connections by cluster IP

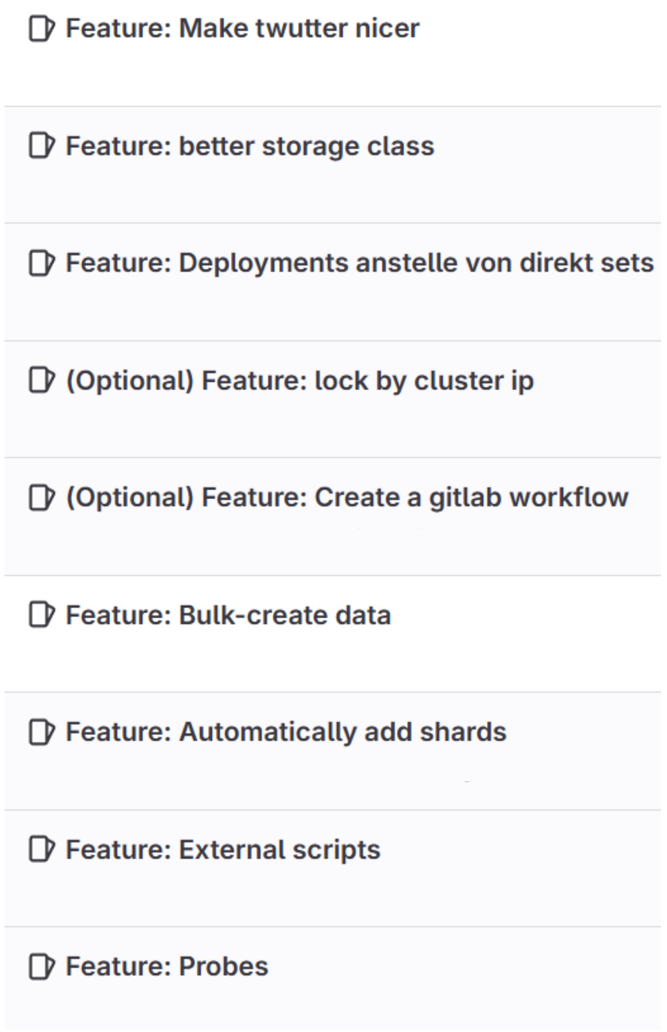
⇢ This feature did turn out to be practically useless as the functions each use their own headless service. Because of this external IPs cannot connect anyways.

Store secrets in a secret manager

⇢ This was scratched since it does not add value by itself. This could help with workflows or larger organizations. For us simply sharing the Kubernetes secrets directly made more sense.

Add a dashboard with statistics

⇢ Everything we want to show can be shown via the shell. Dashboard could be added but would only be interesting once the databases have been running for a couple of days.



Feature: Make twutter nicer
Feature: better storage class
Feature: Deployments anstelle von direkt sets
(Optional) Feature: lock by cluster ip
(Optional) Feature: Create a gitlab workflow
Feature: Bulk-create data
Feature: Automatically add shards
Feature: External scripts
Feature: Probes

Dashboard / Monitoring options

Paywall		Open-Source	
MongoDB Atlas	MongoDB Ops Manager	Percona Monitoring and Management (PMM)	Graphana and Prometheus
Easy to use and configure, real-time performance, data explorer functionality	Self-Hosted management, topology visualization, performance dashboards	Open-Source, detailed dashboards, Kubernetes-friendly, made for database analytics	Open-Source, Complete customizability, real-time metrics
Only runs on MongoDB Atlas hosted instances	Requires licensing, created for enterprises	Less customizability, advanced features require use of percona's MongoDB modification (image)	Requires a lot of additional setup, only some templates come out of the box, "jack of all trades" master of none

Tough nuts to crack

Tough nuts to crack

- **Admin user can only be created after the ReplicaSet is initialized**
  → The setup has to wait for the correct timing
- **Admin user can only be created on the primary node**
  → The primary must be identified and specifically targeted
- **Cinder volumes require a CSI plugin to work with Kubernetes**
  → Without the plugin, volume mounting is not possible
- **Local resources were no longer sufficient**
  → Transition to cloud infrastructure was necessary
- **Kubernetes config gets created on dev machine on apply**

  → CI/CD deployment needs to happen on the same machine that starts terraform

Our Application

Twutter

Why did we choose a social network?

- Social platforms involve rich interactions: **users, posts, comments, timelines** – and those create great opportunities to explore **real-world challenges** in distributed systems, like horizontal scaling and NoSQL data modeling.

⇢ It's the ideal playground to test bulk operations, and performance under high load.

General scenario

- In our MongoDB setup, the three key collections are:
  - users: stores profiles
  - posts: the main content users create
  - comments: user-generated replies tied to posts
- The project uses **Next.js** both for the frontend and backend logic (via API routes).
  On the frontend, we use React and built-in fetch methods to communicate with our APIs.
  Each API route directly talks to MongoDB, performing **CRUD** operations.

Why did we shard certain collections the way we did?

- **Posts**
  - Wir haben posts per Hash geshardet (z. B. über _id)
  - Alternative wäre Sharding nach author
  - Warum Hash?
  - → Feed-Abfragen rufen viele verschiedene Autoren ab
  - → Hash verteilt die Daten gleichmäßig
  - → Vermeidet Überlastung einzelner Shards
- **Comments**
  - Comments sind nach postId geshardet
  - Alle Kommentare zu einem Beitrag liegen auf demselben Shard
  - Gut für: Post-Ansichten und Kommentar-Zählungen
- **Users**
  - Auch users wurde per Hash geshardet
  - Vorteil: gleichmäßige Verteilung der Nutzer über alle Shards

Diese Konfiguration wurde bewusst für unser Testszenario gewählt. In produktiven Systemen kann je nach Zugriffsmuster ein anderer Shard-Key sinnvoll sein.

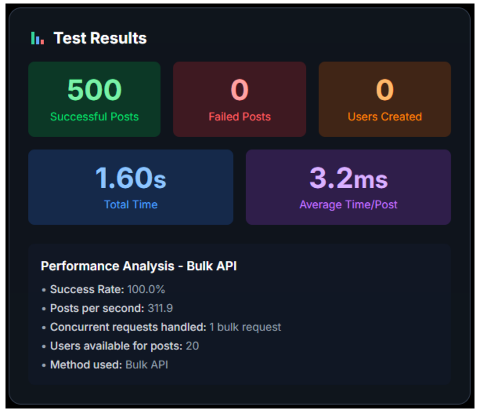
Live demo

Appendix

MongoDB Load Testing

Load Tests - Bulk

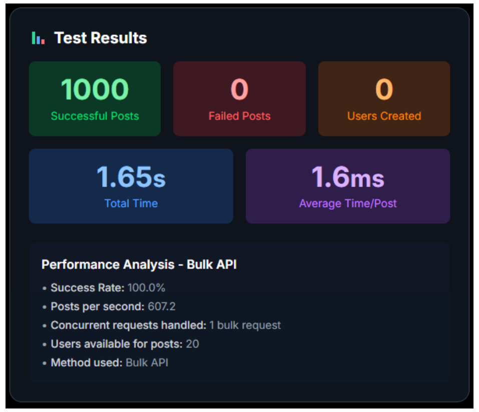
Local



Test Results

500 Successful Posts
0 Failed Posts
0 Users Created
1.60s Total Time
3.2ms Average Time/Post

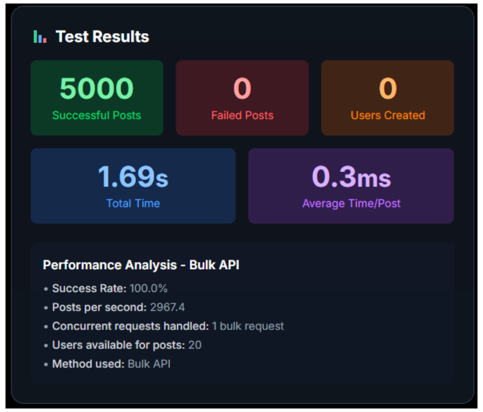
Performance Analysis - Bulk API
- **Success Rate:** 100.0%
- **Posts per second:** 311.9
- **Concurrent requests handled:** 1 bulk request
- **Users available for posts:** 20
- **Method used:** Bulk API



Test Results

1000 Successful Posts
0 Failed Posts
0 Users Created
1.65s Total Time
1.6ms Average Time/Post

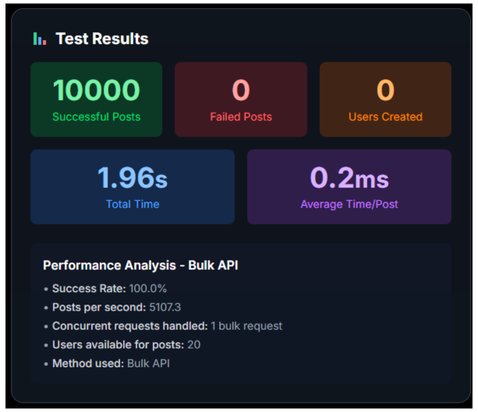
Performance Analysis - Bulk API
- **Success Rate:** 100.0%
- **Posts per second:** 607.2
- **Concurrent requests handled:** 1 bulk request
- **Users available for posts:** 20
- **Method used:** Bulk API



Test Results

5000 Successful Posts
0 Failed Posts
0 Users Created
1.69s Total Time
0.3ms Average Time/Post

Performance Analysis - Bulk API
- **Success Rate:** 100.0%
- **Posts per second:** 2967.4
- **Concurrent requests handled:** 1 bulk request
- **Users available for posts:** 20
- **Method used:** Bulk API

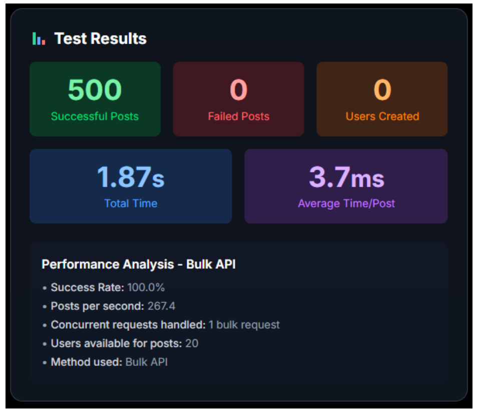


Test Results

10000 Successful Posts
0 Failed Posts
0 Users Created
1.96s Total Time
0.2ms Average Time/Post

Performance Analysis - Bulk API
- **Success Rate:** 100.0%
- **Posts per second:** 5107.3
- **Concurrent requests handled:** 1 bulk request
- **Users available for posts:** 20
- **Method used:** Bulk API

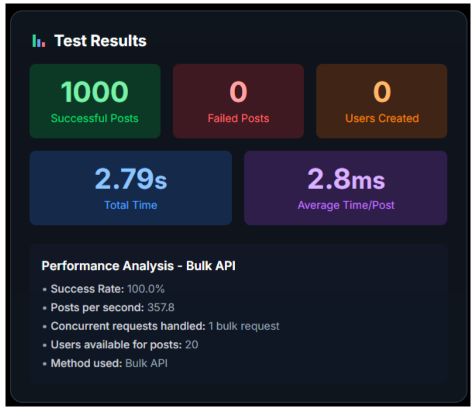
Sharded



Test Results

500 Successful Posts
0 Failed Posts
0 Users Created
1.87s Total Time
3.7ms Average Time/Post

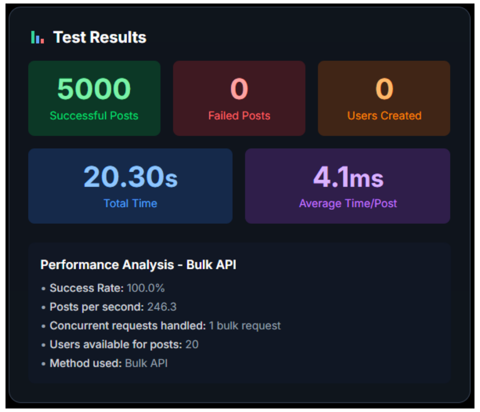
Performance Analysis - Bulk API
- **Success Rate:** 100.0%
- **Posts per second:** 267.4
- **Concurrent requests handled:** 1 bulk request
- **Users available for posts:** 20
- **Method used:** Bulk API



Test Results

1000 Successful Posts
0 Failed Posts
0 Users Created
2.79s Total Time
2.8ms Average Time/Post

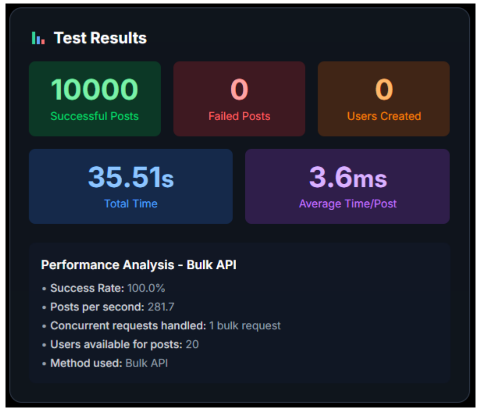
Performance Analysis - Bulk API
- **Success Rate:** 100.0%
- **Posts per second:** 357.8
- **Concurrent requests handled:** 1 bulk request
- **Users available for posts:** 20
- **Method used:** Bulk API



Test Results

5000 Successful Posts
0 Failed Posts
0 Users Created
20.30s Total Time
4.1ms Average Time/Post

Performance Analysis - Bulk API
- **Success Rate:** 100.0%
- **Posts per second:** 246.3
- **Concurrent requests handled:** 1 bulk request
- **Users available for posts:** 20
- **Method used:** Bulk API



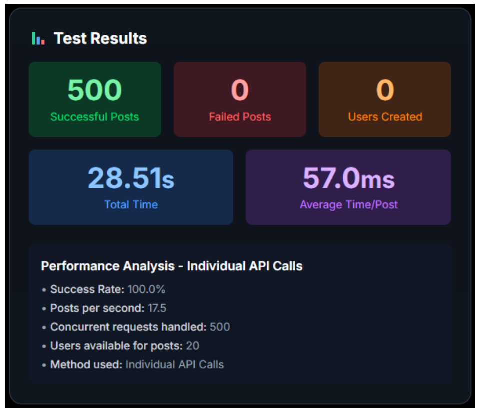
Test Results

10000 Successful Posts
0 Failed Posts
0 Users Created
35.51s Total Time
3.6ms Average Time/Post

Performance Analysis - Bulk API
- **Success Rate:** 100.0%
- **Posts per second:** 281.7
- **Concurrent requests handled:** 1 bulk request
- **Users available for posts:** 20
- **Method used:** Bulk API

Load Tests - Individual

Local

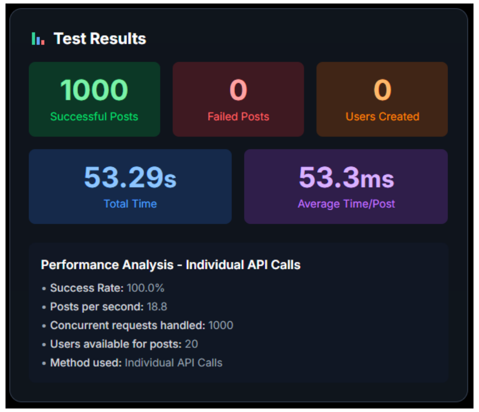


Test Results

500 Successful Posts | 0 Failed Posts | 0 Users Created

28.51s Total Time | 57.0ms Average Time/Post

Performance Analysis - Individual API Calls
- Success Rate: 100.0%
- Posts per second: 17.5
- Concurrent requests handled: 500
- Users available for posts: 20
- Method used: Individual API Calls

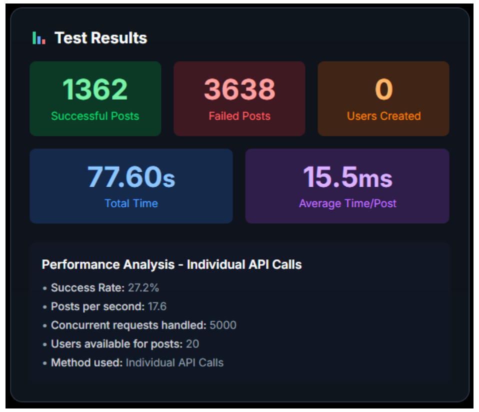


Test Results

1000 Successful Posts | 0 Failed Posts | 0 Users Created

53.29s Total Time | 53.3ms Average Time/Post

Performance Analysis - Individual API Calls
- Success Rate: 100.0%
- Posts per second: 18.8
- Concurrent requests handled: 1000
- Users available for posts: 20
- Method used: Individual API Calls

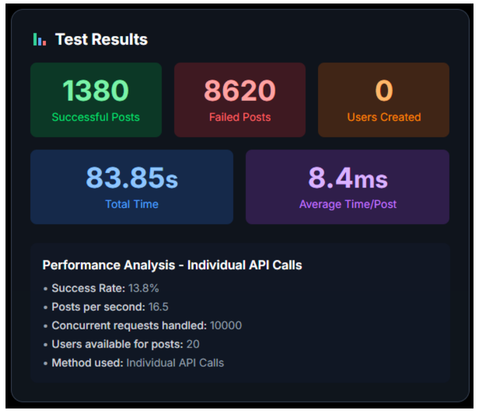


Test Results

1362 Successful Posts | 3638 Failed Posts | 0 Users Created

77.60s Total Time | 15.5ms Average Time/Post

Performance Analysis - Individual API Calls
- Success Rate: 27.2%
- Posts per second: 17.6
- Concurrent requests handled: 5000
- Users available for posts: 20
- Method used: Individual API Calls



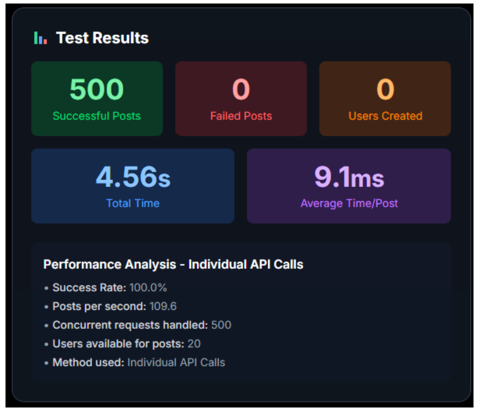
Test Results

1380 Successful Posts | 8620 Failed Posts | 0 Users Created

83.85s Total Time | 8.4ms Average Time/Post

Performance Analysis - Individual API Calls
- Success Rate: 13.8%
- Posts per second: 16.5
- Concurrent requests handled: 10000
- Users available for posts: 20
- Method used: Individual API Calls

Sharded

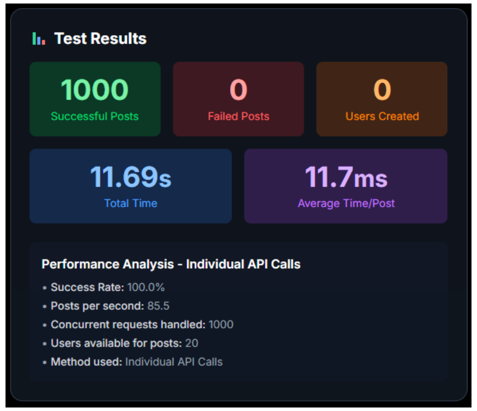


Test Results

500 Successful Posts | 0 Failed Posts | 0 Users Created

4.56s Total Time | 9.1ms Average Time/Post

Performance Analysis - Individual API Calls
- Success Rate: 100.0%
- Posts per second: 109.6
- Concurrent requests handled: 500
- Users available for posts: 20
- Method used: Individual API Calls

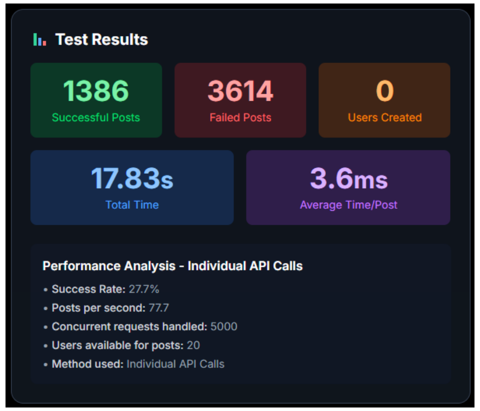


Test Results

1000 Successful Posts | 0 Failed Posts | 0 Users Created

11.69s Total Time | 11.7ms Average Time/Post

Performance Analysis - Individual API Calls
- Success Rate: 100.0%
- Posts per second: 85.5
- Concurrent requests handled: 1000
- Users available for posts: 20
- Method used: Individual API Calls

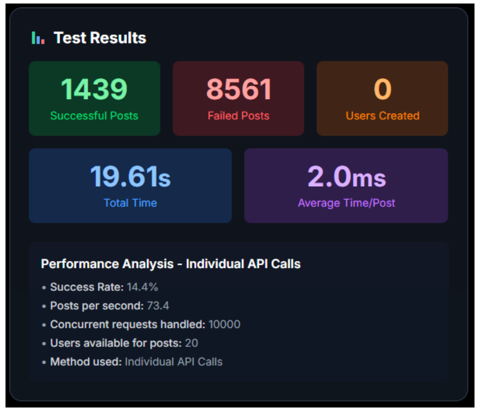


Test Results

1386 Successful Posts | 3614 Failed Posts | 0 Users Created

17.83s Total Time | 3.6ms Average Time/Post

Performance Analysis - Individual API Calls
- Success Rate: 27.7%
- Posts per second: 77.7
- Concurrent requests handled: 5000
- Users available for posts: 20
- Method used: Individual API Calls



Test Results

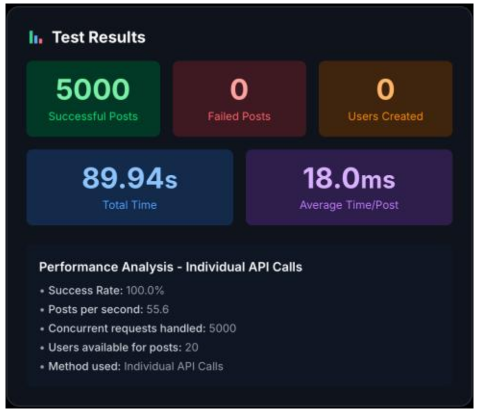
1439 Successful Posts | 8561 Failed Posts | 0 Users Created

19.61s Total Time | 2.0ms Average Time/Post

Performance Analysis - Individual API Calls
- Success Rate: 14.4%
- Posts per second: 73.4
- Concurrent requests handled: 10000
- Users available for posts: 20
- Method used: Individual API Calls

Load Tests Sharded - Individual

Backoff



Test Results

5000 Successful Posts

0 Failed Posts

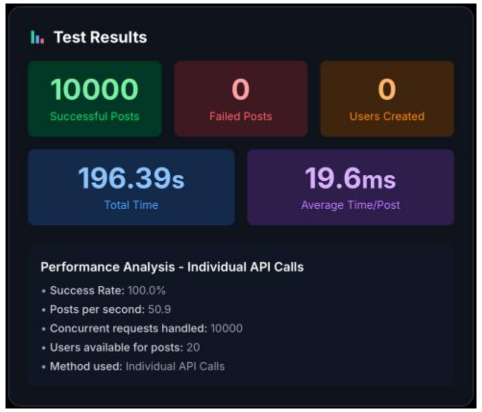
0 Users Created

89.94s Total Time

18.0ms Average Time/Post

Performance Analysis - Individual API Calls

- **Success Rate:** 100.0%
- **Posts per second:** 55.6
- **Concurrent requests handled:** 5000
- **Users available for posts:** 20
- **Method used:** Individual API Calls



Test Results

10000 Successful Posts

0 Failed Posts

0 Users Created

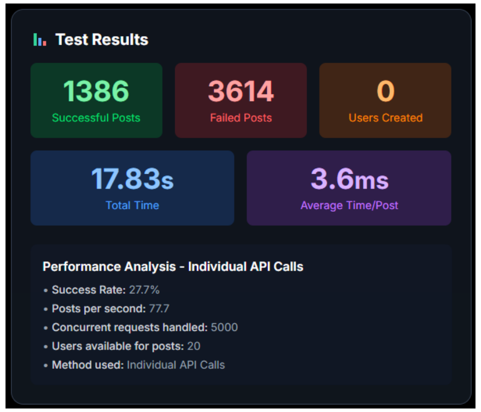
196.39s Total Time

19.6ms Average Time/Post

Performance Analysis - Individual API Calls

- **Success Rate:** 100.0%
- **Posts per second:** 50.9
- **Concurrent requests handled:** 10000
- **Users available for posts:** 20
- **Method used:** Individual API Calls

Single Attempt



Test Results

1386 Successful Posts

3614 Failed Posts

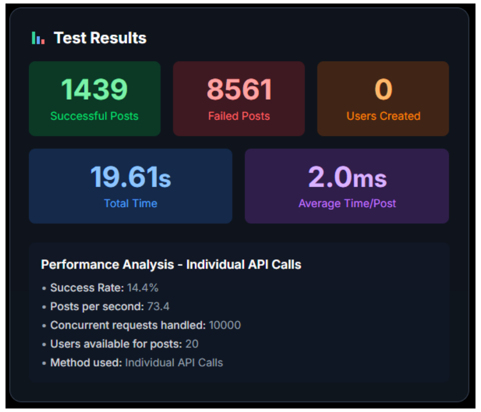
0 Users Created

17.83s Total Time

3.6ms Average Time/Post

Performance Analysis - Individual API Calls

- **Success Rate:** 27.7%
- **Posts per second:** 77.7
- **Concurrent requests handled:** 5000
- **Users available for posts:** 20
- **Method used:** Individual API Calls



Test Results

1439 Successful Posts

8561 Failed Posts

0 Users Created

19.61s Total Time

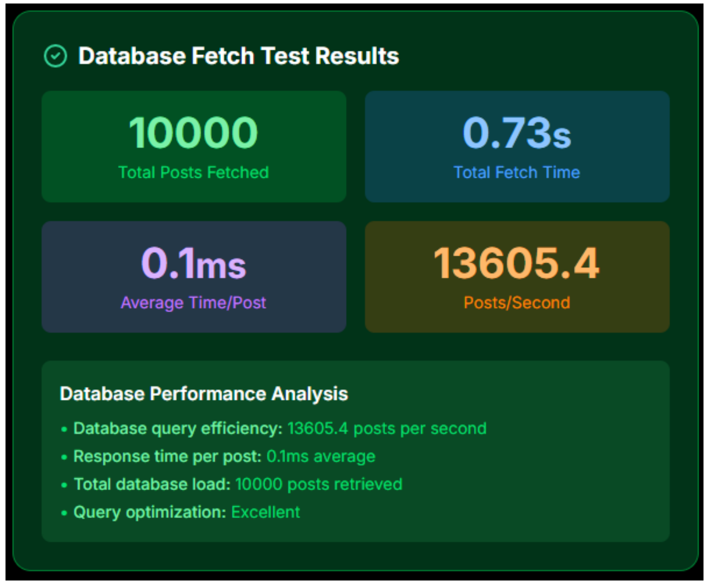
2.0ms Average Time/Post

Performance Analysis - Individual API Calls

- **Success Rate:** 14.4%
- **Posts per second:** 73.4
- **Concurrent requests handled:** 10000
- **Users available for posts:** 20
- **Method used:** Individual API Calls

Fetching

Sharded



Database Fetch Test Results

10000
Total Posts Fetched

0.73s
Total Fetch Time

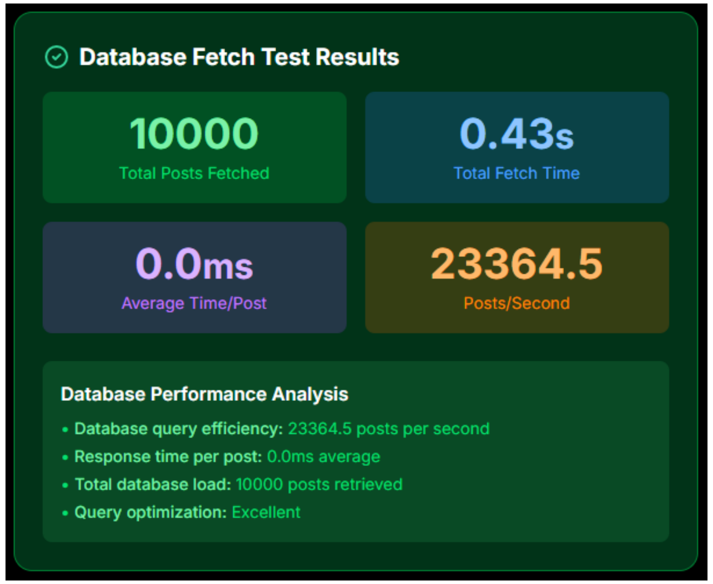
0.1ms
Average Time/Post

13605.4
Posts/Second

Database Performance Analysis

- **Database query efficiency:** 13605.4 posts per second
- **Response time per post:** 0.1ms average
- **Total database load:** 10000 posts retrieved
- **Query optimization:** Excellent

Local



Database Fetch Test Results

10000
Total Posts Fetched

0.43s
Total Fetch Time

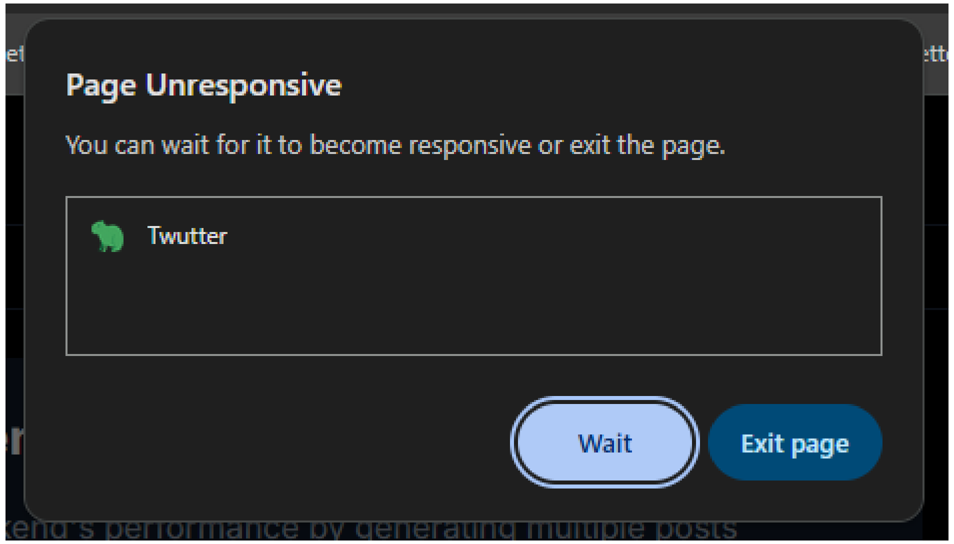
0.0ms
Average Time/Post

23364.5
Posts/Second

Database Performance Analysis

- **Database query efficiency:** 23364.5 posts per second
- **Response time per post:** 0.0ms average
- **Total database load:** 10000 posts retrieved
- **Query optimization:** Excellent

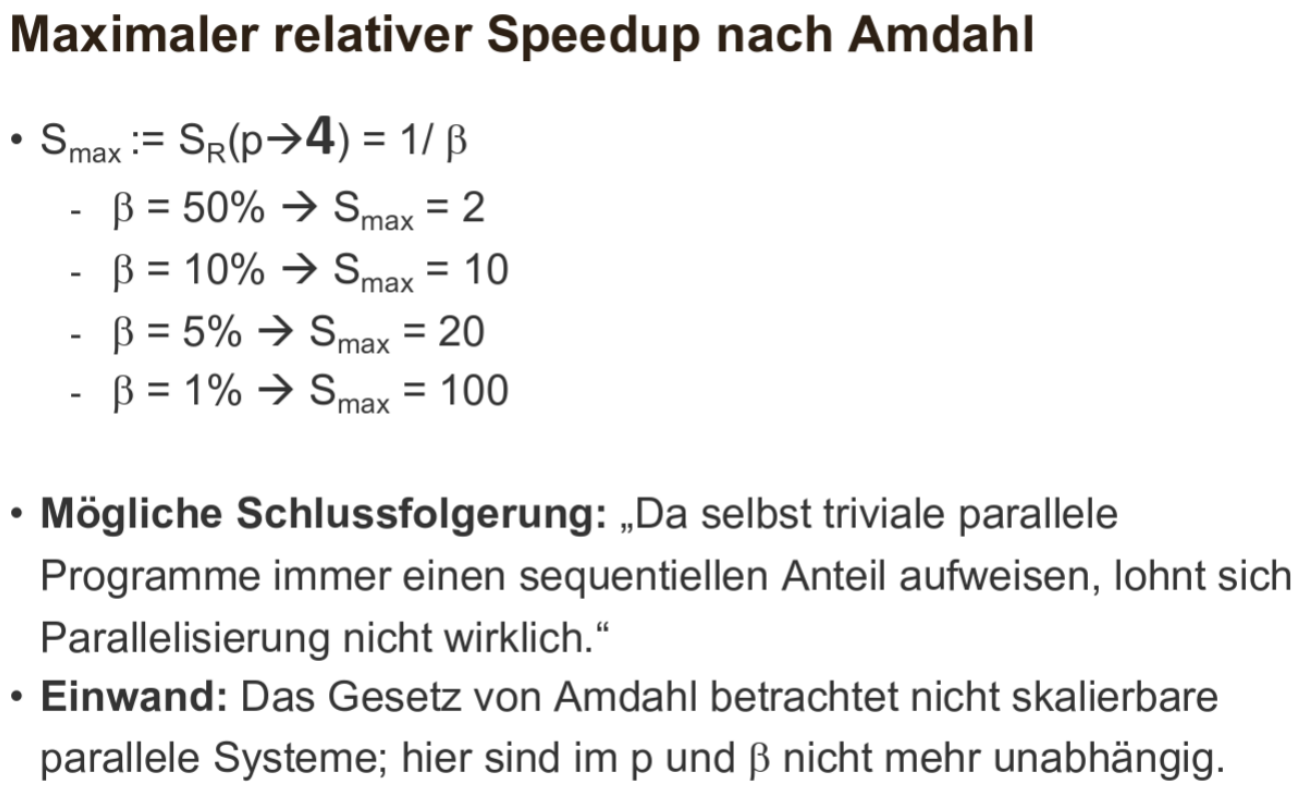
10000 requests local + individual


Page Unresponsive

You can wait for it to become responsive or exit the page.

Twutter

Amdahl



Maximaler relativer Speedup nach Amdahl

- $S_{max} := S_R(p \rightarrow \mathbf{4}) = 1/\beta$
  - $\beta = 50\% \rightarrow S_{max} = 2$
  - $\beta = 10\% \rightarrow S_{max} = 10$
  - $\beta = 5\% \rightarrow S_{max} = 20$
  - $\beta = 1\% \rightarrow S_{max} = 100$

- **Mögliche Schlussfolgerung:** „Da selbst triviale parallele Programme immer einen sequentiellen Anteil aufweisen, lohnt sich Parallelisierung nicht wirklich.“
- **Einwand:** Das Gesetz von Amdahl betrachtet nicht skalierbare parallele Systeme; hier sind im p und β nicht mehr unabhängig.

-> Das Ziel ist nicht die Laufzeitoptimierung bei kleinen Datenmengen,
sondern die Ermöglichung der Handhabung von großen Datenmengen.