ArmoniK: State DB & Deployment Basics

Formation Ops

ANEO

ArmoniK Team

December 3, 2025

Outline

Section 1

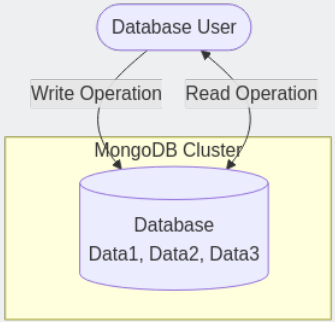
State DB

A Central Component: The Database

- The database is a central and essential component of ArmoniK.
- Stores all data required for orchestration:
  - Task and result metadata
  - Dependencies between tasks and results
- Constantly accessed by:
  - Control plane (on task submission)
  - Compute plane (on task acquisition)

The Naive Approach: Standalone Architecture

- Simplest form of database architecture
- Only one database instance handles all requests
- If the instance fails:
  - Data becomes inaccessible
  - Possible data loss in worst-case scenarios

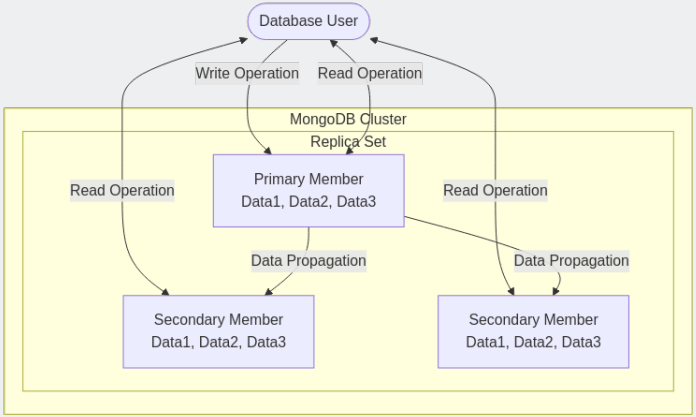


Limitations of Standalone Architecture

- Limited scalability:
  - Only possible through hardware upgrades
- No fault tolerance
- No optimization:
  - All operations go through a single instance

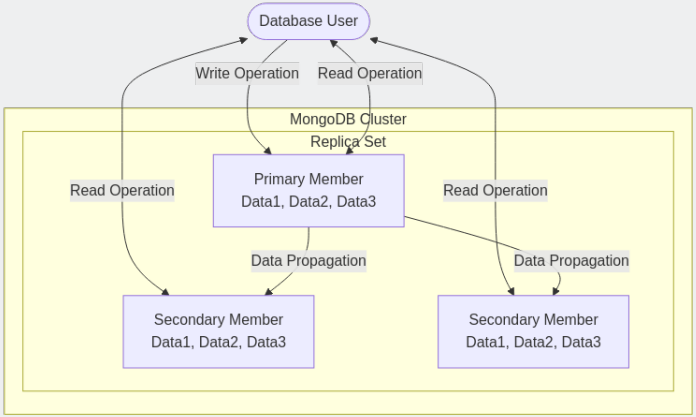
What is a Replica Set?

- Replica set: manages multiple MongoDB instances with identical data
- Each instance is a MongoDB daemon (‘mongod‘) – a full server
- Each instance is a member (or node) of the replica set



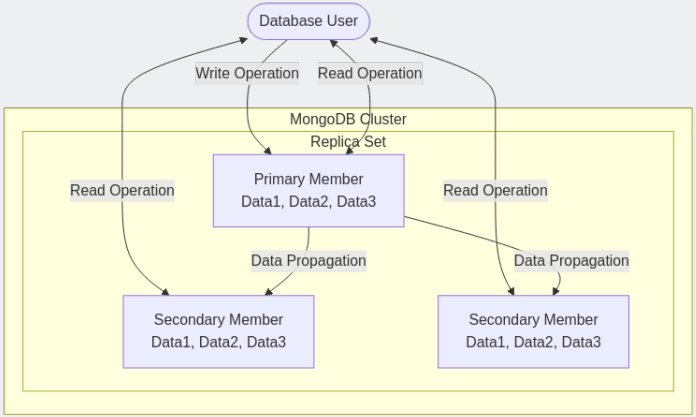
Types of Members

- **Primary Member**:
  - Only one per replica set
  - Handles all write operations
  - Can also handle read operations
- **Secondary Members**:
  - Handle only read operations
  - Offload read load from the primary



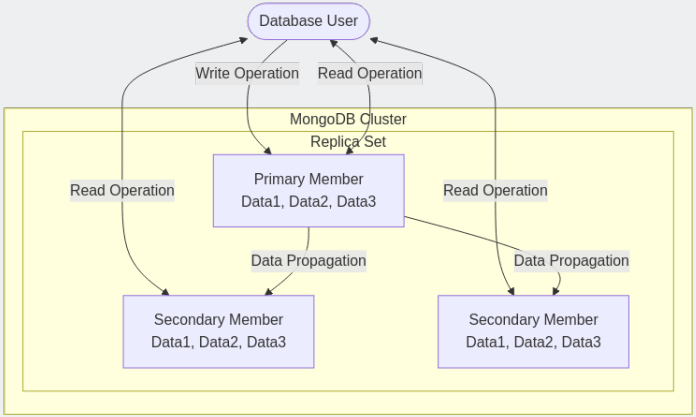
Scalability and Fault Tolerance

- Read operations can be distributed across secondary nodes
- Limited horizontal scalability
- Replica set can elect a new primary if the current one fails
- During the election:
  - Database is read-only



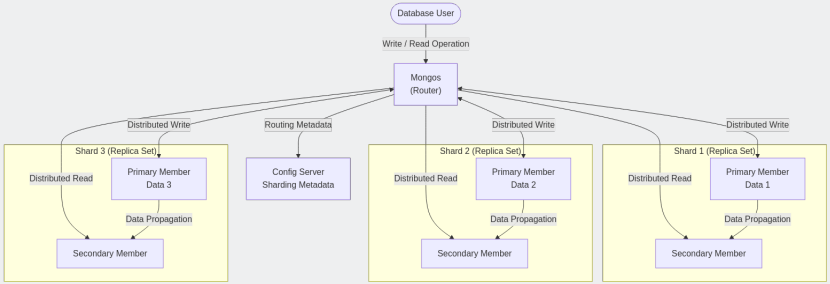
How Replica Sets Work Internally

- Each node is an independent ‘mongod‘ daemon
- The replica set
  - Coordinates nodes
  - Ensures data propagation from primary to secondaries
- Data redundancy improves fault tolerance



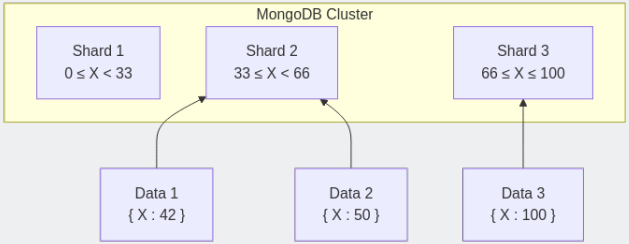
Sharded Architecture: A Step Further

- Splits the database into multiple independent **replica sets**
- Each shard is a full replica set with its own primary and secondaries
- Enables true distribution of data across multiple nodes
- Unlike replica set-only architectures, **write operations** are also distributed



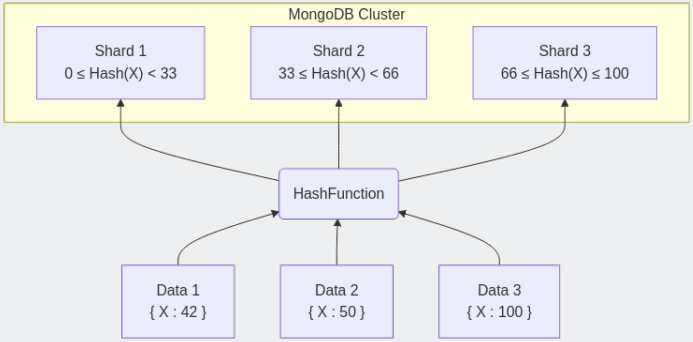
Shard Key and Data Distribution

- Data distribution is based on a **shard key**
- The shard key is an attribute of a MongoDB document
- Determines which shard stores a given document
- Functions like a traditional database index
- Also enables reverse lookup: identify the shard of a given document
- Each shard holds different data subsets



Shard Key and Data Distribution

- Data distribution is based on a **shard key**
- The shard key is an attribute of a MongoDB document
- Determines which shard stores a given document
- Functions like a traditional database index
- Also enables reverse lookup: identify the shard of a given document
- Each shard holds different data subsets



Bitnami MongoDB: Overview

- Pre-configured Helm chart for deploying MongoDB on Kubernetes
- Supports multiple modes:
  - Standalone
  - Replica Set
  - Sharded Cluster
- Production-ready features:
  - StatefulSets for persistent storage
  - Resource limits, liveness and readiness probes
  - TLS encryption and authentication

Bitnami MongoDB: Overview

- Pre-configured Helm chart for deploying MongoDB on Kubernetes
- Supports multiple modes:
  - Standalone
  - Replica Set
  - Sharded Cluster
- Production-ready features:
  - StatefulSets for persistent storage
  - Resource limits, liveness and readiness probes
  - TLS encryption and authentication

For the deployment of a sharded architecture we recommend to use MongoDB Atlas

Bitnami MongoDB: Overview

- Pre-configured Helm chart for deploying MongoDB on Kubernetes
- Supports multiple modes:
  - Standalone
  - Replica Set
  - Sharded Cluster
- Production-ready features:
  - StatefulSets for persistent storage
  - Resource limits, liveness and readiness probes
  - TLS encryption and authentication

For the deployment of a sharded architecture we recommend to use MongoDB Atlas

Since August 28 2025, the Bitnami charts are not longer open source, in the meantime we find an alternative, our reference deployment uses the Bitnami legacy images. For more information see this issue.

Bitnami MongoDB: Architecture and Features

- **Replica Set Architecture**
  - Automatic primary and secondary election
  - Built-in self-healing and failover
- **Persistence**
  - Uses PersistentVolumeClaims (PVCs)
  - Configurable storage classes and volume sizes
- **Security**
  - Password, keyfile, and TLS authentication options
- **Scalability**
  - Scales via Helm values or Kubernetes autoscaling
  - Sharded mode with `architecture=sharded`

TLS Implementation in ArmoniK for MongoDB

- **Purpose:** Secure communication between ArmoniK services and MongoDB using TLS.
- **Certificate Generation:**
  - Automated via Terraform.
  - Generates:
    - 4096-bit RSA private key.
    - Self-signed Certificate Authority (CA) certificate.
    - Certificate Signing Requests (CSRs) for each service.
    - Locally signed certificates for services.
- **Certificate Deployment:**
  - Certificates stored as local files.
  - Mounted into service containers (e.g., MongoDB) for use.

TLS Validation and Service Integration

- **Custom Validation Callback:**
  - Utilizes `CertificateValidator` class.
  - Validates:
    - Certificate chain integrity.
    - SSL policy errors.
    - Trustworthiness of certificates.
- **Integration with Services:**
  - Services like MongoDB, RabbitMQ, Redis use the same validation mechanism.
  - Ensures consistent and secure communication across all services.
- **Certificate Mounting:**
  - Certificates mounted into containers via the `mounts` parameter.
  - Includes CA certificate for validation purposes.

ArmoniK deployment

- Check TP PCs, validate that ArmoniK deploys correctly
- Visit MongoDB configmaps
- Connect to MongoDB from the Kubernetes cluster
- Launch some tasks using Pymonik, check DB changements

ArmoniK deployment

- You are ready to deploy ArmoniK:

```
$:~/ArmoniK/infrastructure/quick-deploy/localhost$ make
```

- A succesful deployment should show an ouput similar to this:

```
Outputs:

armonik = {
  "admin_app_url" = "http://xxxx-xxx-xxxx:5000/admin"
  "chaos_mesh_url" = null
  "control_plane_url" = "http://xxxx-xxx-xxxx:5001"
  "grafana_url" = "http://xxxx-xxx-xxxx:5000/grafana/"
  "seq_web_url" = "http://xxxx-xxx-xxxx:5000/seq/"
}

OUTPUT FILE: /home/ubuntu/ArmoniK/infrastructure/quick-deploy/localhost/generated/armonik-output.json

Run to point your ArmoniK CLI to this deployment:
------------------------------
export AKCONFIG=/home/ubuntu/ArmoniK/infrastructure/quick-deploy/localhost/generated/armonik-cli.yaml
```

Cluster Setup

- **Service Endpoints Overview**

  `admin_app_url` ArmoniK's web interface.
  `control_plane_url` Entry point for submitting tasks graphs.
  `grafana_url` Dashboard for real-time metrics and observability.
  `seq_web_url` Centralized log viewer for structured event traces.
  `chaos_mesh_url` (Optional) Fault injection platform — used only during dedicated Ops trainings to simulate failures and validate resilience.

- **Generated Files**

  `OUTPUT FILE` JSON file containing all deployment output variables — useful for automation or auditing.
  `AKCONFIG` CLI configuration file. Exporting it allows the `armonik` CLI to target this specific deployment without extra parameters.

Connect to MongoDB

- Navigate to the `Armonik/tools/mongodb` directory. There you will find an utility script to connect to MongoDB as user and another to connect as admin:

```
$:~/ArmoniK/tools$ ./access-mongo-from-kubernetes-as-user.sh
.
.
.
.
Connecting to:          mongodb+srv://XXXXX
Using MongoDB:          8.0.10
Using Mongosh:          2.5.3

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically
↪  (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

rs0 [primary] database>
(To exit, press Ctrl+C again or Ctrl+D or type .exit)
rs0 [primary] database>
```

- Once the connection is done you can use the commands `show collections`, to see the list of collections created by ArmoniK, you can inspect them, for example the `PartitionData` with `db.PartitionData.find()`.

Pymonik

- Clone the PymoniK repository in your home directory

```
$ git clone https://github.com/aneoconsulting/PymoniK.git
```

- In a second terminal export the AKCONFIG variable:

```
$ export AKCONFIG=/home/ubuntu/ArmoniK/infrastructure/quick-deploy/localhost/generated/armonik-cli.yaml
```

- Now you are ready to launch some tasks with PymoniK:

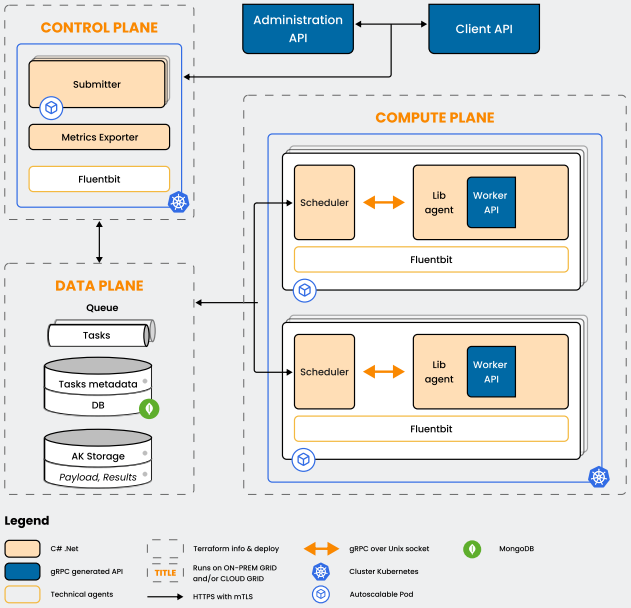
```
$:~$ cd $HOME/PymoniK/test_client/
$:~/PymoniK/test_client$ uv run estimate_pi.py

Session 71f1f54e-ef0d-4727-90b7-b5881bf55aa5 has been created
Submitting 100 parallel tasks for Pi estimation...
Waiting for all tasks to complete...
Estimated value of Pi: 3.142814
Session 71f1f54e-ef0d-4727-90b7-b5881bf55aa5 has been closed
```

- Inspect the data base collections.

Section 2

Deployment Basics

Architecture Internals



ArmoniK CLI Overview

The **ArmoniK CLI** is a tool that provides commands to monitor and manage computations in ArmoniK clusters.

The CLI enables users to:

- Manage results, sessions, and partitions
- Monitor task execution
- Query task results and metadata

Main advantages:

- Supports automation via scripts and scheduled jobs.
- Ideal for DevOps workflows, automated testing, and CI/CD pipelines.

More information

ArmoniK CLI official repository and Documentation

- ArmoniK CLI is already installed in the VMs of the formation, verify this. If not the case, install it with

```
pipx install armonik-cli
```

- Export the `AKCONFIG` environment variable into the current terminal:

```
export
↪  AKCONFIG=$HOME/ArmoniK/infrastructure/quick-deploy/localhost/generated/armonik-cli.yaml
```

- Get familiar with its interface, check the inline help `armonik --help`
- Use the CLI to list, create and delete some sessions.

HTC Mock is a test tool for ArmoniK. It is used to generate tasks customizable tasks that will be run by ArmoniK.

```
docker run --rm \
  -e HtcMock__NTasks=10000 \
  -e HtcMock__TotalCalculationTime=00:00:10 \
  -e HtcMock__DataSize=1 \
  -e HtcMock__MemorySize=1 \
  -e HtcMock__SubTasksLevels=1 0\
  -e HtcMock__Partition=$PARTITION_NAME \
  -e HtcMock__EnableFastCompute=true \
  -e HtcMock__TaskRpcException="" \
  -e GrpcClient__Endpoint=$GRPC_CLIENT_END_POINT \
  dockerhubaneo/armonik_core_htcmock_test_client:$CORE_TAG
```

Complete list of options

HtcMock options list in ArmoniK.Core repository.

```
docker run --rm \
-e BenchOptions__NTasks=400 \
-e BenchOptions__TaskDurationMs=10 \
-e BenchOptions__Partition=bench \
-e GrpcClient__Endpoint=$GRPC_CLIENT_END_POINT \
dockerhubaneo/armonik_core_bench_test_client:$CORE_TAG
```

Complete list of options

Bench options list in ArmoniK.Core repository.

ArmoniK C# Samples

- **HelloWorld**: The name saids it all ...
- **DynamicSubmission**: A subtasking example using the native APIs, fork-join approach with a shared payload, enconding in the payload.
- **LinearSubtasking**: A subtasking example using the native APIs, a task submits a new task and delegates the result responsability to the subtask.
- **MultipleResults**: A task that produces mutliple results.
- **SubTasking:** A subtasking example using the native APIs, fork-join apprach, enconding in the task options.

Hello World

- Add a `helloworld` partition to the `parameters.tfvar` file. You might copy the default partition, rename it and change the image and tag to:

```
worker = [
  {
    image = "dockerhubaneo/armonik_demo_helloworld_worker"
    tag   = "v2.21.0-SNAPSHOT.78.sha.80a7a9d"
  }
  ...
```

- Run the client, a helper script, `runHello.sh`, has been provided in the directory `Armonik/tools`. It escentially executes:

```
TAG=v2.21.0-SNAPSHOT.78.sha.80a7a9d
docker run --rm \
  dockerhubaneo/armonik_demo_helloworld_client:$TAG\
  --endpoint <control plane ip> --partition helloworld
```

Hello World: edited

- Clone the repository ArmoniK.Samples
- Edit the Client code so instead "Hello" as input string it takes "Hasta la vista".
- Edit the Worker code so it appends "Baby_" instead of "World_" to the result message.
- Build both docker images, you could name them `hello_terminator_client` and `hello_terminator_worker` and tag them as `v1`.
- Edit the helloworld partition providing your new worker image an redeploy.
- Run your newly created client.

helloworld: correction

- Edit the client code in `ArmoniK.Samples/csharp/native/Client/Program.cs` and build your own docker image:

```
# From the root of the repository ArmoniK.Samples
docker build -f csharp/native/HelloWorld/Client/Dockerfile -t
↪  hello_terminator_client:v1 csharp/native/
```

helloworld: correction

- Edit worker code in `ArmoniK.Samples/csharp/native/Client/HelloWorldWorkers.cs` and build your own docker image:

```
# From the root of the repository ArmoniK.Samples
docker build -f csharp/native/HelloWorld/Worker/Dockerfile -t
↪  hello_terminator_worker:v1 csharp/native/
```

Hello World: correction

- Edit the `helloworld` partition in the `parameters.tfvar` file so it uses your new worker:

```
worker = [
  {
    image = "hello_terminator_worker"
    tag = "v1"
  }
  ...
```

- Redeploy

helloworld: correction

Running your client with the provided help script should yield:

```
ubuntu@zou:~$ ./runHello.sh

sessionId: 22c0c486-5d3d-4681-9cdd-877ebeb26726
Task id ab1aecd2-bcb9-475f-999a-d07bf0dbf8cd
resultId: a8a24095-2ed2-48a7-bde0-02f3b1f28255, data: Hasta la vista Baby_
↪  a8a24095-2ed2-48a7-bde0-02f3b1f28255
```

Section 3

ArmoniK Load Balancer

Overview

The ArmoniK Load Balancer enables the capability to use multiple ArmoniK clusters from a single endpoint. It is implemented according to the AEP 4.

- A cluster is selected among the configured ones using a round-robin scheme.
- All tasks of a session will be executed on the selected cluster.
- If a cluster becomes unreachable, its sessions are also unreachable, but new sessions will go to remaining available clusters.

Usage

- Redirect your client to the load balancer endpoint.
- No further client modification is needed.
- The load balancer does not listen on TLS; use an nginx ingress for TLS capabilities.

Usage

- Redirect your client to the load balancer endpoint.
- No further client modification is needed.
- The load balancer does not listen on TLS; use an nginx ingress for TLS capabilities.

Note on the admin GUI

The Admin GUI is not part of the load balancer and should be added in front using the same nginx ingress.

Usage

- Redirect your client to the load balancer endpoint.
- No further client modification is needed.
- The load balancer does not listen on TLS; use an nginx ingress for TLS capabilities.

Note on the admin GUI

The Admin GUI is not part of the load balancer and should be added in front using the same nginx ingress.

Authentication

The Load Balancer does not perform authorization/authentication.

- Authentication is handled by an nginx to be placed in front of it.
- It propagates user identity to upstream clusters by forwarding authentication headers.

Configuration

The load balancer can be configured using:

- A configuration file
- Environment variables

See the documentation for more details.

- Make sure the `protobuf-compiler` is installed in the VM:

```
$ sudo apt-get install protobuf-compiler
```

- Clone the `ArmoniK.Infra.Plugins` repository in your home directory

```
$ git clone https://github.com/aneoconsulting/ArmoniK.Infra.Plugins
```

- Now you are ready to build the load balancer (cargo and rust is already installed on all the VMs):

```
$:~$ cd $HOME/ArmoniK.Infra.Plugins/load-balancer
$:~/ArmoniK.Infra.Plugins/load-balancer$ cargo build --release
```

- The previous command should install the load balancer at `ArmoniK.Infra.Plugins/target/release`

- Navigate to the directory `ArmoniK.Infra.Plugins/target/release` and create a file `tp.yaml` with the following content:

```
clusters:
  remote:
    endpoint: https://35.240.66.231:5001/
  local:
    endpoint: https://<PUT HERE YOUR CONTROL PLANE URL>/
refresh_delay: 60
```

  The configured remote is an ArmoniK cluster already deployed for you for this practical lab.

- You can run the load balancer with:

```
$:~/ArmoniK.Infra.Plugins/target/release$ ./load-balancer -c lb.yaml
```

- In a separate terminal launch some tasks using `PymoniK` or one of the Samples covered earlier. Verify the load balancer alternates the session creation between the two clusters.

122 avenue du general Leclerc
91200 Boulogne-Billancourt
www.aneo.eu