MongoDB Backup & Recovery Field Guide

Tim Vaillancourt
Percona

PERCONA

`whoami`

```
{
        name: “tim”,
        lastname: “vaillancourt”,
        employer: “percona”,
        techs: [
                “mongodb”,
                “mysql”,
                “cassandra”,
                “redis”,
                “rabbitmq”,
                “solr”,
                “mesos”
                “kafka”,
                “couch*”,
                “python”,
                “golang”
        ]
}
```

PERCONA

Agenda

- Backups
  - Logical
  - Binary
  - Architecture
  - Security
- Consistent Backups
- Percona-Lab/mongodb_consistent_backup
- Recovery
  - Logical
  - Binary

MongoDB Backups

“An admin is only worth the backups they keep” ~ Unknown

Logical Backups

- Storage-engine agnostic
- Logical representation
  - Is not machine bytes
  - Is not architecture specific / dependent
- Simple backup procedure
- Low disk consumption
- Common logical backup tools
  - **mongodump** - MongoDB
  - **mysqldump** - MySQL
  - ...

Logical Backups: mongodump

- ‘mongodump’ tool from mongo-tools project
- Supports
  - Multi-threaded dumping in 3.2+
  - Optional inline gzip compression of data
  - Optional dumping of oplog for single-node consistency
  - Replica set awareness via --readPreference= flag
    - *Ie: primary, primaryPreferred, secondary, secondaryPreferred, nearest*

PERCONA

Logical Backups: mongodump

- Process
  - Tool issues *.find()* query with *$snapshot* cursor
  - Stores BSON data in a file per collection
  - Stores BSON oplog data in *“oplog.bson”*

Logical Backups: mongodump

- Useful for...
  - upgrades of very old systems, eg: 2.6 -> 3.4 upgrade
  - protection from binary-level/storage-engine corruption
  - export/import to different CPU architecture

Logical Backups: mongodump

- Limitations
  - Index metadata only in backup
    - *Indexes are rebuilt entirely, in serial!!*
    - *Often indexing process takes longer than restoring the data!*
    - *Expect hours or days of restore time*
  - Not Sharding aware
    - *Sharded backups are not Point-in-Time consistent*
    - *Must use mongos, very inefficient*

Logical Backups: mongodump

- Limitations
  - Fetch from storage-engine, serialization, networking, etc is very inefficient
  - Oplogs fetched in batch at end / oplog must be as long as the backup run-time
  - Wire Protocol Compression *(added in 3.4+, default in 3.6)* not supported yet: https://jira.mongodb.org/browse/TOOLS-1668 (Please vote/watch Issue!)

Binary Backups

- Backup data is the storage-engine files
- Options
  - Cold Backup
  - LVM Snapshot
  - Hot Backup
    - Percona Server for MongoDB (FREE!)
    - MongoDB Enterprise Hot Backup ($$$)
    - NOTE: MMAPv1 not supported

Binary Backups

- Benefits
  - Faster backup time
    - *Backups can move at the speed of the disk/host*
  - Faster time to restore
    - *Indexes are backed up entirely*
    - *No time spent rebuilding indexes (!!!)*

Binary Backups

- Limitations
  - Increased backup storage requirements
  - CPU Architecture limitations *(64-bit vs 32-bit)*
  - Cascading corruption
  - Batteries not included
    - Not Sharding aware
    - Not Replica Set aware
    - Oplog is not captured separately

Binary Backups

- Process
  - Cold Backup
    - Stop a mongod SECONDARY, copy/archive dbPath
  - LVM Snapshot
    - Optionally call ‘db.fsyncLock()’ *(not required in 3.2+ with Journaling)*
    - Create LVM snapshot of the dbPath
    - Copy/Archive dbPath
    - Remove LVM snapshot *(as quickly as possible!)*
    - NOTE: LVM snapshots can cause up to 30%* write latency impact to disk (due to COW)

Binary Backups

- Process
  - Hot Backup *(PSMDB or MongoDB Enterprise)*
    - Pay $$$ for MongoDB Enterprise or download PSMDB for free(!)
    - ```
      db.adminCommand({
          createBackup: 1,
          backupDir: "/data/mongodb/backup"
      })
      ```
    - Copy/archive the output path
    - Delete the backup output path
    - NOTE: RocksDB-based createBackup creates filesystem hardlinks whenever possible!
    - NOTE: Delete RocksDB backupDir as soon as possible to reduce bloom filter overhead!


      ```

Backup Security

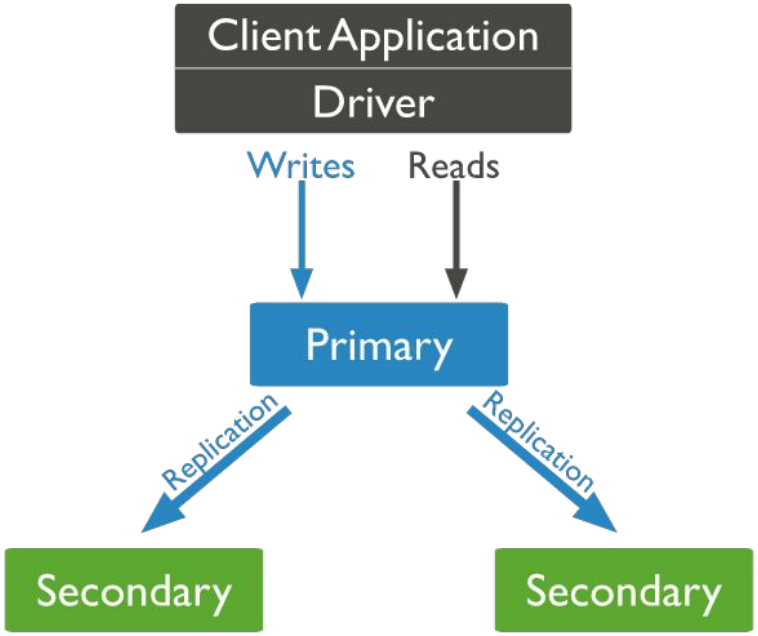
- Authorization
  - *“backup”* built-in role
  - Client Source IP restriction *(new in 3.6!)*
  - x509 Client Certificate vs Passwords
- Transport
  - Use SSL/TLS with MongoDB
    - *“preferSSL” mode allows secure and plain*
  - Upload backups with secure connection

Backup Security

- Storage
  - Who can access the backups?
  - File System access
  - Encryption

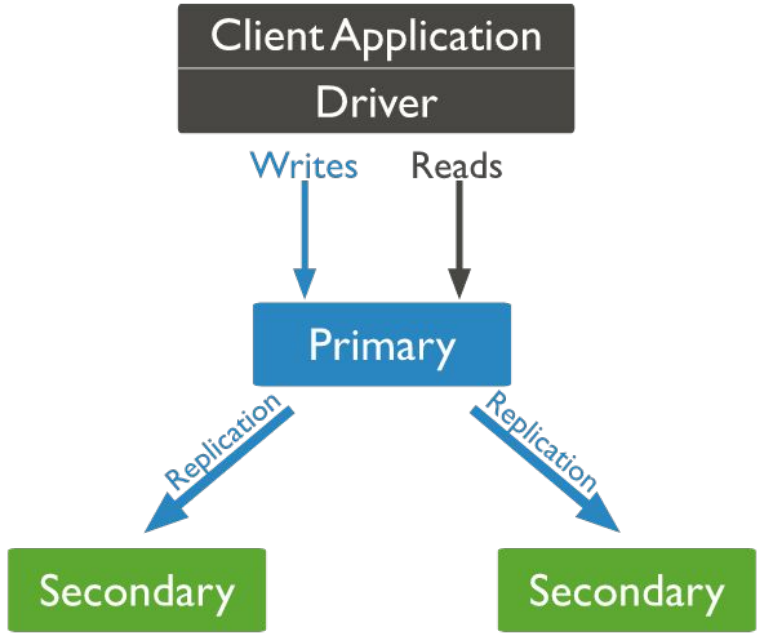
Backup Architecture

- Risks
  - Dynamic nature of Replica Set
  - Impact of backup on live nodes
- Example: Cheap Disaster-Recovery
  - Place a *'hidden: true'* SECONDARY in another location
  - Optionally use cloud object store *(AWS S3, Google GS, etc)*



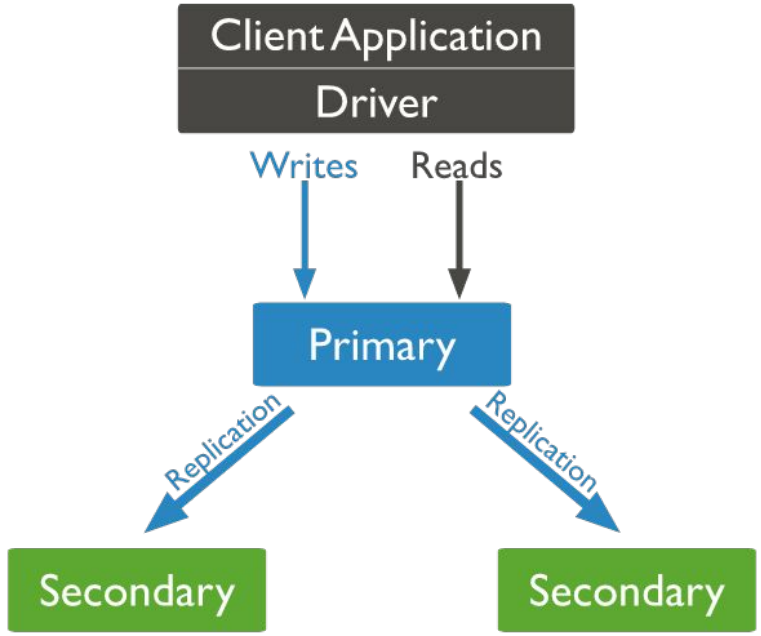
Backup Architecture

- Example: Replica Set Tags
  - “tags” allow fine-grained server selection with key/value pairs
  - Use key/value pair to fence various application workflows
  - Example:
    - *{ “role”: “backup” } == Backup Node*
    - *{ “role”: “application” } == App Node*



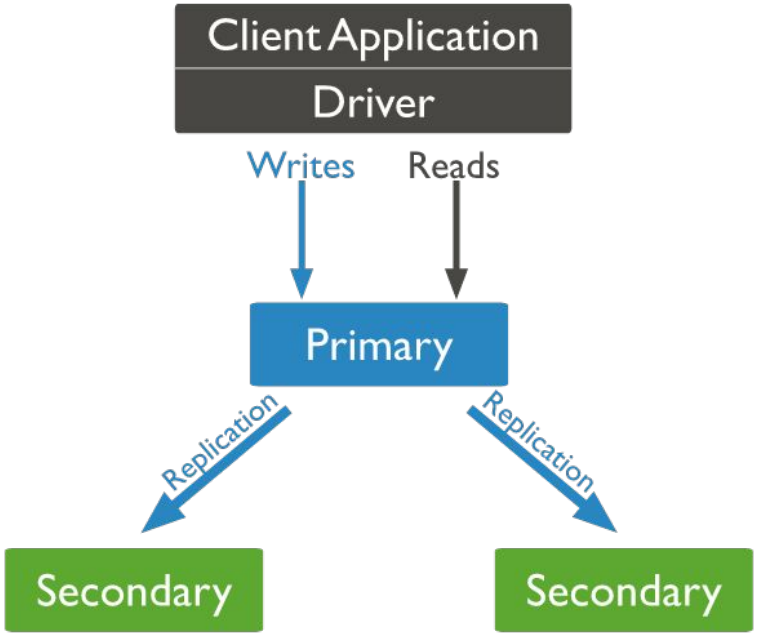
Backup Style

- Full
  - Take a full copy of the database data
  - Simple
  - Very costly to store frequent backups
- Incremental
  - Logical *(oplog)*
  - Binary
- Hybrid
  - Full every N days
  - Incremental every 12-24 hours



Backup Tips

- Store backups in structured paths
  - *<name>-YYYYMMDD_HHMM*
- Also backup the
  - MongoDB configuration
  - (When secure) MongoDB Internal Authentication key

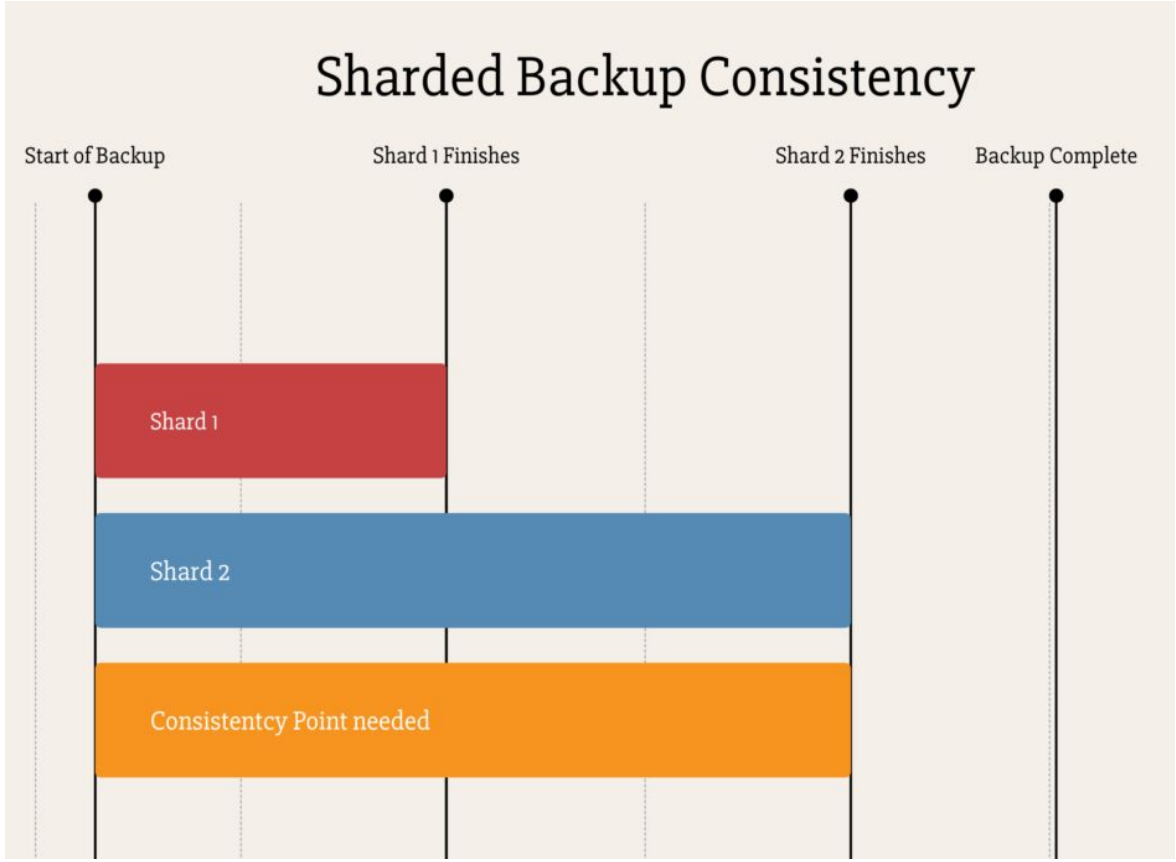


PERCONA

Consistent Backups

Shards and Consistency

- Problem
  - Backup tools are replset consistent but NOT sharding consistent
  - Shards can complete backup at different times
  - ..but a consistent backup means all shards are at the same point in time!



PERCONA

Shards and Consistency

- Solution: a process that
  - *Understands the sharded cluster topology*
  - *Is able to watch the oplogs of all shards*
  - *Handles the backups of all shards*
  - *Ensures the oplogs are aligned for all shards for consistency*
  - *...*

Percona-Lab/mongodb_consistent_backup

MCB: History

- Python project by Percona-Lab for consistent backups
- URL: https://github.com/Percona-Lab/mongodb_consistent_backup
- Best-effort support, not a “Percona Product”
- Created to solve limitations in MongoDB backup tools:
  - Replica Set and Sharded Cluster awareness
  - Cluster-wide Point-in-time consistency
  - In-line Oplog backup *(vs post-backup)*
  - Notifications of success / failure

MCB: Features

- Features
  - Auto replica-set and sharded-cluster discovery
  - Cluster-consistent live oplog backup
  - Remote Upload *(AWS S3, Google Cloud Storage and Rsync)*
  - Archiving *(Tar or ZBackup deduplication and optional AES-at-rest)*
  - CentOS/RHEL7 RPMs and Docker-based releases *(.deb soon!)*
  - Single Python PEX binary
  - Multithreaded / Concurrent and auto-scales to available CPUs

MCB: Features

- Low-Impact
  - Uses Secondary nodes only
  - Considers *(Scoring)*
    - *Replication Lag*
    - *Replication Priority*
    - *Replication Health / State*
    - *Hidden-Secondary State (preferred by tool)*
    - *Fails if chosen Secondary becomes Primary (on purpose)*

MCB: Future

- Future
  - End of life of Python-based tool
  - Productization
  - Incremental Backups
  - Binary-level Backups *(Hot Backup, Cold Backup, LVM, Cloud-based, etc)*
  - Restore Functionality
  - Instrumentation / Metrics

Restore

“An admin is only worth the backups they keep” ~ Unknown

Logical-Backup Restore

- *mongorestore* required with *mongodump* backups
- By default all databases/collections are restored
- Add *--drop* flag for full restore
  - Mongorestore uses inserts that can collide on Primary Key
  - Replaces the data during restore

Logical-Backup Restore

- Parallel collections possible
  - Default 4
  - --numParallelCollections=N
- --oplogReplay flag for consistency
  - Always use if possible!

Binary-Backup Restore

- Process
  - Stop MongoDB process
  - Move the current dbPath to a safe place, if required
  - Move the backup binary files to the correct *storage.dbPath*
  - Ensure the MongoDB configuration file matches the backup node
  - Start MongoDB
    - *MongoDB replication will sync using oplog or (full sync if it can't)*

Sharded Cluster Restore Process

1. Stop all Mongos routers
2. Restore Config Servers from backup
   a. Update *"config.shards"* if shard hostnames changed
3. Restore all Shards from backup
4. Start a Mongos and Stop the Balancer
   a. *sh.stopBalancer()*
5. Test the cluster and data

Sharded Cluster Restore Process

6. Start all Mongos processes and Start the Balancer
   a. *sh.startBalancer()*

Questions?

DATABASE PERFORMANCE
MATTERS