Before moving a MongoDB database to **production**, it's critical to perform **database optimization** to ensure **performance, scalability, and reliability**. Here's a comprehensive checklist grouped by areas to guide you:

## **✅ 1. Data Modeling & Schema Design**

* 🔸 **Embed vs Reference**: Use embedding for frequently accessed sub-documents (1:1, 1:few); reference for large or independent relationships (1:many, many:many).
* 🔸 **Avoid Unbounded Growth**: Don’t let arrays or documents grow without bounds (e.g., capped logs or archive strategies).
* 🔸 **Document Size**: Ensure documents stay <16MB.
* 🔸 **Field Types**: Use appropriate BSON types; avoid large unused fields.

## **✅ 2. Indexing Strategy**

* 🔸 **Use Covered Queries**: Index fields used in filter, sort, and project.
* 🔸 **Avoid Over-indexing**: Unused indexes slow down writes and consume space.
* 🔸 **Compound Indexes**: Prefer compound indexes over multiple single indexes.
* 🔸 **Hashed Indexes**: For sharded \_id or load balancing.
* 🔸 **TTL Indexes**: Automatically remove old data for time-series or logs.
* 🔸 **Use explain()**: Validate query plans and avoid COLLSCAN.

## **✅ 3. Query Optimization**

* 🔸 **Profile Queries**: Use db.setProfilingLevel(1) temporarily to identify slow queries.
* 🔸 **Use Indexes Efficiently**: Ensure queries use indexes with .explain("executionStats").
* 🔸 **Pagination**: Avoid .skip() on large offsets; use range-based pagination instead.
* 🔸 **Projection**: Only fetch needed fields (.find({}, {field1: 1, \_id: 0})).

## **✅ 4. Sharding (If Needed)**

* 🔸 **Shard Key Design**:
  + High cardinality.
  + Even distribution.
  + Avoid monotonically increasing keys (e.g., timestamps).
* 🔸 **Pre-split Chunks**: If migrating large data.
* 🔸 **Balancing**: Ensure balancer is enabled before production.
* 🔸 **Zone Sharding**: If deploying across regions or availability zones.

## **✅ 5. Replication & High Availability**

* 🔸 **Replica Set Setup**: Minimum 3 nodes (Primary + 2 Secondaries or Arbiter).
* 🔸 **Priorities**: Configure member priorities correctly.
* 🔸 **Hidden Members**: For backups or reporting without impacting primary.
* 🔸 **Write Concerns**: Use {w: majority} for durability.

## **✅ 6. Security Hardening**

* 🔸 **Authentication**: Enable role-based access control (RBAC).
* 🔸 **Encryption**:
  + TLS/SSL for transport.
  + Encryption at rest (MongoDB Enterprise or OS-level encryption).
* 🔸 **Firewall**: Only expose necessary ports; bind IPs securely.
* 🔸 **Password Rotation**: Enforce password policies and rotation.
* 🔸 **Audit Logging**: Enable for sensitive operations.

## **✅ 7. Backup & Restore Strategy**

* 🔸 **Enable Regular Backups**:
  + Mongodump + Mongorestore
  + File-level snapshots
  + Ops Manager / PBM for Community Edition
* 🔸 **Test Restores**: Validate backup integrity.
* 🔸 **Point-in-Time Recovery (PITR)**: If needed, enable Oplog-based incremental backups.

## **✅ 8. Monitoring & Alerts**

* 🔸 **Enable Monitoring**:
  + MongoDB Cloud Monitoring
  + Prometheus + Grafana with mongodb\_exporter
* 🔸 **Set Alerts**:
  + Replication lag
  + Disk usage
  + Query performance
  + Node availability

## **✅ 9. Resource Optimization**

* 🔸 **Storage Engine**: Use **WiredTiger** (default); tune compression (snappy or zstd).
* 🔸 **Cache Size**: Tune wiredTiger.cacheSizeGB (typically 50% of RAM).
* 🔸 **Disk IOPS**: Use SSDs, especially for journal and data files.
* 🔸 **NUMA Settings**: Disable NUMA for better performance.

## **✅ 10. Miscellaneous**

* 🔸 **Connection Pooling**: Tune app-side connection pools.
* 🔸 **Connection String**: Use retryWrites, retryReads, and appropriate timeouts.
* 🔸 **Log Rotation**: Set up logrotate or use logRotate.
* 🔸 **Environment Parity**: Match staging and production settings for consistent performance.

### **📄 Optional: Run a Pre-Production Checklist Script**

You can script validations like:

db.getCollectionInfos(); // validate indexes  
db.serverStatus(); // check memory, locks, queues  
db.currentOp(); // running operations