UNIVERSIDADE ESTADUAL DO TOCANTINS – UNITINS
Curso de Sistemas de Informação – Disciplina: Sistemas Distribuídos

Período Letivo: 2025/2 – 7º Período – Câmpus Palmas

Alunos: João Victor Póvoa França e Hemilly Christinne Silva Pereira

Documentação Semana 3 - Arquitetura de Replicação de Set no MongoDB com Docker

Relatório Semana 3

PALMAS-TO

2025

Introdução

Este relatório descreve, de forma didática e técnica, as atividades realizadas na terceira semana do projeto para implementação de um ambiente distribuído de banco de dados MongoDB com replicação (replica set), integração com aplicação Node.js/Express (utilizando TypeScript), configuração de logging das operações e orquestração via Docker. O objetivo principal desta etapa foi validar a replicação de dados entre múltiplas instâncias MongoDB, assegurar que as operações de leitura e escrita fossem registradas, e preparar a infraestrutura para os próximos passos que contemplam migração para ambiente em nuvem e provisionamento em Proxmox.

Escopo e escopo de trabalho realizado

As atividades cobriram os seguintes pontos: (a) provisionamento de três instâncias MongoDB para compor um replica set; (b) configuração do projeto Node.js/Express com TypeScript para conexão com replica set; (c) implementação de middleware de logging (Winston + Morgan) e persistência de logs em coleção específica no MongoDB; (d) teste de replicação usando objeto de teste; (e) configuração parcial de ambiente via Docker Compose; (f) documentação e preparação para a migração ao cloud e ao Proxmox.

Ambiente e pré-configurações

O ambiente de desenvolvimento utilizou máquina local para desenvolvimento do código Node.js com o express. As pré-configurações incluem: Mongoose, Winston e Morgan. No repositório do projeto existem prints (screenshots) que evidenciam estados intermediários do banco e registros — estes arquivos foram incorporados ao material do projeto conforme instruído.

Configuração das instâncias MongoDB

Foram configuradas três instâncias MongoDB com as portas 27017, 27018 e 27019. A configuração seguiu o padrão abaixo:

- Cada instância inicializada com --replSet rs0 --bind_ip_all para habilitar o modo replica set.

- Volumes persistentes foram mapeados para diretórios locais para garantir a durabilidade dos dados entre reinícios de container.

- Após inicialização dos três containers, o mongosh foi utilizado para executar rs.initiate(...) e, quando necessário, rs.reconfig(...) com os hosts corretos. O estado do conjunto foi verificado com rs.status() até que um membro fosse eleito como

PRIMARY e os demais aparecessem como SECONDARY.

```
rs0 [direct: primary] test> rs.status()
{
  set: 'rs0',
  date: ISODate('2025-11-05T13:04:06.188Z'),
  myState: 1,
  term: Long('2'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1762347837, i: 1 }), t: Long('2') },
    lastCommittedWallTime: ISODate('2025-11-05T13:03:57.331Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1762347837, i: 1 }), t: Long('2') },
    appliedOpTime: { ts: Timestamp({ t: 1762347837, i: 1 }), t: Long('2') },
    durableOpTime: { ts: Timestamp({ t: 1762347837, i: 1 }), t: Long('2') },
    lastAppliedWallTime: ISODate('2025-11-05T13:03:57.331Z'),
    lastDurableWallTime: ISODate('2025-11-05T13:03:57.331Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1762347837, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'stepUpRequestSkipDryRun',
    lastElectionDate: ISODate('2025-11-05T12:52:27.277Z'),
    electionTerm: Long('2'),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1762347142, i: 1 }), t: Long('1') },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1762347142, i: 1 }), t: Long('1') },
    numVotesNeeded: 2,
    priorityAtElection: 1,
    electionTimeoutMillis: Long('10000'),
    priorPrimaryMemberId: 0,
    numCatchUpOps: Long('0'),
    newTermStartDate: ISODate('2025-11-05T12:52:27.293Z'),
    wMajorityWriteAvailabilityDate: ISODate('2025-11-05T12:52:27.804Z')
  },
  electionParticipantMetrics: {
    votedForCandidate: true,
    electionTerm: Long('1'),
    lastVoteDate: ISODate('2025-11-05T12:52:02.619Z'),
```

Configuração da aplicação Node.js / Express com TypeScript

A aplicação base já existente foi modularizada conforme boas práticas:

- src/app.ts contém registro de middlewares globais e rotas.
- src/server.ts inicia o servidor HTTP.
- src/config/database.ts implementa a lógica de conexão com o MongoDB, incluindo:
    - Leitura de MONGODB_URI do .env.
    - Conexão com URI que contempla os três nós do replica set (mongodb://host1:27017,host2:27018,host3:27019/dbname?replicaSet=rs0).
    - Registro de listeners em mongoose.connection para eventos connected, disconnected, reconnected e error, que por sua vez criam entradas na coleção de logs.
- src/middlewares/logger.ts implementa Winston + Morgan integrados, com transporte para console e arquivo server.log.
- src/middlewares/dbLogger.ts intercepta as respostas HTTP e persiste operações de leitura e escrita bem sucedidas na coleção logs.
- Modelo Mongoose Log definido com campos: method, route, operation, status, responseTime, createdAt. A persistência foi feita usando new Log(...).save() para evitar problemas de tipagem do Model.create() em TypeScript.

Teste de replicação e verificação de dados

Para validar o fluxo completo foram aplicados os seguintes passos:

1. Inserção de um documento de teste via rota HTTP da aplicação (por exemplo, POST /api/users contendo um objeto de teste com campo nome).

2. Verificação da presença do documento em cada instância MongoDB — realizada conectando localmente em cada porta (ou via MongoDB Compass usando mongodb://localhost:27017,localhost:27018,localhost:27019/?replicaSet=rs0), confirmando que o documento inserido no primário foi replicado para as secundárias.

3. Confirmação de que os eventos de conexão/desconexão/reconexão e as operações CRUD foram registradas na coleção logs do banco, permitindo auditoria de operações e análise de disponibilidade.

```
rs0 [direct: primary] test> use testdb
switched to db testdb
rs0 [direct: primary] testdb> db.teste.insertOne({ nome: "Replica funcionando!", data: new Date() })
{
  acknowledged: true,
  insertedId: ObjectId('690b4c20683f52ec72ce5f47')
}
```

Conclusões da etapa

Nesta terceira semana foi possível estabelecer uma infraestrutura funcional de replicação MongoDB, integrar a aplicação Node.js para uso do replica set e implementar registro persistente de logs das operações e eventos de conectividade. O experimento demonstrou replicação bem-sucedida de um objeto de teste entre todas as instâncias, e o mecanismo de logging confirmou a auditoria das operações.

Próximos passos

1. **Migração para ambiente em nuvem (Cloud):** criar cluster gerenciado (por exemplo, MongoDB Atlas) ou provisionar VMs em cloud provider (AWS, GCP, Azure) com configuração de replica set e regras de segurança (VPC, regras de firewall). Ajustar variáveis de ambiente MONGODB_URI para apontar ao cluster provisionado e validar failover e logs em produção.

2. **Provisionamento em Proxmox:** automatizar a criação de três containers LXC (ou VMs) dentro do Proxmox, aplicar script de instalação do MongoDB, configurar mongod.conf com replSetName: "rs0" e executar rs.initiate() via provisionamento (cloud-init ou Ansible). Registrar procedimentos de backup e restauração dos volumes persistentes.