Steps I followed:

MONGODB

1. Created docker-compose.yml (with 8 containers: 3 configService replicas, 3 replica nodes, 1 routing service, and 1 neo4j):
2. version: '3.8'
3. services:
4. *# Config Servers*
5. cfg1:
6. image: mongo
7. container\_name: cfg1
8. command: mongod --configsvr --replSet rs\_config --port 27017
9. volumes:
10. - ./data/cfg1:/data/db
11. networks:
12. - mongo\_net
13. cfg2:
14. image: mongo
15. container\_name: cfg2
16. command: mongod --configsvr --replSet rs\_config --port 27017
17. volumes:
18. - ./data/cfg2:/data/db
19. networks:
20. - mongo\_net
21. cfg3:
22. image: mongo
23. container\_name: cfg3
24. command: mongod --configsvr --replSet rs\_config --port 27017
25. volumes:
26. - ./data/cfg3:/data/db
27. networks:
28. - mongo\_net
29. *# Shard 1 Replica Set*
30. shard1\_1:
31. image: mongo
32. container\_name: shard1\_1
33. command: mongod --shardsvr --replSet rs\_shard1 --port 27017
34. volumes:
35. - ./data/shard1\_1:/data/db
36. networks:
37. - mongo\_net
38. shard1\_2:
39. image: mongo
40. container\_name: shard1\_2
41. command: mongod --shardsvr --replSet rs\_shard1 --port 27017
42. volumes:
43. - ./data/shard1\_2:/data/db
44. networks:
45. - mongo\_net
46. shard1\_3:
47. image: mongo
48. container\_name: shard1\_3
49. command: mongod --shardsvr --replSet rs\_shard1 --port 27017
50. volumes:
51. - ./data/shard1\_3:/data/db
52. networks:
53. - mongo\_net
54. *# Router (mongos)*
55. router:
56. image: mongo
57. container\_name: router
58. command: mongos --configdb rs\_config/cfg1:27017,cfg2:27017,cfg3:27017
59. ports:
60. - "27017:27017"
61. depends\_on:
62. - cfg1
63. - cfg2
64. - cfg3
65. - shard1\_1
66. - shard1\_2
67. - shard1\_3
68. networks:
69. - mongo\_net
70. neo4j:
71. image: neo4j:latest
72. container\_name: neo4j
73. ports:
74. - "7474:7474" *# HTTP*
75. - "7687:7687" *# Bolt*
76. environment:
77. - NEO4J\_AUTH=neo4j/DiscoDeewaneS
78. - NEO4J\_apoc\_export\_file\_enabled=true
79. - NEO4J\_apoc\_import\_file\_enabled=true
80. - NEO4J\_apoc\_import\_file\_use\_\_neo4j\_\_config=true
81. - NEO4JLABS\_PLUGINS=["apoc"]
82. volumes:
83. - ./neo4j/data:/data
84. - ./neo4j/import:/import
85. networks:
86. - mongo\_net
87. networks:
88. mongo\_net:
89. driver: bridge
90. Run “docker-compose up -d” to start all the containers

A screen shot of a computer

Description automatically generated

1. Entered into one container using “docker exec -it cfg1 mongosh”. It enters into mongosh (mongo shell)
2. Switch to admin by running “use admin”
3. Initiate the replicaSets of configService using command (in cfg1 container):

rs.initiate(

{\_id: 'rs\_config',

members:

[

{\_id: 0, host: 'cfg1:27017'},

{\_id: 1, host: 'cfg2:27017'},

{\_id: 2, host: 'cfg3:27017'}]})

A screen shot of a computer

Description automatically generated

1. Run “docker exec -it shard1\_1 mongosh”, Initiate the replicaSets of shards using command (in shard1\_1 container):

rs.initiate(

{\_id: 'rs\_shard1',

members:

[

{\_id: 0, host: 'shard1\_1:27017'},

{\_id: 1, host: 'shard1\_2:27017'},

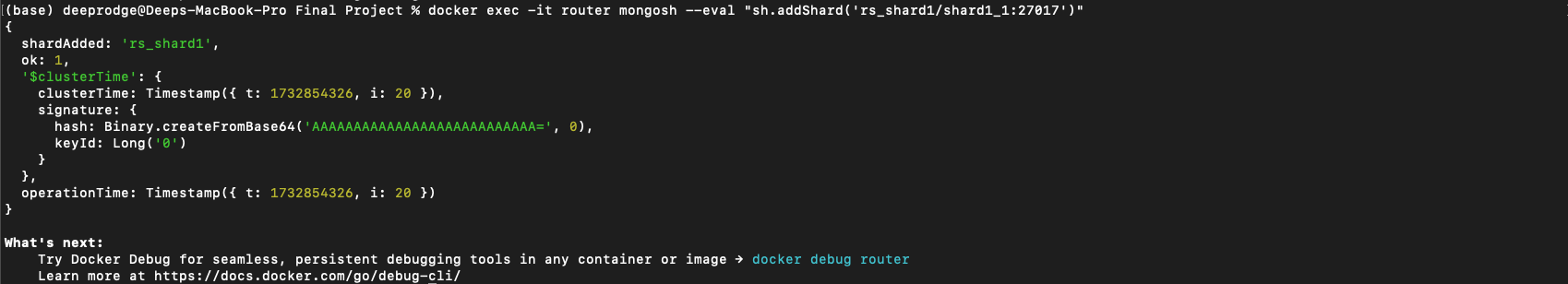
{\_id: 2, host: 'shard1\_3:27017'}]})

A screen shot of a computer

Description automatically generated

1. Run “docker exec -it router mongosh” , Enable sharding for database (in router container, steps 6-12):

sh.addShard('rs\_shard1/shard1\_1:27017')



1. Add shard to cluster using command:

sh.enableSharding('mydb')

A black screen with white text

Description automatically generated

1. Create collections with sharding:

sh.shardCollection("mydb.users", {"location": "hashed"})

sh.shardCollection("mydb.content", {"created\_by": "hashed"})

sh.shardCollection("mydb.interactions", {"user\_id": "hashed"})

sh.shardCollection("mydb.images", {"\_id": "hashed"})

A screenshot of a computer program

Description automatically generated

1. Import data (run these from your host machine):

docker cp users.json router:/users.json

docker exec -it router mongoimport --db mydb --collection users --file /users.json --jsonArray

docker cp content.json router:/content.json

docker exec -it router mongoimport --db mydb --collection content --file /content.json --jsonArray

docker cp interactions.json router:/interactions.json

docker exec -it router mongoimport --db mydb --collection interactions --file /interactions.json --jsonArray

docker cp images.json router:/images.json

docker exec -it router mongoimport --db mydb --collection images --file /images.json --jsonArray

A screenshot of a computer program

Description automatically generated

1. To get shard map showing all servers:

db.adminCommand('getShardMap')

A black screen with green text

Description automatically generated

1. To check replica set status and health (run in ‘shard1\_1’ container):

rs.status()

A computer screen shot of a black screen

Description automatically generated

A screen shot of a computer program

Description automatically generated

rs.isMaster().primary



1. To list all databases and their sizes (in ‘router’ container):

db.adminCommand({listDatabases:1})

A screenshot of a computer program

Description automatically generated

1. To check collection, images distribution across shards:

db.collection.getShardDistribution()

A screenshot of a computer

Description automatically generated

db.images.getShardDistribution()

A screenshot of a computer

Description automatically generated

1. Fault Tolerance Testing: To test failover by stepping down current primary:

rs.stepDown()

A screen shot of a computer

Description automatically generated

1. check replication configuration:

rs.status()

A screen shot of a computer program

Description automatically generated

NEO4J

1. Added neo4j container to docker-compose.yml and re-composed the network:

A screen shot of a computer

Description automatically generated

1. Get into neo4j shell using:

docker exec -it neo4j cypher-shell -u neo4j -p DiscoDeewaneS

1. Added schema using:

// Create constraints

CREATE CONSTRAINT user\_id IF NOT EXISTS FOR (u:User) REQUIRE u.id IS UNIQUE;

CREATE CONSTRAINT content\_id IF NOT EXISTS FOR (c:Content) REQUIRE c.id IS UNIQUE;

CREATE CONSTRAINT tag\_name IF NOT EXISTS FOR (t:Tag) REQUIRE t.name IS UNIQUE;

A screenshot of a computer program

Description automatically generated

// Create indexes

CREATE INDEX user\_location IF NOT EXISTS FOR (u:User) ON (u.location);

CREATE INDEX content\_type IF NOT EXISTS FOR (c:Content) ON (c.type);

A screenshot of a computer

Description automatically generated

1. Import data into graph:

CALL apoc.load.json("file:///import/users.json") YIELD value

MERGE (u:User {id: value.\_id})

SET u.name = value.name,

u.location = value.location;

CALL apoc.load.json("file:///import/content.json") YIELD value

MERGE (c:Content {id: value.\_id})

SET c.type = value.type,

c.description = value.description

WITH c, value

UNWIND value.tags AS tag

MERGE (t:Tag {name: tag})

MERGE (c)-[:HAS\_TAG]->(t);

CALL apoc.load.json("file:///import/interactions.json") YIELD value

MATCH (u:User {id: value.user\_id})

MATCH (c:Content {id: value.content\_id})

WITH u, c, value

CALL {

WITH u, c, value

MERGE (u)-[r:INTERACTED]->(c)

SET r.type = value.interaction\_type,

r.timestamp = value.timestamp,

r.comment\_text = CASE value.interaction\_type

WHEN 'comment' THEN value.comment\_text

ELSE NULL END

};

A screenshot of a computer

Description automatically generated

Final graph UI:

A screenshot of a computer

Description automatically generated