**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Дисциплина:**

«Архитектура систем обработки больших данных»

**КУРСОВОЙ ПРОЕКТ**

«Проектирование хранилища данных на основе трех типов NoSQL баз данных для решения в конкретной предметной области»

**Выполнил:**

Козлов П. В.

**Проверил:**

Королёва Юлия Александровна, доцент (квалификационная категория "ординарный доцент"), факультет ПИиКТ

Санкт-Петербург

2024г.

## **Цель и задача работы**

**Цель:** Изучить достоинства и недостатки различных видов NoSQL хранилищ данных, применяя их на практике для создания информационной системы, моделирующей выбранную предметную область.

**Задачи:**

1. Представить предметную область, для которой производится разработка хранилища, дополнив исходное описание. Данные предметной области должны быть распределены на 3 модуля баз данных с различным типом хранения.
2. Сформировать базу данных для предметной области с использованием типов СУБД, соответствующих трём уровням:

• документно-ориентированная БД – MongoDB/CouchDB/…

• колоночная БД – Cassandra

• графовая БД – Neo4j/OrientDB/…

1. Разработать API, позволяющее пользователю реализовывать предусмотренные предметной областью действия, состоящие в получении запрашиваемой из системы информации, в сохранении новой информации и в обновлении существующей.
2. Расширить спроектированные базы данных до трех кластеров (по одному на каждую базу данных), таким образом, чтобы каждый кластер состоял из трех узлов с состоянием соответствующей базы данных.
3. Обеспечить надежность кластера.

## **ХОД РАБОТЫ**

1. **ПРЕДСТАВЛЕНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

Множество компьютерных игр, продаваемых в магазине. Основной информацией об игре выступает ее название, уникальный идентификатор, дата выхода, описание и изображение(аватар) игры для отображения в магазине. У каждой игры есть своя компания-разработчик и компания-издатель, также каждая игра принадлежит к определенному жанру, для упрощения поиска и группирования схожих игр. В связи с разнообразием стран и народов, возникает необходимость переводить игры на другие языки.

Данные будут распределены между документной, колоночной и графовой бд.

**Документная (MongoDB):**

* ID
* Name

**Колоночная (Cassandra)**

* ID
* Date

**Графовая (Neo4j)**

* ID -[lan]->Language
* ID-[dev]->Developer
* ID-[gen]->Genre
* ID-[pub]->Publisher

Так как данные о языке, разработчике, издателе, жанре являются не настолько многочисленными относительно ID, названия и даты издания, поэтому они располагаются в графовой бд, остальные данные будут распределены для уменьшения размера в каждой бд.

1. **ФОРМИРОВАНИЕ БАЗ ДАННЫХ**

Поскольку среди перечисленных выше баз данных, которые будут использованы далее, структурированные данные хранятся только в колоночной Cassandra, только для нее необходимо предварительное задание отношения:

Листинг 1 – Задание пространства ключей games и таблицы user\_data

CREATE KEYSPACE IF NOT EXISTS games WITH REPLICATION =

{ 'class' : 'SimpleStrategy',

'replication\_factor' : '1'

};

CREATE TABLE IF NOT EXISTS games.data (

ID int PRIMARY KEY,

date text

);

Коллекции в документной базе данных создаются проще (Листинг 2).

Листинг 2 – Задание коллекций profiles и dialogues

use test;

db.createCollection('test');

Все базы данных развернуты будут в Docker.

1. **РАЗРАБОТКА API**

В качестве языка программирования, на котором реализовывалась разработка API, был выбран Python, так как он имеет поддерживаемые и регулярно обновляемые драйверы для взаимодействия с применяемыми базами данных. Также Python позволяет относительно просто конвертировать данные в разные форматы.

Функционал драйверов применяемых баз данных поставляется в модулях pymongo, cassandra и neo4j.

Для каждой базы данных были созданы три класса, каждый отвечающий за подключение и реализации методов получения, записи, обновления, удаления (CRUD) относительно своего языка запросов(Листинг 3):

Листинг 3 - Пример класса методов для Mongo

class MongoAPI:

def \_\_init\_\_(self, data):

log.basicConfig(level=log.DEBUG, format='%(asctime)s %(levelname)s:\n%(message)s\n')

self.client = MongoClient('mongodb://mongodb1:27017/?replicaSet=rs0')

database = data['database']

collection = data['collection']

cursor = self.client[database]

self.collection = cursor[collection]

self.data = data

def read(self):

log.info('Reading All Data')

if 'Filter' in str(self.data):

filt = self.data['Filter']

documents = self.collection.find(filt)

else:

documents = self.collection.find()

if 'count' in str(self.data):

output = documents.count()

else:

output = [{item: data[item] for item in data if item != '\_id'} for data in documents]

return output

def write(self, data):

log.info('Writing Data')

new\_document = data['Document']

response = self.collection.insert\_one({"ID" : new\_document["ID"], "name" : new\_document["name"]})

output = {'Status': 'Successfully Inserted',

'Document\_ID': str(response.inserted\_id)}

return output

def update(self):

log.info('Updating Data')

filt = self.data['Filter']

updated\_data = {"$set": {"name" : self.data['DataToBeUpdated']["name"]}}

response = self.collection.update\_many(filt, updated\_data)

output = {'Status': 'Successfully Updated' if response.modified\_count > 0 else "Nothing was updated."}

return output

def delete(self, data):

log.info('Deleting Data')

filt = data['Filter']

response = self.collection.delete\_many(filt)

output = {'Status': 'Successfully Deleted' if response.deleted\_count > 0 else "Document not found."}

return output

Остальные классы созданы по образу этого, меняются только запросы и как связаться с базой данных.

Для отправки запросов был использован Flask, который принимает и проверяет json на корректность для определенной операции, после выполнят нужный метод у каждого объекта класса.

Листинг 4 - Пример метода POST для МонгоDB.

def mongo\_write():

data = request.json

if data is None or data == {} or 'Document' not in data:

return Response(response=json.dumps({"Error": "Please provide connection information"}),

status=400,

mimetype='application/json')

obj1 = MongoAPI(data)

obj2 = CassandraAPI()

obj3 = Neo4jAPI()

obj2.write(data)

obj3.write(data)

response = obj1.write(data)

return Response(response=json.dumps(response),

status=200,

mimetype='application/json')

Листинг 5 - реализация метода write для МонгоDB.

def write(self, data):

log.info('Writing Data')

new\_document = data['Document']

response = self.collection.insert\_one({"ID" : new\_document["ID"], "name" : new\_document["name"]})

output = {'Status': 'Successfully Inserted',

'Document\_ID': str(response.inserted\_id)}

return output

**4. ТЕСТИРОВАНИЕ API**

В качестве данных для СУБД были сгенерированы 200000 записей с помщью pyhon скрипта (Листинг 5)

Листинг 5 - скрипт для генерации данных

import random, string, json, csv, ast, pandas

def randomword(length):

letters = string.ascii\_lowercase

return ''.join(random.choice(letters) for i in range(length))

# Press the green button in the gutter to run the script.

if \_\_name\_\_ == '\_\_main\_\_':

#fp = open('games.json', 'w')

data = pandas.read\_csv("games.csv", header=0)

res = list(data.ID)

all\_genre = ['Shooter', 'Adventure', 'Puzzle', 'Fighting']

all\_language = ['Russia', 'English', 'German', 'Japan']

developers = ['Activision', 'Blizzard', 'Bethesda', 'Arkane', 'Devolver\_Digital', 'From\_Softaware', 'Ubisoft']

publisher = ['Activision', 'Blizzard', 'Bethesda', 'Ubisoft']

for i in range(1000):

dict = {

#'name': randomword(10),

#'ID': random.randint(1, 10000000),

'ID': res[i],

#'date': str(random.randint(1, 31))+'.'+str(random.randint(1, 12))+'.'+str(random.randint(2000, 2024))

'developer': random.choice(developers),

'publisher': random.choice(publisher),

'genre': random.choice(all\_genre),

'language': random.choice(all\_language)

}

with open("games\_n.csv", "a", newline='') as f:

w = csv.DictWriter(f, dict.keys())

#w.writeheader()

w.writerow(dict)

#json.dump(dict, fp, indent=7)

Далее данные были загружены в свои базы, которые соотносятся по предметной области.

Рисунок 1 - Содержимое Mongo

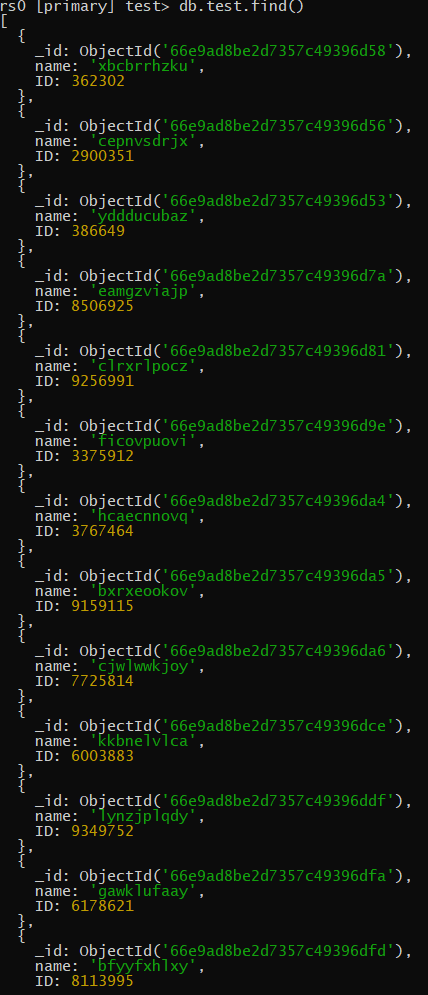


Рисунок 2 - Содержимое Cassandra

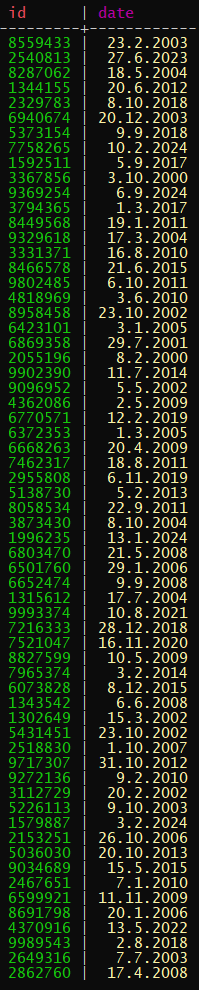
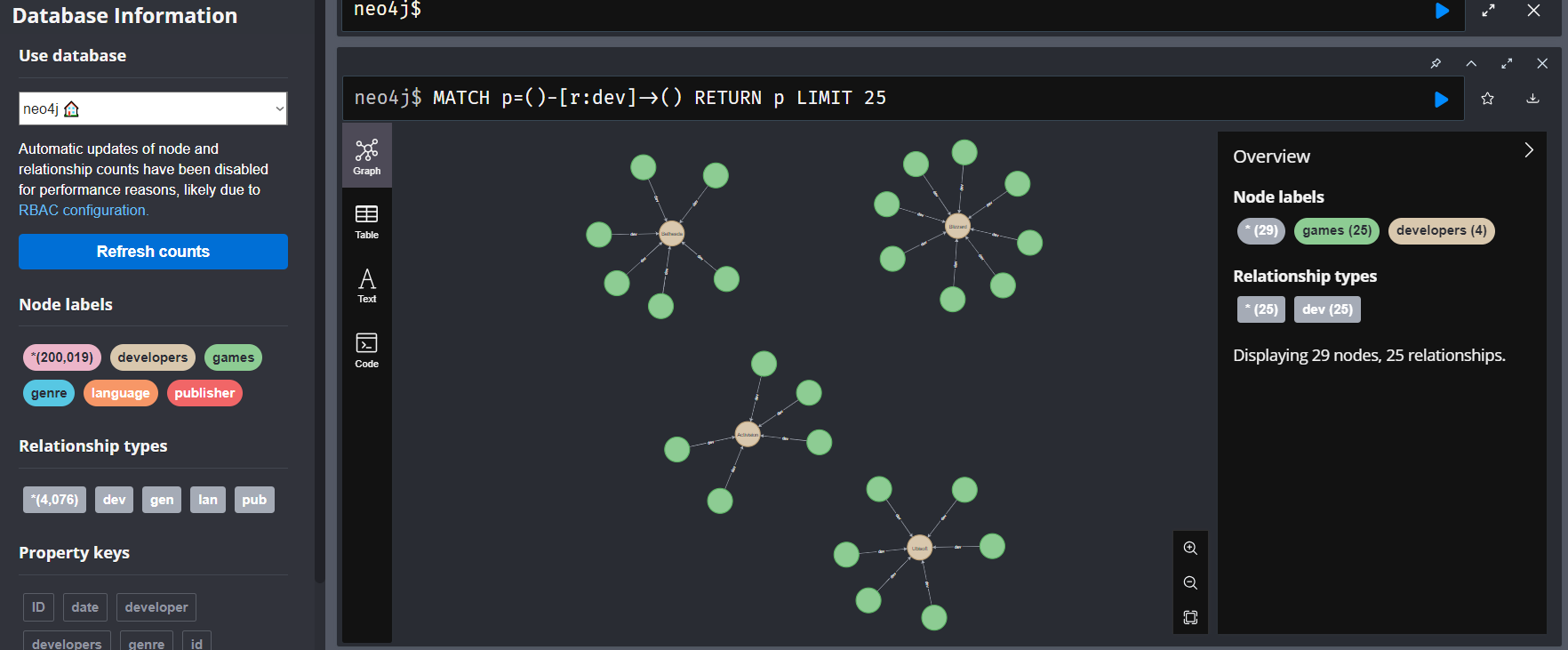


Рисунок 3 - Содержимое Neo4j



**5. СОЗДАНИЕ КЛАСТЕРОВ**

Далее было необходимо распространить созданные базы данных на связанные узлы кластеров. Для налаживания взаимодействия между контейнерами был использован docker-compose.

Листинг 6 - docker-compose Mongodb

version: '3.8'

networks:

app-network:

driver: bridge

services:

mongodb1:

image: mongo:latest

container\_name: mongodb1

restart: always

ports:

- "27017:27017"

networks:

- app-network

volumes:

- mongodb1-data:/data/db

command: ["--replSet", "rs0", "--bind\_ip\_all", "--port", "27017"]

healthcheck:

test: ["CMD", "mongo", "--eval", "db.adminCommand('ping')"]

interval: 5s

start\_period: 10s

mongodb2:

image: mongo:latest

container\_name: mongodb2

restart: always

ports:

- "27018:27018"

networks:

- app-network

volumes:

- mongodb2-data:/data/db

command: ["--replSet", "rs0", "--bind\_ip\_all", "--port", "27018"]

healthcheck:

test: ["CMD", "mongo", "--eval", "db.adminCommand('ping')"]

interval: 5s

start\_period: 10s

mongodb3:

image: mongo:latest

container\_name: mongodb3

restart: always

ports:

- "27019:27019"

networks:

- app-network

volumes:

- mongodb3-data:/data/db

command: ["--replSet", "rs0", "--bind\_ip\_all", "--port", "27019"]

healthcheck:

test: ["CMD", "mongo", "--eval", "db.adminCommand('ping')"]

interval: 5s

start\_period: 10s

volumes:

mongodb1-data:

mongodb2-data:

mongodb3-data:

Листинг 6 - docker-compose Cassandra.

services:

cassandra1:

image: cassandra:latest

container\_name: cassandra1

ports:

- "9042:9042"

environment:

- CASSANDRA\_SEEDS=cassandra1,cassandra2,cassandra3

networks:

- cassandra-network

cassandra2:

image: cassandra:latest

container\_name: cassandra2

ports:

- "9043:9042"

environment:

- CASSANDRA\_SEEDS=cassandra1,cassandra2,cassandra3

networks:

- cassandra-network

cassandra3:

image: cassandra:latest

container\_name: cassandra3

ports:

- "9044:9042"

environment:

- CASSANDRA\_SEEDS=cassandra1,cassandra2,cassandra3

networks:

- cassandra-network

networks:

cassandra-network:

Листинг 7 - Docker-compose Neo4j

networks:

neo4j-internal:

services:

neo4j\_server1:

image: neo4j:latest

hostname: neo4j\_server1

networks:

neo4j-internal:

aliases:

- neo4j-network

ports:

- "7474:7474"

- "7687:7687"

volumes:

- ./neo4j/data/server1:/data

- ./neo4j/logs/server1:/logs

- ./neo4j/conf/server1:/conf

- ./neo4j/import/server1:/import

- ./neo4j/neo4j.conf:/conf/neo4j.conf

environment:

- NEO4J\_ACCEPT\_LICENSE\_AGREEMENT=yes

- NEO4J\_AUTH=neo4j/pass1234321

- NEO4J\_EDITION=docker\_compose

- NEO4J\_initial\_server\_mode\_\_constraint=PRIMARY

- NEO4J\_server\_config\_strict\_\_validation\_enabled=false

user: 0:0

neo4j\_server2:

image: neo4j:latest

hostname: neo4j\_server2

networks:

neo4j-internal:

aliases:

- neo4j-network

ports:

- "7475:7474"

- "7688:7687"

volumes:

- ./neo4j/data/server2:/data

- ./neo4j/logs/server2:/logs

- ./neo4j/conf/server2:/conf

- ./neo4j/import/server2:/import

- ./neo4j/neo4j.conf:/conf/neo4j.conf

environment:

- NEO4J\_ACCEPT\_LICENSE\_AGREEMENT=yes

- NEO4J\_AUTH=neo4j/pass1234321

- NEO4J\_EDITION=docker\_compose

- NEO4J\_initial\_server\_mode\_\_constraint=PRIMARY

- NEO4J\_server\_config\_strict\_\_validation\_enabled=false

user: 0:0

neo4j\_server3:

image: neo4j:latest

hostname: neo4j\_server3

networks:

neo4j-internal:

aliases:

- neo4j-network

ports:

- "7476:7474"

- "7689:7687"

volumes:

- ./neo4j/data/server3:/data

- ./neo4j/logs/server3:/logs

- ./neo4j/conf/server3:/conf

- ./neo4j/import/server3:/import

- ./neo4j/neo4j.conf:/conf/neo4j.conf

environment:

- NEO4J\_ACCEPT\_LICENSE\_AGREEMENT=yes

- NEO4J\_AUTH=neo4j/pass1234321

- NEO4J\_EDITION=docker\_compose

- NEO4J\_initial\_server\_mode\_\_constraint=PRIMARY

- NEO4J\_server\_config\_strict\_\_validation\_enabled=false

user: 0:0

В результате получилось три кластера, в которых узлы осознавали свои роли и реплицировали данные (Рисунки 4, 5, 6).

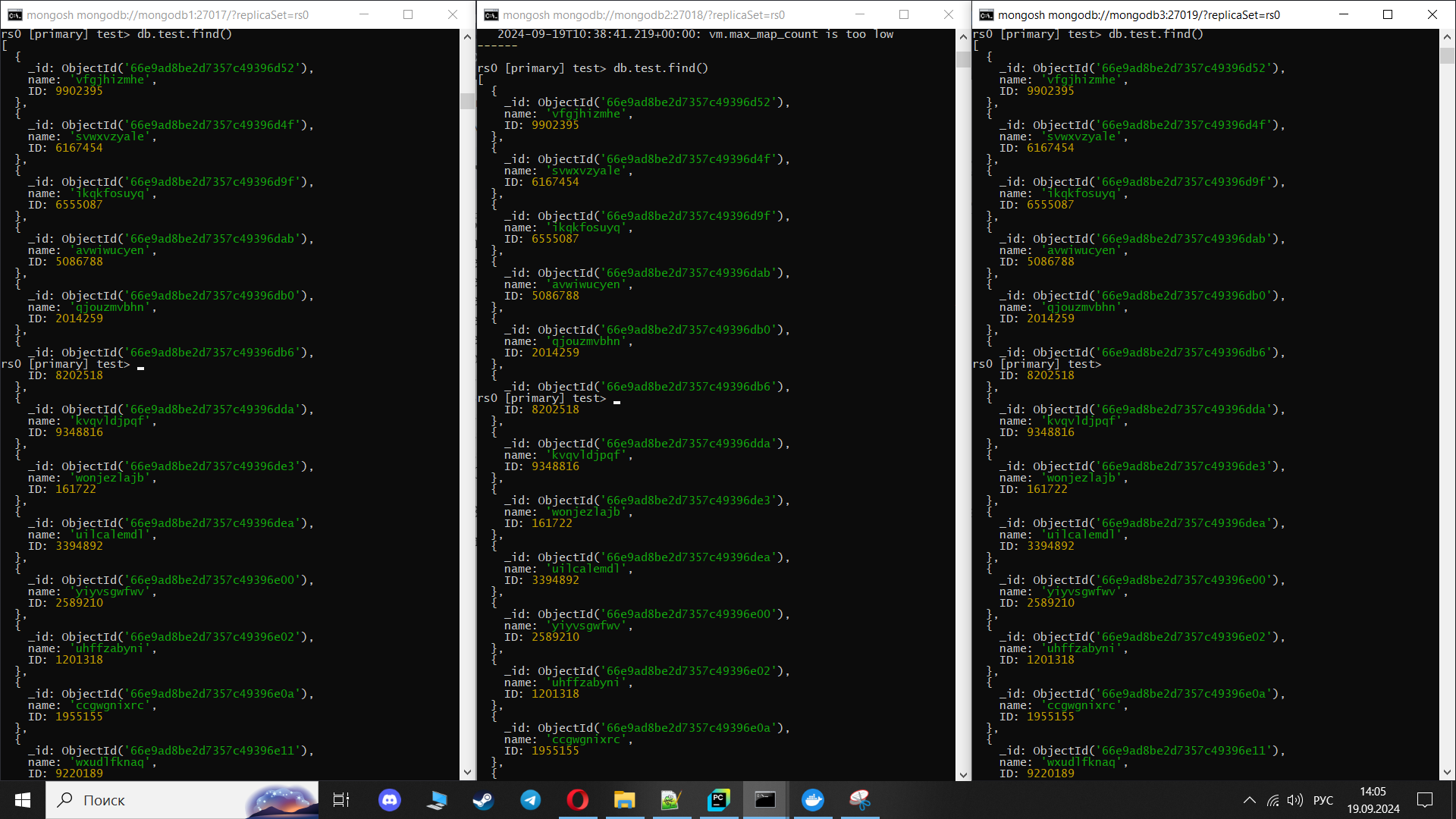


Рисунок 4 - Синхронизированные состояния mongodb

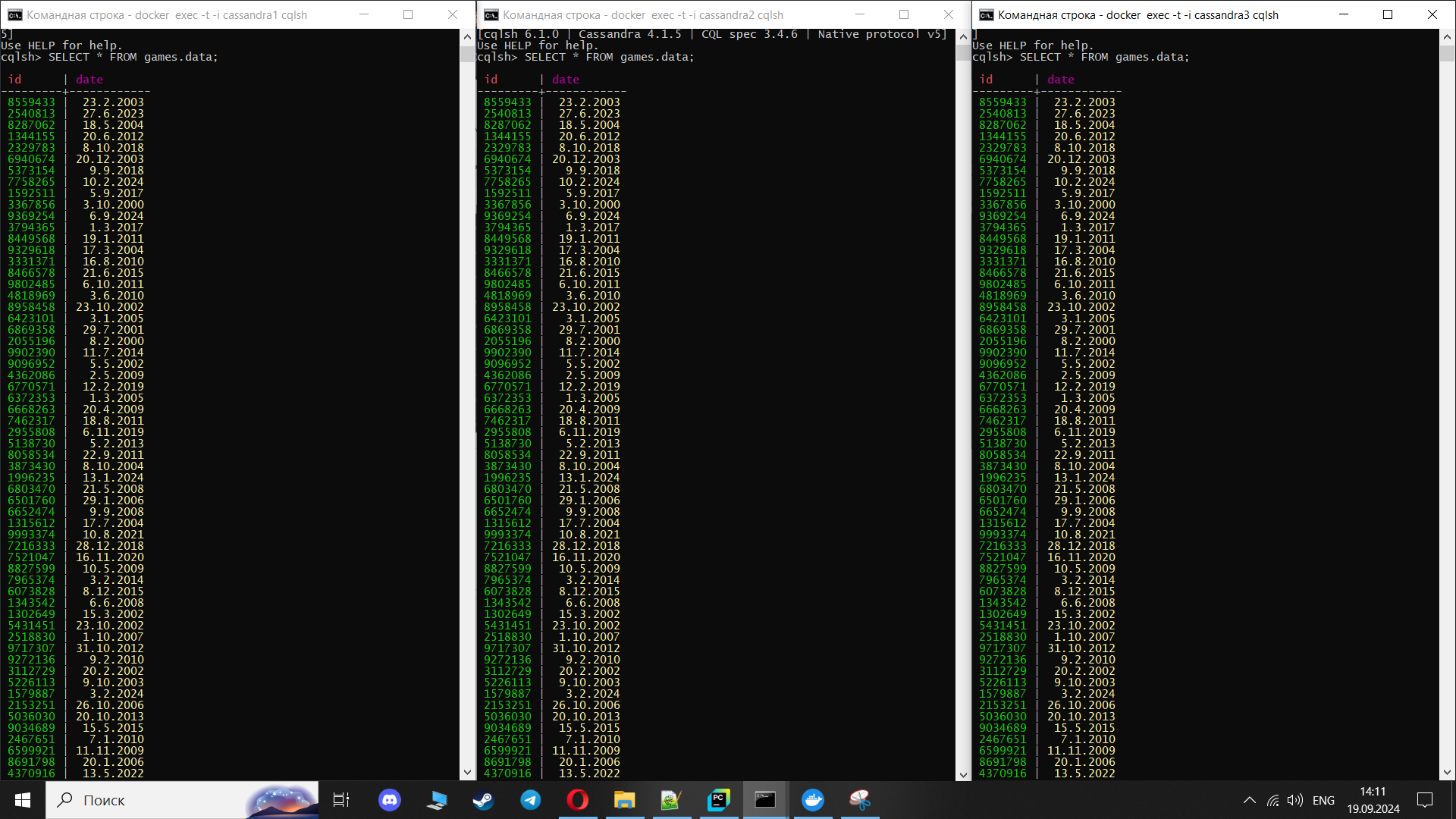


Рисунок 5 - Синхронизация в Cassandra.

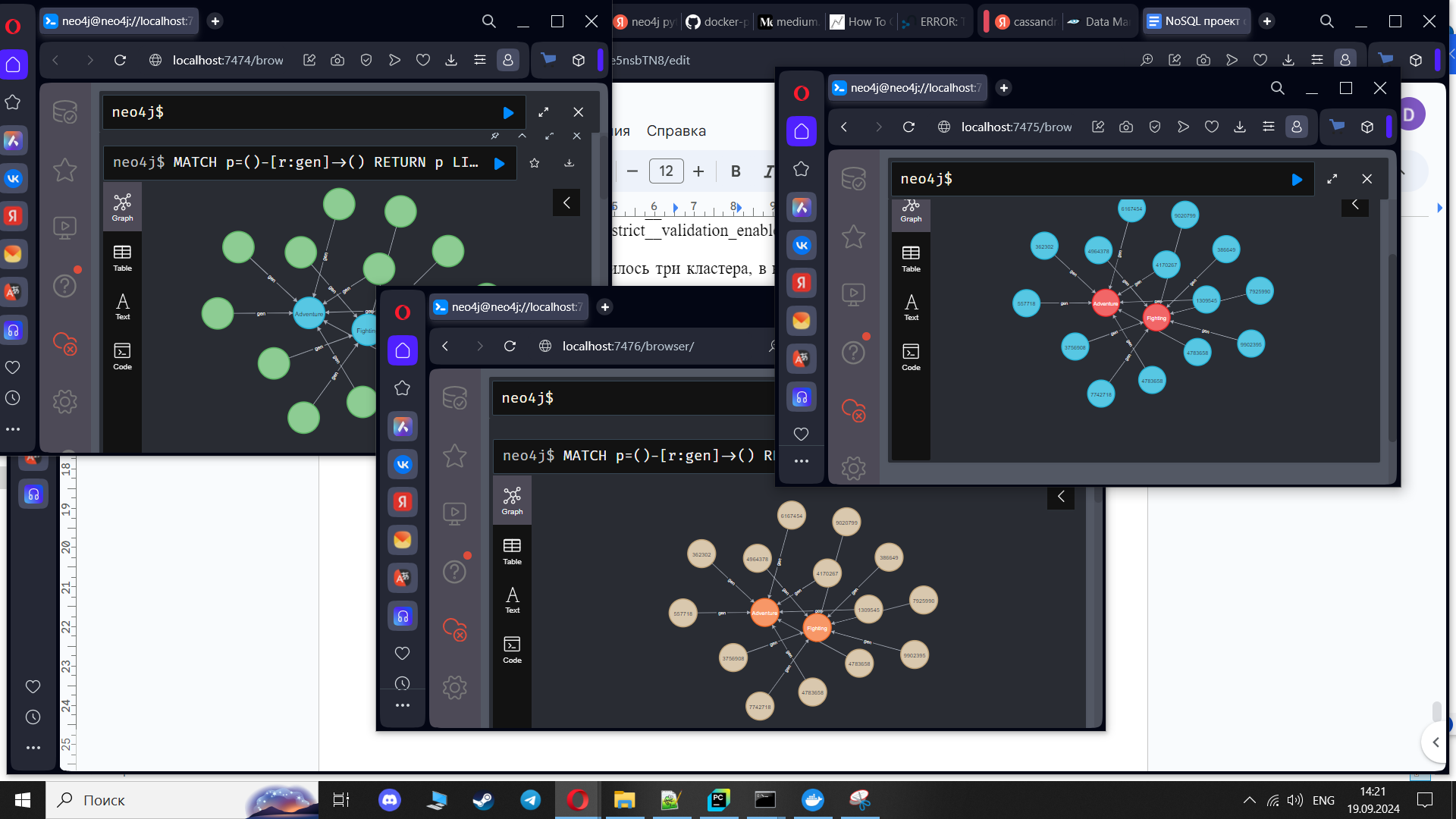


Рисунок 6 - Синхронизация Neo4j.

**6. ОБЕСПЕЧЕНИЕ НАДЕЖНОСТИ**

После отключения одного узла в кластере mongo оставшиеся два узла провели выборы primary node, кластер сохранил работоспособность(Рисунок 8).

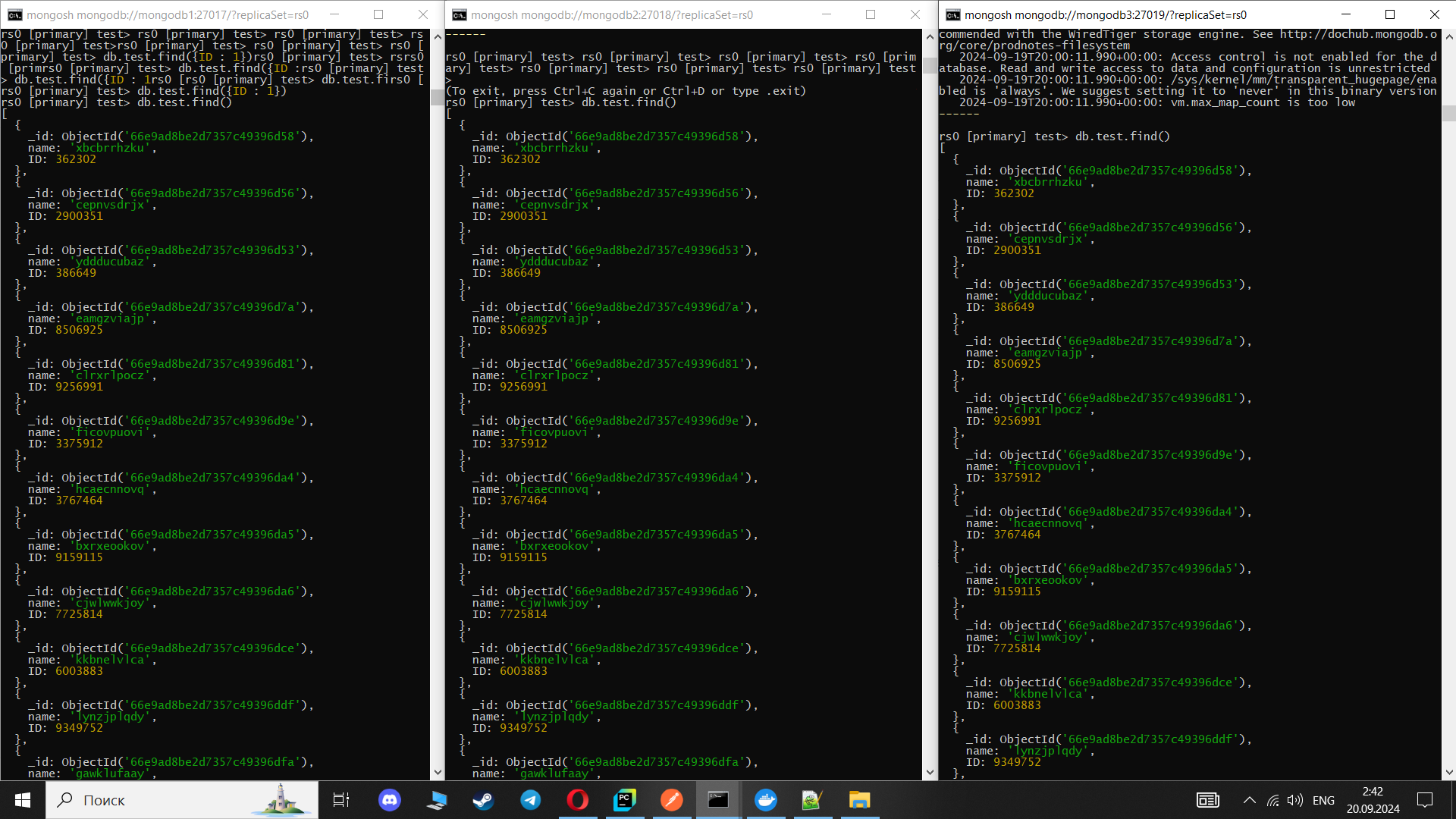


Рисунок 7 - Все три “живых” кластера

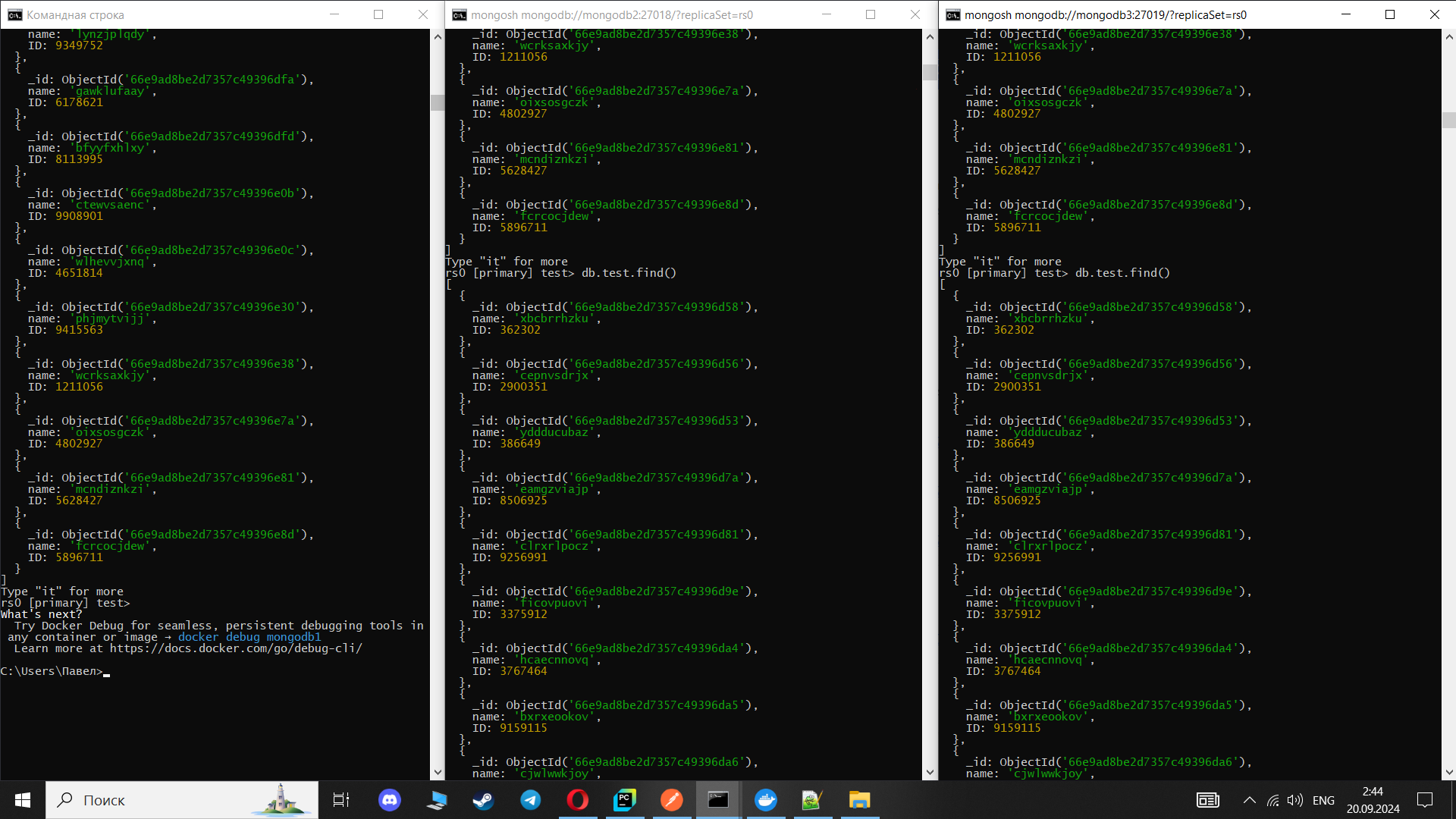


Рисунок 8 - один кластер “упал”.

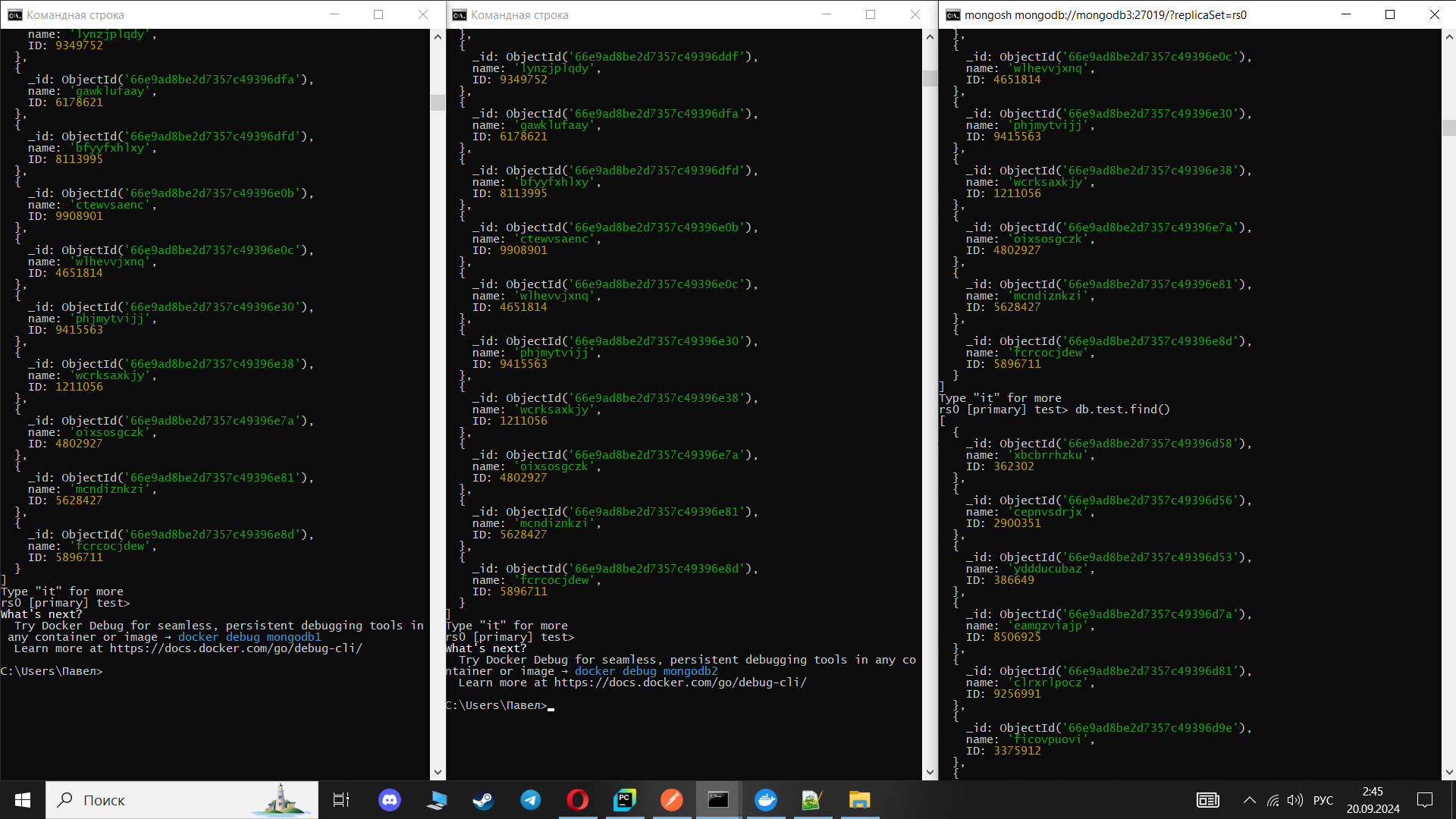


Рисунок 9 - Один кластер остался

## **Выводы**

В процессе выполнения практического задания было осуществлено представление предметной области, формирование баз данных, разработка API и построение кластеров с обеспечением его надежности.

Цель по изучению достоинств и недостатков различных видов NoSQL хранилищ данных, с применением их на практике для создания информационной системы, моделирующей выбранную предметную область достигнута.  
 Все файлы работы: https://github.com/DeepPK/NoSQL-Database/tree/main