Relatório
Exploração e Análise do Couchbase

Matheus Baron Lauritzen[1], **Gustavo Baron Lauritzen**[1], **Gabriel Bósio**[1]

[1]Escola Politécnica – Ciência da Computação – Universidade do Vale do Itajai (UNIVALI)
Itajaí – Santa Catarina – SC – Brasil

Abstract. *The report presents an overview of the Couchbase database, highlighting its main features. It describes the available licensing models and the main data model used by SGDB, which is based on JSON documents. Among the main features are high scalability, performance, and support for ACID operations. The target audience includes companies that need efficient management of large volumes of data. The report also analyzes Couchbase's position and evolution in the DB-Engines ranking and its positioning in the CAP theorem, emphasizing its preference for consistency and availability.*

Resumo. *O relatório apresenta uma visão geral do banco de dados Couchbase, destacando suas características principais. Ele descreve os modelos de licenciamento disponíveis e o principal modelo de dados utilizado pelo SGDB, que é baseado em documentos JSON. Entre as características principais, estão a alta escalabilidade, desempenho, e suporte para operações ACID. O público-alvo inclui empresas que necessitam de gerenciamento eficiente de grandes volumes de dados. O relatório também analisa a posição e evolução do Couchbase no ranking do DB-Engines e seu posicionamento no teorema CAP, enfatizando sua preferência por consistência e disponibilidade.*

1. Visão Geral

1.1. Descrição geral

Couchbase é um banco de dados NoSQL distribuído, conhecido por sua alta performance, flexibilidade e escalabilidade. Ele combina a facilidade de uso do modelo de dados de documentos com a potência de um banco de dados orientado a chave-valor. Couchbase é amplamente utilizado para aplicações web, móveis e de IoT devido à sua capacidade de lidar com grandes volumes de dados e tráfego em tempo real.

1.2. Modelos de licenciamento

Couchbase oferece diferentes modelos de licenciamento, incluindo:

- **Community Edition:** Gratuita e open-source, adequada para desenvolvedores e pequenos projetos.
- **Enterprise Edition:** Licenciamento comercial, fornecendo suporte técnico, ferramentas avançadas de segurança e desempenho, ideal para aplicações empresariais críticas.

1.3. Descrição do modelo de dados (principal)

O modelo de dados principal utilizado pelo Couchbase é o modelo de documentos, onde os dados são armazenados em documentos JSON. Este modelo oferece flexibilidade e permite a representação de dados complexos de maneira intuitiva e eficiente. Cada documento pode conter uma estrutura diferente, facilitando a evolução do esquema sem necessidade de migrações complexas.

1.4. Características principais

Couchbase possui diversas características que o destacam no mercado:

- **Alta performance:** Suporte a operações de leitura e escrita em milissegundos.
- **Escalabilidade horizontal:** Facilmente escalável adicionando mais nós ao cluster.
- **Persistência e memória:** Combina armazenamento em memória com persistência no disco.
- **Consulta avançada:** Suporte a consultas N1QL, uma linguagem similar ao SQL para JSON.
- **Alta disponibilidade:** Replicação de dados e failover automático.
- **Sincronização de dados:** Sincronização de dados eficiente para aplicações móveis e de IoT.

1.5. Público-alvo ou áreas de aplicação

Couchbase é ideal para:

- Desenvolvedores de aplicações web e móveis que precisam de um banco de dados rápido e flexível.
- Empresas que necessitam de escalabilidade e alta disponibilidade para suas aplicações críticas.
- Setores como comércio eletrônico, telecomunicações, serviços financeiros e saúde, onde grandes volumes de dados em tempo real são comuns.

1.6. URL do desenvolvedor e da documentação

- **Desenvolvedor:** `https://www.couchbase.com`
- **Documentação:** `https://docs.couchbase.com`

1.7. Posição e evolução no ranking do DB-Engines

Couchbase caiu algumas posições no ranking do DB-Engines, porém continua em uma posição alta, segundo a última atualização em junho de 2024 ele está na posição de número 36. Ele continua em uma boa posição, mesmo com a queda de 5 posições em relação a Junho de 2023, pois como o total de posições do ranking é de um pouco mais de 400, então a posição de número 36 significa um bom posicionamento. Para detalhes mais atualizados, consulte o site oficial do DB-Engines: `https://db-engines.com/en/ranking`.

1.8. Posicionamento no teorema CAP

O teorema CAP, formulado por Eric Brewer, afirma que em um sistema de banco de dados distribuído, é impossível garantir simultaneamente Consistência, Alta Disponibilidade e Tolerância a Partições. Segundo o teorema, um sistema pode maximizar apenas dois desses três atributos.

O Couchbase é projetado para ser predominantemente **AP (Alta Disponibilidade e Particionamento)**, o que significa que ele prioriza:

- **Alta Disponibilidade (Availability):** Couchbase garante que cada solicitação de cliente receba uma resposta, mesmo que alguns dos nós do sistema estejam inativos.
- **Tolerância a Partições (Partition Tolerance):** O sistema continua operando apesar de eventuais falhas ou interrupções de rede que dividem o cluster em partes isoladas.

Embora o Couchbase priorize alta disponibilidade e tolerância a partições, ele também oferece mecanismos para manter um nível adequado de consistência. O Couchbase utiliza um modelo de consistência eventual, onde, em condições normais, os dados serão eventualmente consistentes em todo o sistema. Para aplicações que necessitam de maior consistência, Couchbase fornece:

- **Durabilidade síncrona:** Opções para garantir que as operações de gravação sejam confirmadas em múltiplos nós antes de serem consideradas completas.
- **Replicação ativa-passiva:** Permite a configuração de múltiplas réplicas dos dados, garantindo que, mesmo em caso de falha de um nó, os dados ainda estejam disponíveis.
- **N1QL com modo de consistência:** Consultas podem ser configuradas para diferentes níveis de consistência, desde leitura mais recente até leitura mais rápida.

Portanto, enquanto o Couchbase se posiciona como AP no teorema CAP, ele proporciona flexibilidade para ajustar os níveis de consistência conforme as necessidades específicas da aplicação. Essa abordagem híbrida permite que desenvolvedores e arquitetos de sistemas otimizem suas soluções para alcançar o melhor equilíbrio entre consistência, disponibilidade e tolerância a partições.

2. Instalação e/ou configuração

O Couchbase oferece diferentes tipos de Sistemas de Gerenciamento de Banco de Dados (SGBD) para atender a diversas necessidades de implantação e uso. Cada tipo possui características específicas que podem ser adequadas para diferentes cenários e requisitos de infraestrutura. A seguir, detalharemos quatro tipos principais de SGBD para o Couchbase:

2.1. Capella: Couchbase as-a-service

Descrição: Capella é a oferta do Couchbase como um serviço (DBaaS - Database as a Service), onde o Couchbase Server é gerenciado e hospedado na nuvem pelo provedor de serviços.

Características:

- É gerenciado pelo provedor de serviços, eliminando a necessidade de configuração e manutenção por parte do usuário.

- Geralmente oferece escalabilidade automática e alta disponibilidade.
- Os usuários pagam pelo uso conforme demanda, em vez de gerenciar infraestrutura localmente.

2.2. Server: Couchbase locally

Descrição: O Couchbase Server instalado localmente refere-se à implementação do Couchbase em um ambiente local, como máquinas físicas ou máquinas virtuais gerenciadas pelo usuário.
Características:

- Os usuários têm controle total sobre a configuração, manutenção e segurança do banco de dados.
- Requer a instalação e configuração manual pelo administrador do sistema.
- É adequado para cenários onde há necessidade de controle total sobre os dados e infraestrutura.

2.3. Kubernetes Operator: Cloud-native database

Descrição: O Kubernetes Operator permite implantar e gerenciar instâncias do Couchbase Server em ambientes Kubernetes.
Características:

- Utiliza os recursos e características nativas do Kubernetes para automação e orquestração de contêineres.
- Facilita a implantação escalável e resiliente do Couchbase em ambientes de contêineres.
- Oferece integração estreita com as ferramentas e políticas de gerenciamento do Kubernetes.

2.4. Mobile & Edge: Embedded NoSQL

Descrição: O Couchbase Mobile & Edge refere-se à utilização do Couchbase em dispositivos móveis e em dispositivos de borda (edge).
Características:

- Oferece uma versão leve e otimizada do Couchbase para execução em dispositivos com recursos limitados.
- Suporta cenários onde há necessidade de armazenamento local, sincronização offline e processamento de dados em dispositivos móveis e de borda.
- Permite aplicativos móveis e de IoT a acessarem dados de forma rápida e eficiente.

2.5. Explorando o Couchbase Server

A seguir, focaremos no Couchbase Server, uma poderosa solução para suas necessidades de gerenciamento de dados.

2.5.1. Guia passo a passo para a instalação/acesso ao SGDB

Para instalar e acessar o Couchbase Server, siga estes passos:

- Baixe o instalador do Couchbase Server no site oficial.

- Execute o instalador e siga as instruções na tela para configurar o Couchbase Server em sua máquina local ou em uma VM.
- Durante a instalação, configure os nós e clusters conforme necessário para seu ambiente.
- Após a instalação, acesse o console de administração do Couchbase Server através do navegador, utilizando o endereço fornecido durante a configuração inicial.

2.5.2. Configurações iniciais do SGDB

Após a instalação inicial, realize as configurações iniciais no Couchbase Server:

- Faça login no console de administração com as credenciais padrão ou as configuradas durante a instalação.
- Configure as políticas de armazenamento, replicação e segurança de acordo com as necessidades do seu aplicativo.
- Adicione buckets para armazenar seus dados, configurando parâmetros como tipo de bucket, políticas de expiração, entre outros.

2.5.3. Configuração de uma interface gráfica

Para facilitar o gerenciamento do Couchbase Server, configure uma interface gráfica:

- Baixe e instale a ferramenta de interface gráfica do Couchbase disponível no site oficial.
- Conecte-se ao servidor Couchbase especificando o endereço e as credenciais de administração.
- Utilize a interface gráfica para administrar clusters, criar e gerenciar buckets, executar consultas e monitorar o desempenho do Couchbase Server.

3. Casos de uso

Couchbase é amplamente utilizado em áreas que exigem processamento em tempo real e manipulação de grandes volumes de dados. Aqui estão três exemplos principais:

3.1. E-Commerce

Este banco de dados garante uma navegação fluida nos sites e um rápido processamento dos dados de perfil do usuário. Sua escalabilidade permite suportar muitos acessos simultâneos, especialmente durante promoções ou datas comemorativas.

3.2. Telecomunicações

A robustez do Couchbase assegura um serviço confiável mesmo com grandes volumes de dados sendo armazenados, processados e transmitidos. O processamento em tempo real contribui para recomendações de marketing mais precisas.

3.3. Jogos eletrônicos

Couchbase possibilita o rápido registro e acesso às informações e estatísticas dos perfis dos jogadores, além de evitar problemas de latência. Também permite que os servidores escalem para suportar um grande número de jogadores conectados simultaneamente.

4. CRUD com Couchbase Server

Neste projeto, implementamos um sistema CRUD (Create, Read, Update, Delete) utilizando o Couchbase Server como banco de dados. Couchbase Server é um banco de dados NoSQL altamente escalável e distribuído, projetado para aplicações modernas(como já mencionado acima).

Um **cluster** em Couchbase consiste em um conjunto de nós que trabalham juntos para fornecer alta disponibilidade e escalabilidade. Cada nó em um cluster armazena uma parte dos dados e a carga de trabalho é distribuída entre os nós.

Dentro de um cluster, temos **buckets**, que são contêineres lógicos para armazenar documentos. Os buckets podem ser comparados a tabelas em bancos de dados relacionais. Cada bucket pode conter múltiplos **scopes** (escopos), que são contêineres lógicos adicionais para organizar os dados dentro de um bucket. Dentro de cada escopo, temos **collections** (coleções), que são conjuntos de documentos, semelhantes a linhas em uma tabela de um banco de dados relacional.

Por fim, temos os **documents** (documentos), que são as unidades de dados armazenadas nas coleções. Documentos são objetos JSON que podem conter diferentes atributos e valores(também já mencionado acima).

Neste sistema CRUD, os atributos dos documentos são genéricos (ID, Tipo e Nome) para manter a simplicidade e funcionalidade. O objetivo é fornecer uma estrutura básica que pode ser adaptada conforme necessário para diferentes casos de uso.

4.1. Conexão com o Cluster Couchbase

Para se conectar ao cluster Couchbase, utilizamos o seguinte código:

```
from datetime import timedelta
from couchbase.auth import PasswordAuthenticator
from couchbase.cluster import Cluster
from couchbase.options import ClusterOptions

username = "Administrator"
password = "password"
bucket_name = "testeCRUD"
scope_name = "escopoCRUD"
collection_name = "colecaoCRUD"

auth = PasswordAuthenticator(username, password)
cluster = Cluster('couchbase://localhost', ClusterOptions(auth))
cluster.wait_until_ready(timedelta(seconds=5))

cb = cluster.bucket(bucket_name)
cb_coll = cb.scope(scope_name).collection(collection_name)
```

Neste trecho, são configuradas as credenciais de autenticação, o bucket, o escopo e a coleção. Em seguida, uma conexão é estabelecida com o cluster Couchbase e as referências necessárias são obtidas.

4.2. Função de Criação de Documentos

A função `create_document_window` abre uma nova janela para criar um novo documento:

```
def create_document_window():
    def create_document():
        try:
            doc = {
                "type": type_entry.get(),
                "id": int(id_entry.get()),
                "name": name_entry.get()
            }
            key = str(doc["id"])
            result = cb_coll.upsert(key, doc)
            display_documents()
            messagebox.showinfo("Success", f"Document created
                ↪ with CAS: {result.cas}")
            window.destroy()
        except Exception as e:
            messagebox.showerror("Error", str(e))

    window = Toplevel(root)
    window.title("Create Document")

    Label(window, text="Type").grid(row=0, column=0, padx=10,
        ↪ pady=10)
    type_entry = Entry(window)
    type_entry.grid(row=0, column=1, padx=10, pady=10)

    Label(window, text="ID").grid(row=1, column=0, padx=10, pady
        ↪ =10)
    id_entry = Entry(window)
    id_entry.grid(row=1, column=1, padx=10, pady=10)

    Label(window, text="Name").grid(row=2, column=0, padx=10,
        ↪ pady=10)
    name_entry = Entry(window)
    name_entry.grid(row=2, column=1, padx=10, pady=10)

    Button(window, text="Create Document", command=
        ↪ create_document).grid(row=3, column=0, columnspan=2,
        ↪ padx=10, pady=10)
```

A função cria um novo documento com os campos `type`, `id` e `name`, e o insere na coleção.

4.3. Função de Atualização de Documentos

A função `update_document_window` abre uma janela para atualizar um documento existente:

```
def update_document_window():
    def update_document():
        try:
            key = id_entry.get()
            try:
                result = cb_coll.get(key)
            except Exception as e:
                messagebox.showerror("Error", "Document does not
                   ↪  exist and cannot be updated")
                return

            doc = {
                "type": type_entry.get(),
                "id": int(id_entry.get()),
                "name": name_entry.get()
            }
            result = cb_coll.upsert(key, doc)
            display_documents()
            messagebox.showinfo("Success", f"Document updated
               ↪ with CAS: {result.cas}")
            window.destroy()
        except Exception as e:
            messagebox.showerror("Error", str(e))

    window = Toplevel(root)
    window.title("Update Document")

    Label(window, text="ID").grid(row=0, column=0, padx=10, pady
       ↪ =10)
    id_entry = Entry(window)
    id_entry.grid(row=0, column=1, padx=10, pady=10)

    Label(window, text="Type").grid(row=1, column=0, padx=10,
       ↪ pady=10)
    type_entry = Entry(window)
    type_entry.grid(row=1, column=1, padx=10, pady=10)

    Label(window, text="Name").grid(row=2, column=0, padx=10,
       ↪ pady=10)
    name_entry = Entry(window)
    name_entry.grid(row=2, column=1, padx=10, pady=10)

    Button(window, text="Update Document", command=
       ↪ update_document).grid(row=3, column=0, columnspan=2,
       ↪ padx=10, pady=10)
```

A função verifica se o documento existe antes de atualizá-lo. Se o documento não existir, uma mensagem de erro é exibida.

4.4. Função de Exclusão de Documentos

A função `delete_document_window` abre uma janela para deletar um documento existente:

```
def delete_document_window():
    def delete_document():
        try:
            key = id_entry.get()
            result = cb_coll.remove(key)
            display_documents()
            messagebox.showinfo("Success", f"Document deleted
               ↪ with CAS: {result.cas}")
            window.destroy()
        except Exception as e:
            messagebox.showerror("Error", str(e))

    window = Toplevel(root)
    window.title("Delete Document")

    Label(window, text="ID").grid(row=0, column=0, padx=10, pady
       ↪ =10)
    id_entry = Entry(window)
    id_entry.grid(row=0, column=1, padx=10, pady=10)

    Button(window, text="Delete Document", command=
       ↪ delete_document).grid(row=1, column=0, columnspan=2,
       ↪ padx=10, pady=10)
```

A função deleta um documento com base no ID fornecido.

4.5. Função para Exibir Documentos

A função `display_documents` lista todos os documentos da coleção e os exibe em uma `Treeview`:

```
def display_documents():
    try:
        query = f'SELECT * FROM `{bucket_name}`.`{scope_name
           ↪ }`.`{collection_name}`'
        result = cluster.query(query)

        for row in tree.get_children():
            tree.delete(row)

        for row in result:
            doc_id = row[collection_name]['id']
            doc_type = row[collection_name]['type']
            doc_name = row[collection_name]['name']
            tree.insert("", "end", values=(doc_id, doc_type,
               ↪ doc_name))
```

```
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

A função executa uma consulta N1QL para selecionar todos os documentos e insere cada documento na `Treeview`.

4.6. Configuração da Interface Tkinter

A interface Tkinter é configurada da seguinte forma:

```
root = Tk()
root.title("CRUD Couchbase")

columns = ("ID", "Type", "Name")
tree = ttk.Treeview(root, columns=columns, show="headings")
tree.heading("ID", text="ID")
tree.heading("Type", text="Type")
tree.heading("Name", text="Name")
tree.pack(expand=True, fill=BOTH, padx=20, pady=20)

refresh_button = Button(root, text="Refresh Documents", command=
    ↪ display_documents)
refresh_button.pack(pady=10)

button_frame = Frame(root)
button_frame.pack(pady=10)

create_button = Button(button_frame, text="Create Document",
    ↪ command=create_document_window)
create_button.grid(row=0, column=0, padx=5)

update_button = Button(button_frame, text="Update Document",
    ↪ command=update_document_window)
update_button.grid(row=0, column=1, padx=5)

delete_button = Button(button_frame, text="Delete Document",
    ↪ command=delete_document_window)
delete_button.grid(row=0, column=2, padx=5)

display_documents()
root.mainloop()
```

Neste trecho, a janela principal do Tkinter é configurada, incluindo a `Treeview` para exibir documentos e os botões para refrescar, criar, atualizar e deletar documentos.

4.7. Conclusão

O desenvolvimento deste sistema CRUD utilizando Couchbase Server demonstrou a eficiência e a flexibilidade de um banco de dados NoSQL para operações de criação, leitura, atualização e exclusão de documentos. Ao utilizar uma estrutura genérica para os atributos dos documentos (ID, Tipo e Nome), conseguimos criar um sistema simples e funcional que pode ser facilmente adaptado para diferentes cenários e aplicações.

A hierarquia de cluster, bucket, scope, collection e document em Couchbase permite uma organização clara e eficiente dos dados, proporcionando alta escalabilidade e desempenho. A conexão ao cluster Couchbase foi configurada de forma segura e eficiente, garantindo que as operações CRUD fossem realizadas de maneira confiável.

A implementação das funções de criação, atualização e exclusão de documentos mostrou-se direta e intuitiva, aproveitando as funcionalidades da API do Couchbase para gerenciar os documentos de forma eficaz. Além disso, a interface gráfica desenvolvida com Tkinter forneceu uma maneira amigável e visual de interagir com o banco de dados, permitindo que os usuários visualizem e gerenciem os documentos facilmente.

Em resumo, este projeto ilustrou como o Couchbase Server pode ser utilizado para desenvolver um sistema CRUD simples e flexível, destacando as vantagens de um banco de dados NoSQL em termos de escalabilidade, desempenho e facilidade de uso. O design genérico dos documentos garante que o sistema possa ser adaptado para diversas necessidades, tornando-o uma solução prática para diversas aplicações.

4.8. Repositório Git com o código fonte

Segue abaixo o link para o repositório Git com o código fonte do CRUD:

- https://github.com/gustavolauritzen/CRUDCouchbaseServer

5. Definição de esquemas no Couchbase

O Couchbase é um banco de dados NoSQL que adota uma abordagem flexível em relação a esquemas de dados. Em contraste com bancos de dados relacionais que requerem esquemas predefinidos, o Couchbase permite a inserção de dados em formato JSON sem uma estrutura rígida de esquema. A seguir, exploraremos como o Couchbase trabalha com a especificação e imposição de esquemas de dados, bem como a implementação de relacionamentos entre os dados.

5.1. Especificação e Imposição de Esquemas de Dados

No Couchbase, os dados são armazenados em documentos JSON, o que proporciona uma grande flexibilidade em termos de estrutura de dados. Isso significa que diferentes documentos dentro de um mesmo bucket podem ter estruturas variadas. No entanto, existem formas de impor esquemas e validar dados:

- **Esquemas Flexíveis:** O Couchbase não impõe um esquema rígido. Cada documento pode ter sua própria estrutura, permitindo fácil evolução do esquema ao longo do tempo.
- **Validação de Esquemas:** Embora o Couchbase não imponha esquemas por padrão, é possível implementar a validação de esquemas na aplicação ou utilizar bibliotecas externas para garantir que os documentos sigam uma estrutura específica antes de serem inseridos no banco de dados.
- **Views e Indexes:** A criação de views e indexes pode auxiliar na imposição de certa estrutura ao permitir consultas que esperam determinados campos e tipos de dados nos documentos.

5.2. Relacionamentos entre Dados

Embora o Couchbase seja um banco de dados NoSQL, ele suporta a implementação de relacionamentos entre dados de maneira similar à ideia de chaves primárias e estrangeiras nos bancos de dados relacionais:

- **Referências Diretas:** Relacionamentos podem ser implementados utilizando referências diretas entre documentos. Por exemplo, um documento pode conter um campo que armazena o ID de outro documento, criando um relacionamento entre eles.
- **Documentos Aninhados:** Outra abordagem comum é aninhar documentos dentro de outros documentos. Por exemplo, um documento de ”pedido” pode conter uma lista de itens diretamente aninhada nele.
- **Consultas N1QL:** O Couchbase oferece N1QL, uma linguagem de consulta similar ao SQL, que permite realizar junções (joins) entre documentos, facilitando a consulta de dados relacionados de maneira eficiente.

5.2.1. Exemplo de Referência Direta

```
{
  "type": "order",
  "order_id": "12345",
  "customer_id": "cust_6789",
  "items": [
    {"product_id": "prod_1", "quantity": 2},
    {"product_id": "prod_2", "quantity": 1}
  ]
}
```

Neste exemplo, o documento de ”pedido” refere-se ao ”customer_id” e ”product_id” de outros documentos, estabelecendo um relacionamento entre os dados.

5.2.2. Exemplo de Documentos Aninhados

```
{
  "type": "customer",
  "customer_id": "cust_6789",
  "name": "John Doe",
  "orders": [
    {
      "order_id": "12345",
      "items": [
        {"product_id": "prod_1", "quantity": 2},
        {"product_id": "prod_2", "quantity": 1}
      ]
    }
  ]
```

}

Aqui, os documentos de ”pedido” são aninhados dentro do documento de ”cliente”, simplificando a estrutura e o acesso aos dados relacionados.

5.2.3. Exemplo de Consulta N1QL

```
SELECT c.name, o.order_id, o.items
FROM `bucket` c
JOIN `bucket` o ON KEYS c.order_ids
WHERE c.type = "customer" AND o.type = "order";
```

Essa consulta N1QL realiza uma junção entre documentos de clientes e pedidos, permitindo a recuperação de dados relacionados de forma eficiente.

6. Questões Adicionais

6.1. Método de Segurança e Controle de Acesso

O Couchbase implementa várias medidas de segurança e métodos de controle de acesso para proteger os dados e garantir que apenas usuários autorizados possam acessá-los. Entre os principais métodos de segurança e controle de acesso estão:

- **Autenticação:** O Couchbase suporta autenticação baseada em usuários, onde cada usuário deve fornecer credenciais válidas (nome de usuário e senha) para acessar o banco de dados. Além disso, o Couchbase pode ser integrado com provedores de identidade externos, como LDAP e PAM, para gerenciar autenticação centralizada.
- **Autorização:** O Couchbase utiliza um modelo de controle de acesso baseado em papéis (RBAC - Role-Based Access Control). Isso permite definir permissões granulares para diferentes usuários ou grupos de usuários, controlando quais operações eles podem realizar (por exemplo, leitura, gravação, gerenciamento de índices) e em quais buckets ou coleções.
- **Criptografia:** O Couchbase oferece suporte à criptografia de dados em repouso e em trânsito. A criptografia de dados em repouso garante que os dados armazenados no disco sejam protegidos, enquanto a criptografia de dados em trânsito protege os dados enquanto eles são transferidos entre clientes e servidores Couchbase, usando protocolos como TLS/SSL.
- **Auditoria:** O Couchbase inclui funcionalidades de auditoria que permitem registrar eventos de segurança significativos, como tentativas de login, alterações nas configurações de segurança e operações de acesso aos dados. Essas auditorias ajudam a monitorar atividades suspeitas e a cumprir com requisitos de conformidade.

6.2. Suporte a Transações

O Couchbase oferece suporte a transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), garantindo que as operações de banco de dados sejam realizadas de maneira confiável e previsível. As principais características do suporte a transações no Couchbase incluem:

- **Atomicidade:** As transações no Couchbase são atômicas, o que significa que todas as operações dentro de uma transação são concluídas com sucesso ou nenhuma delas é aplicada. Isso garante que o banco de dados nunca fique em um estado inconsistente.
- **Consistência:** O Couchbase garante que, após a conclusão de uma transação, o estado do banco de dados estará consistente e todas as restrições e regras de integridade serão respeitadas.
- **Isolamento:** As transações no Couchbase são isoladas umas das outras, garantindo que as operações em uma transação não interfiram nas operações de outra transação. Isso é essencial para evitar problemas como condições de corrida e leituras sujas.
- **Durabilidade:** O Couchbase assegura que, uma vez que uma transação é confirmada, as alterações feitas são permanentemente armazenadas e persistirão mesmo em caso de falhas no sistema. A durabilidade é alcançada por meio de técnicas como gravação em memória e replicação em clusters.

Referências

COUCHBASE. *Consistency Models: Couchbase vs. CockroachDB*. 2024. https://www.couchbase.com/blog/consistency-models-couchbase-vs-cockroachdb/. Acessado em: 30 jun. 2024.

COUCHBASE. *Couchbase Documentation - Getting Started*. 2024. https://docs.couchbase.com/server/current/getting-started/installation.html. Acessado em: 9 jun. 2024.

COUCHBASE. *Couchbase Documentation Home*. 2024. https://docs.couchbase.com/home/index.html. Acessado em: 29 jun. 2024.

COUCHBASE. *Couchbase Transactions*. 2024. https://docs.couchbase.com/server/current/learn/data/transactions.html. Acessado em: 23 jun. 2024.

COUCHBASE. *Run Your First SQL++ Query*. 2024. https://docs.couchbase.com/server/current/learn/try-cbq.html. Acessado em: 16 jun. 2024.

COUCHBASE. *Security Overview*. 2024. https://docs.couchbase.com/server/current/manage/manage-security/security-overview.html. Acessado em: 16 jun. 2024.

COUCHBASE. *Site oficial do Couchbase*. 2024. Acessado em: 8 jun. 2024. Disponível em: <https://www.couchbase.com>.

COUCHBASE, I. *Install — Couchbase Docs*. 2024. https://docs.couchbase.com/server/current/install/install-intro.html. Acessado em: 29 jun. 2024.

FERNANDO. *Por dentro do banco de dados NoSQL Couchbase*. 2016. https://www.devmedia.com.br/por-dentro-do-banco-de-dados-nosql-couchbase/34408. Acessado em: 5 jun. 2024.