

**Bootcamp: Arquiteto(a) de Dados**

**Documento de arquitetura e reposítorio projeto e-commerce Amazonas LTDA**

**Autor**: Walter Gabriel Ferreira Gonzaga

**Professor**: João Paulo Faria

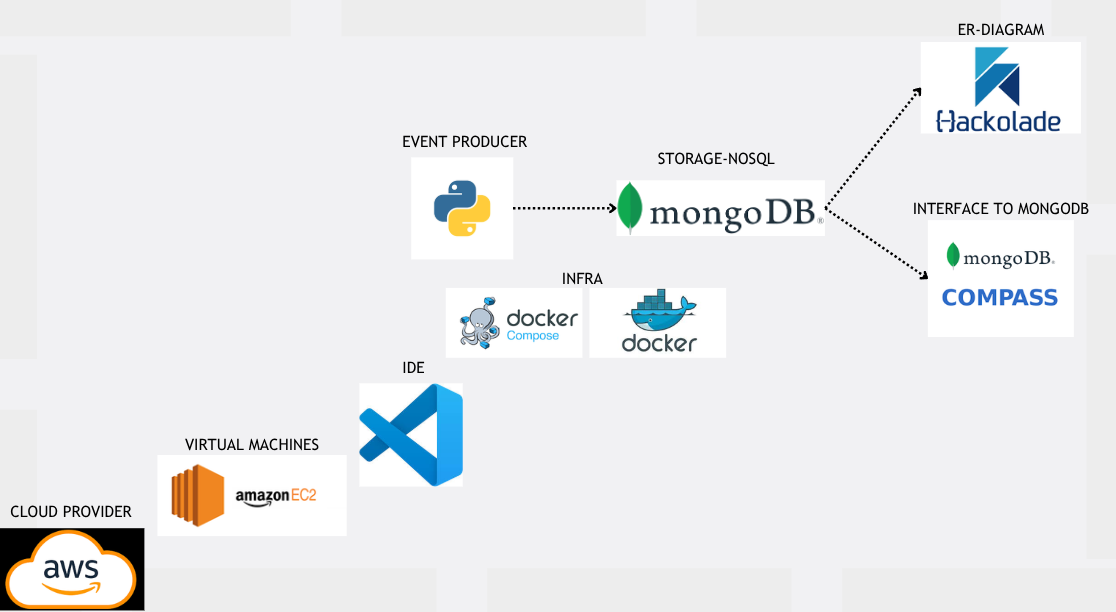
**Módulo**: Desafio Final

**Tecnologias:**

hackolade , Mongodb, Mongodb Compass, Docker, docker compose,

python, AWS

**Github projeto**: https://github.com/Walterbiel/Amazonas-XP-Arquiteturadedados



# ****Sumário****

1. **Introdução**  
   1.1. Contexto e Objetivos  
   1.2. Descrição do Sistema
2. **Estrutura de Dados Proposta**  
   2.1. Modelagem de Dados Não-Relacional  
   2.2. Coleções Principais  
   2.3. Esquema Desnormalizado
3. **Plano de Escalabilidade**  
   3.1. Estratégias de Sharding e Replicação  
   3.2. Infraestrutura com Docker e AWS  
   3.3. Particionamento e Replicação  
   3.4. Crescimento de Dados e Alta Concorrência
4. **Implementação Utilizando AWS Atlas ou DynamoDB**  
   4.1. MongoDB Atlas  
   4.2. DynamoDB
5. **Modelo hackolade**
6. **Explicação projeto prático**

# ****Arquitetura de Dados para o E-commerce "Amazonas"****

## ****1. Descrição do Sistema****

### ****Contexto e Objetivos****

A "Amazonas", uma loja tradicional que vende uma vasta gama de produtos, decidiu expandir para o mercado digital. O objetivo é criar um e-commerce robusto, capaz de lidar com o crescimento rápido e exponencial de clientes e transações. Para garantir alta performance, disponibilidade, escalabilidade e elasticidade, a loja escolheu adotar uma arquitetura de dados moderna utilizando bancos de dados não-relacionais, como o MongoDB, e serviços de cloud computing da AWS.

A principal missão é garantir que a aplicação de e-commerce seja capaz de escalar horizontalmente, utilizando a infraestrutura de **Docker** e a **AWS** para criar clusters de servidores que podem variar de 3 a 10 computadores durante os períodos de pico, garantindo assim alta disponibilidade e escalabilidade do sistema.

## ****2. Estrutura de Dados Proposta****

### ****Modelagem de Dados Não-Relacional****

O sistema será baseado em um banco de dados **NoSQL**, adotando o MongoDB como tecnologia de banco de dados. As coleções serão desnormalizadas para otimizar a performance e evitar o uso de **joins**, o que é crucial para sistemas distribuídos e de alta performance.

### ****Coleções Principais****

Serão projetadas as seguintes coleções principais para atender ao contexto do e-commerce:

1. **Clientes**:  
   Armazenará os dados dos clientes, incluindo:
   * Atributos: cliente\_id, nome, email, endereco, data\_nascimento, preferencias.
   * Relacionamentos: A coleção **Pedidos** terá uma chave estrangeira cliente\_id para relacionar um pedido com o cliente.
2. **Produtos**:  
   Informações sobre os produtos à venda.
   * Atributos: produto\_id, nome, descricao, categoria, preco, estoque.
   * Relacionamentos: Não há um relacionamento direto, mas cada **Pedido** pode conter múltiplos produtos.
3. **Pedidos**:  
   Detalhes dos pedidos feitos pelos clientes.
   * Atributos: pedido\_id, cliente\_id, data, status, total.
   * Relacionamentos: Contém múltiplos **Itens do Pedido**.
4. **Itens do Pedido**:  
   Detalhes sobre os produtos comprados em cada pedido.
   * Atributos: item\_id, pedido\_id, produto\_id, quantidade, preco\_unitario.
   * Relacionamentos: Relacionado a **Produtos** e **Pedidos**.
5. **Carrinho**:  
   Armazenará os itens temporários dos clientes antes da finalização do pedido.
   * Atributos: carrinho\_id, cliente\_id, itens (array de itens temporários), total.
   * Relacionamentos: Cada cliente pode ter apenas um **Carrinho**.

### ****Esquema Desnormalizado****

A escolha de esquemas desnormalizados facilita a escalabilidade, pois evita a necessidade de operações complexas de agregação e joins entre as coleções. Por exemplo, o valor total do pedido será armazenado diretamente na coleção **Pedidos**, o que elimina a necessidade de calcular o total toda vez que um pedido for acessado.

A estrutura será desenhada na ferramenta **Hackolade** com o template do MongoDB, e os diagramas serão incorporados no documento final.

## ****3. Plano de Escalabilidade****

### ****Estratégias de Sharding e Replicação****

A escalabilidade horizontal será implementada utilizando **sharding** e **replicação** no MongoDB, e a infraestrutura será provisionada utilizando **Docker** e **AWS** para garantir a escalabilidade do sistema.

#### ****Sharding****

O sharding será aplicado nas coleções **Pedidos** e **Itens do Pedido**, distribuindo os dados entre os servidores de acordo com a chave de sharding, que pode ser o pedido\_id ou cliente\_id. Com o sharding, os dados serão distribuídos de maneira eficiente entre os nós do MongoDB, o que ajuda a escalar a aplicação conforme o crescimento do número de transações.

#### ****Replicação****

A replicação será configurada nas coleções críticas, como **Produtos** e **Clientes**, para garantir alta disponibilidade. Usando a replicação do MongoDB, os dados serão copiados para múltiplos servidores, permitindo que, em caso de falha de um servidor, o sistema continue funcionando com dados consistentes.

### ****Infraestrutura com Docker e AWS****

A infraestrutura será composta por um cluster de **3 a 10 máquinas** utilizando **AWS EC2** para os servidores, com contêineres Docker executando o MongoDB. O uso de Docker facilita o provisionamento, a escalabilidade e a automação do gerenciamento de infraestrutura.

* **Docker**: Cada instância do MongoDB será empacotada em um contêiner Docker. Isso facilita a criação de ambientes consistentes e a replicação de containers conforme a demanda. Durante períodos de alta demanda, novos contêineres serão iniciados automaticamente, utilizando a elasticidade da AWS.
* **AWS EC2**: Os servidores EC2 serão provisionados de forma dinâmica. Durante os períodos de pico, o número de instâncias EC2 será aumentado automaticamente através de **Auto Scaling Groups**, o que garante que a aplicação tenha capacidade suficiente para lidar com o tráfego elevado.

### ****Particionamento e Replicação****

* **Particionar**: As coleções **Pedidos** e **Itens do Pedido** serão particionadas de acordo com o cliente\_id ou pedido\_id. Isso permitirá que as consultas sejam distribuídas entre diferentes servidores, aumentando a performance.
* **Replicar**: As coleções **Clientes** e **Produtos** serão replicadas, garantindo que os dados críticos permaneçam disponíveis em caso de falhas nos servidores.

### ****Crescimento de Dados e Alta Concorrência****

* **Crescimento de Dados**: À medida que a loja cresce, novos dados serão particionados em servidores adicionais. O uso de sharding garantirá que a carga de trabalho seja distribuída eficientemente, evitando a sobrecarga de qualquer nó individual.
* **Alta Concorrência**: O uso de réplicas e cache otimizado garantirá que o sistema consiga suportar um alto número de acessos simultâneos, especialmente durante períodos de pico, como promoções e datas comemorativas.

## ****4. Implementação Utilizando AWS Atlas ou DynamoDB****

Para implementação na nuvem, utilizaremos o **MongoDB Atlas** ou o **DynamoDB** da AWS.

### ****MongoDB Atlas****

O MongoDB Atlas será utilizado para gerenciar o cluster MongoDB na nuvem, facilitando a configuração de sharding, replicação e backup. O Atlas oferece escalabilidade automática e a capacidade de distribuir os dados entre regiões diferentes, garantindo alta disponibilidade.

### ****DynamoDB****

Se optar por usar o DynamoDB, a configuração de tabelas será feita com base nas chaves de partição, e a escalabilidade será gerenciada automaticamente pela AWS, sem a necessidade de gerenciar servidores manualmente.

## ****5. Modelo Entidade Relacionamento****

## ****6. Projeto Prático****

****Requisitos: Ter o docker e python instalados.****

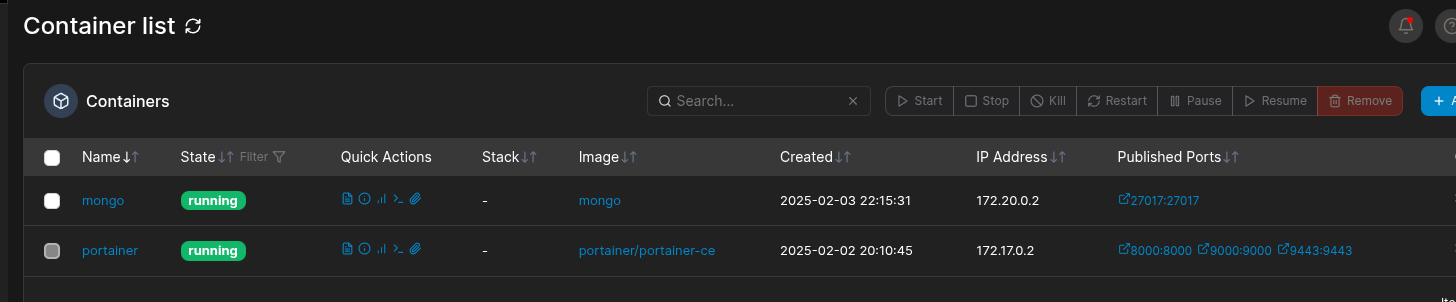
****Github:**** [**https://github.com/Walterbiel/Amazonas-XP-Arquiteturadedados**](https://github.com/Walterbiel/Amazonas-XP-Arquiteturadedados)

****------------------------------------------------------------------------------------------------------  
Digite no bash os sequintes comandos para criar uma network, um volume para persistencia dos dados e o container com a imgaem do mongodb:**$ docker volume create vol1**

$ docker network create net1

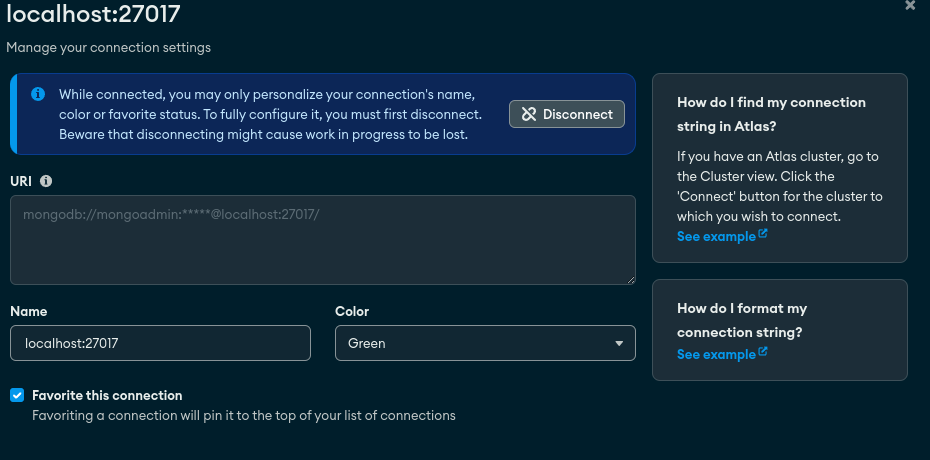
$ docker run -d --network net1 -h mongo --name mongo -e MONGO\_INITDB\_ROOT\_USERNAME=mongoadmin -e MONGO\_INITDB\_ROOT\_PASSWORD=secret -p 27017:27017 -v vol1/data/db mongo

****Assim seu container será criado:****

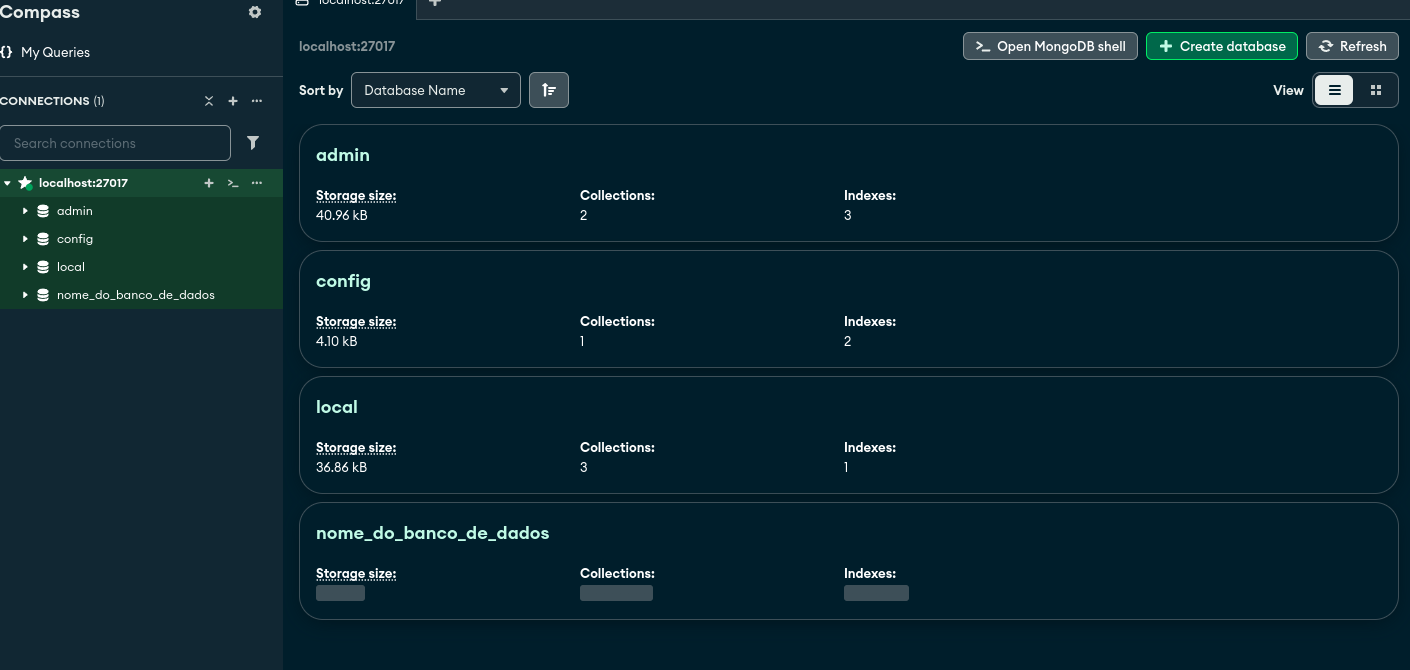


****Baixo o mongodb compass para visualizar em interface gráfica os bancos, coleções e documentos:**** [**https://www.mongodb.com/products/tools/compass**](https://www.mongodb.com/products/tools/compass)

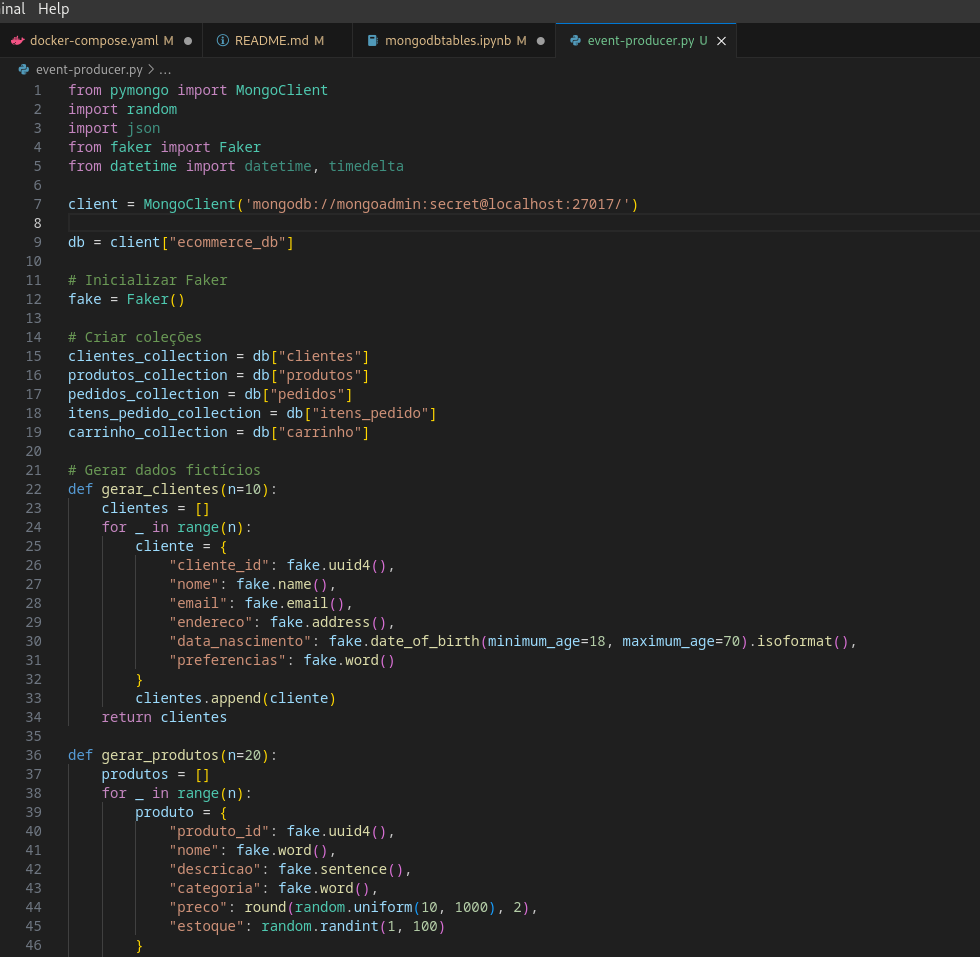
****Agora é encessário conectar o mangodb que está na porta 27017 como configurado no docker, no mongodb compass: mongodb://usuario:senha@localhost:27017****



****Agora é possível visualizar o banco de dados:****



Com nosso arquivo “event-producer.py” vamos gerar dados fake e aleatórios para inputar no banco de dados NoSQL:



Podemos ver as coleções criadas e os documentos inseridos pelo mongodb compass:

