📖Capítulo 7 – DynamoDB

Dia 3 – Building an Internet of Things (IoT) System Around DynamoDB

No terceiro dia com o **DynamoDB**, vamos ver como ele pode ser usado como base para arquiteturas de **Internet das Coisas (IoT)**. Nesse cenário, milhões de dispositivos enviam dados continuamente, exigindo **baixa latência, alta escalabilidade e integração em tempo real**.

🕐IoT e o DynamoDB

O DynamoDB é especialmente adequado para IoT porque: - Aceita **grande volume de escritas** por segundo. - Oferece **latência de milissegundos**. - Integra-se nativamente com serviços da AWS (IoT Core, Kinesis, Lambda, S3). - Permite consultas rápidas com **chaves bem modeladas**.

🛠️Arquitetura Típica de IoT

Fluxo de dados em um sistema IoT com DynamoDB:

```
Dispositivo IoT → AWS IoT Core → DynamoDB → Streams → Lambda/Kinesis → Analytics/Dashboards
```

1. **Dispositivos IoT** coletam dados (sensores, veículos, máquinas industriais).
2. **AWS IoT Core** recebe mensagens via MQTT/HTTP.
3. Dados são armazenados no **DynamoDB**.
4. **Streams + Lambda/Kinesis** processam eventos em tempo real.
5. Dados podem ser enviados para **S3, Redshift, OpenSearch** ou dashboards.

📦Modelagem de Tabelas IoT

Uma tabela típica pode usar: - **Partition Key**: `DeviceId` (identificador do dispositivo). - **Sort Key**: `Timestamp` (quando a leitura foi registrada).

📌Exemplo de item:

```
{
  "DeviceId": "Sensor-001",
  "Timestamp": "2025-08-29T10:15:00Z",
  "Temperatura": 27.5,
  "Umidade": 60,
```

```
  "Localizacao": { "Lat": -23.55, "Long": -46.63 }
}
```

🔔 Essa modelagem permite consultar rapidamente todas as leituras de um dispositivo em um intervalo de tempo.

📊Consultas comuns

- Últimas leituras de um dispositivo:

```
aws dynamodb query
  --table-name Sensores
  --key-condition-expression "DeviceId = :d AND Timestamp > :t"
  --expression-attribute-values '{":d":{"S":"Sensor-001"}, ":t":
{"S":"2025-08-29T00:00:00Z"}}'
```

- Todos os dispositivos em uma área geográfica → DynamoDB pode armazenar a geolocalização, mas para consultas espaciais avançadas é comum integrar com o **Amazon OpenSearch**.

🕐Processamento em Tempo Real

Com **DynamoDB Streams + Lambda**, podemos: - Detectar anomalias (ex.: temperatura fora da faixa). - Notificar usuários ou sistemas de manutenção. - Enviar dados para análise em **Kinesis Analytics**.

Exemplo de lógica em Lambda (Node.js):

```
exports.handler = async (event) => {
  for (const record of event.Records) {
    if (record.eventName === "INSERT") {
      const newItem = record.dynamodb.NewImage;
      if (parseFloat(newItem.Temperatura.N) > 50) {
        console.log("Alerta! Temperatura muito alta:", newItem);
      }
    }
  }
};
```

⚖️Boas Práticas

- Escolher chaves que permitam **balanceamento de carga**.
- Usar TTL (Time to Live) para expirar dados antigos automaticamente.
- Armazenar apenas dados relevantes no DynamoDB; dados brutos podem ir para o **S3**.
- Integrar com **CloudWatch** para monitorar desempenho.

🔗Resumo do Dia 3

- O DynamoDB suporta arquiteturas IoT massivas.
- Tabelas bem modeladas permitem consultas rápidas por dispositivo e tempo.
- Streams + Lambda processam dados em tempo real.
- Integrações com S3, Redshift e OpenSearch completam o pipeline.

🕑Wrap-Up do Capítulo 7 – DynamoDB

- **Dia 1** → CRUD básico, modelagem de chaves e tabela de pedidos.
- **Dia 2** → Streams e pipelines de dados em tempo real.
- **Dia 3** → Uso em IoT, com milhões de dispositivos enviando eventos.

📌 Conclusão: o DynamoDB é ideal para sistemas que exigem **escala massiva, latência baixa e integração perfeita com a AWS**, desde e-commerce até IoT industrial.