Sistemas Paralelos e Distribuídos

Práticas – Aula 12

Práticas

Design e programação no contexto de sistemas distribuídos

- Intro
- Considerações de design (APIs)
- Características e requisitos (APIs)
- Exemplos de esquemas de SD
- Frameworks e linguagens
- Atividade
- Exemplo básico para key-value API
- Key-value API: Problemas
- Conclusões

Intro

- Conceitos de SD (e seu desenvolvimento) caminharam juntamente com o desenvolvimento da Internet. Primórdios no anos 60 e 70.

- A especificação CORBA é uma referência clássica para guiar o desenho e desenvolvimento de SD
https://docs.oracle.com/cd/E13161_01/tuxedo/docs10gr3/tech_articles/CORBA.html#:~:text=CORBA is based on the distributed object,and viewing the balance in the accounts.

- Arquiteturas reconhecidas de SD:
  - Client-Server
  - Peer-to-Peer
  - 3 Tier (presentation-app-data)
  - Microservices
  - Service Oriented
  - Event-Driven
  - Edge assisted

- Exemplos de tecnologias de SD:
  - Banco de dados distribuídos
  - Brokers (fila de mensagens, eventos, etc)
  - Virtual machine
  - Containers
  - Cloud computing

- Desenho e desenvolvimento de programas depende estritamente do modelo de arquitetura adotado, bem como das tecnologias utilizadas.

Considerações de design

- Ao fazer o desenho de APIs para SD, considerar qual melhor modelo a ser utilizado:
  - REST (Representational State Transfer)
  - gRPC (Remote Procedure Call)
  - GraphQL (Query Language)

- Documentação (esquemas)

- Versionamento

	REST	gRPC	GraphQL
Formato	JSON	Protobuf	JSON
Protocolo	HTTP/1.1	HTTP/2	HTTP
Performance	razoável	alta	Alta (conforme complexidade)
Flexibilidade	baixa	razoável	alta
Cenário	API pública	Interno/micro	Dinâmico

Características e requisitos

- Rede: latência, timeout e retry, distribuição de carga, etc.

- Tratamento de dados: formatação, validação, compressão, consistência (eventual ou imediata) e caching.

- Segurança: autenticação, criptogragia, limites de taxa, etc.

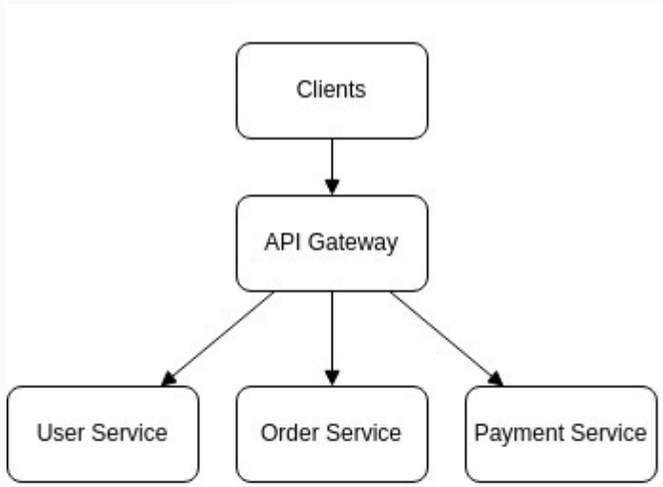
- Erro e depuração: Uso de padrões, logs, sincronismo, etc.

- Escalabilidade e performance: aceleradores, particionamento, clustering, replicação, etc.

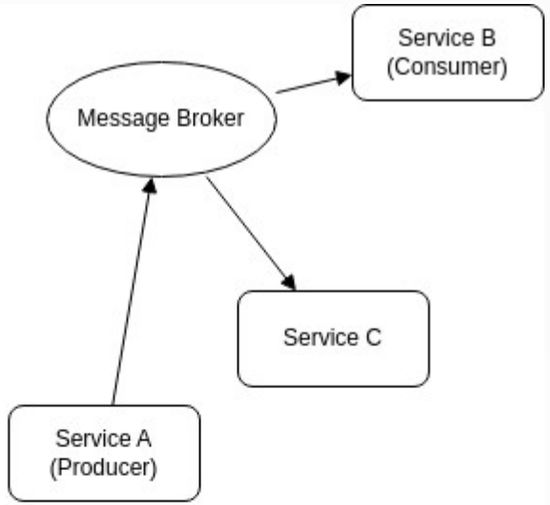
- Implantação e building: técnicas de deployment (blue-green, canary, etc), documentação de procedimentos, metodologia agile, etc.

- Testes, monitorização e recuperação: observabilidade consolidada, load testing, alarmes e procedimentos de recuperação, precedimentos de testes e QA, SLO, SLA, etc.

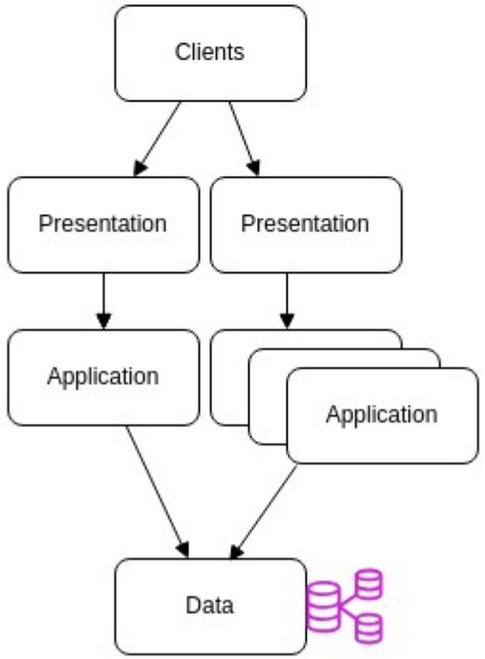
Exemplos de esquemas de SD



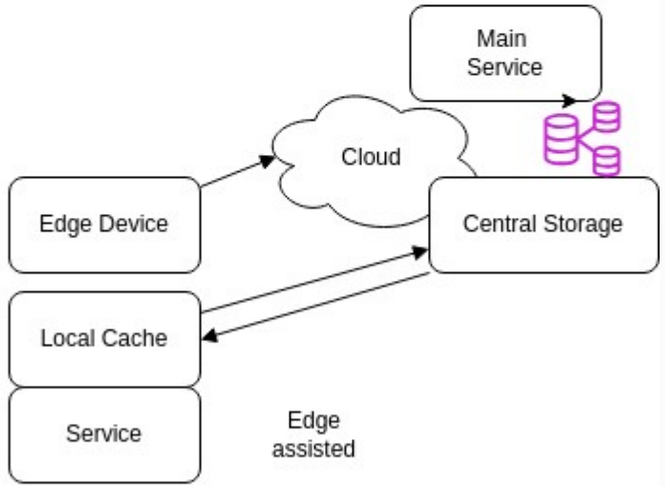
Microservices



Event-Driven



3-Tier



Edge
assisted

Frameworks e linguagens

- Python: Flask, FastAPI, Django, etc.

- JS e TypeScript: Express.js, Nest.js, Fastify, etc.

- Java: Spring Boot, Ktor, Micronaut, etc.

- Go: Gin, Echo, Fiber, etc.

- Elixir: Phoenix, Trot, Plug, etc.

Atividade

- Testar 2 ou 3 frameworks diferentes com simples “web hello world” testes.
  Eleger qual o mais adequado ou preferido.

Atividade

- Golang

```
package main

import "github.com/gin-gonic/gin"

func main() {
	r := gin.Default()

	r.GET("/hello", func(c *gin.Context) {
		c.String(200, "Hello, World!")
	})

	r.Run(":3000")
}
```

Atividade

- Python

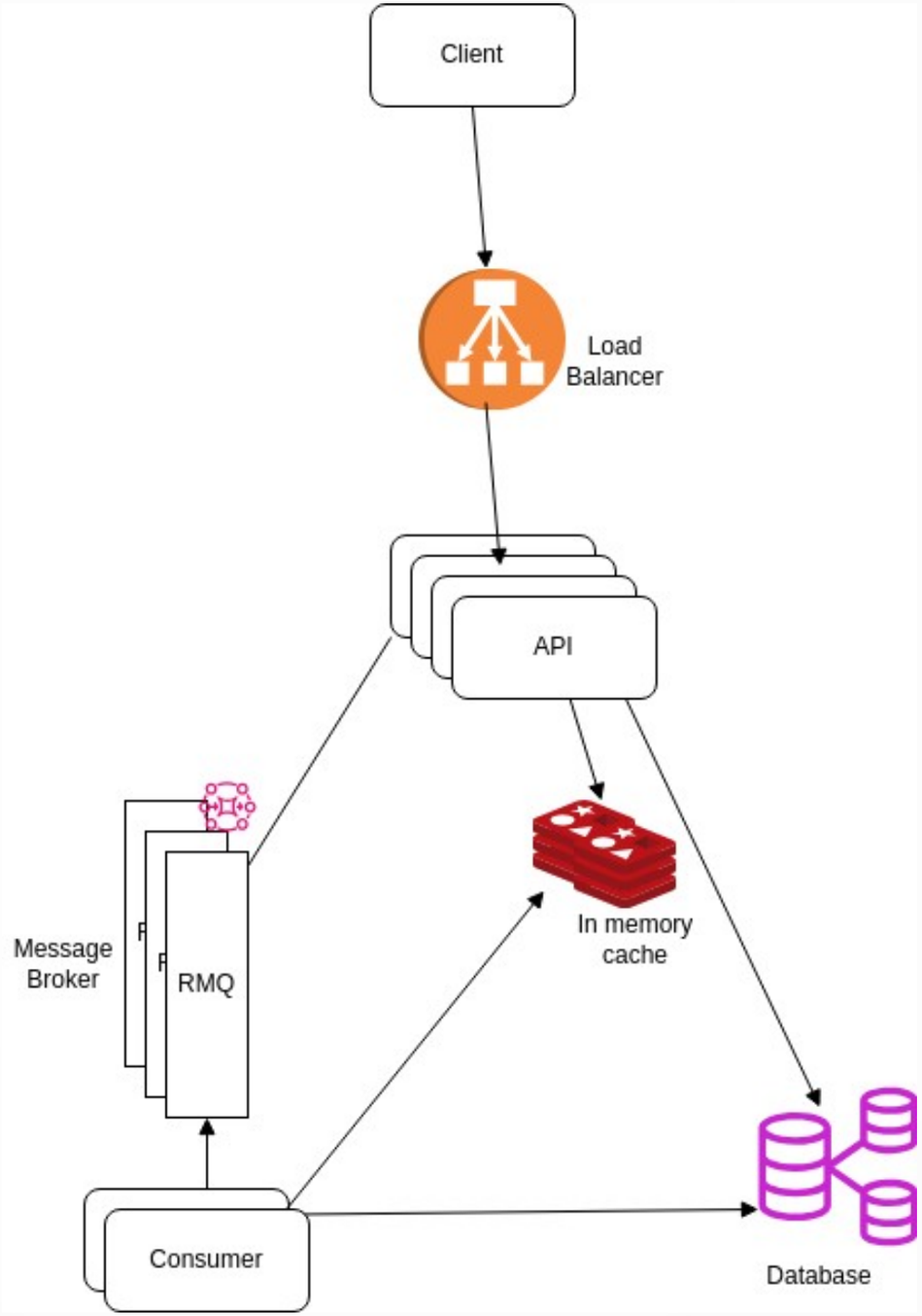
```
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Atividade

- Nodejs

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Exemplo key-value API



Exemplo – API para key-value (Ficheiro compose)

```
version: '3.8'

services:
  api:
    build: ./api
    ports:
      - "3000:3000"
    depends_on:
      - redis
      - postgres
      - rabbitmq
    environment:
      - REDIS_HOST=redis
      - REDIS_PORT=6379
      - POSTGRES_HOST=postgres
      - POSTGRES_PORT=5432
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=appdb

  consumer:
    build: ./api
    command: node consumer.js
    depends_on:
      - postgres
      - rabbitmq
      - api
    environment:
      - REDIS_HOST=redis
      - REDIS_PORT=6379
      - POSTGRES_HOST=postgres
      - POSTGRES_PORT=5432
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=appdb

redis:
  image: redis:7
  ports:
    - "6379:6379"

postgres:
  image: postgres:15
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
    POSTGRES_DB: appdb
  ports:
    - "5432:5432"

rabbitmq:
  image: rabbitmq:3-management
  container_name: golang-rabbitmq-rabbitmq
  ports:
    - "5672:5672"
    - "15672:15672"
  healthcheck:
    test: rabbitmq-diagnostics -q ping
    interval: 30s
    timeout: 30s
    retries: 3
```

Exemplo – API para key-value (API principal)

```
app.get('/', async (req, res) => {
  const parseResult = KeyQuerySchema.safeParse(req.query);
  if (!parseResult.success) {
    return res.status(400).json({
      error: 'Invalid query parameters',
      details: parseResult.error.format(),
    });
  }

  const { key } = parseResult.data;


  try {
    // Try Redis
    const redisVal = await redis.get(key);
    if (redisVal !== null) {
      return res.json({ value: redisVal, source: 'redis' });
    }

    // Fallback to Postgres
    const result = await pgClient.query('SELECT value FROM
kv_store WHERE key = $1', [key]);
    if (result.rows.length > 0) {
      const value = result.rows[0].value;
      // Save to Redis for next time
      await redis.set(key, value);
      return res.json({ value, source: 'postgres' });
    }

    return res.status(404).json({ error: 'Key not found' });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'Internal Server Error' });
  }
});
```

```
app.put('/', async (req, res) => {

  const parseResult = KeyPayloadSchema.safeParse(req.body);
  if (!parseResult.success) {
    return res.status(400).json({ error: 'Invalid payload', details:
parseResult.error.format() });
  }

  const { key_name, key_value } = parseResult.data;
  const payload = { key_name, key_value };

  await mqChannel.sendToQueue('add_key',
Buffer.from(JSON.stringify(payload)));
  return res.status(202).json({ message: 'Queued to
add_key' });
});

app.delete('/', async (req, res) => {

 const parseResult = KeyQuerySchema.safeParse(req.query);
  if (!parseResult.success) {
    return res.status(400).json({
      error: 'Invalid query parameters',
      details: parseResult.error.format(),
    });
  }

  const { key } = parseResult.data;
  const payload = { key };

  await mqChannel.sendToQueue('del_key',
Buffer.from(JSON.stringify(payload)));
  return res.status(202).json({ message: 'Queued to
del_key' });
});
```

Exemplo – API para key-value (Consumer)

```
mqChannel.consume('add_key', async (msg) => {
  if (msg !== null) {
    try {
      const { key_name, key_value } =
JSON.parse(msg.content.toString());

      await pgClient.query(
        'INSERT INTO kv_store (key, value) VALUES ($1, $2) ON
CONFLICT (key) DO UPDATE SET value = $2',
        [key_name, key_value]
      );

      console.log(`     Inserted/Updated key "${key_name}"`);
      mqChannel.ack(msg);
    } catch (err) {
      console.error('     Error handling message:', err.message);
      mqChannel.nack(msg); // Optional: requeue or not
    }
  }
});
```

```
mqChannel.consume('del_key', async (msg) => {
  if (msg) {
    try {
      const { key } = JSON.parse(msg.content.toString());

      await pgClient.query('DELETE FROM kv_store WHERE key
= $1', [key]);

      console.log(`     [del_key] Deleted: ${key}`);
      mqChannel.ack(msg);
    } catch (err) {
      console.error(`     [del_key] Failed: ${err.message}`);
      mqChannel.nack(msg);
    }
  }
});
```

Atividade – API para key-value (Problemas?)

- Quais principais problemas?

- Algo a comentar em relação ao sincronismo e a consistência da informação?

- Forneça soluções de contorno mínimas.

API para key-value (Consistência de Dados)

```
app.get('/', async (req, res) => {
  const parseResult = KeyQuerySchema.safeParse(req.query);
  if (!parseResult.success) {
    return res.status(400).json({
      error: 'Invalid query parameters',
      details: parseResult.error.format(),
    });
  }

  const { key } = parseResult.data;

  try {
    // Try Redis
    const redisVal = await redis.get(key);
    if (redisVal !== null) {
      return res.json({ value: redisVal, source: 'redis' });
    }
    // Fallback to Postgres
    const result = await pgClient.query('SELECT value FROM
kv_store WHERE key = $1', [key]);
    if (result.rows.length > 0) {
      const value = result.rows[0].value;
      // Save to Redis for next time
      await redis.set(key, value);
      return res.json({ value, source: 'postgres' });
    }

    return res.status(404).json({ error: 'Key not found' });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'Internal Server Error' });
  }
});
```

```
app.put('/', async (req, res) => {
  const parseResult = KeyPayloadSchema.safeParse(req.body);
  if (!parseResult.success) {
    return res.status(400).json({ error: 'Invalid payload', details:
parseResult.error.format() });
  }

  const { key_name, key_value } = parseResult.data;
  const payload = { key_name, key_value, timestamp: new
Date().toISOString() };
  await mqChannel.sendToQueue('add_key',
Buffer.from(JSON.stringify(payload)));
  return res.status(202).json({ message: 'Queued to
add_key' });
});

app.delete('/', async (req, res) => {
  const parseResult = KeyDeleteSchema.safeParse(req.query);
  if (!parseResult.success) {
    return res.status(400).json({ error: 'Invalid query
parameters', details: parseResult.error.format() });
  }
  const { key_name } = parseResult.data;
  const payload = { key_name, timestamp: new
Date().toISOString()};

  await mqChannel.sendToQueue('del_key',
Buffer.from(JSON.stringify(payload)));
  return res.status(202).json({ message: 'Queued to
del_key' });
});
app.listen(port, () => {
  console.log(`API listening on port ${port}`);
});
```

API para key-value (Consistência de Dados)

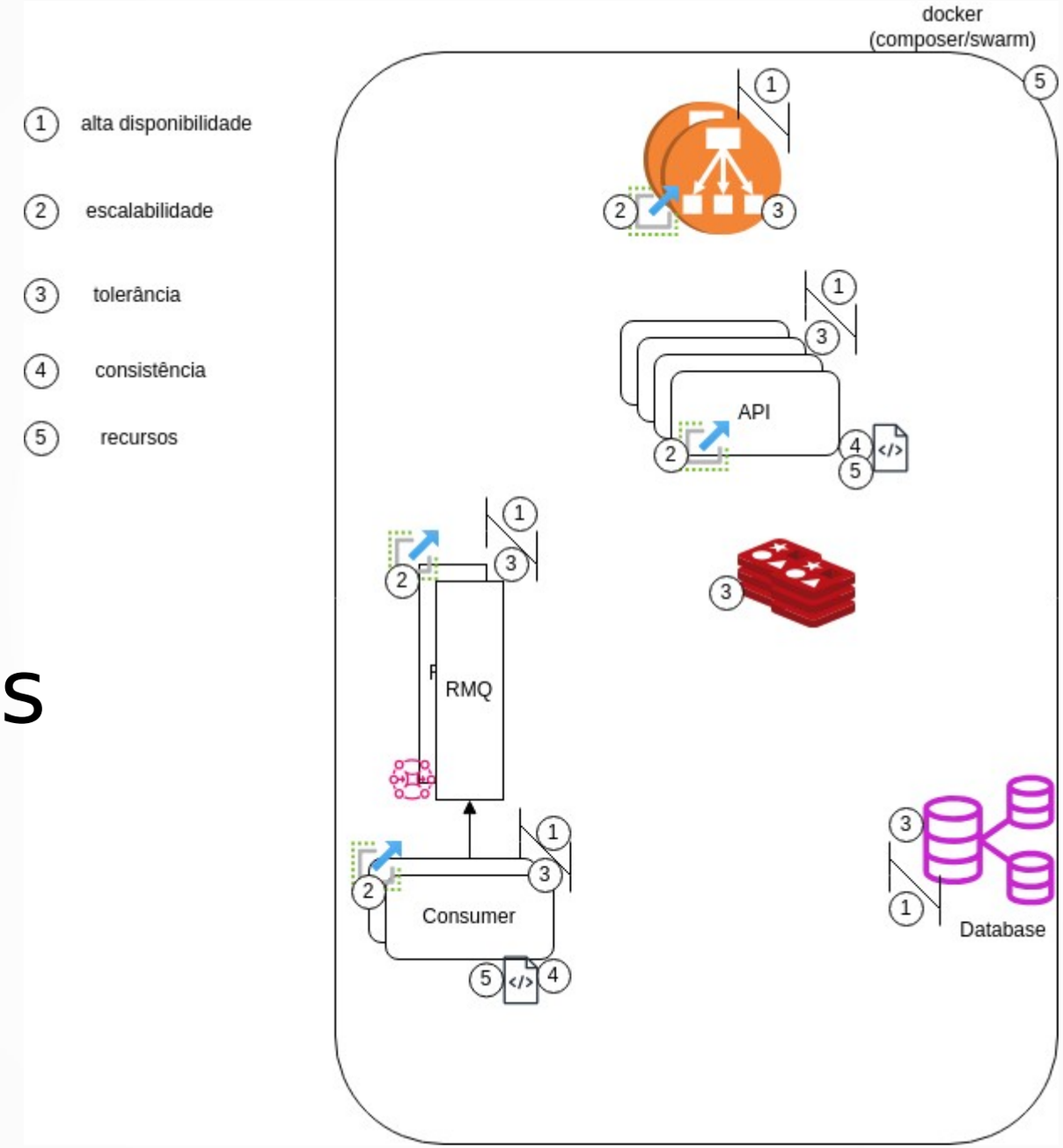
```
mqChannel.consume('add_key', async (msg) => {
  if (!msg) return;

  try {
    const { key_name, key_value, timestamp } =
JSON.parse(msg.content.toString());
    const ts = new Date(timestamp);

    if (!key_name || !key_value || !timestamp) {
      console.warn(`⚠ Invalid add_key message: $
{msg.content.toString()}`);
      mqChannel.nack(msg, false, false);
      return;
    }

    const upt_result = await pgClient.query(
      `INSERT INTO kv_store (key, value, last_updated)
       VALUES ($1, $2, $3)
       ON CONFLICT (key)
       DO UPDATE SET value = $2, last_updated = $3
       WHERE kv_store.last_updated <= $3`,
      [key_name, key_value, ts]
    );

    if (upt_result.rowCount > 0) {
    try {
    // Try Redis
    const redisVal = await redis.get(key_name);
    if (redisVal !== null) {
      await redis.set(key_name, key_value);
     }}catch (err) {
        console.error(`     Error: ${err.message}`);
        }
    }
    console.log(`     [add_key] ${key_name} set to "${key_value}" at $
{timestamp}`);
    mqChannel.ack(msg);
  } catch (err) {
    console.error(`     [add_key] Error: ${err.message}`);
    mqChannel.nack(msg, false, false);
  } });
```

```
mqChannel.consume('del_key', async (msg) => {
  if (!msg) return;

  try {
    const { key_name, timestamp } = JSON.parse(msg.content.toString());
    const ts = new Date(timestamp);

    if (!key_name || !timestamp) {
      console.warn(`⚠ Invalid del_key message: $
{msg.content.toString()}`);
      mqChannel.nack(msg, false, false);
      return;
    }

    const res = await pgClient.query(
      'SELECT last_updated FROM kv_store WHERE key = $1',
      [key_name]
    );

    if (res.rows.length === 0) {
       const retries = msg.properties.headers['x-retry'] || 0;

       if (retries < 3) {
         console.warn(`     [del_key] Key "${key_name}" not found. Retrying...
(attempt ${retries + 1})`);
         mqChannel.nack(msg, false, false); // Don't requeue yet

         // Requeue manually with retry count
         await mqChannel.sendToQueue('del_key',
Buffer.from(msg.content.toString()), {
           headers: { 'x-retry': retries + 1 },
           expiration: 3000, // optional: delay retry
         });

         return;
       } else {
         console.warn(`     [del_key] Key "${key_name}" not found after $
{retries} retries. Dropping.`);
         mqChannel.ack(msg);
         return;
       }
    }  . . .
```

API para key-value (Consistência de Dados)

```
    . . .

    const dbTimestamp = new Date(res.rows[0].last_updated);

    if (dbTimestamp <= ts) {
      await pgClient.query('DELETE FROM kv_store WHERE key = $1',
[key_name]);
      console.log(`     [del_key] Deleted "${key_name}" at ${timestamp}`);
      try {
      // Try Redis
      const redisVal = await redis.get(key_name);
      if (redisVal !== null) {
           await redis.del(key_name);
           }}catch (err) {
           console.error(`     Error: ${err.message}`);
           }

    } else {
      console.log(`     [del_key] Skipped deletion of "${key_name}" — newer
value exists.`);
    }

    mqChannel.ack(msg);
  } catch (err) {
    console.error(`     [del_key] Error: ${err.message}`);
    mqChannel.nack(msg, false, false);
  }
});
```

API para Key-value: Conclusões



Conclusões

- É possível aumentar ainda mais a qualidade e capacidade nos diversos componentes. Um exemplo de tratamento de concorrência para Redis pode ser encontrado no post: https://medium.com/@vishwa.telsang/handle-concurrent-requests-in-distributed-systems-cf5a274116a8

- Náo há arquitetura única para solucionar cada um dos problemas. Ajustá-las, combinando-as e tomando proveito das vantagens das diferentes arquiteturas.

- Linguagens e frameworks adotados podem ter justificações técnicas, mas podem também ser escolhidos mediante preferências ou familiaridade.

- Implementação de formatação, consistência e gestão de recursos nas aplicações.

- Usar capacidades do middleware e rede para alguns desses requisistos (e.g. segurança no escopo de um LB ou API GW).

- Realizar testes de carga e introduzir cenários de falhas.

FIM