Distribuição de Processos e Dados

CitySenseHub - Simulação de Cidade Inteligente com RabbitMQ e gRPC

João Gabriel Borges
João Luca Teixeira Carvalho
Antônio Irineu Filho

link do código

UNIVERSIDADE FEDERAL DO CEARÁ

Objetivo do Projeto

- **Descrição:** Nosso sistema simula uma cidade inteligente com atuadores controláveis e sensores que monitoram o ambiente em tempo real.
- **Atuadores Implementados:** Câmera, Poste, Semáforo.
- **Sensores Implementados:** Sensor de Temperatura, GPS, Sensor de Umidade.

Tecnologias e Bibliotecas

Linguagens: Python, Node.js.

Componentes:

- **Gateway (Python):** Utiliza **Flask** e **Flask-Restful** para a API REST, **PyJWT** para autenticação com JSON Web Tokens, **pika** para a comunicação com o Broker RabbitMQ e **grpcio** para atuar como cliente gRPC.
- **Sensores (Python/Node.js):** Utilizam as bibliotecas nativas `socket` (Python) e `dgram` (Node.js) para a comunicação UDP.

Descoberta de Dispositivos (Multicast UDP)

Objetivo: Permitir que o Gateway encontre os atuadores e, crucialmente, que os dispositivos descubram as informações de conexão do Broker.

Fluxo:

1. O Gateway envia uma mensagem multicast contendo as informações de conexão do Broker (ex: porta `5672` e tópico `sensores`).
2. Os Dispositivos recebem essa mensagem, armazenam os dados do Broker e respondem ao Gateway com suas próprias informações (ID, endereço para gRPC, etc.).

Serviço Web do Gateway (API REST)

- **Framework: Flask**
- **Autenticação:** Utiliza **JSON Web Tokens (JWT)** para proteger os endpoints. O cliente deve primeiro se autenticar no endpoint `/login` para obter um token, que deve ser enviado como `Bearer Token` no cabeçalho `Authorization` das requisições subsequentes.
- **Endpoints Implementados:**
  - `POST /login`: Recebe um JSON com `username` e `password`. Retorna um `token` JWT em caso de sucesso.
  - `GET /protected`: Endpoint de exemplo que valida o token e retorna uma mensagem de boas-vindas.
  - `GET /consultas`: Endpoint protegido que, no futuro, retornaria os estados dos dispositivos. Atualmente, retorna dados de exemplo.

Comunicação dos Sensores (Broker RabbitMQ)

- **Padrão:** Publish/Subscribe, para comunicação assíncrona e desacoplada entre os Sensores e o Gateway.
- **Esquema Utilizado:** Os sensores atuam como **Publishers**, enviando suas leituras para o exchange `smart_city_exchange` (do tipo `topic`). O Gateway atua como **Subscriber**, consumindo as mensagens de uma fila inscrita no tópico `sensores.#`.
- **Formato da Mensagem:** As mensagens são enviadas no formato **JSON**, contendo o ID do dispositivo, timestamp, o valor da leitura e sua unidade. Ex: `{"deviceId": "sensor-01", "value": 25.5, ...}`.

Controle dos Atuadores (gRPC)

- **Objetivo:** O Gateway invoca métodos remotos nos atuadores para alterar ou consultar seus estados.
- **Definição do Serviço (`messages.proto`)**

```
service SmartCity {
  // Altera ou consulta o estado de um atuador
  rpc ChangeState (ChangeStateRequest) returns (Query);
  rpc StateDevice (StateDeviceRequest) returns (Query);

// Altera a configuração de um sensor
rpc ChangeTime (ChangeTimeRequest) returns (Time);
 }
```

Controle dos Atuadores (gRPC)

- **Objetivo:** O Gateway invoca métodos remotos nos atuadores para alterar ou consultar seus estados.
- **Formato das Mensagens gRPC (`messages.proto`):**

```
// Mensagem de requisição para mudar o estado
message ChangeStateRequest {
  string device_id = 1;
  Command command = 2; // Contém o estado 'ligado/desligado'
}

// Mensagem de requisição para consultar o estado
message StateDeviceRequest {
  string device_id = 1;
}

// Mensagem de resposta para ambas as requisições
message Query {
  bool status = 1; // Retorna o estado atual
}
```

Interface do Cliente

- **Tipo:** Linha de Comando (CLI).
- **Funcionalidades:**
  - Interage com o Gateway exclusivamente via **API REST**.
  - Primeiro, realiza a autenticação no endpoint `/login` para obter um token JWT.
  - Envia o token em todas as requisições subsequentes para listar dispositivos, consultar estados ou enviar comandos de ligar/desligar.

Implementação - Gateway

- **Arquitetura:** Implementado como uma classe `Gateway` que utiliza `threading` para executar concorrentemente um servidor de API REST com Flask-Restful, um consumidor RabbitMQ e um servidor de descoberta Multicast.
- **Gerenciamento de Dispositivos:** A lista de dispositivos é *thread-safe*, utilizando `threading.Lock` para evitar race conditions.
- **Robustez (Heartbeat):** O Gateway implementa um sistema de *heartbeat*. Cada anúncio de um dispositivo reseta um timer. Um serviço de "limpeza" (`pruning`) remove automaticamente da lista qualquer dispositivo que fique mais de 20 segundos sem se comunicar, garantindo que a lista de dispositivos online esteja sempre atualizada.

Implementação - Dispositivos

- **Atuadores:** Implementam um servidor gRPC que expõe os métodos `ChangeState` e `StateDevice`, permitindo que o Gateway altere e consulte seus estados de forma remota e segura.
- **Sensores:** Atuam como publicadores RabbitMQ. Após obterem os dados do Broker via descoberta, entram em um loop para enviar suas leituras.
- **Configuração Remota:** Os sensores também implementam o método gRPC `ChangeTime`, permitindo que o Gateway ajuste dinamicamente a frequência de envio de dados de cada sensor.
- **Encerramento Controlado:** Todos os dispositivos implementam uma rotina de encerramento (`parar()`), garantindo que as threads e conexões sejam finalizadas de forma organizada ao receber um sinal de interrupção.

Fluxo de Comunicação

1. O **Cliente** seleciona a opção "Mudar configuração de tempo" para um sensor.
2. Ele envia uma requisição **POST** para o endpoint `/dispositivos/<id>` da API REST do Gateway, contendo o novo intervalo de tempo no corpo da requisição.
3. O **Gateway** recebe a requisição, valida o token JWT, e faz uma **chamada gRPC** para o método `ChangeTime` do sensor correspondente.
4. O **Sensor** (servidor gRPC) recebe a chamada, atualiza sua variável interna `config_time_send`, e retorna uma mensagem de confirmação.
5. O **Gateway** recebe a resposta gRPC e a repassa como uma resposta HTTP de sucesso para o **Cliente**.