**Sistema de Recarga de Carros Elétricos Inteligente (Nuvem):**

**Arquitetura do Sistema da Nuvem:**

O **sistema** e **servidor** do container “Nuvem”, implementado no arquivo “Cloud.py” através do “socket” em Python, é responsável por receber requisições dos clientes (veículos ou postos de recarga), armazenar informações sobre postos disponíveis e seus pontos de carregamento e reservas em um banco de dados em arquivo JSON, e processar ações como: criar ou apagar uma reserva, cadastrar ou excluir um posto, descobrir o posto mais próximo do veículo, através de um plano cartesiano, entre outros.

O servidor utiliza **multithreading** para lidar com a concorrência entre múltiplas conexões simultâneas, evitando que o socket deixe outras requisições esperando, enquanto uma já está sendo atendida. Dessa maneira, cada nova requisição feita ao servidor pelo cliente, gera um novo thread para atendê-la.

O **multithreading** foi escolhido pois é mais simples de implementar e suficiente para o sistema.

O servidor aceita requisições de qualquer dispositivo na rede local, mas primeiro é necessário criar uma rede personalizada no Docker através do seguinte comando:

*“docker network create recharging_manager”*

E então, conectar o cliente ao servidor através dessa rede e do host “cloud_server” na porta “64352”.

A imagem do servidor pode ser executada através do docker-compose, acessando o diretório “app/Cloud” e executando o seguinte comando através do terminal:

*“docker compose up --build"*

**Estrutura dos Arquivos da Nuvem:**

**1. Cloud.py:**

Este é o **servidor principal** do sistema. Ele é responsável por iniciar o socket da Nuvem, ativar o modo de escuta do socket, criar threads para novas solicitações dos clientes, fazer a manipulação de dados necessárias no banco de dados, responder aos clientes e finalizar a conexão com os clientes.

O servidor escuta qualquer dispositivo conectado na mesma rede local, que se comunica com a porta "64352".

Ao receber uma requisição, interpreta o conteúdo em JSON, analisa qual ação é necessária, através de uma chave/campo de ação para cada solicitação do cliente, e executa a ação correspondente.

**Campos/Chaves no JSON enviado pelo Cliente e suas ações no servidor:**

- **newChargingStation**:
  - o cliente do posto de recarga solicita a criação e salvamento de suas informações no banco de dados do servidor.
  - o servidor responde com o ID cadastrado para o posto de recarga no banco de dados.
- **updateChargingStation**:
  - o cliente do posto de recarga solicita a atualização de sua localização no banco de dados do servidor.
  - o servidor responde com a string: "Sucesso".

- **deleteChargingStation**:
    - o cliente do posto de recarga solicita a sua exclusão do banco de dados do servidor.
    - o servidor responde com a string: "Sucesso".
- **newChargingPoint**:
    - o cliente do posto de recarga solicita a criação e salvamento de um dos seus pontos de carregamento no banco de dados do servidor.
    - o servidor responde com o ID cadastrado para o ponto de carregamento no banco de dados.
- **updateChargingPoint**:
    - o cliente do posto de recarga solicita a atualização das informações de um dos seus pontos de carregamento no banco de dados do servidor.
    - o servidor responde com a string: "Sucesso".
- **deleteChargingPoint**:
    - o cliente do posto de recarga solicita a exclusão de um dos seus pontos de carregamento no banco de dados do servidor.
    - o servidor responde com a string: "Sucesso".
- **receiveAllChargingPoints**:
    - o cliente do posto de recarga solicita uma lista de todos os seus pontos de carregamento e suas respectivas informações do banco de dados.
    - o servidor responde com um JSON com todos os pontos de carregamento.
- **receiveAllReservations**:
    - o cliente do posto de recarga solicita uma lista com informações de todas as reservas, para todos os seus pontos de carregamento, no banco de dados.
    - o servidor responde com um JSON com todas as reservas agendadas.
- **scheduleReservation**:
    - o cliente do veículo solicita o agendamento de uma reserva, pois a bateria está no nível crítico.
    - o servidor responde com as informações do agendamento para um ponto de carregamento com a fila vazia, ou com o menor tempo de espera, no posto de recarga mais próximo.
- **findReservation**:
    - o cliente do veículo solicita as informações de sua reserva agendada, se existir alguma.
    - o servidor responde com um JSON com as informações da reserva, armazenados no banco de dados.
- **deleteReservation**:
    - o cliente do veículo solicita a exclusão de uma reserva agendada.
    - o servidor responde com a string: "Sucesso".

Cada ação é tratada por meio de **classes auxiliares**, que manipulam os dados nas listas e os atualizam no banco de dados em JSON, como: ReservationsFile (Para Reservas), ChargingPointsFile (Para Pontos de Carregamento) e ChargingStationsFile (Para Posto de Recarga).

**Além disso, caso ocorra um dos seguintes erros na solicitação, o servidor irá responder com a string "None":**

- Não existem postos de recarga ou pontos de carregamento cadastrados;
- Posto de recarga não encontrado, para atualização ou exclusão;
- Ponto de carregamento não encontrado, para atualização ou exclusão;
- Não existem reservas cadastradas para o posto de recarga ou veículo;
- A reserva não foi encontrada, para exclusão.

**2. ChargingStationsFile.py:**

Este arquivo contém a classe "ChargingStationsFile", responsável por armazenar, manipular e recuperar os dados dos **postos de recarga**, a partir do arquivo "charging_stations.json". Os postos de recargas são identificados por um "chargingStationID" único, e cada um deve conter uma localização única nas coordenadas "x" e "y" do plano cartesiano.

**3. ChargingPointsFile.py:**

A classe "ChargingPointsFile" manipula o arquivo "charging_points.json", que contém os **pontos de carregamento** disponíveis em cada posto de recarga.

**Cada ponto é identificado por um "chargingPointID" único, e contém as seguintes informações:**

- **"chargingStationID":** a qual posto de recarga ele pertence.
- **"power":** a potência do carregador em kW.
- **"kWhPrice":** o preço por kWh.
- **"availability":** o seu estado de disponibilidade, como "livre", "ocupado" ou "reservado".

**4. ReservationsFile.py:**

A classe "ReservationsFile" interage com "reservations.json", armazenando, lendo, criando, editando e excluindo as **reservas** com as seguintes informações:

- **"reservationID":** seu ID, deve ser único para o mesmo ponto de carregamento.
- **"chargingStationID":** o ID do posto de recarga onde ela será realizada.
- **"chargingPointID":** o ID do ponto de carregamento onde ela será realizada.
- **"chargingPointPower":** a potência do carregador em kW.
- **"kWhPrice":** o preço por kWh.
- **"vehicleID":** o ID do veículo que usará a reserva.
- **"duration":** o tempo necessário em horas para finalizar a carga completa do veículo.
- **"startDateTime":** a data e hora de início da reserva.
- **"finishDateTime":** a data e hora de finalização da reserva, a data de início somada com o tempo necessário para finalizar uma carga completa do veículo.
- **"price":** o preço da reserva, a ser pago.

**5. Informações sobre o Dockerfile e Docker Compose do servidor:**

A **imagem base** utilizada foi **"python:3.13.2".**

Os arquivos para gerar a imagem são copiadas do diretório **"src/Cloud".**

O **arquivo principal** a ser executado é **"Cloud.py".**

O **nome do container** do servidor será **"cloud_server".**

A **variável de ambiente** "PYTHONUNBUFFERED=1" foi definida, para **permitir a exibição dos prints** no terminal.

A **rede** ao qual o servidor estará conectado é **"recharging_manager".**

A **porta do servidor** exposta é **"64352".**