REPOSITÓRIO GITHUB

FIAP - Faculdade de Informática e Administração Paulista

FIAP

PROJETO - GLOBAL SOLUTION 2

Grupo 8

Integrantes:

- Jônatas Gomes Alves
- Iolanda Helena Fabbrini Manzali de Oliveira
- Murilo Carone Nasser
- Pedro Eduardo Soares de Sousa
- Amanda Fragnan

https://github.com/Ioiofmanzali/GLOBAL_SOLUTION_2_-GRUPO81TIAO

GLOBAL SOLUTION 2

GRUPO 8

DESAFIO GLOBAL SOLUTION

A cidade de São Paulo enfrenta, ano após ano, o desafio crescente das enchentes, alagamentos, chuvas intensas e enxurradas. Fenômenos como esses têm se tornado cada vez mais frequentes e severos, impactando diretamente a vida dos moradores, a mobilidade urbana e a infraestrutura da capital. Em 2025, por exemplo, episódios de chuva forte colocaram praticamente todas as regiões da cidade em estado de atenção, com registros de ruas e avenidas alagadas, bairros como Santo Amaro e Piraporinha submersos, quedas de árvores e milhares de imóveis sem energia elétrica.

A topografia acidentada, a impermeabilização do solo e o crescimento acelerado da cidade agravam o risco de transbordamento de rios e córregos, além de potencializar o impacto das enxurradas e enchentes. Mesmo com investimentos em drenagem, monitoramento e sistemas de alerta, São Paulo segue vulnerável a eventos extremos, que causam prejuízos materiais, perdas humanas e demandam respostas rápidas do Poder Público.

Diante desse cenário, torna-se fundamental investir em soluções digitais inovadoras, capazes de prever, monitorar e mitigar os impactos desses desastres. A análise de dados reais, o uso de Inteligência Artificial e o cruzamento de informações meteorológicas e ambientais permitem antecipar riscos, emitir alertas e orientar ações preventivas, contribuindo para uma cidade mais segura para todos.

INTRODUÇÃO

SOBRE O PROJETO

Foi desenvolvida a uma aplicação para o monitoramento de eventos e emissão de alertas para desastre hidrológicos (chuvas intensas, enxurradas, alagamentos e inundações) enviados via SMS (API AWS via SNS) exclusivamente para Gestores Públicos, Defesa Civil, Corpo de Bombeiros e entidades envolvidas com a gestão de desastres naturais. Por isso optamos por deixar a aplicação principal somente com as informações necessárias para o nosso objetivo, que é criar uma interface em Streamlit, amigável e que permita a visualização dos dados de nível do rio e chuvas e disparo de um alerta via SMS para os números de telefone previamente cadastrados.

É importante destacar que esta aplicação foi desenvolvida utilizando uma combinação de dados reais e dados gerados sinteticamente por programas (C++ e Python). Nosso objetivo principal ao empregar essa abordagem foi demonstrar a funcionalidade completa do sistema.

Os dados reais foram utilizados para treinamento da IA com o objetivo de prever da forma mais precisa possível, considerando-se o tamanho dos datasets, o risco de enchentes a partir da atribuição do usuário ou dos dados gerados pelo ESP 32 (simulando captação de dados de sensores reais).

Para fins acadêmicos os arquivos relacionados a análise exploratória, treinamento de ML e DL, ESP32 estão disponíveis na pasta 'docs' do nosso repositório do GitHub, porém não são visualizados na aplicação principal do Streamlit.

DESENVOLVIMENTO

PYTHON + STREAMLIT

O sistema é construído em Python e utiliza diversas bibliotecas para diferentes funcionalidades:

- Streamlit: Para a criação da interface de usuário interativa.
- Streamlit_autorefresh: componente auxiliar que permite que a aplicação Streamlit atualize automaticamente em intervalos regulares
- Pandas: Para manipulação e análise de dados tabulares.
- NumPy: Para operações numéricas.
- Scikit-learn (sklearn): Para implementação de modelos de ML
- OS: Para interação com o sistema operacional (criação de diretórios, verificação de arquivos).
- Requests: Para realizar requisições HTTP para obter dados de uma API Oracle.
- Datetime: Para manipulação de datas e horas.
- Matplotlib e Plotly: Para criação de visualizações de dados.
- Locale: Para formatação de números e datas de acordo com a localidade (português do Brasil).
- IO (BytesIO): Para trabalhar com dados binários em memória.
- Joblib: Para serialização do modelo treinado de ML

A interface do usuário foi organizada em uma única página e mostra os níveis atual, esperado e previsto do rio e a classificação do risco de enchente.
Na aba lateral, podemos determinar o nível de água (grave e moderado) e também simular situações com valores atribuídos de nível de rio e chuva.

COMPONENTES DA APLICAÇÃO PRINCIPAL

Arquivo/Pasta	Descrição
`app.py`	**Interface de Usuário:** interface interativa (via Streamlit) que permite aos usuários visualizar as previsões, análises e o status geral do sistema. É o ponto de interação visual com as informações geradas pelos modelos e dados coletados.
`main.py`	**Lógica do ESP32 e Integração com Banco de Dados:** Script principal para a execução da lógica de comunicação com o ESP 32. Sua função central é coletar dados desses dispositivos e integrá-los ao banco de dados Oracle via chamadas de API, atuando como o ponto de entrada para a ingestão de dados brutos do hardware.
`oracle.sql`	**Scripts de Banco de Dados:** Contém os comandos SQL necessários para a criação das tabelas no banco de dados Oracle, especificamente para armazenar dados de nível de água e volume de chuva.
`requirements.txt`	**Gerenciamento de Dependências:** Lista as bibliotecas Python de terceiros e suas respectivas versões das quais o projeto depende.
`treinar_modelos.py`	**Treinamento de Modelos de ML:** Script dedicado ao ciclo de vida dos modelos preditivos. É responsável por carregar os datasets brutos, realizar o pré-processamento de dados, treinar os modelos de machine learning para previsão de chuva e nível esperado, serializá-los e salvá-los no formato `.joblib`.
`utils.py`	**Utilitário e Lógica de Negócio Central:** Contém funções auxiliares e a lógica de negócio crítica do sistema. Inclui as chamadas às APIs de terceiros (Oracle Cloud para dados de sensores e AWS Lambda para alertas SMS), incorpora a lógica de avaliação de risco de enchente (realizando cálculos e classificações), interage com o banco de dados Oracle para salvar leituras adicionais de sensores e garante o disparo automático de alertas SMS quando as condições de risco atingem limiares predefinidos.

ORACLE

Esse projeto utiliza duas funcionalidades Oracle:

- API RESTful da Oracle, hospedada na Oracle Cloud, configurada para permitir tratamento de paginação e erros, garantindo que os dados necessários para as funcionalidades do projeto sejam carregados de maneira confiável.

- DB Oracle, para salvar os dados gerados pelo ESP 32, simulando uma situação real de captação de dados por sensores.

ESTRUTURA DA API ORACLE

- Requisição HTTP GET:

Quando buscar_nivel_rio() ou buscar_volume_chuva() são chamadas, elas executam uma operação requests.get(). Isso instrui o programa a enviar uma requisição HTTP GET para a URL especificada (API_NIVEL_AGUA ou API_VOLUME_CHUVA) em um tempo limite de 5 segundos para a requisição. Se o servidor não responder dentro desse período, uma exceção será levantada.

- Recebimento da Resposta HTTP:

O servidor da API processa a requisição GET e, se tudo estiver correto, envia de volta uma resposta HTTP. Essa resposta contém um código de status (ex: 200 OK, 404 Not Found, 500 Internal Server Error) e o corpo da resposta. Isso garante que o programa não tente processar dados de uma requisição que falhou, tornando o tratamento de erros mais robusto.

- Decodificação JSON (response.json()):

Se a requisição foi bem-sucedida (código de status 2xx), o corpo da resposta é esperado que esteja no formato JSON. A linha data = response.json() é responsável por parsear a string JSON recebida no corpo da resposta HTTP e convertê-la em um objeto Python em um formato específico - {'data_leitura': 'YYYY-MM-DDTHH:MM:SS', 'valor': X.Y} - que será transformado em um dicionário Python com as chaves 'data_leitura' e 'valor'.

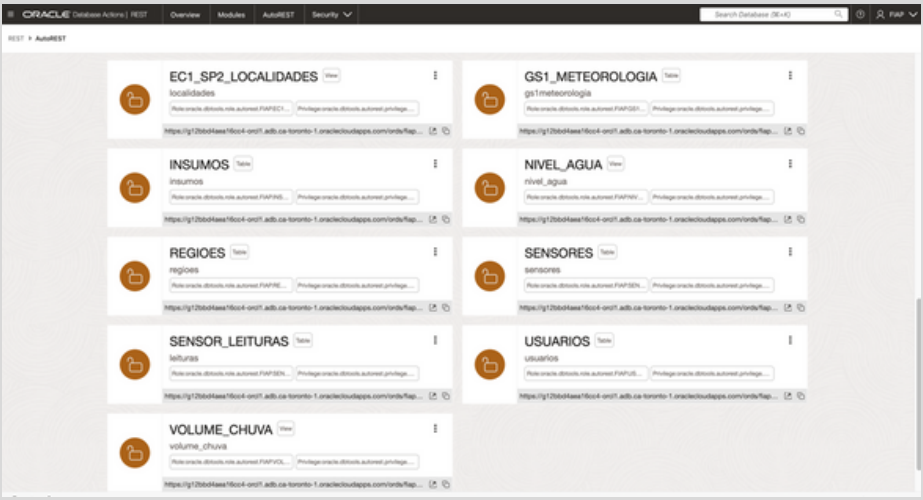
- Tratamento de Erros:

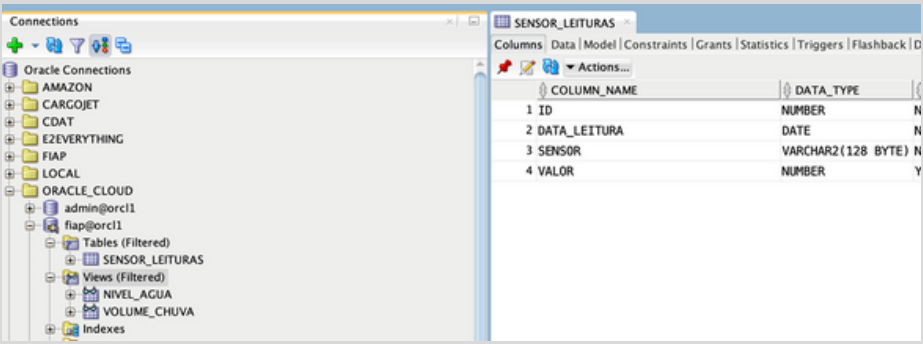
As operações de consumo de API são encapsuladas em blocos try-except.

- except requests.exceptions: Captura qualquer erro relacionado à requisição HTTP (problemas de rede, timeout, erros de status HTTP capturados por raise_for_status()).
- except ValueError: Captura erros que ocorrem se a resposta da API não for um JSON válido ou se houver problemas na sua decodificação.

Em ambos os casos de erro, uma mensagem é exibida usando st.error (a aplicação é construída com Streamlit para exibir esses erros na interface do usuário) e a função retorna None, sinalizando que a operação de busca falhou.

Componentes da API Rest com Oracle Database



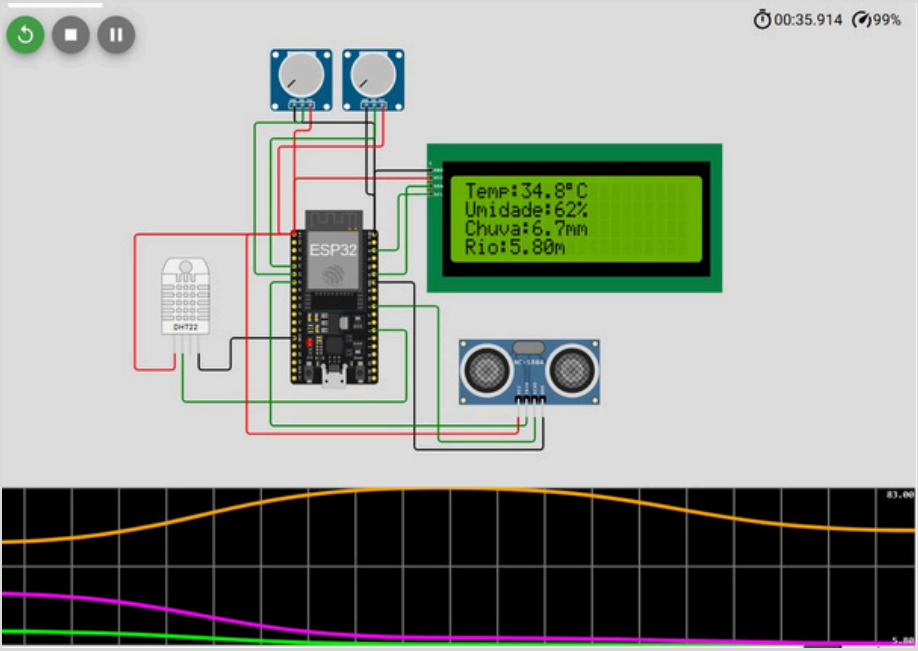


oracle.sql U

```sql
create or replace view nivel_agua as
select to_char(data_leitura, 'YYYY-MM-DD HH24:MI:SS') as data_leitura, valor
  from sensor_leituras l
 where sensor = 'RIO'
 order by l.data_leitura desc
 fetch first row only
/

create or replace view volume_chuva as
select to_char(data_leitura, 'YYYY-MM-DD HH24:MI:SS') as data_leitura, valor
  from sensor_leituras l
 where sensor = 'CHUVA'
 order by l.data_leitura desc
 fetch first row only
/
```

ESP 32



O módulo ESP 32 foi configurado para simular a coleta de dados de sensores reais. Ele gera informações que, na prática, seriam lidas de dispositivos físicos, como sensores de temperatura, umidade ou nível de água.
Para tornar o módulo o mais “real “ possível foram conectados dois potenciômetros e um DHT 22 na placa do ESP 32, com visualização dos dados por um display de LED

obs: os dados são gerados aleatoriamente

```
// Geração de números aleatórios para Temperatura
float generateFakeTemperature() {
  // Gera temperatura entre 15.0 e 40.0 °C
  return random(050, 401) / 10.0; // Multiplica por 10 e divide depois para ter casas decimais
}

// Geração de números aleatórios para Umidade
float generateFakeHumidity() {
  // Gera umidade entre 20% e 100%
  return random(20, 101); // O limite superior é exclusivo, então 96 para incluir 95
}

// Geração de números aleatórios para Chuva (Precipitação)
float generateFakeChuva() { // Alterado o nome da função para 'generateFakeChuva'
  // Gera precipitação entre 0.0 e 50.0 mm (pode simular sem chuva ou chuva forte)
  // random(min * 10, max * 10 + 1) / 10.0 para floats
  return random(0, 501) / 10.0; // De 0.0 a 50.0 mm
}

// Geração de números aleatórios para Nível do Rio
float generateFakeRio() { // NOVA FUNÇÃO para o nível do rio
  // Gera nível do rio entre 0.5 e 15.0 metros
  // Ajuste estes limites para simular níveis normais e de enchente
  return random(5, 151) / 10.0; // De 0.5 a 15.0 metros
}
```

```
// --- Gerar Dados Fake ---
temperatura = generateFakeTemperature();
umidade = generateFakeHumidity();
chuva = generateFakeChuva(); // Chamando a nova função para chuva
rio = generateFakeRio();     // Chamando a nova função para o rio

// --- Enviar dados para o Monitor Serial ---
Serial.print("Temperatura: ");
Serial.print(temperatura, 1); // 1 casa decimal
Serial.print(" °C\t");
Serial.print("Umidade: ");
Serial.print(umidade, 0); // Sem casas decimais
Serial.print(" %\t");
Serial.print("Chuva: "); // Alterado o rótulo
Serial.print(chuva, 1);   // Variável 'chuva'
Serial.println(" mm");
Serial.print("Rio: ");    // NOVO: Exibindo o nível do rio
Serial.print(rio, 2);     // 2 casas decimais para o rio
Serial.println(" m");
```

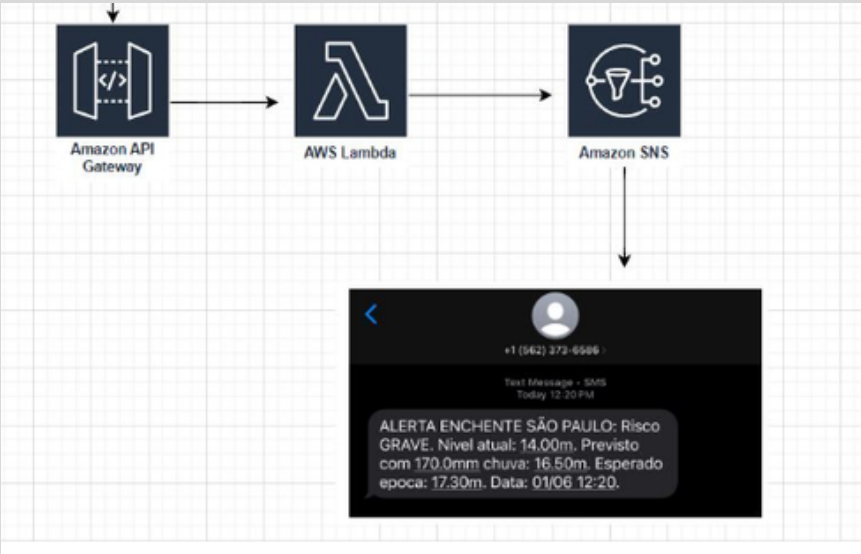
AWS

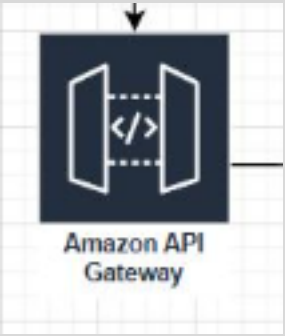
Para a geração de alertas proativos utilizamos os serviços da Amazon Web Services (AWS):

Amazon API Gateway: Atua como um ponto de entrada seguro e escalável para as requisições que acionam o processo de alerta.

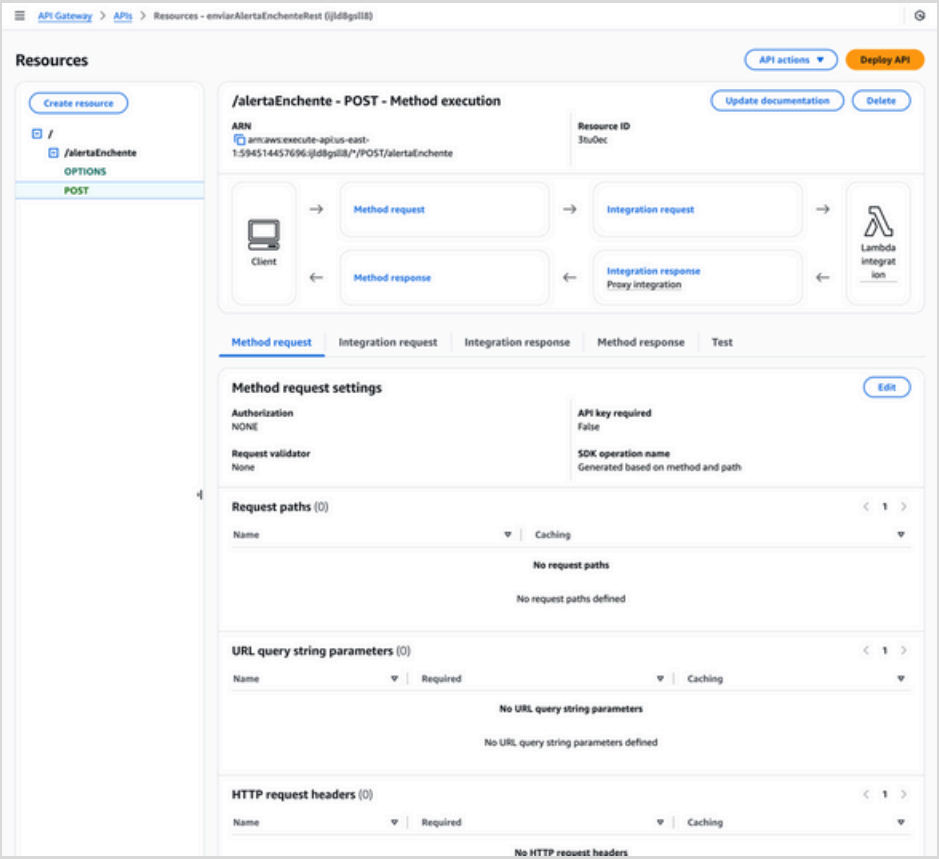
AWS Lambda: Funções serverless que são acionadas via API Gateway para processar os dados de monitoramento e aplicar a lógica de negócio para determinar se um alerta deve ser enviado.

Amazon SNS (Simple Notification Service): Uma vez que a função Lambda decide que um alerta é necessário, o SNS é utilizado para enviar notificações para os numeros de telefone cadastrados.

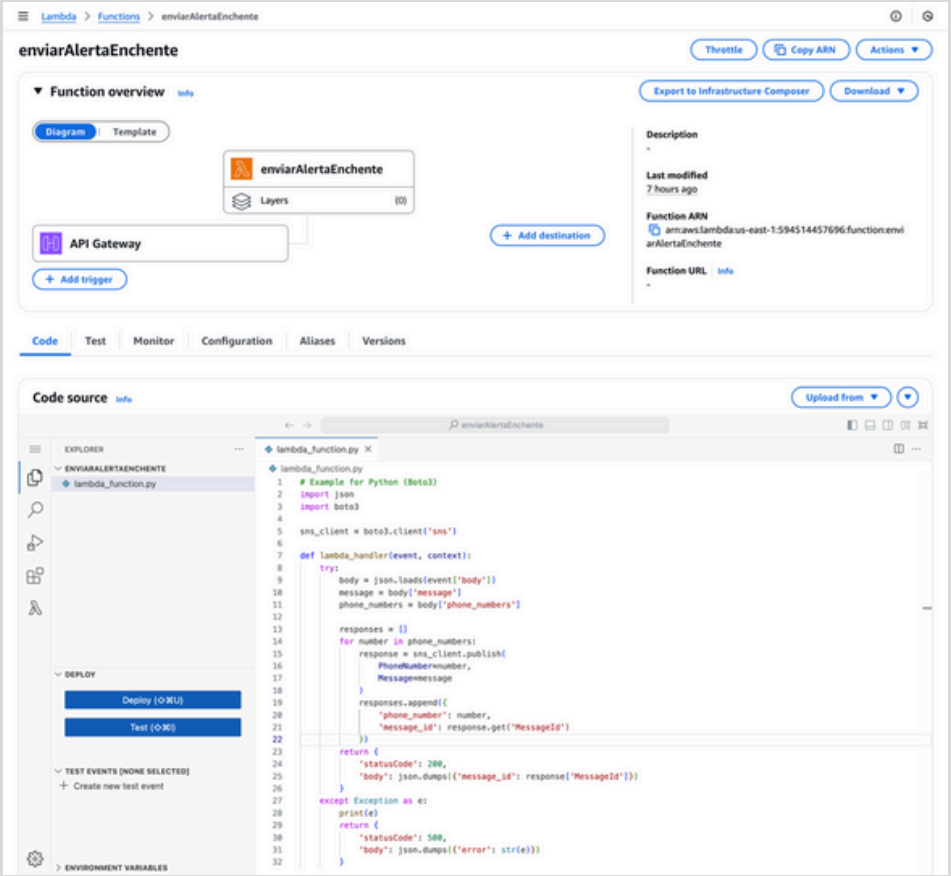




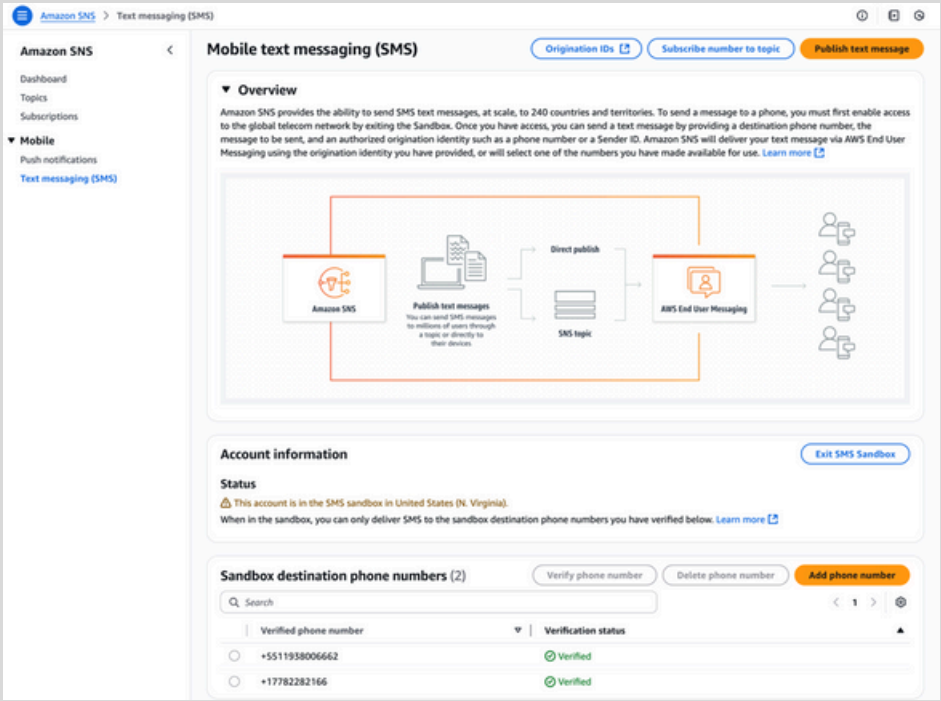
Componentes da API Gateway



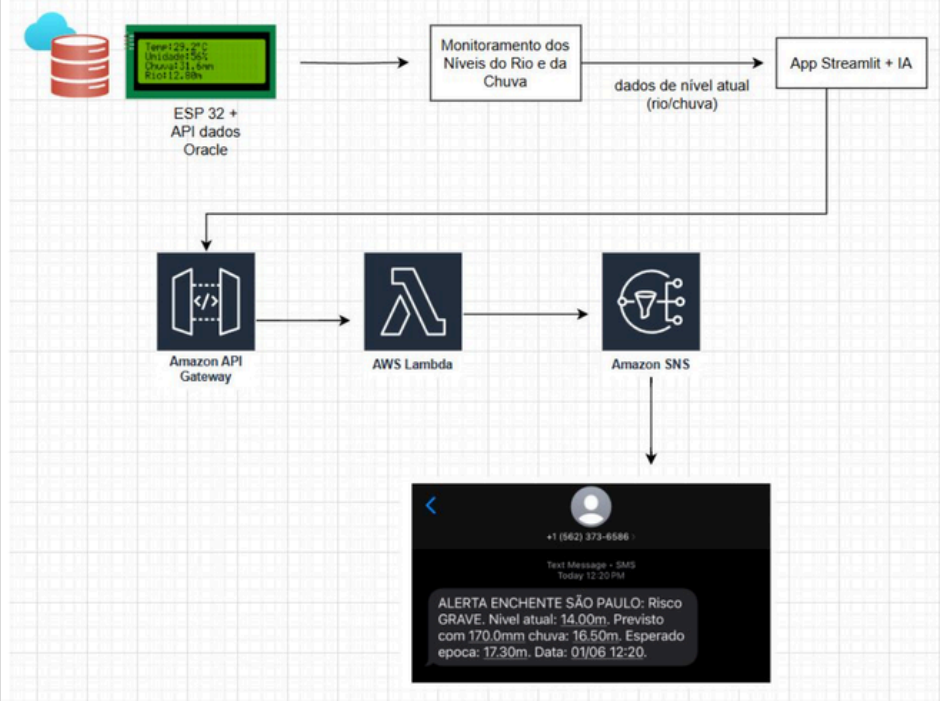
Componentes AWS Lambda



Componentes AWS SNS



ARQUITETURA DO SISTEMA



A arquitetura do sistema é composta por diversas camadas interconectadas, garantindo a coleta, processamento, análise e disseminação das informações:

- dispositivo ESP32 é responsável pela coleta de dados ambientais, como temperatura (Temp), umidade (Humid), nível do rio (RioLevel) e nível da chuva (Chuva). Os dados coletados são enviados para uma API de dados Oracle, onde são armazenados e ficam disponíveis para consumo. Um display local no ESP32 mostra as leituras em tempo real, indicando a temperatura, umidade, nível do rio e o nível da chuva.
- módulo central de Monitoramento dos Níveis do Rio e da Chuva acessa os dados da API Oracle. Os dados de nível atual (do rio e/ou da chuva) são enviados para um Aplicativo Streamlit com Inteligência Artificial.

ARQUITETURA DO SISTEMA

Para a geração de alertas proativos, a arquitetura se integra com serviços da Amazon Web Services (AWS):

Amazon API Gateway: ponto de entrada para as requisições que acionam o processo de alerta.

AWS Lambda: Funções serverless acionadas via API Gateway para processar os dados de monitoramento e aplicar a lógica de negócio para determinar se um alerta deve ser enviado.

Amazon SNS (Simple Notification Service): utilizado para enviar notificações para os números de telefone cadastrados

A mensagem de alerta inclui informações cruciais como:

- **ALERTA ENCHENTE SÃO PAULO** (indicação clara do tipo de evento)
- **Risco GRAVE** (classificação do risco)
- **Nível atual** (nível atual do rio)
- **Previsão** (quantidade de chuva prevista e o nível de rio esperado)
- **Data e Hora** (momento em que o alerta foi emitido)

RESULTADOS ESPERADOS, IMPACTO E BENEFICIOS DO SISTEMA DE MONITORAMENTO DE ALERTA DE ENCHENTES

Nossa aplicação foi desenvolvida com o propósito de simular um sistema de monitoramento de nivel de chuva e de elevacao do nível do rio e enviar alertas para que as autoridades públicas responsáveis pela gestão de crises emergências possam mobilizar as ações necessárias para salvar vidas, proteger patrimônios e fornecer inteligência preditiva diante de eventos climáticos extremos, como enchentes.

Através da coleta contínua e precisa de dados ambientais (temperatura, umidade, nível do rio e chuva) simulados na nossa aplicação pelo dispositivo ESP32, e do processamento inteligente via IA no Streamlit, esperamos identificar padrões e prever comportamentos hidrológicos com maior antecedência.

É importante enfatizar que esse modelo de inteligência artificial foi treinado usando uma combinação de dados reais e dados fictícios, o que nos permite apenas simular uma variedade de cenários.

A capacidade de gerar alertas proativos e direcionados. A integração com a AWS (API Gateway, Lambda e SNS) garante que, tão logo uma condição de risco (identificada pela IA) seja detectada, uma notificação de alerta crucial seja disparada via SMS para os usuários cadastrados. A mensagem é clara e completa, incluindo o tipo de evento ("ALERTA ENCHENTE"), a classificação do risco ("Risco GRAVE"), o nível atual do rio, a previsão de chuva/nível do rio e o horário do alerta. O resultado esperado é uma comunicação de crise mais eficiente e abrangente, alcançando diretamente aqueles que precisam ser avisados.

O módulo central de Monitoramento, alimentado pela API Oracle, fornece um panorama abrangente dos níveis de rio e chuva.

Com a inteligência artificial do Streamlit, que processa dados históricos e atuais para projeções, os gestores terão um suporte robusto para a tomada de decisões informadas e estratégicas durante emergências. Isso poderia incluir a otimização do planejamento de rotas de evacuação, alocação de recursos e mobilização de equipes de resgate, resultando em uma gestão de crises mais coordenada e eficaz.