Refinamento Técnico - Criação da lambda de cadastro

Dono	@Evandro Douglas Capovilla Junior (Unlicensed)
Status	EM REVISÃO
Data	02/08/2025
Estória	<Link da Estória>
Spike	<Link da Spike>
Frente de Desenvolvimento	☐ Front ☑ Back
Figma	<Link do Figma>
ADR	<Link da ADR>
Revisores	☑ @Evandro Douglas Capovilla Junior (Unlicensed) ☐ @Evandro Douglas Capovilla Junior (Unlicensed) ☐

Contexto

Uma nova funcionalidade será implementada para permitir que os jogadores (Treinadores) registrem os Pokémons capturados em suas aventuras. As requisições de cadastro serão processadas de forma

assíncrona na nuvem da AWS, por meio de um novo endpoint exposto pelo **AWS API Gateway**, utilizando o **Lambda** para o processamento dos dados.

Suposições

- O tráfego do site aumentará nos periodos das 12:00 ~ 16:00.
- Os usuários constumam ver seus pokemons no final do dia.
- A criação da lambda de intercepção dos dados será desenvolvida por outro time

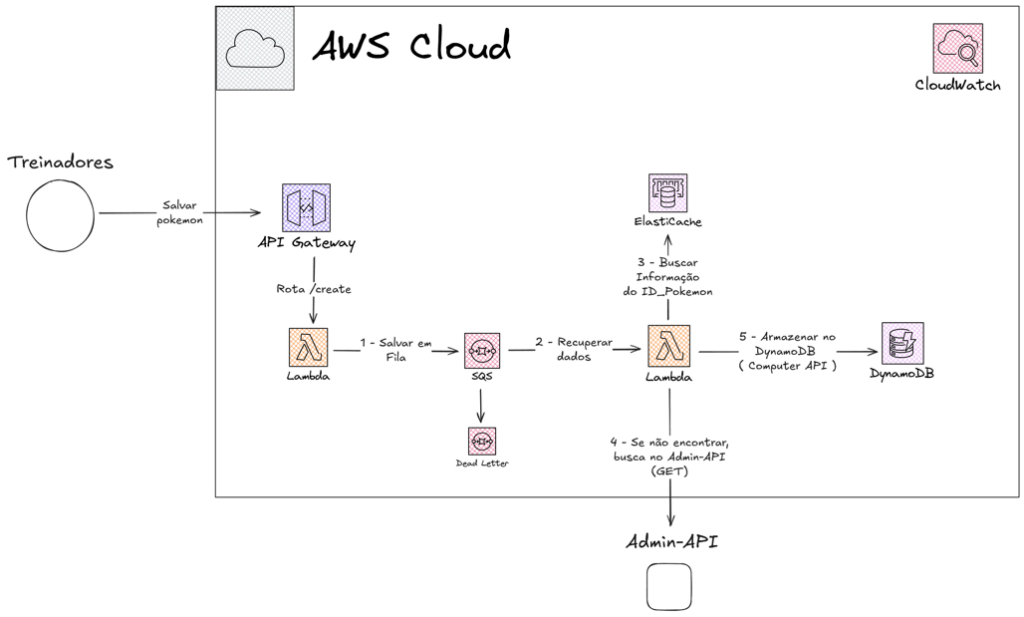
Decisão

Criação de uma lambda para cadastro de API

A solução foi projetada com uma arquitetura **serverless** utilizando **AWS Lambda** para processar as informações. Esta escolha garante alta escalabilidade e eficiência de custos, pois o pagamento é baseado no tempo de execução, eliminando a necessidade de manter um serviço ativo 24 horas por dia. Para otimizar o desempenho, o **Amazon ElastiCache (Redis)** foi implementado para armazenar dados da Pokedex-API, reduzindo a latência e a carga desnecessária sobre a API de origem.

Detalhes

Implementaremos a lambda que fará o processamento dos dados da fila, tendo como responsabilidade validar os dados recebidos, recuperar a informação do pokemon capturado pela propriedade “pokemon_id” da request e por fim, salvar esses dados no DynamoDB. Qualquer problema detectado deverá ser colocado novamente na fila para uma analise manual, se necessario, e a inclusão de logs dessas requisições. Implementaremos um cache distribuído usando Redis, que será integrado entre nossa camada de aplicação e a do Admin-API, tendo como resposabilidade salvar os pokemons já consultados. O diagrama a seguir ilustra a arquitetura proposta:



Desenvolvimento

Função Lambda

- Implementar lógica para lidar com possíveis inconsistências temporárias de cache
- Adicionar indicadores visuais de atualização do carrinho

capture/handler/CapturePokemonHandler.java

```java
// Imports Necessarios.

public class CapturePokemonHandler implements RequestHandler<APIGatewayProxyRequestEvent, APIG

    private final ObjectMapper objectMapper = new ObjectMapper();
    private final DynamoDbTable<PokemonCapturado> pokemonCapturadoTable;
    private final CacheService cacheService;
    private final AdminApiClient adminApiClient;

    public CapturePokemonHandler() {
        DynamoDbClient ddb = DynamoDbClient.builder().region(Region.of(System.getenv("AWS_REGI
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder().dynamoDbClien
        this.pokemonCapturadoTable = enhancedClient.table("PokemonCapturado", TableSchema.from

        String redisHost = System.getenv("REDIS_HOST");
        this.cacheService = new CacheService(redisHost);

        String adminApiUrl = System.getenv("ADMIN_API_URL");
        this.adminApiClient = new AdminApiClient(adminApiUrl);
    }

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent request, Con
        try {
            // 1. Desserializa o payload da requisição
            CapturePokemonRequest captureRequest = objectMapper.readValue(request.getBody(), C
            Long idPokemonBase = captureRequest.getIdPokemonBase();

            // 2. Busca no cache (ElastiCache)
            Pokemon pokemon = cacheService.getPokemon(idPokemonBase);

            if (pokemon == null) {
                // 3. Se não encontrar, busca na Admin-API
                logger.info("Pokémon não encontrado no cache. Buscando na Admin-API...");
                pokemon = adminApiClient.getPokemon(idPokemonBase).block();

                if (pokemon == null) {
                    // Retorna 404 se o Pokémon não existir na Admin-API
                    logger.warning("Pokemon base não encontrado.");
                    return new APIGatewayProxyResponseEvent()
                            .withStatusCode(404)
                            .withBody("{\"message\": \"Pokemon base não encontrado.\"}");
                }

                // Salva no cache para futuras requisições
                cacheService.savePokemon(idPokemonBase, pokemon);
            }

            // 4. Salva a informação completa no DynamoDB
```

```
            PokemonCapturado newPokemon = createPokemonCapturado(
              captureRequest.getIdJogador(),
              captureRequest.getIdPokemon(),
              captureRequest.getLevel(),
            );

            pokemonCapturadoTable.putItem(newPokemon);

            // 5. Retorna 200 OK
            return new APIGatewayProxyResponseEvent()
                    .withStatusCode(200)
                    .withBody("{\"message\": \"Pokemon capturado registrado com sucesso!\"}");

        } catch (Exception e) {
            context.getLogger().log("Erro durante o processamento: " + e.getMessage());
            return new APIGatewayProxyResponseEvent()
                    .withStatusCode(500)
                    .withBody("{\"message\": \"Erro interno do servidor.\"}");
        }
    }

    private PokemonCapturado createPokemonCapturado(
      Long idJogador,
      Long idPokemon,
      int level
    ){
      PokemonCapturado pokemonCapturado = new PokemonCapturado();

      pokemonCapturado.setId(UUID.randomUUID().toString());
      pokemonCapturado.setIdJogador(idJogador);
      pokemonCapturado.setIdPokemonBase(idPokemon);
      pokemonCapturado.setLevel(level);
      pokemonCapturado.setDataCaptura(Instant.now());

      return pokemonCapturado;
    }
}
```

Models

- Será necessario a criação do modelo de requisição, dos dados de Pokemon, e a entidade do DynamoDB
  - PokemonRequest
  - Pokemon
  - PokemonCapturado

model/CapturePokemonRequest.java

```
public class CapturePokemonRequest {
    private Long idJogador;
    private Long idPokemon;
    private int level;
}
```

model/Pokemon.java

```java
public class Pokemon {
    private Long id;
    private String nome;
    private String tipo;
    private String habilidades;
    private String urlImagem;
}

```

model/PokemonCapturado.java

```java
public class PokemonCapturado {
    @DynamoDbPartitionKey
    private String id;

    @DynamoDbSecondaryPartitionKey(indexNames = "idJogador")
    private Long idJogador;

    private Long idPokemonBase;
    private Integer level;
    private Instant dataCaptura;
}
```

Service

- Por fim, iremos adicionar dois services que serão responsaveis pela comunicação do AdminAPI e a consulta dos dados no ElastiCache.

service/AdminApiClient.java

```java
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;

public class AdminApiClient {
    private final WebClient webClient;

    public AdminApiClient(String baseUrl) {
        this.webClient = WebClient.create(baseUrl);
    }

    public Mono<Pokemon> getPokemon(Long id) {
        return webClient.get()
                .uri("/api/pokemons/{id}", id)
                .retrieve()
                .bodyToMono(Pokemon.class);
    }
}
```

service/CacheService.java

```java
import redis.clients.jedis.Jedis;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.io.IOException;

public class CacheService {
    private final Jedis jedis;
    private final ObjectMapper objectMapper;
```

```
8
9       public CacheService(String redisHost) {
10          this.jedis = new Jedis(redisHost);
11          this.objectMapper = new ObjectMapper();
12      }
13
14      public void savePokemon(Long id, Pokemon pokemon) {
15          try {
16              String pokemonJson = objectMapper.writeValueAsString(pokemon);
17              jedis.setex("pokemon-" + id, 3600, pokemonJson); // Expira em 1h
18          } catch (IOException e) {
19              System.err.println("Erro ao salvar no cache: " + e.getMessage());
20          }
21      }
22
23      public Pokemon getPokemon(Long id) {
24          String pokemonJson = jedis.get("pokemon-" + id);
25          if (pokemonJson != null) {
26              try {
27                  return objectMapper.readValue(pokemonJson, Pokemon.class);
28              } catch (IOException e) {
29                  System.err.println("Erro ao ler do cache: " + e.getMessage());
30              }
31          }
32          return null;
33      }
34  }
```

Cenários de testes

Descrição	Etapas	Imagem
Fluxo de adição de pokemon com redis	1. Backend recebe uma requisição da fila para "Adicionar pokemon" 2. Backend valida os dados 3. Busca no Redis 4. Salva no DynamoDB	<IMAGEM>
Fluxo de adição de pokemon consultando Admin API	1. Backend recebe uma requisição da fila para "Adicionar pokemon"	<IMAGEM>

	2. Backend valida os dados 3. Busca no Redis (não encontrado) 4. Busca no Admin API 5. Salva no DynamoDB	
Fluxo de adição de pokemon não existente	1. Backend recebe uma requisição da fila para "Adicionar pokemon" 2. Backend valida os dados 3. Busca no Redis (não encontrado) 4. Busca no Admin API (não encontrado) 5. Salva no DynamoDB	<IMAGEM>
Fluxo de adição de pokemon com falha	1. Backend recebe uma requisição da fila para "Adicionar pokemon" 2. Ocorre algum erro durante o processamento 3. A requisição volta para a fila ***obs: A própria integração entre a***	<IMAGEM>

	Lambda e o SQS faz isso de forma automática, ao cair no catch	

Observações

- Será necessário um período de monitoramento após a implementação para ajustar as configurações de cache. Além de garantir que não tenha nenhuma requisição perdida.
- Planejamos revisar o desempenho desta solução após 3 meses de implementação.
- Futuros aprimoramentos podem incluir
  - Padrão Circuit Breaker, caso ocorra uma queda no Admin API
  - Infraestrutura como Código (IaC), utilizando o **Terraform** ou **CloudFormation**.