Hackathon

Turma 1SOAT

Grupo

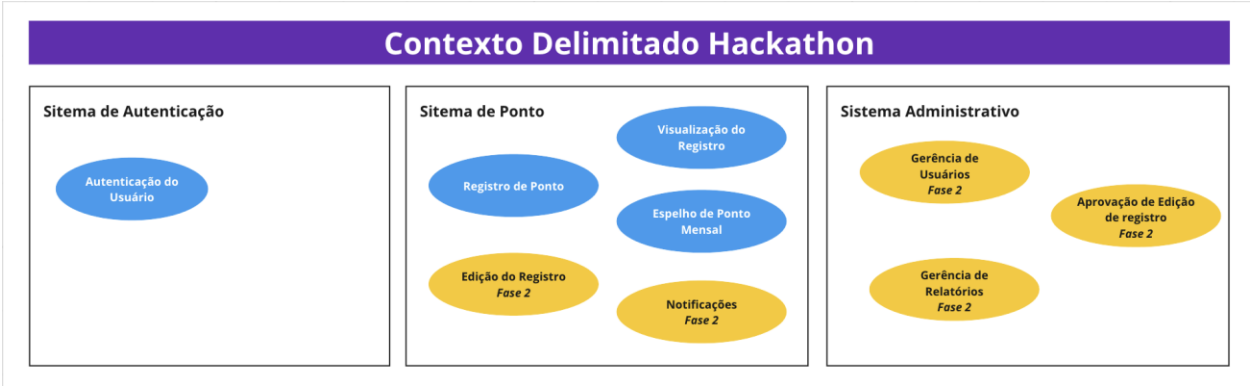
George Baronheid (RM349086)
Lucas Arruda (RM348533)
Murillo de Morais (RM348688)
Ana Lúcia de Faria (RM349133)

Repositórios

Github Hackathon: https://github.com/SOAT1StackGoLang/Hackaton

Vídeos: https://1drv.ms/f/s!AgAxf_qsrSnhhoN0WuhlJQTUtWYBVQ?e=Mhl2iR

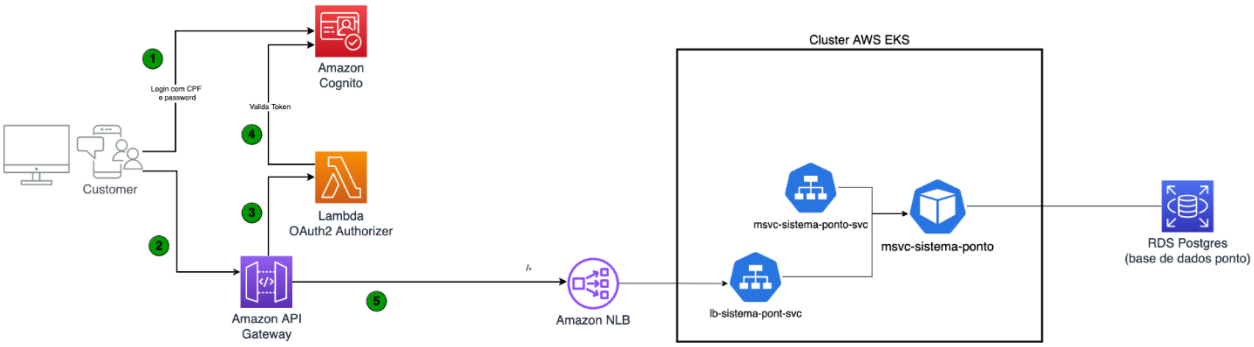
Contexto Delimitado



OBS:

- Os módulos em amarelo não são escopo da primeira fase. Foram colocados aqui apenas para termos uma visão mais clara do sistema como um todo.
- O sistema de autenticação será implementado atarvés do AWS Cognito

Arquitetura Fase 1



Atendimento dos requisitos funcionais

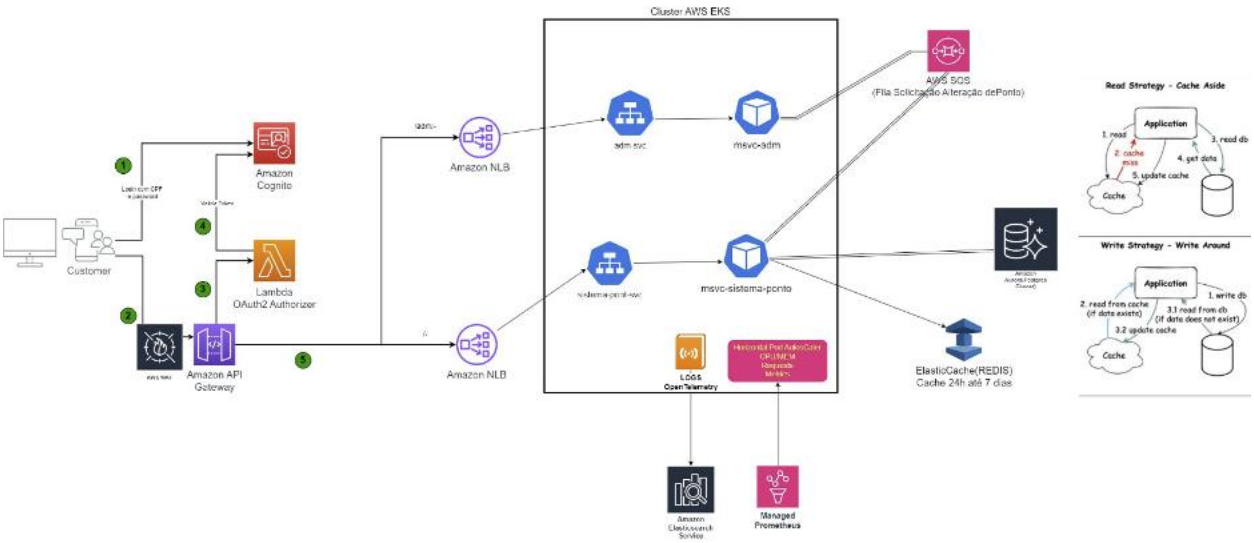
Requisito	Forma de Atendimento
1. Autenticação de Usuário	• Autenticação será feita através do Cognito que está integrado ao sistema como detalhado na seção “Fluxo de Autenticação”.
2. Registro de Ponto	• API **/api/clock-in** implementada pelo microserviço do sistema de ponto
3. Visualização de Registros	• API **/api/reports/daily** implementada pelo microserviço do sistema de ponto
4. Relatórios	• API **/api/reports** implementada pelo microserviço do sistema de ponto
5. Segurança	• *Detalhado na próxima tabela*
6. Disponibilidade	• *Detalhado na próxima tabela*

Atendimento dos requisites não funcionais

Requisito	Forma de Atendimento
1. Escalabilidade	• O serviço é implementado através de kubernetes (AWS EKS) que permite escalar automaticamente a quantidade de mocroserviços provisionados através funcionalidades como HPA (Horizontal Pod Autoscaler). • Além disso, o fato de ter sido implementado em nuvem (AWS), permite que a solução se beneficie da disponibilidade de recursos, ou seja, quando o kubernetes precisar escalar, haverá recursos para isso na nuvem.
2. Desempenho	• O desempenho é conseguido pelo provisionamento de máquinas mais robustas (dimensionadas para atender a demanda) e pelo provisionamento de múltiplos serviços em paralelo (capacidade de HPA do kubernetes). • Adicionalmente, na fase 2 será adicionado um cache Redis, conforme ilustrado na arquitetura da fase2 (não foi implementado na fase 1 por questão de tempo)
3. Disponibilidade	• A alta disponibilidade é alcançada através da implementação de microserviços em kubernetes, com a implementação de mecanismos de *health check* e a instanciação de novos pods caso uma falha seja detectada. Além disso, o microserviço em si foi desenhado levando em

	consideração que novas instâncias podem ser iniciadas, sem que haja perda de funcionalidade. • A implementação em nuvem é outro fator para conferir alta disponibilidade, valendo-se dos KPIs de disponibilidade oferecidos por provedores como AWS.
4. Segurança	• Dados de autenticação do usuário armazenados no AWS Cognito, configurado para usar criptografia • Dados de ponto armazenado em base-de-dados AWS RDS Postgres configurado para usar criptografia • Base-de-dados RDS hospedada na subnet provada do VPC, limitando acesso externo (apenas microserviço precisa acessar essa base-de-dados) • Implementação de logs nos acessos aos recursos da nuvem para monitoramentoe auditoria (AWS CloudWatch e AWS CLoudTrail)
5. Integridade dos Dados	• Os dados de ponto são armazenados no AWS RDS Postgres, que é uma base-de-dados relacional. Uma das principais características oferecidas pelos bancos de dados relacionais é a integridade dos dados, que garante que os dados contidos no banco de dados sejam confiáveis e consistentes.
6. Manutenibilidade	• Implementação de código bem documentado e seguindo arquitetura hexagonal
7. Resiliência	• Implementação de backups regulares, utilizando facilidades providas pela nuvem AWS.
8. Conformidade	• Posteriormente será desenvolvido relatório RIPD, de acordo com LGPD.

Arquitetura Fase 2



Requisitos funcionais

Requisito	Forma de Atendimento
1. Edição de Registros	• Será implementado em duas partes: a solicitação será implementada no microserviços do sistema de pontos; e a aprovação será implementada no microserviços adm. A comunicação entre os dois será feita através de um fila AWS SQS.
2. Notificações	• Será implementado no microserviço adm
3. Administração	• Será implementado no microserviço adm
4. Relatórios	• Será implementado no microserviço adm

Requisitos não funcionais

Adicionalmente na fase 2 serão implementadas uma série de melhorias para reforçar o atendimento dos requisitos não funcionais:

- Implementação de cache Redis no sistema de pontos
- Migração do banco-de-dados AWS RDS para AWS Aurora. O Aurora é um serviço proprietário de banco de dados relacional de alto desempenho projetado para oferecer alto desempenho a um custo menor. Além disso, o Aurora possui tolerância a falhas integrada e capacidade de failover automatizado. Isso facilita o fornecimento de disponibilidade contínua para seus bancos de dados com o mínimo de tempo de inatividade ou interrupção.
- Implementação de logs no padrão Open Telemetry através do serviço Amazon ElasticSearch
- Monitoramento do HPA através de serviço gerenciado Prometheus

Fluxo de Autenticação

1. A autenticação deve ser feita via Cognito. O Cognito retorna um Access Token em resposta à autenticação.
2. Na chamada aos endpoints da aplicação através do API GW, um access token (obtido previamente através da autenticação no Cognito) deve ser enviado através do header Authorization.
3. O API GW chama o lambda techchallenge_oAuth2Authorizer para verificar o token fornecido.
4. O lambda chama o Cognito para validar o token e gera o userID a partir do token.
5. Caso o token seja válido, o API GW encaminha a chamada para LoadBalancer services provisionados no cluster EKS. O userID é adicionando como um parâmetro no header, a ser usado pelo bakcemed na identificação do usuário.