**Relatório de Técnico de Abordagem de Migração e Solução Arquitetural**

Sumário

1. Entendimento de como realizar a migração.........................................................................1
2. Avaliação e Planejamento.....................................................................................................2
   1. AWS *Migration* Hub.......................................................................................................3
3. Identificação de *Workloads* e Análise de Ambiente..............................................................3
4. Escolha de estratégia de migração........................................................................................3
5. Gerenciamento de Dados......................................................................................................5
   1. Ferramentas...................................................................................................................5
6. Modernização da Arquitetura................................................................................................6
7. Automação e CI/CD................................................................................................................7
8. Segurança e Conformidade....................................................................................................7
9. Otimização de Custos.............................................................................................................8
10. Atividade Proposta.................................................................................................................8
    1. Ações de Cobrança ao Cliente......................................................................................10
    2. Negativação junto aos birôs de crédito.......................................................................13
    3. Funcionalidades de Renegociação...............................................................................15
    4. Funcionalidades de Pagamento...................................................................................18
11. **Entendimento de como realizar a migração**

A migração de sistemas monolíticos baseados em mainframe para a nuvem, especialmente usando a AWS, pode envolver uma série de etapas e considerações. Neste documento vamos salientar alguns pontos cruciais para que esta migração aconteça de formar eficaz e seguindo as melhores práticas orientadas para AWS. Encontramos disponível para dar o primeiro passo os pilares do *Well-Architected* *Framework,* que sem sombra de dúvidas é o melhor guia para avaliar e otimizar a arquitetura de sistemas em nuvem.

Tendo esse entendimento, se faz necessário iniciar o projeto de migração com um levantamento do ambiente atual a ser migrado. Logo depois, realizar um plano de migração tento em mente as regras de negócio e também expectativas de migração. Este também deve considerar os pré-requisitos necessários para executar a migração no tempo esperado e com o mínimo de indisponibilidade possível.

Podemos ter a seguinte linha de raciocínio:

1. Levantamento do Ambiente *On-premises*;
2. Análise do Ambiente;
3. Plano de Migração;
4. Migração de Aplicação e Dados;
5. Validação;

Em linhas gerais as migrações podem ser tanto off-line quanto on-line. Existem diferentes técnicas e ferramentas conforme o tipo de dados, armazenamento de arquivos, banco de dados, servidores de aplicação, dentre outras variações. Para saber qual delas utilizar temos que ter um entendimento mais profundo de volume de dados, orçamento, prazo, *complice* dentre outros pontos.

1. **Avaliação e Planejamento**

O primeiro estágio para realizar a migração de sistemas é fazer o levantamento do ambiente atual *on-premises*. Com esse levantamento, fornecerá informações mais detalhadas dos recursos e aplicações a serem migrados, como: quantidade; volume; dependências e outros.  
Dentre alguns recursos de migração temos ferramentas como o AWS *Application Discovery Service* (ADS) para executar este levantamento. Ele é totalmente integrado com o AWS *Migration Hub* e ainda podemos ter de referência o *Well-Architected*. Como falado anteriormente, baseado em seis pilares (excelência operacional, segurança, confiabilidade, eficiência de performance, otimização de custos e sustentabilidade), o AWS *Well-Architected* fornece uma abordagem consistente para que clientes e parceiros avaliem arquiteturas e implementem designs que podem se expandir com o tempo.

* *Well-Architected Review* (WAR): Realizar uma revisão *Well-Architected* para avaliar os princípios-chave, como segurança, confiabilidade, eficiência, desempenho e otimização de custos. [1]
* Avaliação da Migração: Utilizar o AWS *Migration Hub*, para entender a complexidade, riscos e benefícios da migração. [2]
  1. AWS *Migration Hub*: Podemos importar detalhes dos servidores atuais em *on-premises* para a ferramenta e em seguida temos o *Discovery Connector* e *Agent* para aprofundar nas questões apontadas. Construir um plano de migração com o *Hub* identificando os servidores e suas dependências, funções e agrupamento de serviços. Podemos utilizar também o *Strategy Recommendation* para auxiliar na elaboração da estratégia de migração. Juntamente com o Orquestrador do *Hub* conseguimos acelerar e criar um fluxo de trabalho para a migração. Além de um painel gerencial para acompanhar o andamento da migração.

1. **Identificação de *Workloads* e Análise de Ambiente**

* Decomposição de Monólitos: Identificar partes do sistema que podem ser decompostas em *microservices* ou serviços menores. Isso pode envolver a análise de dependências e funcionalidades isoláveis.
* Ambiente: Vale considerar durante a análise de ambiente os seus diferentes tipos. É comum as empresas terem mais de um ambiente para a mesma aplicação; por exemplo, ambientes de DEV, HML e PRD. É uma boa estratégia definir a prioridade de cada ambiente que será alvo da migração e iniciar por um ambiente de menor criticidade, onde o impacto para o negócio será minimizado.

Demais pontos com ambiente estão associados ao custo que cada um deles terá com o provisionamento dos serviços, escalabilidade tanto horizontal quanto vertical, disponibilidade (24/7), dentre outros pontos. Definir com assertividade o tamanho correto de cada VM é um ponto que deve ser levado em consideração.

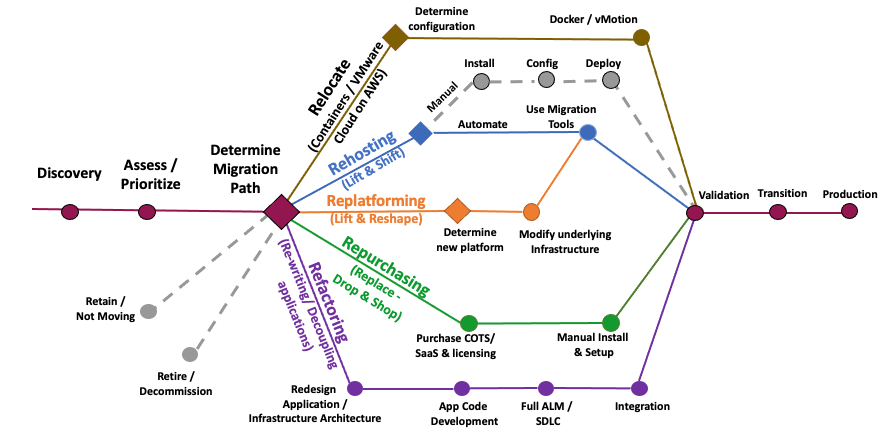
1. **Escolha de estratégia de migração**

Em linhas gerais temos 03 tipos de abordagem para realizar migração para ambiente em nuvem. Não levando em consideração também situações listadas como *Retire* onde chega-se a conclusão que não deve ser feito a migração porque o sistema será descontinuado e *Retain*, onde por tomada de decisão estratégica não será feito no momento atual, temos as seguintes possibilidades:

* *Rehosting (Lift-and-Shift):* Mover o sistema para a nuvem sem modificar sua arquitetura. Recomendação de usar serviços como AWS *Server Migration Service* para facilitar a migração. [3]
  + Obs.: Temos também a nível de nível de *hypervisor* onde a aplicação fica em uma *VMware Cloud* na AWS.
* *Replatforming (Lift-and-Reshape):* Realizar otimizações mínimas no sistema durante a migração, aproveitando alguns serviços nativos da nuvem. Muito comum na migração de apenas bancos de dados *on-primece*. [4]
* *Refactoring (Rearchitecting):* Redesenhar ou reescrever partes do sistema para aproveitar totalmente os serviços nativos da nuvem e melhorar a escalabilidade, desempenho e eficiência. [4]

Podemos falar também uma solução não muito utilizada *Repurchasing* onde o serviço é substituído por uma solução completamente diferente.

Ilustração 1. Tipos de Migração



Fonte. Docs AWS

1. **Gerenciamento de Dados**

Nesta etapa o processo de migração dos servidores, aplicações e/ou dados, terá influência sobre as técnicas de migração apresentadas acima, se servidores inteiros serão migrados de uma só vez, se apenas a aplicação ou apenas os dados, entre outros. Independente da técnica escolhida, será necessário usar alguma ferramenta para executar esta migração.

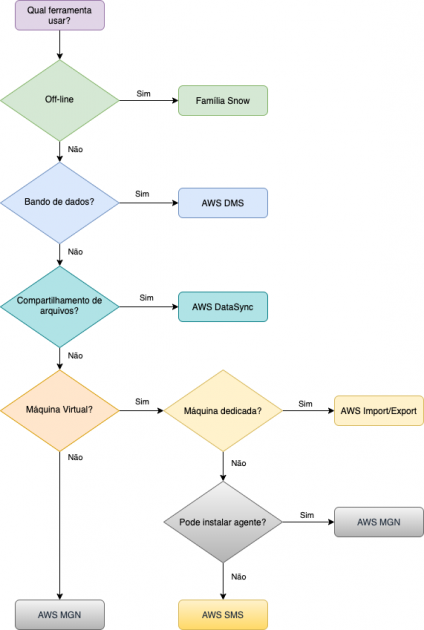
Se for decidido por uma migração off-line, o pré-requisito será a solicitação do dispositivo da família *Snow*. Se trata de migrações de dados por meio de processo físico. Normalmente utilizado como a massa de dados é muito elevada ou o local onde se encontra não tem um amparo de rede satisfatório. Contudo, não será o abordado nesta avaliação.

Para a migração on-line, o pré-requisito será a comunicação de rede entre o ambiente *on-premises* e a AWS. Essa comunicação pode ser pela internet, por *Site-to-Site* VPN ou pelo *Direct Connect*.

Outros pontos comuns relativos à comunicação são as rotas de rede interna do ambiente *on-premises*, uso de proxy e a resolução de nomes de DNS.

* 1. Ferramentas: Existem diversas ferramentas que podem ser usadas, conforme o tipo de dados.
* AWS *Application Migration Service* (AWS MGN): para migrar servidores inteiros;
* AWS *Server Migration Service* (AWS SMS): para migrar máquinas virtuais, sem a necessidade de instalar um agente;
* AWS *Database Migration Service* (AWS DMS): para migrar banco de dados, em cenários de migrações homogêneas e heterogêneas;
* AWS *DataSync*: para migrar compartilhamento e sistemas de arquivos;
* AWS *Import/Export*: para migrar uma imagem de máquina virtual.

Ilustração 2. Tomada Decisão Dados



Fonte. Docs AWS

1. **Modernização da Arquitetura**

Dependendo de qual tipo de migração foi adotada, podemos aproveitar para realizar modernização da arquitetura atual para outra aplicando práticas e conceitos mais modernos ganhando agilidade, mantendo o foco no negócio e menos na gestão e provisionamento de infraestrutura.

* Contêineres e Orquestração: Considerar o uso de contêineres (por exemplo, AWS ECS, EKS) e orquestração de contêineres (por exemplo, *Kubernetes*) para modernizar a entrega de software.
* *Serverless*: Explore opções *serverless* (AWS Lambda) para funções específicas e serviços sem a necessidade de gerenciamento de infraestrutura.

Essa tomada de decisão normalmente vem de uma premissa básica que é o quanto de gestão do recurso você deseja ter. Então saímos desde um provisionamento de um EC2, tomando a responsabilidade de criação da máquina, preocupação com *auto-scaling*, *load balance* dentre outros pontos, até serviços gerenciados pela AWS como o *serveless Fargate, Bath* entre outros. Qual usar e quando usar vai ser uma tomada de decisão após entendimento de alguns pontos como volumetria de dados e custos por exemplo.

1. **Automação e CI/CD**

* Automatização de Implantação: Utilizar ferramentas como AWS *CodePipeline* e AWS *CodeBuild* para automação de compilação e implantação contínua. O processo pode ocorrer também com ferramentas externas visando gerar mais otimização de custo.
* Pipeline CI/CD: Implementar uma *pipeline* de integração contínua/desdobramento contínuo para facilitar a entrega e a atualização de software.

1. **Segurança e Conformidade**

* Configurações de Segurança: Aplicar as melhores práticas de segurança utilizando ferramentas como AWS *Config* e AWS *CloudTrail* para auditoria.
* Gerenciamento de Identidade e Acesso: Utilize AWS *Identity and Access Management* (IAM) para controlar o acesso de maneira granular. Tendo sempre me mente a política da AWS de menor acesso possível para usuários.

1. **Otimização de Custos**

* Monitoramento e Otimização Contínua:
* Implemente monitoramento usando AWS *CloudWatch* e utilize ferramentas como o AWS *Cost Explorer* para otimização contínua dos custos.

1. **Atividade Proposta**

Realizar desenho de solução arquitetural e codificação, visando desacoplamento de serviços, trazendo tecnologias mais modernas em ambiente de nuvem, para simplificar processos, valorizar as funcionalidades, sendo performático e com reuso.

*“1. Caso não tenha conhecimento das funcionalidades crie algumas premissas para realizar o seu exercício e explicite em seu desenho.”*

Como na descrição da atividade não foram passadas informações relevantes de como é o ambiente antigo *on-primeses*, com especificação de módulos, domínios, necessidades, *complice* e até mesmo pontos que devem continuar como por exemplo um banco relacional, toda a linha de raciocínio para o desenvolvimento da solução necessária se dará levando em conta que o processo de migração de **dados** foi realizado da melhor forma possível, seguindo as melhores práticas e planejamento de migração de acordo como o ilustrado neste documento. Toda a apresentação de solução tem por base que as soluções apresentadas são muito próximas no ambiente *on-premises* e a migração está sendo feito no modelo de *Refactoring* como se o desenho já existisse e estamos modernizando e migrando para a nuvem.

Com isso, vamos manter o foco da criação da solução em um frame do todo, levando em conta:

* Ações de Cobrança ao Cliente (Carta, SMS, E-mail, outros possíveis);
* Negativação junto aos birôs de crédito (Serasa, SPC, BVS, outros possíveis);
* Funcionalidades de Renegociação (Agrupamento da Dívida e Simulação de Pagamento, Oferta de Produtos, Efetivação Acordo, entre outros);
* Funcionalidades de Pagamento (Emissão e Pagamento de Boletos, Débito em Conta, outros)

Com esses 04 pontos listados, serão propostas 04 jornadas distintas, uma para cada ação listada. Demais jornadas necessárias para o entendimento do todo, mas que não foram citadas no pedido da atividade, serão explicadas como pré ou pós fluxos junto a explanação de cada um dos pontos listados. Ex. Processo de autenticação de usuário em alguma plataforma deve acontecer quando necessário e será descrito como um pré fluxo, mas não entrará com desenho de serviços na topografia.

Ficamos assim com um fluxo de cobrança, um para negativação em caso de não negociação/pagamento, um para renegociação em casos de clientes que receberam cobrança por um dos veículos de comunicação e por fim um fluxo em comum para geração de formas de pagamento.

Desenho Arquitetural: Arquivo do Draw.io (anexo I). Contendo 04 abas onde cada uma contempla uma jornada.

*“2. Modernização do legado significa aplicar tecnologias novas e emergentes que simplifiquem e acelerem as soluções técnicas, valorizando o desacoplamento das funcionalidades com a criação de peças simples, performáticas e que tenham reuso.”*

Vale ressaltar que as escolhas dos serviços aqui listados se tratam de uma proposta, visto que conseguimos atingir o resultado esperado com mais de uma solução. Como salientado a escolha de qual abordagem adotar deve ser feita de forma mais profunda, entretanto a título de apresentação do desafio temos um ponto de partida de discussão e entendimento.

Obedecendo algumas premissas, temos:

* Modernizar legado dos sistemas de Cobrança e Renegociação.
* Aplicar tecnologias novas e emergentes;
* Simplificar as soluções técnicas;
* Acelerar as soluções técnicas;
* Desacoplamento das funcionalidades;
* Peças simples;
* Peças performáticas;
* Peças que tenham reuso;
* Codifique uma funcionalidade.

Para o desenvolvimento de uma das funcionalidades, fora utilizado a ferramenta do Intellij para desenvolvimento. Desenvolvido uma API com arquitetural hexagonal, utilizando de JAVA 17 com *SpringBoot* 3.0.2. Adicionada dependência do *lombok* para diminuir *boilerplate* nas classes. Mais informações relevantes do projeto estão no README do projeto. Dentre tudo apresentado foi montando uma API para exemplificar o api-gerar-carta-cobraca.

Link do *Github*: https://github.com/ViniciusFernandesAlemeidaTerco/cartacobranca

**Descrição da Solução:**

Como descrito anteriormente vamos tirar por base que a migração dos dados esteja em nuvem. Levando em conta o solicitado de ter uma solução mais moderna e aproveitando o máximo dos serviços da AWS, será utilizado a estratégia de *Refactoring* como se o entendimento da solução *on-primeses* seja o mais próximo do que será apresentado*.*

Os pontos de monitoramento de observabilidade serão citadas como boas práticas junto da solução, contudo, quais e como realizar a utilização de serviços dessa natureza também é subjetiva e vai de acordo com a necessidade de cada cenário. Em especial foi levantando o Cloud Watch e o Cloud Trial por serem serviços mais comuns e corriqueiros que acabam aparecendo em quase toda solução.

* 1. **Ações de Cobrança ao Cliente**
* *Scheduler* usando AWS *CloudWatch Events*: Utilizando o AWS *CloudWatch Events* para criar um *cron job* que aciona o processo de cobrança em intervalos regulares.
* API buscar-clientes-inadimplentes: Através de parâmetros estabelecidos, realizar busca em um RDS para elencar todos os clientes que são elegíveis a cobrança. Podem ser levados em consideração qualquer regra de negócios necessária para cobrança. Tendo ainda a possibilidade de escalada do projeto com mais de um *endpoint* para diversos tipos de cobrança.
* AWS *Lambda* Parametrização: Criado um *function de lambda* na API caso seja necessário buscar qualquer tipo de parametrização dos clientes, como endereço para envio de correspondência, telefone para envio de SMS, endereço de e-mail para o envio do mesmo.
* *Amazon* RDS para Armazenamento de Dados: Armazenar dados dos clientes e suas cobranças no *Amazon* RDS (por exemplo, MySQL, PostgreSQL). RDS simplifica a administração do banco de dados. Foi escolhido essa opção por fazer mais sentido ter amarração forte dos dados dos clientes ao invés de utilização de um banco NoSQL.
* AWS SQS: Foram usados SQS’s em mais de um ponto da solução, o primeiro seria para direcionar todos os clientes que são elegíveis para cobrança para uma próxima *lambda* que irá dividir para qual formato de cobrança o cliente irá receber o contato de inadimplência.
* AWS *Lambda* Fluxo Cobrança: Criada uma *Lambda Function* para processar as mensagens no SQS. Dependendo da ação, a *lambda* pode mandar a cobrança para um SQS para contato por carta, SMS, e-mail, etc. Para cada uma das possibilidades de envio será publicado em um SQS específico.
* Serviços de Envio de Mensagens: Utilizando serviços específicos para cada canal de comunicação. Após postados em seu SQS especifico a transação é destinada posteriormente para seu canal:
  + Para E-mails: *Amazon Simple Email Service* (SES).
  + Para SMS: *Amazon Pinpoint* ou serviços de SMS integrados.
  + Para Cartas: API Gerar Carta Cobrança. Enviar para parceiro externo o pedido de confecção de uma carta de cobrança física. A politica de leitura da mensagem da fila em especial se dará mediante a confirmar de status positivo do parceiro. Ficando assim o *retry* para os casos sem sucesso. Fora adicionado um lambda para autorização antes de envio dos dados para o *endpoint* do parceiro a título de manter boas práticas de segurança da informação trafegada.
* SQS *Call-Back* Cobrança: Utilizado o SQS para realizar *call-back* e atualização no banco dos status de envios de cobrança.
* AWS *Lambda* Processa *Call-Back*: Utilizado *lambda* para leitura do SQS de *call-back* e atualização dos dados no RDS de clientes.
* *Amazon* S3 para Armazenamento de Documentos (opcional): Se as ações de cobrança incluírem documentos anexos, podemos armazená-los no *Amazon* S3.
* *Amazon CloudWatch* e AWS *CloudTrail* para Monitoramento e *Logging* (opcional): Configurar métricas e logs usando *Amazon CloudWatch* e AWS *CloudTrail* para monitorar o desempenho do sistema e rastrear eventos importantes.

Os pontos opcionais querem dizer que todo o processamento pode ocorrer sem ter a necessidade dos mesmos. Contudo, faz parte de boas práticas de desenvolvimento termos a possibilidade de rastreio, monitoramento e logo dos serviços.

Normalmente é levando em conta a escolha de região critérios como custos dos serviços, disponibilidade dos mesmos, *complice* e afins. Em alguns casos mais críticos de latência é interessante deixar os serviços mais próximo do cliente ou pensar em possibilidades de borda ou serviços de link direto. Com relação a VPC e *subnet* temos que ver qual o padrão adotado. Para *subnet* por exemplo podemos dividir os serviços levando em conta níveis de segurança, função, entre outras políticas. No caso do desenho apresentado os serviços de RDS estão em *subnet* separadas. Vamos levar em conta que temos redundância em casos de desastres com replica em mais de uma zona. Utilizaremos uma partição privada (VPC) na nuvem porque a solução proposta não tem necessidade de acesso à internet. Com exceção da comunicação com uma API de um parceiro Externo para produção de cartas de cobrança. Neste caso podemos usar *Gateway* de *Internet* e/ou NAT *Gateway*/Instância para solucionar a questão. Todos os serviços também devem estar devidamente configurados com seus grupos de segurança com os acessos necessários para execução e comunicação com demais serviços. Se faz também presente politicas de *retry* para os SQS de acordo com a necessidade da regra de negócios. Ex. após 10 reprocessamentos sem sucesso encaminhar para DLQ.

**Observação**: Com base no relato da atividade onde é comentado que o processo é feito com *bacth* no monólito, precisaria entender um pouco mais da volumetria de dados para poder trabalhar com esse processamento em lote. Neste caso poderíamos utilizar tanto um *SpringBatch* com o serviço de *Batch* da AWS para buscar os clientes no RDS. Ambas as propostas diferenciam em relação ao grau de gestão que se deseja. Caso a utilização da API faça mais sentido, também pode ser levado em conta a criação da mesma em uma instância EC2 ou utilizar um serviço *serveless* como *Fargate* e *conteiners*. Para isso temos duas possibilidades usando tanto com o EKS quanto o ECS. Em linhas gerais temos que ter o entendimento de Simplicidade vs. Flexibilidade. Se você procura simplicidade e integração estreita com os serviços da AWS, ECS pode ser uma escolha melhor. Se você precisa da flexibilidade e poder do ecossistema *Kubernetes*, EKS pode ser mais apropriado. Essa tomada de decisão de qualquer um dos serviços citados pode variar de acordo com necessidades específicas.

* 1. **Negativação junto aos birôs de crédito**
* *Scheduler* usando AWS *CloudWatch Events*: Utilizando o AWS *CloudWatch Events* para criar um *cron job* que aciona o processo de negativação em intervalos regulares.
* API buscar-clientes-negativacao: Através de parâmetros estabelecidos, realizar busca em um RDS para elencar todos os clientes que são elegíveis a negativação. Podem ser levados em consideração qualquer regra de negócios necessária para envio de negativação. Tendo ainda a possibilidade de escalada do projeto com mais de um *endpoint* para diversos tipos de negativação.
* AWS *Lambda* Parametrização: Criado um *function de lambda* na API caso seja necessário buscar qualquer tipo de parametrização dos clientes, como endereço para envio de correspondência, telefone para envio de SMS, endereço de e-mail para o envio do mesmo ao final do processo de negativação para informar que o mesmo foi negativado.
* *Amazon* RDS para Armazenamento de Dados: Armazenar dados dos clientes e suas cobranças, negativações no *Amazon* RDS (por exemplo, MySQL, PostgreSQL). RDS simplifica a administração do banco de dados. Foi escolhido essa opção por fazer mais sentido ter amarração forte dos dados dos clientes ao invés de utilização de um banco NoSQL.
* AWS SQS: Foram usados SQS’s em mais de um ponto da solução, o primeiro seria para direcionar todos os clientes que são elegíveis para negativação para uma próxima *lambda* que irá dividir para qual birô de negativação o cliente será destinado, podendo ser um ou todos.
* AWS *Lambda* Fluxo Negativação: Criada uma *Lambda Function* para processar as mensagens no SQS. Dependendo da ação programada essa mensagem pode seguir ou não.
* AWS SQS’s Birôs: Para cada processamento do *lambda* de negativação é destinado para cada SQS especifico de um birô. Assim conseguimos ter o controle de volumetria, mandar para mais de um birô o mesmo cliente e mantemos o serviço desacoplado.
* API negativação-cliente: Está API realiza o processo de leitura de cada SQS dos birôs e realiza a comunicação com os parceiros externo para suas respectivas API’s. A política de *retry* do SQS está amarrada em caso de não sucesso recebido pelo parceiro. Tendo status positivo será dado como lido a mensagem da fila.
* *Amazon* S3 para Armazenamento de transações (opcional): Em alguns casos de comunicação com API de terceiros que não temos certeza de seu funcionamento pleno, podemos aproveitar do S3 como uma auditoria, salvando o *request* e *response* da comunicação que foi feita.
* AWS *Lambda* Autorização: Mantendo o nível de segurança adequado, temos um lambda de autorização através de solicitação de *token* para as API’s dos parceiros.
* SNS *Call-Back* Negativação: Utilizado o SNS para realizar *call-back* e atualização no banco dos status devolvidos pelos birôs, também pode ser aproveitado parte do fluxo de cobrança para comunicar o cliente por qualquer meio que o mesmo foi negativado e que ele pode assim que possível entrar em contato para realizar negociação.
* AWS *Lambda* Processa *Call-Back*: Utilizado *lambda* para leitura do SNS de negativação e atualização dos dados no RDS de clientes.
* *Amazon CloudWatch* e AWS *CloudTrail* para Monitoramento e *Logging* (opcional): Configurar métricas e logs usando *Amazon CloudWatch* e AWS *CloudTrail* para monitorar o desempenho do sistema e rastrear eventos importantes.

Normalmente é levando em conta a escolha de região critérios como custos dos serviços, disponibilidade dos mesmos, *complice* e afins. Em alguns casos mais críticos de latência é interessante deixar os serviços mais próximo do cliente ou pensar em possibilidades de borda ou serviços de link direto. Com relação a VPC e *subnet* temos que ver qual o padrão adotado. Para *subnet* por exemplo podemos dividir os serviços levando em conta níveis de segurança, função, entre outras políticas. No caso do desenho apresentado os serviços de RDS estão em *subnet* separadas. Vamos levar em conta que temos redundância em casos de desastres com replica em mais de uma zona. Utilizaremos uma partição privada (VPC) na nuvem porque a solução proposta não tem necessidade de acesso à internet. Com exceção da comunicação com uma API de um parceiro Externo para produção de cartas de cobrança. Neste caso podemos usar *Gateway* de *Internet* e/ou NAT *Gateway*/Instância para solucionar a questão. Todos os serviços também devem estar devidamente configurados com seus grupos de segurança com os acessos necessários para execução e comunicação com demais serviços. Se faz também presente politicas de *retry* para os SQS de acordo com a necessidade da regra de negócios. Ex. após 10 reprocessamentos sem sucesso encaminhar para DLQ.

**Observação:** As mesmas observações feitas para o fluxo de cobrança também são pertinentes para este caso negativação de clientes. Dependendo das necessidades podemos alterar os serviços provisionando algo similar ou de melhor performance. Contudo, não se alteraria o cerne da proposta arquitetural.

* 1. **Funcionalidades de Renegociação**

Para o fluxo de renegociação, seguindo o pedido de montar também um desenho mais técnico voltado para infraestrutura, na aba do draw.io a primeira topografia está o entendimento de infra com as preocupações de região, VPC, zona, boas práticas com *load balance*, *auto scaling* mais bem definidos como apresentado nos padrões da AWS. Logo após o desenho da solução de renegociação no ponto de vista transacional. Se faz necessário entender a necessidade da instituição para montar de acordo com o que melhor se atende. Em alguns casos pode ser o desenho de forma mais completa ou apenas o transacional. Para a *subnet* não foi colocado faixa de endereçamento de IP. Como a quantidade de endereços por si só não influencia em custos, diferentemente da definição de IP’s elásticos, esse entendimento pode ficar para um segundo momento com a definição de solução pronta.

Para este cenário de renegociação, como dependemos de ação vinda dos “usuários/clientes” em um determinado momento, porque o cliente pode escolher o que quer renegociar, como renegociar (mesmo tendo possibilidades previamente informadas e listadas pela instituição financeira), ainda temos essa troca. Sendo assim, entendo que Agrupamento da Dívida, Simulação de Pagamento, Oferta de Produtos, pode e deve ser feito em front deixando o processo dinâmico. Agora, para efetivação da renegociação damos continuidade ao fluxo. Este processo foi nominado como **Fluxo Ativo** onde parte do cliente o despertar em fazer a renegociação.

Ainda em paralelo, nominado **Fluxo Passivo,** apenas o início da jornada é diferente onde parte de a instituição financeira encaminhar algum tipo de possibilidade de renegociação para o cliente. A proposta pode ser elaborada através de uma negociação ativa que foi iniciada, mas não finalizada por algum motivo ou simplesmente elaborada por algum consultor e encaminhada para o cliente.

Favor considerar como pré fluxo toda a jornada de autenticação do usuário em uma plataforma qualquer previamente. Ficando na sequencia a simulação de renegociação e em seguida percorrer o fluxo de renegociação proposto.

* *Front GraphQL*: Esta jornada vamos tomar algumas premissas antes. No desenho foi inserido um S3 para um site estático ou uma *Landing Page* para captação de clientes que estão com dívida. Temos também no início da jornada a estrutura de front, simbolizada com um projeto em *graphql* (Pode ser um site, mobile, sistema interno...) que também pode ser em formato de S3 contendo a parte estática e ações. Qualquer iteração onde o cliente consiga consultar sua dívida e simular uma negociação neste exemplo fica alocado nesta jornada de *front*. Respaldando o *front* temos um BFF usando um *fargate*. Seguindo a mesma lógica dos demais fluxos são necessários os dados do cliente e parametrização de algumas informações. Neste ponto todos os processos necessários para o cliente entrar
* AWS *Fargate*: É uma boa prática utilização de BFF para gerir as necessidades de um *front* *end*. Neste caso utilizado o *fargate* sendo um serviço *servelss* de contêiner. Sendo assim, nossa preocupação fica restrita ao entendimento de regra de negócios necessária e devidamente complexa para buscar os dados dos clientes e suas dívidas para prover uma renegociação do que está em aberto.
* AWS *Dynamo*: Adicionado banco NoSQL para armazenar propostas de renegociação que não foram finalizadas. A não aderência pode ser por desistência, falha, dentre outros. Serve para fins de contatos futuros ou gerar métricas.

* API buscar-clientes-renegociacao: Através de parâmetros estabelecidos, realizar busca em um RDS dos dados do cliente para renegociação. Podem ser levados em consideração qualquer regra de negócios necessária para o processamento. Tendo ainda a possibilidade de escalada do projeto com mais de um *endpoint* para diversos tipos de renegociação.
* AWS *Lambda* Parametrização: Criado um *function de lambda* na API caso seja necessário buscar qualquer tipo de parametrização dos clientes, como endereço para envio de correspondência, telefone para envio de SMS, endereço de e-mail para o envio do mesmo ao final do processo de renegociação para informar que o mesmo foi finalizado.
* *Amazon* RDS para Armazenamento de Dados: Armazenar dados dos clientes e suas cobranças, negativações no *Amazon* RDS (por exemplo, MySQL, PostgreSQL). RDS simplifica a administração do banco de dados. Foi escolhido essa opção por fazer mais sentido ter amarração forte dos dados dos clientes ao invés de utilização de um banco NoSQL.
* AWS SQS: Fora utilizado SQS processa renegociação para deixar o fluxo desacoplado e em seguida encaminhar a proposta de renegociação para um *fargate* poder realizar a comunicação com o *Core* Bancário. É comum termos esse tipo de núcleo centralizado em uma instituição para gerir situações como esta. São sistemas complexos que envolvem tributação, conferencia de saldo, dentre outros pontos.
* AWS *Fargate*: Este serve para fazer a ponto entre nossa solução em nuvem com o servidor *on-primeses* do *Core* Bancário. Ele fica com a incumbência tanto de envio quanto recebimento do processamento na solução.
* AWS *Dynamo*: Adicionado banco NoSQL para armazenar propostas de renegociação que foram enviadas para o Core Bancário. Assim mantemos um ponto de controle e resiliência quando estamos trafegando informações que acabam saindo do ambiente da AWS.
* *Core* Bancário: Como este pode ser considerado como uma outra jornada para validar e efetivar a renegociação que foi criada, a mesma fica como citação no fluxo de renegociação. Acredito que essa parte de processamento do *core* ainda deva ocorrer dentro de legado. [5][6]
* AWS SQS: Fora utilizado SQS *call-back* para receber retorno de processamento do *core* bancário onde informa que a negociação foi efetivada ou não.
* Cobrança Clientes: Encaminhado *call-back* para fluxo de cobrança clientes informando o status da renegociação ao cliente.
* *Amazon CloudWatch* e AWS *CloudTrail* para Monitoramento e *Logging* (opcional): Configurar métricas e logs usando *Amazon CloudWatch* e AWS *CloudTrail* para monitorar o desempenho do sistema e rastrear eventos importantes.

**Observação:** As mesmas observações feitas para nos fluxos de cobrança também no de negociação são pertinentes para este caso de renegociação de clientes. Dependendo das necessidades podemos alterar os serviços provisionando algo similar ou de melhor performance. Contudo, não se alteraria o cerne da proposta arquitetural.

* 1. **Funcionalidades de Pagamento**
* *Front GraphQL*: Temos no início da jornada a estrutura de *front*, simbolizada com um projeto em *graphql* (Pode ser um site, mobile, sistema interno...) qualquer iteração onde o cliente consiga escolher uma fora de pagamento para uma questão qualquer pode ser pesquisada e provisionado o pedido em *front*. Este encaminha para um API gateway que recebe e encaminha as requisições HTTP do *front* para os serviços apropriados. Para o *front* também pode ser em formato de S3 contendo a parte estática e ações.
* API *Gateway*: Este pode receber várias rotas com formas de pagamentos distintas e também gerencia autenticação, autorização e proteção contra ataques. Que por sua vez destina para cada projeto pertinente de pagamento.
* API boletos: responsável por realizar o processamento de boletos. Podendo conter qualquer regra de negócios especifica para validação, conferência, ajuste para um determinado boleto.
* API debito-conta: responsável por realizar o processamento de boletos. Podendo conter qualquer regra de negócios especifica para validação, conferência, ajuste para um determinado débito em conta.
* AWS *Lambda* Parametrização: Criado um *function de lambda* na API para buscar qualquer tipo de parametrização dos clientes. É interessante ter uma camada específica para ter acesso a banco. Sendo assim, podemos ter essa camada fazendo acesso em mais de um banco se necessário retornando as informações necessárias para processar o pagamento.
* AWS SQS: Fora utilizado SQS processa renegociação para deixar o fluxo desacoplado e em seguida encaminhar a proposta de renegociação para o *Core* Bancário. É comum termos esse tipo de núcleo centralizado em uma instituição para gerir situações como esta. São sistemas complexos que envolvem tributação dentre outros pontos.
* AWS *Fargate*: Este serve para fazer a ponto entre nossa solução em nuvem com o servidor *on-primeses* do *Core* Bancário. Ele fica com a incumbência tanto de envio quanto recebimento do processamento na solução.
* *Core* Bancário: Como este pode ser considerado como uma outra jornada apartada para validar, conferir se tem fundo e afins, fica como citação no fluxo de pagamento. As políticas de *retry* para o SQS antes de envio do core bancário deve ser implementada de alguma forma.
* *Amazon CloudWatch* e AWS *CloudTrail* para Monitoramento e *Logging* (opcional): Configurar métricas e logs usando *Amazon CloudWatch* e AWS *CloudTrail* para monitorar o desempenho do sistema e rastrear eventos importantes.

**Observação:** As mesmas observações feitas para nos fluxos de cobrança também no de negociação e renegociação são pertinentes para este caso de pagamentos. Dependendo das necessidades podemos alterar os serviços provisionando algo similar ou de melhor performance. Contudo, não se alteraria o cerne da proposta arquitetural.

**REFERÊNCIAS:**

[1] - https://aws.amazon.com/pt/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitepapers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc

[2] - https://aws.amazon.com/pt/migration-hub/features/?nc=sn&loc=2#Discover\_or\_import\_your\_on-premises\_server\_details

[3] - https://docs.aws.amazon.com/prescriptive-guidance/latest/strategy-rehosting/welcome.html

[4] - https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html

[5] - https://aws.amazon.com/pt/solutions/case-studies/itau-pix/

[6] - https://aws.amazon.com/pt/blogs/aws-brasil/saiba-como-a-natura-construiu-uma-arquitetura-serverless-para-processamento-de-pagamento/