**PROPOSTA DE PROJETO**

**DESAFIO 2**

**OTIMIZAÇÕES**

Prazo de Entrega: **30 DIAS**

**ESCOPO**

A primeira fase do projeto contempla otimizações em curto prazo, visando melhoras para o sistema sem grandes impactos estruturais. O objetivo é analisar a arquitetura atual e promover ações rápidas com impactos imediatos, para que não voltem a ocorrer quedas e instabilidades no período de maior fluxo de uso do sistema (últimos dias do mês).

Para tal, serão analisadas soluções cuja estimativa de horas seja viável para o prazo estabelecido, considerando o uso de alguns recursos da equipe de infraestrutura e da equipe de desenvolvimento. Não serão realizadas grandes alterações arquiteturais na aplicação, visto que exigiriam muitas horas de desenvolvimento e, principalmente, testes de regressão e integração.

**PROBLEMAS**

De acordo com a análise realizada, os principais problemas da aplicação se relacionam ao uso excessivo do banco de dados. Há um gargalo de performance, causando em alguns momentos até a falha na conexão entre a aplicação e o banco de dados, que ocorre possivelmente pela falta de boas-práticas na de código na abertura e uso destas conexões. Além disso, há uma previsão de que em menos de 3 meses a máquina de banco de dados vai exceder o espaço físico provisionado no momento.

Além dos problemas com o banco de dados, a infraestrutura da aplicação hoje não possui fácil escalabilidade por ser executada em um TomEE direto em uma instância de EC2. Como a aplicação recebe picos de uso, em muitos momentos a aplicação está com baixo uso de recursos, e em outros momentos a infraestrutura provisionada não dá conta da quantidade de acessos.

**PROPOSTAS**

1. **Otimização do uso de conexões com o banco de dados**

Como medida inicial para o problema de falta de conexões com o banco de dados, a aplicação deve ser alterada para permitir o uso de um *pool* de conexões com a base. Com isso, ocorre um menor overhead na criação de conexões, ganhando tempo e reduzindo falhas que poderiam ser causadas por conexões que não foram corretamente abertas e fechadas pela aplicação.

Essa configuração deverá ser feita através das *connection strings* da aplicação, nos campos abaixo:

- *Pooling=true*: Deve ser incluído em todas as *strings* de conexão, para habilitar o uso de um pool.

- *MinimumPoolSize=10* e *Maximumpoolsize=5000*: Estes valores indicam a quantidade mínima e máxima de conexões no pool, respectivamente. Serão necessários testes de carga para identificar quais os melhores valores para estas propriedades.

1. **Uso de comandos BATCH para serviços de processamento em lote**

Os serviços de processamento de TED/DOC e de execução de folha de pagamento executam inúmeras transações em lote. Estes devem ser alterados para que a execução de operações no banco de dados por estes serviços utilize comandos BATCH, que são otimizados pela própria *engine* do MySQL para melhor consumo de recursos e conexões, e redução de falhas por *locks*. Para tal, as execuções de SQL que hoje ocorrem com os métodos “*executeInsert*” e “*executeUpdate*” devem ser alteradas para “*addBatch*”, e ao final da geração de todos os comandos deve ser executado o “*executeBatch*”.

1. **Configuração dos agendamentos de serviços de processamento em lote (processamento de TED/DOC e folhas de pagamento)**

Além da otimização do uso de recursos da aplicação, os serviços de processamento de TED/DOC e de execução de folha de pagamento podem ser agendados de forma que sua execução se dê em períodos que reduzam o impacto negativo na utilização do serviço. Para tal, deve ser realizado um estudo histórico para identificar os momentos em que a aplicação recebe picos de uso, e estas informações devem ser cruzadas com a equipe de negócio e jurídico para compreender quais são as obrigações legais que a empresa possui em questão de tempo máximo para efetivação destas transações. Considerando o resultado destes estudos, nos períodos de maior criticidade os serviços devem ser executados com a menor frequência possível dentro do limite permitido pela empresa.

1. **Configuração do Elastic Beanstalk nas instâncias EC2 da aplicação e banco de dados**

O Amazon Elastic Beanstalk (<https://docs.aws.amazon.com/elastic-beanstalk>) permite que sejam executadas aplicações na nuvem sem um provisionamento específico de uma máquina virtual. Este serviço possui uma configuração de *Auto Scalling* (ou seja, escalabilidade automática), onde é possível configurar gatilhos de consumo para aumentar ou diminuir a capacidade de processamento e memória da aplicação.

Dessa forma, é possível termos uma instância que poderá automaticamente aumentar sua capacidade de processamento nos momentos de maior uso, e da mesma forma reduzir quando não houver mais necessidade. Além disso, este serviço não possui custo adicional – a cobrança é feita apenas sobre os demais serviços da AWS que forem utilizados de acordo com a sua escala. Com isso, espera-se que não haja um grande aumento no custo, visto que nos momentos de menor uso teremos também um *scale-down* para uma instância com custo menor que o atual, compensando o *scale-up* nos momentos de pico.

Para realização destas configurações, serão necessários inúmeros testes de carga no novo ambiente com o uso do Apache jMeter (<https://jmeter.apache.org>). A cada nova rodada de testes devem ser analisadas as métricas de performance do serviço, para que seja possível otimizar os gatilhos e configurações de *Auto Scalling* do Elastic Beanstalk.

1. **Separar as instâncias de banco entre OLTP e OLAP**

Deve ser criada uma instância independente no EC2 apenas para OLAP, contendo apenas as tabelas de históricos do banco de dados original. Esta nova instância pode ter uma menor capacidade de processamento, visto que não há a necessidade de tantos acessos simultâneos. Ainda, ao separar as responsabilidades de OLAP e OLTP, se reduz a quantidade de *locks* por escrita/leitura nas mesmas tabelas, bem como reduz a concorrência de usuários nas mesmas conexões e recursos.

A geração de dados de histórico continua sendo feita no banco de dados principal, mantendo o fluxo normal da aplicação sem maiores impactos. Todas as noites deve ser executado um serviço - a ser desenvolvido - que copie os últimos históricos criados no dia, envie para a base de dados históricos, e apague-os da base principal.

A aplicação deve ser alterada para que possua 2 *strings* de conexão, uma para os dados atuais e execução de OLTP, e outra com dados históricos apenas para OLAP. Em todos os pontos da aplicação onde há leitura de dados históricos, como nos relatórios *Jasper Reports*, é necessária a alteração para que passe a utilizar a nova connection *string*. O usuário desta *connection string* deverá ser somente leitura, visando uma economia ainda maior de recursos.

Obs.: Caso algum dos relatórios necessite de dados em real-time, é possível criar uma *federated table* no servidor OLAP que aponte para a tabela de dados históricos na base de dados principal. Com isso, pode ser utilizado um “*UNION*” nas queries para juntar os dados históricos com os dados em *real-time*.