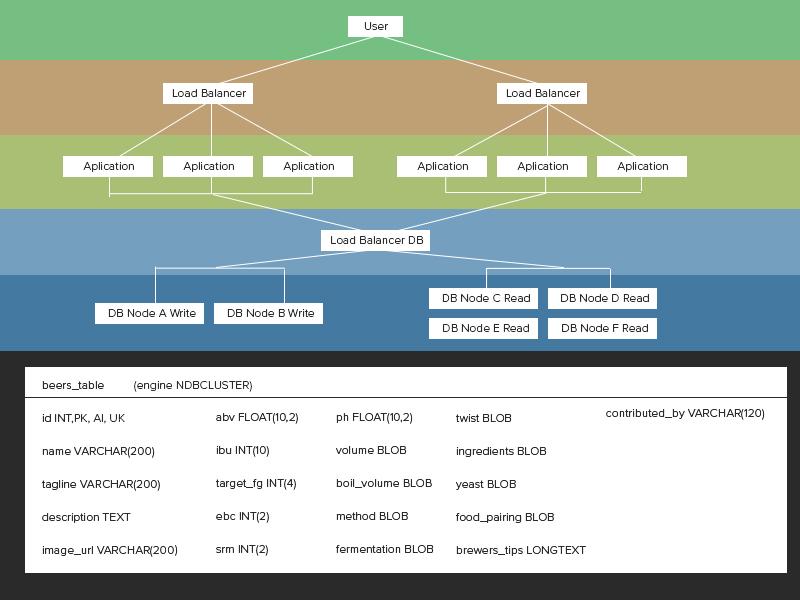
**Soluções para Armazenamento Catálogo de Cervejas**

**Modelo 1**

****

Neste primeiro modelo a solução a ser utilizada consiste em um cluster de dados Mysql, onde temos um Mysql Proxy como balanceador de carga(Load Balancer DB) e agente para tolerância a falhas(failover). O cluster de dados funciona transparente ao usuário final(USER) onde a aplicação em si(Application) trabalha em paralelo ao cluster sem a necessidade de saber sobre o uso do mesmo.

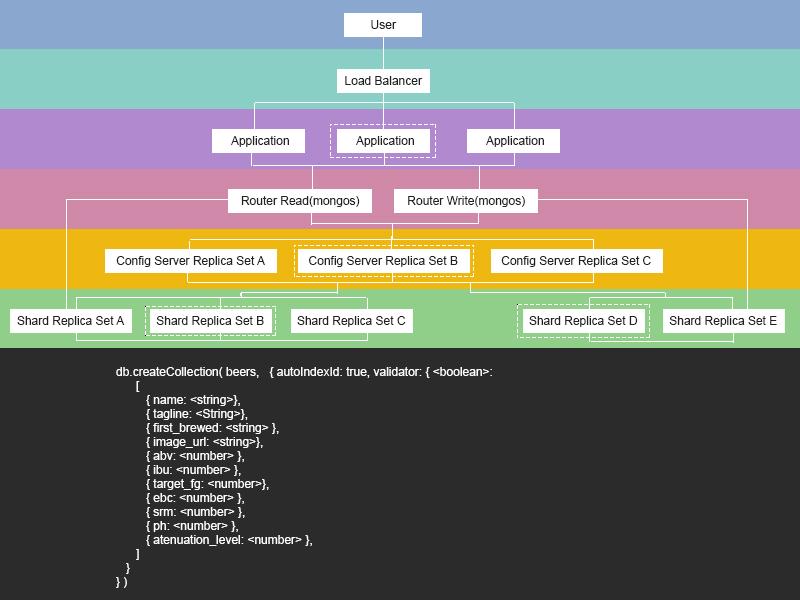
Após feito o balanceamento de carga na camada da aplicação o Mysql Proxy recebe as requisições, administra e separa o fluxo para leitura e escrita e se encarrega de encaminhar estas para os devidos grupos de nós respectivos(Node Write ou Read).

Como a escrita é moderada, criamos um grupo menor de nós(servidor MySQL) e o balanceador de carga irá dividir cada requisição para cada nó de escrita, cuidando também para checagem de status do mesmo.

No caso da leitura mais pesada, criamos um grupo maior de nós, aumentando-se a disponibilidade nos picos de fluxo e diminuindo a sobrecarga.

As tabelas a serem criadas e utilizadas pelo cluster devem ser configuradas com a engine DBCluster, que permite o uso neste cenário. No exemplo acima adicionamos a tabela beers\_table com uma chave única primária, auto\_increment, que recebe campos do tipo BLOB para armazenamento de maior quantidade de dados em formato binário, que trabalha em melhor performance em conjunto com o cluster.

**Modelo 2**

****

Neste segundo modelo a solução encontrada foi A criação de um cluster MongoDB utilizando docker. Esse modelo é orientado a coleções e documentos, diferente do primeiro orientado a tabelas. Após o balanceador de carga(Load Balancer) receber as requisições do usuário(User) e resolver a utilização da aplicação(Application), criamos uma interface(Router / mongos) entre a aplicação e o cluster de nós(shards). Criamos uma interface para leitura e outra para escrita independentes, onde a aplicação já se encarrega de separar as requisições de leitura e escrita na sua respectiva interface.

Criamos também uma camada para os metadados de configuração do cluster(config server réplica), com três servidores distintos para ampliar o failover de configuração. Essa camada também mapeia os dados para os blocos de bancos de dados (shards) e faz as regras de réplicas para garantir persistência e tolerância a falhas.

Cada instância MongoDB(shard) guarda uma parte da coleção de dados que serão utilizados pela aplicação. Em cada shard possui um principal e um secundário onde se fazem as réplicas e garantem a escala.

Como a leitura é mais pesada que a escrita, temos um conjunto maior de shards para esse router.

Ambos os modelos podem ser acessados via API proxy com o retorno em chave valor JSON na camada da aplicação.

Vantagens e Desvantagens

**Escalabilidade**

Levando em consideração a estrutura bem planejada e modelagem correta o Modelo 2 se torna melhor. Bases SQL as vezes apresentam dificuldade de tratamento em grandes demandas. Porém ambos podem trabalhar bem com escalabilidade em grandes demandas.

**Performance**

Não adianta ter uma solução apenas pela escalabilidade e com pouca performance pelo modelo estrutural. O ganho de performance é maior no modelo 1 por ser orientado a tabelas.

**Segurança**

Ambos são boas soluções nesse ponto e podem ser perfeitamente configurados com as devidas restrições pelo administrador da solução.

**Portabilidade**

Modelo 1 relacional seria o mais indicado para portabilidade em outros servidores e sistemas

**Redundância**

Como o Modelo 1 trabalha com entidade e relacionamento, pode acabar caindo em cenários onde tenham dados redundantes em consultas e armazenados. O Modelo 2 acaba sendo mais interessante por poder conter toda informação dos dados dentro do collection.

**Acesso aos dados**

Ambos podem trabalhar com API Proxy acessando pela aplicação e retornando resultados JSON para o usuário. Porém o modelo 2 por se tratar de bases novas no mercado, possuem ainda uma certa dificuldade no manuseio de todos da equipe, tipagem ainda fraca, porém com outras vantagens interessantes.