Projeto Final: Plataforma de Leilão Online Distribuída

Objetivo: Projetar, implementar, implantar e testar uma plataforma de leilão online distribuída utilizando orquestração com Kubernetes (K8s), Redis e Flask. O sistema deve ser escalável, tolerante a falhas e capaz de lidar com um grande número de usuários simultâneos com tempo de inatividade mínimo.

Visão Geral da Aplicação:

O sistema permitirá que os usuários criem leilões para itens, façam lances nesses itens em tempo real. Ele envolverá vários componentes funcionando juntos, cada um desempenhando uma tarefa específica. A plataforma deve suportar lances simultâneos e lidar com escalabilidade dinâmica.

Requisitos e Componentes Principais

1. Gerenciamento de Leilões (Flask + Redis)

- Os usuários poderão criar leilões para itens, e cada leilão pode ter os seguintes dados:
- Título do leilão, descrição, preço inicial e horário de término.
- Lista de lances feitos durante o leilão.
- Um indicador de ativo para determinar se o leilão está aberto para lances.
- Redis será usado para armazenar dados em tempo real, como leilões ativos, status do lance atual e a lista de maiores lances para cada leilão.
- O Pub/Sub do Redis será usado para notificar os usuários quando novos lances forem feitos ou quando os leilões estiverem prestes a terminar (atualizações em tempo real).

2. Lances em Tempo Real e Atualizações (Redis + Flask)

- Assim que um usuário fizer um lance, o sistema Redis Pub/Sub transmitirá o novo lance para todos os usuários que estão visualizando aquele leilão, garantindo atualizações em tempo real.
- Flask implementará a lógica de colocação de lances e validará se os lances são maiores do que o lance atual e se o leilão ainda está ativo.

3. Escalabilidade e Tolerância a Falhas (Kubernetes (K8s))

- O sistema deve ser projetado para escalar horizontalmente, com múltiplas instâncias dos serviços rodando sob Kubernetes (K8s) para garantir balanceamento de carga e tolerância a falhas.
- Contêineres do Docker devem ser orquestrados usando Kubernetes (K8s), o que permite escalar os serviços para cima/baixo com base na carga de tráfego.

- Kubernetes (K8s) será usado para gerenciar os ambientes de contêineres Docker. Cada Pod representará uma instância do serviço no seu sistema distribuído.

- Escalabilidade automática deve ser implementada para que, quando o sistema estiver sob carga pesada (por exemplo, quando muitos usuários estiverem fazendo lances ao mesmo tempo), novas instâncias dos servidores Flask ou Redis possam ser criadas automaticamente.

Estrutura Detalhada do Projeto:

1. Design da Aplicação Flask

- Implementar a aplicação Flask com as seguintes rotas e funcionalidades:
  - Rotas de Leilão:
    - `/create-auction`: Criar um novo leilão.
    - `/view-auctions`: Listar todos os leilões disponíveis com as informações do lance atual.
    - `/place-bid`: Fazer um lance em um leilão selecionado.
    - `/auction/{auction_id}`: Ver detalhes de um leilão específico e atualizações de lances em tempo real.
  - Para o front-end, pode-se usar um cliente simples em HTML/JavaScript que interage com a API para visualizar o leilão, os lances e as interações do usuário.

2. Redis para Lances em Tempo Real:

- Configurar Redis para armazenar:
  - Uma lista de leilões ativos (ID do leilão, título, lance atual).
  - Dados em tempo real do leilão usando Pub/Sub (para notificar os clientes sobre novos lances e atualizações de leilões).
  - Usar conjuntos ordenados (sorted sets) do Redis para gerenciar os valores dos lances para cada leilão.
- Utilizar Pub/Sub para transmitir os novos lances ou mudanças no estado do leilão para os clientes conectados.

3. Kubernetes (K8s) para Escalabilidade e Tolerância a Falhas:

- Configurar um cluster Kubernetes (K8s) com múltiplos nós para implantar os contêineres da aplicação Flask e Redis.
  - Contêineres Flask: Esses contêineres irão lidar com as requisições dos usuários, criar/gerenciar leilões e gerenciar a colocação de lances.
  - Contêineres Redis: Um ou mais nós Redis serão responsáveis por lidar com as atualizações em tempo real e cache.
  - Usar manifestos YAML do Kubernetes para definir a configuração multi-contêiner (Deployments, StatefulSets, Services).

4. Implantação com Kubernetes (K8s)

- Usar Kubernetes (K8s) para gerenciar os Pods que atuarão como diferentes instâncias no seu sistema distribuído.
- Cada nó do Kubernetes hospedará diferentes Pods: Pods para a aplicação Flask e Pods para o Redis.
- Os alunos devem gerenciar a rede interna do Kubernetes e configurar a comunicação entre os diferentes serviços rodando nos contêineres Docker dos Pods.

Entregáveis

1. Código-fonte: Uma base de código (github) totalmente funcional e documentada para a plataforma de leilões, utilizando Flask, Redis, Docker e Kubernetes (K8s).
2. Testes: Testes automatizados para verificar a funcionalidade da API REST, a lógica de lances e o desempenho do sistema sob carga.

• Em **dupla**. Apresentação no dia **16/12/2025** às **14:00 hrs**. Não há necessidade de entrega de relatório.

5. Agente de Inteligência Artificial (Worker)

Você deve implementar um agente de inteligência artificial.

Este agente deve ser um serviço contínuo (worker), implantado no Kubernetes como um `Deployment`, que ficará "ouvindo" por eventos de leilões finalizados.

Arquitetura de Eventos (Redis Pub/Sub):
A aplicação Flask (API) deve publicar uma mensagem em um canal Redis (ex: `leiloes_finalizados`) assim que um leilão terminar.
O Agente de IA (o worker) deve estar inscrito (subscribe) neste canal.

Ao receber uma mensagem do Redis, o agente deve buscar os dados completos daquele leilão e usar uma IA Agentica para executar as seguintes tarefas. Escreva os prompts para o LLM gerar cada saída:

1. Gerar relatórios sobre os resultados do leilão.
   Exemplo de Prompt (para o LLM): “Baseado no resultado do leilão, gere um resumo bem completo do leilão, destacando o item, valor final e número de lances.”

2. Enviar estes relatórios por e-mail.
   Exemplo de Prompt (e-mail do vencedor): “Escreva um e-mail amigável parabenizando[EMAIL_VENCEDOR] pela vitória no leilão do item [NOME_ITEM] pelo valor de [VALOR_FINAL]. Informe os próximos passos para pagamento.”

3. Fazer posts com os resultados do leilão em um canal do Discord.

Exemplo de Prompt (post no Discord): “Crie um post no canal de Discord [NOME_CANAL] anunciando que o item [NOME_ITEM] foi arrematado por [VALOR_FINAL] pelo vencedor [NOME_VENCEDOR]!”

Crie os manifestos `Deployment` para o agente e o `Secret` para gerenciar as credenciais (API do LLM, e-mail, Webhook do Discord) no Kubernetes.