Cloud Computing Systems – 2021/22

Projeto 1

André Santos nº55415

Daniel João nº55797

Martim Andersen nº55104

Introdução

O objetivo deste projeto é usar os serviços de computação em cloud da Azure para criar o back-end de um sistema como o discord, o qual é altamente escalável, rápido, fiável, georreplicado e com uma grande disponibilidade.

O sistema é composto por um conjunto de utilizadores que podem criar canais para troca de mensagens entre si.

Para atingir com sucesso o objetivo pretendido, foram usados os seguintes serviços da Azure:

- Blob Storage: para armazenar os arquivos de média dos usuários;
- Cosmos DB: para armazenar de forma persistente as informações dos utilizadores, as mensagens e os canais do sistema;
- Redis: foi utilizado como cache para aumentar a performance no acesso a dados como as mensagens, os utilizadores e os canais;
- Pesquisa Cognitiva: usada para fazer pesquisas de texto no conteúdo das mensagens;
- Funções da Azure: usadas para replicar geograficamente os recursos de média e também para limpar periodicamente as informações relacionadas aos utilizadores que são removidos do sistema.

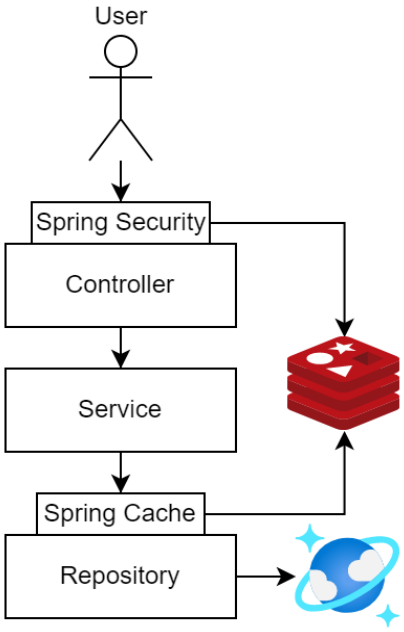
Design

O sistema é composto por três camadas principais: rest (controllers), application (services) e infrastructure (repositories).

O tratamento dos pedidos http é efetuado pelos controllers, que delegam a informação do pedido recebido aos services, que contêm a lógica da aplicação e interagem com os repositories.

Para suportar o mecanismo de autenticação recorre-se ao Spring Security configurado para utilizar redis para armazenamento das sessões e o mecanismo de caching é suportado por anotações de Spring Cache nos repositories.

A cadeia de chamadas da aplicação principal está ilustrada no esquema à direita.



Implementação

A geo-replicação dos Blob Storages é realizada com o apoio de funções azure com BlobTriggers. A eliminação dos utilizadores é realizada assincronamente por uma função azure com TimerTrigger.

A gestão da cache é feita com base em TTLs e adição/remoção de entradas aquando de uma alteração que o justifique.

É utilizado o Azure Cognitive Search para realizar pesquisas sobre o conteúdo das mensagens presentes no sistema.

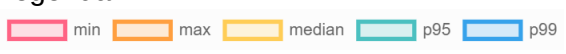
Avaliação

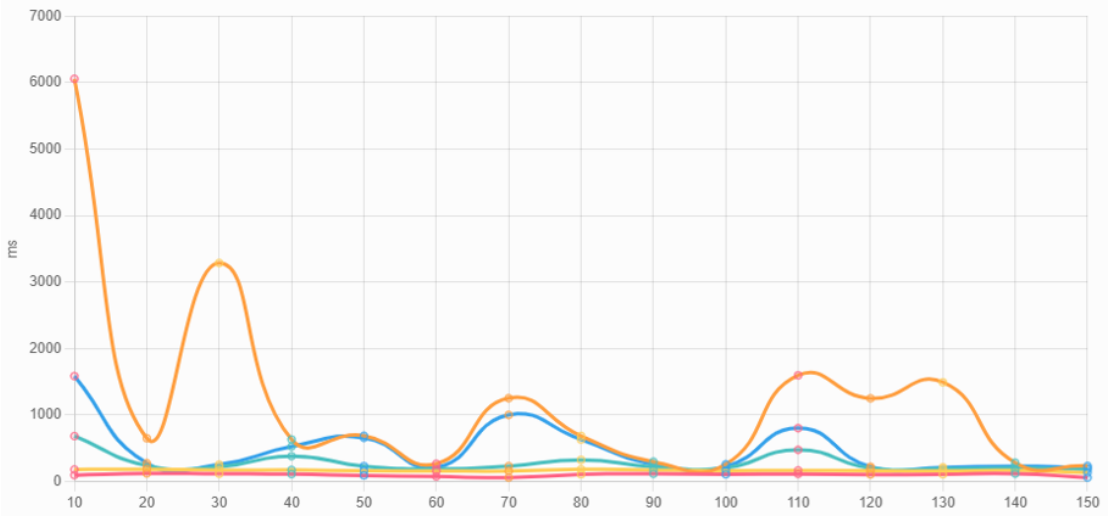
Os gráficos seguintes (latência/tempo) foram obtidos executando os seguintes testes artillery:

- create-messages.yml (arrivalCount: 70, duration: 1)
- workload1.yml (arrivalRate: 50, duration: 20)

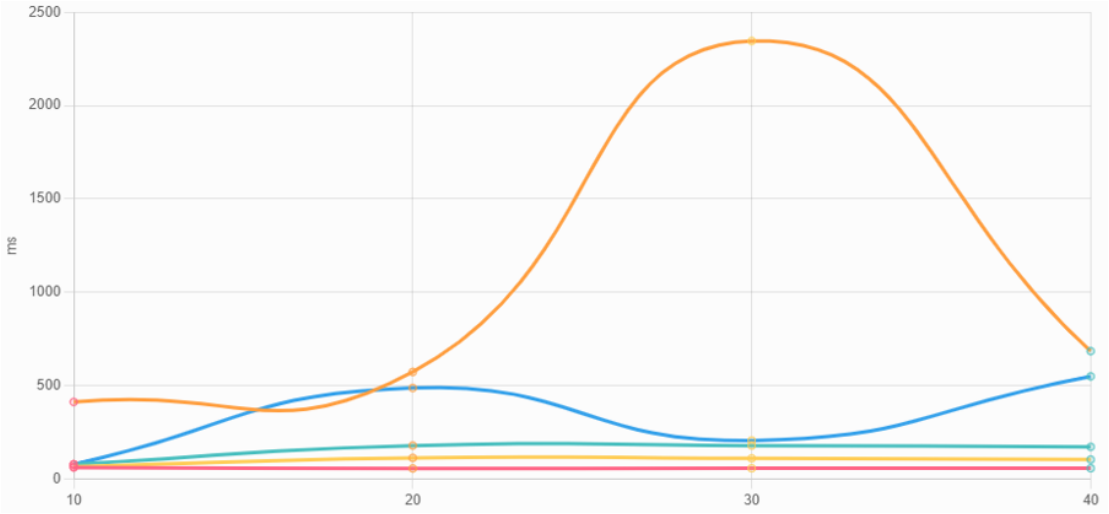
Estes valores foram escolhidos para colocar o deployment em West Europe com cache sem geo-replicação com load elevado mas gerível.

Legenda:

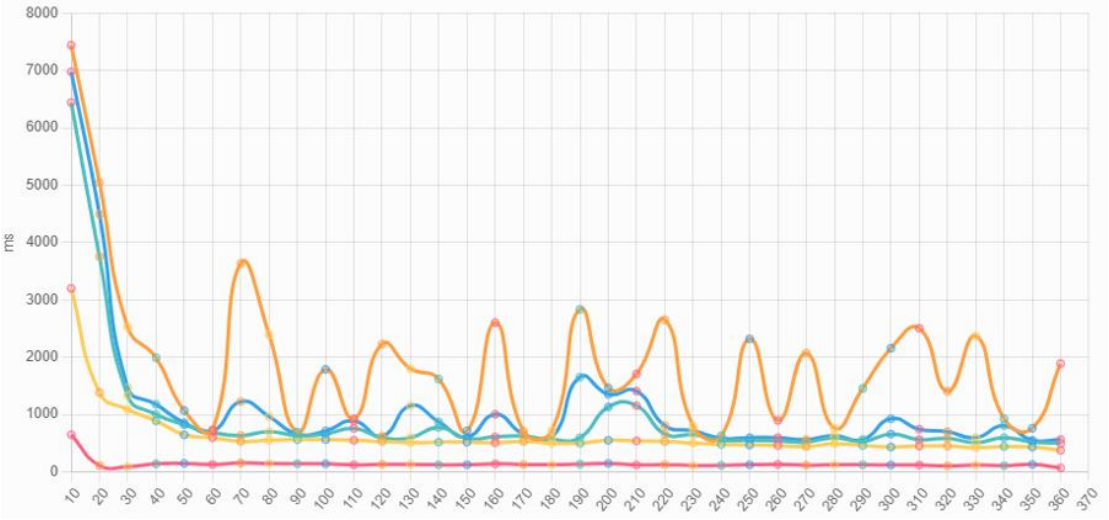




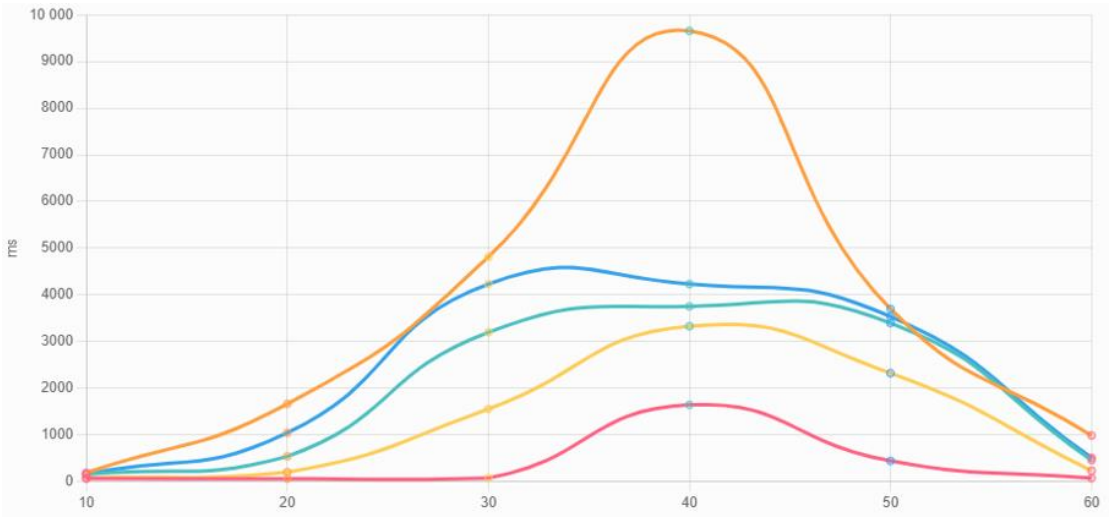
West Europe (cached) - create-messages.yml



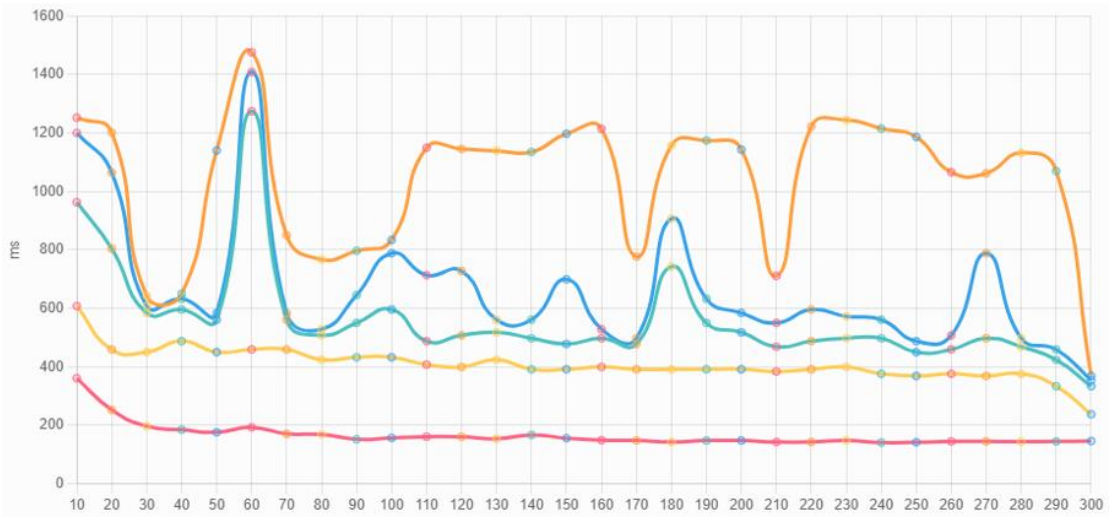
West Europe (cached) - workload1.yml



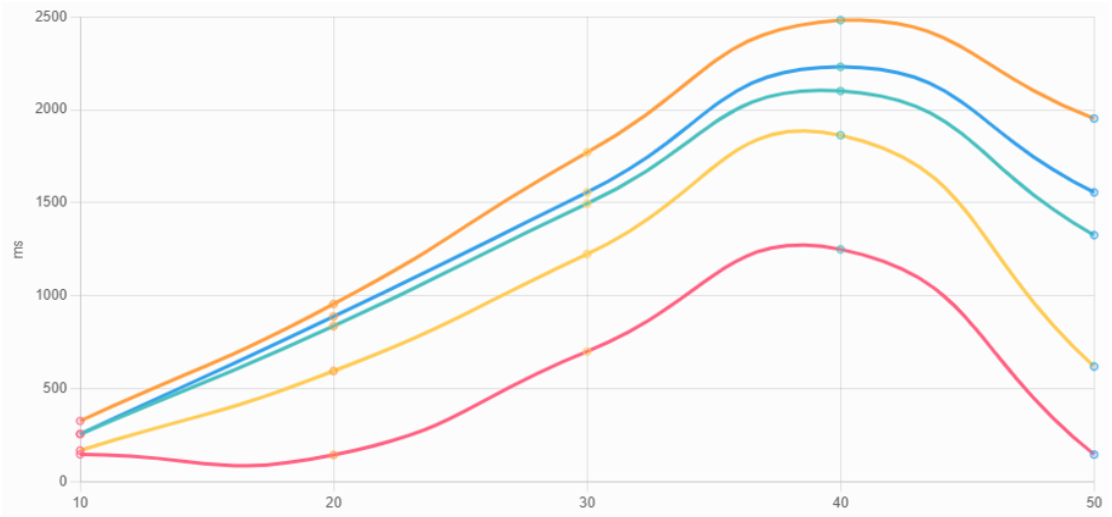
West Europe (not cached) - create-messages.yml



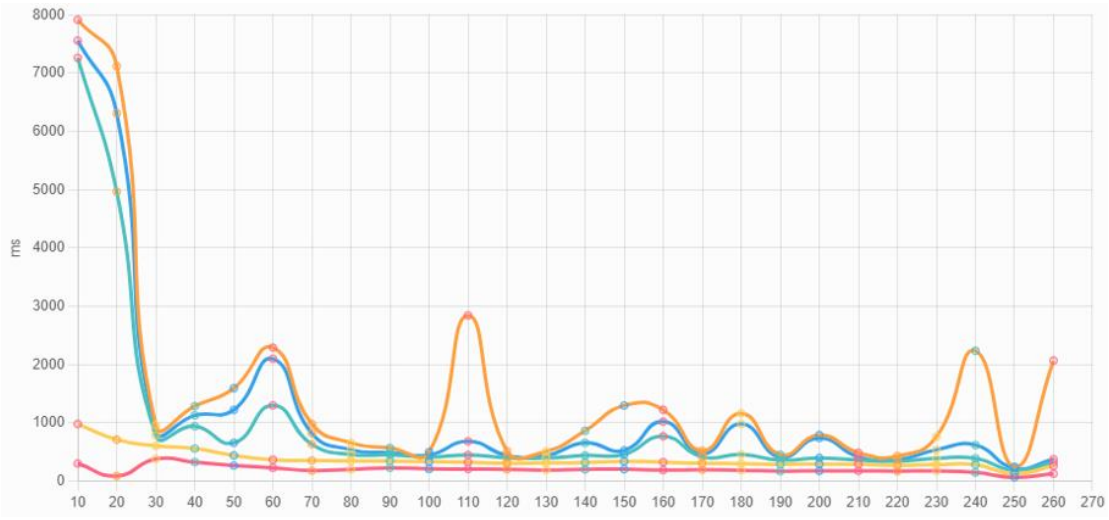
West Europe (not cached) - workload1.yml



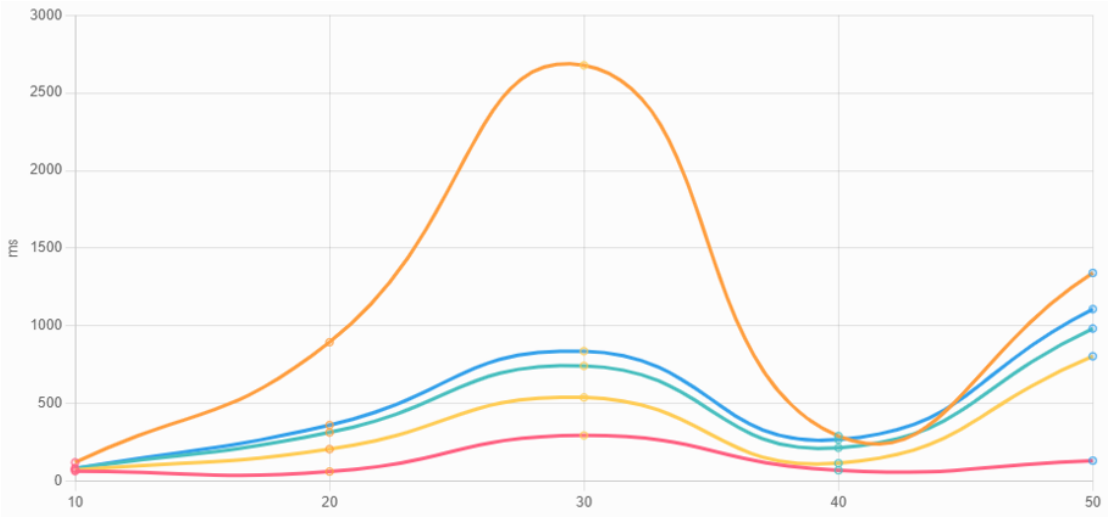
West Central US (cached) - create-messages.yml



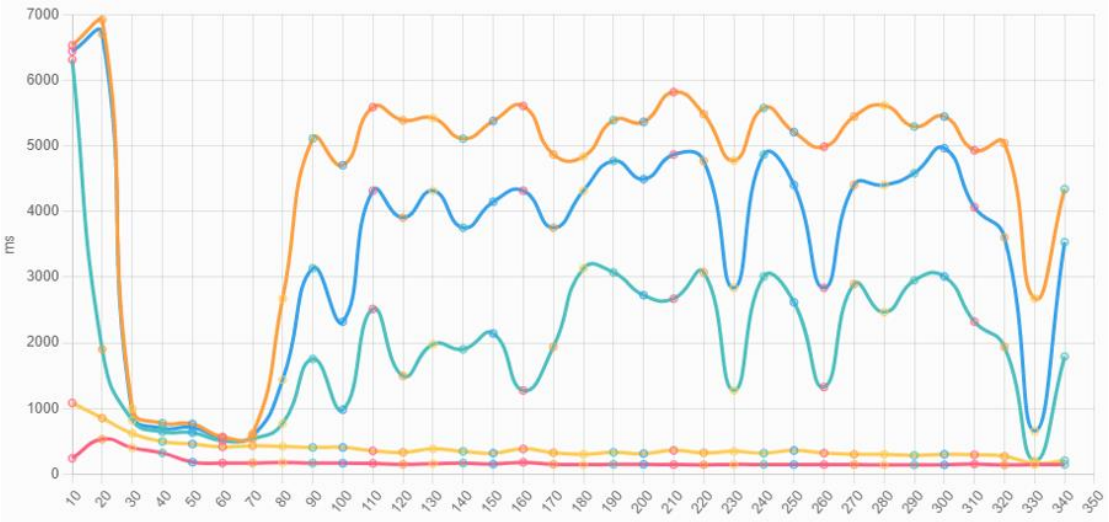
West Central US (cached) - workload1.yml



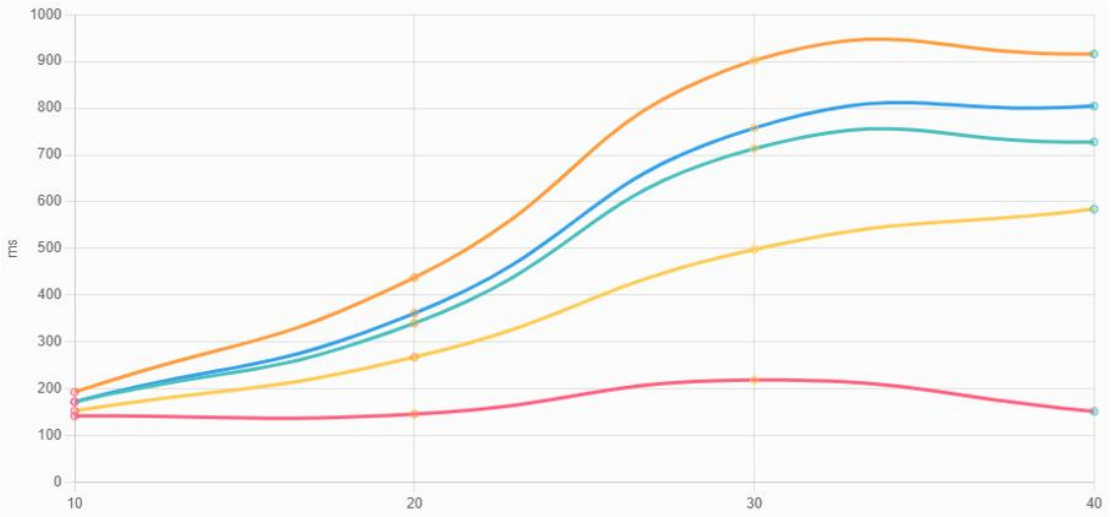
West Europe (cached and geo replicated) - create-messages.yml



West Europe (cached and geo replicated) - workload1.yml



West Central US (cached and geo replicated) - create-messages.yml



West Central US (cached and geo replicated) - workload1.yml

Apesar de as escalas serem bastante diferentes devido à presença de outliers, as observações que podem ser feitas, quer para o create-messages.yml como para o workload1.yml são as seguintes:

- Entre West Europe (cached) e West Europe (non cached) verifica-se que, para a mesma carga, a versão com cache tem uma capacidade de resposta muito superior em relação à outra;
- De West Europe (cached) para West Central US (cached) verifica-se um aumento geral da latência e da sua variância;
- Entre as versões sem e com geo replicação nota-se que a presença da mesma tem um custo no aumento da latência.

É de se notar que o teste “West Central US (cached) - workload1.yml” é provavelmente um outlier quando é tido em conta o resultado de “West Central US (cached and geo replicated) - workload1.yml”, que foi facilmente reproduzido.

Conclusões

Dos resultados obtidos é possível concluir que o uso de cache Redis é muito benéfico para o desempenho da aplicação. Adicionalmente, nota-se que o desempenho é superior quando o cliente faz pedidos a um servidor mais próximo.

Para tornar este último aspeto acessível a todos os utilizadores é necessário recorrer a geo-replicação, que acarreta algum overhead mas provavelmente (não foi testado) melhora o desempenho para clientes em localizações remotas.