Sistemas Distribuídos 2023/24

Trabalho Prático - Cloud Computing

Trabalho realizado por

João Pedro da Rocha Rodrigues - a100896

João Pedro Mota Baptista - a100705

João Andrade Rodrigues - a100711

Mateus Lemos Martins - a100645

Índice

Introdução

O presente relatório aborda a arquitetura, a implementação e a avaliação de um serviço de Cloud Computing com a funcionalidade de Function-as-a-Service (FaaS). Este serviço foi concebido para atender a uma variedade de requisitos, desde a autenticação do utilizador até a execução distribuída de tarefas, proporcionando uma abordagem eficiente e escalável para o processamento de código.

O principal objetivo do projeto é oferecer uma plataforma robusta, capaz de receber pedidos de tarefas, processá-las de forma eficiente e devolver os resultados aos utilizadores. O serviço é projetado para otimizar a utilização dos recursos disponíveis, especialmente a memória, e garantir uma execução ordenada e justa das tarefas, evitando bloqueios indefinidos.

O serviço possui alguns tipos de funcionalidades. A funcionalidade básica inclui autenticação de utilizadores, pedidos de execução de tarefas e consulta do estado atual do serviço. A funcionalidade avançada expande essas capacidades para permitir o envio assíncrono de pedidos e garantir uma ordem de execução que evita bloqueios indefinidos.

Serão apresentadas a biblioteca do cliente (Client) e da interface do utilizador (ClientUI), demonstrando, através de testes realizados, como múltiplos utilizadores podem interagir com o serviço de forma eficaz.

Por fim, o relatório será concluído destacando possíveis melhorias futuras. Este projeto representa um esforço significativo para criar uma infraestrutura de computação em nuvem flexível e escalável, fornecendo uma base sólida para futuros desenvolvimentos nesta área.

Implementação

ThreadExecutor

O ThreadExecutor é uma parte fundamental da implementação de sistemas que envolvem o processamento concorrente em Java. Ele oferece uma abstração conveniente para a execução assíncrona de tarefas por meio da gestão de threads. O conceito central por trás do ThreadExecutor é fornecer uma interface de alto nível para gerir a execução de unidades de trabalho de forma eficiente e controlada.

- Pool de Threads

O ThreadExecutor geralmente opera com um pool de threads reutilizáveis. Esse pool permite a execução eficiente de várias tarefas.

- Fila de Tarefas

As tarefas a serem executadas são mantidas em uma fila. O ThreadExecutor gere a execução dessas tarefas, garantindo uma ordem adequada e evitando a concorrência descontrolada.

- Controlo de Recursos

Oferece controlo sobre a quantidade de threads simultâneas, o que é crucial para evitar congestionamentos e garantir um uso eficiente dos recursos do sistema.

- Configurações de Thread

Permite configurações flexíveis, como o número máximo de threads no pool, políticas de rejeição de tarefas e estratégias de execução.

- Facilidade de Uso

Oferece uma interface simplificada para os desenvolvedores submeterem tarefas para execução assíncrona, ocultando detalhes complexos de gestão de threads.

O uso adequado do ThreadExecutor é essencial para melhorar a eficiência e o desempenho de aplicações que envolvem operações concorrentes. É particularmente útil em situações em que a execução paralela de tarefas pode melhorar a capacidade de resposta e a escalabilidade de um sistema.

CentralServer

A classe CentralServer representa a implementação de um servidor central num sistema distribuído. O servidor é projetado para lidar com diversas operações, incluindo o registo de utilizadores, o login, a execução de tarefas e as consultas de estado do sistema. Abaixo estão os principais componentes e funcionalidades da implementação.

- Inicialização

O servidor é inicializado por meio de um construtor que recebe um número de porta como parâmetro. Ele cria um ServerSocket para esperar por conexões de clientes e um ThreadExecutor para gerenciar threads concorrentes.

- ClienteHandler

A classe interna ClientHandler implementa a interface Runnable e é responsável por lidar com a comunicação de um cliente específico. Ela gere a autenticação, execução de tarefas e outras operações.

- Segurança de Threads

O código utiliza Locks (inputLock e outputLock) para garantir a segurança de threads durante operações de leitura e escrita. Isso evita as disputas entre as threads em ambientes concorrentes.

- Operações do Cliente

O código do ClientHandler processa diferentes tipos de pedidos dos clientes, como registo, login, execução de tarefas, consulta de estado e logout. Cada tipo de pedido é tratado por uma instância de FuncExecutor, uma classe interna que implementa a interface Runnable.

- Autenticação

O servidor verifica as credenciais dos utilizadores durante as operações de registo e login, utilizando um mecanismo básico de autenticação.

- Execução de Tarefas

A classe lida com a execução de tarefas enviadas pelos clientes. A execução é realizada pela classe JobFunction. Antes de executar uma tarefa, o servidor verifica a disponibilidade de memória para garantir que a tarefa pode ser processada.

- Comunicação com Clientes

A comunicação entre o servidor e os clientes ocorre por DataInputStream e DataOutputStream. Os dados são lidos e escritos utilizando os Locks para evitar problemas de concorrência.

- Feedback ao Cliente

O servidor fornece feedback aos clientes sobre o sucesso ou falha de operações, garantindo uma interação clara entre o servidor e os utilizadores.

Client (Biblioteca)

A classe Client representa a implementação de um cliente para interagir com o servidor central num sistema distribuído. Aqui estão os principais componentes e funcionalidades desta implementação:

- Comunicação com o Servidor

O cliente estabelece uma conexão com o servidor central usando a classe Socket para transferência de dados.

- Operações de Entrada/Saída

A classe utiliza DataInputStream e DataOutputStream para se comunicar com o servidor, recorrendo aos Locks para evitar conflitos entre threads.

- Registo e Login

O cliente pode realizar operações de registo e login, enviando as suas credenciais (nome de utilizador e palavra-passe) ao servidor. O resultado destas operações é interpretado a partir das mensagens recebidas do servidor.

- Execução de Tarefas

O cliente pode enviar tarefas para o Servidor. O resultado da execução da tarefa é processado pelo servidor, e o cliente recebe feedback sobre a conclusão da tarefa, guardando-a num ficheiro, numa pasta com o seu <username>.

- Consulta de Estado

O cliente pode solicitar informações sobre o estado atual do sistema (memória disponível e número de tarefas pendentes). O servidor responde fornecendo essas mesmas informações.

- Logout

O cliente pode efetuar logout, encerrando a conexão com o servidor. Isso é feito enviando uma mensagem específica ao servidor, que então desconecta o cliente.

ClientUI

Este código Java representa a implementação de uma interface de utilizador para interagir com o servidor central num sistema distribuído. Abaixo estão os principais componentes e funcionalidades desta implementação.

- Início e Configuração

A classe ClientAPI contém o método main, que inicializa um objeto Client para comunicação com o servidor. Também configura um ThreadExecutor para lidar com operações assíncronas.

- Registo ou Login

Solicita ao utilizador que escolha entre registo ("n") ou login ("y"). Dependendo da escolha, o código executa as funções correspondentes (register ou login).

- Executor de Opções

A classe interna OptionExecutor implementa a interface Runnable e é responsável por executar as opções escolhidas pelo utilizador. As opções incluem executar uma tarefa, consultar o estado do serviço ou sair.

- Execução de Tarefa

Para a opção "Executar Tarefa", o utilizador escolhe um ficheiro de tarefa da lista disponível no cliente (taskFiles). A execução da tarefa é assíncrona, sendo gerida pelo ThreadExecutor.

- Consulta de Estado do Serviço

Para a opção "Consulta de Estado", o cliente solicita e exibe informações sobre o estado do serviço, incluindo memória disponível e número de tarefas pendentes.

- Logout e Saída

A opção "Sair" efetua logout do cliente e encerra a execução do programa.

- Segurança de Threads

Usa a variável proceed para coordenar a execução assíncrona das tarefas. Garante que o utilizador só pode escolher uma nova opção após dar todos os inputs necessários a essa mesma tarefa.

Arquitetura da Solução

Os seguintes diagramas demonstram o funcionamento do programa, nos variados contextos, depois da conexão ao Socket.

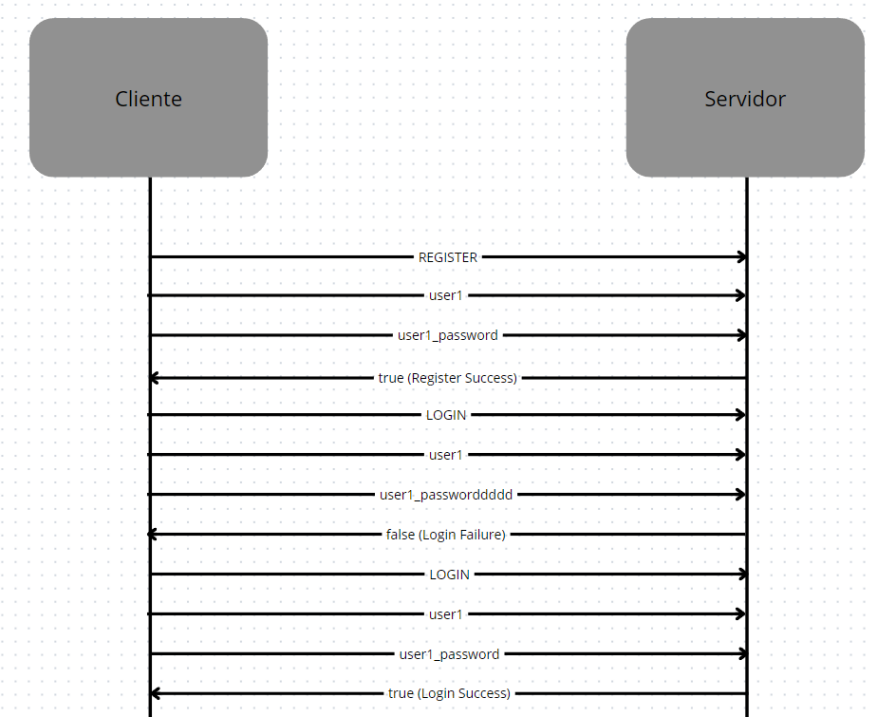


Figura 1. Autenticação do Cliente

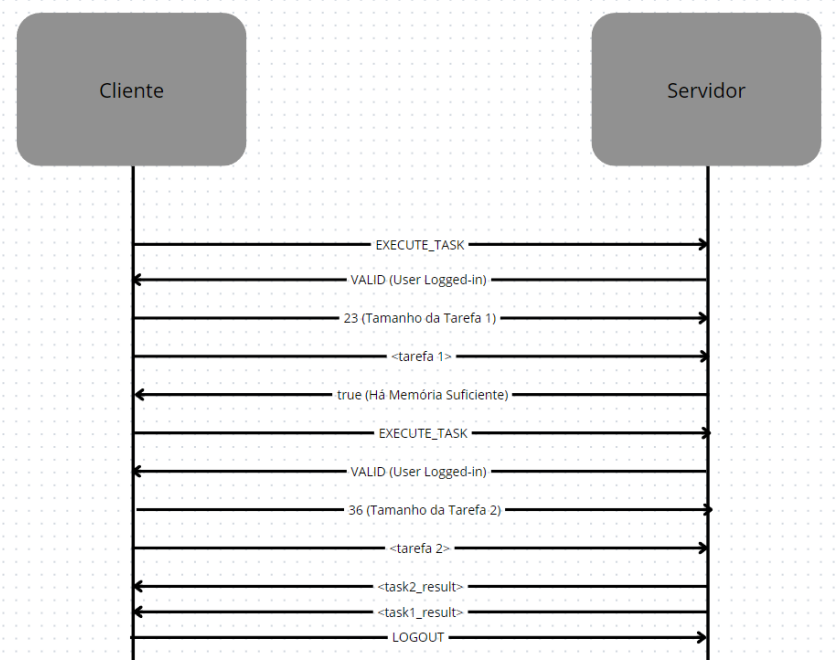
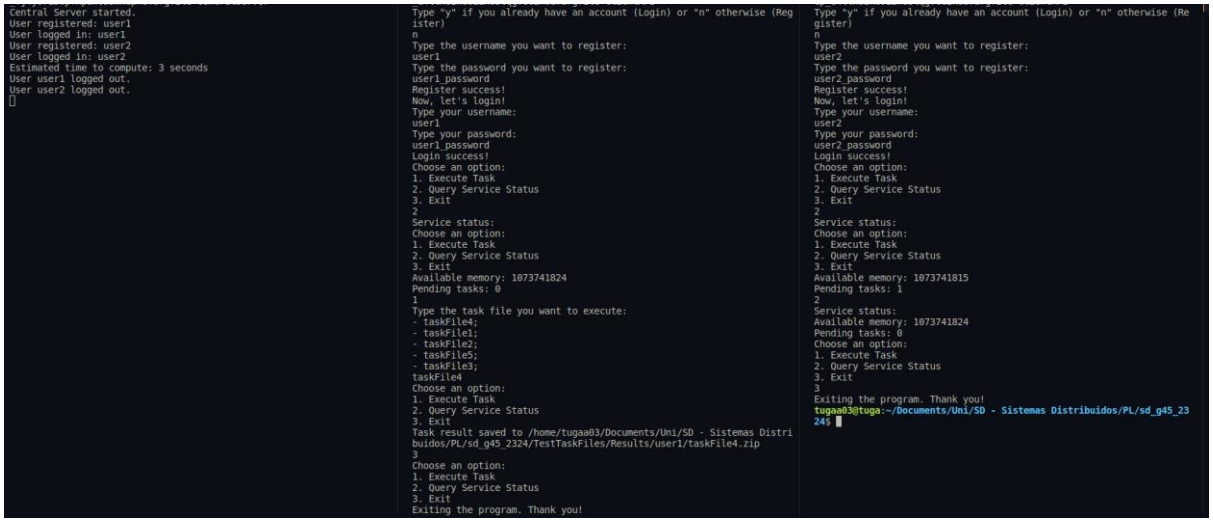


Figura 2. Pedido de Execução de 2 Tarefas ao mesmo tempo.

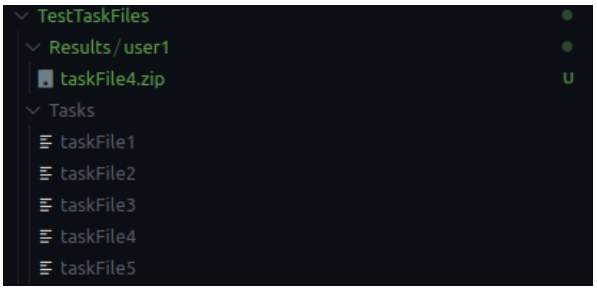
Testes

Para verificar o funcionamento e eficiência da nossa implementação, testamos a utilização de um servidor ligado a múltiplos clientes, tal como demonstrado seguidamente:

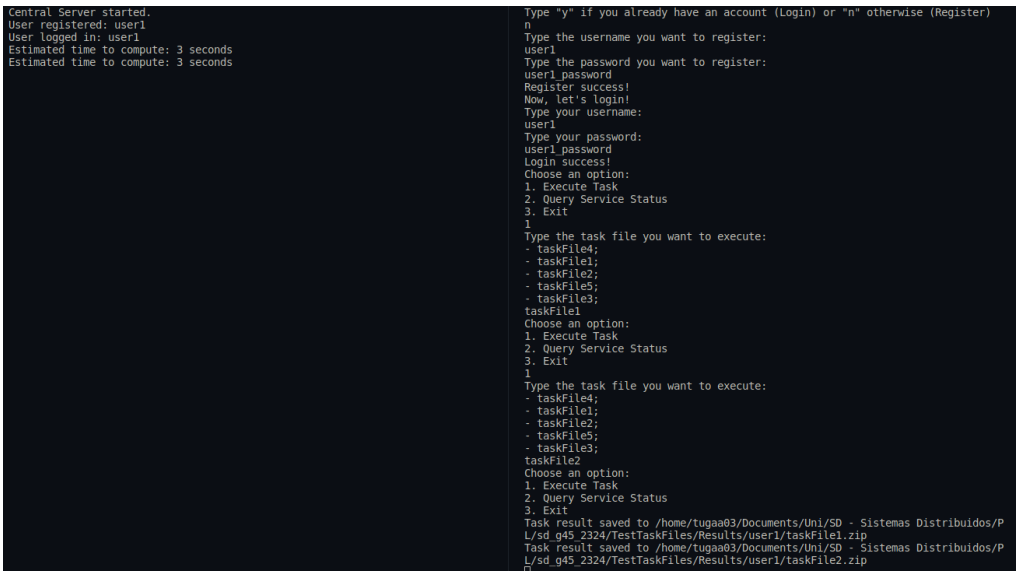
Teste 1



- Inicia-se o servidor, e 2 clientes. Registam-se os dois clientes com os usernames "user1" e "user2", pedindo o estado do servidor no primeiro utilizador. Pediu-se a execução de uma tarefa a partir do cliente "user1" e, enquanto a mesma estava a ser executada pelo Servidor, pediu-se o estado do servidor no “user2”. Como era esperado, verificou-se que estava 1 tarefa pendente (a executar), sendo que a memória baixou 9 bytes (tamanho da tarefa pedida pelo “user1”). No final, o resultado é guardado na pasta “TestTaskFiles/Results/user1”, como é mostrado na figura abaixo:

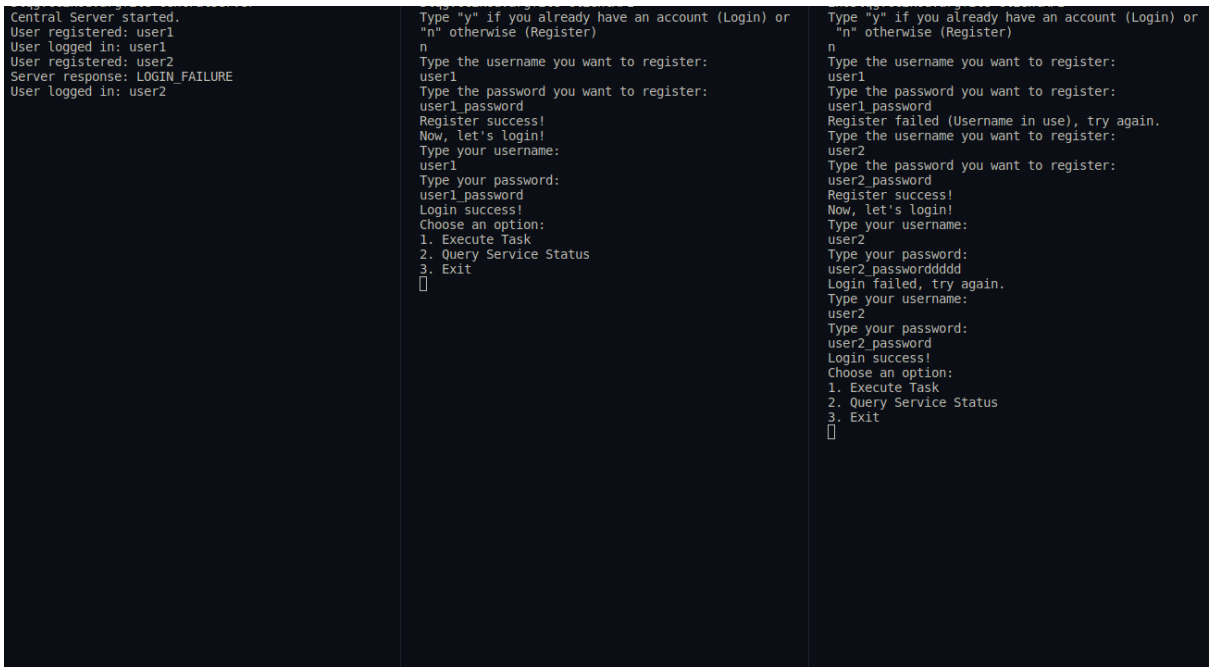


Teste 2



- Neste segundo teste, verificamos a capacidade de executar várias tarefas concorrentemente, no mesmo cliente. Como é verificado, no final as duas são guardadas, após a execução das mesmas, na sua pasta de ficheiros.

Teste 3



- Neste último teste foi testada a autenticação, o registo dos clientes e resolução de erros dos mesmos. Como é possível verificar, não é possível um cliente se registar com o mesmo username de outro, já registado, sendo que depois é necessário inserir a palavra-passe correta.

Conclusões e Trabalho Futuro

Em suma, obtivemos mais conhecimento acerca de programação num ambiente Cliente-Servidor multi-threaded em Java, incluindo Locks, Barreiras, Fluxos de Saída e Entrada de dados (Data[Input|Output]Stream), bem como na implementação de APIs e Interface do Utilizador.

Tentamos também minimizar ao máximo o número de Threads acordadas, ainda que não seja possível (a nosso ver), ter menos do que as atuais, pois para cada Cliente será necessário pelo menos 1 thread acordada para cada tarefa em execução que ele tem.

Acreditamos ter alcançado maior parte dos requisitos para este trabalho, tendo de melhorar em alguns aspetos como a Implementação Distribuída, pelo que preferimos priorizar a resolução de alguns bugs do programa.