Trabalho Prático 3 - Task 3 - Computação em Nuvem

Francisco Teixeira Rocha Aragão - 2021031726
Gabriel Pains de Oliveira Cardoso - 2021096887

13 de dezembro de 2023

1 Introdução

Na **Tarefa 1** o módulo desenvolvido executa no *runtime* fornecido para este trabalho. Na **Tarefa 3**, é desenvolvido um *runtime* para substituir o fornecido, de forma a manter a maior compatibilidade possível. São discutidas a implementação e as escolhas realizadas, bem como a compatibilidade entre os *runtimes*.

2 Implementação

O *runtime* desenvolvido deve ser compatível com funções da mesma assinatura que a usada na **Tarefa 1**. Ou seja, funções do tipo:

def handler(input: dict, context: object) → dict[str, Any]

Ainda, a aplicação do novo *runtime* é feita ao mudar a linha 18 do arquivo *deployment.yaml* da **Tarefa 1** de *lucasmsp/serverless:redis* para *gabrielpains/serverless:redis*. Isto é, o *runtime* desenvolvido deve ser completamente compatível com a especificação *Kubernetes* original.

Dessa forma, as variáveis de ambiente disponíveis nesse arquivo são lidas pelo *runtime* desenvolvido, a fim de estabelecer a conexão com o *Redis* e executar suas funções normalmente.

O funcionamento se resume a algumas etapas, a classe *Context* é inicializada e contém os mesmo campos da classe disponível no *runtime* original, em seguida no *loop* principal, é verificado se existem novos dados disponíveis no *Redis*, se sim, a função *handler* do usuário é chamada com estes dados e o contexto. Logo após, é verificado se o usuário retornou um dicionário com P + 2 entradas, onde P é o número de CPUs da máquina, assim como mencionado no enunciado do trabalho. Em caso afirmativo, a saída do usuário é adicionada ao *Redis*, o contexto é atualizado com o tempo da execução atual e o campo *env* da classe *Context* é modificado naturalmente devido à política de passagem de parâmetro por referência em Python. Em seguida o *loop* se resume à espera de novos dados e o processo se repete, isso significa que a chamada da função *handler* é baseada no evento de mudança de dados disponíveis para o *runtime* no *Redis*.

3 Análise de Compatibilidade

A fim de analisar a compatibilidade, o módulo da **Tarefa 1** foi executado novamente com o novo *runtime*, o *dashboard* da **Tarefa 2** também foi iniciado para testar a interação entre eles.

Durante os testes, não foi percebida nenhuma interrupção ou comportamento inesperado, o *dashboard* continuou atualizando normalmente com os novos dados. Ainda sobre o *dashboard*, foi observado que a taxa de atualização de dados continuou 5 segundos, assim como foi observado na **Tarefa 2**. Como o *dashboard* e o *runtime* atualizam baseados em eventos, isso leva à conclusão que o intervalo de 5 segundos é proveniente da atualização dos dados no *Redis* que são lidos pelo *runtime*. Ou seja, o *runtime* e o *dashboard* são adaptativos quanto ao intervalo de atualização, dependendo apenas da disponibilidade dos dados originais.

Por fim, o runtime desenvolvido conta com várias bibliotecas mais comuns de *Python* pré-instaladas como: *matplotlib, numpy, pandas, requests, scikit-learn e tqdm*. Dessa forma, é expandida a compatibilidade com as funções dos usuários, já que apesar da assinatura ser a mesma, as operações do código podem variar de usuário para usuário.

4 Conclusões

O *runtime* desenvolvido foi capaz de replicar com sucesso o comportamento do *runtime* fornecido. É seguida a mesma especificação do arquivo *deployment.yaml* original e é fornecido suporte as bibliotecas mais comuns usadas em ambientes Python. Dado que o usuário continue aderindo à estas especificações e a assinatura de função esperada, além de retornar a quantidade de valores esperados no dicionário, o *runtime* desenvolvido é compatível com as necessidades deste usuário.