Relatório TP3

Diogo Oliveira Neiss - Pós graduação
https://github.com/diogoneiss/kubernetes-serverless-runtime

Parte 1

Função principal disponível em /`serverless runtime/usermodule.py`, arquivos do configmap disponíveis na mesma pasta

A abordagem para manter o estado consistiu em manter um dicionário de cpu x listas de tuplas no env. Cada tupla consiste em timestamp e valor, então para calcular a média móvel basta inserir o valor atual na lista, filtrar tuplas no intervalo de 60 segundos e salvar novamente no env.

Com o estado salvo, basta somar os itens da lista e dividir pela quantidade para calcular a média móvel

Parte 2

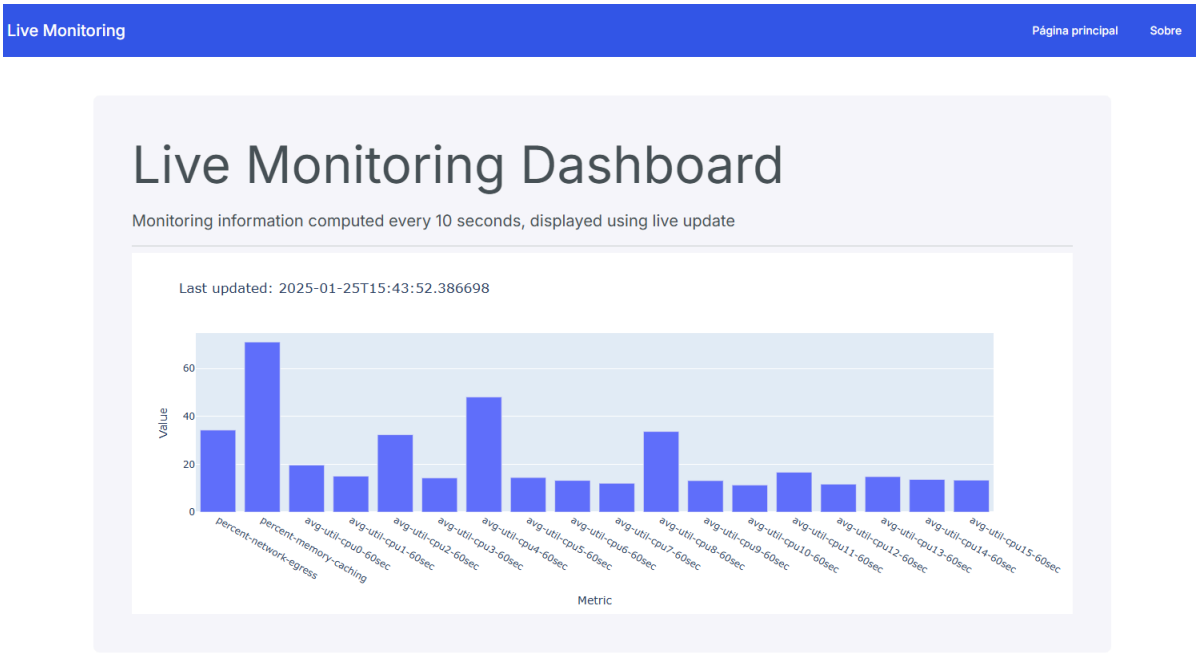
Todos os arquivos disponíveis em `/dashboard`, os arquivos de configmap e service em `/kubernetes`

As métricas salvas no passo 1 consistem nas especificadas no enunciado do trabalho e *timestamp.* Optei por incluir todas as métricas menos timestamp, já que são percentuais, permitindo que fiquem no mesmo eixo.

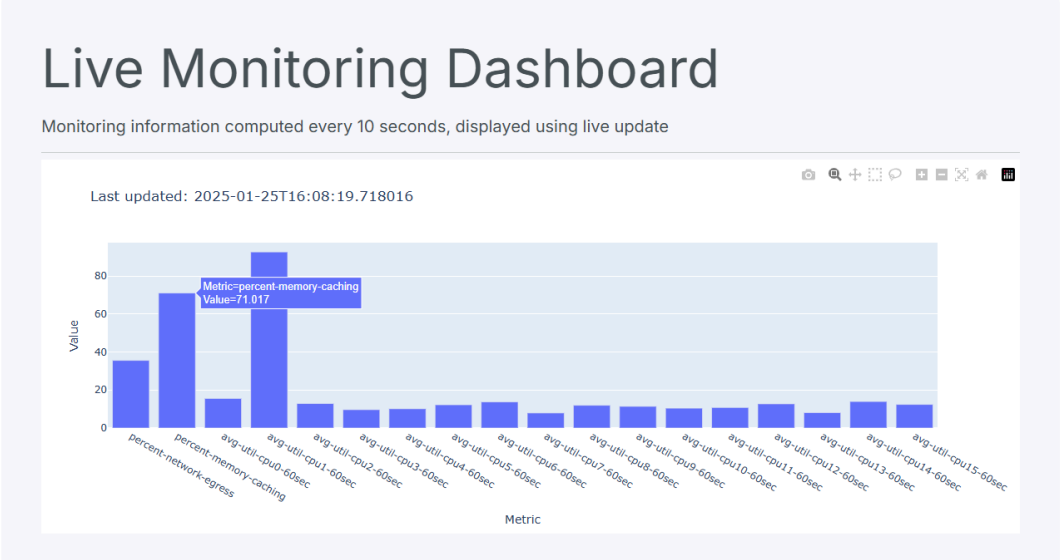
As métricas são

- percent_network_egress: ficou relativamente baixo nos periódos de análise, indicando que era mais comum receber muitos bytes que enviar.
- percent_network_caching: consistentemente acima de 50%, indicando eficiente uso de memória, calculado como `((virtual_memory-cached + virtual_memory-buffers) / virtual_memory-total) * 100.0.`
- avg_util_cpu*N*_60seg, para cada cpu nas métricas, onde *N* é o índice da cpu. Foi observado que era muito mais comum o uso intenso de apenas um dos núcleos, com o resto subutilizado.

O timestamp é mostrado acima do gráfico, permitindo visualizar quando elas foram geradas e acompanhar o live reload.



Dashboard após atualização:



O dashboard foi criado com plotly e um componente de reload periódico foi utilizado para recuperar os dados do redis.

O dashboard está disponível na porta 31001, com os arquivos service e deployment dentro da pasta kubernetes na raiz do repositório.

Parte 3

Código disponível em `/serverless runtime`, deployment na mesma pasta.

O design do meu runtime foi projetado para interoperar o máximo possível com o deployment.yaml fornecido, utilizando a mesma api de configmaps e variáveis de ambiente, sendo diferente apenas na imagem utilizada e nomes.

Minha abordagem foi iniciar o runtime com a validação de que as variáveis de ambiente foram corretamente inseridas, em seguida verificar se o módulo python a ser executado existia.

Por fim, o runtime cria o contexto persistente e entra em um loop periódico, recuperando informações do redis, executando a função com o contexto persistente, salvando o retorno no redis e dormindo por 5 segundos.

Entrega contínua

Tentei utilizar o ArgoCD e configurei o repositório de acordo, porém tive um erro de permissão ao utilizar, então optei por fazer o deployment manualmente