**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS  
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**

**Pós-graduação *Lato Sensu* em Inteligência Artificial e Aprendizado de Máquina**

**FRANDBERTO DE CASTRO FAÇANHA JÚNIOR**

**CLASSIFICAÇÃO DE FLAVORS PARA NUVEM PRIVADA BASEADA EM KUBERNETES**

Fortaleza

Novembro de 2022

**FRANDBERTO DE CASTRO FAÇANHA JÚNIOR**

**CLASSIFICAÇÃO DE FLAVORS PARA NUVEM PRIVADA BASEADA EM KUBERNETES**

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Inteligência Artificial e Aprendizado de Máquina, como requisito parcial à obtenção do título de *Especialista*.

Fortaleza

Novembro de 2022

**SUMÁRIO**

[1. Introdução 4](#__RefHeading___Toc80352482)

[2. Descrição do Problema e da Solução Proposta 4](#__RefHeading___Toc80352483)

[3. Canvas Analítico 4](#__RefHeading___Toc80352484)

[4. Coleta de Dados 4](#__RefHeading___Toc80352485)

[5. Processamento/Tratamento de Dados 5](#__RefHeading___Toc80352486)

[6. Análise e Exploração dos Dados 5](#__RefHeading___Toc80352487)

[7. Preparação dos Dados para os Modelos de Aprendizado de Máquina 6](#__RefHeading___Toc80352488)

[8. Aplicação de Modelos de Aprendizado de Máquina 6](#__RefHeading___Toc80352489)

[9. Discussão dos Resultados 6](#__RefHeading___Toc80352490)

[10. Conclusão 6](#__RefHeading___Toc80352491)

[11. Links 6](#__RefHeading___Toc80352492)

[12. Referências 7](#__RefHeading___Toc80352493)

# 1. Introdução

A computação em nuvem é modelo de uso de recursos computacionais em plena expansão. Esse modelo vem sendo gradualmente adotado por empresas de diferentes portes atraídas por muitos benefícios que em última instância convergem para economia. Dentre os vários benefícios na sua adoção destaca-se: a troca de despesas fixas no uso de recursos e sustentação de Datacenters por uma despesa variável, no qual o pagamento está associado ao uso dos recursos consumidos e que algumas vezes embutem recursos gerenciados e *serverless*, que dispensam procedimentos operacionais; diminuição do risco do investimento de uma infraestrutura inicial necessária para a sustentação de um novo negócio, pois o dimensionamento de recursos computacionais pode não ser trivial, principalmente em um novo negócio quando há incertezas quanto a sua adoção e crescimento; agilidade de implantação e uso pelo provimento de plataformas como serviços (PaaS) e softwares como serviços (SaaS) alicerçados pelos mais variados serviços de gerenciamento de identidades e permissões, segurança, monitoração e outros; maior facilidade na expansão dos negócios, mesmo numa escala global, pois muitas nuvens públicas têm regiões e zonas de disponibilidade em vários continentes do mundo permitindo pontos de presença mais próximos de clientes em diversos lugares; além de outros.

Tudo isso remete para uma mudança de cultura no qual o cliente pode focar mais no seu negócio deixando boa parte das responsabilidades de infraestrutura delegadas ao provedor de nuvem, tendo como referência um modelo de responsabilidade compartilhada.

A computação em nuvem não está restrita às nuvens públicas, representadas por grandes provedores como AWS (Amazon Web Services), GCP (Google Cloud Plataform), IBM, Oracle, Azure e outros. Muitas empresas possuem nuvens privadas e buscam um aproveitamento mais efetivo de investimentos nos seus Centros de Dados. Nesse propósito, a empresa que trabalho, cujo o nome foi omitido por questão de sigilo, desenvolveu em 2016 um serviço de nuvem privada, num conceito primário de Infraestrutura e Plataforma como Serviços (IaaS/PaaS) com objetivo de fornecer a equipes de desenvolvimento a liberdade de desenhar e implementar a arquitetura de produção de seus serviços conforme achar adequado. Essa solução é baseada em Kubernetes (referência), um sistema open-source que permite a distribuição, escalabilidade e gerenciamento de aplicações contêinerizadas. O nome do serviço será referenciado aqui com o nome fictício de **Nuvem Privada (NP)**.

O trabalho proposto tem o intuito de aperfeiçoar o NP com a introdução uma seleção inteligente de *flavors*, utilizando aprendizado de máquina, mais especificamente, modelos de classificação, a fim de predizer qual a configuração de *flavors* mais adequada para a execução de uma aplicação.

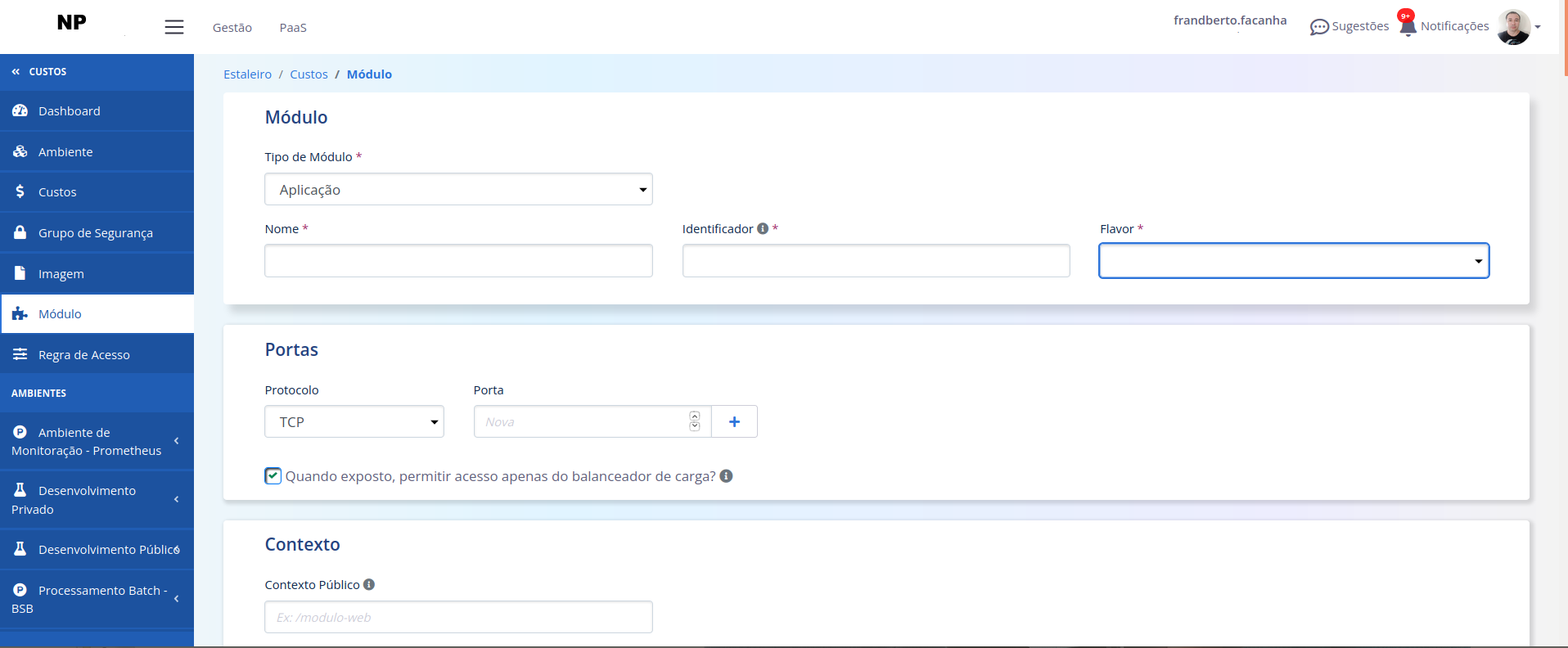
Os *Flavors* são configurações pré-definidas de recursos de CPU e memória associadas a aplicação antes de serem publicadas no **NP.**

Um modelo de classificação se destina a encontrar uma classe, ou categoria, dentre um número limitado de classes existentes, no qual um exemplar é categorizado a partir dos dados de entrada. Ela faz parte do grupo de modelo supervisionado, na qual a amostra utilizada para o treinamento do modelo tem a saída rotulada para o aprendizado do modelo. Modelos de classificação são bastante utilizados em sistemas de predição.

## 2. Descrição do Problema e da Solução Proposta

O **NP** atualmente executa um total de aproximadamente **600** sistemas, publicados em **2100** ambientes que rodam cerca de **11000** aplicações. Ele emprega um modelo de uso para a configuração de aplicações que é baseado na definição de módulos. Um módulo representa uma aplicação que roda em contêiner num ambiente de um sistema, mapeado para um pod no Kubernetes. A execução acontece quando um módulo é publicado num ambiente definido, sendo cada réplica instanciada num contêiner e executada em um pod.

No NP existem 3 tipos de módulos: aplicação, cronjob e addon. Um módulo aplicação é o mais utilizado e corresponde as aplicações que permanecem em execução *daemon* aguardando requisições em uma porta TCP, podendo ser implementadas em diversas linguagens. Um Cronjob é uma aplicação que roda acionada por um escalonador com base num agendamento pré-definido para a realização de uma tarefa específica que depois de ser concluída, a aplicação é finalizada. Por fim, o tipo Addon criado a partir de imagens pré-configuradas de serviços, como por exemplo, um serviço de cache Redis, um serviço de Elastic Search, dentre outros.

Tela de criação de Módulo da Aplicação no NP

(A imagem original teve o logotipo trocado para NP)

Todos os tipos de módulos trazem em comum, sob o ponto de vista de configuração, uma identificação e um flavo*r*. Como mencionado anteriormente, o flavor corresponde a configuração de recursos de CPU e Memória pré-definidos. Atualmente existem para seleção os flavors: **m0** – 4mCPU / 64MB; **m1** – 8mCPU / 128MB; **m2** – 16mCPU / 256MB; **m3** – 24mCPU / 384MB; **m4** – 32mCPU / 512MB; **m5** – 48mCPU / 768MB; **m6** – 64mCPU / 1024MB; **m7** – 80mCPU / 1280MB; **m8** – 96mCPU / 1536MB; **m9** – 96mCPU / 2048MB; **n2** – 256mCPU / 4096 MB; além de outros flavors especializados com GPU.

As unidades utilizadas para definição dos flavors são: mCPU (miliCPU) – divide o tempo de execução de 1 core de CPU por 1000; para memória a unidade é MB (Mega Bytes).

Sob o ponto de vista do Kubernetes, esses dois parâmetros vão indicar, respectivamente, a quantidade de tempo de CPU solicitada (request) e o limite de memória utilizado pelo contêiner.

O que ocorre na prática, é que a escolha do tamanho do flavor muitas vezes acontece de forma imprecisa, pois o desenvolvedor que implementa a aplicação em sua estação de trabalho não tem como mensurar de forma fácil os recursos necessários de cpu e memória para a execução ideal da aplicação. É difícil fazer um mapeamento entre os recursos locais e o que roda num contêiner, pois o flavoremprega unidades de processamento, como mCPUs, que não têm um relacionamento direto com os recursos na máquina do desenvolvedor que utiliza cores compartilhados com outros diversos processos que executam no sistema operacional.

A escolha de um flavornão adequado impacta o processamento da aplicação, pois um flavorsubdimensionado pode levar a aplicação processar sempre próximo do limite dos recursos podendo, em momentos de pico, ter o contêiner da aplicação finalizado pelo Kubernetes devido ao consumo além do limite permitido, no caso de memória. Num outro extremo, um flavorsuperdimensionado faz com que a aplicação execute sem falhas, devido aos recursos em abundância, no entanto haverá desperdício, o que limita a criação de outros pods no mesmo nó mesmo tendo recursos disponíveis para tal.

Outra consequência factível, é que um flavorescolhido idealmente hoje pode não ser amanhã, pois as aplicações passam por evoluções que podem refletir no aumento ou diminuição do consumo de recursos.

No intuito de endereçar essas questões é proposto nesse trabalho um mecanismo de predição de flavors para aplicações baseadas na classificação das aplicações a partir da análise dos recursos computacionais consumidos. Ou seja, uma vez que uma aplicação está em execução com um flavorassociado, ela passa a ser continuamente monitorada e classificada, quando for predito um flavormais adequado será gerada uma notificação aos gestores do serviço sugerindo a mudança do flavor.

Modelos preditivos de classificação abordam problemas nos quais a partir de uma entrada é possível classificar e predizer o dado associando-o a uma classe rotulada. A tarefa de classificação requer um treinamento a partir de um conjunto de dados, com exemplos de entradas e saídas (rótulos) para o aprendizado. O desafio está na preparação de um conjunto de dados (dataset) de treinamento que seja suficientemente representativo para o problema e tenha muitos exemplos de cada classe de rótulo. Esses fatores influenciam nos resultados obtidos.

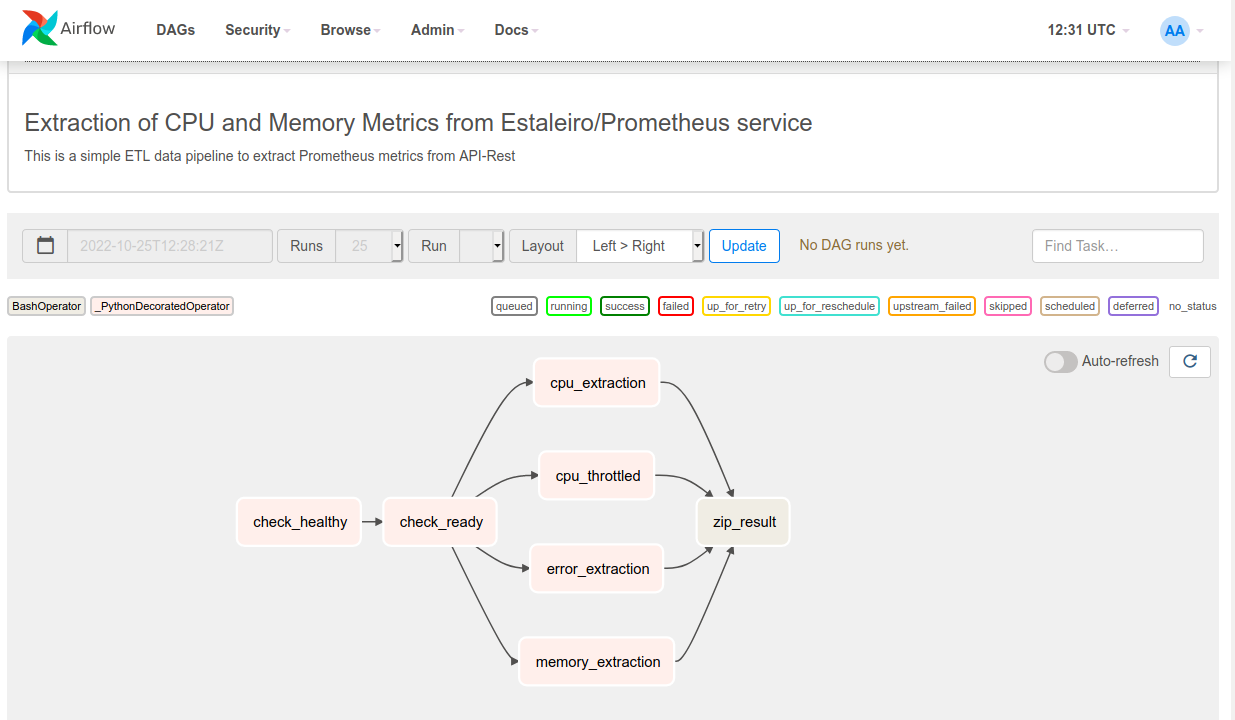
Existem diferentes tipos de algoritmos de classificação para modelar problemas de modelagem preditiva de classificação. Dentre entre eles estão

Importante observar que não há uma indicação do melhor algoritmo sem antes realizar experimentos controlados para descobrir qual algoritmo e configuração resulta num melhor desempenho.

# 4. Coleta de Dados

Os dados utilizados são oriundos de medições realizadas no cluster Kubernetes, por meio de sua api de métricas que é acionada pelo serviço de Monitoração do Prometheus. O Prometheus é um sistema de coleta de métricas de aplicações e serviços que armazena os dados num banco de dados próprio de séries temporais. Outras aplicações podem acessar os dados do Prometheus para obtenção de métricas coletadas por meio de sua api (http://<ip prometheus>:9090/api/v1/query) por meio de consultas onde são indicas métricas e filltros (Referência consulta Prometheus) .

Para esse projeto foi criado um DAG (Directed Acyclic Graph) no Airflow (referência) para coletar métricas a partir de requisições à API do Prometheus. O DAG criado define de 7 tarefas: 2 tarefas de verificação, a **check\_health** que testa a conectividade com o Prometheus e a **check\_ready**, que verifica se o Prometheus está pronto para receber requisições; 4 tarefas paralelas de coleta de métricas, sendo elas, a **cpu\_extraction** que coleta o consumo de CPU por contêiner; a **memory\_extraction** que coleta o consumo de memória do contêiner em bytes; a **error\_extraction** que coleta a quantidade de erros relacionados ao estouro de limite de memória (reason = OOMKilled) ; e a **cpu\_throttled\_extraction** que coleta o consumo de cpu adicional requerido pelo contêiner; e a última tarefa **zip\_result** que realiza a compactação dos arquivos gerados.



DAG no Airflow para coleta das Métricas

Os dados coletados fornecem uma foto do consumo de recursos das aplicações (contêineres) numa janela dos últimos 5 minutos. As coletas foram agendadas para serem realizadas a cada 15 minutos por um período de 24 horas. O Prometheus é configurado para coletar as métricas do cluster Kubernetes a cada 1 minuto.

No final do período de 24 horas foram gerados 384 arquivos, em formato CSV, no qual cada um deles está identificado pelo tipo da coleta e um timestamp. Os arquivos gerados estão publicados na pasta do projeto (referência).

Os arquivos coletados têm as seguintes informações:

|  |  |  |
| --- | --- | --- |
| **Nome do dataset:** Consumo de CPU pelo contêiner  **Arquivo: cpu\_hhmmssDDMMAAA.csv**  **Descrição:** Medição da taxa de uso de CPU pelo contêiner nos últimos 5 minutos.  **Consulta Prometheus da Coleta:**  sum(rate(container\_cpu\_usage\_seconds\_total{[5m])) by (namespace, pod, container) | | |
| **Nome do Atributo** | **Descrição** | **Tipo** |
| sistema | Nome do sistema | Categórico (string) |
| ambiente | Representa o ambiente do sistema no qual o contêiner executa | Categórico (string) |
| modulo | Nome do módulo (aplicação) | Categórico (string) |
| contêiner | Identificação do Contêiner (pod) que executa o módulo | Categórico (string) |
| consumo de CPU | Taxa de consumo de CPU pelo contêiner | Numérico (float) |

|  |  |  |
| --- | --- | --- |
| **Nome do dataset:** Consumo de Memória pelo contêiner  **Arquivo: memory\_hhmmssDDMMAAA.csv**  **Descrição:** Medição do consumo de memória em Bytes pelo contêiner.  **Consulta Prometheus da Coleta:**  sum(container\_memory\_usage\_bytes{}) by (namespace,pod,container) | | |
| **Nome do Atributo** | **Descrição** | **Tipo** |
| sistema | Nome do sistema | Categórico (string) |
| ambiente | Representa o ambiente do sistema no qual o contêiner executa | Categórico (string) |
| modulo | Nome do módulo (aplicação) | Categórico (string) |
| contêiner | Identificação do Contêiner (pod) que executa o módulo | Categórico (string) |
| consumo de Memória | Consumo de Memória pelo contêiner em Bytes | Numérico (bigint) |

|  |  |  |
| --- | --- | --- |
| **Nome do dataset:** Quantidade de Erros por Estouro de Limite de Memória (OOMKilled)  **Arquivo: error\_hhmmssDDMMAAA.csv**  **Descrição:** Quantidade de Erros por Estouro de Limite de Memória (OOMKilled) de contêineres. Quando ocorrem esses erros o contêiner é finalizado e outro é criado no lugar.  **Consulta Prometheus da Coleta:**  sum(kube\_pod\_container\_status\_last\_terminated\_reason{reason="OOMKilled"}) by (container, namespace, pod) | | |
| **Nome do Atributo** | **Descrição** | **Tipo** |
| sistema | Nome do sistema | Categórico (string) |
| ambiente | Representa o ambiente do sistema no qual o contêiner executa | Categórico (string) |
| modulo | Nome do módulo (aplicação) | Categórico (string) |
| contêiner | Identificação do Contêiner (pod) que executa o módulo | Categórico (string) |
| tipo de erro | Tipo de erro filtrado ( OOMKilled) | Categórico (string) |
| quantidade de erros | Quantidade de erros | Numérico (int) |

|  |  |  |
| --- | --- | --- |
| **Nome do dataset:** Consumo extra de CPU  **Arquivo: cpu\_throttled\_hhmmssDDMMAAA.csv**  **Descrição:** Medição da taxa de uso de CPU extra (além do limite configurado) pelo contêiner nos últimos 5 minutos.  **Consulta Prometheus da Coleta:**  sum(rate(container\_cpu\_cfs\_throttled\_seconds\_total{}[5m])) by (namespace, pod, container) | | |
| **Nome do Atributo** | **Descrição** | **Tipo** |
| sistema | Nome do sistema | Categórico (string) |
| ambiente | Representa o ambiente do sistema no qual o contêiner executa | Categórico (string) |
| modulo | Nome do módulo (aplicação) | Categórico (string) |
| contêiner | Identificação do Contêiner (pod) que executa o módulo | Categórico (string) |
| consumo de CPU | Taxa de consumo extra de CPU pelo contêiner | Numérico (bigint) |

# 5. Processamento/Tratamento de Dados

O processamento e tratamento dos dados compreendem ao conjunto de ações a fim de trabalhar o dado deixando-o mais adequado para a utilização do modelo de classificação.

As ações realizadas foram desenvolvidas em Jupyter Notebooks e executadas no ambiente Anaconda e encontram-se publicadas na pasta notebooks do projeto. As ações implementaram as seguintes etapas:

1) Pré-processamento dos arquivos de métricas com adição de novas colunas

Esse pré-processamento, implementado no notebook **01-pre\_processar\_arquivos.ipynb,** adicionou a cada arquivo as informações de tempo e identificador único da aplicação (hash). Foram adicionados os novos campos, **hora** (tipo inteiro), **minuto** (tipo inteiro) e **hash** (Categórico - String), aos campos existentes. O campo hash funciona como o identificador da aplicação pois é gerado da composição do ambiente com o contêiner (pod).

2) Consolidação dos arquivos de métricas num único arquivo

Esse processamento, implementado no notebook **02-agregar\_coletas\_metrica.ipynb**, agregou todos os arquivos da mesma métrica num único arquivo consolidado. Nesse passo, os campos de identificação da aplicação como sistema, ambiente e módulo foram suprimidos, não sendo mais necessários pois a identificação da aplicação estava garantida pelo campo hash.

O resultado das consolidações de cada métrica segue uma estrutura similar ao apresentado a seguir, num trecho do arquivo consolidado do consumo de CPU.

hash,hora,min,consumo\_cpu

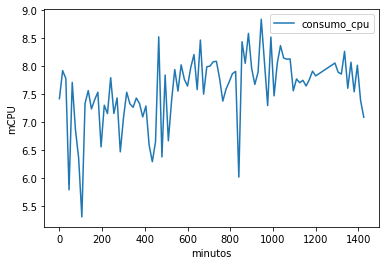
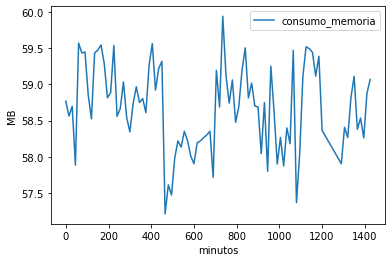
ce4468ee8389709c20b40ed1c09a36c1,12,0,0.0007187651231046

0dcf61756612a9edfc06d254802e4b28,12,0,0.0010247318639428

...

Com os dados consolidados por métricas torna-se mais fácil trabalhar os dados para explorar analiticamente e realizar as análises.

Tomando, por exemplo, uma única aplicação filtrando pelo hash, implementada no notebook **03-exploracao\_dos\_dados.ipynb**, é possível visualizar facilmente o consumo de CPU e Memória no período de 24 horas. Os resultados dos consumos estão apresentado a seguir.



# Obviamente, o consumo de recurso da aplicação varia de acordo com natureza da aplicação.

# 6. Análise e Exploração dos Dados

Dos dados coletados, os mais significativos são valores de consumo de CPU e Memória. É importante perceber que essas variáveis são independentes.

O dado da quantidade de erro do tipo OOMKilled indica que o limite de memória configurado pelo flavor escolhido não é suficiente ocorrendo um estouro do limite de memória. A consequência desse erro é a destruição do contêiner e sua recriação com perda do estado da execução anterior.

O outro dado relevante é o consumo de cpu “throttled”, que indica um consumo de CPU além do limite configurado no flavor. Os dois últimos casos são situações de exceção que, a fim de simplificação, não serão abordadas nesse projeto, ou seja, as aplicações que estão nesses cenários serão removidas na amostra.

# 7. Preparação dos Dados para os Modelos de Aprendizado de Máquina

A amostra de dados coletada assumiu como premissa que o flavor atribuído para a execução de cada contêiner não está adequado e não pode ser utilizado como um rótulo para o treinamento do modelo. Diante disso, um novo rótulo deve ser identificado e associado a cada contêiner para garantir uma configuração de recursos mais próxima de sua necessidade dada as suas características de execução.

Nesse sentido, foram realizadas ações de preparação dos dados para poder ser aplicado a um modelo supervisionado. As ações estão detalhadas no notebook **04-preparar\_dados.ipynb**, disponível no site do projeto e descrito nessa seção.

1) Agregação dos Consumos de CPU

Os consumos de CPU de cada contêiner foram agrupados buscando-se medidas mais significativas de representação. Foram escolhidas duas medidas para representar o conjunto dos valores de consumo de cada contêiner, sendo elas: o valor de pico do consumo e a mediana do consumo. O valor de pico dá uma indicação do limite esperado de consumo. Enquanto a mediana, é uma medida de tendência central que divide o conjunto de dados exatamente na metade, de forma que metade dos elementos são menores que esse valor, e a outra metade é maior que esse valor. Ao utilizar a mediana evitamos que os valores extremos de uma distribuição assimétrica influenciem no resultado, como ocorre com a média.

Antes de aplicar a agregação, foram eliminados os valores zerados e feita uma conversão do valor de consumo para mCPU, multiplicando o valor original por 1000.

*# Remoção dos valores zerados de consumo de CPU*

df\_consolidado\_cpu = df\_consolidado\_cpu[df\_consolidado\_cpu['consumo\_cpu'] > 0]

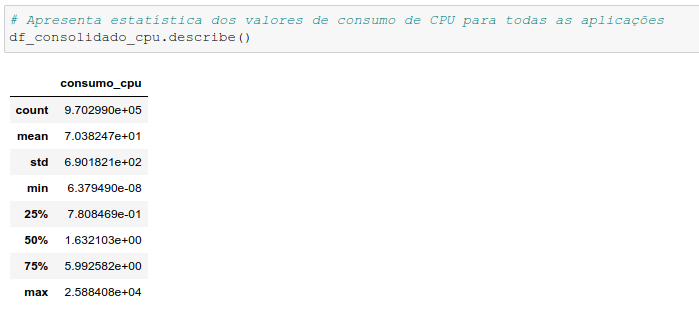
*# Transformação do consumo de CPU em miliCPU (miliCPU) (x 1000)*

df\_consolidado\_cpu['consumo\_cpu'] = df\_consolidado\_cpu['consumo\_cpu'] \* 1000

# Em seguida foram excluídas as linhas de hora e minuto resultando apenas as colunas de hash e consumo\_cpu, pois não eram mais necessárias para a agregação.

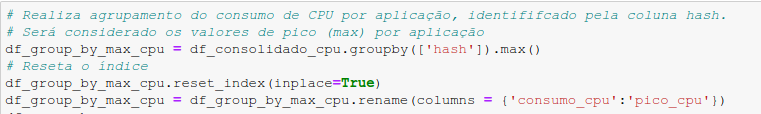


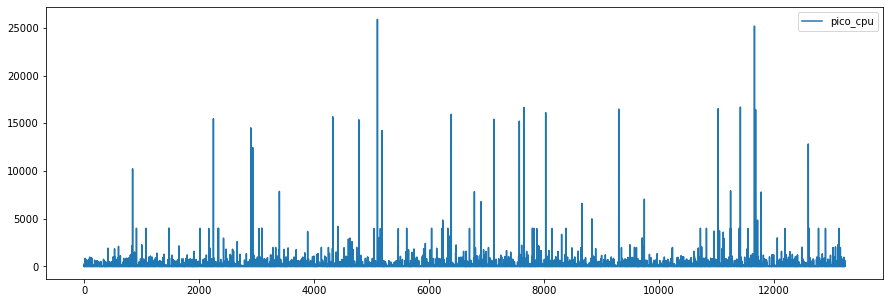
Foi realizado um levantamento estatístico dos dados resultando nas informações a seguir:



Os resultados indicam que 75% dos dados apresenta um consumo de CPU menor que 6.0 mCPUs. No entanto 25% dos dados apresentam valores bem superiores, chegando ao máximo de 25884 mCPU. O que leva ao um desvio padrão bem maior que a média. A conclusão é que grande maioria dos contêineres apresenta baixo consumo de cpu e memória.

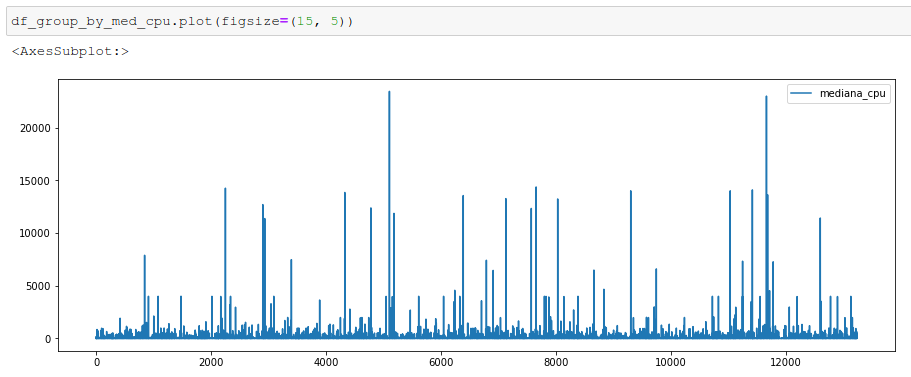
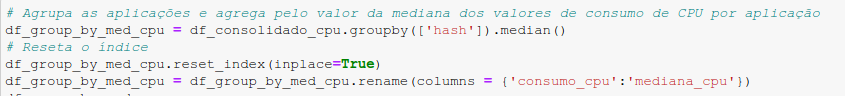
Esse comportamento foi corroborado com os dados obtidos após a agregação dos contêineres com valores de pico, o que pode ser observado a seguir.

 O gráfico apresentado a seguir mostra a distribuição dos valores de picos



Percebe-se a grande concentração de contêineres na parte mais do eixo y (pico\_cpu).

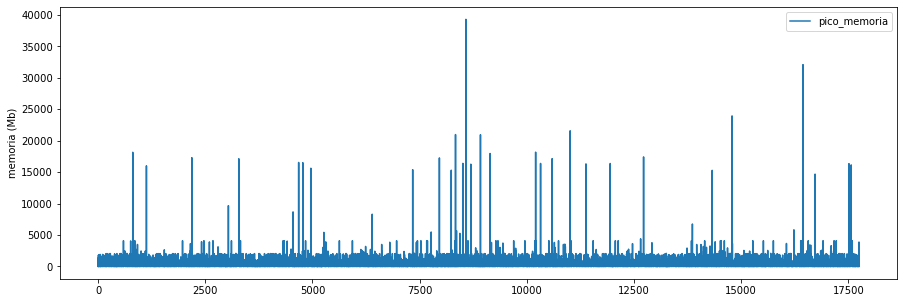
Fazendo processamento análogo para obter o valor da mediana do consumo de cpu, o gráfico apresenta formado similar mas com valores menores para aplicações, com exceção do jobs que mantêm os mesmos valores.

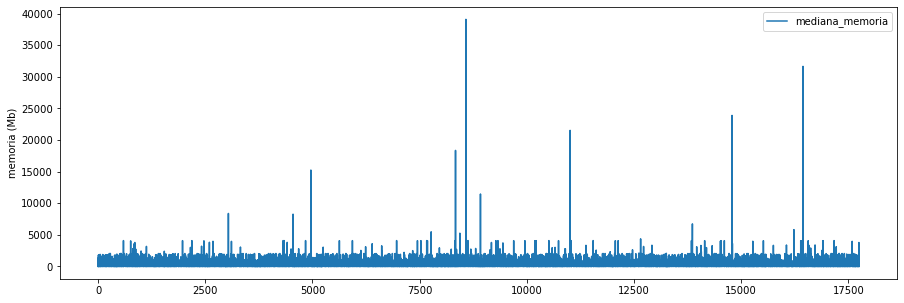


2) Agregação dos Consumos de Memória

O mesmo procedimento foi realizado para obter a agregação do consumo de memória pelo valor de pico e mediana.

Os valores obtidos estão representados nos gráficos a seguir.

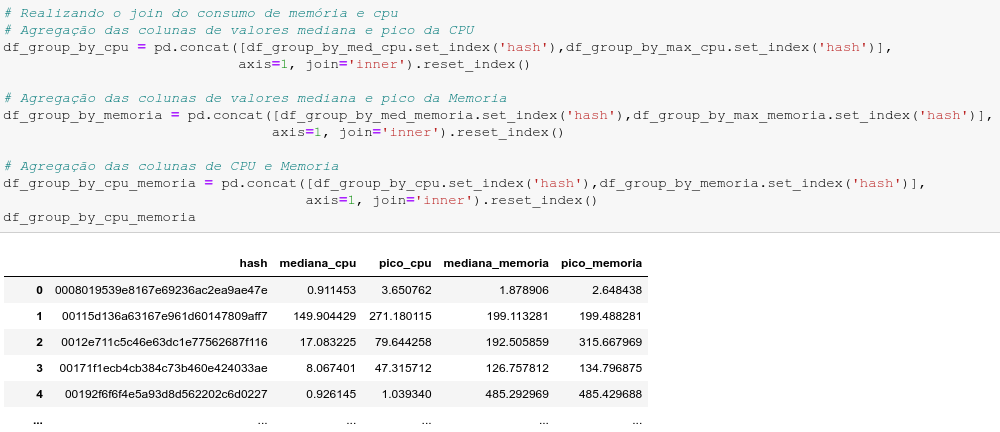




Notadamente, percebe-se uma concentração dos consumos de memória na parte inferior do gráfico para ambas as situações.

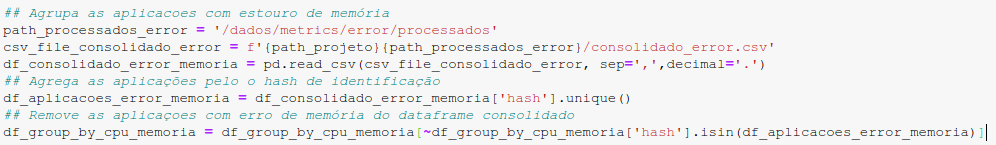
3) Agregação dos valores médios e picos de CPU e Memória

Foi feita a agregação dos dados considerando as features de valores da mediana e pico de CPU e Memória



4) Remoção de Valores Discrepantes

Nessa etapa foram removidos os contêineres que apresentam comportamento de consumo fora dos limites, identificados pelas métricas de erro causado por estouro de memória (OOMKilled) e consumo excessivo de cpu (throttled).

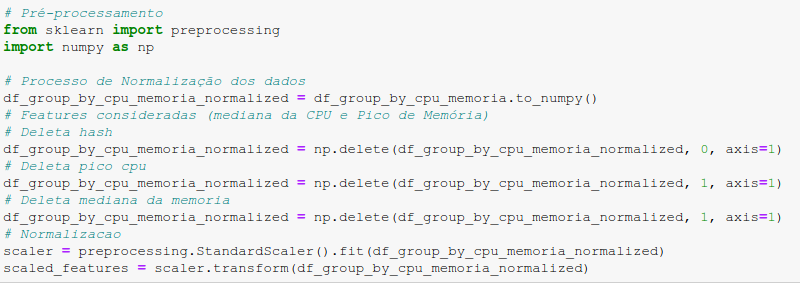
O resultado foi a eliminação de cerca de 13% dos contêineres, ficando cerca de 11400 contêineres na amostra.

5) Avaliação dos dados com K-Means

Foi aplicado o modelo K-Means (referência) aos dados com o intuito de buscar um entendimento melhor sobre como os dados de consumo dos contêineres estão distribuídos e relacionados entre si.

O K-Means é um modelo não supervisionado, ou seja, não se baseia em rótulos preestabelecidos, que é uma característica dessa amostra. Ao invés disso, o modelo busca identificar classes entre os exemplares, chamados clusters, observando a similaridade com relação a distâncias entre as features analisadas.

Uma etapa importante para a aplicação do modelo é realizar “feature scaling”, que objetiva fazer com que as features tenham a mesma escala, para que uma feature não seja influenciada pela grandeza de outra maior.

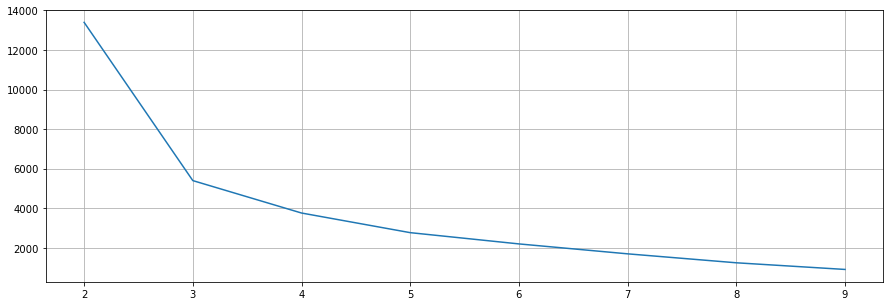


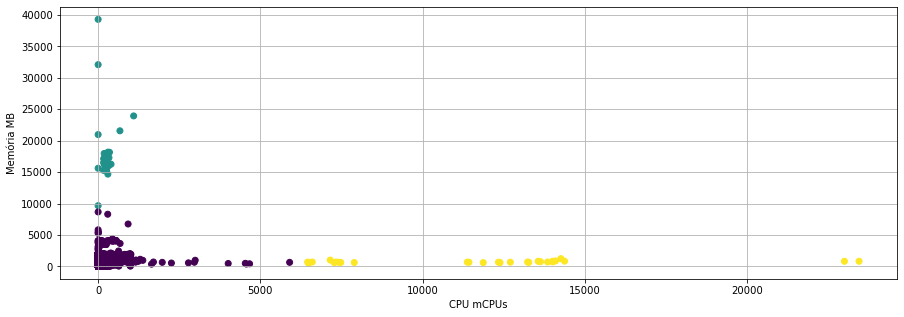
Um hiperparâmetro obrigatório para a aplicação do K-Means é determinação da quantidade de clusters (classes). Esse parâmetro é desconhecido, para tentar identificá-lo foi utilizada o método Elbow. No qual são avaliados resultados do K-Means com diversos números de classes plotado num gráfico, os pontos que unem as curvas com maior ângulo, como um cotovelo, são sugestivos para classes.

O resultado obtido está representado a seguir.



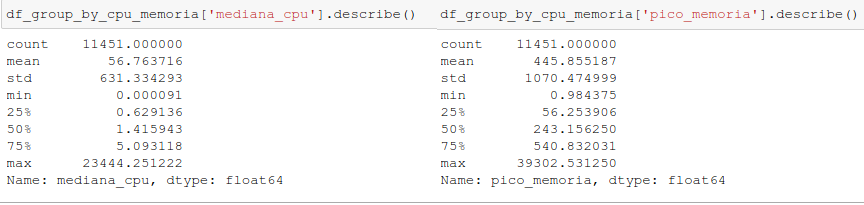
Plotando o gráfico de distorções versus n\_cluster, é obtido.

O número de cluster sugerido é de 3. A plotagem do gráfico considerando o número de classes igual a 3 para as features mediana\_cpu e pico\_memoria, é obtido:



Percebe-se uma concentração clara de amostras da classe azul com valores de consumo de memória e CPU bem mais baixo que as outras classes.

Observando novamente as estatísticas dessas métricas:

Os 75% dos valores da mediana de CPU se concentram abaixo de **5,10mCPU** e 75% do pico de memória está abaixo de **541MB**.

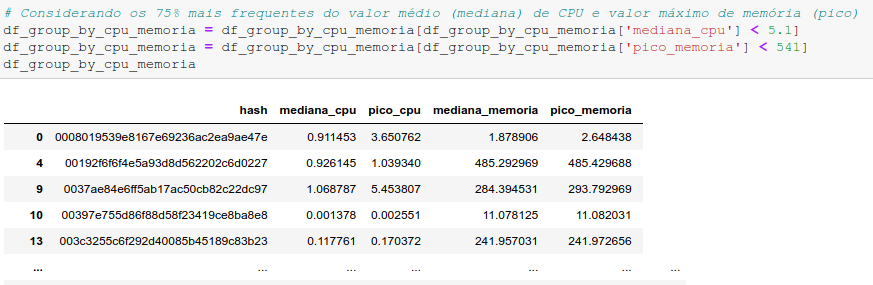
Comparando esses valores com os flavors atualmente definidos: flavors: **m0** – 4mCPU / 64MB; **m1** – 8mCPU / 128MB; **m2** – 16mCPU / 256MB; **m3** – 24mCPU / 384MB; **m4** – 32mCPU / 512MB; **m5** – 48mCPU / 768MB; **m6** – 64mCPU / 1024MB; **m7** – 80mCPU / 1280MB; **m8** – 96mCPU / 1536MB; **m9** – 96mCPU / 2048MB.

Os valores de consumo de recursos se localizariam entre o flavor m0 até o flavor m5 devido ao limite necessário de memória, embora o consumo de CPU seja bem menor. Percebe-se então, que a definição dos flavors não parece estar adequada pois já conduz a um desperdício de recursos.

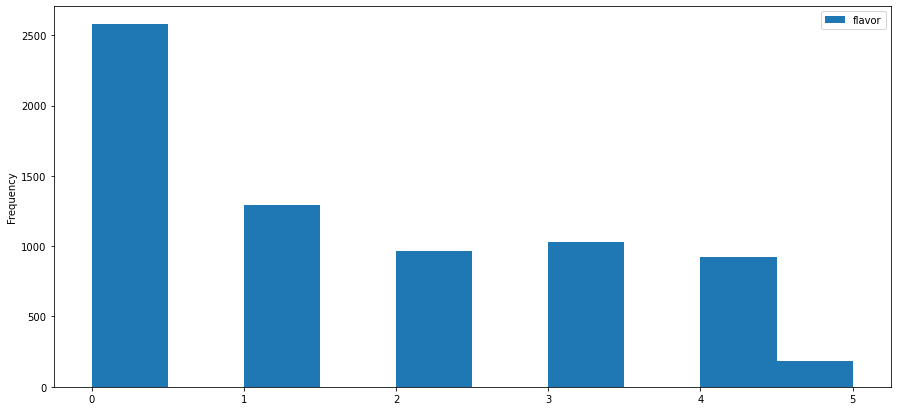
Propõem-se então uma nova definição de flavors com valores menores. Os novos valores chamados de f, variam de f0 a f5 e tem as seguintes definições: f0 - 1mCPU / 100MB, f1 - 2mCPU / 200MB, f2 - 3mCPU / 300MB, f3 - 4mCPU / 400MB, f4 - 5mCPU / 500MB e f5 - 6mCPU / 600MB

Uma graduação menor permitirá a escolha de recursos mais próxima da necessidade do contêiner. Com essa nova definição de flavors foi feita a rotulação da amostra de contêineres de acordo com o consumo de CPU (mediana) e memória (pico).

Também foi realizado um corte na amostra para contemplar apenas 75% de acordo com os valores de CPU e memória.



O gráfico a seguir mostra um histograma de distribuição dos contêineres pelos flavors rotulados.

A maior quantidade de contêineres utilizam recursos com baixo consumo de CPU iguais ou inferiores a 1mCPU / 100MB, e a menor quantidade apresenta um maior consumo de recursos entre 5 – 6 mCPU e 500 – 600 MB.

# 8. Aplicação de Modelos de Aprendizado de Máquina

Com o dataset pronto para ser utilizado, a atenção volta-se para o modelo. Para esse caso de uso um bom modelo candidato é a árvore de decisão.

# 9. Discussão dos Resultados

Nesta seção você deve relatar os resultados alcançados ao final do trabalho. Mostre os resultados das métricas adotadas, seja através de gráficos, tabelas, dentre outros, que permitam a validação do seu trabalho.

# 10. Conclusão

Nesta seção você deve apresentar um fechamento para o trabalho. É importante apresentar um breve resumo do trabalho, resgatando o problema, como foi tratado e os resultados obtidos, bem como as limitações e perspectivas (trabalhos futuros).

# 11. Links

Nesta seção você pode disponibilizar *links* para repositórios, como é o caso do GitHub, onde podem ser encontrados o seu projeto, códigos-fonte, vídeos demonstrativos, dentre outros.

# 12. Referências

Um Relatório Técnico para conclusão do curso de Pós-graduação *Lato Sensu* em Inteligência Artificial e Aprendizado de Máquina não requer revisão bibliográfica. Portanto, a inclusão das referências não é obrigatória. No entanto, caso você deseje incluir referências relacionadas às tecnologias ou às metodologias usadas em seu trabalho, relacione-as de acordo com as normas ABNT disponibilizadas no link <http://portal.pucminas.br/biblioteca/documentos/citacoes-referencias.pdf>.

<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/six-advantages-of-cloud-computing.html>

https://kubernetes.io/

<https://www.serpro.gov.br/tema/edicao-238/estaleiro-a-nuvem-do-serpro>

ANACONDA Inc. Anaconda: Data science technology for a better world; c2022. Página inicial. Disponível em: <https://www.anaconda.com/>. Acesso em: 04 de nov. de 2022.

APACHE SOFTWARE FOUNDATION, Airflow, c2022. Página inicial. Disponível em: <https://airflow.apache.org/>. Acesso em: 04 de nov. de 2022.

PROMETHEUS. Prometheus: From metrics to insight, c2022. Página inicial. Disponível em: <https://tecnoblog.net/>. Acesso em: 04 de nov. de 2022.

PROMETHEUS, Querying Prometheus, Disponível em: <https://prometheus.io/docs/prometheus/latest/querying/basics/#querying-prometheus>, Acesso em: 04 de nov. de 2022.

FAÇANHA, Frandberto, Projeto PUC, Git Hub, 2022, Disponível em: , Acesso em: 01 de nov. de 2022.