Sistemas Operacionais e Computação em Nuvem
CCOMP 5° Semestre

Entrega 2

Aleff Silva Souza
João Paulo Colombo
Luis Felipe Torelli Sparrapan
Matheus Morais Zimmer

São Paulo
2025

Requisitos entrega:
Implementação de um Algoritmo de IA

Formato de Entrega: Código-Fonte, Relatório de Monitoramento e Análise de Desempenho

Objetivo: Os estudantes devem criar um sistema de monitoramento de recursos de uma instância na nuvem (CPU, memória, etc.), implementando um modelo simples de IA, referente a 3 ou mais exercícios praticados na disciplina de Inteligência Artificial e Aprendizado de Máquina. O código, os gráficos de monitoramento e um relatório detalhado de como a IA foi aplicada

---

1 . Estrutura da pasta

```
ia-collector-cloud/
│
├── app.py          ← código principal Flask + IA + gráficos
├── Dockerfile      ← imagem da aplicação
```

Essa pasta contém todo o código-fonte e arquivos necessários para que o sistema seja construído e executado dentro de um container Docker.
Ela funciona como o diretório raiz do projeto, centralizando:

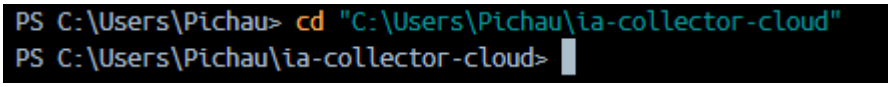
O código Python (app.py) é responsável pela coleta, aplicação dos modelos e geração de gráficos.

O Dockerfile, que descreve o ambiente e as dependências necessárias para o sistema funcionar de forma independente do computador local. (Explicação de ambos mais detalhada abaixo).

2 . Criar a pasta e entrar nela

```
mkdir "C:\Users\Pichau\ia-collector-cloud"
```

```
cd "C:\Users\Pichau\ia-collector-cloud"
```



Esta pasta pode ser criada na mão também, tomando cuidado somente onde ela será criada para pegar o caminho certo.

---

3 . Criar o arquivo app.py

Este arquivo foi criado com as regras de coleta de métricas, criação do arquivo principal da aplicação, coleta (thread) grava métricas em SQLite dentro do container, APIs (endpoints) Para coleta de informações e testes: /metrics, /ai/forecast, /ai/anomaly, /ai/clusters, /dashboard, /about, /healthz e os modelos de machine learning (no próprio container): Regressão Linear (previsão), IsolationForest (anomalia), KMeans (clusters).

O arquivo estará na pasta do Github e o código poderá ser consultado por lá.

---

4 . Criar o arquivo Dockerfile

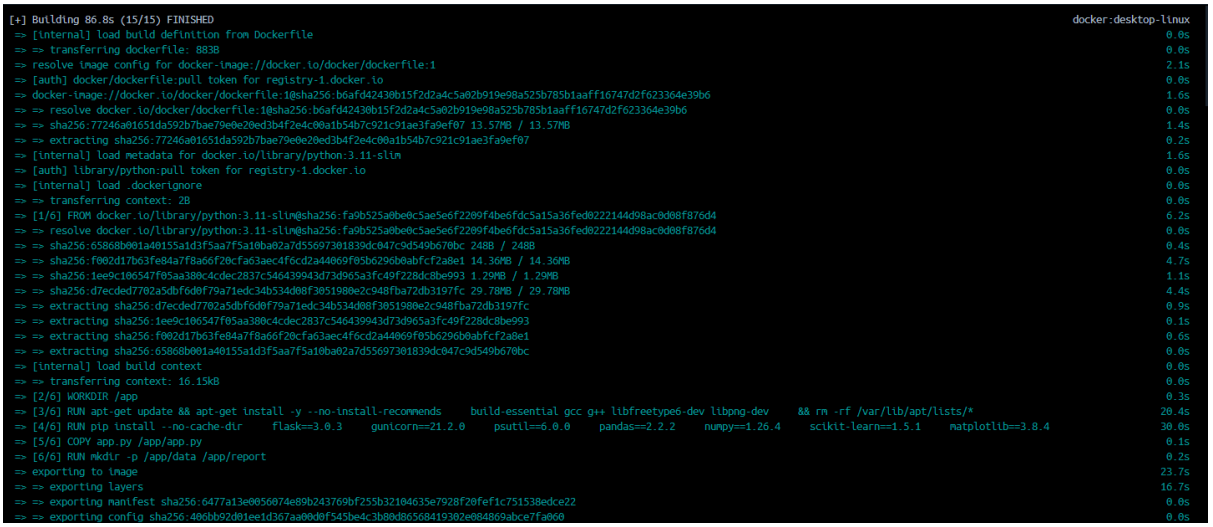
O que faz: define a imagem (Python), instala libs de IA, copia o código, cria pasta do banco e executa com gunicorn (produção).

O arquivo estará na pasta do Github e o código poderá ser consultado por lá.

---

5 . Construir a imagem Docker

- **docker build -t zimmer911/ia-collector:2.0 .**



Isso empacota todo o código na imagem `zimmer911/ia-collector:2.2`.

---

6 . Rodar o container

docker rm -f ia-collector 2>$null - Remove qualquer container que tenha este nome, usado para evitar erros.

docker run -d --name ia-collector -p 8080:8080 -e INTERVAL_SEC=5 zimmer911/ia-collector:2.2 - Roda o Container

```
PS C:\Users\Pichau\ia-collector-cloud> docker rm -f ia-collector 2>$null
>> docker run -d --name ia-collector -p 8080:8080 -e INTERVAL_SEC=5 zimmer911/ia-collector:2.2
>>
8383ad0b98e6bfa40c1edfc6cef172814145f382022c03269563d4e17de5eb55
PS C:\Users\Pichau\ia-collector-cloud>
```

Explicação do comando:

-d → modo background

--name → nome do container

-p 8080:8080 → expõe a porta 8080 local

-e INTERVAL_SEC=5 → coleta a cada 5 segundos

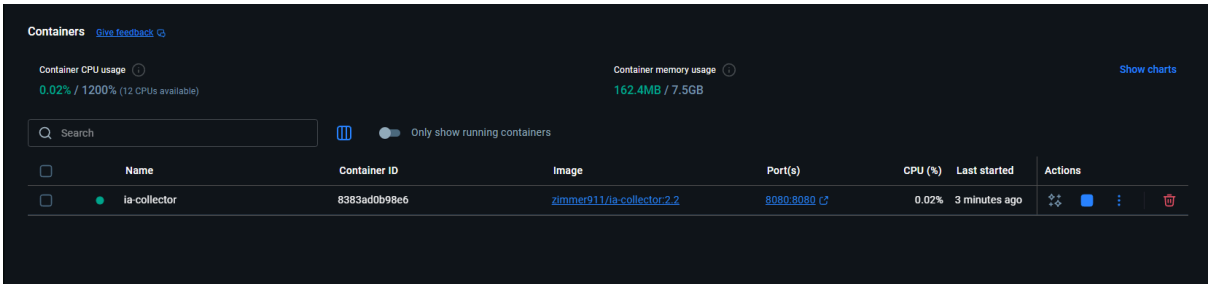
---

7 . Verificar se está rodando

docker ps

```
PS C:\Users\Pichau\ia-collector-cloud> docker ps
CONTAINER ID   IMAGE                        COMMAND                  CREATED         STATUS         PORTS                                       NAMES
8383ad0b98e6   zimmer911/ia-collector:2.2   "gunicorn --bind 0.0…"   2 minutes ago   Up 2 minutes   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp   ia-collector
```

Ou verificamos indo para a aba container e vemos ele sendo rodado.



---

8 . Testar saída

- **curl http://localhost:8080/healthz**
- **curl http://localhost:8080/about**
- **curl http://localhost:8080/metrics**

Caso de erro, utilizar curl.exe para teste.

---

9 . Geração dos gráficos

- **curl.exe -X POST http://localhost:8080/snapshot**

```
PS C:\Users\Pichau\ia-collector-cloud> curl.exe -X POST http://localhost:8080/snapshot
>>
{"generated_at":"2025-11-07T20:30:13.944673","ok":true}
```

Após rodar código para geração dos gráficos, abra no navegador:

http://localhost:8080/report/html

Você verá um relatório completo:

Relatório — IA Collector Cloud

Intervalo: 5.0s | **Janela:** 60.0 min | **Gerado:** 2025-11-07T20:30:32.946228Z

1. Introdução

Este relatório apresenta um sistema de monitoramento de recursos (CPU, memória e cargas) executado em container Docker, com aplicação de três modelos de IA: Regressão Linear (previsão), IsolationForest (detecção de anomalias) e KMeans (clusterização). Os dados são amostrados periodicamente e persistidos em `/app/data/metrics.db` (SQLite interno ao container).

2. Desenvolvimento

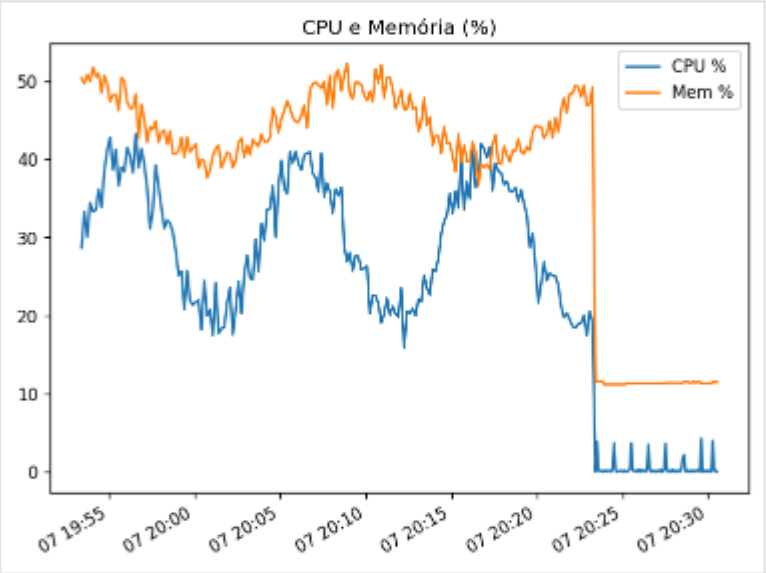
- **Coleta:** `psutil` captura CPU%, Mem% e load average (1m/5m/15m) em intervalos configuráveis (`INTERVAL_SEC`).
- **Persistência:** amostras gravadas em SQLite para consultas por janela (`WINDOW_MINUTES`).
- **Gráficos:** PNGs gerados via `matplotlib` (backend headless) e servidos pelo app.

3. Modelos de IA

- **Previsão (Regressão Linear):** usa defasagens (lags) de CPU para prever CPU futura e reporta MAE e R^2.
- **Anomalias (IsolationForest):** aprende o padrão multivariado e assinala outliers; exibe taxa de anomalia.
- **Clusters (KMeans):** agrupa perfis de carga (ex.: ocioso, carga, crítico) a partir de CPU%×Mem% e loads.

4. Resultados e Gráficos

4.1 CPU e Memória (%)



4.2 Loads (1m/5m/15m)

10 . Para ter acesso a nossa imagem e/ou container, acesse no navegador :

https://hub.docker.com/r/zimmer911/ia-collector/tags

E para acesso ao meu repositório no docker:

https://hub.docker.com/repositories/zimmer911

11 . Para testar na sua máquina:

Basta abrir o terminal do docker desktop e rodar somente isso:

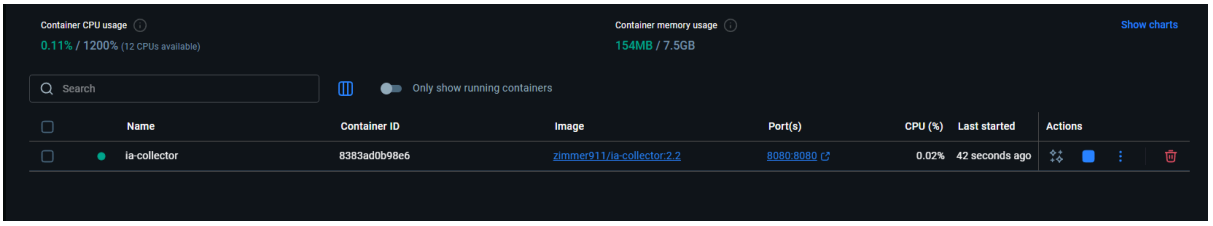
docker pull zimmer911/ia-collector:2.2

docker run -d --name ia-collector -p 8080:8080 zimmer911/ia-collector:2.2

Depois acessar no navegador:

http://localhost:8080/report/html

(Lembrando que o container tem que estar rodando para conseguir abrir)



E pronto, você verá:

- Coleta em tempo real (CPU, memória);
- Relatório HTML com os gráficos (PNG);
- Ia rodando dentro do container.

Para verificar se o container está ativo (opcional)

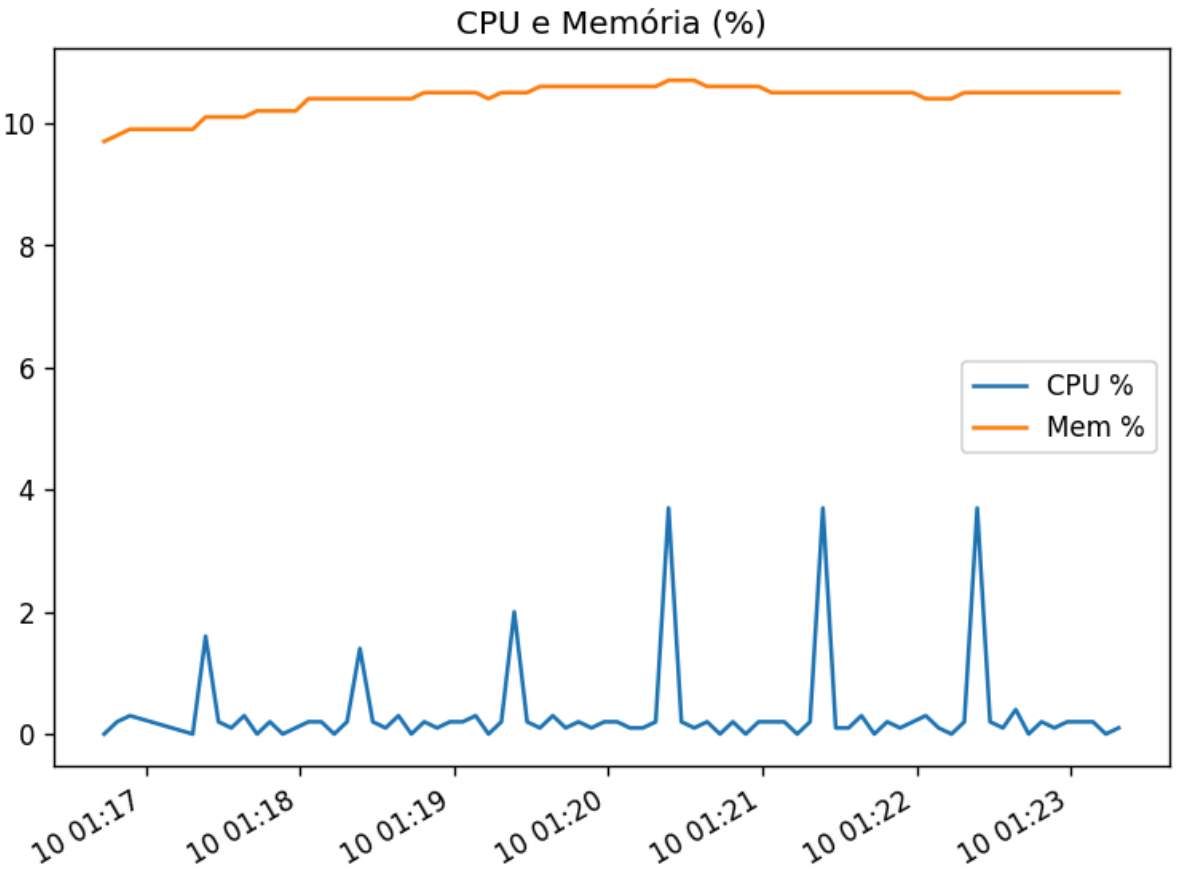
docker ps

Repositório da imagem no Docker Hub
https://hub.docker.com/r/zimmer911/ia-collector/tags

12 . Interpretação dos Gráficos

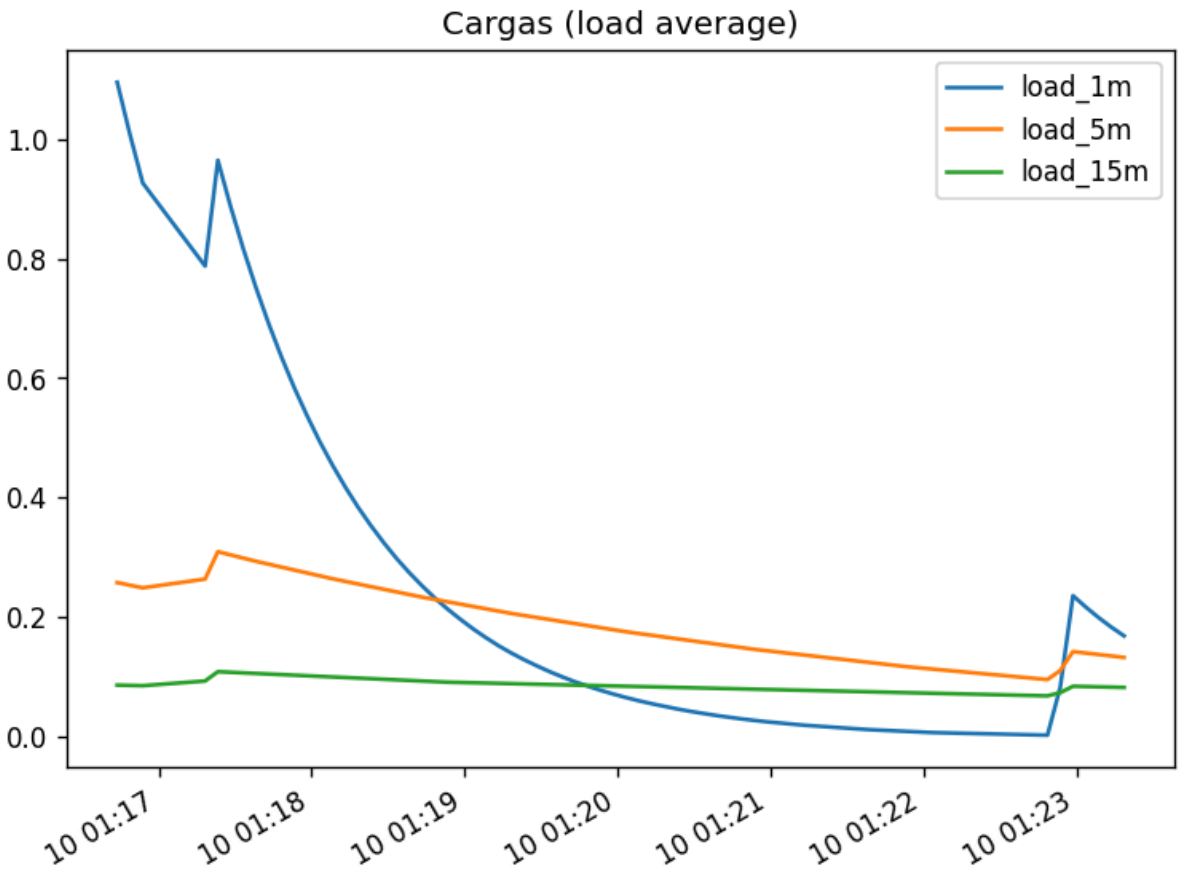
1. CPU e Memória (%)

- Eixo X: Tempo (ts)
- Eixo Y: Percentual (0–100%)
- O que mostra:
  Exibe a evolução do uso de CPU e Memória, indicando picos de processamento e consumo de RAM.



2. Loads (1m / 5m / 15m)

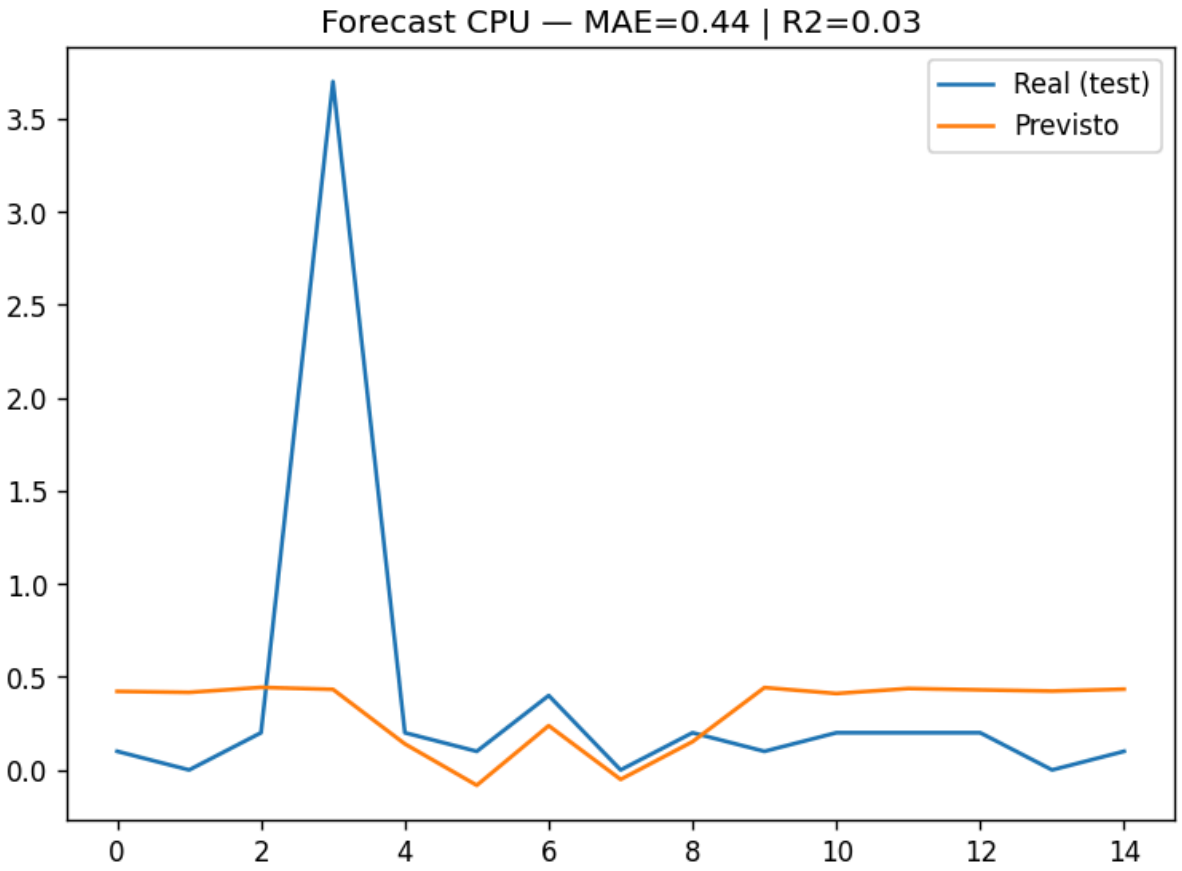
- Eixo X: Tempo (ts)
- Eixo Y: Load average
- O que mostra:
  Representa a carga média do sistema em janelas de 1, 5 e 15 minutos.
  No Windows, o valor pode ser 0 (sem suporte a métricas de load average).



---

3. Forecast (Regressão Linear)

- Eixo X: Índices de tempo (teste)
- Eixo Y: CPU (%)
- O que mostra:
  Exibe a curva real vs prevista da CPU, destacando: Erro médio (MAE), Coeficiente de ajuste (R^2)



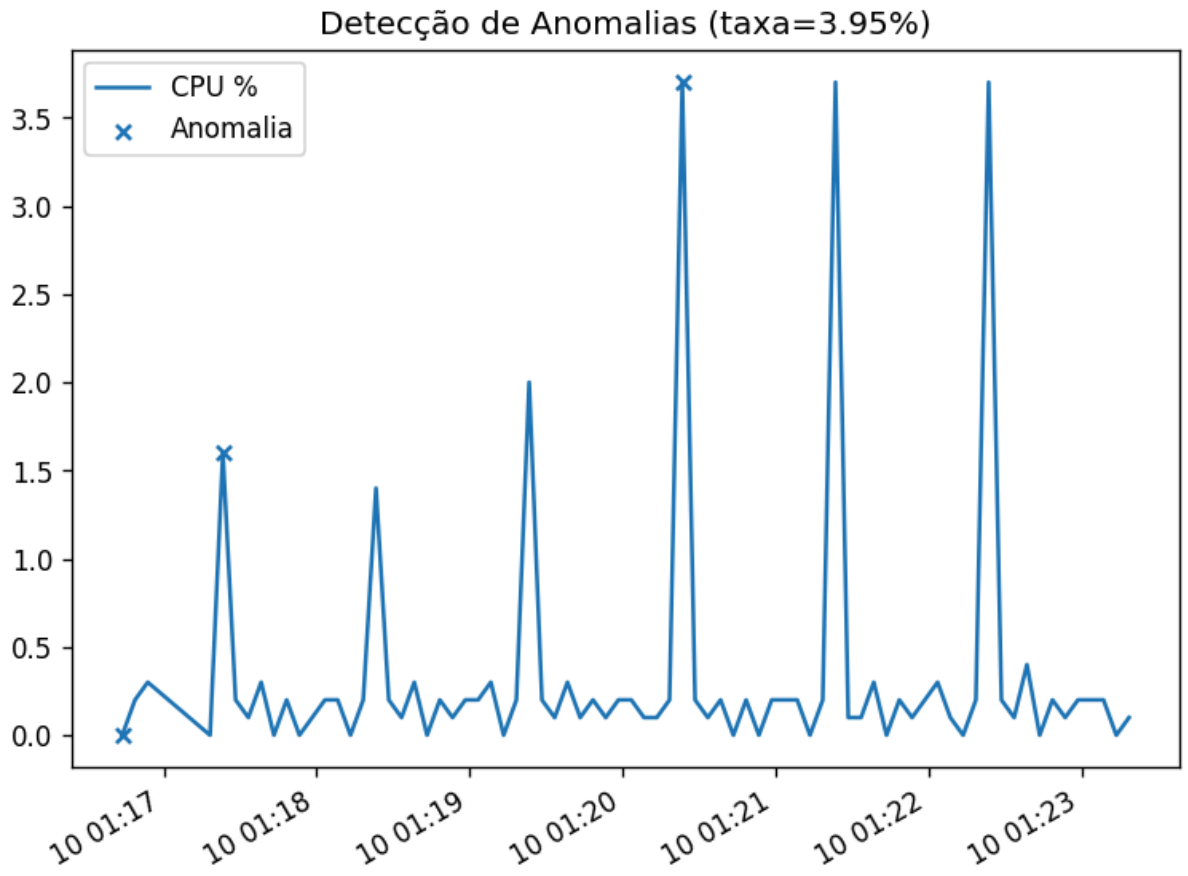
---

4. Anomalias (IsolationForest)

- Eixo X: Tempo (ts)

- Eixo Y: CPU (%)

- O que mostra:Identifica picos fora do padrão, marcados com “X”, representando momentos anômalos de uso da CPU.

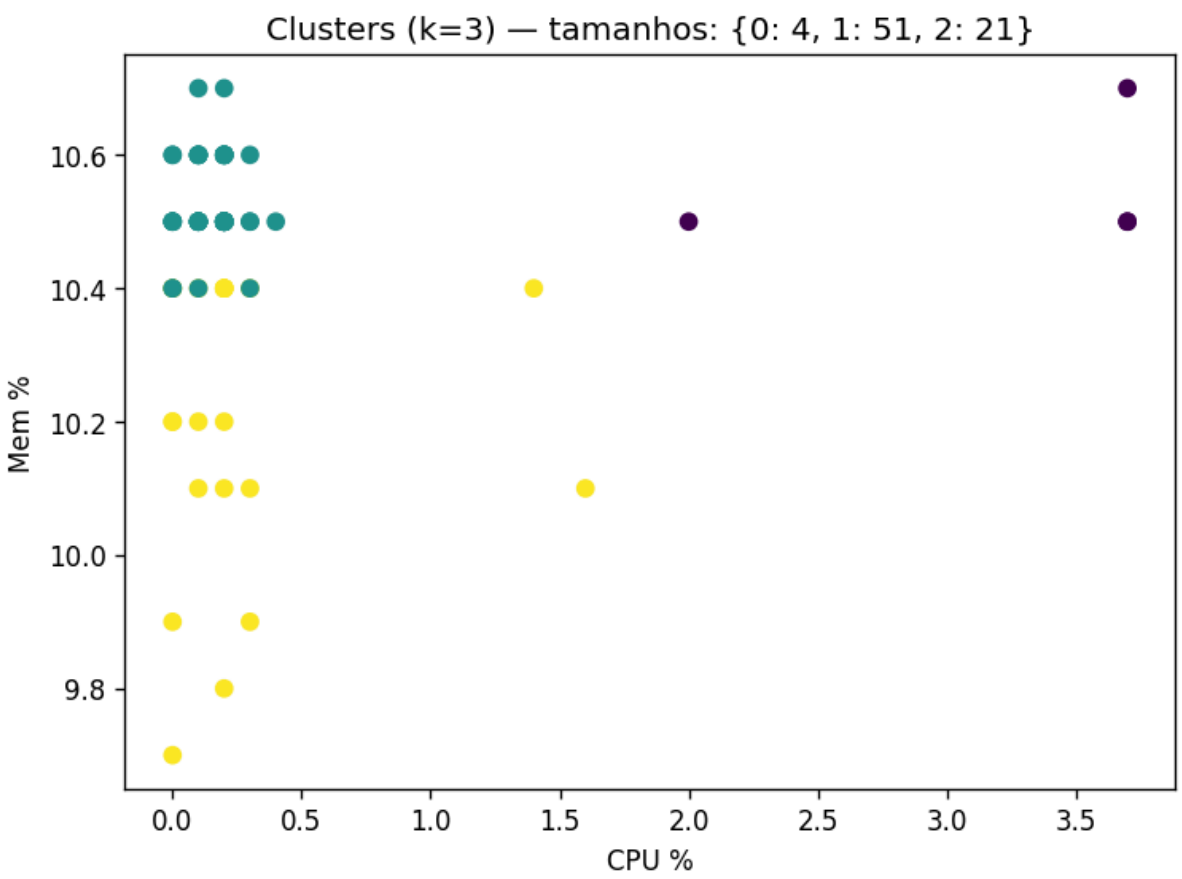


---

5. Clusters (KMeans)

- Eixo X: CPU (%)
- Eixo Y: Memória (%)
- O que mostra:

  Mostra agrupamentos de comportamento do sistema, como: Ocioso, normal ou Sobrecarga.
  Cada cor representa um grupo distinto.


Clusters (k=3) — tamanhos: {0: 4, 1: 51, 2: 21}
Mem %
10.6
10.4
10.2
10.0
9.8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
CPU %