

**INSTITUTO INFNET**  
**Integração Contínua, DevOps e Computação em Nuvem [25E1\_3]**

RICARDO JOSÉ NUNES

**PROJETO DA DISCIPLINA**  
APLICAÇÃO GUIA DE ESTUDO EXECUTANDO COM DOCKER SWARM NA AWS E  
MONITORIA COM GRAFANA, PROMETHEUS E CADVISOR USANDO GITHUB E  
DOCKER HUB COMO ESTEIRA DE CI/CD

## SUMÁRIO

<b>Introdução.....</b>	<b>2</b>
<b>Informações do Projeto.....</b>	<b>2</b>
<b>Docker Hub.....</b>	<b>3</b>
Passos para Criar, Taggear e Publicar uma Imagem no Docker Hub.....	3
<b>Ambiente Cloud AWS.....</b>	<b>5</b>
Deployment com 4 réplicas.....	5
Health Check, o Readness/Liveness probe do Docker Swarm.....	6
Volumes.....	7
<b>Configuração de CI/CD com GitHub Workflows.....</b>	<b>9</b>
<b>Visibilidade e exposição dos containers.....</b>	<b>11</b>
<b>Banco de Dados.....</b>	<b>13</b>
<b>Aplicação e Monitoramento.....</b>	<b>13</b>
Monitoramento com Prometheus, Grafana e cAdvisor.....	14
Dashboards de Monitoramento.....	15
<b>Stress Test.....</b>	<b>17</b>
Ambiente Pré-Stress Test.....	18
Ambiente durante Stress Test.....	19
Ambiente Pós-Stress Test.....	20
Resultado do Stress Test.....	21
<b>Conclusão.....</b>	<b>21</b>

# Introdução

A adoção de contêineres revolucionou a maneira como aplicações são desenvolvidas, implantadas e gerenciadas. Dentre as diversas soluções para orquestração de contêineres, o Docker Swarm se destaca como uma opção leve e integrada ao ecossistema Docker, permitindo a criação de clusters altamente disponíveis e escaláveis. Este projeto tem como objetivo demonstrar a implementação de uma aplicação previamente desenvolvida no framework Next pelo professor Renan Oliveira, utilizando Docker Swarm para orquestração de contêineres e GitHub Workflows como ferramenta de integração e entrega contínua (CI/CD).

Para isso, foi criada uma imagem Docker personalizada da aplicação e publicada no Docker Hub. Em seguida, essa imagem foi implantada em um cluster Docker Swarm, através de máquinas EC2 na AWS Cloud garantindo a escalabilidade com múltiplas réplicas. A exposição da aplicação foi feita via um serviço de ingress do Docker Swarm, permitindo o acesso externo e balanceamento ao sistema. Caso a aplicação evolua e necessite de um banco de dados, um serviço pode ser criado dentro do cluster apenas com comunicação interna.

Além disso, foram configurados mecanismos de monitoramento utilizando cAdvisor, Prometheus e Grafana, garantindo a visibilidade do desempenho da aplicação. Os dados coletados serão armazenados de forma persistente por meio de um Persistent Volume Claim (PVC). O Grafana será a única ferramenta de monitoramento acessível externamente, apresentando dashboards com métricas como uso de memória e CPU.

A automação do pipeline de CI/CD é realizada com GitHub Workflows, permitindo a construção, teste e implantação contínua da aplicação no ambiente de produção. Por fim, foram conduzidos testes de estresse para avaliar a resiliência da solução, e os resultados registrados e analisados por meio dos dashboards do Grafana, AWS Monitoring e painéis do JMeter, que foi a ferramenta utilizada para os testes de carga.

Com essa abordagem, o projeto proporcionará uma experiência prática e abrangente sobre a utilização do Docker Swarm e GitHub Workflows na automação e orquestração de aplicações em ambientes cloud.

## Informações do Projeto

Aqui podemos encontrar informações do projeto, como repositório, imagem do Docker Hub

- Repositório do código da aplicação, execução do pipeline de CI/CD, comandos e configurações para o Docker Swarm:  
<https://github.com/ricardo-jnunes/infnet-guia/tree/main>
- Imagem no Docker Hub:  
<https://hub.docker.com/repository/docker/nunes222/infnet-guia-estudo-ricardonunes/tags>
  - Comando para baixar imagem da aplicação: `docker pull nunes222/infnet-guia-estudo-ricardonunes:latest`
- Esteira CI/CD com GitHub Workflows:  
<https://github.com/ricardo-jnunes/infnet-guia/actions/runs/14251898951>

# Docker Hub

O Docker Hub é um repositório centralizado para armazenar, compartilhar e distribuir imagens Docker. Ele facilita a colaboração entre desenvolvedores e automatiza o deploy de aplicações baseadas em contêineres. Neste projeto, utilizaremos o Docker Hub para armazenar e distribuir a imagem da aplicação.

## Passos para Criar, Taggear e Publicar uma Imagem no Docker Hub

### 1. Criar a imagem Docker

Certifique-se de que seu projeto contém um Dockerfile configurado corretamente. Em seguida, execute o seguinte comando para construir a imagem:  
*docker build -t nome-da-imagem .*

### 2. Autenticar no Docker Hub

Antes de publicar a imagem, é necessário fazer login no Docker Hub:  
*docker login*  
Isso solicitará suas credenciais do Docker Hub.

### 3. Criar uma tag para a imagem

Para enviar a imagem ao Docker Hub, ela precisa ser associada a um repositório existente no serviço. O comando abaixo cria uma tag associando sua imagem ao repositório:  
*docker tag nome-da-imagem usuario-dockerhub/nome-da-imagem:latest*

### 4. Fazer o push da imagem

Após a criação da tag, envie a imagem para o Docker Hub com o seguinte comando:  
*docker push usuario-dockerhub/nome-da-imagem:latest*  
Esse comando enviará sua imagem para o repositório especificado.

Com esses passos, a imagem estará disponível para ser utilizada em diferentes ambientes, incluindo o cluster Docker Swarm configurado neste projeto.


Abaixo podemos ver a imagem infnet-guia-estudo-ricardonunes publicada para o usuário nunes222, que foi usado neste projeto.


Figura - Imagem infnet-guia-estudo-ricardonunes e suas tags publicadas no Docker Hub

[Repositories](#) / [infnet-guia-estudo-ricardonunes](#) / [General](#)

## nunes222/infnet-guia-estudo-ricardonunes

Last pushed less than a minute ago • Repository size: 220.4 MB



<https://github.com/ricardo-jnunes/infnet-guia> 

Add a category  

**General** Tags Image Management BETA Collaborators Webhooks Settings

### Tags

This repository contains 1 tag(s).


Tag	OS	Type	Pulled	Pushed
 latest		Image	less than 1 day	less than a minute


[See all](#)

Na próxima figura, podemos ver o uso da imagem hospedada no Docker Hub e uso dela no arquivo de Swarm/Deployment, usando modo Ingress:

Figura - Arquivo Dockerfile usando a imagem infnet-guia-estudo-ricardonunes

[infnet-guia](#) / [docker-compose-stack.yaml](#) 

 **ricardo-jnunes** core: use infnet-my-image

**Code** Blame 86 lines (86 loc) • 2.12 KB  Code 55% faster with GitHub Copilot

```
1  services:
2    app:
3      image: nunes222/infnet-guia-estudo-ricardonunes
4      deploy:
5        replicas: 4
6        update_config:
7          parallelism: 2
8          delay: 10s
9        restart_policy:
10         condition: on-failure
11      ports:
12        - target: 3000
13          published: 80
14          mode: ingress # Swarm faz o balanceamento de carga automaticamente entre as réplicas
15      networks:
16        - infnet-network
```

# Ambiente Cloud AWS

Foi configurado o Docker Swarm na Cloud da AWS, sendo instanciadas 4 máquinas EC2, sendo uma delas o Manager e mais três sendo Workers.

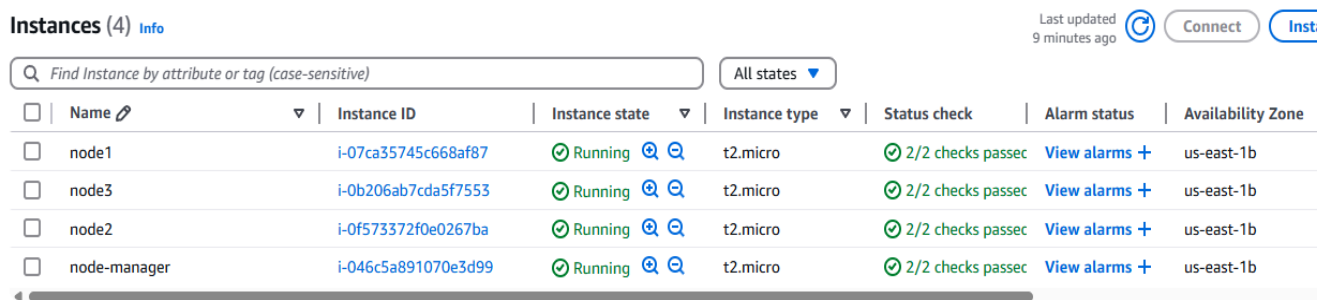
Houve também a instalação do Docker em todas EC2. Realizado a montagem de um EBS no na máquina do Manager que ficou responsável por ter os containers do Prometheus e Grafana, e os Workers com os containers de aplicação.

## Deployment com 4 réplicas

Para o Deployment da aplicação foi utilizado o comando *docker stack deploy*, o Docker Stack é uma funcionalidade do Docker Swarm que permite a implantação de aplicações complexas, aplicando quantidade de réplicas de um container, isso se dá utilizando arquivos de configuração YAML, para este projeto podemos ver as configurações arquivo *docker-compose-stack.yaml* disponível no link do repositório citado anteriormente.

Nas figuras seguintes poderemos ver o ambiente e o deployment realizado.

Figura - Quatro máquinas EC2 - Um Manager e três Workers



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
node1	i-07ca35745c668af87	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b
node3	i-0b206ab7cda5f7553	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b
node2	i-0f573372f0e0267ba	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b
node-manager	i-046c5a891070e3d99	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b

Figura - Verificando o Docker Swarm com seu Manager e seus respectivos Nodes.

```
[ec2-user@ip-172-31-92-226 ~]$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
7mybmjvwxuqiwwsg3wgegr3i *	ip-172-31-92-226.ec2.internal	Ready	Active	Leader	25.0.8
q3f2czow7wg574so2ks8t2caj	ip-172-31-93-136.ec2.internal	Ready	Active		25.0.8
qc632k3jli24o2ncdsbaln33w	ip-172-31-94-57.ec2.internal	Ready	Active		25.0.8
pbrf1lcufdoc4brvim87llktzh	ip-172-31-94-206.ec2.internal	Ready	Active		25.0.8

Figura - Criação da Stack/Deployment:

```
[ec2-user@ip-172-31-92-226 infnet-guia]$ docker stack deploy -c docker-compose-stack.yaml infnet-guide-stack
Ignoring deprecated options:
expose: Exposing ports is unnecessary - services on the same network can access each other's containers on any port.
Creating network infnet-guide-stack_infnet-network
Creating service infnet-guide-stack_app
Creating service infnet-guide-stack_database
Creating service infnet-guide-stack_cadvisor
Creating service infnet-guide-stack_prometheus
Creating service infnet-guide-stack_grafana
```

Figura - Verificando os serviços sendo iniciados

```
[ec2-user@ip-172-31-92-226 infnet-guia]$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
e7nmoabl56ol	infnet-guide-stack_app	replicated	4/4	nunes222/infnet-guia-estudo-ricardonunes:latest	*:80->3000/tcp
73dtbic08e8b	infnet-guide-stack_cadvisor	global	4/4	gcr.io/cadvisor/cadvisor:latest	
kn8yxgofol6h	infnet-guide-stack_database	replicated	0/1	mysql:latest	
xhqtisi5gmosj	infnet-guide-stack_grafana	replicated	0/1	grafana/grafana:latest	*:3000->3000/tcp
nqa8xcsex929	infnet-guide-stack_prometheus	replicated	0/1	prom/prometheus:latest	

```
[ec2-user@ip-172-31-92-226 infnet-guia]$
```

Figura - Serviços iniciados

```
[ec2-user@ip-172-31-92-226 infnet-guia]$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
7iir1rl1ll5e	infnet-guide-stack_app	replicated	4/4	nunes222/infnet-guia-estudo-ricardonunes:latest	*:80->3000/tcp
lwsyojvkkw5i	infnet-guide-stack_cadvisor	global	4/4	gcr.io/cadvisor/cadvisor:latest	
e6wx88nc3dww	infnet-guide-stack_grafana	replicated	1/1	grafana/grafana:latest	*:3000->3000/tcp
7obr6a0u14ut	infnet-guide-stack_prometheus	replicated	1/1	prom/prometheus:latest	

Figura - Verificando o container da aplicação Infnet-Guia e o CAdvisor.

```
[ec2-user@ip-172-31-94-57 ~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c076abab7da0	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	2 minutes ago	Up 2 minutes	3000/tcp	infnet-guide-stack_app.4.e1nryeltcbugiorc0m8e7u9h6
1bb3cc8b3fa6	gcr.io/cadvisor/cadvisor:latest	"/usr/bin/cadvisor -..."	2 minutes ago	Up 2 minutes (healthy)	8080/tcp	infnet-guide-stack_cadvisor.qc632k3ji24o2ncdsbaln
33w.wfqo4a76r13wxn0iesapvm0vk						

## Health Check, o Readiness/Liveness probe do Docker Swarm

O Docker Swarm permite a configuração de health checks para garantir que os serviços implantados estejam operacionais e saudáveis. Essa funcionalidade monitora periodicamente os contêineres e pode acionar ações específicas, como reiniciar contêineres defeituosos ou acionar um rollback em caso de falha.

Com essa configuração, o Docker Swarm assegura que apenas instâncias saudáveis do serviço sejam mantidas em execução, proporcionando maior confiabilidade e resiliência ao sistema.

Figura - Health Check do Docker Swarm avaliando a saúde dos containers da aplicação

```
services:
  app:
    image: nunes222/infnet-guia-estudo-ricardonunes
    deploy:
      replicas: 4
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
    healthcheck:
      test: ["CMD", "wget", "--spider", "-q", "http://localhost:3000/"]
      interval: 10s
      timeout: 30s
      retries: 3
      start_period: 20s
    ports:
      - target: 3000
        published: 80
        mode: ingress # Swarm faz o balanceamento de carga automaticamente entre as réplicas
  networks:
    - infnet-network
```

Figura - Health Check do Docker Swarm avaliando a saúde dos containers da aplicação

```
[ec2-user@ip-172-31-92-226 infnet-guia]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
a69ac37511a5	gcr.io/cadvisor/cadvisor:latest	"/usr/bin/cadvisor -..."	39 seconds ago	Up 35 seconds (healthy)
infnet-guide-stack_cadvisor.7mybmjvwuqiwwsg3wgegr3i.vjghzdhd4fziy26z8nmuq3jow	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	39 seconds ago	Up 36 seconds (health: starting)
infnet-guide-stack_app.1.itq2e3xtpzrwh721618dlu6r	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	39 seconds ago	Up 35 seconds (health: starting)
deb8b1fe23d3	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	39 seconds ago	Up 35 seconds (health: starting)
infnet-guide-stack_app.4.wplx4raet4tg84yddm9fwdu9	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	39 seconds ago	Up 35 seconds (health: starting)
8d31bc23ca34	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	39 seconds ago	Up 35 seconds (health: starting)
infnet-guide-stack_app.2.lxsnlv4izptjc58wblidsm2rr6	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	39 seconds ago	Up 37 seconds (health: starting)
b2bf5d8dc459	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	39 seconds ago	Up 37 seconds (health: starting)
infnet-guide-stack_app.3.iiiunaf6lchyu5mx5opw7sex2	grafana/grafana:latest	"/run.sh"	40 seconds ago	Up 37 seconds
1814e698a75e	infnet-guide-stack_grafana.1.gga7glarsodbudrjneicqehdj	"/bin/prometheus --c..."	40 seconds ago	Up 37 seconds
4b3397086b40	prom/prometheus:latest			
infnet-guide-stack_prometheus.1.kqjv9by006rwjs3c9hotdr7kk				

Figura - Containers em estado de healthy

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
cc17744280d2	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	33 seconds ago	Up 31 seconds (healthy)	3000/tcp
uide-stack_app.1.l3xzdes67cmi4pwrqcp860y8p	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	33 seconds ago	Up 31 seconds (healthy)	3000/tcp
baffa318129d	nunes222/infnet-guia-estudo-ricardonunes:latest	"docker-entrypoint.s..."	33 seconds ago	Up 31 seconds (healthy)	3000/tcp
uide-stack_app.2.bdluvh05aj59t38f2msuf1b1x					

## Volumes

O uso de volumes no Docker Swarm é essencial para garantir a persistência de dados em serviços que precisam armazenar informações além do ciclo de vida do contêiner. Volumes permitem que os dados permaneçam disponíveis mesmo quando um contêiner é reiniciado ou removido.

Para este projeto foi adicionado um EBS ao Node Manager, onde roda o Prometheus, Grafana e o banco de dados MySQL.

Alguns comandos usados no EC2 - Node Manager

### Montagem

- `sudo mkfs -t xfs /dev/sdf`
- `sudo mkdir /mnt/data`
- `sudo mount /dev/sdf /mnt/data`

### Crie os diretórios necessários

- `mkdir /mnt/data/prometheus_data`
- `mkdir /mnt/data/grafana_data`
- `mkdir /mnt/data/mysql_data`








Figura - EBS montado no EC2 Node Manager

**Volumes (1/5)** [Info](#)

Saved filter sets  
Choose filter set ▼ Q Search

<input type="checkbox"/>	Name	Volume ID	Type	Size
<input checked="" type="checkbox"/>	-	vol-0d77b77771e4f63d2	gp3	4 GiB
<input type="checkbox"/>	-	vol-09ae0903d497008b2	gp3	8 GiB
<input type="checkbox"/>	-	vol-00ef87383cd3aa782	gp3	8 GiB
<input type="checkbox"/>	-	vol-01314704b7b61e868	gp3	8 GiB

**Volume ID: vol-0d77b77771e4f63d2**

Details	Status checks	Monitoring	Tags
<b>Volume ID</b>  vol-0d77b77771e4f63d2	<b>Size</b>  4 GiB		
<b>AWS Compute Optimizer finding</b>  Opt-in to AWS Compute Optimizer for recommendation s.   <a href="#">Learn more</a> 	<b>Volume state</b>  In-use		
<b>Fast snapshot restored</b> No	<b>Availability Zone</b> us-east-1b		
<b>Attached resources</b> i-046c5a891070e3d99 (node-manager): /dev/sdf (attached)	<b>Outposts ARN</b> -		

As configurações dos volumes podem ser conferidas no arquivo docker-compose-stack.yaml com essas configurações pode ser vista no link do repositório no GitHub citado anteriormente.

Figura - Volume configurado para o Grafana

```
grafana:
  image: grafana/grafana
  deploy:
    placement:
      constraints:
        - node.role==manager
  ports:
    - 3000:3000
  environment:
    - GF_SECURITY_ADMIN_USER=admin
    - GF_SECURITY_ADMIN_PASSWORD=grafana
  volumes:
    - ./grafana:/etc/grafana/provisioning/datasources
    - /mnt/data/grafana_data:/var/lib/grafana
```

# Configuração de CI/CD com GitHub Workflows

Para o pipeline de integração e entrega contínua (CI/CD), usamos o GitHub Workflows que permite a automação do build e deploy de aplicações. Neste projeto, utilizamos uma configuração para realizar o build da imagem e enviar a imagem ao Docker Hub, assim automatizando a parte anteriormente vista Passos para Criar, Taguear e Publicar uma Imagem no Docker Hub

Através do Workflows, os eventos de pull request e de push à branches com prefixo de release, automaticamente disparam um build da imagem da aplicação e o disponibiliza no Docker Hub para ser usado nos containers rodando no Docker Swarm na AWS.

- O arquivo de configuração pode ser encontrado no repositório, no diretório de workflows:  
<https://github.com/ricardo-jnunes/infnet-guia/blob/main/.github/workflows/docker-image.yml>

E para tal configuração funcionar corretamente, devemos usar as secrets do repositório no GitHub que possuem as credenciais do Docker Hub.

Figura - Secrets configuradas no GitHub

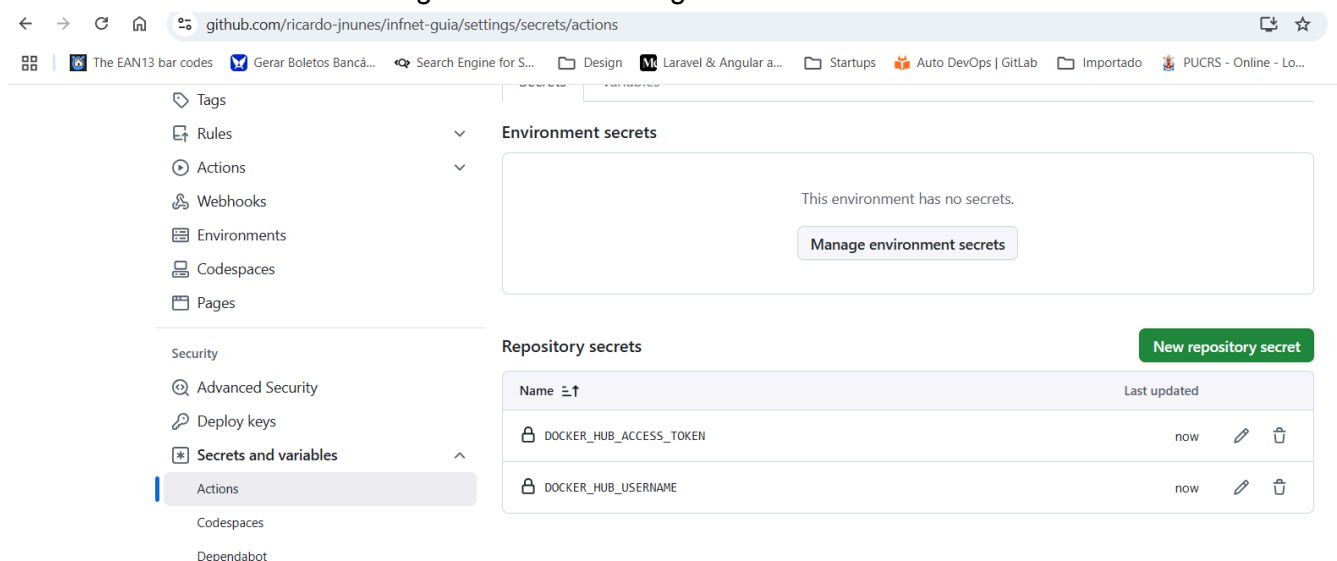
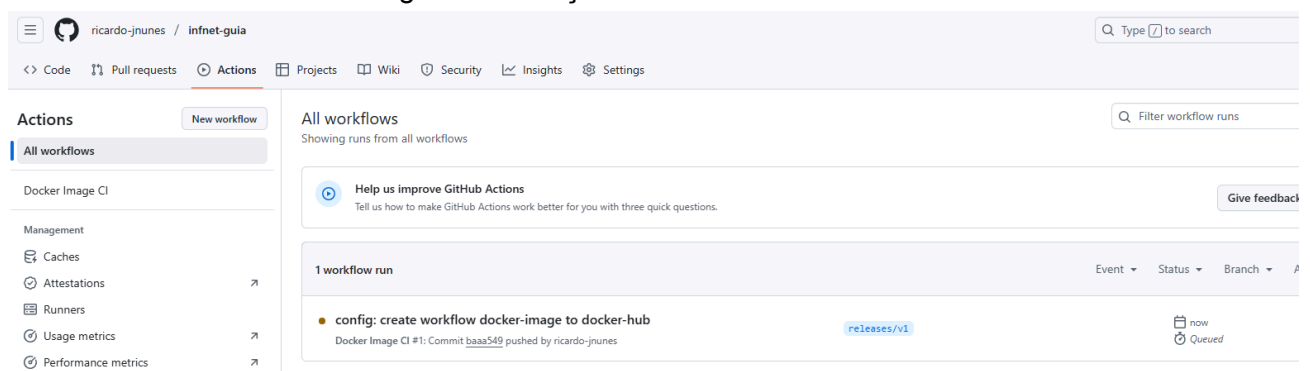


Figura - Execução da esteira CI/CD



1 workflow run

Event
Status
Branch
Actor

config: create workflow docker-image to docker-hub

releases/v1

1 minute ago  
In progress

...

Docker Image CI #1: Commit [baaa549](#) pushed by ricardo-jnunes

Figura - Execução com sucesso da esteira CI/CD

✓ config: create workflow docker-image to docker-hub #1

Summary

Jobs

✓ build

Run details

Usage

Workflow file

Triggered via push 3 minutes ago

Status

ricardo-jnunes pushed

baaa549

releases/v1

Success

docker-image.yml

on: push

✓ build

1m 20s

Figura - Passos da esteira CI/CD

build

succeeded 3 minutes ago in 1m 20s

> ✓ Set up job

> ✓ Checkout repository

> ✓ Set Docker tag dynamically

> ✓ Log in to Docker Hub

> ✓ Build the Docker image

> ✓ Push the Docker image to Docker Hub

> ✓ Post Log in to Docker Hub

> ✓ Post Checkout repository

> ✓ Complete job

Figura - Imagem publicada com sucesso no Docker Hub através da esteira CI/CD

The screenshot shows the Docker Hub interface for the repository 'nunes222/my-app'. The top navigation bar is blue with 'My Hub' and a search bar. Below, the 'Repositories' section shows a list of repositories in the 'nunes222' namespace. The 'nunes222/my-app' repository is highlighted, showing it was last pushed 14 minutes ago and contains an image. Below this, the repository details for 'nunes222/my-app' are shown, including a description and category section. The 'General' tab is active, displaying the 'Tags' section with a table of tags. The table shows one tag, 'releases-v1', which is an image type, pushed less than 1 day ago, 6 minutes ago. A 'DOCKER SCOUT INACTIVE' banner is visible in the top right of the tags section.

Name	Last Pushed	Contains
nunes222/my-app	14 minutes ago	IMAGE
nunes222/infnet-guia-estudo-ricardonunes	7 days ago	IMAGE

**nunes222/my-app** ⓘ  
Last pushed 1 minute ago

[Add a description](#) ⓘ ⓘ  
[Add a category](#) ⓘ ⓘ

**General** | Tags | Image Management BETA | Collaborators | Webhooks | Settings

**Tags** DOCKER SCOUT INACTIVE [Activate](#)

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
releases-v1	linux	Image	less than 1 day	6 minutes

[See all](#)

## Visibilidade e exposição dos containers

No ambiente Docker Swarm, é importante definir corretamente quais contêineres devem estar acessíveis externamente e quais devem permanecer internos para garantir segurança e organização da infraestrutura.

- Serviços internos: Contêineres que não devem ser expostos ao público devem utilizar o modo de rede ClusterIP. Isso é útil para bancos de dados, serviços de cache, e outros componentes que apenas a aplicação precisa acessar.
- Serviços expostos ao usuário: Aplicações que precisam estar acessíveis ao cliente devem ser expostas através de NodePort ou balanceadores de carga.

Neste projetos deixamos o Prometheus e o Banco de Dados MySQL apenas visível internamente, como podemos ver nas figuras demonstrando as configurações, o arquivo docker-compose-stack.yaml com essas configurações pode ser vista no link do repositório no GitHub citado anteriormente.

Os outros serviços com Grafana e a aplicação Infnet - Guia de Estudos são expostas para garantir que as métricas e a aplicação possam ser visíveis aos usuários.

Figura - Configuração do Stack/Deployment para garantir que o Banco de Dados MySQL apenas seja acessível dentro do Cluster.

```
database:
  image: mysql:latest
  deploy:
    placement:
      constraints:
        - node.role == manager
    restart_policy:
      condition: on-failure
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: infnet-studies-mngmt-db
    MYSQL_USER: myuser
    MYSQL_PASSWORD:
  expose:
    - 3306 #Only accessible by services in the same network
  volumes:
    - /mnt/data/mysql_data:/var/lib/mysql
  networks:
    - infnet-network
```

Figura - Configuração do Stack/Deployment para garantir que o Prometheus apenas seja acessível dentro do Cluster

```
prometheus:
  image: prom/prometheus
  deploy:
    placement:
      constraints:
        - node.role==manager
    restart_policy:
      condition: on-failure
  command:
    - '--config.file=/etc/prometheus/prometheus.yml'
  expose:
    - 9090 #Only accessible by services in the same network
  volumes:
    - ./prometheus:/etc/prometheus/
    - /mnt/data/prometheus_data:/prometheus
  networks:
    - infnet-network
```

Figura - Regras de Segurança na AWS - Permite apenas a aplicação Web executando na porta 80 e o Grafana na porta 3000

Inbound rules <small>Info</small>							
Security group rule ID	Type <small>Info</small>	Protocol <small>Info</small>	Port range <small>Info</small>	Source <small>Info</small>	Description - optional <small>Info</small>		
sgr-053c232f0ab35f944	SSH	TCP	22	Custom	Q		Delete
					0.0.0.0/0 X		
sgr-0b570d7bb59146c26	HTTP	TCP	80	Custom	Q		Delete
					0.0.0.0/0 X		
sgr-0a07a7b31c1fc2e28	HTTPS	TCP	443	Custom	Q		Delete
					0.0.0.0/0 X		
sgr-057e230ff1239ffe4	Custom TCP	TCP	2377	Custom	Q	Docker	Delete
					0.0.0.0/0 X		
-	Custom TCP	TCP	3000	Anywh...	Q 0.0.0.0/0	Grafana	Delete
					0.0.0.0/0 X		

## Banco de Dados

Como vimos anteriormente o banco de dados MySQL foi configurado para ser visível apenas internamente no Cluster e garantindo que apenas tenhamos uma instância executando no Manager.

Neste projeto o banco de dados foi criado puramente para testes e não é usado na aplicação atual, dessa forma permitindo que quando a aplicação evoluir estejamos preparado para suportar persistência de dados.

Dessa forma, caso a aplicação necessite de um banco de dados MySQL, podemos configurá-lo como um serviço interno no cluster Docker Swarm. Para isso, criamos um serviço MySQL com armazenamento persistente e protegemos o acesso para que apenas outros contêineres dentro do cluster possam se conectar a ele.

Também garantimos a persistência das informações do banco de dados MySQL através da configuração do volume montado no EBS.

Figura - Montagem do EBS para persistência do MySQL

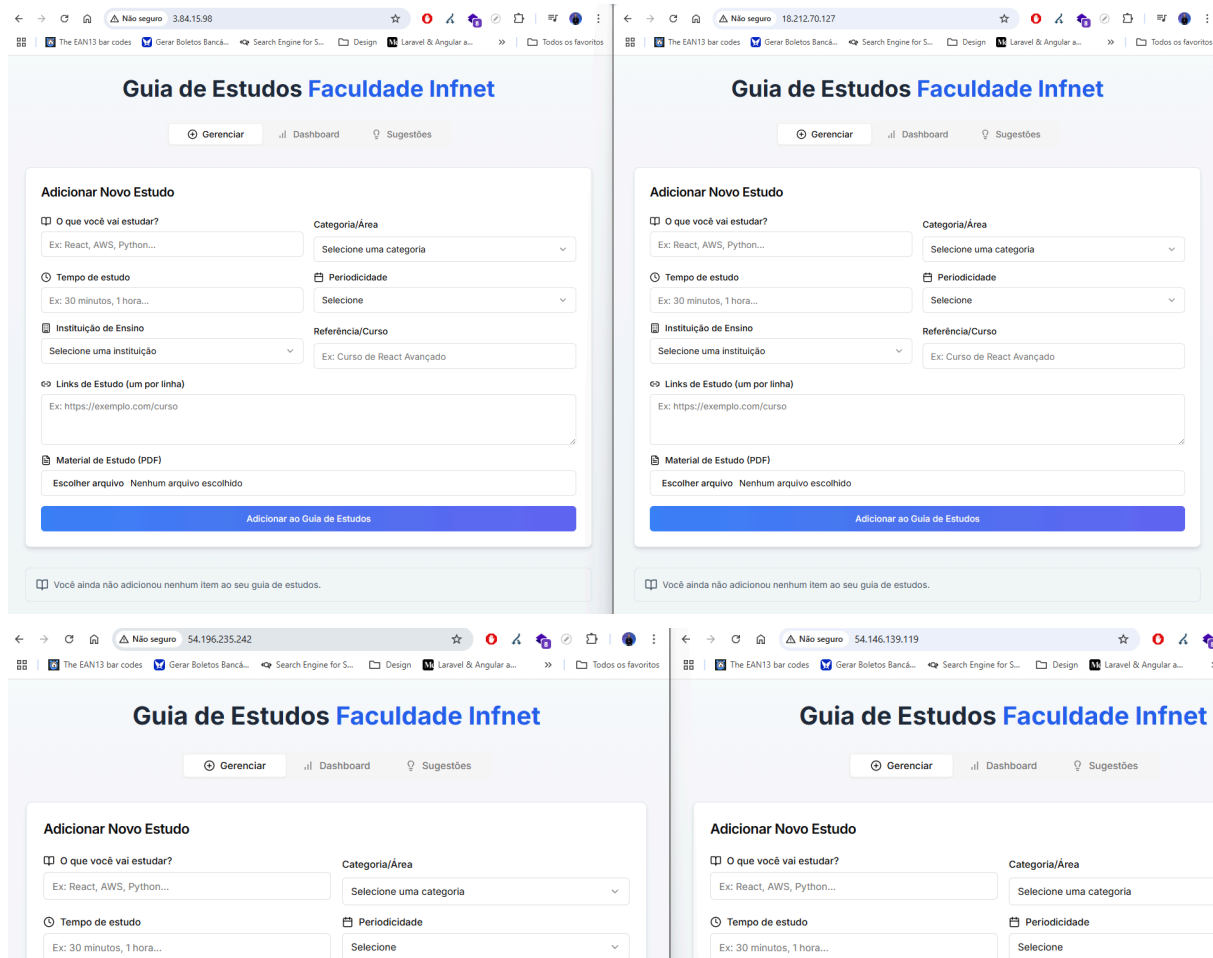
```
[ec2-user@ip-172-31-92-226 infnet-guia]$ ls /mnt/data/mysql_data
'#ib_16384_0.dblwr'  '#innodb_temp'  binlog.index  client-cert.pem  ibdata1  mysql.ibd  private_key.pem
'#ib_16384_1.dblwr'  auto.cnf        ca-key.pem    client-key.pem   ibtmp1   mysql_upgrade_history  public_key.pem
'#innodb_redo'      binlog.000001  ca.pem        ib_buffer_pool   mysql    performance_schema  server-cert.pem
```

## Aplicação e Monitoramento

A aplicação desenvolvida pelo professor Renan Oliveira e proveniente de um fork no GitHub foi utilizada para este projeto (link ao repositório pode ser encontrada na seção Informações do Projeto), a aplicação utiliza Next.js, um framework moderno para React que permite renderização híbrida, alto desempenho e flexibilidade. A aplicação será containerizada utilizando Docker e implantada no cluster Docker Swarm.

Após criar e publicar a imagem no Docker Hub, a aplicação Next.js foi implantada no cluster Docker Swarm com 4 réplicas.

Figura - Aplicação acessível entre os nós configurados

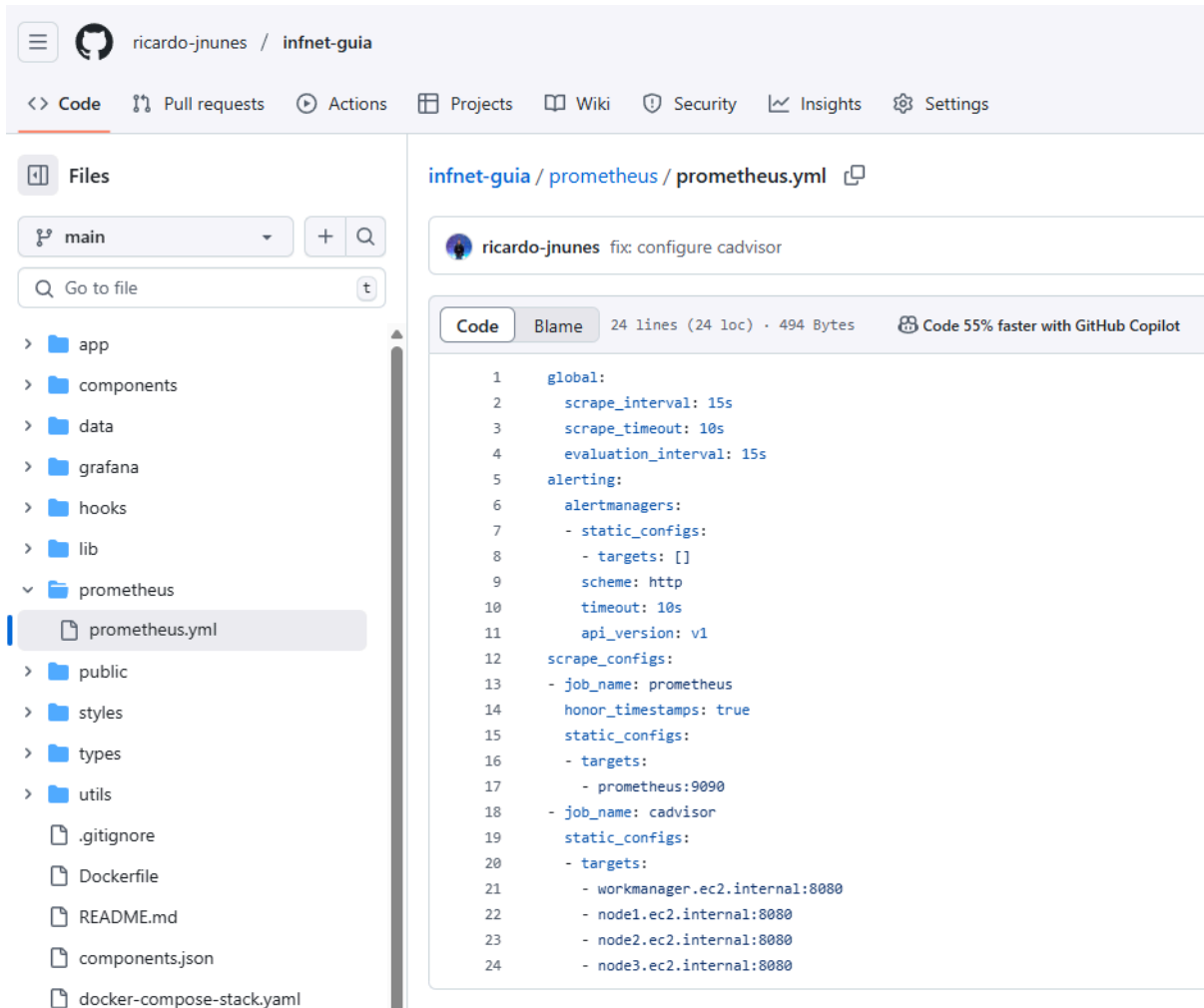


## Monitoramento com Prometheus, Grafana e cAdvisor

Para garantir a visibilidade do desempenho da aplicação e da infraestrutura, utilizamos um stack de monitoramento composto por Prometheus, Grafana e cAdvisor.

- O cAdvisor é responsável por coletar métricas detalhadas dos contêineres, como uso de CPU, memória, disco e rede.
- O Prometheus coleta métricas do cAdvisor e outros serviços do cluster, armazenando os dados para análise e alerta. Um arquivo de configuração `prometheus.yml` deve ser criado com a definição dos alvos de monitoramento.
- O Grafana será utilizado para visualizar métricas coletadas pelo Prometheus. Dentro do Grafana, podemos configurar um datasource apontando para o Prometheus e criar dashboards personalizados para monitorar métricas da aplicação e infraestrutura.

Figura - Configuração do Scrape do Prometheus para realizar a captura das métricas de si próprio e do cAdvisor:



## Dashboards de Monitoramento

Para monitoramento das aplicações foram criadas dois dashboards, uma para acompanharmos a saúde do Prometheus e outro da aplicação em NextJS.

Também poderemos acompanhar através dos dashboards padrões da AWS.



Figura - Dashboards de Monitoramento do Prometheus

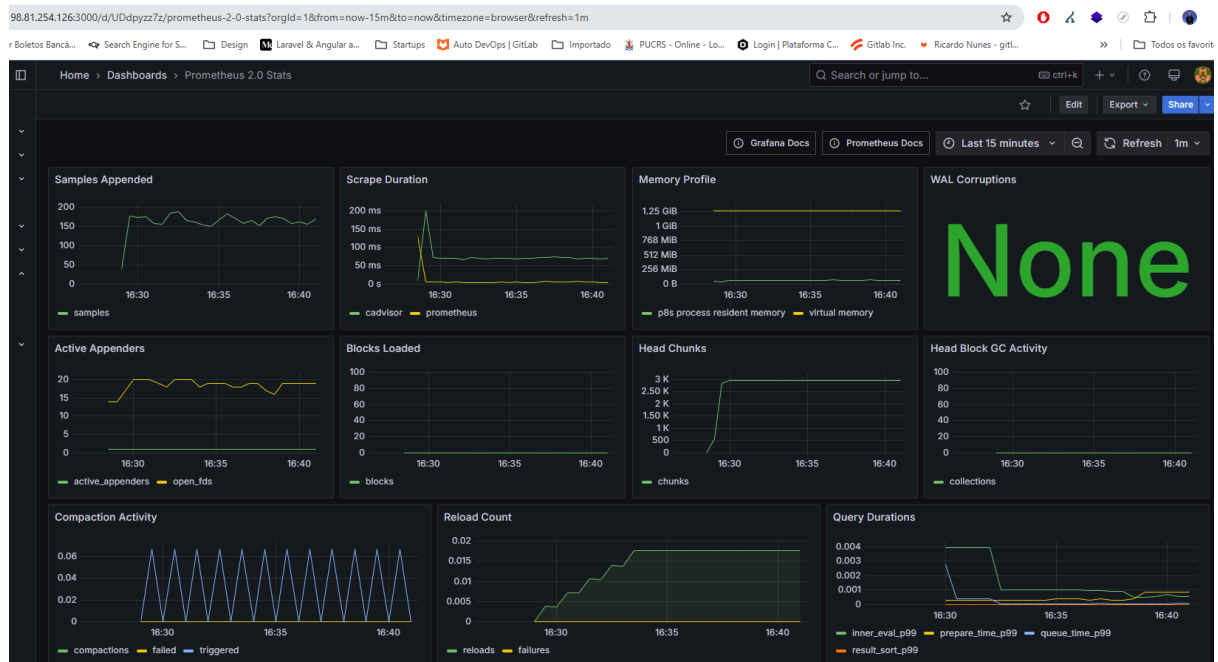


Figura - Dashboards de Monitoramento dos Containers

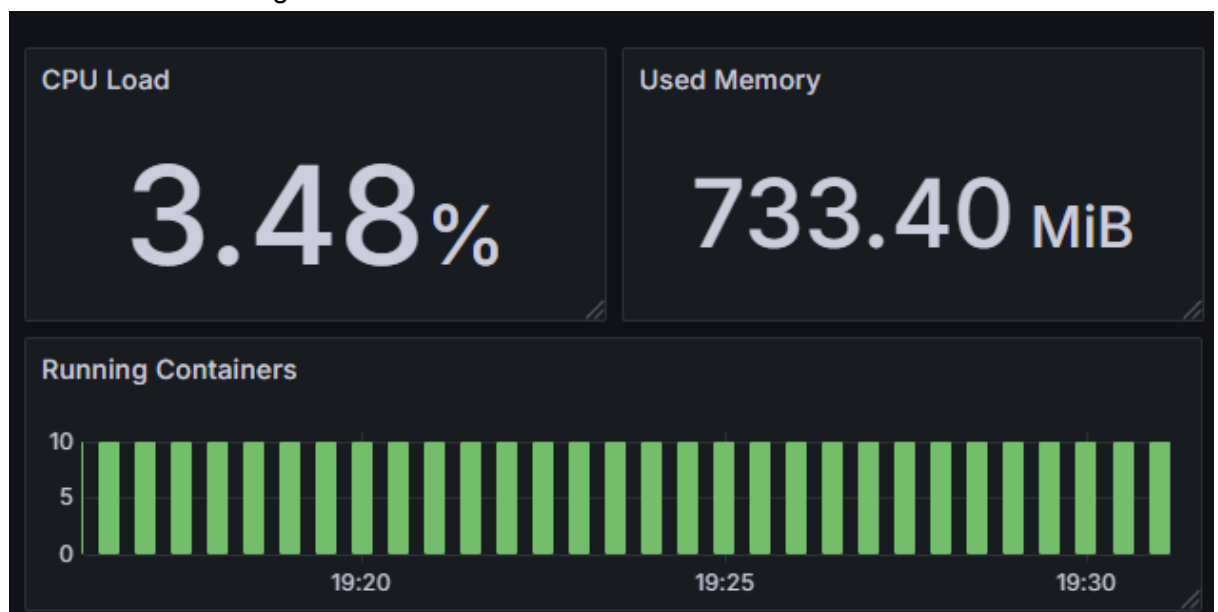


Figura - Dashboards de Monitoramento dos Containers da aplicação - 4 réplicas

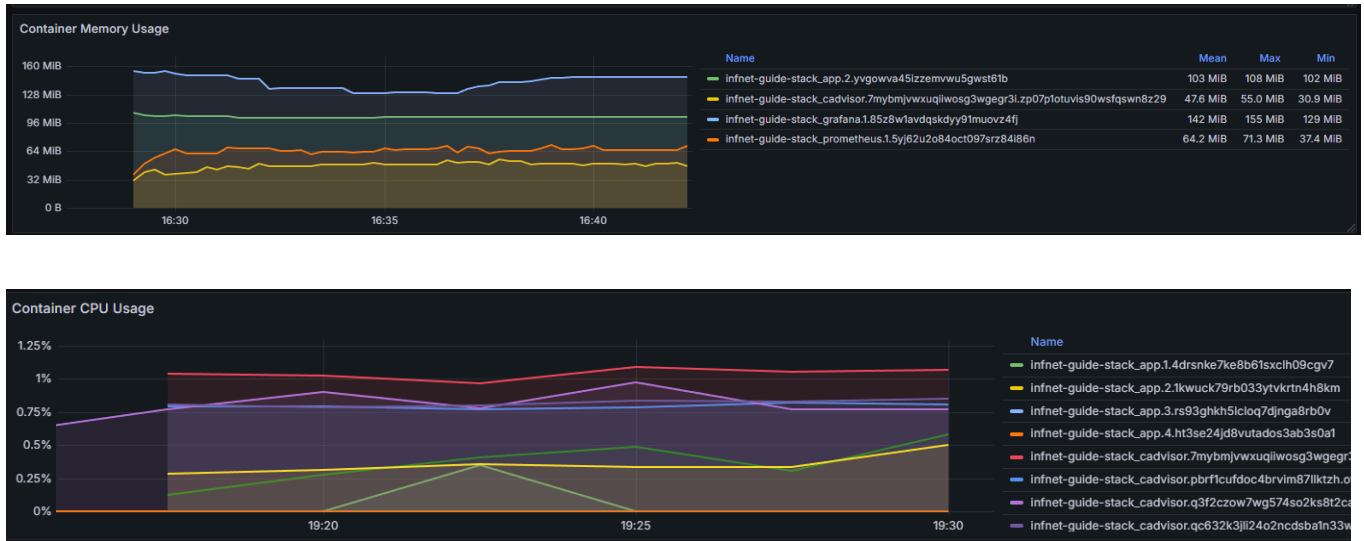
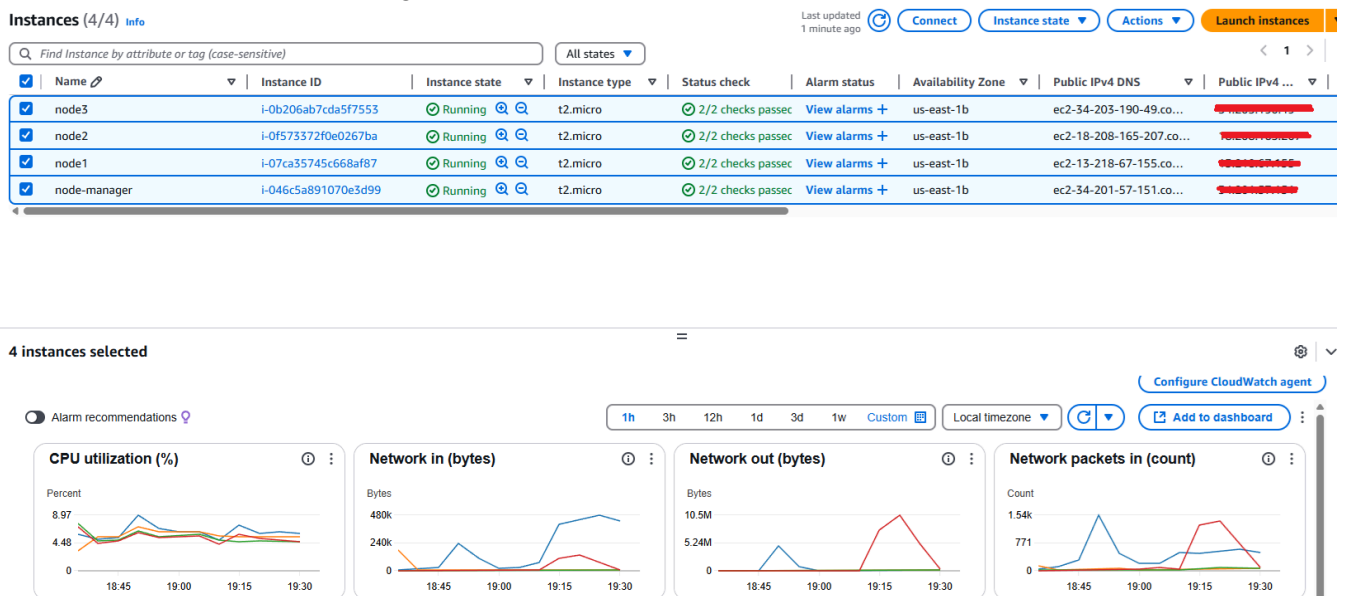


Figura - Monitoria padrão na AWS



## Stress Test

Para validar a resiliência e o desempenho da aplicação sob carga, utilizaremos o Apache JMeter para simular múltiplas requisições concorrentes.

### Configuração

- Usuários virtuais: 100
- Ramp-up: 10 segundos
- Duração: 60 segundos

Observação: O teste tem duração de 60 segundos, devido a restrições de orçamento na Cloud AWS.

Figura - Usuário/Threads ao longo do tempo

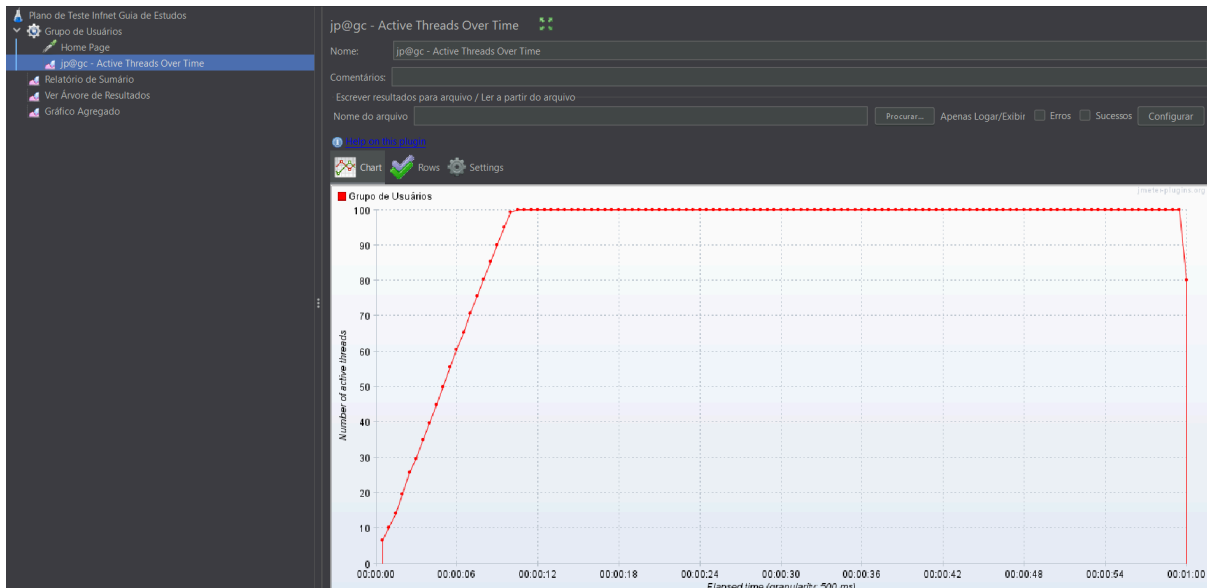
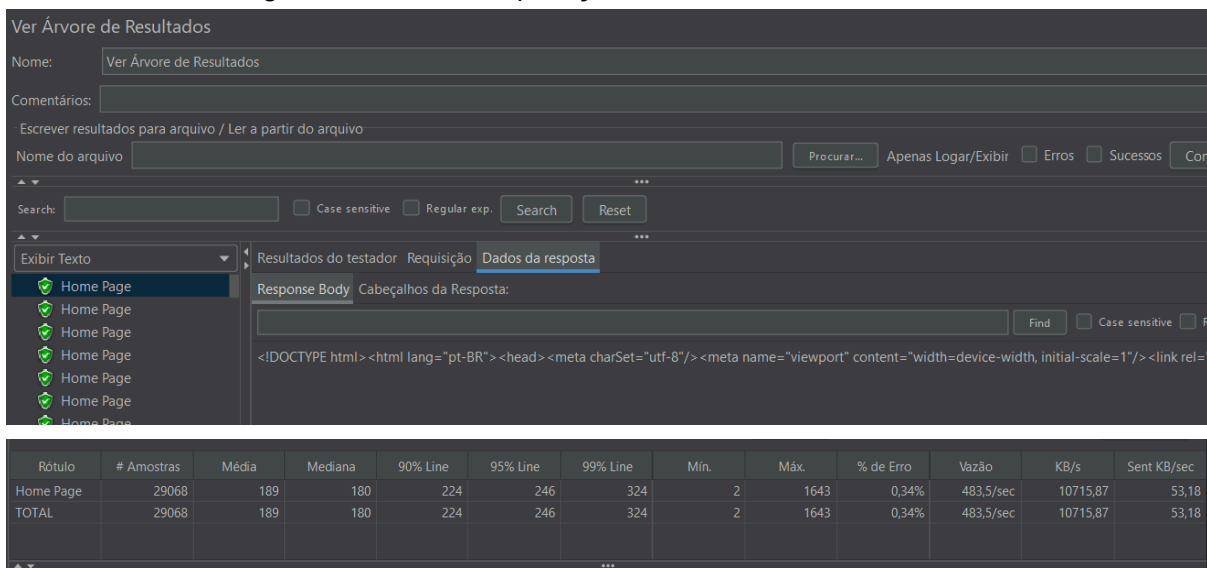
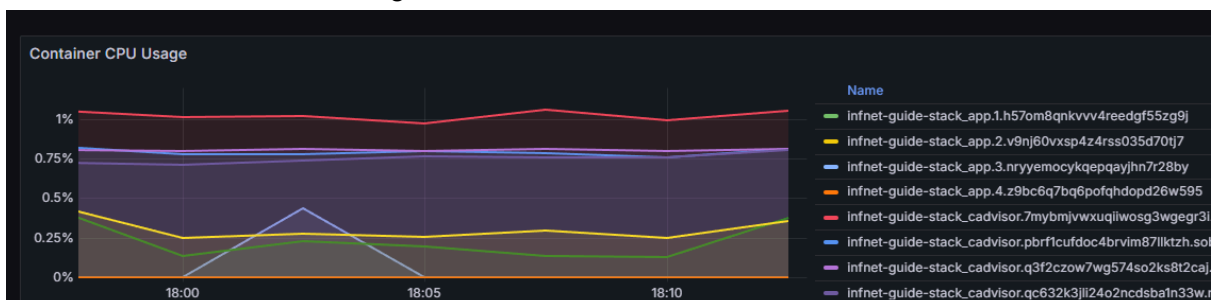


Figura - Retorno da aplicação NextJS - Guia de Estudos



## Ambiente Pré-Stress Test

Figura - Uso de CPU dos Containers



## Ambiente durante Stress Test

Figura - Uso de recursos na AWS durante Stress Test

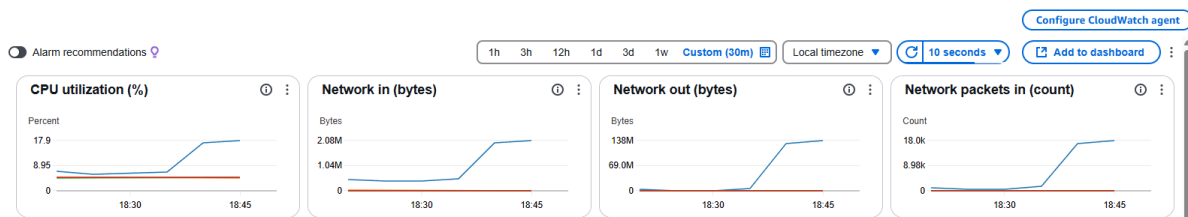
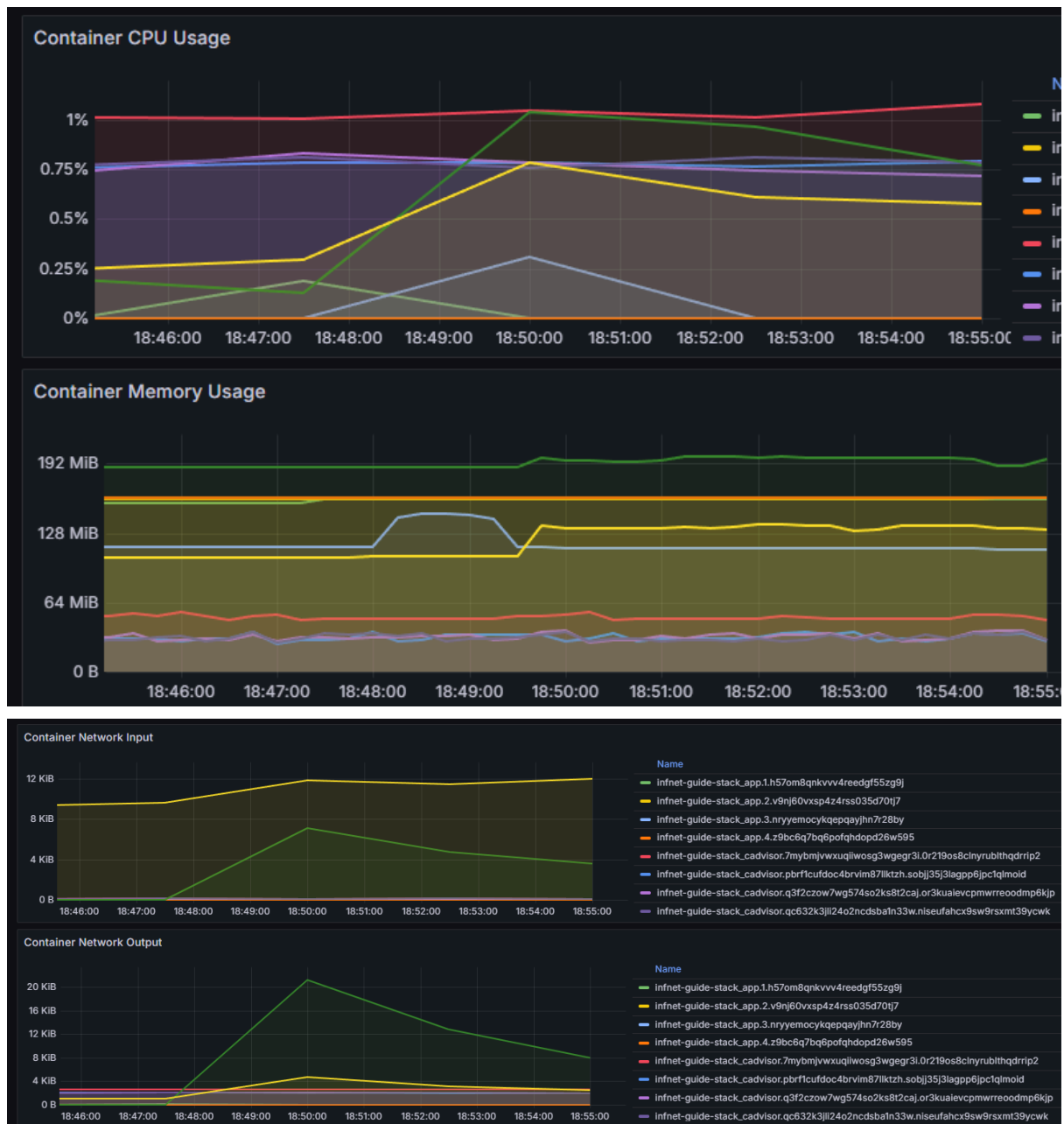


Figura - Uso de recursos visto no Grafana durante Stress Test

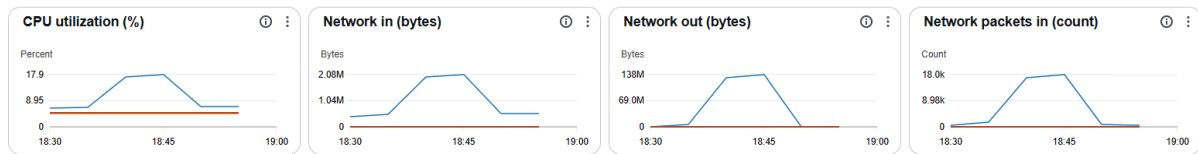


## Ambiente Pós-Stress Test

Figura - Uso de recursos no Grafana após Stress Test



Figura - Uso de recursos na AWS após Stress Test



## Resultado do Stress Test

Os resultados dos testes de estresse demonstraram que a aplicação e sua infraestrutura foram capazes de lidar com as cargas de requisições sem degradação significativa de desempenho. As métricas coletadas no Grafana indicaram que os contêineres mantiveram um consumo de recursos estável e dentro dos limites. Além disso, o sistema se mostrou resiliente, recuperando-se automaticamente após os picos de carga sem necessidade de intervenção manual. Isso valida a eficácia da solução implementada com Docker Swarm, garantindo alta disponibilidade e escalabilidade para a aplicação.

## Conclusão

Este projeto demonstrou a viabilidade do uso de Docker Swarm para orquestração de contêineres em um ambiente distribuído, garantindo escalabilidade e alta disponibilidade para uma aplicação Next.js. Além disso, a integração com GitHub Workflows possibilitou um fluxo contínuo de integração e entrega (CI/CD), facilitando a implantação e manutenção da aplicação.

O monitoramento eficiente com Prometheus, Grafana e cAdvisor permitiu a coleta de métricas importantes, proporcionando maior visibilidade sobre o desempenho da aplicação. Por fim, os testes de estresse com JMeter mostraram que o sistema foi capaz de lidar com altas cargas de tráfego sem comprometer a estabilidade, reforçando a robustez da solução implementada.

Dessa forma, este projeto serviu como uma experiência prática e abrangente sobre a implantação de aplicações modernas utilizando contêineres, automação de pipelines e monitoramento avançado.