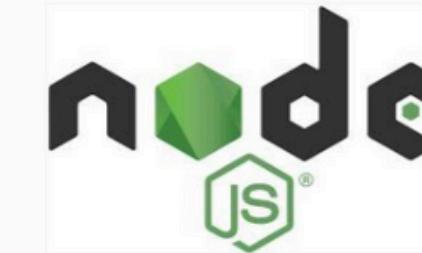


Containerização de Aplicações Node.js com Docker: Benefícios e processos.



Introdução ao Tema

O que é Node.js



- Node.js: Plataforma amplamente reconhecida para o desenvolvimento de backend.
- Baseada no motor V8 do Google Chrome.
- Permite a utilização de JavaScript no lado do servidor.
- Facilita o desenvolvimento ao empregar a mesma linguagem tanto no front-end quanto no back-end.
- Apresenta alta performance e escalabilidade.
- Persistem desafios em relação à otimização e segurança em sistemas de grande porte.

O que é Docker



- Revolucionou o desenvolvimento e a implantação de aplicativos.
- Utiliza a containerização para encapsular aplicações e suas dependências.
- Proporciona portabilidade e consistência entre diferentes ambientes.
- Facilita a criação de ambientes de teste que são reproduzíveis.
- Melhora o controle sobre o ciclo de vida do software.

Benefícios da Containerização

A containerização com **Docker** oferece uma série de benefícios que tornam o desenvolvimento e a implantação de aplicações mais eficientes. Um dos principais é a **portabilidade**, que permite que aplicações rodem da mesma forma em diferentes ambientes, independentemente do sistema operacional. Segundo *P. Mell e T. Grance (2011)*, a containerização aumenta a eficiência e facilita o desenvolvimento em múltiplos sistemas, **sem a necessidade de reconfigurações constantes**.

Além disso, Docker garante a **consistência** entre os ambientes de desenvolvimento, teste e produção. Conforme *L. Bass, I. Weber e L. Zhu (2015)*, as práticas de DevOps apoiadas por containers asseguram que os ambientes sejam idênticos, minimizando erros que podem ocorrer devido a diferenças entre esses ambientes.



Por fim, o **isolamento** oferecido por Docker é um grande diferencial. Cada container roda de forma independente, o que significa que **problemas em uma aplicação não afetam outras que estão rodando no mesmo servidor**. Como *J. Merkel (2014)* explica, esse isolamento aumenta a segurança e a confiabilidade dos ambientes de produção.

Processo de Containerização

Inicialização o projeto Node.js e criação do arquivo server.js

```
tomaz@debian:~/projetos/my-node-app$ cat package.json
```

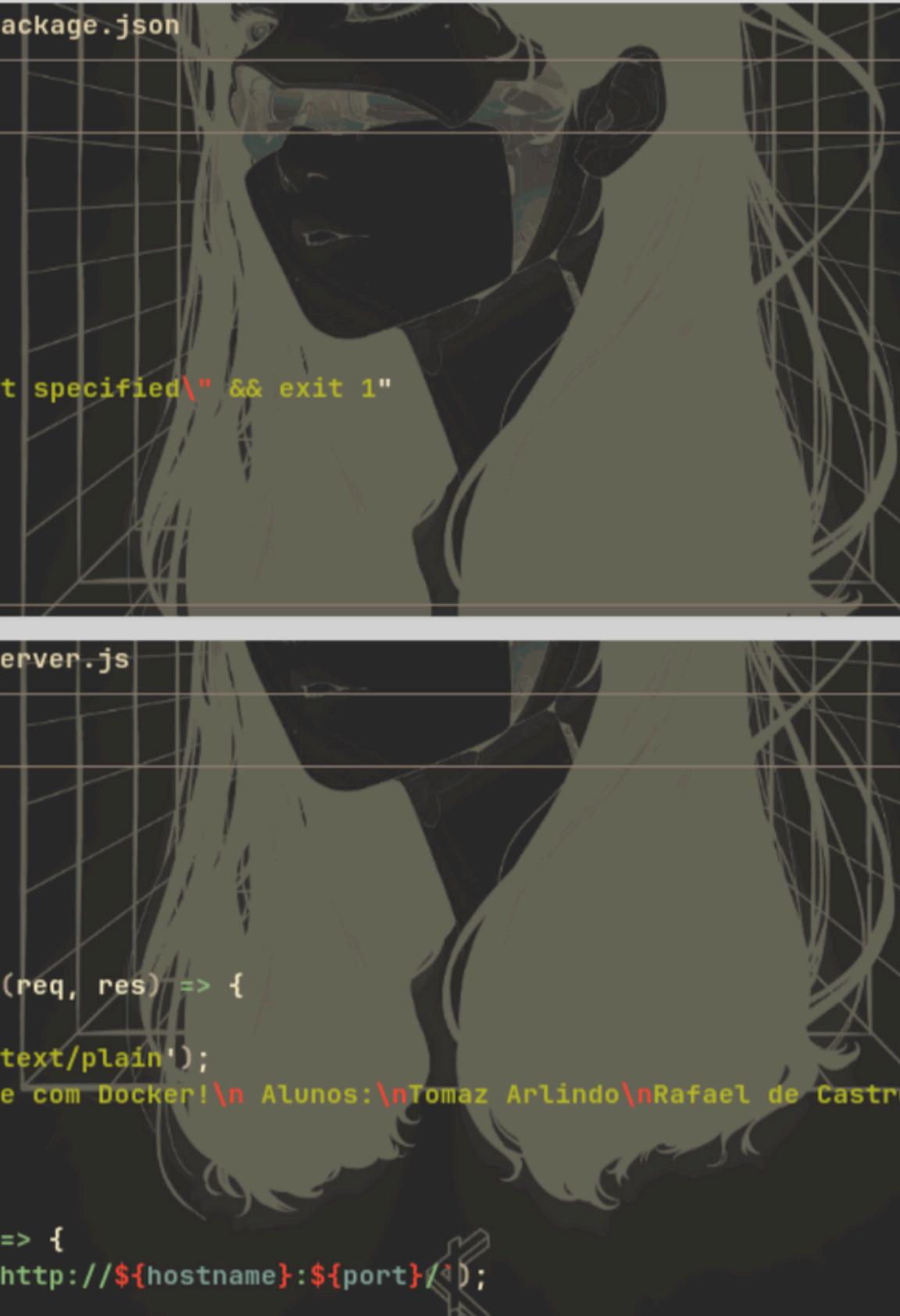
```
File: package.json
```

```
1  {
2    "name": "my-node-app",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC"
12 }
```

```
tomaz@debian:~/projetos/my-node-app$ cat server.js
```

```
File: server.js
```

```
1  const http = require('http');
2
3  const hostname = '0.0.0.0';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Teste de aplicacao node com Docker!\n Alunos:\nTomaz Arlindo\nRafael de Castro\nRai Guilherme\nPaulo Mateus\nLucas Aires\n');
10   });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });


```

Dockerfile

```
tomaz@debian:~/projetos/my-node-app$ cat Dockerfile
```

```
File: Dockerfile
```

```
1 # Usando a imagem oficial do Node.js como base
2 FROM node:14
3
4 # Defina o diretório de trabalho dentro do container
5 WORKDIR /usr/src/app
6
7 # Copie o package.json e package-lock.json para o diretório de trabalho
8 COPY package*.json .
9
10 # Instale as dependências do projeto
11 RUN npm install
12
13 # Copie o código da aplicação
14 COPY .
15
16 # Exponha a porta 3000
17 EXPOSE 3000
18
19 # Comando para rodar a aplicação
20 CMD ["node", "server.js"]
```

Criação da imagem Docker

```
tomaz@debian:~/projetos/my-node-app$ sudo docker build -t my-node-app .
Sending build context to Docker daemon 4.096kB
Step 1/7 : FROM node:14
--> 1d12470fa662
Step 2/7 : WORKDIR /usr/src/app
--> Running in 811f7a758eb1
Removing intermediate container 811f7a758eb1
--> c394e4165745
Step 3/7 : COPY package*.json .
--> d78f0154d7f6
Step 4/7 : RUN npm install
--> Running in 1b06d7cf05ff
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN my-node-app@1.0.0 No description
npm WARN my-node-app@1.0.0 No repository field.

up to date in 0.534s
found 0 vulnerabilities

Removing intermediate container 1b06d7cf05ff
--> f577eefa866a
Step 5/7 : COPY . .
--> 18fea5955d96
Step 6/7 : EXPOSE 3000
--> Running in afa48e87b5dc
Removing intermediate container afa48e87b5dc
--> bbf8996c4966
Step 7/7 : CMD ["node", "server.js"]
--> Running in b68b248ff082
Removing intermediate container b68b248ff082
--> 1356ed58afbd
Successfully built 1356ed58afbd
Successfully tagged my-node-app:latest
tomaz@debian:~/projetos/my-node-app$
```

Verificação de imagens e operando containers

```
tomaz@debian:~/projetos/my-node-app$ sudo docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
my-node-app    latest    1356ed58afbd  About a minute ago  912MB
node            14       1d12470fa662  17 months ago   912MB
tomaz@debian:~/projetos/my-node-app$ sudo docker run -p 3000:3000 -d my-node-app
6051725c02b3f0644d7976944b5a28fc355d20bcccc9dbc45be3264bf5c7842f
tomaz@debian:~/projetos/my-node-app$ sudo docker ps --all
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
6051725c02b3    my-node-app "docker-entrypoint.s..."  14 seconds ago      Up 13 seconds      0.0.0.0:3000->3000/tcp, :::3000->3000/tcp      adoring_curl
ran
tomaz@debian:~/projetos/my-node-app$ 
```

Aplicação do Node.js rodando



Conclusão

A containerização de aplicações **Node.js** com **Docker** é uma abordagem que transforma significativamente o desenvolvimento e a implantação de sistemas modernos. **Docker** permite encapsular a aplicação e suas dependências em containers, garantindo **portabilidade**, **consistência** e **escalabilidade**. Segundo *P. Mell e T. Grance (2011)*, essa técnica facilita o desenvolvimento em múltiplos ambientes, sem a necessidade de reconfigurações constantes, promovendo eficiência em diversos sistemas.

Além disso, a capacidade de isolamento dos containers aumenta a **segurança** e a **confiabilidade** das aplicações em produção, conforme discutido por *J. Merkel (2014)*. A adoção de containers em processos de DevOps permite que os desenvolvedores mantenham **ambientes de desenvolvimento, teste e produção idênticos**, como observado por *L. Bass, I. Weber e L. Zhu (2015)*, minimizando problemas causados por diferenças entre esses ambientes.

A integração entre **Node.js** e **Docker** não só otimiza o desenvolvimento backend, como também torna os processos de entrega contínua mais ágeis e eficientes, conforme discutido ao longo deste seminário. O uso dessa tecnologia é cada vez mais comum em empresas que buscam **flexibilidade** e **controle** sobre suas infraestruturas de produção, demonstrando que **Docker** é uma ferramenta indispensável no cenário de DevOps e no ciclo de vida do software.

Referências

P. Mell, T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, vol. 145, 2011.

J. Merkel, "Docker: lightweight linux containers for consistent development and deployment," Linux Journal, vol. 239, 2014.

L. Bass, I. Weber, L. Zhu, "DevOps: A software architect's perspective," Addison-Wesley Professional, 2015.

Equipe



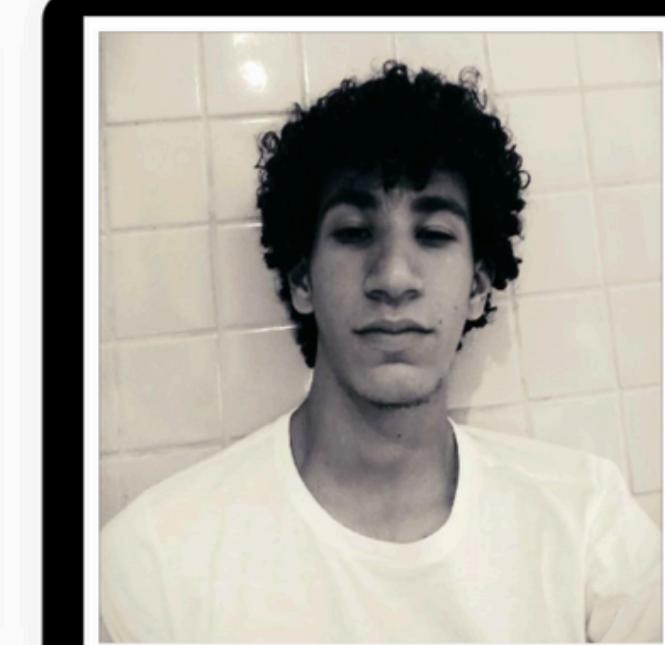
Rafael de Castro

Scrum Master



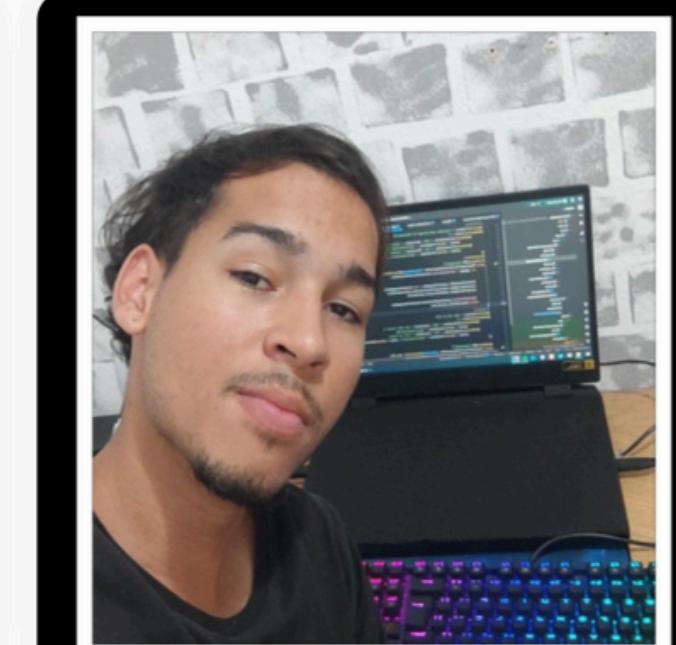
Lucas Aires

Gerente de Configuração



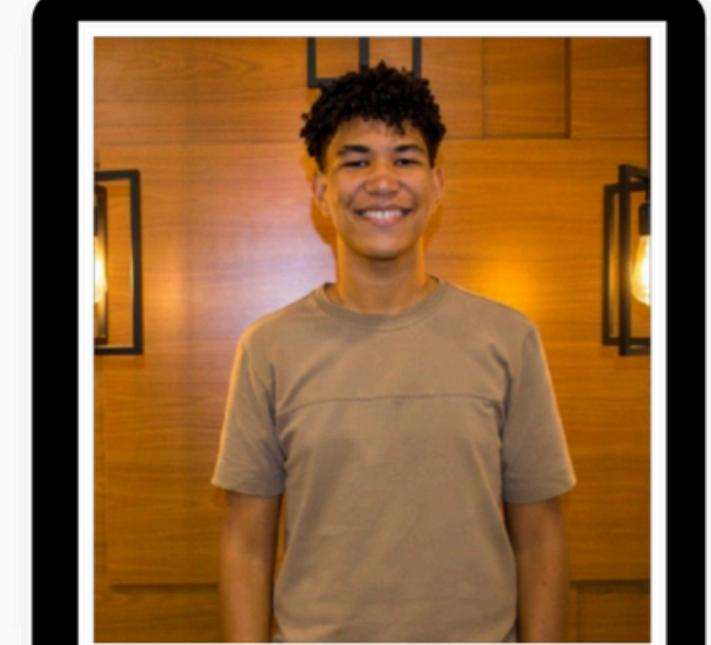
Tomaz Arlindo

Analista Desenvolvedor



Raí Guilherme

Analista Desenvolvedor



Paulo Matheus

Analista de Dados e Negócio