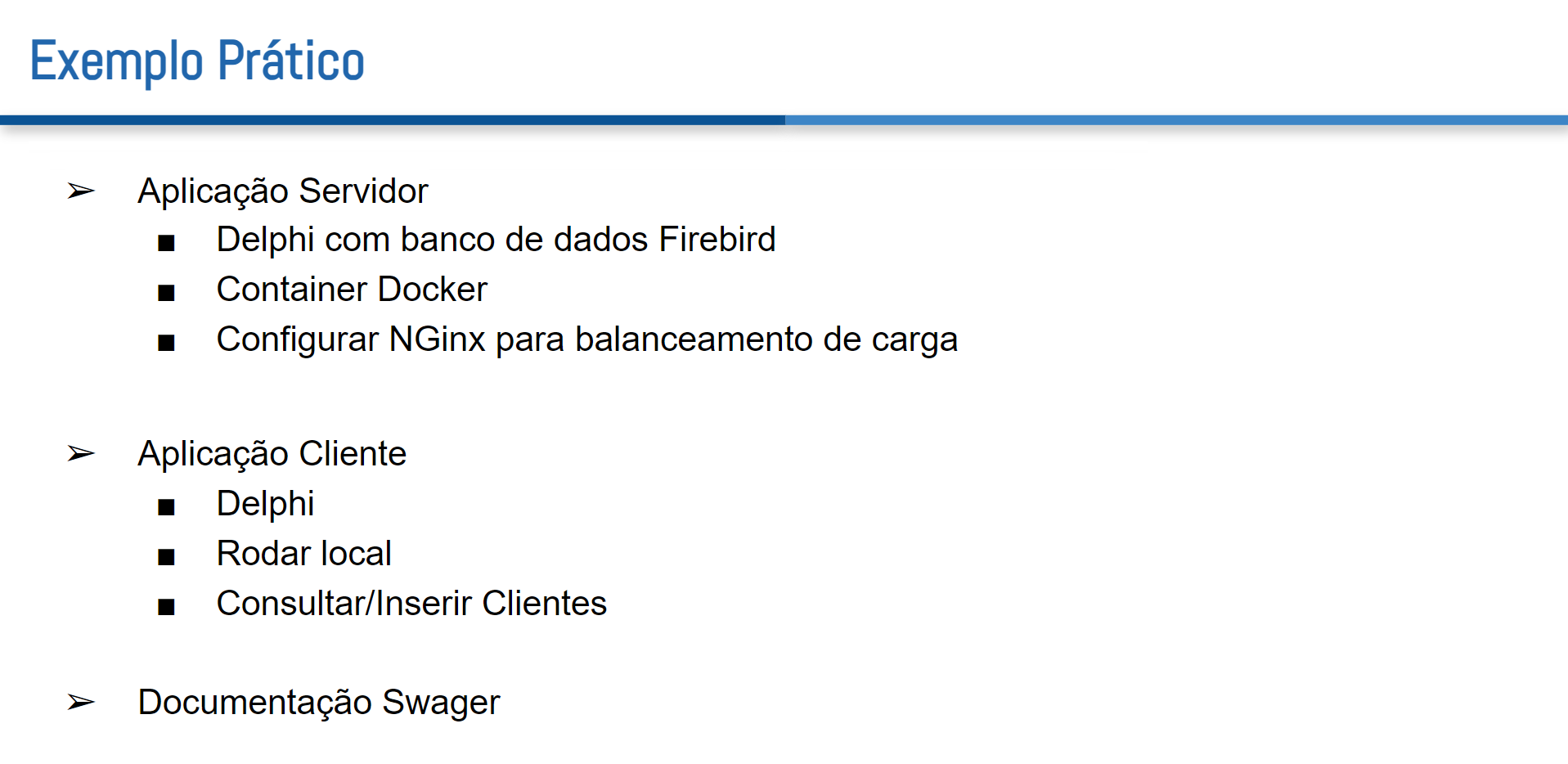
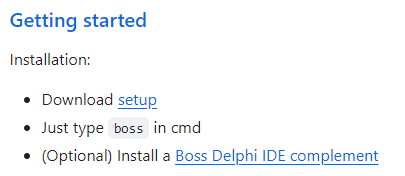
**Micro4Delphi**

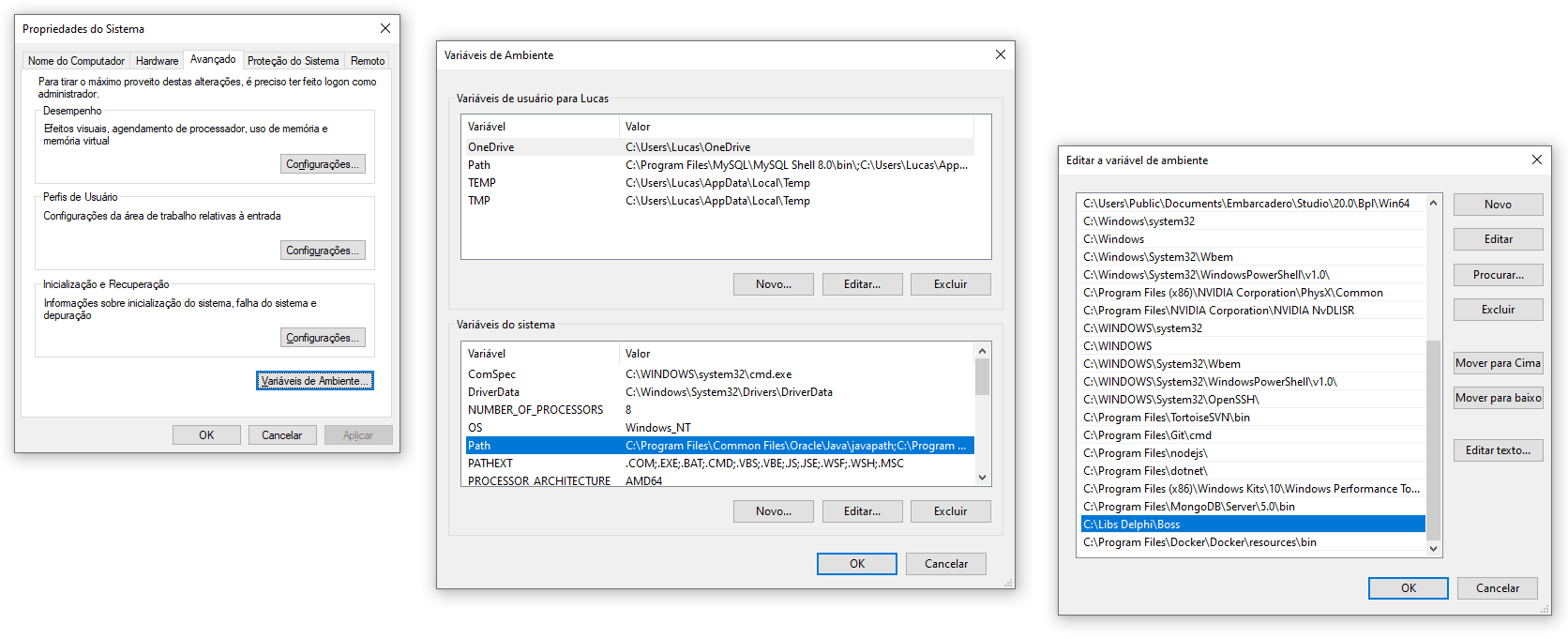
****

****

**Instalando o Boss**

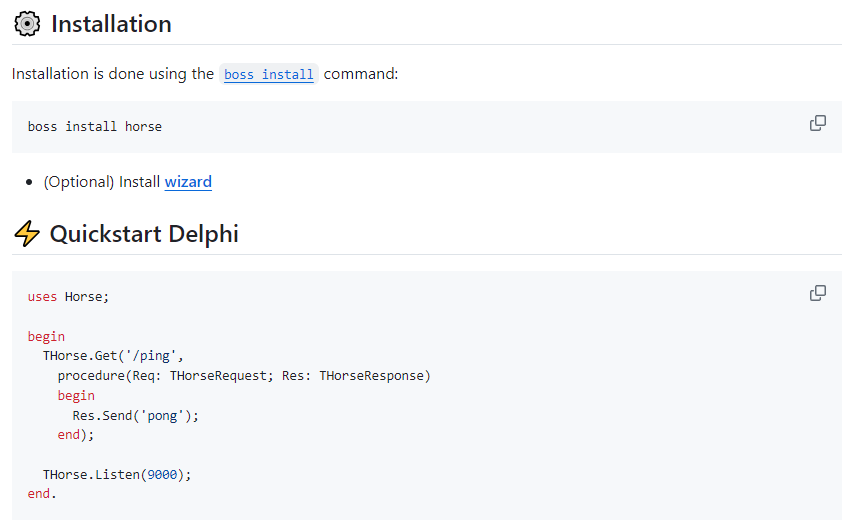
<https://github.com/HashLoad/boss>



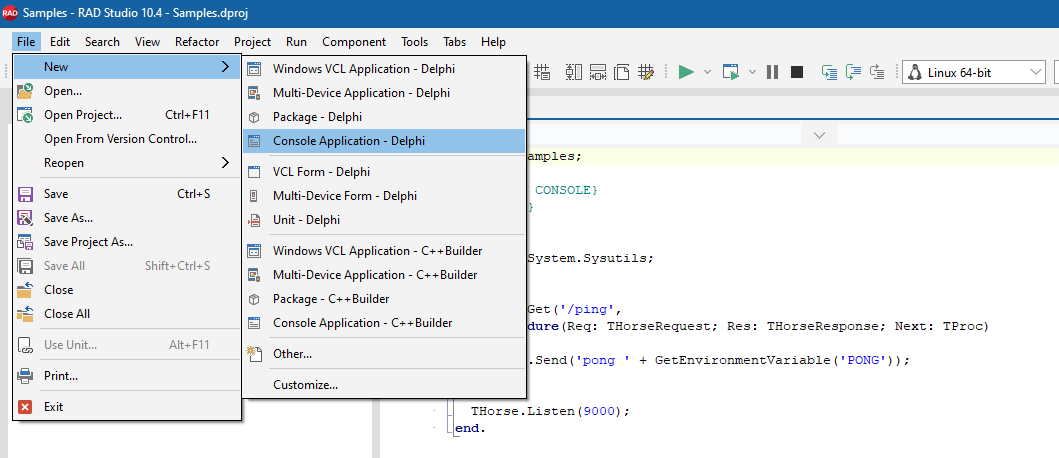


**Instalando o Horse**

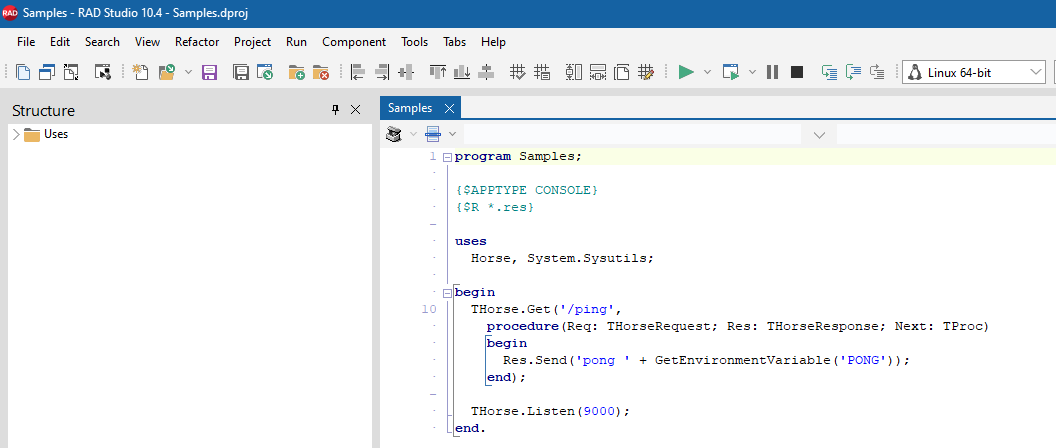
<https://github.com/HashLoad/horse>

****

No Delphi criar um novo projeto como “Console Application - Delphi”.



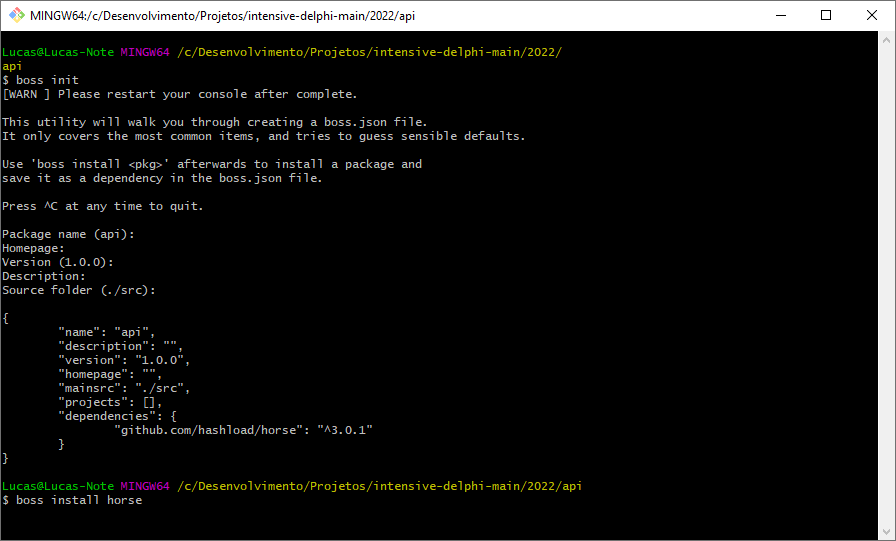
Colar os comandos do Horse.



Salvar o Projeto e abrir o console na pasta do projeto para executar os comandos do Boss.

Primeiro executar “boss init” e confirmar as informações.

Depois executar “boss install horse” para realizar a instalação do horse.



Executar a aplicação do Delphi e permitir acesso.

Testar a aplicação em “localhost:9000/ping”.

Vai retornar “Pong”.

**Compilando a aplicação Delphi para Linux**

Clicar com o botão direito do mouse em “Add Platform”.

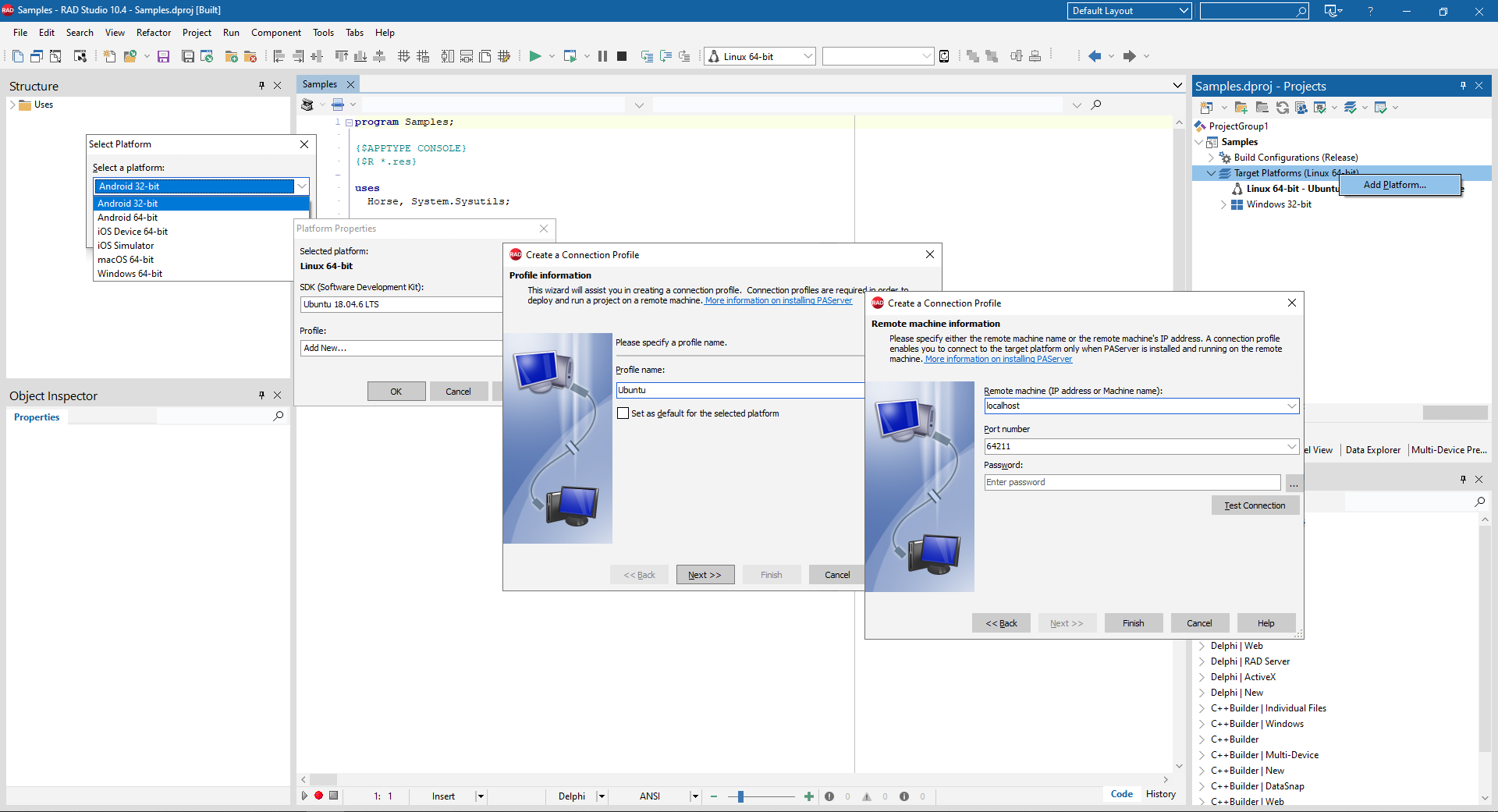
Selecionar “Linux 64-bit”.

Em Profile clicar em “Add New”.

Definir um Profile name “Ubuntu”.

Em Remote informar “localhost”.

Clicar em “Test Connection”.



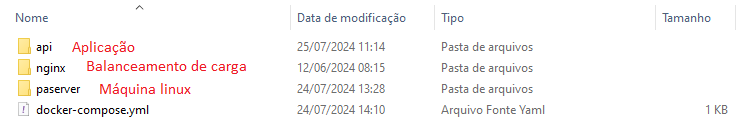
**Instalando o Docker**

[**https://www.docker.com/products/docker-desktop/**](https://www.docker.com/products/docker-desktop/)

****

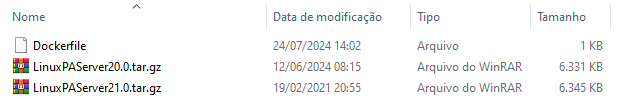
**Configurando a máquina Linux no Docker**

Na pasta onde o Projeto foi salvo, precisamos ter a seguinte estrutura.

****

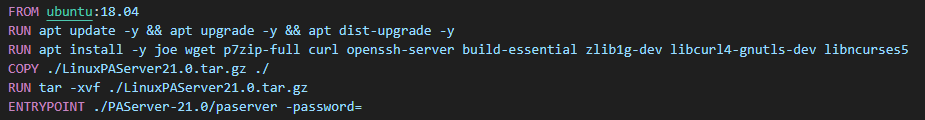
Acessar a pasta “paserver”, onde vamos criar o arquivo “Dockerfile” e inserir o arquivo “LinuxPAServer21.0.tar.gz” de acordo com a versão do Delphi.

Copiar em “C:\Program Files (x86)\Embarcadero\Studio\21.0\PAServer”.

****

[**https://docwiki.embarcadero.com/RADStudio/Sydney/en/Linux\_Application\_Development**](https://docwiki.embarcadero.com/RADStudio/Sydney/en/Linux_Application_Development)

Analisar o “Dockerfile”, nossa receita com base na documentação da Embarcadero.



# Criar uma máquina virtual ubuntu

# Rodar os comandos que estão na documentação

# Aceitar todas as perguntas que são feitas durante a instalação (-y)

# Executar os comandos que a Embarcadero disponibiliza

# Copiar o arquivo “LinuxPAServer21.0.tar.gz” para dentro da nossa imagem

Abrir um console e executar:

Primeiro “docker build -t micro4delphi/paserver .” para criar uma imagem da aplicação no Docker.

Depois “docker run -p 64211:64211 -d micro4delphi/paserver” para executar a aplicação no Docker.

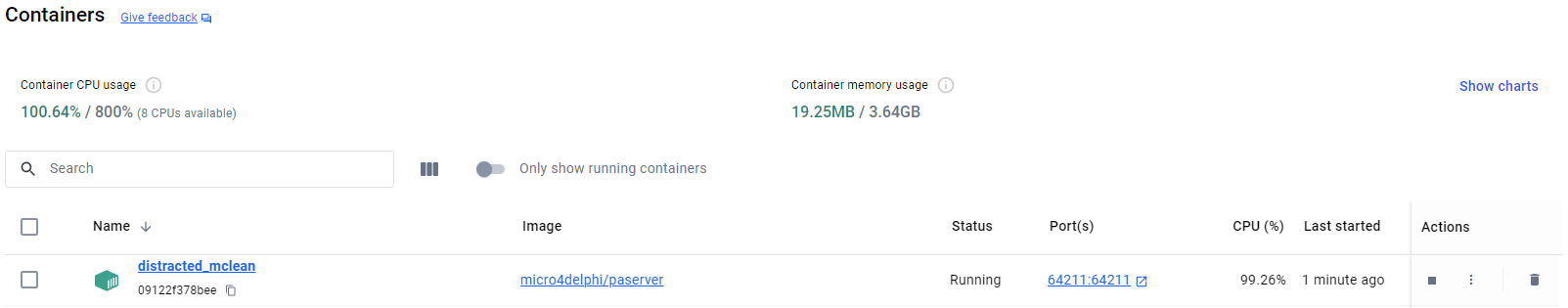
Comandos adicionais:

Executar “docker ps” para visualizar o container.

Voltar para o Delphi e testar a conexão.

Compilar a aplicação para baixar para o Windows todo o conteúdo que está na imagem do Linux.

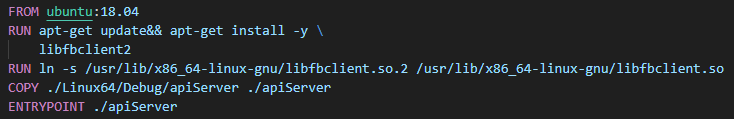
Com isso vamos ter o nosso “paserver” no Docker como imagem e rodando como Container.



**Subindo a aplicação Delphi no Docker como API**

Acessar a pasta “API”, onde vamos criar um arquivo “Dockerfile” e configurá-lo parar criar uma imagem da aplicação no Docker e executá-la.

Analisar o “Dockerfile”.

****

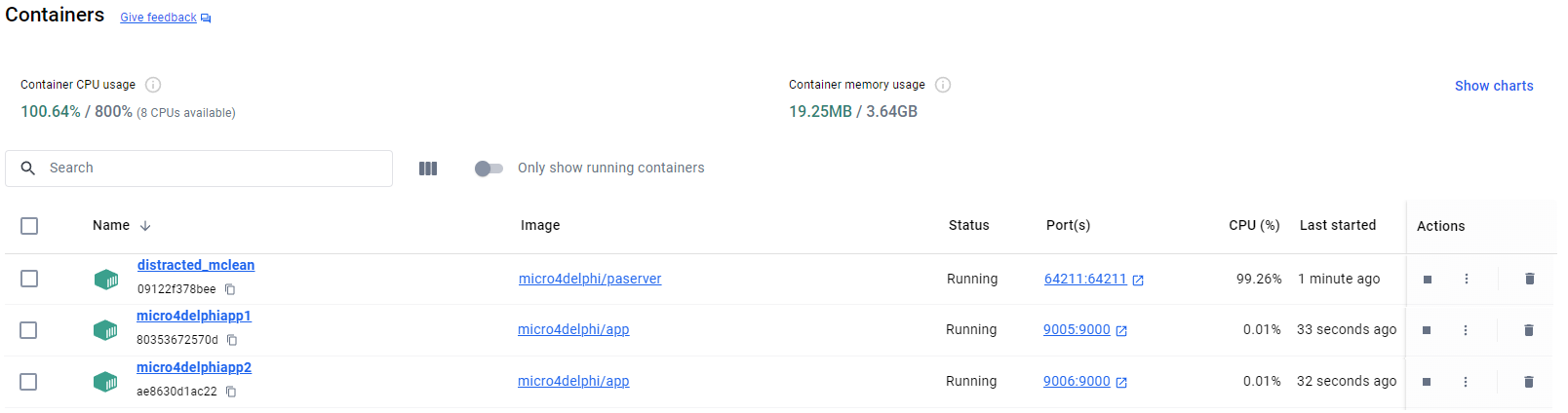
Abrir um console e executar:

Primeiro “docker build -t micro4delphi/app .” para criar uma imagem da API no Docker.

Depois “docker run -d -p 9005:9000 --name micro4delphiapp1 -e PONG=1 --restart unless-stopped micro4delphi/app” para executar a API no Docker.

Depois “docker run -d -p 9006:9000 --name micro4delphiapp2 -e PONG=2 --restart unless-stopped micro4delphi/app” para executar outra instância da API no Docker.

Com isso vamos ter, além do nosso “paserver” no Docker rodando como Container, também duas APIs da nossa aplicação rodando como Container.



Para testar podemos acessar:

<http://localhost:9005/ping> : para API1, que vai retornar “pong1”.

<http://localhost:9006/ping> : para API2, que vai retornar “pong2”.

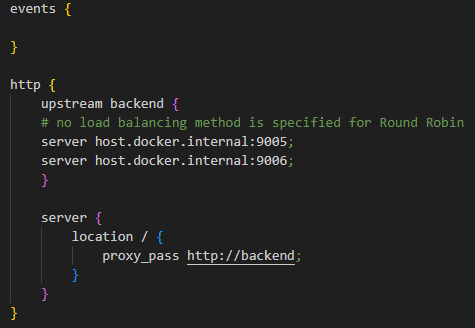
**Configurar o balanceamento de carga**

Acessar a pasta “nginx”, onde vamos criar um arquivo “Dockerfile” e configurá-lo parar criar uma imagem do “nginx” no Docker e executá-la. Essa imagem é padrão do Docker Hub.

Analisar o “Dockerfile”.



Analisar o arquivo “nginx.conf”.

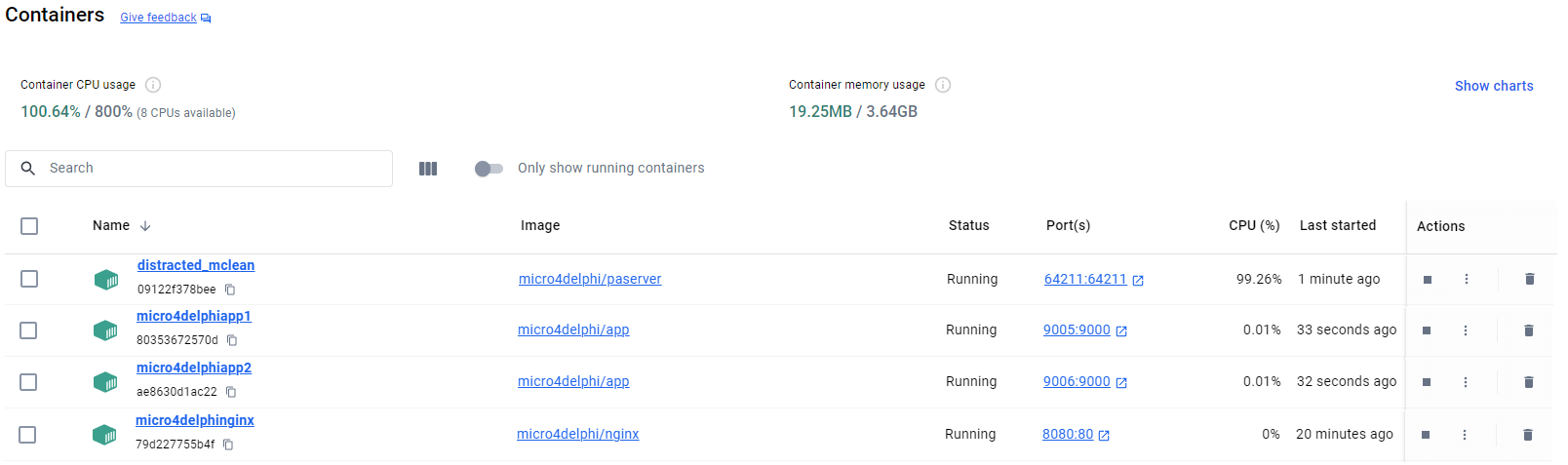


Abrir um console e executar:

Primeiro “docker build -t micro4delphi/nginx .” para criar uma imagem d “nginx” no Docker.

Depois “docker run -d -p 8080:80 --name nginx micro4delphi/nginx” para executar o “nginx” no Docker.

Com isso vamos ter, além do nosso “paserver” e das nossas “APIs” no Docker rodando como Container, também o “nginx” da nossa aplicação para balanceamento de carga rodando como Container.



Para testar podemos acessar:

<http://localhost:9005/ping> : para API1, que vai retornar “pong1”.

<http://localhost:9006/ping> : para API2, que vai retornar “pong2”.

<http://localhost:8080/ping> : para ambas as APIs, que vai retornar hora “pong1” e hora “pong2” comprovando que o balanceamento de carga está funcionando.

Para testar ainda mais podemos:

Parar a API1 no Docker, utilizando o comando “docker stop + ID do Container”.

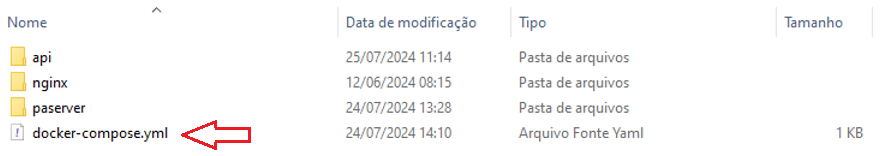
Executar <http://localhost:8080/ping> : que vai retornar sempre “pong2”, pois a API1 está parada.

Para executar novamente a API no Docker, basta utilizar o comando “docker start + ID do Container ”.

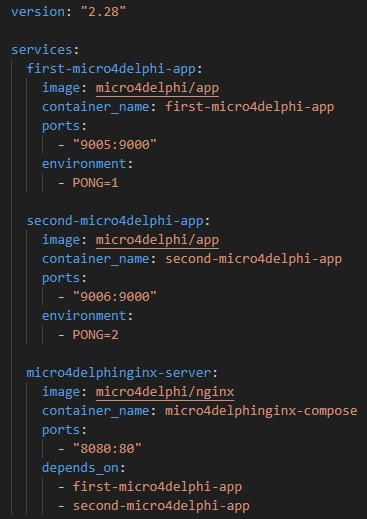
Executar novamente <http://localhost:8080/ping> : que vai retornar hora “pong1” e hora “pong2” comprovando que o balanceamento de carga do “nginx” está funcionando, pois ele já identificou que a API1 voltou a funcionar.

**Configurar o docker-compose**

Acessar a pasta do projeto, onde vamos criar um arquivo “docker-compose.yml” e configurá-lo parar criar todas as imagens de uma única vez e executar as aplicações no Docker.



Analisar o “Docker-compose.yml”.



O “docker-compose.yml” é uma receita de bolo aprimorada.

Aqui está configurado que é para subir duas “APIs”, cada uma com seu nome e porta definidos e o “nginx” com a porta que vai acessar as duas “APIs” para fazer o balanceamento de carga.

Vamos para os Containers.

Abrir um console e executar “docker-compose up -d” para criar os Containers no Docker.

