**FIAP – SHIFT - MICROSERVIÇOS**

Este documento contempla os Laboratórios do curso.

Os laboratórios de docker podem ser feitos pelo site <http://play-with-docker.com>

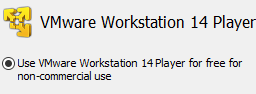
# LAB : Preparação

1. Prepararando LAB FIAP

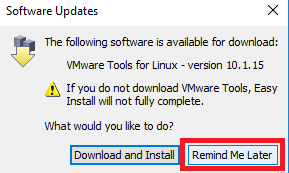
* Abrir máquina virtual



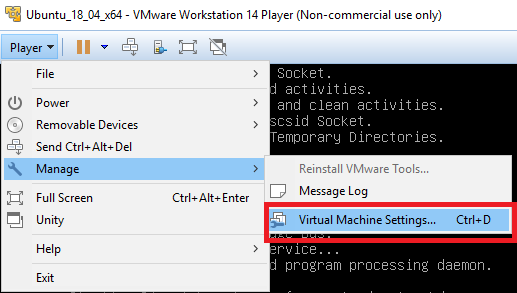
* Clicar em Continuar



* Clicar em Remind me Later



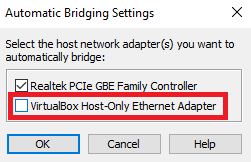
* Quando a máquina estiver iniciando, **apertar CTRL+D** para abrir VM Setting



* Clicar em Configure Adapters



* Desmarcar placa VirtualBox



* Acessar a VM

> login: ubuntu / pass: fiap

* Reload da rede e reiniciar a máquina virtual:

> sudo /etc/init.d/networking reload

> sudo init 6

1. Iniciada a VM

* Anotar o IP do seu servidor

> ifconfig

* Acessar a VM a partir do [Putty](https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe)

> login: ubuntu / pass: fiap

* Testar acesso a internet a partir do laboratório

> curl <http://google.com>

Resposta em caso de **SUCESSO**:

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">

<TITLE>301 Moved</TITLE></HEAD><BODY>

<H1>301 Moved</H1>

**The document has moved**

<A HREF="http://www.google.com/">here</A>.

</BODY></HTML>

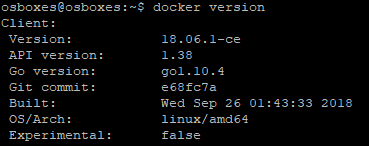
Resposta em caso de **ERRO**:

<html><head><title>**Firewall Authentication**</title></head><body>**Redirected to the secure channel**.<a href="http://10.20.91.254:1000/fgtauth?030263d9d8f90426">Click here</a> to load the secure authentication page.</body></html>

# LAB 1 A: Introdução ao Docker

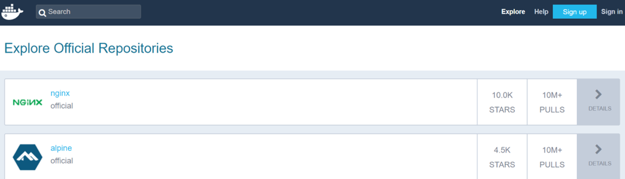
1. Verificar a versão instalada.

> docker version



1. Acessar o Docker Hub e verificar algumas imagens disponíveis para utilização.

> <https://hub.docker.com/>explore/



1. Instalar a imagem do servidor HTTP (NGINX) para hospedagem de página estática.

> docker pull nginx

> docker images

1. Executar o contêiner com NGINX padrão e verificar a página de teste.

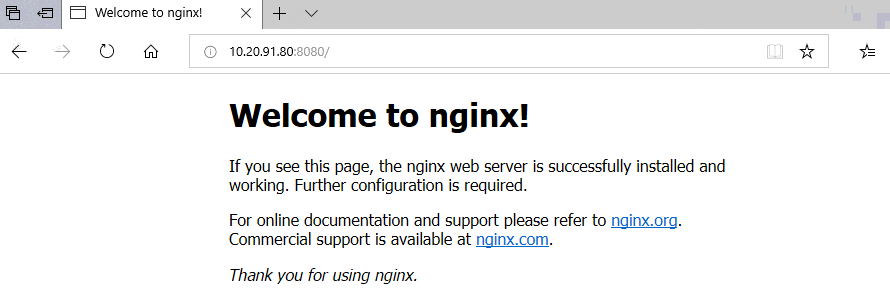
> docker run --name meu-nginx -p 8090:80 -d nginx

1. Verificar localmente se a página de teste está funcionando

> curl <http://localhost:8090/>

1. Acessar a página de teste pelo navegador do Windows

> Acessar: <http://ip:8090/>



1. Verificar em qual porta o container está executando:

> docker port meu-nginx

1. Listar contêineres em execução:

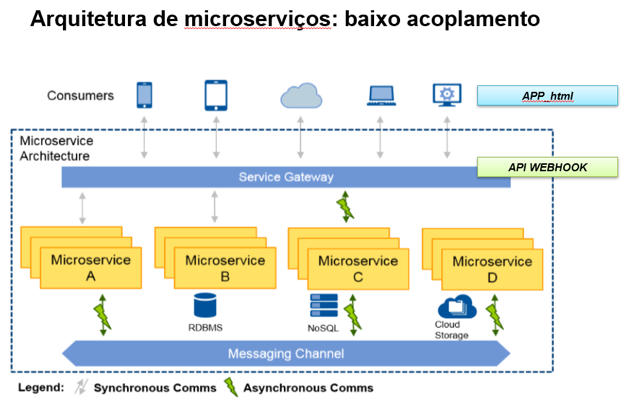
> docker ps

# LAB 1 B

Docker permite a utilização de imagens prontas de ambientes, sem a necessidade de instalar sistema operacional e todas as dependências da aplicação.

A imagem portátil do ambiente é definida no Dockerfile, garantindo com que as configurações encapsuladas no contêiner façam com que a aplicação seja executada da mesma forma independente do ambiente.

Uma imagem criada pode ser disponibilizar no Docker Hub para facilitar o deploy



1. Criando nosso APP (página estática que vai chamar uma API):

* Acessar<http://shiftfiap.slack.com> ( link para realizar cadastro caso ainda não tenha feito na aula anterior: https://bit.ly/slackshift )
* WebHook já foi criado anteriormente

> Webhook URL:

https://hooks.slack.com/services/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

* Criar um diretorio (ex: “slackpage\_nginx”)

> mkdir slackpage\_nginx

> cd slackpage\_nginx

* Inserir uma pagina estatica com o código do arquivo index.html:

> nano index.html

<html>

<head>

<script type="text/javascript" src="http://code.jquery.com/jquery-1.5.2.js"></script>

</head>

<body>

<h1>Envio de mensagem : Slack webhook test</h1> <br>

<form id="slackform">

Channel : <input id="target" value="#fiap"/>

<br/>

Messsage : <input id="msgInput"/>

<br/> <br>

<input type="button" value="enviar" onclick="postMSG()"/>

</form>

<div id='result'></div>

<script type="text/javascript">

**// Definir a função postMSG(), conferir Canal e Webhook:**

function postMSG(){

// get message to be posted

var msg = document.getElementById('msgInput').value

// get target channel or team member

var chan = document.getElementById('target').value

// format payload for slack

var sdata = formatForSlack(msg, chan)

// log in console

console.log(sdata)

// post

$.ajax({

// url is what you get from activating the "Incoming WebHooks" slack integration

// if you leave, you should see an error message "No Team", status 404

url: 'https://hooks.slack.com/services/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',

type: 'POST',

processData: true,

data : sdata ,

success : function(data) {

// success will show on page

console.log(data)

$('#result').html(data);

},

error: function(data){

// error will show error object

console.log(data)

$('#result').html("error:"+JSON.stringify(data));

}

});

}

</script>

<script type="text/javascript">

**// Definir a função formatForSlack() que ajusta os dados:**

*function* *formatForSlack(msg, chan){*

*var payload ={*

*"channel":chan,*

*"username":'app\_html',*

*"text": msg,*

*// https://www.webfx.com/tools/emoji-cheat-sheet/*

*"icon\_emoji":':ghost:'*

*};*

*// return json string of payload*

*return JSON.stringify(payload)*

*}*

*</script>*

1. Definindo nosso Dockerfile

* Criar Dockerfile da imagem nginx:alpine

> nano Dockerfile

* Inserir o seguinte conteúdo:

FROM nginx:alpine

COPY index.html /usr/share/nginx/html

* Criar uma imagem do contêiner

> docker build -t imagem\_nginx\_alpine\_slackpage .

*OBS: O ponto no final é o PATH do diretório que contém o Dockerfile*

1. Executar o contêiner

* Comando para executar o contêiner:

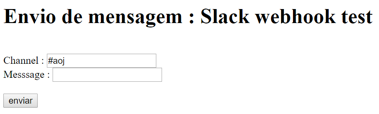
> docker run --name slackpage\_nginx -it -p 8020:80 -d imagem\_nginx\_alpine\_slackpage

* Verificar localmente se a página está funcionando

> curl <http://localhost:8020/>

* Acessar a página pelo navegador.

> Acessar: <http://ip:8020/>



* Verificação da execução da chamada API através da APP\_HTML



1. Utilizando um ambiente de Desenvolvimento

* Executar o Docker com a opção -v para mapear uma pasta do *node* para dentro do *container*.
* Caso de uso: criar uma pasta DEV e colocar uma versão modificada do arquivo index.html

> mkdir ~/slackpage\_nginx/dev

> cp ~/slackpage\_nginx/index.html ~/slackpage\_nginx/dev/

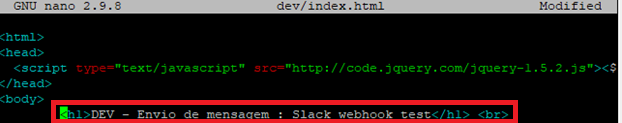
> docker run --name slack\_nginx\_2 -it -p 8021:80 -v ~/slackpage\_nginx/dev/:/usr/share/nginx/html/dev -d imagem\_nginx\_alpine\_slackpage

* Acessar a página pelo navegador.

> Acessar: <http://ip:8021/>dev/

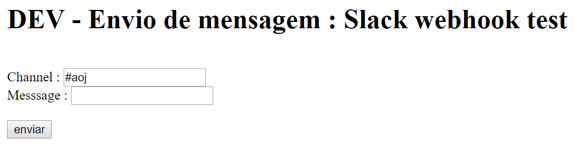
* Alterar arquivo local

> alterar a página dev/index.html e testar se aparece a nova versão



* Acessar nova versão da página pelo navegador.

> Acessar: <http://ip:8021/>dev/



# LAB 1 C

Demonstrando a portabilidade da imagem que foi criada:

1. Login no Docker Hub:

* Acessar a página <http://hub.docker.com> e criar username/password
* Fazer o login no docker (hub)

> docker logout

> docker login

1. Registrar e publicar a imagem criada no Docker Hub

Utilizaremos o registro do Docker Hub, porém poderia ser utilizado outro, que tem como papel funcionar como um repositório de imagens.

Definir o TAG da imagem (utilizado no versionamento)

O registro da imagem é feito com base na notação: USER/REPOSITORIO:TAG

* Registrar a imagem criada no Docker Hub

> docker tag imagem\_nginx\_alpine\_slackpage:latest *usuario*/fiap\_slackpage:latest

* Verificar TAG da imagem

> docker image ls

* Publicar a imagem criada no Docker Hub

> docker push usuario/fiap\_slackpage:latest

1. Executar a imagem em outro computador diretamente do repositório na nuvem (hub.docker)

* Execute o container disponibilizado pela equipe vizinha

> docker run --name vizinho\_docker -ti -p 8025:80 -d usuario\_vizinho/fiap\_slackpage:latest

* Acessar nova página criada a partir do container publicado do Vizinho.

> Acessar: <http://ip:8025/>

# LAB 2

1. Montando nossa solução baseada em API REST:

* Criar um diretório e definir um ambiente virtual (virtualenv) dentro do diretório:

> cd ~

> mkdir alunos\_microservice

> cd alunos\_microservice

> python3 -m venv alunos\_venv

* Instalação do Python (pip/venv), **caso necessário**.

> sudo add-apt-repository universe

> sudo apt-get install python3-pip

> sudo apt-get install python3-venv

* Esse comando cria um diretório chamado **alunos\_venv**, que contém arquivos isolados do nosso projeto em uma estrutura de diretório:
  + bin: binário do ambiente virtual
  + include: cabeçalhos dos pacotes Python
  + lib: uma cópia da versão do Python junto com uma pasta onde ficará cada dependência instalada

├── bin

│ ├── activate

│ ├── activate.csh

│ ├── activate.fish

│ ├── easy\_install

│ ├── easy\_install-3.6

│ ├── pip

│ ├── pip3

│ ├── pip3.6

│ ├── python -> python3.6

│ ├── python3 -> python3.6

│ └── python3.5 -> /Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6

├── include

├── lib

│ └── python3.6

│ └── site-packages

└── pyvenv.cfg

* Para usar os pacotes deste ambiente isoladamente, você precisa "ativá-lo". Em seguida, vamos instalar os modulos: *Flask* e o *CORS*

> source alunos\_venv/bin/activate

(alunos\_venv) $ pip3 install flask

(alunos\_venv) $ pip3 install flask-cors

* Criar o arquivo server.py:

> nano server.py

from flask import Flask, render\_template

app = Flask(\_\_name\_\_, template\_folder="templates")

@app.route('/')

def home():

return render\_template('alunos.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000, debug=True)

* Criar a pasta “templates”

(alunos\_venv) $ mkdir templates

(alunos\_venv) $ cd templates

* Dentro dela, criar o arquivo alunos.html

> nano alunos.html

<!DOCTYPE html>

<head>

<meta charset="UTF-8">

<title>Alunos Home Page</title>

</head>

<body>

<h2>

Alunos, sejam bem vindos!

</h2>

</body>

</html>

* Executar o servidor.

(alunos\_venv) $ cd ..

(alunos\_venv) $ python server.py

* Acessar pelo navegador:

> [http://IP:5000](http://ip:3000/)

* Para finalizar a execução do servidor: CTRL + C

1. Configurando um endpoint da API REST com o *Connexion*

* Vamos adicionar um endpoint API REST com o módulo Connexion, o qual é instalado usando o pip:

(alunos\_venv) $ pip3 install connexion

* O módulo Connexion permite que um programa Python use a especificação Swagger. Vamos instalar

(alunos\_venv) $ *pip3 install connexion[swagger-ui]*

* Vamos adicionar o Connexion em nosso server.py:

*> rm server.py*

*> nano server.py*

from flask import render\_template

import connexion

from flask\_cors import CORS

app = connexion.App(\_\_name\_\_, specification\_dir='./')

app.add\_api('swagger.yml')

CORS(app.app,resources=r'/api/\*',methods=['GET', 'POST', 'OPTIONS', 'PUT', 'DELETE'])

@app.route('/')

def home():

return render\_template('alunos.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000, debug=True)

* Vamos criar o arquivo swagger.yml:

*> nano swagger.yml*

swagger: "2.0"

info:

description: Swagger file that goes with our server code

version: "1.0.0"

title: Swagger ReST Alunos

consumes:

- application/json

produces:

- application/json

basePath: /api

paths:

/alunos:

get:

operationId: alunos.read\_all

tags:

- Alunos

summary: Lista de alunos

description: Read Lista de Alunos

parameters:

- name: length

in: query

type: integer

description: Number of Alunos to get from Alunos

required: false

- name: offset

in: query

type: integer

description: Offset from beginning of list where to start gathering Alunos

required: false

responses:

200:

description: Successfully read Alunos list operation

schema:

type: array

items:

properties:

fname:

type: string

lname:

type: string

timestamp:

type: string

* Vamos criar o endpoint alunos.py:

*> nano alunos.py*

from datetime import datetime

from flask import jsonify, make\_response, abort

def get\_timestamp():

return datetime.now().strftime(("%Y-%m-%d %H:%M:%S"))

PEOPLE = {

"Jones": {

"fname": "Indiana",

"lname": "Jones",

"timestamp": get\_timestamp(),

},

" Sparrow": {

"fname": "Jack",

"lname": " Sparrow",

"timestamp": get\_timestamp(),

},

"Snow": {

"fname": "John",

"lname": "Snow",

"timestamp": get\_timestamp(),

},

}

def read\_all():

dict\_alunos = [PEOPLE[key] for key in sorted(PEOPLE.keys())]

alunos = jsonify(dict\_alunos)

qtd = len(dict\_alunos)

content\_range = "alunos 0-"+str(qtd)+"/"+str(qtd)

# Configura headers

alunos.headers['Access-Control-Allow-Origin'] = '\*'

alunos.headers['Access-Control-Expose-Headers'] = 'Content-Range'

alunos.headers['Content-Range'] = content\_range

return alunos

* Executar o servidor.

(alunos\_venv) $ python server.py

* Acessar pelo navegador:

> [http://IP:5000](http://ip:3000/)/api/alunos

* Acessar pelo navegador, permitindo que você explore sua API:

> [http://IP:5000](http://ip:3000/)/api/ui

1. Concluindo o Swagger e demais verbos HTTP

* Atulizar o arquivo: *swagger.yml* , **acrescentando** **os** **paths no final do arquivo**:

*> nano swagger.yml*

post:

operationId: alunos.create

tags:

- Alunos

summary: Create a person and add it to the Alunos list

description: Create a new person in the Alunos list

parameters:

- name: person

in: body

description: Person to create

required: True

schema:

type: object

properties:

fname:

type: string

description: First name of person to create

lname:

type: string

description: Last name of person to create

responses:

201:

description: Successfully created person in list

/alunos/{lname}:

get:

operationId: alunos.read\_one

tags:

- Alunos

summary: Read one person from the Alunos list

description: Read one person from the Alunos list

parameters:

- name: lname

in: path

description: Last name of the person to get from the list

type: string

required: True

responses:

200:

description: Successfully read person from Alunos list operation

schema:

properties:

fname:

type: string

lname:

type: string

timestamp:

type: string

put:

operationId: alunos.update

tags:

- Alunos

summary: Update a person in the Alunos list

description: Update a person in the Alunos list

parameters:

- name: lname

in: path

description: Last name of the person to update in the list

type: string

required: True

- name: person

in: body

schema:

type: object

properties:

fname:

type: string

lname:

type: string

responses:

200:

description: Successfully updated person in Alunos list

delete:

operationId: alunos.delete

tags:

- Alunos

summary: Delete a person from the Alunos list

description: Delete a person

parameters:

- name: lname

in: path

type: string

required: True

responses:

200:

description: Successfully deleted a person from Alunos list

* Atulizar o arquivo: *alunos.py*, acrescentando as demais funções **no final do arquivo**:

*>* *nano alunos.py*

def read\_one(lname):

if lname in PEOPLE:

person = PEOPLE.get(lname)

else:

abort(

404, "Person with last name {lname} not found".format(lname=lname)

)

return person

def create(person):

lname = person.get("lname", None)

fname = person.get("fname", None)

if lname not in PEOPLE and lname is not None:

PEOPLE[lname] = {

"lname": lname,

"fname": fname,

"timestamp": get\_timestamp(),

}

return make\_response(

"{lname} successfully created".format(lname=lname), 201

)

else:

abort(

406,

"Person with last name {lname} already exists".format(lname=lname),

)

def update(lname, person):

if lname in PEOPLE:

PEOPLE[lname]["fname"] = person.get("fname")

PEOPLE[lname]["timestamp"] = get\_timestamp()

return PEOPLE[lname]

else:

abort(

404, "Person with last name {lname} not found".format(lname=lname)

)

def delete(lname):

if lname in PEOPLE:

del PEOPLE[lname]

return make\_response(

"{lname} successfully deleted".format(lname=lname), 200

)

else:

abort(

404, "Person with last name {lname} not found".format(lname=lname)

)

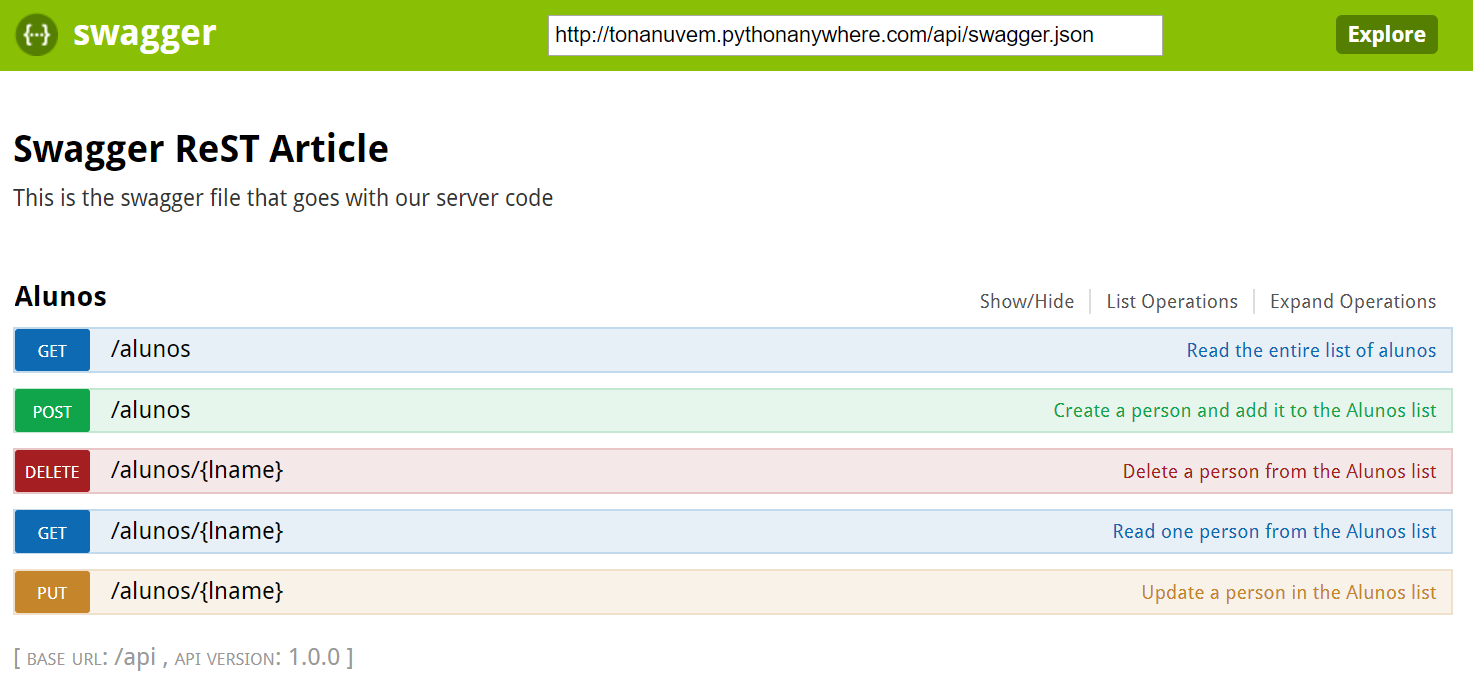
* Executar o servidor.

(alunos\_venv) $ python server.py

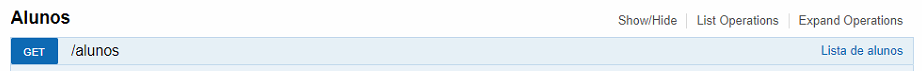
* Acessar pelo navegador, permitindo que você explore sua API:

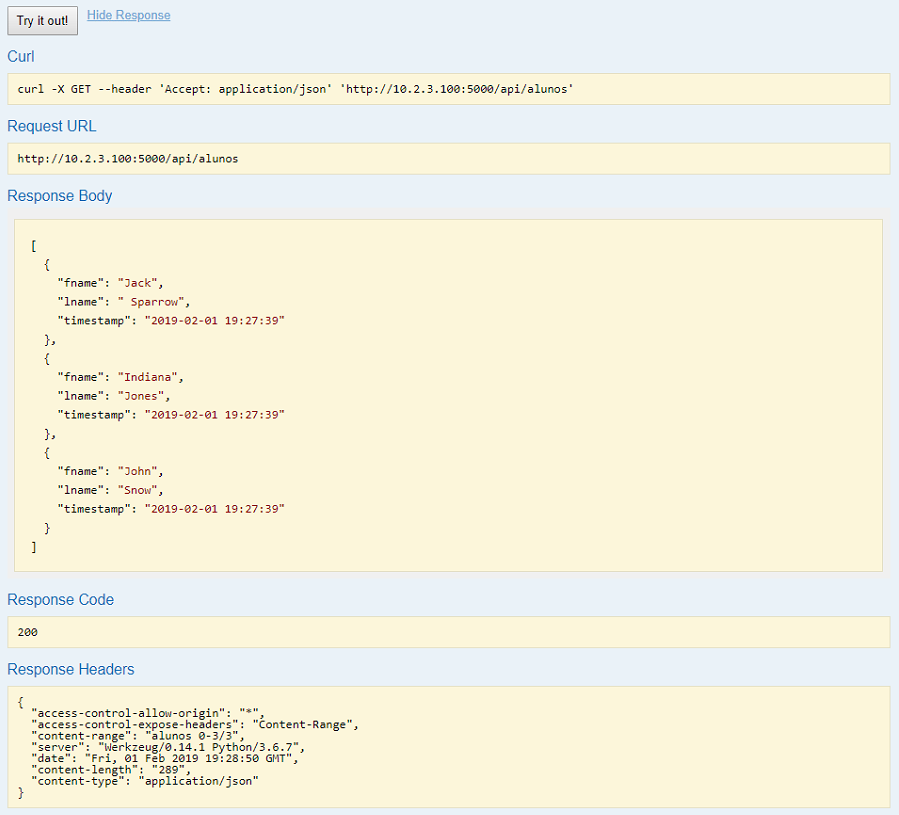
> [http://IP:5000](http://ip:3000/)/api/ui

* A API agora tem outras funcionalidades para realizer operações CRUD, e a interface do Swagger está atualizada com os novos verbos HTTP:

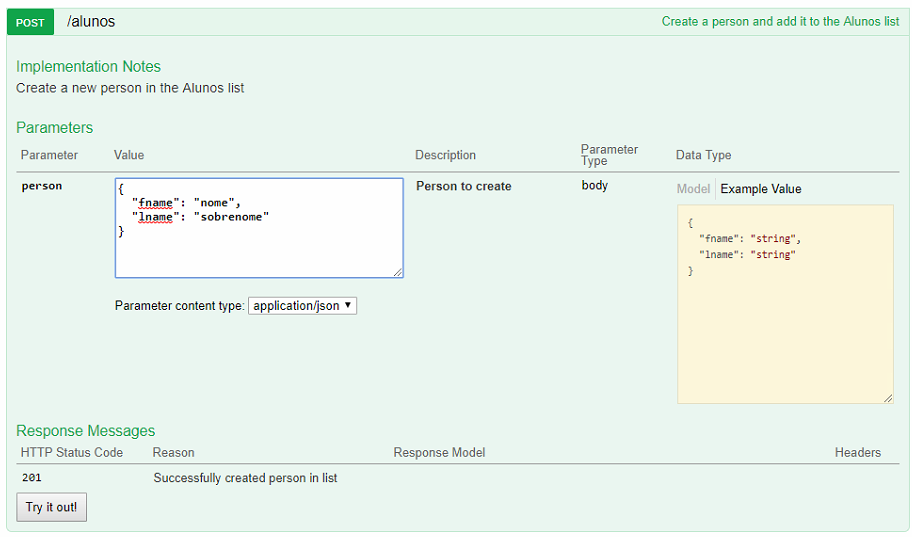


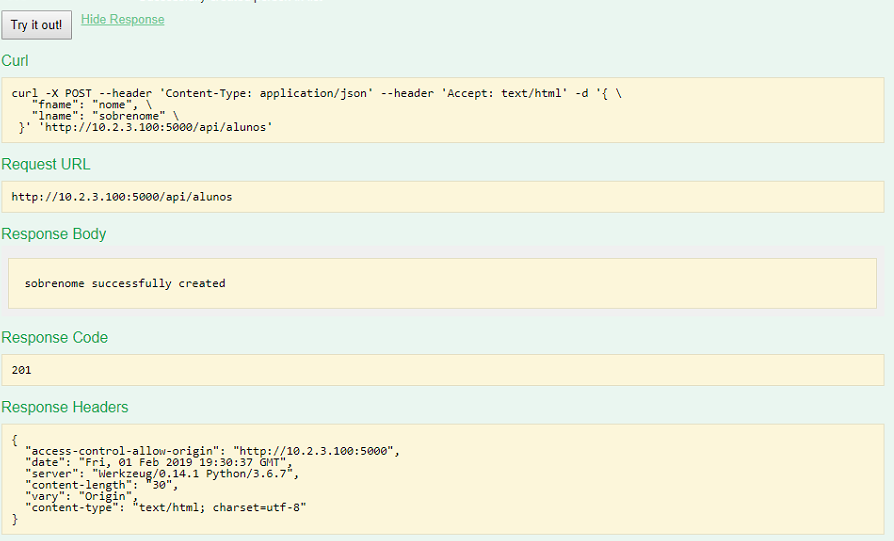
* Para testar o READ, vamos Clicar no botão “Try it out” dentro do GET, conforme telas abaixo:



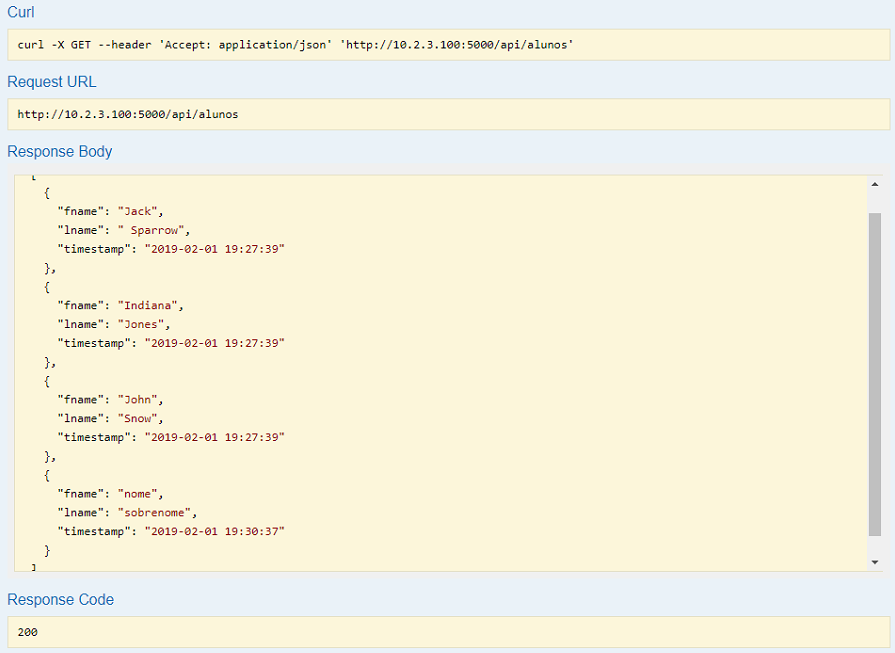


* Para testar o POST, preencha um nome e sobrenome e clique no botão “Try it out” dentro do POST, conforme telas abaixo:





* Finalmente, realize um novo READ para confirmar se o novo valor foi inserido, clicando no botão “Try it out” dentro do GET, conforme telas abaixo:



1. Montando nossa aplicação cliente SPA (Single-Page Application)

* Criar o aplicativo da web Front-end. Tudo isso será tratado por chamadas AJAX do JavaScript para endpoints de URL da API.
* Atualizar o arquivo: *templates/alunos.html*

*> cd ~/alunos\_microservices*

*> rm templates/alunos.html*

*>* *nano templates/alunos.html*

<html lang="pt-br">

<head>

<!-- Meta tags Obrigatórias: https://getbootstrap.com.br/docs/3.3/css/#tables -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

<title>Alunos App</title>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.0/normalize.min.css">

<script src="http://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=" crossorigin="anonymous">

</script>

</head>

<body>

<div class="container">

<h1>Alunos App</h1>

<div class="section editor"> <label for="fname">Primeiro Nome <input id="fname" class="form-control"

type="text"> </label> <br>

<label for="lname">Ultimo Nome <input id="lname" type="text" class="form-control"> </label>

<br>

<button id="create" class="btn btn-primary">Create</button> <button id="reset" class="btn btn-info">Read</button> <button id="update" class="btn btn-warning">Update</button>

<button id="delete" class="btn btn-danger">Delete</button> </div>

<br>

<blockquote>

<p>Clique duas vezes nos nomes abaixo para editar.</p>

</blockquote>

<div class="alunos">

<table class="table table-striped">

<thead>

<tr>

<th>Primeiro Nome</th>

<th>Ultimo Nome</th>

<th>Update Time</th>

</tr>

</thead>

<tbody>

</tbody>

</table>

</div>

<div class="error"> </div>

</div>

<script src="static/js/alunos.js"></script>

<!-- Dependencias Bootstrap: jQuery.js, Popper.js, Bootstrap JS -->

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy" crossorigin="anonymous"></script>

</body>

</html>

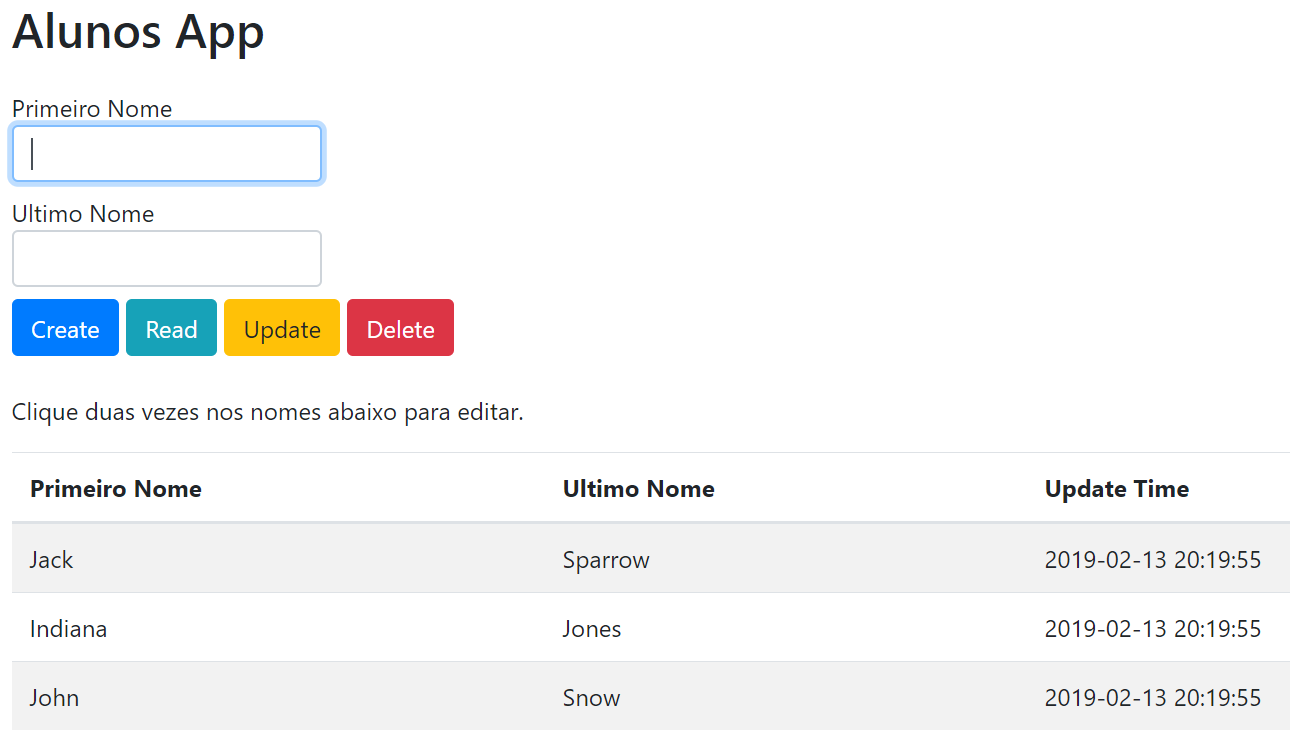
* Criar o arquivo: *static/js/alunos.js*

*>* *mkdir -p* *~/alunos\_microservice/static/js/*

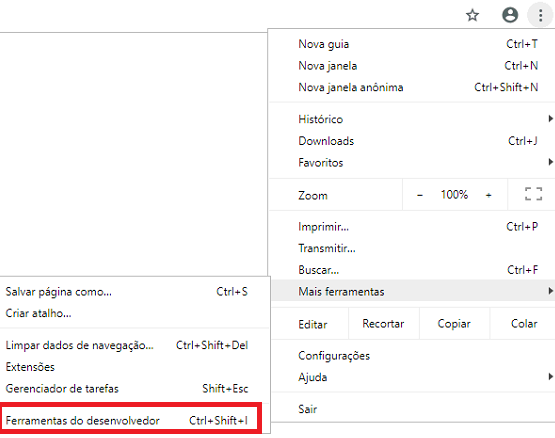
*>* *wget -O* *~/alunos\_microservice/static/js/alunos.js* *https://tonanuvem.net/apps/aoj/alunos.js*

* Analise o conteudo do arquivo JavaScript “alunos.js”. Esse arquivo é responsável pela chamada da API e está dividido em três partes usando o design pattern MVC (Model / View / Controller).
* Acessar pelo navegador e testar a aplicação:

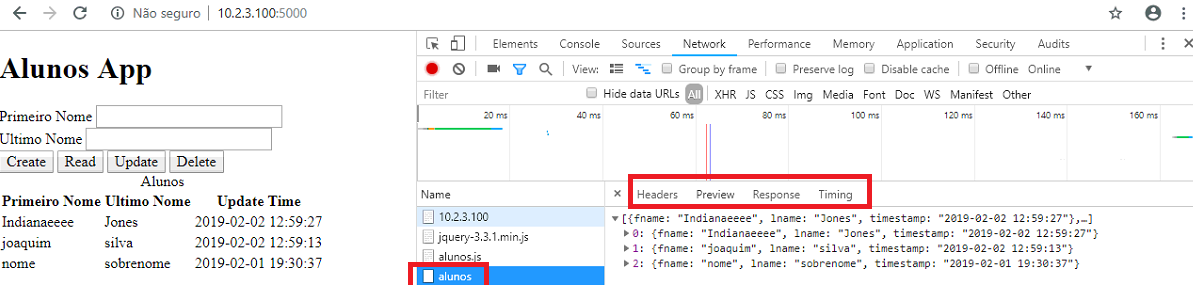
> [http://IP:5000](http://ip:3000/)/



* Abrir a aba de Ferramentas do Desenvolvedor (imagem do Chrome):



* Ao clicar nos botões “CRUD”, verifique no endpoint alunos : Headers; Preview/Response e o Tempo de Resposta (Timing):



# LAB 2 – BONUS

1. Criar um Dockerfile para construção automática da imagem:

* Gerar arquivo de dependencias dos pacotes que utilizamos (requirements.txt)

(alunos\_venv) $ touch requirements.txt

(alunos\_venv) $ pip3 freeze > requirements.txt

(alunos\_venv) $ nano requirements.txt

🡪 remover linha (bug): pkg-resources==0.0.0

🡪 ajustar p/ ultima versão: PyYAML==5.1.1

* Criar Dockerfile

> nano Dockerfile

* Inserir o seguinte conteúdo:

FROM python:3.6-alpine

ADD . /code

WORKDIR /code

RUN pip3 install -r requirements.txt

CMD ["python", "server.py"]

* Criar uma imagem do contêiner

> docker build -t imagem\_alunos\_microservice .

*OBS: O ponto no final é o PATH do diretório que contém o Dockerfile*

1. Executar o contêiner

* Comando para executar o contêiner:

> docker run --name alunos-microservice -it -p 5050:5000 -d imagem\_alunos\_microservice

* Verificar localmente se a página está funcionando

> curl <http://localhost:5050/>

* Acessar a página pelo navegador.

> Acessar: <http://ip:5050/>

* Salvar seu container no Hub.Docker.

> docker tag imagem\_alunos\_microservice usuario/alunos\_microservice:latest

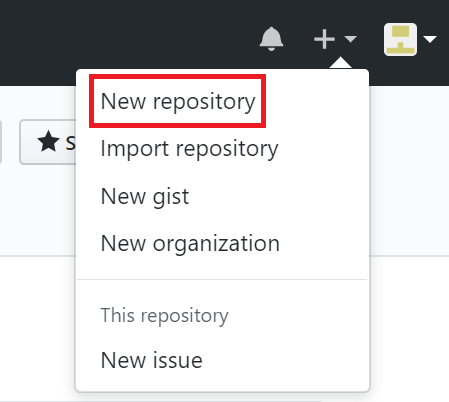
> docker login

> docker push usuario/alunos\_microservice:latest

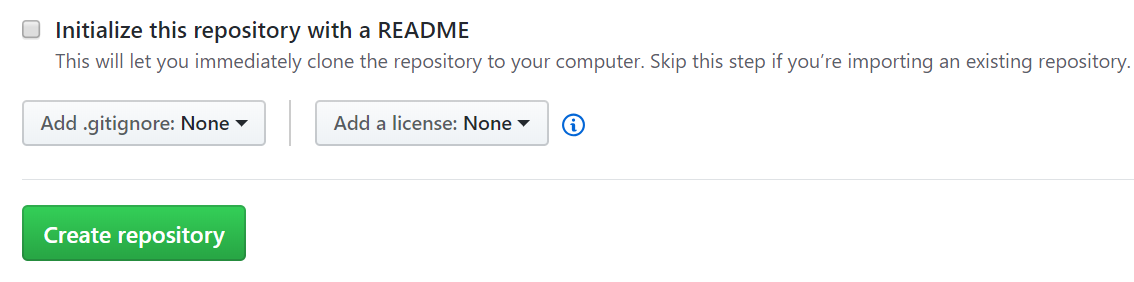
1. Salvando seus arquivos no Github:

Essa etapa depende de credenciais para autenticação no Github.com

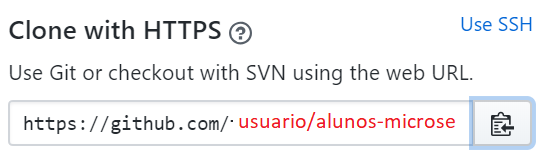
* Criar repositório Github:



* NÃO inicializar o repositório com o README (deixar essa opção desmarcada no momento da criação do repositório):



* Anotar URL do repositório:



* Salvar seus arquivos no repositório:

> git config --global user.name "aluno"

> git config --global user.email "email@fiap.com.br"

> echo -e 'alunos\_venv/\*\n\_\_pycache\_\_/\*\n\*.pyc' >> .gitignore

> git init

> git add \*

> git commit -m "criando repo"

> git remote add origin https://github.com/usuario/alunos-microservice.git

> git push -f origin master

* Acessar sua conta do Github e verificar se os arquivos criados estão no seu novo repositório.

# LAB 3

1. Montando nossa solução baseada em Apache Kafka:

* Anotar o **IP** do seu servidor

> ifconfig

* Acessar a VM a partir do [Putty](https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe)

> login: ubuntu / pass: fiap

* Rodar o Kafka e Zookeper a partir do container público spotify/kafka:

> docker run -p 2181:2181 -p 9092:9092 --net=host --env ADVERTISED\_HOST=**IP** --env ADVERTISED\_PORT=9092 --name=kafka -d spotify/kafka

* Entrar na linha de commando do container Kafka:

> docker exec -ti kafka bash

*root@94d27a5ea88a:/# 🡪 indicação de que está dentro do container*

* Listar os tópicos:

# $KAFKA\_HOME/bin/kafka-topics.sh --zookeeper localhost:2181 --list

* Criando um tópico:

# $KAFKA\_HOME/bin/kafka-topics.sh --create --topic meu-topico --partitions 1 --replication-factor 1 --if-not-exists --zookeeper localhost:2181

* Confirmar que o tópico foi criado:

# $KAFKA\_HOME/bin/kafka-topics.sh --zookeeper localhost:2181 --list

* Vamos abrir **2 telas (2º e 3º Putty)** e deixar abertos para demonstrar o envio e consumo das mensagens
* Criar mensagens

Abrir novo **Putty (2º)** e conectar no conteiner do Kafka:

> docker exec -ti kafka bash

# $KAFKA\_HOME/bin/kafka-console-producer.sh --topic meu-topico --broker-list **IP**:9092

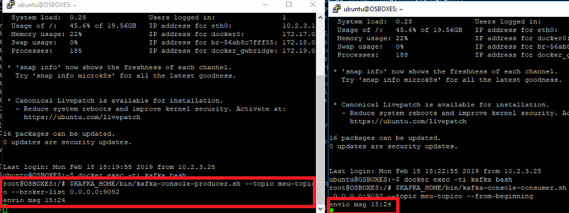
* Ler mensagens:

Abrir novo **Putty (3º)** e conectar no conteiner do Kafka:

> docker exec -ti kafka bash

# $KAFKA\_HOME/bin/kafka-console-consumer.sh --bootstrap-server IP:9092 --topic meu-topico --from-beginning

* Enviar mensagens a partir do **Producer** e verificar se aparecem no **Consumer**:



1. Criar nossa aplicação Producer:

* No Putty original (utilizado para criar o tópico), sair do container Kafka e voltar para a VM:

# exit

* Executando o ambiente:

> git clone https://github.com/tonanuvem/kafka-producer-microservice.git

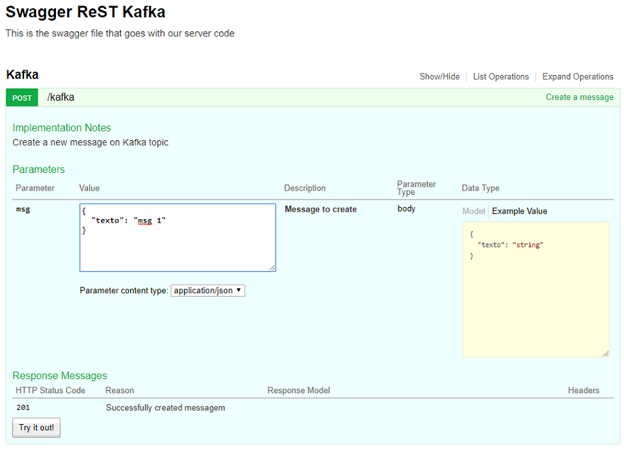
> cd kafka-producer-microservice

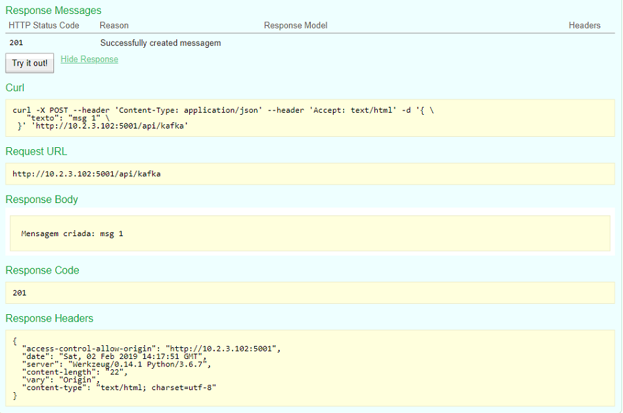
> docker build -t producer .

> docker run --name produtor\_Kafka -p 5001:5001 --rm --net=host -d producer

* Acessar pelo navegador:

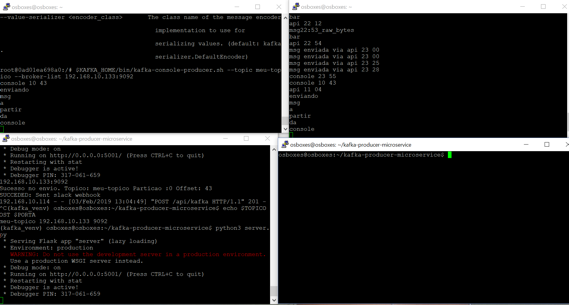
> <http://IP:5001>





1. Criar nossa aplicação Consumer:

* Vamos abrir um **novo Putty (4º)** para trabalhar com Consumidor:



* Executando o ambiente:

> git clone https://github.com/tonanuvem/kafka-consumer-microservice.git

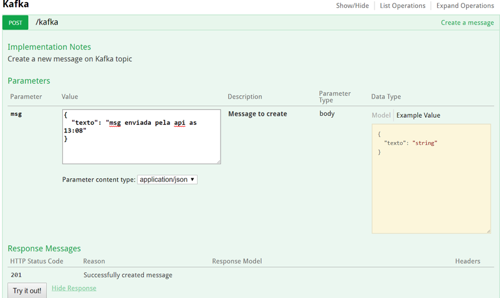
> cd kafka-consumer-microservice

> docker build -t consumer .

> docker run --name consumer\_Kafka --rm --net=host -d consumer

* Executar novos testes de envio de mensagem pelo navegador:

> <http://IP:5001/api/ui>



* Mensagens deverão aparecer nos canais #lab-produtor e #lab-consumidor do Slack:



* Publicar suas imagens no Hub Docker:

> docker tag producer usuario/kafka\_producer:latest

> docker tag consumer usuario/kafka\_consumer:latest

Caso tenha mudado de computador:

> docker logout

> docker login

> docker push usuario/kafka\_producer:latest

> docker push usuario/kafka\_consumer:latest

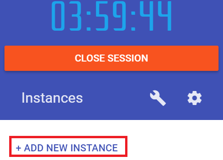
# LAB 3 – BONUS

1. Testando sua imagem no play-with-docker.com a partir do **docker-compose.yml**:

* Acessar o site play-with-docker.com e inserir suas credenciais que foram criadas no página do Hub Docker:



* Iniciar uma nova instância Docker:



* Criar o arquivo docker-compose.yml:

> touch docker-compose.yml



* Clicar em EDITOR para inserir o conteúdo abaixo no arquivo docker-compose.yml:

version: '3'

services:

kafka:

image: spotify/kafka

container\_name: kafka

network\_mode: "host"

ports:

- 9092:9092

- 2181:2181

environment:

- "ADVERTISED\_HOST=0.0.0.0"

- "ADVERTISED\_PORT=9092"

producer:

image: **usuario**/kafka\_producer:latest

container\_name: producer

network\_mode: "host"

ports:

- 5001:5001

restart: on-failure

environment:

- "HOST=0.0.0.0"

- "PORTA=9092"

- "TOPICO=meu-topico"

consumer:

image: **usuario**/kafka\_consumer:latest

container\_name: consumer

network\_mode: "host"

environment:

- "HOST=0.0.0.0"

- "PORTA=9092"

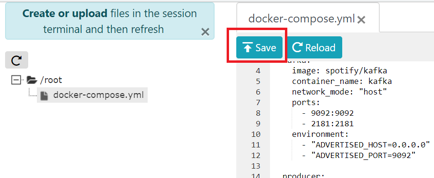
- "TOPICO=meu-topico"

restart: on-failure

depends\_on:

- kafka

- producer



* Executar os diversos serviços através do docker-compose:

> docker-compose up

* Para acessar a aplicação publicada, acessar o url fornecida, substituindo conforme abaixo:

**de**:

ip172*-18-0-10-bhlgk2b0mkfg00dedl3g***@**direct.labs.play-with-docker.com

**para**:

ip172*-18-0-10-bhlgk2b0mkfg00dedl3g***-5001.**direct.labs.play-with-docker.com

# LAB 4

1. Monitoração de Performance de Aplicações



**Essa etapa do laboratório será executada no LAB FIAP.**

**Montando nossa solução ELK:**

* Fazer o download da solução:

> cd ~

> git clone <https://github.com/tonanuvem/docker-apm-monitoring.git>

> cd docker-apm-monitoring

* Executar a solução:

> docker-compose up

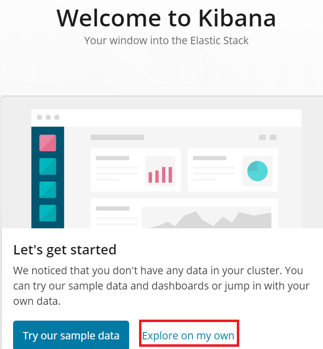
* Acessando nossa aplicação:

> http://IP:3000/

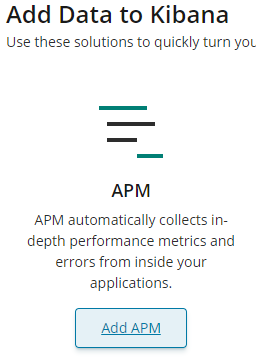
* Utilizando o APM no Gerenciamento de Serviços:

> <http://IP:5601/>

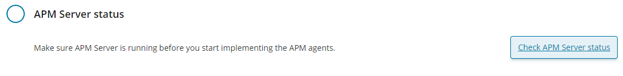
Caso apareça a tela de boas vindas, clicar em “Explore on my own”

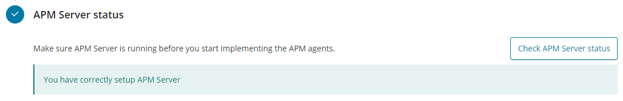


**Configurando o APM: clicar em ADD APM**

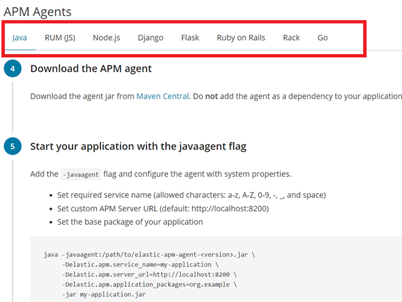


* Rolar a barra de rolagem e clicar em CHECK APM SERVER STATUS

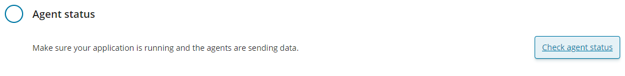


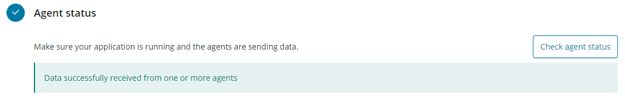


* O Elastic APM suporta as seguintes linguagens de programação:

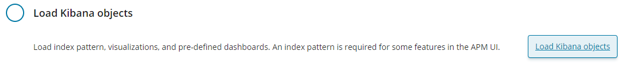


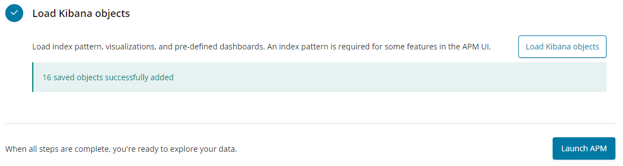
* Rolar a barra de rolagem e clicar em CHECK AGENT STATUS



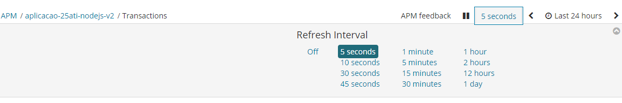


* Clicar em LOAD KIBANA OBJECTS





* Clicar em LAUNCH APM
* Clicar em AUTO-REFRESH e Configurar o tempo de atualização no Kibana para 5 segs:



* Acesse outras páginas da aplicação:

> <http://IP:3000/healthcheck/>

> <http://IP:3000/fiap>/

> <http://IP:3000/bar>/

* Verifique novamente o APM clicando em REQUESTS, apareceu o tempo de resposta de cada rota?



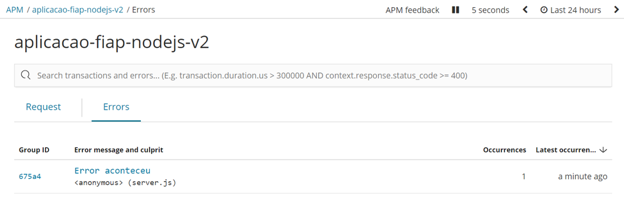
Uma imagem contendo captura de tela

Descrição gerada automaticamente

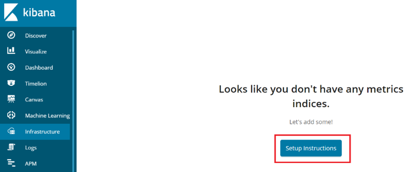
* Acesse página que gerará ERROR e verifique novamente o APM:

> <http://IP:3000/erro>/

* Clique em Errors. O Elastic mostra até a linha de Código que apresentou problema.

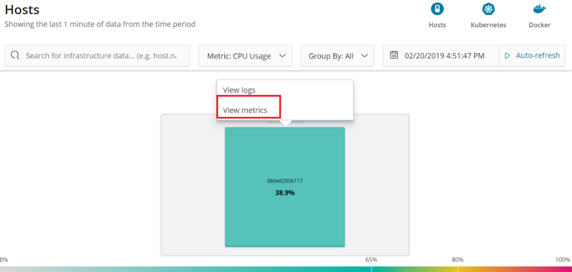


1. **Conhecendo o Monitoramento do Elastic através de Métricas:**

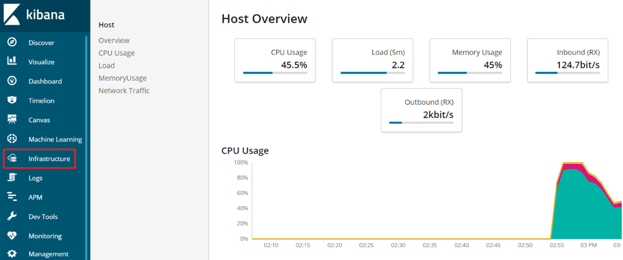




* Clicar em “View Metrics”:



* Verificar as informações disponíveis sobre Monitoração



* Finalizar a execução:

> cd ~/docker-apm-monitoring

> docker-compose down