Sumário

[O que são? 1](#_Toc83315516)

[Qual é o tamanho do microsserviço? 1](#_Toc83315517)

[Serviços 1](#_Toc83315518)

[Impactos na empresa 1](#_Toc83315519)

[Microsserviços e DDD (domain driven design) 1](#_Toc83315520)

[Dependências 1](#_Toc83315521)

[Rotas e padrão REST 2](#_Toc83315522)

[Classe startup 2](#_Toc83315523)

[Criando APi’s com node 2](#_Toc83315524)

[Instalar os pacotes 3](#_Toc83315525)

[Testar nosso primeiro ponto de acesso 8](#_Toc83315526)

[Instalar o nodemon 8](#_Toc83315527)

[Configurar o body-parser 8](#_Toc83315528)

[Criar um endpoint de teste 9](#_Toc83315529)

[Configurar o MongoDb 10](#_Toc83315530)

# O que são?

* Determina um modelo de desenvolvimento;
* Também é um conjunto de práticas;
* Aumenta a velocidade de desenvolvimento;
* Agnóstico de tecnologia;
* Permite escalabilidade;
* Princípios de e padrões de tecnologias.

# Qual é o tamanho do microsserviço?

* Não dá para mensurar o tamanho;
* Faz apenas uma coisa bem-feita;
* Delimitação de negócio; (Bounded contexto);
* Agir independentemente;
  + Incluir o *deploy*.
* É o ato de dividir o domínio em subdomínios;

# Serviços

* São componentes independentes;
* Interoperabilidade;
* SOA.

# Impactos na empresa

* O software muito grande, leva mais tempo para ficar pronto;
* Quando o software é grande, a dependência também é enorme.

# Microsserviços e DDD (domain driven design)

* Podemos relacionar o microsserviço ao DDD;
* Domínio do negócio;
* Sub-domínio;
  + Uma parte do domínio: usuário, pedido, produto, etc.
* Contexto delimitado. (Bounded contexto);
  + Capacidade de negócio;

# Dependências

Os serviços não podem depender um do outro;

O DDD permite replicar a entidade;

# Rotas e padrão REST

* Definir um padrão para que a equipe tenha um caminho a seguir;
* Por falta de padrões, endpoints podem deixar de seguir boa práticas.
  + Exemplo de endpoints: getAllStudents ou get-all-students.

# Classe startup

* Utilizada para configurar algumas características da aplicação, como por exemplo:
  + Injeção de dependência;
  + Configuração de Swagger;
  + Configurar o CORS.

# Criando APi’s com node

* Precisaremos instalar ou atualizar (caso já tenha instalado) a versão do node;
* Baixar a versão: <https://nodejs.org/en/download/>
* O comando node --version retorna a versão instalada na sua máquina.

Texto

Descrição gerada automaticamente

* Vamos utilizar o Visual Studio Code. <https://code.visualstudio.com/docs/?dv=win>
* npm -- version é um gerenciador de pacote Javascript;

Iniciar o projeto node

* Abrir a linha de comando na pasta onde o projeto será criado;
* npm init -y ( comando -y evita solicitar um formulário para preencher sobre o projeto);

Texto

Descrição gerada automaticamente

* Abrir o Visual Studio Code na pasta do projeto:

Texto

Descrição gerada automaticamente

# Instalar os pacotes

* npm install http express debug --save
* O comando **npm install** vai fazer a instalação de todos os pacotes existentes;
* Criar as variáveis e fazer os importes necessários para o servidor

Criar o arquivo App.js

* Criar o arquivo src/app.js

const express = require('express')

const app = express()

app.use(express.json())

app.use(express.urlencoded({extended: true}))

// Habilita o CORS

app.use(function (req, res, next) {

res.header('Access-Control-Allow-Origin', '\*');

res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, x-access-token');

res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');

next();

});

//registrar os models

//criar rotas

const index = require('./routes/index')

app.use('/', index)

module.exports = app;

Criar as configurações do server

* Criar o arquivo /bin/server.js.

const http = require('http')

const debug = require('debug')('nodestr:server')

const app = require('../src/app');

const port = searchPort(process.env.PORT || '3000');

const server = http.createServer(app)

server.listen(port)

server.on('error', onError);

server.on('listening', onListening);

console.log(`Api executada na porta: ${port}`)

function searchPort(val) {

const port = parseInt(val, 10)

if (isNaN(port))

return val;

if (port >= 0)

return port;

return false;

}

function onError(error) {

if (error.syscall !== 'listen') {

throw error;

}

const bind = typeof port === 'string'

? 'Pipe ' + port

: 'Port ' + port;

switch (error.code) {

case 'EACCES':

console.error(bind + ' requires elevated privileges');

process.exit(1);

break;

case 'EADDRINUSE':

console.error(bind + ' is already in use');

process.exit(1);

break;

default:

throw error;

}

}

function onListening() {

const addr = server.address();

const bind = typeof addr === 'string'

? 'pipe ' + addr

: 'port ' + addr.port;

debug('Listening on ' + bind);

}

Criar a primeira rota

* Criar o seguinte arquivo src/routes/index.js

'use strict'

const express = require('express')

const router = new express.Router()

//Apenas para testar a API

router.get('/', (req, res, next) => {

res.status(200).send({

title: 'API',

version : '0.0.1'

});

});

module.exports = router

Configurar o start do servidor no package.json

* No arquivo package.json insira o start do server.

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start" : "node ./bin/server.js"

},

# Testar nosso primeiro ponto de acesso

Vamos utilizar o postman

* Incluir um atributo de start no package.json para iniciar a aplicação;
* Após esta alteração para executar ficará simples: npm start

curl --location --request GET 'http://localhost:3000'

# Instalar o nodemon

* Utilizado para escutar os arquivos js e resetar o servidor automaticamente;
* Não precisa ficar executando a aplicação várias vezes;
* Comando para instalação:

npm install nodemon --save-dev

* Após a execução, se olharmos no package.json, teremos uma dependência.
* Para executar:

nodemon .\bin\server.js

# Configurar o body-parser

* Os endpoints normalmente receberão json, para convertermos na aplicação, vamos configurar o body-parser do próprio express.

Inserir as informações no app.js

app.use(express.json())

app.use(express.urlencoded({ extended: true}))

# Criar um endpoint de teste

* Primeiro vamos criar a controller:

src/controllers/product-controller.js

exports.get = async(req, res, next) =>{

res.status(200).send(

{

message : 'rota de produto'

}

);

}

* Agora vamos criar a rota

src/routes/product-route.js

const express = require('express')

const router = express.Router();

const controller = require('../controllers/product-controller')

router.get('/', controller.get);

module.exports = router;

* Vamos registrar a rota no app.js

const productRoute = require('./routes/product-route')

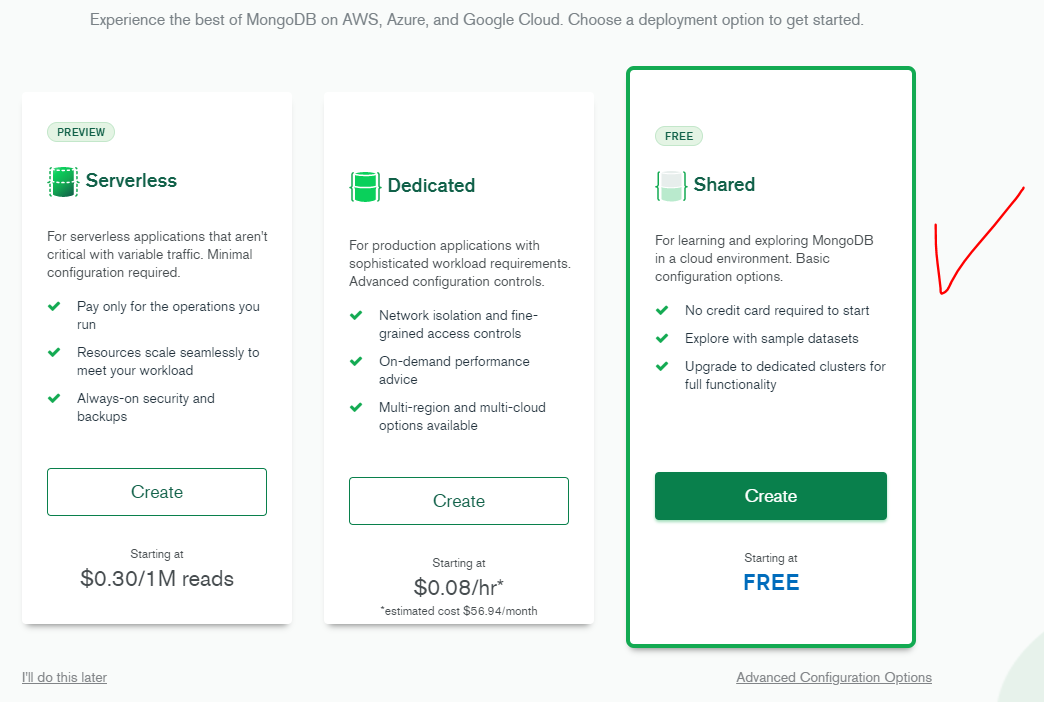
app.use('/products', productRoute)

# Configurar o MongoDb

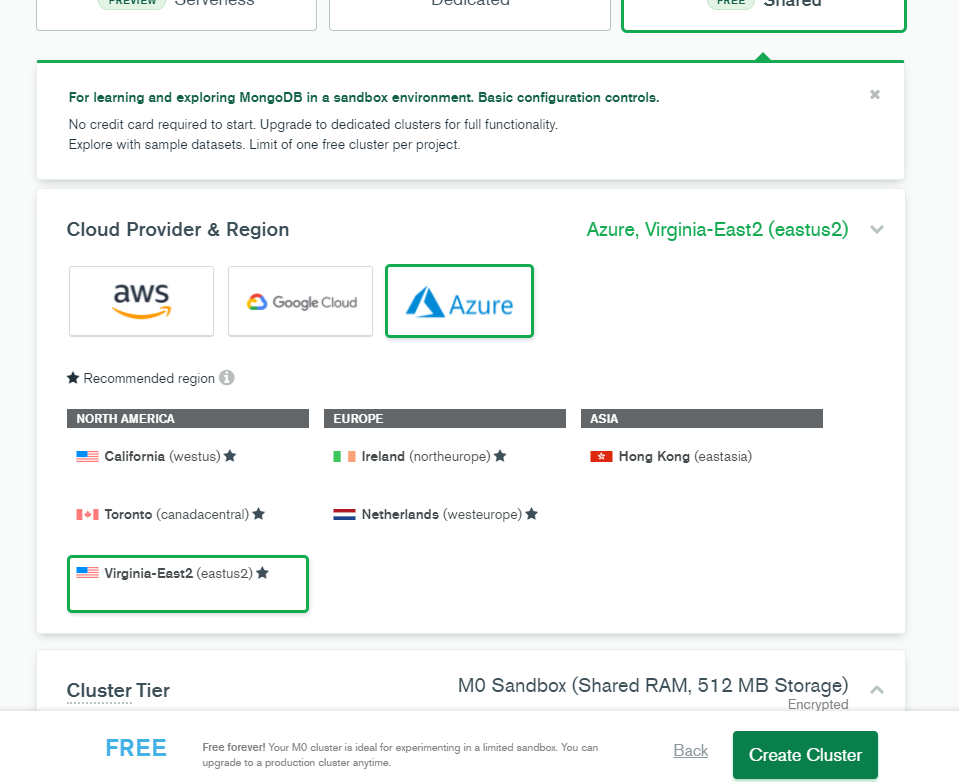
Vamos fazer uma conexão com mongoDb, para isso, cria uma conta free:

https://www.mongodb.com/

* Basta se cadastrar e confirmar o e-mail enviado.

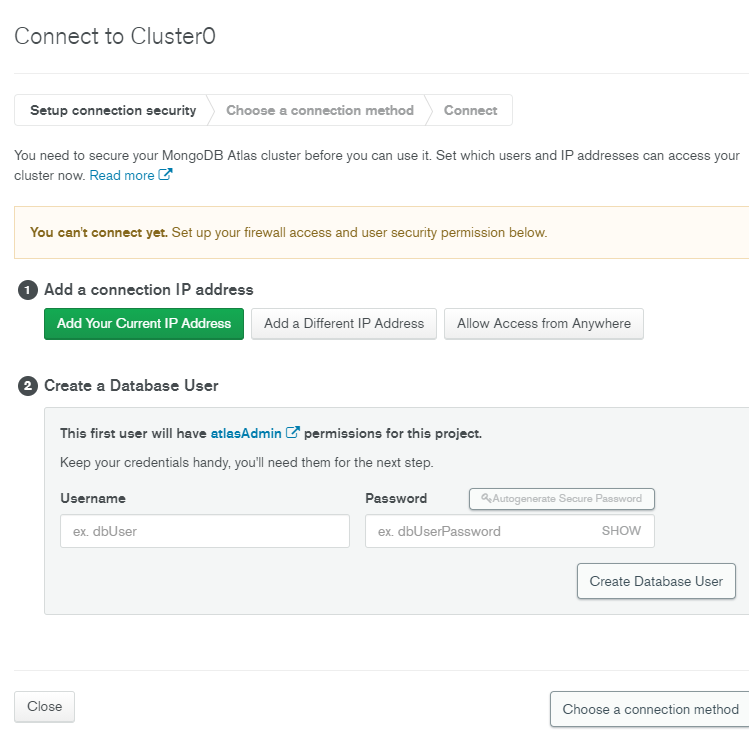


Criar um cluster



Recuperar a string de conexão

Clique em Connect



Importante permitir qualquer acesso por enquanto (Allow Access from Anywhere)

Criar um database

Vá até collection e crie uma collection

Voltar para aplicação - Instalar mongoose

npm install mongoose --save

Criar a model

src/models/product.js

'use strict'

const mongoose = require('mongoose')

const Schema = mongoose.Schema;;

const schema = new Schema({

title: {

type: String,

required: true,

trim: true

},

description: {

type: String,

required: true

},

price: {

type: Number,

required: true

},

active: {

type: Boolean,

required: true,

default: true

}

});

module.exports = mongoose.model('Product', schema)

Registrar a model

Abra o arquivo app.js e faça o registro da model

//registar as models

require('./models/product')

Criar a repository

* Vamos criar dentro de src uma pasta para repositories

src/repositories/product-repository

const mongoose = require('mongoose')

const Product = mongoose.model('Product')

exports.get = async () => {

const res = await Product.find({}, 'title price \_id active');

return res;

}

Conexão mongo

Volte ao arquivo app.js e insira a conexao:

const mongoose = require('mongoose')

mongoose.connect('sua-connection-aqui')