ASPNET - 7 - Blog Pessoal - Publicar

1. Cloud
2. Heroku
3. Preparando ambiente no Heroku
4. Preparando ambiente para publicação
5. Publicando

1. Cloud

A definição de cloud poderá parecer obscura mas, essencialmente, é um termo utilizado para descrever uma rede global de servidores, cada um deles com uma função única. A cloud não é uma entidade física, mas sim uma rede vasta de servidores remotos em todo o mundo que estão interligados e que devem funcionar como um ecossistema único. Estes servidores foram concebidos para armazenar e gerir dados, executar aplicações ou fornecer conteúdos ou um serviço, como vídeos em transmissão de fluxo, webmail, software de produtividade para escritórios ou comunicação social. Em vez de aceder aos ficheiros e dados a partir de um computador local ou pessoal, está a aceder-lhes online a partir de um dispositivo com Internet — a informação estará disponível onde quer que esteja e em qualquer altura.

As empresas utilizam quatro métodos diferentes para implementar recursos da cloud. Existe uma cloud pública que partilha recursos e oferece serviços ao público através da Internet, uma cloud privada que não é partilhada e oferece serviços através de uma rede privada interna normalmente alojada no local, uma cloud híbrida que partilha serviços entre clouds públicas e privadas consoante a finalidade, e uma cloud da comunidade que partilha recursos apenas entre organizações, como instituições do governo.

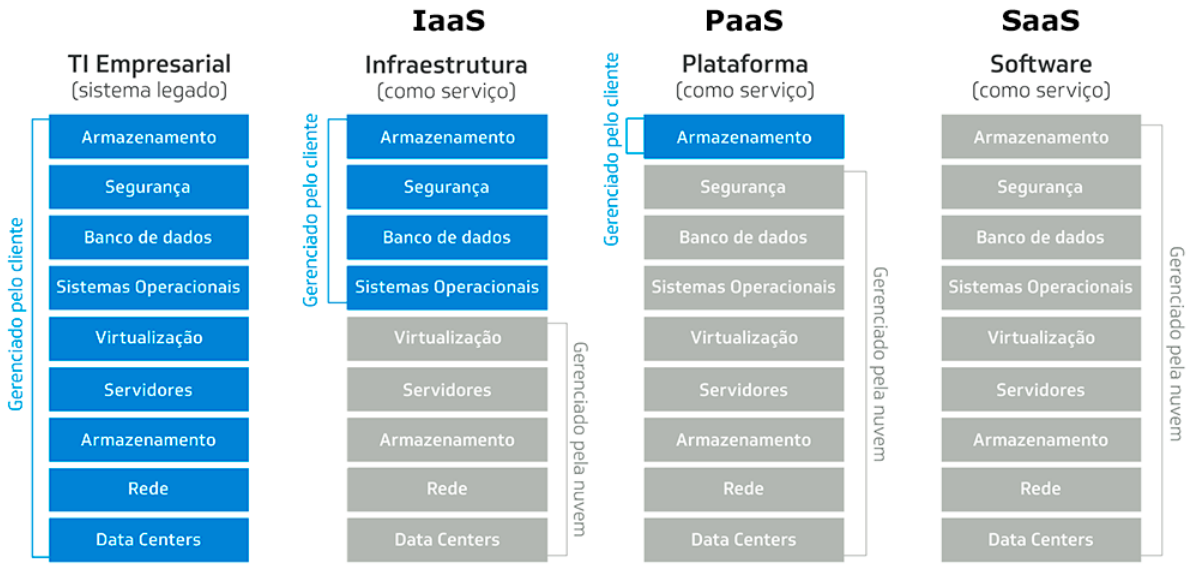
Cloud pública

A cloud pública é definida como serviços de informática oferecidos por fornecedores externos através da Internet pública, que os disponibilizam a todas as pessoas que os queiram comprar. Podem ser gratuitos ou vendidos a pedido, o que permite aos clientes pagar apenas pela utilização de ciclos de CPU, armazenamento ou largura de banda que consomem.

Cloud privada

A cloud privada é definida como os serviços de informática oferecidos através da Internet ou de uma rede interna privada e apenas a determinados utilizadores, em vez do grande público. Também denominada cloud interna ou empresarial, a computação na cloud privada proporciona às empresas muitos dos benefícios de uma cloud pública - incluindo self-service, escalabilidade e elasticidade - com controlo e personalização adicionais, disponíveis a partir de recursos dedicados numa infraestrutura de informática alojada no local. Além disso, as clouds privadas proporcionam um nível mais elevado de segurança e privacidade, através de firewalls da empresa e de alojamento interno, para garantir que as operações e os dados confidenciais não ficam acessíveis a fornecedores terceiros. Uma desvantagem é que o departamento de TI da empresa tem a seu cargo os custos e a responsabilidade de gerir a cloud privada. Por conseguinte, as clouds privadas requerem as mesmas despesas de pessoal, gestão e manutenção que a propriedade de um datacenter tradicional.

Dois modelos de serviços cloud podem ser fornecidos numa cloud privada. O primeiro é a infraestrutura como serviço (IaaS), que permite a uma empresa utilizar recursos de infraestrutura, como computação, rede e armazenamento como um serviço. O segundo é a plataforma como serviço (PaaS), que permite a uma empresa fornecer tudo, desde aplicações simples com base na cloud a aplicações empresariais sofisticadas.



Cloud híbrida

Uma cloud híbrida, por vezes denominada híbrida cloud, é um ambiente informático que combina um datacenter no local (também denominado cloud privada) com uma cloud pública, ao permitir que os dados e as aplicações sejam partilhados entre si. Algumas pessoas definem a cloud híbrida para incluir configurações de "multicloud", em que uma organização utiliza mais de uma cloud pública além do respetivo datacenter no local.

2. Heroku

Heroku é uma plataforma amplamente confiável como uma oferta de serviço que permite aos desenvolvedores realizar implantação, escalonamento e gerenciamento de aplicativos sem complicações. Esta plataforma oferece suporte para uma ampla gama de linguagens de programação, como Java, Ruby, PHP, Node.js, Python, Scala e Clojure. O Heroku executa aplicativos por meio de contêineres virtuais conhecidos como Dynos.

O Heroku cobra de seus usuários com base no número de máquinas virtuais que são necessárias para seus aplicativos. A plataforma Heroku e os aplicativos criados pelo usuário usam Amazon Web Services como infraestrutura subjacente. Os desenvolvedores podem obter um rápido desenvolvimento de aplicativos usando-o, pois é bastante conveniente.

Vantagens

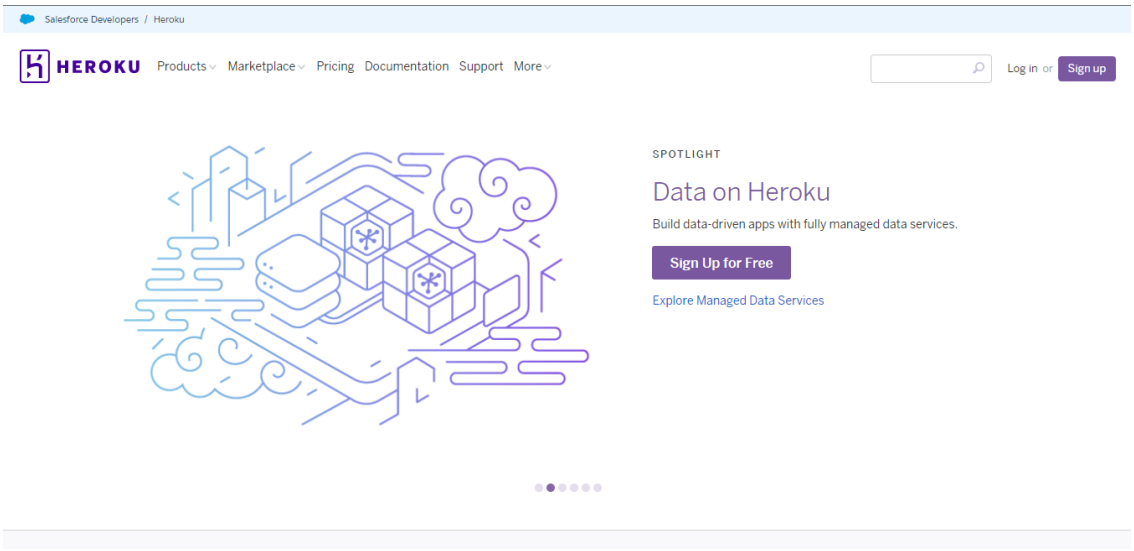
- Gratuito para começar;
- Fácil de usar;
- Centrado no desenvolvedor;
- Enquanto o desenvolvedor foca na codificação o heroku gerencia o servidor;
- Fácil de escalar;
- Seguro;
- Possui CLI (Cliente de linha de comando).

Desvantagens

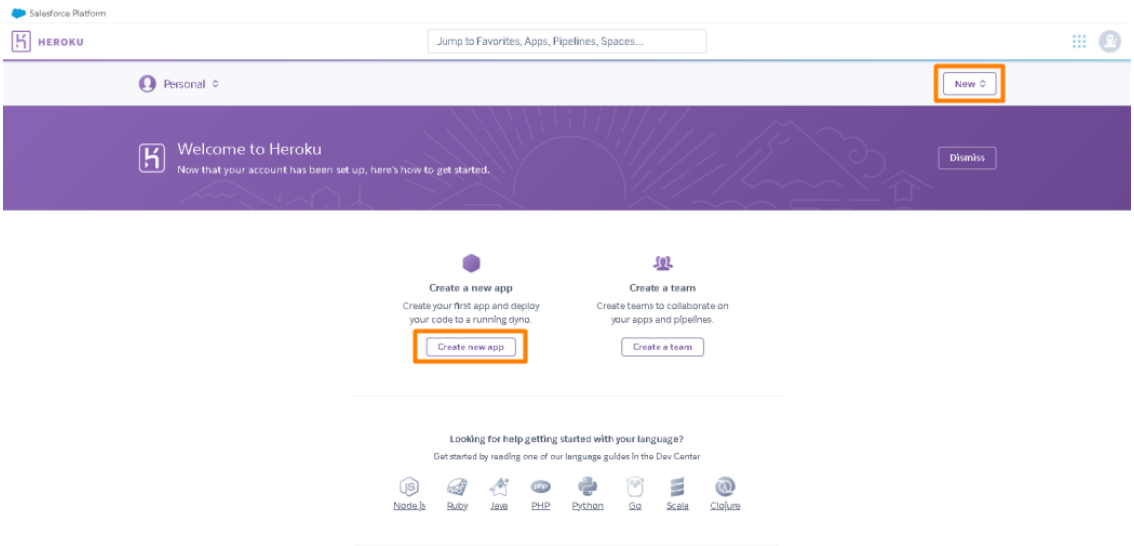
- Valor;
- Sleeping Apps;
- Limitada a regiões EUA e Europa;
- Não fornece endereços de IP estáticos;
- Cobrado em dólares e não da suporte para moeda local.

3. Preparando ambiente no Heroku

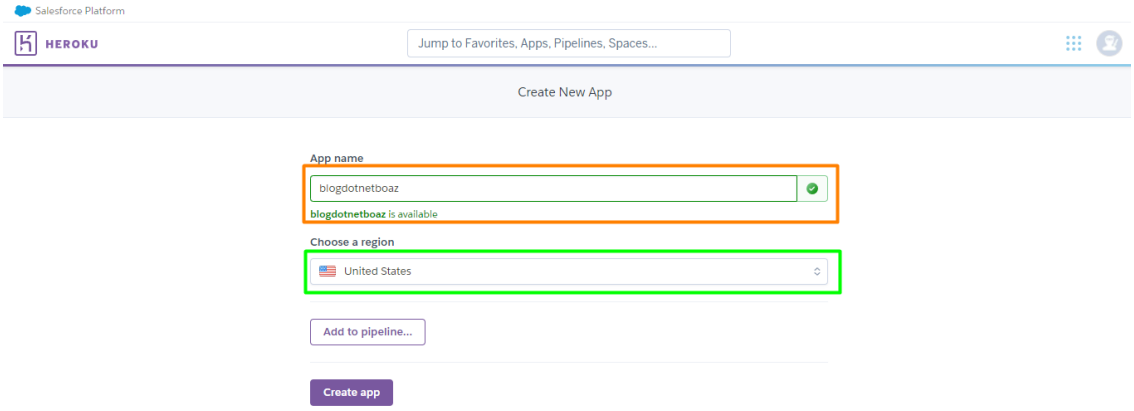
1. Fazer cadastro no site. Link:



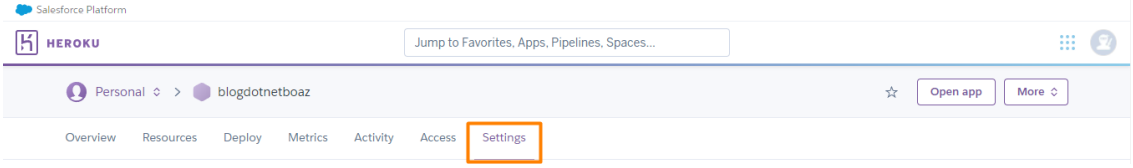
2. Logar no sistema.
3. Clicar em ***Create new app***:



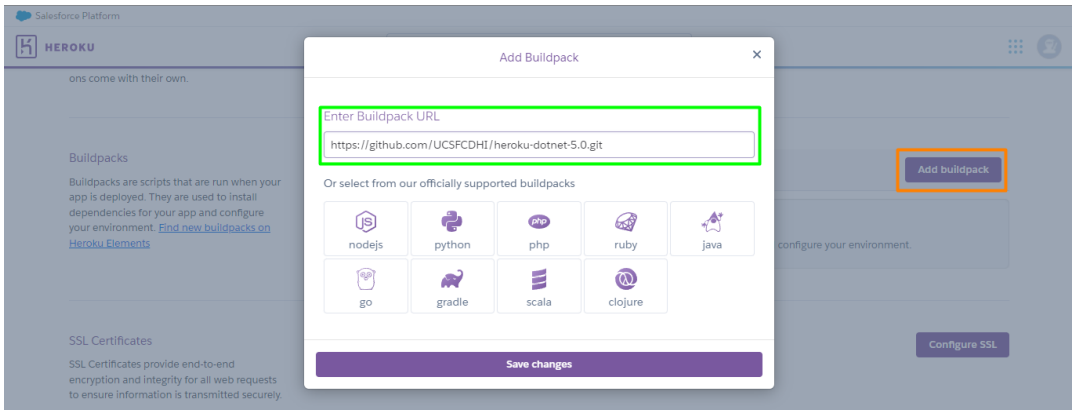
4. Defina o nome do app (laranja) e a região do servidor (verde), clicar em ***Create app***:



5. Para adicionar o **Buildpack** para dotnet, acessar a aba ***Settings***:

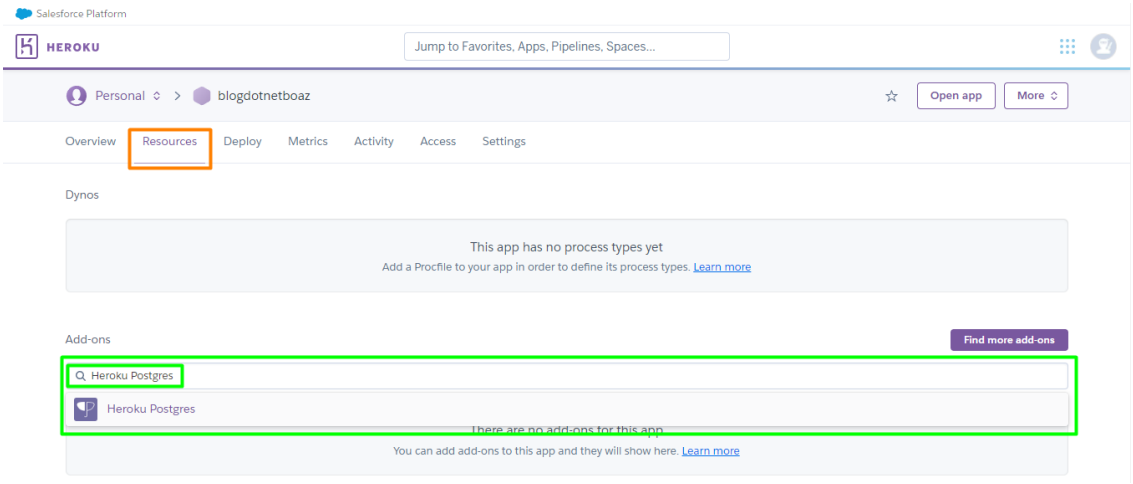


   - Na sessão ***Settings*** clicar no botão ***Add buildpack*** e inserir a URL Buildpack abaixo no campo correspondente:
     URL: https://github.com/UCSFCDHI/heroku-dotnet-5.0.git

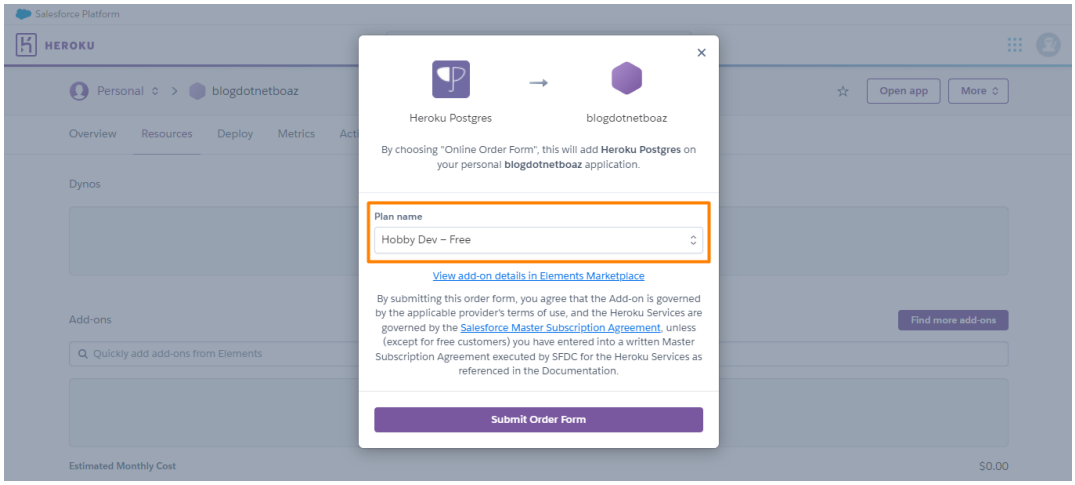


   - Salve a operação clicando em ***Save changes***

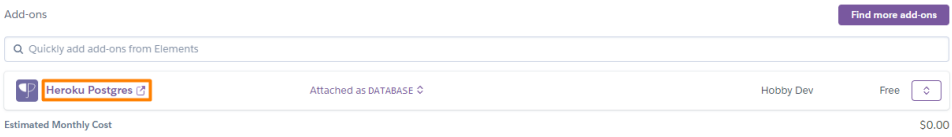
6. Para adicionar um banco de dados, entre na aba ***Resources*** vá até a sessão ***Add-ons*** e pesquise por ***Heroku Postgres***, click na opção:



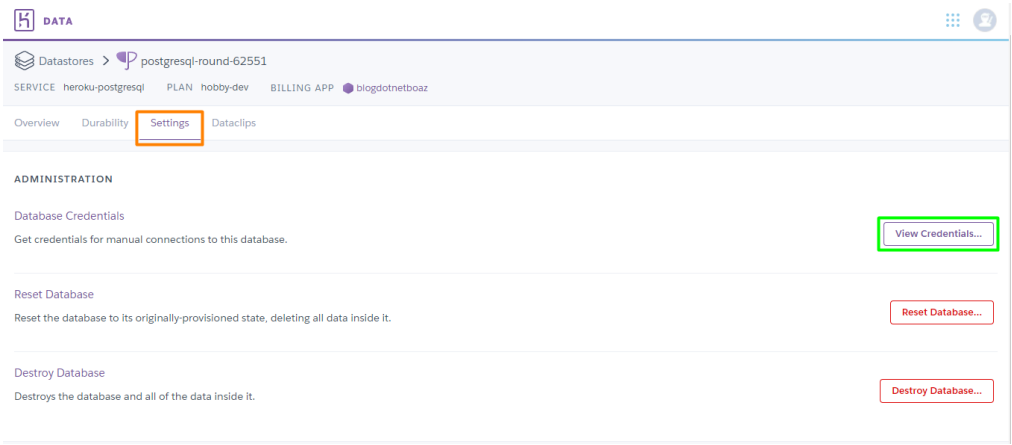
   - Selecionar a opção Hobby Dev - Free:



- Salve a operação clicando em ***Submit Order Form***
- Click em ***Heroku Postgres***, esta ação abrira uma nova aba no navegador:



  - Na aba ***Settigns***, clicar em ***View Credentials***:



    - Guarde as credenciais passadas, pois será utilizada na string de conexão do banco de dados que teremos dentro do heroku:

Definição	Utilidade
`Host`	Local de hospedagem de seu banco de dados;
`Database`	Nome do banco de dados localizado no host;
`User`	Autenticação de usuário;
`Password`	Senha de usuário;
`Port`	Porta do banco de dados.

4. Preparando ambiente para publicação

Será necessário configurar sua aplicação para que a mesma atenda alguns requisitos do servidor, então validar os pontos abaixo:

Adicionar string de conexão em appsettings.json (raís do projeto):

Este documento é utilizado para guardar algumas configurações de nossa aplicação e vai ser usado para armazenar dados do tipo: Ambiente, string de conexão, palavra chave etc. Abaixo segue o esquema, alterar apenas a sessão de `ConnectionStringProd`:

```
{
  "Enviroment": {
    "Start": "PROD"
  },
  "ConnectionStringsDev": {
    "DefaultConnection": "Server=localhost; Initial Catalog=db_blogpessoal;
Trusted_Connection=True;"
  },
  "ConnectionStringsProd": {
    "DefaultConnection":
      "Host={HOST};Database={DATABASE};User Id={USER};Port={PORT};Password=
{PASSWORD};Pooling=true;SSL Mode=Require;TrustServerCertificate=True;"
  },
  "Settings": {
    "Secret": "fedaf7d8863b48e197b9287d492b708e"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Campos:

Campo	Utilização
`Enviroment`	Ambiente de codificação. Pode ser DEV(Quando for rodar em sua maquina) ou PROD(Quando for implantar no servidor);
`ConnectionStringsDev`	Guarda string de conexão para banco local;
`ConnectionStringsProd`	Guarda string de conexão para servidor. Neste campo é importante alterar (Host={HOST};Database={DATABASE};User Id={USER};Port={PORT};Password={PASSWORD}) colocando os parâmetros fornecidos pelo Heroku para conexão do banco. Alterar de acordo com o que for passado na definição passada pelo servidor.
`Settings`	Guarda string de *secret*, utilizada na aprte de segurança JWT;
`Logging`	Variáveis utilizadas pelo sistema de logs;
`AllowedHosts`	Permissão de host, também esta sendo utilizada pelo sistema;

Instalando pacote de PostgreSQL:

Dentro do diretório de `BlogPessoal/` executar o comando abaixo para instalar o pacote que da suporte ao *PostgreSQL*. O comando deve ser executado no diretório em que se encontra o arquivo *BlogPessoal.csproj*.

```
dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL --version 5.0.1
```

Modificar método `ConfigureServices` no documento *Startup.cs*:

Neste método modificar a configuração do Banco de dados. Neste momento sua aplicação deverá atender a dois ambientes, um de desenvolvimento (para dar manutenção) e outro de produção. Para isso definiremos uma condicional que irá analisar se uma variável de configuração esta definida como `PROD`, caso verdadeiro, a configuração utilizada para banco de dados será a do banco PostgreSQL.

```
// Configuraçãp Banco de Dados
if (Configuration["Enviroment:Start"] == "PROD")
{
    services.AddEntityFrameworkNpgsql()
        .AddDbContext<BlogPessoalContexto>(
        opt =>
opt.UseNpgsql(Configuration["ConnectionStringsProd:DefaultConnection"]));
}
else
{
    services.AddDbContext<BlogPessoalContexto>(
        opt =>
opt.UseSqlServer(Configuration["ConnectionStringsDev:DefaultConnection"]));
}
```

O método ira ficar desta maneira:

```
public void ConfigureServices(IServiceCollection services)
{
    // Configuraçãp Banco de Dados
    if (Configuration["Enviroment:Start"] == "PROD")
    {
        services.AddEntityFrameworkNpgsql()
            .AddDbContext<BlogPessoalContexto>(
            opt =>
opt.UseNpgsql(Configuration["ConnectionStringsProd:DefaultConnection"]));
    }
    else
    {
        services.AddDbContext<BlogPessoalContexto>(
            opt =>
opt.UseSqlServer(Configuration["ConnectionStringsDev:DefaultConnection"]));
    }

    // Configuração Repositorios
    services.AddScoped<IUsuario, UsuarioRepositorio>();
    services.AddScoped<ITema, TemaRepositorio>();
    services.AddScoped<IPostagem, PostagemRepositorio>();

    // Configuração de Controladores
    services.AddCors();
    services.AddControllers();

    // Configuração de Serviços
    services.AddScoped<IAutenticacao, AutenticacaoServicos>();

    // Configuração do Token Autenticação JWTBearer
    var chave = Encoding.ASCII.GetBytes(Configuration["Settings:Secret"]);
    services.AddAuthentication(
        a =>
        {
            a.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
            a.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        }).AddJwtBearer(
        b =>
        {
            b.RequireHttpsMetadata = false;
            b.SaveToken = true;
            b.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new SymmetricSecurityKey(chave),
                ValidateIssuer = false,
                ValidateAudience = false
            };
        }
    );

    // Configuração Swagger
    services.AddSwaggerGen(
        s =>
```

```
        {
            s.SwaggerDoc("v1", new OpenApiInfo { Title = "Blog Pessoal", Version
= "v1" });

            s.AddSecurityDefinition(
                "Bearer",
                new OpenApiSecurityScheme()
                {
                    Name = "Authorization",
                    Type = SecuritySchemeType.ApiKey,
                    Scheme = "Bearer",
                    BearerFormat = "JWT",
                    In = ParameterLocation.Header,
                    Description = "JWT authorization header utiliza: Bearer +
JWT Token",
                });

            s.AddSecurityRequirement(
                new OpenApiSecurityRequirement
                {
                    {
                        new OpenApiSecurityScheme
                        {
                            Reference = new OpenApiReference
                            {
                                Type = ReferenceType.SecurityScheme,
                                Id = "Bearer"
                            }
                        },
                        new List<string>()
                    }
                });

            var xmlFile = $"
{Assembly.GetExecutingAssembly().GetName().Name}.xml";
            var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
            s.IncludeXmlComments(xmlPath);
        }
    );
}
```

Modificar método `Configure` no documento *Startup.cs*:

Neste método adicionar configurações que estão dentro do ambiente de desenvolvimento, fora do condicional. Desta maneira quando a aplicação rodar dentro do servidor será possivel permanecer com todo contexto da aplicação.

Incluir código abaixo:

```
// Ambiente de produção
contexto.Database.EnsureCreated();
app.UseDeveloperExceptionPage();
app.UseSwagger();
app.UseSwaggerUI(c => {
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "BlogPessoal v1");
    c.RoutePrefix = string.Empty;
});
```

O método irá ficar desta maneira:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
BlogPessoalContexto contexto)
{
    // Ambiente de Desenvolvimento
    if (env.IsDevelopment())
    {
        contexto.Database.EnsureCreated();
        app.UseDeveloperExceptionPage();
        app.UseSwagger();
        app.UseSwaggerUI(c => {
            c.SwaggerEndpoint("/swagger/v1/swagger.json", "BlogPessoal v1");
            c.RoutePrefix = string.Empty;
        });
    }

    // Ambiente de produção
    contexto.Database.EnsureCreated();
    app.UseDeveloperExceptionPage();
    app.UseSwagger();
    app.UseSwaggerUI(c => {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "BlogPessoal v1");
        c.RoutePrefix = string.Empty;
    });

    // Rotas
    app.UseRouting();

    app.UseCors(c => c
                .AllowAnyOrigin()
                .AllowAnyMethod()
                .AllowAnyHeader()
               );

    // Autenticação e Autorização
    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(
        endpoints =>
        {
            endpoints.MapControllers();
        }
    );
}
```

5. Publicando

Pra executar esta sessão deve ser considerado a instalação do Heroku CLI (Cliente de linha de comando) no Link. Após sua instalação logar no terminal antes dos passos abaixo com o comando:

```
heroku login
```

Este comando irá solicitar que seja confirmado suas credenciais em um navegador, a mensagem ira aparecer com seu email confirmando que esta logado:

```
heroku: Press any key to open up the browser to login or q to exit
 ›   Warning: If browser does not open, visit
 ›   https://cli-auth.heroku.com/auth/browser/***
heroku: Waiting for login...
Logging in... done
Logged in as me@example.com
```

Caso projeto já tenha um repositório GIT

1. Adicionar o remote do projeto:

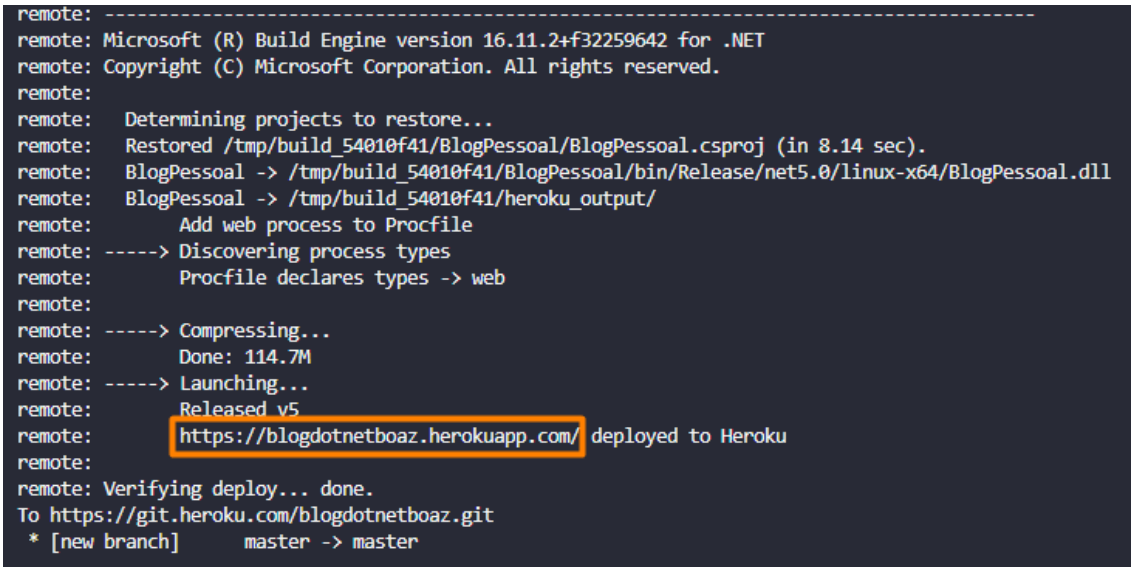
   ```
   heroku git:remote -a blogdotnetboaz
   ```

   Tenha atenção no comando acima pois o nome definido após a comando `-a`, é o nome do projeto no Heroku;

2. Implantar com comando GIT PUSH:

   ```
   git push heroku master
   ```

   Desta maneira sua aplicação será implantada. Validar depois acessando o link fornecido al final da implantação:



Caso sua aplicação não abrir, é possivel validar a causa digitando no terminal o comando abaixo:

```
heroku logs --tail
```

Este comando ira buscar todas as falhas possíveis que poderão causar a não implantação. Corrigir os erros e voltar a implantar.

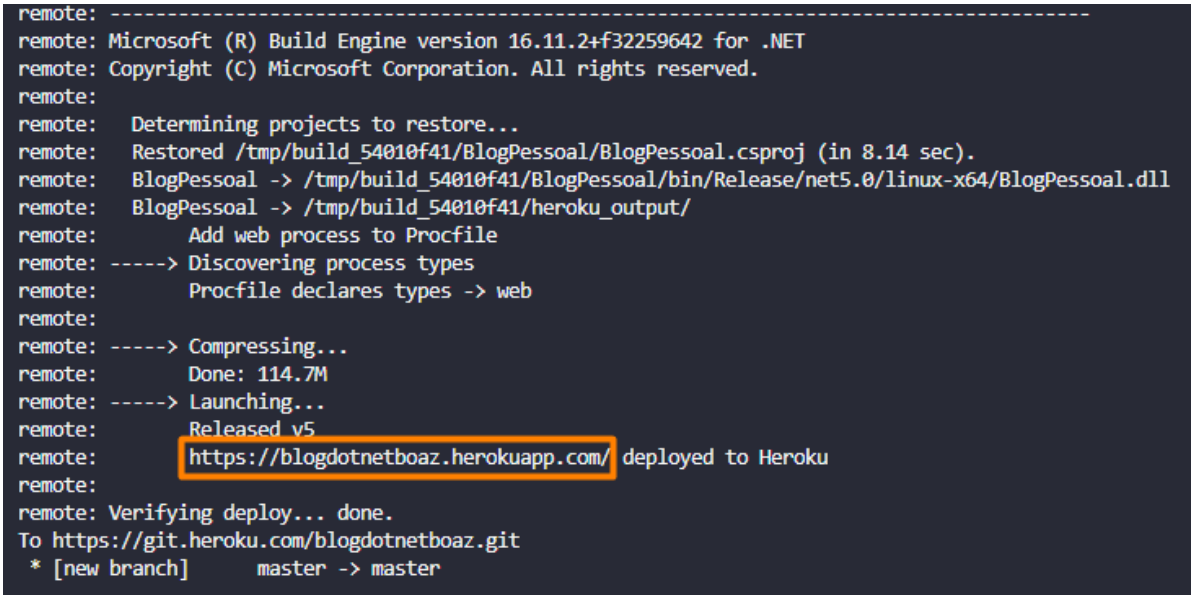
Caso projeto não tenha um repositório GIT

Seguir passos:

```
git init
heroku git:remote -a blogdotnetboaz
git add .
git commit -am "Deploy Heroku"
git push heroku master
```

Tenha atenção no comando acima pois o nome definido após a comando `-a` na segunda linha, é o nome do projeto no Heroku.

Desta maneira sua aplicação será implantada. Validar depois acessando o link fornecido al final da implantação:



Caso sua aplicação não abrir, é possivel validar a causa digitando no terminal o comando abaixo:

```
heroku logs --tail
```

Este comando ira buscar todas as falhas possíveis que poderão causar a não implantação. Corrigir os erros e voltar a implantar.