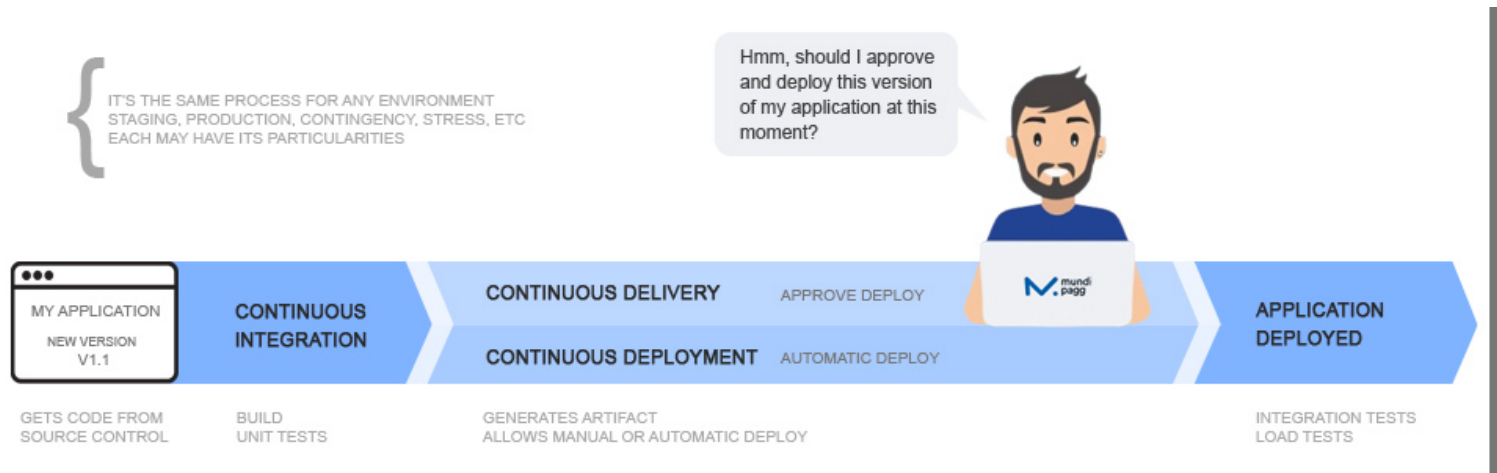


## Continuous Integration, Continuous Delivery e Continuous Deployment de aplicações .NET com AppVeyor

O dia-a-dia de um desenvolvedor costuma ser um tanto agitado. Além da preocupação com o desenvolvimento de uma aplicação que siga exatamente todas as lógicas e regras de negócio ao pé da letra, tem a preocupação com o processo de validação de uma nova *feature* e com o processo de implantação de uma nova versão da sua aplicação (entrega efetiva). Ambos processos são repetitivos e, em muitas empresas, ainda feito de maneira manual e, para manter a qualidade, o tempo para as entregas podem aumentar. Para otimizar este passo no desenvolvimento de software, vamos fazer uma pequena demonstração de como tornar esse processo automático em aplicações .NET com o uso do nosso querido amiguinho *AppVeyor*. Mas antes, precisamos entender alguns conceitos.



Estágios para automatização do processo de Deploy de uma aplicação

**DevOps** é um conjunto de cultura, práticas e ferramentas usadas para aumentar a capacidade e a produtividade de uma empresa na distribuição de suas aplicações rapidamente. Essa combinação aproxima e envolve a área de desenvolvimento com a área de operações (infraestrutura). A implantação de qualquer nível de maturidade de integração contínua é parte do *DevOps*.

**Build Server** é basicamente um servidor que executa a construção da sua aplicação (podendo executar testes unitários) e que gera o artefato final, ou seja, o arquivo ou conjunto de arquivos que devem ser distribuídos para um determinado ambiente.




**Continuous Integration** ou integração contínua é basicamente uma prática de *DevOps* que tem como objetivo integrar toda mudança no código que ocorre no seu repositório (local, *GitHub*, *BitBucket*, *TFS*, etc) com o servidor de *build*. Garantindo que toda modificação na sua aplicação esteja sendo validada nos quesitos de compilação e testes unitários. Os principais objetivos da integração contínua são de encontrar e corrigir *bugs* mais rapidamente, otimizar a qualidade do *software* e reduzir o tempo que leva para validar e lançar novas atualizações de uma aplicação.

**Continuous Delivery** ou entrega contínua é uma etapa que ocorre depois da integração contínua que, após construir e testar a aplicação e garantir que está tudo correto, prepara a aplicação para a distribuição e disponibiliza esse artefato para que essa distribuição seja feita de forma simples,

normalmente com poucos cliques acionados por uma pessoa responsável por definir o momento ideal para o *deploy*.

**Continuous Deployment** ou implantação contínua é bem parecido com a entrega contínua, porém, nesse modo, após o servidor de *build* executar a construção e a validação do código, o pacote também é preparado mas sua distribuição / *deploy* é feita automaticamente no ambiente de desejado.

Entre os benefícios da integração contínua, podemos citar como principais:

 <p><b>Aumento da Produtividade</b></p> <p>Tira do dia-a-dia dos devs e do time de infraestrutura tarefas manuais que ocupam grande parte do tempo e estimula toda sua equipe à prática e cultura de integrar com frequência.</p>	 <p><b>Identificação de <i>Bugs</i></b></p> <p>Com construção e testes feitos com mais frequência, podemos identificar e corrigir erros mais cedo, evitando que fiquem perdidos e mais complexos no futuro.</p>	 <p><b>Distribuição Simples e Rápida</b></p> <p>Com a automatização das tarefas de construção, teste, empacotamento, distribuição automática ou com poucos cliques sua equipe conseguirá entregar muito mais.</p>
--	--	--

**AppVeyor** é uma ferramenta específica para auxiliar na automatização dessas tarefas (integração, entrega e implantação contínua). Ele é bem fácil e prático de ser utilizado e pode ser utilizado nas diversas implementações para plataforma Windows. Integra com os principais versionadores e oferece bastante versatilidade em suas configurações, que podem ser feitas via interface ou por um arquivo de configuração (*yaml*).

Neste artigo vamos fazer uma demonstração de integração, entrega e implantação contínua usando a ferramenta *AppVeyor* com uma aplicação *.NET*.

Projeto de Demonstração no GitHub

<https://github.com/ThiagoBarradas/DotNet.WebApi.AppVeyor.Demo>

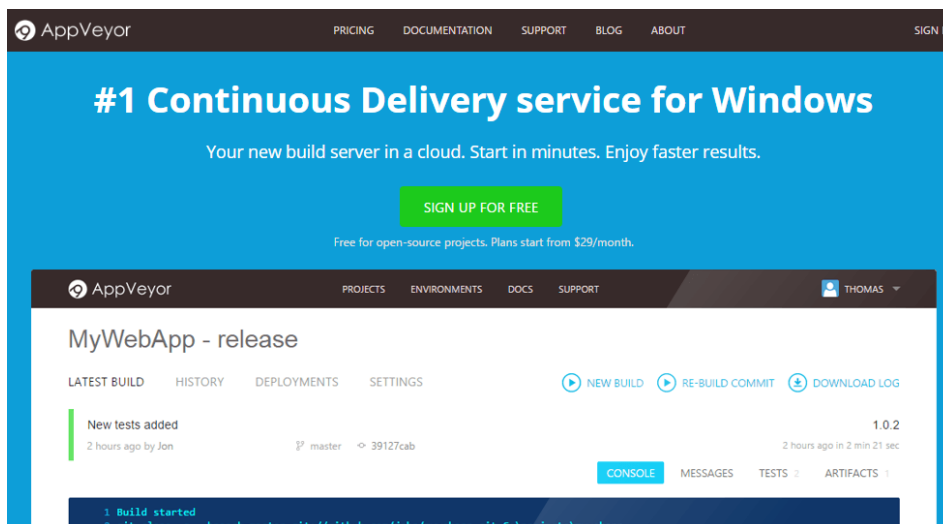
Referência do AppVeyor

<https://www.appveyor.com/docs/>

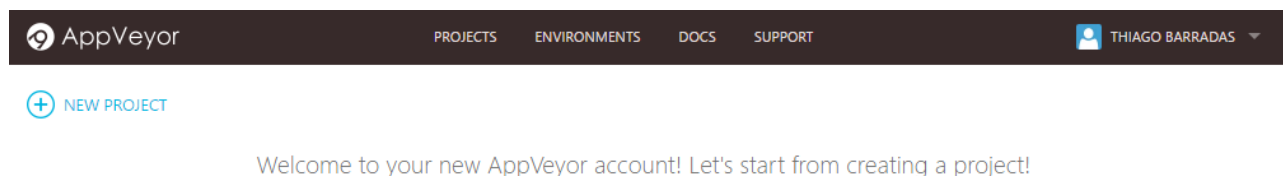
## MÃO NA MASSA!

Primeiramente foi criado um repositório no *GitHub* para o nosso exemplo. Não tem problema se for um projeto já existente. Toda mágica sempre ocorrerá após as modificações no repositório (novos *commits*).

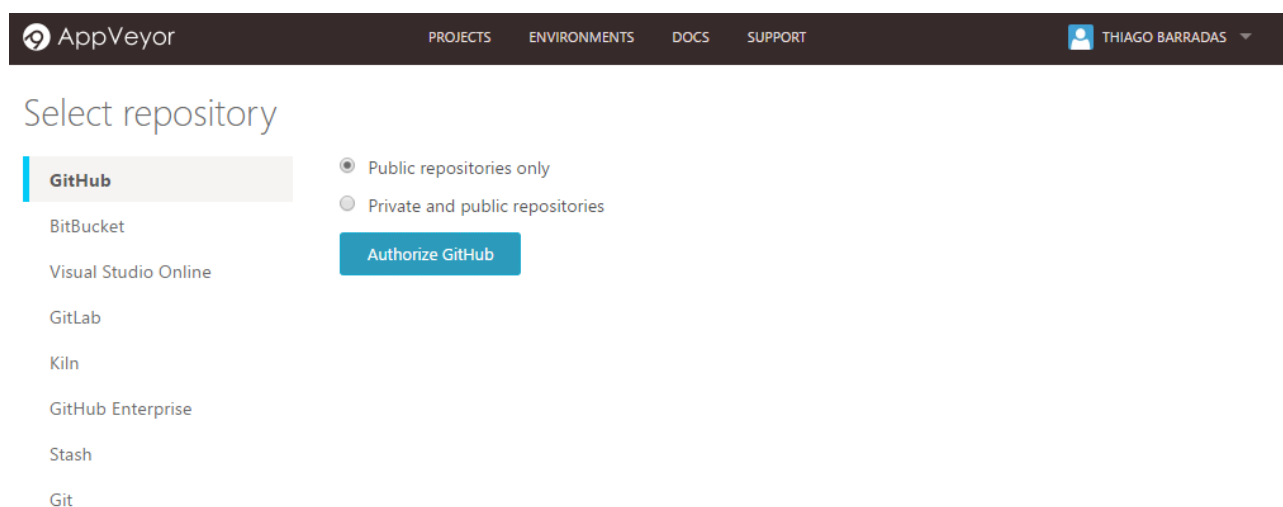
Agora vamos acessar o AppVeyor e criar uma conta para que a integração possa ser feita.



É possível usar a sua conta do *GitHub*, *BitBucket*, *Visual Studio Online* ou criar uma conta com o seu e-mail. Com a conta criada, após o login teremos o seguinte painel:

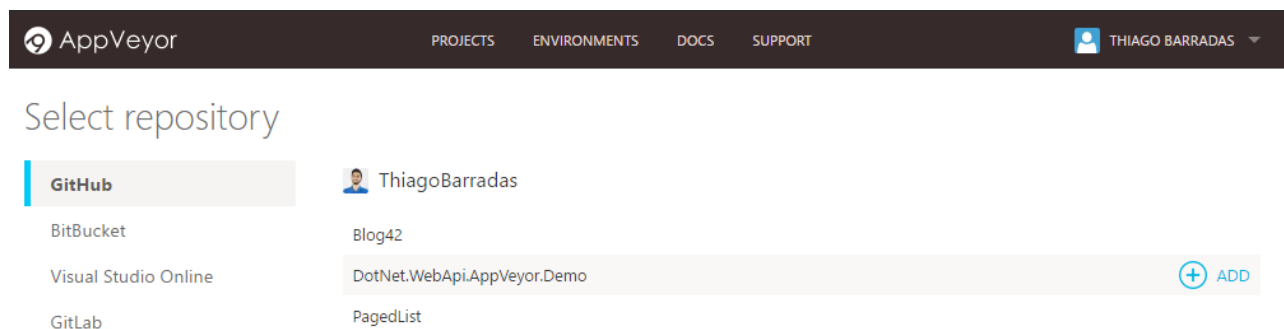


Para adicionar um novo projeto, clique em **New Project**:



É possível integrar com diversos repositórios de código, mas vamos usar o *GitHub*, já que ele foi o escolhido para hospedar o repositório do nosso projeto de exemplo.

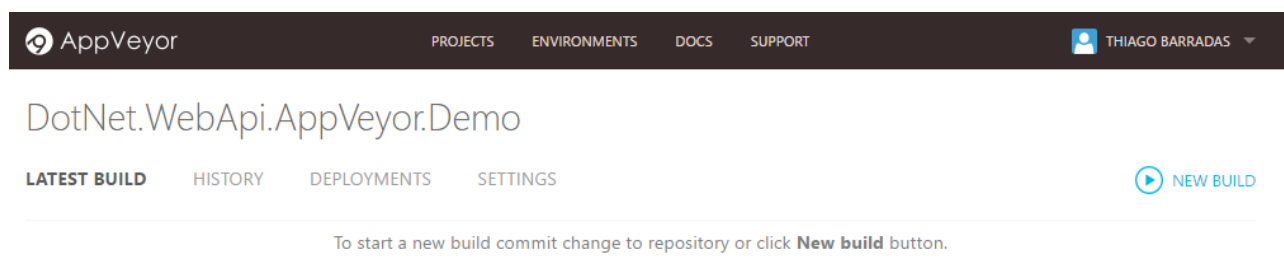
Basta selecionar o tipo de projeto que deseja autorizar o acesso pelo *AppVeyor* e clicar em **Authorize GitHub**, você será redirecionado para uma página de solicitação de permissão do *GitHub*, e após autorizar, no *AppVeyor* já será possível visualizar os seus projetos:



Após feito isso, clique em **Add** para adicionar o projeto, botão que aparece no projeto ao passar o mouse por cima do link.

Após adicionar, iremos direto para a página do projeto, que também pode ser acessada clicando em **Projects** no menu superior. Nessa opção listamos todos os projetos integrados a sua conta.

Abaixo a página de detalhamento do projeto:



Podemos “forçar” o *AppVeyor* a executar o *build* referente ao último *commit* clicando em **New Build**, ou, automaticamente o *build* será executado para novos *commits*.

Primeira etapa concluída. Agora vamos criar uma aplicação *WebApi* em *.NET* para o nosso exemplo. Lembrando que todo esse processo é basicamente o mesmo para os diversos tipos de aplicações, podendo ter algumas diferenças na etapa de configuração do pacote, do ambiente e da distribuição.

A aplicação criada tem apenas um *endpoint* para exemplificar o resultado final.

```
10 [HttpGet]
11 [Route("")]
12 public HttpResponseMessage Home()
13 {
14     Message message = new Message();
15
16     message.Title = "Hey Jude";
17     message.Content = "don't make it bad";
18
19     return Request.CreateResponse(HttpStatusCode.OK, message);
20 }
```

Ao rodar a aplicação o resultado da chamada do *endpoint* seria o seguinte *Json*:

```
localhost:64505
{
  "title": "Hey Jude",
  "content": "don't make it bad"
}
```

Associei o projeto da *WebApi* com o meu repositório do *GitHub* e subi o código. O *AppVeyor* automaticamente identificou o novo *commit* e já executou o seu *build*.

The screenshot shows the AppVeyor interface for the project 'DotNet.WebApi.AppVeyor.Demo'. The build is titled 'Creates a webapi with simple endpoint' and is version 1.0.0. It was built 2 minutes ago by Thiago Barradas on the master branch, commit d607ba2b. The build status is 'Succeeded' and it completed in 12 seconds. The console output shows the following steps:

```
1 Build started
2 git clone -q --branch=master https://github.com/ThiagoBarradas/DotNet.WebApi.AppVeyor.Demo.git C:\projects\dotnet-webapi-appveyor-demo
3 git checkout -qf d607ba2b2a89db60a9b9f2b7c5c569cb29606ad6
4 "C:\Program Files (x86)\MSBuild\12.0\Bin\MSBuild.exe" "C:\projects\dotnet-webapi-appveyor-demo\DotNet.WebApi.AppVeyor.Demo.sln" /verbosity:minimal /logger:"C:\Program Files\AppVeyor\BuildAgent\Appveyor.MSBuildLogger.dll"
5 Microsoft (R) Build Engine version 12.0.40629.0
6 [Microsoft .NET Framework, version 4.0.30319.42000]
7 Copyright (C) Microsoft Corporation. All rights reserved.
8
9 Restoring NuGet packages...
10 To prevent NuGet from downloading packages during build, open the Visual Studio Options dialog, click on the Package Manager node and uncheck 'Allow NuGet to download missing packages'.
11 Feeds used:
12 C:\Users\appveyor\.nuget\packages\
13 https://www.nuget.org/api/v2
```

Podemos verificar que o *build* foi concluído com sucesso. Para exemplificar um erro no momento da construção da aplicação, fiz um novo *commit* com um erro no código para forçar o erro na construção.

The screenshot shows the AppVeyor interface for the project 'DotNet.WebApi.AppVeyor.Demo'. The build is titled 'Forces broken building' and is version 1.0.2. It was built 2 minutes ago by Thiago Barradas on the master branch, commit e5e93aab. The build status is 'Failed' and it completed in 9 seconds. The console output shows the following steps:

```
1 Build started
2 git clone -q --branch=master https://github.com/ThiagoBarradas/DotNet.WebApi.AppVeyor.Demo.git C:\projects\dotnet-webapi-appveyor-demo
3 git checkout -qf e5e93aab3d0a9c6f49540e6646fa51ce098d8a91
4 "C:\Program Files (x86)\MSBuild\12.0\Bin\MSBuild.exe" "C:\projects\dotnet-webapi-appveyor-demo\DotNet.WebApi.AppVeyor.Demo.sln" /verbosity:minimal /logger:"C:\Program Files\AppVeyor\BuildAgent\Appveyor.MSBuildLogger.dll"
5 Microsoft (R) Build Engine version 12.0.40629.0
6 [Microsoft .NET Framework, version 4.0.30319.42000]
7 Copyright (C) Microsoft Corporation. All rights reserved.
8
9 Restoring NuGet packages...
10 To prevent NuGet from downloading packages during build, open the Visual Studio Options dialog, click on the Package Manager node and uncheck 'Allow NuGet to download missing packages'.
11 Feeds used:
12 C:\Users\appveyor\.nuget\packages\
```

Agora, além de corrigir o código, vou adicionar um simples teste unitário:

```
15 [TestMethod]
16 public void Home_Should_Return_OK()
17 {
18     // Arrange
19     HomeController controller = this.GenerateController<HomeController>();
20
21     // Act
22     var result = controller.Home();
23
24     // Assert
25     Assert.IsNotNull(result);
26     Assert.AreEqual(result.StatusCode, HttpStatusCode.OK);
27 }
```

Após subir as novas implementações, no *AppVeyor* podemos conferir o *build* do projeto e o resultado do teste implementado.

The screenshot shows the AppVeyor dashboard for the project 'DotNet.WebApi.AppVeyor.Demo'. The build is titled 'Implements a simple unit test' and was triggered 2 minutes ago by Thiago Barradas. The build is successful, with a console output showing the build process and test execution. The console output includes the following text:

```
1 Build started
2 git clone -q --branch=master https://github.com/ThiagoBarradas/DotNet.WebApi.AppVeyor.Demo.git C:\projects\dotnet-webapi-
3 appveyor-demo
4 git checkout -qf 763bce44895f4c423edad92329b4ac9b53b94be5
5 C:\Program Files (x86)\MSBuild\14.0\bin\MSBuild.exe "/c:\projects\dotnet-webapi-appveyor-demo\DotNet.WebApi.AppVeyor.Demo\1"
```

O console informará o resultado além do processo de *build*, o resultado da execução dos testes.

The screenshot shows the console output of the build process, specifically the test execution results. The output includes the following text:

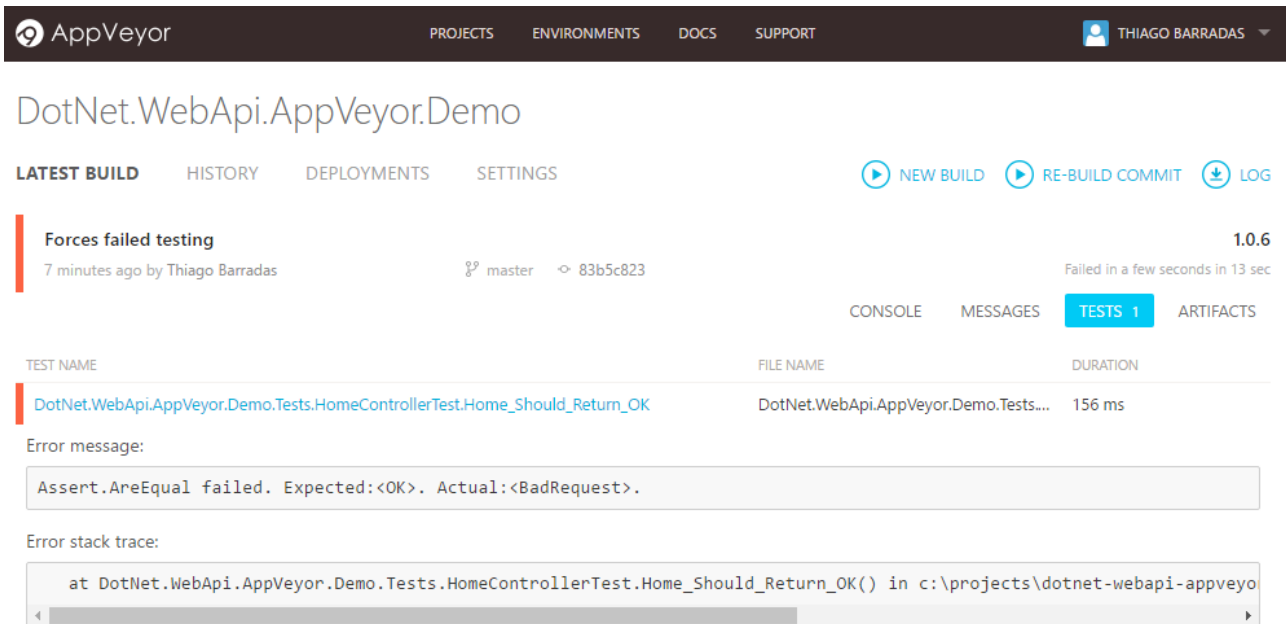
```
70 DotNet.WebApi.AppVeyor.Demo.Tests -> C:\projects\dotnet-webapi-appveyor-
71 demo\DotNet.WebApi.AppVeyor.Demo.Tests\bin\Debug\DotNet.WebApi.AppVeyor.Demo.Tests.dll
72 Discovering tests...OK
73 vstest.console /logger:Appveyor "C:\projects\dotnet-webapi-appveyor-
74 demo\DotNet.WebApi.AppVeyor.Demo.Tests\bin\Debug\DotNet.WebApi.AppVeyor.Demo.Tests.dll"
75 Microsoft (R) Test Execution Command Line Tool Version 14.0.25420.1
76 Copyright (c) Microsoft Corporation. All rights reserved.
77 Starting test execution, please wait...
78 Passed Home_Should_Return_OK
79 Total tests: 1. Passed: 1. Failed: 0. Skipped: 0.
80 Test Run Successful.
81 Test execution time: 0.4558 Seconds
82 Build success
```

Também podemos visualizar os testes executados na aba **Tests**:

The screenshot shows the AppVeyor dashboard for the project 'DotNet.WebApi.AppVeyor.Demo'. The build is titled 'Implements a simple unit test' and was triggered 12 minutes ago by Thiago Barradas. The build is successful, with a console output showing the build process and test execution. The console output includes the following text:

```
TEST NAME FILE NAME DURATION
DotNet.WebApi.AppVeyor.Demo.Tests.HomeControllerTest.Home_Should_Return_OK DotNet.WebApi.AppVeyor.Demo.Tests... 97 ms
```

Forçando a falha dos testes, essa mesma tela ficaria da seguinte forma:



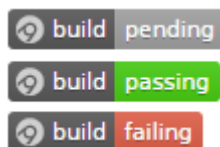
The screenshot shows the AppVeyor interface for a project named 'DotNet.WebApi.AppVeyor.Demo'. The top navigation bar includes 'PROJECTS', 'ENVIRONMENTS', 'DOCS', 'SUPPORT', and a user profile for 'THIAGO BARRADAS'. The main content area shows the 'LATEST BUILD' with a status of 'Forces failed testing' at version '1.0.6', which failed 'in a few seconds in 13 sec'. Below this, there are tabs for 'CONSOLE', 'MESSAGES', 'TESTS 1', and 'ARTIFACTS'. The 'TESTS' tab is active, displaying a table with columns for 'TEST NAME', 'FILE NAME', and 'DURATION'. One test is listed: 'DotNet.WebApi.AppVeyor.Demo.Tests.HomeControllerTest.Home\_Should\_Return\_OK' with a duration of '156 ms'. Below the table, the 'Error message' is 'Assert.AreEqual failed. Expected:<OK>. Actual:<BadRequest>.' and the 'Error stack trace' shows the error occurred in 'c:\projects\dotnet-webapi-appveyo'.

Nesse ponto já atingimos o primeiro nível de maturidade. A integração contínua (*Continuous Integration*), construção da aplicação e testes após modificações em nosso projeto.

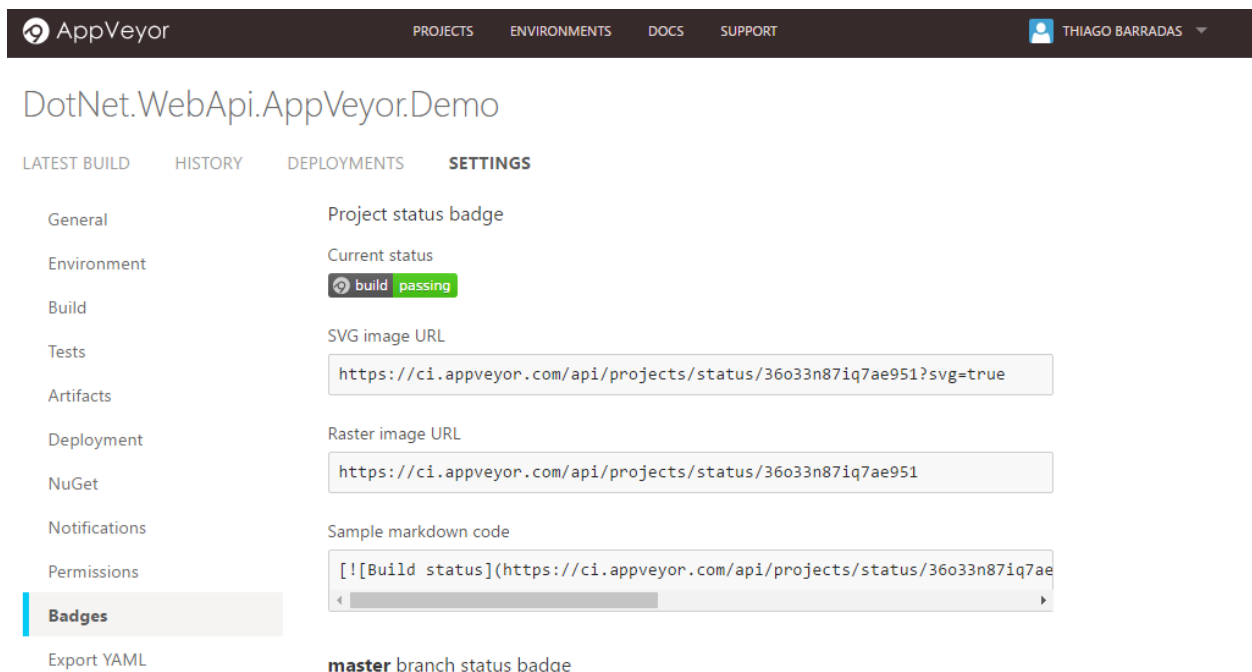
É possível configurar integrações com serviços de notificação como E-mail, Slack, HipChat, Webhooks, entre outros, para notificá-lo sobre o status de novos *builds*.

Para ficar mais bacana ainda podemos adicionar *Badges* fornecidos pelo próprio *AppVeyor* para indicar o último status (por exemplo incluir no Readme do seu projeto no GitHub).

Exemplos dos Badges:



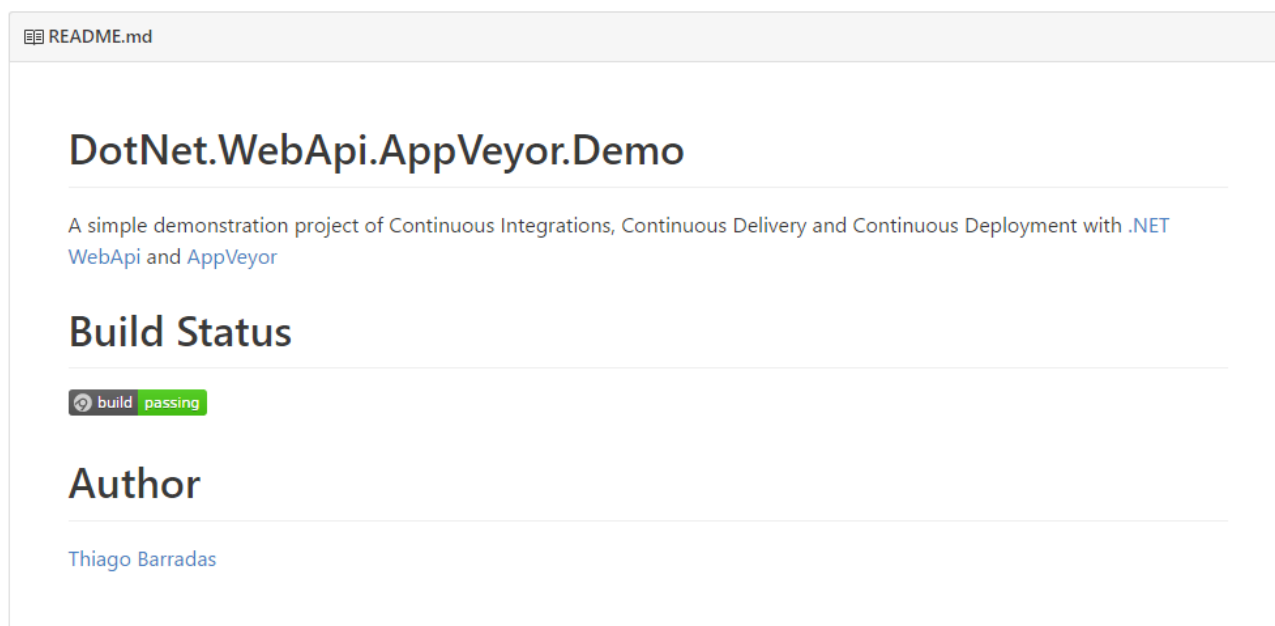
Para obter o código para a inclusão dos Badges basta acessar Settings > Badges:



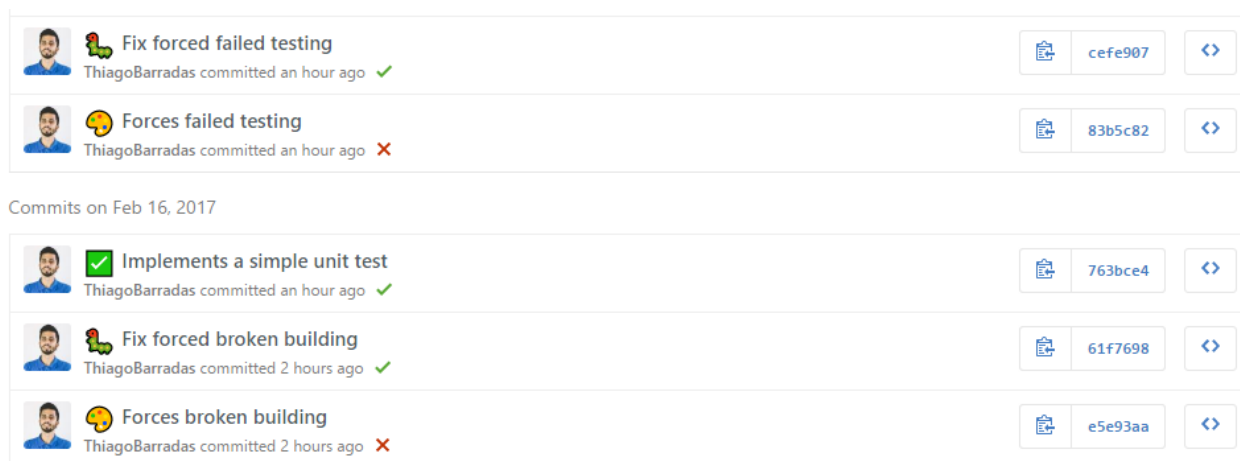
The screenshot shows the 'Settings' page for the 'DotNet.WebApi.AppVeyor.Demo' project, specifically the 'Badges' section. The left sidebar has 'Badges' selected. The main content area is titled 'Project status badge' and shows the 'Current status' as 'build passing'. Below this, there are input fields for 'SVG image URL' (https://ci.appveyor.com/api/projects/status/36033n87iq7ae951?svg=true), 'Raster image URL' (https://ci.appveyor.com/api/projects/status/36033n87iq7ae951), and 'Sample markdown code' ([[Build status]](https://ci.appveyor.com/api/projects/status/36033n87iq7ae951)). At the bottom, there is a section for 'master branch status badge'.



Exemplo aplicado ao *Readme* do projeto no *GitHub*.

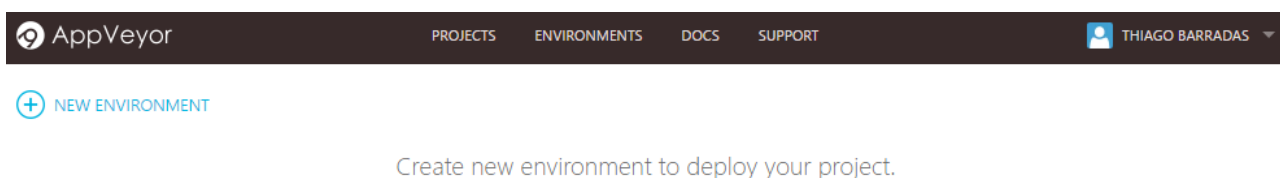


Também podemos ver o status de cada *commit* em que o *AppVeyor* fez a construção e a execução dos testes a partir da listagem de *commits* do *GitHub*.



Para configurar a entrega contínua (*Continuous Delivery*) precisamos ter o ambiente que receberá o pacote configurado, e fazer com que o *AppVeyor* prepare o pacote, disponibilizando o artefato para a distribuição.

Vamos configurar nosso ambiente. Acesse **Environments** no menu superior do *AppVeyor*.



Clique em **New Environment** para criar uma nova configuração para um ambiente:



## New environment

Provider

- Select deployment provider -
- Select deployment provider -
- Web Deploy
- FTP
- Amazon S3
- NuGet
- Azure WebJob
- Azure Blob Storage
- Azure Cloud Service
- AppVeyor Agent
- SQL Server Database
- Webhook
- GitHub Releases
- BinTray

Existem diversas formas para integrar em diversos ambientes como *Azure*, *Amazon*, etc. Para nosso exemplo, supondo que aplicação estará em uma máquina qualquer, em qualquer lugar, em qualquer servidor, vamos usar o *AppVeyor Agent*, um serviço que rodará nas máquinas que hospedarão a aplicação (podendo ser mais de uma máquina).

Então selecionamos *AppVeyor Agent*, damos um nome para o nosso ambiente e inserimos algumas configurações:

## New environment

Provider

AppVeyor Agent

Environment name

AppDemo Production

Provider settings

DotNet.WebApi.AppVeyor.Demo.deploy_website	true
DotNet.WebApi.AppVeyor.Demo.website_name	DotNet.WebApi.AppVeyor.Demo
DotNet.WebApi.AppVeyor.Demo.remove_files	true
DotNet.WebApi.AppVeyor.Demo.hostname	*
DotNet.WebApi.AppVeyor.Demo.port	8080
DotNet.WebApi.AppVeyor.Demo.protocol	http
DotNet.WebApi.AppVeyor.Demo.apppool_name	DotNet.WebApi.AppVeyor.Demo

As configurações devem ter o padrão **{artifact\_name}.{config}**.

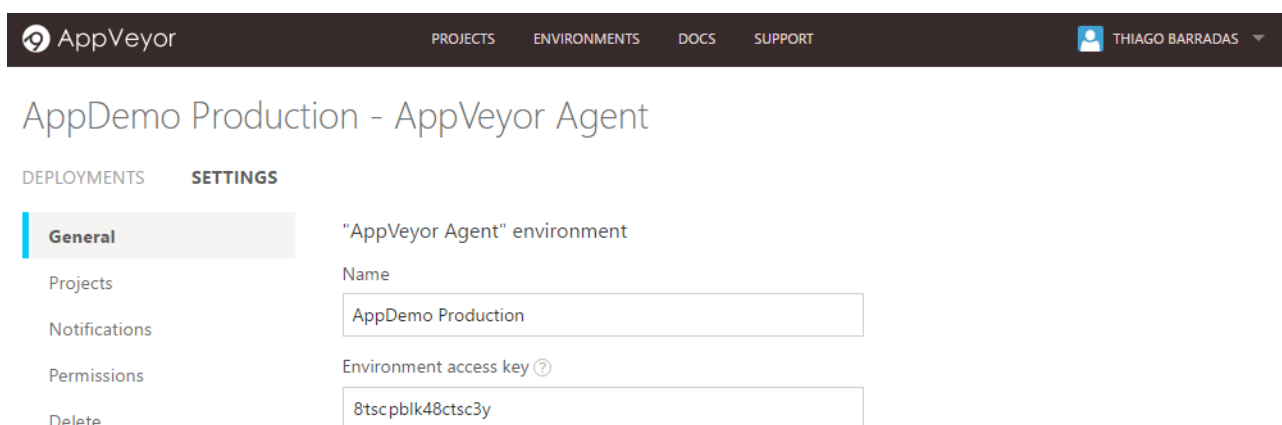
Nesse caso o artefato que vamos gerar, por padrão, usará o nome do projeto:  
**DotNet.WebApi.AppVeyor.Demo.**

As configurações setadas para permitir o *deploy* de uma aplicação *Web* no *IIS* foram:

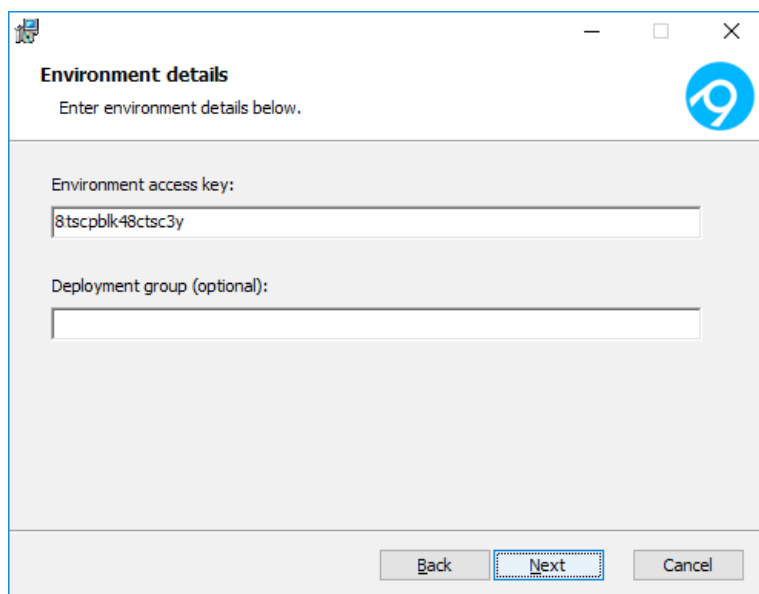
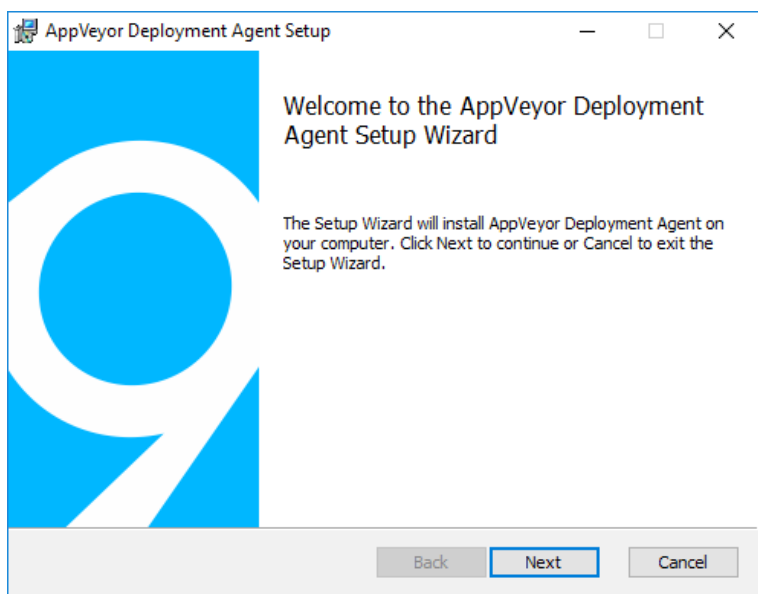
- **deploy\_website**: Informa se o *deploy* será de uma aplicação *web*;
- **website\_name**: Nome da aplicação no *IIS*;
- **remove\_files**: Informa se os arquivos serão removidos antes do *deploy* ou se ocorrerá apenas atualização do que foi modificado;
- **hostname**: Caso aplicação no *IIS* esteja associada a algum *hostname* o mesmo deverá ser informado aqui. Essa é a única aplicação que vai rodar nesse servidor, então deixamos configurado para qualquer *hostname*;
- **port**: A porta de comunicação da aplicação;
- **protocol**: Define o protocolo da aplicação;
- **appool\_name**: O nome do *pool* de aplicação que a aplicação pertence.

Para configurar mais de uma aplicação no mesmo ambiente, basta incluir as configurações dentro do ambiente utilizando sempre como referência o nome do artefato que será gerado.

Após a criação, seremos direcionados para o detalhe do novo ambiente, que conterà uma chave que identifica nosso ambiente.

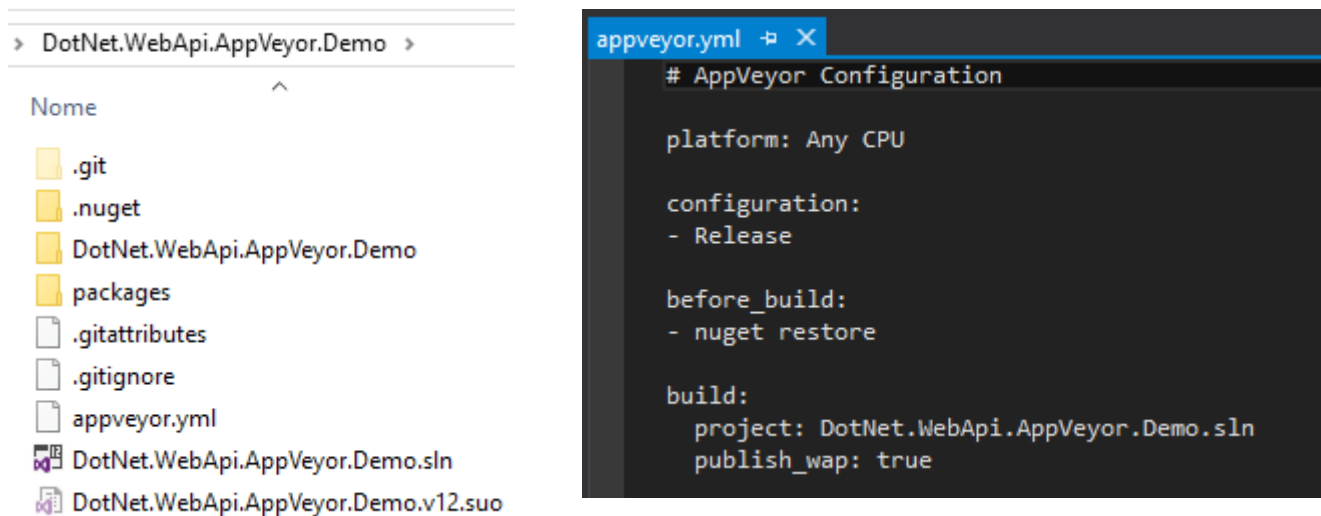


Para instalar o *AppVeyor Agent* nas máquinas, basta fazer o download do instalador e executar nas máquinas desejadas. Durante a instalação será solicitado o *Environment Access Key* e informaremos a chave do nosso ambiente:



Após a instalação, restará apenas configurar nossa aplicação para que o pacote seja preparado.

Para configurar a geração do pacote da aplicação, podemos fazer via interface gráfica no próprio painel do *AppVeyor* (em *Settings* no detalhe do projeto), ou, incluir um arquivo de configuração **appveyor.yml** no diretório raiz da aplicação (junto com a *solution*).



Na primeira linha temos apenas um comentário. Abaixo configuramos, assim como no *Visual Studio*, a plataforma (**platform**) e qual configuração (**configuration**) queremos usar para gerar o pacote. Podemos informar mais de uma configuração, e pra cada configuração, um novo *build* será feito e um novo artefato gerado.

Na opção **before\_build** (evento que ocorre antes do *build*) foi executado o comando para restaurar os pacotes instalados via *nuget*. Para que isso funcione é importante que seja habilitado no *Visual Studio* a opção “*Enabled nuget package restore*” clicando com o botão direito em cima da solução e habilitando.

No **build** configuramos o projeto que vai ser construído, neste caso foi apontado para a *solution* para que os testes sejam inclusos e automaticamente detectados pelo *AppVeyor*.

Em **publish\_wap** informamos o valor *true* para informar que um pacote de uma aplicação *web* deve ser criado (ou seja, esse comando é o responsável por gerar o artefato).

Cuidado ao escrever o seu arquivo de configuração *appveyor.yml* pois ele não funciona com tabulação, apenas espaços.

Após incluir esse arquivo basta fazer o *commit* e o pacote estará pronto e disponível para a distribuição.

The screenshot shows the AppVeyor interface for a build. At the top, there's a navigation bar with 'PROJECTS', 'ENVIRONMENTS', 'DOCS', and 'SUPPORT'. The user 'THIAGO BARRADAS' is logged in. The main heading is 'DotNet.WebApi.AppVeyor.Demo'. Below it, there are tabs for 'LATEST BUILD', 'HISTORY', 'DEPLOYMENTS', and 'SETTINGS'. On the right, there are buttons for 'NEW BUILD', 'RE-BUILD COMMIT', 'DEPLOY', and 'LOG'. The build title is 'Integrates with AppVeyor - Continuous Delivery' with version '1.0.9', built 'a minute ago by Thiago Barradas' on the 'master' branch with commit '38e636e1'. Below the build info, there are tabs for 'CONSOLE', 'MESSAGES', 'TESTS', and 'ARTIFACTS'. The console output shows the following commands and their results:

```
1 Build started
2 git clone -q --branch=master https://github.com/ThiagoBarradas/DotNet.WebApi.AppVeyor.Demo.git C:\projects\dotnet-webapi-appveyor-demo
3 git checkout -qf 38e636e12379f6bc1508979654b6ed36782b0e68
4 nuget
5 NuGet Version: 3.5.0.1938
6 usage: NuGet <command> [args] [options]
7 Type 'NuGet help <command>' for help on a specific command.
8
9 Available commands:
10
11 add
    Adds the given package to a hierarchical source. http sources are not supported. For more info, goto
```

Podemos visualizar no menu acima do console que além do teste identificado, temos um artefato disponível.

This screenshot shows the 'ARTIFACTS' tab selected in the AppVeyor interface. The build information is the same as in the previous screenshot. The 'ARTIFACTS' tab shows a table with the following data:

FILE NAME	TYPE	SIZE	DEPLOYMENT NAME
DotNet.WebApi.AppVeyor.Demo.zip	WebDeploy package	500 KB	DotNet.WebApi.AppVey...

Para efetuar o *deploy* desse pacote basta clicar no botão **Deploy**.

As permissões para os usuários na sua conta do *AppVeyor* podem ser definidas para, entre diversas opções, informar quem tem permissão para efetuar *deploy*.

Após clicar no botão *Deploy*, escolhemos em qual ambiente queremos subir o pacote.

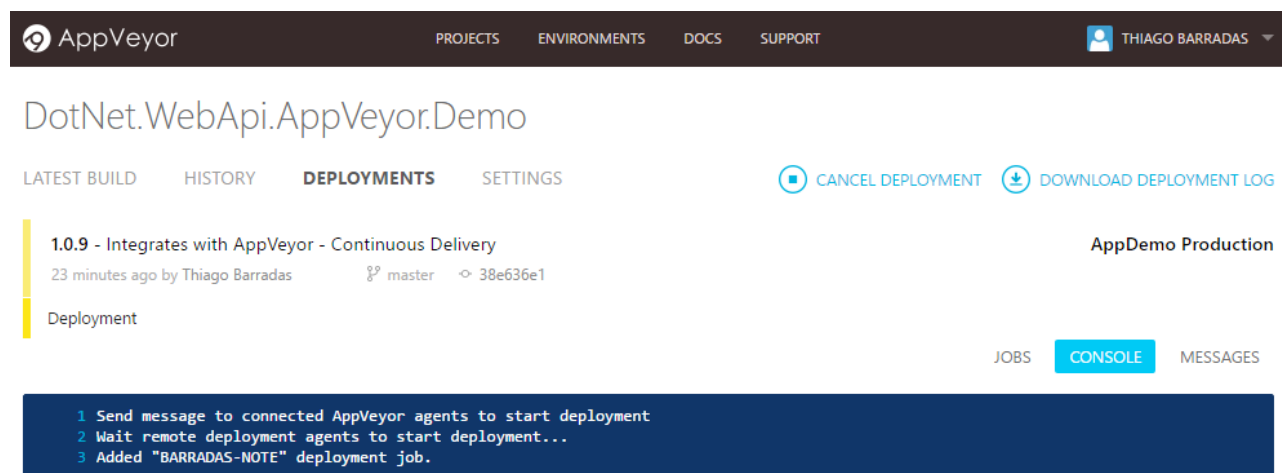
Após a escolha, para efetivar a operação, clique novamente em *Deploy*.

The screenshot shows the 'Deploy' dialog box in AppVeyor. The title is 'New deployment of "DotNet.WebApi.AppVeyor.Demo"'. It displays the following information:

- Version: 1.0.9
- Branch: master
- Environment: AppDemo Production - AppVeyor Agent (selected in a dropdown menu)

At the bottom, there is a blue 'Deploy' button.

O *AppVeyor* identificará as máquinas que o agente está instalado com a chave do ambiente escolhido e iniciará os *Jobs* referentes a cada máquina.



The screenshot shows the AppVeyor dashboard for a project named 'DotNet.WebApi.AppVeyor.Demo'. The user is 'THIAGO BARRADAS'. The 'DEPLOYMENTS' tab is active, showing a deployment for version '1.0.9' titled 'Integrates with AppVeyor - Continuous Delivery', made 23 minutes ago by Thiago Barradas. The deployment is for the 'AppDemo Production' environment. Below the deployment details, there are buttons for 'CANCEL DEPLOYMENT' and 'DOWNLOAD DEPLOYMENT LOG'. A console log is visible, showing three steps: 1. Send message to connected AppVeyor agents to start deployment, 2. Wait remote deployment agents to start deployment..., and 3. Added "BARRADAS-NOTE" deployment job.

Ao término, a aplicação estará atualizada.

Como exemplo, também atualizamos as informações retornadas do nosso *endpoint* de exemplo. Após o *deploy*, temos a aplicação atualizada corretamente.

Antes:

```
localhost:8080
{
  "title": "Hey Jude",
  "content": "don't make it bad"
}
```

Depois:

```
localhost:8080
{
  "title": "Hey Jude",
  "content": "don't be afraid"
}
```

Nesse ponto estamos com entrega contínua (*Continuous Delivery*) implementada em nossa aplicação.

Neste ponto de maturidade você tem uma aplicação que após qualquer modificação, te informa sobre o *status* do *build*, a execução dos testes e o preparo do pacote, disponibilizado para o *deploy* com apenas poucos cliques.

Tudo isso vai auxiliar a sua equipe a produzir mais. Identificar erros mais rápido, logo, corrigir mais rápido, garantir a simplicidade no processo de *deploy*, assim como no processo de *rollback* de possíveis erros (o processo de *rollback* é igual ao *deploy* de uma nova atualização, sendo necessário apenas escolher a versão que deseja subir).

Para implementar o *continuous deployment* (implantação contínua) é necessário muito cuidado e maturidade da equipe, já que toda modificação irá, imediatamente, afetar o seu ambiente após o *build* e a execução dos testes acontecerem com sucesso.

Para configurar o famoso *deploy* automático dessa aplicação basta modificar o seu arquivo de configuração *appveyor.yml*.

```
appveyor.yml  X
# AppVeyor Configuration

platform: Any CPU

configuration:
- Release

before_build:
- nuget restore

build:
  project: DotNet.WebApi.AppVeyor.Demo.sln
  publish_wap: true

deploy:
  provider: Environment
  name: AppDemo Production
```

Adicionado a configuração **deploy**, informando qual o **provider** (Environment) e o nome (**name**) do *provider* que escolhemos (baseado no ambiente que criamos anteriormente).

Após esse *commit* (também modificamos o conteúdo do nosso *endpoint* de exemplo) o *deploy* ocorreu automaticamente. Ou seja, fizemos o *commit* e *push* de nosso projeto no *GitHub* e pronto. A aplicação foi construída, testada e sua implantação foi feita automaticamente para o ambiente escolhido.

Após um *commit* com o *Continuous Deployment* configurado, o console informará também esta etapa.

```
demo \DotNet.WebApi.AppVeyor.Demo.Tests\bin\Release\DotNet.WebApi.AppV
284 Microsoft (R) Test Execution Command Line Tool Version 14.0.25420.1
285 Copyright (c) Microsoft Corporation. All rights reserved.
286
287 Starting test execution, please wait...
288 Passed   Home_Should_Return_OK
289
290 Total tests: 1. Passed: 1. Failed: 0. Skipped: 0.
291 Test Run Successful.
292 Test execution time: 0.4657 Seconds
293 Uploading artifacts...
294 [1/1] DotNet.WebApi.AppVeyor.Demo.zip (511,595 bytes)...100%
295 Deploying using Environment provider
296 Started deployment to AppDemo Production environment
297 Build success
```

Antes:

```
localhost:8080
{
  "title": "Hey Jude",
  "content": "don't be afraid"
}
```

Depois:

```
localhost:8080
{
  "title": "Hey Jude",
  "content": "don't let me down"
}
```

Pronto. O *deploy* dessa aplicação está totalmente automatizado.

O *AppVeyor* é uma ótima ferramenta e fácil de implementar. Ela oferece diversos recursos não apresentados nesse artigo. É possível executar diversos *workflows*, antes e depois dos eventos (*build*, *test*, *deploy*, etc), configurar *scripts*, preparar o ambiente instalando serviços, configurar banco de dados, acionar testes de integração, testes de carga, notificações, entre outros.

Espero que esse artigo sirva para incentivar e auxiliar a todos que precisam de uma ferramenta prática para automatizar o processo de entrega para plataforma *Windows*.

Links:

Projeto de Demonstração no GitHub

<https://github.com/ThiagoBarradas/DotNet.WebApi.AppVeyor.Demo>

Referência do AppVeyor

<https://www.appveyor.com/docs/>