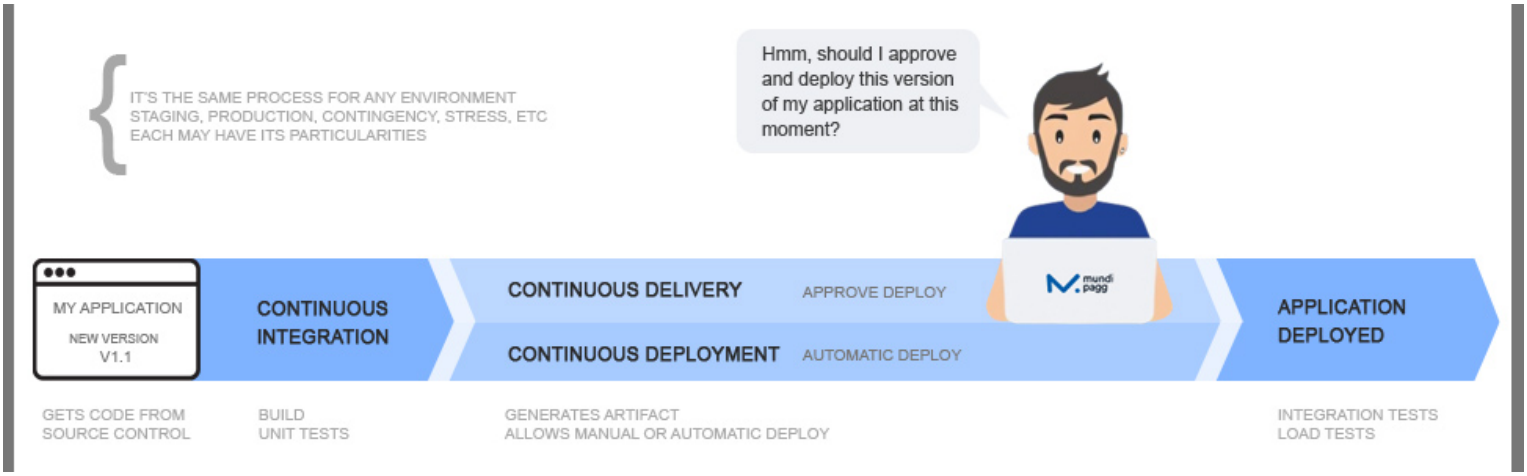
Continuous Integration, Continuous Delivery e Continuous Deployment de aplicações .NET com AppVeyor

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 demostração 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:

		
Aumento da Produtividade 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.	**Identificação de *Bugs*** Com construção e testes feitos com mais frequência, podemos identificar e corrigir erros mais cedo, evitando que fiquem perdidos e mais complexos no futuro.	**Distribuição Simples e Rápida** Com a automatização das tarefas de construção, teste, empacotamento, distribuição automática ou com poucos cliques sua equipe conseguirá entregar muito mais.

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 (*yml*).

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

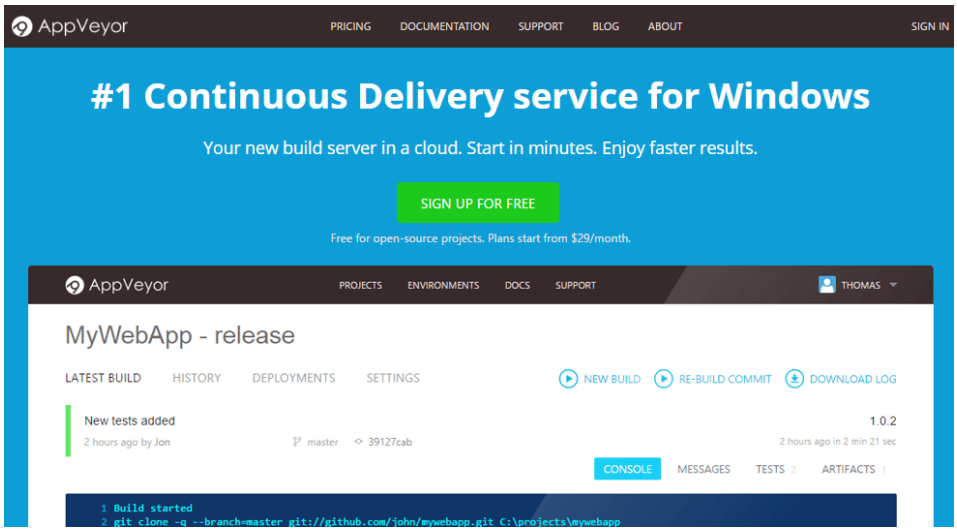
Projeto de Demonstração no GitHub
https://github.com/ThiagoBarradas/DotNet.WebApi.AppVeyor.Demo

Referência do AppVeyor
https://www.appveyor.com/docs/

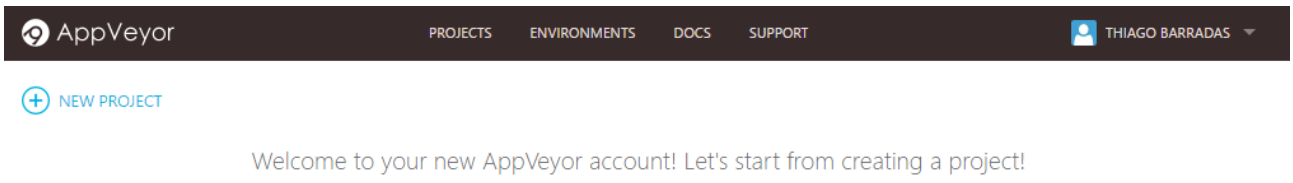
MÃO NA MASSA!

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*).

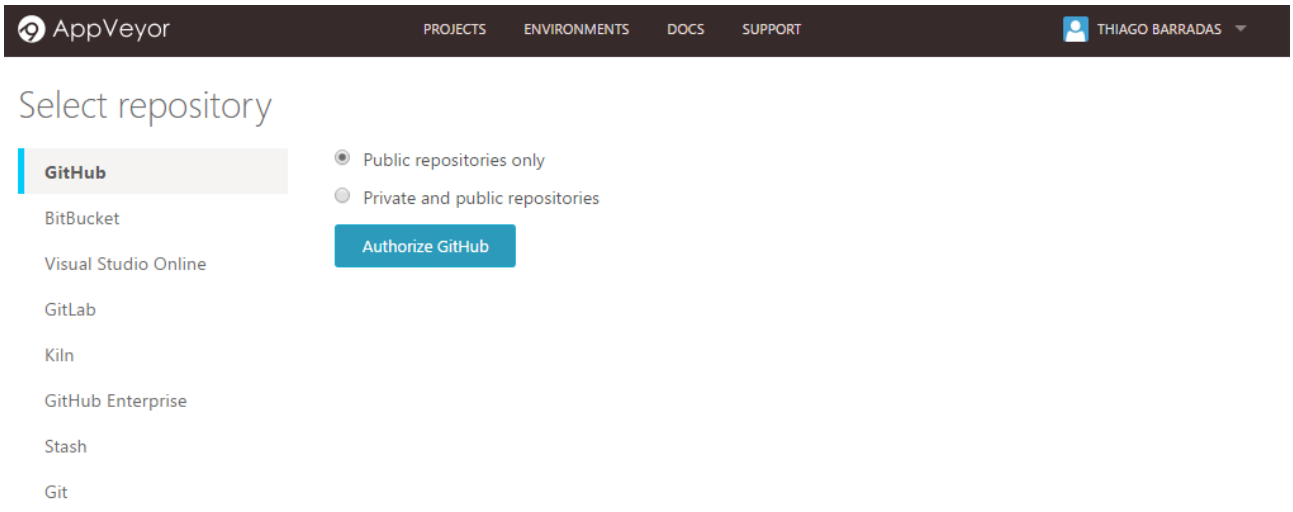
Agora vamos acessaar o AppVeyor e criar uma conta para que a integração possa ser feita.



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:

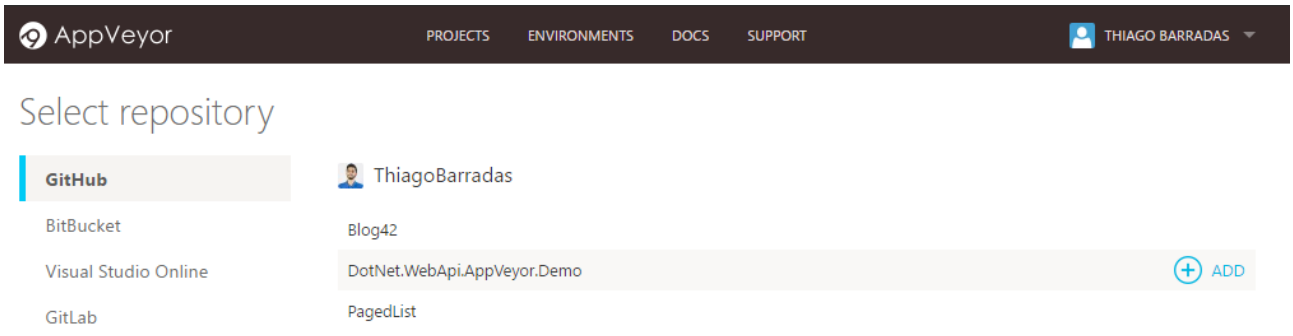


Para adicionar um novo projeto, clique em **New Project**:



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.

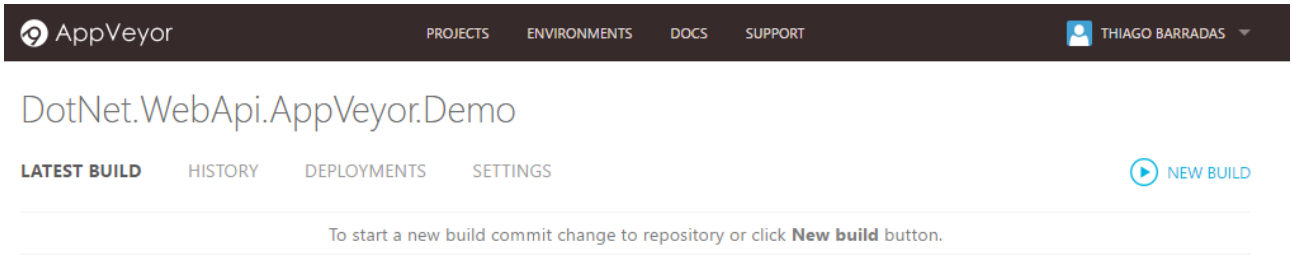
Basta selecionar o tipo de projeto que deseja autorizar o acesso pelo *AppVeyor* e clicar em **Authorize GitHub**, você será redirecionado para uma página de solicitação de permissão do *GitHub*, e após autorizar, no *AppVeyor* já será possível visualizar os seus projetos:



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:



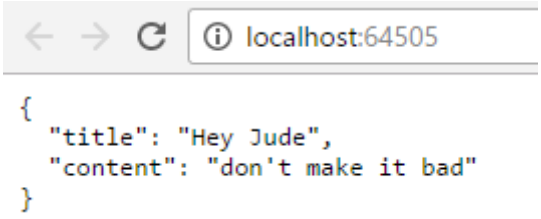
Podemos “forçar” o *AppVeyor* a executar o *build* referente ao último *commit* clicando em **New Build**, ou, automaticamente o *build* será executado para novos *commits*.

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.

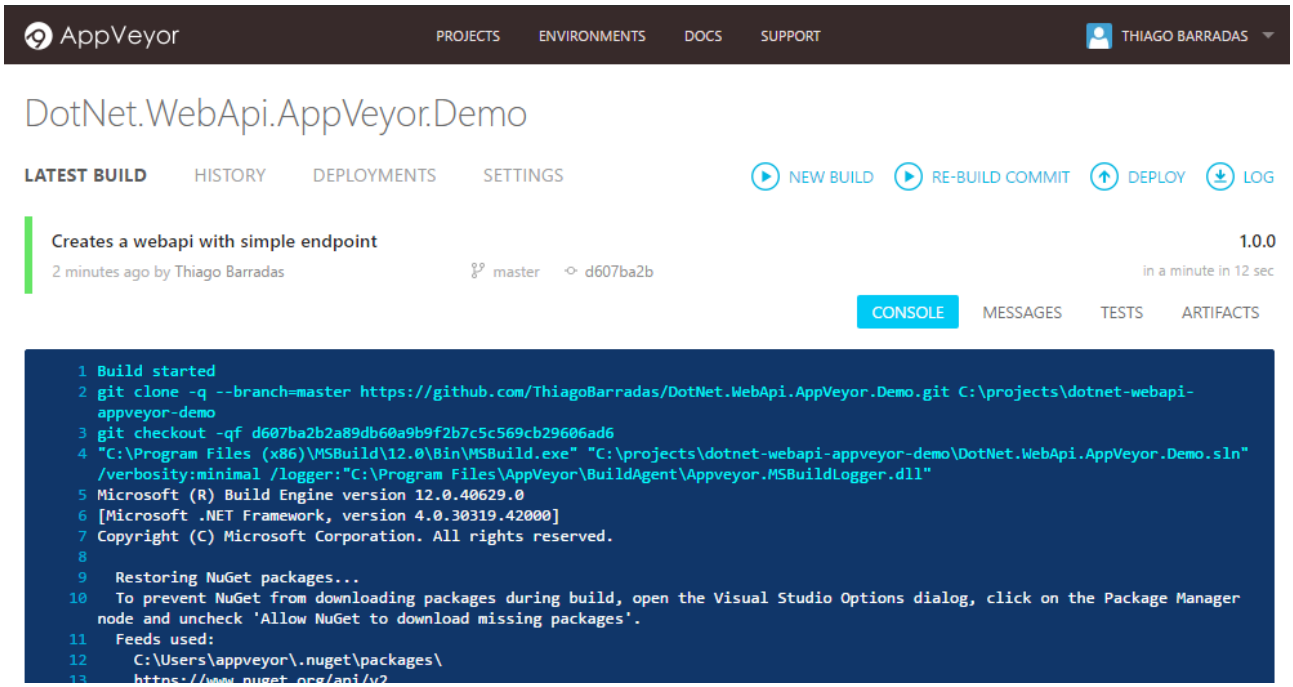
A aplicação criada tem apenas um *endpoint* para exemplificar o resultado final.

```
10          [HttpGet]
11          [Route("")]
            0 references | Thiago Barradas, 13 minutes ago | 1 change
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          }
```

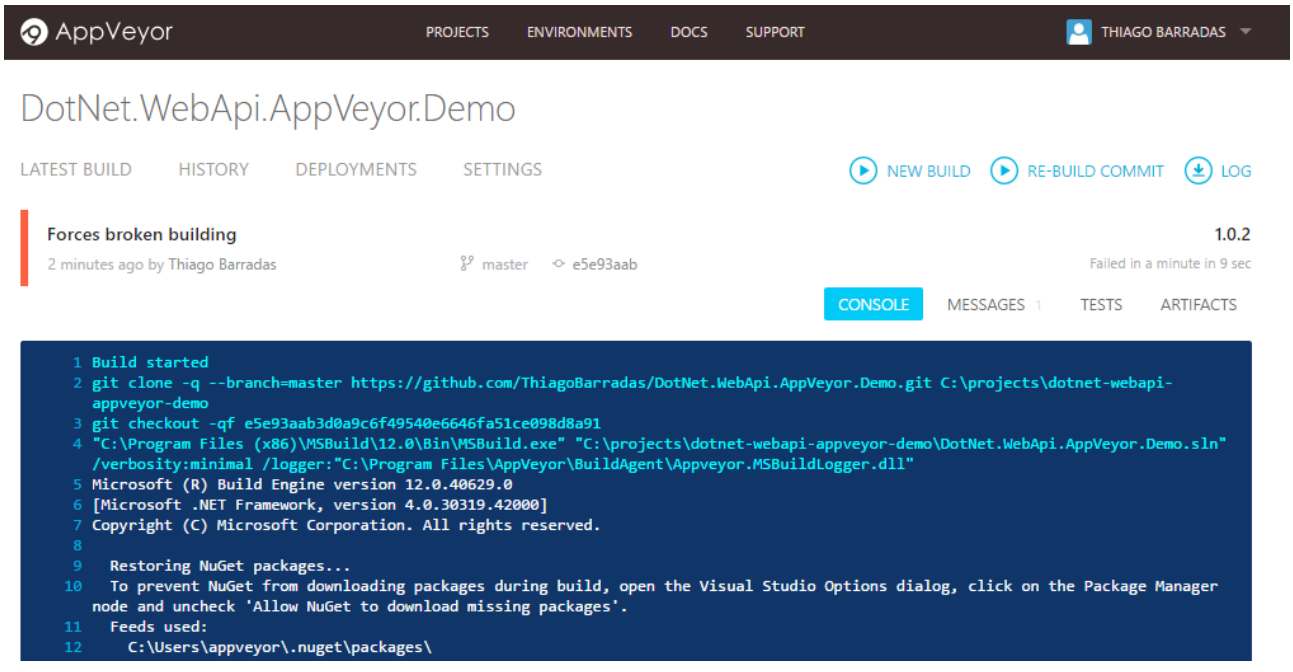
Ao rodar a aplicação o resultado da chamada do *endpoint* seria o seguinte *Json*:



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*.



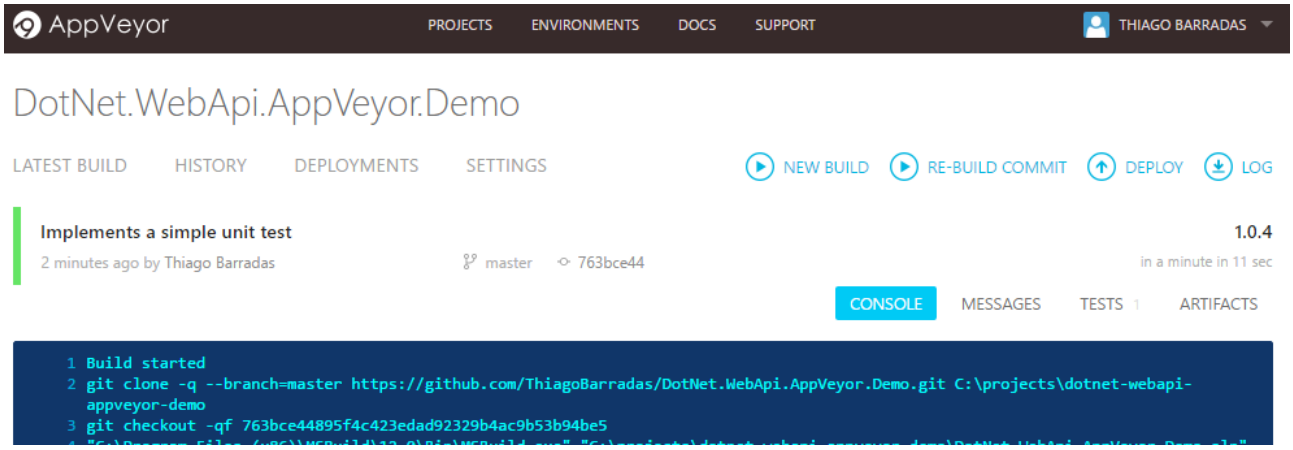
Podemos verificar que o *build* foi concluido com sucesso. Para exemplificar um erro no momento da construção da aplicação, fiz um novo *commit* com um erro no código para forçar o erro na construção.



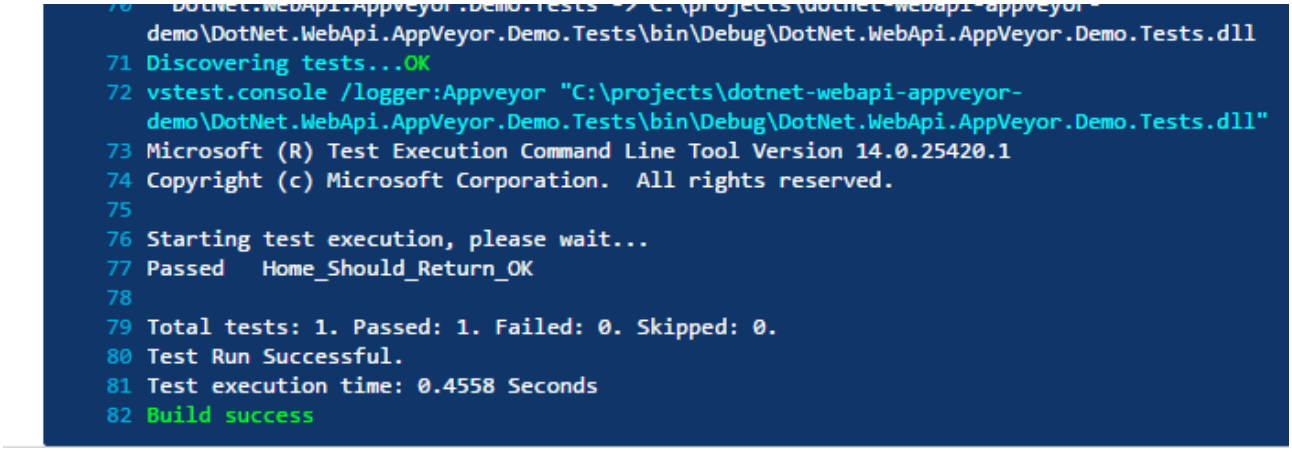
Agora, além de corrigir o código, vou adicionar um simples teste unitário:

```
15          [TestMethod]
            0 references | Thiago Barradas, Less than 5 minutes ago | 1 change
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          }
```

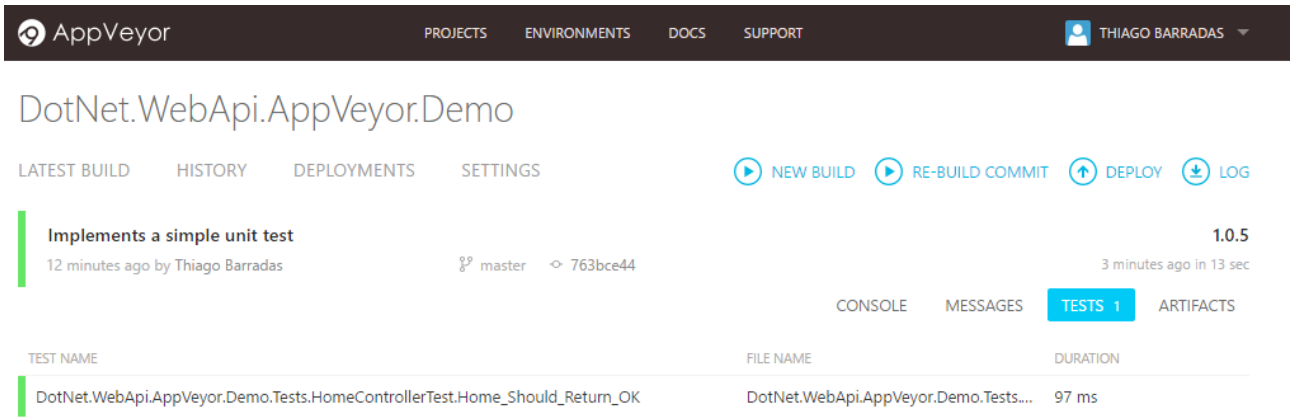
Após subir as novas implementações, no *AppVeyor* podemos conferir o *build* do projeto e o resultado do teste implementado.



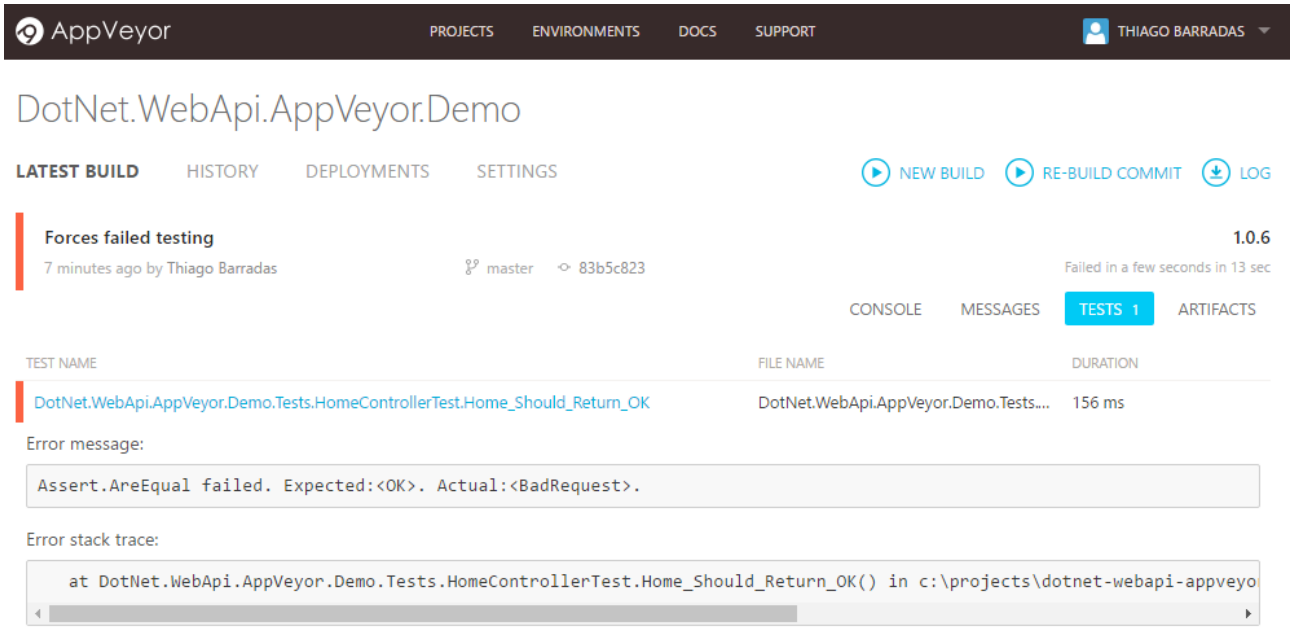
O console informará o resultado além do processo de *build*, o resultado da execução dos testes.



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



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

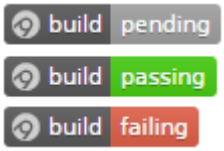


Nesse ponto já atingimos o primeiro nível de maturidade. A integração contínua (*Continuous Integration*), construção da aplicação e testes após modificações em nosso projeto.

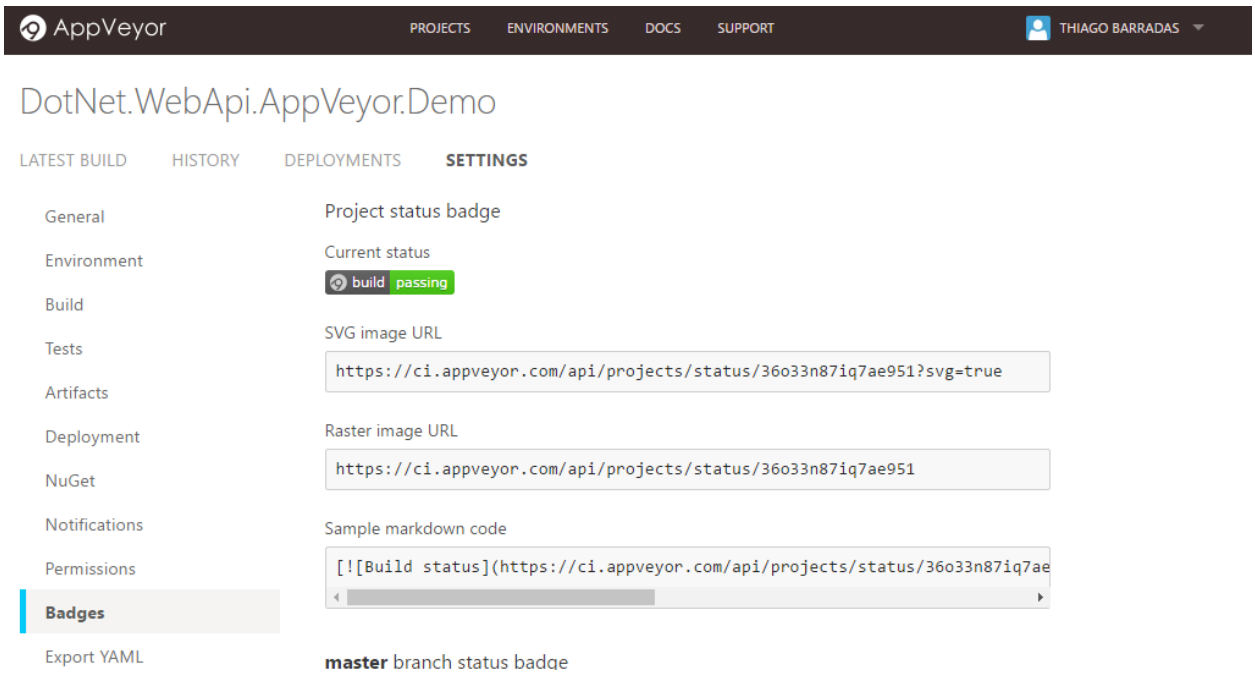
É possível configurar integrações com serviços de notificação como E-mail, Slack, HipChat, Webhooks, entre outros, para notificá-lo sobre o status de novos *builds*.

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).

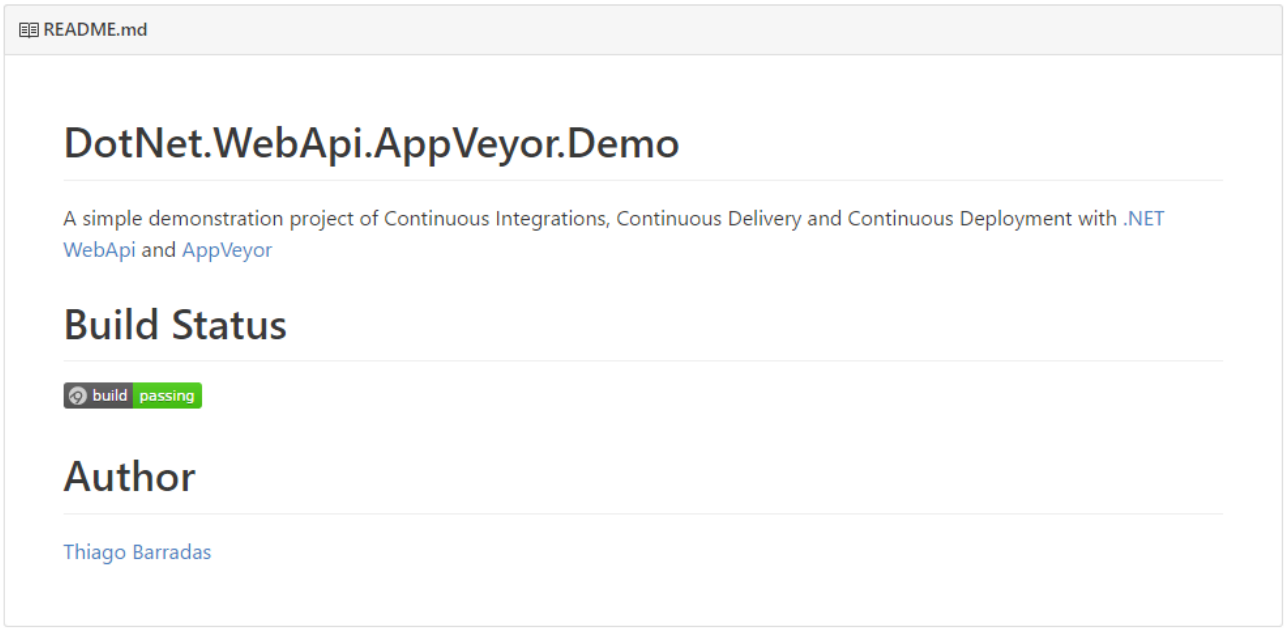
Exemplos dos Badges:



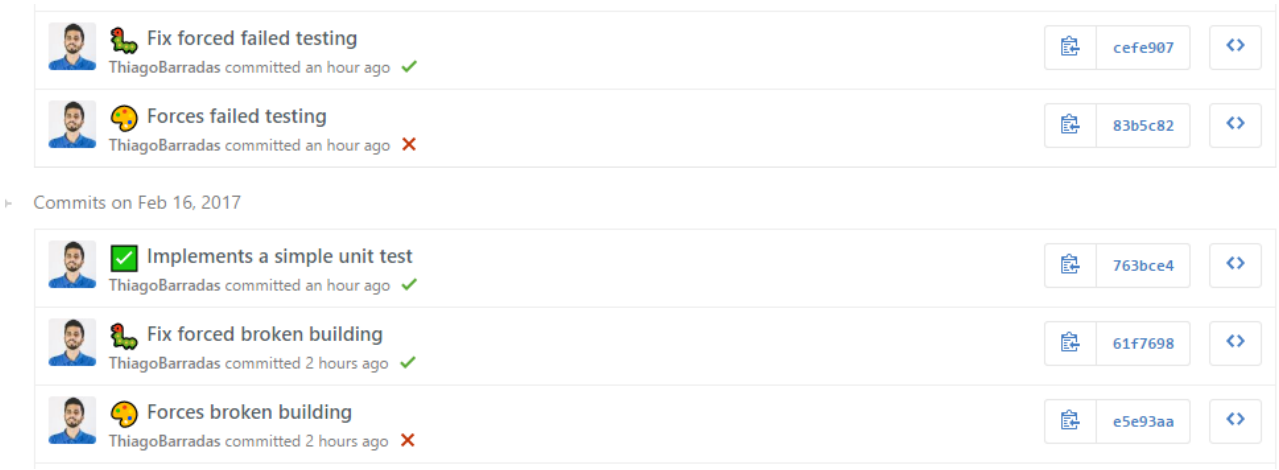
Para obter o código para a inclusão dos Badges basta acessar Settings > Badges:



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

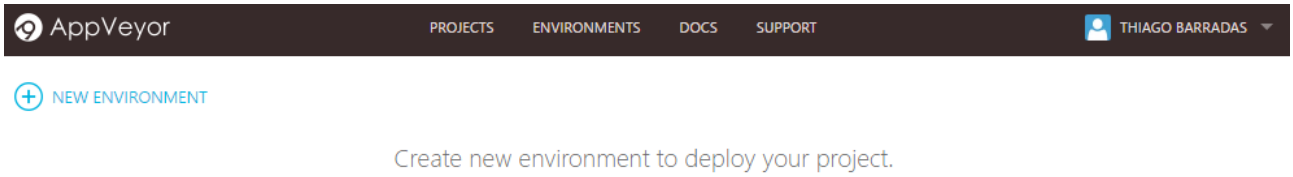


Também podemos ver o status de cada *commit* em que o *AppVeyor* fez a construção e a execução dos tesets a partir da listagem de *commits* do *GitHub*.

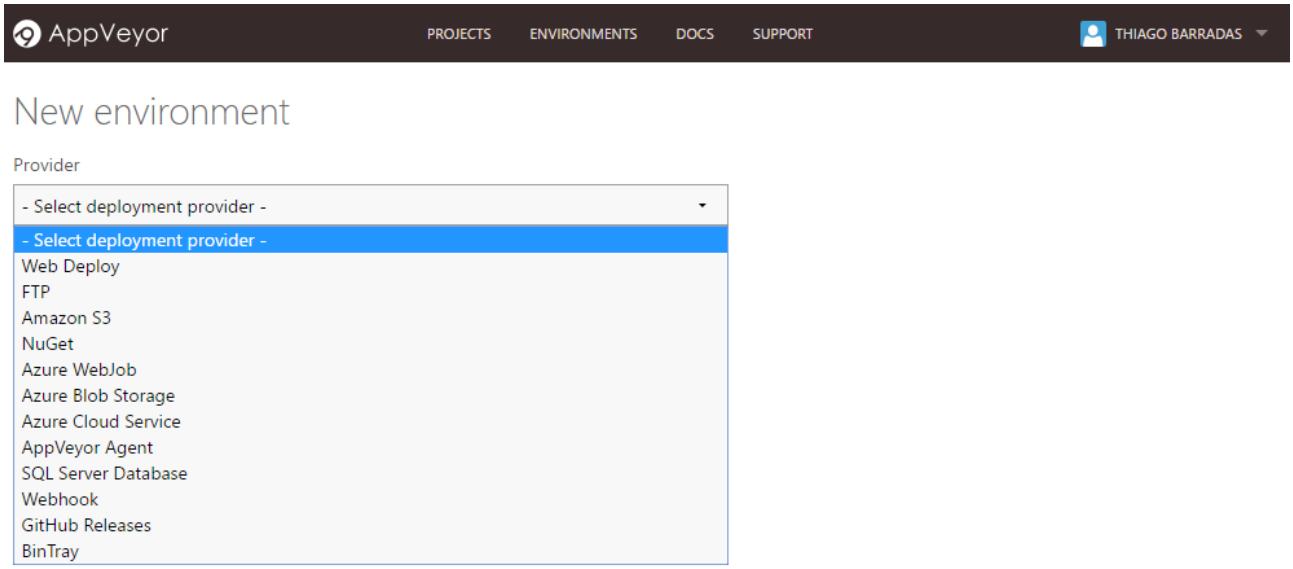


Para configurar a entrega contínua (*Continuous Delivery*) precisamos ter o ambiente que receberá o pacote configurado, e fazer com que o *AppVeyor* prepare o pacote, disponibilizando o artefato para a distribuição.

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

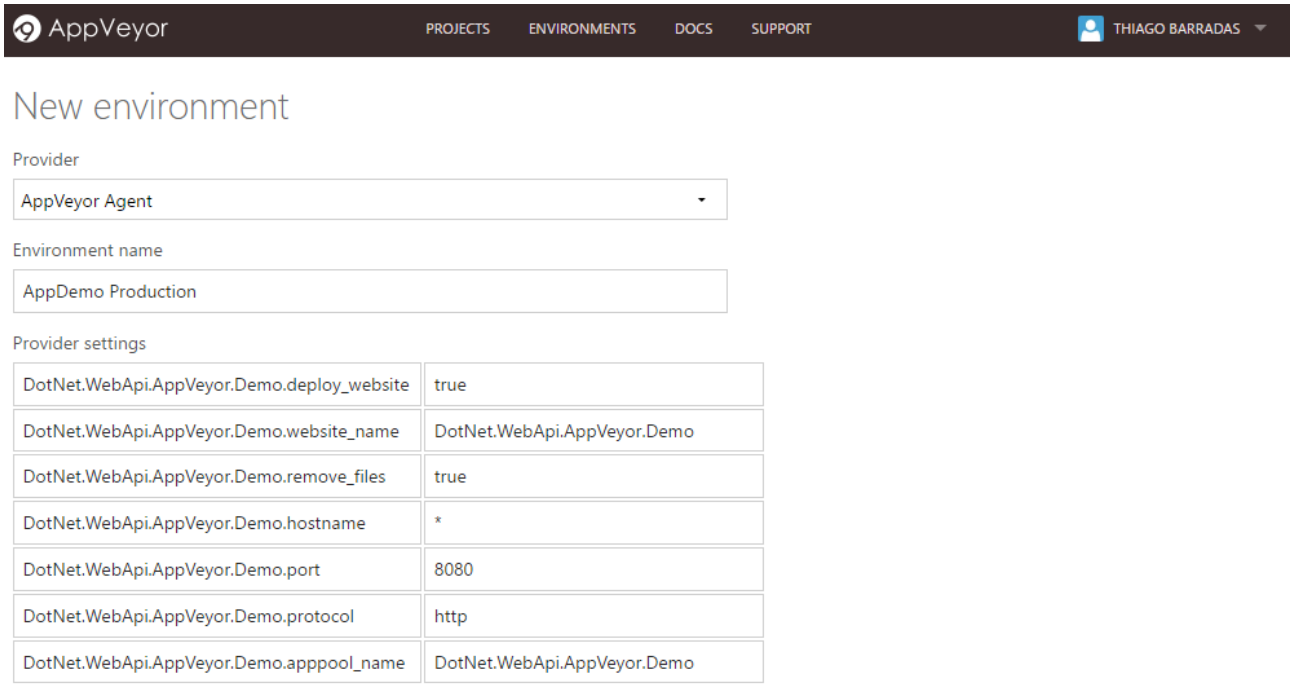


Clique em **New Environment** para criar uma nova configuração para um ambiente:



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:



As configurações devem ter o padrão **{artifact_name}.{config}**.

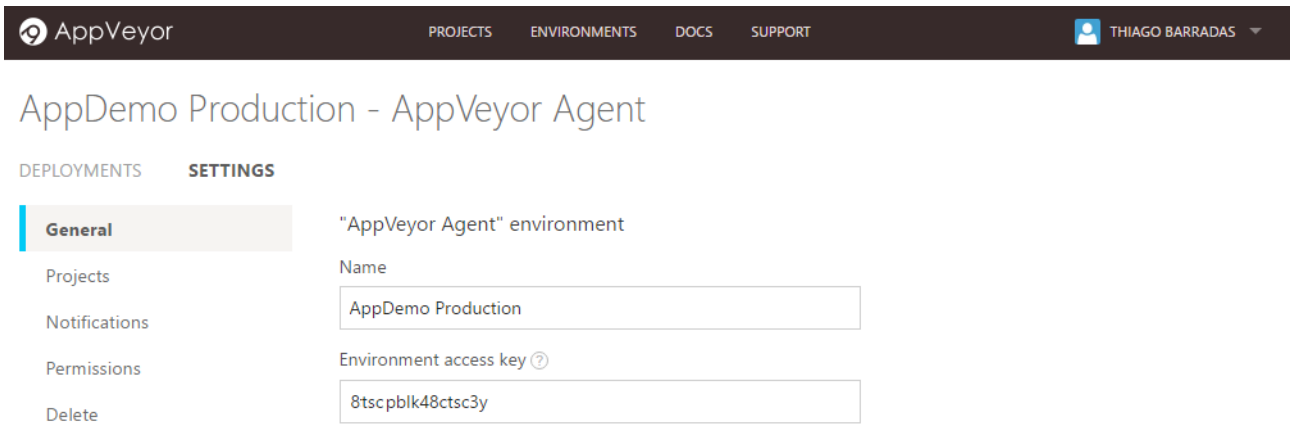
Nesse caso o artefato que vamos gerar, por padrão, usará o nome do projeto: **DotNet.WebApi.AppVeyor.Demo**.

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

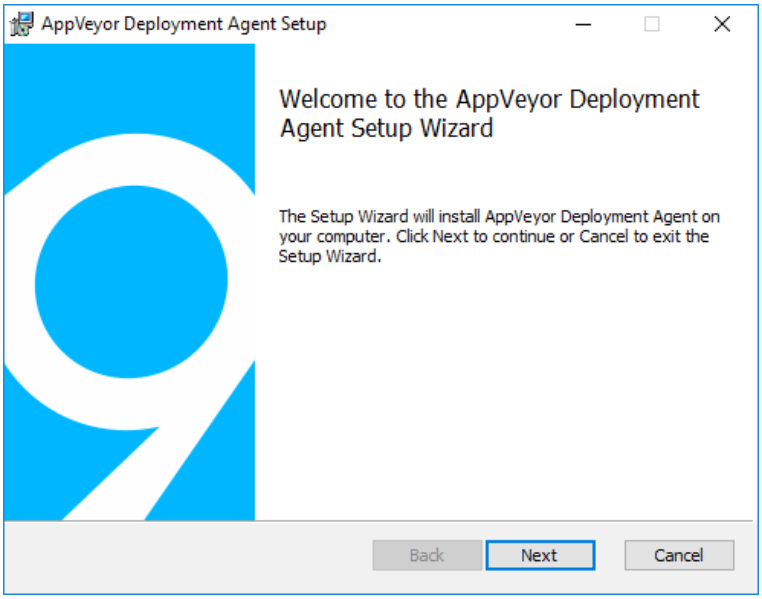
- **deploy_website**: Informa se o *deploy* será de uma aplicação *web*;
- **website_name**: Nome da aplicação no *IIS*;
- **remove_files**: Informa se os arquivos serão removidos antes do *deploy* ou se ocorrerá apenas atualização do que foi modificado;
- **hostname**: Caso aplicação no *IIS* esteja 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;
- **apppool_name**: O nome do *pool* de aplicação que a aplicação pertence.

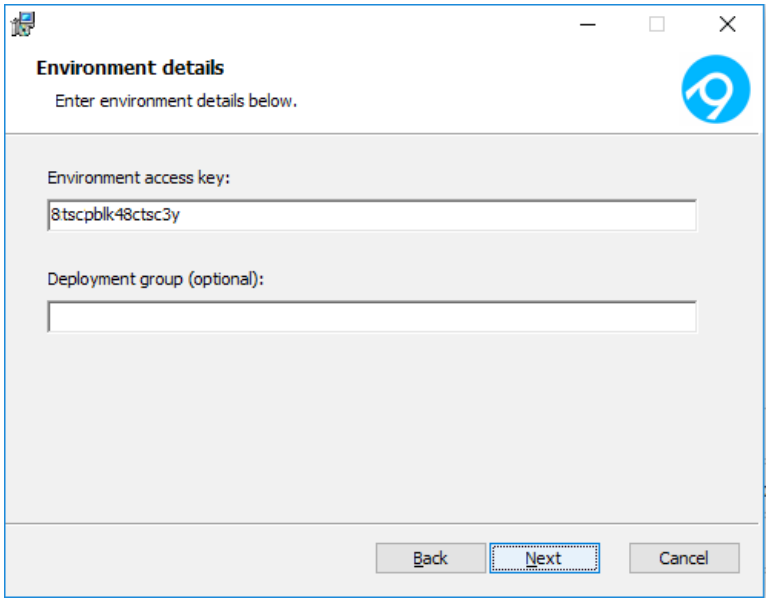
Para configurar mais de uma aplicação no mesmo ambiente, basta incluir as configurações dentro do ambiente utilizando sempre como referência o nome do artefato que será gerado.

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



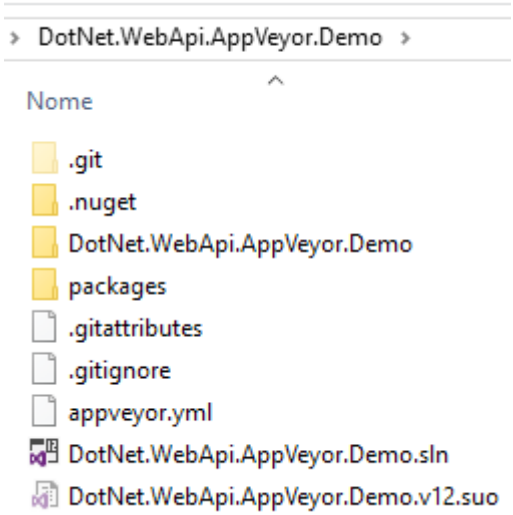
Para instalar o *AppVeyor Agent* nas máquinas, basta fazer o download do instalador e executar nas máquinas desejadas. Durante a instalação será solicitado o *Environment Access Key* e informaremos a chave do nosso ambiente:

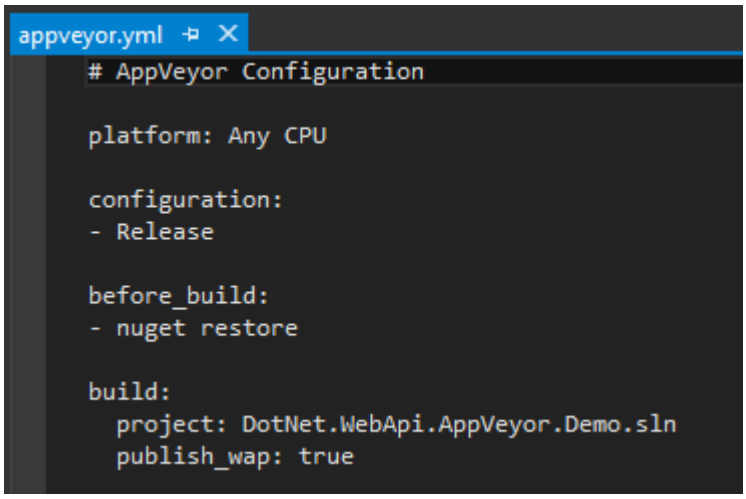




Após a instalação, restará apenas configurar nossa aplicação para que o pacote seja preparado.

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.

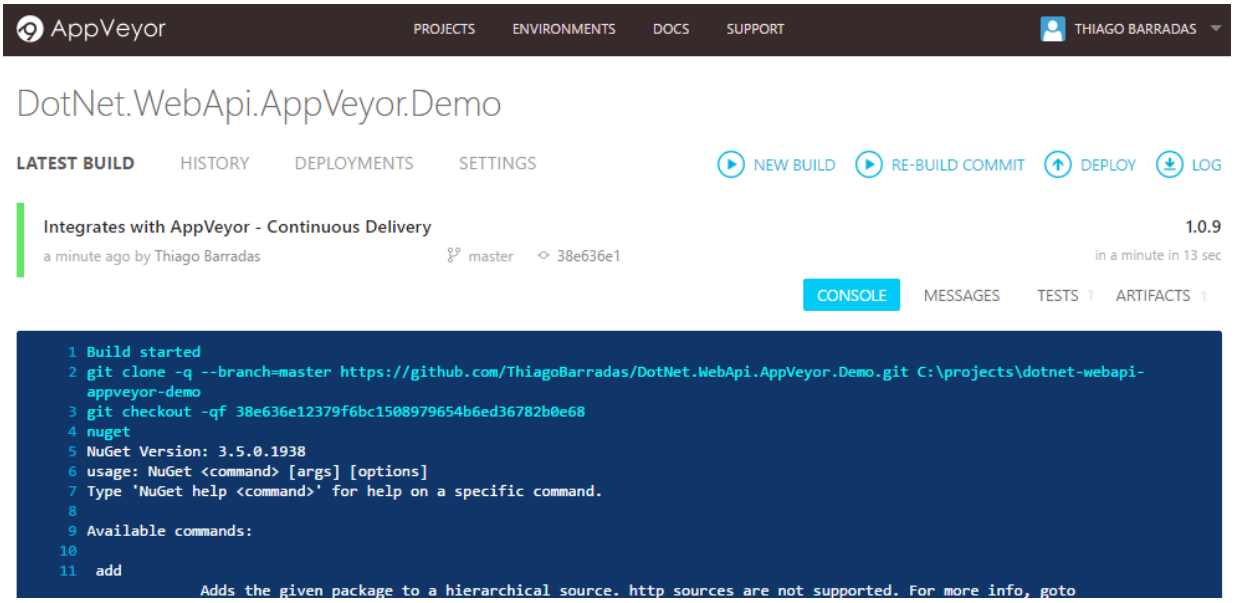
Na opção **before_build** (evento que ocorre antes do *build*) foi executado o comando para restaurar os pacotes instalados via *nuget*. Para que isso funcione é importante que seja habilitado no *Visual Studio* a opção "*Enabled nuget package restore*" clicando com o botão direito em cima da solução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

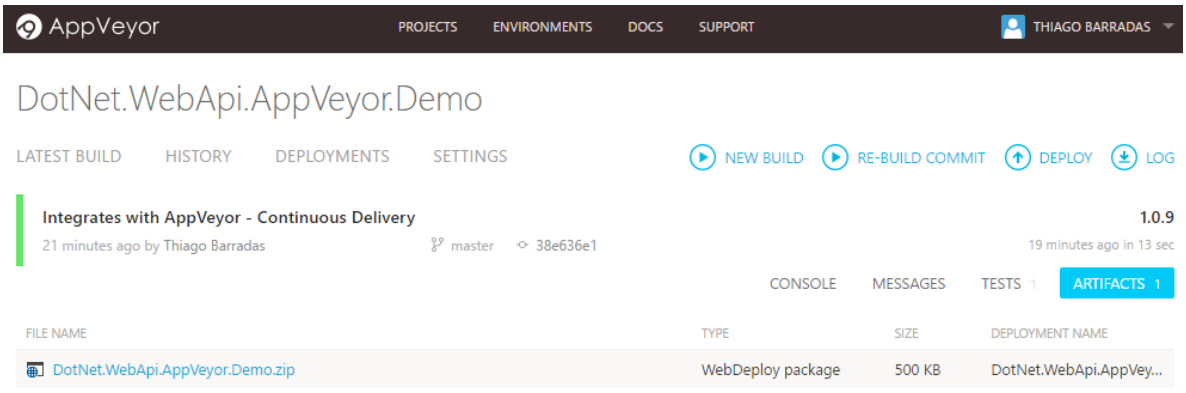
Em **publish_wap** informamos o valor *true* para informar que um pacote de uma aplicação *web* deve ser criado (ou seja, esse comando é o responsável por gerar o artefato).

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.



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

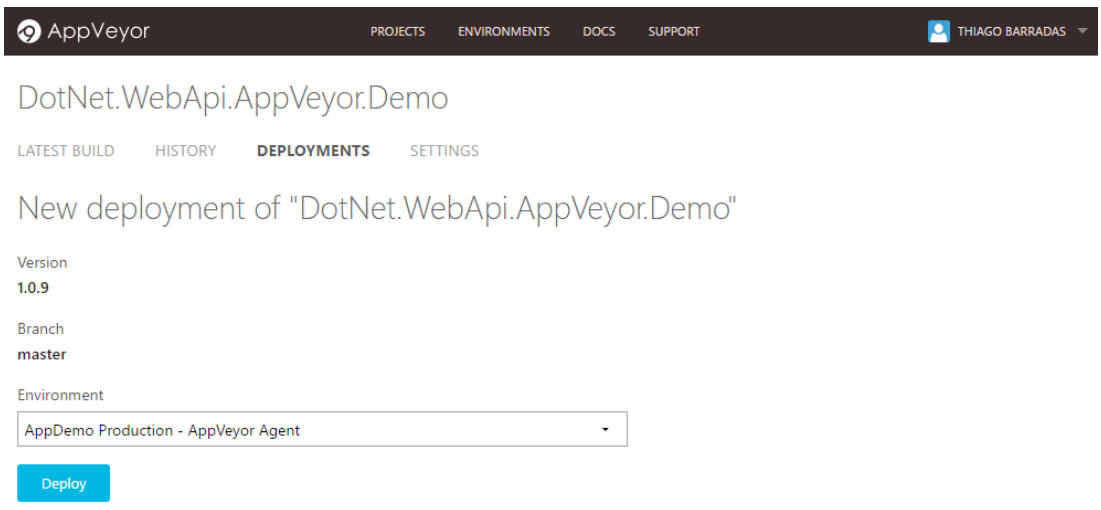


Para efetuar o *deploy* desse pacote basta clicar no botão **Deploy**.

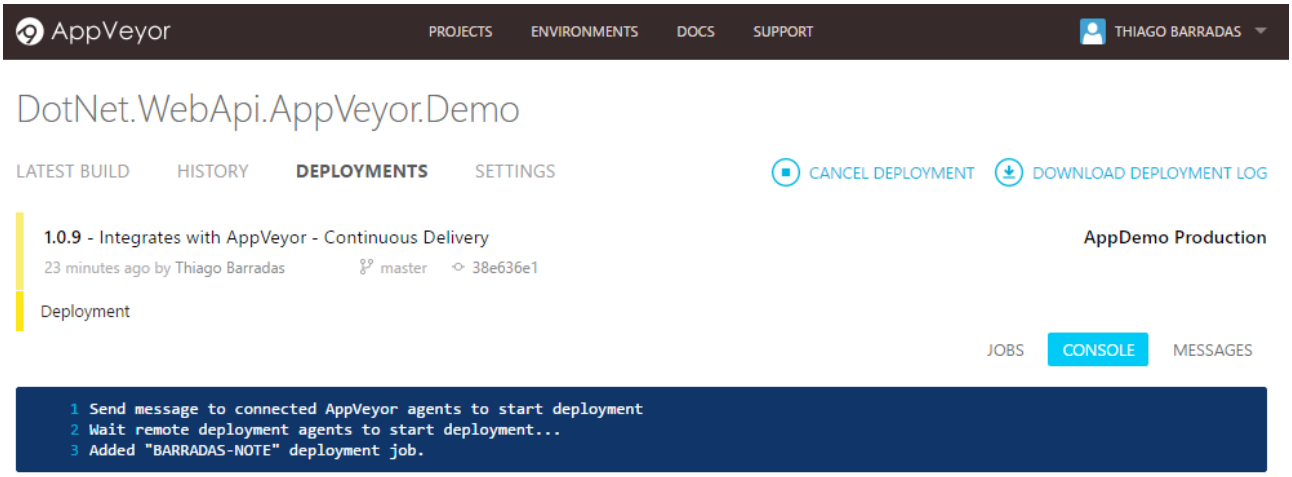
As permissões para os usuários na sua conta do *AppVeyor* podem ser definidas para, entre diversas opções, informar quem tem permissão para efetuar *deploy*.

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

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



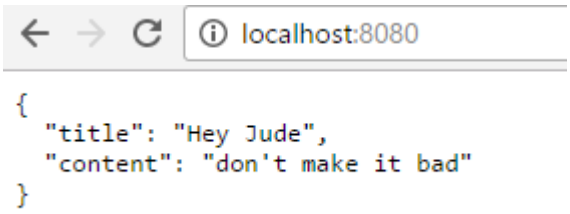
O *AppVeyor* identificará as máquinas que o agente está instalado com a chave do ambiente escolhido e iniciará os *Jobs* referentes a cada máquina.



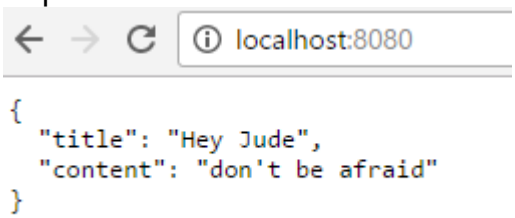
Ao término, a aplicação estará atualizada.

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

Antes:



Depois:



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

```
# 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.

```
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/