

Instituto Nacional de Telecomunicações - INATEL

**Pós-graduação em Desenvolvimento de Aplicações
para Dispositivos Móveis e Cloud Computing**

DM113

Desenvolvimento de serviço SOAP com WCF em C#

PARTE 1

Autor: Prof. M.Sc. Felipe Andery Reis

Sumário

1.	Introdução.....	4
	Projeto do curso	4
2.	Introdução ao <i>Windows Communication Foundation</i>	5
	PRÁTICA 01 - Criação do projeto <i>ProductsEntityModel</i>	6
	Modelo de dados.....	7
	Adicionando o Migrations	12
	PRÁTICA 02 - Criação do projeto <i>ProductsService</i>	19
	Definição do Contrato de Dados.....	22
	Definição do Contrato de Serviço	23
	Implementação do Serviço	24
	Configuração do Serviço	29
	Teste do Serviço usando um <i>Browser</i>	30
	PRÁTICA 03 - Teste do serviço com o cliente de teste do WCF.....	31
	PRÁTICA 04 - Criação da aplicação Cliente.....	33
	Execução da Aplicação Cliente	38
3.	Hospedagem de Serviços WCF	39
	Hospedagem do Serviço no IIS Express	40
	Hospedagem do Serviço usando o <i>Internet Information Services</i>	41
	Hospedagem do Serviço em uma aplicação (self-hosting)	41
	PRÁTICA 05 - Criação do projeto <i>ProductsServiceLibrary</i>	42
	Reconstrução do <i>ProductsService</i> como uma Biblioteca	42
	PRÁTICA 06 - Criação do projeto <i>ProductsServiceHost</i>	45
	Criação de um Aplicativo para Hospedar o Serviço	45
	Configuração da Aplicação de Hospedagem	46
	Adição de um <i>Endpoint</i> TCP	46
	Teste da Aplicação Hospedeira.....	48
	Adição de um <i>Endpoint</i> HTTP	50
	Configuração do Cliente para Conexão HTTP	51
	<i>Endpoints</i> e <i>Bindings</i>	52
	Hospedagem do Serviço no Windows Azure	53
	PRÁTICA 07 - Configurando os recursos no Azure	53
	Publicando o serviço <i>ProductsService</i> no Azure.....	66

Configuração do Cliente para Conexão HTTP ao serviço no Azure.....	70
4. Referências Bibliográficas.....	73

1. Introdução

A disciplina DM113 tem como objetivo apresentar a plataforma *Windows Communication Foundation* (WCF) [JSHP], que disponibiliza ferramentas para a criação de sistemas orientados a serviços para o *Microsoft Windows*.

Utilizando a linguagem C#, serviços e aplicações serão criados para a demonstração das funcionalidades do WCF. A comunicação entre os serviços e aplicações se dará através do SOAP (*Simple Object Access Protocol* - protocolo de acesso a objetos simples), que tem a função de formatar as mensagens XML (*Extensible Markup Language* - linguagem de marcação extensível) e tratar seu envio e recebimento.

Os exemplos demonstrados nesta apostila serão simples e didáticos para que você aluno possa assimilar com mais facilidade os conceitos iniciais do WCF de forma a prepará-lo para exemplos mais complexos.

As seguintes ferramentas e recursos serão utilizados durante as aulas:

- Microsoft Visual Studio Community 2017
- Recursos do Portal Azure

Projeto do curso

O projeto de um sistema completo orientado a serviços está sendo desenvolvido durante o curso de pós-graduação.

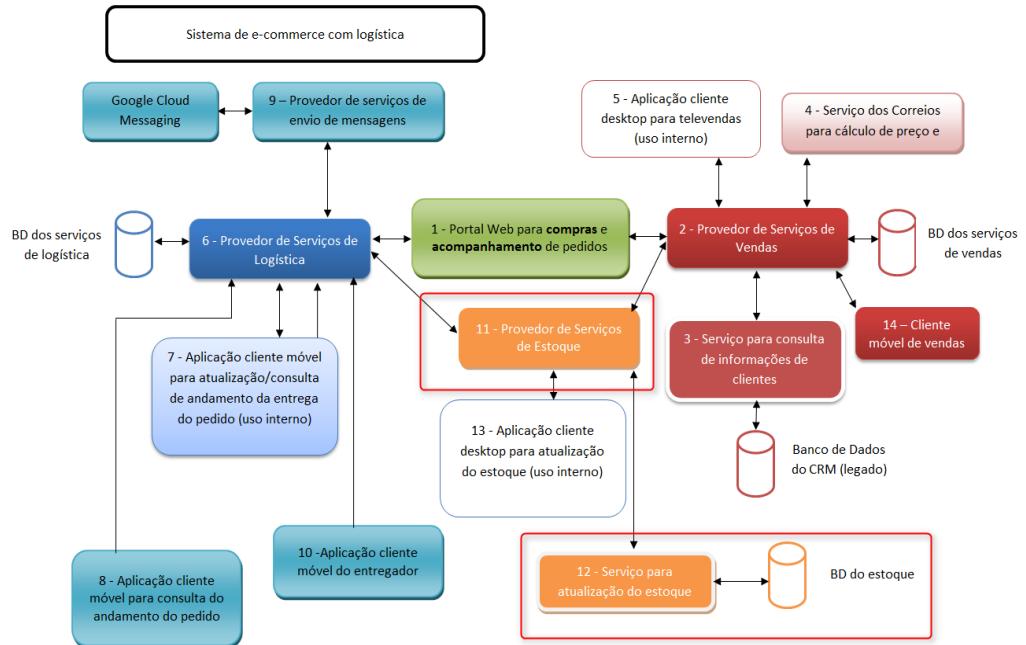


Figura 1 - Sistema completo orientado a serviços

A disciplina DM113 apresentará as ferramentas necessárias para a criação do provedor de serviços de estoque, abordando os seguintes tópicos:

- Introdução ao *Windows Communication Foundation*;
- Hospedagem de serviços WCF;
- Mudança nos contratos de serviço e de dados;

2. Introdução ao *Windows Communication Foundation*

O *Windows Communication Foundation* (WCF), parte do .NET 3.0, consiste em um modelo de programação unificada da Microsoft para a criação de aplicativos orientados a serviços (SOA). É um conjunto de ferramentas que permite a criação de sistemas que enviam mensagens entre serviços e clientes [WIWCF]. A ideia da criação do WCF pela Microsoft foi agrupar as funcionalidades das tecnologias de aplicações distribuídas já existentes como Message Queue, COM+, .Net Remoting e Web Services, em uma única plataforma. Resumindo, o WCF agrupa uma grande variedade de recursos de sistemas distribuídos em uma arquitetura extensível que suporta vários meios transportes, padrões de mensagens, codificações, topologias de rede e modelos de hospedagem [MLIU].

Componentes para criação de um serviço WCF

Toda comunicação com um serviço WCF ocorre através de *endpoints*. Estes *endpoints* fornecem aos clientes o acesso às funcionalidades oferecidas pelo serviço WCF. Cada *endpoint* consiste basicamente de 3 propriedades:

- endereço (*address*) que indica onde o *endpoint* pode ser localizado, ou seja, onde ele está hospedado;
- associação (*binding*) que especifica quais protocolos de transporte (HTTP, TCP, WS e outros) que um cliente pode se comunicar com o serviço e o endereço do *endpoint*;
- contrato (*contract*) que identifica as operações disponibilizadas pelo serviço para as aplicações clientes;

Algumas características do WCF são listadas a seguir:

Orientação a serviços - permite a criação de aplicativos orientados a serviços (SOA).

Interoperabilidade - o WCF implementa os padrões modernos da indústria para a interoperabilidade de serviço Web.

Metadados de serviço - o WCF dá suporte aos metadados de serviço de publicação usando formatos especificados nos padrões da indústria como WSDL, Esquema XML e WS-Policy.

Vários meios de transportes e codificações - as mensagens podem ser enviadas de qualquer um dos vários protocolos e codificações internos de transporte. O protocolo e a codificação mais comum é enviar mensagens SOAP codificadas por texto usando o protocolo HTTP para uso na internet. Como alternativa, o WCF permite o envio de mensagens por TCP, *named pipes* ou MSMQ.

Segurança - as mensagens podem ser criptografadas para proteger a privacidade e exigir que os usuários se autentiquem antes de terem permissão de receber mensagens.

Nos exemplos a seguir serão demonstrados os procedimentos para a criação de um serviço WCF, o *ProductsService*, que irá fornecer o acesso à base de dados *ProductsModel*. Serão definidos os contratos de serviço e de dados, assim como a implementação, configuração e teste do serviço.

Projeto Exemplo - *ProductsService*

Para demonstração da utilização da plataforma WCF um projeto será criado utilizando o Microsoft Visual Studio. No Visual Studio, uma solução chamada *ProductsService* será criada e será composta pelos seguintes projetos:

- *ProductsEntityModel* – para a criação do banco de dados
- *ProductsService* – para a criação do serviço
- *ProductsClient* – para a criação da aplicação cliente
- *ProductsServiceLibrary* – para construção do serviço em uma biblioteca.
- *ProductsServiceHost* – para hospedagem do serviço em uma aplicação console

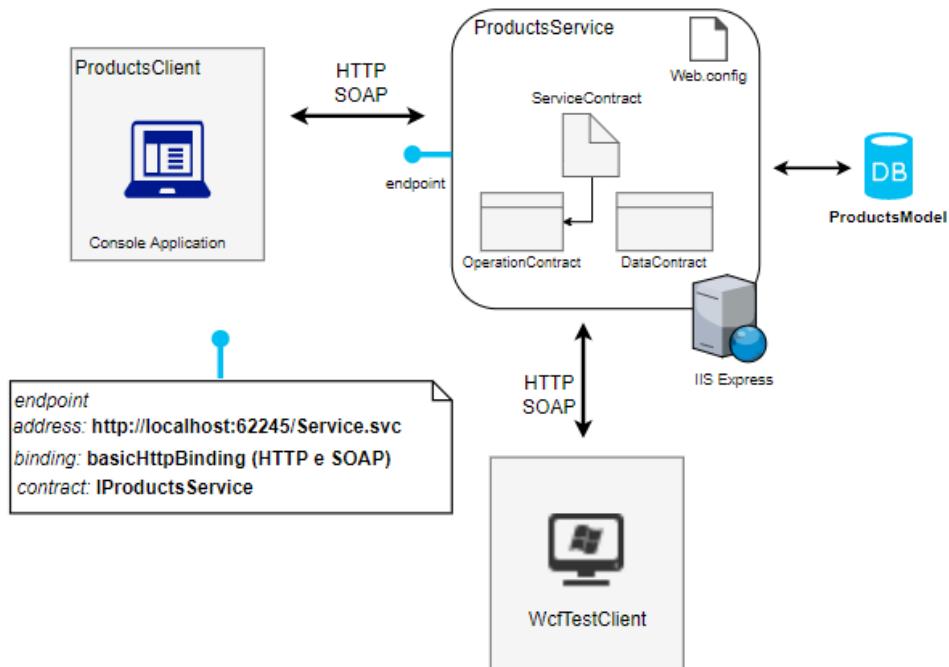


Figura 2 - Projeto Exemplo - *ProductsService*

PRÁTICA 01 - Criação do projeto *ProductsEntityModel*

O banco de dados utilizado no projeto, que será chamado de *ProductsModel*, será criado no projeto *ProductsEntityModel*. A base de dados será gerada através do *Entity Framework Code First* onde também será utilizado o recurso de *Migrations*.

Modelo de dados

Antes de iniciar a criação do serviço, é necessário criar o modelo de dados que será utilizado para comunicação com o banco de dados. Para simplificar e eliminar a necessidade de escrever código para o acesso a base de dados, será utilizado o *ADO.NET Entity Framework* (EF) [EFRA], que é parte do .NET Framework 4.0 e é fornecido junto ao Visual Studio.

O EF é um *Object Relational Mapper (ORM)* que fornece um mapeamento entre as tabelas da base de dados e um conjunto de objetos para ser utilizado nos aplicativos e serviços. Para a criação do modelo de dados será utilizado o *Entity Framework Code First*, que é uma alternativa disponibilizada pelo *Entity Framework*.

Utilizando o *Entity Framework Code First*, os modelos de entidades que especificam as tabelas do banco de dados serão criados e a partir deles o banco de dados será criado. O diagrama de funcionamento do Framework é ilustrado na Figura 3.

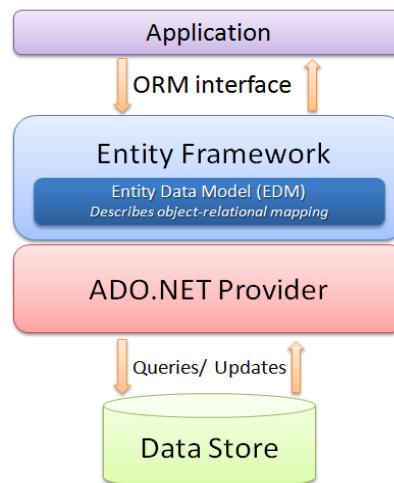


Figura 3 - Diagrama de funcionamento do Entity Framework [EFRA]

O banco de dados *ProductsModel* será constituído por duas tabelas: a tabela *Products*, que representa o cadastro de produtos de uma determinada loja, e a tabela *Stocks*, que manterá um histórico dos estoques para cada produto. O diagrama abaixo ilustra o bloco para a construção da base de dados *ProductsModel*. O arquivo *App.Config* irá conter a string de conexão que será utilizada para a comunicação com o banco de dados.

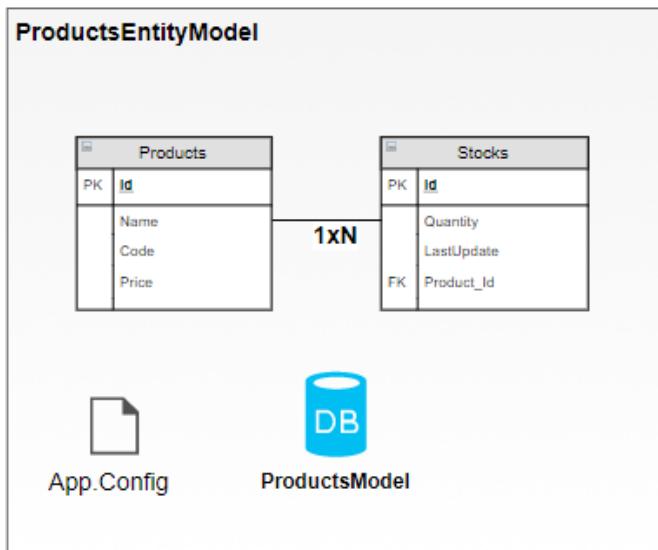


Figura 4 – Projeto ProductsEntityModel

Para gerar o banco de dados *ProductsModel*, os modelos de dados serão criados. Para a criação destes modelos, veja os passos a seguir:

1. Inicie o Visual Studio com privilégios de administrador e crie um novo projeto usando o template *Class Library*, conforme a Figura 5. Nomeie o projeto como *ProductsEntityModel* e a solução como *ProductsService*.

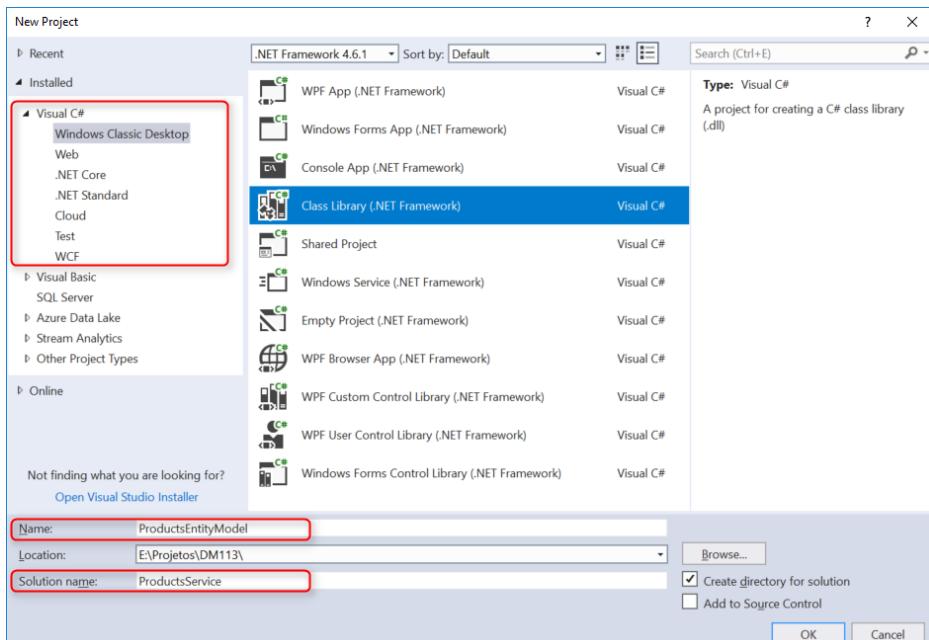


Figura 5 - Novo Projeto e Modelo *Entity*

2. Apague a *Class1.cs* que foi criada na solução.
 3. Clique com o botão direito sobre o projeto *ProductsEntityModel*, *Add* e clique em *New Item*

4. Crie um novo *ADO.NET Entity Data Model* de nome *ProductsModel* como mostrado na Figura 6.

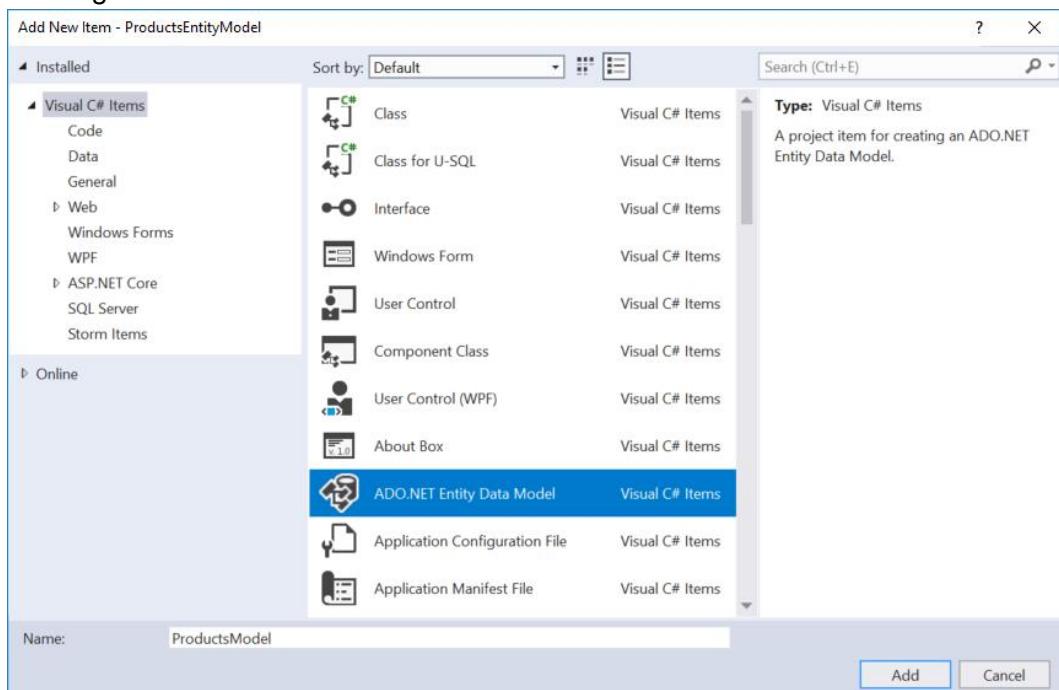


Figura 6 - Criação do ADO.NET Entity Data Model

5. Clique em *Empty Code First Model* e, em seguida, *Finish*, como mostrado na Figura 7.

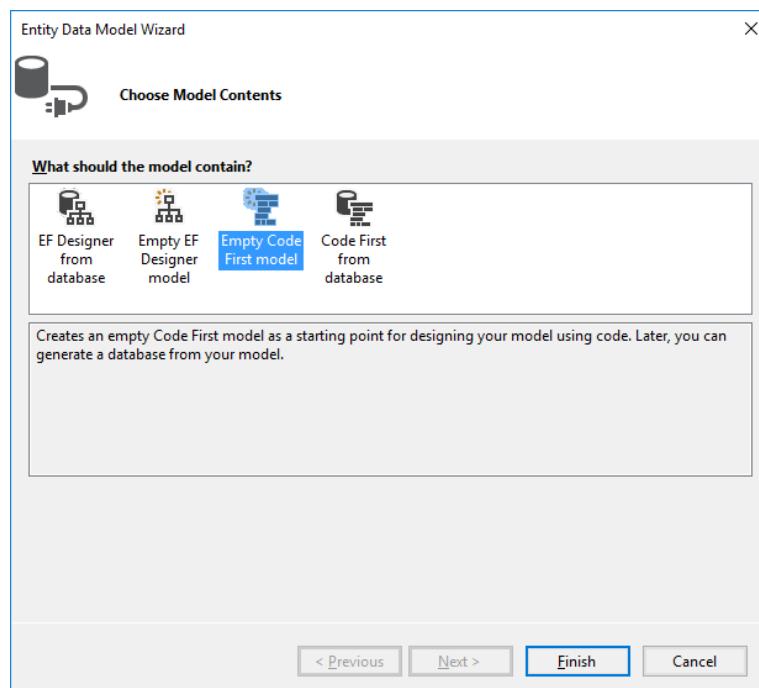


Figura 7 – Criação do modelo *Empty Code First Model*

6. Um arquivo chamado *ProductModel.cs* será criado com um exemplo de implementação. A partir dele serão criados os modelos de entidades que representarão as tabelas *Products* e *Stocks* do banco de dados.

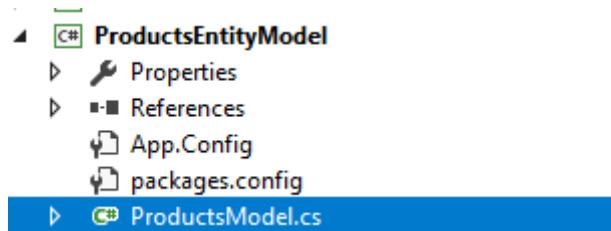


Figura 8 – Criação do arquivo ProductsModel.cs

7. Uma string de conexão também foi gerada no arquivo *App.Config*. Ela será utilizada futuramente para a conexão com o banco de dados que será gerado.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3      <configSections>
4          <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5          <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Cul
6      </configSections>
7      <connectionStrings>
8          <add name="ProductsModel" connectionString="data source=(LocalDb)\MSSQLLocalDB;initial catalog=ProductsEntityModel.ProductsModel;integrat
9      </connectionStrings>
10     <entityFramework>
11         <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />
12         <providers>
13             <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
14         </providers>
15     </entityFramework>
16 </configuration>

```

Figura 9 – String de conexão gerada

8. Abra o arquivo *ProductModel.cs*. Verifique que ele já possui a implementação da classe *ProductModel* que herda um contexto de *DbContext*. O contexto representa uma sessão com o banco de dados, que nos permite realizar as operações básicas com o banco de dados, como salvar, deletar, adicionar e atualizar. A partir desta classe será criado o contexto para trabalhar com o *Migrations*.

```

1  namespace ProductsEntityModel
2  {
3      using System;
4      using System.Data.Entity;
5      using System.Linq;
6
7      public class ProductsModel : DbContext
8      {
9          // Your context has been configured to use a 'ProductsModel' connection string from your application's
10         // configuration file (App.config or Web.config). By default, this connection string targets the
11         // 'ProductsEntityModel.ProductsModel' database on your LocalDb instance.
12         //
13         // If you wish to target a different database and/or database provider, modify the 'ProductsModel'
14         // connection string in the application configuration file.
15         public ProductsModel()
16             : base("name=ProductsModel")
17         {
18         }
19
20         // Add a DbSet for each entity type that you want to include in your model. For more information
21         // on configuring and using a Code First model, see http://go.microsoft.com/fwlink/?LinkId=390109.
22
23         // public virtual DbSet<MyEntity> MyEntities { get; set; }
24
25
26         //public class MyEntity
27         //{
28         //    public int Id { get; set; }
29         //    public string Name { get; set; }
30         //}
31     }

```

Figura 10 – Conteúdo do arquivo ProductsModel.cs

9. Dentro do namespace *ProductsEntityModel* e após a classe *ProductsModel*, adicione o modelo de dados *Product* e *Stock* conforme o código a seguir. O relacionamento entre *Product* e *Stock* será de 1 x N.

```

public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Code { get; set; }
    public decimal Price { get; set; }
    public List<Stock> Stock { get; set; }
}

public class Stock
{
    public int Id { get; set; }
    public decimal Quantity { get; set; }
    public DateTime LastUpdate { get; set; }
    public Product Product { get; set; }
}

```

10. Após a inclusão dos modelos de entidades, será adicionado um *DbSet* que representa cada tipo de entidade criada no contexto. Dentro da classe *ProductsModel*, adicione o código conforme a seguir:

```

public virtual DbSet<Product> Product { get; set; }
public virtual DbSet<Stock> Stock { get; set; }

```

Como as classes já foram criadas e o arquivo de configuração atualizado com a string de conexão, o próximo passo será adicionar o Migrations.

```
public class ProductsModel : DbContext
{
    // Your context has been configured to use a 'ProductsModel' connection string from your application's
    // configuration file (App.config or Web.config). By default, this connection string targets the
    // 'ProductsEntityModel.ProductsModel' database on your LocalDb instance.
    //
    // If you wish to target a different database and/or database provider, modify the 'ProductsModel'
    // connection string in the application configuration file.
    public ProductsModel()
        : base("name=ProductsModel")
    {

    }

    // Add a DbSet for each entity type that you want to include in your model. For more information
    // on configuring and using a Code First model, see http://go.microsoft.com/fwlink/?LinkId=390109.
    public virtual DbSet<Product> Products { get; set; }
    public virtual DbSet<Stock> Stocks { get; set; }

}

public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Code { get; set; }
    public decimal Price { get; set; }
    public List<Stock> Stock { get; set; }
}

public class Stock
{
    public int Id { get; set; }
    public decimal Quantity { get; set; }
    public DateTime LastUpdate { get; set; }
    public Product Product { get; set; }
}
```

Figura 11 – Arquivo ProductsModel.cs final

Criação do banco de dados ProductsModel

Adicionando o Migrations

O Migrations é um recurso do *Entity Framework* que permite realizar alterações nos modelos de entidades e ter uma atualização automática do banco de dados trazendo essas mudanças. Siga os passos para adicionar e utilizar o recurso Migrations:

1. Atualize a versão do *EntityFramework* clicando com o botão direito sobre o projeto *ProductsEntityModel* e em seguida *Manage NuGet Packages*. Em *Browse*, busque pelo nome *EntityFramework* e ao lado direito clique em *Install*.

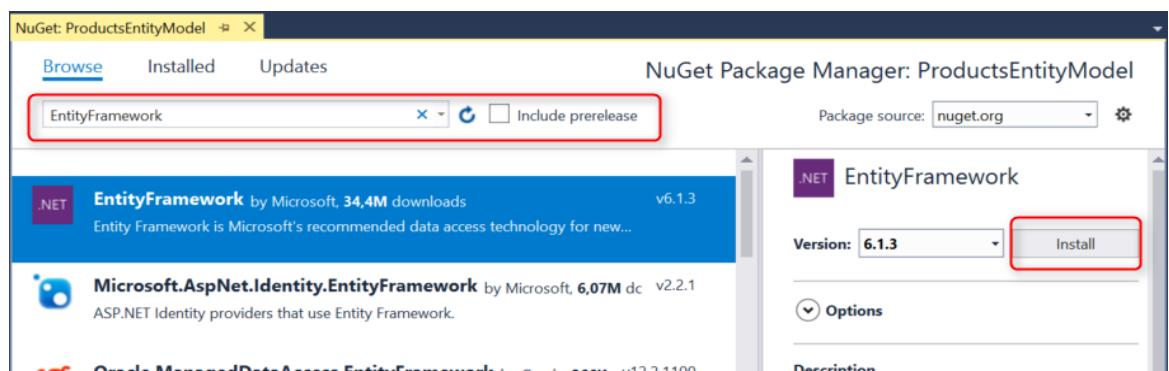


Figura 12 – Atualização do Entity Framework

2. No Visual Studio, abra o menu *Tools* -> *NuGet Package Manager* -> *Package Manager Console*

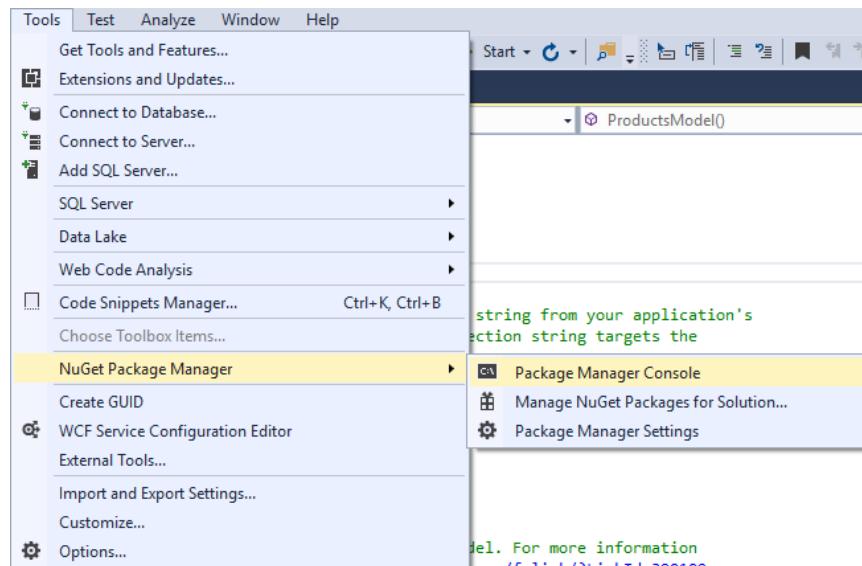


Figura 13 – Package Manager Console

3. No console do *Package Manager*, adicione o *Migrations* ao projeto através do comando a seguir:

Enable-Migrations

The image shows the 'Package Manager Console' window. The command 'PM> Enable-Migrations' is highlighted with a red box. The output shows the command being run and the confirmation that 'Code First Migrations enabled for project ProductsEntityModel.'

```
Package Manager Console
Package source: All | Default project: ProductsEntityModel
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include dependencies which are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 4.3.1.4445

Type 'get-help NuGet' to see all available NuGet commands.

PM> Enable-Migrations
Checking if the context targets an existing database...
Code First Migrations enabled for project ProductsEntityModel.
PM> |
```

Figura 14 – Adicionando o *Migrations*

4. Após o comando de habilitar o *Migrations* para o projeto, observe que a pasta *Migrations* foi criada no projeto *ProductsEntityModel*. Nela se encontra o arquivo *Configurations.cs* que poderá ser editado futuramente para incluir novas configurações.

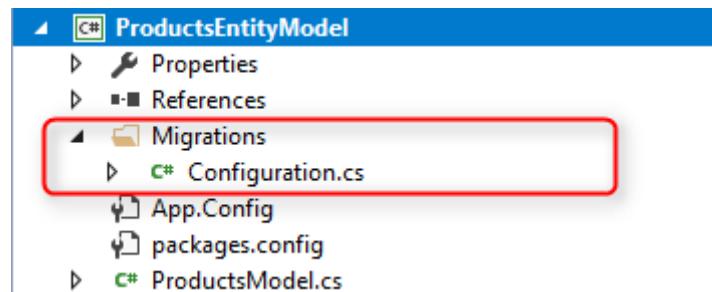


Figura 15 – Pasta *Migrations* criada

5. O próximo passo agora é criar uma alteração no banco através do comando **Add-Migrations** “migration_name”, onde “migration_name” é o nome que será dado para a atualização. No console, entre com o comando a seguir:

Add-Migration “Criacao_do_Banco”

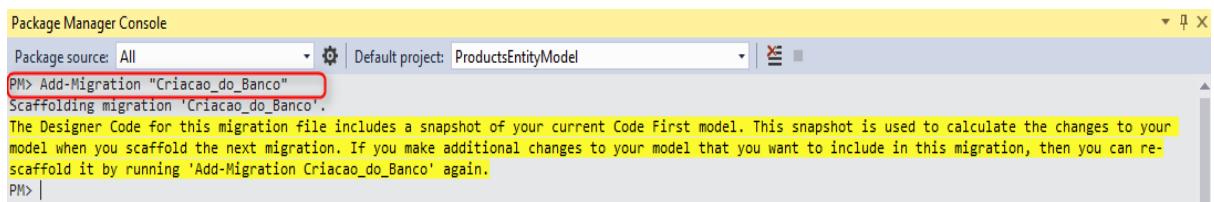


Figura 16 – Comando Add-Migration

6. Após o último comando, observe que na pasta *Migrations* foi adicionado um novo arquivo, que contém os comandos *Migrations* para o banco de dados. O nome deste arquivo é composto por um *timestamp* e o nome dado para a atualização.

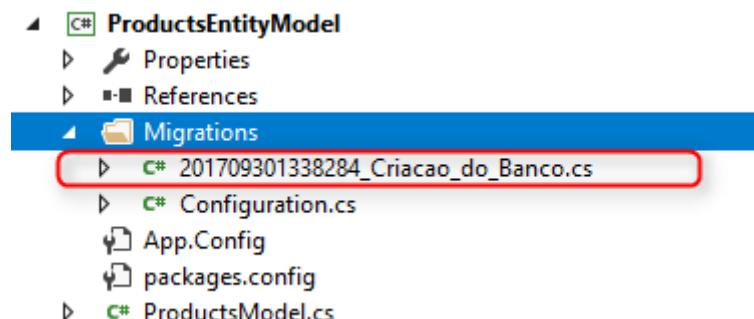


Figura 17 – Arquivo de atualização do Migrations

7. Abra o arquivo *201709301338284_Criacao_do_Banco.cs* que foi gerado através do último comando, e observe que no seu conteúdo, dentro do método *Up()*, contém as alterações que serão realizadas no banco de dados, onde as tabelas *Products* e *Stocks* serão criadas.

```

namespace ProductsEntityModel.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class Criacao_do_Banco : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Products",
                c => new
                {
                    Id = c.Int(nullable: false, identity: true),
                    Name = c.String(),
                    Code = c.String(),
                    Price = c.Decimal(nullable: false, precision: 18, scale: 2),
                })
                .PrimaryKey(t => t.Id);

            CreateTable(
                "dbo.Stocks",
                c => new
                {
                    Id = c.Int(nullable: false, identity: true),
                    Quantity = c.Decimal(nullable: false, precision: 18, scale: 2),
                    LastUpdate = c.DateTime(nullable: false),
                    Product_Id = c.Int(),
                })
                .PrimaryKey(t => t.Id)
                .ForeignKey("dbo.Products", t => t.Product_Id)
                .Index(t => t.Product_Id);
        }
    }
}

```

Figura 18 – Conteúdo do arquivo de atualização do *Migrations*

8. Antes de executar o último comando do *Migrations*, o **Update-Database**, que será responsável por criar o banco de dados com as tabelas informadas no passo anterior, é interessante adicionar alguns registros para popular as tabelas Products e Stock que serão criadas. Para isto, abra o arquivo *Configurations.cs*, dentro da pasta *Migrations*.
9. Dentro do método Seed, do arquivo *Configurations.cs*, adicione o código a seguir.

```

protected override void Seed(ProductsEntityModel.ProductsModel context)
{
    context.Products.AddOrUpdate(
        p => p.Id,
        new Product { Name = "Produto 01", Code = "0001", Price = 100 },
        new Product { Name = "Produto 02", Code = "0002", Price = 150 },
        new Product { Name = "Produto 03", Code = "0003", Price = 200 },
        new Product { Name = "Produto 04", Code = "0004", Price = 250 });
}

context.SaveChanges();

context.Stocks.AddOrUpdate(
    s => s.Id,
    new Stock
    {
        Quantity = 1000,
    });

```

```

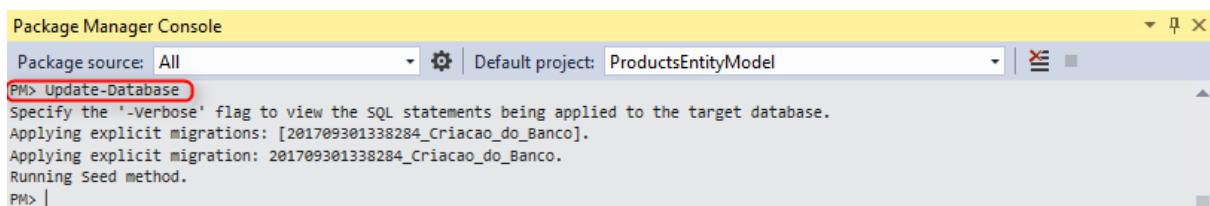
        LastUpdate = DateTime.Now,
        Product = context.Products.FirstOrDefault(x => x.Code == "0001")
    },
    new Stock
    {
        Quantity = 2000,
        LastUpdate = DateTime.Now,
        Product = context.Products.FirstOrDefault(x => x.Code == "0002")
    },
    new Stock
    {
        Quantity = 3000,
        LastUpdate = DateTime.Now,
        Product = context.Products.FirstOrDefault(x => x.Code == "0003")
    },
    new Stock
    {
        Quantity = 4000,
        LastUpdate = DateTime.Now,
        Product = context.Products.FirstOrDefault(x => x.Code == "0004")
    }
);
}

}

```

10. O próximo passo agora é criar uma criado o banco de dados *ProductsModel* através do comando **Update-Database** com as tabelas populadas. No console, entre com o comando a seguir:

Update-Database



```

Package Manager Console
Package source: All | Default project: ProductsEntityModel | X
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [201709301338284_Criacao_do_Banco].
Applying explicit migration: 201709301338284_Criacao_do_Banco.
Running Seed method.
PM> |

```

Figura 19 – Comando Update-Database

11. Verifique se a base de dados foi criada. No Visual Studio, clique em View ->SQL Server Object Explorer. Verifique se a instância para o banco **(localdb)\MSSQLLocalDB** está criada.

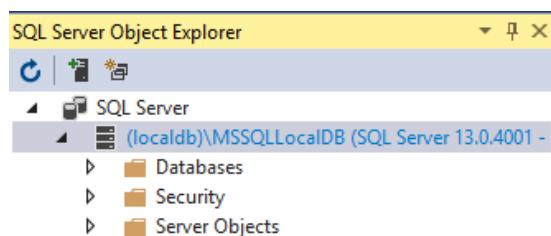
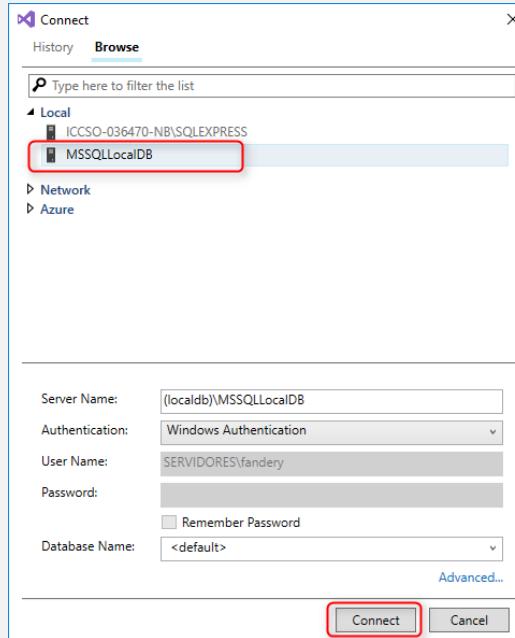


Figura 20 – Instância do *localdb*

[ATENÇÃO]

Caso a instância para o **(localdb)\MSSQLLocalDB** não esteja criada, na opção **SQL Server**, clique com o botão direito e em *Add SQL Server*.

No menu **Local**, escolha a instância **MSSQLLocalDB**, em seguida clique em **Connect**.



12. Com a instância do banco carregada, abra a pasta *Database* e verifique se a base de dados *ProductsEntityModel*.*ProductsModel* foi criada. Caso sim, abra a pasta *Tables* e verifique se as tabelas *Products* e *Stocks* foram criadas. Observe que a tabela *_MigrationHistory* foi criada mantendo o histórico das atualizações de migração.

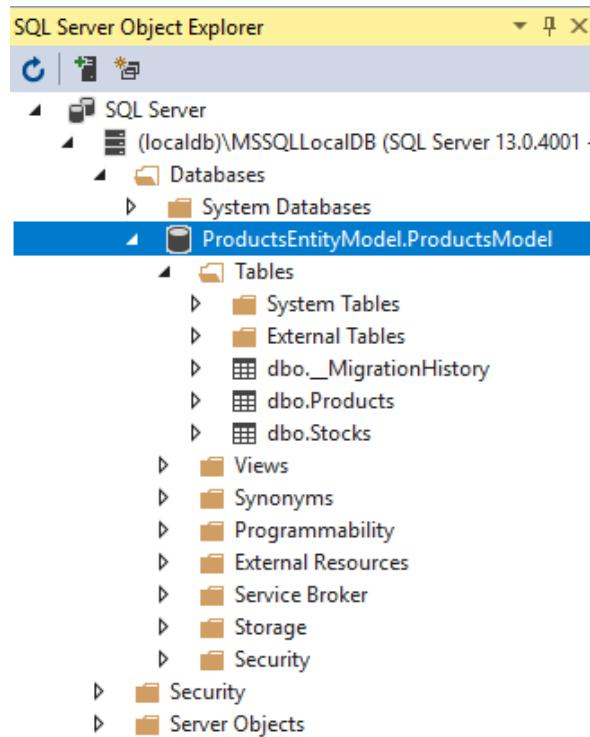


Figura 21 – Banco de dados *ProductsModel* criado

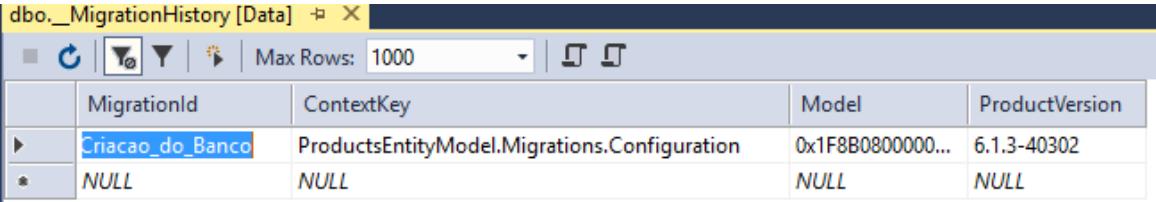
13. Clique com o botão direito sobre as tabelas e em *View Data*. Verifique se as tabelas estão populadas com os dados criados no método *Seed*.

	Id	Name	Code	Price
▶	1	Produto 01	0001	100,00
	2	Produto 02	0002	150,00
	3	Produto 03	0003	200,00
	4	Produto 04	0004	250,00
*	NULL	NULL	NULL	NULL

Figura 22 – Tabela Products

	Id	Quantity	LastUpdate	Product_Id
▶	1	1000,00	05/09/2017 10:09:49	1
	2	2000,00	05/09/2017 10:09:49	2
	3	3000,00	05/09/2017 10:09:49	3
	4	4000,00	05/09/2017 00:09:49	4
*	NULL	NULL	NULL	NULL

Figura 23 – Tabela Stocks



	MigrationId	ContextKey	Model	ProductVersion
▶	Criacao_do_Banco	ProductsEntityModel.Migrations.Configuration	0x1F8B0800000...	6.1.3-40302
*	NULL	NULL	NULL	NULL

Figura 24 – Tabela MigrationHistory

14. Se todos os passos anteriores foram concluídos com sucesso, clique em *Build* >*Build Solution* para compilar o projeto.

[DICA]

Para compilar o projeto, utilize as teclas Ctrl + Shift + B

Com o modelo de entidades pronto, é possível agora iniciar a criação do serviço que irá fornecer as operações necessárias para acessar a base de dados *ProductsModel*.

PRÁTICA 02 - Criação do projeto *ProductsService*

Na solução *ProductsService*, o serviço chamado *ProductsService* será criado para ser usado como parte de uma página *web*. A criação de um aplicativo para hospedagem será apresentada em um capítulo subsequente. Neste momento, o serviço será hospedado no servidor *IIS Express*, que vem junto à ferramenta Visual Studio. O *endpoint* para acesso ao serviço não será criado neste momento. Porém, ao criar o projeto utilizando o template do WCF, uma configuração default de *endpoint* será utilizada. Esta configuração consiste nos três componentes do *endpoint*:

- *address* é a URL para acesso ao serviço, que neste caso é a seguinte: **http://localhost:62245/Service.svc**. A porta 62245 é gerada aleatoriamente dependendo da máquina;
- *binding* especifica o mecanismo de transporte utilizado para acessar o serviço e o protocolo utilizado. Neste caso o *binding* configurado por default é o **basicHttpBinding**, que utilizada o protocolo HTTP como transporte e o SOAP como protocolo para troca de informações;
- *contract* especifica o contrato que o serviço implementa, que no caso é a interface do serviço *ProductsService.IProductsService*.

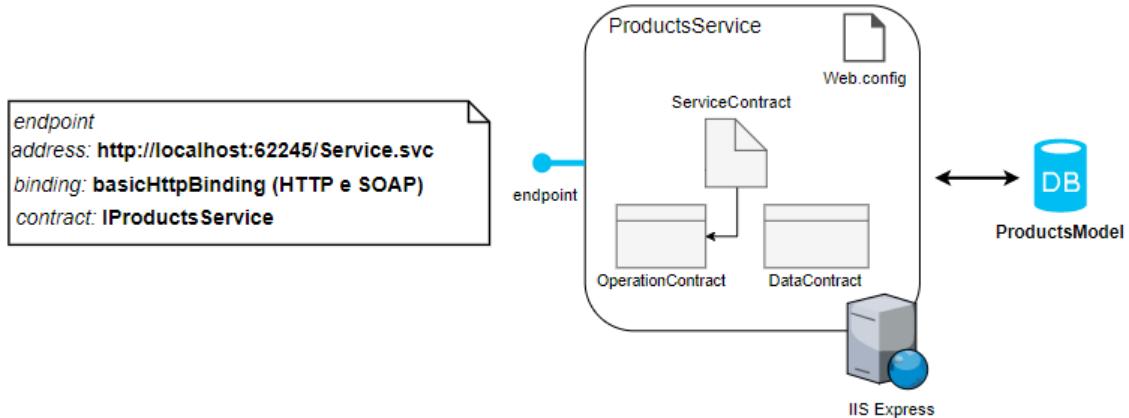


Figura 25 - Projeto ProductsService

1. Clique com o botão direito sobre a solução *ProductsService*, *Add* e clique em *New Web Site*
2. Selecione o *template WCF Service* e em *Web location*, aponte para a pasta *ProductsService* (que será criada neste momento) dentro da solução *ProductsService*, conforme a Figura 26.

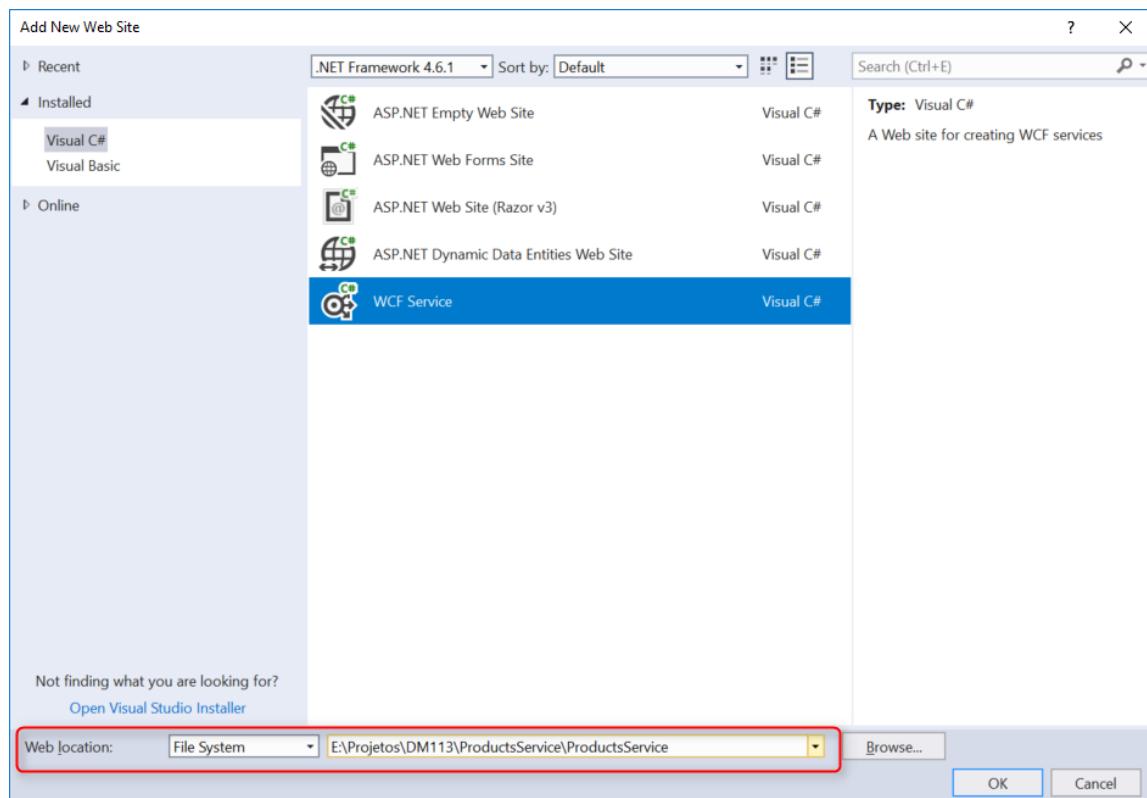


Figura 26 - Criação do ProductsService

Assim que o serviço é criado, um código de serviço simples é mostrado na classe *Service.cs*. Duas operações do serviço são implementadas, cujos retornos são do tipo *string* e do tipo *CompositeType*.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the class name "Service" in code, svc and config file
public class Service : IService
{
    public string GetData(int value)
    {
        return string.Format("You entered: {0}", value);
    }

    public CompositeType GetDataUsingDataContract(CompositeType composite)
    {
        if (composite == null)
        {
            throw new ArgumentNullException("composite");
        }
        if (composite.BoolValue)
        {
            composite.StringValue += "Suffix";
        }
        return composite;
    }
}

```

Service

Use the dropdown to view and navigate to other items in this file.

Figura 27 – Implementação do contrato de serviço

3. No *Solution Explorer*, dentro de *ProductsService\App_Code*, abra o arquivo */Service.cs*, como mostrado na Figura 28.

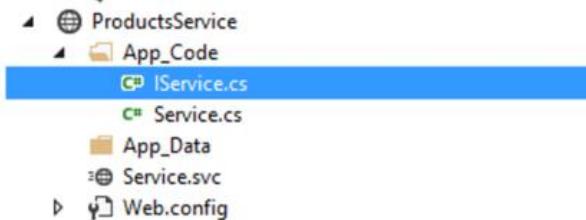


Figura 28 – Interface do serviço

4. Verifique que esta *interface* define os métodos implementados na classe *Service.cs*, através de contrato de serviço, de operações e de dados utilizando os atributos *[ServiceContract]*, *[OperationContract]* e *[DataContract]* respectivamente.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface name "IService" in both code and config.
[ServiceContract]
public interface IService
{
    [OperationContract]
    string GetData(int value);

    [OperationContract]
    CompositeType GetDataUsingDataContract(CompositeType composite);

    // TODO: Add your service operations here
}

```

Figura 29 – Definição do ServiceContract e OperationContract

```

// Use a data contract as illustrated in the sample below to add composite types to service operations.
[DataContract]
public class CompositeType
{
    bool boolValue = true;
    string stringValue = "Hello ";

    [DataMember]
    public bool BoolValue
    {
        get { return boolValue; }
        set { boolValue = value; }
    }

    [DataMember]
    public string StringValue
    {
        get { return stringValue; }
        set { stringValue = value; }
    }
}

```

Figura 30 – Definição do DataContract

Definição do Contrato de Dados

Um exemplo de contrato de dados [DataContract] é mostrado na classe *CompositeType*, dentro da interface *IService*. Os contratos de dados são tipos de dados de um serviço WCF e devem ser descritos em metadados para garantir a interoperabilidade com outras aplicações. Um contrato de dados pode ser utilizado por uma operação de contrato [OperationContract] como um parâmetro ou um tipo de retorno. A seguir, o contrato de dados do exemplo gerado será alterado para corresponder ao formato dos dados do banco *ProductsModel*, gerado anteriormente. Ele irá especificar os seguintes detalhes do produto e estoque que o serviço poderá passar para as aplicações clientes, tais como:

Produto

- nome
- código
- preço

Estoque

- quantidade
- última atualização

1. No *Solution Explorer*, renomeie o arquivo *IService.cs* para *IProductsService.cs*
2. Apague o código de *IProductsService.cs*, deixando apenas as cláusulas *using*
3. Após as cláusulas *using*, adicione o *namespace Products* e o contrato de dados *ProductData*, conforme o código a seguir:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace Products
{
    // Data contract describing the details of a product passed to
    // client applications
    [DataContract]
    public class ProductData
    {
        [DataMember]
        public string Name;

        [DataMember]
        public string Code;

        [DataMember]
        public decimal Price;
    }
}
```

Definição do Contrato de Serviço

O contrato de serviço *[ServiceContract]* é a interface do serviço WCF. Ela irá descrever as operações que serão fornecidas pelo serviço WCF para que as aplicações clientes tenham acesso. Cada método que será exposto por este serviço deverá ser marcado com o atributo *[OperationContract]*. As operações seguintes serão criadas para ter acesso às informações dos Produtos da base de dados *ProductsModel*:

- Buscar todos os produtos
- Buscar os detalhes do produto
- Buscar o estoque atual do produto
- Atualizar o estoque do produto

1. Adicione o contrato de serviço *IProductsService*, a seguir, à *namespace Products*

```
namespace Products
```

```
{  
    // Service contract describing the operations provided by the WCF service  
    [ServiceContract]  
    public interface IProductsService  
    {  
        // Get all products  
        [OperationContract]  
        List<ProductData> ListProducts();  
  
        // Get the details of a single product  
        [OperationContract]  
        ProductData GetProduct(string productCode);  
  
        // Get the current stock for a product  
        [OperationContract]  
        int CurrentStock(string productCode);  
  
        // Add stock for a product  
        [OperationContract]  
        bool AddStock (string productCode, decimal quantity);  
    }  
}
```

[DICA]

Para identar o código, utilize as teclas Ctrl + K + D

Implementação do Serviço

1. No *Solution Explorer*, clique com o botão direito no projeto *ProductsService* e depois clique em *Add Reference*
2. Clique em *Projects* e *Solution*
3. Marque a *check box* *ProductsEntityModel*, como na Figura 14, e clique *OK*

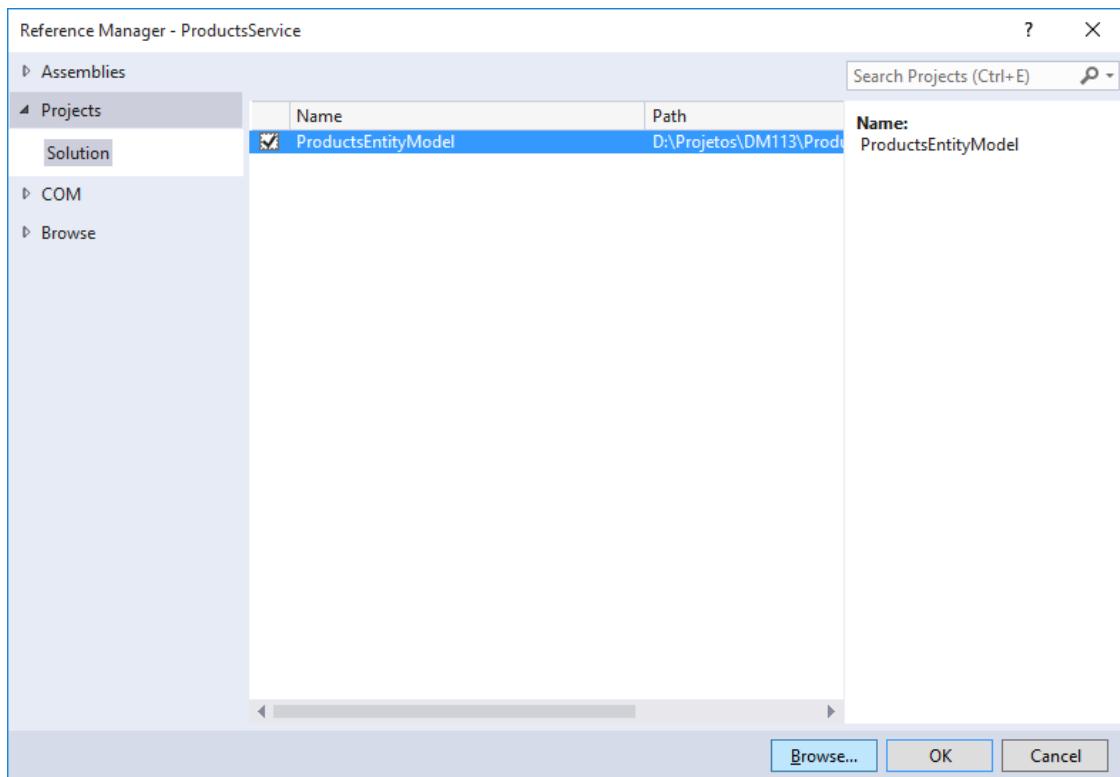


Figura 31 - Adição de referência ao projeto *ProductsService*

[ATENÇÃO]

Ao importar o projeto *ProductsEntityModel*, observe que na pasta *bin* do projeto *ProductsService* foi adicionada a biblioteca *ProductsEntityModel.dll*.

4. Usando o mesmo procedimento dos itens 1-3, adicione a referência para *System.Data.Entity* clicando em *Assemblies* e *Framework*, como mostrado na Figura 32.

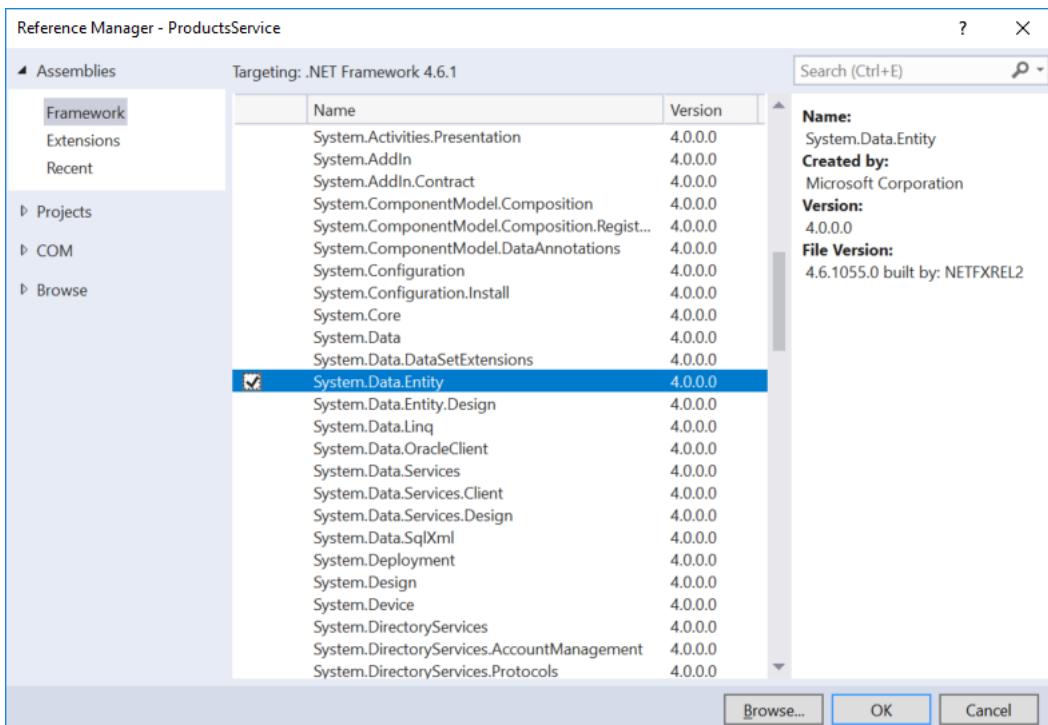


Figura 32 – Referência para *System.Data.Entity*

5. Renomeie o arquivo *Service.cs* para *ProductsService.cs*
6. Abra o arquivo *ProductsService.cs* e apague todo o código e comentários, deixando apenas as cláusulas *using*
7. Adicione as cláusulas *using* abaixo e a seguir adicione o *namespace Products*

```
using ProductsEntityModel;
using System.ServiceModel.Activation;
```

```
namespace Products
{
}
```

8. Adicione a classe a seguir ao *namespace*:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using ProductsEntityModel;
using System.ServiceModel.Activation;
```

```
namespace Products
{
```

```

// WCF service that implements the service contract
// This implementation performs minimal error checking and exception handling
[AspNetCompatibilityRequirements(
    RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]
public class ProductsService : IProductsService
{
    }

}

```

Verifique que a classe implementa o contrato de serviço *IProductsService*. A classe *AspNetCompatibilityRequirements* indica que o serviço WCF pode ser executado em modo de compatibilidade com o ASP.NET.

9. Implemente o método *ListProducts*, abaixo, à classe *ProductsService*:

```

public List<ProductData> ListProducts()
{
    // Create a list of products
    List<ProductData> productsList = new List<ProductData>();
    try
    {
        // Connect to the ProductsModel database
        using (ProductsModel database = new ProductsModel())
        {
            // Fetch the products in the database
            List<Product> products = (from product in database.Product select
product).ToList();

            foreach(Product product in products)
            {
                ProductData productData = new ProductData()
                {
                    Name = product.Name,
                    Code = product.Code,
                    Price = product.Price
                };
                productsList.Add(productData);
            }
        }
    catch
    {
        // Ignore exceptions in this implementation
    }
    // Return the list of products
    return productsList;
}

```

10. Implemente o método *GetProduct*, abaixo, à classe *ProductsService*:

```

public ProductData GetProduct(string productCode)
{
    ProductData productData = null;
    try
    {

```

```

        // Connect to the ProductsModel database
        using (ProductsModel database = new ProductsModel())
        {
            // Find the first product that matches the specified product code
            Product matchingProduct = database.Product.First(
                p => String.Compare(p.Code, productCode) == 0);
            productData = new ProductData()
            {
                Name = matchingProduct.Name,
                Code = matchingProduct.Code,
                Price = matchingProduct.Price
            };
        }
    }
    catch
    {
        // Ignore exceptions in this implementation
    }

    // Return the product
    return productData;
}

```

11. Implemente o método *CurrentStock*, abaixo, à classe *ProductsService*:

```

public int CurrentStock(string productCode)
{
    int quantityTotal= 0;
    try
    {
        // Connect to the ProductsModel database
        using (ProductsModel database = new ProductsModel())
        {
            // Calculate the sum of all quantities for the specified product
            quantityTotal = (from s in database.Stock
                            join p in database.Product
                            on s.ProductId equals p.Id
                            where String.Compare(p.Code, productCode) == 0
                            select (int)s.Quantity).Sum();
        }
    }
    catch
    {
        // Ignore exceptions in this implementation
    }
    // Return the stock level
    return quantityTotal;
}

```

12. Implemente o método *AddStock*, abaixo, à classe *ProductsService*:

```

public bool AddStock(string productCode, decimal quantity)
{
    try
    {
        // Connect to the ProductsModel database
        using (ProductsModel database = new ProductsModel())
        {

            // Find the ProductID for the specified product
            int productID = (from p in database.Product

```

```

        where String.Compare(p.Code, productCode) == 0
        select p.Id).First();

    // Find the Stock object that matches the parameters passed
    // in to the operation
    Stock stock = database.Stock.First(pi => pi.Id == productID);

    stock.Quantity = quantity;
    stock.lastUpdate = DateTime.Now;

    database.Stock.Add(stock);

    // Save the change back to the database
    database.SaveChanges();
}
}
catch
{
    // If an exception occurs, return false to indicate failure
    return false;
}

// Return true to indicate success
return true;
}

```

13. Atualize a versão do *EntityFramework* clicando com o botão direito sobre o projeto e em *Manage NuGet Packages*. No campo de filtro, procure por *Entity Framework* e em seguida instale a última versão.
14. Compile o projeto em *Build->Build Solution*.

Configuração do Serviço

Como o serviço foi criado como uma página *web*, ele será hospedado no *IIS Express*, instalado com o *Visual Studio*. Apesar de a maioria das configurações assumirem valores padrão quando o serviço é criado, pode ser necessário sobrescrever algumas delas para atingir um requisito específico. Uma configuração mínima será realizada a seguir, garantindo que um navegador consiga acessar o banco de dados através do serviço.

1. No *Solution Explorer*, abra o arquivo *Service.svc* e altere-o conforme a seguir:

```
<%@ ServiceHost Language="C#" Debug="true" Service="Products.ProductsService"
CodeBehind="~/App_Code/ProductsService.cs" %>
```

2. No projeto *ProductsEntityModel*, abra o arquivo *App.Config* e copie a string de conexão que foi gerada pelo *Migrations* para a conexão ao banco de dados *ProductsModel*, como mostrado na Figura 33.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3      <configSections>
4          <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5          <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b71d1f8932418a08" />
6      </configSections>
7      <connectionStrings>
8          <add name="ProductsModel" connectionString="data source=(LocalDb)\MSSQLLocalDB;initial catalog=ProductsEntityModel.ProductsModel;integrated security=true;multipleactiveresultsets=true;pooling=true" />
9      </connectionStrings>
10     <entityFramework>
11         <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />
12         <providers>
13             <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
14         </providers>
15     </entityFramework>
16 </configuration>

```

Figura 33 – String de conexão do *App.config*

3. No projeto *ProductsService*, abra o arquivo *Web.config* e, dentro da seção *<configuration>*, na linha anterior a *<system.web>*, adicione a string de conexão copiada do arquivo de configuração *App.config*, como mostrado na Figura 34.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3      <configSections>
4          <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5          <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b71d1f8932418a08" />
6      </configSections>
7      <appSettings>
8          <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
9      </appSettings>
10     <connectionStrings>
11         <add name="ProductsModel" connectionString="data source=(LocalDb)\MSSQLLocalDB;initial catalog=ProductsEntityModel.ProductsModel;integrated security=true;multipleactiveresultsets=true;pooling=true" />
12     </connectionStrings>
13     <system.web>
14         <compilation debug="false" targetFramework="4.6.1">
15             <assemblies>
16                 <add assembly="System.Data.Entity, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />
17             </assemblies>
18         </compilation>
19         <httpRuntime targetFramework="4.6.1" />
20     </system.web>

```

Figura 34 – String de conexão do *Web.config*

Teste do Serviço usando um Browser

1. No projeto *ProductsService*, clique com o botão direito sobre *Service.svc* e clique em *View in Browser* para abrir a página mostrada na Figura 35.

ProductsService Serviço

Você criou um serviço.

Para testar esse serviço, você precisará criar um cliente e usá-lo para chamar o serviço. Você pode fazer isso usando a ferramenta svcutil.exe na linha de comando com a seguinte sintaxe:

```
svcutil.exe http://localhost:62245/Service.svc?wsdl
```

Você também pode acessar a descrição do serviço como um único arquivo:

```
http://localhost:62245/Service.svc?singleWSDL
```

Isto gerará um arquivo de configuração e um arquivo de código que contém a classe cliente. Adicione os dois arquivos ao seu aplicativo cliente e use a classe cliente gerada para chamar o Serviço. Por exemplo:

```
C#
class Test
{
    static void Main()
    {
        ProductsServiceClient client = new ProductsServiceClient();

        // Use a variável 'client' para chamar as operações no serviço.

        // Sempre feche o cliente.
        client.Close();
    }
}
```

Figura 35 - Página Web do serviço criado

2. Abra o *link* do primeiro quadro no navegador para ver os metadados que descrevem o serviço. Como mostrado na Figura 37, a página aberta corresponde a um documento XML que utiliza o esquema WSDL. Os elementos deste documento descrevem as operações expostas do serviço WCF criado e as mensagens que ele pode enviar e receber.

svutil.exe <http://localhost:62245/Service.svc?wsdl>

Figura 36 – URL do documento do serviço

```

<?xml version="1.0" encoding="UTF-8"?>
- <wsdl:definitions xmlns:wsan="http://www.w3.org/2007/05/addressing/metadata" xmlns:wsa10="http://www.w3.org/2005/08/addressing"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
  xmlns:wsav="http://www.w3.org/2006/05/addressing/ws1" xmlns:wap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
  xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:tns="http://tempuri.org/" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://tempuri.org/" name="ProductsService">
  - <wsdl:types>
    - <xsd:schema targetNamespace="http://tempuri.org/Imports">
      <xsd:import namespace="http://tempuri.org/" schemaLocation="http://localhost:62245/Service.svc?xsd=xsd0"/>
      <xsd:import namespace="http://schemas.microsoft.com/2003/10/Serialization/" schemaLocation="http://localhost:62245/Service.svc?xsd=xsd1"/>
      <xsd:import namespace="http://schemas.datacontract.org/2004/07/Products" schemaLocation="http://localhost:62245/Service.svc?xsd=xsd2"/>
    </xsd:schema>
  - <wsdl:types>
  - <wsdl:message name="IProductsService_ListProducts_InputMessage">
    <wsdl:part name="parameters" element="tns>ListProducts"/>
  </wsdl:message>
  - <wsdl:message name="IProductsService_ListProducts_OutputMessage">
    <wsdl:part name="parameters" element="tns>ListProductsResponse"/>
  </wsdl:message>
  - <wsdl:message name="IProductsService_GetProduct_InputMessage">
    <wsdl:part name="parameters" element="tns:GetProduct"/>
  </wsdl:message>
  - <wsdl:message name="IProductsService_GetProduct_OutputMessage">
    <wsdl:part name="parameters" element="tns:GetProductResponse"/>
  </wsdl:message>
  - <wsdl:message name="IProductsService_CurrentStock_InputMessage">
    <wsdl:part name="parameters" element="tns:CurrentStock"/>
  </wsdl:message>
  - <wsdl:message name="IProductsService_CurrentStock_OutputMessage">
    <wsdl:part name="parameters" element="tns:CurrentStockResponse"/>
  </wsdl:message>
  - <wsdl:message name="IProductsService_AddStock_InputMessage">
    <wsdl:part name="parameters" element="tns:AddStock"/>
  </wsdl:message>

```

Figura 37 - Documento XML utilizando o esquema WSDL

PRÁTICA 03 - Teste do serviço com o cliente de teste do WCF

Para realizar um teste rápido do serviço WCF criado, o Visual Studio fornece um Cliente de Teste do WCF (WcfTestClient.exe). Esta ferramenta permite ao usuário entrar com parâmetros de teste, enviar essa entrada ao serviço e exibir a resposta retornada pelo serviço.

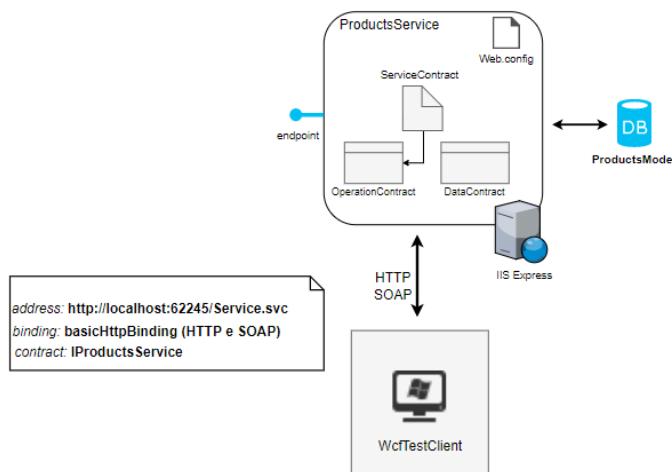


Figura 38 – Testando o serviço com o WcfTestClient

O cliente de teste (WcfTestClient.exe) pode ser localizado no seguinte diretório:

C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE

1. Execute o cliente de teste localizado no diretório citado.
2. Na tela do cliente, clique com o botão direito em *My Service Projects*, e em *Add Service*.
3. Entre com o endereço do serviço, por exemplo **http://localhost:62245/Service.svc**
4. Observe que o serviço foi localizado e as operações estão disponíveis para acesso, como mostrado na Figura 39.

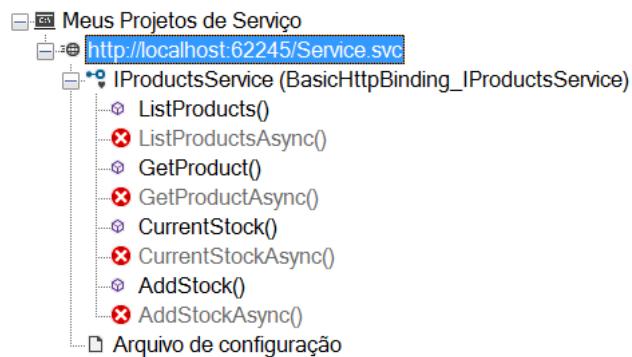


Figura 39 – Operações do serviço localizado

5. Clique duas vezes na operação *ListProducts()*. Na tela de *Request*, clique no botão *Invoke* para requisitar esta operação ao serviço.
6. A tela de *Response* irá apresentar o resultado da solicitação, como mostrado na Figura 40.

Resposta			<input type="checkbox"/> Iniciar um novo proxy	<input type="button"/> Invocar
Nome	Valor	Tipo		
▲ (return)	length=4	Products.ProductData[]		
▲ [0]		Products.ProductData		
Code	"0001"	System.String		
Name	"Produto 01"	System.String		
Price	100,00	System.Decimal		
▲ [1]		Products.ProductData		
Code	"0002"	System.String		
Name	"Produto 02"	System.String		
Price	150,00	System.Decimal		
▲ [2]		Products.ProductData		
Code	"0003"	System.String		
Name	"Produto 03"	System.String		
Price	200,00	System.Decimal		
▲ [3]		Products.ProductData		
Code	"0004"	System.String		
Name	"Produto 04"	System.String		
Price	250,00	System.Decimal		
Formatado <input type="button"/> XML				

Figura 40 – Resposta da operação *ListProducts()* do serviço

7. Clique na aba XML, localizada abaixo da tela de *Response*.

8. Observe a utilização do protocolo SOAP no formato das mensagens de *Request* e *Response*.

Solicitação

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1" xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">http://tempuri.org/IProductsService/ListProducts</Action>
  </s:Header>
  <s:Body>
    <ListProducts xmlns="http://tempuri.org/" />
  </s:Body>
</s:Envelope>
```

Resposta

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header />
  <s:Body>
    <ListProductsResponse xmlns="http://tempuri.org/">
      <ListProductsResult xmlns:a="http://schemas.datacontract.org/2004/07/Products" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <a:ProductData>
          <a:Code>0001</a:Code>
          <a:Name>Produto 01</a:Name>
          <a:Price>100.00</a:Price>
        </a:ProductData>
        <a:ProductData>
          <a:Code>0002</a:Code>
          <a:Name>Produto 02</a:Name>
          <a:Price>150.00</a:Price>
        </a:ProductData>
        <a:ProductData>
          <a:Code>0003</a:Code>
          <a:Name>Produto 03</a:Name>
          <a:Price>200.00</a:Price>
        </a:ProductData>
        <a:ProductData>
          <a:Code>0004</a:Code>
          <a:Name>Produto 04</a:Name>
          <a:Price>250.00</a:Price>
        </a:ProductData>
      </ListProductsResult>
    </ListProductsResponse>
```

Formatado XML

Figura 41 – Protocolo SOAP

PRÁTICA 04 - Criação da aplicação Cliente

Como visto na seção anterior, o serviço *ProductsService* foi criado, expondo alguns métodos que trazem informações sobre a tabela *Products* da base de dados *ProductsModel*. Nesta seção será criada uma aplicação cliente para o teste do serviço *ProductsService*.

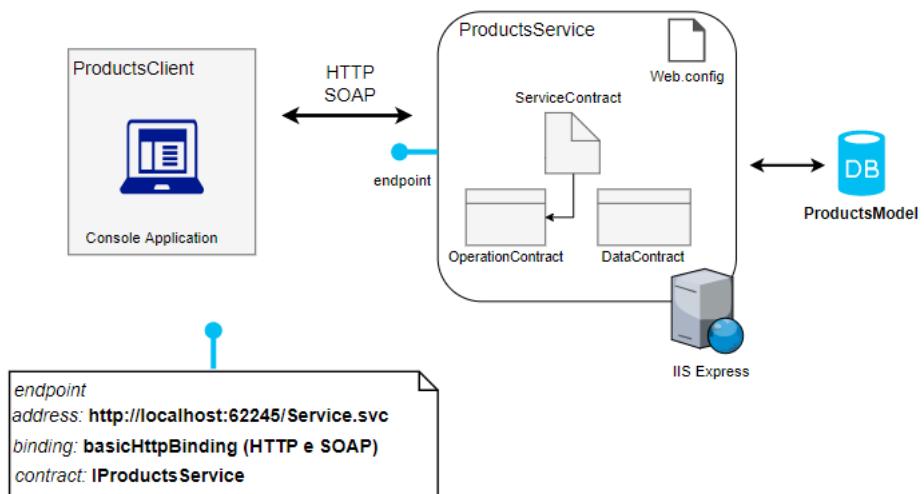


Figura 42 - Criação da aplicação Cliente

Como foi observado anteriormente, os metadados que descrevem o serviço WCF correspondem a um documento XML e podem ser acessados através de uma URL com extensão ?wsdl. O Visual Studio pode utilizar estes metadados publicado pelo serviço para gerar uma classe *proxy* que a aplicação cliente pode usar para se conectar ao serviço. Esta classe *proxy* fornece as mesmas operações e mensagens que o serviço WCF, tornando-as disponíveis em forma de métodos, operações e valores de retorno.

[ATENÇÃO]

O termo *proxy* aqui chamado se refere a uma classe cliente que possui os métodos que serão utilizados para chamar o serviço.

Para criar a aplicação cliente, siga os seguintes passos:

1. Adicione um novo projeto para a solução *ProductsService*. No *Solution Explorer*, clique com o botão direito sobre a solução e depois em *Add New Project*.
2. Crie o projeto de nome *ProductsClient* usando o *template Console Application* como mostrado na Figura 43.

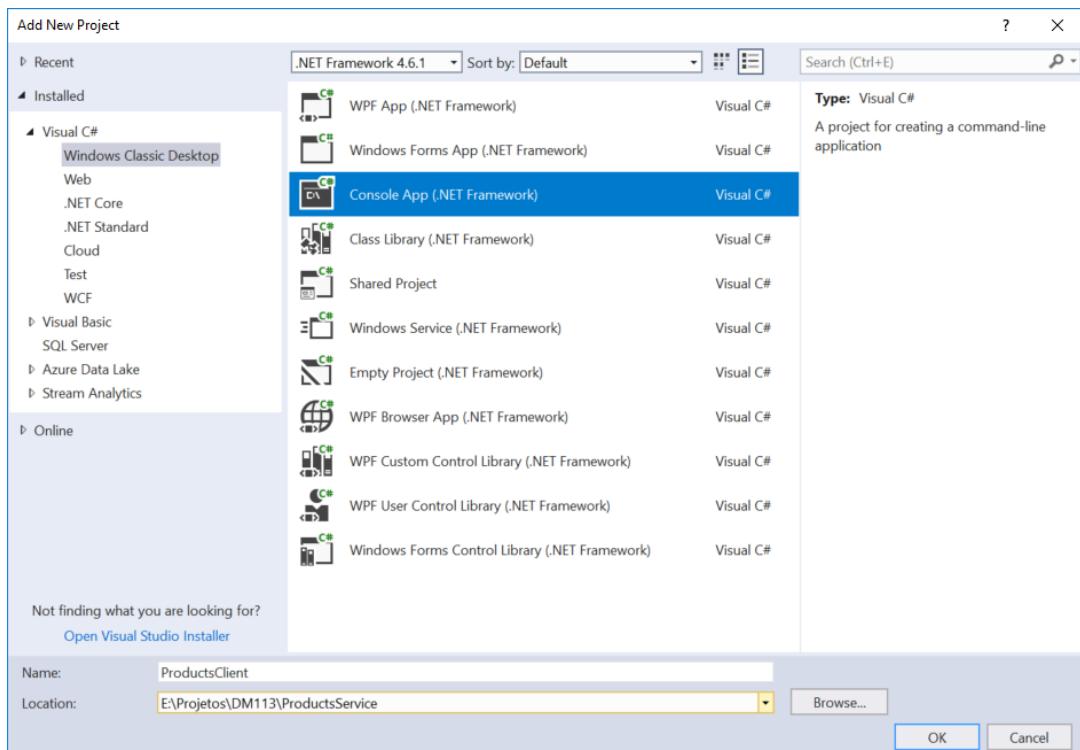


Figura 43 – Criação da aplicação cliente

3. Clique com o botão direito sobre o projeto criado e em *Add Reference*.
4. Dentro de *Assemblies\Framework*, marque a *check box* de *System.ServiceModel*
5. Novamente, clique com o botão direito sobre o projeto e, então, em *Add Service Reference*.
6. Clique no botão *Discover* para que o serviço *ProductsService* hospedado no IIS Express seja mostrado, como na Figura 44. Ao encontrar o serviço *Service.svc*, expanda-o e então clique sobre *IProductsService*. Observe a lista de operações que são expostas pelo serviço. Digite *ProductsService* no campo *Namespace* e clique *OK*.

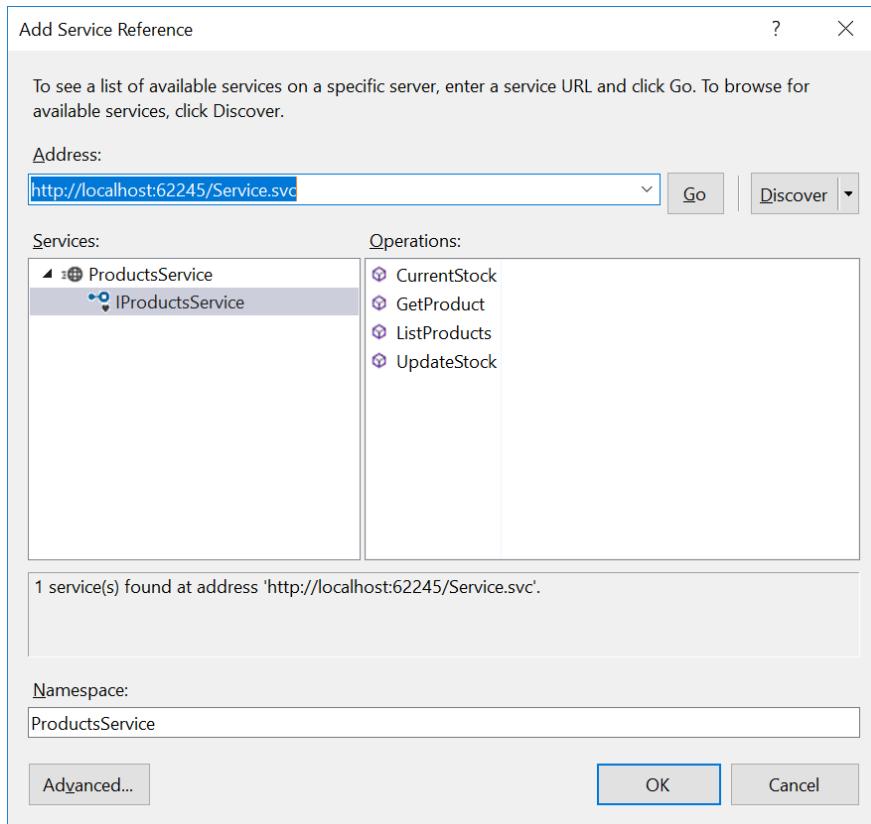


Figura 44 - Adição de uma referência ao serviço

7. No *Solution Explorer*, clique sobre o arquivo *App.config* e veja as características do elemento `<endpoint>` criado após a tag `<client>`. O *endpoint* especifica como a aplicação cliente irá se conectar ao serviço. Ele possui basicamente três informações importantes: *address*, *binding* e *contract*.

O *address* é a URL para acesso ao serviço, que neste caso é a seguinte:
http://localhost:62245/Service.svc

O *binding* especifica o mecanismo de transporte utilizado para acessar o serviço e o protocolo utilizado. Neste caso o *binding* configurado por default é o **basicHttpBinding**, que utiliza o protocolo HTTP.

O *contract* especifica o contrato que o serviço implementa, que no caso é a interface do serviço `ProductsService.IProductsService`

```

<client>
  <endpoint address="http://localhost:62245/Service.svc" binding="basicHttpBinding"
    bindingConfiguration="BasicHttpBinding_IProductsService" contract="ProductsService.IProductsService"
    name="BasicHttpBinding_IProductsService" />
</client>

```

Figura 45 – Endpoint do serviço

8. Abra o arquivo *Program.cs* e adicione as seguintes cláusulas *using*:

```

using System.ServiceModel;
using ProductsClient.ProductsService;

```

9. Acrescente o código a seguir ao método *Main* para criar uma instância da classe *proxy* que se conecta ao serviço:

```
static void Main(string[] args)
{
    // Create a proxy object and connect to the service
    ProductsServiceClient proxy = new ProductsServiceClient();
```

10. Acrescente o código a seguir ao método *Main* para testar a operação *ListProducts* do serviço:

```
// Test the operations in the service
// Obtain a list of all products
Console.WriteLine("Test 1: List all products");
List<ProductData> products = proxy.ListProducts().ToList();
foreach (ProductData p in products)
{
    Console.WriteLine("Name: {0}", p.Name);
    Console.WriteLine("Code: {0}", p.Code);
    Console.WriteLine("Price: {0}", p.Price);
    Console.WriteLine();
}
Console.WriteLine();
```

11. Acrescente o código a seguir ao método *Main* para testar a operação *GetProduct* do serviço:

```
// Get details of this product
Console.WriteLine("Test 2: Display the details of a product");
ProductData product = proxy.GetProduct("0001");
Console.WriteLine("Name: {0}", product.Name);
Console.WriteLine("Code: {0}", product.Code);
Console.WriteLine("Price: {0}", product.Price);
Console.WriteLine();
```

12. Acrescente o código a seguir ao método *Main* para testar a operação *CurrentStock* do serviço:

```
// Query the stock of this product
Console.WriteLine("Test 3: Display stock of a product");
int quantity = proxy.CurrentStock("0001");
Console.WriteLine("Current stock: {0}", quantity);
Console.WriteLine();
```

13. Acrescente o código a seguir ao método *Main* para testar a operação *AddStock* do serviço:

```
// Add stock of this product
Console.WriteLine("Test 4: Add stock for a product");
if (proxy.AddStock("0001", 100))
{
    quantity = proxy.CurrentStock("0001");
    Console.WriteLine("Stock changed. Current stock: {0}", quantity);
}
```

```

else
{
    Console.WriteLine("Stock update failed");
}
Console.WriteLine();

```

14. Acrescente o código a seguir ao método *Main* para se desconectar do serviço:

```

// Disconnect from the service
proxy.Close();
Console.WriteLine("Press ENTER to finish");
Console.ReadLine();

```

15. Salve e compile o projeto.

Execução da Aplicação Cliente

1. No *Solution Explorer*, clique com o botão direito na solução *ProductsService* e clique em *Set StartUp Projects*
2. Selecione *Multiple startup projects* e escolha *Start* para os projetos *ProductsClient* e *ProductsService*, como na Figura 46.

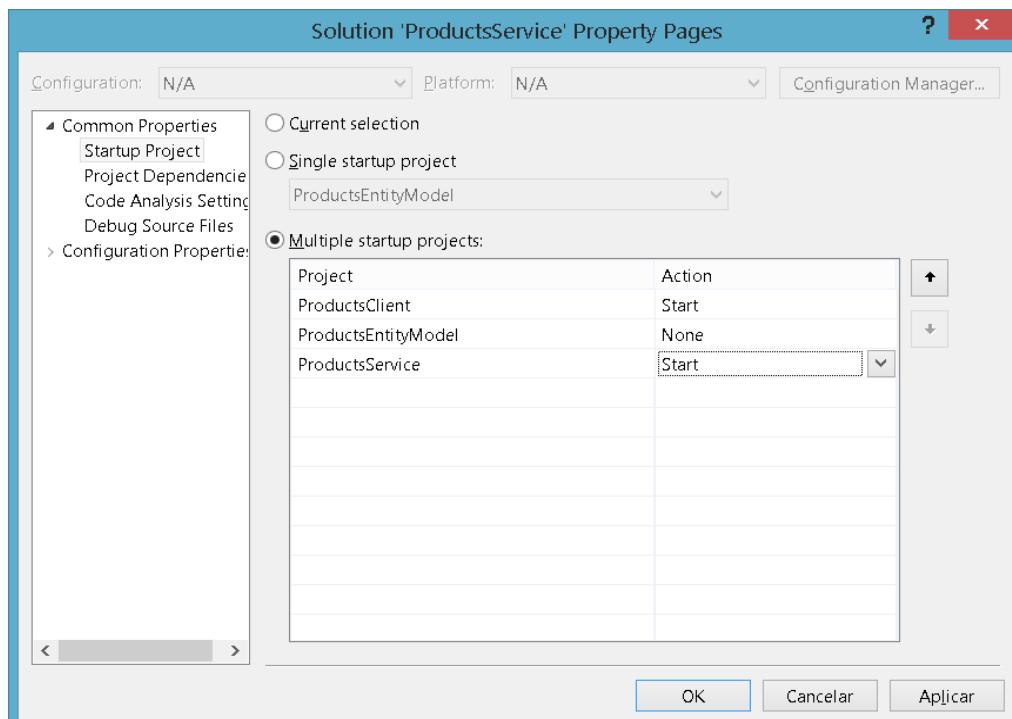


Figura 46 - Configurando a Execução dos Projetos contidos na Solução

3. No *Solution Explorer*, clique com o botão direito sobre o projeto *ProductsService* e clique em *Start options*
4. Selecione o *radio button* *Don't open a page. Wait for a request from an external application* e clique *OK*

5. Pressione *Ctrl+F5* para iniciar a execução sem *debug*, o que deve gerar o resultado da Figura 47.

```
ca: C:\Windows\system32\cmd.exe
Name: Produto 01
Code: 0001
Price: 100,00

Name: Produto 02
Code: 0002
Price: 150,00

Name: Produto 03
Code: 0003
Price: 200,00

Name: Produto 04
Code: 0004
Price: 250,00

Test 2: Display the details of a product
Name: Produto 01
Code: 0001
Price: 100,00

Test 3: Display stock of a product
Current stock: 1000

Test 4: Add stock for a product
Stock changed. Current stock: 1100

Press ENTER to finish
```

Figura 47 - Execução da Aplicação Cliente

6. Pressione uma tecla para terminar a execução

3. Hospedagem de Serviços WCF

Um serviço WCF é um objeto que disponibiliza operações que podem ser acessadas por clientes. Para que seja acessado, o serviço deve estar hospedado em um ambiente de execução, o qual deve ser capaz de iniciar e interromper o serviço, receber requisições e encaminhá-las ao serviço e receber respostas do serviço e enviá-las de volta ao cliente. Essas funcionalidades *host* são disponibilizadas através de *endpoints*.

Os *endpoints* contêm, basicamente, três informações: o endereço do serviço, o *binding* suportado pelo serviço e o contrato implementado pelo serviço. Como o endereço e o contrato já foram discutidos anteriormente, resta explicar o *binding*. Um *binding* especifica o método de conexão que deve ser utilizado entre o cliente e o serviço e, também, o formato dos dados que o serviço espera. Quando um *endpoint* recebe uma requisição, um canal é criado usando as configurações especificadas pelo *binding* para encaminhá-la para o serviço.

Como o serviço deve ser capaz de receber requisições de vários clientes simultaneamente, a configuração *InstanceContextMode* será criada, suportando três variações: *PerCall*, *PerSession* e *Single*. Na primeira configuração, cada requisição causará a criação de uma nova instância do serviço. Na segunda, uma instância permanecerá criada até que o cliente feche a sessão. E, na última, todos os clientes compartilham uma mesma

instância do serviço. A utilização do *InstanceContextMode* será demonstrada em um capítulo posterior.

Baseado nas funcionalidades apresentadas, esse capítulo explorará a utilização de serviços utilizando o HTTP e o TCP como transporte, o que requer diferentes métodos de hospedagem do serviço.

Hospedagem do Serviço no IIS Express

Os serviços WCF podem ser hospedados de várias formas. Como comentado anteriormente, o serviço criado foi hospedado no IIS Express, que é um servidor Web que a Microsoft fornece junto ao Visual Studio e permite que as aplicações sejam executadas na máquina local, sem a necessidade de um ambiente externo. O IIS Express é uma versão mais leve e otimizada para desenvolvedores. Este servidor web é usado por desenvolvedores para testes iniciais e tem funcionalidade semelhante à do servidor do Internet Information Services (IIS) do Windows, porém com algumas limitações, suportando apenas os protocolos HTTP e HTTPS.

Ao executar o serviço *ProductsService* clicando com o botão direito sobre *Service.svc* e em seguida em *View in Browser*, a página referente ao serviço será aberta. Você pode verificar o serviço rodando no servidor IIS Express, verificando o ícone IIS Express que se encontra oculto ao lado do relógio do Windows, conforme a figura a seguir.



Figura 48 – Ícone do IIS Express

Ao clicar com o botão direito sobre o ícone IIS Express, clique em *Mostrar todos os aplicativos*. Uma janela do IIS Express será aberta mostrando que o serviço *ProductsService* está sendo executado.

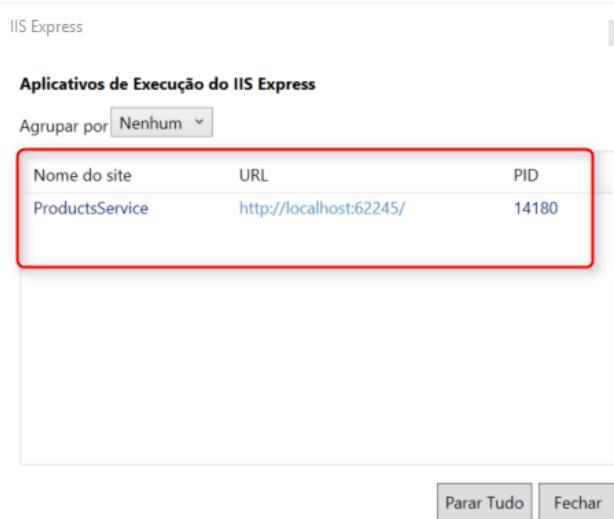


Figura 49 – Gerenciador do IIS Express

[ESTUDO COMPLEMENTAR]

Hospedagem do Serviço usando o *Internet Information Services*

O *Internet Information Services* (IIS) é um servidor de aplicativos Web para Windows, flexível, seguro e gerenciável para qualquer hospedagem na Web. O IIS é uma arquitetura escalável e aberta e está pronto para lidar com tarefas mais exigentes [IIS]. Para hospedar um serviço WCF para o IIS, é necessário a existência de um arquivo físico com a extensão *.svc [HTIIS]. Os passos para a hospedagem do *ProductsService* no servidor IIS do Windows se encontra detalhado no Apêndice A.

Hospedagem do Serviço em uma aplicação (self-hosting)

Outra forma de hospedar um serviço WCF é a hospedagem do serviço através de alguma aplicação (self-hosting), tais como Console Application, Windows Form, Windows Service e outras. Desta forma, o código do serviço ficará dentro do código do aplicativo. Para exemplificar este tipo de hospedagem, será criada uma aplicação WPF (Windows Presentation Foundation) onde o serviço *ProductsService* será hospedado.

Para que o serviço possa ser mais facilmente hospedado em uma aplicação, ele pode ser reconstruído em uma biblioteca.

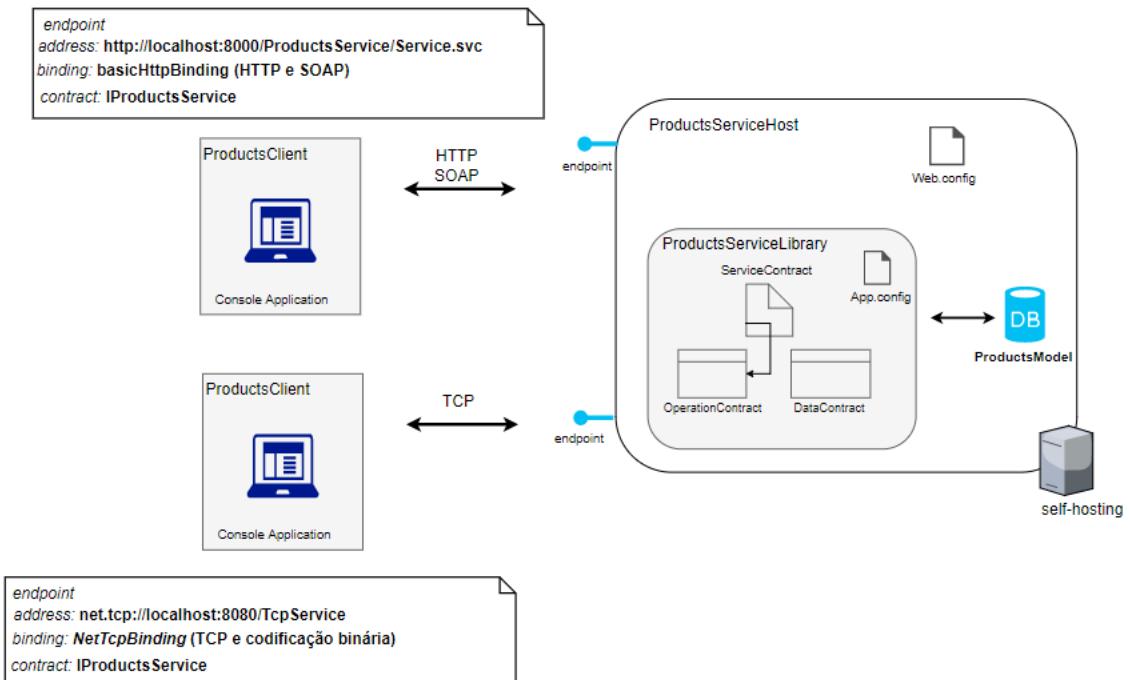


Figura 50 – Hospedagem do Serviço em uma aplicação

PRÁTICA 05 - Criação do projeto *ProductsServiceLibrary*

Reconstrução do *ProductsService* como uma Biblioteca

Um serviço pode ser mais facilmente incorporado a um aplicativo se for transformado em uma biblioteca, conforme apresentado a seguir.

1. Crie um novo projeto usando o modelo *WCF Service Library*, nomeando-o *ProductsServiceLibrary*, como mostrado na Figura 51.

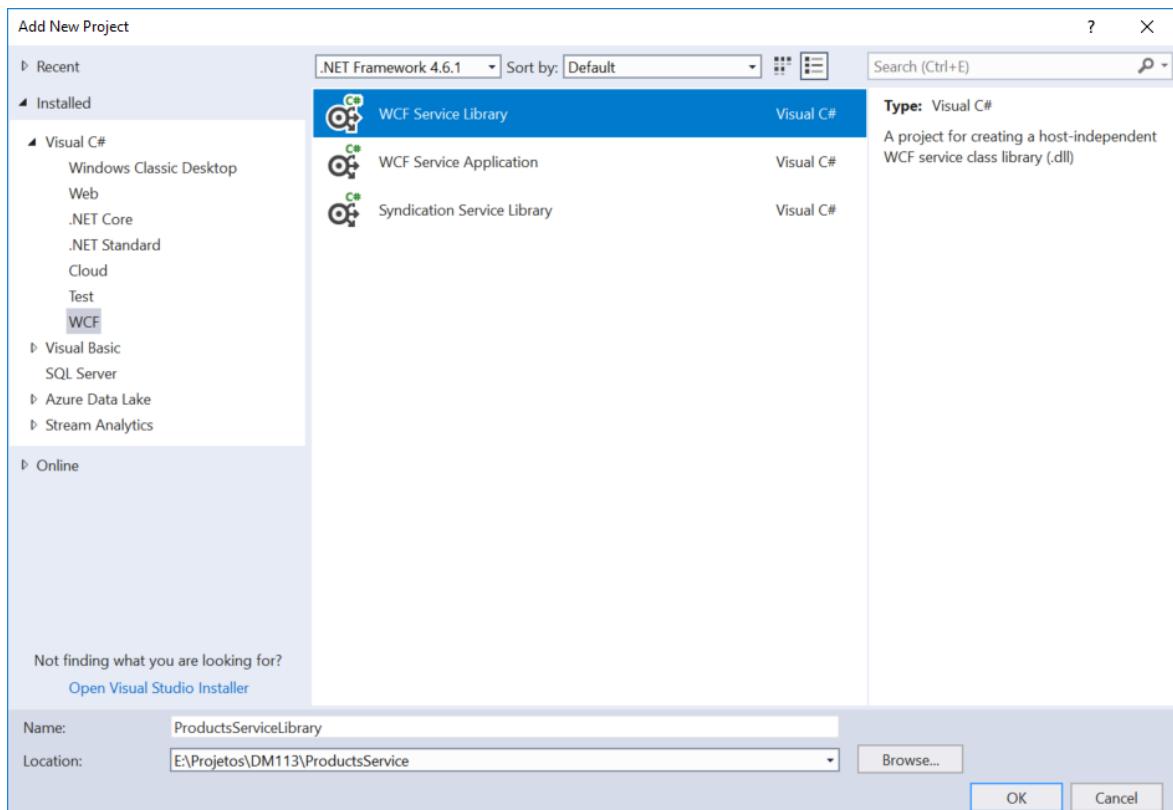


Figura 51 - Criação do Serviço como um *WCF Service Library*

2. Apague os arquivos *IService1.cs* e *Service1.cs*
3. Copie os arquivos *IProductsService.cs* e *ProductsService.cs* criados no projeto *ProductsService* anteriormente
4. Remova a seguinte cláusula de ambos arquivos copiados:

```
using System.ServiceModel.Web;
```

5. Clique com o botão direito sobre o projeto *ProductsServiceLibrary* e clique em *Add Reference*
6. Clique em *Solution\Projects* e marque a *checkbox ProductsEntityModel* e clique em *OK*
7. Usando os mesmos passos anteriores, adicione uma referência a *System.Data.Entity* e clique *OK*

Caso o código apresente um erro de referência a *System.Data.Entity*, clique com o botão direito sobre o projeto *ProductsServiceLibrary* e clique *Manage NuGet Packages* e instale o *EntityFramework*.

8. No arquivo *App.config*, troque

```
<service name="ProductsServiceLibrary.Service1">
```

por

```
<service name="Products.ProductsService">
```

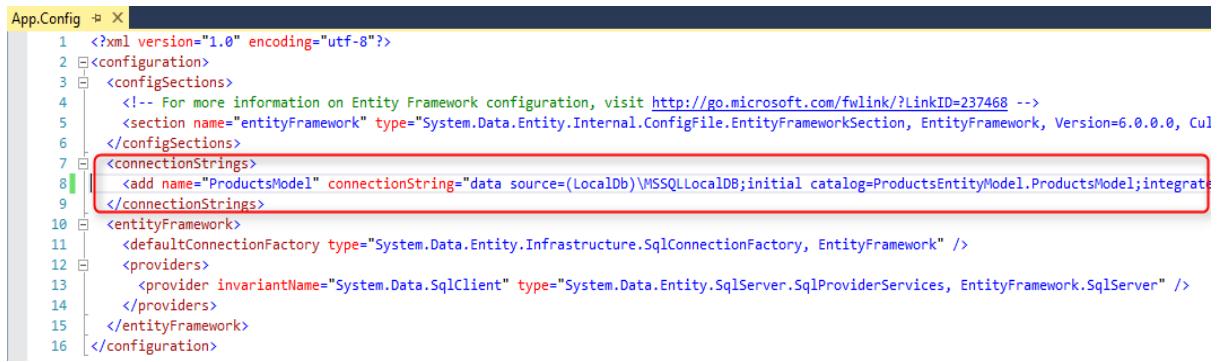
9. No mesmo arquivo, troque

```
<endpoint address="" binding="basicHttpBinding"
contract="ProductsServiceLibrary.IService1">
```

por

```
<endpoint address="" binding="basicHttpBinding"
contract="Products.IProductsService">
```

10. No projeto *ProductsEntityModel*, abra o arquivo *App.Config* e copie a string de conexão do trecho abaixo, para serem coladas em outro arquivo:



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <configuration>
3   <configSections>
4     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
6   </configSections>
7   <connectionStrings>
8     <add name="ProductsModel" connectionString="data source=(LocalDb)\MSSQLLocalDB;initial catalog=ProductsEntityModel.ProductsModel;integrated security=true;multipleactiveresultsets=true;charSet=UTF8" />
9   </connectionStrings>
10  <entityFramework>
11    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />
12    <providers>
13      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
14    </providers>
15  </entityFramework>
16</configuration>
```

Figura 52 – String de conexão no App.config de ProductsEntityModel

11. No projeto *ProductsServiceLibrary*, abra o arquivo *App.config* e, dentro da seção *<configuration>*, na linha anterior a *<system.web>*, cole o trecho copiado e salve o arquivo.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <configuration>
3   <configSections>
4     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
6   </configSections>
7   <appSettings>
8     <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
9   </appSettings>
10  <connectionStrings>
11    <add name="ProductsModel" connectionString="data source=(LocalDb)\MSSQLLocalDB;initial catalog=ProductsEntityModel.ProductsModel;integrated security=true;multipleactiveresultsets=true;charSet=UTF8" />
12  </connectionStrings>
13  <system.web>
14    <compilation debug="true" />
15  </system.web>
```

Figura 53 – String de conexão no App.config de ProductsServiceLibrary

12. Compile o projeto em *Build->Build Solution*

PRÁTICA 06 - Criação do projeto *ProductsServiceHost*

Criação de um Aplicativo para Hospedar o Serviço

Um serviço WCF pode ser hospedado por alguma aplicação (*self-hosting*), tais como *Console Application*, *Windows Form*, *Windows Service* e outras. Desta forma, o código do serviço ficará dentro do código do aplicativo. Para fazer o serviço disponível, basta criar uma instância para a classe *ServiceHost* e definir um *endpoint* para o serviço. Um aplicação console para a hospedagem do serviço será criada.

1. Adicione um novo projeto à solução, nomeando-o *ProductsServiceHost* e usando o modelo *Console Application*, como na Figura 54.

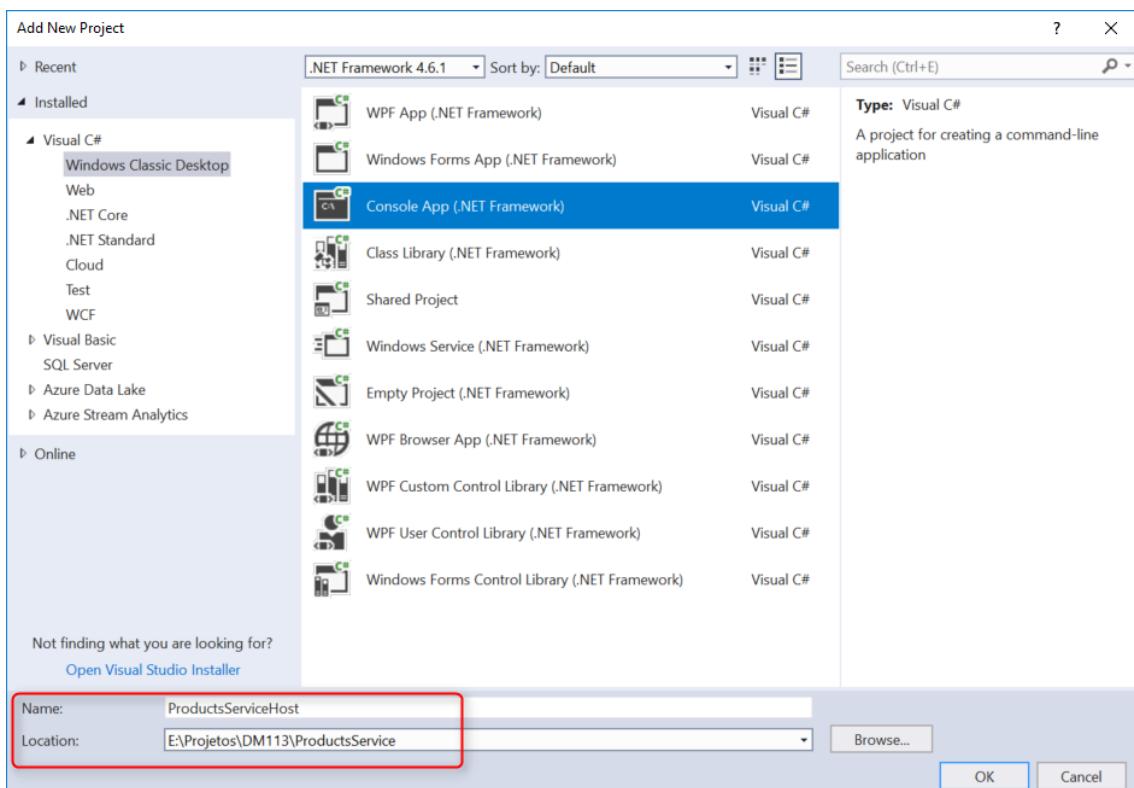


Figura 54 - Novo *Console Application* para Hospedar o Serviço

2. No projeto *ProductsServiceHost*, adicione uma referência a *System.ServiceModel*
3. No mesmo projeto, adicione uma referência ao projeto *ProductsServiceLibrary*
4. Acrescente as seguintes linhas à lista de cláusulas *using* em *Program.cs*:

```
using System.ServiceModel;
using Products;
```

5. No mesmo arquivo, adicione ao método *Main*, o seguinte código:

```

    static void Main(string[] args)
    {
        ServiceHost productsServiceHost = new
        ServiceHost(typeof(ProductsService));
        productsServiceHost.Open();
        Console.WriteLine("Service Running");

        Console.ReadLine();
        Console.WriteLine("Service Stopping");
        productsServiceHost.Close();
    }

```

6. Compile o projeto em *Build->Build Solution*

Configuração da Aplicação de Hospedagem

Agora será necessário criar os *endpoints* do serviço hospedado pelo *ProductServiceHost*. O serviço disponibilizará 2 endpoints, um endpoint TCP e outro HTTP. Para isto, siga os passos abaixo.

Adição de um *Endpoint* TCP

1. Abra o arquivo *App.config* e entre as tags *<configuration>* e *</configuration>*, após *</startup>*, acrescente a *string* de conexão do *Entity Framework*, a mesma utilizada no arquivo *App.config* do projeto *ProductsEntityModel*.
2. No projeto da página *web ProductsService*, abra o arquivo *Web.config* e copie a tag *<system.serviceModel>* e seu conteúdo, incluindo seu delimitador *</system.serviceModel>*
3. Cole no arquivo *App.config*, logo após a tag *</connectionStrings>*
4. Salve e feche o arquivo *App.config*
5. Clique com o botão direito sobre o mesmo *App.config* e clique em *Edit WCF Configuration* para abrir a janela mostrada na Figura 55.

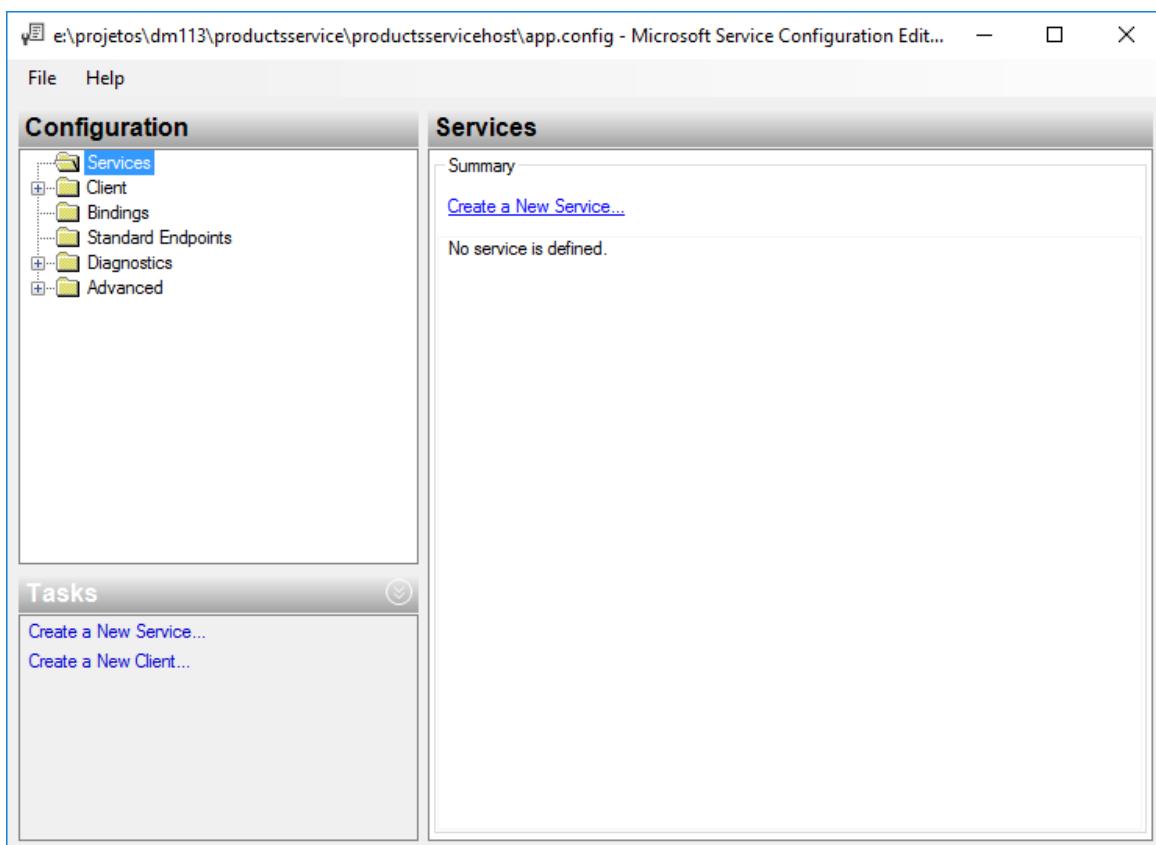


Figura 55 - Editor de Configurações de Serviços

6. Clique em *Create a New Service*
7. Na nova janela, clique em *Browse*
8. Dê um duplo clique em *bin*, depois em *Debug*, *ProductsServiceLibrary.dll* e, finalmente, em *Products.ProductsService*

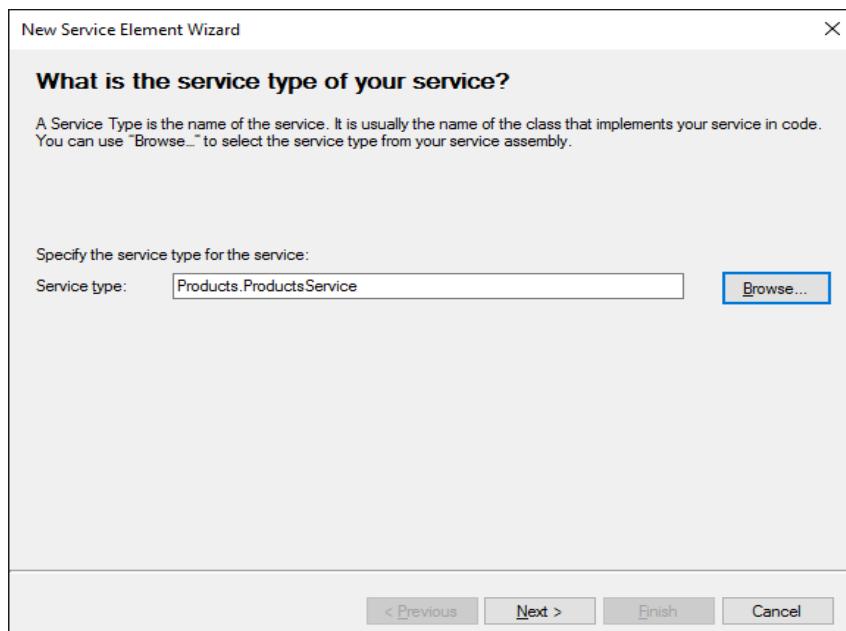


Figura 56 – Adicionando o *ProductsService*

9. Clique em *Next*
10. Verifique se o contrato de serviço escrito é o *Products.IProductsService* e clique em *Next*
11. Na lista de conexões, selecione *TCP* e clique em *Next*
12. No endereço, digite *net.tcp://localhost:8080/TcpService* e clique em *Next*. A Figura 57 mostra a configuração final do serviço.

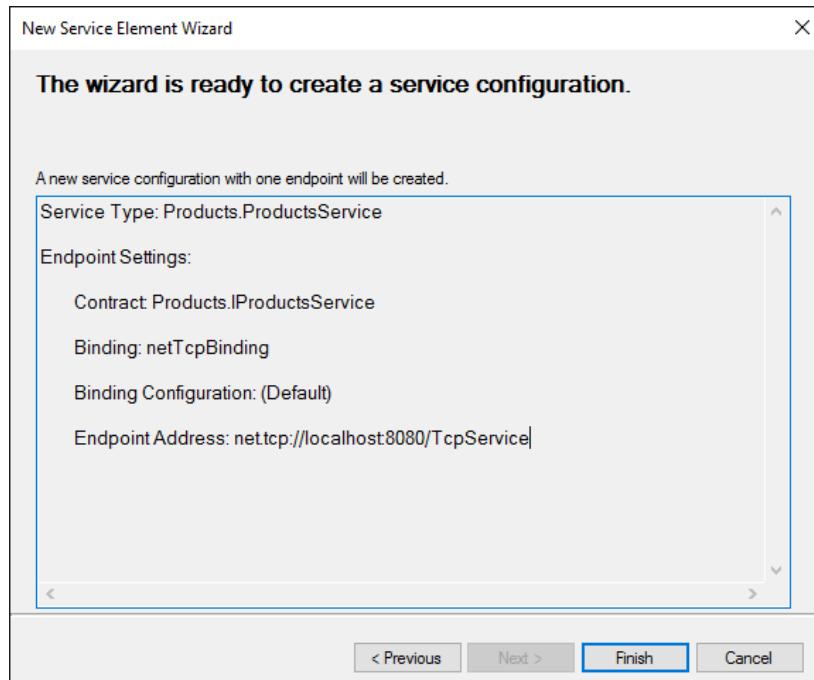


Figura 57 – Configuração do *endpoint* usando TCP

13. Clique em *Finish*
14. No painel *Services*, clique em *(Empty Name)* na frente de *Endpoint*
15. No campo *Name*, digite *NetTcpBinding_IProductsService*
16. Clique no menu *File*, depois em *Save* e feche o editor
17. Abra novamente o arquivo *App.config* e verifique que o editor criou a seção *<services>* com as configurações feitas nos passos anteriores
18. Na seção *<serviceBehaviors>*, altere as configurações de todos os elementos que contenham *http* para *false*
19. Compile o projeto

Teste da Aplicação Hospedeira

Configuração do Cliente para se conectar usando o TCP

1. No projeto *ProductsClient*, abra o arquivo *App.config*
2. Adicione dentro da tag *<cliente>* o endereço do *endpoint* para conexão TCP.

```
<endpoint address="net.tcp://localhost:8080/TcpService" binding="netTcpBinding"
bindingConfiguration="" name="NetTcpBinding_IProductsService"
contract="ProductsService.IProductsService" />
```

3. Abra o arquivo *Program.cs* e acrescente as seguintes linhas ao método *Main*, antes da criação da instância cliente da classe *ProductsServiceClient*:

```
Console.WriteLine("Press ENTER when the service has started");
Console.ReadLine();
```

Agora que há mais de um endpoint disponível, a execução do cliente gerará uma exceção. Abra o arquivo *Program.cs*.

Modifique a linha onde o objeto proxy da classe *ProductsServiceClient* é instanciado para:

```
ProductsServiceClient proxy = new
ProductsServiceClient("NetTcpBinding_IProductsService");
```

4. Compile o projeto.
5. No *Solution Explorer*, clique com o botão direito na solução e clique *Set StartUp Projects*
6. Na nova janela, selecione *Multiple Startup Projects*, selecione a *Action Start* para os projetos *ProductsClient* e *ProductsServiceHost*, deixando a *Action* para os outros projetos em *None*
7. Pressione *CTRL+F5* para iniciar a solução sem *debug*
8. Na janela console do *ProductServiceHost*, verifique a mensagem *Service Running*.



Figura 58 – *ProductsServiceHost* em execução

9. Na janela do console do cliente, pressione *Enter* para iniciá-lo

```
C:\Windows\system32\cmd.exe
Press ENTER when the service has started

Test 1: List all products
Name: Produto 01
Code: 0001
Price: 100,00

Name: Produto 02
Code: 0002
Price: 150,00

Name: Produto 03
Code: 0003
Price: 200,00

Name: Produto 04
Code: 0004
Price: 250,00

Test 2: Display the details of a product
Name: Produto 01
Code: 0001
Price: 100,00

Test 3: Display stock of a product
Current stock: 1200

Test 4: Add stock for a product
Stock changed. Current stock: 1300

Press ENTER to finish
```

Figura 59 – Execução da aplicação cliente

[ATENÇÃO]

Caso a comunicação entre o cliente e o servidor resulte em exceção, verifique se a *connectionStrings* da aplicação hospedeira está configurada corretamente e se o *EntityFramework* está instalado usando o item *Manage NuGet Packages*, mostrado ao se clicar com o botão direito sobre um projeto.

10. Verifique o funcionamento e feche o cliente e o servidor

Adição de um *Endpoint HTTP*

1. No *Solution Explorer*, no projeto *ProductsServiceHost*, clique com o botão direito em *App.config* e clique em *Edit WCF Configuration*
2. No painel dos serviços, clique em *Create a New Service Endpoint*
3. Crie um novo *endpoint* de nome *BasicHttpBinding_IProductsService*, com endereço <http://localhost:8000/ProductsService/Service.svc>, com *binding basicHttpBinding* usando o contrato *Products.IProductsService*

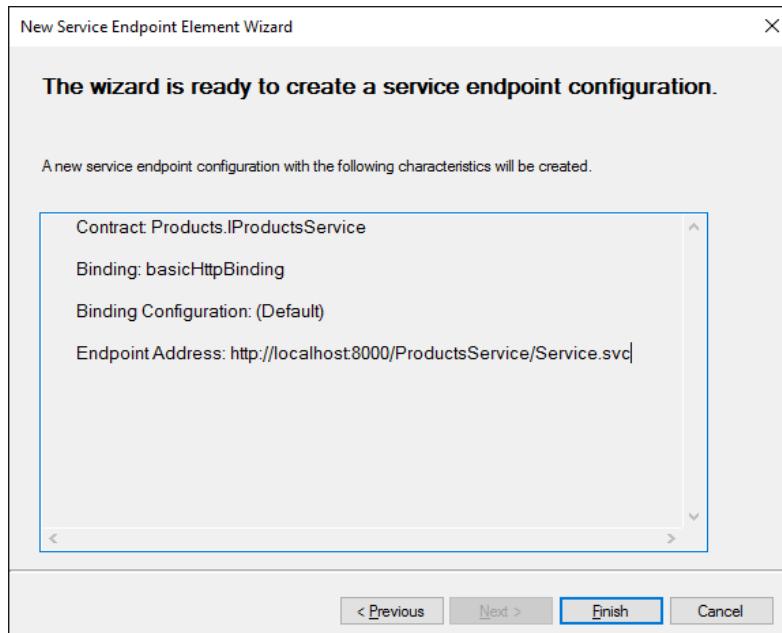


Figura 60 – Configuração do endpoint usando HTTP

4. Salve a nova configuração e feche o editor
5. Abra o arquivo *App.config* no editor
6. Caso as *tags* `<identity>` e `</identity>` tenham sido acrescentadas ao novo *endpoint*, apague-as, junto com seu conteúdo
7. Compile o projeto

Configuração do Cliente para Conexão HTTP

1. No *Solution Explorer*, no projeto *ProductsClient*, clique com o botão direito em *App.config* e clique em *Edit WCF Configuration*
2. Na pasta *Client*, clique no *endpoint BasicHttpBinding_IProductsService* e altere seu endereço para <http://localhost:8000/ProductsService/Service.svc>
3. Salve a configuração e feche o editor
4. No arquivo *Program.cs* de *ProductsClient*, encontre a criação da instância *cliente* da classe *ProductsServiceClient* e altere o parâmetro da chamada do construtor para *BasicHttpBinding_IProductsService*

```
ProductsServiceClient proxy = new
ProductsServiceClient("BasicHttpBinding_IProductsService");
```

5. Compile a solução.
6. Abra o prompt de comando como administrador
7. Para disponibilizar o uso da porta 8000, digite o seguinte comando:

```
netsh http add urlacl url=http://+:8000/ user=(usuário do Windows)
```

8. No *Visual Studio*, pressione *CTRL+F5* para iniciar o servidor e o cliente
9. Na janela do servidor, verifique a mensagem *Service Running*

10. Na janela do cliente, pressione *Enter*
11. Verifique o funcionamento e feche o cliente e o servidor

Endpoints e Bindings

Através dos experimentos anteriores, foi possível concluir que os *endpoints* são os pontos de acesso que um serviço disponibiliza. Além disso, foi possível verificar que a eles devem ser atrelados os *bindings*, que são a política de comunicação com o serviço através de um determinado *endpoint*. Quando o serviço é utilizado, as informações de *binding* são utilizadas para gerar o canal, que é um objeto de classes do namespace *System.ServiceModel.Channels*.

Os dois *bindings* utilizados até agora, o *BasicHttpBinding* e o *NetTcpBinding*, são predefinidos e garantem a interoperabilidade com serviços criados fora do ambiente do WCF. Os *bindings* predefinidos disponibilizados pela biblioteca do WCF e algumas de suas características são mostrados na Tabela I.

Repare que, no primeiro serviço criado, um *binding* não foi especificado. Quando um *endpoint* é criado mas não tem um *binding* associado, o padrão será utilizado para cada *endpoint*. Quando um *endpoint* que utiliza o HTTP como transporte é criado, o *binding* *basicHttpBinding* é utilizado. Para um que utiliza TCP, o *binding* utilizado é o *netTcpBinding*. A lista de associações pode ser encontrada no arquivo *machine.config*, na pasta onde o .NET Framework foi instalado.

Tabela I - Bindings Predefinidos da Biblioteca do WCF

Binding	Características
<i>BasicHttpBinding</i>	Compatível com <i>web services</i> mais antigos. Usa HTTP e HTTPS como transporte e XML como codificação das mensagens.
<i>BasicHttpContextBinding</i>	Similar ao anterior, com suporte a <i>cookies</i> .
<i>WS2007HttpBinding</i>	Suporta sessões seguras. Transporte e codificação iguais aos anteriores, com adição de <i>Message Transmission Optimization Mechanism (MTOM)</i> , para transmissão de dados binários.
<i>WSHttpBinding</i>	Compatível com as versões do WCF e do .NET Framework baseadas no rascunho das especificações para <i>web services</i> .
<i>WSHttpContextBinding</i>	Estende o <i>binding</i> acima para uso dos cabeçalhos <i>SOAP</i> para manutenção de estados.
<i>WSDualHttpBinding</i>	Similar ao <i>WS2007HttpBinding</i> , com suporte a comunicação duplex mas não é capaz de utilizar HTTPS
<i>WebHttpBinding</i>	Suporta <i>web services</i> que utilizam <i>REST</i>
<i>WS2007FederationHttpBinding</i>	Suporta a especificação <i>WS-Federation</i> para comunicação de <i>web services</i> em diferentes domínios de segurança.
<i>WSFederationHttpBinding</i>	Mesmo anterior com suporte a versões mais antigas do WCF e .NET Framework
<i>NetTcpBinding</i>	Utiliza o TCP como transporte e codificação binária. Oferece melhor desempenho que o HTTP e é ideal para comunicação em rede local e entre sistemas <i>Windows</i>
<i>NetTcpContextBinding</i>	Estende o anterior para utilizar cabeçalhos <i>SOAP</i> para enviar informações de

	contexto.
<i>NetPeerTcpBinding</i>	Adiciona suporte a comunicação <i>Peer-to-Peer</i> .
<i>NetNamedPipeBinding</i>	Utiliza <i>named pipes</i> para máximo desempenho entre serviços em um mesmo computador.
<i>NetMsmqBinding</i>	Usa o <i>Microsoft Message Queue (MSMQ)</i> como transporte, permitindo que servidor e cliente se comuniquem mesmo se não estiverem sendo executados ao mesmo tempo.
<i>MsmqIntegrationBinding</i>	Permite que aplicativos WCF se comuniquem com uma fila MSMQ.

Hospedagem do Serviço no Windows Azure

PRÁTICA 07 - Configurando os recursos no Azure

Nesta seção o serviço *ProductsService* será hospedado na plataforma Azure. Desta forma, ele ficará disponível na nuvem para que qualquer aplicação cliente conectada à Internet possa acessá-lo.

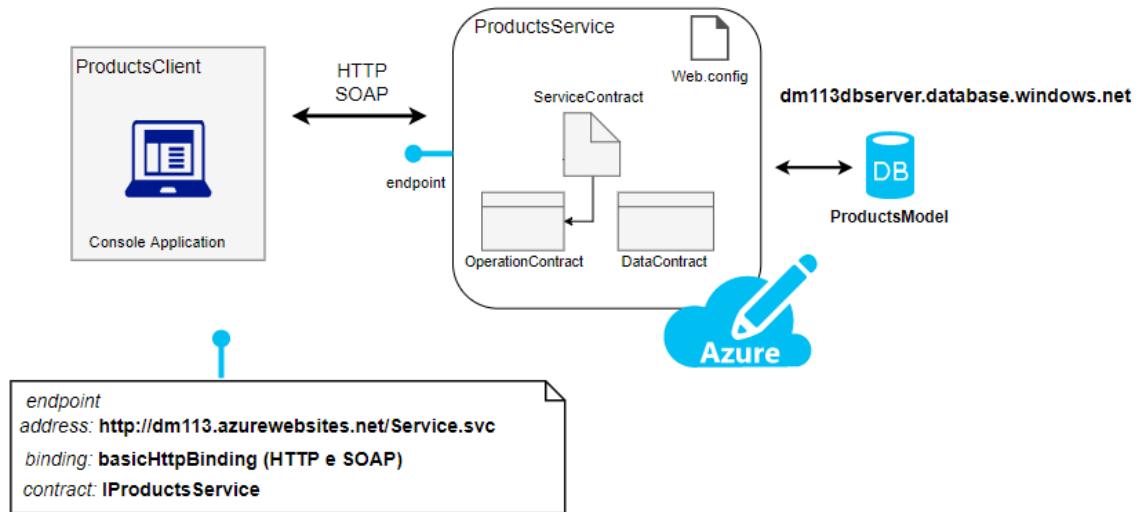


Figura 61 – Hospedagem do Serviço no Windows Azure

O primeiro passo é criar os recursos no Azure para que o serviço possa ser publicado.

1. Faça o login em sua conta no portal do Windows Azure.
<https://portal.azure.com>
2. Certifique-se que o Visual Studio esteja conectado com sua conta do Azure.

3. Na página do portal do Azure, clique em *New*, e em seguida, na opção *Get started*, clique na opção *Web App*.

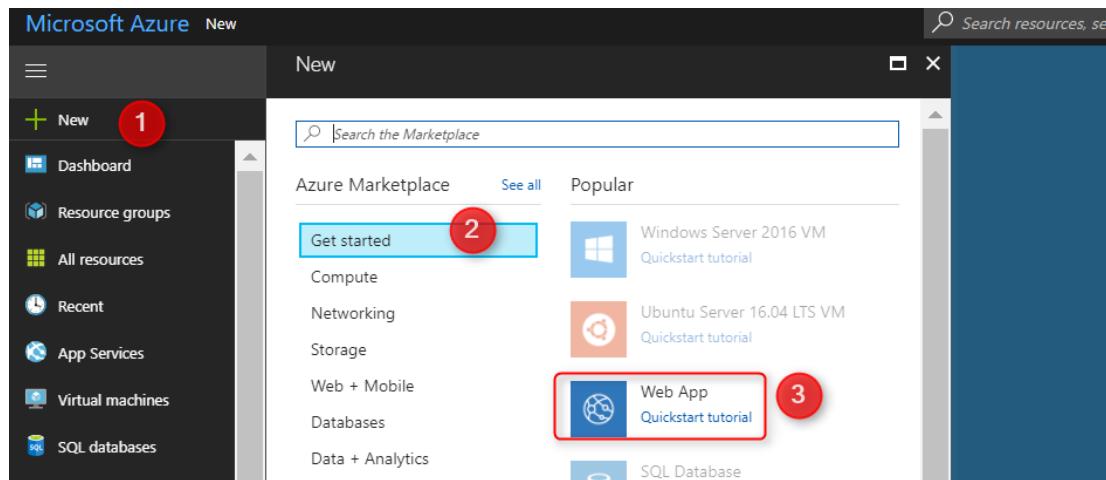


Figura 62 – Opção Web App

[ATENÇÃO]

Outra opção para criar a aplicação web no Azure está localizada no menu *App Services*. Ao entrar nesta opção, clique no botão *Add* e em seguida em *Web Apps*.

4. Em *Web App* será criada uma instância de uma aplicação web para que o serviço *ProductsService* possa ser publicado.
 - Em *App Name*, digite o nome da aplicação;
 - Em *Subscription*, escolha sua conta da Microsoft;
 - Em *Resource Group*, clique na opção *Create New* e dê um nome para o grupo de recursos;
 - Em *App Service plan/Location*, no campo *App Service Plan*, dê o nome para o plano de serviço, em *Location*, escolha a região *Brazil South* e em *Pricing tier*, escolha a opção gratuita, no caso, *F1 Free*.

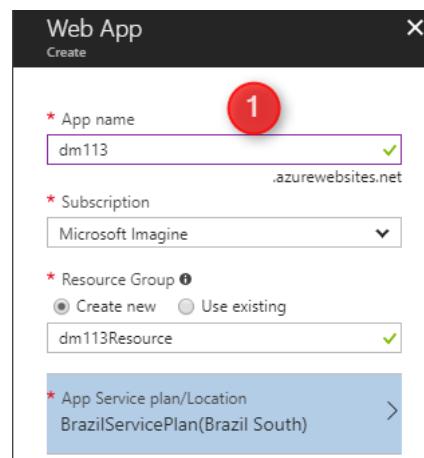


Figura 63 – Configuração da aplicação web e criação do grupo de recursos

Figura 64 – Configuração do Service Plan

5. Após estas configurações, marque a opção *Pin to dashboard* e em seguida clique no botão *Create*.



Figura 65 – Criação da aplicação web

6. Se todos os passos acima foram executados com sucesso, a aplicação web será criada e poderá ser visualizada no Dashboard.

Figura 66 – Visualização da aplicação web no Dashboard

7. A próxima etapa agora é criar o servidor de banco de dados. No menu da página do portal do Azure, clique em *SQL servers*, e em seguida, na opção *Create SQL servers*.

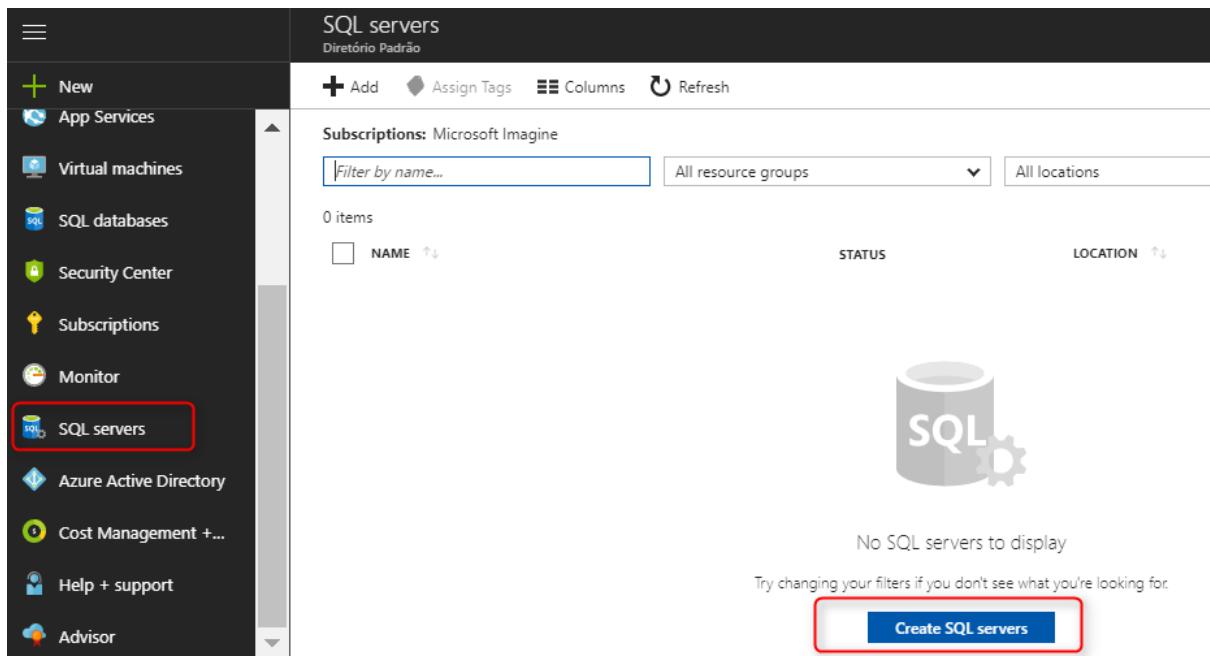


Figura 67 – Criação do servidor SQL

8. Em *SQL Server* será criada uma instância para o servidor de banco de dados para que o banco *ProductsData* possa ser publicado.
 - Em *Server Name*, digite o nome do servidor de banco de dados;
 - Em *Server admin login*, escolha um nome para login para acesso ao servidor de banco de dados;
 - Em *Password* escolha uma senha para acesso ao servidor de banco de dados;
 - Em *Confirm Password*, confirme sua senha;
 - Em *Subscription*, escolha sua conta da Microsoft;
 - Em *Resource Group*, clique na opção *Use existing* e escolha o nome do grupo de recursos criado anteriormente;
 - Em *Location*, escolha a região *Brazil South*;

SQL Server (logical server on... □ X

* Server name
dm113dbserver .database.windows.net

* Server admin login
dm113admin

* Password

* Confirm password

* Subscription
Microsoft Imagine

* Resource group ⓘ
Create new Use existing
dm113Resource

* Location
Brazil South

Allow azure services to access server ⓘ

Pin to dashboard

Create Automation options

Figura 68 – Configuração do servidor SQL

9. Após estas configurações, marque a opção *Pin to dashboard* e em seguida clique no botão *Create*.

Pin to dashboard

Create Automation options

Figura 69 – Criação do servidor SQL

10. Se todos os passos acima foram executados com sucesso, o servidor de banco de dados será criada e poderá também ser visualizado no Dashboard.

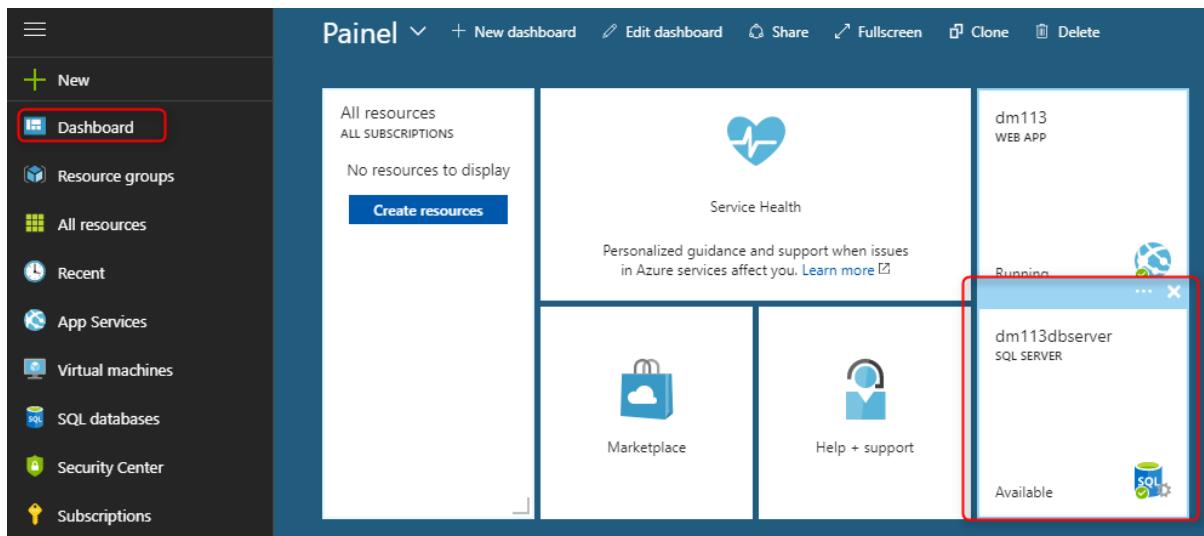


Figura 70 – Visualização do servidor SQL no Dashboard

11. A próxima etapa agora é criar o banco de dados para hospedar o *ProductsModel*. No menu da página do portal do Azure, clique em *SQL databases*, e em seguida, na opção *Create SQL databases*.

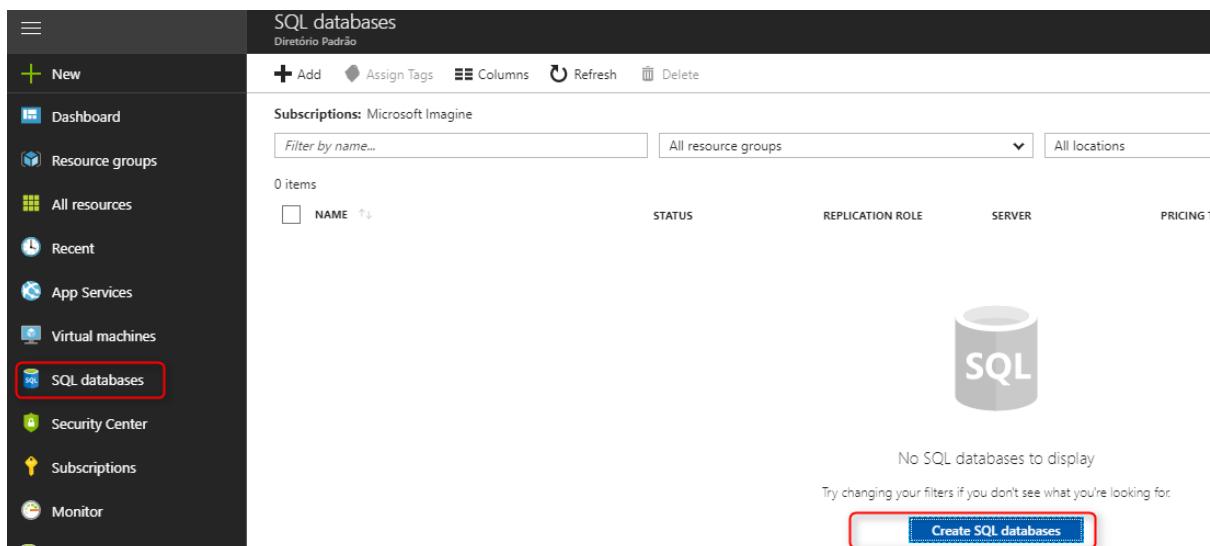


Figura 71 – Criação do banco de dados

12. Em *SQL Database* será criado um banco de dados no servidor de banco de dados criado anteriormente.

- Em *Database Name*, digite o nome do banco de dados;
- Em *Subscription*, escolha sua conta da Microsoft;
- Em *Resource Group*, clique na opção *Use existing* e escolha o nome do grupo de recursos criado anteriormente;
- Em *Select source*, escolha a opção *Blank database*;
- Em *Server*, escolha o servidor de banco de dados criado anteriormente;
- Em *Pricing Tier*, escolha a opção gratuita;

- Em *Collation*, deixe a opção default;

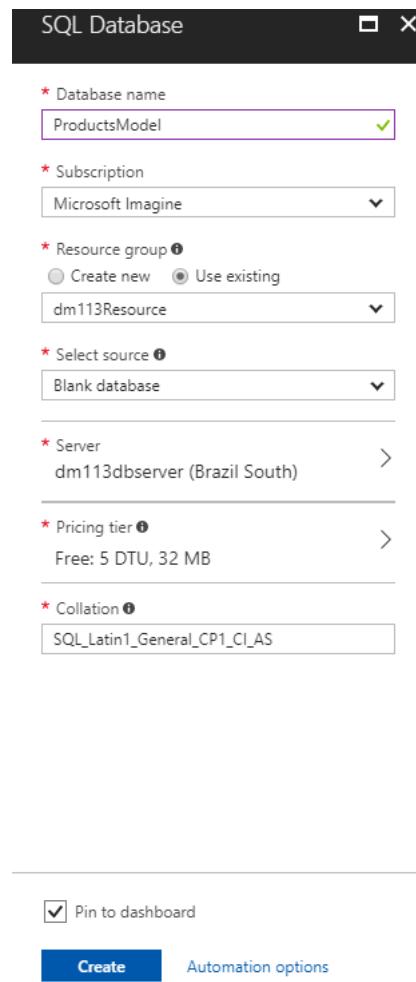


Figura 72 – Configuração do banco de dados

13. Após estas configurações, marque a opção *Pin to dashboard* e em seguida clique no botão *Create*.



Figura 73 – Criação do banco de dados

14. Se todos os passos acima foram executados com sucesso, o banco de dados será criado e poderá também ser visualizado no Dashboard.

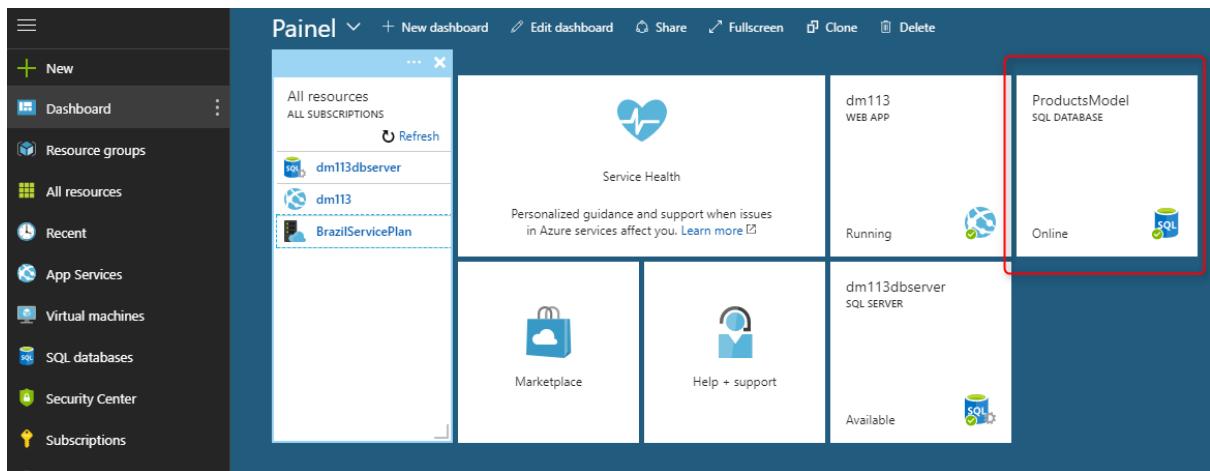


Figura 74 – Visualização do banco de dados no Dashboard

Até aqui foram criados os seguintes recursos:

- App Web chamada de **dm113** com endereço:
<http://dm113.azurewebsites.net>
- SQL Server chamado **dm113dbserver** com o endereço
<dm113dbserver.database.windows.net>
- SQL Databases chamado **ProductsModel**

Após a criação destes recursos, será necessário gerar uma string de conexão para o banco de dados criado anteriormente para ser utilizado no Visual Studio durante a publicação do serviço *ProductService*. Para isto, veja os seguintes procedimentos:

15. No menu da página do portal do Azure, clique na opção *Resource groups*, e em seguida, no recurso criado na etapa anterior.

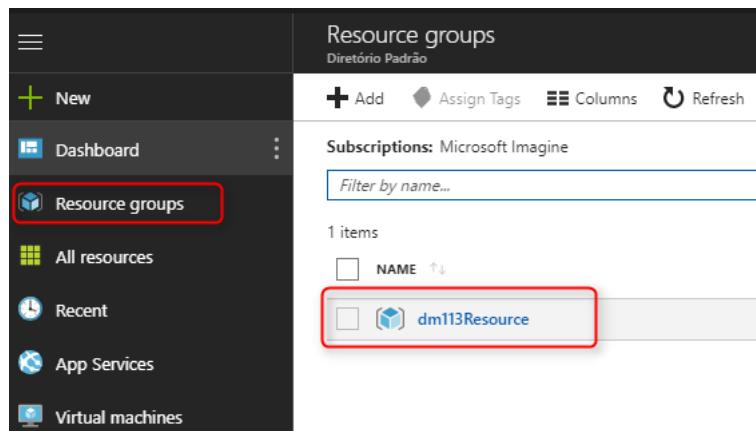


Figura 75 – Seleção do grupo de recursos

16. Em *Resource Group*, clique na aplicação web criada nos passos anteriores, neste caso, a **dm113**.

NAME	TYPE	LOCATION
BrazilServicePlan	App Service plan	Brazil South
dm113	App Service	Brazil South
dm113dbserver	SQL server	Brazil South
ProductsModel	SQL database	Brazil South

Figura 76 – Seleção da aplicação Web

17. Em *App Service*, clique na opção *Data connections*, no menu lateral.

URL	App Service plan/pricing tier	FTP/deployment username
http://dm113.azurewebsites.net	BrazServicePlan (Free 0 Small)	No FTP/deployment user set
		FTP hostname
		FTPS hostname

Figura 77 – Seleção do Data connections da aplicação Web

18. Em *Data connections*, clique em *Add*.

19. Em *Add data connection*, será criada a string de conexão com o banco de dados criado anteriormente.

- Em *Type*, escolha a opção *SQL database*;
- Em *SQL Database*, escolha o banco de dados *ProductsModel* criado anteriormente;
- Em *Resource Group*, clique na opção *Use existing* e escolha o nome do grupo de recursos criado anteriormente;
- Em *Select source*, escolha a opção *Blank database*;
- Em *Server*, escolha o servidor de banco de dados criado anteriormente;

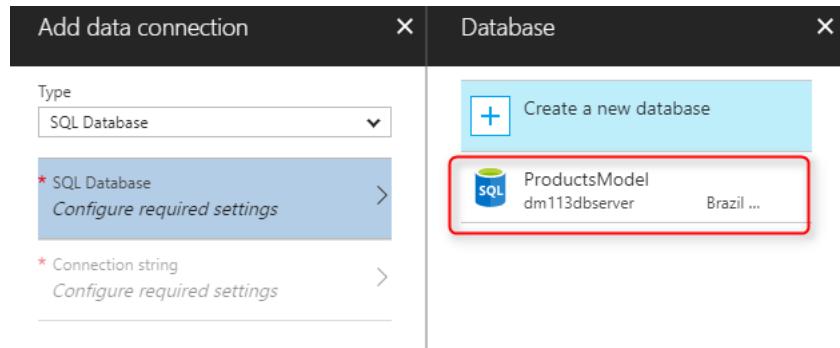


Figura 78 – Seleção do servidor SQL

- Em *Connection string*, em *Name*, crie um nome para a string de conexão;
- Em *SQL Admin Username* e *SQL Admin Password* digite o nome do login e senha do servidor de banco de dados criado; Clique em *OK* em seguida.

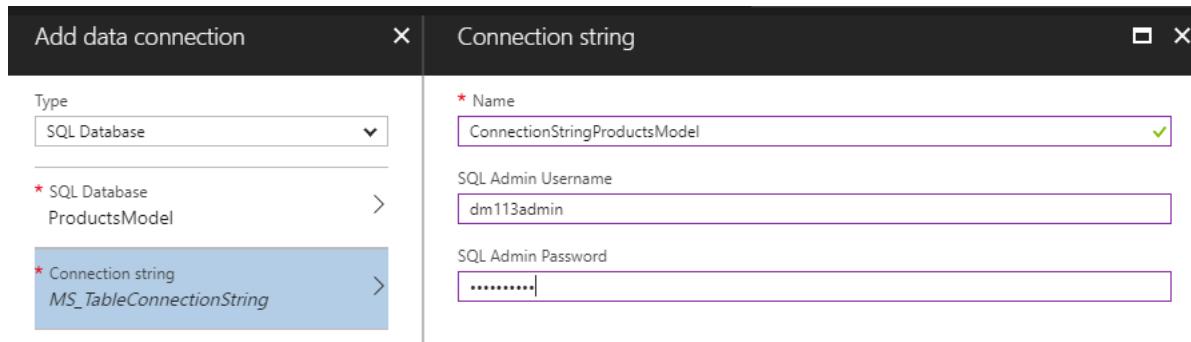


Figura 79 – Criação da connection string

20. Ao clicar em *OK* na tela de *Add data connection* a string de conexão será criada.

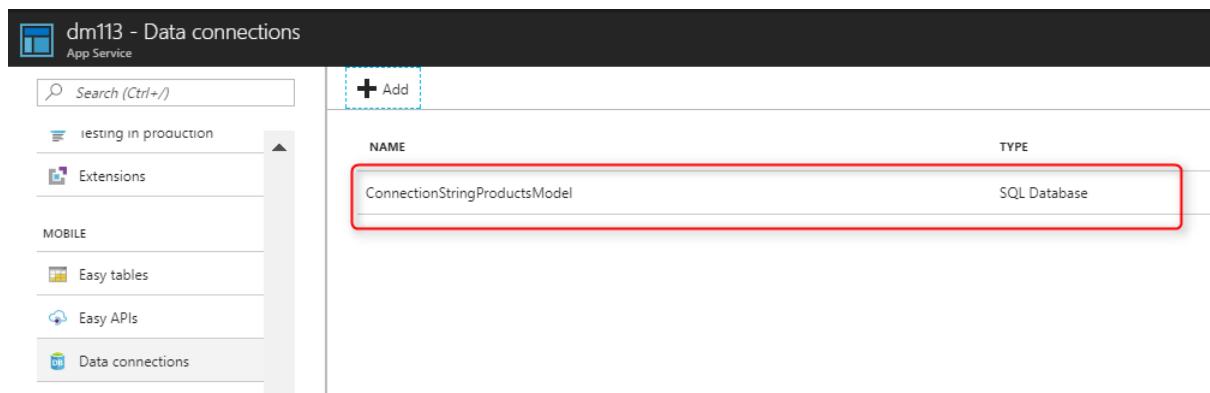
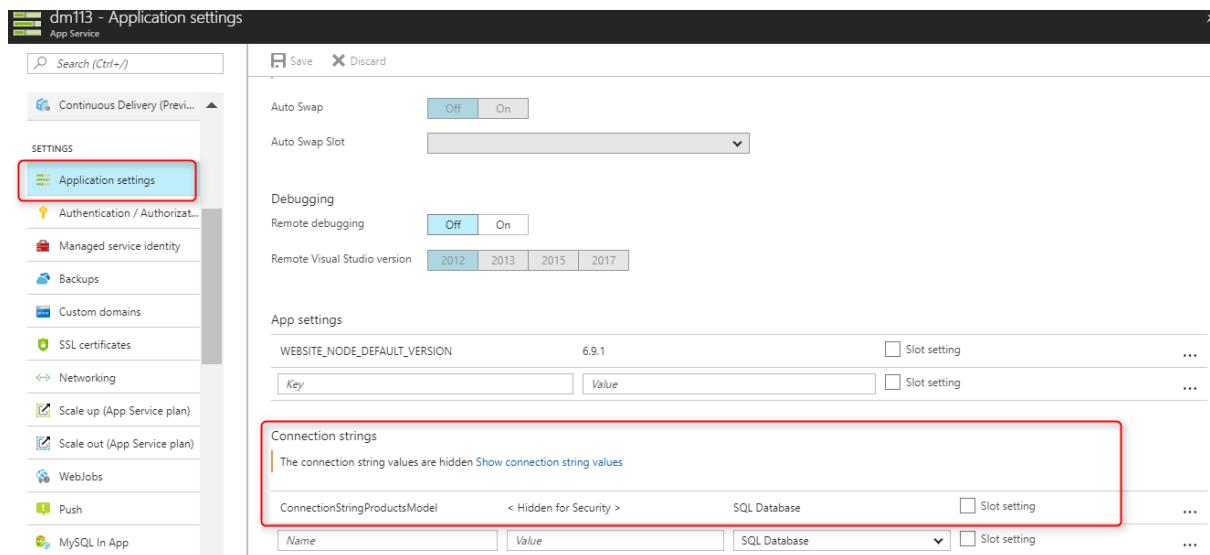


Figura 80 – Visualização da criação da connection string

21. Após este último procedimento, repita agora os passos 15 e 16 e abra o recurso da aplicação web criada.

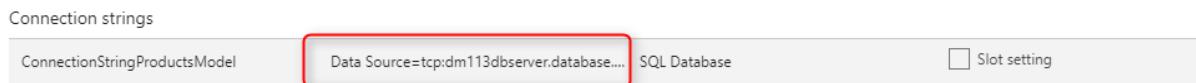
22. Em App Service, clique na opção *Application settings*, no menu lateral, e em seguida procure por *Connection strings* ao lado direito.



The screenshot shows the 'Application settings' page for an Azure App Service named 'dm113'. The left sidebar lists various settings: Continuous Delivery, SETTINGS (with 'Application settings' highlighted), Authentication / Authorization, Managed service identity, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs, Push, and MySQL In App. The main area shows 'Auto Swap' (Off/On), 'Auto Swap Slot' (dropdown), 'Debugging' (Remote debugging Off/On), 'Remote Visual Studio version' (2012, 2013, 2015, 2017), and 'App settings' (WEBSITE_NODE_DEFAULT_VERSION: 6.9.1). The 'Connection strings' section is highlighted with a red box. It contains a note: 'The connection string values are hidden [Show connection string values](#)'. A connection string named 'ConnectionStringProductsModel' is listed, with a dropdown menu showing 'Hidden for Security >' and 'SQL Database'. The 'ConnectionStringProductsModel' row is also highlighted with a red box. The 'ConnectionStringProductsModel' row has columns for 'Name' (ConnectionStringProductsModel), 'Value' (Data Source=tcp:dm113dbserver.database....), 'Type' (SQL Database), and 'Slot setting' (checkbox).

Figura 81 – Visualização do Application Settings

23. Verifique que a string de conexão criada anteriormente aparece na lista de conexões. Ao clicar em *Show connection string values*, observe o valor da string de conexão que foi criado. Esta string será utilizada ao publicar o serviço no Azure pelo Visual Studio.



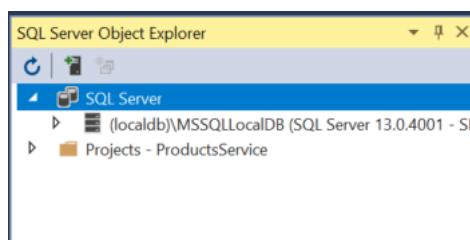
The screenshot shows the 'Connection strings' section of the 'Application settings' page. It lists a single connection string: 'ConnectionStringProductsModel' with the value 'Data Source=tcp:dm113dbserver.database....'. The 'ConnectionStringProductsModel' row is highlighted with a red box. The 'ConnectionStringProductsModel' row has columns for 'Name' (ConnectionStringProductsModel), 'Value' (Data Source=tcp:dm113dbserver.database....), 'Type' (SQL Database), and 'Slot setting' (checkbox).

Figura 82 – Visualização da connection string

Após a criação dos recursos no Azure, agora é a vez de conectar o Visual Studio a estes recursos.

24. No Visual Studio, clique em *View* e abra o *SQL Server Object Explorer*.

25. Clique com o botão direito sobre *SQL Server* e em seguida *Add SQL Server*



The screenshot shows the 'SQL Server Object Explorer' window in Visual Studio. The tree view shows 'SQL Server' expanded, with '(localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - SE)' and 'Projects - ProductsService' as children. The 'SQL Server' node is highlighted with a red box.

Figura 83 – SQL Server Object Explorer

26. Na janela *Connect*, abra a opção Azure e clique no link *Refresh* para trazer as últimas atualizações do Azure.
27. Verifique se o banco de dados *ProductsModel* aparece. Clique sobre ele, entre com as informações de login e senha e em seguida clique em *Connect*.

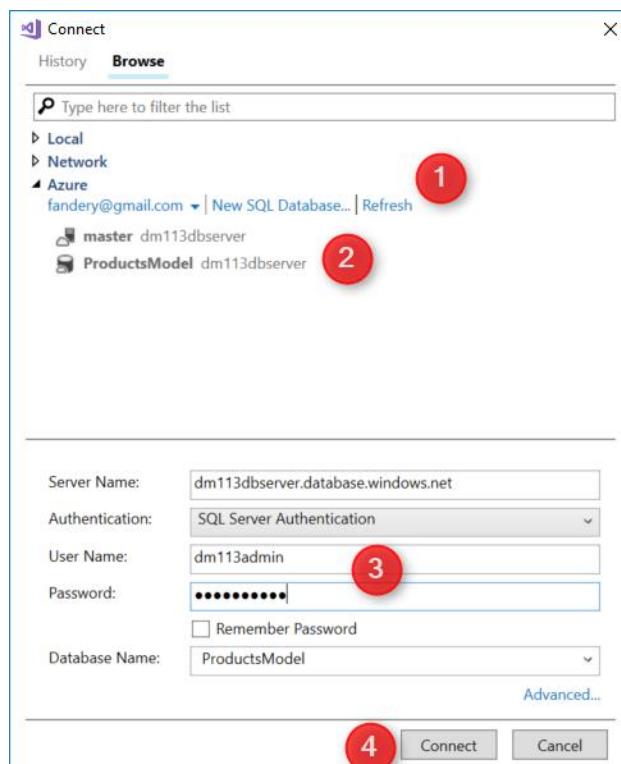


Figura 84 – Conexão ao servidor SQL do Azure pelo Visual Studio

28. Ao clicar em *Connect*, a janela *Create new firewall rule* irá aparecer. Em *Firewall rule*, deixe marcada a opção *Add my client IP* e em seguida clique em *OK*. Isso fará com que seu IP seja adicionado às regras de firewall no Azure para permitir a conexão aos recursos criados.

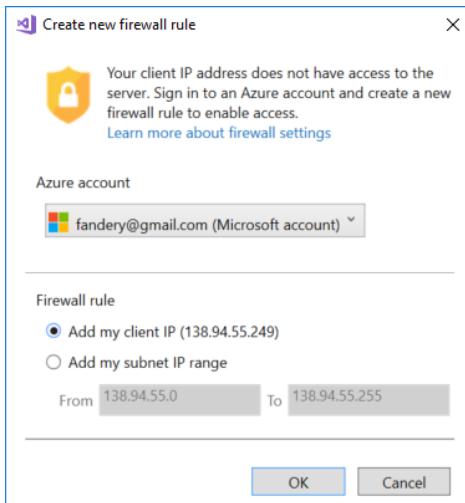


Figura 85 – Criação da regra do firewall

29. Verifique no *SQL Server Object Explorer* que o servidor **dm113dbserver** aparece com a base de dados **ProductsModel**.

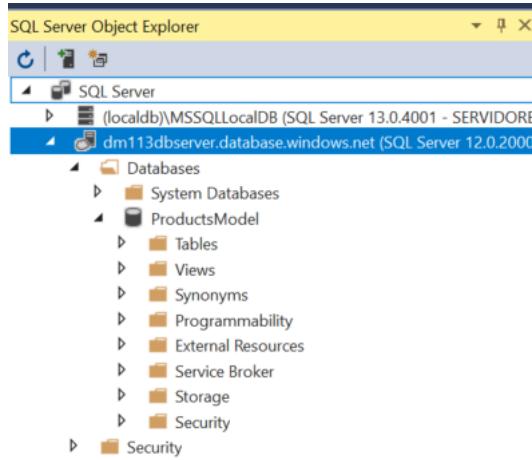


Figura 86 –Visualização da base de dados do Azure

30. No Visual Studio, clique em *View* e abra o *Cloud Explorer*.
31. Em *Cloud Explorer*, entre em sua assinatura do Microsoft Imagine e verifique os recursos que foram criados no Azure.

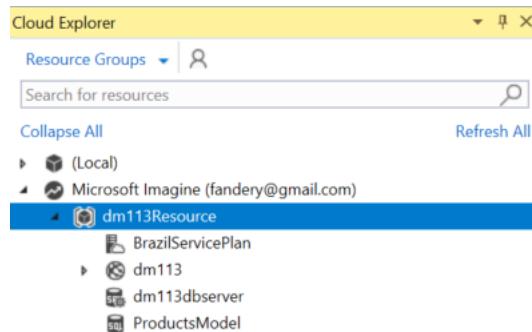


Figura 87 –Visualização do grupo de recursos do Azure

Publicando o serviço *ProductsService* no Azure

32. No Visual Studio, no *Solution Explorer*, clique com o botão direito sobre o projeto *ProductsService* e em seguida em *Publish Web App*.

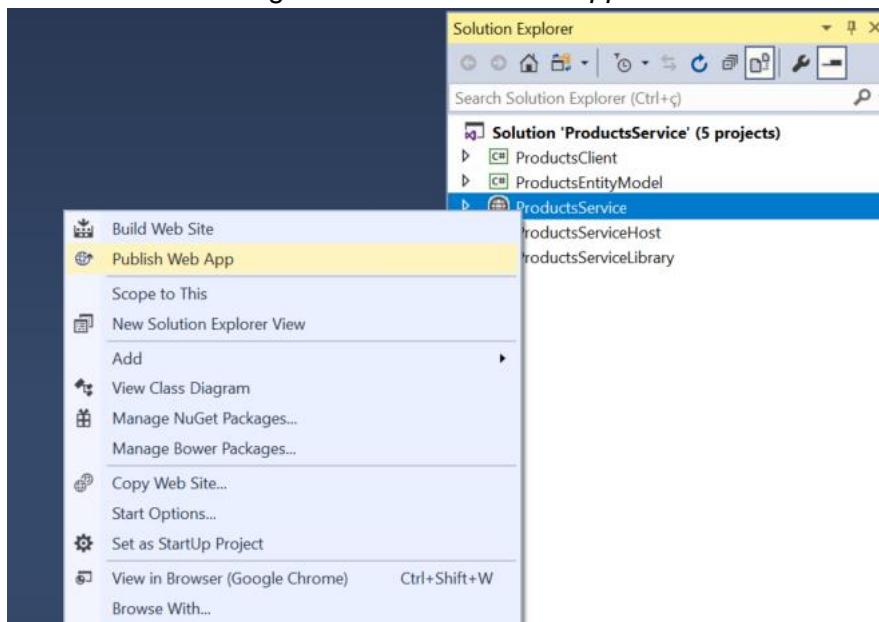


Figura 88 – Publicação do serviço no Azure

33. Na tela *Publish*, clique em *Microsoft Azure App Service*

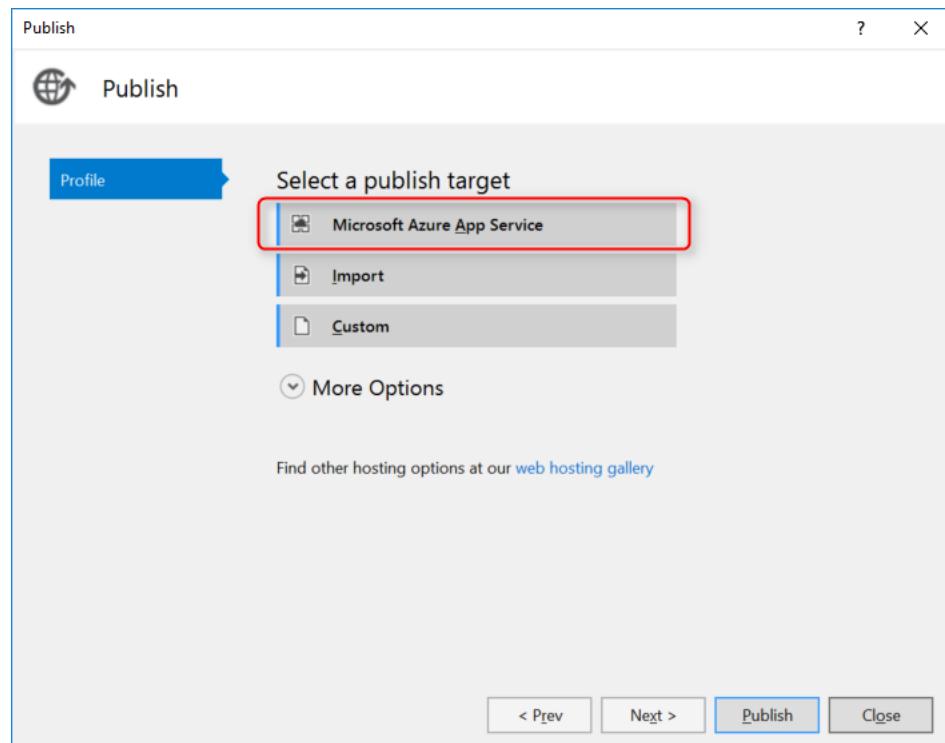


Figura 89 – Criação de um novo Profile

34. Na tela *App Service*, veja se o grupo de recursos criado, neste caso o **dm113Resource**, é encontrado.

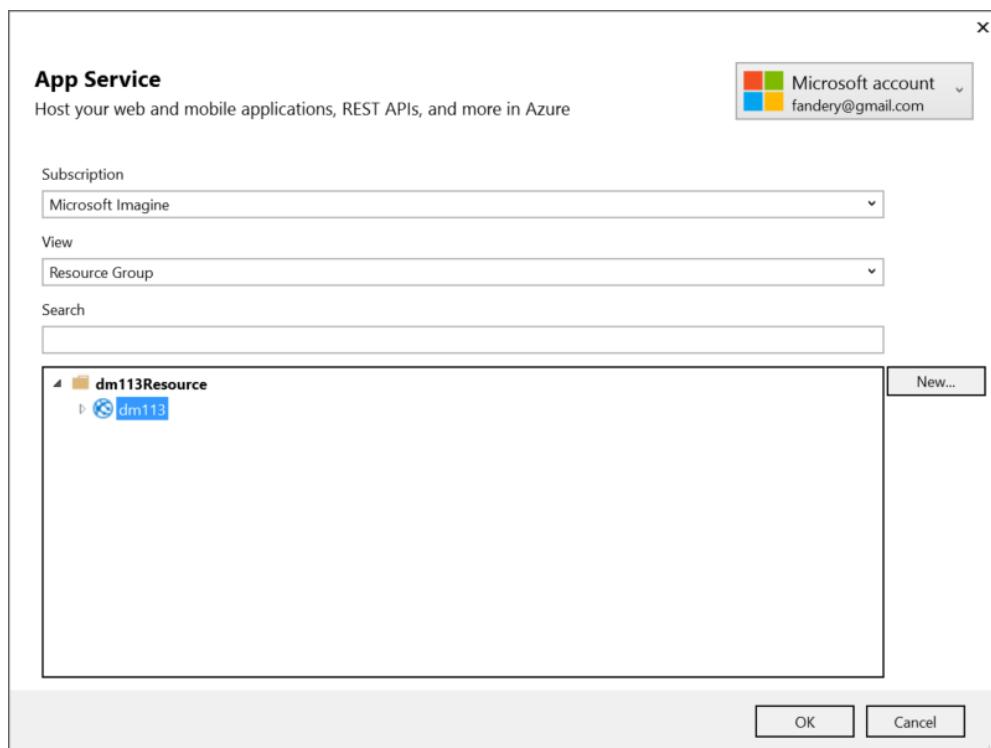


Figura 90 – Seleção do grupo de recursos no Azure

35. Abra o recurso e em seguida clique no nome do aplicativo web, neste caso, o **dm113**. Em seguida clique em **OK**.

36. A janela de *Publish* abrirá novamente e no menu *Connection*, as configurações do aplicativo web criado no Azure serão carregadas. Confira os nomes e clique em *Validate Connection*. Caso esteja tudo OK, clique em *Next*.

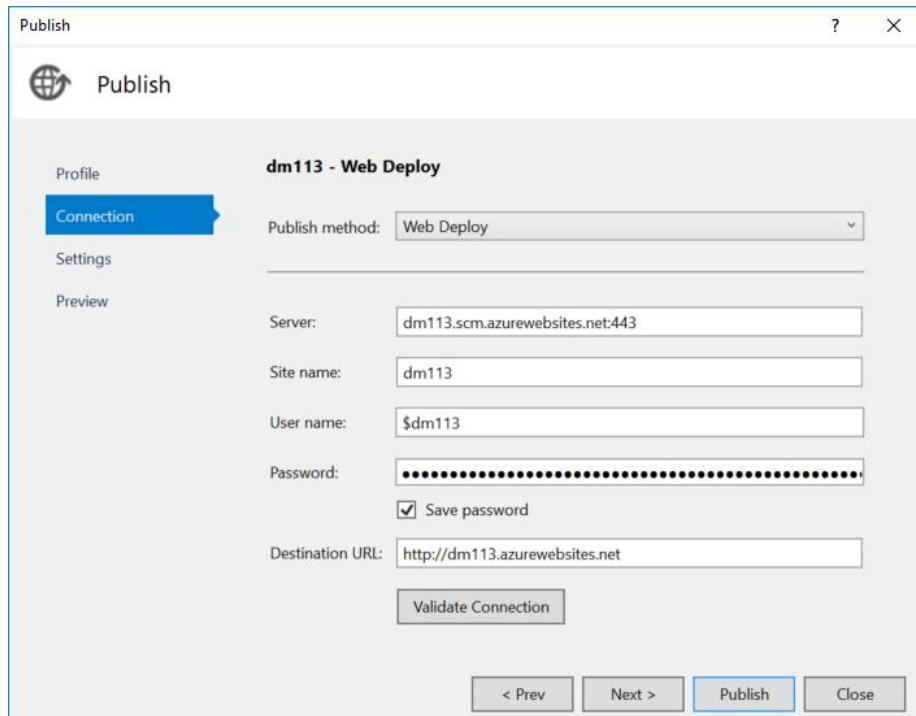


Figura 91 – Dados para conexão à aplicação web no Azure

37. A próxima tela da janela de *Publish*, menu *Settings*, irá trazer as configurações do banco de dados criados no Azure com a string de conexão que foi criada. Analise a string de conexão e marque a opção *Execute Code First Migrations*. Esta última opção utilizará o *Migrations* do *Entity Framework* e fará com que a base de dados local (*ProductsModel*) seja migrada para o banco de dados remoto, também chamado de *ProductsModel*, no Azure. As tabelas *Products* e *Stocks* serão criadas e populadas com os valores iniciais de *seed*.

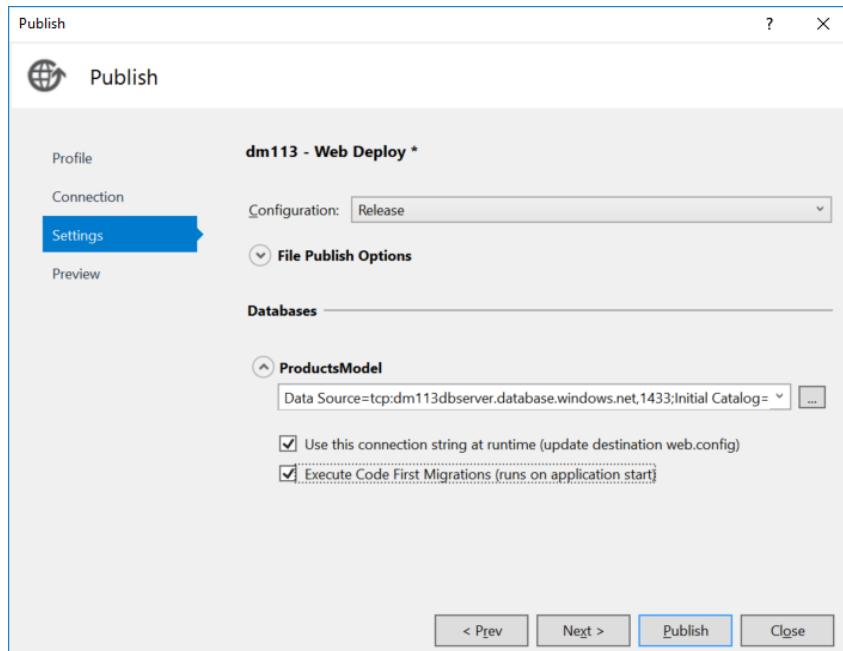


Figura 92 – Dados para conexão ao banco de dados no Azure

[ATENÇÃO]

Ao publicar o serviço novamente no Azure, desabilite a opção *Execute Code First Migrations* caso não tenha alterações a serem feitas na base de dados remota.

38. Clique em *Next* e em seguida em *Publish*.

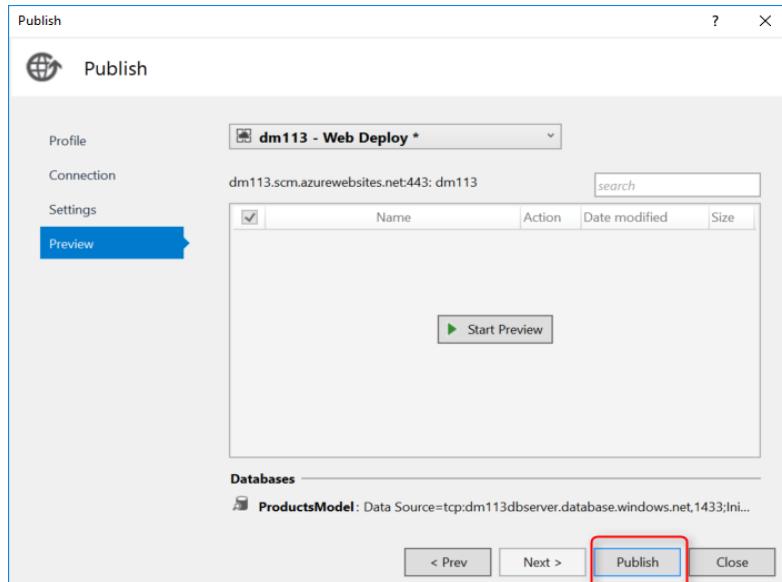


Figura 93 – Publicação no Azure

39. Verifique em *Output* do Visual Studio se a publicação foi realizada com sucesso. Ao finalizar a publicação, a página do serviço será aberta automaticamente no browser.

```
Output
Show output from: Build
3>----- Publish started: Project: ProductsService, Configuration: Debug Any CPU -----
3>Transformou Web.config usando E:\dm113\ProductsService\Web.config em C:\Users\fandery\AppData\Local\Temp\WebSitePublish\ProductsService-1798232759\obj\Debug\TransformWebConfig\transformedWeb.config
3>Inserir Implantação de Banco de Dados adicional de EFCodeFirst Transformou C:\Users\fandery\AppData\Local\Temp\WebSitePublish\ProductsService-1798232759\obj\Debug\TransformWebConfig\transformedWeb.config em C:\Users\fandery\AppData\Local\Temp\WebSitePublish\ProductsService-1798232759\obj\Debug\InsertEFCodeFirstDeploy\transformedWeb.config em C:\Users\fandery\AppData\Local\Temp\WebSitePublish\ProductsService-1798232759\obj\Debug\InsertEFCodeFirstDeploy\transformedWeb.config
3>Copiando todos os arquivos para o local temporário abaixo para empacotar\publicar\ProductsService-1798232759\obj\Debug\TransformWebConfig\transformedWeb.config
3>Iniciar a execução da implantação de web no Aplicativo pacote em https://dm113.scm.azurewebsites.net/msdeploy.axd?site=dm113 ...
3>Adicionando o diretório (dm113\app_code).
3>Adicionando o diretório (dm113\bin).
3>Adicionando ACLs para o caminho (dm113).
3>Adicionando o arquivo (dm113\app_code\ProductsService.cs).
3>Adicionando o arquivo (dm113\app_code\ProductsService.cs).
3>Adicionando o arquivo (dm113\bin\EntityFramework.dll).
3>Adicionando o arquivo (dm113\bin\EntityFramework.SqlServer.dll).
3>Adicionando o arquivo (dm113\bin\ProductsEntityModel.dll).
3>Adicionando o arquivo (dm113\bin\ProductsEntityModel.dll.config).
3>Adicionando o arquivo (dm113\bin\ProductsEntityModel.pdb).
3>Adicionando o arquivo (dm113\packages.config).
3>Adicionando o arquivo (dm113\Service.svc).
3>Adicionando o arquivo (dm113\Web.config).
3>Adicionando ACLs para o caminho (dm113).
3>Adicionando ACLs para o caminho (dm113)
3>Publicação Com Êxito.
3>Aplicativo Web publicado com êxito http://dm113.azurewebsites.net/
=====
===== Build: 2 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Publish: 1 succeeded, 0 failed, 0 skipped =====
```

Figura 94 – Log da publicação do serviço

40. Verifique se o serviço foi publicado corretamente acessando o endereço pelo browser.

<http://dm113.azurewebsites.net/Service.svc>

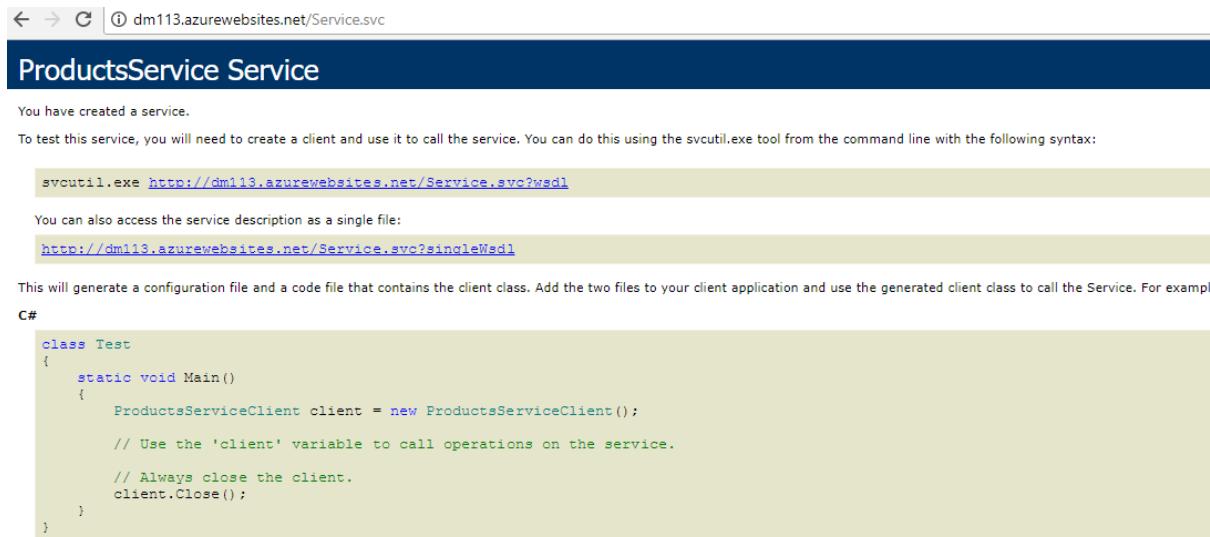


Figura 95 – Serviço publicado

[ATENÇÃO]

As tabelas Products e Stocks serão criadas na base de dados ProductsModel no Azure após o primeiro acesso ao serviço. Ou seja, a migração ocorrerá quando a primeira requisição ao serviço, que consulta a base de dados, ocorrer.

Configuração do Cliente para Conexão HTTP ao serviço no Azure

1. No *Solution Explorer*, no projeto *ProductsClient*, clique com o botão direito em *Connected Services* e em seguida em *Add Service Reference*.

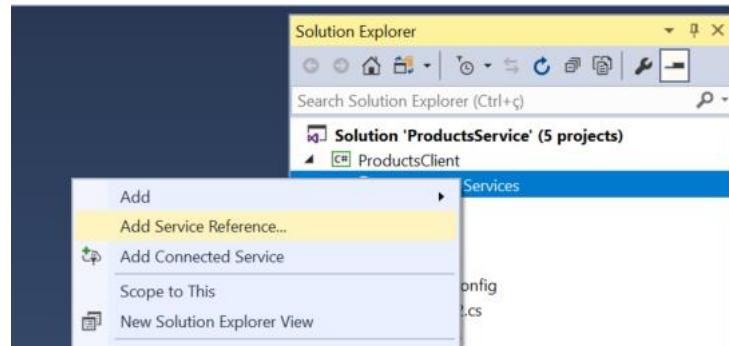


Figura 96 – Adicionar referência para o serviço no Azure

2. No campo *Address*, entre com o endereço do serviço publicado no Azure e clique em *Go*.

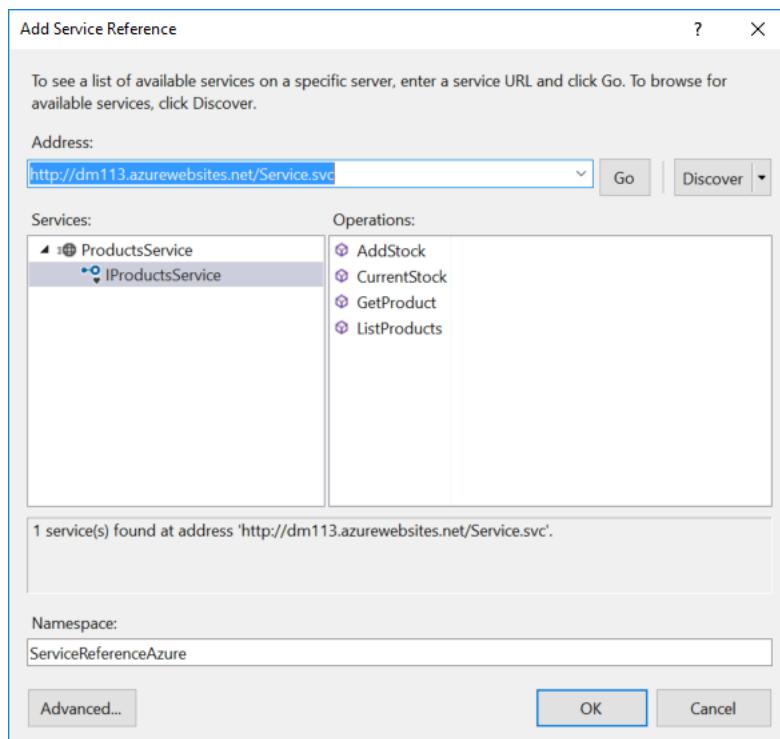


Figura 97 – Descoberta do serviço no Azure

3. Escolha um nome para o *Namespace* e clique em *OK*.
4. No arquivo *App.config* verifique se o endpoint para o serviço foi adicionado.

```

<endpoint address="http://dm113.azurewebsites.net/Service.svc"
          binding="basicHttpBinding"
          bindingConfiguration="BasicHttpBinding_IProductsService"
          contract="ServiceReferenceAzure.IProductsService"
          name="BasicHttpBinding_IProductsService" />

```

5. No arquivo *Program.cs*, remova a seguinte cláusula using:

```
using ProductsClient.ProductsService;
```

6. Acrescente a seguinte cláusula using:

```
using ProductsClient.ServiceReferenceAzure;
```

7. No arquivo *Program.cs* de *ProductsClient*, encontre a criação da instância *cliente* da classe *ProductsServiceClient* e altere o parâmetro da chamada do construtor para *BasicHttpBinding_IProductsService*

```
ProductsServiceClient proxy = new
ProductsServiceClient("BasicHttpBinding_IProductsService");
```

8. Compile a solução.

```
Test 1: List all products
Name: Produto 01
Code: 0001
Price: 100,00

Name: Produto 02
Code: 0002
Price: 150,00

Name: Produto 03
Code: 0003
Price: 200,00

Name: Produto 04
Code: 0004
Price: 250,00

Test 2: Display the details of a product
Name: Produto 01
Code: 0001
Price: 100,00

Test 3: Display stock of a product
Current stock: 1000

Test 4: Add stock for a product
Stock changed. Current stock: 1100

Press ENTER to finish
```

Figura 98 – Execução da aplicação cliente

9. Verifique na base de dados *ProductsModel* do Azure que as tabelas *Products* e *Stocks* foram criadas e populadas com os dados iniciais.

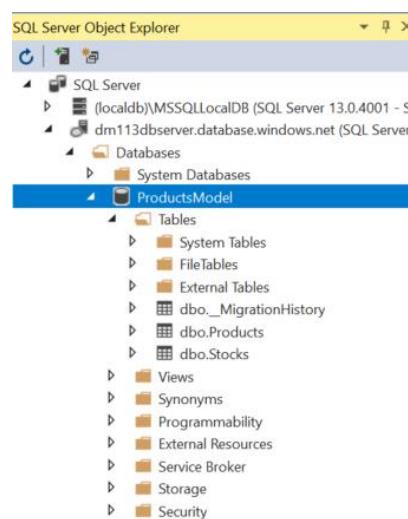


Figura 99 – Visualização do banco de dados do Azure pelo SQL Server Explorer

4. Referências Bibliográficas

[AZU] Azure: Plataforma em Nuvem da Microsoft. Microsoft.

<http://azure.microsoft.com/pt-br/>

[EFRA] *Entity Framework*.

<http://msdn.microsoft.com/en-us/data/aa937723>.

[HTIIS] *How to: Host a WCF Service in IIS*.

[https://msdn.microsoft.com/en-us/library/ms733766\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms733766(v=vs.110).aspx)

[HTWAS] *How to: Host a WCF Service in WAS*.

[https://msdn.microsoft.com/en-us/library/ms733109\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms733109(v=vs.110).aspx)

[IIS] *Internet Information Services*.

<http://www.iis.net/>

[MAC] WCF - Windows Communication Foundation – Introdução. <http://www.macoratti.net>

[MLIU] LIU, Mike. *Wcf 4.0 Multi-Tier Services Development with Linq to Entities*. Packt Publishing Ltd, 2010.

[JSHP] SHARP, John. *Windows Communication Foundation 4 Step by Step*. Sebastopol: O'Reilly Media, 2010. 740 p.

[WIWCF] What Is Windows Communication Foundation.

[https://msdn.microsoft.com/en-us/library/ms731082\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx)

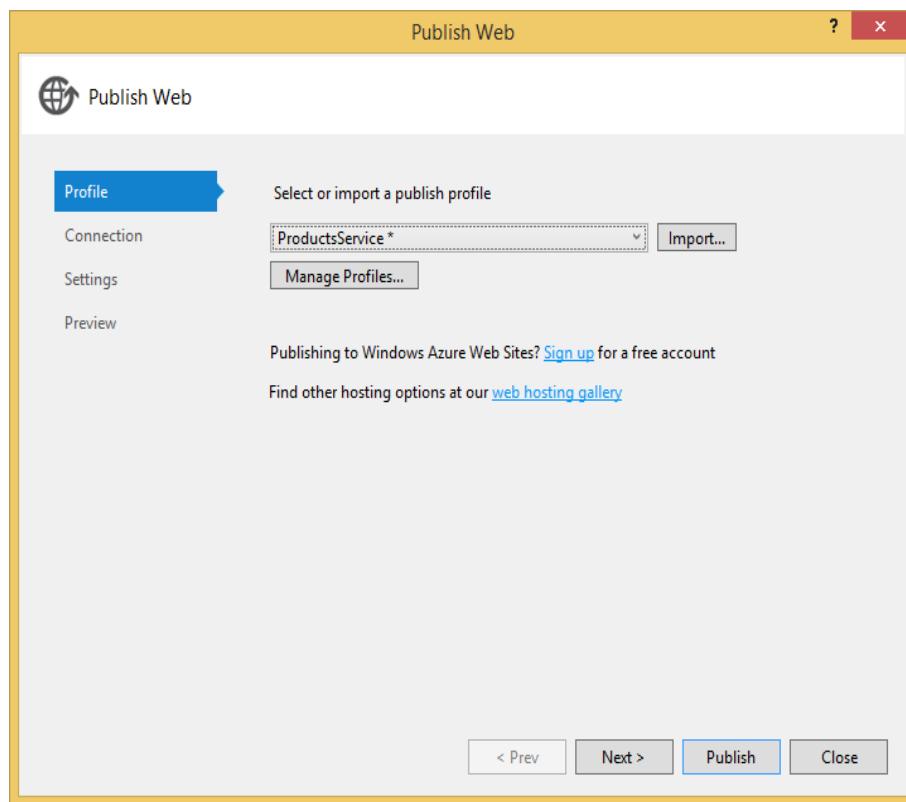
Apêndice A

Hospedagem do Serviço usando o *Internet Information Services*

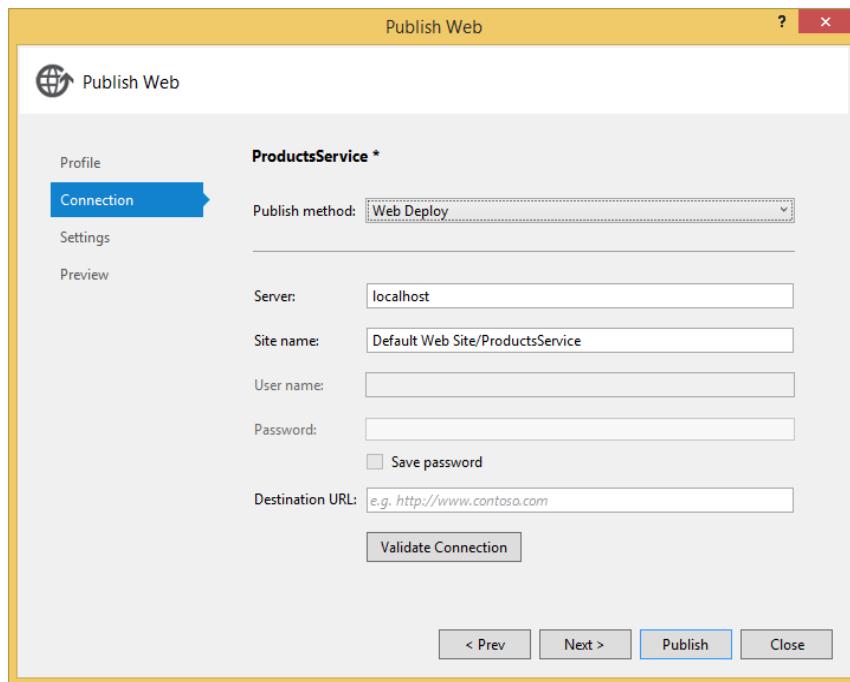
O *Internet Information Services* (IIS) é um servidor de aplicativos Web para Windows, flexível, seguro e gerenciável para qualquer hospedagem na Web. O IIS é uma arquitetura escalável e aberta e está pronto para lidar com tarefas mais exigentes [IIS]. Para hospedar um serviço WCF para o IIS, é necessário a existência de um arquivo físico com a extensão *.svc [HTIIS].

Siga os seguintes passos para a hospedagem do *ProductsService* no servidor IIS:

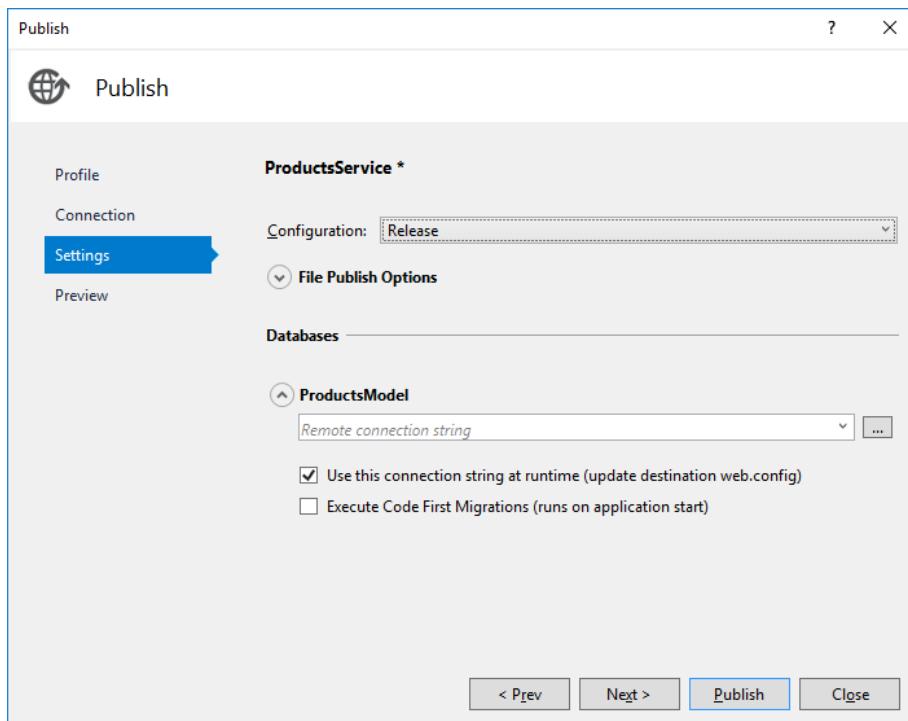
1. Caso o *Visual Studio* não tenha sido aberto com privilégios de administrador, feche-o, clique com o botão direito sobre seu ícone e clique *Executar como administrador* (*Run as administrator*)
2. No *Solution Explorer*, clique com o botão direito sobre o projeto *ProductsService* e clique em *Publish Web App*.
3. Na tela de *Publish*, na opção *Profile*, crie um novo Profile com o nome *ProductsService*.



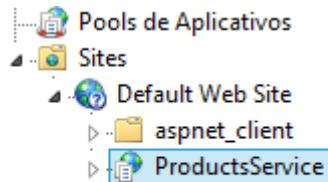
4. Na opção *Connection*, no campo *Publish method*, selecione a opção *Web Deploy*, no campo *Server*, entre com o valor *localhost* e em *Site name*, entre com o valor *Default Web Site/ProductsService* (este será o nome do serviço que será hospedado no IIS).



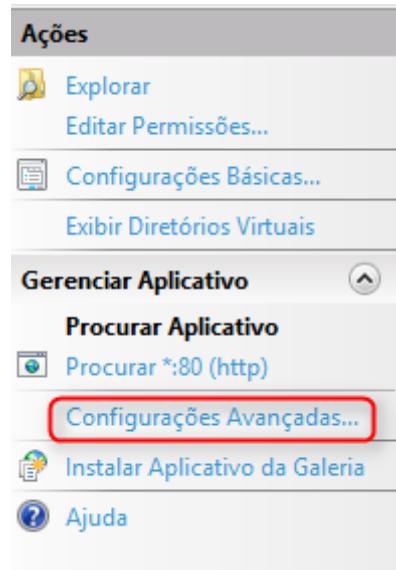
- Clique no botão *Validate Connection* para checar se a conexão com o servidor IIS está correta. Clique em *Next*.
- Na opção *Settings* certifique que a base de dados *ProductsModel* está incluída na seção *Databases*. Clique em *Publish*.



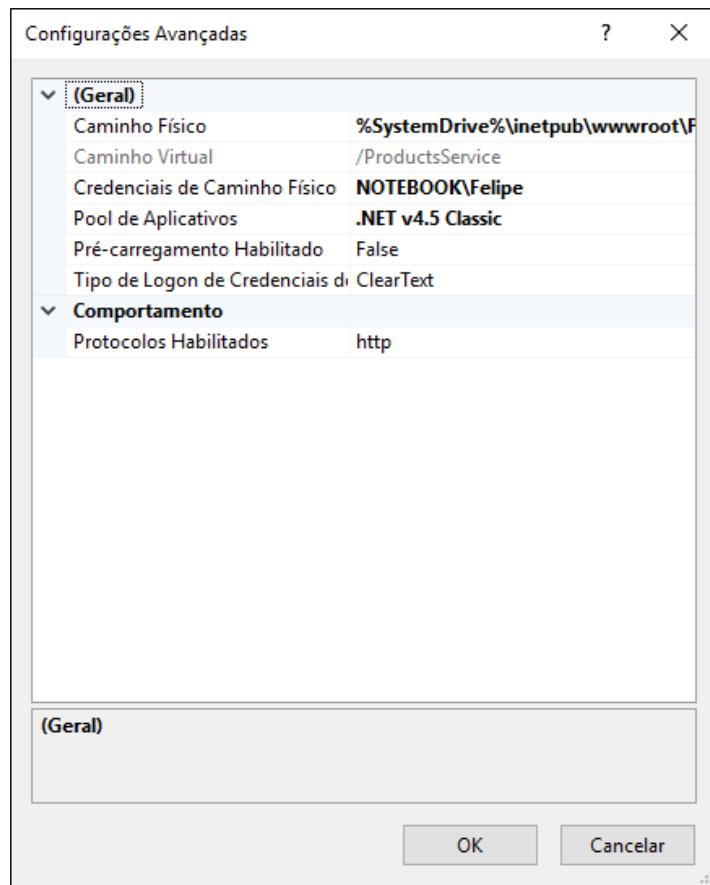
- Abra o *Gerenciador de Serviços de Informações da Internet (IIS)*
- Na lista de conexões à esquerda, expanda *Default Web Site* e observe que o serviço *ProductsService* foi publicado. Clique no serviço.



9. Na lista de ações, à direita, clique em *Configurações Avançadas*

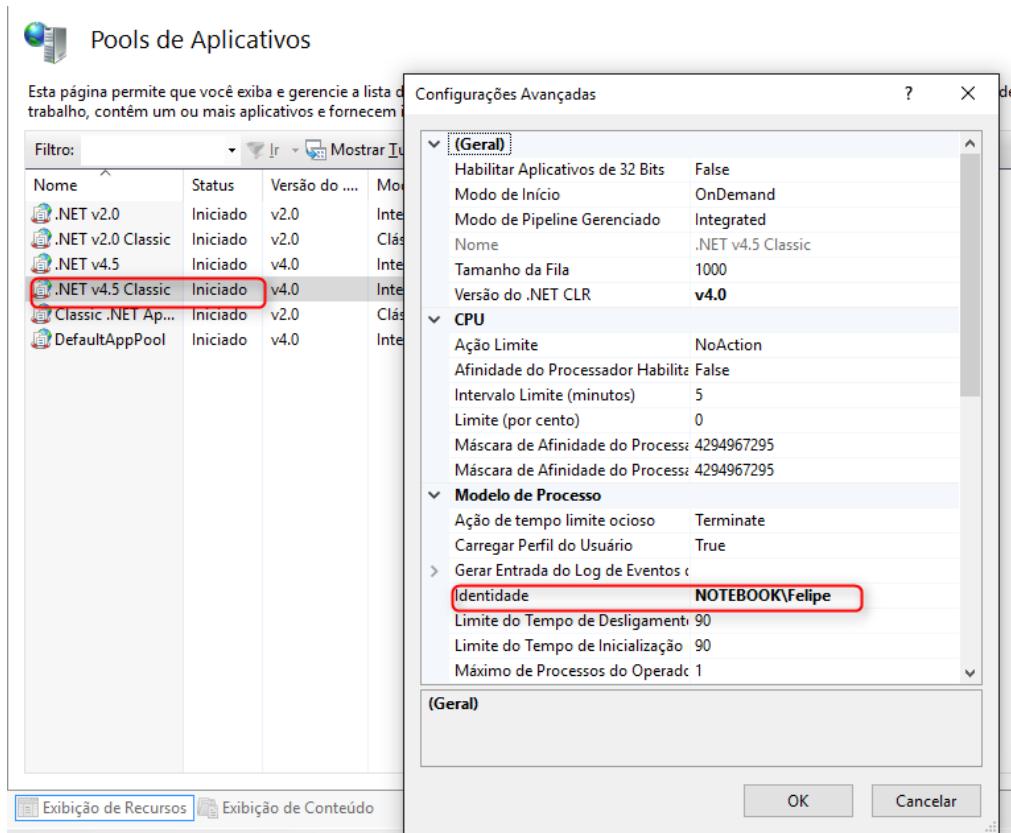


10. Altere o *Pool de Aplicativos* para *.NET v4.5 Classic*
11. No item *Credenciais de Caminho Físico*, clique no botão (...)
12. Clique em *Usuário Específico* e no botão *Definir*
13. Insira seu usuário e senha do *Windows*, que serão utilizados no acesso ao banco de dados sempre que alguém se conectar ao serviço, e clique *OK* e, novamente, *OK*, chegando à janela mostrada a seguir.



Observação:

- Para que a conexão *net.tcp*, utilizada em uma experiência posterior, funcione, as credenciais do *.NET v4.5 Classic* também deverão ser alteradas. Para isso, no painel *Conexões*, à esquerda, clique em *Pools de Aplicativos*. Então, selecione, no painel central, o *.NET v4.5 Classic* e clique, no painel à direita, em *Configurações Avançadas*. Por fim, procure pelo item *Identidade* e altere-o para seu usuário do Windows.



Configuração do Cliente e Teste do Serviço

1. Abra o arquivo *App.config* do projeto *ProductsClient*
2. Na seção `<endpoint>`, altere o endereço para
<http://localhost/ProductsService/Service.svc>
3. Pressione *CTRL+F5* para rodar o cliente e verifique o funcionamento do serviço

Caso o *Apache* ou algum outro servidor que utilize a porta *80* esteja rodando, o *Default Web Site* do *IIS* não será capaz de iniciar. Pare qualquer serviço que utilize a porta *80* e inicie o *Default Web Site*. Caso o problema persista, mude o número da porta clicando com o botão direito em *Default Web Site* e clique em *Editar Associações*.

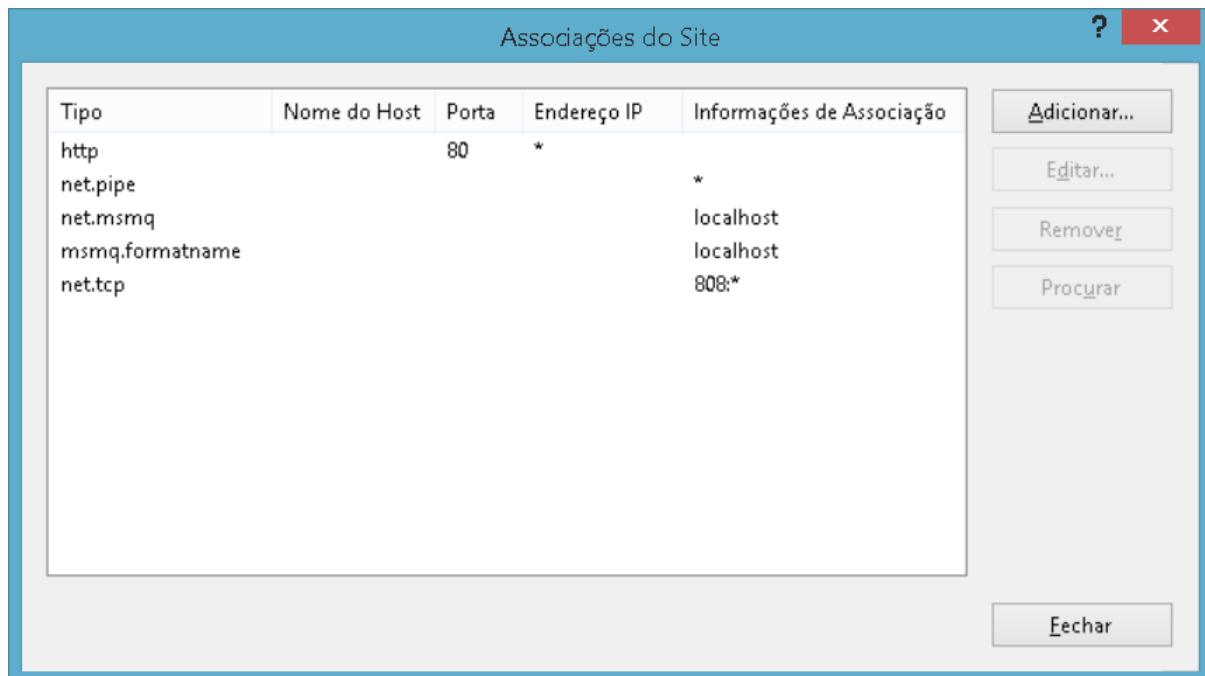
Ainda, caso a configuração do *IIS* para acesso ao banco de dados não funcione, clique sobre o serviço, depois no ícone *Autenticação*, no painel central. Então, edite a *Autenticação Anônima* para que utilize também seu usuário do Windows.

Configuração do Ambiente de Hospedagem para Suporte ao TCP

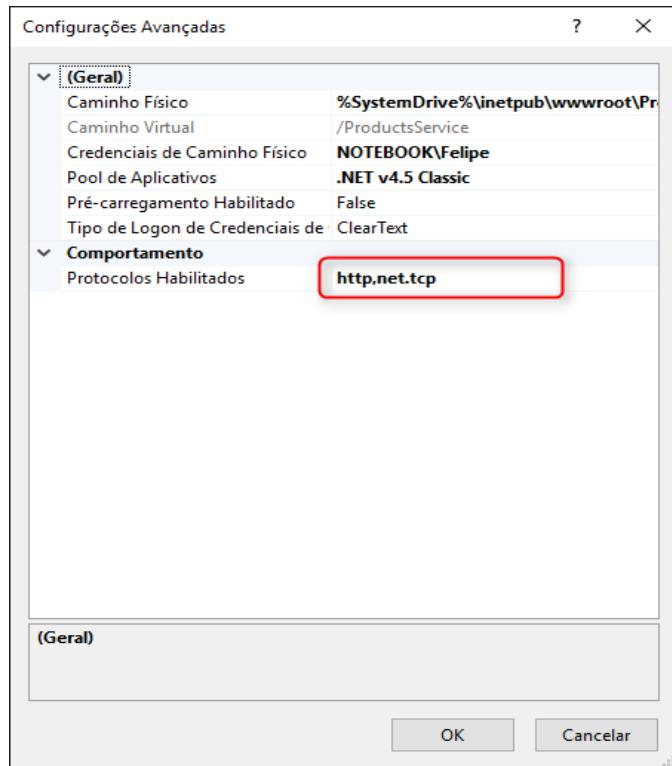
Para que o *IIS* suporte mecanismos de transporte que não sejam *HTTP*, é necessária a instalação do *WAS - Windows Activation Services*, um novo mecanismo de ativação de processo que é instalado dentro do *IIS*, que remove a dependência da

arquitetura de ativação do IIS e permite a comunicação através de outros protocolos como TCP, MSMQ, e *named pipes* [HTWAS].

1. Inicie o *Gerenciador de Serviços de Informações da Internet* como administrador
2. Na lista de conexões, à esquerda, clique com o botão direito em *Default Web Site* e clique em *Editar Associações*
3. Verifique, no diálogo que se abriu, que o servidor espera conexões TCP (linha *net.tcp*) na porta 808.



4. Clique em *Fechar*
5. Na lista de conexões, clique em *ProductsService*
6. Na lista de ações, à direita, clique em *Configurações Avançadas*
7. Em *Protocolos Habilitados*, adicione uma vírgula após *http* e digite *net.tcp* após a vírgula.



8. Clique em **OK**

Configuração do Cliente para se conectar usando o TCP

1. No *Solution Explorer*, abra o arquivo *App.config* do projeto *ProductsClient*
2. Na seção *<client>*, adicione o trecho a seguir:

```
<endpoint address="net.tcp://localhost/ProductsService/Service.svc"
           binding="netTcpBinding" contract="ProductsService.IProductsService"
           name="NetTcpBinding_IProductsService" />
```

Agora que há mais de um *endpoint* disponível, a execução do cliente gerará uma exceção.

3. Abra o arquivo *Program.cs*
4. Modifique a linha onde o objeto *proxy* da classe *ProductsServiceClient* é instanciado para:

```
ProductsServiceClient proxy = new
ProductsServiceClient("NetTcpBinding_IProductsService");
```

5. Compile o projeto e *CTRL+F5* para executar o cliente
6. O cliente se conectará ao serviço usando o TCP e o funcionamento deverá ser idêntico ao anterior

Caso a conexão TCP seja recusada pela máquina de destino, verifique se o serviço do Windows *Listener Adapter Net.Tcp* está habilitado.