Azure

Coloque suas plataformas e serviços no cloud

Casa do Código

THIAGO CUSTÓDIO

Edição

Adriano Almeida
Vivian Matsui

Revisão

Bianca Hubert
Vivian Matsui

[2016]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

AGRADECIMENTOS

Existem muitas pessoas que me ajudaram ao longo destes mais de dez anos na área de tecnologia da informação. Seria inevitável tentar lembrar de todos, e de qual foi a sua colaboração para o meu desenvolvimento como profissional, sem me esquecer de citar alguém. Sendo assim, vou me limitar a agradecer a minha família, até mesmo para não deixar esta seção extensa.

Mãe Cássia Rita e Pai José Wilson, que sempre me deram o exemplo de que o trabalho é o meio pelo qual o homem busca suprir suas necessidades, alcançar seus objetivos e se realizar. Nunca mediram esforços para me pagar treinamentos na área de tecnologia e estudos em boas escolas.

Irmão Victor Custódio, o grande responsável por eu querer trabalhar na área de tecnologia da informação. Sempre fiquei fascinado com a maneira como ele aprendeu Basic, HTML e muitas outras tecnologias e conceitos, de forma autodidata e em uma época na qual não havia muita informação disponível em português (muito menos sites de busca eficientes).

Avó Maria, meu grande obrigado por todo o amor e o carinho.

Noiva Natália, minha melhor amiga e companheira, obrigado pela paciência nos momentos em que estive ausente escrevendo este livro (mesmo estando ao seu lado), ministrando treinamentos, participando de eventos adquirindo e compartilhando conhecimento.

SOBRE O AUTOR

Decidi muito cedo o que eu gostaria de fazer pelo resto de minha vida. Graças a isso, comecei a me preparar antes do que muita gente, o que permitiu que eu fosse contratado ainda adolescente, aos 15 anos de idade.

Tive a felicidade de trabalhar na mesma escola onde fiz meu ensino médio técnico. Nessa instituição, durante o horário comercial, funcionava uma divisão de uma das maiores consultorias em tecnologia da informação daquela época. Por obra do destino, consegui ser contratado por essa consultoria aos 16 anos e, desde então, continuo programando na plataforma .NET, embora eu também conheça outras linguagens de programação.

Meu primeiro contato com o Windows Azure foi no ano de 2011, quando realizei um treinamento sobre o tema com o MVP de C#, Giovanni Bassi. Naquela época, a plataforma estava engatinhando, mas já se mostrava muito promissora. Desde então, dedico uma boa parcela do meu tempo com estudos relacionados a ela.

Compartilho meus aprendizados no meu blog (http://thiagocustodio.azurewebsites.net), palestrando em grandes eventos nacionais, além de responder a perguntas nos principais fóruns nacionais e internacionais.

Posso ser encontrado também pelo Twitter, em http://twitter.com/thdotnet.

PREFÁCIO

Eu, Giovanni Bassi, tive meu primeiro contato com Azure em 2009. Ele era um conjunto de poucos serviços que não faziam muita coisa. Enquanto plataforma de nuvem, deixava muito a desejar quando comparado aos concorrentes, tanto no tipo de serviços oferecidos (era recém-lançado) quanto na qualidade e flexibilidade.

No entanto, duas qualidades não faltavam: visão e ambição. Lembro-me claramente que a visão da Microsoft sobre o novo serviço era muito clara: ser a melhor plataforma de nuvem do mercado. Lembro de que, ao palestrar sobre Azure naquela época e observar os poucos serviços ofertados, todos notavam que o caminho para alcançar a visão era distante, e o projeto codinome "Red Dog" ainda teria que comer muito arroz com feijão para chegar lá.

Seis anos depois, encontramos uma realidade bastante diferente daquela do começo. Hoje, o Azure é sem dúvida uma plataforma muito completa, oferecendo de IaaS (*Infrastructure as a Service*) a SaaS (*Software as a Service*), passando por tudo que existe no meio, sendo referência em tudo que faz, e indo além.

Você pode hospedar desde um simples blog feito com WordPress até um complexo ERP baseado em Linux e Windows, com um back-end extremamente complexo. A escala é realmente infinita, e qualquer serviço parece pequeno diante da magnitude dos *data centers* disponibilizados a todos nós. Não é à toa que o Azure é considerado um dos líderes no segmento, sendo já hoje a melhor opção para diversos cenários, e disputando de forma competitiva em todas as outras em que atua.

No entanto, para mim, o maior apelo não é esse. Entregar uma

ampla gama de serviços é o mínimo que um *player* de *cloud* moderno tem a obrigação de oferecer. Ter tornado toda a plataforma fácil de usar e intuitiva foi muito bem-vindo, mas também não é o mais importante.

São duas as funcionalidades que considero que trazem o maior diferencial. A primeira é que Azure não é só uma plataforma de nuvem, mas parte de um ecossistema infinitamente maior construído pela Microsoft. Esse ecossistema inclui produtos como Windows e Office, mas vai além com serviços como o Office 365, e a total integração com o ambiente que já temos rodando com Windows Server e Active Directory, como também toda a suíte de gestão que os acompanham e que não há um segundo colocado nem mesmo próximo.

Gerenciar Azure é tão fácil quanto gerenciar os servidores que você já tem dentro da sua empresa, só que com um potencial infinitamente maior. Podemos rodar .NET, Java, NodeJS, C++, Ruby, Python, PHP, Go, entre diversas outras linguagens e plataformas no Azure, com Linux e Windows, e com todo suporte ao ALM que precisamos para construir a aplicação, e todo o suporte à operação depois que ela entrar no ar. Você vai de uma ponta a outra sem sair da plataforma.

O segundo ponto que me ganhou no Azure é o fato de que a Microsoft não quer te trancar em seu data center. Você pode, a qualquer momento, pegar os serviços, levar para dentro do seu próprio data center, e não usar mais os serviços da nuvem pública. Eles chegam ao ponto de oferecer ferramentas para auxiliar nesse processo, tornando essa ida ou volta à nuvem pública (ou privada) muito fácil, e fica melhor a cada dia.

A Microsoft permite que você tenha a sua própria nuvem privada, com os mesmos serviços que estão no Azure. Isso sem contar o investimento em padronização, facilitando ainda mais

processo, e a possibilidade de rodar contêineres com Docker, inclusive no Windows, algo que estamos agora começando a ver.

Sou desenvolvedor e arquiteto de software. Hoje, não consigo mais imaginar o que seria escrever uma aplicação sem ter planejado o software no Visual Studio Online, utilizando Scrum, com amplo suporte, e depois controlar todo o desenvolvimento, instalando uma nova versão em um ambiente no Azure a cada *check-in* do código, em um excelente suporte a DevOps.

Ao colocar a aplicação em produção usando IaaS ou PaaS, sei se ela continua saudável com o apoio do *Application Insights*, e da integração com *System Center*. Em momentos de pico, é fácil provisionar novas máquinas ou contêineres para dar conta da demanda. Tudo isso integrado à minha rede e ao meu domínio, com *single sign on* e segurança.

Deixar de usar o Azure significaria voltar à idade da pedra do desenvolvimento de software e da operação.

Fiquei feliz quando o Thiago me disse que estava escrevendo o livro que você está prestes a ler. Sei da sua paixão pela tecnologia, que ele detalhou nestas páginas. Desde que o conheço, temos animadas conversas sobre o Azure. Tenho certeza de que você vai confirmar que, depois de começar a usar o Azure, não terá mais motivos para usar outro serviço.

Com este livro, você vai entender mais a fundo por que a nuvem e a elasticidade são importantes e, logo no começo, vai conseguir criar uma aplicação com Visual Studio que roda no Azure. Ao longo do livro, você aprenderá alguns dos diversos serviços disponíveis e, quando terminar, você terá entendido e se aprofundado nas principais características da plataforma.

Boa leitura e bem-vindo a este mundo novo!

Por Giovanni Bassi

A quem se destina este livro?

Este livro é destinado principalmente aos desenvolvedores familiarizados com a plataforma .NET e aos que querem aprender a utilizar a plataforma de computação em nuvem Microsoft Azure. Não se preocupe caso você ainda não tenha muita experiência, isso se adquire com prática e, com algum tempo de estudo, você já conseguirá publicar seus aplicativos web no Azure e usufruir dos serviços integrados que ele oferece.

Os códigos foram escritos em C# utilizando a versão 2.5 do SDK. Programadores Java, PHP, Node.js, Python e Ruby poderão usufruir destes exemplos, dado que todas essas linguagens possuem SDK para Azure, disponível gratuitamente para download em http://bit.ly/AzureSDKs.

Caso você possua alguma dúvida durante a leitura, ou queira discutir algum dos assuntos tratados, entre no Google Groups do livro, em http://forum.casadocodigo.com.br/. Você será muito bem-vindo!

Lembre-se também de que todos os exemplos do livro encontram-se em https://github.com/thdotnet/ExemplosLivroAzure.

Sumário

Versão: 19.5.14

Capítulo 1

VISÃO GERAL DA COMPUTAÇÃO EM NUVEM

Nos últimos anos, a consultoria de pesquisas sobre tecnologias, a Gartner, aponta a computação em nuvem como uma das principais tendências estratégicas na área de tecnologia de informação. A consultoria define como tendência estratégica aquela com potencial de causar impacto significativo em uma corporação em um período de até três anos.

1.1 MAS AFINAL, O QUE É COMPUTAÇÃO EM NUVEM?

Você com certeza já é cliente de serviços em nuvem: seu e-mail pessoal, serviços de armazenamento de arquivos, como OneDrive ou Dropbox, redes sociais, entre muitos outros.

Computação em nuvem (ou *cloud computing*) diz respeito à entrega sob demanda de recursos de TI e aplicativos pela internet, seguindo o princípio da computação em grade (*grid computing*). Em outras palavras, você passa a acessar dados e softwares diretamente na internet em vez de tê-los instalados e armazenados no seu próprio computador. A palavra "nuvem" é apenas uma analogia para internet.

Computação em grade

"É um modelo computacional capaz de alcançar uma alta taxa de processamento dividindo as tarefas entre diversas máquinas, podendo ser em rede local ou rede de longa distância, que formam uma máquina virtual." — Fonte: Wikipédia (http://bit.ly/ComputacaoEmGrade)

1.2 TIPOS DE NUVEM

Basicamente, existem dois tipos de nuvem: a pública e a privada.

Nuvem pública

É um data center acessível via internet em um domínio público. Algumas plataformas de nuvem pública permitem uma integração com a rede da sua empresa por meio de federação e redes virtuais privadas. Uma importante característica de nuvens públicas é que não há um custo mínimo ou mensal. Você paga apenas pelo que você usou, e apenas pelo período de utilização.

Nuvem privada

É uma infraestrutura de *cloud* que roda no data center da sua empresa. Você pode construir seu próprio Azure utilizando tecnologias Microsoft, como o Windows Server 2012 e Azure Pack.

1.3 O QUE SIGNIFICA IAAS, PAAS E SAAS?

É comum ouvirmos termos IaaS, PaaS e SaaS quando o assunto é computação em nuvem. Vamos ver o que cada sigla significa:

IaaS – Infraestrutura como Serviço (Infrastructure as a Service)

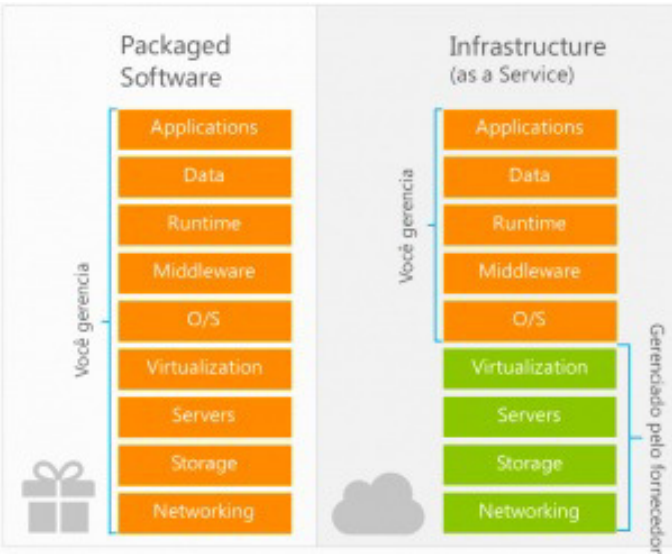

Figura 1.1: Comparativo – Infraestrutura como serviço vs. Hospedar no próprio data center (on-premises)

IaaS abstrai o hardware e a infraestrutura de virtualização. Neste modelo, o cliente "aluga" a infraestrutura fornecida por um fornecedor, e hospeda sua aplicação sobre ela. Poucos fornecedores oferecem um sistema operacional (SO), logo, o cliente é responsável por instalar e manter o SO.

PaaS – Plataforma como Serviço (Platform as a Service)

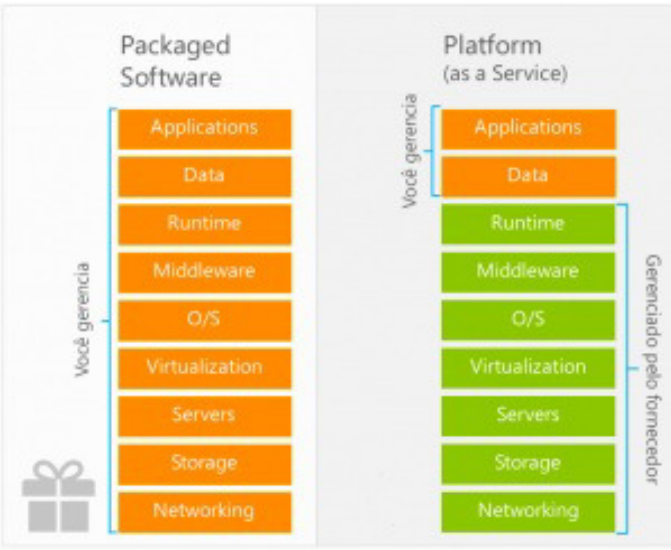

Figura 1.2: Comparativo – Plataforma como serviço vs. Hospedar no próprio data center (on-premises)

PaaS fornece hardware, sistemas operacionais e *runtime* para a execução dos aplicativos nos data centers do fornecedor. Neste modelo, o cliente paga por uma plataforma em que ele publica seus aplicativos sem se preocupar com a configuração da infraestrutura.

SaaS – Software como Serviço, ou software sob demanda (*Software as a Service*)

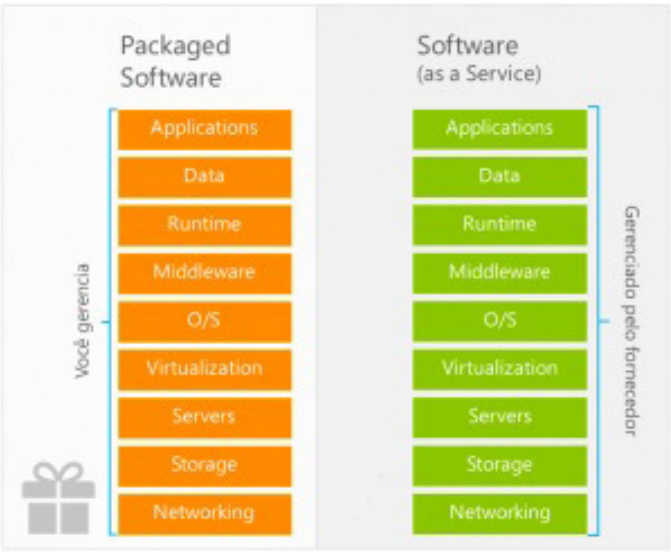

Figura 1.3: Comparativo – Software como serviço vs. Hospedar no próprio data center (on-premises)

SaaS oferece uma aplicação de ponta a ponta. Neste modelo, o cliente paga por aplicativos que são entregues por meio de um modelo de prestação de serviço. Os aplicativos estão hospedados na nuvem, e o cliente acessa através da internet.

Em resumo:

Figura 1.4: Hospede, construa, consuma

1.4 QUANDO USAR COMPUTAÇÃO EM

NUVEM?

Existem alguns cenários que são ideais para computação em nuvem, isto é, ao analisarmos casos de sucesso de migração ou construção de novas aplicações para a nuvem, os seguintes padrões foram identificados:

Rápido crescimento

Tipicamente representado por uma startup que começa operando com o mínimo necessário, e vai escalando a infraestrutura conforme a demanda.

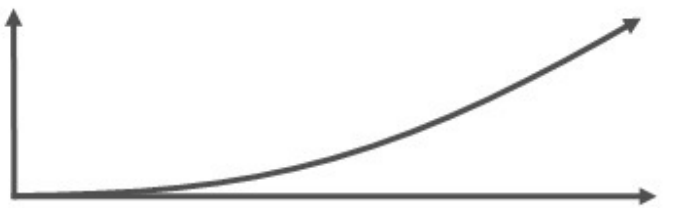

Figura 1.5: Padrão – Rápido crescimento

Crescimento esperado

Cenário ideal para aplicativos que vão demandar uma maior quantidade de servidores por um determinado período de tempo. Pensando em *e-commerce*, pode ser aquela época festiva ou de promoções, como a *black friday*, por exemplo. A nuvem é ideal neste cenário, pois você pode contratar mais servidores apenas para este período e, logo após a sua utilização, voltar para a quantia inicial.

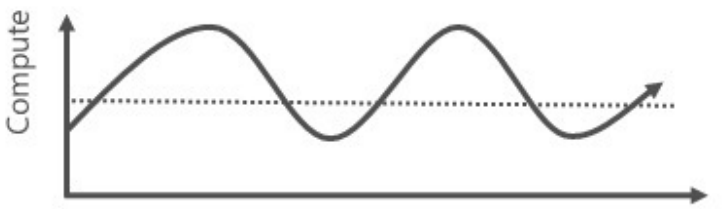

Figura 1.6: Padrão – Crescimento esperado

Crescimento inesperado

Suponha que você lançou uma aplicação *beta* e divulgou apenas para seus amigos. Ao monitorar os acessos em tempo real, você identificou que muito mais gente está acessando a sua aplicação, e aquele dimensionamento inicial de servidor já não é o suficiente. Este "sucesso" pode acontecer por N motivos. Imagine que acidentalmente o link da sua aplicação chegou até alguma celebridade, e esta fez uma divulgação da sua aplicação em uma rede social.

Com o sucesso meteórico, sua aplicação pode cair facilmente no limbo se uma grande quantidade de usuários tentar acessá-la e receber uma tela de erro, ou uma mensagem para aguardar em uma "fila". Para evitar a "queima" da aplicação/marca, ela deve estar preparada para crescer rapidamente (escalar).

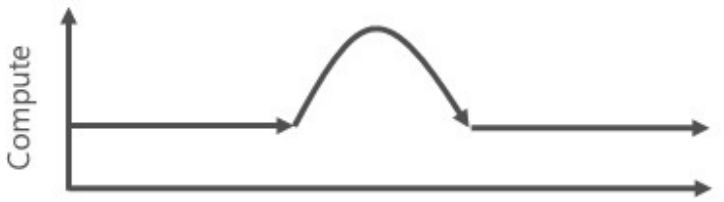

Figura 1.7: Padrão – Crescimento inesperado

Ligar e desligar

Este cenário é similar ao **crescimento esperado**, no entanto, para curtos períodos de tempo. Um relatório que demora horas para gerar um gráfico poderia ter a mesma resposta e em muito menos tempo "ligando" mais servidores até o término do processamento, e desligando-os logo em seguida.

Figura 1.8: Padrão – Ligar e desligar

1.5 CONCLUINDO

Neste capítulo, vimos alguns termos comuns quando o assunto é computação em nuvem. No capítulo seguinte, vamos conhecer um pouco mais sobre a plataforma de computação em nuvem da Microsoft, o Microsoft Azure.

Capítulo 2

CONHECENDO O MICROSOFT AZURE

2.1 O NASCIMENTO DO AZURE

Era o ano de 2008, durante o evento *Professional Developers Conference* (PDC) realizado no Los Angeles Convention Center, quando a Microsoft anunciou o então chamado *Windows Azure*, sua mais nova aposta e plataforma para computação em nuvem. Ray Ozzie, arquiteto chefe de software da Microsoft, discursava no keynote:

"É uma transformação do nosso software e uma transformação da nossa estratégia." — Ray Ozzie

Embora muito embrionário naquele momento, já era possível ver que a Microsoft estava investindo muito neste segmento, e que a plataforma seria uma peça chave na mudança de estratégia da empresa.

Caso você queira assistir ao Keynote completo do PDC 2008, ele encontra-se disponível gratuitamente em http://bit.ly/KeynotePDC2008.

Mas, afinal, o que é a plataforma Azure?

Eis a definição da própria Microsoft, retirada dos slides do Scott Guthrie na conferência AzureConf de 2013:

"Uma coleção crescente de serviços integrados, computação, armazenamento, dados, rede e aplicativos, que ajudam você a avançar mais rápido, realizar mais e economizar dinheiro." — Scott Guthrie

Para assistir ao keynote do AzureConf de 2013, acesse http://bit.ly/Azureconf2013Keynote.

Na sequência, veja alguns exemplos práticos desta definição para entender melhor.

2.2 COMPUTAÇÃO

Anualmente e em diversas cidades do mundo, é realizado o Global Windows Azure Bootcamp, um evento gratuito que tem a finalidade de difundir e ensinar como utilizar o Azure. No evento de 2014, o poder computacional do Azure foi usado para ajudar na pesquisa da doença diabetes.

Durante uma das atividades, os participantes do evento, além de aprenderem como utilizar máquinas virtuais, estavam compartilhando o poder de processamento dessas máquinas com o instituto *Pacific Northwest National Lab* (PNNL), que desenvolve tecnologias para análise avançada de carboidratos a nível molecular, usando espectrometria de massa.

Observação: esse é um exemplo de processamento distribuído utilizando computação em grade.

2.3 ARMAZENAMENTO

Existem diversas opções para armazenamento de dados no Microsoft Azure: Blob Storage, Table Services, SQL Database e DocumentDB. Na comunidade de desenvolvimento de software, há

um movimento crescente que acredita em persistência poliglota, isto é, usar o melhor mecanismo de armazenamento para determinado cenário.

Caso você queira construir uma rede social de imagens, você poderia utilizar o serviço de Blob Storage para armazenar os arquivos dos usuários. Já se você precisar de armazenamento não relacional baseado em chave/valor, você pode recorrer ao Azure Table Services.

Você precisa apenas das features básicas do SQL Server? Você pode utilizar o SQL Database, que é uma versão customizada do SQL Server, na qual a parte administrativa do banco (particionamento, escalabilidade e backup) ficam sob responsabilidade da própria Microsoft.

O DocumentDB é uma nova opção de banco de dados baseado em documentos, mas que possui suporte a ferramentas conhecidas e consolidadas do mundo de banco dados relacional, como *triggers*, *stored procedures* e a própria linguagem SQL.

Já palestrei sobre Azure DocumentDB e sobre esse movimento de persistência poliglota, na edição de 2014 do evento Azure Summit Brasil. Você pode conferir essa palestra em http://bit.ly/PalestraAzureSummit.

Observação: vamos explorar cada uma destas opções nos capítulos a seguir.

2.4 SERVIÇOS INTEGRADOS

Existem diversos serviços integrados à plataforma Azure. Estes visam resolver um problema em específico, e facilitar a vida do desenvolvedor e/ou melhorar a experiência do usuário final.

- **Azure Search:** já pensou em integrar a inteligência dos sites de busca no seu site, mas sem a complexidade de usar frameworks, nem manter uma infraestrutura para isso? Esta é a proposta do Azure Search, um serviço de busca como serviço (*Search as a Service*), que evita o aborrecimento de lidar com corrompimento de índice, disponibilidade do serviço, dimensionamento, entre outras complexidades.

- **Traffic Manager:** serviço que efetua cópias do seu aplicativo entre os data centers do Azure, e distribui o tráfego ao mais próximo do usuário, garantindo menor tempo de resposta.

2.5 DIFERENCIAIS DO AZURE

A plataforma Microsoft Azure foi desenhada para suportar sistemas operacionais da família Windows ou Linux. Se você já trabalha com algum software de virtualização na sua empresa, você pode efetuar o upload do disco virtual para o Azure, e a Microsoft se encarrega do provisionamento da máquina, energia, internet, cabeamento, firewall etc. Em outras palavras, você passa a utilizar a infraestrutura do Azure como serviço.

SEM TRAVA!

Você pode a qualquer momento efetuar o download desse disco virtual e hospedar novamente na infraestrutura da sua empresa, ou até mesmo ir para outro fornecedor de nuvem pública.

Não é necessário o uso de linguagens da plataforma .NET, você

pode criar soluções e usufruir dos serviços disponíveis no Azure programando em diversas linguagens (Java, Node.js, Php, Ruby, Python). O SDK para Azure de cada uma destas linguagens está disponível para uso no GitHub. Você inclusive pode colaborar com o desenvolvimento e melhoria destes SDKs para a sua linguagem favorita.

Quer ver como o SDK funciona? Acesse diretamente a conta do Azure no GitHub: https://github.com/Azure.

2.6 QUEM UTILIZA O AZURE?

A Microsoft é cliente do Azure. Diversos serviços como Xbox Live, Outlook, Hotmail, Office 365, Msn e muitos outros são hospedados no Azure. Além da Microsoft, 57% das 500 maiores empresas, segundo a Fortune, também utilizam o Azure.

São tantos clientes que, atualmente, existem 19 data centers espalhados pelo globo, sendo que um deles se encontra na região sudeste do Brasil.

Figura 2.1: Data centers do Azure disponíveis até o momento

Veja alguns números:

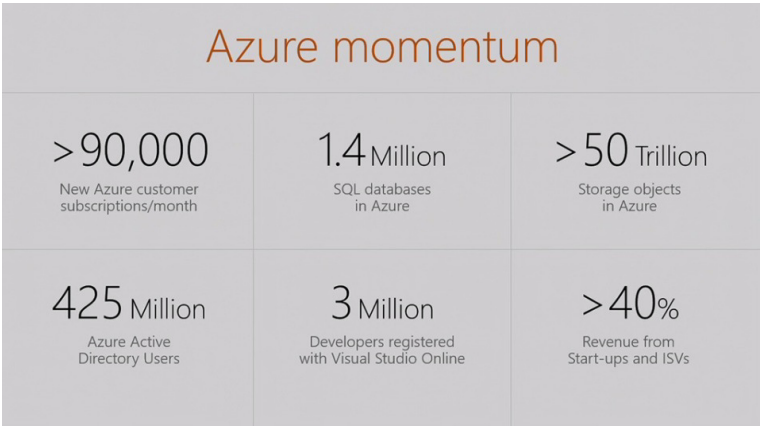

Figura 2.2: Números do Azure divulgados no evento Build 2015

Convido você a ler alguns cases de sucesso utilizando Azure, visitando a seguinte URL: http://azure.microsoft.com/en-us/case-studies/.

2.7 DE WINDOWS AZURE PARA MICROSOFT AZURE

No dia 03 de março de 2014, a Microsoft decidiu trocar o nome de "Windows Azure" para "Microsoft Azure". Esse ato está relacionado com a mudança de estratégia da empresa: de não ser vista como uma empresa de produtos, mas como uma empresa fornecedora de dispositivos e serviços integrados. Além disso, a mudança de nome teve uma outra importância: fortalecer a imagem de nuvem pública do Azure desvinculando ao produto Windows.

2.8 CONCLUINDO

Neste capítulo, aprendemos um pouco sobre a plataforma Microsoft Azure e seus principais diferenciais em relação aos outros fornecedores. No capítulo seguinte, vamos instalar o Visual Studio Community e o SDK para o Azure.

Capítulo 3

SETUP SOFTWARES E ASSINATURA

Antes de iniciarmos com os desenvolvimentos, vamos instalar a IDE Visual Studio Community e o SDK para o Azure.

3.1 PRÉ-REQUISITOS

Nos próximos capítulos, vamos começar a usar de fato o Microsoft Azure. Os pré-requisitos para usufruir dos exemplos que serão dados são:

- Assinatura do Azure;
- Visual Studio;
- Azure SDK 2.5.

R$ 560 em créditos

Crie sua assinatura no Microsoft Azure em http://bit.ly/TrialMSAzure.

Esse link lhe dá R$ 560,00 em créditos para você testar e avaliar o Azure por 30 dias.

Caso você não possua o Visual Studio, faça o download gratuito

do Microsoft Visual Studio Community Edition em http://bit.ly/VsCommunityEdition.

Após abrir a página, basta clicar sobre o link `Download` , e seguir o assistente de instalação.

Para download do SDK do Azure, acesse http://bit.ly/AzureSDKs.

3.2 PORTAL AZURE

No evento Build de 2014, a Microsoft anunciou uma nova versão do portal para a administração dos serviços contratados no Azure e um melhor acompanhamento dos gastos. Determinados serviços só podem ser contratados nesta nova versão, enquanto alguns outros só podem ser na versão antiga.

Sempre ao iniciarmos um novo capítulo, informarei qual versão do portal deverá ser utilizada. Para isto, armazene as URLs a seguir, pois elas serão usadas constantemente.

- Portal Ibiza (Anunciado no Build 2014) – http://portal.azure.com

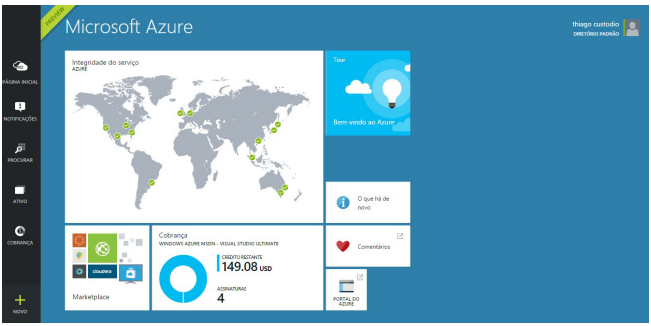

Figura 3.1: Portal Ibiza

- Portal de Gerenciamento do Azure

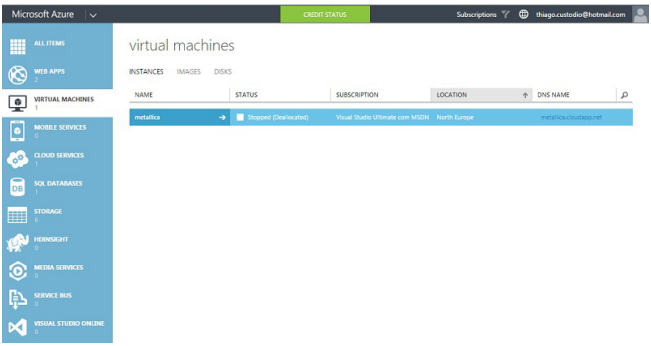

Figura 3.2: Portal de gerenciamento do Azure

CAPÍTULO 4

HOSPEDANDO NO AZURE WEBAPP

4.1 PRIMEIRO WEBAPP NO AZURE

Um dos possíveis modelos de execução no Microsoft Azure são os WebApps, ou aplicativos web. Os WebApps permitem a criação de aplicativos altamente escaláveis.

Para criar o seu primeiro WebApp, autentique-se no Portal de Gerenciamento do Azure, por meio da URL: http://manage.windowsazure.com.

Localize e clique no menu `Novo` no canto inferior à esquerda e, em seguida, selecione a opção: `Compute` (Computação) -> `Web App` (Aplicativo Web) -> `Quick Create` (Criação Rápida).

Para prosseguir com a criação, informe o prefixo da URL para o seu aplicativo:

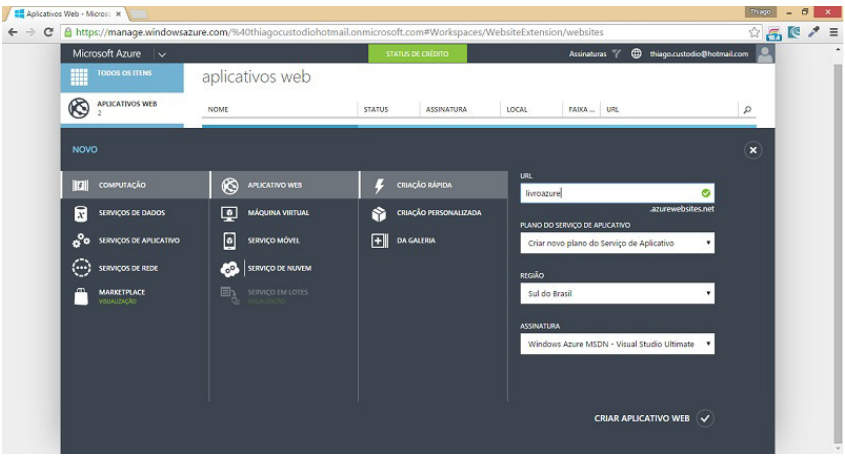

Figura 4.1: Criação de novo WebApp

Sobre os prefixos das URLs

Os prefixos precisam ser únicos e possuir de 2 a 60 caracteres, sendo aceitos apenas letras, números e hifens.

Além da URL para o aplicativo, também é possível criar um plano de serviço de aplicativo e escolher em qual assinatura e data center o WebApp vai ficar (caso você possua mais de uma assinatura no Azure vinculada ao seu usuário).

Por fim, basta clicar em `Criar aplicativo Web` e, em poucos segundos, seu aplicativo web estará disponível para acesso.

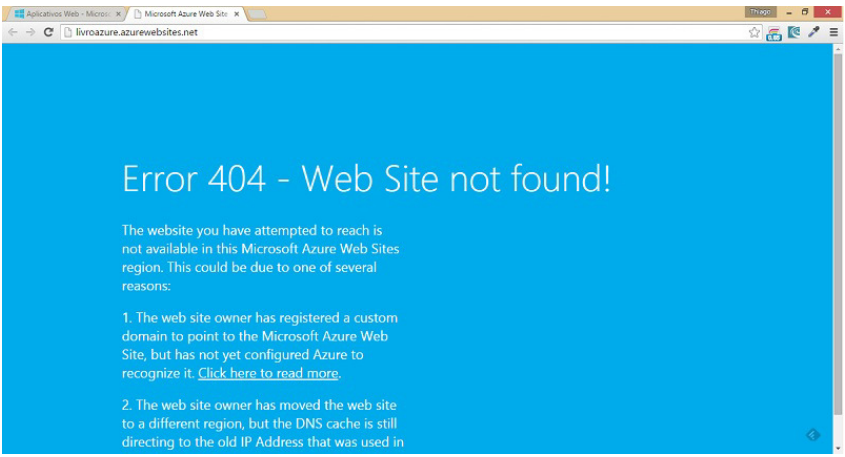

Figura 4.2: Novo WebApp criado

4.2 PUBLICANDO COM VISUAL STUDIO E O PERFIL DE PUBLICAÇÃO

Nesta seção, vamos ver o modo de publicação utilizando o arquivo `XML` chamado de **Perfil de publicação**. Assim que seu WebApp é criado, o Azure cria automaticamente esse arquivo com as informações necessárias para publicação, como credenciais (usuário/senha) e o endereço do WebApp.

Basta importar esse arquivo no Visual Studio, e ele se encarrega de copiar os arquivos para o servidor. Para baixar o arquivo do **Perfil de Publicação**, basta selecionar a opção `Aplicativos Web` no menu lateral à esquerda, depois clicar no nome do seu WebApp (primeira coluna).

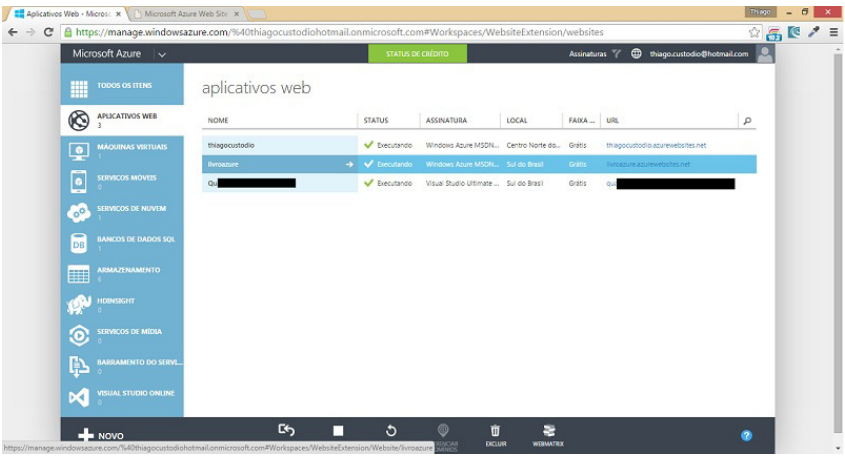

Figura 4.3: Seleção do WebApp no portal de gerenciamento do Azure

Na tela seguinte, localize e clique sobre o link `Baixar o perfil de publicação` , como mostra a figura a seguir:

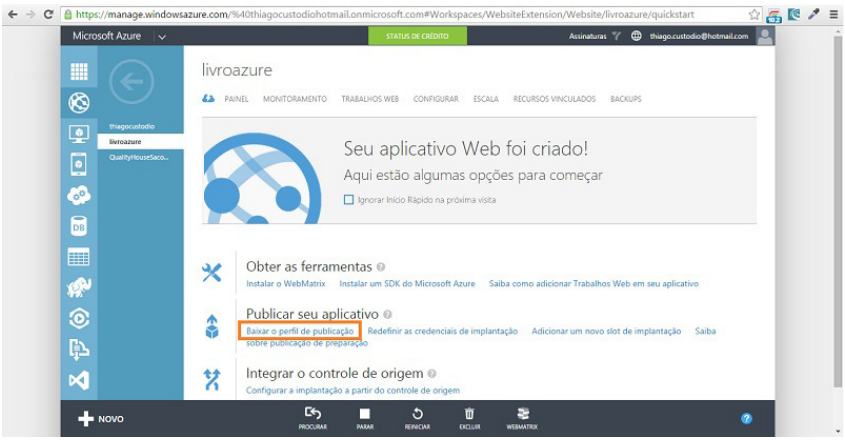

Figura 4.4: Link para download do perfil de publicação"

Será realizado o download do XML no diretório padrão de downloads utilizado pelo seu navegador. Neste arquivo estão configuradas as publicações via FTP/Webdeploy ao WebApp que acabamos de criar. Sendo assim, basta importar esse arquivo no Visual Studio.

```xml
<publishData>
    <publishProfile profileName="livroazure - Web Deploy"
                    publishMethod="MSDeploy"
                    publishUrl="livroazure.scm.azurewebsites.net:443"
                    msdeploySite="livroazure"
                    userName="$livroazure"
                    userPWD="PveKfjFETAblZ119Yj5XgQBa7elp11tBTeNmWJ3bFph4Minpcnu06cgB6ynw"
                    destinationAppUrl="http://livroazure.azurewebsites.net"
                    SQLServerDBConnectionString=""
                    mySQLDBConnectionString=""
                    hostingProviderForumLink=""
                    controlPanelLink="http://windows.azure.com"
                    webSystem="WebSites">
        <databases/>
    </publishProfile>
    <publishProfile profileName="livroazure - FTP"
                    publishMethod="FTP"
                    publishUrl="ftp://waws-prod-cq1-003.ftp.azurewebsites.windows.net/site/wwwroot"
                    ftpPassiveMode="True"
                    userName="livroazure\$livroazure"
                    userPWD="PveKfjFETAblZ119Yj5XgQBa7elp11tBTeNmWJ3bFph4Minpcnu06cgB6ynw"
                    destinationAppUrl="http://livroazure.azurewebsites.net"
                    SQLServerDBConnectionString=""
                    mySQLDBConnectionString=""
                    hostingProviderForumLink=""
                    controlPanelLink="http://windows.azure.com"
                    webSystem="WebSites">
        <databases/>
    </publishProfile>
</publishData>
```

Figura 4.5: Conteúdo do XML do perfil de publicação

Para demonstrar a publicação, vamos criar uma nova aplicação WEB no Visual Studio.

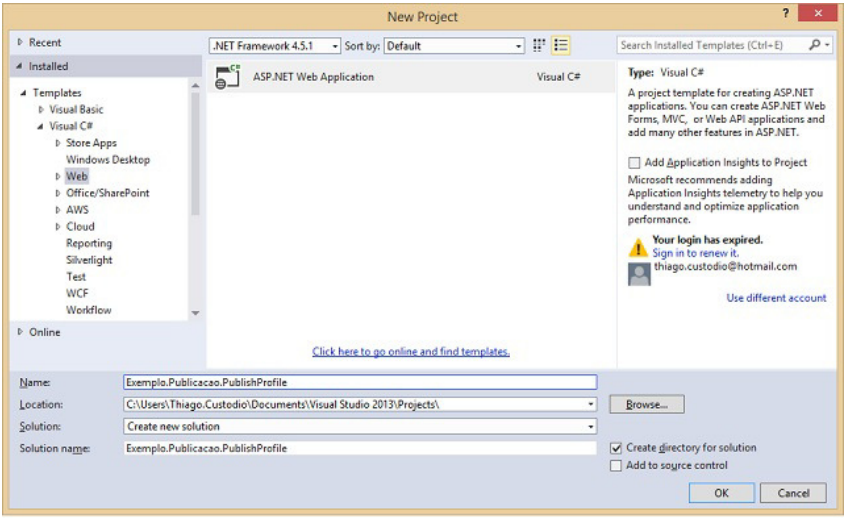

Figura 4.6: Novo projeto no Visual Studio

Em seguida, selecione a opção `MVC` e clique em `OK`.

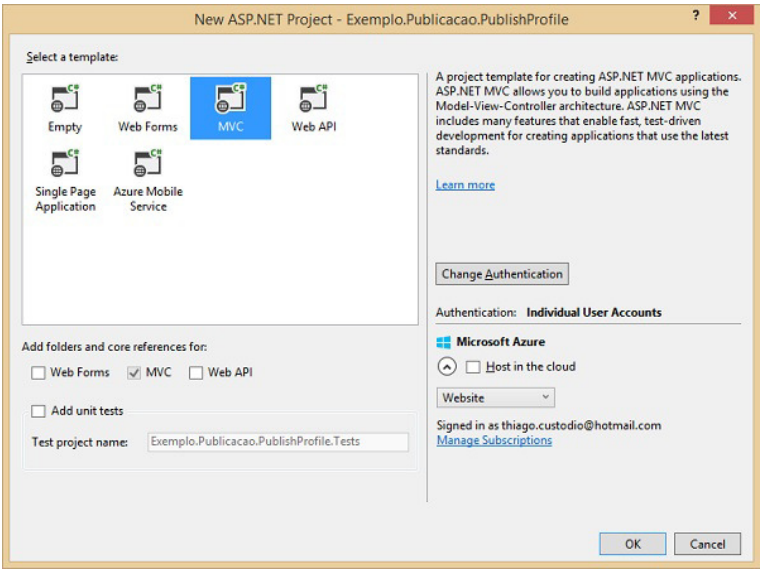

Figura 4.7: Seleção do tipo do novo projeto

Logo após isso, o Visual Studio criará o projeto. Vamos alterar o código da `View Index` da pasta `Home` .

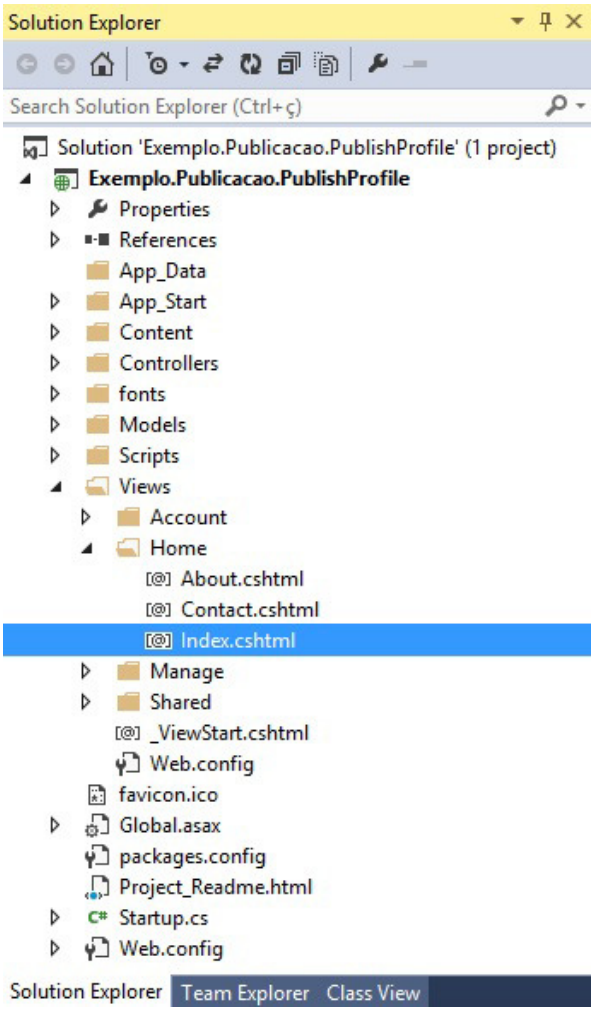

Figura 4.8: Caminho onde se encontra a View Index do controller Home

Substitua o código da `View` pelo trecho a seguir:

```
@{
    ViewBag.Title = "Home Page";
}
<div>
    Publicando um webapp utilizando o perfil de publicação.
</div>
<div>
    Data da publicação: @DateTime.UtcNow.AddHours(-3);
```

```
</div>
```

Após esta alteração, basta clicar com o botão da direita sob o nome do projeto no menu `Solution Explorer`, e selecionar a opção `Publish`:

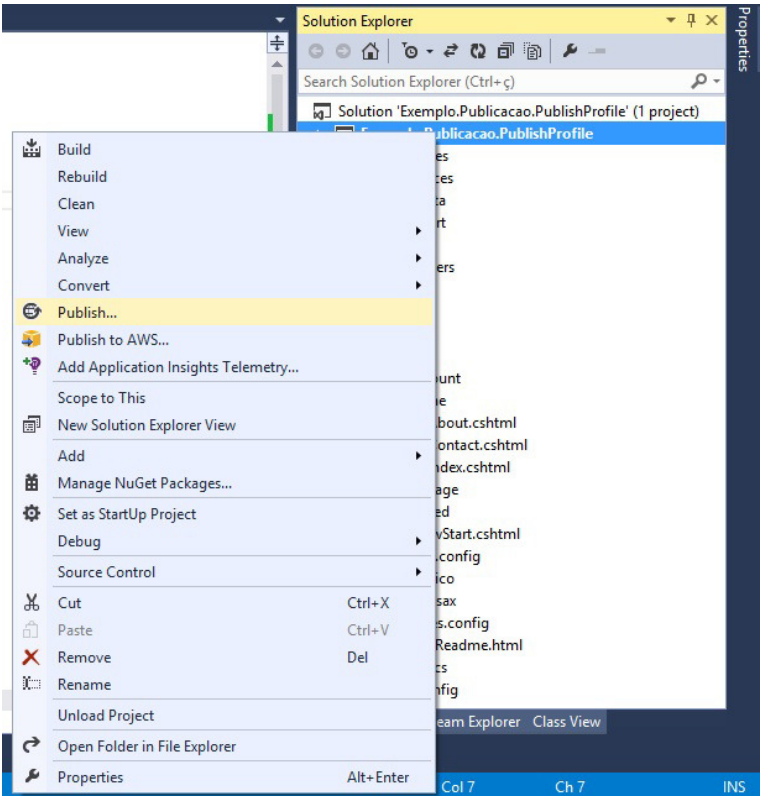

Figura 4.9: Publicando pelo Visual Studio

Na tela seguinte, selecione a opção `Import`:

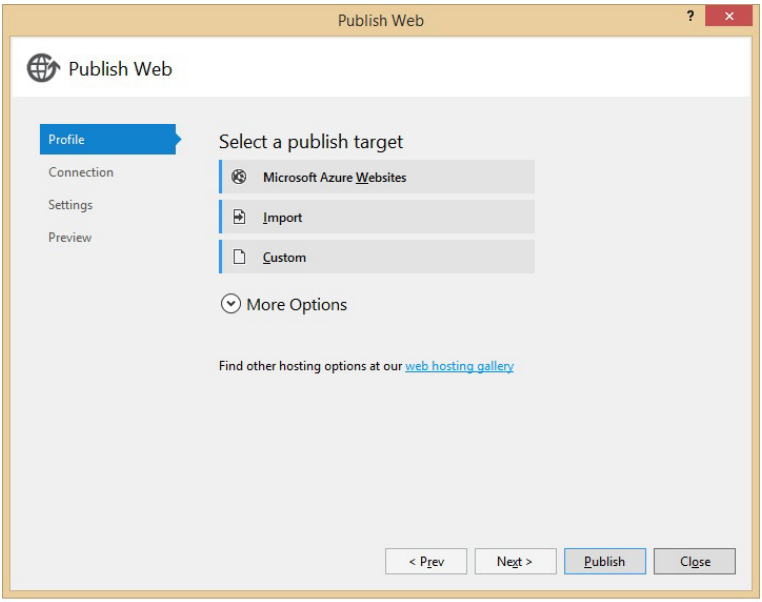

Figura 4.10: Seleção do tipo de publicação

Então, localize o diretório padrão de download do seu navegador e pressione `Ok` :

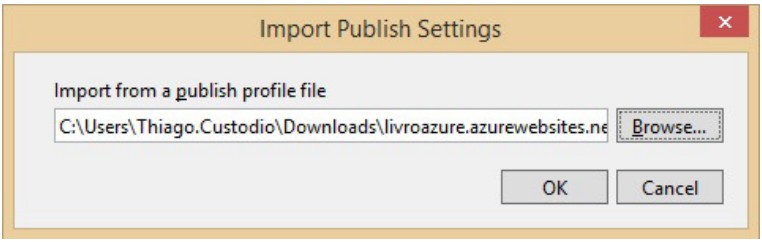

Figura 4.11: Caminho no qual se encontra o arquivo baixado anteriormente

Repare que, na figura a seguir, as informações para a publicação já estão preenchidas nos campos. Basta selecionar o botão `Publish` e aguardar a publicação.

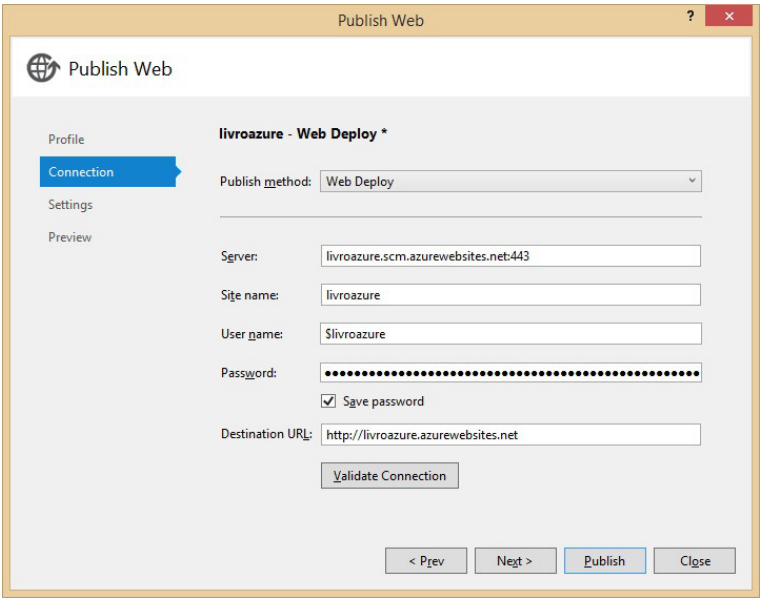

Figura 4.12: Configurações setadas automaticamente

O próprio Visual Studio abre o WebApp para que você possa conferir o resultado ao término da operação:

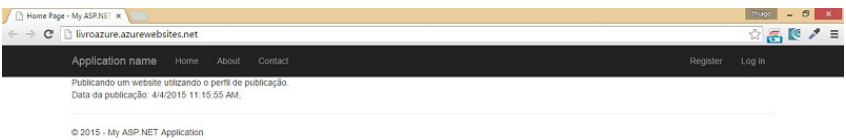

Figura 4.13: Fim da publicação – WebApp publicada"

4.3 SEU WEBAPP VIROU UM SUCESSO?

ESCALE OS SERVIDORES!

Além da facilidade para publicar WebApps no Azure, umas das principais vantagens é a habilidade para escala. Ou seja, você pode aumentar a capacidade de processamento (vertical) e/ou adicionar mais servidores (horizontal) para atender as requisições de acesso, sem ter de se preocupar com complexidades de infraestrutura.

- **Escalabilidade vertical**:

  Significa adicionar recursos em um servidor. Exemplo: adicionar mais memória, mais disco, mais CPU.

- **Escalabilidade horizontal**:

  Significa adicionar mais servidores para atender à demanda. A distribuição da carga é balanceada entre os servidores.

4.4 ESCALANDO VERTICALMENTE

Modo de hospedagem gratuito

Ao publicar um WebApp no Azure, por padrão ele está sob o modo de hospedagem gratuito. Neste modo, o seu aplicativo compartilha os recursos de um servidor com aplicativos de outros clientes.

Para garantir que os recursos sejam compartilhados de maneira uniforme entre os WebApps, alguns limites são estabelecidos. No modo gratuito, 165MB de saída são permitidos por dia. Essa limitação está atrelada à sua assinatura, sendo assim, se você possui mais de um WebApp no modo gratuito, essa quantia é distribuída entre eles.

Importante

Ao atingir a quota (limite), **TODOS** os WebApps daquela assinatura são paralisados. Você pode acompanhar a utilização dos recursos pelo painel de administração do Azure, na seção `Monitoramento`.

As quotas são automaticamente renovadas diariamente. Se seu WebApp parou, considere migrar para outro modo de hospedagem, ou aguarde até a próxima renovação de quotas no dia seguinte.

	GRATUITO	COMPARTILHADO	BÁSICO	STANDARD
Número de sites/aplicativos exclusivos	10	100	Unlimited	Unlimited
Espaço em Disco	1 GB	1 GB	10 GB	50 GB
Largura de banda diária máxima	Até 165 MB por dia	Unlimited	Unlimited	Unlimited
Suporte de domínio DNS personalizado	--	✓	✓	✓
Suporte de certificado SSL	--	--	Os preços de SSL se aplicam	5 SSL SNI e 1 IP SSL incluídos gratuitamente
Instâncias de VM Máximas	--	6 instâncias compartilhadas	3 instâncias de VM dedicadas	10 instâncias de VM dedicadas
Suporte de Autoescala	--	--	--	✓
Slots de Publicação de Preparo	--	--	--	5 slots por site
Soquetes Web	5 connections	35 connections	350 connections	Ilimitado
Backups Automatizados	--	--	--	✓
Gerenciamento de Tráfego Global	--	--	--	✓
Contrato de Nível de Serviço	Não disponível	Não disponível	99,95%	99,95%

Figura 4.14: Listagem das quotas nos modos de hospedagem

Modo de hospedagem compartilhado

O modo de hospedagem compartilhado é bem similar ao modo gratuito, no entanto, não existem limitações em relação ao tráfego de saída. Os primeiros 5GB de dados de saída são gratuitos e, ao exceder essa quantia, os dados são cobrados no modelo "pague conforme o uso".

Modo standard

WebApps neste modo rodam em sua própria máquina virtual. Graças a isto, eles conseguem obter uma performance melhor e sem limitações. Nesse modo, também é possível escolher o tamanho das máquinas virtuais:

- Pequena (1 core e 1.75GB de memória);
- Média (2 cores, 3.5GB de memória);
- Grande (4 cores, 7GB de memória).

> **Importante**
>
> Se você possui mais de um aplicativo nesse modo, eles vão ficar sob a mesma máquina virtual, logo, os recursos computacionais (CPU, memória e disco) serão compartilhados entre os aplicativos.

Para maiores informações sobre as quotas, consulte http://bit.ly/QuotasAzureFree.

4.5 ESCALANDO HORIZONTALMENTE

Modo de hospedagem compartilhado

No modo de hospedagem compartilhado, você pode escalar para

até 6 instâncias, isto é, existirão 6 cópias do seu WebApp que receberão os acessos dos usuários.

Modo standard

No modo de hospedagem *standard*, você pode escalar para até 10 instâncias. Além disso, é possível escolher se todos os WebApps neste modo receberão essa escala, ou se apenas o WebApp selecionado.

Um recurso muito interessante do modo standard é o *Auto Scaling*, no qual você pode configurar para adicionar mais instâncias em horários predeterminados, ou quando um determinado nível de processamento (CPU) for atingido.

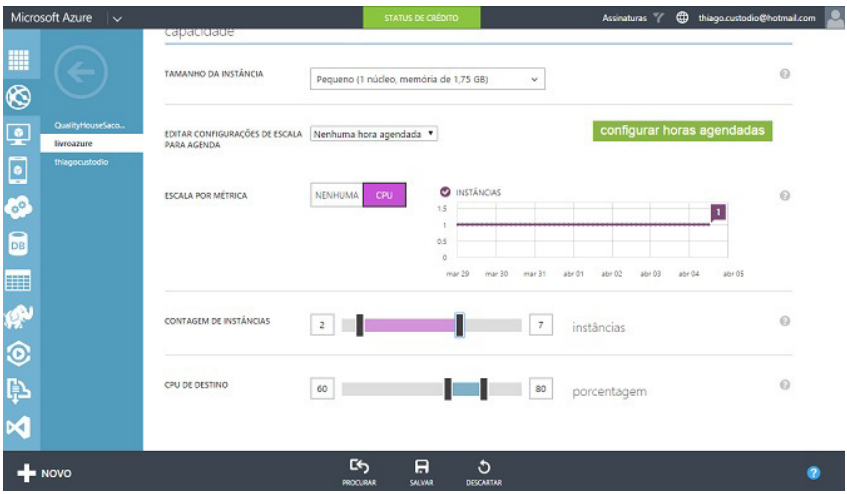

Figura 4.15: Escalando o WebApp pelo portal do Azure

No exemplo anterior, configurei nosso WebApp para começar com duas instâncias e aumentar de uma em uma até o limite de sete, sempre que o nível de CPU atingir 60%.

4.6 INTEGRAÇÃO COM VERSIONADORES DE

CÓDIGO

Existem diversas maneiras para habilitar a publicação automática em Azure WebApps. Você pode usar TFS, Git Local, Dropbox e muitos outros meios. Após promover uma nova versão de código ao versionador, é possível configurar para que ele publique automaticamente em um Azure WebApps.

Eu poderia escrever sobre esses meios aqui neste livro, no entanto, acredito que para deixar bem detalhado, seriam necessárias muitas páginas e diversos screenshots. Meu amigo Ricardo Serradas (ALM Ranger) já criou uma série de vídeos relacionados a esse tema. Sendo assim, vou deixar os links para esses vídeos para irmos para o próximo assunto:

- **Publicando no Azure com Visual Studio Online** – http://bit.ly/PublicandoNoAzureVSO

  Neste vídeo, você aprenderá como configurar a publicação contínua no Azure WebApp utilizando o versionador Visual Studio Online, uma versão online do Team Foundation Service. A cada novo *check-in*, uma versão é publicada no Azure WebApp.

- **Continuous Deployment no Azure com Dropbox** – http://bit.ly/PublicandoNoAzureDropbox

  Neste vídeo, você vai aprender como configurar a publicação contínua utilizando um pacote no Dropbox.

- **Continuous Deployment no Azure com GitHub** – http://bit.ly/PublicandoNoAzureGitHub

  Neste vídeo, você aprenderá como configurar a publicação contínua a partir do GitHub. Você entra com suas credenciais do GitHub, escolhe o repositório,

e o Azure monitora alterações nesse repositório, além de efetuar a publicação no WebApp.

- **Continuous Deployment no Azure com Git Local** – http://bit.ly/PublicandoNoAzureGitLocal

  Neste vídeo, você aprenderá como configurar a publicação contínua a partir de um repositório local no próprio WebApp. A cada novo *push* para o repositório, uma nova versão será publicada.

- **Continuous Deployment no Azure com Bitbucket** – http://bit.ly/PublicandoNoAzureBitBucket

  Neste vídeo, você aprenderá como configurar a publicação contínua usando o Bitbucket.

4.7 SLOT DE IMPLANTAÇÃO

Um excelente recurso que foi adicionado aos Azure WebApps são os slots de implantação. Um slot de implantação é uma cópia de seu WebApp que pode ser usada para validar determinado comportamento antes de se publicar de fato no ambiente de produção.

Por exemplo, você pode criar slots para homologação, QA, Dev e validar uma alteração em tempo real, sem afetar o que está em produção. A grande vantagem é que, se algo der errado, os usuários do site não serão afetados.

A princípio, esse recurso só estava disponível em Web Roles, mas após inúmeros feedbacks dos clientes do Azure, o time da Microsoft o adicionou aos WebAps também.

Observação

Este recurso só está disponível no portal Ibiza (http://portal.azure.com). Para maiores informações, consulte http://bit.ly/SlotImplantacaoAzure.

O recurso de slots de implantação é uma oferta do modo standard, você pode utilizá-lo ou não. No entanto, não é adicionado nenhum custo extra, caso você use todos os slots disponíveis.

4.8 MONITORAMENTO

No portal de administração do Azure, é possível coletar algumas informações do seu WebApp, como, por exemplo, erros que aconteceram no lado do servidor, tráfego de dados de entrada/saída, tempo de processamento, entre outros.

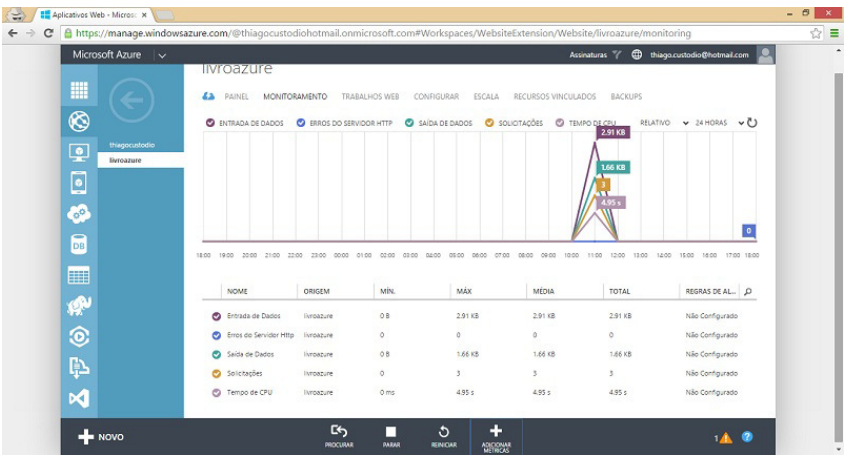

Figura 4.16: Painel de monitoramento

O gráfico responde em tempo real, sendo assim, você pode habilitar/desabilitar métricas para que o gráfico exiba exatamente os eventos que são de seu interesse.

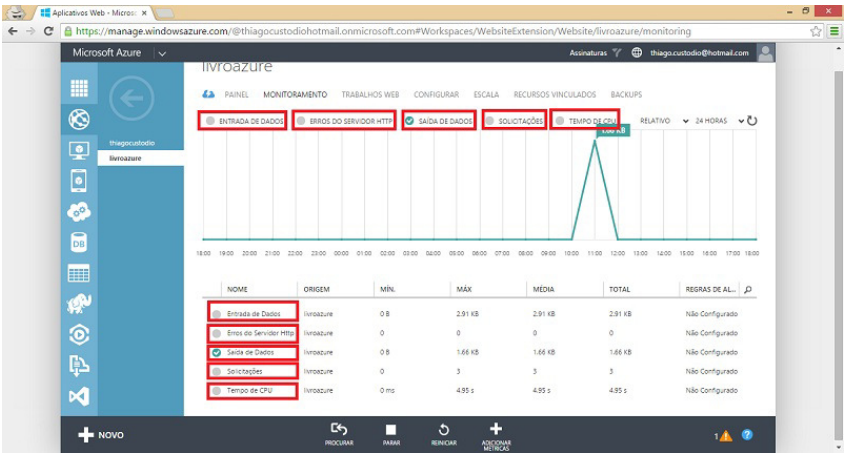

Figura 4.17: Métricas utilizadas

Além das métricas que são exibidas por padrão nesta seção, você pode adicionar/remover métricas. Veja na figura a seguir:

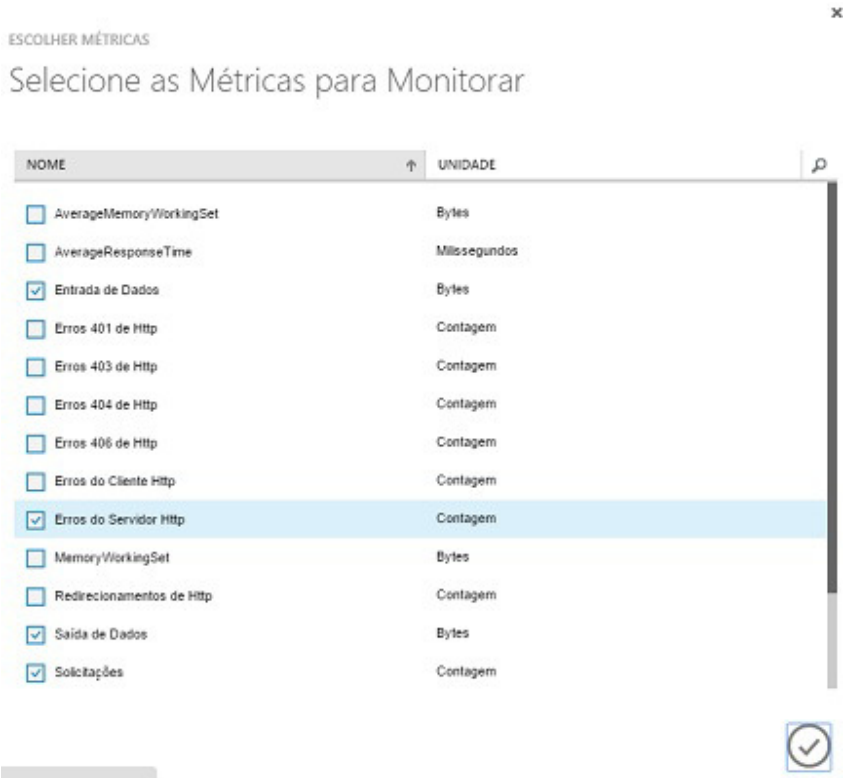

Figura 4.18: Adicionando ou removendo métricas ao painel

Você também pode configurar para ser notificado via e-mail quando determinado evento ocorrer. Para isso, basta criar uma regra de alerta:

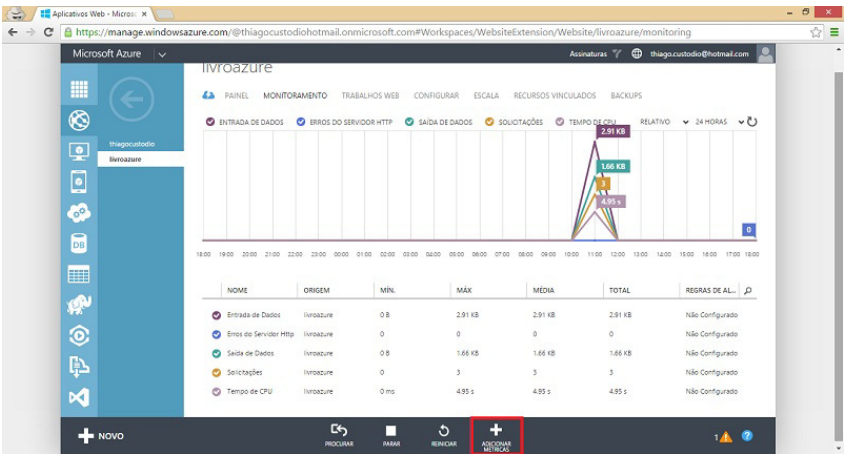

Figura 4.19: Adicionando uma nova regra de alerta

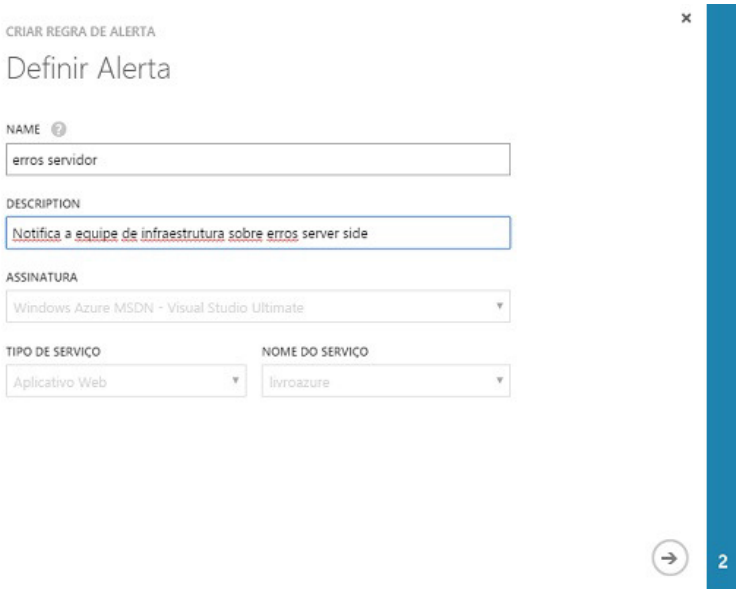

Figura 4.20: Definindo o nome da regra de alerta

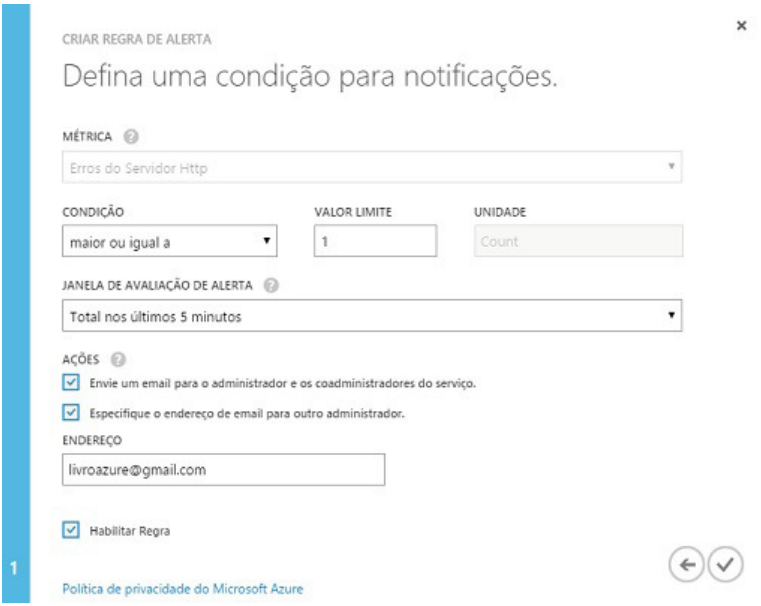

Figura 4.21: Definindo as condições para a notificação

4.9 WEBAPPS DA GALERIA

A galeria disponibiliza uma vasta gama de aplicações web populares desenvolvidos pela Microsoft, empresas terceirizadas, e iniciativas de software de código aberto. Dentre essas aplicações, existem gerenciadores de conteúdo, e-commerces, blogs, fóruns, wikis etc. Para usufruir, basta criar um novo aplicativo, mas, desta vez, selecionando a opção `Da Galeria` .

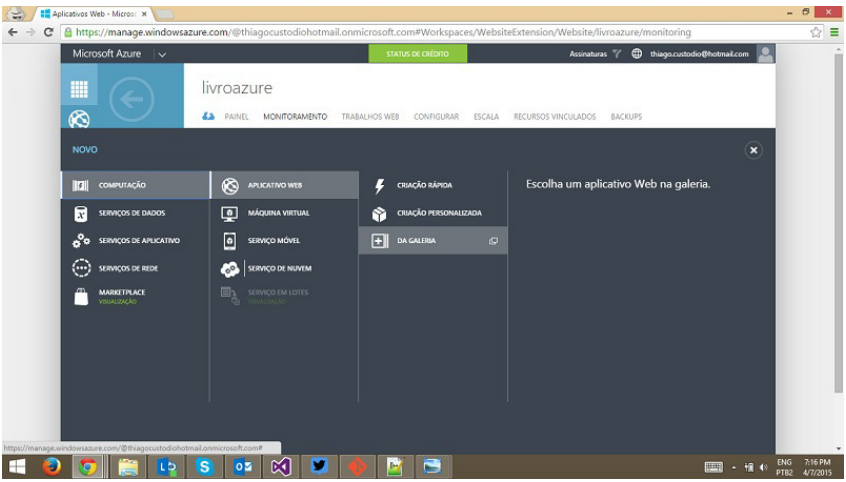

Figura 4.22: Criando um WebApp da galeria

Na tela seguinte, basta selecionar a opção desejada e seguir o assistente para a criação.

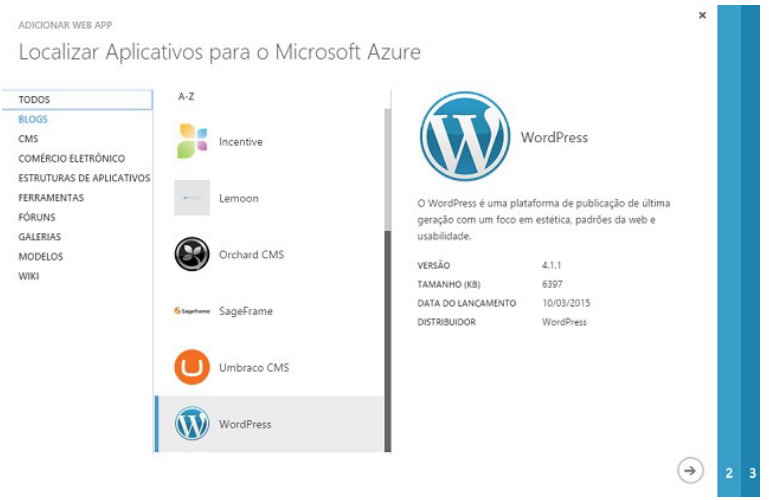

Figura 4.23: Escolha do tipo de WebApp

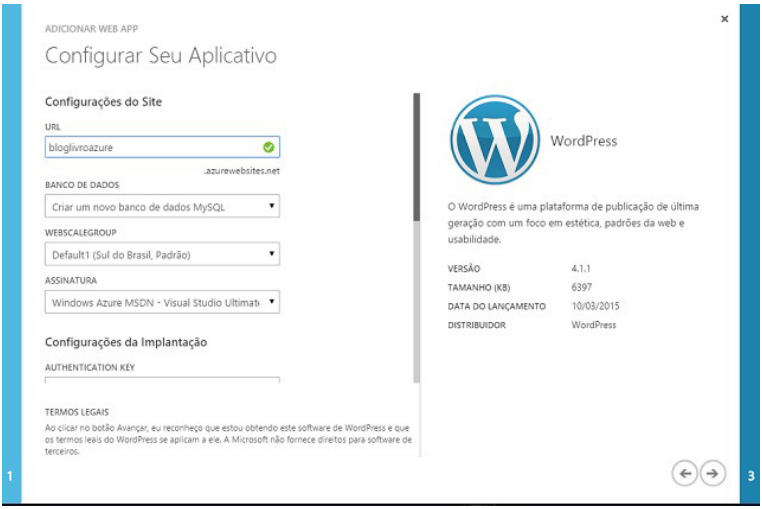

Figura 4.24: Definindo o nome do WebApp

Recursos adicionais

Um blog utilizando WordPress necessita de um banco de dados MySQL para persistir as informações. No Azure, o MySQL é mantido por uma empresa chamada ClearDB. Consulte os planos e veja se atenderá a sua necessidade. Caso contrário, instale um MySQL em uma máquina virtual e configure a string de conexão do WordPress para esse novo banco.

Vale lembrar de que você pode criar até 10 WebApps no modo gratuito. Fique atento apenas às limitações de processamento/dados de saída (quotas), e se o aplicativo utiliza recursos adicionais (MySQL, por exemplo).

4.10 SUPORTE A SSL E CONFIGURAÇÃO DE

DOMÍNIOS

- Suporte a SSL são oferecidos no modo de hospedagem *básico* ou *standard*. Caso você precise desse recurso, confira no próprio site do portal um minitutorial ensinando passo a passo (http://bit.ly/ConfigurarSSLAzure).
- Suporte a domínios customizados são oferecidos no modo de hospedagem *compartilhado*, *básico* ou *standard*. Caso você precise desse recurso, existe no próprio site do portal um minitutorial (http://bit.ly/DominioCustomizadoAzure).

4.11 CONCLUINDO

Azure WebApps são ideais para aplicativos ou web sites simples, ou seja, que não dependem de componentes de terceiros que precisam ser registrados ou instalados. Programadores de .NET, PHP, Node.js, Java (veja o box a seguir) e Python podem usufruir desse modelo de execução e rapidamente publicar WebApps que podem ser escalados com apenas alguns cliques.

WEBAPP EM JAVA

Recentemente, a Microsoft possibilitou a publicação de aplicações web feitas em Java. Antes de publicar seu WebApp, você pode escolher entre dois servidores de aplicação: Jetty ou Apache Tomcat.

Para maiores informações sobre a utilização de Azure Web Sites com Java, visite http://bit.ly/AzureWebSiteComJava.

PRECISO REGISTRAR UMA DLL NO GLOBAL ASSEMBLY CACHE, E AGORA?

Se você precisa utilizar algum componente de terceiro na sua aplicação, recomendo que você use outros modelos de execução como Web Roles (*Cloud Services*) ou Máquinas Virtuais, justamente pelo fato de oferecerem um maior controle para o desenvolvedor.

Capítulo 5

ROTINAS EM BACKGROUND COM AZURE WEBJOBS

Se você já estudou um pouco sobre sistemas operacionais, já deve ter visto sobre processos que são executados em segundo plano (*background*) e os que são executados em primeiro plano (*foreground*). Basicamente, a diferença entre os dois é que os processos executados em primeiro plano exigem interação direta do usuário.

Quando se tem acesso ao servidor, é simples criar um programa que será executado em segundo plano. Entretanto, no caso do Azure, onde não temos acesso direto aos servidores que se encontram nos data centers da Microsoft, o que devemos fazer, quando surge a necessidade de processamento em background?

Um dos possíveis meios para solucionar este problema é a utilização de Azure WebJobs, que são arquivos em script ou executáveis que dão a habilidade de processamento sem a necessidade de uma tela para interação. Os WebJobs são hospedados e executados sob o mesmo contexto dos WebApps, sendo assim, é possível acessar as mesmas configurações e sistema de arquivos. Sem contar que não há custo adicional, dado que eles compartilham os mesmos recursos computacionais (CPU, memória RAM e disco). Você pode configurar para que o seu WebJob execute

de algumas maneiras:

- **Modo agendado:**

  Neste modo, você configura quando e com que frequência o serviço será executado, o Azure Scheduler se encarrega de executá-lo na data especificada.

- **Modo contínuo:**

  Aqui, o processamento é contínuo. Imagine um trecho de código dentro de uma estrutura de repetição:

```
while(true)
{
    //processar
}
```

- **Modo sob demanda:**

  Neste modo, você cria um WebJob que será acionado manualmente, isto é, após a criação, você inicia o processamento diretamente via portal. Ele é ideal para demandas que não possuem uma recorrência definida. Sempre que necessário, basta disparar sua execução.

5.1 CENÁRIOS DE UTILIZAÇÃO

Alguns cenários nos quais o Azure WebJobs pode ser usado:

- Envio de e-mails;
- Consumo de mensagens em uma fila;
- Processamento de imagens (conversão de formatos e geração de *thumbnails*);
- Tarefas rotineiras como expurgo de logs, informação muito antiga armazenada em algum serviço do Azure ou até mesmo logs do IIS.

O bacana é que não necessariamente eles precisam ser construídos com .NET. Os seguintes tipos também são aceitos, basta efetuar o upload do arquivo no portal do Azure:

- `.cmd / .bat`
- `.sh` (feitos em bash)
- `.php` (construídos com a linguagem PHP)
- `.py` (contruídos com Python)
- `.js`

5.2 CRIANDO UM WEBJOB COM VISUAL STUDIO

Para exemplificar este recurso do Azure, vamos um criar um novo projeto do tipo `Console Application`. Esse projeto fará uma requisição HTTP a um feed RSS de um blog, e enviará um e-mail contendo o post mais recente.

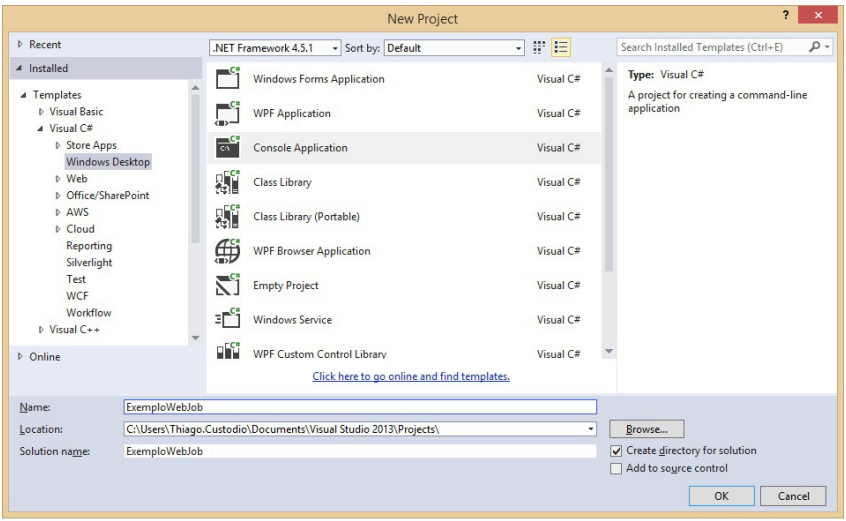

Figura 5.1: Novo projeto no Visual Studio

Com o projeto criado, precisamos adicionar uma referência a

System.Net. Para isto, basta clicar com o botão da direita sobre a opção `References` e, em seguida, `Add Reference`.

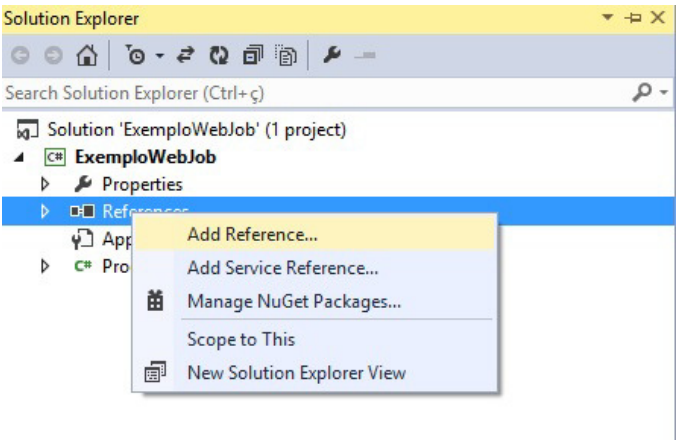

Figura 5.2: Adicionando uma nova referência ao projeto

Sob o menu à esquerda `Assemblies`, procure por `System.Net`. Selecione o checkbox, e clique em `OK`.

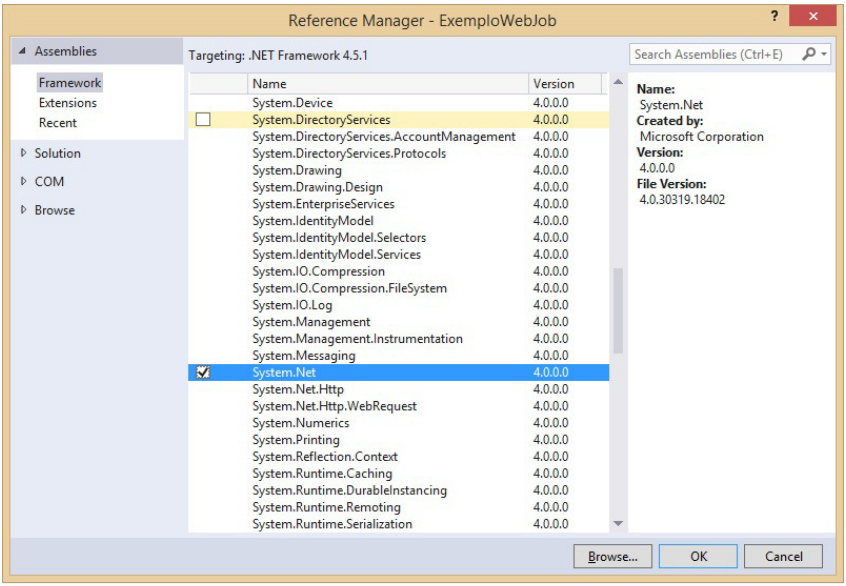

Figura 5.3: Lista de assemblies disponíveis

Agora podemos fazer uso das classes que estão sob o namespace `System.Net`. No arquivo `program.cs`, adicione os seguintes `using`s:

```
using System.Net;
using System.Net.Mail;
```

Em seguida, vamos adicionar uma classe chamada `Post` para facilitar o entendimento. Esta possui apenas três propriedades: título, data de publicação e conteúdo:

```
public class Post
{
    public string Titulo { get; set; }

    public DateTime DataPublicacao { get; set; }

    public string Conteudo { get; set; }
}
```

O processamento está dividido em três etapas:

1. Envio da requisição HTTP;
2. Parse do XML do feed;
3. Envio do e-mail.

Para a primeira etapa, vamos adicionar o método que faz o envio da requisição HTTP ao feed RSS. Vou usar meu próprio blog como exemplo, mas sinta-se à vontade para utilizar outro blog como exemplo. Este código apenas dispara a requisição e retorna uma string com o conteúdo da resposta. Para retornar o conteúdo da resposta como string, precisamos importar mais um namespace:

```
using System.IO;
```

Após essa importação, podemos utilizar a classe `StreamReader` para obter uma string com o retorno da requisição HTTP.

```
private static string RequestRssFeed()
{
    var xmlRss = string.Empty;
    var url = "http://thiagocustodio.azurewebsites.net/feed/";
    var request = WebRequest.Create(url);
    request.Method = "GET";

    var response = request.GetResponse();

    using (var stream = response.GetResponseStream())
    {
        using (var sr = new StreamReader(stream))
        {
            xmlRss = sr.ReadToEnd();
        }
    }
    return xmlRss;
}
```

Para efetuar o parse do `XML`, vamos adicionar um novo `using`, mas desta ver ao namespace `using System.Xml.Linq`

```
using System.Xml.Linq;
```

O método que faz o parse do `XML` recebe a string com o conteúdo da resposta da requisição como parâmetro. Em seguida, é

feito o parse do `XML` e a navegação até o elemento `item`, que contém os posts. Por último, instanciamos a classe `Post`, setando as propriedades com os valores lidos do `XML`:

```
private static Post ParseXml(string xmlRss)
{
    var xml = XElement.Parse(xmlRss);
    var ultimoPost = xml.Elements().First().Element("item");

    var post = new Post
    {
        Titulo = ultimoPost.Element("title").Value,
        DataPublicacao = DateTime.Parse(
                            ultimoPost.Element("pubDate").Value
                        ),
        Conteudo = ultimoPost.Element("description").Value
    };
    return post;
}
```

Em seguida, disparamos um e-mail enviando as informações coletadas em relação à última publicação no blog:

```
private static void EnviarEmail(Post post)
{
    var titulo =
            string.Concat(post.DataPublicacao, ":", post.Titulo);
    var email = new MailMessage("livroazure@gmail.com",
                                "emaildestino@gmail.com",
                                titulo,
                                post.Conteudo);

    using (SmtpClient smtp =
                        new SmtpClient("smtp.gmail.com", 587))
    {
        smtp.Credentials =
                new NetworkCredential("livroazure@gmail.com",
                                      livrocasacodigo");

        smtp.EnableSsl = true;
        smtp.Send(email);
    }
}
```

Observação

A caixa de e-mail `emaildestino@gmail.com` foi usada apenas como exemplo. Altere para a sua caixa de e-mail a fim de validar o funcionamento do WebJob.

Agora basta adicionar as chamadas aos métodos que acabamos de criar ao método `Main`, e compilar o projeto (`Ctrl + Shift + B`, ou `Menu Build -> Build Solution`).

Eis o código completo do WebJob:

```
using System;
using System.Net;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using System.Net.Mail;
using System.IO;

namespace ExemploWebJob
{
    class Post
    {
        public string Titulo { get; set; }

        public DateTime DataPublicacao { get; set; }

        public string Conteudo { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var xmlRss = RequestRssFeed();

            var post = ParseXml(xmlRss);
```

```
    EnviarEmail(post);
}

private static string RequestRssFeed()
{
    var xmlRss = string.Empty;
    var url =
          http://thiagocustodio.azurewebsites.net/feed/";
    var request = WebRequest.Create(url);
    request.Method = "GET";

    var response = request.GetResponse();

    using (var stream = response.GetResponseStream())
    {
        using (var sr = new StreamReader(stream))
        {
            xmlRss = sr.ReadToEnd();
        }
    }
    return xmlRss;
}

private static Post ParseXml(string xmlRss)
{
    var xml = XElement.Parse(xmlRss);
    var ultimoPost =
            xml.Elements().First().Element("item");

    var post = new Post
    {
        Titulo = ultimoPost.Element("title").Value,
        DataPublicacao =
            DateTime
            .Parse(ultimoPost.Element("pubDate")
            .Value),
        Conteudo =
            ultimoPost.Element("description").Value
    };
    return post;
}

private static void EnviarEmail(Post post)
{
    var titulo = string.Concat(
                                post.DataPublicacao,
                                ":",
                                post.Titulo
```

```
                );

            var email = new MailMessage("livroazure@gmail.com",
                                        "emaildestino@gmail.com",
                                        titulo,
                                        post.Conteudo);

            using (SmtpClient smtp = new SmtpClient
                    (
                        "smtp.gmail.com",
                        587
                    )
                )
            {
                smtp.Credentials =
                    new NetworkCredential
                    (
                        "livroazure@gmail.com",
                        "livrocasacodigo"
                    );
                smtp.EnableSsl = true;
                smtp.Send(email);
            }
        }
    }
}
```

Com o código concluído, precisamos apenas fazer o upload dos arquivos de saída do nosso projeto no portal de gerenciamento do Azure, e configurar a frequência com que este WebJob vai executar. Você pode fazer isso de um muito jeito simples, clicando com o botão da direita sob o nome do projeto e, em seguida, selecionando a opção `Open Folder in File Explorer`:

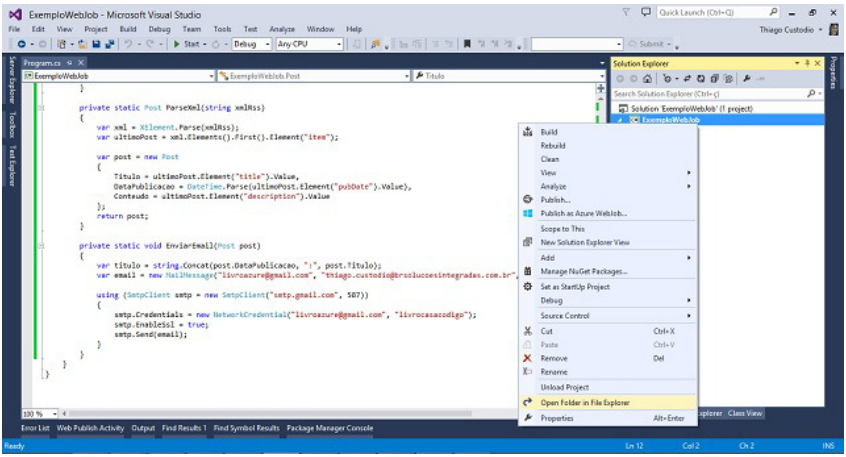

Figura 5.4: Abrindo o diretório da solução

Dê um duplo clique sob a pasta `bin` e, depois, clique com o botão da direita sob a pasta `Debug`. Selecione a opção de menu `Enviar Para` e, em seguida, na `Pasta Compactada`, um arquivo `Debug.zip` deverá ser criado em alguns poucos segundos.

> **Modo de compilação**
>
> O modo de compilação debug foi usado apenas como fins educativos. O ideal seria usar o modo *release*, pois o compilador gera uma versão otimizada.

No portal de gerenciamento do Azure, localize o seu WebApp e, depois, clique sob o nome:

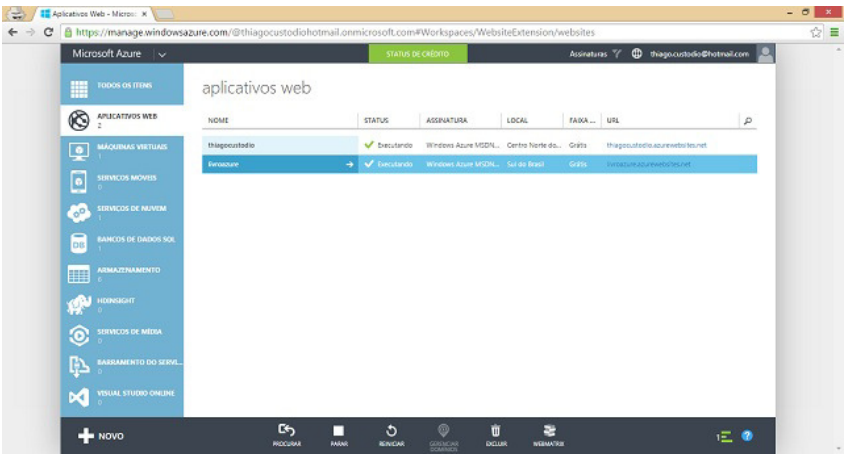

Figura 5.5: Selecionando o WebApp

Selecione o menu `Trabalhos Web` :

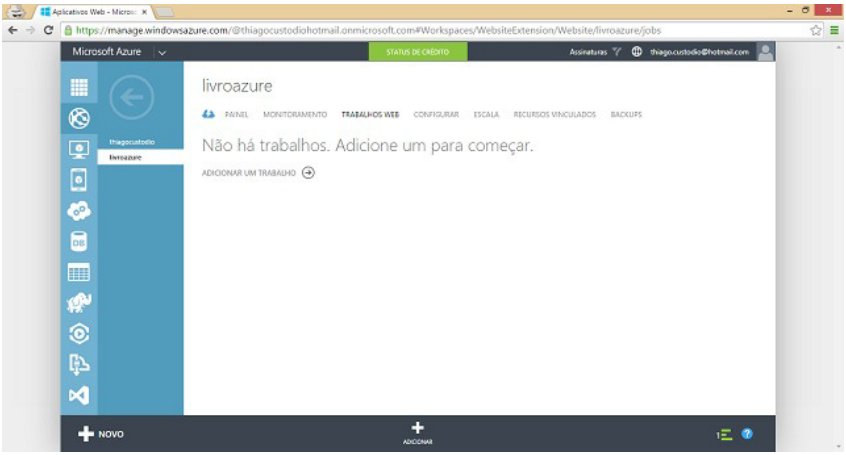

Figura 5.6: Selecionando o menu Trabalhos Web (Web Job)

Logo após, clique em `Adicionar um Trabalho` . Informe o nome, o diretório onde se encontra o arquivo `zip` que criamos anteriormente e o modo que o WebJob será executado (agendado, contínuo ou sob demanda). Para esse exemplo, vou usar o modo agendado:

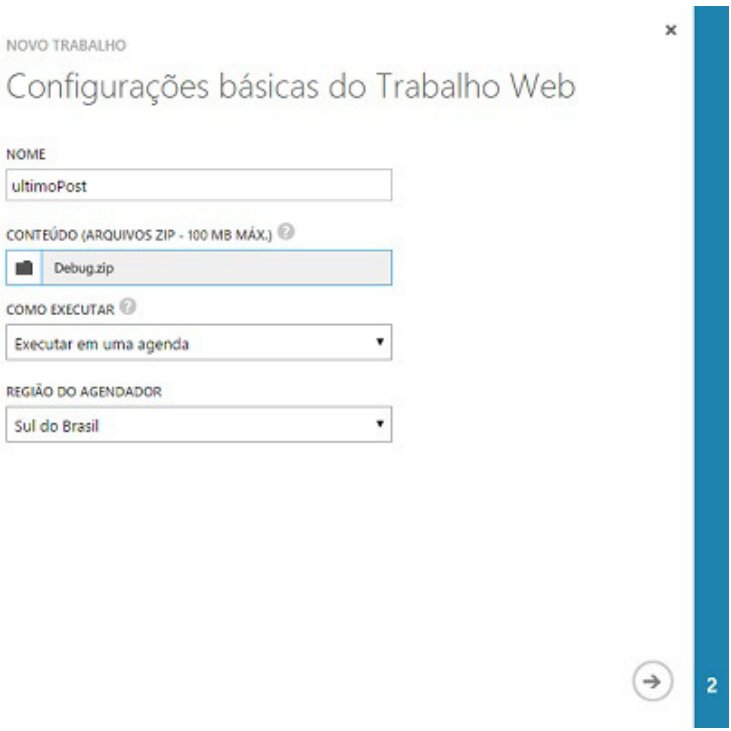

Figura 5.7: Seleção do arquivo zip criado anteriormente

Selecione a seta para avançar para o passo seguinte, onde vamos configurar se o serviço vai ser executado apenas uma vez, ou de maneira recorrente:

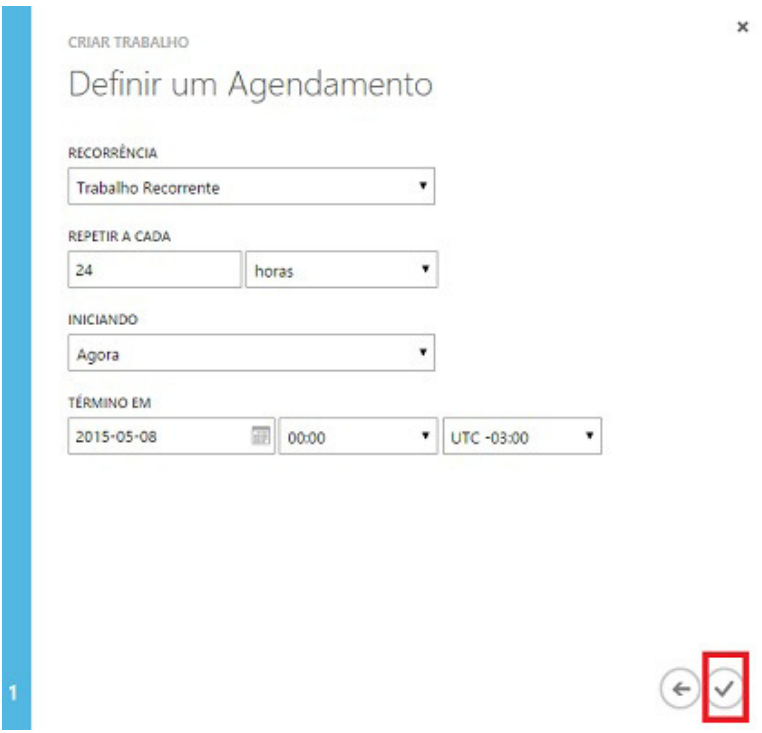

Figura 5.8: Configuração da recorrência do WebJob

Após alguns segundos, o serviço será executado e você receberá um e-mail com o conteúdo do último post:

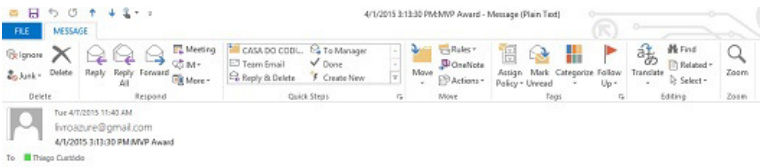

Figura 5.9: Recebimento do e-mail

5.3 CONCLUINDO

Neste capítulo, aprendemos como utilizar os Azure WebJobs para executar processamento em background no modelo Plataforma como Serviço, bem como alguns exemplos de cenários para a sua utilização. A seguir vamos conhecer o Azure Redis Cache.

Capítulo 6

CACHE COMO SERVIÇO USANDO AZURE REDIS CACHE

Já tive a oportunidade de trabalhar em um dos dez maiores portais web do Brasil. Posso assegurá-lo de que, se não existissem soluções de cache, seria muito difícil manter o site no ar e, muito provavelmente, o site não seria um dos dez maiores portais do Brasil pelo simples fato de que usuários odeiam esperar pelo carregamento das páginas.

Neste capítulo, vamos aprender como usufruir do Azure Redis Cache para otimização de páginas e armazenamento de objetos em memória, evitando consultas repetidas no banco de dados.

6.1 INTRODUÇÃO AO REDIS

Redis é um *NoSQL* baseado em chave/valor que armazena informações em memória, mas também permite que estas informações sejam persistidas em disco. Algumas pessoas consideram o Redis como uma ferramenta de *cache++*, pois, além de possibilitar armazenamento no formato chave/valor, ele também suporta estrutura de dados de maneira nativa.

Você pode trabalhar com listas (*Lists*), listas ordenadas (*Ordered Lists*), conjuntos (*Sets*), filas (*Queue*), publicação/assinatura

(*Publish/Subscribe*) e transações. Se necessário, você pode combinar todos esses recursos, por exemplo: quando alguém alterar um valor em uma lista, todos os clientes interessados receberão notificação em relação a essa alteração.

Esses e outros cenários já foram abordados no excelente livro do Rodrigo Lazoti *Armazenando dados com Redis*, também publicado pela editora Casa do Código. Para maiores informações a respeito desse livro, acesse http://www.casadocodigo.com.br/products/livro-redis.

6.2 UTILIZANDO O AZURE REDIS CACHE

O Azure Redis Cache é uma plataforma de Cache como Serviço (*Cache as a Service*), mantido e gerenciado pela Microsoft nos data centers do Azure. Você só tem o trabalho de plugar sua aplicação ao Azure Redis Cache, e utilizá-lo. Todo o provisionamento, gerenciamento da infraestrutura, disponibilidade e escalabilidade ficam por conta da Microsoft.

Provisionando o Azure Redis Cache

Esse serviço está disponível apenas na nova versão do portal (Ibiza). Efetue o login por meio da URL: http://portal.azure.com.

Em seguida, selecione a opção `Novo -> Dados + Armazenamento -> Redis Cache`.

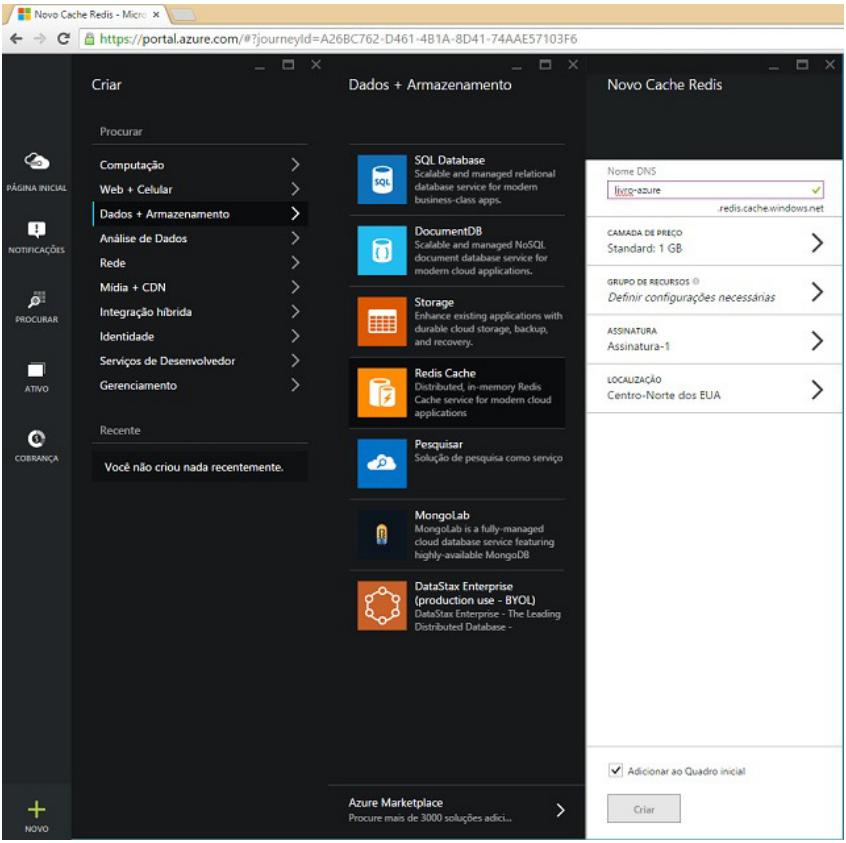

Figura 6.1: Criação do redis Cache no portal Ibiza

Precisamos selecionar um `Grupo de Recursos` ou criar um novo. Um grupo de recursos nada mais é do que um agrupador.

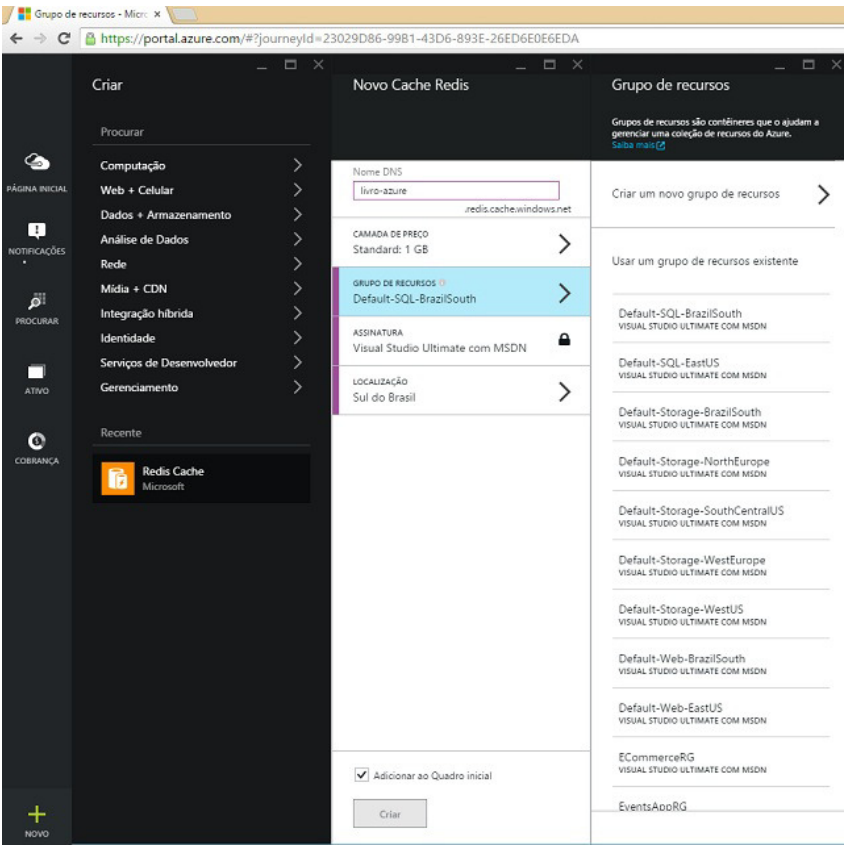

Figura 6.2: Seleção do grupo de recursos

O intuito ao usar o Redis é melhorar a performance da aplicação. Sendo assim, faz sentido manter o serviço de cache próximo à aplicação para evitarmos problemas de latência:

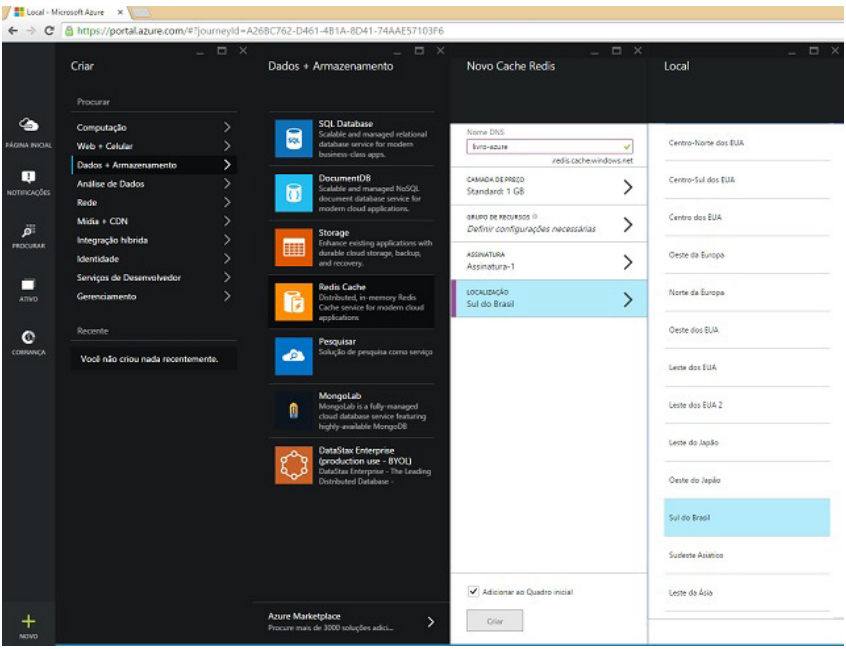

Figura 6.3: Seleção do data center

Por último, basta clicar no botão `Criar` e aguardar o término do provisionamento.

Persistindo cache de páginas no Azure Redis Cache

Armazenar cache de páginas no Azure Redis Cache é uma tarefa extremamente simples e que vai causar um enorme impacto na performance do seu website. Precisamos apenas adicionar um *Nuget package* ao projeto Web, e configurar o `web.config` com as chaves de acesso e URL do serviço que criamos no passo anterior.

Aproveitando o projeto web criado no capítulo sobre WebApp, clique com o botão da direita sob o nome do projeto e, em seguida, selecione a opção `Manage Nuget Packages` :

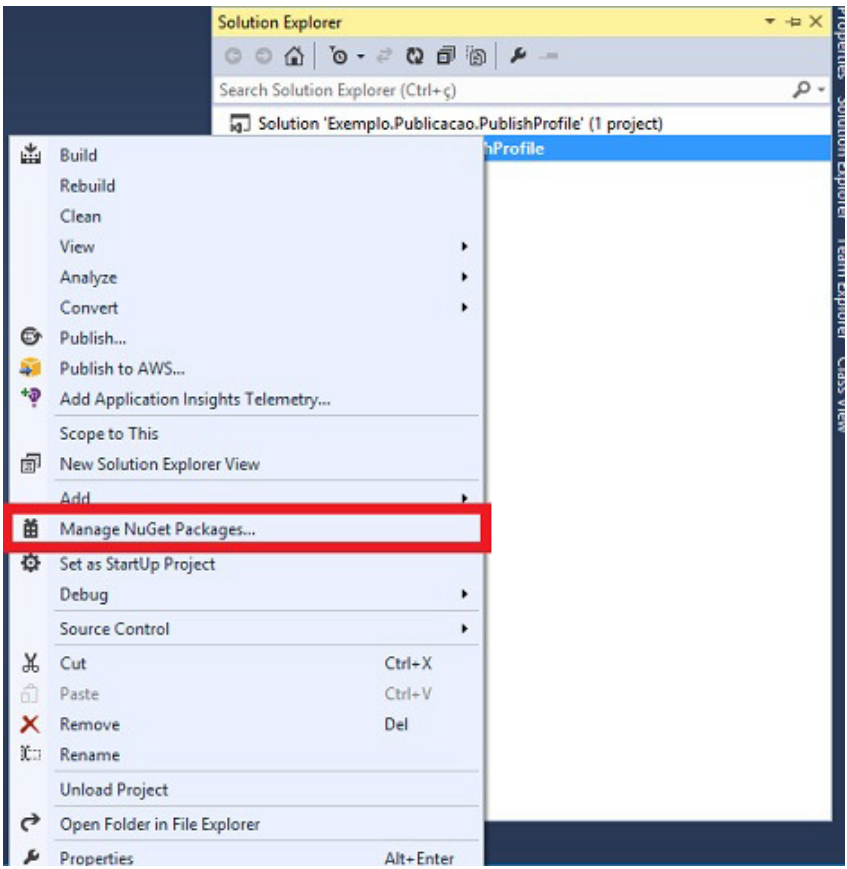

Figura 6.4: Seleção da opção Manage Nuget Packages

Depois, procure na seção `Online` pelo pacote `RedisOutputCacheProvider` e, em seguida, em `Install`:

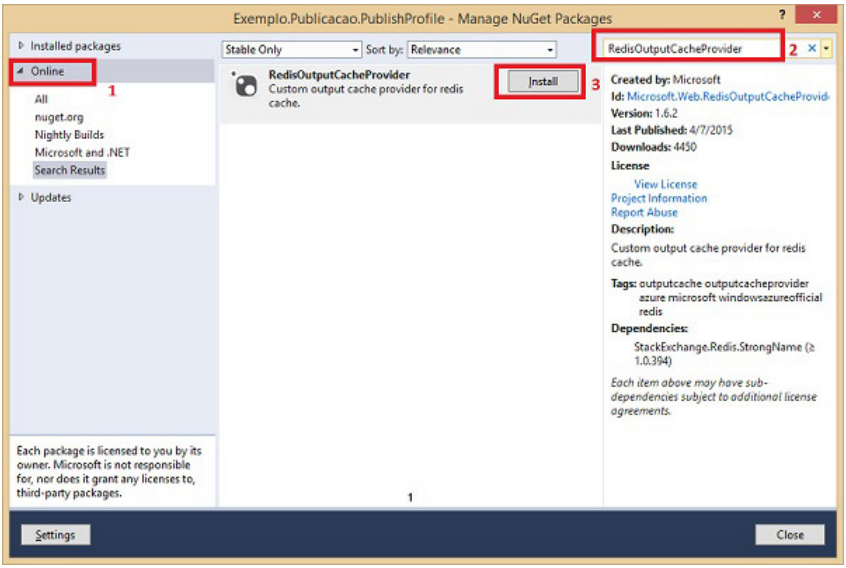

Figura 6.5: Pesquisa pelo nuget RedisOutputCacheProvider

Após aceitar os termos de uso, o pacote será adicionado ao nosso projeto, e uma nova seção (`caching`) será adicionada ao arquivo `Web.config` . Precisamos apenas informar os atributos `host` e `accessKey` . Essas informações podem ser obtidas diretamente no portal de administração do Azure (Ibiza), basta clicar sobre o ícone `Keys` :

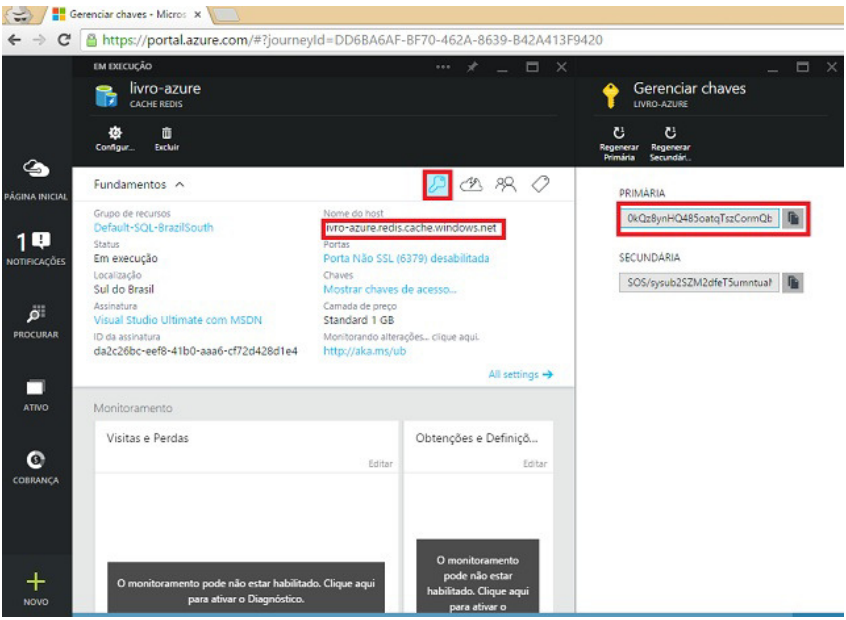

Figura 6.6: Obtendo as chaves de acesso

Para concluir a configuração, basta adicionar os valores coletados aos atributos no `Web.config` :

```
<configuration>
    <!-- manter as demais configurações já existentes -->
    <system.web>
    <authentication mode="None" />
    <compilation debug="true" targetFramework="4.5.1" />
    <httpRuntime targetFramework="4.5.1" />
    <caching>
      <outputCache defaultProvider="MyRedisOutputCache">
        <providers>
          <add name="MyRedisOutputCache"
            type="Microsoft.Web.Redis.RedisOutputCacheProvider"
            host="livro-azure.redis.cache.windows.net"
            accessKey=
                "0kQz8ynHQ485oatqTszCormQbUANo48Rbb+ar88kbNg="
            ssl="true" />
        </providers>
      </outputCache>
    </caching>
  </system.web>
</configuration>
```

Para demonstrar o funcionamento, vamos adicionar o atributo `OutputCache` à `Action Index` do arquivo `HomeController.cs`, localizado na pasta `Controllers` da solução. Em seguida, vamos setar a duração do cache para 2 minutos e rodar a aplicação.

```
public class HomeController : Controller
{
    [OutputCache(Duration=120)]
    public ActionResult Index()
    {
        return View();
    }
}
```

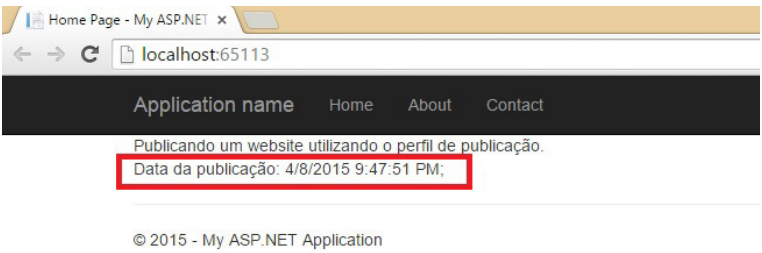

Figura 6.7: Exibição da data/hora

Repare que, ao atualizar a página, a data/hora não é alterada, pois essa versão está armazenada no Redis Cache por 120 segundos. Após este prazo, uma nova versão da página é armazenada no Cache, e servirá as próximas requisições pelos próximos 120 segundos.

Esse simples recurso vai consumir menos processamento do servidor e, consequentemente, permitirá que mais requisições sejam atendidas, além de um menor tempo de espera pelo carregamento da página.

Armazenando objetos no Azure Redis Cache

Para armazenar objetos no Azure Redis Cache, vamos utilizar um outro Nuget package chamado `StackExchange.Redis` (desenvolvido e utilizado no Fórum StackOverflow):

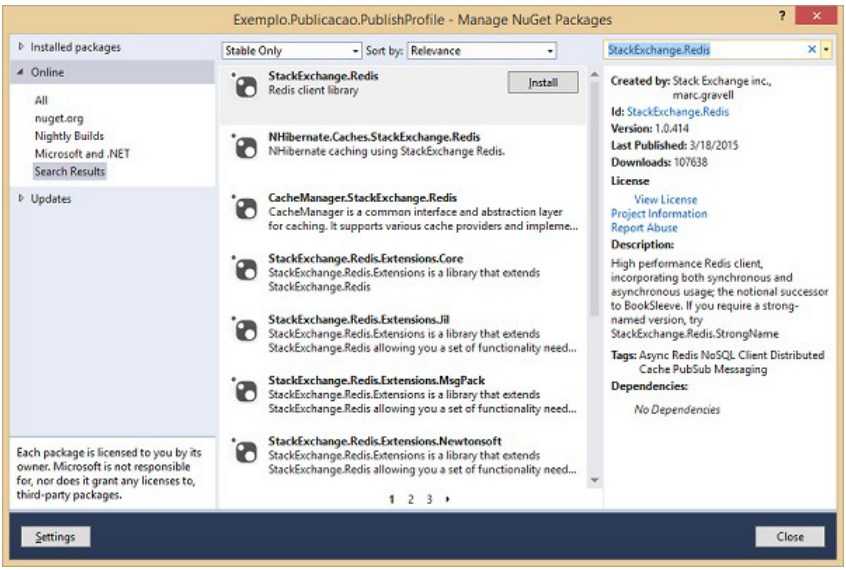

Figura 6.8: Adicionando o pacote StackExchange.Redis

> **Clientes para o Redis**
>
> Existem diversos clientes para Redis e para diversas linguagens. Confira a lista completa diretamente em http://redis.io/clients.

Neste exemplo, vamos imaginar que exista algum cálculo intensivo na nossa aplicação, e o resultado desse cálculo deverá ser apresentado ao usuário final. Apenas para exemplificar, alterei novamente a `View Index` do arquivo `HomeController.cs`, incluindo um laço utilizando a estrutura de repetição `for`. Dentro do laço, adicionei um `Thread.Sleep` de dois segundos:

```
public ActionResult Index()
{
    ViewBag.Inicio = DateTime.Now;

    var soma = 0;
    for (int i = 0; i < 10; i++ )
    {
        soma += i;
        System.Threading.Thread.Sleep(2000);
    }
    ViewBag.Soma = soma;
    ViewBag.Fim = DateTime.Now;

    return View();
}
```

Em seguida, alterei a `View Index` para exibir estas informações:

```
@{
    ViewBag.Title = "Home Page";
}
<div>
    Publicando um website utilizando o perfil de publicação.
</div>
<div>
    Data da publicação: @DateTime.UtcNow.AddHours(-3);
</div>
<br />
<br />
<div>
    <div>
        Processamento Iniciado em: @ViewBag.Inicio
    </div>
    <div>
        Concluído em: @ViewBag.Fim
    </div>
</div>
```

Ao executar a aplicação, repare na quantidade de segundos entre o início e o fim (20 segundos) desse cálculo:

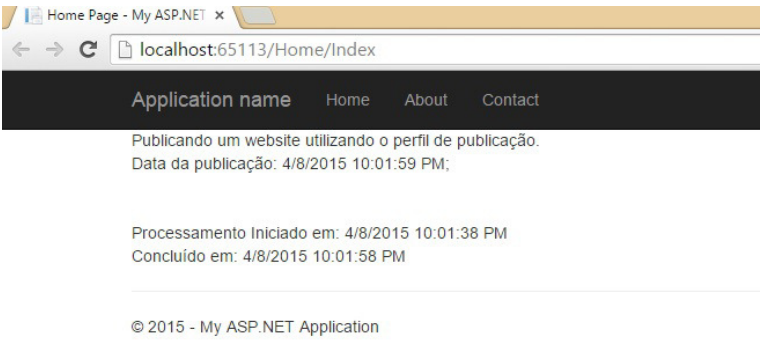

Figura 6.9: Exibição da data/hora sem a utilização do cache

Embora esse exemplo seja meio absurdo, existem diversos cenários no nosso dia a dia em que podemos poupar o servidor de executar sempre as mesmas tarefas desnecessariamente. Por exemplo, não há necessidade de ir até o banco de dados para obter informações que sofrem pouca ou nenhuma alteração a cada requisição. Podemos otimizar o processo, utilizando cache de objetos. Para isso, basta usarmos as classes disponíveis no pacote instalado anteriormente, e adicionar o item ao cache (caso não exista):

Primeiro, vamos importar o namespace `StackExchange.Redis`:

```
using StackExchange.Redis;
```

Em seguida, vamos estabelecer uma conexão com o Azure Redis Cache. Para isso, basta chamar o método `Connect` da classe `ConnectionMultiplexer`, passando a string de conexão com o cache:

```
var sEndpoint = "livro-azure.redis.cache.windows.net";
var sHabilitarSSL = "ssl=true";
//mesmo valor do accessKey no web.config
var senha = "0kQz8ynHQ485oatqTszCormQbUANo48Rbb+ar88kbNg=";
```

```
var connString = string.Concat
                (
                    sEndpoint,
                    "," ,
                    sHabilitarSSL ,
                    "," ,
                    senha
                );
var connection = ConnectionMultiplexer.Connect(connString);
```

Em seguida, chamamos o método `GetDatabase` para obter uma instância do objeto `DataBase` , e manipular o cache:

```
var db = connection.GetDatabase();
```

Então usamos os métodos `StringGet` para recuperar valores, e `StringSet` para armazenar valores no Redis Cache. Caso você queira armazenar um objeto, basta serializá-lo como JSON. O código da `Action` ficou assim:

```
public ActionResult Index()
{
    ViewBag.Inicio = DateTime.Now;

    var sEndpoint = "livro-azure.redis.cache.windows.net";
    var sHabilitarSSL = "ssl=true";
    var senha = "0kQz8ynHQ485oatqTszCormQbUANo48Rbb+ar88kbNg=";

    var connString = string.Concat
            (
                sEndpoint,
                ",",
                sHabilitarSSL,
                ",",
                senha
            );
    var connection = ConnectionMultiplexer.Connect(connString);

    var db = connection.GetDatabase();

    var resultado = db.StringGet("resultadoCalculo");
    var soma = "";

    if(string.IsNullOrEmpty(resultado))
    {
        for (int i = 0; i < 10; i++)
```

```
        {
            soma += i;
            System.Threading.Thread.Sleep(2000);
        }

        db.StringSet("resultadoCalculo", soma);
    }
    else
    {
        soma = resultado;
    }

    connection.Close();

    ViewBag.Soma = soma;
    ViewBag.Fim = DateTime.Now;

    return View();
}
```

Repare como o tempo de resposta final sofreu uma redução bem significativa:

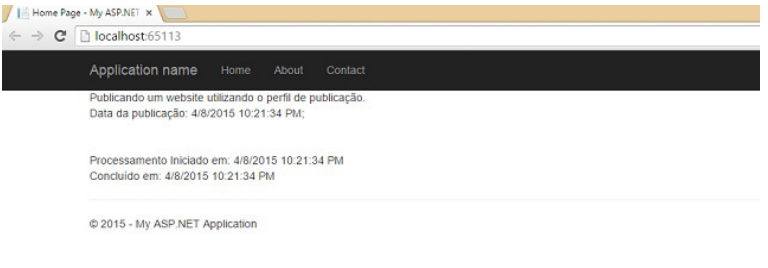

Figura 6.10: Exibição da data/hora com a utilização do cache

Em cenários de extrema concorrência, economizar alguns segundos, ou até mesmo milissegundos, faz **toda** a diferença.

6.3 CONCLUINDO

O Azure Redis Cache é um serviço integrado do Azure que disponibiliza um cache como serviço. Existem diversos cenários em

que o Redis pode melhorar a performance da sua aplicação. Utilizando como serviço no Azure, as complicações de provisionamento, gerenciamento da infraestrutura, disponibilidade e escalabilidade são delegadas para a Microsoft.

Capítulo 7

CONHECENDO O AZURE TABLES

Azure Tables é um outro tipo de NoSQL baseado em chave/valor disponível no Azure, e que faz parte do serviço de armazenamento Azure Storage. Apesar de o nome sugerir uma relação com bancos de dados relacionais, não há qualquer relação entre estes dois mecanismos de armazenamento. Ou seja, ele não oferece as mesmas funções disponíveis em um SQL Server (por exemplo), como relacionamento com outras tabelas (*joins*) ou *stored procedures*.

O Azure Tables é indicado para aplicativos que precisam armazenar grandes quantidades de dados não relacionais. A estrutura do serviço de armazenamento do Azure é retratado da seguinte maneira:

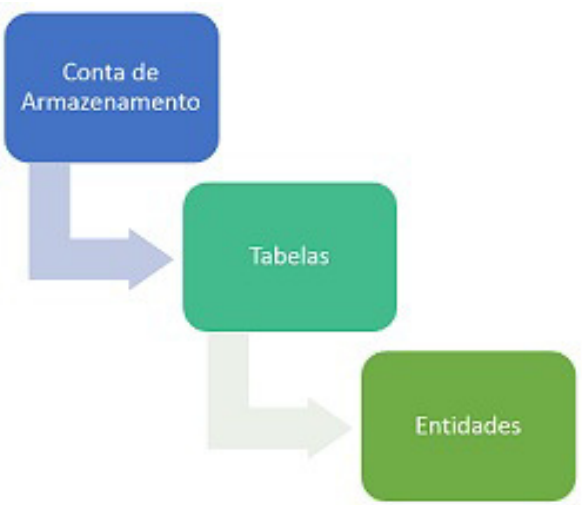

Figura 7.1: Estrutura do serviço de armazenamento

7.1 CRIANDO UMA CONTA DE ARMAZENAMENTO

Para usar o Azure Tables, precisamos criar uma conta de armazenamento no portal de gerenciamento do Azure. A função da conta de armazenamento é atuar como um contêiner para os serviços disponíveis no Azure Storage (*Queues*, *Tables*, *Blobs*).

Acesse o portal de gerenciamento do Azure, e selecione a opção: `NOVO -> SERVIÇO DE DADOS -> ARMAZENAMENTO -> CRIAÇÃO RÁPIDA .`

Figura 7.2: Criação de uma nova conta de armazenamento

Informe o nome da conta de armazenamento, o local/grupo de afinidade e a assinatura, e selecione a opção `CRIAR CONTA DE ARMAZENAMENTO`.

Figura 7.3: Informando o nome da conta de armazenamento

Você pode criar até 20 contas de armazenamento por assinatura. Este é o valor padrão, mas é possível aumentá-lo para 50, entrando em contato diretamente com o suporte do Azure. Em cada conta de armazenamento, é possível armazenar até 200TB de informações. Não há limitações em relação ao número de objetos por conta de armazenamento, desde que não ultrapassem os 200TB disponíveis.

Após o provisionamento dessa conta de armazenamento, precisamos acessar as chaves de acesso para criar os objetos via programação. Para isso, basta selecionar no menu da esquerda o ícone referente a `ARMAZENAMENTO` e, em seguida, clicar sob o nome da conta de armazenamento criada no passo anterior:

Figura 7.4: Menu de acesso rápido às contas de armazenamento

Na tela seguinte, clique sobre o ícone `GERENCIAR CHAVES DE ACESSO` para exibi-las. Copie e cole para um bloco de notas essas chaves.

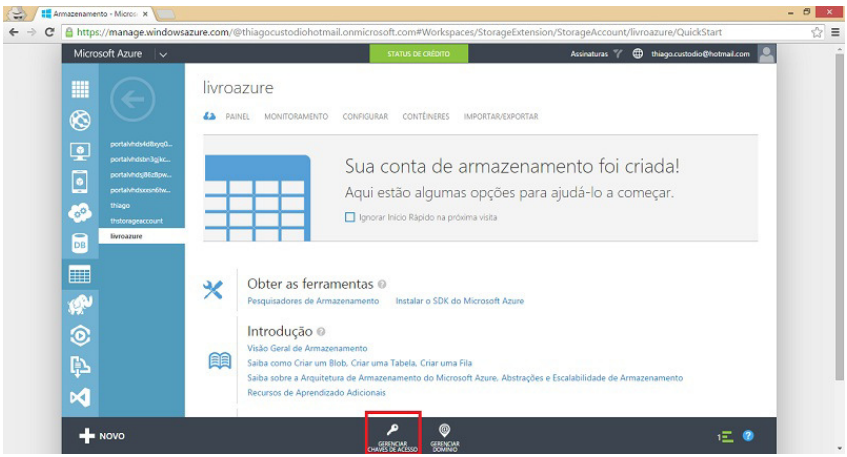

Figura 7.5: Exibir chaves de acesso

Figura 7.6: Exibição do nome da conta, chave primária e secundária

7.2 ARMAZENANDO DADOS EM AZURE TABLES

Composição da Chave

Estruturas de dados baseadas em chave/valor possuem um identificador único (chave) para recuperar e armazenar determinado valor. No Azure Tables, esse identificador único é composto por duas chaves: *Partition Key* e *Row Key* (definidas a seguir). A combinação dessas chaves compõe um índice clusterizado (chave primária).

Observação

O tamanho dessas chaves não pode ultrapassar 1KB.

Composição do valor

O Azure Tables armazena tabelas, que, por sua vez, são coleções de entidades. Entidades são um conjunto de propriedades, que, por sua vez, são estruturas de dados baseado em chave/valor. Você pode criar até 252 propriedades por entidade, e cada entidade pode ter até 1MB.

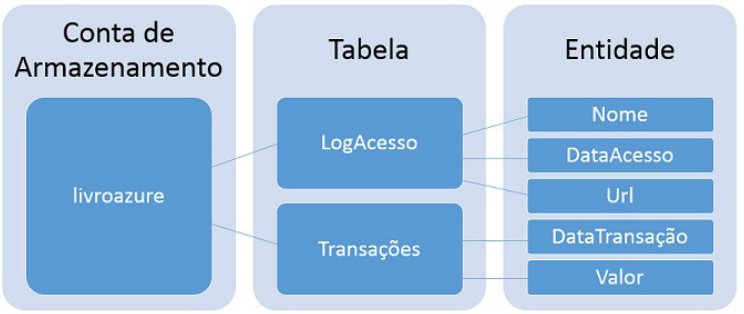

Figura 7.7: Relação conta de armazenamento, tabela, entidade

Cada entidade precisa definir três propriedades de sistema:

- **Partition Key:** armazena um valor em string que identifica a partição a que a entidade pertence;
- **Row Key:** armazena um valor em string e identificam entidades dentro de cada partição;
- **Timestamp:** essa propriedade é uma data mantida pelo serviço e informa a última vez em que a entidade foi alterada.

Antes de iniciarmos a codificação, precisamos adicionar o pacote `WindowsAzure.Storage` via Nuget:

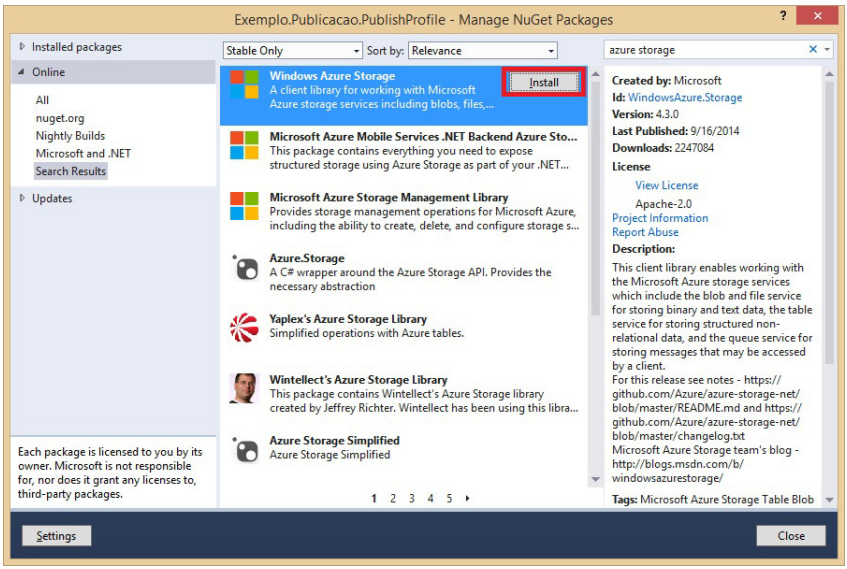

Figura 7.8: Instalando o Nuget WindowsAzure.Storage

Em seguida, precisamos aceitar os termos:

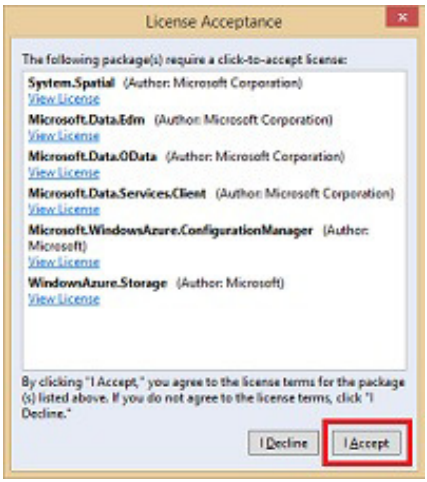

Figura 7.9: Aceitando os termos de uso do pacote

Para demonstrar o funcionamento, vamos criar uma nova classe chamada `LogAcesso`, que vai armazenar a data e hora de acesso, o nome do usuário (se autenticado), o endereço IP e qual página estava sendo acessada.

```
using System;

namespace Exemplo.Publicacao.PublishProfile.Models
{
    public class LogAcesso
    {
        public string Usuario { get; set; }

        public string EnderecoIP { get; set; }
    }
}
```

Para persistir essa classe em uma Azure Table, precisamos adicionar as três propriedades de sistema: Partition Key, Row Key e Timestamp. Estas já estão definidas na interface `ITableEntity`, basta adicionarmos o namespace `Microsoft.WindowsAzure.Storage.Table` e informar que a classe `LogAcesso` herda de uma classe que implemente esta interface. Também vamos aproveitar e informar no construtor da classe os valores para as chaves `PartitionKey` e `RowKey`:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Web;
using Microsoft.WindowsAzure.Storage.Table;

namespace Exemplo.Publicacao.PublishProfile.Models
{
    public class LogAcesso : TableEntity
    {
        public LogAcesso()
        {

        }

        public LogAcesso(DateTime dataAcesso, string url)
```

```
        {
            this.PartitionKey =
                dataAcesso.ToString("ddMMyyyyhhmmss");
            this.RowKey = url;
        }

        public string Usuario { get; set; }

        public string EnderecoIP { get; set; }
    }
}
```

Repare que informamos a data de acesso como `PartitionKey` e a URL como `RowKey`.

Escolha da Partition Key e Row Key

Uma boa dica para definir quais serão essas propriedades é pensar em quais maneiras você gostaria de efetuar buscas nas suas tabelas. Neste exemplo, posso efetuar buscas pela data e hora de acesso e a URL que estava sendo acessada.

É possível efetuar buscas por outras propriedades, no entanto, não será beneficiada pelo uso do índice clusterizado, o que ocasionará em uma performance ruim dado que será feito *fullscan* na tabela.

Definida a classe, agora vamos criar um `ActionFilter` para logar essas informações. Em primeiro lugar, vamos adicionar uma nova classe chamada `LogAcessoFilters`. Eu criei uma nova pasta chamada `Filters` e adicionei essa classe a ela.

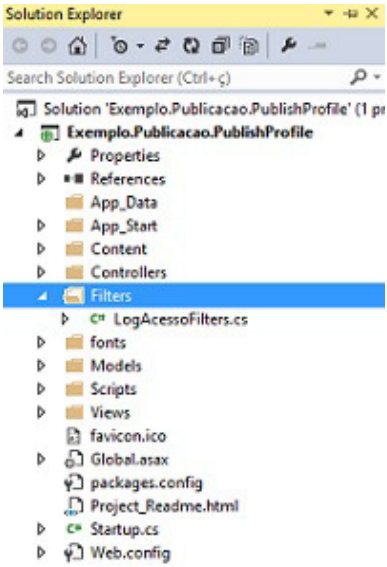

Figura 7.10: Organização do filtro dentro da solução

Em seguida, vamos adicionar o namespace `System.Web.Mvc` a essa classe e informar que a classe `LogAcessoFilters` herda de `ActionFilterAttribute`:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Exemplo.Publicacao.PublishProfile.Filters
{
    public class LogAcessoFilters : ActionFilterAttribute
    {
    }
}
```

Precisamos agora definir em qual momento o log será gravado: antes ou após a execução da `Action`. Para este exemplo, vamos gravar as informações antes que essa classe `Action` seja executada. Basta sobrescrevermos o método `OnActionExecuting` definido na classe `ActionFilterAttribute`:

```
public class LogAcessoFilters : ActionFilterAttribute
{
    public override void
        OnActionExecuting(ActionExecutingContext filterContext)
    {
        base.OnActionExecuting(filterContext);
    }
}
```

Feito isto, basta instanciarmos a classe `LogAcesso` , informando os valores:

```
public override void
    OnActionExecuting(ActionExecutingContext filterContext)
{
    var context = filterContext.RequestContext.HttpContext;
    var request = context.Request;

    var url = request.Url.LocalPath;
    //substitui o caracter '/' por '-'
    url = System.Text.RegularExpressions
                    .Regex.Replace(url, @"[\ /?#]", "-");

    var usuario = "";
    var dataLog = DateTime.Now;
    var enderecoIP = request.UserHostAddress;

    if (context.User.Identity.IsAuthenticated)
        usuario = context.User.Identity.Name;
    else
        usuario = "";

    var logAcesso = new LogAcesso(dataLog, url)
    {
        Usuario = usuario,
        EnderecoIP = enderecoIP
    };

    base.OnActionExecuting(filterContext);
}
```

Em seguida, vamos criar um novo método para persistir o log de acesso. Esse método recebe a instância criada anteriormente como parâmetro de entrada:

```
private static void GravarLog(LogAcesso logAcesso)
{
```

```
}
```

Precisamos usar nossa conta de armazenamento para ter acesso ao serviço Azure Tables. Vamos utilizar a chave de acesso obtida anteriormente e criar a string de conexão para a conta de armazenamento. A string de conexão será dividida em 3 partes:

1. Definição do protocolo http/https;
2. Definição do nome da conta de armazenamento;
3. Definição da chave de acesso (primária ou secundária).

```
var protocolo = "DefaultEndpointsProtocol=http";
var conta = "AccountName=livroazure";

var accessKey =
      "AccountKey=onbwUNOcknqZYDaQzFc8Eata9C+UTE7wqRItJ7Ge2G8D/";
accessKey += "FK3RbbTzgAoLUOxG6CcQVoQDc+1+BPf7s6n8KIq+A==";

var sConexao =
        string.Concat(protocolo, ";", conta, ";", accessKey);
```

Após montarmos a string de conexão, precisamos importar o namespace `Microsoft.WindowsAzure.Storage` para ter acesso à nossa conta de armazenamento:

```
using Microsoft.WindowsAzure.Storage;
```

Sob esse namespace, existe a classe `CloudStorageAccount` que será utilizada para acessar a conta de armazenamento. Precisamos invocar o método `Parse` dessa classe, passando a string de conexão como parâmetro e, depois, obter uma instância para o cliente que vai manipular as tabelas do Azure Tables:

```
var cloudStorageAccount = CloudStorageAccount.Parse(sConexao);
var cloudTableClient =
        cloudStorageAccount.CreateCloudTableClient();
```

Antes de adicionarmos itens na nossa tabela, precisamos garantir que ela exista. Podemos obter uma referência à tabela e, caso ela não exista, disparar uma operação para criá-la:

```
//Criar uma nova tabela, caso não exista
var table = cloudTableClient.GetTableReference("logacesso");
table.CreateIfNotExists();
```

> **Performance**
>
> Validar se a tabela existe ou não, a cada momento que o filter é executado, vai afetar a performance da aplicação. O ideal seria criar os objetos necessários uma única vez, e pular essa checagem nas demais execuções. Para o nosso exemplo, podemos seguir dessa maneira.

Precisamos agora informar qual será o tipo de operação que vamos realizar: `Insert`, `Insert` ou `Replace`, `Replace`, `Merge`, `Delete`, `Retrieve`. Essas operações estão definidas no enumerador `TableOperation`, sendo assim, basta informar qual será a operação desejada.

Para o nosso exemplo, sempre vamos adicionar um novo item utilizando a opção `Insert`:

```
//Comando para inserir
TableOperation insertOperation =
                    TableOperation.Insert(logAcesso);

//executando o comando de insert na nossa tabela
table.Execute(insertOperation);
```

Por último, basta efetuar a chamada ao método `GravarLog` no método `OnActionExecuting`:

```
public override void
    OnActionExecuting(ActionExecutingContext filterContext)
{
    //...código criado anteriormente
    GravarLog(logAcesso);
    base.OnActionExecuting(filterContext);
}
```

Concluímos o desenvolvimento do `Filter` que será executado antes de cada `Action` . Basta agora adicioná-lo às `Action` s em que queremos esse tipo de rastreabilidade:

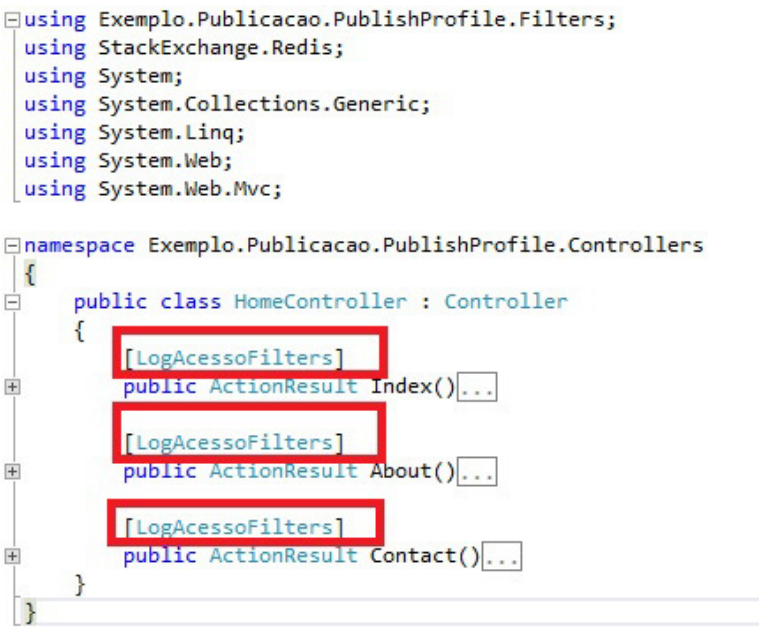

Figura 7.11: Utilização do Action Filter no Controller

Por último, adicionar este filter no `Global.asax.cs` :

```
using Exemplo.Publicacao.PublishProfile.Filters;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace Exemplo.Publicacao.PublishProfile
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
```

```
            GlobalFilters.Filters.Add(new LogAcessoFilters());

            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(
                GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

Ao executarmos e navegarmos pela aplicação, logs de acesso serão armazenados no Azure Tables. Para consultá-los, você pode usar uma ferramenta externa, como o Azure Storage Explorer (http://bit.ly/AzureStorageExplorer); utilizar a operação `TableOperation.Retrieve` em vez de `TableOperation.Insert`; ou criar uma consulta via Linq:

```
TableQuery<LogAcesso> query =
    new TableQuery<LogAcesso>()
    .Where
    (
        TableQuery.GenerateFilterCondition
        (
            "PartitionKey",
            QueryComparisons.GreaterThan ,
            DateTime.Now.ToString("ddMMyyyyhhmm")
        )
    );
IEnumerable<LogAcesso> logs = table.ExecuteQuery(query);
```

7.3 CONCLUINDO

Neste capítulo, aprendemos como usar os Azure Tables para persistir informações não relacionais. Toda informação armazenada no Azure Storage (seja `Table`s, `Queue`s ou `Blob`s) são feitas 3 cópias no mesmo data center e, automaticamente, georreplicado para outro data center no mesmo continente.

Essa ação visa garantir a disponibilidade, tolerância a falhas, desastres naturais, ou qualquer outro problema que possa afetar o

data center. Considere essa opção sempre que você precisar armazenar um grande volume de dados não relacionais de maneira performática e com baixo custo.

Capítulo 8

AZURE SEARCH: A INTELIGÊNCIA DOS SITES DE BUSCA NA SUA APLICAÇÃO

Você sabia que, ao introduzir o recurso "Você quis dizer ...?", o Google duplicou a quantidade de buscas realizadas diariamente? A correção ortográfica é apenas um dos recursos que compõem a árvore de decisão da engine de busca até a exibição dos resultados. Além deste, recursos como sugestões de resultados por equivalência, busca com facetas, highlight de termos, busca por geolocalização e muitos outros são utilizados por sites de busca e e-commerce.

De certa forma, já estamos acostumados a realizar pesquisas dessas maneiras, e não admitimos sites que apresentem menos recursos ou resultados que não correspondam aos critérios especificados. Neste capítulo, vamos aprender como utilizar o serviço Azure Search, uma engine de busca disponível como serviço no Azure.

8.1 BUSCA COMO SERVIÇO (SEARCH AS A SERVICE)

Implementar e manter uma engine de busca não são uma tarefa

fácil. Ou, pelo menos, manter essa engine funcionando com milhões (ou até mesmo bilhões) de registros é uma atividade extremamente complexa. A proposta do Azure Search é entregar uma plataforma completa de busca como serviço (*Search as a Service*). Desta maneira, as complexidades para manter, escalar e até mesmo entregar os resultados da busca ficam por conta da Microsoft. Você só tem o trabalho de:

1. Criar um índice;
2. Definir os campos do índice;
3. Enviar os documentos que você quer indexar (modelo push);
4. Efetuar buscas.

Legal, mas onde posso usar o Azure Search?

Existem diversas aplicações onde o Azure Search pode lhe ajudar. Outro possível cenário são aplicações que fazem análise sobre conteúdos que são produzidos em mídias sociais.

Imagine determinada marca ou produto que você precise analisar a opinião dos clientes. Realizar uma busca utilizando o recurso *fulltext search*, pode não apresentar resultados em um tempo de resposta aceitável.

Imagine agora um aplicativo para dispositivos móveis que use as coordenadas onde o usuário se encontra, e apresente apenas resultados em um raio de até X quilômetros de distância.

Estes são apenas alguns exemplos do que é possível construir utilizando o Azure Search. Você foca no negócio e delega as complexidades para o Azure.

8.2 PROVISIONANDO O AZURE SEARCH

O primeiro passo é acessar a nova versão do portal (Ibiza).

Efetue o login em http://portal.azure.com.

Em seguida, selecione a opção `Novo -> Dados + Armazenamento -> Pesquisar`:

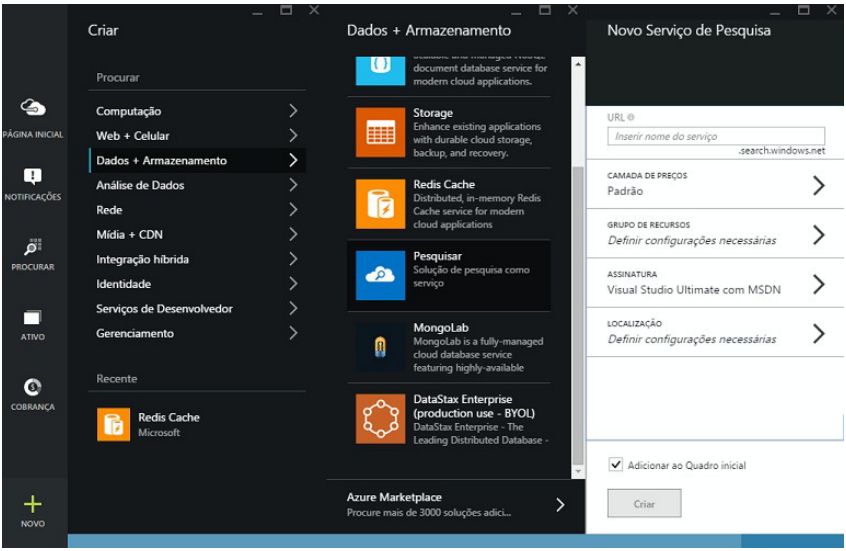

Figura 8.1: Provisionando o Azure Search no portal Ibiza

Assim como diversos outros serviços do Azure, será disponibilizado um *endpoint* (URL) para que você consuma esse serviço de busca. Informe o prefixo da URL e selecione a opção `Gratuito` na seção `CAMADA DE PREÇOS`.

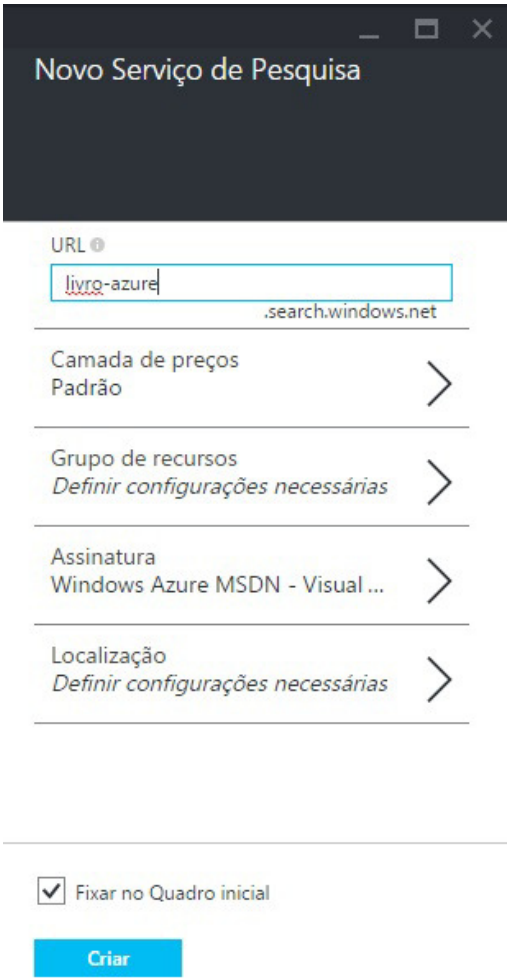

Figura 8.2: Informando o nome do serviço de busca

Camada de preços

O modo gratuito suporta 10.000 documentos, 3 índices e a máquina é compartilhada com outros clientes. Para os exemplos deste livro, podemos prosseguir com este modo. No entanto, se você precisa utilizar em produção, migre para o modo *standard* (padrão), que suporta até 15 milhões de documentos e 50 índices em uma máquina dedicada e com suporte para aumentar o número de instâncias (máquinas).

Depois, crie um novo grupo de recursos para armazenar esse serviço:

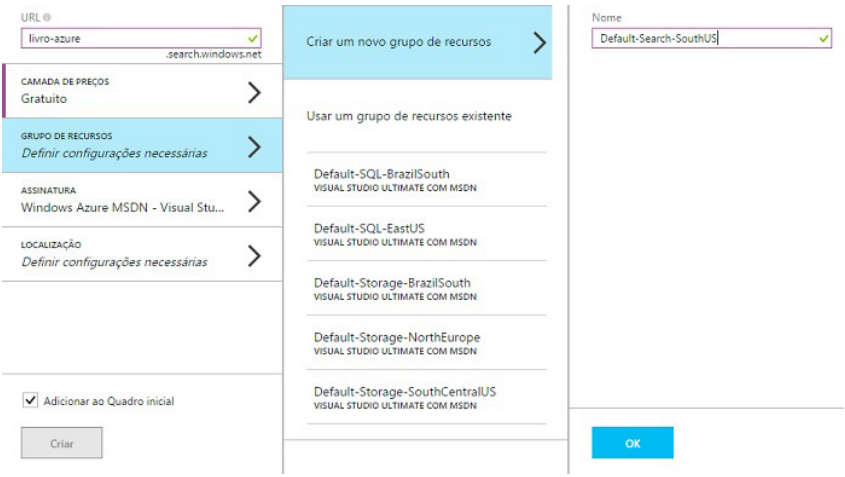

Figura 8.3: Criando um novo grupo de recursos

Selecione a assinatura e o data center, e clique no botão `Criar`. Após a conclusão do provisionamento, copie e cole a URL e as chaves de acesso para um bloco de notas, para prosseguirmos com os exemplos.

Opções de data centers

Observação: no momento em que escrevo este capítulo, o Azure Search ainda não está disponível no data center do Brasil. Selecione o data center mais próximo para evitar problemas de latência.

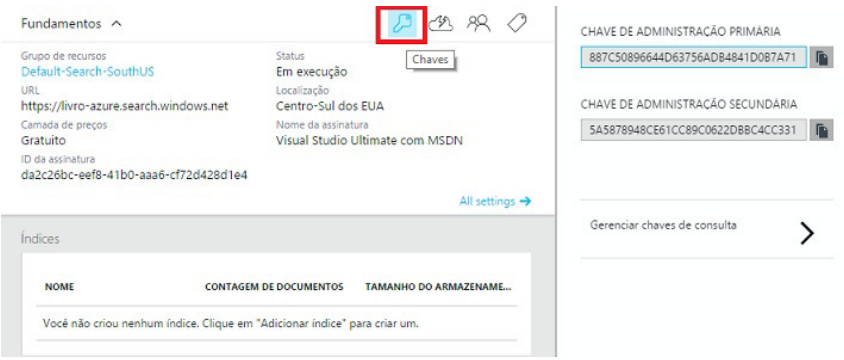

Figura 8.4: Obtendo as chaves de acesso

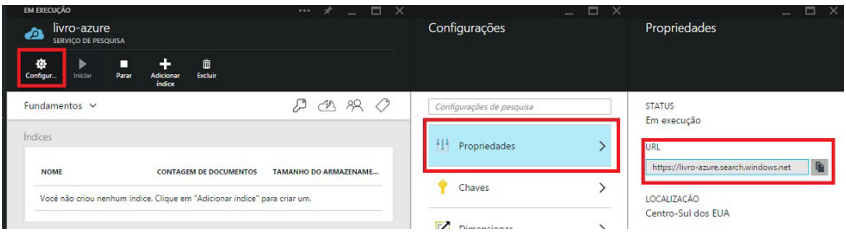

Figura 8.5: Obtendo o endpoint

8.3 DEFININDO A ESTRUTURA DOS DOCUMENTOS

Antes de indexarmos os documentos (conteúdo), precisamos definir suas estruturas. Para isso, precisamos criar um índice no

portal de gerenciamento do Azure. Clique sobre a opção `Adicionar índice` e, em seguida, informe o nome do índice e clique sobre o botão `OK`:

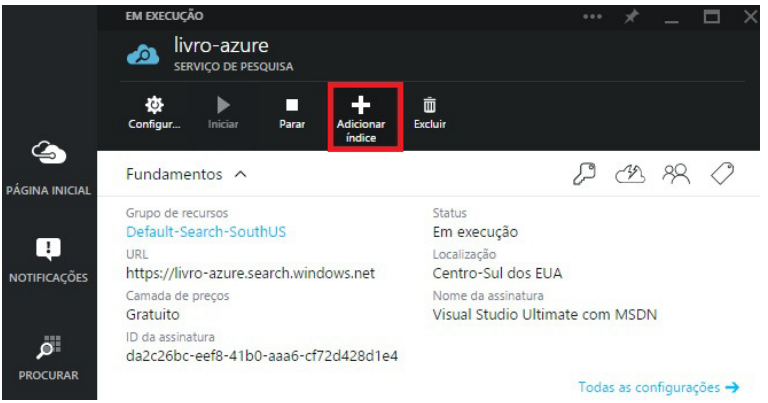

Figura 8.6: Adicionando um novo índice

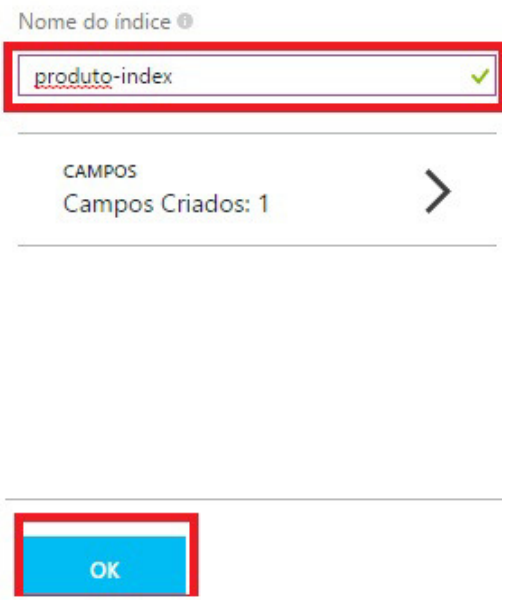

Figura 8.7: Informando o nome do novo índice

Depois, clique sobre o nome do índice informado no passo anterior para definir a estrutura dos documentos e, em seguida, clique sobre a opção `Adicionar campos`:

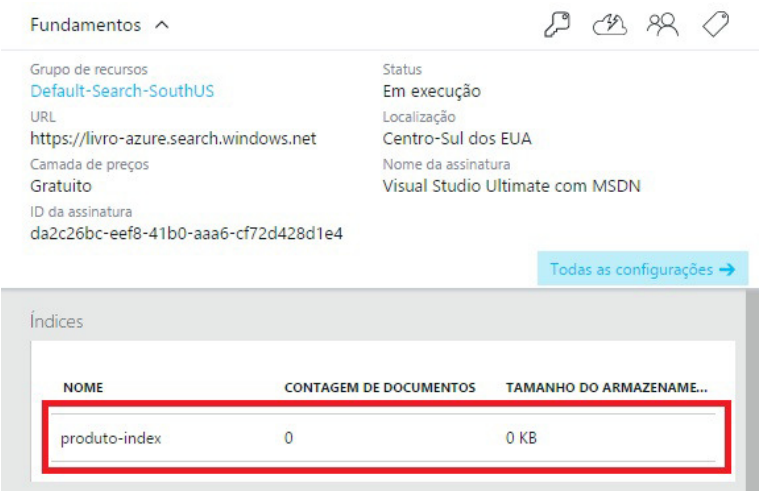

Figura 8.8: Nenhum documento indexado

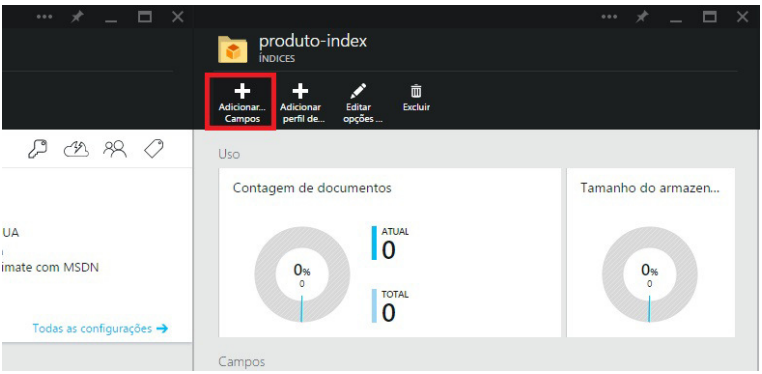

Figura 8.9: Adicionando os campos

A estrutura para esse exemplo ficou definida da seguinte maneira:

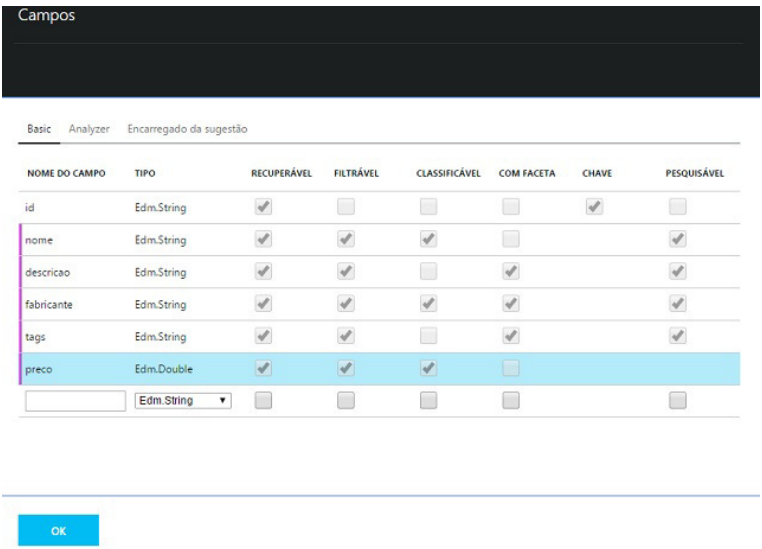

Figura 8.10: Definição da estrutura do documento

Agora, precisamos enviar documentos que respeitem esse índice ao serviço Azure Search. Para facilitar o exemplo, vou recorrer à ferramenta Fiddler para enviar as requisições HTTP ao serviço.

> **Download Fiddler**
>
> Faça o download gratuitamente do Fiddler em http://bit.ly/FiddlerDownload.

8.4 INDEXANDO DOCUMENTOS NO AZURE SEARCH

Agora que já criamos o índice com a definição dos campos, precisamos enviar um `POST` ao serviço com documentos que respeitem a estrutura definida. Vamos utilizar a URL do serviço

disponibilizada no portal do Azure, seguida do nome do índice criado na seção anterior, e a operação e versão da API REST:

Decompondo a URL do Azure Search, temos as seguintes partes:

1. **Prefixo da URL do serviço:** https://livro-azure.search.windows.net/
2. **Nome do índice:** indexes/produto-index/
3. **Operação:** docs/index?api-version=2015-02-28-preview

A URL completa ficou assim: https://livro-azure.search.windows.net/indexes/produto-index/docs/index?api-version=2015-02-28-preview.

Em seguida, precisamos informar os cabeçalhos `api-key`, com uma das chaves de acesso do serviço (primária ou secundária), e `Content-Type`, informando `application/json`. Por fim, basta colar o JSON no corpo da requisição:

```
{
"value":
[
    {
        "id": "1",
        "nome": "televisão led 42",
        "descricao": "Smart Tv Led 3d Full Hd 42 polegadas",
        "fabricante": "TH",
        "tags": "smart tv, tv led, 42, wi-fi",
        "preco": 1900.00
    },
    {
        "id": "2",
        "nome": "notebook",
        "descricao": "Notebook SSD 256GB 8GB RAM 13 polegadas",
        "fabricante": "HELL",
        "tags": "notebook, ssd",
        "preco": 3900.00
    },
    {
        "id": "3",
        "nome": "mouse",
        "descricao": "Mouse ambidestro ergonômico Sensor Duplo
```

```
                    4G 8200 dpi",
        "fabricante": "TH",
        "tags": "mouse, games",
        "preco": 900.00
    }
]
}
```

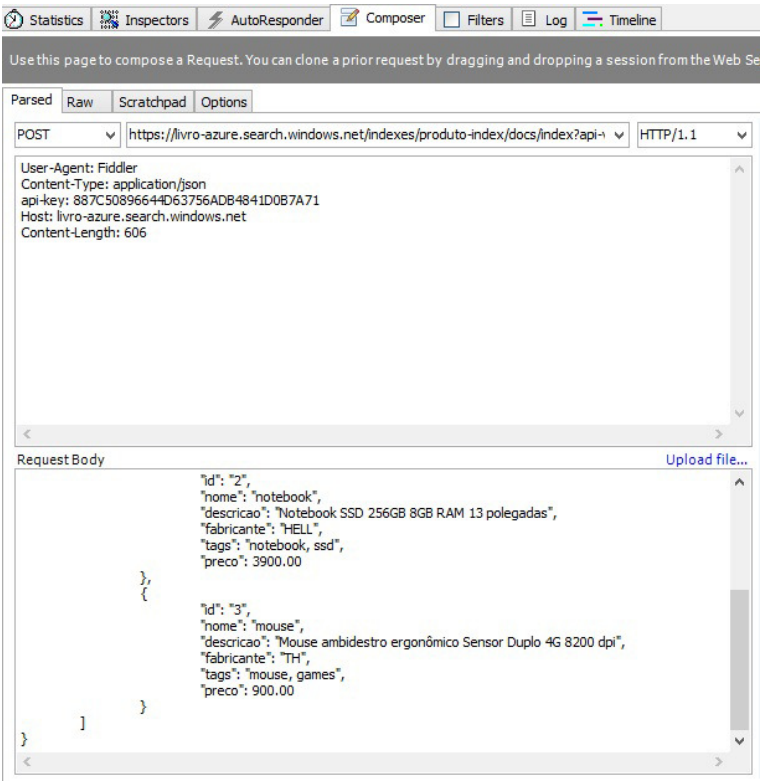

Figura 8.11: Utilizando o fiddler para compor a requisição HTTP

Após o envio da requisição, confira no portal se os documentos foram indexados corretamente:

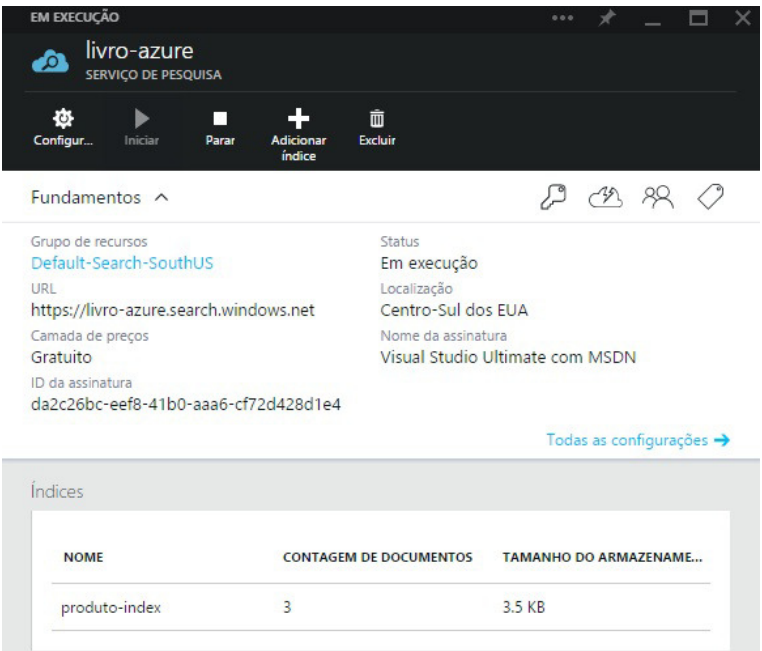

Figura 8.12: Documentos indexados

8.5 PESQUISANDO DOCUMENTOS NO AZURE SEARCH

Como visto anteriormente, para indexar documentos, o serviço Azure Search expõe uma REST API. Através dela, vamos efetuar as nossas buscas.

Buscando documentos

O primeiro parâmetro de busca que vamos utilizar é o `search`, onde vamos especificar um termo e o serviço procurará em cada um dos campos que foram marcados como "Pesquisável" na criação do índice.

A sintaxe da URL de pesquisa é: `https://` **`[sn]`** `.search.windows.net/indexes/` **`[in]`** `/docs?`

`[qp]` .

Onde: [**sn**] é o prefixo da URL do seu serviço de busca; [**in**] é o nome do índice criado na seção anterior; e [**qp**] são os parâmetros de busca.

Seguindo essa definição, minha URL de pesquisa ficou assim:

```
https://livro-azure.search.windows.net
    /indexes
    /produto-index
    /docs?search=th&api-version=2015-02-28-preview
```

Observação: a URL foi quebrada em várias linhas para facilitar o entendimento.

> **Versão da API**
>
> No momento em que escrevo este capítulo, a versão da API mais recente é a 2015-02-28 (ainda em preview). É possível que uma nova versão já esteja disponível no momento em que você lê este capítulo. Confira a documentação no site do Azure.

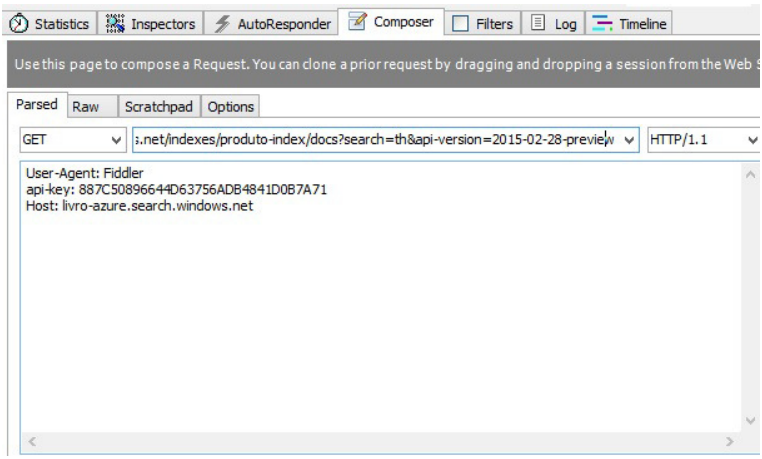

Figura 8.13: Versão da API na URL

Após disparar essa requisição via Fiddler, o serviço retornou a seguinte resposta:

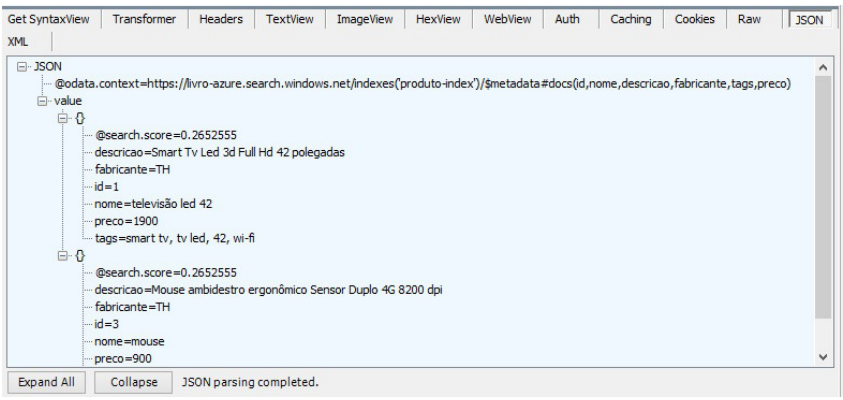

Figura 8.14: JSON de resposta

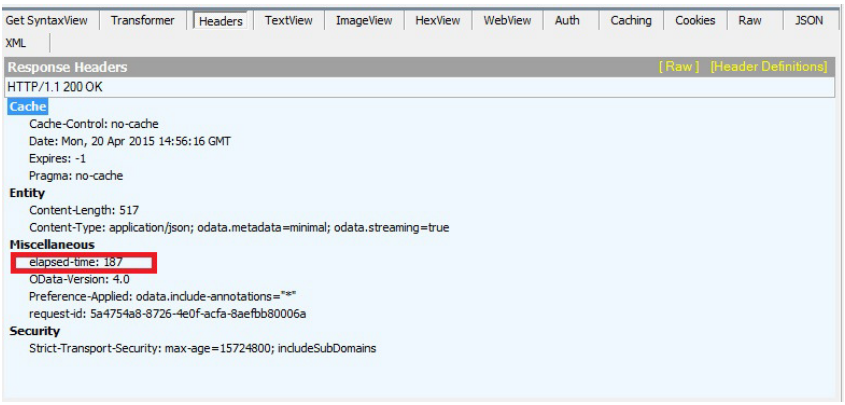

Figura 8.15: Tempo de resposta da requisição HTTP

Repare que o tempo total entre o disparo da requisição e a resposta foi de apenas 187 milissegundos.

Efetuando busca em apenas um campo

No exemplo anterior, o termo especificado foi pesquisado em cada um dos campos que foram marcados como "Pesquisável". Pode ser que você queira realizar a pesquisa em apenas um campo; para isso, basta informar o parâmetro `searchFields`, informando o nome do campo que deve conter o termo:

```
GET /indexes
    /produto-index //nome do índice
    /docs?search=hd //termo a ser pesquisado
    &searchFields=descricao //campo que deve conter o termo
    &api-version=2015-02-28-preview //versão da API REST
```

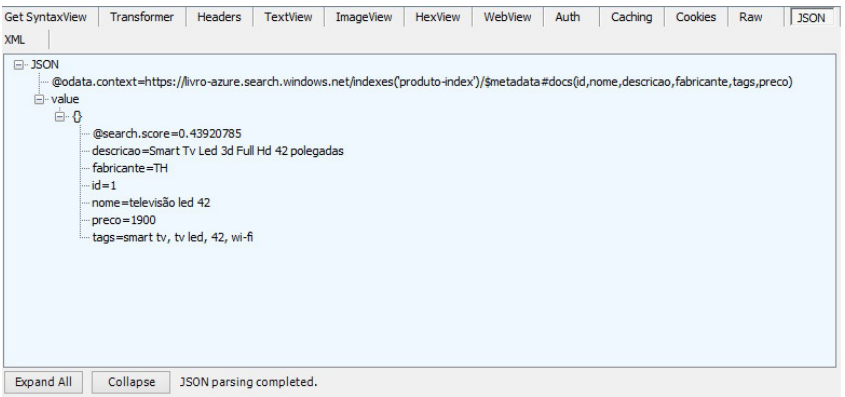

Figura 8.16: JSON de resposta à consulta

Você também pode destacar o trecho no qual o termo pesquisado aparece; para isso, basta informar o parâmetro de pesquisa `highligh` e o valor do campo em questão:

```
GET /indexes
    /produto-index  //nome do índice
    /docs?search=hd  //termo a ser pesquisado
    &searchFields=descricao //campo que deve conter o termo
    &highlight=descricao //destacar o termo no campo desricacao
    &api-version=2015-02-28-preview //versão da API REST
```

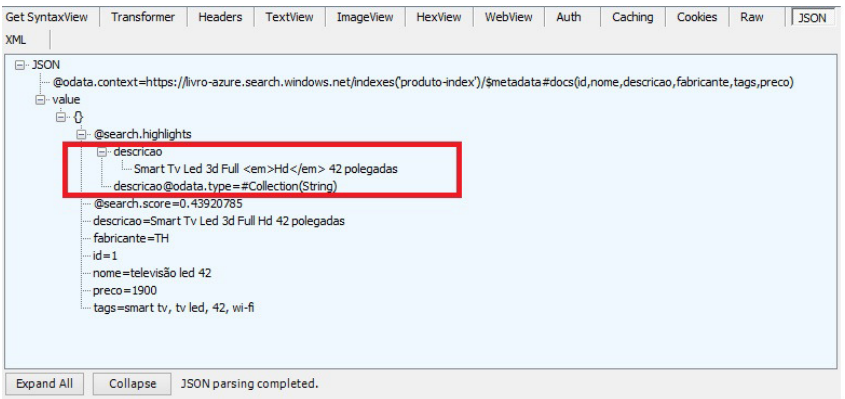

Figura 8.17: JSON de resposta à consulta

Eventualmente, os documentos podem possuir diversos campos,

ou o resultado da pesquisa pode retornar diversos documentos. O que fazer para filtrar e manipular os documentos de resposta?

Utilizando OData

O OData é um protocolo aberto criado pela Microsoft que tem como objetivo definir as melhores práticas para construir e consumir APIs REST. A API do Azure Search já está preparada para suportar OData, sendo assim, para selecionar apenas alguns campos do documento, vamos utilizar o parâmetro `$select`, passando a lista de campos desejada:

```
GET /indexes
    /produto-index/   //nome do índice
    /docs?search=hd //termo a ser pesquisado
    &searchFields=descricao  //campo que deve conter o termo
    &$select=id,nome,preco //lista de campos desejados
    &api-version=2015-02-28-preview //versão da API REST
```

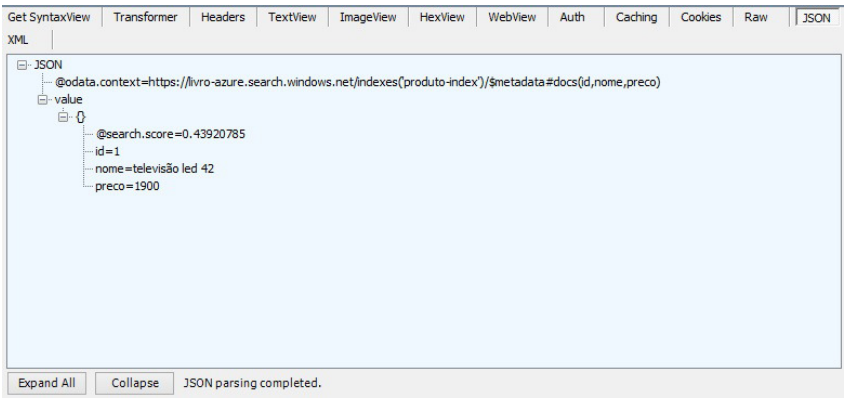

Figura 8.18: JSON de resposta à consulta

No exemplo anterior, selecionamos apenas os campos `id`, `nome` e `preco`. Para filtrar a quantidade de documentos da resposta, podemos usar o parâmetro `$top`:

```
GET /indexes
    /produto-index //nome do índice
    /docs?search=*
```

```
    &$top=1 //apenas o primeiro registro
    &api-version=2015-02-28-preview //versão da API REST
```

Você também pode retornar a resposta de maneira ordenada, basta usar o parâmetro `$orderby` :

```
GET /indexes
    /produto-index //nome do índice
    /docs?search=*
    &$orderby=preco //ordena pelo preco
    &api-version=2015-02-28-preview //versão da API REST
```

Para ordenar de maneira decrescente, basta adicionar a palavra `desc` após o nome do campo de ordenação:

```
GET /indexes
    /produto-index //nome do índice
    /docs?search=* //todos os documentos
    &$orderby=preco%20desc //ordena pelo preco. %20 adiciona
                           //espaço
    &api-version=2015-02-28-preview //versão da API REST
```

Figura 8.19: JSON de resposta à consulta

ODATA

Para maiores informações sobre OData, confira o site http://www.odata.org.

Busca com facetas

Você se lembra de quando criamos a definição dos campos do índice, e selecionamos a opção `COM FACETA` para alguns campos? Este recurso permite a navegação *drill-down* pelos resultados de busca. Vamos supor que, ao pesquisar por determinado termo, você queira retornar os produtos de mesmo fabricante:

```
GET
    /indexes
    /produto-index //nome do índice
    /docs?search=* //todos os documentos
    &facet=fabricante //busca com faceta sobre o campo fabricante
    &api-version=2015-02-28-preview //versão da API
```

Como resposta, o serviço retornou:

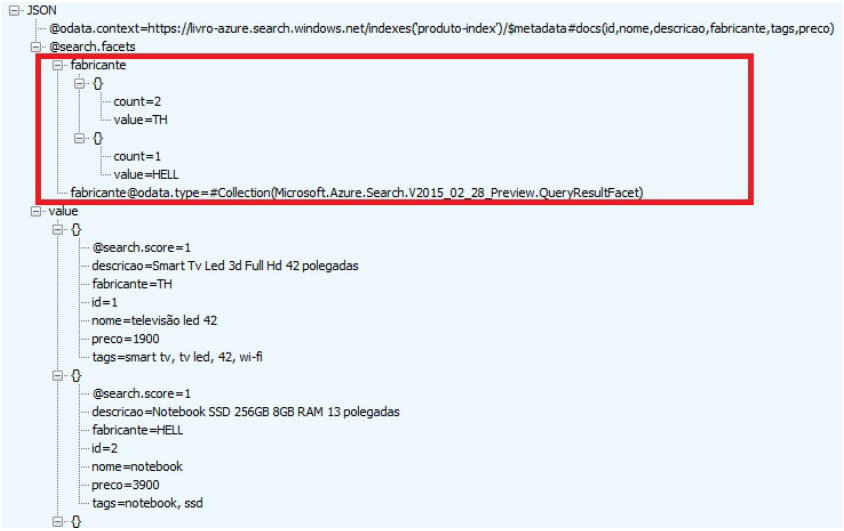

Figura 8.20: JSON de resposta à consulta

É possível montar um menu mostrando ao usuário que, para o fabricante `'TH'`, existem dois produtos e, para o fabricante `'HELL'`, existe um.

Adicionando este recurso, você enriquecerá a experiência do usuário sem prejudicar a performance do seu banco de dados com queries complexas para produzir esse mesmo resultado. Ao clicar sobre o link do fabricante `'TH'`, você só tem de filtrar o resultado especificando que apenas os documentos daquele fabricante devem ser retornados:

```
/indexes/
/produto-index //nome do índice
/docs?search=* //todos os documentos
&facet=fabricante //busca com faceta sobre o campo fabricante
&$filter=fabricante eq 'TH' //filtrando apenas do fabricante TH
&api-version=2015-02-28-preview //versão da API
```

8.6 CONCLUINDO

Neste capítulo, vimos como utilizar o serviço integrado Azure

Search, que fornece uma rica experiência de busca para sites, aplicativos móveis ou qualquer outro tipo de aplicação que você esteja desenvolvendo.

Para maiores informações sobre as possibilidades de consulta ao Azure Search e a utilização de OData, confira a documentação oficial da Microsoft em http://bit.ly/AzureSearchDoc.

Capítulo 9

PERSISTINDO DOCUMENTOS COM O MICROSOFT AZURE DOCUMENTDB

Nos capítulos anteriores, estudamos Azure Tables e Azure Redis Cache, e dois NoSQLs baseados em chave/valor. Neste capítulo, vamos estudar um outro tipo de banco de dados não relacional: baseados em documentos.

9.1 PERSISTÊNCIA DE DADOS NÃO RELACIONAIS – #NOSQL

Nos últimos anos, vimos um movimento crescer inspirado pela publicação de *whitepapers* e pelo lançamento do DynamoDB e BigTable (NoSQLs da Amazon e do Google, respectivamente). Estes dois grandes players viram a necessidade de criar um novo mecanismo para armazenarem seus dados, pois mantê-los em um banco de dados relacional já não era mais uma opção viável.

Com a publicação desses novos mecanismos para persistência, alguns desenvolvedores resolveram se reunir para discutir outras maneiras de persistência não relacional. Este encontro acabou ganhando a hashtag *#NoSQL*, e passou a ser o termo de referência

quando o assunto é persistência não relacional.

Até o momento, os NoSQLs podem ser classificados em quatro categorias: chave/valor, documentos, família de colunas e grafos. Apesar de resolverem problemas diferentes, existem algumas características comuns entre estes tipos de NoSQL:

- **Agnóstico a modelos:** um modelo (esquema) de banco de dados é a descrição de todos os possíveis dados e estrutura de dados em um banco de dados relacional. Em um NoSQL, um modelo não é necessário, o que lhe dá liberdade para armazenar documentos (informações), sem a necessidade de a ferramenta conhecer a estrutura deste documento. Também resolve a questão de *impedance mismatch*, que é ter objetos representados na memória de uma forma e estes mesmos dados armazenados de outra maneira em disco. Isto é, dados divididos entre diversas tabelas em um banco de dados relacional.

- **Não relacional:** relacionamentos em um banco de dados estabelecem conexões entre tabelas. Por exemplo, uma tabela de pedidos pode estar relacionada a outra, com os itens desses pedidos. Em um NoSQL, esses dados são armazenados como um agregado (ver significado no box a seguir), um único registro com todos os dados relacionados – pedido, itens, endereço de entrega etc.

- **Capacidade para rodar em cluster:** um banco de dados relacional foi projetado para executar em uma única máquina. Muitas vezes, é mais econômico executar grandes volumes de dados/cargas de dados em clusters (grupos) de máquinas pequenas e baratas, favorecendo cenários de big data.

Observação: banco de dados relacionais podem rodar em cluster, no entanto, garantir transações ACID (acrônimo de *Atomicidade, Consistência, Isolamento e Durabilidade*, propriedades de transações em banco de dados) é extremamente custoso do ponto de vista de manutenção, escalabilidade e performance.

> **AGREGADO**
>
> Agregado é um padrão catalogado em *Domain-Driven Design*. Um agregado é um conjunto de objetos relacionados que passam a ser tratados como uma unidade única. Para maiores informações, consulte http://bit.ly/AggregateDDD.

JSON

NoSQLs baseados em documentos geralmente utilizam a notação JSON (*Javascript Object Notation*) para a representação de dados. JSON é uma maneira de armazenar informações de uma forma organizada e de fácil acesso. Por exemplo:

```
{
    "Alunos" :
    [
        { "nome": "João", "notas": [ 8, 9, 7 ]  },
        { "nome": "Maria", "notas": [ 8, 10, 7 ] },
        { "nome": "Pedro", "notas": [ 10, 10, 9 ] }
    ]
}
```

Nesse JSON, temos um array de `Alunos` com 3 posições (`João` , `Maria e Pedro`), e cada aluno, possui 3 `notas` .

Se tentássemos representar essa mesma estrutura usando XML ou qualquer outro formato, muito provavelmente usaríamos muito mais texto para obter o mesmo resultado. Consequentemente,

precisaríamos de mais dados para armazenar essa estrutura, ou seja, seria necessário mais espaço em disco para armazenar a mesma informação.

9.2 AZURE DOCUMENTDB, O NASCIMENTO

Embora já existissem diversas opções de bancos de dados não relacional baseados em documentos, a Microsoft enxergou a possibilidade de aproximar o mundo relacional do não relacional. Além de oferecer o DocumentDB como serviço gerenciado no Azure, ela adicionou recursos consagrados dos bancos de dados relacionais, como suporte a stored procedures, funções definidas por usuários, gatilhos (*triggers*), consultas ricas usando SQL e processamento transacional.

O Azure DocumentDB foi criado do zero para atender (inicialmente) as demandas internas da Microsoft utilizando tecnologias desenvolvidas pelo centro de pesquisa Microsoft Research. Tais tecnologias já estão disponíveis em outros produtos, como os recursos em memória introduzidas no SQL Server 2014, por exemplo (Hekaton).

Algumas verticais do portal MSN.com, OneNote do Office 365 e outros serviços utilizam o DocumentDB como mecanismo de persistência. Outro grande diferencial desse serviço é a baixa latência e a otimização de gravação dos dados em discos SSDs.

Iniciando com o DocumentDB

Como primeiro passo, precisamos logar na nova versão do portal (Ibiza) por meio da URL: http://portal.azure.com.

Em seguida, selecione a opção `Novo -> Data + Storage + Azure DocumentDB`.

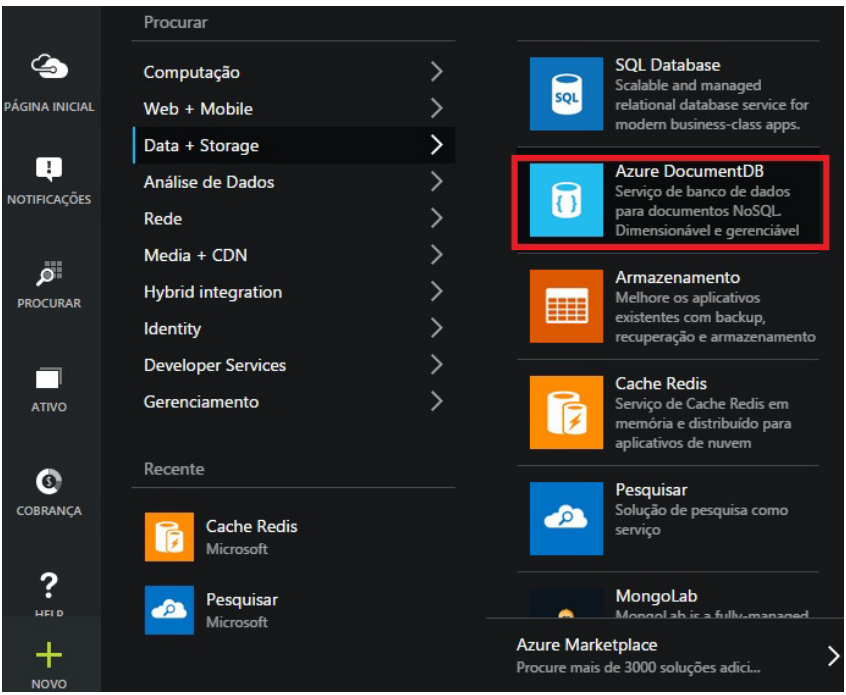

Figura 9.1: Provisionando o DocumentDB no portal Ibiza

Informe o nome da conta do DocumentDB, e selecione a assinatura e o data center. Por fim, clique em `Criar`:

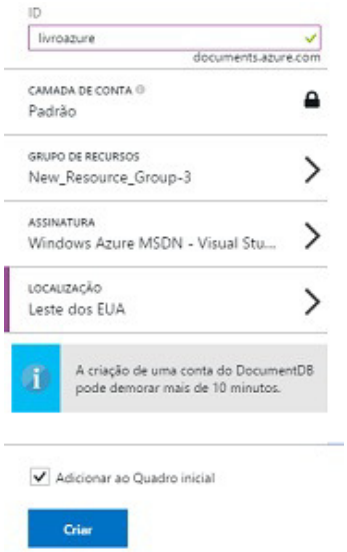

Figura 9.2: Informando o nome do serviço DocumentDB

Observação

O provisionamento desse serviço leva aproximadamente 10 minutos. O time do Azure já está trabalhando para reduzir esse tempo.

Após a criação da nossa conta no serviço DocumentDB, podemos criar *Databases*, usuários e permissões de usuário, coleções de documentos, stored procedures, triggers, user defined functions e anexos.

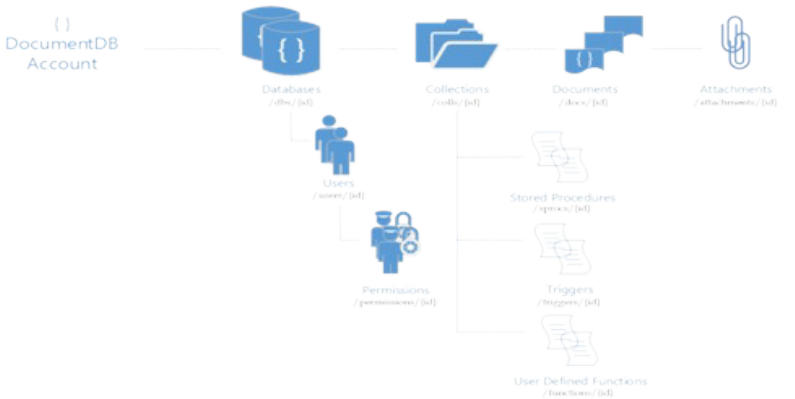

Figura 9.3: Estrutura DocumentDB

Para seguir com o nosso exemplo, precisamos criar um database. Basta clicar sob o botão `Adicionar Banco de Dados` e informar um nome para ele. Em seguida, clique sobre o botão `OK` :

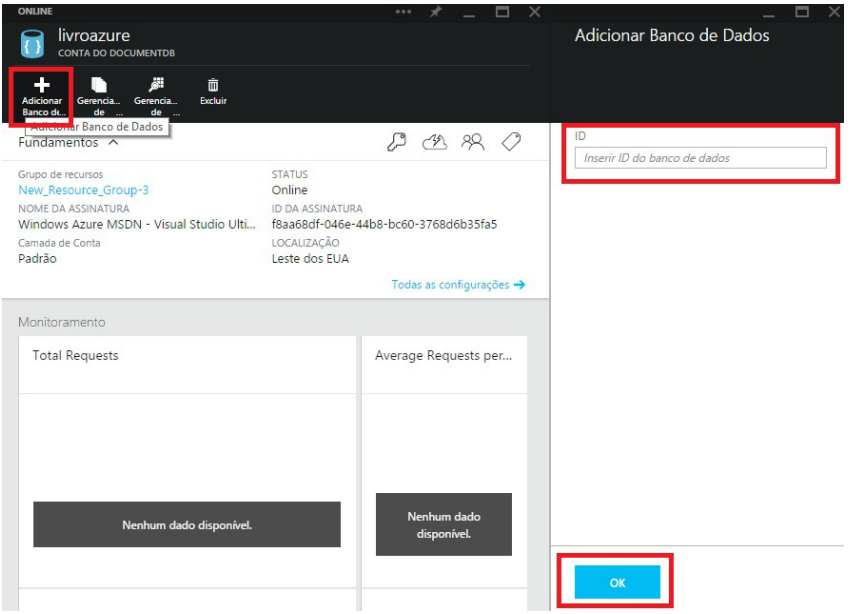

Figura 9.4: Informando o nome do banco de dados

Após o provisionamento desse database, precisamos acessar as chaves de acesso para criar coleções, documentos, stored procedures e os demais objetos via programação. Selecione no menu da esquerda o ícone referente a `CHAVES` :

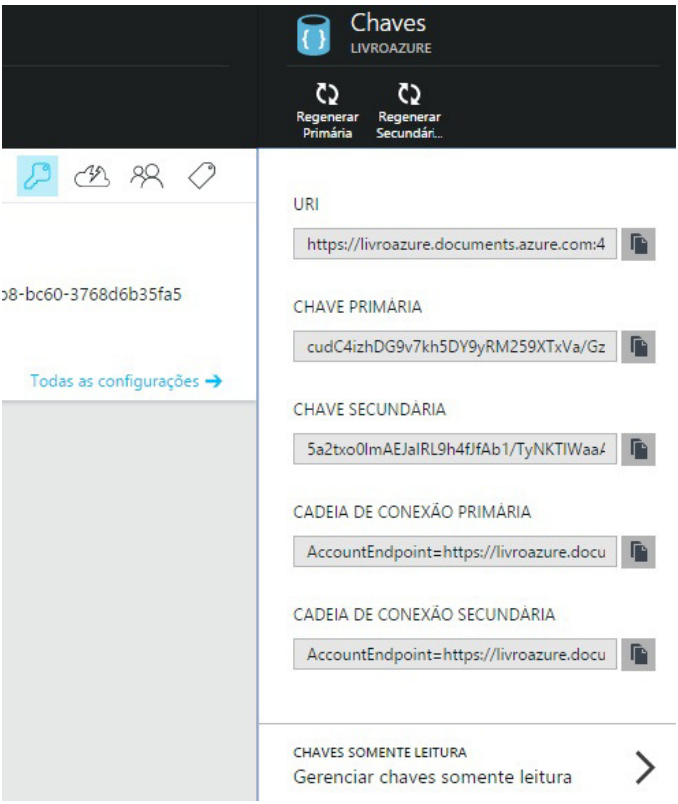

Figura 9.5: Chaves de acesso

9.3 PRIMEIROS TESTES COM DOCUMENTDB

Vamos criar um novo projeto no Visual Studio para armazenarmos documentos no DocumentDB que acabamos de criar:

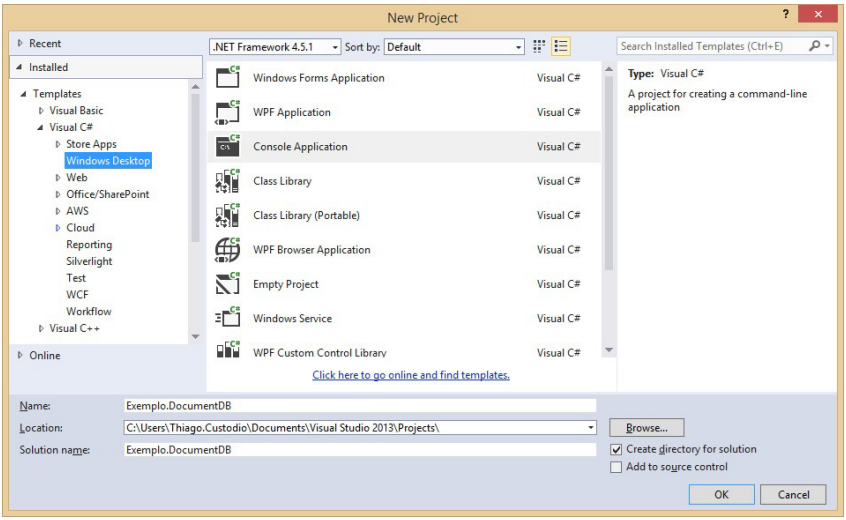

Figura 9.6: Criando um novo projeto no Visual Studio

Antes de iniciarmos a codificação, precisamos adicionar o pacote `Microsoft.Azure.DocumentDB` via Nuget:

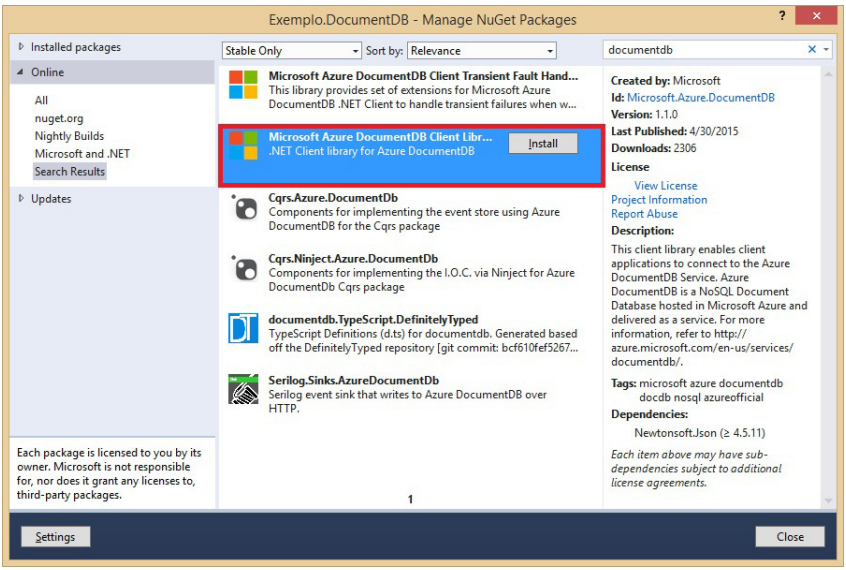

Figura 9.7: Adicionando o pacote DocumentDB

Após a instalação, importe os namespaces abaixo da classe `Program.cs`:

```
using Microsoft.Azure.Documents;
using Microsoft.Azure.Documents.Client;
using Microsoft.Azure.Documents.Linq;
```

Em seguida, crie duas variáveis do tipo `string` para armazenar a URL do DocumentDB criado, e a chave primária e secundária para acesso:

```
class Program
{
    static string _endpoint =
            "https://livroazure.documents.azure.com:443/";

    // utilize sua chave primária aqui
    static string _primaryKey = "";
}
```

Precisamos agora definir de que maneira será realizado o acesso ao serviço definindo o protocolo (via tcp ou https) e o modo da conexão (direto ou gateway).

```
static void Main(string[] args)
{
    var connectionPolicy = new ConnectionPolicy
    {};

        ConnectionMode = ConnectionMode.Gateway,
        ConnectionProtocol = Protocol.Https

    //continuação a seguir
}
```

Observação: caso você esteja acessando o DocumentDB diretamente do Azure (de um Azure WebApp, por exemplo), use `ConnectionMode.Direct`.

Na sequência, vamos adicionar duas classes para representar nossos documentos no DocumentDB: `Evento` e `Participante`:

```
using Microsoft.Azure.Documents;
using System;
```

```
using System.Collections.Generic;

public class Evento : Resource
{
    public Evento()
    {
        Participantes = new List<Participante>();
    }

    public string Nome { get; set; }

    public DateTime? DataEvento { get; set; }

    public List<Participante> Participantes { get; set; }
}
```

```
using System;

namespace Exemplo.DocumentDB
{
    public class Participante
    {
        public string Nome { get; set; }

        public string Email { get; set; }
    }
}
```

Definida a `connectionPolicy` e com as classes de `Participante` e `Evento` criadas, precisamos instanciar o `DocumentClient` para manipular o nosso DocumentDB:

```
using ( var client = new DocumentClient(
            new Uri(_endpoint),
            _primaryKey,
            connectionPolicy)
       )
{
    //continuação a seguir
}
```

Através do objeto `DocumentClient`, criaremos um `Database`, ou obteremos referência a um `Database` criado previamente:

```
private static async Task<Database>
    CriarDatabase(DocumentClient client, string dbID)
{
```

```
    var databases = client.CreateDatabaseQuery()
          .Where(db => db.Id == dbID).ToArray();

    if (databases.Any())
    {
        return databases.First();
    }

    return await client.CreateDatabaseAsync(
                                new Database { Id = dbID});
}
```

Como criamos o `Database` diretamente via portal do Azure, o `client` retornará uma referência a esse database. Para criar uma coleção de documentos, precisamos apenas invocar o método `CreateDocumentCollectionQuery` do objeto `DocumentClient`, informando o `ID` da Coleção e passando a referência ao `Database` que estamos usando (propriedade `SelfLink`):

```
private static async Task<DocumentCollection>
        CriarColecao(
                        DocumentClient client,
                        Database database,
                        string colID
                    )
{
    var collections =
            client
                .CreateDocumentCollectionQuery(database.SelfLink)
                .Where(col => col.Id == colID).ToArray();

    if (collections.Any())
    {
        return collections.First();
    }

    return await client.
            CreateDocumentCollectionAsync
                    (database.SelfLink,
                        new DocumentCollection
                        {
                            Id = colID
                        }
                    );
}
```

Com o database e a coleção criados, precisamos apenas criar um novo `Evento` e armazená-lo, utilizando o método `CreateDocumentAsync`:

```
var evento = new Evento
{
    Nome = "Azure Summit Brasil 2015",
    Participantes = new List<Participante>
    {
        new Participante
        {
            Nome = "Thiago Custódio",
            Email = "thiago.custodio@hotmail.com"
        }
    }
};

var result = CriarDocumento(client, col, evento).Result;
```

O método `CriarDocumento` recebe o `DocumentClient`, o `DocumentCollection` e o `evento` como parâmetros:

```
private static async Task<bool> CriarDocumento
   (DocumentClient client, DocumentCollection col, Evento evento)
{
    await client.CreateDocumentAsync(col.SelfLink, evento);
    return true;
}
```

9.4 CONSULTANDO DOCUMENTOS NO DOCUMENTDB

Como dito anteriormente, a Microsoft oferece suporte à linguagem T-SQL para realizar consultas em documentos:

```
private static void
ConsultaComSQL(DocumentClient client, DocumentCollection col)
{
    var resultado = client.CreateDocumentQuery
    (
        col.SelfLink,
            "SELECT E.Nome, E.Participantes " +
            "FROM root E WHERE "+
            "E.Nome = 'Azure Summit Brasil 2015' "
```

```
    );

    foreach (var ev in resultado)
    {
        Console.WriteLine("O evento possui {0} participante(s).",
                ev.Participantes.Count);
    }
}
```

No próprio portal, há uma funcionalidade para efetuar consultas utilizando SQL:

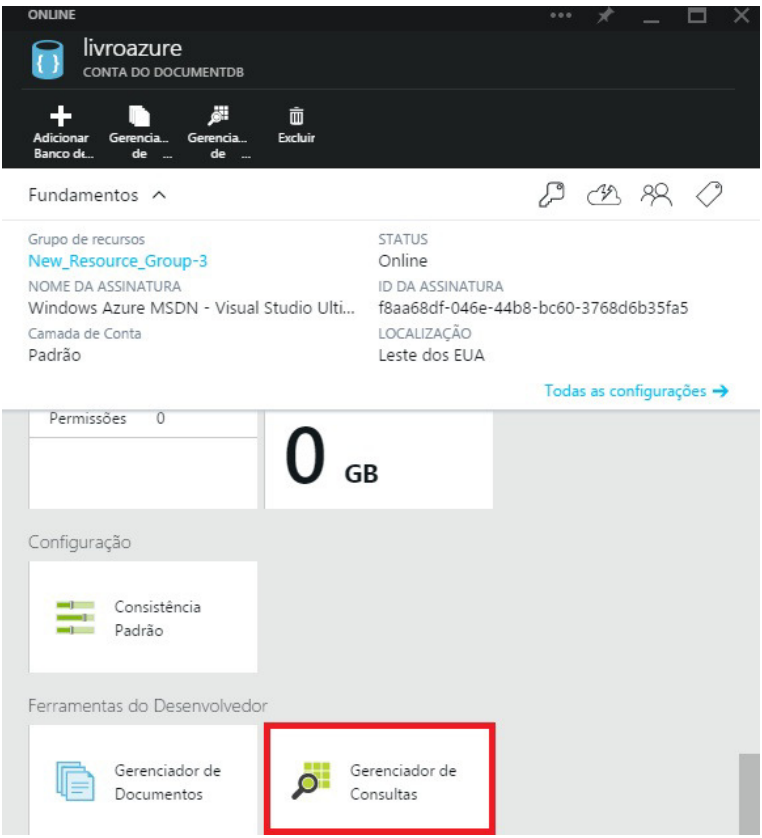

Figura 9.8: Gerenciador de consultas no portal Ibiza

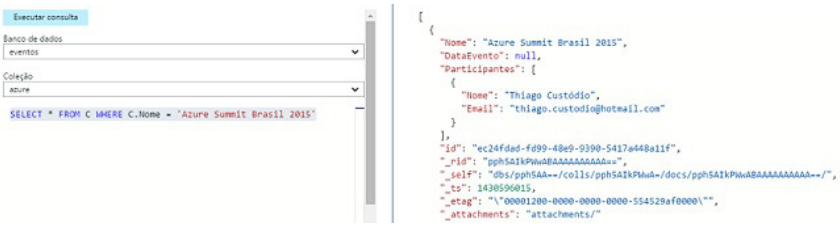

Figura 9.9: Exemplo de consulta diretamente pelo gerenciador

SANDBOX

Você pode treinar consultas SQL com joins, projeções e filtros usando `DocumentDB` diretamente pela URL: http://bit.ly/SandboxDocumentDB.

O mesmo resultado da consulta anterior pode ser obtido ao usarmos LINQ:

```
private static void ConsultaComLinq
    (
        DocumentClient client,
        DocumentCollection col
    )
{
    var eventos =
            from evt in client.CreateDocumentQuery<Evento>(
                                    col.SelfLink)
            where evt.Nome == "Azure Summit Brasil 2015"
            select new { evt.Nome, evt.Participantes };

    foreach (var ev in eventos)
    {
        Console.WriteLine("O evento possui {0} participante(s).",
                ev.Participantes.Count);
    }
}
```

Particularmente, prefiro o uso de LINQ, pois evita que erros de digitação sejam pegos durante a execução da aplicação.

Método Main completo

Definidos os métodos auxiliares, basta organizar as chamadas no método `Main`:

```
static void Main(string[] args)
{
    var connectionPolicy = new ConnectionPolicy
    {
        ConnectionMode = ConnectionMode.Gateway,
        ConnectionProtocol = Protocol.Https
    };

    using(
            var client = new DocumentClient(
            new Uri(_endpoint),
            _primaryKey,
            connectionPolicy)
        )
    {
        var db = CriarDatabase(client, "eventos").Result;

        var col = CriarColecao(client, db, "azure").Result;

        var evento = new Evento{
            Nome = "Azure Summit Brasil 2015",
            Participantes = new List<Participante>
            {
                new Participante{
                    Nome = "Thiago Custódio",
                    Email = "thiago.custodio@hotmail.com"
                }
            }
        };

        var result = CriarDocumento(client, col, evento).Result;

        ConsultaComLinq(client, col);

        ConsultaComSQL(client, col);
    }
}
```

9.5 CRIANDO STORED PROCEDURES, TRIGGERS E FUNÇÕES DEFINIDAS POR

USUÁRIOS

O DocumentDB, além de suportar a linguagem SQL, utiliza a linguagem JavaScript como sua T-SQL. Isto é, triggers, UDFs (*User Defined Functions*) e stored procedures são escritas com JavaScript.

Criando uma stored procedure com JavaScript

Vamos adicionar uma nova pasta à solução chamada `JS`. Nessa pasta, vamos adicionar dois arquivos JavaScript: `normalizarNomeEvento.js` e `validaInscricao.js`.

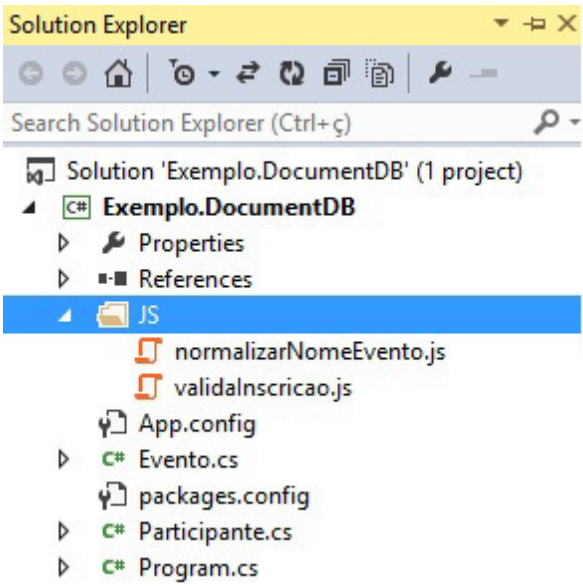

Figura 9.10: Exibição do Solution Explorer

Observação: a fim de evitar problemas relacionados ao caminho desses arquivos, vamos alterar o `Build Action` deles e setar a propriedade `Copy to Output Directory`:

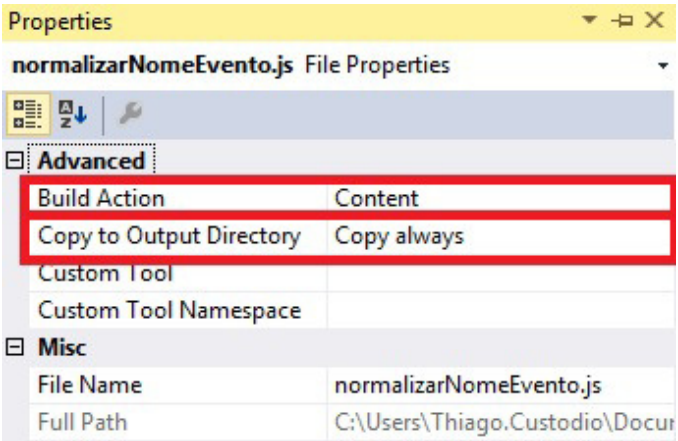

Figura 9.11: Alterando a propriedade Build Action e Copy to Output Directory dos arquivos estáticos

Feitos os devidos ajustes, precisamos apenas definir a lógica da stored procedure. Para demonstrar o funcionamento, adicionei ao arquivo `validaInscricao.js` uma lógica para validar se a inscrição ao evento foi realizada com sucesso:

```
//código da procedure
function validaEvento(nomeEvento, nomeParticipante)
{
    var context = getContext();
    var collection = context.getCollection();

    var query = "SELECT E.Nome FROM root E " +
                "join p in E.Participantes " +
                "where E.Nome  = '" + nomeEvento + "' " +
                "AND p.Nome = '" + nomeParticipante + "' ";

    collection.queryDocuments(collection.getSelfLink(),
       query, {},
       function (err, resposta, responseOptions)
       {
           if (err) throw new Error("Error" + err.message);

           if (resposta.length == 0)
                   throw "Inscrição não realizada";

           context.getResponse().setBody("OK");
       }
```

```
    );
}
```

Repare que a stored procedure espera dois parâmetros de entrada: o nome do evento e o nome do participante.

Em seguida, precisamos "anexar" a stored procedure à coleção:

```
private static async Task<ResourceResponse<StoredProcedure>>
        CriarStoredProcedure(DocumentClient client,
        DocumentCollection col)
{
    return await
        client.CreateStoredProcedureAsync
        (
            col.SelfLink,
            new StoredProcedure
            {
                Id = "validaInscricao",
                Body = File.ReadAllText(@"JS\validaInscricao.js")
            }
        );
}
```

Por último, basta efetuar uma chamada à stored procedure, passando os valores para os parâmetros de entrada:

```
private static async Task<StoredProcedureResponse<dynamic>>
        ExecutarStoredProcedure(DocumentClient client,
        ResourceResponse<StoredProcedure> proc)
{
    return await
        client.
            ExecuteStoredProcedureAsync<dynamic>(
                                proc.Resource.SelfLink,
            "Azure Summit Brasil 2015",  //nomeEvento
            "Thiago Custódio");  //nomeParticipante
}
```

Criando uma trigger com JavaScript

O procedimento para criar uma trigger é parecido com o de criação de stored procedure. O primeiro passo é definir o conteúdo do arquivo `normalizarNomeEvento.js` :

```
function normalizarNomeEvento()
{
    var item = getContext().getRequest().getBody();
    item.Nome = item.Nome.toUpperCase();
    getContext().getRequest().setBody(item);
}
```

A lógica anterior apenas transforma para caixa alta (`Uppercase`) o nome do evento. Definida a lógica da trigger, precisamos "anexar" a trigger à coleção:

```
private static async Task<ResourceResponse<Trigger>>
    CriarTrigger(DocumentClient client, DocumentCollection col)
{
    return await client.CreateTriggerAsync(col.SelfLink,
    new Trigger
    {
        TriggerType = TriggerType.Pre,
        Id = "normalizarNomeEvento.js",
        Body = File.ReadAllText(@"JS\normalizarNomeEvento.js"),
        TriggerOperation = TriggerOperation.All
    });
}
```

Nesse trecho, definimos os parâmetros `TriggerOperation` , com o qual informamos que a trigger poderá ser executada para operações de `Insert` , `Update` , `Delete` ou `Replace` ; e `TriggerType` , com o qual definimos que a lógica será executada antes das operações (`Insert` , `Update` , `Delete` e `Replace`).

Ao contrário dos bancos de dados relacionais, a trigger não será executada automaticamente; precisamos informar qual trigger será executada durante cada uma destas operações. Sendo assim, vamos definir um novo método para criar documentos, mas que execute essa trigger:

```
private static async Task<bool> CriarDocumentoComTrigger
    (
        DocumentClient client,
        DocumentCollection col,
        Evento evento
    )
{
```

```
    await client.CreateDocumentAsync
    (
        col.SelfLink,
        evento,
        new RequestOptions
        {
            PreTriggerInclude = new List<string>
            {
                "normalizarNomeEvento.js"
            },
        }
    );
    return true;
}
```

Para testar, basta adicionar um novo evento e passá-lo como parâmetro para esse método que acabamos de criar:

```
var eventoAzureConf = new Evento
{
    Nome = "azure conf 2015",
    Participantes = new List<Participante>
    {
        new Participante{
            Nome = "Thiago Custódio",
            Email = "thiago.custodio@hotmail.com"
        }
    }
};

var result = CriarDocumentoComTrigger(client, col,
                                      eventoAzureConf).Result;
```

Você pode acessar amostras de códigos e a documentação do serviço diretamente pelo portal:

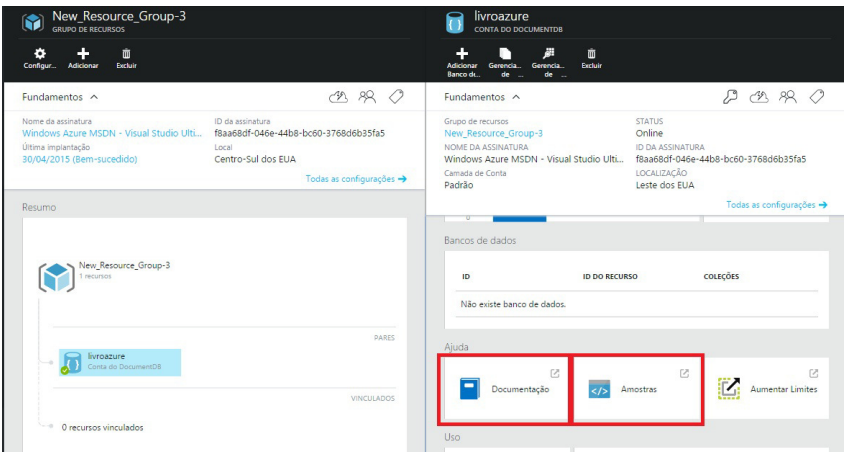

Figura 9.12: Links de exemplo e documentação diretamente pelo portal

9.6 CONCLUINDO

Neste capítulo, aprendemos como usar o DocumentDB. Considere utilizá-lo se você precisa de um *NoSQL as a Service* baseado em documentos para algum aplicativo que será hospedado no Azure.

> **MAIS SOBRE A ORIGEM DO NOSQL**
>
> Existe uma palestra muito bacana do Martin Fowler, na qual ele apresenta mais sobre o nascimento deste movimento e a origem do nome NoSQL. Confira em http://bit.ly/MartinFowlerNoSQL.

Para maiores informações sobre o DocumentDB e exemplos de seu uso com .NET, confira a página e a documentação oficial do produto nos links:

- Página do produto: http://aka.ms/documentdb.

- Documentação: http://aka.ms/documentdbdocs.
- Exemplos em .NET: http://aka.ms/documentdbsamples.

Caso você precise importar dados para o seu DocumentDB, utilize o projeto DocumentDB Import, que possui suporte a diversas fontes de dados como MongoDB, SQL Server, e arquivos `JSON` e `csv`. Saiba mais no Projeto DocumentDB Import (http://bit.ly/DocumentDBImport).

Capítulo 10

DADOS RELACIONAIS COM SQL SEVER

Vimos nos capítulos anteriores que o Azure oferece diversos serviços para armazenamento de dados não relacionais. Isso não quer dizer que não é possível trabalhar com banco de dados relacional, ou que eles deixarão de existir. Neste capítulo, vamos estudar como trabalhar com dados relacionais.

No Azure é possível trabalhar com SQL Server de duas maneiras: **IaaS** (Infraestrutura como Serviço), em que você é o responsável pelo banco de dados e pela máquina virtual; e **DaaS** (Database como Serviço), em que você apenas cria tabelas e manipula os dados, sendo que a administração, disponibilidade e escalabilidade ficam por conta da Microsoft.

10.1 SQL DATABASE – DATABASE COMO SERVIÇO

SQL Database é um banco de dados fornecido pela Microsoft baseado no SQL Sever, ou melhor dizendo, um SQL Server customizado. Ele é ideal se você não deseja lidar com as questões relacionadas à infraestrutura; você simplesmente cria um banco diretamente pelo portal, obtém a string de conexão e o utiliza.

Internamente, esse SQL Database trabalha com o protocolo *Tabular Data Stream* (TDS), que é o mesmo usado pelo SQL Server,

sendo assim, você pode usá-lo da mesma forma. No entanto, você não terá acesso aos recursos de administração (backup, mirror etc.), dado que essa parte fica à cargo da Microsoft (lembrando de que esse modo é um banco de dados como serviço).

E como ficam as questões de manutenção, disponibilidade e escalabilidade?

Dado que esse modo é um banco de dados como serviço, você não tem de se preocupar com a aplicação de *patches* de segurança, ou atualização de software e hardware. Um SQL Database sempre mantém duas cópias do seu banco de dados automaticamente para garantir a disponibilidade, e um acordo de nível de serviço (SLA) de 99.9%.

Caso você identifique que o banco de dados é o gargalo na sua aplicação, você pode escalar horizontalmente, isto é, adicionando mais servidores para atender à demanda e evitando gargalos no sistema.

Pago conforme o uso

Vale lembrar de que haverá um aumento no custo, dado que, no modelo de computação em nuvem, você paga pelo que você utilizou e apenas durante o tempo de utilização.

Outras diferenças em relação ao SQL Server

Embora exista a compatibilidade entre SQL Server e SQL Database, o SQL Database não suporta tipos da .NET CLR, nem queries distribuídas, além dos recursos de administração citados anteriormente.

Criando um SQL Database

Autentique-se no Portal de Gerenciamento do Azure, em http://manage.windowsazure.com.

Localize e clique no menu `Novo` no canto inferior à esquerda e, em seguida, selecione a opção: `Data Services` (Serviço de dados) -> `SQL Database` (Banco de dados SQL) -> `Quick Create` (Criação Rápida).

Figura 10.1: Criando um novo SQL Database

Depois, informe o nome do banco de dados. Por meio desse nome, será criado um *endpoint* para que você possa conectar ao banco de dados.

Selecione a assinatura (caso você possua mais de uma) e, em seguida, selecione a opção `Novo servidor do banco de dados SQL`.

Selecione o data center, e informe um usuário e senha que será usado para estabelecer a conexão com o seu banco de dados:

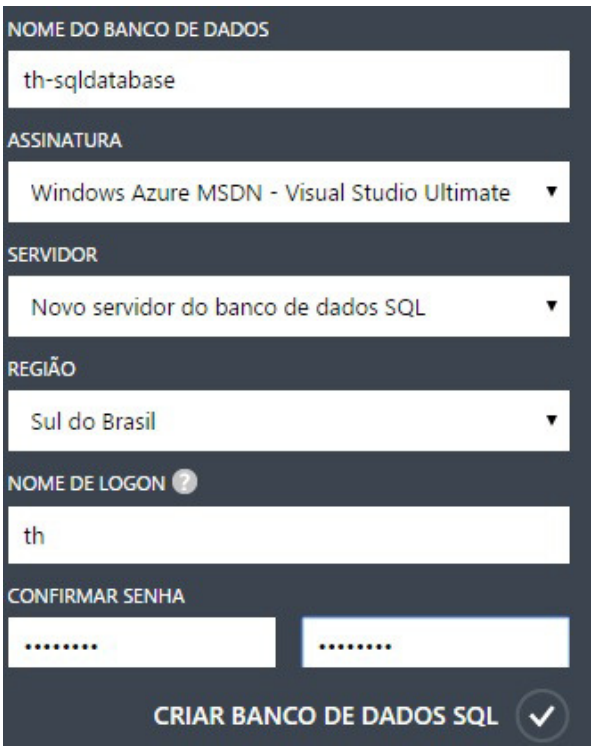

Figura 10.2: Dados para a criação do SQL Database

Por último, basta clicar na opção `CRIAR BANCO DE DADOS SQL` . Em poucos segundos, será disponibilizado um banco de dados para que você possa utilizar. Repare que, no menu lateral à esquerda, há uma opção para rápido acesso aos SQL Database:

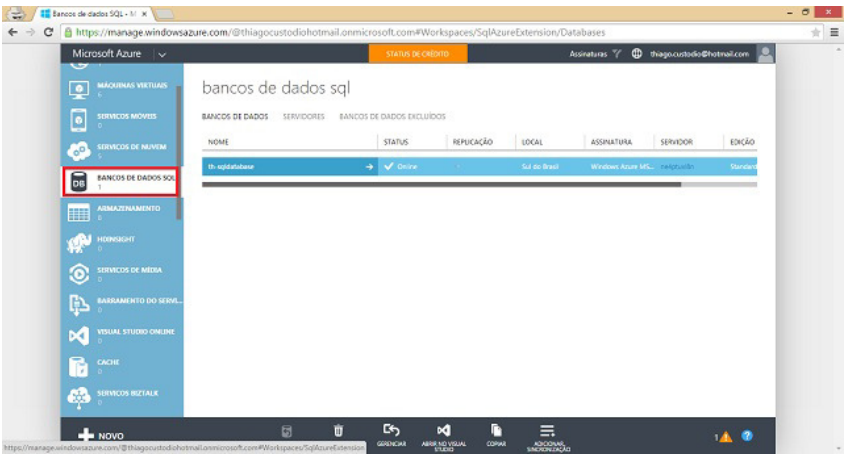

Figura 10.3: Menu de acesso rápido aos SQL Databases

Para acessá-lo, é preciso configurar o firewall para liberar acesso ao seu endereço IP. Apenas para testes, vamos liberar uma larga faixa de IP, para que não ocorra nenhum problema em relação às regras de firewall. Localize a seção `SERVIDORES` :

Figura 10.4: Menu de acesso aos servidores SQL Database

Depois, clique sobre o nome do servidor:

Figura 10.5: Seleção do servidor de banco de dados

Na tela à seguir, clique sobre a opção `CONFIGURAR` :

Figura 10.6: Configurando regras ao firewall do servidor

Nessa tela, o próprio Azure já informa o seu endereço de IP para configurar o acesso ao SQL Database. No momento em que escrevo este capítulo, meu endereço de IP é o 191.188.47.94, como mostra a figura.

Clicando sobre o botão `ADICIONAR A TODOS OS ENDEREÇOS IP PERMITIDOS` , uma nova linha é adicionada representando a regra que será adicionada ao firewall:

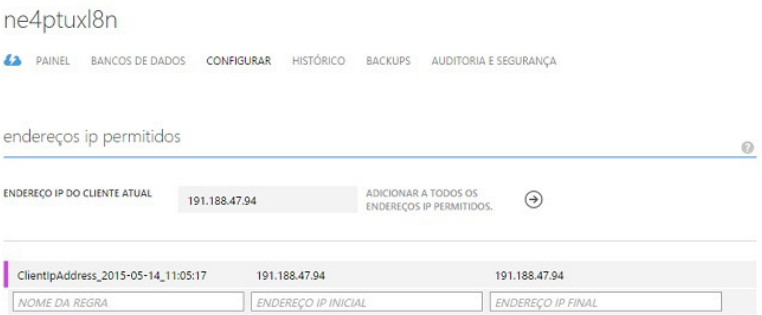

Figura 10.7: Regra configurada no firewall do servidor

Basta clicar sobre o botão `SALVAR` na barra inferior da tela para que essa regra de firewall seja configurada.

Figura 10.8: Salvando as regras adicionadas

Vale lembrar de que os endereços IPs mudam constantemente, sendo assim, você precisará adicionar o seu endereço de IP a lista de endereços permitidos repetidamente. Uma outra opção é liberar um *range* de IP, por exemplo: 0.0.0.0 até 255.255.255.255; no entanto, não é uma boa prática de segurança.

Por último, precisamos obter a string de conexão para efetuar o acesso ao SQL Database. Ela pode ser obtida na seção `Painel`:

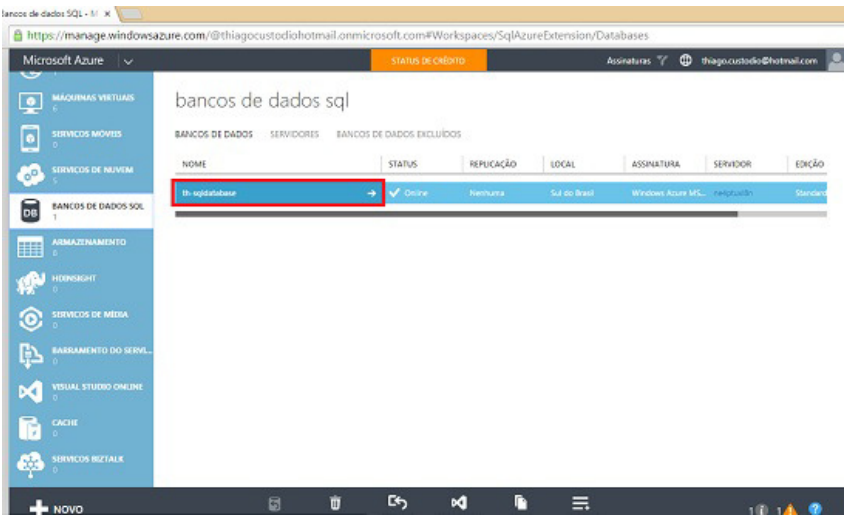

Figura 10.9: Selecionando o SQL Database

Figura 10.10: Acessando o menu Painel do SQL Database selecionado

No menu lateral à direita, localize e clique sobre a opção `Mostrar Cadeias de Conexão`:

Figura 10.11: Exibição das strings de conexão

O Azure exibirá um *pop-up* com string de conexões para .NET, PHP, Java ou ODBC (*Open Database Connectivity*). Use a string que melhor se adequa às suas necessidades.

Para criar as tabelas, basta se conectar ao SQL Database usando a string de conexão via SQL Management Studio, ou usando algum OR/M como o Entity Framework.

Acesso diretamente pelo Azure

Essa regra de firewall é necessária apenas se a origem do acesso for externa aos data centers do Azure. Se o acesso partir de um Azure WebApp, por exemplo, não será necessário configurar endereços de IP para acessar o SQL Database.

10.2 SQL SERVER – INFRAESTRUTURA COMO SERVIÇO

Utilizando o SQL Server no modo IaaS, você pode criar uma máquina virtual do zero e instalar o SQL Server com a sua licença, ou utilizar um template disponível na galeria de imagens de máquinas virtuais, que já disponibiliza um SQL Server instalado.

A vantagem de usar um template da galeria com SQL Server é que o custo da licença já está embutido no valor a ser pago pelo tempo de utilização da máquina virtual.

Criando uma máquina virtual com um template da galeria de imagens

Autentique-se no Portal de Gerenciamento do Azure, em http://manage.windowsazure.com.

Localize e clique no menu `Novo` no canto inferior à esquerda e, em seguida, selecione a opção: `Compute` (Computação) -> `Virtual Machine` (Máquina Virtual) -> `From Gallery` (Da Galeria).

Figura 10.12: Criando uma nova máquina virtual da galeria

Repare que há um menu lateral à esquerda e existem diversos templates disponíveis, com diversos produtos previamente instalados e configurados. Já existe um template com Visual Studio 2015, outro com diversas versões do SQL Server 2014, Biztalk Server, entre outros.

Repare também que não são apenas produtos Microsoft, existem templates com o banco de dados Oracle instalado e diversas distribuições do sistema operacional Linux. Isso comprova que a plataforma é aberta e você não fica preso a tecnologias e produtos Microsoft.

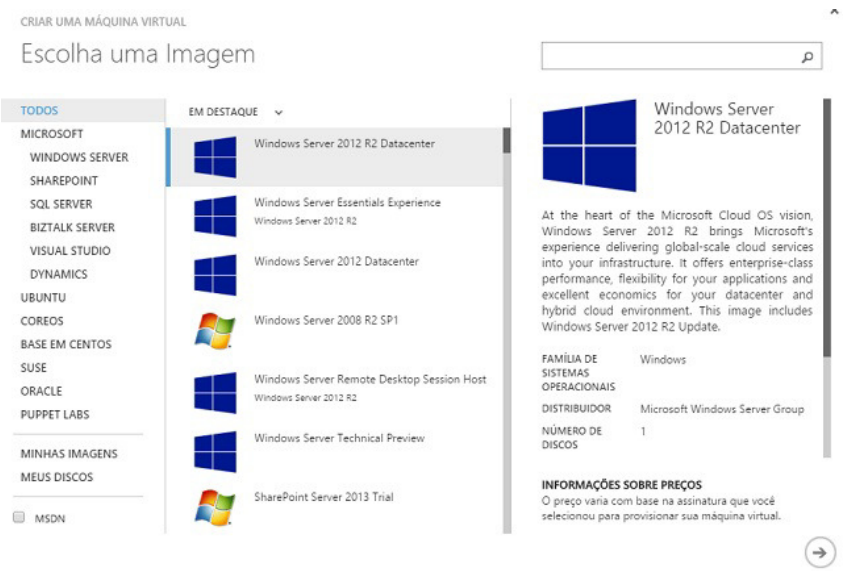

Figura 10.13: Seleção do template da máquina virtual

Selecione no menu a opção `SQL Server` e, depois, a versão de sua preferência. Então, clique na seta para avançar. Para demonstrar a criação, vou selecionar a versão `SQL Server 2012 SP2 Enterprise`.

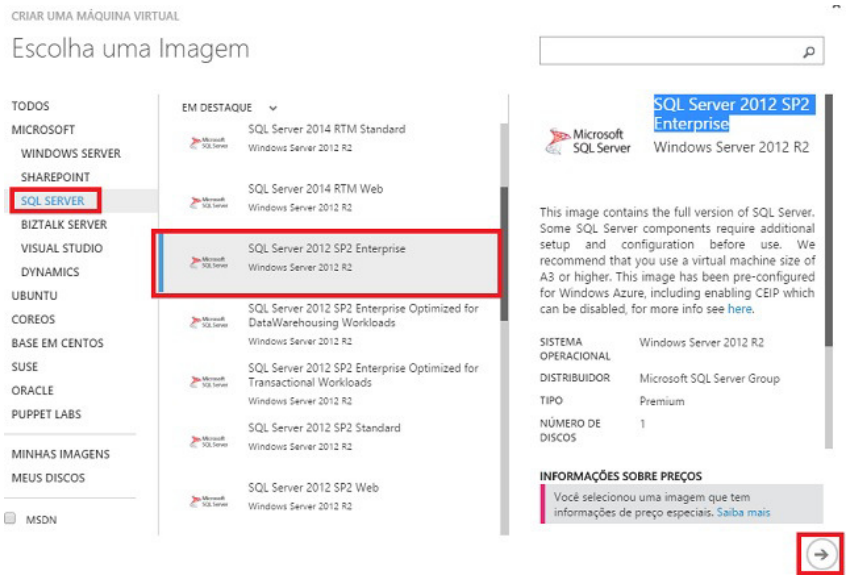

Figura 10.14: Escolha do template da máquina virtual

Após selecionar a versão do SQL Server, precisamos informar o nome da máquina virtual, o tamanho na instância (CPU e memória), um nome de usuário e uma senha para nos autenticarmos nessa máquina virtual:

Figura 10.15: Informando o nome da máquina virtual, tamanho da instância, usuário e senha para autenticação

Na tela à seguir, podemos informar diversas configurações para a máquina virtual. A primeira configuração é informar um serviço de nuvem existente, ou criar um novo.

Observação: um serviço de nuvem é um "agrupador".

Vou manter o nome do serviço de nuvem com o mesmo nome da máquina virtual.

Em seguida, podemos escolher qual assinatura (*subscription*) será usada para criar essa máquina virtual. No meu caso, tenho várias assinaturas disponíveis, pois pertencem a clientes, benefício MVP e assinatura MSDN.

Na próxima configuração, devemos escolher em qual data center a máquina virtual será criada. Selecione a opção `Sul do Brasil`, pois representa o data center disponível na região de São Paulo.

Em seguida, podemos usar uma conta de armazenamento já criada, ou solicitar que uma nova seja criada neste momento. A conta de armazenamento servirá para armazenar o disco da máquina virtual.

Por fim, existe a possibilidade de criar um novo conjunto de disponibilidade, ou utilizar algum que já exista. Um conjunto de disponibilidade serve para indicar para o Azure não alocar as máquinas no mesmo servidor físico, evitando, assim, que uma possível falha de hardware afete todo o ambiente.

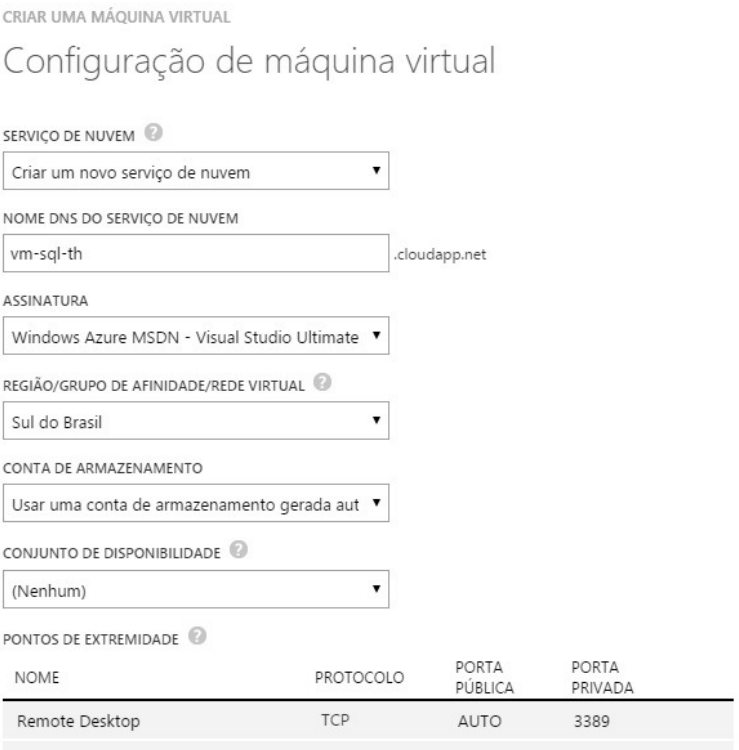

Figura 10.16: Configuração do serviço de nuvem

Logo na sequência, existe uma seção para configurar os pontos de extremidade (*endpoints*). Eles servem para liberar acesso a determinadas portas. Vamos liberar acesso à porta `80`, que permite acesso à internet, e à `1433`, utilizada pelo SQL Server.

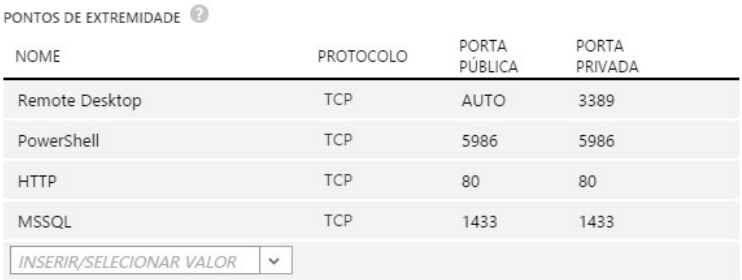

Figura 10.17: Configurando endpoints à máquina virtual

Na última parte do assistente para criação de máquina virtual, existem diversas extensões que podem ser instaladas na máquina que permitem uma agilidade na sua configuração, utilizando frameworks (como Puppet e Chef, por exemplo).

Deixe apenas selecionada a opção `Instalar o Agente de VM` e clique em avançar para que a sua máquina virtual com SQL Server seja provisionada. Ao término do provisionamento, basta clicar sob a opção `Painel`, depois em `CONECTAR` no menu inferior para efetuar o download do arquivo `.rdp`.

Abra o arquivo `.rdp`, e informe o usuário e a senha para se autenticar na máquina virtual.

Figura 10.18: Instalando agentes na máquina virtual

USANDO PUPPET NO AZURE

Gravei um vídeo em que demonstro o uso do Puppet para configurar uma máquina virtual no Azure, que se encontra disponível em http://bit.ly/PuppetNoAzure.

10.3 CONCLUINDO

Grandes pensadores da nossa área acreditam que, nesta década e nas próximas, trabalharemos com um modelo de persistência

poliglota, isto é, identificar qual é o melhor meio para armazenar a informação, em determinada área do sistema. Por exemplo:

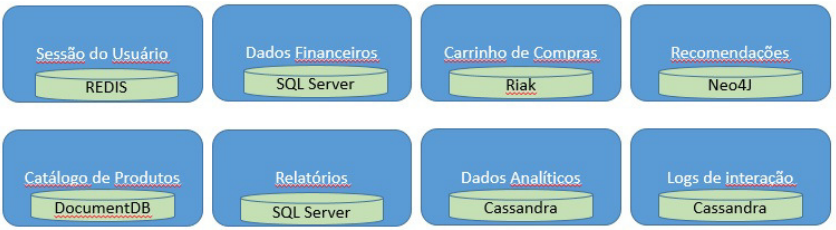

Figura 10.19: Exemplo de persistência poliglota

Persistência poliglota

Para maiores informações sobre persistência poliglota e sobre os outros tipos de NoSQL, recomendo a leitura do livro *NoSQL destilado*, de Martin Fowler.

Ao longo deste livro, vimos diversos meios para armazenamento de dados. Recomendo que você se aprofunde em persistência poliglota e saiba reconhecer o melhor meio de persistência para o seu cenário em específico, seja usando NoSQL, banco de dados relacional SQL Server ou SQL Database.

Capítulo 11

UM POUCO MAIS SOBRE MÁQUINAS VIRTUAIS

No capítulo sobre Azure WebApps, vimos que é possível publicar uma aplicação web facilmente e de diversas maneiras para o Azure. Mas talvez você precise de um pouco mais de liberdade em relação ao ambiente, por exemplo, sua aplicação utiliza algum componente de terceiros para renderizar gráficos, ou realizar conversão de arquivos de um formato para outro.

Para que o componente funcione corretamente, às vezes é necessário registrar `DLL` s no Global Assembly Cache (GAC), ou criar registros no Windows. Esse tipo de liberdade não é possível utilizando o modo WebApps.

No entanto, com o uso de máquinas virtuais, você passa a usar o Azure como infraestrutura (IaaS) e tem total liberdade para escolher desde o sistema operacional até a maneira como os dados serão armazenados. Em outras palavras, você utiliza apenas servidores físicos, cabeamento, internet e firewall do Azure. Todo o resto fica por sua conta, ou seja, escolher e manter o sistema operacional, adicionar discos extras à máquina virtual, instalar o *runtime*, publicar e configurar a aplicação etc.

No capítulo anterior, vimos como criar uma máquina virtual à partir de um template da galeria que já possui o SQL Server instalado e com uma licença do software aplicada. Neste capítulo,

veremos como criar uma máquina virtual com o sistema operacional Ubuntu (distribuição do Linux).

11.1 CRIANDO UMA MÁQUINA VIRTUAL COM UBUNTU

Autentique-se no Portal de Gerenciamento do Azure, em http://manage.windowsazure.com.

Localize e clique no menu `Novo` no canto inferior à esquerda e, em seguida, selecione a opção `Compute` (Computação) -> `Virtual Machine` (Máquina Virtual) -> `From Gallery` (Da Galeria).

Selecione no menu à esquerda a seção `Ubuntu` e, depois, a opção `Ubuntu 12.04 LTS`:

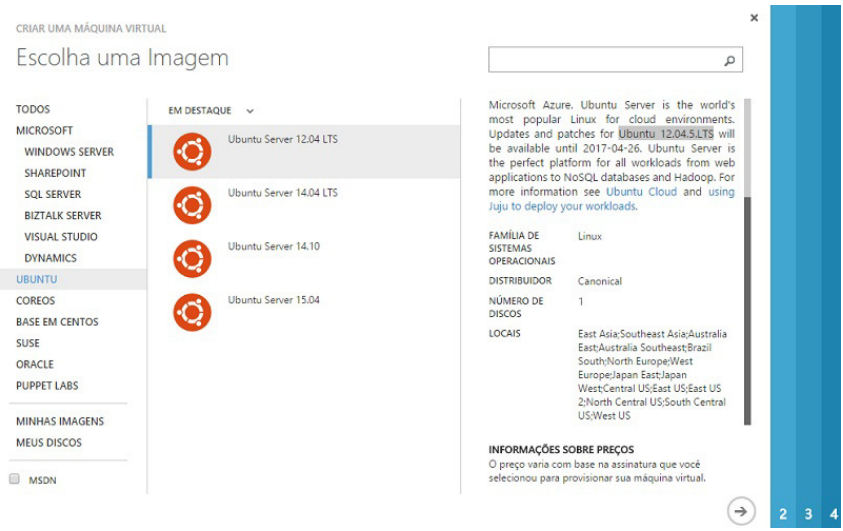

Figura 11.1: Seleção do template da máquina virtual

Na tela seguinte, informe o nome da máquina virtual e selecione o tamanho da instância. Para se autenticar, você pode usar uma chave SSH, ou usar o padrão usuário e senha, desmarcando o

checkbox `Carregar Uma Chave Ssh Compatível Para Autenticação` e marcando o checkbox `Fornecer Uma Senha`:

Figura 11.2: Informando o nome de usuário e senha para autenticação"

Em seguida, podemos escolher se essa máquina virtual ficará em um *cloud service* já existente, ou se vamos criar um novo. Selecione a assinatura, o data center e a opção para gerar uma nova conta de armazenamento. Configure também um novo *endpoint* para a porta `80`:

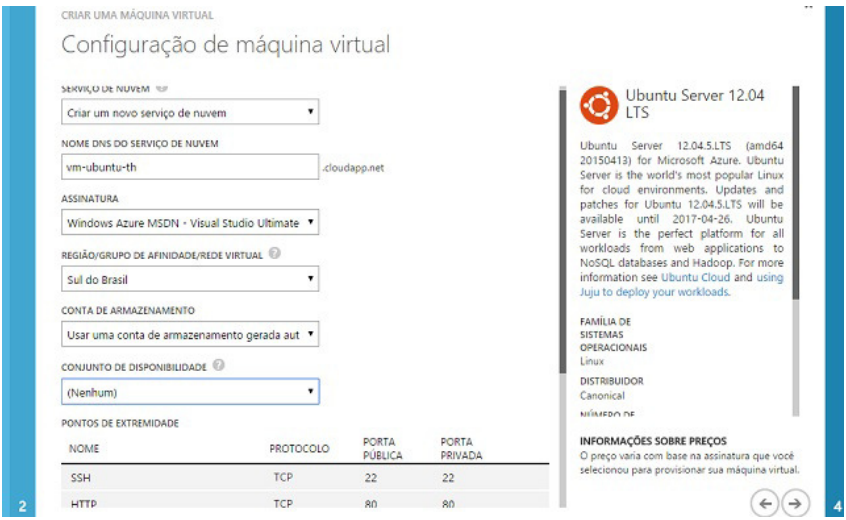

Figura 11.3: Criando o serviço de nuvem

Por fim, podemos escolher algumas extensões para facilitar a administração e a configuração dessa máquina virtual. Mantenha a opção `Instalar o Agente de VM` selecionada, e clique no botão para iniciar o provisionamento dessa máquina virtual:

Figura 11.4: Instalando agente de máquina virtual

Para se conectar a uma máquina virtual Linux, você precisará de um cliente SSH. Costumo utilizar o Putty por ser gratuito e pela facilidade. Faça o download em www.putty.org.

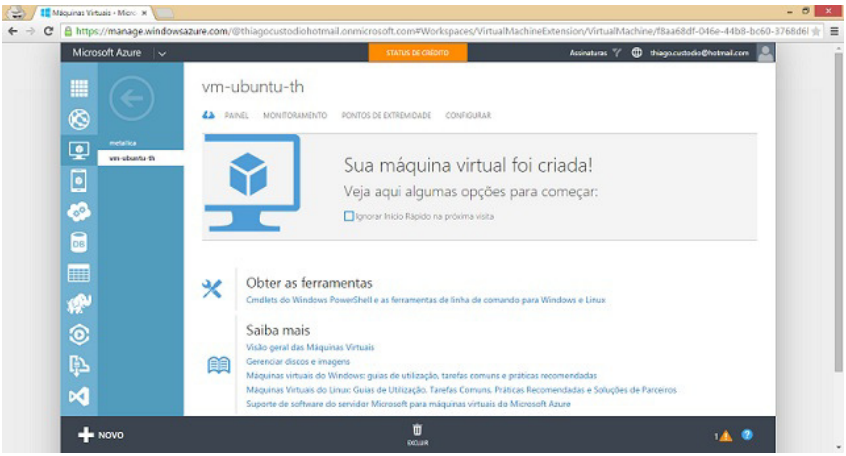

Figura 11.5: Menu painel da máquina virtual criada

Após a criação da máquina virtual, selecione a opção `Painel` e localize, no menu à direita, o `NOME DNS`, ele será utilizado para a conexão via Putty.

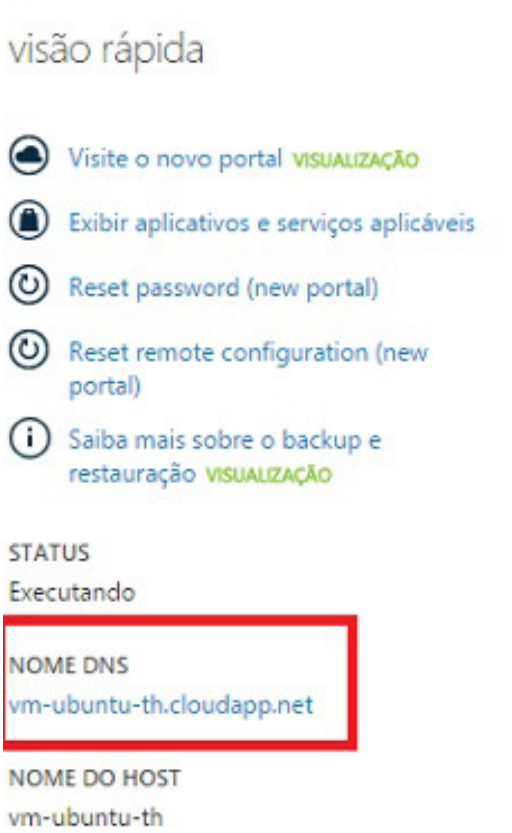

Figura 11.6: Visualizando informações da máquina virtual criada

Digite, ou copie e cole, o `NOME DNS` no campo `Host Name` (ou `IP address`) do Putty:

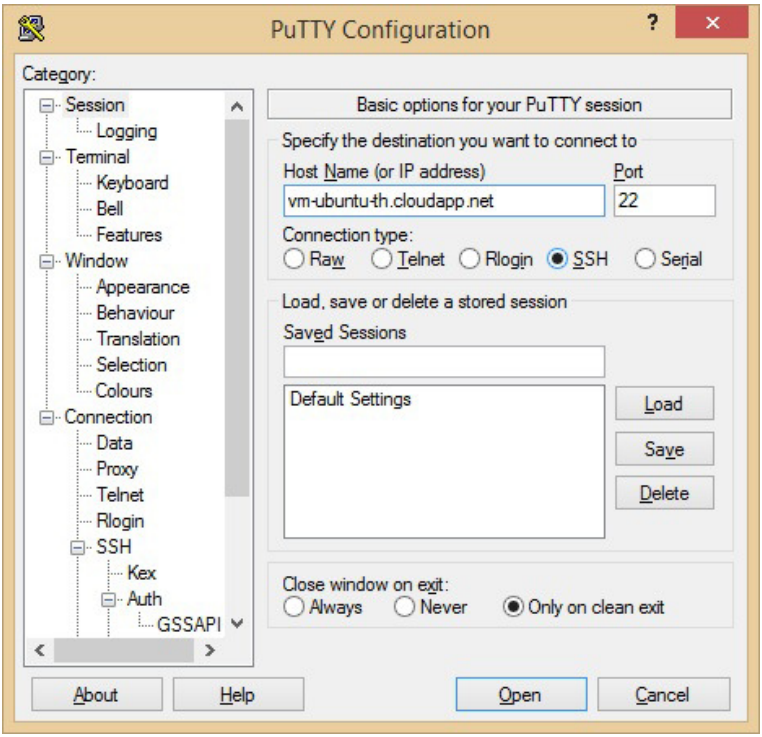

Figura 11.7: Conectando via Putty

Em seguida, basta clicar na opção `Open` para iniciar o acesso remoto à máquina virtual com Ubuntu Server que acabamos de criar. Será exibida uma mensagem de segurança informando que não há um registro da chave de acesso. Basta clicar sobre a opção `Yes`:

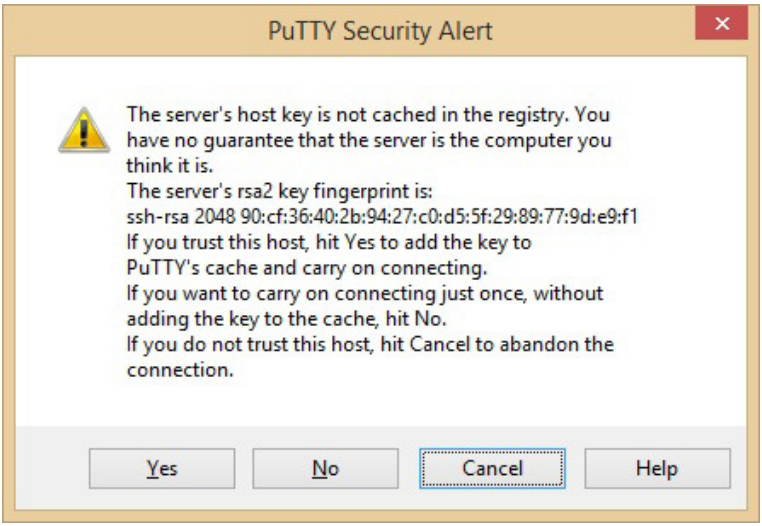

Figura 11.8: Alerta de segurança do cliente SSH Putty

Logo após, será exibido o prompt de comando do Linux, solicitando pelo usuário e senha para acesso. Utilize o usuário e senha informados durante a criação da máquina virtual:

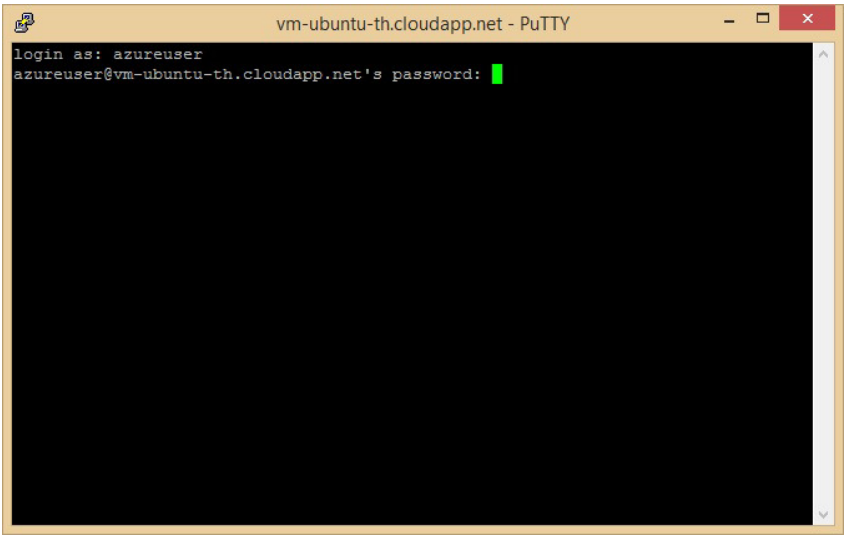

Figura 11.9: Terminal do Linux

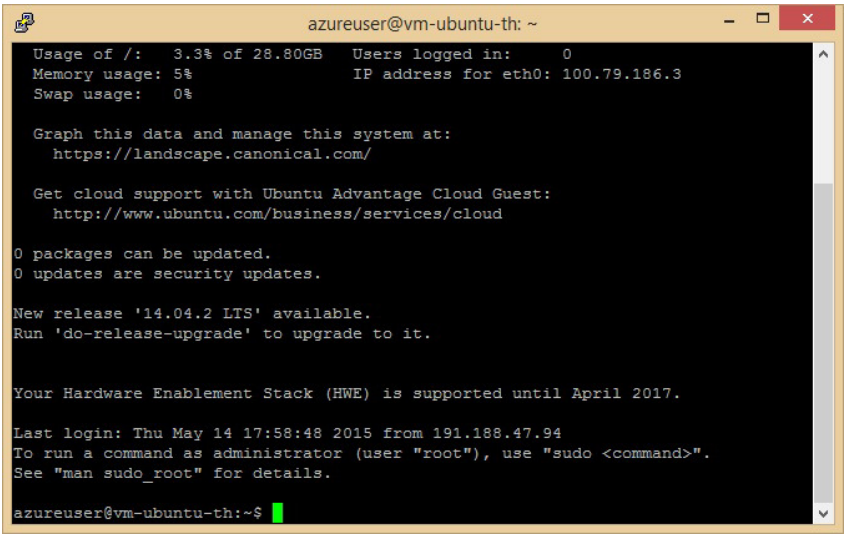

Figura 11.10: Usuário autenticado com sucesso

11.2 CONECTANDO DATA CENTERS, SERVIÇOS E MÁQUINAS COM REDES VIRTUAIS

Você pode usar redes virtuais no Microsoft Azure para:

1. Conectar serviços e websites criados no Azure utilizando uma rede; desta maneira, a comunicação entre eles será feita ao usar essa rede.

2. Estender seu data center para o Azure criando uma VPN *site-to-site*. Desta maneira, serviços hospedados na nuvem pública da Microsoft (Azure) poderão acessar servidores e serviços que rodam no seu data center através de uma rede segura (nuvem híbrida).

3. Conectar um único computador a uma rede no Microsoft Azure utilizando uma VPN *point-to-site*.

Em alguns cenários, será necessário conhecer o endereço IP de determinada máquina virtual, como por exemplo um servidor controlador de domínio. Colocando esse servidor em uma rede virtual, você pode reservar um único endereço IP para essa máquina, mesmo que ela reinicie.

Criando uma rede virtual no Azure

Autentique-se no Portal de Gerenciamento do Azure, em http://manage.windowsazure.com.

Localize e clique no menu `Novo` no canto inferior à esquerda e, em seguida, selecione a opção `Network Services` (Serviços De Rede) -> `Virtual Network` (Rede Virtual) -> `Custom Create` (Criação Personalizada).

Informe o nome da rede virtual, e selecione o data center e a assinatura:

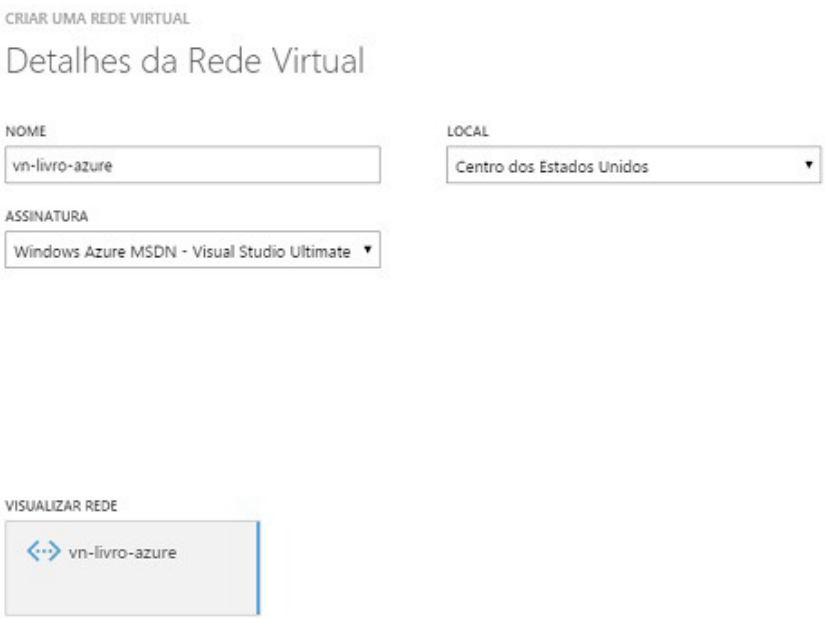

Figura 11.11: Criando uma nova rede virtual

Na tela seguinte, você pode configurar os servidores DNS para essa rede, e criar uma VPN site-to-site ou point-to-site. Clique em avançar e, depois, você pode escolher quantos endereços IPs estarão nessa rede virtual. Selecione a opção `/24` que permite até 256 endereços IP. Por fim, selecione a opção para criar a rede virtual.

Figura 11.12: Escolha do range de IPs da rede virtual

Depois de criada a rede virtual, você poderá informar que determinada máquina virtual fará parte dessa rede durante a sua criação:

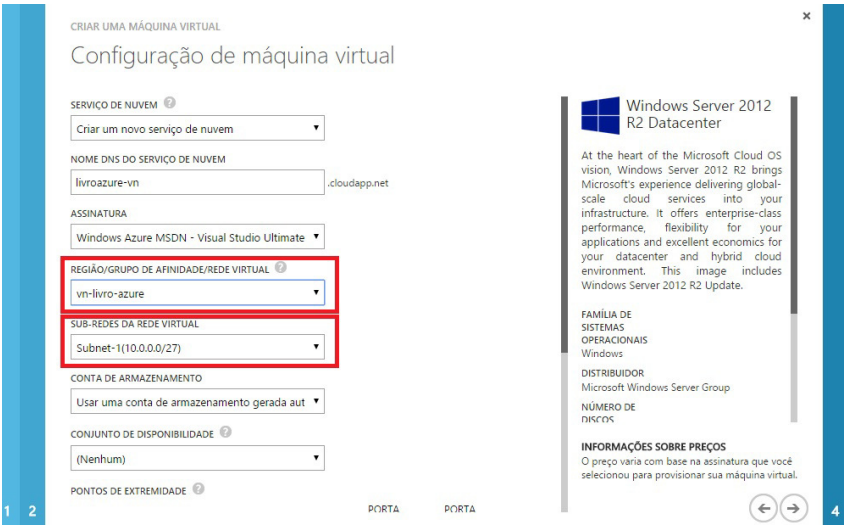

Figura 11.13: Adicionando uma máquina virtual à rede virtual criada

Pode acontecer de você precisar adicionar uma máquina virtual já existente a uma rede virtual do Azure. Até o momento em que escrevo este livro, existem duas opções para realizar essa tarefa:

1. Acionar o suporte do Azure;
2. Excluir a máquina virtual (mantendo os discos). Em seguida, recrie a máquina virtual, mas a partir do disco já existente. Para maiores informações, consulte http://bit.ly/AdicionarVMVirtualNetwork.

As duas opções são viáveis, tudo vai depender do nível de permissão que você possui na assinatura do Azure. Existem casos em que o cliente vai lhe conceder acesso à assinatura dele e, provavelmente, você não terá permissões para realizar a segunda opção. Sendo assim, recorra ao suporte do Azure que tem um excelente tempo de resposta.

11.3 CONCLUINDO

Neste breve capítulo, vimos como criar e conectar a uma máquina virtual com o sistema operacional Linux, distribuição Ubuntu Server. Também aprendemos como criar uma rede virtual para conectar serviços e websites e/ou para estabelecer uma conexão segura com o seu data center (VPN).

Capítulo 12

CONSIDERAÇÕES FINAIS

A plataforma Microsoft Azure não é uma aposta, mas sim uma realidade. Basta ver a quantidade de data centers que já estão em operação ao redor do globo. A quantia é superior à soma dos dois principais concorrentes.

Neste livro, meu intuito foi demonstrar algumas das features disponíveis na plataforma. Existem diversos outros assuntos que gostaria de escrever aqui, como integração utilizando Azure Service Bus, consumo de WebApis com Azure App Service, entre muitos outros. No entanto, seriam necessárias muitas outras páginas e, dada a velocidade com que novas features são adicionadas à plataforma, eu correria um sério risco de ter um capítulo desatualizado antes mesmo da publicação deste livro.

Como dito anteriormente, o Azure evolui em uma velocidade que é difícil acompanhar. Caso ocorra alguma mudança em relação aos conteúdos aqui abordados, escreverei as atualizações para a publicação de uma versão deste livro.

Espero que este livro seja o pontapé inicial para que você utilize a plataforma de nuvem da Microsoft e que ele lhe ajude de alguma maneira. Mesmo que você não use o Azure em produção, você pode utilizá-lo como ambiente para desenvolvimento e testes.

Se você sentiu falta de algum assunto ou ficou com dúvidas sobre algo abordado no livro, entre em contato comigo via Twitter

(@thdotnet) ou e-mail (thiago.custodio@hotmail.com).

Você também pode tirar suas dúvidas pela lista de discussão do livro, em http://forum.casadocodigo.com.br/.