UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

INTEGRAÇÃO DE APLICAÇÕES E SERVIÇOS UTILIZANDO COMPUTAÇÃO NA NUVEM COM A PLATAFORMA MICROSOFT WINDOWS AZURE

RICARDO LINHARES

BLUMENAU
2011

2011/2-24

RICARDO LINHARES

INTEGRAÇÃO DE APLICAÇÕES E SERVIÇOS UTILIZANDO COMPUTAÇÃO NA NUVEM COM A PLATAFORMA MICROSOFT WINDOWS AZURE

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Paulo Fernando da Silva, Mestre - Orientador

BLUMENAU
2011

2011/2-24

INTEGRAÇÃO DE APLICAÇÕES E SERVIÇOS UTILIZANDO COMPUTAÇÃO NA NUVEM COM A PLATAFORMA MICROSOFT WINDOWS AZURE

Por

RICARDO LINHARES

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Paulo Fernando da Silva, Mestre – Orientador, FURB.
	_____
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB.
	_____
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB

Blumenau, 30 de janeiro de 2012

Dedico este trabalho a todos que colocaram fé em minha pessoa.

AGRADECIMENTOS

À minha família, que sempre esteve presente.

A minha namorada Bruna, por sua confiança e por acreditar em minha competência.

Ao Rock N Roll, que a cada dia me fortalece e me faz sentir mais jovem.

Ao time Microsoft, por suas ajudas em fóruns e materiais didáticos de grande valia.

Ao meu orientador, Paulo Fernando da Silva, por ter acreditado na conclusão deste trabalho.

O medo é uma cena que a mim não me assiste.

Hélio

RESUMO

Computação na nuvem é um termo que está se tornando comum nos meios empresariais, sendo um assunto fortemente discutido na busca de novas tecnologias para computação, escalabilidade e disponibilidade. Para suprir estas necessidades, a Microsoft criou a plataforma Azure, um serviço que vende computação e cobra de acordo com seu uso. O objetivo deste trabalho é demonstrar um protótipo de testes de escalabilidade utilizando a plataforma Microsoft Windows Azure.

Palavras-chave: Computação na nuvem. Escalabilidade. Azure.

ABSTRACT

Cloud computing is a term that is becoming common in business circles, being a strongly debated issue in search of new technologies for computing, scalability and availability. To meet these needs, Microsoft created the Azure, a service that sells computer and charges according to their use. The objective of this study is to demonstrate a prototype of scalability tests using the Microsoft Windows Azure.

Key-words: Cloud computing. Scalability. Azure.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ASP – *Active Server Page*

FTP – *File Transport Protocol*

HTTP - *HyperText Transfer Protocol*

HTTPS – *HyperText Transfer Protocol Secure*

IaaS – *Infrastructure as a Service*

IIS – *Internet Information Service*

IPC – *Inter-Process Communication*

MB – *Mega Bytes*

PaaS – *Platform as a Service*

REST - *Representational State Transfer*

SaaS – *Software as a Service*

SDK – *Software Development Kit*

SOA – *Service Oriented Architecture*

SSL – *Secure Socket Layer*

TCP – *Transmission Control Protocol*

UML – *Unified Modeling Language;*

WCF – *Windows Communication Foundation*

WSDL – *Web Services Description Language*

XML - *eXtensible Markup Language*

LISTA DE SÍMBOLOS

% - por cento

SUMÁRIO

1 INTRODUÇÃO

A constante evolução das tecnologias e o aumento de complexidade dos sistemas, com consequente incremento nas necessidades de recursos de armazenamento e processamento, contribuem para a crescente busca de novos meios para disponibilizar serviços e aplicações.

Com o advento da internet, as integrações e interoperabilidade entre softwares tornaram – se populares, criando desta maneira, uma gigantesca plataforma para hospedagem de serviços.

Segundo Hirshman et al. (2007, p. 2), no início da era da internet, *sites* web simples e estáticos eram populares mas relativamente desconhecidos do grande público. A infraestrutura construída para dar suporte a este movimento era igualmente básica e concentrada mais em conseguir o máximo possível de clientes do que fornecer serviços de alta tecnologia, como aplicativos interativos e alta disponibilidade. A infraestrutura era normalmente composta de um único servidor web independente, com serviços de *File Transport Protocol* (FTP), executando *Internet Information Services* (IIS)[1] com conteúdo hospedado localmente no servidor.

> A principal meta do projeto de uma plataforma hospedada é otimizar escalabilidade, disponibilidade, densidade e paridade de preço e adotar, ao mesmo tempo, um nível de segurança tão granular quanto possível, com isolamentos dos clientes entre si. (HIRSHMAN et al., 2007, p. 5).

Diante destes cenários, com a meta de atingir um nível otimizado na prestação de serviços, depara-se com a plataforma da Microsoft desenvolvida para computação na nuvem, o Windows Azure. A plataforma Azure utiliza a computação na nuvem. Sosinsky (2011, p. 3) detalha que a computação na nuvem refere-se a aplicativos que são executados pela internet, utilizando virtualização[2], abstraindo os recursos físicos para o usuário.

Segundo Skonnard e Brown (2009), o Windows Azure fornece recursos básicos de sistema operacional, teoricamente sem limites. A escalabilidade horizontal seria apenas uma questão de configuração, evitando assim as complexidades da Tecnologia da Informação (TI) relacionadas ao provisionamento e configuração de recursos, gerenciamento de software e gerenciamento de servidores físicos.

> A plataforma Windows Azure está prestes a mudar radicalmente o modo como os

[1] IIS é um conjunto integrado de serviços para disponibilizar conteúdo Web, ou seja, um servidor de aplicação Web da Microsoft.

[2] Virtualização é a emulação de uma ou mais estações ou servidores, dentro de um único computador físico (MENKEN; BLOKDIJK, 2008, p. 7).

> arquitetos e desenvolvedores da plataforma Microsoft pensam a respeito da construção e do gerenciamento de aplicações. A plataforma Windows Azure [...] fornece um ambiente de computação de nuvem baseado na Internet para executar aplicações e armazenar dados em data centers da Microsoft localizados em todo o mundo. Você pode pensar nela como o Windows na nuvem. (SKONNARD, BROWN, 2009).

Diante deste cenário, o presente trabalho propõe o desenvolvimento de um protótipo de testes de escalabilidade que utilizará a computação na nuvem, buscando escalabilidade e alta disponibilidade, abstraindo a estrutura de hardware e exibindo o poder da computação na nuvem com a plataforma Microsoft Windows Azure.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo que utilize a plataforma Windows Azure para efetuar a computação na nuvem.

Os objetivos específicos do trabalho são:

a) desenvolver um serviço utilizando *Windows Communication Foundation* (WCF) e disponibilizá-lo como um *web service* no Windows Azure;
b) desenvolver um protótipo que efetue execução de algoritmos matemáticos e algoritmos de manipulação de grafos a fim de efetuar alto consumo de memória e de processamento;
c) identificar as vantagens de manter um serviço ou aplicação no ambiente do Windows Azure.

1.2 ESTRUTURA DO TRABALHO

No capítulo dois é descrita a fundamentação teórica da pesquisa. O capítulo três aborda a especificação do protótipo, aspectos técnicos e de desenvolvimento, aplicando a teoria apresentada no capítulo anterior. Por fim, no capítulo quatro, conclui-se a pesquisa, levando em consideração os objetivos propostos e sugestões para pesquisas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

No presente capítulo é apresentada a fundamentação teórica na qual se baseia o desenvolvimento do protótipo de teste de escalabilidade. São abordadas as características de um ambiente na nuvem, descrevendo o conceito de computação na nuvem, e como ela é consumida e vendida, através de modelos de serviços. Para o desenvolvimento de uma aplicação e/ou serviço na nuvem, será detalhada a plataforma na nuvem da Microsoft, o Windows Azure. Também serão detalhadas tecnologias que serão utilizadas para o desenvolvimento do protótipo de teste de escalabilidade, como a biblioteca de comunicação WCF e Azure Storage, um espaço de dados disponibilizado para armazenamento de dados das instâncias das máquinas virtuais do Azure.

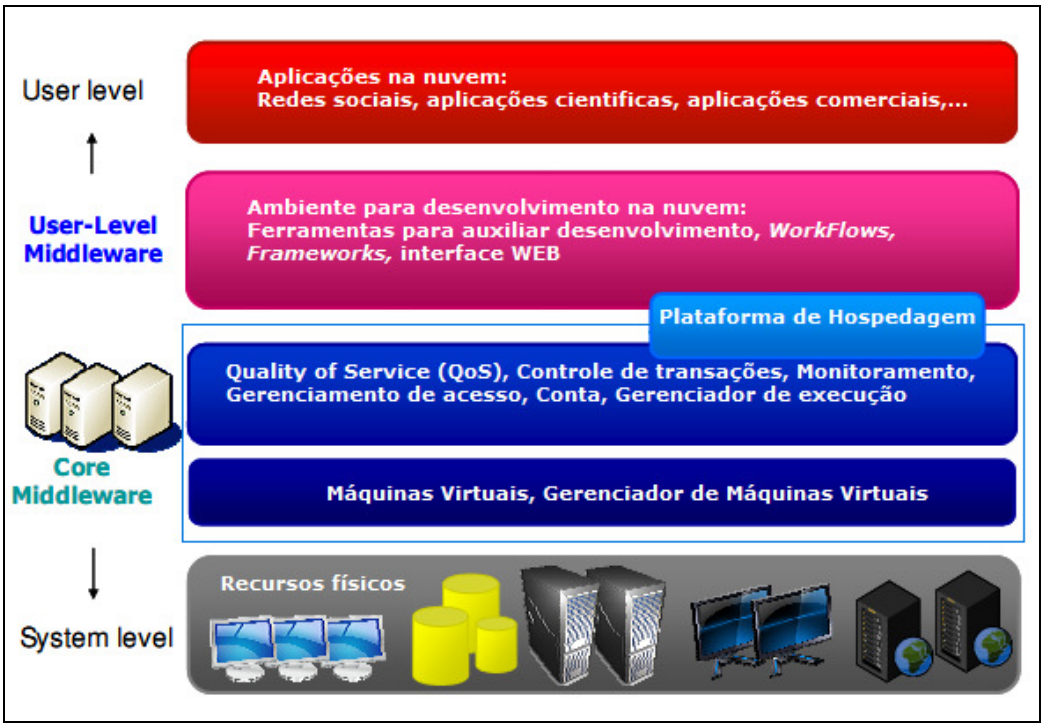
2.1 COMPUTAÇÃO NA NUVEM

Taurion (2009, p. 2) define a computação na nuvem como um conjunto de recursos, sendo eles capacidade de processamento, armazenamento, conectividade, plataformas, aplicações e serviços, disponibilizados na Internet. Algumas características importantes da computação na nuvem podem ser resumidas da seguinte maneira:

> **A computação em nuvem cria uma ilusão** da disponibilidade de recursos infinitos, acessáveis sob demanda.
>
> **A computação em nuvem elimina a necessidade** de adquirir e provisionar recursos antecipadamente.
>
> **A computação em nuvem oferece elasticidade**, permitindo que as empresas usem os recursos na quantidade que forem necessários, aumentando e diminuindo a capacidade computacional de forma dinâmica.
>
> **O pagamento dos serviços em nuvem é** pela quantidade de recursos utilizados (*pay-per-use*). (TAURION, 2009, p. 2-3, grifo do autor).

Conforme Sousa, Moreira e Machado (2010), a infraestrutura do ambiente de computação em nuvem normalmente é composta por um grande número, centenas ou milhares de máquinas físicas ou nós físicos de baixo custo, conectadas por meio de uma rede.

Na figura 1 são exibidas as camadas da computação na nuvem e suas respectivas associações.



Fonte: adaptado de Vecchila, Chu e Buyya (2009).

Figura 1 - Arquitetura da computação na nuvem

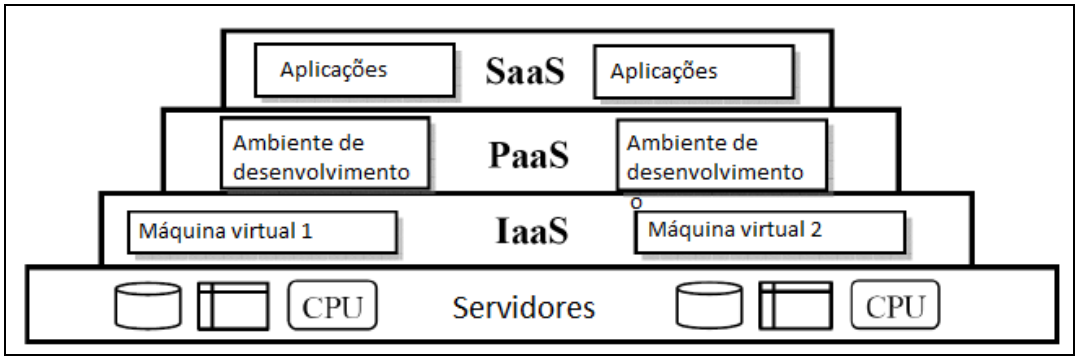
Sousa, Moreira e Machado (2010) detalham o funcionamento da computação na nuvem através de suas camadas, sendo elas *System level*, *Core Middleware*, *User-level Middleware e User level*.

System level é a camada de mais baixo nível, que representa a infraestrutura física, contendo os centros de dados, *clusters*, *desktops* e outros recursos de hardware. Devido à possibilidade destes recursos serem heterogêneos, tal estrutura possui alta flexibilidade com agregação de novos recursos na medida em que se tornam necessários. O gerenciamento da infraestrutura física é responsabilidade da camada Core *Middleware*, fornecendo um núcleo lógico de uma nuvem, tornando-se responsável por verificar requisições, gerenciamento de virtualização entre outros. *User-Level Middleware* é responsável por auxiliar e dar suporte ao desenvolvimento de aplicações, com *frameworks* para auxílio na construção de softwares e sua publicação na nuvem. Por fim, a camada *user level* é aquela pela qual o usuário interage com as aplicações hospedadas na nuvem (SOUSA; MOREIRA; MACHADO, 2010).

Para o usuário final, não é visível este gerenciamento e interação entre as camadas. O serviço é acessado sem a necessidade de que ele saiba a tecnologia utilizada e a localização da mesma (SOUSA; MOREIRA; MACHADO, 2010).

2.2 MODELOS DE SERVIÇOS

Lin e Shih (2010, p. 34) detalham que na arquitetura de serviços, a computação na nuvem oferece serviços que podem ser agrupados em três categorias: *Software as a Service* (SaaS), *Platform as a Service* (PaaS) *e Infrastrucutre as a Service* (IaaS). A figura 2 exibe um breve relacionamento destas categorias.



Fonte: adaptado de Lin e Shih (2010, p. 34).

Figura 2 – Modelo de serviços da computação na nuvem

SaaS consiste em vários serviços oferecidos e disponibilizados aos usuários finais. A transferência através do navegador elimina a necessidade de instalar e executar aplicativos no computador local. Editores de texto na web, como o Google Docs e o Windows Live Office, são os exemplos mais comuns desta abordagem (LIN; SHIH, 2010, p. 35).

Conforme detalhamento de Lin e Shih (2010, p. 35), o PaaS oferece serviços como aplicações web e servidores de banco de dados. Esta camada está disponível para depurar, testar e executar aplicações dos desenvolvedores. Desta forma, os desenvolvedores podem executar as aplicações na infraestrutura da nuvem, facilitando a implantação, sem o custo da gestão do hardware e licenças de software.

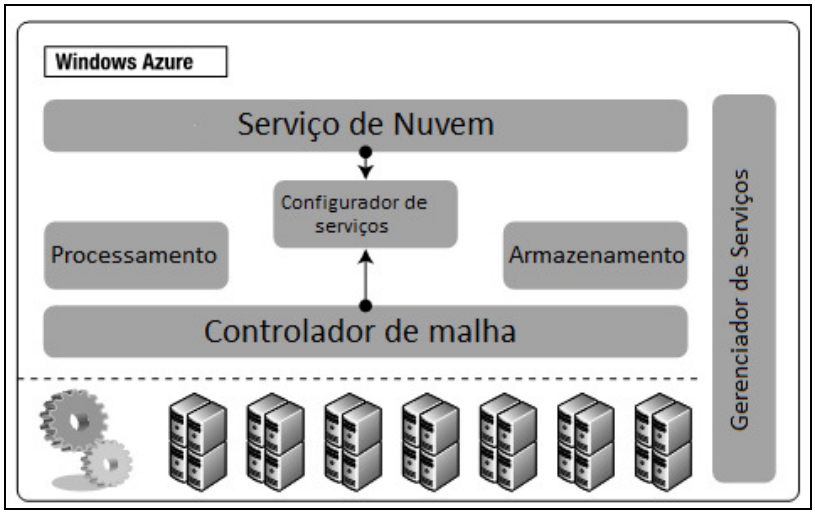
O IaaS é um serviço na nuvem, que virtualiza os equipamentos de hardware e oferece serviços de computação, memória, armazenamento, entre outros. O IaaS fornece um ambiente virtualizado na nuvem, que terceiriza o equipamento utilizado, vendendo seu serviço sob demanda. O prestador de serviços é proprietário do equipamento e responsável por manter toda a infraestrutura. O cliente normalmente compra o serviço sob demanda, contando com uma moderna tecnologia (LIN, SHIH, 2010, p. 35).

2.3 WINDOWS AZURE

O Windows Azure é uma oferta de PaaS da Microsoft. O Azure fornece um servidor de aplicativo hospedado e a infraestrutura necessária de armazenamento, redes e integração para execução de aplicações Windows. As políticas de provisionamento de recursos ocorrem em tempo real e o controlador de malha é o núcleo que controla todo o hardware e software de maneira redundante (KOMMALAPATI, 2010).

Os aplicativos são hospedados em ambientes virtualizados, para impedir quaisquer dependências físicas do hardware sendo assim qualquer falha de aplicativo não afeta outros aplicativos em execução no mesmo recurso físico (KOMMALAPATI, 2010).

A figura 3 exibe uma visão geral da arquitetura do Windows Azure.



Fonte: adaptado de (REDKAR, 2009, p. 106).

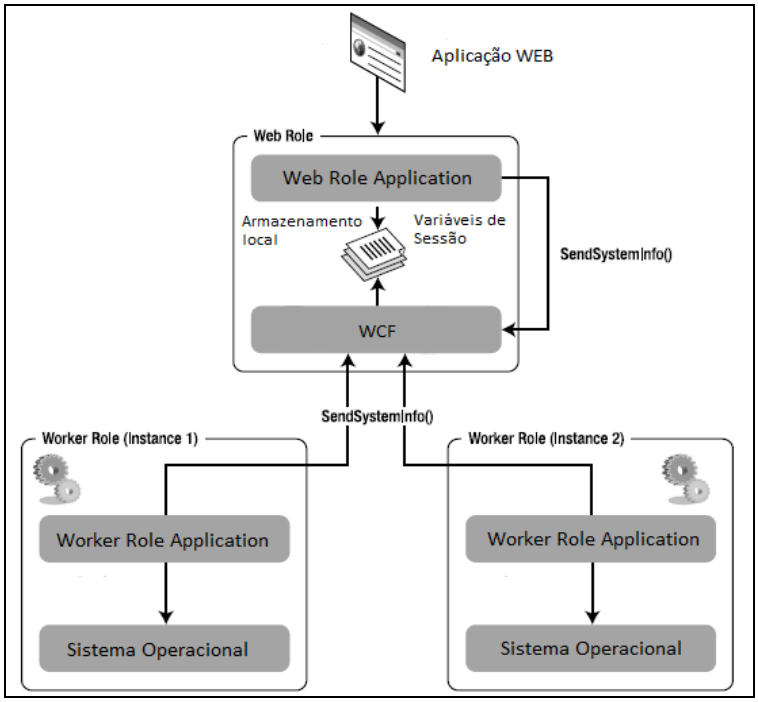
Figura 3 – Controlador de malha

Redkar (2009, p. 106) detalha que o Windows Azure é projetado para ser altamente escalável e disponível. O controlador de malha lê as informações de configuração do serviço enviado pelo serviço na nuvem (aplicação) e, consequentemente, gera as máquinas virtuais necessárias para rodar o serviço na nuvem. As instâncias de máquinas virtuais são transparentes para o desenvolvedor. Caso tenha demanda de mais processamento ou armazenamento, o Windows Azure tomará as providências necessárias para prover mais recursos ao ambiente.

O serviço de processamento é divido em duas categorias, w*orker role application* e *web role application. Worker role application* são processos que rodam em segundo plano continuamente na nuvem, podendo expor e*ndpoints* externos e internos. Uma instância de *worker role application* é independente de uma instância *de web role application*, mesmo que

eles pertençam a um mesmo serviço. *Web role application* é uma aplicação web que roda em um ambiente IIS, normalmente uma aplicação *Active Server Page* (ASP) ou uma aplicação WCF. Tanto uma *worker role application* quanto uma *web role application* podem ser instanciadas em diversas máquinas virtuais, permitindo assim um alto nível de escalabilidade para o serviço hospedado na nuvem (REDKAR, 2009, p. 110-111).

A figura 4 exibe a arquitetura dos *worker roles* no Windows Azure.



Fonte: adaptado de Redkar (2009, p. 166).

Figura 4 – Exemplo de *worker roles*

2.3.1 Azure Storage

Conforme Kommalapati (2010), aplicativos e serviços implantados na plataforma Windows Azure podem usar o Azure Storage para persistência de conteúdo, tendo acesso baseado em *Secure Socket Layer* (SSL), *HyperText Transfer Protocol Secure* (HTTPS) juntamente com o protocolo *HyperText Transfer Protocol* (HTTP) básico.

O Azure Storage é amplamente dimensionável e sua escalabilidade é obtida por meio de um conjunto de servidores denominados nós. Cada nó é responsável por uma quantidade finita de armazenamento e seu acesso é feito por balanceamento de carga. A confiabilidade é obtida por meio de redundância e por transações atômicas, somente sendo concluída quando salva três réplicas (KOMMALAPATI, 2010).

Nas próximas seções serão detalhados os três aspectos de abstração de dados disponíveis no Azure Storage.

2.3.1.1 Azure blob

O Azure blob é um serviço que prove armazenamento escalável e altamente disponível para qualquer tipo de entidade, tais como arquivos binários e documentos. Sua escalabilidade e alta disponibilidade são devido à distribuição de arquivos em vários servidores (REDKAR, 2009, p. 207).

Redkar (2009, p. 211 - 212) destaca os componentes da estrutura do Azure blob:

a) *blob container*: é um agrupamento lógico de um conjunto de *blobs*;
b) *blobs*: são as entidades do serviço. Os *blobs* contém metadados na forma nome-valor;
c) *blocks*: é uma estrutura que possibilita carregar o arquivo em partes de até 4 MB (*Mega Bytes*), otimizando o carregamento de arquivos através do protocolo HTTP;
d) pages: é uma estrutura otimizada para leitura/gravação e fornece a capacidade de copiar uma série de *bytes* em um *blob.*

2.3.1.2 Azure queue

Azure queue é uma estrutura de fila de mensagens disponibilizado pelo Azure Storage, provendo um mecanismo simples e robusto para troca de mensagens assíncronas. O Azure queue é utilizado de duas formas, como barramento de comunicação via mensagens e como componente para desacoplamento de módulo. (LI, 2009, p. 67).

Li (2009, p. 68) destaca as vantagens de se utilizar o Azure queue para aplicações de alta escalabilidade na relação a seguir:

a) medição de tráfego usando o comprimento da fila: se um aplicativo é construído com o Azure queue e precisa determinar a escalabilidade com base do tráfego, é possível obter a resposta medindo o comprimento da fila, uma vez que o comprimento da fila reflete diretamente a carga de processamento da aplicação. Se a fila é crescente, significa que há uma demanda de processamento, sendo necessário aumentar as instâncias de máquinas virtuais para processar. Contudo, se

o tamanho da fila é próximo a zero, é uma indicação que há mais oferta do que demanda de processamento, sendo possível diminuir as instâncias das máquinas virtuais;

b) módulo de desacoplamento do sistema: as aplicações construídas com base na troca de mensagens, são fracamente acoplados, o que dá a flexibilidade para ampliação e expansão. Utilizando a fila de mensagens para comunicação, viabiliza-se a independência entre os componentes, tornando mais fácil a substituição e alteração dos mesmos;
c) gestão eficiente dos recursos: aplicações baseadas em fila de mensagens, podem gerenciar a alocação de recursos do sistema com mais eficiência. Recursos do sistema podem ser agrupados e distribuídos em filas distintas, de acordo com suas necessidades. Aplicações críticas podem possuir suas próprias filas, reduzindo o impacto sobre outros componentes do sistema;
d) *buffer* de mensagens: o Azure Queue foi concebido de maneira que seja garantida a entrega das mensagens. Com o *buffer* não há perda de dados caso o tráfego aumente radicalmente, evitando erros e necessidade de nova transmissão.

2.3.1.3 Azure table

Li (2009, p. 2) explica que o Azure table é o armazenamento estruturado fornecido pela plataforma Azure. Suporta tabelas massivamente escaláveis na nuvem, que são armazenadas em diversos servidores. A plataforma abstrai para o desenvolvedor o controle de transação e a simultaneidade para atualizações e exclusões. O Azure table possui a seguinte estrutura:

a) *table*: contém um conjunto de *entities*. Nomes de tabelas são associados à conta do usuário no Windows Azure. Não há limite de quantas tabelas um aplicativo pode criar em uma conta do Windows Azure;
b) *entity*: é um registro de uma *table*. Uma *entity* possui várias *properties*;
c) *property*: é o valor que está sendo persistido em uma entidade.

2.4 WINDOWS COMMUNICATION FOUNDATION

Conforme definição da MSDN (2011), WCF é uma parte do *.NET Framework* que fornece um modelo unificado de programação para construir de forma rápida aplicações distribuídas orientadas a serviço ou *Service Oriented Architecture* (SOA).

Krishnamurthy e Thothadri (2007) detalham que o WCF foi desenvolvido com o objetivo de unificar as até então existentes tecnologias de programação distribuídas como *COM+, MSQM-Message Queue, Enterprise Services, .NET Remoting e Web Services*.

A definição da MSDN (2011) diz que toda comunicação com um serviço WCF ocorre através de e*ndpoints* do serviço, que fornecem aos clientes o acesso às funcionalidades oferecidas por um serviço WCF. Para um serviço ser exposto, o e*ndpoint* necessita de alguns elementos: um *address* que define onde o serviço reside, um *contract* que define que métodos o serviço irá expor e um *binding* que define como se comunicar com o serviço. O relacionamento entre estes elementos, *address*, *binding* e *contract,* é denominado *ABC's endpoints.*

Löwy (2010, p. 24) destaca que há vários aspectos de comunicação com um determinado serviço e há muitos padrões de comunicação possíveis. Para isto existem diversos tipos de *bindings*, ou seja, diversos tipos de ligações possíveis com WCF.

Lowy (2010, p.25) detalha os mais frequentes conforme lista abaixo:

a) *Basic binding*: é uma ligação implementada pela classe `BasicHttpBinding`, onde o serviço utiliza a implementação do web servisse.

TCP binding: implementado através da classe `NetTcpBinding`, utiliza o protocolo TCP para comunicação entre máquinas na intranet. Suporta uma variedade de recursos, incluindo transações;

b) *IPC binding*: implementado pela classe `NetNamedPipeBinding`, utiliza *pipes* como um transporte para o mesmo aparelho de comunicação. É a ligação mais segura, uma vez que não pode aceitar chamadas de fora da máquina. É também, a ligação de melhor desempenho;

c) *Web Service (WS) binding*: ligação implementada pela classe `WSHttpBinding`, utiliza o HTTP ou HTTPS para o transporte e oferece características, como transações e segurança. Essa ligação é projetada para interoperar com qualquer serviço que suporta os padrões de web service;

d) MSMQ binding: é uma ligação que oferece suporte para chamadas em fila. Ela é implementada pela classe `NetMsmqBinding`.

Löwy (2010, p. 29) destaca também outro tipo de ligação comum, o *WebHttpBinding*, que é bastante popular entre as redes sociais. Esta ligação permite que o serviço aceite chamada simples através de protocolos web, como HTTP-GET, usando *Representational State Transfer* (REST)[3].

2.5 TRABALHOS CORRELATOS

Atualmente é possível utilizar diversas ferramentas que abordam o conceito da computação na nuvem. Aplicações como CloudAV estão disponíveis na internet, de fácil acesso ao usuário, não sendo necessária a instalação do software para seu uso. Outras como a Amazon Web Services são estruturas disponibilizadas para receberem aplicações e disponibilizá-las na nuvem. Além de aplicações há *frameworks.* Por exemplo, a Microsoft está disponibilizando versões *beta* de *frameworks* para administração de uma aplicação no Windows Azure, aplicando escalabilidade de acordo com um comportamento. Abaixo, segue uma breve descrição das ferramentas citadas.

2.5.1 CloudAV

CloudAV é um novo modelo para detecção de vírus em máquinas baseado no fornecimento de um antivírus como um serviço de computação na nuvem. Sousa, Moreira e Machado (2010) afirmam que este modelo utiliza uma técnica chamada de *N-version protection,* que permite a identificação de arquivos maliciosos e indesejados por múltiplos mecanismos de detecção em paralelo. Desta maneira, cada arquivo é analisado por diversos aplicativos antivírus, utilizando diferentes heurísticas, que fornecem uma melhor identificação de arquivos maliciosos.

Sousa, Moreira e Machado (2010) explicam que o CloudAV utiliza dois componentes,

[3] Segundo Singh e Huhns (2005, p. 50), REST é um estilo de arquitetura para a concepção de serviços web. Ele utiliza o padrão HTTP e representações de recursos tal como XML entre outros.

um agente e um serviço de rede. O agente é hospedado na máquina do usuário e tem por função verificar a existência de novos arquivos e enviá-los ao serviço de rede. O serviço de rede é composto por diversos outros serviços e ele tem a responsabilidade de identificar os arquivos maliciosos e suspeitos.

2.5.2 WASABi

WASABi é um framework para aplicar a auto escalabilidade no Windows Azure através de comportamentos configurados pelos usuários (MICROSOFT, 2011, p. 6).

A escala é dimensionada automaticamente de acordo com as regras definidas pelo desenvolvedor na aplicação. Desta maneira, ele escala a aplicação, e ao mesmo tempo mantém o controle sobre os custos de manter a aplicação no Windows Azure (MICROSOFT, 2011, p. 7).

O WASABi coleta informações das aplicações executando no Windows Azure, através da bibliotecas `Microsoft WindowsAzure Diagnostics` e `Microsoft WindowsAzure ServiceRuntime`. Através destes coletores de dados, o WASABi consegui medir o uso de processador de um *role*, tais como uso de memória, e utiliza-los como parâmetro para efetuar escalabilidade (MICROSOFT, 2011, p. 33).

2.5.3 Amazon Web Services

O Amazon Web Services (AWS) é um ambiente em computação na nuvem de alta escalabilidade, disponibilidade, elasticidade e desempenho. Neste ambiente é disponibilizada toda uma infraestrutura para computação em diversos níveis, onde contempla tanto tarefas simples como tarefas de alto desempenho, com uma gerência eficaz de recursos (SOUSA; MOREIRA; MACHADO, 2010).

O AWS possui um sistema responsável pelo gerenciamento das execuções de aplicações na infraestrutura Amazon. Este sistema é conhecido como *Elastic Compute Cloud* (EC2). Ele permite um controle sobre as instâncias dos sistemas e sua interação com elas (SOUSA; MOREIRA; MACHADO, 2010).

O sistema de armazenamento, *SimpleDB,* fornece as funcionalidades de um sistema

banco de dados como armazenamento, indexação e consultas em ambientes de nuvem. Sua arquitetura é utilizada para o armazenamento e recuperação dos estados do sistema (SOUSA; MOREIRA; MACHADO, 2010).

Sousa, Moreira e Machado (2010) destacam que o AWS disponibiliza *Amazon Machine Image* (AMI), uma imagem armazenada em um repositório. As AMIs permitem a execução de variados sistemas operacionais, tais como Red Hat Linux, Windows Server 2003, OpenSolaris, entre outros.

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo é apresentado o desenvolvimento do protótipo de teste de escalabilidade, sendo dividido em várias etapas. Na seção 3.1 será apresentada a visão geral do protótipo. Na seção 3.2 será exibida a especificação dos requisitos funcionais e não funcionais do protótipo. Na seção 3.3 será apresentada a especificação do protótipo, tais como diagramas de casos de uso, diagramas de sequência e diagramas de classes. Por fim na seção 3.4 será apresentada a implementação do protótipo.

3.1 VISÃO GERAL DO PROTÓTIPO

O protótipo de testes de escalabilidade tem por finalidade demonstrar a plataforma Windows Azure com pagamento por uso (Anexo D) e exibir a escalabilidade que a plataforma disponibiliza, através de processamento de algoritmos que utilizam exaustivamente memória e processamento.

A arquitetura do protótipo de testes de escalabilidade é composto por quatro sub-protótipos listados abaixo:

a) `Cliente de Testes`: é um protótipo que executa em um computador local e consome o serviço disponibilizado pelo Servidor de Requisições através de um *web service;*

b) `Servidor de Requisições`: é o serviço desenvolvido para receber requisições e armazena-las na fila de mensagens. O serviço é exposto através de um *web service*;

c) `Executor de Testes`: é um processo que executa continuamente na nuvem, verificando a fila de mensagens e executando o processamento de requisições contidas nela;

d) `Monitor de Testes`: é um protótipo que é executado em um computador local e verifica o status da fila de mensagens, quantidade de Executores de testes em execução, com o objetivo de exibir a situação do protótipo executando na nuvem.

A figura 5 exibe uma visão geral do protótipo desenvolvido.

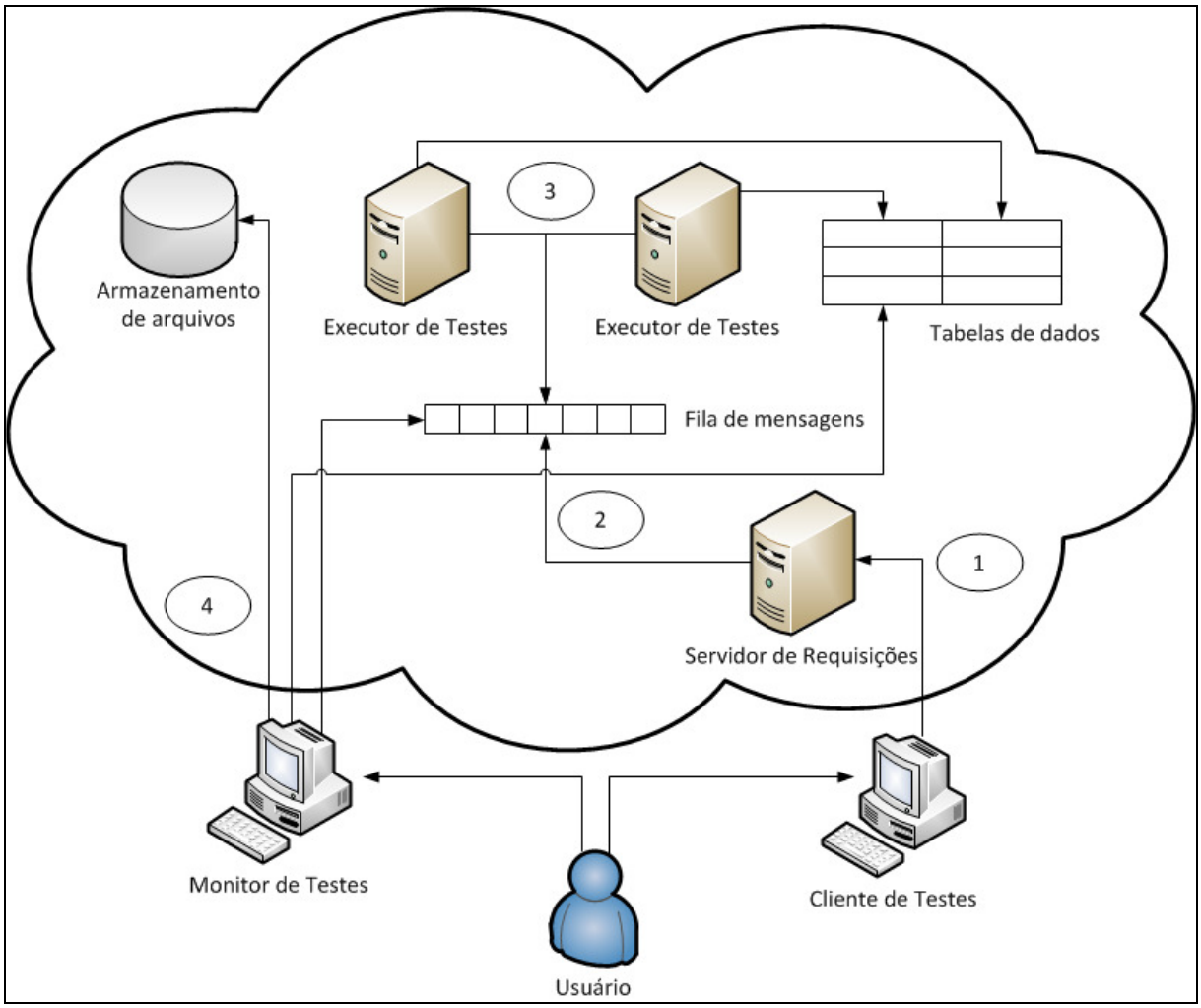


Figura 5 - Visão geral do protótipo de teste de escalabilidade

Para facilitar o entendimento da figura 5, foram numerados os protótipos para facilitar o entendimento. No passo 1 o `Cliente de Testes` efetua uma requisição através um *web service* que está disponível no Servidor de Requisições. No passo 2 o `Servidor de Requisições` adiciona esta requisição na fila de mensagens e retorna se foi ou não bem sucedido a adição da requisição na fila de mensagens para o `Cliente de Testes`. Enquanto isto no passo 3 os `Executores de Testes` estão lendo a fila de mensagens, retirando uma mensagem de cada vez e efetuando seu processamento. Os resultados são armazenados nas Tabelas de dados. O passo 4 é monitorar a fila de mensagens e as tabelas de dados, desta maneira é possível medir quantas requisições foram processadas e o tamanho da fila de mensagens. Baseando-se nas regras configuradas, o monitor executa a escalabilidade de novos executores de testes medindo a fila de mensagens. Tende-se a alocar novos `Executores de Testes` quando a fila está grande, e remover `Executores de Testes` quando a fila está menor.

Para o protótipo de testes de escalabilidade ser altamente escalável ele utiliza uma

arquitetura que disponibiliza isto. No protótipo em questão, dois itens foram fundamentais para que isto ocorresse, o uso processamento assíncrono e seções sem estado.

O processamento é assíncrono, de maneira que o resultado de uma requisição não é obtido no momento em que é requisitado, mas sim no momento em que um dos Executores de Testes efetua o seu processamento, o que pode ocorrer esporadicamente, levando segundos ou até horas. A resposta que se tem após efetuar uma requisição é a de sucesso de adicionar o item na fila de mensagens.

As seções não possuem afinidade, ou seja, não é criada nenhuma seção no IIS para processamentos futuros, sendo assim, cada requisição é processada em qualquer um dos Executores de Testes que estiverem disponíveis.

A fila de mensagens e a tabela de dados são estruturas disponibilizadas pelo Windows Azure Storage(Anexo A).

3.2 REQUISITOS DO PROTÓTIPO

Com o objetivo de testar a escalabilidade do Windows Azure, o protótipo para testes de escalabilidade atenderá os requisitos relacionados nos quadros 1 e 2:

REQUISITOS FUNCIONAIS
RF01: O protótipo deve implementar um *web service* e ser publicado no Windows Azure para receber requisições de processamento de algoritmos com o objetivo de efetuar uso de processamento e memória dos Executores de Testes.
RF02: O protótipo deve inserir as requisições recebidas no *web service* em uma fila de mensagens.
RF03: O protótipo deve implementar sessões *state-less* (sem estado) no Windows Azure.
RF04: O protótipo deve processar as requisições no Windows Azure de forma assíncrona.
RF05: O protótipo deve armazenar os resultados das requisições em uma tabela de dados.
RF06: O protótipo deve alocar novos recursos no Windows Azure conforme configuração do o usuário no `Protótipo de Monitoramento`.

Quadro 1 – Requisitos funcionais

REQUISITOS NÃO FUNCIONAIS
RNF01: Implementar o protótipo utilizando o ambiente de desenvolvimento Microsoft Visual Studio 2010.
RNF02: Implementar o protótipo utilizando o *framework .NET* 4.0.
RNF03: Implementar o protótipo utilizando a linguagem C#.
RNF04: Implementar o protótipo utilizando o *Software Development Kit* (SDK) Azure 1.4
RNF05: Implementar o *web service* utilizando WCF.

Quadro 2 – Requisitos não funcionais

3.3 ESPECIFICAÇÃO

Em todo o desenvolvimento foi utilizada a orientação a objetos e para a especificação da implementação foi utilizada a *Unified Modeling Language* (UML) através de diagramas de casos de uso, diagrama de classe e diagrama de sequência. Para a modelagem dos diagramas foi utilizada a ferramenta Microsoft Visio 2010.

3.3.1 Diagrama de casos de uso

Os casos de uso irão documentar os requisitos funcionais especificados para o protótipo de teste de escalabilidade. Os requisitos divididos em seções detalhando separadamente cada protótipo. Na seção 3.2.1 serão detalhados os casos de uso relacionados ao `Cliente de Testes`. Na seção 3.2.1.2, serão detalhados os casos de uso relacionados ao `Servidor de Requisições`. Na seção 3.2.1.3, serão detalhados casos de uso do `Executor de Testes`. Por fim, na seção 3.2.1.4, serão detalhados os casos de uso referente ao `Monitor de Testes`.

3.3.1.1 Cliente de Testes

A figura 9 apresenta os casos de uso relacionados `ao Cliente de Testes` do protótipo.

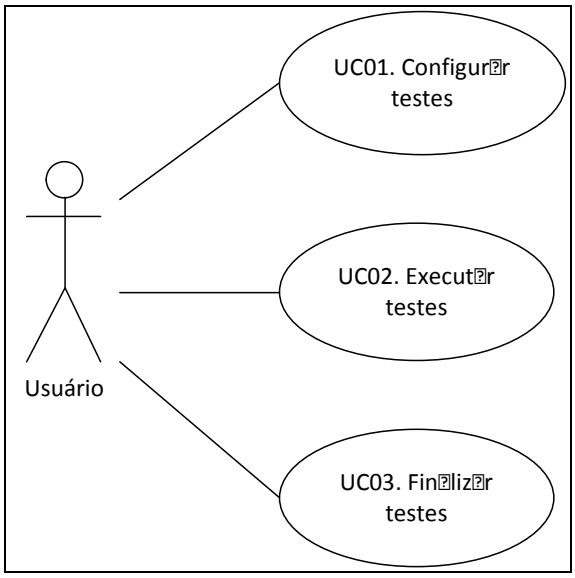


Figura 6 – Casos de uso do `Cliente de Testes`

O quadro 3 detalha caso de uso `UC01`, apresentado na figura 6.

UC01 – Configurar testes	
Resumo	Parametriza o `Cliente de Testes` para os testes conforme configuração do usuário. O usuário irá parametrizar os testes que irá executar, sendo eles de memória ou de processamento. A intensidade de cada um dos testes deve ser configurada assim como a quantidade de requisições por *thread.*
Pré-condição	Não existe.
Cenário principal	1 – O usuário deve informar quais os testes deseja executar; 2 – O usuário deve informar a intensidade de cada teste; 3 – O usuário deverá informar quantas requisições simultâneas terá em execução através do campo de quantidade de *threads*; 4 – O usuário deverá informar quantas solicitações cada *thread* deve executar para cada teste.
Pós-condição	Estará disponível para testes um cenário configurado pelo usuário.

Quadro 3 – Detalhamento do caso de uso `UC01`

O quadro 4 detalha o caso de uso `UC02`, apresentado na figura 6.

UC02 – Executar testes	
Resumo	Executa os testes parametrizados pelo usuário.
Pré-condição	O `Servidor de Requisições` deverá estar disponível para consumo.
Cenário principal	1 – Parametriza o teste de processador conforme configuração do usuário; 2 – Parametriza o teste de memória conforme configuração do usuário; 3 – Executa os testes de memória e de processador em *threads* distintas; 4 – O gráfico de requisições enviadas deverá exibir a quantidade de solicitações enviadas por segundo de cada teste; 5 – O gráfico de requisições processadas deverá exibir a quantidade de solicitações processadas por segundo de cada teste.
Pós-condição	O gráfico dos testes deverá parar de ser atualizado.

Quadro 4 – Detalhamento do caso de uso `UC02`

O quadro 5 detalha o caso de uso `UC03`, apresentado na figura 6.

UC03 – Finalizar testes	
Resumo	Finaliza os testes que estão em execução.
Pré-condição	O `Cliente de Testes` deverá estar executando os testes.
Cenário principal	1 – O usuário clica em “Finalizar”; 2 – Os testes são abortados.
Pós-condição	O gráfico dos testes deverá parar de ser atualizado.

Quadro 5 – Detalhamento do caso de uso `UC03`

3.3.1.2 Servidor de Requisições

A figura 7 representa os casos de uso do `Servidor de Requisições`. O `Cliente de Testes` é o ator neste diagrama, pois irá consumir o `Servidor de Requisições` através de um *web service.*

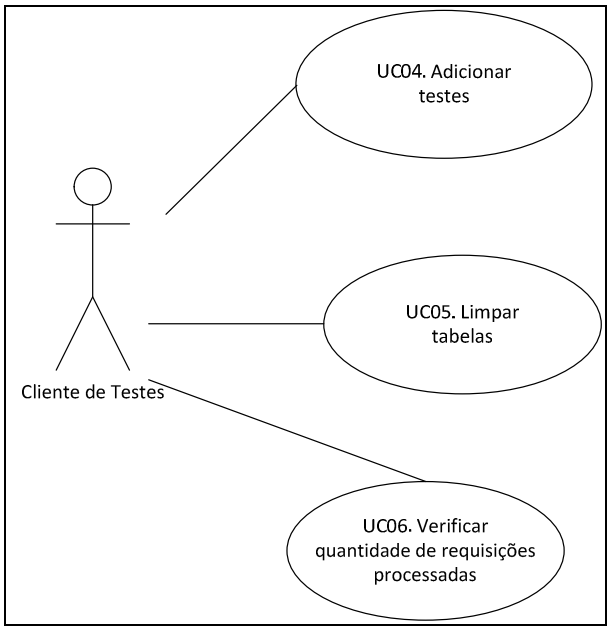


Figura 7 – Casos de uso de requisições enviadas ao `Servidor de Requisições`

O quadro 6 detalha o caso de uso `UC04`, apresentado na figura 7.

UC04 – Adicionar testes	
Resumo	Adiciona as requisições recebidas na fila de mensagens.
Pré-condição	A fila de mensagens deverá estar criada.
Cenário principal	1 – Formatar a mensagem no seguinte formato: Funcao|Parâmetros Onde função é o tipo de teste, e Parâmetros são os parâmetros necessários para sua execução. 2 – Retornar *true* caso a mensagem tenha sido adicionada com sucesso, e *false* caso contrário.
Pós-condição	A requisição será adicionada a fila de mensagens.

Quadro 6 – Detalhamento do caso de uso `UC04`

O quadro 7 detalha o caso de uso `UC05`, apresentado na figura 7.

UC05 – Limpar tabelas	
Resumo	Limpa a tabela de dados que contém os resultados das requisições.
Pré-condição	A tabela de dados deverá estar criada.
Cenário principal	1 – O usuário deve efetuar uma requisição através do *web service* solicitando para limpar a tabela de dados; 2 – O protótipo deverá excluir todos os registros da tabela de dados dos testes executados.
Pós-condição	A tabela de dados dos testes deverá estar sem nenhum registro.

Quadro 7 – Detalhamento do caso de uso `UC05`

O quadro 8 detalha o caso de uso UC06, apresentado na figura 7.

UC06 – Verificar quantidade de requisições processadas	
Resumo	Verifica e retorna a quantidade de requisições processadas.
Pré-condição	A tabela de dados deverá estar criada.
Cenário principal	1 – O usuário deve efetuar uma requisição através do *web service* para consultar a quantidade de requisições processadas; 2 – Será enviada a quantidade de requisições processadas.
Pós-condição	O usuário receberá a quantidade de requisições processadas.

Quadro 8 – Detalhamento do caso de uso UC06

3.3.1.3 Executor de Testes

Os casos de uso desta seção estão relacionados aos requisitos funcionais RF07 e RF 08. A figura 7 apresenta os casos de uso relacionados ao Executor de Testes.

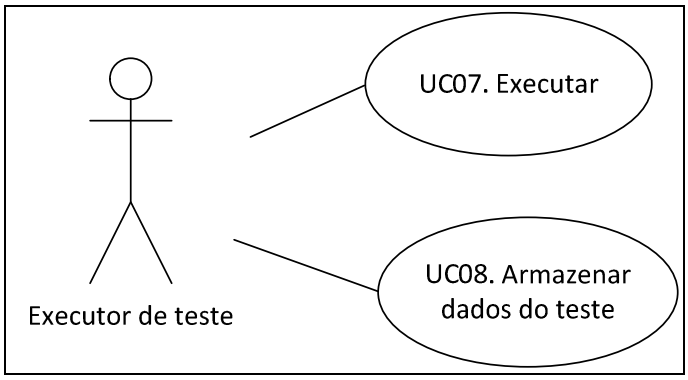


Figura 8 – Casos de uso do Executor de Testes

O quadro 9 detalha o caso de uso UC07, apresentado na figura 8.

UC07 – Executar	
Resumo	Lê a requisição da fila de mensagem e executa a mesma.
Pré-condição	Deverá haver requisições na fila.
Cenário principal	1 – Retirar uma mensagem da fila; 2 – Ler o formato da mensagem e separar os parâmetros; 3 – Executar o teste especificado; 4 – Executar o UC08.
Pós-condição	Haverá um registro contendo o teste e o tempo para executá-lo.

Quadro 9 – Detalhamento do caso de uso UC07

O quadro 10 detalha o caso de uso UC08, apresentado na figura 8.

UC08 – Armazenar dados do teste	
Resumo	Armazena o resultado de execução do teste em uma tabela.
Pré-condição	Não existe.
Cenário principal	1 – Criar uma entidade para armazenar o resultado; 2 – Salvar o nome do teste e o tempo de execução na entidade; 3 – Salvar na tabela de dados.
Pós-condição	Haverá um registro contendo o teste e o tempo para executá-lo.

Quadro 10 – Detalhamento do caso de uso UC08

3.3.1.4 Monitor de Testes

Os casos de uso do Monitor de Testes estão relacionados ao requisito funcional RF06. A figura 9 representa os casos de uso relacionados ao Monitor de Testes do protótipo.

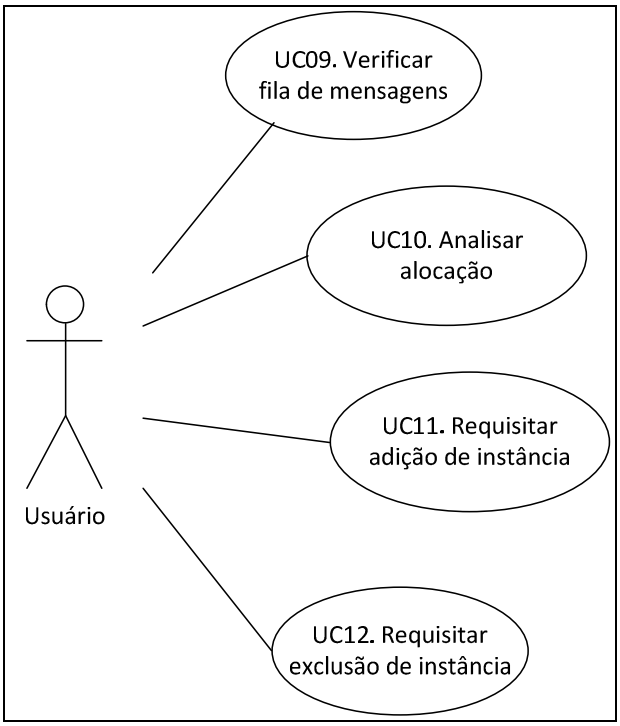


Figura 9 – Casos de uso do `Monitor do Protótipo`.

O quadro 11 detalha o caso de uso `UC09`, apresentado na figura 9.

UC09 – Verificar fila de mensagens	
Resumo	Monitora o tamanho de fila de mensagens contendo as requisições para processamento.
Pré-condição	Fila de mensagens deve estar criada.
Cenário principal	1 – Conectar se a fila de mensagens; 2 – Consultar o tamanho da fila de mensagens.
Pós-condição	Ter informação do tamanho da fila de mensagens.

Quadro 11 – Detalhamento do caso de uso `UC09`

O quadro 12 detalha o caso de uso `UC10`, apresentado na figura 9.

UC10 – Analisar alocação	
Resumo	Analisa as regras para incremento e decremento de recurso e efetua a adição ou remoção de uma nova instância de máquina virtual.
Pré-condição	Não existe.
Cenário principal	1 – Ler as configurações parametrizadas pelo o usuário; 2 – No período parametrizado, ler a fila de mensagens e analisar a quantidade de mensagens em espera; 3 – Verificar se deve incrementar ou decrementar novas instâncias de acordo com os parâmetros configurados.
Pós-condição	Deverá ser incrementado ou decrementado o número de executores de testes.

Quadro 12 – Detalhamento do caso de uso UC10

O quadro 13 detalha o caso de uso UC05, apresentado na figura 9.

UC11 – Requisitar adição de instância	
Resumo	Requisita uma nova instância de máquina virtual ao Windows Azure.
Pré-condição	O arquivo `configuracao.xml` deverá estar salvo na estrutura de armazenamento de arquivos.
Cenário principal	1 – Conectar-se a estrutura de armazenamento de arquivos e carregar o arquivo `configuracao.xml`. 2 – Alterar o arquivo `configuracao.xml`, incrementando o número de instâncias das máquinas virtuais do tipo processador. 3 – Enviar o novo arquivo de configuração para o *Windows Management Service*. 4 – Atualizar o arquivo `configuracao.xml` na estrutura de armazenamento de arquivos.
Pós-condição	Uma nova instância de máquina virtual deverá estar inserida no serviço.

Quadro 13 – Detalhamento do caso de uso UC11

O quadro 14 detalha o caso de uso UC12, apresentado na figura 9.

UC12 – Requisitar exclusão de instância	
Resumo	Requisita a exclusão de uma máquina virtual no Windows Azure.
Pré-condição	O arquivo `configuracao.xml` deverá estar salvo na estrutura de armazenamento de arquivos.
Cenário principal	1 – Conectar-se ao Azure blob e carregar o arquivo `configuracao.xml`. 2 – Alterar o arquivo `configuracao.xml`, decrementando o número de instâncias das máquinas virtuais do tipo processador. 3 – Enviar o novo arquivo de configuração para o Windows Management Service. 4 – Atualizar o arquivo `configuracao.xml` na estrutura de armazenamento de arquivos.
Cenário alternativo	Se no passo 2 o arquivo de configuração constar como quantidade de processadores igual a dois, não poderá ser alterado o arquivo de configuração e não deverá ser enviado nenhuma requisição ao Windows Management Service.
Pós-condição	Uma instância de máquina virtual deverá ser excluída do serviço.

Quadro 14 – Detalhamento do caso de uso `UC12`

3.3.2 Diagrama de classes

Os diagramas de classe do protótipo serão exibidos pelos quatro protótipos que o compõem como um todo. Na seção 3.3.2.1 será detalhado o diagrama de classes do protótipo `Cliente de Testes`. Na seção 3.3.2.2 será detalhado o diagrama de classes do `Servidor de Requisições`. Na seção 3.3.2.3 será detalhado o diagrama de classes do `Executor de Testes`. Por fim, na seção 3.3.2.4 será detalhado o diagrama de classes do `Monitor de Testes`.

3.3.2.1 Cliente de Testes

A figura 10 apresenta o diagrama de classes do `Cliente de Testes`. A classe `Painel` implementa a tela na qual o usuário parametriza e dispara a execução dos testes. A classe

`Memoria` é responsável de executar testes relacionados ao consumo de memória. A classe `Processador` é responsável de executar testes relacionados ao consumo de processamento. Ambas as classes `Memoria` e `Processador` utilizam a classes `ServicoClient` para consumir o *web service* para requisição dos testes. A classe `ServicoClient` contém as referencias para o *web service* responsável por adicionar as requisições na fila de mensagens. Ela implementa a interface `IServico` responsável pela definição dos contratos disponíveis e a interface `IServicoChannel` responsável pela parametrização da comunicação com o *web service.*

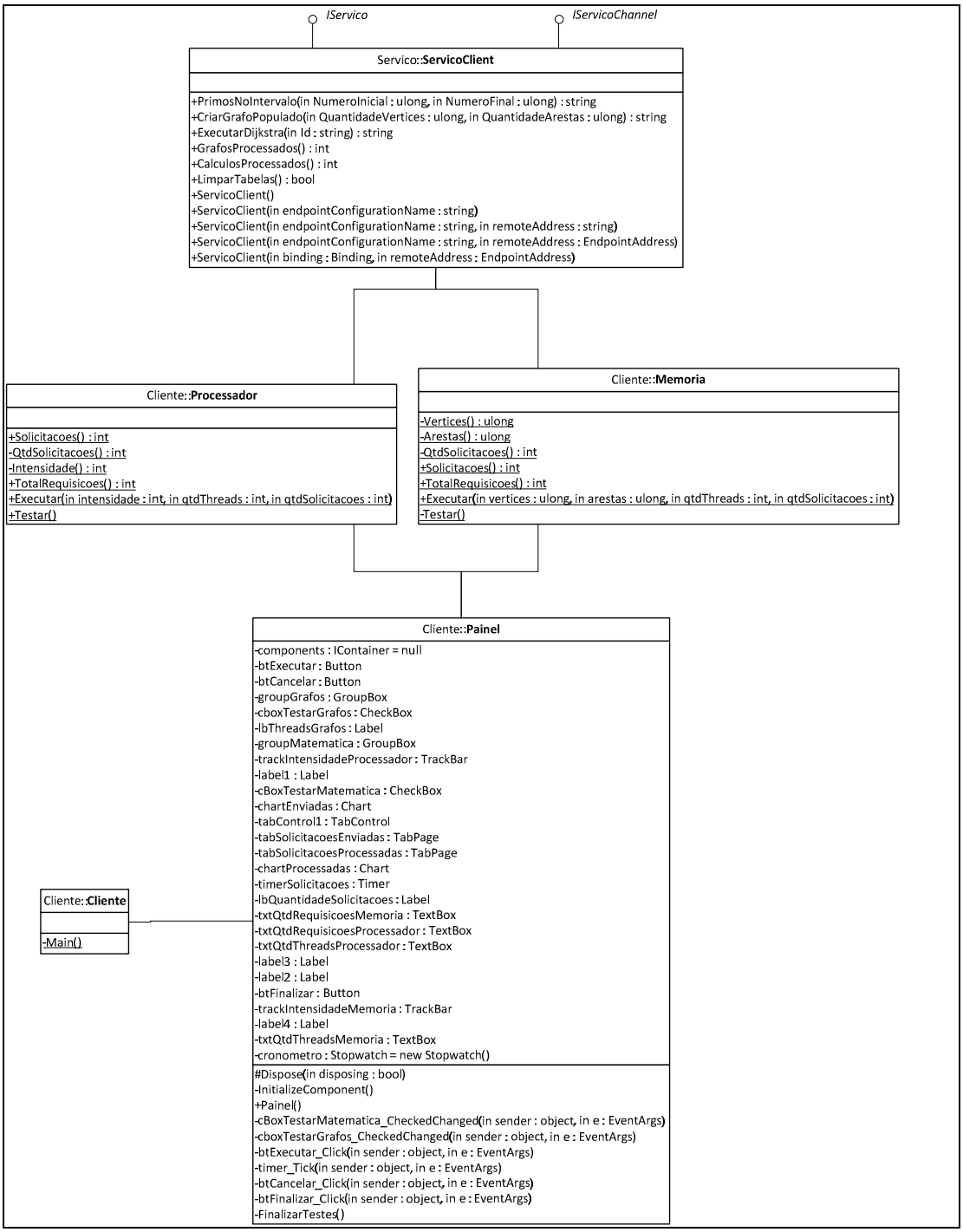


Figura 10 – Diagrama de classes do Cliente de Testes

3.3.2.2 Servidor de Requisições

A classe `Serviço` implementa a interface `IServico` e disponibiliza os contratos a serem consumidos através do *web service.* Ao receber as requisições, a classe `Serviço` formata a requisição utilizando a classe `Mensagem`, inserindo a mensagem na fila de mensagens. A fila de mensagens utiliza a estrutura Azure Queue do Windows Azure.

A classe `Servidor` implementa a classe abstrata `RoleEntryPoint` pertencente ao SDK do Windows Azure. Desta maneira ao executar o projeto no Windows Azure, a classe `Servidor` executará o método `OnStart()`, que irá inicializar o Azure Storage através da chamada do método `Inicializa Armazenamento()`.

A classe `Conexao` implementa a conexão das entidades ao Azure Table, onde estão as tabelas de dados. Através dela, a classe `Servico` disponibiliza os contratos `LimparTabelas()`, `TotalGrafos()` e `TotalCalculos()`, que efetuam ações diretamente nas entidades criadas no Azure Table.

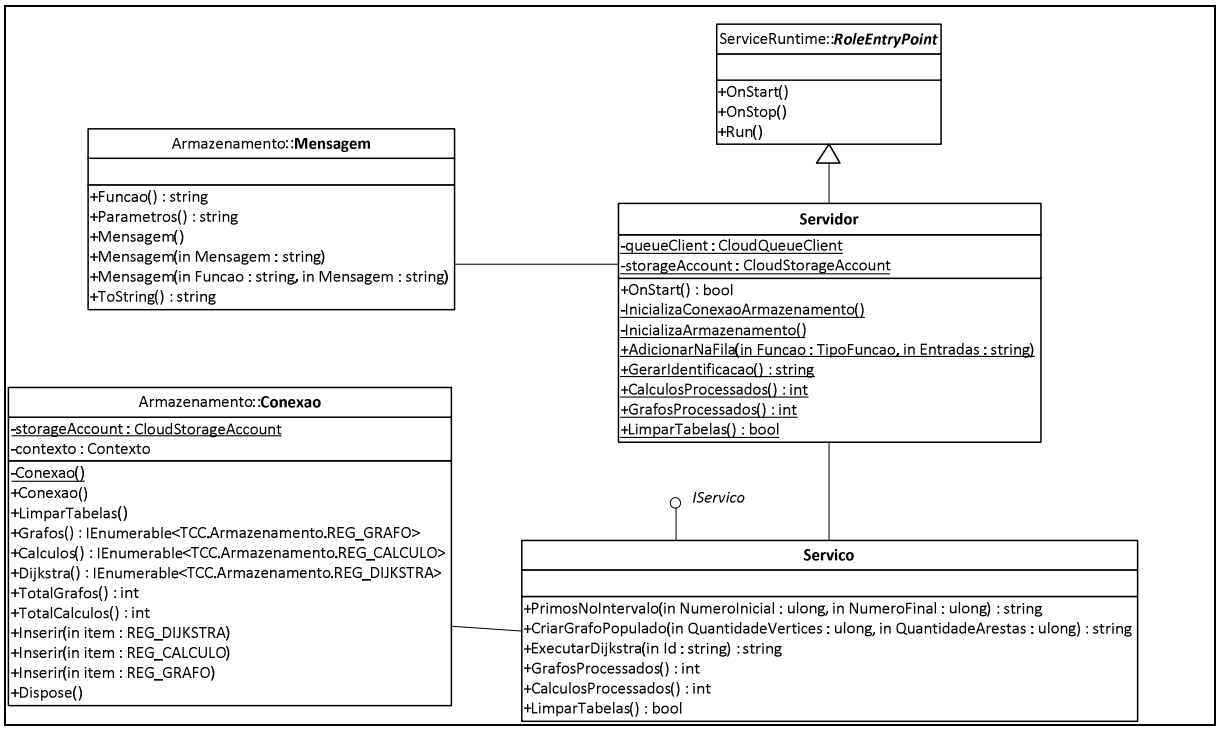


Figura 11 – Diagrama de classes do `Servidor de Requisições`

3.3.2.3 Executor de Testes

A figura 12 apresenta o diagrama de classes que implementam o protótipo de execução

de testes, responsável pelo processamento das requisições armazenadas na fila de mensagens.

A classe `Robo` implementa a classe abstrata `RoleEntryPoint` executando o método `Onstart()` assim que o projeto é publicado no Windows Azure. O método `Run()` é sobrescrito, de maneira que a classe fique em um *loop* lendo mensagens da fila. As mensagens da fila são processadas pela classe `Executor`, responsável por processar as mensagens corretamente utilizando as classes de negócio adequadas. No caso do protótipo em questão, as regras de negócios estão contidas na classe `Matematica` e `Grafo`.

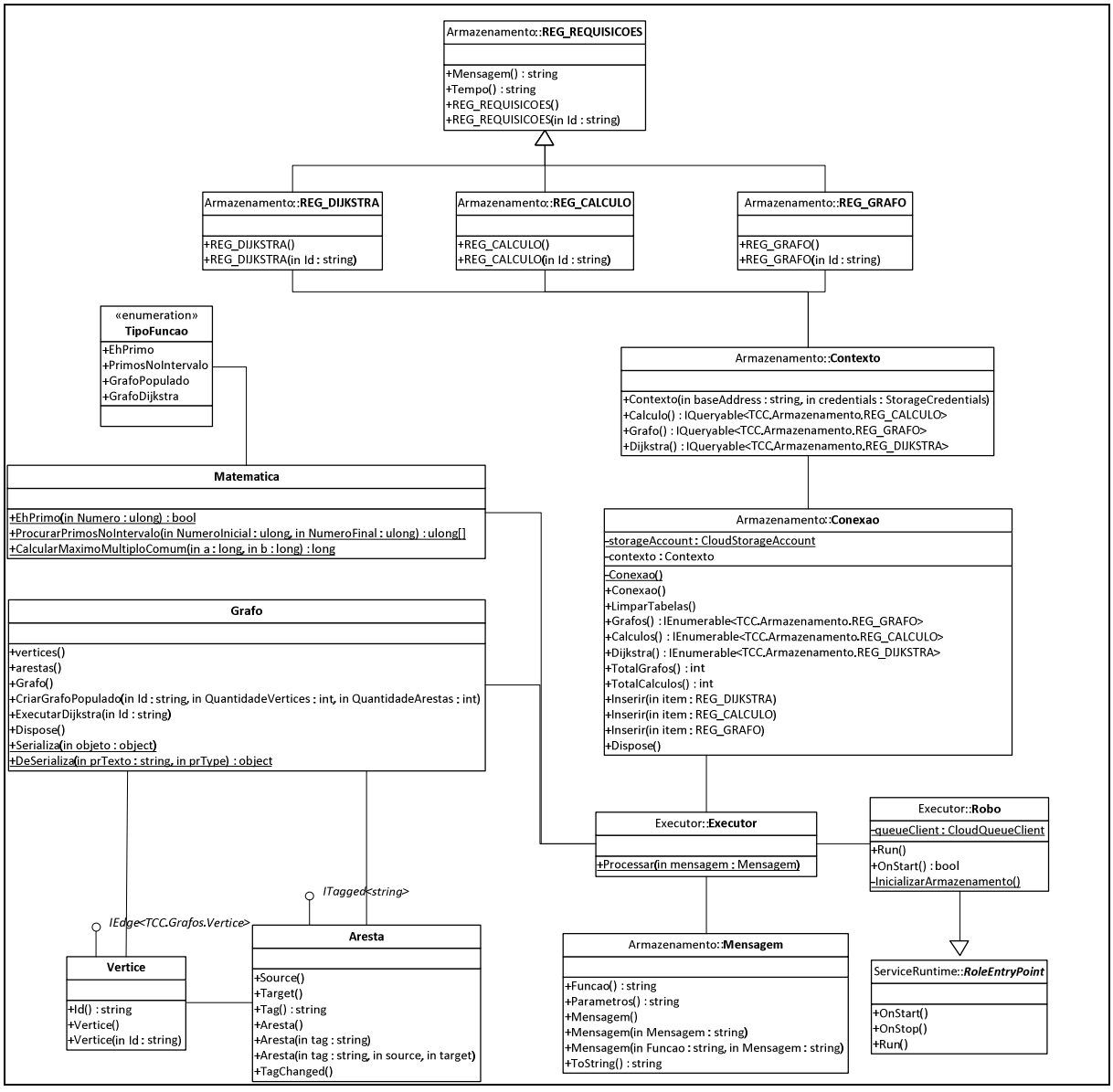


Figura 12 – Diagrama de classes do `Executor de Testes`

As classes `Grafo` e `Matematica` executam testes de memória e processador respectivamente. A classe `Grafo` cria grafos e executa o algoritmo de Dijkstra. Quando o grafo possui um grande número de vértices e arestas, ele utiliza consideravelmente a memória do computador, e ao executar o algoritmo de Dijkstra, grandes quantidades de memória são

alocadas e liberadas constantemente, criando um cenário ideal para testes de memória.

A classe `Matematica` executa a consulta de números primos em um intervalo. Para tal são efetuados cálculos matemáticos de divisão constantemente, exigindo grande parcela de tempo do processador, criando um cenário propício a testes de processamento.

3.3.2.4 Monitor de Testes

A figura 13 representa as classes do projeto que monitora o serviço hospedado na nuvem. A classe `Painel` implementa a tela para visualização das configurações e também as quantidades de mensagens contidas na fila de mensagens e quantidade de executores processando os testes. A classe `Instancias` é responsável por adicionar e remover novas instâncias de máquinas virtuais para execução dos testes.

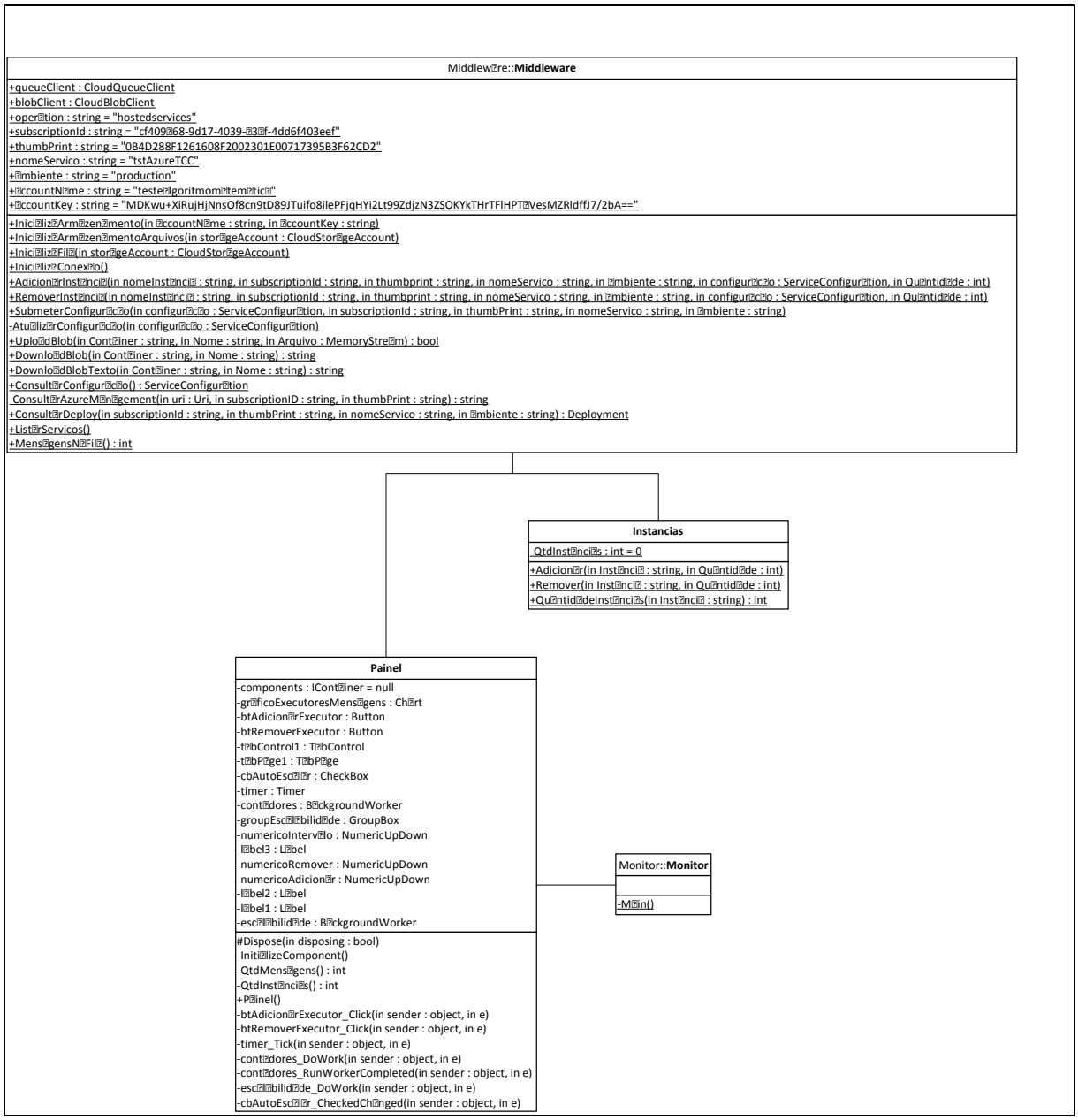


Figura 13 – Diagrama de classes do `Monitor de Testes`

Por fim a classe `Middleware` é responsável pela comunicação com o ambiente do Windows Azure através de requisições REST. Para efetuar as requisições REST, é necessario a adicionar o certificado digital do computador que efetua a requisicao (Anexo C). Ela é responsável por efetuar *uploads* e *downloads* do arquivo de configuração contido na estrutura de armazenamento de arquivos e também de submeter as novas configurações ao Windows Azure para que seja executada a adição ou remoção de instâncias de máquinas virtuais.

3.3.3 Diagrama de sequência

Nesta seção são apresentados os diagramas de sequência do Protótipo de teste de escalabilidade, com o objetivo de exibir a troca de mensagem entre suas classes para a realização de suas tarefas. Na seção 3.3.3.1 será detalhado o diagrama dos protótipos de `Cliente de Testes` e `Servidor de Requisições`. Na seção 3.3.3.2 será detalhado o diagrama de sequência do `Monitor de Testes`. Na seção 3.3.3.3 será detalhado o diagrama de sequência do `Executor de Testes`.

3.3.3.1 Cliente de Testes e Servidor de Requisições

As classes `Memoria` e `Processador` executam suas ações simultaneamente, pois elas são executadas em *threads* distintas, permitindo os testes em paralelo. A classe `Servico` recebe as requisições e as adiciona na fila de mensagens através da classe `Servidor`, responsável por criar uma identificação e formatar a requisição para ser inserida na fila de mensagens.

A figura 14 apresenta o diagrama de sequência do `Cliente de Testes` e do `Servidor de Requisições`.

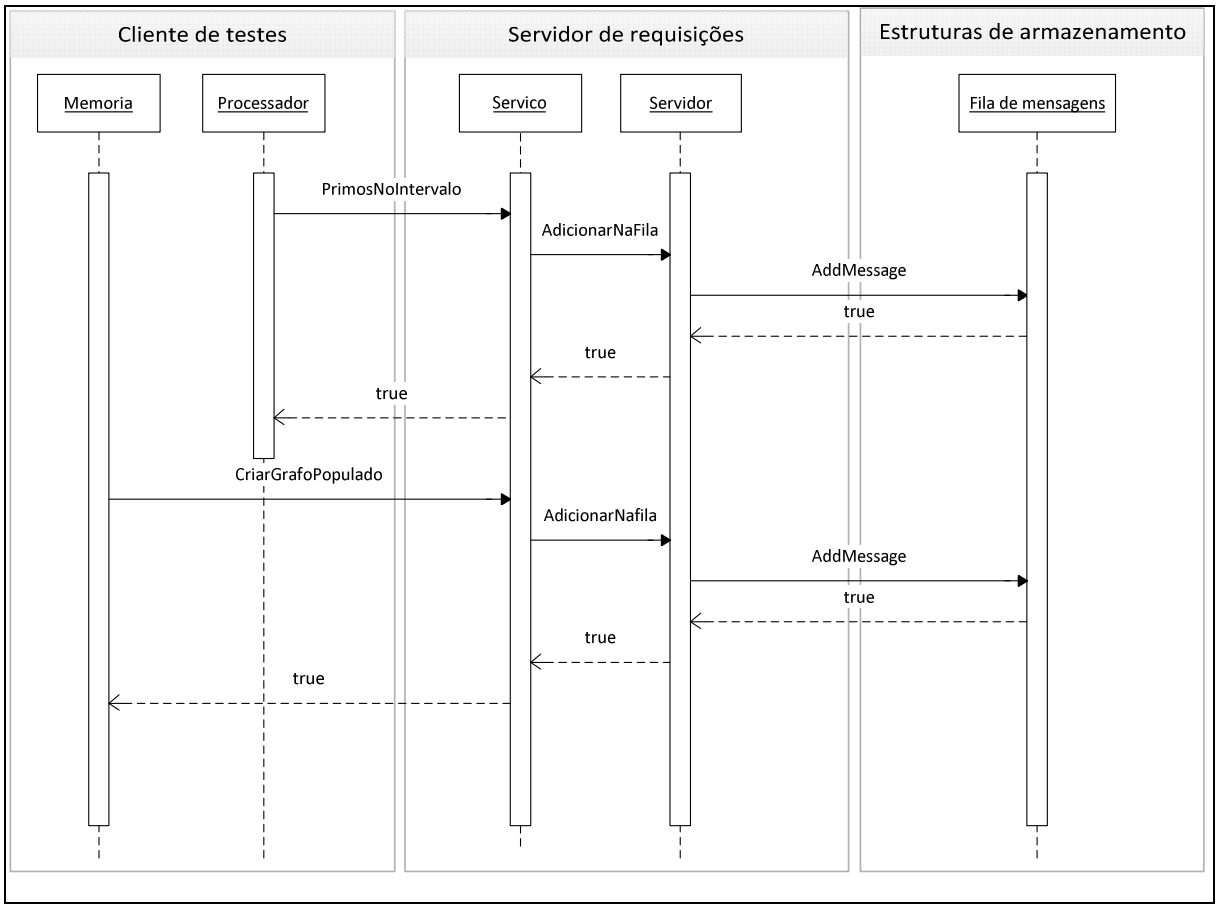


Figura 14 – Diagrama de sequência do `Cliente de Testes` e `Servidor de Requisições`

3.3.3.2 Monitor de Testes

No diagrama apresentado pela figura 15 está sendo exibido o processo de escalabilidade do protótipo. A classe `Painel` permanece em um *loop* na qual constantemente verifica a quantidade de mensagens na fila e a quantidade de instâncias de `Executores de Testes`.

A área retangular nomeada `Auto-escalar` representa o *flag* indicando se deve ou não aplicar auto escalabilidade que é parametrizada na tela do `Monitor de Testes`. Neste processo, após verificar a quantidade de mensagens na fila, o `Painel` irá solicitar a adição de instâncias ou remoção de acordo com os parâmetros configurados na tela do `Monitor de Testes`.

A funcionalidade de adicionar e remover instâncias também podem ser executadas através de dois botões visíveis na tela do `Monitor de Testes`.

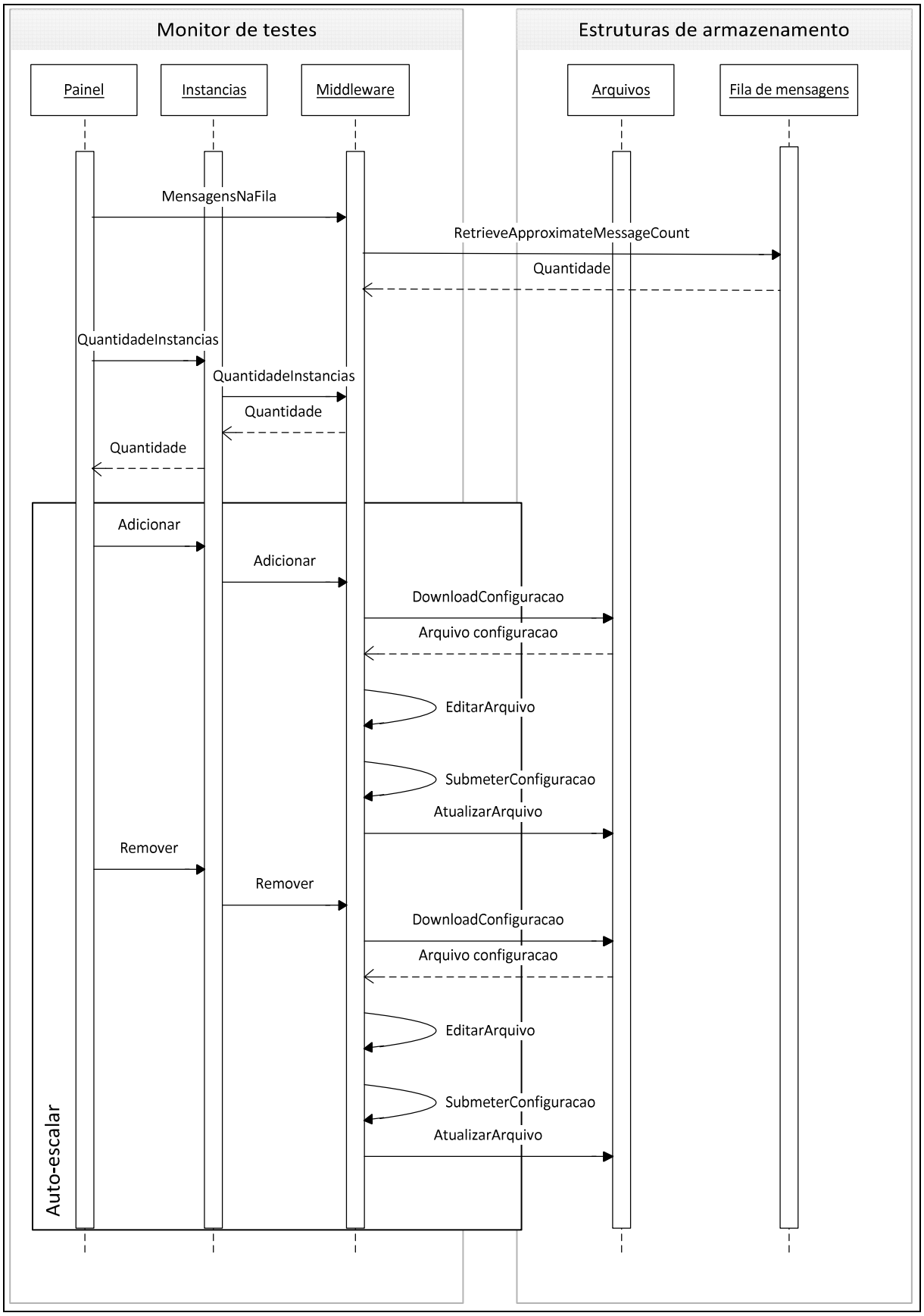


Figura 15 – Diagrama de sequência do monitoramento do protótipo

3.3.3.3 Executor de Testes

A classe `Robo` permanece em um *loop* e a cada 3 segundos retira uma mensagem da fila e a envia para a classe `Executor` que por sua vez, executa o teste solicitado. Ao finalizar o teste, informações tais como nome do teste e tempo de execução são armazenadas nas tabelas de dados. A figura 16 apresenta do diagrama de sequência do `Executor de Testes`.

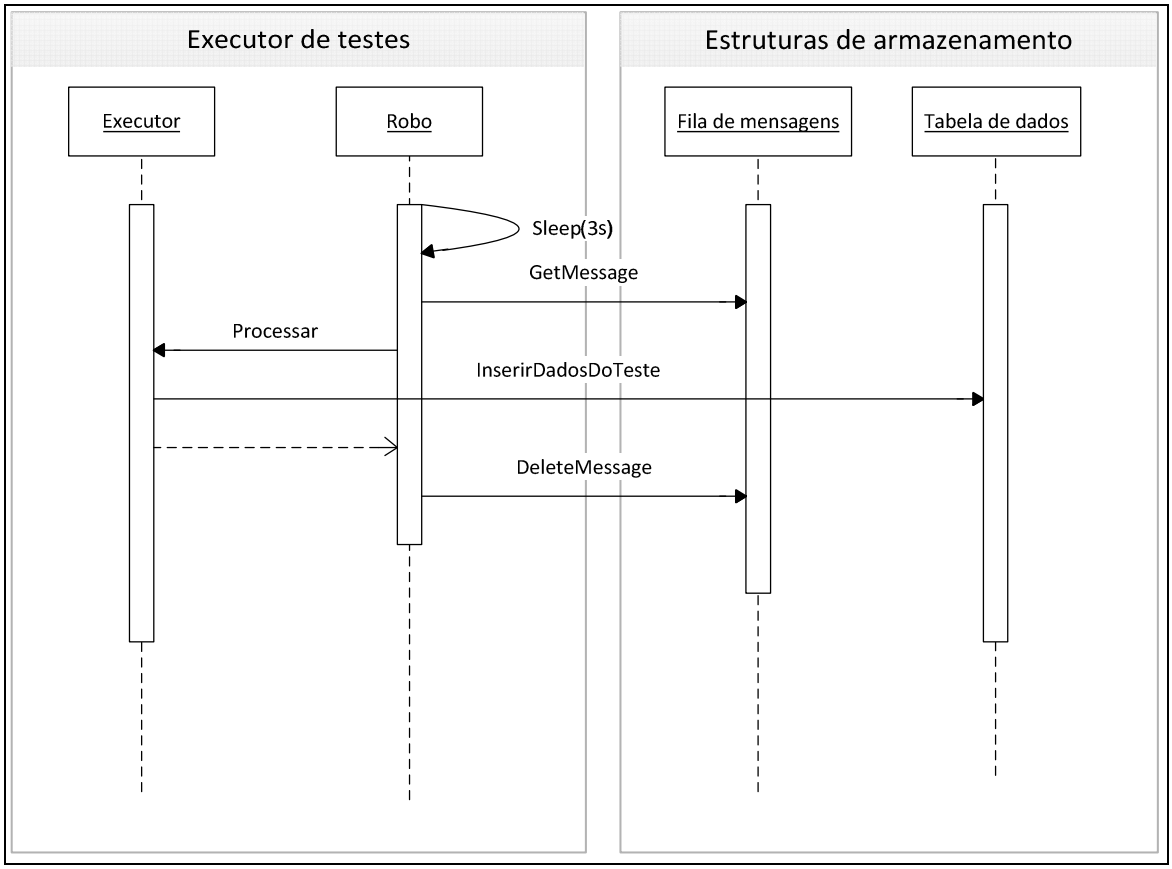


Figura 16 – Diagrama de sequência do processamento do protótipo

A figura 17 detalha o diagrama de sequência dos testes que foram utilizados para serem executados no Windows Azure.

A classe `Executor` é responsável por executar o teste solicitado, podendo ser a criação de grafos e o caminhamento no mesmo, como a execução de cálculos matemáticos através da busca de números primos em um intervalo.

A classe `Grafo` efetua a conversão objeto contendo o grafo, para um arquivo no formato `XML` a fim de consumir os recursos de memória disponíveis. A execução do algoritmo de Dijkstra também contribui para o alto consumo de memória.

A classe `Matematica` calcula números primos em um determinado intervalo, elevando o uso do processador a 100%, visto que os cálculos de divisão utilizados no algoritmo exigem

constante processamento.

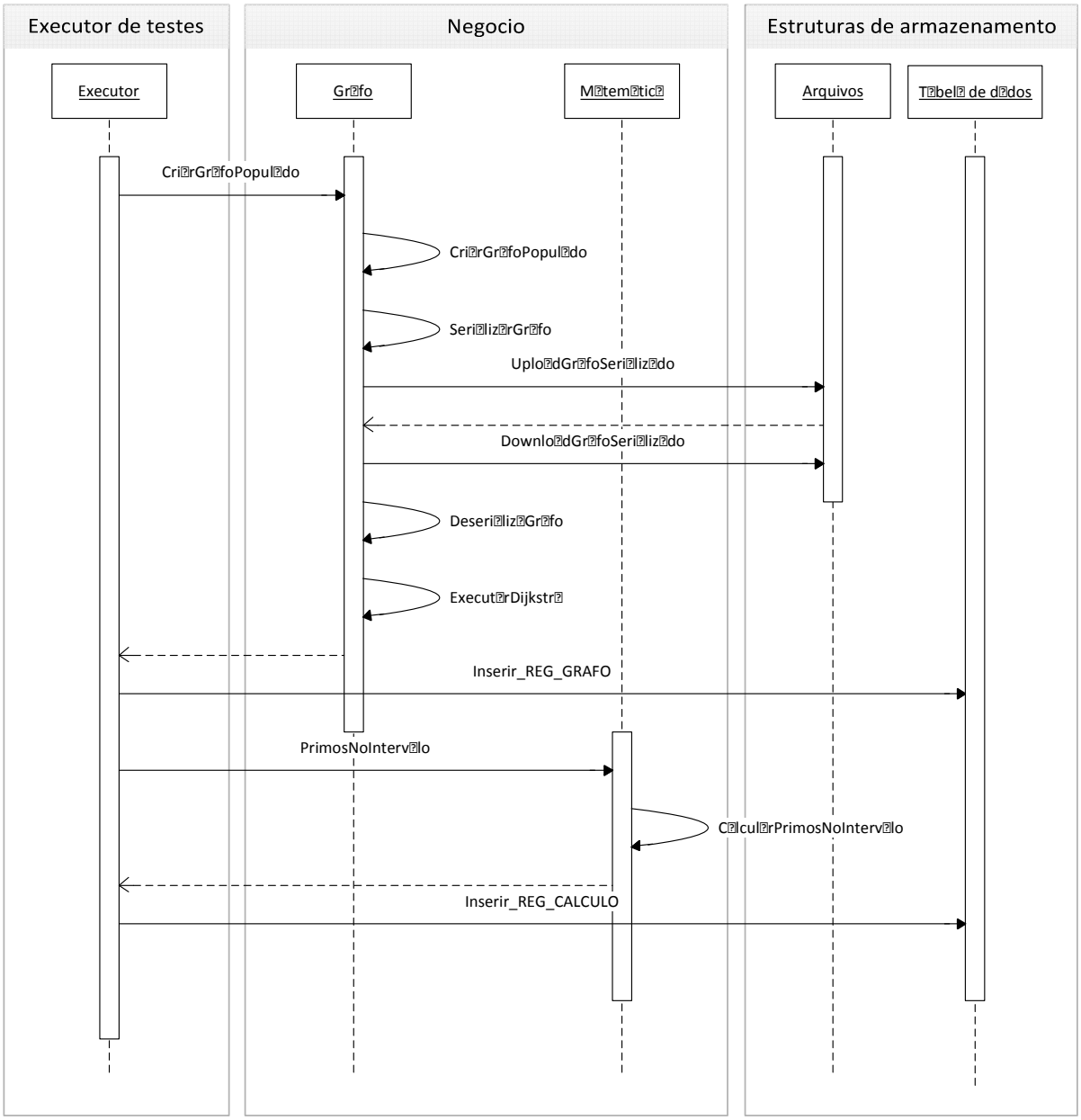


Figura 17 – Diagrama de sequência das regras de negócio

3.4 IMPLEMENTAÇÃO

Nesta seção são apresentadas informações sobre técnicas e ferramentas utilizadas para a implementação, bem como a própria implementação e cada etapa de seu desenvolvimento. Ao final, será descrita a operacionalidade do protótipo desenvolvido.

3.4.1 Técnicas e ferramentas utilizadas

A linguagem de programação utilizada para o desenvolvimento do protótipo é o C# juntamente com o *framework .NET* 4.0 com o Windows Azure SDK 1.4. O ambiente de desenvolvimento utilizado foi o Visual Studio 2010.

3.4.2 Codificação

As seções a seguir irão detalhar a codificação do protótipo de teste de escalabilidade separadamente para cada protótipo que o compõe. A seção 3.4.2.1 irá tratar da codificação do `Cliente de Testes`. A seção 3.4.2.2 irá exibir a codificação do `Servidor de Requisições`. A seção 3.4.2.3 irá exibir a codificação do `Executor de Testes` e por fim, a seção 3.4.2.4 irá exibir a codificação do `Monitor de Testes`.

3.4.2.1 Cliente de Testes

O `Cliente de Testes` aplica os testes consumindo o `Servidor de Requisições` conforme parametrização do usuário. O quadro 15 exibe um trecho do código, onde o `Cliente de Testes` configura os testes baseando-se nas parametrizações do usuário. Desta maneira, para os testes de memória, o grafo é criado com base na intensidade configurada, sendo assim, quanto mais intenso o teste, mais vértices e arestas o grafo terá.

```
94          #region [ Grafos ]
95
96          ulong vertices = 0;
97          ulong arestas = 0;
98          int qtdThreadsTesteMemoria = 0;
99          int qtdRequisicoesTesteMemoria = 0;
100
101         if (cboxTestarGrafos.Checked)
102         {
103             try
104             {
105                 vertices = ulong.Parse((100 * (trackIntensidadeMemoria.Value * 100)).ToString());
106                 arestas = ulong.Parse((100 * (trackIntensidadeMemoria.Value * 100)).ToString());
107                 qtdThreadsTesteMemoria = int.Parse(txtQtdThreadsMemoria.Text);
108                 qtdRequisicoesTesteMemoria = int.Parse(txtQtdRequisicoesMemoria.Text);
109                 Memoria.TotalRequisicoes = qtdThreadsTesteMemoria * qtdRequisicoesTesteMemoria;
110             }
111             catch (Exception ex)
112             {
113                 MessageBox.Show(ex.Message);
114                 return;
115             }
116         }
117
118         #endregion
```

Quadro 15 – Trecho de código do `Cliente de Testes` configurando testes de memória

O quadro 16 exibe o trecho de código onde o `Cliente de Testes` parametriza os testes de cálculos matemáticos. Quanto maior a intensidade configurada, maior será o intervalo a serem calculados os números primos no algoritmo.

```
68          #region [ Matematica ]
69
70          int qtdThreadsTesteProcessador = 0;
71          int qtdRequisicoesTesteProcessador = 0;
72          int intensidade = 0;
73
74          if (cBoxTestarMatematica.Checked)
75          {
76              try
77              {
78                  qtdThreadsTesteProcessador = int.Parse(this.txtQtdThreadsProcessador.Text);
79                  qtdRequisicoesTesteProcessador = int.Parse(txtQtdRequisicoesProcessador.Text);
80                  intensidade = trackIntensidadeProcessador.Value;
81                  Processador.TotalRequisicoes = qtdThreadsTesteProcessador * qtdRequisicoesTesteProcessador;
82              }
83              catch (Exception ex)
84              {
85                  MessageBox.Show(ex.Message);
86                  return;
87              }
88          }
89
```

Quadro 16 – Trecho de código do `Cliente de Testes` configurando testes de processamento

O quadro 17 exibe o trecho de código onde os testes são disparados. Os testes de memória e processador ocorrem paralelamente, pois são executados através de *threads* distintas.

```
120             #region [ Execuções ]
121
122             if (cBoxTestarMatematica.Checked)
123             {
124                 Action<int, int, int> actionThreadTesteProcessador = Processador.Executar;
125                 Thread threadTesteProcessador = new Thread(() =>
126                     actionThreadTesteProcessador(intensidade, qtdThreadsTesteProcessador, qtdRequisicoesTesteProcessador));
127                 threadTesteProcessador.Start();
128             }
129
130             if (cboxTestarGrafos.Checked)
131             {
132                 Action<ulong, ulong, int, int> actionTesteMemoria = Memoria.Executar;
133                 Thread threadTesteMemoria = new Thread(() =>
134                     actionTesteMemoria(vertices, arestas, qtdThreadsTesteMemoria, qtdRequisicoesTesteMemoria));
135                 threadTesteMemoria.Start();
136             }
137
138             #endregion
139         }
```

Quadro 17 – Trecho de código do `Cliente de Testes` iniciando execução dos testes

O quadro 18 exibe o trecho de código onde são criadas as threads para testes de memória de acordo com a quantidade parametrizada. Cada thread executa o método Testar(), que possui as definições e proxys para efetuar requisição ao `Servidor de Requisições`.

```
17          public static void Executar(ulong vertices, ulong arestas, int qtdThreads, int qtdSolicitacoes)
18          {
19              Solicitacoes = 0;
20              QtdSolicitacoes = qtdSolicitacoes;
21              Vertices = vertices;
22              Arestas = arestas;
23
24              Thread[] threads = new Thread[qtdThreads];
25
26              for (int i = 0; i < qtdThreads; i++)
27              {
28                  Thread.Sleep(1000);
29                  threads[i] = new Thread(Testar);
30                  threads[i].IsBackground = true;
31                  threads[i].Start();
32              }
33          }
34
35          private static void Testar()
36          {
37              Servico.ServicoClient servico = new Servico.ServicoClient();
38
39              for (int i = 0; i < QtdSolicitacoes; i++)
40              {
41                  string id = servico.CriarGrafoPopulado(Vertices, Arestas);
42                  Solicitacoes++;
43              }
44          }
```

Quadro 18 – Trecho de código do cliente de exibindo as *threads* dos testes de memória

O quadro 19 exibe o código onde são criadas as *threads* para testes de memória. A codificação é semelhante ao teste de memória, porém o serviço que é consumido do *web service* é outro.

```
17      public static void Executar(int intensidade, int qtdThreads, int qtdSolicitacoes)
18      {
19          Solicitacoes = 0;
20          Intensidade = intensidade;
21          QtdSolicitacoes = qtdSolicitacoes;
22
23          Thread[] threads = new Thread[qtdThreads];
24
25          for (int i = 0; i < qtdThreads; i++)
26          {
27              Thread.Sleep(1000);
28              threads[i] = new Thread(Testar);
29              threads[i].IsBackground = true;
30              threads[i].Start();
31          }
32      }
33
34      public static void Testar()
35      {
36          Servico.ServicoClient servico = new Servico.ServicoClient();
37
38          for (int i = 0; i < QtdSolicitacoes; i++)
39          {
40              ulong final = 1000 * ulong.Parse(Intensidade.ToString());
41              servico.PrimosNoIntervalo(100, final);
42              Thread.Sleep(500);
43              Solicitacoes++;
44          }
45      }
```

Quadro 19 – Trecho de código exibindo as *threads* de testes

3.4.2.2 Servidor de Requisições

O `Servidor de Requisições` é um projeto do tipo `WCF Service Web Role`, sendo ele responsável por implementar o serviço e os contratos que serão disponibilizados para consumo. O quadro 20 ilustra os contratos disponibilizados pelo `Servidor de Requisições`.

```
11      [ServiceContract]
12      public interface IServico
13      {
14          [OperationContract]
15          string PrimosNoIntervalo(ulong NumeroInicial, ulong NumeroFinal);
16
17          [OperationContract]
18          string CriarGrafoPopulado(ulong QuantidadeVertices, ulong QuantidadeArestas);
19
20          [OperationContract]
21          string ExecutarDijkstra(string Id);
22
23          [OperationContract]
24          int GrafosProcessados();
25
26          [OperationContract]
27          int CalculosProcessados();
28
29          [OperationContract]
30          bool LimparTabelas();
31
32      }
```

Quadro 20 – Contratos disponibilizados pelo *web service*

Todas as requisições que são enviadas ao *web service*, são formatadas para um padrão

de mensagens e são adicionadas na fila de mensagens do Azure Storage. Ao haver sucesso da adição da mensagem na fila, o consumidor recebe a resposta `true`.

O quadro 21 exibe a implementação da interface `IServico`, onde nela são enviados as entradas da requisição e o tipo de função, representado pelo enumerador `TipoFuncao`.

```
14      [ServiceBehavior(AddressFilterMode = AddressFilterMode.Any)]
15      public class Servico : IServico
16      {
17          public string PrimosNoIntervalo(ulong NumeroInicial, ulong NumeroFinal)
18          {
19              string id = Servidor.GerarIdentificacao();
20              Servidor.AdicionarNaFila(TipoFuncao.PrimosNoIntervalo, string.Format("{0},{1},{2}", id, NumeroInicial, NumeroFinal));
21              return id;
22          }
23
24          public string CriarGrafoPopulado(ulong QuantidadeVertices, ulong QuantidadeArestas)
25          {
26              string id = Servidor.GerarIdentificacao();
27              Servidor.AdicionarNaFila(TipoFuncao.GrafoPopulado, string.Format("{0},{1},{2}", id, QuantidadeVertices, QuantidadeArestas));
28              return id;
29          }
30
31          public string ExecutarDijkstra(String IdGrafo)
32          {
33              string id = Servidor.GerarIdentificacao();
34              Servidor.AdicionarNaFila(TipoFuncao.GrafoDijkstra, string.Format("{0},{1}", id, IdGrafo));
35              return id;
36          }
```

Quadro 21 – Trecho de implementação de um contrato disponibilizado para o *web service*

A classe `Servidor` efetivamente adiciona a requisição no formato desejado na fila de mensagens. Ela utiliza classe `Mensagem` para formatar o tipo de função e as entradas da requisição. As mensagens adicionadas na fila possuem um tempo finito de vida, padronizado pelo próprio Azure Storage. Como o método de adição de mensagem permite especificar o tempo e vida da mensagem, no protótipo de desenvolvido adotou-se o tempo de 7 horas. O quadro 22 ilustra o trecho do código em que é feito a adição de uma mensagem na fila de mensagens.

```
50          public static void AdicionarNaFila(TipoFuncao Funcao, String Entradas)
51          {
52              InicializaArmazenamento();
53              Mensagem msg = new Mensagem(Funcao, Entradas);
54              CloudQueueMessage mensagem = new CloudQueueMessage(msg.ToString());
55              CloudQueue Fila = queueClient.GetQueueReference("fila");
56              Fila.AddMessage(mensagem, new TimeSpan(7,0,0));
57          }
```

Quadro 22 – Trecho do código onde são inseridas as mensagens na fila

3.4.2.3 Executor de Testes

O `Executor de Testes` é uma instância do tipo *worker role,* sendo ele responsável pelo processamento de todas as requisições recebidas pelo *web service* e pelo armazenamento dos resultados na tabela de dados, uma *table* localizada no Azure Storage.

O `Executor de Testes` funciona com uma classe `Robo` que em um *loop* infinito, a

cada 3 segundos retira uma mensagem da fila, e o envia para a classe `Executor` para que seja processada corretamente. O quadro 23 ilustra o código fonte da classe `Robo`.

```
20          public override void Run()
21          {
22              InicializarArmazenamento();
23
24              CloudQueue Fila = queueClient.GetQueueReference("fila");
25
26              while (true)
27              {
28                  Thread.Sleep(new TimeSpan(0, 0, 3));
29                  CloudQueueMessage FilaMsg = Fila.GetMessage(new TimeSpan(1, 0, 0));
30                  if (FilaMsg != null)
31                  {
32
33                      Mensagem mensagem = new Mensagem(FilaMsg.AsString);
34                      try
35                      {
36                          Executor.Processar(mensagem);
37                          Fila.DeleteMessage(FilaMsg);
38                      }
39                      catch(Exception ex)
40                      {
41                          Trace.WriteLine(ex.Message);
42                      }
43                  }
44              }
45          }
```

Quadro 23 – Trecho de código do processamento das mensagens da fila

A classe `Executor` é responsável por encaminhar corretamente as solicitações às suas respectivas regras de negócio. Ela recebe a solicitação encapsulada através da classe `Mensagem` e identifica o tipo de processamento através do enumerador `TipoFuncao`. Após avaliar tipo de teste, a mensagem é enviada para a classe de negócio, sendo ela a classe `Matematica` ou `Grafo`. O processo de encaminhamento da mensagem para processamento é ilustrado no quadro 24.

```
15  public class Executor
16  {
17      public static void Processar(Mensagem mensagem)
18      {
19          [ Inicialização ]
26
27          #region [ Funcoes ]
28
29          switch (teste)
30          {
31              case TipoFuncao.GrafoPopulado:
32                  {
33                      #region [ GrafoPopulado ]
34
35                      Stopwatch cronometro = new Stopwatch();
36                      cronometro.Start();
37                      using (Grafos.Grafo grafo = new Grafo())
38                      {
39                          grafo.CriarGrafoPopulado(idRequisicao, int.Parse(parametros[1]), int.Parse(parametros[2]));
40                      }
41                      cronometro.Stop();
42
43                      [ Inserir registro ]
57
58                      #endregion
59
60                  } break;
61              case TipoFuncao.GrafoDijkstra:
62                  {
63                      [ GrafoDijkstra ]
82                  } break;
83              case TipoFuncao.PrimosNoIntervalo:
84                  {
85                      [ PrimoNoIntervalo ]
103                 } break;
104         }
105         #endregion
106     }
```

Quadro 24 – Trecho de código do encaminhamento das solicitações

Após executar o processamento da requisição e obter o resultado, é inserido um registro no Azure Storage na estrutura *table*. O quadro 25 ilustra a inserção do registro na table `Registro` que pertence ao Azure Storage.

```
10  public class Processador
11  {
12
13      public static void Processar(Mensagem mensagem)
14      {
15          String resultado = String.Empty;
16
17          [ Funcoes ]
39
40          #region [ Inserir registro na table ]
41
42          Registro registro = new Registro()
43          {
44              Resultado = resultado,
45              Mensagem = mensagem.ToString()
46          };
47
48          using (Conexao conexao = new Conexao())
49          {
50              conexao.Inserir(registro);
51          }
52
53          #endregion
54      }
```

Quadro 25 – Trecho de código de inserção de registro

O gerenciamento da persistência dos dados em *tables* do Azure está contido no projeto `Armazenamento`, e nele é feito todo o procedimento de criação da tabela, definição da

entidade, e criação do contexto para manipulação de dados. O quadro 26 ilustra a criação da entidade `Registro` para inserção dos resultados no Azure Storage.

```
7  namespace Armazenamento
8  {
9      public class Registro : TableServiceEntity
10     {
11         public string Mensagem { get; set; }
12         public string Resultado { get; set; }
13
14         public Registro()
15         {
16             PartitionKey = DateTime.Now.ToString("MMddyyyy");
17             RowKey = string.Format("{0:10}_{1}", DateTime.MaxValue.Ticks - DateTime.Now.Ticks, Guid.NewGuid());
18         }
19     }
20 }
```

Quadro 26 – Trecho de código de criação da entidade `Registro`

3.4.2.4 Monitor de Testes

O quadro 27 ilustra o *loop* de monitoramento da fila de mensagens. O *loop* ocorre através do método `timer_Tick`, pois ele é executado em um período definido no componente `Timer`. O método executa de forma assíncrona o método `contadores_DoWork`, que consulta a quantidade de instâncias de executores de teste e quantidade de mensagens na fila.

```
35         private void timer_Tick(object sender, EventArgs e)
36         {
37             if(!contadores.IsBusy)
38                 contadores.RunWorkerAsync();
39         }
40
41         private void contadores_DoWork(object sender, DoWorkEventArgs e)
42         {
43             QtdMensagens = Middleware.Middleware.MensagensNaFila();
44             QtdInstancias = Instancias.QuantidadeInstancias("TCC.Executor");
45         }
46
47         private void contadores_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
48         {
49             graficoExecutoresMensagens.Series["Mensagens"].Points.Add(QtdMensagens);
50             graficoExecutoresMensagens.Series["Executores"].Points.Add(QtdInstancias);
51         }
52
```

Quadro 27 – Trecho de código do `Monitor de Testes`

O quadro 28 ilustra o método que aplica a auto escalabilidade, isto se ela estiver habilitada. O método `escalabilidade_DoWork` é executado de forma assíncrona, efetuando a verificação do tamanho da fila de mensagens e confrontando com os dados configurados no monitor de testes, avaliando a necessidade de alterar o número de instâncias de `Executores de Testes`.

```
59          private void escalabilidade_DoWork(object sender, DoWorkEventArgs e)
60          {
61              while (cbAutoEscalar.Checked)
62              {
63                  Thread.Sleep(new TimeSpan(0, int.Parse(numericoIntervalo.Value.ToString()), 0));
64
65                  int qtdMensagens = Middleware.Middleware.MensagensNaFila();
66
67                  if (numericoAdicionar.Value <= qtdMensagens)
68                  {
69                      Instancias.Adicionar("TCC.Executor", 1);
70                  }
71                  else
72                  {
73                      if (numericoRemover.Value >= qtdMensagens)
74                      {
75                          Instancias.Remover("TCC.Executor", 1);
76                      }
77                  }
78              }
79          }
```

Quadro 28 – Trecho de código de avaliação de escalabilidade

Para poder alterar o número de instâncias das máquinas virtuais, é necessário alterar o arquivo que define as configurações dos serviços no Windows Azure, ou seja, o arquivo `ServiceConfiguration.Cloud.csfg`. É gravada uma cópia do arquivo no Azure blob, com o objetivo de ser alterado e ser submetido novamente ao Windows Azure, de modo que propicie as novas configurações enquanto os serviços permanecem em execução. O quadro 29 ilustra o trecho do código fonte onde é efetuado o *download* do arquivo de configuração do Azure blob para posterior alteração.

```
200         /// <summary>
201         /// Consulta as configurações do serviço no BLOB
202         /// </summary>
203         /// <returns>Configurações do serviço</returns>
204         public static ServiceConfiguration ConsultarConfiguracao()
205         {
206             CloudBlob Blob = blobClient.GetBlobReference("configuracao");
207             CloudBlobContainer container = new CloudBlobContainer(Blob.Uri.AbsoluteUri);
208             ServiceConfiguration configuracao = new ServiceConfiguration();
209
210             foreach (CloudBlob item in container.ListBlobs())
211             {
212                 item.FetchAttributes();
213                 if (item.Uri.Segments.Last<string>() == "ServiceConfiguration.xml")
214                 {
215                     configuracao = ServiceConfiguration.Deserialize(item.DownloadText());
216                     break;
217                 }
218             }
219
220             return configuracao;
221         }
```

Quadro 29 – Trecho de código da leitura do arquivo de configuração

O quadro 30 ilustra o trecho de código, onde é alterado o arquivo de configuração para incremento de uma instância de máquina virtual. O parâmetro `role.Instances.count` contém o número de máquinas virtuais que estão executando. Desta maneira, é alterado o parâmetro referente à máquina virtual dos `Executores de Teste`.

```
103         public static void AdicionarInstancia(string nomeInstancia, string subscriptionId,
104             string thumbprint, string nomeServico,
105             string ambiente, ServiceConfiguration configuracao, int Quantidade = 1)
106         {
107             if (configuracao == null)
108             {
109                 configuracao = ConsultarConfiguracao();
110             }
111 
112             foreach (ServiceConfigurationRole role in configuracao.Role)
113             {
114                 if (role.name == nomeInstancia)
115                 {
116                     role.Instances.count = Convert.ToByte((Convert.ToInt32(role.Instances.count.ToString()) + Quantidade));
117                     break;
118                 }
119             }
```

Quadro 30 – Trecho de código de inserção de máquina virtual

Após feitas as alterações desejadas no arquivo de configuração, ele deve ser enviado ao Windows Azure Management para que sejam efetuadas as alterações no ambiente de execução.

Devido às regras de segurança do Windows Azure, o `Monitor de Testes` é um protótipo que executa localmente. Para que haja sucesso na comunicação entre o `Monitor de Testes` e o Windows Azure, o `Monitor de Testes` deve estar executando em um computador que contenha os mesmos certificados de segurança adicionados no Windows Azure. Isto ocorre porque ao efetuar as requisições REST ao Windows Azure Management, é necessário adicionar os certificados de segurança na solicitação HTTP.

O quadro 31 exibe um trecho do código disponibilizado pela Microsoft, disponível em http://msdn.microsoft.com/pt-br/library/gg651127.aspx, detalhando a adição dos certificados digitais em uma chamada REST ao serviços de gerenciamento do Windows Azure.

```
112         public static void SubmeterConfiguracao(ServiceConfiguration configuracao,
113             string subscriptionId,
114             string thumbPrint,
115             string nomeServico,
116             string ambiente)
117         {
118             // X.509 certificate variables.
119             X509Store certStore = null; X509Certificate2Collection certCollection = null; X509Certificate2 certificate = null;
120             // Request and response variables.
121             HttpWebRequest httpWebRequest = null; HttpWebResponse httpWebResponse = null;
122             // Stream variables.
123             Stream responseStream = null;
124 
125             // Open the certificate store for the current user.
126             certStore = new X509Store(StoreName.My, StoreLocation.CurrentUser);
127             certStore.Open(OpenFlags.ReadOnly);
128 
129             // Find the certificate with the specified thumbprint.
130             certCollection = certStore.Certificates.Find(X509FindType.FindByThumbprint, thumbPrint, false);
131 
132             // Close the certificate store.
133             certStore.Close();
134 
135             // Check to see if a matching certificate was found.
136             if (0 == certCollection.Count)
137                 throw new Exception("No certificate found containing thumbprint " + thumbPrint);
138 
139             // A matching certificate was found.
140             certificate = certCollection[0];
141 
142             Uri uri = new Uri(string.Format("{0}/{1}/{2}/{3}/{4}/deploymentslots/{5}/?comp=config",
143                 "https://management.core.windows.net", subscriptionId, "services", operation, nomeServico, ambiente));
144 
145             httpWebRequest = (HttpWebRequest)HttpWebRequest.Create(uri);
146 
147             // Add the certificate to the request.
148             httpWebRequest.ClientCertificates.Add(certificate);
```

Quadro 31 – Trecho de código do método `SubmeterConfiguracao`

O quadro 32, exibe a continuação do método `SubmenterConfiguracao`, destacando a conversão do arquivo de configuração em XML para `Base64String` para ser enviado via REST para o Windows Azure Management.

```
127             // Specify the version information in the header.
128             httpWebRequest.Headers.Add("x-ms-version", "2011-02-25");
129             httpWebRequest.Method = "POST";
130             httpWebRequest.ContentType = "application/xml";
131 
132             byte[] byt = System.Text.Encoding.UTF8.GetBytes(configuracao.Serialize());
133 
134             using (Stream requestStream = httpWebRequest.GetRequestStream())
135             {
136                 using (StreamWriter writer = new StreamWriter(requestStream))
137                 {
138                     writer.Write("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
139                     writer.Write("<ChangeConfiguration xmlns=\"http://schemas.microsoft.com/windowsazure\">");
140                     writer.Write("<Configuration>");
141                     writer.Write(Convert.ToBase64String(byt));
142                     writer.Write("</Configuration>");
143                     writer.Write("</ChangeConfiguration>");
144                 }
145             }
146 
147             // Make the call using the web request.
148             try
149             {
150                 httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();
151             }
152             catch (WebException we)
153             {
154                 throw new Exception(we.Message);
155             }
156 
157             if (httpWebResponse.StatusCode != HttpStatusCode.Accepted)
158             {
159                 throw new Exception("Status code: " + httpWebResponse.StatusCode.ToString());
160             }
161             // Parse the web response.
162             responseStream = httpWebResponse.GetResponseStream();
163             httpWebResponse.Close();
164             responseStream.Close();
165         }
```

Quadro 32 – Trecho de código do método `SubmeterConfiguracao` com *post*

Após o arquivo ser submetido ao Windows Azure Management, ele é salvo novamente no Azure blob, com o objetivo de mantê-lo atualizado com o protótipo em execução.

O quadro 33 exibe o arquivo que é submetido ao Windows Azure quando é feita uma alteração na quantidade de `Executores de Testes`. A *tag* `Instances count` indica a quantidade de instâncias, e é este valor que é alterado toda vez que é feita uma adição ou remoção de nova instância.

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceConfiguration serviceName="Prototipo"
                      xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration"
                      osFamily="1" osVersion="*">
  <Role name="TCC.Servidor">...</Role>
  <Role name="TCC.Executor">
    <Instances count="1" />
    <ConfigurationSettings>
      <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
               value="DefaultEndpointsProtocol=https;
               AccountName=testealgoritmomatematica;
               AccountKey=MDKwu+XiRujHjNnsOf8cn9tD89JTuifo
               8ilePFjqHYi2Lt99ZdjzN3ZSOKYkTHrTFlHPTaVesMZRldffJ7/2bA==" />
      <Setting name="DataConnectionString"
               value="DefaultEndpointsProtocol=https;
               AccountName=testealgoritmomatematica;
               AccountKey=MDKwu+XiRujHjNnsOf8cn9tD89JTuif
               o8ilePFjqHYi2Lt99ZdjzN3ZSOKYkTHrTFlHPTaVesMZRldffJ7/2bA==" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteForwarder.Enabled"
               value="true" />
    </ConfigurationSettings>
    <Certificates>
      <Certificate name="Microsoft.WindowsAzure.Plugins.RemoteAccess.PasswordEncryption"
                   thumbprint="0B4D288F1261608F2002301E00717395B3F62CD2"
                   thumbprintAlgorithm="sha1" />
    </Certificates>
  </Role>
</ServiceConfiguration>
```

Quadro 33 – Arquivo de configuração das instâncias executando no Windows Azure

3.4.3 Operacionalidade da implementação

Esta seção irá detalhar a operacionalidade no protótipo de teste de escalabilidade com base nos testes efetuados.

A figura 18 exibe a tela do `Cliente de Testes`. Nela são configurados os testes que serão executados. O gráfico exibido na figura abaixo está informando a quantidade de solicitações efetuadas pelo `Cliente de Testes` ao `Servidor de Requisições`.

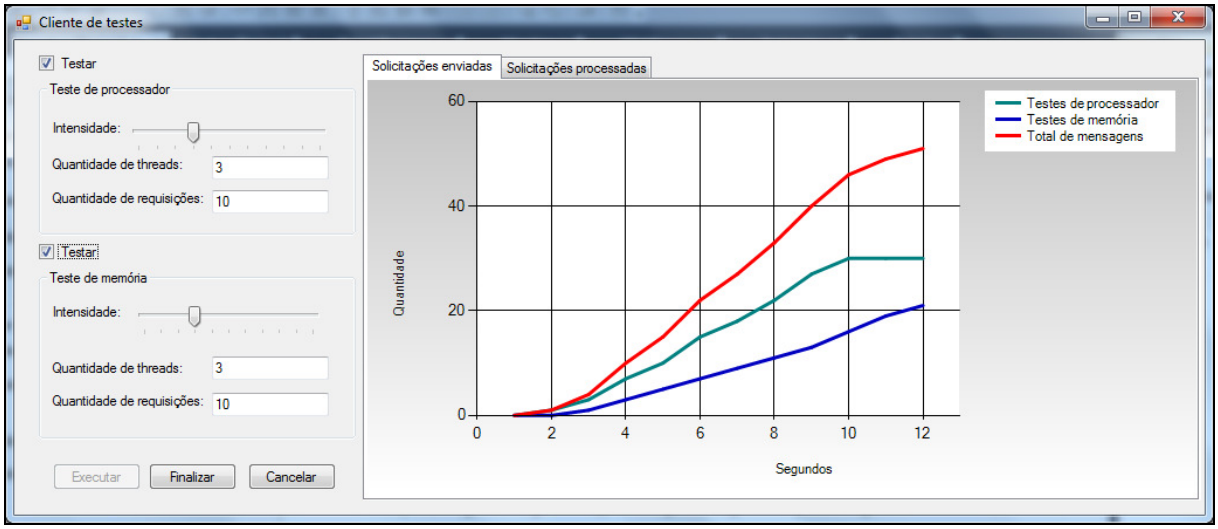


Figura 18 – Execução dos testes utilizando o `Cliente de Testes`

A figura 19 exibe a tela do `Cliente de Testes` com os gráficos de solicitações processadas, desta maneira, é possível acompanhar os testes em execução.

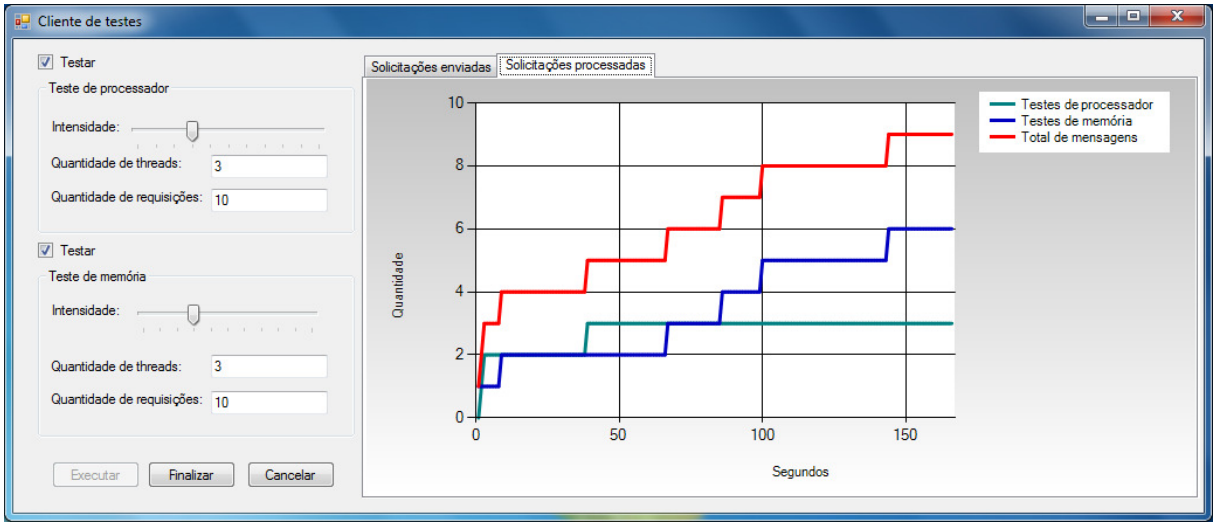


Figura 19 – `Cliente de testes` exibindo o gráfico de solicitações processadas

Enquanto o `Cliente de Testes` procede solicitando as requisições, o `Monitor de Testes` executa em paralelo medindo o tamanho da fila de mensagens e a quantidade de `Executores de Testes` que estão sendo executados.

A figura 20 exibe a tela do `Monitor de Testes`, com o gráfico da atual situação do ambiente, com a quantidade de mensagens na fila de mensagens e a quantidade de instâncias de máquinas virtuais executando os testes.

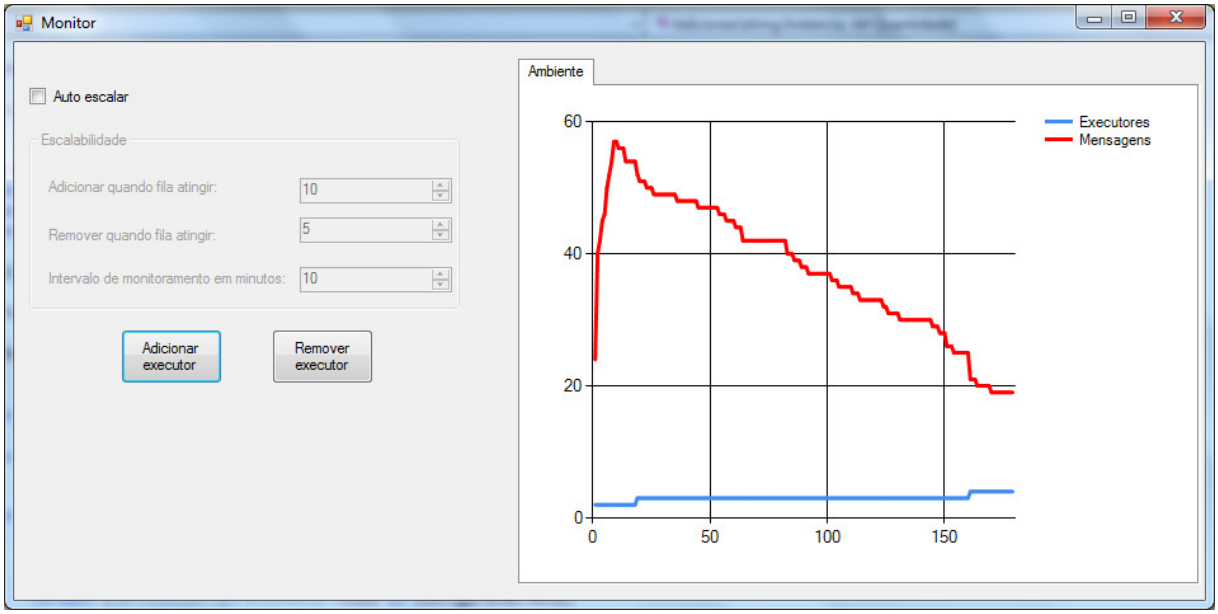


Figura 20 – Monitor de Testes adicionando `Executores de Testes`

Aplicando a auto escalabilidade, o monitor de testes irá utilizar as regras que estão no grupo escalabilidade da tela do monitor de testes para adicionar ou remover um novo `Executor de Testes`. Desta maneira, o protótipo será escalado da seguinte maneira:

a) Adicionar quando fila atingir: significa que quando a fila de mensagens atingir o valor configurado, ela irá adicionar uma nova instância de máquina virtual do tipo `Executor de testes`;

b) Remover quando fila atingir: significa que quando a fila de mensagens atingir o valor configurado, ela irá remover uma instância de máquina virtual do tipo `Executor de testes`;

c) Intervalo de monitoramento em minutos: representa de quanto em quanto tempo deverá ser avaliado a necessidade de escalabilidade. Em testes efetuados no Windows Azure, verificou-se que o Windows leva entre 3 e 5 minutos para iniciar uma nova instância de máquina virtual. Diante disto, verifica se que o intervalo de monitoramento deve ter um tempo mínimo de aproximadamente 10 minutos.

A figura 21 exibe a tela do `Monitor de Testes` utilizando a escalabilidade automática.

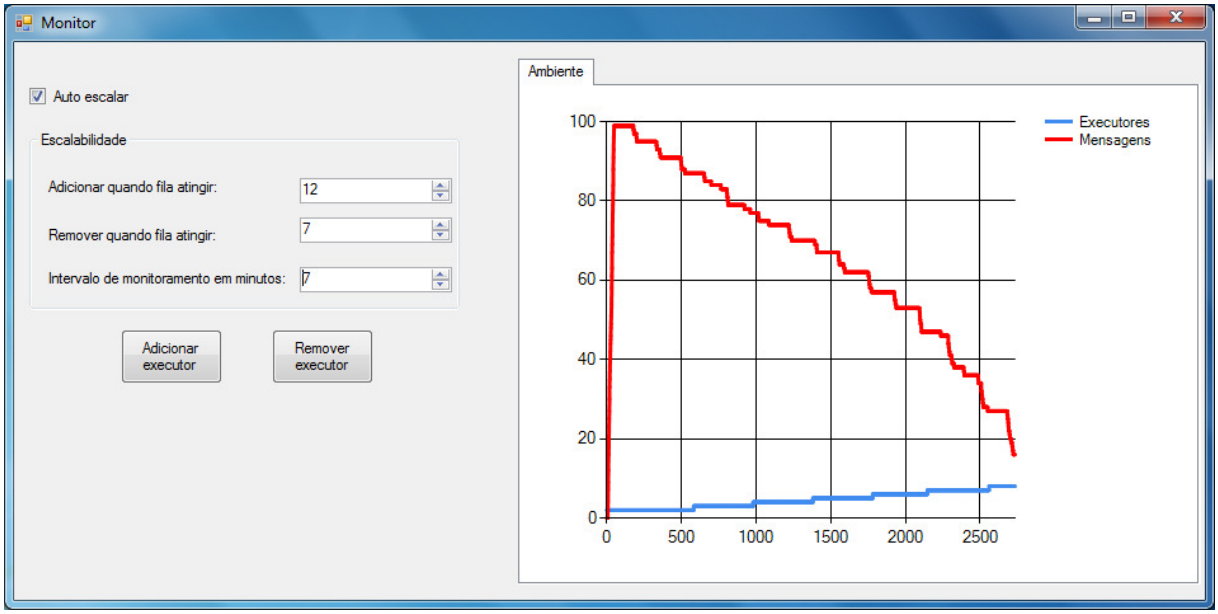


Figura 21 – `Monitor de Testes` aplicando auto escalabilidade

3.5 RESULTADOS E DISCUSSÃO

Para demonstrar o uso da escalabilidade do Windows Azure, foram realizados dois testes, onde em um deles foi realizada a computação destas requisições com um número mais elevado de instâncias de máquinas virtuais, e em outro foram utilizadas com menos máquinas virtuais.

O Windows Azure disponibiliza diferentes tipos de máquinas virtuais, tendo elas diferença de preço e de configuração. Em todos os testes realizados, foram utilizados máquinas virtuais do tipo *extra-small*, onde seu valor por hora é de quatro centavos de dólar.

A figura 22 exibe o `Cliente de Testes` ao final da execução do primeiro teste, onde foi efetuado o processamento de 100 requisições em aproximadamente 700 segundos, utilizando quinze máquinas virtuais para processar todas as requisições.

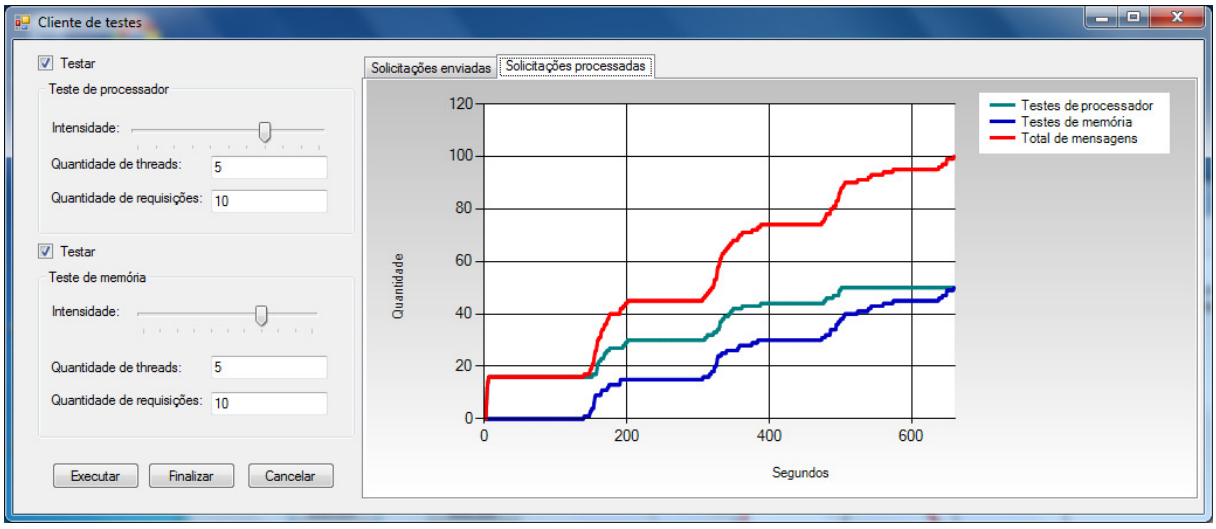


Figura 22 – `Cliente de Testes` efetuando o processamento de 100 requisições

A figura 23 exibe o `Monitor de Testes` ao final do teste.

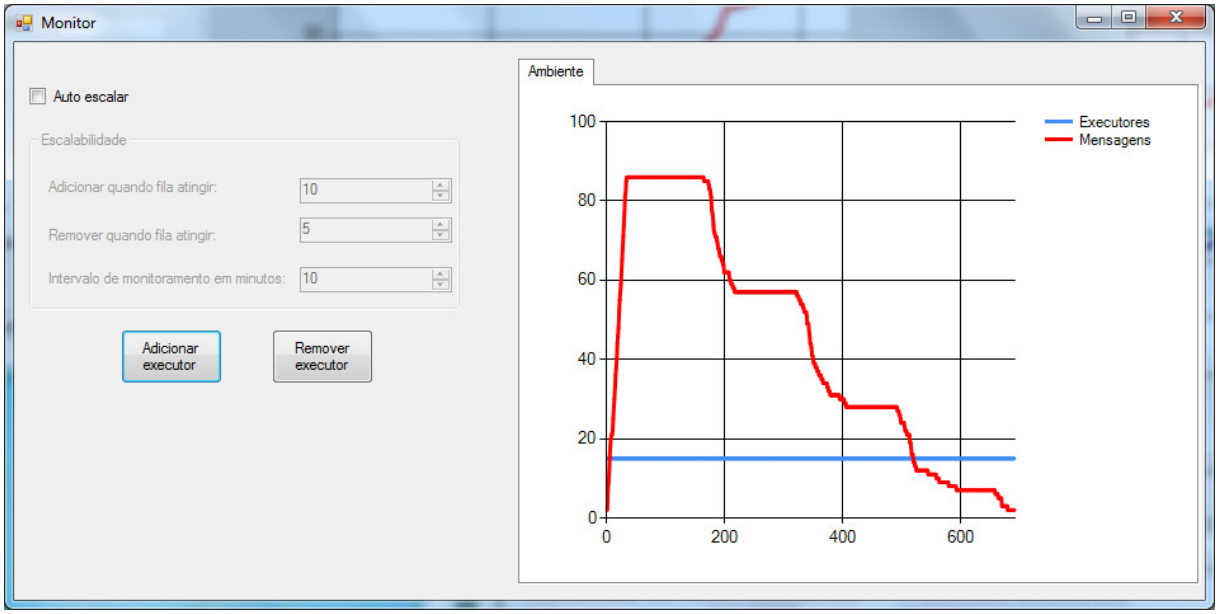


Figura 23 – `Monitor de Testes` monitorando os testes com 15 `Executores de Testes`

O segundo teste consistiu em efetuar o processamento das mesmas 100 requisições utilizando menos máquinas virtuais. No decorrer do teste, foram adicionadas novas instâncias, variando entre três e cinco instâncias durante todo o teste.

Para realizar o processamento destas requisições, o protótipo levou certa de 2.000

segundos. A figura 24 exibe o `Cliente de Testes` após a finalização do teste.

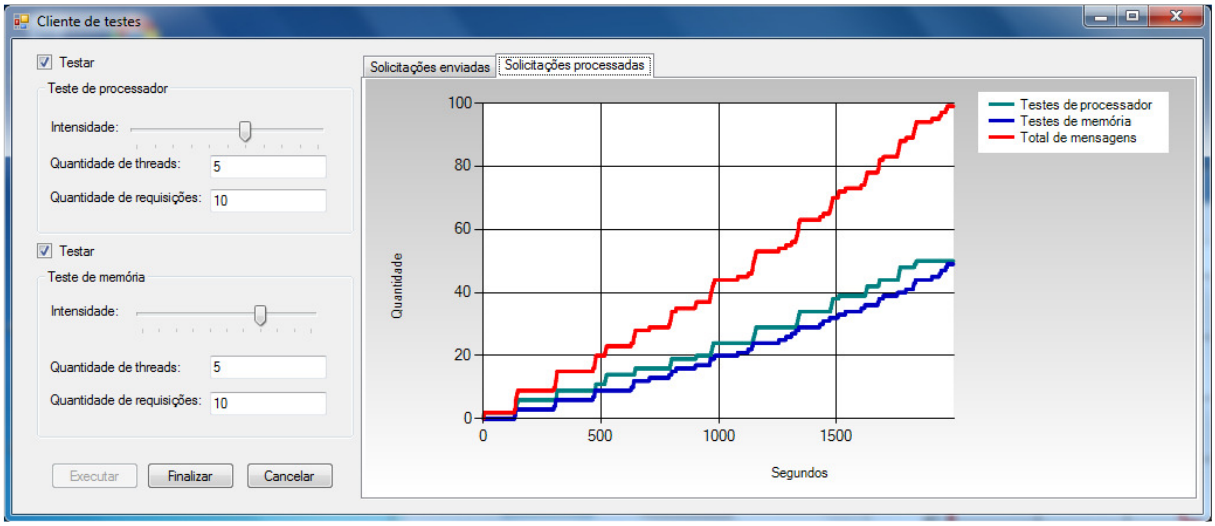


Figura 24 - `Cliente de Testes` efetuando o processamento de 100 requisições

A figura 25 exibe o `Monitor de Testes` ao final da execução do teste.

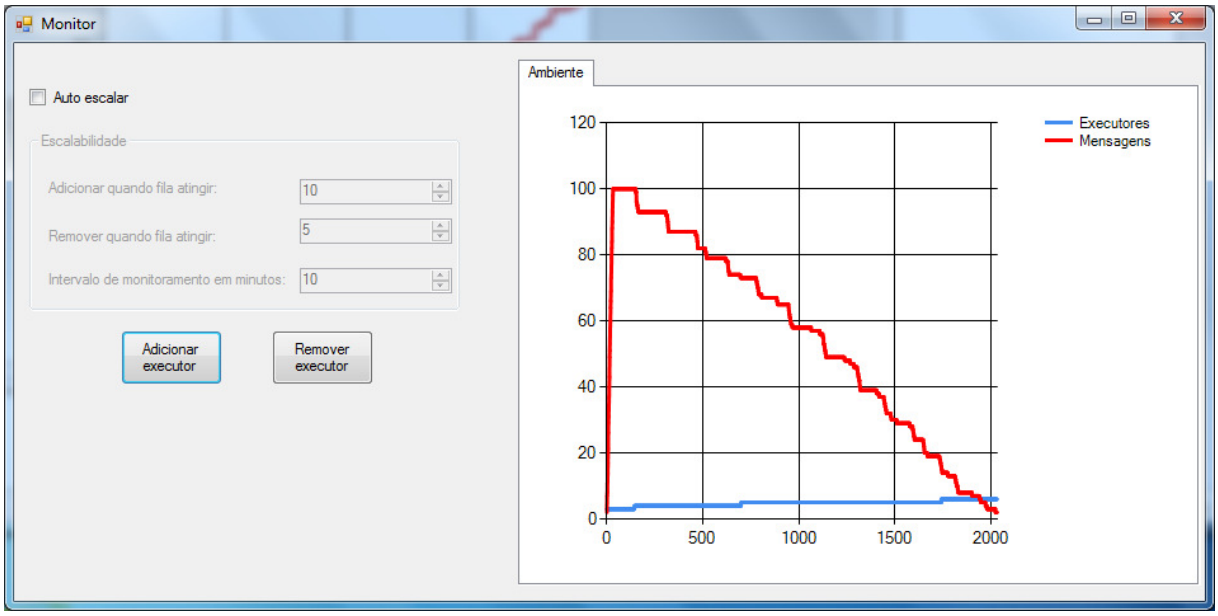


Figura 25 – `Monitor de Testes` monitorando os testes com três `Executores de Testes`

Verificando estes dois cenários, é visível o ganho de desempenho quando se utiliza um maior número de máquinas virtuais para efetuar o processamento de várias requisições, sendo elas independentes uma da outra.

A grande vantagem disponibilizada pela plataforma do Windows Azure é a facilidade de escalar a estrutura a fim de adaptar as necessidades momentâneas. Por exemplo, durante a execução dos testes, foi possível a adição de novas instâncias de máquinas virtuais para processamento, tal como foi possível a remoção destas quando não era mais necessária sua computação.

4 CONCLUSÕES

O desenvolvimento do trabalho atendeu a todos os objetivos propostos. O primeiro objetivo da pesquisa foi desenvolver um serviço utilizando WCF e disponibilizá-lo em forma de *web service* no Windows Azure.

O segundo objetivo atendido é a computação na nuvem através de algoritmos matemáticos e algoritmos de manipulação e caminhamento em grafo. Estes dois algoritmos são executados pelos `Executores de Testes` causando o consumo de processador e memória das maquinas virtuais que os executam.

Após atender os dois primeiros objetivos, foram efetuados testes de escalabilidade na plataforma Windows Azure. Escalando o protótipo periodicamente, foi obtido um desempenho superior diante da utilização dele de maneira estática, ou seja, sem a escalabilidade de recursos.

Comparando o protótipo ao *framework* WASABi, pode se verificar que há mais variáveis que podem ser analisadas para adotar fatores de medição para escalar as aplicações. Confrontando o protótipo ao WASABi, pode - se notar que existe uma gama de variáveis que o protótipo não explora. Ao contrário do *framework* citado, o protótipo não analisa a situação dos *roles*, tais como nível de processamento e nível de memória. O protótipo é restrito a fatores externos, como o tamanho da fila, para aplicar escalabilidade. Diferente do protótipo desenvolvido, o WASABi limita-se a ser um *framework*, enquanto o protótipo, além de aplicar regras de escalabilidade, permite avaliar o desempenho através dos protótipos de `Monitor de Testes` e de `Cliente de Testes` criados.

Concluindo os testes de escalabilidade, são visíveis outros fatores a serem abordados em novos testes na plataforma Windows Azure. O protótipo desenvolvido limitou-se a utilizar recursos de escalabilidade de instâncias, não abordando alta alocação de novas instâncias. Outro fator importante, é o teste relacionada a tolerância a falhas. O Windows Azure permite utilizar o método `OnStop()` em caso de um desligamento programado da máquina virtual, porém em casos em que há um desligamento não programado deve se estudar alternativas de tratamento para evitar perda de dados. Diante destas considerações, torna-se claro que ao desenvolver para a plataforma Windows Azure, são necessários estudos sobre a arquitetura a ser utilizada, buscando uma aplicação que possibilita usufruir dos recursos disponibilizados afim de extrair as vantagens da computação na nuvem.

4.1 EXTENSÕES

Ao longo do desenvolvimento do protótipo, foram pesquisadas alternativas para auxiliar o monitoramento da aplicação, porém bibliotecas para o auxílio de diagnósticos em aplicações na plataforma do Windows Azure ainda estão em desenvolvimento. Diante disto torna-se interessante desenvolver novos frameworks para o auxílio de monitoramento e também o auxílio à alteração de configurações dos *roles* enquanto eles estão em plena execução, provendo assim uma elasticidade para o serviço de maneira ágil e rápida.

A plataforma do Windows Azure fornece serviços para auxílio ao desenvolvimento de aplicações que utilizam *cache* de memória. Seria interessante o desenvolvimento de um protótipo que explore serviços onde se faz necessária a utilização de *cache*, tanto para desempenho do sistema, quanto para a construção de aplicações que utilizam sessões com estado, ou seja, sessões *state-full*.

Durante o levantamento das regras de negócio a serem utilizadas, foi levantada a hipótese de se utilizar a nuvem para o processamento de algoritmos em paralelo, onde várias máquinas utilizariam seu poder de processamento para um determinado fim. Devido a não utilização desta técnica, fica como sugestão para uma pesquisa futura, onde poderiam ser usados a escalabilidade do Windows Azure para efetuar o processamento de um algoritmo paralelo, para geração de criptografia ou até mesmo, algoritmos paralelos para problemas computacionais muito comuns, como o problema do caixeiro viajante.

REFERÊNCIAS BIBLIOGRÁFICAS

HIRSHMAN, Shaun et al. Arquiteturas de alta disponibilidade e hospedagem em massa. **The Architecture Journal**, Washington, n. 11, p. 2-6, 2007. Disponível em: <http://download.microsoft.com/download/5/a/5/5a58d817-90d1-4878-b275-26ab3552e6d3/JOURNAL/Journal11.pdf>. Acesso em: 04 abr. 2011.

KOMMALAPATI, Hanu. Windows Azure platform for enterprises. **MSDN Magazine,** [S.l.], v. 25, n. 2, 2010. Disponível em: <http://msdn.microsoft.com/en-us/magazine/ee309870.aspx>. Acesso em: 16 maio 2011.

KRISHNAMURTHY, Ganesan; THOTHADRI, Sripriya. **Integration and migration of COM+ services to WCF.** [S.l.], 2007. Disponível em: <http://msdn.microsoft.com/en-us/library/bb978523.aspx>. Acesso em: 05 abr. 2011.

LI, Henry. **Introducing Windows Azure:** learn Windows Azure to create the next-generation, cloud-based applications. New York: Apress, 2009.

LIN, Feng-Tse; SHIH, Teng-San. Cloud computing: the emerging computing technology. **International Journal of Innovative Computing, Information and Control (IJICIC),** Tawian, n. 1, p. 33-38, 2010. Disponível em: <http://www.ijicic.org/elb10-0515.pdf>. Acesso em: 05 abr. 2011.

LÖWY, Juval; **Programming WCF services.** Sebastopol: O'Reilly, 2010.

MENKEN, Ivanka; BLOKDIJK, Gerard. **Virtualization**: the complete cornerstone guide to virtualization best practices: concepts, terms, and techniques for successfully planning, implementing and managing enterprise it virtualization technology. Australia: Emereo Pty, 2008.

MICROSOFT. Autoscaling application block (WASABi). [S.l.], 2011. Disponível em: <http://entlib.codeplex.com/releases/view/74618#DownloadId=292691>. Acesso em: 21 nov. 2011.

MSDN. **What is Windows Communication Foundation**. [S.l.], 2011. Disponível em: <http://msdn.microsoft.com/pt-br/library/ms731082.aspx> . Acesso em: 05 abr. 2011.

REDKAR, Tejaswi. **Windows Azure platform:** unlock the power of cloud computing. New York: Apress, 2009.

SINGH, Munindar P.; HUHNS, Michael N.; **Service-oriented computing –** semantics, process, agents. West Sussex: John Wiley & Sons Ltd, 2005.

SKONNARD, Aaron; BROWN, Keith. **Introdução ao Windows Azure platform AppFabric para desenvolvedores:** estendendo as aplicações .NET à plataforma Windows Azure. [S.l.]: PluralSight, 2009. Não paginada. Disponível em: <http://download.microsoft.com/download/E/D/C/EDCDB1AF-A0B2-45E8-BE84-553C90688EEC/An_Introduction_to_Windows_Azure_platform_AppFabric_for_Developers-BRZ.doc>. Acesso em: 04 abr. 2011

SOSINSKY, Barrie. **Cloud computing bible**. New Jersey: John Wiley and Sons, 2011.

SOUSA, Flávio R. C.; MOREIRA, Leonardo O.; MACHADO, Javam C. **Computação em nuvem**: conceitos, tecnologias, aplicações e desafios. Versão revisada e estendida. Maranhão, 2010. Disponível em: <http://www.ufpi.br/ercemapi/index/pagina/id/3442>. Acesso em: 06 abr. 2011.

VECCHILA, Christian; CHU, Xingchen; BUYYA, Rajkumar. Aneka: a software platform for .NET-based cloud computing. In: GENTZSH, Wolfgang; GRANDINETTI, Lucio; JOUBERT, Gerhard (Ed.). **High speed and large scale scientific computing.** Amsterdam: IOS Press, 2009. p. 267-295. Disponível em: <http://www.buyya.com/gridbus/reports/AnekaCloudPlatform2009.pdf>. Acesso em: 06 abr. 2011.

TAURION, Cezar. **Cloud computing – computação em nuvem:** transformando o mundo da tecnologia da informação. Rio de Janeiro: Brasport, 2009.

ANEXO A – Configuração do Azure Storage

Para utilizar o ambiente na nuvem, deve se ter uma conta no portal da Microsoft e epara tal é necessário possuir uma conta no domínio Hotmail. Para poder utilizar o Azure Storage, deverá também ser criada uma conta e deverá ser vinculada a *subscription* criada anteriormente para o Windows Azure.

O `Servidor de Requisições` e os `Executores de Testes`, devem ser inicialmente configurados para que possam ser enviados ao Windows Azure com a estrutura desejada. As principais configurações necessárias irão abranger a quantidade de instâncias para cada protótipo, o tipo de máquina virtual e o endereço da s*torage account*, a conta criada para utilizar o Azure Storage.

A figura 26 ilustra a configuração da *storage account* no Visual Studio 2010.

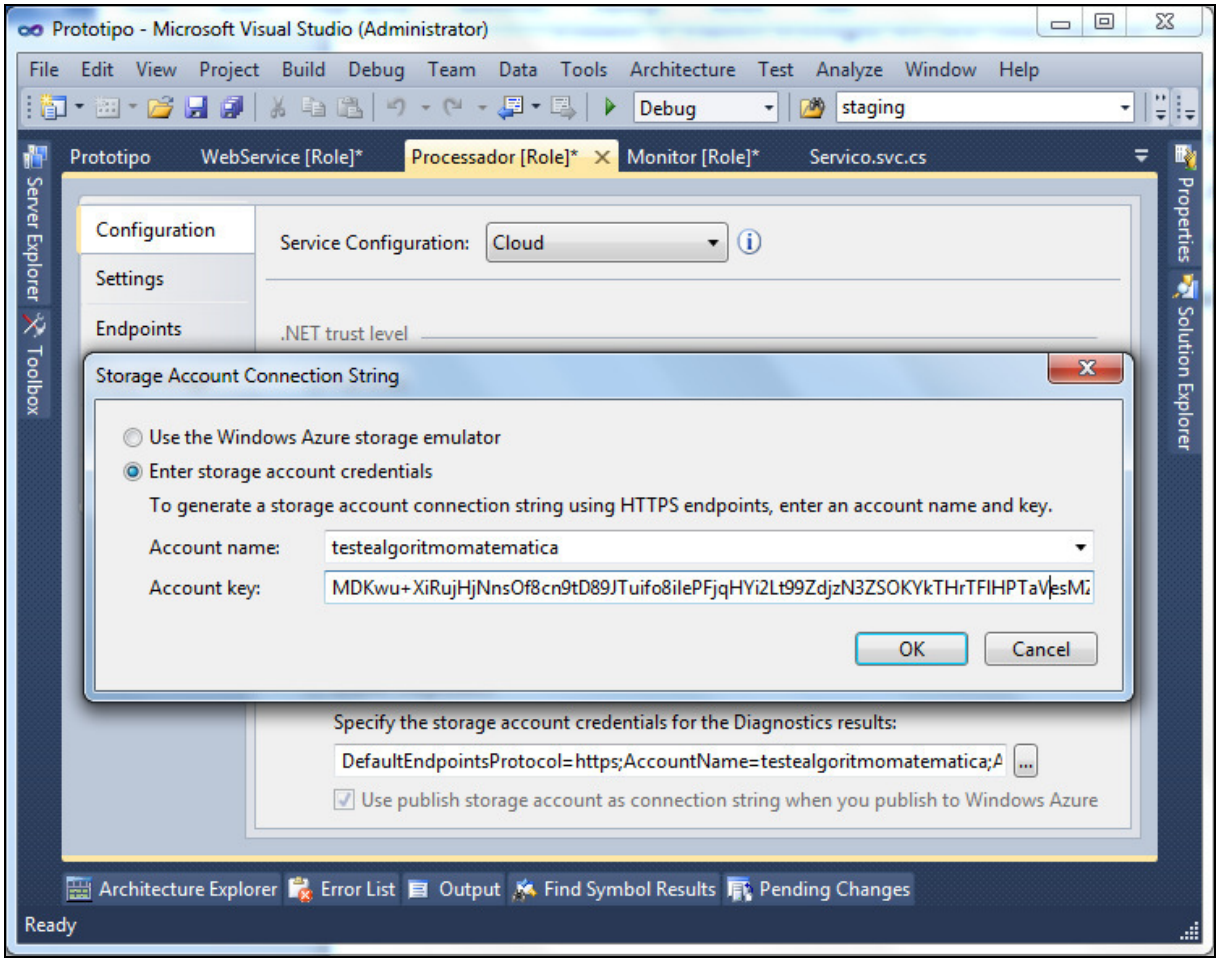


Figura 26 – Configuração do Azure Storage

ANEXO B – Processo de *deploy* no Windows Azure

Uma das maiores facilidades que foram vistas no processo de desenvolvimento do protótipo, e a execução do *deploy* no Windows Azure. Há diversas maneiras para executá-lo, porém a utilizada no desenvolvimento do trabalho, foi através de criação de um *pack* para ser importado no portal do Windows Azure Azure. A figura 27 detalha o *menu* onde é possível criar o *pack* para criar os arquivos para o *deploy*.

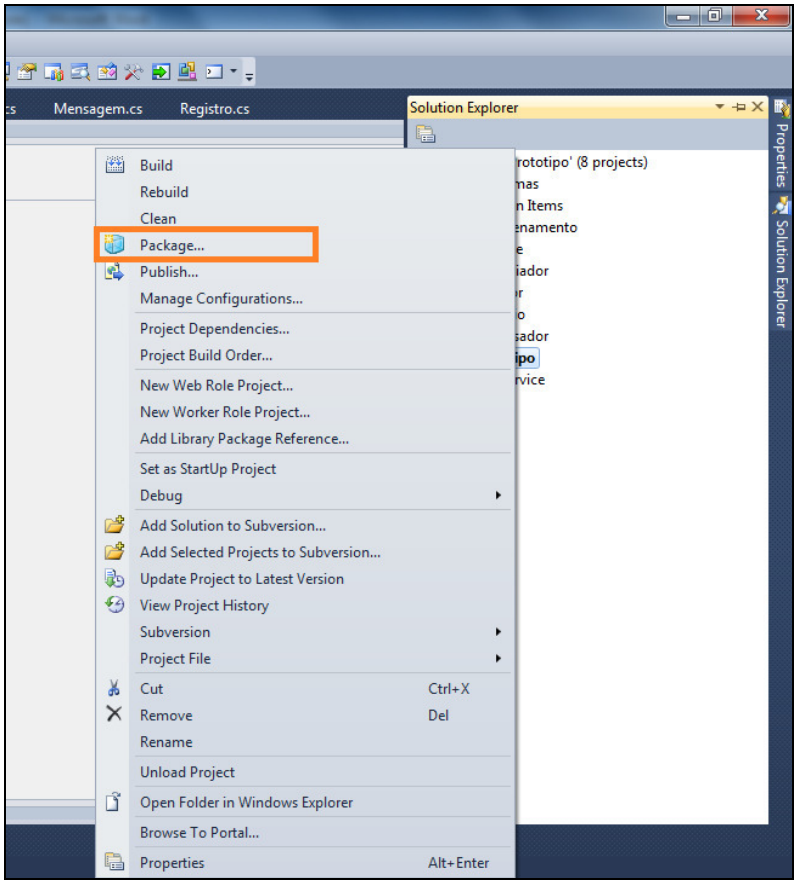


Figura 27 – Criação do *pack* para *deploy*

Após executado o comando *Package*, o Visual Studio criará dois arquivos, o `Prototipo.cspkg` e o `ServiceConfiguration.Cloud.cscfg`. Para carregar a aplicação no Windows Azure, basta acessar o portal, e adicionar um *Hosted Service*, então será exibirá a tela para configurações do *deploy* e solicitará os arquivos gerados pelo Visual Studio conforme detalhamento na figura 28.

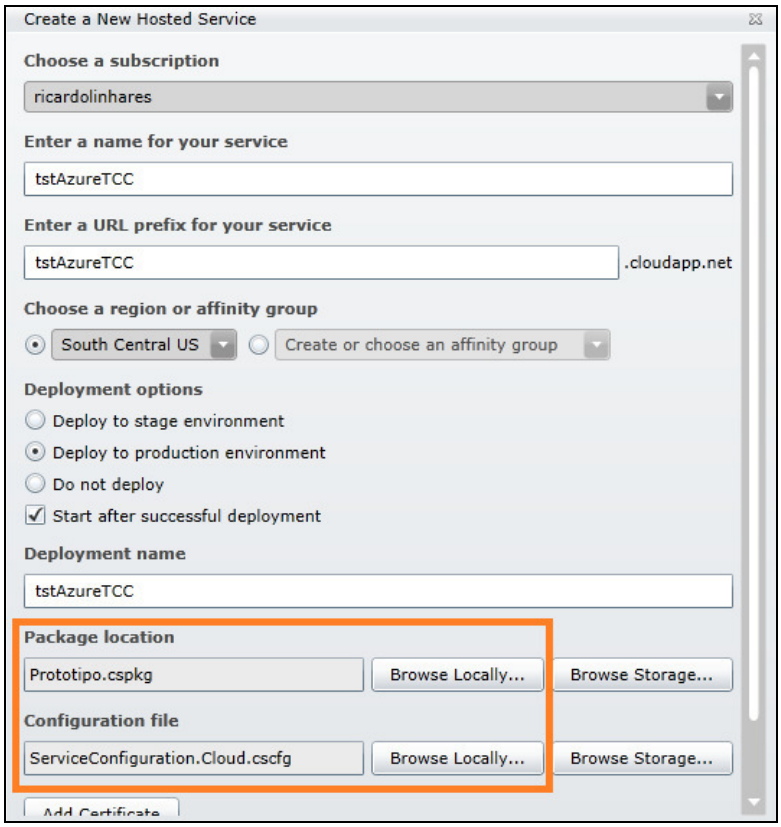


Figura 28 – Adição dos arquivos de definição do protótipo no Windows Azure

Depois de feito o *upload* dos arquivos, o Windows Azure iniciará o processo de publicação e as instâncias de máquinas virtuais estarão disponíveis para utilização. A figura 29 ilustra a publicação ocorrida com sucesso do protótipo.

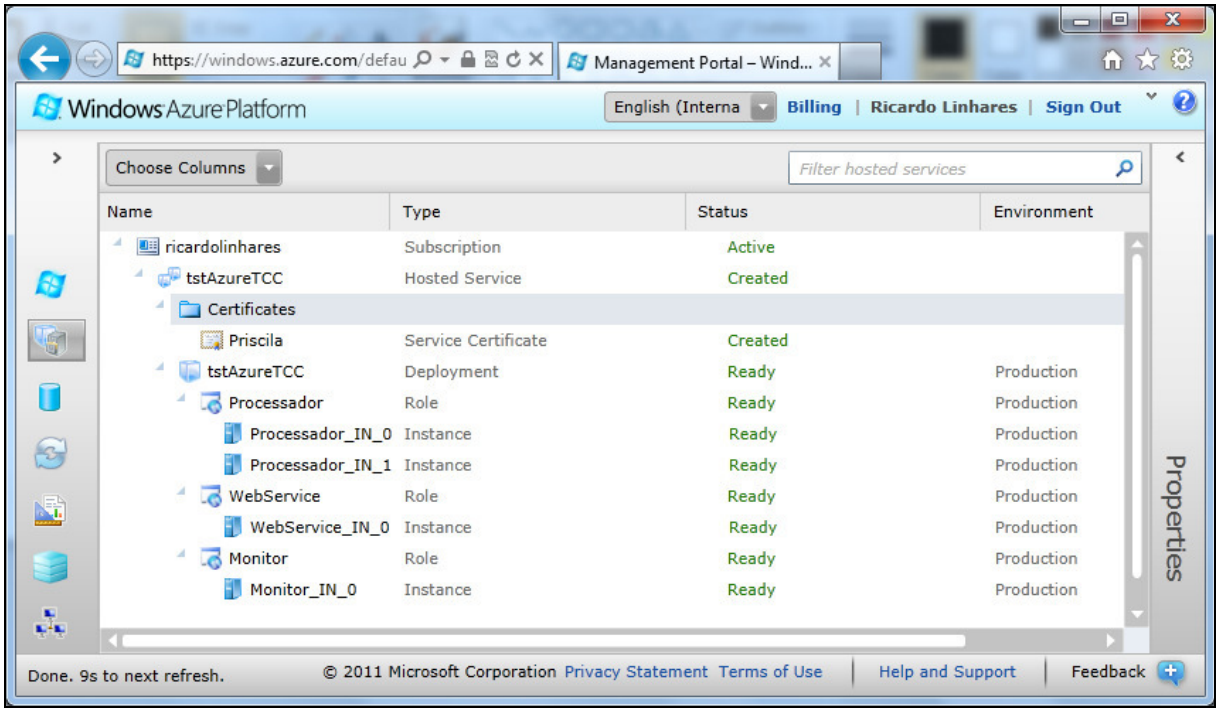


Figura 29 – Máquinas virtuais executando no portal do Windows Azure

ANEXO C – Certificados digitais para requisições REST

O certificado digital, em formato .CER deve ser gerado no computador na qual irá executar o `Monitor de Testes` para escalar o protótipo, e posteriormente é inserido através do portal do Windows Azure, na *subscription* responsável por executar o protótipo.

A figura 30 ilustra os certificados digitais inseridos no portal para execução do protótipo.

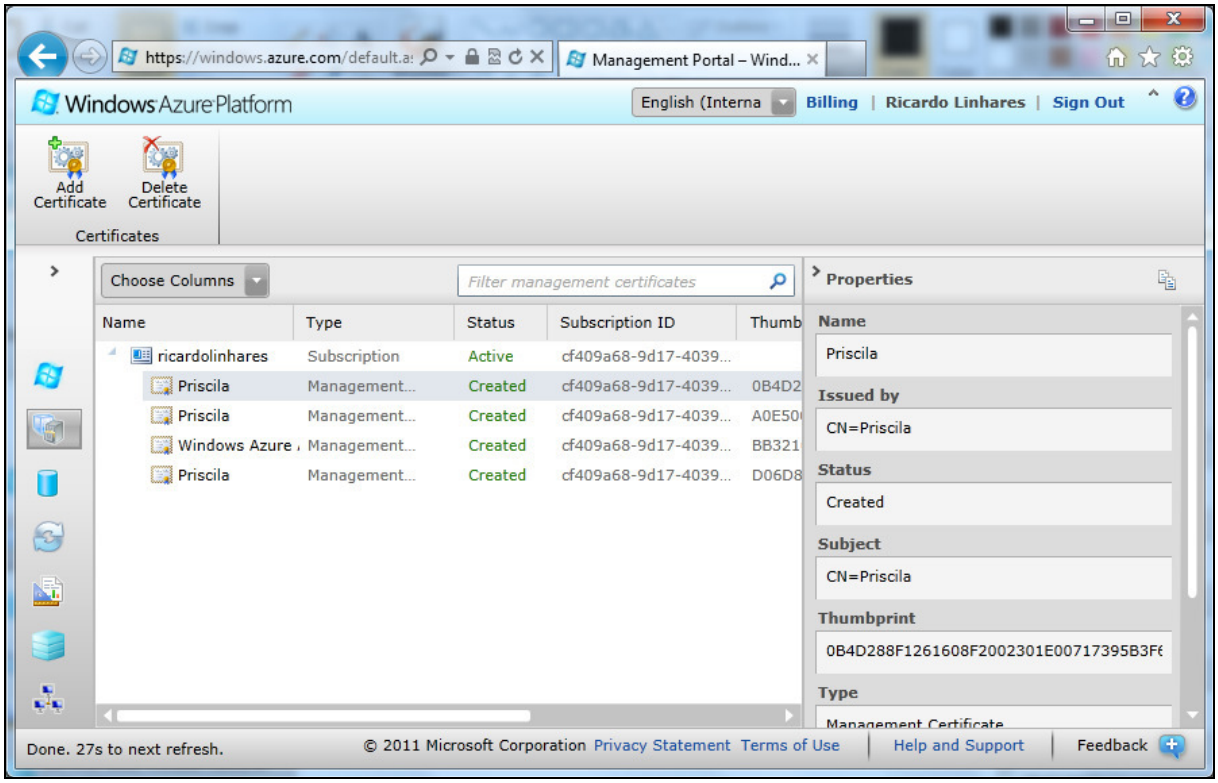


Figura 30 – Certificados digitais para gerenciamento do protótipo

ANEXO D – Cobrança por uso no Windows Azure

A cobrança do uso no Windows Azure, é feita através de horas de uso para as instâncias de máquinas virtuais, quantidade de tráfego e quantidade de dados armazenados no Azure Storage.

Ao ligar ativar uma instância no Windows Azure, passará a ser cobrado seu valor por hora. Ao remover a instância do Windows Azure, alterando o arquivo de configuração, não passa mais a ser cobrado o processamento desta instância. A figura 31 exibe um resumo dos custos a serem cobrados no cartão de crédito devido ao uso da plataforma.

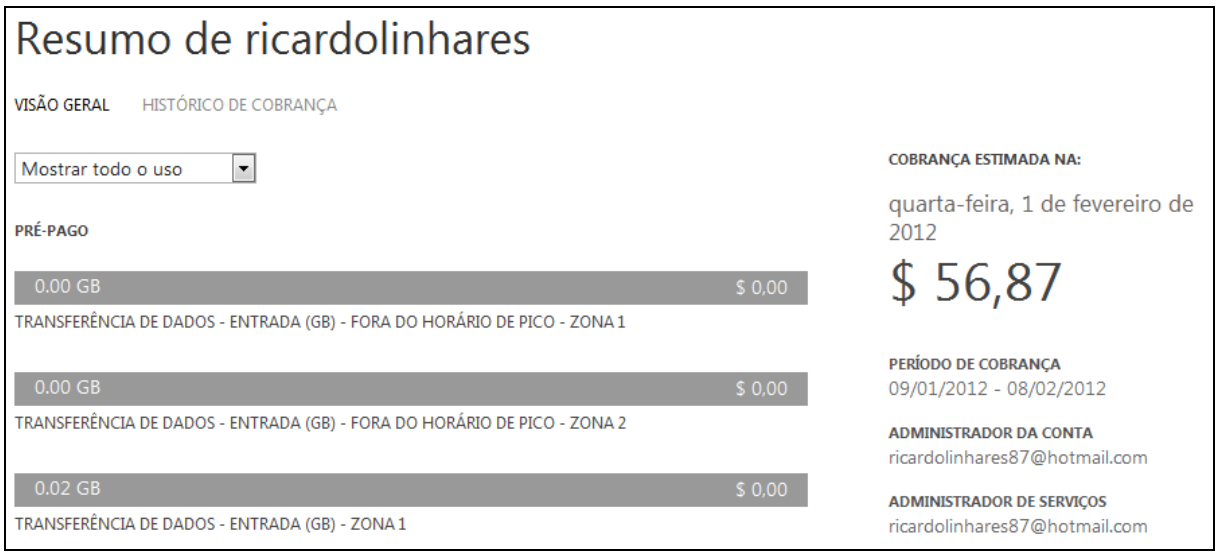


Figura 31 – Resumo de custos dos serviços do Windows Azure

A figura 32 exibe o detalhamento das horas de computação utilizada nos testes do protótipo.

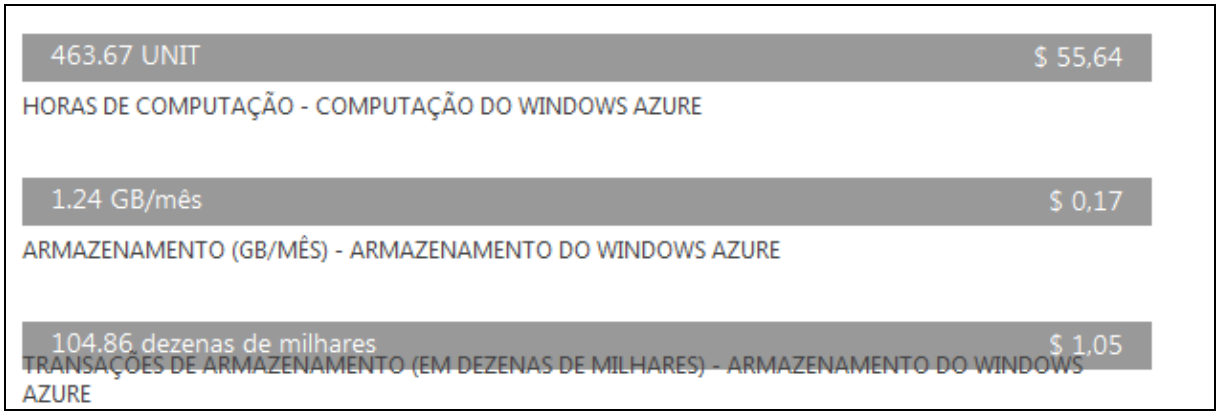


Figura 32 – Detalhamento das horas de computação a serem cobradas