

**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO**

BRUNO TEIXEIRA JEVEAUX

**PROTÓTIPO DE SISTEMA SCADA ELIPSE E3 COM
TRANSAÇÃO DE DADOS EM NUVEM DE SERVIÇOS
MICROSOFT AZURE**

VITÓRIA
2019

BRUNO TEIXEIRA JEVEAUX

**PROTÓTIPO DE SISTEMA SCADA ELIPSE E3 COM TRANSAÇÃO
DE DADOS EM NUVEM DE SERVIÇOS MICROSOFT AZURE**

Parte manuscrita do Projeto de Graduação do aluno **Bruno Teixeira Jevaux**, apresentada ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Klaus Fabian Côco

VITÓRIA
2019

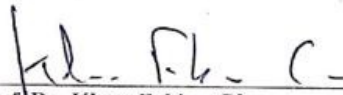
BRUNO TEIXEIRA JEVEAUX

**PROTÓTIPO DE SISTEMA SCADA ELIPSE® E3 COM TRANSAÇÃO
DE DADOS EM NUVEM DE SERVIÇOS MICROSOFT® AZURE**

Parte manuscrita do Projeto de Graduação do aluno **Bruno Teixeira Jevaux**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovada em 06, de dezembro de 2019.

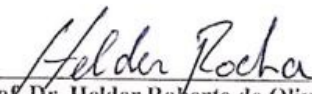
COMISSÃO EXAMINADORA:



Prof. Dr. Klaus Fabian Côco
Universidade Federal do Espírito Santo
Orientador



Prof. MSc. Lucas de Assis Soares
Instituto Federal do Espírito Santo
Examinador



Prof. Dr. Helder Roberto de Oliveira Rocha
Universidade Federal do Espírito Santo
Examinador

Aos meus pais, que sempre me apoiaram em todas as decisões que fiz em minha vida.

Agradeço primeiramente à Deus, que me deu força e sabedoria para passar por cada etapa e superar todos os desafios que surgiram ao longo do curso e da minha vida. Agradeço aos meus pais, que sempre proporcionaram todas as condições para que eu pudesse realizar meus sonhos e ter uma boa vida. Aos meus amigos e colegas de faculdade, que sempre estiveram presentes, dando suporte e sendo companheiros incríveis durante essa importante etapa em minha vida. Agradeço aos professores que fizeram ao máximo para contribuir para o crescimento dos alunos, lutando sempre pelo melhor. Agradeço, por fim, ao meu orientador Klaus Fabian Côco pela confiança, pelo apoio, pelo tempo e pela atenção.

RESUMO

Com os avanços tecnológicos no meio industrial e com o desenvolvimento da Indústria 4.0, surge uma crescente necessidade de atualizar e aprimorar a estrutura da cadeia de automação industrial. Uma forma de aderir a essa evolução é por meio de melhorias na comunicação e na interação entre processos e seus dados, com foco na nova importância dada ao armazenamento e utilização de informações. O presente trabalho teve como objetivo principal elaborar uma proposta de desenvolvimento de um protótipo de comunicação para realizar a integração entre um sistema supervisório e um banco de dados em nuvem. O alvo do trabalho foi avaliar a comunicação entre os dois modelos de sistemas em um ambiente industrial simulado. Durante a elaboração do protótipo foi utilizada a plataforma Elipse E3, para o desenvolvimento do sistema supervisório, e o Banco de Dados SQL *Server* na plataforma em nuvem Microsoft Azure, para armazenamento dos dados. O protótipo foi, ao final, após realizados todos os ajustes necessários, testado e validado por meio de medições, testes de consistência e confiabilidade.

LISTA DE FIGURAS

Figura 1 – Pirâmide de automação industrial.....	14
Figura 2 – Componentes de um sistema SCADA.....	21
Figura 3 – Setores de aplicação de sistemas SCADA.....	24
Figura 4 – Primeira geração: sistemas SCADA monolíticos.....	25
Figura 5 – Segunda geração: sistemas SCADA distribuídos.....	25
Figura 6 – Terceira geração: sistemas SCADA em rede	26
Figura 7 – Quarta geração: sistemas SCADA baseados em IoT e nuvem.....	26
Figura 8 – As três camadas da computação em nuvem: SaaS, PaaS e IaaS	29
Figura 9 – Níveis de escopo do Azure	38
Figura 10 – Painel de gerenciamento do grupo de recursos Foregaster.....	41
Figura 11 – Painel de gerenciamento do servidor de banco de dados foregaster	43
Figura 12 – <i>Login</i> de administrador criado no banco de dados mestre, no ambiente SSMS	46
Figura 13 – Painel de gerenciamento do banco de dados ValeDB	51
Figura 14 – Camadas de segurança da plataforma Azure para proteção dos dados	52
Figura 15 – Fluxograma de funcionamento do <i>firewall</i>	53
Figura 16 – Painel de gerenciamento das regras de <i>firewall</i> a nível de servidor.....	56
Figura 17 – Controle de acesso às linhas de uma tabela por meio de RLS	58
Figura 18 – Proteção do banco de dados por meio da ferramenta Proteção Avançada de Dados	59
Figura 19 – Funcionamento do recurso de mascaramento dinâmico de dados.....	64
Figura 20 – Ambiente <i>SQL Server Management Studio</i>	65
Figura 21 – Aba Propriedades da Conexão do menu Options do SSMS.....	66
Figura 22 – Ambiente SSMS conectado com o banco de dados ValeDB	67
Figura 23 – Formas de monitoramento na arquitetura E3	71
Figura 24 – Tela inicial do E3 <i>Studio</i> , após criação do domínio “PrototipoSCADA”	72
Figura 25 – Objeto <i>Viewer</i> criado no E3 <i>Studio</i>	73
Figura 26 – Tela sem objetos, após criação	75
Figura 27 – Configuração provisória da tela inicial do protótipo	76
Figura 28 – <i>Tag demo</i> e atributos no E3 <i>Studio</i>	78
Figura 29 – Configurações da pena PenaVazOleo criada.....	79
Figura 30 – Protótipo em funcionamento inicial no E3 <i>Viewer</i>	80
Figura 31 – Conexão do E3 <i>Studio</i> com o banco de dados SQL do Azure	82

Figura 32 – Interfaces de histórico no E3 <i>Studio</i>	85
Figura 33 – Visualização no SSMS das estruturas das tabelas criadas pelo E3 no banco de dados	85
Figura 34 – Teste do protótipo por meio do E3 <i>Viewer</i>	87
Figura 35 – Registros de período, forma de onda e taxa de amostragem no banco de dados, visto pelo SSMS.....	88
Figura 36 – Registros de módulo de vazão no banco de dados, visto pelo SSMS	88
Figura 37 – Exemplo de execução de um <i>WebViewer</i>	90
Figura 38 – Aplicação do protótipo monitorada por meio do <i>WebViewer</i> em uma máquina distinta do servidor.....	91
Figura 39 – Telas de monitoramento visualizadas pelo <i>Viewer</i> do protótipo do E3 melhorado	92
Figura 40 – Telas de alarmes e de E3Browser visualizadas pelo <i>Viewer</i> do protótipo do E3 melhorado.....	92

LISTA DE QUADROS

Quadro 1 – Características de cada camada de serviço	50
Quadro 2 – Faixa de valores da qualidade e significados	86
Quadro 3 – Pontos de operação do novo sistema criado.....	93

LISTA DE ABREVIATURAS E SIGLAS

IaaS	<i>Infrastructure as a Service</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
IHM	Interface Homem Máquina
IOPS	<i>Input/Output Operations Per Second</i>
IoT	<i>Internet of Things</i>
JSON	<i>JavaScript Object Notation</i>
MTU	<i>Master Terminal Unity</i>
NIST	<i>National Institute of Standards and Technology</i>
OLTP	<i>Online Transaction Processing</i>
PaaS	<i>Platform as a Service</i>
PL	<i>Procedural Language</i>
RTU	<i>Remote Terminal Unity</i>
SaaS	<i>Software as a Service</i>
SCADA	<i>Supervisory Control and Data Acquisition</i>
SGBD	<i>Software de Gerenciamento de Banco de Dados</i>
SQL	<i>Structured Query Language</i>
SSMS	<i>SQL Server Management Studio</i>
TI	Tecnologia da Informação
TLS	<i>Transport Layer Security</i>
UFES	Universidade Federal do Espírito Santo
XML	<i>Extensible Markup Language</i>

SUMÁRIO

1	INTRODUÇÃO.....	13
1.1	Apresentação	13
1.1.1	Indústria 4.0	14
1.1.2	Nova pirâmide de automação industrial.....	15
1.2	Motivação	17
1.3	Objetivos	18
1.3.1	Objetivo geral.....	18
1.3.2	Objetivos específicos.....	18
2	SISTEMAS DE SUPERVISÃO E AQUISIÇÃO DE DADOS.....	20
2.1	Funcionamento e Importância.....	20
2.2	Estrutura.....	20
2.3	Benefícios Proporcionados	22
2.4	Uso no Mundo Contemporâneo.....	23
3	COMPUTAÇÃO EM NUVEM E BANCO DE DADOS.....	28
3.1	Computação em Nuvem	28
3.2	Banco de Dados em Nuvem.....	31
3.2.1	Definição de banco de dados.....	31
3.2.2	Bancos de dados relacionais e não relacionais	31
3.2.2.1	Banco de dados relacional	32
3.2.2.2	Banco de dados não relacional	32
3.2.2.3	Diferenças entre banco de dados relacional e não relacional	32
3.2.3	Operação em nuvem e benefícios	33
3.2.4	Microsoft SQL Server	35
3.2.5	Banco de dados SQL do Azure	36
4	PLATAFORMA AZURE E CONFIGURAÇÕES.....	38
4.1	Grupos de Recursos	39
4.2	Servidores de Banco de Dados SQL do Azure	40
4.3	Criando um Servidor de Banco de Dados SQL	40
4.4	Usuários e Logins do SQL do Azure	43
4.4.1	Modelo tradicional baseado em login e usuário.....	44
4.4.2	Modelo de usuário em banco de dados independente	44

4.5 Criando um Banco de Dados SQL do Azure.....	46
4.5.1 Modelo de compra baseado em vCore	47
4.5.2 Modelo de compra baseado em DTU	47
4.5.3 Camadas de serviço e preço	48
4.6 Segurança e Configurações	51
4.6.1 Segurança de rede.....	52
4.6.1.1 Regras de <i>firewall</i> para IP a nível de servidor	53
4.6.1.2 Regras de <i>firewall</i> para IP a nível de banco de dados	54
4.6.2 Gerenciamento de acesso	56
4.6.3 Proteção contra ameaças.....	58
4.6.4 Proteção e criptografia da informação	60
4.6.4.1 Segurança da camada de transporte	60
4.6.4.2 <i>Transparent Data Encryption</i>	62
4.6.4.3 Demais proteções.....	63
4.7 <i>SQL Server Management Studio</i>	64
4.7.1 Conectando ao banco de dados.....	64
5 COMUNICAÇÃO ENTRE ELIPSE E3 E AZURE	68
5.1 Elipse E3	68
5.2 Arquitetura do Elipse E3	69
5.3 Configurações Iniciais do Elipse E3	71
5.3.1 Criação do domínio e do projeto	71
5.3.2 Criação do <i>Viewer</i>	72
5.3.3 Criação da tela inicial.....	74
5.3.4 Adicionando objetos à tela criada	75
5.4 Desenvolvimento e Configurações do Projeto Criado	77
5.5 Conectando a um Banco de Dados	80
5.5.1 Comunicação do E3 com o banco de dados SQL do Azure	81
5.5.2 Testando a conexão.....	82
5.5.3 Armazenando registros de variáveis.....	83
5.6 Testando o Protótipo com o Banco de Dados	86
5.7 <i>WebViewer</i>	89
5.7.1 Configurando o <i>WebViewer</i>	90
5.8 Funcionamento do Protótipo Final	91
6 CONCLUSÕES E TRABALHOS FUTUROS	96

REFERÊNCIAS BIBLIOGRÁFICAS.....98

1 INTRODUÇÃO

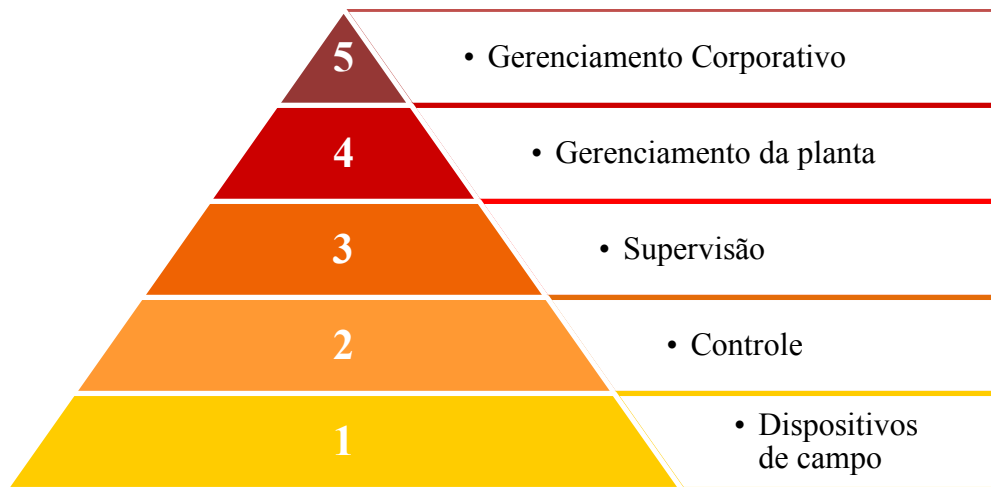
Neste capítulo é feita uma breve contextualização do tema do projeto, abordando em seguida as motivações que justificam sua produção, bem como os objetivos gerais e específicos a serem alcançados.

1.1 Apresentação

As Revoluções Industriais provocaram grandes melhorias nos sistemas de manufatura e de serviços. A primeira Revolução Industrial, por exemplo, é marcada pela invenção da máquina a vapor, que foi aplicada na indústria têxtil. Já a segunda contou com a utilização do aço, da energia elétrica e dos combustíveis derivados do petróleo. A terceira, também conhecida como revolução digital, é caracterizada pela eletrônica digital, pela computação, produção em massa, automação e outros. Por conta disso, entende-se que grandes mudanças aconteceram no processo de manufatura e na tecnologia da informação, onde surgiu uma crescente necessidade de coordenação e conexão de conceitos disruptivos como comunicação e *networking*. Com o desenvolvimento dessas áreas e sendo essas tecnologias coordenadas e comunicativas, constituiu-se o termo Indústria 4.0 (CEVIKCAN; USTUNDAG, 2018). Desse modo, uma área onde houve crescente procura e estudo foi a relacionada a melhorias na comunicação e na interação entre processos, seus dados e os estudos de análises desses. Nesse sentido, o presente trabalho buscou ir ao encontro dessa procura tomando como foco um estudo aplicado a plantas de automação industrial.

A constante modernização da estruturação da cadeia de automação industrial e mudanças na constituição de sua pirâmide organizacional vem ocorrendo por conta dos avanços tecnológicos já citados e foram motivação para o desenvolvimento deste trabalho. O interesse no estudo foi despertado por meio das mudanças e inovações que refletem em atualizações nos níveis 3 e 4 da Pirâmide de Automação Industrial (ilustrada pela Figura 1 e explicada em seção posterior), que englobam a supervisão, armazenamento e gerenciamento de dados na planta. O propósito, então, é, dentro do escopo desses dois níveis, estudar e aplicar novos conceitos que estão surgindo e sendo gradualmente implementados pelas empresas no país.

Figura 1 – Pirâmide de automação industrial



Fonte: Moraes e Castrucci (2012).
 Nota: Modificado pelo autor.

Busca-se, dessa forma, agregar os recursos disponíveis por meio de bancos de dados em nuvem com os de um sistema de supervisão e realizar uma comunicação entre os dois serviços objetivando possibilitar atuação e monitoramento de processos a distância, permitir disponibilização de dados de forma mais acessível, compartilhar dados de um processo de forma rápida e eficaz, operar em diferentes meios e setores, diminuir a quantidade de servidores locais, reduzir a infraestrutura de *hardware* das empresas e garantir a segurança e a redundância em armazenamento dos dados.

A implementação desses procedimentos de comunicação tem sua importância no processo, pois refletem, a partir da evolução na acessibilidade dos dados, em melhorias na produção. Por meio de seu desenvolvimento é possível entender os processos de produção, aumentar a qualidade de serviço dos produtos além de realizar otimização dos processos.

1.1.1 Indústria 4.0

O conceito Indústria 4.0 traz consigo uma expectativa de transformação, flexibilização e progresso do ambiente industrial no século XXI. Com ela espera-se um novo processo de fabricação por meio da descentralização com necessidades atreladas a um ágil e acelerado desenvolvimento de produtos, além de produção flexível e personalizada de acordo com a necessidade do consumidor. Nesse sentido, torna-se possível a criação de uma rede que contenha recursos, informação, objetos e pessoas de modo a compor a Internet das Coisas e

Serviços, de modo que os efeitos desse fenômeno afetem muito positivamente a indústria (KAGERMANN; WAHLSTER; HELBIG, 2013).

Diante das novas delimitações e direcionamentos, para o processo industrial se adaptar às novas necessidades de produção, tornou-se inevitável a realização de uma ampliação e um aprimoramento da comunicação, interação e integração entre processos. Desde então, essas novas delimitações e o desenvolvimento de novas tecnologias industriais levaram a uma reorganização e remodelação dos processos de automação industrial. Isso implica em uma necessidade de gerenciamento de processos industriais complexos onde diferentes tarefas são executadas por diferentes parceiros, em diferentes localizações geográficas (KAGERMANN; WAHLSTER; HELBIG, 2013).

O processo atual engloba grandes sistemas de gerenciamento e supervisão que atuam de forma remota, sistemas de controle automatizados, redes de dispositivos IoT (*Internet of Things*) que interagem com o ambiente, *softwares* e infraestruturas conectadas à rede, além de aplicativos e redes de dados armazenados em nuvem. Isso significa que produtos e serviços novos ou melhorados se tornam possíveis de serem criados, bem como custos podem ser reduzidos e a produtividade pode ser aumentada (PETRASCH; HENTSCHE, 2016).

Portanto, diante da remodelação industrial, torna-se importante conhecer e verificar a nova organização do processo e sua dinâmica atual englobadas na pirâmide de automação industrial e também usufruir das tecnologias para aplicações que tragam melhorias ao processo ou planta a ser tratada.

1.1.2 Nova pirâmide de automação industrial

A pirâmide de automação industrial representa os níveis da cadeia de controle de uma organização e mostra, em níveis hierárquicos, como os equipamentos que constituem essa cadeia estão estruturados e como estão relacionados. A hierarquia é organizada de forma que os níveis da base estão relacionados com maior fluxo de dados e baixo tratamento dos mesmos enquanto os níveis mais superiores têm capacidade de tratar e gerenciar esses dados e de tomar decisões. É possível observar a pirâmide de automação industrial por meio da Figura 1.

Segundo Moraes e Castrucci (2012), a pirâmide de automação industrial é constituída de cinco níveis principais:

- Nível 1: A base da pirâmide engloba os dispositivos de campo que estão diretamente associados a dinâmica da planta. Nesse nível se encontram dispositivos responsáveis pelo sensoriamento e atuação no processo.
- Nível 2: Esse nível é responsável por realizar o controle automático das atividades da planta. Eles estão diretamente relacionados com os sensores e atuadores no processo de forma local enviando e recebendo dados, e executando comandos de forma automática. Um exemplo de equipamento deste nível são os PLCs (Controladores Lógico Programáveis).
- Nível 3: O nível intermediário é responsável pela supervisão do processo. Nele é possível encontrar os sistemas supervisórios. Esses sistemas recebem e enviam comandos e dados para equipamentos localizados no Nível 2 de forma remota e estão diretamente associados a bancos de dados que armazenam informações das variáveis de produção da planta.
- Nível 4: Esse nível é responsável pelo gerenciamento da planta, ou seja, a partir das informações processadas no nível 3, buscam, com auxílio de *softwares* de gerenciamento, planejar a futura produção e regular a logística de suprimentos.
- Nível 5: O nível mais alto é responsável pelo gerenciamento corporativo e administração de recursos. Esse gerenciamento toma como base todas as informações geradas pelos níveis inferiores para tomada de decisão. Dentre os principais componentes desse nível estão *softwares* responsáveis pela gestão de vendas e financeira.

As constantes melhorias associadas a tecnologia afetam a pirâmide de automação em diferentes níveis. No Nível 2, os dispositivos de controle automático vêm adquirindo maior facilidade em sua programação, maior precisão em sua atuação e resposta às mudanças no sistema, além disso, vêm apresentando maior robustez e vitalidade em seus equipamentos, os tornando menos susceptíveis a falhas.

Nos níveis 3 e 4, os dispositivos de supervisão e gerenciamento passam a contar com grandes melhorias e inovações. Portanto, fala-se em maior disposição de variáveis para análise, melhor desempenho, maior integração entre os equipamentos e sistemas automáticos, maior centralização de monitoramento para atuação na planta e maior capacidade de armazenamento

nos bancos de dados. Também é importante ressaltar a maior acessibilidade por meio dos bancos de dados *online*.

As mudanças relacionadas à automação e aos dispositivos de sua cadeia vêm objetivando o aprimoramento no tratamento dos dados coletados da planta para a melhoria da precisão nas tomadas de decisão, na otimização de processos, na correlação direta entre variáveis e falhas do processo, em soluções integradas de plataformas e *softwares*, no fornecimento de ferramentas configuráveis e de soluções personalizáveis, e no desenvolvimento de dispositivos inteligentes mais simplificados (ZAMPRONHA, [20--?]).

De acordo com a alteração e modernização na Pirâmide de Automação, torna-se relevante estudar, inovar e aplicar conceitos que vão ao encontro do foco do desenvolvimento de forma a aprimorar o controle de processos em ambientes industriais.

1.2 Motivação

Atualmente, com a crescente inovação e busca de melhoria dos processos, que têm como objetivo tornar as indústrias competitivas no mercado em um processo cada vez mais ágil e personalizado ao consumidor, o ato de buscar entender os processos de produção, além de estudar e aplicar ferramentas e combinações que possam prover uma evolução nos quesitos solicitados é de grande expressividade. Dessa forma, é plausível dizer que uma dessas ferramentas é o protótipo de comunicação proposto, que permite atuar e monitorar o processo remotamente, o que exige uma melhor acessibilidade dos dados.

Segundo Gaushell e Darlington (1987), a aquisição de dados, o processamento desses dados para uso do operador, e o controle pelo operador de dispositivos remotos são atividades fundamentais nas quais todos os sistemas de controle modernos se baseiam. Nesse sentido, é esperado que existam estudos nessa área em questão. Por exemplo, Church e outros (2017) dizem que sistemas SCADA enviam e armazenam milhares de mensagens por segundo, de modo a gerar grandes quantidades de dados. Como esses sistemas são críticos para processos industriais, frequentemente são utilizados em *hardwares* dedicados de alta confiabilidade. Todavia, Church e outros (2017) defendem que isso é um contraste à realidade atual da computação, que deixa de utilizar aplicações hospedadas em servidores internos para utilizar

aplicações que funcionam em nuvem (ambientes externos e mais baratos), e propõe técnicas para migrar sistemas SCADA para ambientes em nuvem.

A partir dos argumentos expostos, tendo em mente o suporte de estudos anteriores como os já citados, pode-se concluir que o uso de tecnologias em voga com os conceitos da Indústria 4.0, especificamente o uso de dados de processos industriais, coletados em sistemas SCADA, para processamento e/ou armazenamento em nuvem, são de grande relevância para as aplicações e justificam o presente estudo. Pode-se citar, em especial, a contribuição do trabalho na análise das particularidades das conexões para transferência de dados, como velocidade e segurança, além da natureza dos serviços desenvolvidos neste tipo de aplicação.

1.3 Objetivos

1.3.1 Objetivo geral

O objetivo geral deste trabalho foi desenvolver um protótipo de aplicação que visa integrar um sistema SCADA (*Supervisory Control and Data Acquisition*), utilizando o *software* de supervisão Elipse E3, com acesso a banco de dados em nuvem Microsoft Azure, com o pressuposto de avaliar a comunicação entre os dois modelos de sistemas em um ambiente industrial simulado.

1.3.2 Objetivos específicos

- Implementar uma Interface Homem Máquina (IHM) SCADA no Elipse E3 *Studio* para simular a aquisição de dados de um processo industrial;
- Implementar um acesso *Web Service* do E3 na rede corporativa;
- Configurar um sistema de banco de dados *SQL Server* no ambiente Azure da Microsoft para receber e fornecer dados à IHM SCADA;
- Elaborar a programação do sistema SCADA para ler e escrever no sistema de Banco de Dados Azure;
- Escrever um tutorial sucinto passo-a-passo das etapas de configuração do Banco de Dados em Nuvem com o SCADA;
- Aplicar a comunicação em ambiente simulado e verificar seu funcionamento e desempenho;

- Observar as particularidades de conectividade e de segurança envolvidas no processo de comunicação.

O presente estudo está dividido de maneira que cada capítulo realize uma revisão teórica do conteúdo abordado e contextualize com o trabalho proposto. No capítulo 2 é apresentada a estrutura SCADA, seus benefícios e a forma em que é utilizada no mundo contemporâneo. No capítulo 3 discute-se a importância dos bancos de dados, sua utilidade na indústria 4.0 e sua forma de aplicação explorada por este trabalho. O funcionamento do banco de dados utilizado, sua configuração e formas de conexão serão abordados no capítulo 4. Já no capítulo 5 é onde foi detalhada a configuração de comunicação entre o sistema SCADA e o banco de dados em nuvem utilizado, explicando os protocolos de comunicação e abordando questões de segurança. Por fim, o capítulo 6 sintetiza as conclusões do trabalho, indicando propostas de trabalhos futuros relacionados ao tema abordado.

2 SISTEMAS DE SUPERVISÃO E AQUISIÇÃO DE DADOS

A seguir é apresentada a importância dos sistemas de supervisão e aquisição de dados (SCADA, do inglês *Supervisory Control and Data Acquisition*) no contexto industrial, discutindo a sua estrutura, seus benefícios proporcionados e por fim a forma de uso no mundo atual.

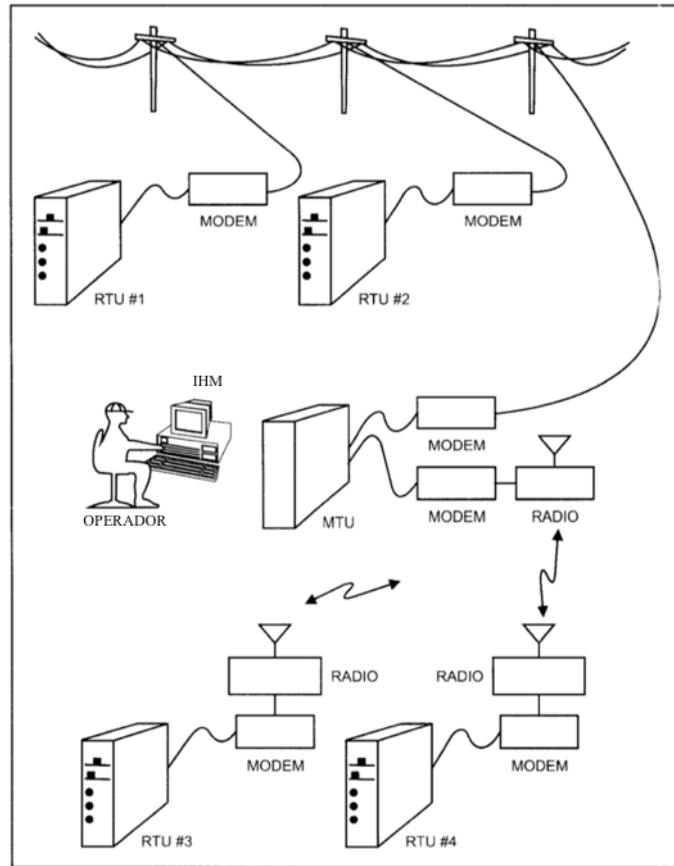
2.1 Funcionamento e Importância

SCADA é uma tecnologia que permite o usuário a coletar dados de uma ou mais instalações remotas e permite enviar algumas instruções de controle para esses ambientes. Dessa forma, com a utilização de SCADA torna-se desnecessária a permanência e visitas frequentes de operadores nessas instalações, quando estas estão em operação normal. Em outras palavras, esse sistema dá ao usuário um controle centralizado de um amplo processo distribuído, permitindo alterações de *set point* em controladores de processo distantes, abertura e fechamento de válvulas ou interruptores, monitoramento de alarmes e coleta de dados e medidas. Nesse sentido, entende-se que os custos de visita de rotina e de monitoramento reduzem e os benefícios proporcionados pela utilização de SCADA aumentam conforme as dimensões do processo e a dificuldade de acesso ao local aumentam (BOYER, 2004).

2.2 Estrutura

Um sistema SCADA é composto por diversos elementos, conforme pode-se verificar na Figura 2. O controle e monitoramento do processo é feito pelo operador, por meio de uma interface chamada console do operador ou IHM. Essa interface se comunica com a Unidade Terminal Mestre (MTU, do inglês *Master Terminal Unit*), que é uma unidade de controle e medição central do sistema. O MTU deve se comunicar com as Unidades Terminais Remotas (RTU, do inglês *Remote Terminal Unit*) do sistema, que estão em locais distantes do controle central, onde fica o operador. Essas RTUs estão ligadas à instrumentação de campo, como sensores. Por fim, deve haver uma rede de comunicação entre MTU e RTU, que normalmente é feita por cabos de fibra ótica ou pelo cabeamento elétrico, ou utilizando comunicação sem fio (BOYER, 2004).

Figura 2 – Componentes de um sistema SCADA



Fonte: Boyer (2004).
 Nota: Modificado pelo autor.

A MTU corresponde ao módulo principal do sistema SCADA. Ele é um servidor que tem a responsabilidade de realizar todo o controle do sistema. Esse servidor tem a capacidade de monitorar e controlar o processo até mesmo sem a presença do operador, por meio de armazenamentos de instruções a serem executadas em determinados intervalos de tempo (BOYER, 2004).

Como dito, as RTUs precisam se comunicar com a MTU, de modo que recebam mensagens enviadas pela MTU, executem as instruções recebidas, respondam a mensagem caso necessário e entrem em modo *stand-by*, até que uma nova mensagem seja recebida. Ou seja, é capaz de fazer aquisição, processamento e controle de dados. Como todos esses procedimentos podem ser complexos, as RTUs devem ser compostas por computadores robustos e de alto processamento. Essas unidades estão conectadas com os dispositivos de campo normalmente por meio de cabos condutores, e geralmente fornecem a energia necessária para a operação dos sensores e atuadores (BOYER, 2004).

2.3 Benefícios Proporcionados

As características do sistema SCADA proporcionam economia na execução do processo, uma vez que são reduzidos, por exemplo, os custos de visitas rotineiras nas instalações da planta para fins de monitoramento. Além disso, é possível citar outros benefícios como escalabilidade, redundância, confiabilidade e robustez, facilidade de uso e possibilidade de gerenciar alarmes. Esses ganhos se devem aos recursos disponibilizados pela estruturação do sistema SCADA que podem ser sintetizados por meio dos seguintes pontos:

- a) Monitoramento e operação do processo de forma centralizada e remota em tempo real por meio de interface interativa com o operador;
- b) Armazenamento de variáveis do processo por meio da comunicação com bancos de dados para análise e monitoramento da operação;
- c) Múltiplos servidores de dados conectados a múltiplos controladores;
- d) *Software* de redundância embutido a nível de servidor;
- e) Utilização de um *framework* estável.

O operador do sistema tem conhecimento sobre a dinâmica e o funcionamento do processo em tempo real e também tem a permissão e capacidade de realizar alterações no mesmo. Nota-se que essa dinâmica de leitura e ação do operador de forma centralizada contribui para que sistemas de grande escala funcionem de forma ágil e organizada, o que não ocorreria caso o sistema fosse descentralizado e possuísse análise desalinhada de seus componentes e variáveis. (BOYER, 2004).

De forma complementar, o sistema permite comunicação e armazenamento de dados das variáveis do sistema e dados de operação que passam a compor um banco de dados. Esses dados podem ser utilizados de forma a melhorar o planejamento e logística da produção, identificação, entendimento e previsão de tendências, possíveis falhas e necessidades de manutenção de acordo com a estruturação da planta (FRANCIS, 2016). Esse banco de dados pode ser um banco de dados local ou também um banco de dados em nuvem, como já apresentado no presente trabalho.

2.4 Uso no Mundo Contemporâneo

Segundo Sajid, Abbas e Saleem (2016), as indústrias estão sempre preocupadas em reduzir seus custos operacionais bem como outras despesas. Dessa forma, há uma constante busca por soluções cujas finalidades sejam melhorar a estabilidade de seus sistemas, a tolerância a falhas, flexibilidade e eficiência. Na indústria moderna uma das soluções adotadas é o conceito de IoT, que envolve computação em nuvem. Isso permite a integração de Sistemas Cyber-Físicos (CPS, do inglês *Cyber Physical Systems*) como SCADA, o que leva ao conceito de sistemas industriais inteligentes.

Nesse novo contexto, integrando conceitos de IoT, computação em nuvem e CPS, tem-se sistemas inteligentes com maior segurança, custos reduzidos, maior redundância e flexibilidade, onde suas aplicações são, por exemplo, em transporte inteligente, tecnologia médica inteligente, *smart grids*, controle de tráfego aéreo, dentre outros.

Nesses sistemas inteligentes, quando se fala em nível de supervisor, a maior parte da responsabilidade de sistemas SCADA é monitorar os processos do sistema e aplicar os comandos de controle apropriadamente. Sistemas SCADA são, basicamente, CPSs usados nas indústrias e possuem inúmeros setores onde podem ser aplicados, conforme é mostrado na Figura 3. Um exemplo é no setor de saúde, onde sistemas SCADA fornecem soluções que permitem que a equipe médica monitore e controle o estado de saúde de um paciente de maneira econômica e eficiente. Uma dessas soluções presente na indústria é o *WebSCADA*, que proporciona acesso ao sistema a qualquer hora e de qualquer lugar, por meio de uma conexão segura utilizando um navegador *web*. *WebSCADA* é um sistema escalável e flexível que pode facilmente se integrar com novos recursos de projeto, e é personalizável para outras aplicações industriais como petróleo, manufatura, serviços de gás, monitoramento de segurança, etc (SAJID; ABBAS; SALEEM, 2016).

Figura 3 – Setores de aplicação de sistemas SCADA



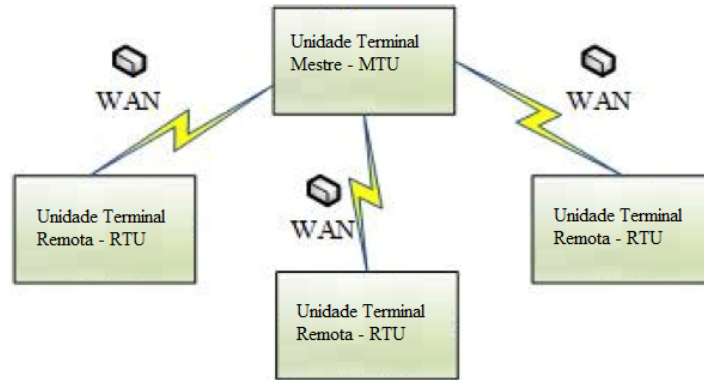
Fonte: Sajid, Abbas e Saleem (2016).

Nota: Modificado pelo autor.

De acordo com Sajid, Abbas e Saleem (2016), com o desenvolvimento da tecnologia, os sistemas SCADA evoluíram em gerações, cada uma possuindo características em sua estrutura que refletem os recursos disponíveis em cada época. As figuras a seguir descrevem a trajetória evolutiva de sistemas SCADA desde a primeira geração até a geração baseada em IoT, utilizada atualmente.

Na Figura 4 encontra-se a primeira geração dos sistemas SCADA, onde utilizava-se sistemas de *mainframe* para realizar a computação, não sendo esses sistemas interligados. Uma rede WAN era usada somente para realizar a comunicação com as RTUs. Ou seja, são sistemas monolíticos (totalmente unidos, homogêneos).

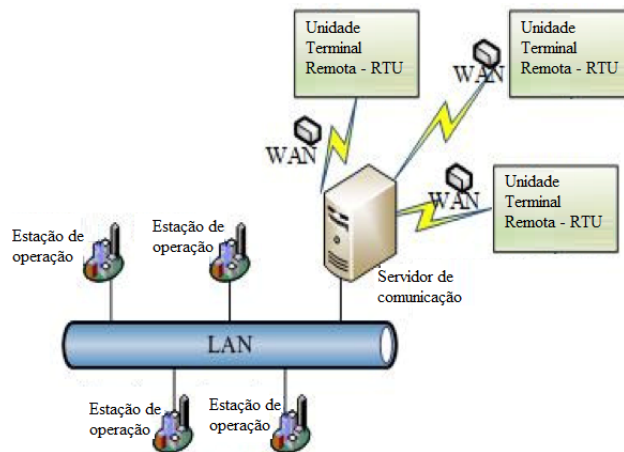
Figura 4 – Primeira geração: sistemas SCADA monolíticos



Fonte: Sajid, Abbas e Saleem (2016).
Nota: Modificado pelo autor.

A Figura 5 ilustra a segunda geração, que passou a adotar a tecnologia de LAN's. A miniaturização de sistemas fez com que os sistemas SCADA distribuídos se tornassem menores e mais baratos em relação à geração anterior. Os sistemas distribuídos aumentaram o desempenho dos sistemas SCADA em termos de redundância, poder de processamento e confiabilidade.

Figura 5 – Segunda geração: sistemas SCADA distribuídos

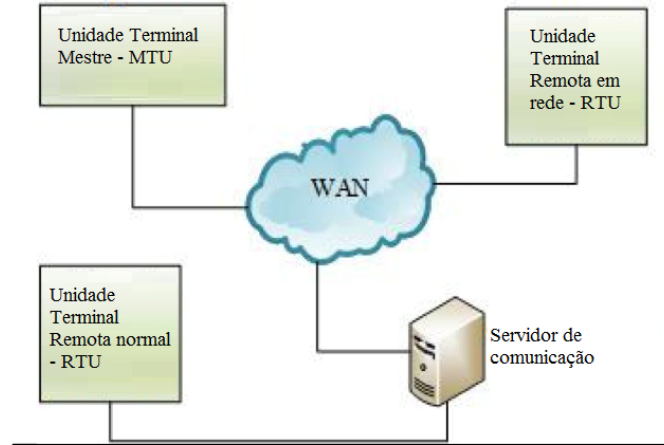


Fonte: Sajid, Abbas e Saleem (2016).
Nota: Modificado pelo autor.

As duas gerações anteriores utilizavam *softwares* e tecnologias proprietárias. Na terceira geração, representada pela Figura 6, a maior diferença é o uso de sistemas de código aberto ao invés de sistemas proprietários. Isso proporcionou a eliminação de limitações inerentes a

sistemas proprietários, dando espaço a sistemas prontos para uso. Nesta geração passou a ser utilizado o protocolo IP (*Internet Protocol*) para comunicações.

Figura 6 – Terceira geração: sistemas SCADA em rede

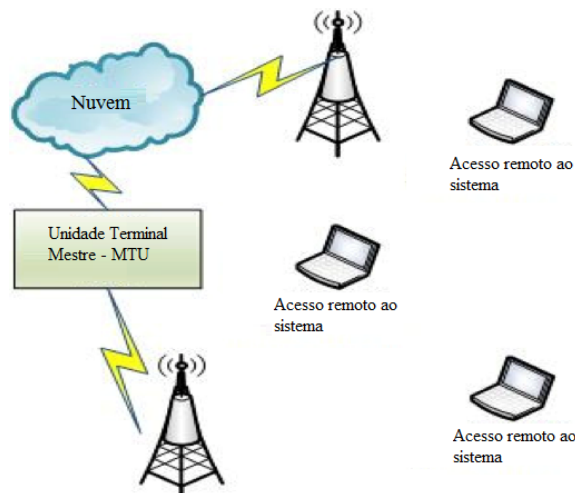


Fonte: Sajid, Abbas e Saleem (2016).

Nota: Modificado pelo autor.

Por fim, a Figura 7 representa a quarta geração de sistemas SCADA, que é a mais atual. Essa geração utiliza tecnologias de IoT e serviços de computação em nuvem. São sistemas de fácil manutenção e integração e que proporcionam uma maior acessibilidade de dados, maior eficiência econômica, flexibilidade, otimização, disponibilidade e escalabilidade.

Figura 7 – Quarta geração: sistemas SCADA baseados em IoT e nuvem



Fonte: Sajid, Abbas e Saleem (2016).

Nota: Modificado pelo autor.

Nesse sentido, a partir do que foi exposto neste capítulo, pode-se perceber a importância do uso de sistemas SCADA na indústria moderna, bem como suas aplicações e benefícios. Além disso, é possível também compreender como esses sistemas se estruturam na atualidade e como estão inseridos no contexto da indústria 4.0.

3 COMPUTAÇÃO EM NUVEM E BANCO DE DADOS

Neste capítulo há a introdução dos conceitos que envolvem computação em nuvem, bem como é feita a devida contextualização com o trabalho proposto. Também há uma breve discussão acerca de banco de dados e de suas formas de uso.

3.1 Computação em Nuvem

Para se falar de banco de dados, de como ele é utilizado no mundo contemporâneo e de como é abordado no presente trabalho, é necessário primeiramente falar de computação em nuvem. Dessa forma, pode-se definir seu conceito, seus benefícios e suas formas de serviços e, a partir disso, caracterizar o que é banco de dados em nuvem.

Segundo Fox e Hao (2018), a nuvem virtual é definida como uma coleção de tecnologias e recursos virtuais que podem ser acessados por meio de um portal da *web*. A nuvem, dessa forma, permite a conexão entre servidores e componentes de processamento, e a interface com o operador ou consumidor dos recursos ou serviços disponíveis por ela. Dentre os componentes para processamento estão aplicações ou servidores *web* conectados a conjuntos de *hardwares*, *softwares*, plataformas de desenvolvimento e serviços que compõe grandes *data centers* e *development centers* (VERAS, 2015).

Atualmente, a nuvem tem a capacidade de oferecer serviços de processamento, armazenamento, plataformas *online*, além de *softwares* para diferentes aplicações. Os serviços são ofertados de forma que não seja mais exigido do cliente local um alto poder de processamento e armazenamento (FOX; HAO, 2018). Há necessidade, entretanto, de conexões seguras e robustas para o acesso à nuvem pela rede de computadores.

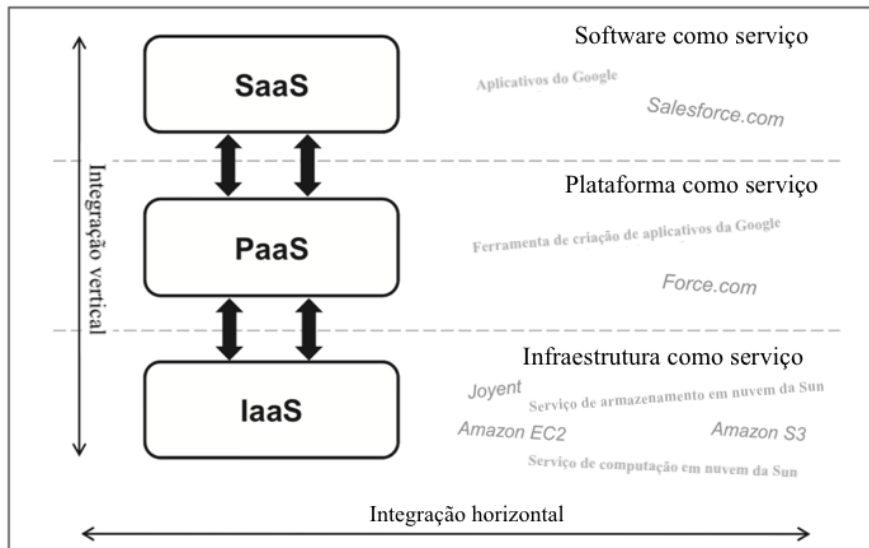
Normalmente o serviço é fornecido para as empresas com o intuito de complementar ou substituir sua infraestrutura de armazenamento e processamento (CEZAR, 2009). Os serviços fornecidos, dessa forma, apresentam maior possibilidade de personalização, de acordo os desejos do consumidor, sem necessidade de grandes investimentos em infraestrutura imediata.

Dentro das principais características e benefícios da computação em nuvem estão incluídos o atendimento e disponibilização de recursos sob demanda, agilidade e ampla elasticidade, alto

desempenho de processamento de *softwares*, redução no investimento e na manutenção dos recursos de *hardware*, facilidade no desenvolvimento de novos *web softwares* além de segurança de dados e redundância em armazenamento. Em suma, a computação em nuvem é marcada, essencialmente, por sua escalabilidade, modelo de utilização onde se paga exatamente o que usou e virtualização da infraestrutura (STANOEVSKA-SLABEVA; WOZNIAK; RISTOL, 2010).

Stanoevska-Slabeva, Wozniak e Ristol (2010) afirmam que a computação em nuvem compreende diferentes recursos de TI, que são infraestrutura, plataformas e *software*. Esses recursos podem ser chamados de camadas, pois são construídos subsequentemente ao nível anterior e estão logicamente conectados como camadas diferentes de uma arquitetura em nuvem. Dessa forma, as três camadas da computação em nuvem são infraestrutura como serviço (IaaS, do inglês *Infrastructure as a Service*), plataforma como serviço (PaaS, do inglês *Platform as a Service*) e *software* como serviço (SaaS, do inglês *Software as a Service*). A Figura 8 ilustra a organização dessas camadas.

Figura 8 – As três camadas da computação em nuvem: SaaS, PaaS e IaaS



Fonte: Stanoevska-Slabeva, Wozniak e Ristol (2010).

Nota: Modificado pelo autor.

Quando se fala de IaaS, pode-se destacar como característica principal a oferta de recursos computacionais, como processamento ou armazenamento, que podem ser obtidos como um serviço. Os provedores de PaaS e SaaS podem usufruir dos benefícios de IaaS utilizando interfaces padronizadas. Ou seja, ao invés de vender puramente uma infraestrutura de

hardware, provedores de IaaS normalmente fornecem uma infraestrutura virtualizada como serviço (STANOEVSKA-SLABEVA; WOZNIAK; RISTOL, 2010). Dessa forma, recursos a nível de *hardware* são abstraídos e encapsulados e podem, portanto, serem expostos a camadas superiores e aos usuários finais por meio de uma interface padronizada como recursos unificados na forma de IaaS (FOSTER et al., 2008).

Já para PaaS, Stanoevska-Slabeva, Wozniak e Ristol (2010) afirmam que plataformas são uma camada de abstração entre as aplicações de *software* (SaaS) e a infraestrutura virtualizada (IaaS). Os serviços de PaaS são voltados para desenvolvedores de *software*, que criam seus aplicativos de acordo com as especificações de uma determinada plataforma, sem se preocuparem com a infraestrutura de *hardware* (IaaS) por trás disso. Nesse sentido, os desenvolvedores enviam seus códigos de programação para a plataforma, que normalmente faz o gerenciamento conforme o uso do aplicativo aumenta. Em suma, os serviços de PaaS podem cobrir todas as etapas de desenvolvimento de *software* ou podem ser especializadas em uma área específica, como gerenciamento de conteúdo.

A terceira camada da computação em nuvem é a de *software* como serviço. SaaS é um *software* que pertence, é entregue e é gerenciado por um ou mais fornecedores, e que é fornecido no modelo de “pague por uso”. Isto é, paga-se exatamente o que foi utilizado. Essa é a camada mais visível de computação em nuvem para os usuários finais, uma vez que são exatamente as aplicações de *software*, que são acessadas e utilizadas por eles. Da perspectiva do usuário, utilizar *software* como serviço é interessante principalmente pelas vantagens de custo, devido à forma com que esses serviços são contratados e pagos. Ou seja, não é necessário investir em infraestrutura e paga-se somente o que foi utilizado (STANOEVSKA-SLABEVA; WOZNIAK; RISTOL, 2010).

Normalmente os usuários de serviços SaaS não têm conhecimento nem controle sobre a infraestrutura que opera por trás, seja sobre a plataforma de *software* (PaaS) ou sobre a infraestrutura de *hardware* (IaaS). Entretanto, essas camadas são muito importantes para fornecedores SaaS, pois são necessárias e podem ser terceirizadas, isto é, uma aplicação SaaS pode ser desenvolvida em uma plataforma já existente e operar sobre uma infraestrutura de terceiros. Como exemplo deste tipo de serviço, é possível citar as ferramentas de e-mail, de

edição de texto, de apresentações e de planilhas oferecidas de forma *online* (STANOEVSKA-SLABEVA; WOZNIAK; RISTOL, 2010).

Todas as três possibilidades de serviço são muito utilizadas no contexto de avanço tecnológico nas pequenas, médias e grandes empresas, com o objetivo de otimizar os seus processos além de armazenar seus dados de forma mais fácil, compartilhada e segura.

3.2 Banco de Dados em Nuvem

3.2.1 Definição de banco de dados

Segundo Puga, França e Goya (2014), um banco de dados é definido como um conjunto de dados, que são organizados e armazenados para que se atendam as necessidades integradas dos seus usuários. Ele permite que haja consulta e manipulação de dados, e pode ser classificado como manual ou computadorizado. Um exemplo de banco de dados manual é o prontuário médico, que é guardado em pastas e é organizado por ordem alfabética, de acordo com o nome do paciente. Já no banco de dados computadorizado, há o armazenamento de dados em estruturas organizadas e implementadas em um *software*.

Um sistema de banco de dados, de maneira geral, consiste em um sistema computadorizado de armazenamento de dados no qual seus usuários têm a capacidade e a possibilidade de ter acesso, realizar alterações, exclusões e a inserção de dados e de arquivos. Esses sistemas de banco de dados são compostos por *hardware*, voltados ao amplo volume de armazenamento e de processamento, além do *Software* de Gerenciamento do Banco de Dados (SGBD) que permite a visualização do mesmo e a realização de ações e operações por parte do usuário (DATE, 2013).

3.2.2 Bancos de dados relacionais e não relacionais

Os bancos de dados podem ser classificados em relacionais e não relacionais. No mundo atual, onde há grandes avanços na área de *Big Data* decorrentes do crescimento contínuo do volume de dados gerados por empresas e indústrias, escolher qual desses bancos de dados utilizar é fundamental para o bom desempenho dos processos.

3.2.2.1 Banco de dados relacional

É o tipo de banco de dados mais popular no mercado. Foi criado em 1970 por Edgar Frank Codd, e consiste em um conjunto de dados organizados em tabelas que possuem uma certa estrutura, das quais os dados podem ser acessados ou manipulados. A estrutura dessas tabelas que compõem o banco de dados é feita pela categorização dos dados em colunas. Cada linha contém uma instância de um dado e cada coluna contém informação desse dado que se encaixa na respectiva categoria. Ao se criar as tabelas e suas estruturas, é possível também aplicar restrições aos dados que serão inseridos. Como esse tipo de banco de dados apresenta relações entre suas tabelas, surge o nome banco de dados relacional (JATANA, 2012).

A criação de tabelas, o acesso e a manipulação dos dados são feitos por meio da Linguagem de Consulta Estruturada (SQL, do inglês *Structured Query Language*) e é por meio dela que se interage com o banco de dados relacional. Como essa linguagem tem origem em cálculo e álgebra relacional, é subdividida em elementos como cláusulas, predicados, consultas, etc.

3.2.2.2 Banco de dados não relacional

Essa classe de sistemas difere amplamente dos sistemas relacionais de várias maneiras significativas, sendo a mais importante delas o fato de não utilizar relações (ou tabelas) como forma de armazenamento. Outras diferenças são a não utilização de SQL como linguagem de consulta, adotando o conceito de NoSQL (*Not Only SQL*), e a super-escalabilidade.

Como não são usadas tabelas tradicionais, esses bancos de dados podem armazenar os dados e realizar consultas de várias formas diferentes, seja por chave-valor, formato XML, bancos de dados multidimensionais, dentre outros.

3.2.2.3 Diferenças entre banco de dados relacional e não relacional

A maior diferença entre esses dois tipos de banco de dados está na forma como os dados são armazenados e organizados. O banco de dados relacional apresenta maior consistência e confiabilidade, exigindo para tanto o relacionamento entre tabelas. Já o banco de dados não relacional é a solução mais adequada para o crescente aumento de armazenamento de dados exigido atualmente, sendo apropriada para trabalhos que envolvem dados em grande escala.

Jatana et. al (2012) estabelece as seguintes diferenças entre o modelo relacional e o não relacional:

- Alta transferência de dados apresentada pelo modelo não relacional, enquanto o modelo relacional apresenta baixa transferência.
- O modelo relacional tem escalabilidade, mas o não relacional tem a característica de super-escalabilidade, sendo adequado para situações de grande volume de armazenamento de dados.
- O modelo não relacional não exige uma estrutura para inserir os dados, enquanto no relacional os dados precisam se encaixar nas estruturas pré-definidas das tabelas.
- Uso de linguagem SQL pelo modelo relacional.
- O modelo relacional tem maior consistência que bancos de dados não relacionais, uma vez que estes permitem dados duplicados, ameaçando a integridade dos dados, o que não acontece nos bancos de dados relacionais, uma vez que sua estrutura elimina registros duplicados e previne a existência de dados inconsistentes.
- O processo de busca nos bancos de dados não relacionais é mais ineficiente que o modelo relacional, dado que nos bancos de dados relacionais a sua organização permite que a linguagem SQL use chaves primárias, que são compartilhadas entre tabelas, para buscar e retornar os registros desejados de maneira ágil e eficiente.

O tipo de banco de dados utilizado neste trabalho é o relacional, dado que suas características de consistência, confiabilidade e escalabilidade atendem a maior parte das necessidades de mercado atuais, além de ser o modelo mais popular utilizado.

3.2.3 Operação em nuvem e benefícios

O *hardware* de um banco de dados relacional pode ser adquirido e mantido por meio do próprio consumidor do acesso, ou pode-se contratar um serviço terceirizado. Uma das formas de realizar a terceirização desse serviço é por meio de contratação do serviço de banco de dados em nuvem.

O serviço de banco de dados em nuvem corresponde a um dos formatos de Computação em Nuvem ofertados atualmente como PaaS. Essa forma de serviço traz consigo todos os benefícios atrelados ao conceito de nuvem virtual. Isto é, provê escalabilidade, agilidade e elasticidade,

maior desempenho de processamento de *softwares* e redução dos custos de investimento em *hardware*.

Quando se fala de escalabilidade, por exemplo, é importante lembrar da dificuldade de se prever o quão popular um projeto será e o quanto de recursos será demandado. Dessa forma, é necessário que os bancos de dados dessas aplicações tenham uma característica inerente de escalabilidade, que se reflete tanto em termos de *software* como *hardware*. Exemplificando, quando há picos de demanda de recursos computacionais, a computação em nuvem consegue atender de maneira mais rápida e eficiente do que em relação à simples compra de mais máquinas físicas, que seria feita no caso onde não se utiliza computação em nuvem. Até mesmo em casos onde a popularidade e o uso da aplicação caem é possível diminuir com facilidade as dimensões dos recursos disponíveis, o que seria mais complicado em casos com *hardware* físico (FIORE; ALOISIO, 2011).

Existem, atualmente, diversas empresas que fornecem serviços de banco de dados em nuvem. Segundo Fiore e Aloisio (2011), a maior parte desses provedores em nuvem oferece um serviço de banco de dados que funciona perfeitamente com os bancos de dados já existentes, juntamente com seus códigos, aplicações e ferramentas. O SQL Azure, que é utilizado neste trabalho, é um desses serviços e, como é compatível com as plataformas SQL já existentes, não requer alterações nessas aplicações.

Os benefícios de se utilizar um banco de dados em nuvem variam, mas geralmente incluem as possibilidades de gerenciamento, melhorias, atualização e *backup* de forma automática, o que reduz os custos totais que representam o banco de dados. Outra vantagem de se utilizar esse tipo de serviço é, por exemplo, o fato de os dados funcionarem em várias instâncias diferentes de banco de dados, de modo a garantir alta disponibilidade, como é o caso do SQL Azure utilizado no presente trabalho (FIORE; ALOISIO, 2011).

Há, ainda na filosofia de nuvem, outros benefícios quanto à comodidade de *hardware* e escalabilidade. Segundo Fiore e Aloisio (2011), uma única instância de banco de dados será eventualmente um gargalo no sistema, sendo vantajoso, portanto, aplicar técnicas para contornar esse problema. O que normalmente pode ser feito é a melhoria do *hardware* do banco de dados, de modo a gerar melhor desempenho. Entretanto, isso gera custo e tem limitação do

hardware adquirido. Uma forma melhor de lidar com essa situação é aproveitando os benefícios de banco de dados em nuvem, realizando o que é conhecido como fragmentação (e compartilhamento) de banco de dados. Nessa técnica, um grande banco de dados é dividido em várias instâncias menores de banco de dados, de modo que a carga seja distribuída de forma balanceada e que resulte em melhor escalabilidade, disponibilidade e desempenho. Dessa forma, aproveita-se a escalabilidade e o *hardware* fornecidos pelo serviço em nuvem para evitar possíveis sobrecargas de processamento.

Na Era da Informação em que vivemos, utilizar um sistema integrado que armazene e possibilite acesso aos dados de maneira eficiente, como os bancos de dados em nuvem, é tratado como sobrevivência de mercado. Isto é, essa forma de utilizar os bancos de dados conquista a cada dia um espaço maior no mundo corporativo, e isso se deve a todas as ferramentas e vantagens proporcionadas por esse tipo de serviço mencionadas anteriormente. Tendo isso em vista, foi escolhida para a realização deste trabalho uma plataforma moderna e muito utilizada, que atenda as necessidades de mercado e apresente vantagens em sua utilização, o SQL Azure. Na seção seguinte será discutido sobre a linguagem SQL e sobre um dos principais SGBDs do mercado atual, para que na posterior seção seja possível entender e abordar em maiores detalhes as características e o funcionamento do SQL do Azure.

3.2.4 Microsoft SQL Server

A linguagem SQL é uma linguagem de banco de dados utilizada para formular instruções que são processadas por um servidor de banco de dados (VAN DER LANS, 2006). Em outras palavras, SQL é uma linguagem padrão projetada para realizar o gerenciamento de dados em um banco de dados de modelo relacional, ou seja, um modelo que o usuário visualiza seus dados por meio de tabelas e pode ser manipulado de forma relacional. O Microsoft SQL Server, dessa forma, é um SGBD em modelo relacional, desenvolvido pela Microsoft, que implementa o SQL e corresponde a parte central da plataforma de dados da Microsoft (MICROSOFT, 2017d).

A ferramenta apresenta mecanismos de bancos de dados que corresponde ao “[...] serviço principal para armazenamento, processamento e segurança de dados [...]” (MICROSOFT, 2019i). Além disso, ela possibilita, por exemplo, a adição de complementos que tratam de *Machine Learning*, para fornecimento de análise avançada, e *Analysis Services*, para criar relatórios e aplicativos por meio de serviços de ferramentas de visualização *online*, a exemplo

do Excel e Power BI, além de também poder ser utilizada para realizar mineração de dados, de modo a descobrir padrões e relações ocultas dentro de grandes volumes de dados. (MICROSOFT, 2019i).

O *SQL Server* possui uma ferramenta de gerenciamento integrado de infraestruturas SQL, o *Management Studio* (SSMS). Nela estão incluídas ferramentas gráficas e editores de *scripts* que podem ser utilizados para acessar, configurar, gerenciar, administrar e desenvolver todos os componentes do *SQL Server*, do Banco de Dados SQL do Azure e do *SQL Data Warehouse*. É possível, portanto, consultar, criar e gerenciar bancos de dados e *data warehouses*, independentemente da localização do usuário. (MICROSOFT, 2019j).

3.2.5 Banco de dados SQL do Azure

O Microsoft Azure é uma plataforma que oferece diferentes serviços em nuvem que estão inseridas nas três formas de serviço: IaaS, PaaS e SaaS. Dentre os serviços PaaS oferecidos está o Banco de Dados SQL, que é um serviço gerenciado de banco de dados relacional de uso geral no Microsoft Azure. Esse banco de dados permite criar uma camada de armazenamento com grande disponibilidade e alto desempenho para os aplicativos e soluções, e dá suporte a estruturas como XML [*Extensible Markup Language*], JSON [*JavaScript Object Notation*], espacial e dados relacionais (MICROSOFT, 2019q).

Esse serviço compartilha uma base de código comum com o *SQL Server*. Nesse sentido, boa parte dos recursos do *SQL Server* são compatíveis com o Banco de Dados SQL do Azure, dependendo do tipo de Banco de Dados SQL do Azure criado. Seu gerenciamento, por exemplo, é realizado por meio da ferramenta SSMS (MICROSOFT, 2019e).

Devido ao armazenamento em nuvem dos dados, a solução oferece, além dos benefícios oferecidos pelo *SQL Server*, escalabilidade dinâmica sem inatividade, opções de segurança avançadas e os demais benefícios em decorrência do serviço PaaS, como já abordado no presente trabalho. Entretanto, o Banco de Dados SQL do Azure é mais do que um mecanismo de banco de dados relacional clássico, pois também oferece recursos avançados como sincronização de dados, automação de trabalho, tecnologias *in-memory* e replicação transacional.

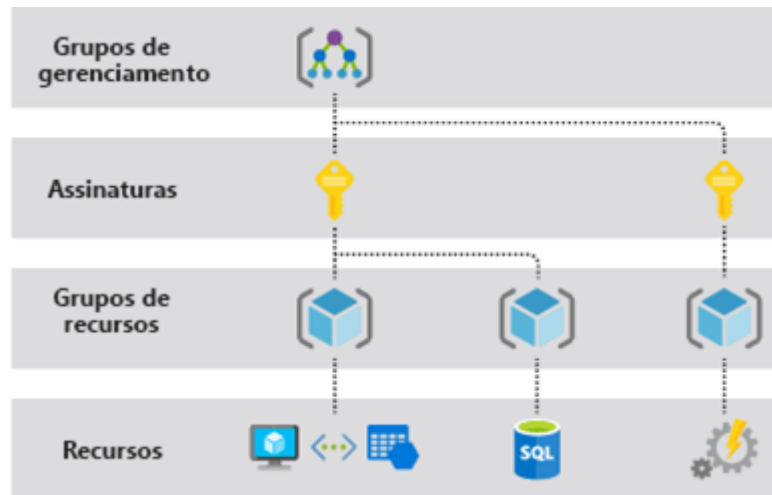
Em se tratando dos benefícios decorrentes da utilização do SQL do Azure, pode-se ressaltar que ele é um serviço de banco de dados totalmente gerenciado, de modo que a empresa, a Microsoft, opera o *SQL Server* para o cliente e garante disponibilidade e desempenho. Isso significa que não é necessário dispêndio de tempo e recursos para gerenciar o banco de dados, seja aplicando *patches* ou realizando *backups*. Além disso, ainda estão inclusos recursos que têm como objetivo aumentar a continuidade e segurança, permitindo por exemplo a implementação de recursos inteligentes como detecção de ameaças.

Conforme explicado ao longo deste capítulo, o SQL Azure apresenta ampla compatibilidade com o mecanismo do *SQL Server*, de modo a compartilhar uma base de código comum com o *SQL Server*. Isso significa que a linguagem utilizada pelo banco de dados do Azure é a linguagem SQL, sendo a maior parte dos recursos do SQL idênticos entre *SQL Server* e os bancos de dados SQL do Azure, tais como processamento de consulta e recursos de gerenciamento de banco de dados. Nesse sentido, o Banco de Dados SQL do Azure suporta diversas ferramentas que auxiliam seu gerenciamento. Isto é, diversos recursos como tabelas e regras de *Firewall* podem ser criados e configurados por meio de diferentes alternativas como o portal do Azure, Azure CLI, Azure PowerShell, *SQL Server Management Studio*, *SQL Server Data Tools* para *Visual Studio* e extensão SQL para *Visual Studio Code*, sem necessariamente ter conhecimentos na linguagem SQL dependendo da ferramenta utilizada.

4 PLATAFORMA AZURE E CONFIGURAÇÕES

Para entender como funcionam e como se criam e configuram as estruturas e os recursos disponíveis na plataforma Azure, necessários à realização do presente trabalho, é preciso primeiramente entender como se dá sua organização nessa plataforma. Isto é, o Azure organiza sua estrutura em um escopo, que é composto por quatro níveis hierárquicos. Esses níveis são classificados como grupos de gerenciamento, assinaturas, grupos de recursos e recursos. A Figura 9 ilustra graficamente esse escopo.

Figura 9 – Níveis de escopo do Azure



Fonte: Microsoft (2019b).

Nesse cenário, é importante entender o significado desses níveis, sua função e a correlação que eles apresentam entre si. Dessa forma, pode-se definir os recursos como itens gerenciáveis, disponíveis por meio do Azure. Exemplos de recursos são máquinas virtuais, aplicativos *Web* e os bancos de dados e servidores de bancos de dados que são utilizados neste trabalho. Para se criar recursos, é necessário atribuí-los a grupos de recursos, que são *containers* que agrupam recursos relacionados a uma solução do Azure. Da mesma forma, esses grupos de recursos precisam estar associados a assinaturas, que por sua vez poderão ou não estar associadas a grupos de gerenciamento.

Nos níveis do escopo apresentado pela Figura 9 podem ser aplicadas configurações de gerenciamento. O nível escolhido determina a profundidade da configuração aplicada, uma vez que essa configuração será herdada pelos níveis inferiores. Isto é, ao se aplicar uma

configuração em uma assinatura, ela será herdada por todos os grupos de recursos dessa assinatura e por todos os recursos associados a eles. Todavia, se for aplicada uma configuração em um determinado grupo de recursos, essa configuração é aplicada somente ao próprio grupo de recursos e aos recursos associados, não sendo os demais grupos de recursos afetados (MICROSOFT, 2019b).

4.1 Grupos de Recursos

Uma vez entendida a lógica de funcionamento do Azure, bem como a relação entre seus níveis, pode-se entrar em mais detalhes a respeito das características e da função dos grupos de recursos. Conforme dito anteriormente, esses grupos são *containers* onde se colocam os recursos ofertados pelo Azure e possuem alguns fatores que devem ser considerados na sua construção.

Um desses fatores é que todos os recursos presentes em um grupo devem compartilhar o mesmo ciclo de vida (recursos que normalmente são implementados, atualizados e deletados em conjunto). Isto é, se algum recurso precisar ser implementado em um momento distinto dos demais recursos, ele deverá pertencer a outro grupo de recursos. Além disso, cada recurso deve pertencer a somente um grupo de recursos, podendo também ser apagado, adicionado ou movido a outro grupo de recursos a qualquer instante desejado. Outro fator é que um grupo de recursos pode conter recursos que estão localizados em diferentes regiões, podendo também interagir com recursos que pertencem a outro grupo de recursos. Essa interação é mais frequente entre recursos que estão relacionados, mas que não compartilham do mesmo ciclo de vida, como um aplicativo *Web* se conectando a um banco de dados (MICROSOFT, 2019b).

Nesse sentido, os grupos de recursos tornam-se importantes no presente trabalho pois agrupam o servidor e o banco de dados utilizado. Ou seja, para que seja criado um banco de dados, que deve estar contido em um servidor de banco de dados, é necessário também criar um grupo de recursos que agrupará este servidor e o banco de dados. Para criar um grupo de recursos, é necessário escolher uma assinatura a qual este grupo será vinculado, conforme visto no escopo do Azure, e também a região na qual este grupo estará localizado. Essa etapa de criação pode ser feita de várias formas, como por exemplo pelo portal do Azure.

4.2 Servidores de Banco de Dados SQL do Azure

Um servidor de Banco de Dados SQL é uma estrutura lógica que atua como ponto administrativo central para vários bancos de dados individuais, *logins*, regras de *firewall*, regras de auditoria, políticas de detecção de ameaças, dentre outros. Isto é, um servidor é um *container* lógico que precisa ser criado antes de se criar um banco de dados, podendo gerenciar todos os demais recursos essenciais à operação de bancos de dados (MICROSOFT, 2019f).

Um servidor de bancos de dados possui diversas características, sendo o recurso pai de bancos de dados, *pools* elásticos (ferramenta para gerenciar de forma eficiente os recursos de um grupo de bancos de dados) e *data warehouses* (armazém de dados voltado para assuntos empresariais, decisões de gerência). Nesse sentido, ele provê um escopo para políticas de gerenciamento que podem ser aplicadas em bancos de dados, como *logins*, *firewall*, detecção de ameaças e outros. Como é um *container* lógico e possui uma forte semântica de tempo de vida, quando se deleta um servidor os bancos de dados e demais recursos contidos por ele também são deletados.

Outro ponto, segundo Microsoft (2019f), é que o servidor de banco de dados aloca seus recursos em uma região. Isto é, todos os bancos de dados gerenciados por esse servidor são criados na mesma região em que o servidor opera. Além disso, é o servidor que fornece um ponto de conexão para acessar o banco de dados (no caso do Azure, por meio do endereço <NomeDoServidor>.database.windows.net). Por ser um recurso pai e de ordem superior na hierarquia, *logins* a nível de servidor podem gerenciar todos os bancos de dados contidos por esse servidor. Essa relação de acesso a nível de servidor ou a nível de banco de dados será explicada em maiores detalhes na seção de *firewall*, discutida mais a frente.

4.3 Criando um Servidor de Banco de Dados SQL

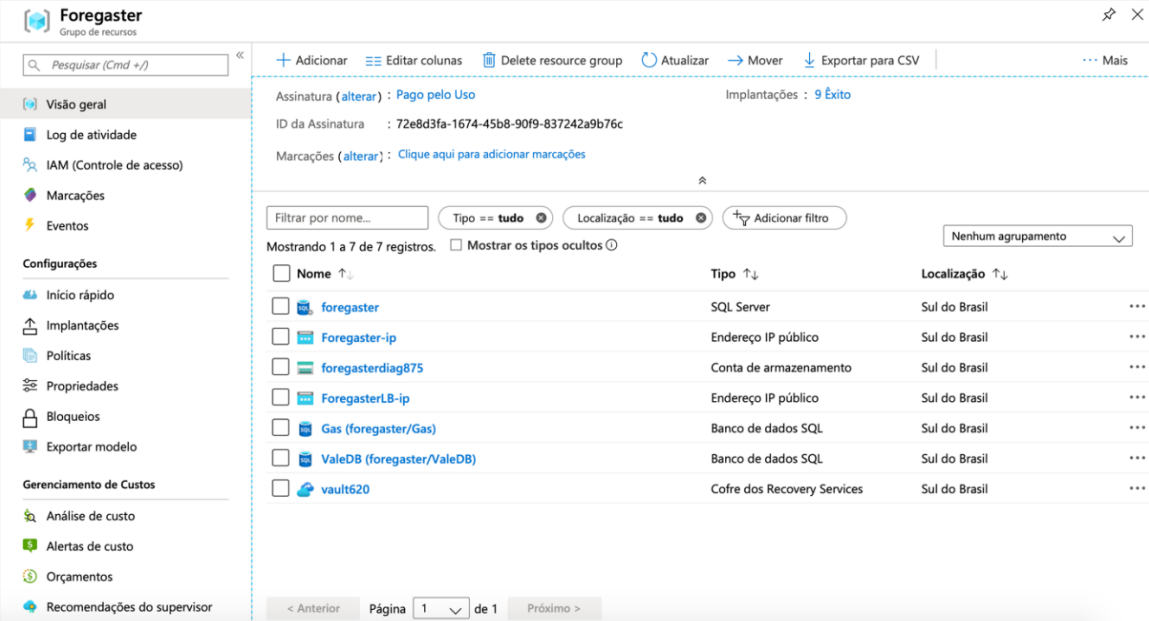
Conforme discutido anteriormente e visto pelos níveis de escopo, ilustrados pela Figura 9, os servidores de bancos de dados são classificados como recursos e, como tais, devem estar associados a algum grupo de recursos. O servidor de banco de dados poderá ser criado em uma região diferente do seu grupo de recursos, o que levanta dúvidas a respeito da necessidade de definir um local (ou região) para o grupo de recursos. Todavia, segundo Microsoft (2019b), isso é explicado pois esse grupo armazena metadados a respeito dos recursos que ele contém. Ou seja, a região especificada para o grupo de recursos é a localidade onde os metadados serão

armazenados. Por razões de comodidade, pode ser desejável garantir que os dados sejam armazenados em uma região específica. Em outras palavras, quando a localidade do grupo de recursos estiver indisponível, os servidores de bancos de dados contidos por ele não poderão ser atualizados uma vez que os metadados estarão indisponíveis. Entretanto, mesmo sem a possibilidade de atualização, os servidores que estiverem localizados em outras regiões permanecerão funcionando normalmente.

Há diversas maneiras de se criar um servidor de banco de dados. Dentre as possibilidades, algumas são o portal do Azure, o Azure PowerShell e o Azure CLI. Para realizar a criação por meio do PowerShell, será necessário instalar o módulo Azure PowerShell e utilizar o comando específico de criação, fornecendo alguns parâmetros obrigatórios. Para criar o servidor por meio do Azure CLI, é necessário primeiramente instalar a interface de linha de comando e da mesma forma aplicar o comando de criação fornecendo certos parâmetros. Neste trabalho, a criação do servidor de banco de dados é feita pelo portal do Azure, sendo mais simples e intuitivo.

No momento da criação, é necessário informar a assinatura e o grupo de recursos aos quais o servidor será vinculado. Neste trabalho, foi utilizada apenas uma assinatura e um grupo de recursos, cujo nome se deu por Foregaster. A Figura 10 mostra o painel de gerenciamento desse grupo de recursos pelo portal do Azure, bem como todos os recursos que ele contém.

Figura 10 – Painel de gerenciamento do grupo de recursos Foregaster



The screenshot shows the Azure portal interface for the 'Foregaster' resource group. The left sidebar contains navigation options like 'Visão geral', 'Log de atividade', 'IAM', 'Configurações', and 'Gerenciamento de Custos'. The main area displays the group's details, including the subscription ID and a list of resources. The resources are listed in a table with columns for Name, Type, and Location.

Nome	Tipo	Localização
foregaster	SQL Server	Sul do Brasil
Foregaster-ip	Endereço IP público	Sul do Brasil
foregasterdiag875	Conta de armazenamento	Sul do Brasil
ForegasterLB-ip	Endereço IP público	Sul do Brasil
Gas (foregaster/Gas)	Banco de dados SQL	Sul do Brasil
ValeDB (foregaster/ValeDB)	Banco de dados SQL	Sul do Brasil
vault620	Cofre dos Recovery Services	Sul do Brasil

Fonte: Produção do próprio autor.

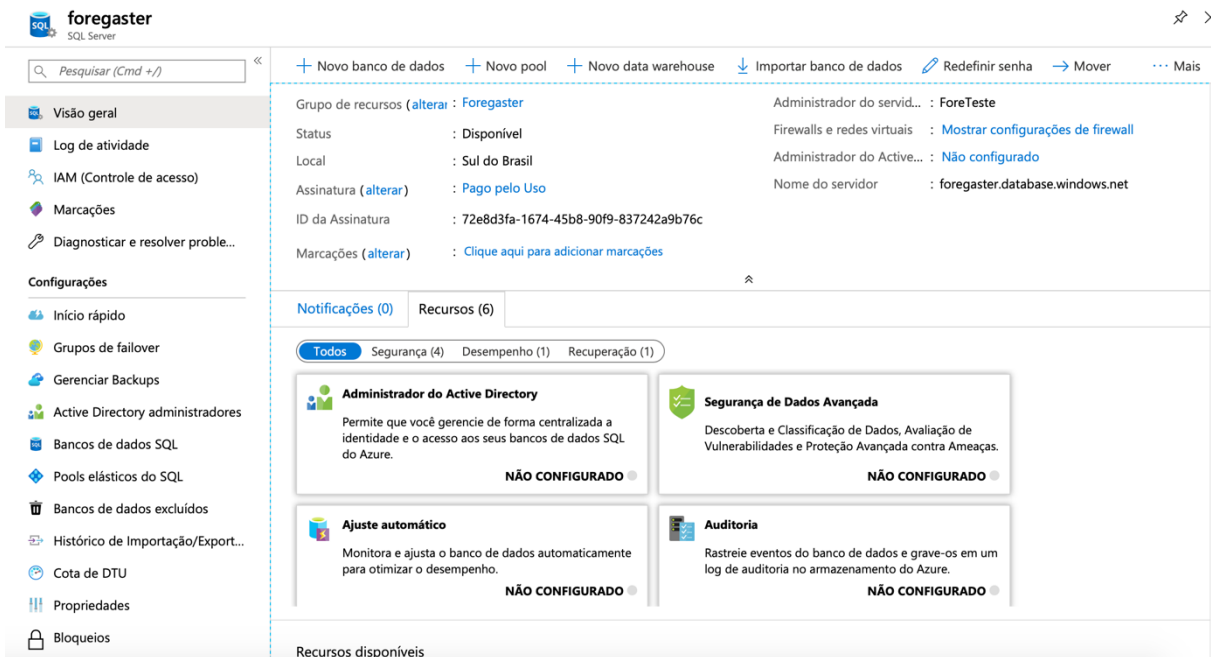
Em seguida, faz-se necessário o preenchimento da seção denominada Detalhes do servidor. Nela, é necessário definir um nome para o servidor, que também foi escolhido como *foregaster*, ao qual será atribuído um endereço para conexão. Esse endereço é a junção do nome do servidor com o pós-fixado “.database.windows.net” e é utilizado tanto para o gerenciamento dos bancos de dados em plataformas como o SSMS quanto para a conexão do sistema SCADA com o banco de dados. Nesse sentido, o endereço resultante neste caso em específico foi *foregaster.database.windows.net*.

O próximo passo é definir a localização do servidor de banco de dados que, conforme explicado em seções anteriores, pode ser diferente do grupo de recursos escolhido. Apesar dessa possibilidade, tanto a localização do grupo *Foregaster* como a do servidor *foregaster* foram definidas como (America do Sul) Sul do Brasil.

Por fim, a última informação obrigatória refere-se à conta de administrador. Para tanto, é necessário informar uma conta de *login* e uma senha que serão utilizados para criar uma conta com direitos administrativos, que poderá gerenciar tanto o banco de dados mestre daquele servidor como todos os outros bancos de dados criados nele. Dessa forma, essa conta é utilizada para fazer conexão com o servidor e com os bancos de dados por meio das diversas plataformas de gerenciamento, como o portal do Azure e o SSMS, funcionando como uma conta de *login* SQL. Segundo Microsoft (2019f), o Banco de Dados SQL do Azure dá suporte a dois tipos de autenticação: a Autenticação SQL e a Autenticação do Azure Active Directory, que são discutidas em maiores detalhes em seções posteriores. Portanto, a conta administrativa definida no momento da criação do servidor de banco de dados é utilizada na autenticação SQL, que possui foco especial neste trabalho.

É possível, ainda, configurar no momento da criação outros recursos do servidor, como regras de *firewall*, recurso de segurança de dados avançada e rótulos. Entretanto, essas configurações são opcionais e neste trabalho são feitas em um momento posterior, sendo detalhadas ao longo deste capítulo. Por meio da Figura 11, pode-se ver o painel administrativo fornecido pelo portal do Azure, que proporciona ferramentas para gerenciar o servidor *foregaster* criado. Nele, é possível identificar alguns dados do servidor, como a conta de administrador, o nome (e endereço) do servidor, a localização, o tipo de assinatura, bem como outros recursos de segurança e de gerenciamento disponíveis para uso.

Figura 11 – Painel de gerenciamento do servidor de banco de dados foregaster



Fonte: Produção do próprio autor.

4.4 Usuários e Logins do SQL do Azure

Quando se fala em conta de administrador, essencial para a criação do servidor de banco de dados conforme visto anteriormente, é importante lembrar que ela é um *login* usado para autenticação SQL. Isto é, é por meio desse *login* que o administrador do banco de dados consegue se conectar ao servidor e, conseqüentemente, a todos os bancos de dados contidos nesse servidor. Todavia, é importante destacar que quando se trabalha com outros funcionários que também precisam ter acesso aos bancos de dados, é primordial que sejam criados outros *logins* que possuam os acessos e as permissões adequadas a cada funcionário, estabelecidas de acordo com a natureza do cargo e função exercida por ele.

Entretanto, é necessário compreender que há dois modelos usados para autenticar a conexão com os servidores e bancos de dados: modelo tradicional baseado em *login* e usuário, e o modelo de usuário em banco de dados independente. A Microsoft recomenda a utilização do modelo de usuário em banco de dados independente, embora reconheça que há alguns casos específicos onde a logística de negócio ou a capacidade de gerenciamento ainda exijam o uso do modelo tradicional baseado em *login* e usuário. Nas seções a seguir serão apresentados os dois modelos, juntamente com as suas diferenças.

4.4.1 Modelo tradicional baseado em *login* e usuário

Neste modelo, a conexão realizada com o Mecanismo de Banco de Dados é feita por meio do uso de um nome de *login* e senha, utilizando-se a autenticação do *SQL Server*. O banco de dados mestre, contido no servidor de banco de dados, precisa possuir um *login* que corresponda com as credenciais fornecidas na conexão. Depois que o Mecanismo de Banco de Dados autentica as credenciais da autenticação *SQL Server*, normalmente tenta-se conectar com um banco de dados do usuário, podendo esse banco ser especificado no ato da conexão com o servidor. Para que haja essa conexão, o *login* precisa estar associado a um usuário nesse banco de dados (MICROSOFT, 2019h).

Nesse processo, o mais importante é que tanto o *login* (existente no banco de dados mestre) como o usuário (existente no banco de dados a ser conectado) precisam existir e estar relacionados um com o outro. Nesse sentido, a conexão com o banco de dados do usuário apresenta uma dependência em relação ao *login* existente no banco de dados mestre, o que limita a capacidade de mover um banco de dados para um servidor de banco de dados *SQL* diferente. Além disso, se por alguma razão a conexão com o banco de dados mestre não estiver disponível, o tempo de conexão total será maior e o tempo de conexão poderá ser esgotado, reduzindo a escalabilidade de conexão (MICROSOFT, 2019h).

4.4.2 Modelo de usuário em banco de dados independente

Neste modelo, não há *logins* no banco de dados mestre. Dessa forma, a autenticação passa a ser feita com o banco de dados do usuário, de modo que não haja mais a associação do usuário existente no banco de dados em questão com algum *login* existente no banco de dados mestre. Isso faz com que o banco de dados possa ser movido para outro servidor de banco de dados mais facilmente. Entretanto, a conexão neste modelo exige que um banco de dados seja especificado para que o Mecanismo de Banco de Dados saiba qual banco de dados será responsável pelo processo de autenticação. Como o usuário contido no banco de dados está limitado a este banco, caso haja a necessidade de conexões a outros bancos de dados será preciso criar um usuário em cada um desses bancos. Dessa forma, para que se alterne entre bancos de dados, é necessário realizar uma nova conexão (MICROSOFT, 2019h).

É importante entender que conexões baseadas em *login* são feitas a nível de servidor, enquanto conexões por usuário em banco de dados independente são feitas a nível de banco de dados. Nesse sentido, como as regras de *firewall* de servidores podem ser separadas das regras de *firewall* de bancos de dados, permite-se que a cada um dos tipos conexão sejam aplicadas regras diferentes, de acordo com o nível em questão. Além disso, no modelo tradicional as funções e permissões em servidores podem limitar o acesso a todos os bancos de dados gerenciados pelo servidor. Quando se usa o modelo de usuário em banco de dados independente, os proprietários do banco de dados e os usuários que possuem a permissão *ALTER ANY USER* podem conceder acesso ao banco de dados. Dessa forma, reduz-se o controle de acesso de *logins* de servidor com altos privilégios e aumenta o controle de acesso de usuários contidos no banco de dados que possuem altos privilégios (MICROSOFT, 2019h).

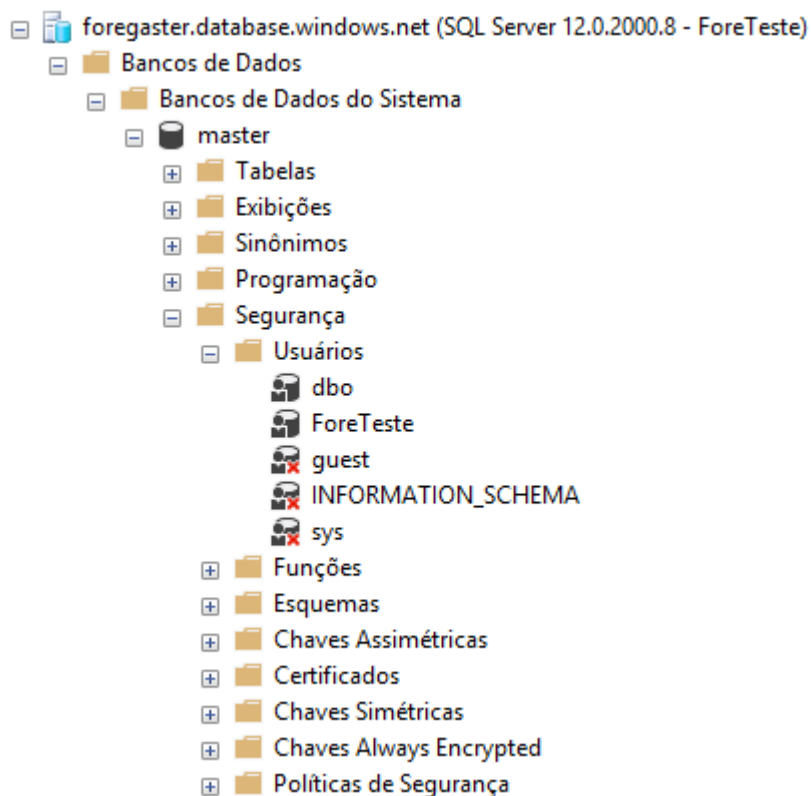
Após o exposto nas seções anteriores, é de fácil compreensão que a conta de administrador definida durante a criação do servidor de banco de dados opera no modelo baseado em *login* e usuário. Entretanto, por ser uma conta administrativa e única, ela apresenta algumas características, sendo as principais:

- É a única conta que pode automaticamente se conectar a qualquer banco de dados do servidor. Isto é, as outras contas (ou *logins*) precisam ser proprietárias do banco de dados ou possuir um usuário no banco de dados em questão vinculado ao seu *login*;
- Como opera no modelo baseado em *login* e usuário, a conta de administrador, assim como o proprietário, acessa o banco de dados utilizando o usuário *DBO*, possuindo todas as permissões;
- Não entra no banco de dados mestre como usuário *DBO*, possuindo permissões limitadas nesse banco de dados;
- Podem criar, alterar e excluir bancos de dados, *logins*, usuários no banco de dados mestre e regras de *firewall* a nível de servidor.

É possível, ainda, definir funções com permissões desejadas e atribuir usuários a essas funções. Essas funções podem ser a nível de servidor, onde normalmente são funções administrativas e exigem a existência de um *login*, ou podem ser a nível de banco de dados, que podem ser atribuídas a qualquer usuário contido em um banco de dados.

Dada a natureza deste trabalho, a única conta necessária à sua produção foi a conta de administrador. Desse modo, não foram criados outros *logins* ou usuários, assim como não houve atribuição de funções ou alteração de permissões. Todavia, caso necessário, isso pode ser feito por meio do SSMS ou utilizando Transact-SQL. A Figura 12 ilustra, no ambiente SSMS, como se dá a organização do banco de dados mestre no servidor e a conta de administrador criada e utilizada no presente trabalho, ForeTeste.

Figura 12 – *Login* de administrador criado no banco de dados mestre, no ambiente SSMS



Fonte: Produção do próprio autor.

4.5 Criando um Banco de Dados SQL do Azure

Há duas opções a serem consideradas na criação de um banco de dados individual: criá-lo na camada de computação provisionada ou na camada de computação sem servidor. Na opção provisionada, há uma quantidade pré-alocada de recursos computacionais, incluindo a CPU e memória, sendo permitido escolher entre dois modelos de compra possíveis: o baseado em vCore (*Virtual Core* - recomendado pela Microsoft) ou o baseado em DTU (*Database Transaction Unit*). Na opção sem servidor, os recursos computacionais são automaticamente dimensionados em um intervalo configurável. Entretanto, esse modo só está disponível no

modelo de compra vCore (MICROSOFT, 2019m). Nesse sentido, é importante conhecer os conceitos, as características e vantagens de cada um dos modelos de compra, de modo a fornecer condições para escolher a opção mais adequada ao fluxo de trabalho a ser realizado.

4.5.1 Modelo de compra baseado em vCore

Segundo Microsoft (2019g), o modelo de compra vCore representa uma CPU lógica e possibilita a escolha entre gerações de *hardware* e as características físicas desse *hardware*, tais como o número de núcleos, a memória e a capacidade de armazenamento. Este modelo fornece maior flexibilidade, controle e transparência de consumo individual de recursos. Dessa forma, torna-se possível ao usuário escolher o nível de computação, a memória e os recursos de armazenamento que melhor se encaixem às necessidades do fluxo de trabalho. Por fim, para bancos de dados individuais é possível escolher dentre três camadas de serviço: o de uso geral, o essencial para os negócios e a camada de serviço de hiperescala.

4.5.2 Modelo de compra baseado em DTU

Já o modelo baseado em DTU é uma medida que resulta da combinação de CPU, memória, leituras e escritas. Esse modelo de compra fornece um conjunto de pacotes de recursos computacionais pré-configurados, além de incluir armazenamento para impulsionar diferentes níveis de desempenho do trabalho. Nesse sentido, entende-se que este é o modelo mais adequado para clientes que desejam simplicidade de configuração e pagamentos mensais fixos. Assim como no modelo vCore, é possível escolher entre três camadas de serviço: *Basic*, *Standard* e *Premium* (MICROSOFT, 2019g).

Assim como para os servidores de bancos de dados, há diversas formas de se criar um banco de dados SQL do Azure. Algumas opções são o portal do Azure, o PowerShell, o Azure CLI e até mesmo o SSMS. Neste trabalho, optou-se pela utilização do portal do Azure, devido à simplicidade de uso e para realizar mais facilmente as configurações de pagamento adequadas.

No momento da criação, é necessário informar a assinatura e o grupo de recursos ao qual o banco de dados pertencerá. A assinatura escolhida é a única disponível e o grupo de recursos definido foi o Foregaster, criado anteriormente. Dessa forma, o próximo passo é preencher a seção Detalhes do Banco de Dados, onde define-se um nome para o banco de dados e o servidor

pelo qual o banco de dados será gerenciado. Neste trabalho, o nome foi definido como ValeDB e o servidor escolhido foi o foregaster, criado neste capítulo.

É importante lembrar que embora a localização do recurso (neste caso, o banco de dados) possa ser diferente da localização do grupo de recursos, o banco de dados será localizado na mesma região do servidor ao qual ele pertencerá. Portanto, neste caso a localização é definida como Sul do Brasil automaticamente no ato da seleção do servidor.

O próximo passo é definir se haverá utilização de *pool* elástico SQL para o banco de dados a ser criado. *Pools* elásticos foram criados para promover eficiência de custo e de desempenho no gerenciamento de um grupo de bancos de dados. Isto é, cada consumidor normalmente possui uma forma própria de uso de banco de dados e que pode variar, o que torna difícil prever os recursos que cada usuário de banco de dados irá consumir. Normalmente a solução para isso é ou superdimensionar os recursos fornecidos para o usuário, baseando-se em picos de uso ou nas condições de pagamento, ou subdimensionar esses recursos, de modo a reduzir os custos em detrimento de desempenho e satisfação de usuários durante picos de uso. Nesse sentido, os *pools* elásticos surgem para lidar com esse problema da melhor forma. Isto é, garantem que os bancos de dados recebam os recursos computacionais necessários no momento em que forem solicitados. Em outras palavras, fornecem um mecanismo simples de alocação de recursos dentro de um orçamento previsível. Portanto, *pools* elásticos permitem a otimização dos preços e custos de um grupo de bancos de dados que apresentam demandas de uso variadas e imprevisíveis, enquanto mantém um bom desempenho entregue a cada banco de dados (MICROSOFT, 2019k).

Entretanto, dada a natureza deste trabalho e levando-se em conta a falta de necessidade de utilizar um grupo de bancos de dados, visto que somente um único banco de dados é necessário para desenvolver o protótipo proposto, não foi contratado um *pool* elástico para o banco de dados a ser criado.

4.5.3 Camadas de serviço e preço

A última informação básica necessária é a definição sobre o modelo de compra e a camada de serviço para o banco de dados, conforme discutido anteriormente. Cada modelo de compra apresenta custos mensais diferentes, de acordo com a camada de serviço escolhida e de acordo

com os recursos computacionais configurados. Dessa forma, o modelo de compra vCore possui uma faixa de valores para os custos mensais estimados, que vão de aproximadamente R\$ 2.150 até valores muito elevados como R\$ 220.000. Esse custo é estimado de acordo com o quanto alguns recursos computacionais são alocados, como o número de núcleos e a capacidade máxima de armazenamento de dados, e de acordo com a camada de serviço escolhida (uso geral, hiperescala e essencial para os negócios).

Quanto ao modelo de compra baseado em DTU, a faixa de valores para os custos estimados é consideravelmente mais acessível, sendo possível pagar de R\$ 22,43 até aproximadamente R\$ 72.000. Esses valores variam de acordo com o número de DTU's utilizados, com a capacidade máxima de armazenamento de dados e com a camada de serviço utilizada (*basic, standard e premium*). Como neste trabalho não há a necessidade de recursos computacionais avançados e tendo em vista o custo total, o modelo de compra adotado foi o baseado em DTU.

Nesse modelo de compra escolhido, há três camadas diferentes dentre as quais deve-se optar por uma, como já mencionado anteriormente. Todas elas têm a mesma finalidade: desenvolvimento e produção. Da mesma forma, possuem um *uptime* (tempo de atividade, o quanto o banco de dados estará disponível ao longo do tempo) equivalente, sendo de 99,99%. Os breves períodos em que o serviço estiver indisponível, devido a manutenção, *patches* ou problemas da plataforma, podem ser contornados por meio de lógicas de repetição. Embora as camadas de serviço possuam algumas características semelhantes, ressalta-se que cada uma delas possuem custos diferentes, tendo características distintas e limites diferentes quanto à quantidade de recursos computacionais configurados. Dessa forma, pode-se sintetizá-las de acordo com o Quadro 1 (MICROSOFT, 2019o).

Quadro 1 – Características de cada camada de serviço

Característica	<i>Basic</i>	<i>Standard</i>	<i>Premium</i>
Período máximo de retenção de backup	7 dias	35 dias	35 dias
Processamento	Baixo	Baixo, médio, alto	Médio, alto
Taxa de transferência aproximada de E/S	1-5 IOPS por DTU	1-5 IOPS por DTU	25 IOPS por DTU
Tempo de latência de E/S	5 ms (leitura), 10 ms (escrita)	5 ms (leitura), 10 ms (escrita)	2 ms (leitura/escrita)
Indexação <i>Columnstore</i>	Não suportado	Instâncias S3 e acima	Suportado
OLTP em memória	Não suportado	Não suportado	Suportado
Tamanho máximo de armazenamento	2 GB	1 TB	4 TB
Número máximo de DTUs	5	3000	4000

Fonte: Microsoft (2019o).

Nota: Modificado pelo autor.

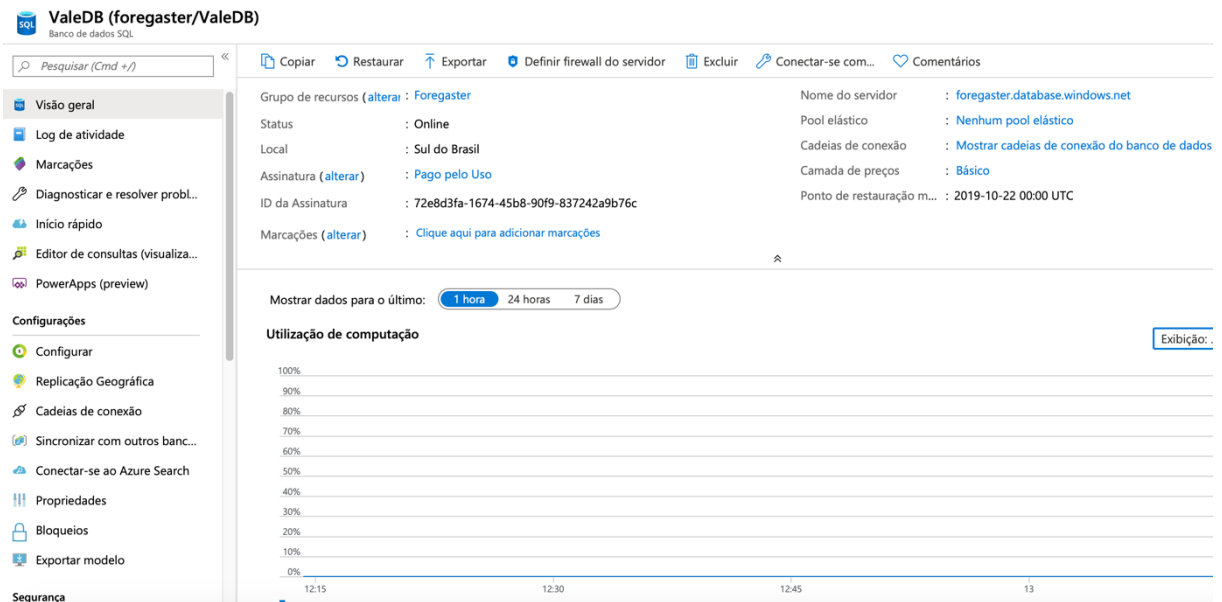
Como pode-se ver, as camadas *Standard* e *Premium* são adequadas para clientes que necessitam de maior poder de processamento, maior capacidade de armazenamento, maior segurança e outros recursos avançados. Tendo em vista que este trabalho exige poucos recursos para sua confecção somado à busca por um baixo custo, a camada de serviço escolhida para a criação do banco de dados foi a *Basic*.

Além das configurações básicas mencionadas anteriormente, necessárias à criação do banco de dados SQL do Azure, há ainda outras configurações opcionais que podem ser feitas, como pontos de extremidade privados, fonte de dados para iniciar o banco de dados (restaurar *backup* ou aplicar um modelo), ordenação do banco de dados, segurança de dados avançada e rótulos. Entretanto, optou-se por não realizar essas configurações no ato da criação, dado que algumas delas são feitas em um momento posterior, ao longo das seções seguintes.

Por meio da Figura 13, pode-se ver o painel administrativo fornecido pelo portal do Azure, que proporciona ferramentas para gerenciar o banco de dados ValeDB criado. Nele, é possível identificar alguns dados do banco de dados, como seu nome, o nome do servidor, a localização, o tipo de assinatura e o grupo de recursos, a camada de serviço, bem como outros recursos de segurança e de gerenciamento disponíveis para uso. Além disso, também é possível visualizar

um gráfico que informa a utilização do banco de dados ao longo do tempo, de acordo com a escala especificada.

Figura 13 – Painel de gerenciamento do banco de dados ValeDB



Fonte: Produção do próprio autor.

4.6 Segurança e Configurações

Uma vez criados o servidor e o banco de dados, torna-se primordial discutir questões de segurança e analisar os recursos disponíveis pela plataforma que proporcionam a proteção dos dados e do sistema como um todo. A Figura 14 ilustra uma estratégia de segurança fornecida pela plataforma Azure que segue uma abordagem de defesa em profundidade composta por camadas de segurança, nas quais a movimentação se dá de fora para dentro.

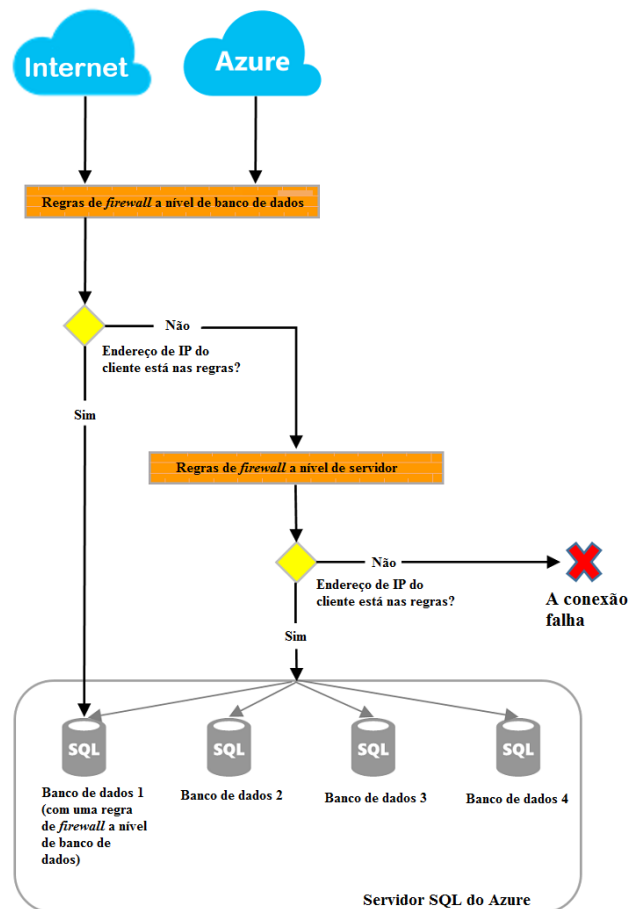
Figura 14 – Camadas de segurança da plataforma Azure para proteção dos dados



Fonte: Microsoft (2019a).
Nota: Modificado pelo autor.

4.6.1 Segurança de rede

A primeira camada de segurança, a Segurança de rede (*Network Security*), é basicamente composta por regras de *firewall*. Essas regras impedem o acesso de rede ao servidor de banco de dados a menos que o endereço IP de acesso seja explicitamente incluído nas regras. Em outras palavras, tentativas de conexão provenientes da internet e do Azure precisam passar pelo *firewall* antes de chegar ao servidor ou ao banco de dados. A Figura 15 ilustra a lógica de funcionamento do *firewall*, bem como os níveis de regras de acesso e a ordem em que essas regras são consultadas.

Figura 15 – Fluxograma de funcionamento do *firewall*

Fonte: Microsoft (2019d).
Nota: Modificado pelo autor.

Pela Figura 15, pode-se entender que existem dois níveis de regras de *firewall*: regras a nível de servidor e a nível de banco de dados.

4.6.1.1 Regras de *firewall* para IP a nível de servidor

Essas regras concedem permissão aos usuários para acessar todo o servidor SQL do Azure. Isso significa que o acesso é garantido a todos os bancos de dados gerenciados por esse servidor. Portanto, para permitir o acesso dos usuários, basta adicionar como regra os endereços IP correspondentes, ou um intervalo de endereços que inclua os usuários em questão. Dessa forma, as regras são armazenadas no banco de dados mestre e, no contexto Azure, há um limite de 128 regras para cada servidor (MICROSOFT, 2019d).

É possível definir as regras a nível de servidor de diversas formas, utilizando o portal do Azure, o Azure PowerShell ou *Transact-SQL*. Para que seja feito por meio do portal ou pelo

PowerShell, é necessário ter acesso à conta proprietária da assinatura ou a alguma conta colaboradora dessa assinatura. Para definir as regras por *Transact-SQL*, exige-se que seja feita uma conexão ao banco de dados por meio de um *login* a nível de servidor com certas permissões. Neste caso, uma regra de *firewall* deve ser previamente criada por algum usuário que tenha permissões a nível de Azure, para que seja possível o acesso ao banco de dados (MICROSOFT, 2019d).

4.6.1.2 Regras de *firewall* para IP a nível de banco de dados

Esse tipo de regra permite ao usuário acessar determinados bancos de dados em um servidor de banco de dados. As regras são criadas para cada banco de dados, incluindo o banco de dados mestre, e são armazenadas em cada um deles. Assim como no caso anterior, basta adicionar como regra o endereço IP de cada cliente que terá acesso ao banco de dados ou um intervalo de endereços que inclua esses clientes, havendo um limite de 128 regras de *firewall* para esse banco de dados (MICROSOFT, 2019d).

Vale ressaltar, entretanto, que as regras de *firewall* a nível de banco de dados somente podem ser criadas por meio de *Transact-SQL*, exigindo para tanto uma prévia configuração do *firewall* a nível de servidor.

Segundo Microsoft (2019d), o recomendado é a utilização de regras de *firewall* a nível de banco de dados sempre que possível, uma vez que essa prática aumenta a segurança e torna o banco de dados mais portátil. Nesse sentido, o ideal é que as regras de *firewall* a nível de servidor sejam usadas apenas para contas administrativas ou quando há um grande número de bancos de dados no servidor que possuem os mesmos requisitos de acesso, evitando a necessidade de configurar cada um dos bancos de dados individualmente.

Levando em consideração o exposto anteriormente e retomando a Figura 15 como base, pode-se resumir o funcionamento do *firewall* da seguinte forma: quando um computador tenta se conectar por meio da internet ao servidor de banco de dados, o *firewall* primeiramente verifica o endereço de IP que fez a requisição de acesso e confere com as regras a nível de banco de dados para o banco de dados que está sendo acessado. Com isso, há três consequências (MICROSOFT, 2019d):

- Se o endereço de IP está dentro do intervalo especificado pelas regras de *firewall* a nível de banco de dados, então a conexão ao banco de dados em questão é permitida.
- Se o endereço de IP não está dentro do intervalo especificado pelas regras de *firewall* a nível de banco de dados, então o *firewall* verifica as regras a nível de servidor. Se o endereço de IP estiver dentro do intervalo especificado por essas novas regras, então a conexão é permitida, uma vez que as regras de *firewall* a nível de servidor se aplicam a todos os bancos de dados SQL de um mesmo servidor SQL.
- Se o endereço de IP não estiver dentro de nenhum intervalo especificado tanto pelas regras a nível de banco de dados como pelas regras a nível de servidor, então a conexão é negada.

Já para conexões provenientes do Azure, é necessário permiti-las explicitamente. Isto é, para que aplicativos hospedados no Azure possam conectar ao servidor de banco de dados, é necessário especificar o endereço de IP *0.0.0.0* no *firewall*, que indica que as conexões provenientes do Azure são permitidas.

É válido lembrar que o *firewall* apenas dá a oportunidade ao cliente de tentar se conectar ao servidor de banco de dados. Isto é, ainda é necessário que o cliente forneça as credenciais de segurança (*login* e senha) e que ele tenha permissões nesse servidor. Caso contrário, a conexão ao servidor será negada.

Dada a natureza deste trabalho e como não houve a necessidade de criar contas de usuário além da conta administrativa ForeTeste já mencionada, as únicas configurações de *firewall* feitas foram a nível de servidor. Isto é, foram criadas regras de *firewall* para os endereços de IP para os computadores utilizados para o desenvolvimento do protótipo proposto, utilizando para tanto o portal do Azure. A Figura 16 demonstra o painel administrativo do portal, que permite gerenciar as regras de IP a nível de servidor.

Figura 16 – Painel de gerenciamento das regras de *firewall* a nível de servidor

Configurações de firewall
foregaster (SQL Server)

Salvar Descartar + Adicionar IP de cliente

As conexões dos IPs especificados abaixo fornecem acesso a todos os bancos de dados em foregaster.

Permitir que serviços e recursos do Azure acessem este servidor

ATIVAR DESLIGAR

Endereço IP do cliente 179.105.94.119

Nome da regra	IP Inicial	IP Final	
<input type="text"/>	<input type="text"/>	<input type="text"/>	...
Bruno	179.105.94.119	179.105.94.119	...
Klaus	181.213.90.172	181.213.90.172	...
RICARDO-DHCP	177.40.212.121	177.40.212.121	...
UFES	200.137.65.106	200.137.65.106	...

As conexões da VNET/Sub-rede especificadas abaixo fornecem acesso a todos os bancos de dados em foregaster.

Redes virtuais + Adicionar rede virtual existente + Criar nova rede virtual

Nome da regra	Rede virtual	Sub-rede	Intervalo de ...	Status do po...	Grupo de rec...
Não há regras de vnet para este servidor.					

Fonte: Produção do próprio autor.

É permitido, ainda, a utilização de regras de *firewall* para redes virtuais. Isto é, por meio dessas regras torna-se possível que os bancos de dados SQL do Azure só aceitem conexões provenientes de subredes de uma rede virtual especificada. Entretanto, essa opção não foi explorada neste trabalho.

4.6.2 Gerenciamento de acesso

Ao passar pelo *firewall* descrito na seção anterior, ainda há a camada de gerenciamento de acesso, que é definida principalmente pelo processo de autenticação. Autenticação é a forma de provar que o usuário é quem ele diz ser. A plataforma Azure dá suporte a dois tipos de autenticação, já mencionados ao longo deste capítulo: Autenticação SQL e Autenticação do Azure *Active Directory*. Segundo Microsoft (2019a), esses tipos são definidos da seguinte forma:

- **Autenticação SQL:** é a autenticação de um usuário quando ele se conecta a um banco de dados SQL do Azure utilizando um nome de usuário e senha. A conta administrativa criada anteriormente possui um nome de usuário e senha, que são usados para autenticar

a conexão a qualquer banco de dados do servidor como o proprietário desses bancos. Outros *logins* e usuários podem ser criados por essa conta administrativa, que podem se conectar ao servidor fornecendo o nome de usuário e senha.

- *Autenticação do Azure Active Directory*: é um mecanismo de conexão com um banco de dados SQL do Azure que utiliza identidades do *Azure Active Directory* (Azure AD). Essa forma de autenticação permite aos administradores gerenciar as identidades e permissões de usuários de banco de dados de forma centralizada, em um único local. Isso minimiza o armazenamento de senhas e permite políticas centralizadas de rotação de senhas. Para usar esse tipo de autenticação, é necessário criar uma conta administrativa chamada administrador do *Active Directory*. Entretanto, neste trabalho a autenticação utilizada foi definida como a Autenticação SQL. Nesse sentido, não houve criação dessa conta administrativa e de nenhuma outra identidade do Azure AD.

Outro ponto importante na camada de gerenciamento de acesso são as autorizações. Autorizações se referem às permissões dadas a um usuário dentro de um banco de dados SQL do Azure, que determinam o que o usuário pode fazer. Essas permissões são gerenciadas por meio de funções de bancos de dados. Isto é, definem-se as permissões a nível de banco de dados que as funções necessitam ter, e em seguida pode-se atribuir uma dessas funções ao usuário em questão. Há, ainda, a possibilidade de conceder ao usuário algumas permissões diretamente, sem a necessidade de funções (MICROSOFT, 2019a).

Entretanto, segundo Microsoft (2019a), a prática recomendada é utilizar as funções disponíveis sempre que possível e, caso seja necessário, criar funções personalizadas ao invés de atribuir permissões diretamente aos usuários, uma vez que com funções o gerenciamento de acesso é mais eficiente. Além disso, é importante saber que a conta de administrador é atribuída à função nativa *db_owner*, que possui uma grande quantidade de permissões e deve ser associada somente a usuários que possuam responsabilidades administrativas.

Existem inúmeras permissões que podem ser concedidas ou negadas aos usuários nos bancos de dados SQL, tais como *CREATE USER*, *ALTER ANY USER*, *ALTER ROLE*, *INSERT*, *DELETE* e *EXECUTE*, que podem ser usadas para criar usuários, alterar suas permissões, modificar funções, inserir ou apagar dados, executar scripts externos, etc.

Há, ainda, uma outra forma de proteção na camada de gerenciamento de acesso: a segurança a nível de linha (RLS, do inglês *Row-Level Security*). Essa ferramenta proporciona o controle de acesso às linhas de uma tabela de um banco de dados, de acordo com as características do usuário que está executando uma consulta. Nesse sentido, é possível restringir o acesso dos funcionários somente às linhas da tabela cujos dados são pertinentes ao departamento ao qual eles fazem parte. Outro exemplo é a restrição de acesso dos clientes aos dados que são de relevância para as empresas às quais eles pertencem. A Figura 17 resume esse processo de maneira gráfica (MICROSOFT, 2019n).

Figura 17 – Controle de acesso às linhas de uma tabela por meio de RLS



Fonte: Microsoft (2019a).
Nota: Modificado pelo autor.

4.6.3 Proteção contra ameaças

A penúltima camada de proteção é composta basicamente por auditorias e detecção de ameaças. A ferramenta de auditoria para banco de dados SQL do Azure permite o rastreamento de atividades e dos eventos que ocorrem no banco de dados, armazenando essas informações em um *log* de auditoria na conta de armazenamento do Azure, no ambiente de trabalho *Log Analytics* ou no *Hub* de Eventos. A auditoria facilita manter conformidade regulatória, ajuda a entender a atividade do banco de dados, permite ter uma visão geral sobre discrepâncias e anomalias que poderiam indicar preocupações para o negócio ou supostas violações de segurança. Além disso, facilita a aderência a padrões de conformidade, embora não seja garantida (MICROSOFT, 2019l).

Embora não haja necessidade de implementação desse esquema de proteção neste trabalho, é importante levar em conta que em empresas com grande atividade nos bancos de dados e com considerável número de funcionários, o uso dessa ferramenta pode ser essencial.

A outra ferramenta desta camada de segurança é a Proteção Avançada contra Ameaças, que analisa os *logs* do servidor SQL no intuito de detectar comportamentos incomuns e tentativas potencialmente prejudiciais de acesso ou de exploração de brechas nos bancos de dados. Dessa forma, são criados alertas para atividades suspeitas, como injeção de SQL, uma potencial infiltração de dados, ataques de força bruta ou anomalias nos padrões de acesso de modo a se aproveitar de progressão de privilégios ou do uso de credenciais vulneráveis. Esses alertas podem ser visualizados no painel do Centro de Segurança do Azure, onde são fornecidos detalhes sobre as atividades suspeitas, juntamente com recomendações de investigação e de ações para lidar com a ameaça (MICROSOFT, 2019c).

Essa proteção pode ser adquirida por meio da ativação da Segurança de Dados Avançada, no portal do Azure, que inclui as ferramentas de Descoberta e Classificação de Dados, Avaliação de Vulnerabilidades e Proteção Avançada contra Ameaças. Para utilizá-las, é necessário pagar uma taxa, sendo possível aplicar essa proteção em todos os bancos de dados do servidor pagando taxas adicionais. A Figura 18 demonstra de maneira gráfica uma situação de uso da ferramenta, ressaltando a sua grande importância em grande parte das empresas que dependem de bancos de dados e armazenam informações importantes e/ou sigilosas.

Figura 18 – Proteção do banco de dados por meio da ferramenta Proteção Avançada de Dados



Fonte: Microsoft (2019a).
Nota: Modificado pelo autor.

4.6.4 Proteção e criptografia da informação

A última camada do esquema de proteção, prevista na Figura 14, é focada na proteção contra acessos não autorizados aos dados, estejam eles em movimento ou não. Ou seja, a camada é composta por um protocolo de segurança, tecnologias de criptografia e outras ferramentas que visam proteger a informação.

4.6.4.1 Segurança da camada de transporte

O banco de dados SQL do Azure protege a informação manipulada por meio da criptografia dos dados em movimento, utilizando o protocolo de Segurança da Camada de Transporte (TLS, do inglês *Transport Layer Security*).

O protocolo TLS é um protocolo de segurança do tipo cliente-servidor, que atua na camada de transporte da arquitetura TCP/IP. Ele é o sucessor do protocolo *Secure Sockets Layer* (SSL) e mantém muitas semelhanças com o SSL, porém trouxe alterações, melhorias e recursos adicionais. Dessa forma, é possível mencionar alguns serviços básicos de segurança que são aplicados à comunicação: serviços de autenticação (tanto da origem dos dados quanto das entidades que se comunicam), serviços de confidencialidade de conexão, serviços de integridade de conexão.

Oppliger (2009) destaca, dentro do funcionamento do protocolo SSL, o estabelecimento da conexão entre os pares comunicantes. Para que isso aconteça, entra em ação um processo que é considerado o coração do protocolo: o processo de *handshake* (aperto de mão). Ele permite que o cliente e o servidor autentiquem um ao outro, que negociem *cipher suites* (conjunto de algoritmos que ajudam a manter segura uma conexão de rede) e métodos de compressão, que definam a chave secreta que será utilizada na criptografia simétrica, e outros itens necessários. Nesse sentido, Oppliger (2009) descreve o processo de *handshake* da seguinte forma:

1. O cliente envia uma mensagem de *Client Hello* (que indica a requisição de conexão) para o servidor;
2. O servidor responde ao cliente com uma mensagem de *Server Hello* (que indica que a requisição foi recebida);

3. Em seguida, o servidor envia uma mensagem contendo o certificado digital que autentica a si mesmo. Essa mensagem também contém a chave pública de criptografia do servidor;
4. Em algumas ocasiões, o servidor pode enviar uma mensagem de *Server Key Exchange* para o cliente (quando o certificado fornecido pelo servidor não for suficiente para o cliente);
5. Se o servidor necessitar da autenticação do cliente, ele enviará uma mensagem de requisição de certificado para o cliente;
6. Nesse momento, o servidor envia uma mensagem de *Server Hello Done*, que indica o fim da etapa inicial do processo.

Nessa troca de mensagens de *Client Hello* e *Server Hello*, o cliente e o servidor negociam a versão do protocolo que será utilizada na comunicação, um identificador de sessão, um *cipher suite* e um método de compressão. De acordo com Oppliger (2009), ao fim do *Server Hello*, o cliente passa a responder de acordo com o que foi requisitado, isto é:

1. Se o servidor requisitou a autenticação do cliente, então o cliente envia uma mensagem para o servidor contendo o certificado digital que autentica a si mesmo;
2. Nesta etapa, o cliente envia uma mensagem de *Client Key Exchange* ao servidor. O conteúdo dessa mensagem dependerá do método de troca de chaves utilizado, que pode ser o método RSA ou Diffie-Hellman. Entretanto, independente do método utilizado, essa mensagem permite que o cliente e o servidor definam a chave secreta que será utilizada para a criptografia da conexão;
3. Depois de concluir o processo de autenticação e de geração da chave secreta com sucesso, tanto o cliente como o servidor enviam mensagens de término do *handshake*, indicando que o resto da comunicação a partir daquele momento será criptografado com a chave definida. Esse tipo de criptografia onde os dois entes comunicantes utilizam a mesma chave para criptografar suas mensagens é conhecido como criptografia simétrica.

É importante, entretanto, entender um pouco como ocorre a definição da chave secreta. De maneira geral, a chave secreta é definida pelo cliente e enviada para o servidor, na etapa de *Client Key Exchange*, mencionada anteriormente. Para que essa transmissão seja feita com segurança, sem que terceiros consigam interceptar essa informação, o cliente criptografa a

mensagem que contém essa chave secreta utilizando a chave pública do servidor, recebida em etapa anterior no processo de *handshake*. Todavia, para que seja possível descriptografar a mensagem, é necessário utilizar uma chave privada que somente o servidor conhece. Isso significa que nenhuma entidade além do servidor consegue acessar o conteúdo e descobrir a chave secreta. Essa forma de criptografia é conhecida como criptografia assimétrica (OPPLIGER, 2009).

Ao conhecer o funcionamento básico do protocolo TLS, é possível compreender a sua importância na utilização da comunicação com os bancos de dados. Segundo Microsoft (2019a), o servidor SQL exige que o protocolo TLS seja utilizado a todo o momento em todas as conexões. Isso significa que todos os dados são criptografados quando em movimento entre o cliente e o servidor, independente da configuração de criptografia ou de confiança no certificado do servidor, especificadas no ato da conexão.

Entretanto, como boa prática, recomenda-se que no ato da conexão sejam especificadas a conexão como criptografada e a não confiança no certificado do servidor. Dessa maneira, a aplicação será forçada a validar o certificado do servidor e evita que ela esteja vulnerável a interceptações (MICROSOFT, 2019a).

4.6.4.2 *Transparent Data Encryption*

Transparent Data Encryption (TDE) é uma outra ferramenta utilizada pela plataforma Azure para a proteção dos dados. Ela adiciona uma camada de segurança que ajuda a proteger os dados em repouso (que não estão em movimento em uma conexão, como no caso do protocolo TLS) contra acessos *offline* ou não autorizados a arquivos brutos ou *backups*. Os cenários mais comuns de aplicação são, por exemplo, roubos de *datacenters*, exposição insegura de *hardwares* ou mídias como discos rígidos e fitas de *backup*. O TDE criptografa todo o banco de dados utilizando o algoritmo de criptografia AES (*Advanced Encryption Standard*), que não exige nenhuma alteração por parte dos clientes (MICROSOFT, 2019a).

Na plataforma Azure, todos os bancos de dados SQL criados são criptografados por padrão, e a chave de criptografia é protegida por um certificado de servidor interno. O gerenciamento desse certificado é feito pelo próprio serviço, não exigindo ações dos usuários. Entretanto, é

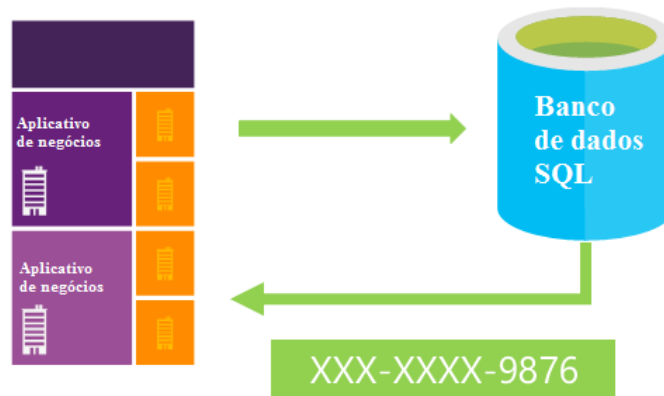
permitido o controle das chaves de criptografia, por meio do *Azure Key Vault*, que é um sistema de gerenciamento de chaves fornecido pela plataforma Azure.

4.6.4.3 Demais proteções

Além do protocolo TLS e do TDE, mencionados anteriormente, ainda há a ferramenta *Always Encrypted*, que é uma ferramenta utilizada para proteger contra o acesso a dados sensíveis armazenados em colunas específicas de um banco de dados, tais como números de cartões de crédito, dados de identificação pessoal ou outros dados sigilosos. Isso inclui administradores de bancos de dados ou outros usuários com altos privilégios que estão autorizados a acessar o banco de dados para realizar suas funções, mas que não possuem necessidades ou competências para acessar algum dado em específico que está nas colunas criptografadas. Dessa forma, faz sentido dizer que essa ferramenta realiza criptografia-em-uso, de maneira que os dados estão sempre criptografados e são descriptografados somente para fins de processamento, pelas aplicações do cliente que possuem acesso à chave de criptografia. Essa chave nunca é exposta ao sistema SQL e pode ser armazenada no *Windows Certificate Store* ou no *Azure Key Vault* (MICROSOFT, 2019a).

Há, ainda, a ferramenta Mascaramento Dinâmico de Dados (*Dynamic data masking*), que limita a exposição de dados sensíveis mascarando-os para usuários com baixos privilégios. Esse recurso atua automaticamente descobrindo dados potencialmente sensíveis no banco de dados SQL do Azure e fornecendo recomendações de ações para mascarar esses campos, causando impacto mínimo na camada de aplicação do cliente. Seu funcionamento se dá basicamente ofuscando os dados sensíveis como o resultado de uma consulta aplicada aos campos especificados do banco de dados, de modo que os dados nesse banco de dados não são alterados. A figura a seguir ilustra o funcionamento básico desse recurso, mascarando um dado sensível (MICROSOFT, 2019a).

Figura 19 – Funcionamento do recurso de mascaramento dinâmico de dados



Fonte: Microsoft (2019a).
Nota: Modificado pelo autor.

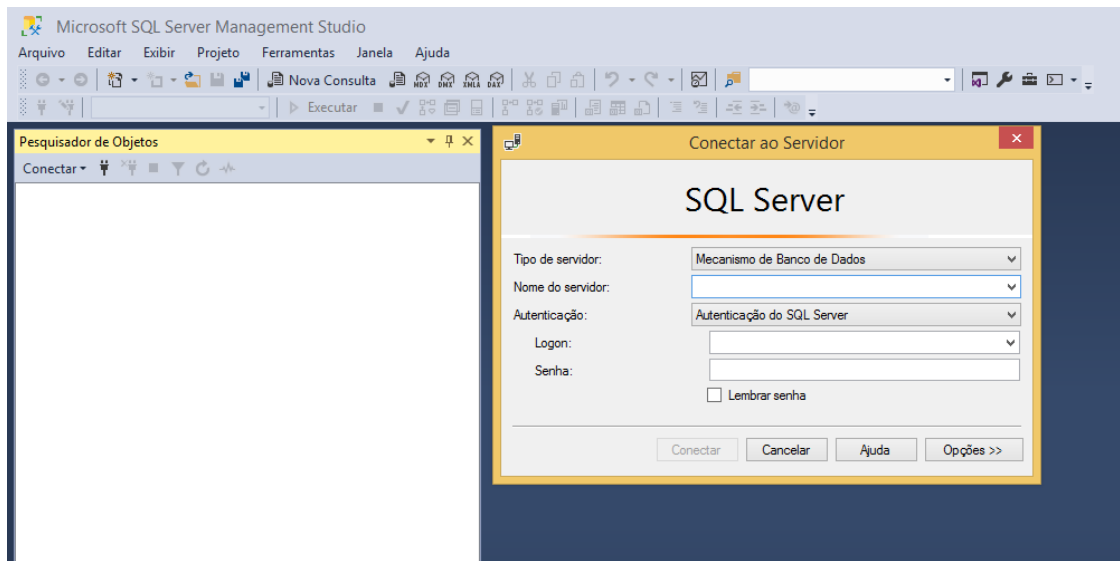
4.7 SQL Server Management Studio

O *SQL Server Management Studio* (SSMS) é um ambiente integrado que pode ser usado para gerenciar qualquer infraestrutura SQL. Nesse sentido, pode-se utilizá-lo para acessar, configurar, gerenciar, administrar e desenvolver todos os componentes do servidor SQL e do bando de dados SQL do Azure. O SSMS disponibiliza um grande grupo de ferramentas gráficas com vários editores de *script* avançados para proporcionar acesso ao servidor SQL aos desenvolvedores e demais usuários (MICROSOFT, 2019p).

4.7.1 Conectando ao banco de dados

Para realizar a conexão com o banco de dados criado ao longo deste capítulo, é necessário coletar algumas informações, como o endereço do servidor, o nome do banco de dados e informações de *login* conforme já mencionado anteriormente. A Figura 20 ilustra a ferramenta SSMS, juntamente com o campo onde são necessárias as informações mencionadas para efetuar a conexão com o banco de dados.

Figura 20 – Ambiente *SQL Server Management Studio*



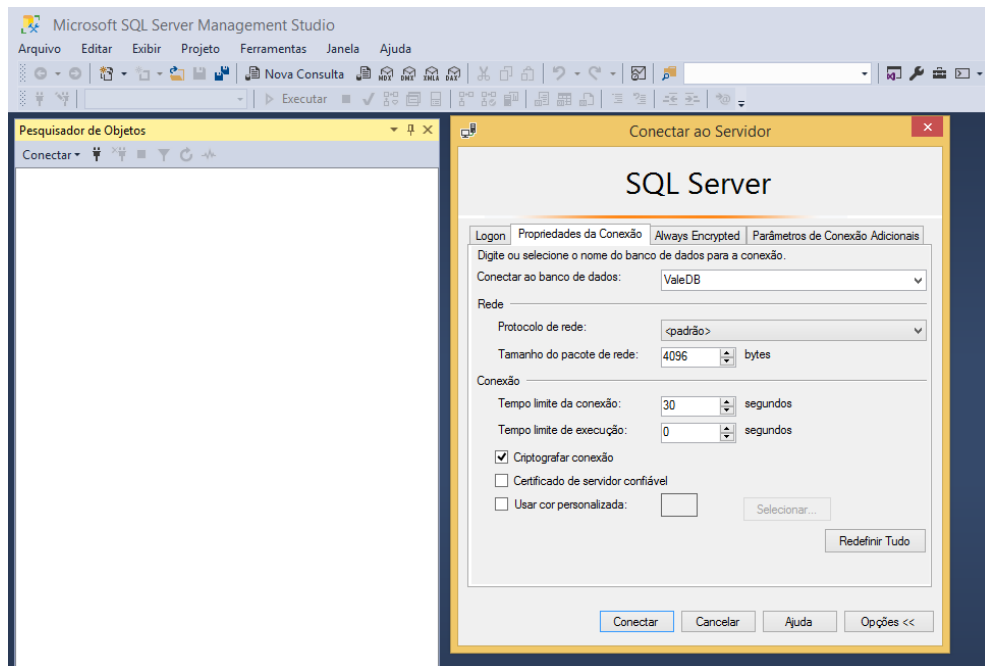
Fonte: Produção do próprio autor.

O próximo passo é preencher todos os campos necessários, seguindo as instruções:

1. No campo Tipo de servidor deve ser especificada a opção Mecanismo de Banco de Dados, uma vez que a conexão será com um servidor SQL e com um banco de dados SQL;
2. O campo Nome do servidor deve ser preenchido com o endereço do servidor mencionado ao longo deste capítulo: *foregaster.database.windows.net*;
3. No campo de autenticação deve ser especificada a opção Autenticação do SQL Server, que é a forma de autenticação utilizada neste trabalho, conforme já mencionado;
4. Os campos de *logon* e senha devem ser preenchidos com os dados de *login* (para contas baseadas em *login*) ou com o nome de usuário e senha (para contas baseadas em usuário em banco de dados), que o usuário possui para acessar o banco de dados. No caso deste trabalho, é utilizada os dados da conta de administrador ForeTeste criada anteriormente.

É possível expandir mais opções que podem ser usadas para determinar outras características da conexão, ao clicar no botão Opções no canto inferior direito da janela de conexão. A Figura 21 ilustra a nova janela aberta, na aba de Propriedades da Conexão.

Figura 21 – Aba Propriedades da Conexão do menu Options do SSMS



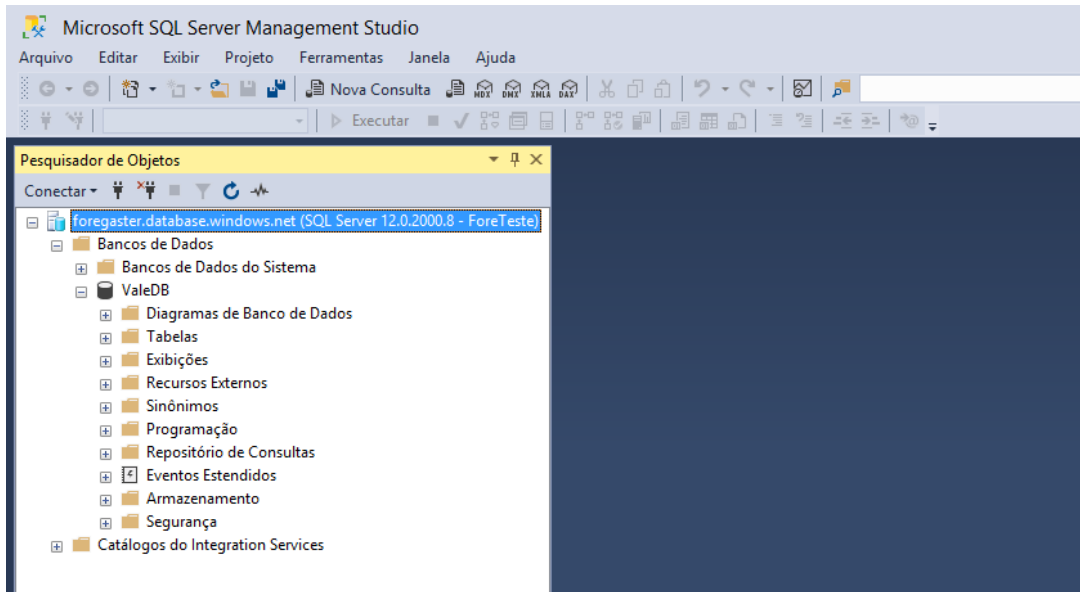
Fonte: Produção do próprio autor.

No campo Conectar ao banco de dados deve ser especificado o banco de dados ao qual o usuário irá se conectar. Neste trabalho, conforme demonstra a figura anterior, a conexão será feita com o banco de dados criado neste capítulo: ValeDB.

É possível, ainda, realizar outras configurações de segurança, como optar por criptografar a conexão, confiar no certificado do servidor, habilitar a ferramenta de segurança *Always Encrypted* e também especificar parâmetros de conexão adicionais (que são realizados por linha de código). Conforme demonstrado na Figura 21, a conexão foi especificada como criptografada e optou-se por não confiar no certificado do servidor, de modo a forçar a validação do certificado. Além disso, como não há dados sensíveis neste trabalho, não foi habilitada a opção *Always Encrypted*.

Ao clicar em conectar, o SSMS realizará a conexão com o banco de dados, resultando na Figura 22.

Figura 22 – Ambiente SSMS conectado com o banco de dados ValeDB



Fonte: Produção do próprio autor.

A partir desse momento, podem ser efetuadas diversas consultas no banco de dados, como visualizar, inserir, modificar e apagar dados e tabelas da maneira que for necessária. Isso pode ser feito tanto por meio da interface gráfica como por meio de código SQL.

5 COMUNICAÇÃO ENTRE ELIPSE E3 E AZURE

Nas seções seguintes é apresentada a plataforma Elipse E3, mostrando as configurações e os recursos essenciais para o correto funcionamento da aplicação. Também é mostrada a construção do protótipo proposto, juntamente com a abordagem da comunicação do E3 com o banco de dados do Azure, principal objeto de interesse deste trabalho.

5.1 Elipse E3

O Elipse E3 é uma plataforma IHM/SCADA utilizada principalmente em centros de controle para efetuar monitoramentos e acionamentos em processos e sistemas presentes em uma planta. O *software* disponível é utilizado para realizar o “[...] monitoramento e controle de processos, oferecendo escalabilidade e constante evolução para diversos tipos de aplicações, desde simples IHM até complexos centros de operação em tempo real” (ELIPSE, [20--?]).

Com a utilização dessa plataforma, é possível aproveitar todos os benefícios que sistemas SCADA proporcionam, conforme elencado ao longo deste trabalho. Segundo ELIPSE ([20--?]), é disponibilizado conexão com a maioria dos equipamentos de mercado, há redução no tempo de desenvolvimento e manutenção, possui integração com sistemas corporativos e de gestão, gera um retorno rápido e duradouro do investimento e possibilita monitoramento e gerenciamento de informações de tempo real.

O Elipse E3 conta com uma interface que permite o desenvolvimento e gerenciamento das aplicações desenvolvidas de forma simplificada, com um suporte nativo aos principais bancos de dados comerciais como o Microsoft SQL *Server*, Access e Oracle, e com alta segurança e rastreabilidade (ELIPSE, [20--?]).

Segundo o próprio desenvolvedor, Elipse ([20--?]), o *software* é dividido em três principais componentes:

- a) O *E3 Studio* que é a ferramenta que possibilita a configuração de todo o sistema podendo ser acessada por múltiplos usuários simultaneamente. Nessa plataforma é possível criar a IHM com o uso de gráficos disponíveis e realizar as configurações de comunicação com o banco de dados, por exemplo.

- b) O *E3 Server*, que “[...] é o servidor de aplicações. A ferramenta é responsável pelo gerenciamento dos principais processos do sistema, além de realizar a redundância e sincronismo de bases de dados” (ELIPSE, [20--?]). É o *E3 Server* que realiza a interação entre o *E3 Studio* e o *E3 Viewer* para a disponibilização das informações ao usuário em qualquer local;
- c) O *E3 Viewer*, que é a ferramenta que possibilita a visualização e operação da aplicação que está no servidor pelo usuário, por meio da interface desenvolvida no *E3 Studio*. Sua execução é realizada via *browser*, permitindo acesso facilitado em qualquer computador

Há, ainda, o *E3 Admin*. Esse módulo é responsável pelo *E3 Server* e outros módulos de interface que se comunicam com os usuários. Com o *E3 Admin*, os usuários podem enviar comandos para o *E3 Server* e também podem controlar o domínio por meio de linha de comando.

5.2 Arquitetura do Elipse E3

De acordo com Elipse Software (2019b), para monitorar um processo específico com um sistema SCADA, normalmente é construída uma aplicação contendo a definição das variáveis envolvidas, com nomes e endereços, telas, definições de alarmes e outras configurações. Essa aplicação se chama Banco de Dados da Aplicação. Dessa forma, quando esse processo exige o uso de dois ou mais computadores, é necessário que cada aplicação em cada computador se comunique com as demais. Isso gera problemas de gerenciamento, uma vez que será necessário aplicar eventuais mudanças a todos os servidores, controlar as versões da aplicação ou até trabalhar com diferentes fabricantes de *softwares* e *hardwares*. Nesse sentido, o E3 resolve esse problema utilizando o conceito de domínio.

Um domínio inclui em um único ambiente a definição dos computadores executando tarefas em tempo real (os servidores) e os bancos de dados do projeto que devem ser executados nesses servidores, além de ser possível executar diversos projetos em cada servidor. É possível, também, que os usuários adicionem, deletem ou modifiquem projetos que estão em execução, sem afetar outras partes do domínio em questão (ELIPSE SOFTWARE, 2019b).

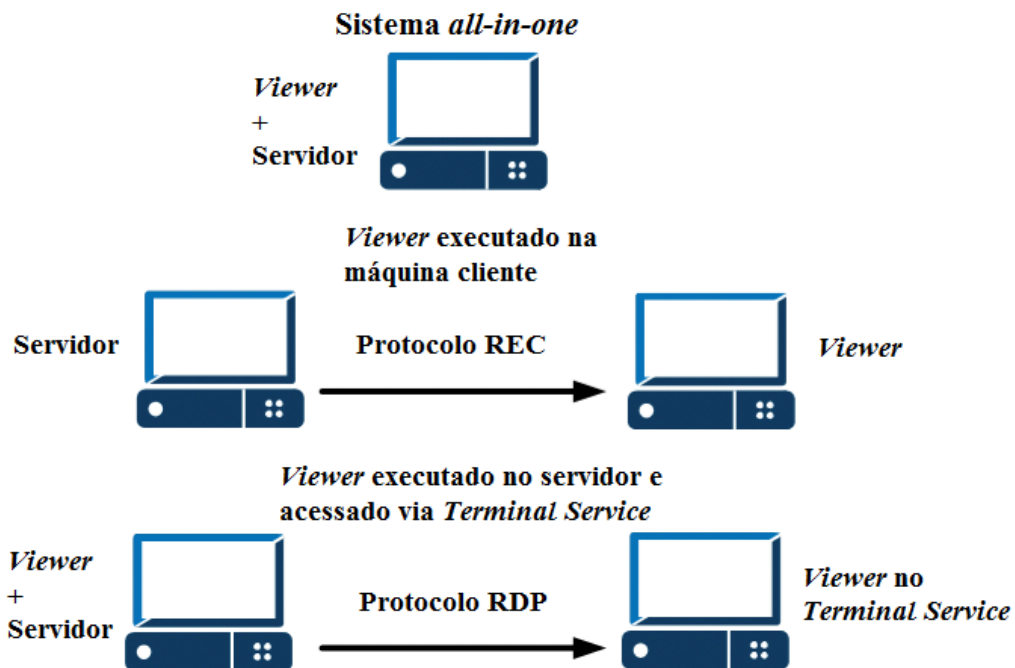
Cada projeto pode conter vários tipos de objeto, tais como telas, *drivers* de entrada e saída, alarmes, históricos, relatórios e bancos de dados. Neste trabalho há a criação de um projeto com o objetivo de monitorar um ambiente simulado, que pode ser associado a um processo real da

indústria. Sua criação, bem como a de seus componentes, é feita ao longo deste capítulo, demonstrando suas formas de configuração e suas representações gráficas no E3.

Retomando o conceito de domínio, ressalta-se que sua estrutura é restrita a servidores e entes similares, como máquinas de servidor, projetos, usuários e senhas. A interface que o cliente utiliza para operar e visualizar o processo é chamada de *Viewer* e ela pode conectar a qualquer E3 *Server* diretamente. Com isso, há alguns pontos interessantes a se observar, a respeito da utilização do *Viewer*: os projetos de aplicação estão contidos exclusivamente no servidor; o navegador *web Internet Explorer* pode ser utilizado como uma interface de operação; a interface do cliente pode alternar de um servidor que se encontra *offline* ou em falha para um que esteja disponível, sem interromper o processo de monitoramento (ELIPSE SOFTWARE, 2019b).

Segundo Elipse Software (2019b), há ainda uma alternativa ao *Viewer* para o cliente, que é a utilização da tecnologia *Terminal Service*. Essa alternativa é um serviço que tem o objetivo de permitir acesso remoto entre computadores e utiliza o protocolo de comunicação RDP (*Remote Desktop Protocol*), que permite a interação remota entre um cliente e um servidor. Já o *Viewer* é executado em uma nova sessão de usuário criada na máquina do servidor, que transfere dados de vídeo à máquina do cliente e recebe dele eventos de mouse e de teclado. A Figura 23 ilustra três casos possíveis de supervisão do processo na arquitetura E3: quando o *Viewer* é executado na mesma máquina do servidor; quando o *Viewer* é executado em uma máquina distinta do servidor utilizando, para tanto, o protocolo REC (*Remote Elipse Call*); quando o *Viewer* é executado no servidor e acessado pelo cliente por meio do *Terminal Service*, que utiliza o protocolo RDP.

Figura 23 – Formas de monitoramento na arquitetura E3



Fonte: Elipse Software (2019b).
 Nota: Modificado pelo autor.

Por fim, é importante saber que, caso o usuário não possua uma licença de utilização do E3, é possível executá-lo no modo demonstração. Dada a natureza deste trabalho, optou-se por utilizar esse modo, que de acordo com Elipse Software (2019b) possui várias limitações, dentre as quais algumas são:

- Permite-se salvar projetos com até 20 *tags* de entrada e saída;
- Não permite trabalhar com domínios remotos;
- Permite comunicar com somente um *driver* de entrada e saída de nível 0. Níveis superiores não são permitidos;
- Apenas a primeira imagem de cada categoria na galeria de símbolos pode ser utilizada;
- Somente permite executar um *Viewer* ou um *WebViewer*;
- Apresenta um limite de execução máximo de um domínio igual a duas horas;
- Permite acesso como um servidor OPC (*Open Platform Communications*);

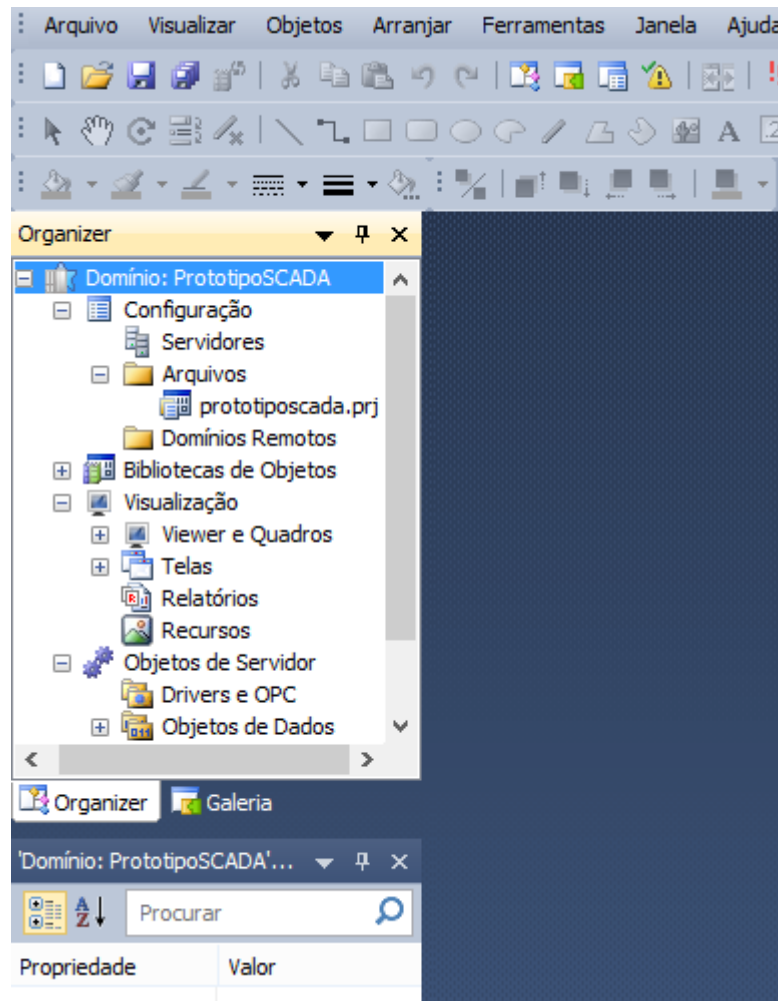
5.3 Configurações Iniciais do Elipse E3

5.3.1 Criação do domínio e do projeto

O primeiro passo para construir o supervisão é criar um projeto no E3 *Studio*, que é o ambiente de desenvolvimento do E3. Para tanto, exige-se a criação de um novo domínio, que deve possuir

um nome, uma configuração de resolução que o *Viewer* utilizará para exibir as telas, um possível *driver* de entrada e saída, configurações de alarme e outras configurações adicionais. Para este trabalho, optou-se pelo nome do projeto e do domínio como “PrototipoSCADA”, com resolução automática de telas, sem configurações iniciais de *drivers* ou alarmes. A Figura 24 demonstra parte da tela inicial do *software*, após a criação do domínio e do projeto.

Figura 24 – Tela inicial do E3 *Studio*, após criação do domínio “PrototipoSCADA”



Fonte: Produção do próprio autor.

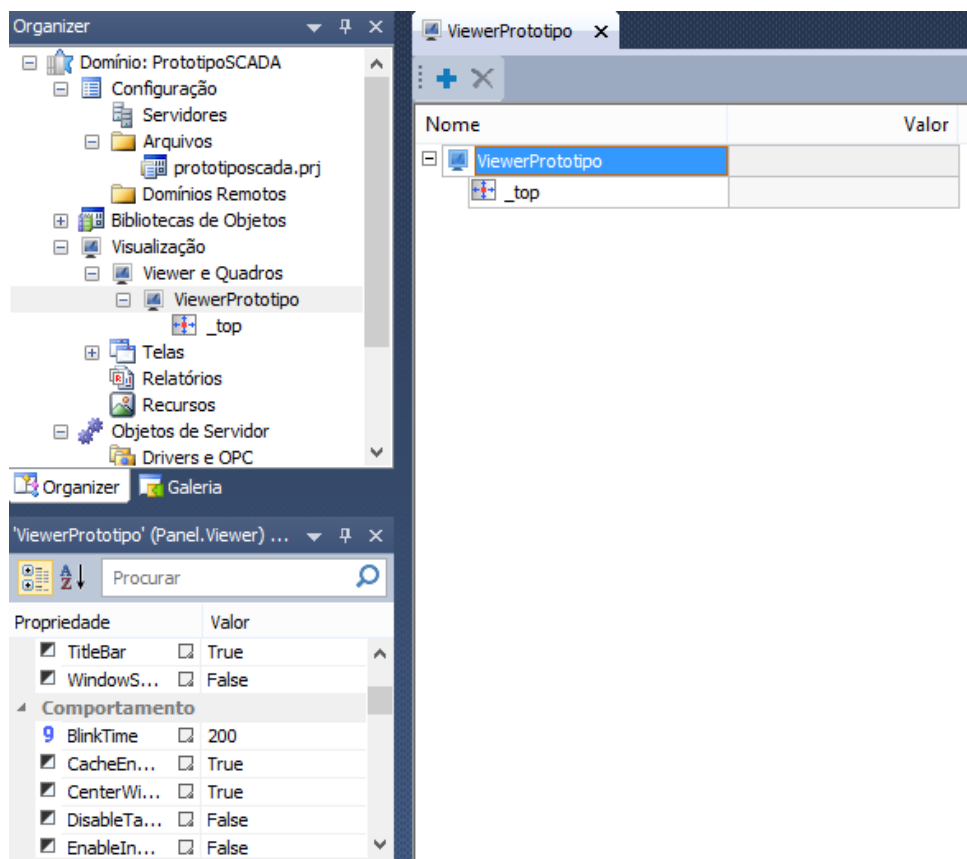
5.3.2 Criação do *Viewer*

A próxima etapa é criar um *Viewer*, que é o objeto que permite a visualização da aplicação. Para que os usuários possam criar telas, visualizá-las em tempo de execução e manipulá-las, é necessária a existência do *Viewer*, uma vez que ele fornece os recursos necessários para isso.

Em outras palavras, o *Viewer* é um *container* para as telas. Nesse sentido, entende-se que esse objeto fornece o contexto necessário para a operação das telas, definindo diversas propriedades que serão aplicadas a elas, como *zoom*, a definição da tela inicial, permitir uso de *scroll bar* na tela inicial, manter ou não as telas carregadas em memória, definir o período para inatividade, resolução das telas e várias outras configurações. Vale ressaltar que somente pode existir um *Viewer* por domínio (ELIPSE SOFTWARE, 2019b).

Neste trabalho não houve inicialmente configurações específicas acerca do *Viewer*, optando-se por deixar as opções padrões marcadas. Entretanto, a tela inicial deve ser definida no *Viewer* para a correta execução do supervisório, o que é feito mais adiante logo após a criação da primeira tela. Dessa forma, definiu-se o nome do *Viewer* criado como “ViewerPrototipo”, conforme ilustrado na Figura 25.

Figura 25 – Objeto *Viewer* criado no E3 Studio



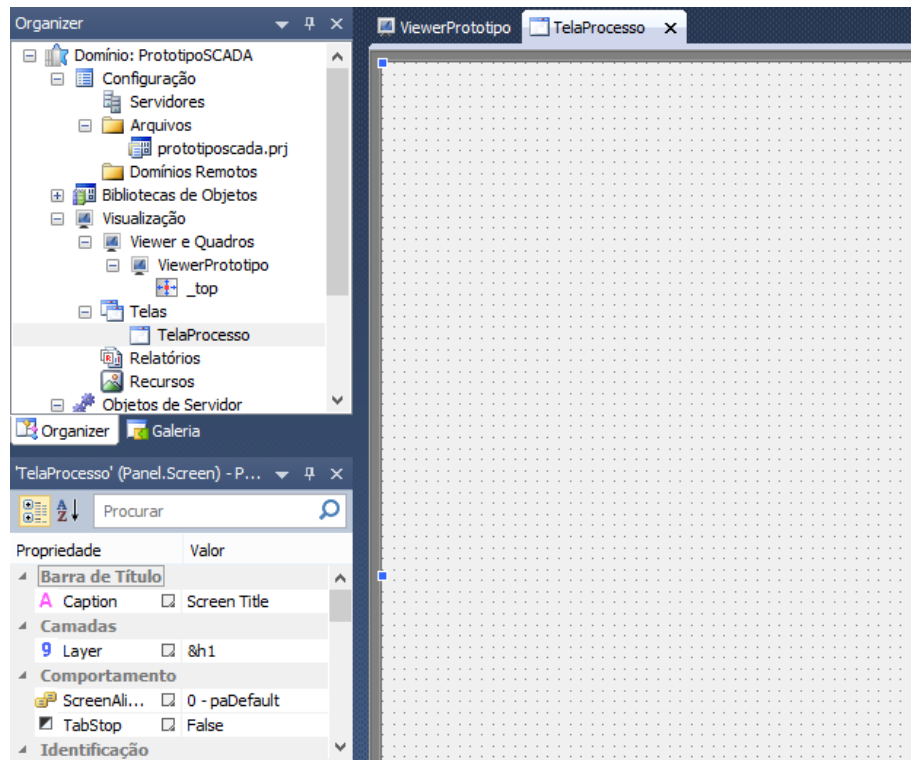
Fonte: Produção do próprio autor.

5.3.3 Criação da tela inicial

De acordo com Eclipse Software (2019b), telas são as janelas que são utilizadas para monitorar os processos. Em cada tela, os usuários podem inserir os objetos para compor a interface que o operador utilizará para se comunicar com o sistema, sendo eles chamados de objetos de tela. Vale ressaltar que cada aplicação pode ter um número ilimitado de telas e de objetos de tela.

No ato de criação de uma tela, são exigidas algumas informações básicas: o *splitter* onde a tela será aberta, o nome da tela, bem como sua largura e altura, em *pixels*. *Splitters* são objetos que exibem uma tela e que, quando combinados, podem dividir a exibição em múltiplas telas simultaneamente. A largura e altura da tela têm valores iniciais preenchidos automaticamente, de modo que a tela tenha tamanho suficiente para ser exibida sem a necessidade de *scroll bars* (ELIPSE SOFTWARE, 2019b). Há também a possibilidade de configurar a tela a ser criada como sendo a tela inicial do *splitter*. Entretanto, optou-se num primeiro momento por não alterar o tamanho da tela (por já ser o suficiente para o protótipo) e por não a estabelecer como a tela inicial, sendo isso feito posteriormente. Portanto, o *splitter* escolhido foi o padrão (criado automaticamente com o *Viewer*), *ViewerPrototipo.[_top]*, e o nome da tela foi definido como *TelaProcesso*. A Figura 26 ilustra graficamente como a tela é representada após sua criação, no *software E3 Studio*.

Figura 26 – Tela sem objetos, após criação



Fonte: Produção do próprio autor.

Vale ressaltar que o usuário pode configurar diversas propriedades da tela após a criação, mesmo as já definidas antes de ela ser criada. Isto é, pode-se configurar na lista de propriedades seu tamanho, cor, outros aspectos de comportamento e aparência e vários tipos de eventos, sem a necessidade de criar *scripts* para essas configurações.

5.3.4 Adicionando objetos à tela criada

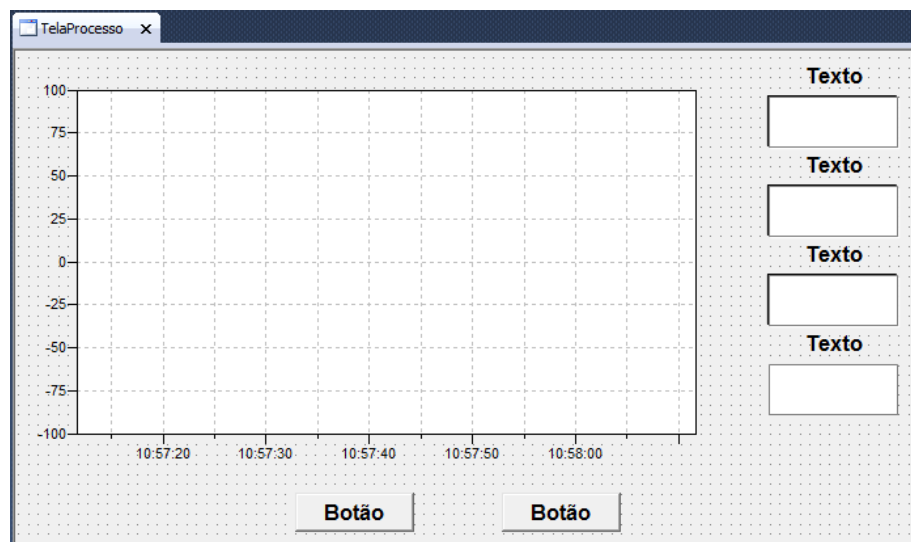
O *E3 Studio* possui um editor que pode ser utilizado para configurar ou adicionar diversos objetos a uma tela. Esses objetos podem ser objetos primitivos, como linhas, círculos, retângulos, polígonos, assim como podem ser imagens, controles de *ActiveX* como *E3Alarm*, *E3Browser*, *E3Chart* e *E3Playback*, além de diversos outros tipos de objetos (ELIPSE SOFTWARE, 2019b).

Como o protótipo proposto nesse trabalho simula um processo industrial, onde há a geração de dados e seu armazenamento em banco de dados, há a necessidade de utilizar alguns objetos na tela, como o *E3Chart*, *display*, botões e outros.

O primeiro objeto necessário à criação do protótipo é o *E3Chart*, que é basicamente um gráfico que exibe variáveis desejadas em tempo real, bem como dados históricos armazenados em um banco de dados. Por meio do editor do *E3 Studio*, pode-se adicionar um objeto *E3Chart* na tela desejada, neste caso *TelaProcesso*. Assim, define-se um tamanho inicial por meio do cursor, que pode ser alterado conforme as necessidades. Da mesma forma que o tamanho, a sua posição, nome, cores, variáveis a serem exibidas e diversas outras propriedades podem ser configuradas. Essas configurações são feitas ao longo deste capítulo, conforme a necessidade.

Os próximos objetos a serem inseridos na tela são: campos de texto, que auxiliam o usuário a identificar cada objeto na tela; *setpoints*, que são caixas de edição onde podem ser inseridas informações que serão atribuídas a *tags* (unidade de comunicação de um sistema supervisório, representando um equipamento, como por exemplo velocidade de rotação de um motor, temperatura de um forno, etc) associadas; *display*, que é um objeto que exibe os valores de alguma *tag* em tempo real; botões, que executam ações específicas ao serem pressionados. Desse modo, a Figura 27 ilustra o resultado provisório inicial da tela do protótipo proposto, ainda sem nenhuma configuração específica.

Figura 27 – Configuração provisória da tela inicial do protótipo



Fonte: Produção do próprio autor.

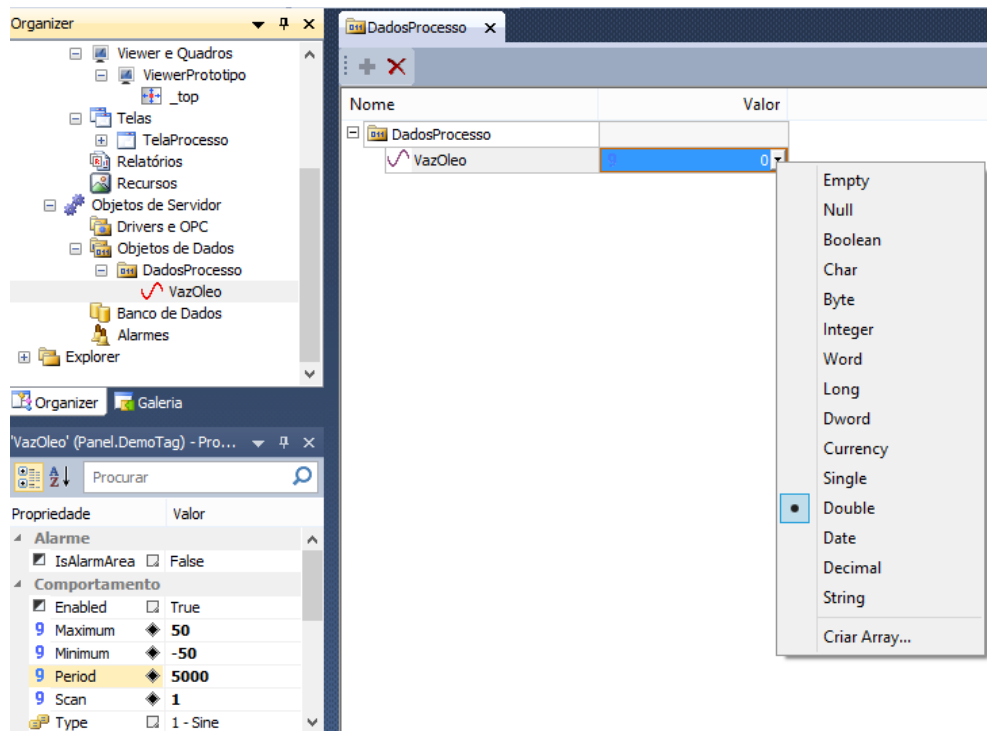
5.4 Desenvolvimento e Configurações do Projeto Criado

Uma vez realizadas as configurações básicas iniciais do projeto, torna-se fundamental realizar configurações mais avançadas para que o protótipo funcione adequadamente. Isto é, é preciso relacionar a tela com o *Viewer* criado, definir as funções e configurações de cada objeto criado na tela, criar e configurar as *tags demo* que serão utilizadas como variáveis simuladas de um processo industrial, criar e configurar o objeto de banco de dados que será associado ao banco de dados do Azure criado ao longo deste trabalho, além de outras configurações necessárias.

Nas propriedades do *ViewerPrototipo*, criado anteriormente, há a opção de definir uma tela inicial. Portanto, a tela inicial foi definida como a *TelaProcesso*, que é a utilizada na produção do protótipo. A partir deste momento, já é possível executar a aplicação e testar as configurações realizadas.

Entretanto, antes de iniciar os testes é necessário gerar valores com o objetivo de visualizá-los no gráfico criado. Para tanto, serão criadas *tags demo*. Uma *tag demo* é um objeto que gera valores de acordo com a forma de onda selecionada, sendo normalmente utilizada para simular valores. É possível, portanto, definir uma forma de onda ou determinar uma geração aleatória de números. Para criar uma *tag demo*, é preciso criar um servidor de dados, que é um objeto responsável por gerenciar e executar variáveis do sistema, como *tags* internas, *tags demo* e *tags* temporizadas. O servidor de dados criado foi nomeado de *DadosProcesso*. Em seguida, foi inserida uma *tag demo* com o nome de *VazOleo*, simulando dados de vazão de óleo provenientes de um processo industrial. A Figura 28 demonstra a *tag* criada no *E3 Studio*, sendo possível configurar o tipo de dado a ser gerado (*boolean*, *char*, *byte*, *integer*, *double*, *decimal*, entre outros) e diversas propriedades, no painel de propriedades, quanto a suas características, como valor máximo, valor mínimo, período, taxa de amostragem, forma de onda, etc. Para fins de praticidade e de didática, inicialmente o *range* da variável foi definido entre +50 e -50, com um período de 5 segundos e taxa de amostragem igual a 1 milissegundo, sendo selecionada a forma de onda como sendo senoidal.

Figura 28 – Tag demo e atributos no E3 Studio



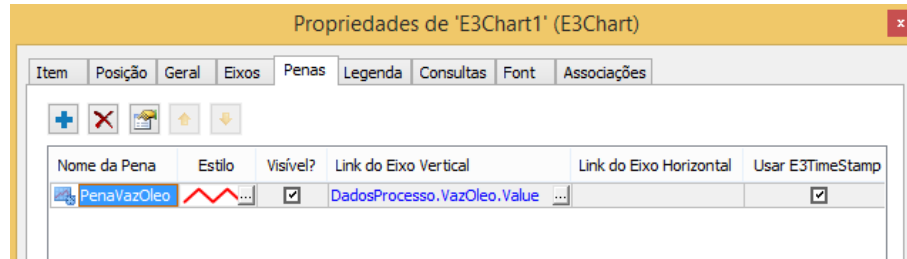
Fonte: Produção do próprio autor.

Uma vez criada a variável, é fundamental associá-la ao gráfico no qual ela será exibida. Isso é feito nas propriedades do gráfico criado na tela inicial, na aba de penas. Penas são objetos de um gráfico E3 que tem a função de exibir dados de acordo com a configuração feita. Nesse sentido, é possível criar uma pena para exibir no eixo vertical e/ou horizontal os valores de determinadas variáveis, bem como especificar diversas propriedades como estilo da linha, sua cor, espessura, cor de fundo, tipo de pena (tempo real, histórica, automática), cálculos estatísticos e diversas outras funções.

Para este trabalho foi criada uma pena com nome PenaVazOleo. O E3 Studio permite configurar o tipo de pena, que pode ser tempo real, histórica, tempo real e histórica, e automática. No tipo tempo real, os dados serão coletados de uma fonte (neste caso, a tag demo VazOleo criada anteriormente) e exibidos em tempo real, no momento da geração. No tipo histórica, os dados são coletados por consulta a um banco de dados local ou externo, de modo a exibir valores históricos de alguma variável. O tipo selecionado para esta pena foi tempo real, associando em seguida o eixo vertical à tag demo VazOleo. O eixo horizontal não precisou de configurações, pois foi especificada a utilização de estampas de tempo, que são valores de tempo que acompanham o valor da tag, indicando o momento em que esse valor foi alterado. A Figura 29

ilustra a janela de propriedades do gráfico, mostrando algumas configurações básicas da pena criada.

Figura 29 – Configurações da pena PenaVazOleo criada



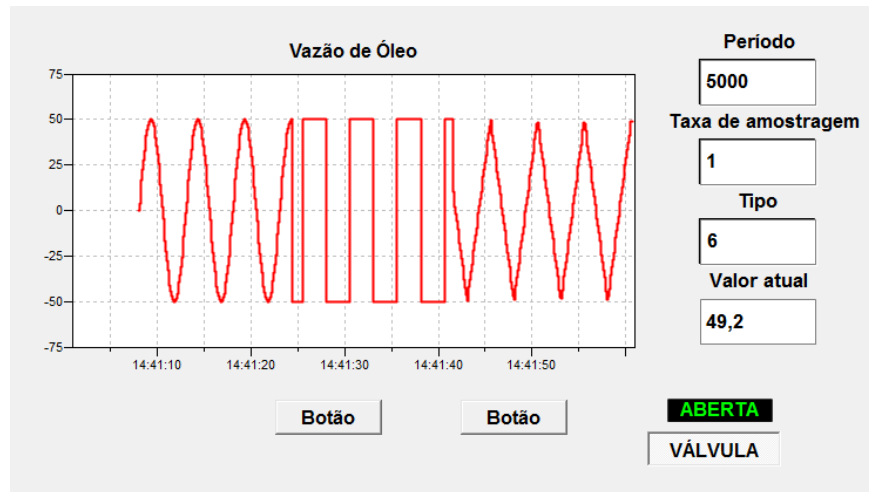
Fonte: Produção do próprio autor.

Ao finalizar essa configuração, o protótipo pode ser executado e os valores simulados da *tag demo* serão exibidos no gráfico da tela inicial. Entretanto, com o objetivo de simular as ações do operador, que deve monitorar tanto de forma passiva (observando os gráficos de tendência e alarmes) quanto de forma ativa (atuando no processo, estabelecendo *setpoints*, enviando comandos de ligamento ou desligamento, dentre outros), as caixas de edição de texto foram associadas às propriedades da *tag demo* (período, taxa de amostragem e tipo), o valor atual da variável foi associado ao display e foi criado um novo objeto, um botão liga/desliga que foi associado à uma válvula fictícia que permite a vazão de óleo, impedindo ou permitindo a variação de valores da *tag demo*.

Essas associações são feitas por meio de *links* que, de acordo com Eclipse Software (2019b), são conexões realizadas entre propriedades e objetos, ou entre propriedades. Com isso, torna-se mais fácil manipular a lógica da aplicação e desenvolver o projeto, minimizando o uso de *scripts*. Os *links* podem ser criados nas propriedades de cada objeto, na aba de associações. Desse modo, podem ser feitas conexões entre as propriedades disponíveis para esse objeto e uma outra fonte, sendo necessário especificar o tipo de conexão a ser realizado. Há 7 tipos de conexão, sendo os principais a conexão simples (a propriedade recebe o valor da fonte da fonte), conexão bidirecional (fonte escreve na propriedade e vice-versa) e a conexão reversa (propriedade escreve na fonte). Neste trabalho, as conexões adequadas foram feitas em cada objeto, de acordo com a necessidade. Sendo assim, a Figura 30 ilustra o funcionamento no E3 Viewer desse protótipo alterado ao executarmos a aplicação. Pode-se alterar os valores de período, taxa de amostragem e tipo inserindo os novos valores na caixa de texto, bem como

pode-se alterar o estado da válvula ao clicar no botão, que alterna entre os estados ligado/desligado.

Figura 30 – Protótipo em funcionamento inicial no E3 Viewer



Fonte: Produção do próprio autor.

5.5 Conectando a um Banco de Dados

Estando o supervisório configurado adequadamente, de modo a simular o monitoramento de um processo industrial, o próximo passo se dá por meio da comunicação da plataforma Elipse E3 com o banco de dados do Azure criado no capítulo anterior. No contexto do E3, o objeto banco de dados é utilizado para guardar informações relacionadas ao projeto, como dados históricos, fórmulas, alarmes e outros. De acordo com Elipse Software (2019b), o E3 dá suporte a três tipos de bancos de dados: Access, *SQL Server* e Oracle. Como esperado, a configuração neste trabalho é feita utilizando *SQL Server*.

No ato da criação do objeto banco de dados no projeto PrototipoSCADA, é preciso efetuar certas configurações, como o tipo de banco de dados, o endereço do servidor, o nome do banco de dados, a biblioteca de rede e as credenciais de *login*. É permitido ao usuário informar parâmetros adicionais de conexão, embora não utilizados neste primeiro momento. O tipo de banco de dados selecionado deve ser o *SQL Server*, como mencionado, enquanto o endereço do servidor é o endereço do servidor *SQL foregaster*, criado no capítulo anterior: *foregaster.database.windows.net*. O nome do banco de dados corresponde a ValeDB, conforme indica a Figura 13, e nos campos de usuário e senha deve-se informar as credenciais de *login*,

sendo neste trabalho os dados da conta de administrador ForeTeste. A seção a seguir discorrerá sobre um parâmetro importante de configuração da conexão, a biblioteca de rede.

5.5.1 Comunicação do E3 com o banco de dados SQL do Azure

Para que haja a troca de informações entre um programa e um banco de dados, é necessário utilizar uma ferramenta que permita isso. Do ponto de vista do Azure, segundo Microsoft (2018c), é oferecido suporte a diversas bibliotecas e *frameworks* para realizar a conexão com bancos de dados SQL do Azure. Dessa forma, há uma grande quantidade de linguagens de programação que podem ser usadas para efetuar a conexão. Dentre as bibliotecas ou *drivers* que as aplicações podem usar para se conectar, algumas são via ADO.NET, *driver* JDBC, *driver* ODBC, *drivers* para PHP, Node.js e Python, além de vários outros.

De acordo com Elipse Software (2019b), o E3 utiliza um mecanismo chamado *ActiveX Data Objects* (ADO). Esse mecanismo foi criado pela Microsoft e corresponde a um conjunto de objetos COM (*Component Object Model*), que fornece a desenvolvedores um modelo de objeto lógico para acessar, editar e atualizar dados de diversas fontes de dados por meio de interfaces do sistema OLE DB (MICROSOFT, 2017a). Nesse sentido, o E3 utiliza o ADO para realizar consultas a tabelas no banco de dados SQL do Azure, de modo a fazer leituras e gravações.

O OLE DB é uma API desenvolvida com o objetivo de possibilitar o acesso a diversas fontes de dados. No contexto deste trabalho, há um provedor OLE DB voltado para *SQL Server*, que permite que o ADO acesse os dados de algum banco de dados SQL. Em outras palavras, ele é um provedor de dados que os expõe em forma de tabelas, utilizando um número mínimo de componentes para isso (MICROSOFT, 2017c). Para utilizá-lo, é necessário especificar nos parâmetros da cadeia de conexão exigidos pelo ADO o provedor desejado, o OLE DB, assim como os dados usuais de endereço do servidor, nome do banco de dados, nome de usuário e senha. Além desses parâmetros especificados pelo ADO, há outros que são característicos desse provedor, sendo um deles a biblioteca de rede. Esse parâmetro indica o nome da biblioteca de rede que será utilizada para se comunicar com o *SQL Server* (MICROSOFT, 2017b).

Dessa forma, o E3 disponibiliza 5 opções de biblioteca de rede a serem utilizadas: *Named Pipes*, *Winsock TCP/IP*, *SPX/IPX*, *Banyan Vines* e *Multi-Protocol* (RPC). Em se tratando de *SQL Server*, normalmente pode-se configurar qualquer um desses protocolos para sua devida

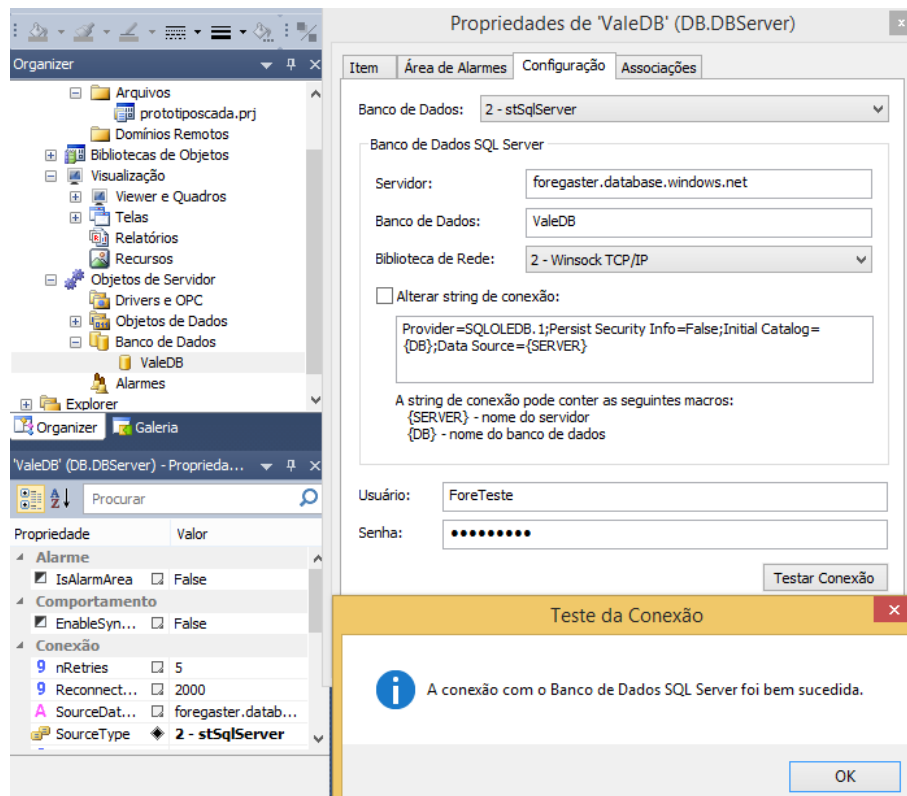
utilização. Entretanto, de acordo com Microsoft (2018b), o banco de dados do Azure fornece suporte somente ao protocolo da camada de aplicação TDS (*Tabular Data Stream*), que requer que o banco de dados seja acessível somente pela porta TCP/1433. Isso significa que a biblioteca de rede utilizada necessariamente deve ser a TCP/IP.

Por fim, vale ressaltar que a comunicação com o Azure é feita via protocolo de segurança TLS, conforme visto no capítulo anterior. Isto é, independente das configurações do E3, o Azure força que a comunicação utilize uma das versões do TLS e criptografa toda a comunicação, seguindo os princípios das 4 camadas de segurança para a proteção de dados.

5.5.2 Testando a conexão

Ao finalizar essas configurações iniciais, é possível realizar o teste da conexão para verificar se as informações foram preenchidas corretamente, sendo assim retornada uma conexão bem-sucedida. A Figura 31 ilustra a configuração final do objeto banco de dados, também apelidado de ValeDB, e uma tentativa de conexão efetuada com sucesso com o banco de dados SQL do Azure.

Figura 31 – Conexão do E3 *Studio* com o banco de dados SQL do Azure



Fonte: Produção do próprio autor.

5.5.3 Armazenando registros de variáveis

Para armazenar dados de quaisquer variáveis de processo desejadas, é necessário inserir um objeto Histórico no projeto do *E3 Studio*. Segundo Elipse Software (2019b), esses objetos são módulos cuja responsabilidade é armazenar dados da aplicação em um banco de dados. Dessa forma, cada objeto Histórico pode criar uma tabela em um banco de dados ou utilizar uma já existente, na qual será feito o registro dos dados. Vale ressaltar que esse registro pode ser feito de duas maneiras, sendo por tempo ou por evento, o que será explicado mais adiante.

Neste trabalho, o Histórico criado foi nomeado *HistoricoProcessoTempo*. Após a criação, abre-se uma interface de histórico onde permite-se realizar algumas configurações da estrutura tabela a ser utilizada, tais como adicionar um objeto de índices, um objeto *PrimaryKey*, campos de histórico (são as colunas da tabela, armazenando informações de cada registro), um índice de campo ou um campo de *PrimaryKey*. Além disso, é criado automaticamente um campo de histórico chamado *E3TimeStamp*, que representa a hora e a data na qual o registro foi realizado. Entretanto, é importante saber que essas configurações devem ser feitas manualmente somente caso seja optado pela criação da tabela pelo *E3 Studio*. Isto é, caso seja utilizada uma tabela pré-existente do banco de dados, o *E3 Studio* realiza a importação da estrutura da tabela, criando os campos, índices e *PrimaryKey* automaticamente, de acordo com as configurações da tabela original.

Levando-se em conta a praticidade e o melhor funcionamento do *software*, optou-se por configurar a tabela e realizar sua criação por meio do *E3 Studio*. Para tanto, é necessário criar primeiramente os campos de histórico e associar as *tags* a eles. Todavia, optou-se por utilizar os dois tipos de gravação de dados mencionados anteriormente: por tempo e por evento, o que leva a um desdobramento importante. Na gravação por tempo, os valores selecionados são armazenados periodicamente na tabela do banco de dados, de acordo com o intervalo de tempo definido nas propriedades do Histórico. Já a gravação por eventos é conhecida como registro manual, uma vez que não é realizada automaticamente conforme o tempo passa. Isto é, ela é feita com a ocorrência de um evento, como quando um botão é pressionado. Isso significa que, uma vez determinada a utilização das duas abordagens de registro, surge a necessidade de criar um Histórico para cada tipo de gravação, uma vez que não é permitido armazenar dados numa mesma tabela em decorrência de tempo e evento.

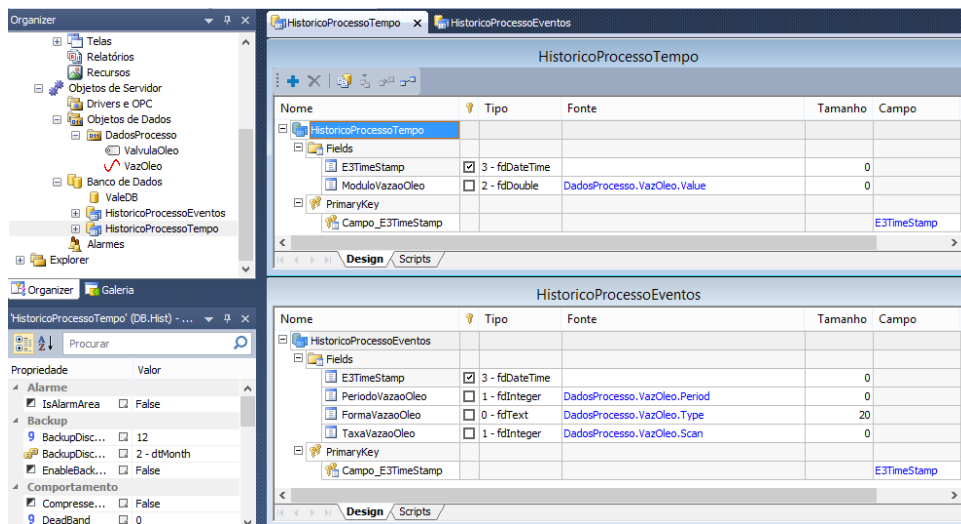
Neste primeiro momento, os valores que serão armazenados no banco de dados são o módulo da *tag* de vazão de óleo, o período da onda simulada, a forma de onda, a taxa de amostragem e o *E3TimeStamp*, que é o tempo no qual o registro é realizado. Dessa forma, para evitar a escrita contínua de valores que são estáticos durante a maior parte do tempo, optou-se por fazer a gravação por tempo somente do módulo da vazão e do tempo em que isso ocorre, sendo o período, a forma de onda e a taxa de amostragem armazenados por meio de gravação por eventos, em decorrência de um comando manual de um botão. Dessa forma, no *HistoricoProcessoTempo* foi criado um campo de histórico (equivalente a uma coluna da tabela) chamado *ModuloVazaoOleo*, sendo associado ao valor instantâneo da *tag demo VazOleo* e configurado para conter valores do tipo *double*. Além disso, foi criado um objeto *PrimaryKey*, que representa um conjunto de campos de histórico que identifica de forma única cada registro na tabela. Nesse objeto, foi adicionado um campo *PrimaryKey*, que tem como objetivo compor esse conjunto que define uma *PrimaryKey*. Esse campo foi associado ao *E3TimeStamp*, visto que é o campo de histórico que identifica cada registro de forma única. Isso porque o armazenamento é feito a cada segundo, ou seja, não é permitido dois registros diferentes de módulo de vazão ao mesmo tempo.

Para os demais valores a serem armazenados (período, forma de onda e taxa de amostragem), foi preciso criar um novo objeto Histórico de nome *HistoricoProcessoEventos*. Nesse objeto foram criados os campos de histórico *PeriodoVazaoOleo*, *FormaVazaoOleo* e *TaxaVazaoOleo*, que correspondem ao período da onda, à forma de onda e à taxa de amostragem, respectivamente. Os três campos foram configurados para armazenar valores do tipo inteiro. Neste caso também foi criado um objeto *PrimaryKey*, sendo criado um campo associado ao *E3TimeStamp* deste novo Histórico.

Após a definição inicial da estrutura básica das tabelas a serem criadas, torna-se necessário realizar algumas configurações acerca das propriedades dos Históricos, tais como especificar o objeto banco de dados criado no E3, determinar um nome para as tabelas a serem criadas, escolher um intervalo de tempo que indica a frequência em que os dados serão armazenados na tabela (sendo igual a zero para o tipo gravação por eventos) e configurar opções de descarte e *backup* de dados. O banco de dados selecionado foi o criado anteriormente, ValeDB. O nome da tabela do *HistoricoProcessoTempo* foi definido como *ModuloVazaoOleoProcesso* e o

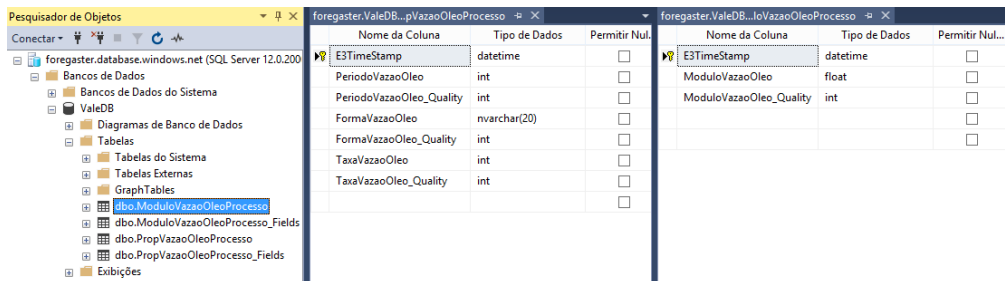
intervalo entre registros foi escolhido como 1 segundo. Já o nome da tabela do HistoricoProcessoEventos foi determinado como PropVazaoOleoProcesso, sendo o intervalo de registros igual a zero. Dada a natureza deste trabalho e a falta de necessidade em um primeiro momento, a opção de descarte de dados não foi ativada, sendo mantidos os dados na tabela indefinidamente. Por fim, confirmam-se as configurações ao dar o comando de geração da estrutura no banco de dados, onde o E3 se conecta ao banco de dados e cria as tabelas com as propriedades definidas. A Figura 32 e a Figura 33 ilustram as interfaces de histórico do E3 *Studio* com as configurações das tabelas, bem como o resultado no banco de dados visualizado no SSMS, após a criação das estruturas pelo E3.

Figura 32 – Interfaces de histórico no E3 *Studio*



Fonte: Produção do próprio autor.

Figura 33 – Visualização no SSMS das estruturas das tabelas criadas pelo E3 no banco de dados



Fonte: Produção do próprio autor.

Pela Figura 33, pode-se perceber que foram criadas duas tabelas além das mencionadas anteriormente, com terminações *_Fields*. Essas tabelas foram criadas automaticamente pelo E3 no momento da geração da estrutura das tabelas. Elas contêm informações sobre cada campo

armazenado no objeto de histórico, tais como nome, descrição, tipo de qualidade, tamanho, fonte dos dados, tipo dos dados, dentre outros. Além disso, também é perceptível que foram criados campos de histórico adicionais, com a terminação *_Quality*. Esses campos representam o estado da qualidade do valor de uma variável, podendo assumir valores na faixa de 0 a 255. Como neste trabalho há a utilização somente de *tags demo*, esses campos não têm utilidade prática, uma vez que não estão sendo utilizadas *tags OPC* (objetos cujos valores são lidos ou escritos por um dispositivo externo), de modo que a qualidade sempre será apontada como boa (ELIPSE SOFTWARE, 2019b). O Quadro 2 demonstra em mais detalhes as classificações dos valores de cada *tag* de acordo com a faixa de valores que o campo *_Quality* assume, assim como uma breve descrição.

Quadro 2 – Faixa de valores da qualidade e significados

<i>Quality</i>	Qualidade do valor	Descrição
0 – 63	Ruim	Este valor não pode ser utilizado, por motivos de erros, falhas ou razões não especificadas
64 – 127	Incerta	A qualidade deste valor é incerta devido a imprecisão nos sensores, ultrapassagem do limite de valores ou outros motivos
128 – 191	(Reservado)	Não é utilizado pelo padrão OPC
192 – 255	Boa	A qualidade do valor é boa

Fonte: Elipse Software (2019b).

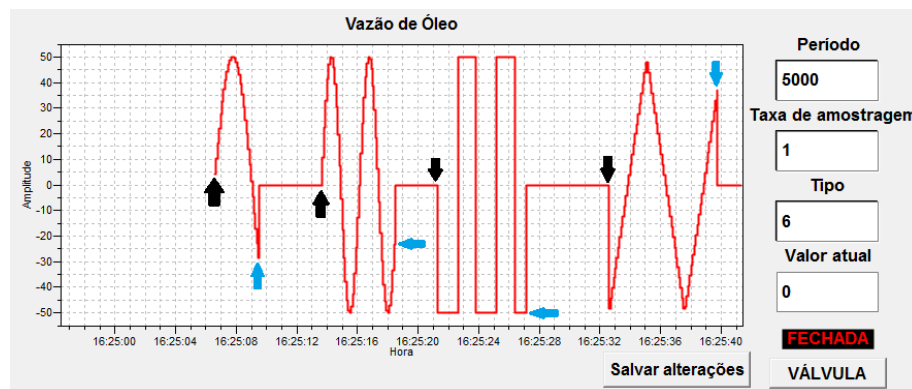
Nota: Modificado pelo autor.

5.6 Testando o Protótipo com o Banco de Dados

Uma vez realizadas todas as configurações com o banco de dados, pode-se fazer outro teste do protótipo, avaliando o funcionamento da comunicação do E3 com o banco de dados do Azure. Para tanto, houve a necessidade de ajustar os componentes da tela exibida ao usuário, de modo a criar *scripts* para ajustar as novas propriedades da *tag* e para realizar a gravação por eventos, que é feita por meio de um clique em um botão específico. Além disso, outros ajustes foram

feitos para melhorar visualização e compreensão do processo envolvido. O teste em questão envolveu algumas operações: primeiramente iniciou-se o funcionamento do E3 *Server* e do E3 *Viewer*, que inicializou a *tag demo* com os valores pré-definidos (uma senoide com amplitude igual a 50, taxa de amostragem de 1 milissegundo e período igual a 5 segundos); em seguida, deu-se início a algumas operações de mudança no processo. Antes de cada mudança, fecha-se a válvula de vazão por meio de um botão do tipo *switch* (que interrompe a vazão de óleo, zerando o valor da *tag*). Enquanto fechada, alteram-se as propriedades da *tag* de vazão e confirma-se as alterações por meio do botão Salvar alterações, que as implementa e dá o comando para que o E3 armazene os novos valores no histórico (realizando, portanto, gravação por evento). Com isso, abre-se novamente a válvula de vazão de óleo e sua amplitude passa a variar de acordo com as novas configurações. Esse procedimento foi feito três vezes: alterou-se o período da senoide para 2,5 segundos, seguido de uma alteração na forma de onda para onda quadrada e finalizou-se o teste ao retornar o período para 5 segundos com mudança de forma de onda para onda triangular. A Figura 34 ilustra todo esse procedimento que foi feito utilizando o E3 *Viewer*, onde as setas pretas indicam o momento em que houve gravação por evento de alguns parâmetros da *tag demo* (foi configurado para realizar a primeira gravação por evento assim que o protótipo entrasse em execução) e, conseqüentemente, a abertura da válvula, e as setas azuis indicam os momentos em que houve o fechamento da válvula para alteração das propriedades. Já a Figura 35 e a Figura 36 demonstram o resultado dessas operações no banco de dados, visto pela ferramenta SSMS.

Figura 34 – Teste do protótipo por meio do E3 *Viewer*



Fonte: Produção do próprio autor.

Figura 35 – Registros de período, forma de onda e taxa de amostragem no banco de dados, visto pelo SSMS

```

SELECT [E3TimeStamp],
       [PeríodoVazaoOleo],
       [FormaVazaoOleo],
       [TaxaVazaoOleo]
FROM [dbo].[PropVazaoOleoProcesso]

```

	E3TimeStamp	PeríodoVazaoOleo	FormaVazaoOleo	TaxaVazaoOleo
1	2019-11-14 16:25:06...	5000	1	1
2	2019-11-14 16:25:13...	2500	1	1
3	2019-11-14 16:25:20...	2500	2	1
4	2019-11-14 16:25:32...	5000	6	1

Fonte: Produção do próprio autor.

Figura 36 – Registros de módulo de vazão no banco de dados, visto pelo SSMS

```

SELECT [E3TimeStamp],
       [ModuloVazaoOleo]
FROM [dbo].[ModuloVazaoOleoP]

```

	E3TimeStamp	ModuloVazaoOleo
1	2019-11-14 16:25:06...	4,142149708010
2	2019-11-14 16:25:07...	48,72634363932
3	2019-11-14 16:25:08...	25,45207078751
4	2019-11-14 16:25:09...	0
5	2019-11-14 16:25:10...	0
6	2019-11-14 16:25:11...	0
7	2019-11-14 16:25:12...	0
8	2019-11-14 16:25:13...	0
9	2019-11-14 16:25:14...	30,39651488473
10	2019-11-14 16:25:15...	-47,70574914452
11	2019-11-14 16:25:16...	47,51384345011
12	2019-11-14 16:25:17...	-29,79442696355
13	2019-11-14 16:25:18...	0
14	2019-11-14 16:25:19...	0
15	2019-11-14 16:25:20...	0
16	2019-11-14 16:25:21...	-50
17	2019-11-14 16:25:22...	50
18	2019-11-14 16:25:23...	50
19	2019-11-14 16:25:24...	-50
20	2019-11-14 16:25:25...	50
21	2019-11-14 16:25:26...	-50

Fonte: Produção do próprio autor.

A Figura 35 está em consonância com o processo ilustrado pela Figura 34, onde os momentos de gravação no banco de dados são os mesmos em que houve alterações no E3 Viewer. Da mesma forma, os valores armazenados estão coerentes com o que foi comandado. Vale ressaltar que o período foi armazenado em unidade de milissegundo, e as formas de onda 1, 2 e 6 são senoidal, quadrada e triangular, respectivamente. A Figura 36 demonstra os valores que a tag

demo assumiu durante o processo, sendo claramente perceptível os momentos em que a válvula entrou em atuação (zerando a vazão), que são coerentes com as alterações ilustradas pelas demais figuras.

5.7 WebViewer

Há duas formas de se monitorar um processo pelo E3: por meio do *Viewer* ou pelo *WebViewer*. O *Viewer*, conforme já mencionado ao longo deste capítulo, é o módulo pelo qual usuários podem monitorar as aplicações criadas no *E3 Studio*, observando as variáveis de interesse e dando comandos de operação. Nesse sentido, o *WebViewer* é uma ferramenta baseada em *ActiveX* que permite que o navegador *Internet Explorer* se comporte como o *Viewer*, de modo que se torne possível manipular o supervisório via *internet*. Em outras palavras, com essa ferramenta pode-se monitorar e atuar em processos e em equipamentos de chão de fábrica por meio do *Internet Explorer*, por meio de qualquer computador que esteja na mesma rede do *E3 Server* (ELIPSE SOFTWARE, 2019b).

Vale ressaltar que não é necessário ter nenhum componente do E3 instalado na máquina em questão, uma vez que todos os arquivos necessários serão baixados no momento do acesso ao supervisório, pelo *Internet Explorer*. Dessa forma, o *WebViewer* funcionará da mesma forma que o *Viewer*, dando suporte a todas as ferramentas e recursos. A Figura 37 demonstra um exemplo de *WebViewer*, sendo executado de forma a visualizar uma aplicação qualquer.

Figura 37 – Exemplo de execução de um *WebViewer*



Fonte: Elipse Software (2019b).

Nota: Modificado pelo autor.

5.7.1 Configurando o *WebViewer*

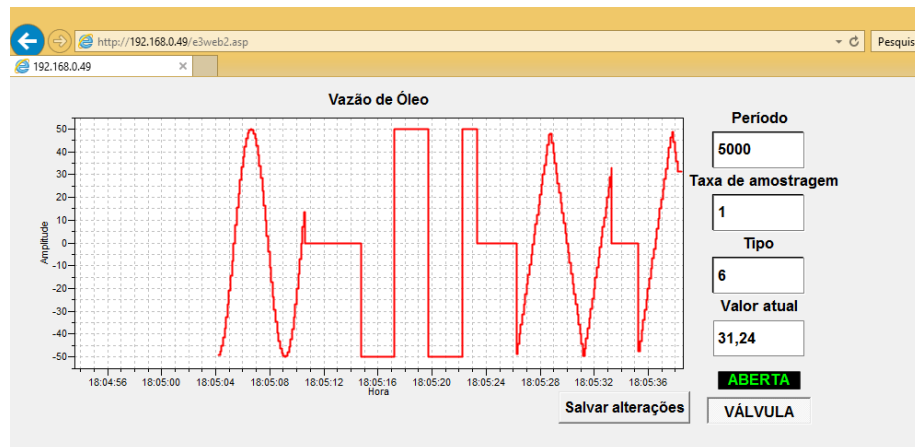
De acordo com Elipse Software (2019a), para que seja possível utilizar o *WebViewer* é necessário que seja instalado na máquina do servidor algum servidor *web* que seja compatível com páginas ASP (*Active Server Pages*). A empresa sugere o *Internet Information Services* (IIS) como o servidor *web* a ser utilizado, uma vez que ele já vem instalado com o sistema operacional do Windows. Entretanto, por motivos de maior familiaridade e maior praticidade, optou-se pela utilização do software *Uniform Server*, que é uma solução de servidor WAMP gratuita e disponível para a plataforma Windows e que permite executar um servidor *web*. Essa solução implementa as últimas versões do Apache2, MySQL, PHP e de outros recursos, sendo mais do que o suficiente para este trabalho.

Ao instalar o servidor *web* na máquina, é necessário lidar com alguns arquivos que são criados na instalação do E3. Para uma melhor compreensão, é importante saber que o processo de implementação do *WebViewer* é semelhante à hospedagem de um *site*. Isto é, move-se determinados arquivos do E3 para a pasta em que o servidor *web* utiliza para hospedar páginas *web*, sendo assim o *WebViewer* acessado pelo navegador como se um *site* fosse. No total, são 5 arquivos essenciais para o funcionamento: *e3web.asp* e *docwrite.asp* (responsáveis por carregar e iniciar o *E3Downloader*); *e3downloader.cab* (*ActiveX* responsável por instalar arquivos necessários à execução do *WebViewer* na máquina do cliente); *e3web2.asp* e

docwrite2.asp (responsáveis por carregar e executar o *WebViewer* na máquina do cliente). Esses arquivos podem sofrer configurações adicionais pelo usuário, com o objetivo de atingir as necessidades do projeto. Além disso, é necessário mover para a pasta o instalador do *WebViewer*, que deve ser compatível com a versão do *E3 Server* (ELIPSE SOFTWARE, 2019b).

Após o passo anterior, resta realizar algumas configurações opcionais, como definir um diretório virtual e uma página padrão para ser aberta toda vez que o servidor *web* for acessado. Com isso, basta abrir o navegador *Internet Explorer* (único navegador ao qual é oferecido suporte de maneira nativa pelo E3) e acessar pela barra de navegação o endereço de IP da máquina que executa tanto o *E3 Server* quanto o servidor *web* do *WebViewer*. A Figura 38 demonstra a tela executada pelo *Viewer*, vista pela Figura 34, sendo exibida por meio do *WebViewer*, que neste caso é executado em uma máquina distinta da que opera o *E3 Server*.

Figura 38 – Aplicação do protótipo monitorada por meio do *WebViewer* em uma máquina distinta do servidor



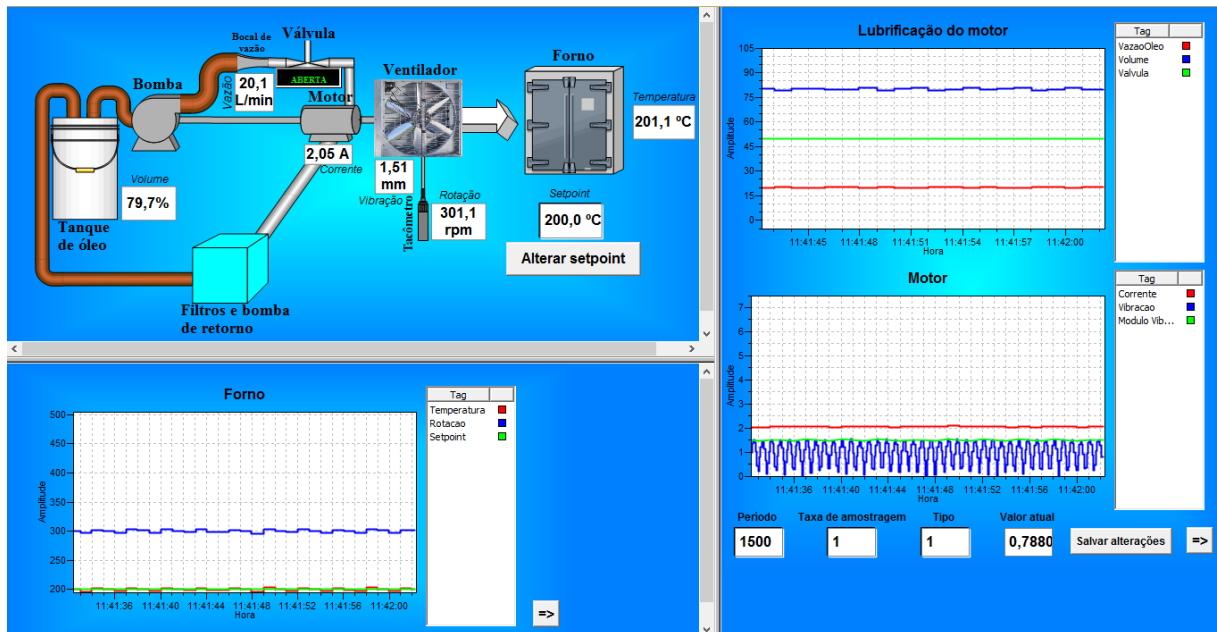
Fonte: Produção do próprio autor.

5.8 Funcionamento do Protótipo Final

Ao conhecer e, por conseguinte, realizar as configurações de todos os componentes essenciais ao funcionamento do protótipo proposto, entende-se que os objetivos propostos neste trabalho foram alcançados. Entretanto, com o objetivo de aperfeiçoar o protótipo construído e de trazê-lo à realidade da indústria moderna, foram feitas diversas melhorias, como alterações em seu *layout*, adições de novas variáveis monitoradas (*tags* internas que possuem objetivo de armazenar valores), inclusão de alarmes para as novas *tags* e também do componente

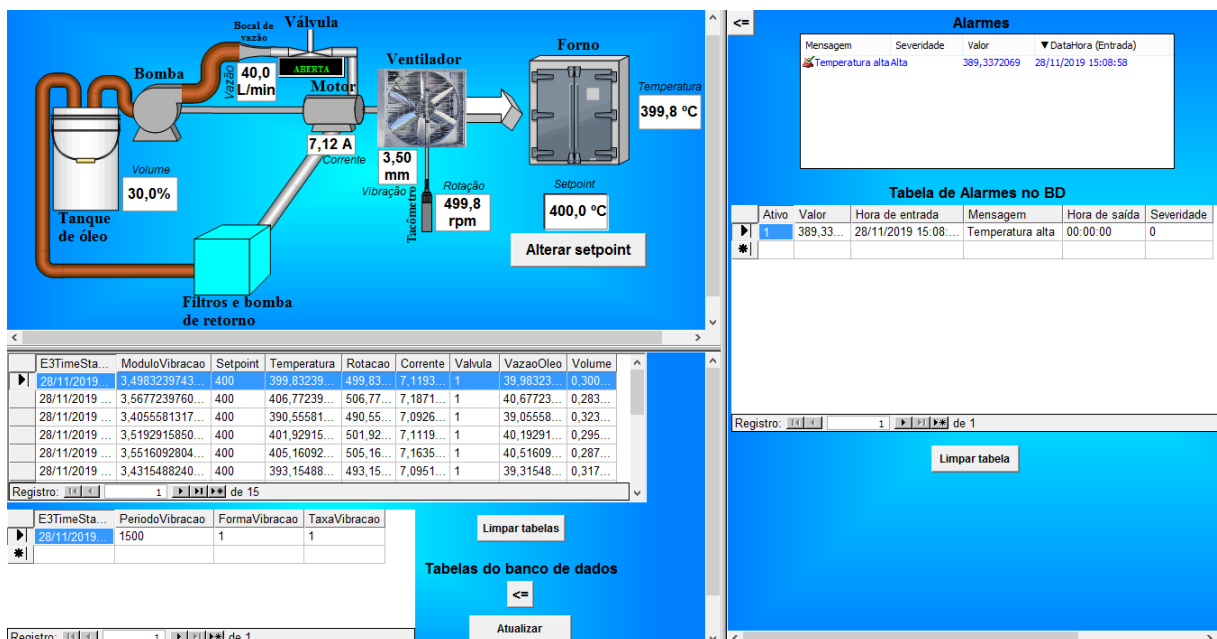
E3Browser, que faz consultas a tabelas do banco de dados e expõe os resultados nas telas do Viewer, com o objetivo de facilitar a compreensão do processo envolvido neste trabalho. Essas melhorias de modo geral foram feitas seguindo a mesma metodologia exposta ao longo deste estudo, resultando na versão final da aplicação que é demonstrada na Figura 39 e na Figura 40.

Figura 39 – Telas de monitoramento visualizadas pelo Viewer do protótipo do E3 melhorado



Fonte: Produção do próprio autor.

Figura 40 – Telas de alarmes e de E3Browser visualizadas pelo Viewer do protótipo do E3 melhorado



Fonte: Produção do próprio autor.

As melhorias expostas nas figuras anteriores foram feitas tomando como inspiração o processo de queima de pelotas de minério de ferro feito em diversas siderúrgicas no país. Dessa forma, foi implementado um esquema simplificado como supervisorio desse processo sob as limitações impostas pelo uso da versão demo do E3, conforme explicitado no início do capítulo. A nova estrutura consiste em um tanque de óleo, de onde uma bomba mecânica extrai óleo e envia aos mancais de um motor elétrico que aciona um ventilador industrial, que por sua vez sopra ar quente nas pelotas dentro do forno. A bomba mecânica está acoplada ao eixo do motor elétrico, de maneira que lubrificação dos mancais cresce conforme a rotação do motor. O sistema também contempla uma válvula que interrompe a vazão de óleo e inclui alguns medidores, que possuem o objetivo de informar os valores das novas variáveis criadas, sendo elas o volume do tanque, a vazão de óleo, a corrente e vibração do motor elétrico, a rotação do ventilador, a temperatura das pelotas e o *setpoint* de temperatura. Existe, portanto, a possibilidade de alteração do *setpoint* de temperatura das pelotas do forno por meio de simples comando pelo supervisorio, alterando o valor da caixa de texto e pressionando o botão correspondente para executar a ação.

Para esse novo sistema criado, foram definidos valores arbitrários para dois pontos de operação (mínimo e máximo) e foi feita uma linearização entre esses pontos, tendo as variáveis uma dependência simples e linear uma com a outra. Os pontos de operação escolhidos (que possuem valores totalmente fictícios e sem conexão com a realidade) são sintetizados de acordo com o Quadro 3.

Quadro 3 – Pontos de operação do novo sistema criado

Ponto de operação	Temperatura (°C)	Rotação (rpm)	Vibração (mm)	Corrente (A)	Vazão (L/min)	Volume (%)
1 (mínimo)	200,0	300,0	1,5	2,0	20,0	80,0
2 (máximo)	400,0	500,0	3,5	7,0	40,0	30,0

Fonte: Produção do próprio autor.

À exceção da rotação, todas as variáveis são uma função linear de somente uma única outra variável, onde essa função foi criada utilizando os dois pontos de operação expostos no Quadro 3. Para a rotação, foi necessário criar uma função linear de duas variáveis: a corrente elétrica e a vibração. Isso foi feito tendo em vista a alteração da *tag demo*, que passou a representar a

vibração ao invés da vazão de óleo. Nesse sentido, a vibração passou a ser um fenômeno senoidal que possui amplitude igual aos valores dos pontos de operação mencionados, com o objetivo de causar uma pequena flutuação no sistema de modo a trazê-lo um pouco mais para a realidade, buscando evitar que os valores das variáveis no supervisório fiquem estáticos. Para tanto, foi necessário introduzir na rotação uma certa dependência da vibração (tendo apenas 10% do peso da dependência da corrente elétrica), que agora influenciará pequenas mudanças em todas as demais variáveis a todo o instante. As equações produzidas para corrente, amplitude de vibração, rotação, temperatura, vazão de óleo e volume do tanque são representadas pelas equações (1) a (6), respectivamente.

$$I = I + k \times 0,025 \times (S - T) - 3 \quad (1)$$

$$ModV_i = 0,01 \times R - 1,5 \quad (2)$$

$$R = 38,4615 \times I + 3,8462 \times V_i + 217,3077 \quad (3)$$

$$T = R - 100 \quad (4)$$

$$Q = 0,1 \times R - 10 \quad (5)$$

$$V_o = -0,025 \times Q + 1,3 \quad (6)$$

Onde S é o *setpoint* definido pelo usuário, V_i é o valor instantâneo da vibração e k é um novo parâmetro introduzido, cujo valor foi arbitrado em 0,4 e cujo objetivo é fazer com que o sistema caminhe de maneira gradual para o ponto de operação estabelecido pelo *setpoint*, ao invés de maneira instantânea. Assim, a variável de corrente atualiza-se iterativamente, assumindo seu valor prévio somado com uma fração da diferença entre o *setpoint* e a temperatura atual. Essa rotina de atualização do valor das variáveis é feita por meio de uma *tag* temporizadora criada, que executa o *script* criado em intervalos de um segundo. Dessa forma, tem-se um sistema interdependente que pode ser alterado por meio do *setpoint* ou por flutuações da vibração do motor.

Para facilitar o monitoramento por parte do operador, foram inseridos três gráficos que exibem os valores que as variáveis do sistema assumem, conforme já demonstrado pela Figura 39. Vale mencionar que a vibração, para todos os efeitos, foi utilizada extraindo-se o módulo de seu valor atual, assumindo, portanto, a forma de onda módulo de seno. Além disso, no gráfico pertinente a válvula assume valor 50 se estiver aberta e 0 se estiver fechada, apenas com fins

de melhor visualização. A Figura 39 ilustra o sistema no ponto de operação 1, com valores flutuando de acordo com a vibração do motor.

Com o objetivo de melhorar a observação da operação do supervisor junto ao banco de dados, foram inseridos objetos *E3Browser*, que realizam consultas via SQL ao banco de dados em nuvem criado e exibem seus valores em tabelas, conforme visto pela Figura 40, reunindo todos os dados referentes a cada uma das variáveis utilizadas. Outro ponto é a criação de um alarme para a variável temperatura, que é disparado quando ela ultrapassa o valor de 380 °C, exibindo o alarme em um painel juntamente com uma mensagem de alerta pré-configurada, além de armazenar sua ocorrência em uma nova tabela no banco de dados em nuvem. A Figura 40 demonstra o sistema no ponto de operação 2, tendo disparado o alarme de temperatura alta, uma vez que a temperatura das pelotas ultrapassou o limiar estabelecido (380 °C).

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi apresentado um protótipo de sistema SCADA criado por meio da plataforma Eclipse E3, com o objetivo de realizar a comunicação com um banco de dados em nuvem, neste caso do ambiente Azure, ao invés de um banco de dados local. Vale ressaltar que todo o desenvolvimento teve um foco voltado para o contexto industrial, embora seja passível de aplicação em outras situações.

Inicialmente foram apresentados todos os conceitos que envolvem os sistemas que adotam a estrutura SCADA, bem como todos os benefícios proporcionados por sua utilização. Dessa maneira, foi explicada a importância desse tipo de sistema no mundo moderno, dadas as inúmeras aplicações industriais. Em seguida, foi trazida à discussão toda a filosofia da computação em nuvem, em contraste com a aquisição de *hardware* próprio. Isto é, dado que sistemas industriais geram um grande volume de dados e, portanto, demandam armazenamento, foram abordados o funcionamento e as vantagens proporcionadas pelo uso de recursos em nuvem, de forma a reduzir custos operacionais desses processos industriais, com foco em sistemas SCADA.

Tendo essa discussão em vista, foi sugerida uma plataforma de grande uso atualmente que adota os conceitos de computação em nuvem mencionados. Nesse sentido, foi abordado todo o conhecimento necessário para a produção do protótipo proposto, levando em consideração a manipulação da plataforma em si e formas de criação do banco de dados. Entretanto, optou-se por detalhar outros assuntos envolvendo a plataforma e banco de dados em nuvem que, embora não tenham tido aplicação direta neste projeto, possuem relevância para o contexto industrial real. Tomando essa ideia como sustentação, foram abordadas todas as implicações no que tange à segurança da informação, decorrentes do uso de bancos de dados em nuvem. Isto é, como a comunicação passa a ser feita via *internet*, substituindo a comunicação local, falou-se sobre o surgimento de novas questões de segurança quanto ao acesso e gerenciamento dos dados armazenados, que são fundamentais para a discussão.

Por fim, a plataforma Eclipse E3 foi introduzida, discutindo conceitos básicos e mostrando recursos fundamentais, que foram utilizados neste projeto. Com isso, mostrou-se os passos de construção do protótipo SCADA, bem como sua comunicação com o banco de dados em nuvem

criado anteriormente. Além disso, foi configurado o *WebViewer*, de forma a possibilitar o gerenciamento em rede do sistema SCADA, o que está em consonância com a adoção de um banco de dados em nuvem. Finalmente, a Figura 39 e a Figura 40 demonstraram o resultado final do trabalho e permitem concluir que os objetivos visados inicialmente foram atingidos com êxito.

Ainda que este projeto tenha ido de encontro com os objetivos, é possível realizar diversas melhorias, que podem ser empregadas em trabalhos futuros. Nesse sentido, são feitas algumas sugestões para aprofundar esse estudo e investigar possíveis avanços no protótipo:

- Abordagem e implementação de outras soluções de segurança oferecidas pela plataforma Azure, visando maior proteção tanto do tráfego de dados como de seu armazenamento;
- Avaliação de possível utilização de outra plataforma de serviços em nuvem, por exemplo a plataforma da Amazon, tendo em vista alcançar menores custos, maior segurança e eficiência;
- Extensão das configurações de rede do *WebViewer* para que seja possível operar o supervisório a partir de outros computadores conectados à *internet*, sem a exigência de estarem presentes na mesma rede do *E3 Server*;

REFERÊNCIAS BIBLIOGRÁFICAS

- BOYER, S. A. **SCADA**: Supervisory Control and Data Acquisition. 3. ed. [S.l.]: ISA, 2004.
- CEVIKCAN, E.; USTUNDAG, A. **Industry 4.0**: Managing The Digital Transformation. 1. ed. Cham: Springer International Publishing AG, 2018.
- CEZAR, T. **Cloud Computing**: Computação em Nuvem: transformando o mundo da tecnologia da informação. Rio de Janeiro: Brasport, 2009.
- CHURCH, P.; MUELLER, H.; RYAN, C.; GOGOUVITIS, S.; GOSCINSKI, A.; HAITOF, H.; TARI, Z. SCADA systems in the Cloud. *In*: CHURCH, P.; MUELLER, H.; RYAN, C.; GOGOUVITIS, S.; GOSCINSKI, A.; HAITOF, H.; TARI, Z. **Handbook of Big Data Technologies**. Cham: Springer, 2017.
- DATE, C. J. **Introdução a sistemas de bancos de dados**. 8. ed. Rio de Janeiro: Elsevier, 2013.
- ELIPSE. **Elipse E3**. [20--?]. Disponível em: <https://www.elipse.com.br/produto/elipse-e3/>. Acesso em: 15 ago. 2019.
- ELIPSE SOFTWARE. **E3 Installation Guide**. [S.l.]: Elipse Software, 2019a.
- ELIPSE SOFTWARE. **E3 User's Manual**. [S.l.]: Elipse Software, 2019b.
- FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud Computing and Grid Computing 360-Degree Compared. *In*: GRID COMPUTING ENVIRONMENTS WORKSHOP, 2008, Austin. **Proceedings [...]**. Austin: IEEE, 2009. p. 1-10.
- FOX, R.; HAO, W. **Internet Infrastructure**: Networking, Web Services, and Cloud Computing. Boca Raton: Taylor & Francis, 2017.
- FRANCIS, G. L. **SCADA**: Beginner's Guide. [S.l.: s.n], 2016.
- GAUSHELL, D. J.; DARLINGTON, H. T. Supervisory control and data acquisition. **Proceedings of the IEEE**, v. 75, n. 12, p. 1645-1658, 1987.
- JATANA, N.; PURI, S.; AHUJA, M; KATHURIA, I.; GOSAIN, D. A survey and comparison of relational and non-relational database. **International Journal of Engineering Research & Technology**, v. 1, n. 6, p. 1-5, 2012.
- KAGERMANN, H.; WAHLSTER, W.; HELBIG, J. **Securing the future of German manufacturing industry. Recommendations for Implementing the strategic initiative INDUSTRIE 4.0**: Final report of the Industrie 4.0 Working Group. [S.l.: s.n], 2013.
- FIORE, S; ALOISIO, G. **Grid and cloud database management**. Heidelberg: Springer Science & Business Media, 2011.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**: Recommendations of the National Institute of Standards and Technology. [NIST Special Publication 800-145]. [S.l.]: Forschungsunion, 2011.

MICROSOFT. **ADO Fundamentals**. 2017a. Disponível em: <https://docs.microsoft.com/en-us/sql/ado/guide/data/ado-fundamentals>. Acesso em: 12 nov. 2019.

_____. **An overview of Azure SQL Database security capabilities**. 2019a. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-security-overview>. Acesso em: 01 nov. 2019.

_____. **Azure database security overview**. 2018a. Disponível em: <https://docs.microsoft.com/pt-br/azure/security/azure-database-security-overview>. Acesso em: 15 ago. 2019.

_____. **Azure Resource Manager overview**. 2019b. Disponível em: <https://docs.microsoft.com/pt-br/azure/azure-resource-manager/resource-group-overview>. Acesso em: 01 nov. 2019.

_____. **Azure SQL Database Advanced Threat Protection for single or pooled databases**. 2019c. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-threat-detection>. Acesso em: 01 nov. 2019.

_____. **Azure SQL Database and Azure SQL Data Warehouse IP firewall rules**. 2019d. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-firewall-configure>. Acesso em: 01 nov. 2019.

_____. **Azure SQL Database Features**. 2019e. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-features>. Acesso em: 15 ago. 2019.

_____. **Azure SQL Database security features**. 2018b. Disponível em: <https://docs.microsoft.com/en-us/azure/security/fundamentals/infrastructure-sql>. Acesso em: 12 nov. 2019.

_____. **Azure SQL Database servers and their management**. 2019f. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-servers>. Acesso em: 01 nov. 2019.

_____. **Choose between the vCore and the DTU purchasing models**. 2019g. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-purchase-models>. Acesso em: 01 nov. 2019.

_____. **Contained Database Users – Making Your Database Portable**. 2019h. Disponível em: <https://docs.microsoft.com/pt-br/sql/relational-databases/security/contained-database-users-making-your-database-portable?view=sql-server-ver15>. Acesso em: 01 nov. 2019.

- MICROSOFT. **Documentação Técnica do SQL Server**. 2019i. Disponível em: <https://docs.microsoft.com/pt-br/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017>. Acesso em: 15 ago. 2019.
- _____. **Download SQL Server Management Studio (SSMS)**. 2019j. Disponível em: <https://docs.microsoft.com/pt-br/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017>. Acesso em: 15 ago. 2019.
- _____. **Elastic pools help you manage and scale multiple Azure SQL databases**. 2019k. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-elastic-pool>. Acesso em: 01 nov. 2019.
- _____. **Get started with SQL database auditing**. 2019l. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-auditing>. Acesso em: 01 nov. 2019.
- _____. **Microsoft OLE DB Provider for SQL Server Overview**. 2017b. Disponível em: <https://docs.microsoft.com/en-us/sql/ado/guide/appendixes/microsoft-ole-db-provider-for-sql-server>. Acesso em: 12 nov. 2019.
- _____. **OLE DB Providers (ADO)**. 2017c. Disponível em: <https://docs.microsoft.com/en-us/sql/ado/guide/data/ole-db-providers-ado>. Acesso em: 12 nov. 2019.
- _____. **Quickstart: Create a single database in Azure SQL Database using the Azure portal, PowerShell, and Azure CLI**. 2019m. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-single-database-get-started>. Acesso em: 01 nov. 2019.
- _____. **Quickstarts: Azure SQL Database connect and query**. 2018c. Disponível em: <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-connect-query>. Acesso em: 12 nov. 2019.
- _____. **Row-Level Security**. 2019n. Disponível em: <https://docs.microsoft.com/pt-br/sql/relational-databases/security/row-level-security?view=sql-server-ver15>. Acesso em: 01 nov. 2019.
- _____. **Service tiers in the DTU-based purchase model**. 2019o. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-service-tiers-dtu>. Acesso em: 01 nov. 2019.
- _____. **SQL Server Database Engine**. 2017d. Disponível em: <https://docs.microsoft.com/pt-br/sql/database-engine/sql-server-database-engine-overview?view=sql-server-2017>. Acesso em: 15 ago. 2019.
- _____. **What is SQL Server Management Studio (SSMS)?**. 2019p. Disponível em: <https://docs.microsoft.com/pt-br/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15>. Acesso em: 01 nov. 2019.

- MICROSOFT. **What is the Azure SQL Database service?**. 2019q. Disponível em: <https://docs.microsoft.com/pt-br/azure/sql-database/sql-database-technical-overview>. Acesso em: 15 ago. 2019.
- MORAES, C. C.; CASTRUCCI, P. L. **Engenharia de Automação Industrial**. 2. ed. Rio de Janeiro: LTC, 2012.
- OPPLIGER, R. **SSL and TLS: Theory and Practice**. Norwood: Artech House, 2009.
- PETRASCH, R.; HENTSCHKE, R. Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method. *In: INTERNATIONAL JOINT CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING (JCSEE)*, 13., 2016, Khon Kaen. **Proceedings [...]**. Khon Kaen: IEEE, 2016. p. 1-5.
- PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de Dados: Implementação em SQL, PL/SQL e Oracle 11g**. 1. ed. São Paulo: Pearson Education do Brasil, 2013.
- SAJID, A.; ABBAS, H.; SALEEM, K. Cloud-assisted IoT-based SCADA systems security: A review of the state of the art and future challenges. **IEEE Access**, v. 4, p. 1375-1384, 2016.
- STANOEVSKA-SLABEVA, K.; WOZNIAK, T.; RISTOL, S. **Grid and Cloud Computing: A Business Perspective on Technology and Applications**. [S.l.]: Springer, 2010.
- VAN DER LANS, R. F. **Introduction to SQL: Mastering the Relational Database Language**. 4. ed. Crawfordsville: Addison-Wesley Professional, 2006.
- VERAS, M. **Computação em Nuvem: Nova Arquitetura de TI**. 1. ed. Rio de Janeiro: Brasport, 2015.
- ZAMPRONHA, R. **A evolução dos sistemas supervisórios no Brasil**. [20--?]. Disponível em: <https://pt.scribd.com/document/44301887/A-Evolucao-dos-Sistemas-Supervisorios>. Acesso em: 15 ago. 2019.