PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# DESENVOLVIMENTO DE UM SISTEMA DE BACKUP E RESTAURAÇÃO COM ARMAZENAMENTO SEGURO EM NUVEM

LUCAS AUGUSTO DE SOUZA

GOIÂNIA
2025

LUCAS AUGUSTO DE SOUZA

# DESENVOLVIMENTO DE UM SISTEMA DE BACKUP E RESTAURAÇÃO COM ARMAZENAMENTO SEGURO EM NUVEM

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Orientadora:

Profª Me. Angélica da Silva Nunes

Banca examinadora:

Profº Me. Fernando Gonçalves Abadia

Profº Me. Rafael Leal Martins

GOIÂNIA

2025

LUCAS AUGUSTO DE SOUZA

# DESENVOLVIMENTO DE UM SISTEMA DE BACKUP E RESTAURAÇÃO COM ARMAZENAMENTO SEGURO EM NUVEM

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Ciência da Computação, em _____/_____/_____.

_____

Orientadora: Prof. Me. Angélica da Silva Nunes

_____

Profº Me. Fernando Gonçalves Abadia

_____

Profº Me. Rafael Leal Martins

GOIÂNIA

2025

## AGRADECIMENTOS

Agradeço primeiramente a Deus, pela minha vida e capacitação para superar cada obstáculo encontrado ao longo deste caminho.

À minha esposa, deixo um agradecimento especial pelo amor e por sua compreensão, mesmo nas minhas horas de ausência enquanto me dedicava a este trabalho, pois seu apoio foi fundamental para que eu pudesse seguir em frente com confiança e equilíbrio.

Aos meus pais, pelo apoio incondicional e presença constante nos momentos mais difíceis. Com o suporte, carinho e compreensão de ambos, que me incentivaram durante a realização deste trabalho.

Agradeço também aos professores, pelos ensinamentos transmitidos, pela dedicação e pelo compromisso. Cada orientação e cada aula contribuíram de forma fundamental para o meu desenvolvimento profissional e pessoal.

Aos amigos de trabalho, deixo meu reconhecimento pelo incentivo, pelas conversas, pelos aprendizados compartilhados e pelo apoio no início da minha trajetória profissional. A contribuição de cada um foi essencial para meu crescimento, tanto como pessoa quanto como profissional.

## RESUMO

Apresenta-se o desenvolvimento e a implementação de um sistema de backup e restauração de dados para bancos PostgreSQL, com foco na segurança da informação e na utilização de serviços em nuvem. A solução implementada contempla a geração de backups completos, sua compactação, criptografia utilizando AES-256 e validação de integridade por meio do algoritmo SHA-256. Os arquivos resultantes são enviados automaticamente ao Azure Blob Storage, seguindo uma política de backup definida pelo usuário e executada pelo agendador Cron. Também foi desenvolvida uma interface gráfica que permite a restauração de forma assistida, tanto a partir de arquivos locais quanto diretamente da nuvem. Os testes realizados demonstraram a capacidade do sistema em restaurar diferentes versões do banco de dados, incluindo a recuperação de backups gerados em ambiente Linux para uma instância PostgreSQL executando no Windows 11, evidenciando portabilidade e consistência na recuperação dos dados. Os resultados indicam que a solução proposta oferece um fluxo seguro, reproduzível e tecnicamente viável para proteção e recuperação de dados em ambientes diferentes.



Palavras-Chave: *Backup*, Restauração, Azure Blob Storage, PostgreSQL, Segurança da Informação.

## ABSTRACT

This work presents the development and evaluation of a data backup and restoration system for PostgreSQL databases, with emphasis on information security and the use of cloud services. The proposed solution includes the generation of full backups, file compression, AES-256 encryption, and integrity verification through the SHA-256 algorithm. The resulting artifacts are automatically uploaded to Azure Blob Storage according to a user-defined backup policy executed by the Cron scheduler. A graphical interface was also implemented to support assisted restoration, allowing the recovery of data from both local files and the cloud. The experiments demonstrated the system's ability to restore different versions of the database, including the recovery of backups generated on a Linux environment into a PostgreSQL instance running on Windows 11, highlighting portability and consistency in data recovery across heterogeneous environments. The results indicate that the solution provides a secure, reproducible, and technically feasible workflow for data protection and restoration.



*Keywords*: *Backup*, *Restoration*, Azure Blob Storage, PostgreSQL, *Information Security*.

## LISTA DE FIGURAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ABS | Azure Blob Storage |
| AE | *Anchor-based Evaluation* |
| AES | *Advanced Encryption Standard*, Padrão Avançado de Criptografia |
| AWS | Amazon *Web Services* |
| BFS | *Boundary Fingerprint Selection* |
| CDC | *Content-Defined Chunking* |
| CID | Confidencialidade, Integridade e Disponibilidade |
| CPU | *Central Processing Units*, Unidade Central de Processamento |
| CRM | *Customer Relationship Management,* Gestão de Relacionamento com o Cliente |
| DES | *Data Encryption Standard*, Padrão de Encriptação de Dados |
| DNS | *Domain Name System*, Sistema de Nomes de Domínio |
| EUA | Estados Unidos da América |
| FIPS | *Federal Information Processing Standard*, Padrão Federal de Processamento de Informações |
| GDPR | *General Data Protection Regulation*, Regulamento Geral sobre a Proteção de Dados |
| GRS | *Geo-Redundant Storage,* Armazenamento com Redundância Geográfica |
| HIPAA | *Health Insurance Portability and Accountability Act*, |
| I/O | *Input/Output, Leitura e Escrita* |
| IaaS | *Infrastructure as a Service,* Infraestrutura como serviço |
| IV | Vetor de Inicialização |
| LGPD | Lei Geral de Proteção de Dados |
| MD-5 | *Message-Digest Algorithm* 5 |
| MMK | *Microsoft Managed Keys,* Chaves Gerenciadas pela Microsoft |
| NAS | *Network Attached Storage*, Servidor de Armazenamento em Rede |
| NIST | *National Institute of Standards and Technology* |
| PaaS | *Plataform as a Service,* Plataforma como serviço |
| RaaS | *Rasomware as a Service*, Rasomware como serviço |

| | |
|---|---|
| RBAC | *Role-Based Access Control,* Controle de acesso baseado em função |
| RSA | Rivest-Shamir-Adleman |
| SaaS | *Software as a Service, Software como Serviço* |
| SHA | *Secure Hash Algorithm*, Algoritmo Seguro de *Hash* |
| SHA-1 | *Secure Hash Algorithm-1,* Algoritmo Seguro de *Hash*-1 |
| SHA-256 | *Secure Hash Algorithm-256,* Algoritmo Seguro de *Hash*-256 |
| SHA-512 | *Secure Hash Algorithm-512,* Algoritmo Seguro de *Hash*-512 |
| TD | *Two Divisors*, Dois Divisores |
| TI | Tecnologia da Informação |
| VMM | *Virtual Machine Monitor*, Monitor de Máquina Virtual |
| VMs | *Virtual Machines*, Máquinas Virtuais |
| VPC | *Virtual Private Cloud*, Nuvem Privada Virtual |

**SUMÁRIO**

# 1 INTRODUÇÃO

A computação em nuvem tornou-se uma solução inovadora e economicamente viável para o armazenamento e gerenciamento de dados no campo da Tecnologia da Informação. Sua adoção tem se expandido devido à oferta crescente de serviços com níveis aprimorados de segurança, escalabilidade e eficiência operacional (Opus Software, 2015).

Entretanto, a transição para a computação em nuvem não está isenta de desafios significativos, especialmente no que tange à proteção de dados e à garantia da continuidade operacional. Segundo Preston (2021), o *backup* é um aspecto importante, considerando a necessidade de manter os dados sensíveis seguros contra ameaças cibernéticas, como ataques de *Ransomware*, contra falhas técnicas e erros humanos. Além disso, a implementação de estratégias de *backup* confiáveis e eficientes é indispensável para mitigar os riscos de perda irreversível de informações, interrupções operacionais e prejuízos à imagem institucional.

O planejamento de sistemas de *backup* em nuvem exige uma abordagem abrangente e estruturada, que inclua a adoção de práticas como criptografia, políticas de *backup* e algoritmos de deduplicação de dados. Essas medidas têm como objetivo preservar a integridade e a disponibilidade das informações armazenadas, aumentando a resiliência organizacional frente a um cenário de ameaças em constante mudança.

Segundo projeções da consultoria Gartner, os investimentos globais em serviços de computação em nuvem devem ultrapassar US$ 723 bilhões até 2025, evidenciando a intensificação do uso de soluções digitais para o armazenamento e a gestão de informações sensíveis (CIO Dive, 2024).

## 1.1 Questão de pesquisa

Diante da crescente necessidade de garantir *backups* mais eficientes e seguros, como implantar um sistema de *backup* em nuvem com segurança?

## 1.2 Objetivo geral

- Implantar um sistema de *backup* em nuvem com segurança, desenvolvendo uma aplicação capaz de realizar cópias de segurança do banco de dados PostgreSQL e armazená-los de forma segura no Azure Blob Storage.

## 1.3 Objetivos específicos

- Destacar as vantagens, aplicações e desafios de sistemas de *backup*;
- Conhecer os conceitos fundamentais de segurança da informação;
- Conhecer a estrutura dos processos de *backup*, suas políticas e infraestrutura;
- Conhecer as principais abordagens e algoritmos utilizados nos sistemas de *backup*;
- Desenvolver uma aplicação capaz de gerar, criptografar, validar e enviar *backups* para a nuvem;
- Implementar um sistema de restauração, capaz de recuperar o estado do banco de dados a partir de *backups* armazenados localmente ou no Azure.

## 1.4 Metodologia

A pesquisa foi conduzida por meio de revisão bibliográfica, com base em livros especializados, artigos científicos e documentação técnica atualizada sobre os temas abordados. Foram também utilizados exemplos práticos e estudos de caso, com o intuito de ilustrar e contextualizar os conceitos discutidos.

E quanto aos procedimentos técnicos, a pesquisa é experimental, pois consiste em implementar por conta própria as tecnologias estudadas.

## 1.5 Estrutura da monografia

No Capítulo 1, são apresentados o contexto e a relevância do tema da monografia, estabelecendo a justificativa e a questão de pesquisa. São definidos o objetivo geral e os

objetivos específicos do estudo, além da metodologia utilizada para a realização da pesquisa.

O Capítulo 2 discute os fundamentos essenciais de segurança da informação, abrangendo os pilares de segurança, os mecanismos criptográficos, a verificação de integridade, os métodos de autenticação e as estratégias de controle de acesso associadas a sistemas de proteção de dados.

No Capítulo 3, é apresentada uma revisão sobre computação em nuvem, contemplando seus conceitos fundamentais, modelos de serviço, tipos de implantação, desafios de adoção e características das regiões e zonas de disponibilidade utilizadas por provedores de nuvem.

O Capítulo 4 descreve a implementação do sistema desenvolvido. São detalhadas as tecnologias empregadas, a arquitetura da aplicação, o processo de geração, compactação, criptografia e *upload* dos *backups*, bem como o fluxo de restauração baseado no Azure Blob Storage.

No Capítulo 5, são apresentados os testes realizados sobre o sistema, incluindo a execução de *backups* completos, envio para a nuvem, verificação de integridade, restauração em diferentes ambientes e análise dos resultados obtidos.

O Capítulo 6 reúne as considerações finais, sintetizando os principais resultados alcançados, discutindo as limitações identificadas e apresentando sugestões de trabalhos futuros que podem ampliar as funcionalidades do sistema.

# 2 CONCEITOS DE SEGURANÇA DE DADOS

A segurança de computadores é considerada essencial no âmbito da computação em nuvem, e é normalmente implementada através da tríade Confidencialidade, Integridade e Disponibilidade (CID). A confidencialidade engloba a não revelação de informações privadas a indivíduos não autorizados, enquanto a integridade garante que informações e programas só sejam alterados de maneira autorizada. A disponibilidade assegura que os sistemas funcionem prontamente, sem negação de serviço a usuários autorizados (Stallings, 2014).

O princípio da disponibilidade está centrado em garantir que as informações estejam acessíveis sempre que necessário, de acordo com as necessidades previamente acordadas entre as partes envolvidas. Isso não significa, necessariamente, que os dados precisam estar disponíveis em tempo integral (24 horas por dia, 7 dias por semana). Em muitos casos, o acesso pode ser restrito a períodos específicos, desde que esses intervalos tenham sido definidos com antecedência em um acordo formal. Assim, o foco está em assegurar que os recursos informacionais estejam disponíveis nos momentos previamente estabelecidos para atender às demandas operacionais ou contratuais (Brasil, 2024).

## 2.1 Criptografia

De acordo com o Google Cloud (2024), a criptografia atua como mecanismo essencial de proteção de dados, convertendo informações em códigos indecifráveis que exigem chaves digitais específicas para acesso. Essa tecnologia garante a segurança tanto de dados armazenados (em repouso) quanto durante sua transmissão (em trânsito), aplicando-se igualmente a ambientes locais e *cloud*.

A criptografia é uma técnica que converte dados compreensíveis, também chamados de texto simples, em uma forma codificada conhecida como texto cifrado. Esse processo é realizado por meio de algoritmos criptográficos, fórmulas matemáticas desenvolvidas para garantir a segurança da informação. Para que os dados criptografados possam ser acessados novamente em seu formato original, é necessário utilizar uma chave de descriptografia. Essa chave, que pode ser uma sequência de

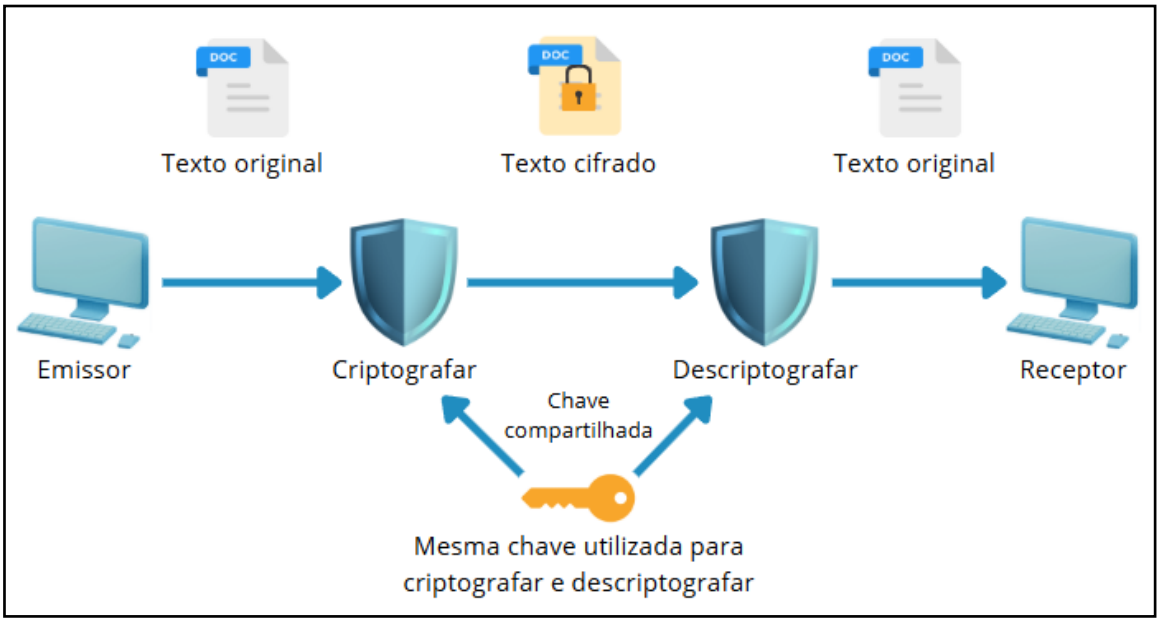
números, letras ou uma senha complexa, também é gerada por meio de algoritmos específicos e funciona como um elemento essencial para a leitura correta das informações (Google Cloud, 2024).

## 2.2 Algoritmos de criptografia

Segundo Stallings (2014), as principais técnicas de cifração incluem: criptografia simétrica, criptografia assimétrica e funções *hash*.

Uma representação da criptografia simétrica é demonstrada na Figura 1, que utiliza uma única chave para realizar tanto a cifração quanto a decifração das informações. Uma de suas maiores vantagens é a simplicidade, o que facilita sua implementação e proporciona alto desempenho no processamento de grandes quantidades de dados. A confidencialidade das informações é garantida pelo compartilhamento exclusivo da chave entre as partes autorizadas.

Figura 1: Criptografia simétrica




Fonte: Redesenhado pelo autor do trabalho, baseado em HostGator, 2025

Entre os algoritmos mais conhecidos dessa categoria estão o *Advanced Encryption Standard* – Padrão Avançado de Criptografia (AES) e o *Data Encryption Standard* – Padrão de Encriptação de Dados (DES).

Essa análise técnica do Google (2024) sobre o algoritmo AES corrobora com os princípios apresentados por Stallings (2014) em seu trabalho sobre criptografia e segurança de redes, que também enfatiza a relação custo-benefício entre esses dois sistemas de criptografia.

Desenvolvido no início do ano 1970, o DES foi oficialmente adotado pelo governo dos Estados Unidos em 1977 como um padrão de criptografia. Apesar de ter sido inovador em sua época, o DES utiliza uma chave de apenas 56 *bits*, o que, com o avanço da tecnologia e da capacidade computacional, tornou-se insuficiente para garantir a segurança dos dados, levando à sua obsolescência nos sistemas atuais. Ainda assim, o DES teve um papel importante na evolução da criptografia, influenciando o surgimento de novas abordagens e incentivando especialistas da área a desenvolverem algoritmos mais robustos e seguros.

O AES se consolidou como um dos algoritmos de criptografia mais amplamente utilizados até 2024. Conforme descrito pelo Google Cloud (2024), ele foi adotado pelo governo dos Estados Unidos em 2001 para substituir o DES e o 3DES, oferecendo maior segurança e eficiência. O AES opera com blocos de dados de 128 *bits* e permite o uso de chaves com tamanhos variados, como 128, 192 ou 256 *bits*, sendo baseado em uma estrutura conhecida como rede de substituição-permutação, que realiza múltiplas camadas de transformação nos dados para protegê-los contra ataques.

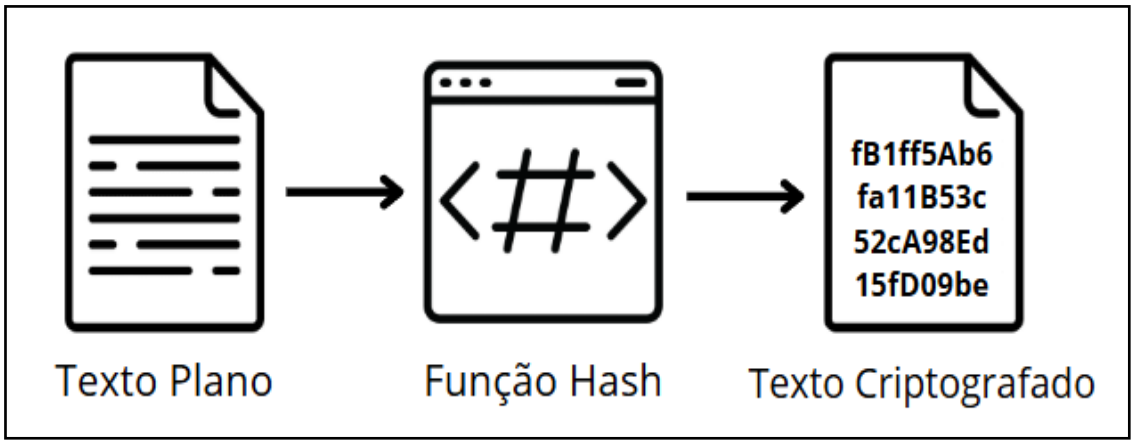
A criptografia assimétrica opera por meio da geração de um par de chaves: uma pública e outra privada. Qualquer indivíduo pode empregar a chave pública para criptografar dados. No entanto, somente os detentores da chave privada correspondente têm a capacidade de descriptografar esses dados (IBM, 2024).

Conforme o Google Cloud (2024), a criptografia simétrica utiliza uma única chave tanto para codificar quanto para decodificar informações, apresentando vantagens como menor custo computacional e maior velocidade no processamento. No entanto a criptografia assimétrica emprega um par de chaves distintas (pública e privada), o que

demanda maior capacidade de processamento e se mostra menos eficiente para grandes volumes de dados.

Uma função *hash*, conforme ilustrado na Figura 2, é um tipo de algoritmo utilizado para transformar dados de entrada, que podem ter qualquer tamanho, em uma saída compacta de tamanho fixo, normalmente representada por uma sequência curta de caracteres alfanuméricos. Esse valor resultante, chamado de *hash*, é gerado de forma determinística, ou seja, a mesma entrada sempre produzirá o mesmo resultado. Contudo, diferentemente da criptografia, o processo é irreversível, o que significa que não é possível deduzir os dados originais a partir do *hash* gerado.

Figura 2: Funções *hash* em armazenamento de senhas




Fonte: Redesenhado pelo autor do trabalho, baseado em About SSL, 2025

O objetivo principal das funções *hash* é garantir a integridade dos dados, servindo como uma espécie de "impressão digital" única para arquivos ou informações. Isso é útil em processos de autenticação de usuários ou verificação de dados, onde, por exemplo, o sistema pode armazenar apenas o *hash* de senhas ao invés das senhas propriamente ditas. Durante o login, o sistema compara o *hash* da senha fornecida pelo usuário com o valor previamente armazenado, aumentando a segurança ao não expor informações sensíveis.

Dentro da tríade CID, as funções *hash* atendem ao pilar de Integridade. Elas garantem que os dados não foram alterados ou corrompidos durante o armazenamento

ou transmissão, permitindo que se detectem mudanças não autorizadas ou erros acidentais.

A função *hash* mais utilizada em 2014 é o *Algoritmo Seguro de Hash*, *Secure Hash Algorithm* (SHA). O SHA foi desenvolvido pelo *National Institute of Standards and Technology* (NIST) e publicado como *Federal Information Processing Standard*, Padrão Federal de Processamento de Informações (FIPS 180) em 1993. Quando foram descobertas fraquezas no SHA, uma versão revisada foi publicada como FIPS 180-1 em 1995, normalmente denominada SHA-1. O SHA-1 produz um valor de *hash* de 160 *bits*. Em 2002, o NIST produziu uma versão revisada do padrão, o FIPS 180–2, que definiu três novas versões do SHA, com comprimentos de valor de *hash* de 256, 384 e 512 *bits*, conhecidas como SHA-256, SHA384 e SHA-512. Essas novas versões têm a mesma estrutura subjacente e usam os mesmos tipos de operações de aritmética modular e binárias lógicas que o SHA-1. Em 2005, o NIST anunciou a intenção de descontinuar o SHA-1 como algoritmo aprovado e passou a confiar nas outras versões do SHA em 2010 (Stallings, 2014).

## 2.3 Vulnerabilidades e principais ataques a sistemas de arquivos

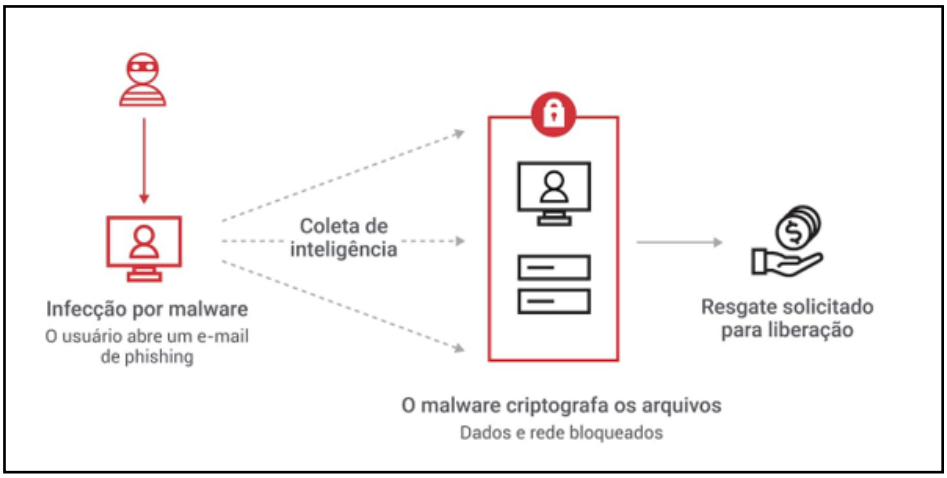
Segundo Preston (2021), vulnerabilidades são falhas ou fraquezas em sistemas, *softwares*, protocolos de segurança ou no próprio comportamento humano que podem ser exploradas por agentes maliciosos (invasores mal-intencionados), para obter acesso não autorizado, alterar, corromper ou destruir dados. Essas falhas podem ser técnicas, como *bugs* em código, portas abertas ou má configuração de servidores, ou sociais, como o erro humano, que continua sendo uma das principais portas de entrada para ameaças.

Os ataques eletrônicos representam uma das ameaças mais frequentes às organizações, superando até mesmo a exploração direta de falhas em *firewalls*. Na maioria das vezes, os ataques acontecem por meio da inserção de *malwares* no ambiente computacional interno, frequentemente facilitados por ações humanas, como clicar em *links* maliciosos em *e-mails* ou conectar dispositivos comprometidos.

O *Ransomware*, por exemplo, se dissemina silenciosamente pelos sistemas, criptografando arquivos e exigindo pagamentos em troca da chave de recuperação, conforme apresenta a Figura 3. O surgimento de serviços como o *Ransomware-as-a-*

*Service*, *Ransomware* como Serviço (RaaS) torna ainda mais grave esse cenário, já que possibilita que indivíduos sem conhecimento técnico avancem com ataques cibernéticos com facilidade, bastando contratar serviços criminosos especializados por meio da *dark web* (Preston, 2021).

Figura 3: Como funciona o *Ransomware*




Fonte: Akamai, 2022

Alguns dos impactos de ataques *Ransomware* a sistemas de arquivos em servidores são:

- Perda ou sequestro de dados críticos;
- Paralisação de operações;
- Danos a reputação da organização;
- Custos financeiros com resgate, recuperação e reforço de segurança;
- Possíveis sanções legais em função da exposição de dados pessoais.

## 2.4 Conceitos básicos de *backup*

O *backup* é uma cópia de dados armazenada separadamente do original e usada para restaurar esses dados ao seu estado anterior, geralmente após terem sido excluídos ou danificados de alguma forma (Preston, 2021).

Uma cópia é uma reprodução exata, *byte* por *byte*, do arquivo original, que mantém o conteúdo. Quando ele é realizado um *backup* completo de um arquivo, este incluirá

todos os metadados, como permissões e configurações de segurança. No entanto, nem toda cópia pode ser considerada um *backup*. Para isso, ela deve atender a outros critérios, como o armazenamento em local distinto do original (Preston, 2021).

Uma cópia que está armazenada no mesmo sistema de arquivos, computador ou banco de dados que o original, deve ser considerada apenas uma cópia de conveniência, e não um *backup* verdadeiro. Isso ocorre porque uma cópia que pode ser destruída pela mesma ação que compromete o original, não está protegendo os dados, apenas está guardada juntamente aos dados originais.

Para que algo seja reconhecido como um *backup*, ele precisa estar armazenado em um meio diferente daquele que contém o dado original. Esse meio pode ser físico ou lógico, mas o ponto central é que a separação garanta resiliência: caso um desastre (como falha de *hardware* ou ataque de *ransomware*) ocorra, ele não afetará tanto o *backup* quanto o dado original (Preston, 2021).

## 2.5 Políticas de *backup*

As políticas de *backup* e recuperação são fundamentais para garantir a continuidade dos negócios e a integridade dos dados em qualquer ambiente corporativo. Este tópico apresenta os conceitos essenciais para a implementação eficaz de estratégias de *backup*, abrangendo os níveis de *backup* e janelas de *backup*.

### 2.5.1 Níveis de *backup*

Segundo Preston (2021), existem, essencialmente, duas categorias muito amplas de níveis de *backup*, ou você está fazendo *backup* de tudo (*backup* completo) ou está fazendo *backup* apenas do que foi alterado (*backup* incremental). Esses dois tipos possuem variações que se comportam de maneira ligeiramente diferente.

O *backup* completo consiste na cópia integral de todos os dados selecionados para proteção, com exceção dos itens previamente marcados para exclusão. Isso inclui, por exemplo, todos os arquivos de um sistema de arquivos ou todos os registros de um banco de dados. Esse tipo de *backup* exige grande capacidade de *Input/Output,* Leitura e Escrita (I/O), o que pode impactar de forma considerável o desempenho do ambiente,

especialmente quando se trata de *Virtual Machines*, Máquinas Virtuais (VMs) operando como se fossem máquinas físicas e múltiplas cópias completas são realizadas simultaneamente.

O *backup* incremental é responsável por copiar apenas os dados que tiveram alterações desde a última operação de *backup*, seja ela completa ou incremental. Há diversas variações dentro dessa categoria e os fornecedores de *software* de *backup* frequentemente utilizam terminologias diferentes para diferenciá-las.

Em geral, os *backups* incrementais tradicionais funcionam com base em arquivos completos: o sistema realiza o *backup* de um arquivo inteiro se houver qualquer alteração em seu conteúdo, seja por meio da modificação da data de alteração ou pela ativação do *bit* de arquivamento no sistema operacional, como no Windows. Assim, mesmo que apenas um pequeno bloco do arquivo tenha sido alterado, o arquivo completo será copiado novamente.

No entanto, existem exceções a esse comportamento, como os *backups* incrementais em nível de bloco e os *backups* com deduplicação na origem. Esses métodos são mais eficientes, pois identificam e copiam apenas os blocos de dados que foram realmente modificados, reduzindo o volume de dados trafegados e armazenados (Preston, 2021).

Segundo Preston (2021), o uso mais comum do *backup* incremental em nível de bloco ocorre em um *Virtual Machine Monitor*, Monitor de Máquina Virtual (VMM). O VMM e suas VMs mantêm um *bitmap*, que contém um mapeamento de todos os *bits* modificados desde um determinado ponto no tempo. O *software* de *backup* pode simplesmente consultar esse *bitmap* para obter todos os *bytes* alterados desde a data especificada, e a VM responde com os resultados após essa consulta.

Como vantagem, o *backup* com deduplicação na origem exige menos entra/saída e largura de banda do que a abordagem de *backup* incremental de arquivo completo. Com o uso de discos em sistemas de *backup*, essa técnica se tornou mais comum, pois gera *backups* menores, os quais precisam ser lidos durante uma restauração.

A deduplicação na origem é descrita mais detalhadamente no Capítulo 3 deste trabalho, sendo um tipo de *backup* incremental. Trata-se de uma extensão da abordagem

de *backup* incremental em nível de bloco, onde blocos novos ou modificados passam por processamento adicional antes de serem enviados ao servidor de *backup*.

### 2.5.2 Janela de *backup*

Segundo Preston (2021), um sistema de *backup* tradicional pode causar impacto no desempenho dos sistemas principais durante sua execução. Esses sistemas realizam uma combinação de *backups* completos e incrementais, os quais, em diferentes níveis, sobrecarregam os recursos computacionais.

Diante do impacto operacional causado pelos processos tradicionais de *backup*, é necessário estabelecer um período específico para sua execução, conhecido como janela de *backup*. Essa janela corresponde ao intervalo de tempo em que os processos de cópia de segurança são autorizados a ocorrer, minimizando interferências nas atividades produtivas.

Em ambientes corporativos, essas janelas costumam ocorrer entre as 18h e as 6h nos dias úteis, sendo ampliadas durante os finais de semana. Essa prática visa aproveitar os períodos de menor utilização dos sistemas pelos usuários.

Ao definir uma janela de *backup*, deve-se monitorar constantemente a sua ocupação. Caso o tempo total de *backup* se aproxime do limite disponível, torna-se necessário reavaliar o período estabelecido ou redesenhar a arquitetura do sistema de *backup*.

## 2.6 Infraestrutura de *backup*

Os servidores físicos são servidores que operam utilizando um único sistema operacional, como Windows, Linux, OpenBSD ou diversos sistemas Unix comerciais. Isso também abrange servidores especializados, como o *Network Attached Storage*, Servidor de Armazenamento em Rede (NAS).

Os servidores podem ter diversas finalidades, como servidor *Domain Name System*, Sistema de Nomes de Domínio (DNS), NAS, servidor de aplicações ou servidor de banco de dados. Ao fazer *backup* de um servidor, é necessário considerar o *backup* completo e o incremental.

Os servidores em nuvem representam uma evolução em relação aos modelos tradicionais de infraestrutura de Tecnologia da Informação (TI). Ao contrário dos servidores físicos, que exigem espaço físico, manutenção constante e investimentos robustos em *hardware*, os servidores em nuvem são baseados em ambientes virtualizados fornecidos por *data centers* remotos. Esses servidores são acessados via *Internet* e oferecem alta flexibilidade, escalabilidade e eficiência de custos.

Na prática, um servidor em nuvem funciona como qualquer outro servidor: ele pode hospedar aplicações, bancos de dados, sistemas operacionais e arquivos. A diferença central está na forma como os recursos são provisionados e gerenciados. Com a computação em nuvem, é possível aumentar ou reduzir a capacidade de processamento, memória e armazenamento conforme a demanda, de maneira automática ou manual. Isso torna os servidores em nuvem ideais para negócios que precisam se adaptar rapidamente a picos de acesso ou que desejam evitar ociosidade de recursos.

# 3 *BACKUP* EM NUVEM

Neste capítulo, é abordado o conceito de computação em nuvem, explorando seus benefícios e os motivos para sua adoção. Além disso, será apresentada a técnica de deduplicação em *chunks*, um método eficiente para eliminar redundâncias e otimizar o armazenamento de dados. Por fim, são apresentadas soluções de mercado que implementam essas tecnologias.

## 3.1 Computação em nuvem

A computação em nuvem é utilizada para armazenar, gerenciar e processar dados como uma alternativa ao armazenamento local limitado, especialmente para *backups*. Grandes empresas adotam o armazenamento em nuvem para guardar seus dados, sendo seu uso mais comum relacionado ao *backup* de informações de clientes. No entanto, devido ao grande volume de dados exigido para *backups*, o armazenamento em nuvem enfrenta desafios como desempenho, limitação de espaço e problemas de segurança (Bhalerao; Pawar, 2017).

Diversas soluções são apresentadas para melhorar o desempenho do armazenamento em nuvem e aumentar a eficiência no uso do espaço. A deduplicação de dados é uma das técnicas promissoras para resolver esses problemas de armazenamento (Bhalerao; Pawar, 2017).

### 3.1.1 Vantagens da computação em nuvem

A computação em nuvem revolucionou a forma como as empresas gerenciam seus recursos tecnológicos, oferecendo benefícios estratégicos que impulsionam a competitividade e a eficiência operacional.

Segundo Opus Software (2015), estão entre suas principais vantagens:

- Redução de custos: elimina a necessidade de investimentos pesados em *hardware*, substituindo aquisições de servidores por um modelo de aluguel sob demanda. Isso transforma custos fixos em variáveis, permitindo que empresas direcionem recursos para outras áreas estratégicas;

- Escalabilidade flexível: ajusta automaticamente a capacidade computacional conforme a demanda, evitando desperdício com infraestrutura ociosa. O modelo *pay-as-you-go* garante que organizações paguem apenas pelos recursos utilizados, otimizando o orçamento;
- Agilidade operacional: facilita o acesso remoto a aplicações e dados, acelerando processos e permitindo respostas rápidas a mudanças de mercado. Atualizações e implantações de novas funcionalidades ocorrem de forma ágil, sem grandes interrupções;
- Resiliência e disponibilidade: com estruturas distribuídas e redundantes, provedores de nuvem garantem alta taxa de disponibilidade, minimizando riscos de indisponibilidade e perda de dados;
- Democratização tecnológica: Pequenas e médias empresas passam a competir em igualdade com grandes corporações, acessando tecnologia de ponta sem altos investimentos iniciais.

Como destacado por Opus Software (2015), a nuvem se tornou inevitável por seu papel na aceleração da inovação:

- Experimentação sem riscos: empresas podem testar novas ideias e tecnologias com baixo custo inicial, reduzindo barreiras à inovação;
- Proteção contra falhas: projetos malsucedidos não geram muitos prejuízos, já que não há infraestrutura física subutilizada;
- Alta agilidade em casos de sucesso: iniciativas bem-sucedidas podem ser rapidamente ampliadas, aproveitando a elasticidade da nuvem;
- Economia de Escala e concorrência: o crescimento do mercado de nuvem beneficia tanto provedores quanto clientes.

Para provedores, o aumento da base de usuários permite negociações mais vantajosas em aquisições de hardware e energia, reduzindo custos operacionais. Para os clientes, a concorrência entre fornecedores resulta em preços mais acessíveis e serviços otimizados, criando um ciclo virtuoso de melhoria contínua e custos decrescentes.

### 3.1.2 Dificuldades de implantação

Apesar dos inúmeros benefícios, a migração para a computação em nuvem ainda enfrenta obstáculos significativos que retardam sua adoção plena. Conforme apontado por Opus Software (2015), os principais desafios incluem:

- Integração com sistemas legados: muitas organizações possuem infraestruturas de TI antigas, e a compatibilidade entre esses sistemas e os ambientes de nuvem pode ser complexa, exigindo soluções personalizadas e custos adicionais;
- Conformidade legal e regulatória: as exigências variam conforme setores e jurisdições, especialmente em áreas sensíveis como finanças e saúde. A adequação a normas como Lei Geral de Proteção de Dados Pessoais (LGPD), *General Data Protection Regulation*, Regulamento Geral sobre a Proteção de Dados (GDPR) ou *Health Insurance Portability and Accountability Act*, Lei de Portabilidade e Responsabilidade de Seguros de Saúde (HIPAA) torna-se um requisito crítico, porém complexo;
- Falta de clareza conceitual: o termo "nuvem" ainda é abstrato para muitos tomadores de decisão, levando a equívocos sobre suas capacidades e limitações. Essa ambiguidade gera resistência ou expectativas irreais;
- Preocupações com segurança: a delegação do armazenamento e processamento de dados a terceiros cria receios quanto a vazamentos, acesso não autorizado ou perda de controle. Apesar dos avanços em criptografia e certificações, a percepção de risco persiste;
- Resistência organizacional: a resistência dos gestores e líderes de uma organização pode ser um obstáculo na implementação da computação em nuvem. A transição para a nuvem pode em muitas vezes significar perda de poderes para um gestor, o que leva a uma postura conservadora em relação à adoção da nuvem;
- Dependência de conectividade: a eficiência da nuvem está diretamente vinculada à qualidade da conexão de Internet. Em regiões com infraestrutura

digital precária ou para aplicações que demandam baixa latência, essa limitação pode inviabilizar o modelo;

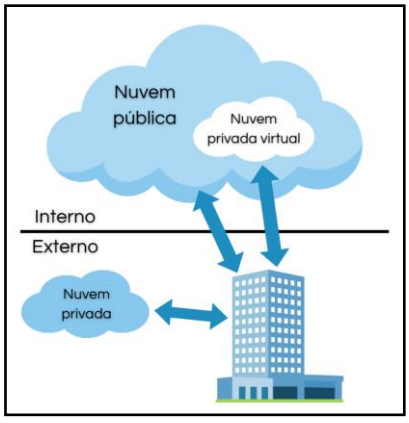
- A complexidade técnica: a transição requer um planejamento detalhado, que inclui a escolha do modelo (público, privado ou híbrido) e a migração de dados. Este processo demanda um nível de expertise que frequentemente é raro nas equipes internas;
- Custos enterrados e transição gradual: empresas com investimentos consolidados em *Data Centers* locais enfrentam o dilema de abandonar ativos subutilizados. Além disso, os custos iniciais de migração (treinamento, adaptação de processos e possíveis paralisações) podem ofuscar os ganhos de longo prazo.

### 3.1.3 Soluções de mercado

A computação em nuvem revolucionou a maneira como empresas e indivíduos utilizam recursos tecnológicos, oferecendo escalabilidade, flexibilidade e redução de custos. Com diferentes modelos de implantação disponíveis no mercado, cada um atende a necessidades específicas, desde infraestrutura básica até soluções completas de *software*.

De acordo com Opus Software (2015), os tipos de nuvem geralmente propostos são: nuvem pública, nuvem privada e nuvem híbrida, conforme ilustrado na Figura 4.

Figura 4: Tipos de nuvem




Fonte: Redesenhado pelo autor do trabalho, baseado em Opus Software, 2015

A nuvem pública é oferecida pela *Internet*, por um provedor de serviços, nos quais os recursos computacionais são distribuídos para seus diversos clientes e o controle das instâncias, máquinas virtuais e recursos de processamento e armazenamento ficam completamente delegados ao provedor.

Na nuvem privada, os recursos computacionais são dedicados a uma determinada organização, estão isolados dos utilizados por outras empresas. Os recursos computacionais e equipamentos podem estar alocados tanto fisicamente dentro da organização, como também podem ser configurados em um provedor público.

O termo “nuvem privada” pode significar duas coisas:

- Uma nuvem é criada internamente dentro de uma organização, na qual toda a infraestrutura é controlada e é utilizada localmente pela própria empresa;
- Em uma *Virtual Private Cloud*, Nuvem Privada Virtual (VPC), a infraestrutura é controlada por um provedor de serviços, mas os recursos alocados para uma organização estão isolados dos recursos compartilhados pela nuvem pública.

A nuvem hibrida é constituída de uma junção de serviços de nuvem pública e privada, permitindo que a organização mantenha algumas aplicações na nuvem privada da empresa, e outras em nuvem pública ou privada virtual.

Em teoria, uma nuvem privada pode oferecer um nível elevado de segurança em comparação com uma nuvem pública, pois o tráfego de dados e a migração entre servidores físicos e virtuais estão restritos à infraestrutura controlada diretamente pela empresa. No entanto, a segurança de um sistema depende de diversos fatores, como a arquitetura de rede, políticas de acesso, mecanismos de criptografia e práticas de governança de dados, que devem ser implementados independentemente do tipo de nuvem escolhido.

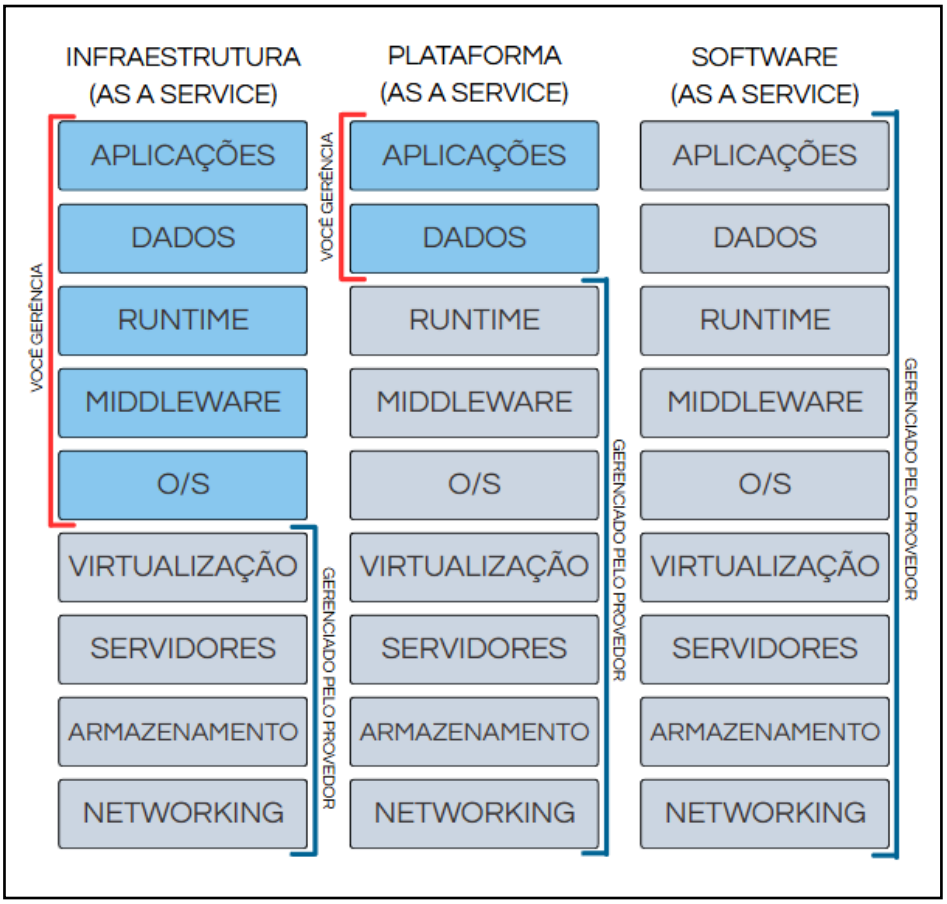
### 3.1.4 Tipos de serviço

De acordo com Opus Software (2015), os modelos de serviço em nuvem podem ser classificados em três tipos, conforme ilustrado na Figura 5.

- *Infrastructure as a Service*, Infraestrutura como serviço (IaaS);
- *Plataform as a Service*, Plataforma como Serviço (PaaS);

- *Software as a Service*, Software como Serviço (SaaS).

Figura 5: Modelos de serviços




Fonte: Redesenhado pelo autor do trabalho, baseado em Opus Software, 2015

A IaaS consiste na disponibilização, ao usuário, de um conjunto de recursos computacionais fundamentais, como processamento, armazenamento e redes. Sobre essa infraestrutura, é possível instalar e executar qualquer tipo de *software*, incluindo sistemas operacionais e aplicações diversas. Apesar de a infraestrutura física permanecer oculta ao usuário, ele possui total controle sobre os sistemas operacionais, o espaço de armazenamento e os softwares que utiliza. Exemplos representativos desse modelo são o *Amazon Web Services* (AWS), o *Google Compute Engine* e o Microsoft Azure. Conforme descrito por Opus Software (2015), essa camada de serviço oferece uma base flexível e escalável para diferentes tipos de soluções tecnológicas.

No modelo PaaS, o usuário tem a capacidade de desenvolver, instalar e gerenciar suas próprias aplicações, que podem ser criadas internamente ou adquiridas de terceiros. Para isso, utiliza as ferramentas, bibliotecas e ambientes oferecidos pelo provedor da plataforma. As aplicações desenvolvidas nesse contexto são geralmente construídas para funcionar especificamente na plataforma escolhida. Por exemplo, uma aplicação escrita em Python para o *Google App Engine* pode necessitar de adaptações para serem executadas no Microsoft Azure *Cloud Services*, mesmo que ambas suportem a mesma linguagem. A grande vantagem do PaaS, como destacado por Opus Software (2015), é que ele elimina a necessidade de aquisição, configuração e administração de infraestrutura física ou lógica, permitindo que o desenvolvedor foque apenas na aplicação. Embora a infraestrutura continue invisível, o usuário pode ajustar as configurações das aplicações e, em alguns casos, elementos do ambiente de execução.

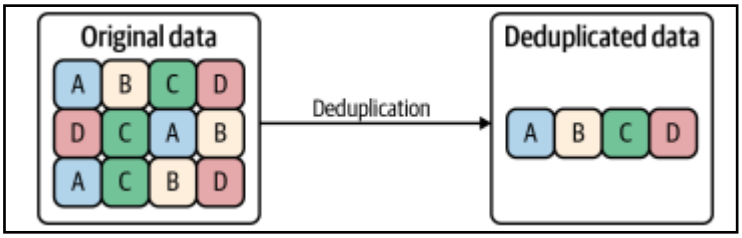
No modelo SaaS, o usuário final acessa diretamente um *software* disponibilizado pelo provedor, o qual é executado em uma infraestrutura baseada na nuvem. Toda a complexidade relacionada ao gerenciamento de recursos como servidores, redes, sistemas operacionais e armazenamento é responsabilidade do provedor. A infraestrutura subjacente permanece completamente invisível ao usuário, que apenas utiliza o *software* como uma ferramenta pronta para uso. Um bom exemplo desse modelo é o Google *Apps for Work*, que permite a criação e manipulação de documentos, planilhas e apresentações diretamente nos servidores do provedor. Outros exemplos relevantes incluem o Microsoft Office 365 e a plataforma de *Customer Relationship Management*, Gestão de Relacionamento com o Cliente (CRM) da Salesforce.com. Segundo Opus Software (2015), o SaaS representa uma das formas mais maduras e difundidas de computação em nuvem, oferecendo agilidade e redução de custos operacionais para os usuários.

## 3.2 Deduplicação de *backup*

A deduplicação consiste na identificação e remoção de dados redundantes dentro de um conjunto de dados, que podem incluir múltiplos *backups* realizados em diversos locais ao longo do tempo. A Figura 6 ilustra uma representação básica desse processo,

evidenciando uma redução de 66% no número de blocos necessários para armazenar os dados com a aplicação da deduplicação. Esse método é capaz de eliminar dados duplicados entre diferentes arquivos, mesmo quando armazenados em diretórios distintos. Além disso, alguns sistemas avançados de deduplicação têm a capacidade de identificar dados duplicados entre múltiplos hosts e data centers (Preston, 2021).

Figura 6: Deduplicação de dados




Fonte: Preston, 2021

As etapas no processo de deduplicação são as seguintes:

- *Chunking*: na etapa de *chunking*, o fluxo de dados de entrada é dividido em vários blocos usando um algoritmo de segmentação;
- *Fingerprinting*: nesta etapa, valores *hash* para cada bloco são gerados utilizando funções *hash* como SHA-1 ou *Message-Digest Algorithm* 5 (MD-5). Esses valores *hash* são chamados de impressões digitais dos blocos;
- Indexação: na indexação, comparam-se os valores *hash* previamente armazenados com os recém-gerados, a fim de identificar blocos de dados duplicados. Se dois ou mais blocos tiverem impressões digitais idênticas, eles são considerados duplicados, e apenas os blocos únicos são armazenados;
- Gravação: é o armazenamento da cópia única dos dados na mídia de armazenamento.

### 3.2.1 Métricas de avaliação

Diversas soluções têm sido propostas para melhorar o desempenho do armazenamento em nuvem e aumentar a eficiência no uso do espaço. A deduplicação

de dados é uma das técnicas promissoras para resolver esses problemas de armazenamento.

Segundo Bhalerao e Pawar (2017), as principais métricas para medir a eficiência da deduplicação são:

- Eficiência de deduplicação: o algoritmo de fragmentação deve levar menos tempo para gerar os fragmentos. A fragmentação de tamanho variável é mais eficiente em comparação com a fragmentação de tamanho fixo;
- Vazão da deduplicação: é a taxa na qual os fragmentos são gerados com sucesso. Um algoritmo de fragmentação ideal deve ter alta vazão, o que significa que ele deve gerar os fragmentos em uma taxa mais rápida;
- Sobrecarga computacional: para obter melhor eficiência de deduplicação, a sobrecarga computacional idealmente deve ser baixa. Quando o fluxo de dados é processado no mecanismo de deduplicação, cada *byte* deve ser verificado para encontrar os limites dos fragmentos. Isso leva a um uso enorme dos recursos da *Central Processing Units*, Unidade Central de Processamento (CPU), o que causa alta sobrecarga computacional.
- Variação no tamanho dos fragmentos: indica a variação no tamanho dos fragmentos. O tamanho dos fragmentos não deve ser muito pequeno nem muito grande. Essa variação afeta a eficiência da deduplicação, o que leva à baixa vazão da fragmentação. Quanto menor a variação no tamanho dos fragmentos, melhor será a eficiência da deduplicação;
- Cadeia de baixa entropia: em algumas cadeias é difícil encontrar seus limites de fragmentação. Essas cadeias podem ter os mesmos caracteres repetidamente, o que torna difícil identificar os limites dos fragmentos. O algoritmo de fragmentação deve ser capaz de detectar essas cadeias de baixa entropia.

### 3.2.2 Principais algoritmos de *chunking*

Diversos algoritmos de fragmentação são utilizados e discutidos em estudos e no mercado. A seguir são abordados alguns algoritmos de *chunking*: o algoritmo de

impressão digital de Rabin, o algoritmo *Two Divisors*, Dois Divisores (TD), o algoritmo MAXP, o algoritmo Bimodal e o algoritmo *Anchor-based Evaluation* (AE) (Bhalerao; Pawar, 2017).

O algoritmo de impressão digital de Rabin, proposto por Michael O. Rabin, usa polinômios sobre um campo finito para criar impressões digitais. É empregado no algoritmo *Content-Defined Chunking* (CDC), mas sua baixa vazão de fragmentação prejudica a eficiência da deduplicação.

O algoritmo TD tenta dividir fragmentos grandes usando um divisor secundário menor, com maior probabilidade de encontrar impressões digitais duplicadas. Este método é empregado para resolver o problema de deslocamento de fronteira *Boundary Fingerprint Selection* (BFS), que divide grandes fragmentos em tamanhos fixos. O TD utiliza um divisor secundário para identificar pontos de ruptura. Se o primeiro divisor não conseguir encontrar um ponto dentro de um valor limite definido, chamado de Tmax, o TD começa a escanear o fluxo de dados a partir do último ponto de ruptura e gera impressões digitais para cada fragmento. Ele verifica a coincidência de impressões digitais em ambos os divisores. Se o primeiro divisor localizar uma correspondência antes do valor limite, ele declara um ponto de ruptura nessa posição. Se não conseguir, tenta com o divisor secundário. Caso ambos falhem, a posição atual é declarada como ponto de ruptura.

O algoritmo MAXP supera o problema de variação no tamanho dos fragmentos do algoritmo de Rabin tratando valores extremos locais. É utilizado principalmente em sistemas de armazenamento em rede. Ele divide janelas com base em cada *byte*. O MAXP encontra o valor extremo local para definir o limite de fragmento e reanalisa todos os *bytes* anteriores na direção reversa, o que afeta a vazão da fragmentação devido ao aumento na sobrecarga de comparação.

O algoritmo Bimodal inicialmente divide o fluxo de dados em grandes fragmentos e posteriormente os subdivide em fragmentos menores. Dois princípios são aplicados para eliminar duplicidades nos fragmentos maiores. O primeiro princípio estabelece que o algoritmo deve gerar fragmentos com tamanho médio maior em extensas regiões de dados não detectados anteriormente. Dados estão em uma região de mudança se alguns setores existirem tanto em fragmentos anteriores quanto em novos. A variação no

tamanho da região e a forma como os dados não vistos são fragmentados resultam em diferentes variantes do algoritmo bimodal. O segundo princípio requer fragmentos pequenos para resolver o problema de deslocamento de fronteira.

O algoritmo AE utiliza uma janela assimétrica de tamanho variável para superar o problema de deslocamento de fronteira. Ele não faz retrocesso e, portanto, requer apenas uma comparação; por isso, é muito rápido e apresenta menor variação no tamanho dos fragmentos (Bhalerao; Pawar, 2017).

# 4 IMPLEMENTAÇÃO DE SISTEMA DE *BACKUP* AUTOMATIZADO

A escolha da plataforma Microsoft Azure fundamenta-se nas suas características de segurança e nas possibilidades de implementação em ambientes de nuvem híbrida ou pública. Conforme a documentação da Microsoft, o Azure oferece vantagens de segurança e monitoramento, que são disponibilizados de forma econômica à organização e ajudam a proteger plataformas híbridas, aplicativos e dados (Microsoft, 2025).

## 4.1 Serviço de armazenamento utilizado

Entre as opções de armazenamento disponibilizadas pela plataforma Azure, o presente trabalho faz uso do *Azure Blob Storage* (ABS), por tratar-se de um serviço projetado para o armazenamento de dados não estruturados em formato binário, como arquivos de *backup* e *logs* de execução. De acordo com a Microsoft (2025a), o ABS é otimizado para manipular grandes volumes de dados e oferece integração nativa com linguagens e *frameworks* modernos, incluindo .NET, utilizado no desenvolvimento deste sistema.

A escolha por esse serviço fundamenta-se em três aspectos principais. O primeiro é a compatibilidade com arquivos gerados por aplicações externas, como os *dumps* de banco de dados produzidos pelo PostgreSQL, que podem ser enviados diretamente para o contêiner de armazenamento na nuvem. O segundo aspecto é o suporte a operações via API REST e bibliotecas oficiais da Microsoft para .NET, o que permite a automação completa do processo de *upload*, *download* e verificação de integridade dos arquivos, sem necessidade de intervenção manual. Por fim, destaca-se o suporte a mecanismos de criptografia em repouso e em trânsito, atendendo às práticas recomendadas de segurança para ambientes corporativos.

O armazenamento dos arquivos de *backup* foi configurado utilizando o serviço, na camada de acesso do tipo *Archive Tier* (Arquivo). De acordo com a Microsoft (2025a), o ABS oferece três camadas distintas de acesso: *Hot* (Quente), *Cool* (Fria) e *Archive* (Arquivo), sendo cada uma delas voltada a um padrão de uso e custo específico.

A camada *Hot* é destinada a dados acessados com frequência, apresentando menor latência, porém custo de armazenamento mais elevado. Já a camada *Cool* oferece

um equilíbrio entre custo e desempenho, sendo recomendada para dados que permanecem armazenados por períodos prolongados, mas ainda requerem acesso ocasional.

Por fim, a camada *Archive*, utilizada neste trabalho, é voltada para o armazenamento de longo prazo de dados raramente acessados, como *backups* históricos e registros de auditoria. Nessa camada, o custo de armazenamento é reduzido, pois os dados são mantidos em estado de arquivamento, podendo ser reidratados (reativados) quando necessário.

Essa configuração mostrou-se mais adequada ao contexto da aplicação, uma vez que os *backups* de bancos de dados são consultados com baixa frequência, apenas em casos de restauração ou verificação de integridade. Dessa forma, a utilização da camada *Archive* reduz o custo de armazenamento e mantém a integridade e disponibilidade dos dados conforme as necessidades do sistema.

## 4.2 Objetivo da implementação

O objetivo deste trabalho, é desenvolver um sistema de *backup* automatizado e seguro, com integração ao serviço de armazenamento em nuvem, visando à proteção e à disponibilidade dos dados em ambientes corporativos.

O sistema foi projetado para realizar cópias de segurança completas e incrementais de bancos de dados, permitindo o armazenamento local e remoto, com suporte a criptografia e autenticação. O projeto contempla ainda a automação de rotinas de *backup* por meio do agendador de tarefas do sistema operacional Linux, garantindo a execução periódica e confiável das operações.

A implementação adota o modelo de computação em nuvem no contexto de IaaS, conforme descrito por Opus Software (2015), aproveitando os benefícios de escalabilidade e resiliência desse paradigma. A segurança dos dados em trânsito e em repouso é assegurada por mecanismos de criptografia simétrica AES, seguindo as recomendações de Stallings (2014), e pela aplicação de políticas de controle de acesso inspiradas no modelo *Role-Based Access Control* (RBAC).

Além de atender aos requisitos técnicos de segurança e confiabilidade, a implementação visa demonstrar, em ambiente real, a viabilidade da integração entre

infraestrutura local e serviços de nuvem para fins de *backup* corporativo, conforme preconizado por Kavis (2014) e Preston (2021), no contexto da continuidade de negócios e da proteção contra falhas operacionais e incidentes de segurança.

### 4.3 Ambiente de Testes

Os testes e validações do sistema foram realizados em ambiente controlado, configurado especificamente para avaliar o desempenho e a estabilidade do aplicativo de *backup* em condições reais de operação. O ambiente de *hardware* utilizado consistiu em uma plataforma baseada em uma placa-mãe H610M com suporte a memórias DDR5, equipada com 32 GB de RAM DDR5 operando a 4.800 MHz e um processador Intel Core i5-14400F com 16 núcleos e frequência máxima de 4,7 GHz. O subsistema gráfico foi composto por uma GPU AMD Radeon RX 9060 XT, com 16 GB de memória VRAM dedicada, e o armazenamento principal foi provido por uma unidade SSD NVMe M.2 de 1 TB, destinada tanto ao sistema operacional quanto aos dados de *backup*.

O sistema operacional utilizado foi o Ubuntu 25.10 (Questing), com kernel Linux 6.17.0-6-generic e ambiente gráfico GNOME 49.0. A distribuição foi escolhida por oferecer suporte nativo a ferramentas de automação como o Cron, utilizadas em ambientes Unix-like para o agendamento de tarefas recorrentes.

O banco de dados utilizado foi o PostgreSQL 17.6, configurado localmente para simular o ambiente de produção. Essa versão foi selecionada por incorporar melhorias de desempenho e segurança e oferecer suporte nativo às ferramentas de exportação e restauração de dados, essenciais para o funcionamento do sistema desenvolvido (PostgreSQL, 2025).

Todo o processo de *backup* e restauração foi executado sob um perfil de usuário comum, garantindo que os testes representassem um cenário operacional típico, sem privilégios administrativos diretos. A integração com o ABS foi utilizada para a etapa de armazenamento em nuvem, validando o envio, a criptografia e a integridade dos arquivos de *backup*.

## 4.4 Tecnologias utilizadas

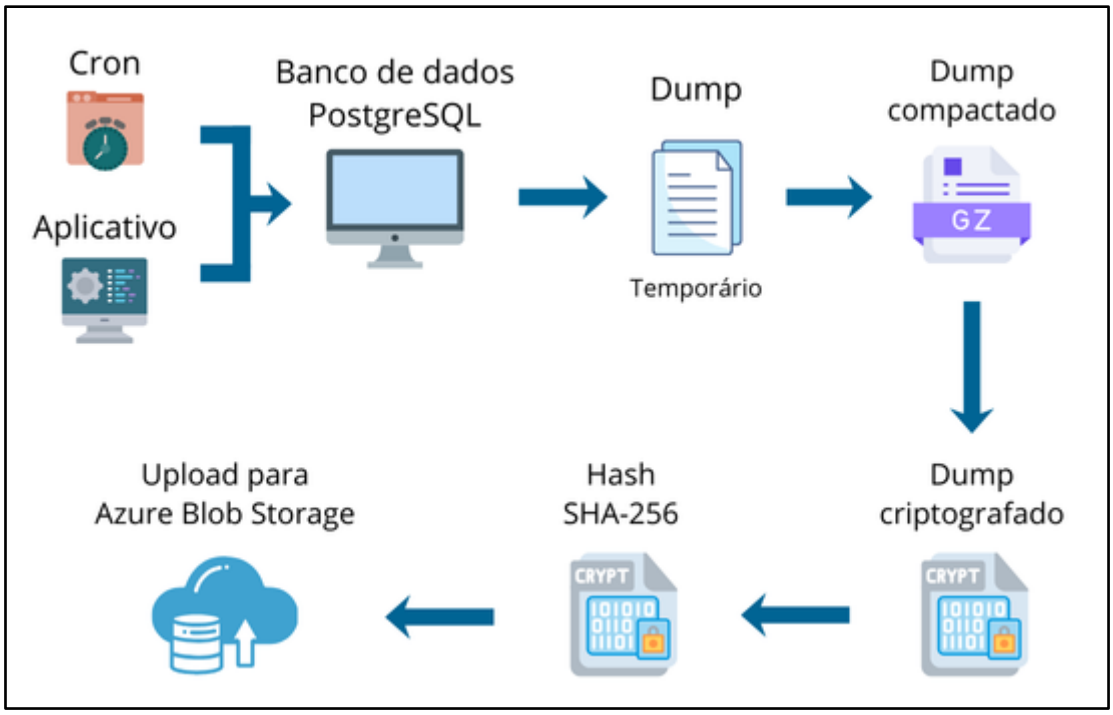
Para o desenvolvimento do sistema proposto foi adotada a plataforma .NET, em sua versão 9.0, mantida e distribuída pela Microsoft. A escolha fundamenta-se na sua capacidade de execução multiplataforma, que permite a criação de aplicações compatíveis com sistemas Windows e Linux. De acordo com a documentação oficial da Microsoft, o .NET possui um ciclo contínuo de atualização que aprimora aspectos de desempenho, segurança e interoperabilidade, além de oferecer uma extensa base de bibliotecas e ferramentas integradas (Microsoft, 2025c).

A utilização do .NET também favorece a integração com serviços corporativos e de nuvem, como o Azure, possibilitando a implementação de soluções seguras e escaláveis. Outro ponto relevante é o suporte ativo da comunidade e da própria Microsoft, que contribui para a manutenção de um ecossistema estável e bem documentado, facilitando a evolução do projeto e a portabilidade entre diferentes ambientes de execução.

Para a camada de *interface* gráfica, foi adotado o Avalonia UI, um *framework* de código aberto projetado para o desenvolvimento de *interfaces* modernas e multiplataforma. O Avalonia UI permite a construção de aplicações com *design* responsivo e consistente em diferentes sistemas operacionais, eliminando a necessidade de reescrita de código para cada plataforma. Segundo a própria documentação do projeto (Avalonia, 2024), o *framework* oferece suporte nativo a estilos modernos, carregamento rápido e arquitetura modular, o que contribui para a criação de aplicações com boa usabilidade e baixo consumo de recursos.

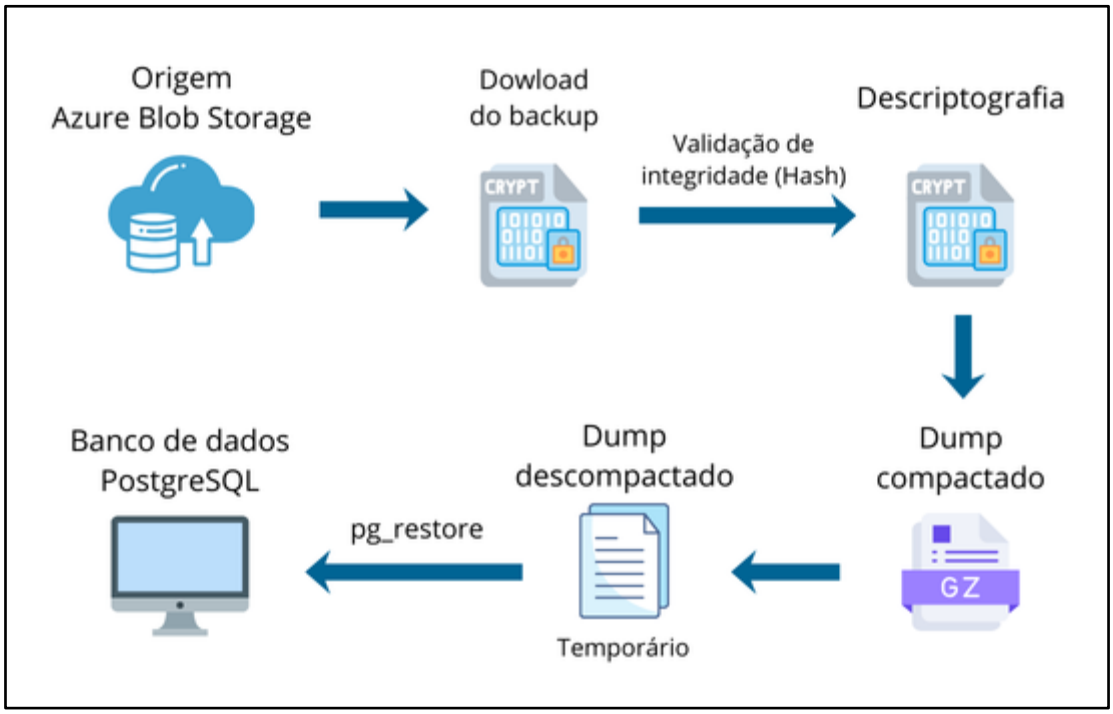
A combinação entre .NET e Avalonia UI viabiliza o desenvolvimento de uma aplicação com arquitetura unificada, capaz de integrar-se de forma eficiente aos serviços em nuvem e aos ambientes locais, preservando a compatibilidade e a segurança em todas as etapas do ciclo de execução. Essa integração é especialmente visível nos dois fluxos principais do sistema: o fluxo de backup e o fluxo de restauração, representados nas Figuras 7 e 8.

Figura 7: Fluxo de *backup*




Fonte: Desenhado pelo autor desse trabalho em Canva, 2025.

Figura 8: Fluxo de *restore*




Fonte: Desenhado pelo autor desse trabalho em Canva, 2025.

A Figura 7 apresenta o fluxo completo do processo de *backup*, no qual o aplicativo, ou agendador Cron, solicita a geração do *dump* do banco de dados, que é compactado, criptografado e então é realizado o cálculo do *hash* SHA-256. Em seguida, o artefato resultante é enviado ao Azure Blob Storage, onde fica armazenado. Esse fluxo demonstra a integração entre as tecnologias utilizadas e o serviço de nuvem do Azure.

A Figura 8 ilustra o fluxo de restauração. Nesse processo, o usuário seleciona um arquivo de origem do Azure. Após a validação de integridade por meio do *hash*, o arquivo é descriptografado e descompactado, resultando no *dump* a ser restaurado no banco de destino via pg_restore. Essa representação demonstra a integração entre ambiente distintos, uma vez que o processo de *restore* pode ocorrer tanto em sistemas Linux quanto Windows, preservando a consistência dos dados e a compatibilidade operacional.

## 4.5 Implementação do aplicativo de gerenciamento de *backups*

Este tópico descreve detalhadamente as etapas de implementação do sistema de gerenciamento de *backups* desenvolvido neste trabalho. O aplicativo foi estruturado para integrar as operações locais de cópia de segurança de bancos de dados PostgreSQL ao serviço de armazenamento em nuvem ABS, automatizando processos, assegurando a integridade dos dados e reduzindo a necessidade de intervenção manual.

As etapas a seguir abordam desde a configuração inicial da conta na nuvem até o processo de restauração dos dados, evidenciando as principais decisões técnicas e os componentes de *software* envolvidos.

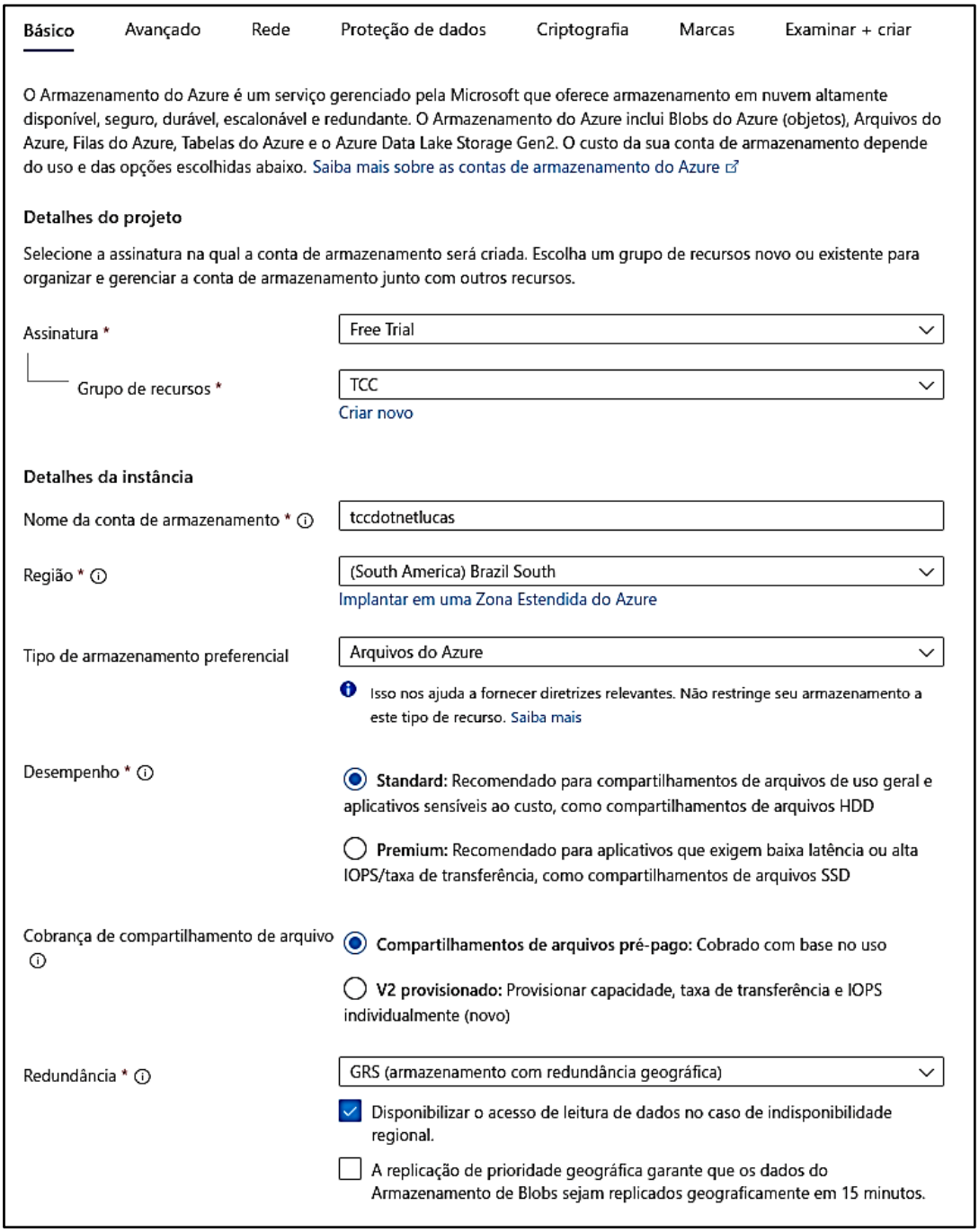
### 4.5.1 Criação da conta

O primeiro passo da implementação consistiu na criação da conta na plataforma Azure, necessária para a utilização do serviço ABS.

Durante essa etapa, foi criada uma *Storage Account*, Conta de Armazenamento, com o tipo de desempenho padrão e replicação *Geo-Redundant Storage,* Armazenamento com Redundância Geográfica (GRS), garantindo que os dados fossem replicados entre diferentes regiões físicas do *datacenter*. Essa configuração proporciona maior disponibilidade e resiliência dos *backups* corporativos, assegurando a continuidade

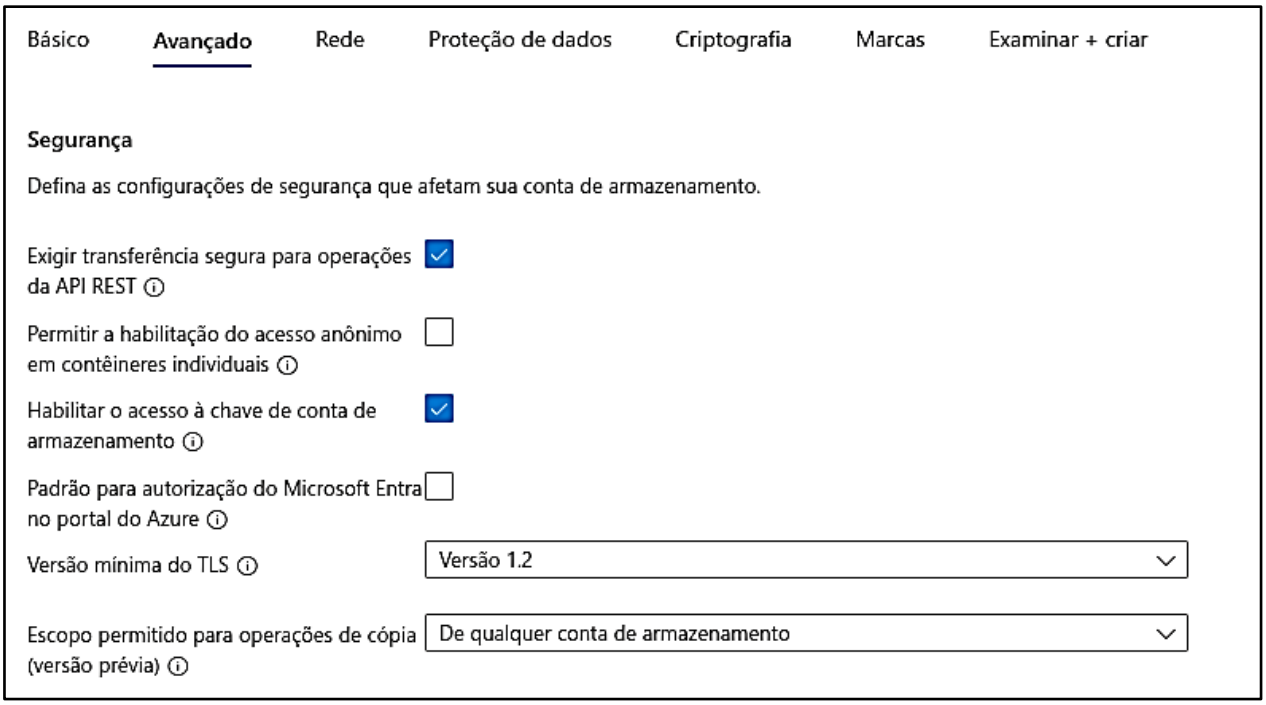
das operações mesmo em caso de falhas regionais, configurações ilustradas nas Figuras 9, 10 e 11.

Figura 9: Configuração do Azure Blob Storage



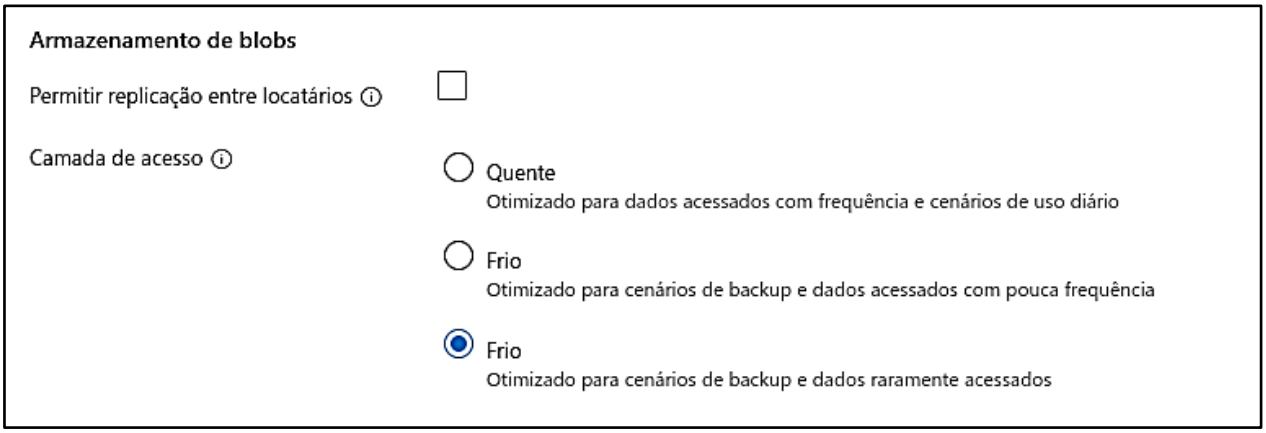

Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

Figura 10: Configurações de segurança




Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

Figura 11: Configuração do armazenamento de blobs

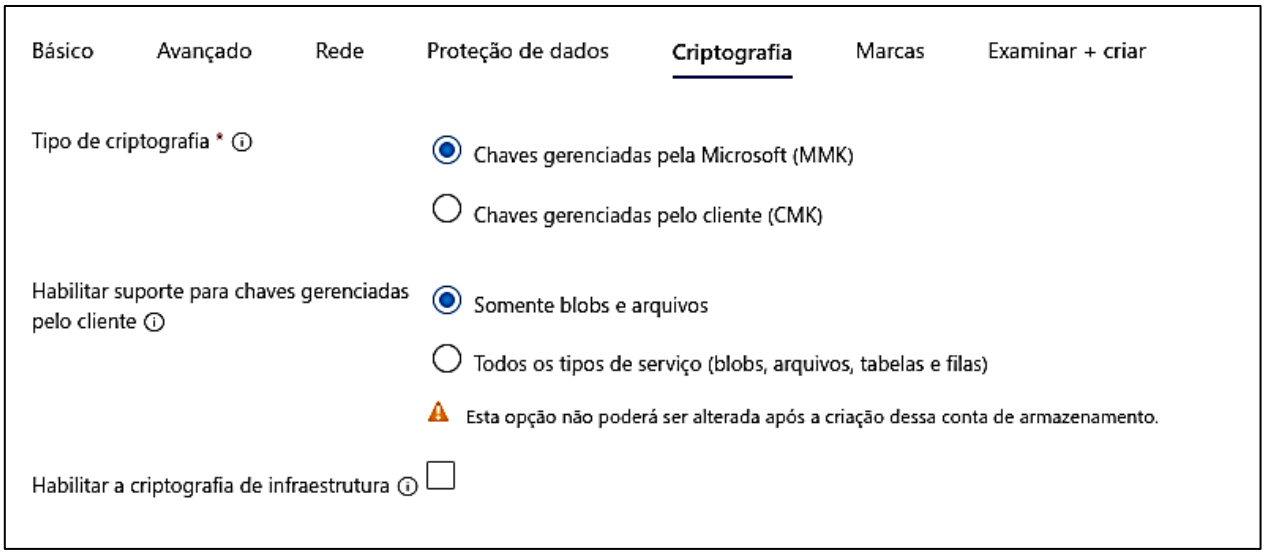



Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

Foi habilitada a criptografia em repouso (*Encryption at Rest*) utilizando o modelo *Microsoft Managed Keys,* Chaves Gerenciadas pela Microsoft (MMK). Nesse modelo, a própria plataforma Azure é responsável pela geração, rotação e proteção das chaves criptográficas utilizadas para assegurar a confidencialidade dos dados armazenados.

Também foi ativado o suporte para criptografia apenas de *Blobs* e Arquivos (*Blobs and Files*), abrangendo especificamente os dados enviados como *dumps* e compactações de *backup*, conforme ilustrado na Figura 12.

Figura 12: Configuração de criptografia no Blob Storage




Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

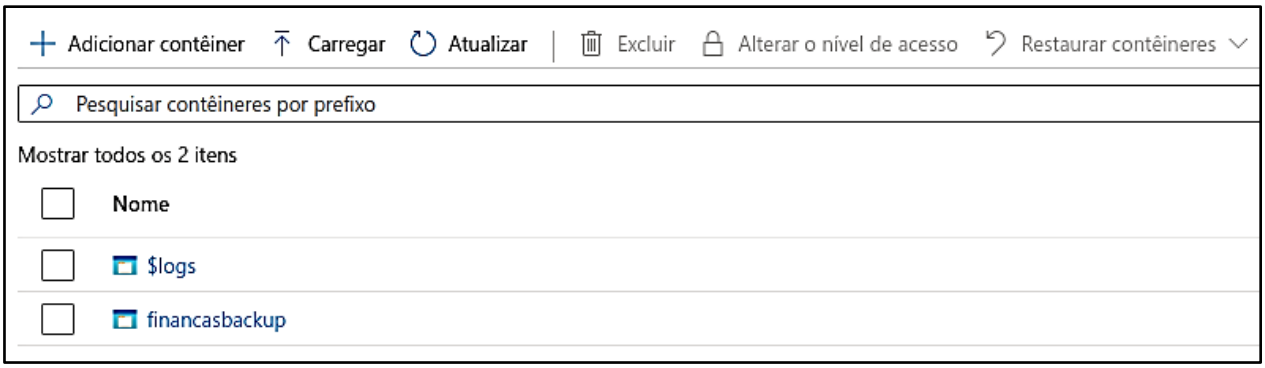
Embora o serviço ABS ofereça criptografia em repouso por padrão, gerenciada pela Microsoft, este trabalho adota uma camada adicional de segurança por meio da criptografia local com o AES.

Essa abordagem garante que os dados de *backup* sejam protegidos antes mesmo do envio à nuvem, assegurando a confidencialidade ponta a ponta e mitigando riscos de exposição durante o transporte ou em caso de comprometimento da conta de armazenamento.

A aplicação do algoritmo AES é complementada pela geração de um *hash* SHA-256, utilizado para verificação da integridade dos arquivos após o *upload*. Essa dupla proteção segue as boas práticas recomendadas por Stallings (2014) e pelos padrões de segurança corporativa, assegurando que os dados armazenados mantenham sua integridade e sigilo em todas as etapas do processo.

Em seguida, um contêiner destinado ao armazenamento dos *backups* criptografados é criado, conforme ilustrado na Figura 13.

Figura 13: Contêineres configurados




Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

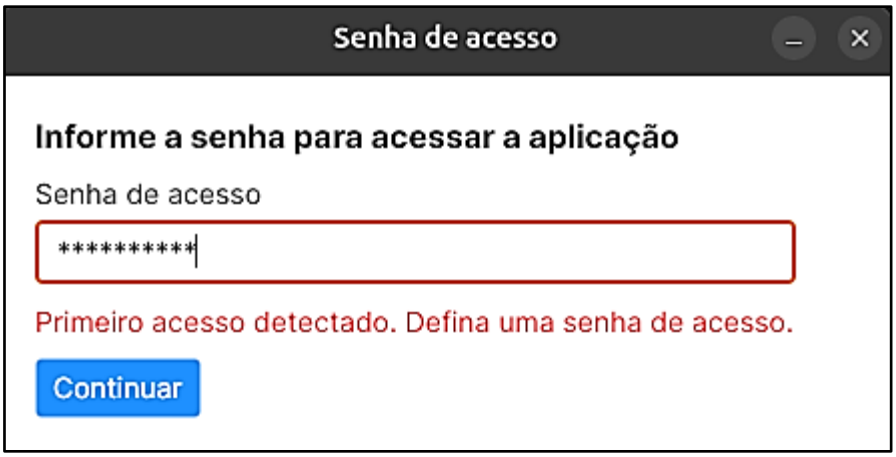
### 4.5.2 Configuração do banco de dados

Inicialmente foi criado o banco de dados denominado financas, que serve como base de teste para os processos de *backup* e restauração. Nesse banco de dados foram inseridos dados fictícios para testes de integridade e restauração. Para garantir o isolamento das permissões, foi criado um usuário específico chamado *backupuser*, limitado a operações de leitura necessárias para a exportação de dados.

Esse modelo segue as recomendações de segurança do PostgreSQL (PostgreSQL, 2024), reduzindo o risco de acesso indevido a tabelas sensíveis.

Durante a primeira execução do sistema, é exibida uma janela para a definição da senha de acesso, conforme ilustrado na Figura 14. Essa senha é utilizada como chave de criptografia para proteger os dados sensíveis da aplicação, incluindo as credenciais de conexão com o banco de dados e o serviço de armazenamento em nuvem. Além disso, ela é requerida em execuções posteriores para liberar o acesso às funcionalidades do sistema, como a gravação das configurações do PostgreSQL e a restauração de *backups*.

Figura 14: Janela de primeiro acesso

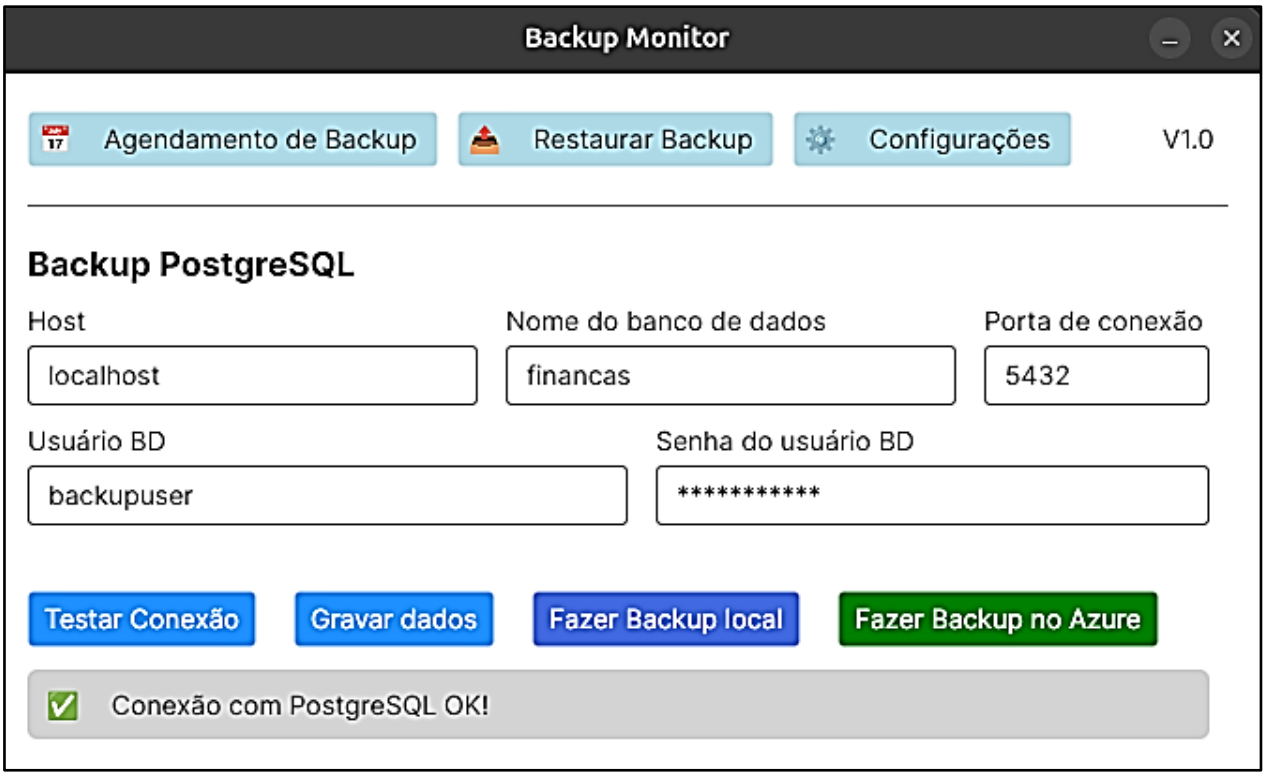



Fonte: Capturado pelo autor desse trabalho a partir do sistema Ubuntu, 2025.

O mecanismo de proteção adotado baseia-se na geração de um arquivo criptografado denominado config.enc, criado após a definição da senha inicial. Esse arquivo contém as informações de configuração do sistema e é protegido por meio do algoritmo AES em modo simétrico, o que garante a confidencialidade dos dados, conforme descrito por Stallings (2014). Caso o arquivo seja ausente ou corrompido, o sistema reconhece automaticamente a situação como um primeiro acesso e solicita ao usuário a criação de uma nova senha.

Após a definição da senha, o aplicativo é reiniciado automaticamente, carregando a *interface* principal para permitir a configuração do ambiente de banco de dados. As credenciais e o endereço do servidor PostgreSQL são configurados por meio da tela principal da aplicação, onde o usuário informa o *Host*, nome do banco de dados, porta de conexão, usuário do banco de dados, que será utilizado para realizar os *backups* e senha do usuário para acesso ao PostgreSQL. Esses dados podem ser testados através do botão Testar conexão, antes de serem gravados no arquivo de configuração criptografado através da ação do botão Gravar dados, garantindo a funcionalidade da aplicação, conforme ilustrado na Figura 15.

Figura 15: Janela principal

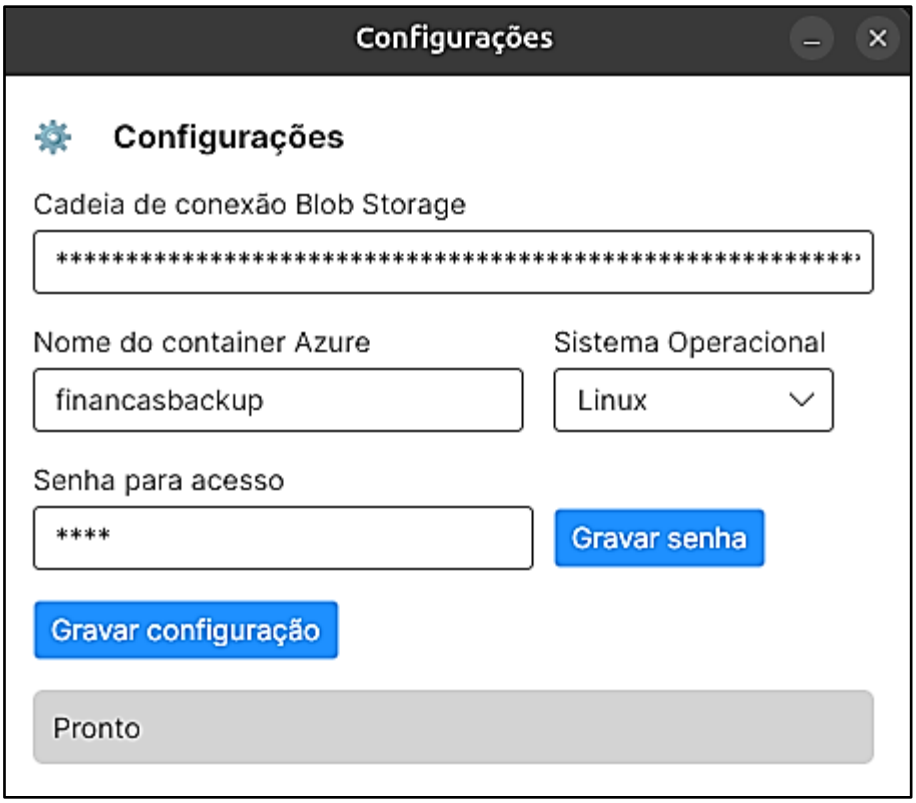



Fonte: Capturado pelo autor desse trabalho a partir do sistema Ubuntu, 2025.

### 4.5.3 Configuração da cadeia de conexão e contêiner de armazenamento

A etapa de configuração da aplicação foi projetada para permitir o registro das informações necessárias à integração entre o sistema de *backup* local e o serviço de armazenamento em nuvem da plataforma Azure. Essa configuração é acessada por meio da janela “Configurações”, conforme ilustrado na Figura 16, o usuário pode inserir e gerenciar as credenciais de acesso, bem como definir os parâmetros operacionais do ambiente.

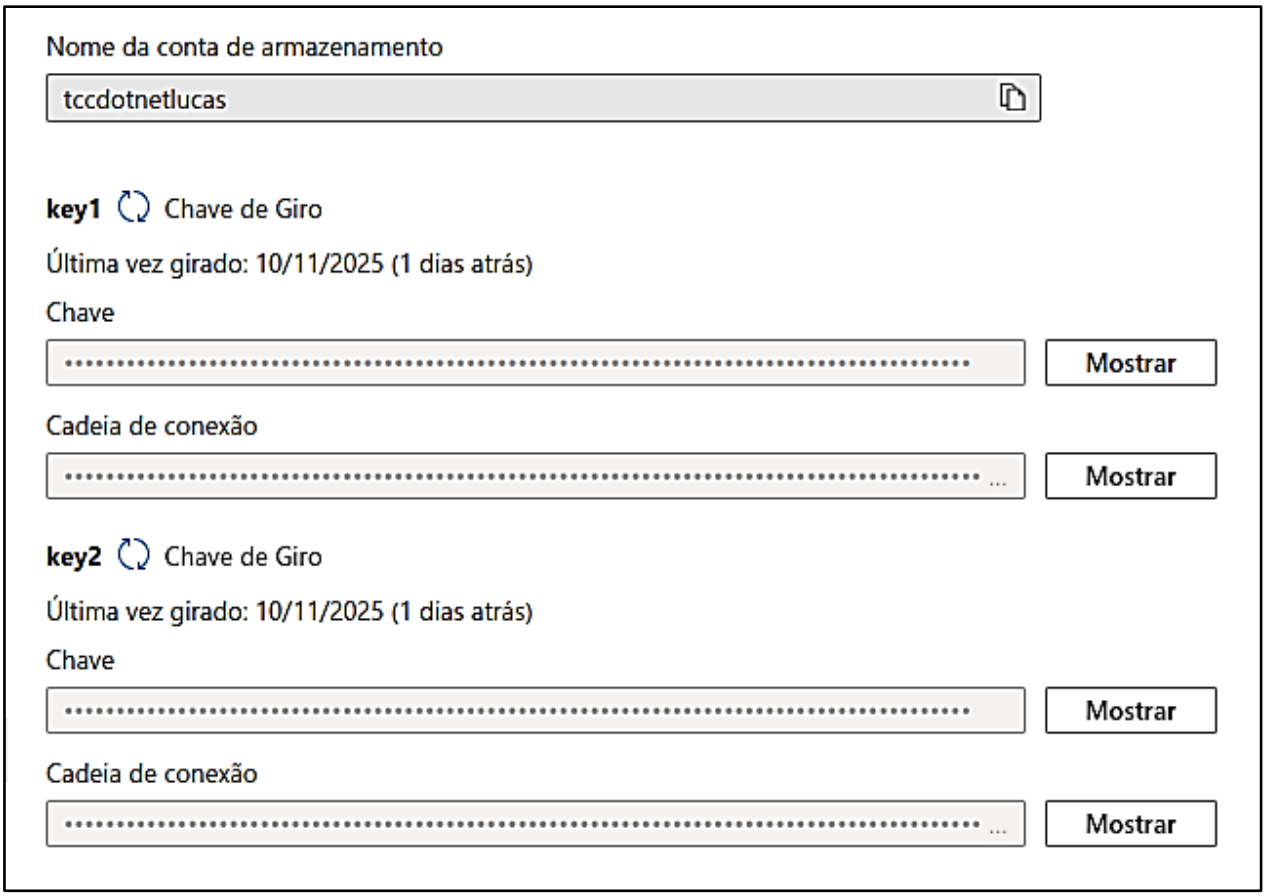
Figura 16: Janela de configurações




Fonte: Capturado pelo autor desse trabalho a partir do sistema Ubuntu, 2025.

Entre os campos disponibilizados, destaca-se a cadeia de conexão do Blob *Storage* (*connection string*), responsável por estabelecer a autenticação e o vínculo direto com a conta de armazenamento configurada no portal do Azure. Esse identificador contém as chaves de acesso geradas automaticamente pela plataforma, e seu uso é essencial para a execução das operações de envio (*upload*) e recuperação (*download*) dos arquivos de *backup*. A cadeia de conexão pode ser obtida através do portal Azure, acessando a opção Chaves de acesso no menu Segurança + rede, conforme ilustrado na Figura 17.

Figura 17: Funções hash em armazenamento de senhas




Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

O sistema também solicita o nome do contêiner Azure, que representa o diretório lógico onde os arquivos de *backup* serão armazenados na nuvem. Cada contêiner é associado a uma conta de armazenamento e funciona como o ponto de destino das operações realizadas pela aplicação. A criação e o gerenciamento desses contêineres seguem as diretrizes estabelecidas na documentação oficial do Azure (Microsoft, 2025b), que recomenda a utilização de nomes curtos e descritivos, compatíveis com a política de nomenclatura do serviço.

Além dos parâmetros relacionados ao serviço em nuvem, o sistema solicita a indicação do sistema operacional em uso, permitindo ajustar automaticamente o comportamento das rotinas de agendamento de *backup*. No caso de ambientes Linux, é utilizado o agendador Cron, enquanto em sistemas Windows a execução é realizada pelo

Task Scheduler. Essa distinção garante compatibilidade entre plataformas, uma vez que o aplicativo foi desenvolvido sobre o framework .NET 9.0 com suporte multiplataforma.

Outro elemento da janela de configuração é o campo senha de acesso, utilizado para proteger o arquivo de configuração principal config.enc. Essa é senha definida pelo usuário durante a primeira execução da aplicação e serve como chave de criptografia para a proteção das credenciais de conexão. O armazenamento das informações é realizado localmente, empregando o algoritmo AES em modo simétrico, garantindo confidencialidade e integridade conforme as recomendações de Stallings (2014).

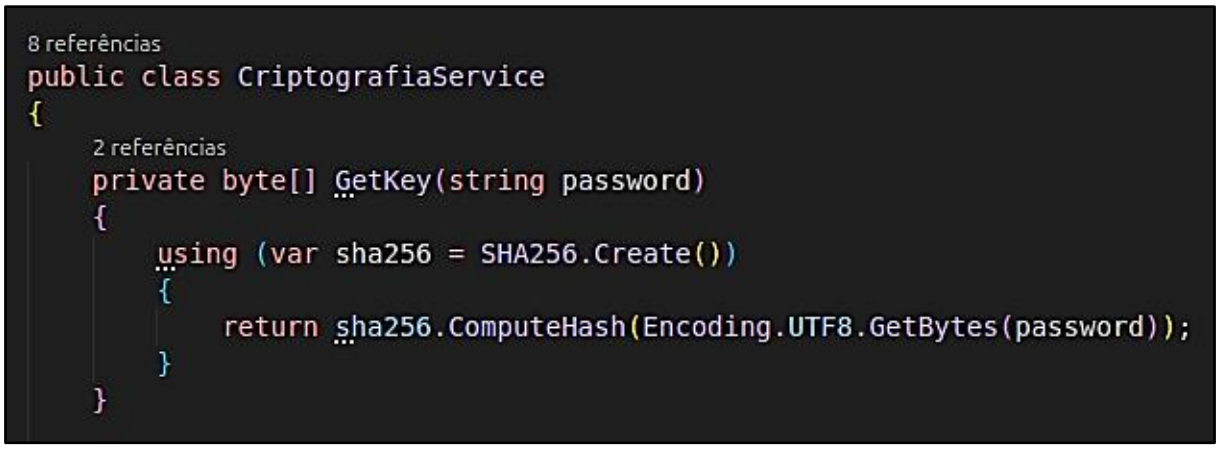
A *interface* disponibiliza duas ações principais: “Gravar Configuração”, que salva as informações inseridas de forma criptografada e “Gravar Senha”, que permite redefinir a senha de acesso. O *status* das operações é apresentado dinamicamente no painel inferior, informando o usuário sobre o resultado de cada ação executada.

Essa estrutura de configuração foi projetada para oferecer simplicidade de uso, segurança dos dados e flexibilidade operacional, permitindo que o sistema seja implantado em diferentes ambientes corporativos sem a necessidade de reconfiguração manual complexa.

### 4.5.4 Configurações de segurança

A segurança das informações armazenadas localmente é assegurada por meio do uso do algoritmo AES, aplicado em conjunto com a função de *hash* SHA-256. A implementação do processo de criptografia e geração de chave foi realizada na classe CriptografiaService, responsável por encapsular as operações de proteção e restauração de arquivos sensíveis, conforme ilustrado nas Figuras 18 e 19.

Figura 18: Método GetKey



```
8 referências
public class CriptografiaService
{
    2 referências
    private byte[] GetKey(string password)
    {
        using (var sha256 = SHA256.Create())
        {
            return sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
        }
    }
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

Figura 19: Método EncryptFile

```
4 referências
public void EncryptFile(string inputPath, string outputPath, string
password)
{
    using (var aes = Aes.Create())
    {
        aes.Key = GetKey(password);
        aes.GenerateIV();

        using (var inputStream = File.OpenRead(inputPath))
        using (var outputStream = File.Create(outputPath))
        {
            outputStream.Write(aes.IV, 0, aes.IV.Length);

            using (var cryptoStream = new CryptoStream(outputStream, aes.
            CreateEncryptor(), CryptoStreamMode.Write))
            {
                inputStream.CopyTo(cryptoStream);
            }
        }
    }
}
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

O método GetKey() converte a senha fornecida pelo usuário em uma chave criptográfica de 256 *bits*, aplicando a função de *hash* SHA-256 por meio da classe SHA256.Create(), pertencente ao namespace System.Security.Cryptography do *framework* .NET.

Essa abordagem assegura que a senha seja processada de maneira unidirecional, impedindo sua recuperação direta, ao mesmo tempo em que produz uma chave compatível com o algoritmo AES-256 utilizado na criptografia simétrica.

Durante a execução do método EncryptFile(), o sistema gera um Vetor de Inicialização (IV) aleatório por meio do método aes.GenerateIV(), garantindo que cada operação produza um resultado criptográfico distinto, mesmo quando a mesma senha é utilizada em múltiplas execuções.

O IV é gravado no início do arquivo criptografado, possibilitando sua leitura posterior no método DecryptFile(), que realiza o processo inverso de descriptografia, conforme ilustrado na Figura 20. Essa prática segue as diretrizes do NIST para o uso seguro de algoritmos simétricos, assegurando tanto a confidencialidade quanto a não repetição de padrões nos dados cifrados.

Figura 20: Método DecryptFile

```
4 referências
public void DecryptFile(string encryptedPath, string outputPath, string
password)
{
    using (var aes = Aes.Create())
    using (var inputStream = File.OpenRead(encryptedPath))
    {
        byte[] iv = new byte[16];
        inputStream.ReadExactly(iv);

        aes.Key = GetKey(password);
        aes.IV = iv;

        using (var cryptoStream = new CryptoStream(inputStream, aes.
        CreateDecryptor(), CryptoStreamMode.Read))
        using (var outputStream = File.Create(outputPath))
        {
            cryptoStream.CopyTo(outputStream);
        }
    }
}
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

A estrutura modular da classe CriptografiaService permite que o mesmo mecanismo seja empregado em diferentes etapas do sistema, como a criptografia do arquivo de configuração (config.enc) e dos *backups* antes do envio ao ABS, mantendo um padrão de segurança uniforme em todo o ciclo de operação. Essa implementação reforça a integridade dos dados e garante conformidade com as boas práticas de segurança recomendadas por Stallings (2014) e NIST (2015).

### 4.5.5 *Backup* manual local

A realização de *backups* manuais é possível através da tela principal do aplicativo, o usuário pode optar por armazenar a cópia de segurança localmente ou enviá-la diretamente ao ambiente em nuvem. Ambas as operações utilizam, em sua base, o utilitário nativo pg_dump do PostgreSQL para exportação dos dados.

O processo é conduzido de forma automatizada pela classe MainWindowViewModel, a qual implementa métodos distintos para cada modalidade: BackupLocalAsync() e BackupDatabase(). Na ação de *backup* local, o usuário é solicitado a escolher o caminho e o nome do arquivo de destino através de uma janela de seleção, que sugere um nome padrão no formato: {nome_do_banco}_backup_{data}_{hora}.dump

Após a confirmação, a aplicação inicia a geração do arquivo de *backup*, exibindo o *status* do processo na *interface*. A operação é realizada pelo método BackupLocalAsync(), conforme o trecho de código apresentado na Figura 21.

Figura 21: Método BackupLocalAsync

```
var dialog = new SaveFileDialog
{
    Title = "Salvar backup do banco",
    Filters =
    {
        new FileDialogFilter() { Name = "Arquivos de backup PostgreSQL", Extensions = { "dump" } },
        new FileDialogFilter() { Name = "Todos os arquivos", Extensions = { "*" } }
    },
    InitialFileName = $"{Modelo.DbName}_backup_{DateTime.Now:yyyyMMdd_HHmm}.dump"
};

string? filePath = await dialog.ShowAsync(ownerWindow);
if (string.IsNullOrEmpty(filePath))
{
    Modelo.Status = "Operação cancelada pelo usuário.";
    return;
}

Modelo.Status = "⏳ Gerando dump...";

// Chama o serviço de dump
string dumpPath = pgService.BackupDatabase(
    Modelo.DbHost,
    Modelo.DbPort,
    Modelo.DbName,
    Modelo.DbUser,
    Modelo.DbPassword,
    filePath
);
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

O método aciona a função BackupDatabase pertencente à classe PostgresService, que encapsula a execução do comando pg_dump. Esse utilitário é o responsável por exportar os dados do banco de dados PostgreSQL em formato binário customizado (-F c), garantindo que a estrutura, índices e dados sejam preservados de

forma consistente. O comando é montado dinamicamente, vemos isso no código, na linha 18 dentro do método BackupDatabase, conforme ilustrado na Figura 22.

Figura 22: Método BackupDatabase

```
public string BackupDatabase(string host, int port, string dbName, string user, string password, string outputFile)
{
    Directory.CreateDirectory(Path.GetDirectoryName(outputFile)!);

    var psi = new ProcessStartInfo
    {
        FileName = "pg_dump",
        Arguments = $"-h {host} -p {port} -U {user} -F c -f \"{outputFile}\" {dbName}",
        UseShellExecute = false,
        RedirectStandardError = true,
        RedirectStandardOutput = true,
        CreateNoWindow = true
    };

    psi.Environment["PGPASSWORD"] = password;

    using var process = Process.Start(psi)!;
    string error = process.StandardError.ReadToEnd();
    process.WaitForExit();

    if (process.ExitCode != 0)
        throw new Exception($"Erro ao gerar dump: {error}");

    return outputFile;
}
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

A autenticação é realizada através da variável de ambiente PGPASSWORD, definida no sistema durante a execução do processo, o que elimina a necessidade de intervenção manual. Após a conclusão do processo, o arquivo .dump é gerado no local indicado pelo usuário, podendo ser restaurado posteriormente por meio do utilitário pg_restore. Por se tratar de uma cópia não criptografada, recomenda-se que o usuário mantenha o arquivo em um diretório protegido, com permissões de acesso restritas, evitando assim a exposição de informações sensíveis.

### 4.6 *Backup* manual para o Azure

A aplicação oferece a opção de realizar o *backup* manual com envio direto ao armazenamento em nuvem, o qual amplia a segurança e disponibilidade dos dados. Nesse caso, o método executado é o BackupDatabaseAzure, que segue uma sequência

de operações controladas para garantir a integridade e confidencialidade da cópia de segurança.

Inicialmente, o banco de dados PostgreSQL é exportado para um diretório temporário definido pela aplicação, evitando que o usuário precise manipular arquivos intermediários ou lidar diretamente com dados sensíveis no sistema local. Essa etapa resulta na criação de um arquivo de *dump* contendo a estrutura e os dados do banco, conforme ilustrado na Figura 23.

Figura 23: Criação do *dump* de banco de dados

```
//Gera o dump do banco de dados em diretório temporário
string dumpPath = pgService.BackupDatabase(
    Modelo.DbHost,
    Modelo.DbPort,
    Modelo.DbName,
    Modelo.DbUser,
    Modelo.DbPassword,
    Path.Combine(Path.GetTempPath(), $"{Modelo.DbName}_azure_{DateTime.Now:yyyyMMdd_HHmmss}.dump")
);

Modelo.Status = $"Dump criado: {Path.GetFileName(dumpPath)}\nCriptografando arquivo.";
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

Na sequência, o arquivo de *dump* gerado é submetido ao processo de criptografia local por meio do serviço CriptografiaService, que implementa o algoritmo AES. O método EncryptFile() é responsável por esse procedimento, produzindo um novo arquivo com a extensão .enc. A chave criptográfica utilizada é derivada a partir da senha de acesso definida pelo usuário e armazenada de forma segura no arquivo de configuração da aplicação. Esse mecanismo assegura que apenas usuários autorizados, detentores da senha correta, possam restaurar o conteúdo do *backup*, conforme apresentado na Figura 24.

Figura 24: Criptografia do arquivo de *backup*

```
//Criptografa o arquivo antes do envio
string encPath = Path.Combine(Path.GetTempPath(), Path.GetFileName(dumpPath) + ".enc");
criptoService.EncryptFile(dumpPath, encPath, appConfig.AccessPassword);
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

Após a criptografia, o sistema realiza a geração de um *hash* criptográfico utilizando o algoritmo SHA-256 sobre o arquivo já criptografado. Diferente de abordagens tradicionais, essa estratégia garante a verificação de integridade do conteúdo exatamente como ele será armazenado na nuvem, protegendo não apenas os dados originais, mas também sua forma cifrada. O valor do *hash* gerado é utilizado como identificador adicional do *backup* e será empregado posteriormente para validações de integridade durante o processo de restauração.

Na etapa seguinte, o arquivo criptografado é enviado automaticamente ao ABS por meio do método Upload da classe AzureBlobService. Durante esse envio, o *hash* calculado é gravado nos metadados do *blob*, permitindo que a integridade do arquivo seja verificada diretamente no ambiente de armazenamento em nuvem, sem a necessidade de manter registros locais adicionais. O nome do arquivo enviado inclui informações como o nome do banco de dados, data, hora da operação e um trecho do *hash*, facilitando sua identificação e rastreabilidade.

Concluído o envio, os arquivos temporários utilizados durante o processo são removidos automaticamente do sistema local, reduzindo o risco de exposição de dados sensíveis. Ao final da operação, a aplicação atualiza o status exibido na interface gráfica, informando o sucesso do procedimento e confirmando o armazenamento seguro do *backup* no ambiente em nuvem, conforme ilustrado na Figura 25.

Figura 25: Envio de *backup* para o Azure Blob Storage

```
//Calcula o hash SHA-256
Modelo.Status = "Calculando hash SHA-256 do arquivo criptografado.";
string hash = hashService.ComputeSha256(encPath);
string hashPreview = hash.Length >= 12 ? hash.Substring(0, 12) : hash;
Modelo.Status = $"Hash (SHA-256): {hashPreview}.\nEnviando para Azure Blob Storage.";

//Envia para o Azure, gravando o hash nos metadados
string blobFileName = $"{Modelo.DbName}_{DateTime.Now:yyyyMMdd_HHmmss}_{hash.Substring(0, 16)}.enc";
azureService.Upload(encPath, blobFileName, hash);

Modelo.Status = "Backup concluído! Limpando arquivos temporários...";

//Limpa os arquivos temporários
if (File.Exists(dumpPath)) File.Delete(dumpPath);
if (File.Exists(encPath)) File.Delete(encPath);

Modelo.Status = "✅ Backup finalizado com sucesso!";
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

Em síntese, o *backup* manual local oferece rapidez e praticidade em ambientes onde o controle físico do arquivo é suficiente, enquanto o *backup* manual com envio ao Azure prioriza a segurança, a integridade e a disponibilidade contínua dos dados. Ambas as abordagens se complementam dentro do contexto da aplicação, permitindo que o usuário adote a estratégia de *backup* mais adequada às suas necessidades operacionais.

### 4.6.1 Configuração da política de *backup*

A política de *backup* adotada neste projeto foi definida com o objetivo de assegurar a disponibilidade e a integridade dos dados armazenados no banco PostgreSQL, considerando práticas reconhecidas na área de segurança da informação (Stallings, 2014; Kavis, 2014) e recomendações oficiais da Microsoft (2025). A política concentra-se exclusivamente na realização de *backups* completos, em razão do escopo da aplicação e da viabilidade técnica do ambiente utilizado.

#### 4.6.1.1 Tipo de *backup* utilizado

No sistema desenvolvido, o *backup* completo é produzido pelo utilitário pg_dump no formato binário customizado (-F c), que permite restauração via pg_restore e compressão eficiente. Todo o processo segue o seguinte fluxo:

- Geração do *dump* com pg_dump, incluindo todas as tabelas e objetos do banco;
- Compactação em formato .gz para reduzir espaço local e tamanho de upload;
- Criptografia local utilizando AES-256;
- Envio automático para o Azure Blob Storage, na camada Archive;
- Exclusão dos arquivos temporários gerados no sistema local.

Esse procedimento garante que cada arquivo represente o estado completo do banco de dados no momento da cópia, possibilitando uma restauração íntegra sem dependência de *backups* anteriores.

#### 4.6.1.2 Frequência dos *backups*

A periodicidade da execução é definida pelo usuário por meio da *interface* gráfica e aplicada no sistema operacional por meio do agendador Cron. As opções disponíveis incluem:

- Diário;
- Semanal;
- Mensal.

Nos testes realizados, adotou-se como política recomendada:

- *Backup* completo semanal, executado preferencialmente em horários de baixa atividade.

Essa frequência reduz o custo de armazenamento no Azure e, ao mesmo tempo, garante cópias suficientes para recuperação em eventuais falhas.

#### 4.6.1.3 Retenção dos arquivos de *backup*

A política de retenção estabelecida é:

- Retenção mínima de 4 semanas para backups completos armazenados na nuvem.

A remoção ou extensão da retenção pode ser ajustada manualmente no portal do Azure conforme a necessidade da organização. Essa retenção foi adotada considerando:

- O balanço entre custo de armazenamento;
- A necessidade de disponibilidade histórica;
- A simplicidade operacional do sistema.

#### 4.6.1.4 Validação de integridade

Durante a restauração, o *hash* SHA-256 previamente gerado é comparado ao *hash* recalculado a partir do arquivo recuperado.

Esse processo assegura:

- Integridade fim-a-fim do artefato;

- Ausência de alterações acidentais durante o armazenamento;
- Confiabilidade do *backup* antes de iniciar qualquer operação de restauração.

### 4.6.2 Agendamento pelo Cron do Linux

O agendamento automático dos *backups* completos é realizado por meio do Cron, um serviço nativo dos sistemas operacionais baseados em Unix, responsável pela execução periódica de tarefas em segundo plano. O Cron utiliza uma tabela de instruções denominada crontab, na qual cada linha define um comando, intervalo de execução e o usuário sob o qual a tarefa será executada (Negus, 2015).

O agendamento é essencial para assegurar que o processo de *backup* ocorra de forma contínua e independente de intervenção humana, garantindo regularidade e previsibilidade no ciclo de proteção dos dados. O aplicativo desenvolvido automatiza completamente a criação do *script* de *backup*, bem como a sua inserção na agenda do sistema, reduzindo a possibilidade de erros manuais por parte do usuário. Toda a parte de configuração do agendamento pode ser feita diretamente no aplicativo, conforme ilustrado na Figura 26.

Figura 26: Tela de agendamento de *backup*


Fonte: Capturado pelo autor desse trabalho a partir do sistema Ubuntu, 2025.

#### 4.6.2.1 Geração do *script* de *backup*

A classe CronScriptService é responsável pela criação dos *scripts* de *backup* utilizados pelo agendador Cron. Seu funcionamento foi estruturado de forma modular,

permitindo que cada etapa, geração de diretórios, composição do *script*, gravação do arquivo e definição das permissões, seja realizada de forma clara e programática. A seguir descrevem-se os principais trechos da implementação.

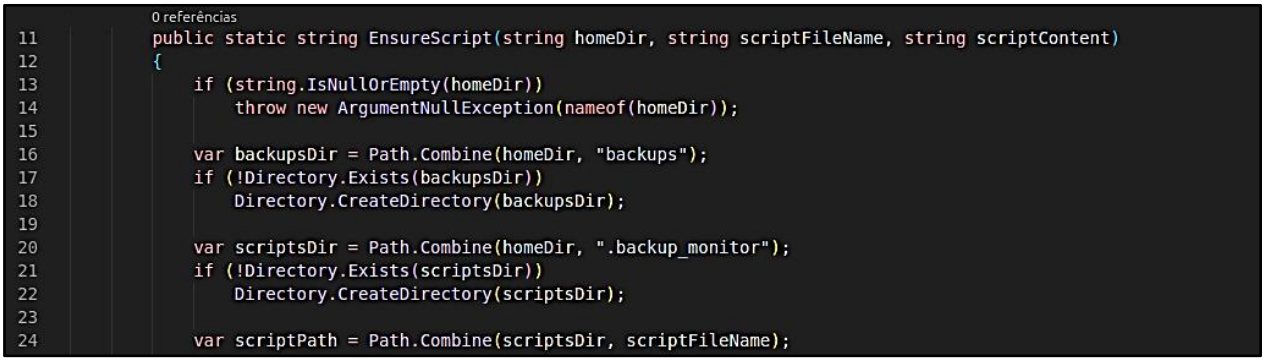
4.6.2.1.1 Criação dos diretórios e salvamento do *script*

O método EnsureScript realiza a preparação do ambiente necessário para os *backups*. Nas linhas iniciais, 13 e 14, há uma validação do diretório base informado, evitando que o método prossiga com um caminho inválido.

Em seguida, nas linhas 16 a 18, é criado o diretório $HOME/backups, utilizado para armazenar temporariamente os arquivos .dump antes da compactação.

Nas linhas 20 a 22 é criado o diretório oculto $HOME/.backup_monitor, onde ficam os *scripts* e arquivos de log. E na linha 24 define-se o caminho completo do *script* que será gravado, conforme ilustrado na Figura 27.

Figura 27: Método EnsureScript




Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

A gravação do conteúdo ocorre na linha 27, utilizando File.WriteAllText, com codificação UTF-8 sem BOM, para evitar caracteres invisíveis que poderiam invalidar a execução no Bash.

Nas linhas 30 a 38, o método executa um processo externo que aplica permissão de execução (chmod +x) ao *script* que acabou de ser criado. O uso de ProcessStartInfo garante compatibilidade com ambientes Linux sem depender de bibliotecas adicionais.

Esse método retorna o caminho final do *script*, que será posteriormente configurado no Cron, conforme ilustrado na Figura 28.

Figura 28: Aplicação de permissão para execução de arquivo

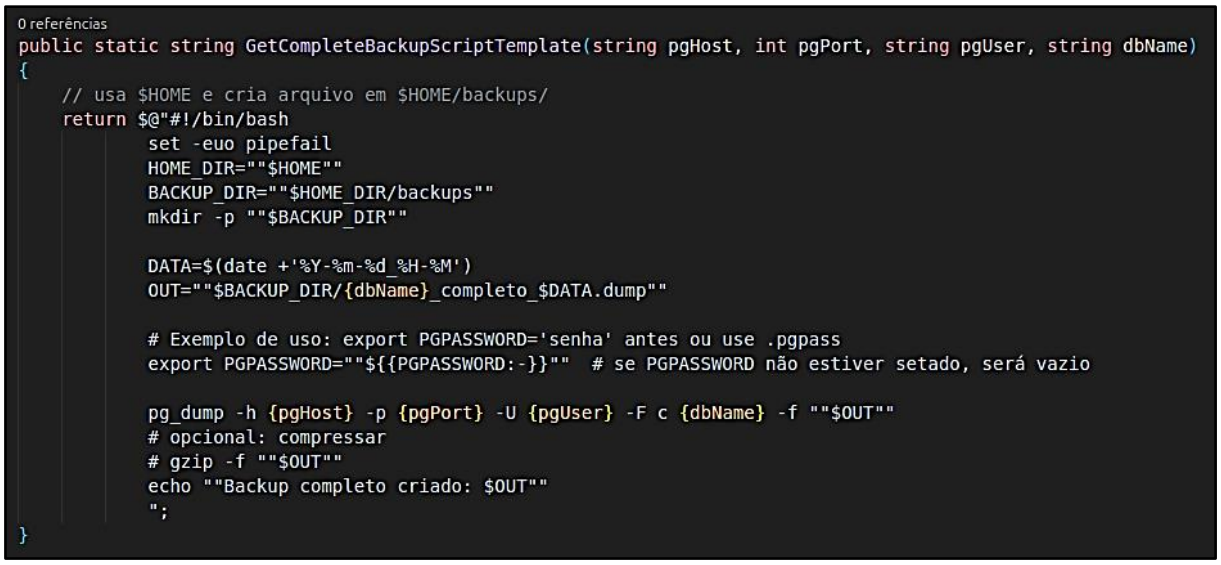
```
28
29              // Garante permissões executáveis (chmod +x)
30              var psi = new ProcessStartInfo
31              {
32                  FileName = "/bin/bash",
33                  Arguments = $"-lc \"chmod +x '{scriptPath}'\"",
34                  UseShellExecute = false,
35                  CreateNoWindow = true
36              };
37              using var p = Process.Start(psi);
38              p?.WaitForExit();
39
40              return scriptPath;
41          }
42
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

4.6.2.1.2 Geração do modelo de *script* de *backup* completo

O método GetCompleteScriptTemplate, ilustrado na Figura 29, retorna um *template string* contendo o conteúdo literal de um *script* de *backup*.

Figura 29: Método para criação do *script* de *backup*



```
0 referências
public static string GetCompleteBackupScriptTemplate(string pgHost, int pgPort, string pgUser, string dbName)
{
    // usa $HOME e cria arquivo em $HOME/backups/
    return $@"#!/bin/bash
            set -euo pipefail
            HOME_DIR=""$HOME""
            BACKUP_DIR=""$HOME_DIR/backups""
            mkdir -p ""$BACKUP_DIR""

            DATA=$(date +'%Y-%m-%d_%H-%M')
            OUT=""$BACKUP_DIR/{dbName}_completo_$DATA.dump""

            # Exemplo de uso: export PGPASSWORD='senha' antes ou use .pgpass
            export PGPASSWORD=""${{PGPASSWORD:-}}""  # se PGPASSWORD não estiver setado, será vazio

            pg_dump -h {pgHost} -p {pgPort} -U {pgUser} -F c {dbName} -f ""$OUT""
            # opcional: compressar
            # gzip -f ""$OUT""
            echo ""Backup completo criado: $OUT""
            ";
}
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

Esse arquivo inclui:

- Definição de variáveis de ambiente (HOME_DIR, BACKUP_DIR);
- Criação da pasta de saída;
- Geração de um nome dinâmico contendo data e hora (DATA=$(date ...));
- Atribuição da variável de ambiente PGPASSWORD, que impede que o PostgreSQL solicite senha no terminal, que poderia interromper o fluxo de automação do *backup*;
- Execução do utilitário pg_dump com parâmetros -F c, que gera um arquivo compatível com o pg_restore.

Todas essas etapas aparecem estruturadas nas linhas aproximadamente 50 a 75 do código. O *script* é retornado como texto puro, permitindo sua posterior gravação com o método EnsureScript.

#### 4.6.2.1.3 Geração do *script* final de *backup* completo

Este método é responsável por gerar o *script* completo contendo todas as etapas necessárias para que:

1. O banco seja exportado;

2. O arquivo seja compactado;
3. O aplicativo execute o envio ao Azure.

Portanto, é necessário realizar a declaração de variáveis de ambiente essenciais, que são:

- PATH atualizado para garantir que o Dotnet seja encontrado pelo Cron;
- DOTNET_ROOT para ambientes que utilizam SDK via pacote;
- BACKUPMONITOR_PASSWORD, que é lido pela aplicação no modo --auto-backup.

Também é necessário realizar a geração do arquivo ABS, que é criado com a chamada do pg_dump com o nome do banco, usuário, *host* e porta de conexão, que foram definidas anteriormente e salvas no arquivo de configuração. Esse arquivo é salvo em ${BACKUP_DIR}, usando nomenclatura temporal.

Após a geração do arquivo .dump, é realizada a compactação do arquivo utilizando o comando gzip, que substitui o arquivo .dump por um arquivo .dump.gz.

4.6.2.1.4 Criação e gravação do *script* final no diretório do usuário

O método CriarScript reúne todas as partes descritas anteriormente, conforme ilustrado na Figura 30.

Figura 30: Método CriarScript

```
2 referências
public static string CriarScript(string tipo, AppConfig cfg)
{
    string home = Environment.GetEnvironmentVariable("HOME") ?? $"/home/{Environment.UserName}";
    string scriptsDir = Path.Combine(home, ".backup_monitor");
    Directory.CreateDirectory(scriptsDir);

    string scriptPath = Path.Combine(scriptsDir, $"backup_{tipo}.sh");
    string content = GetBackupScriptTemplate(tipo, cfg);
    File.WriteAllText(scriptPath, content, new UTF8Encoding(false));

    var psi = new ProcessStartInfo
    {
        FileName = "/bin/bash",
        Arguments = $"-c \"chmod +x '{scriptPath}'\"",
        UseShellExecute = false,
        CreateNoWindow = true
    };
    Process.Start(psi)?.WaitForExit();

    return scriptPath;
}
```

Fonte: Capturado pelo autor desse trabalho a partir de Visual Code, 2025.

1. Linhas 124 a 126 determina o diretório do usuário ($HOME) e cria a pasta .backup_monitor;
2. Linha 128 define o nome do arquivo do *script*, como *backup*_completo.sh;
3. Linhas 129 e 130 gera o conteúdo do *script* usando o modelo produzido anteriormente;
4. Linhas 132 a 139 aplica permissão de execução com chmod +x;
5. Linha 141 retorna o caminho final do *script*, que será utilizado pela etapa de configuração no Cron.

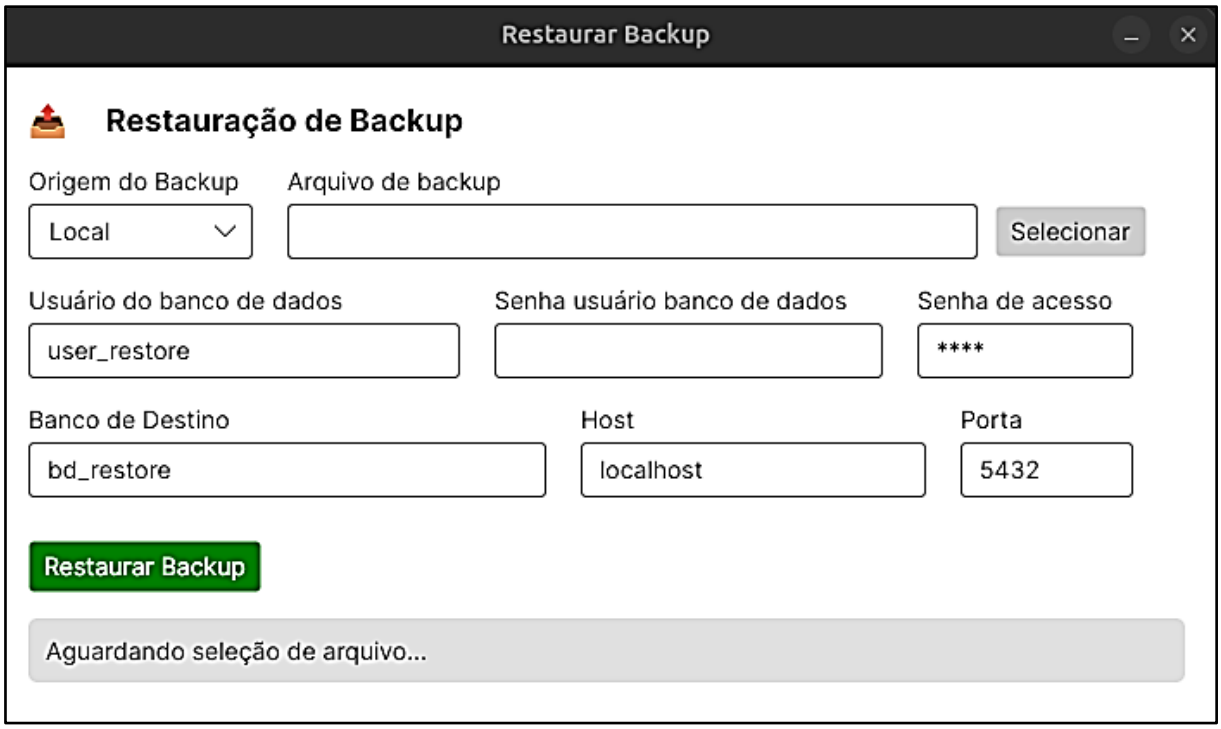
Esse método age como ponto central da classe, garantindo que o *script* criado esteja estruturado corretamente, salvo no local apropriado e com as permissões adequadas.

### 4.6.3 Configuração do *restore*

A configuração do processo de restauração, tem como objetivo proporcionar ao usuário um mecanismo controlado para reaplicação de *backups* sobre uma instância de banco de dados PostgreSQL. A *interface* de restauração disponibiliza os campos necessários para especificação das credenciais e do destino, que são, usuário, senha, nome da base, *host* e porta, também é possível escolher a origem do *backup, sendo* local ou no serviço de armazenamento em nuvem. Independentemente da origem, o arquivo selecionado é utilizado para restaurar diretamente o banco de dados indicado na própria tela da aplicação.

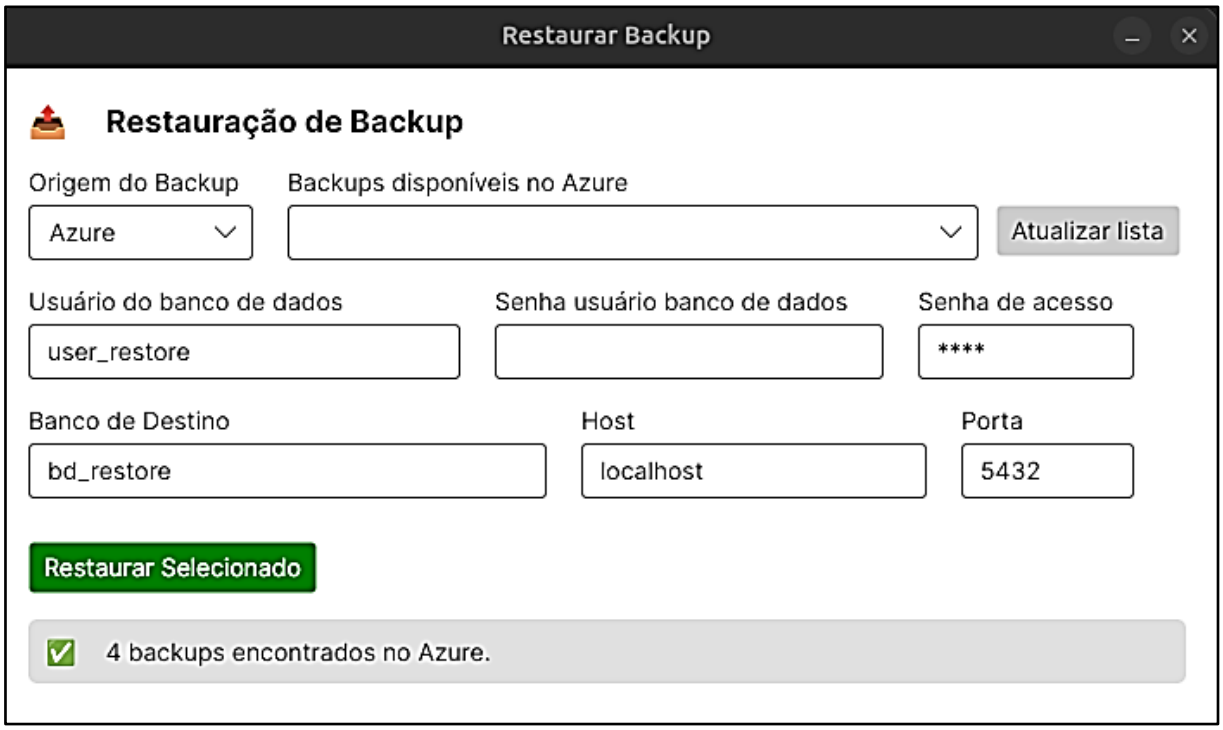
Quando a origem é local, o usuário seleciona, por meio do seletor de arquivos, o arquivo de *backup* previamente armazenado na máquina. Quando a origem é Azure, o usuário deve acionar o botão Atualizar lista para que o aplicativo consulte o serviço de armazenamento e preencha o campo que lista os *backups* disponíveis. Essa lista apresenta os nomes dos arquivos enviados anteriormente ao Azure, possibilitando a seleção de qualquer item para restauração. O procedimento de seleção determina apenas onde o *backup* está armazenado; a operação de restauração em si permanece idêntica para ambas as origens, conforme ilustrado nas Figuras 31 e 32.

Figura 31: Tela de restauração com origem local




Fonte: Capturado pelo autor desse trabalho a partir do sistema Ubuntu, 2025.

Figura 32: Tela de restauração com origem Azure




Fonte: Capturado pelo autor desse trabalho a partir do sistema Ubuntu, 2025.

O sistema exige que o usuário informe novamente as credenciais de acesso ao banco de dados, mesmo que essas informações já tenham sido cadastradas ou utilizadas anteriormente. Essa decisão busca reforçar a segurança da operação, garantindo que apenas usuários autorizados possam executar uma ação sensível como a restauração de dados. Além disso, recomenda-se a utilização de usuários distintos para as operações de *backup* e de *restore*, uma vez que cada uma possui escopos e permissões específicas. Por exemplo, enquanto um usuário dedicado ao *backup* pode ter permissões limitadas à leitura da base, um usuário responsável pela restauração deve possuir permissões adequadas para sobrescrever dados existentes.

Após a seleção do arquivo, a aplicação realiza as etapas prévias necessárias antes de invocar a rotina de restauração: validação de integridade por meio do *hash* armazenado nos metadados, validação do formato do arquivo, eventual descriptografia e descompactação do conteúdo, se aplicável. Concluídas essas etapas de preparação, o sistema monta a chamada ao utilitário pg_restore, incorporando os parâmetros fornecidos na *interface*. A execução do pg_restore é delegada ao sistema operacional, o aplicativo monitora o processo para capturar o seu retorno e apresentar mensagens de progresso ou erro ao usuário.

A centralização das configurações na janela de restauração oferece flexibilidade para restaurar *backups* em ambientes distintos, por exemplo, em instâncias locais, em servidores de desenvolvimento ou em bases temporárias, sem necessidade de editar o arquivo de configuração do sistema. Essa abordagem favorece a segurança dos dados armazenados e a autonomia do usuário para definir o destino da restauração no momento da operação, reduzindo o risco de aplicação acidental em ambientes não desejados.

## 5 TESTES E RESULTADOS

Os testes de *backup* foram conduzidos com o objetivo de validar a política definida na Seção 5.7, assegurando que o sistema é capaz de produzir cópias completas consistentes, criptografadas e corretamente enviadas ao ABS. Para isso, foram executados quatro ciclos de *backup* completo, cada um refletindo um estado distinto do banco de dados. A evolução da base foi realizada progressivamente, permitindo verificar se cada arquivo produzido representava fielmente o estado do sistema no momento da captura. O banco de dados inicialmente possuía 150 usuários, 300 registros financeiros e 150 cartões cadastrados, ilustrado na Figura 33.

Figura 33: Tabelas do banco de dados


Fonte: Imagem feita pelo autor deste trabalho, com capturas a partir de PgAdmin 4, 2025.

Todos os testes seguiram o fluxo operacional estabelecido:

- Geração do *dump* utilizando pg_dump no formato customizado;
- Compactação automática em .gz;
- Criptografia com AES-256;
- *Upload* para o Azure Blob Storage, camada Archive;
- Remoção dos artefatos temporários no ambiente local.

O objetivo principal foi confirmar a integridade do procedimento e a aderência à política de *backup* completo, conforme descrito na Seção 5.7.1.

## 5.1 Testes de *backup*

Antes de cada execução, o banco foi modificado de forma incremental, inserindo novos registros nas tabelas Usuario, Cartoes e Financeiro. Após cada alteração, um novo *backup* completo foi produzido e posteriormente enviado ao ABS.

O primeiro *backup* foi executado logo após a inserção de um novo usuário no sistema: Lucas Augusto de Souza, e-mail: lsouza.core@gmail.com.

Após a confirmação da inserção, o *backup* completo foi gerado pelo sistema, representando o estado inicial modificado da base. A execução ocorreu sem intercorrências, e o arquivo criptografado foi corretamente enviado ao Azure, conforme ilustrado na Figura 34.

Figura 34: Primeiro *backup* realizado em nuvem


Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

No segundo cenário de teste, foi adicionado um cartão associado ao usuário cadastrado. O cartão inserido foi: Cartão Inter Black, vinculado ao usuário Lucas Augusto de Souza.

Com essa nova entrada registrada na tabela Cartoes, o segundo *backup* completo foi executado. Novamente, o procedimento gerou o arquivo compactado e criptografado conforme a política definida, armazenando-o no ABS, conforme ilustrado nas Figuras 35 e 36.

Figura 35: Dados de cartão inseridos no banco de dados

| | | | | | |
|---|---|---|---|---|---|
| 149 | 149 | 149 | Cartão João | 4111 1111 1111 1771 | 10481.00 |
| 150 | 150 | 150 | Cartão Sérgio | 4111 1111 1111 0420 | 6111.00 |
| 151 | 151 | 151 | Inter Black | 4111 1111 1111 2002 | 10150.13 |

Total rows: 151 | Query complete 00:00:00.052

Fonte: Capturado pelo autor desse trabalho a partir de PgAdmin 4, 2025.

Figura 36: Segundo *backup* realizado em nuvem


Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

Para o terceiro cenário de teste, foi registrada uma nova despesa na tabela Financeiro, associada ao usuário 151. A despesa adicionada refere-se a uma movimentação de cartão de crédito. Após essa alteração, o terceiro *backup* completo foi executado. A verificação dos logs confirmou a conclusão bem-sucedida do processo e o correto envio do arquivo ao serviço de nuvem, ilustrado nas Figuras 37 e 38.

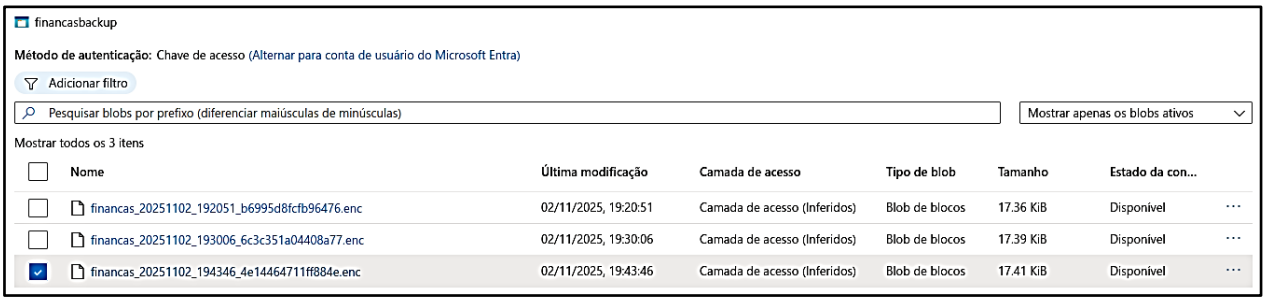
Figura 37: Dados financeiros inseridos no banco de dados

| | id [PK] integer | usuario_id integer | tipo character varying (20) | categoria character varying (60) | valor numeric (12,2) | data date |
|---|---|---|---|---|---|---|
| 285 | 285 | 143 | receita | Salário | 2296.00 | 2025-03-02 |
| 286 | 286 | 143 | despesa | Educação | 1150.00 | 2025-02-04 |
| 287 | 287 | 144 | receita | Salário | 5216.00 | 2025-04-05 |
| 288 | 288 | 144 | despesa | Saúde | 1156.00 | 2025-04-18 |
| 289 | 289 | 145 | receita | Salário | 7580.00 | 2025-07-06 |
| 290 | 290 | 145 | despesa | Educação | 366.00 | 2025-08-15 |
| 291 | 291 | 146 | receita | Salário | 6865.00 | 2025-01-26 |
| 292 | 292 | 146 | despesa | Educação | 1052.00 | 2025-06-07 |
| 293 | 293 | 147 | receita | Salário | 3791.00 | 2025-01-09 |
| 294 | 294 | 147 | despesa | Educação | 808.00 | 2025-02-25 |
| 295 | 295 | 148 | receita | Salário | 5142.00 | 2025-03-02 |
| 296 | 296 | 148 | despesa | Cartão | 792.00 | 2025-01-26 |
| 297 | 297 | 149 | receita | Salário | 4509.00 | 2025-05-20 |
| 298 | 298 | 149 | despesa | Educação | 784.00 | 2025-01-16 |
| 299 | 299 | 150 | receita | Salário | 3760.00 | 2025-02-18 |
| 300 | 300 | 150 | despesa | Lazer | 300.00 | 2025-08-06 |
| 301 | 301 | 151 | despesa | Cartão | 357.33 | 2025-11-02 |

Total rows: 301 | Query complete 00:00:00.042

Fonte: Capturado pelo autor desse trabalho a partir de PgAdmin 4, 2025.

Figura 38: Terceiro *backup* em nuvem realizado




Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

No quarto e último teste, novas informações financeiras foram adicionadas para o mesmo usuário. Foram incluídos:

- Uma receita do tipo “Salário”;
- Duas despesas, sendo uma classificada como Saúde e outra como Educação.

Essas inserções tornaram o estado da base distinto dos testes anteriores. Em seguida, foi realizado o quarto *backup* completo. Conforme esperado, o arquivo resultante refletiu todas as alterações realizadas desde o primeiro ciclo, demonstrando a consistência da abordagem de *backup* completo adotada, ilustrado nas Figuras 39 e 40.

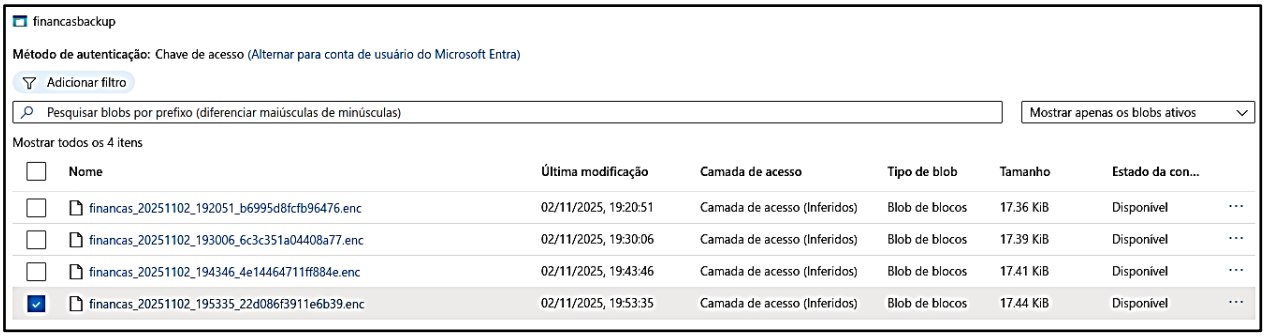
Figura 39: Tabela Financeiro com novos registros

| | id [PK] integer | usuario_id integer | tipo character varying (20) | categoria character varying (60) | valor numeric (12,2) | data date |
|---|---|---|---|---|---|---|
| 288 | 288 | 144 | despesa | Saúde | 1156.00 | 2025-04-18 |
| 289 | 289 | 145 | receita | Salário | 7580.00 | 2025-07-06 |
| 290 | 290 | 145 | despesa | Educação | 366.00 | 2025-08-15 |
| 291 | 291 | 146 | receita | Salário | 6865.00 | 2025-01-26 |
| 292 | 292 | 146 | despesa | Educação | 1052.00 | 2025-06-07 |
| 293 | 293 | 147 | receita | Salário | 3791.00 | 2025-01-09 |
| 294 | 294 | 147 | despesa | Educação | 808.00 | 2025-02-25 |
| 295 | 295 | 148 | receita | Salário | 5142.00 | 2025-03-02 |
| 296 | 296 | 148 | despesa | Cartão | 792.00 | 2025-01-26 |
| 297 | 297 | 149 | receita | Salário | 4509.00 | 2025-05-20 |
| 298 | 298 | 149 | despesa | Educação | 784.00 | 2025-01-16 |
| 299 | 299 | 150 | receita | Salário | 3760.00 | 2025-02-18 |
| 300 | 300 | 150 | despesa | Lazer | 300.00 | 2025-08-06 |
| 301 | 301 | 151 | despesa | Cartão | 357.33 | 2025-11-02 |
| 302 | 302 | 151 | receita | Salário | 5000.00 | 2025-11-02 |
| 303 | 303 | 151 | despesa | Saúde | 290.13 | 2025-11-02 |
| 304 | 304 | 151 | despesa | Educação | 2140.73 | 2025-11-03 |

Total rows: 304 | Query complete 00:00:00.049

Fonte: Capturado pelo autor desse trabalho a partir de PgAdmin 4, 2025.

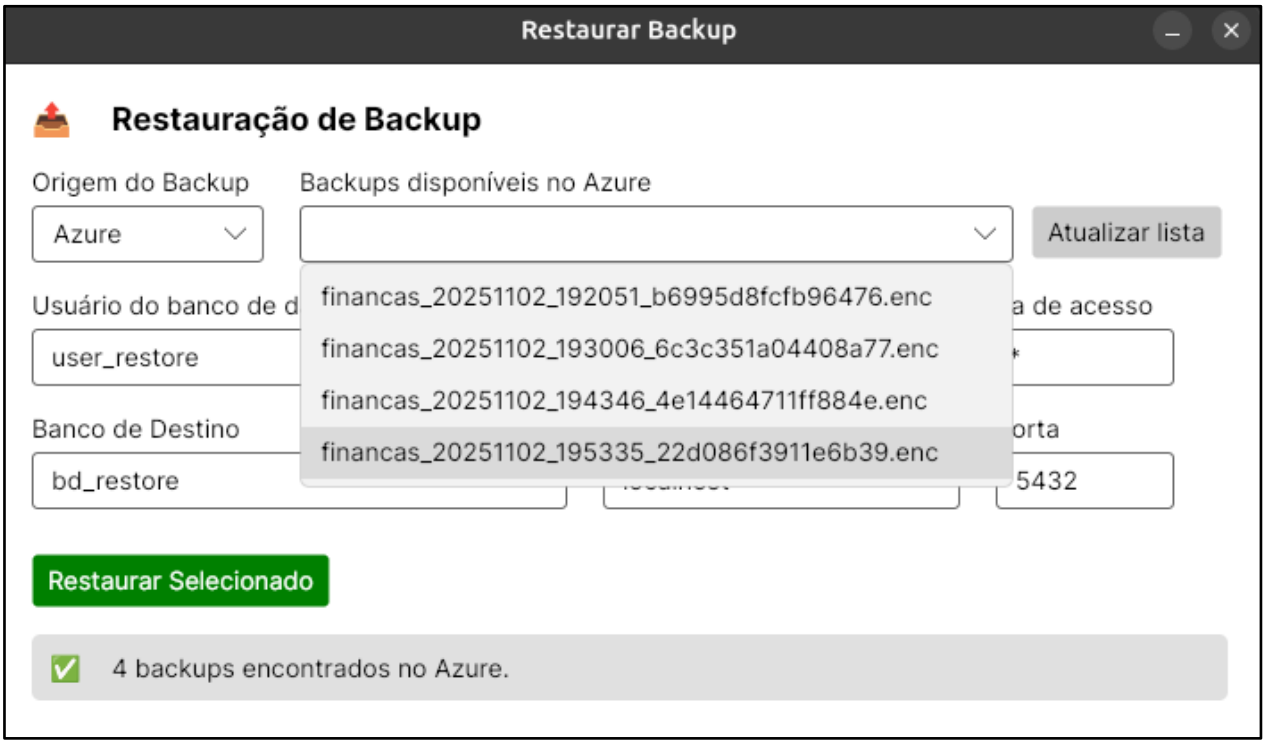
Figura 40: *Backups* armazenados no Azure




Fonte: Capturado pelo autor desse trabalho a partir de Portal Azure, 2025.

Os quatro *backups* ficaram disponíveis no ABS, e a *interface* da aplicação evidenciou corretamente todos os artefatos após o uso da função Atualizar Lista, conforme ilustrado na Figura 41.

Figura 41: Restauração de *backup*




Fonte: Capturado pelo autor desse trabalho a partir do sistema Ubuntu, 2025.

A execução de cada ciclo confirmou:

- Aderência ao fluxo definido na política de *backup*;
- Consistência dos arquivos gerados;
- Correta criptografia e compactação;
- Sucesso no envio para a nuvem;
- Ausência de falhas na remoção dos arquivos temporários.

Esses resultados validam o funcionamento do módulo de *backup* e servem de base para os testes de restauração, apresentados na Seção 5.2 desse trabalho.

## 5.2 Testes de restauração

Os testes de restauração foram realizados com o objetivo de verificar a capacidade do sistema em recuperar o conteúdo armazenado nos *backups* e restaurar corretamente o estado do banco de dados PostgreSQL em uma instância separada, denominada db_restore. Para isso, foram utilizados dois arquivos de *backup* gerados durante a fase de testes descrita na Seção 5.1: o quarto *backup* completo, que representa o estado final do banco de dados, e um *backup* automático produzido pelo agendador Cron, contendo exatamente o mesmo conjunto de dados. Esse *backup* automático, foi restaurado em um ambiente distinto, especificamente o PostgreSQL instalado no Windows 11, demonstrando a capacidade do sistema em recuperar dados de forma consistente mesmo quando o processo ocorre em um sistema operacional diferente daquele onde o artefato foi originalmente gerado. Esses testes permitem avaliar tanto a integridade quanto a precisão dos dados restaurados, considerando diferentes cenários de obtenção e ambientes de destino.

A restauração foi conduzida por meio da *interface* gráfica do aplicativo, selecionando a origem do *backup* como Azure, o arquivo desejado e informando os parâmetros de acesso ao banco de destino. Conforme descrito anteriormente no capítulo de implementação, a restauração requer autenticação explícita, reforçando o controle de acesso e permitindo que ambientes distintos sejam utilizados para fins de teste.

### 5.2.1 Restauração a partir do quarto *backup*

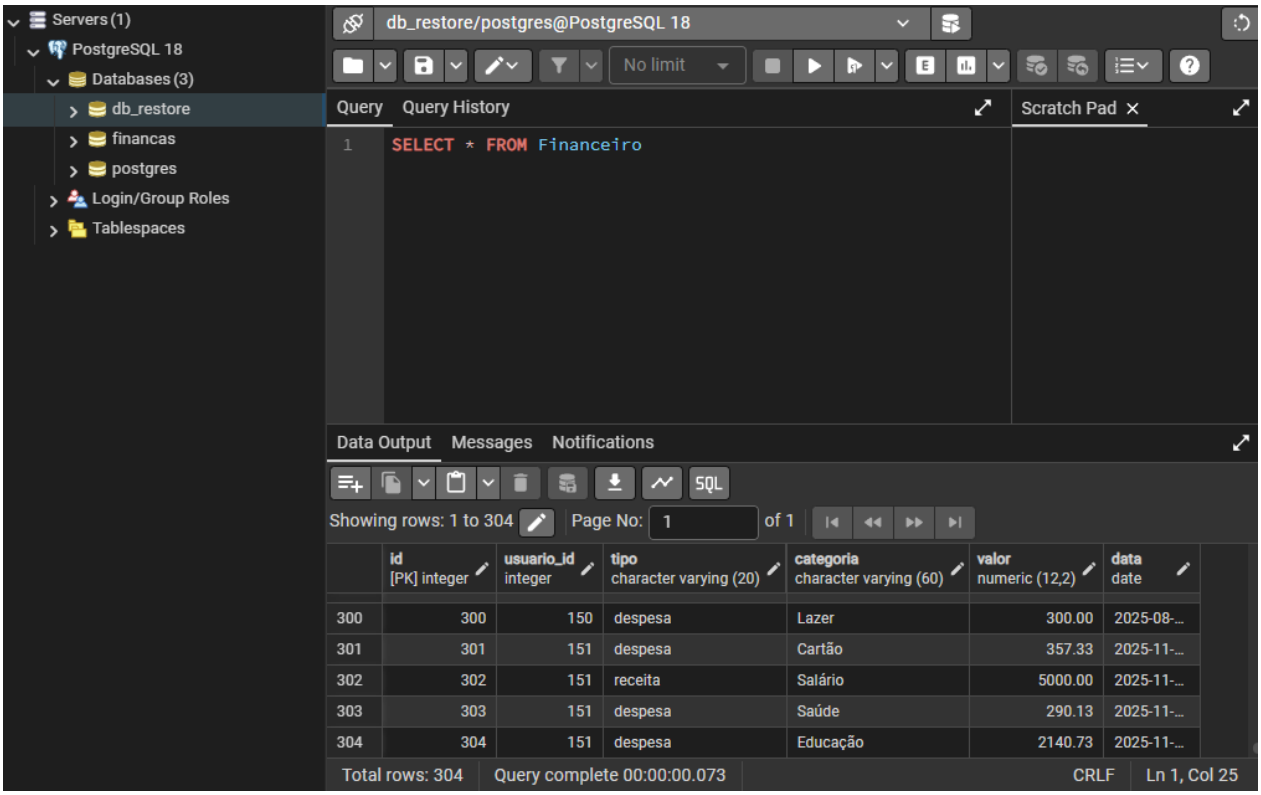
No primeiro cenário, foi selecionado o arquivo correspondente ao quarto *backup* completo, o qual havia sido gerado após a inclusão do usuário Lucas Augusto de Souza, do respectivo cartão associado e dos lançamentos financeiros adicionados no estágio final dos testes. A origem escolhida para este procedimento foi o Azure, utilizando a funcionalidade de listagem e seleção de arquivo disponibilizada pela *interface*, conforme ilustrado na Figura 41.

Após a seleção do arquivo, os dados de conexão com o banco db_restore foram preenchidos e o processo de restauração foi iniciado. O sistema executou o procedimento

de verificação de integridade por meio da comparação do *hash* SHA-256 armazenado no Azure com o *hash* calculado localmente, confirmando que o arquivo não havia sido alterado durante o armazenamento ou a transferência. Em seguida, o utilitário pg_restore foi acionado automaticamente pela aplicação, reconstituindo a base de dados no ambiente de destino.

O resultado obtido após a restauração confirmou a presença de todos os registros esperados: o usuário Lucas Augusto de Souza, o cartão vinculado e as movimentações financeiras adicionadas por último. Dessa forma, o teste demonstrou que o sistema é capaz de reconstruir o estado final da base de dados com precisão, preservando relações e vínculos entre as tabelas envolvidas e garantindo a integridade do processo, ilustrado na Figura 42.

Figura 42: Banco de dados db_restore restaurado



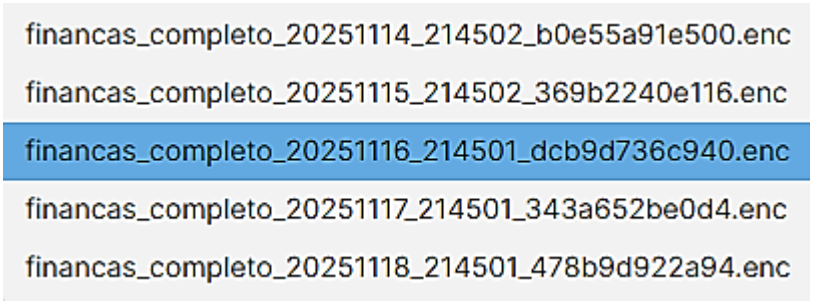

Fonte: Capturado pelo autor desse trabalho a partir de PgAdmin 4, 2025.

### 5.2.2 Restauração a partir do *backup* agendado

O segundo cenário de teste consistiu na restauração de um *backup* automático produzido pelo sistema a partir da rotina agendada via Cron. Diferentemente do procedimento manual descrito no item anterior, esse *backup* foi gerado de forma totalmente automatizada, seguindo a política definida na Seção 4.6 deste trabalho, que estabelece a periodicidade e o formato utilizado na criação dos artefatos. O conteúdo desse *backup* correspondia ao estado final da base de dados no momento da execução agendada, refletindo as mesmas inserções presentes no quarto *backup* completo.

Para iniciar o processo, a origem do *backup* foi definida como Azure, sendo selecionado o arquivo disponibilizado no contêiner remoto, ilustrado na Figura 43. Assim como no cenário anterior, o sistema aplicou a verificação de integridade baseada no algoritmo SHA-256, comparando o *hash* armazenado nos metadados do Azure com o *hash* calculado localmente após o *download*. A correspondência entre os valores confirmou que o arquivo mantinha sua integridade e que poderia ser utilizado com segurança no processo de restauração.

Figura 43: *Backup* automático realizado

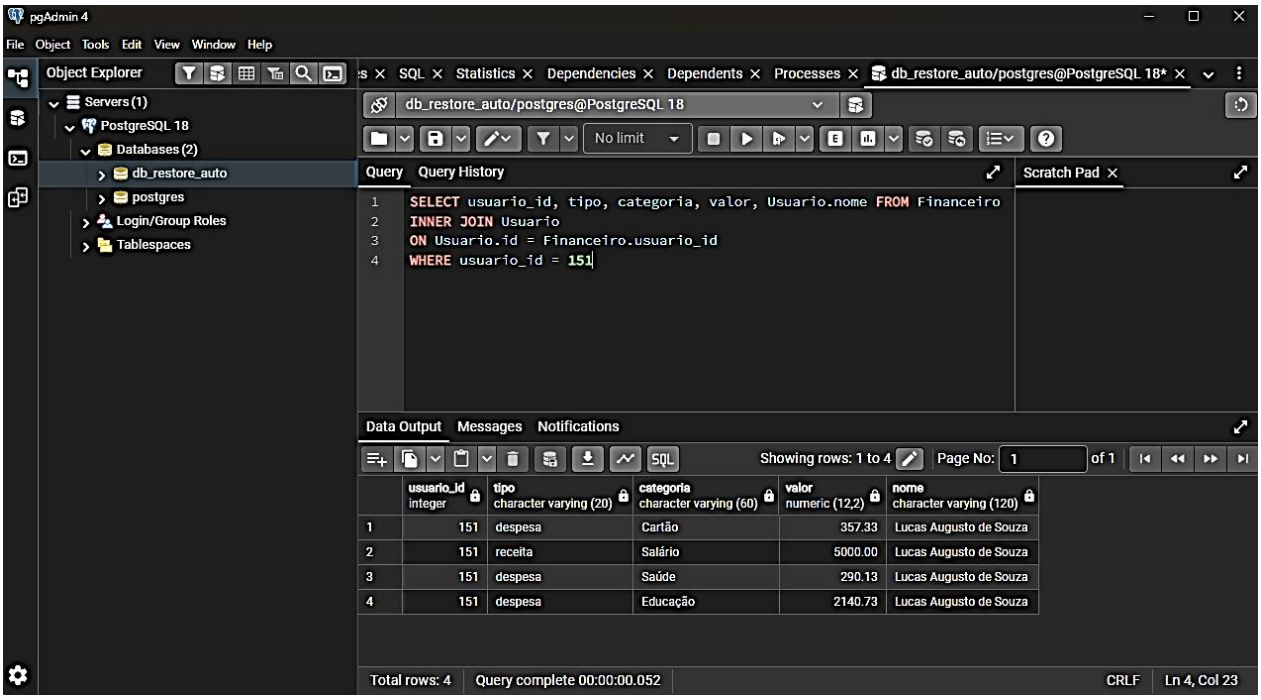



Fonte: Capturado pelo autor desse trabalho a partir de PgAdmin 4, 2025.

Com a integridade validada, o utilitário pg_restore foi acionado pelo aplicativo, reconstituindo a base db_restore_auto, localizada em uma instalação do PostgreSQL executando no sistema operacional Windows 11. Esse procedimento demonstrou que os artefatos armazenados no Azure podem ser restaurados corretamente mesmo em ambientes distintos daquele onde foram originalmente gerados, evidenciando a portabilidade e a independência de plataforma das operações realizadas.

As consultas efetuadas após a restauração mostraram que todas as informações previstas, incluindo o usuário Lucas Augusto de Souza, o cartão associado e as movimentações financeiras adicionadas no último estágio dos testes, estavam presentes e coerentes com o estado original da base no momento da criação do *backup*, conforme ilustrado na Figura 44.

Figura 44: Banco db_restore_auto restaurado no Windows




Fonte: Capturado pelo autor desse trabalho a partir de PgAdmin 4, 2025.

Esse resultado demonstra que o mecanismo de *backup* automático, quando combinado à verificação de integridade e ao processo de restauração guiado pela aplicação, é capaz de recuperar corretamente o estado mais recente do banco de dados. Além disso, evidencia a robustez do fluxo automatizado, garantindo que a política de *backup* definida possa ser aplicada em ambientes reais de forma segura, previsível e com intervenção mínima do usuário.

# 6 CONSIDERAÇÕES FINAIS

O presente trabalho teve como propósito demonstrar a implementação de um sistema de *backup* em nuvem capaz de assegurar a proteção, a integridade e a disponibilidade dos dados armazenados. A solução foi desenvolvida considerando a necessidade crescente de mecanismos confiáveis de salvamento de informações e explorou, de forma prática, o uso de criptografia, validação de integridade e armazenamento em ambiente de nuvem. O sistema proposto integra procedimentos de *backup* completo, envio automatizado ao ABS, verificação de integridade por meio de *hash* criptográfico e restauração assistida pela *interface* gráfica, consolidando um fluxo de proteção de dados que combina simplicidade operacional com práticas amplamente reconhecidas pela comunidade de segurança da informação.

Quanto aos objetivos estabelecidos inicialmente, parte deles foi alcançada com êxito. A aplicação permite realizar *backups* completos, automatizar sua execução por meio do Cron, enviar os artefatos criptografados para a nuvem, verificar a integridade utilizando SHA-256 e restaurar versões anteriores do banco de dados de forma confiável. Esses elementos constituem o núcleo funcional definido no início do projeto. Contudo, algumas metas complementares não puderam ser concluídas dentro do prazo, como a implementação de deduplicação de dados e a criação de rotinas de *backup* incremental e diferencial. Ainda assim, os resultados obtidos atendem ao escopo central proposto e demonstram a viabilidade e a efetividade do sistema desenvolvido.

A metodologia adotada contribuiu de forma concreta para os resultados obtidos. A combinação entre análise de documentação técnica, exploração prática de ferramentas e desenvolvimento iterativo permitiu consolidar uma visão integrada do processo de *backup* e restauração. As dificuldades encontradas ao longo do desenvolvimento foram superadas principalmente por meio de revisões incrementais, testes contínuos e sucessivas validações. Entre os principais desafios, destaca-se a alteração na estratégia de derivação de chave criptográfica, inicialmente baseada em PBKDF2 e posteriormente substituída por SHA-256, o que exigiu adequações e testes adicionais. Além disso, a tentativa inicial de desenvolver um mecanismo de *backup* parcial por tabela mostrou-se mais complexa do que o previsto, sendo interrompida para priorizar funcionalidades

essenciais dentro das limitações de tempo do projeto. Portanto, as correções de rota realizadas contribuíram para manter o foco no objetivo central e assegurar a qualidade das funcionalidades concluídas.

Os resultados apresentados mostraram-se coerentes com a proposta. Os testes realizados comprovaram que o sistema é capaz de produzir *backups* funcionais, enviá-los ao Azure e restaurá-los de forma íntegra por meio do utilitário pg_restore, preservando as relações e a consistência dos dados. A verificação de integridade por meio de *hash* SHA-256 garantiu a confiabilidade dos artefatos armazenados, assegurando que nenhum arquivo alterado ou corrompido fosse utilizado no processo de restauração. Embora algumas limitações persistam, como o armazenamento da senha do banco de dados em variável de ambiente para execução automática (prática funcional, porém não ideal do ponto de vista de segurança), a solução implementada corresponde aos requisitos essenciais propostos e oferece bases sólidas para futuras expansões.

A contribuição deste trabalho se manifesta em diferentes esferas. Do ponto de vista técnico, proporciona uma abordagem prática para a construção de mecanismos de *backup* seguro integrados à nuvem, utilizando tecnologias acessíveis e amplamente empregadas no mercado. Para a comunidade acadêmica, o projeto oferece um estudo aplicado que discute a interação entre criptografia, verificação de integridade, automação e serviços de nuvem, podendo servir como referência para novos estudos na área. Do ponto de vista da sociedade e do setor profissional, apresenta uma solução simplificada e economicamente viável para organizações que necessitam proteger seus dados, reforçando a importância do uso de práticas de segurança mesmo em ambientes de pequeno e médio porte.

## 6.1 Sugestões de trabalhos futuros

Com base no trabalho desenvolvido, destacam-se três direções para pesquisas futuras que podem ampliar significativamente o potencial da solução:

- Implementar *backup* incremental, reduzindo o volume de dados transferidos e otimizando tempo e espaço de armazenamento;

- Implementar *backup* diferencial, possibilitando um equilíbrio entre desempenho e segurança para cenários de atualização frequente de dados;
- Implementar deduplicação de dados, aumentando a eficiência do armazenamento em nuvem ao evitar envio repetido de blocos idênticos entre diferentes versões do *backup*.

Essas melhorias permitirão um sistema mais robusto, eficiente e economicamente vantajoso, com capacidades amplas de gerenciamento de dados e recuperação de desastres.

# REFERÊNCIAS


ASHARE, Matt. ***Cloud spend growth forecast* 2025**: *Global cloud spend to surpass $700B in* 2025 *as hybrid adoption spreads*: Gartner. [*S. l.*]: Gartner, 19 nov. 2024. Disponível em: https://www.ciodive.com/news/cloud-spend-growth-forecast-2025-gartner/733401/. Acesso em: 2 jun. 2025.

BADMAN, Annie; KOSINSKI, Matt. **O que é criptografia assimétrica?**. [*S. l.*]: IBM, 8 ago. 2024. Disponível em: https://www.ibm.com/br-pt/think/topics/asymmetric-encryption. Acesso em: 1 jun. 2025.

BHALERAO, Anand; PAWAR, Ambika. ***A survey: On data deduplication for efficiently utilizing cloud storage for big data backups***. Tirunelveli, India: IEEE, 12 maio 2017. Disponível em: https://ieeexplore.ieee.org/document/8300844. Acesso em: 18 abr. 2025.

BRASIL. **Ministério da Ciência, Tecnologia e Inovações. Os quatro pilares da segurança da informação: Confidencialidade, Disponibilidade, Integridade e Autenticidade.** [S. l.], 8 ago. 2024. Disponível em: https://www.gov.br/lncc/pt-br/centrais-de-conteudo/campanhas-de-conscientizacao/gestao-de-seguranca-da-informacao/os-quatro-pilares-da-seguranca-da-informacao-2013-confidencialidade-disponibilidade-integridade-e-autenticidade. Acesso em: 30 abr. 2025.

GOOGLE CLOUD. **O que é a criptografia?** [S. l.], 2024. Disponível em: https://cloud.google.com/learn/what-is-encryption?hl=pt-BR. Acesso em: 25 abr. 2025.

JAMES, Mike. ***Avalonia Docs - What is Avalonia?***. [*S. l.*]: IBM, 30 dez. 2024. Disponível em: https://docs.avaloniaui.net/docs/overview/what-is-avalonia. Acesso em: 5 set. 2025.

KAVIS, Michael. ***Architecting the Cloud:*** *Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*. Wiley, 2014.

MICROSOFT. **Documentação do Armazenamento de Blobs do Azure**. Disponível em: https://learn.microsoft.com/pt-br/azure/storage/blobs/. Acesso em: 21 de ago. de 2025a.

MICROSOFT. **Documentação do Azure**. Disponível em: https://learn.microsoft.com/pt-br/azure/. Acesso em: 20 de ago. de 2025b.

MICROSOFT. **Documentação do .NET**. Disponível em: https://learn.microsoft.com/pt-br/dotnet/. Acesso em: 28 de ago. de 2025c.

NOGUEIRA, Alexandre. **Criptografia**: o que é e como ela funciona?. [*S. l.*]: HostGator, 4 dez. 2020. Disponível em: https://www.hostgator.com.br/blog/o-que-e-criptografia-e-quando-usar/. Acesso em: 29 maio 2025.

OPUS SOFTWARE**. O Que Você Realmente Precisa Saber Sobre Computação Em Nuvem**. Opus Software Com. e Repr. Ltda., 2015.

POSTGRESQL. **Documentation – PostgreSQL 17**. Disponível em: https://www.postgresql.org/docs/17/backup.html. Acesso em: 10 de set. de 2025.

PRESTON, W. Curtis. ***Modern Data Protection****: Ensuring Recoverability of All Modern Workloads*. 1. ed. Sebastopol, CA: O'Reilly Media, 2021. Disponível em: https://www.amazon.com/_/dp/1492094056?smid=ATVPDKIKX0DER&_encoding=UTF8. Acesso em: 20 abr. 2025.

SSL STORE. ***What is Hashing Algorithm & How Does It Work*?:** *What's the Hash Function?.* [S. l.], 2025. Disponível em: https://aboutssl.org/what-is-*hash*ing-algorithm-how-does-it-work/. Acesso em: 14 maio 2025.

STALLINGS, W.; BROWN, L. **Segurança de Computadores**. Elsevier, 2014.

Pontifícia Universidade Católica de Goiás
*Pontifical Catholic University of Goias*
Av. Universitária, 1069, Setor Universitário
Caixa Postal 86 - CEP 74.605-010
Goiânia – Goiás – Brasil

**RESOLUÇÃO n° 038/2020 – CEPE**

**ANEXO I**

APÊNDICE ao TCC

**Termo de autorização de publicação de produção acadêmica**

O estudante Lucas Augusto de Souza do Curso de Ciência da Computação matrícula 20242002800567, telefone: 62 98464-7148 e-mail lsouza.core@gmail.com, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado Serviço de backup em nuvem

_____

_____

_____, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 02 de outubro de 2025.

Assinatura do autor: _____ gov.br Documento assinado digitalmente LUCAS AUGUSTO DE SOUZA Data: 02/10/2025 23:06:01-0300 Verifique em https://validar.iti.gov.br

Nome completo do autor: Lucas Augusto de Souza

Assinatura do professor-orientador: _____ gov.br Documento assinado digitalmente ANGELICA DA SILVA NUNES Data: 18/12/2025 18:51:15-0300 Verifique em https://validar.iti.gov.br

Nome completo do professor-orientador: