Hyper-Scale Managed Identities and Access Control

Ivan Alagenchev[1], Shobhit Trehan[2], Kang Gui[1], Dragos Avadanei[2],
Raghavendra Kammara[1], Will Bartlett[1], Rajat Jain[2], Raghav Kaushik[2]
[1]Microsoft Identity
[2]Microsoft Cloud and AI

Abstract

Modern distributed systems demand scalable, low-latency, and flexible authorization, yet conventional directory-based Identity and Role-Based Access Control (RBAC) frameworks remain overly centralized and inflexible, falling short of the requirements posed by dynamic, distributed service-to-service (S2S) architectures. These legacy models lack the scalability, adaptability, and fine-grained, context-aware control essential for decentralized authentication and authorization at hyper-scale. This paper presents the design and architecture of Hyper-Scale Managed Identities (HSMIs) and their integration with decentralized access control policies. We analyze the limitations of traditional identity models and outline the HSMI architecture, including processes for attestation, credential issuance, binding, and the enforcement of access control policies. In addition, we introduce Attested Credential Release and its associated security guarantees. The structure of authorization tokens is presented, emphasizing their role in enabling hyper-scale, ultra-low latency scenarios. Further, we detail a capability-based authorization model that utilizes authorization tokens, attribute tokens, and namespacing to facilitate decentralized, high-performance access control.

1 Introduction

Internal services within cloud platforms require secure, reliable mechanisms to communicate with each other. For example, within Azure, data services such as Azure SQL [19] must interact with Azure Storage [20] to manage database files. Although both components belong to the same provider and operate within a shared trust boundary, robust authentication between services remains essential. This necessity stems from the principle that, even in integrated environments, each service must safeguard against lateral compromise should a breach occur elsewhere. Consequently, cloud architectures typically avoid distinguishing between first-party and third-party callers, instead enforcing authentication to ensure that vulnerabilities in one service do not proliferate across the platform.

The authentication model within cloud services reflects the complexity of their internal structure. Rather than treating a service as a monolithic entity, authentication is applied at the data-plane level: individual SQL instances and Storage containers each authenticate independently. With millions of such instances and containers operating simultaneously in large-scale environments, establishing and maintaining secure authentication relationships presents significant challenges. One commonly employed approach is the use of Shared Access Signatures (SAS) [17], whereby the Storage service uses account-specific keys to derive container-level SAS credentials. The many-to-many mapping between SQL instances and storage containers, necessitate millions of unique SAS credentials distributed across the platform. To mitigate risks, storage access keys are regularly rotated, and SAS credentials regenerated; however, this practice creates substantial operational overhead and places significant demands on the Storage service itself.

Reliance on static access keys for authentication, while straightforward, introduces limitations in both security and scalability. Static credentials behave like passwords—susceptible to theft, reuse, and

misuse—and pose ongoing security risks due to the challenges of managing and rotating them effectively. To address these issues, cloud platforms have adopted mechanisms such as Managed Identities(MIs) [14]. Under this model, a central authentication service—Microsoft Entra [18] (formerly Azure Active Directory), for instance, issues identities to individual service instances. These identities are then used to obtain short-lived access tokens, which replace static keys and offer enhanced security thanks to their limited validity. By shifting the authentication burden to a centralized authority, the MI model reduces operational complexity for service teams and mitigates the risks associated with static credentials.

Despite the advantages of Managed Identities, this approach is not without its own scalability challenges. The central authentication service must maintain an extensive directory of service identities, effectively representing a closed world of continuously changing authenticating entities. As the number of service instances scales into the billions, directory management becomes increasingly difficult, and practical limitations emerge. Furthermore, the certificate-based process for establishing MIs retains some of the operational drawbacks seen in access key rotation, such as management overhead and scheduling for periodic renewal.

In response to these persistent challenges, we propose a novel authentication method designed to enhance both security and scalability. Our approach reimagines service authentication by basing it on the health of the hosting Virtual Machine [21] (VM), leveraging hardware roots of trust such as Trusted Execution Environments [16] (TEEs)—for example, Trusted Platform Module (TPMs) or enclaves—to attest to VM integrity. Rather than verifying identities against a directory, the central authentication service validates health reports signed by keys rooted in the TEE, rendering the process stateless and highly scalable. This technique not only mitigates the risks associated with static secrets and directory-based bottlenecks but also introduces a dynamic, context-aware security posture. We illustrate the improvement in security posture through an example. In the previous approaches discussed above, suppose a malicious actor compromises a cloud VM, for instance, by installing malware on the VM. The malware would be potentially undetected at least until all secrets on the VM are rotated—this could mean days or weeks of compromise. In contrast, with the dynamic approach based on secure booting, the presence of the malware would be measured by the TPM and hence fail an attestation check, reducing the period of compromise.

The authenticating party (SQL in the above example) also needs the right permissions to access resources (Storage in the above example). Hence, there is the need to solve an authorization problem based on the same stateless infrastructure. As an example, data services like Azure SQL manage data across millions of storage accounts, with each individual storage account capable of hosting thousands of containers. Given that data service instances (for example, SQL instances) maintain complex many-to-many relationships with storage containers to optimize both resource utilization and packing, the volume of mappings between SQL instances and storage containers increases substantially. Modelling this authorization pattern through centralized directory services for resolution would necessitate billions of role assignments per subscription and surpass the practical limits for both role assignment storage and evaluation. We also describe our Attribute Based Access Control solution to the authorization problem.

The proposed architecture involves calling multiple components, which on the face of it, poses a performance challenge, since calling multiple components in the data path is expensive. However, we use the notion of tokens and prefetching to ensure that the data path in the hot case does not call all components. This allows us to approach our goal of sub-millisecond latencies in the 99th percentile (P99), which is comparable to the SAS and MSI methods described above. Therefore, the improved security and scalability of HSMI does not carry a performance price.

Azure is currently piloting this technology in production, with plans for broad deployment across its ecosystem. We believe this evolution marks a significant step forward for service-to-service authentication in cloud environments, setting a new standard for secure and scalable operations.

2 Background and Terminology

2.1 Microsoft Entra

Microsoft Entra [18] (formerly Azure Active Directory) is a family of identity and network access products. It lets organizations implement a security strategy and create a trust fabric that verifies identities, validates access conditions, checks permissions, encrypts connection channels, and monitors for compromise.

2.2 Azure SQL

Azure SQL [19] is a Platform as a Service (PaaS) offering built upon the SQL Server engine, hosted on Azure Service Fabric. This architecture features separated compute and storage layers, with Azure Storage used for the persistence of database files, such as log (ldf) and data (mdf) files. The service incorporates a control plane, hereafter referred to as the SQL Control Plane, which oversees the lifecycle management of various databases. Additionally, the service comprises a data plane, representing the sqlservr.exe processes running on compute nodes within the Service Fabric cluster. In this context, the term 'SQL Instance' denotes a single sqlservr.exe process. Although the Azure SQL service supports multiple storage configurations, the approach discussed in this paper is exemplified by a SQL database that utilizes Azure Blob Storage containers to host its data. Azure SQL runs on Azure Service Fabric [22], Microsoft's distributed systems platform for deploying and managing microservices and containers across a cluster of machines. In this context, a cluster is the overall set of machines that run Service Fabric, while a node refers to an individual compute unit (typically a virtual machine in Azure) within that cluster.

2.3 Managed Identities (MI)

Azure provides automatically managed identities [14] that can be assigned to compute resources such as Virtual Machines, Virtual Machine Scale Sets, Azure Kubernetes Service clusters, or supported application hosting platforms. These identities allow services to authenticate to Azure resources without the need for developers to provision or manage secrets, credentials, certificates, or keys, thereby reducing the risk of credential exposure and simplifying secure communication between services.

2.4 SPIFFE (Secure Production Identity Framework for Everyone)

An open-source framework [10], the Secure Production Identity Framework for Everyone (SPIFFE), provides a standard for securely issuing and managing identities for services in a cloud-native or distributed system to enable secure service-to-service authentication and workload identity across heterogeneous environments. A SPIFFE ID is a unique identity assigned to a workload (e.g., a service or process). A SPIFFE Verifiable Identity Document (SVID) is the cryptographic credential—typically an X.509 certificate or JWT—that proves possession of a SPIFFE ID and is used by workloads to authenticate to each other securely.

2.5 Hyper-Scale Managed Identities (HSMIs)

A novel identity framework designed for cloud-native environments that decouples identity from centralized directories and leverages hardware attestation for scalable authentication. Each HSMI is uniquely identified by its SPIFFE identifier and we use the verb 'SPIFFE identifier' or 'HSMI identifier' interchangeably in this paper.

2.6 Trusted Execution Environments (TEEs)

A Trusted Execution Environment [16] is a segregated area of memory and CPU that's protected from the rest of the CPU by using encryption. Any code outside that environment can't read or tamper with the data in the TEE. Authorized code can manipulate the data inside the TEE.

2.7 Resource Management Authorities (RMAs)

Resource Management Authorities (RMA)s are compute orchestration engines such as Kubernetes and Service Fabric, a commonly used orchestration engine within Azure. Typically, RMAs manage VM capacity by bundling them into clusters. Within each cluster, a process or container running the application (SQL in our example) is launched. RMAs attest to a centralized credential authority regarding the validity of the resources they manage. Consequently, the credential authority issues HSMI credentials. Traditionally, identity providers (IdPs) issue identity tokens for entities within their administrative domain; however, in this model, the resource providers do not issue identity tokens themselves. Instead, responsibility for token issuance is delegated to a centralized service, which retains control over the signing keys trusted by the broader application ecosystem. Thus, these resource providers manage the lifecycle of the resources under their supervision and are referred to as Resource Management Authorities (RMAs) to distinguish them from traditional IdPs. RMAs assign identifiers to each application instance. Resource identifiers follow a hierarchical format compatible with SPIFFE standards, with each identifier prefixed by the RMA's reserved namespace. Notably, scaling the RMAs to handle the number of resources they manage is a non-optional requirement. Furthermore, RMAs possess an intimate awareness of partitioning opportunities and constraints that their topology may impose, which is often more nuanced than the topology managed by Microsoft Entra. The hierarchical structure encodes resource-specific metadata, thereby enabling logical segmentation of the resource domain into distinct security zones. In addition, it facilitates efficient bulk operations such as credential and token revocation.

```
spiffe://<internal-prefix>/s/10ef5b45-a7e5-4f96-9d11-90e8b5e06a87/rg/test-eus-rg/sf/test-eus-cluster
    /7af6ddcc-8407-427d-ac61-5a47a0ea8e00/SqlApplicationType/SqlApplicationName
```

In the above example, let's consider a SQL instance (process) running on an Azure Service Fabric (RMA) cluster, the resource identifier encodes the subscription '10ef5b45-a7e5-4f96-9d11-90e8b5e06a87', the resource group 'test-eus-rg' and the cluster 'test-eus-cluster', followed by a unique identifier of the cluster i.e. '7af6ddcc-8407-427d-ac61-5a47a0ea8e00'. These values represent the service fabric cluster where this SQL instance is being hosted. The next two values 'SqlApplicationType' and 'SqlApplicationName' encode the SQL instance and represent the sqlservr.exe process running on the compute instance. This resource identifier uniquely identifies an HSMI.

2.8 Attribute-Based Access Control (ABAC)

An access control model that evaluates permissions based on attributes of the subject, resource, and environment, enabling more flexible and context-aware authorization decisions [15]. There are three types of attributes we highlight—Principal Attributes, Resource Attributes and Environmental Attributes. In a distributed setting, Principal Attributes describe the identity which is requesting access e.g., tenant, environment or service role. In our approach, HSMI principals are directory-less, and hence, we describe principal attributes with an attribute token bound to particular HSMI. We introduce this in Section 2.11. Environmental Attributes represent the contextual information about the access request e.g. time of day, IP address, device posture etc. Lastly, Resource Attributes are associated with the resource which is being accessed e.g. Azure SQL might access Azure Storage as a resource and resource attribute in this flow could be Storage Blob Container Metadata—a key value pair that stores properties of the

container. As an example, let's consider the storage account container 'mycontainer' in the storage account 'mystorageaccount', and that the container hosts data for a SQL Server, managed by SQL Control Plane, with a server identifier 'b3f2c9d4-8a7e-4f1a-9d3b-7e6c2a1f5e8d'. The resource attributes, stored in the metadata can be—attribute key:readAccessGroups and attribute value: b3f2c9d4-8a7e-4f1a-9d3b-7e6c2a1f5e8d. This example is for illustration and we explain how to interpret and use these values in Section 3.

2.9 Attribute Authority

The Attribute Authority manages and issues cryptographically signed attribute tokens that bind attribute values to their respective HSMI instances. Attribute tokens are signed by a trusted central signing authority, ensuring that attribute modifications are prevented even in the event of a compromised HSMI instance. Authorized control planes, such as the SQL control plane, that manages the lifecycle of SQL databases running on hosting platforms or RMAs like Service Fabric, possess the ability to request attribute tokens for HSMI identities under their management from the attribute authority. Management of the attribute authority service is conducted by the authorization authority, guaranteeing consistency and robust security.

2.10 Attribute Namespace

Hyperscale identity systems require a decentralized approach to attribute management to avoid bottlenecks and single points of failure. To facilitate this, the Attribute Authority employs a namespacing model that enables decentralized attribute management by control planes. The control plane owns the attribute names and is responsible for defining the association between attributes and HSMI identities. Each attribute is uniquely identified within a namespace owned by the control plane. This design supports horizontal scaling and reduces state management overhead while maintaining clear ownership boundaries. Namespace ownership is non-transferable and ensures that only authorized control planes can attest to attributes within their designated scope. An example of a namespace owned by the SQL Control Plane in East US region, for principal attributes assigned to SQL databases in that region is—`Microsoft.AttributeAttestation/AttributeNamespaces/SqlEus`

2.11 Attribute Tokens

Attribute tokens encapsulate principal attributes—such as tenant, environment, and service role—in a cryptographically signed format. Issued by the Attribute Authority, these tokens function as inputs to Attribute-Based Access Control (ABAC) systems. Unlike traditional Microsoft Entra directory-based group memberships, attribute tokens are both portable and verifiable. The Attribute Authority addresses a fundamental challenge presented by directory-less identity architectures: securely binding attributes to HSMI principals that lack directory representation. As HSMIs do not inherently support principal attributes necessary for ABAC grouping, the Attribute Authority furnishes a secure mechanism for delegating attribute association to authorized control planes, while preserving robust cryptographic security guarantees. The format of such an attribute token is:

```
{
   "sub": "<HSMI_ID>",
   "xms_attr": {
      "Microsoft.AttributeAttestation/AttributeNamespaces/{namespace}/Attributes/{attribute1}": <
          Value1>,
      "Microsoft.AttributeAttestation/AttributeNamespaces/{namespace}/Attributes/{attribute2}": <
          Value2>,
   },
   "xms_acb": "<ACB_value_to_bind_to_auth_token>",
```

```
}
```

Let's consider the same example as Section 2.8, a SQL Server, managed by SQL Control Plane, with a server identifier 'b3f2c9d4-8a7e-4f1a-9d3b-7e6c2a1f5e8d'. This SQL Server will host multiple SQL instances in the data plane scattered across various Compute Nodes and Azure Service Fabric clusters. We take the example of a single instance named 'SqlApplicationName', also referred in Section 2.7. The below attribute token encodes namespaced attributes assigned to the HSMI for this instance. The 'sub' claim identifies the HSMI, and 'xms_attr' claim holds namespaced attribute key-value pairs. This attribute value helps SQL Control plane maintain a logical grouping over all the SQL Instances hosted on various RMAs. We further explain how to interpret these values in Section 3 and Section 5. Additionally, we explain the 'xms_acb' claim in Section 6.1.

```
{
   "sub": "<internal-prefix>/s/10ef5b45-a7e5-4f96-9d11-90e8b5e06a87/rg/test-eus-rg/sf/test-eus-
       cluster/7af6ddcc-8407-427d-ac61-5a47a0ea8e00/SqlApplicationType/SqlApplicationName",
   "xms_attr": {
      "Microsoft.AttributeAttestation/AttributeNamespaces/SqlEus/Attributes/readAccessGroups": "
          b3f2c9d4-8a7e-4f1a-9d3b-7e6c2a1f5e8d",
   },
   "xms_acb": "<ACB_value_to_bind_to_auth_token>",
}
```

2.12 Policy Decision Point - Remote (PDP-R)

A distributed authorization service that evaluates access policies and issues capability tokens based on principal attributes and resource permissions. It is designed to evaluate access control decisions remotely, rather than locally within the resource itself.

3 Architecture Overview

To understand the architecture of HSMIs, we consider an example flow of the Azure SQL service provisioning a database. In contemporary platforms such as Microsoft Azure, services like Azure SQL Database interact directly with Azure Storage to manage data and log files, thus ensuring transactional consistency and enabling crash recovery in the event that the compute VM crashes, without replication. Authentication is enforced at the data-plane, with each SQL instance and storage container handling authentication and authorization autonomously. SQL service instances are instantiated by the Data Control Plane on a hosting platform such as Service Fabric (the Resource Management Authority or RMA). Logical grouping of SQL instances is orchestrated at the server level, with each group distinguished by a unique serverId, typically in GUID format. These instances may operate across multiple clusters or migrate between them, while storage containers are shared among instances that belong to the same logical server group. In this example, access control can leverage the serverId as an attribute, linking instances and containers within the same group.

As seen in Figure 1, the process commences with the regional SQL Control Plane claiming an attribute namespace by interfacing with the Attribute Authority. For example a regional SQL Control Plane in East US might claim the namespace `Microsoft.AttributeAttestation/AttributeNamespaces/SqlEus`. Once this namespace is registered, only the originating control plane is authorized to obtain attributes from it, thereby establishing strict authorization boundaries. Storage subscriptions are maintained by the control plane, which provisions a conditional role assignment on each such subscription—only those identities possessing attributes from a specific namespace are permitted access to storage containers tagged with matching metadata attributes within a subscription scope. This configuration is a necessary

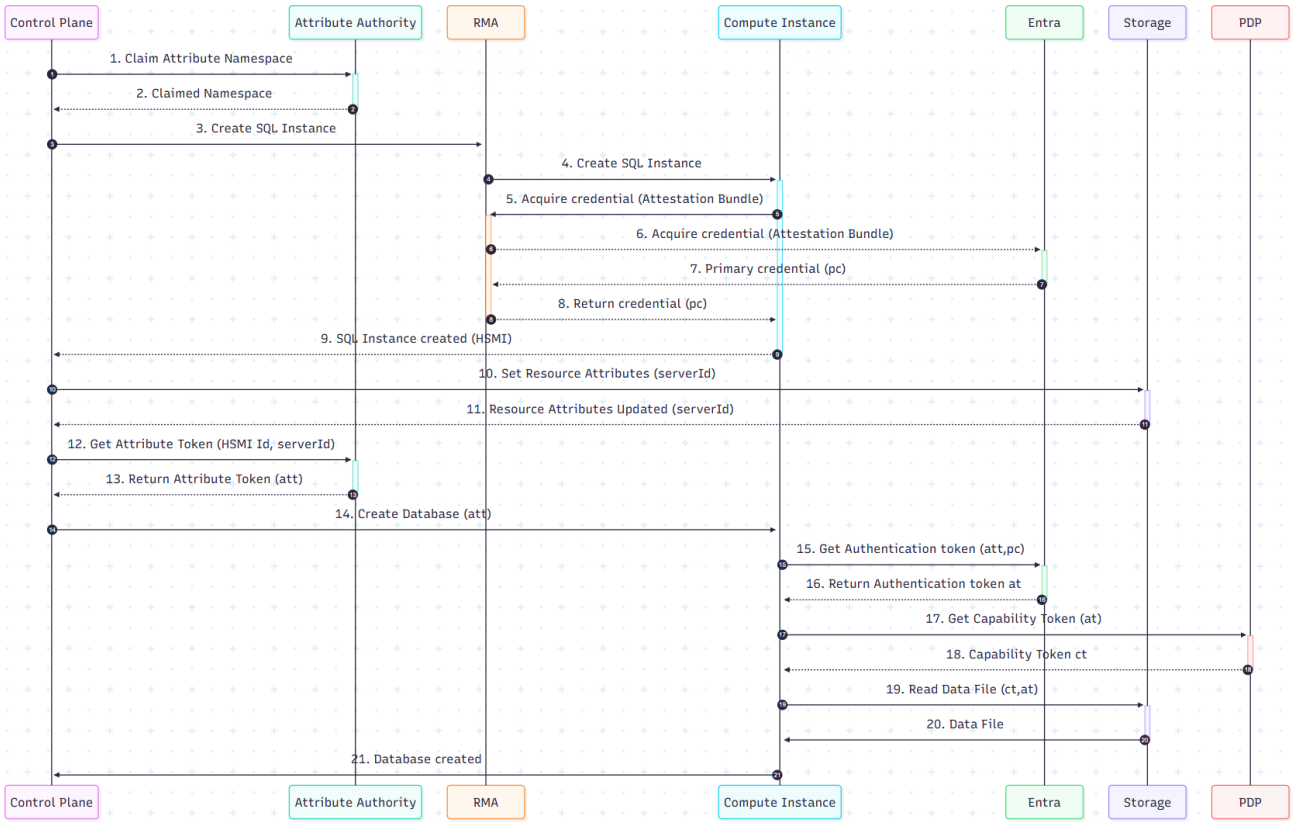

Figure 1: Example End-to-End Flow

precursor for SQL instances to access storage resources based on attributes. An example of such a role assignment, further described in Section 6 is—

```
{
   "Id": "b69889ff-3281-4ffa-8f75-6d7bd3be6616",
   "RoleDefinitionId": "ba92f5b42d11453da403e96b0029c9fe",
   "PrincipalId": "00000000-0000-0000-0000-000000000000",
   "Scope": "/subscriptions/f984cbdd-9e7e-4b97-9744-5c5d9295e332",
   "Condition": "@Principal[Microsoft.AttributeAttestation/AttributeNamespaces/SqlEus/attributes/
       readAccessGroups] ForAnyOfAnyValues:StringEqualsIgnoreCase SplitString{@Resource[Microsoft.
       Storage/storageAccounts/blobServices/containers/metadata:readAccessGroups]}"
}
```

Upon initiation of database creation, the SQL Control Plane instructs the Resource Management Authority (RMA), Service Fabric, to create a new SQL instance, marking eligibility for Hyperscale Managed Identity (HSMI). The RMA delegates provisioning to the compute instance, which then acquires a credential for the newly created HSMI—cryptographically bound to the resource instance. We explain credential issuance in Section 4. After acquiring the credentials, the compute instance returns an HSMI identifier back to SQL Control Plane—

```
spiffe://<internal-prefix>/s/10ef5b45-a7e5-4f96-9d11-90e8b5e06a87/rg/test-eus-rg/sf/test-eus-cluster
    /7af6ddcc-8407-427d-ac61-5a47a0ea8e00/SqlApplicationType/SqlApplicationName
```

In the example above, the identifier encodes the subscription '10ef5b45-a7e5-4f96-9d11-90e8b5e06a87', the resource group 'test-eus-rg' and cluster name 'test-eus-cluster', followed by a unique identifier of the cluster i.e. '7af6ddcc-8407-427d-ac61-5a47a0ea8e00'. These values represent the service fabric cluster where this SQL instance is being hosted. The next two values 'SqlApplicationType' and 'SqlApplicationName' encode the SQL instance and represent the sqlservr.exe process running on the

compute instance. Subsequent to instance provisioning, the SQL Control Plane designates appropriate storage containers for the instance and marks their metadata with the 'serverId' attribute, thus enabling access for all instances within the same logical grouping. As an example let's consider a new SQL database 'testdb' being provisioned in a server 'testsvr', and SQL control plane identifies 'testsvr' with a GUID 'b3f2c9d4-8a7e-4f1a-9d3b-7e6c2a1f5e8d'. SQL Control Plane will update designate a storage account (e.g. 'myaccount') and container (e.g. 'mycontainer') and update the container's metadata—

```
PUT https://mystorageaccount.blob.core.windows.net/mycontainer?comp=metadata
x-ms-meta-readAccessGroups: b3f2c9d4-8a7e-4f1a-9d3b-7e6c2a1f5e8d
```

The next operational step involves obtaining an encrypted attribute token from the Attribute Authority within the established namespace, embedding the same serverId. An example of such an attribute token is—

```
{
   "sub": "<internal-prefix>/s/10ef5b45-a7e5-4f96-9d11-90e8b5e06a87/rg/test-eus-rg/sf/test-eus-
       cluster/7af6ddcc-8407-427d-ac61-5a47a0ea8e00/SqlApplicationType/SqlApplicationName",
   "xms_attr": {
      "Microsoft.AttributeAttestation/AttributeNamespaces/SqlEus/Attributes/readAccessGroups": "
          b3f2c9d4-8a7e-4f1a-9d3b-7e6c2a1f5e8d",
   },
   ...
}
```

This token is securely stored in a key-value repository accessible by the SQL instance. During the database bootstrapping phase, the SQL instance utilizes its primary credential and the attribute token to request an authentication token from Microsoft Entra. We describe this authentication token flow with the primary credential and attributes in Section 5. The SQL instance subsequently calls the Authorization Service and Policy Decision Point (PDP) to retrieve a capability token containing precomputed authorization decisions. We describe the capability token based authorization model further in Section 4. Armed with authentication and authorization tokens, the SQL instance is authorized to interact with storage to retrieve or update data files as required. Note that for illustration we use the example of 'Read' permissions, and the same principle can be applied to 'Write' permissions.

4 Primary Credential Issuance

The credentialing process begins when a resource owner initiates the creation of a resource and designates it as eligible for HSMI identity. At this stage, the RMA assigns an HSMI identifier to the resource and records it within its internal registry. Credential issuance then produces artifacts—such as JWTs [12], X.509 certificates [13], and SPIFFE SVIDs [10]—that are cryptographically tied to a specific attested resource instance. Attestation serves as a key security primitive in the credential issuance lifecycle, acting as the mechanism by which the integrity of a compute instance is verified. During startup, the virtual Trusted Platform Module, or vTPM, records cryptographic measurements of the firmware, bootloader, operating system kernel, and critical system configuration into Platform Configuration Registers. These measurements are digitally signed using a hardware-backed root of trust, producing attestation evidence that is then submitted to the Microsoft Azure Attestation service. The attestation service maintains minimal policy state, which defines the expected secure configuration of the virtual machine, including legitimate software versions, enabled code integrity checks, and secure boot settings. The service verifies that the measurements conform to this policy and, if so, issues a signed attestation token.

As seen in Figure 2, upon resource creation, the RMA, assigns the compute resource a compute unit ID and an access ID. In step 2, a per-boot proof-of-possession (PoP) key is generated within protected memory. In step 4, an attestation token is retrieved from a trusted attestation provider by providing an attestation evidence. This attestation token is cryptographically bound to the per-boot key. This

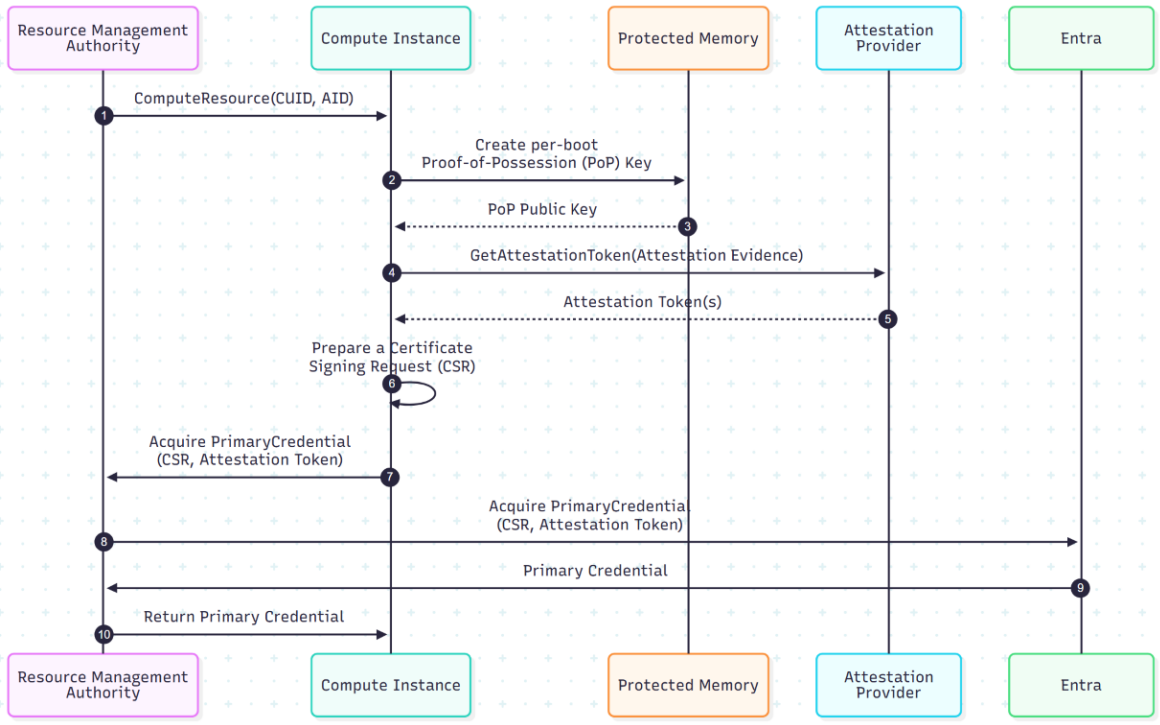

Figure 2: Credential Issuance Flow

token, along with the PoP public key, is returned to the Compute Instance in step 5. Reboots require re-attestation. Steps 2 and 4 occur within a protected execution environment (e.g., a secure enclave) during the resource's bootstrap phase. In step 8, the RMA verifies that the attestation token and the compute unit identifier (e.g., VMID or equivalent) correspond to a trusted resource in its registry and then requests a primary credential from the credential authority on behalf of the resource. This request includes the attestation bundle (comprising tokens from various attestation providers), the device binding key, a Certificate Signing Request (CSR) the resource creation timestamp, and two identifiers: (1) the HSMI ID and (2) a platform-specific resource ID. Both identifiers are included to maintain compatibility with existing access control policies. If either identifier changes, the RMA must issue a new assertion and request a new primary credential. Authorization assignments tied to the old identifier are migrated to the new one using a three-step process: (1) create new, (2) inherit from old, and (3) deprecate old. Upon successful validation of the RMA's assertion, the attestation tokens, and the namespace ownership of the HSMI, the identity authority issues a primary credential (e.g., JWT [12], X.509 [13], SPIFFE SVID [10]) that is cryptographically bound to the resource. This credential includes the finalized HSMI ID and the resource creation timestamp. In step 10, the primary credential is returned to the resource, which can then use it along with the proof of possession (PoP) of the binding key—to request access tokens from the credential authority.

5 Authentication

Once a resource has obtained its primary credential, it can request access tokens from the credential authority to authenticate to other services. These access tokens must be bound to the compute instance to ensure highest level of security is maintained. In Figure 3, Step 1, the compute instance uses the

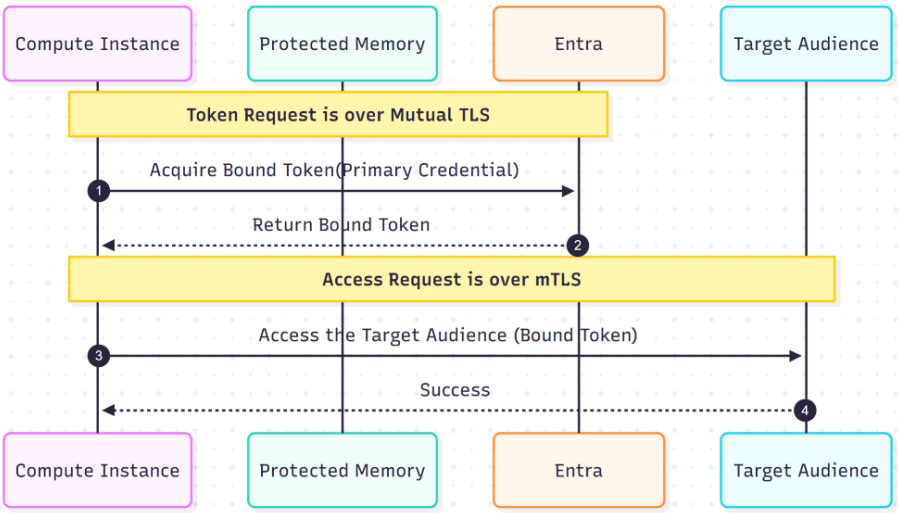

Figure 3: Token Issuance and Usage Flow

bound credential obtained in the previous step for acquiring a bound authentication token. Microsoft Entra token issuance service validates the credential and Proof-of-Possession of the private key used to obtain the primary credential. Once all the necessary validations are done, Microsoft Entra returns a bound token to the compute instance. In step 3, the compute instance uses this bound token to access the target audience i.e. Azure Storage.

As described in Section 2.11, the SQL control plane requests a signed attribute token from the Attribute Authority for a given HSMI and its associated attributes. The Attribute Authority validates that the SQL control plane possesses appropriate access rights, subsequently generating and returning a cryptographically signed attribute token containing the HSMI ID and requested attributes. This token is stored by the SQL instance for use in subsequent authentication requests. In the authentication token request to Microsoft Entra, the SQL instance provides the signed attribute token; Microsoft Entra verifies the signature and federates the attributes as claims in the resultant authentication token. This mechanism securely binds principal attributes to the authentication context. HMSIs can obtain attribute tokens from many attribute authorities and present them to Entra ID in a single authentication request. The resulting authentication token contains all attributes, segregated by their respective attribute authority.

In the example below we illustrate an authentication token obtained for the example HSMI described in Section 3. The claims 'cnf' and 'xms_tbflags' are used for token binding, and the 'xms_attr' claim encodes the principal attributes within the authentication token. We explain the 'xms_acb' claim in Section 6.1.

```
{
   "sub": "<internal-prefix>/s/10ef5b45-a7e5-4f96-9d11-90e8b5e06a87/rg/test-eus-rg/sf/test-eus-
       cluster/7af6ddcc-8407-427d-ac61-5a47a0ea8e00/SqlApplicationType/SqlApplicationName",
   "cnf": {
      "x5t#S256": "<hash_of_the_credential>"
   },
   "xms_tbflags":"1",
   "xms_attr": {
      "<encoded_attribute_authority_identifier>":
      {
         "Microsoft.AttributeAttestation/AttributeNamespaces/SqlEus/Attributes/readAccessGroups": "
             b3f2c9d4-8a7e-4f1a-9d3b-7e6c2a1f5e8d",
      }
   },
```

```
    "xms_acb": "<ACB_value_to_bind_to_auth_token>",
    ...
}
```

6 Authorization

We leverage Capability tokens that encapsulate pre-evaluated access rights in a signed, portable format, issued by authorization systems and authenticated by a trusted authority (e.g., Microsoft Entra). The token is carried by the accessing identity and presented to resource providers as part of access requests. Resource providers validate the token and authorize requests according to the client's request and the access rights delineated within the token. This approach eliminates the need for runtime policy fetching and evaluation, thereby facilitating strong caching affinity—a feature that remains challenging in distributed systems. This model delegates policy evaluation to the token issuer, allowing the resource to validate and enforce permissions without runtime policy loading and checks.

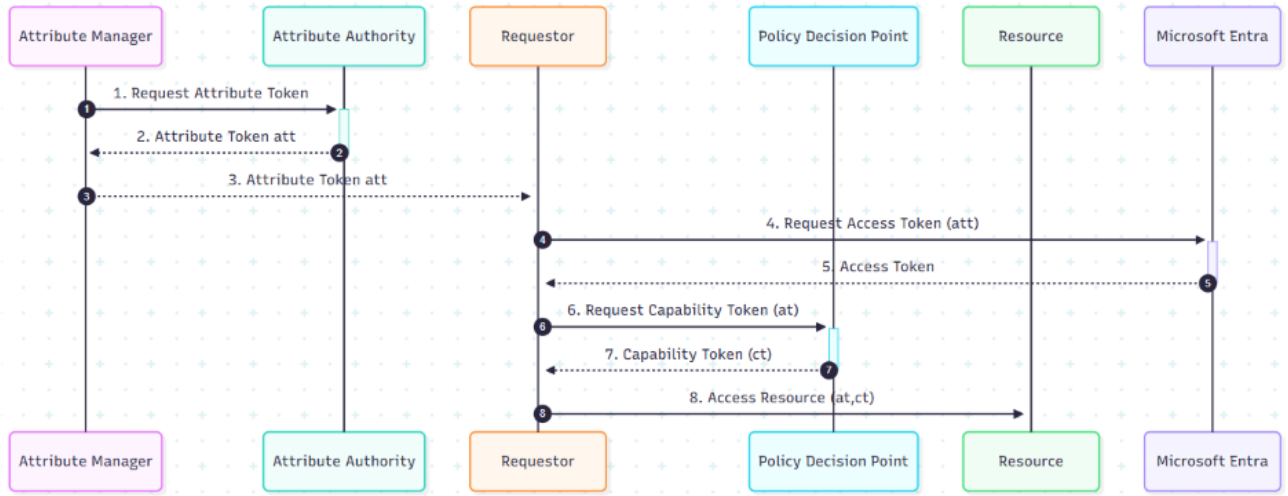

Figure 4: Combined Authorization Flow

Partial evaluation is a concept discussed in [11]. Resource attributes, along with certain environment attributes, may not be accessible at the time of capability token issuance. Consequently, the authorization service may be unable to fully evaluate all ABAC policies during issuance if such policies rely on unavailable attributes. To address these cases, the authorization service provides support for partial evaluation of ABAC policies where feasible, while retaining any remaining policies for deferred evaluation upon subsequent token requests. These partially evaluated policies are stored within the capability token. The resource owner bears responsibility for furnishing the relevant attributes and conducting the secondary evaluation of the deferred policies from the token, thereby ensuring that all necessary attributes are present and verified prior to granting access.

In the proposed ABAC model, a single role assignment, for each permission, added at a pre-defined scope encodes the policy such that only the identities possessing principal attributes from a specific namespace are permitted access to resources tagged with matching resource attributes. In the example below we illustrate the role assignment for the scenario described in Section 3, the 'Blob Container Read' role, identified by 'ba92f5b42d11453da403e96b0029c9fe' is granted to all principals i.e. represented by the universal identifier '00000000-0000-0000-0000-000000000000'. This assignment is scoped within the subscription 'f984cbdd-9e7e-4b97-9744-5c5d9295e332', along with an ABAC condition. The ABAC

condition ensures that HSMIs possessing a Principal attribute value for the key 'readAccessGroups', under the 'SqlEus' namespace, owned by SQL Control Plane, are authorized to read Storage containers provided that the Resource Attribute i.e. container metadata value for the same key matches.

```
{
   "Id": "b69889ff-3281-4ffa-8f75-6d7bd3be6616",
   "RoleDefinitionId": "ba92f5b42d11453da403e96b0029c9fe",
   "PrincipalId": "00000000-0000-0000-0000-000000000000",
   "Scope": "/subscriptions/f984cbdd-9e7e-4b97-9744-5c5d9295e332",
   "Condition": "@Principal[Microsoft.AttributeAttestation/AttributeNamespaces/SqlEus/attributes/
       readAccessGroups] ForAnyOfAnyValues:StringEqualsIgnoreCase SplitString{@Resource[Microsoft.
       Storage/storageAccounts/blobServices/containers/metadata:readAccessGroups]}"
}
```

Using the authentication token with federated attributes, the SQL instance requests a capability token from the authorization service for specific target storage resources. The authorization service performs authorization evaluation using PDP-R, considering both the principal attributes and target resource permissions. Upon successful evaluation, the authorization service returns a signed capability token containing pre-evaluated access decisions for the requested resource-action pairs. The claims in the capability token include—the identity performing the action (e.g. HSMI identifier), the targeted resource scope (e.g. Storage Account Container), and the permissions specifying which actions such e.g. Read or Write are authorized. An illustrative example of a capability token for the scenario described in Section 3 is provided below, where 'sub' claim identifies the HSMI, and 'xms_authz' claim encodes precomputed decisions for the 'Blob Container Read' permissions in the subscription scope 'a7c23d70-ffbd-4e6e-eee2-fc18f9518db2'. The claim 'ac' refers to 'access check' and the claim 'c' refers to 'condition'.

```
{
   "sub": "<internal-prefix>/s/10ef5b45-a7e5-4f96-9d11-90e8b5e06a87/rg/test-eus-rg/sf/test-eus-
       cluster/7af6ddcc-8407-427d-ac61-5a47a0ea8e00/SqlApplicationType/SqlApplicationName",
   "xms_acb": "<ACB_value_to_bind_to_auth_token>",
   "xms_authz": {
      "ac":{
         "/subscriptions/a7c23d70-ffbd-4e6e-eee2-fc18f9518db2":{
            "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read":{
               "ac":[{
                  "c":"@Principal[Microsoft.AttributeAttestation/AttributeNamespaces/SqlEus/
                      attributes/readAccessGroups] ForAnyOfAnyValues:StringEqualsIgnoreCase
                      SplitString{@Resource[Microsoft.Storage/storageAccounts/blobServices/containers
                      /metadata:readAccessGroups]}",
                  }]
   ...
}
```

The SQL Instance presents both the authentication token and capability token when accessing Azure Storage resources. The storage service validates both tokens and performs authorization based on the pre-evaluated permissions in the capability token. All tokens are cryptographically bound. The xms_acb claim ensures that all tokens are bound to the same authentication context. The capability token includes the xms_acb claim from the original authentication token, ensuring that leaked capability tokens cannot be used by unauthorized identities. This claim is further described in Section 6.1

6.1 Authorization Token Binding for Context Integrity

In distributed identity and access management systems, authorization token binding is essential for maintaining the integrity and contextual consistency of security tokens. This mechanism mitigates token mix-up attacks, where tokens intended for one context are erroneously or maliciously reused elsewhere. Token binding associates each token with a unique representation of its authorization context, ensuring validity strictly within the intended scope. Authorization and attribute tokens encapsulate claims about

a principal and its context, including client identity, IP address, cryptographic keys, identity type, roles, and external attributes. Without effective binding, tokens are vulnerable to substitution, resulting in potential unauthorized access. The Authorization Context Binding (ACB) claim i.e. 'xms_acb' addresses this by uniquely identifying the token's context. The ACB is computed as a cryptographic digest (e.g., SHA-256) over a normalized, serialized dictionary of context claims, ensuring consistency despite application-specific configurations. The ACB is embedded in authentication tokens and propagated to attribute and authorization tokens. Resources validate incoming requests by confirming matched ACB values across authentication and authorization tokens; mismatched or absent ACB values result in access denial. This process enforces context-specific token usage and supports independent refresh operations for authentication and authorization tokens. This mechanism enhances security by maintaining contextual integrity, enabling flexible token lifecycles, and providing strong guarantees against token misuse. The ACB claim is designed to be unique per principal and context, stable across token expirations, consistent for different audiences, and applicable across authentication, attribute, and authorization tokens, thereby facilitating secure token chaining.

7 Performance

Large data services such as Azure SQL are highly sensitive to I/O performance, especially during database recovery, failover, and cold start operations, where increased I/O latency can impact system reliability. The design described above involves calling multiple services. A straightforward implementation that calls all components every single time would significantly increase the latency of the service-to-service call. However, while multiple components are logically involved, the hot path does not call most of them. In the SQL-Storage example used above, the calls to the various components result in authentication and authorization tokens that are available to the SQL resource. Once the tokens are fetched, the data path involves SQL presenting the tokens to Storage, which redeems them efficiently. Current performance targets set the 99th percentile (P99) hot I/O path latency at 1 millisecond, which is comparable to the SAS and MSI methods described in Section 1; in particular, the improved security and scalability of HSMI does not carry a performance price. Experimental results for the described scenario, without token binding, show a P99 latency that is close to our target. Ongoing architectural improvements aim to reach the sub-millisecond P99 target.

The tokens, once fetched, have a Time-To-Live (TTL) in the order of hours. In order to avoid latencies when tokens expire, they are refreshed in the background at half their TTL, ensuring valid tokens are always available and minimizing latency spikes during critical operations. Capability tokens also prefetch most authorization decisions, enabling Azure Storage to process remaining authorization logic with minimal delay.

8 Related Work

Our work builds upon a rich body of research and industry practices in capability-based authorization and identity management. Several prior efforts have explored these concepts in depth. For instance, Li et al. extend OAuth to support enforcement of permission sequences and contextual constraints in distributed systems [3]. Alphabet's Fuchsia operating system adopts a capability-based model to govern inter-component permissions [4], and as of August 2021, it has been deployed across all Nest Hub devices [5]. Amazon Web Services offers a related mechanism through its session token model [6], though it is tailored to specific services like S3. In contrast, our approach generalizes the concept to support broader, cross-service scenarios. Similarly, the shift toward managed identities as a replacement for secrets and service accounts is gaining momentum across all major cloud providers [8], [7]. This

trend includes the adoption of hierarchical identity frameworks such as SPIFFE [10], which simplify permission management and delegation workflows [9]. What sets our work apart is the integration of these ideas into a unified model that treats attestation as a first-class primitive. By addressing the scalability and operational challenges of identity and authorization in cloud-native environments, our solution delivers a robust, extensible foundation for secure access control in modern distributed systems.

9 Conclusion

HSMIs and capability-based authorization tokens provide a scalable, secure, and flexible foundation for access control in cloud-native environments. By decoupling identity from directories and leveraging attestation and token binding, HSMIs address the limitations of traditional models and enable new scenarios in distributed systems.The capability-based authorization model, augmented with attribute tokens and namespacing, offers a scalable, flexible, and performant alternative to RBAC for modern distributed systems. By supporting decentralized, low-latency access control, this model is particularly well-suited for S2S scenarios. As enterprises adopt dynamic, federated architectures, this approach provides a strong foundation for secure and efficient authorization.

References

[1] I. Alagenchev et al., "Managed Identity Client Credential Release API Spec," Microsoft Internal Document, 2025.

[2] I. Alagenchev et al., "End to End S2S Token Binding Design," Microsoft Internal Document, 2025.

[3] A. S. Li, R. Safavi-Naini, and P. W. L. Fong, "A Capability-based Distributed Authorization System to Enforce Context-aware Permission Sequences," in *Proc. of the 27th ACM Symposium on Access Control Models and Technologies (SACMAT '22)*, New York, NY, USA: ACM, 2022, pp. 13–24. DOI: 10.1145/3532105.3535014.

[4] Fuchsia Project, "Capabilities - Fuchsia Component Framework," *Fuchsia.dev*, [Online]. Available: `https://fuchsia.dev/fuchsia-src/concepts/components/v2/capabilities`. [Accessed: Sep. 2, 2025].

[5] Wikipedia contributors, "Fuchsia (operating system)," *Wikipedia, The Free Encyclopedia*, [Online]. Available: `https://en.wikipedia.org/wiki/Fuchsia_(operating_system)`. [Accessed: Sep. 2, 2025].

[6] Amazon Web Services, "GetSessionToken - AWS Security Token Service," *AWS Documentation*, [Online]. Available: `https://docs.aws.amazon.com/STS/latest/APIReference/API_GetSessionToken.html`. [Accessed: Sep. 2, 2025].

[7] Google Cloud, "Configure managed workload identity authentication," *Google Cloud IAM Documentation*, [Online]. Available: `https://cloud.google.com/iam/docs/create-managed-workload-identities?authuser=1`. [Accessed: Sep. 2, 2025].

[8] Amazon Web Services, "IAM Roles - AWS Identity and Access Management," *AWS Documentation*, [Online]. Available: `https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html`. [Accessed: Sep. 2, 2025].

[9] SPIFFE Project, "SPIRE," *spiffe.io*, [Online]. Available: `https://spiffe.io/spire/`. [Accessed: Sep. 2, 2025].

[10] SPIFFE Project, "SPIFFE – Secure Production Identity Framework for Everyone," *spiffe.io*, [Online]. Available: `https://spiffe.io`. [Accessed: Sep. 2, 2025].

[11] S. Ramaswamy, T. Slepnev, and P. Allen, "Guide to Secure Web Services," *NIST Special Publication*

800-95, Aug. 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf.

[12] Michael B. Jones and John Bradley and Nat Sakimura, "JSON Web Token (JWT)," *Request for Comments 7519*, May. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7519.

[13] International Telecommunication Union, "Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks (X.509)," *Recommendation X.509*, Oct. 2019. [Online]. Available: https://www.itu.int/rec/T-REC-X.509/en.

[14] Microsoft, "What is managed identities for Azure resources?," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/entra/identity/managed-identities-azure-resources/overview.

[15] Microsoft, "What is Azure attribute-based access control (Azure ABAC)?," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/azure/role-based-access-control/conditions-overview.

[16] Microsoft, "Trusted Execution Environment (TEE)," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/trusted-execution-environment.

[17] Microsoft, "Grant limited access to Azure Storage resources using shared access signatures (SAS)," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/common/storage-sas-overview.

[18] Microsoft, "What is Microsoft Entra?," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/entra/fundamentals/what-is-entra.

[19] Microsoft, "What is Azure SQL?," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-overview?view=azuresql.

[20] Microsoft, "Introduction to Azure Storage," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction.

[21] Microsoft, "Virtual machines in Azure," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/overview.

[22] Microsoft, "Service Fabric terminology overview," *Microsoft Documentation*, [Online]. Available: https://learn.microsoft.com/en-us/azure/service-fabric/service-fabric-technical-overview.