## 1. Solution Architecture Scope, Structure & Architectural Views

A solution architecture is a **holistic engineering blueprint** that aligns business objectives with technical execution. A complete & well-defined solution architecture typically covers the following scopes & structures (usually included in the solution architecture documents):

- **Business objectives & success criteria**
  Clear articulation of the problem being solved, expected outcomes, and measurable KPIs.
- **Constraints & assumptions**
  Including regulatory, security, budget, latency, data residency, operational, and organizational constraints.
- **Major solution components & responsibilities**
  Logical decomposition into subsystems (e.g., presentation, service, data, integration, analytics), with clear ownership and boundaries.
- **Communication interfaces**
  - Interaction patterns (synchronous vs asynchronous)
  - Data formats (JSON, Avro, Protobuf, CSV, etc.)
  - Protocols (HTTP/REST, gRPC, WebSocket, AMQP, Kafka)
  - Event contracts and API specifications
- **Security architecture**
  Identity and access management, network segmentation, secrets management, encryption (in transit and at rest), zero-trust principles, and least-privilege access.
- **Observability & operability**
  Centralized logging, metrics, tracing, alerting, health checks, and operational dashboards to support SRE and DevOps practices.
- **Auditing, governance & compliance**
  Policy enforcement, audit trails, data lineage, regulatory controls, and security posture management.
- **Resource management & scalability**
  Elastic scaling strategies, capacity planning, throttling, quotas, and performance optimization.
- **Infrastructure & deployment design**
  Cloud-native, hybrid, or on-prem patterns; high availability, fault tolerance, disaster recovery, and environment isolation (dev/test/prod).
- **Cost modeling & optimization**
  Cost drivers, trade-offs, consumption-based pricing, and FinOps considerations embedded early in the design.

This structured approach ensures that architectures are **resilient, secure, scalable, operable, and cost-aware**, while remaining adaptable to evolving business needs.

**Architectural Views:**

A comprehensive solution architecture is not limited to a single diagram or technology choice. It is a **multi-view design exercise** intended to address business objectives, technical constraints, and operational requirements in a structured and traceable manner. During the architecture design process, multiple **architectural complementary views** are produced to address distinct stakeholder concerns & project scope to ensure traceability from business needs to technical implementation, and support informed decision-making throughout the lifecycle of the solution.

- **Executive View** – Business-aligned, decision-oriented overview highlighting trade-offs, risks, cost drivers, and strategic choices for leadership and decision-makers. Focuses on architectural trade-offs, cost drivers, scalability assumptions, and risk areas to support leadership decisions. Provides a business-aligned, high-level overview tailored for decision-makers. It highlights major architectural choices, cost drivers, scalability assumptions, risks, and strategic trade-offs. This view supports validation of the architecture direction and investment decisions. This view is illustrated through architecture choices such as serverless vs. container-based approaches, multi-zone deployments, and cost-optimized designs adapted to expected usage and geographic scope.

  **Audience**: Executives, sponsors, business owners
  **Purpose**: Why are we doing this? What value does it deliver?

- **Conceptual View (Business Specifications)** – High-level representation of business capabilities, domains, and major building blocks, used to align with business requirements and objectives. Provides a high-level representation of business capabilities, domains, and major building blocks. Defines what the business needs to achieve. It focuses on business capabilities, high-level processes, actors, and key outcomes without considering technology or implementation constraints. This view establishes the *functional* scope and shared understanding between business and IT stakeholders.

  **Audience**: Business architects, product owners
  **Purpose**: What systems and data domains exist to deliver that value?

- **Logical View** – Functional decomposition of the system, showing major components, service boundaries, integrations, interfaces, data flows, and protocols. Describes how the system works, including service boundaries, integrations, APIs, data flows, messaging, and processing pipelines. We may need to produce several different logical views to outline different level of details & data flows.

  **Audience**: Solution & data architects
  **Purpose**: How does data flow and get transformed to meet the business goals?

- **Physical View** – Deployment and infrastructure perspective, detailing cloud & on-prem resources, network topologies (subnets, LANs, routers), involved physical machines, servers, virtual machines, runtime environments, data centers, scaling strategies, security controls and resilience patterns. Details on where and on what the system runs. Within the provided Cloud Infrastructure Design & Networking Principals document, the included cloud infrastructure design diagram shows the VPC/VNet, public and private subnets, load balancers, NAT gateways, security layers & represent the cloud physical view which serves as a reusable deployment baseline (the attached diagram is missing firewalls at IP subnet level for each subnet). Usually within this view, we need to see what is our deployment strategy, as what we deploy, where we deploy it, & how it gets connected to its physical environment.

  **Audience:** Engineers, platform teams

**Purpose:** Where and with what services is this implemented?

- **ArchiMate Views** – Standardized viewpoints describing relationships across **business**, **application**, and **technology** layers, enabling traceability and governance in complex enterprises. Used when required to formalize relationships across business, application, and technology layers, ensuring traceability, governance, and alignment in complex enterprise environments. Its latest version contains also the motivation (*defines the why*) & strategy (*defines what & how*) views.

In the above architecture views, I did not include the Business Architecture view which outlines the business processes. I did outline ArchiMate framework which contain the business process architecture view as part of the set of architecture produced views.

**Important Distinction Between Architecture Views**

This layered approach ensures architectural clarity, stakeholder alignment, and informed decision-making throughout the solution lifecycle.

- **Executive View** → What decision-makers need to decide
- **Conceptual View** → What the business needs
- **Logical View** → How the system works
- **Physical View** → Where and on what the system runs

**Diagram Mapping**

- **Executive View** diagrams summarize system boundaries and strategic components
- **Conceptual View** diagrams illustrate business capabilities
- **Logical View** diagrams describe data flows, integrations, and processing layers
- **Physical View** diagrams detail cloud infrastructure, networking, & deployment topology

Development approach to solution architecture should be structured, methodical, and driven by both business objectives and technical constraints. A solution architecture is not a single diagram, but a set of complementary views that together describing **what must be built, why it must be built, how it will work, and where it will run**.

**Well Architected Pillars:**

- **Security Principals**
  - **Authentication & Authorization:** Application of IAM principals with the definition of set of permissions & policies applied to users, applications & resources (use of credentials, API Keys, access tokens with technologies such as OAuth 2.0 & OIDC).
  - **Application Security:** Securing the application layer with different type of firewalls (WAF, application & network level firewalls etc.). Access to an application should be secured at network ip level (source & destination), port numbers (allowed) and used protocols.
  - **Data Protection:** Encryption mechanism & key management influences our design to protect the data in Rest & Transit (being encrypted). Keeping the security key secure and having rotation mechanism to change it could also affect the level of data protection. Data Replication (RPO & RTO) plus Disaster Recovery strategies also affect our data security

- **Resilience & Reliability**
  - **Designing multi-layer apps & solution**: having a multi-layer solution allow for better management of resilience since there would not be a single point of failure & increase our chance to have better scalability options
  - **HA & Fault-Tolerant solution:** Deploying our solution within different physical location to ensure HA & fault-tolerant solution with different type of failover mechanism (active/active & active/passive)
  - **Distributed decoupled solution:** It impacts the management and evolution of a solution as well as its capacity to scale and being secure.
- **Performance**
  - **Computing solution:** Designing highly elastic & scalable computing solution. Elasticity of compute solution are offered within cloud environments. It provides the core ability to automatically and quickly expand or decrease computing resources (CPU, memory, storage) to meet changing demand without manual intervention or over-provisioning.
    - **Elasticity** is typically easier to achieve with **cloud architectures**, where computing resources can be provisioned on demand or billed on a pay-as-you-go basis, depending on the cloud service used. Both serverless and server-based solutions commonly provide built-in autoscaling capabilities that dynamically adjust resources based on workload demand.
    - In an **on-premises** environment, elasticity can also be achieved by deploying applications as containers and using Kubernetes as the container orchestration platform. **Kubernetes enables dynamic horizontal and vertical scaling of compute resources to match application needs**. However, while **Kubernetes** supports **elastic computing** in on-premises environments, achieving this level of elasticity generally requires additional setup and integration compared to public cloud platforms, where autoscaling is often provided as a managed service. In on-premises scenarios, elasticity depends on integrating Kubernetes' native autoscaling mechanisms with the capabilities of the underlying infrastructure.
  - **Storage solution:** Designing highly elastic (cloud), resilient & scalable storage solution. A resilient storage solution ensures continuous, reliable data access by protecting against failures (hardware, software, cyberattacks) through redundancy, replication, and rapid recovery, using features like immutable snapshots, geo0replication & automated failover to achieve high availability and minimize downtime (RTO/RPO). It goes beyond basic backup, focusing on fault-tolerance & data integrity in diverse environments (on-prem, cloud, hybrid)
  - **Network solution:** Designing a scalable, resilient & redundant IP network design solution
  - **Database solution:** Designing an elastic (cloud), scalable, reliable (having read & write replicas) solution. (In cloud environment database services can automatically scales its resources such as compute, storage & memory up or down in real-time to match fluctuating demands).
- **Cost Optimized Solution**

    - Designing a cost optimized **compute solution**
    - Designing a cost optimized **storage solution**
    - Designing a cost optimized **network solution**
    - Designing a cost optimized **database solution**

- **Operational Excellency**
    - **CI/CD pipeline & DevOps practices**
    - **Monitoring & auditing:**
        - Monitoring**:** Effective monitoring provides the data and insights necessary for data-driven decisions and continuous improvement. It allows to set automatic alerts for preventing miss behavior of production environment (which requires a well-defined monitoring strategy).
        - Auditing acting as a crucial tool to identify inefficiencies, reduce waste, improve processes, ensure compliance, strengthen controls & drive data driven decisions, ultimately leading to higher performance, cost saving, and better customer satisfaction. Audits provide objective insights, turning potential risks into opportunities for continuous improvement, making them fundamental for achieving OpEx goals.
    - **Governance & compliance:**
        - Governance fundamentally affects Operational Excellence (OpEx) by providing the essential structure, rules, and oversight for processes, ensuring they align with strategy & remain efficient. Strong governance defines clear roles, standards & permissions which affect how our resources are operating in a day-to-day production environment.
        - Compliance: Integrating compliance with OpEx creates a more resilient, high-performing organization that boosts throughput, reputation, and bottom-line results. It acts as a baseline for quality and safety, reducing risks, preventing costly penalties

## 2. Cloud Infrastructure Design & Networking Principals

Across AWS and Azure environments, I generally apply a **layered virtual private network (VPC/VNet) architecture** that balances security, scalability, and operational clarity. While implementation details vary by cloud provider, the underlying principles remain consistent. In case of Amazon, we need an Internet Gateway to enter our Virtual Private Cloud which contains our IP subnets.

**Network & Subnet Design**

A typical deployment includes a **Virtual Private Network** segmented into three logical subnet layers, usually duplicated across **at least two Availability Zones** for high availability:

1. **Public Subnet – Web / Edge Layer:** Internet facing IP subnet
    - Hosts internet-facing components such as:
        - Application Load Balancer (AWS ALB) or Azure Application Gateway
        - API Gateway integration points
    - Includes a **NAT Gateway** to allow controlled outbound access for private subnets
    - Entry point for external traffic
    - Protected by dedicated IP subnet **application (at instance level) & network level firewalls**
    - IP subnet layer used to communicate with the service layer private IP subnet
2. **Private Subnet – Service / Application Layer**
    - Contains compute and application workloads, not directly exposed to the internet
    - Examples:
        - AWS: EKS, ECS, Lambda (VPC-attached), EC2
        - Azure: AKS, Azure Container Apps, App Services (VNet-integrated), Virtual Machines
    - No public IP exposure
    - Communication occurs through internal load balancers and private endpoints
    - Scales horizontally based on demand
    - Protected by dedicated IP subnet **application & network level firewalls**
    - IP subnet layer used to communicate with the data layer private IP subnet

    - Outbound connections to internet must pass Nat Gateway within Public web layer subnet

3. **Private Subnet – Data Layer**
    - Hosts or provides private access to data services:
        - AWS: Aurora, RDS, DynamoDB via VPC endpoints
        - Azure: Azure SQL, Cosmos DB, Storage via Private Endpoints
    - No public IP exposure
    - Strict inbound access from approved service layers only
    - Protected by dedicated IP subnet **application & network level firewalls**
    - Communicates only with service layer private IP subnet.
    - Outbound connections to internet must pass Nat Gateway within Public web layer subnet

**Network Security Controls**

Security is enforced through **defense-in-depth**, implemented differently but equivalently across AWS and Azure:

- **Amazon**
    - **Security Groups**
        - Instance-level, stateful firewalls (first line of defense)
    - **Network ACLs**
        - Subnet-level, stateless firewalls (second line of defense)
    - **AWS Network Firewall** for advanced traffic inspection at VPC level (offering features like threat protection, intrusion detection and prevention systems (IDPS), and automatic scalability for cloud workloads)
- **Azure**
    - **Network Security Groups (NSG)**
        - Network-level traffic filtering
    - **Application Security Groups (ASG)**
        - Logical grouping of workloads by role to simplify NSG rules (e.g. logically grouping a group of web servers for which we want to apply the same NSG protection rules)
    - **Azure Firewall** for centralized policy enforcement

**Traffic Management & High Availability**

- **Application Load Balancers** distribute traffic across multiple Availability Zones
- Supports:

    - **Active/Active** deployments for fault tolerance and performance
    - **Active/Passive** patterns for controlled failover scenarios for fault-tolerance at data center level.
- **Network Load Balancers** are used when exposing services to other private networks or partner environments

**API & Integration Layer**

- **API Gateway** is placed in front of load balancers to:
    - Enforce authentication and authorization: The gateway centralizes security policies. It can authenticate and authorize incoming requests before they reach the backend services
    - Apply throttling, rate limiting, logging, monitoring, caching and request validation
    - Decouple external consumers from internal service topology
    - **Used when backend APIs need to support multiple communication protocols**, such as HTTP and WebSocket. The gateway typically accepts standard HTTP requests and routes them to HTTP-based microservices. It can also manage persistent WebSocket connections by acting as a protocol-aware proxy. The gateway handles the initial HTTP handshake required to upgrade a connection to WebSocket and maintains the connection lifecycle. Additionally, it can bridge to other protocols such as gRPC or message queues (e.g., AMQP, MQTT), translating frontend HTTP or WebSocket requests into the formats required by backend systems. Backend traffic is routed to internal services via ALB or internal load balancers

**Identity, Access & DNS**

- **Identity & Access Management**
    - AWS IAM or Azure Entra ID
    - Used for both human and workload identities
    - Supports OAuth2, OIDC, managed identities, and service principals
- **DNS**
    - Amazon Route 53 or Azure DNS
    - Routes both internal and external traffic to the appropriate endpoints
    - Integrates with load balancers and private endpoints

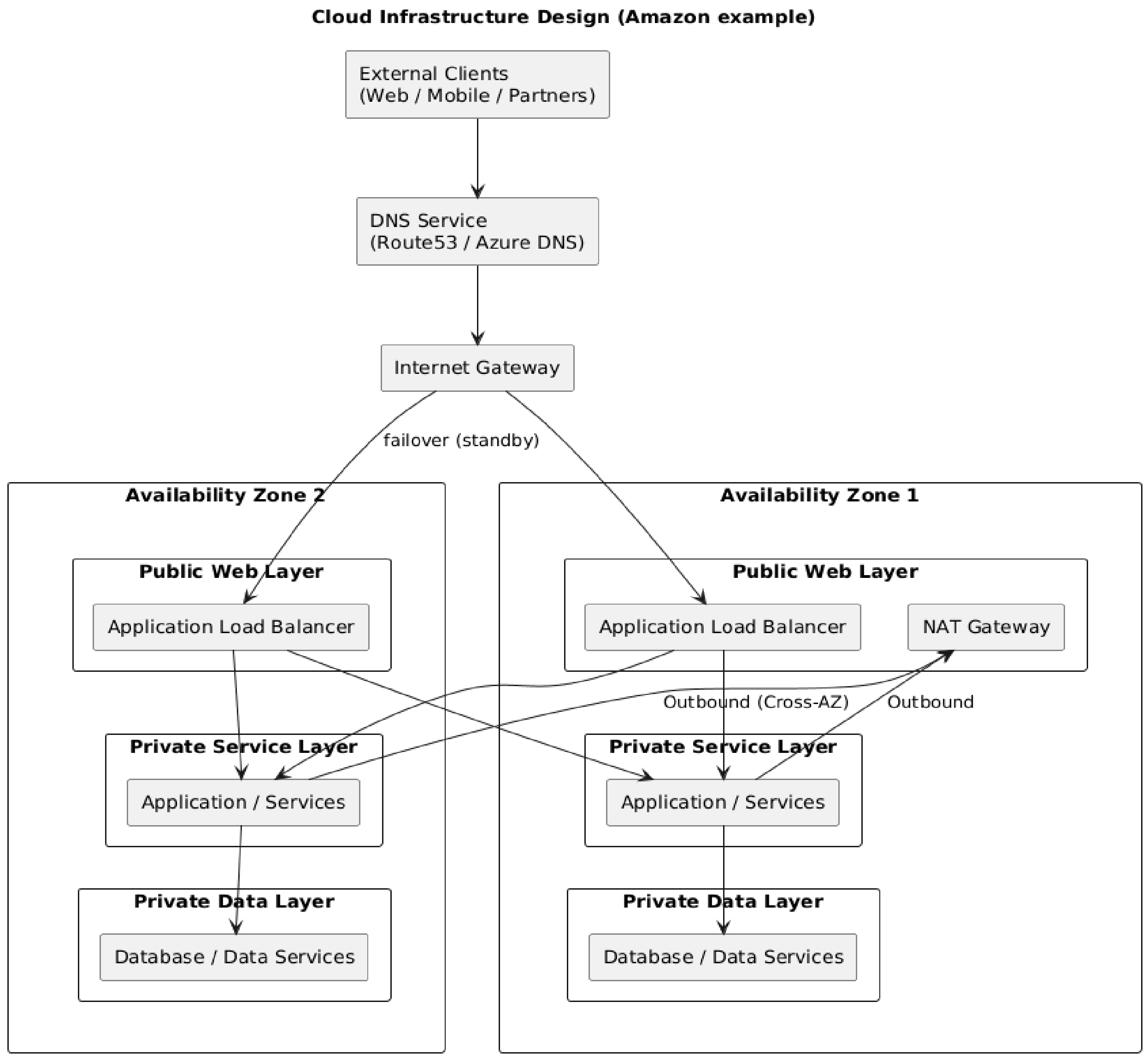
**Monitoring:** There are 2 ways of establishing monitoring activities within cloud. A monitoring activity could be processed at

- **Resource level**
    - **Monitors application and resource performance** (e.g., CPU, network) using metrics, logs, and alarms to ensure operational health (Amazon CloudWatch, Azure Monitor). Resource-level monitoring can also be extended to on-premises environments. This type of monitoring is commonly used to enable automatic resource scaling in cloud environments by defining alarm rules based on performance

metrics such as CPU utilization, network throughput, or requests per second. When thresholds are reached, alerts trigger scaling policies within the cloud provider's autoscaling services.

- **API level**
  - **Audits API calls** (who, what, when, where) for security/compliance, logging user actions across. It is an event-based logging service focused specifically on API calls & actions *within* (Amazon CloudTrail, Azure Activity Log)

**Private cloud infrastructure and network architecture:** Deployment across two Availability Zones to ensure high availability and fault tolerance at the data center level, using either active/active or active/passive topologies, within a dedicated Virtual Private Cloud (Amazon VPC or Azure VNet).

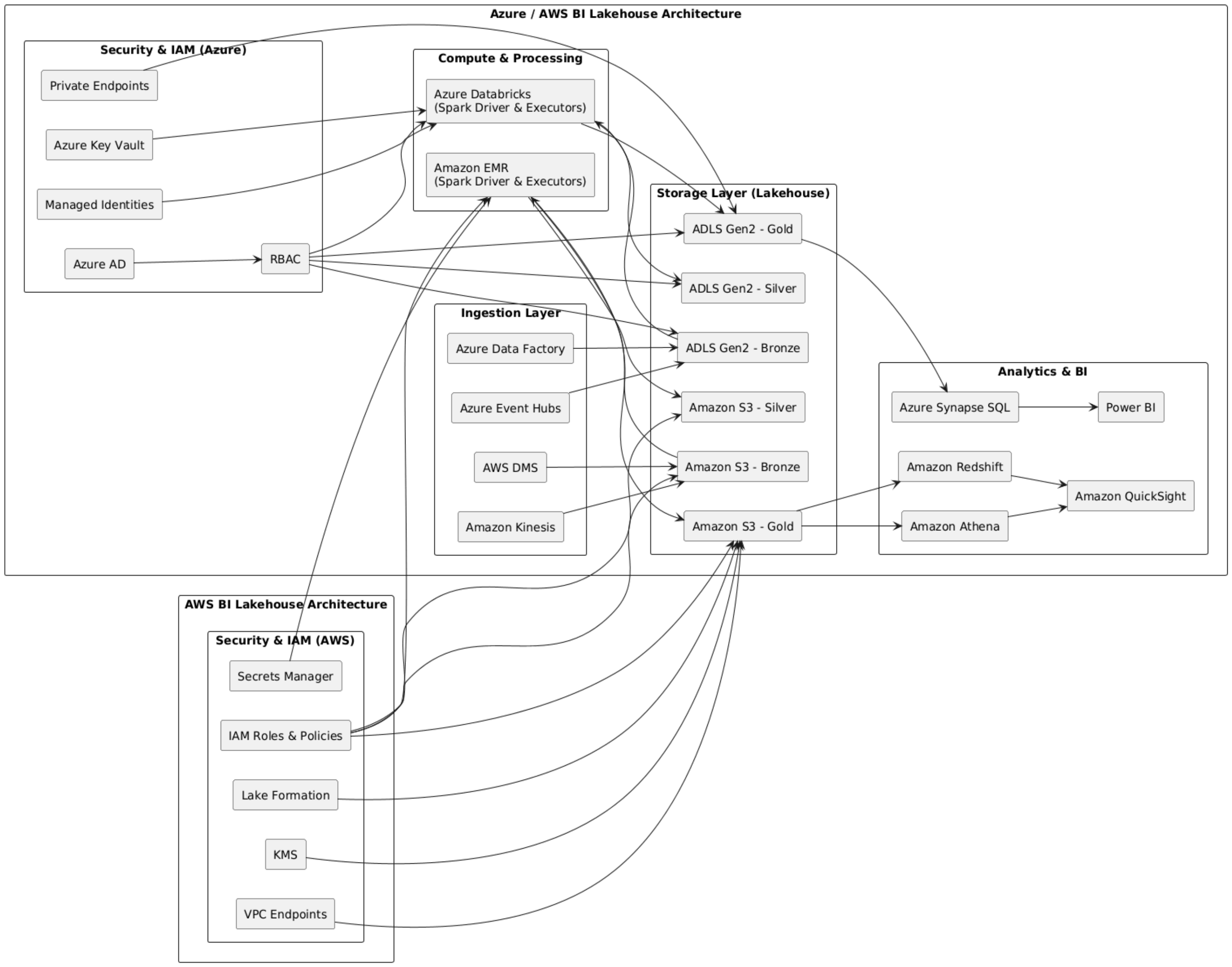


**Note:** Placing a second load balancer in a standby state in the second availability zone (2nd data center) impacts costs in the same way as implementing a highly available and fault-tolerant architecture. It is deployed to handle scenarios where the load balancer in the primary data center becomes unavailable, allowing traffic to be redirected to the secondary load balancer without experiencing downtime. At the enterprise level, high availability and fault-tolerant solutions are often critical non-functional requirements. However, in many cases, it is possible to operate without such stringent measures and instead adopt more cost-oriented solutions that still align with the required non-functional requirements for availability, reliability, performance, and latency. Ultimately, architectural solutions must remain feasible within the constraints of the supported costs and the targeted budget.

**The architecture pillars** defined in the Solution Architecture scope and structure document are particularly applicable to cloud-architected solutions, namely Security, Resilience, Performance, Cost Optimization, and Operational Excellence.

**Azure/AWS BI Lakehouse Architecture:** The architecture diagram below illustrates the architecture of a BI Lakehouse cloud solution using Azure or AWS infrastructure. The given diagram outlines the below architectural layers:

- **Security & IAM layer – Azure** infrastructure (Azure Entra ID formerly known as Azure Active Directory/AD)
- **Security & IAM layer – AWS BI Lakehouse Architecture**
- **Compute layer** - Azure & AWS infrastructure
    - The architecture diagram below illustrates the use of Apache Spark with AWS EMR (Elastic Map Reduce) or Azure Databricks. In many cases, or for more modern architecture, Spark could be used within AWS EKS or Azure AKS (Spark driver & executors' Kubernetes pods). The choice of Kubernetes platform for running Spark could differ depending on whether we are on Azure or AWS infrastructure (Please refer to the next page for notes regarding this choice).
- **Ingestion layer –** Azure & AWS infrastructure
- **Storage layer (Lakehouse) –** Azure & AWS infrastructure
- **Analytic layer –** Azure & AWS infrastructure



**Running Spark on AWS EMR (Elastic Map Reduce) or AWS EKS (Elastic Kubernetes Service):**

- Apache Spark on AWS EMR (EC2-based): Traditionally, Amazon EMR has provided a managed platform for running big data frameworks like Spark on EC2 instances.
- Apache Spark on AWS EKS (Kubernetes-based): A more modern approach involves running Spark applications on Amazon EKS, allowing users to consolidate workloads, improve resource utilization, and simplify infrastructure management by using Spark driver and executor pods within the Kubernetes cluster.

**Running Spark on Azure Databrick or AKS (Azure Kubernetes Service):**

**Azure Databricks** is a unified, Apache Spark-based analytics platform that provides a collaborative environment with significant performance optimizations and ease of use. It is generally better for enterprise-scale AI/ML and real-time analytics projects that prioritize speed of development and operational simplicity.

**Pros:**

- **Managed Service:** Databricks handles cluster setup, configuration, and scaling automatically, reducing operational overhead.
- **Performance Optimizations:** It uses an optimized Databricks Runtime that often improves query execution and performance by 10-100x over standard Spark.
- **Rich Feature Set:** It includes built-in collaborative notebooks, managed MLflow for machine learning, Delta Lake support for ACID transactions, and auto-scaling.
- **Ease of Use & Collaboration:** Provides a user-friendly environment well-suited for data engineers, data scientists, and analysts working together.

**Cons:**

- **Cost:** It is a paid subscription service and can be more expensive than open-source alternatives, especially for frequent or "always-on" workloads.
- **Less Control:** We have less granular control over the underlying infrastructure and Spark configurations compared to a self-managed environment.

**Azure Kubernetes Service** (AKS) allows us to deploy and manage containerized applications, including self-managed Apache Spark jobs. This approach gives our team complete control over the environment and infrastructure, making it ideal for highly customized pipelines and situations where cost efficiency is a primary driver.

**Pros:**

- **Cost Efficiency:** By using ephemeral (short-lived) pods and having full control over resource management, we can significantly reduce costs for specific workloads.

- **Full Control & Customization:** Ideal for technical teams with strong engineering skills who require granular control over the data pipelines, job scheduling, and performance tuning.

- **CI/CD Integration:** AKS is highly suitable for configuring continuous integration and continuous deployment (CI/CD) pipelines with automated, one-click deployments.

- **Portability:** Spark jobs packaged in containers on Kubernetes are more portable across different cloud or on-premises environments.

**Cons:**

- **Higher Complexity:** Requires significant manual effort for setup, configuration, maintenance, and optimization.

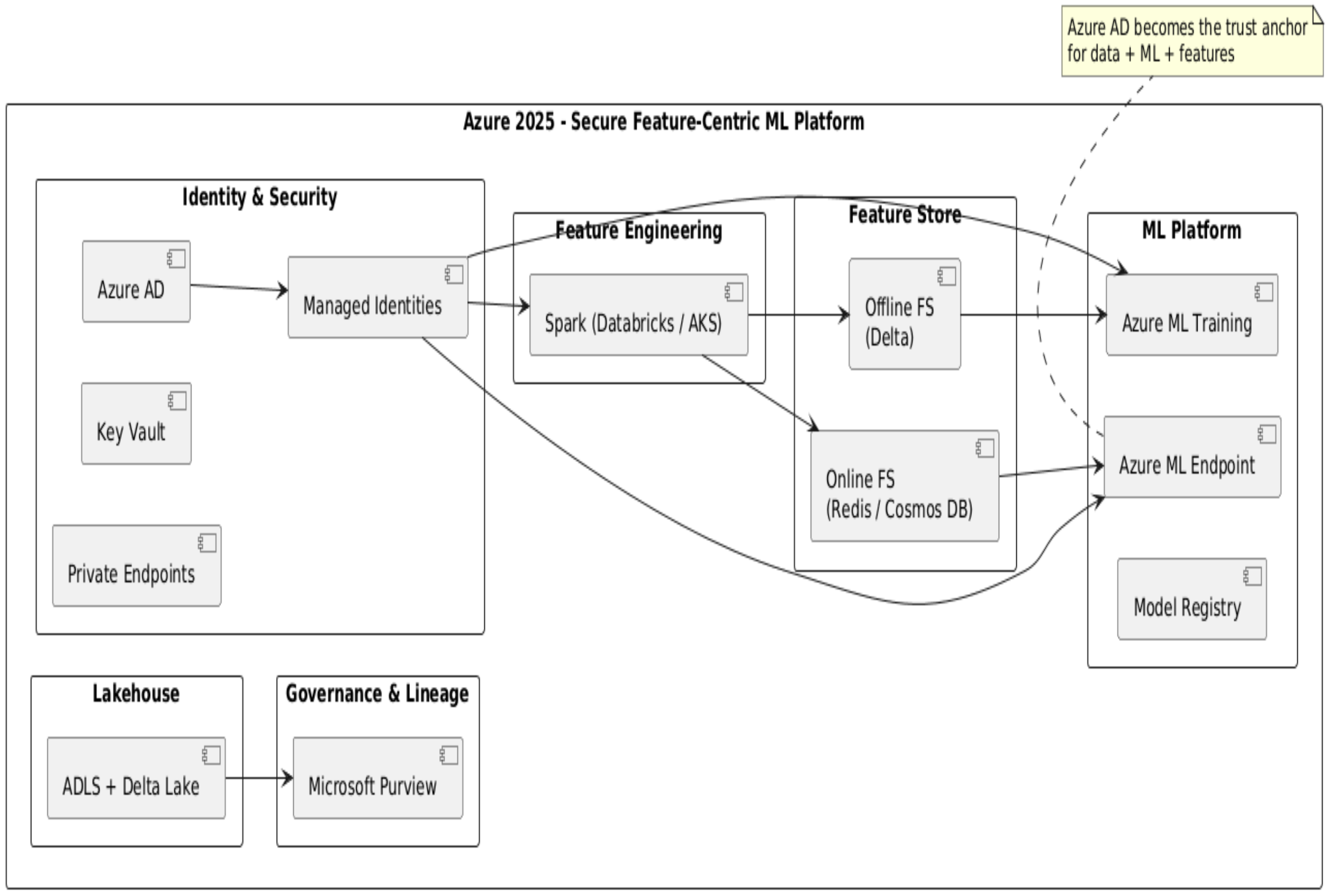
- **Operational Overhead:** Our team is responsible for managing the infrastructure, which demands specialized Kubernetes and Spark expertise.

- **Fewer Built-in Tools:** Lacks the integrated, out-of-the-box features like the collaborative notebooks and automated optimizations found in Databricks.

## Secure Feature-Centric Machine Learning Platform Architecture – Azure Infrastructure

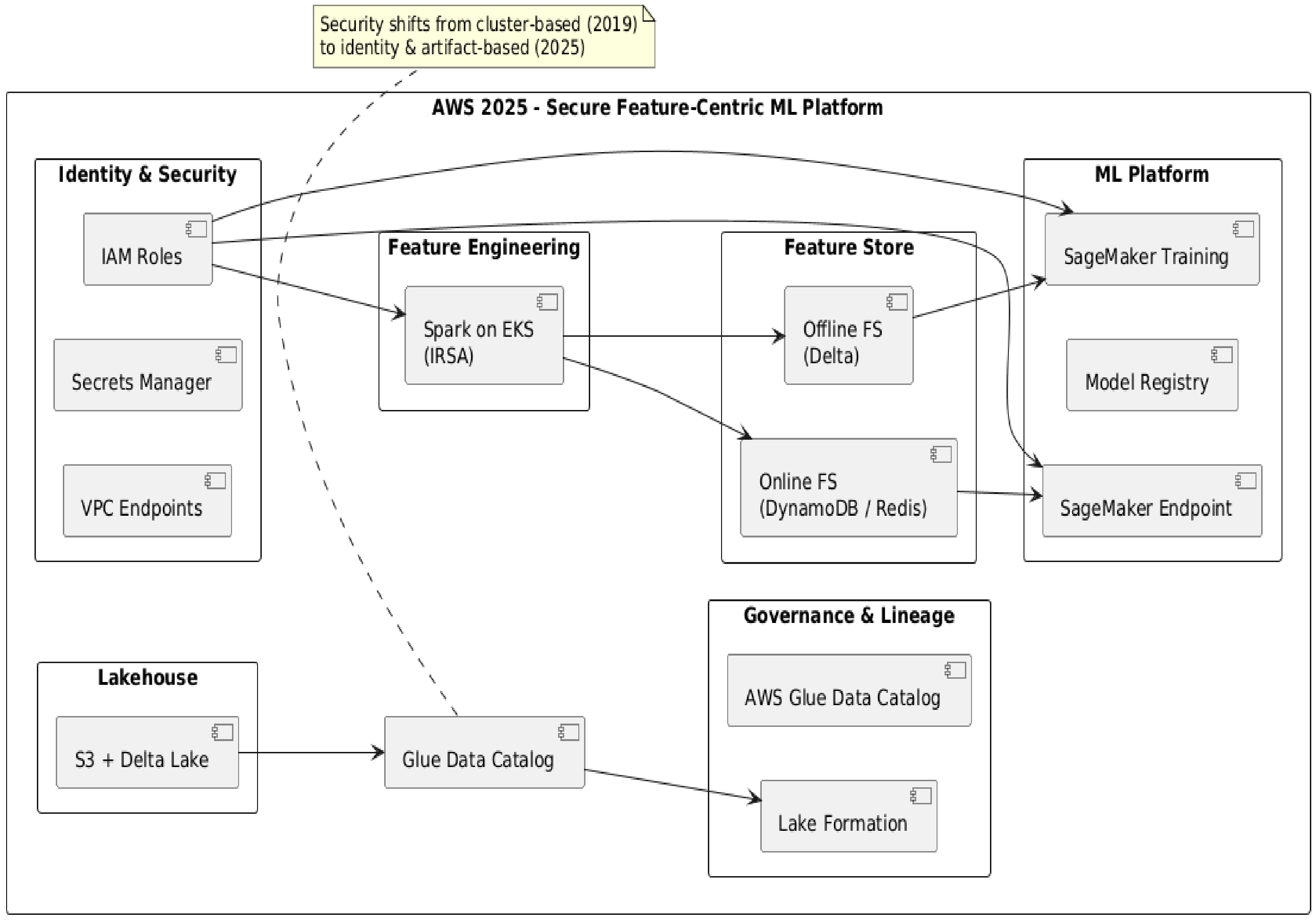
*Solution & Cloud Architecture Scope, Structure*

The architecture diagram below illustrates the design of a Secure Feature-Centric Machine Learning Platform using Azure infrastructure.



## Secure Feature-Centric Machine Learning Platform Architecture – AWS Infrastructure

The architecture diagram below illustrates the design of a Secure Feature-Centric Machine Learning Platform using AWS infrastructure.



The provided Azure and AWS Secure Feature-Centric Machine Learning Platform Architecture illustrate the following architectural layers.

- **Identity & Security**
- **Lakehouse**
- **Governance & linage**
- **Feature Engineering**
- **Feature Store**
- **Machine Learning Platform (model training, registry & endpoint)**

## Security & Governance for Feature-Centric Machine Learning Platform Architecture

Example: customer_risk_score (feature data) exists in *multiple physical forms*, but it's associated with only *one logical feature definition*.

**AWS Context**

| **Purpose** | **AWS service(s)** | **What is stored** | **Governance / Metadata** |
|---|---|---|---|
| **Offline feature (training, batch inference)** | Amazon S3 (Parquet / Delta Lake) | Historical feature values | **AWS Glue Data Catalog + Lake Formation** |
| **Feature engineering / processing** | AWS Glue ETL / EMR Serverless / Spark on EKS | Derived features, transformations | Glue lineage + CloudTrail |
| **Online feature (real-time inference)** | Amazon DynamoDB, ElastiCache (Redis) or Aurora | Latest feature value per entity | IAM / resource policies |
| **Feature registry (logical definition)** | Amazon SageMaker Feature Store | Feature schema, owners, versions | SageMaker + Glue metadata |
| **Consumption (training / inference)** | Amazon SageMaker / Lambda / EKS / ECS | Feature lookups | IAM role per workload |

On AWS, features such as *customer_risk_score* have a single logical definition managed through SageMaker Feature Store, while their offline representation is stored in S3 using Parquet or Delta Lake and governed by Glue Data Catalog and Lake Formation, and a synchronized online representation is served from DynamoDB or Redis for low-latency inference.

**Glue governs features, S3 stores them offline, DynamoDB (ElastiCache for Redis or Aurora) serves them online, SageMaker Feature Store defines them logically.**

If the feature needs joins or transactions → Aurora. If it needs speed → DynamoDB or Redis. Delta Lake tables stored on S3 are registered in AWS Glue Data Catalog to provide centralized metadata management, security enforcement via Lake Formation, and consistent access across analytics and ML services.

**Delta Lake tables stored on S3 *should* be referenced through AWS Glue Data Catalog** in a modern AWS data / ML architecture.

**Azure Context**

| <u>**Purpose**</u> | <u>**Azure service**</u> | <u>**What is stored**</u> | <u>**Governance / Metadata**</u> |
|---|---|---|---|
| **Offline feature (training, batch scoring)** | Azure Data Lake Storage Gen2 (Parquet / Delta) | Historical feature values | Microsoft Purview (catalog, lineage) |
| **Feature engineering / processing** | Azure Databricks or Synapse Spark | Derived features, transforms | Purview lineage |
| **Online feature (real-time inference)** | Azure Cosmos DB (Core API), Azure Cache for Redis or Azure SQL / PostgreSQL | Latest feature value per entity | RBAC / Managed Identity |
| **Feature registry (logical definition)** | Azure ML Feature Store *(or custom metadata)* | Feature schema, version, owners | Azure ML + Purview |
| **Consumption (training / scoring)** | Azure ML / Azure Functions / Container Apps | Feature lookups | Managed Identity |

On Azure, features such as *customer_risk_score* have a single logical definition managed in Azure ML Feature Store, while their offline representation is stored in ADLS Gen2 using Delta Lake and governed through Microsoft Purview, and a synchronized online representation is served from Cosmos DB or Redis for low-latency inference.

**Purview governs features, ADLS stores them offline, Cosmos DB (Azure Cache for Redis or Azure SQL/PostgreSQL) serves them online.**

Delta Lake tables stored on Azure Data Lake Storage Gen2 are governed through Microsoft Purview to provide centralized metadata management, data discovery, lineage, and access governance, with security enforced via Azure RBAC and data-plane ACLs, enabling consistent access across analytics and machine learning services such as Azure Databricks, Synapse Analytics, and Azure Machine Learning.

<u>**AWS vs Azure Security & Governance Symmetry for Feature Store & Data Lake solutions**</u>

**Feature Store**

| <u>**Need**</u> | <u>**AWS**</u> | <u>**Azure**</u> |
|---|---|---|
| **Key-value features** | DynamoDB | Cosmos DB |
| **In-memory features** | Redis | Azure Cache for Redis |
| **Relational features** | Aurora Serverless | Azure SQL / PostgreSQL |
| **Offline training** | S3 + Delta | ADLS + Delta |

**Data lake**

*Solution & Cloud Architecture Scope, Structure*

| **AWS** | **Azure** |
|---|---|
| Delta Lake on S3 | Delta Lake on ADLS Gen2 |
| Glue Data Catalog | Microsoft Purview |
| Lake Formation | Purview + RBAC + ACLs |
| IAM policies | Azure AD (Entra ID) |
| Athena / EMR / SageMaker | Databricks / Synapse / Azure ML |

| **Concept** | **AWS** | **Azure** |
|---|---|---|
| **Delta storage** | S3 | ADLS Gen2 |
| **Catalog** | Glue Data Catalog | Microsoft Purview |
| **Transaction log** | _delta_log | _delta_log |
| **Security** | Lake Formation | Purview + RBAC |

Features are often materialized in both offline analytical stores and online low-latency stores. The offline Delta Lake version supports training, governance, and historical analysis, while the online store serves real-time inference. Both reference the same logical feature definition.