IT457 – Cloud Computing

Course Information

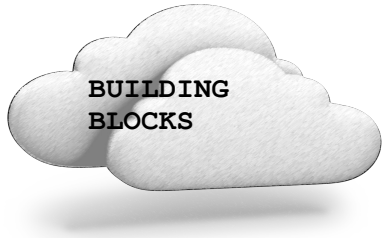
- Grading Scheme

  - Mid-sem exam – 40%
  - End-sem exam – 40%
  - Labs – 20%

  *subject to online exam system working well

Course Information

- Course Plan
  - Cloud Computing – Developer perspective
  - Cloud Computing – Architect perspective
  - Cloud Computing – IT perspective
- Cloud Service Providers
  - Azure
- References
  - Online material (like here, here and here).
  - No single reference book



Cloud Computing

- Definition

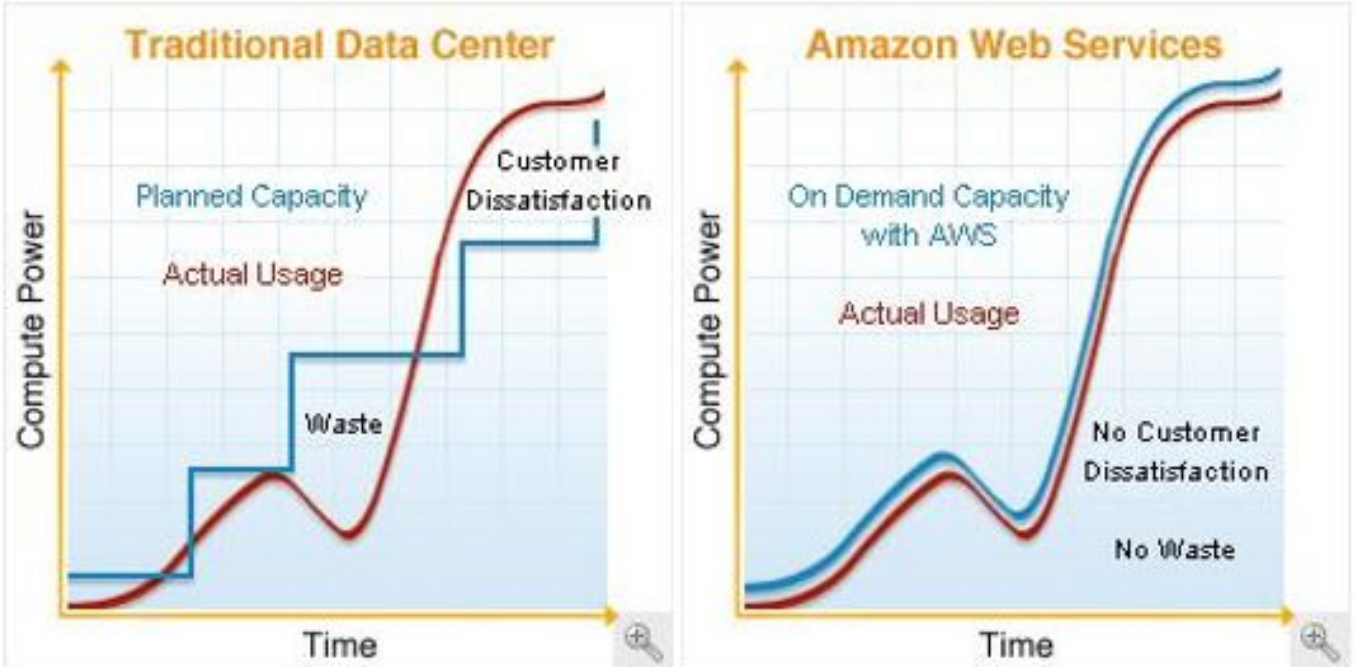
"Cloud computing is a specialized form of distributed computing that introduces utilization models for remotely provisioning scalable and measured resources" and platform services

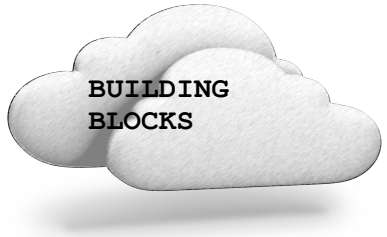
- Business Drivers (i.e. moti

Capacity Planning

Cost Reduction

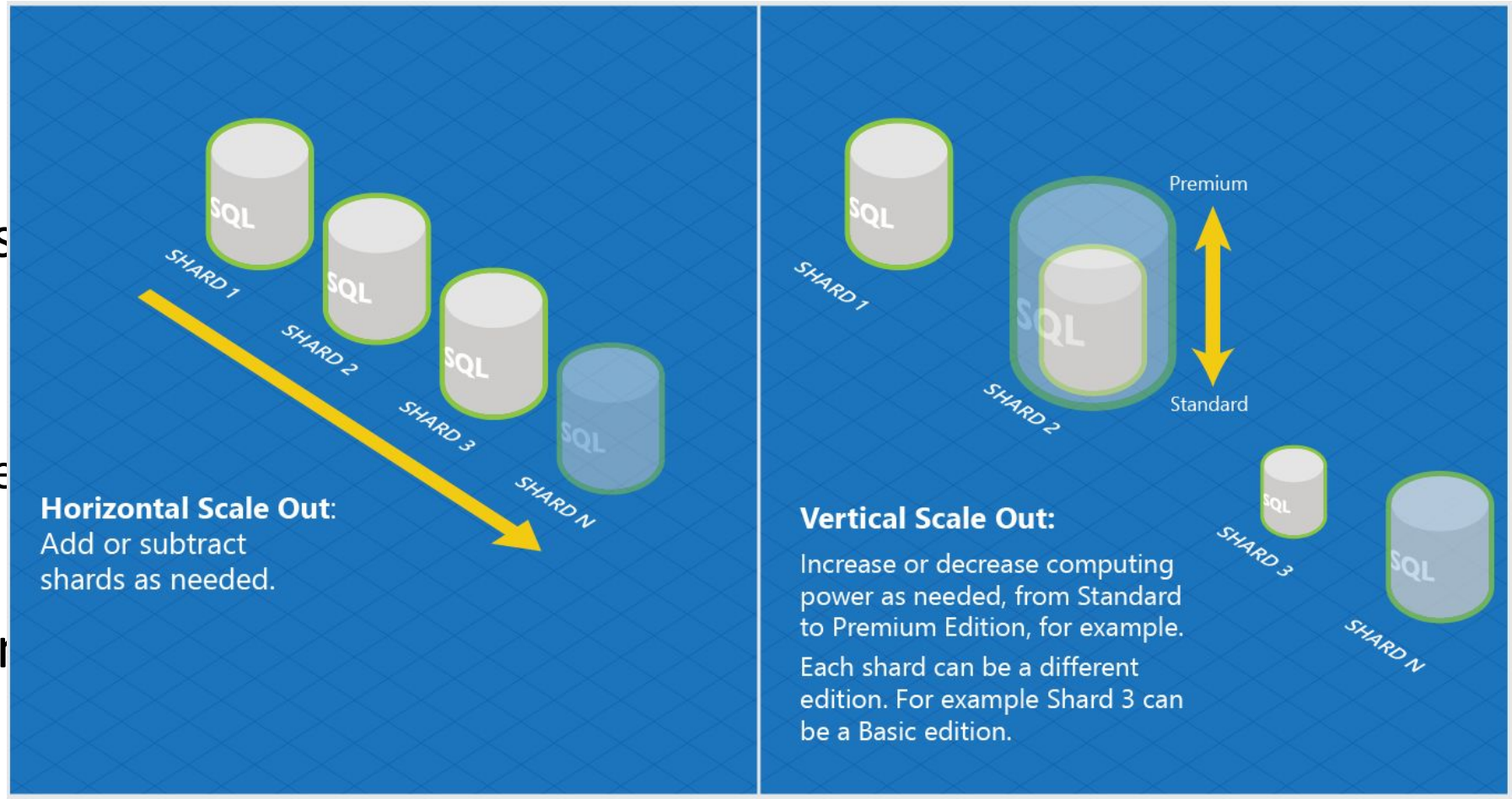
Organizational Agility

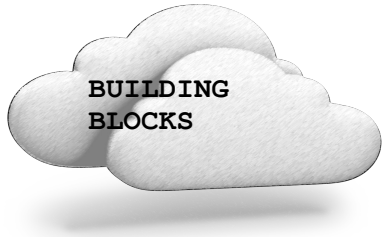




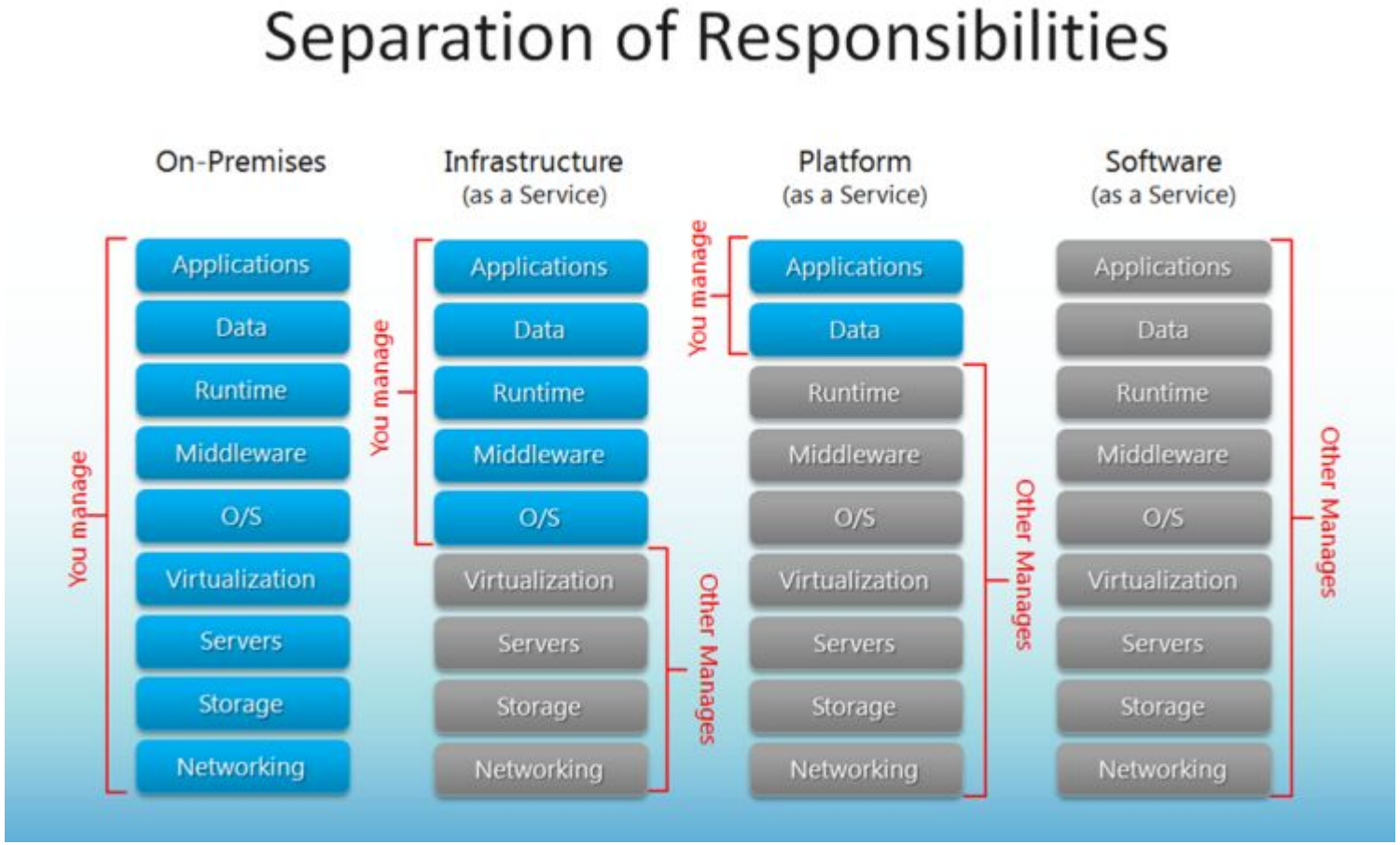
Basic Concepts & Terminologies

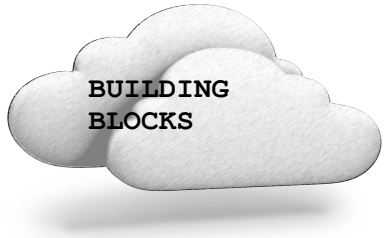
- Cloud (same as internet?)
- IT Resource
- On-Premise
- Cloud Consumers
- Scaling
  - Vertical (Scale U
  - Horizontal (Scale
- Cloud Service
- Cloud Service Co





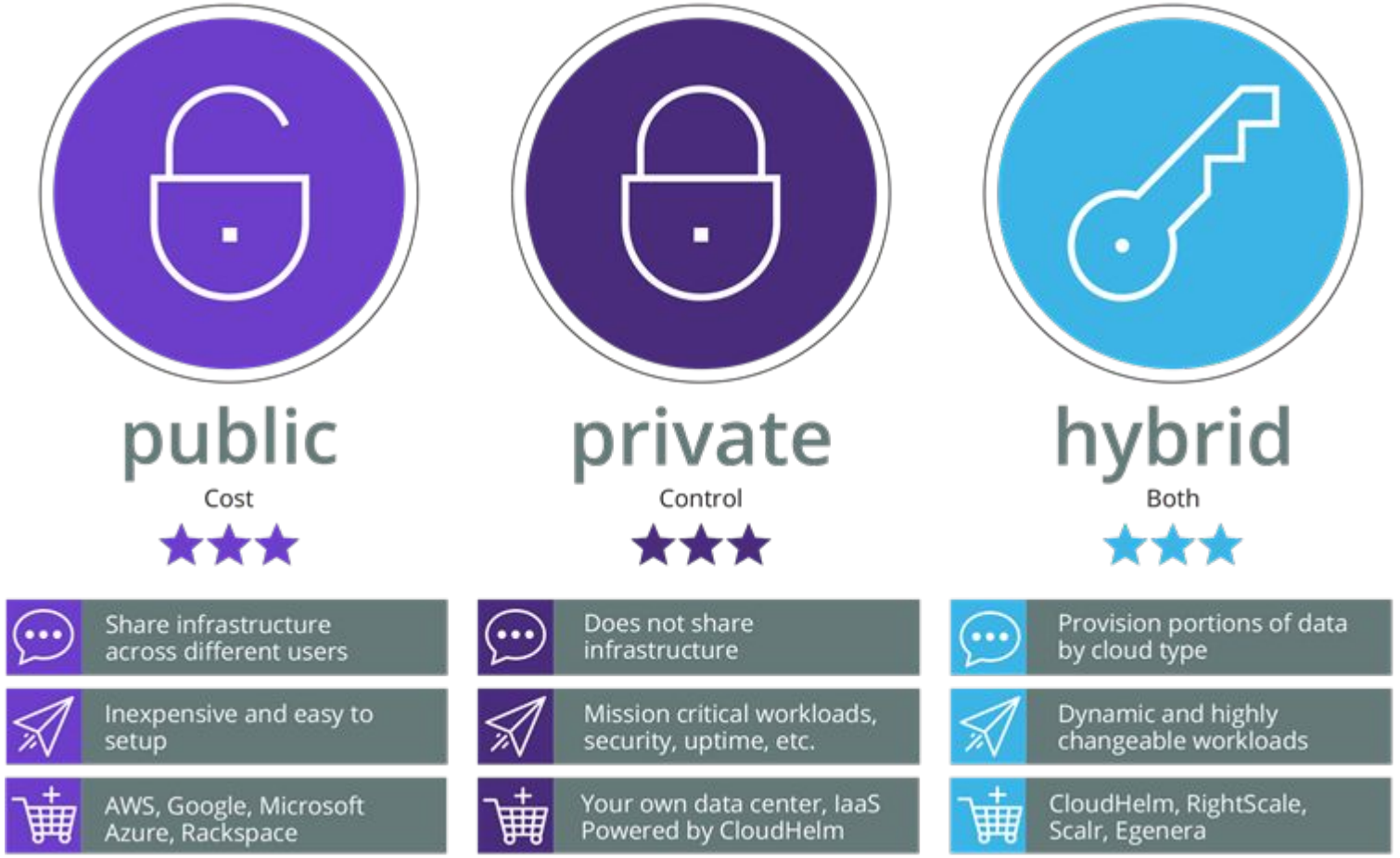
Cloud Delivery Models





Cloud Deployment Models

- Public Cloud
- Private Cloud
- Hybrid Cloud
- Community Cloud



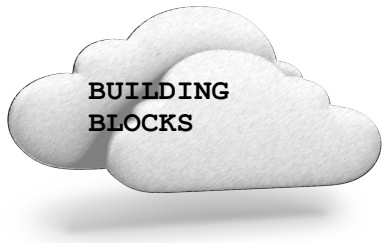
Cloud vs Traditional

Traditional (on-premises) Systems

- Monolithic, centralised
- Design for predictable scalability
- Relational database
- Strong consistency
- Serial and synchronised processing
- Design to avoid failures (MTBF)
- Occasional big updates
- Manual management
- Snowflake servers

Cloud based Systems

- Deconstructed, decentralised
- Design for elastic scale
- Polyglot persistence (mix of storage technologies)
- Eventual consistency
- Parallel and asynchronous processing
- Design for failure (MTTR)
- Frequent small updates
- Automated self-management
- Immutable infrastructure



Cloud Provider – AWS

Amazon Web Services

AWS World

AWS Account

AWS Company Account

AWS Account (Master)

AWS Account

AWS Account

AWS Account

AWS Account

AWS Account

AWS Account

AWS Organizations

OU: Admins

SCP: Admin

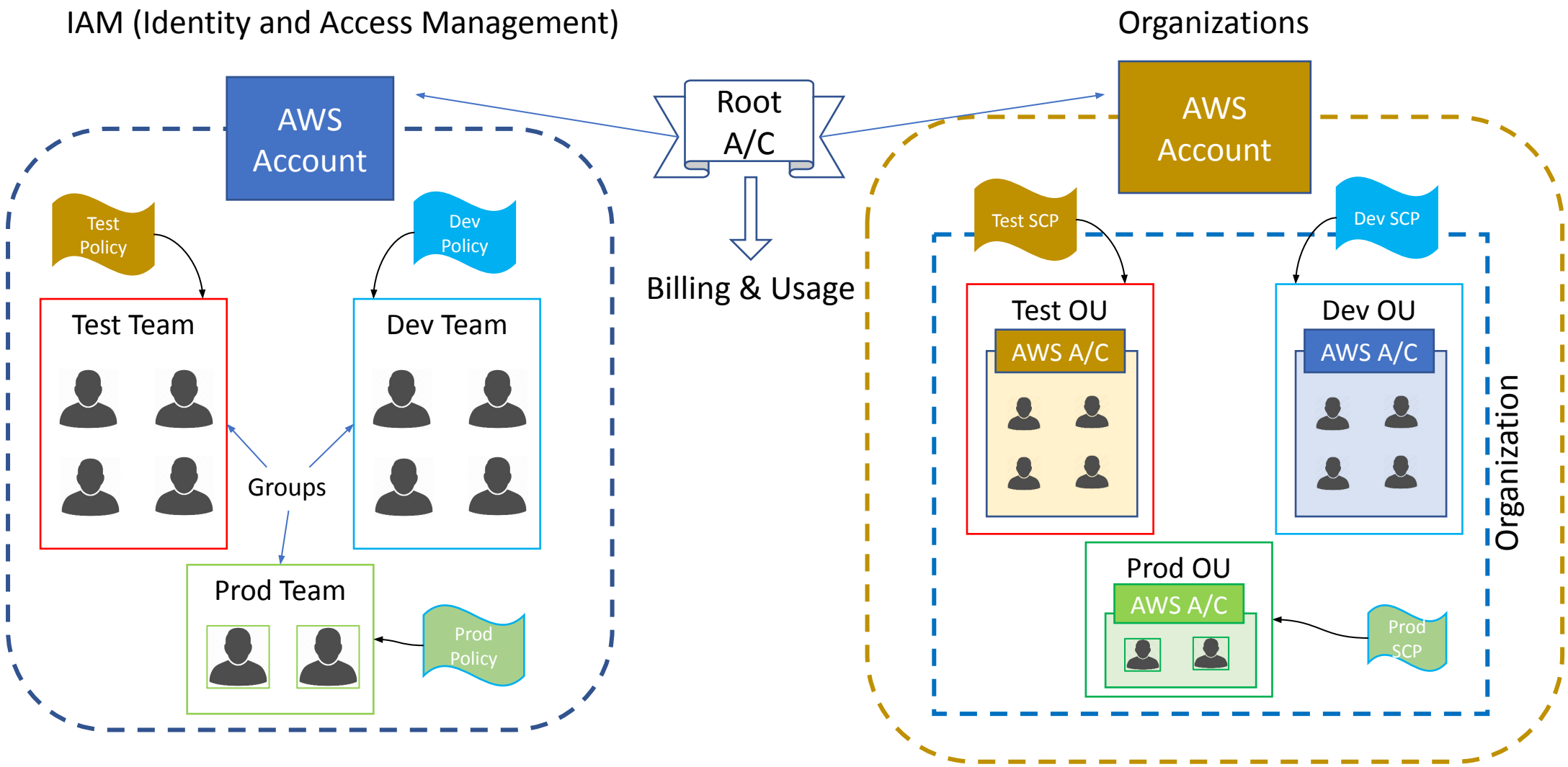
OU: Dev

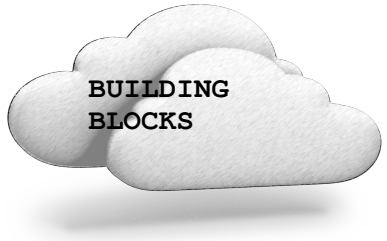
OU: Test

SCP: DevTest

BUILDING BLOCKS

AWS – IAM vs Organizations





Cloud Provider – Azure

Azure World

<id>@Hotmail.com; <id>@outlook.com; <id>@live.com;
Used for accessing various Microsoft services.

Microsoft Account

Organizational Account

1:1

Account Administrator

Azure Account

<id>@<org name>.onmicrosoft.com => Gives a unique domain for accessing Azure services for the org.

Subscriptions

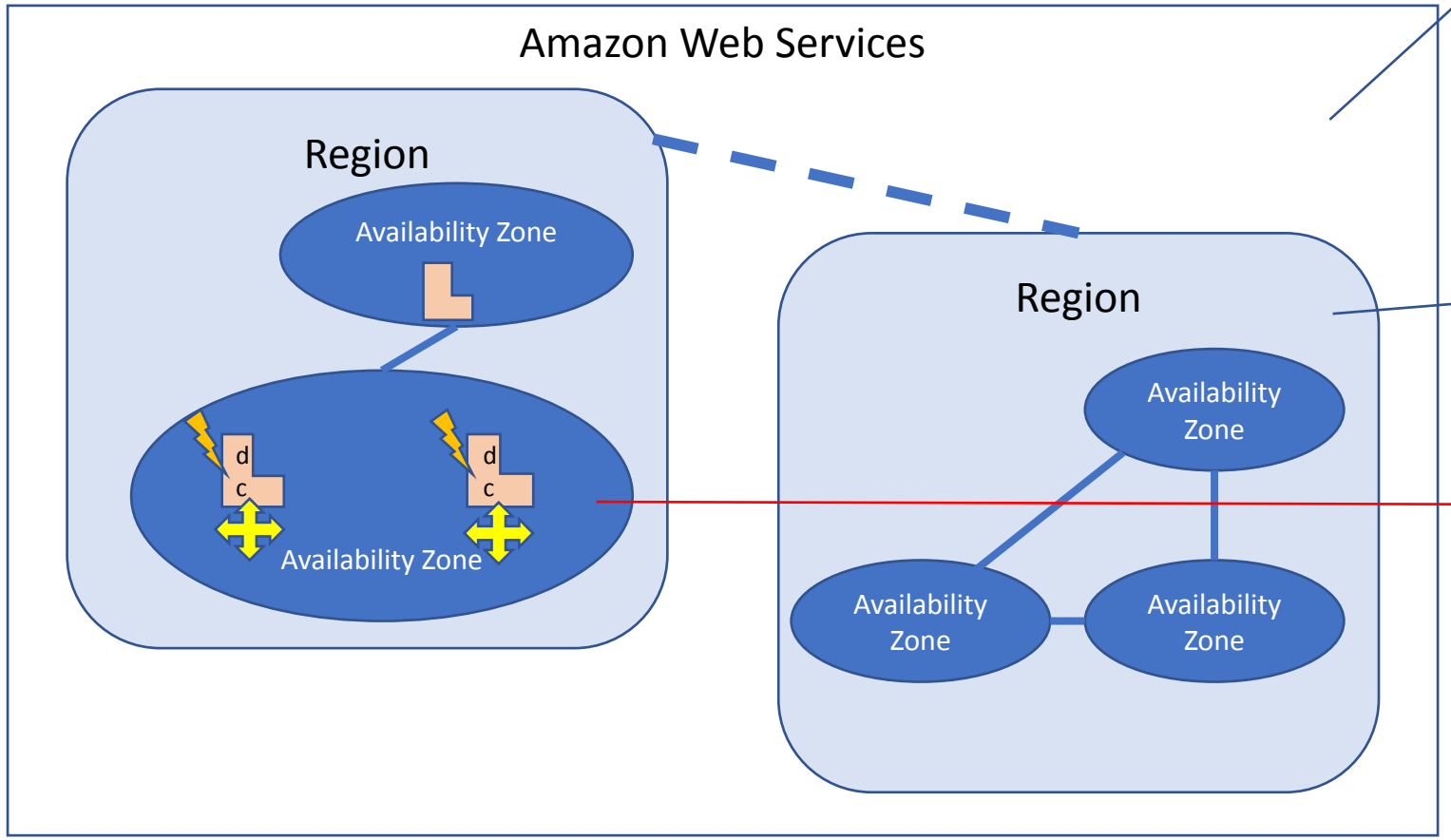
Service Administrator 1:1 Development 1:10 Co-Administrator

Service Administrator 1:1 Staging 1:10 Co-Administrator

Service Administrator 1:1 Production 1:10 Co-Administrator

BUILDING BLOCKS

Data Centres Technology - AWS



Global scope (eg. AWS Account)

Regional scope (eg. EC2 Instance ID)

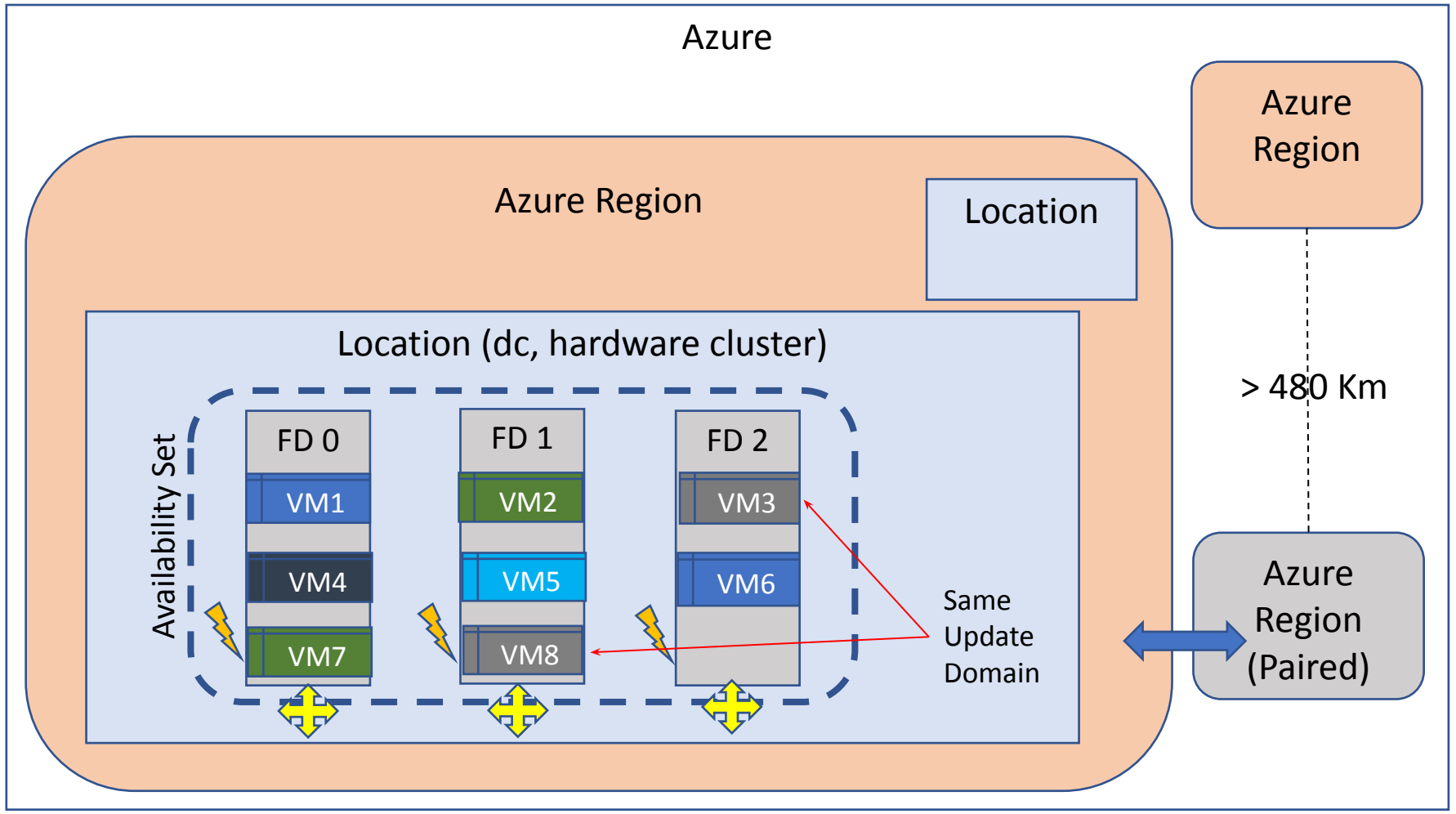
- Regions are completely isolated from other regions
- Greatest possible fault tolerance & stability
- Will not replicate resource across regions automatically
- Charge for data transfer across regions

with fast, private FO networking.

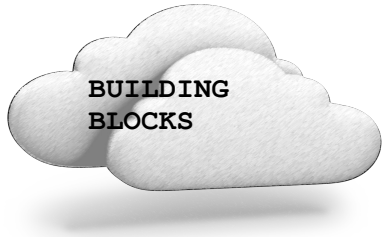
- Distribute instances across AZs for continuity of application.

BUILDING BLOCKS

Data Centre Technology - Azure



- Storage data is triplicated in region and it's paired region
- Availability set is created by user. SLA of 99.95% holds only if two or more VMs are put in an AvSet.
- Fault Domains – up to 3 per AvSet.
- Update Domains – up to 5 (20 via RM) per AvSet.



Quiz time!

Q. The SQL database instance in cloud is throttling. I move it from Standard tier to Premium tier. This move is an example of Horizontal or Vertical scaling?

A. Vertical scaling.

Q. I need to create my resume for placement interviews. I run MS Word online through Office 365 subscription to create my resume. This is an example of IaaS/PaaS/SaaS?

A. SaaS.

Q. I provisioned (created) an VM instance in my AWS cloud for installing a web server. This is an example of IaaS/PaaS/SaaS?

A. IaaS

Q. My company has multiple AWS accounts for running its business in AWS cloud. So many accounts are causing access and tracking confusions. My boss just asked me to look for a solution in AWS. What is that solution?

A. Use AWS organizations to create OUs & assign SCPs and make one AWS account as master account for that organization. The billing then will happen against that master AWS account.

Q. In Azure, an Azure subscription can have multiple Azure accounts. True or false?

A. False. An Azure account can have multiple subscriptions but not vice versa.

Q. In Azure, I create my Azure account and then created a VM under it. All good. True or False?

A. False. Without first creating a subscription, no resources can be created under an Azure account.

Exercise

For a web application, following are the resources that you are creating in Azure: Two servers, S1 & S2, for serving HTTP requests, two data base servers Db1 and Db2 for the databases and B1 & B2 servers for

processing back-end/business jobs for the service. How will you distribute these machines in availability sets, fault and upgrade domains to make a system resilience to failure?

Resource Management in Azure

Azure Resource Manager is the deployment and management service for Azure.

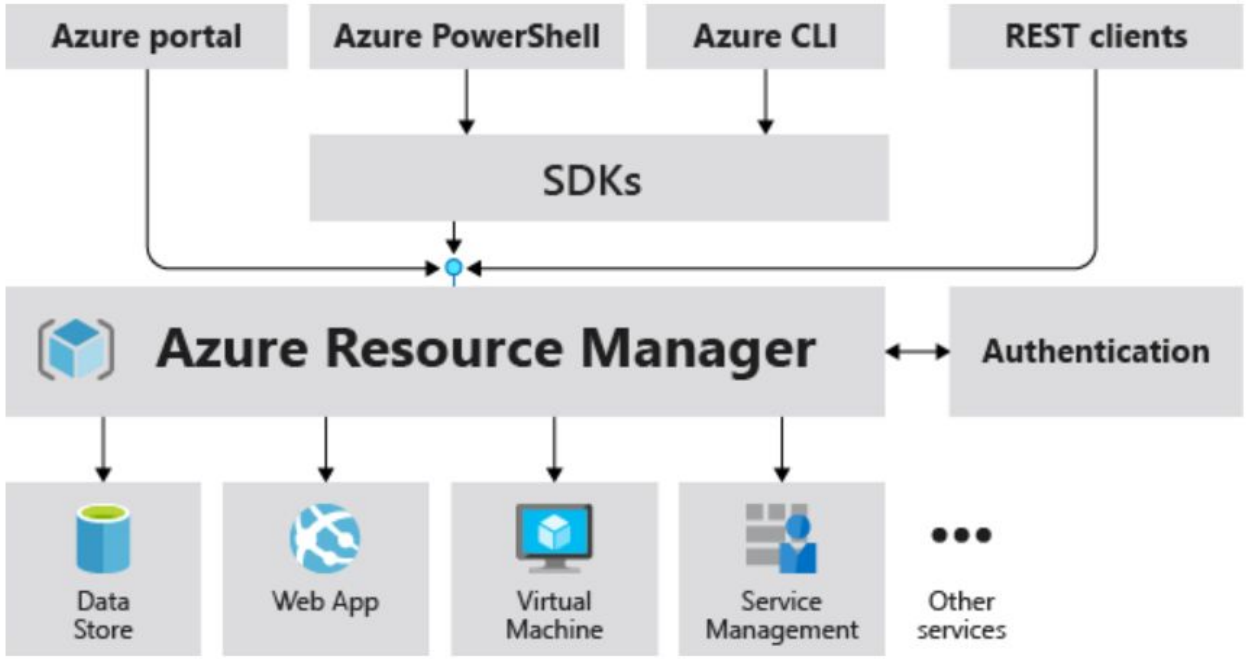
It provides a management layer that enables you to create, update, and delete resources in your Azure account.

You use management features, like access control, locks, and tags, to secure and organize your resources after deployment.

Consistent management layer

When a user sends a request from any of the Azure tools, APIs, or SDKs, Resource Manager receives the request.

When a user sends a request from any of the Azure tools, APIs, or SDKs, Resource Manager receives the request.



Terminology

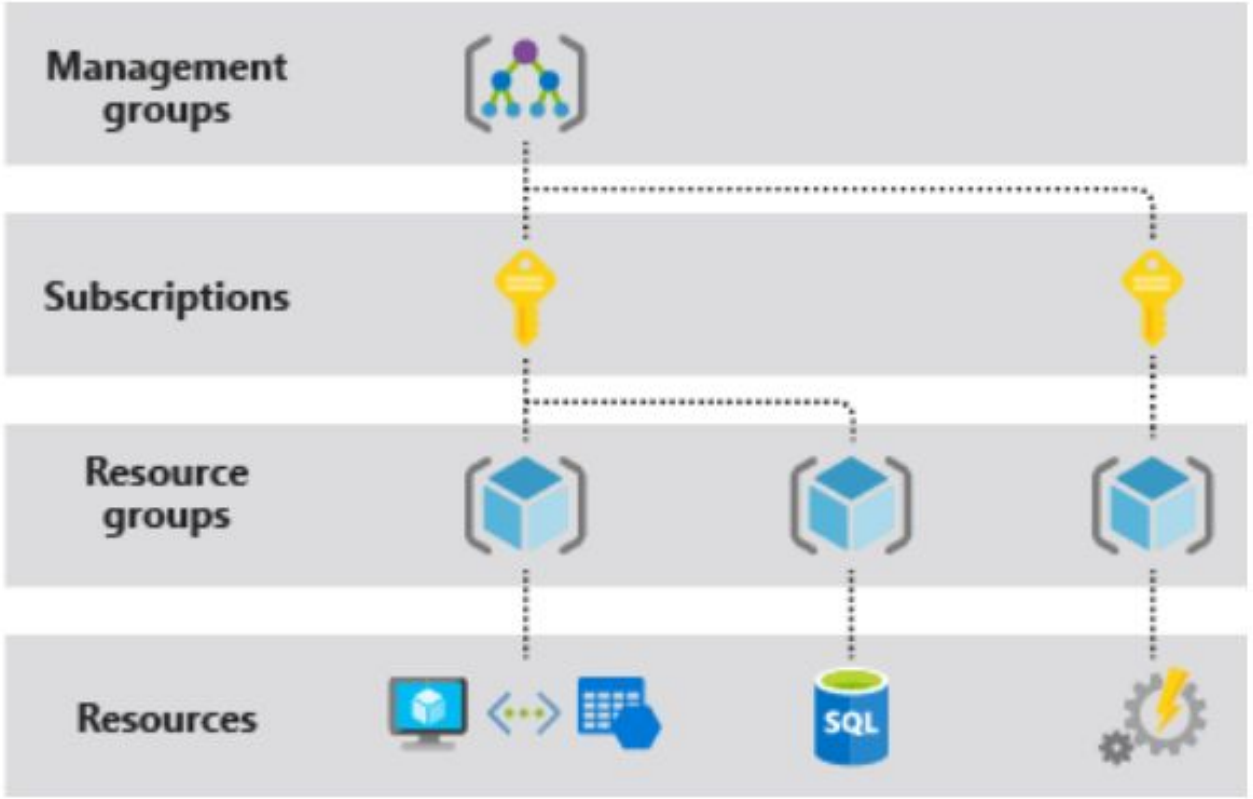
- **resource** – A manageable item that is available through Azure. Virtual machines, storage accounts, web apps, databases, and virtual networks are examples of resources. Resource groups, subscriptions, management groups, and tags are also examples of resources.
- **resource group** – A container that holds related resources for an Azure solution. The resource group includes those resources that you want to manage as a group.
- **resource provider** – A service that supplies Azure resources. For example, a common resource provider is Microsoft.Compute, which supplies the virtual machine resource. Microsoft.Storage is another common resource provider.
- **Resource Manager template** – A JavaScript Object Notation (JSON) file that defines one or more resources to deploy to a resource group, subscription, management group, or tenant. The template can be used to deploy the resources consistently and repeatedly.

Understand scope

Azure provides four levels of scope: management groups, subscriptions, resource groups, and resources.

You apply management settings at any of these levels of scope

Lower levels inherit settings from higher levels.



Resource groups

- All the resources in your resource group should share the same lifecycle. You deploy, update, and delete them together. If one resource, such as a server, needs to exist on a different deployment cycle it should be in another resource group.
- Each resource can exist in only one resource group.
- The resources in a resource group can be located in different regions than the resource group.
- When creating a resource group, you need to provide a location for that resource group.
- A resource group can be used to scope access control for administrative actions.
- A resource can connect to resources in other resource groups. This scenario is common when the two resources are related but don't share the same lifecycle. For example, you can have a web app that connects to a database in a different resource group.
- When you delete a resource group, all resources in the resource group are also deleted.
- Some resources can exist outside of a resource group. These resources are deployed to the subscription, management group, or tenant. Only specific resource types are supported at these scopes.

AZURE AUTOMATION USING AZ CLI

az cli

- The CLI is a tool designed to get you working quickly and efficiently with Azure services, with an emphasis on automation
- Works on Windows, Mac OS, Linux
- Can also be run in Azure portal as cloud shell
- Sign In
  - az login

Common Commands

- Commands in the CLI are organized as *commands* of *groups*.
  - Each group represents an Azure service, and commands operate on that service.

Resource type	Azure CLI command group
Resource group	az group
Virtual machines	az vm
Storage accounts	az storage account
Key Vault	az keyvault
Web applications	az webapp
SQL databases	az sql server
CosmosDB	az cosmosdb

Globally available arguments

--**help** prints CLI reference information about commands and their arguments and lists available subgroups and commands.

--**output** changes the output format. The available output formats are json, jsonc (colorized JSON), tsv (Tab-Separated Values), table (human-readable ASCII tables), and yaml. By default the CLI outputs json. To learn more about the available output formats, see Output formats for Azure CLI.

--**query** uses the JMESPath query language to filter the output returned from Azure services. To learn more about queries, see Query command results with Azure CLI and the JMESPath tutorial.

--**verbose** prints information about resources created in Azure during an operation, and other useful information.

--**debug** prints even more information about CLI operations, used for debugging purposes. If you find a bug, provide output generated with the --debug flag on when submitting a bug report.

Creating a virtual machine using az cli

- az login
- In Azure, all resources are allocated in a resource management group.
  - Resource groups provide logical groupings of resources that make them easier to work with as a collection.
- Create a resource group
  - az group create –name mscitrg –location centralindia
- Create a vm under resource group
  - az vm create --resource-group mscitrg --name testvm1 --image ubuntults --admin-user lavneetsingh --generate-ssh-keys --verbose --output json

- az vm show --name testvm1 --resource-group mscitrg
- az group delete --name mscitrg --no-wait
- az group wait --name mscitrg --deleted

Infrastructure as Code

- With the move to the cloud, many teams have adopted agile development methods. These teams iterate quickly
- They need to repeatedly deploy their solutions to the cloud, and know their infrastructure is in a reliable state
- As infrastructure has become part of the iterative process, the division between operations and development has disappeared
- Teams need to manage infrastructure and application code through a unified process (**DevOps**)

- To meet these challenges, you can automate deployments and use the practice of infrastructure as code.

- In code, you define the infrastructure that needs to be deployed. The infrastructure code becomes part of your project.

- Just like application code, you store the infrastructure code in a source repository and version it.

- Any one on your team can run the code and deploy similar environments.

ARM Templates – Infrastructure as Code in Azure

- The template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project.
- In the template, you specify the resources to deploy and the properties for those resources.
- Hello World template:

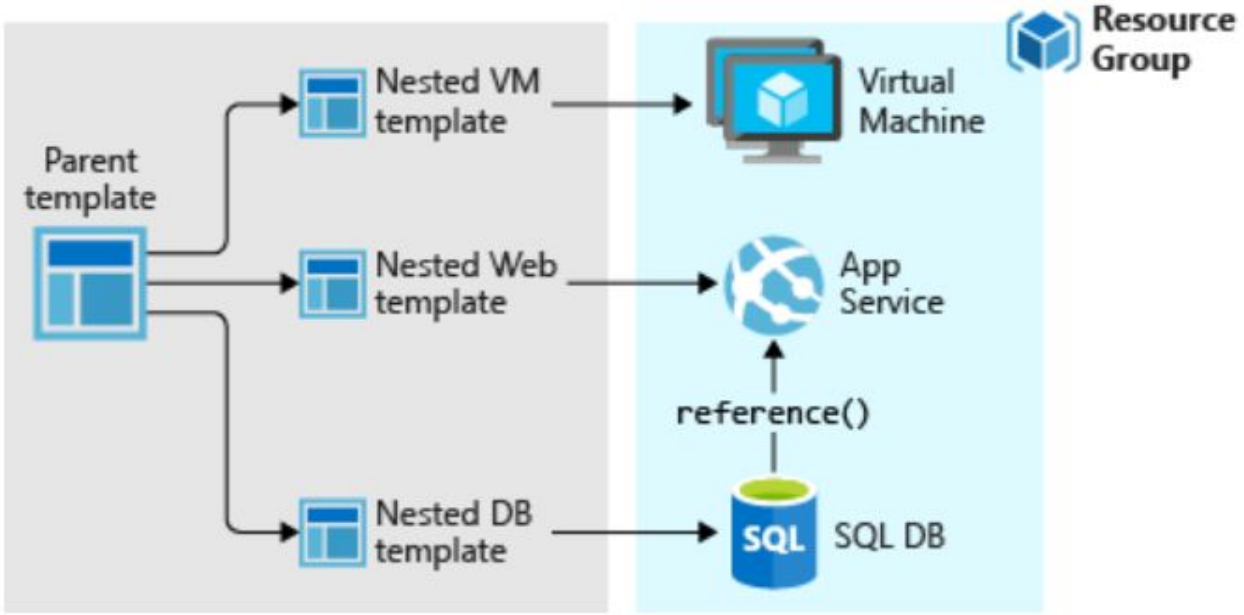
```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "resources": []
}
```

Template files

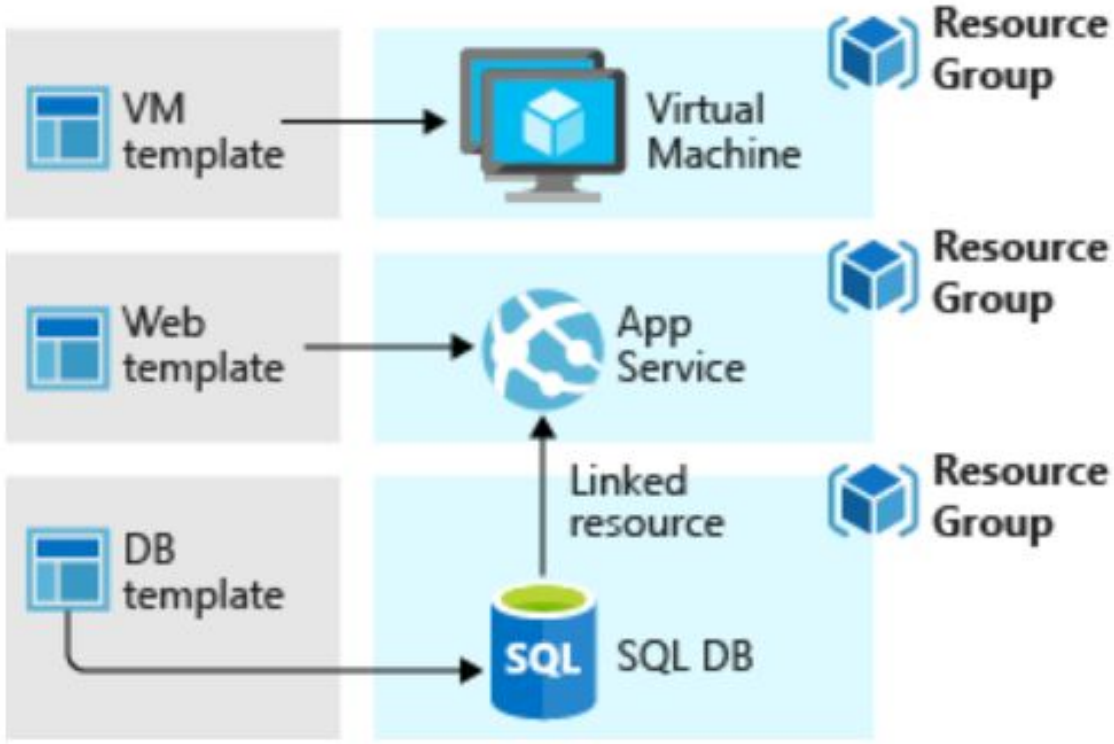
- Within your template, you can write template expressions that extend the capabilities of JSON. These expressions make use of the functions provided by Resource Manager.
- The template has the following sections:
  - **Parameters** - Provide values during deployment that allow the same template to be used with different environments.
  - **Variables** - Define values that are reused in your templates. They can be constructed from parameter values.
  - **User-defined function**s - Create customized functions that simplify your template.
  - **Resources** - Specify the resources to deploy.
  - **Outputs** - Return values from the deployed resources.

Template Design

- You don't have to define your entire infrastructure in a single template.
- Often, it makes sense to divide your deployment requirements into a set of targeted, purpose-specific templates
- To deploy a particular solution, you create a master template that links all the required templates.



- If you envision your tiers having separate lifecycles, you can deploy your three tiers to separate resource groups.
- The resources can still be linked to resources in other resource groups

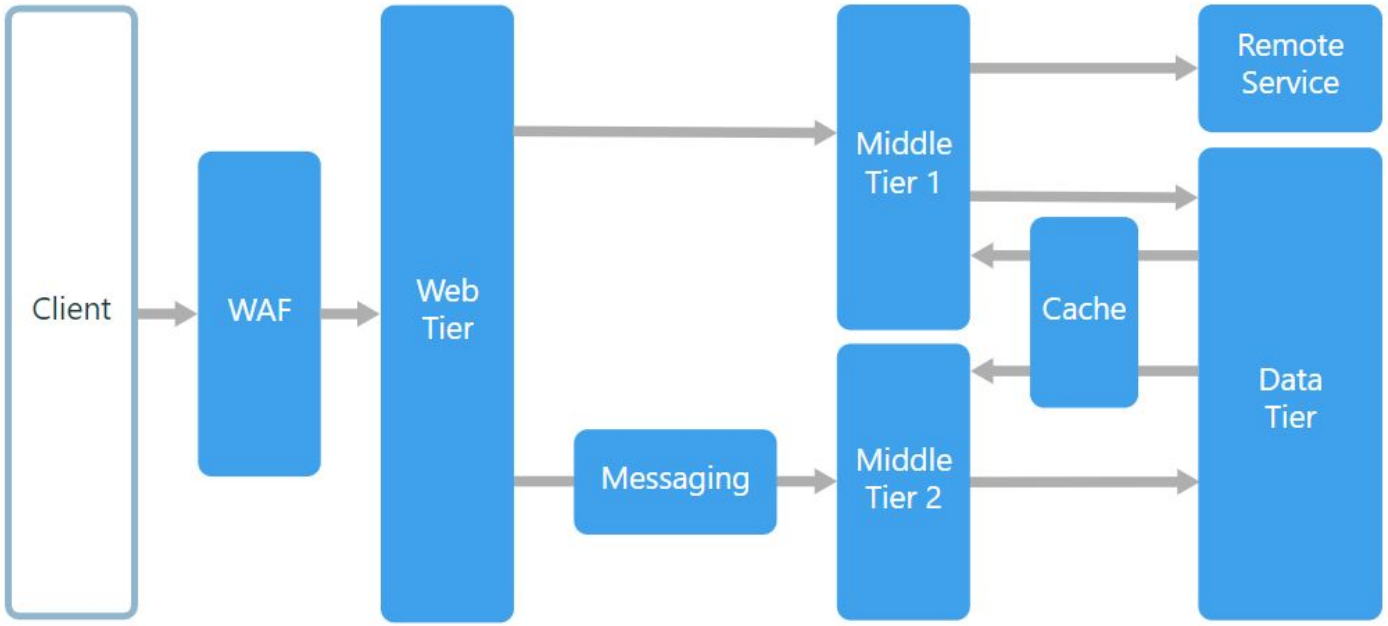


Cloud Architectures

N-Tier	• Traditional layered • Higher layer calls lower ones, not vice-versa • Can be a liability – not easy to update individual components • Suitable for app migration • Generally IaaS
Web-Queue-Worker	• Purely PaaS • Web FE + Worker BE • FE <-> BE communication through async msg Q • Suitable for simple domains
Microservices	• Many small independent services • Services are loosely coupled • Communicate through API contracts
Event-driven Architecture	• Uses pub-sub model • Pubs & Subs are independent • Suitable for apps that ingest large volume of data • Suitable when subsystems need to perform different actions on same data
Big Data Big Compute	• Specialized architecture that fit certain profiles • Divides large datasets into chunks, parallel process and analysis

N-tier Architecture Style

- An N-tier architecture divides an application into **logical layers** and **physical tiers**.
- Layers are a way to separate responsibilities and manage dependencies.
- Tiers are physically separated, running on separate machines.

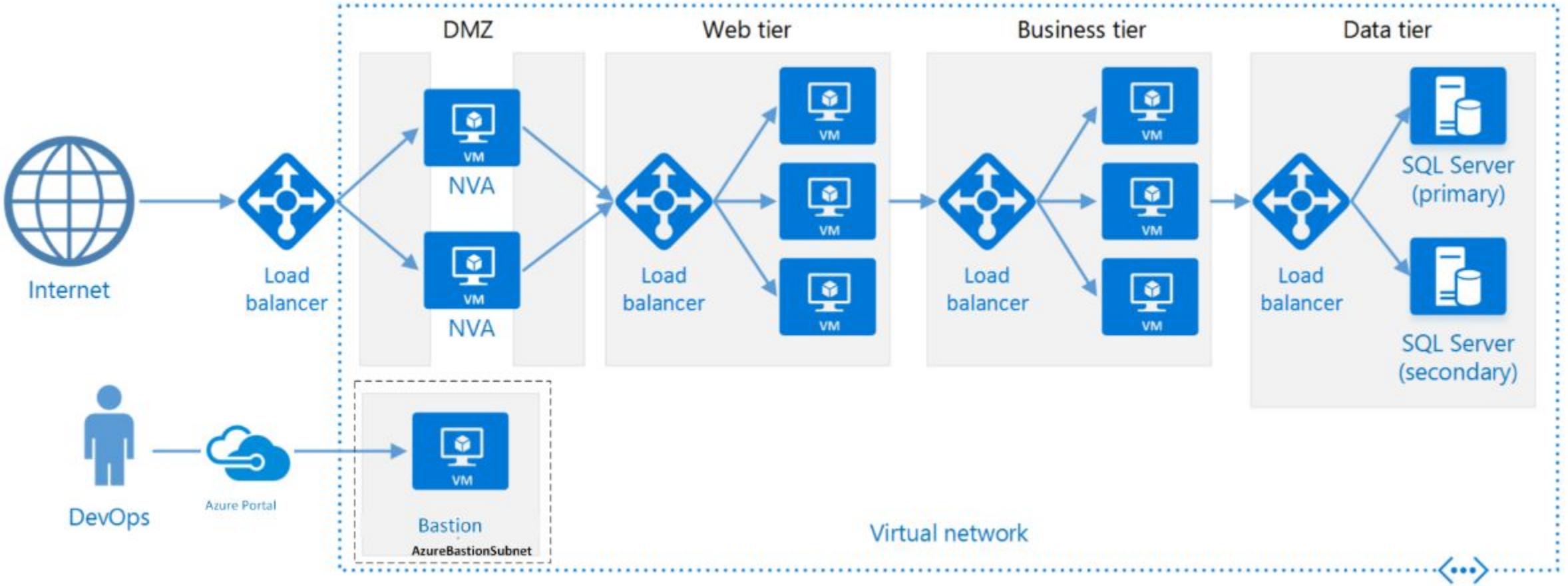


When to use this architecture

- Typically implemented as infrastructure-as-service (IaaS) applications, with each tier running on a separate set of VMs.
- Consider an N-tier architecture for:
  - Simple web applications.
  - Migrating an on-premises application to Azure with minimal refactoring.
  - Unified development of on-premises and cloud applications.

- **Best practices**
- Use autoscaling to handle changes in load. See Autoscaling best practices.
- Use asynchronous messaging to decouple tiers.
- Cache semi static data. See Caching best practices.
- Configure the database tier for high availability, using a solution such as SQL Server Always On availability groups.
- Place a web application firewall (WAF) between the front end and the Internet.
- Place each tier in its own subnet, and use subnets as a security boundary.
- Restrict access to the data tier, by allowing requests only from the middle tier(s).

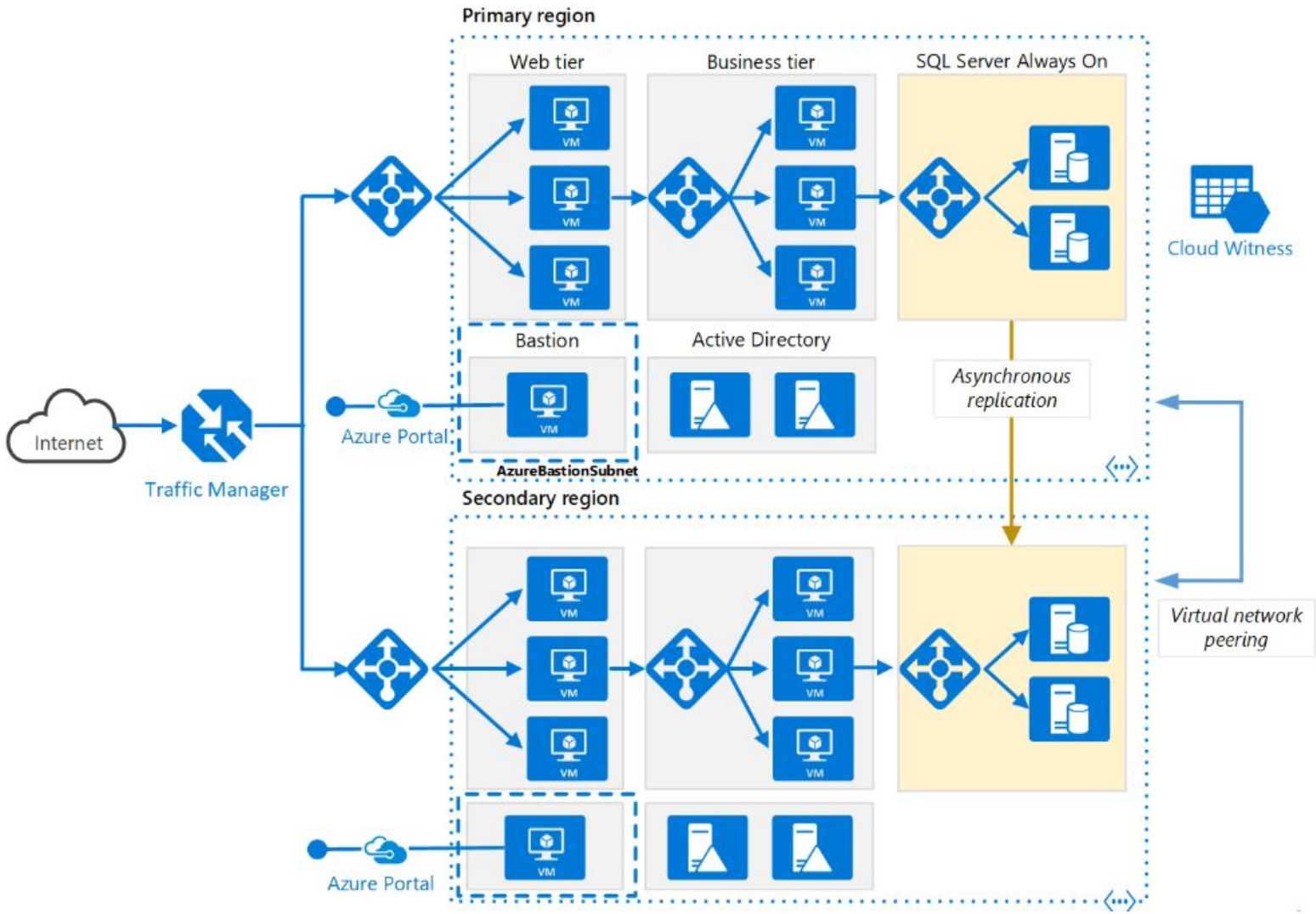
N-tier architecture on Virtual Machines



- Each tier consists of two or more VMs, placed in an availability set or virtual machine scale set.
- Multiple VMs provide resiliency in case one VM fails.
- Load balancers are used to distribute requests across the VMs in a tier.
- A tier can be scaled horizontally by adding more VMs to the pool.
- Each tier is also placed inside its own subnet, meaning their internal IP addresses fall within the same address range.
- That makes it easy to apply network security group rules and route tables to individual tiers.
- The web and business tiers are stateless. Any VM can handle any request for that tier. The data tier should consist of a replicated database.
- Network security groups restrict access to each tier. For example, the database tier only allows access from the business tier.

Role of Application Gateway

- Web traffic load balancer enables us to manage traffic to our web applications
- Operates at application level (OSI level 7 - Application).
  - Traditional load balancers operate at transport layer (OSI level 4 – TCP & UDP) and route traffic based on IP address and port.
  - Application Gateway can route traffic based on URL
- SSL termination
  - supports terminate encryption at gateway, so data travels un-encrypted to the backend servers. Saves overhead.
- Web application firewall – protection against common exploits
- Multiple site hosting, Redirection, Session affinity etc.

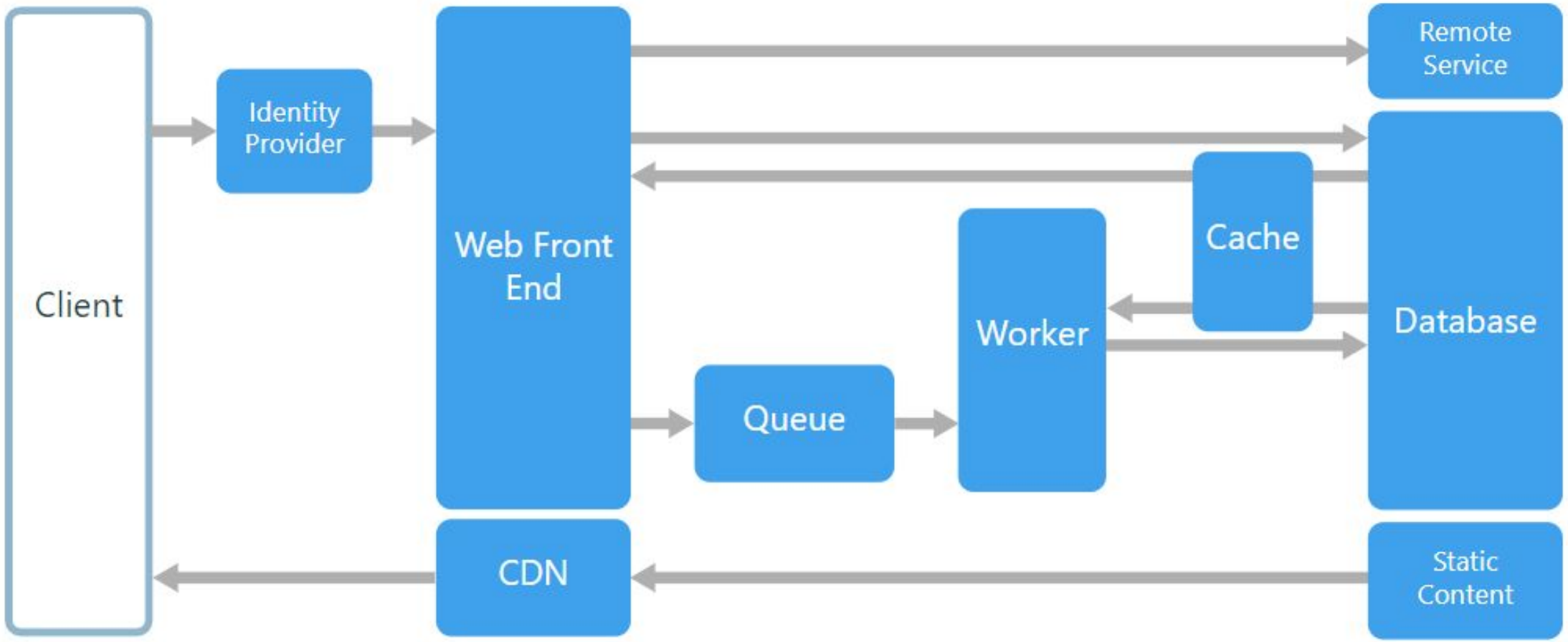


N-tier architecture – Multi-region deployment

Role of Traffic Manager in Multi-region deployment

- Traffic Manager allows users to control traffic distribution across various application endpoints.
- The Traffic Manager distributes the traffic using the configured routing method
  - Priority, Weighted, Performance, Geographic
- Constant monitoring of the endpoint health.
- Automatic failover in case the endpoints fail.
- Functions at the DNS level. It uses the DNS to route clients to precise service endpoints, according to the traffic-routing rules and methods.
- Traffic Manager operation modes:
  - Active | Passive – hot failover
  - Active | Passive – cold failover
  - Active | Active – load balanced

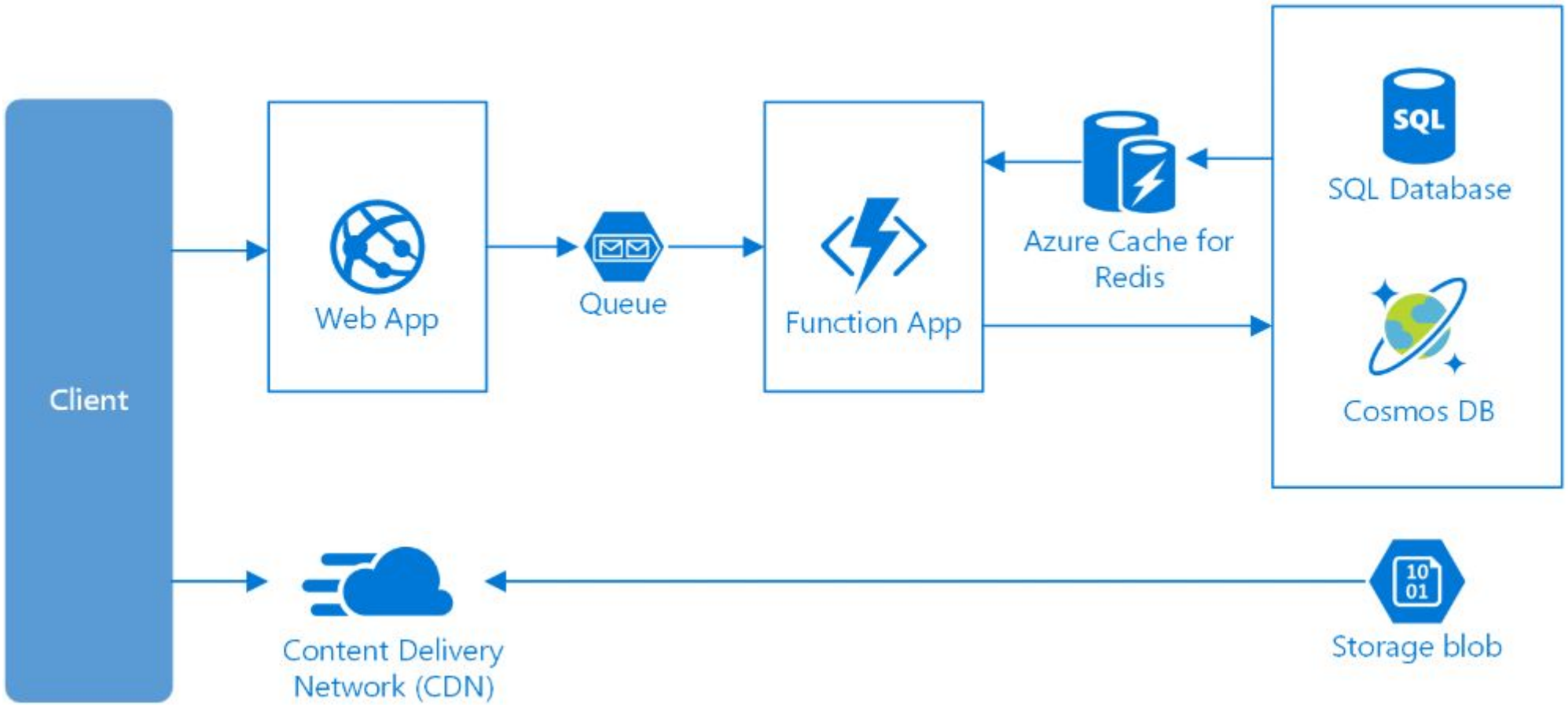
Web-Queue-Worker architecture



- The core components of this architecture are
  - a **web front end** that serves client requests,
  - a **worker** that performs resource-intensive tasks, long-running workflows, or batch jobs.
  - a **message queue –** the web front end communicates with the worker through it.
- Other components that are commonly incorporated into this architecture include:
  - One or more databases.
  - A cache to store values from the database for quick reads.
  - CDN to serve static content
  - Remote services, such as email or SMS service. Often these are provided by third parties.
  - Identity provider for authentication.

- The web and worker are both stateless.
- Session state can be stored in a distributed cache.
- Any long-running work is done asynchronously by the worker.
- The worker can be triggered by messages on the queue or run on a schedule for batch processing.
  - The worker is an optional component. If there are no long-running operations, the worker can be omitted.
- The front end might consist of a web API.
- On the client side, the web API can be consumed by a single-page application that makes AJAX calls, or by a native client application.

Web-Queue-Worker on Azure



- Consider this architecture style for:
  - Applications with a relatively simple domain.
  - Applications with some long-running workflows or batch operations.
  - When you want to use managed services, rather than infrastructure as a service (IaaS).

Cloud Storage

Scenario: Disks & File system in cloud for VMs

- VMs in cloud need disk drive for
  - Operating System
  - Temporary Data
  - Data storage

- Applications running on VMs may also want to access files in a file system just like they do while running on actual h/w machine. Important for
  - cloud migration
  - shared file stores
  - developer tools etc.

AWS Solution: EFS & EBS

Azure Solution: Azure Disks & Azure Files

- Azure Files offers fully managed file shares in the cloud that are accessible via the Server Message Block (SMB) protocol.
  - ideal replacement for file shares on-premises
- Azure file shares can be mounted concurrently by cloud or on-premises deployments of Windows, Linux, and macOS.
  - share a file system across multiple machines, applications/instances.
- Azure file shares can be cached on Windows Servers with Azure File Sync for fast access near where the data is being used.

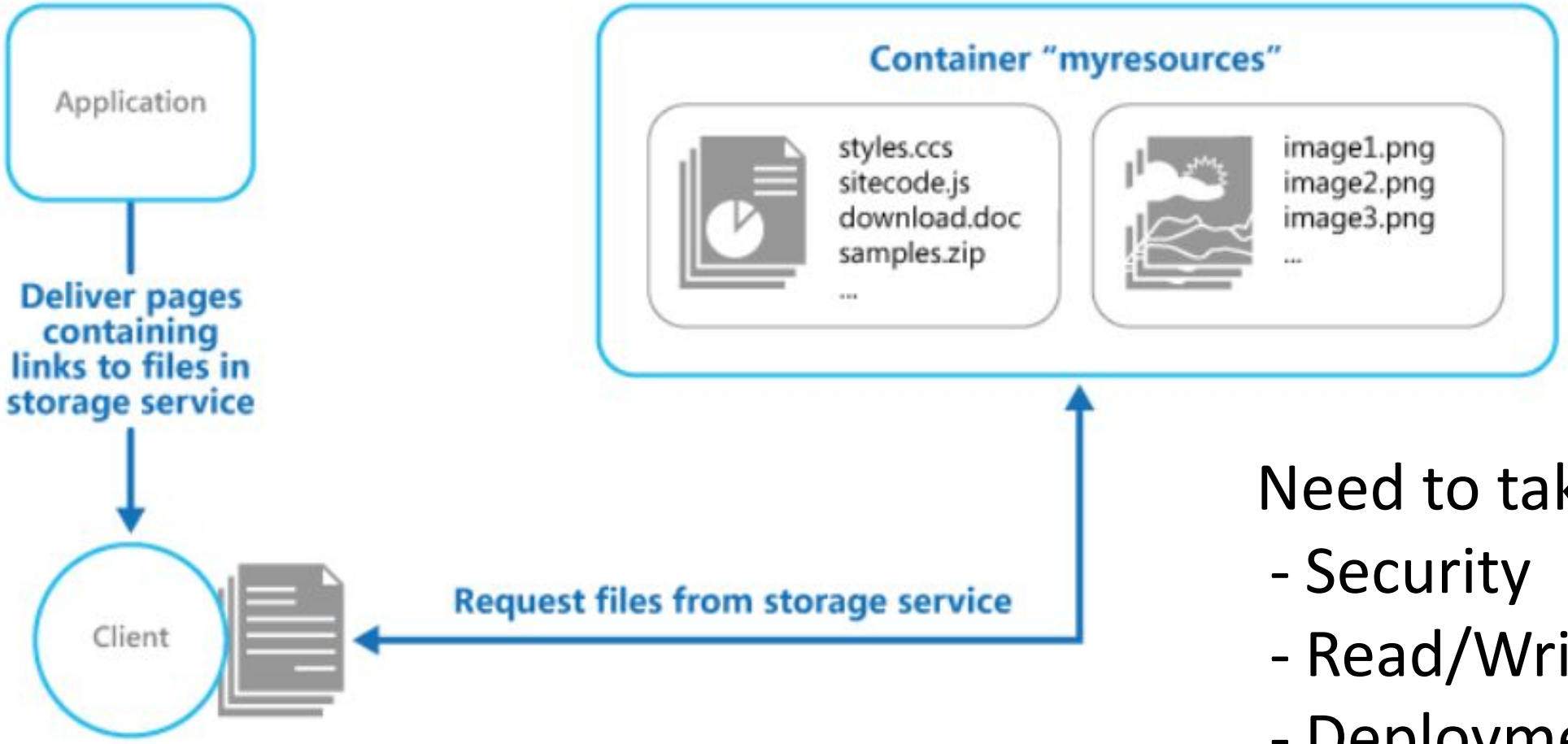
- Azure Disks: OS disk, Temporary disk, Data disk. Unmanaged and Managed disks. Always use managed disks

Scenario: Serving Static web content and media

- Static web content typically include HTML pages and other resources like images, video and documents – scripts, pdf downloads etc.
- Although web servers are well tuned to optimize requests through efficient dynamic page code execution and output caching, they still have to handle requests to download static content.
  - This means spending money on CPU cycles

GOAL: Minimise need for compute instances

Solution: URL based storage service



Need to take care of –

- Security
- Read/Write access
- Deployment

AWS Solution: Amazon S3
(Simple Storage Service)

- Store data objects from 1 byte to 5 TB of data.
- **Each object has its public / access URL**
- Highly scalable – allows concurrent read and write access to data by many separate clients or application threads
- In same region using S3 with EC2 instance (ie. a VM) is designed to be fast.

- Can’t be used in place of file system
- Can’t query data
- Not suitable for very rapidly changing data
- Better solutions available for archiving data
- Not suitable for dynamic website hosting

Azure Solution: Blob storage

- Blob storage is optimized for storing massive amounts of unstructured data, such as text or binary data
- **Each blob / object has a public URL**
- Storage account -> Containers -> Blobs
- Three types of blobs – block, append and page
  - Block blobs store text and binary data, up to about 4.7 TB. Block blobs are made up of blocks of data that can be managed individually.
  - Append blobs are made up of blocks like block blobs, but are optimized for append operations. Append blobs are ideal for scenarios such as logging data from virtual machines.
  - Page blobs store random access files up to 8 TB in size. Page blobs store the VHD files that back VMs.

- Same type of constrains like in S3 (no queries, search, file based access etc.)

Scenario: Static Content Via CDN & Archiving

- Content Delivery Network (CDN) nodes are use to service static content. Provides additional options like caching, routing optimizations.
- AWS CloudFront – a CDN solution that can be used with S3 in the back-end for a CDN based static content delivery solution
- Azure CDN – global solution for rapidly delivering high bandwidth content to users.

- Archived data should cost less as it is accessed very infrequently.
- AWS Glacier – an archiving solution that costs much less than S3.
- Azure Blob – has archive tier (in addition to hot & cool tiers)

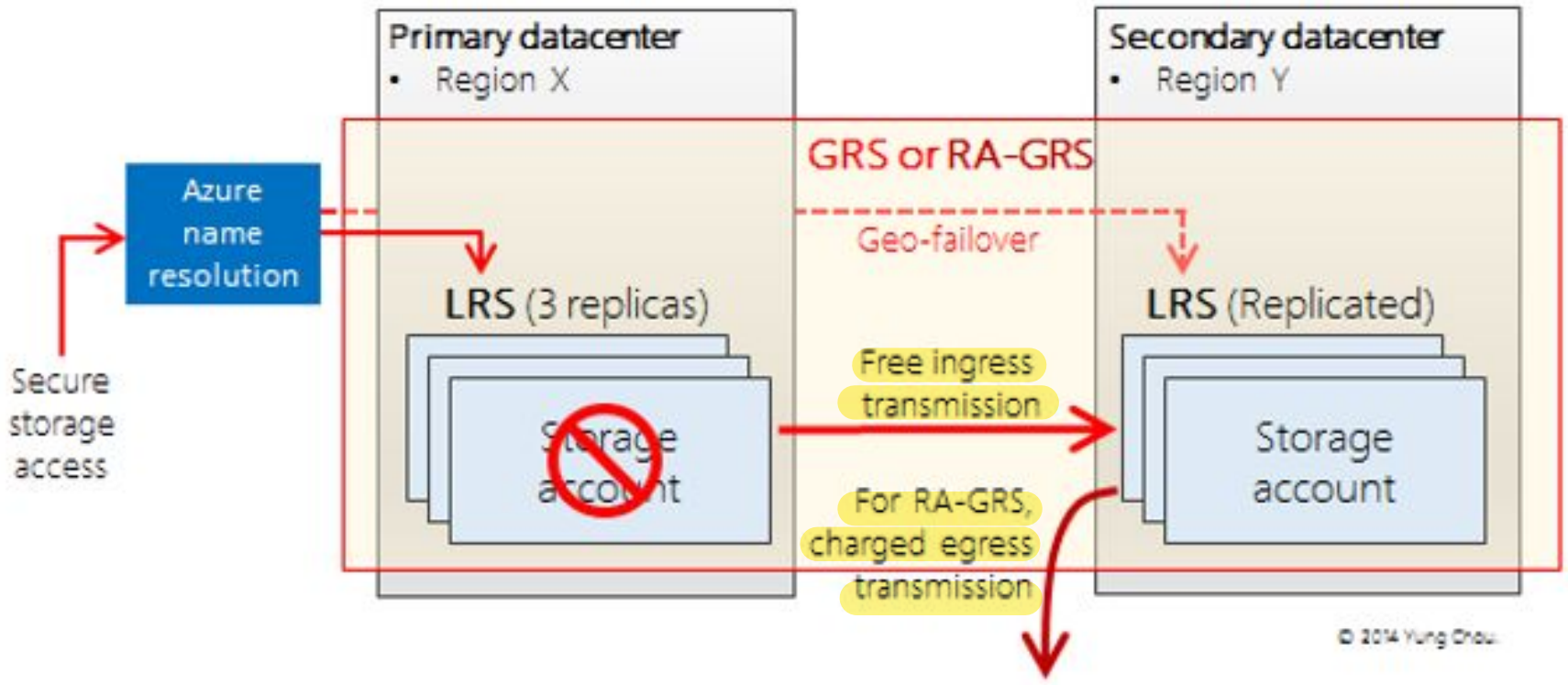
Azure Storage - Redundancy

Locally Redundant Storage (LRS)

Geo-Redundant Storage (GRS)

Read-Access Geo-Redundant Storage (RA-GRS)

Zone-Redundant Storage (ZRS)



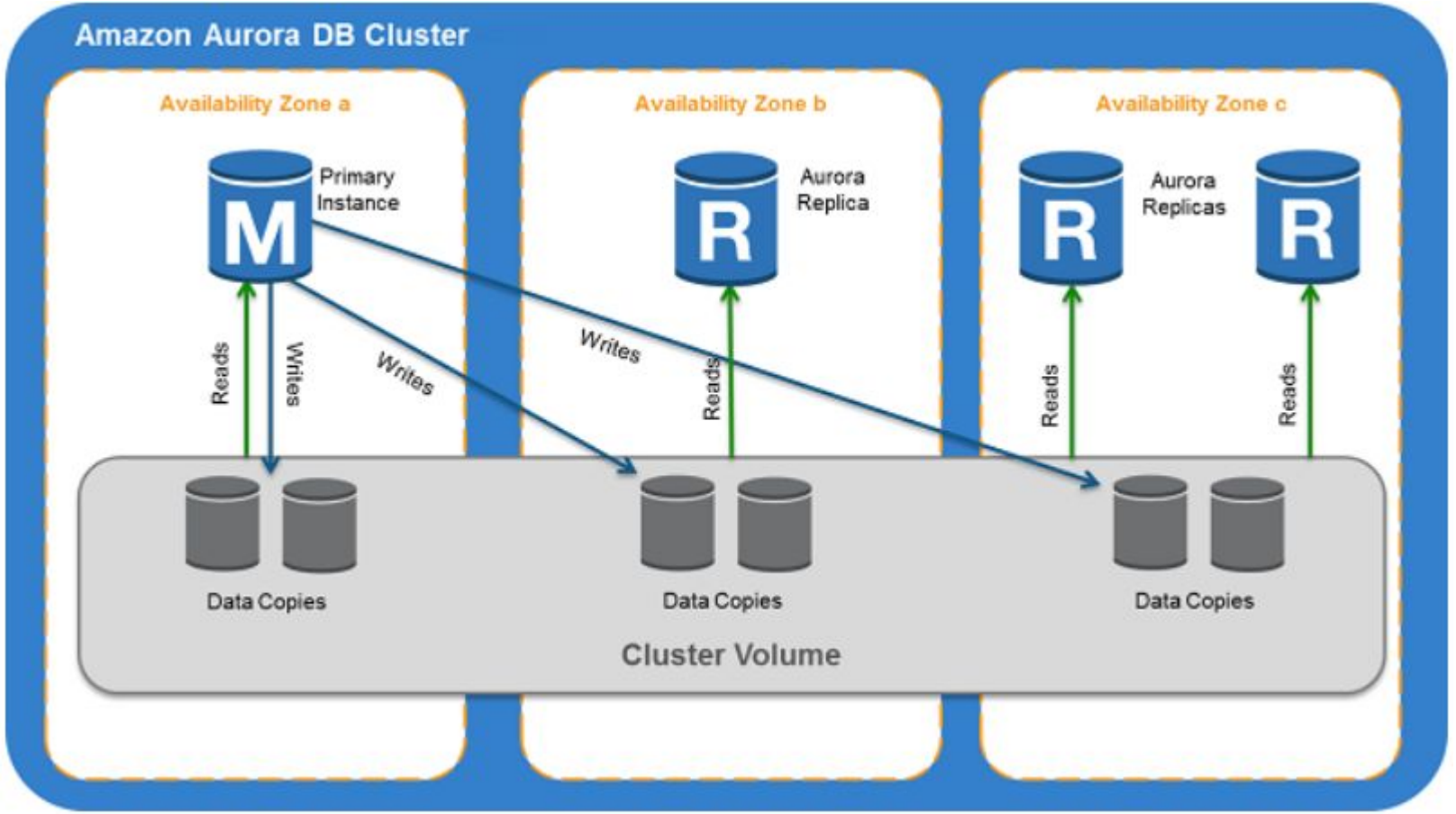
Scenario: Query-able Relational Database (PaaS) in cloud for transactional apps

- Transactional scenarios need to use RDBS
- RDBM on cloud needs to
  - dynamically scale
  - transactional processing
  - PaaS service, hence no infrastructure management
  - High availability
  - Business continuity and Disaster Recovery

AWS Solution: AWS RDS (Relational Database Services)

AWS Aurora – PaaS DB Service

- Fully managed, MySQL- and PostgreSQL-compatible, relational database engine.
- DB cluster defines one or more DB instances + a cluster volume spanning different AZs.
- End points – cluster, reader and instance.



Azure Solution: SQL Database service

Scenario: NoSQL Databases in Cloud

- Motivation to use NoSQL DBs
  - Bigness
    - NoSQL is key part of new data stack supporting big data – big number of users, big number of computers, big supply chains, big science and so on.
    - Data stored all around in a distributed manner does not fit well with RDBMS
  - High Volume
    - FB needs to store 135 billions messages a month, Twitter has problem of storing 7 TB data per day.
    - All such data needs to be distributed
  - Fast Key-Value access
    - Most NoSQL solutions provide hashing on a key and reading its value directly from the memory.
  - Flexible Schema and Flexible Data types
    - NoSQL products support a whole range of new data types
    - Complex objects can be easily stored without a lot of mapping.
    - Lack of structure allows much more flexibility

- Cost of using NoSQL
  - Things taken for granted in RDBMS –
    - Transactional atomic writes
    - Indexing of non-key columns
    - Query optimizing
    - Declarative set oriented queries

    – are sacrificed in NoSQL.
- NoSQL – Provides AP of the CAP theorem – i.e. Consistency is sacrificed.
- SQL – Provides CA of the CAP theorem – Partition tolerance is sacrificed.

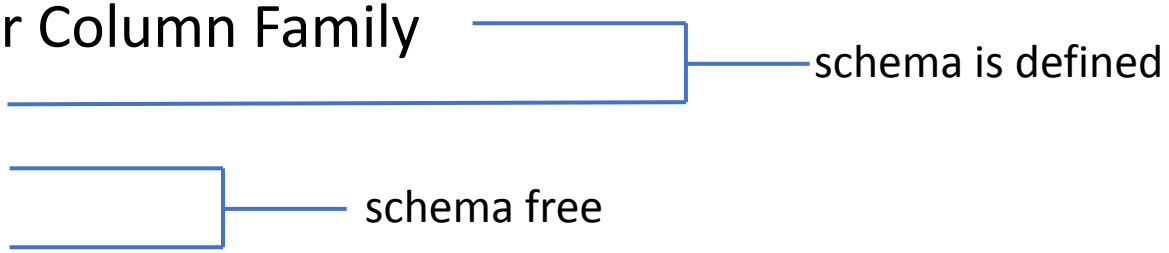
Categories of NoSQL DBs

- Key-Value Stores
  - Data stored in key and value pairs
- Document Stores
  - Stored record is a ‘document’
- Wide Column Stores
  - Semi-schematized and semi-hierarchical
- Graph Databases
  - Explicitly track relationship between entities

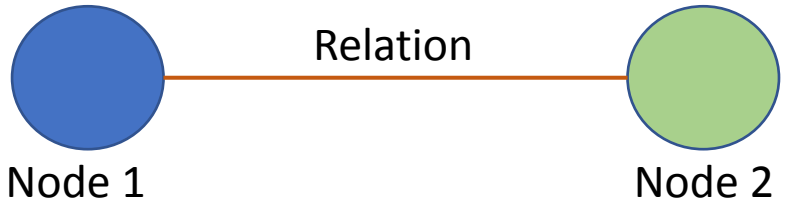
- Key Value stores
  - Structure of one record can differ from other in the same collection
  - This concept is basis of other NoSQL categories as well
  - Cache like speed and simpler data
  - most of the data is non-hierarchical – so RDBMS is not required for such data
  - Data is simple, but storage technology mush be more capable
    - Repos are distributed
    - Fault tolerance
    - Fast
    - Highly available
  - Examples
    - Azure Table Storage
    - Memcached DB
    - Voldmort of linked-in

- Document Stores
  - A record is a ‘document’
  - Each document is a set of Key-Values
  - JSON Object <-> Document for many NoSQL DBs
  - Supports Versioning
  - Documents have URLs, therefore, REST friendly
  - HTTP orientation of Document Stores –
    - CouchDB has Show functions that return HTML
    - HTML is stored in ‘design’ documents
    - Each show function is accessible via URL
    - So, entire app could be implemented in a Document DB
  - Examples
    - Azure Document DB (Now subsumed in Cosmos DB)
    - Apache Couch DB
    - Amazon Dynamo DB
    - Open Source Mongo DB

- Wide Column Stores
  - Semi-structured and Semi-hierarchical
  - Also called Column Family stores
  - Table Name – Super Column Family
    - Super Columns
      - Column 1
      - Column 2



- Graph Databases
  - Explicitly track relationships between entities

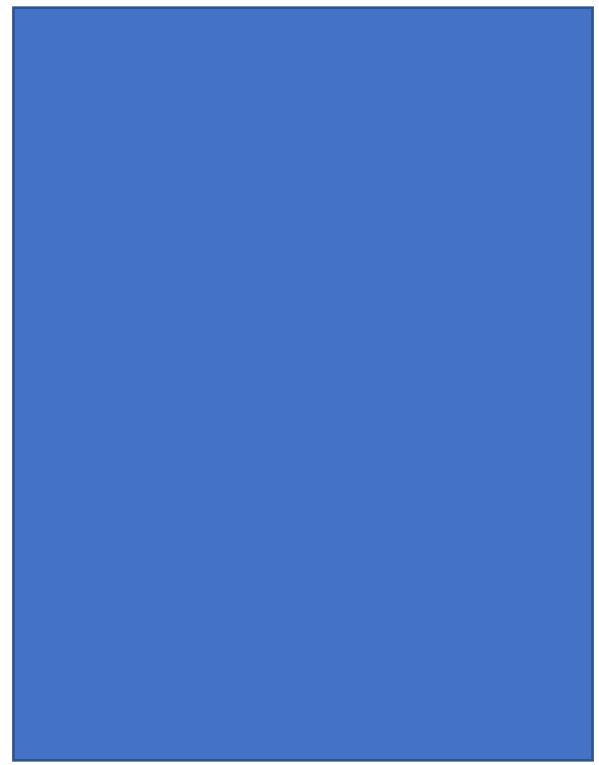


    - e.g. Chris City Auckland

  - Examples
    - Allegro Graph
    - Neo4j
    - Twitter's Flock DB

NoSQL – Common Traits

GOOGLE	APACHE / OPEN SOURCE



Why Parallel Job Processing Engines?

- Querying NoSQL is less straightforward than RDBMS
  - data is stored in less structured way
  - data stored is distributed
- Queries are therefore broken up and executed across multiple repos on different servers
- Sub-query’s results need to be collected and unified

Common Consistency Models

- Eventual Consistency
  - Changes made in one replica will be transmitted to others in asynchronous manner
  - There is propagation delay which might fetch older data
  - Eventually, the queries will find the correct data

- Optimistic Concurrency
  - No locking of rows by single user
  - In low contention of data scenarios
  - No persistent connection to db server needed
  - DB servers free to serve large number of users in lesser time

- Buffered Writes
  - Writes in aggregates

Common NoSQL Indexing Models

- Most of the NoSQL databases index on keys used for rows / entities / documents and / or partitions
- CouchDB indexes Views as well
- Allegro Graph indexes everything (Id, Subject, Predicate, Object & groups)
- Some NoSQL DBs provide indexing on secondary columns
  - Cassandra

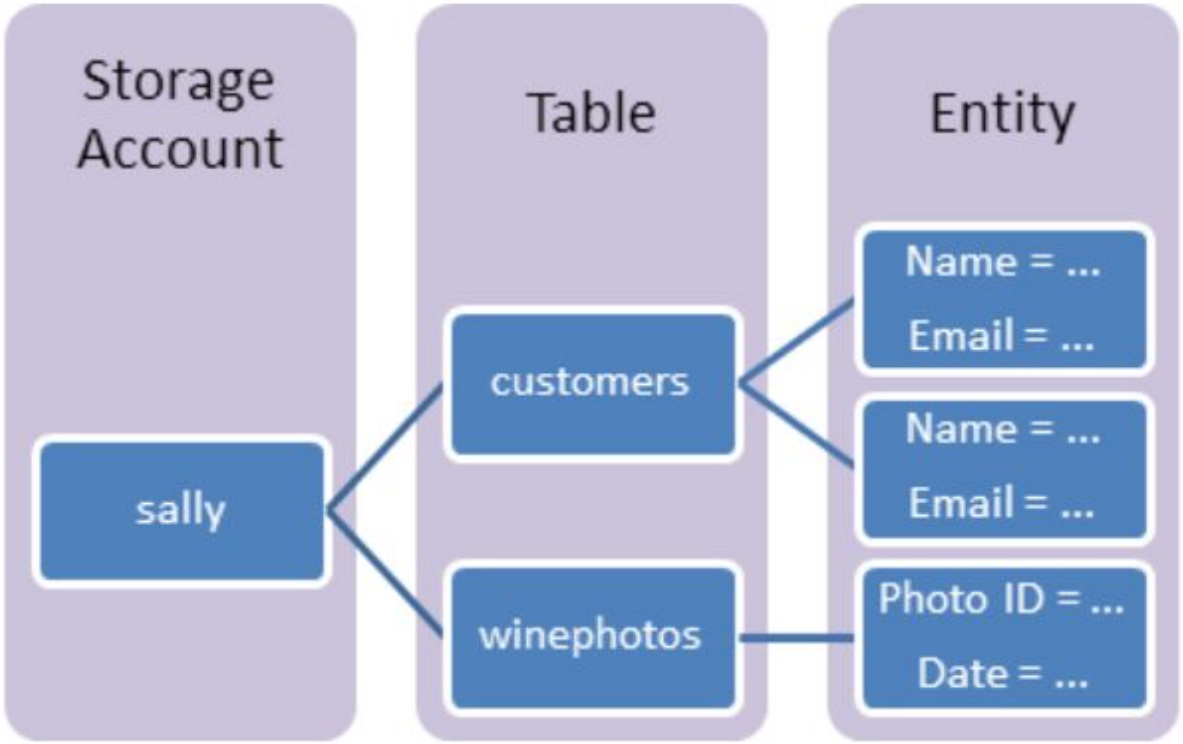
Common NoSQL upsides / advantages

- Light weight, low friction
  - easy provisioning, deployment
  - quick availability due to non-schematic characteristics
  - devs can start coding against NoSQL due to REST / HTTP based API
- Minimalistic Tools requirement
  - Browser UI serves the purpose
- Sharding & Replication
  - Horizontal partitioning & fan-out queries
  - Geographic content replication
- Web developer friendliness
- Cross platform / device operation

Azure Table Storage

- Azure Table storage is a service that stores structured NoSQL data in the cloud, providing a key/attribute store with a schemaless design
- Common uses of Table storage include:
  - Storing TBs of structured data capable of serving web scale applications
  - Storing datasets that don't require complex joins, foreign keys, or stored procedures and can be denormalized for fast access
  - Quickly querying data using a clustered index
  - Accessing data using the OData protocol and LINQ queries with WCF Data Service .NET Libraries

- **Table**: A table is a collection of entities. Tables don't enforce a schema on entities, which means a single table can contain entities that have different sets of properties.
- **Entity**: An entity is a set of properties, similar to a database row. An entity in Azure Storage can be up to 1MB in size. An entity in Azure Cosmos DB can be up to 2MB in size.
- **Properties**: A property is a name-value pair.
  - Each entity can include up to 252 properties to store data.
  - Each entity also has three system properties that specify a partition key, a row key, and a timestamp.
  - Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations.
  - An entity's row key is its unique identifier within a partition.



Partition Key Property

- Tables are partitioned to support load balancing across storage nodes.
- A table's entities are organized by partition. A partition is a consecutive range of entities possessing the same partition key value.
- The partition key is a unique identifier for the partition within a given table, specified by the PartitionKey property.
- The partition key forms the first part of an entity's primary key. The partition key may be a string value up to 1 KiB in size.

You must include the PartitionKey property in every insert, update, and delete operation.

RowKey Property

- The second part of the primary key is the row key, specified by the RowKey property.
- The row key is a unique identifier for an entity within a given partition.
- Together the PartitionKey and RowKey uniquely identify every entity within a table.

The row key is a string value that may be up to 1 KiB in size.

You must include the RowKey property in every insert, update, and delete operation.

Timestamp Property

The Timestamp property is a DateTime value that is maintained on the server side to record the time an entity was last modified. The Table service uses the Timestamp property internally to provide optimistic concurrency. The value of Timestamp is a **monotonically increasing** value, meaning that each time the entity is modified, the value of Timestamp increases for that entity. This property should not be set on insert or update operations (the value will be ignored).

Exercise

1. A gift shop wants to start a portal which allows users to view various items present in its store. The items details like name, price, available quantity etc. are shown. Along with that, pictures of item from different angles are also available. There's a high-res image also available for every item. How will you design a data solution for this?
2. A company has training period for employees. It has around 500 training modules that an employee needs to cover over the period of 6 months. A module could be a video, document, e-book or other form or content. The requirement is to track the status of modules covered by employee over the training period. How will you design a data base solution to for this.

NoSQL in Azure – Cosmos DB

- globally distributed, multi-model database service for mission-critical applications
- elastic scaling of throughput and storage worldwide
- single digit millisecond latencies at 99th percentile
- five well defined consistency models
- guaranteed high availability
- comprehensive SLAs
- automatically indexes all data without requiring schema or indexing management
- multi-model service – supports document, key-value, graph and wide column-family data models

Cosmos DB API

Features of Cosmos DB

- Associate unlimited number of regions with Cosmos DB account
- Policy based geo-fencing
- Failover priorities for regions
- Dynamically configurable read and write regions
- Geo-local reads and writes
- Multi-homing
- Well defined consistency models

Consistency Models in Cosmos DB

- Strong
  - Linearizability. Reads are guaranteed to return the most recent version of an item.
- Bounded Staleness
  - Consistent Prefix. Reads lag behind writes by at most k prefixes or t interval
- Session
  - Consistent Prefix. Monotonic reads, monotonic writes, read-your-writes, write-follows-reads
- Consistent Prefix
  - Updates returned are some prefix of all the updates, with no gaps
- Eventual
  - Out of order reads

Strong Consistency

- Strong consistency offers a linearizability guarantee with the reads guaranteed to return the most recent version of an item.
- Strong consistency guarantees that a write is only visible after it is committed durably by the majority quorum of replicas. A write is either synchronously committed durably by both the primary and the quorum of secondaries, or it is aborted. A read is always acknowledged by the majority read quorum, a client can never see an uncommitted or partial write and is always guaranteed to read the latest acknowledged write.
- Azure Cosmos DB accounts that are configured to use strong consistency cannot associate more than one Azure region with their Azure Cosmos DB account.
- The cost of a read operation (in terms of request units consumed) with strong consistency is higher than session and eventual, but the same as bounded staleness.

Bounded staleness Consistency

- Bounded staleness consistency guarantees that the reads may lag behind writes by at most *K* versions or prefixes of an item or *t* time-interval.
- Therefore, when choosing bounded staleness, the "staleness" can be configured in two ways: number of versions *K* of the item by which the reads lag behind the writes, and the time interval *t*
- Bounded staleness offers total global order except within the "staleness window." The monotonic read guarantees exist within a region both inside and outside the "staleness window."
- Bounded staleness provides a stronger consistency guarantee than session, consistent-prefix, or eventual consistency. For globally distributed applications, use bounded staleness for scenarios where you would like to have strong consistency but also want 99.99% availability and low latency.
- Azure Cosmos DB accounts that are configured with bounded staleness consistency can associate any number of Azure regions with their Azure Cosmos DB account.
- The cost of a read operation (in terms of RUs consumed) with bounded staleness is higher than session and eventual consistency, but the same as strong consistency.

Session Consistency

- Unlike the global consistency models offered by strong and bounded staleness consistency levels, session consistency is scoped to a client session.
- Session consistency is ideal for all scenarios where a device or user session is involved since it guarantees monotonic reads, monotonic writes, and read your own writes (RYW) guarantees.
- Session consistency provides predictable consistency for a session, and maximum read throughput while offering the lowest latency writes and reads.
- Azure Cosmos DB accounts that are configured with session consistency can associate any number of Azure regions with their Azure Cosmos DB account.
- The cost of a read operation (in terms of RUs consumed) with session consistency level is less than strong and bounded staleness, but more than eventual consistency.

Consistent Prefix Consistency

- Consistent prefix guarantees that in absence of any further writes, the replicas within the group eventually converge.
- Consistent prefix guarantees that reads never see out of order writes. If writes were performed in the order A, B, C, then a client sees either A, A,B, or A,B,C, but never out of order like A,C or B,A,C.
- Azure Cosmos DB accounts that are configured with consistent prefix consistency can associate any number of Azure regions with their Azure Cosmos DB account.

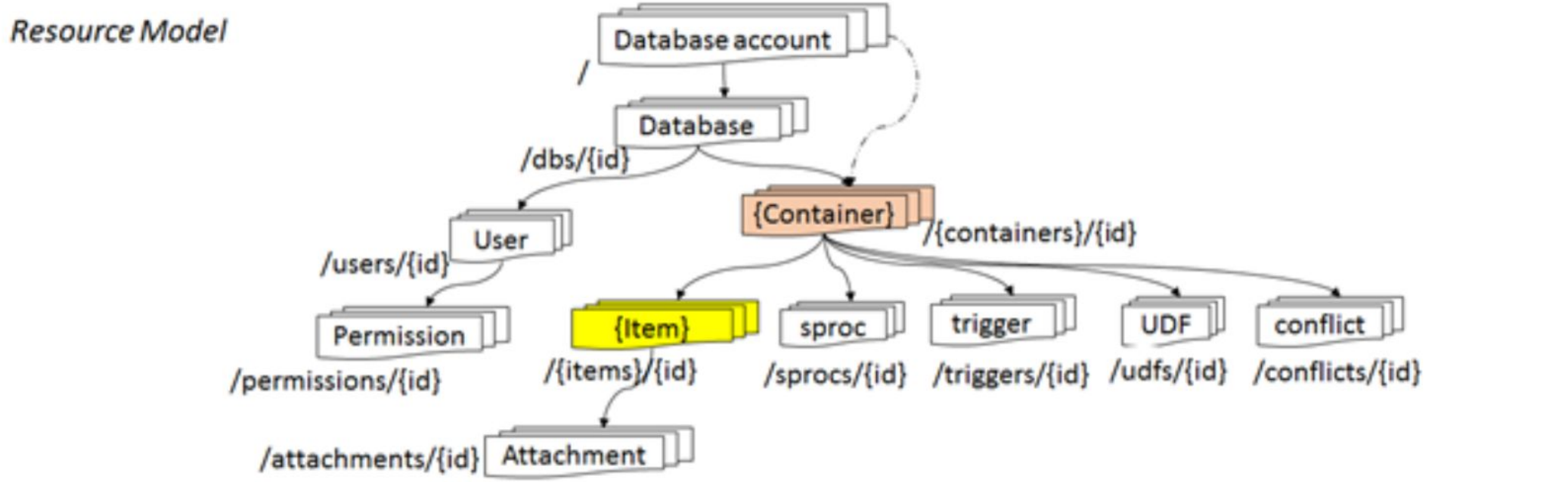
Eventual Consistency

- Eventual consistency guarantees that in absence of any further writes, the replicas within the group eventually converge.
- Eventual consistency is the weakest form of consistency where a client may get the values that are older than the ones it had seen before.
- Eventual consistency provides the weakest read consistency but offers the lowest latency for both reads and writes.
- Azure Cosmos DB accounts that are configured with eventual consistency can associate any number of Azure regions with their Azure Cosmos DB account.
- The cost of a read operation (in terms of RUs consumed) with the eventual consistency level is the lowest of all the Azure Cosmos DB consistency levels.

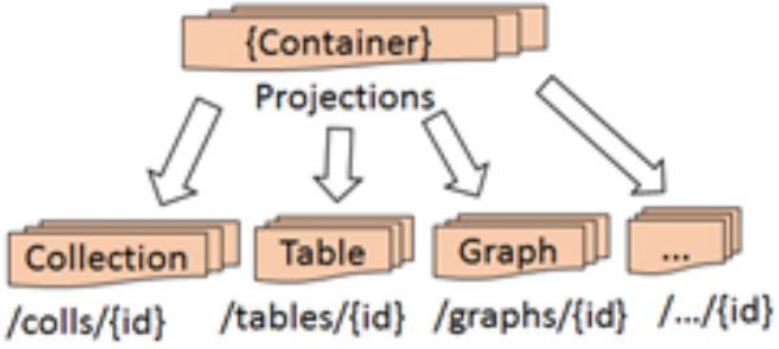
A technical overview of Azure Cosmos DB

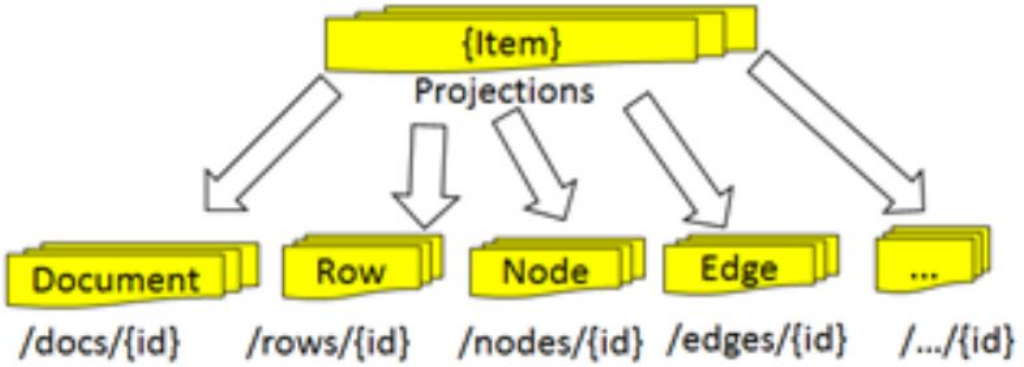
- The core type system of Azure Cosmos DB's database engine is atom-record-sequence (ARS) based
  - Atoms consist of a small set of primitive types e.g. string, bool, number etc.
  - Records are structs
  - Sequences are arrays consisting of atoms, records or sequences
- The database engine of Azure Cosmos DB
  - is capable of efficiently translating and projecting the data models onto the ARS based data model
  - The core data model is natively accessible from dynamically typed programming languages and can be exposed as-is using JSON or other similar representations.

Resource Model and API Projections



Depending on the API, container and item resources are projected as specialized resource types

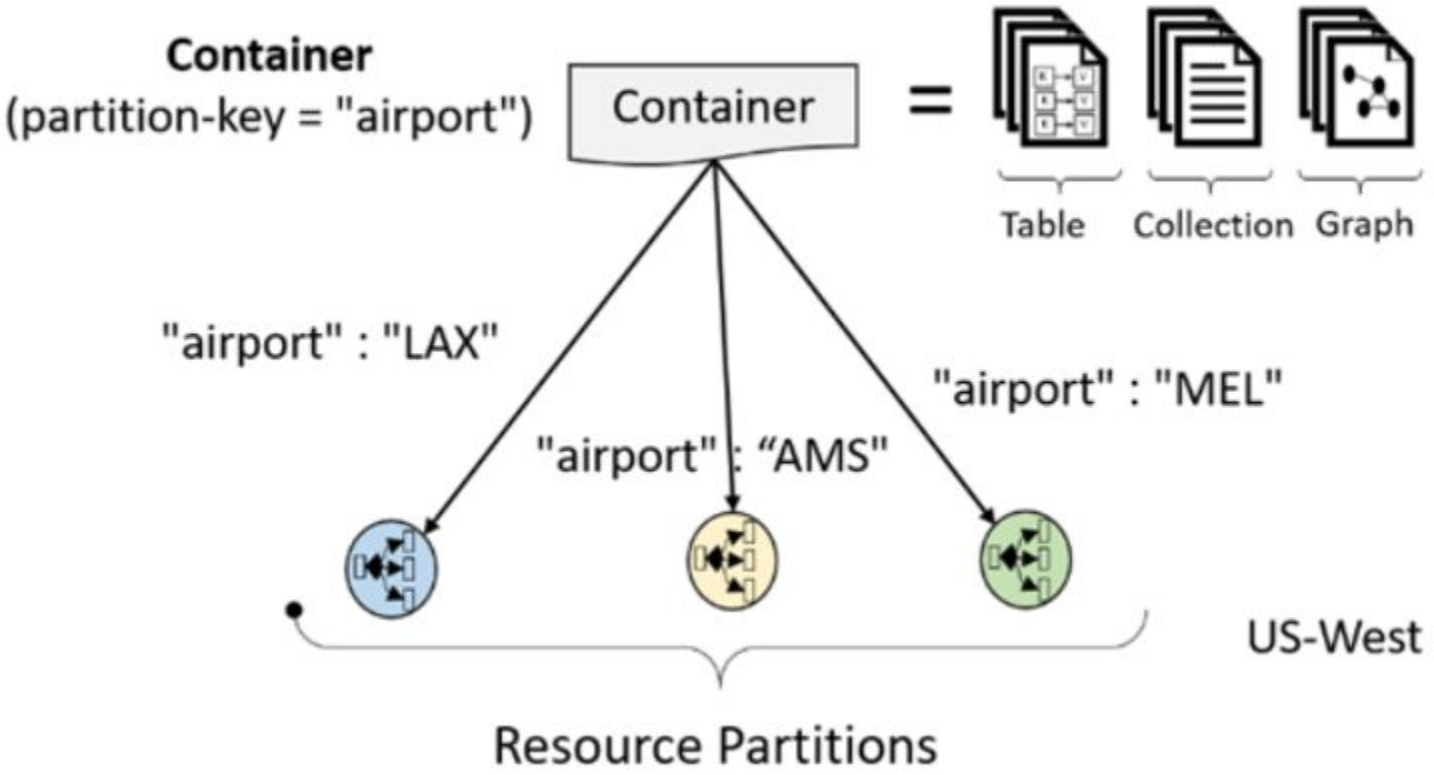




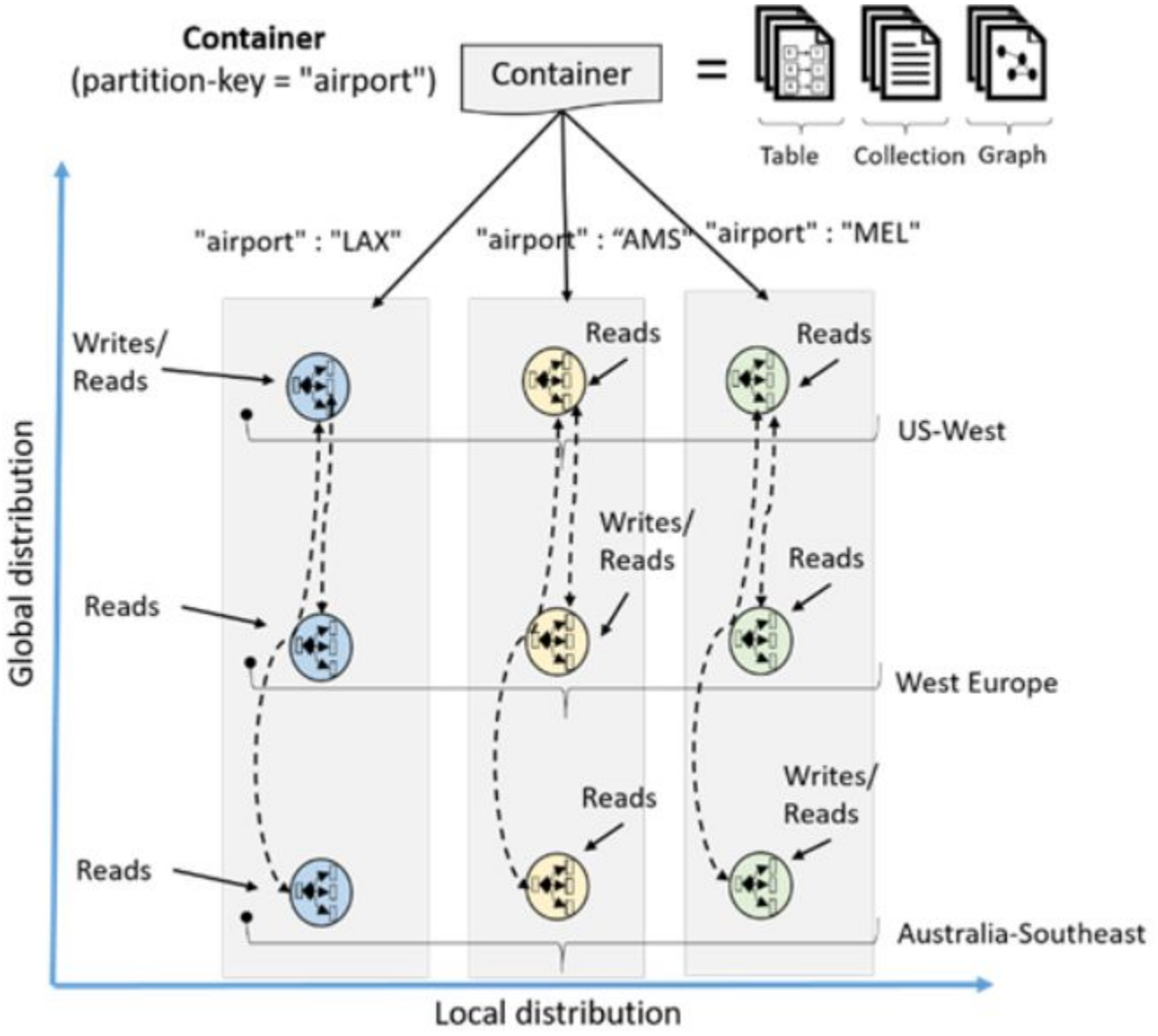
- The overall *resource model* of an application using Azure Cosmos DB is a hierarchical overlay of the resources rooted under the database account, and can be navigated using hyperlinks.
- With the exception of the *item* resource - which is used to represent arbitrary user defined content, all other resources have a system-defined schema.
- The content model of the item resource is based on atom-record-sequence (ARS) described earlier.
- Both, container and item resources are further *projected* as reified resource types for a specific type of API interface

API	Container is projected as ...	Item is projected as ...
DocumentDB SQL	Collection	Document
MongoDB	Collection	Document
Azure Table Storage	Table	Item
Gremlin	Graph	Node and Edge

- Horizontal Partitioning
  - All the data within an Azure Cosmos DB *container* (e.g. collection, table, graph etc.) is horizontally partitioned and transparently managed by *resource partitions*



- Global distribution
  - a customer's resources are distributed along two dimensions: within a given region, all resources are horizontally partitioned using resource partitions (*local distribution*).
  - Each resource partition is also replicated across geographical regions (*global distribution*).
- Currency
  - RUs (Request Units)
  - RUs/sec or RUs/minute



AWS NoSQL - DynamoDB

- DynamoDB organizes data into tables consisting of items.
- Each item in a DynamoDB table can define an arbitrary set of attributes, but all items in the table must define a primary key that uniquely identifies the item.
- This key must contain an attribute known as the “hash key” and optionally an attribute called the “range key.”

Cloud App Services (PaaS)

Azure – App Service

- PaaS service for hosting web applications, REST APIs and mobile back-ends
- Web Apps adds cloud features to application, such as security, load balancing, autoscaling, and automated management.
- Take advantage of its DevOps capabilities, such as continuous deployment from VSTS, GitHub, Docker Hub, and other sources, package management, staging environments, custom domain, and SSL certificates.
- With App Service, pay for the Azure compute resources you use. The compute resources you use is determined by the *App Service plan* that you run your Web Apps on

App Service Plan

- In App Service, an app runs in an *App Service plan*.
- An App Service plan defines a set of compute resources for a web app to run
  - One or more apps can be configured to run on the same computing resources (or in the same App Service plan).

- Each App Service plan defines
  - Region (West US, East US, etc.)
  - Number of VM instances
  - Size of VM instances (Small, Medium, Large)
  - Pricing tier (Free, Shared, Basic, Standard, Premium, Premium V2, Isolated, Consumption)

- Running the app
  - In the **Free** and **Shared** tiers, an app receives CPU minutes on a shared VM instance and cannot scale out.

- Scaling the app
  - the App Service plan is the scale unit of the App Service apps
  - If the plan is configured to run five VM instances, then all apps in the plan run on all five instances
  - If the plan is configured for autoscaling, then all apps in the plan are scaled out together based on the autoscale settings