Cloud Networking

TEKsystems Global Services

WORKFORCE DEVELOPMENT

PARTICIPANT GUIDE

Content Usage Parameters

Content refers to material including instructor guides, student guides, lab guides, lab or hands-on activities, computer programs, etc. designed for use in a training program

1

Content is subject to copyright protection

2

Content may only be leveraged by students enrolled in the training program

3

Students agree not to reproduce, make derivative works of, distribute, publicly perform and publicly display content in any form or medium outside of the training program

4

Content is intended as reference material only to supplement the instructor-led training

AZURE DNS & CONTENT DELIVERY NETWORK (CDN)

OBJECTIVES

Manage Public DNS Zones: Host and administer public DNS records for your domains in Azure.

Implement Azure Traffic Manager: Configure health checks, failover, weighted, and geo-based routing policies.

Utilize Azure Private DNS Zones: Set up private DNS for internal VNet resolution.

Transfer Existing Domains: Understand options for migrating domain registration to Azure.

Configure Azure CDN: Implement CDN for caching and accelerating content delivery.

Secure CDN Content: Apply signed URLs and cookies for content access control.

Troubleshoot DNS & CDN: Identify common issues and resolution strategies.

DNS RECORD TYPES OVERVIEW

Managed DNS Service

Global DNS Network

Record Sets

Common Record Types

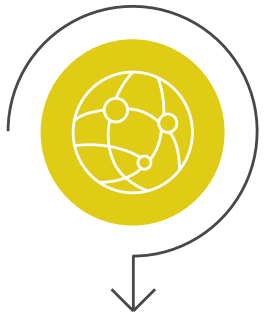
DNS RECORD TYPES OVERVIEW

A (Address) Record: Maps a hostname to an IPv4 address (e.g., www.example.com to 203.0.113.1).
AAAA (IPv6 Address) Record: Maps a hostname to an IPv6 address.
CNAME (Canonical Name) Record: Creates an alias for another hostname (e.g., blog.example.com to example.azurewebsites.net).
MX (Mail Exchange) Record: Specifies mail servers responsible for receiving email for a domain.
TXT (Text) Record: Stores text information about a domain, often used for SPF, DKIM, or domain verification.
NS (Name Server) Record: Delegates a DNS zone or a subdomain to a set of name servers.
SRV (Service) Record: Locates services (e.g., SIP, XMPP) by specifying hostname and port.

LAB 7: CREATING PUBLIC DNS ZONES

Creating a Public DNS Zone: Steps to provision a new DNS Zone resource in Azure.
Adding A Records: Demonstrating how to create A records for hostnames like www or @ (root domain).
Adding CNAME Records: Creating CNAME records to alias a hostname to an Azure service's FQDN.
Configuring MX Records: Setting up Mail Exchange records for email delivery.
Delegating Domain to Azure: Overview of updating domain registrar's name servers to Azure DNS.
Verifying DNS Propagation: Using tools to check global DNS resolution.

DNS BASED GLOBAL LOAD BALANCING



Global DNS

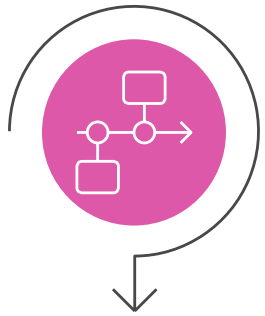
Distributes traffic based on DNS responses.

No Proxy

Operates at the DNS level, not as a proxy.

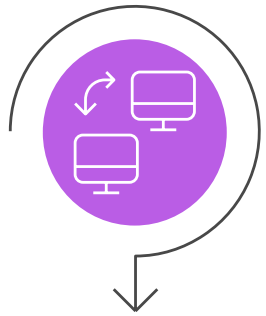
Health Monitoring

Monitors endpoint health for availability assurance.



Routing Methods

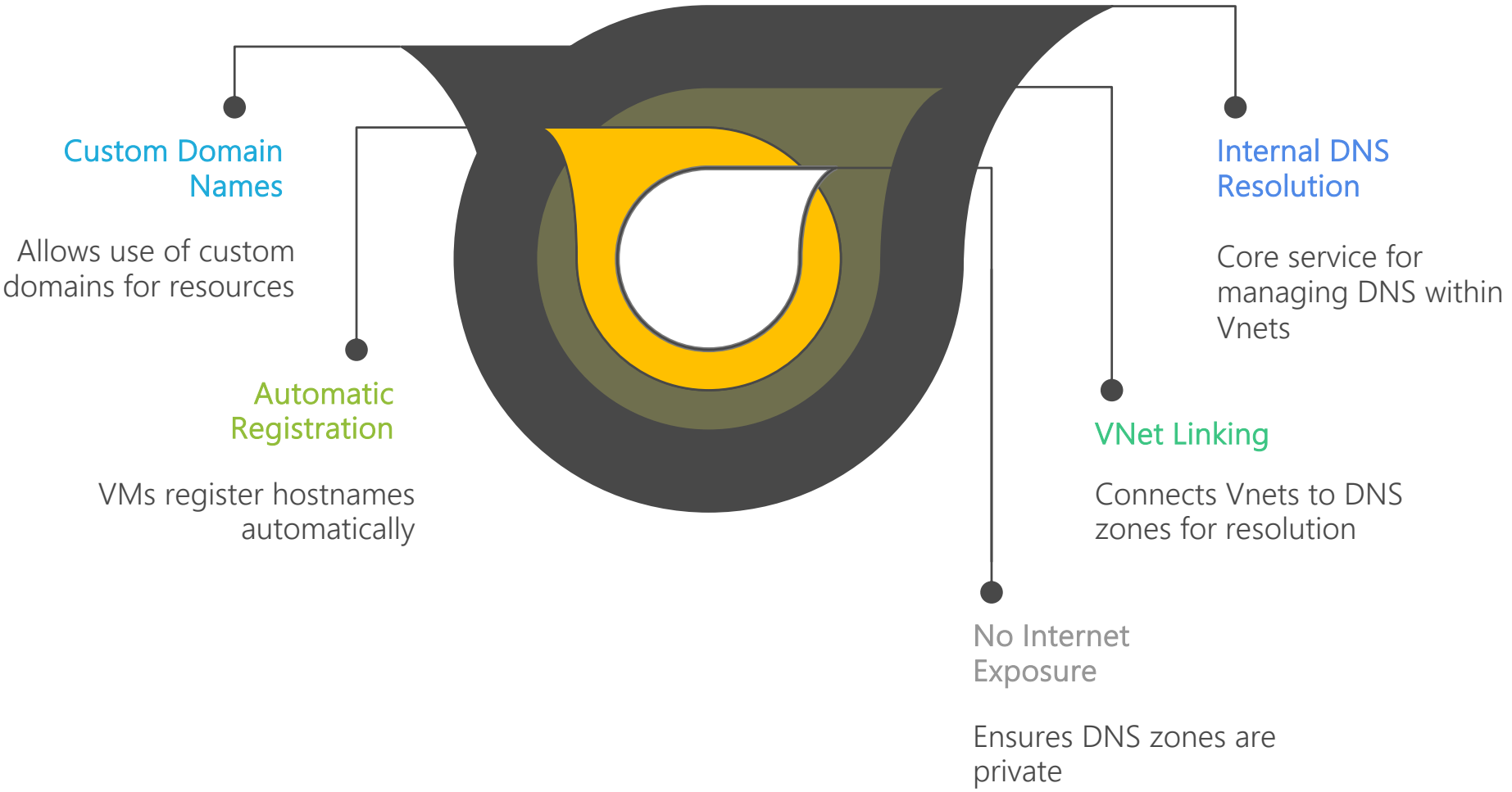
Offers methods like Priority, Weighted, and Geographic.



High Availability

Ideal for multi-region active-active deployments.

PRIVATE DNS ZONES



Custom Domain Names

Allows use of custom domains for resources

Automatic Registration

VMs register hostnames automatically

Internal DNS Resolution

Core service for managing DNS within Vnets

VNet Linking

Connects Vnets to DNS zones for resolution

No Internet Exposure

Ensures DNS zones are private

PRIVATE DNS ZONE COMPONENTS

- **Private DNS Zone Resource:** The container for private DNS records, scoped to a resource group.
- **Virtual Network Links:** Associations between the Private DNS Zone and one or more VNets.
- **Registration VNet Link:** A VNet link where automatic VM hostname registration is enabled.
- **Resolution VNet Link:** A VNet link where resources can query the private DNS zone for name resolution.
- **Common Record Types:** Supports A, AAAA, CNAME, MX, PTR, SRV, TXT records, similar to public zones.

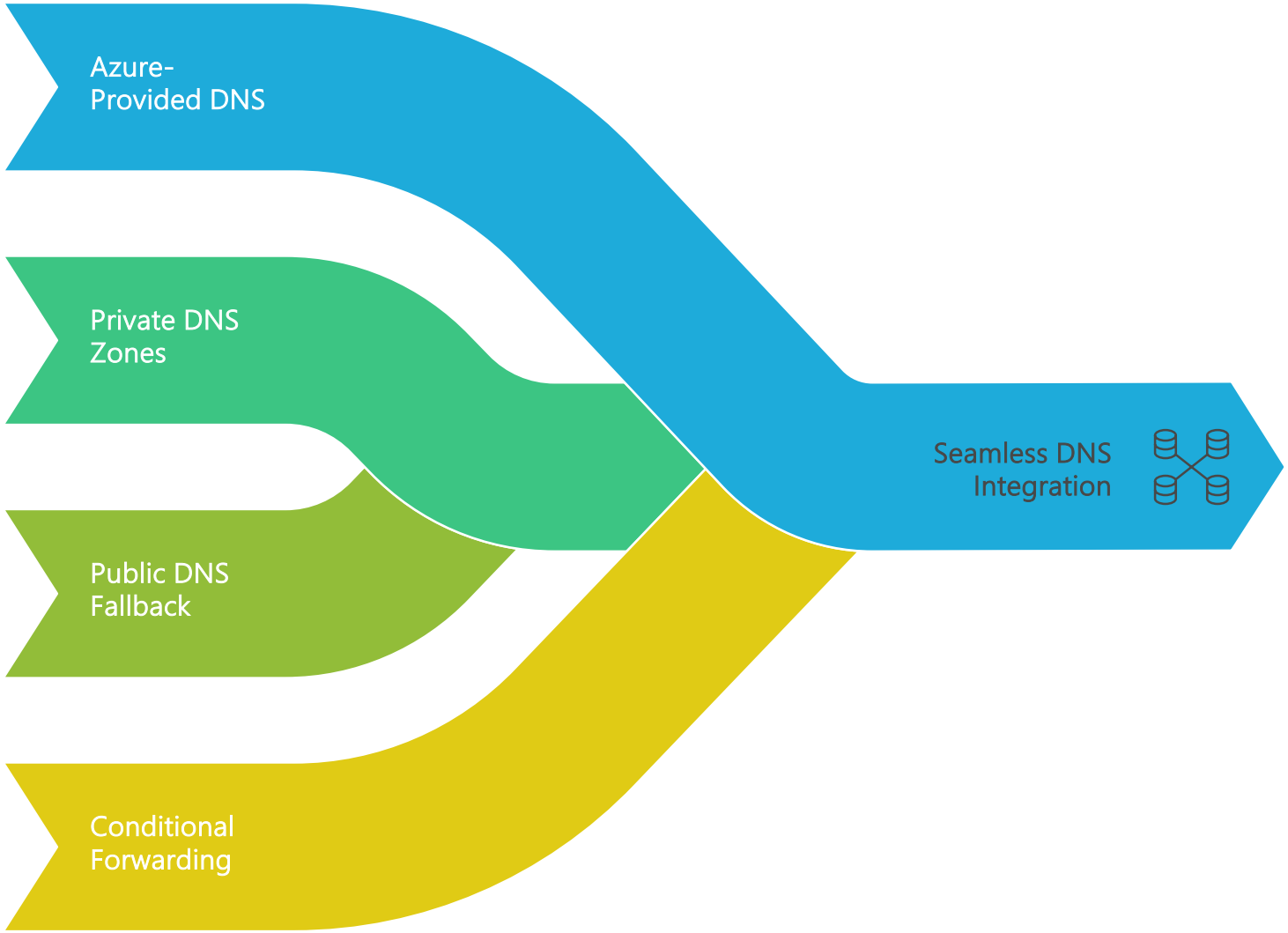
DNS RESOLUTION FLOW WITH PRIVATE DNS ZONES

1 Default DNS resolvers provided by Azure for VMs.

2 First-check DNS zones linked to Virtual Networks.

3 Azure Public DNS or custom servers as fallback.

4 Integrates with on-premises DNS using conditional forwarders.



TRANSFERRING EXISTING DOMAIN TO THE CLOUD

- **Option 1: Change Name Servers (Delegation):** Keep domain registrar, point name servers to Azure DNS.
  - **Pros:** Familiar registrar interface, quick setup of DNS in Azure.
  - **Cons:** DNS management split between registrar (for NS) and Azure (for records).

- **Option 2: Transfer Domain Registration:** Move entire domain registration to a cloud-friendly registrar (e.g., Azure Domain Services, GoDaddy, Namecheap).
  - **Pros:** Consolidated management, potentially simpler billing.
  - **Cons:** Involves domain transfer process, may require unlocking domain.

- **Option 3: Use Azure DNS for Records, Third-Party for NS:** Keep existing NS servers, replicate Azure DNS records to them.
  - **Pros:** No registrar change, no NS change.
  - **Cons:** Manual synchronization, higher risk of configuration drift.

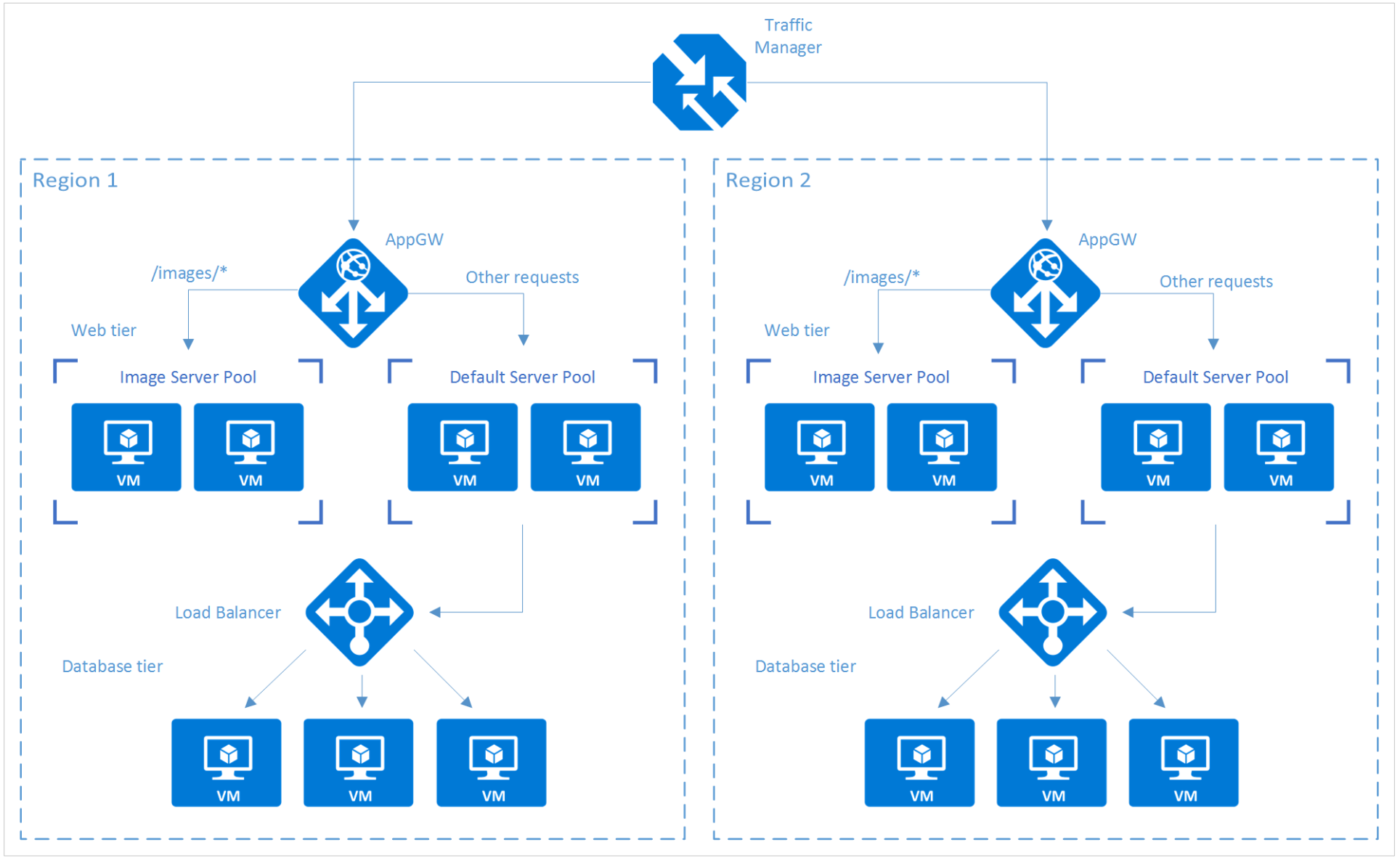
LAB 8: IMPLEMENT PRIVATE DNS ZONES

Creating a Private DNS Zone: Steps to provision a new Private DNS Zone resource.
Linking to a Virtual Network: Associating the private zone with your VNet for resolution.
Enabling Auto-Registration (Optional): Demonstrating how to enable automatic hostname registration for a linked VNet.
Adding Custom DNS Records: Manually creating A records for internal resources (e.g., private load balancer).
Testing Internal Resolution: From a VM in the linked VNet, perform a DNS lookup for a private hostname.
Confirming No External Resolution: Attempting to resolve the private hostname from outside the VNet (should fail).

TRAFFIC MANAGER ROUTING METHODS

- **Priority (Failover):** Routes all traffic to a primary endpoint; if it fails, traffic switches to the next priority endpoint.
- **Weighted:** Distributes traffic across endpoints based on pre-defined weights. Ideal for A/B testing or gradual rollout.
- **Performance (Latency):** Routes traffic to the endpoint with the lowest network latency for the client.
- **Geographic:** Routes users to specific endpoints based on their geographic location.
- **Multivalue:** Returns multiple IP addresses in a single DNS response. Client selects one (often random).
- **Subnet:** Routes users based on the subnet range of their originating IP address.

TRAFFIC MANAGER

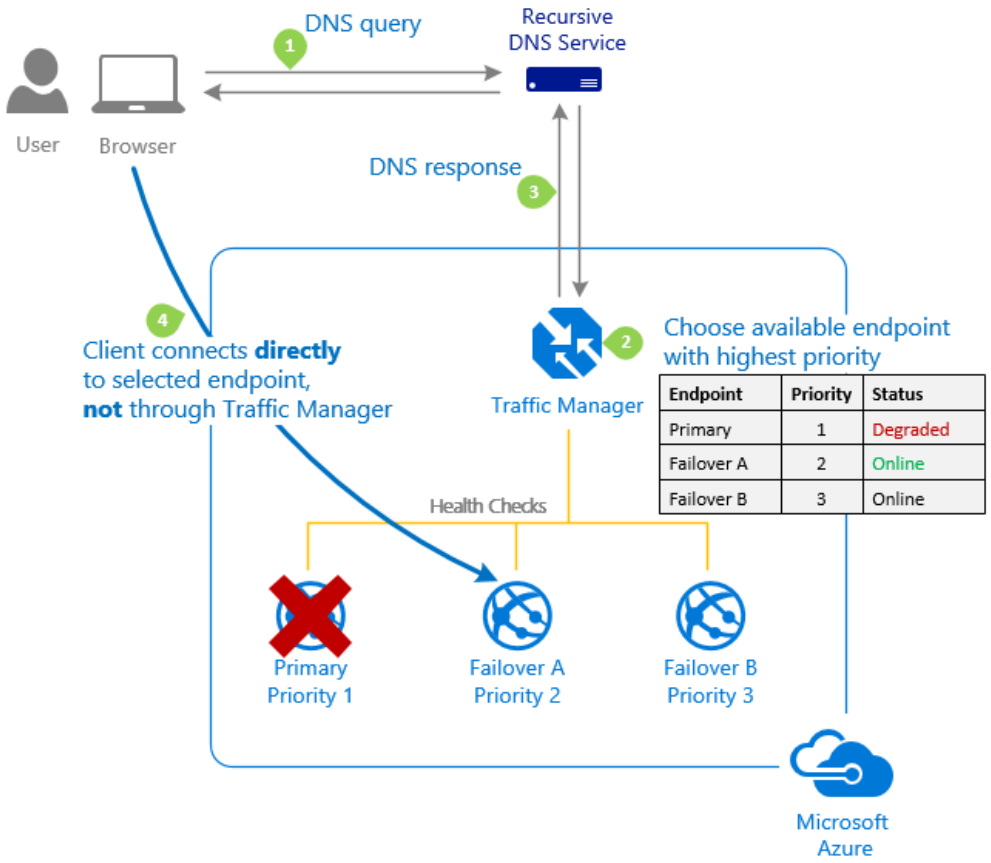


TRAFFIC MANAGER

- DNS-based load balancer
- Distribute traffic across global regions
- Six traffic-routing methods
- Endpoint health monitoring and automatic failover

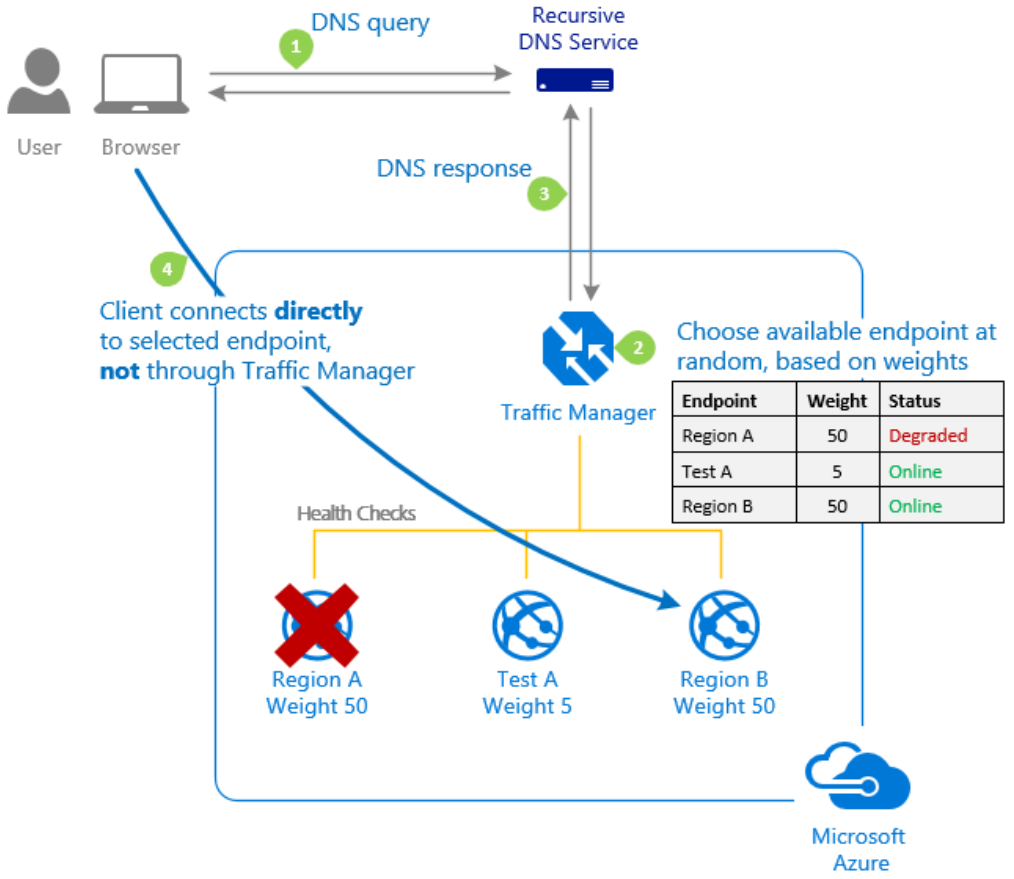
TRAFFIC MANAGER

- Priority method



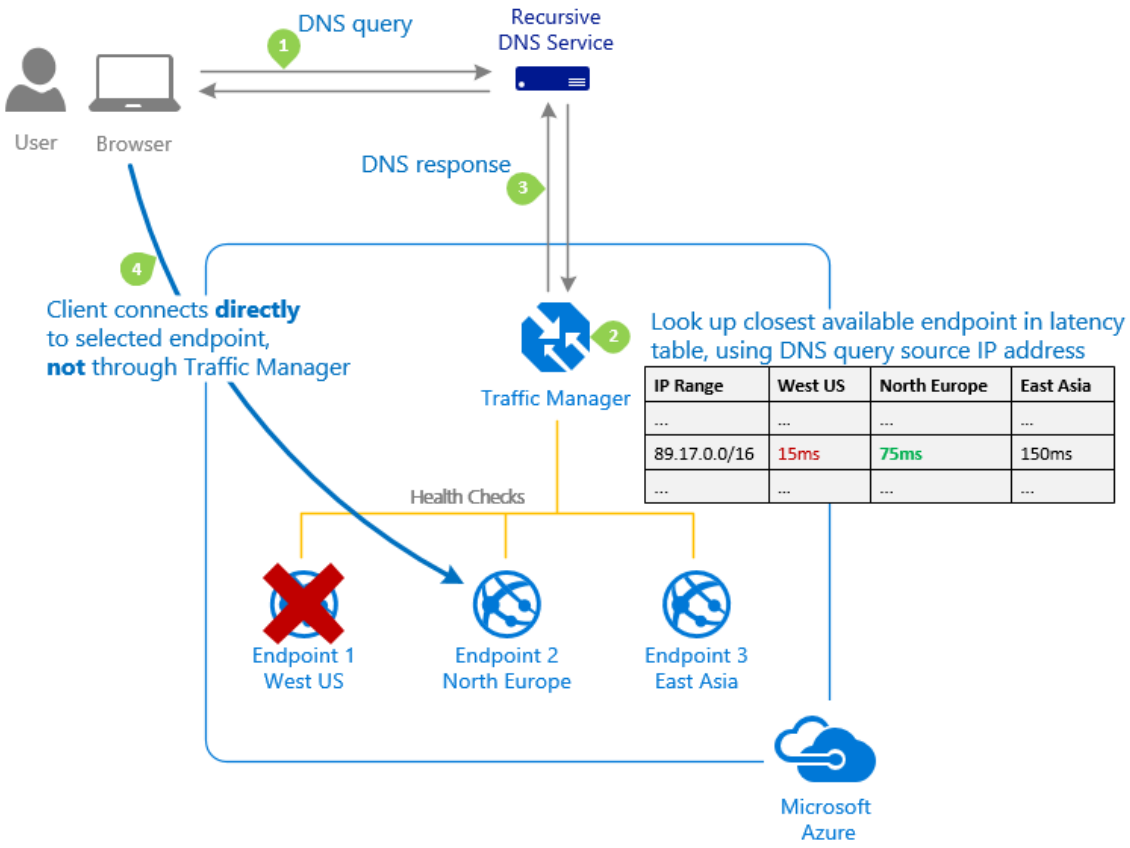
TRAFFIC MANAGER

- Weighted method



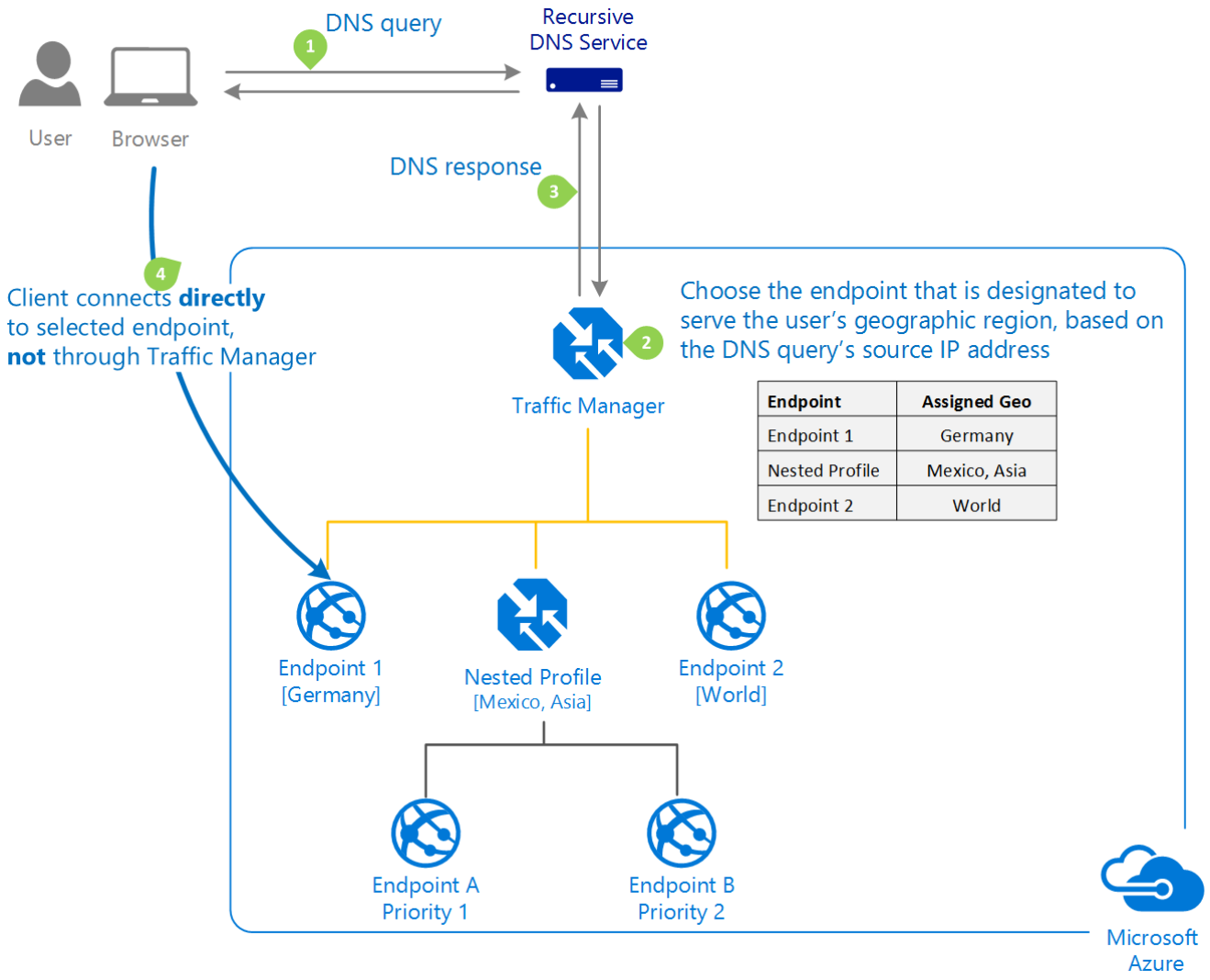
TRAFFIC MANAGER

- Performance method

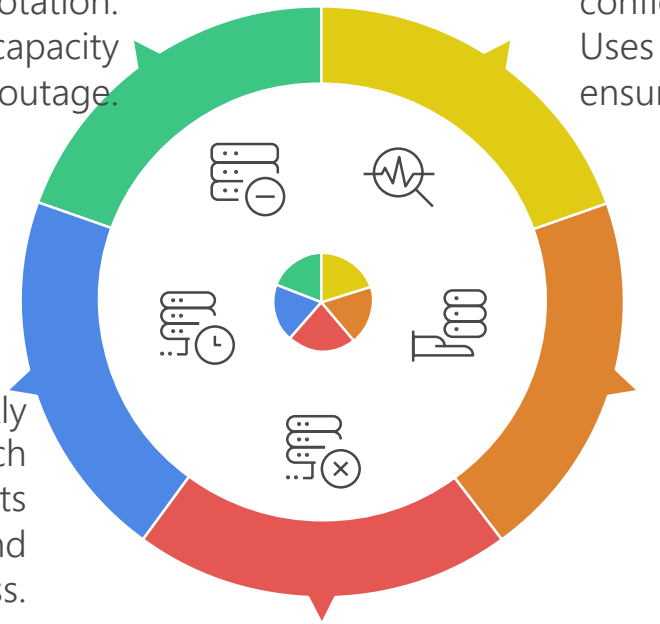


TRAFFIC MANAGER

- Geographic method



HEALTH CHECKS & FAILOVER RECORDS



Endpoint Monitoring

Checks availability of configured endpoints. Uses health probes to ensure functionality.

Probe Protocols

Supports HTTP, HTTPS, and TCP probes. Targets specified ports and paths for monitoring.

Automatic Failover

Removes unhealthy endpoints from DNS rotation. Ensures traffic is directed to healthy resources.

DNS TTL

Determines how quickly clients switch endpoints. Affects failover speed and responsiveness.

Failback

Re-adds recovered endpoints to rotation. Restores full capacity after an outage.

WEIGHTED RECORDS FOR LOAD DISTRIBUTION

Configurable Weights: Assign a numerical weight to each endpoint (e.g., 1-1000).

Proportional Distribution: Traffic is distributed proportionally based on the assigned weights.

Dynamic Adjustment: Weights can be adjusted dynamically to influence traffic flow (e.g., for scale-out or maintenance).

Use Cases:

A/B Testing: Send a small percentage of traffic to a new version.

Gradual Rollouts: Incrementally shift traffic to a new deployment.

Capacity Management: Direct more traffic to regions with more available capacity.

GEOLOCATION ROUTING

- **User Location-Based Routing:** Routes user requests to endpoints based on the geographic location of the DNS query origin.
- **Data Sovereignty:** Helps meet data residency requirements by keeping user data in a specific region.
- **Content Localization:** Deliver localized content or language-specific application versions.
- **Default Fallback:** Configure a default endpoint or profile to handle requests from unspecified geographic regions.
- **Nested Profiles:** Can combine with other routing methods by nesting Traffic Manager profiles.

LAB 9: CONFIGURE AZURE TRAFFIC MANAGER

In this exercise, you will set up Azure Traffic Manager to route traffic to endpoints based on performance (lowest latency), add health checks, and simulate failover. This lab covers global DNS load balancing.

CLOUD CDN

Content Delivery Acceleration: Delivers web content (images, videos, files) to users from geographically closest edge servers.

Global POP Network: Leverages a vast network of Points of Presence (POPs) located worldwide.

Reduced Latency: Content served from nearby POPs, minimizing distance to user, improving performance.

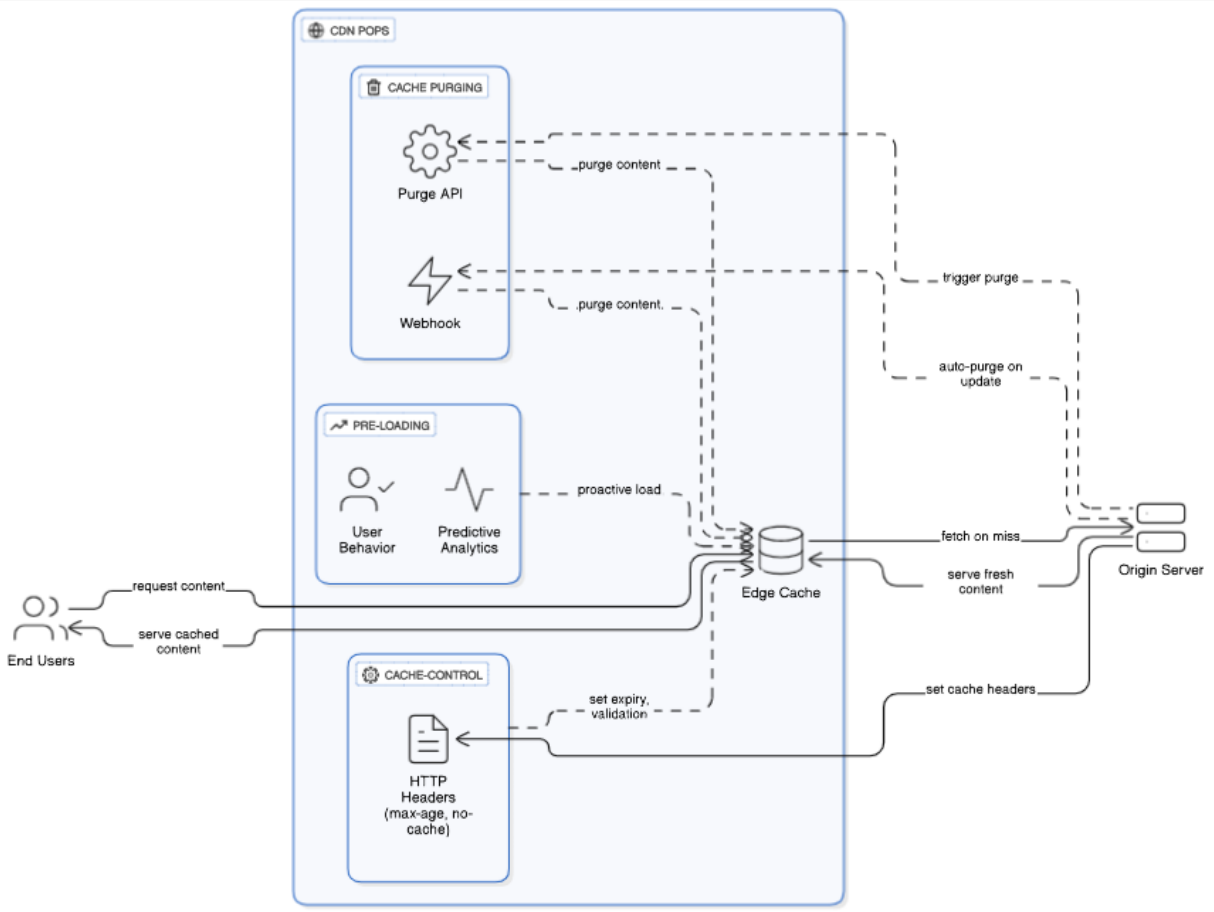
Reduced Origin Load: Offloads traffic from your origin servers, reducing their processing load and bandwidth consumption.

Improved User Experience: Faster loading times, smoother streaming, and higher satisfaction.

AZURE CDN FEATURES

- **Caching Rules:** Fine-tune how content is cached at the POPs (e.g., cache-control headers, query string caching).
- **Compression:** Supports Gzip/Brotli compression to reduce file size and accelerate delivery.
- **Geo-filtering:** Restrict access to your content based on country or region.
- **Dynamic Site Acceleration (DSA):** Optimizes routing and network for dynamic content, not just static.
- **Rule Engine (Standard/Premium):** Advanced rule processing for custom caching, header modification, and redirects.
- **Integrated with Azure Services:** Seamlessly integrates with Azure Storage, Web Apps, Cloud Services

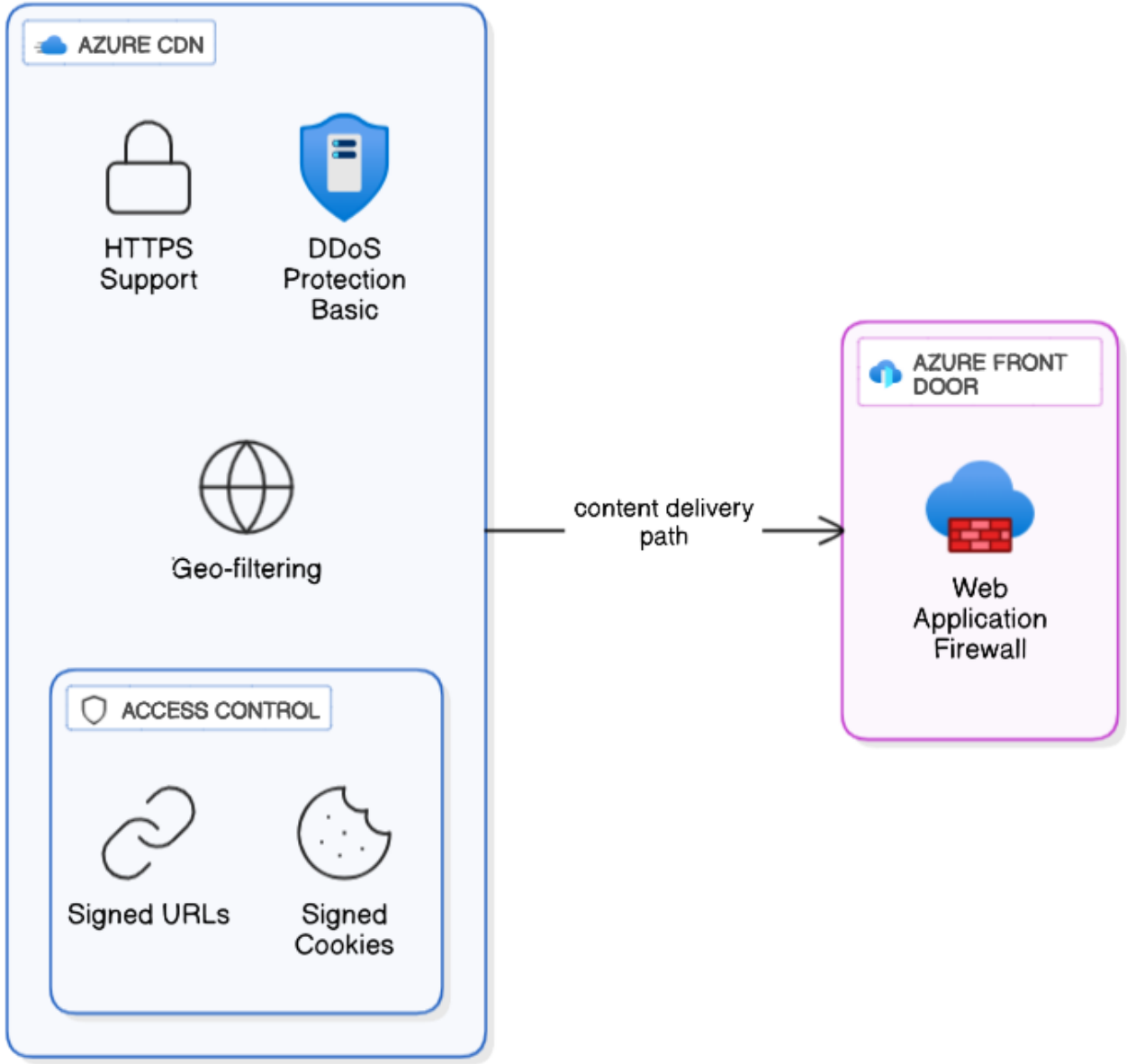
CDN CACHING



- **Edge Caching:** Content is stored at CDN Points of Presence (POPs) closer to users.
- **First Request Slow:** The very first request for an item might be slow as it's fetched from the origin.
- **Subsequent Requests Fast:** Subsequent requests for the same item are served directly from the POP cache.
- **Cache-Control Headers:** Use HTTP Cache-Control headers (e.g., max-age, no-cache) to control caching behavior.
- **Cache Purging:** Manually purge cached content from POPs when origin content is updated.
- **Pre-loading Content:** Proactively load content into POPs to ensure fast first access.

CDN SECURITY

Azure CDN itself is protected by Azure DDoS Protection Basic, offering foundational defense against network attacks. It fully supports HTTPS, ensuring secure content delivery from the POPs to your clients.



SIGNED URLS

Time-Limited Access: Grant temporary, restricted access to private CDN content without exposing it publicly.

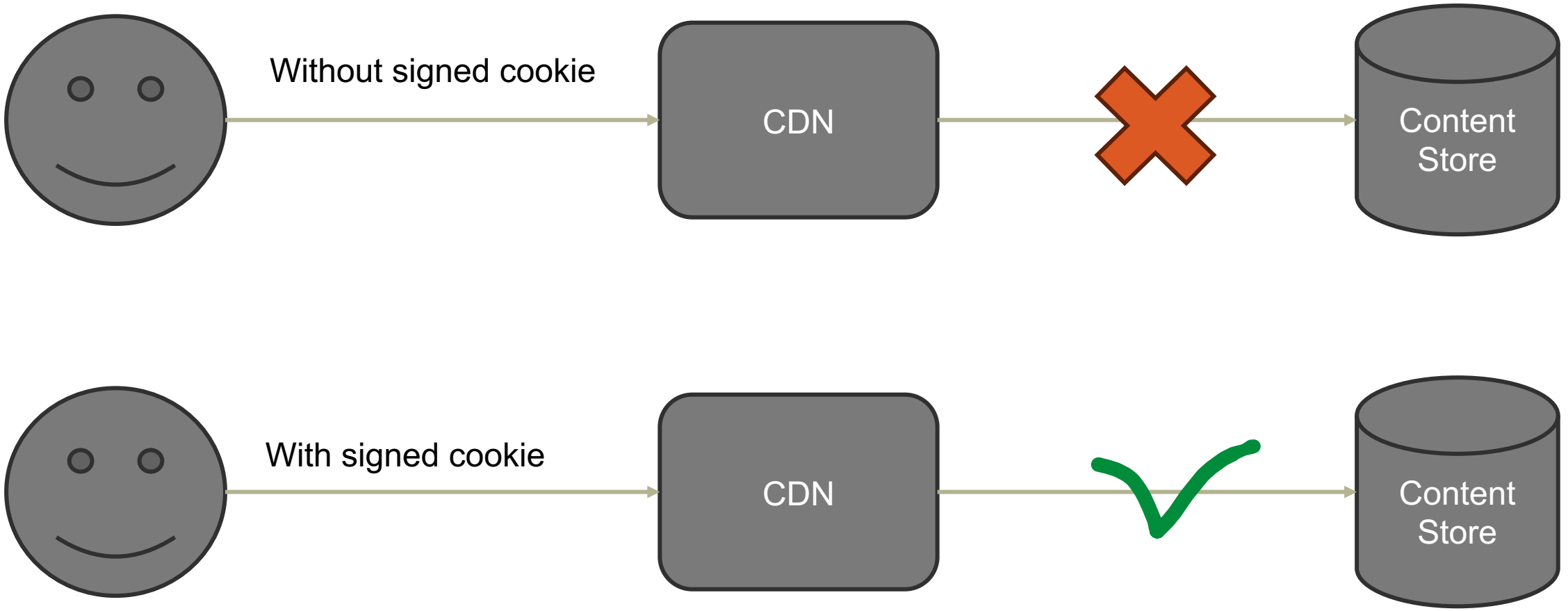
Query String Parameter: A unique signature and expiration time are appended as query string parameters to the URL.

Shared Secret: The signature is generated using a shared secret known only to your application and the CDN.

Use Cases: Delivering premium content, video streaming, or controlled file downloads.

How it Works: Client receives signed URL, CDN validates signature/expiry, serves content if valid

SIGNED COOKIES



TUNING & TROUBLESHOOTING

- **HTTP Headers (Cache-Control):** Correctly configure Cache-Control and Expires headers on your origin.
- **Query String Caching:** Configure CDN's query string caching behavior (ignore, unique, bypass).
- **Origin Optimization:** Ensure your origin server is performant and reliable for first-time fetches.
- **Purging:** Understand when and how to purge content effectively after updates.
- **Monitoring & Analytics:** Use Azure Monitor for CDN performance metrics, hit/miss ratios, and errors.
- **Troubleshooting Steps:**
  - Verify DNS (CNAME points to CDN endpoint).
  - Check origin accessibility.
  - Clear browser cache.
  - Review CDN caching rules.
  - Use CDN diagnostic tools.

LAB 10: IMPLEMENT AZURE CDN

Accelerate content delivery: Set up Azure CDN to cache static files from Blob Storage, configure rules, and test edge caching for improved performance and reduced latency.

POP QUIZ:

You have a new public-facing web application hosted on Azure Virtual Machines behind an Azure Application Gateway. You want to register a custom domain, www.contoso.com, and point it to the Application Gateway's public IP address. Which type of DNS record should you create in your Azure Public DNS Zone to achieve this, given that the Application Gateway's frontend is a Public IP?

A. MX record
B. CNAME record
C. A record
D. TXT record

POP QUIZ:

You have a new public-facing web application hosted on Azure Virtual Machines behind an Azure Application Gateway. You want to register a custom domain, www.contoso.com, and point it to the Application Gateway's public IP address. Which type of DNS record should you create in your Azure Public DNS Zone to achieve this, given that the Application Gateway's frontend is a Public IP?

A. MX record
B. CNAME record
C. A record
D. TXT record

POP QUIZ:

Your global application is deployed across three Azure regions: East US, West Europe, and Southeast Asia. You want to route user traffic to the geographically closest available backend endpoint to minimize latency. If an endpoint in a region becomes unhealthy, users in that region should automatically be routed to the next closest healthy region. Which Azure Traffic Manager routing method should you configure?

A. Weighted
B. Performance
C. Priority
D. Geographic

POP QUIZ:

Your global application is deployed across three Azure regions: East US, West Europe, and Southeast Asia. You want to route user traffic to the geographically closest available backend endpoint to minimize latency. If an endpoint in a region becomes unhealthy, users in that region should automatically be routed to the next closest healthy region. Which Azure Traffic Manager routing method should you configure?

A. Weighted
B. Performance
C. Priority
D. Geographic

POP QUIZ:

You have an Azure Virtual Network (VNet) where several Virtual Machines (VMs) need to resolve hostnames for internal resources, such as internal-api.privatedomain.com, which are only accessible within your VNet. You do not want to deploy and manage your own DNS servers within the VNet. Which Azure service should you use to enable this private name resolution?
A. Azure Public DNS Zones
B. Azure DNS Resolver (deprecated for this scenario)
C. Custom DNS servers on VMs
D. Azure Private DNS Zones

POP QUIZ:

You have an Azure Virtual Network (VNet) where several Virtual Machines (VMs) need to resolve hostnames for internal resources, such as internal-api.privatedomain.com, which are only accessible within your VNet. You do not want to deploy and manage your own DNS servers within the VNet. Which Azure service should you use to enable this private name resolution?

A. Azure Public DNS Zones
B. Azure DNS Resolver (deprecated for this scenario)
C. Custom DNS servers on VMs
D. Azure Private DNS Zones

POP QUIZ:

You are hosting high-resolution images for your e-commerce website in Azure Blob Storage. You want to improve page load times for users globally by caching these images closer to them and reducing the load on your storage account. Which Azure service is ideal for this purpose?

A. Azure Front Door
B. Azure ExpressRoute
C. Azure Content Delivery Network (CDN)
D. Azure Load Balancer (Standard)

POP QUIZ:

You are hosting high-resolution images for your e-commerce website in Azure Blob Storage. You want to improve page load times for users globally by caching these images closer to them and reducing the load on your storage account. Which Azure service is ideal for this purpose?

A. Azure Front Door
B. Azure ExpressRoute
C. Azure Content Delivery Network (CDN)
D. Azure Load Balancer (Standard)

POP QUIZ:

You are delivering premium video content via Azure CDN, and you need to ensure that only authorized users can access the videos for a limited time after their purchase. Which CDN security feature should you implement?

A. Geo-filtering
B. WAF Integration
C. Signed URLs or Signed Cookies
D. Always-on DDoS Protection

POP QUIZ:

You are delivering premium video content via Azure CDN, and you need to ensure that only authorized users can access the videos for a limited time after their purchase. Which CDN security feature should you implement?

A. Geo-filtering
B. WAF Integration
C. Signed URLs or Signed Cookies
D. Always-on DDoS Protection

COMMON AZURE NETWORK ARCHITECTURES & AUTOMATION

OBJECTIVES

Identify Single Zone Deployment Patterns: Understand basic availability within an Availability Zone.

Design Multi-Zone Deployments: Leverage Availability Zones for intra-regional high availability.

Architect Single Region Deployments: Implement comprehensive HA within a single Azure region.

Plan Multi-Region Deployments: Design for global resilience and disaster recovery.

Explain Network Connectivity Patterns: Understand how components connect in various architectures.

Automate Network Deployment with Terraform: Introduce Infrastructure as Code for Azure networking.

AZURE ARCHITECTURE CENTER

Definition: A central resource hub that provides best practices, architectural guidance, and design patterns for building solutions on Azure.

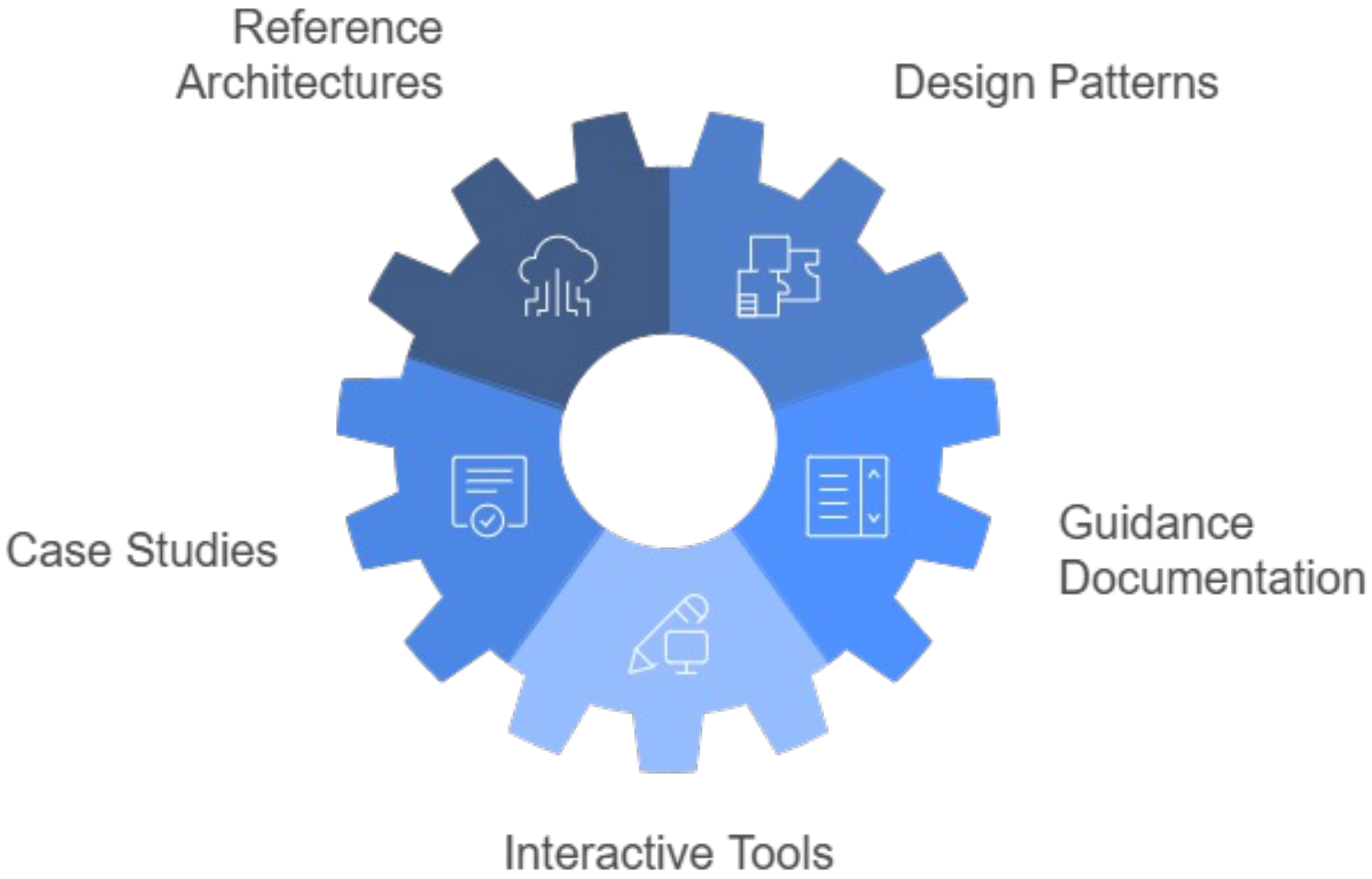
Purpose: Helps organizations design, implement, and optimize cloud architectures to meet strategic business goals.

Content Offered: Includes reference architectures, blueprints, case studies, and design guides.

Target Audience: Designed for both technical teams and leadership to drive informed decision-making.

Continuous Updates: Regularly refreshed to reflect evolving technology trends and industry standards.

KEY COMPONENTS

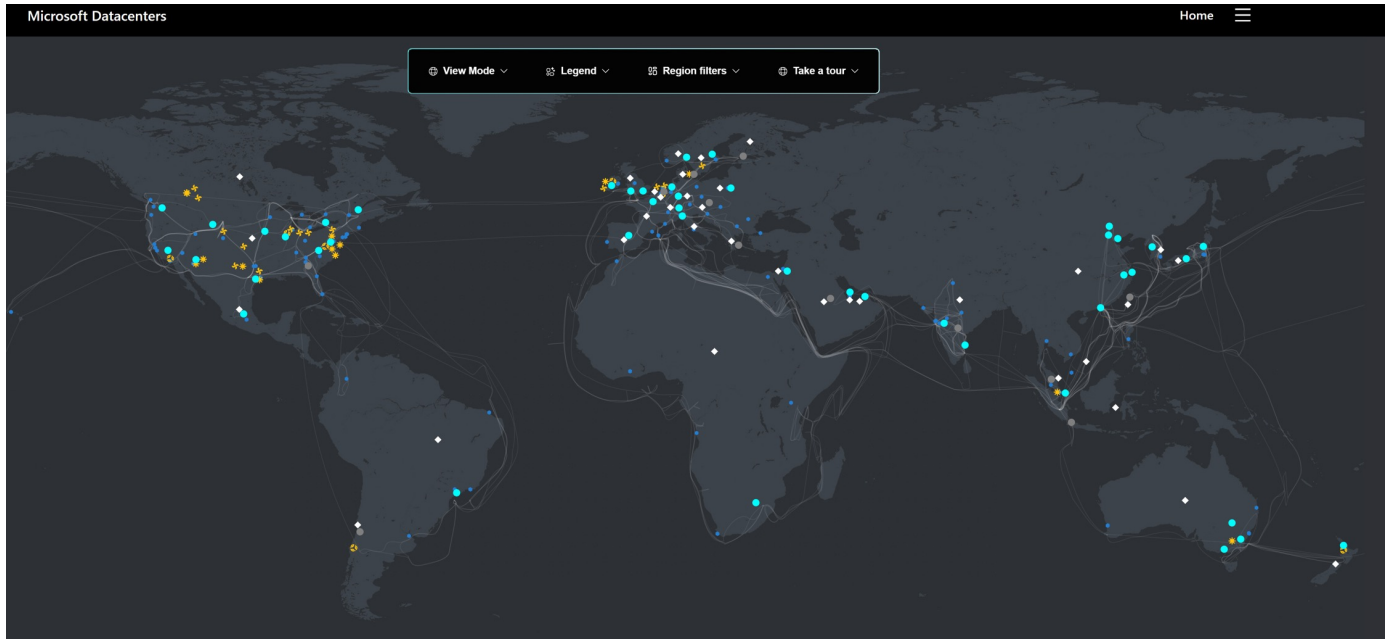


AZURE GLOBAL INFRASTRUCTURE

Global Datacenters:
Azure operates in over 60 regions worldwide, ensuring global reach and low latency for users across different geographies.

Regional Distribution:
Regions are designed to meet local regulatory and compliance needs, providing tailored data residency options.

Availability Zones:
These are physically separate locations within a region that offer high availability and fault tolerance.



Azure global infrastructure experience

Connectivity and Performance:
Azure's extensive network infrastructure supports rapid data transfer and optimized connectivity between regions.

Compliance and Certifications:
Azure's global datacenters meet a wide range of international compliance standards, including GDPR and ISO 27001.

AVAILABILITY SETS VS. AVAILABILITY ZONES

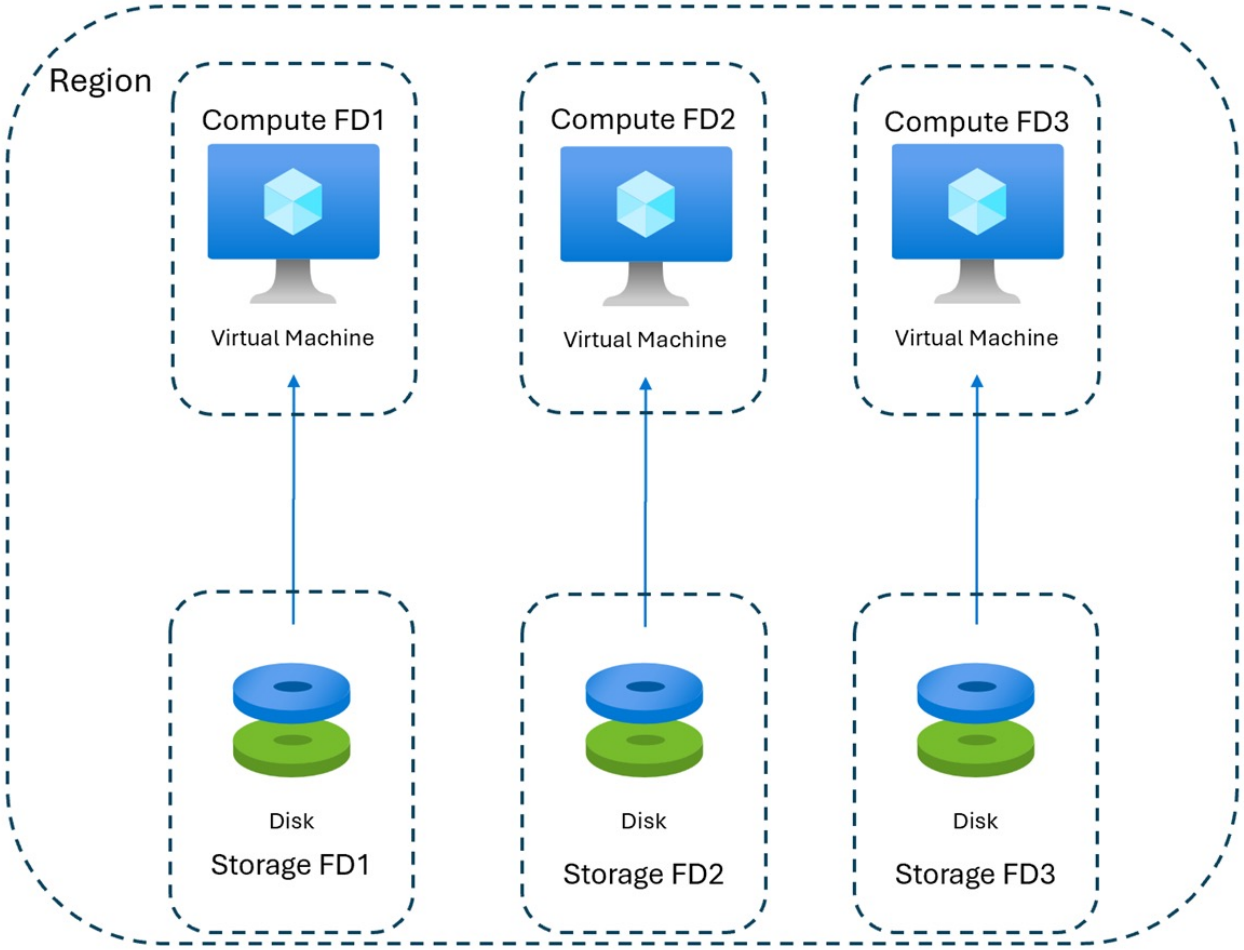
Availability Sets:

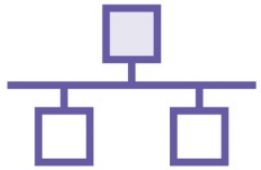
- **Scope:** Regional, but distributes VMs within a single data center.
- **Fault Domains:** Physical separation of hardware (rack, power, network). Max 3.
- **Update Domains:** Logical groups for planned maintenance. Max 20.
- **Protection:** Guards against single point of hardware failure or planned maintenance within a data center.
- **SLA:** 99.95% VM availability (for 2+ VMs in AS).

Availability Zones:

- **Scope:** Regional, but distributes VMs across physically separate data centers within a region.
- **Fault Domains:** Each zone is its own fault domain (power, cooling, networking).
- **Update Domains:** Each zone is its own update domain.
- **Protection:** Guards against entire data center failures within a region.
- **SLA:** 99.99% VM availability (for 2+ VMs across AZs).

AVAILABILITY SETS





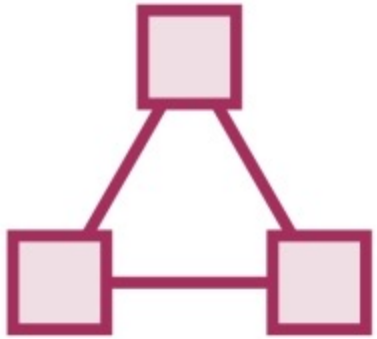
VMs are placed on nodes in a rack

Failures can occur at node and rack levels

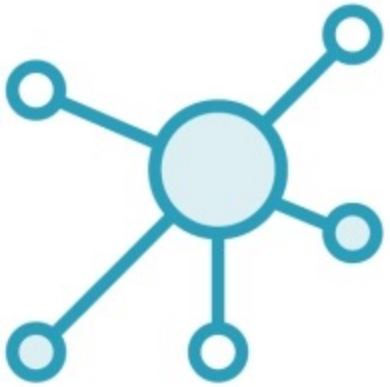
Maintenance is also required on hosts and VMs are **not** live migrated

AVAILABILITY SETS

To ensure availability of services, always deploy minimum 2 instances of any service and place in a unique availability set

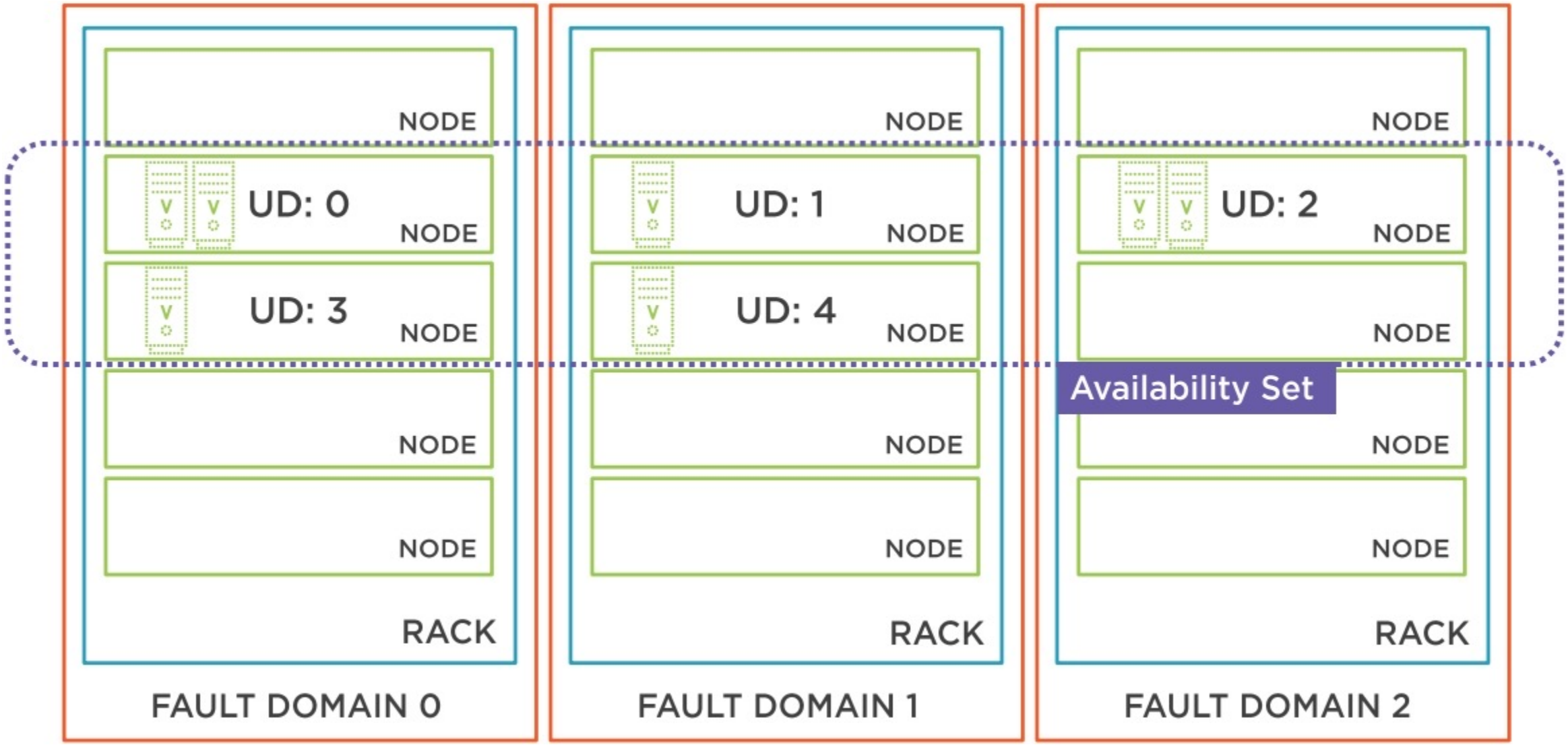


This ensures VMs are spread over three fault domains (racks) and five (by default) update domains



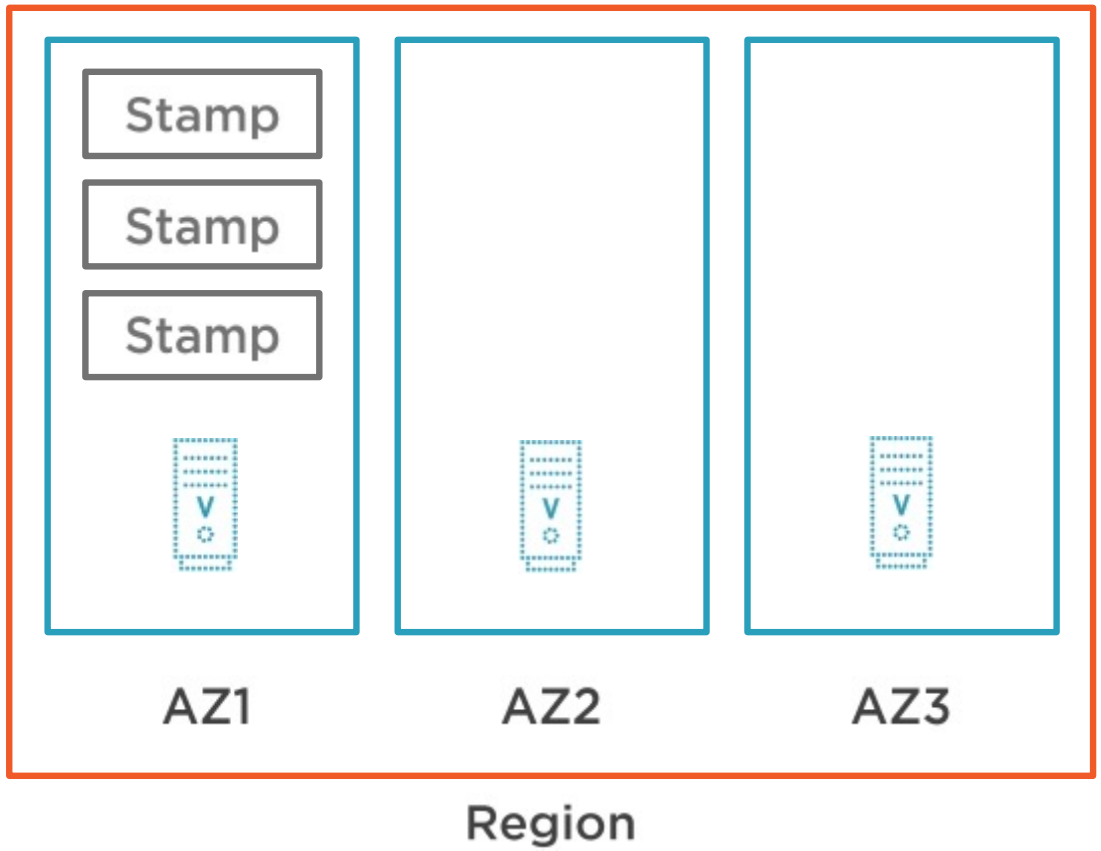
Must have minimum 2 VMs to receive SLA of 99.95%

AVAILABILITY SETS



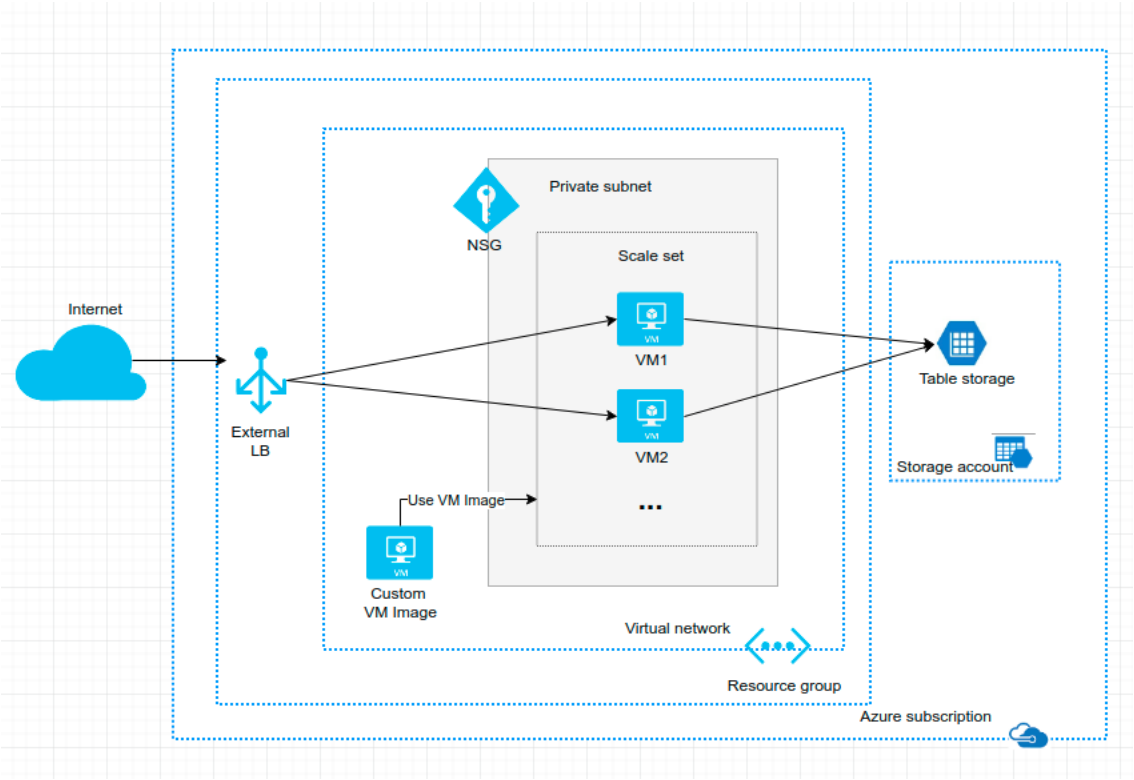
AVAILABILITY ZONES

- Regions are broken up into physically separate AZs
- AZs have indepenent power, cooling and networking
- 3 AZs are exposed per subscription
- VMs spread over AZs receive 99.99% SLA
- Each AZ can be thought of as separate fault domain and update domain
- Virtual networks span AZs



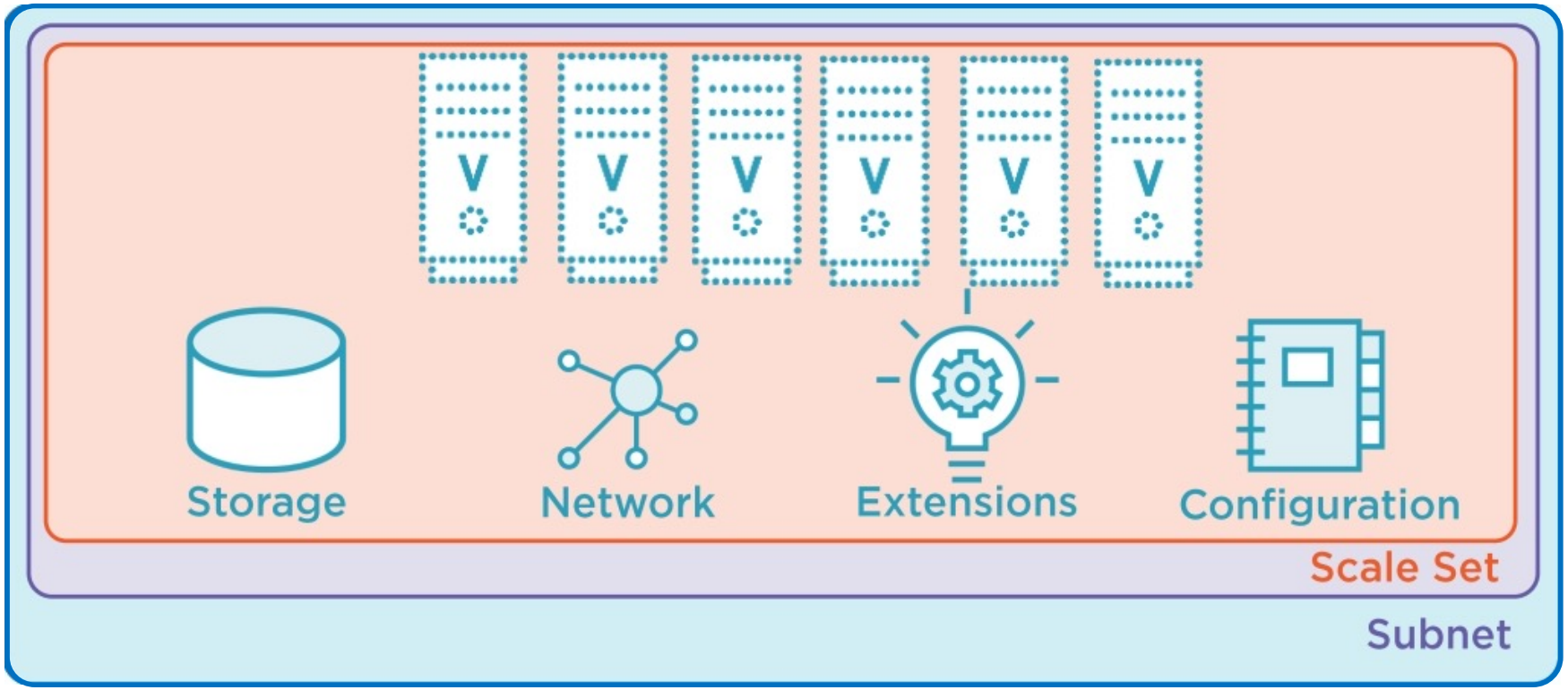
SCALE SETS

- Enables large scale deployment of VMs from single gold image with automatic configuration
- Supports auto-scale based on metrics and schedules
- Scale sets can be updated without taking down the entire set



SCALE SETS

- Azure VM Scale Sets enable the entire deployment to be managed in a simple fashion
- Enable rollout of updates without taking down the entire service.

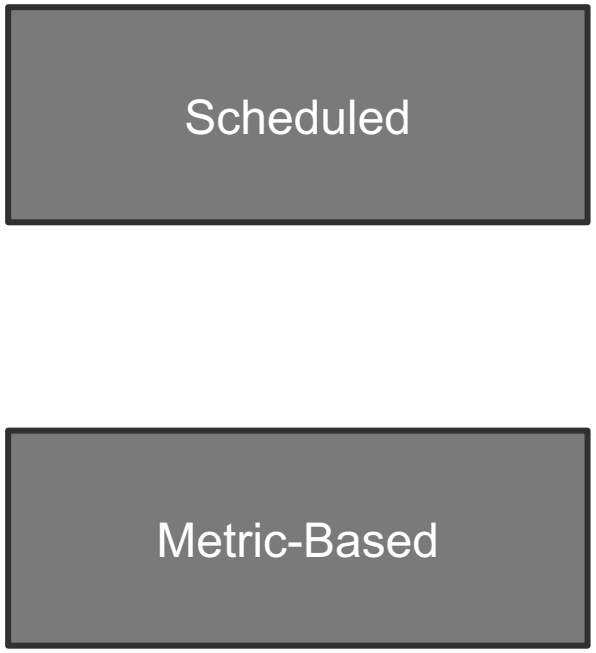


SCALE SETS LIMITATIONS

- 1000 VMs per Scale Set maximum and use with Azure Standard Load Balancer for matching scale

- 2000 Scale Sets per region per subscription

Auto-scaling Scale Sets



Scale Sets support many types of scale

- On a schedule both specific day and recurrence (enables scaling ahead of the load increase)
- Based on resource metrics
- Can combine (schedule rules take precedence over metric rules)

POP QUIZ:

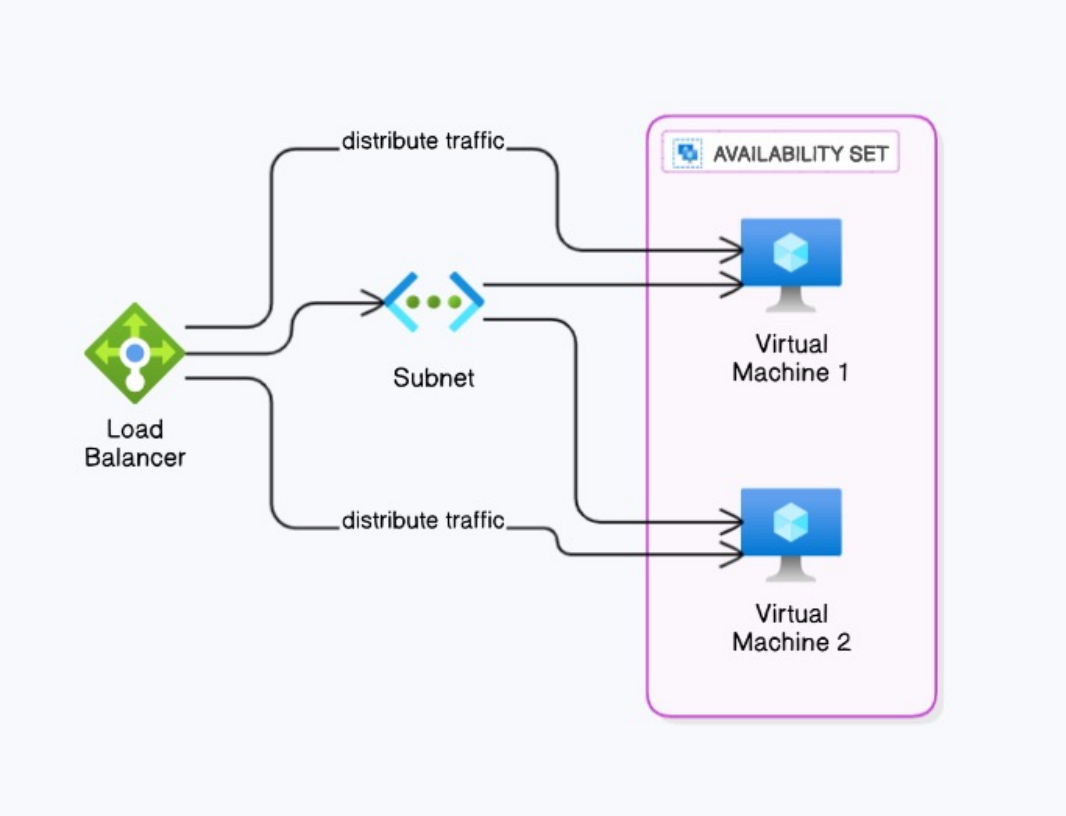
You need high availability in a single region and want to support 200 instances in a scale set. Which option should you choose?
A. Regional scale set (non-zonal) without placement groups
B. Zonal scale set with true Availability Zones
C. Regional scale set with placement groups
D. Single VM in an Availability Set

POP QUIZ:

You need high availability in a single region and want to support 200 instances in a scale set. Which option should you choose?

A. Regional scale set (non-zonal) without placement groups
B. Zonal scale set with true Availability Zones
C. Regional scale set with placement groups
D. Single VM in an Availability Set

SINGLE ZONE DEPLOYMENT



Concept: Deploying application components within a single Azure Availability Zone, typically using Availability Sets.

Resilience: Provides protection against localized hardware failures within a data center.

Network Layout: All VMs, subnets, and load balancers reside within the same Availability Zone.

Use Cases: Non-critical applications, dev/test environments, or regions without Availability Zones.

Limitations: Vulnerable to entire Availability Zone outages or data center-wide failures.

SINGLE ZONE DEPLOYMENT - COMPONENTS

Virtual Network (VNet): A single VNet is created in the chosen region.

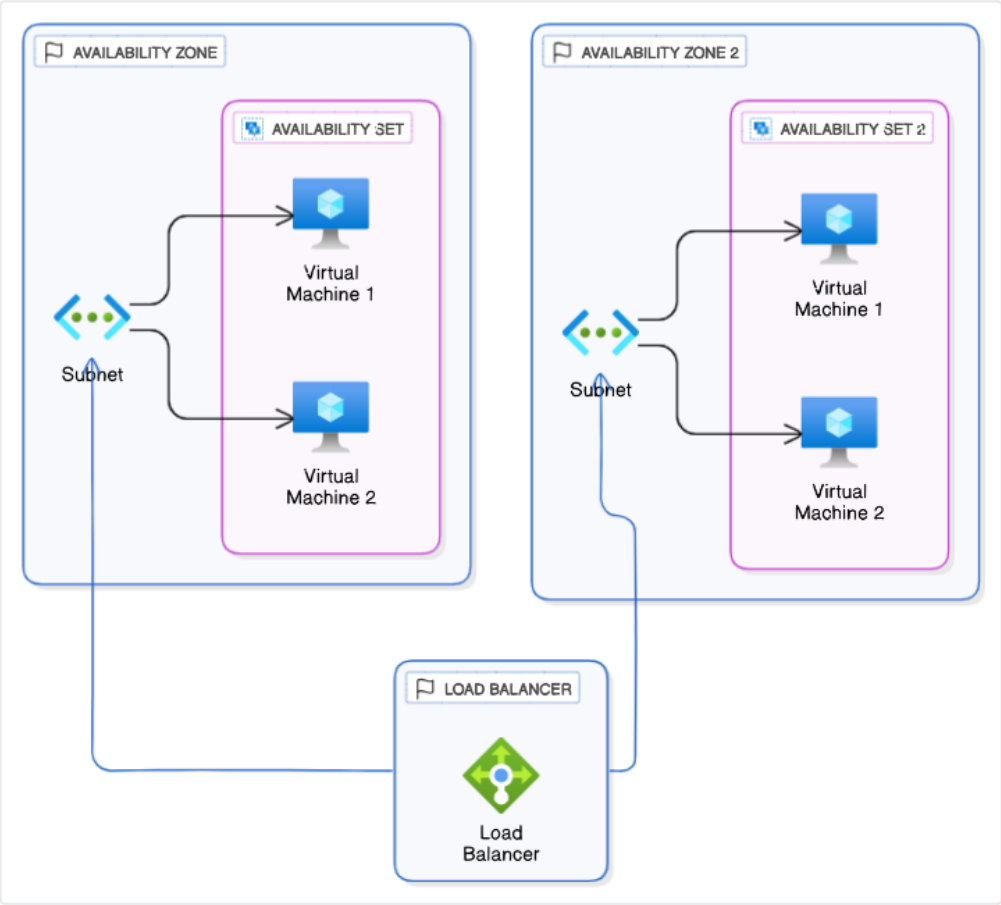
Subnets: Subnets are defined within the VNet.

Availability Set: VMs for each tier are placed into an Availability Set for high availability within a data center.

Azure Load Balancer (Standard): Deployed as zonal if pinned to a specific zone, or just regional. Distributes traffic to VMs in the Availability Set.

NSGs: Applied to subnets or NICs for traffic filtering.

MULTI ZONE DEPLOYMENT



Concept: Distributing application components across multiple Availability Zones within a single Azure region.

High Availability: Provides protection against entire data center failures within a region.

Zone-Redundant Services: Leverage Azure services that are inherently zone-redundant (e.g., Standard Load Balancer, Azure SQL Database).

Network Layout: VNets are regional; subnets are made zone-aware (implicitly or explicitly). VMs deployed across different zones.

Use Cases: Production applications requiring high intra-region resilience

MULTI ZONE DEPLOYMENT - COMPONENTS

- **Virtual Network (VNet):** A single VNet spanning the Azure region. Subnets are implicitly zone-aware.
- **Subnets:** Same subnets as single zone, but VMs are deployed into different zones within these subnets.
- **Zone-Redundant Azure Load Balancer:** Standard SKU is deployed as 'zone-redundant', distributing traffic across zones.
- **Zone-Aware VMs:** Virtual Machines are explicitly deployed into specific Availability Zones.
- **Zone-Redundant PaaS:** Utilize zone-redundant Azure PaaS services (e.g., Azure SQL DB, Service Bus).
- **NSGs:** Applied to subnets or NICs as usual.

SINGLE REGION DEPLOYMENT

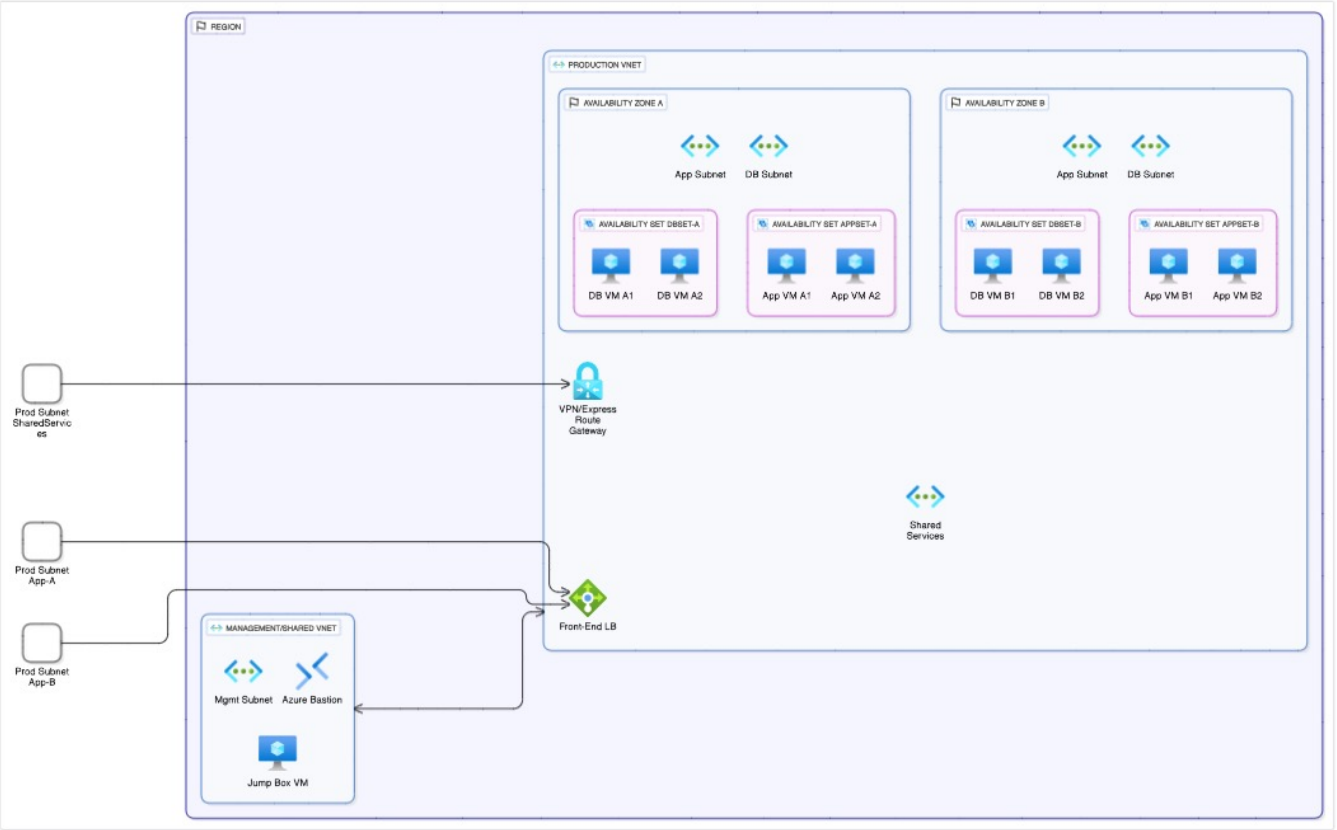
Concept: A complete application stack deployed within a single Azure region, leveraging Availability Zones for high availability.

Comprehensive HA: Includes network, compute, storage, and database components designed for regional resilience.

VNet Peering (Optional): May use VNet peering to connect production and management VNets within the same region.

Connectivity: VPN Gateway or ExpressRoute f hybrid connectivity to on-premises.

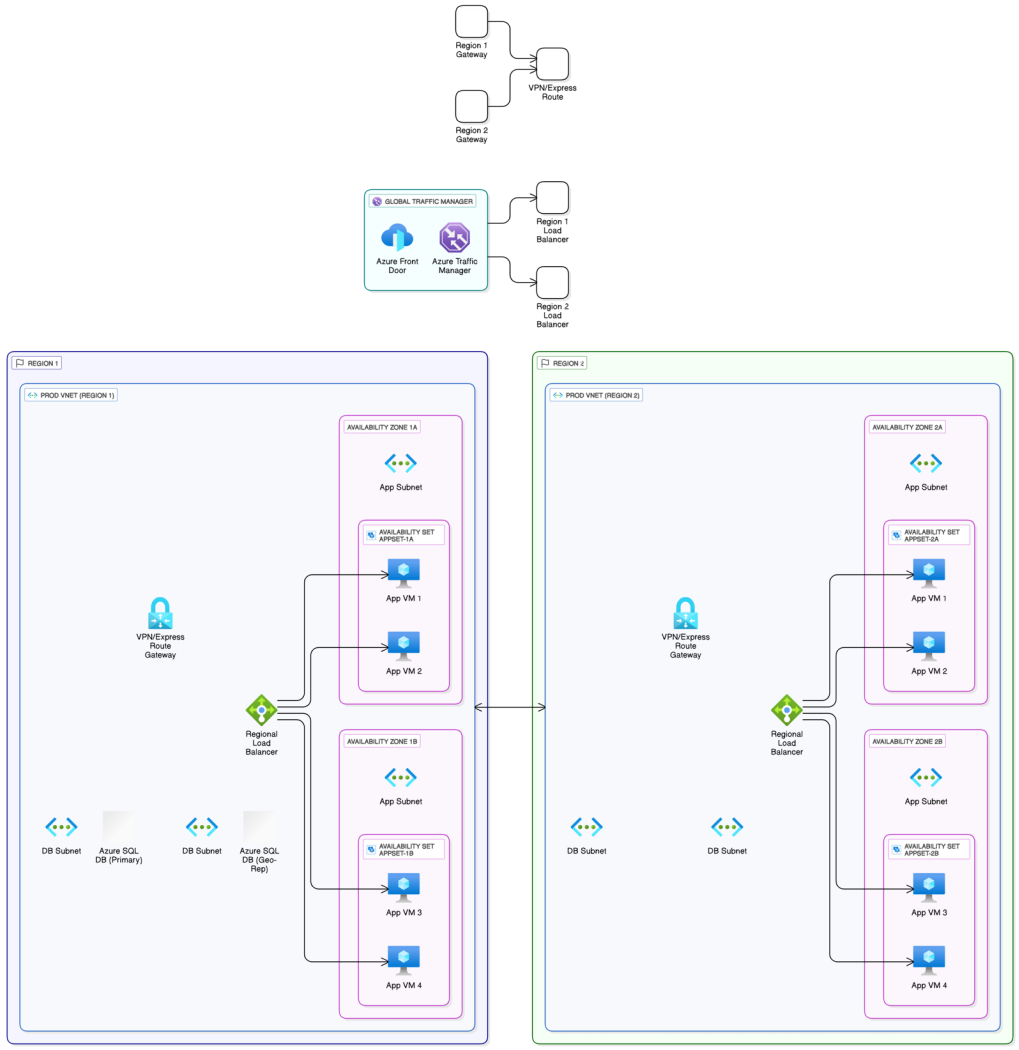
Use Cases: Applications requiring high availability within a region, suitable for most production workloads



SINGLE REGION DEPLOYMENT - COMPONENTS

- **Hub-Spoke Topology:** Often includes a central "hub" VNet for shared services (e.g., Azure Firewall, VPN Gateway) peered to "spoke" workload VNets.
- **Azure Firewall:** Deployed in the hub VNet for centralized North-South and East-West traffic inspection.
- **VPN Gateway/ExpressRoute Gateway:** In the hub VNet for hybrid connectivity.
- **Azure Application Gateway:** For Layer 7 web traffic load balancing and WAF.
- **Azure Load Balancer:** For Layer 4 internal/external traffic distribution.
- **Azure Private DNS Zones:** For internal name resolution across peered VNets.

MULTI REGION DEPLOYMENT



Concept: Deploying application components across two or more Azure regions for global availability and disaster recovery.

Disaster Recovery (DR): Provides failover capabilities in case of an entire Azure region outage.

Global Traffic Management: Uses services like Azure Front Door or Traffic Manager to direct users to the healthiest/closest region.

Data Replication: Critical for DR; ensure data is replicated between regions (e.g., Azure SQL Database Geo-replication).

Global VNet Peering (Optional): Can connect VNets in different regions for backend communication.

MULTI REGION DEPLOYMENT - COMPONENTS

- **Regional VNets:** Each region has its own VNet, typically following a single-region hub-spoke model.
- **Azure Front Door:** For global Layer 7 load balancing and WAF, directing users to the best region.
- **Azure Traffic Manager:** For DNS-based global traffic routing and failover across regions (e.g., for non-HTTP/HTTPS services).
- **Global VNet Peering:** To connect VNets in different regions (e.g., for cross-region data sync, central management).
- **ExpressRoute Global Reach:** Extends your on-premises ExpressRoute circuit to another ExpressRoute circuit in another region.
- **Regional Load Balancers/App Gateways:** Within each region, local load balancing is still handled by regional services.

NETWORK CONNECTIVITY PATTERNS

Within a VNet

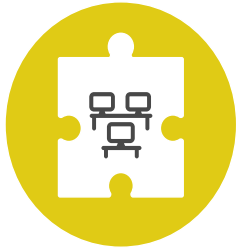
Resources communicate seamlessly via subnets within a virtual network.

VPN Gateway

Establishes secure VPN tunnels between Azure VNets and on-premises networks.

Azure Front Door and Traffic Manager

Connects distributed users to geographically dispersed applications using intelligent routing.



VNet Peering

Connects two Azure VNets, enabling private and secure communication.

ExpressRoute

Creates a private, dedicated connection directly to Azure's network.

VNET PEERING

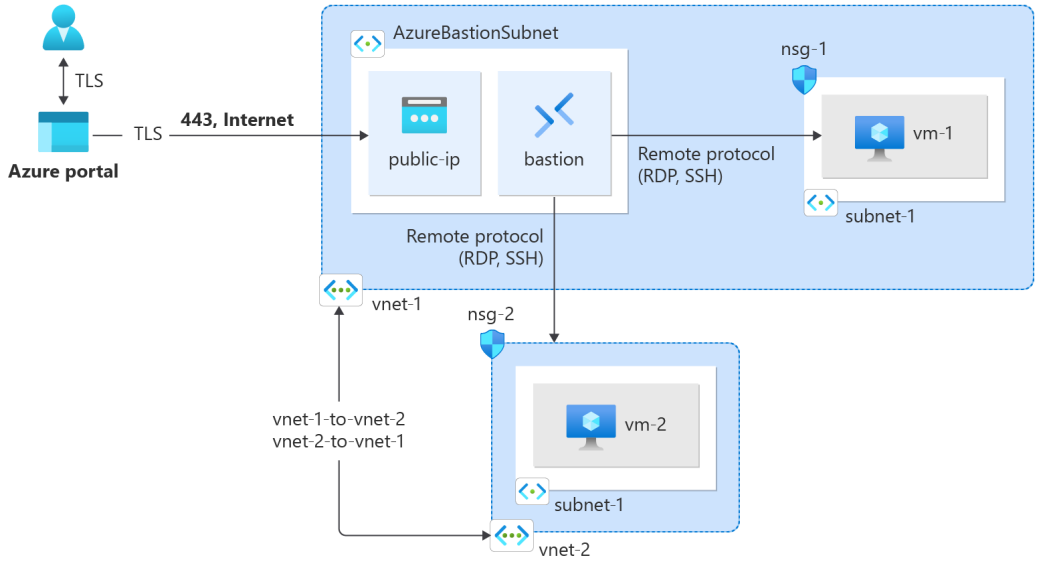
Connects Two VNets: Creates a low-latency, high-bandwidth connection between two Azure Virtual Networks.

Transparent Connectivity: Resources in peered VNets communicate as if they were in the same VNet.

Traffic Routing: Azure automatically routes traffic directly between peered VNets.

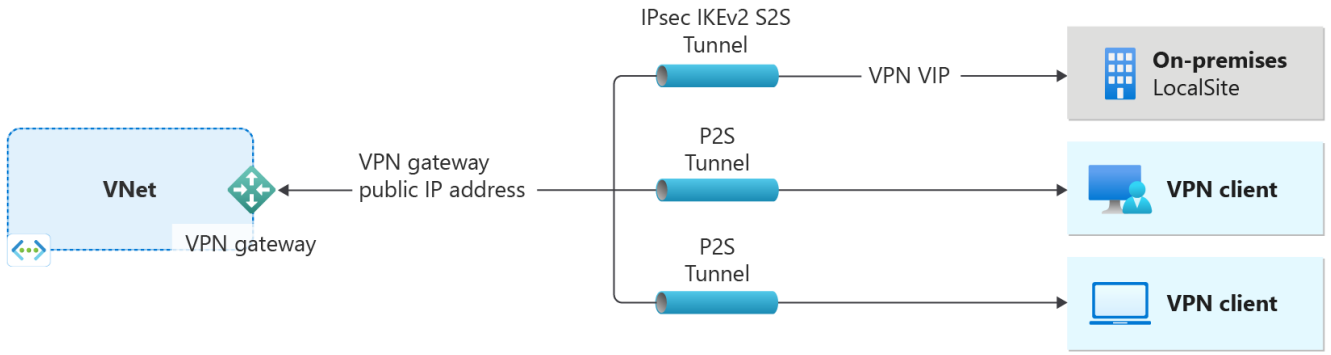
Non-Transitive: If VNet A is peered to VNet B, and VNet B is peered to VNet C, A and C are *not* automatically peered.

Cross-Region Support: Global VNet peering allows connecting VNets in different Azure regions.



https://learn.microsoft.com/en-us/azure/virtual-network/media/tutorial-connect-virtual-networks-portal/resources-diagram.png

HYBRID CONNECTIVITY: VPN GATEWAY



IPsec VPN Tunnels: Creates secure IPsec (Internet Protocol Security) VPN tunnels over the public internet.

Site-to-Site (S2S): Connects your on-premises network to an Azure VNet, usually via a VPN device.

Point-to-Site (P2S): Enables individual clients to securely connect to an Azure VNet.

VNet-to-VNet: Connects two Azure VNets securely over the Azure backbone.

Managed Service: Azure manages the underlying VPN hardware and software.

Cost-Effective: More economical than ExpressRoute for smaller bandwidth needs.

HYBRID CONNECTIVITY: EXPRESSROUTE

Private, Dedicated Connection: Creates a private, high-bandwidth, low-latency connection from on-premises to Azure.

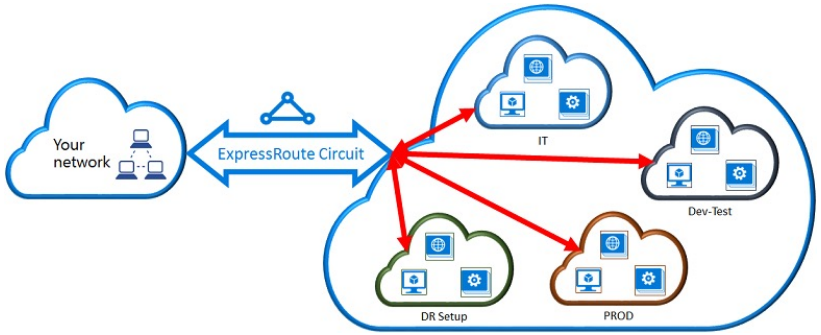
Bypasses Public Internet: Traffic does not traverse the public internet, enhancing security and predictability.

Higher Bandwidth: Offers bandwidths from 50 Mbps up to 100 Gbps.

SLA and Reliability: Provides higher SLAs and reliability compared to internet-based VPNs.

Global Reach: ExpressRoute Global Reach connects your on-premises networks across multiple ExpressRoute locations.

Use Cases: Mission-critical applications, large data migrations, consistent performance.



https://learn.microsoft.com/en-us/azure/expressroute/media/expressroute-howto-linkvnet-portal-resource-manager/cross-subscription.png

VPN GATEWAY VS. EXPRESSROUTE

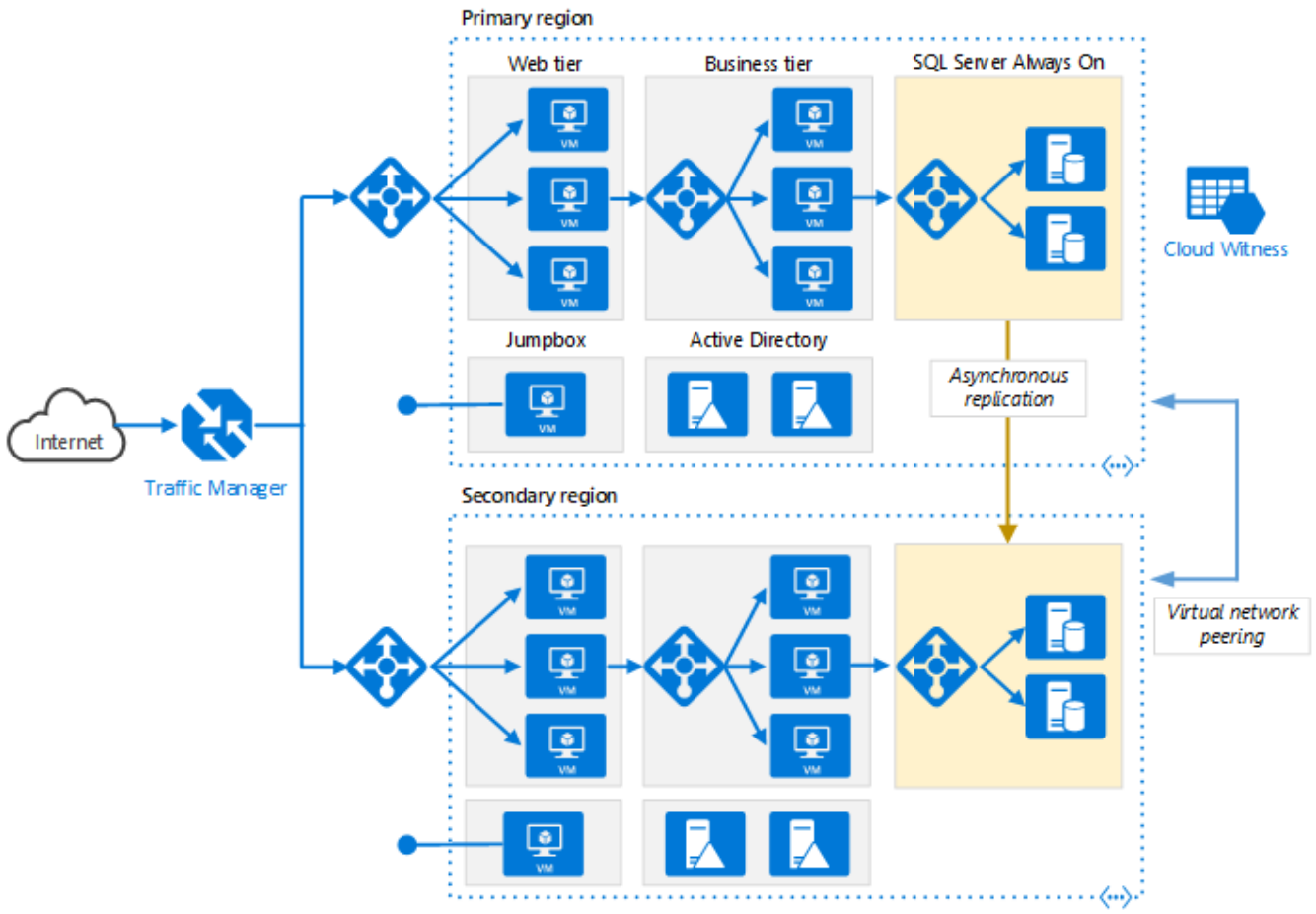
VPN Gateway:

- **Connectivity:** Over public internet.
- **Cost:** Lower.
- **Bandwidth:** Max ~1.25 Gbps (VPN GW UltraPerformance).
- **SLA:** 99.9% (for Basic/Standard/HighPerformance SKUs).
- **Ideal For:** Dev/test, small/medium workloads, branch offices.

ExpressRoute:

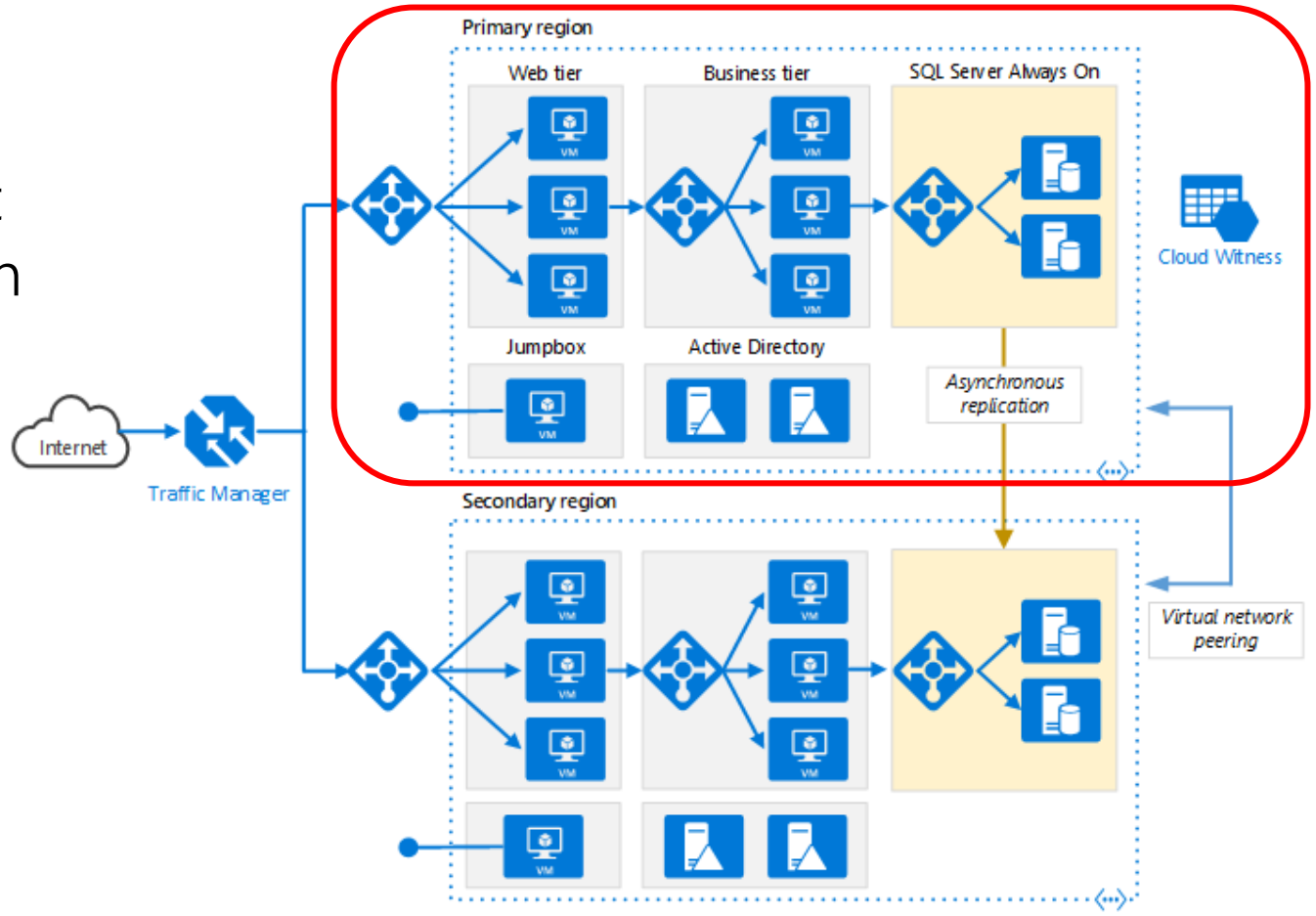
- **Connectivity:** Private connection (MPLS, Ethernet).
- **Cost:** Higher.
- **Bandwidth:** Max 100 Gbps.
- **SLA:** 99.9% (Azure side) + provider SLA.
- **Ideal For:** Enterprise-grade, mission-critical, large data transfer, strict compliance.

RESILIENT APPLICATION ARCHITECTURE



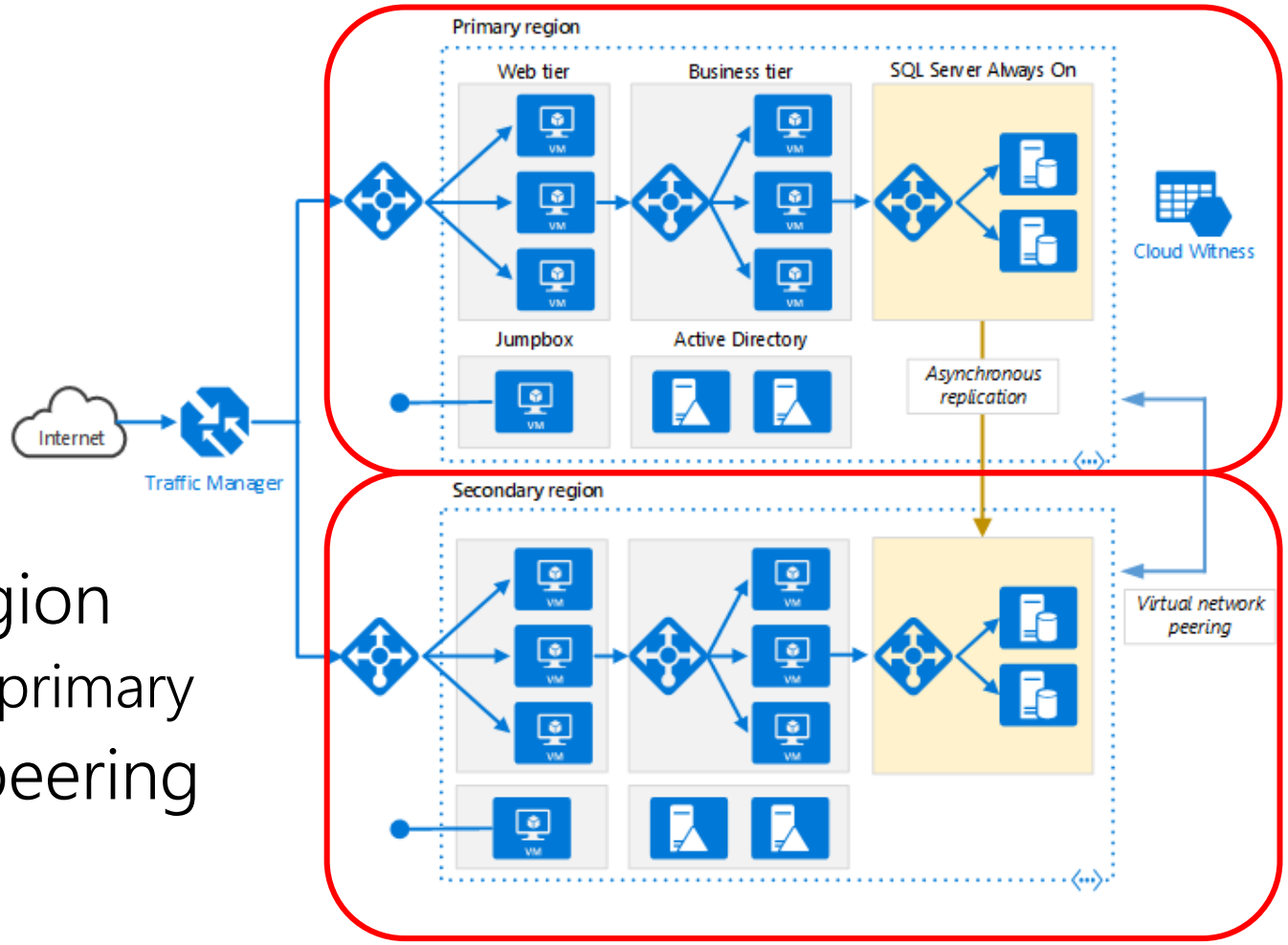
RESILIENT APPLICATION ARCHITECTURE

- Primary region
  - Separate VNet for each region
- VMs in ScaleSet



RESILIENT APPLICATION ARCHITECTURE

- Secondary region
  - duplicate of primary
- Global VNet peering



FAILOVER ARCHITECTURES AND BEST PRACTICES

Active-Active vs. Active-Passive
Trade-offs in complexity, cost, and consistency

Health-Based Failover
Automate using probes and Route 53–style policies

Automated Runbooks
Use Automation or Functions to orchestrate failover tasks

Data Replication
Geo-redundant storage, Cosmos DB multi-master, or SQL failover groups

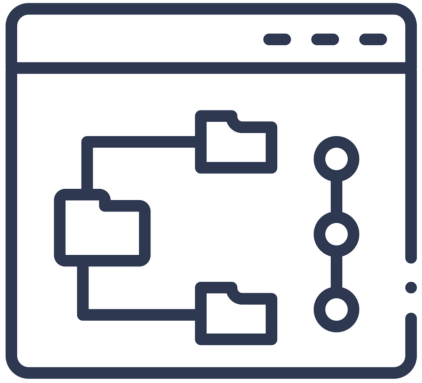
Chaos Engineering
Regularly test failover paths with controlled experiments

LAB 11: DEPLOYING VMS WITH AVAILABILITY ZONES

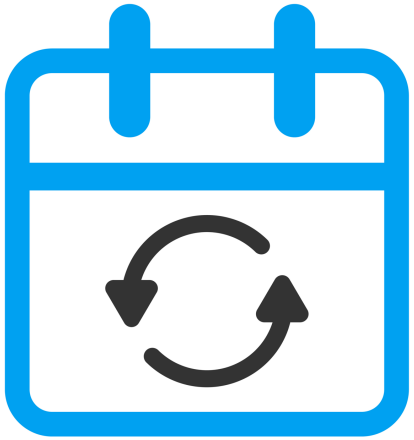
Enhance fault tolerance: Deploy VMs across Availability Zones, add a zone-redundant load balancer, and validate high availability during simulated failures.

AUTOMATING INFRASTRUCTURE

Provisioning resources



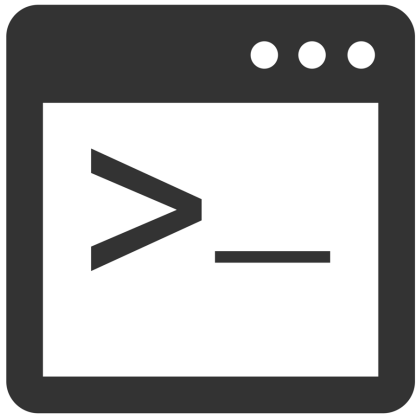
Version Control



Plan Updates

Reusable Templates

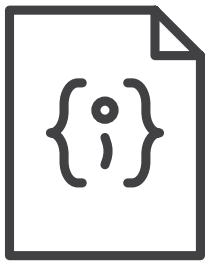
TERRAFORM COMPONENTS



Terraform executable

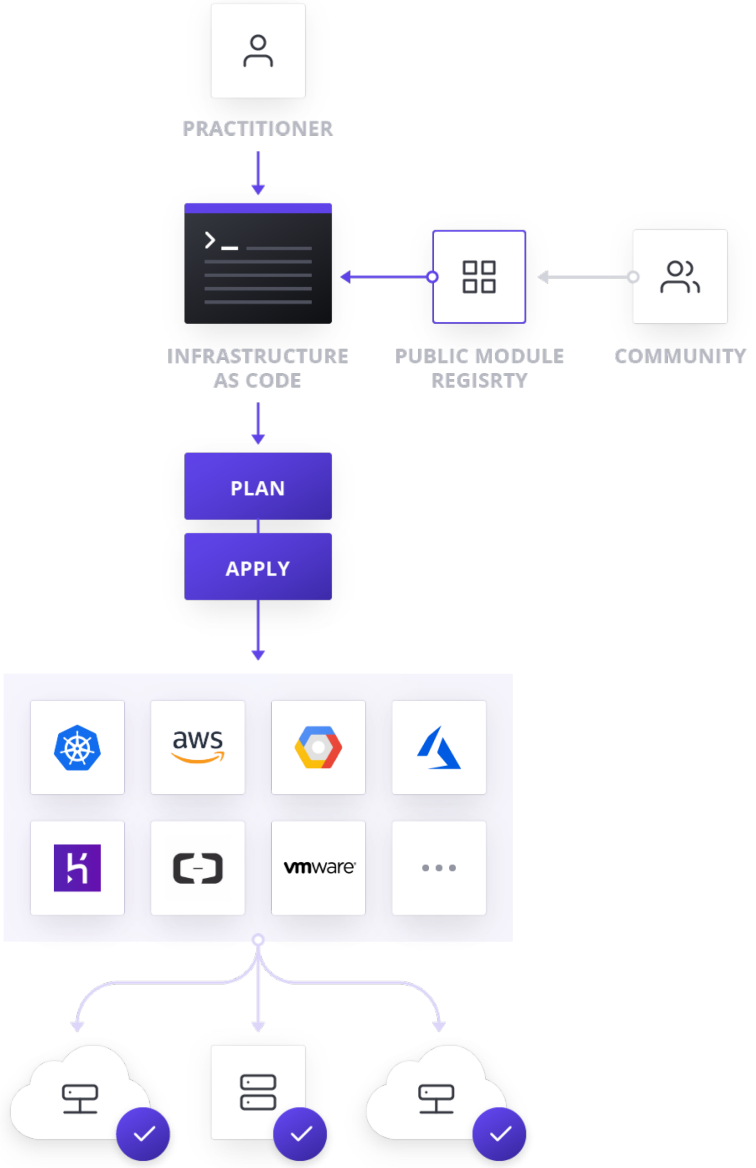
Terraform files

Terraform plugins



Terraform state

TERRAFORM ARCHITECTURE



Terraform files

```
##Amazon Infrastructure
provider "aws" {
  region = "${var.aws_region}"
}
```

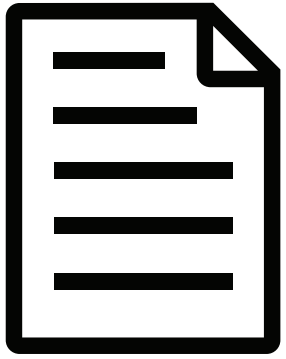
- Provider defines what infrastructure Terraform will be managing
  - Pass variables to provider
    - Region, Flavor, Credentials

Terraform files

```
resource "aws_instance" "aws-k8s-master" {
  subnet_id              = "${var.aws_subnet_id}"
  depends_on             = ["aws_security_group.k8s_sg"]
  ami                    = "${data.aws_ami.ubuntu.id}"
  instance_type          = "${var.aws_instance_size}"
  vpc_security_group_ids = ["${aws_security_group.k8s_sg.id}"]
  key_name               = "${var.aws_key_name}"
  count                  = "${var.aws_master_count}"
  root_block_device {
    volume_type           = "gp2"
    volume_size           = 20
    delete_on_termination = true
  }
  tags {
    Name = "k8s-master-${count.index}"
    role = "k8s-master"
  }
}
```

- Resource: Defines specifications for creation of infrastructure.

Terraform files



```
resource "aws_instance" "aws-k8s-master" {
  subnet_id              = "${var.aws_subnet_id}"
  depends_on             = ["aws_security_group.k8s_sg"]
  ami                    = "${data.aws_ami.ubuntu.id}"
  instance_type          = "${var.aws_instance_size}"
  vpc_security_group_ids = ["${aws_security_group.k8s_sg.id}"]
  key_name               = "${var.aws_key_name}"
  count                  = "${var.aws_master_count}"
  root_block_device {
    volume_type           = "gp2"
    volume_size           = 20
    delete_on_termination = true
  }
  tags {
    Name = "k8s-master-${count.index}"
    role = "k8s-master"
  }
}
```

- Value is 'count' is setup by variable in variables.tf

TERRAFORM CLI

Run `terraform` command

Output:

```
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform
interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
    get                Download and install modules for the
configuration
    graph              Create a visual graph of Terraform
resources
```

TERRAFORM CLI

Command:

```
terraform init
```

Output:

```
Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "docker"...
```

Terraform fetches any required providers and modules and stores them in the .terraform directory. Check it out and you'll see a plugins folder.

TERRAFORM CLI

Command:

```
terraform validate
```

Validate all of the Terraform files in the current directory. Validation includes basic syntax check as well as all variables declared in the configuration are specified.

TERRAFORM CLI

Command:

```
terraform plan
```

Output:

```
Terraform will perform the following actions:

  + aws_instance.aws-k8s-master
      id:                                   <computed>
      ami:                                  "ami-01b45..."
      instance_type:                        "t3.small"
```

Plan is used to show what Terraform will do if applied. It is a dry run and does not make any changes.

TERRAFORM CLI

Command:

```
terraform apply
```

Output:

```
❯ terraform apply "rapid-app.out"
aws_security_group.k8s_sg: Creating...
  arn:                                      "" => "<computed>"
  description:                              "" => "Allow all
inbound traffic necessary for k8s"
  egress.#:                                 "" => "1"
  egress.482069346.cidr_blocks.#:           "" => "1"
  egress.482069346.cidr_blocks.0:           "" => "0.0.0.0/0"
```

Performs the actions defined in the plan

TERRAFORM CLI

Command:

```
terraform state show <resource>
```

```
terraform state show aws_instance.lab2-tf-example
```

Output:

```
ami                           = "ami-830c94e3"
availability_zone             = "us-west-2b"
cpu_core_count                = 1
...
```

Shows information about provided resource.

TERRAFORM CLI

The `terraform refresh` command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. This can be used to detect any drift from the last-known state, and to update the state file.

This does not modify infrastructure but does modify the state file. If the state is changed, this may cause changes to occur during the next `plan` or `apply`.

TERRAFORM CLI

Command:

```
terraform destroy [-auto-approve]
```

Destroys all the resources in the state file.
-auto-approve (don't prompt for confirmation)

AUTOMATION WITH TERRAFORM

Infrastructure as Code (IaC): Defines infrastructure in configuration files rather than manual processes.

HashiCorp Terraform: A popular, open-source IaC tool supporting multiple cloud providers, including Azure.

Declarative Syntax: You declare the desired state of your infrastructure, and Terraform figures out how to achieve it.

State Management: Terraform maintains a state file to track deployed resources and manage dependencies.

Plan, Apply, Destroy: Core workflow involves planning changes, applying them, and destroying resources.

WHY AUTOMATE AZURE N/WORKING WITH TERRAFORM

- **Consistency:** Ensures identical network configurations across environments (dev, test, prod).
- **Repeatability:** Deploy complex network topologies repeatedly and reliably.
- **Version Control:** Store network configurations in Git, enabling change tracking, rollbacks, and collaboration.
- **Reduced Errors:** Eliminates manual misconfigurations, leading to fewer network issues.
- **Speed & Agility:** Accelerate provisioning of network infrastructure for new projects or environments.
- **Documentation:** Terraform code itself serves as living documentation of your network design.

TERRAFORM FOR AZURE NETWORKING

- **Azure Provider:** Terraform's Azure provider (azurerm) enables interaction with Azure resources.
- **Resources:** Define specific Azure network components (e.g., azurerm_virtual_network, azurerm_subnet).
- **Data Sources:** Retrieve information about existing Azure resources.
- **Variables:** Parameterize your configurations for reusability.
- **Outputs:** Define values to be displayed or used by other Terraform configurations.
- **Modules:** Package reusable blocks of Terraform configurations (e.g., a common VNet setup).

CODE EXAMPLE

```
resource "azurerm_resource_group" "rg" {
  name     = "my-network-rg"
  location = "East US"
}

resource "azurerm_virtual_network" "vnet" {
  name                = "my-vnet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}

resource "azurerm_subnet" "subnet1" {
  name                 = "subnet-app"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}

resource "azurerm_subnet" "subnet2" {
  name                 = "subnet-db"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.2.0/24"]
}
```

DEMO: USING TERRAFORM TO DEPLOY RESOURCES TO A MULTI-CLOUD ENVIRONMENT

Step 1: Install Terraform
Step 2: Write config for AWS/Azure
Step 3: Initialize and apply config

STEP 1: INSTALL TERRAFORM

Download Terraform:
Visit the Terraform Downloads page and select the appropriate package for your operating system.

Install Terraform:
Follow the installation instructions specific to your OS.

Verify Installation:
Open your terminal and run:

terraform version

```
PS C:\Users\ishea> terraform version
Terraform v1.11.4
on windows_amd64
```

STEP 2: WRITE CONFIGURATION FOR AWS AND AZURE

Create a Project Directory:
Initialize a new directory for your Terraform project.

Define Providers:
Create a providers.tf file with the following content ->

providers.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  region = "us-west-2"
}

provider "azurerm" {
  features {}
}

```

STEP 2: WRITE CONFIGURATION FOR AWS AND AZURE

Create Resource Definitions:

In a main.tf file, define resources for both AWS and Azure. For example:

providers.tf main.tf

main.tf

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}

resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "East US"
}

resource "azurerm_virtual_network" "example" {
  name                = "example-network"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
}

```

STEP 3: INITIALIZE TERRAFORM

Run the following command in your project root directory to initialize the project and download the necessary providers:

terraform init

```
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/azurerm versions matching "~> 3.0"...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.96.0...
- Installed hashicorp/aws v5.96.0 (signed by HashiCorp)
- Installing hashicorp/azurerm v3.117.1...
- Installed hashicorp/azurerm v3.117.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

STEP 4: REVIEW THE EXECUTION PLAN

Check what Terraform plans to do by running this command:

terraform plan

```
Terraform will perform the following actions:

  # aws_instance.example will be created
  + resource "aws_instance" "example" {
      + ami                                  = "ami-0c55b159cbfafe1f0"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + enable_primary_ipv6                  = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + spot_instance_request_id             = (known after apply)
      + subnet_id                            = (known after apply)
      + tags_all                             = (known after apply)
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)

      + capacity_reservation_specification (known after apply)

      + cpu_options (known after apply)

      + ebs_block_device (known after apply)

      + enclave_options (known after apply)

      + ephemeral_block_device (known after apply)

      + instance_market_options (known after apply)

      + maintenance_options (known after apply)

      + metadata_options (known after apply)

      + network_interface (known after apply)

      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

  # azurerm_resource_group.example will be created
  + resource "azurerm_resource_group" "example" {
      + id       = (known after apply)
      + location = "eastus"
      + name     = "example-resources"
    }

  # azurerm_virtual_network.example will be created
  + resource "azurerm_virtual_network" "example" {
      + address_space       = [
          + "10.0.0.0/16",
        ]
      + dns_servers         = (known after apply)
      + guid                = (known after apply)
      + id                  = (known after apply)
      + location            = "eastus"
      + name                = "example-network"
      + resource_group_name = "example-resources"
      + subnet              = (known after apply)
    }

Plan: 3 to add, 0 to change, 0 to destroy.
```

STEP 5: APPLY THE CONFIGURATION

Deploy the resources by running this command:

terraform apply

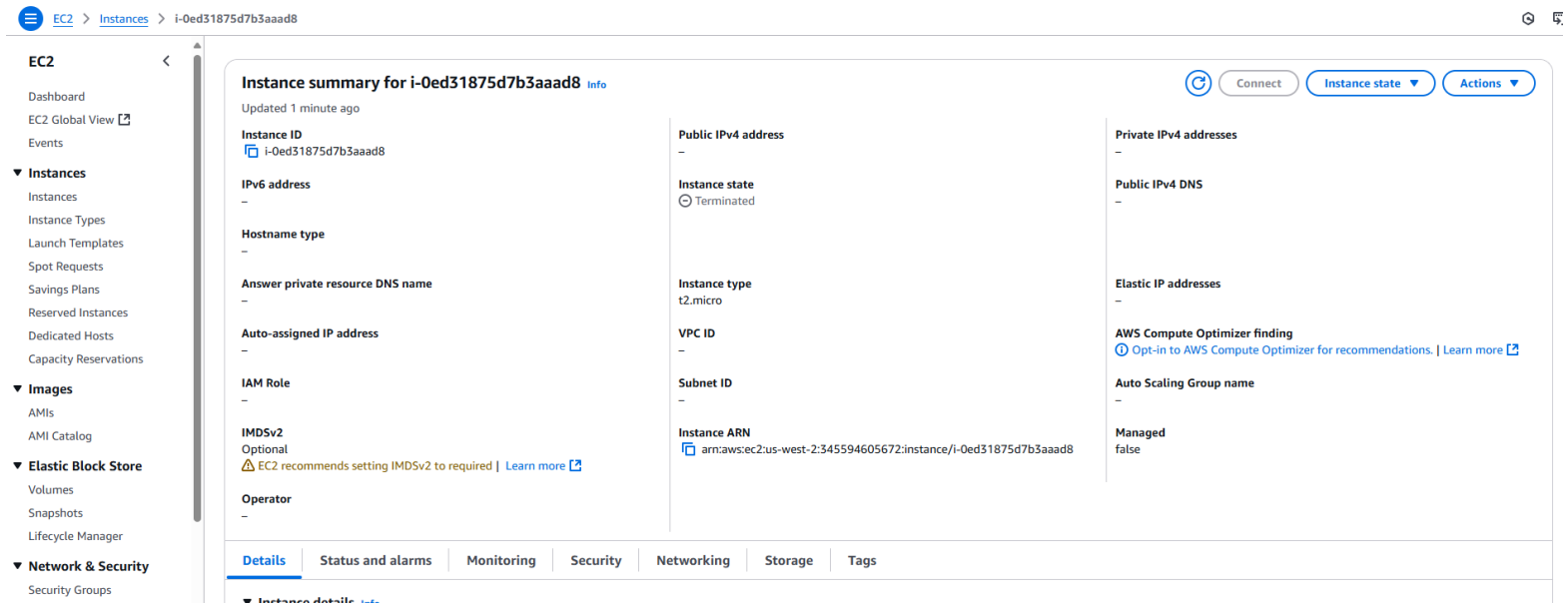
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.example: Creating...
azurerm_resource_group.example: Creating...
azurerm_resource_group.example: Creation complete after 10s [id=/subscriptions[redacted]/resourceGroups/example-resources]
azurerm_virtual_network.example: Creating...
azurerm_virtual_network.example: Creation complete after 5s [id=/subscriptions[redacted]/resourceGroups/example-resources/providers/Microsoft.Network/virtualNetworks/example-network]
```

STEP 6: VERIFY DEPLOYMENTS

Log into your AWS and Azure portals to confirm the resources have been created as defined.



STEP 7: DESTROY RESOURCES

To remove all resources created by Terraform:

terraform destroy

```
aws_instance.example: Refreshing state... [id=i-0ed31875d7b3aaad8]
azurerm_resource_group.example: Refreshing state... [id=/subscriptions/33266a41-134d-4de7-a780-665b38b0f7b8/resourceGroups/example-resources]
azurerm_virtual_network.example: Refreshing state... [id=/subscriptions/33266a41-134d-4de7-a780-665b38b0f7b8/resourceGroups/example-resources/providers/Microsoft.Network/virtualNetworks/example-network]

No changes. No objects need to be destroyed.

Either you have not created any objects yet or the existing objects were already deleted outside of Terraform.

Destroy complete! Resources: 0 destroyed.
```

LAB 12: AUTOMATE NETWORK DEPLOYMENT

Use Terraform to provision a VNet and subnet, apply configurations, and manage resources repeatably for automated Azure networking

INDIVIDUAL KEY TAKEAWAYS

Write down three key insights from today's session.

Highlight how these take aways influence your work.

Q&A AND OPEN DISCUSSION


THANK
YOU