WELCOME to Today's Session

Sit tight. We will start shortly.

Mute your phone.

Register your Attendance via link in chat window.

Prepare to participate via chat and annotations.

AZ-301 TSI Exam Preparation

Today’s Session

There will be a 15-minute break 75-90 minutes into the session

At the end of each section I’ll review the questions/comments in chat before proceeding, and leave time for additional Q&A

If you have found resources that have helped you understand a topic, share them with others via chat

Agenda

- Managed Server Applications
- Serverless Applications
- Integration
- High Performance Hosting
- Patterns
- Building Applications

Creating Managed Server Applications

Infrastructure Backed PaaS

Azure provides several services to cater for highly scaled isolated applications:

- App Service Environments
- Azure Service Fabric
- Azure Container Service

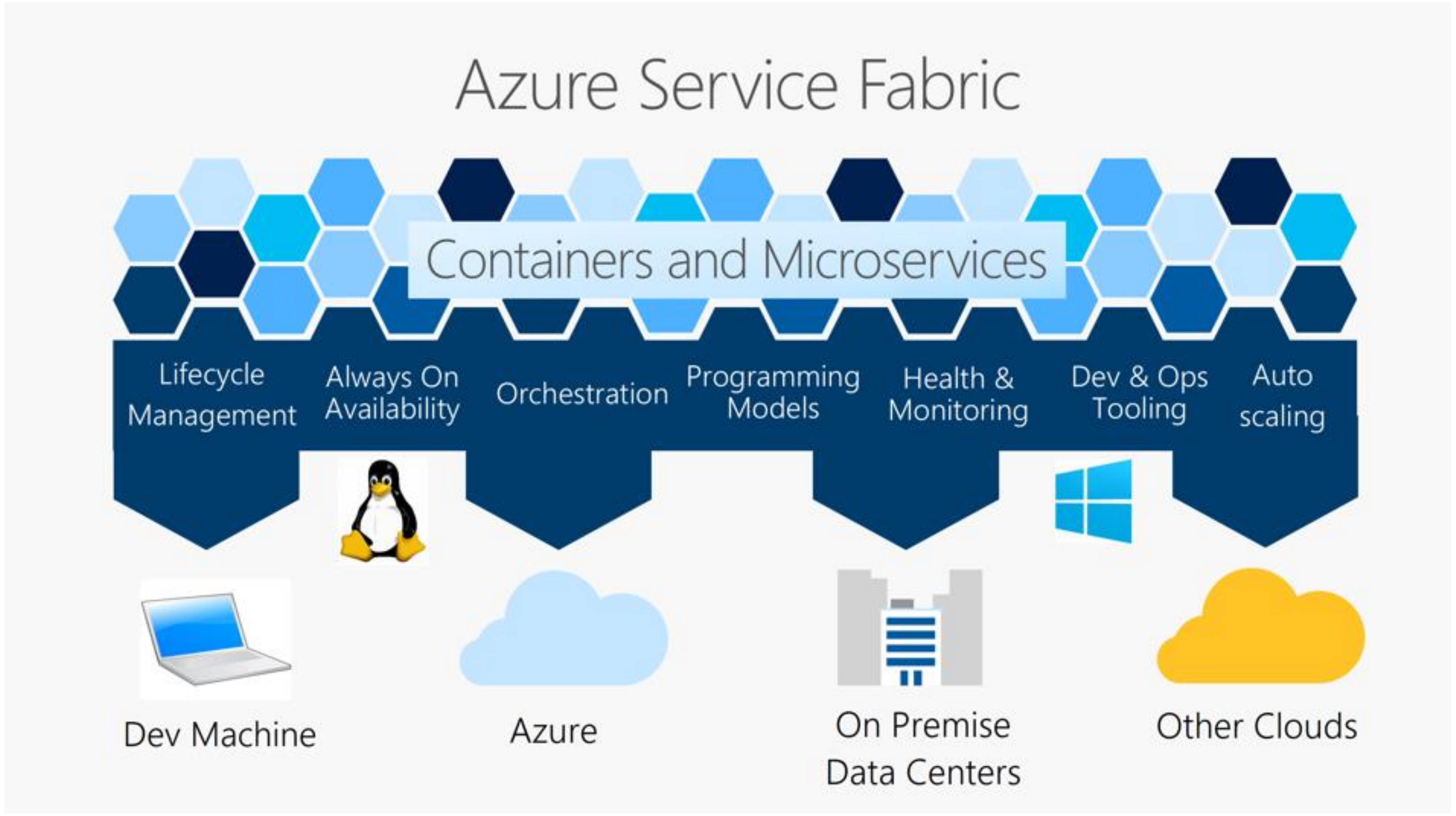
App Service Environments

- Dedicated environment for high scale, secure Apps:
  - Very high scale
  - Isolation with secure network access
  - High memory utilization
- Single or Multi region
- Deployed to a Virtual Network
- An ASE is dedicated exclusively to a single subscription (Max 100 instances)

Azure Service Fabric

- Distributed systems platform to package, deploy and manage microservices and containers
- Avoids complex infrastructure problems
- High density microservice applications running on a shared cluster of machines
- Container deployment and orchestration
- Stateless and stateful services

Azure Service Fabric



Azure Container Service

Simple management of Cluster of VMs using either Docker, Mesosphere DC/OS or Kubernetes

- Removes infrastructure complication and planning
- No cluster charges, just used resources
- Secure, reliable, highly scalable

High Performance Computing (HPC)

HPC is commonly defined as the use of super computers and parallel processing techniques for solving complex computational problems

Azure provides solutions to manage this function:

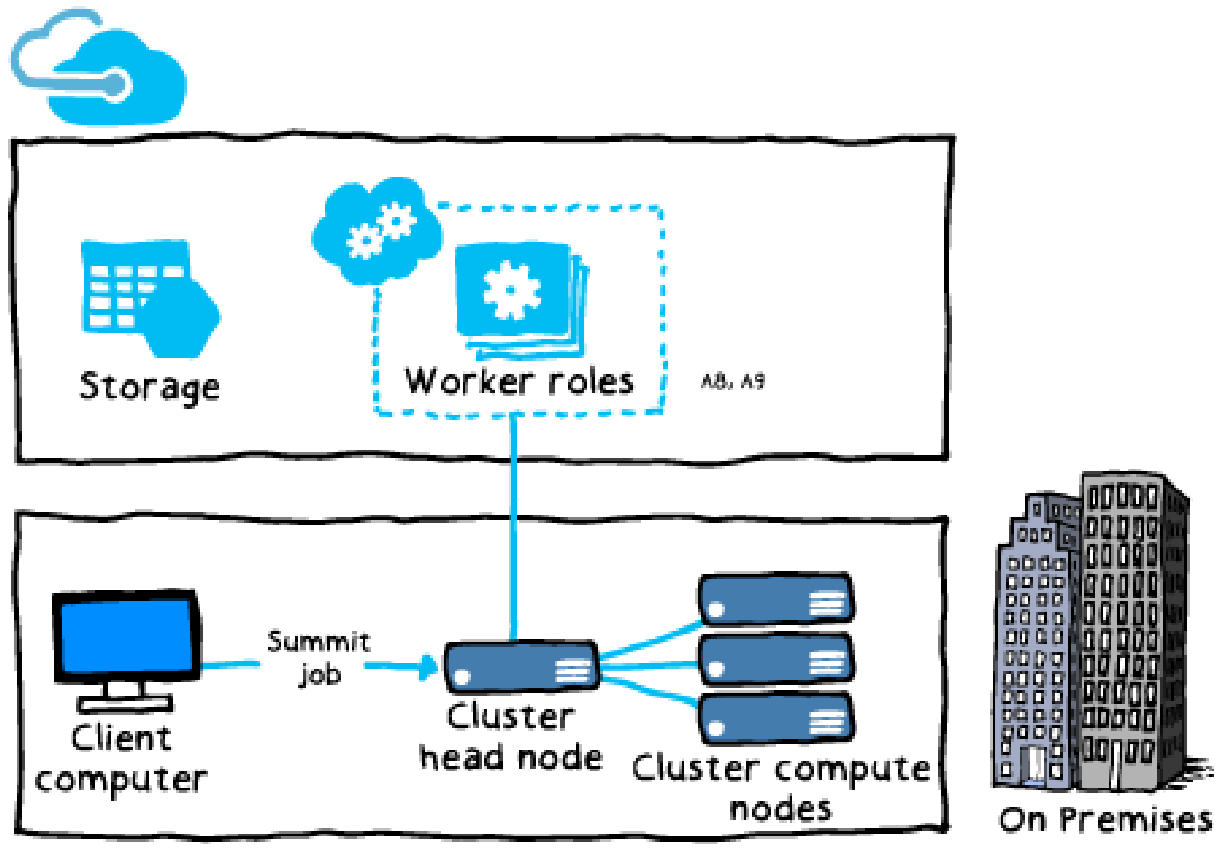
- Custom Workloads on IaaS
- Azure Batch

Custom Workloads on IaaS

HPC Pack using Azure Virtual Machines:

- HPC Pack is Microsoft’s HPC cluster and job management solution for Windows
- Uses Head Nodes and compute nodes – Suggested as A8 and A9 VM sizes
- HPC Pack can also be used in hybrid scenarios to “burst to Azure” with A8 or A9 instances to obtain more processing power

Custom Workloads on IaaS

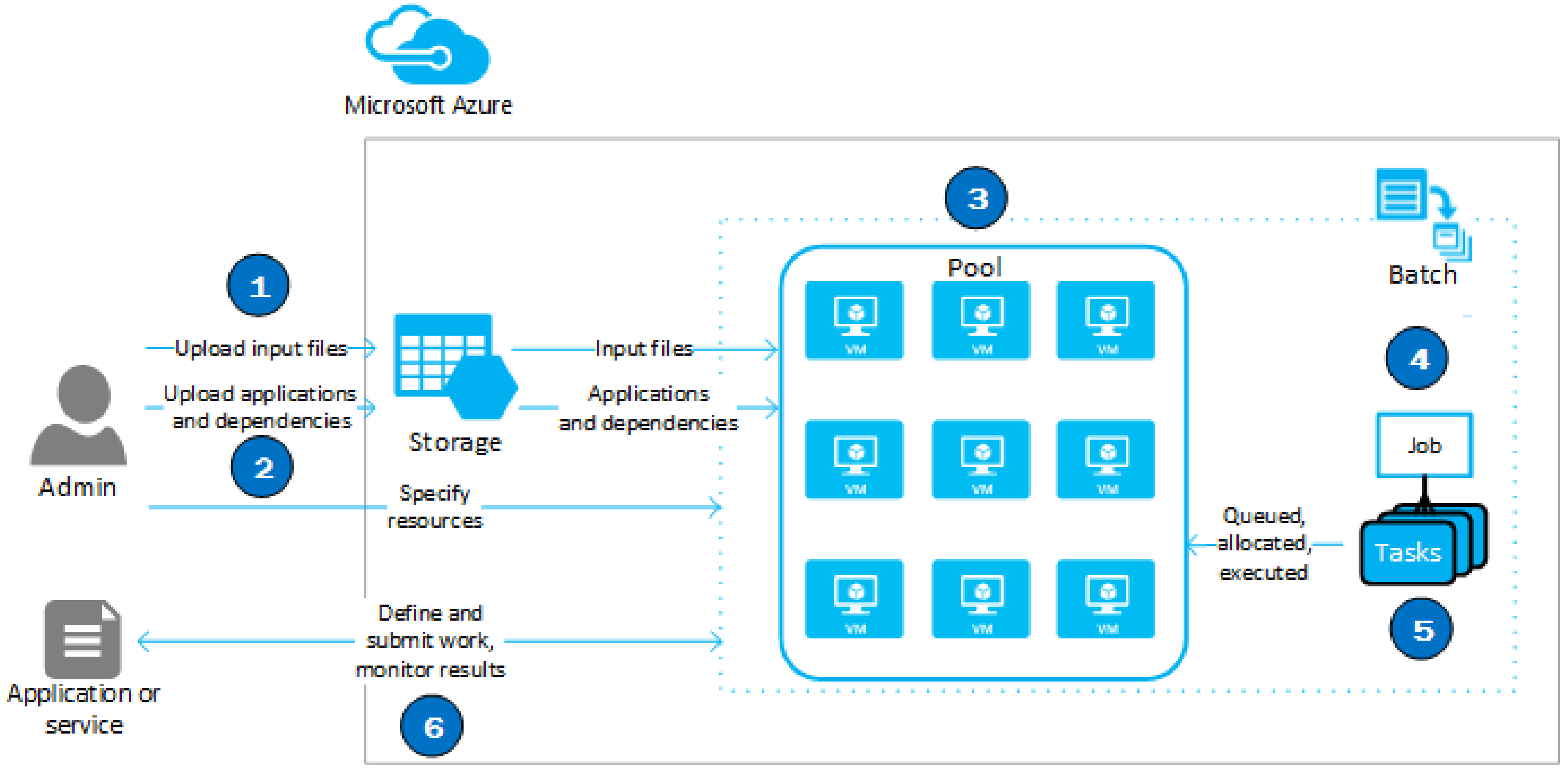


Azure Batch

A free service designed for large data set manipulation and transform workloads:

- Job Scheduling
- Compute resource management
- Large-scale parallel workloads
- Batch API to enable scaling to thousand of compute nodes

Azure Batch


Microsoft Azure
1
Upload input files
Upload applications and dependencies
2
Admin
Storage
Input files
Applications and dependencies
Specify resources
3
Pool
VM
Batch
4
Job
Tasks
Queued, allocated, executed
5
Application or service
Define and submit work, monitor results
6

Stateless Component Workloads

In addition there are several services that are the foundations for HPC Solutions in Azure:

- Virtual Machines (VMs)
- VM Scale Sets
- Azure Container Services
- HDInsight
- Machine Learning

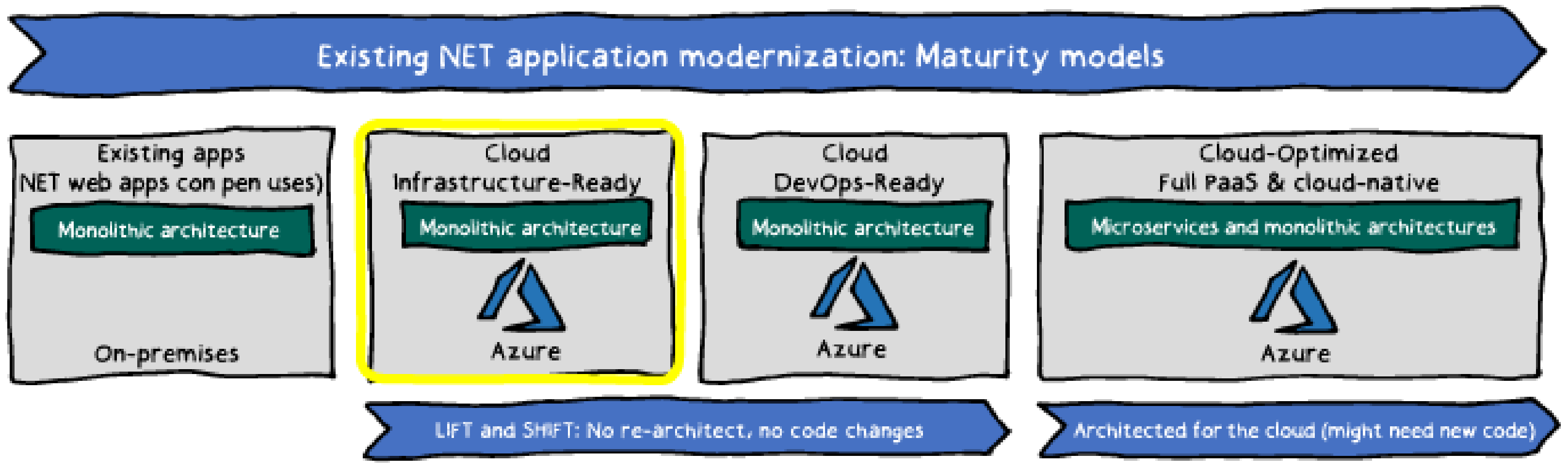
Migration

Several options exist when migrating workloads from on-premises to Azure and from Azure IaaS to PaaS:

- Migrate on-premises to Azure
- Migrate from IaaS to PaaS
- Migrate from Cloud Services to PaaS
- Native app or Migration

On-Premises Lift and Shift

First stage of a migration may be move the workload direct to IaaS in Azure, having completed this the potential is to follow the modernization maturity model


Existing NET application modernization: Maturity models
Existing apps
NET web apps con pen uses)
Monolithic architecture
On-premises
Cloud
Infrastructure-Ready
Monolithic architecture
Azure
Cloud
DevOps-Ready
Monolithic architecture
Azure
Cloud-Optimized
Full PaaS & cloud-native
Microservices and monolithic architectures
Azure
LIFT and SHIFT: No re-architect, no code changes
Architected for the cloud (might need new code)

On-Premises Migration

Two options:

- To IaaS – little code changes – need to manage the OS
- To PaaS – rewrite the code but no OS management required

In either case the data can be hosted in either IaaS or PaaS SQL databases

Migration from Classic IaaS

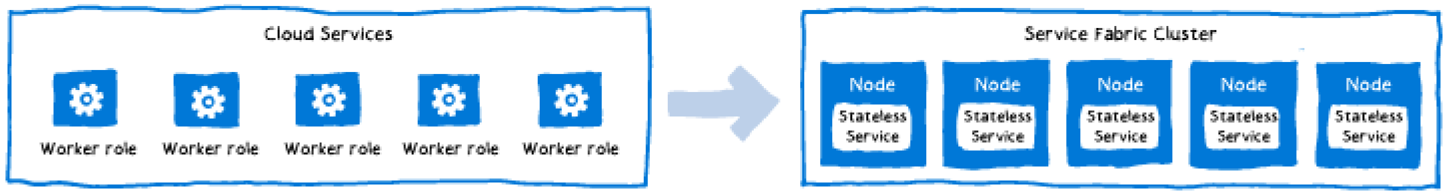
For IaaS migration from Classic to Azure Resource Manager the following can be migrated:

Virtual Machines	VPN Gateways
Availability Sets	Network Security Groups
Cloud Services	Route Tables
Storage Accounts	Reserved IPs
Virtual Networks	Express Route Gateways

Check carefully for unsupported configurations, which could affect successful migration

Migration from Cloud Services

Migrating Web and Worker roles in a cloud service to Azure Fabric stateless services is the simplest method of migration to Service Fabric



Authoring Serverless Applications

Web Apps

- Web Apps:
  - Near instant deployment
  - SSL and Custom Domain Names available in some tiers
  - WebJobs provide background processing for independent scaling
  - Can Scale to larger machines without redeploying applications
- Virtual Machines:
  - Need Availability Sets or Load Balancers to prevent simultaneous restarts for maintenance or hardware failures
  - Additional machines needed for background processing

Web App Deployment

- Create Packages:
  - Continuous Delivery with VSO or GitHub
  - Can use Team Foundation Version Control (TFVC) or Git for source control
- Deployment Slots:
  - Can create slots such as: Staging, Production, Testing
- Web Deploy:
  - Older IIS Extension method to Export and Import
- FTP Deployment

Web App for Containers

Deploy applications and solutions that are containerized directly to App Service Web Apps

- Simplifies deployment
- Matches the already popular container workflow using:
  - CI/CD with Docker Hub, Azure Container Registry or GitHub
- Compatible with existing App Service Features:
  - Auto-scale, Deployment Slots, etc.

Container Orchestration

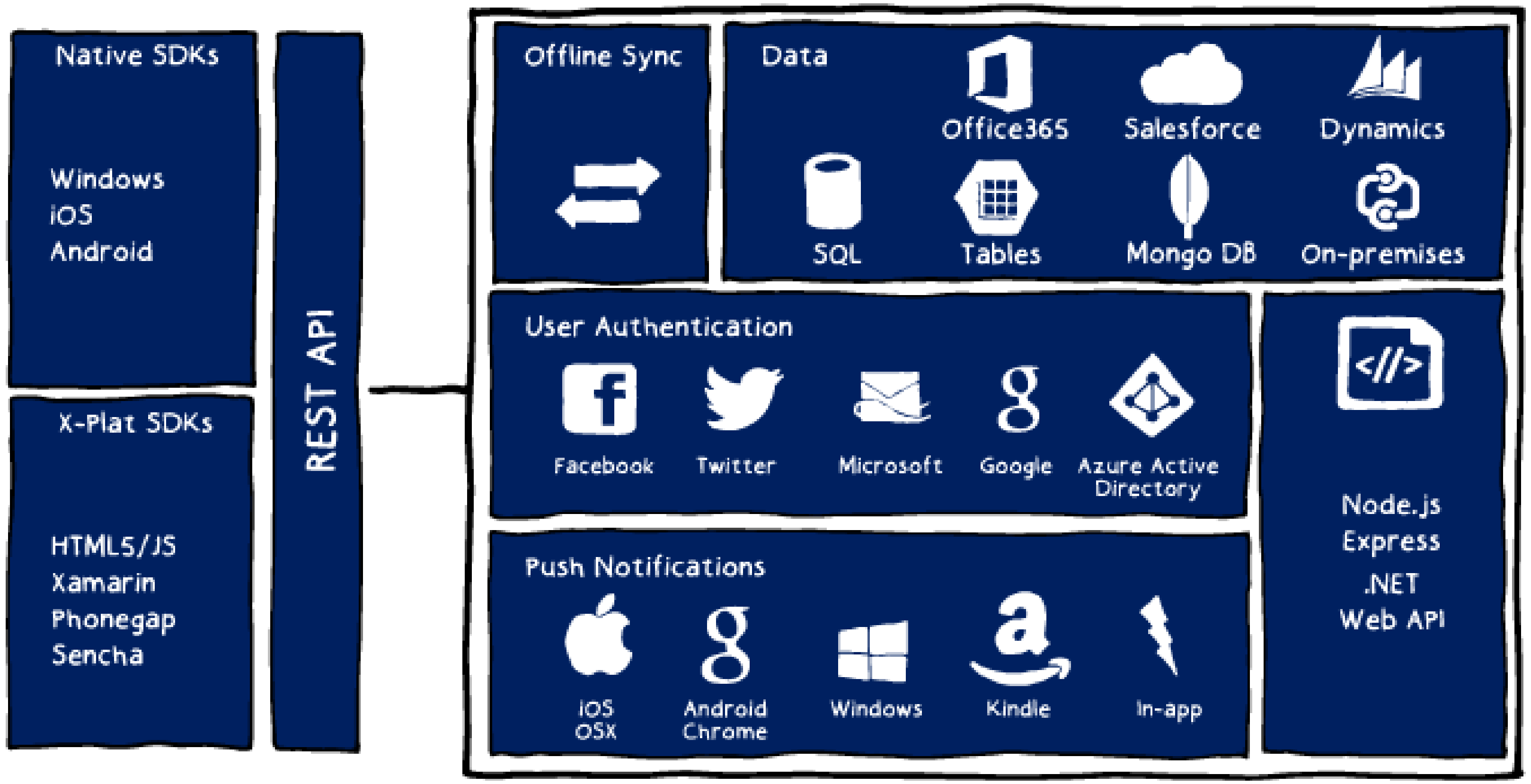
Containers can be sourced from your existing registries

- Docker Hub:
  - Deploy images already shared on Docker Hub
  - Deploy the most popular official images
  - Private images are available on Docker Hub
- Azure Container Registry:
  - Managed service for hosting Docker images
  - Can deploy to Docker Swarm, Kubernetes or Web App for Containers

API Apps

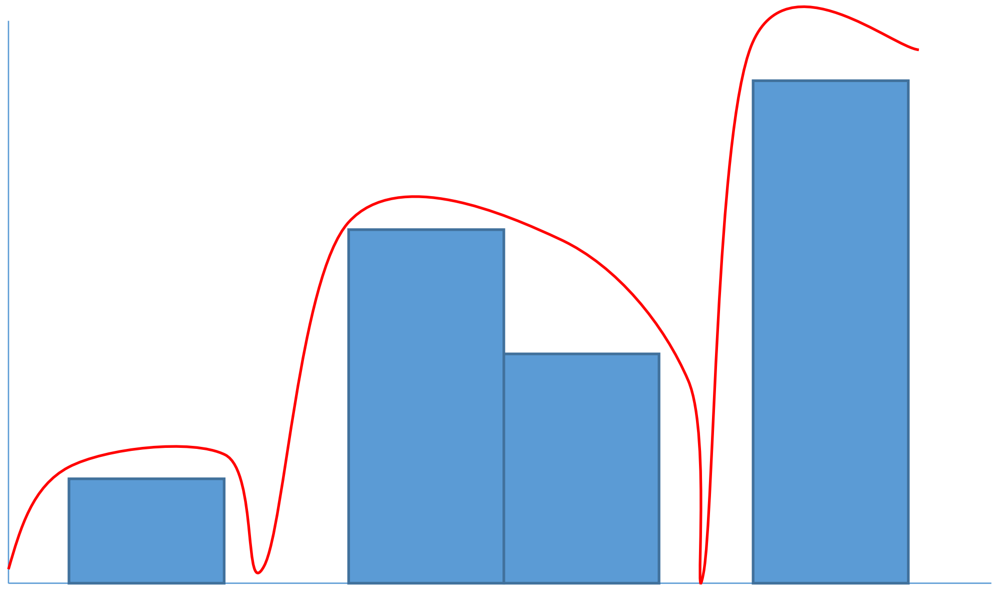
- Quickly implement Custom APIs:
  - Publish to External, Partner and Internal developers
  - Extend Operations for data and services:
    - Each API can have 1 or more operations
  - API Apps can be integrated into Logic App workflows

Mobile Apps


Native SDKs
Windows
iOS
Android
X-Plat SDKs
HTML5/JS
Xamarin
Phonegap
Sencha
REST API
Offline Sync
Data
Office365
Salesforce
Dynamics
SQL
Tables
Mongo DB
On-premises
User Authentication
Facebook
Twitter
Microsoft
Google
Azure Active Directory
Push Notifications
iOS OSX
Android Chrome
Windows
Kindle
In-app
Node.js
Express
.NET
Web API

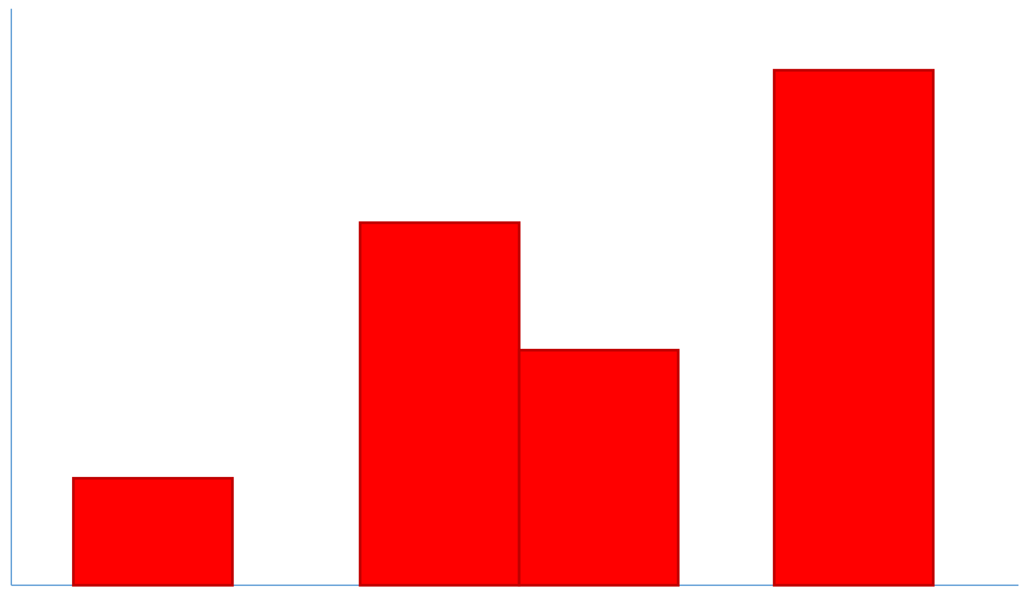
Serverless Processing

- Many compute tasks vary in their consumption:
  - Sometimes the compute units are unused or unnecessary
  - The compute must be around to respond to bursts in usage
- Traditional compute models use auto-scale and metrics to try and better prioritize compute spend based on utilization:
  - The compute spend never “exactly” matches the utilization



Serverless Services

Serverless refers to services that offer consumption-based billing where you only pay for services you consume



Azure Functions

- Azure Functions:
  - Build on WebJobs Technology
  - Available in Consumption and App Service Plan billing modes
  - Can be deployed using Scripts or Pre-Compiled
  - Managed and Edited directly in the portal:
    - Supports CI from GitHub or VSO if preferred

Azure Functions Features

- Functions support a wide variety of programming languages:
  - C#
  - F#
  - Node.js
  - Java
  - PHP
  - Python
- Functions also support scripting languages:
  - Bash
  - PowerShell

Event-Based Triggers

- Azure Functions features no-code triggers that can invoke a function based on changes in the following services:
  - Azure:
    - Storage Blobs
    - Cosmos DB
    - Storage Tables
    - Mobile Apps
    - Office 365 Files
  - Third-Party:
    - Twilio
    - SendGrid

Messaging Triggers

- Azure Functions can also integrate into existing workflows that use Azure messaging services:
  - Service Bus
  - Event Grid
  - Storage Queues
  - Event Hubs

Integration

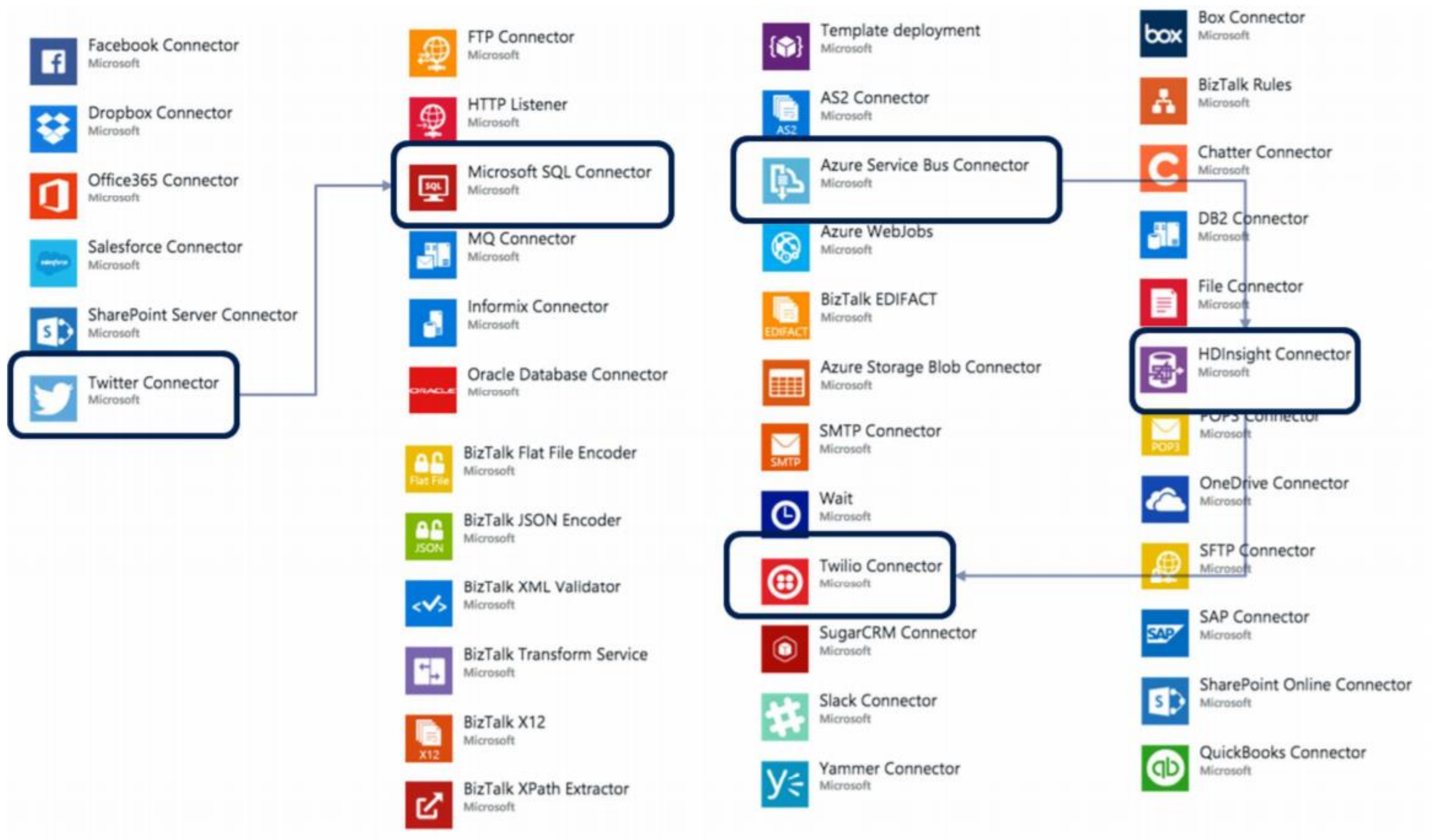
API Management

- API Proxy Service:
  - Protect your API Endpoints
  - Add billing, throttling, monitoring and policies to existing APIs without changing their code
  - Manipulate requests and responses in-flight to meet business requirements

Logic Apps

- Automation workflow solution:
  - No-code designer for rapid creation of integration solutions
  - Pre-built templates to simplify getting started
  - Out-of-box support for popular SaaS and on-premises integrations
  - BizTalk APIs available to advanced integration solutions
- JSON-based workflow definition:
  - Can be deployed using ARM templates

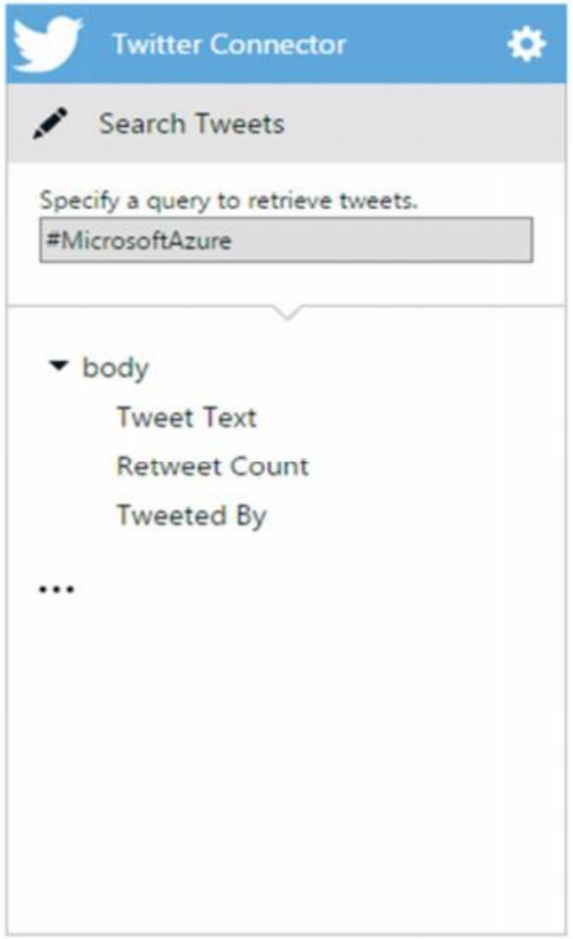
Logic App Connectors



Logic App Components

- Workflow:
  - The business process described as a series of steps
- Triggers:
  - The step that invokes a new workflow instance
- Actions:
  - A individual step in a workflow, typically a Connector or custom API App
- Connectors:
  - A special case of an API App that is pre-built and ready to integrate with a specific service or data source:
    - For example: Twitter and SQL Server Connectors

Example Logic App


Twitter Connector

Search Tweets

Specify a query to retrieve tweets.

#MicrosoftAzure

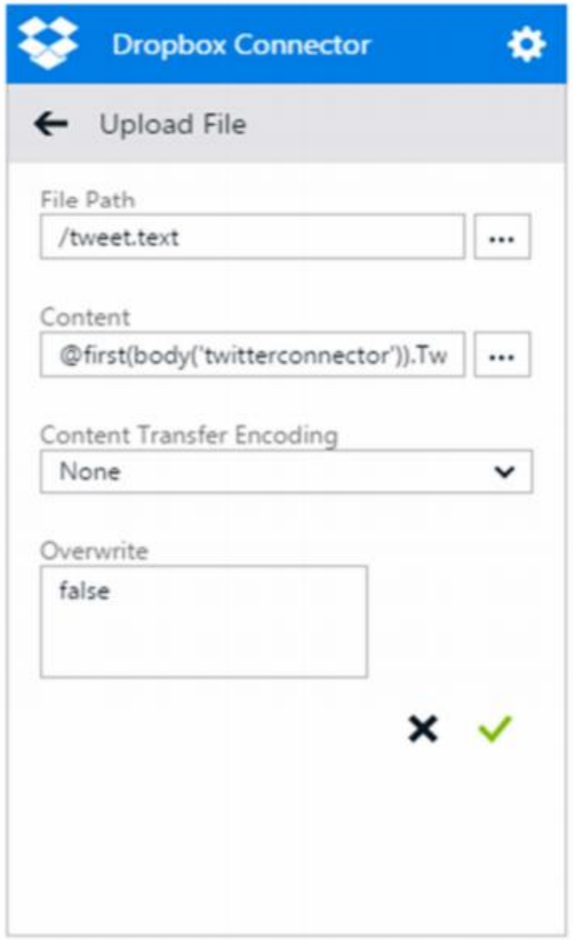
body

Tweet Text

Retweet Count

Tweeted By

...


Dropbox Connector

Upload File

File Path: /tweet.text

Content: @first(body('twitterconnector')).Tw

Content Transfer Encoding: None

Overwrite: false

API Apps in Logic Apps

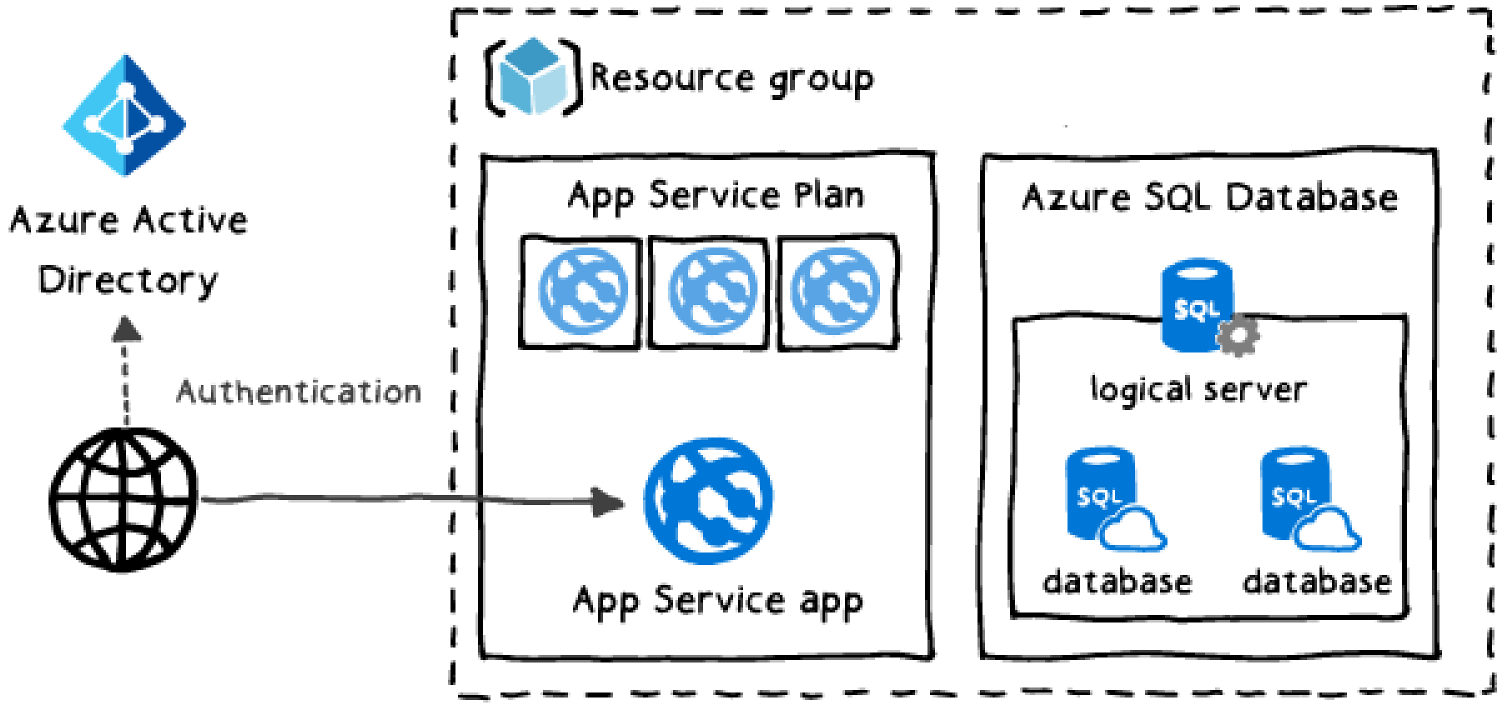
Any API App can be used as a first-class **Action** in a Logic App **Workflow**

High Performance Hosting

Best Practices

- In a multi-tier model create each tier in a separate App Service Tier
- Create Deployment slots in separate App Service Tiers
- Make use of the VNET Integration with service end-points to SQL Databases and Storage Accounts
- Distribute your application across Azure regions and use Traffic Manager to manage it

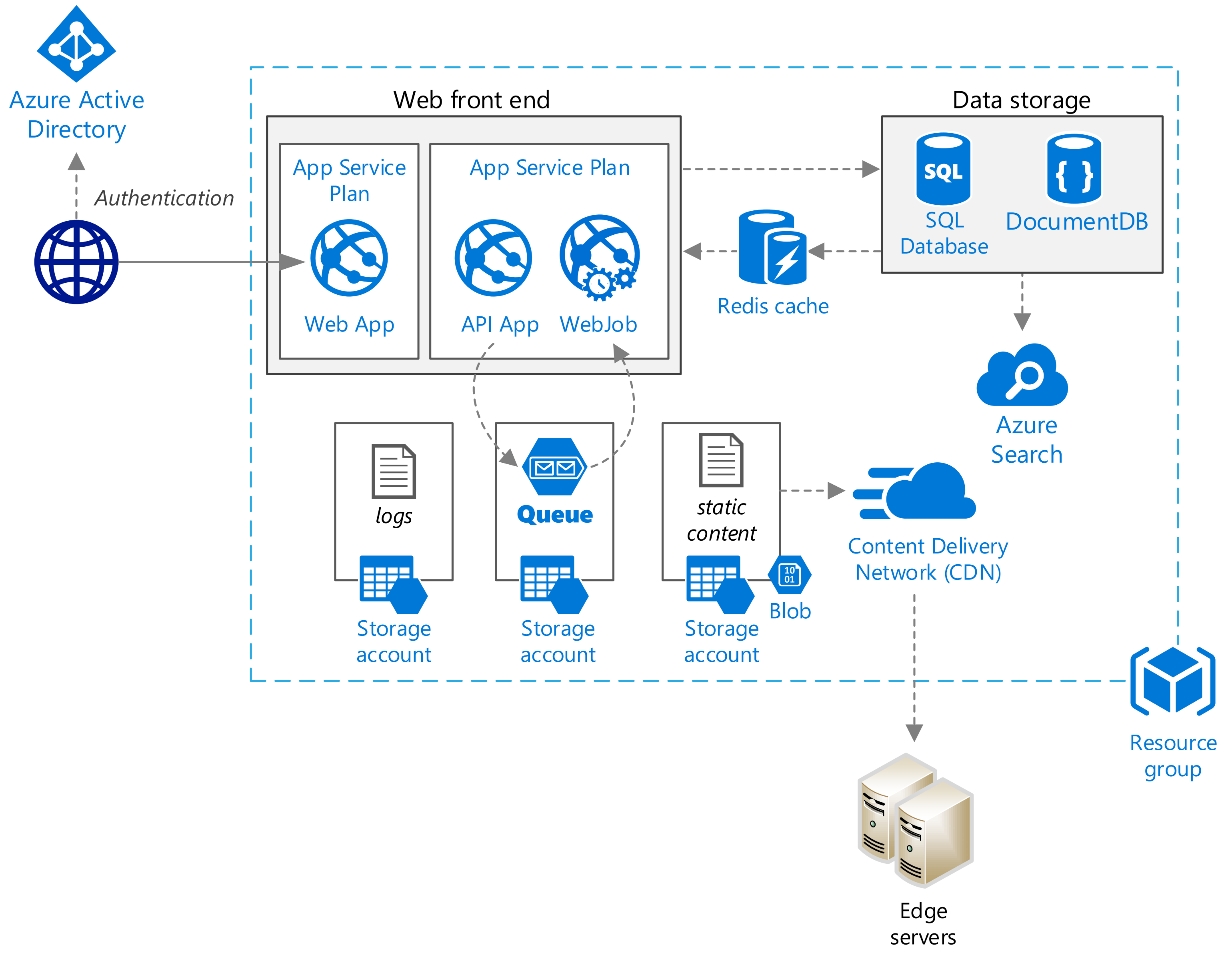
Basic Web Application



Scaling

- Horizontal scaling allows you to create a multi-instance app
- Performance and reliability
- Have always at least 2 instances
- App Service can scale up to 100 Instances:
  - Basic tier: up to 3 instances (only manual scaling)
  - Standard tier: up to 10 instances
  - Premium and PremiumV2 tier: up to 20 instances
  - Isolated tier: up to 100 instances

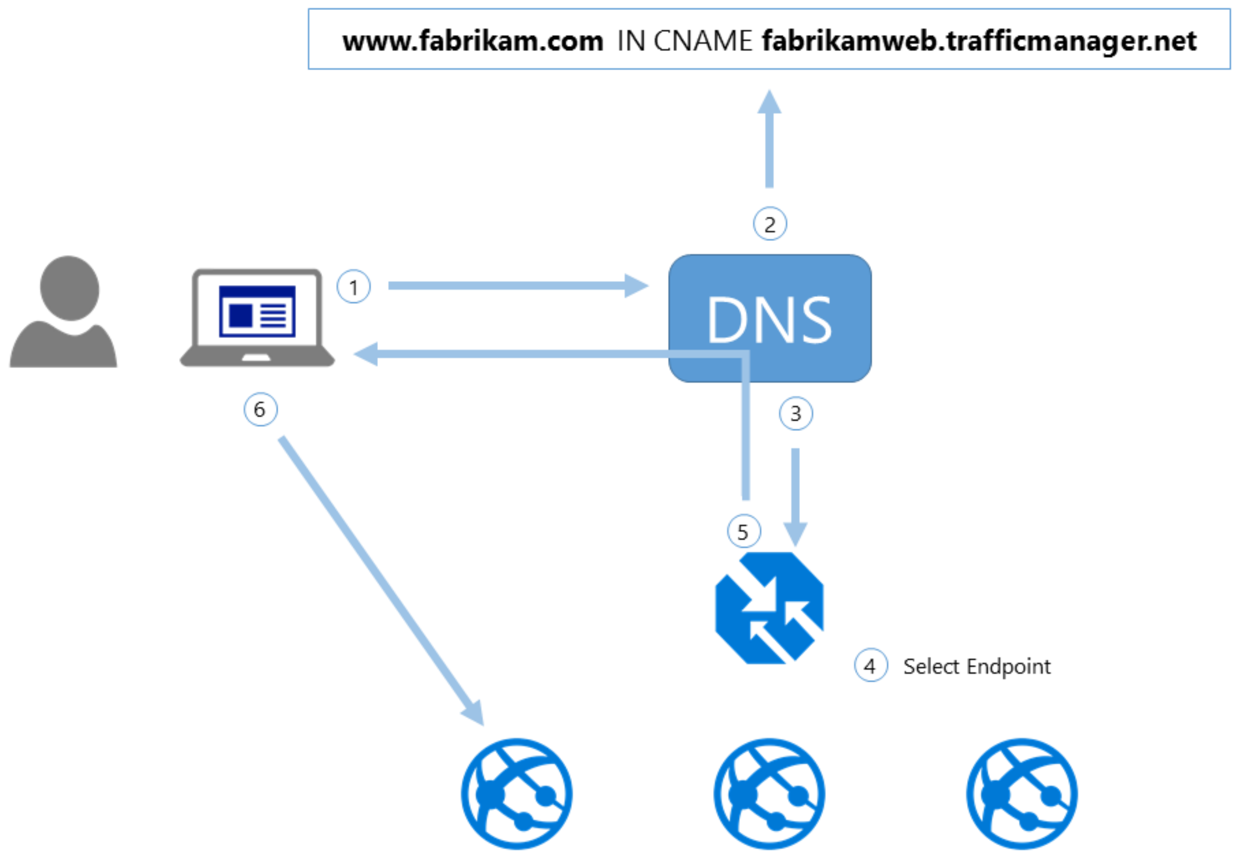
Scaling


Azure Active Directory
Authentication
Web front end
App Service Plan
Web App
App Service Plan
API App
WebJob
Data storage
SQL Database
DocumentDB
Redis cache
Azure Search
logs
Storage account
Queue
Storage account
static content
Storage account
Blob
Content Delivery Network (CDN)
Resource group
Edge servers

Traffic Manager

- Allows to control user traffic distribution
- Endpoints can include:
  - App Services
  - Cloud Services (Legacy)
  - Other endpoints (even on-premises with internet connection)
- Can be used in several modes:
  - Failover
  - Geography
  - Distribution

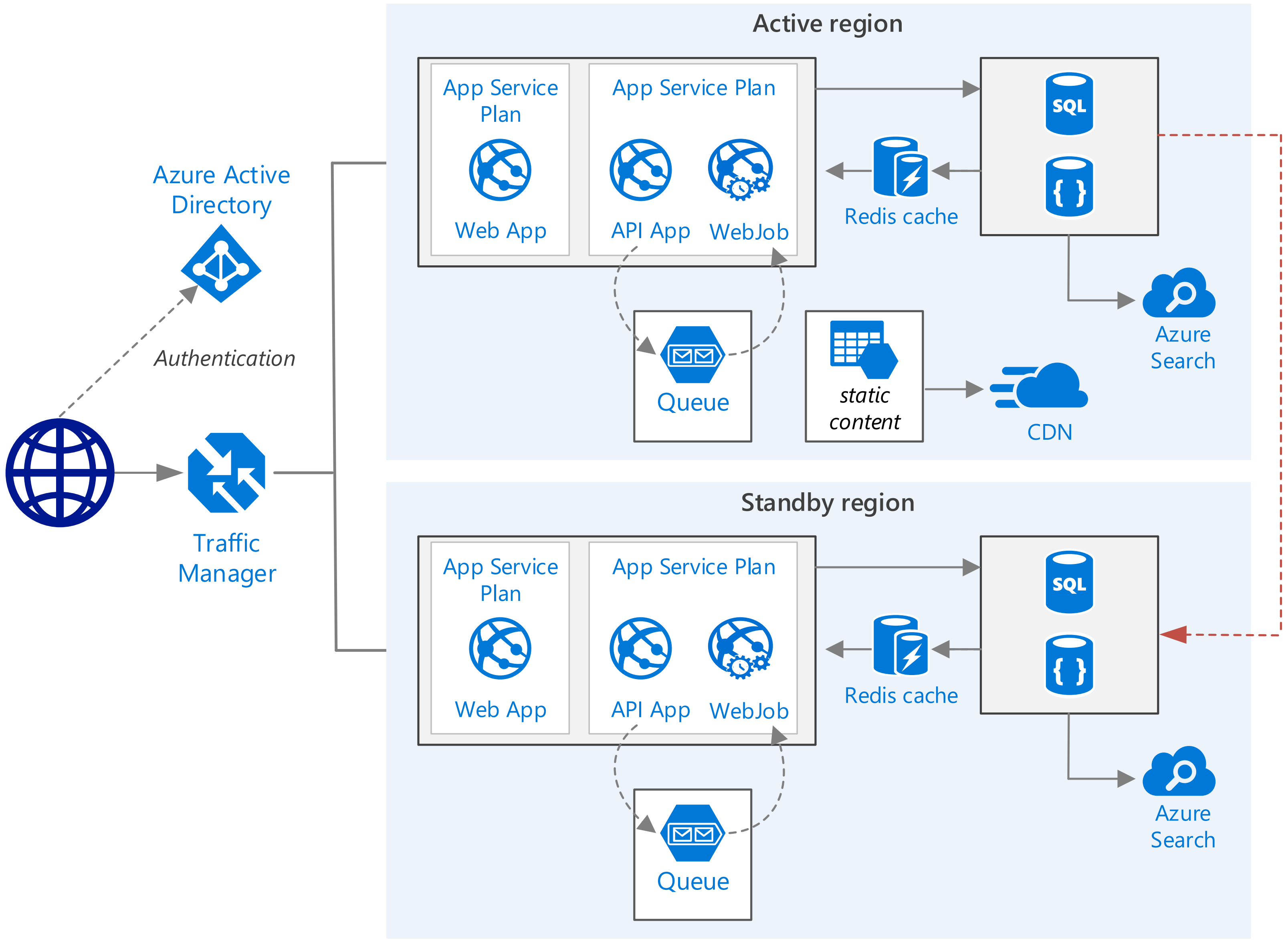
Traffic Manager



Multi-Region Model

- Primary and secondary regions:
  - Choose regions from the same regional pair (Example: East US 2 and Central US)
- Azure Traffic Manager
- Geo-replication of data (SQL Database or/and Cosmos DB)

Multi-Region Model



Application Architecture Patterns

Why Patterns?

- Most problems are already solved by other professionals in the industry
- Years of experience and research have defined best practices and the “best practice” when developing new solutions
- Most patterns are platform-agnostic

Microsoft Patterns & Practices

- Engineering focused group at Microsoft
  - Collects real-world scenarios from customers
  - Engineers solutions using best-practices
  - Analyze trends and services used by the community
  - Shares findings using
    - GitHub
    - Whitepapers
    - Conference Sessions
    - http://docs.microsoft.com

Cloud Design Patterns

- The Cloud Design Patterns is a collection of developer-oriented prescriptive architecture guidance

- Includes topics and patterns that help you design your cloud solutions

- Patterns are platform and framework-agnostic

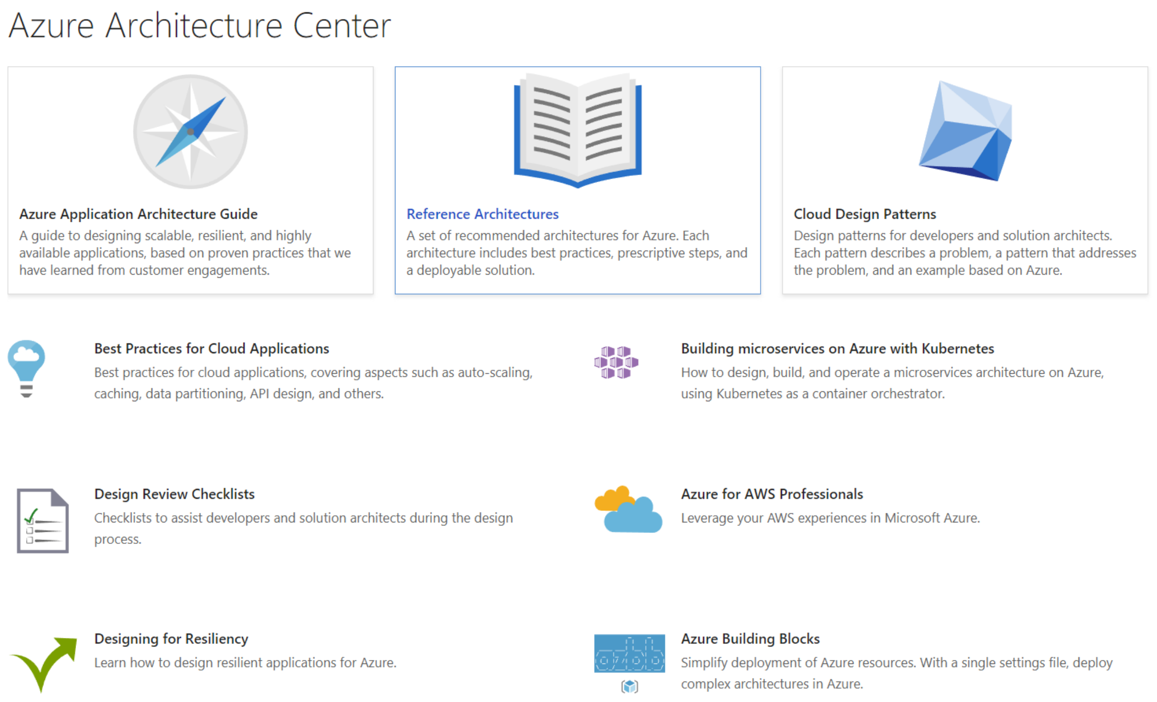
- Examples are provided in the context of Azure and C#

Patterns & Practices on GitHub

- Microsoft Patterns & Practices shares much of their documentation, projects and findings today on GitHub:
  - https://github.com/mspnp
- For example, the Microservices Reference Implementation shares best practices when designing a microservices solution running on Azure using Kubernetes:
  - https://github.com/mspnp/microservices-reference-implementation

Azure Architecture Center

Landing page for reference architectures, patterns and guidance for solutions on the Azure Platform



Azure Architecture Center

Azure Application Architecture Guide
A guide to designing scalable, resilient, and highly available applications, based on proven practices that we have learned from customer engagements.

Reference Architectures
A set of recommended architectures for Azure. Each architecture includes best practices, prescriptive steps, and a deployable solution.

Cloud Design Patterns
Design patterns for developers and solution architects. Each pattern describes a problem, a pattern that addresses the problem, and an example based on Azure.

Best Practices for Cloud Applications
Best practices for cloud applications, covering aspects such as auto-scaling, caching, data partitioning, API design, and others.

Building microservices on Azure with Kubernetes
How to design, build, and operate a microservices architecture on Azure, using Kubernetes as a container orchestrator.

Design Review Checklists
Checklists to assist developers and solution architects during the design process.

Azure for AWS Professionals
Leverage your AWS experiences in Microsoft Azure.

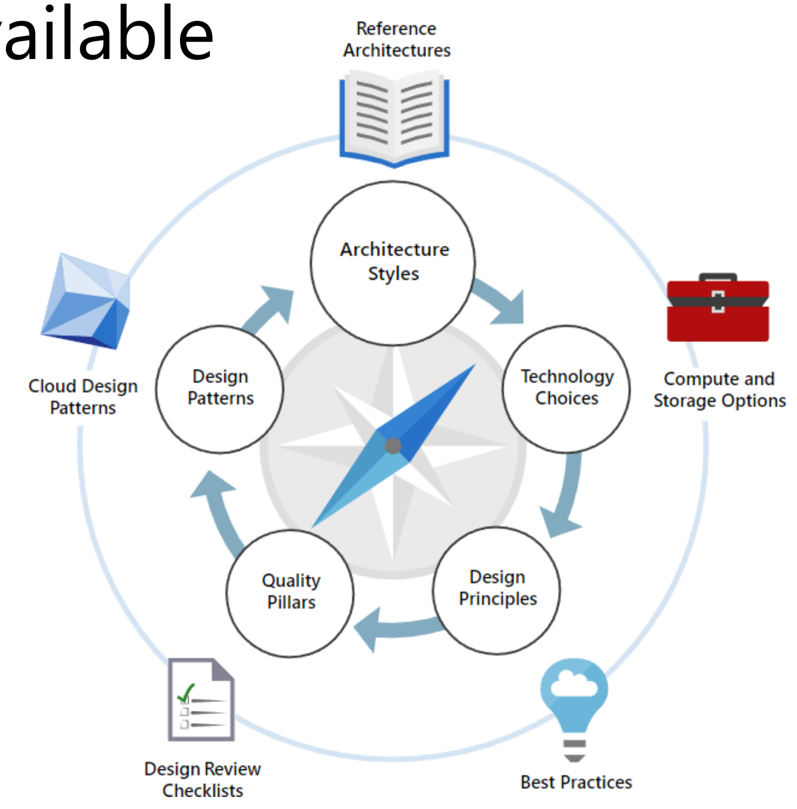
Designing for Resiliency
Learn how to design resilient applications for Azure.

Azure Building Blocks
Simplify deployment of Azure resources. With a single settings file, deploy complex architectures in Azure.

https://docs.microsoft.com/azure/architecture/

Azure Architecture Center Guide

The Architecture Center features an all-up guide to creating solutions that are scalable, resilient and highly available



https://docs.microsoft.com/azure/architecture/guide/

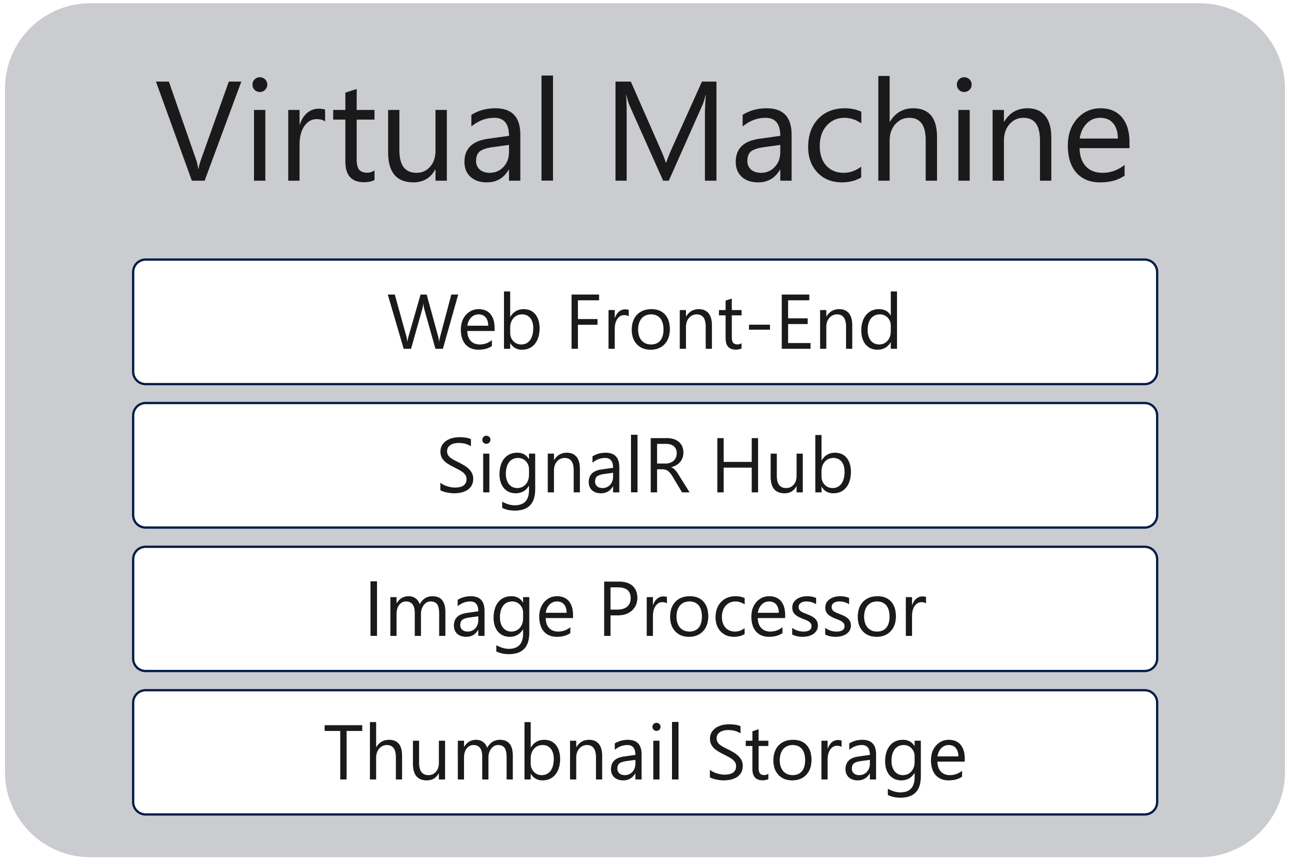
Performance Patterns

Stateless Applications

- When designing web applications, split your business processes into Partitioning Workloads
- Partitioning Workloads:
  - Can be handled in modular websites, cloud services, or virtual machines
  - Provides the ability to scale the different components of your application in isolation

Partitioning Workloads

- Example Photo Sharing Application

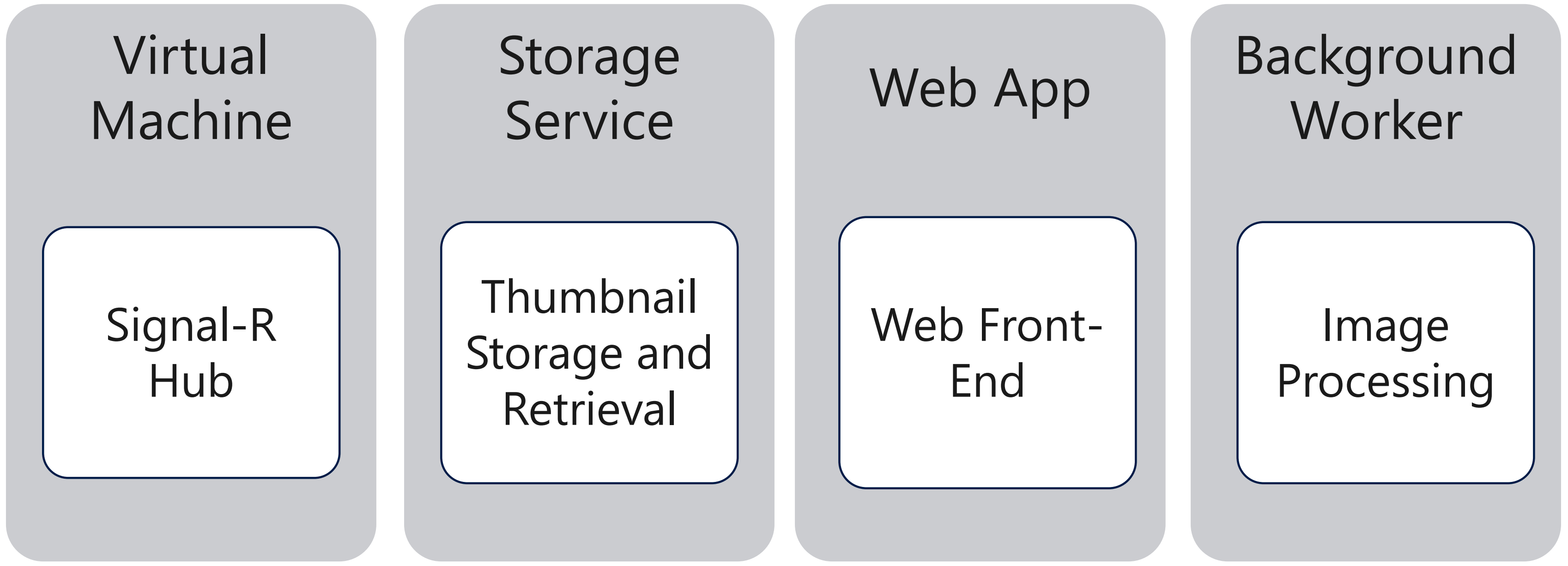


Partitioning Workloads

- Example Photo Sharing Application
  - The Web Front-End shows thumbnails of images users have uploaded today
  - The application code takes an image that a user uploads, processes it into a thumbnail and then stores the thumbnail
  - The SignalR hub notifies the client web browser that the image is finished processing

Partitioning Workloads

- Example Photo Sharing Application



- Each component of the application can be scaled in isolation to meet its specific demand

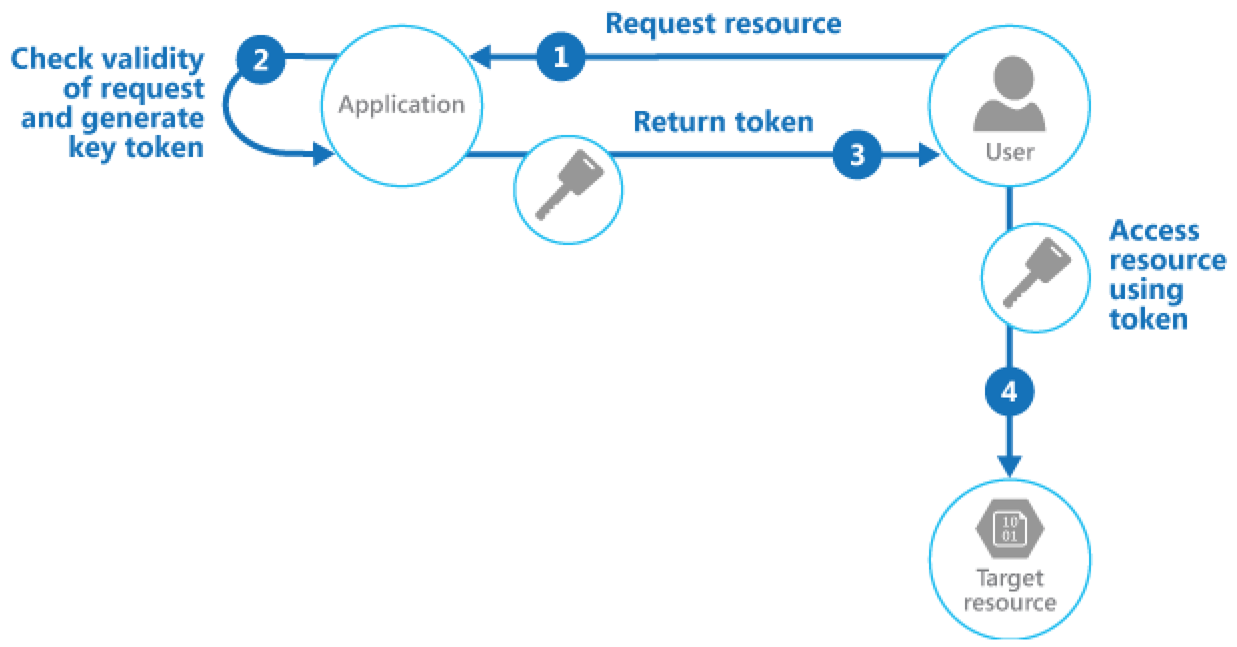
The Valet Key Pattern

- If your application access a resource on behalf of your clients, your servers take on additional load

- You do not want to make your resources publicly available so you are typically forced to have your application validate a client

- The Valet Key pattern dictates that your application simply validates the client and then returns a token to the client. The client can then retrieves the resource directly using its own hardware and bandwidth

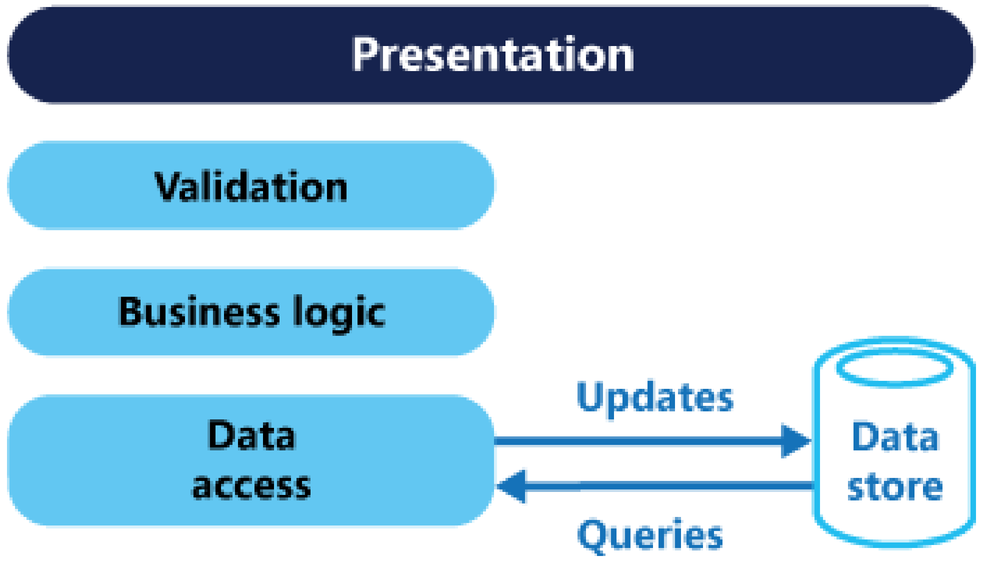
The Valet Key Pattern


Check validity of request and generate key token
2
Application
1
Request resource
Return token
3
User
Access resource using token
4
Target resource

- The client requests a resource from your application
- Your application validates the client and then returns an access token
- The client then directly accesses the resource by using the provided token

CQRS Pattern

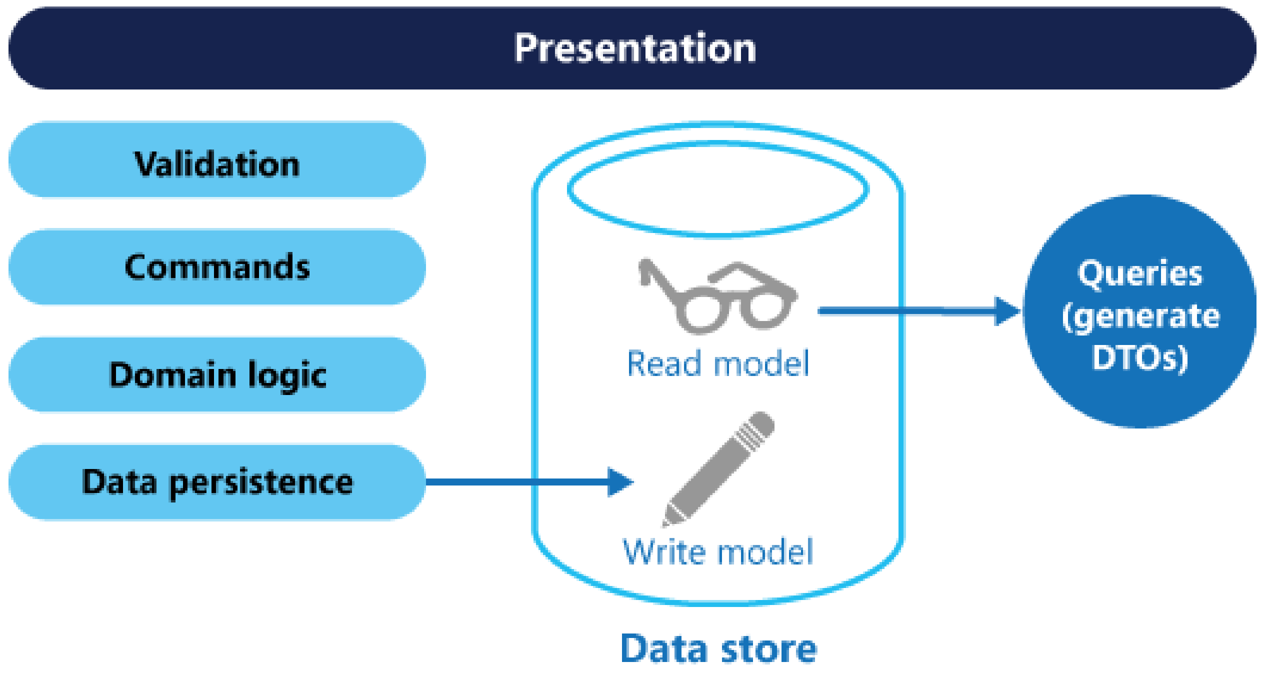
The Problem



- Potential mismatch between read and write
- Data Access layer contains largest amount of requests (for database, identity and security)

CQRS Pattern

- Solves problem by separating read and write models
  - Each model can be managed in isolation
  - Models can be deployed to separate compute instances
    - Example: A write-heavy workload will need more write instances



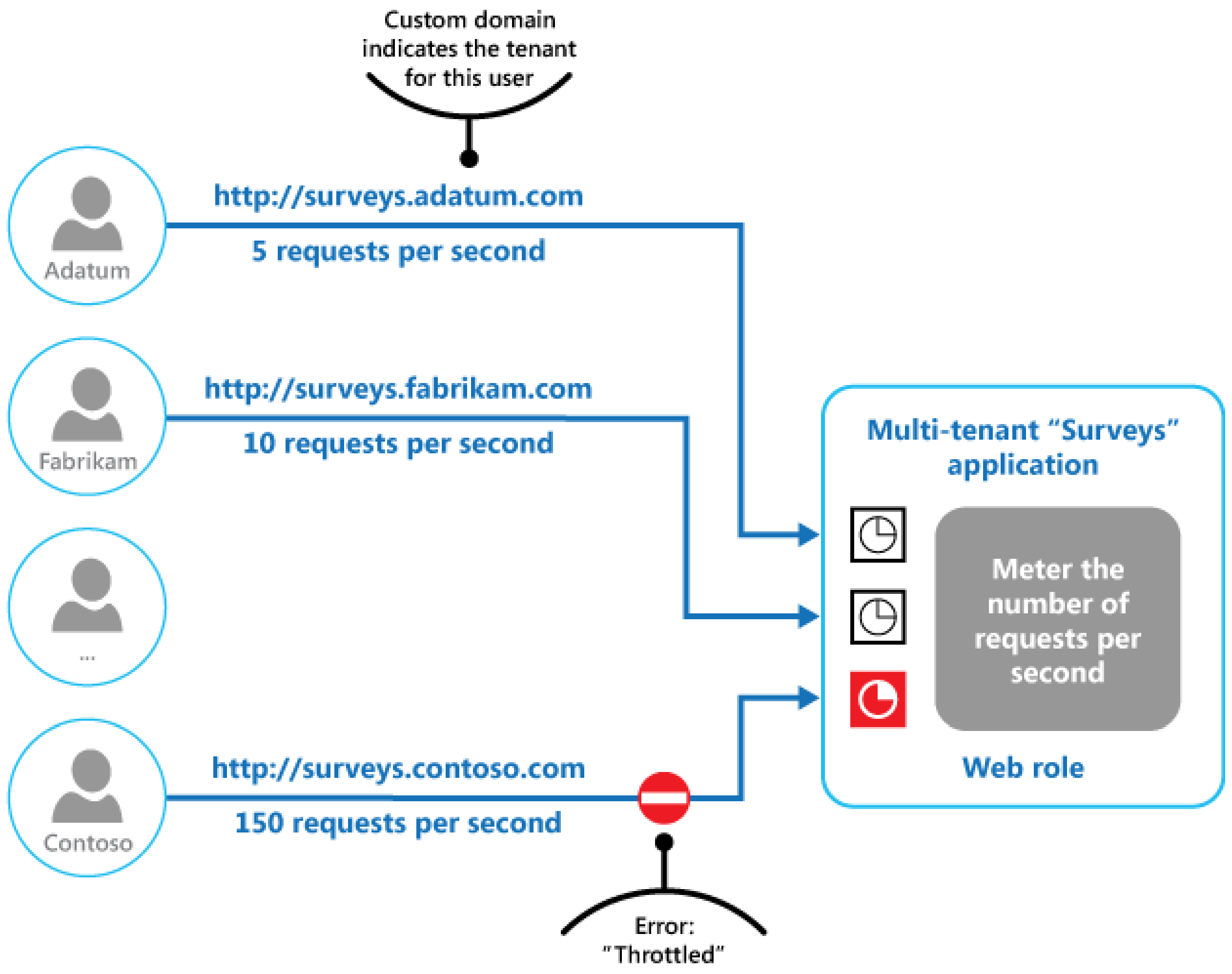
Throttling Pattern

The Problem:

- Load on a cloud application varies:
  - By Time
  - By Quantity of Users
  - By Specific User Activities
  - By Performance of Actual Underlying Hardware
- Handling load correctly for one user/client may “starve” other clients of resources
- Solutions must be able to handle sudden bursts of usage

Throttling Pattern

- Solves problem by creating a “ceiling” on usage


Custom domain indicates the tenant for this user
Adatum
http://surveys.adatum.com
5 requests per second
Fabrikam
http://surveys.fabrikam.com
10 requests per second
...
Contoso
http://surveys.contoso.com
150 requests per second
Error: "Throttled"
Multi-tenant “Surveys” application
Meter the number of requests per second
Web role

Resiliency Patterns

Transient Errors

- Transient faults can occur because of a temporary condition
  - Service is unavailable
  - Network connectivity issue
  - Service is under heavy load
- Retrying your request can resolve temporary issues that normally would crash an application
- You can retry using different strategies
  - Retry after a fixed time period
  - Exponentially wait longer to retry
  - Retry in timed increments

Transient Fault Handling

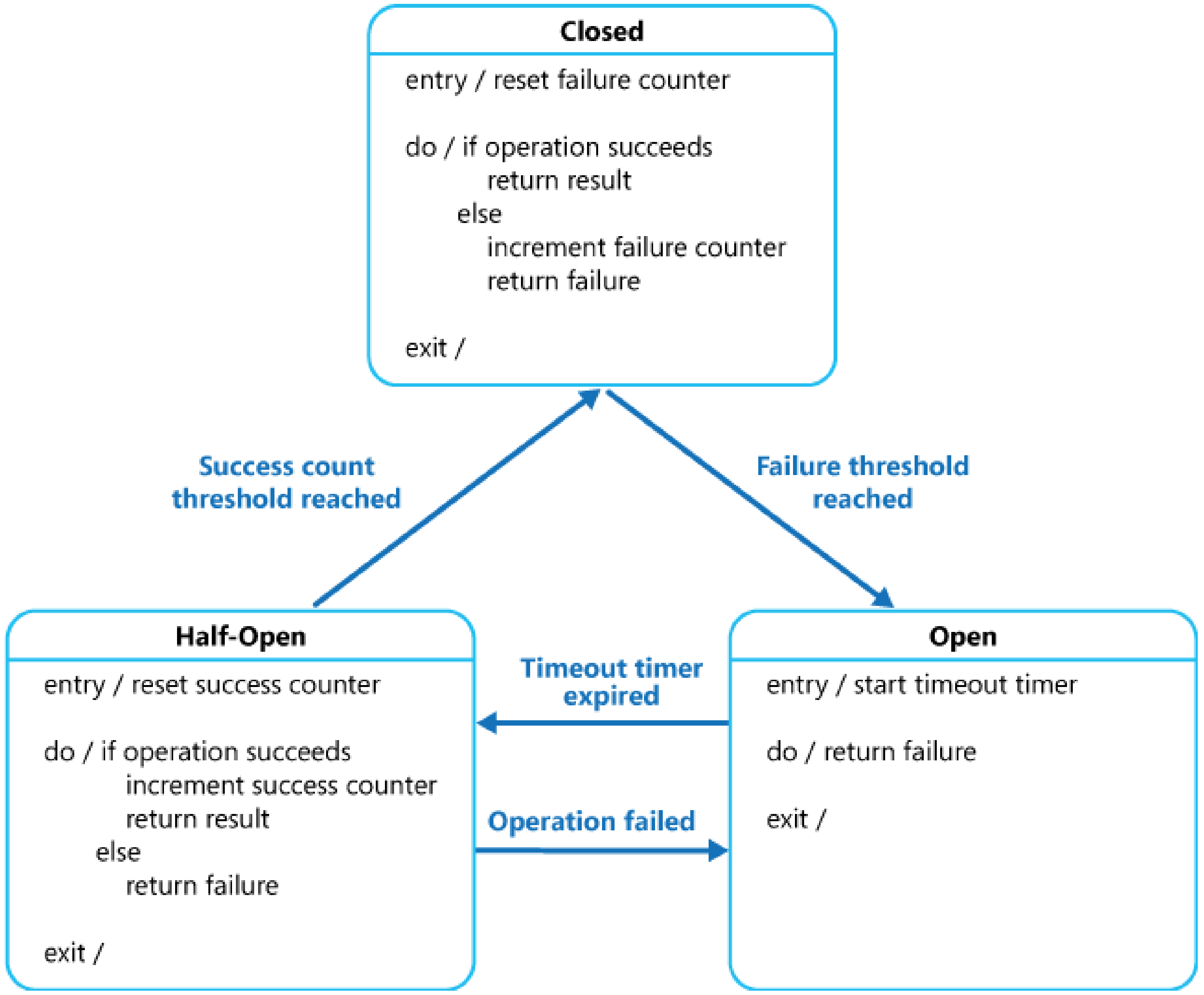
- The Transient Fault Handling application block is a part of the Enterprise Library
- The Enterprise library contains a lot of code that is necessary to implement the pattern and the retry strategies
- Retry strategies are prebuilt for common scenarios including accessing Azure services
- You can build custom strategies and extend the library for the unique needs of your application

Circuit Breaker Pattern

Prevents applications from repeatedly retrying an operation that will most like fail

- Acts as a proxy between application and service
- Prevents waste of CPU cycles on long-lasting faults

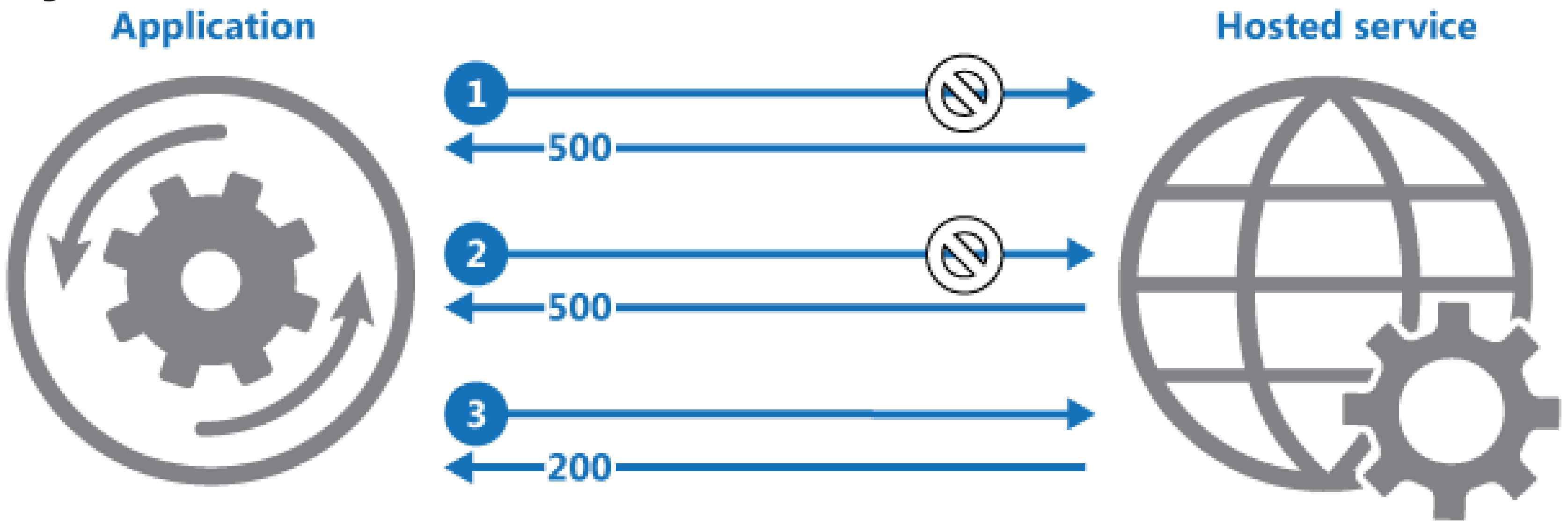
Circuit Breaker Pattern


Closed
entry / reset failure counter
do / if operation succeeds
return result
else
increment failure counter
return failure
exit /
Success count threshold reached
Failure threshold reached
Half-Open
entry / reset success counter
do / if operation succeeds
increment success counter
return result
else
return failure
exit /
Timeout timer expired
Operation failed
Open
entry / start timeout timer
do / return failure
exit /

The Retry Pattern

- The Retry pattern is designed to handle temporary failures
- Failures are assumed to be transient until they exceed the retry policy
- The Transient Fault Handling Block is an example of a library that is designed to implement the Retry pattern (and more)
- Entity Framework provides a built-in retry policy implementation
    - Implemented in version 6.0

The Retry Pattern



- The application sends a request to a hosted service
- The hosted service responds with a HTTP 500 (Internal Server Error) code
- The application retries until it exceeds the retry count for its policy or is met with an acceptable status code such as HTTP 200 (OK)

Queues

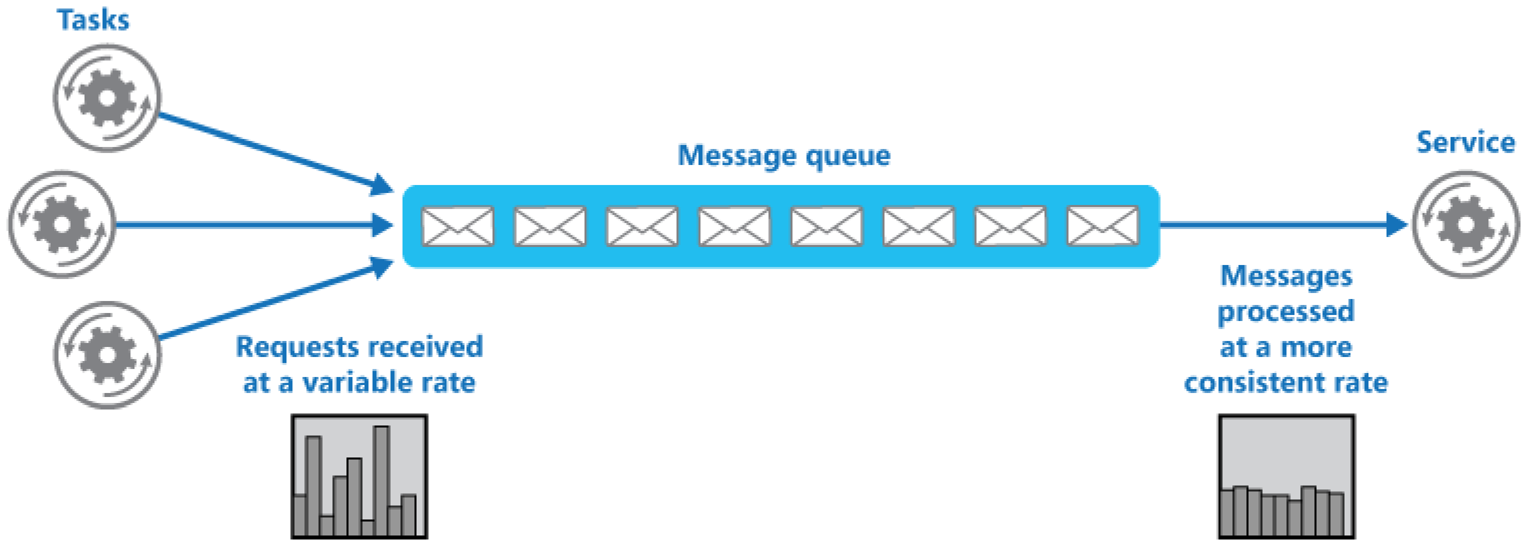
- A modular web application can behave like a monolithic application if each component relies on a direct two-way communication with a persistent connection
- Persistent queue messages allow your application to handle requests if one of your application components fail or is temporary unavailable
- An external queue allows your application to audit requests or measure your load without adding any overhead to the code of your primary application

Queues

- Queues can be used to communicate in a disconnected manner between the different components of your application
  - If an instance of your application module fails, another instance can process the queue messages
  - Messages that fail to be processed can either be reinserted into the queue or not marked as complete by the client module
  - Messages can be measured or audited in isolation from your application
- Queues become very important when dealing with background processing (Worker roles and WebJobs)

Queue-Based Load Leveling Pattern

Use a queue to act as a "buffer" between requestor generators and request services


Tasks
Message queue
Service
Requests received at a variable rate
Messages processed at a more consistent rate

Queue decouples the tasks from the service

- Service can work at it's own pace

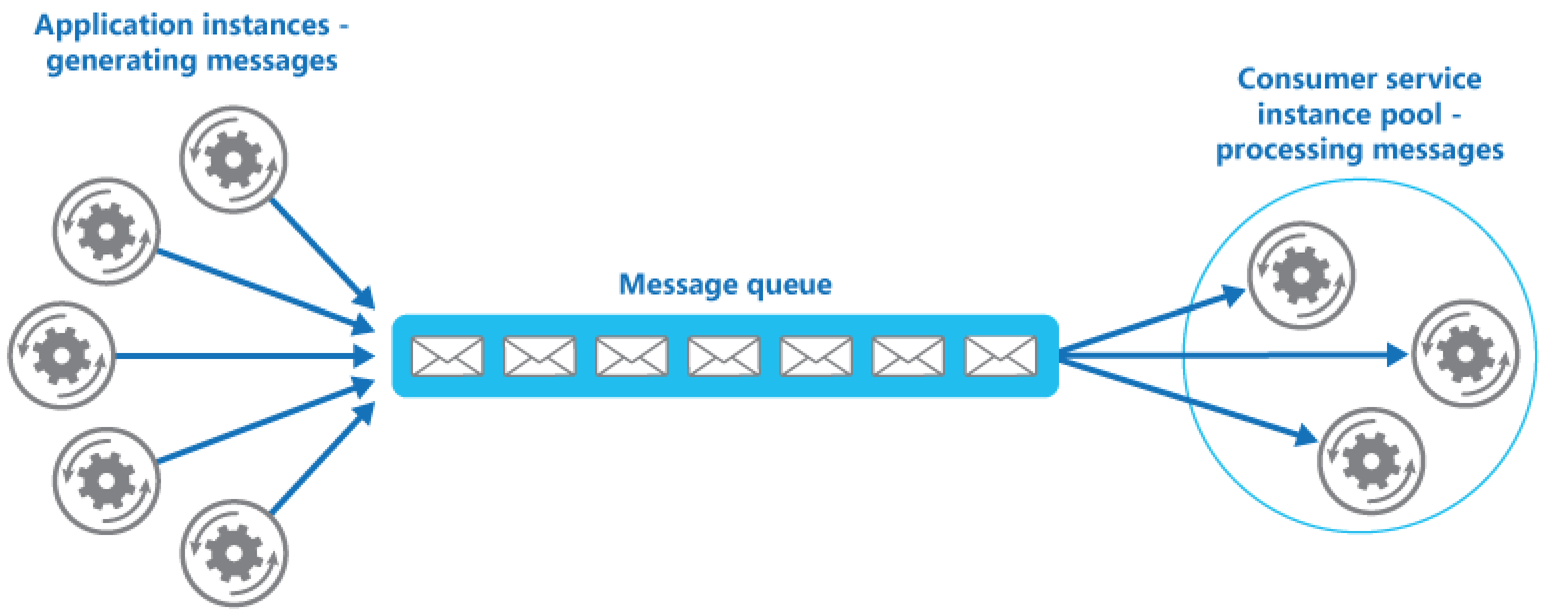
Scalability Patterns

Asynchronous Messaging

- With synchronous messaging, the service processing some logic must run immediately when requested
  - This becomes a problem with large, varying quantities of requests
  - This can prevent an application from scaling both up or down
    - Scaling down can drop requests “in flight”

Competing Consumers Pattern

- Application instances (producers) generate messages to be placed in the queue
- Service instances (consumers) poll the queue to see if any messages are waiting to be processed
- The service instance that receives the message will process the message and then flag the message as processed in the queue
  - If the service instance fails to process the message, the queue will eventually make the message available to other instances after a period of time


Application instances - generating messages
Message queue
Consumer service instance pool - processing messages

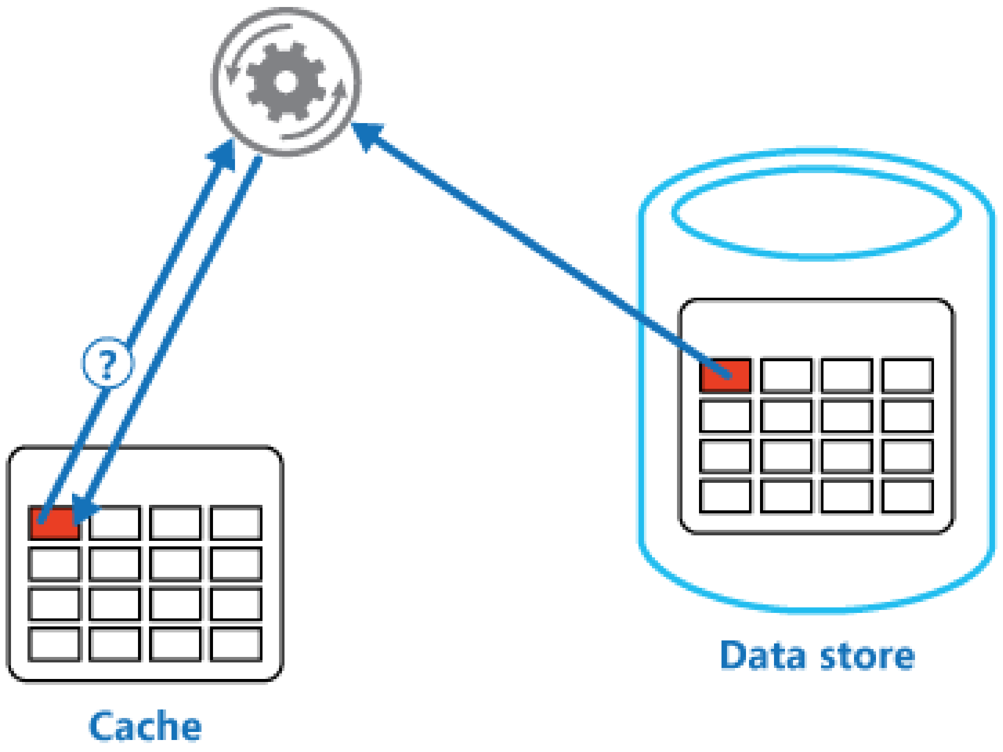
Cached Data Consistency

- Cache data can quickly become stale

- Example:
  - Application caches a list of the 10 latest records
  - Whenever a new record is added, the list is officially stale
    - Do you re-query the database constantly?
    - Do you skip using a cache to have real-time data?

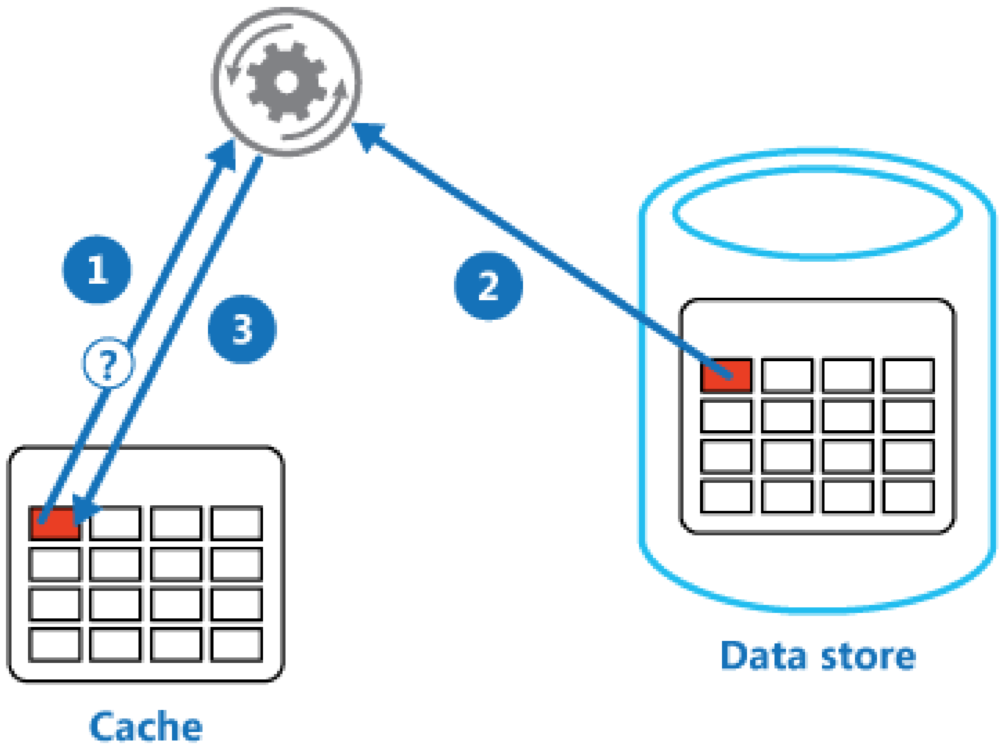
Cache-Aside Pattern

- Treats cache as a read/write store
- Ensures synced data between cache and data store



Cache-Aside Pattern

1. Determine whether the item is currently stored in cache
2. If the item is not currently stored in ache, retrieve the item from the source data store
3. Store a copy of the item in the cache

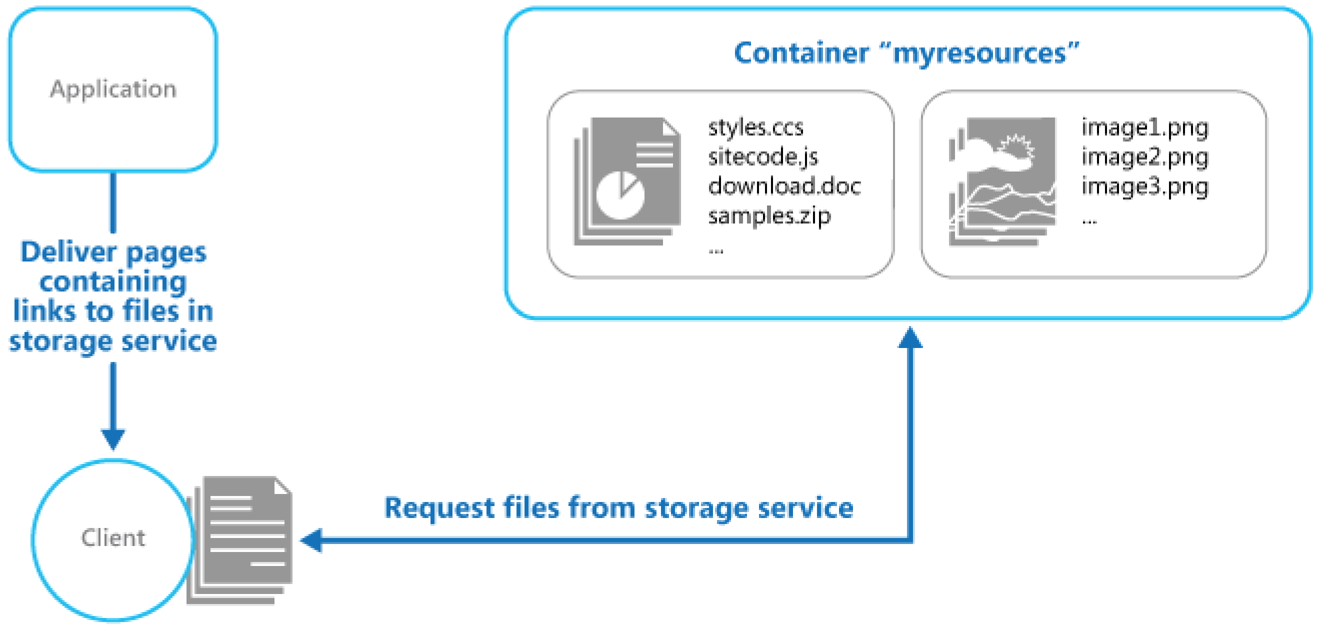


Static Content

- Static content is often hosted in a CDN to deliver content directly to clients in the most efficient manner possible
- Web servers can serve static content, but they are often not the right choice
  - Developers often serve multiple CSS, JS and HTML files as part of a single web page “visit”
- Serving static content from a CDN can save on compute and memory utilization of web servers

Static Content Hosting Pattern

- Minimizes web hosting compute costs
  - Especially so for web sites that consist only of static content
- Improves end-user content performance



Load Balancing

- Provide the same service from multiple instances and use a load balancer to distribute requests across all of the instances
- Considerations for selecting a load balancing strategy:
  - Hardware or software load balancers
  - Load balancing algorithms (round robin)
  - Load balancer stickiness
- Load balancing becomes critical even if you have a single service instance as it offers the capability to scale seamlessly

Load Balancing and Geographic Resiliency

- Load balancing can be combined with geographic redundancy to help achieve high availability
  - You can use a load balancer to direct client requests to their closest data center
    - Traffic Manager is often used for this task
  - A load balancer can be used to implement a failover scenario
    - When a data center or compute instance is down, clients can be directed to the next desirable instance that is available
- Designing a load balancing strategy along with distributing your application across data centers is key to high availability across the globe

Data Patterns

Redis Cache

- Based on the open-source Redis platform
  - Multiple tiers are available that offer different numbers of nodes
  - Supports transactions
  - Supports message aggregation using a publish subscribe model
  - Considered a key-value store where the keys can be simple or complex values
  - Massive Redis ecosystem already exists with many different clients

Database Partitioning

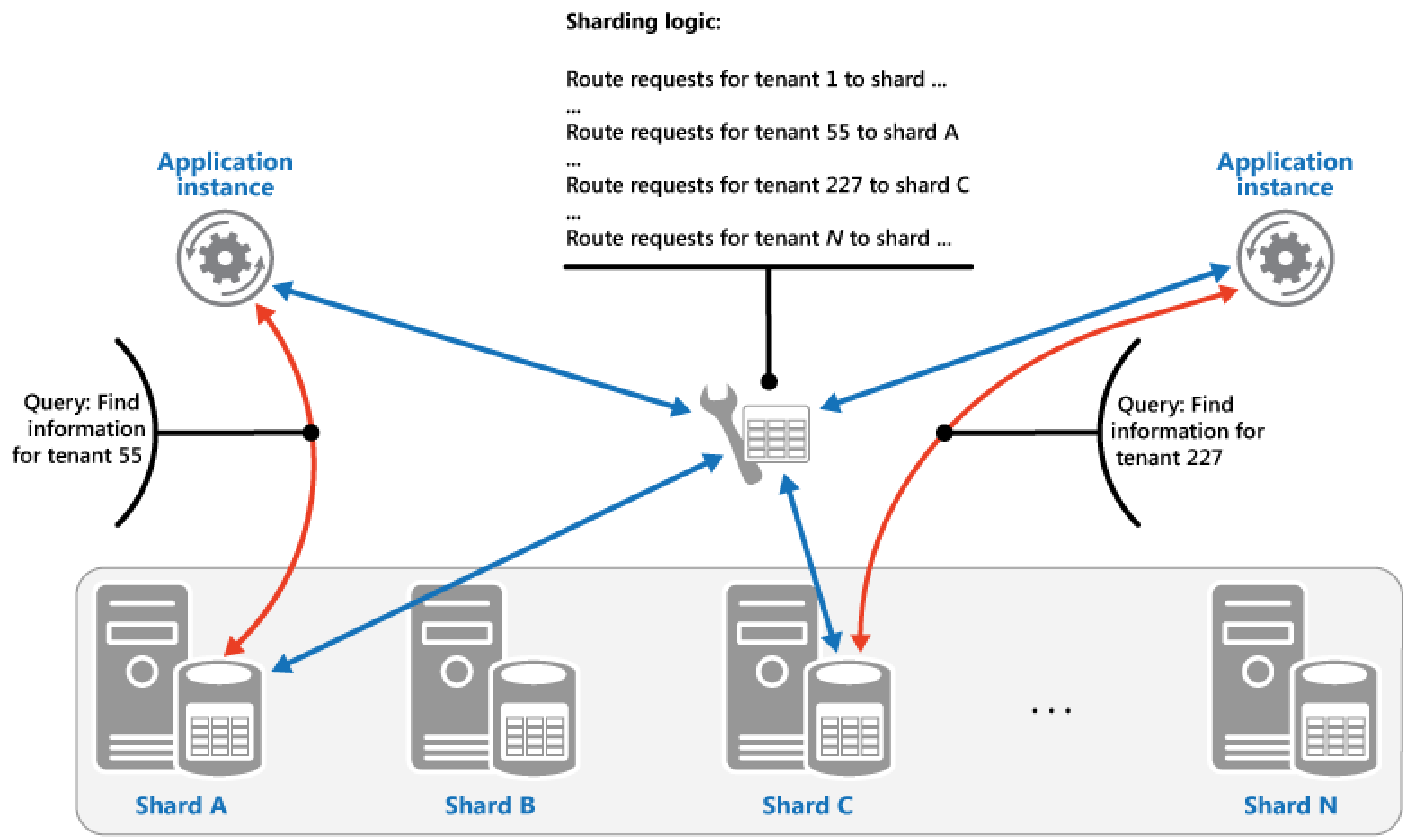
- Most cloud applications and services store and retrieve data as part of their operations
  - The design of the data stores that an application uses can have a significant bearing on the performance, throughput, and scalability of a system
- One technique that is commonly applied in large-scale systems is to divide the data into separate partitions
  - Partitioning refers to the physical separation of data for scale
  - Modern data stores understand that data may be spread across many different instances

Sharding Pattern

A map is implemented that contains lookup data mapped by shard key. With a multi-tenant application, data using the same shard key will be stored in an identical shard. In this example, the tenant ID is the shard key.

1. An application instance will make a request to the map for the shard which contains tenant #55. The map will return "Shard A".
2. The application instance will then make a request directly to the database at "Shard A" for records related to tenant #55.
3. A new application instance will make a request to the map for tenant #227 and will receive a response of "Shard C".
4. The new application instance makes a request directly to the database at "Shard C" for records related to tenant #227.
5. As new tenants are added and more space is necessary, new shards can be added to the map. Tenant IDs can than be associated with the new shards.

Sharding Pattern - Lookup

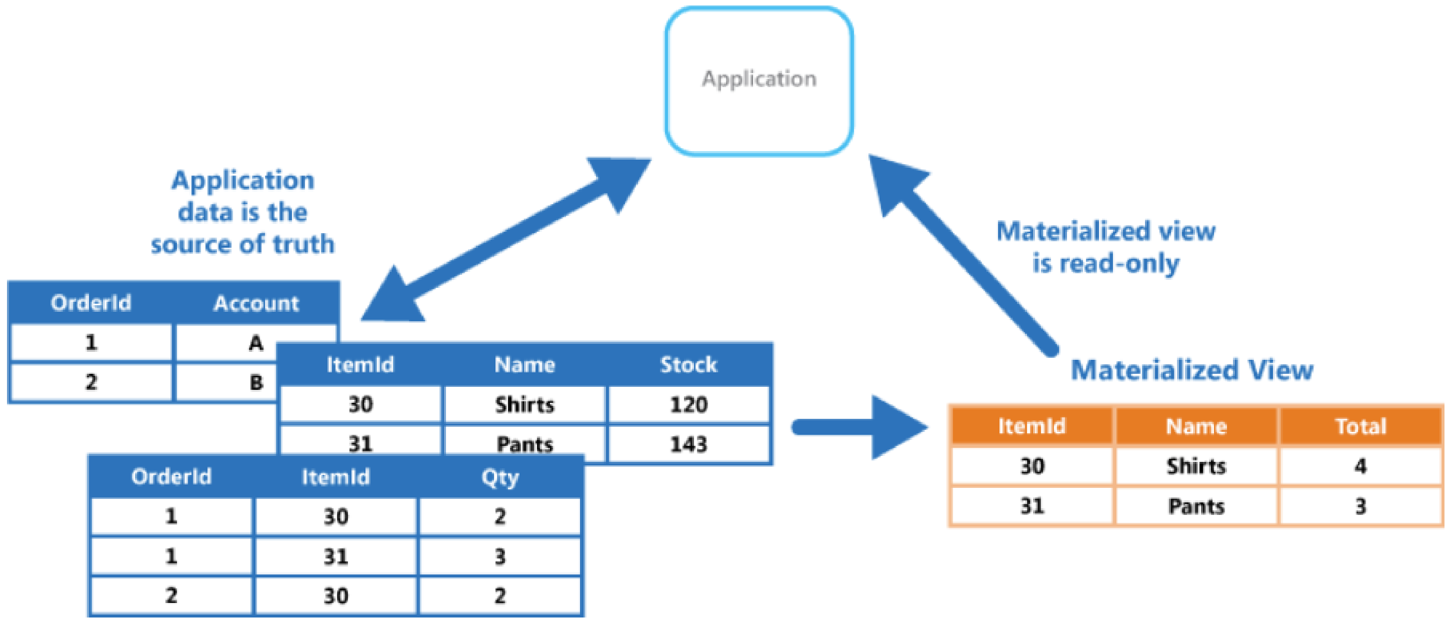

Sharding logic:
Route requests for tenant 1 to shard ...
...
Route requests for tenant 55 to shard A
...
Route requests for tenant 227 to shard C
...
Route requests for tenant *N* to shard ...
Application instance
Application instance
Query: Find information for tenant 55
Query: Find information for tenant 227
Shard A
Shard B
Shard C
. . .
Shard N

Database Query Performance

- Database Administrators and Developers focus on how data is stored not read
  - Database systems are designed for fast data writes
- Reading data can require complex queries that uses relationships between multiple collections
  - RDBMS systems are notorious for having 3+ JOINs in a single query
  - NoSQL systems make require cross-partition querying to find relevant data for a record

Materialized View Pattern

- Pre-built view of most queried data
- Automatically updated when source data is changed
- Functions as a specialized cache


Application
Application data is the source of truth
Materialized view is read-only
Materialized View

OrderId	Account
1	A
2	B

ItemId	Name	Stock
30	Shirts	120
31	Pants	143

OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

ItemId	Name	Total
30	Shirts	4
31	Pants	3

Building IaaS Applications

Azure Availability

- Azure provides money-backed SLAs for IaaS services:
  - Two Instances or more in an Availability Set = 99.95%
  - Single Instance VM using Premium Storage = 99.9%

- Decisions should be based on cost and availability requirements

Stand-Alone VMs

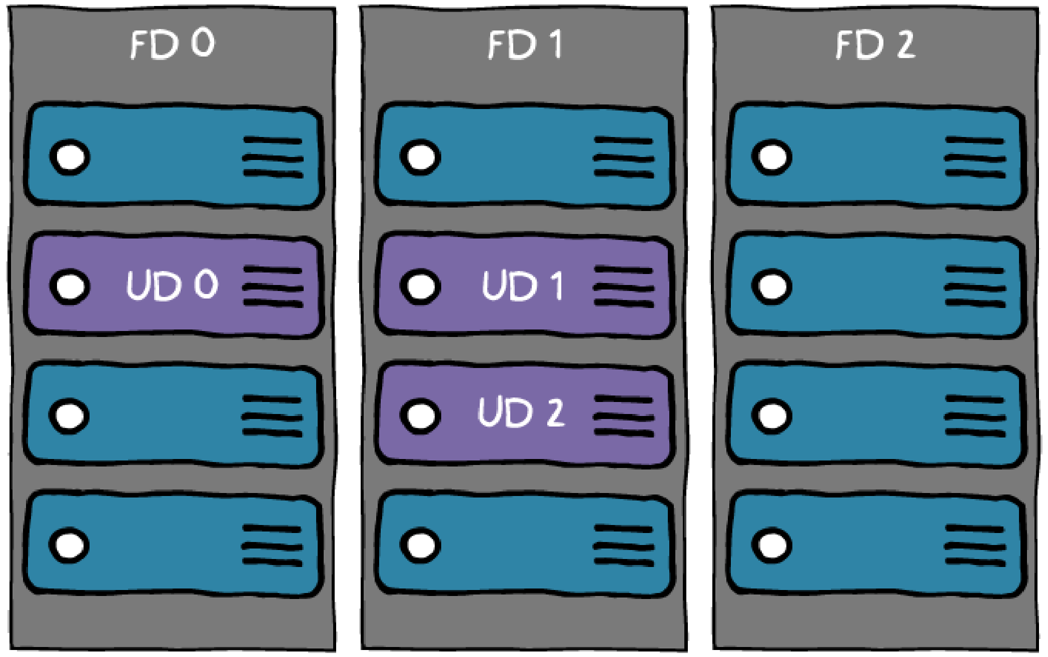
- Single instance VM would gain 99.9% SLA if it complies with:
  - Premium Storage for all Operating System Disks and Data Disks

- Any single instance VM without Premium storage receives no SLA

Availability Sets

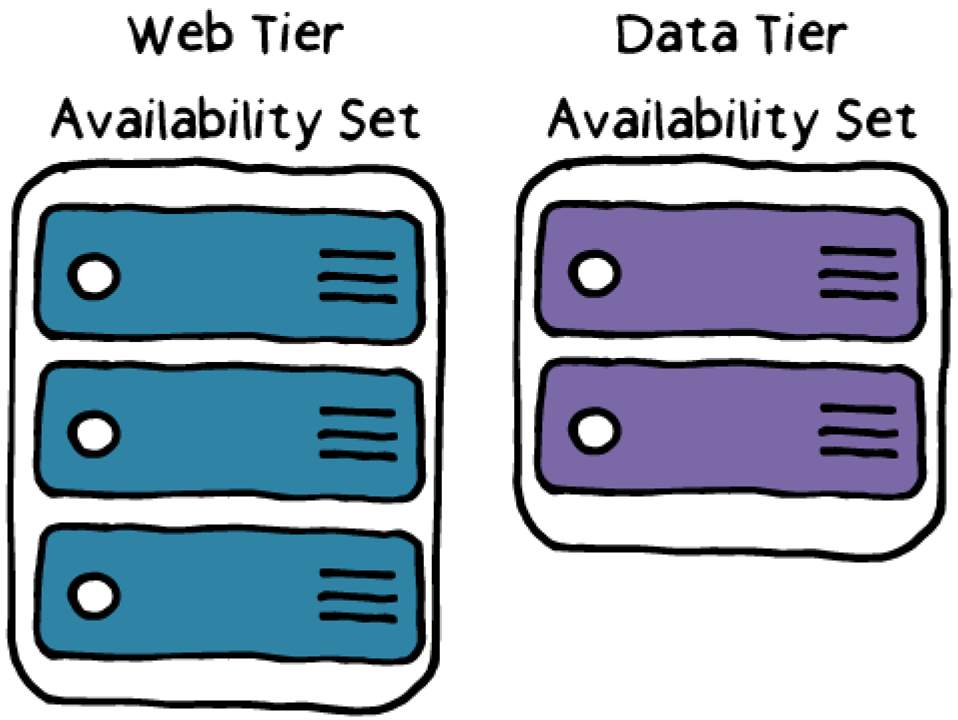
- Availability Sets provide assurance that any multiple instance VM will be available 99.95% of the time

Availability Sets cater for planned and unplanned maintenance using Update Domains and Fault Domains



Availability Sets

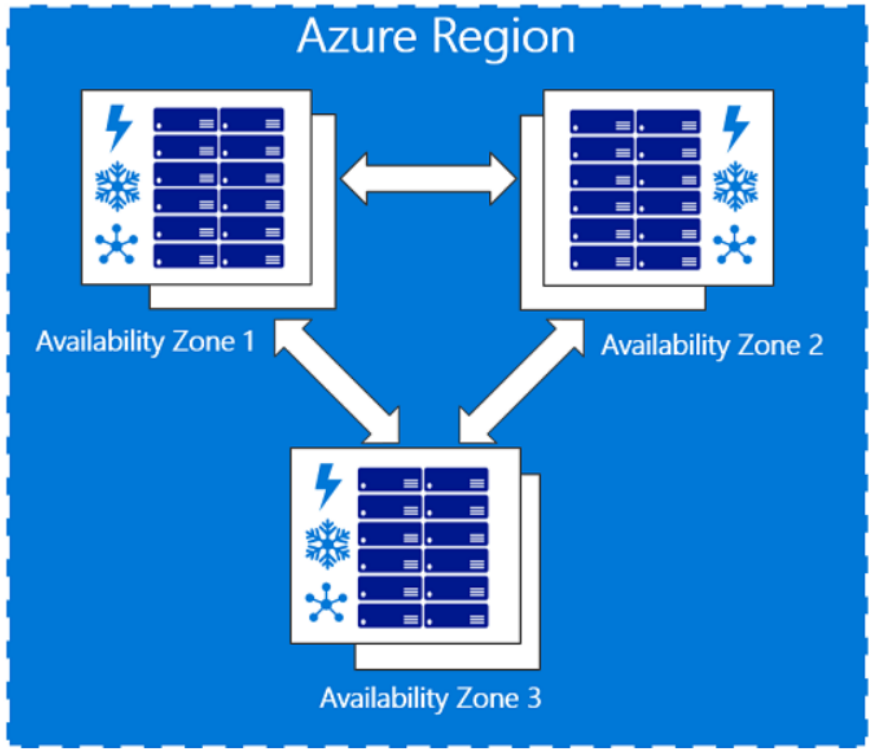
When planning multiple tier applications use multiple Availability sets, one per tier



Availability Zones

Service helps to protect resources from datacenter level failures

Provides the ability to place VMs with resilience to the loss of an entire Datacenter building. These are all located within the same Azure region



Domain Joined Virtual Machines

Domain and IaaS Applications

Azure provides a number of options for Domains:

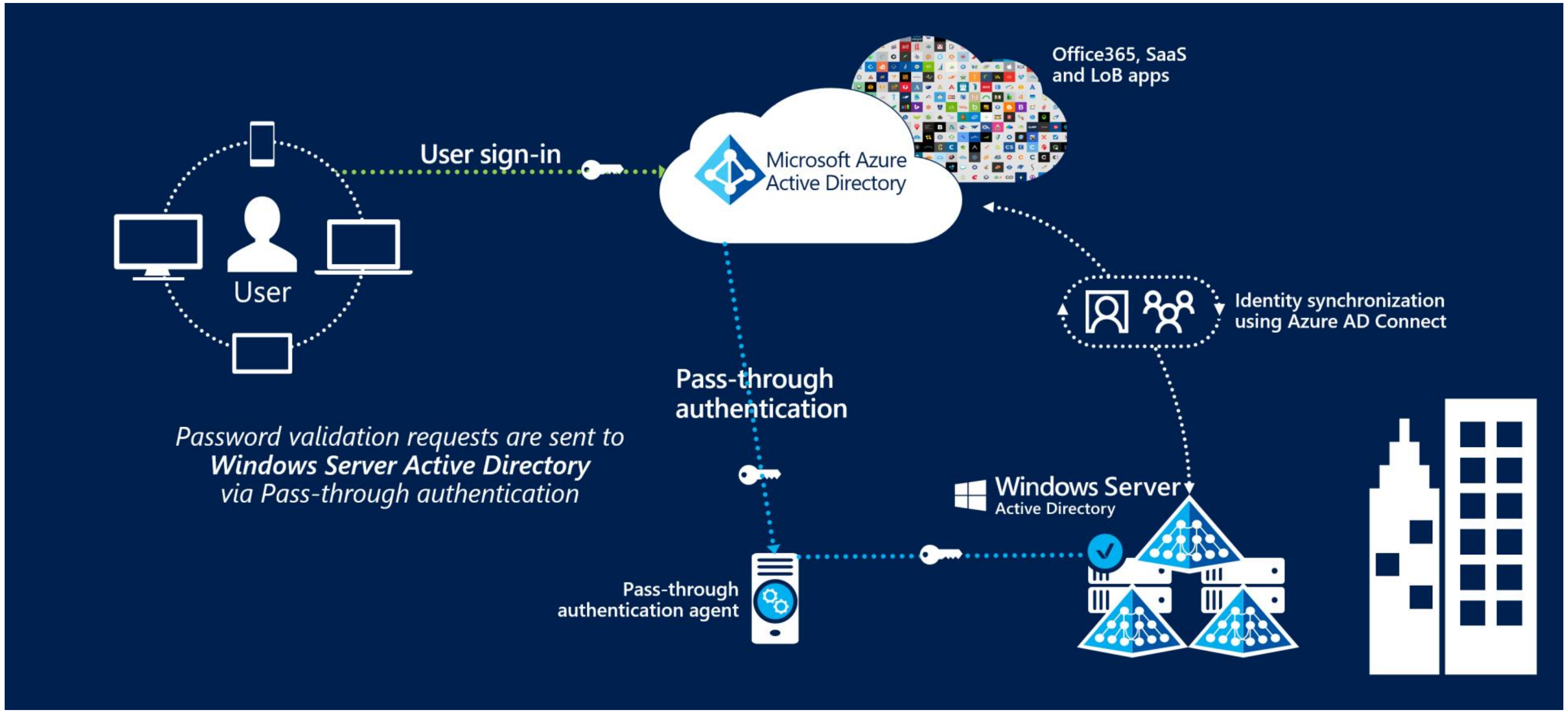
- Azure AD (and B2B, B2C)
- Hybrid ADDS And Azure AD
- Azure AD Domain Services

Hybrid Connectivity

- Azure AD Connect
- Active Directory Federation Services
- AD Connect Passthrough
- Deploy AD DS to an Azure VM

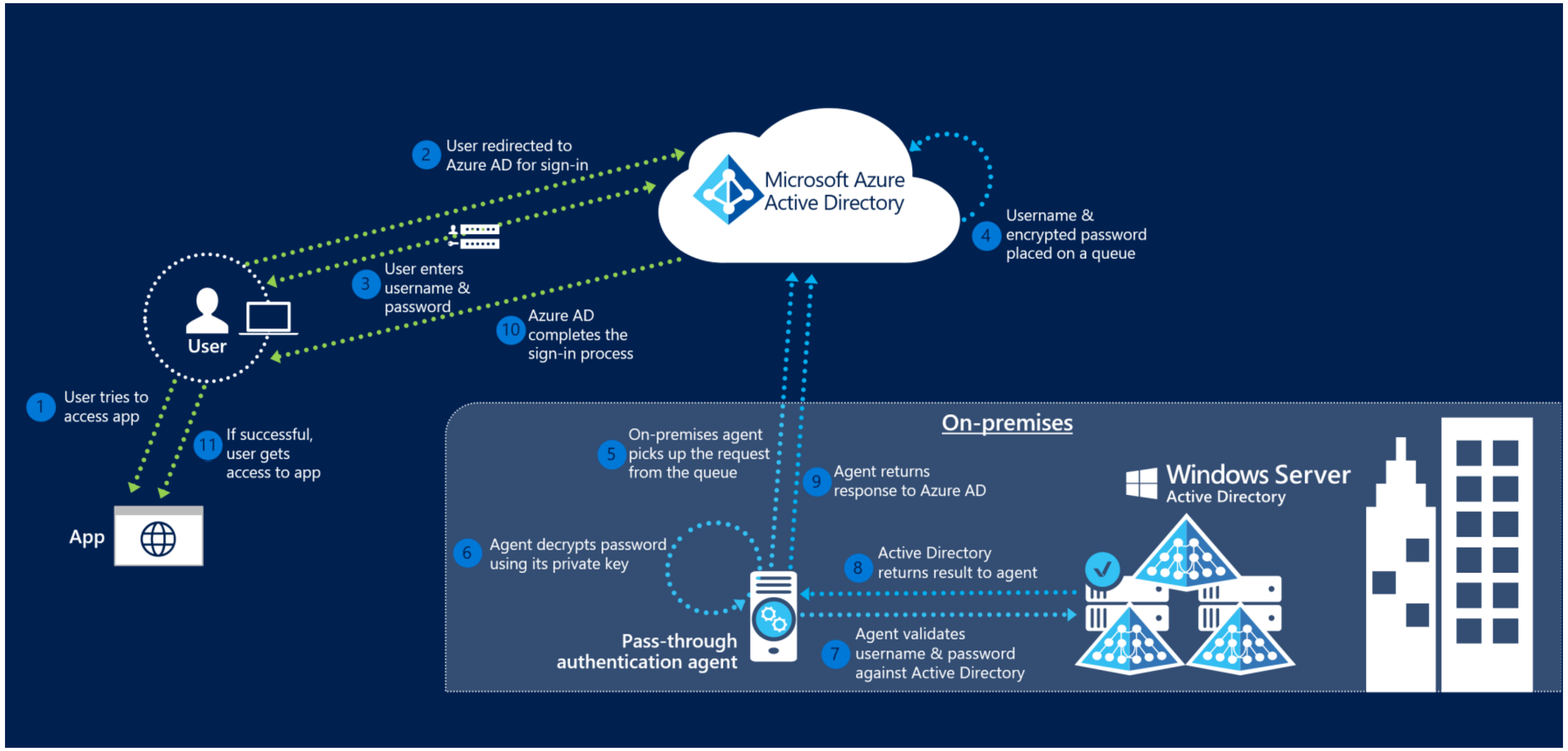
Azure AD Hybrid

- Passthrough vs. ADFS


Office365, SaaS and LoB apps
User sign-in
Microsoft Azure Active Directory
User
Identity synchronization using Azure AD Connect
Pass-through authentication
Password validation requests are sent to *Windows Server Active Directory* via Pass-through authentication
Windows Server Active Directory
Pass-through authentication agent

Azure AD Hybrid

- Passthrough vs. ADFS


Microsoft Azure Active Directory
2 User redirected to Azure AD for sign-in
3 User enters username & password
4 Username & encrypted password placed on a queue
10 Azure AD completes the sign-in process
User
1 User tries to access app
11 If successful, user gets access to app
App
On-premises
5 On-premises agent picks up the request from the queue
9 Agent returns response to Azure AD
Windows Server Active Directory
6 Agent decrypts password using its private key
8 Active Directory returns result to agent
7 Agent validates username & password against Active Directory
Pass-through authentication agent

Azure AD Domain Services

Azure AD Domain Services integrates previously created Hybrid scenarios or works as a cloud only solution. The benefits are:

- Simplicity – few clicks to setup
- Integrated – deep Azure AD integration
- Compatible – Windows Server AD
- Cost-effective – no infrastructure burden

Lab Exercises

- https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/tree/master/Instructions
- Deploying Managed Containerized Workloads to Azure
- Deploying Serverless Workloads to Azure
- Building Azure IaaS-Based Server Applications by using Azure Resource Manager Templates and Azure Building Blocks.

THANK YOU for Participating in Today’s Session

1

Register your attendance

1. Go to **Link** provided in **chat window**
2. Click **Register Now**
3. Click **Confirm**

2

We'd love your feedback!

Please provide your feedback by clicking on the “***short survey***” link in the “**Did your session make the grade?**” **email you** will soon **receive**