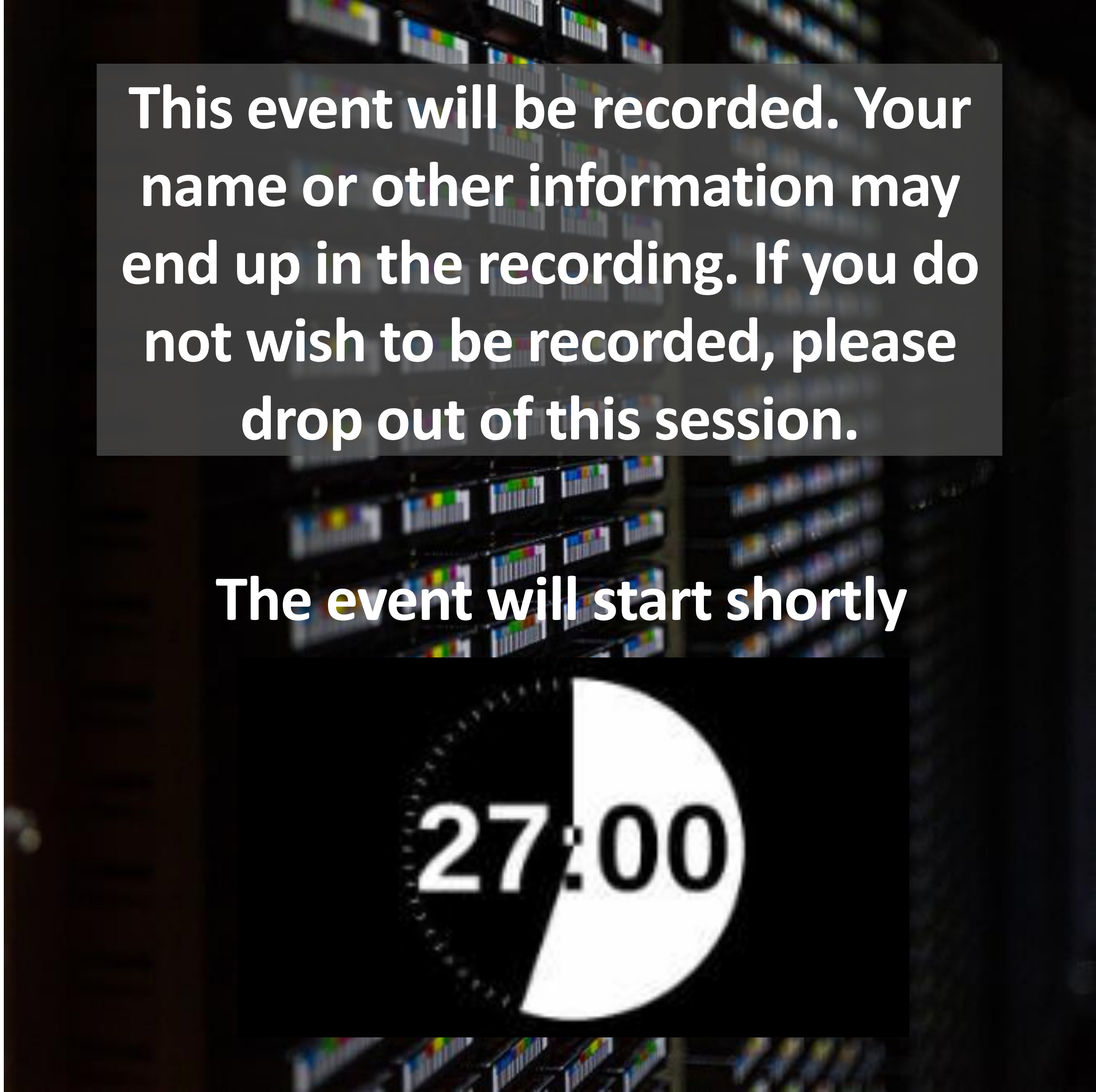
Microsoft

Welcome to the

SaaS Lab Program

Session 2

Building Killer Apps with Azure



This event will be recorded. Your name or other information may end up in the recording. If you do not wish to be recorded, please drop out of this session.

The event will start shortly

Hello, meet your session presenters

About: With over 11 years of experience in the IT industry, Sajeetharan is a Cloud Solution Architect, an enthusiast in Cloud and Opensource. He currently works at Microsoft as a Cloud Solution Architect for ISVs in the APAC OCP Tech team, helping partners to build high-quality solutions using Azure Cloud. He mainly focus on channeling his knowledge into opensource projects and sharing it with the community by mentoring, creating POCS, running workshops, writing blogs to help make the world a better and more developed place.

Sajeetharan Sinnathurai

Cloud Solution Architect (ISV), APAC OCP

Sajeetharan.sinnathurai@microsoft.com

Sajeetharan Sinnathurai | LinkedIn

About : Vorapat (Guide) has been actively engaging enterprise customers and ISVs to help them with Azure architecture for the past years. He brought his software development and DevOps skills during his time as a site engineer of a high-transacting flight booking platform. Now he is expanding his DevOps journey into MLOps.

Vorapat Nicklamai

Cloud Solution Architect (ISV), APAC OCP

Vorapat.Nicklamai@microsoft.com

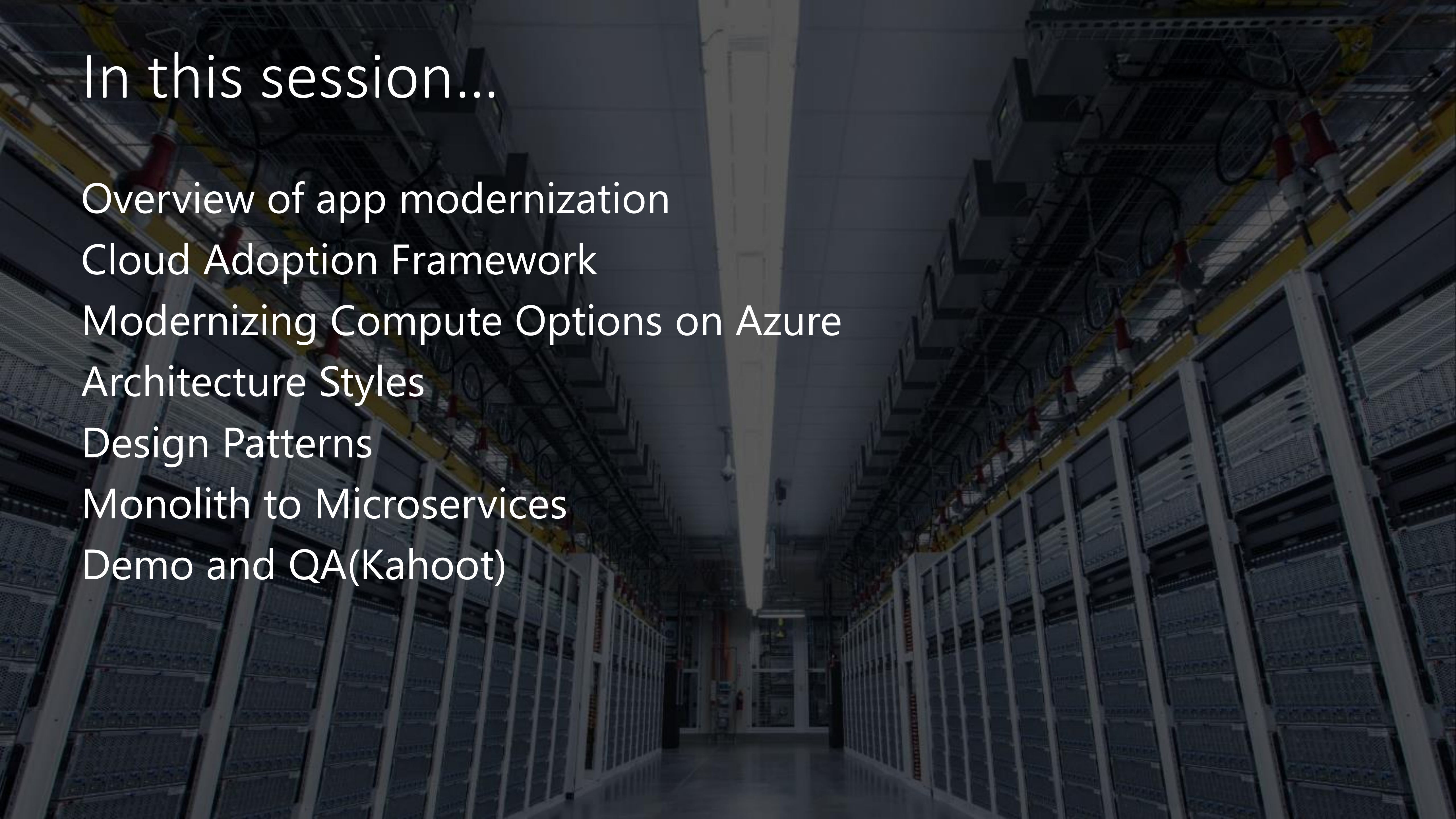
Vorapat Nicklamai | LinkedIn

Your feedback is important

Please help us improve this program by completing this short feedback form.

https://aka.ms/saaslabfeedback2

In this session…



Overview of app modernization

Cloud Adoption Framework

Modernizing Compute Options on Azure

Architecture Styles

Design Patterns

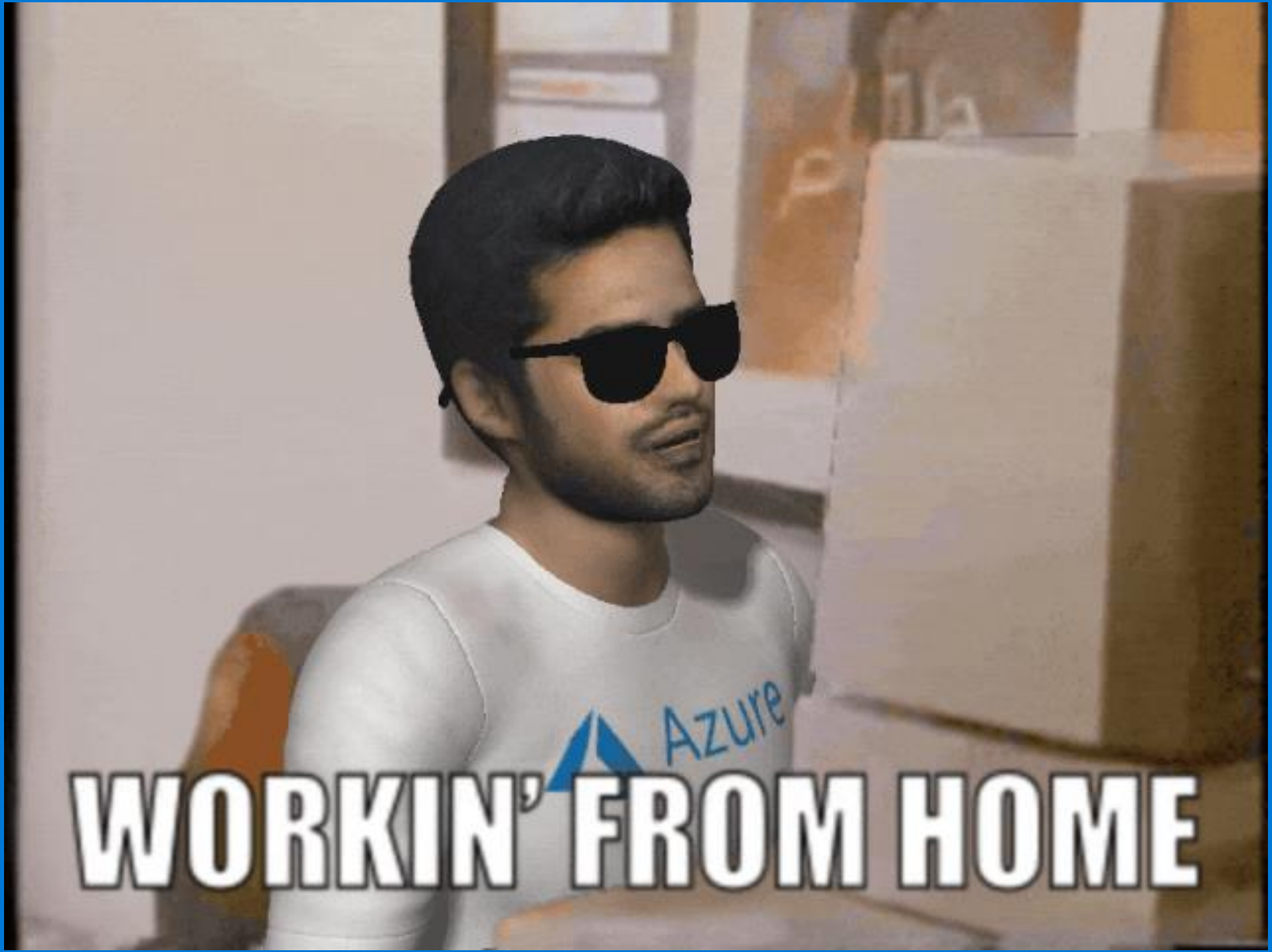
Monolith to Microservices

Demo and QA(Kahoot)

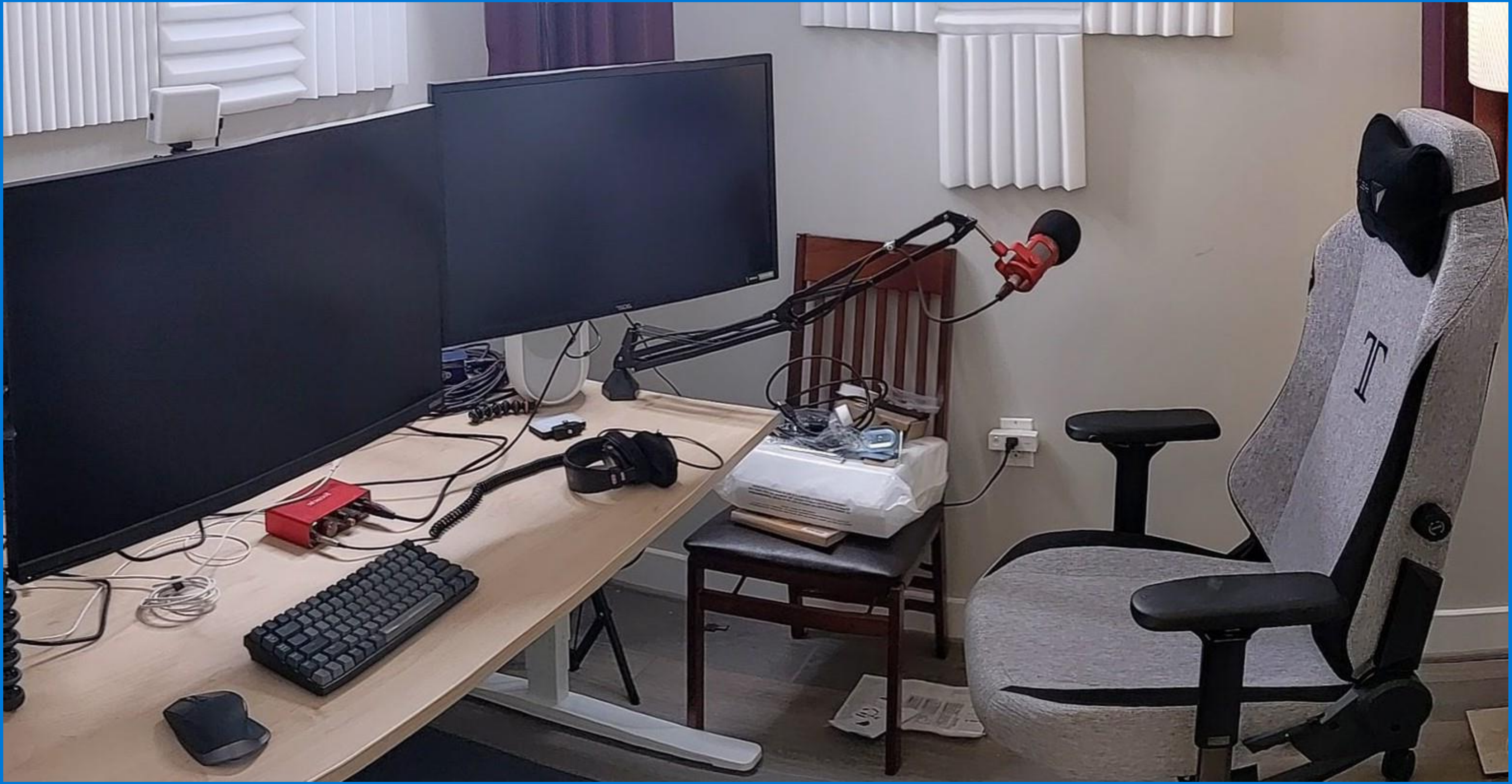
POLL Time:

What’s your application stack or main programming language?

Let me share a story



My Modernized work from home setup



Traditional application has a set of challenges

Aging infrastructure

- Aging hardware, operating systems, and business applications in the datacenter can impact:
- Operational costs, efficiency, and reliability
- Capital expenditure requirements
- Security, audit, and regulatory compliance

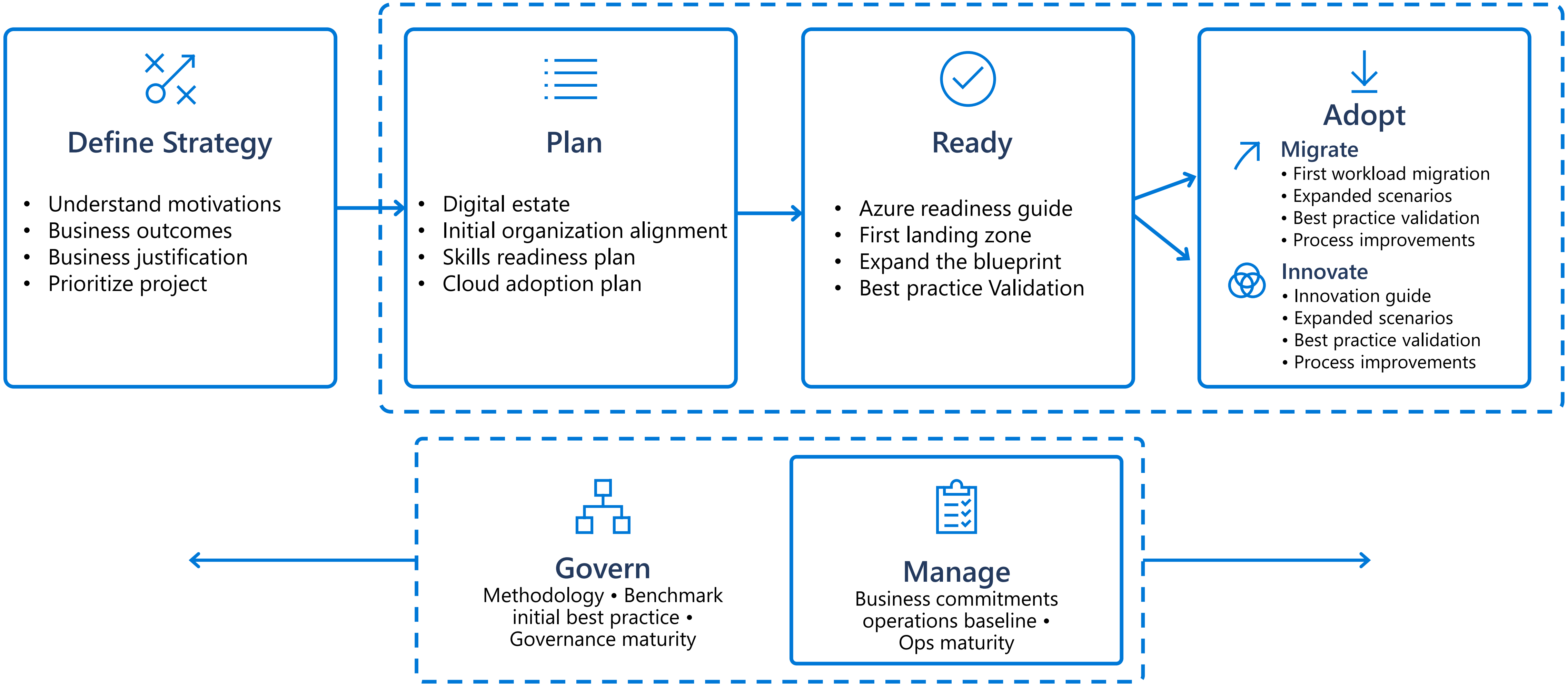
Lack of agility

- Deployment time of new services
- Operation is time (and budget) consuming
- Innovation is happening outside IT inside business areas

Legacy applications

- Longer release cycles, monolithic and highly coupled architecture
- Highly IT dependent
- Low application performance and time-to-market compromise business agility

Azure Cloud Adoption Framework



Define Strategy

- Understand motivations
- Business outcomes
- Business justification
- Prioritize project

Plan

- Digital estate
- Initial organization alignment
- Skills readiness plan
- Cloud adoption plan

Ready

- Azure readiness guide
- First landing zone
- Expand the blueprint
- Best practice Validation

Adopt

Migrate

- First workload migration
- Expanded scenarios
- Best practice validation
- Process improvements

Innovate

- Innovation guide
- Expanded scenarios
- Best practice validation
- Process improvements

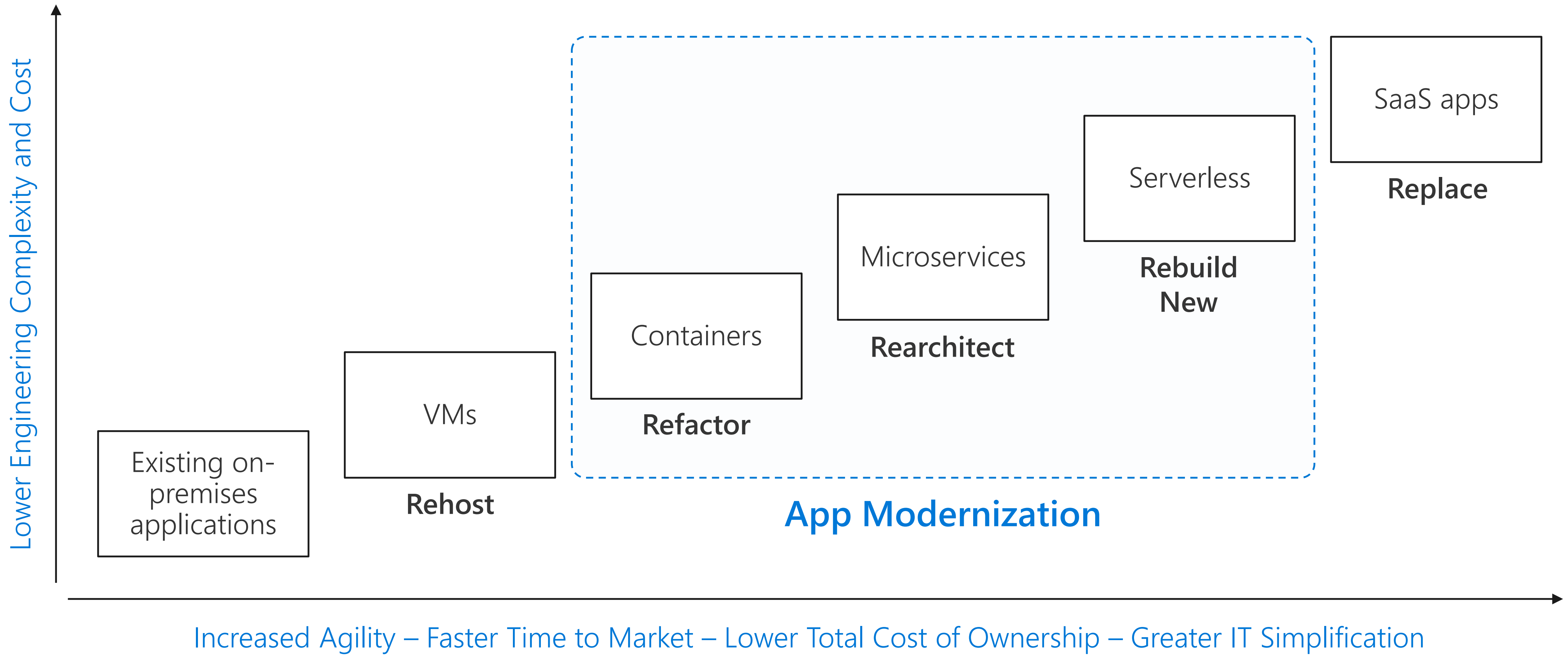
Govern

Methodology • Benchmark
initial best practice •
Governance maturity

Manage

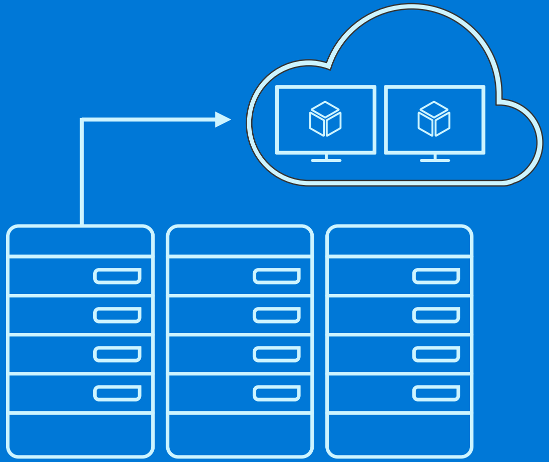
Business commitments
operations baseline •
Ops maturity

Cloud app continuum


Lower Engineering Complexity and Cost
Existing on-premises applications
VMs
Rehost
Containers
Refactor
Microservices
Rearchitect
Serverless
Rebuild New
SaaS apps
Replace
App Modernization
Increased Agility – Faster Time to Market – Lower Total Cost of Ownership – Greater IT Simplification

Disclaimer : Not be the case on every scenario!

Lift and Shift(Rehost)



Definition:
Redeploy the application to a different hardware environment or change the application's infrastructure configuration

When to consider

- Ideal when your goal is to improve operational efficiencies, and free up data center space
- Maintenance apps for which the hardware is not worth additional investment
- Compute-intensive applications that are built for parallelism but don't require high-performance interprocess communications (IPC) and have independent datasets, and applications for which load balancing already increases scalability and availability.

Benefits

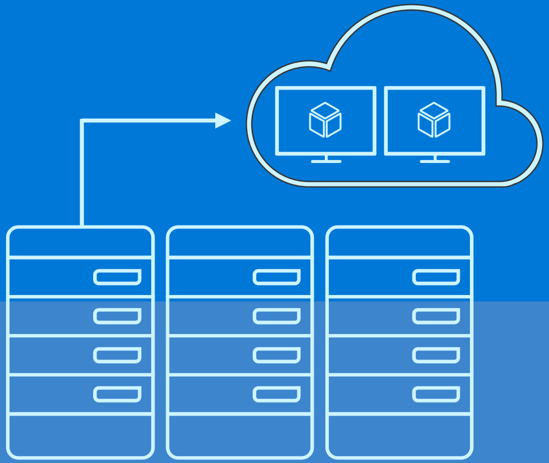
- Drives instant reduction in TCO - 30% on average
- No need to manage data centers
- Enjoy flexible and scalable infrastructure

Core technologies

- VM, VM Scale Set

Source: Decision Point for Choosing a Cloud Application Migration Strategy, Gartner. Published: 29 March 2016

Refactor



Definition

Modify your application so that it can begin to take advantage of cloud capabilities for agility, elasticity and minimized resource use

When to consider

- You want to leverage existing development skills and codebase is paramount
- When code portability is a concern.
- You prefer a quick way to modernize your apps

Benefits

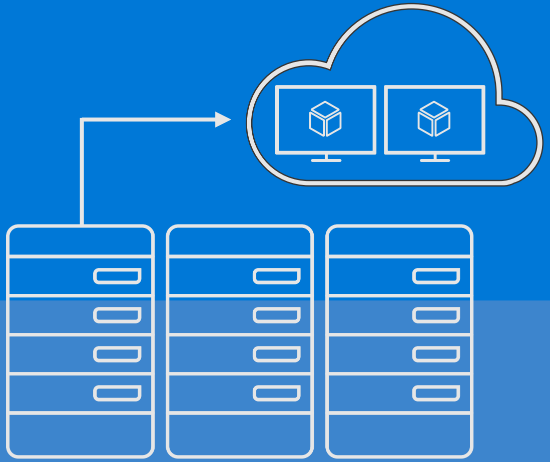
- **Drive continuous innovation by leveraging built-in DevOps for PaaS or using Containers.**
- Existing programming models, languages and frameworks that can be easily used and extended.
- Easily scale up or down to meet the changing needs of the business

Core technologies

- Containers, container ochestration
- DevOps tools

Source: Decision Point for Choosing a Cloud Application Migration Strategy, Gartner. Published: 29 March 2016

Rebuild



Definition:
Build new application using cloud native environment. Wherever possible, prioritize high-productivity PaaS - model driven or rapid application development

When to consider

- You want to build for cloud-native PaaS environments from ground up.
- Leverage previous investment in a cloud platform, e.g. when customer data has already moved to the Cloud.
- Rapid prototyping is crucial or the scope of a current application is too limited in terms of functionality and lifespan.

Benefits

- Reduce TCO
- Fully leverage the cloud native capabilities and build applications faster
- Expedite your business innovation

Core technologies

- Serverless, PaaS

Source: Decision Point for Choosing a Cloud Application Migration Strategy, Gartner. Published: 29 March 2016

Choosing migration strategy and technology

		Objectives	Cloud strategy					Options to consider
			Rehost	Refactor	Rearchitect	Re-build	Replace	
Innovation	1	Deliver new capabilities faster				✓		PaaS, Serverless
	2	Provide multichannel access, including mobile				✓	✓	PaaS, Serverless
	3	Enable business agility with continuous innovation		✓	✓			PaaS, Containers
Differentiation	1	More easily integrate with other web and cloud apps			✓	✓		PaaS, Serverless
	2	Infuse intelligence into processes leveraging existing investments		✓	✓			PaaS, Serverless
	3	Increase agility & support scalability requirements of existing applications more cost effectively		✓	✓			PaaS, Containers
Record	1	Free up data center space quickly	✓				✓	VMs, SaaS
	2	Reduce capital expenditure of existing applications	✓				✓	VMs, SaaS
	3	Achieve rapid time to cloud	✓					VMs

Note: Some of the objective might apply to more than one category of applications

POLL Time:

What are the compute services that you are familiar with on Azure ?

Is there any challenges when adopting those services?

Key Components of Cloud Native

Containers

Tool to package your app, run it portably on different hosts in a consistent way

Serverless

Platform for running and scaling apps where almost all of the operations tasks are managed by the cloud provider. Optimized to let developers focus on code and business value.

Kubernetes

Platform to manage and scale your app reliably (made up of containers) that may span many physical and virtual machines.

A tool for operations, not development

By 2021, 40% of production apps will be cloud native

Azure: The Power Of Choice

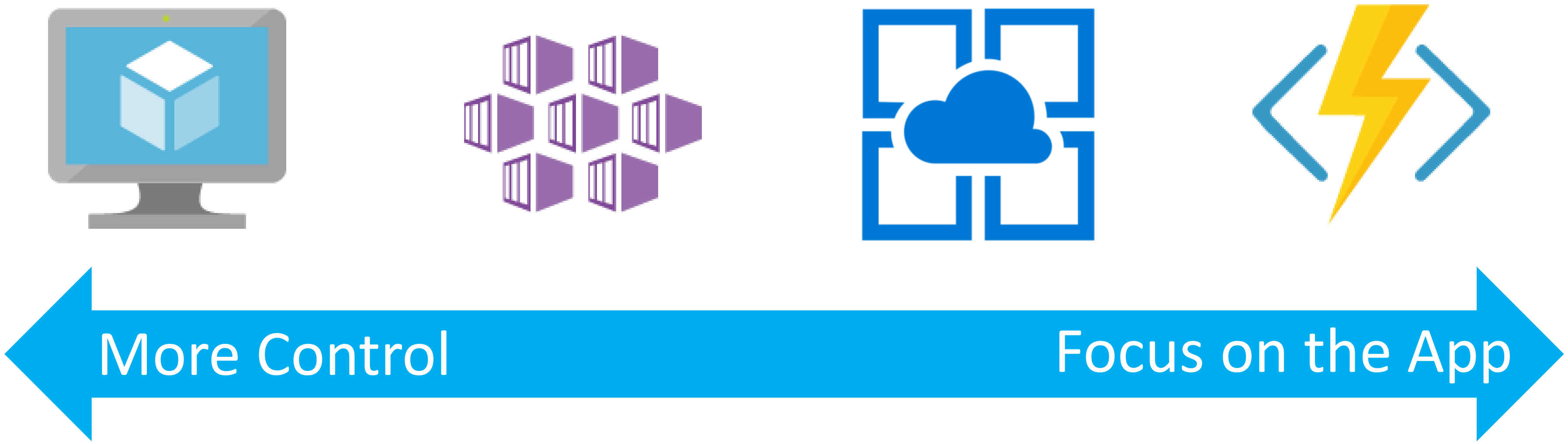
Compute

Virtual Machines

Container Services

App Service

Functions



More Control

Focus on the App

Customer-managed
(IaaS)

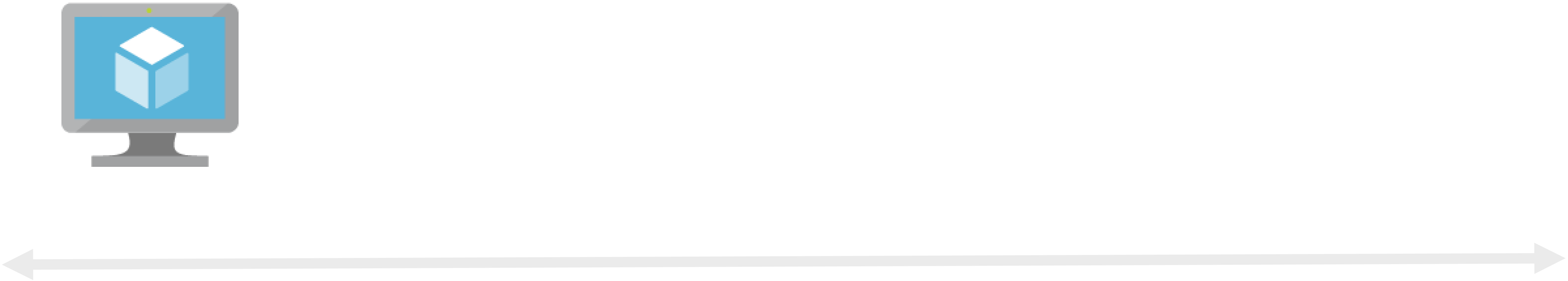
Platform-managed
(PaaS)

Code-only
(serverless)

Azure: The Power Of Choice

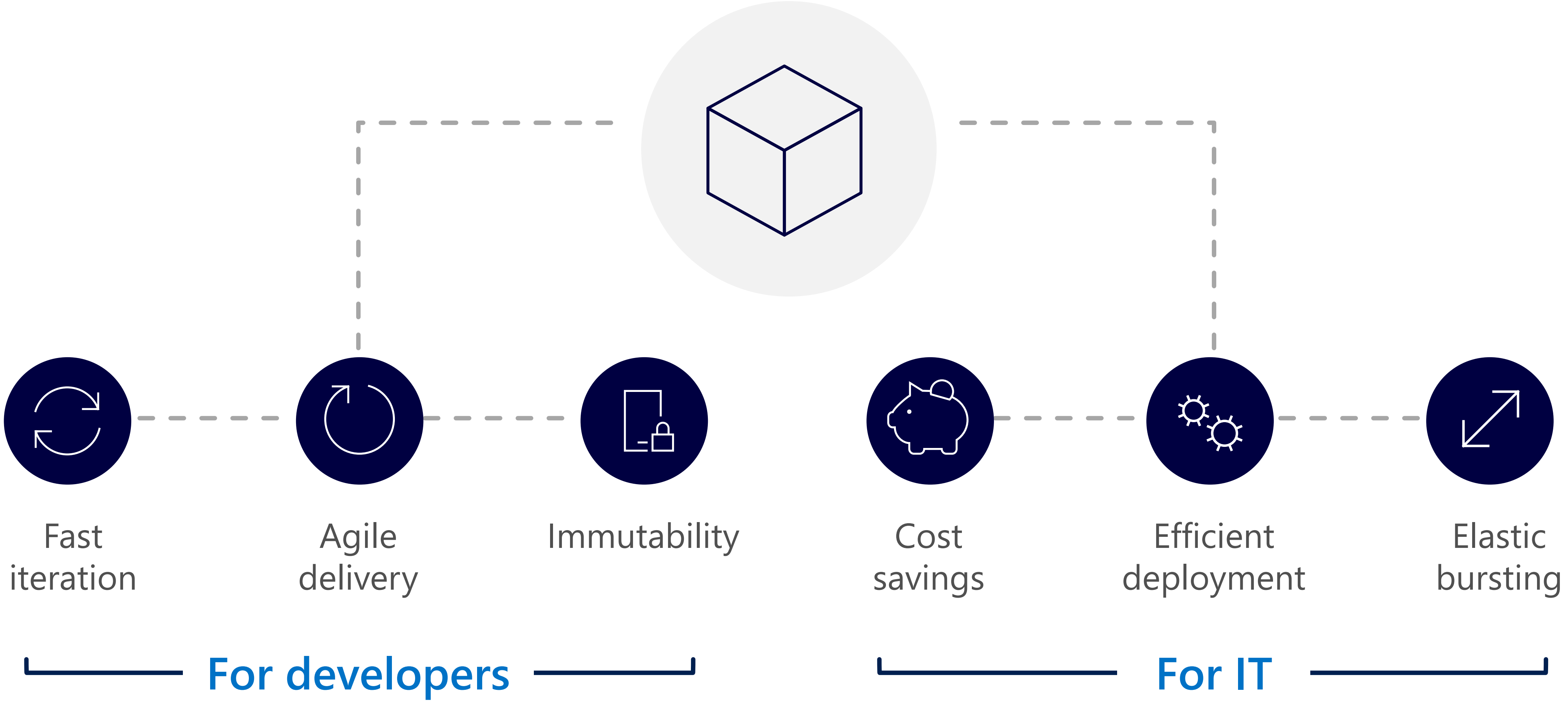
Application Hosting

Virtual Machines



Customer-managed
(IaaS)

The container advantage



Fast iteration

Agile delivery

Immutability

Cost savings

Efficient deployment

Elastic bursting

For developers

For IT

AKS: Simplify the deployment, management, and operations of Kubernetes

Deploy and manage Kubernetes with ease

Accelerate containerized application development

Set up CI/CD in a few clicks

Secure your Kubernetes environment

Scale and run applications with confidence

Work how you want with open-source tools & APIs

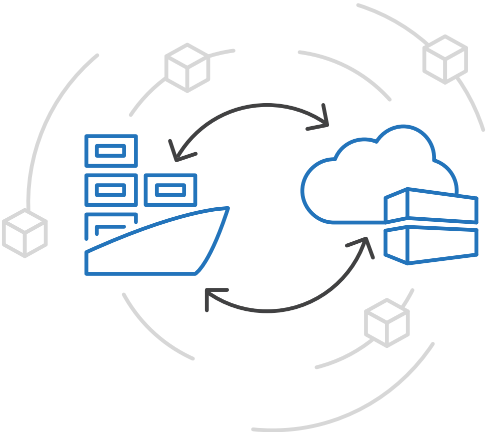
Azure Kubernetes Service (AKS)

Azure Container Instances (ACI)

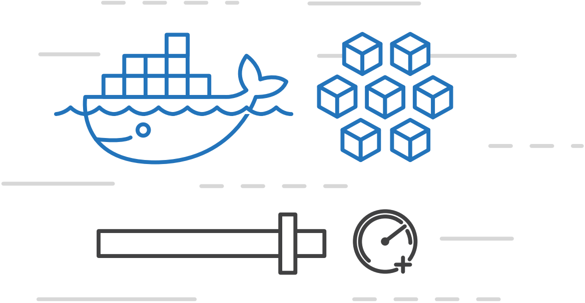
Azure Container Registry

Azure Container Instances (ACI)

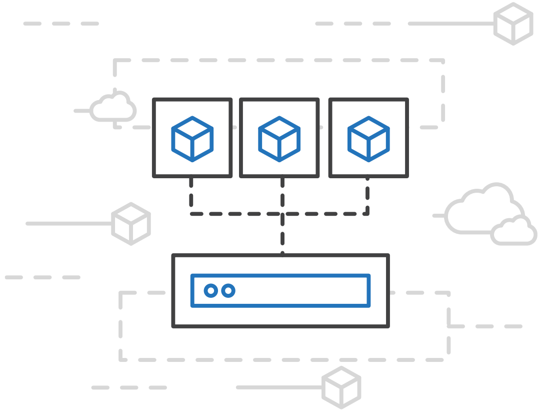
Easily run containers on Azure with a single command



Start using containers right away



Cloud-scale container capacity



Hyper-visor isolation

Azure App Service

Quickly build, deploy and scale powerful cloud applications without worrying about infrastructure

High productivity	.NET, Node, Java, Docker, PHP, Ruby, Python	Staging & deployment	Testing in production	App gallery marketplace
Fully managed	Auto scale & load balancing	High availability w/ auto patching	Reduced operations costs	Backup & recovery
Enterprise grade	Global data center footprint	Hybrid support	AAD integration	Secure & compliance

What is serverless?

Full abstraction of servers

Developers can just focus on their code—there are no distractions around server management, capacity planning, or availability.

Instant, event-driven scalability

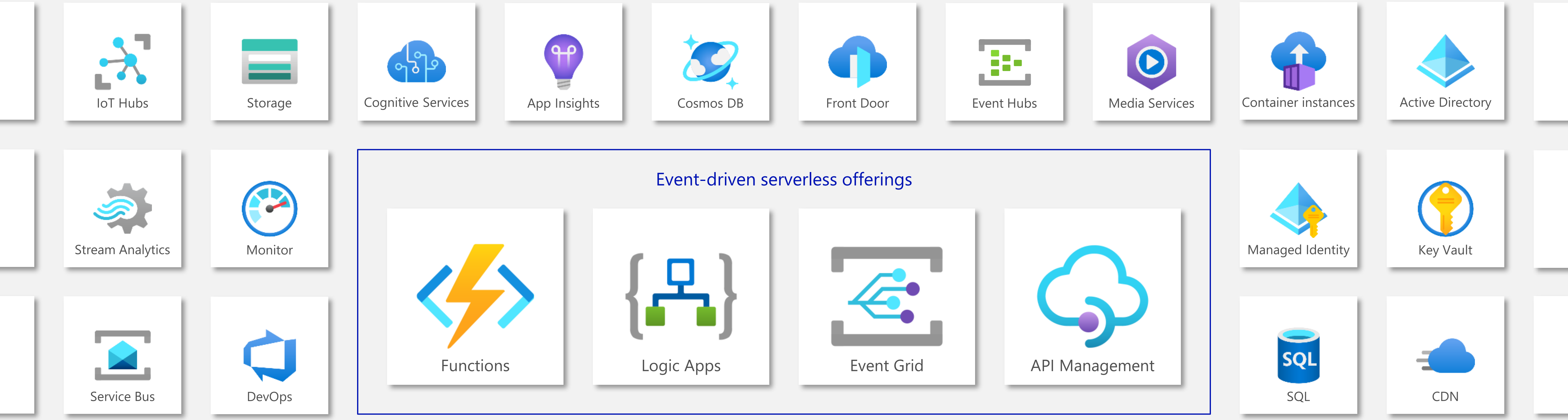
Application components react to events and triggers in near real-time with virtually unlimited scalability; compute resources are used as needed.

Pay-per-use

Only pay for what you use: billing is typically calculated on the number of function calls, code execution time, and memory used.*

*Supporting services, like storage and networking, may be charged separately.

Azure serverless ecosystem



IoT Hubs

Storage

Cognitive Services

App Insights

Cosmos DB

Front Door

Event Hubs

Media Services

Container instances

Active Directory

Stream Analytics

Monitor

Event-driven serverless offerings

Functions

Logic Apps

Event Grid

API Management

Managed Identity

Key Vault

Service Bus

DevOps

SQL

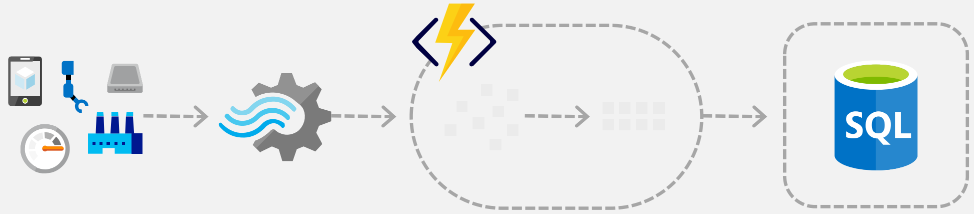
CDN

- IDE integration
- Local development
- Flexible deployment options
- Built-in security
- Rich monitoring
- Compliance and management

Scenarios for Serverless

Anything that needs to respond to events

Real-time stream processing

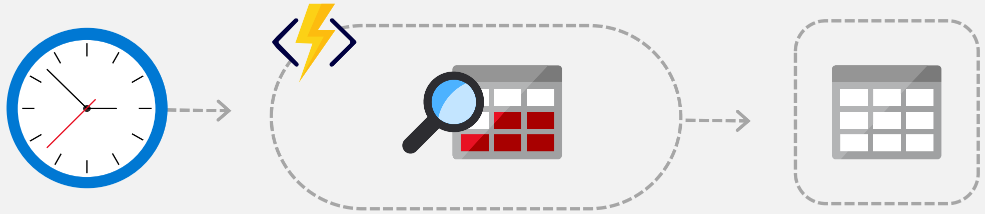


Millions of devices feed into Stream Analytics

Transform to structured data

Store data in SQL DB

Timer-based processing



Every 15 minutes

Find and clean invalid data

Clean table

Backends (Mobile/IoT/Web)

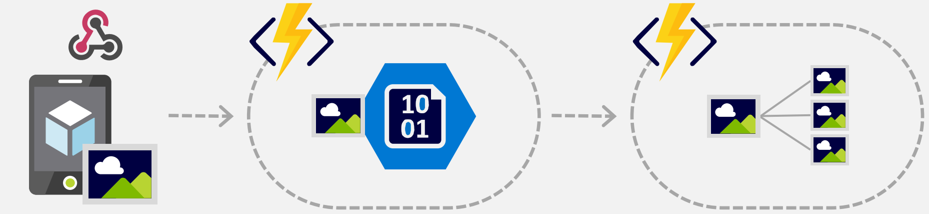
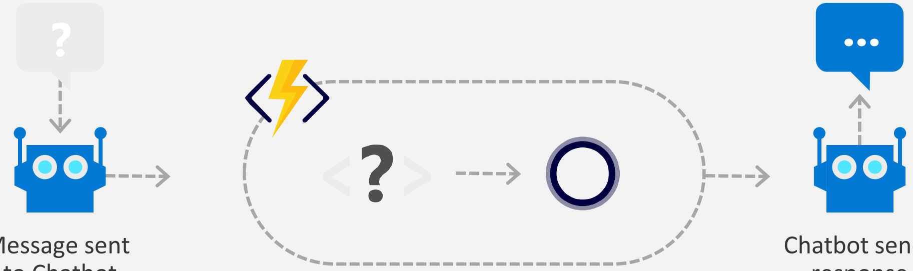


Photo taken and WebHook called

Stores in blob storage

Produces scaled images

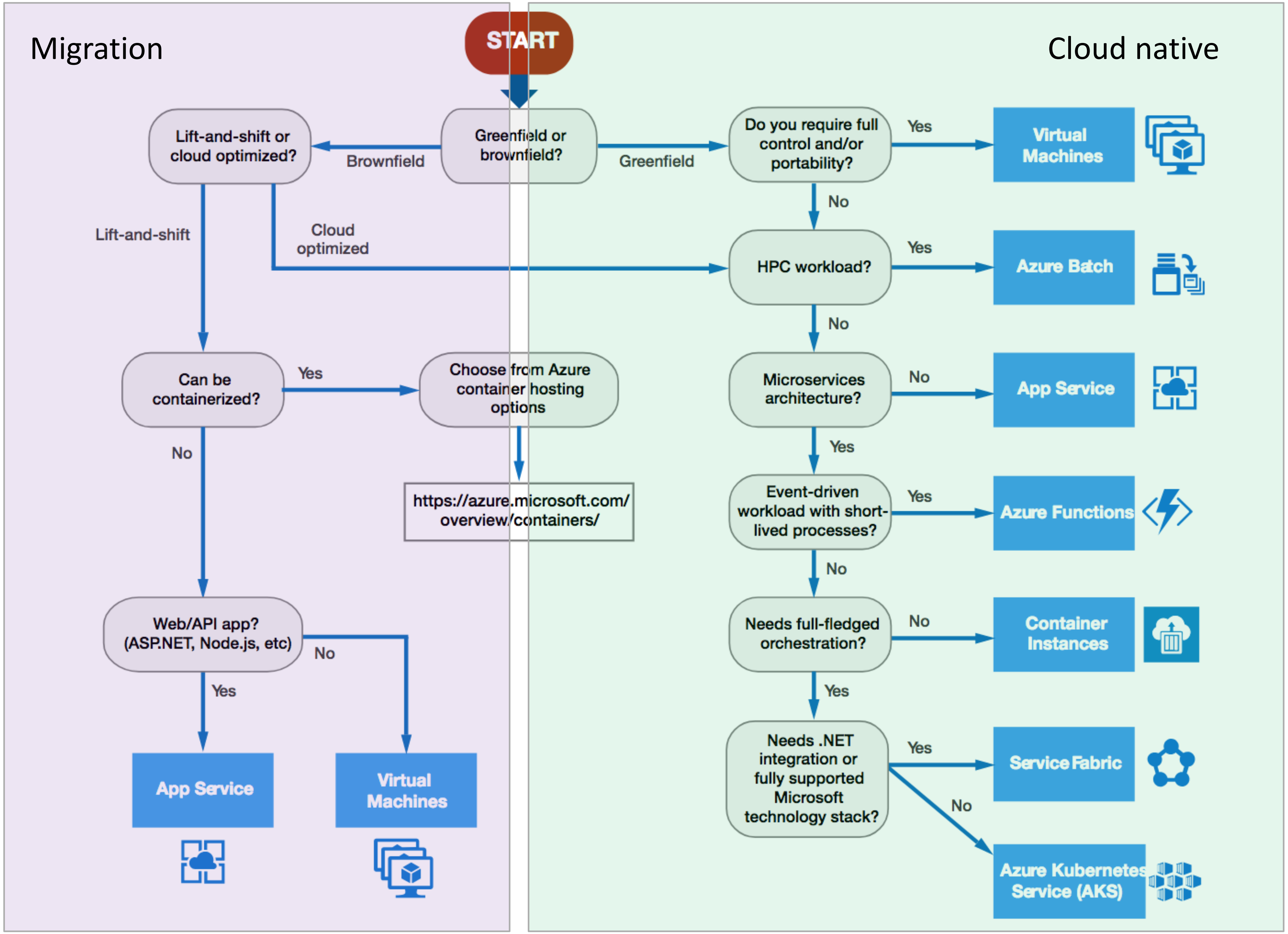
Real-time bot messaging



Message sent to Chatbot

Cortana Analytics answers questions

Chatbot sends response



Migration
START
Cloud native
Lift-and-shift or cloud optimized?
Brownfield
Greenfield or brownfield?
Greenfield
Do you require full control and/or portability?
Yes
Virtual Machines
No
Lift-and-shift
Cloud optimized
HPC workload?
Yes
Azure Batch
No
Can be containerized?
Yes
Choose from Azure container hosting options
Microservices architecture?
No
App Service
No
https://azure.microsoft.com/overview/containers/
Yes
Event-driven workload with short-lived processes?
Yes
Azure Functions
No
Web/API app? (ASP.NET, Node.js, etc)
No
Needs full-fledged orchestration?
No
Container Instances
Yes
Yes
Needs .NET integration or fully supported Microsoft technology stack?
Yes
Service Fabric
No
Azure Kubernetes Service (AKS)
App Service
Virtual Machines

Note: Analyse the scenarios holistically

https://aka.ms/comparecompute

Brain Teaser Time

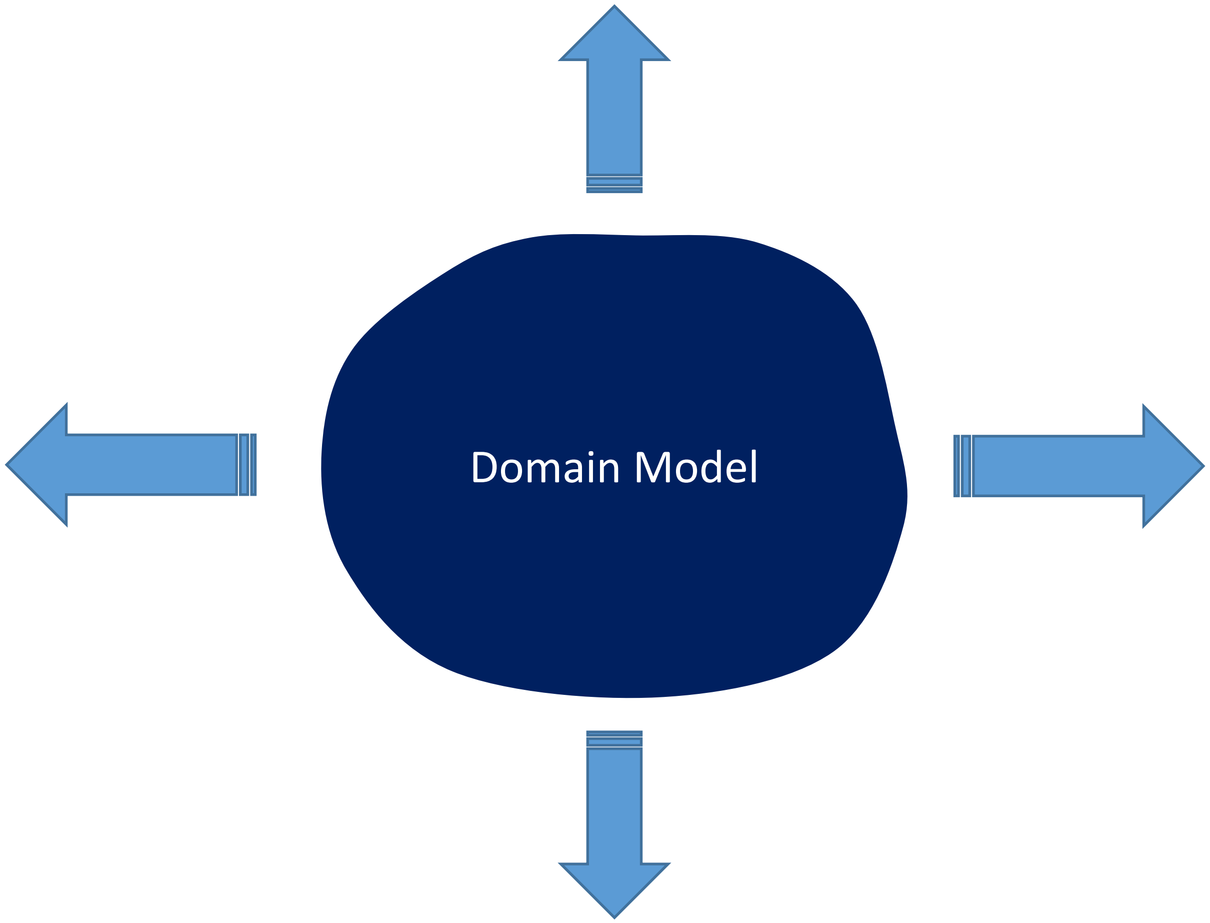
You can earn a free Microsoft Certification exam by taking part in the ……………………..

The official social media #hashtag used during Ignite was ……………………………..

https://news.microsoft.com/ignite-march-2021-book-of-news/

Reference architectures (aka.ms/msmspnp)

Choosing architecture style



Web-queue-worker?

Microservices?

Domain Model

Monolith?

CQRS?

patterns & practices
proven practices for predictable results

Choosing architecture style

Let’s say Microservices

Business domain

- Complex domain
- Frequent update
- Many independent teams

Prerequisites

- Skill set for distributed system
- Domain knowledge
- DevOps culture
- Monitoring capability

Benefits

- Independent deployment
- Fault isolation
- Diverse technology
- Small focused team
- Separate scalability/availability

Challenges

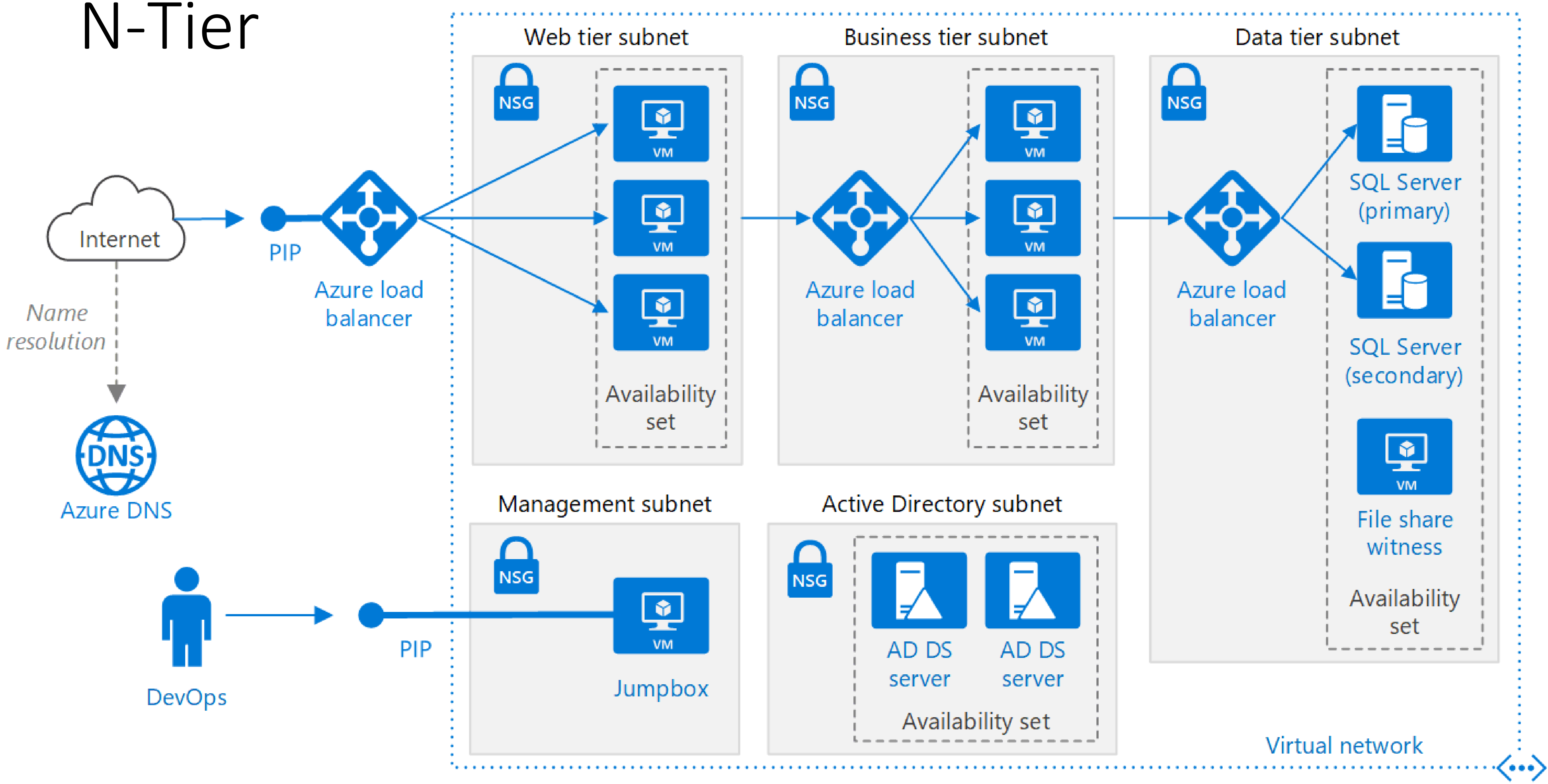
- Complexity
- Network congestion
- Data integrity/consistency
- Testing
- Reliability

patterns & practices
proven practices for predictable results

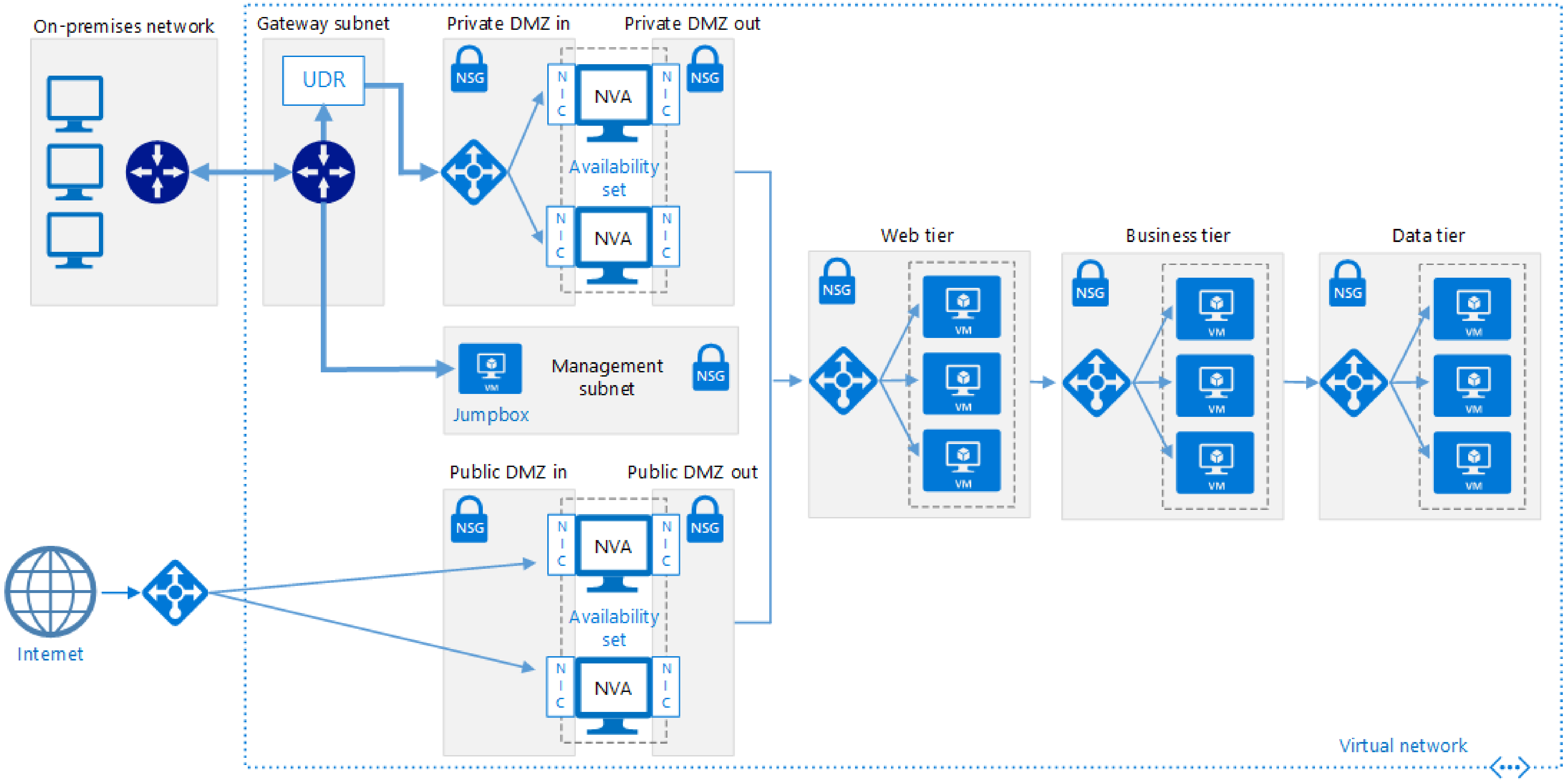
Choosing architecture styles

	Dependency management	Domain type/complexity
N-Tier	Horizontal layers (open/close)	Majority of business logic is CRUD
Web-Queue-Worker	Front/Backend jobs Decoupled by async messaging	Relatively simple domain with some resource intensive tasks
Microservices	Vertical (functional) decoupling Service calls via API	Complicated domain logic that requires each service to encapsulate domain knowledge
CQRS	R/W segregation Schema/Scale are optimized separately	Collaborative domain where lots of users access the same data
EDA(IoT)	Data ingested into streaming Independent view per sub-system	Internet of things
Big data	Divide huge dataset into small chunks Parallel processing on local dataset	Batch and real-time data analysis Predictive analysis using ML
Big compute	Data allocation to thousands of cores	Compute intensive domain such as simulation, number crunching

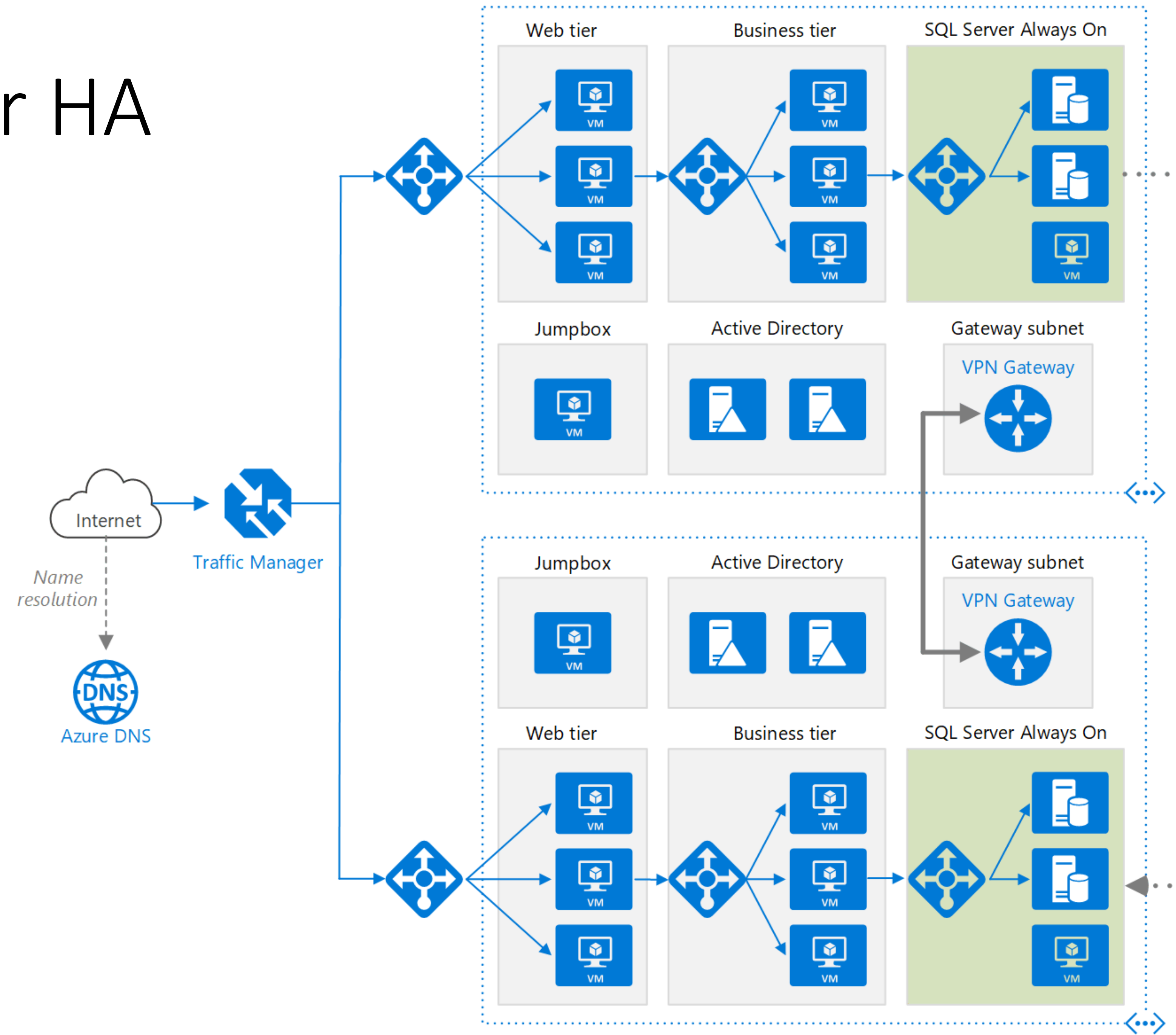
N-Tier


Internet
PIP
Azure load balancer
Name resolution
Azure DNS
DevOps
PIP
Web tier subnet
NSG
VM
VM
VM
Availability set
Business tier subnet
NSG
Azure load balancer
VM
VM
VM
Availability set
Data tier subnet
NSG
Azure load balancer
SQL Server (primary)
SQL Server (secondary)
VM
File share witness
Availability set
Management subnet
NSG
VM
Jumpbox
Active Directory subnet
NSG
AD DS server
AD DS server
Availability set
Virtual network

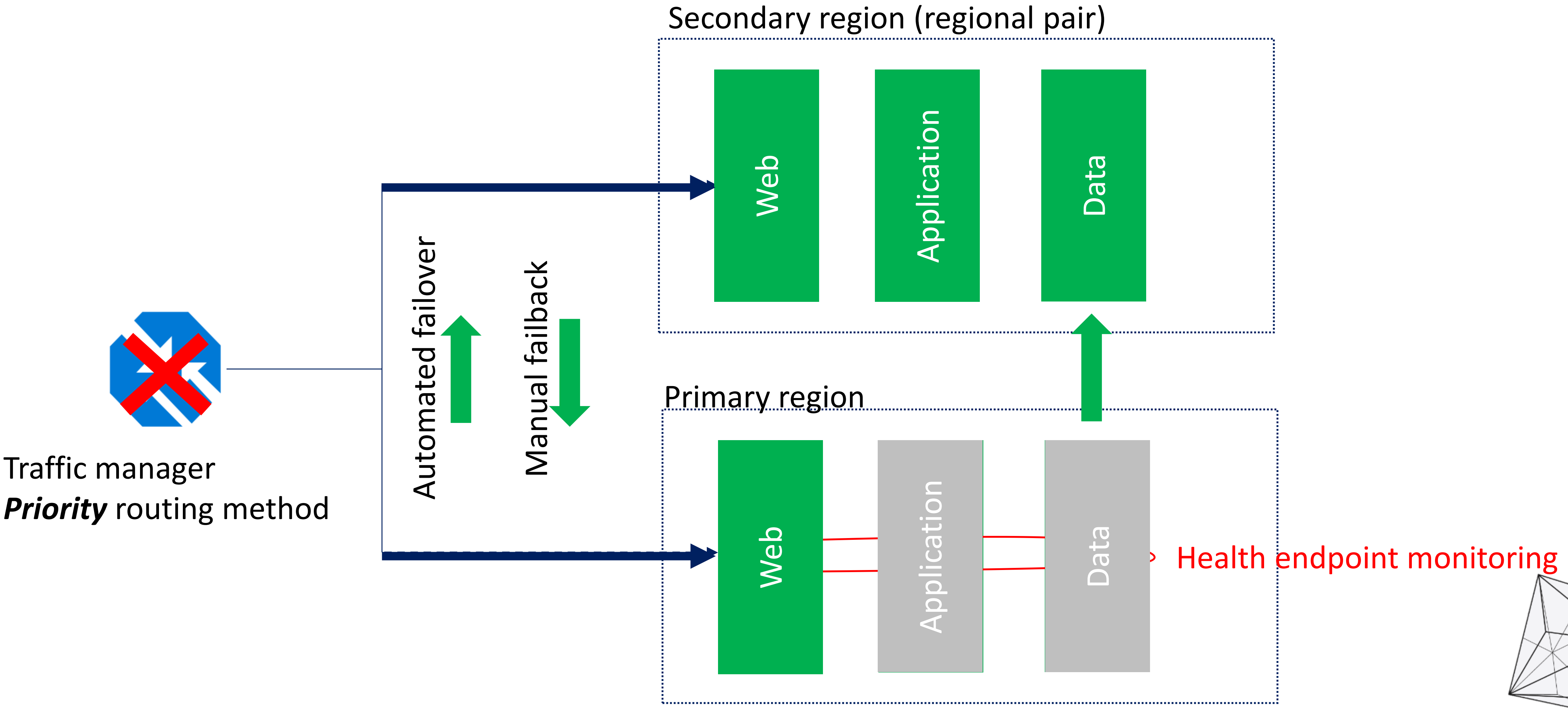
N-Tier+DMZ


On-premises network
Gateway subnet
UDR
Private DMZ in
Private DMZ out
NSG
NIC
NVA
NIC
NSG
Availability set
NIC
NVA
NIC
Management subnet
NSG
VM
Jumpbox
Web tier
NSG
VM
VM
VM
Business tier
NSG
VM
VM
VM
Data tier
NSG
VM
VM
VM
Public DMZ in
Public DMZ out
NSG
NIC
NVA
NIC
NSG
Availability set
NIC
NVA
NIC
Internet
Virtual network

N-Tier HA


Web tier
Business tier
SQL Server Always On
VM
Jumpbox
Active Directory
Gateway subnet
VPN Gateway
Internet
Traffic Manager
Name resolution
Azure DNS
Jumpbox
Active Directory
Gateway subnet
VPN Gateway
Web tier
Business tier
SQL Server Always On

Failover / Failback



Secondary region (regional pair)

Web

Application

Data

Automated failover

Manual failback

Traffic manager
Priority routing method

Primary region

Web

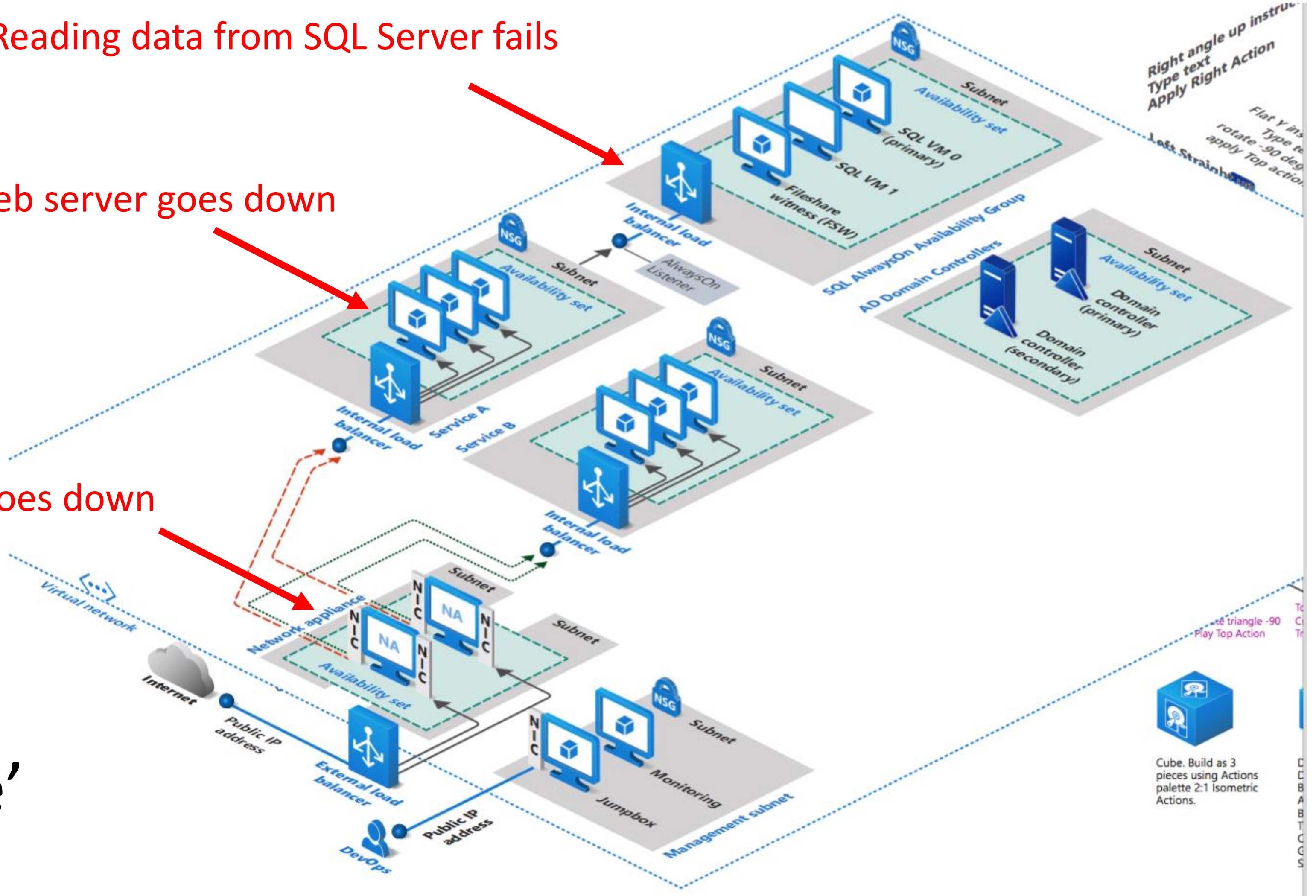
Application

Data

Health endpoint monitoring

Designing for resiliency

1. Identify possible failures
2. Rate risk of each failure (impact x likelihood)
3. Design resiliency strategy
   - Detection
   - Recovery
   - Diagnostics



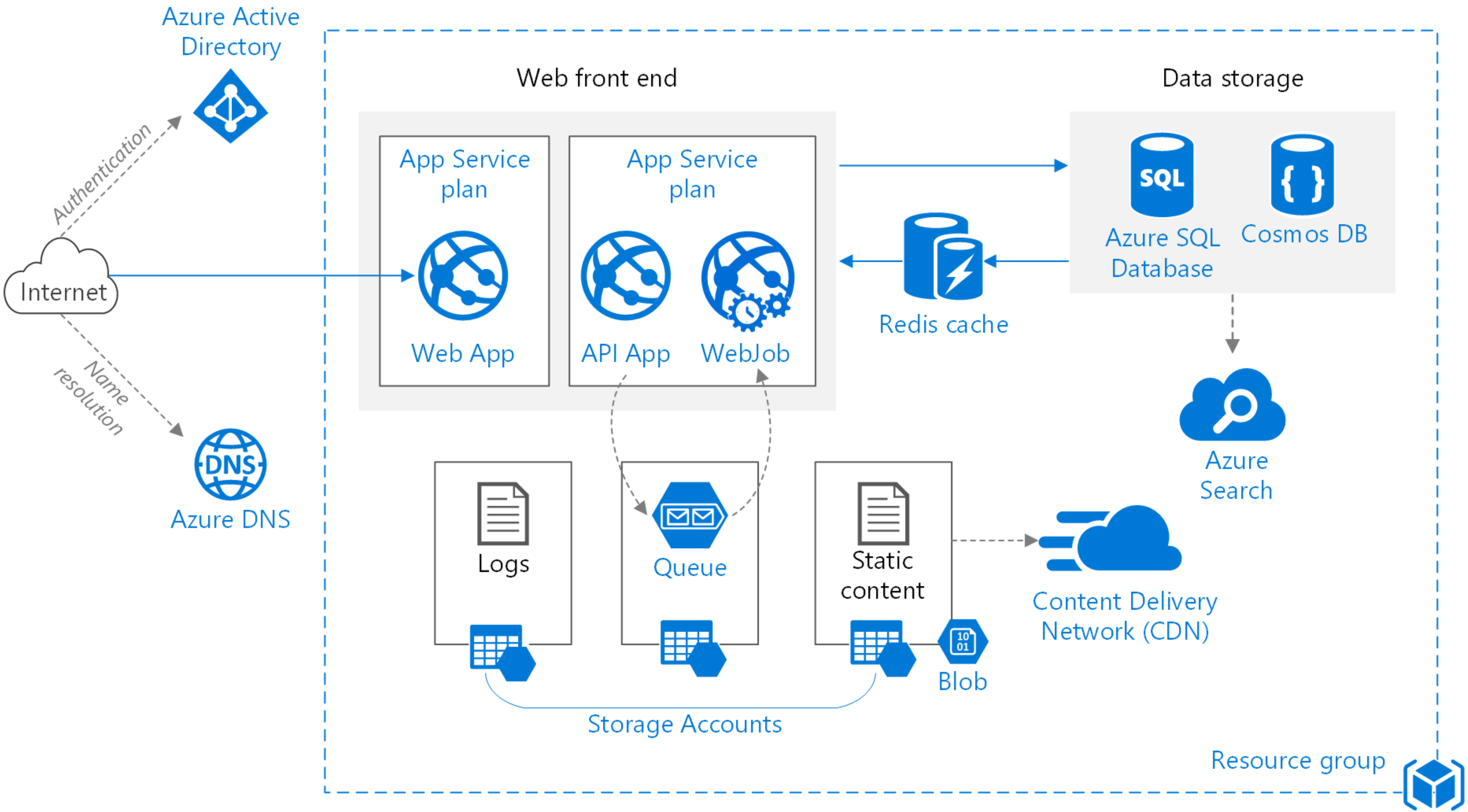
Reading data from SQL Server fails

A web server goes down

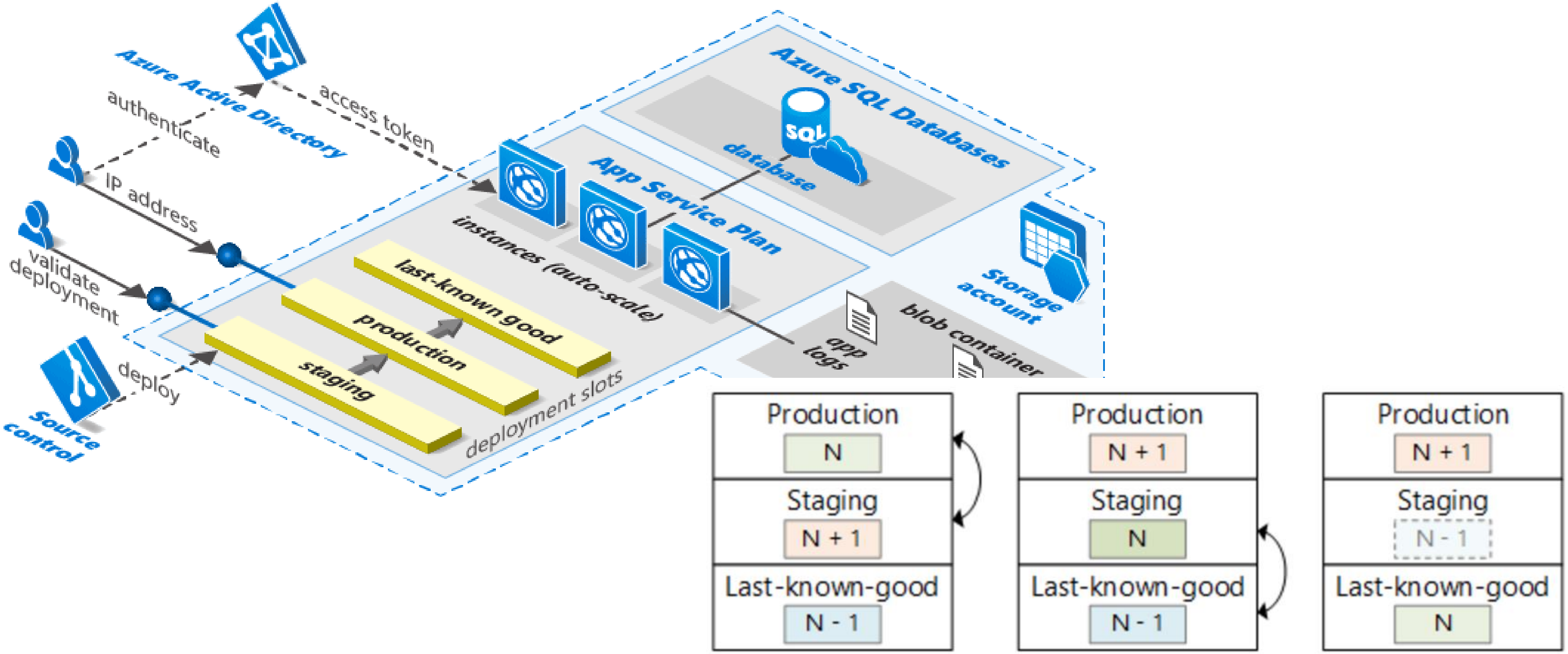
A NVA goes down

‘Azure resiliency guidance’

Web-Queue-Worker



Deployment slots at App Service



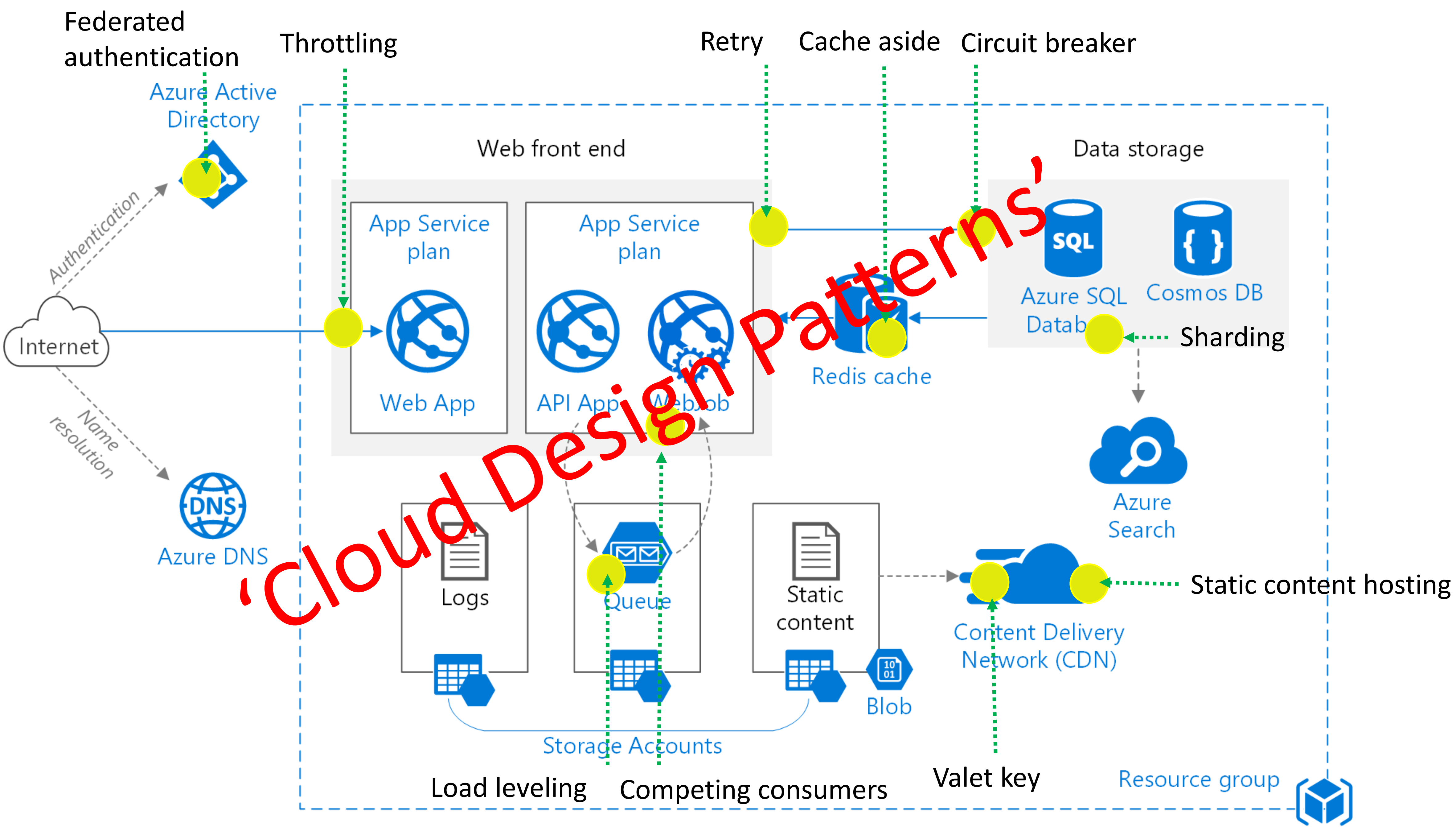
Production	Staging	Last-known-good
N	N + 1	N - 1
N + 1	N	N - 1
N + 1	N - 1	N

Swap production with staging

Swap staging with last-known-good

Ready to stage next update

‘Cloud Design Patterns’



Federated authentication

Throttling

Retry

Cache aside

Circuit breaker

Sharding

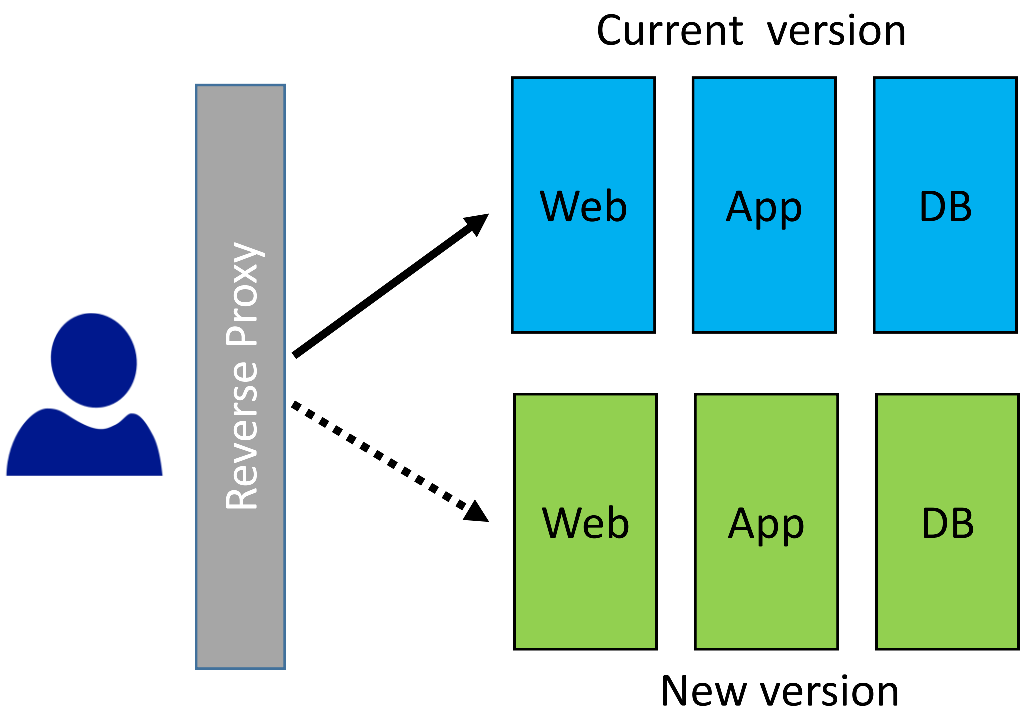
Static content hosting

Load leveling

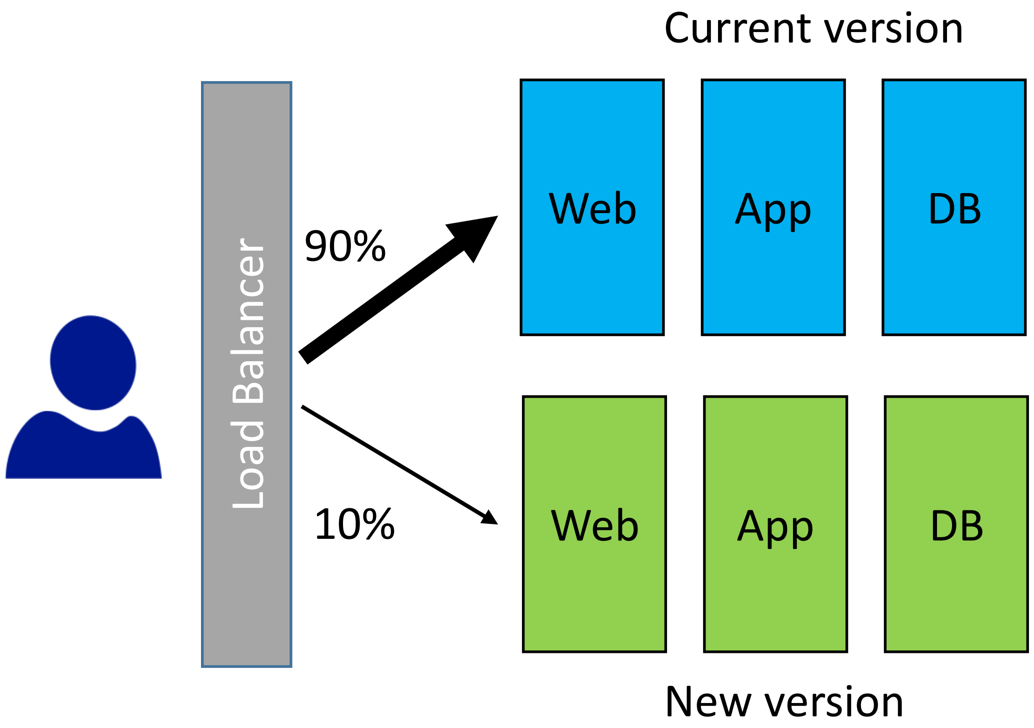
Competing consumers

Valet key

Blue/Green and Canary release



Blue/Green Deployment



Canary release

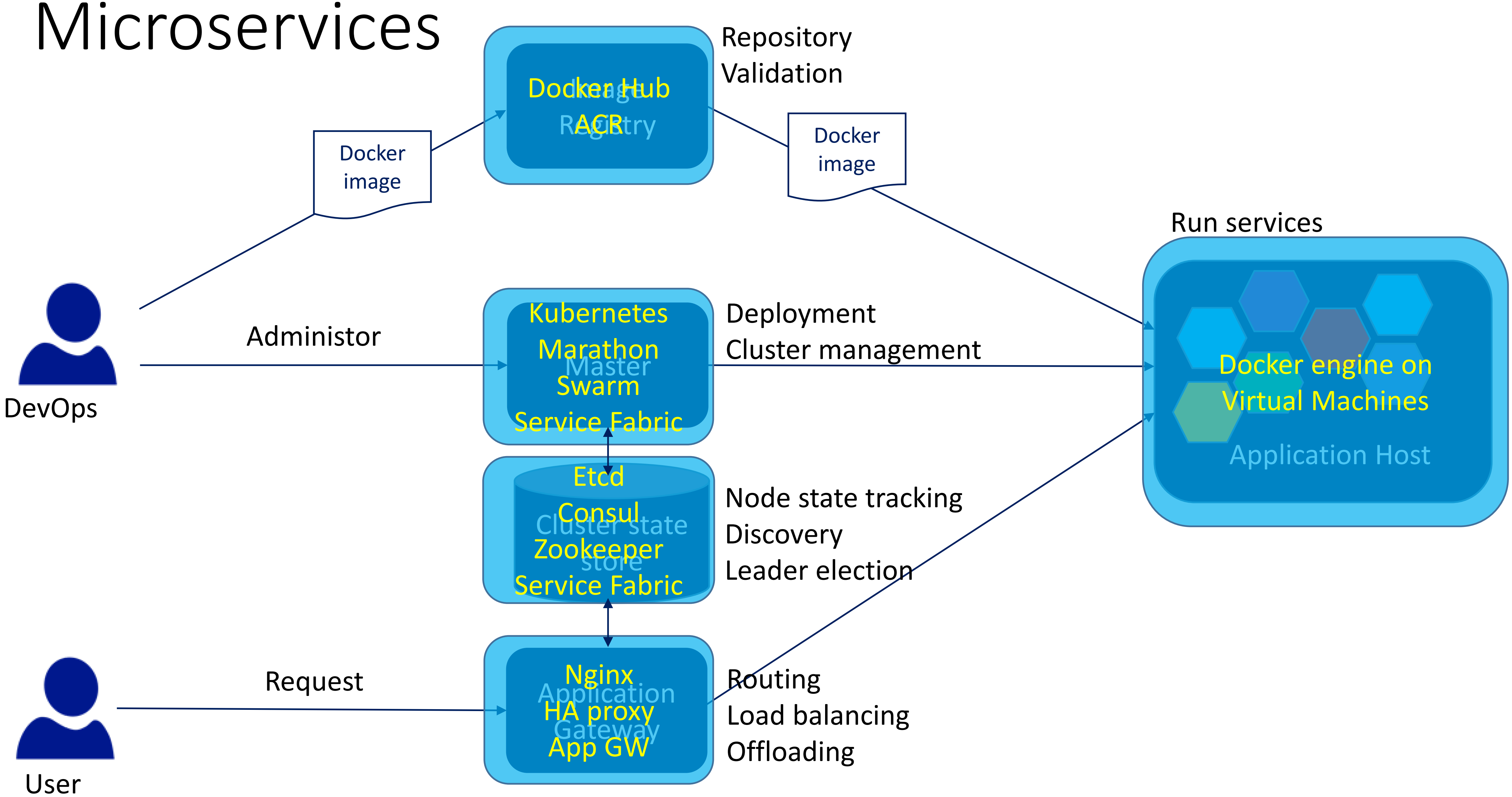
Technology choice - Storage

https://db-engines.com/en/ranking/

- RDBMS: SQL DB, MySQL, Postgres
- Key-Value Store: CosmosDB , Azure Redis, Redis
- Document: CosmosDB, MongoDB
- Column-Family: CosmosDB, Cassandra, HBase
- Graph: CosmosDB, Neo4j
- Search: Azure search, Elasticsearch
- Time series: Time Series Insight, InfluxDB
- Data lake: ADLA/S, HDI
- Object store: Blob storage
- Shared file: File storage

‘Azure data store comparison’

Microservices



Docker Hub

ACR

Image Registry

Repository

Validation

Docker image

Docker image

Run services

DevOps

Administor

Kubernetes

Marathon

Master

Swarm

Service Fabric

Deployment

Cluster management

Docker engine on Virtual Machines

Application Host

Etcd

Consul

Cluster state store

Zookeeper

Service Fabric

Node state tracking

Discovery

Leader election

User

Request

Nginx

Application Gateway

HA proxy

App GW

Routing

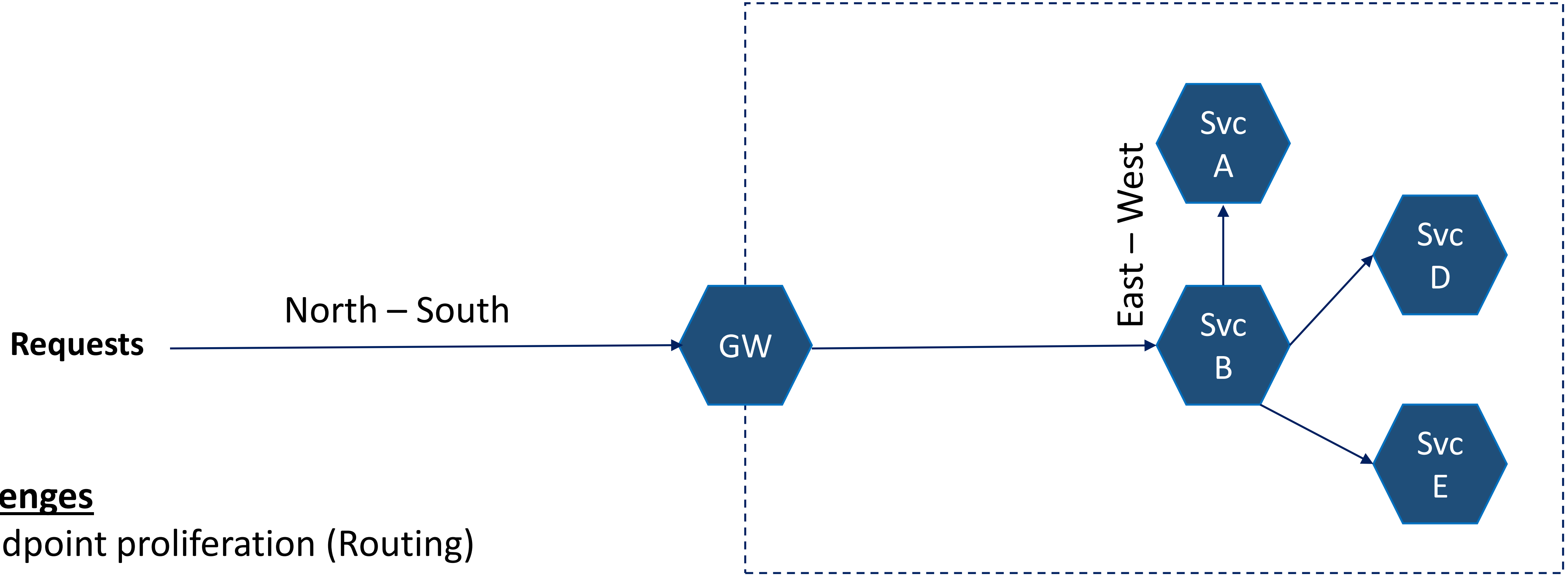
Load balancing

Offloading

Microservices – Other key components

- Messaging framework
  - Queue vs. Streaming vs. Grid
- Monitoring/Logging
  - App Insights, Prometheus, InfluxDB, Zipkin, Fluentd
  - Cost, Scalability, Timestamp resolution
- CI/CD pipeline
  - Devops, Github Actions , Jenkins, Spinnaker
- Service-mesh (Inter-service communication)
  - Linkerd, Istio

Inter service communication



Challenges

- Endpoint proliferation (Routing)
- East – West chattiness (LB)
- Resiliency (Retry, FI)
- Versioning (SxS, B/G)
- Monitoring (Distributed tracing)
- Security (Encryption, Authentication)

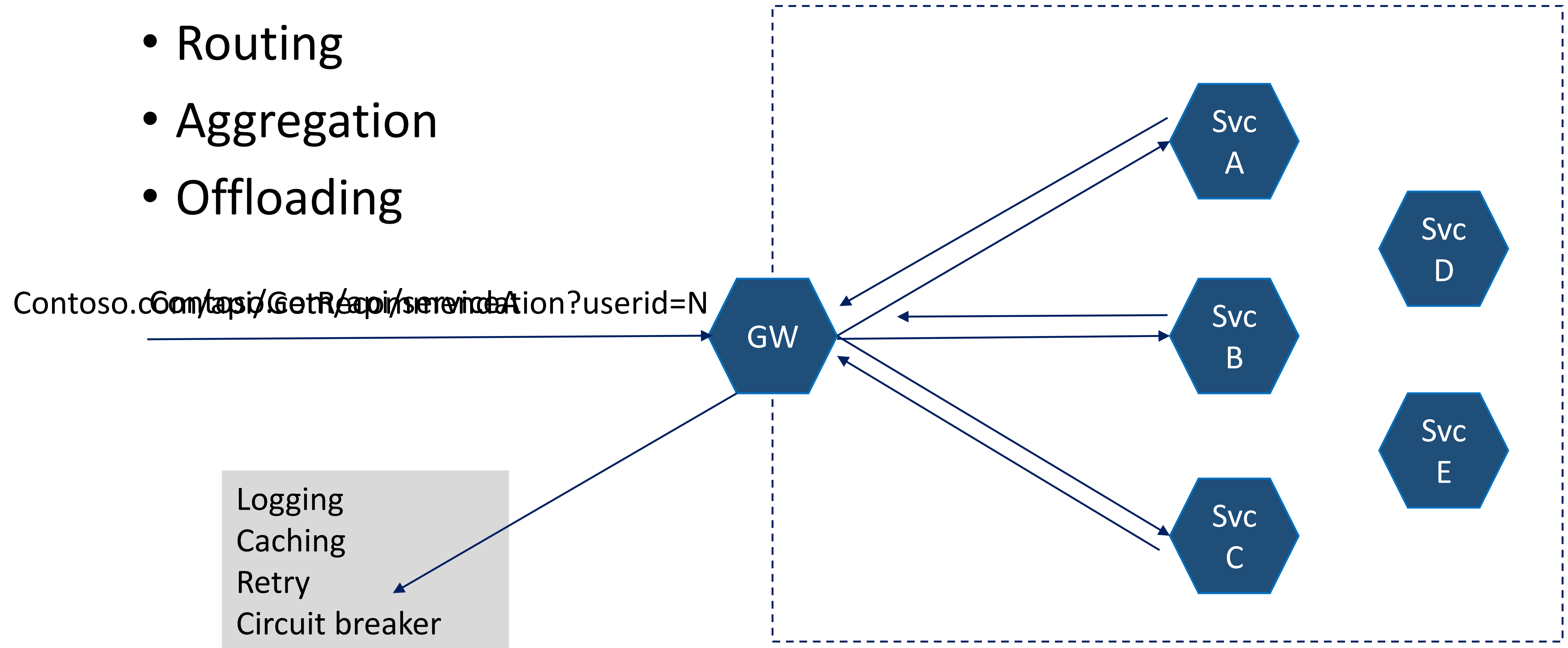
Istio linkerd

The solution is **Service Mesh**

patterns & practices
proven practices for predictable results

API gateway

- Routing
- Aggregation
- Offloading



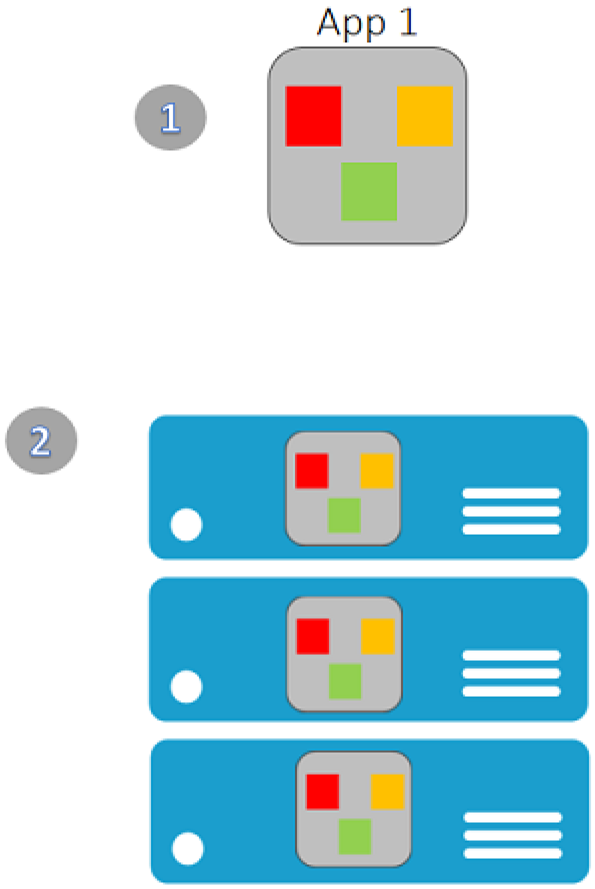
Logging
Caching
Retry
Circuit breaker
Throttling
SSL termination
Authentication

NGINX

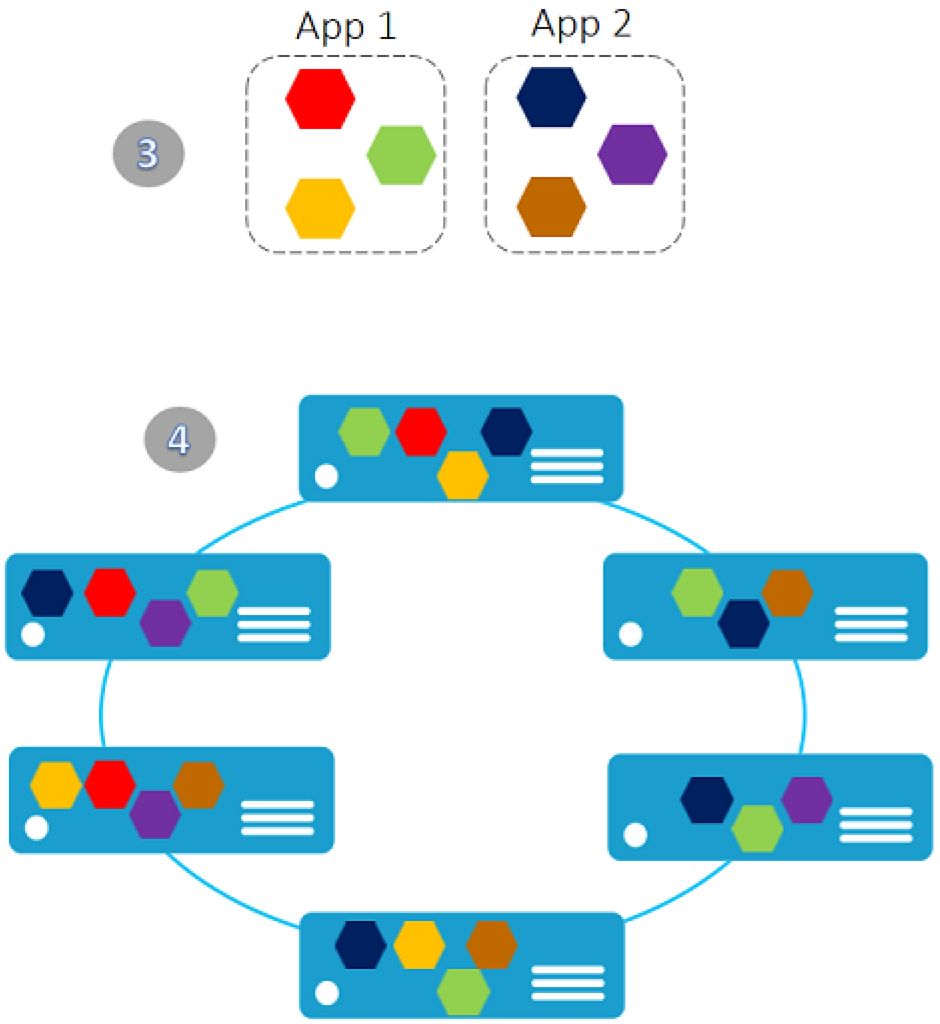
HAPROXY

træfik

Monolithic application approach

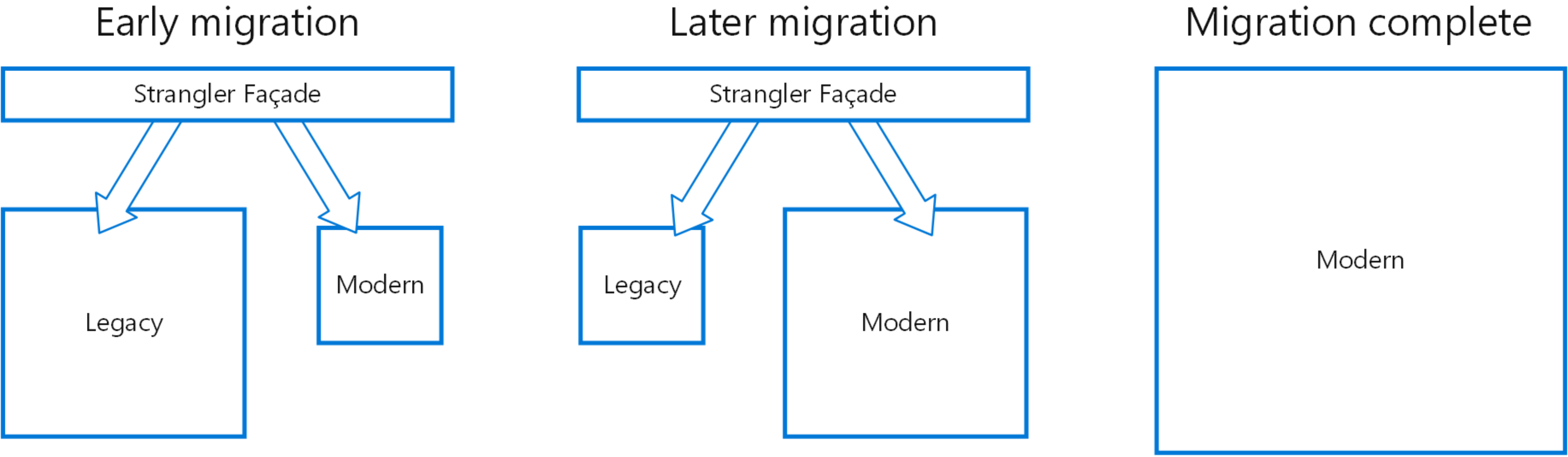


Microservices application approach



Consideration

Benefits	**Challenges**
Enables the continuous delivery and deployment of large, complex applications.	There is an additional complexity of creating a distributed system.
Improved maintainability: Each service is relatively small so it’s easier to understand and change.	Implementing requests that span multiple services is more difficult. Maintaining data consistency between service(s) is a challenge.
Better Testability: services are smaller and faster to test.	Testing the interactions between services is more difficult.
Better deployability: services can be deployed independently.	Increased operational and deployment complexity of deploying and managing a system comprised of many different services.
Each team can develop, test, deploy and scale their services independently of all of the other teams.	Implementing requests that span multiple services requires careful coordination between the teams.
Improved fault isolation.	Inter-service communication and dealing with partial failure implementation is challenging.
Eliminates long-term commitment to a technology stack.	Overhead of multiple JVM runtimes (or equivalent) and increase in memory consumption needs to be taken care of.



Early migration

Strangler Façade

Legacy

Modern

Later migration

Strangler Façade

Legacy

Modern

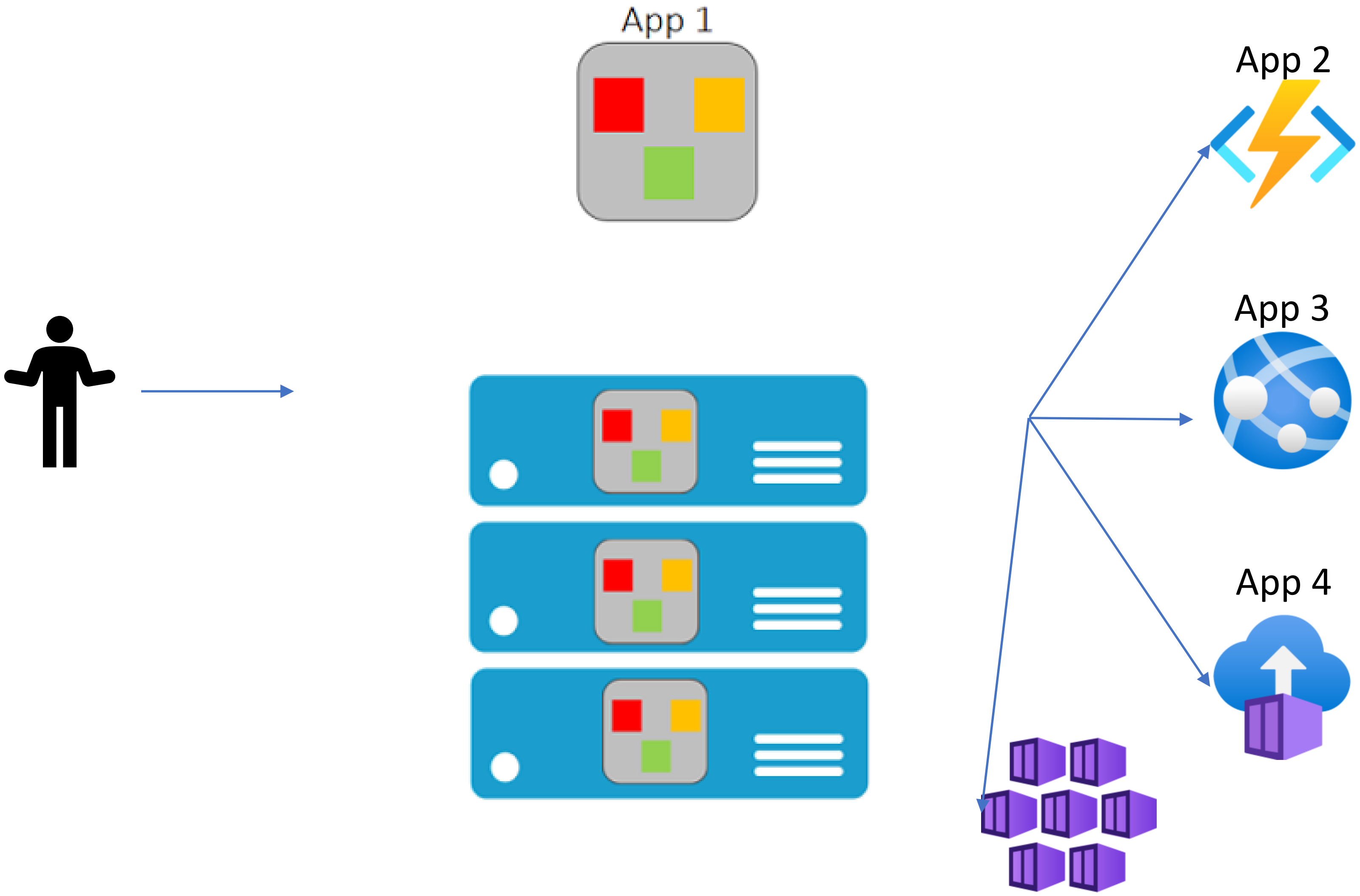
Migration complete

Modern

Which service to move first?

- High usage
- High availability needed
- Low effort to recode
- Or just start with new module!

Majestic Monolith



Let’s see it in action : Build and Deploy Microservices with project Tye

Prerequisites:

- Azure Subscription
- Dotnet SDK latest
- Visual Studio / Visual Studio Code
- Docker/Kubernetes Extension VSCode (Optional)

Github : https://aka.ms/AAbf8fq

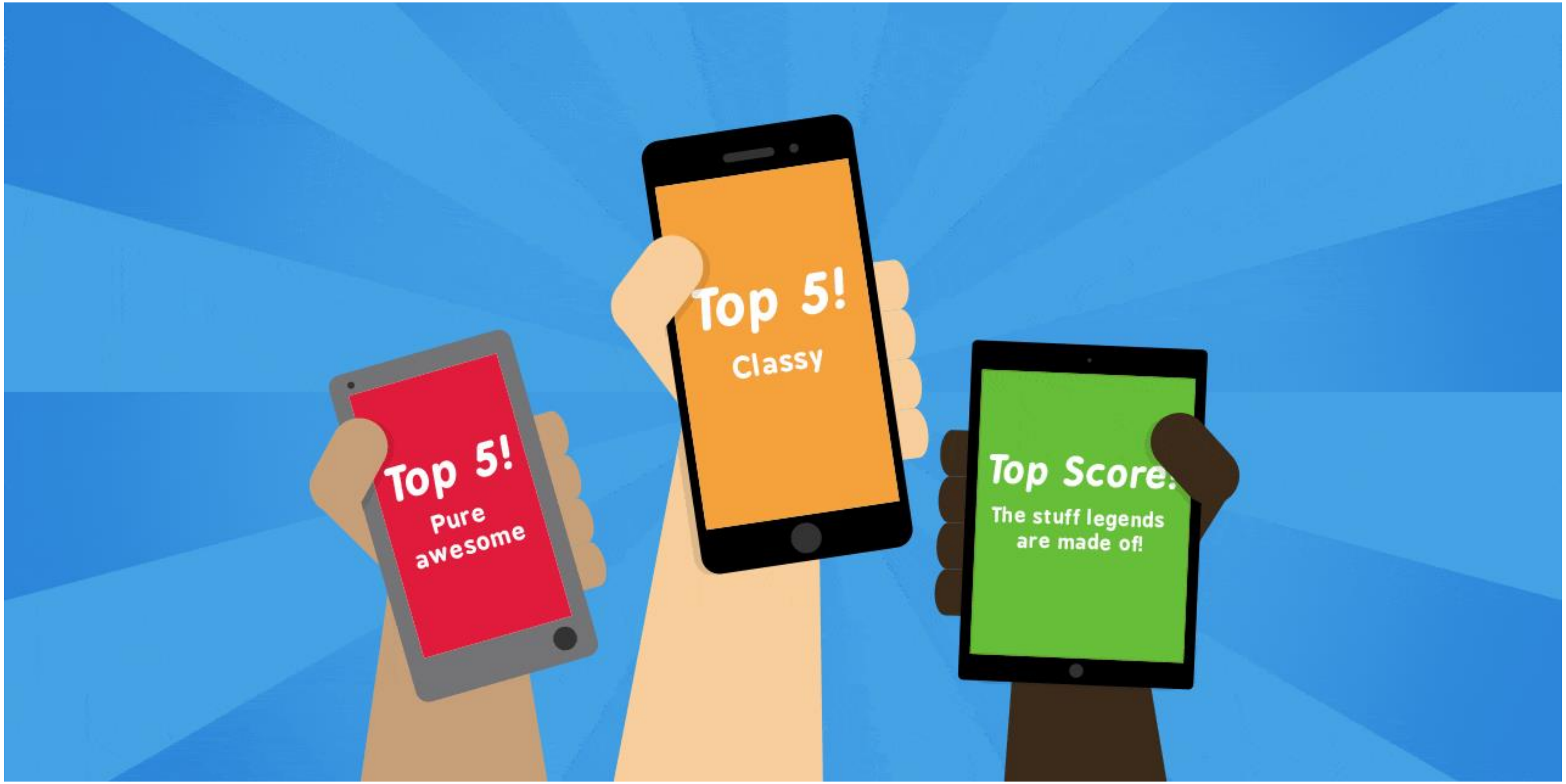
Design Principles

- **Design for self healing**. In a distributed system, failures happen. Design your application to be self healing when failures occur.
- **Make all things redundant**. Build redundancy into your application, to avoid having single points of failure.
- **Minimize coordination**. Minimize coordination between application services to achieve scalability.
- **Design to scale out**. Design your application so that it can scale horizontally, adding or removing new instances as demand requires.
- **Partition around limits**. Use partitioning to work around database, network, and compute limits.

Design Principles

- **Design for operations**. Design your application so that the operations team has the tools they need.
- **Use managed services**. When possible, use platform as a service (PaaS) rather than infrastructure as a service (IaaS).
- **Use the best data store for the job**. Pick the storage technology that is the best fit for your data and how it will be used.
- **Design for evolution**. All successful applications change over time. An evolutionary design is key for continuous innovation.
- **Build for the needs of business**. Every design decision must be justified by a business requirement.

Fun Time (Kahoot)



Your feedback is important

Please help us improve this program by completing this short feedback form.

https://aka.ms/saaslabfeedback2

If you’d like more help on your Azure modernization journey, please e-mail the SaaS Lab team

saaslab@microsoft.com

Thank you for being part of the SaaS Lab Program