



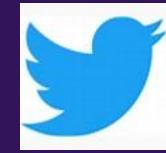
Journey to the Cloud

Azure Infrastructure, Platform, AI and Beyond...

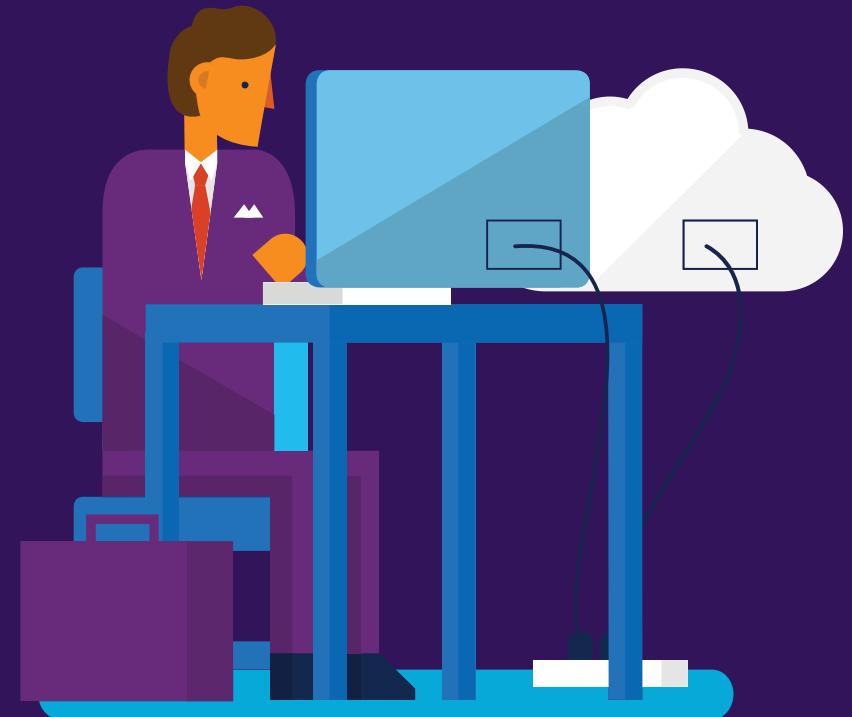
Will Eastbury, David Gristwood,
Gabriel Nepomuceno



<http://aka.ms/azureevent>



#AzureEvent
#BuildWithAzure



vipazure@microsoft.com

Introduction and Housekeeping

Fire Alarms – none today

Emergency Exits

Toilets

Lunch 1pm - Dietary Requirements?

Finish around 4pm

Feedback forms

1:1 sessions Thursday 31st

Are you a software partner and want to discuss these and other Azure topics in more depth? Has todays journey sparked off some thoughts

Limited number of 90 min slots available. Speak to Adam at the break

Today's brief – we shall ...

Show you some of the best ways to migrate and to design software for Microsoft Azure.

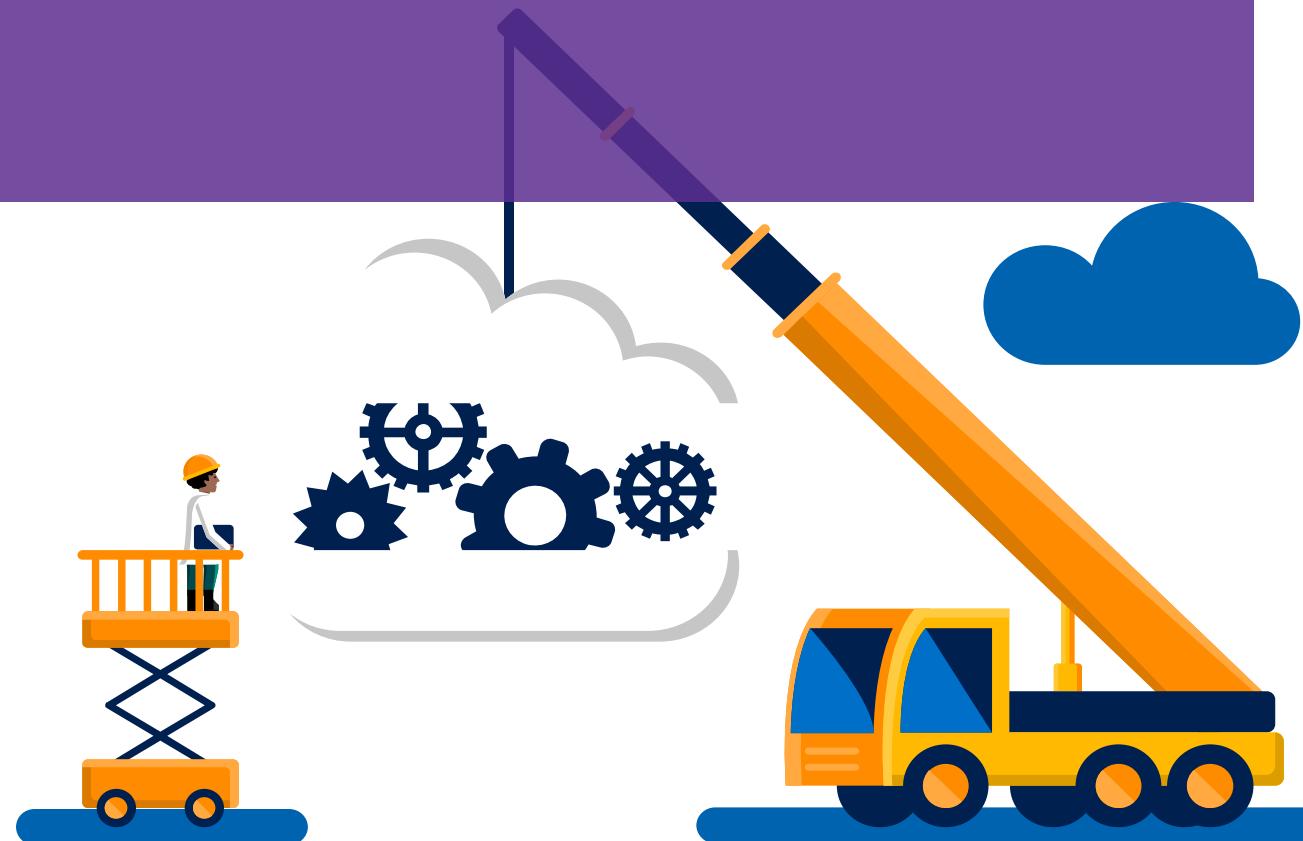
We will share the secrets of the well-trodden and some less-known pathways to design, build and migrate apps in the cloud when you deploy with Azure.

Talk a little about optimising existing solutions and systems for cost-efficiency and better margins when running on Azure.

Discuss when to use advanced services and concepts like Bots, AI and Cognitive Services.

Close the day with a showcase of some of the patterns and practices to use and the mistakes not to make whilst building and migrating your applications.

Why the journey through the cloud?



Edge DC

Application
Data
Middleware
Operating System
Virtualisation
Storage
Hardware
Networking
Power

IaaS

Application
Data
Middleware
Operating System
Virtualisation
Storage
Hardware
Networking
Power

PaaS

Application
Data
Middleware
Operating System
Virtualisation
Storage
Hardware
Networking
Power

SaaS

Application
Data
Middleware
Operating System
Virtualisation
Storage
Hardware
Networking
Power

Spin up Agility + Operational Efficiency

Legacy Support + Flexibility

SaaS Provider

Edge DC IaaS PaaS SaaS

Application Code	Application Code	Application Code	
Data Schema	Data Schema	Data Schema	
Middleware	Middleware		
Operating System	Operating System		
Virtualisation			
Storage			
Hardware			
Networking			
Power			

SaaS

BI/Data aaS

Big Data Storage
Big Compute
Analytics
Business Intelligence

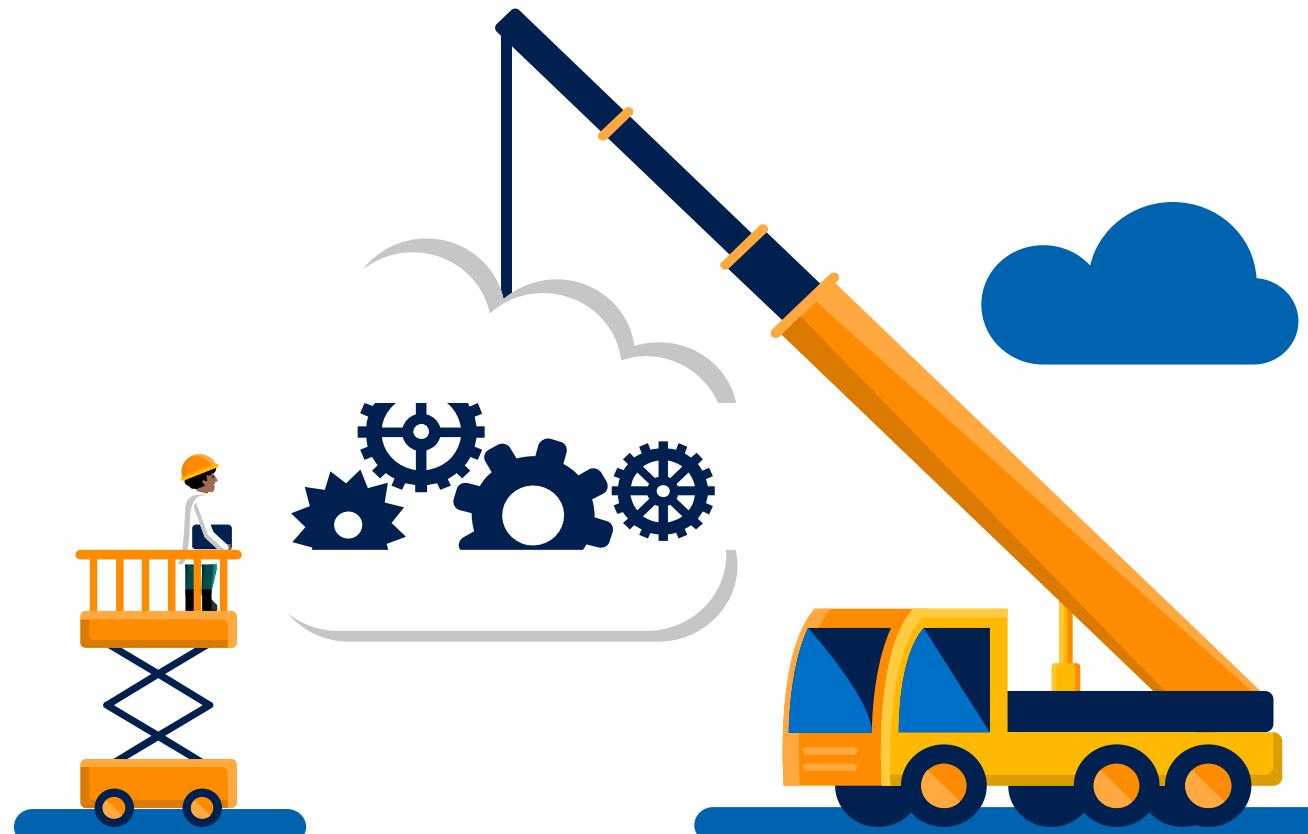
AI+ML aaS

App Intelligence
Machine Learning
Cognitive
Bots
Personal Agents

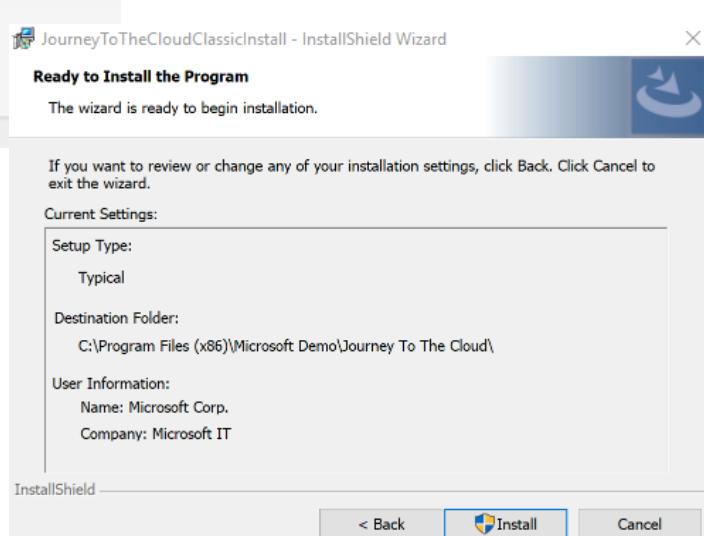
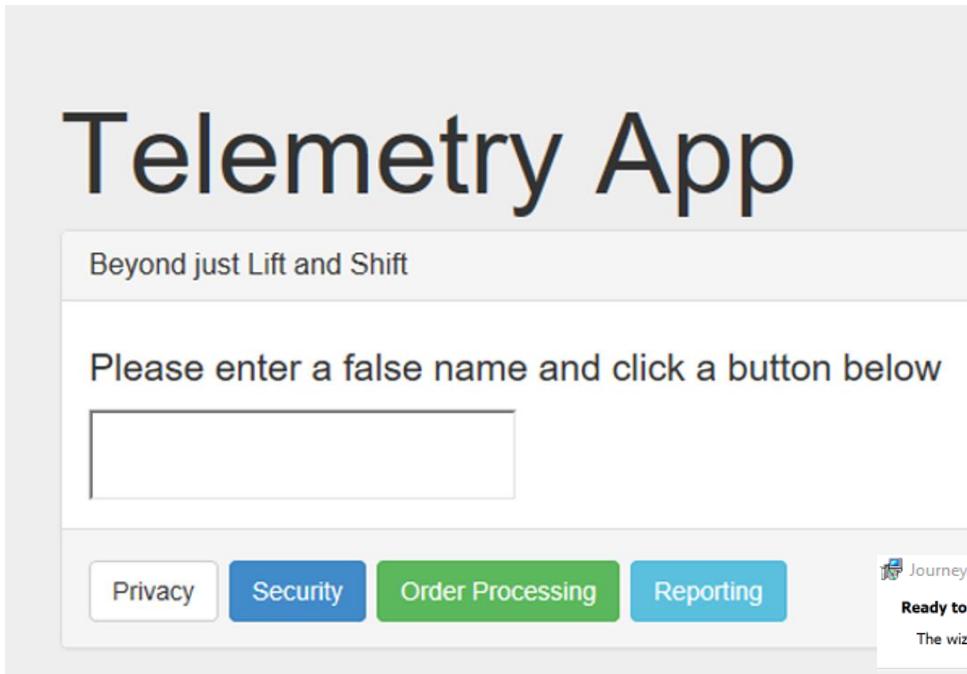
These are the basics, moving to the right

Enables the capability, time, resource and budget for this

So let's begin the Journey ...



Let meet our App...



- Classic web forms app
 - No real functionality, simply logs input field and button pressed
 - Log outputs data to a .dat file in ingress directory

■ Windows Service

- Watches for files in ingress directory
- Moves and updates them to egress directory

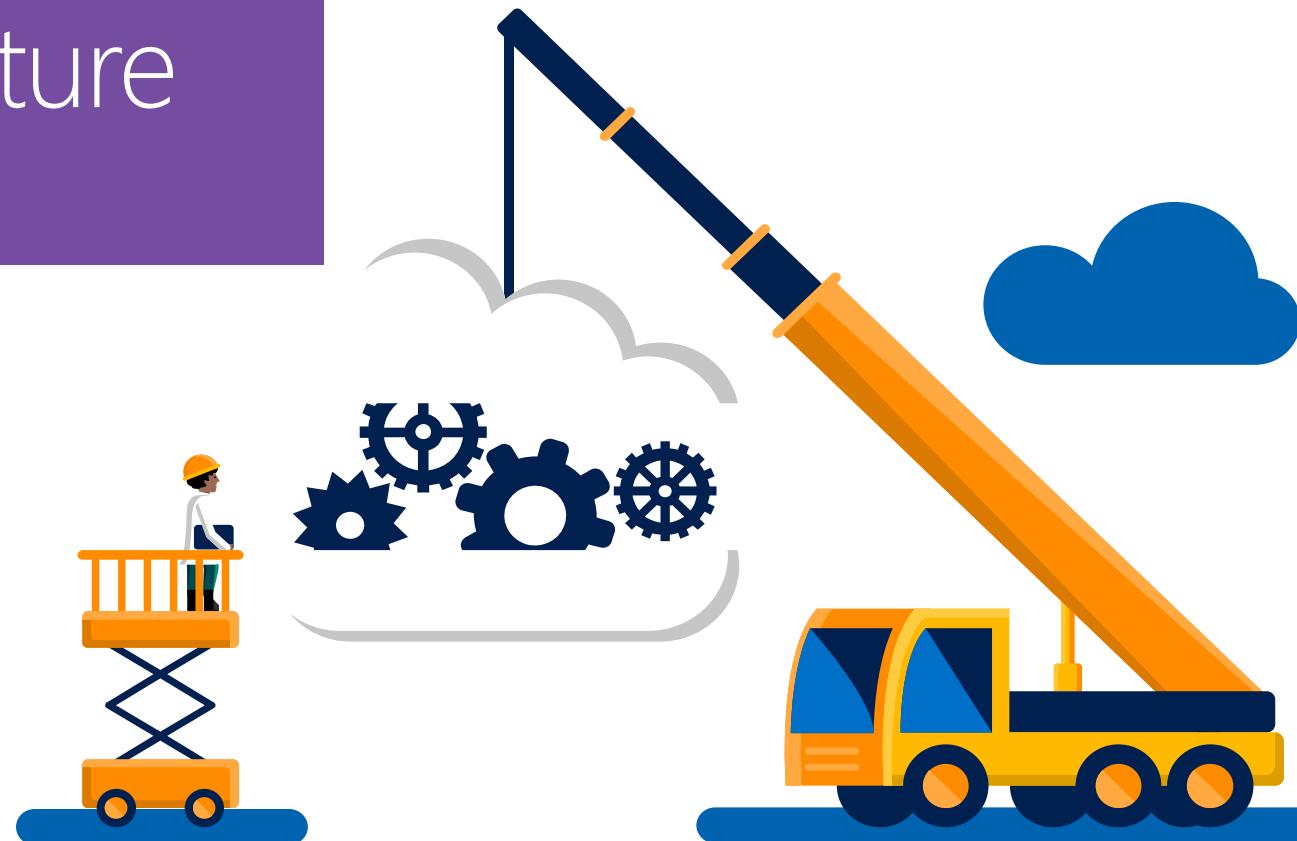
```
<appSettings>
<add key="ingresspath" value="C:\DataWriteArea\Ingress\" />
<add key="egresspath" value="C:\DataWriteArea\Egress\" />
<add key="ingresspattern" value="*.dat" />
</appSettings>
```

■ Started off life as MSI

Stage 1 – Lift and Shift, Edge DC to IaaS

Lift and Shift your applications
and move your infrastructure
to Azure

Will Eastbury
Technical Evangelist



'Lift and shift' to IaaS at speed

Templating and DevOps deployment automation game



1. Create an ARM Template for the infrastructure of my App
2. Deploy my IaaS services
3. Install the application stack using a DSC tool
4. Migrate my data
5. Use Traffic Manager to move the DNS seamlessly when ready

Lets talk a little about ARM, or Azure Resource Manager in terms of how this speeds things up and makes this simpler for you.

Using Azure Resource Manager and Resource Group Templates



What does Azure Resource Manager mean for me?

Azure Resource Manager enables you to work with the resources in your solution as a *related group of resources*.

The infrastructure for your application is typically made up of many components – maybe a virtual machine, storage account, and virtual network, or a web app, database, database server, and 3rd party services – you want to deploy Cookie-Cutter deployments by ‘stamp’ methodologies via templates, perfect for deploying micro-service environments.

People typically do not want to see these components as separate entities, instead they see them as related and interdependent parts of a single entity or application. People want to deploy, manage, and monitor them as a group – and to deploy, update or delete all of the resources for your solution in a single, coordinated operation.

Because resources are grouped at the control plane level, you can apply access control at the group level too, because Role-Based Access Control (RBAC) is natively integrated into ARM via Azure Active Directory.

When it comes to billing for resources, you can apply tags to resources to logically organize all of the resources in your subscription and groups – and these will show up in our billing APIs (Yes we have billing and ratecard APIs!) and reports to track your usage and costs.

ARM vs Other platforms

Other Platforms - Programmatic infrastructure deployment

"Dear AWS, here's [a list of everything I want you do], in order, in excruciating detail, and here's what to do differently if some of it already exists"

Azure ARM - Declarative end-state deployment

"Dear Azure, here's {what I want}, make it so"

Ways to fire ARM deployments in Azure

- Manually Through the ARM Portal (portal.azure.com)
- Through CI from Github, TeamCity, VSTS or other Source Control (youraccount.visualstudio.com)
- Through CD from VSTS, Jenkins, etc
- Directly, via the ARM REST APIs (management.azure.com)
- Through PowerShell or the Azure CLI

*So what can we
deploy ?*



Key Services in Azure IaaS



- Azure Virtual Machines
- Azure Virtual Machine Scale Sets
- Azure Traffic Manager
- Azure Virtual Network
- Azure Load Balancer
- Azure Storage (Blob, Disk, Table, Queue)

It's Demo Time:

On the 1st Step, You lifted and shifted the
{World}
(With an ARM template)

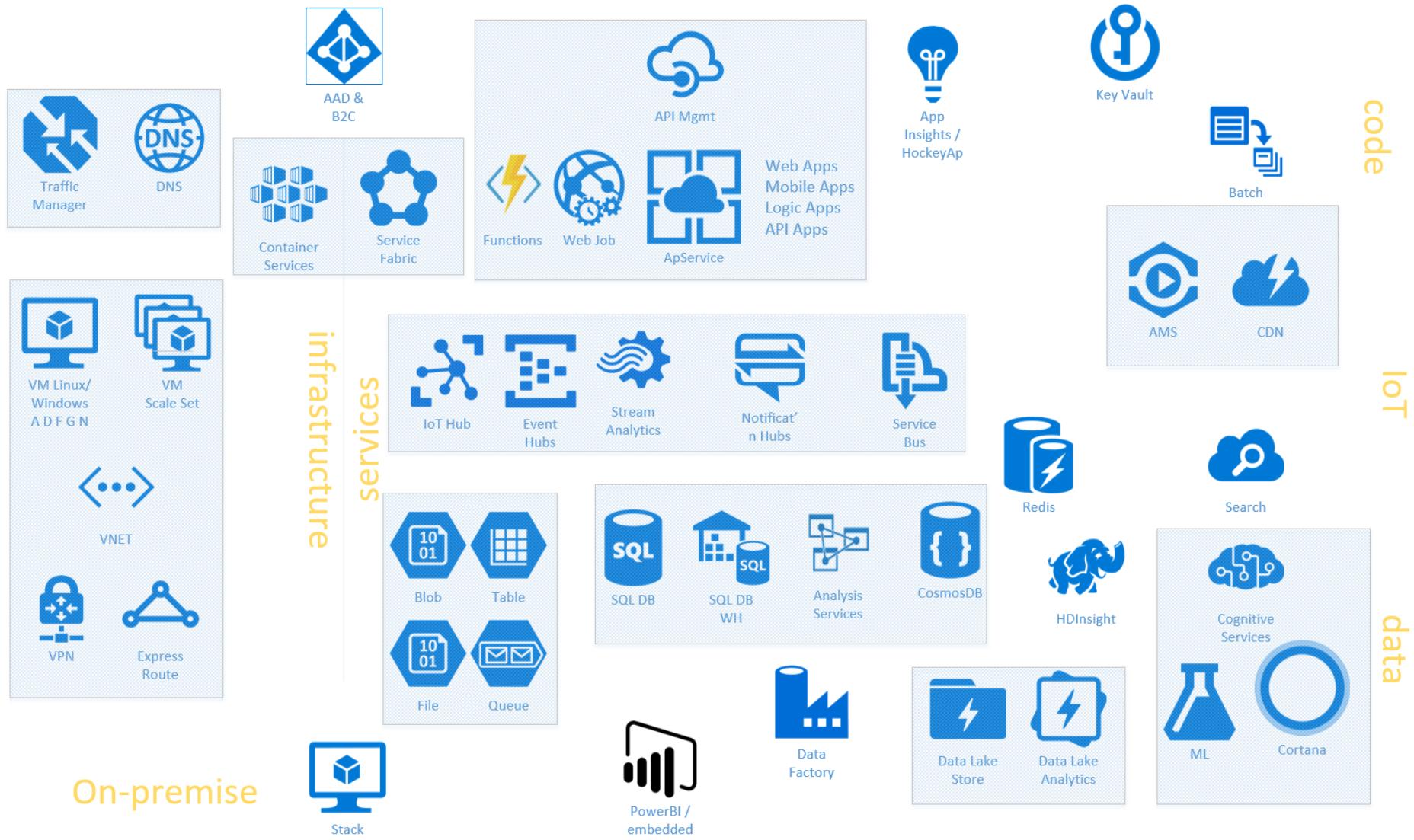
Stage 2 - Optimise for PaaS

*So what can we
optimise ?*

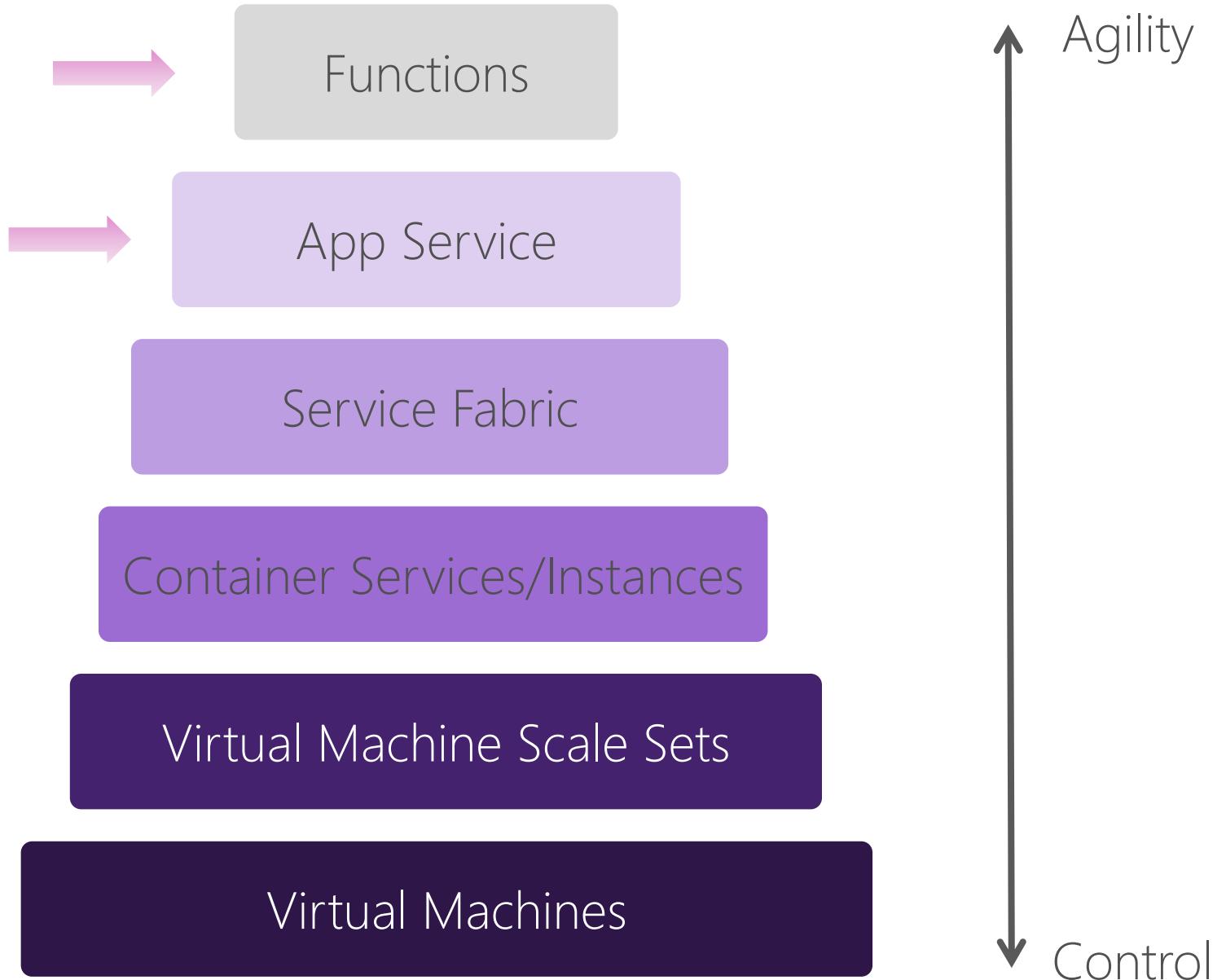
Will Eastbury &
David Gristwood
Technical Evangelists



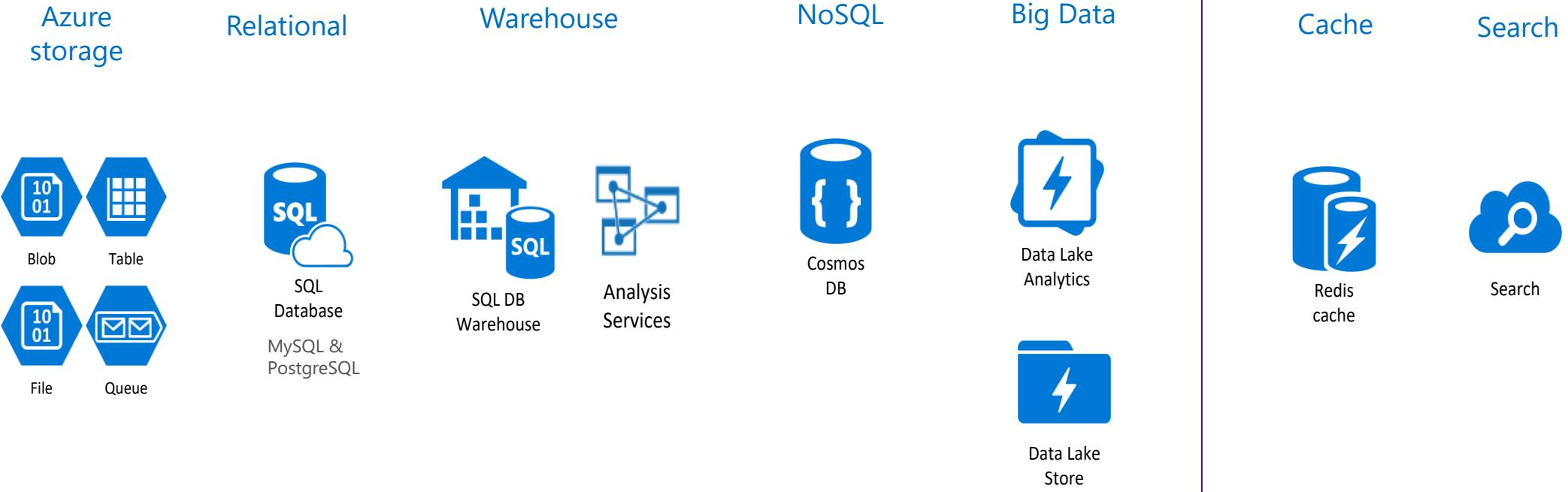
Azure Services



Code



Data



Polyglot Persistence:

- **Human derived data** (sales, orders, parts, invoices, inventory, etc)
- **Machine derived data** (telemetry, sensor, movement, GPS, etc)

Stage 2 – Optimize for PaaS

- Compile the IIS site using MSBuild
- Port the Windows Service into a webjob and add it to the site
- Template it
- Deploy it

It's Demo Time:

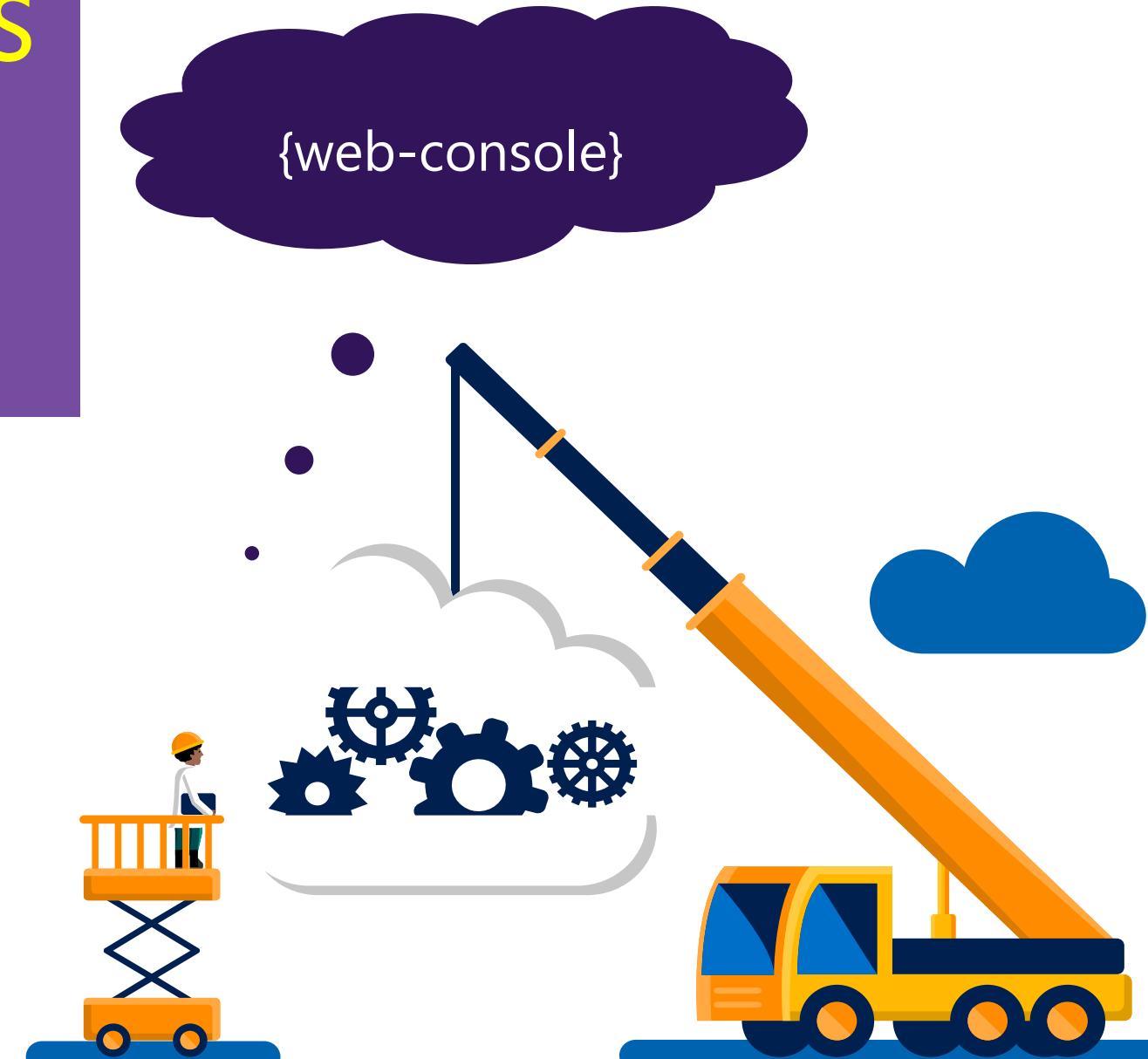
On the 2nd Step, You created the
{Webapp and
Webjobs}

(With an ARM template)

Stage 3 - Serverless

Where's my console?

Will Eastbury &
David Gristwood
Technical Evangelists



Key aspects of 'Serverless' technologies.

Technical

Deploy code that **just runs**, no servers (or containers) to patch, manage or maintain.

Business

Operate services that **just run**, no servers to pay for or monitor – just pay for the resources consumed via the operation of the service itself.

Lots of Azure services can be considered ‘serverless’

- Azure Functions (of course!)
- Azure Storage
- Azure CDN
- Azure Service Bus
- Azure Event Hubs
- Power BI Premium
- + Many, many more

Why would a Developer want to use serverless ?

Ultimate DevOps

Deploy your code package to a single location, deploy it once and see it scale out massively.

Ultimate Encapsulation and Testability

The logical end game when building applications on serverless is highly isolated nano (or) pico-services collected together into a logical API, whether that is RESTful or RPC/https or Service Bus/Queue based.

Ultimate Simplicity

Nothing to manage, the server architecture is all provided for you and scaled and managed on your behalf. You focus on the code and bindings and only the code and bindings.

Why would an IT Pro want to use serverless ?

Predictable and variable cost

My budget will stretch further and I can chargeback based on usage.

Hugely simplified management and ops landscape

No network / patch management on the servers, as there are no servers.

It's awesome for running operations and management + monitoring services too.

Now we can run monitoring and clean-up batch scripts and console apps without needing a permanently available server to run them on, what's not to like?

Azure Functions

Why Azure Functions ?

Charging model - Serverless or PaaS

Plan		
Consumption	Serverless	Includes a monthly free grant of 1 million requests and 400,000 GB-s of resource consumption per month. After that it's charged at Execution Time * ~£0.000012/GB-s (1.2 ten thousandths of a penny for a gigabyte of RAM used for 1 second) + Total Executions * ~£0.15 per Million invocations)(*)
App Service Plan	PaaS	Charged for dedicated server instances

Execution Model - Event driven or timed

Event Trigger	Use cases
http request	Web API
Storage queue Service bus	Queue based load levelling pattern + competing consumers
Timed execution	Simple polling / batch processing

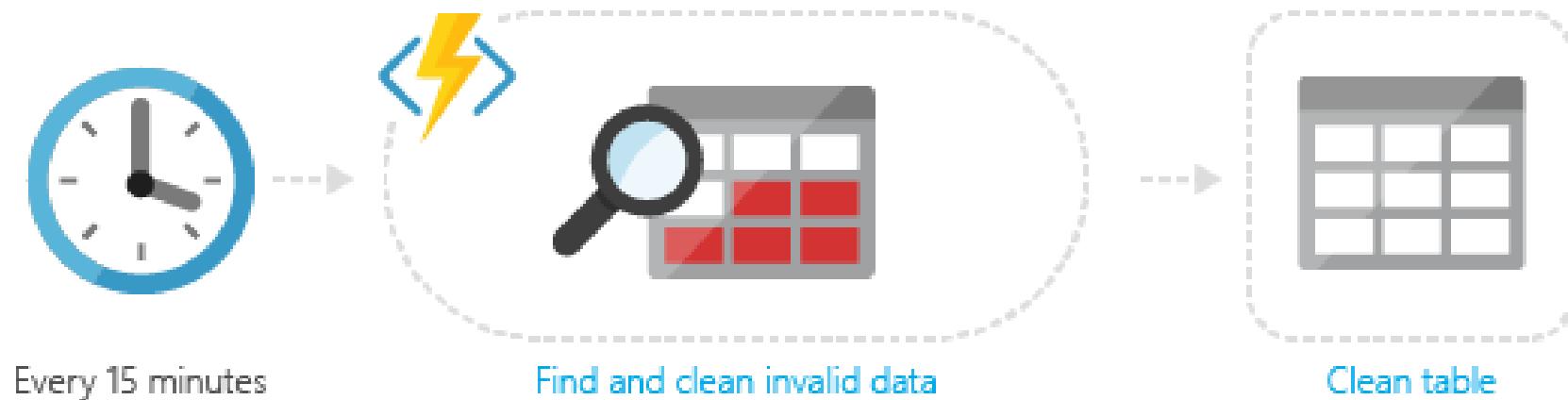
(*) E&OE - Pricing can vary based on time and region

Why Azure Functions ?

- Wide reaching language support
 - C#, F#, Python, JavaScript (Node.js), bash, CMD Batch, PowerShell, Custom EXE
- Large number of bindings supported
 - Timers, Event Hubs, queues, webhooks, https requests, Service bus, Blob Store, table store, SQL, mobile services, Twilio SMS, Bot Framework, SendGrid etc. etc. etc.
- Simple, proven deployment models
 - Under the covers this is Azure App Service
 - Use Web deploy to deploy a package of functions
 - Use FTP deploy to simply copy your function code up

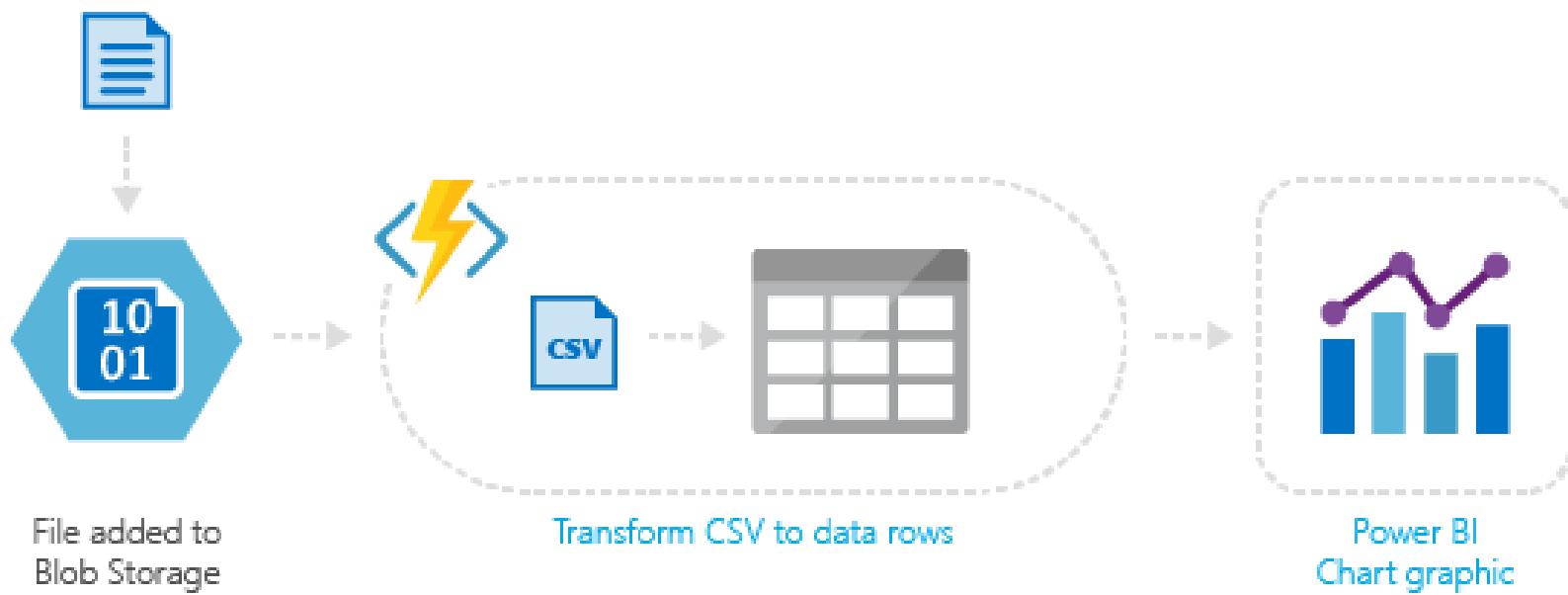
Common Scenarios for Functions

- Timer-based processing
 - Azure Functions supports an event based on a timer using CRON job syntax
 - For example, you could execute code that runs every 15 minutes and cleans up a database table based on custom business logic



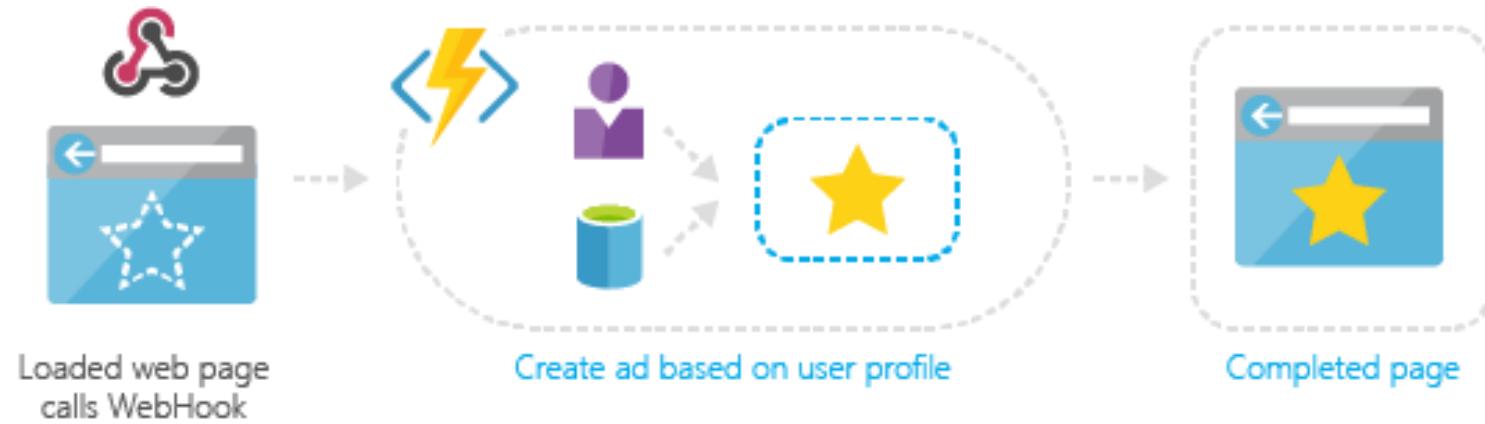
Common Scenarios for Functions

- Azure service event processing
 - Azure Functions supports triggering an event based on an activity in an Azure service
 - For example, you could execute serverless code that reads newly discovered test log files in an Azure Blob Storage container and transforms this into a row in an Azure SQL Database table



Common Scenarios for Functions

- Serverless web application architectures
 - Azure Functions can power a single page app
 - The app calls functions using the WebHook URL, saving user data and deciding what data to display
 - Or, you can do simple customisations, such as changing ad targeting by calling a function and passing it user profile information



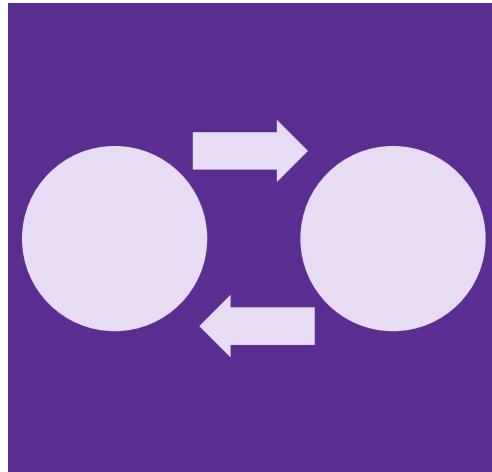
Common Scenarios for Functions

- Real-time bot messaging
 - Azure Functions can be used to customise the behaviour of a bot using a WebHook
 - For example, you can create an Azure Function that processes a message using Cortana Analytics and call this function using Bot Framework



Best practices for the Functions programming model

- Functions *should* “do one thing”
- Functions *should* be **idempotent**
- Functions *should* finish as quickly as possible



Stage 3 – Build with Serverless

- Create a Serverless Storage Account
- Create a Serverless Azure Functions App
- Port the Telemetry app to be an HTML/ JS SPA + Upload it to the blob storage area on our Storage Account
- Create a Nano-Service to port our service across and add new functionality.
- Push Some Data into our storage area
- Move on.

It's Demo Time:

On the 3rd Step, You just deployed
your code to

{Functions}

<https://aka.ms/blobapp>

Stage 4 – BI

Adding insight over code.

Will Eastbury & Gabriel
Nepomuceno
Technical Evangelists



Stage 4 – Integrate BI Services

- Take the telemetry from our service and add it to Power BI to build some dashboards and gain insights into usage.
- Add Automated Insights, to find things that are not immediately obvious from the raw data.

It's Demo Time:

On the 4th Step, You just gained

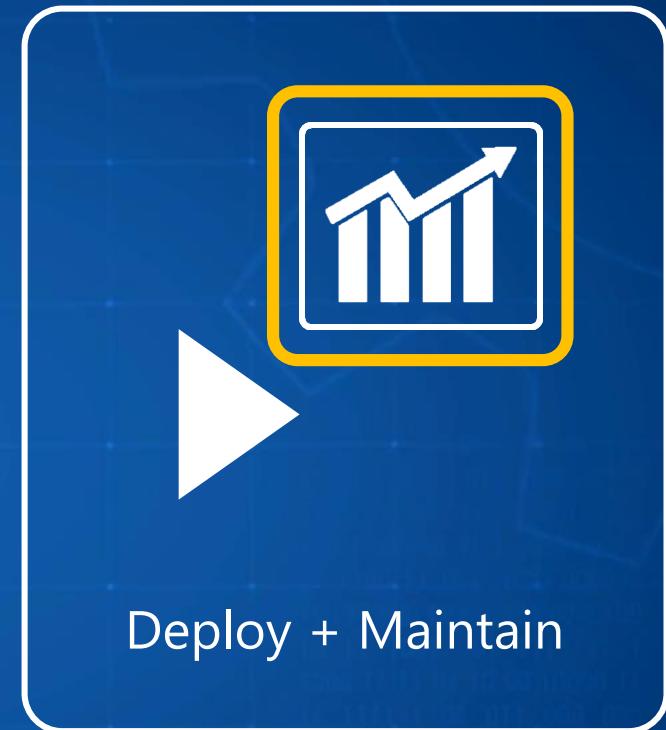
Insights in {Power BI}



Develop + Build

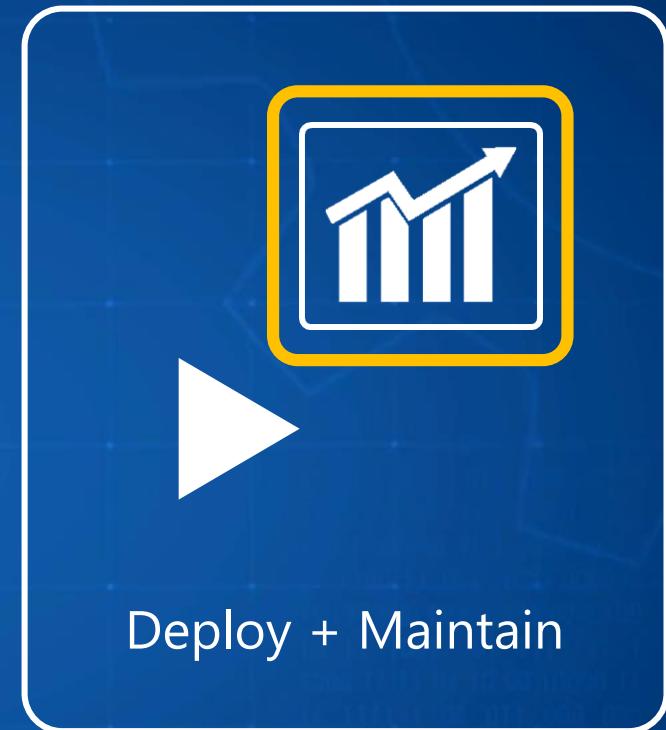
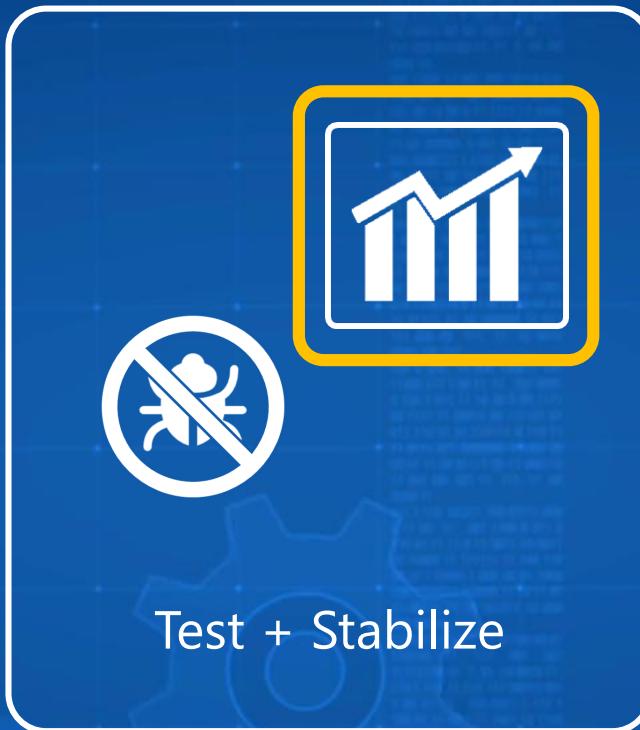


Test + Stabilize



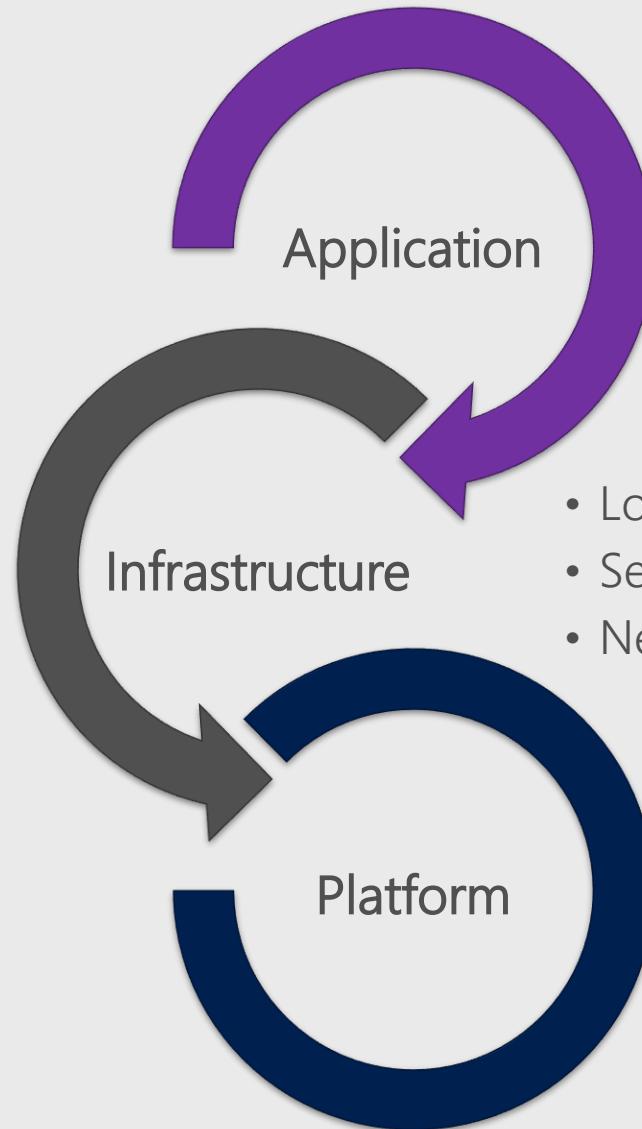
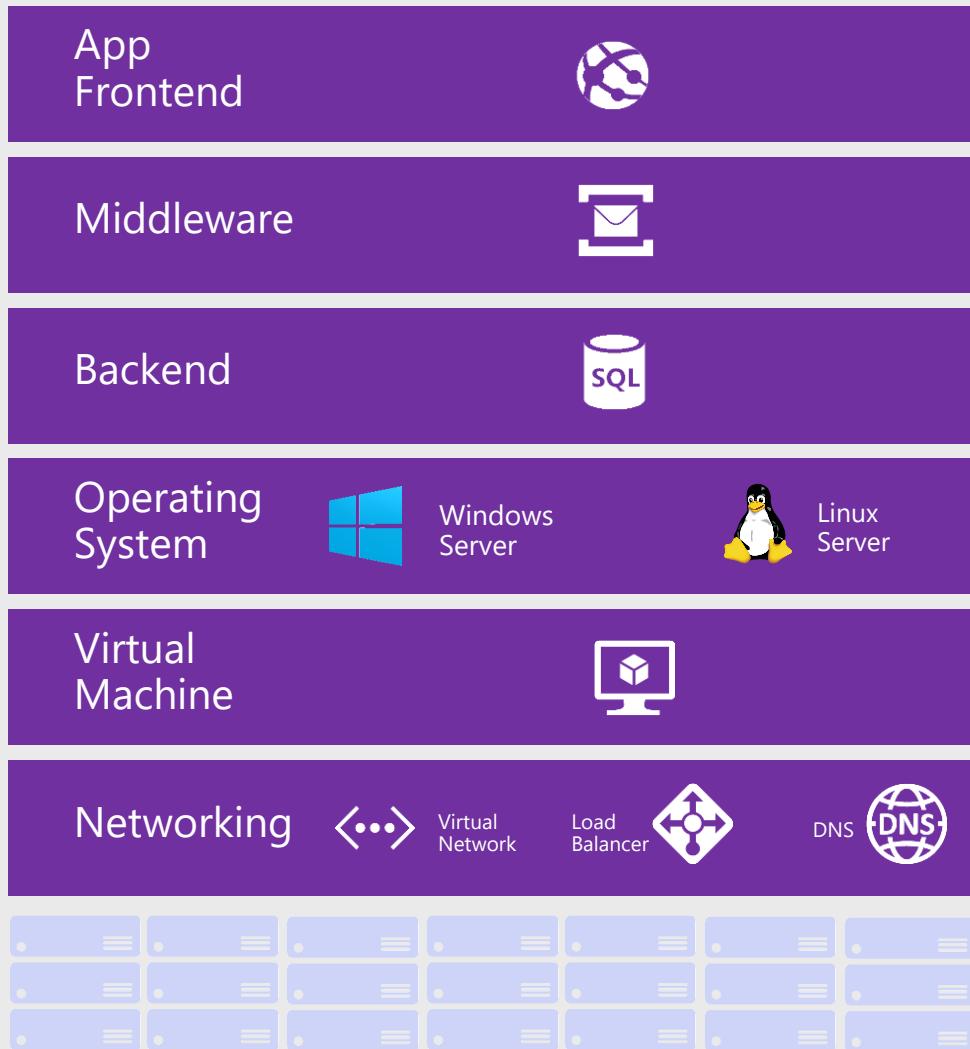
Deploy + Maintain

Traditional Monitoring



Continuous Monitoring

Bridging the Gap Between App and Infra



- Application Insights



- Log Analytics
- Service Map
- Network Watcher



- Azure Monitor
- Azure Advisor



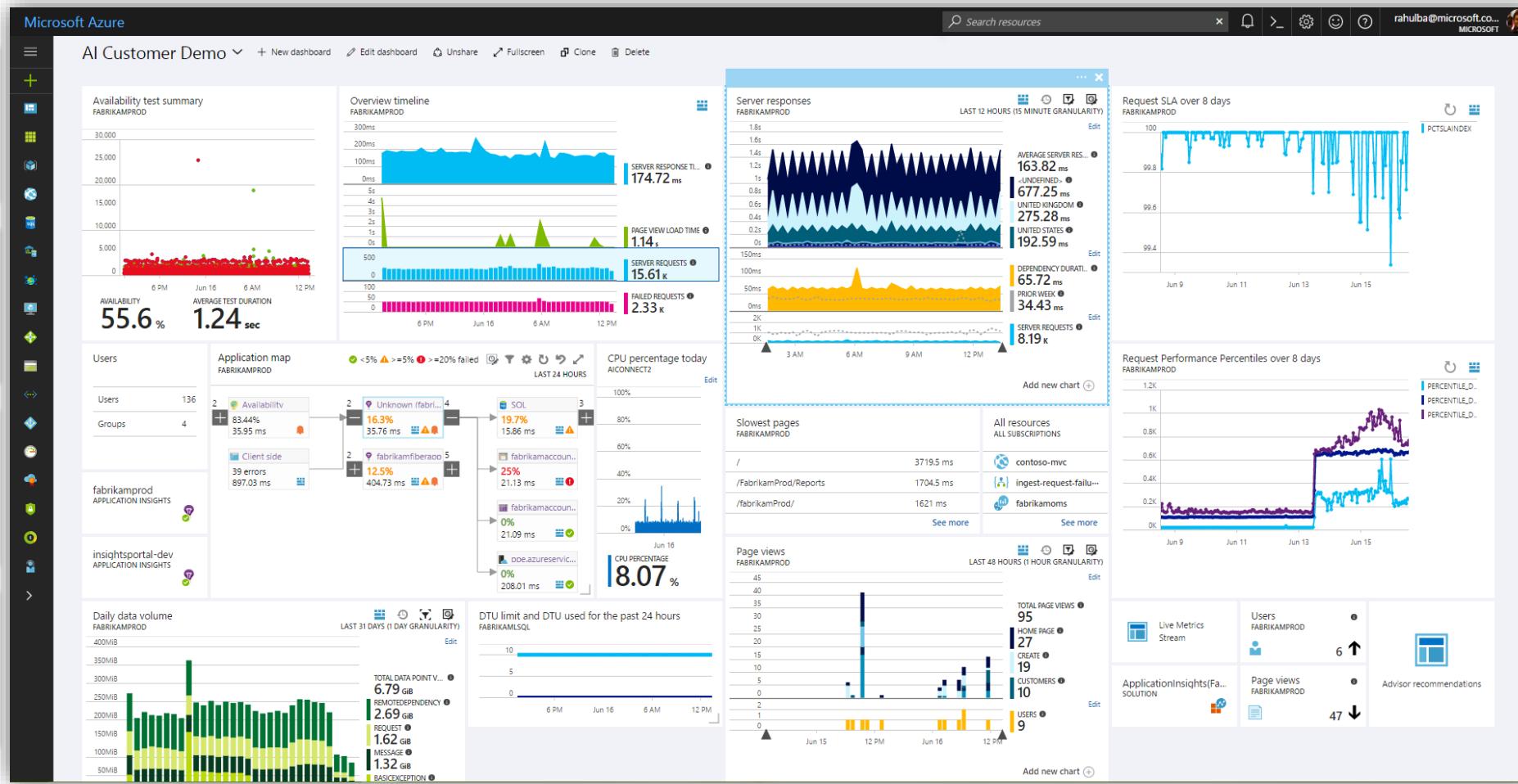
Azure Application Insights

Health Check

Monitor & Optimize

Detect & Debug

ML/Data Analytics



Generate Alerts

Take Actions

Customer Insights

Export & Correlate

It's Demo Time:

On the 5th Step, You just gained
Insights in {Your App}

Real Time Health Check

Live Metrics Stream
fabrikamprod

Incoming Requests

Requests/Sec

Request Duration (ms)

Requests Failed/Sec

Pin Pause 2 servers online

Sample Telemetry ▾

1:50:09 PM | Dependency | 409 | 51 ms @RD00155DA9A9C0
PUT fabrikamaccount | Container=fabrikamfiber
1:50:09 PM | Trace
New Request Received
1:50:06 PM | Request | 500 | 71 ms @AIConnect2
POST Customers/Create
1:50:06 PM | Exception | DbUpdateException An error occurred while updating the entries. The statement has been terminated.
1:50:06 PM | Dependency | 547 | 16 ms @AIConnect2
tcp:fabrikamxyz.database.windows.net:1433 | FabrikamSQL | ErrorMessage=The INSERT statement
1:50:06 PM | Trace
Starting to save New Customer
1:50:06 PM | Trace
New Request Received
1:50:02 PM | Request | 500 | 137 ms @AIConnect2
POST Customers/Create
1:50:02 PM | Exception | DbUpdateException An error occurred while updating the entries. The statement has been terminated.
1:50:02 PM | Request | 500 | 1.39 sec @RD00155DA9A9C0
GET Customers/Details
1:50:02 PM | Exception | FormatException Input string was not in a correct format.
1:50:01 PM | Dependency | 409 | 134 ms @RD00155DA9A9C0
POST fabrikamaccount/fabrikamfiber
1:50:01 PM | Dependency | 547 | 18 ms @AIConnect2
tcp:fabrikamxyz.database.windows.net:1433 | FabrikamSQL | ErrorMessage=The INSERT statement
1:50:01 PM | Trace
New Request Received
1:50:00 PM | Dependency | 409 | 99 ms @RD00155DA9A9C0
POST fabrikamaccount/Tables
1:50:00 PM | Trace
New Request Received
1:50:00 PM | Trace
@AIConnect2
Time 1:50:06 PM
Exception message An error occurred while updating the entries. See the inner exception for details. --> The INSERT statement conflicted with the CHECK constraint "chk_read_only". The conflict occurred in database "FabrikamSQL", table "dbo.Customers". The statement has been terminated.
Instance name AIConnect2
System.Data.Entity.Infrastructure.DbUpdateException: An error occurred while updating the entries. See the inner exception for details. --> The INSERT statement conflicted with the CHECK constraint "chk_read_only". The conflict occurred in database "FabrikamSQL", table "dbo.Customers". The statement has been terminated.
at System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean failed)
at System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj)
at System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataHandler, BulkCopySimpleResultSet bulkCopyHandler, TdsParserStateObject stateObj, Boolean& dataStream)
at System.Data.SqlClient.SqlDataReader.get_MetaData()
at System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, Boolean async, Int32 timeout, TaskCompletionSource< SqlDataReader> taskCompletionSource, String method, Boolean asyncWrite, Boolean iniets)
at System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, String method)
at System.Data.SqlClient.SqlCommand.ExecuteReader(CommandBehavior behavior, String method)
at System.Data.SqlClient.SqlCommand.ExecuteDbDataReader(CommandBehavior behavior)
at System.Data.Common.DbCommand.ExecuteReader()
at System.Data.EntityClient.EntityCommandDefinition.ExecuteDbDataReader(CommandBehavior behavior)
at System.Data.EntityClient.EntityContainer.ExecuteDbDataReader(CommandBehavior behavior)

Outgoing Requests

Dependency Calls/Sec

Dependency Call Duration (ms)

Dependencies Failed/Sec

Overall Health

Committed Memory (MB)

CPU Total (%)

Exceptions/Sec

select columns

Servers ⓘ

SERVER NAME	REQUESTS	REQUESTS FAILED	CPU TOTAL	COMMITTED MEMORY
AIConnect2	0/sec	0/sec	7%	1708 MB
RD00155DA9A9C0	0/sec	0/sec	0%	125 MB

Application Topology

fabrikamprod - Application map
Application Insights - Last 24 hours - fabrikamprod

Search (Ctrl+/
Time range Filters Options Refresh Restore defaults Learn more

Warning thresholds: <5% >=5% >=20% failed

Overview Activity log Access control (IAM) Tags

INVESTIGATE Application map Smart Detection Live Metrics Stream Metrics Explorer Metrics (preview) Availability Failures Performance Servers Browser

USAGE (PREVIEW) Users Sessions Events Retention Groups

Availability: 2 83.43% passing 35.96 ms

Unknown (fabrikam...): 4 9.8K requests 35.86 ms 16.4% failures

Client: fabrikamprod: 2 38 errors 831.33 ms 48 page views

fabrikamfiberapp: 5 5.9K requests 403.11 ms 12.5% failures

SQL: 3 3.7K calls 15.83 ms 19.6% failures

tcp:fabrikamxyz.dat...: 3 3.7K calls 15.82 ms 19.6% failures

tcp:fabrikamxyz.dat...: 10 calls 19.4 ms 0% failures

tcp:fabrikamxyz.dat...: 2 calls 16 ms 0% failures

tcp:fabrikamxyz.dat...: 2.2K calls 21.02 ms 0% failures

ppe.azureservicepro...: 96 calls 207.94 ms 0% failures

Unknown (fabrikamprod)

Top Errors See all errors

- GET Home/Index 1 problem, 3030 instances Failed Dependency: 409
- GET Customers/Details 3 problems, 2876 instances Failed Request: 500 System.FormatException at Fabrikam... Failed Dependency: 409
- POST ServiceTickets/Create 3 problems, 2172 instances Failed Request: 500 System.Data.SqlClient.SqlException a... Failed Dependency: 547
- GET Employees/Create 2 problems, 1727 instances Failed Request: 404 System.Web.HttpException at System...
- POST Customers/Create 3 problems, 9 instances
- GET Reports/Employees 3 problems, 8 instances
- GET /Content/fonts/segoewp... 1 problem, 7 instances
- GET /Content/fonts/segoewp... 1 problem, 7 instances
- GET robots.txt/Index 2 problems, 4 instances

Ad-Hoc Data Analytics

Application Insights fabrikamprod > Analytics Jeremy Demo +

Report bug ☺ 📁 ⓘ ⚙️ Rahul Bagaria

Home Page fabrikam_inci... Last 24 hours GO

SCHEMA FILTER

Filter by Name or Type...

COLLAPSE ALL EXPAND ALL

APPLICATION INSIGHTS

- traces
- customEvents
- pageViews
- requests
- dependencies
- exceptions
- availabilityResults
- customMetrics
- performanceCounters
- browserTimings

OTHER DATA SOURCES

Add new data source

```
//Latest 100 Records
requests | take 100

//Identifying too long requests with dynamic buckets
requests
| where timestamp >= ago(1d)
| extend responseBucket = iff(duration > 3000, "too long", "ok")
| project name, duration, responseBucket

//Am I meeting my SLA?
requests
| where timestamp >= ago(8d)
| summarize slaMet=count(duration < 3000), slaBreached=count(duration >= 3000), totalCount=count() by bin(timestamp, 1h)
| extend pctSLAIndex = slaMet*100.0 / totalCount
| project pctSLAIndex, timestamp
| render timechart

//Identifying spikes in request load
requests
| where timestamp between (datetime('2017-01-07')..datetime('2017-01-10'))
| summarize count() by bin(timestamp, 4h)
| render timechart
```

Completed 00:00:00.955 78 records loaded

TABLE CHART Line ▾ Timestamp ▾ Count_ ▾ Sum ▾

For Smart Diagnostics click on a highlighted data variation ⓘ

Legend count_

The chart displays a single data series named 'count_'. The Y-axis represents the count, ranging from 0 to 8k. The X-axis represents the timestamp in UTC, from January 8 to January 19, 2017. The data shows a sharp increase starting around January 11, peaking at approximately 7500 on January 15, and then fluctuating between 6000 and 7000 for the remainder of the period.

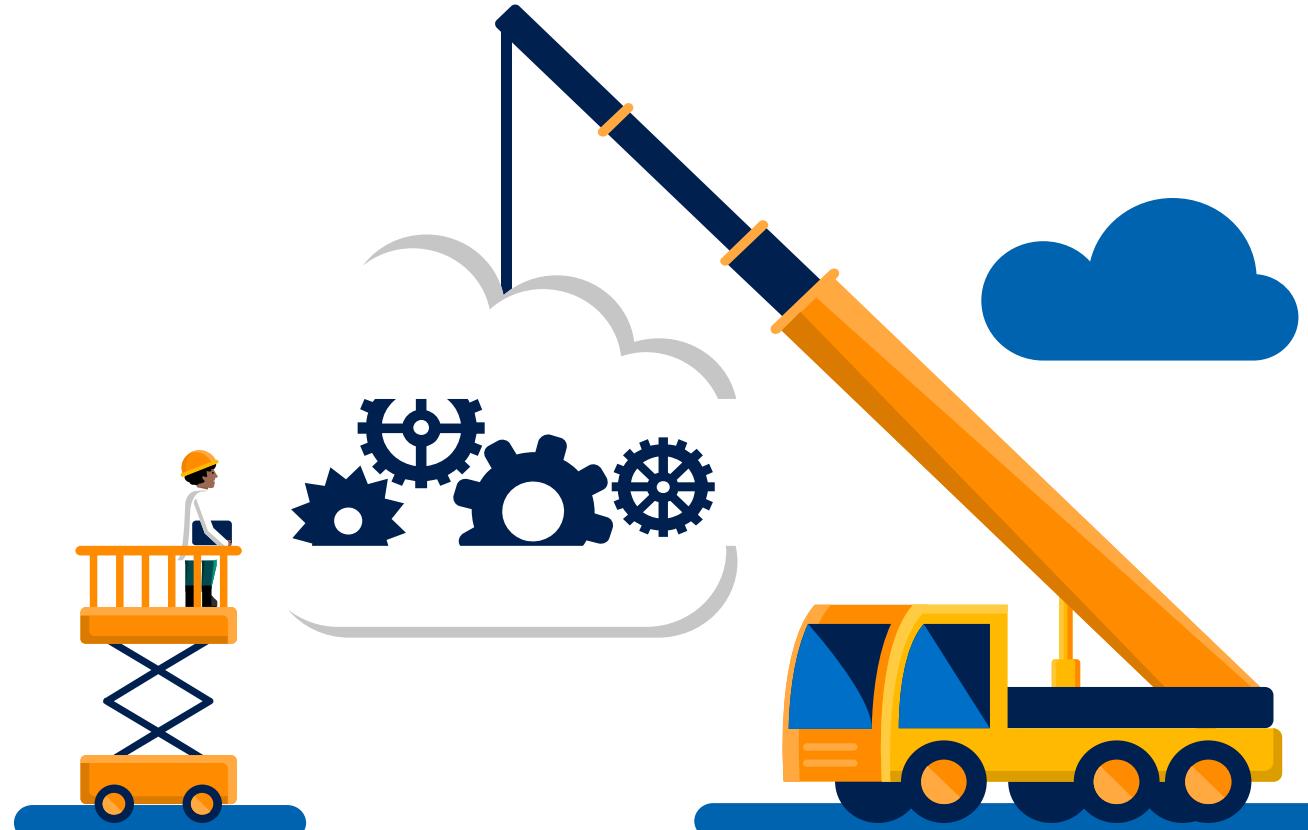
Timestamp [UTC]	Count_
2017-01-08 00:00	~1500
2017-01-09 00:00	~500
2017-01-10 00:00	~500
2017-01-11 00:00	~500
2017-01-12 00:00	~6500
2017-01-13 00:00	~6800
2017-01-14 00:00	~5500
2017-01-15 00:00	~7500
2017-01-16 00:00	~6200
2017-01-17 00:00	~6800
2017-01-18 00:00	~7000
2017-01-19 00:00	~6000
2017-01-20 00:00	~7000

timestamp [UTC]

It's Demo Time:

On the 6th Step, You just gained Insights in
{Deep data
analytics}

What's next? Let's continue the Journey and introduce AI ...



Stage 5 – AI & ML

"I'm sorry, Mike. I'm afraid I can't do that."

Mike Ormond &
Gabriel Nepomuceno
Technical Evangelists



$$7 \times 6 = ?$$

$$75 \times 75 = ?$$

$$569 \times 176 = ?$$







AI SOLUTIONS

Planning &
Scheduling

Machine
Learning

Pattern
Analysis &
Recognition

Robotics

Vision

Natural
Language

Speech
Recognition

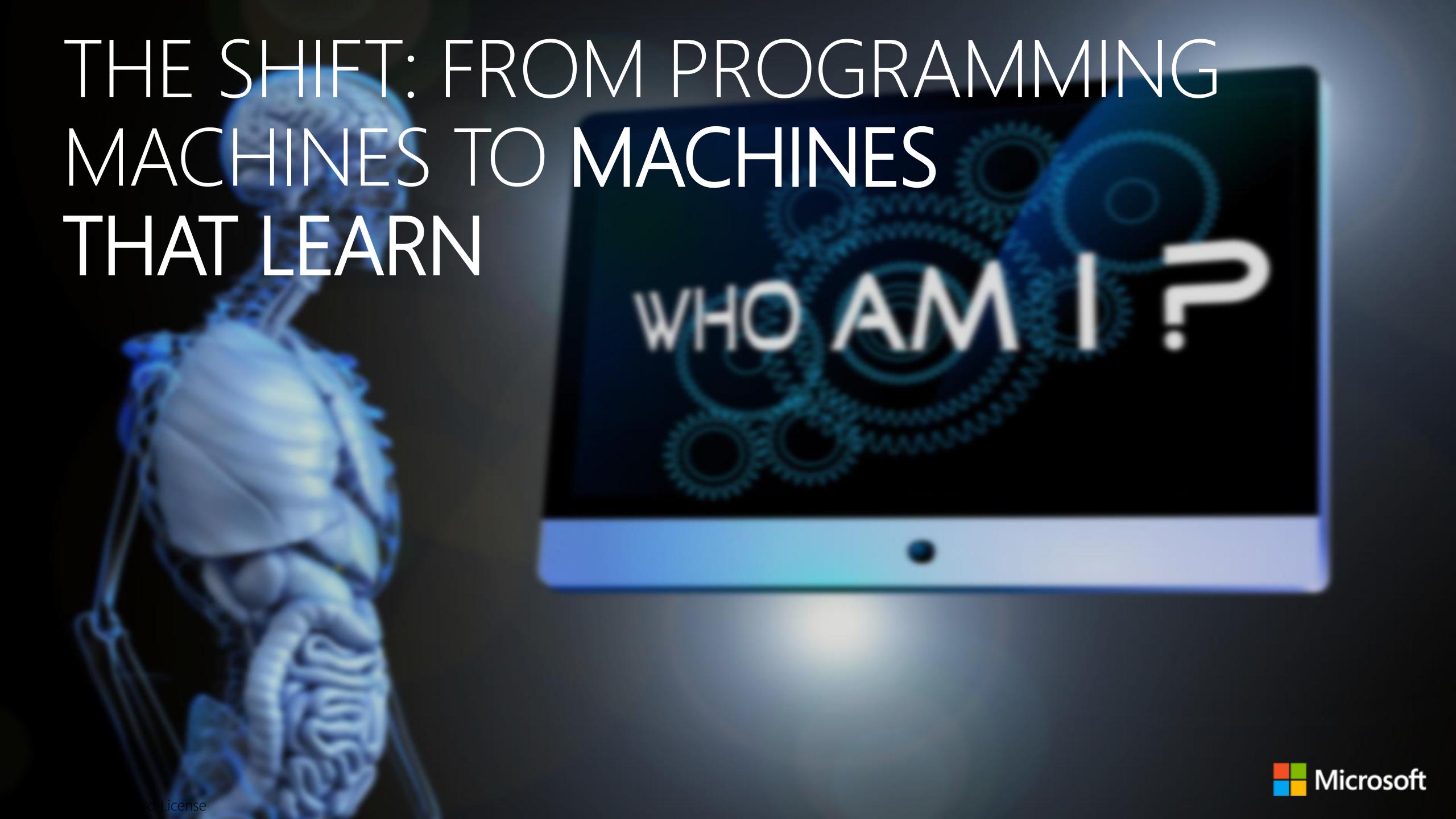
Knowledge
Representation
& Reasoning

Swarms & Self-
organising
Systems

Autonomous
Systems

"Amplify human ingenuity"

THE SHIFT: FROM PROGRAMMING MACHINES TO MACHINES THAT LEARN

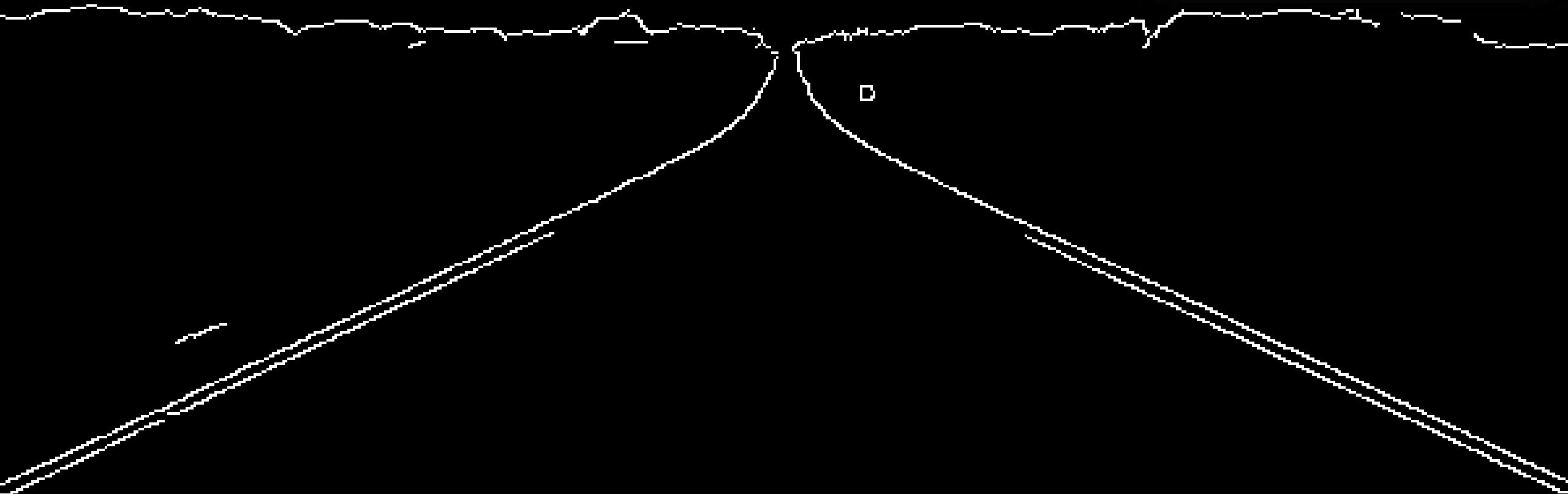


WHO AM I ?

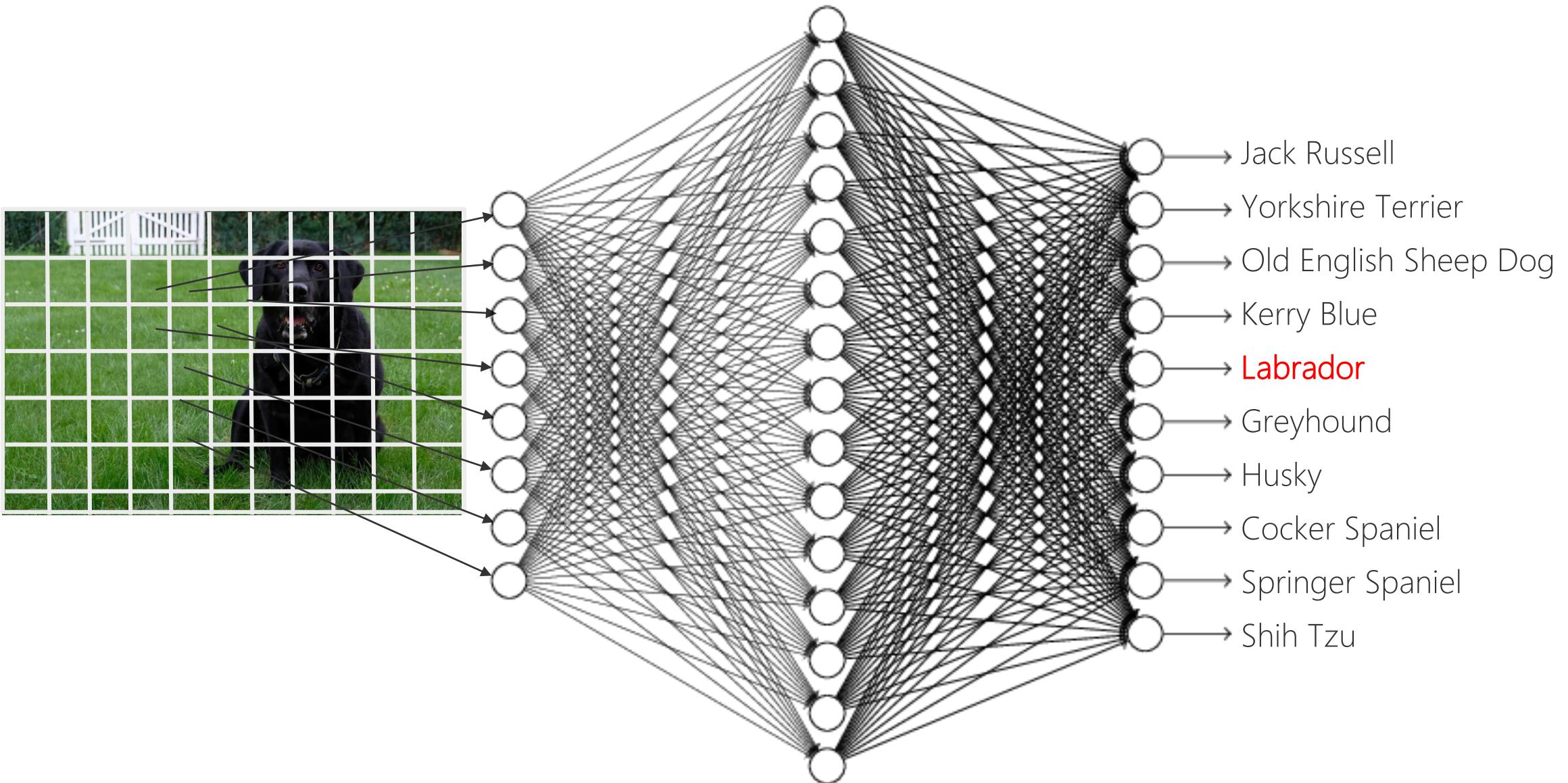
MACHINE LEARNING FOR DOGS



Classical AI



ARTIFICIAL NEURAL NETWORKS





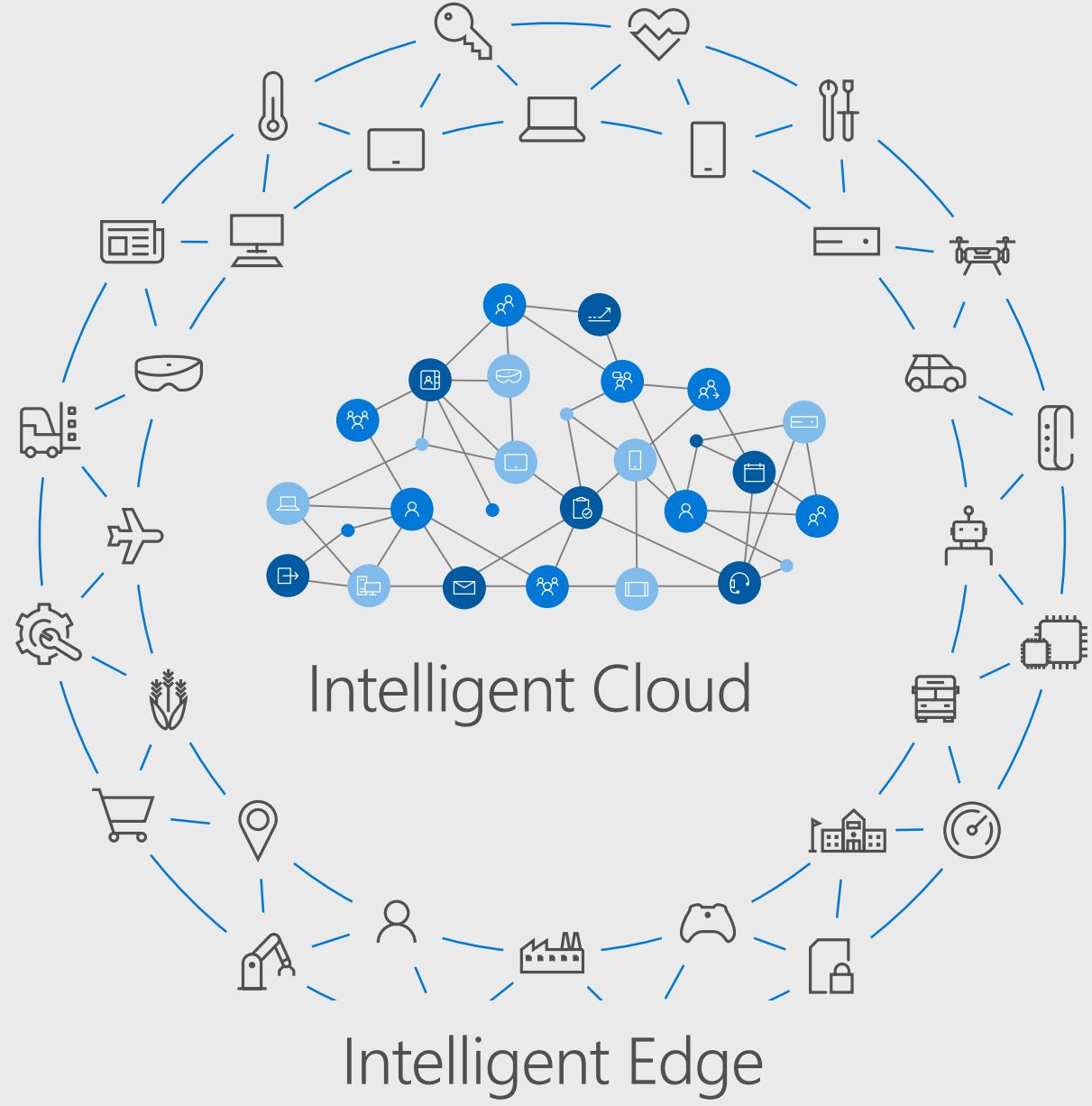
Multi-device

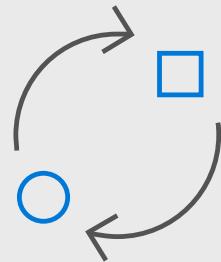


Artificial Intelligence

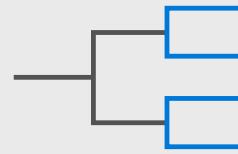


Serverless





Big
compute



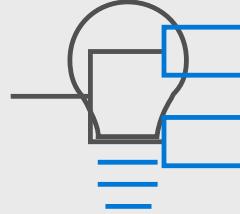
Powerful
algorithms

101010
010101
101010

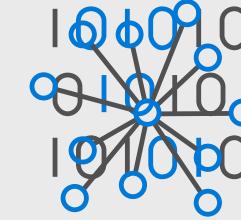
Massive
data



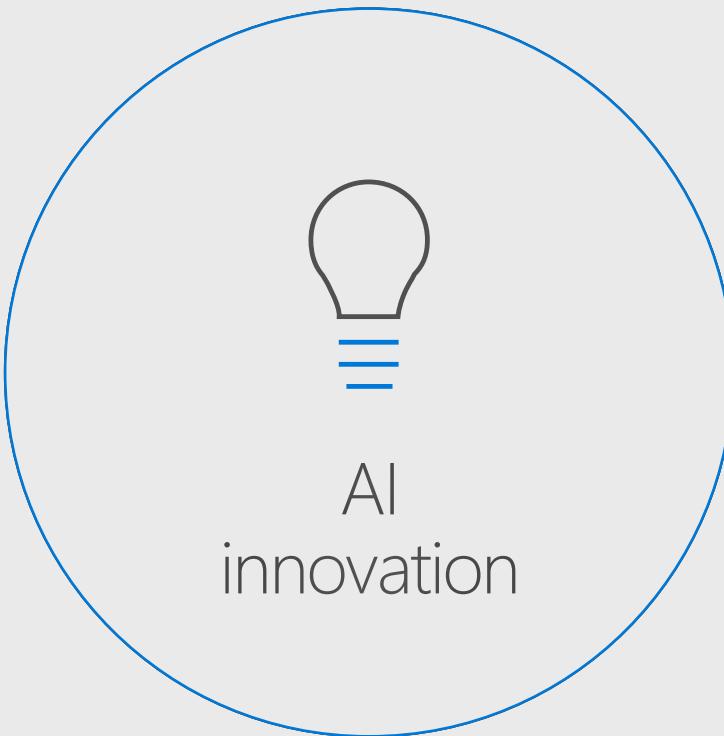
Microsoft
Cognitive



Powerful
algorithms



Microsoft
Graph

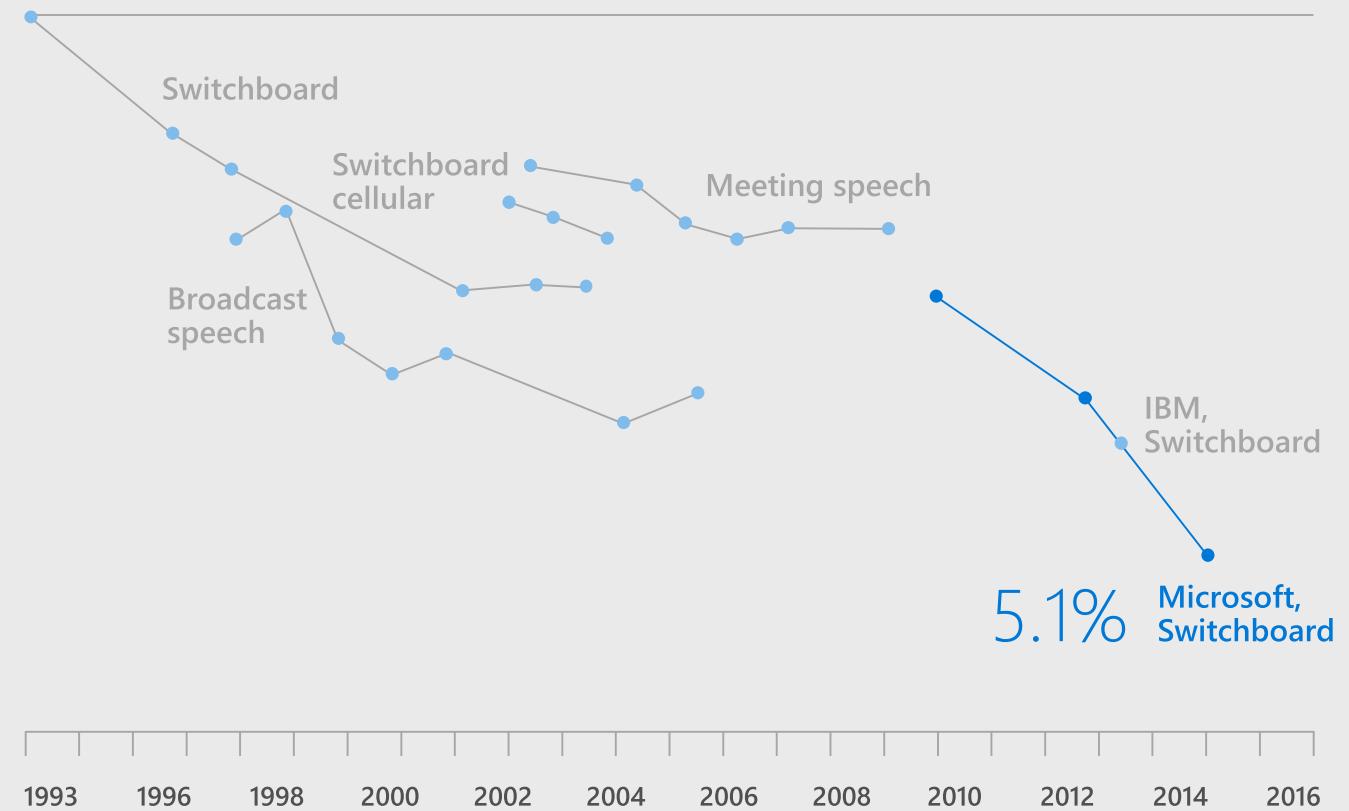


Breakthroughs in speech

5.1% human error rate

Switchboard

Loud and clear
Speech-recognition word-error rate, selected benchmarks, %



Sources: Microsoft: research papers

Bringing AI to every developer



Stage 4 – Integrate AI Services

- Take the function app and implement a new feature with Cognitive Services LUIS.

It's Demo Time:

On the 7th Step, You just added
conversational AI with {LUIS}

Let's talk patterns and best practices, things to look out for and things to avoid

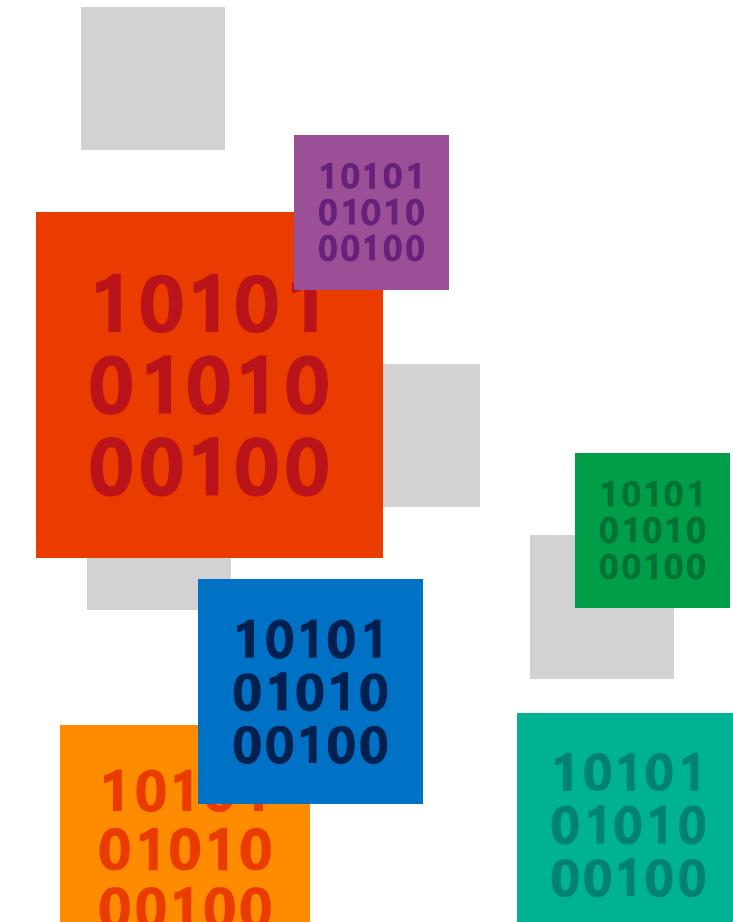
Will Eastbury
and David Gristwood
Technical Evangelists



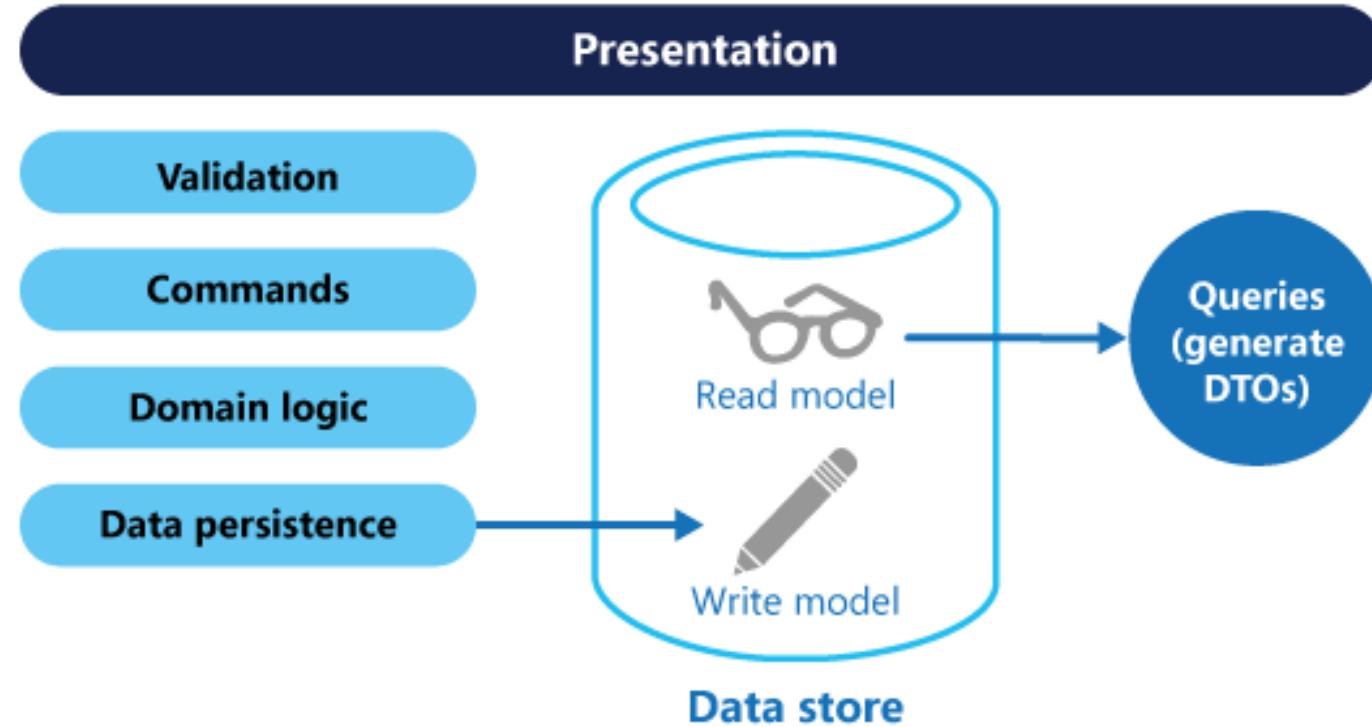
Architecture Patterns

Cloud Design Patterns

[https://docs.microsoft.com/azure/
architecture/patterns/](https://docs.microsoft.com/azure/architecture/patterns/)

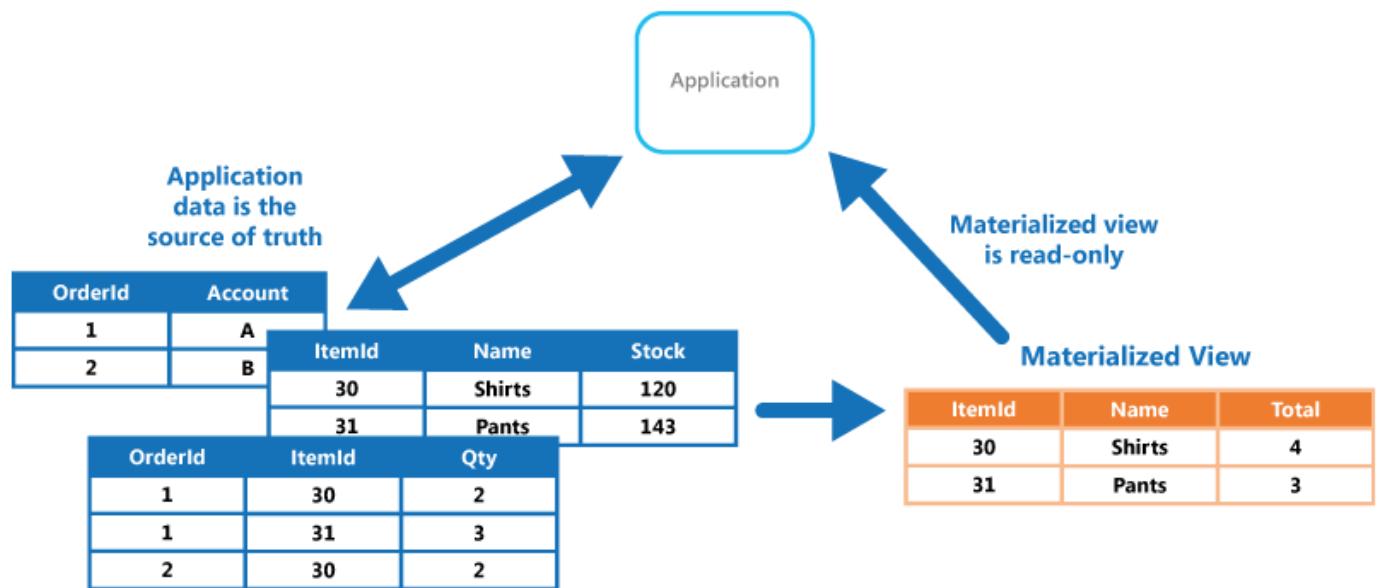


“Command and Query Responsibility Separation”



- *“Separate reads from writes”*
- *Trade off of complexity for performance*
- *Avoid multi-master issues in geo-distributed systems*

Materialized View pattern

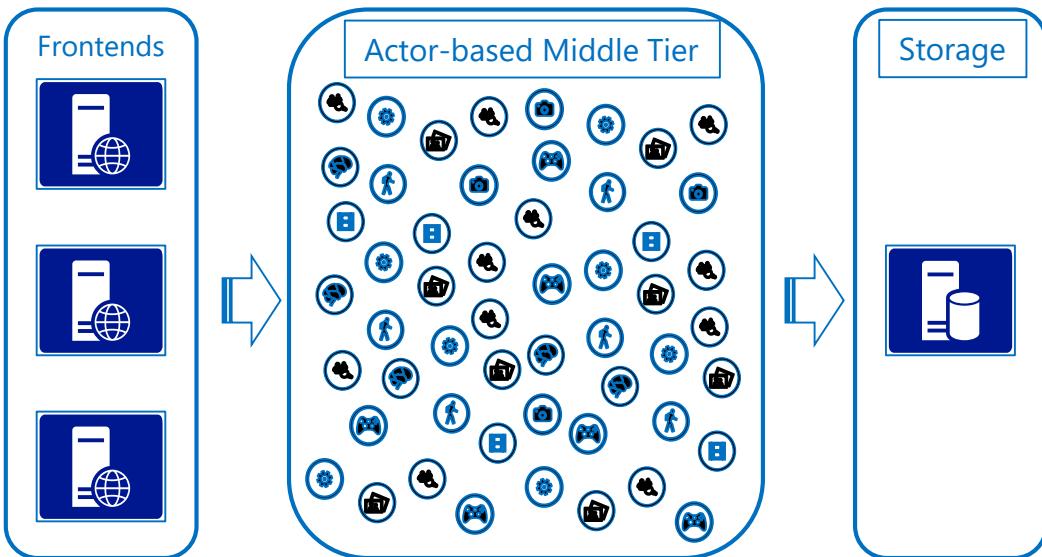


Data often optimised for storage rather than reads

Used when data isn't suitable format for querying, where generating suitable query is difficult, or query performance poor due to nature of the data or its store.

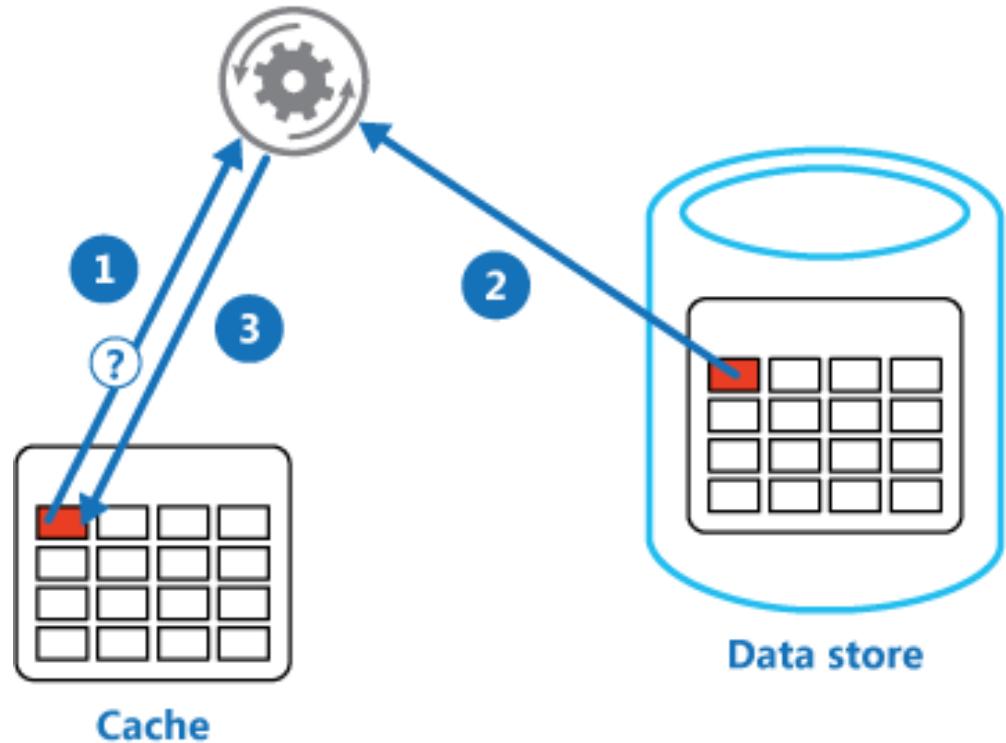
Often part of CQRS

Actor model



- Model for concurrent computation back in 1973
 - Erlang early implementation
- Reduce code complexity in complex systems
 - Actors always exist, virtually
 - Deadlock and re-entrant issues handled by system
 - Asynchronous communications between actors
- Improve performance by holding state in middle tier
 - Middle layer isn't just wrapper round database CRUD calls
- Project "Orleans"
 - Open source - <https://github.com/dotnet/orleans>
 - Developed by Microsoft Research, used in Xbox Halo 4 and 5
- Service Fabric
 - Implements Reliable Actors application framework

Cache-Aside Pattern



- 1: Determine whether the item is currently held in the cache.
- 2: If the item is not currently in the cache, read the item from the data store.
- 3: Store a copy of the item in the cache.

If data is present in the cache and not stale, then read it from there.

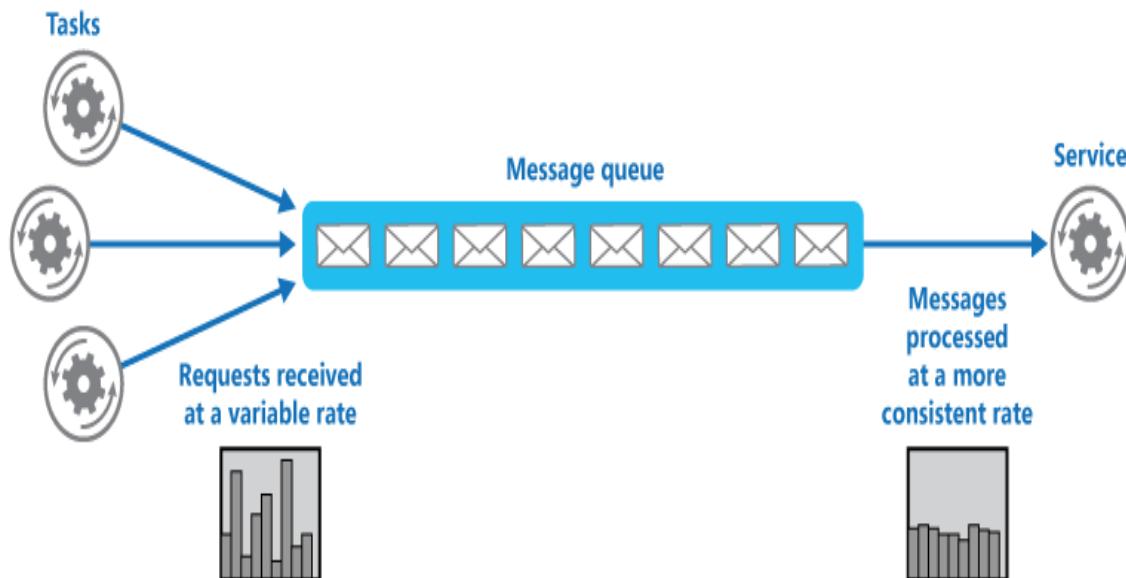
If data is not present, or it's stale, read it from the main store and pass it through into the main cache.

Issues and Considerations:

- Lifetime of cached data
- Evicting data e.g. LRU
- Consistency



Queue Based Load Levelling



Use where there is a highly variable load pattern and idle periods to 'smooth' out the peaks in load

- Maximise scalability
- Help control costs

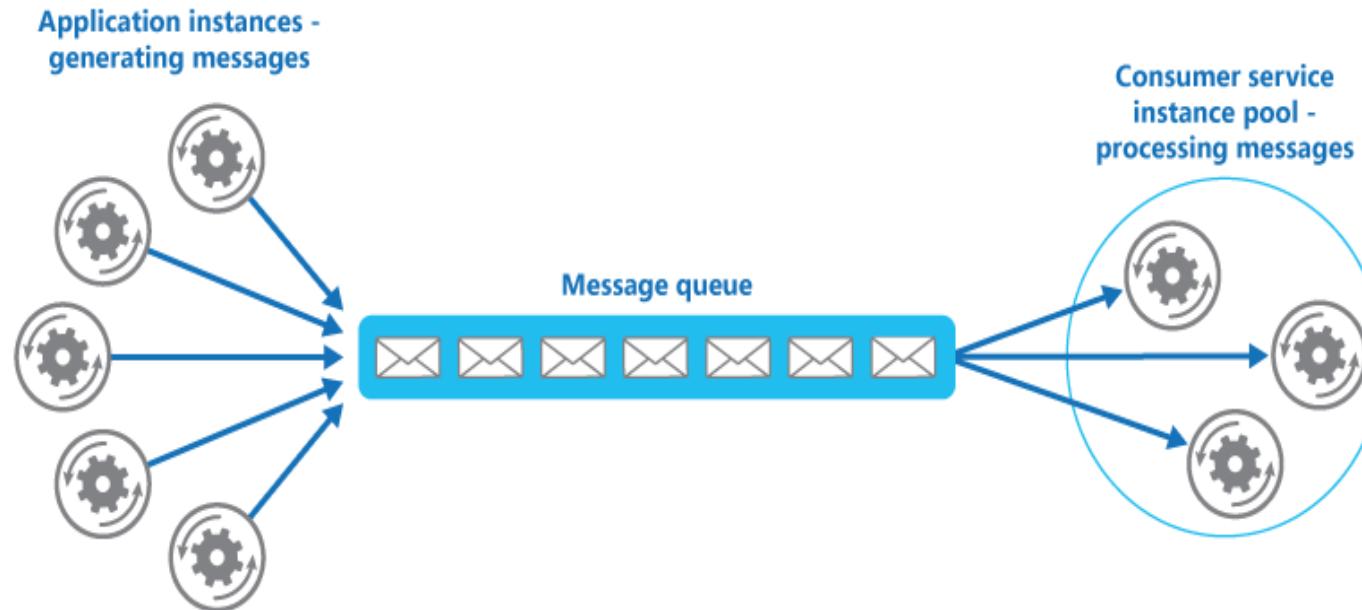
Use as a precursor to enable the competing consumers (autoscaling) pattern

Issues and Considerations:

- Control message rate to avoid overwhelming the target resource
- Check autoscale doesn't increase contention



"Competing Consumers"



Issues and Considerations:

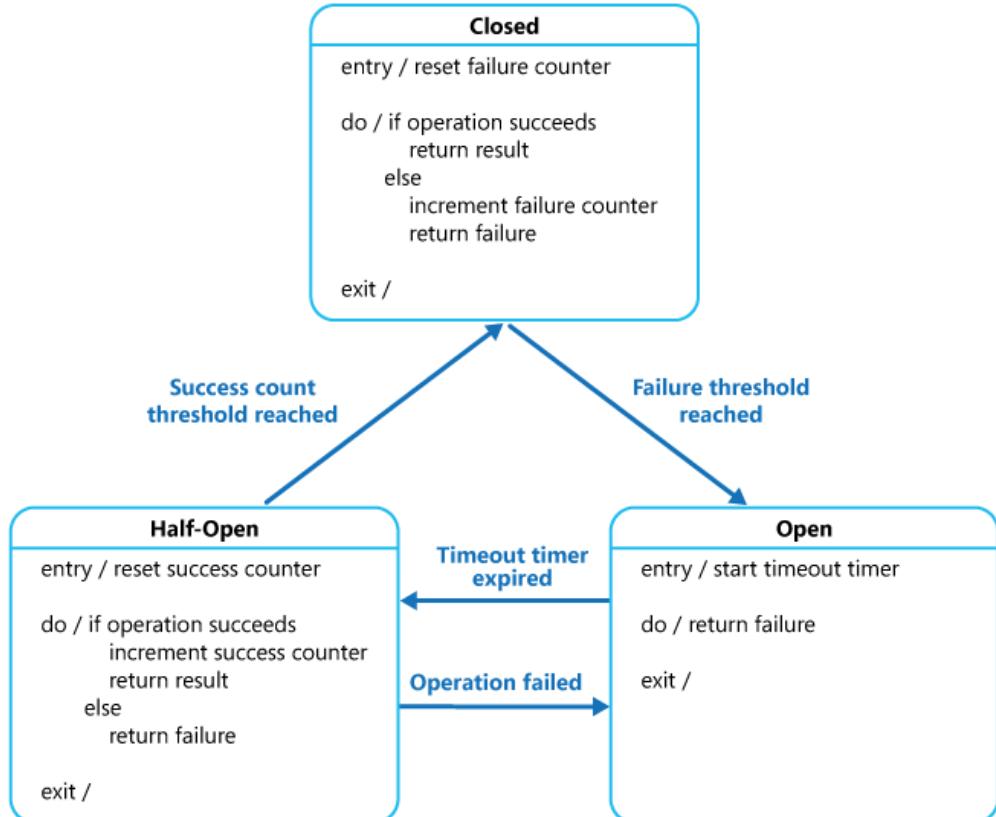
- Message ordering dependencies
- Ensure Idempotent messages
- Detect poison messages
- Potentially dampen autoscale

Scaling out via queues: when our queue consumer becomes a bottleneck, *simply run more instances of the consumer*.

If you are really slick, then auto scale instances based on the Queue's Length – if it starts to back up, then add more capacity until the backlog is resolved, then scale down.

Azure Functions allows you to scale compute resource like this by default and makes it trivial to consume a queue in a massively scalable manner.

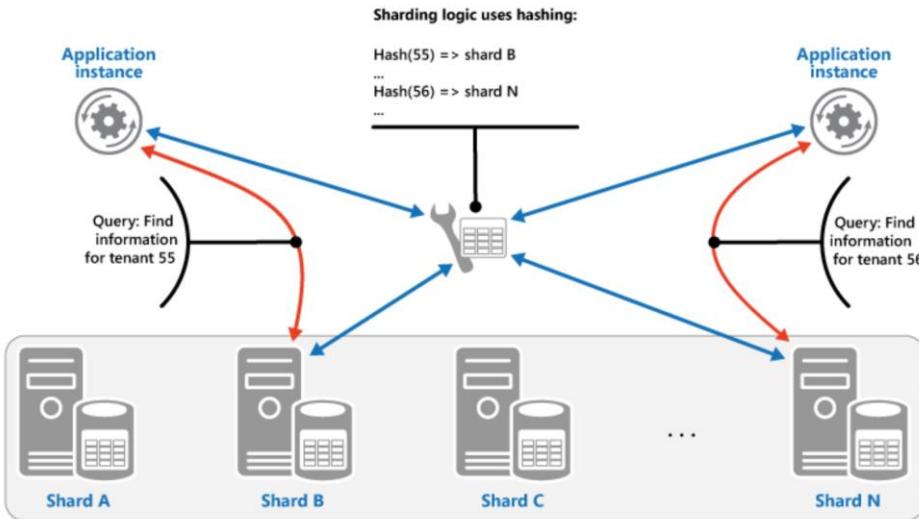
“Failover” and “Circuit Breaker”



- When we know a system is down, behave accordingly and let it recover.
- If we have a service that can be used instead for full functionality or degraded operations then use that instead until our primary service is back online.
- Could encompass anything from secondary credit card processing services, or RA-GRS read mirrors.



Sharding



Sharding strategies

- Lookup
- Key range
- Hashing

- This technique is used to partition a data store into a number of smaller horizontal partitions or shards.
- This pattern can improve scalability when storing and accessing large volumes of data.
- This is what you do when you hit a hard limit of scale in the data tier, effectively split the database in two.
- This pattern can also be used to split services geographically and 'home' certain users data to a specific region for geo-political purposes.
- Azure SQL Database Elastic Scale Libraries can be used to make this process easier
- Azure Cosmos DB has partitioning built in

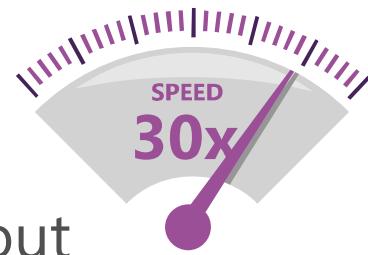


External Cache

Scale out, not up even when your app is stateful!

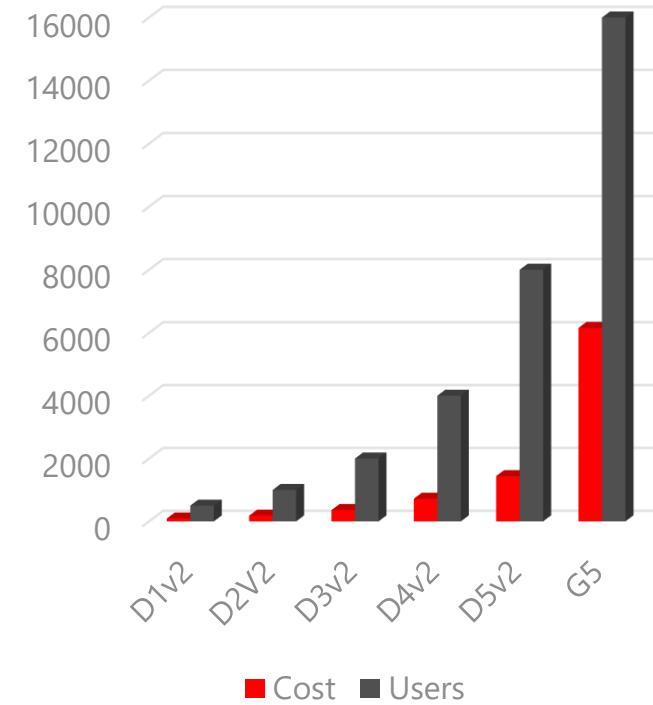
How to make a classic .aspx app stateless without resorting to a SQL Database provider.

<https://azure.microsoft.com/en-us/documentation/articles/cache-aspnet-session-state-provider/>

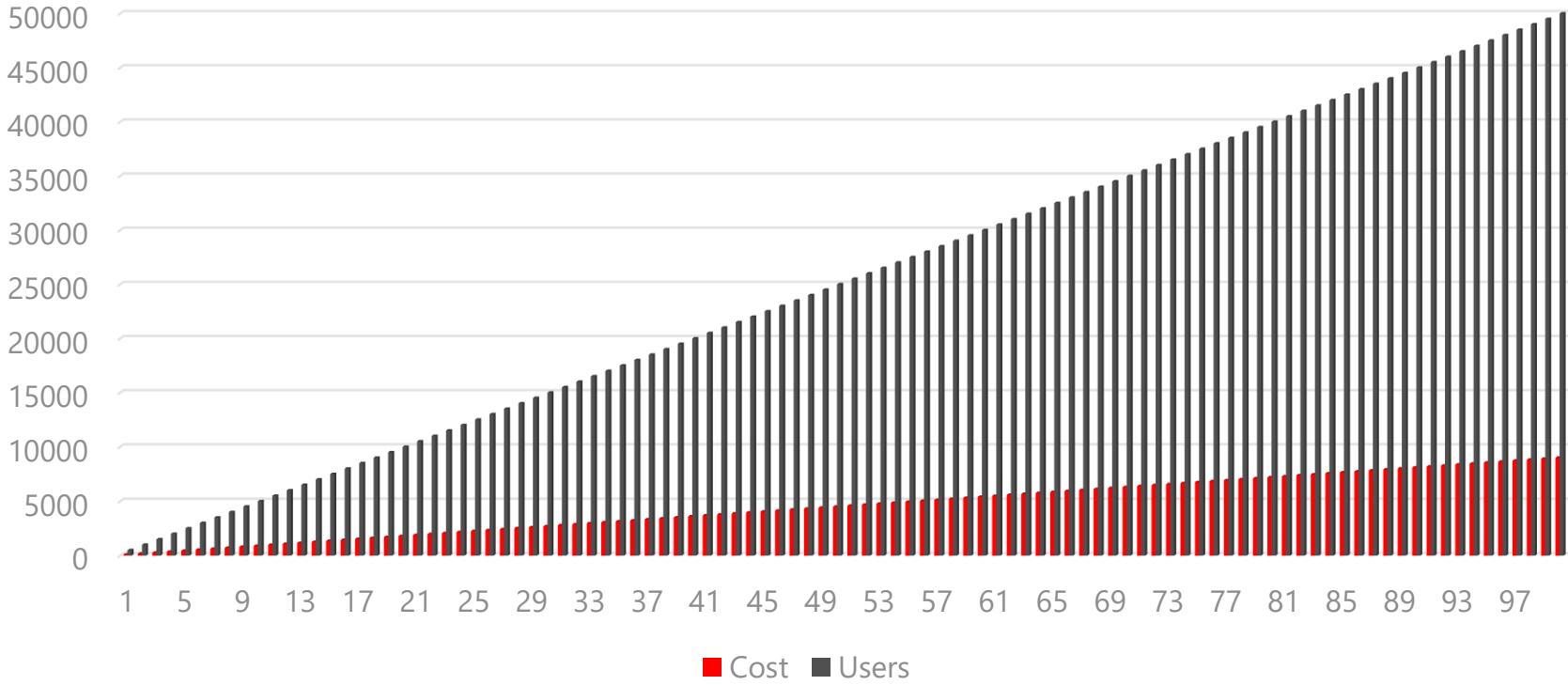


"Scale out"

Single Server (Scale Up)



Stamps / Scale Units + Autoscaling out



Q: A 10 user system costs how much to run in this model ? ~£90 (Single instance, use the compute resource consolidation pattern to group tenants together onto hardware here, 1 single VM stamp should support 50x10 user tenants).

Q: A 1000 user tenant costs how much to run ? ~£180 (2x VM instance)

Q: A 35,000 user tenant costs how much to run ? 72 x D1v2 VM or 3x G5 VM ? How much are they ?

Federated Identity



Simply delegate *authentication* to an external identity provider, such as Azure AD.

This pattern can simplify development, minimize the requirement for user administration, and improve the user experience of the application.

Why roll your own authentication services when there are highly secure, well tested and stable authentication services available for free*.

Note that this pattern is also sometimes used to refer to identity delegation *between providers* where there is a trusted identity that resides in an external domain, such as an on-premise directory accessed over ADFS or Azure AD B2B.

Gatekeeper



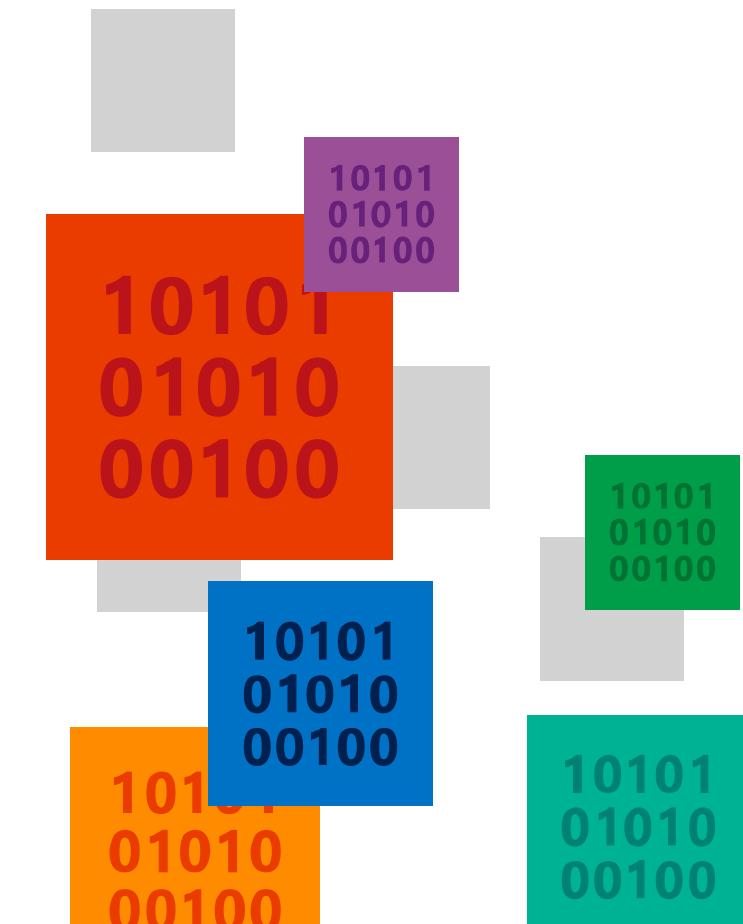
- Protects applications and services by using a dedicated host instance that acts as a broker between clients and the application or service, validates and sanitizes requests and passes requests and data between them.
- This pattern can provide an additional layer of security, and limit the attack surface of the system.
- This is usually implemented as a PaaS service (such as Azure Application Gateway's WAF tier) or a VM / IaaS Web application firewall – such as a Barracuda or F5 Firewall appliance.

Valet Key



- Use a token or key that provides clients with restricted direct access to a specific resource or service in order to offload data transfer operations from the application code.
- This pattern is useful in applications that use cloud-hosted storage systems or queues, and can minimize cost and maximize scalability and performance, without compromising security by opening up the entire system publicly for writes by anyone.
- A common use of this pattern is with Azure Service Bus or Azure Storage, to issue a short-lived access key to write a message to a queue or topic, or to directly read part of a table – this style of usage is known as a “Shared Access Signature”.

Best Practices (Do's)



Let the Azure platform do the
heavy lifting

Content Delivery Networks

(And/or the “Static content hosting pattern”)



Why waste CPU and Memory rendering something that could be served by someone else's resources?

Caching Storage is usually significantly cheaper than compute used to generate a resource each time.

Azure offers integrated use of TWO CDN providers (both Akamai and Verizon CDN services) through Azure CDN.

Either generate the content in advance and store it for retrieval (for example in Azure Blob Storage) or use a CDN to cache that data in an Edge network endpoint.

These services are not just of use for web apps, they can help with metadata distribution over high latency far-flung WANs as well (Suppose you have users that access a central, slowly changing file resource in Australia).

Do: Use an appropriate type of Data Store

Storage Mechanism	Interaction	Starts at (Avg)*	Min GB	Max TB	Use Case
Azure 'Cool' Block Blob storage	REST (Blob), SDKs	£0.01/GB	0	500	Archive, nearline unstructured https storage
Azure 'Hot' Block Blob storage	REST (Blob), SDKs	£0.02/GB	0	500	Online unstructured https storage
Azure Virtual Machine Disk / Page Blob Storage	VM Only	£0.05/GB	0	500 (Not per VM)	High Performance storage, up to 500 IOPS per disk volume (up to max volumes per VM type).
Azure Table Storage	REST (Blob), SDKs	£0.07/GB	0	500	Tabular, non-relational (NoSQL) Mass-scale dictionary https lookup service, partitioned by default, no secondary indexes allowed.
Azure Premium Virtual Machine Disk/ Page Blob Storage	VM Only	£0.13/GB	128	35	GUARANTEED High Performance storage, up to 5000 IOPS per disk volume (up to max volumes per VM type > 80,000 IOPS & 2TB / Second).
Azure SQL Database Basic	T-SQL (TDS)	£2.33/GB *	2	1	Tabular, Scalable, Classic Relational DBMS, Always On
Azure SQL Data Warehouse	T-SQL (TDS)	£1.03/GB *	1024	~250 Compressed (~1PB)	Scalable, Tabular, Parallelized, Relational DBMS, can be paused.
Cosmos DB	REST (Blob), SDKs	£4.62/GB *	1	0.01 (10GB)	JSON Indexed document storage, can be partitioned.
Azure Data Lake Store	WebHDFS	£0.06/GB *	1024	1PB per file (!)	Hadoop based unstructured data storage layer. – Preview price includes 50% discount off US pricing, not available in Europe yet.

* These are indicative numbers, these numbers are variable on a number of factors. For example SQL DW starts billing at 1TB of capacity, storage has an access charge per million transactions.

Do: Use an appropriate type of Storage

Replication strategy	LRS	ZRS	GRS	RA-GRS	Premium-LRS
Data is written synchronously to multiple nodes.	Yes	Yes	Yes	Yes	Yes
Data is replicated async across multiple <i>facilities</i> .	No	Yes	Yes	Yes	No
Data can be <u>read</u> from the secondary location as well as from the primary location.	No	No	No	Yes	No
Number of copies of data maintained on <i>separate nodes</i> .	3	3	6	6	3
Number of copies of data maintained in a <i>separate geographically paired region</i> .	0	0	3	3	0
Relative Cost / GB / Month (UK South – Standard List Price)	Block (Cold) £0.01 Block (Warm) £0.02 Table £0.07 Disk £0.05	Block £0.03 Table N/A Disk N/A	Block £0.04 Table £0.09 Disk £0.09	Block £0.06 Table £0.11 Disk £0.11	Block N/A Table N/A Disk £0.13
Performance Tier	+	+	+	+	+++

<https://azure.microsoft.com/en-us/pricing/calculator/>

<https://azure.microsoft.com/en-us/documentation/articles/storage-redundancy/>

DO: Determine uptime & Non-Functional Requirements

What are the Fault Tolerance / Backup + Recovery objectives?

- Recovery Time Objective (RTO)
 - Amount of "real time" that can pass before disruption begins to seriously & unacceptably impede the flow of business operations
- Recovery Point Objective (RPO)
 - Amount of data that will be lost or will have to be re-entered during downtime
- Max Downtime / 9's
- Mean Time Between Failures (MTBF)
- Mean Time to Recovery (MTTR)

What is the COST OF DOWNTIME?

- Availability/resilience/SLAs
- Data retention policy
- Location / Residency Restrictions
- Schema management requirements

DO: Clearly define boundaries and parameters of support, Both Internally and Externally.

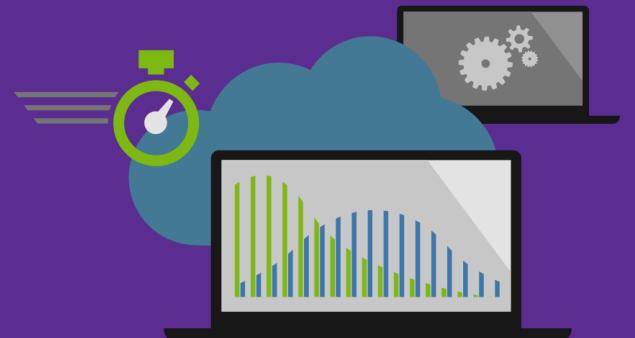
- You don't want to be waiting on other people, on another continent to wake up when you have an emergency.
- You want a clearly defined and executable escalation path in an emergency.
- You want to have an external indicator of service status (a Service Health Dashboard), so you still have a phone / email system under the stress of a significant event. This also needs to be hosted in a different place from your primary systems.
- Have a 'break-glass-and-audit' admin delegation policy where users require intermittent admin access to production services.

DO: Release software in a controlled manner, with progressive testing and a promote to production process

- Identify a 'Release Manager(s)' in the organisation who is responsible for the Go or No-Go decision to deploy
- Script the deployment process end-to-end and use a release tool like VSTS Release Management to trigger the scripts. AND CHECK THEM INTO THE SOURCE REPOSITORY!
- Use a single branch for deployment and release where possible of a production service
- When there is a failure in the deployment process, fix it and restart the process as another release, try not to hotfix deployments

Do: Load test your application before deployment

- At the absolute least, ensure that some kind of automated load testing is performed to detect a noticeable change in performance of specific, critical endpoints.
- Test the application at significant load to ensure that spikes in user activity and load will not take down the application at burst times.
- Test your application automatically **before** deployment using CI / CD Practices, the earlier you find problems, the cheaper they are to fix.
- Knowing your baseline operational cost of a service on a fixed hardware stamp makes it simpler to predict how to cost and upscale the service.

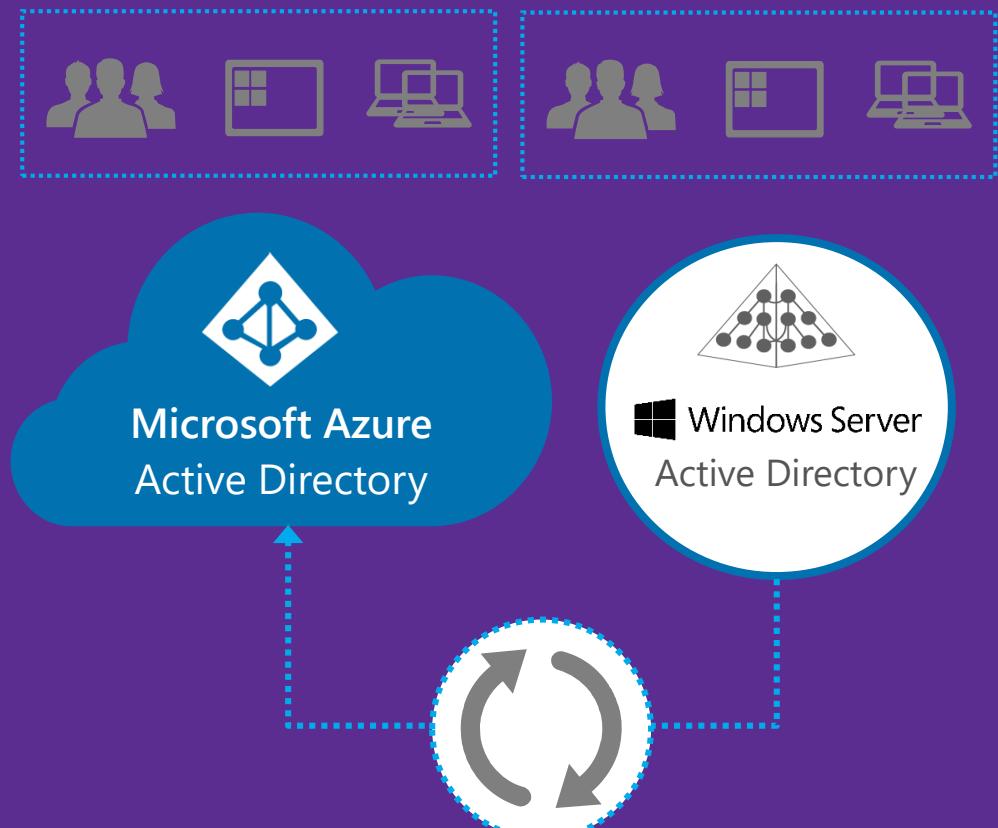


DO: Consider the Chaos Monkey

- Software tool developed by Netflix engineers to test the resiliency and recoverability
- Simulates failures by shutting down one or more of the VMs
 - Named for the way it wreaks havoc like a wild and armed monkey set loose in a data center
- Failures set to times when they can be closely monitored



Do Implement: Multi Factor Authentication



- ✓ Protect sensitive data and applications both on-premises and in the cloud with Multi Factor Authentication
- ✓ Can use Active Directory (on-premises) with Azure Active Directory (in cloud) to enable single sign-on, a single directory, and centralized identity management
- ✓ Multi Factor Authentication can be implemented with Phone Factor or with AD on-premises

Do Implement: Encryption



In Transit:

AZURE:

- Encrypts most communication between Azure datacenters
- Encrypts transactions through Azure Portal using HTTPS
- Supports FIPS 140-2

CUSTOMER:

- Can choose HTTPS for REST API (recommended)
- Configures HTTPS endpoints for application running in Azure
- Encrypts traffic between Web client and server by implementing TLS on IIS
- IPSec VPN tunnels between Virtual Networks

At Rest:

Virtual Machines:

- Data drives – full disk encryption using BitLocker
- Boot drives – BitLocker and partner solutions
- SQL Server – Transparent Data and Column Level Encryption
- Files & folders - EFS in Windows Server

Storage:

- Bitlocker encryption of drives using Azure Import/Export service
- StorSimple with AES-256 encryption

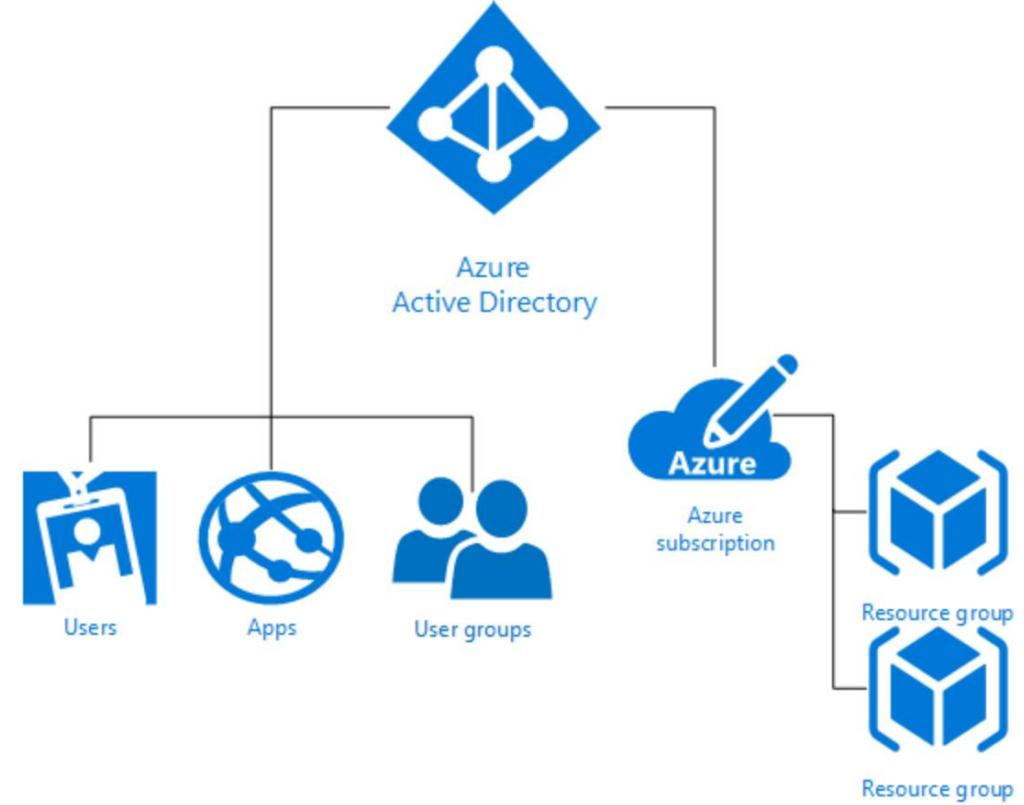
Applications:

- Client Side encryption through .NET Crypto API
- RMS Service and SDK for file encryption by your applications

Do Implement: RBAC

Azure Role-Based Access Control (RBAC) enables fine-grained access management for Azure resources in subscriptions that are backed by an Azure Active Directory Subscription.

Using RBAC, you can segregate duties within your DevOps team and grant only the amount of access to users that they need to perform their jobs.



<https://azure.microsoft.com/en-us/documentation/articles/role-based-access-control-configure/>

Anti-Practices (Don'ts)



Don'ts: Dependency and Service Management



SINGLE-REGION DEPLOYMENT IN AZURE

Some customers believe that Azure automatically deals with scalability and resiliency across regions.

Customers should plan for an outage of services in a particular region and failover where necessary

EXPECT 'MAGIC' RESILIENCE WITHIN SERVICES

Some Azure Services have functionality built in, to deal with availability, use them!

Customers should be aware of these services, e.g. Traffic Manager, or Service Bus Paired Namespaces, Storage GRS vs RA-GRS (and failover)

IGNORE THIRD PARTY DEPENDENCIES THAT COULD KILL YOUR SERVICE

Most solutions have included dependencies outside of Azure.

Customers should ensure that a graceful degradation occurs, for components inside Azure and outside.

IGNORE SINGLE POINTS OF FAILURE

Some solutions we have seen have single points of failure in their solution. If this goes down, your application will be down.

Customers should run all tiers of their application in a resilient manner if the SLA requires it.

Don'ts: Support and Incident Response



EXPECT EVERYONE TO AUTOMATICALLY KNOW WHAT TO DO IN A MAJOR INCIDENT

A number of Azure customers do not have a Major Incident response plan in place.

Customers should ensure that they have a MIRP in place, clearly defining responsibilities across the solution, escalation protocol and any other necessary processes to follow in a disastrous scenario.



EXPECT YOUR DATA TO RECONCILE ITSELF IN THE EVENT OF AN OUTAGE

Many solutions are based around the concept of Eventual Consistency and / or multi-region deployment.

When faced with region failover, how do you know that critical data is present and valid in the new region, what was lost from the primary region and may now never arrive in the secondary ?



EXPECT DR / HA STRATEGIES TO 'JUST WORK'. TEST THEM !

A number of customers had either an automated Disaster Recovery mechanism or well-documented approach. However, this approach had not yet been tested.

Customers should test their disaster recovery semi-regularly, to ensure that the process is still relevant and that all parties are aware of the required steps.

Don'ts: Subscription Management



GIVE AWAY THE KEYS TO THE KINGDOM TO EVERYONE

Most customers have:

- Large number of admin accounts in production
- No Multi Factor Auth. for admins
- Admin accounts using Microsoft accounts

Customers should be aware of the potential impact of a compromise of an admin account..

TURN DEVELOPERS INTO GODS

We see customers with a large number of admins that were developers on their production subscription.

Customers should be aware of the risk here, that a developer could accidentally deploy into live, causing potential business impact.

DEPLOY CRITICAL RESOURCES WITHOUT LOCKING FOR ACCIDENTAL DELETION

Many customers have resources in their environment that, if deleted, could bring down the solution.

Customers should be aware of the risk of accidental deletion. Consider using Azure Resource locks as a mitigation step.

DEPLOY MULTIPLE ENVIRONMENTS IN ONE CLASSIC SUBSCRIPTION

Many customers are running multiple environments (Dev, Test and Production in one subscription).

Customers should be aware that ASM services do not have Role Based access in place, so can't be granularly controlled,

Don'ts: Deployment



RELEASE WITHOUT LOAD TESTING

If a customer introduces a performance bottleneck in code, they will be unaware until it is in users' hands.

Customer should load test their solution ahead of every major release. This will act as a guideline for any performance defects.



CI =! AUTOMATED TESTING

A number of customers do not follow recommended practices around continuous testing and integration.

Customers should employ DevOps practices around continuously integrating their code, to ensure that their developers are working on the latest set. Additionally, the customer can run regular unit / integration tests, to check for regression in code quality.

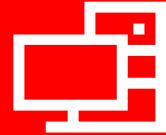


CD =! AUTOMATED DEPLOYMENT

A number of customers do not operate a continuous delivery pipeline. This is not essential, but could aid in agility of software delivery.

If desirable, customers should employ DevOps practices around continuously deploying their code. Once a good build is found in Dev/Test, the team can push this through to production, using their typical release process.

What to avoid in Cost Management



LACK OF COST OWNERSHIP AND REPORTING

Many Customers do not have someone in place to monitor cost on Azure.

This should be done at a level appropriate for the organization, e.g. project or team level.

OVER-PROVISIONING OF RESOURCES CAUSING WASTAGE

Some customers over-provisioned resources, even though such sizing was not required.

Customers should perform load testing, and be aware of the necessary limits of their service.

PAYMENTS MADE BY CORPORATE CREDIT CARD

Some customers are managing their Azure payments via credit card.

The customer could be missing out on a number of discounts and enhanced support offerings via their Enterprise Agreement.

DEV/TEST ENVIRONMENTS NOT DE-PROVISIONED

A number of customers have not de-provisioned their dev/test when not required.

To reduce cost, customers should consider scaling down / shutting off their dev/test environments when not in use.

What to avoid in Efficiency



OVER COMPLEX SOLUTION DESIGN

In a number of scenarios, customers have over-engineered their solution, to solve the problem at hand. This has typically introduced unnecessary cost and risk/complexity within the solution.

Take an external view when working with the customer, and challenge their approach. Could they be using an alternate approach, which would be simpler and reduce their end-cost?



LACK OF CLARITY ON NON-FUNCTIONAL REQUIREMENTS

Some solutions have been over-engineered for the level of SLA that they require, or they have an SLA which is impossible with their solution architecture. This can result in either excessive risk, or cost.

Customers should ensure that they are fully aware of their requirements in advance of design. Strongly challenge them on this point – How can they build a solution meeting availability, scalability and cost requirements, if they are undetermined?

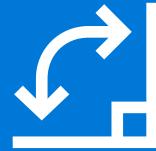
Don'ts: Scalability



IGNORE YOUR SCALABILITY APPROACH

Some customers believed that the Azure platform automatically handled scaling across all Azure services, without any configuration. This is not true.

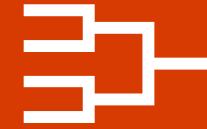
This is an important concept, which customers must be aware of. For example, Web Apps require scalability rules to be configured, as do VM Scale Sets.



IGNORE DOWNTIME FOR SCALE UP

A number of customers had Single Points of Failure, due to having single instances of certain resources. As such, if they wanted to scale up/out, there could potentially be some down time.

Customers should be aware of the requirements to meet Azure SLAs (particularly for VMs), and also how the platform scales services in use.



SHARE APP SERVICE STAGING SLOTS

Some customers have planned to host their separate environments on the same Web Application, but across different staging slots. All slots run on the same machine, so any strain on load would effect all environments.

Customers should ensure that the environments are appropriately separated, so that load in one environment, does not negatively impact another.

Don'ts: Performance



IGNORE CACHING / OR USE EXCESSIVE CACHING

Many customers caching approach is not appropriate for their solution. This is true across various levels (e.g. Caching of data via Redis cache, through to caching of assets via CDN).

Customers should consider their SLAs (in terms of service requirements), and understand how caching can improve performance compared to adding additional complexity and cache lifecycle headaches.



LACK OF DESIGN FOR LATENCY

Highly variable latency is normal for access to internet based services as are random timeouts and transient errors as services auto-scale and auto-recover.

When operating services in the cloud, latency constraints usually need to be relaxed and there needs to be an acceptance that some operations will timeout and need to be retried automatically.

Don'ts: Deployment



LACK OF LOAD TESTING ON RELEASE

If a customer introduces a performance bottleneck in code, they will be unaware until it is in users' hands.

Customer should load test their solution ahead of every major release. This will act as a guideline for any performance defects.



LACK OF CONTINUOUS INTEGRATION

A number of customers did not follow recommended practices around continuous testing and integration.

Customers should employ DevOps practices around continuously integrating their code, to ensure that their developers are working on the latest set. Additionally, the customer can run regular unit / integration tests, to check for regression in code quality.



LACK OF CONTINUOUS DELIVERY

A number of customers did not operate a continuous delivery pipeline. This is not essential, but could aid in agility of software delivery.

If desirable, customers should employ DevOps practices around continuously deploying their code. Once a good build is found in Dev/Test, the team can push this through to production, using their typical release process.

Don'ts: Security and DevOps



USE PERMANENT KEYS AND SECRETS

Most customers do not rotate their storage account keys or secret credentials.

Customers should investigate Azure Key Vault and rotate keys in line with their usual procedures

IGNORE A SECURE DEVELOPMENT LIFECYCLE PROCESS

Security begins at the design stage and extends throughout the whole product lifecycle.

Review best practice around SDL and train developers to develop with SDL in mind.

LEAVE SECURITY AT THE PERIMETER

Some customers believe that managing their perimeter is enough to remain secure, this is not true – on-premises or in the cloud.

Customers should be aware of the threats, and adopt a defense in depth approach.

PUT SHARED SECRETS IN SOURCE CONTROL

Developers should not generally be able to see these secrets and connect to production databases .

Customers should be aware of the risk to production data from staff leaving the organisation.

EXPECT BREACHES TO IDENTIFY AND RESOLVE THEMSELVES

Some customers do not baseline usual activity of their system, so are unable to detect unusual activity.

Customers should consider operating as though they have been breached.

Don'ts: Subscriptions and Admin



IGNORE ADMIN BEST-PRACTICE

Most customers should have;

- Minimum number of admin accounts in production
- Multi Factor Auth. for admins
- Enabled Azure AD accts, NOT using MSA accounts

Customers should be aware of the potential impact of a compromise of an admin account.

MAKE DEVS ADMINS ON PRODUCTION SUBSCRIPTIONS

We see customers with a large number of admins that were developers on their production subscription.

Customers should be aware of the risk here, that a developer could accidentally deploy into live, causing potential business impact.

LEAVE RESOURCES OPEN TO ACCIDENTAL DELETION

Many customers have resources in their environment that, if deleted by an attacker, could bring down the solution.

Customers should be aware of the risk of malicious deletion by an employee. Consider using Azure Resource locks as a mitigation step.

LEAVE EVERYTHING IN ONE SUBSCRIPTION ('CLASSIC' ENVIRONMENTS)

Many customers are running multiple environments (Dev, Test and Production in one subscription).

Customers should be aware that 'classic' (non ARM) services do not have Role Based access in place, so can't be granularly controlled.



Thanks for coming today.

Please don't forget to fill in your
feedback forms!

