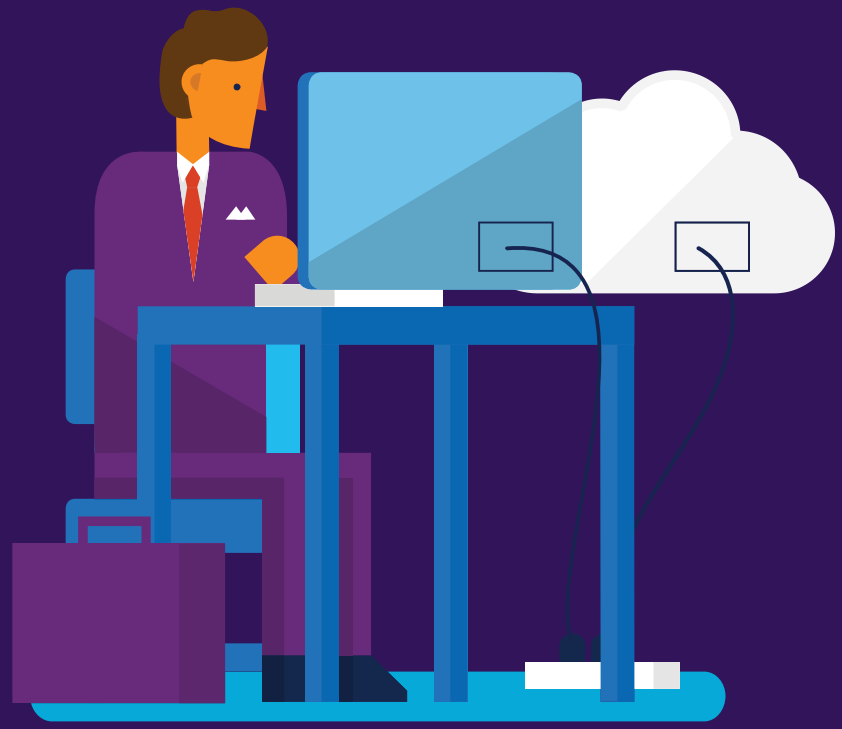
Journey to the Cloud

Azure Infrastructure, Platform, AI and Beyond…

Will Eastbury, David Gristwood, Mike Ormond
Gabriel Nepomuceno, Adam Jackson

http://aka.ms/azureevent

#Azure
#BuildWithAzure



Microsoft

vipazure@microsoft.com

Introduction and Housekeeping

Fire Alarms – none today

Emergency Exits

Toilets

Lunch 1pm - Dietary Requirements?

Finish around 4pm

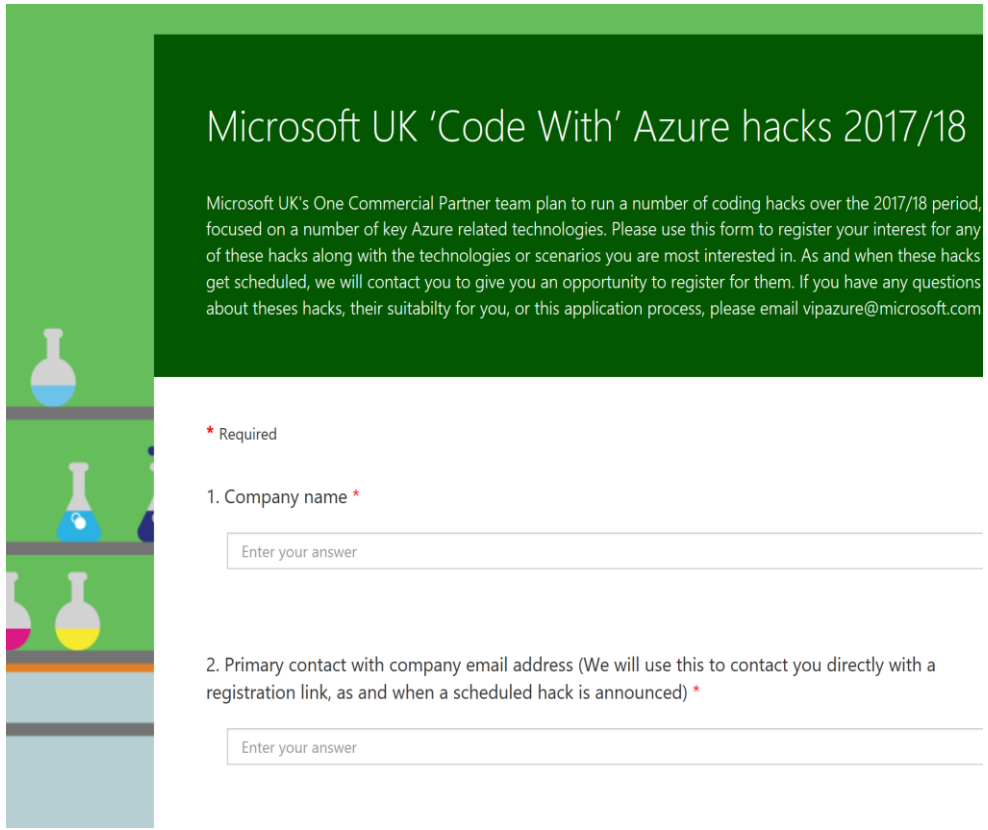
Feedback forms

Microsoft

Other technical activities from our team

Code with hacks

https://aka.ms/ukocpisvhack0817



Workshops

https://aka.ms/AzureEventList

Designing Highly Scalable and Available Cloud Solutions – What to do and What to avoid – London, September 27th 2017

Our technical experts will explore the key critical considerations when building secure, scalable and highly available cloud-based solutions. We will share the secrets, based on a wealth of real-life implementation scenarios, of what to look for and what to avoid. Which are the architectural patterns that are effective and what are the implementation and business benefits of using them?

Register for London September 27th

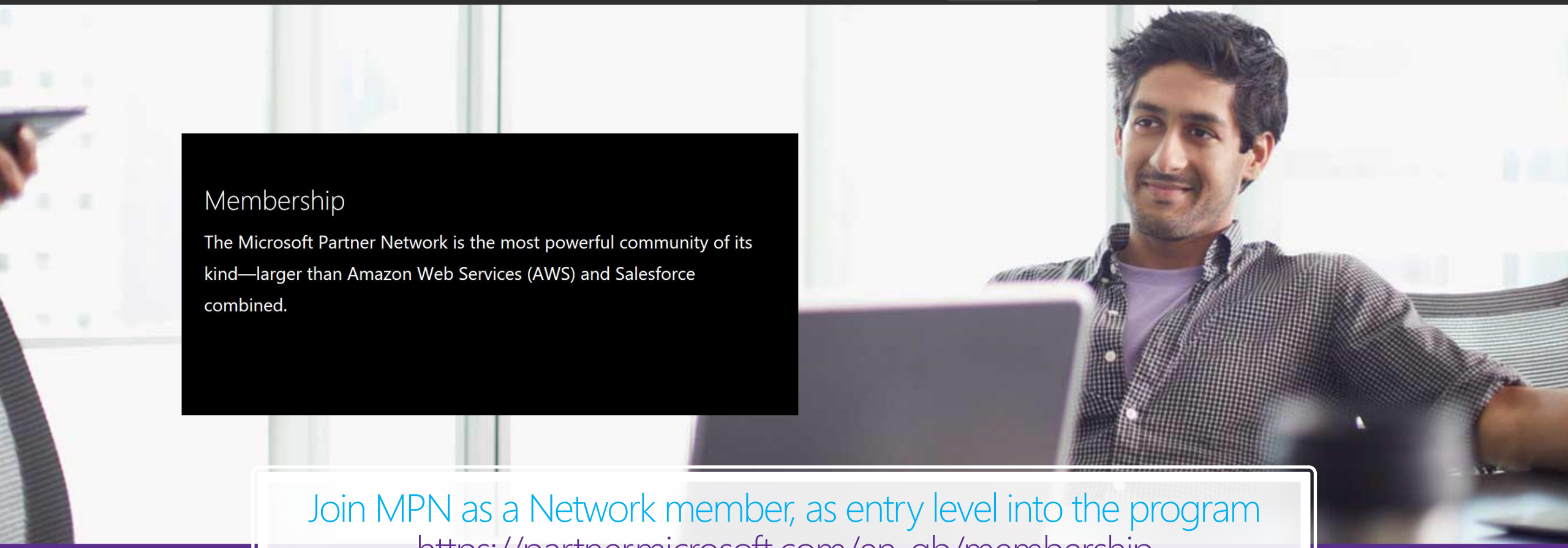
Azure Serverless & Microservices Workshop – London October 10th 2017

What if you could spend all your time building and deploying great apps, and none of your time managing servers? Server less computing lets you do just that, because the infrastructure you need to run and scale

It will be a practical "hands on" development "hack", in which attendees will move their own software to Azure.

Applications close at 5pm on Monday 2nd October. All applicants will be informed by 4th October.

Apply - https://aka.ms/ukocpisvsohackoct17



Membership

The Microsoft Partner Network is the most powerful community of its kind—larger than Amazon Web Services (AWS) and Salesforce combined.

Join MPN as a Network member, as entry level into the program
https://partner.microsoft.com/en-gb/membership

You want to grow your business. We know how.

Partnering with Microsoft pays off.

Partner sign in >

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.

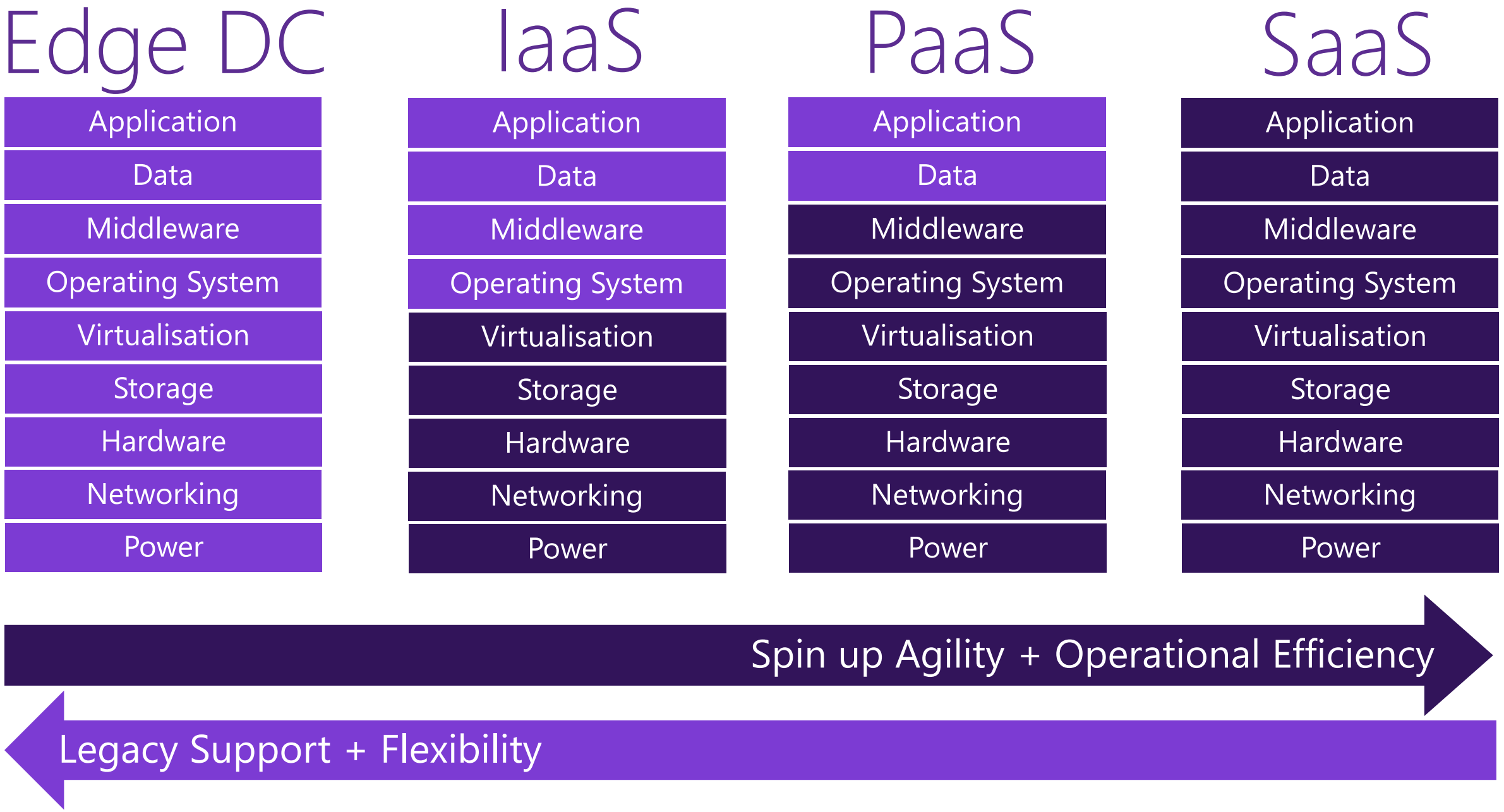
Discuss when to use advanced services and concepts like Bots, AI and Cognitive Services.

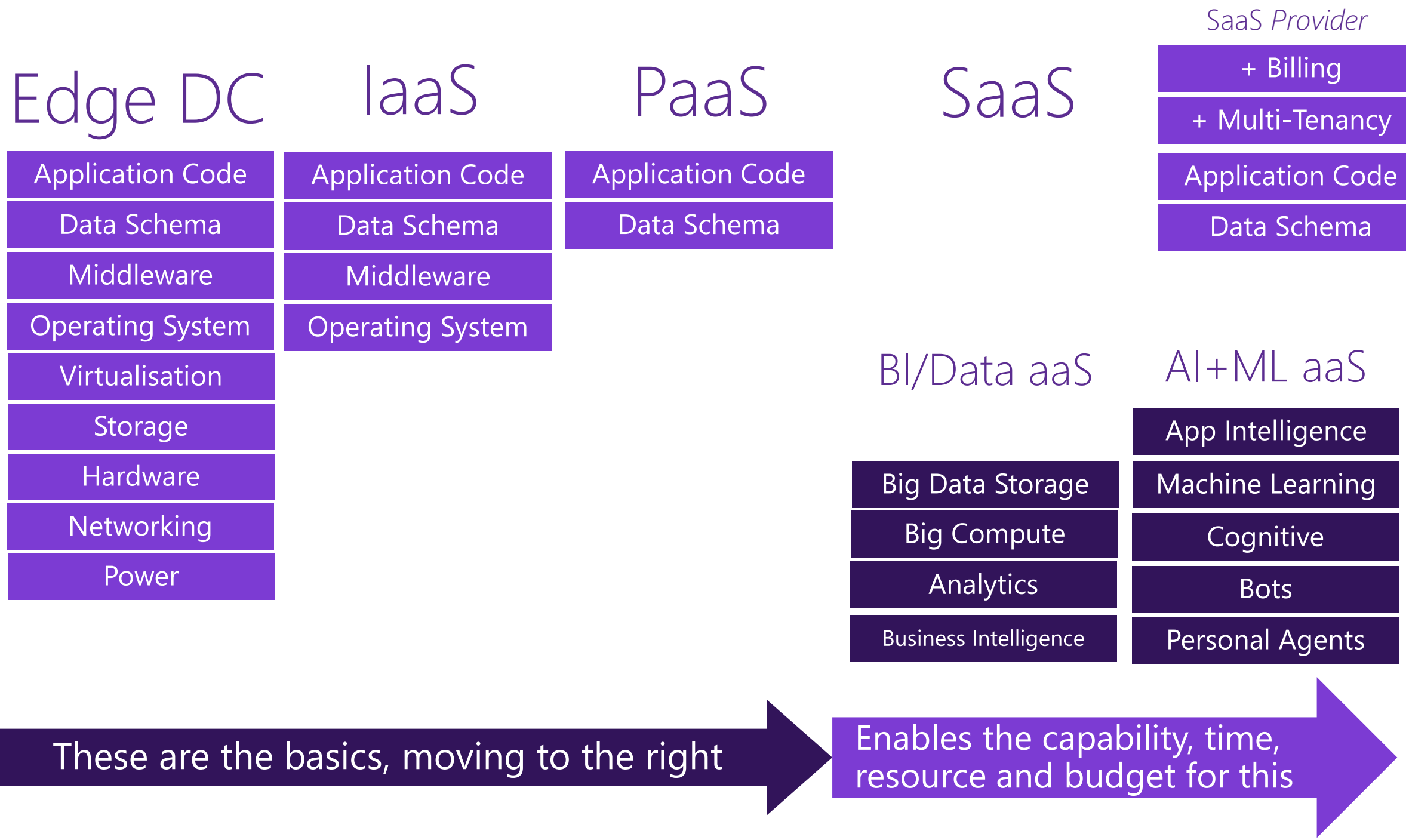
Close the day with a showcase of some of the patterns and practices to use and the mistakes not to make whilst building and migrating your applications.

Microsoft

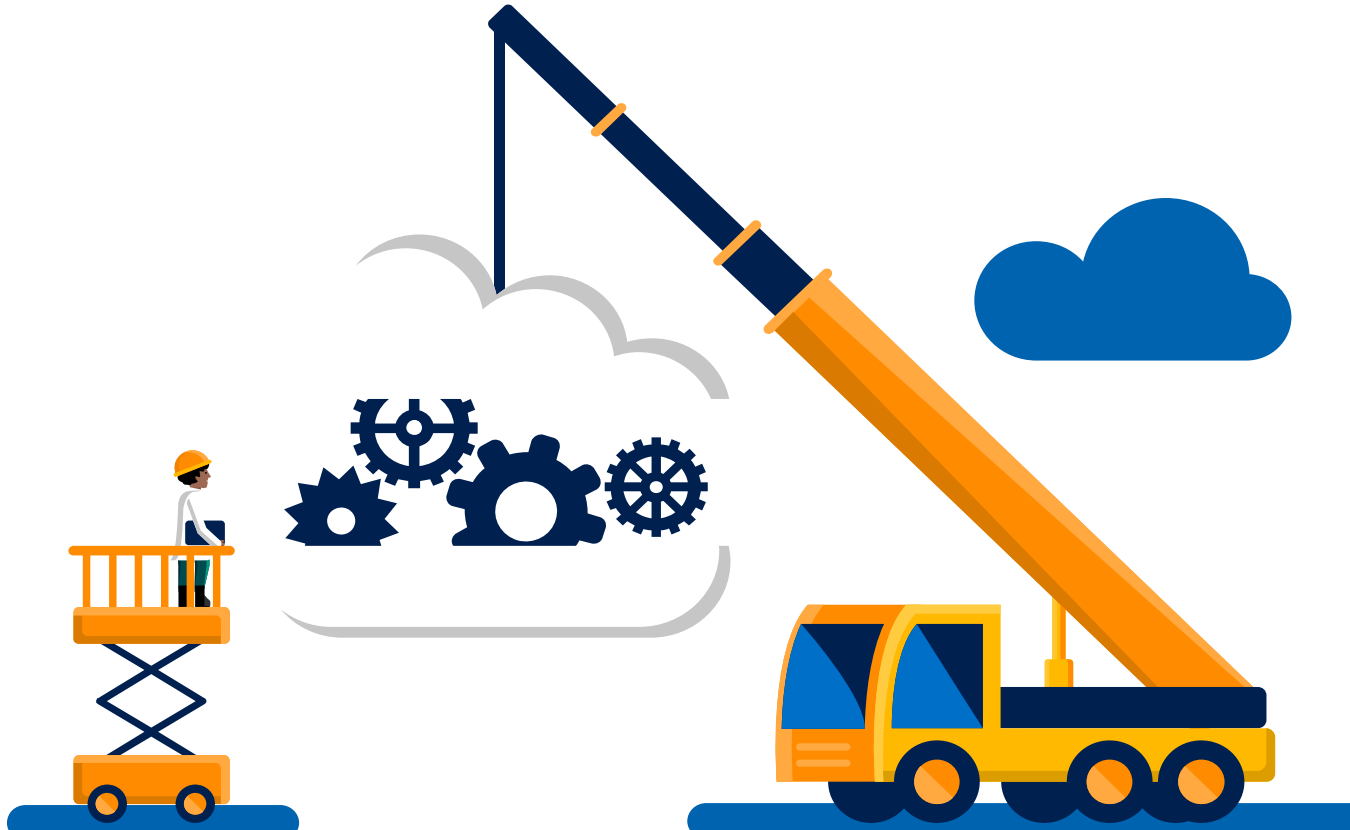
Why the journey through the cloud?

Microsoft


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

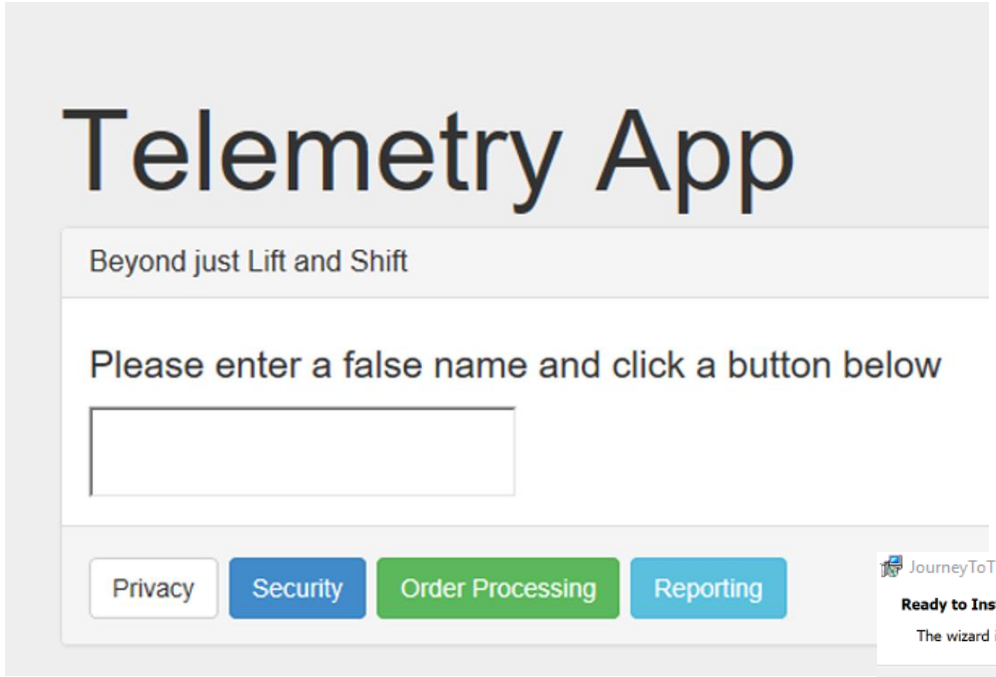

Edge DC
Application Code
Data Schema
Middleware
Operating System
Virtualisation
Storage
Hardware
Networking
Power
IaaS
Application Code
Data Schema
Middleware
Operating System
PaaS
Application Code
Data Schema
SaaS
SaaS Provider
+ Billing
+ Multi-Tenancy
Application Code
Data Schema
BI/Data aaS
Big Data 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

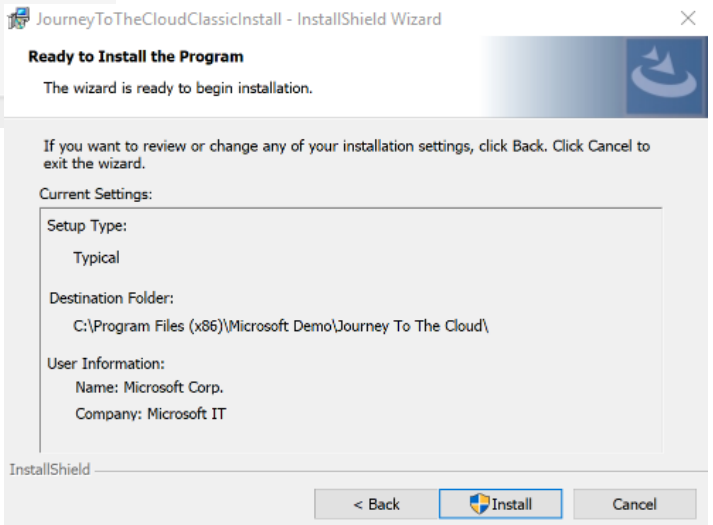
So let's begin the Journey …



Microsoft

Let meet our App…



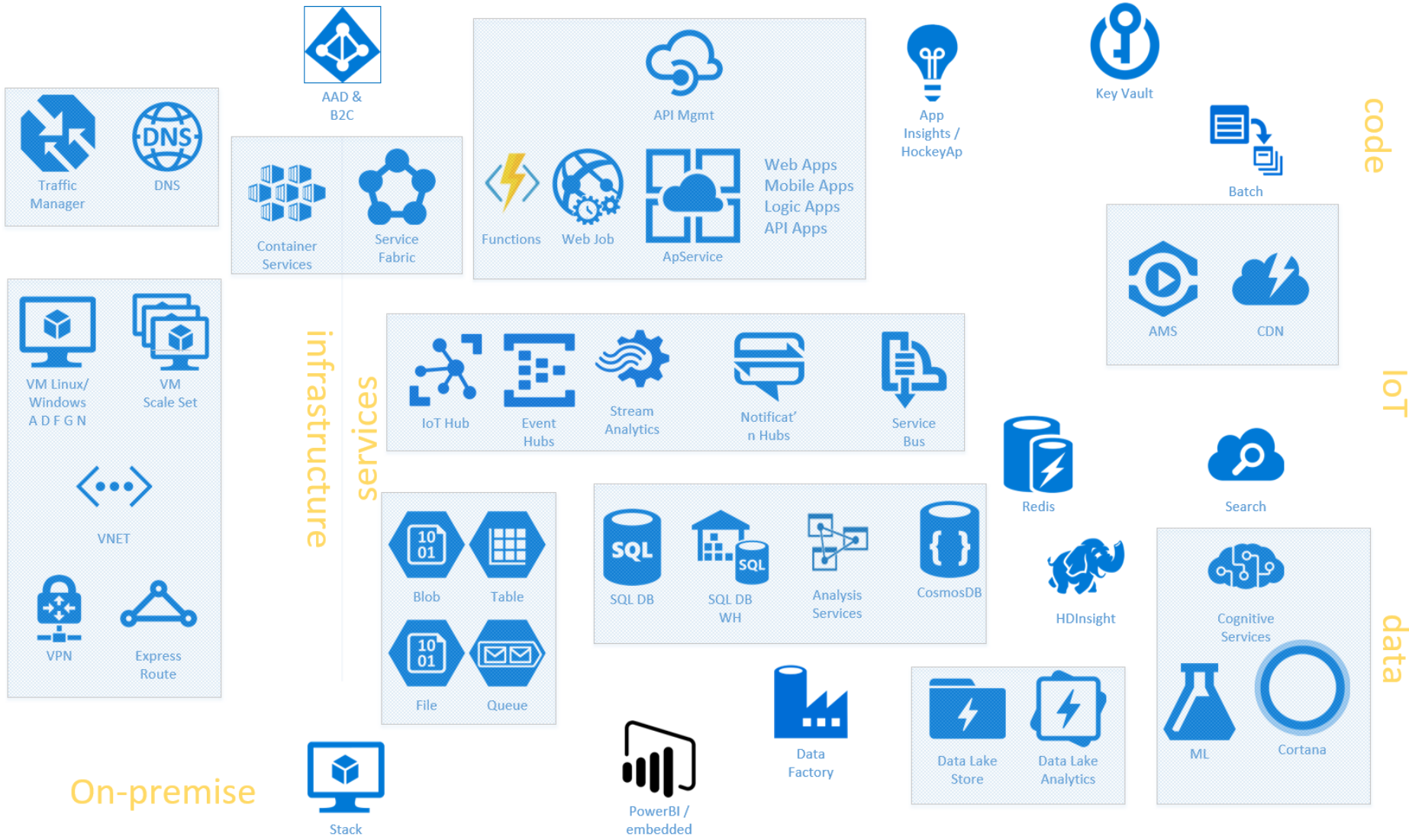


- Classic web forms app
  - No real functionality, simply logs input field and button pressed
  - Log outputs data to a .dat file in ingress directory
- Windows Service
  - Watches for files in ingress directory
  - Moves and updates them to egress directory

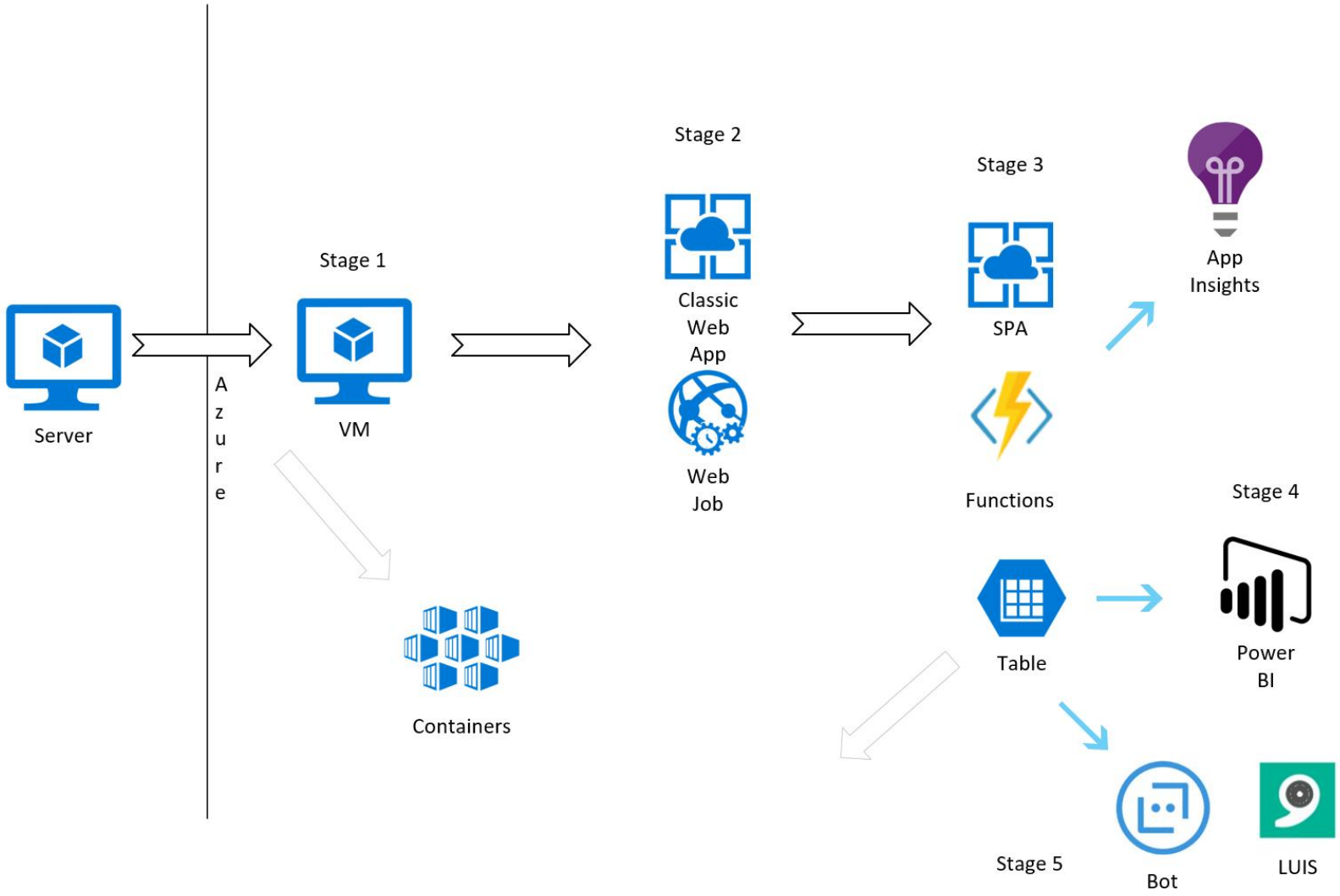
```
<appSettings>
  <add key="ingresspath" value="C:\DataWriteArea\Ingress\" />
  <add key="egresspath" value="C:\DataWriteArea\Egress\" />
  <add key="ingresspattern" value="*.dat" />
</appSettings>
```

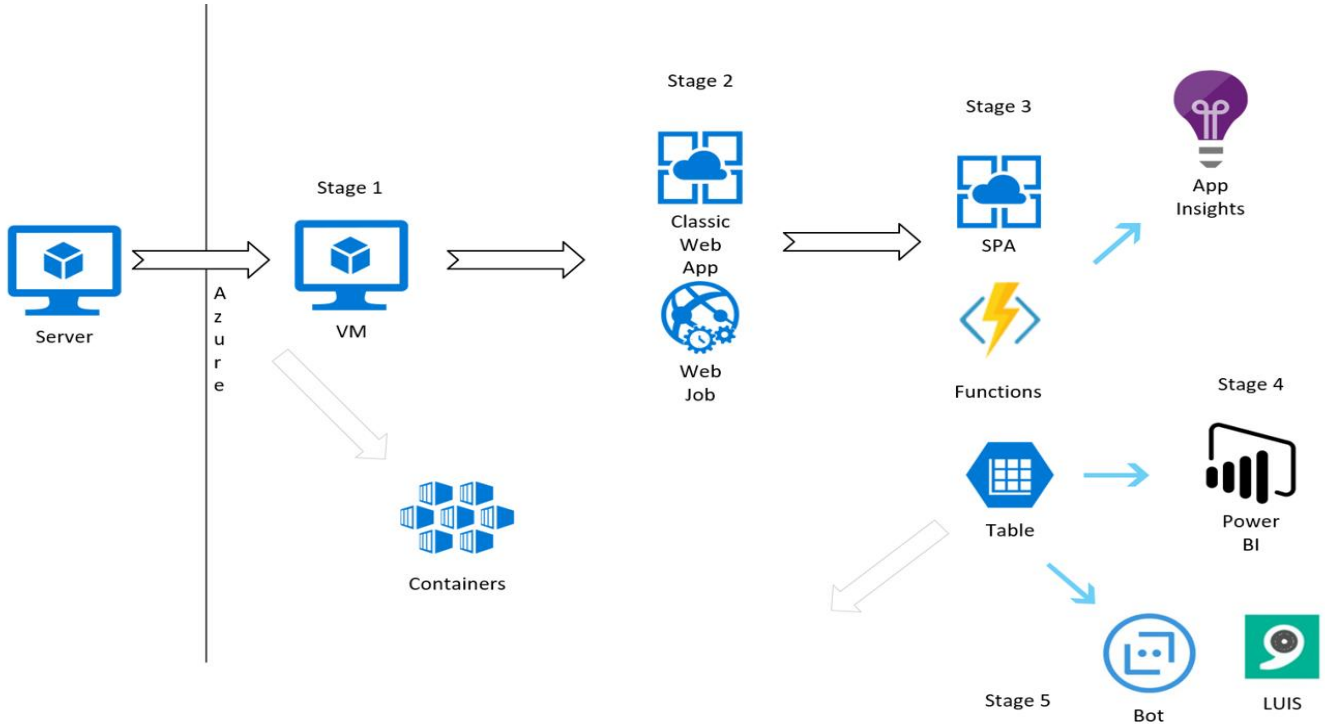
- Started off life as MSI

Azure Services



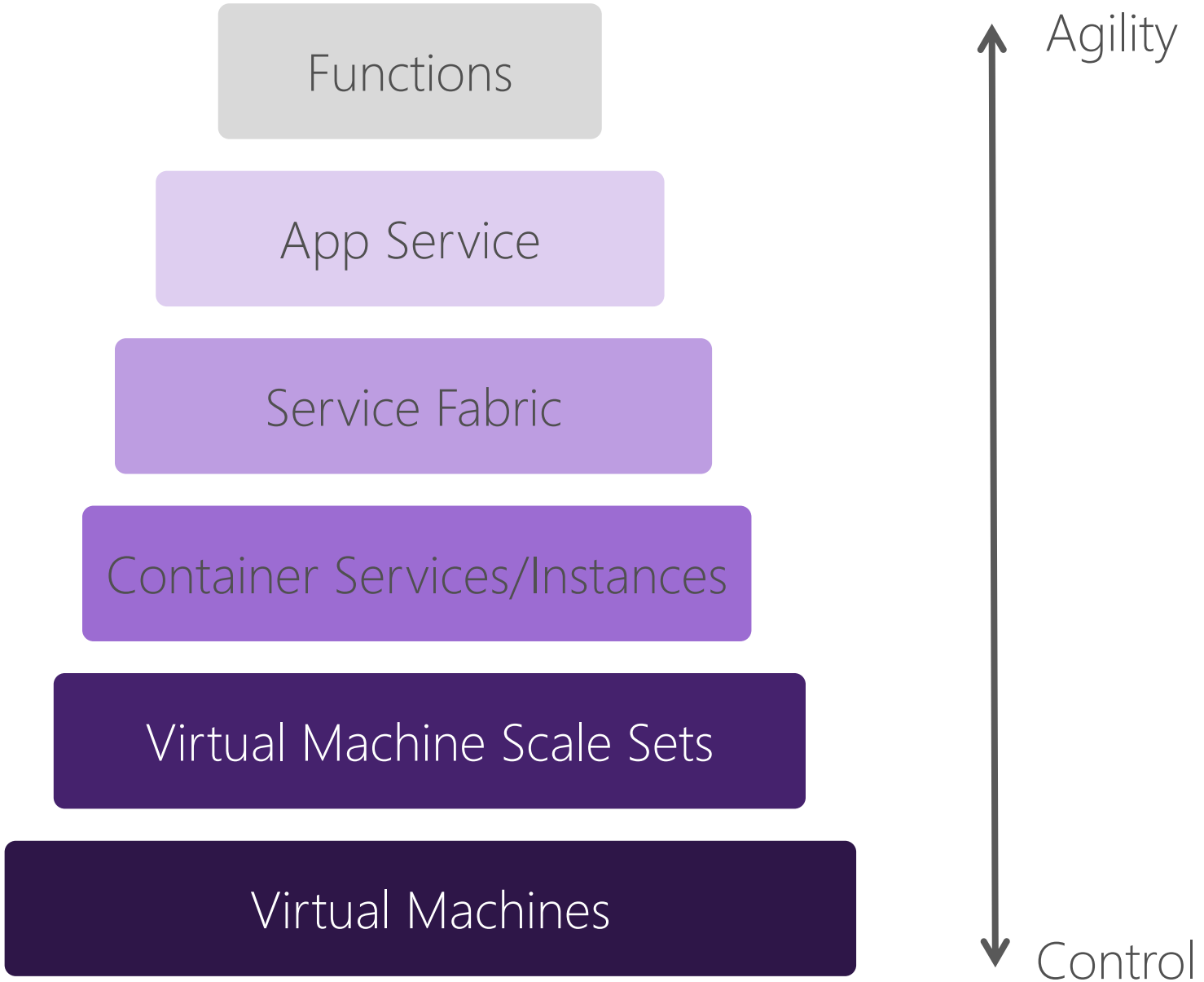
Journey to the Cloud – Our Roadmap



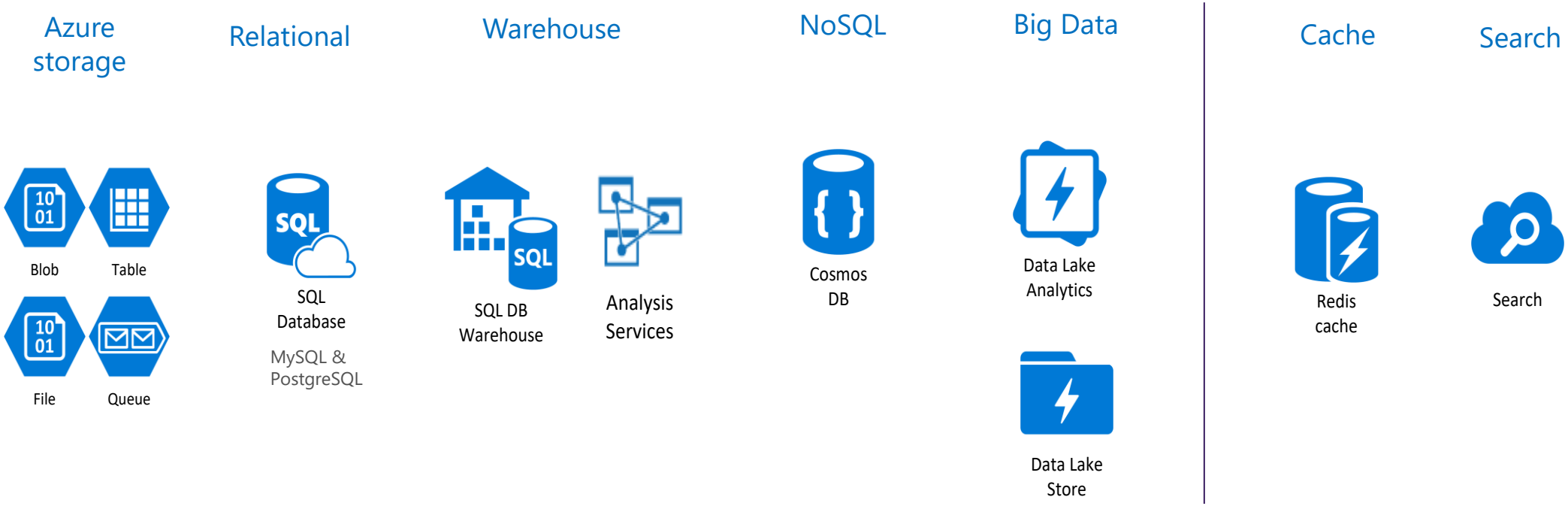


Edge DC	IaaS	PaaS	SaaS
Application	Application	Application	Application
Data	Data	Data	Data
Middleware	Middleware	Middleware	Middleware
Operating System	Operating System	Operating System	Operating System
Virtualisation	Virtualisation	Virtualisation	Virtualisation
Storage	Storage	Storage	Storage
Hardware	Hardware	Hardware	Hardware
Networking	Networking	Networking	Networking
Power	Power	Power	Power

Code


Functions
App Service
Service Fabric
Container Services/Instances
Virtual Machine Scale Sets
Virtual Machines
Agility
Control

Data



Polyglot Persistence:

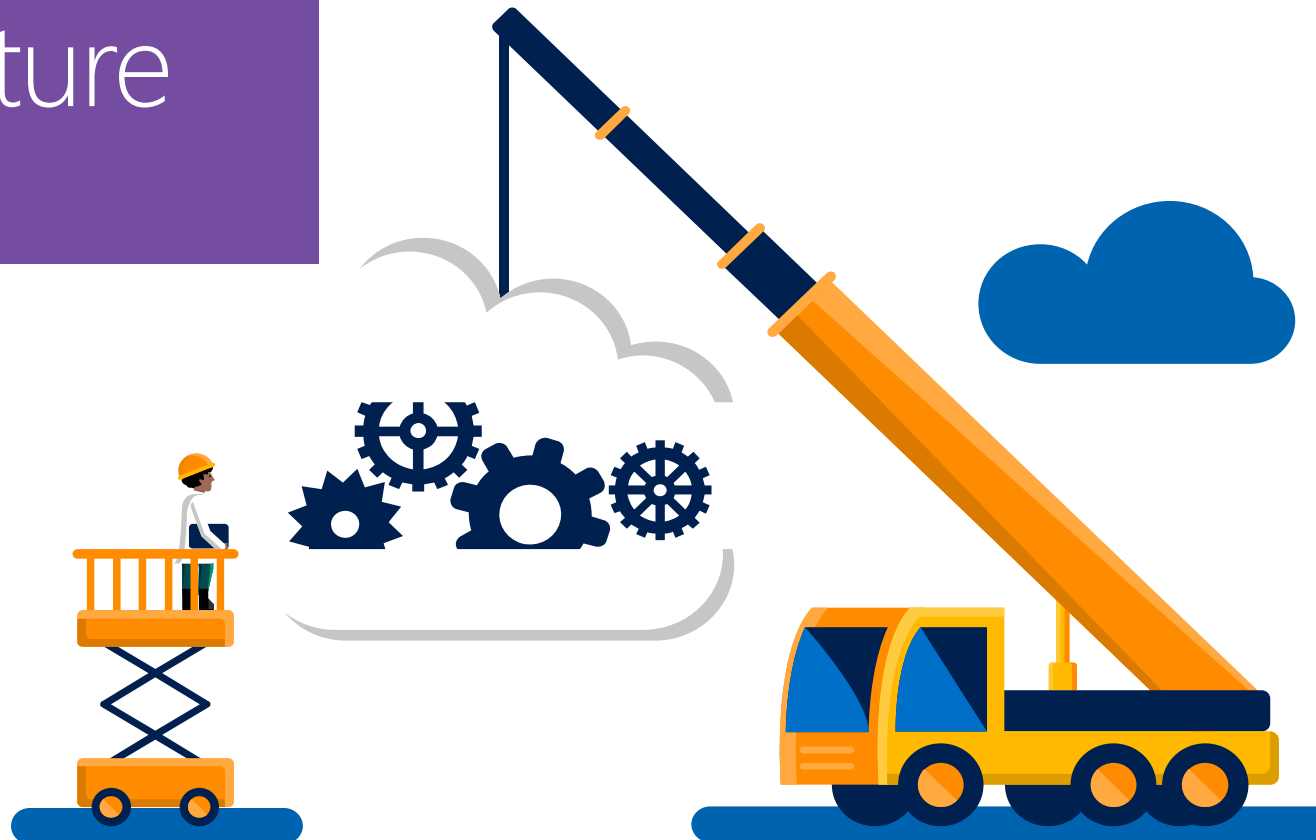
- **Human derived data** (sales, orders, parts, invoices, inventory, etc)
- **Machine derived data** (telemetry, sensor, movement, GPS, etc)

Stage 1 – Lift and Shift, Edge DC to IaaS

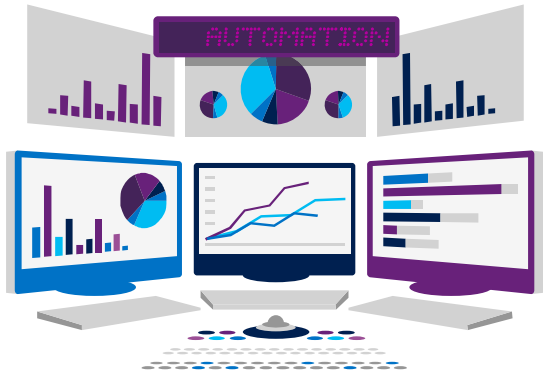
Lift and Shift your applications and move your infrastructure to Azure

Will Eastbury
Technical Evangelist

Microsoft



‘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

Microsoft

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"

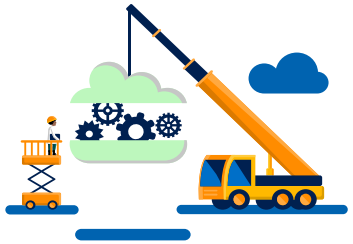
Ways to fire ARM deployments in Azure

- 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 ?

Microsoft

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

Microsoft

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}

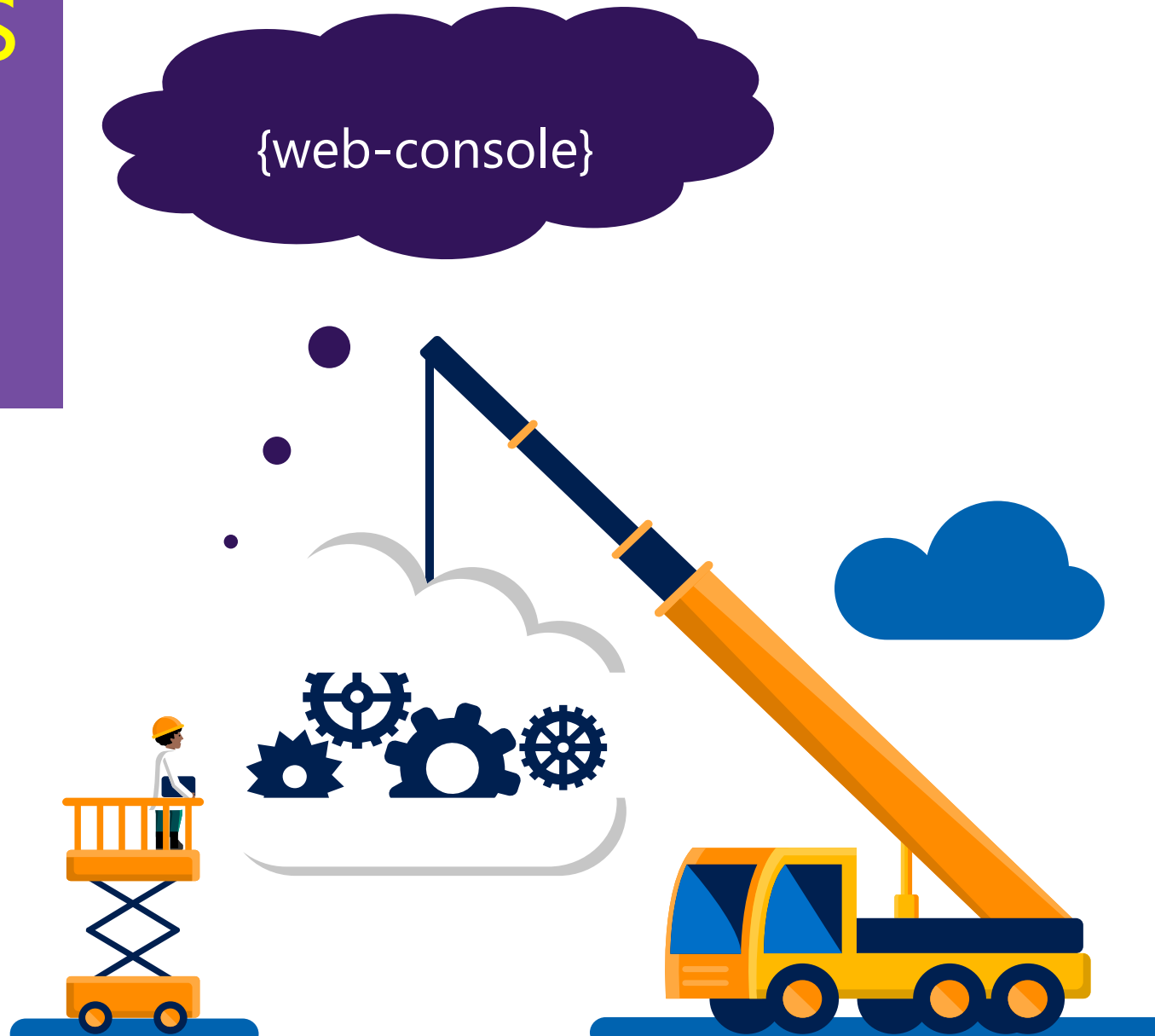
(With an ARM template)

Stage 3 - Serverless

Where's my console?

Will Eastbury &
David Gristwood
Technical Evangelists

Microsoft



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 Simplicity

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

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.

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 ?

- 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. 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

Why Azure Functions ?

Charging model - Serverless or PaaS

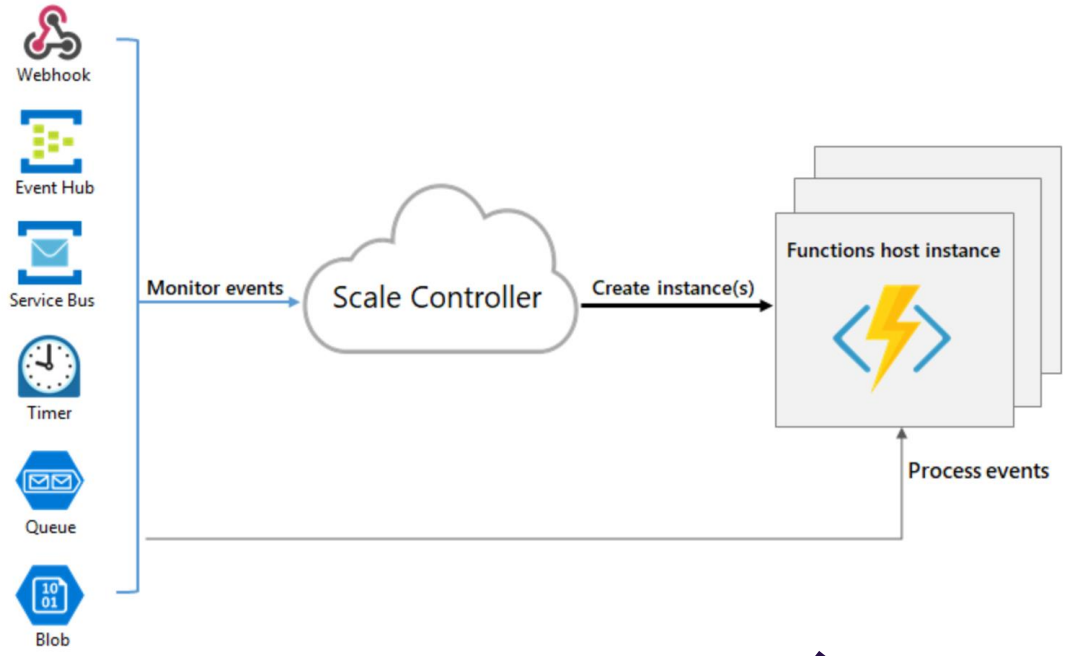
Plan		
Consumption	Serverless	Includes a monthly free grant of 1 million requests and 400,000 GB-s of resource consumption per month. After that it's charged at Execution Time * ~£0.000012/GB-s (1.2 ten thousandths of a penny for a gigabyte of RAM used for 1 second) + Total Executions * ~£0.15 per Million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

How Functions Work

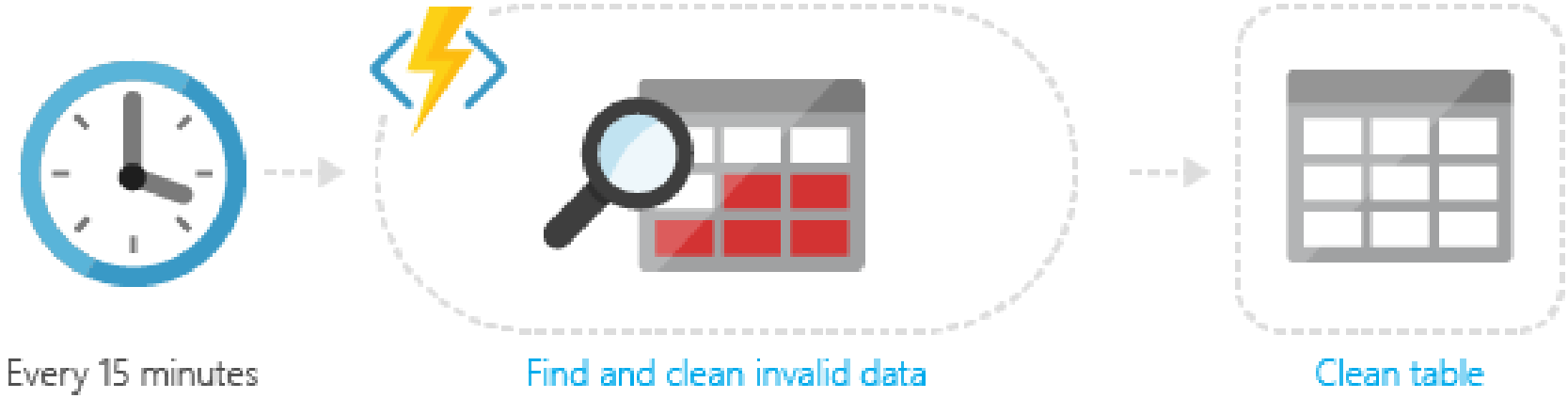


run.csx Save ▶ Run </> Get function

```
1  using System.Net;
2
3  public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, TraceWrit
4  {
5      // The sentiment category defaults to 'GREEN'.
6      string category = "GREEN";
7
8      // Get the sentiment score from the request body.
9      double score = await req.Content.ReadAsAsync<double>();
10     log.Info(string.Format("The sentiment score received is '{0}'.",
11                 score.ToString()));
12
13     // Set the category based on the sentiment score.
14     if (score < .3)
15     {
16         category = "RED";
17     }
18     else if (score < .6)
19     {
20         category = "YELLOW";
21     }
22     return req.CreateResponse(HttpStatusCode.OK, category);
```

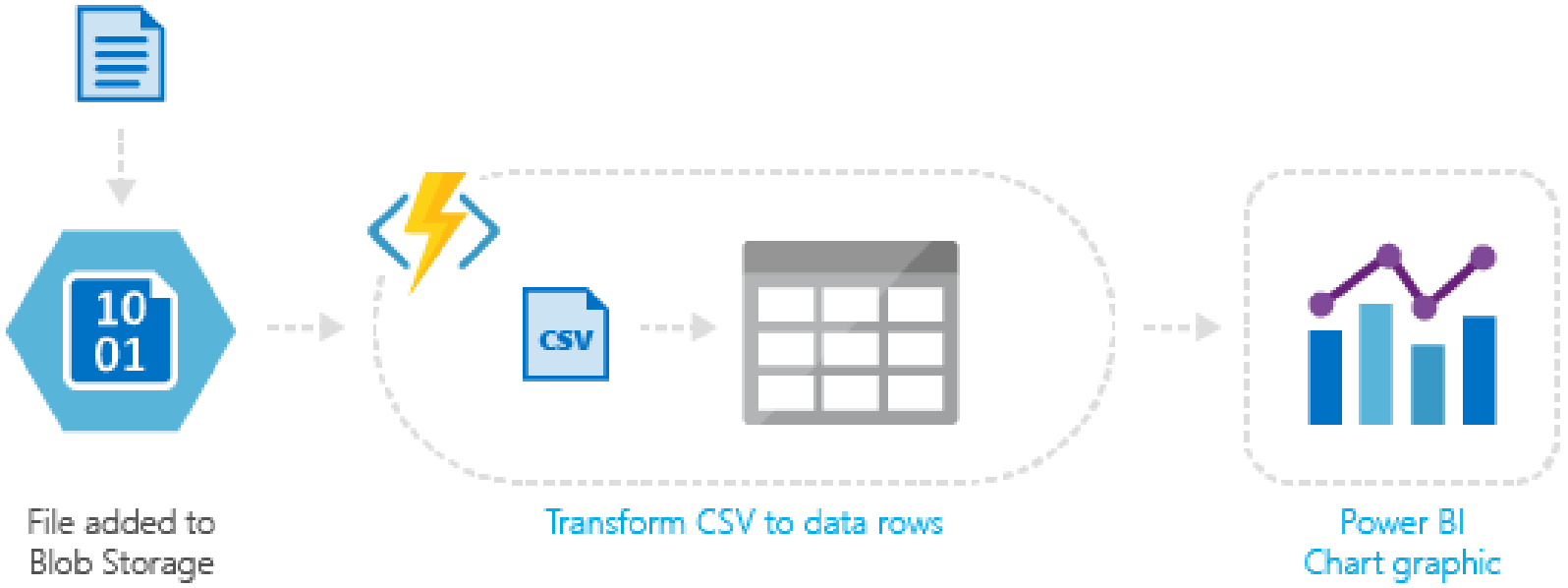
Common Scenarios for Functions

- 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



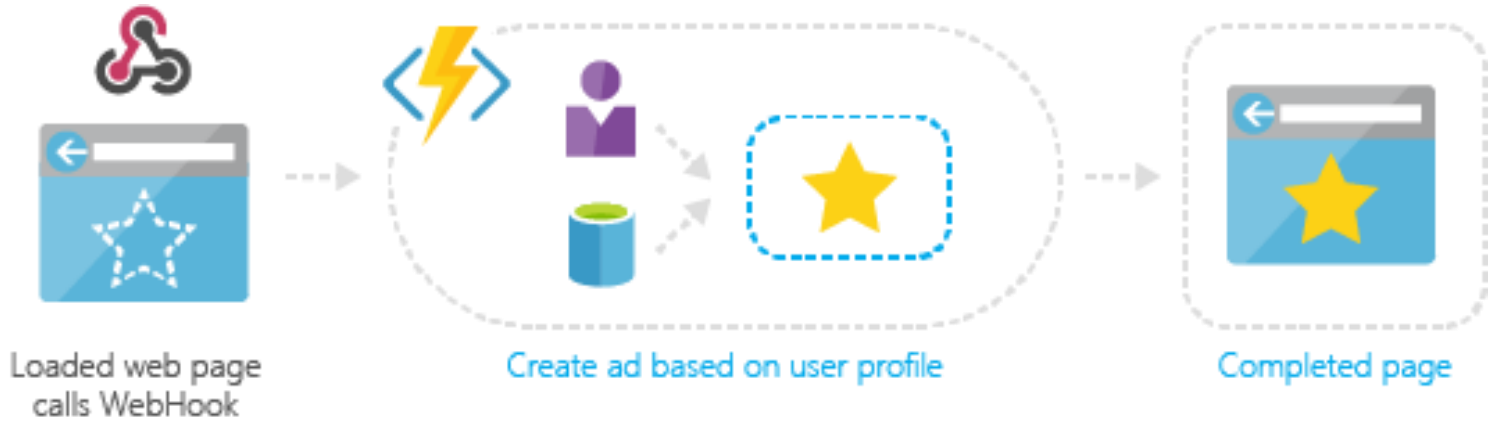
Common Scenarios for Functions

- 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

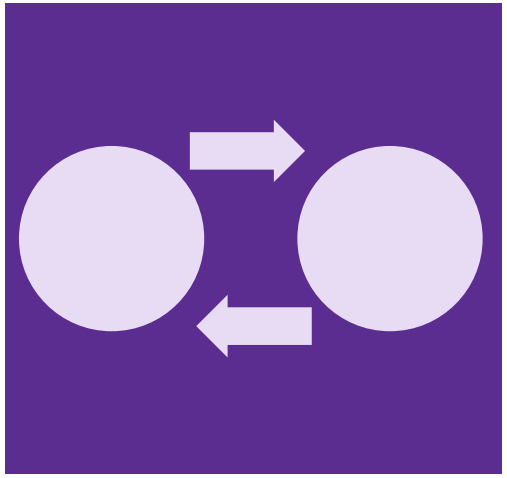


Common Scenarios for Functions

- 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

Microsoft

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

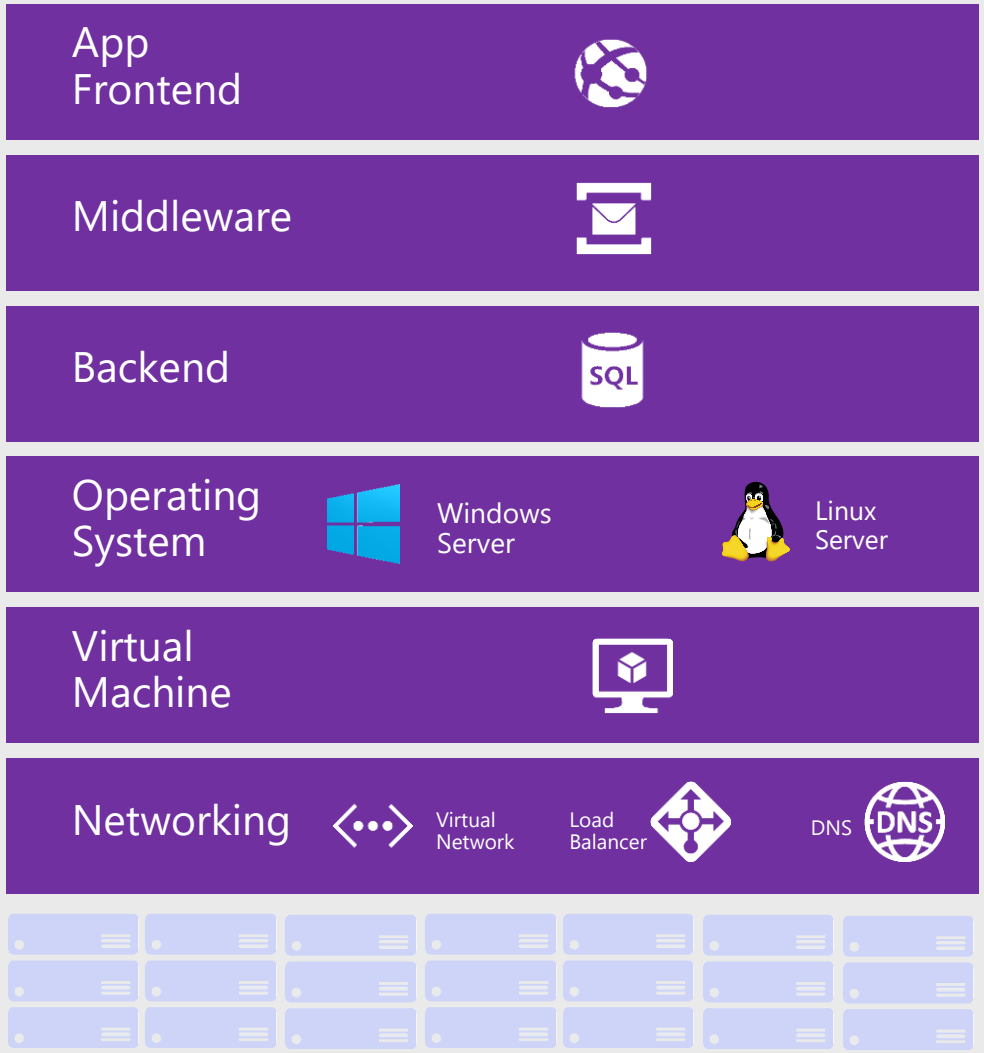
Test + Stabilize

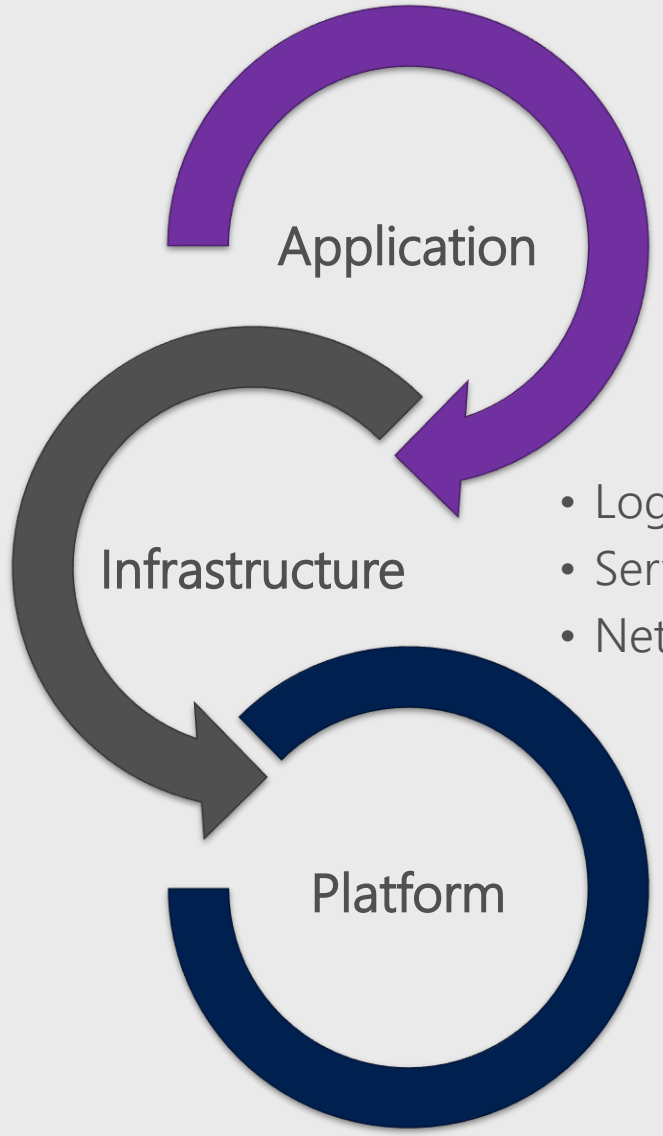
Deploy + Maintain

Traditional Monitoring

Continuous Monitoring

Bridging the Gap Between App and Infra





- Application Insights
- Log Analytics
- Service Map
- Network Watcher
- Azure Monitor
- Azure Advisor

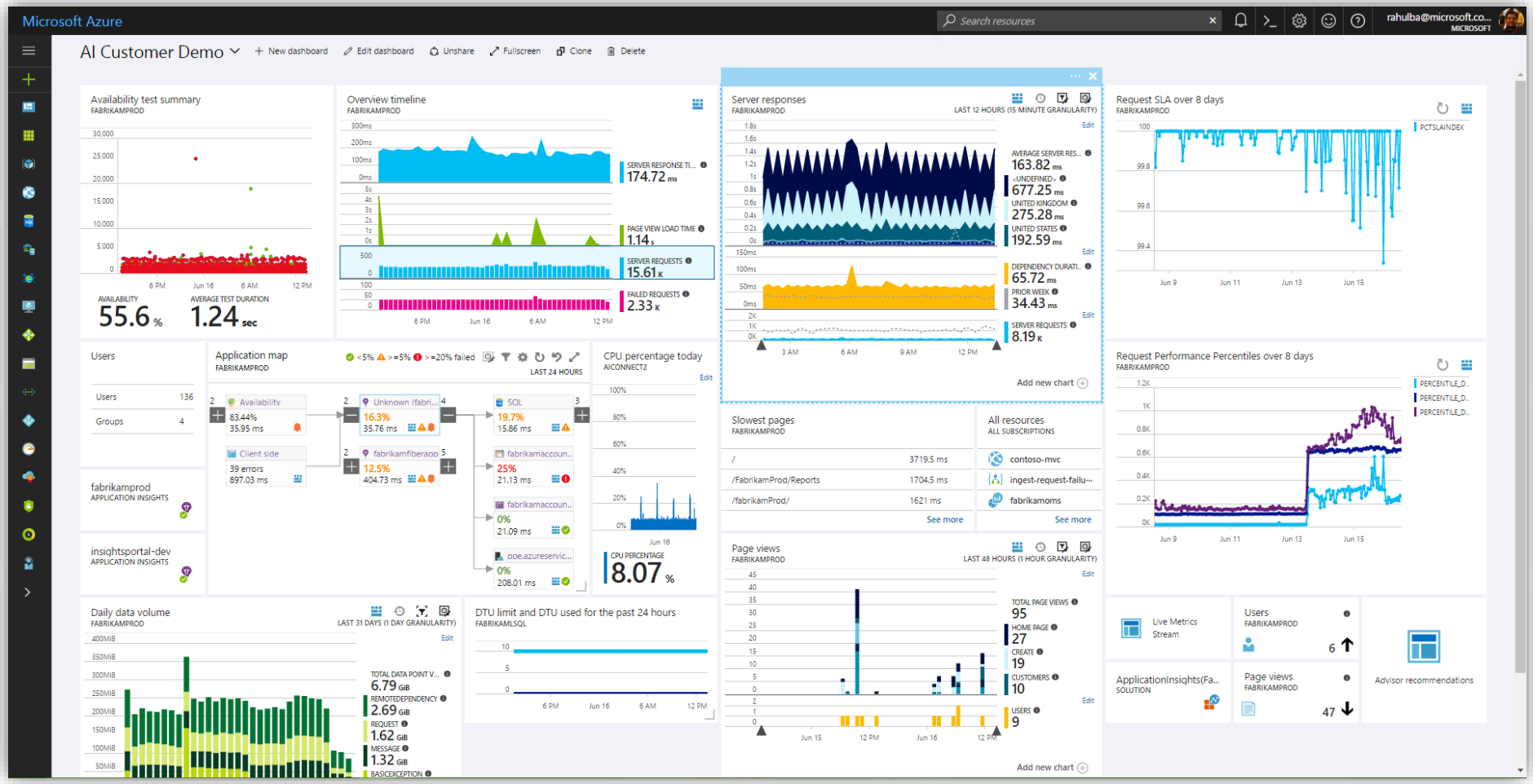
Azure Application Insights

Health Check

Monitor & Optimize

Detect & Debug

ML/Data Analytics



Generate Alerts

Take Actions

Customer Insights

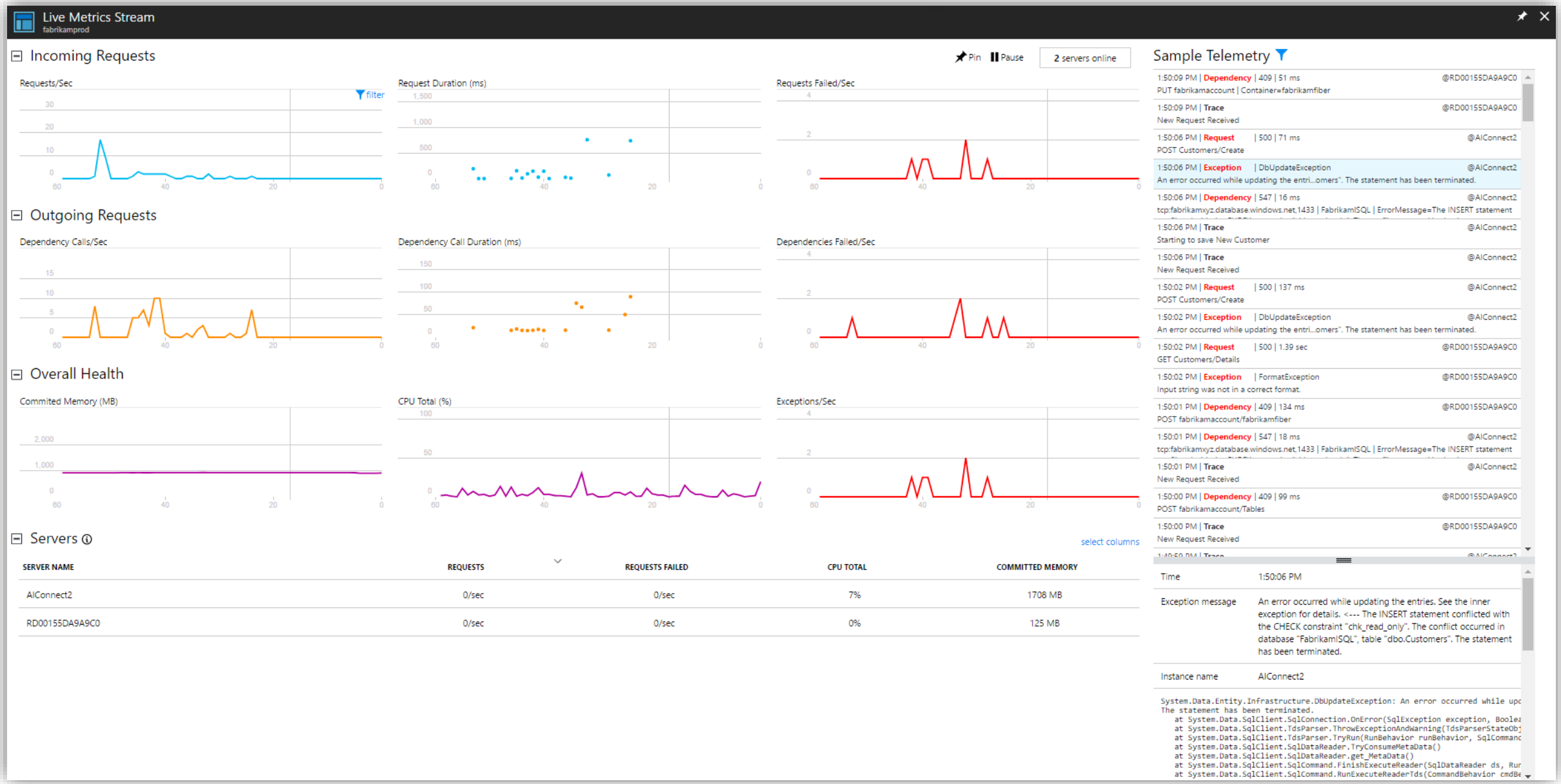
Export & Correlate

It's Demo Time:

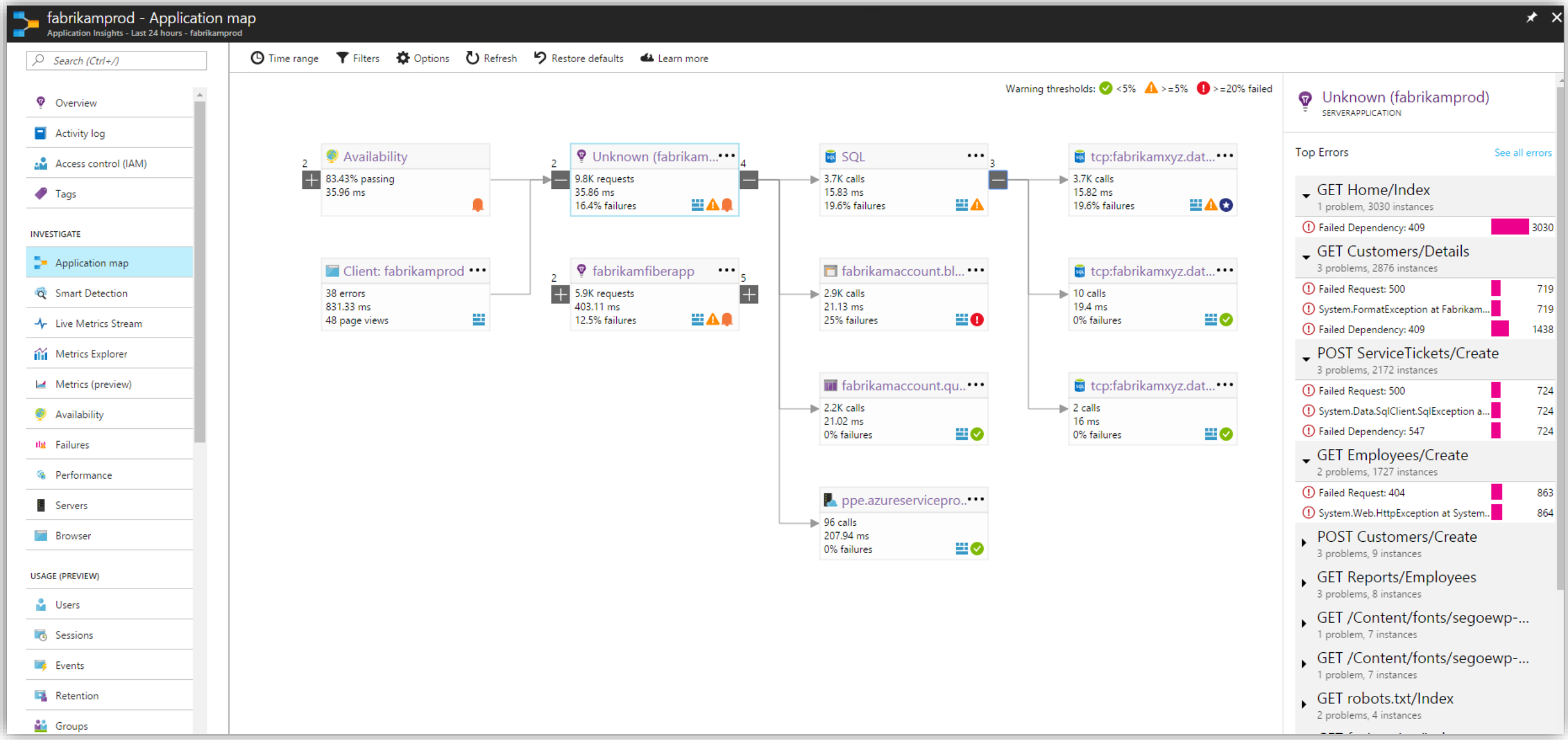
On the 5th Step, You just gained

Insights in {Your App}

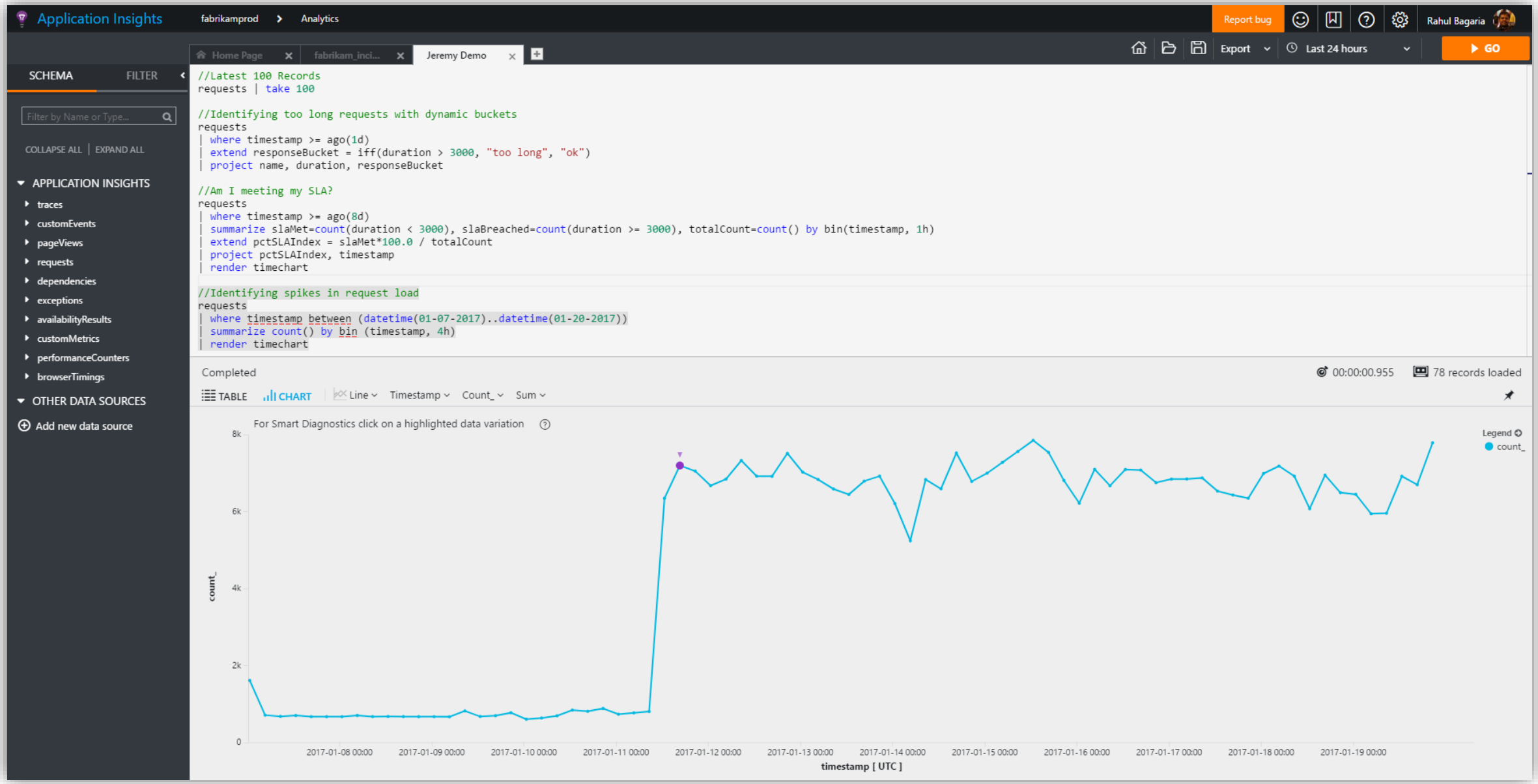
Real Time Health Check



Application Topology



Ad-Hoc Data Analytics



It's Demo Time:

On the 6th Step, You just gained Insights in

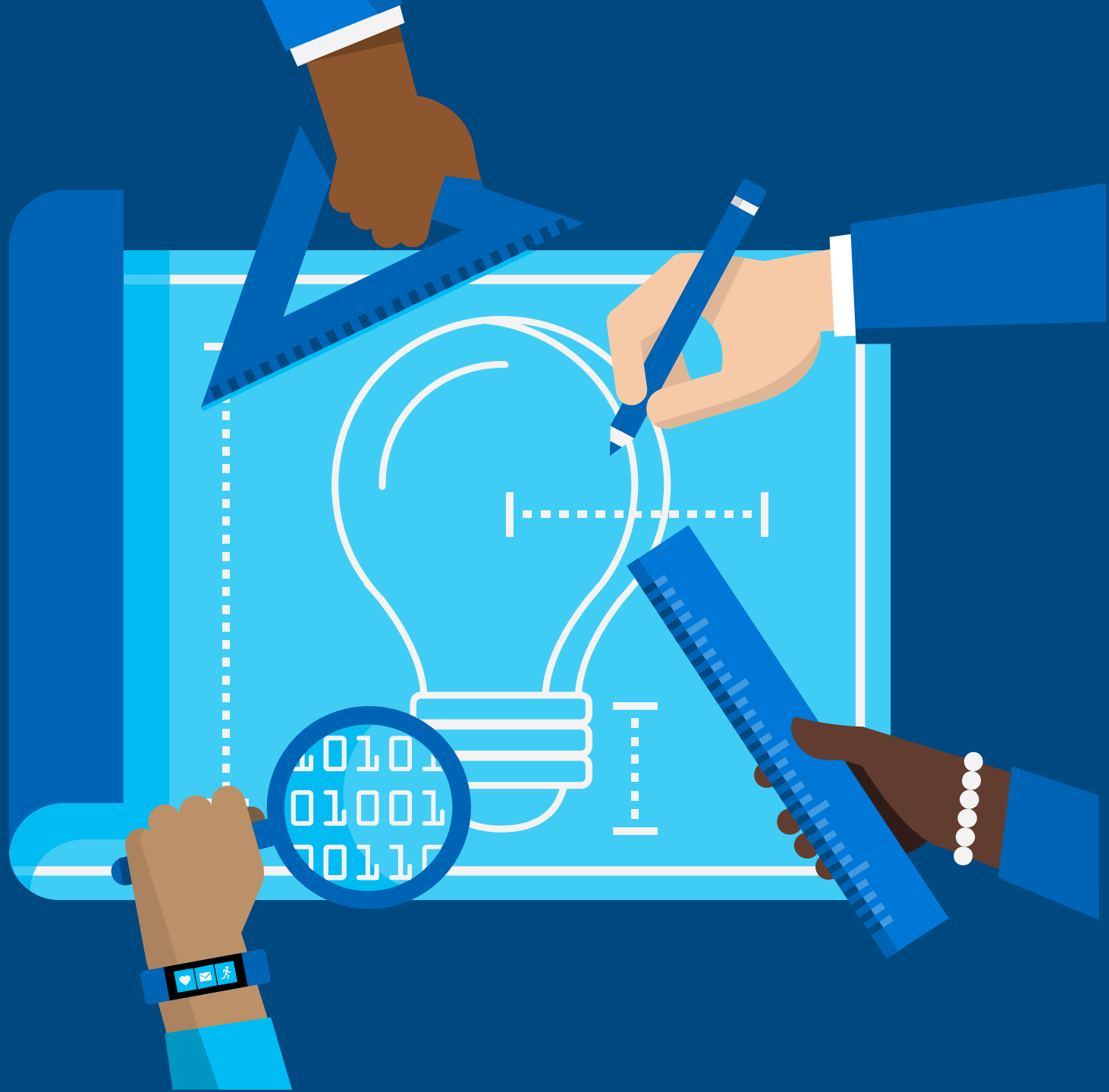
{Deep data analytics}

Microsoft UK

One Commercial Partner (OCP) Team

Typical Microsoft Partner Journey

Technical Enablement



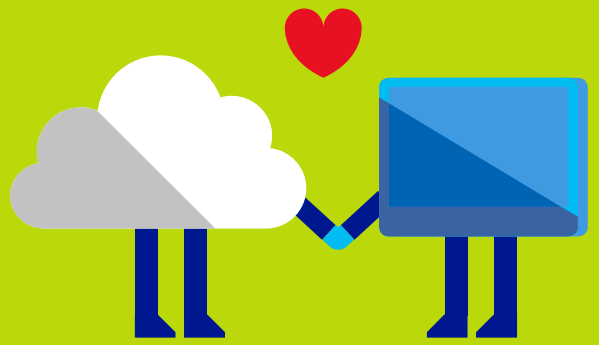
Develop Partnerships

Marketing Resources

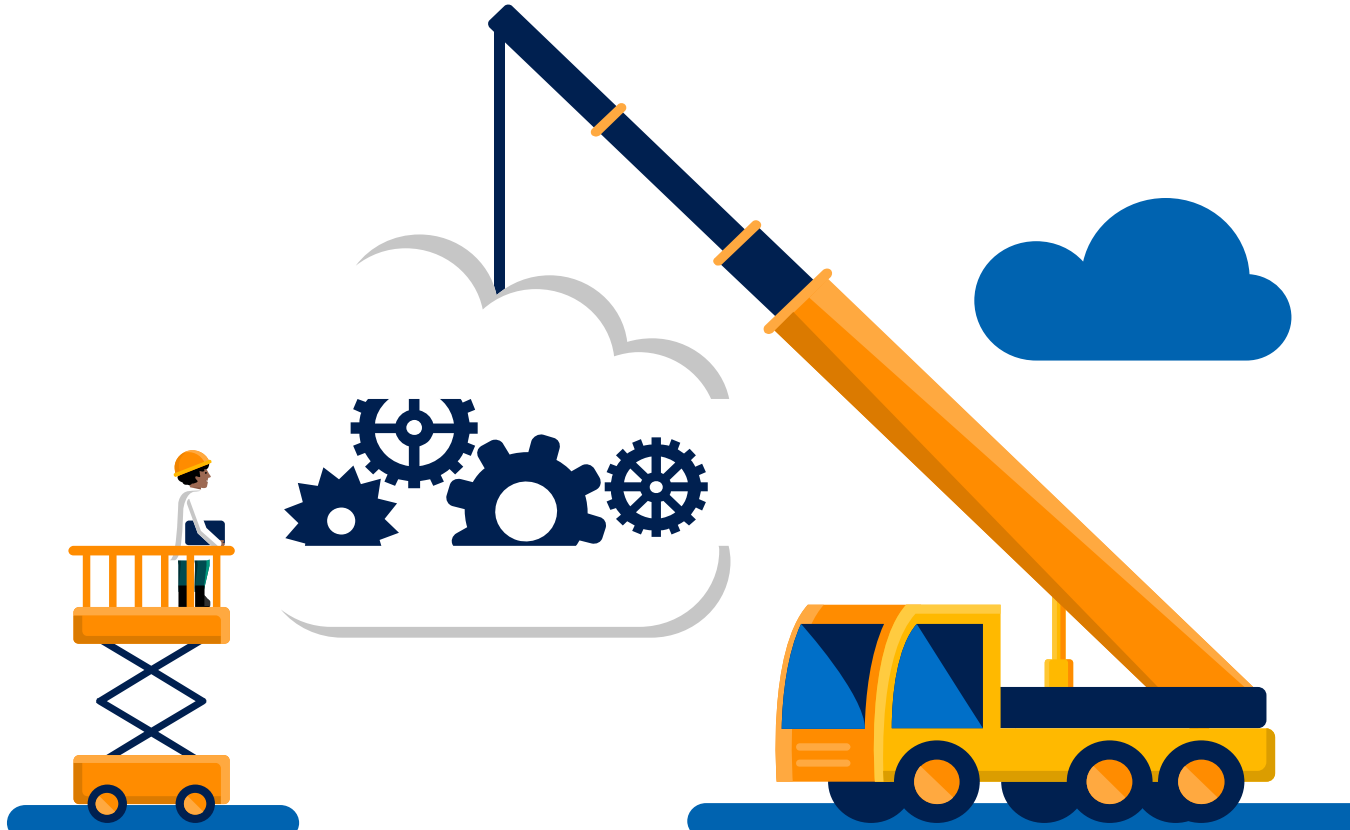
Adam Jackson

Partner Development Manager

adam.jackson@microsoft.com



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



Microsoft

Stage 5 – AI & ML

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

Mike Ormond &
Gabriel Nepomuceno
Technical Evangelists

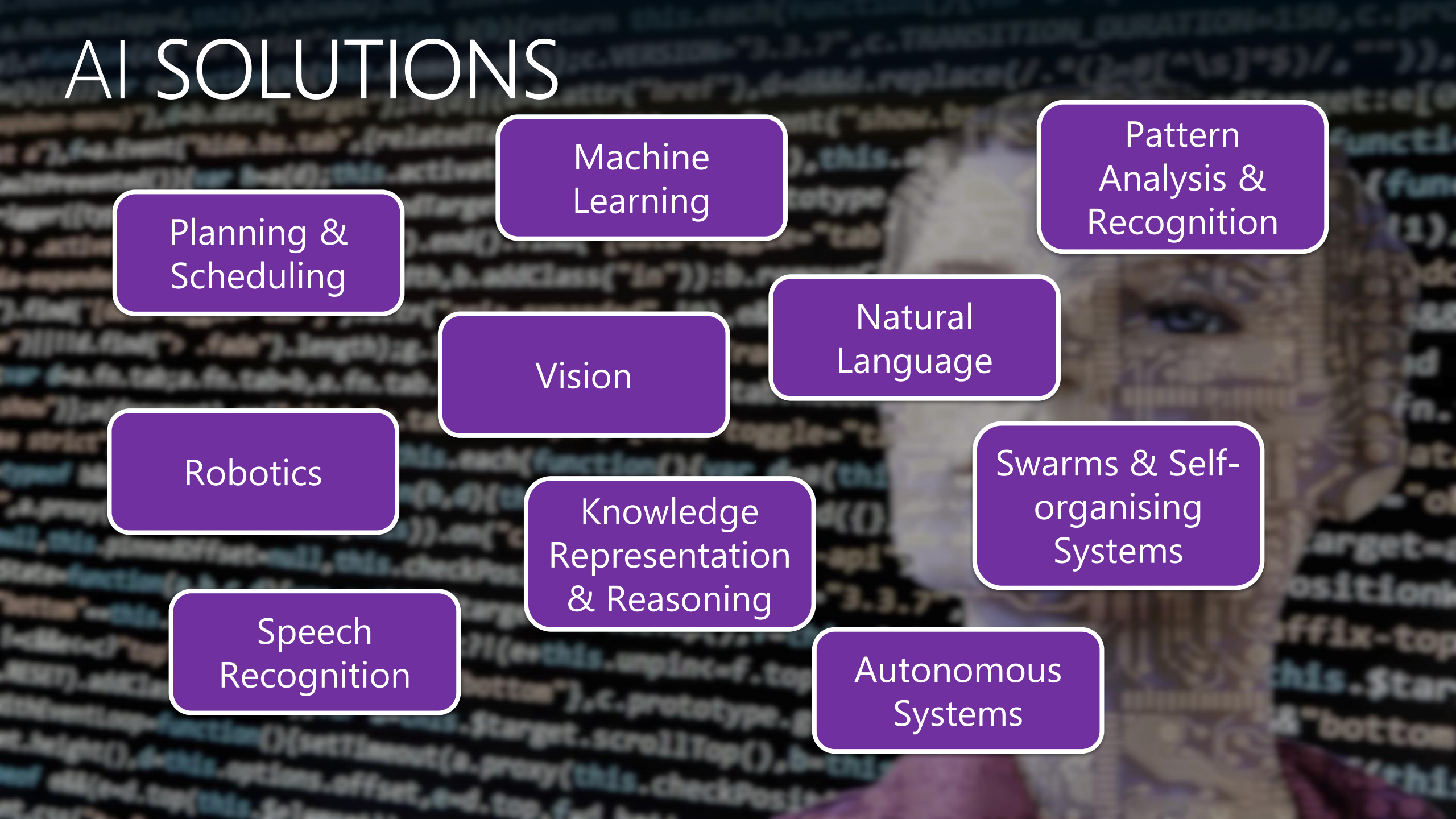
Microsoft

$$7 \times 6 = ?$$

75 x 75= ?

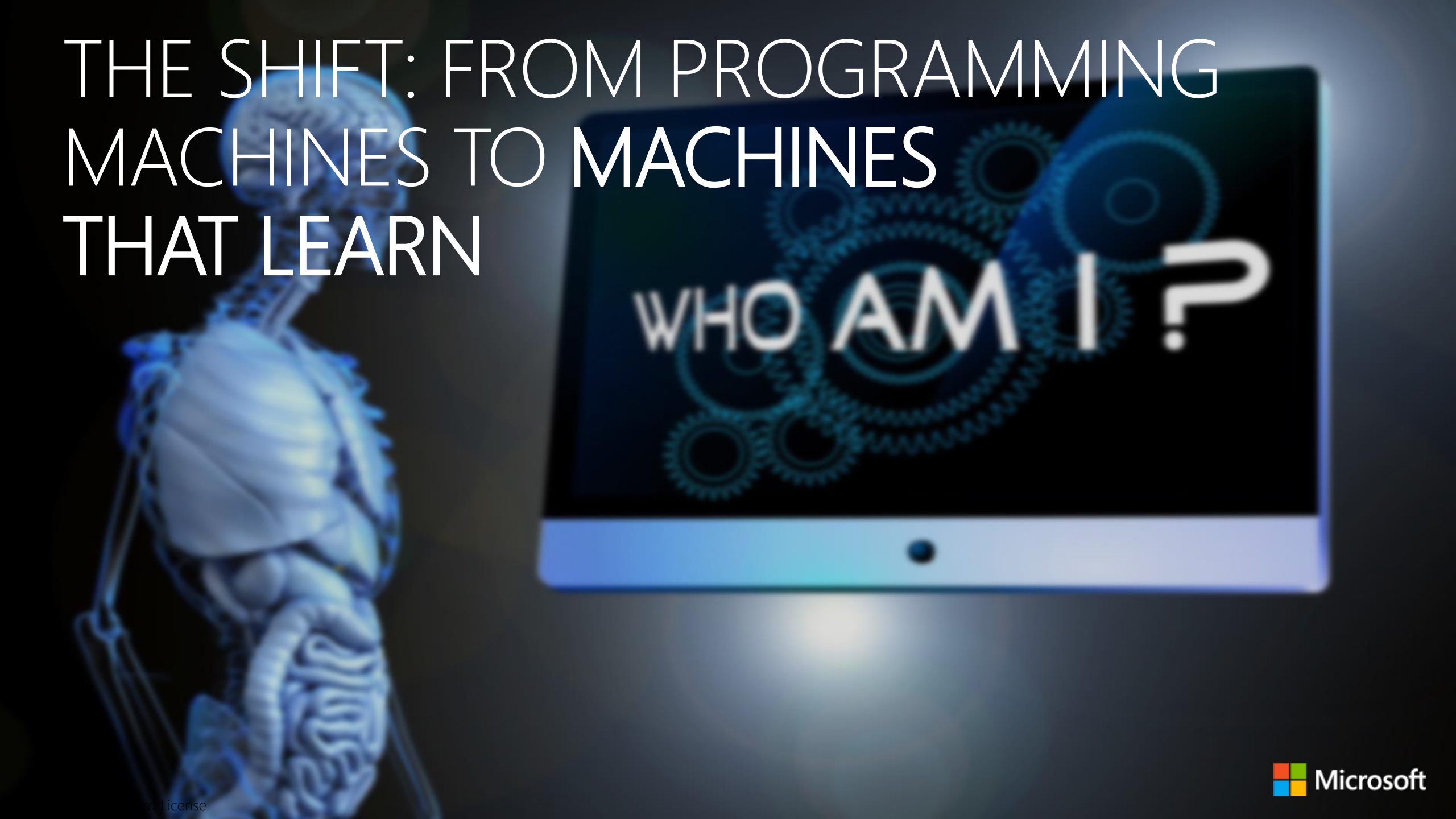
569 x 176= ?

AI SOLUTIONS



“Amplify human ingenuity”

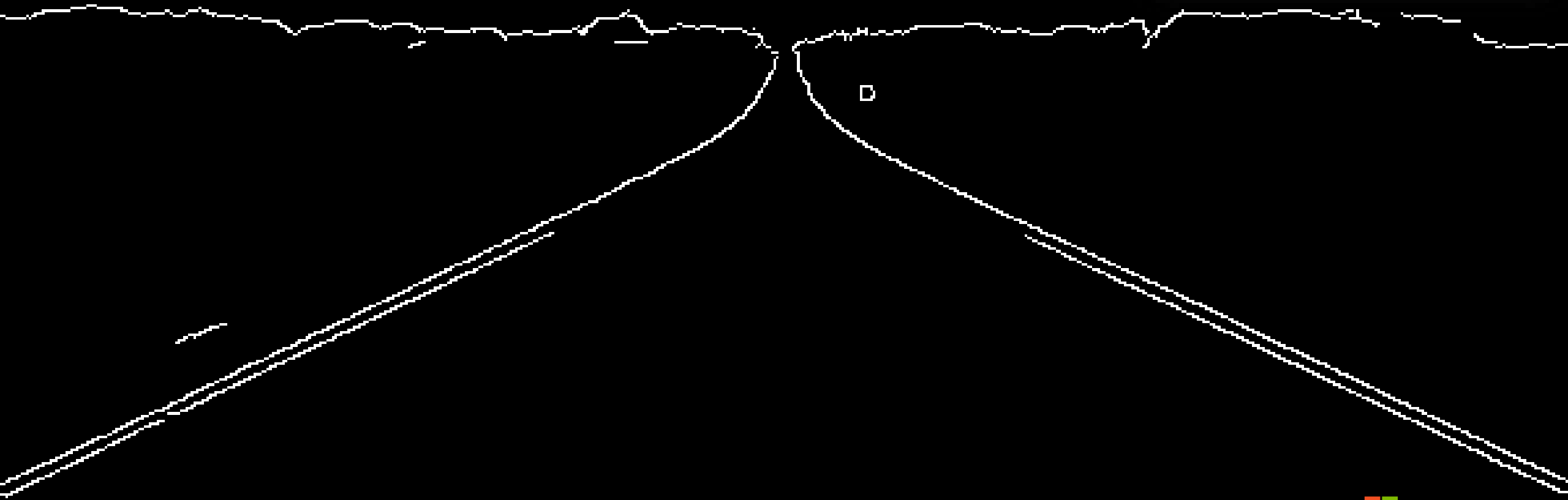
THE SHIFT: FROM PROGRAMMING MACHINES TO MACHINES THAT LEARN



Microsoft

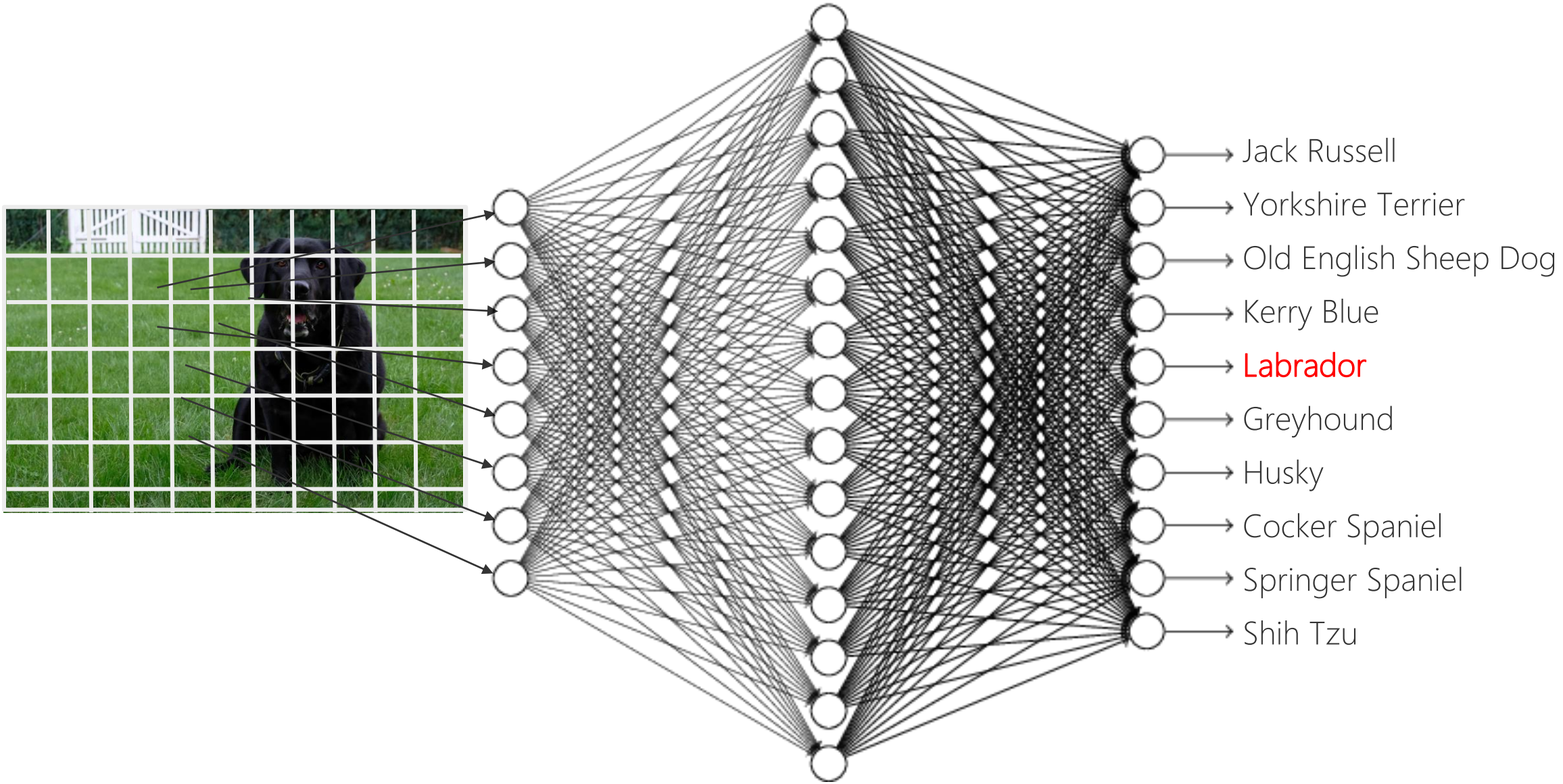
MACHINE LEARNING FOR DOGS

Classical AI



Microsoft

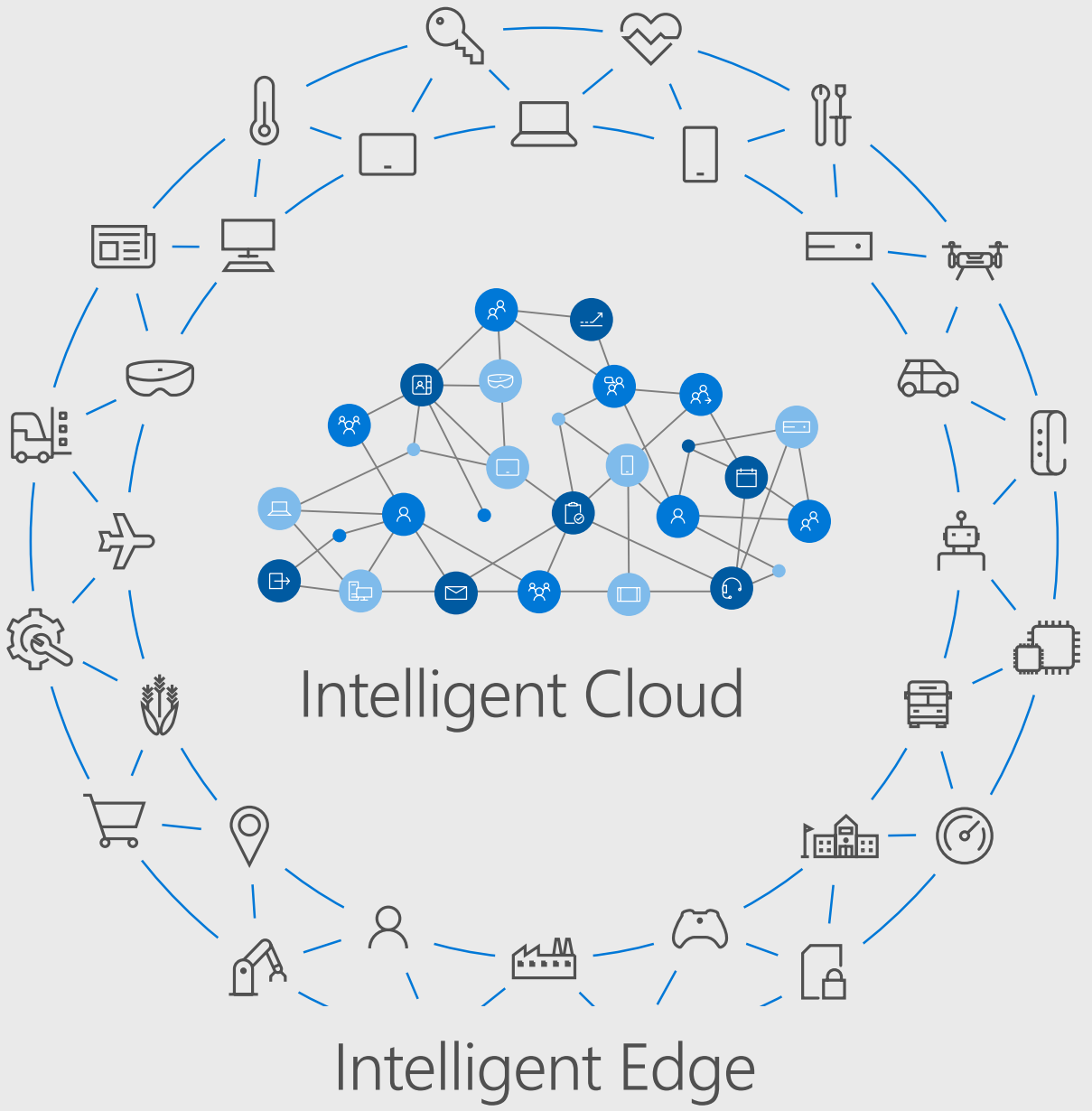
ARTIFICAL NEURAL NETWORKS



Multi-device

Artificial Intelligence

Serverless



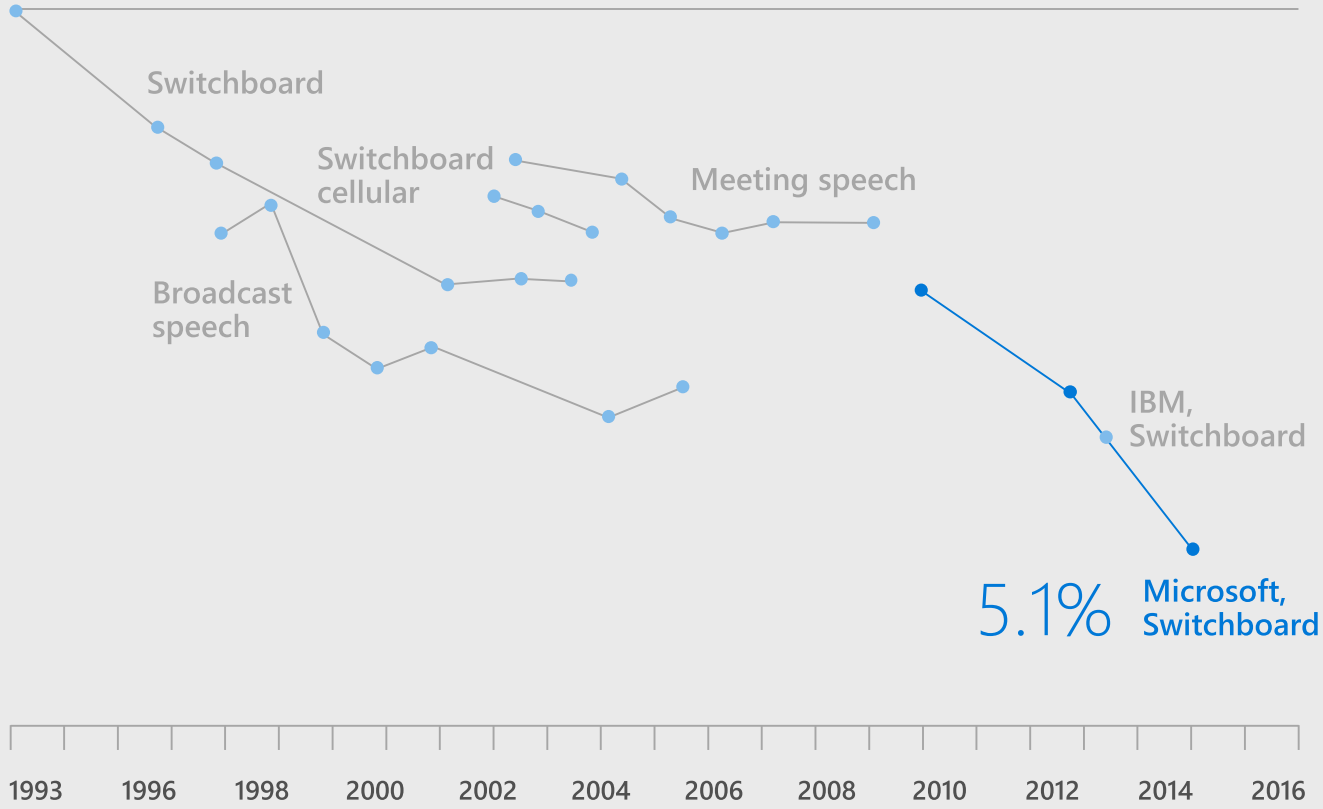
Breakthroughs in speech

5.1% human error rate

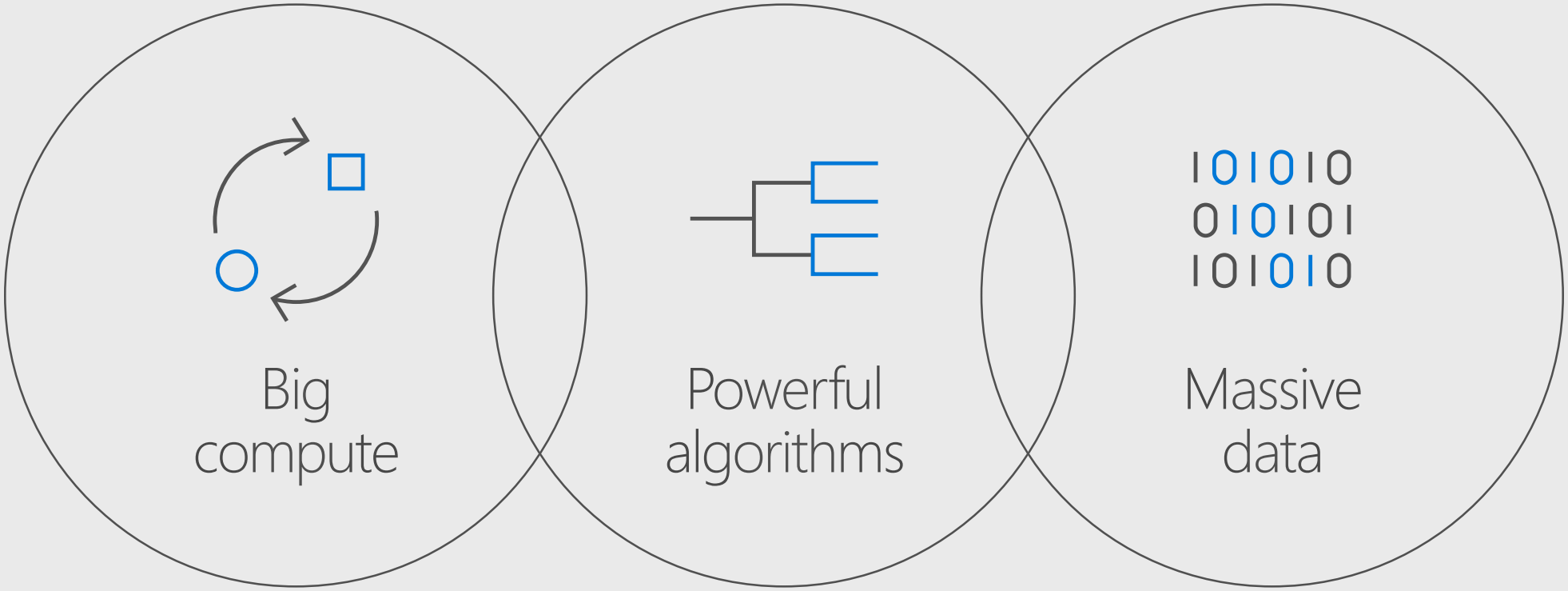
Switchboard

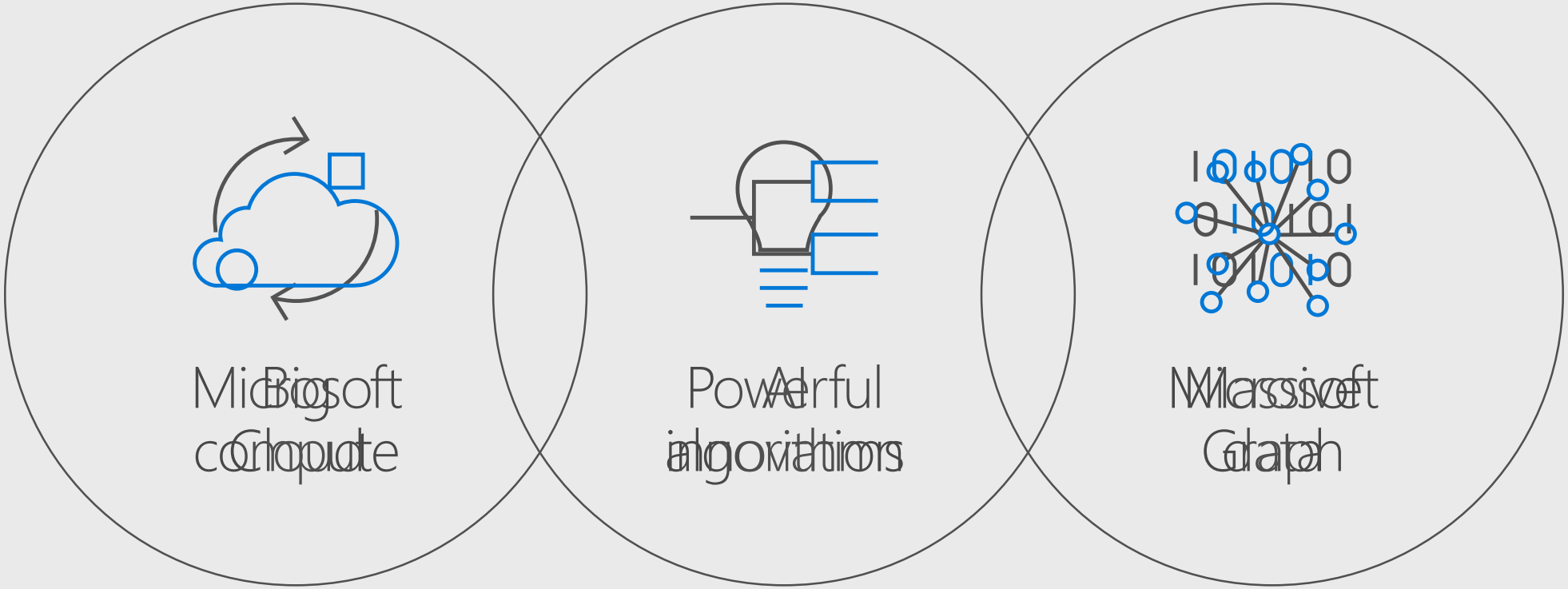
Loud and clear

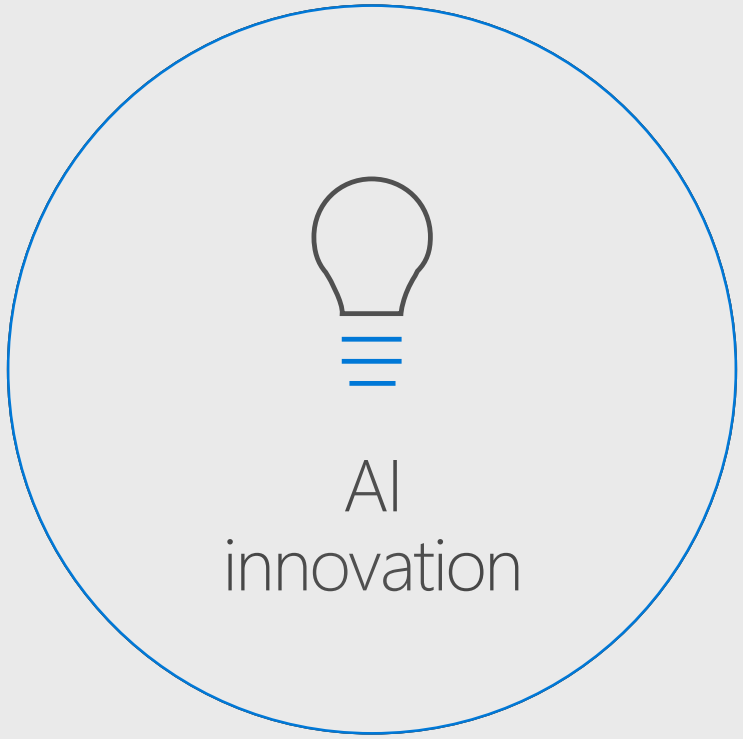
Speech-recognition word-error rate, selected benchmarks, %



Sources: Microsoft: research papers


Big compute
Powerful algorithms
Massive data


Microsoft
Cloud
Big
compute
Powerful
algorithms
AI
innovations
Massive
Graph
Microsoft
data


AI
innovation

Bringing AI to every developer

Stage 4 – Integrate AI Services

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

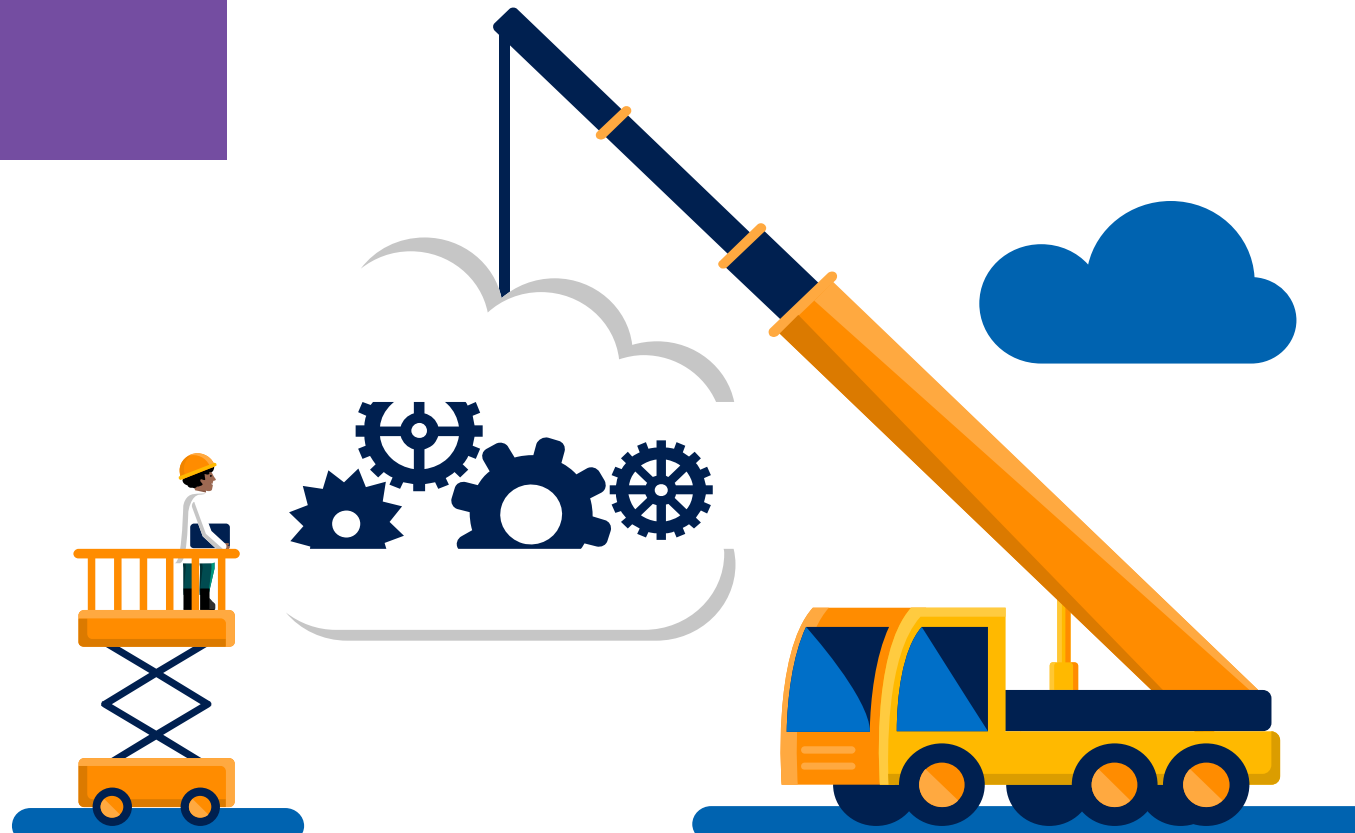
It's Demo Time:

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

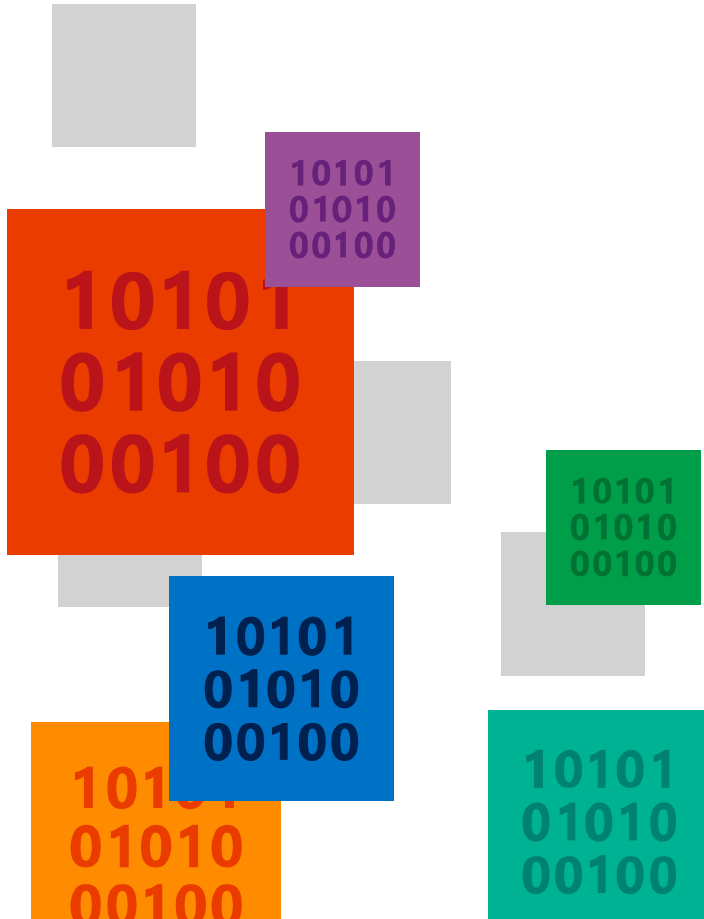
Will Eastbury
and David Gristwood
Technical Evangelists

Microsoft



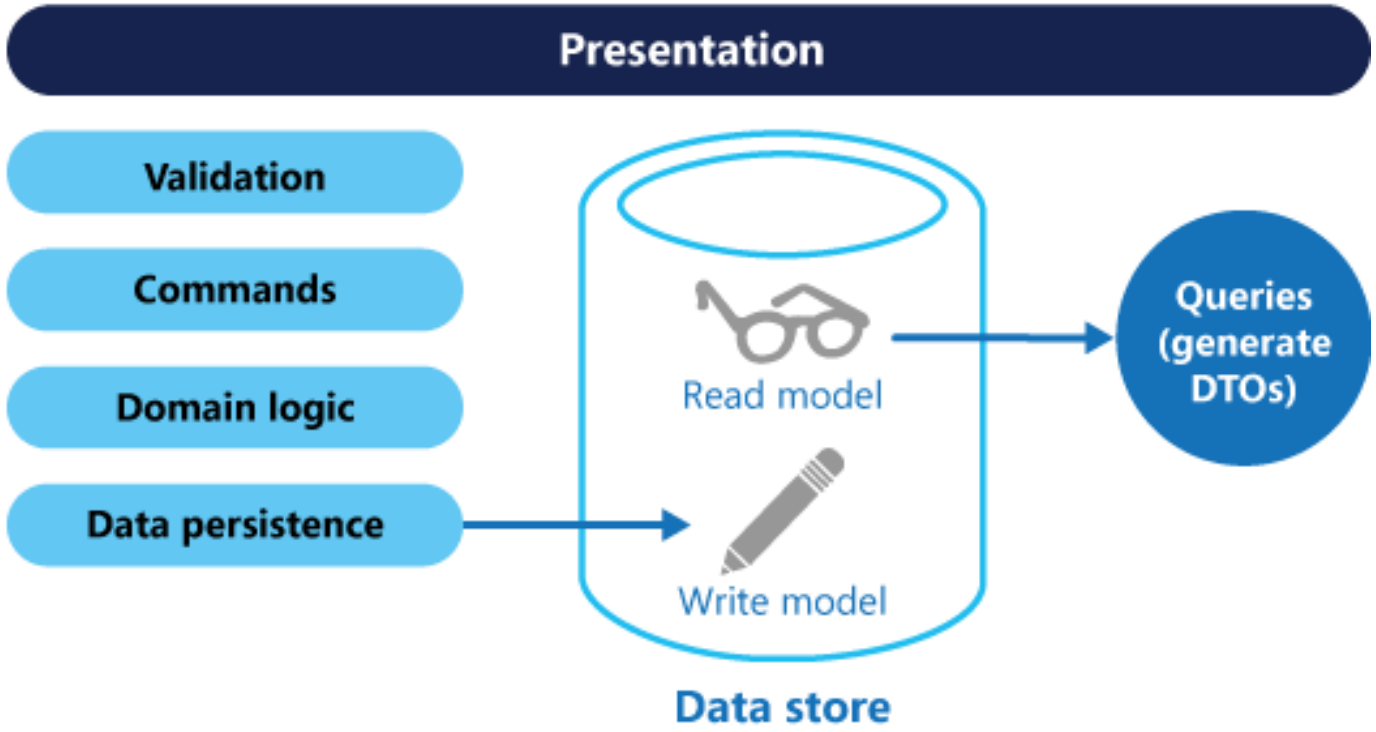
Architecture Patterns

Cloud Design Patterns
https://docs.microsoft.com/azure/architecture/patterns/



Microsoft

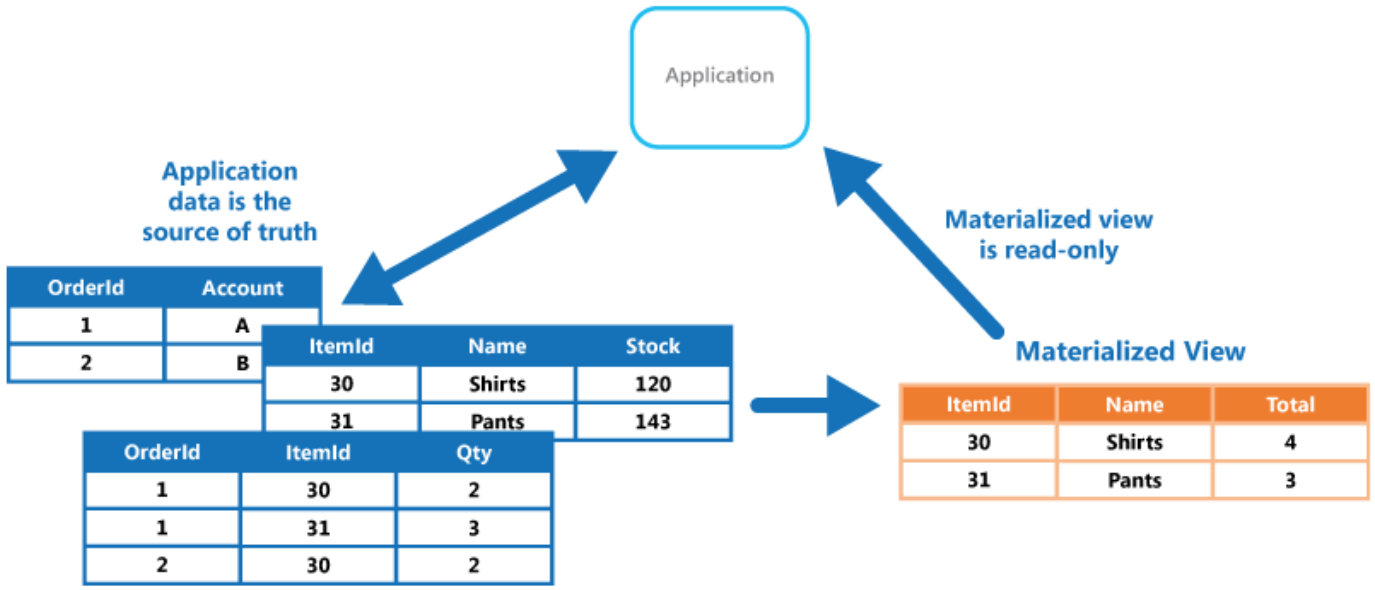
“Command and Query Responsibility Separation”



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

https://aka.ms/CQRS-Pattern

Materialized View pattern

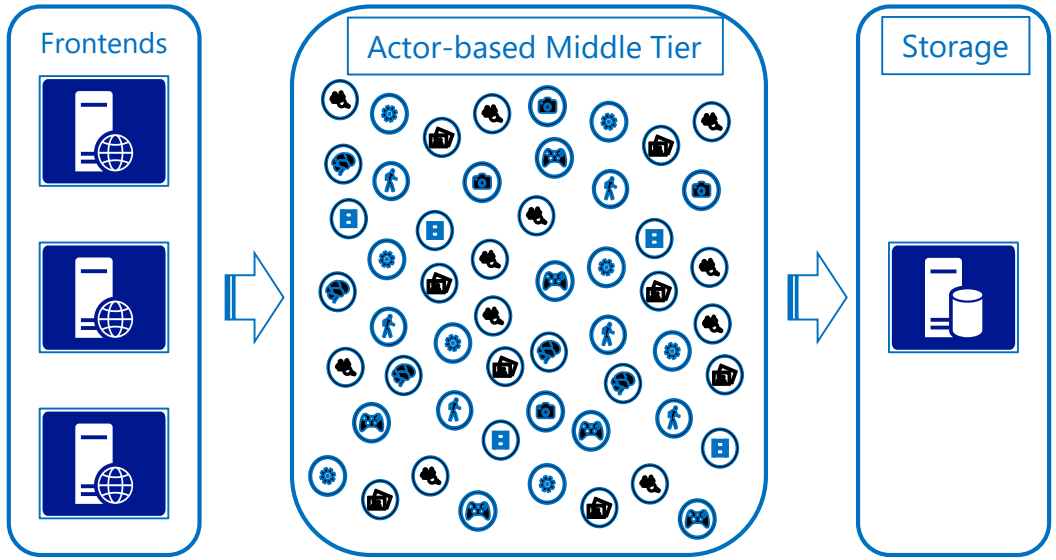


Data often optimised for storage rather than reads

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.

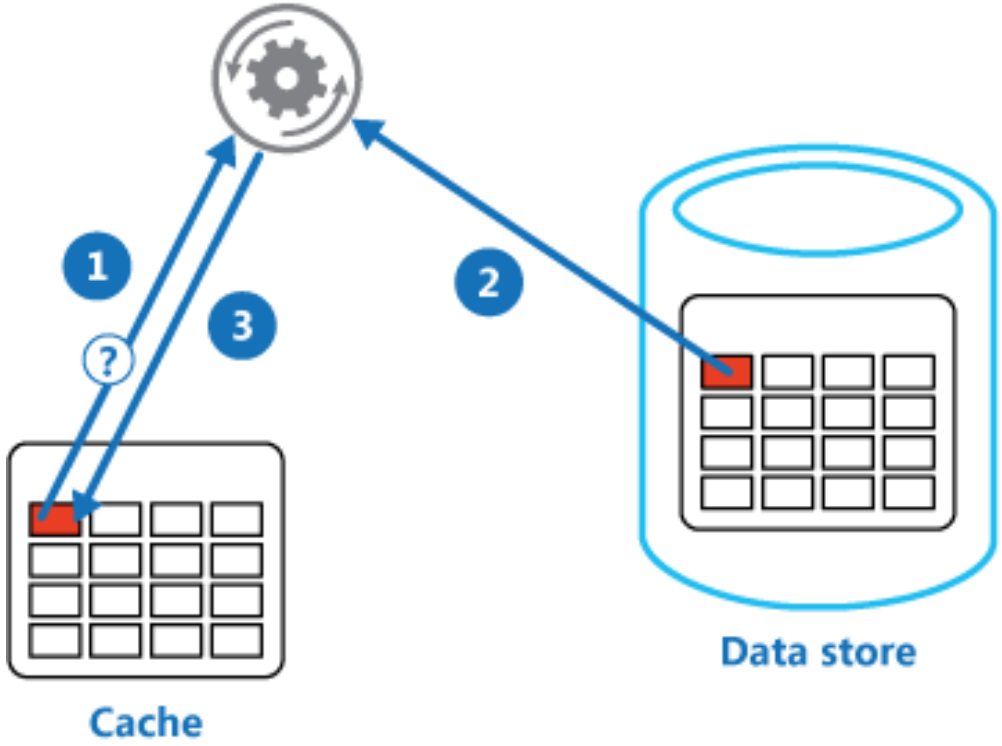
Often part of CQRS

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.

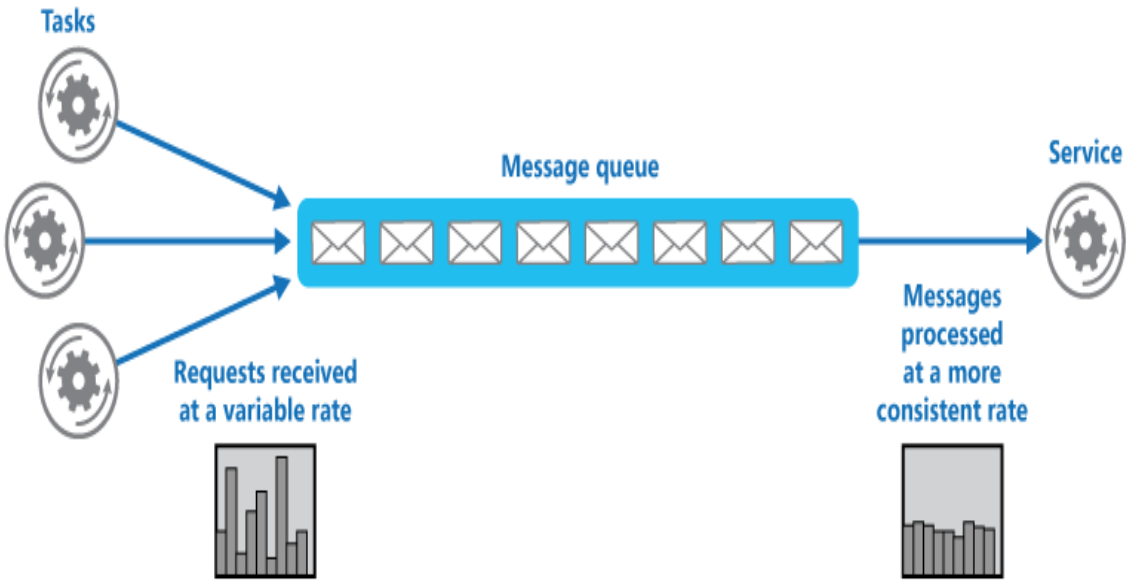
If data is not present, or it's stale, read it from the main store and pass it through into the main cache.

Issues and Considerations:

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

https://aka.ms/cache-aside-pattern

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*

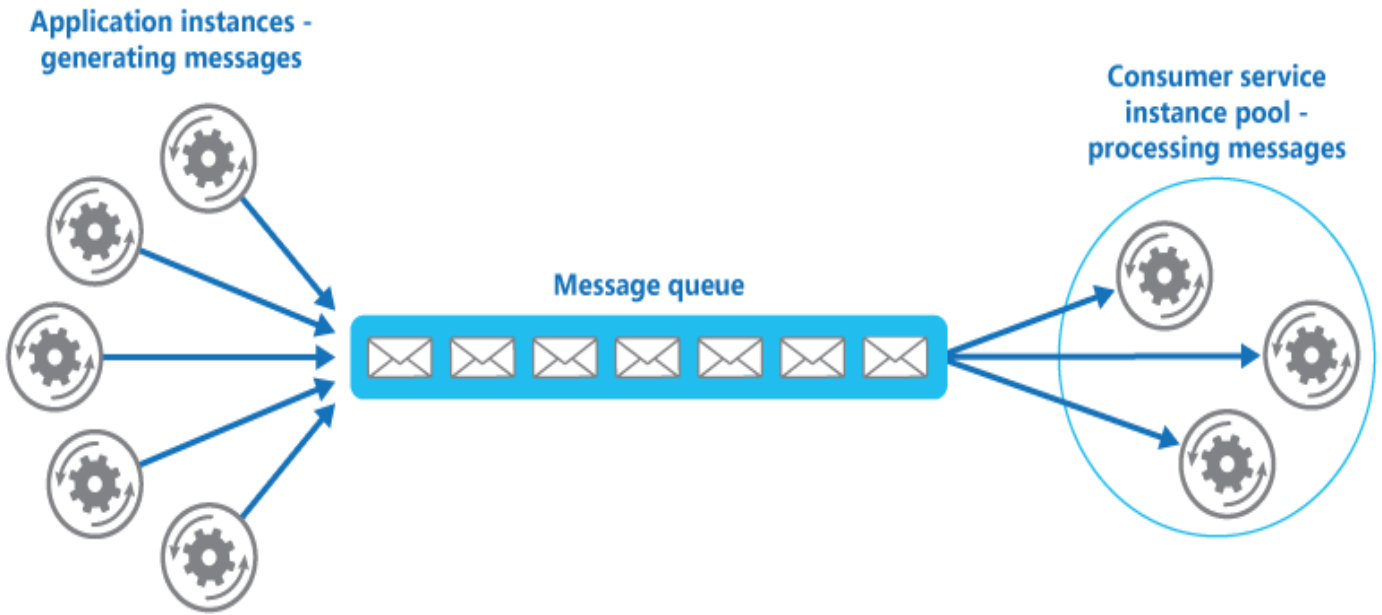
Use as a precursor to enable the competing consumers (autoscaling) pattern

Issues and Considerations:

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

https://aka.ms/queue-based-load-leveling-pattern

"Competing Consumers"



Issues and Considerations:

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

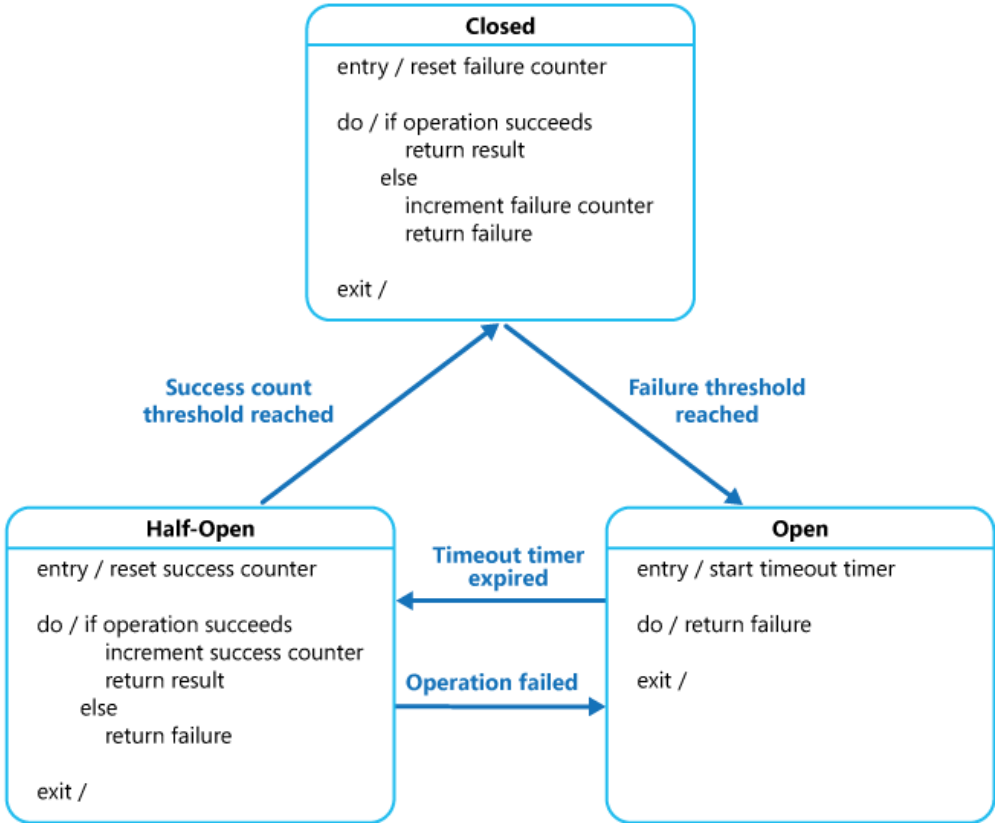
Scaling out via queues: when our queue consumer becomes a bottleneck, *simply run more instances of the consumer.*

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.

https://aka.ms/competing-consumers-pattern

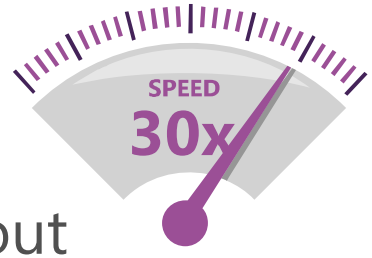
"Failover" and "Circuit Breaker"



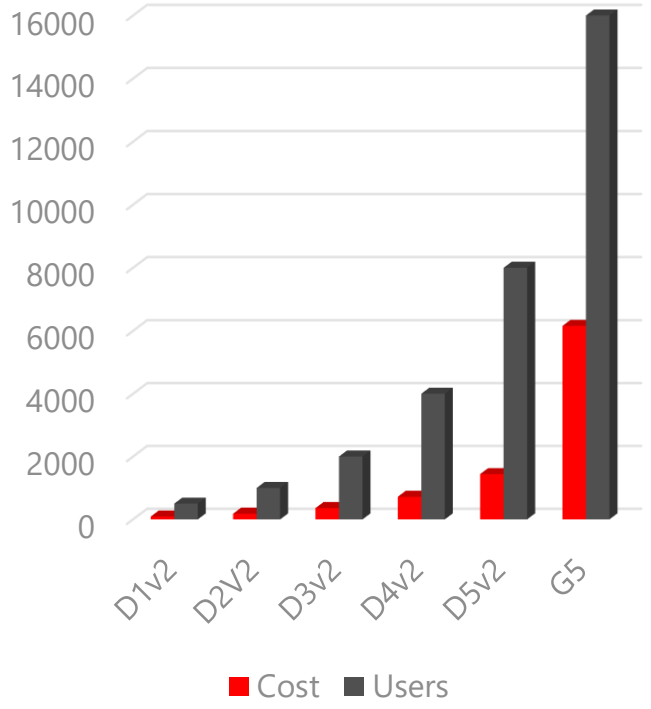
- When we know a system is down, behave accordingly and let it recover.
- If we have a service that can be used instead for full functionality or degraded operations then use that instead until our primary service is back online.
- Could encompass anything from secondary credit card processing services,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

https://aka.ms/Circuit-Breaker-Pattern

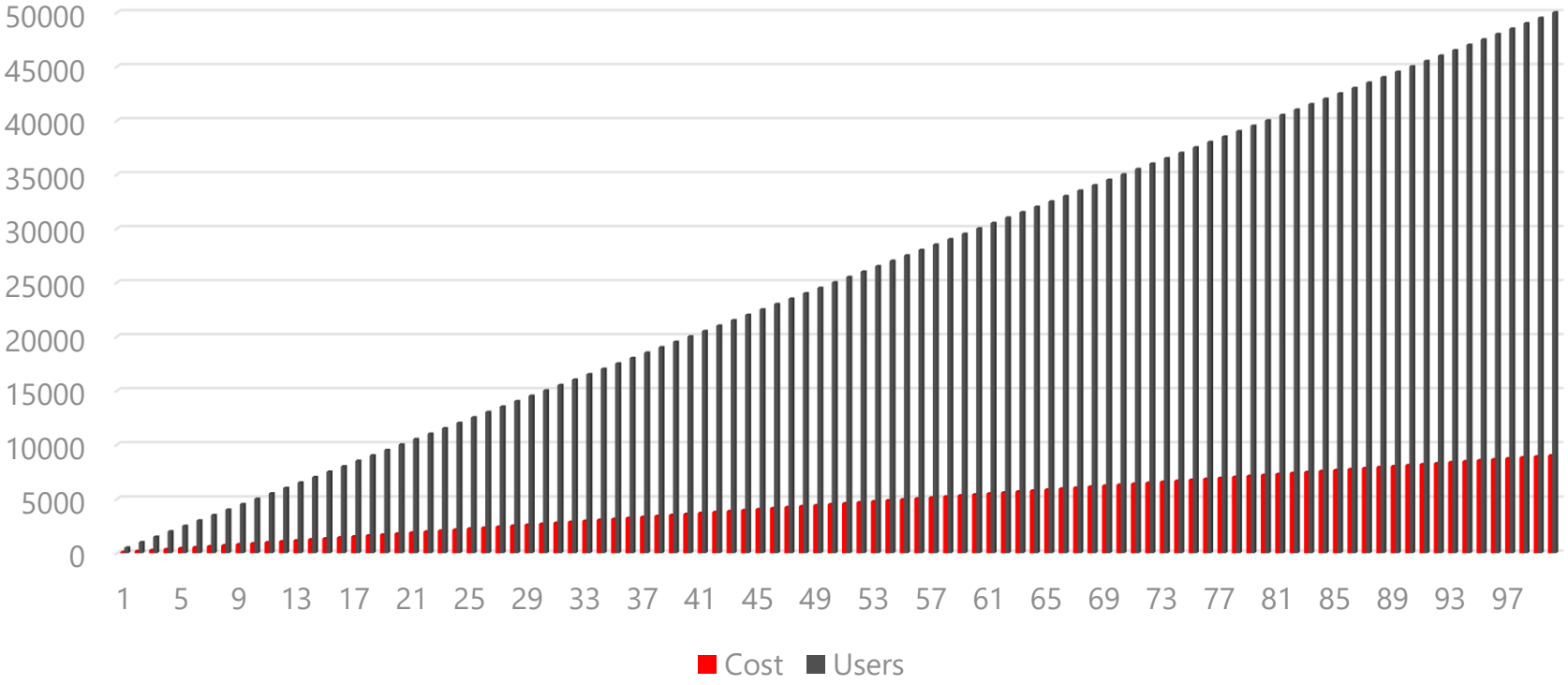
"Scale out"



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 and 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”.

Anti-Practices (Don’ts)

Microsoft

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.

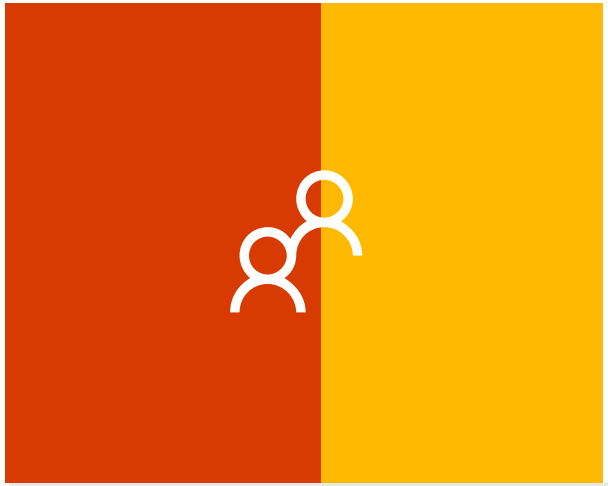
Don’ts: Subscription Management

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.

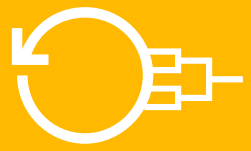
Customers should be aware that ASM services do not have Role Based access in place, so can’t be granularly controlled,

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.

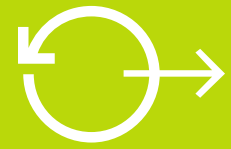
Customer should load test their solution ahead of every major release. This will act as a guideline for any performance defects.



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

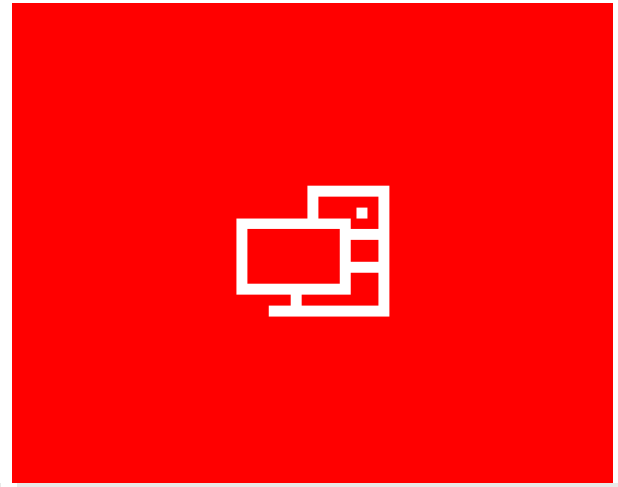
Some customers over-provisioned resources, even though such sizing was not required.

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

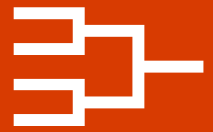
Some customers believed that the Azure platform automatically handled scaling across all Azure services, without any configuration. This is not true.

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).

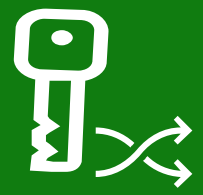
Customers should consider their SLAs (in terms of service requirements), and understand how caching can improve performance compared to adding additional complexity and cache lifecycle headaches.

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: 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.

Microsoft

Thanks for coming today.

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