Microsoft Azure - Intro

Milan Pekardy

pekardy@dcs.uni-pannon.hu

https://github.com/pekmil/CloudProgrammingAzure

There is no cloud
it's just someone else's computer

Cloud development patterns

- „Pattern”: recommended way to do things
  - Automate everything
  - Source control
  - Continuous integration and delivery
  - Web development best practicies
  - Single sign-on
  - Data storage options
  - Data partitioning strategies
  - Unstructured blob storage
  - Design to survive failures
  - Monitoring and telemetry
  - Transient fault handling
  - Distributed caching
  - Queue-centric work pattern

Automate everything

- Automating development tasks
- Manual processes are slow, error-prone
- Automating as many tasks as you can helps set up a fast, reliable and agile workflow
- In the cloud environment with automate tasks you can set up:
  - Whole test environments
  - Web servers
  - Virtual Machines
  - Databases
  - Blob storages
  - Message queues
  - Etc.

Automate everything

- Azure can support automate tasks with the following tools:
    - Azure Management Portal: web based UI to monitor and manage all of the deployed resources
    - REST management API
    - Windows PowerShell scripts

```
# Create a new website
$website = New-AzureWebsite -Name $Name -Location $Location -Verbose
```

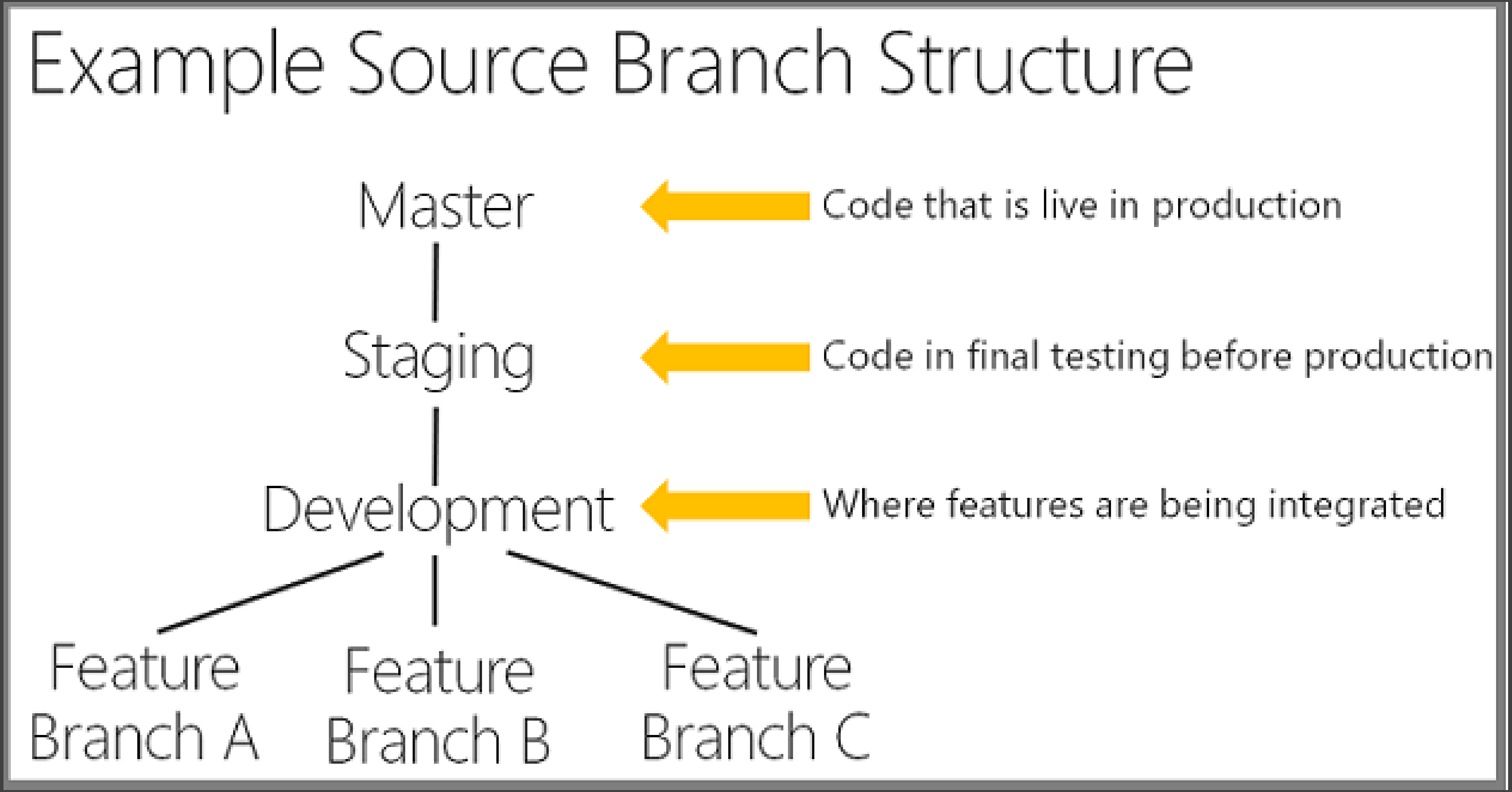
```
New-AzureSqlDatabaseServer -AdministratorLogin $UserName -AdministratorLoginPassword $Password -Location $Location
```

```
# Use the database context to create app database
New-AzureSqlDatabase -DatabaseName $AppDatabaseName -Context $context -Verbose
```

Source control

- Source control is essential for all cloud development projects
- Best practices:
  - Treat automation scripts as source code and version them together with your application code (all of the scripts need to be in sync with your application source code),
  - Never check in secrets (sensitive data such as credentials) into a source code repository (you can store app settings and connection strings in the web app's configuration in Azure, you can use Azure Key Vault),
  - Set up source branches to enable the DevOps workflow
- You can use e.g. Git in Visual Studio and Visual Studio Online

Source control - Branches

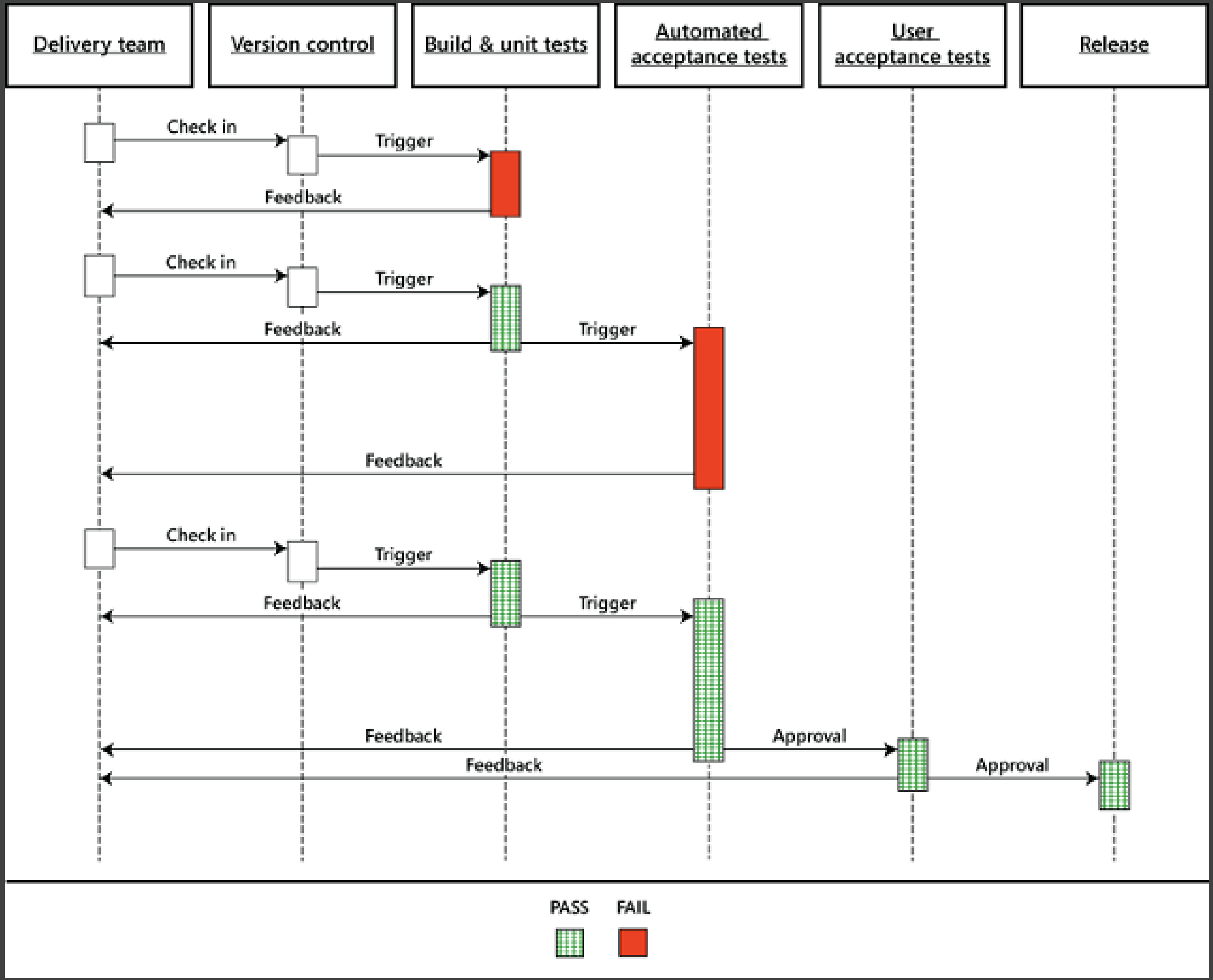


Example Source Branch Structure
Master
Code that is live in production
Staging
Code in final testing before production
Development
Where features are being integrated
Feature Branch A
Feature Branch B
Feature Branch C

Continuous Integration and Continuous Delivery

- Continuous Integration (CI): whenever a developer checks in code to the source repository, a build is automatically triggered
- Continuous Delivery (CD): after a build and automated unit tests are successful, you can automatically deploy the application to an environment where you can do more in-depth testing
- Do CD to your development and staging environments
- Production deployments may require manual review and approval process
- Azure Cloud environment enables cost-effective CI and CD: with every build you can spin up a test environment using automation scripts, run tests and when you're done just tear it down

Continuous Integration and Continuous Delivery



Web development best practices

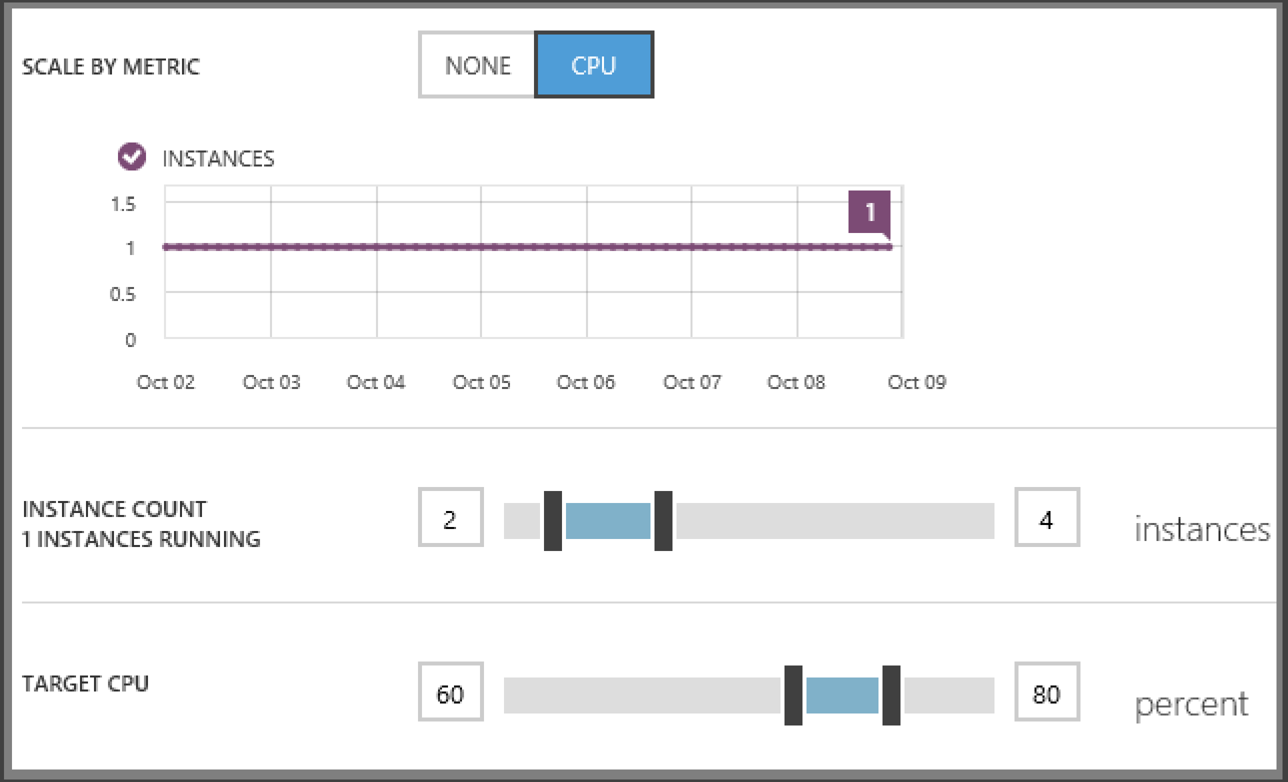
- Stateless web servers behind a smart load balancer
- Avoid session state (or if you can’t avoid it, use distributed cache rather than a database)
- Use a CDN to edge-cache static file assets (images, scripts)
- Use .NET 4.5’s async support to avoid blocking calls

- These practices work together to help you make optimal use of the highly flexible scaling offered by the cloud environment

Stateless web tier behind a smart load balancer

- *Stateless web tier* means you don't store any application data in the web server memory or file system. Keeping your web tier stateless enables you to both provide a better customer experience and save money:
    - If the web tier is stateless and it sits behind a load balancer, you can quickly respond to changes in application traffic by dynamically adding or removing servers,
    - A stateless web tier is architecturally much simpler to scale out the application,
    - Cloud servers, like on-premises servers, need to be patched and rebooted occasionally; and if the web tier is stateless, re-routing traffic when a server goes down temporarily won't cause errors or unexpected behavior.

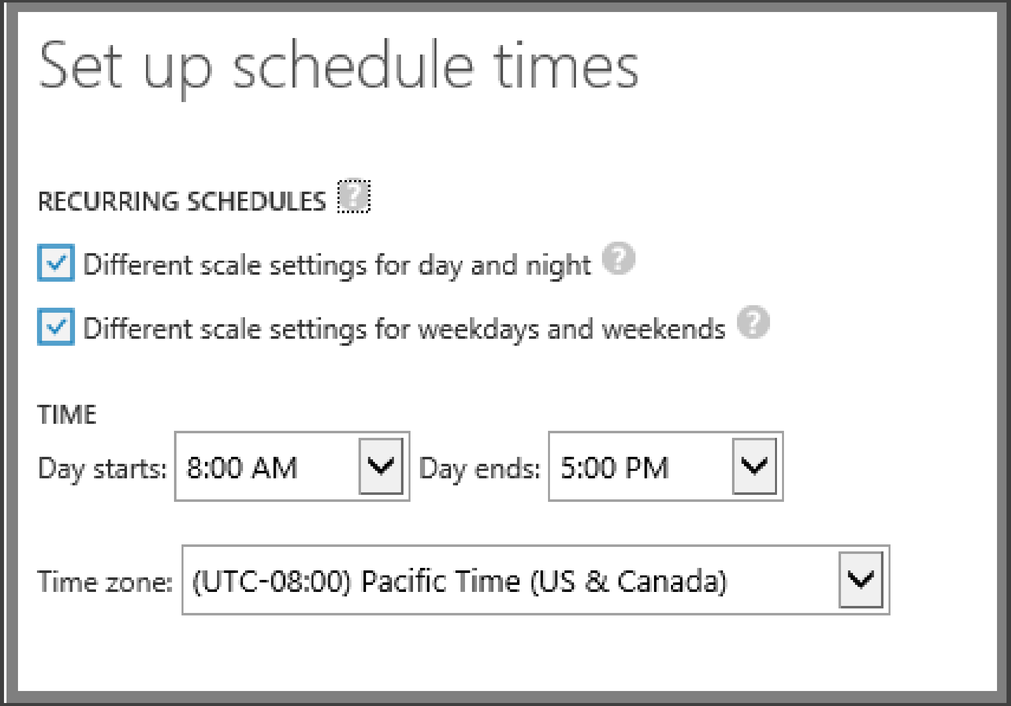
Stateless web tier behind a smart load balancer



SCALE BY METRIC: NONE [✓ CPU]

INSTANCE COUNT
1 INSTANCES RUNNING: 2 – 4 instances

TARGET CPU: 60 – 80 percent



Set up schedule times

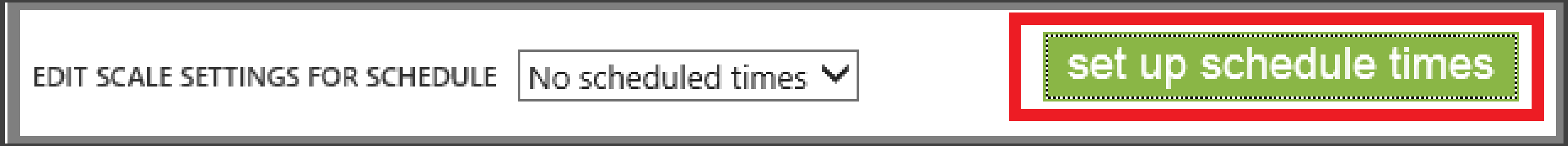
RECURRING SCHEDULES

- [✓] Different scale settings for day and night
- [✓] Different scale settings for weekdays and weekends

TIME

Day starts: 8:00 AM Day ends: 5:00 PM

Time zone: (UTC-08:00) Pacific Time (US & Canada)



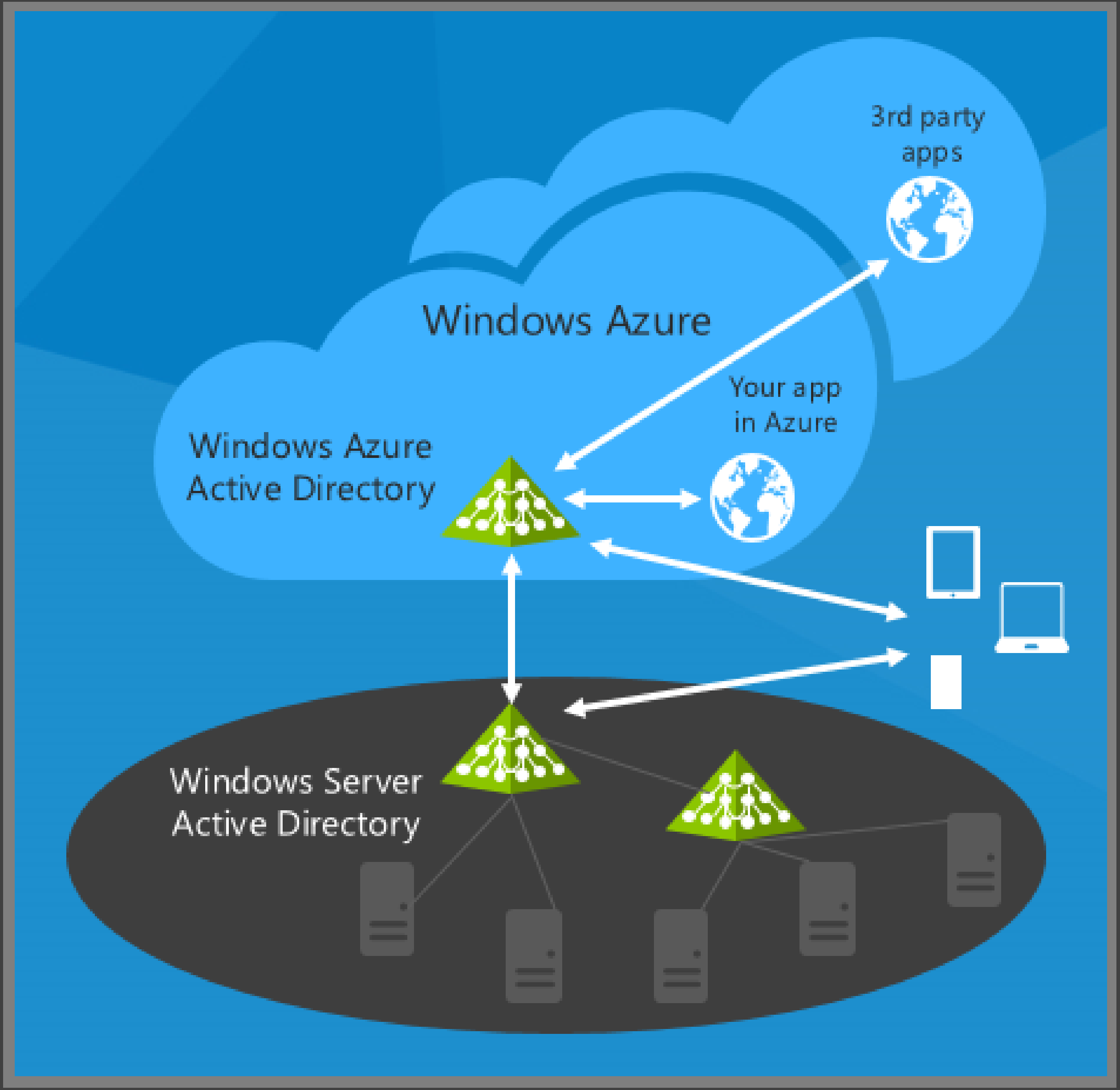
EDIT SCALE SETTINGS FOR SCHEDULE: No scheduled times

Single Sign-On

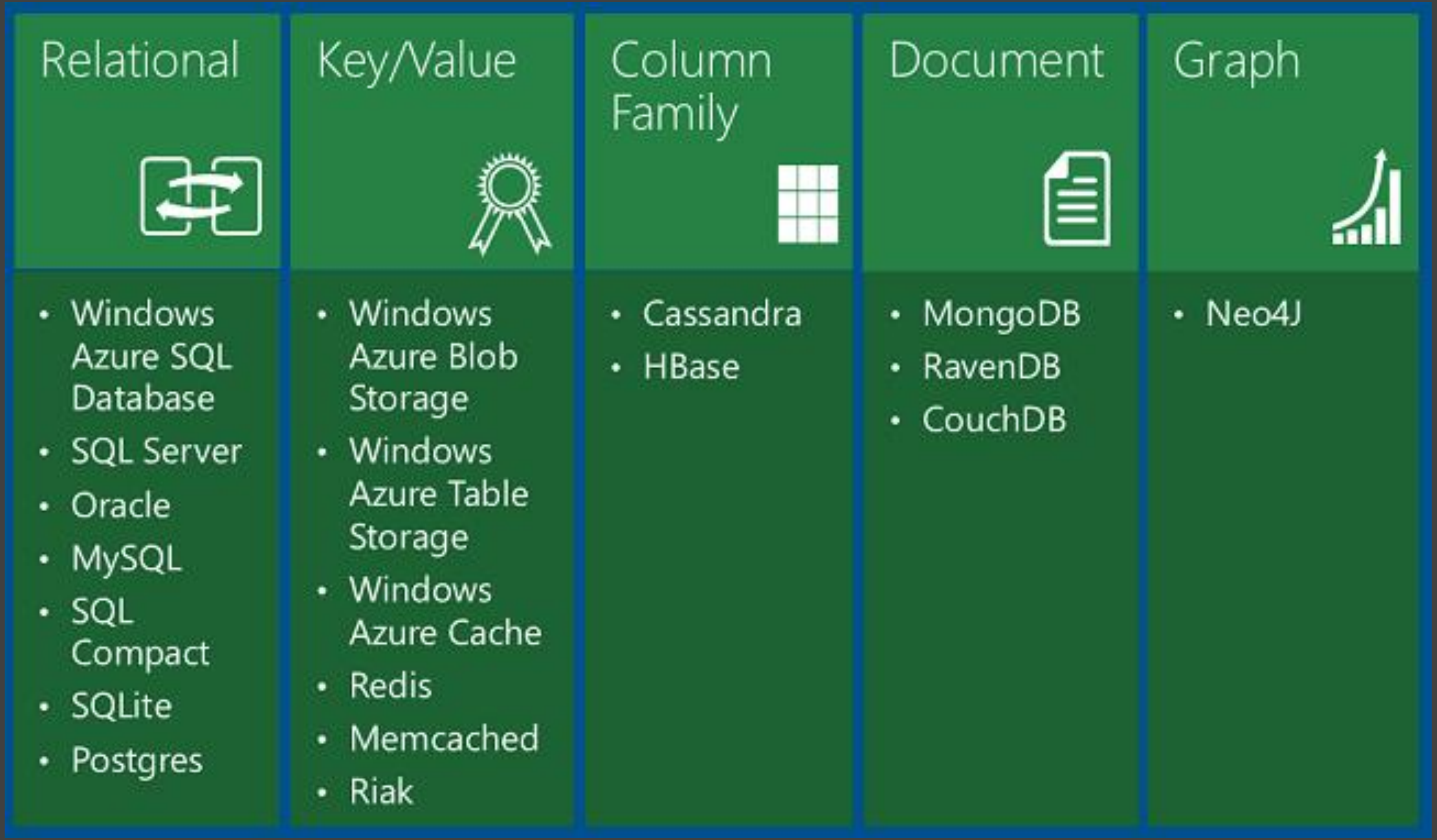
- Azure AD provides Active Directory in the cloud. Key features include the following:
  - It integrates with on-premises Active Directory.
  - It enables single sign-on with your apps.
  - It supports open standards such as SAML, WS-Fed, and OAuth 2.0.
  - It supports Enterprise Graph REST API.

Single Sign-On

- It can be entirely independent from your on-premises Active Directory, or you can integrate it with your on-premises AD
- All of your on-premises AD changes are automatically propagated to the cloud environment


3rd party apps
Windows Azure
Your app in Azure
Windows Azure Active Directory
Windows Server Active Directory

Data storage options



Relational	Key/Value	Column Family	Document	Graph
Windows Azure SQL Database	Windows Azure Blob Storage	Cassandra	MongoDB	Neo4J
SQL Server	Windows Azure Table Storage	HBase	RavenDB	
Oracle	Windows Azure Cache		CouchDB	
MySQL	Redis			
SQL Compact	Memcached			
SQLite	Riak			
Postgres				

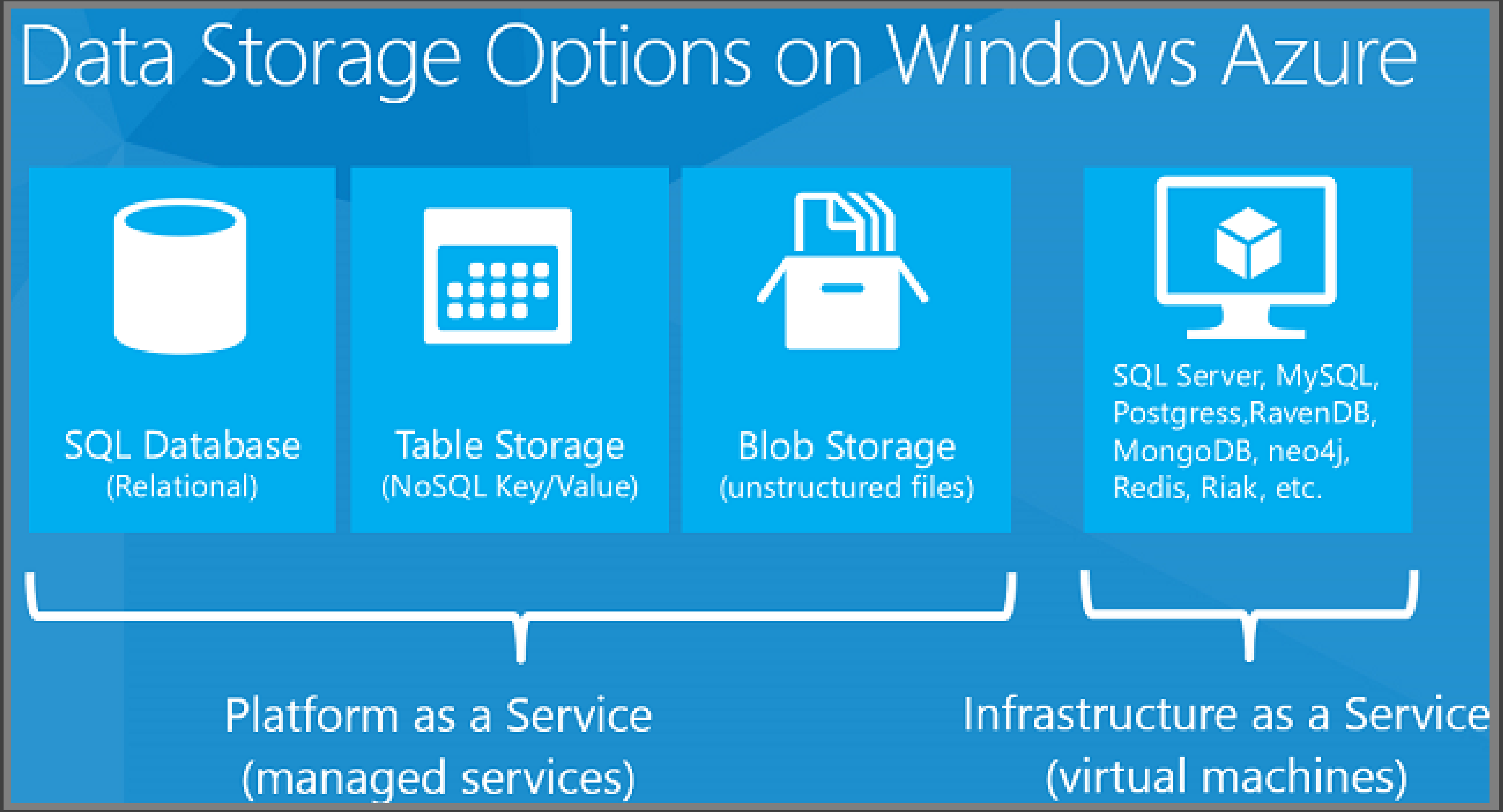
Data storage options

- Key/value databases store a single serialized object for each key value
- Azure Blob storage is a key/value database that functions like file storage in the cloud, with key values that correspond to folder and file names
- Azure Table storage is also a key/value database. Each value is called an entity (similar to a row, identified by a partition key and row key) and contains multiple properties (similar to columns, but not all entities in a table have to share the same columns)
- Document databases are key/value databases in which the values are documents. "Document" means a collection of named fields and values, any of which could be a child document
- Graph databases store information as a collection of objects and relationships. The purpose of a graph database is to enable an application to efficiently perform queries that traverse the network of objects and the relationships between them

Data storage options

- PaaS data solutions that Azure offers include:
  - Azure SQL Database (formerly known as SQL Azure). A cloud relational database based on SQL Server.
  - Azure Table storage. A key/value NoSQL database.
  - Azure Blob storage. File storage in the cloud.
- For IaaS, you can run anything you can load onto a VM, for example:
  - Relational databases such as SQL Server, Oracle, MySQL, SQL Compact, SQLite, or Postgres.
  - Key/value data stores such as Memcached, Redis, Cassandra, and Riak.
  - Column data stores such as HBase.
  - Document databases such as MongoDB, RavenDB, and CouchDB.
  - Graph databases such as Neo4j.

Data storage options

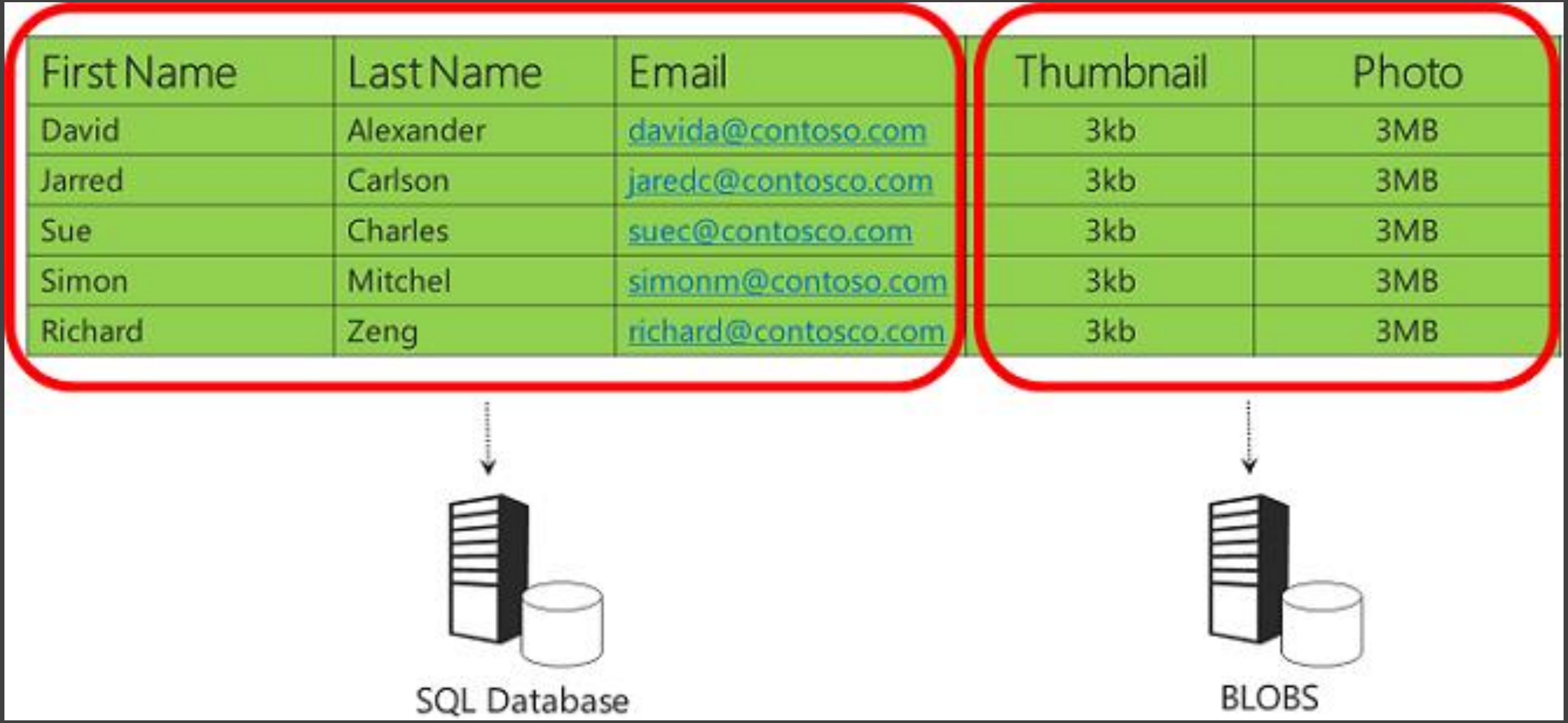


Data partitioning strategies

- In order to determine whether you need a partitioning strategy and what it should be, consider three questions about your data:
  - Volume – How much data will you ultimately store? A couple gigabytes? A couple hundred gigabytes? Terabytes? Petabytes?
  - Velocity – What is the rate at which your data will grow? Is it an internal app that isn't generating a lot of data? An external app that customers will be uploading images and videos into?
  - Variety – What type of data will you store? Relational, images, key-value pairs, social graphs?
- There are basically three approaches to partitioning:
  - Vertical partitioning
  - Horizontal partitioning
  - Hybrid partitioning

Data partitioning strategies

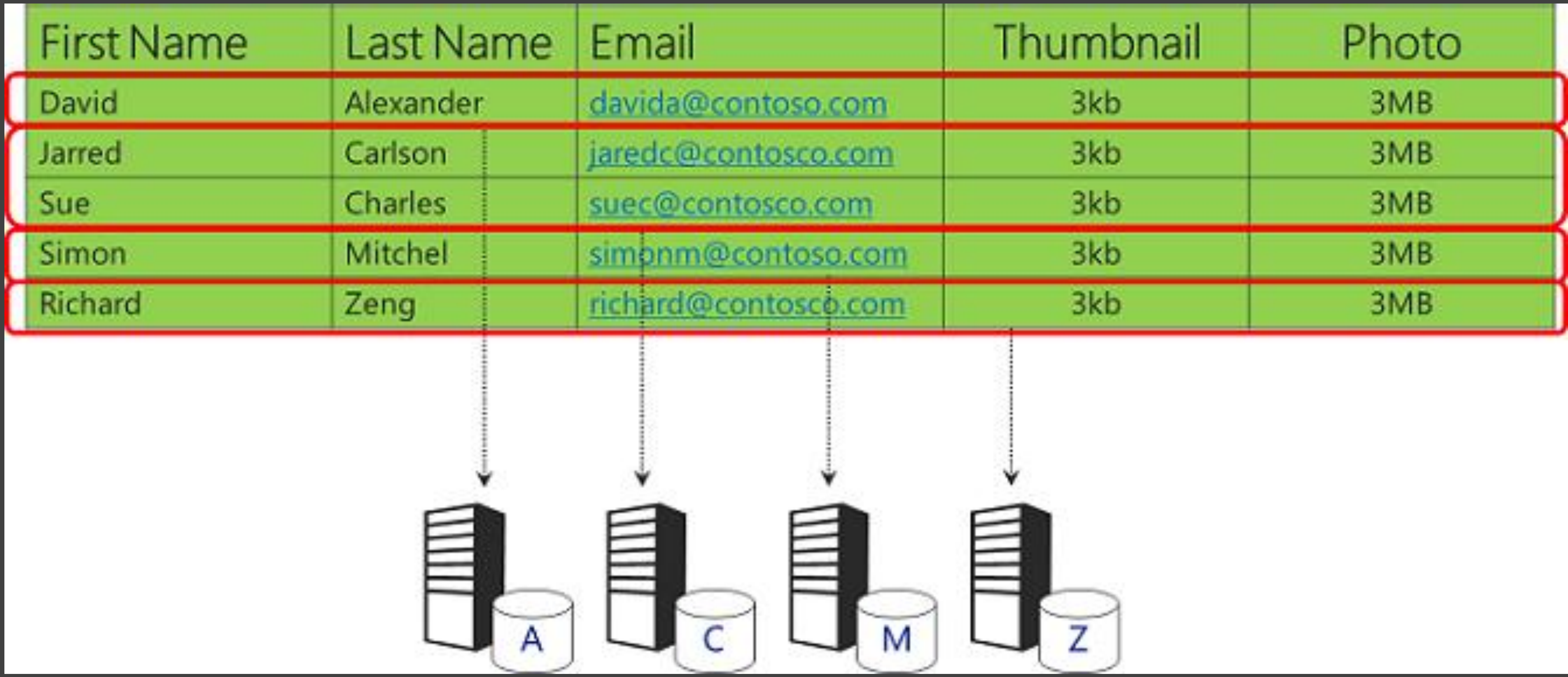
- Vertical partitioning
  - Vertical portioning is like splitting up a table by columns: one set of columns goes into one data store, and another set of columns goes into a different data store



First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB

Data partitioning strategies

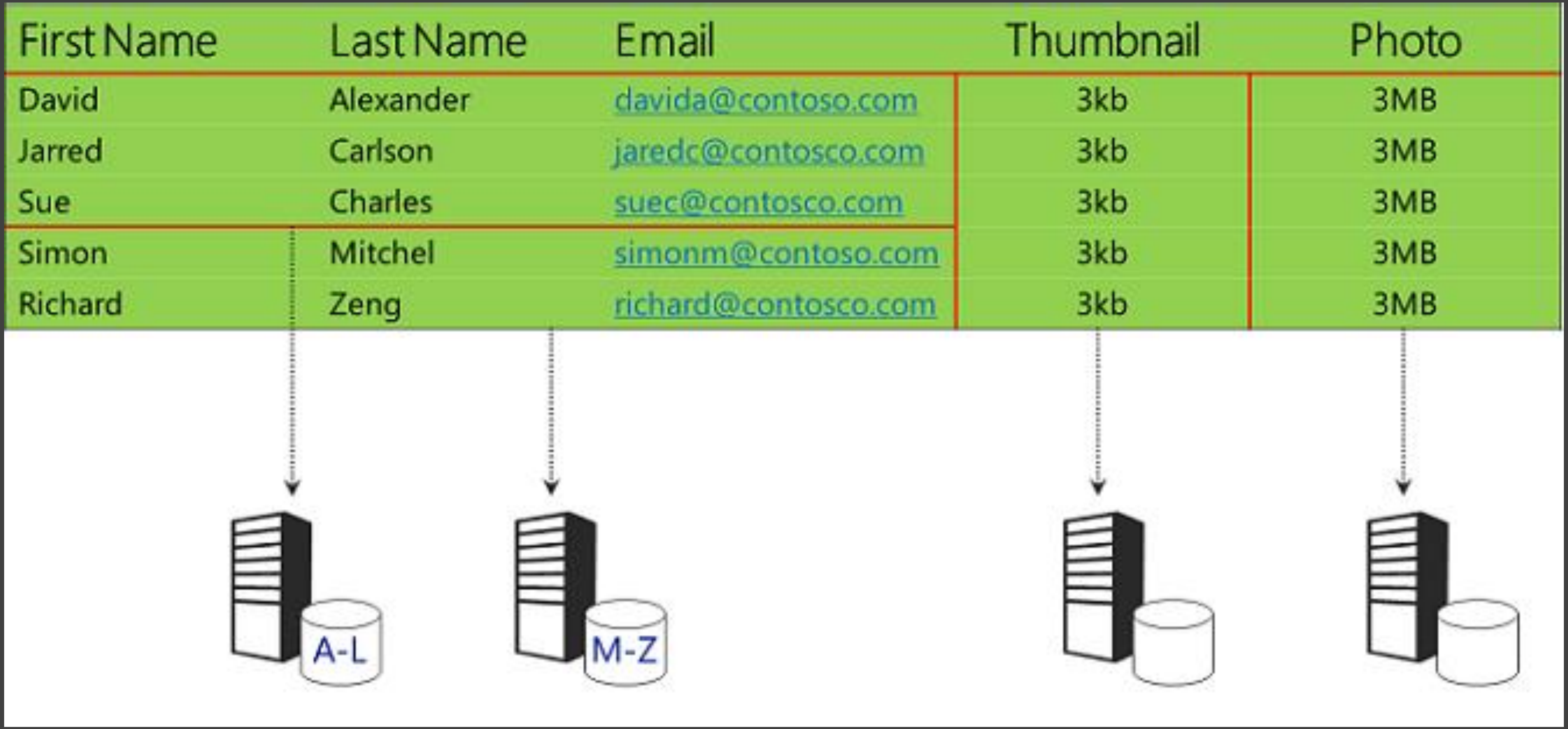
- Horizontal partitioning (sharding)
  - Horizontal portioning is like splitting up a table by rows: one set of rows goes into one data store, and another set of rows goes into a different data store
  - You want to be very careful about your sharding scheme to make sure that data is evenly distributed in order to avoid hot spots



First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB

Data partitioning strategies

- Hybrid partitioning
  - You can combine vertical and horizontal partitioning



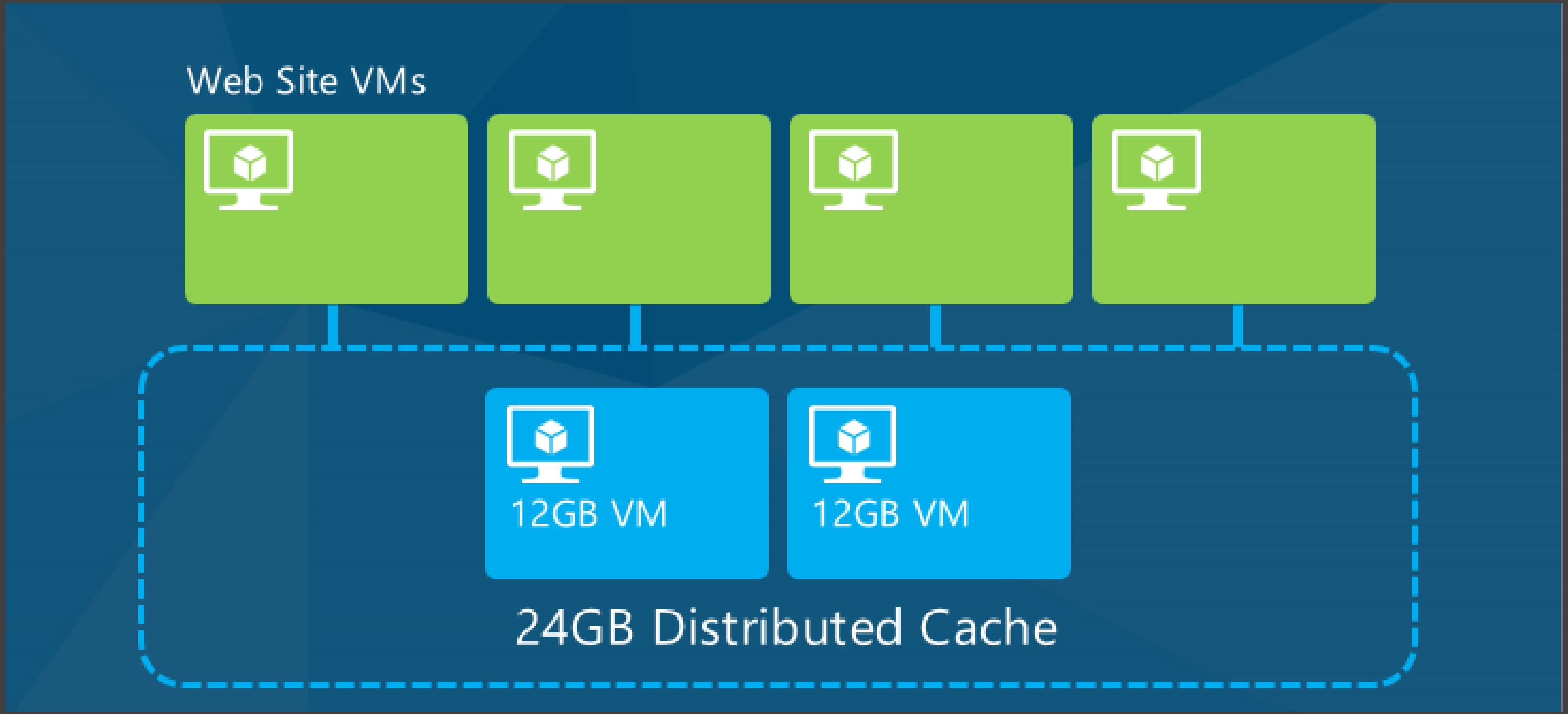
First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB

Unstructured Blob Storage

- The Azure Storage Blob service provides a way to store files in the cloud. The Blob service has a number of advantages over storing files in a local network file system:
  - It's highly scalable. A single Storage account can store hundreds of terabytes, and you can have multiple Storage accounts. Some of the biggest Azure customers store hundreds of petabytes.
  - It's durable. Every file you store in the Blob service is automatically backed up.
  - It provides high availability. The SLA for Storage promises 99.9% or 99.99% uptime, depending on which geo-redundancy option you choose.
  - It's a platform-as-a-service (PaaS) feature of Azure, which means you just store and retrieve files, paying only for the actual amount of storage you use, and Azure automatically takes care of setting up and managing all of the VMs and disk drives required for the service.
  - You can access the Blob service by using a REST API or by using a programming language API. SDKs are available for .NET, Java, Ruby, and others.
  - When you store a file in the Blob service, you can easily make it publicly available over the Internet.
  - You can secure files in the Blob service so they can accessed only by authorized users, or you can provide temporary access tokens that makes them available to someone only for a limited period of time.

Distributed caching

- A cache provides high throughput, low-latency access to commonly accessed application data, by storing the data in memory
- When the application scales by adding or removing servers, or when servers are replaced due to upgrades or faults, the cached data remains accessible to every server that runs the application



Distributed caching – Cacha population strategies

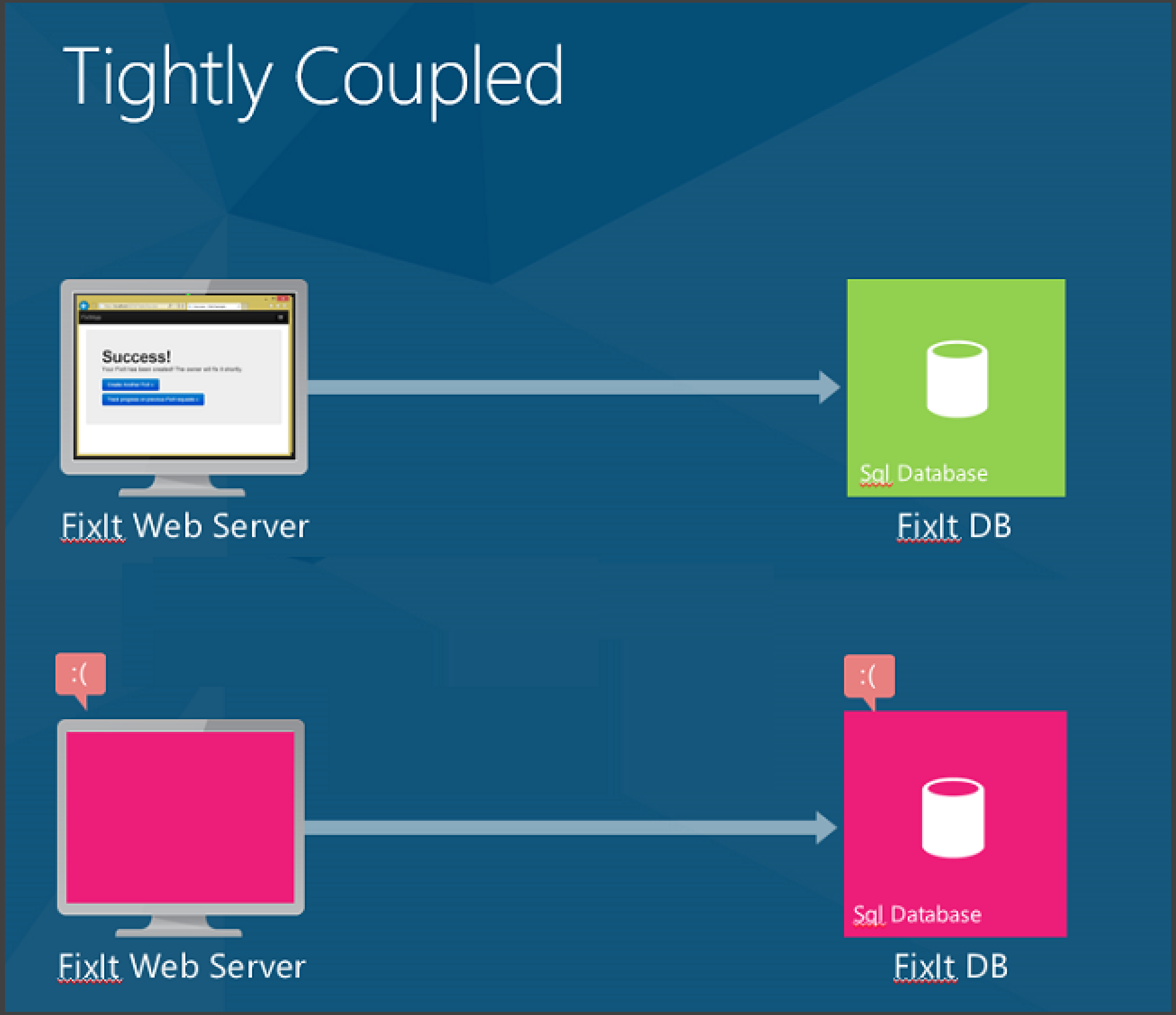
- On Demand / Cache Aside
  - The application tries to retrieve data from cache, and when the cache doesn't have the data (a "miss"), the application stores the data in the cache so that it will be available the next time. The next time the application tries to get the same data, it finds what it's looking for in the cache (a "hit"). To prevent fetching cached data that has changed on the database, you invalidate the cache when making changes to the data store.
- Background Data Push
  - Background services push data into the cache on a regular schedule, and the app always pulls from the cache. This approach works great with high latency data sources that don't require you always return the latest data.
- Circuit Breaker
  - The application normally communicates directly with the persistent data store, but when the persistent data store has availability problems, the application retrieves data from cache. Data may have been put in cache using either the cache aside or background data push strategy. This is a fault handling strategy rather than a performance enhancing strategy.

Queue-Centric Work Pattern

- This pattern enables loose coupling between a web tier and a backend service,
- When the application gets a request, it puts a work item onto a queue and immediately returns the response. Then a separate backend process pulls work items from the queue and does the work
- The queue-centric work pattern is useful for:
  - Work that is time consuming (high latency).
  - Work that requires an external service that might not always be available.
  - Work that is resource-intensive (high CPU).
  - Work that would benefit from rate leveling (subject to sudden load bursts).

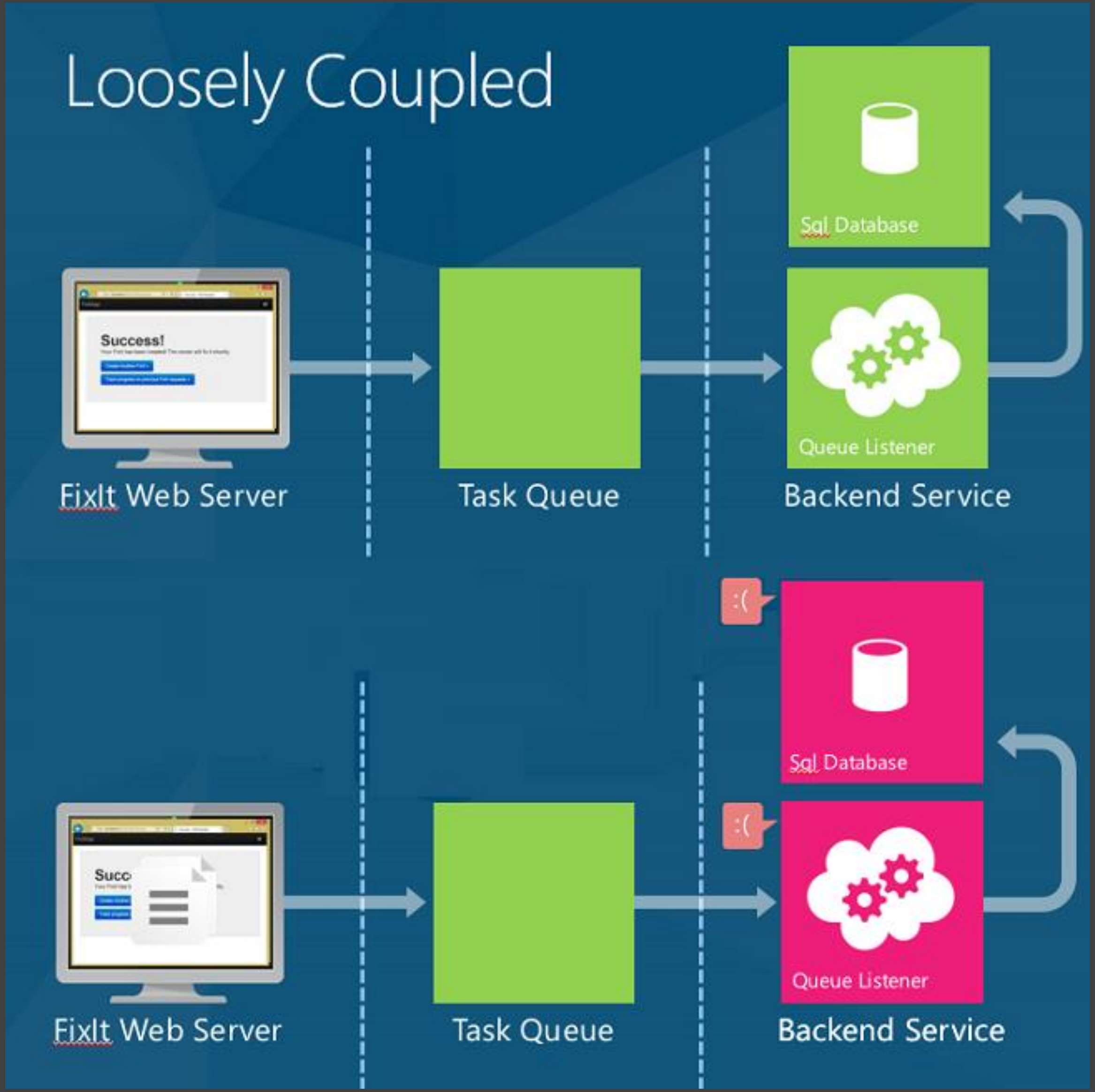
Queue-Centric Work Pattern

- The web front-end is tightly coupled with the SQL Database back-end.
- If the SQL database service is unavailable, the user gets an error.
- If retries don't work (that is, the failure is more than transient), the only thing you can do is show an error and ask the user to try again later



Queue-Centric Work Pattern

- Using queues, when a user submits a task, the app writes a message to the queue.
- As soon as the message is written to the queue, the app returns and immediately shows a success message to the user.
- If any of the backend services – such as the SQL database or the queue listener -- go offline, users can still submit new tasks.
- The messages will just queue up until the backend services are available again. At that point, the backend services will catch up on the backlog.


Loosely Coupled
Success!
FixIt Web Server
Task Queue
Sql Database
Queue Listener
Backend Service
:(
Sql Database
Queue Listener
:(
FixIt Web Server
Task Queue
Backend Service