



Serverless Azure Functions: Lessons Learned

Joe Koletar
RBA



INTRODUCTIONS



Joe Koletar
Managing Principal
Engineer



DESIGNED TO DELIVER

RBA is a digital and technology consultancy headquartered in Wayzata, MN with roots in strategy, design, and technology.

We have designed our services and engagement types to align to the unique needs of our clients. Whether that is adding capacity to a team, helping to deliver a solution on new technologies, or provide support after launch, we are designed to deliver.

What we do

 Custom & Cloud Applications	 Digital Experience	 Cloud Infrastructure	 Modern Workplace
 Data and Analytics	 B2C & B2B Commerce	 DevOps	 CRM

How we engage

 Project Services	 Co-Delivery & Staffing	 Managed Services
----------------------	----------------------------	----------------------

BACKGROUND

What we are doing





Project Background

- Re-platform eCommerce for Caleres
- Supports 15 different brands
- Azure based project using Sitecore XP CMS and Sitecore Commerce
- Started on envisioning last summer with plan to launch first sites soon



16

Unique brands

26%

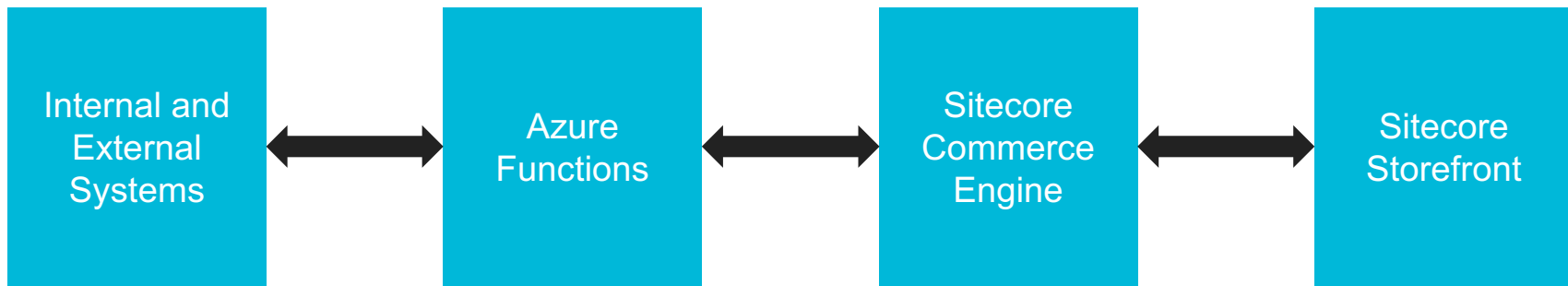
Growth YoY for Cyber Monday

200M

Page views per month



Project Diagram



PIM
DAM
Fulfillment
Partners

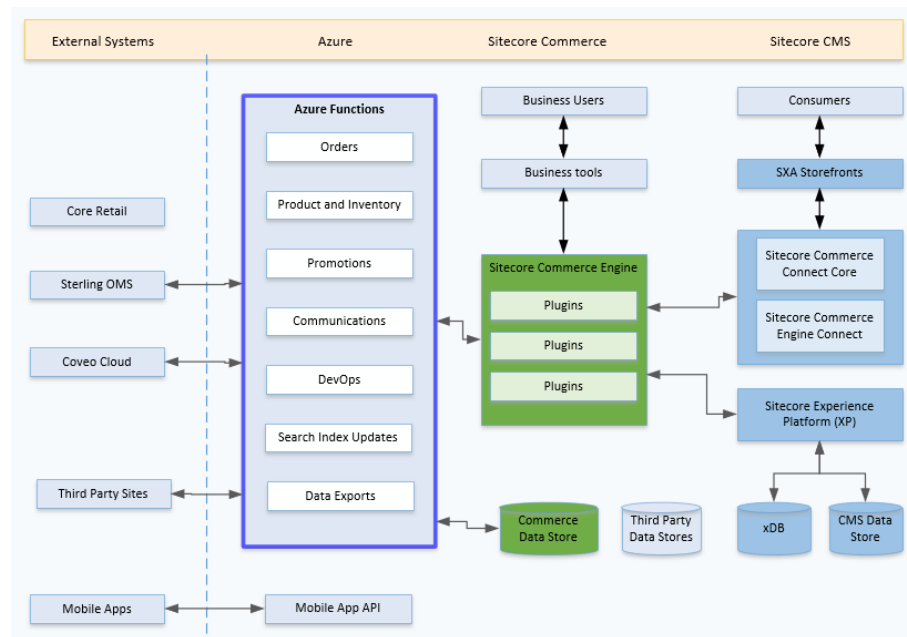
Customers
Products
Carts
Orders

MVC Pages

Project Intent



- Functions are replacing a set of Windows services
- Intent was to create highly scalable microservices



Azure Functions

- Functions support small pieces of code in the cloud
- Can be developed in JavaScript, C#, F#, Python, and others
- Can be developed in the portal or IDE
- Capable of being serverless



Serverless

- Abstraction of servers
- Event-driven scalability
- Pay per use



Azure Function Triggers

Triggers define how a function is invoked

- Available Triggers
 - HTTP
 - Timer
 - GitHub
 - Generic webhook
 - CosmosDB
- Blob Storage
- Azure Storage Queue
- EventHub
- ServiceBus Queue
- ServiceBus Topic

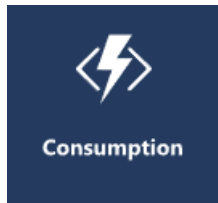


SETUP OPTIONS

Configuration options for
your Azure Functions



Hosting Options



Consumption

Serverless
Scales automatically
Pay for what you use



App Service

Runs on dedicated VM
Pay for the App Service
Can scale with App Service



Premium

Serverless
Min/Max scaling
Instance sizing
Perpetually warm instances
Vnet Connectivity



Function App vs Function

Function App

Can contain more than one function

Deployable/scalable unit

Function

Is uniquely addressable

Performs specific action



Home > Function App > [redacted]-feeds-orchestration

[redacted]-feeds-orchestration

Function Apps

Search: "[redacted]-feeds-orchestr..."

Dropdown: Ecommerce Prod Subscription

Function Apps

- prod-feeds... >>
- Functions (Read Only)
 - BazaarvoiceProductExport...
 - ClearanceBadgesDurableA...
 - DataFeedOrchestration
 - DataFeedOrchestrator
 - FeednomicsExportDurable...
 - GoogleExportDurableActi...
 - InventoryReportRefreshD...
 - LoadStagingTablesDurabl...
 - ModcoExportDurableActi...
 - ...

Overview | Platform features

Stop | Swap | Restart | Get publish profile | Reset publish profile | Download app content | Delete

Status	Subscription	Resource group	URL
Running	[redacted] Subscription	[redacted] prod sub	[redacted]-feeds-orchestration.azurewebsites.net
Availability	Subscription ID	Location	App Service plan / pricing tier
Available	[redacted]	Central US	[redacted] az-premium-g10 (ElasticPremium)

Configured features

- Function app settings
- Configuration
- Deployment options configured with VSTSRM
- Application Insights



Versions

Version 1.x

Uses .NET Framework

Only supports Azure and Windows

Version 2.x

Runs on .NET Core 2

Support macOS and Linux

All Functions in Function App must share same language

More bindings

Support for dependency injection

SECURITY

Locked up tight





Connectivity

- Project need to support PCI compliance
- Required ability to communicate over Vnet and Hybrid connection
- Azure Functions talking to internal systems
- Use of Vnet required premium plan support
- Use of Hybrid connection required app service plan

App Service Plan	Premium Plan
Vnet – No	Vnet – Yes
Hybrid Connection – Yes	Hybrid Connection – No

DATA FLOW

Getting from here to there

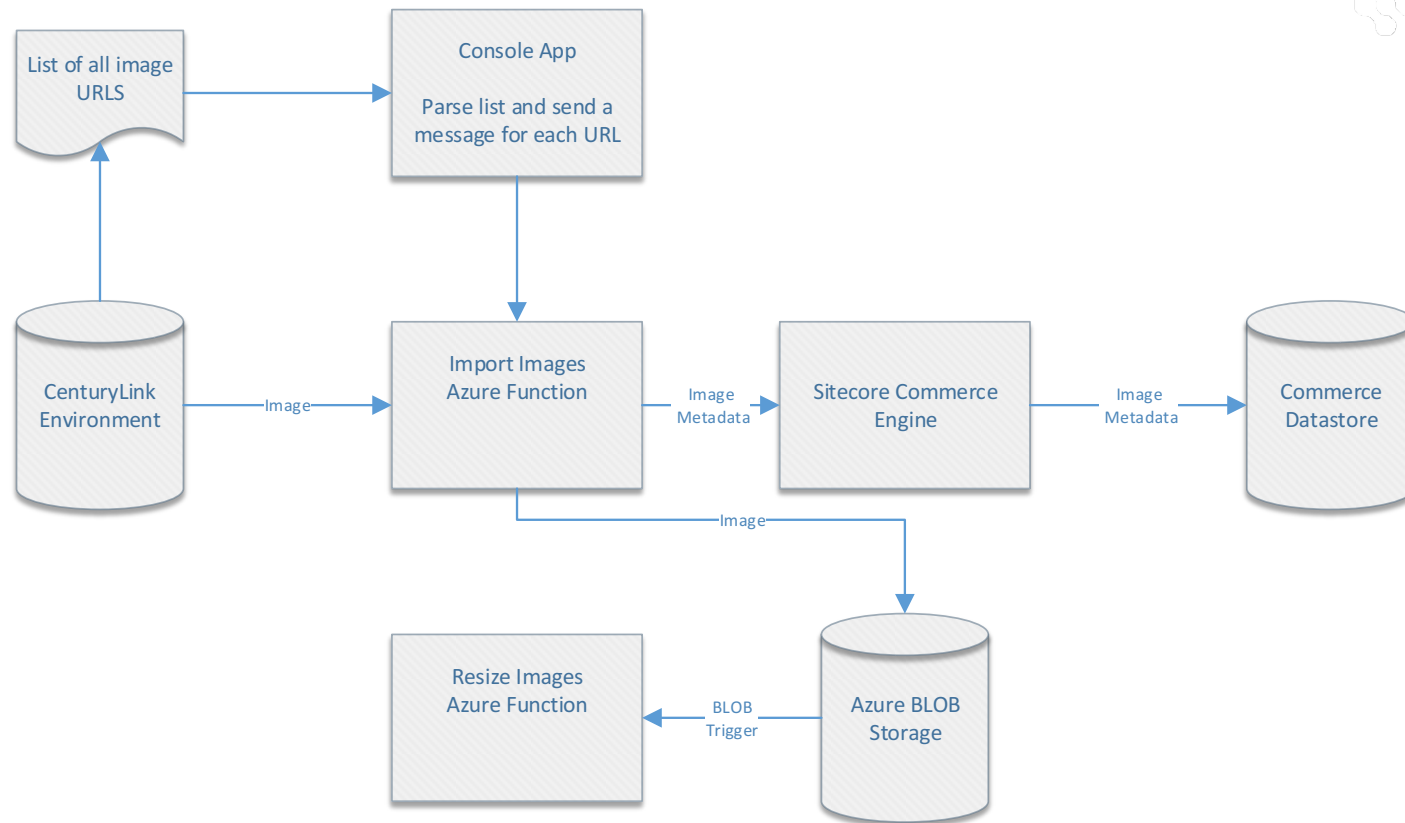


Azure Service Bus

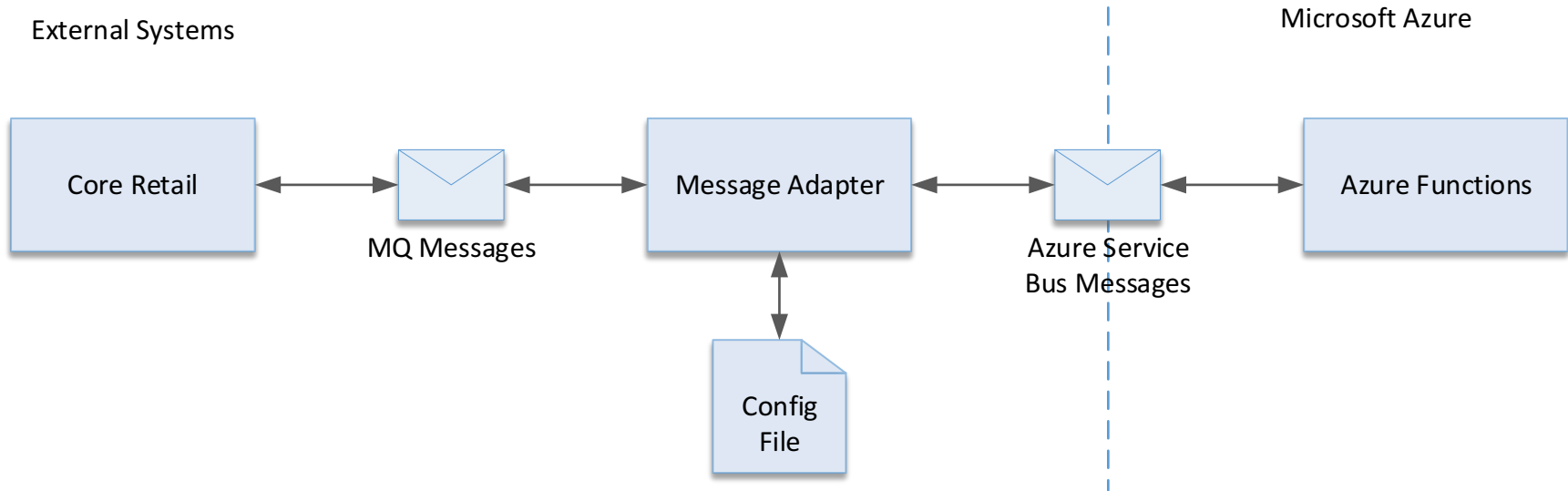
- Using Azure Service Bus Messages as trigger on most functions
- Advantages of Messages
 - Guaranteed delivery
 - Automatic retries
 - Dead letter queue for failures
 - Ability to delay delivery
 - Automatic scaling of functions



Messages For Flow



Connecting to Internal Systems



SCALABILITY

Handling the load



Service Bus Scalability

- Each function processes one message
- Need to handle thousands of messages arriving within seconds
- Weekly catalog refreshes send hundreds of thousands of messages in under an hour
- Treating functions as a microservice



Function Scalability

- Functions in a function app share resources
- Functions are inherently instance based
- Scalability requires shared resources
 - Database connections
 - HTTP connections
 - Azure storage connections





Static clients

- Do not create a new client with every function invocation.
- Do create a single, static client that every function invocation can use.
- Consider creating a single, static client in a shared helper class if different functions use the same service.

```
// Create a single, static HttpClient
private static HttpClient httpClient = new HttpClient();

public static async Task Run(string input)
{
    var response = await httpClient.GetAsync("https://example.com");
    // Rest of function
}
```

Dependency Injection



```
public class HttpTrigger
{
    private readonly IMyService _service;
    private readonly HttpClient _client;

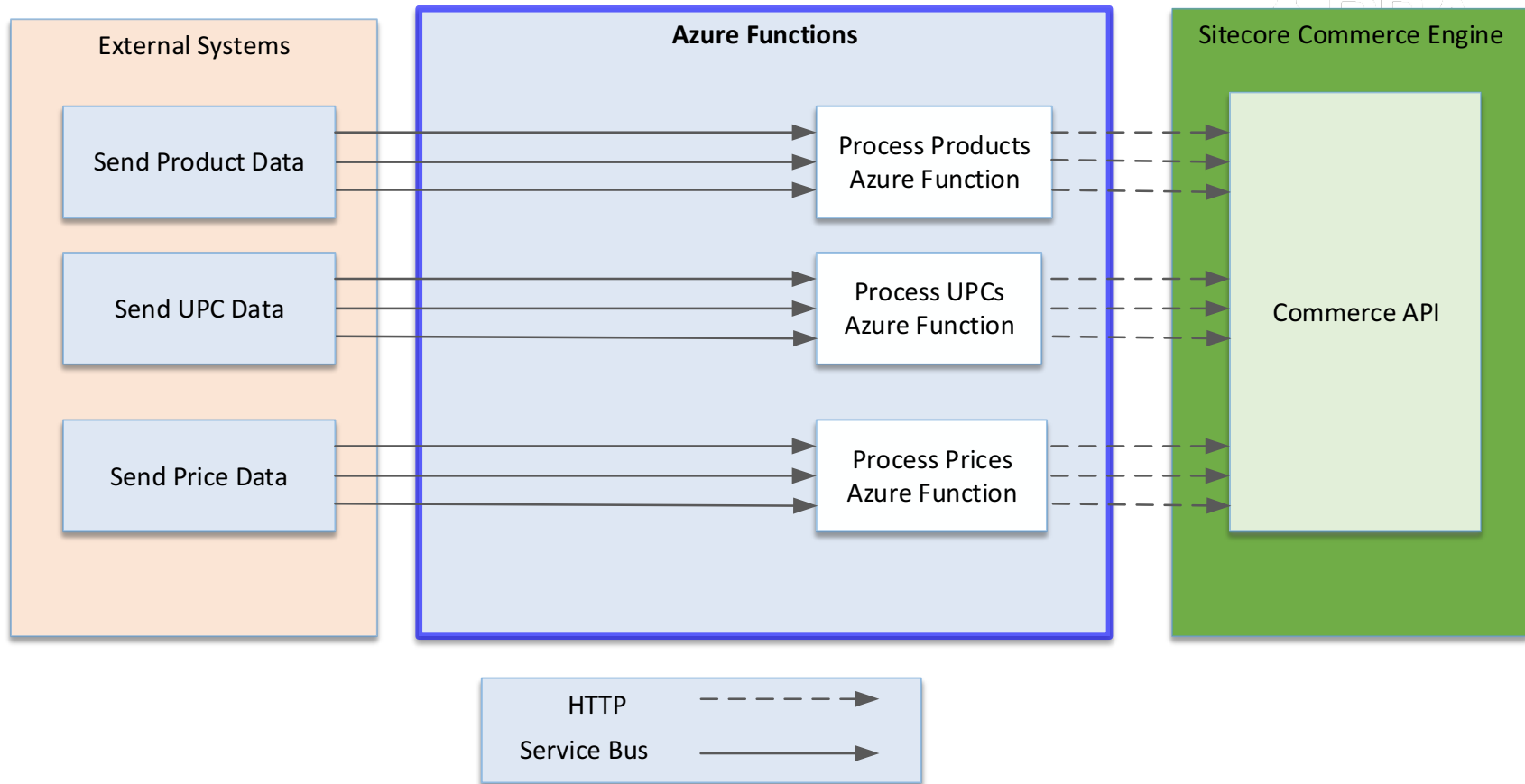
    public HttpTrigger(IMyService service, IHttpConnectionFactory httpClientFactory)
    {
        _service = service;
        _client = httpClientFactory.CreateClient();
    }

    [FunctionName("GetPosts")]
    public async Task<IActionResult> Get(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = "posts")] HttpRequest req,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a request.");
        var res = await _client.GetAsync("https://microsoft.com");
        await _service.AddResponse(res);

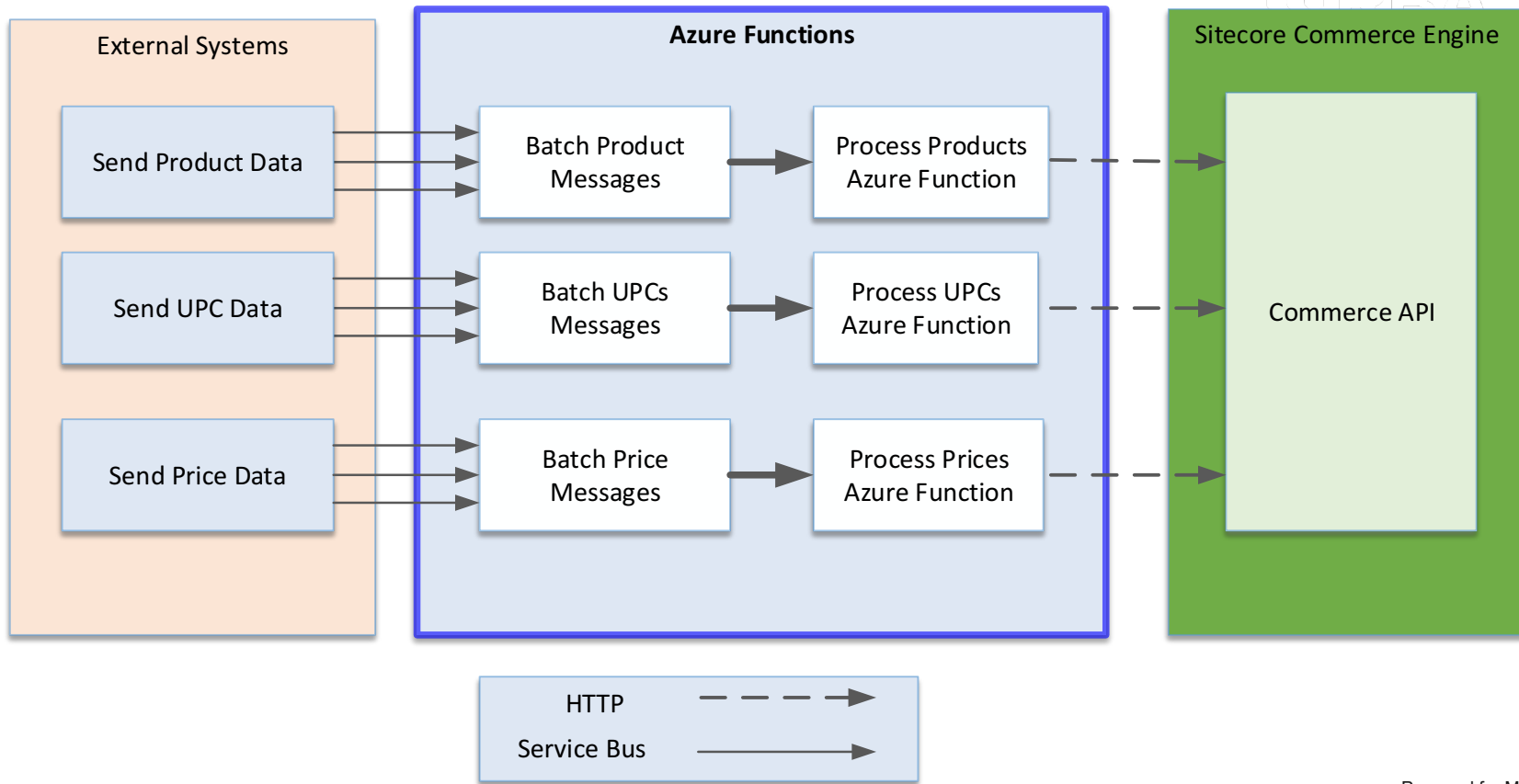
        return new OkResult();
    }
}
```



Scalability Issues



Scalability Improved



DURABLE FUNCTIONS

It's getting complicated

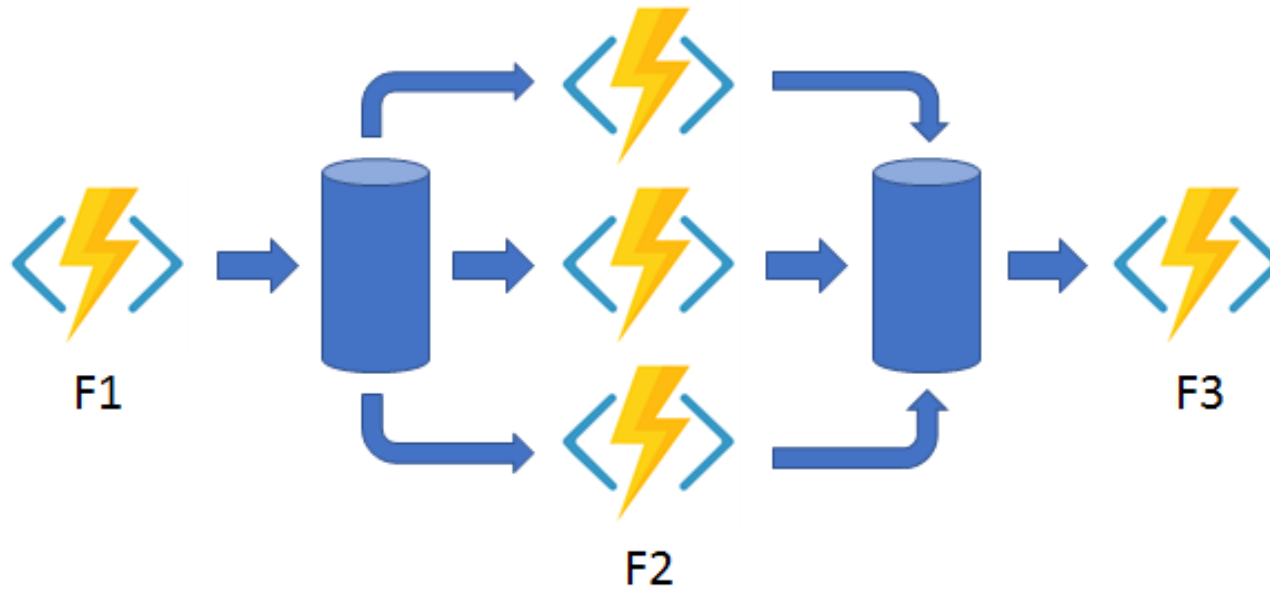


Durable Functions

- Support stateful functions using an orchestrator
- Allows coordination between serverless applications
- The orchestrator manages state, checkpoints, and restarts for you.



Fan Out / Fan In



Reporting Use Case



- Reporting database periodically built to support multiple activities
 - External exports
 - Product activation checks
 - Product classification operations
- Activities are run on different schedules
 - Every 30 minutes
 - Once an hour
 - Twice a day
 - Once a day
- Needed to coordinate the refreshing of the reporting database with tolerance for stale data

Reporting Implementation

- Bundled Functions and orchestrator in a single Function App
- Orchestration fires every 10 minutes and kicks off other activities
- Each function determines if it needs to be executed and if it needs updated data



A screenshot of the Azure Portal interface for a Function App named 'feeds-orchestration'. The breadcrumb navigation shows 'Home > Function App > feeds-orchestration'. The page title is 'feeds-orchestration' with 'Function Apps' below it. On the left, a search bar contains 'feeds-orchestration' and a dropdown menu shows 'Ecommerce Prod Subscription'. Below this is a 'Function Apps' section with a tree view. The tree view is expanded to show 'prod-feeds...' with a lightning bolt icon and a refresh icon. Underneath, there is a 'Functions (Read Only)' section with a list of functions: 'BazaarvoiceProductExport...', 'ClearanceBadgesDurableA...', 'DataFeedOrchestration', 'DataFeedOrchestrator', 'FeednomicsExportDurable...', 'GoogleExportDurableActi...', 'InventoryReportRefreshD...', 'LoadStagingTablesDurabl...', 'ModcoExportDurableActi...', and 'NewProductDurableActi...'. On the right, the 'Overview' tab is active. It shows control buttons for 'Stop', 'Swap', and 'Res'. Below these, the 'Status' is 'Running' with a green checkmark, and 'Availability' is 'Available' with a green checkmark. Underneath, there is a 'Configured features' section with icons and labels for 'Function app settings', 'Configuration', 'Deployment options configu', and 'Application Insights'. At the bottom right, there is a small text 'Prepared for MSDN 2019'.

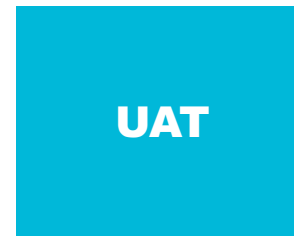
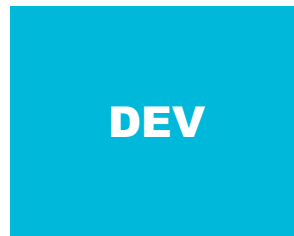
DEPLOYMENT & CONFIGURATION

Putting things in their place



Deployment and Configuration

- Using Azure DevOps for Build and Release
- Using Git for source control
- All functions are housed in a single Integration VS solution



Deployment – Initial Approach



- Single build for the whole VS Integration solution
- Release had a step per function
 - Eventually 20+ steps
- Issues identified:
 - Required replicating the build per environment
 - Each time a new function was developed, it had to be added to the build in each environment
 - It was a lot to keep up with
 - Hard to track status of release as functions soon number more than 20

Deployment – Refined Approach

- Every function has its own build and release
- Tokenized replacement per environment in release
- Creation of ARM templates are part of development process
- Allows devs to update app settings via PR/code check in
- Functions deploy very fast



Home > Functions > upcs-external

Tasks Variables Triggers Options Retention History Save & queue

Get sources

Sitcore_Commerce master

Agent job 1

Run on agent



Set Assembly Manifest Data

Assembly Info



Use NuGet 4.3.0

NuGet tool installer



NuGet restore

NuGet



Build solution Azure/Functions/Products/Caleres.I...

MSBuild



VsTest - testAssemblies

Visual Studio Test



Copy ***azuredeploy*.json

Copy files



Publish Artifact: \$(Build.DefinitionName)

Publish build artifacts



All pipelines > upcs-external

Pipeline Tasks Variables Retention Options History

ecom-dev-pr

Deployment process

Run on agent

Run on agent



Replace tokens in ***azuredeploy*.json

Replace Tokens



\$(Release.EnvironmentName)-\$(Release.DefinitionN...

Azure resource group deployment



Deploy Azure App Service

Azure App Service deploy

Configuration

- Each function has a variety of configuration parameters
 - Trigger settings
 - Database connections
 - Service Bus connection strings
 - Service Bus Queue names
 - Certificates thumbprints
- Many of the parameters vary per environment
- Often need to store arrays of settings per store
- Considerable volume of settings to manage

Configuration Initial Approach

- Stored parameters in Azure App Settings
- Awkward to manage
- No good way to handle nested/arrays of settings
- Could not be easily checked into source control
- Required coordination between developers and DevOps
- Multiple environments required different settings



Configuration – Revised Approach

- Use ARM templates for trigger settings
- Store settings in a centralized file
 - JSON format supports complex settings
 - File is copied to every function in the deployment
 - Environment based settings are transformed between environments
- Easier to manage
- Good balance between source control access and configurability



Tools



Service Bus Explorer



Service Bus Explorer 3.0.4

File Edit Actions View Help

Microsoft Azure

Service Bus Namespace

Queues

- brands (0, 1)
- expeditedshipping (0, 0)
- map_prices (0, 0)
- order_ids (0, 0)
- orders (0, 0)
- prices (0, 5610)
- product_images (0, 1)
- product_images_import (0, 0)
- products (0, 1924)
- search_updater_reindex (0, 0)
- search_updater_remove (0, 0)
- store_inventory (0, 0)
- upcs_external (112883, 0)**
- upcs_internal (0, 48319)
- web_inventory (0, 0)

Topics

Event Hubs

Notification Hubs

Relays

View Queue: upcs_external

Description | Authorization Rules | Metrics

Path

Relative URI: upcs_external

Auto Delete On Idle

Days: 106751 Hours: 2 Minutes: 48 Seconds: 5 Millisecs: 477

Duplicate Detection History Time Window

Days: 0 Hours: 0 Minutes: 10 Seconds: 0 Millisecs: 0

Default Message Time To Live

Days: 14 Hours: 0 Minutes: 0 Seconds: 0 Millisecs: 0

Queue Properties

Max Queue Size In GB: 5 GB

Max Delivery Count: 10

User Description:

Forward To:

Forward Dead Lettered Messages To:

Lock Duration

Days: 0 Hours: 0 Minutes: 0 Seconds: 30 Millisecs: 0

Queue Settings

- Enable Batched Operations
- Enable Dead Lettering On Message Expiration
- Enable Partitioning
- Enable Express
- Requires Duplicate Detection
- Requires Session

Queue Information

Name	Value
Status	Active
Is ReadOnly	False
Size In Bytes	30,141,817
Created At	11/14/2018 4:08:49
Accessed At	3/22/2019 1:54:16 PM
Updated At	11/14/2018 4:08:49
Active Message Count	112,883
DeadLetter Message Count	0
Scheduled Message Count	0
Transfer Message Count	0
Transfer DL Message Count	0
Message Count	112,883

Get Metrics Close Tabs Messages Deadletter Refresh Disable Delete Update

Service Bus Explorer



-
- Available at <https://github.com/paolosalvatori/ServiceBusExplorer>

Tip:

- Edit the ServiceBusExplorer.vshost.exe.config and add your connection string to the `<serviceBusNamespaces>` nodes

Microsoft Azure Storage Explorer



The screenshot shows the Microsoft Azure Storage Explorer application. The interface includes a top menu bar (File, Edit, View, Preview, Help), a toolbar with various actions like Upload, Download, Open, New Folder, Copy URL, Select All, Copy, Paste, Rename, Connect VM, Delete, View Share Snapshots, Directory Statistics, and Refresh. The Explorer pane on the left shows a tree view of storage resources, with 'pockoletarb2e9' selected under 'koletarinventorypoc'. The main pane displays the contents of the selected file share, showing four folders: 'ASP.NET', 'data', 'LogFiles', and 'site'. The bottom pane shows the 'Properties' tab for the selected file share, with the following details:

Property	Value
URL	https://koletarinventorypoc.file.core.w
Type	File Share
Last Modified	Wed, 28 Nov 2018 22:46:55 GMT
Quota	5120 GB
Usage	1 GB

The bottom right pane shows 'Activities' with two entries: 'Clear completed' and 'Clear successful'.

Microsoft Azure Storage Explorer

- Uses your Azure accounts
- Available at <https://azure.microsoft.com/en-us/features/storage-explorer/>



Build Azure Functions to be scalable



RBA



RBA

Microsoft Guidance

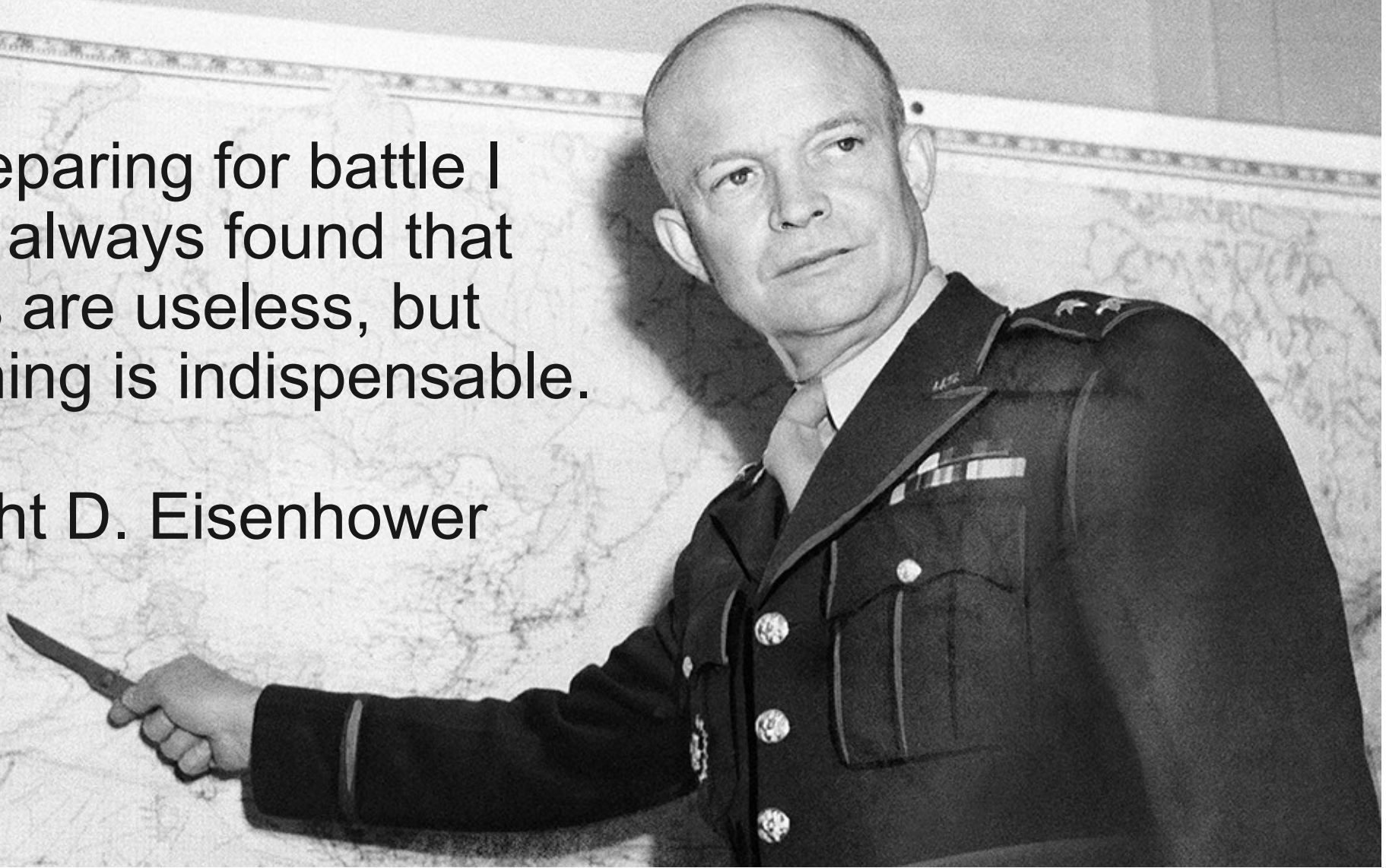
- <https://docs.microsoft.com/en-us/azure/azure-functions/functions-dotnet-dependency-injection>
- <https://docs.microsoft.com/en-gb/azure/azure-functions/manage-connections>

TAKEWAYS



In preparing for battle I
have always found that
plans are useless, but
planning is indispensable.

Dwight D. Eisenhower





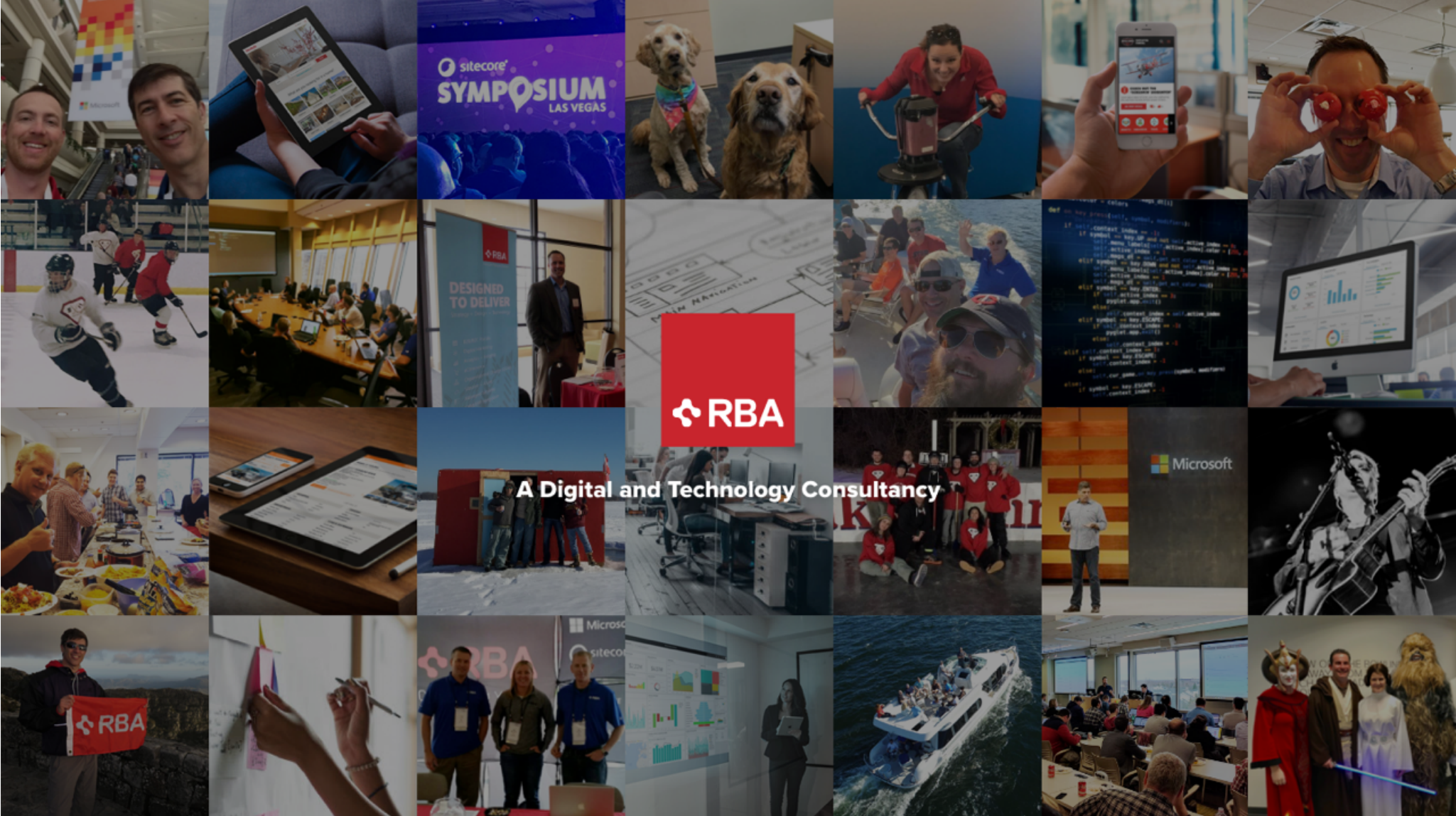
Lessons Learned



- Plan your approach, know your choices, but be flexible
- Know your trigger options and make sure your choices support how you plan to scale
- Functions scale rapidly, but you still need to consider being chunky over chatty to achieve scalability
- Durable functions support complex processing, but should be approached carefully as they are tricky to develop and debug
- Deploy Function Apps individually. Atomic DevOps processes are easier to manage.
- Configuration parameters can grow rapidly. Centralize what you can with plans to transform per environment.
- Determine your approach to monitoring the functions

QUESTIONS





A Digital and Technology Consultancy