BIZTALK360 SERVERLESS360 ATOMIC SCOPE

TECHNOLOGY PARTNER Microsoft

INTEGRATE 2019

PLATINUM SPONSORS

POWER OF INTEGRATION QUIBIQ

codit

GOLD SPONSORS

MIDDLEWAY DATA IN MOTION

REPLY CLUSTER

affinus

Richard Seroter

Microsoft MVP - Azure

Modernizing Integrations

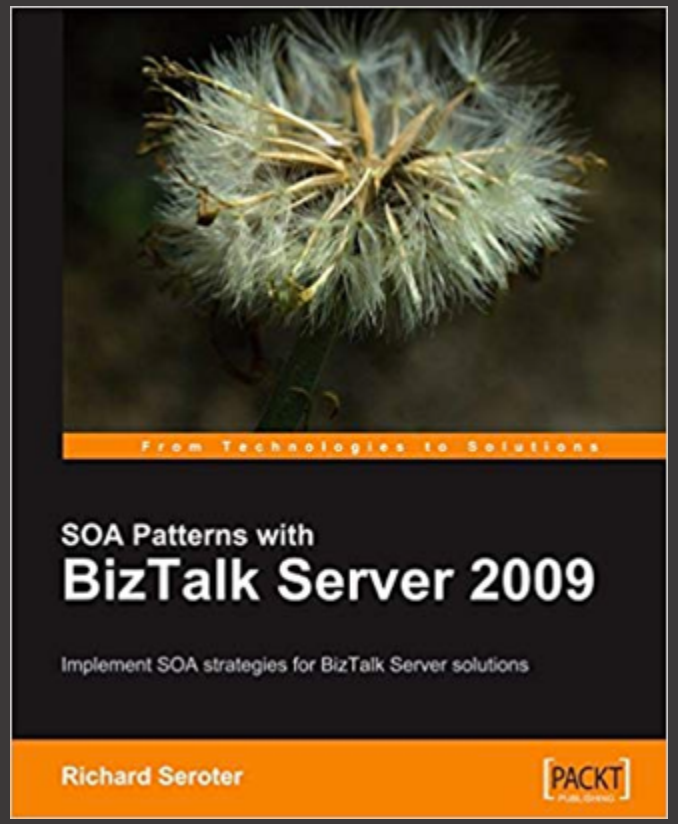
Integrate UK

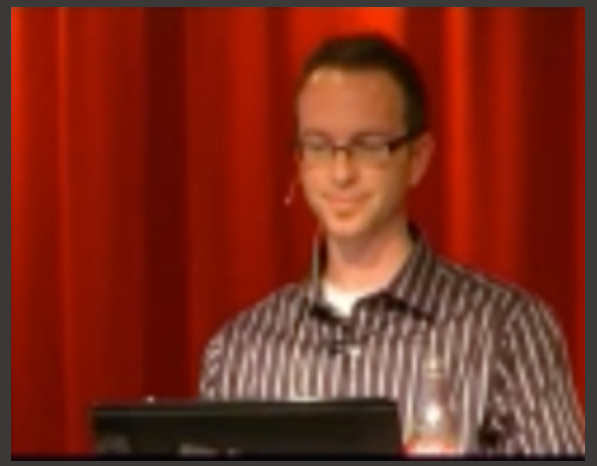
June 2019

modernizing integrations

Richard Seroter | @rseroter
Vice President, Pivotal

Let's go back to 2009.





Since 2009, lots of things have changed: new patterns, technologies, and expectations.


If you followed my advice in 2009, it's **time for an upgrade**!
@rseroter

#1 The “what” + “why” of modernization

#2 Considerations when modernizing integrations

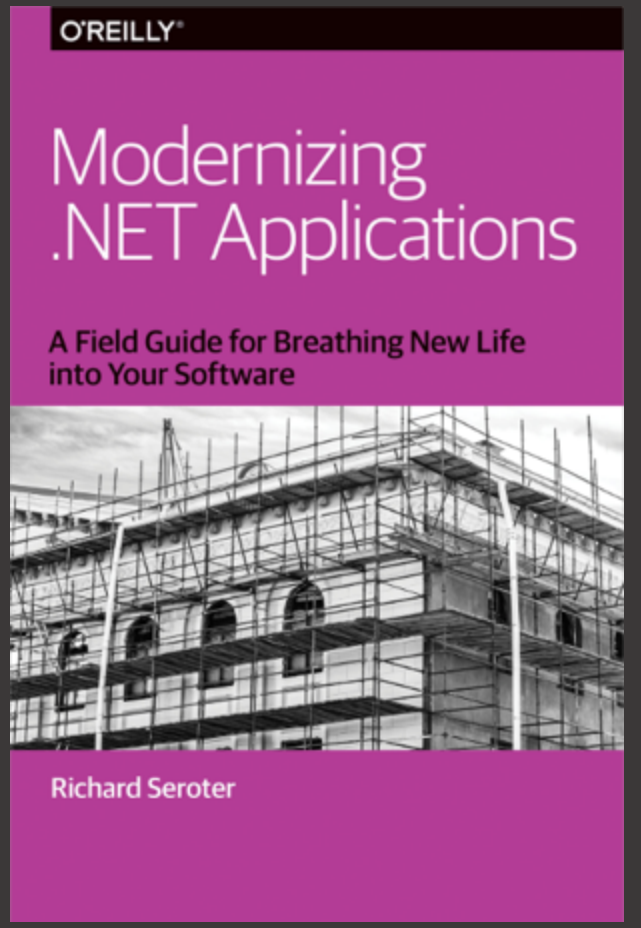
#3 Practices for modernization

#1 The “what” + “why” of modernization

#2 Considerations when modernizing integrations

#3 Practices for modernization

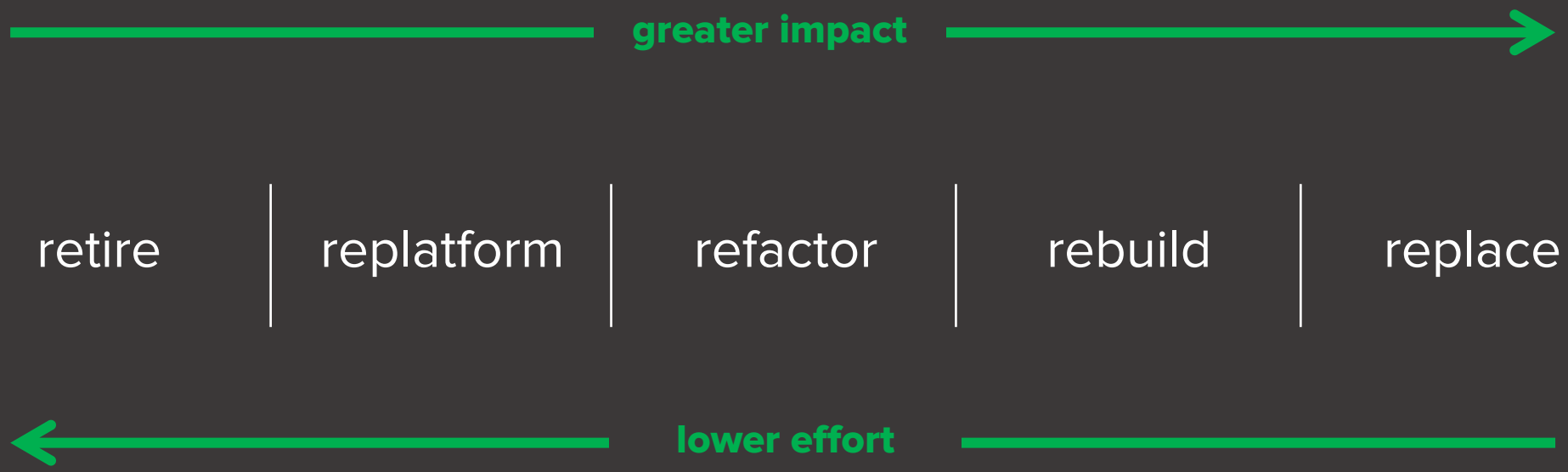
Last year I wrote a book.



How does Forrester define application modernization?

"The transformation of application assets to adapt and optimize them to migrate to, or more readily integrate with, more modern digital software and cloud architectures or to retire them outright."

Modernization is a spectrum.



What do you have in place right now?

BizTalk Server

Business Rules Engine **MSMQ**

Custom Functoids

Windows Communication Foundation

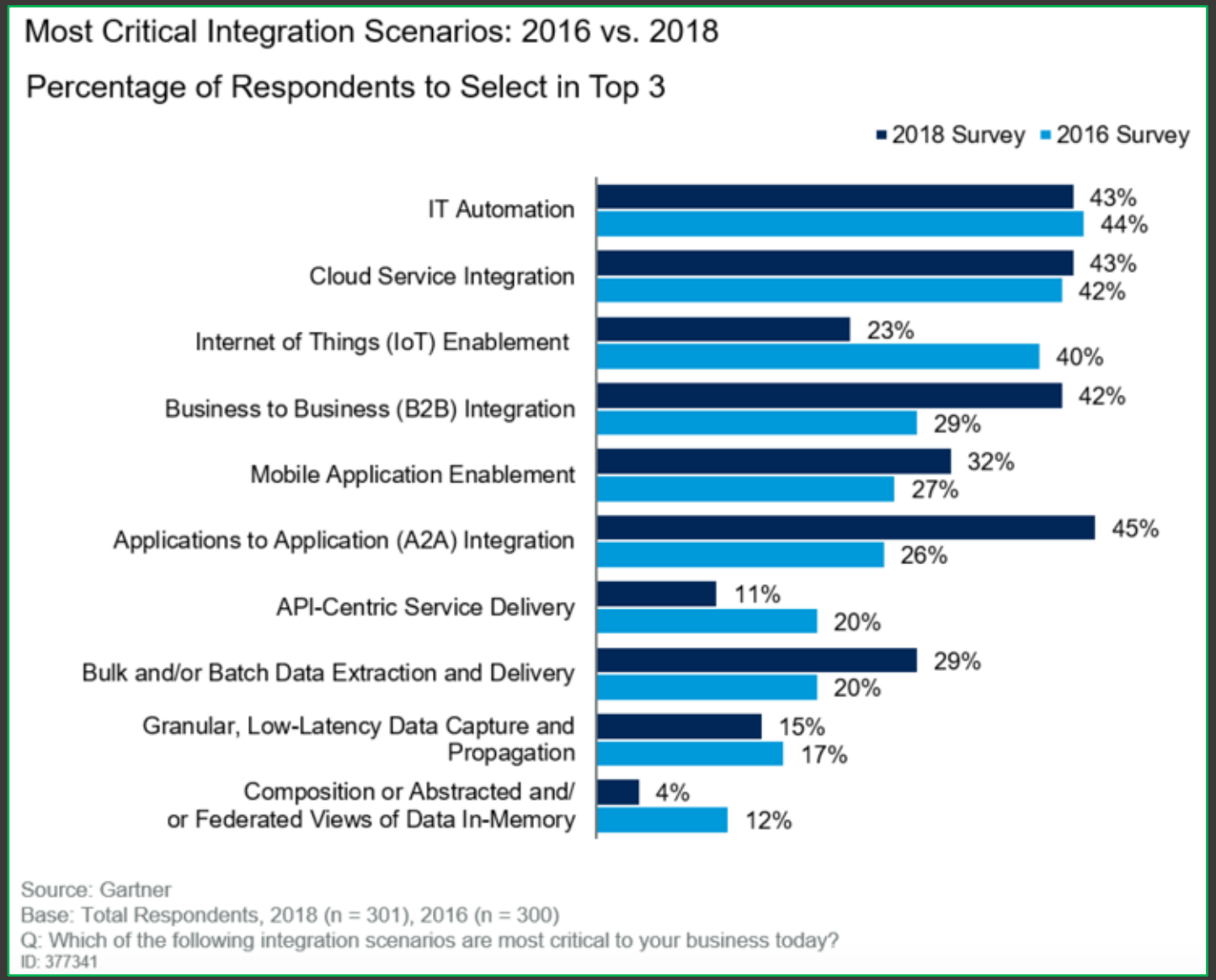
SSIS

Workflow Foundation

Windows Server AppFabric

Microsoft Azure Service Bus

What are you asked to create?



We want integrations that get delivered to production **faster**, have **newer** capabilities, **cost less** to operate, and are optimized for **maintainability**.

#1 The “what” + “why” of modernization

#2 Considerations when modernizing integrations

#3 Practices for modernization

Evaluate system maturity

Make different choices based on system and problem maturity

For experiments, use emerging tech or the simplest options

Mature problems or end-of-life systems should use commodity tech and an architecture focused on maintainability

“Unlearn” what you know

Don’t default to XML formats

Less centralized data storage and processing

Move biz logic into endpoints

Be integration enables vs gatekeepers

Reduce dependency on Windows

Introduce new components

Public cloud

Managed services

Functions

Event brokers

API gateways

Service meshes

Protocols (e.g. gRPC, RSocket)

Uncover new endpoints and users

SaaS and cloud-hosted systems

Custom and commercial API endpoints

Ad-hoc subscribers to data streams

Citizen integrators

Audit existing skills

Team skills will impact modernization approach

Identify skills gaps in API design, infrastructure automation, event stream processing

Assess the cost of missing skills and how/if to acquire

Upgrade interaction patterns

Introduce Event Thinking where it makes sense

Evolve from sense-and-respond (poll) to push triggers

Change how you access and interact with with production environments

Add automated delivery

On-demand environments for developers

Continuous integration pipelines for every integration

Automated deployment to production for all components

Choose a host location

Rehosting, replatforming, or refactoring?

Consider proximity to key systems and data sources

Evaluate usage expectations

Decide who takes ownership of the integration post-deploy

Decide how to manage it all

Moving from monolithic platform to micro-platforms

Create consistent approach to identity mgmt, logging, monitoring

Management centralized, or by team?

What's the hard stuff?

"Modern" platforms don't have feature parity with what you have today

Performance of cloud-based systems doesn't match on-premises throughput

Modernizing integration may require a rewrite versus replatform

Missing business imperative to upgrade integrations

Critical systems aren't in the cloud, or fail to expose useful APIs

Validating and measuring impact of modernization

#1 The “what” + “why” of modernization

#2 Considerations when modernizing integrations

#3 Practices for modernization

Practice

Content-based routing

→ My advice in 2009

Use BizTalk Server with send port subscriptions.

→ My advice in 2019

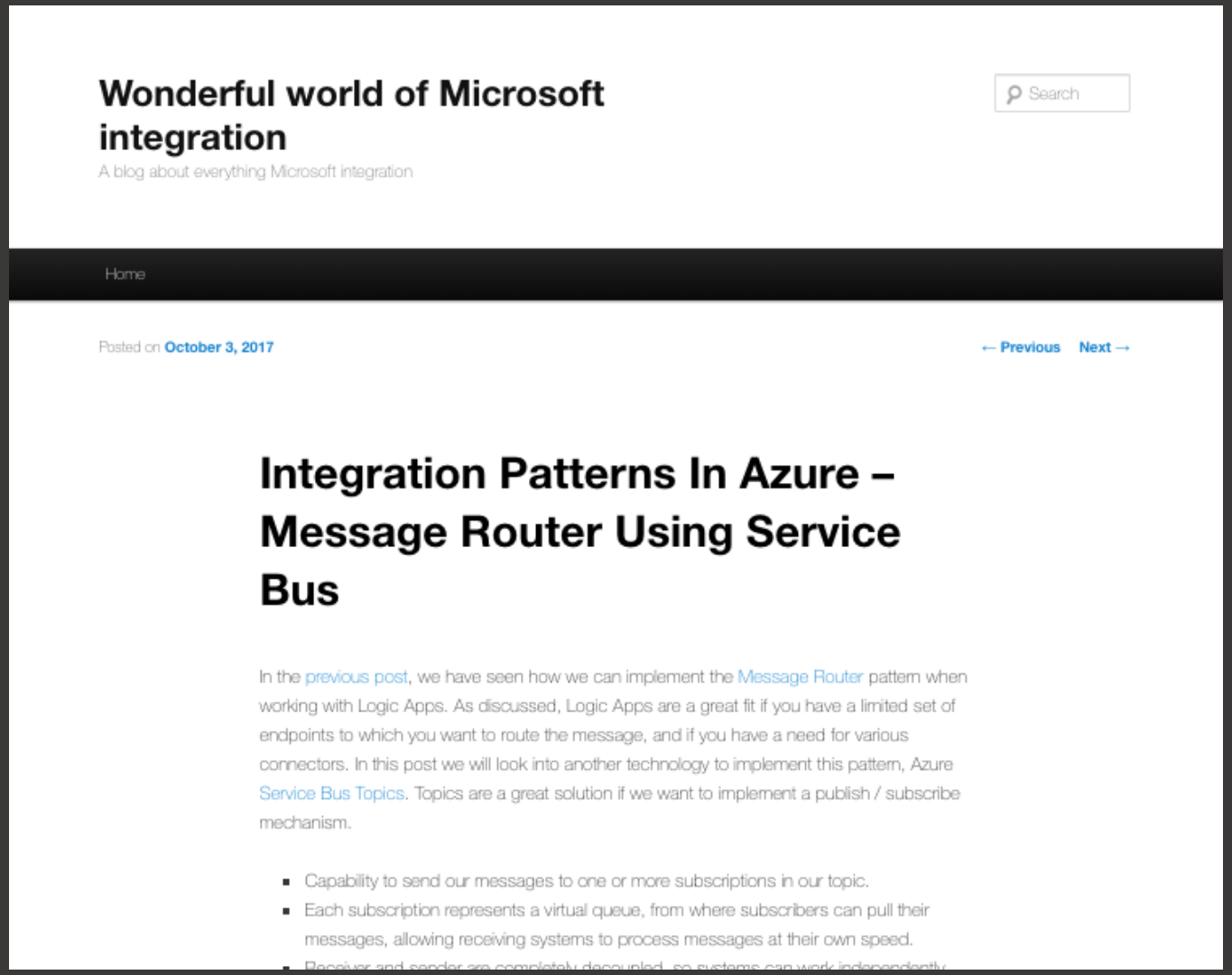
Use BizTalk Server on-premises, and Service Bus (and Logic Apps) for cloud-based routing.

→ Benefits of 2019 advice

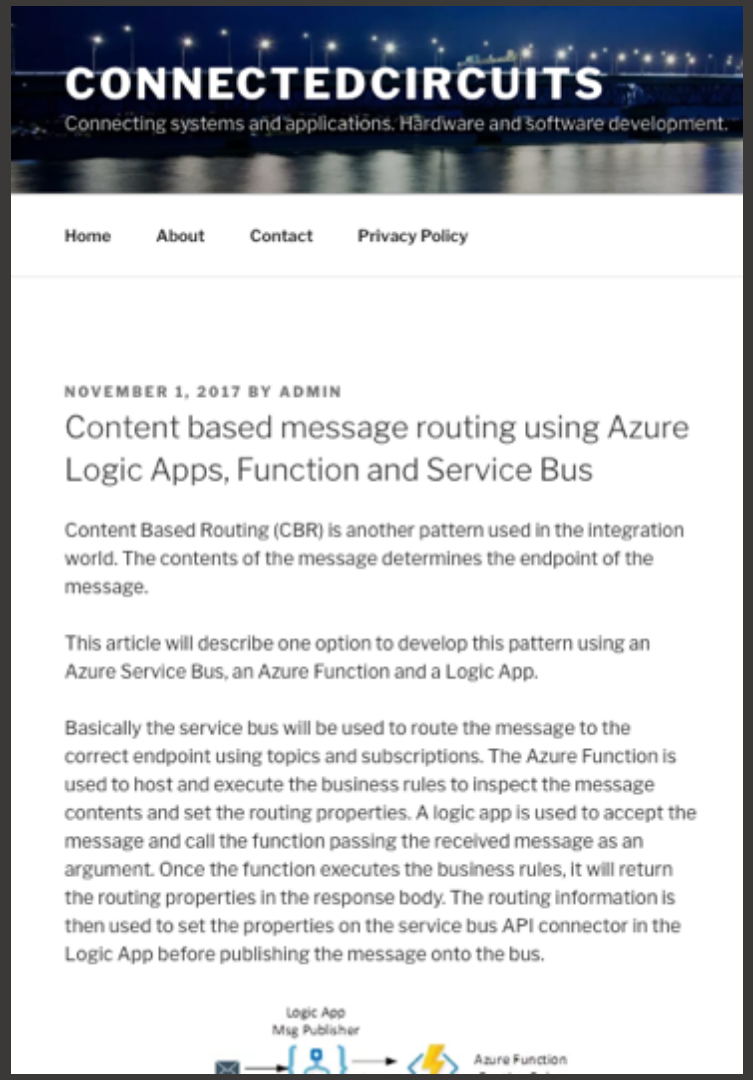
Your messaging engine is scalable and flexible.

→ Risks with 2019 advice

Explicit property promotion needed for Service Bus, or you need Logic Apps to parse the messages. Cloud-based routing rules aren't centralized.



https://blog.eldert.net/integration-patterns-in-azure-message-router-using-service-bus/



https://connectedcircuits.blog/2017/11/01/content-based-message-routing-using-azure-logic-apps-function-and-service-bus/

Practice

De-batching from a database

➔ My advice in 2009

Configure the BizTalk SQL Adapter and de-batch payload in the receive pipeline.

➔ My advice in 2019

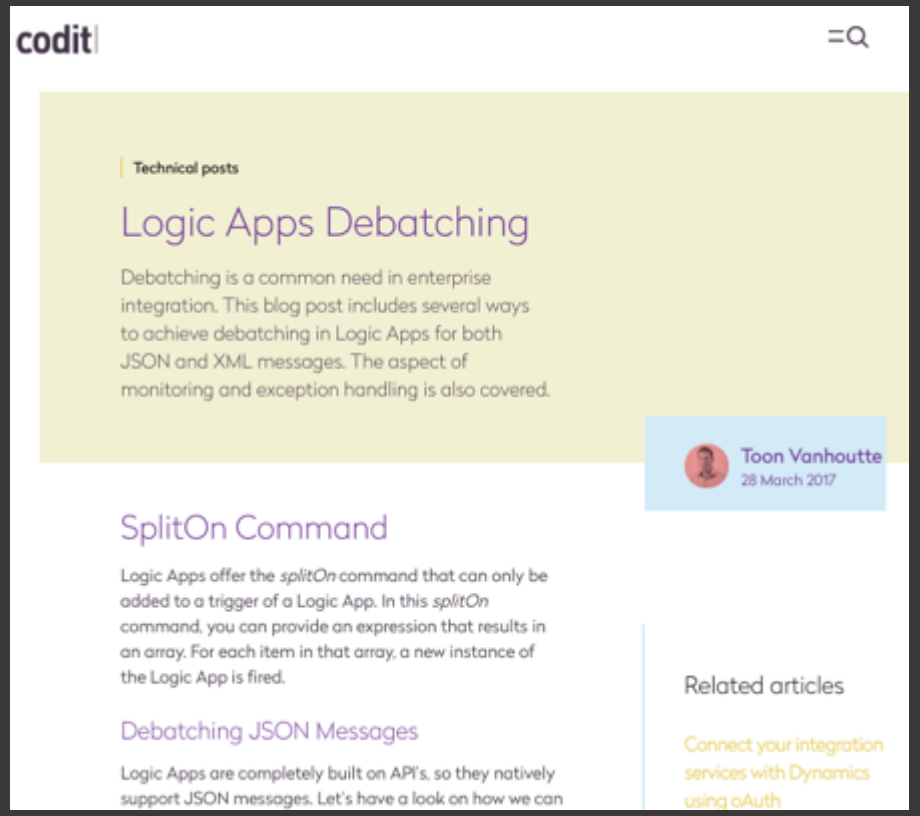
For bulk data, de-batch in a Logic App. Switch to real-time, event-driven change feeds where possible.

➔ Benefits of 2019 advice

With change feeds, process data faster, with less engine-based magic.

➔ Risks with 2019 advice

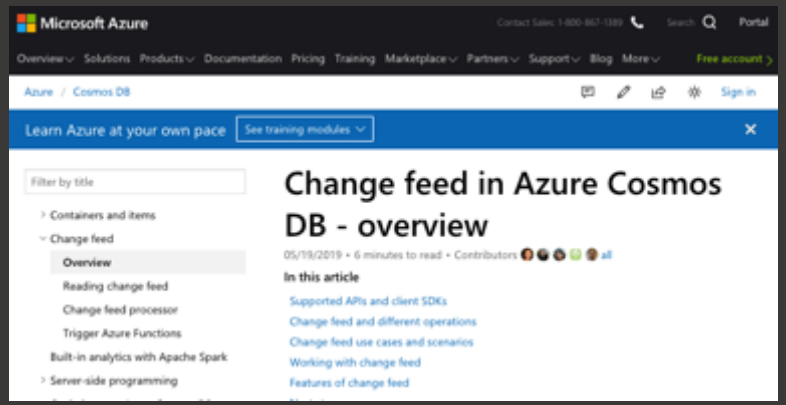
De-batching requires orchestration (Logic Apps) versus pipeline-based de-batching. Can be a more manual setup.



https://www.codit.eu/blog/logic-apps-debatching/

https://azurebiztalkread.wordpress.com/2018/05/26/debatching-using-azure-logic-apps/

https://docs.microsoft.com/en-us/azure/cosmos-db/change-feed



@rseroter

Practice

High-volume data processing

➔ My advice in 2009

Deploy separate servers to run each host type, and avoid orchestration.

➔ My advice in 2019

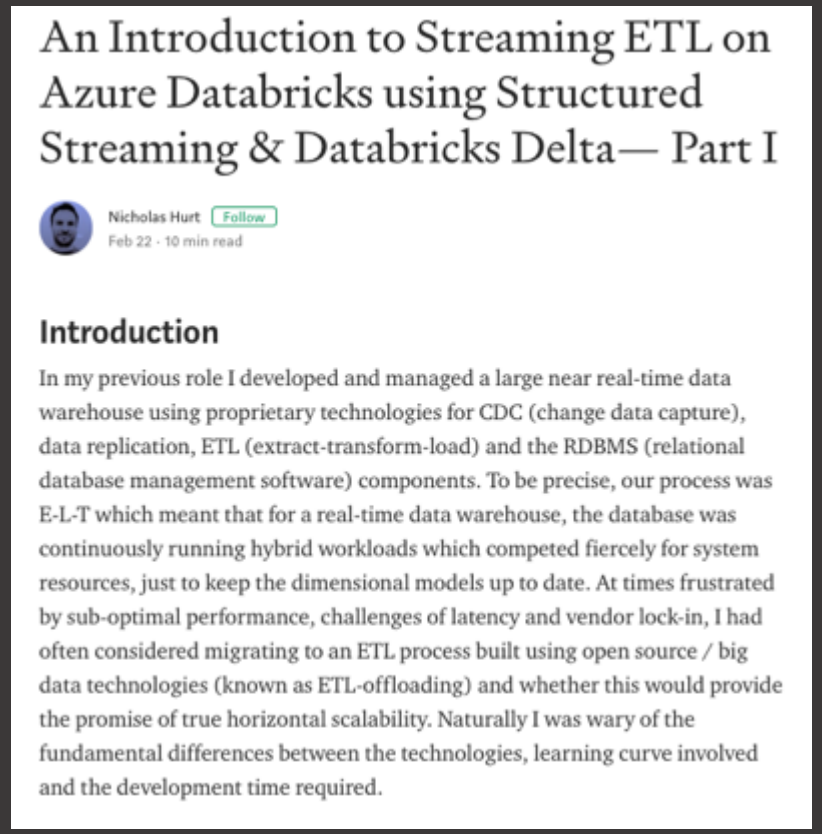
Use cloud services (Event Hubs, Databricks) for real-time event intake and processing. Use services like Azure Data Factory for bulk processing.

➔ Benefits of 2019 advice

Get scale and low maintenance. Have access to novel functionality.

➔ Risks with 2019 advice

Could be surprised by quotas, or bandwidth and storage costs. May also face troubleshooting and recoverability challenges.



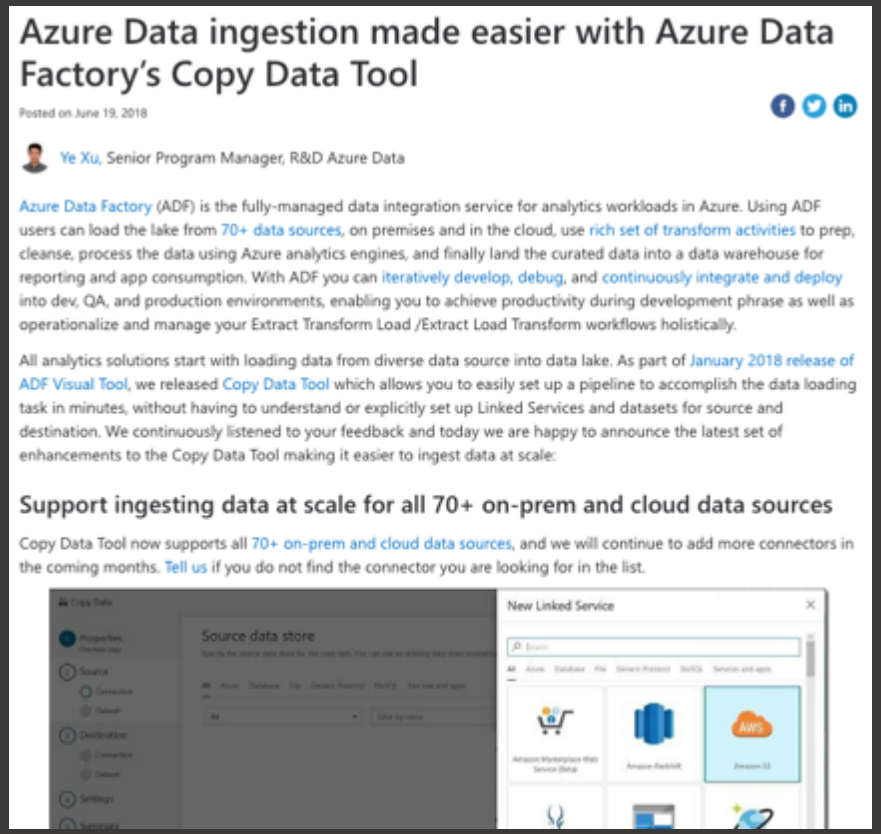
An Introduction to Streaming ETL on Azure Databricks using Structured Streaming & Databricks Delta— Part I

Nicholas Hurt Follow
Feb 22 · 10 min read

Introduction

In my previous role I developed and managed a large near real-time data warehouse using proprietary technologies for CDC (change data capture), data replication, ETL (extract-transform-load) and the RDBMS (relational database management software) components. To be precise, our process was E-L-T which meant that for a real-time data warehouse, the database was continuously running hybrid workloads which competed fiercely for system resources, just to keep the dimensional models up to date. At times frustrated by sub-optimal performance, challenges of latency and vendor lock-in, I had often considered migrating to an ETL process built using open source / big data technologies (known as ETL-offloading) and whether this would provide the promise of true horizontal scalability. Naturally I was wary of the fundamental differences between the technologies, learning curve involved and the development time required.

https://medium.com/microsoftazure/an-introduction-to-streaming-etl-on-azure-databricks-using-structured-streaming-databricks-16b369d77e34



Azure Data ingestion made easier with Azure Data Factory's Copy Data Tool

Posted on June 19, 2018

Ye Xu, Senior Program Manager, R&D Azure Data

Azure Data Factory (ADF) is the fully-managed data integration service for analytics workloads in Azure. Using ADF users can load the lake from 70+ data sources, on premises and in the cloud, use rich set of transform activities to prep, cleanse, process the data using Azure analytics engines, and finally land the curated data into a data warehouse for reporting and app consumption. With ADF you can iteratively develop, debug, and continuously integrate and deploy into dev, QA, and production environments, enabling you to achieve productivity during development phrase as well as operationalize and manage your Extract Transform Load /Extract Load Transform workflows holistically.

All analytics solutions start with loading data from diverse data source into data lake. As part of January 2018 release of ADF Visual Tool, we released Copy Data Tool which allows you to easily set up a pipeline to accomplish the data loading task in minutes, without having to understand or explicitly set up Linked Services and datasets for source and destination. We continuously listened to your feedback and today we are happy to announce the latest set of enhancements to the Copy Data Tool making it easier to ingest data at scale:

Support ingesting data at scale for all 70+ on-prem and cloud data sources

Copy Data Tool now supports all 70+ on-prem and cloud data sources, and we will continue to add more connectors in the coming months. Tell us if you do not find the connector you are looking for in the list.

https://azure.microsoft.com/en-us/blog/data-ingestion-into-azure-at-scale-made-easier-with-latest-enhancements-to-adf-copy-data-tool/

Practice

Replaying data stream

➔ My advice in 2009

Use a BizTalk receive location (or send port that subscribes to all incoming data) that writes to a historian. Send data back through BizTalk as needed.

➔ My advice in 2019

Employ a distributed log like Azure Event Hubs and leave it to clients to access any point in the log.

➔ Benefits of 2019 advice

No extra machinery for data storage, and consumers can retrieve events after the fact.

➔ Risks with 2019 advice

May go overboard and swap out queues for logs when your use case demands the decoupling and event grouping of queues. Long term storage still an issue.

Choose between Azure messaging services - Event Grid, Event Hubs, and Service Bus

01/29/2019 • 4 minutes to read • Contributors all

Azure offers three services that assist with delivering event messages throughout a solution. These services are:

- Event Grid
- Event Hubs
- Service Bus

Although they have some similarities, each service is designed for particular scenarios. This article describes the differences between these services, and helps you understand which one to choose for your application. In many cases, the messaging services are complementary and can be used together.

Event vs. message services

There's an important distinction to note between services that deliver an event and services that deliver a message.

Event

An event is a lightweight notification of a condition or a state change. The publisher of the event has no expectation about how the event is handled. The consumer of the event decides what to do with the notification. Events can be discrete units or part of a series.

Discrete events report state change and are actionable. To take the next step, the consumer only needs to know that something happened. The event data has information about what happened but doesn't have the data that triggered the event. For example, an event notifies consumers that a file was created. It may have general information about the file, but it doesn't have the file itself. Discrete events are ideal for serverless solutions that need to scale.

Series events report a condition and are analyzable. The events are time-ordered and interrelated. The consumer needs the sequenced series of events to analyze what happened.

https://docs.microsoft.com/en-us/azure/event-grid/compare-messaging-services

Connecting your Java microservices to each other? Here's how to use Spring Cloud Stream with Azure Event Hubs.

BY RICHARD SEROTER *on* APRIL 3, 2019 • (3)

You've got microservices. Great. They're being continuous delivered. Neato. Ok … now what? The next hurdle you may face is data processing amongst this distributed mesh o' things. Brokered messaging engines like Azure Service Bus or RabbitMQ are nice choices if you want pub/sub routing and smarts residing inside the *broker*. Lately, many folks have gotten excited by stateful stream processing scenarios and using distributed logs as a shared source of events. In those cases, you use something like Apache Kafka or Azure Event Hubs and rely on smart(er) *clients* to figure out what to read and what to process. **What should you use to build these smart stream processing clients?**

I've written about Spring Cloud Stream a handful of times, and last year showed how to integrate with the Kafka interface on Azure Event Hubs. Just today, Microsoft shipped a brand new "binder" for Spring Cloud Stream that works *directly* with Azure Event Hubs. Event processing engines aren't useful if you aren't actually publishing or subscribing to events, so I thought I'd try out this new binder and see how to light up Azure Event Hubs.

https://seroter.wordpress.com/2019/04/03/connecting-your-java-microservices-to-each-other-heres-how-to-use-spring-cloud-stream-with-azure-event-hubs/

Practice

Sophisticated business rules

➔ My advice in 2009

Use BizTalk's Business Rules Engine, and employ functoids or orchestration-embedded logic sparingly.

➔ My advice in 2019

Extract business rules into standalone APIs and functions. Use maps sparingly when it comes to business logic.

➔ Benefits of 2019 advice

Solution is more flexible, and maintainable by *any* developer.

➔ Risks with 2019 advice

More moving parts and callouts from in-process code.

https://pacodelacruzag.wordpress.com/2018/01/18/business-rules-on-azure-logic-apps-with-liquid-templates/

Practice

Stateful workflow with correlation

➔ My advice in 2009

Use orchestration and take advantage of dehydration, correlation, and transactions with compensation.

➔ My advice in 2019

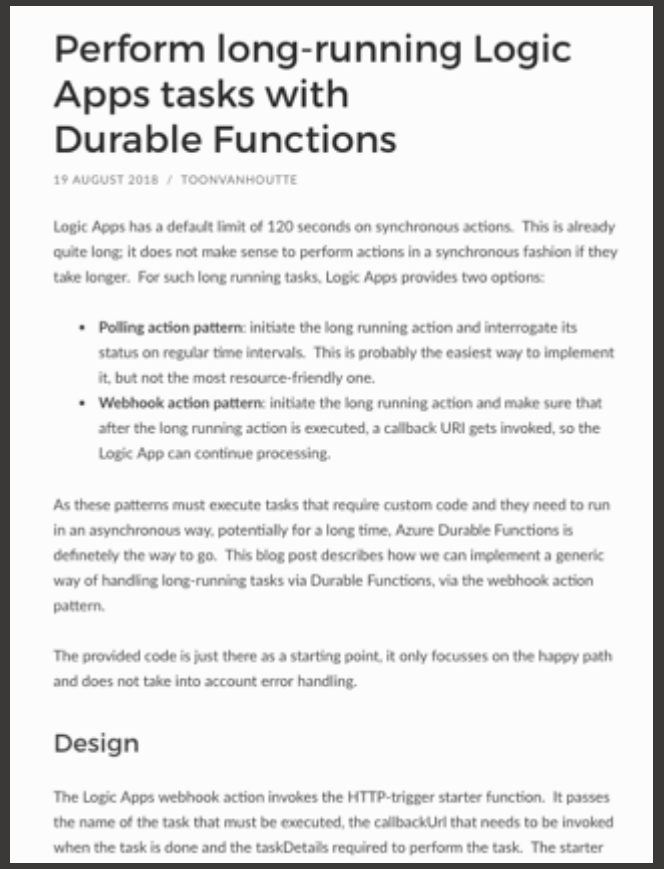
Use Durable Functions for long-running sequences, along with Logic Apps and Service Bus. Break apart giant orchestrations into choreographed sequences.

➔ Benefits of 2019 advice

Easier for any developer to build workflows.

➔ Risks with 2019 advice

You may come across limits in how long a workflow can “wait”, and there is less centralized coordination and observability.



Perform long-running Logic Apps tasks with Durable Functions

19 AUGUST 2018 / TOONVANHOUTTE

Logic Apps has a default limit of 120 seconds on synchronous actions. This is already quite long; it does not make sense to perform actions in a synchronous fashion if they take longer. For such long running tasks, Logic Apps provides two options:

- **Polling action pattern**: initiate the long running action and interrogate its status on regular time intervals. This is probably the easiest way to implement it, but not the most resource-friendly one.
- **Webhook action pattern**: initiate the long running action and make sure that after the long running action is executed, a callback URI gets invoked, so the Logic App can continue processing.

As these patterns must execute tasks that require custom code and they need to run in an asynchronous way, potentially for a long time, Azure Durable Functions is definetely the way to go. This blog post describes how we can implement a generic way of handling long-running tasks via Durable Functions, via the webhook action pattern.

The provided code is just there as a starting point, it only focusses on the happy path and does not take into account error handling.

Design

The Logic Apps webhook action invokes the HTTP-trigger starter function. It passes the name of the task that must be executed, the callbackUrl that needs to be invoked when the task is done and the taskDetails required to perform the task. The starter

https://toonvanhoutte.wordpress.com/2018/08/19/perform-long-running-logic-apps-tasks-with-durable-functions/

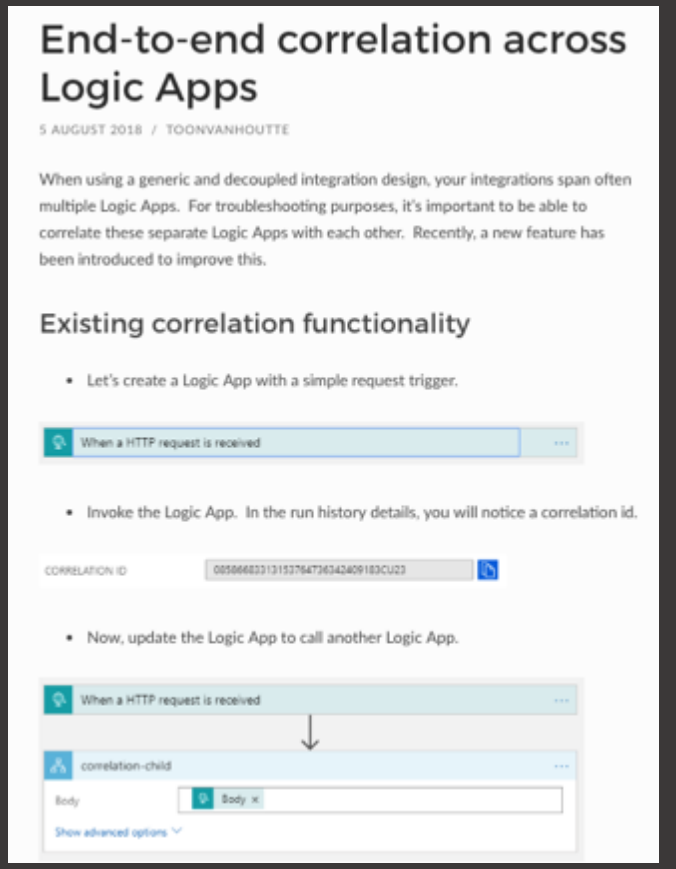
Implementing the Correlation Identifier Pattern on Stateful Logic Apps using the Webhook Action

Posted on July 17, 2017 by Paco de la Cruz

Introduction

In many business scenarios, there is the need to implement long-running processes which first send a message to a second process and then pause and wait for an asynchronous response before they continue. Being this an asynchronous communication, the challenge is to correlate the response to the original request. The Correlation Identifier enterprise integration pattern targets this scenario.

https://pacodelacruzag.wordpress.com/2017/07/17/correlation-identifier-pattern-on-logic-apps/



End-to-end correlation across Logic Apps

5 AUGUST 2018 / TOONVANHOUTTE

When using a generic and decoupled integration design, your integrations span often multiple Logic Apps. For troubleshooting purposes, it's important to be able to correlate these separate Logic Apps with each other. Recently, a new feature has been introduced to improve this.

Existing correlation functionality

- Let's create a Logic App with a simple request trigger.

- Invoke the Logic App. In the run history details, you will notice a correlation id.

- Now, update the Logic App to call another Logic App.

https://toonvanhoutte.wordpress.com/2018/08/05/end-to-end-correlation-across-logic-apps/

Practice

Complex data trans-formation

➔ My advice in 2009

Use the BizTalk Mapper to transform data structures, and take advantage of functoids and inline code.

➔ My advice in 2019

Map data on the way out if at all, and use Liquid Templates for transformation, but not business logic. Also consider transforming in code (Functions).

➔ Benefits of 2019 advice

Avoid embedded too much brittle logic within a map, and leave it up to receivers to handle data structure changes.

➔ Risks with 2019 advice

Not suitable for flat files or extremely difficult transformations. Puts new responsibilities on client consumers.

Published 2019-01-06 by Justin Yoo

Building XSL Mapper with Azure Functions

> DISCLAIMER: This post is purely a personal opinion, not representing or affiliating my employer's.

In order to use BizTalk server for your service integration project, you can't avoid using XML transformation. If you're about to migrate these integration components into Azure iPaaS, Logic Apps and Integration Account should be necessary. Integration Account provides many features like XML schema mapping, XML data transformation, extension objects storage, etc. However, it's way too expensive, which costs more than AU$410 per month.

Therefore, using Integration Account might not be cost-efficient, especially if you're using Azure Functions and/or Logic Apps, which is way cheaper than Integration Account. Fortunately, our fellow Azure MVP, Toon Vanhoutte wrote an awesome blog post that discussed how to write a Web API application as an alternative to Integration Account. In addition to that, throughout this post, I'm going to write an Azure Function app doing the same job, instead of Web API app.

> All sample codes used in this post can be found at here.

Azure Functions Version Selection

The `XslCompiledTransform` class must be used for this feature. Especially the XSLT mapper feature in the BizTalk Server relies on the inline C# script feature, but unfortunately, it's not supported yet. If you run the Function app 2.x, it throws the following error message.

```
Compiling JScript/CSharp scripts is not supported
```

At the time of writing this post, .NET Core doesn't supports this feature. Even there's no plan to implement this feature. In other words, we can't use Azure Functions 2.x, which uses .NET Core 2.x. Therefore, we have to stay on 1.x.

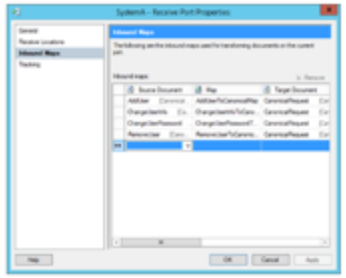
Configurations

We now got the correct version of Azure Functions. Now we need to setup environment variables. In your local dev box, simply use `local.settings.json`. Then set the App Settings blade on your Azure Functions instance.

https://devkimchi.com/2019/01/07/building-xsl-mapper-with-azure-functions/

Inbound / Outbound Maps in Logic Apps

BizTalk Server offers a great feature that both inbound (receive ports) and outbound maps (send ports) can be executed in dynamic fashion, depending on the message type of the message. This message type is defined as rootNodeNamespace#rootNodeName. Below, you can find an example of a receive port configured with several inbound maps.



When migrating ports of BizTalk solutions to Azure Logic Apps, it's really handy to reuse this pattern. This blog post explains how you can do this.

https://www.codit.eu/blog/inbound-outbound-maps-in-logic-apps/

Technical posts

Complex Transformations in Logic Apps

Recently, we faced the challenge to perform complex transformations in Logic Apps. We had an EDIFACT D96A ORDER parsed into XML, that had to be transformed into a generic JSON Order format. Let's have a look the issues we faced!

Liquid templates are insufficient

The first reflex was to go for Liquid templates, because the expected output format was JSON. Pretty soon, we realized that Liquid has too many limitations for our scenario:

- Input needs to be transformed into JSON first
- Unreadible syntax if your input XML has complex namespace structures
- Conditional select (e.g. Party with qualifier = BY) not possible without a for each
- No way to inject custom .NET code into Liquid templates

Let's go with XSLT

As a second attempt, we decided to go for XSLT! We leveraged the *Transform XML* action, which executes an XSLT mapping. We transformed the EDI XML into the XML representation of

https://www.codit.eu/blog/complex-transformations-in-logic-apps/

@rseroter

Practice

Integrating with cloud endpoints

➔ My advice in 2009

Call cloud endpoints using HTTP adapter and custom pipeline components for credentials or special formatting.

➔ My advice in 2019

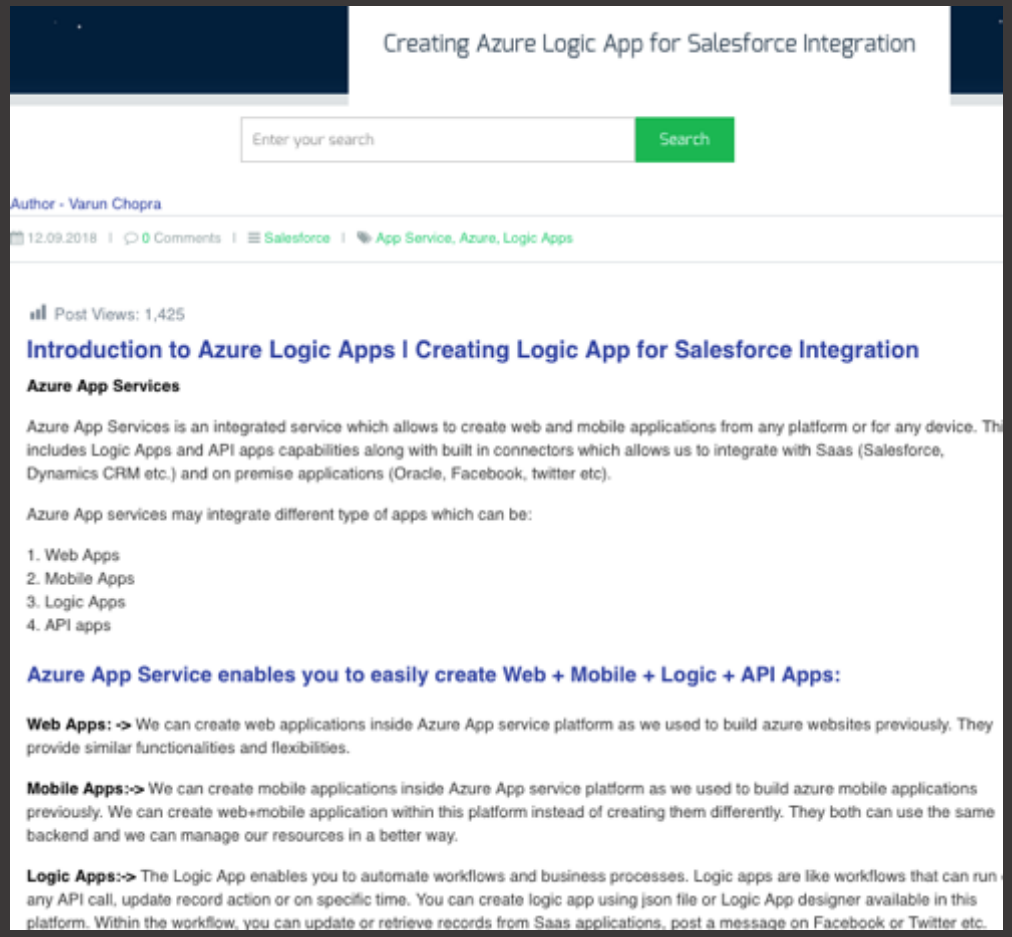
Use Logic Apps and connectors for integration with public cloud services. Use Logics Apps adapter for BizTalk where needed.

➔ Benefits of 2019 advice

Any developer can integrate with cloud endpoints, and you have more maintainable integrations.

➔ Risks with 2019 advice

More components from more platforms participating in an integration.



Creating Azure Logic App for Salesforce Integration

Enter your search

Search

Author - Varun Chopra

12.09.2018 | 0 Comments | Salesforce | App Service, Azure, Logic Apps

Post Views: 1,425

Introduction to Azure Logic Apps I Creating Logic App for Salesforce Integration

Azure App Services

Azure App Services is an integrated service which allows to create web and mobile applications from any platform or for any device. Th
includes Logic Apps and API apps capabilities along with built in connectors which allows us to integrate with Saas (Salesforce,
Dynamics CRM etc.) and on premise applications (Oracle, Facebook, twitter etc).

Azure App services may integrate different type of apps which can be:

1. Web Apps
2. Mobile Apps
3. Logic Apps
4. API apps

Azure App Service enables you to easily create Web + Mobile + Logic + API Apps:

Web Apps: -> We can create web applications inside Azure App service platform as we used to build azure websites previously. They provide similar functionalities and flexibilities.

Mobile Apps:-> We can create mobile applications inside Azure App service platform as we used to build azure mobile applications previously. We can create web+mobile application within this platform instead of creating them differently. They both can use the same backend and we can manage our resources in a better way.

Logic Apps:-> The Logic App enables you to automate workflows and business processes. Logic apps are like workflows that can run
any API call, update record action or on specific time. You can create logic app using json file or Logic App designer available in this
platform. Within the workflow, you can update or retrieve records from Saas applications, post a message on Facebook or Twitter etc.

https://blog.webnersolutions.com/creating-azure-logic-app-for-salesforce-integration

How to send daily SMS messages with Azure Logic Apps and Twilio – no code required

Posted on December 26, 2017 by santoshhari

> "Life is really simple, but we insist on making it complicated."
> – Confucius

Recently, I had to transition from a legacy messaging provider to Twilio for sending SMS messages and found it ridiculously simple. Also, the built-in hooks between different Azure services and Twilio appeared to have reduced the integration process to a few button clicks instead of code. Based on this experience, I wanted to show how easy it was to send a daily SMS message with zero coding.

What you will need:

1. Azure account: here's how you can signup for a free one.
2. Twilio account: here's how you can signup for a free one although I recommend paying – individual messages are relatively low cost and so is getting a dedicated phone number.

Steps:

https://santoshhari.wordpress.com/2017/12/26/send-daily-sms-azure-logic-apps-twilio-no-code/

Practice

Throttling and load-leveling

→ My advice in 2009

Configure throttling settings in BizTalk Server, and use an outside queue or database as buffer.

→ My advice in 2019

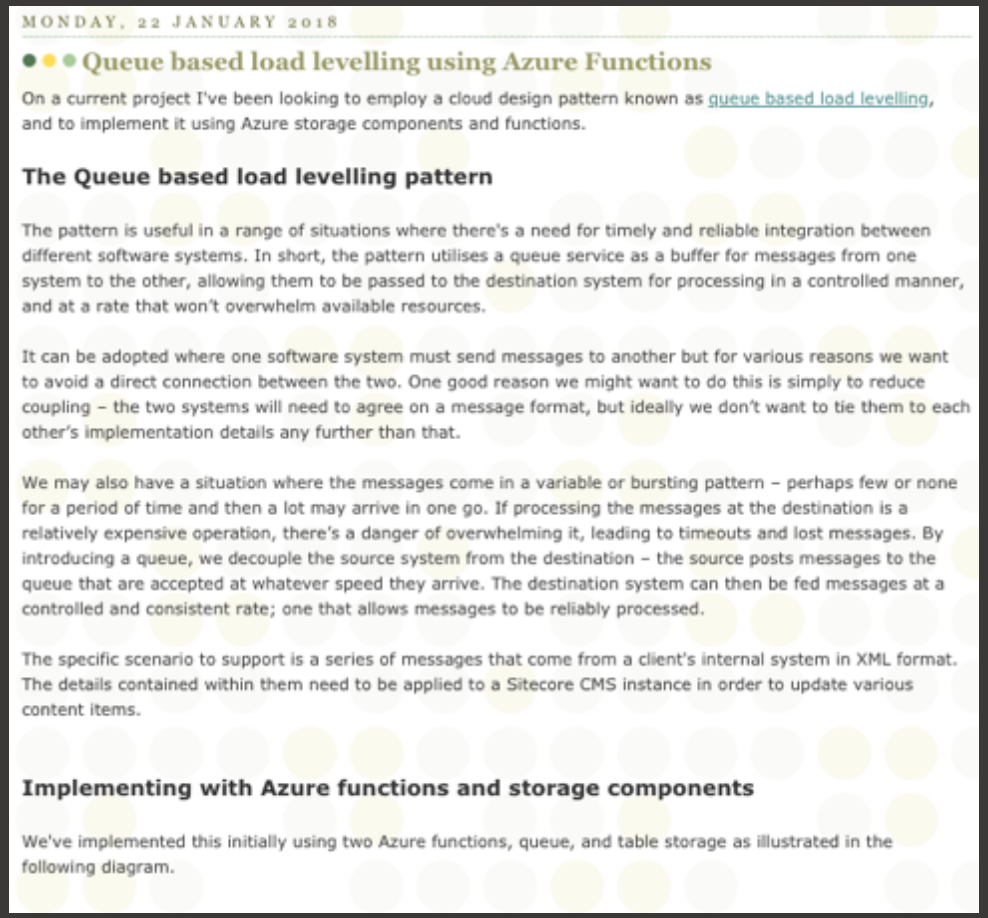
Separate intake from processing. Use a cloud-based queue or log for intake, and consider Azure API Management for web request throttling.

→ Benefits of 2019 advice

Near infinite scale without pre-provisioning or maintenance concerns. Protect your cloud or on-premises systems.

→ Risks with 2019 advice

Latency is a risk, and you must secure all transport paths (plus payloads).



MONDAY, 22 JANUARY 2018

Queue based load levelling using Azure Functions

On a current project I've been looking to employ a cloud design pattern known as queue based load levelling, and to implement it using Azure storage components and functions.

The Queue based load levelling pattern

The pattern is useful in a range of situations where there's a need for timely and reliable integration between different software systems. In short, the pattern utilises a queue service as a buffer for messages from one system to the other, allowing them to be passed to the destination system for processing in a controlled manner, and at a rate that won't overwhelm available resources.

It can be adopted where one software system must send messages to another but for various reasons we want to avoid a direct connection between the two. One good reason we might want to do this is simply to reduce coupling – the two systems will need to agree on a message format, but ideally we don't want to tie them to each other's implementation details any further than that.

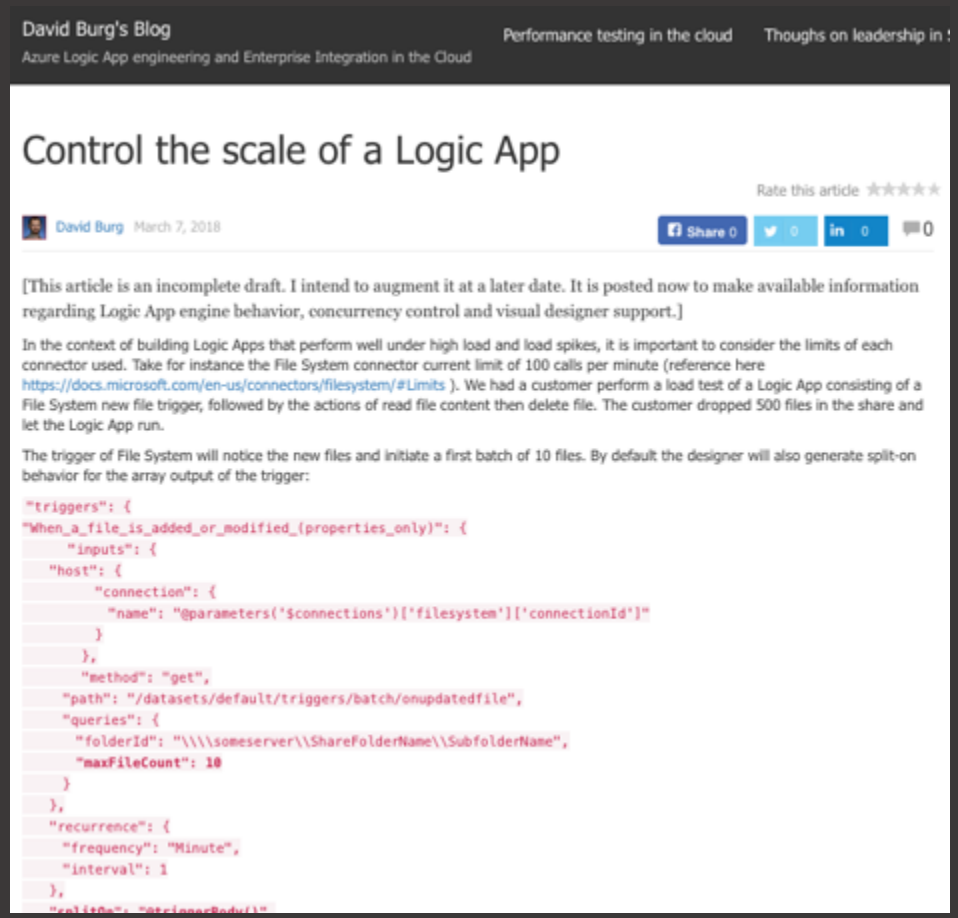
We may also have a situation where the messages come in a variable or bursting pattern – perhaps few or none for a period of time and then a lot may arrive in one go. If processing the messages at the destination is a relatively expensive operation, there's a danger of overwhelming it, leading to timeouts and lost messages. By introducing a queue, we decouple the source system from the destination – the source posts messages to the queue that are accepted at whatever speed they arrive. The destination system can then be fed messages at a controlled and consistent rate; one that allows messages to be reliably processed.

The specific scenario to support is a series of messages that come from a client's internal system in XML format. The details contained within them need to be applied to a Sitecore CMS instance in order to update various content items.

Implementing with Azure functions and storage components

We've implemented this initially using two Azure functions, queue, and table storage as illustrated in the following diagram.

https://web-matters.blogspot.com/2018/01/queue-based-load-levelling-using-azure.html



David Burg's Blog

Azure Logic App engineering and Enterprise Integration in the Cloud

Performance testing in the cloud Thoughs on leadership in

Control the scale of a Logic App

Rate this article

David Burg March 7, 2018

Share 0 0 in 0 0

[This article is an incomplete draft. I intend to augment it at a later date. It is posted now to make available information regarding Logic App engine behavior, concurrency control and visual designer support.]

In the context of building Logic Apps that perform well under high load and load spikes, it is important to consider the limits of each connector used. Take for instance the File System connector current limit of 100 calls per minute (reference here https://docs.microsoft.com/en-us/connectors/filesystem/#Limits). We had a customer perform a load test of a Logic App consisting of a File System new file trigger, followed by the actions of read file content then delete file. The customer dropped 500 files in the share and let the Logic App run.

The trigger of File System will notice the new files and initiate a first batch of 10 files. By default the designer will also generate split-on behavior for the array output of the trigger:

```
"triggers": {
"When_a_file_is_added_or_modified_(properties_only)": {
    "inputs": {
  "host": {
      "connection": {
        "name": "@parameters('$connections')['filesystem']['connectionId']"
      }
    },
    "method": "get",
  "path": "/datasets/default/triggers/batch/onupdatedfile",
  "queries": {
   "folderId": "\\\\someserver\\ShareFolderName\\SubfolderName",
   "maxFileCount": 10
  }
 },
 "recurrence": {
  "frequency": "Minute",
  "interval": 1
 },
```

https://blogs.msdn.microsoft.com/david_burgs_blog/2018/03/07/control-the-scale-of-a-logic-app/

Practice

Strangling your legacy ESB

➔ My advice in 2009

Put new integrations into the new system, and rebuild existing ones over time.

➔ My advice in 2019

Similar to 2009, but avoid modernizing to a single environment or instance. Use Event Storming to find seams to carve out.

➔ Benefits of 2019 advice

Get into managed systems that offload operational cost and are inviting to more developers.

➔ Risks with 2019 advice

You'll have a (lengthy?) period of dual costs and skillsets.

Practice

Getting integrations into production

➔ My advice in 2009

Package up BizTalk libraries, bindings, scripts, and policies into an MSI and deploy carefully.

➔ My advice in 2019

Put on-premises and cloud apps onto continuous integration and delivery pipelines. Aim for zero-downtime deploys.

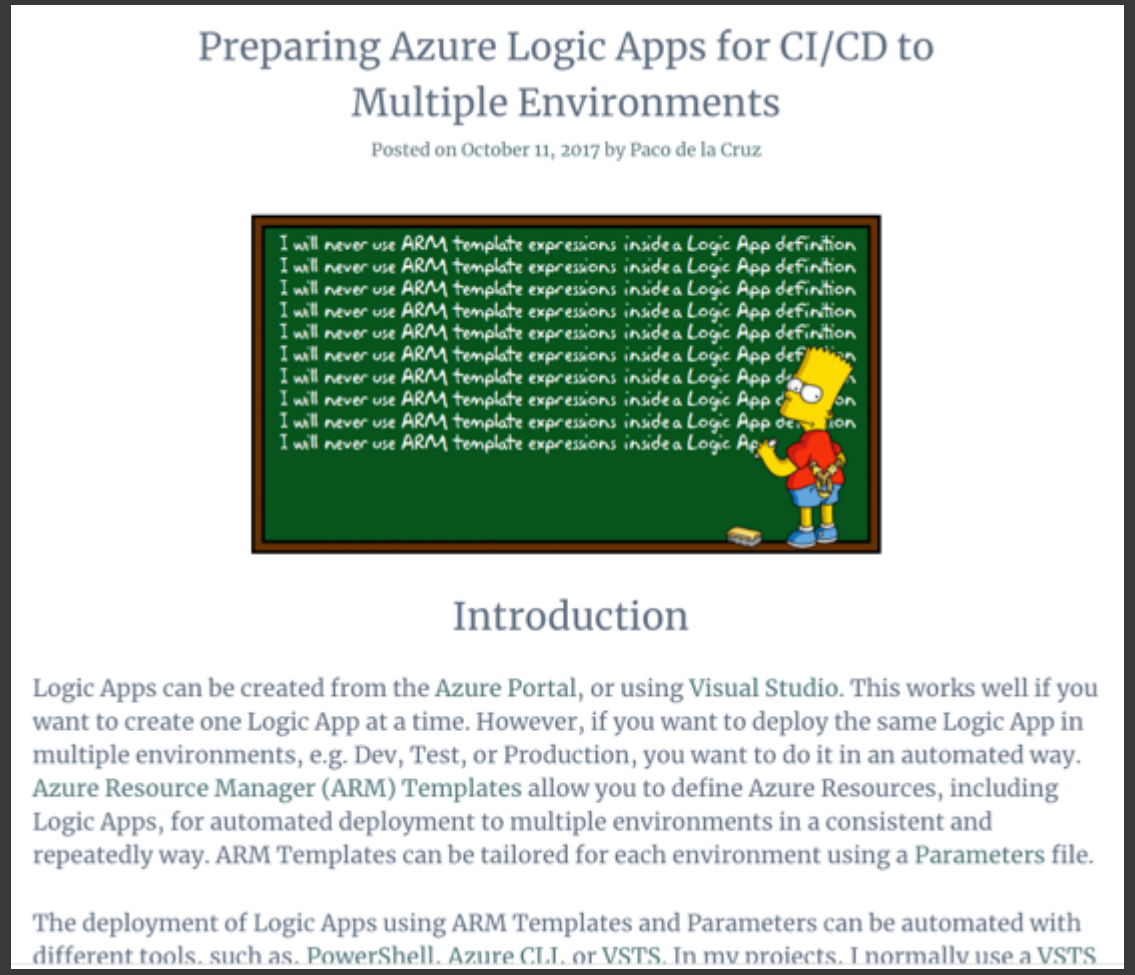
➔ Benefits of 2019 advice

Reduce downtime, improve delivery velocity and reliability. Introduce automation that replaces human intervention.

➔ Risks with 2019 advice

Complicated to set up with multi-component integrations. Risk of data loss or ordering anomalies when upgrades roll out.

https://www.serverless360.com/blog/azure-logic-apps-life-cycle-the-big-picture



https://pacodelacruzag.wordpress.com/2017/10/11/preparing-azure-logic-apps-for-cicd/

Practice

Building integration teams

➔ My advice in 2009

"Invest in training and building a Center of Excellence."

➔ My advice in 2019

Integration experts should coach and mentor developers who use a variety of platforms to connect systems together.

➔ Benefits of 2019 advice

Fewer bottlenecks waiting for the "integration team" to engage, and more types of simple integration get deployed.

➔ Risks with 2019 advice

More distributed ownership and less visibility into all the integrations within the company.

You need a thoughtful modernization strategy for integration. Make clear decisions about the approaches and technologies for your portfolio.