Microsoft IT: An integration journey to Azure Logic Apps

July 17, 2017, Version 1.0, prepared by Divya Swarnkar and Anil Kumar Sehgal

Microsoft has an integration footprint that spans businesses and service lines. As Microsoft enters new business segments and builds new partner relationships, the scale and complexity of integration continues to grow. Like any other enterprise, Microsoft wants a solution that scales, supports business needs and demands, lets them focus on the business, and shields them from the underlying complexities of integrations.

The Microsoft integration footprint includes hundreds of partners that send millions of messages per month. Most of this integration uses BizTalk Server, an infrastructure-as-a-service (IaaS) offering. To a lesser extent, this integration also uses Microsoft Azure BizTalk Services (MABS), a platform-as-a-service (PaaS) offering. However, Microsoft wants to incrementally move their integration to Microsoft Azure.

This case study covers how Microsoft IT built an integration service on the Azure Logic Apps platform along with a migration strategy to move from on premises to the cloud.

In this article

Why move to the cloud?

Microsoft IT has their integration footprint on BizTalk Server and Microsoft Azure BizTalk Services (MABS). They wanted to migrate to a cloud integration-platform-as-a-service (iPaaS) offering, like Azure Logic Apps, for these reasons:

- **Business agility**: At Microsoft, the trading partner and customer ecosystem is huge and constantly growing. As Microsoft IT, we wanted an integration platform that helps us rapidly meet business needs, scales our service along with growing demand, and supports hybrid integrations, while we incrementally move our partner and customer ecosystem completely to the cloud.

- **Serverless computing and managed services**: One of our goals is to spend more time on growing the business and less time on managing the integration service. The key to focusing on the business is not worrying about service hosting, deployment, and scalability. We also wanted to stay on a modern stack and benefit from the latest features and capabilities in Azure Logic Apps and the Enterprise Integration Pack (EIP).

- **Simplify and accelerate the development and onboarding efforts**: To support the growth of businesses that use the Electronic Data Interchange (EDI) and non-EDI protocols, we wanted fast onboarding, rapid development, and the ability to seamlessly support the service.

- **Cost savings - control costs and pay per use**: Huge investments upfront can become underused. We wanted more control over what we paid and pay only for what we used.

- **Business continuity**: Microsoft promises and provides business continuity to partners and customers. We needed a platform that provided high availability and disaster recovery capabilities for integration solutions.

What were our business goals?

Here are our goals for migrating to Azure Logic Apps from MABS and BizTalk Server, along with core principles that support these goals:

- No impact on the business
  - Minimal involvement and changes required from partners during migration
  - No impact to existing partners while we migrate other partners
- Accelerate onboarding
  - Speed onboarding and decouple platform from onboarding, which lets us onboard partners and line-of-business (LOB) apps without affecting the platform.

What were our requirements?

We wanted to migrate our integration footprint to platform-as-a-service (PaaS) as much as possible, but planned to implement this move in phases. Before we chose to move to PaaS and use Logic Apps with the Enterprise Integration Pack, we had these key requirements:

- **Support EDI capabilities**: To enable business-to-business (B2B) integration scenarios using Logic Apps, we needed support for AS2, X12, and EDIFACT protocols, along with XML processing and the capability to write custom components. Logic Apps provides this capability through out-of-the-box integration account connectors. Along with these connectors, Logic Apps supports custom code through Azure Functions.

- **Hybrid integrations**: Most of our line-of-business (LOB) applications are on premises, so we needed a platform that supports integrations across these environments. Logic Apps provides first-class support for hybrid patterns and out-of-the-box connectors for this purpose.

- **Performance and scalability**: Organizations have very specific performance and scalability requirements. We reviewed the published limits for Logic Apps and connectors to verify that they met our requirements.

- **Business continuity**: To promise a highly available and reliable integration service to our customers, we need the ability to run the service in multiple regions. Logic Apps provides cross-region, active-passive disaster recovery capabilities.

- **Management and supportability**: To manage and support the integration service, we needed the capability to track messages and monitor resources. Logic Apps provides first-class, out of the box tracking and monitoring experiences.

- **Compliance**: As a business requirement, we must only use a Drummond-Certified™ platform for EDI communication over Applicability Statement 2 (AS2) protocol. The Enterprise Integration Pack supports the AS2 protocol for communicating EDI messages and is Drummond-Certified™.

What architecture did we use?

This diagram illustrates end-to-end (E2E) architecture of B2B integration in cloud. The left side shows an external trading partner, while the right side shows internal line-of-business (LOB) partners. The middle highlights the heart of the integration pipeline, in this case, B2B processing using Azure Logic Apps.
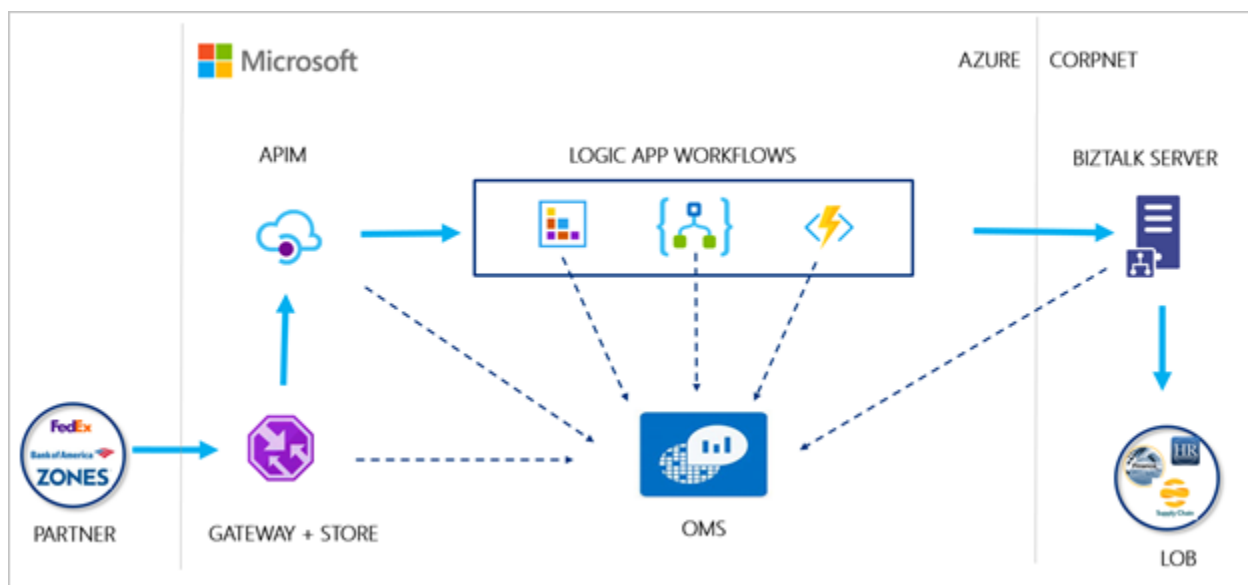
A core principle for any migration is putting minimal to zero impact on partners and systems, both upstream and downstream. This principle is the reason why **BizTalk Server still appears in the pipeline**. Here, BizTalk Server is only used as a proxy for connecting from Azure to LOB apps on the corporate network (CorpNet), without requiring any changes from downstream systems. This approach is an interim strategy, so we plan to remove BizTalk Server in the next phases.

For message processing, requests that we receive from a trading partner arrive at our internal gateway first. We store the incoming request and then send the request to Azure API Management (APIM). APIM provides an abstraction layer and is also used for applying any policies to an incoming message before that message is

processed. The message is then routed to Logic Apps workflows for more processing. The processed message is delivered to the LOB apps through Azure Express Route.

Tracking and monitoring messages

Along with message processing, we needed the capability to track the messages at any time and monitor for failures. All the components in the end-to-end flow shown by the diagram are enabled to send diagnostics data to Microsoft Operations Management Suite (OMS). The diagnostics data in OMS lets us get a very cohesive view of the processing pipeline and messages in this hybrid system.

How did we implement our migration strategy?

We'll now cover our migration strategy and the steps we followed to enable B2B integration using the features in Logic Apps and the Enterprise Integration Pack.

Migrating artifacts

In B2B communication, the building blocks are trading partners, agreements, schemas, and certificates. If you use BizTalk Server or Microsoft Azure BizTalk Services (MABS), the Enterprise Integration Pack features use similar concepts, so they might be familiar and easier to use.

However, one major difference is that the Enterprise Integration Pack uses integration accounts to simplify the way you store and manage the artifacts used in B2B communication. So, any created B2B workflow will use an integration account to access these artifacts, like schemas, maps, certificates, and so on. You can upload B2B artifacts to an integration account directly through the Azure portal or with PowerShell commandlets, if you want to automate the process.

Partners

You create and manage B2B trading partners in your integration account through the Azure portal. You can migrate all partner information from MABS and BizTalk Server without changing the integration account.

Agreements

You create and manage B2B agreements between partners in your integration account. Like MABS and BizTalk Server, we can create agreements between partners using protocols, like AS2, X12, and EDIFACT. You can also configure agreements with options for sending and receiving messages, acknowledgements, and for other advanced scenarios.

Notes

- An integration account can have two agreements with same identifiers. However, you might experience errors or unexpected behavior at runtime. So as a best practice, use unique identifiers.

- Unlike MABS and BizTalk Server, EDIFACT agreements require qualifiers for partners.

Schemas

Like other artifacts, you upload schemas to your integration account. If your schemas are smaller than 2 MB, you can upload them directly to your integration account. Otherwise, you must upload your schemas to Azure Storage and link those schemas to your integration account using their Azure Storage URI. If you use PowerShell commandlets, you can upload schemas directly to your integration account without adding those schemas to Azure Storage.

Notes

- UTF-8 encoding is required for all schemas.
- If your schemas have references, you must follow the order of reference (referenced ones first) when you upload the schemas to your integration account.
- BizTalk Server identifies schemas with a combination of DLL name and namespace. Logic Apps identifies schemas by their names. To avoid overrides, make sure that you give unique names to your schemas.

Maps

In Logic Apps, maps are in XSLT format and are stored in your integration account. However, migrating maps, or "transforms", from Microsoft Azure BizTalk Services (MABS), is more complex. Currently, there isn't a tool to convert MABS transforms to XSLT maps for Logic Apps. So, maps were entirely rewritten in XSLT. Alternatively, you can create maps in BizTalk Server and generate XSLT using Visual Studio.

Like schemas, maps use names as identifiers. To avoid overrides, make sure you give unique names to your maps.

Developing logic app workflows

Here are some goals and principles we used to design and architect logic app workflows:

- Pattern-based workflows, rather than partner-based
  - Make dynamic decisions using partner information.
- Modular and extensible
  - Self-contained workflows that provide unique capabilities
  - Extensible in nature so we can add more capabilities without impacting existing components
  - Fail fast and resume from point of failure
- Test in production
  - Support test in production and if necessary, can fall back to the last known good state

Designing workflows with Logic Apps

This diagram shows the design for our B2B workflows using Logic Apps. We have an example inbound flow that can process an EDI document, like a purchase order, and deliver that document to LOB apps. Processing works similarly for both inbound and outbound scenarios.
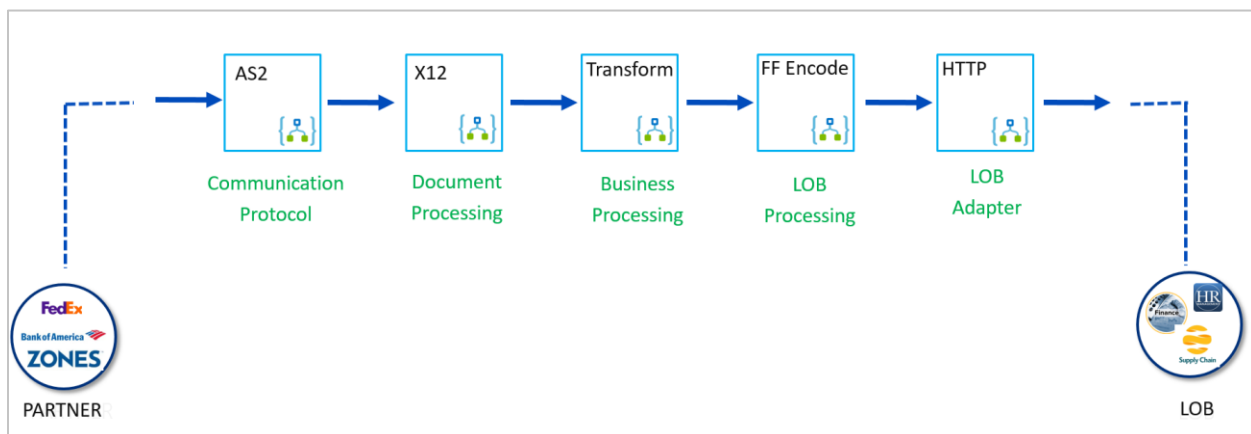
Here we separated our logic apps into discrete and self-contained workflows, which help us make the whole system more modular. This separation is based on the unique processing stages in a typical B2B processing pipeline. Most of these logic apps are linked to our integration account so they can reference our B2B artifacts. We also heavily used Azure Functions for custom processing.

If you have integration services in MABS or BizTalk Server, you might find that logic app workflows are like bridges and pipelines in MABS and BizTalk Server.

Processing messages

Now we'll discuss key steps for processing messages through the logic apps shown in the diagram.

- **Communication Protocol**: This logic app supports multiple transport protocols. For this scenario, we were receiving EDI messages with AS2 protocol from a trading partner. So, our logic app has these key steps:

  o AS2 Decode: Processes the message based on how the AS2 agreement is configured.

  o HTTP Response: AS2 Decode also generates a Message Disposition Notification (MDN) for the received message. The generated MDN is returned to the trading partner over a synchronous or asynchronous connection, per agreement settings. In MABS and BizTalk Server, the MDNs were returned to partners natively, while in Logic Apps, we had to create the appropriate response.

- **Document Processing**: This logic app validates the EDI message against the EDI agreement settings and schema. The received batch is also split into transactions sets as specified in the agreement settings. So, our logic app has these key steps:

  o X12 Decode: Validate the EDI message against the agreement settings. X12 Decode generates these arrays:

    - **Good Messages**: Contains messages that were successfully processed. These messages are passed downstream for further processing.
    - **Bad Messages**: Contains messages that failed X12 processing. Their diagnostic logs are used for alerts and troubleshooting.
    - **Generated Acks**: Contains functional acknowledgements (997) that were generated for the processed messages. These acknowledgements can be positive or negative, depending on the processing outcome.
    - **Received Acks**: Contains functional acknowledgements that were received from the trading partner.

  o HTTP Response: Enumerate and return the generated functional acknowledgements to the trading partner through outbound flow.

- **Business Processing:** This logic app is used for XML processing and has these key steps:
  o XML validation and transformation to validate and convert the XML per the XSLT based maps
  o Promote properties before and after transformation, and add them to tracked properties.

- **LOB Processing:** This logic app has processing steps as required by the LOB app. For example, an XML message going to SAP undergoes flat file encoding in this workflow.

- **LOB Adapter:** This logic app is used for connecting to an LOB app. In our scenario, the logic app workflow has an "HTTP Post" action with a URI to the BizTalk Server "Receive Location".

Testing logic apps and end-to-end workflows

With the B2B integration solution including multiple logic apps, we have test cases that target individual logic apps as well as the end-to-end flows. To deploy and run a test suite, we used Visual Studio Team Services. The test suite runs on a test agent that is provisioned by Team Services and runs only after successful deployment. After the run completes, the test results are available on the Team Services dashboard.

Test types

The tests that targeted individual logic apps focused mainly on testing key actions in the logic app, for example, X12 decoding, property promotion, transformation, and so on. The tests that targeted end-to-end scenarios

check the final output and other expected processing results from the message flow, for example, logging, archiving, and so on.

- Individual logic app tests

  - Input: Typically, the input for these tests would include a message (payload), any headers required by individual connectors in the logic app, or any metadata (as headers) that are required by the logic apps to make decisions.
  - Verification: These tests check the output from the main connectors used in the logic app along with any logging.

- End-to-end tests

  - Input: The input for these tests would include a message and required headers that the flow receives from an external partner or internal LOB app, depending on flow direction. The metadata is picked up from the integration account at runtime based on the input message.
  - Verification: These tests check the message and headers at the end of the end-to-end flow. Along with the final output, these tests also perform checks based on OMS logging, message archival, and so on.

All the tests are implemented using MS Test (Visual Studio Tests) and are data-driven. For a logic app, one test method is written, and a dataset is passed to the test as input. The dataset includes both positive tests and negative tests. The negative tests are a variation of the wrong input message or wrong metadata.

Typically, the data in dataset has the path to the input file and input headers, along with the data for verification, as key value pairs for logic apps. The property values to be verified depend on the input message. The dataset also has the path to the file, which includes output from the logic app.

Here's an example record from such a dataset:

```
<PAExchangeProtocolAS2V1Test>
    <Enabled>true</Enabled>
    <Name>Happy Path: Regular AS2 message, Post it to Edi Workflow Controller, and Send Mdn back to Partner</Name>
    <InputMessageFile>TestData\DataFiles\PAExchangeProtocolAS2V1\850AS2EncryptedMessage.txt</InputMessageFile>
    <Headers>AS2-From:SimpleNoMdn,AS2-To:Host,Application:VL,Message-Id:123</Headers>
    <ResponseExpected>true</ResponseExpected>
    <OutputMessageFile>TestData\DataFiles\PAExchangeProtocolAS2V1\850Message.txt</OutputMessageFile>
    <PropertiesToBeValidated>aS2Message.isMdn:False,aS2Message.mdnExpected:Expected</PropertiesToBeValidated>
</PAExchangeProtocolAS2V1Test>
```

Tracking and monitoring

Logic Apps provides rich tracking and monitoring capabilities with the Operations Management Suite (OMS) B2B solution. So, we are using these capabilities to manage tracking and monitoring for our integration service. Diagnostics data from every component is sent to OMS, and all the events corresponding to a message are correlated using a unique correlation ID (Client Tracking ID). OMS provides rich, out-of-the-box alerting capabilities based on queries. Alerts are sent as email and configured to include the query and other rich information that you can use for troubleshooting and resolution.

For monitoring Azure resources, synthetic transactions are run in the production environment. Monitoring using synthetic transactions provides a holistic view of the integration pipeline. We created a prefixed client tracking ID for synthetic transactions, so we could differentiate between real traffic and synthetic messages.

For message tracking, we used native Log Search capabilities in OMS to get an end-to-end view and message processing status. Here's an example query that uses the client tracking ID to trace the message and its processing status across all logic apps, along with the sample results shown in the following diagram:

```
Type=AzureDiagnostics (OperationName="Microsoft.Logic/workflows/workflowRunCompleted")
<unique_client_tracking_ID> | measure count() by resource_workflowName_s, status_s
```
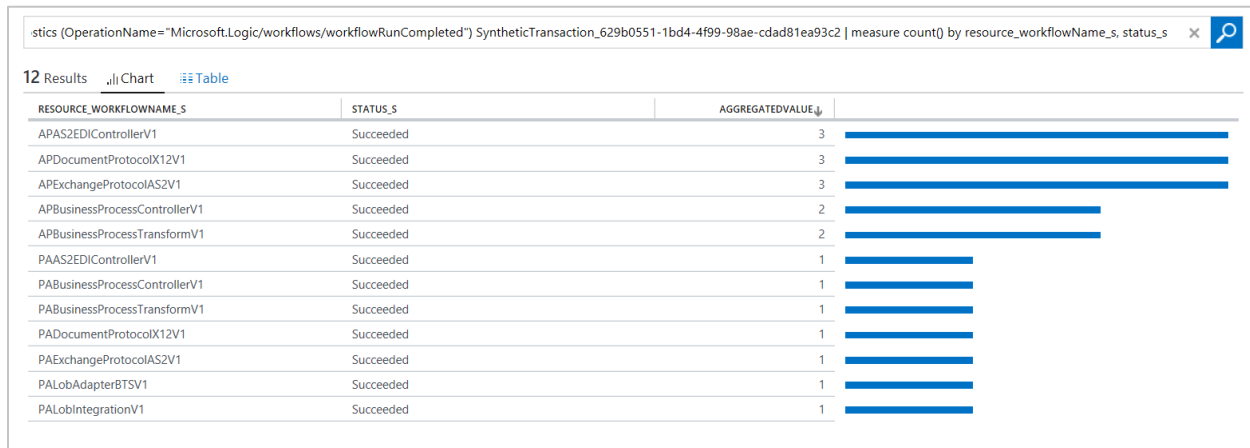

Note: If you are just starting, we suggest using Kusto Query Language (KQL), which will eventually replace OMS query language.

Scenarios using advanced capabilities

This section covers some advanced capabilities that we used in Logic Apps to build an integration service that's extensible, highly available, and manageable.

Dynamic processing using metadata

Previously, we discussed the design for logic app workflows. If you are using MABS or BizTalk Server for integration services, you'll find similarity between our separation of logic app workflows and the bridges and pipelines in MABS and BizTalk Server.

However, bridges and pipelines have a limitation – they're created for every partner and LOB app. If we applied the same principle and created a logic app for every partner and LOB app, we would have thousands of logic apps, making integration very complex to manage.
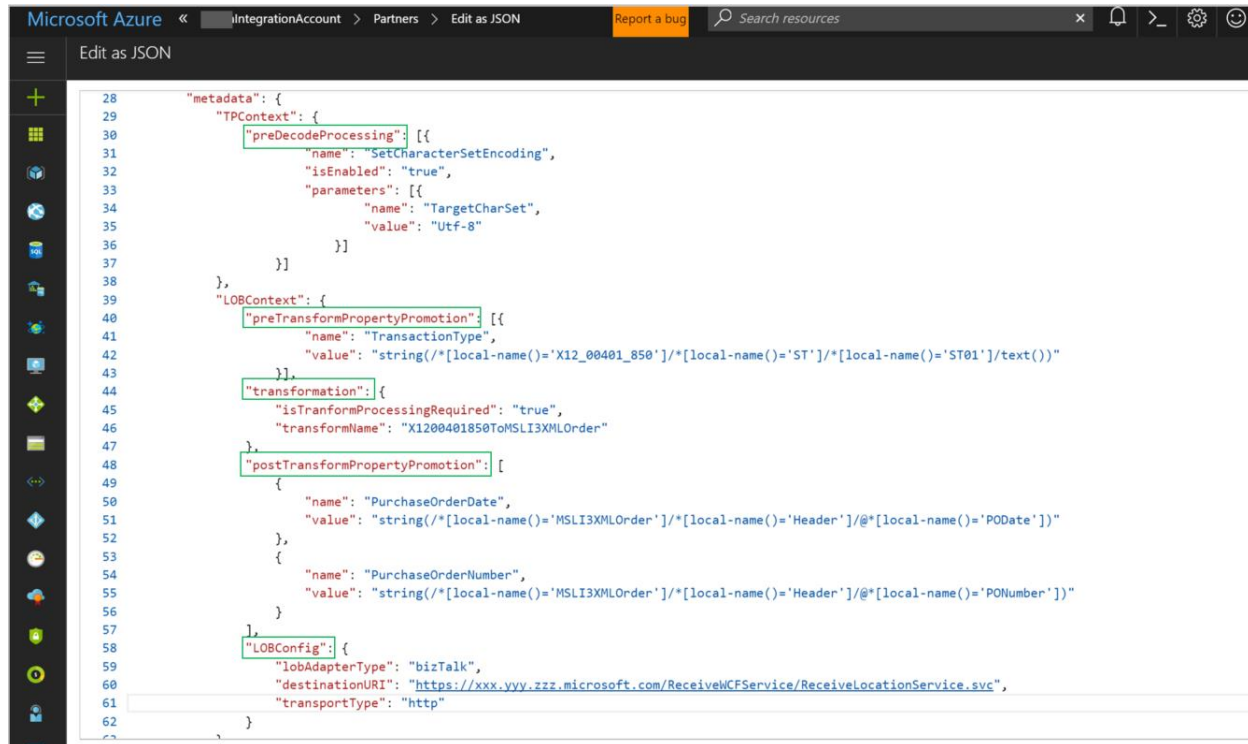
The logic app workflows that we created are aligned with B2B patterns. These workflows support EDI and non-EDI protocols like AS2, X12, and EDIFACT. These protocols are similar in nature, but have different properties for actions. The properties provided to these actions are the attributes that separate partners from each other. So, we decided to use "metadata" instead.

Logic Apps support the capability for adding metadata to partners and agreements, providing richer information. This metadata can then be looked up at runtime, used to decide the processing steps and to provide parameters for those processing steps. We created Logic app workflows for each pattern, and we used metadata to drive processing at runtime. This approach let us create reusable patterns of flows across multiple partners.

When you use metadata, you can decouple onboarding from the platform, so that onboarding primarily becomes a provisioning activity for artifacts and metadata. Also, separated workflows, along with metadata, provide a very extensible design. When you need to more patterns or protocols, you can do so easily and independently without impacting existing flows.

Metadata example

Here's an example that shows how a logic app workflow uses metadata for our service. Metadata defines the actions (green boxes), and the parameters that are passed to those actions and are used at runtime.

Custom processing

B2B integration flows might require data massaging depending on the message origin or the message recipient. For example, changing the message encoding from ANSI to Utf-8, enriching a message, and so on.

Note: Logic apps can handle Utf-8 only. Before processing. You should convert any message that comes from a partner with any other encoding to Utf-8.

These custom processing steps are not included in the standard patterns identified for EDI processing because not every message requires them. Logic app workflows have various extensibility points for performing these kinds of operations. Execution for these actions is driven by metadata, as discussed earlier. The metadata might include these details:

- The exact action that is required
- Parameters for the action
- The assembly that hosts the action (method) to execute

Custom processing is performed by a generic Azure function that receives the name of the action, the parameters required to execute the action, and the message on which the action should be performed. This function returns the processed message as output.

Based on the received metadata, the function loads an assembly and executes the action. The action's methods are kept in an assembly. This way, when an existing method is changed or a new method is added, only the assembly is updated and deployed in the Azure function, rather than changing the function itself. The function and assembly send logs to OMS for message tracking.

Hybrid connectivity

While all B2B processing logic has moved to the cloud, we still use BizTalk Server as a proxy for connecting LOB apps in the corporate network (CorpNet). This strategy provides a smoother transition to Logic Apps and minimizes the risk of impacting downstream partners.

This diagram shows the chosen approach for establishing connectivity between Azure and CorpNet after considering the network topology, security requirements, and high availability.
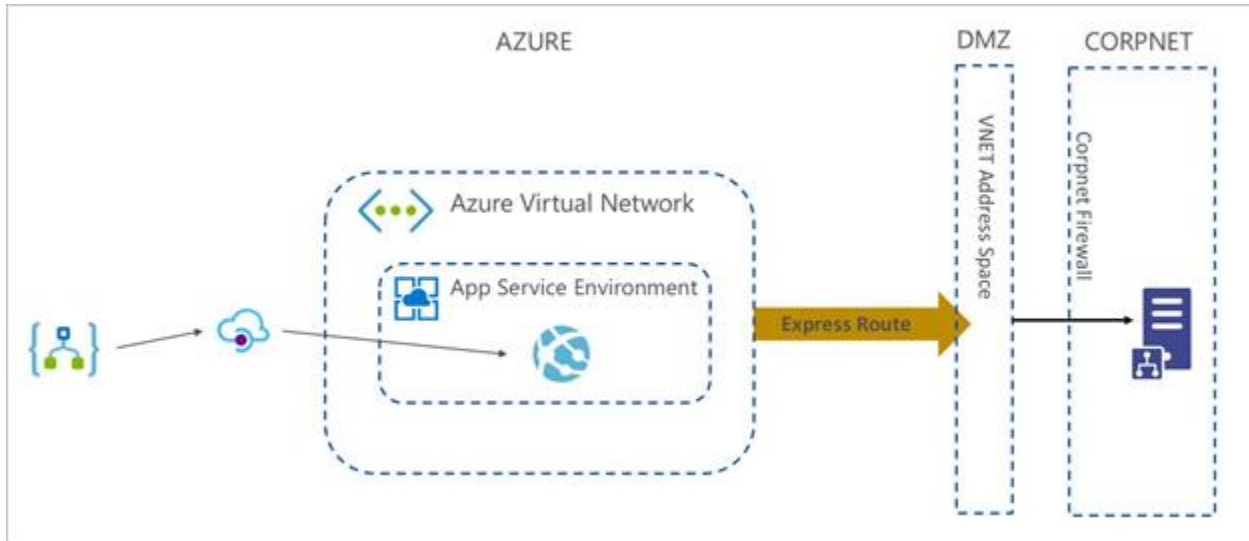
The BizTalk Server sits behind a firewall in CorpNet, so there is no direct connectivity between BizTalk Server and the cloud. So, the requests to CorpNet are routed through a Web API, which posts the request to the endpoint behind firewall in CorpNet through Azure ExpressRoute. The Web API is associated with an App Service Environment (ASE), which has visibility to CorpNet through a firewall exception.

Tracking messages in a hybrid environment

The integration service processes highly-critical business transactions, so it's important that we know where the message is at any time and its processing status. So, we set up every component in the integration pipeline to send diagnostic data to OMS. For logic apps and connectors, this happens natively. For other resources like APIM, Web API, Azure functions, and BizTalk Server, we use custom schema and this REST API that's available in the integration account:
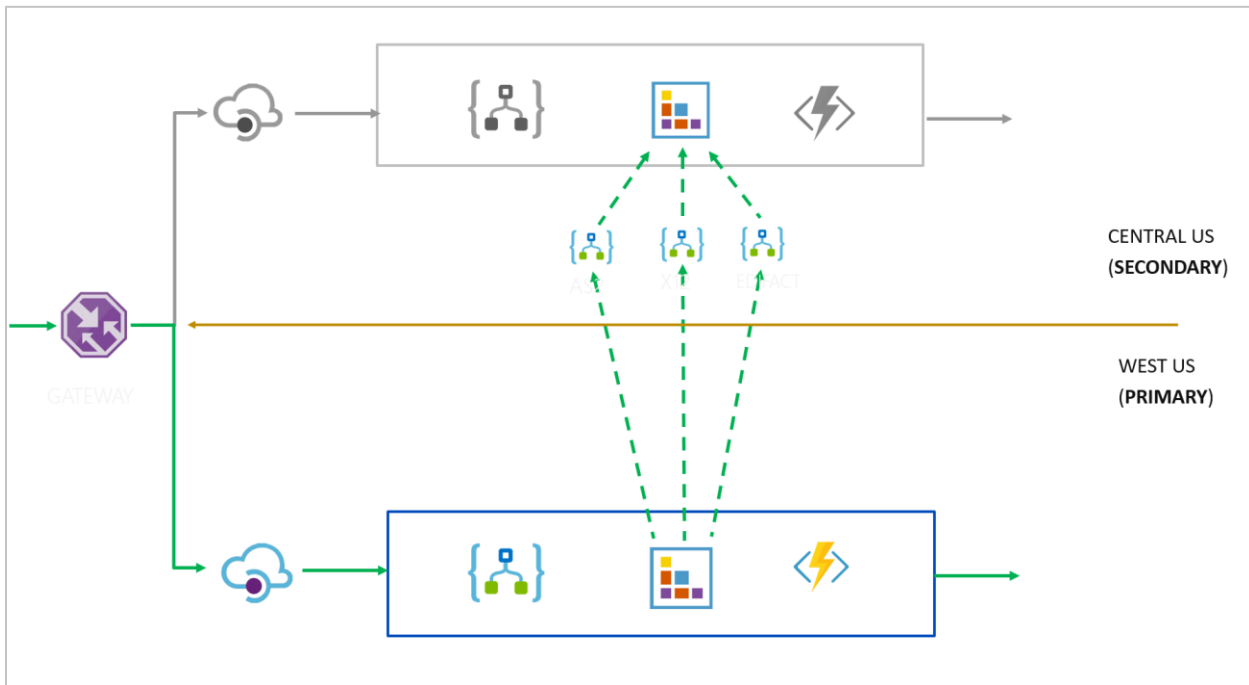
> POST - /IntegrationAccounts/{IntegrationAccountName}/trackEvents?api-verison=2016-06-01

To correlate the events across all components, we manually specify the client tracking ID from a trigger by passing an **x-ms-client-tracking-id** header with the ID value in the trigger request, which can be a request trigger, HTTP trigger, or webhook trigger.

Business continuity (cross-region disaster recovery)

For Microsoft, business continuity is important because any downtime in Azure can significantly impact our business. We have requirements for a disaster recovery mechanism to resume integration services and to meet Recovery Time Objective (RTO) and Recovery Point Objective (RPO) goals.

This diagram shows how we use integration account replication features to enable **active-passive disaster recovery** across two regions.

For B2B integration scenarios, the runtime state for stateful services or protocols (AS2, X12, and EDIFACT) is in the integration account. We have one logic app for each protocol with a recurrence trigger and an action that pulls runtime state from the primary integration account and replicates that data to the secondary integration account. So, if disaster happens, these tasks are performed:

- Change the state of all passive or disabled resources to active.
- Increase the control numbers in the secondary region by a constant because they can be out-of-sync depending on the last time that synchronization happened.
- Update Azure Traffic Manager (ATM) policies to start routing traffic to the secondary region.

Notes

- Not all Azure resources are available in all regions. Choose the primary and secondary regions where all the Azure resources for your service are available.
- Always keep both primary and secondary regions synchronized by regularly using deployment and provisioning for your regions. That way, you can ensure a smooth transition from your primary region to your secondary region.

Managing messages and workflows

Logic Apps provides these capabilities for managing messages in workflows.

Resubmit messages and failures in bulk

Due to high throughput scenarios, the impact of problems on an integration service is substantial. Many times, the failures are intermittent due to transmission or unavailable resources, or they are recoverable due to configuration issues. In these scenarios, we'd want to resubmit messages from the point of failure. Also, it's more common practice to resubmit all the failures in bulk. Today, we perform this task through custom code, which enumerates all the logic app runs that need to be resubmitted and calls this REST API:

```
POST workflows/{workflowName}/triggers/{triggerName}/histories/{historyName}/resubmit
```

Note: Currently, Logic Apps doesn't have the concept of suspend or pause like MABS and BizTalk Server. However, you can disable logic app runs and after failure, access messages through the run history. Together, these capabilities help simulate some suspend or pause requirements.

Store and forward messages

We wanted to provide a highly available and reliable integration service to our partners. So as soon as our internal gateway gets a request from a trading partner, we archive that request in a custom store. This approach lets us:

- Reprocess the message if the integration service experiences any failures from which we can recover.
- Retain and reprocess the message in the secondary region, if necessary in case of a disaster.
- Control throughput should traffic bursts or processing delays happen.

Check for duplicate messages

Logic Apps guarantees to deliver a message at least once, but in rare scenarios, Logic Apps might deliver duplicate messages. Duplicate message delivery is unacceptable, particularly for upstream and downstream partners, because duplicates might result in monetary losses along with other inconsistencies. While EDI connectors, like X12 and EDIFACT, natively check for duplicates, we built a custom component to check for duplicates and prevent duplicate processing at other steps in the pipeline.

Our duplicate check is based on a checksum that's calculated using the payload. We store the checksum in table storage and use that value for comparison to identify duplicates. If a duplicate exists, message processing is stopped, and an alert is generated.

We developed this component using Azure Functions, and we can plug the component into any workflow. Usually, we apply this check at the boundaries, before the message leaves our integration service.

Conclusion

After we successfully moved our integration footprint from MABS to Logic Apps, we're now ready to retire our MABS instance. As part of our migration, we also enabled these transactions and businesses:

- Order to cash flows for digital supply chain
- Trade integrations and all Custom Declarations transactions

We've now finished describing how we used the Logic Apps platform to build a B2B processing service in Azure. However, we've only just started our integration journey, so we'll continue to work on increasing our integration footprint in PaaS.