

Session

The way of the exploding workflow

Steef-Jan Wiggers

Azure Technology Consultant



intelligent
cloud conference

Who am I?



codit | Azure Technology Consultant



Microsoft MVP – Azure



InfoQ Cloud Editor



WAZUG board member



Azure Lowlands Organizer

Way of the exploding fist



iPaaS - Market

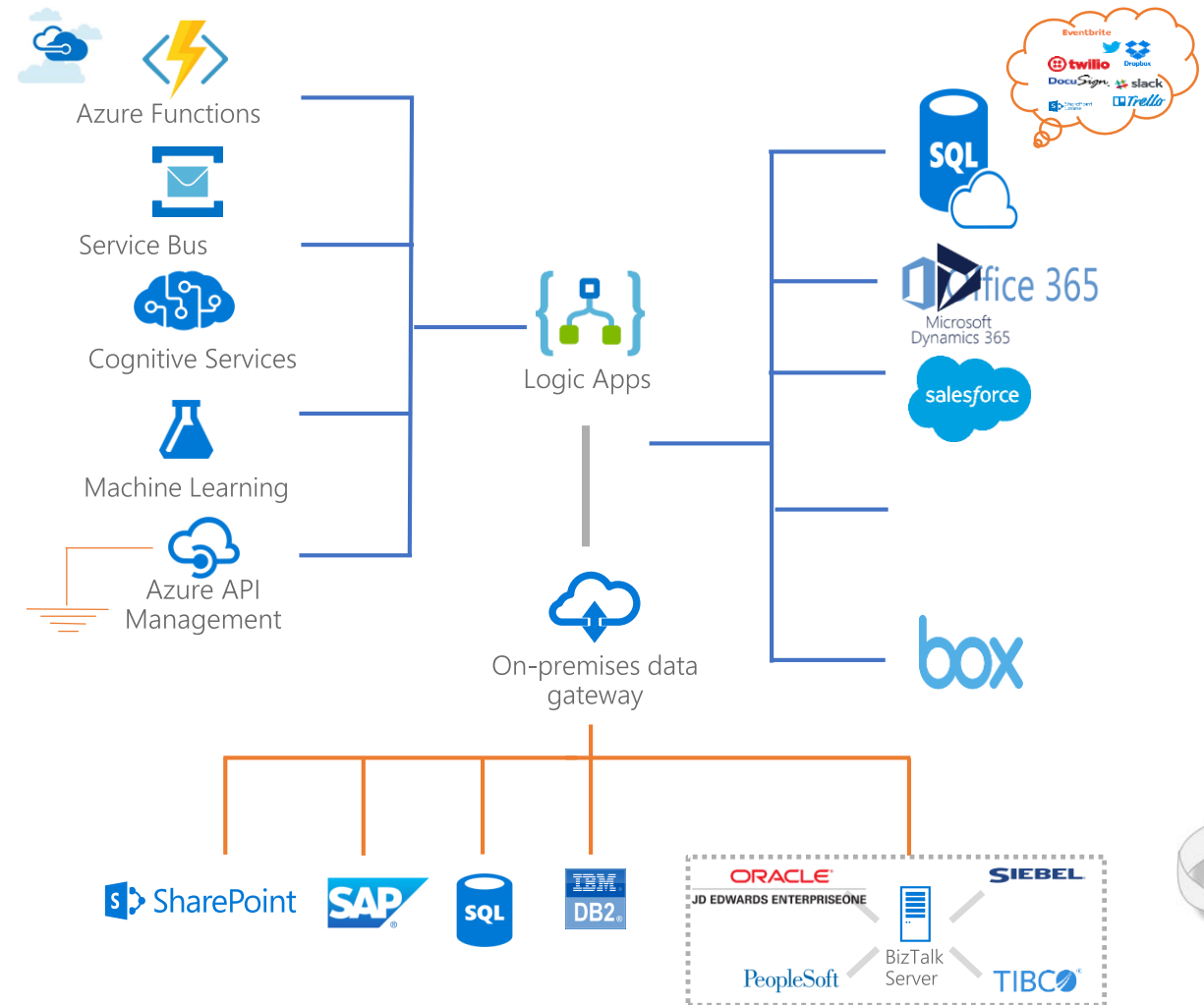


- Solving the business problem first
- Fit for purpose for cloud integration
- Less cost, fast time to market

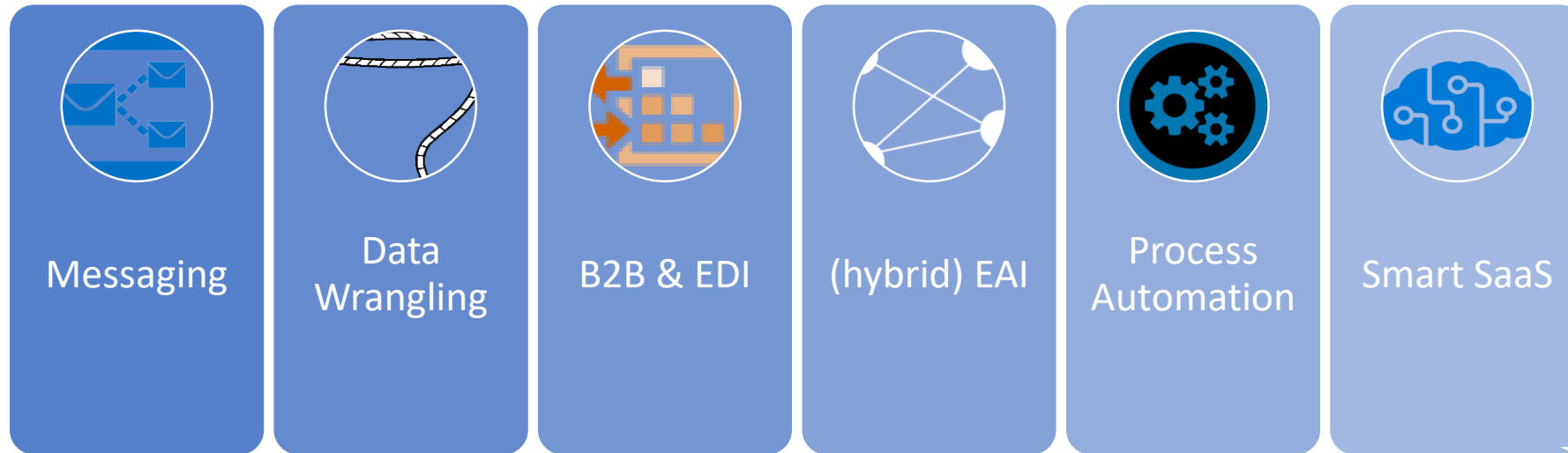


What are Logic Apps?

- Connect to on-premises, hybrid, and cloud applications
- Run mission critical, complex integration scenarios with ease
- Build smart integrations leveraging machine learning, cognitive services



The service supports various scenario's



Serverless



Event-driven
scale



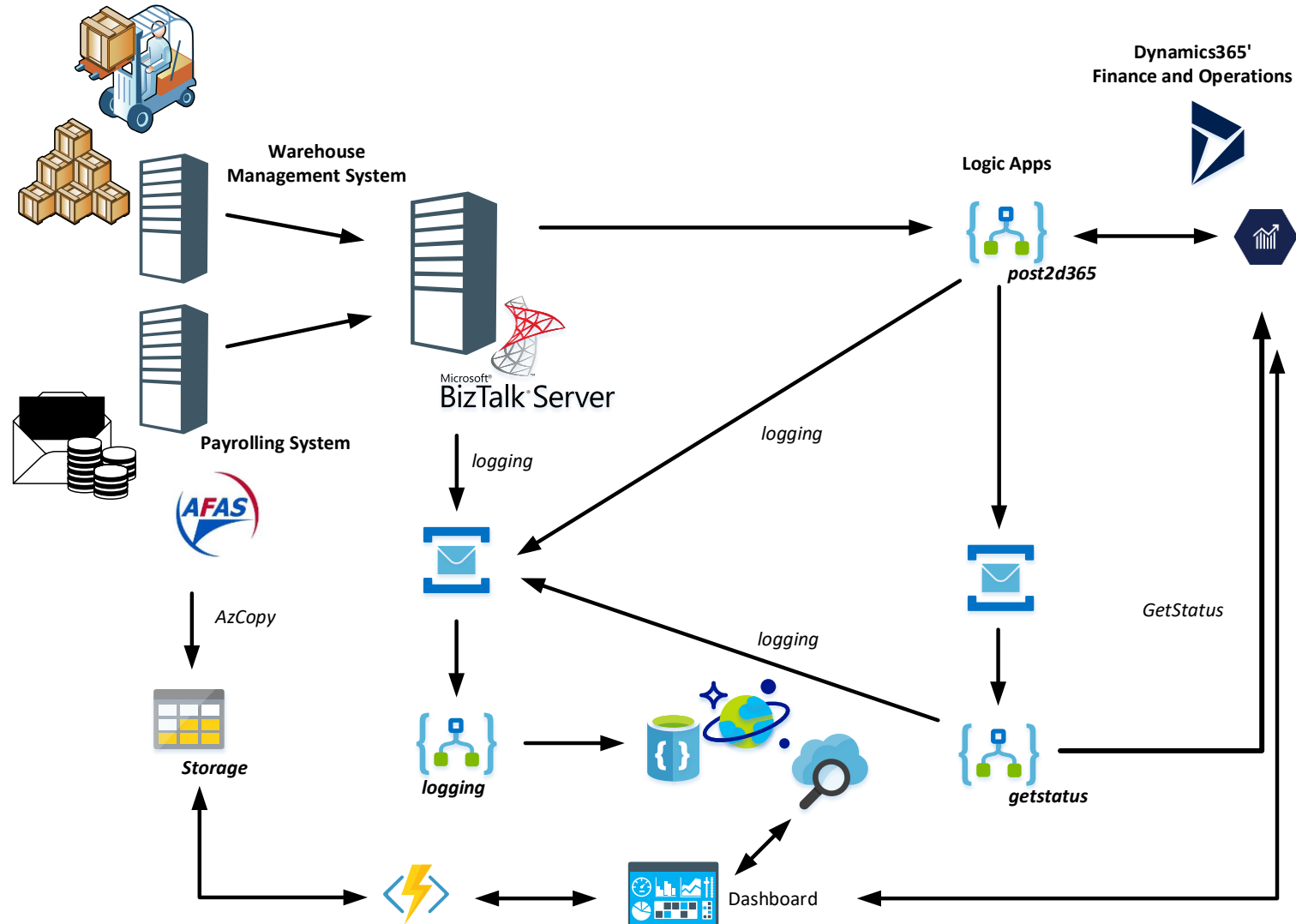
Abstraction of
servers



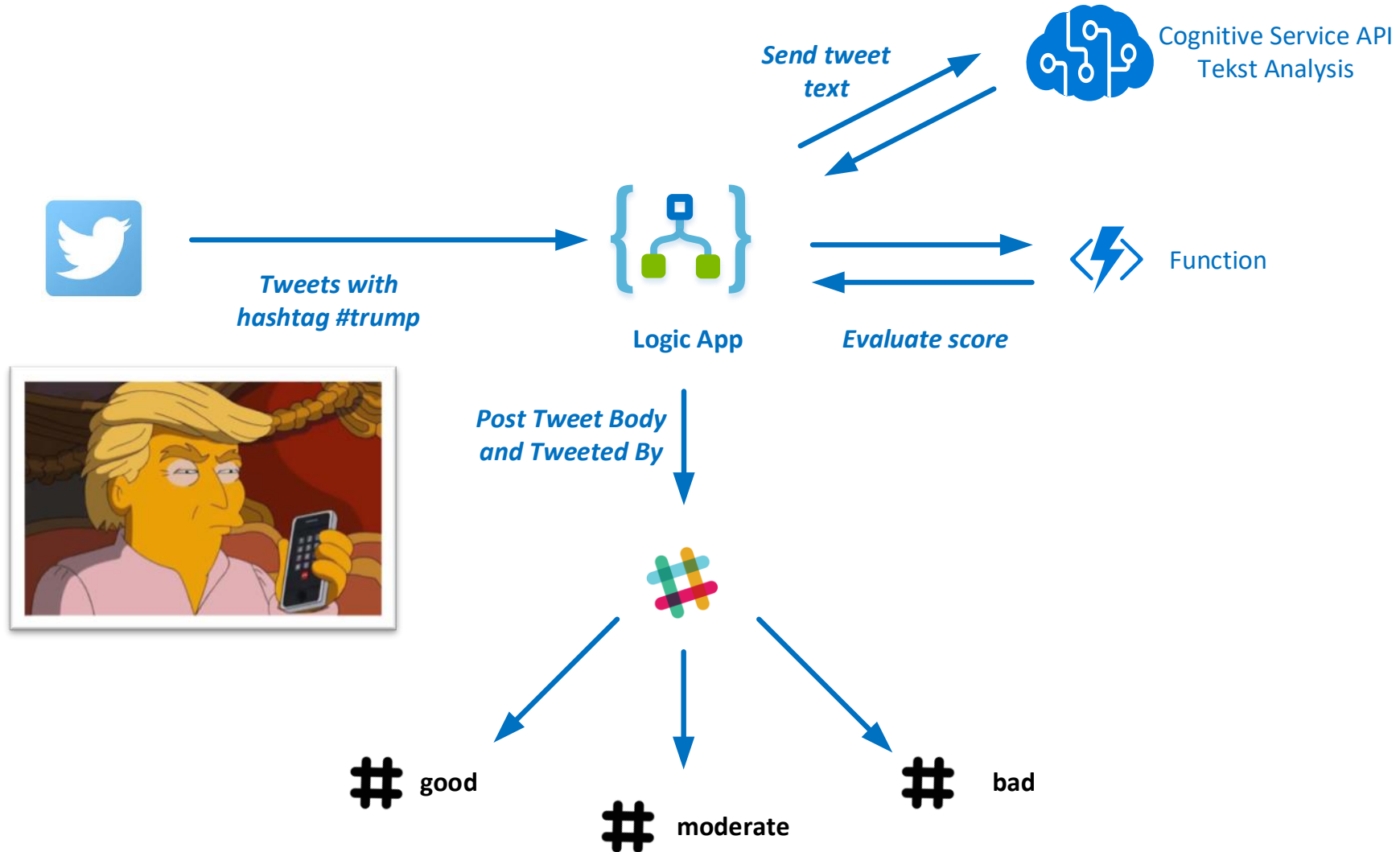
Micro-billing



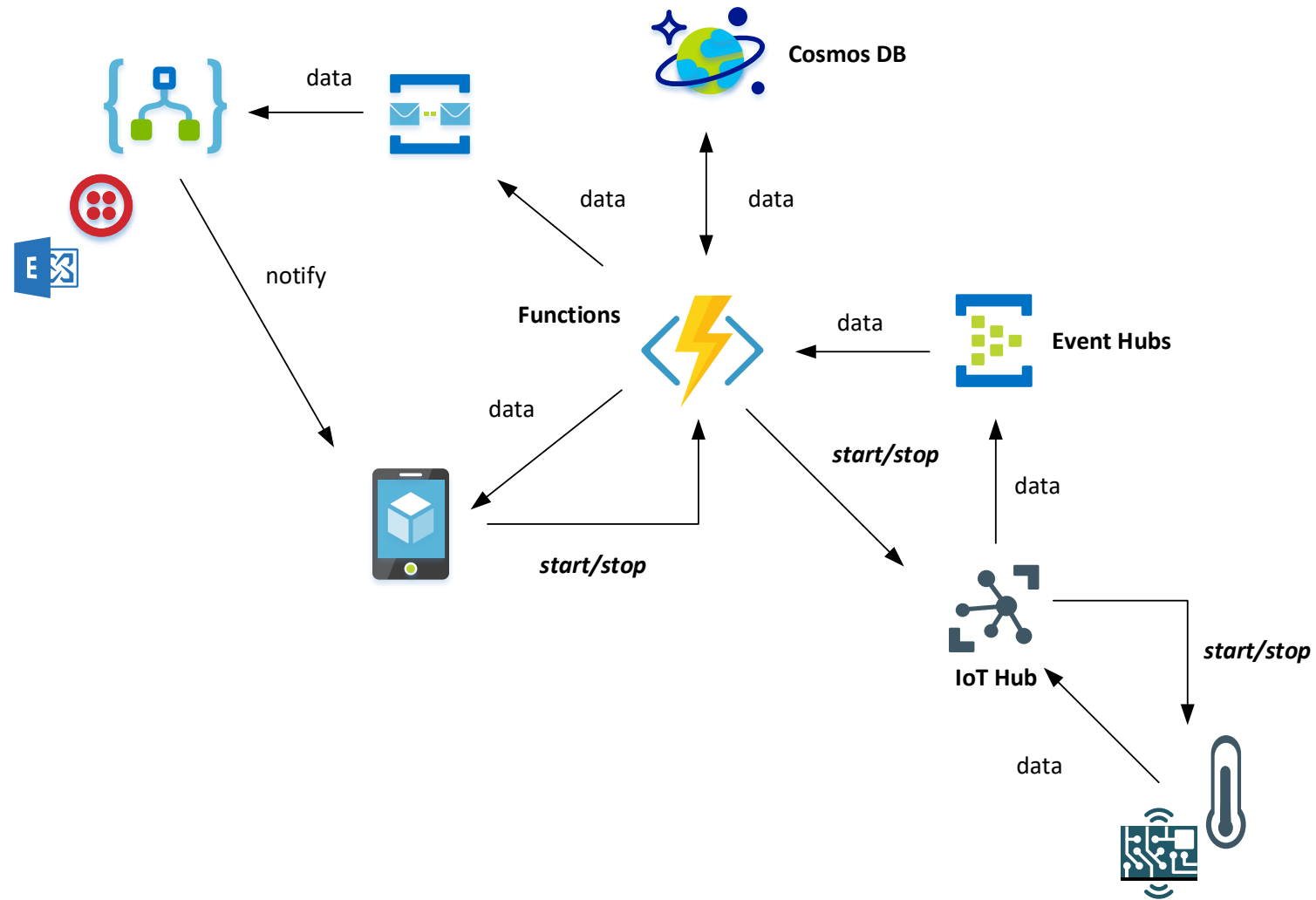
Scenario - EAI



Scenario – Smart SaaS



Demo – Messaging (Send Notification)



Connectors

Cloud APIs and platform functionality

- Over 200 out of box connectors
 - SaaS, on-prem, protocols, B2B and message manipulation
- Hybrid connectivity
- Hosted and managed within the platform
- Scales to meet your needs
- First class designer experience



Custom Connectors

- Access any REST/SOAP API
- Cloud or on-premises
- Simple creation wizard
- Connections and managed secrets
- First class designer experience

API connections

- Authenticate once and reuse
- Differentiate connection configuration
- Simple to deploy
- Portal experience for managing API Connections



Triggers

Creates
new
instances
of Logic
Apps

Recurrence/advanced scheduling

Polling

Webhook

Request



Actions

Invoke services

Managed Connectors

App Service APIs

API Management

Azure Functions

Workflow

HTTP + Swagger

HTTP

Control behaviour

Retry Policy

Run After

Limit

Response

Webhook

Batch

Wait

Terminate

Message Handling

Compose

Query

Table

Request schema

Parse JSON

Xpath

XSLT

XML validation

Expression conversion

Flow Control

Scope

Condition

Switch
Case

For Each

Until

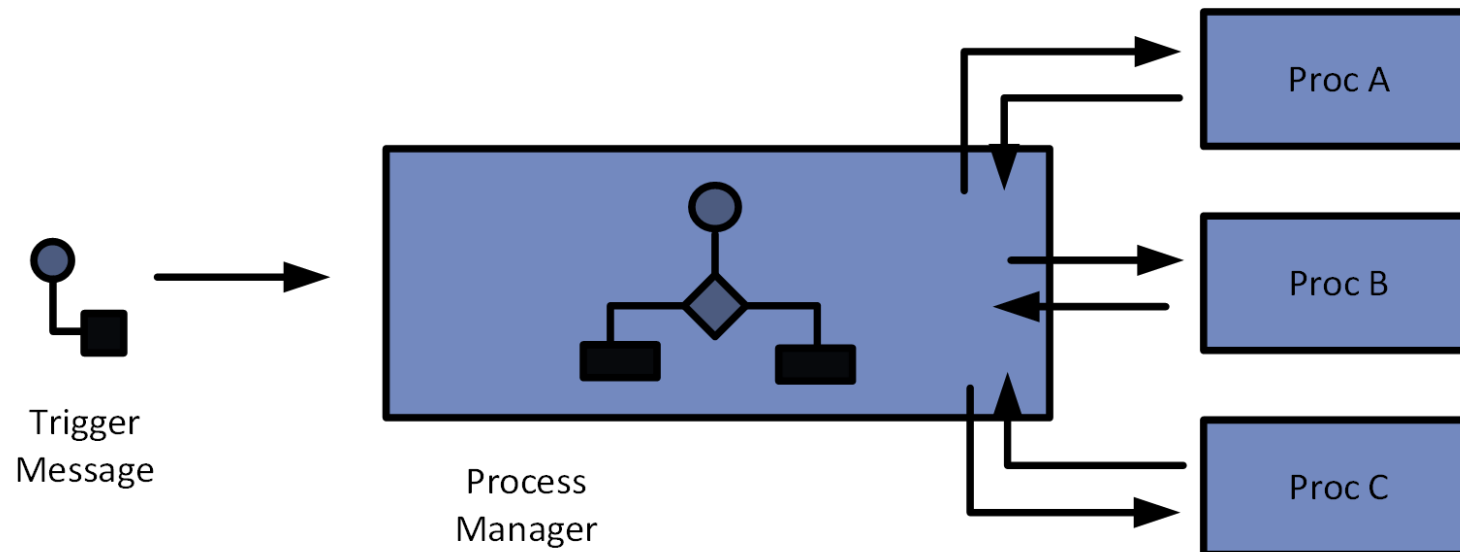


Patterns

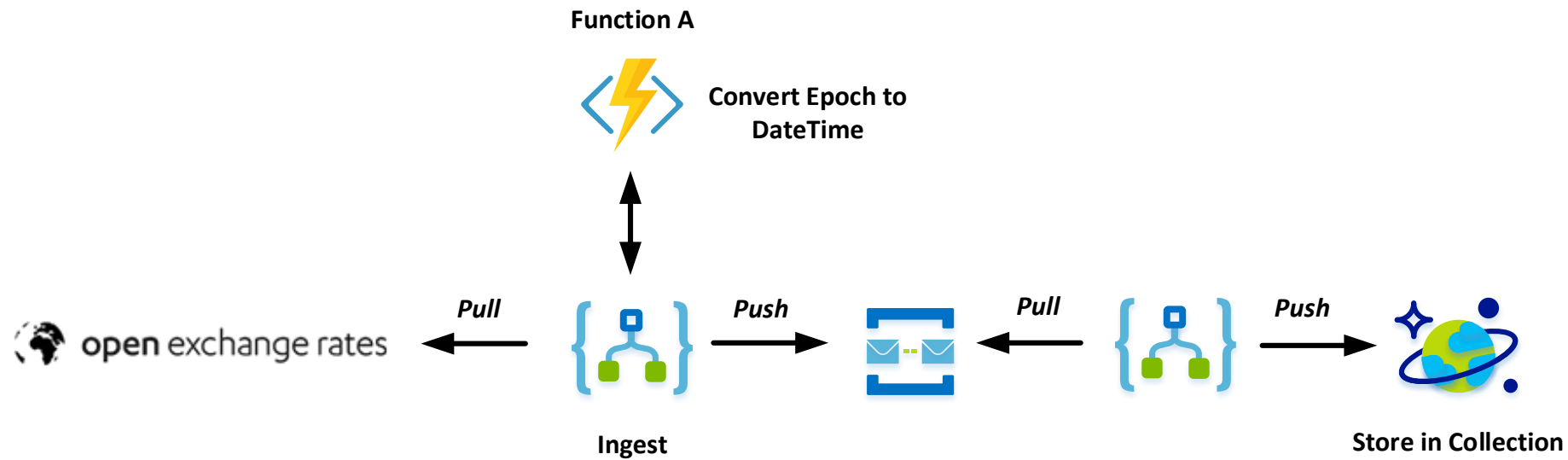
- Various patterns possible ranging from request-reply to process manager

Process Manager

Central processing unit, determine steps based on intermediate result

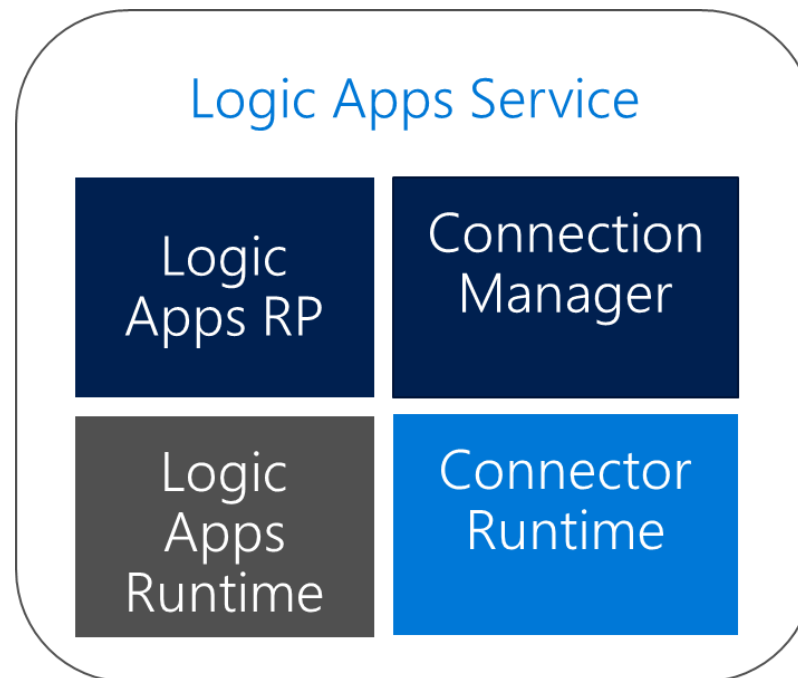


Demo – Process Manager



Runtime - Logic App Workflow Engine

- Logic Apps is a job scheduler with a JSON-based DSL describing a dependency graph of actions
- Highly parallelized concurrent job execution engine



Workflow Definition

Save

When a message is received in a queue (peek-lock)

For each

* Select an output from previous steps

ContentData.LineItem x

Insert row

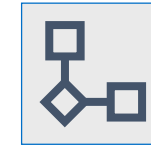
Add an action

Add a condition

More

Complete the message in a queue

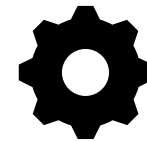
Trigger Task



Service Bus

Run

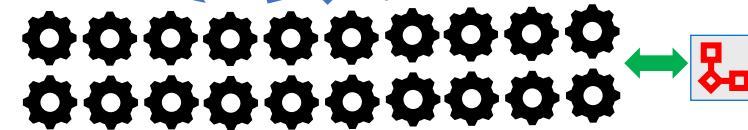
On new message



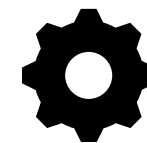
Workflow Orchestrator

Workflow Complete

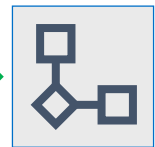
ForEach



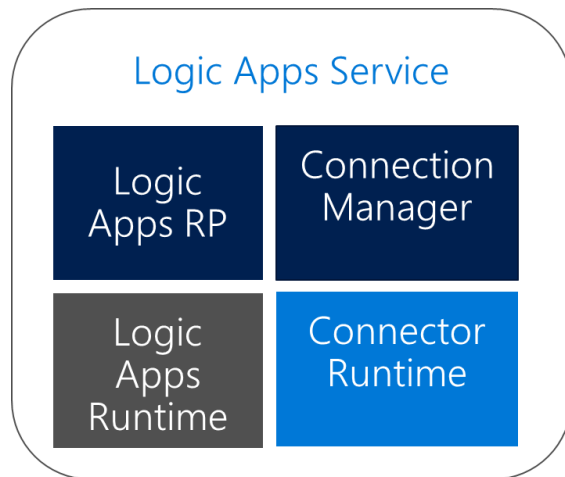
SQL Insert



Complete Message



Component Architecture



- **Logic Apps RP**

Reads the workflow definition and breaks down into a composition of tasks with dependencies

- **Logic Apps Runtime**

Distributed compute/workers are coordinated to complete tasks on-demand

- **Connection Manager**

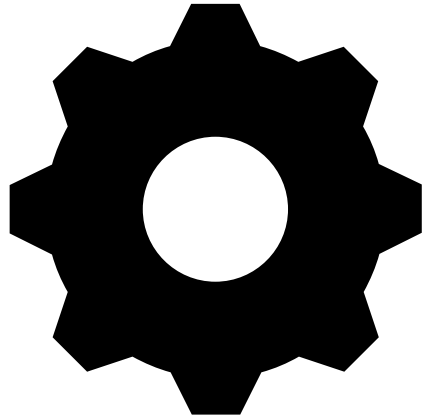
Manages connection configuration, credentials and token refreshment

- **Connector Runtime**

API abstraction via Open API descriptions



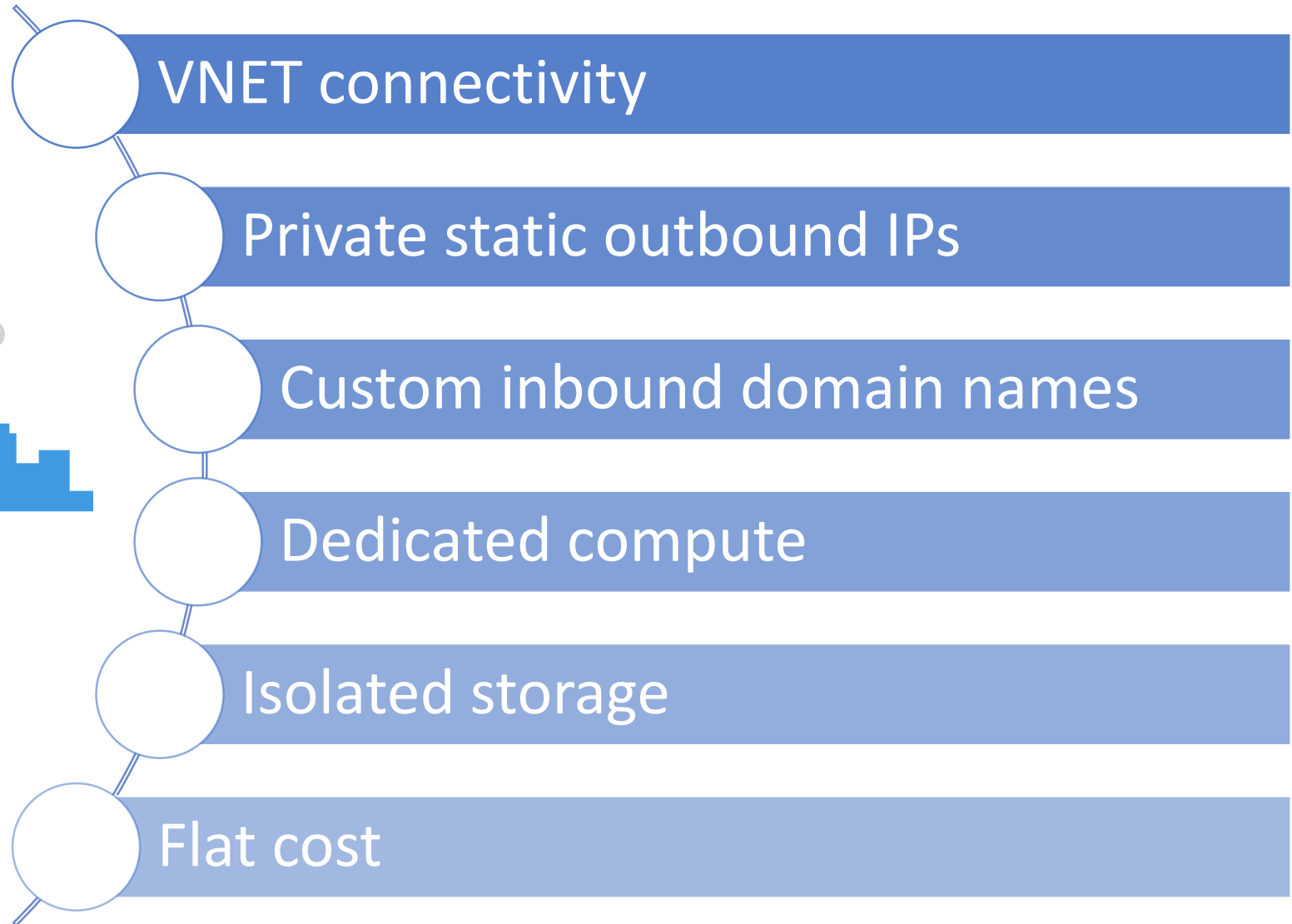
Task Resiliency



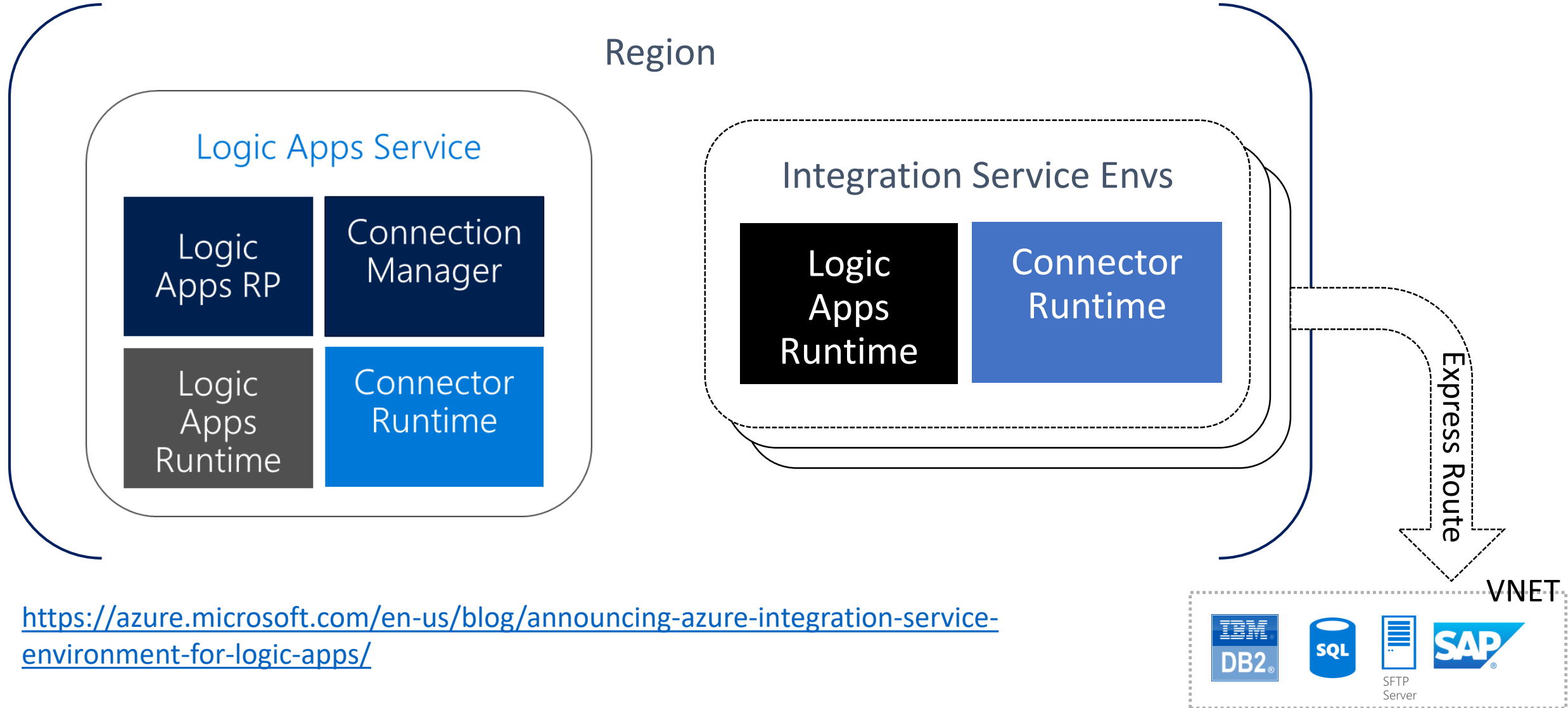
- No active thread management – tasks and runs can exist in parallel and at massive scale
- At least once guaranteed execution
- Transient failures invoke retry-policies (DNS issues, throttles, or 5xx responses)
- If the task doesn't respond, workflow orchestrator will assign a new task (at least once guarantee)



Integration Service Environments (ISE)

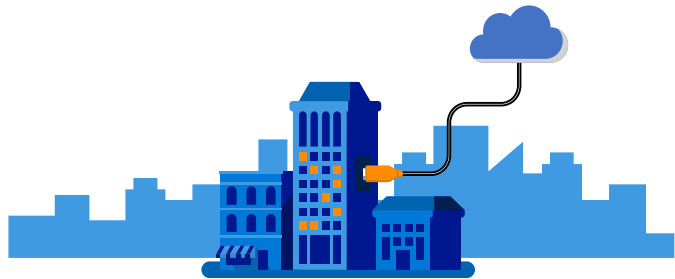


ISE Architecture



<https://azure.microsoft.com/en-us/blog/announcing-azure-integration-service-environment-for-logic-apps/>

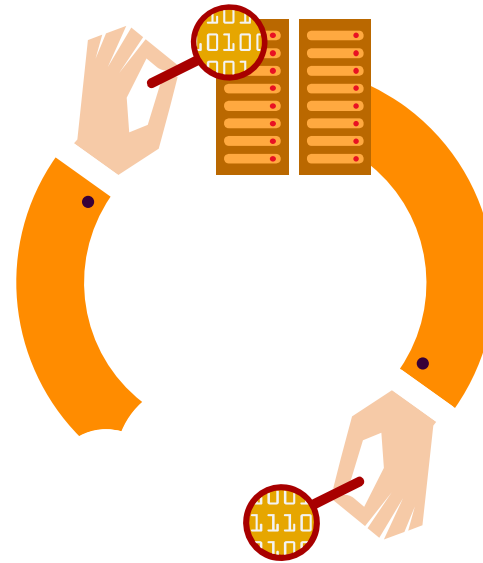
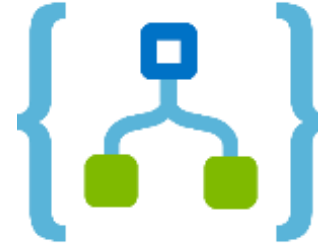
Deployment model



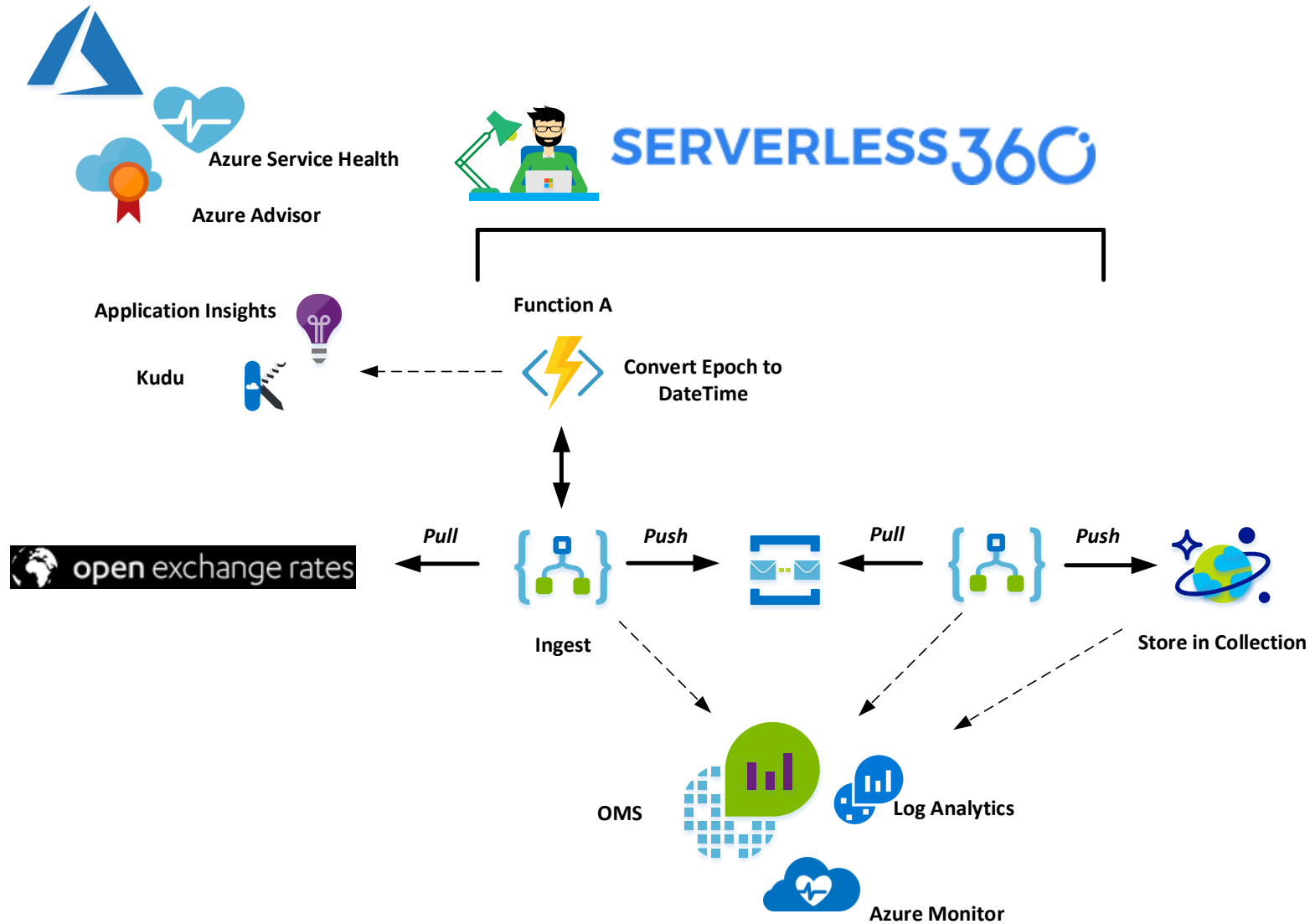
- Base Unit:
 - 75 M action executions / month
 - 1 standard integration account
 - 1 enterprise connector
 - Includes unlimited connections
 - VNET connectivity
- Each additional processing unit:
 - Additional 50M executions / month

Monitoring

- Trigger and run history
- Monitoring View
- Diagnostics
- Alerts
- Tracked Properties
- Tracking API
- Operation Management



Demo - Monitoring



DevOps

Visual Studio



Resource Group Project				
ARM Template	Edit in Designer	Deploy	Source Control	CI/CD

Cloud Explorer				
Browse Azure	Edit in Designer	Manage	Execute & Monitor	Download



Demo - DevOps

Pipeline Variables History | [+ Deploy](#) [Cancel](#) [Refresh](#) [Edit](#) ...

Release

Manually triggered

by  Steef-Jan Wiggers
3/6/2019, 9:27 AM

Artifacts




_Logic App D365 Post2...

329

 master

Stages

TST

 Succeeded

on 3/6/2019, 9:28 AM

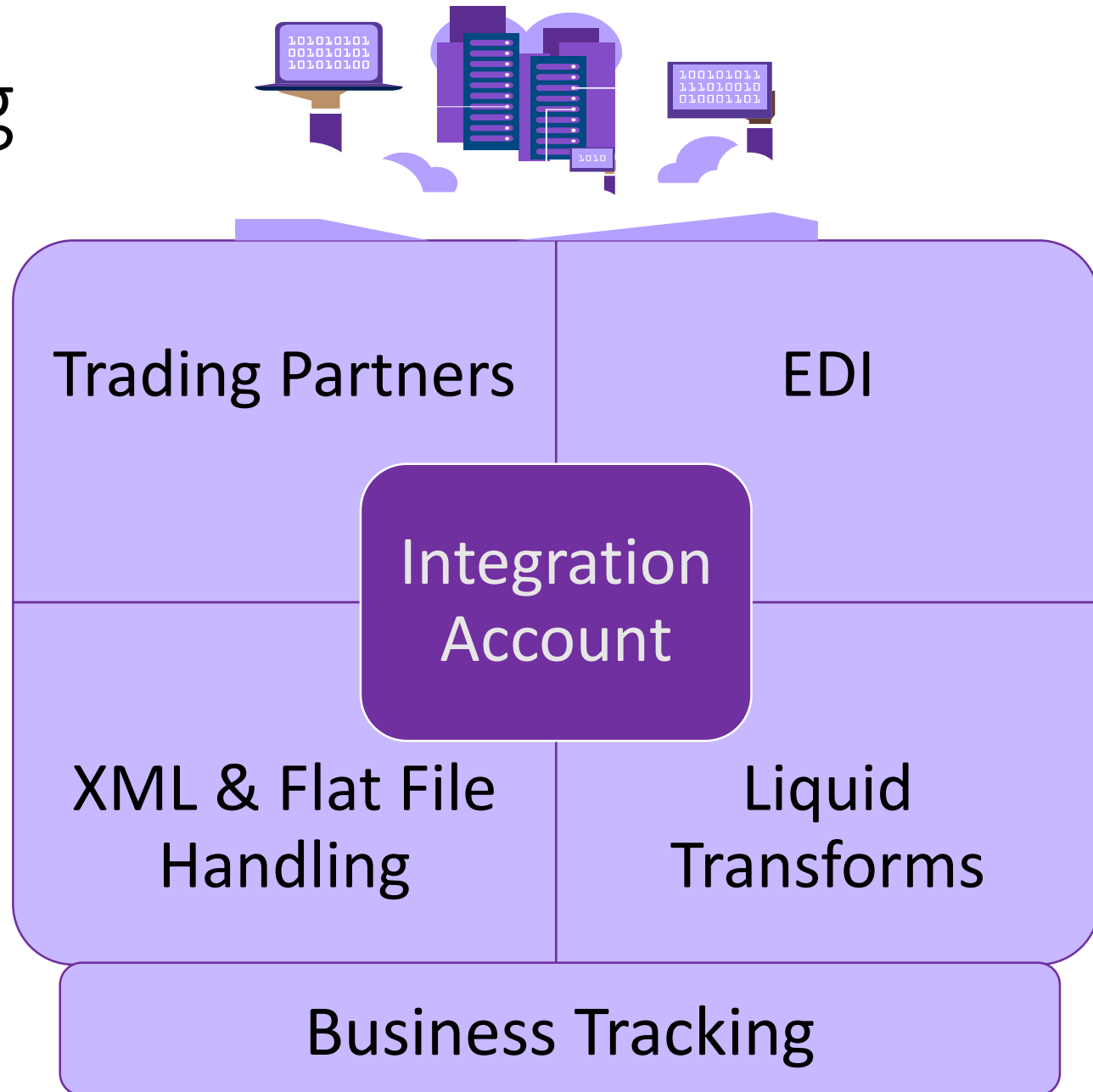


Tips – Mapping options

- XML and JSON mappings
- Integration Account
- Alternative is Azure Functions



B2B Messaging

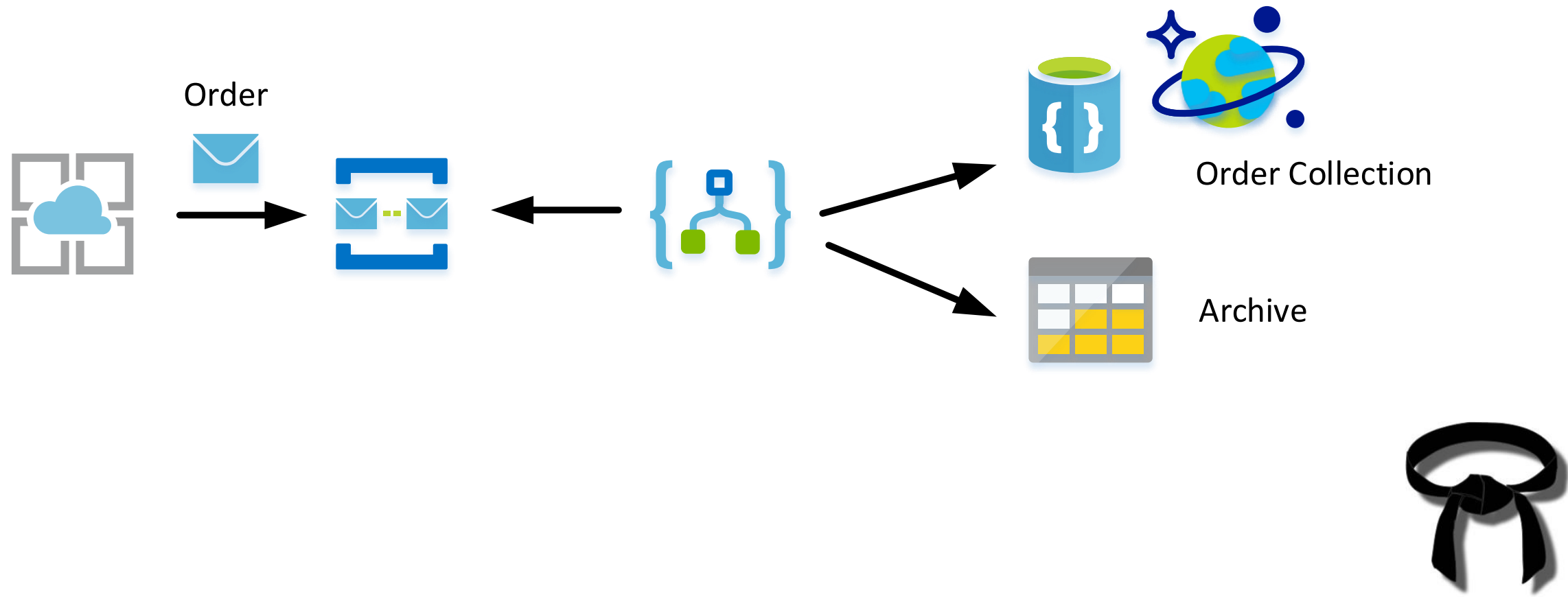


Tip – Function expressions

- Perform actions in Runtime:
 - - String Functions
 - - Collection Functions
 - - Logical Comparison
 - - Conversion Functions
 - - Math functions
 - - Date and time functions
 - - Workflow functions
 - - URI Parsing functions
 - - Manipulation functions: JSON and XML
- Source: <https://docs.microsoft.com/en-us/azure/logic-apps/workflow-definition-language-functions-reference>



Scenario



Conversions

The screenshot displays the Azure Logic Apps editor interface. On the left, a workflow is shown with two steps: 'Initialize variable' and 'Parse JSON'. The 'Initialize variable' step is highlighted with a purple border. It has the following configuration:

- Name:** OrderMessage
- Type:** String
- Value:** `json(...)` (with a red box around the `json` function name)

Below the steps is a '+ New step' button. On the right, a 'Dynamic content' pane is open, showing a list of functions under the 'Expression' tab. The first function, `json(xml)`, is highlighted with a red box. Below this, several other functions are listed under categories like 'String functions', 'Collection', and 'Logical functions'.

Conversions to other types, data or format. For instance, *base64ToString*.

When working with JSON objects and XML node you can use functions like *addProperty*, *xml*, *json* and *coalesce*.



Conversions - continued

- Manipulating string such as *concat*, *split*, *replace*, and *substring*

The screenshot shows the 'Compose' step editor in Azure Logic Apps. The JSON body is as follows:

```
{
  "customerfullName": fx concat(...) x ,
  "id": { } customerid x ,
  "orderdateUTC": fx utcNow() x ,
  "orderdatetime": { } date x ,
  "orderid": { } orderid x ,
  "price": fx int(...) x ,
  "product": { } productname x ,
  "productid": { } productid x ,
  "quantity": fx int(...) x ,
  "totalprice": fx mul(...) x
}
```

Below the JSON body, there is a section for 'Dynamic content' and 'Expression'. The 'Expression' tab is selected, and the following expression is entered:

```
fx t(body('Parse_JSON')?['order']?['customerf
```

The 'Update' button is visible at the bottom of the expression editor.

`concat(body('Parse_JSON')?['order']?['customerfirstname'],body('Parse_JSON')?['order']?['customerlastname'])`



Conversions - continued

- To manipulate date and time data types: *utcNow*, *addDays*, *addSeconds*, or *ConvertTimeZone*.

The screenshot displays the 'Compose' step editor in Azure Logic Apps. The main area shows a JSON payload with the following fields and expressions:

```
{
  "customerfullName": fx concat(...) x,
  "id": {} customerid x,
  "orderdateUTC": fx utcNow() x,
  "orderdatetime": {} date x,
  "orderid": {} orderid x,
  "price": fx int(...) x,
  "product": {} productname x,
  "productid": {} productid x,
  "quantity": fx int(...) x,
  "totalprice": fx mul(...) x
}
```

The 'orderdateUTC' field is highlighted with a red box. The sidebar on the right shows the 'Expression' tab with 'fx utcNow()' selected and highlighted with a red box. Below the sidebar, a 'String functions' section shows the 'concat' function:

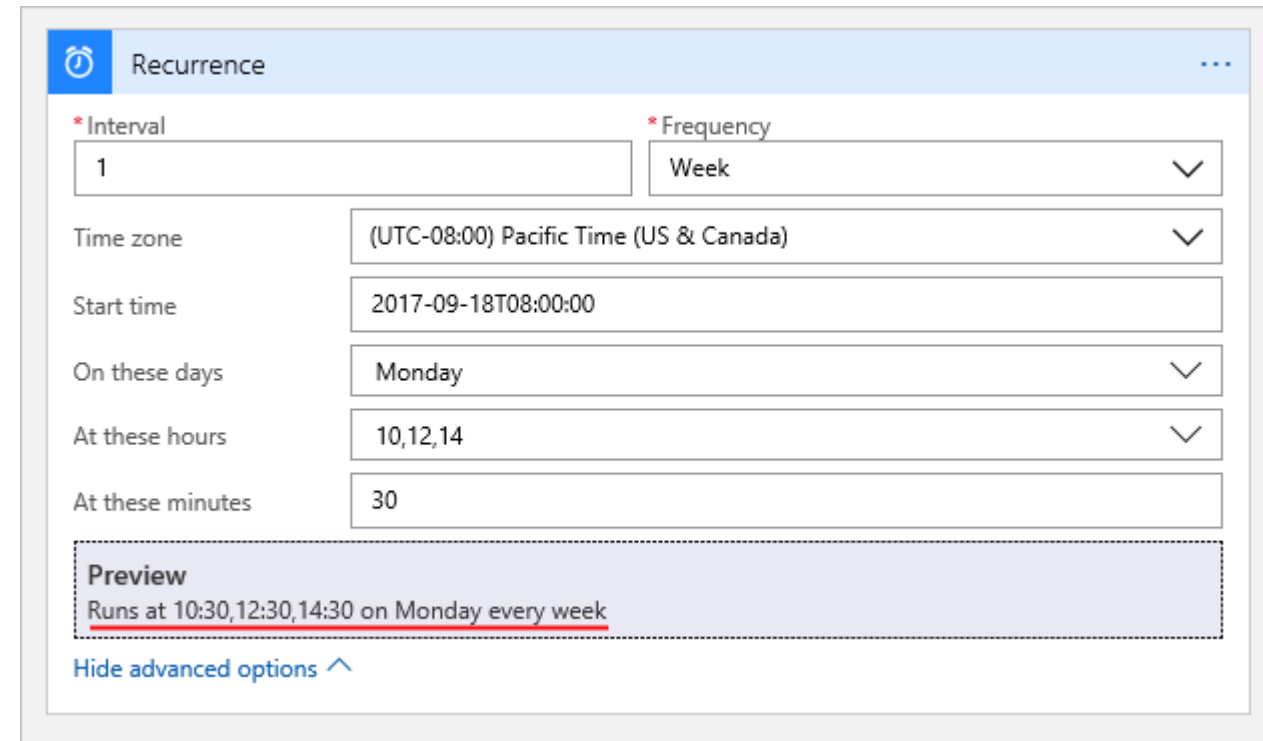
```
fx concat(text_1, text_2?, ...)
```

Combines any number of strings together



Tip – Schedule Tasks

```
{
  "triggers": {
    "Recurrence": {
      "type": "Recurrence",
      "recurrence": {
        "frequency": "Week",
        "interval": 1,
        "schedule": {
          "hours": [
            10,
            12,
            14
          ],
          "minutes": [
            30
          ],
          "weekDays": [
            "Monday"
          ]
        },
        "startTime": "2017-09-07T14:00:00",
        "timeZone": "Pacific Standard Time"
      }
    }
  }
}
```



The screenshot shows the 'Recurrence' configuration pane in Azure Logic Apps. It includes fields for Interval (1), Frequency (Week), Time zone ((UTC-08:00) Pacific Time (US & Canada)), Start time (2017-09-18T08:00:00), On these days (Monday), At these hours (10,12,14), and At these minutes (30). A Preview section at the bottom states: 'Runs at 10:30,12:30,14:30 on Monday every week'. A link 'Hide advanced options' is also visible.

* Interval	* Frequency
1	Week

Time zone	(UTC-08:00) Pacific Time (US & Canada)
Start time	2017-09-18T08:00:00
On these days	Monday
At these hours	10,12,14
At these minutes	30

Preview

Runs at 10:30,12:30,14:30 on Monday every week

[Hide advanced options](#)

Note: Azure Logic Apps is replacing Azure Scheduler, which is being retired. To schedule jobs, try Azure Logic Apps instead.



Tips – Making Logic Apps Robust

- Retries
- Scopes
- Run After



Retry policies

- Retry policies can become active once an action fails. They ensure that the action is retried automatically, before the Logic App ends in a failed state.
- Logic Apps only retries transient errors
 - If HTTP status code equals:
 - HTTP 408
 - HTTP 429
 - HTTP 5XX
 - If HTTP times out

Settings for 'Send message'

Asynchronous Pattern
With the asynchronous pattern, if the remote server indicates that the request is accepted for processing with a 202 (Accepted) response, the Logic Apps engine will keep polling the URL specified in the response's location header until reaching a terminal state.

Asynchronous Pattern ☒ On

Automatic decompression
Automatically decompress gzip response.

Automatic decompression ☒ On

Timeout
Limit the maximum duration an asynchronous pattern may take. Note: this does not alter the request timeout of a single request.

Duration ⓘ

Retry Policy
A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. The default is an exponential interval policy set to retry 4 times.

Type

Tracked Properties
Properties



Retry policies

- Default:
 - Retry 4 times, with 20 sec interval
- None:
 - No retries, fail immediately
- Exponential:
 - Configure exponential back off retry
- Fixed Interval:
 - Configure retry count and interval
 - Duration in [ISO 8601 format](#)

Retry Policy
A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. The default is an exponential interval policy set to retry 4 times.

Type

Tracked Properties Properties

- Default
- None
- Exponential Interval
- Fixed Interval



Retry policies

Think about retry policies!

- Only a short retry cycle is accepted for *synchronous* flows
- *Asynchronous* flows are better off with a longer retry policy

Retry Policy

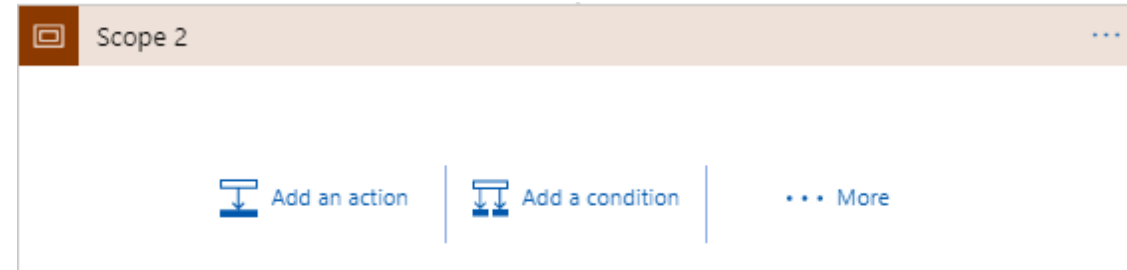
A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. The default is an exponential interval policy set to retry 4 times.

Type	Exponential Interval
*Count	Specify a retry count from 1 to 90
*Interval ⓘ	Example: PT20S
Minimum Interval ⓘ	Example: PT10S
Maximum Interval ⓘ	Example: PT1H



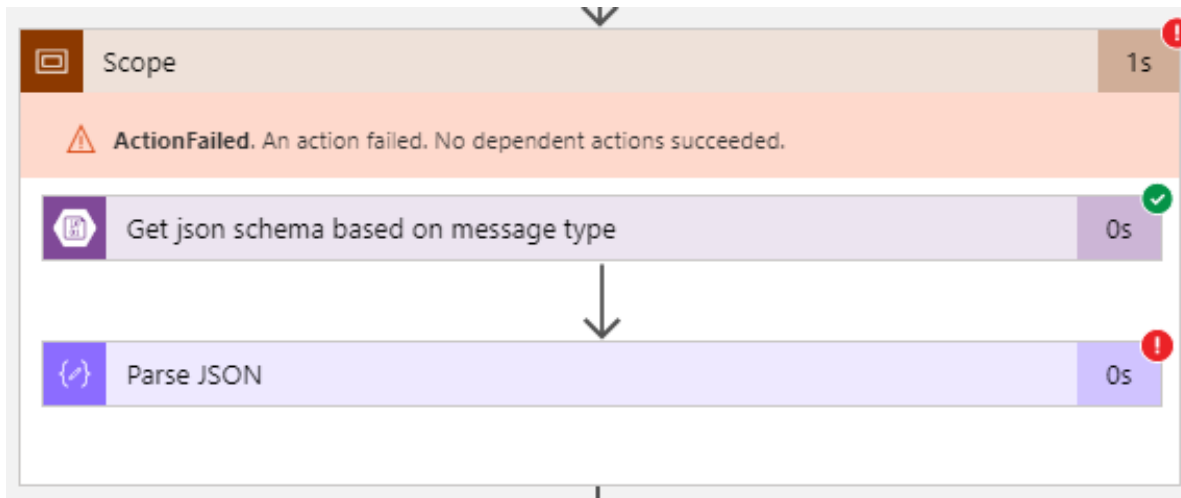
Scopes

- Scopes logically group several actions together. The scope result contains a detailed execution outcome of the actions inside the scope.
- A scope allows you to change the workflow if any of the grouped actions fails.



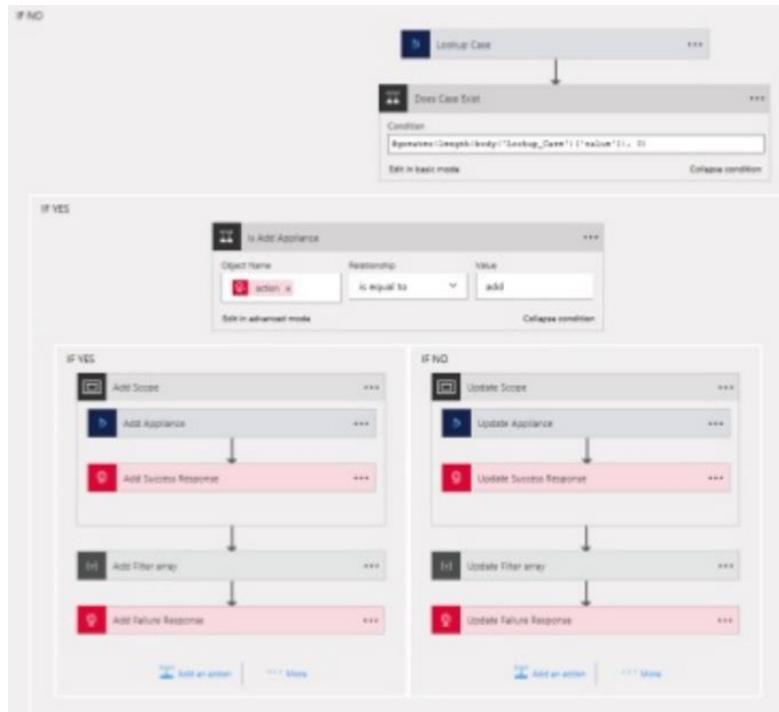
Status of a scope

- Status is set after all actions are finished
- If final action in branch is failed → scope failed

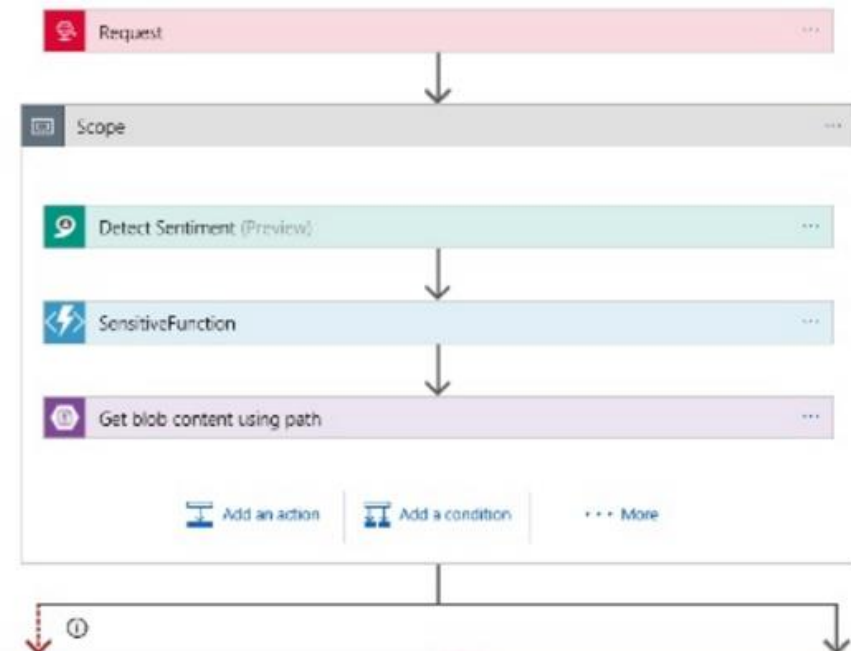


Purpose of a scope

- Logical grouping
 - Make complex workflows more readable

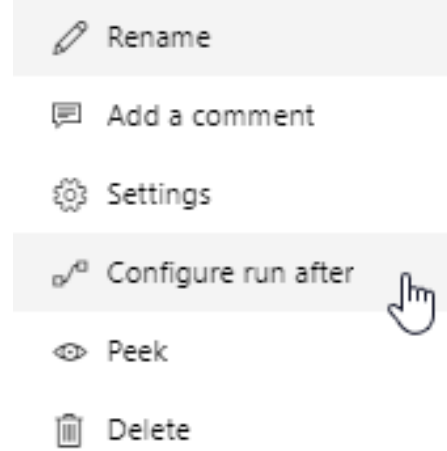


- Use as a “try” block
 - To compensate in a “catch” block




Run after

- The RunAfter defines which action must be executed with a specific status, before another action starts.
- Default RunAfter:
 - If previous succeeded
- You can select one of these statuses:
 - Succeeded
 - Failed
 - Skipped
 - Timed Out



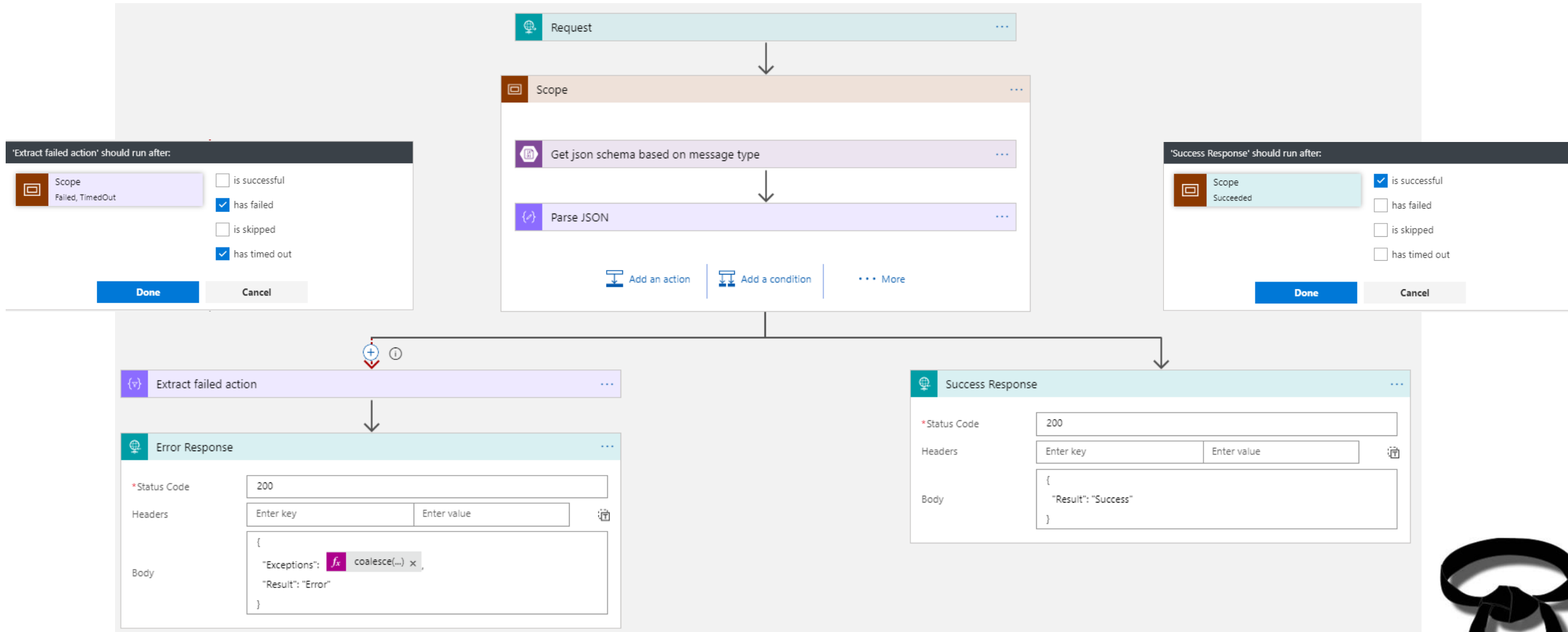
'Success Response' should run after:

 Scope	<input checked="" type="checkbox"/> is successful
Succeeded	<input type="checkbox"/> has failed
	<input type="checkbox"/> is skipped
	<input type="checkbox"/> has timed out

Done Cancel



Try-Catch



Tips – Avoid Connector Throttling

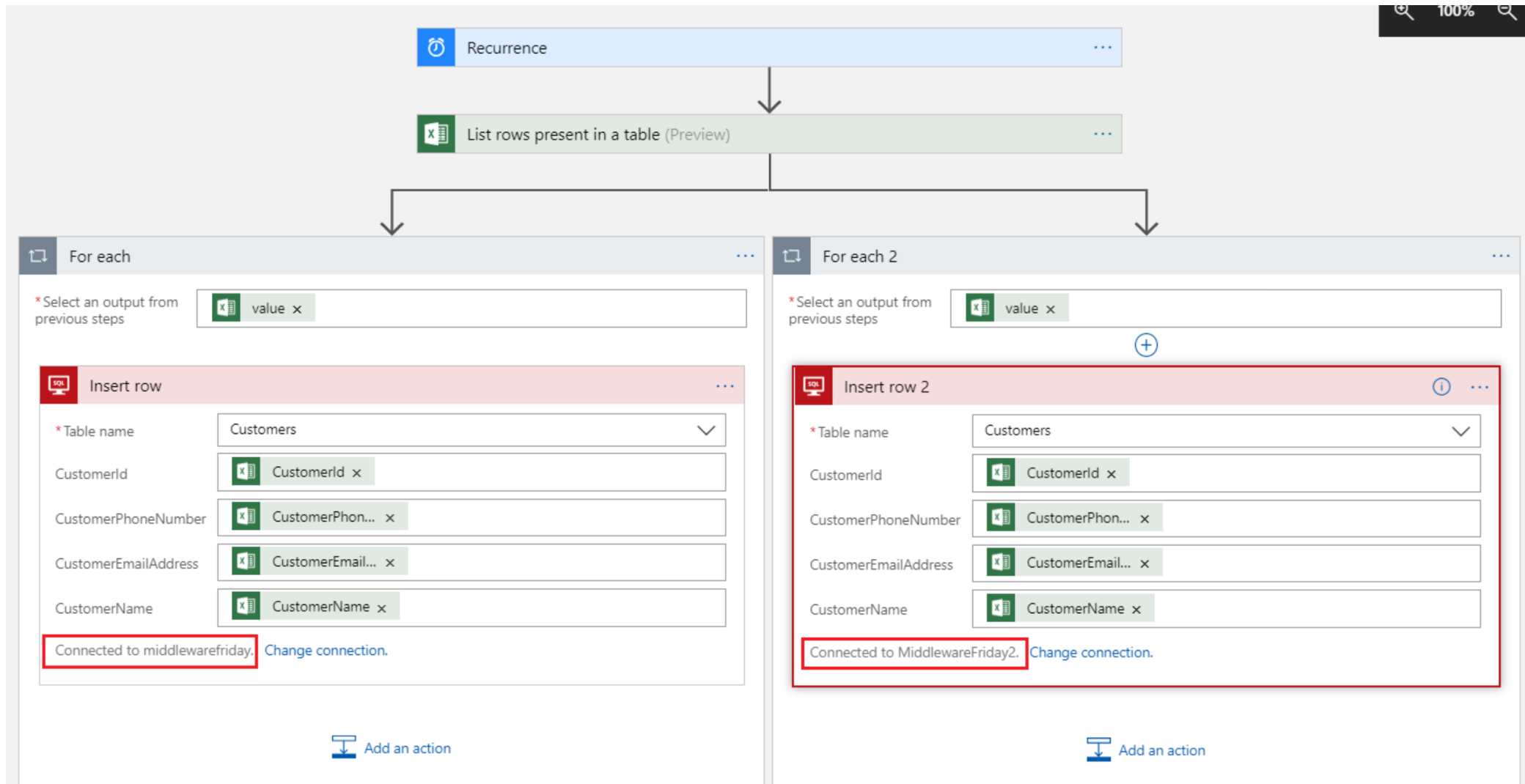
- Each connector has different throttling limits!
 - For instance: SQL Server connector allows for 800 calls per 10 seconds
- Push too much: HTTP 429

A 429 error represents 'too many requests' and forces the client to wait before sending a subsequent request. This creates delays in processing and can impact business process performance.

To avoid rate limiting - create multiple connections and then parallelize your processing.



Tips – Avoid Connector Throttling



Tip – Improve Performance

- Singleton setting concurrency to 1 (first in, first out)
- Parallelism up to 50

Settings for 'For each'

Concurrency Control
For each loops execute in parallel by default. Customize the degree of parallelism, or set it to 1 to execute in sequence.

Concurrency Control ☒ On

Degree of Parallelism 1

Tracked Properties

Properties

--	--

Done **Cancel**



Tips – Secure HTTP Endpoints

- By default there is a SAS key in the URL of the endpoint

<https://prod-60.westeurope.logic.azure.com:443/workflows/f3f11467cfd841ff94c12f11c947c565/triggers/request/paths/invoke?api-version=2016-10-01&sp=%2Ftriggers%2Frequest%2Frun&sv=1.0&sig=AqrIM3fVZb000eh7YfN3pb2TWbWLzuPGzk54jzp5kNA>

- Further hardening of the endpoint can be:
 - IP Whitelisting
 - API Management

Access control configuration

Allowed inbound IP addresses

Restrict calls to triggers in this logic app to the provided IP ranges. IP addresses can be either IPv4 or IPv6 and accepts range and bitmask range formats.

Trigger access option

Any IP	^
Any IP	
Only other Logic Apps	
Specific IP ranges	

IP RANGES FOR CONTENTS

input the valid IP ranges, format like x.x.x.x/x or x.... ...



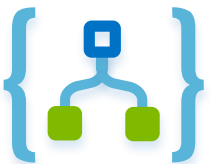
Wrap-up

- Versatile workflow cloud service
- Supports many integration type scenarios
- No steep learning curve
- Solving the business problem first
- Less cost, fast time to market
- High level of maturity



Resources

- Logic App Documentation: <https://docs.microsoft.com/en-us/azure/logic-apps/>
- Serverless notes: <https://www.serverlessnotes.com/>
- Codit Blog: <https://www.codit.eu/blog/>
- Serverless360: <https://www.serverless360.com/?s=Logic+Apps>
- Middleware Friday: <http://www.integrationusergroup.com/middleware-friday/>
- YouTube: <https://bit.ly/2N1iA8W>
- Logic App Blog: https://blogs.msdn.microsoft.com/david_burgs_blog/tag/logic-apps/
- Pluralsight:
 - <https://www.pluralsight.com/courses/azure-logic-apps-fundamentals>
 - <https://www.pluralsight.com/courses/azure-logic-apps-getting-started>



Do you feel your expert now?



Contact



@SteefJan

codit

| Steef-Jan.Wiggers@codit.eu



<https://github.com/steefjan>

Event
partners
























Expo
partners



Expo light
partners



15:00
16:00

	Room 1 (IMAX)	Advanced Data Factory: Let the Data Flow! Simon Whiteley  Data Platform  Business Intelligence  Cloud in the Enterprise	
	Room 6	8 lessons learned running K8S with Azure Kubernetes Service Pascal Naber  Cloud Management  Cloud in the Enterprise  Cloud Native	
	Room 7	Aggregations in Power BI Marco Russo  Business Intelligence	
	Room 8	DevOps for Artificial Intelligence, the road to production Henk Boelman  Everyday AI	
	Room 9	A guide through the Azure Messaging services Eldert Grootenboer  Cloud in the Enterprise	
	Room 10	Azure IoT Hub for client data collection Rasmus Wätjen  IoT & Streaming Data  Serverless	