



**Keverion**

# Azure Automation Best Practices Guide

Published by

Greg Charman

VP of Solutions & Services at Keverion

# Contents

1	INTRODUCTION.....	3
1.1	ABOUT KELVERION.....	3
2	AZURE AUTOMATION BEST PRACTICES.....	4
2.1	NAME RUNBOOKS WITH A STRUCTURE.....	4
2.1.1	Using Numbers within the "Descriptive Function" part of the runbook name.....	5
2.1.2	Meaningful names for runbooks and activities.....	5
2.1.3	Link Colours and link labels.....	5
2.2	HYBRID WORKER DEPLOYMENT.....	6
2.3	SOLUTION SEGREGATION VIA AUTOMATION ACCOUNTS.....	6
2.4	KEEP CONNECTION ASSET NAMES GENERIC.....	6
2.5	DON'T USE FILES FOR DATA REPOSITORIES.....	7
2.6	USE VARIABLE ASSETS SPARINGLY.....	7
2.7	CREATE COMMON TASK RUNBOOKS.....	7
2.8	CREATE A ROBUST ERROR REPORTING ROUTINE.....	8
2.9	FORWARD JOB LOGS TO OMS.....	10
2.10	ADD A BRANCH FOR "NOTHING TO PROCESS".....	11
2.11	IMPLEMENT AUTOMATION WITH A RUNTIME DB.....	8
2.12	ALWAYS DEPLOY THE IM FOR SQL.....	8
2.13	ADD INITIALISATION PHASE TO EACH OF YOUR RUNBOOKS.....	12
2.14	DON'T USE POWERSHELL WORKFLOW.....	13
2.15	DEBUG TO THE PDS.....	10
2.16	PSPRIVATEMETADATA AND "FINDING YOURSELF".....	13
2.17	SCHEDULING RUNBOOK EXECUTION.....	15
2.18	HEARTBEAT RUNBOOKS.....	16

# 1 Introduction

## 1.1 About Kelverion

Kelverion specializes in Service Request Automation and offers organizations automation solutions, integrations and supporting services.

Kelverion's solutions wrap around your existing ITSM tool and enable organizations to harness the power of automation in the service management space, delivering a 400% return on investment over the first 12 months.

Kelverion has been involved with Microsoft's Azure Automation since its inception and has a deep understanding of what is required to build efficient and effective automation solutions using Azure Automation. These best practices incorporate Kelverion's own experiences building Azure Solutions and feedback from customers around their challenges.

For more information or a demonstration, please chat with a member of our team via [info@kelverion.com](mailto:info@kelverion.com) or visit: [www.kelverion.com](http://www.kelverion.com).

## 2 Azure Automation Best Practices

The following section contains several best practices that come from Kelverion's experience with Azure Automation and the automations built using other tools. It's important to recognize that these are recommendations rather than hard and fast rules.

Each recommendation should be considered for use within a user's own automations, and while you may find that some of the recommendations don't fit for development or proof of concept runbooks, the Kelverion team have found that they have been of great benefit in production solutions.

### 2.1 Name runbooks with a structure

Always use a structure for Runbooks names to simulate a folder structure within Azure. As readers may be aware, there is no folder structure for runbooks within Azure, however, by using a 3-part identifier for your runbooks, this can be simulated. The recommended approach is

Project\_Type\_DescriptiveFunction

e.g. Alerting\_Monitor\_SCOM

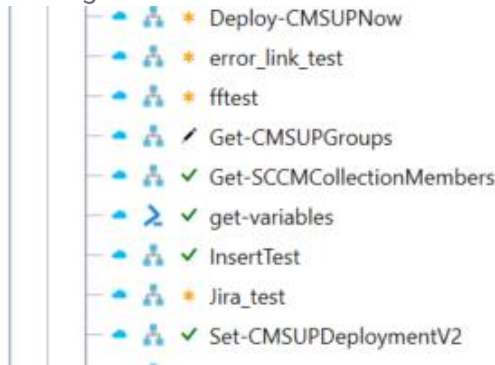
Alerting\_Monitor\_ServiceDesk

Alerting\_Update\_Incident

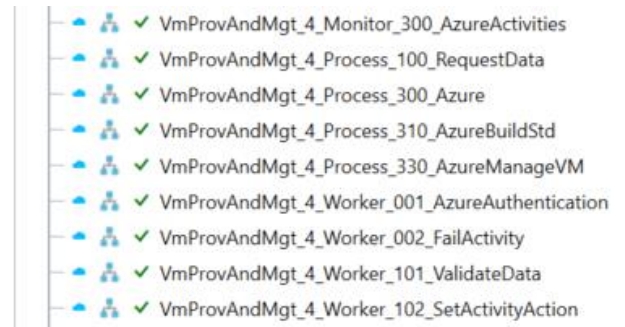
Alerting\_Close\_Event

WHY? This helps keep things simple. It Groups all the related runbooks together and makes it easier to manage them.

### Poor Naming



### Good Naming



## 2.2 Using numbers within the "Descriptive Function" part of the runbook name

As in the pictures above, adding a numeric identifier to the third segment of the runbook name helps define the expected order of execution and provide an association between the Type segment.

i.e. all the 3xx runbooks above are related.

## 2.3 Meaningful names for runbooks and activities

Give all runbooks and the activities within meaningful names rather than using defaults; this will save time when documenting and supporting your solution. Don't forget that both runbooks and the activities they contain have description fields.

## 2.4 Link Colours and link labels

Using the Runbook Studio, users can colour code activities and links to make the flow of execution, error handling and other vital features much easier to understand. Combine colour with link labels and descriptive names, and the runbooks will almost document themselves.

## 2.5 Hybrid Worker Deployment

Hybrid-worker groups should contain at least two machines for resilience. If the hybrid-workers are not available, then runbooks that should be executed will go into a suspended state.

Ideally, the deployment and configuration of your hybrid workers should be automated. Cloud Hybrid workers can easily be deployed as a Hybrid Runbook Worker role through the Azure virtual machine (VM) extension framework. The Azure VM agent is responsible for management of the extension on Azure VMs on Windows and Linux VMs.

This can also be done for on-premise Hybrid Workers, ensure these are Azure Arc enabled servers using the Connected Machine agent.

## 2.6 Solution Segregation via Automation Accounts

Individual Automation Accounts are an excellent way to isolate unrelated solutions, this could include placing the Automation Accounts into separate Resource Groups for improved security separation.

This needs to be considered carefully at the planning stage, as it can introduce additional overheads for managing the solutions. Many of those limitations can be overcome using ARM templates for Automation Account deployment.

However, where hybrid-workers are currently used, they can only be assigned to a single Automation Account. This means that the number of hybrid workers will grow rapidly as the number of automation accounts increases.

## 2.7 Keep Connection Asset names generic

The properties of the connection asset hold all the specific information (User Name, Password, Server Name). If the user includes the server name or the name of the environment in the Asset name, then a user will need to modify it (and the runbooks that utilize that information) when deployed to a different environment or change servers.

For example: use the connection name 'ServiceNow' or 'SCOM' rather than 'SN-Istanbul-dev' or 'SRV-SCOM-MS-01'.

## 2.8 Don't use files for data repositories

Using files to hold temporary data or logging information is not recommended because once the runbook completes, the sandbox in which it was executed will be disposed of and the files removed from the system. This makes it challenging to troubleshoot.

Using a database to hold the temporary data or logging information makes it possible to centralize information independently of where the runbook is executed or which runbook is performing the task.

## 2.9 Use Variable assets sparingly

Variables are useful for pieces of information to be shared across multiple runbooks. However, they don't serve the same purpose as true variables within the runbooks. Although variables can be updated using 'Set-AzureRmAutomationVariable', it can be challenging to manage multiple runbooks reading and writing the same set of variables.

Instead, think of variables as a 'global constant' to be shared across an automation account.

If data must be shared across multiple runbooks (or runbook executions), see 2.11.

## 2.10 Create Common task runbooks

Frequently, users reuse existing patterns within automation, such as logging in to an Azure Account, checking if a type of resource exists, or something more exotic.

If users create common "task" runbooks to carry out those activities, then Azure Automation makes it simple to call those tasks and execute them consistently.

Using the Kelverion Runbook Studio, users can drag and drop a task runbook into the main flow. When calling a child runbook, the child executes in the same process as the parent and executes synchronously, similar to calling a function within a script.

Alternatively, users can invoke a child runbook asynchronously using the 'Start-AzureAzAutomation' activity in the parent runbook. This is like executing a PowerShell Script block with 'start-job'.

## 2.11 Implement automation with a runtime Database

Many automation challenges can be made more manageable through careful supervision of the runbooks and data they use to drive them.

Using a database to support the automation gives users an easy to use centralized store for data that can be shared across runbooks and processes simultaneously as being accessed concurrently by those runbooks.

This database is called the Persistent Data Store (PDS), and it enables many of our best practices.

## 2.12 Always deploy the Integration Module for SQL Server

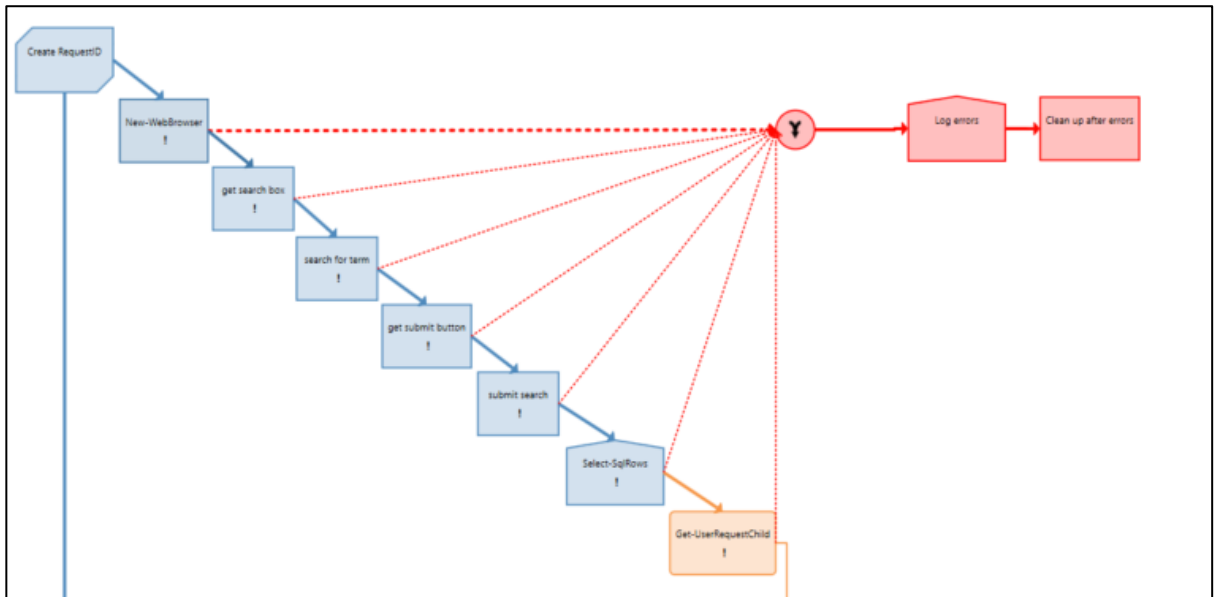
As noted above, the Persistent Data Store (PDS) is a crucial enabler for many Kelverion best practices. To enable the easy deployment of a PDS as part of your automation solution, Kelverion include the Integration Module for SQL Server free with our authoring tool the Runbook Studio.

By deploying the Integration Module for SQL Server, users can exploit the power of smart discovery within the Runbook Studio to accelerate the development of runbooks. This allows the database interactions to happen without the need to write a single line of SQL code.

## 2.13 Create a robust error reporting routine

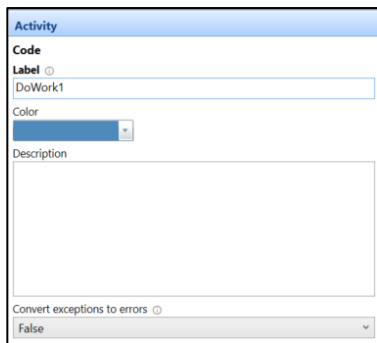
Create a standard Event handling process; within each of the runbooks, write the error to the PDS and allow the error handling runbook to forward the errors to Azure Monitor, System Center Operations Manager, your Service Desk or any other alert notification system you wish to use.





Within a runbook, the hash table `$ActivityError` contains the details for activities that have raised errors. Within a user's own code, either use 'Write-Error' to generate non-terminating errors, or you can "Throw" an exception.

*Note that users can turn fatal errors into non-fatal errors to be handled in a controlled fashion within the runbook when you configure an activity.*



The screenshot shows the 'Activity' configuration window in Azure Automation. It includes a 'Code' section with a 'Label' dropdown set to 'DoWork1', a 'Color' dropdown, a 'Description' text area, and a 'Convert exceptions to errors' dropdown set to 'False'.

## 2.14 Debug to the PDS

The job logs for runbooks are quite verbose and can be hard to understand, even for relatively simple runbooks where all the actions are executed in-line. As the complexity of the runbooks increases, or as soon as a child runbook is executed out-of-band, users will benefit from writing debug information to an external store.

The PDS is the perfect location for that debug information. Consider setting a variable within the initialization section of the runbook to turn the logging on and off and then using a link condition before an 'Insert-SQLRows' Activity to log key pieces of debug information.

## 2.15 Forward job logs to Azure Monitor

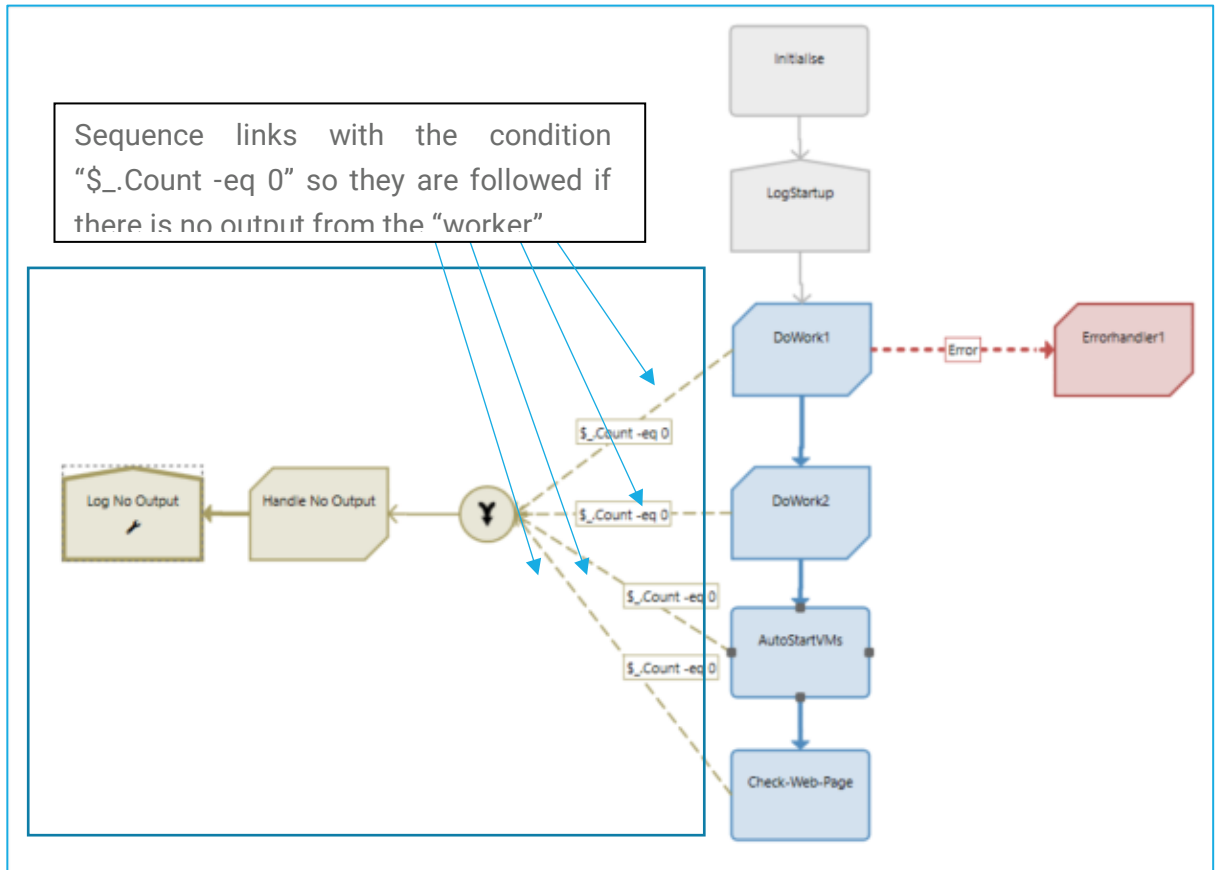
Forward your Azure Automation Job logs to Azure Monitor. The following article shows exactly how to do that, and once this is done, users can configure alerts to warn if issues are affecting the execution of their runbooks.

[Forward Azure Automation diagnostic logs to Azure Monitor](#)

## 2.16 Add a branch for "nothing to process"

When troubleshooting runbooks via the logging in Azure, it can sometimes be challenging if there is no output from an activity when using pipeline mode because the runbook appears to stop executing (this is because there is no work to do.)

Using Sequence links and a junction activity, it's possible to add a generic handler to the runbook if there is nothing to do.



The code activity "Handle No Output" contains the following

```

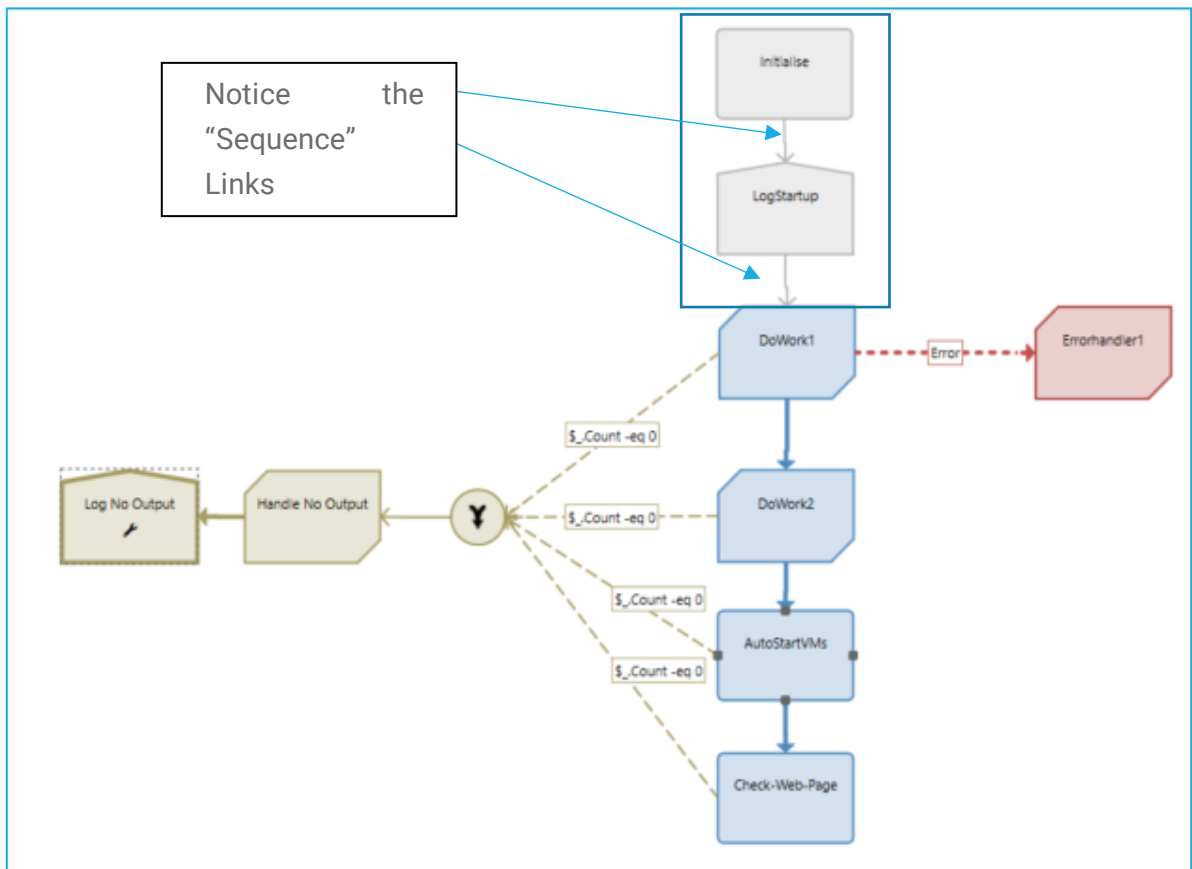
foreach ($o in $activityOutput.GetEnumerator())
{
    if ($o.Value.Count -eq 0)
    {
        write-verbose $("No Output from '{0}'" -f $o.key)
        $o
    }
    else
    {
        write-verbose $("Output from '{0}' = {1}" -f $o.key, $o.value)
    }
}

```

## 2.17 Add Initialization Phase to each of your runbooks

Create a linear section at the start of each runbook to initialize essential data to be used within the rest of the runbook. This initialization phase should use sequence links to hook its activities up (to ensure that each one runs once (and only once.))

Use consistent colour coding through all runbooks for the initialization phase so that other users looking at the runbook realize that it's not part of the "main" flow of the runbook.



## 2.18 Don't use PowerShell Workflow

Initially, Azure Automation only supported PowerShell workflow. Users may find some documentation on the internet or examples of runbooks that use workflow. PowerShell workflow has some interesting features and capabilities; however, it also introduces some new challenges. If the user is already aware of the features and wants them, they are probably ready to deal with the challenges. If not, Kelverion recommends that users just focus on the power of ordinary PowerShell.

## 2.19 PSPrivateMetadata and "Finding yourself"

For every runbook executing, the PowerShell host creates an object holding some valuable information '\$PSPrivateMetadata'.

Within the properties of this object, the JobID can be found for the currently executing runbook '\$PSPrivateMetadata.JobId.Guid'

When recording any information to the PDS about a running runbook, it's valuable to record the JobID as this allows users to retrieve job logs for the runbook.

Within a runbook, users can even use the JobID to find information about the current runbook, e.g. Automation Account Name, Runbook Name.

The following PowerShell Code does precisely that. Please remember that there is no direct method to get the runbook name, so the code below iterates through all the automation accounts for the current subscription. While this is not a particularly resource-intensive operation, this shouldn't be done more than once for a runbook execution (as the data will not have changed.)

This is a perfect example to become a "common task runbook", and its execution probably should be added at the start of the runbook during the "initialization phase" see 2.17

```
$Conn = Get-AutomationConnection -Name "AzureRunAsConnection"
$JobId = $PSPriVateMetadata.JobId.Guid

$Add_Acct_parms = @{
    ServicePrincipal          = $True;
    TenantId                 = $Conn.TenantId;
    ApplicationId            = $Conn.ApplicationId;
    CertificateThumbprint    = $Conn.CertificateThumbprint
}

Add-AzureRmAccount @Add_Acct_parms | out-null
# in there are multiple subs
Select-AzureRmSubscription -SubscriptionId $Conn.SubscriptionID | out-null

foreach ($Res in @(Find-AzureRmResource -ResourceType
Microsoft.Automation/AutomationAccounts))
{
    write-verbose $("checking ({0}:{1}) for JobID:{2}" -f
$Res.ResourceGroupName, $Res.Name, $JobID)
    $Job = Get-AzureRmAutomationJob -ResourceGroupName
$Res.ResourceGroupName -AutomationAccountName $Res.Name -Id $JobID -
ErrorAction SilentlyContinue
    if (!(([string]::IsNullOrEmpty($Job)))
    {
        New-Object PSObject -Property @{
            "Location" = $res.Location;
            "ResourceGroupName" = $res.ResourceGroupName;
            "AutomationAccountName" = $job.AutomationAccountName;
            "RunbookName" = $job.RunbookName
        }
        break;
    }
}
```

**Note:** if creating a task runbook containing this code, users will only receive information about the "parent" runbook if the child is executed "in-line" if the child is executed "out-of-band" with (Start-AzureRmAutomationRunbook), then the information returned will be for the child runbook alone.

## 2.20 Scheduling Runbook Execution

There are several options available for the scheduled execution of runbooks. There are pros and cons to each option.

- 1) **Runbook Schedules** – Runbook schedules can easily be configured in each Automation Account and assigned to one or more runbooks. This is a straightforward approach to scheduling runbook execution.

### Pros

- No additional cost
- Simple

### Cons

- Limited granularity (1hour)
- Lack of flexibility

- 2) **Webhooks and Logic Apps** – Runbooks can easily have webhooks configured to start the runbook when an HTTP Post is received. Azure has a comprehensive scheduling capability called Logic Apps that can trigger these webhooks.

### Pros

- Simple
- Flexible
- Down to 1 second scheduling
- Easily manage multiple Automation Accounts
- There are additional triggers in Logic Apps like database, so can be used with the PDS to only trigger Azure Automation when there is rows in the database to process.

### Cons

- Only time based
- There is a cost associated with Logic Apps use

- 3) Dispatcher runbook and external configuration store – A custom dispatcher runbook is driven by data within the PDS and can give a considerable level of control over the scheduled execution of runbooks within one or many Automation Accounts. The time element and decision elements can be as simple or complex as required.

Pros

- Powerful
- Flexible

Cons

- Complex (but can use graphical runbooks)
- Requires automation minutes for execution
- It depends on one of the other solutions to trigger the Dispatcher

- 4) Webhooks and Event Grid – Although this is not truly a scheduling approach, it is possible to configure the Azure Event Grid to trigger a runbook via a webhook based on a publish and subscribe model. For example, when a virtual machine is deployed to a subscription, an event is posted to the event grid. An appropriate subscription can then trigger a runbook to perform some actions against that virtual machine

Pros

- No additional cost
- Simple
- No "polling" for work to do

Cons

- Not a scheduling solution as such
- Recurring actions are still dependant on publishing a "trigger event"

## 2.21 Heartbeat Runbooks

If utilizing the power of Azure Monitor, then consider using the alerting to build a "heartbeat" mechanism that will provide out-of-band alerting for any or all the following

- Hybrid Worker Groups
- Scheduling facilities



Create a runbook that sends an event to Azure Monitor periodically (e.g. every 15 minutes) and then configure an alert if there are less than three alerts in any one hour period.