# Runbook concept with PowerShell

Here I describe how to create automations using runbooks. The PowerShell scripts can either run in the Azure environment or on the hybrid workers, which are joined in a domain and can thus access internal resources. The file structure and all resources are displayed and edited via the Azure portal.

Runbooks are executed asynchronously. This means that the first execution is completely finished until the next request is processed (big difference to Azure Functions!). This also means that several executions are stacked (queued) and only one execution is possible in parallel at a time.

# Runbooks

In Azure Automation Accounts, runbooks are the PowerShell scripts that are executed regularly, on user action or via third party applications. These are created in the corresponding automation account according to the technology. These runbooks can then be equipped with other resources such as schedules or credentials.

The runbooks should be named according to the abbreviation corresponding to the environment (AD, MEID, CPC, MW, SPO, EXO, TEA, etc.) and the abbreviation "RB". In addition, a short name that is as meaningful as possible should be used for the action of the script.
Example:

- RB-EXO-PRD-PS1-CreateDomainByCompany-PROD-WE
- RB-MEID-ALL-PS1-CleanUpMEIDDevices-PROD-WE

# Automation Credentials

App Registrations and Service Accounts Credentials can be stored in the Automation Credentials. These are to be stored according to the following concept. Usernames and passwords, but also Secret ID and Client Secrets can be stored here. It is also important here that the corresponding resource names (account name, app registration name) are contained one-to-one.
Example:

- CRED-RB-PRD-CRED-CleanUpMEIDDevices-PROD-WE
- CRED-RB-ALL-CRED-CleanUpMEIDDevices-PROD-WE

## Access in the PowerShell script to the corresponding variables

```
$CredentialObject=Get-AutomationPSCredential -Name 'CRED-RB-ALL-CRED-CleanUpMEIDDevices-PROD-WE'
$Password = $CredentialObject.GetNetworkcredential().password

$Username = $CredentialObject.GetNetworkcredential().username
```

# Certificates

The certificate storage of Azure Runbooks can be used so that authentication can also be performed using certificates.
Example:

- CERT-RB-PRD-CERT-CleanUpMEIDDevices-PROD-WE
- CERT-RB-ALL-CERT-CleanUpMEIDDevices-PROD-WE

# Variables

Under variables, contents and strings can be stored that simply have to be adapted or should not be hardcoded in the script.
Example:

- VAR-RB-PRD-CLIENTID-CleanUpMEIDDevices-PROD-WE
- VAR-RB-ALL-CLIENTID-CleanUpMEIDDevices-PROD-WE

## Access in the PowerShell script to the corresponding variables

```
$ClientId=Get-AutomationVariable -Name "VAR-RB-PRD-CLIENTID-CleanUpMEIDDevices-PROD-WE"
```

# PowerShell Modules

In order to use PowerShell modules, they must first be added to the Azure Automation account. This can be achieved in the Automation Account via "Modules" → "Add a Module" → "Browse from Gallery".

## Import module into PowerShell script

```
Import-Module AzureAD
```

# Schedules

The schedules are finally mapped to the runbook. These can be set granularly on a weekly, daily or hourly basis. The following concept is also used in the schedule name. In addition to the action name, the abbreviation "RB" for runbook is added here. In addition, there is one of the following regularity descriptions:

- Hourly
- Daily
- Weekly
- Monthly
- Yearly
- 12Hourly

Example:

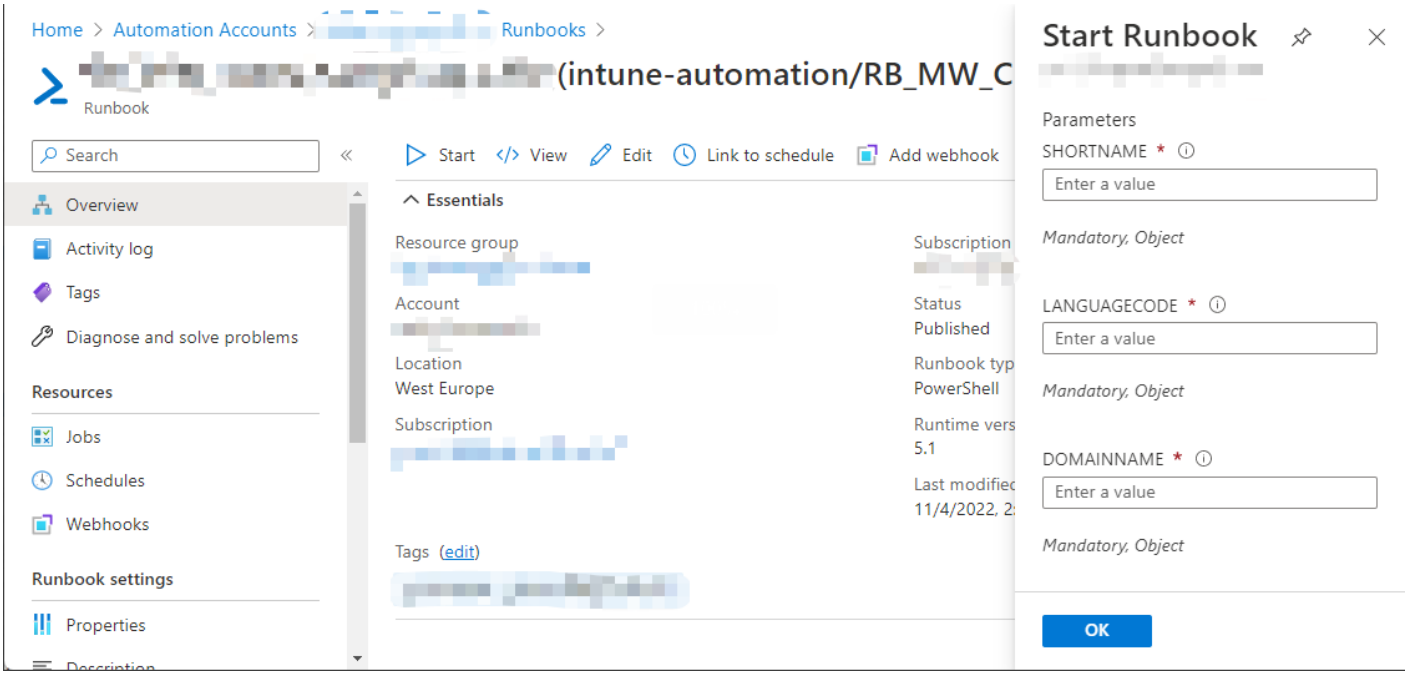
- SCED-RB-PRD-DAILY-CheckLicenseCount-PROD-WE
- SCED-RB-PRD-WEEKLY-CleanUpMEIDDevices-PROD-WE

# Triggers

Triggers mean types to call the Azure Runbooks and execute the code.

## Manual

Azure Runbooks can be conveniently executed via the GUI. The required parameters can also be entered in the GUI Form.




## Schedules

Schedules can be defined in the Automation Account. This allows scripts to be run on a regular basis. These schedules can then be assigned to the runbooks. To remain transparent, only one schedule per runbook and one runbook per schedule may be used.

## PowerShell

A runbook can also be executed via PowerShell. However, a webhook must be created for this, via which a runbook job can be created. Since runbooks do not automatically return the response and can take several hours to execute, the PowerShell script must also wait for the output. Here we see an example where a PowerShell script waits 165 seconds for the job output and terminates otherwise.

```
# Create AzRUN Job
$url = "<yourrunbookwebhookurl>"
$Body = @"
{
    "email":"$Email"
}
"@
$JobId = Invoke-RestMethod -Method POST -Uri $url -Body $Body
$JobId = $JobId.JobIds

$whilecounter = 1
```

```
# Get AzRUN Job Output
$url =
"https://management.azure.com/subscriptions/<subscriptionid>/resourceGroups/<resourcegroupname
>/providers/Microsoft.Automation/automationAccounts/<automationaccountname>>/jobs/$JobId/?api-
version=2019-06-01"
$Response = Invoke-Restmethod -uri $url -Method GET -Headers $Headers
$response.properties.provisioningstate
while($response.properties.provisioningstate -ne "Succeeded"){
    Start-Sleep 15
    $Response = Invoke-Restmethod -uri $url -Method GET -Headers $Headers
    $response.properties.provisioningstate
    if($whilecounter -le 10){
        $whilecounter ++
    }
    else{
        Write-Error "Get JobOutput from Runbook failed. Exiting Script."
        Exit 1
    }
}

$outputurl =
"https://management.azure.com/subscriptions/<subscriptionid>/resourceGroups/<resourcegroupname
>/providers/Microsoft.Automation/automationAccounts/<automationaccountname>>/jobs/$JobId/outpu
t?api-version=2019-06-01"
$Language = Invoke-Restmethod -uri $outputurl -Method GET -Headers $Headers
```

# ServiceNow

ServiceNow has a spoke for interaction with Azure Automation accounts and can use it to execute runbooks and read out their feedback. SNOW receives authorisation for access via a service account. This account must be able to read all resource groups on the subscription and have the rights to execute runbooks on the automation account.

# Authentication to Azure Automation Account

Authentication to the Azure Automation Account happens in exactly the same way in the frontend as well as via code. Here, the app registration must have the permission "user_impersonation" on the Azure API. Then all requests go via the API https://management.azure.com. Accordingly, the app registration must also be authorised in the IAM of the Azure Automation Account by means of Managed Identity.

---

Revision #7
Created 2022-12-01 17:00:17 UTC

Updated 2026-08-04 16:18:37 UTC by Luca Noah Caprez