Automating MSFT Fabric Deployments

avanade

Hector Lopez
Manager, Data Engineering
hector.lopez@avanade.com
www.linkedin.com/in/svenchio

Extra info…

Originally from Mexico

Live in Norway since 2018

Azure Associate-level certifications (Developer, Data Engineer, DB Admin & Power BI Data Analyst and Fabric)

DevOps Engineer Expert certified

Content creator and instructor

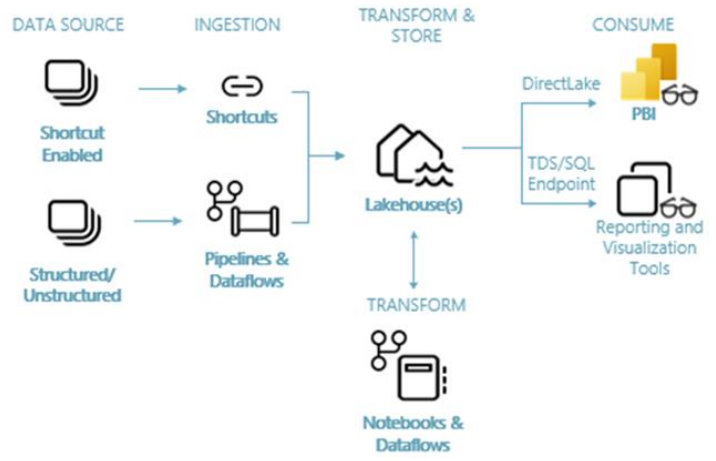
#TechTacoFriday - my blog on data engineering & automation topics

Agenda for today

"Today, we'll explore Fabric Catalyst's Auto Deployment, practical PowerShell techniques, and REST API best practices."

Inception: Solving a Real-World Problem

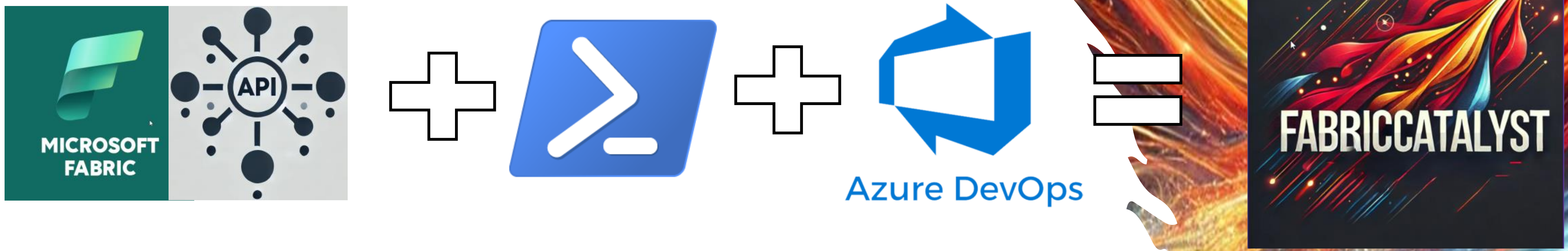
"It all started with a workshop I was designing for developers to learn best practices for working with Git-enabled Fabric workspaces, requiring fresh workspaces for each group to complete exercises.“



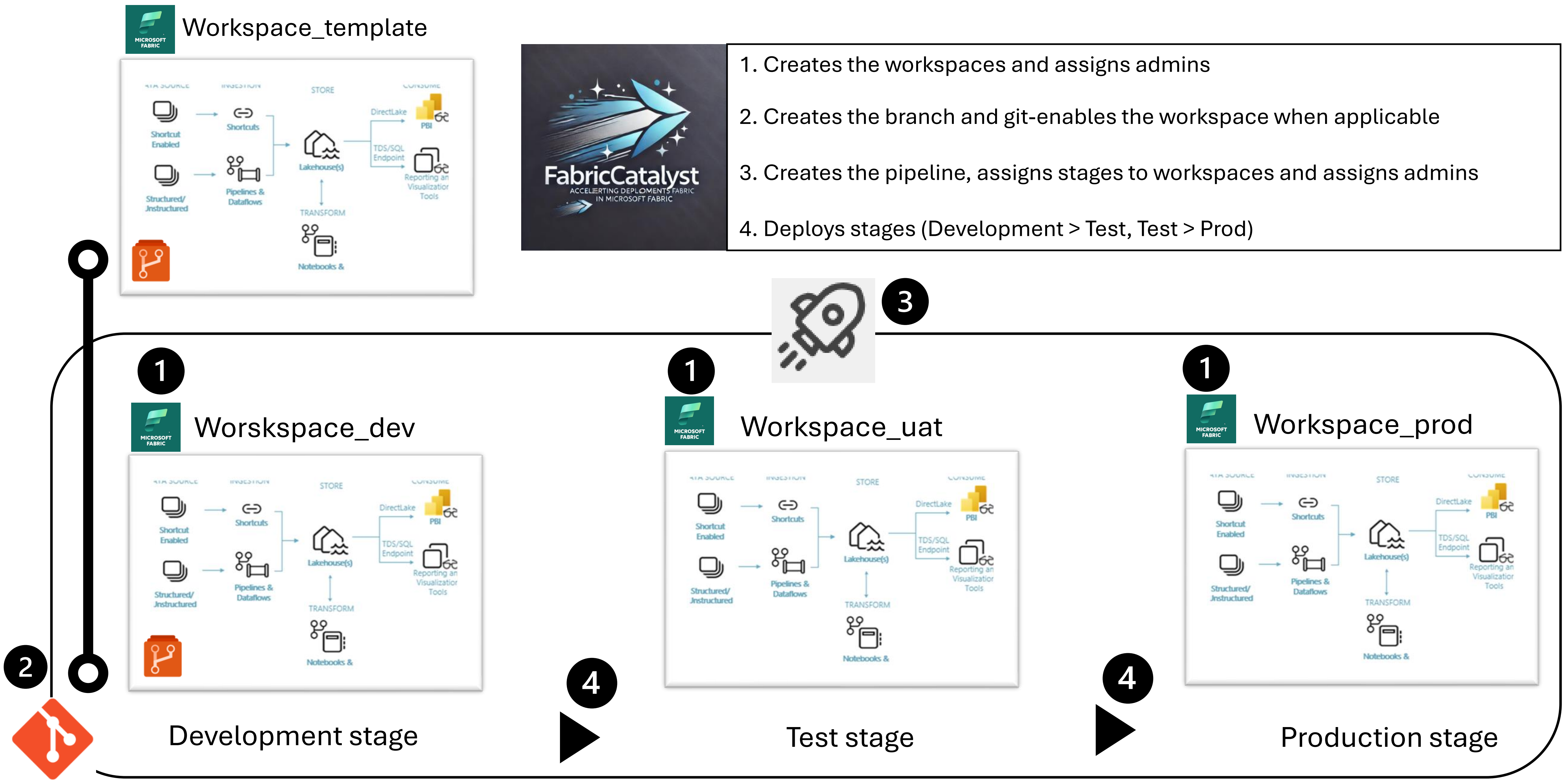
"Manually provisioning these environments was impractical & time consuming. I needed a way to efficiently replicate my workshop template for multiple groups. This challenge sparked the idea of creating a series of PowerShell scripts and using DevOps pipelines to automate the deployment of Fabric environments

Introduction to FabricCatalyst

- Project designed **to automate the deployment of Fabric environments** through PowerShell scripts that interact with the Fabric REST API.
- **Integrates into DevOps pipelines**, providing a user-friendly mechanism for modifying parameters and triggering automated deployments.
- Offers distinct **deployment methods**: **<u>auto</u>**, **custom** & **map**; each tailored to different user needs.

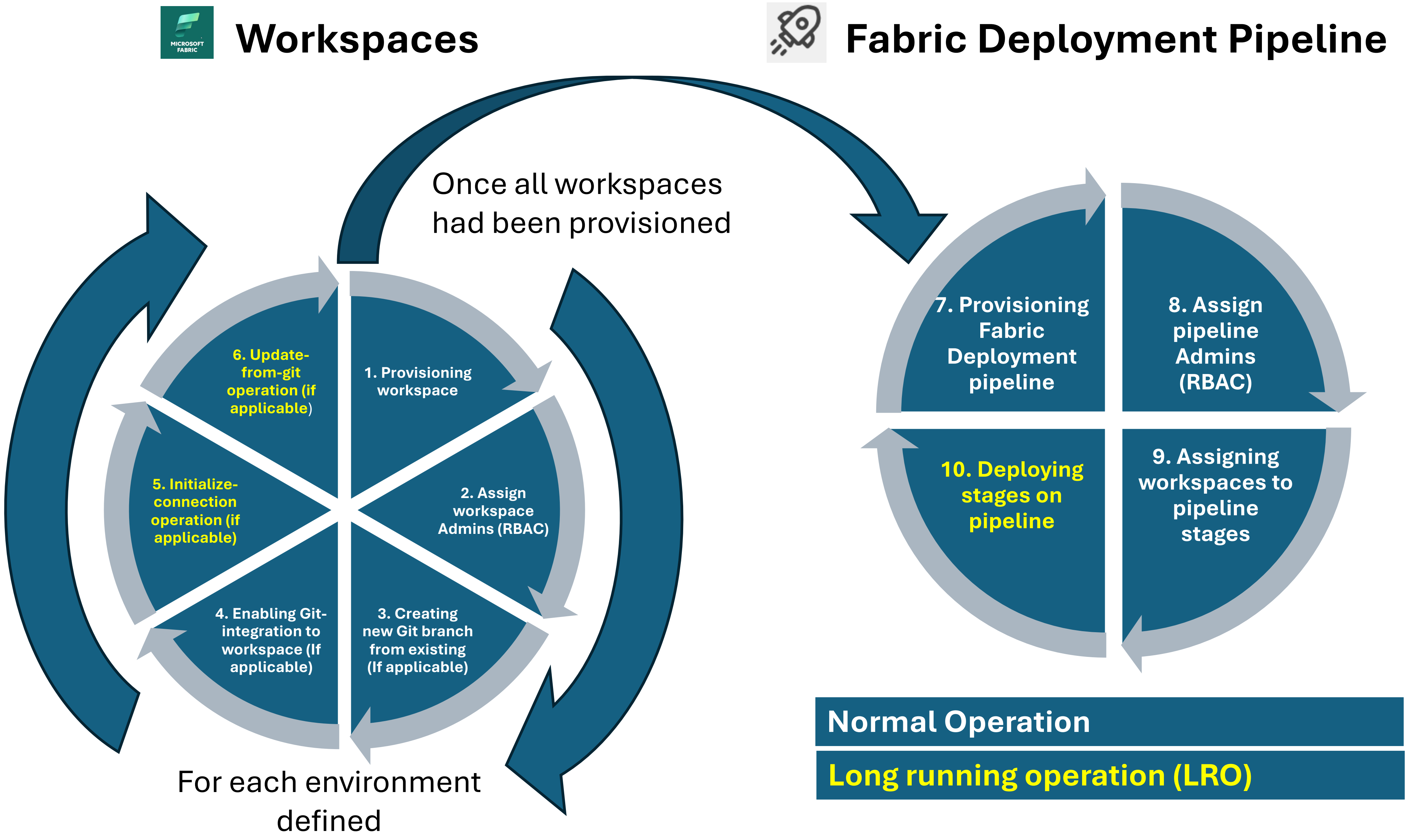


Fabric arch. pattern in a nutshell



1. Creates the workspaces and assigns admins
2. Creates the branch and git-enables the workspace when applicable
3. Creates the pipeline, assigns stages to workspaces and assigns admins
4. Deploys stages (Development > Test, Test > Prod)

Auto Deployment Operations Workflow


Workspaces
Fabric Deployment Pipeline
Once all workspaces had been provisioned
1. Provisioning workspace
2. Assign workspace Admins (RBAC)
3. Creating new Git branch from existing (If applicable)
4. Enabling Git-integration to workspace (If applicable)
5. Initialize-connection operation (if applicable)
6. Update-from-git operation (if applicable)
For each environment defined
7. Provisioning Fabric Deployment pipeline
8. Assign pipeline Admins (RBAC)
9. Assigning workspaces to pipeline stages
10. Deploying stages on pipeline
Normal Operation
Long running operation (LRO)

Long Running Operations (LRO)

- Pattern for Fabric APIs
- You will get one of two responses:
  - ✓Operation completed, **status code 201 created or 200 OK**: the operation is completed, and the result (if exists) is returned in the body.
  - ✓Operation ongoing, **status code 202 accepted**: this means the operation is ongoing, and the next steps would be to poll on the state until the operation is complete and then to get the result (if exists).

Important! This is why you need to use
Invoke-WebRequest [1]
when working with Fabric APIs

(1) Invoke-WebRequest gives you full access to the Response object and all the details it provides, whereas Invoke-RestMethod is just for APIs that have no special response information

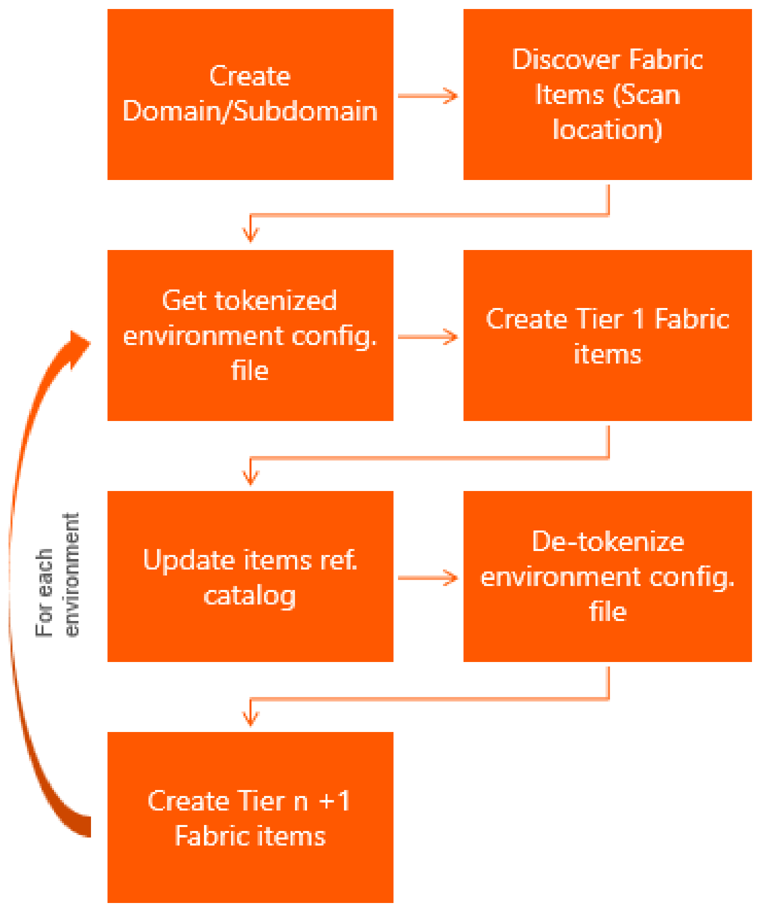
Demo time!

Let's watch Auto Deployment in Action

The other two methods: Custom & Map

Custom

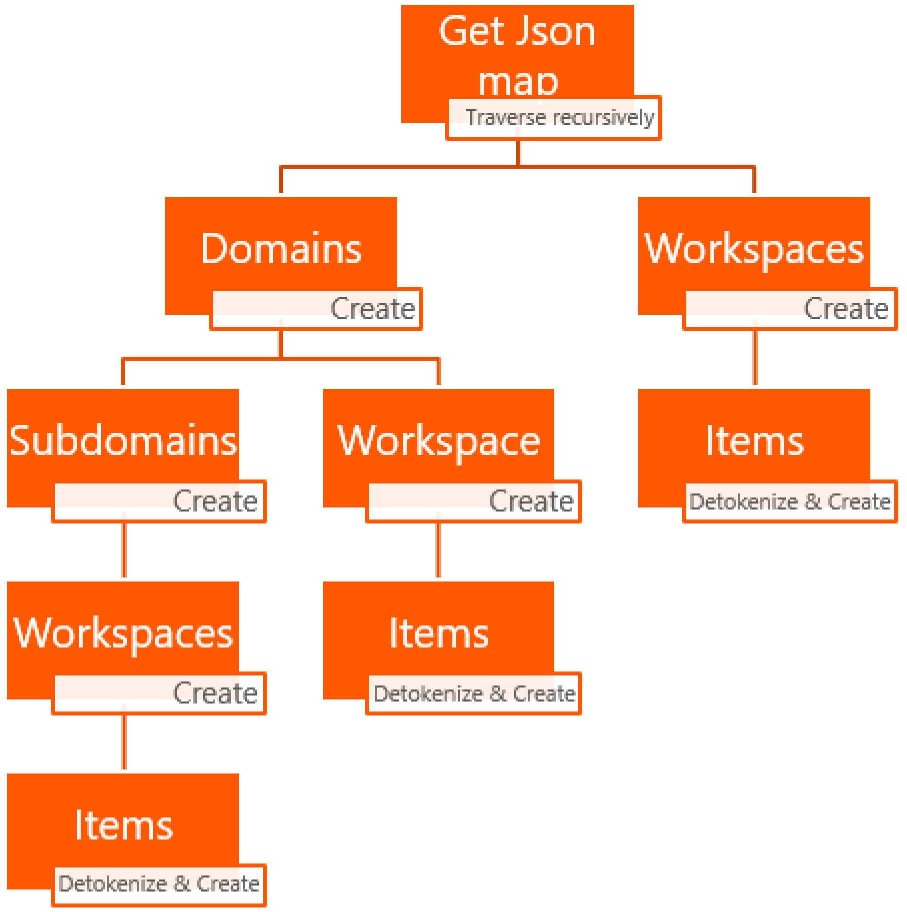
Offers a more **flexible, Git-less approach** that allows users to specify where the process should scan for Fabric items.



Create Domain/Subdomain
Discover Fabric Items (Scan location)
Get tokenized environment config. file
Create Tier 1 Fabric items
Update items ref. catalog
De-tokenize environment config. file
Create Tier n +1 Fabric items
For each environment

Map

Map is designed for **large-scale, complex deployments** that involve multiple environments and dependencies.



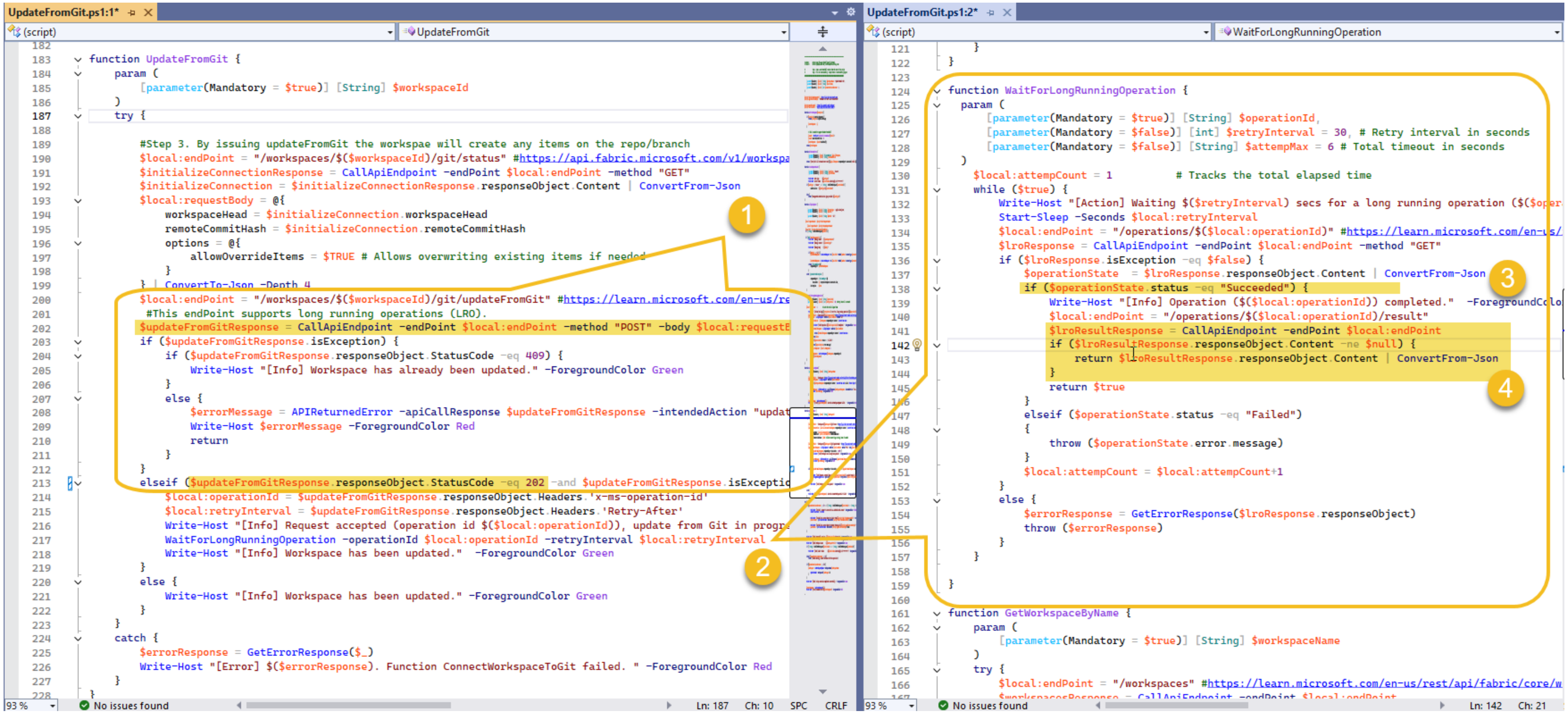
Current challenges with the Fabric Automation (Summary)

- Growing pains (e.g. Constant changes, Lacking documentation)
- Compromise between “what’s coming” vs “what’s available”
- Arrival fallacy, psychology of anticipation
- Bring-your-own (BYO) vs Terraform

Have a question? Ask away!

Every inquiry sparks growth!

Extra: Handling Long Running Operations



```powershell
function UpdateFromGit {
    param (
        [parameter(Mandatory = $true)] [String] $workspaceId
    )
    try {

        #Step 3. By issuing updateFromGit the workspae will create any items on the repo/branch
        $local:endPoint = "/workspaces/$($workspaceId)/git/status" #https://api.fabric.microsoft.com/v1/workspa
        $initializeConnectionResponse = CallApiEndpoint -endPoint $local:endPoint -method "GET"
        $initializeConnection = $initializeConnectionResponse.responseObject.Content | ConvertFrom-Json
        $local:requestBody = @{
            workspaceHead = $initializeConnection.workspaceHead
            remoteCommitHash = $initializeConnection.remoteCommitHash
            options = @{
                allowOverrideItems = $TRUE # Allows overwriting existing items if needed
            }
        } | ConvertTo-Json -Depth 4
        $local:endPoint = "/workspaces/$($workspaceId)/git/updateFromGit" #https://learn.microsoft.com/en-us/re
         #This endPoint supports long running operations (LRO).
        $updateFromGitResponse = CallApiEndpoint -endPoint $local:endPoint -method "POST" -body $local:requestE
        if ($updateFromGitResponse.isException) {
            if ($updateFromGitResponse.responseObject.StatusCode -eq 409) {
                Write-Host "[Info] Workspace has already been updated." -ForegroundColor Green
            }
            else {
                $errorMessage = APIReturnedError -apiCallResponse $updateFromGitResponse -intendedAction "updat
                Write-Host $errorMessage -ForegroundColor Red
                return
            }
        }
        elseif ($updateFromGitResponse.responseObject.StatusCode -eq 202 -and $updateFromGitResponse.isExceptic
            $local:operationId = $updateFromGitResponse.responseObject.Headers.'x-ms-operation-id'
            $local:retryInterval = $updateFromGitResponse.responseObject.Headers.'Retry-After'
            Write-Host "[Info] Request accepted (operation id $($local:operationId)), update from Git in progr
            WaitForLongRunningOperation -operationId $local:operationId -retryInterval $local:retryInterval
            Write-Host "[Info] Workspace has been updated."  -ForegroundColor Green
        }
        else {
            Write-Host "[Info] Workspace has been updated." -ForegroundColor Green
        }
    }
    catch {
        $errorResponse = GetErrorResponse($_)
        Write-Host "[Error] $($errorResponse). Function ConnectWorkspaceToGit failed. " -ForegroundColor Red
    }
}
```

93 % · No issues found · Ln: 187 Ch: 10 SPC CRLF

```powershell
        }
    }

function WaitForLongRunningOperation {
  param (
      [parameter(Mandatory = $true)] [String] $operationId,
      [parameter(Mandatory = $false)] [int] $retryInterval = 30, # Retry interval in seconds
      [parameter(Mandatory = $false)] [String] $attempMax = 6 # Total timeout in seconds
  )
    $local:attempCount = 1          # Tracks the total elapsed time
    while ($true) {
        Write-Host "[Action] Waiting $($retryInterval) secs for a long running operation ($($oper
        Start-Sleep -Seconds $local:retryInterval
        $local:endPoint = "/operations/$($local:operationId)" #https://learn.microsoft.com/en-us/
        $lroResponse = CallApiEndpoint -endPoint $local:endPoint -method "GET"
        if ($lroResponse.isException -eq $false) {
            $operationState  = $lroResponse.responseObject.Content | ConvertFrom-Json
            if ($operationState.status -eq "Succeeded") {
                Write-Host "[Info] Operation ($($local:operationId)) completed."  -ForegroundColo
                $local:endPoint = "/operations/$($local:operationId)/result"
                $lroResultResponse = CallApiEndpoint -endPoint $local:endPoint
                if ($lroResultResponse.responseObject.Content -ne $null) {
                    return $lroResultResponse.responseObject.Content | ConvertFrom-Json
                }
                return $true
            }
            elseif ($operationState.status -eq "Failed")
            {
                throw ($operationState.error.message)
            }
            $local:attempCount = $local:attempCount+1
        }
        else {
            $errorResponse = GetErrorResponse($lroResponse.responseObject)
            throw ($errorResponse)
        }
    }

}
function GetWorkspaceByName {
    param (
        [parameter(Mandatory = $true)] [String] $workspaceName
    )
    try {
        $local:endPoint = "/workspaces" #https://learn.microsoft.com/en-us/rest/api/fabric/core/w
        $workspacesResponse = CallApiEndpoint -endPoint $local:endPoint
```

93 % · No issues found · Ln: 142 Ch: 21

Thanks for your attention!

Visit my blog #TechTacoFriday

svenchio@techtacofriday.com

SVENCHIO
TechTacoFriday
Technology spiced with a taco flavor mix

Hector Lopez

Manager, Data Engineering

svenchio@techtacofriday.com

www.techtacofriday.com

Oslo, Norway

Provided upon request

avanade

Hector Lopez
Manager, Data Engineering
hector.lopez@avanade.com
www.linkedin.com/in/svenchio

Project designed **to automate the deployment of MSFT Fabric environments** through PowerShell scripts that interact with the Fabric REST API powered by DevOps pipelines

MICROSOFT FABRIC API + + Azure DevOps

Offers distinct **deployment methods**: **auto**, **custom** & **map**; each tailored to different user needs.