How to handle Azure Firewall Policies & Rules in IaC & DevOps

A personal story

Didier Van Hoye - @WorkingHardInIT

Technical Architect & Technology Strategist

Blogger/Author/Speaker

http://blog.workinghardinit.work

@workinghardinit

VEEAM VANGUARD

Dell TechCenter ROCKSTAR

Microsoft® MVP Most Valuable Professional

How to manage Azure Firewall rules in IaC?

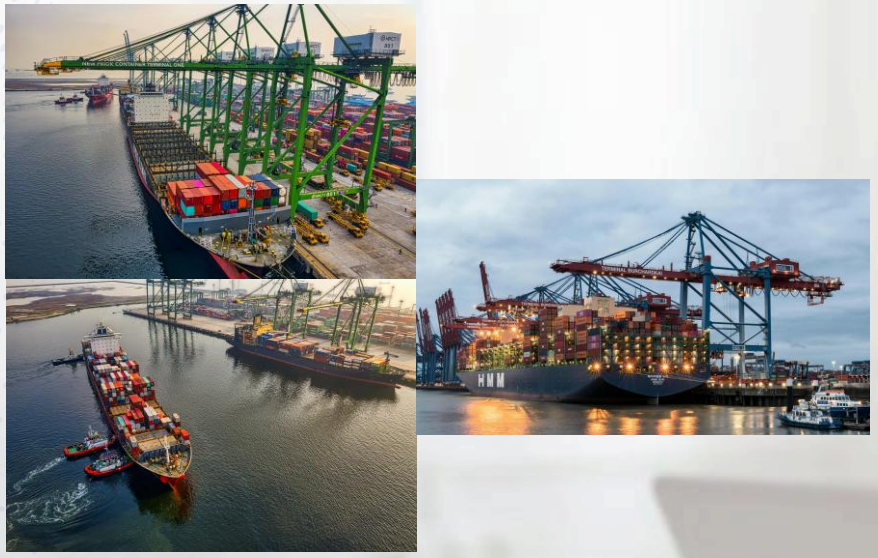


Photo by Andrea Piacquadio: https://www.pexels.com/photo/thoughtful-coworking-women-discussing-project-in-home-office-3831879/

Reaching out to the community & MSFT

LinkedIn

Microsoft® MVP Most Valuable Professional

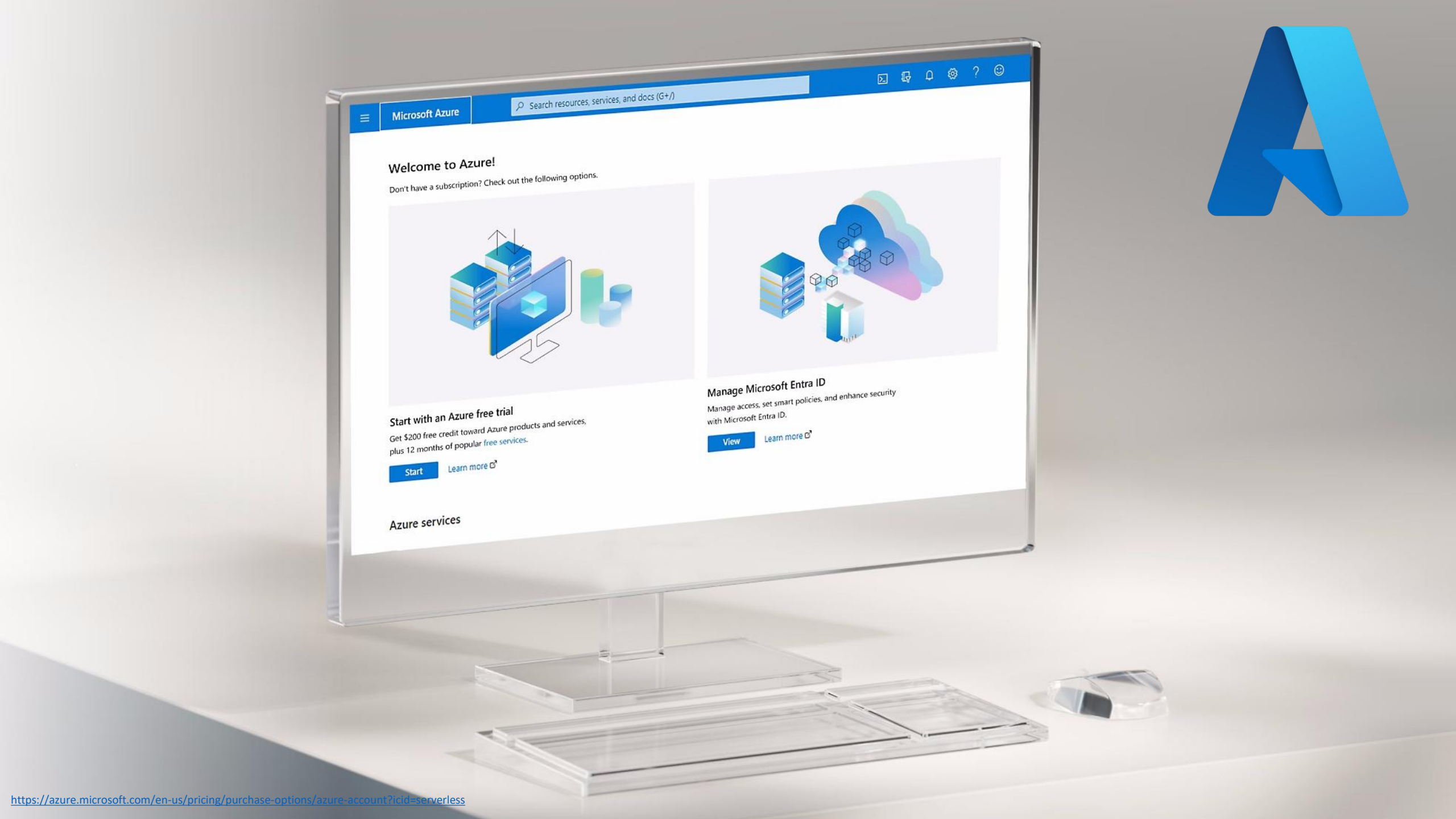
Microsoft

Photo by Wolfgang Hasselmann on Unsplash

```
resource networkRuleCollectionGroup 'Microsoft.Network/firewallPolicies/ruleCollectionGroups@2022-01-01' = {
  parent: firewallPolicy
  name: 'DefaultNetworkRuleCollectionGroup'
  properties: {
    priority: 200
    ruleCollections: [
      {
        ruleCollectionType: 'FirewallPolicyFilterRuleCollection'
        action: {
          type: 'Allow'
        }
        name: 'azure-global-services-nrc'
        priority: 1250
        rules: [
          {
            ruleType: 'NetworkRule'
            name: 'time-windows'
            ipProtocols: [
              'UDP'
            ]
            destinationAddresses: [
              '13.86.101.172'
            ]
            sourceIpGroups: [
              workloadIpGroup.id
              infraIpGroup.id
            ]
            destinationPorts: [
              '123'
            ]
          }
        ]
      }
    ]
  }
}
```

```powershell
########################################## POWERSHELL ##########################################


#Create the network rule collection group
$firewallpolicy = Get-AzFirewallPolicy -Name EUS-Policy -ResourceGroupName Test-FWPolicy-RG

$newnetworkrulecollectiongroup = New-AzFirewallPolicyRuleCollectionGroup  -Name "NetworkRuleCollectionGroup"
 -Priority 200 -ResourceGroupName Test-FWPolicy-RG -FirewallPolicyName EUS-Policy

$networkrulecollectiongroup = Get-AzFirewallPolicyRuleCollectionGroup -Name "NetworkRuleCollectionGroup"
 -ResourceGroupName Test-FWPolicy-RG -AzureFirewallPolicyName EUS-Policy


#Create network rules
$networkrule1= New-AzFirewallPolicyNetworkRule -Name NwRule1 -Description testRule1  -SourceAddress 10.0.0.0/24
 -Protocol TCP -DestinationAddress 192.168.0.1/32 -DestinationPort 22

$networkrule2= New-AzFirewallPolicyNetworkRule -Name NWRule2 -Description TestRule2  -SourceAddress 10.0.0.0/24
 -Protocol UDP -DestinationAddress 192.168.0.10/32 -DestinationPort 1434


#Create a network rule collection and add new rules
$newrulecollectionconfig=New-AzFirewallPolicyFilterRuleCollection -Name myfirstrulecollection -Priority 1000
 -Rule $networkrule1,$networkrule2 -ActionType Allow

$newrulecollection = $networkrulecollectiongroup.Properties.RuleCollection.Add($newrulecollectionconfig)

#Update the network rule collection group
Set-AzFirewallPolicyRuleCollectionGroup -Name "NetworkRuleCollectionGroup" -Priority "200"
-FirewallPolicyObject $firewallpolicy -RuleCollection $networkrulecollectiongroup.Properties.RuleCollection
```

Photo by John Guccione www.advergroup.com: https://www.pexels.com/photo/100-us-dollar-banknotes-3483098/



https://azure.microsoft.com/en-us/pricing/purchase-options/azure-account?icid=serverless

```
using './main.bicep'

param exampleString = 'test string'
param exampleInt = 2 + 2
param exampleBool = true
param exampleArray = [
  'value 1'
  'value 2'
]
param exampleObject = {
  property1: 'value 1'
  property2: 'value 2'
}
```

```
{
  "$schema":
"https://schema.management.azure.com/schemas/2019-04-
01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "exampleString": {
      "value": "test string"
    },
    "exampleInt": {
      "value": 4
    },
    "exampleBool": {
      "value": true
    },
    "exampleArray": {
      "value": [
        "value 1",
        "value 2"
      ]
    },
    "exampleObject": {
      "value": {
        "property1": "value1",
        "property2": "value2"
      }
    }
  }
}
```

Photo by Andrea Piacquadio: https://www.pexels.com/photo/woman-in-gray-sports-bra-holding-black-dumbbell-3757376/

A Pragmatic Approach in DevOps

Photo by RealToughCandy.com: https://www.pexels.com/photo/person-holding-a-sticker-11035393/

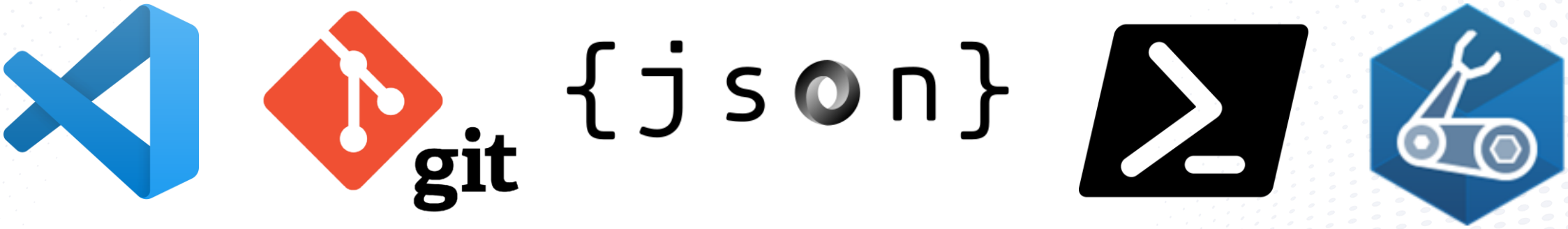
A pragmatic approach (1/2)

- We need CI/CD-friendly automation
  - We use Azure DevOps. It could also be GitHub or Bitbucket.
  - We use Bicep (IT Pros with a focus on Azure), feel free to use Terraform, OpenTofu or Pulumi
  - Handle parameters & orchestration in PowerShell

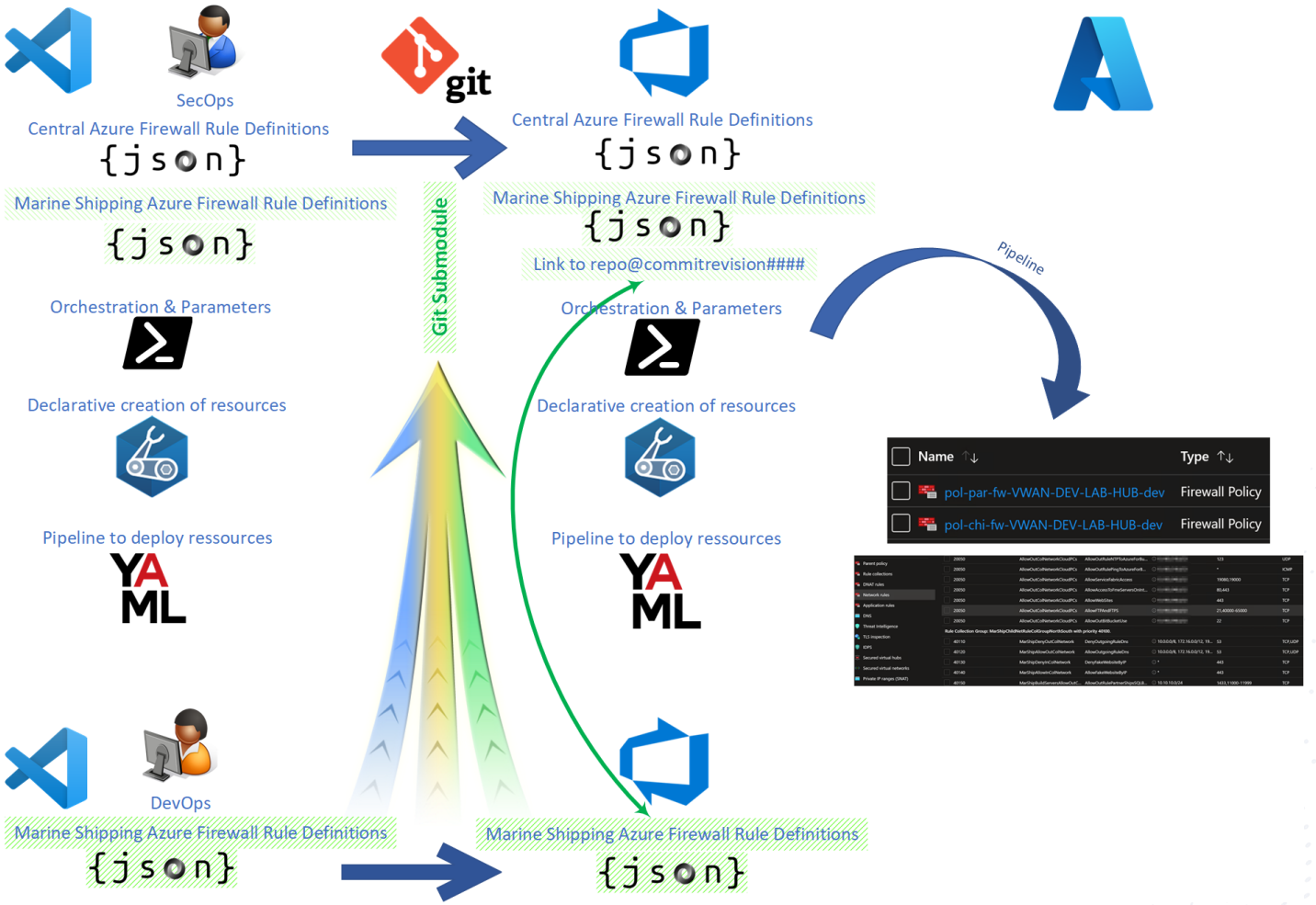
- We want to avoid complexity
  - Keep Bicep as simple as possible (handover, maintenance)
  - Keep PowerShell easy to read and maintain
  - Code should not be touched for “mere” rule maintenance (CRUD)

A pragmatic approach (2/2)

- To achieve this, the firewall policy rules are edited in dedicated JSON files, read via PowerShell, and passed on to Bicep.
- No need to touch the PowerShell or Bicep for rule maintenance
- We only touch the PowerShell when we onboard a new DevOps Team that wants to view or maintain its own rules.



A road map


SecOps
Central Azure Firewall Rule Definitions
{json}
Marine Shipping Azure Firewall Rule Definitions
{json}
Orchestration & Parameters
Declarative creation of resources
Pipeline to deploy ressources
YAML
DevOps
Marine Shipping Azure Firewall Rule Definitions
{json}
git
Git Submodule
Central Azure Firewall Rule Definitions
{json}
Marine Shipping Azure Firewall Rule Definitions
{json}
Link to repo@commitrevision####
Orchestration & Parameters
Declarative creation of resources
Pipeline to deploy ressources
YAML
Marine Shipping Azure Firewall Rule Definitions
{json}
Pipeline
Name
Type
pol-par-fw-VWAN-DEV-LAB-HUB-dev
Firewall Policy
pol-chi-fw-VWAN-DEV-LAB-HUB-dev
Firewall Policy

Why use JSON files for creating & editing Firewalls Policy Rules

Photo by RealToughCandy.com: https://www.pexels.com/photo/a-person-holding-a-paper-11035481/

Avoids large & complex PoSh parameter objects

- Avoids recreating the entire parent and child firewall policies in large, complex, nested PowerShell parameter constructs that are arrays of hash tables
- This approach avoids "mind mapping" the Azure JSON schema to PowerShell parameter nested objects while writing them. Just use JSON directly.

Avoid deeply nested loops in Bicep

- Deploying nested parameters requires nested looping in Bicep. While that is possible with "helper" modules, it is not the most straightforward, easy-to-follow code for maintenance.
- Rules can have different properties or empty properties, handling this json to Bicep conversion can be tedious and complex.
- It recreates the JSON that I translated into a PowerShell parameter object back to pieces of JSON to deploy. That's a lot of needless conversions.

Easy rule creation & maintenance (1/2)

- This is easily readable for both Devs & Ops profiles
- Create one Rule Collection Group with Rule Collections & rules in a JSON file
- A file contains one Rule Collection Group (max 90 RCGs)
- Assign priority ranges to each team
- Split those up per type, direction (North-South, East-West) and business units or dev teams or a mixture of the above.

```
{
  "ruleType": "NetworkRule",
  "name": "AllowOutDeviceRegistration",
  "ipProtocols": [
    "TCP"
  ],
  "sourceAddresses": [
    "10.0.0.0/8",
    "172.16.0.0/12",
    "192.168.0.0/16"
  ],
  "sourceIpGroups": [],
  "destinationAddresses": [],
  "destinationIpGroups": [],
  "destinationFqdns": [
    "hm-iot-in-prod-preu01.azure-devices.net",
    "hm-iot-in-prod-prap01.azure-devices.net",
    "hm-iot-in-prod-prna01.azure-devices.net",
    "hm-iot-in-prod-prau01.azure-devices.net"
  ],
  "destinationPorts": [
    "443",
    "5671"
  ]
}
```

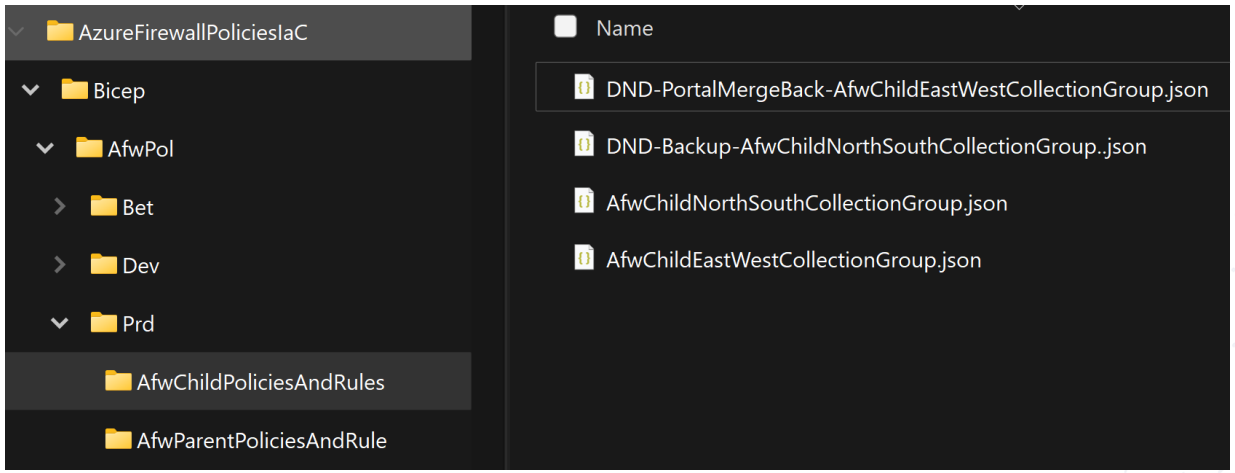
Easy rule creation & maintenance (2/2)

- Multiple files reduce the complexity and size of the files and makes it easier to find where to create or edit the rule at hand.
- You can easily estimate the size of your rule collection groups (max size = 2MB)*****.
- Keep the file (containing a single rule collection group) below 1,9 MB. The difference allows for the overhead. Do reduce unneeded characters in JSON, and keep it clean and lean for a decent estimate.

*** If using JSONC create a JSON file without comments for size estimates**

Leverage folder structures & JSON file names

- Subfolders per environment (Dev, Bet, Prd, …).
- Subfolders for parent & child policy rules.
- Different files and rules per environment or copy/paste to keep them the same.
- Use file names to filter out work in progress during deploys

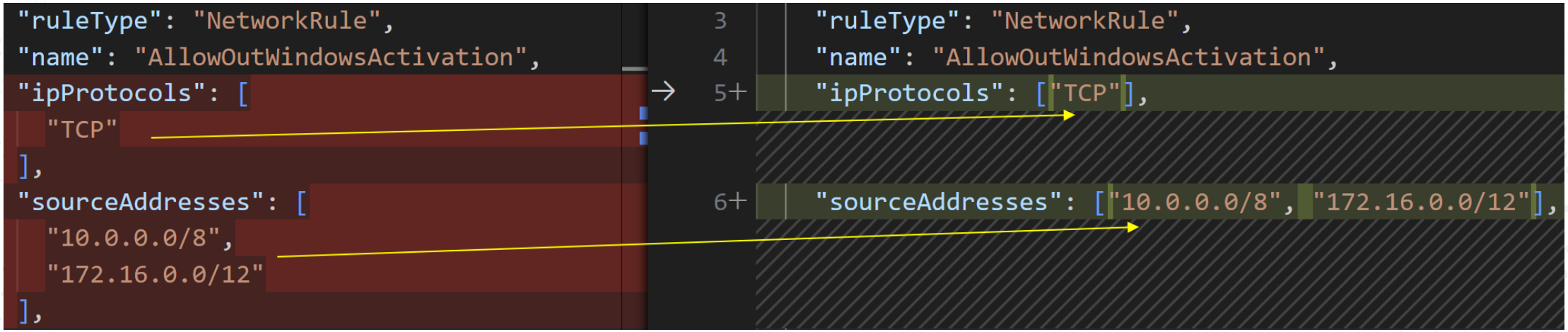


Stick to Azure JSON layout

- Use the JSON layout that Azure uses.
- Easier comparison between Azure JSON & your own JSON.
  - Helps avoid false positives
- It is complete and explicit. That way, a JSON definition block contains all you might need, allowing easy copy/paste when creating new ones.
- Being explicit helps with sizing estimates

```
{  I put the name and prioroty of a rule collection definition at the
   bottom. That is the same place Azure ARM templat JSON puts it
  "ruleCollectionType": "FirewallPolicyFilterRuleCollection",
                "action": {
    "type": "Allow"
  },
  "rules": [
    {
      "ruleType": "NetworkRule",
      "name": "AllowOutgoingRdp",
      "ipProtocols": [
        "TCP",
        "UDP"
      ],
      "sourceAddresses": [
        "10.0.0.0/8",
        "172.16.0.0/12",
        "192.168.0.0/16"
      ],
      "sourceIpGroups": [],
      "destinationAddresses": [
        "10.0.0.0/8",
        "172.16.0.0/12",
        "192.168.0.0/16"
      ],
      "destinationIpGroups": [],
      "destinationFqdns": [],
      "destinationPorts": [
        "3389"
      ]
    }
  ],
  "name": "AllowOutColRdpPrivateNetworks",
  "priority": 20040
}
```

Stick to Azure JSON formatting



- This allows for more correct identification between Azure JSON and your own JSON in VS Code file compares
- Less false positives due to mere formatting differences you need to check.
- I found and use the Meezilla extension for this.

Easy Azure FW rule creation & maintenance

- Yes, rules will be added via the portal in some environments and this comes in handy to add those to your IaC solution

Azure DevOps

Photo by Kelly Sikkema on Unsplash

Straightforward DevSecOps integration

- Simple security model: the repository is the security boundary
- We have a centrally maintained and managed DevOps Repo
- This repo has our PowerShell, Bicep, and YAML Pipeline
- Contains the generic, shared, as well as service & app-specific rules in JSON we maintain for customers
- If a parent AzFW Policy is used this is where those rules live
- SecOps is responsible for the maintenance & deployment.

Straightforward DevSecOps integration

- We keep dev teams out of our centrally maintained and managed DevOps Repo
- They don’t need to view or worry in any way about the JSON, PoSh, Bicep and YAML pipeline to make it all deploy and work
- They do need to provide input on what rules are needed by their apps.
- But what if dev teams want/need to create & maintain their own rules?
- Or gain visibility into their rules without needing to see everything?
- Give them their own repo and Git provides some integration options!

Photo by RealToughCandy.com: https://www.pexels.com/photo/person-holding-a-small-paper-110355309/

Git: open source distributed version control system

1. **Version Tracking**:
   - Git tracks changes to files, ensuring a comprehensive record of all modifications
   - This allows you to revert to specific versions when needed
2. **Performance**:
   - Git's lightweight design ensures efficient operations, making it ideal for small and large projects alike
3. **Flexibility**:
   - Git adapts well to various workflows and branching strategies.
   - Whether you work solo or collaborate with a team, Git has you covered.
4. **Security**:
   - Git employs cryptographic hashing to secure data integrity.
   - It also supports access controls and authentication mechanisms.

Git

5. **Distributed:**
   - Collaborate locally, globally, publicly and privately
6. **Wide Acceptance**:
   - Git is widely adopted across the software development industry
   - You cannot go wrong learning an using it
7. **Quality Open Source Project**:
   - Ensures continuous improvement and robustness

Git submodules

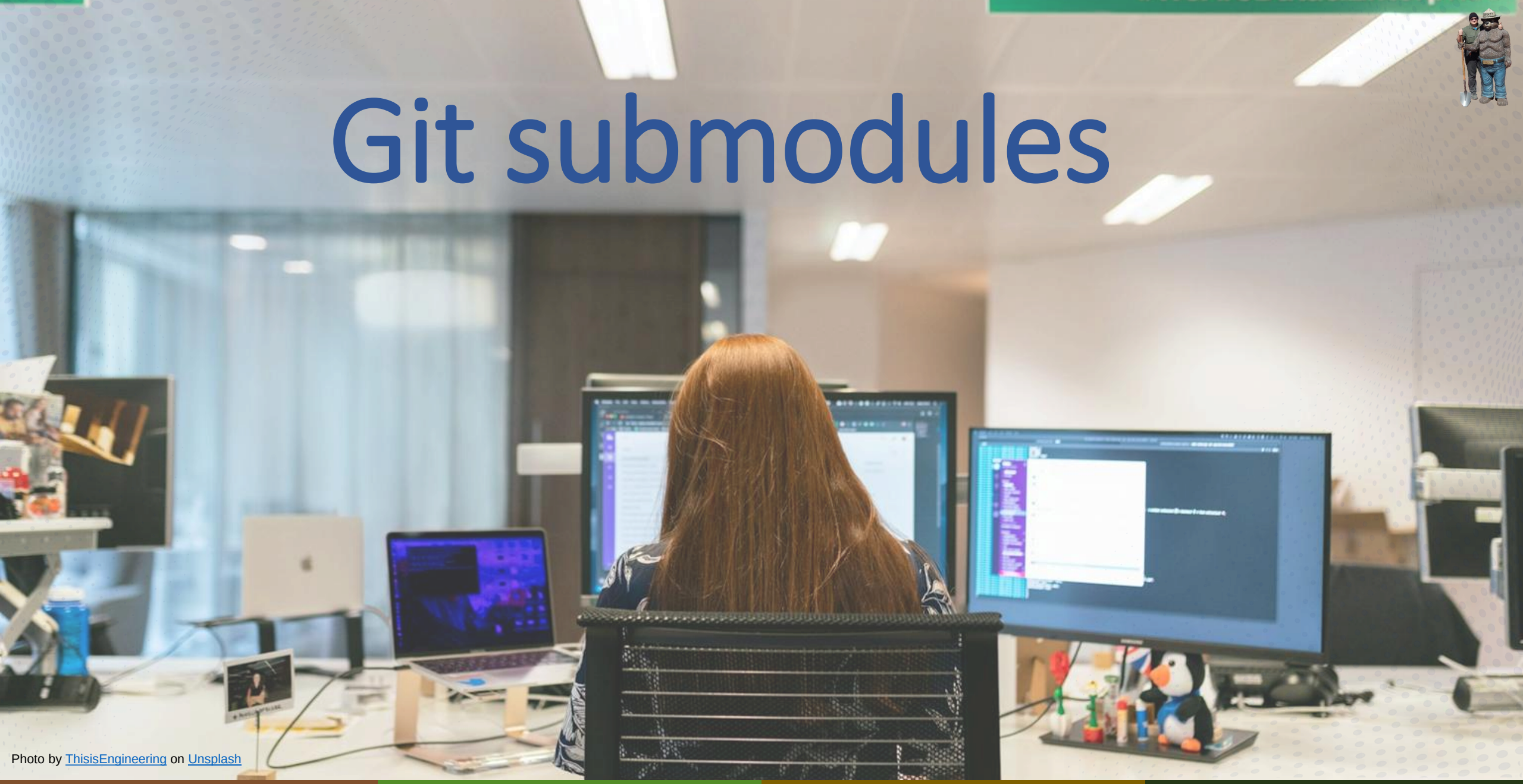


Photo by ThisisEngineering on Unsplash

Git Submodules to the rescue!

- Create a separate repository with only the folder structure for the firewall rule JSON files.
  - Per team, app, business unit, … or whatever works in your environment.
  - They create their branches, work on them & create pull requests for their work.
  - SecOps reviews the PRs and approves them when accepted
  - Merge them into “main”
- Devs can add the repo as a submodule to their app repository if so desired
- Super easy security model – the repository is the security boundary

Git Submodules to the rescue!

- We add those app-specific repos as git submodules to the SecOps repo.
- We update the submodules in our centralized Azure Firewall repo to get the latest rules
- It is our centralized repo that is deployed in the pipeline. Our PowerShell will read out JSON files to get our firewall rules and those in the submodule(s).

IAC › Up › Shared › bicep › AzureFwChildPolFleetMgnt › Dev › AfwChildPoliciesAndRules

- DevOps
- FleetManagement
- MaritimeShipping
- SecOpsCentralAzureFwPol
  - .git
  - .vscode
  - IAC
    - Down
    - Up
      - App
      - Shared
        - ARM
        - bicep
          - AzureFirewallPolicies
          - AzureFwChildPolFleetMgr
            - .vscode
            - Bet
            - Dev
              - AfwChildPoliciesAndRu
            - Prd
          - AzureFwChildPolMarShip
          - modules
        - Scripts
      - src
  - pipelines

Name	Status	Date modif
DND-DefaultApplicationRuleCollectionGroup.json		3/20/2024
DND-DefaultDnatRuleCollectionGroup.json		3/20/2024
DND-DefaultNetworkRuleCollectionGroup.json		3/20/2024
FleetMgmtChildAppRuleColGroups.json		3/20/2024
FleetMgmtChildDnatRuleColGroups.json		3/20/2024
FleetMgmtChildNetRuleColGroups.json		3/20/2024
forgit.txt		3/20/2024

Git submodules
So much to tell …

Article series:
https://www.starwindsoftware.com/blog/getting-git-submodules-in-private-azure-devops-repositories-to-work-in-a-pipeline/
https://www.starwindsoftware.com/blog/using-git-submodules-in-your-main-azure-devops-repository-part/
https://www.starwindsoftware.com/blog/using-git-submodules-in-your-main-azure-devops-repository-part-ii/

Photo by ThisisEngineering on Unsplash

Adding a Git submodule

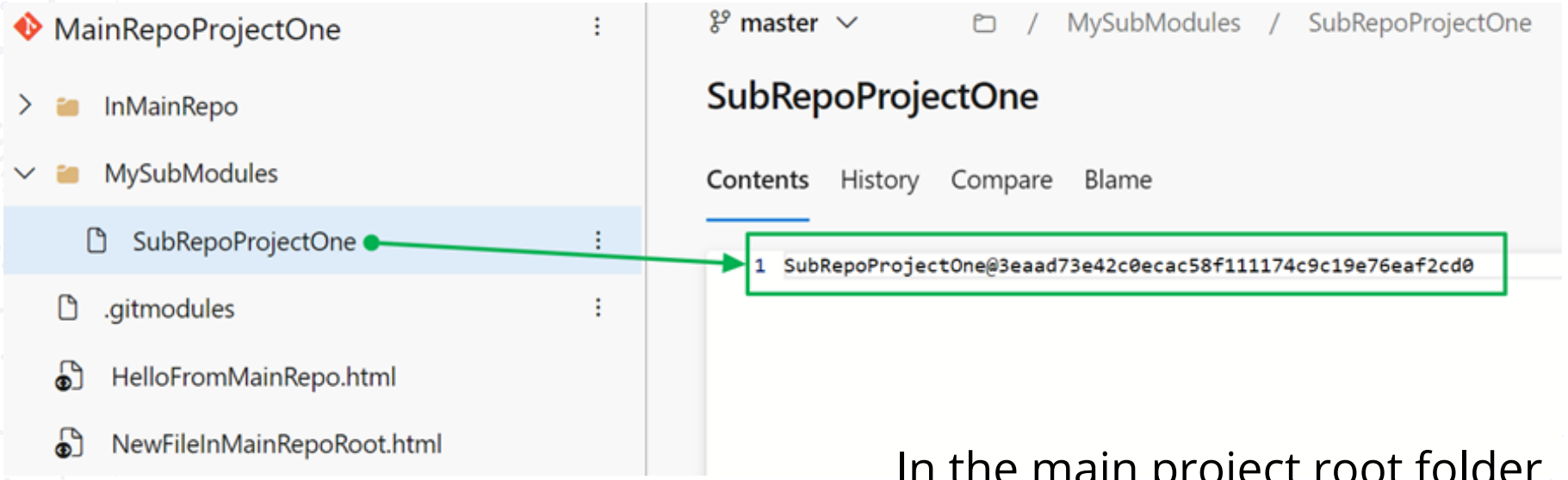
Adds a submodule with a repo in the same project in the same organization

git submodule add ../SubRepoProjectOne .\MySubModules\SubRepoProjectOne

Adds a submodule with a repo in a different project in the same organization

git submodule add ../../../ProjectTwo/_git/SubRepoProjectTwo .\MySubModules\SubRepoProjectTwo

How does a submodule work?



In the main project root folder

git submodule update --remote

git commit –m "Update SubModules"

git push

Update Git main repo

In the main project root folder

git add .

git commit –m “My comment”

git push

Update a Git submodule repo

- In submodule repo subfolder in the main project
- ***git add .***
- ***git commit –m “My comment”***
- ***git push origin HEAD:master***

https://carlosschults.net/en/git-detached-head

Delete a git submodule

- git rm MySubModule/SubRepoProjectOne
- rm .git/modules/MySubModules/SubRepoProjectOne -Recurse -Force
- git config --remove-section submodule.MySubModules/SubRepoProjectOne

Pipeline

Photo by Desola Lanre-Ologun on Unsplash

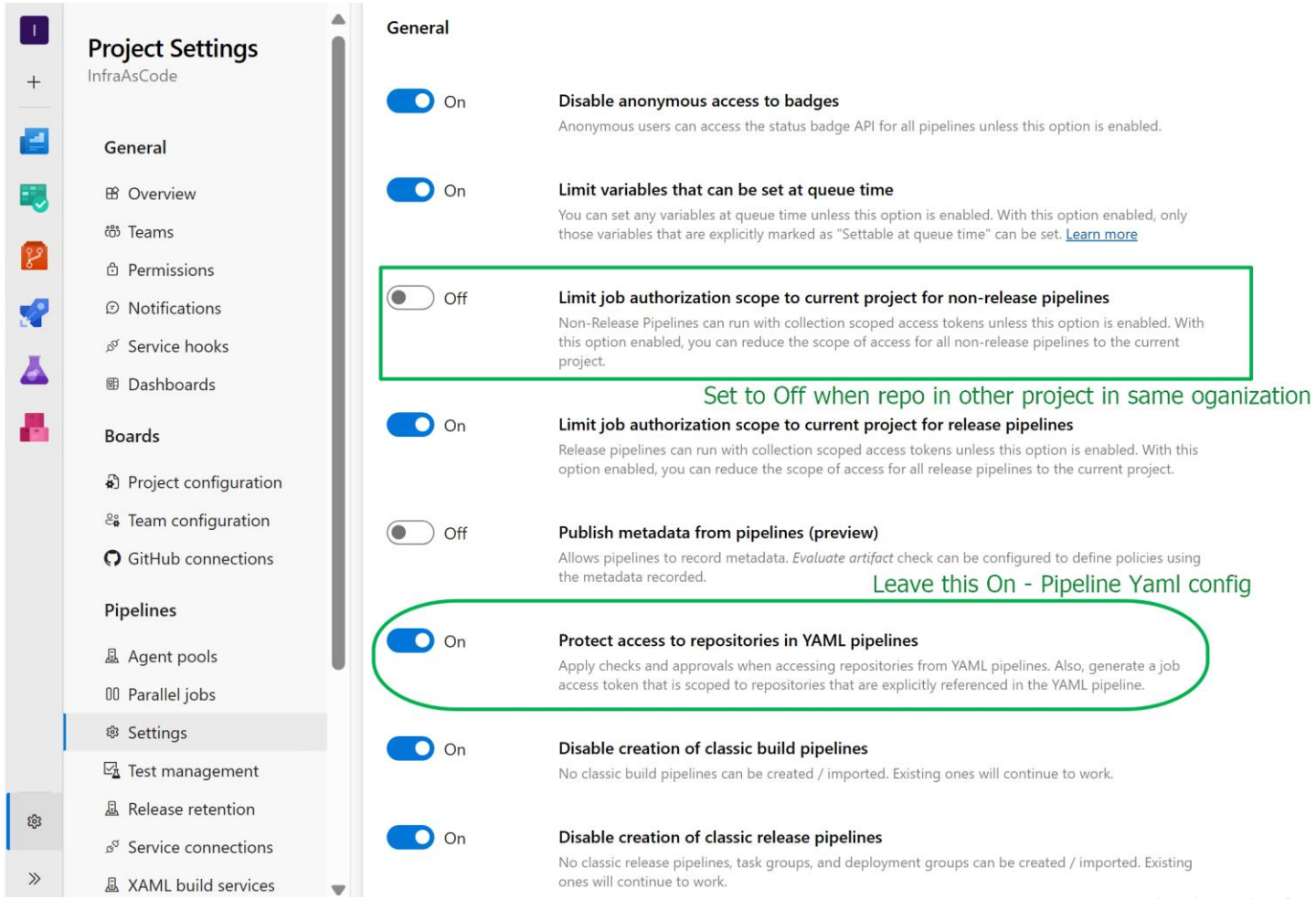
Define the submodule repos

```yaml
resources:
  repositories:
  - repository: AzureFwChildPolFleetMgntID
    type: git
    name: AzureFwChildPolFleetMgnt
  - repository: AzureFwChildPolMarShipID
    type: git
    name: AzureFwChildPolMarShip
```

Consume submodules

```yaml
stages:
  - stage: checkout
    jobs:
      - job:
        steps:
          - checkout: self
            submodules: true
          - task: PublishBuildArtifacts@1
            inputs:
              PathtoPublish: '$(Build.Repository.LocalPath)'
              ArtifactName: 'iac'
              publishLocation: 'Container'
        uses:
          repositories:
          - AzureFwChildPolFleetMgntID
          - AzureFwChildPolMarShipID
```

Project & Security settings



Takeaways from the field

- Security in Azure DevOps or GitHub is set per repository
- Create separate repos when needed to have security boundaries
- If security is less of a concern, such a repo can contain JSON for firewall rules of multiple apps/business units.
- Submodules in git can track branches if so desired, but there be dragons … if one is not disciplined
- Use branches in the repos but have the submodules track the main branch of their respective repos.

PowerShell

Photo by Christina Morillo: https://www.pexels.com/photo/woman-sitting-in-front-laptop-1181677

PowerShell to handle parameters

The full power of PowerShell scripting capabilities

Low barrier to entry, most people know PowerShell

Use via dot sourcing or modules in Azure DevOps pipelines

- Dot sourcing is easy and accessible
- Using modules is a bit of a rabbit hole & there are dragons
  - Experienced, skilled personnel with a well-taught out plan & process only

Validate Afw JSON against Schema file

```powershell
function Test-AzureFwRulesJson {
    [CmdletBinding()]
    [OutputType([System.Object[]])]
    param (
        [Parameter(Mandatory = $true)]
        [array]$Files,
        [Parameter(Mandatory = $true)]
        [String]$SchemaFile
    )
    $schemaFile = Join-Path $path 'AfwRulesSchema.json'
    $IsAfwRulesJsonValid = Get-ChildItem -Path $ParentFilePath -Filter *.jsonc -Recurse -Exclude "schema.json" | ForEach-Object {
        $json = Get-Content -Path $_.FullName | ConvertFrom-Json -Depth 100 | ConvertTo-Json -Depth 100
        #Check every Afw rules JSON file against the schema
        $result = ($json | Test-Json -SchemaFile $schemaFile)
        if (!$result) {
            throw "The Azure Firewall rules JSON file '$($_.FullName)' does not match the schema '$schemaFile'"
        }
        return $IsAfwRulesJsonValid
    }
}
```

Grab Parent Policy rules from main repo

```powershell
$ParentFilePath = "../bicep/AzureFirewallPolicies/$environment/AfwParentPoliciesAndRules/*"
$Files = Get-ChildItem -File $ChildPath -Include '*.jsonc' -exclude 'DONOTUSE*.jsonc', 'DND-*.jsonc'
$IsJsonValid = Test-AzureFwRulesJson -Files $Files -SchemaFile $SchemaFile
If ($IsJsonValid) {
    Write-Host -ForegroundColor Green "All Azure Firewall JSON files are valid"
    $AfwParentCollectionGroups = @()
    Foreach ($File in $Files) {
        $JSON = $Null # We use this with ConvertFrom-Json to validate that the JSON file is OK, but cannot use this to pass as a param to Bicep
        try {
            #region Handling jsonc - Option 1 - Only works with PowerShell Core
            write-host -ForegroundColor Cyan "Using ConvertFrom-Json on $($File.Name) to check for JSON errors"
            $JSON = Get-Content $File.FullName | ConvertFrom-Json | ConvertTo-Json -Depth 100
            write-host -ForegroundColor green "JSON in $File is valid, adding content to child policy firewall rules"
            # DO NOT PUT ConvertFrom-Json in here just by itself - the PSCustomObject is not serializable and passing this param to bicep will than be empty!
            $AfwParentCollectionGroups += $JSON
            #endregion
        }
        Catch {
            write-host -ForegroundColor Red "ConvertFrom-Json for $File threw an error. Check your JSON in the RCG/RC/R files"
            Exit
        }
        write-host -ForegroundColor green "Done reading $($File.Name)"
    }
}
else {
    Write-Host -ForegroundColor Red "One or more Azure Firewall rule JSON files are invalid"
}
```

Grab Child Policy rules from everywhere

```powershell
$ChildPolRulesPaths = @(
    "../bicep/AzureFirewallPolicies/$environment/AfwChildPoliciesAndRules/*",
    "../bicep/AzureFwChildPolMarShip/$environment/AfwChildPoliciesAndRules/*",
    "../bicep/AzureFwChildPolFleetMgnt/$environment/AfwChildPoliciesAndRules/*"
)
$AfwChildCollectionGroups = @()
foreach ($ChildPath in $ChildPolRulesPaths) {
    $Files = Get-ChildItem -File $ChildPath -include '*.json', '*.jsonc' -exclude 'DONOTUSE*.json', 'DND-*.json', 'DONOTUSE*.jsonc', 'DND-*.jsonc'
    write-host -ForegroundColor Cyan "All Azure Firewall Rule Collection Group files in $ChildPath will be tested against schema $SchemaFile."
    $IsJsonValid = Test-AzureFwRulesJson -Files $Files -SchemaFile $SchemaFile
    if ($IsJsonValid) {
        write-host -ForegroundColor Green "All Azure Firewall Rule Collection Group files in $ChildPath are valid according to schema $SchemaFile."
        Foreach ($File in $Files) {
            $JSON = $Null
            try {
                write-host -ForegroundColor yellow "Using ConvertFrom-Json on $($File.Name) to check for JSON errors"
                $JSON = Get-Content $File.FullName | ConvertFrom-Json  | ConvertTo-Json -Depth 100
                write-host -ForegroundColor green "JSON in $File is valid, adding content to child policy firewall rules"
                # DON'T put ConvertFrom-Json in here just by itself - the PSCustomObject is not serializable & passing this param to bicep will than be empty!
                $AfwChildCollectionGroups += $JSON
            }
            Catch {
                write-host -ForegroundColor Red "ConvertFrom-Json for $File threw an error. Check your JSON in the RCG/RC/R files"
                Break
            }
            write-host -ForegroundColor blue "Done reading $($File.Name)"
        }
    }
    else {
        write-host -ForegroundColor Red "One or more Azure Firewall Rule Collection Group files are INVALID according to schema $SchemaFile."
        Break
    }
}
```

Parent Policy definition in main repo

```
If  $AfwParentCollectionGroups.count -eq 0 -or $AfwChildCollectionGroups.count -eq 0) {
    write-host "$AfwChildCollectionGroups is empty - that will not work - aborting"
}
Else {
# Azure Firewall parent policy
$firewallParentPolicy = @{}
$firewallParentPolicy.PolicyName = "pol-par-fw-$($vwanHub.Name)- $($templateSettings.environment)"
$firewallParentPolicy.Tier = 'premium'
$firewallParentPolicy.DnsProxyEnabled = $true
$firewallParentPolicy.DnsServers = $AdditionalParameters.DnsProxyServerIpAddresses
$firewallParentPolicy.ThreatIntelMode = 'Alert'
$firewallParentPolicy.ThreatIntelFqdns = @('microsoft.com', 'workinghardinit.work')
# Example only - not a requirement or good idea
$firewallParentPolicy.ThreatIntelIps = @('185.185.185.185', '192.195.195.195')
# Azure Firewall parent Rule Collection Groups
$firewallParentPolicy.RuleCollectionGroups = $AfwParentCollectionGroups
```

Child Policy definition in main repo

```
#Azure Firewall child policies
$firewallChildPolicy = @{}
$firewallChildPolicy.PolicyName = "pol-chi-fw-$($vwanHub.Name)-$($templateSettings.environment)"
$firewallChildPolicy.Tier = 'premium'
$firewallChildPolicy.DnsProxyEnabled = $true
$firewallChildPolicy.DnsServers = $AdditionalParameters.DnsProxyServerIpAddresses
$firewallChildPolicy.ThreatIntelMode = 'Alert'
$firewallChildPolicy.ThreatIntelFqdns = @('microsoft.com', 'workinghardinit.work’)
#Example only - not a requirement or good idea
$firewallChildPolicy.ThreatIntelIps = @('185.185.185.185', '192.195.195.195 ’)
#Example only - Firewalls on premises for example
$firewallChildPolicy.RuleCollectionGroups = $AfwChildCollectionGroups
```

Kick off the deployment

```
$inputParameters = @{
   'templateSettings'              = $templateSettings;
   'firewallParentPolicy'          = $firewallParentPolicy;
   'firewallChildPolicy'           = $firewallChildPolicy;
   'applicationResourceGroup'      = $target.resourceGroupName;
   }

# Do the actual deployment
if ($pscmdlet.ShouldProcess("Subscription " + $target.subscriptionId, "Deploy-AzLinkedTemplate")){
   Deploy-AzLinkedTemplate -inputParameters $inputParameters `
      -sharedStorageSettings $shared `
      -templateTarget $target
   }

[New-AzResourceGroupDeployment]
```

ConvertFrom-Json is not serializable

- There is no error in PowerShell – the parameter is just empty
- The Bicep deployment does bark at us

```
Line |
  30 |         New-AzResourceGroupDeployment @params -DeploymentDebugLogLeve …
     |         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     | 2:46:20 PM - Error: Code=InvalidTemplate; Message=Deployment template validation failed: 'The template variable 'ChildRCGs' is not valid: The language expression property 'name' doesn't exist, available properties are ''.. Please see
     | https://aka.ms/arm-functions for usage details.'.
```

https://blog.workinghardinit.work/2023/04/03/convertfrom-json-is-not-serializable/

ConvertFrom-Json is not serializable

- Pass the JSON file as raw text, a string, which is serializable
- In Bicep we use the json() function to convert the string to JSON
- No need for LoadTextContext(), which cannot use dynamic parameters

```
// Roll out the parent Rule Collection Group(s)
var ParentRCGs = [for (rulecol, index) in
firewallParentpolicy.RuleCollectionGroups: {
  name: json(rulecol).name
  properties: json(rulecol).properties
}]
```

https://blog.workinghardinit.work/2023/04/03/convertfrom-json-is-not-serializable/

Bicep code

Photo by ThisisEngineering on Unsplash

Bicep is less complicated & easier to maintain

- The actual Bicep code is:
  - Short
  - Simple
  - Easy to read and maintain

Deploy AzFW Parent Policy

```
/* Deploy the parent firewall policy */
resource firewallParentPolicyWEU 'Microsoft.Network/firewallPolicies@2023-04-01' = {
  name: firewallParentpolicy.PolicyName
  location: location
  tags: {
    department: templateSettings.department
    environment: templateSettings.environment
    projectCode: templateSettings.projectCode
    projectName: templateSettings.projectName
    managedBy: templateSettings.managedBy
    tier: templateSettings.tier
    dataProfile: templateSettings.dataProfile
    division: templateSettings.division
  }
  dependsOn: []
  properties: {
    sku: {
      tier: firewallParentpolicy.Tier
    }
    dnsSettings: {
      enableProxy: firewallParentpolicy.DnsProxyEnabled
      servers: firewallParentpolicy.DnsServers
    }
  }
}
output firewallParentPolicyWEU string = 'Created AFW parent policy ${firewallParentPolicyWEU.name} with ID:
${firewallParentPolicyWEU.id}'
```

Deploy AzFW Child Policy (1/2)

```
/* Deploy the child firewall policy */
resource firewallChildPolicyWEU 'Microsoft.Network/firewallPolicies@2023-04-01' = {
  name: firewallChildpolicy.PolicyName
  location: location
  dependsOn: [firewallParentPolicyWEUColGroups]
  tags: {
    department: templateSettings.department
    environment: templateSettings.environment
    projectCode: templateSettings.projectCode
    projectName: templateSettings.projectName
    managedBy: templateSettings.managedBy
    tier: templateSettings.tier
    dataProfile: templateSettings.dataProfile
    division: templateSettings.division
  }
  properties: {
    sku: {
      tier: firewallChildpolicy.Tier
    }
```

Deploy AzFW Child Policy (2/2)

```
    basePolicy: {
      id: firewallParentPolicyWEU.id
    }
    dnsSettings: {
      enableProxy: firewallChildpolicy.DnsProxyEnabled
      servers: firewallChildpolicy.DnsServers
    }
    threatIntelMode: firewallChildpolicy.ThreatIntelMode
    threatIntelWhitelist: {
      fqdns: firewallChildpolicy.ThreatIntelFqdns
      ipAddresses: firewallChildpolicy.ThreatIntelIps
    }
  }
}
output firewallChildPolicyWEU string = 'Created AFW child policy ${firewallChildPolicyWEU.name}
with ID: ${firewallChildPolicyWEU.id}'
```

Put parent rules collection groups into an array

```
// Grab all Parent Rule Collection Group(s) in to an array
var ParentRCGs = [for (rulecol, index) in
firewallParentpolicy.RuleCollectionGroups: {
  name: json(rulecol).name
  properties: json(rulecol).properties
}]

output ParentRCGs array = [for (rulecol, index) in
firewallParentpolicy.RuleCollectionGroups: {
  rulecollectiongroups: json(rulecol)
}]
```

Deploy all parent policy rules

```
// Roll out the parent Rule Collection Group(s)
@batchSize(1)
resource firewallParentPolicyWEUColGroups
'Microsoft.Network/firewallPolicies/ruleCollectionGroups@2023-04-01'
= [for (parrcg, index) in ParentRCGs: {
  parent: firewallParentPolicyWEU
  name: parrcg.name
  properties: parrcg.properties
}]
output firewallParentPolicyWEUColGroups array = [for (parrcg, index)
in ParentRCGs: {
  RCGParent: 'Created AFW Parent Rule Policy groups ${parrcg.name}
with ${parrcg.properties}'
}]
```

Put child rules collection groups into an array

```
// Grab all Child Rule Collection Group(s) in to an array
var ChildRCGs = [for (rulecol, index) in firewallChildpolicy.RuleCollectionGroups: {
  name: json(rulecol).name
  properties: json(rulecol).properties
}]

output ChildRCGs array = [for (rulecol, index) in firewallChildpolicy.RuleCollectionGroups: {
  rulecollectiongroups: json(rulecol)
}]
```

Deploy all child policy rules

```
// Roll out the child Rule Collection Group(s)
@batchSize(1)
resource firewallChildPolicyWEUColGroups
'Microsoft.Network/firewallPolicies/ruleCollectionGroups@2023-04-01' =
[for (childrcg, index) in ChildRCGs: {
  parent: firewallChildPolicyWEU
  name: childrcg.name
  dependsOn: [firewallParentPolicyWEUColGroups]
  properties: childrcg.properties
}]
output firewallChildPolicyWEUColGroups array = [for (childrcg, index) in
ChildRCGs: {
  RCGChild: 'Created AFW Parent Rule Policy groups ${childrcg.name} with
following properties: ${items((childrcg.properties))}'
}]
```

DEMO TIME

Photo by Tom Fisk: https://www.pexels.com/photo/aerial-shot-of-cargo-ship-on-sea-3840441/

To Recap

- I shared with you how I handle Azure Firewall rules in real life
- Perfection is the enemy of the good
- It functions well and has a low threshold for adoption
- Keeps Bicep simple and free from hard-coded paths and values
- PowerShell allows folder structure & file name as a force multiplier
- Integrates well into DevSecOps with Git submodules
- No need to grant to grant extra rights to extra people on shared, critical infrastructure

Conclusion

- We have too many piecemeal examples and quick starts
- We need to share more real-world, detailed tactics and examples
- Feedback is welcome
- Not the only way of doing things …

Any questions?