

PowerShell Conference Europe 2019

Hannover, Germany

June 4-7, 2019

Test Infrastructure as Code?

MARK WARNEKE

pscone.eu

Platinum
Sponsor



After this Session...

I am able to create a test-suite for an Infrastructure as Code project from scratch.

I can articulate why Infrastructure as Code testing is necessary and increases the quality and reliability of provisioned services.

I have the ability to create an Infrastructure as Code project from scratch, quicker and much more mature.



Agenda

- Introduction to Infrastructure as Code
- DevOps foundations
- Quality & Maturity Framework
- Running Test Code (Hopefully)

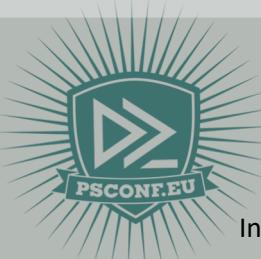


“
Infrastructure as code is an approach to infrastructure automation **based on practices from software development.**



It emphasizes consistent, repeatable routines for provisioning and changing systems and their configuration.

– Kief Morris



Infrastructure as Code Kief Morris 9781491924358

 @MarkWarneke

AppDev – InfraDev



Application Development



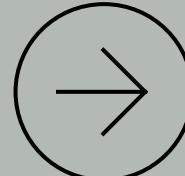
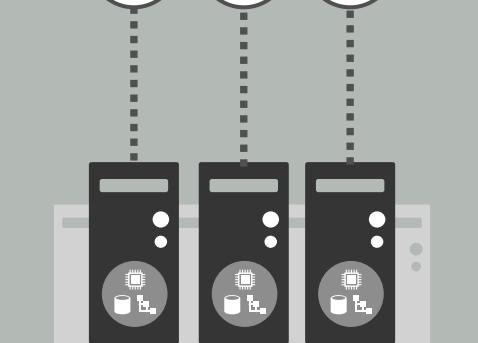
Infrastructure Development



IaaS - SaaS



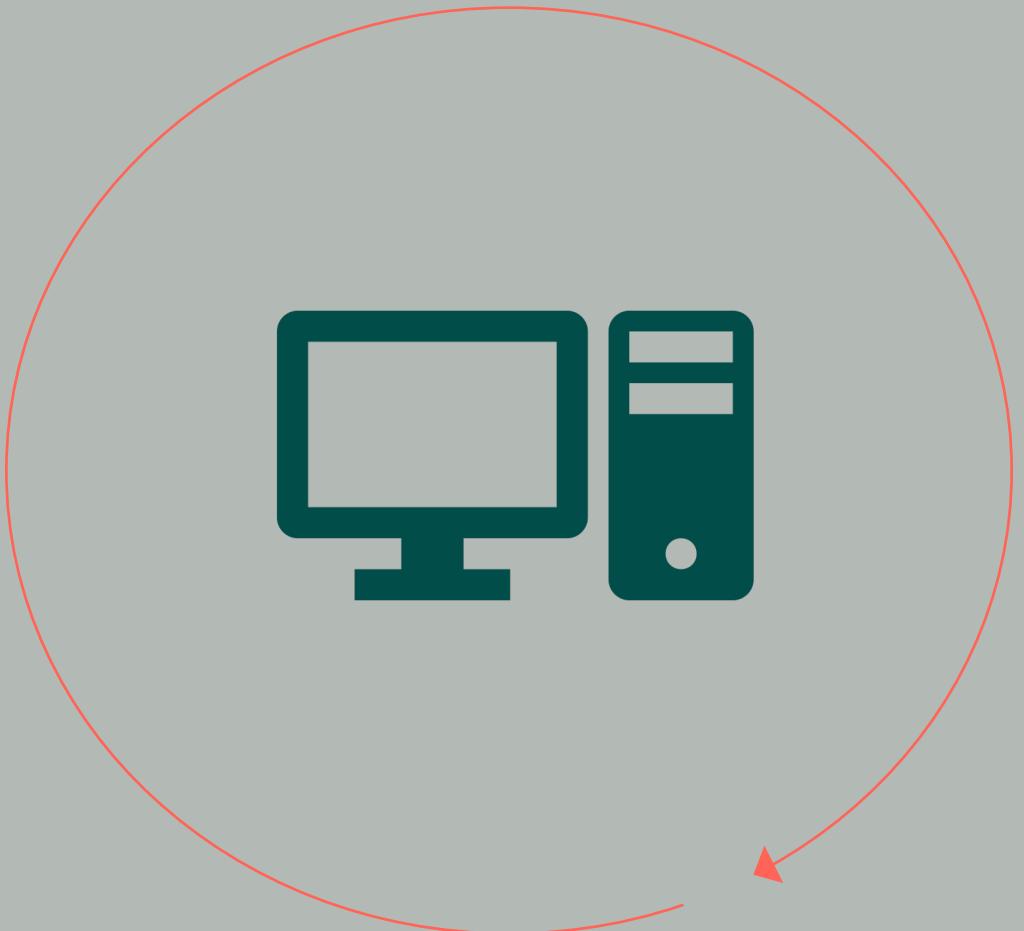
Servers



Services



Outside – Inside View



Outside

Hardware Configuration

- VM Size, Disks, Network
- RBAC, secrets etc.

Inside

Software

- Desired state
- Extensions & scripts

→ @IrwinStrachan
→ @devblackops (Brandon Olin)



@MarkWarneke

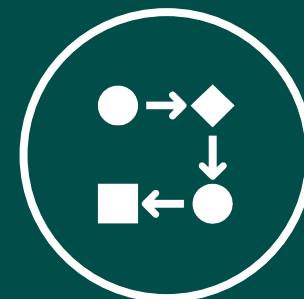
Approaches

Declarative vs Imperative



functional

describe final state



procedural

executing steps to get to final state



@MarkWarneke



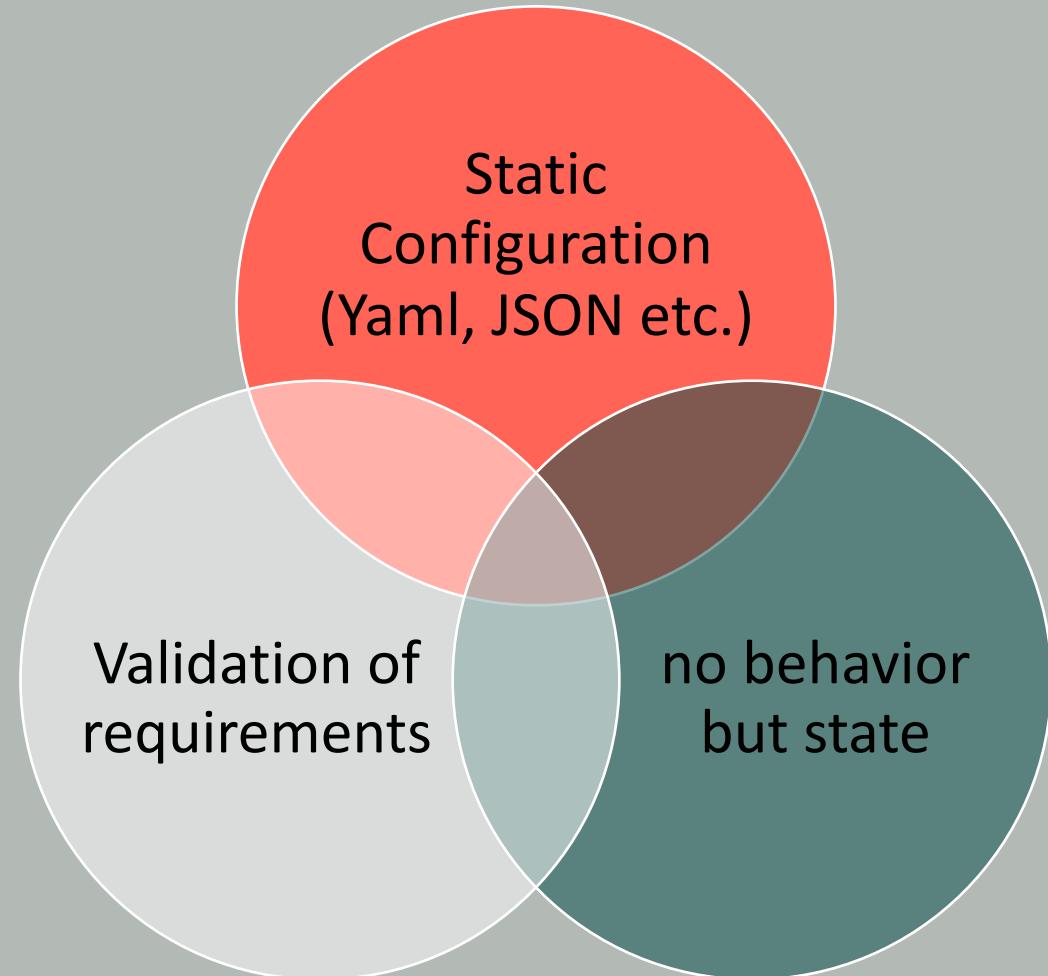
treat your servers
like cattle, not pets

– Bill Baker

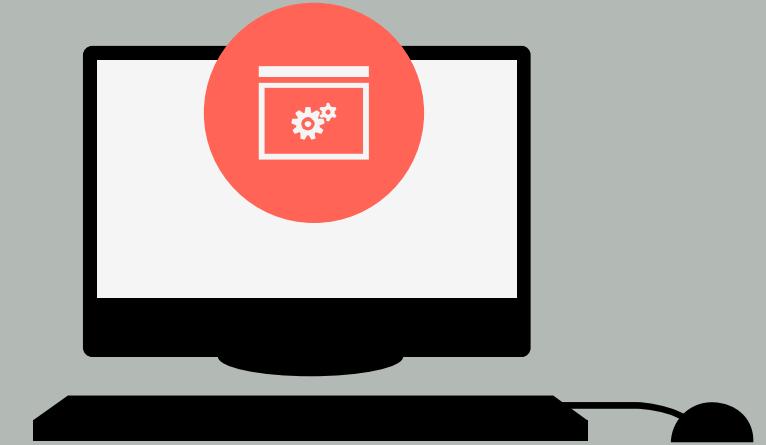
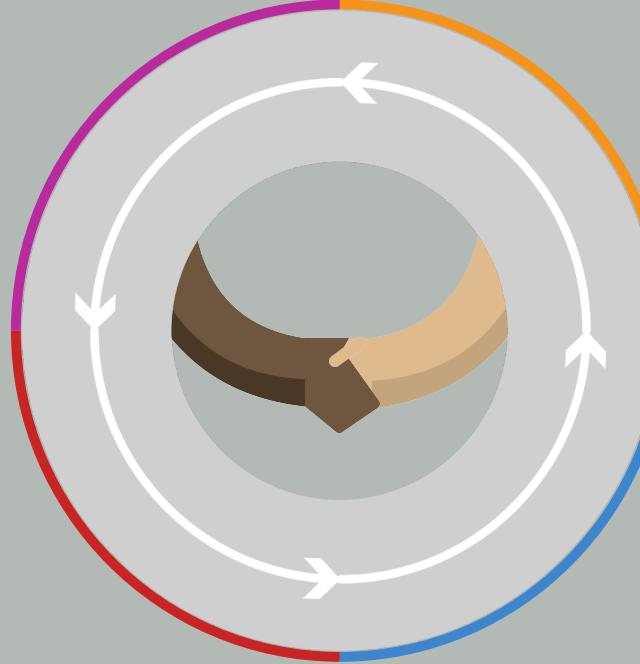
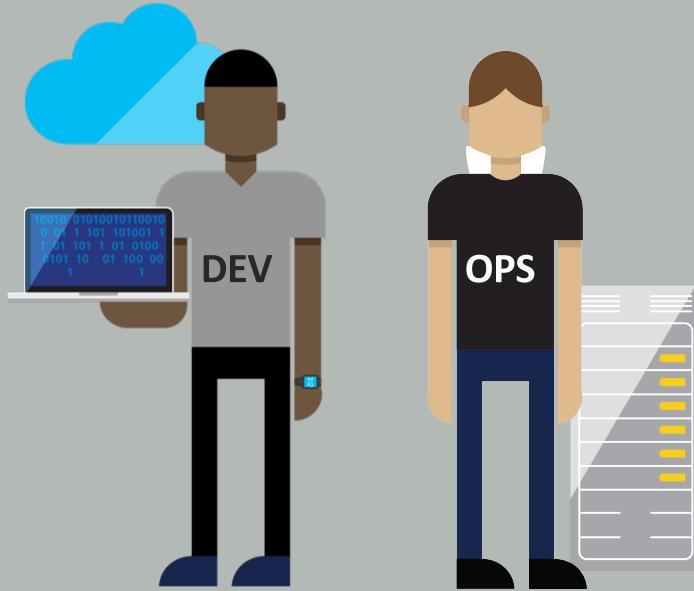


@MarkWarneke

Problem Statement



DevOps = People + Process + Tools



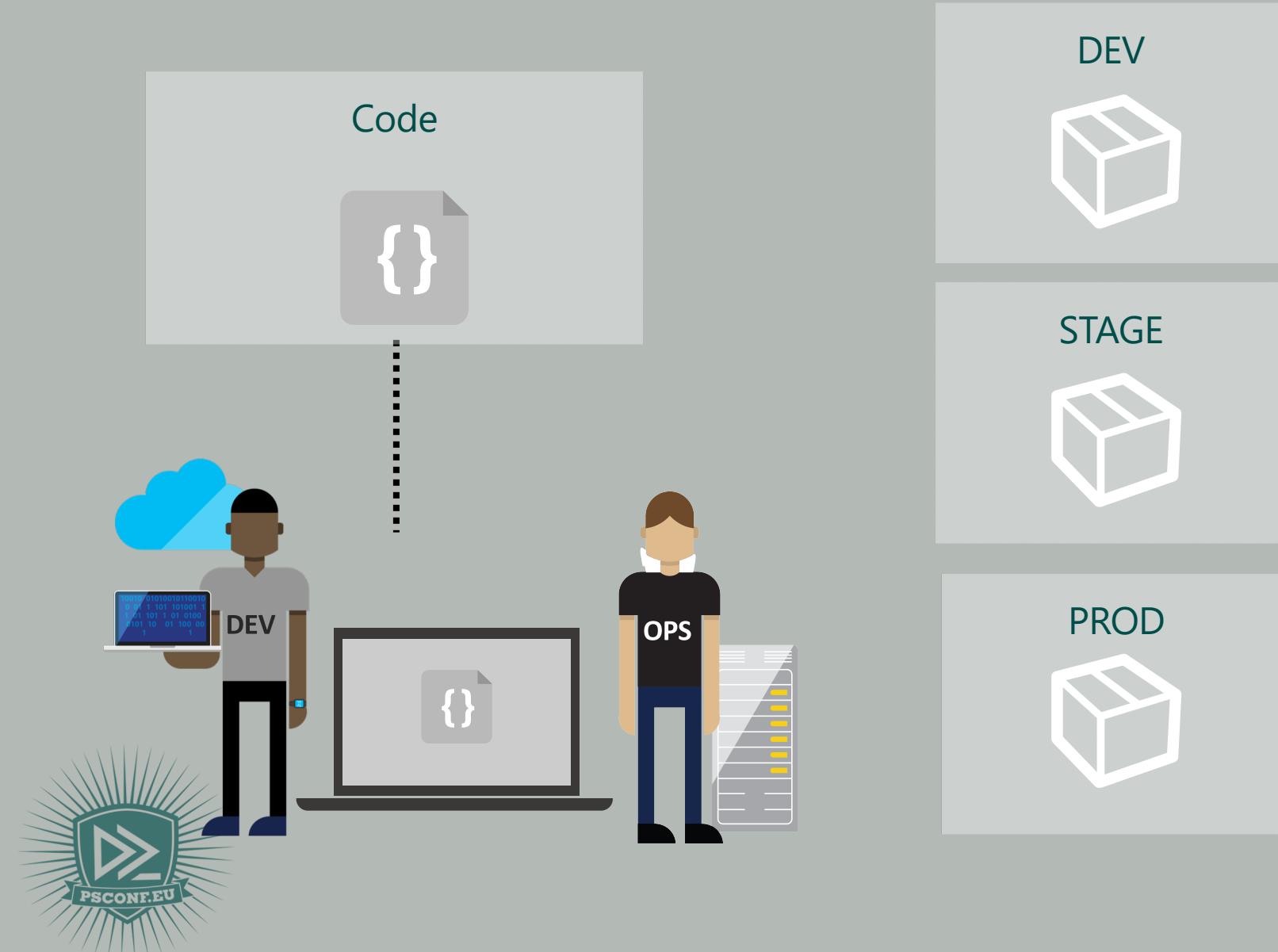
1 | People

2 | Process

3 | Tools



Infrastructure as Code



Value

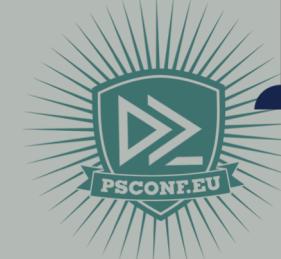
- Optimized Resources
- Accelerate Delivery

Measure

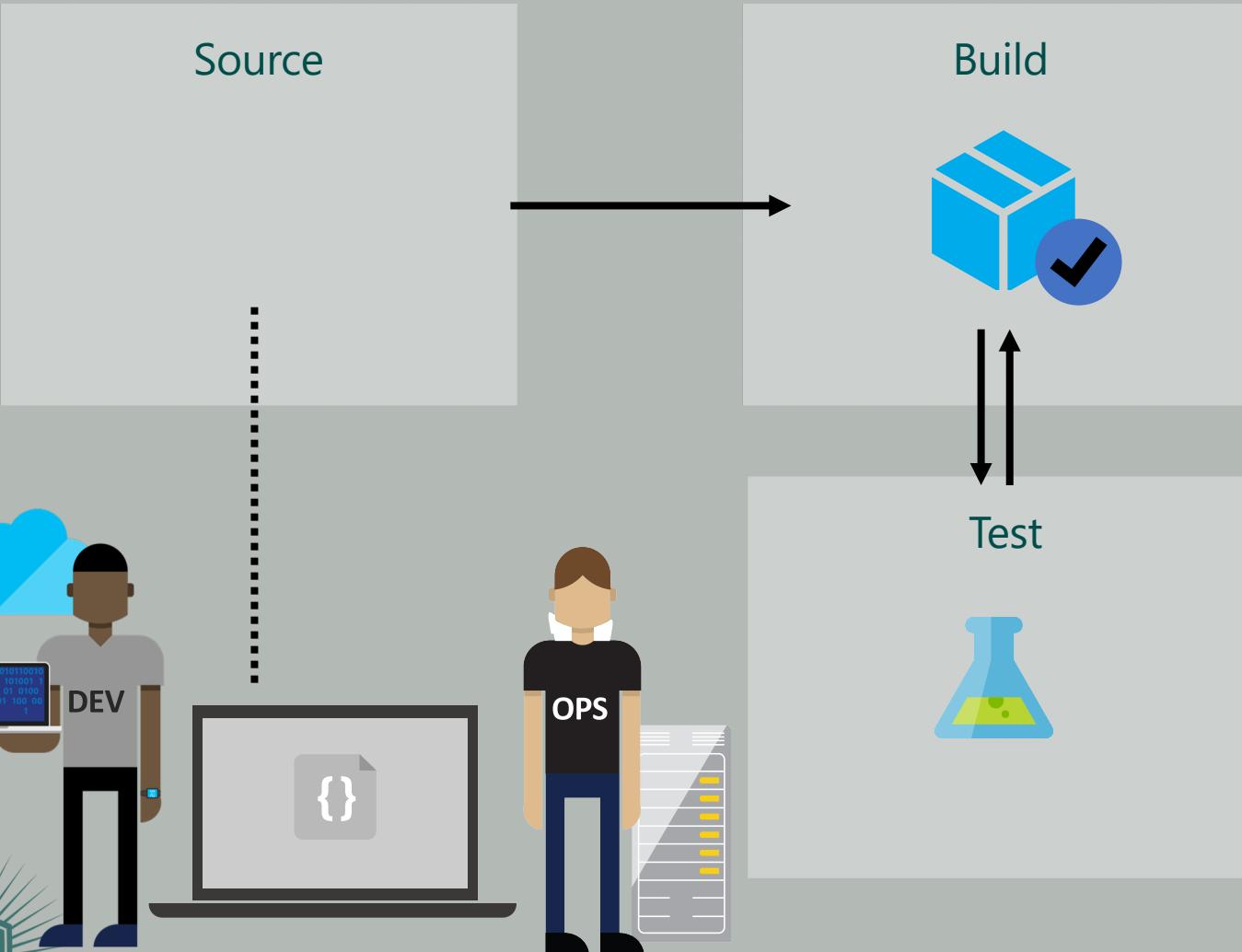
- Deployment Rate
- Mean Time To Release



@MarkWarneke



Continuous Integration



Value

- Accelerate Delivery
- Repeatability
- Optimized Resources

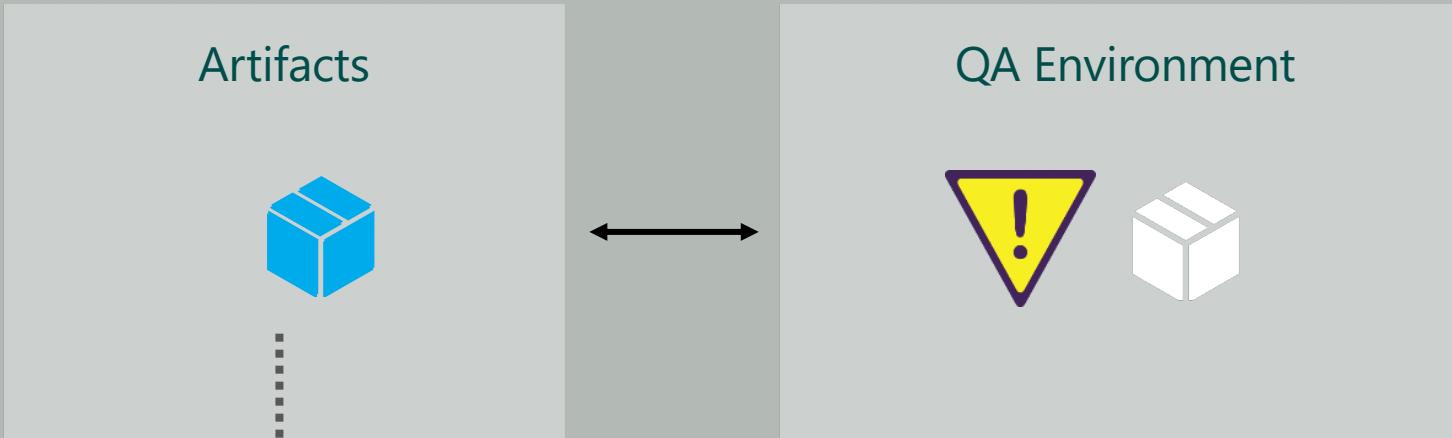
Measure

- More frequent releases
- Mean Time To Release
- Mean Time To Deploy



@MarkWarneke

Continuous Deployment



Value

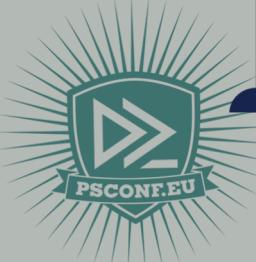
- Optimized Resources
- Accelerate Delivery

Measure

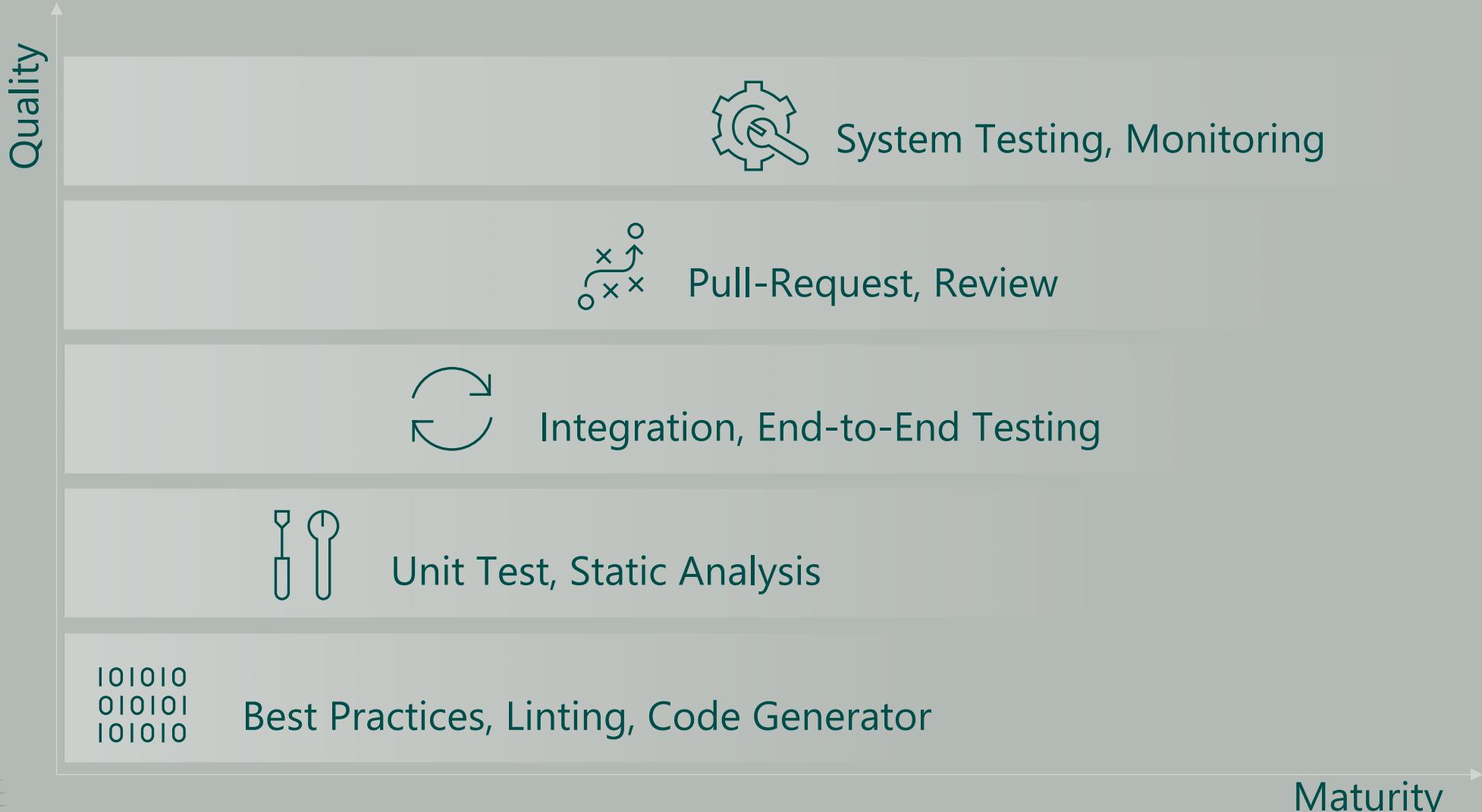
- Deployment Rate
- Mean Time To Release
- Availability



@MarkWarneke



Quality & Maturity



@MarkWarneke





I Am Developer

@iamdevloper



10 lines of code = 10 issues.

500 lines of code = "looks fine."

Code reviews.

10:58 - 5. Nov. 2013



115



8,3 Tsd.



5,6 Tsd.



<https://twitter.com/iamdevloper/status/397664295875805184?lang=en>



@MarkWarneke

DEMO

- 📝 aka.ms/az.new
- 📋 aka.ms/az.new/resources

PSCONF.EU



@MarkWarneke

DEMO

VSCode Extensions & Liniting

PSCONF.EU



@MarkWarneke



```
Describe "Get-Name function parameter validation" -Tags Unit {

    It "Should return name by convention" {

        $Company = 'PSConf'
        $Environment = 'TEST'

        $ComponentName = 'psconf-test'

        $name = @{
            Company $Company
            Environment $Environment
        }

        $AutomationAccountName = Get-xAz.AutomationAccountName @Name

        $AutomationAccountName | should BeLike "$Company*"
        $AutomationAccountName | should BeLike "*$Environment*"
        $AutomationAccountName | should BeLike "*-*" -Because 'Name should use delimiter'
        $AutomationAccountName | should Not BeLike "*-" -Because 'Name should not end in delimiter'
        $AutomationAccountName | should BeExactly $ComponentName
    }
}
```



```
# Ensure valid json -> azuredeploy.json

param (
    $Path = 'azuredeploy.json'
)

Describe "..." {

    try {
        $text = Get-Content $Path -Raw -ErrorAction Stop
        $json = ConvertFrom-Json $text -ErrorAction Stop
    }
    catch {
        $JsonException = $_
    }

    it "should throw no exception" {
        $JsonException | Should -BeNullOrEmpty
    }
}
```





```
# Validate presence of mandatory attributes
$TestCases = @(
    @{
        Expected = "parameters"
    },
    @{
        Expected = "variables"
    },
    @{
        Expected = "resources"
    },
    @{
        Expected = "outputs"
    }
)

it "should have <Expected>" -TestCases $TestCases {
    param(
        $Expected
    )
    $json.PSObject.Members.Name | Should -Contain $Expected
}
```





```
# Ensure metadata (description) is present
context "parameters tests" {
    $parameters = $json.parameters | Get-Member -MemberType NoteProperty

    foreach ($parameter in $parameters) {
        $ParameterName = $($parameter.Name)
        it "$ParameterName should have metadata" {
            $json.parameters.$ParameterName.metadata | Should -Not -BeNullOrEmpty
        }
    }
}
```



```
# Ensure the order of properties within azuredeploy.json
context "resources structure test" {
    foreach ($resource in $json.resources) {
        it "should follow comment > type > apiVersion > name > properties" {
            "$resource" | Should -BeLike "*comments*type*apiVersion*name*properties*"
        }
    }
}
```



DEMO

WhatIf

PSCONF.EU





```
Describe "New-xAz.NetVnetComponent WhatIf" -Tags Unit {

    InModuleScope $script:ModuleName {
        It "should return valid parameter" {

            $Location = 'westeurope'
            $VnetName = 'Net-01-WE'
            $TestResourceGroupName = 'Dev-Vnet'

            Mock Get-AzResourceGroup { return @{ ResourceGroupName = 'Dev-Vnet' } }

            $InputObject = @{
                #...
            }
            $OutputObject = New-xAz.NetVnetComponent @InputObject -WhatIf

            $OutputObject.VirtualNetwork | Should Be $VnetName
            $OutputObject.ResourceGroupName | Should Be $TestResourceGroupName
        }
    }
}
```

DEMO

Integration





```
Describe "New-xAz.KvltComponent integration tests" -Tags Build {
    # ...
    try {
        $OutputObject = New-xAz.KvltComponent @InputObject -ErrorAction Stop
        Write-Verbose -Message ("Got outputobject {0}" -f $OutputObject)

        $Vault = Get-AzKeyVault $ComponentName
    }
    catch {
        $Exception = $_
        Write-Verbose -Message $Exception.Exception
    }

    it "should not throw an exception" {
        $Exception | Should -BeNullOrEmpty
    }

    it "should have firewall" {
        $Vault.NetworkAcls.IpAddressRanges.Count | Should -BeGreaterOrEqual 1
    }

    it "should have access policies" {
        $Vault.AccessPolicies.Count | Should -BeGreaterOrEqual 1
    }
}
```





```
Connect-AzAccount
```

```
Describe "how to clean up" {
    $TestResourceGroupName = "AUTOTEST-$((Get-Date -Format FileDateTime))"
    $Location = 'WestEurope'

    BeforeEach {
        Write-Host "Create Test ResourceGroup $TestResourceGroupName..." -ForegroundColor Blue
        $null = New-AzResourceGroup -Name $TestResourceGroupName -Location $Location
        Write-Host "Test started." -ForegroundColor Blue
    }

    it "should cleanup" { "A" | Should Be "A" }

    AfterEach {
        Write-Host "Remove ResourceGroup $TestResourceGroupName..." -ForegroundColor Blue
        Get-AzResourceGroup -Name $TestResourceGroupName -ErrorAction SilentlyContinue |
            Remove-AzResourceGroup -Force
        Write-Host "Test completed." -ForegroundColor Blue
    }
}
```

```
# Pass configuration file to test
param(
    [string] $ConfigurationFile = 'config.test.json'
)

$script:ModuleName = 'xAz.La' # log analytics module

# Get the config values from file
$Path = Join-Path $PSScriptRoot $ConfigurationFile
$script:config = (Get-Content -Raw $Path) | ConvertFrom-Json

# Test Case
Describe "..." {

    $InputObject = @{
        Environment = $config.Environment
        # other parameter
    }
    $OutputObject = New-xAz.LaComponent @InputObject -Verbose

    it "..." {
        $OutputObject | Should # ...
    }
}
```



DEMO

Azure DevOps



Azure DevOps YAML

<https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=schema>



```
# Trigger pipeline, can provide wildcard  
# and file path  
trigger:  
- master  
  
# Variables for pipeline e.g. module name  
variables:  
    azureSubscription: "Mark"  
    feed.name: "xAz"  
    organization: "az-new"  
    module.name: "xAz.Cosmos"  
  
# one or more jobs (job per agent)  
jobs:  
    # could be self hosted agent  
    - job: Build_PS_Win2016  
        pool:  
            vmImage: vs2017-win2016  
  
        steps:  
            - checkout: self  
                persistCredentials: true  
  
            - task: AzurePowerShell@4  
            # ...  
            # more tasks with configuration
```





```
- task: AzurePowerShell@4          # Version 4 in preview for az module
  inputs:
    azureSubscription: $(azureSubscription)      # Scope could be handled (RG or Subscription)
    scriptType: "FilePath"
    scriptPath: $(Build.SourcesDirectory)\$(Module.Name)\psake.ps1
    scriptArguments: -TaskList Test -Verbose
    azurePowerShellVersion: "latestVersion"
    errorActionPreference: "continue"           # Result will handle termination

- task: PublishTestResults@2
  inputs:
    testRunner: 'NUnit'
    testResultsFiles: '**/TestResults.module.xml'
    testRunTitle: 'PS_Win2016_Module'
    failTaskOnFailedTests: true                 # Leave empty if test result should not terminate
    displayName: 'Publish Module Test Results'
    condition: in(variables['Agent.JobStatus'], 'Succeeded', 'SucceededWithIssues', 'Failed')
```



```
Task Test -Depends Init, PrepareTest {

    $Folder = "Unit" # "Module", "Integration" ...

    foreach ($module in $ModuleBase) {
        # Execute tests
        $moduleRoot = Join-Path -Path $ProjectRoot -ChildPath $module
        $testScriptsPath = Join-Path -Path $moduleRoot -ChildPath 'test' | Join-Path -ChildPath $Folder
        $testResultsFile = Join-Path -Path $ProjectRoot -ChildPath "TestResults.$Folder.xml"

        $pester = @{
            Script      = $testScriptsPath
            OutputFormat = 'NUnitXml'          # Expected Format of Test Result Publisher
            outputFile  = $testResultsFile # location on build agent path -> For Result Publish
            PassThru    = $true
            ExcludeTag  = 'Incomplete, Unit'
        }
        $null = Invoke-Pester @pester
    }
}
```

x xAz.New +

✓ #20190212.9: fix interactive prompt

Release

All logs

⋮

Overview

Boards

Repos

Pipelines

Builds

Releases

Library

Task groups

Deployment groups

Test Plans

Artifacts

Compliance

Logs Summary Tests

Summary

3 Run(s) Completed (3 Passed, 0 Failed)

1,405

Total tests

+1,405

1,405
0
0
● Passed
● Failed
● Others

100%

Pass percentage

↑ 100%

45s 63ms

Run duration ⓘ

↑ +45s 63ms

Bug ▾ Link

Test run ▾ Column Options

T

Filter by test or run name

Test file ▾ Owner ▾ Outcome: Aborted (+1) ▾ X



Hooray! There are no test failures.

Change the test outcome filter to view tests relevant to you.

Project settings

<<

Summary

- What is infrastructure as code
- DevOps foundation
- Script Analyzer & Help Checker
- Static Analysis of configuration file
- Unit Test of IaC deployment
- Unit Test of pipeline
- Integration / System Test with Pester



questions?

Use the conference app to vote for this session:

<https://my.eventraft.com/psconfeu>



@MarkWarneke

Maturity Model

Practice	Build management and continuous integration	Environments and deployment	Release management and compliance	Testing	Data management	Configuration management
Level 3 - Optimizing: Focus on process improvement	Teams regularly meet to discuss integration problems and resolve them with automation, faster feedback, and better visibility.	All environments managed effectively. Provisioning fully automated. Virtualization used if applicable.	Operations and delivery teams regularly collaborate to manage risks and reduce cycle time.	Production rollbacks rare. Defects found and fixed immediately.	Release to release feedback loop of database performance and deployment process.	Regular validation that CM policy supports effective collaboration, rapid development, and auditable change management processes.
Level 2 - Quantitatively managed: Process measured and controlled	Build metrics gathered, made visible, and acted on. Builds are not left broken.	Orchestrated deployments managed. Release and rollback processes tested.	Environment and application health monitored and proactively managed. Cycle time monitored.	Quality metrics and trends tracked. Non functional requirements defined and measured.	Database upgrades and rollbacks tested with every deployment. Database performance monitored and optimized.	Developers check in to mainline at least once a day. Branching only used for releases.
Level 1 - Consistent: Automated processes applied across whole application lifecycle	Automated build and test cycle every time a change is committed. Dependencies managed. Re-use of scripts and tools.	Fully automated, self-service push-button process for deploying software. Same process to deploy to every environment.	Change management and approvals processes defined and enforced. Regulatory and compliance conditions met.	Automated unit and acceptance tests, the latter written with testers. Testing part of development process.	Database changes performed automatically as part of deployment process.	Libraries and dependencies managed. Version control usage policies determined by change management process.
Level 0 – Repeatable: Process documented and partly automated	Regular automated build and testing. Any build can be re-created from source control using automated process.	Automated deployment to some environments. Creation of new environments is cheap. All configuration externalized / versioned	Painful and infrequent, but reliable, releases. Limited traceability from requirements to release.	Automated tests written as part of story development.	Changes to databases done with automated scripts versioned with application.	Version control in use for everything required to recreate software: source code, configuration, build and deploy scripts, data migrations.
Level -1 – Regressive: processes unrepeatable, poorly controlled, and reactive	Manual processes for building software. No management of artifacts and reports.	Manual process for deploying software. Environment-specific binaries. Environments provisioned manually.	Infrequent and unreliable releases.	Manual testing after development.	Data migrations unversioned and performed manually.	Version control either not used, or check-ins happen infrequently.

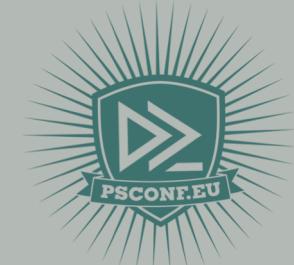


Testing, Testing, 1...2...3: Using Pester for Infrastructure Validation by Brandon Olin

- <https://www.youtube.com/watch?v=6bPByJX5euc>

Infrastructure validation using Pester,
Irwin Strachan

- https://www.youtube.com/watch?v=Qfi_H7IZyHg

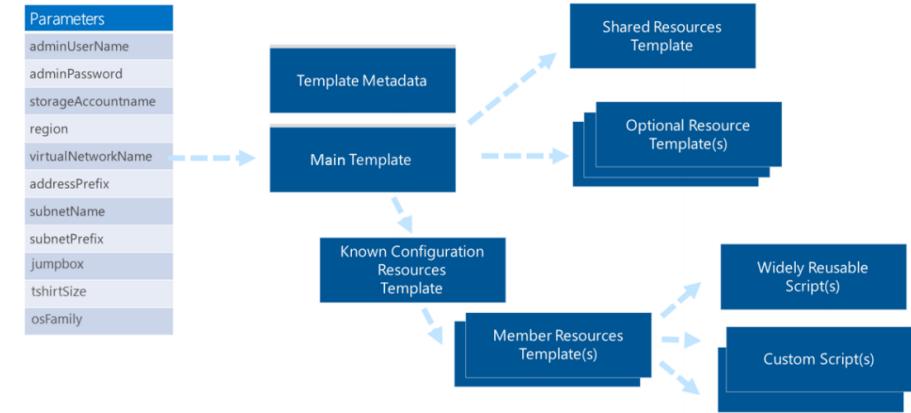


World Class ARM Templates - Considerations and Proven Practices

Marc Mercuri, Principal Program Manager,
Ulrich Homann, Distinguished Architect
George Moore, Principal Program Manager Lead

Reviewers – Silvano Coriani, Rafael Godinho, Paige Lu, Rama Ramani, Jeremiah Talker, Arsen
Vladimirskiy, Tim Wieman , Geert Baeke

June 30, 2015



<http://download.microsoft.com/download/8/E/1/8E1DBEFA-CECE-4DC9-A813-93520A5D7CFE/World%20Class%20ARM%20Templates%20-%20Considerations%20and%20Proven%20Practices.pdf>



Best Practices Guide

start-templates

Watch ▾ 564

Unstar 4,629

Fork

Pull requests 23

Projects 0

Wiki

Insights

quickstart-templates / 1-CONTRIBUTION-GUIDE / best-practices.md

Find file Cop

ces doc

52ab5ef on Oct 7



KB

Raw Blame History

Source Manager Templates - Best Practices Guide

This document contains the best practices for reviewing and troubleshooting Azure Resource Manager (ARM) Templates, and provides guidelines for creating ARM templates for the Azure Marketplaces. This document is intended to help you design effective templates or applications for getting applications certified for the Azure Marketplace and Azure QuickStart templates.

This document lists currently available Azure Resource Manager templates contributed by the community. A full list of templates can be found at <https://azure.microsoft.com/en-us/documentation/templates/>.

Before you contribute your template to the Azure Marketplace, you must read and follow these best practices as well as the guidelines listed in the following sections:

Contributing to the repository:

Code review and testing:

Deployment and troubleshooting:

Marketplace certification and compliance:

Feedback and support:

- <https://github.com/Azure/azure-quickstart-templates>
- <https://github.com/Azure/azure-quickstart-templates/blob/master/1-CONTRIBUTION-GUIDE/best-practices.md>



Best Practices Guide

Best Practices For Using Azure Resource Manager Templates

Rate this article 



MVP Award Program May 1, 2018

 Share 19

 0

 0

 1

Editor's note: The following post was written by Visual Studio and Development Technologies MVP Peter Groenewegen Microsoft Azure MVP Pascal Naber as part of our Technical Tuesday series. Mia Chang of the Technical Committee served as the Technical Reviewer of this piece.

This article focuses on best practices regarding the automated deployment of resources to Azure. We have implemented Continuous Deployment (CD) pipelines including the provisioning of Azure resources for many customers, and we would like to share our experience so you can benefit from it. These practices will help you create more reliable, testable, reusable, and maintainable templates.

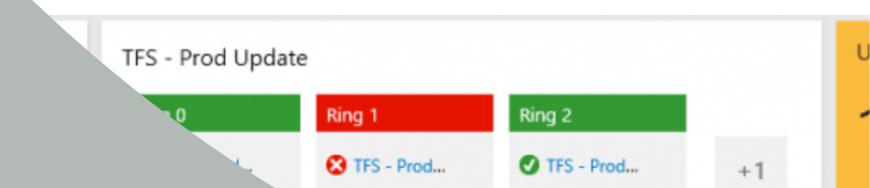
Automate deployments to Azure

Resource Manager templates (ARM templates) are the preferred way of automating the deployment of resources to Azure Resource Manager (AzureRM). ARM templates are JavaScript Object Notation (JSON) files. Resources that you want to deploy are declaratively described within JSON. An ARM template is repeatable, which means it can be executed as many times as you wish, and the result will be the same every time. It takes care of the execution and identifies the changes that need to be executed.

When provisioning infrastructure, we apply the same best practices as with deploying applications. This is also known as [Infrastructure as Code](#). Applying CD enables you to develop your infrastructure in a repeatable and reliable manner. You can reuse your ARM templates over multiple teams by applying these practices. This allows you to use a dashboard to monitor the quality of the infrastructure provisioning.

To provision resources in a CD pipeline, our preferred method uses Visual Studio Team Services (VSTS). The easiest way is to add a VSTS task: "Azure Resource Group Deployment".

It is very useful to monitor all your builds and releases, and this will give you a quick overview of the quality of your templates. It is very useful to show your team and other stakeholders.



- <https://blogs.msdn.microsoft.com/mvpawardprogram/2018/05/01/azure-resource-manager/>



Videos

- <https://app.pluralsight.com/library/courses/microsoft-azure-resource-manager-mastering/table-of-contents>
- <https://channel9.msdn.com/Events/Build/2015/3-618>
- <https://channel9.msdn.com/Events/Build/2015/2-659>
- <https://channel9.msdn.com/Events/Ignite/2015/BRK4453>



A screenshot of a video player interface. The title "Mastering Microsoft Azure Resource Manager" is visible at the top. Below the title, there is descriptive text about the course content. A large play button with the text "Play course overview" is centered in a white box. In the background, a person's hands are shown typing on a keyboard.

Blogs

Azure Security Audits with Pester:

<https://samcogan.com/azure-security-audits-with-pester/>

Infrastructure as Code Maturity Model:

<https://medium.com/@GaryStafford/infrastructure-as-code-maturity-model-9206b21d5dad>

An introduction to infrastructure testing with PowerShell Pester

<https://4sysops.com/archives/an-introduction-to-infrastructure-testing-with-powershell-pester/>



@MarkWarneke

A black and white portrait of a young man with short, wavy hair, smiling at the camera. He is wearing a light-colored button-down shirt. The background is a plain, light color.

about_Speaker

Mark Warneke

Consultant



Microsoft



<http://aka.ms/mark/>



github.com/MarkWarneke



[@MarkWarneke](https://twitter.com/MarkWarneke)

