

Automatisiertes Testen mit PowerShell und Pester

30.11.2022



Vorstellung

Melanie Eibl

mail@melanie-eibl.de



@melanieeibl

Blog: <https://melanie-eibl.de>

Freiberufliche Beraterin
und Softwareentwicklerin

<https://dotnet-cologne.de/>
<https://www.identitysummit.cloud/>



Agenda

Einrichten einer Entwicklungsumgebung für PowerShell und Pester

Arbeitsweise von Pester

- Schlüsselworte
- Discovery und Run
- Arrange / Act / Assert

Beispiele für Aufbau und Ausführung von Tests

Motivation

In agilen Softwareprojekten ist automatisiertes Testen unerlässlich, um kontinuierliche Produktivität gewährleisten zu können

Test Level:

- UnitTests
- Komponententests
- Integrationstests
- Systemtests
- User Acceptance Tests
- Performancetests

PowerShell (Installation)



Zu unterscheiden: Windows PowerShell powershell.exe und PowerShell Core (seit Version 7 nur noch PowerShell) pwsh.exe

Basiert auf CLR – alle Ein- und Ausgaben sind .NET-Objekte

Plattformunabhängig (Windows, Linux, iOS, Docker usw.)

Installation über <https://github.com/powershell/powershell> (oder über Microsoft Doku)

Abfrage der aktuell installierten Version: `$PSVersionTable`

Für Entwicklungsumgebung:

- Visual Studio Code (VS Code) – Installation über <https://code.visualstudio.com/>
- PowerShell Extension <https://marketplace.visualstudio.com/VSCode> oder über Extensions in VS Code

ExecutionPolicy => Abfrage mit `Get-ExecutionPolicy`

Auf Windows Server ist RemoteSigned voreingestellt

Vergleichbare Einstellung auf Windows Desktop mit
`Set-ExecutionPolicy -ExecutionPolicy RemoteSigned`



Pester (Installation)

(Unit-)Test-Runner für PowerShell

GitHub Open Source Projekt <https://github.com/pester/pester>

PowerShell selber ist mit Pester getestet => weiterhin posh-git

Auf Windows Systemen bereits in Version 3.40 vorinstalliert => nicht empfohlen

Abfrage der aktuellen Installation: `Get-InstalledModule Pester`

PowerShell Modul – PowerShell Gallery <https://www.powershellgallery.com/>

`Install-Module Pester`



Pester Anwendungsgebiete

Eigentlich Test-Framework für PowerShell-Entwicklungen

Auch verwendbar zum Testen von:

- APIs
- Infrastruktur
- Legacy Systeme
- Wenn UnitTests nicht unterstützt werden => Altanwendung erst mit Tests abdecken und dann modernisieren
- UIs
- Audits
- u.v.a.

Pester Schlüsselworte (Funktionen)

Context / Describe	Logische Gruppierung
It	Test
Should	Verifikation
BeforeDiscovery	Wird vor der Discovery-Phase ausgeführt
BeforeAll / AfterAll	Wird nach Discovery-Phase vor / nach allen Tests ausgeführt
BeforeEach / AfterEach	Wird vor / nach jedem It ausgeführt

Demo Hello World

'Hello World!'

```
Import-Module Pester
```

```
BeforeAll {  
    # Arrange  
    . $PSScriptRoot\HelloWorld.ps1  
}  
  
Context "Hello World" {  
    It 'Output Should Be Hello World!' {  
        # Act  
        $ret = Write-HelloWorld  
  
        # Assert  
        $ret | Should -Be 'Hello World!'  
    }  
}
```

Invoke, Discovery und Run

Pester-Test-Dateien enden auf "*.Tests.ps1"

Invoke-Pester führt alle Pester-Tests aus

Der Test-Runner führt zunächst ein Discovery durch => Schlüsselworte werden gesucht und in eine interne Datenstruktur überführt

In der Run-Phase wird die Datenstruktur ausgeführt

Datengetriebene Tests

Einem It kann mit dem Parameter -ForEach ein Array übergeben werden, um Tests parametrisiert ausführen zu lassen

Datengetriebene Tests

Einem It kann mit dem Parameter -ForEach ein Array übergeben werden, um Tests parametrisiert ausführen zu lassen

```
Context 'When calling add' {  
  It '<a> plus <b> should equal <c>' -ForEach @(  
    @ { a = 4; b = 5; c = 9 }  
    @ { a = 9; b = 9; c = 18 }  
    @ { a = 7; b = 5; c = 12 }  
    @ { a = 40; b = 50; c = 90 }  
  ) {  
    add -a $a -b $b | Should -Be $c  
  }  
  
  It "-2 plus 10 should equal 8" {  
    add -a -2 -b 10 | Should -Be 8  
  }  
}
```

Mocking

```
BeforeAll {  
    function Get-Stuff {  
        return Get-Random -Minimum 0 -Maximum 10  
    }  
}
```

```
Describe "Mocking Get-Random" {  
    It "Test Get-Stuff" {  
        Mock Get-Random { return 7 }  
        Get-Stuff | Should -Be 7  
    }  
}
```

Beispiele

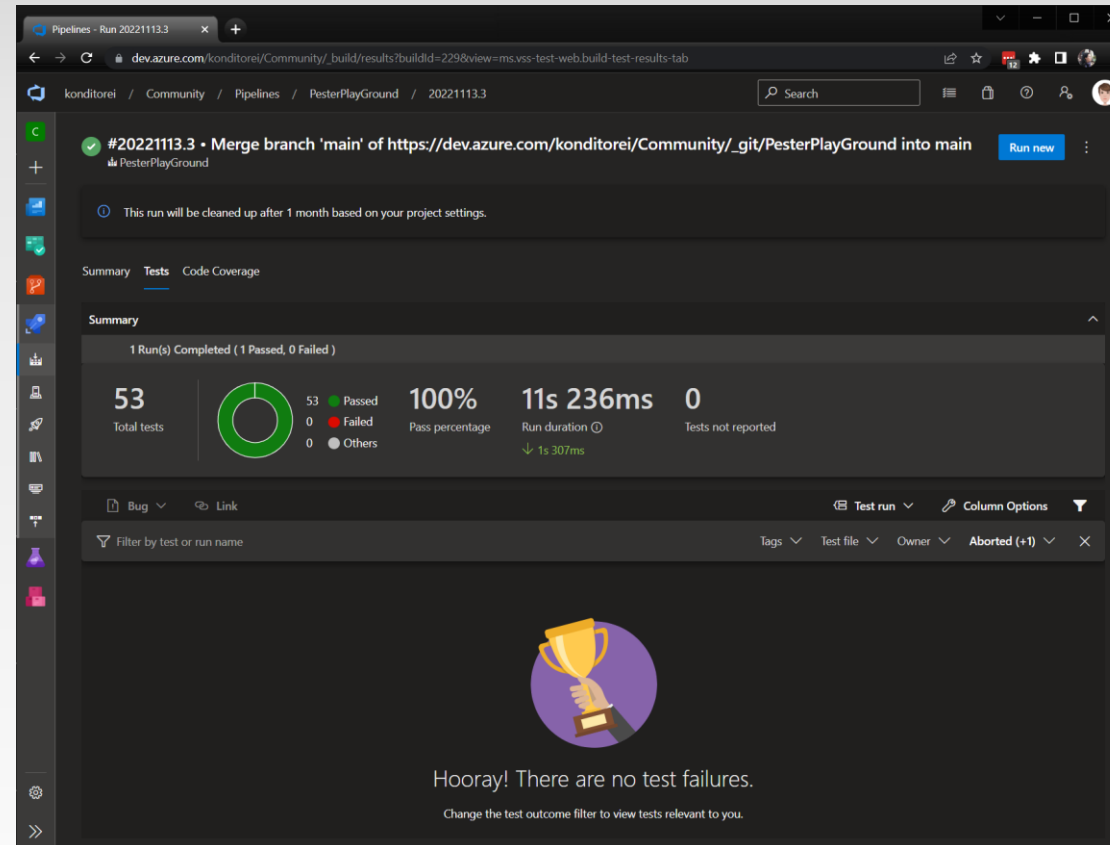
UI-Tests mit Selenium am Beispiel Login-Prozess

Audit am Beispiel OIDC Discovery Document

Serialisieren von .NET Objekten

Commandlets (Script/Binary)

Azure Pipeline



Zusammenfassung

Testen mit Pester nicht nur auf PowerShell Entwicklungen beschränkt

Im Black-Box-Verfahren für alles geeignet, wo Ergebnisse über PowerShell ermittelt werden können

Beliebige Bereitstellung von Infrastruktur und anderen Umgebungen

Durch Verwendung von Context und Describe Titel verbesserte Lesbarkeit für Anwender
=> Transparenz und Nachvollziehbarkeit => BDD

Danke!

[HTTPS://GITHUB.COM/MELANIEEIBL/PESTERPLAYGROUND](https://github.com/melanieeibl/pesterplayground)