HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

COMPUTER SCIENCE

Developer Manual

System Readiness Inspector

Authors:

Claudio Mattes
claudio.mattes@hsr.ch

Lukas Kellenberger
lukas.kellenberger@hsr.ch

Departement Computer Sciences
HSR University of Applied Sciences Rapperswil
CH-8640 Rapperswil, Switzerland

December 13, 2018

Contents

General Information

1.1 System Overview

The "System Readiness Inspector" is a Windows PowerShell tool that helps you to check the readiness of a system to detect advanced persistent threats and lateral movement.After the SRI ran successfully it generates a PDF-Document showing wrong or missung configurations. The SRI was developed during a student research project by the two bachelor of science in computer science students, Claudio Mattes and Lukas Kellenberger.

The SRI has four different modes: Online, Offline, GroupPolicy, AllGroupPolicies. The online mode is limited to the current system and thus determines readiness. The offline mode is used to be able to make a statement about any system by means of exports. The GroupPolicy mode is limited to a specific Group Policy, which is checked for its audit settings. In the AllGroupPolicies mode, all group policies of the current domain are examined.

1.2 Organization of the Manual

The developer manual contains the following parts:

- **General Information:**
  The General Information section explains the tool and the purpose for which it is intended.
- **System Requirements:**
  The System Requirements sections describes the requirements for a developer environment to getting started with coding.
- **Continuous Integration:**
  This sections provides hints for a accurate continuous integration environment built with Microsoft Azure DevOps.
- **Test Framework:**
  Within this section some key findings with the test framework Pester are provided.

System Requirements

This section is about how to get started with coding for the "System Readiness Inspector" (SRI). The focus in this section is about the used software and extensions to provide an environment to develop. But basically, there is no restriction how to handle your environment to get started.

2.1 Operating System

To develop the SRI the operating system "Microsoft Windows 10 Professional - Version 1803" was used.

2.2 Windows PowerShell 5.0

The used language in this project is Windows PowerShell. Be sure that you have installed the "Windows Management Framework 5.1". Check your version with the following command:

```
1 $PSVersionTable.PSVersion
```

Windows Management Framework 5.1:
https://www.microsoft.com/en-us/download/details.aspx?id=54616

2.3 Integrated Development Environment (IDE)

During this study thesis "Microsoft Visual Studio Code - Version 1.29.1" served as the IDE to develop in Windows PowerShell. Microsofts integrated IDE for Windows PowerShell "Integrated Script Environment (ISE)" was refused to use because the Microsoft Visual Studio Code IDE provides a very large set on extensions useful for any kind of developing. In addition, the integrated "Source Control" tab makes it extremely easy to maintain strict version control.

Microsoft Visual Studio Code:
https://code.visualstudio.com/

2.4 Microsoft Visual Studio Code Extensions

To develop efficiently in Windows PowerShell with "Microsoft Visual Studio Code" the following extensions are used during the development:

- PowerShell (extension identifier: `ms-vscode.powershell`)
- Code Spell Checker (extension identifier: `streetsidesoftware.code-spell-checker`)

Continuous Integration

To provide a continuous integration environment for this project Microsoft Azure DevOps was used.

3.1 Microsoft Azure DevOps

To use the ”Microsoft Azure DevOps” a Microsoft account is required:

Microsoft Registration:
https://account.microsoft.com/account?lang=en-us

After the registration sign in ”Microsoft Azure DevOps”:

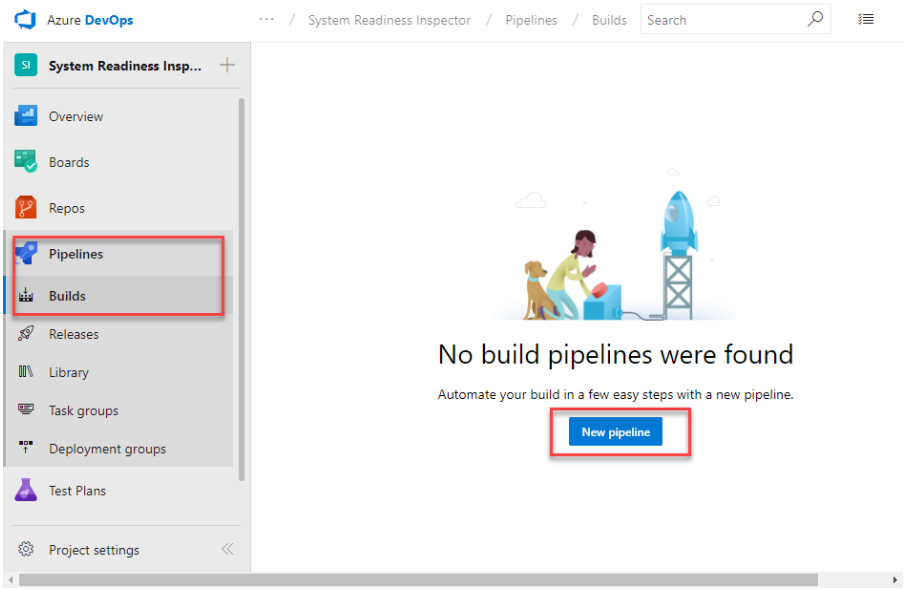
Microsoft Azure DevOps Sign-In:
https://dev.azure.com/

Use the following link to get started with ”Microsoft Azure DevOps”:

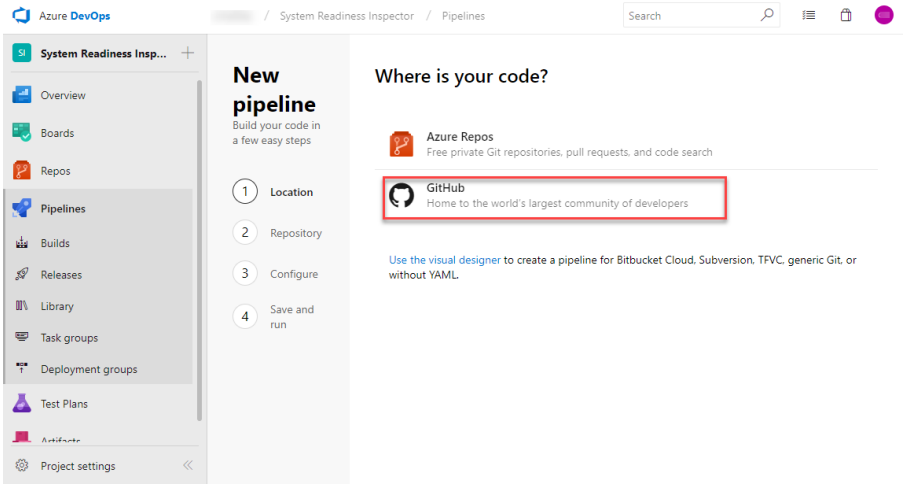
Getting Started with Microsoft Azure DevOps:
https://docs.microsoft.com/en-us/azure/devops/user-guide/sign-up-invite-teammates?view=vsts

3.1.1 Configuration

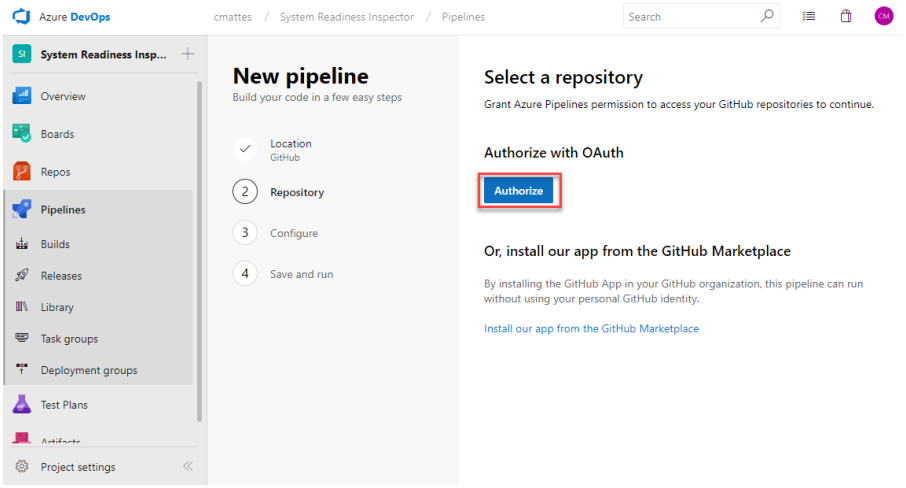
There are some configurations to make for a minimal continuous integration with ”Microsoft Azure DevOps”. On the left hand side click on ”Pipelines - Builds” and click on ”New Pipeline”:



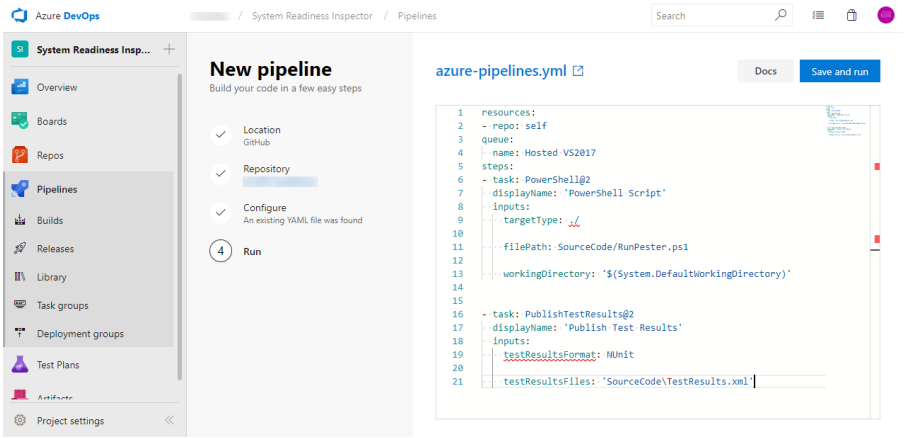
Select your location of the source code. In this project GitHub was used:



After that, select your repository to use in your source code location:



Modify the YAML in the next step as provided after the printscreen and your continuous integration is ready to go:



```
resources:
- repo: self
queue:
  name: Hosted VS2017
steps:
- task: PowerShell@2
  displayName: 'PowerShell Script'
  inputs:
    targetType: ./

    filePath: SourceCode/RunPester.ps1

    workingDirectory: '$(System.DefaultWorkingDirectory)'

- task: PublishTestResults@2
  displayName: 'Publish Test Results'
  inputs:
    testResultsFormat: NUnit

    testResultsFiles: 'SourceCode\TestResults.xml'
```

This YAML-File is adjusted for the use of the Pester test framework within the project. For more information use the following link, which provides additional information if needed: https://www.powershellmagazine.com/2018/09/20/converting-a-powershell-project-to-use-azure-devops-pipelines/

Test Framework

4.1 Pester

Pester is a framework to provide a test environment for PowerShell projects. More specific, the framework supports tests for any written function in PowerShell. To provide tests on the continuous integration server, it is recommended to clone the Pester repository from GitHub and save it into your root path of your source code. During the project we used to have the following directory structure:

```
Source Code Path:.
   |-- RunPester.ps1
   |-- TestResults.xml
   |
   |---Pester
   |
   |---SRI
      |-- sri.ps1
      |
      |---Config
      |     audit_by_category.xml
      |     event_log_list.xml
      |     targetlist_auditpolicies.xml
      |
      |---Modules
         |  GetAndAnalyseAuditPolicies.psm1
         |  GetAndAnalyseAuditPolicies.Tests.ps1
         |  GetAndCompareLogs.psm1
         |  GetAndCompareLogs.Tests.ps1
         |  itextsharp.dll
         |  Visualize.psm1
         |
         |---TestFiles
```

In addition to place the Pester repository in the root path, you have to provide a `RunPester.ps1`-File to invoke the tests on the continuous integration server. The `RunPester.ps1` should contain the following code snippet:

```
Import-Module "$PSScriptRoot\Pester\Pester.psm1"
Invoke-Pester -Script "$PSScriptRoot\SRI\Modules" -OutputFormat NUnitXml -OutputFile
    "$PSScriptRoot\TestResults.xml" -PassThru -ExcludeTag Incomplete
```

This code snippet defines the starting point for Pester. Moreover, with the parameter combination `-OutputFormat NUnitXml -OutputFile <PATH> -PassThru -ExcludeTag Incomplete` Pester generates a `TestResults.xml` which is supported by "Microsoft Azure DevOps". This file is then use by the continuous integration server to represent the test results in a nice view for each build.