Powershell Monitoring Framework

Lightweight Configuration Management Database

Author: Hugh Scott

Version: 1.0

Date: March 11, 2015

# Version Control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Changes** | **Comments** |
| 1.0 | 2015/03/11 | Hugh Scott | Original |  |

Table of Contents

[Version Control 2](#_Toc416359548)

[What’s New 4](#_Toc416359549)

[Background 5](#_Toc416359550)

[Setup Requirement and Permissions 5](#_Toc416359551)

[Server Installation Requirements 5](#_Toc416359552)

[Domain and Server Permissions 5](#_Toc416359553)

[Active Directory Domain Permissions 5](#_Toc416359554)

[WMI and PowerShell Permissions 6](#_Toc416359555)

[SQL Server Permissions (Central Repository) 6](#_Toc416359556)

[SQL Server Permissions (Target Servers) 6](#_Toc416359557)

[Modules 6](#_Toc416359558)

[Monitor.ps1 6](#_Toc416359559)

[Configuration File 6](#_Toc416359560)

[Parameters 6](#_Toc416359561)

[Process overview for Monitor.ps1 8](#_Toc416359562)

[Sub Modules 8](#_Toc416359563)

[MonitorDomain.ps1 11](#_Toc416359564)

[Parameters 12](#_Toc416359565)

[Active Directory Objects 12](#_Toc416359566)

[MonitorAdmin.ps1 13](#_Toc416359567)

[Scheduling 13](#_Toc416359568)

[General Notes on the Data Model 14](#_Toc416359569)

[Primary Keys and Unique Keys 14](#_Toc416359570)

[Active, dbAddDate, dbLastUpdate attributes 15](#_Toc416359571)

[Object Creation Time Stamp versus dbAddDate 15](#_Toc416359572)

[Historical Data 16](#_Toc416359573)

[Use of Data for Auditing 16](#_Toc416359574)

[Database Diagram: Computer 17](#_Toc416359575)

[Database Diagram: Database Instance 18](#_Toc416359576)

# What’s New

March 11, 2015

Version 1.0

First release.

# Background

The Powershell Monitoring Framework is a collection of Powershell scripts used to monitor a Microsoft Windows environment, with a focus on Microsoft SQL Server assets. The tool is agentless and uses standard WMI and PowerShell cmdlets for most monitoring activities (there are still a few checks based on registry entries).

The primary purpose of the tool is to provide answers to the following types of questions:

1. How many computers do I have in my Active Directory environment, right now?
2. How many computers did I have last year?
3. How much storage do I have allocated to all of my Windows computers right now?
4. How much storage did I have allocated last year?
5. How many databases do I have, right now?
6. How many databases did I have last year?
7. How many users do I have in Active Directory and what groups are they members of?
8. What Windows applications do I have installed on these computers?
9. What services are running on these computers (or, which computers are running this Service)?
10. What SQL Agent jobs are scheduled in the environment (and which are failing)?

In summary, this tool is designed to measure and monitor your environment at a high level (and often in the aggregate). It’s an attempt to describe your Windows environment as a whole and not just as individual piece parts. It is also an attempt to measure your environment over long periods of time (even years).

***This is NOT a detailed performance monitoring tool.*** It is intended to be lightweight enough to store ***years*** *of data about your environment without consuming entire racks of storage.*

# Setup Requirements and Permissions

## Server Installation Requirements

Operating System: Windows 2012 or Windows 2012 R2

Powershell Version: 3.0/4.0

Software Requirements:

Active Directory Powershell Module

Failover Cluster Powershell Cmdlets

SQL Server Shared Management Objects

## Domain and Server Permissions

### Active Directory Domain Permissions

No special permissions are required for Active Directory. The Service account used to schedule the check process just needs to be a member of the domain (or have an established trust identity if the domain being checked is in a separate forest).

### WMI and PowerShell Permissions

Currently, the user account performing checks on remote computers needs to be a member of the local administrators group in order for all checks to be performed successfully.

### SQL Server Permissions (Central Repository)

There are predefined roles for each schema used for the checks to be successfully performed. The service account must be a member of the pre-defined roles in order to successfully update the database.

### SQL Server Permissions (Target Servers)

Although not confirmed, it should only be necessary to grant the following rights/privileges for SQL Server checks on remote computers to be performed successfully:

VIEW SERVER STATE

VIEW ANY DEFINITION

VIEW ANY DATABASE

In msdb:

SQLAgentReaderRole

In each database:

Create the user from the Windows login (needs only Public role).

### Connectivity and Port Requirements

Currently, port and network requirements are assessed to be:

ICMP Enabled (Ping)

TCP/UDP Port 135 Microsoft End Point Mapper / RPC Locator

## TCP/UDP Port 137: Netbios name service

## TCP/UDP Port 138: Netbios Datagram service

## TCP/UDP Port 139: Netbios Session service

TCP/UDP Port 445: Microsoft Active Directory, Windows Shares

# Modules

## Monitor.ps1

### Configuration File

Monitor.ps1 (as well as other components) use a single configuration file (app.monitor.config) which contains settings and values in XML format. These include:

*ThrottleLimit:* The number of concurrent checks that may be performed.

AgentName: The default agent name used when sourcing the list of computers to check from the central repository. The default may be overridden by specifying the –AgentName parameter in the monitor.ps1 command line.

ConnectionString: The connection string for the central repository.

### Parameters

#### Checks

An array of strings used to define which checks will be performed during execution of the script. This available values for this parameter are listed; invalid entries will be rejected and cause the script to fail. Current values (and their corresponding checks) are listed here:

| **Abbr** | **Check performed** | **Comment** |
| --- | --- | --- |
| Comp | Check computer system; all |  |
| Os | Check operating system; all |  |
| Disk | Check logical volume, physical disk and partitions; phys only |  |
| DiskSpc | Check logical volume space utilization; phys only |  |
| Clus | Check cluster configuration; all |  |
| CompShr | Check for computer share information; all |  |
| Hotifx | Check for Installed Windows Updates |  |
| InstApp | Check installed applications; phys only |  |
| Net | Check network configuration; phys only |  |
| NLB | Check network load balance configuration; phys only |  |
| Svcs | Check services; phys only |  |
| DiscSQL | Discover installed SQL instances; all |  |
| SQLEng | Check SQL Database Engine; only for discovered instances |  |
| SQLSize | Check SQL Database Size; only for discovered instances |  |
| Olap | Check SQL OLAP Engine; only for discovered instances |  |
| RptSvr | Check SQL Reporting Services; only for discovered instances |  |
| SQLJob | Check SQL Agent Jobs; only for discovered instances |  |
| SQLPerm | Check SQL Permissions; only for discovered instances |  |
| SysEvtLog | Check Windows System Event Log for critical events (WARNING!!!!! This can consume a large amount of space and time); phys only |  |
| AppEvtLog | Check Windows Application Event Log for critical events (WARNING!!!!! This can consume a large amount of space and time); phys only |  |
| WebURL | Test a URL; records response code and response time. |  |

#### Maintenance

An array of strings specifying maintenance to be performed on the database. Current valid values are:

|  |  |  |
| --- | --- | --- |
| **Abbr** | **Maintenance performed** | **Comment** |
| WriteHist | Writes daily summary data to historical tables from Raw tables |  |
| PurgeLog | Deletes entries from the Process Log |  |
| PurgeEvt | Deletes entries from the [cm].[Event] table |  |
| PurgeHist | Deletes historical data from daily summary and raw tables |  |

The retention values for each of these maintenance processes is stored as a varchar value in [dbo].[Config].

#### FileName

The user may specify a file name from which to retrieve computers for checks and validation. This should be a text file and each object to be checked should be entered on a separate line as a fully qualified domain name. For example:

<computer1>.<domain>.<net>

<computer2>.<domain>.<net>

<computer3>.<domain>.<net>

#### AgentName

When using the Central Repository to source the list of computers for checks, this parameter can be used to filter that list. This may be useful if you are monitoring multiple domains (with different credentials) or because the numbers of computers to monitor exceeds the workload capacity for one computer to process all the checks.

#### WhatIf

WhatIf is a switch parameter; if present (-whatif), then the process will simulate the actions indicated without actually performing any work. The console will output the simulated results. This can be useful to verify that the command syntax is correct, or to verify that the list of computers to be checked matches what the administrator is expecting to see (for example, when using multiple agents).

### Process overview for Monitor.ps1

At a high level, the following tasks are performed:

1. A connection is made to the central repository
2. Threading values are configured for parallelization
3. The list of computers to be checked is compiled; if no checks are specified, then this step is skipped.
   1. Each computer is tested for connectivity (ping, port 135)
   2. Each check is performed by spawning a new process (up to the throttle limit specified)
   3. The results are returned and errors/warnings tabulated
4. The list of maintenance tasks is enumerated; if no maintenance tasks are specified, then this step is skipped.

For a more detailed review of the tasks performed during each check, please see the notes below.

### Sub Modules

Each of the following modules is called from the primary module ***Monitor.ps1.*** These modules are NOT intended to be run by themselves.

#### CheckComputer.ps1

This module accesses the CIM\_ComputerSystem class and Win32\_ComputerSystemProduct to update table [cm].[Computer]. This is the base table (or “hardware” layer) of the setup.

Virtual network resources (for example SQL Server Virtual Network Names and MSCS Virtual Network names that are used for clustered resources are identified by a bit flag IsClusteredResource. These are added to the table for referential integrity. They are not physical entities.

Virtual machines are identified with a flag IsVirtual. The logic is embedded in the code. If the manufacturer is VMWare or Microsoft (Hyper-V), then the flag is set to true.

Some thought has been given to moving these entities to the [cm].[OperatingSystem] table and making that table the primary entity for everything except physical resources (disk drives, network adapaters, etc). That may be accomplished in a future release.

#### CheckOperatingSystem.ps1

The module accesses CIM\_OperatingSystem class to update the table [cm].[OperatingSystem].

Clustered resources (virtual network names) are also tracked in this entity. For this reason, it would probably be best to leave clustered entities out of [cm].[Computer] (which is meant to represent physical entities and virtual computer resources). That may be accomplished in a future release.

As part of the OperatingSystem check, the members of the Local Administrators group are enumerated and written to the table [cm].[ComputerGroupMember]. Active directory user accounts/groups may be cross referenced to the [ad].[User] or [ad].[Group] tables.

#### CheckCluster.ps1

This module attempts to discover and describe computers that are members of a Microsoft Failover Cluster. The module attempts to enumerate cluster components using Powershell cmdlets. For this reason, it is necessary to install the Microsoft Failover Cluster components (in Windows 7, these are found in the Add Windows Components after installing Remote Server Administration Tools).

The module attempts to enumerate Cluster Nodes, Cluster Resources, Cluster Groups and the Failover Cluster name. Any resource that is identified as a Network Name is added to the [cm].[Computer] table with the value IsClusterResource set to true.

#### CheckDisk.ps1

This module attempts to enumerate Logical Volumes, Physical Disks, Drive Partitions and the relationships found between Logical Volumes and Physical Disks. In order to describe mounted volumes, the component uses the Win32\_Volume (instead of CIM\_LogicalDisk). Only local drive types (DriveType = 3) are described.

In an effort to reduce complexity and duplication, an effort is made to identify the owner of a clustered disk before adding it to the database.

#### CheckNetwork.ps1

This module uses Win32\_NetworkAdapter and Win32\_NetworkAdapterConfiguration to identify adapters where TCP is enabled.

These entities are related to [cm].[ComputerSystem]. All IP addresses for a Network adapter are enumerated and listed in the IPAddress column.

#### CheckEvents.ps1

This module will attempt to enumerate entries in the specified Windows Event log (currently the System Event log and the Application Event log). The first time that the check is performed for a particular computer, it will retrieve all critical events logged for the past 14 days. After the initial load, the logic of the process is to retrieve only critical events that have occurred since the last critical entry was logged.

**WARNING!!!** This process can be resource intensive and can generate a lot of network traffic (as well as a lot of entries in the [cm].[Event] table.

**NOTE:** Only critical events are collected. Warnings and informational events are ignored.

**NOTE:** In an effort to limit traffic and load, virtual network resources are ignored when collecting events.

#### CheckInstalledApplications.ps1

This module attempts to enumerate entries from the Windows uninstall registry hive. An effort is made to create a “normalized” table of installed applications, along with a linked table showing which computers those applications are installed on.

**NOTE:** Some applications do not use the Windows uninstall registry. Applications such as Oracle have their own install/uninstall routines and therefore will not be discovered as part of this process.

**NOTE:** This process will ***not*** enumerate Windows Updates that have been installed. See the description for *CheckInstalledHotfix.ps1*.

#### CheckInstalledHotfix

This module uses the Powershell cmdlet Get-Hotfix to enumerate the installed hotfixes on a computer. Entries are written to [cm].[WindowsUpdateInstallation]. Similar to the module *CheckInstalledApplications.ps1*, this module maintains a “master” copy of hotfixes in [cm].[WindowsUpdate].

#### CheckServices.ps1

This module enumerates and describes entries from the Win32\_Service class. An attempt was made to use the CIM\_Service class, but it proved to be too slow for practical use.

#### CheckComputerShare.ps1

This module enumerates Shared resources (printers, file shares and administrative shares) for each computer object. The module uses Win32\_Share to list these resources.

#### CheckSQLInstance.ps1

This module attempts to discover installed components of SQL Server (the database engine, Analysis Services and Reporting Services). As yet, SSIS is not included in the discovery routing.

This process is heavily dependent on the Windows Registry to identify installed SQL components and to list out their properties. Due to complications involved in connecting to non-default instances when port 1434 is blocked (or when the SQL Browser is disabled), the process tries to identify non-default ports for the database engine using the entries found in **SuperSocketNetLib\Tcp\IPAll**. This is not always valid.

Alone among the various SQL modules, this module does not use alternate database credentials (thought it can use alternate credentials for OS checks). This module does not actually log in to the database instance at all.

For each of the discovered SQL Classes and instances, an effort is made to determine if the appropriate service for that resource is running.

For Reporting Services, some manual work will need to occur to update the URL for the more detailed Reporting Services process to work.

#### CheckSQLEngine.ps1

This module lists the databases, data files, database properties, server properties and linked servers for the discovered instances on a particular host.

#### CheckSQLJobs.ps1

This module checks the SQL Agent jobs for the discovered instances on a particular host. It does NOT return the detailed list of steps, nor a detailed job history for each job. It is intended to be as fast and lightweight as possible in order to facilitate a quick check (which can therefore be run multiple times).

#### CheckSQLPermissions.ps1

This module enumerates all the logins on a SQL database instance as well as server permissions, database users and database permissions. Object, Server and Database Permissions are all stored in a single table called [cm].[DatabasePermission]. This was an effort to simplify the data solution. A surrogate column was added to define which type of permission is being described.

#### CheckSQLReportingServer.ps1

This module enumerates reports, data sources, data sets, subscriptions and subscription parameters that are related to an instance of SQL Server Reporting Services. Note that the URL for the report server instance must be manually configured by an administrator.

#### CheckSQLAnalysisServer.ps1

This module enumerates the databases and cubes and the server properties for each discovered instance of SQL Analysis Services.

#### CheckSQLDatabaseSize.ps1

This module updates the attributes DataFileSize, DataFileSpaceUsed, LogFileSize and LogFileSpaceUsed for each database in an instance. It also writes an entry into [pm].[DabaseSizeRaw]. The entries in [pm].[DatabaseSizeRaw] are summarized daily into [pm].[DatabaseSizeDaily] as part of the WriteHistory task which is part of the maintenance process.

#### CheckDiskSpace.ps1

This module updates the attribute SpaceUsed in [cm].[LogicalVolume]. In addition, it writes an entry into [pm].[LogicalVolumeSizeDaily]. The entries in [pm].[LogicalVolumeSizeRaw] are summarized daily into [pm].[LogicalVolumeSizeDaily as part of the WriteHistory task which is part of the maintenance process.

#### CheckWebURL.ps1

This module tests a URL for response code and response time. The latest response time and response code are written to [cm].[WebApplicationURL], while a record of each check is written to [pm].[WebApplicationURLResponseRaw]. This check is useful for validating that web services are up and responding to url requests. It is also useful to “warm” .Net services (such as Reporting Services) whose application pools periodically shut down (thus leading to lengthy response times when accessed by end users).

#### CheckNLBCluster.ps1

This module checks for Windows Network Load Balanced clusters, discovers the underlying node(s) and updates the status of the underlying node(s). This module requires installation of Windows Network Load Balance client tools and uses NLB PowerShell cmdlets.

## MonitorDomain.ps1

MonitorDomain attempts to enumerate objects in Active Directory and load these into corresponding tables in the database. This can be useful for correlating users in Active Directory with users in SQL Server and discovering new computers which may have been added to the Domain (which can then be added to the list of computers to check). Each object checked is listed below. Note that not all attributes for every object are returned.

### Parameters

#### adDomain

The Active Directory domain to which the process connects. This is best supplied in the form: <domain>.<com>. Alternate domains can be connected to as long as the user is defined (ie, a trust relationship exists).

#### adObjectType

Currently this parameter accepts a string which must contain characters in the set {domain, forest, computer, user, group, groupmember, site, subnet}. Multiple objects may be checked in a single execution.

#### syncType

The synchronization process can be set to **Full** or **Incremental** (the default value is **Full**). A Full synchronization will retrieve all of the objects of the specified class in Active Directory. An Incremental synchronization will only retrieve those objects that have been changed since the last synchronization was performed.

If an Incremental update is specified, the process first checks the [ad].[SyncStatus] table to determine when the last incremental update was performed. It then uses the date/time of **start** of the last incremental update as the basis for the whenChanged filter when retrieving objects from AD.

If the process cannot find an entry corresponding to that object, then it will automatically escalate to a Full synchronization.

Active objects in the database are not updated to “Inactive” during Incremental updates. These are only updated after a Full update.

### Active Directory Objects

#### Computers

All AD computer objects are loaded into the database into table [ad].[Computer].

#### Users

All AD user objects are loaded into the database into [ad].[User].

#### Groups

All AD groups are loaded into the database into [ad].[Group].

#### GroupMembers

All AD groups are enumerated and them members of those groups are then listed and synchronized to the database.

**NOTE:** The default group for a user is not listed in the MemberOf property for that group. This is a known issue and will be addressed at a future date.

**WARNING:** The process for enumerating and updating user group membership relationships can be VERY time consuming in a large environment with many users and many group members.

#### Sites

All AD sites are loaded into the database into [ad].[Site].

#### Subnets

All AD subnets are loaded into the database into [ad].[Subnet].

## MonitorAdmin.ps1

# Scheduling

The key to being able to answer these questions effectively is to schedule the various scripts to run at appropriate and predictable intervals. The following tables should serve as a guideline for the frequency with which checks should be performed:

#### Active Directory Checks Schedule (Sample)

| **Abbr** | **Frequency** | **Comment** |
| --- | --- | --- |
| Users | Daily |  |
| Computers | Daily |  |
| Groups | Daily |  |
| Group Membership | Daily\* |  |
| Sites and Subnets | Weekly |  |
| Domain | Daily |  |
| Forest | Daily |  |

(\*) Group Membership can be a very lengthy process depending on the number of groups and the number of members in each group. This process can potentially take hours for very large environments.

#### WMI and PowerShell based Checks Schedule (Sample)

| **Abbr** | **Frequency** | **Comment** |
| --- | --- | --- |
| Comp | Daily |  |
| Os | Daily |  |
| Disk | Daily |  |
| DiskSpc | Hourly |  |
| Clus | Daily |  |
| CompShr | Daily |  |
| Hotfix | Weekly |  |
| InstApp | Weekly |  |
| Net | Daily |  |
| NLB | Hourly (\*) |  |
| Svcs | Hourly (\*) |  |
| DiscSQL | Daily |  |
| SQLEng | Daily |  |
| SQLSize | Hourly (\*) |  |
| Olap | Daily |  |
| RptSvr | Daily |  |
| SQLJob | Hourly (\*) |  |
| SQLPerm | Weekly |  |
| SysEvtLog | Daily (\*\*) |  |
| AppEvtLog | Daily (\*\*) |  |
| WebURL | Every 10-15 minutes (\*\*\*) |  |

(\*) These checks are designed to be lightweight and fast; they may be run more frequently, but in no case would I attempt to run them more than every ten minutes. Remember! This is not intended to be a real time monitoring system.

(\*\*) The Event Log checks can be very intensive depending on how many entries are in the individual log files. There are better tools for monitoring Windows Event logs, but this functionality can be useful in a pinch.

(\*\*\*) The WebURL check is intended to be very lightweight and fast. In order to serve as a “cache warming” service, it needs to be run frequently enough to prevent application pools from shutting down.

#### Maintenance Tasks

| **Abbr** | **Frequency** | **Comment** |
| --- | --- | --- |
| WriteHist | Daily | Schedule shortly after midnight |
| PurgeLog | Daily | Schedule off hours |
| PurgeEvt | Daily | Schedule off hours |
| PurgeHist | Daily | Schedule off hours |

# General Notes on the Data Model

## Primary Keys and Unique Keys

In most cases, entities have a primary key defined (usually [objectGUID] which is a uniqueidentifier). There should also be a unique index described for each entity to prevent duplicate values from being loaded into the table (for example, dnsHostName is the unique key for [cm].[Computer]. In some cases, the unique index is a compound key (such as [DatabaseInstanceGUID] and [DatabaseName] for [cm].[Database].

In many cases, if the object being described already had a clearly defined GUID value (for example, ID for SQL Agent Job), then I used that value as-is for the [objectGUID] value. This is probably a violation of best practices, but it made sense to me at the time.

For Active Directory objects (users, computers, groups, sites and subnets), the Unique index is on the Distinguished Name. This can potentially cause issues when, for example, a computer is deleted from AD and then re-added with the same Distinguished Name. To deal specifically with this issue, there is a table object called [ad].[DeletedObject]; this is a generic entity for Users, Computers, Groups, Sites and Subnets. In the Upsert stored procedures for each of these objects, there’s an initial check to see if a duplicate object exists with a different [objectGUID]. If a duplicate exists, meta data for the object is added to [ad].[DeletedObject], the object is removed from the entity table and the new record is inserted.

Objects which are deleted and re-added with a different Distinguished Name will not cause an issue; they will get added and the old record will persist (but it should be marked InActive).

Objects that have their Distinguished Names updates (such as when the object is moved from one Organizational Unit to another) will not cause an issue. The base GUID for the object will not change and the database will be updated to reflect the new DN.

## Active, dbAddDate, dbLastUpdate attributes

Each entity contains three common attributes: Active (Boolean), dbAddDate and dbLastUpdate. The state of these three attributes is related to whether an object currently exists, when it was first discovered and when it was last updated.

An entity with an Active attribute of 1 means that -- as of the value in dbLastUpdate – that entity was found to exist in the environment. An entity with an Active attribute of 0 means that that entity was inactivated (as of dbLastUpdate) and ***not found*** during the last check process.

As part of the check process, each routine will first “Inactivate” (set the Active flag to 0) for all objects (of the same object type) belonging to the parent of that object. For example, before enumerating the Logical Volumes on a computer, all existing Logical Volumes, ***for that computer*** are first “Inactivated”. As each volume is then enumerated, the value of the flag is set back to 1.

If a previously discovered object has been removed from a computer (for instance, following the above example, if a Logicial Volume was removed from a computer) then the Active flag (which had been set to 0) will not get set back to 1. Effectively, the application has “discovered” that this object has been delete

In most cases, this process is very fast and the user won’t notice if an object momentarily “goes away”. However, administrators must take care not to schedule reports during mass discovery operations (for example, you should take care not to schedule reports against the database during a period when the disk entities are being updated).

The value of inactivating (versus hard deleting a record) is that we retain the history of the environment from the time that an object was first discovered, even after that object has been retired from the environment. In addition, it can help identify certain Active Directory objects which persist, even though the object has been disabled.

## Object Creation Time Stamp versus dbAddDate

In some cases, entities have their own “Creation time stamp” (such as the Active Directory attribute “whenCreated”). Always keep in mind that the time an object is created does not necessarily correlate with the time that an object is discovered by the application. In cases where these is no creation time stamp attribute, **you must not infer too much information from dbAddDate**. This attribute simply describes when the entity was first added to the database. Only after you have established a set schedule for running the various processes (and have been running it for a while) can you begin to infer things like: “well, the application wasn’t here last week, and it’s here now: therefore it must have been added in the last week.”

Bear in mind that this process/tool was **never intended for real time monitoring.** In most cases, the major checks were only scheduled to be run daily or at most twice daily.

## Historical Data

Certain exceptions exist: the *CheckDiskSpace.ps1* and *CheckDatabaseSize.ps1* scripts do NOT inactivate objects. They simply check the sizes of known entities and update the Raw data tables so that a history of these statistics can be maintained.

## Use of Data for Auditing

It’s important to point out another key limitation of this application: **although this tool can be useful for auditing the current state of your environment, it is not a proper auditing solution.** The tool lacks the necessary transactional component(s) -- not to mention the necessary security controls -- to be used in an auditing role. Nevertheless, it can be used to quickly answer questions such as:

1. What permissions do my SQL Server users have and what permissions did they have last month?
2. Who was a member of this Active Directory group last month (and are they still a member today)?
3. How do the permissions in SQL production compare with those in my other environments?

# Database Diagram: Computer



# Database Diagram: Database Instance

