Contents

[Introduction to EPM Framework 2](#_Toc412055786)

[1. Set up Central Management Server 3](#_Toc412055787)

[2. Configure/Create Policies and Centralize on the Central Management Server 4](#_Toc412055788)

[3. Create Database and Database Objects To Store Policy Evaluation Results 6](#_Toc412055789)

[4. Configure PowerShell Script 7](#_Toc412055790)

[5. Test the PowerShell Script 8](#_Toc412055791)

[6. Create SQL Server Agent Job to Automate Execution of Policy Evaluation 9](#_Toc412055792)

[7. Deploy Reports to SQL Server Reporting Services 11](#_Toc412055793)

[Appendix A: Additional Considerations 15](#_Toc412055794)

[Security 15](#_Toc412055795)

[Archive 16](#_Toc412055796)

[Execution Strategies 17](#_Toc412055797)

[Appendix B: Database Objects 19](#_Toc412055798)

[Tables 19](#_Toc412055799)

[Views 19](#_Toc412055800)

[Stored Procedure 19](#_Toc412055801)

[Function 19](#_Toc412055802)

[Appendix C: Upgrade from previous version 20](#_Toc412055803)

[Database Objects 20](#_Toc412055804)

[PowerShell Script 20](#_Toc412055805)

[Reports 20](#_Toc412055806)

# Introduction to EPM Framework

The Enterprise Policy Management Framework (EPM) is a solution to extend SQL Server [Policy-Based Management](http://msdn.microsoft.com/en-us/library/bb510667.aspx) to all versions of SQL Server in an enterprise, including SQL Server 2000 and SQL Server 2005. The EPM Framework will report the state of specified SQL Server instances against policies that define intent, desired configuration, and deployment standards.

When the Enterprise Policy Management Framework (EPM) is implemented, policies will be evaluated against specified instances of SQL Server through PowerShell. This solution will require at least one instance of SQL Server 2008. The PowerShell script will run from this instance through a SQL Server Agent job or manually through the PowerShell interface. The PowerShell script will capture the policy evaluation output and insert the output to a SQL Server table. SQL Server Reporting Services (2008 or above) reports will deliver information from the centralized table.

This solution requires the following components to be configured in your environment. All SQL Server requirements listed below may be executed from and managed on the same instance:

* SQL Server (2012 or above) instance to store policies
* SQL Server (2012 or above) instance to act as the Central Management Server
* SQL Server (2012 or above) instance to execute the PowerShell script
* SQL Server management database to archive policy evaluation results
* SQL Server (2008 or above) Reporting Services to render and deliver policy history reports

Please refer to Microsoft documentation ([Features Supported by the Editions of SQL Server](http://msdn.microsoft.com/en-us/library/cc645993.aspx)) to determine the appropriate editions to support central management server, policy evaluation, SQL Server Agent, and Reporting Services. All components are supported on SQL Server Enterprise and SQL Server Standard.

This document identifies the steps to configure the Enterprise Policy Management Framework objects in a SQL Server environment.

1. Set up Central Management Server

The EPM Framework requires an instance of SQL Server (2012 or above) designated as a Central Management Server (CMS). The CMS will manage the logical groups of servers. The names of the logical groups are passed into variables in the Enterprise Policy Management Framework for policy evaluation against these groups.

If an instance has not yet been designated as a Central Management Server, execute the following steps:

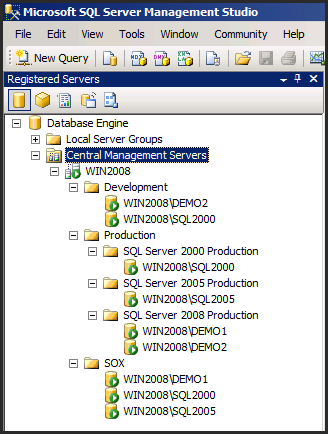
1. Open SQL Server Management Studio. Select the View menu. Click Registered Servers.
2. In Registered Servers, expand Database Engine, right-click Central Management Servers, point to New, and then click Central Management Servers.
3. In the New Server Registration dialog box, register the instance of SQL Server that you want to become the Central Management Server.
4. ****After a SQL Server (2012 or above) instance has been designated as a CMS, create the logical server groups and register SQL Server instances into these logical server group. Logical groups must exist. SQL Server Books Online details the manual steps to register instances into logical server groups: [Create a Central Management Server and Server Group](http://msdn.microsoft.com/en-us/library/bb934126.aspx). Figure 1 is an example of a Central Management Server (individual server groups and registered instances will differ based on each environment).

Figure 1

1. Configure/Create Policies and Centralize on the Central Management Server

All policies that will be evaluated using the EPM framework will be stored and managed on the SQL Server instance defined as the Central Management Server. The PowerShell script will loop through these policies during execution.

Policies stored on the Central Management Server should be configured to improve execution scale and minimize/eliminate false policy failures. The following are best practices for designing policies to scale with the EPM Framework.

**Categorize policies**

The EPM framework PowerShell script must be given a parameter for policy category (example: *Microsoft Best Practices: Performance*). Therefore all policies should be placed in a category to enable a scalable solution for policy evaluation over a large environment.

**Ensure all policies are configured with appropriate server restrictions and defined targets.**

Policies may not be relevant for all versions and/or editions of SQL Server. For example, a policy which checks that a database is enabled for Transparent Data Encryption will not be relevant on SQL Server 2000 or SQL Server 2005, or SQL Server 2008 editions other than Enterprise. Furthermore, this policy may not be relevant for every database on an instance. A policy that checks for existing number of data files might be more relevant for tempDB only. Define server restrictions on polices to eliminate evaluation failures due to incompatible editions. Define targets on polices to eliminate false failures for databases which are not relevant to the policy.

If at a given point in time you choose to stop collecting for a specific policy that belonged to a category YY, instead of just deleting that policy (which would not clear its previous result from the reports), we recommend you move it to a new category named **“Disabled”**, which you would not be collecting results on that category in any job. This way, not only will you keep the policy should you choose to use it again, but also the reports will exclude that specific category and any policies inside, including historic data on any policy inside that “Disabled” category.

Environments may wish to leverage the Best Practices Policies which are installed with SQL Server (Figure 2 below shows the SQL Server 2012 Policies setup) or available from the [SQL Server 2008 R2 Feature Pack](http://www.microsoft.com/en-us/download/details.aspx?id=16978). If installed already, these policies may be imported to the Central Management Server – for example, SQL Server 2012 includes pre-configured best practice policies typically located at *C:\Program Files (x86)\Microsoft SQL Server\110\Tools\Policies\DatabaseEngine\1033*. For further information on this topic, see [How to: Export and Import a Policy-Based Management Policy](http://msdn.microsoft.com/en-us/library/bb522584.aspx).

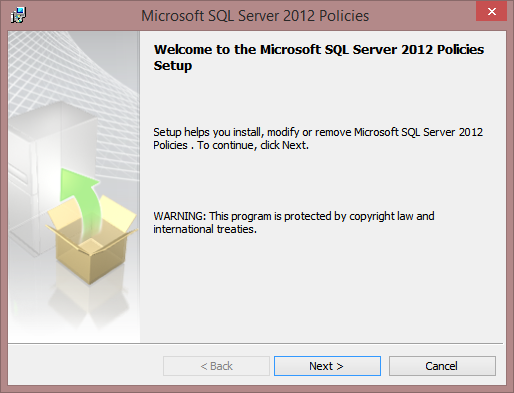


Figure 2

If you choose to import these policies, you may want to run the script **PBM\_Fix\_ImportedPolicies.sql** afterwards for some needed actions. Please read the script notes to understand what is being changed.

1. Create Database and Database Objects To Store Policy Evaluation Results

The script **EPM\_Create\_Database\_Objects.sql** will create the database to store the policy history, the policy schema, the tables, views, functions and stored procedure to support the framework. If installing on an instance that supports Database Compression, then Page Compression will be used in this database.

1. Open the SQLCMD script **EPM\_Create\_Database\_Objects.sql** in SQL Server Management Studio.
2. Change the query execution to SQLCMD Mode. In the menu bar select *Query* -> *SQLCMD* Mode.
3. In the query, configure the variable *ServerName* with the name of the Central Management Server instance.
4. Configure the variable *ManagementDatabase* with the name of the database where the policy evaluation history will be created. See Figure 3.

:SETVAR ServerName "WIN2012"

:SETVAR ManagementDatabase "MDW"

GO

:CONNECT $(ServerName)

GO

Figure 3

1. Execute the script.

Policy evaluation results will be stored in a SQL Server database designated for database management purposes. Many environments may choose to use a Management Data Warehouse created for Data Collectors, but this is not required. Sizing this database will depend on how many policies are evaluated, how many properties are evaluated in each policy, how many instances are evaluated, and how long the historical data will be maintained.

1. Configure PowerShell Script

Update the **EPM\_EnterpriseEvaluation\_412.ps1** PowerShell script variables for the enterprise environment.

1. Open **EPM\_EnterpriseEvaluation\_412.ps1** in Notepad or a PowerShell script editor.
2. Update the variables identified below with your values. See Figure 4.

*$CentralManagementServer:* Required.Centralized location of the SQL Server instance where the policy evaluation history database is located. This is the instance and database where the policy evaluation results are written and policies are stored.

*$HistoryDatabase:* Required. Name of the history database where the policy evaluation results are written.

*$ResultDir:* Required. File location to write the policy evaluation results during the PowerShell execution. Results are written to this location temporarily.

# Declare variables to define the central warehouse

# in which to write the output, store the policies

$CentralManagementServer = "Win2012"

$HistoryDatabase = "MDW"

# Define the location to write the results of the policy evaluation

$ResultDir = "E:\Results\"

Figure 4

1. Save the script in a folder which can be accessed by the scheduling process.
2. Test the PowerShell Script

Review the PowerShell script **EPM\_EnterpriseEvaluation\_412.ps1.** The scriptrequires three parameters. These parameters will support a granular evaluation strategy.

Please also mind the notes in the Security section of Appendix A.

***-ConfigurationGroup****:* Define the Central Management Server group to evaluate. If an empty string (“”), the PowerShell script will evaluate all servers registered in all of the CMS groups.

***-PolicyCategoryFilter****:* Identifies which category of policies will be evaluated. If an empty string (“”), the PowerShell script will evaluate all polices stored on the Central Management Server.

***-EvalMode****:* Specify the action to take during policy evaluation. Options are *Check*, *Configure*. Check will evaluate and report on the evaluation of the policy against the target. Configure will evaluate and report on the evaluation of the policy against the target, and reconfigure any deterministic options that are not in compliance with the policies.

To test the script prior to creating a job, open PowerShell in SQL Server Management Studio.

1. Open SQL Server Management Studio. Connect to the Central Management Server.
2. In *Object Explorer*, right-click on the *Server* and select “*Start PowerShell*”
3. Configure the following commands to your environment. Paste the commands to the SQL Server PowerShell console.

SL "*Insert script folder location*”

.\EPM\_EnterpriseEvaluation\_412.ps1 -ConfigurationGroup "*Insert Central Management Server Group*" -PolicyCategoryFilter "*Insert Policy Category*" –EvalMode “Check”

1. When the script has completed, run the following statements against the database created in step 3. Results should appear from one or both views.

SELECT \* FROM policy.v\_PolicyHistory

GO

SELECT \* FROM policy.v\_EvaluationErrorHistory

GO

1. Create SQL Server Agent Job to Automate Execution of Policy Evaluation

To create the SQL Server Agent job(s):

1. Open SQL Server Management Studio. Connect to the Central Management Server. Open SQL Server Agent.
2. Right-click the Jobs folder and select “*New Job*…”.
3. Name the job. In the left pane select “*Steps*”.
4. Select the “*New*…” at the bottom.
5. In the *New Job Step* window name the *Job Step*. Select PowerShell from the *Type* drop-down.
6. Select the appropriate proxy account in the “Run As” drop-down. This step is required if SQL Server Agent is not running with a domain account that has elevated rights on all instances. See [Use a SQL Agent Proxy for Special Tasks](http://sqlblog.com/blogs/allen_white/archive/2008/05/06/use-a-sql-agent-proxy-for-special-tasks.aspx) for details on configuring a proxy account.
7. Enter the following PowerShell script in the command window. Replace the sample parameter values*.*

SL "E:\PowerShell Scripts\"

.\EPM\_EnterpriseEvaluation\_412.ps1 -ConfigurationGroup "Production" -PolicyCategoryFilter "Microsoft Best Practices: Performance" –EvalMode “Check”

After SL, place the folder location where the **EPM\_EnterpriseEvaluation\_412.ps1** is stored from step 4. In the next line, configure the parameters with a Central Management Server Group and a Policy Category. The above is an example which will evaluate all servers in the Production Group and subgroups with the policies in the category “Microsoft Best Practices: Performance”

1. Select OK to save the job step. See Figure 5 for an example of the job step.
2. In the *New Job* window, select “*Schedule*” in the left pane. Configure the schedule to meet the evaluation requirements.
3. Select OK to save job.

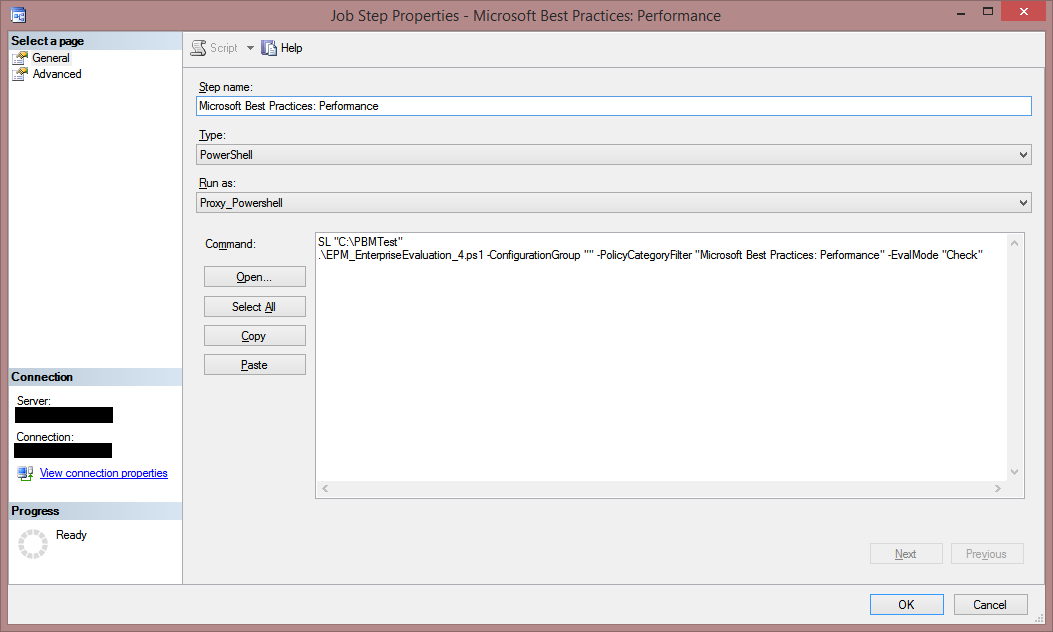


Figure 5

If you created one job per category, and you want several jobs to execute at once, keep in mind that some job sub systems do not allow more than *2* amount of simultaneous threads, such as the case of PowerShell. As such, you might want to update SQL Agent PowerShell simultaneous threads to a larger number, say 10, using the following example:

UPDATE msdb.dbo.syssubsystems

SET max\_worker\_threads = 10

WHERE subsystem = N'Powershell'

GO

**Note**: Running the above command requires a restart of the SQL Agent on SQL Server 2008 and SQL Server 2008R2.

**Note2**: The above no longer works in SQL Server 2012 or above, where the max\_worker\_thread value will assume its default value of 2 after a SQL Agent restart.

1. Deploy Reports to SQL Server Reporting Services

After the PowerShell script was successfully executed and the results of policy evaluation were collected in the management data warehouse, it is very convenient to visualize the data through Reporting Services reports. Steps below describe the process of report configuration and deployment.

Configure the Project properties with SSRS deployment options.

1. Double-click *..\ 2Reporting\PolicyReports.sln* to open the *PolicyReports* project in Visual Studio, Business Intelligence Development Studio or SQL Server Data Tools.
2. In the right *Solution Explorer* pane, right-click *PolicyDW.rds* in the *Shared Data Sources* folder and select Open.
3. Configure the connection string to the Central Management Server instance and database which stores the policy history. Select OK to save.
4. Right click on the *PolicyReports* project and select *Properties*. See Figure 6. Set the appropriate ***TargetServerUrl***, ***TargetReportFolder*** and ***TargetDataSourceFolder*** properties of the “*PolicyReports*” project. These will align with the SQL Server Reporting Services instance.

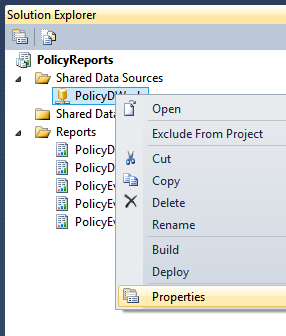


Figure 6

The visual indicators in the *Last Execution Status* table and the *Failed Policy % By Month* chart of the *PolicyDashboard* report will dynamically change color (see Figure 7) based on thresholds configured in hidden parameters of the report.

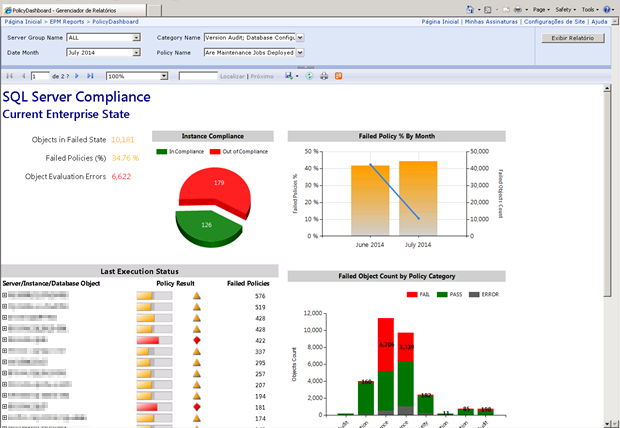


Figure 7

1. Double-click the *PolicyDashboard* report in the *Reports* folder to open.
2. In the menu, select *View* -> *Report Data*.
3. Open the *Parameters* folder in the left *Report Data* tab. See Figure 8.

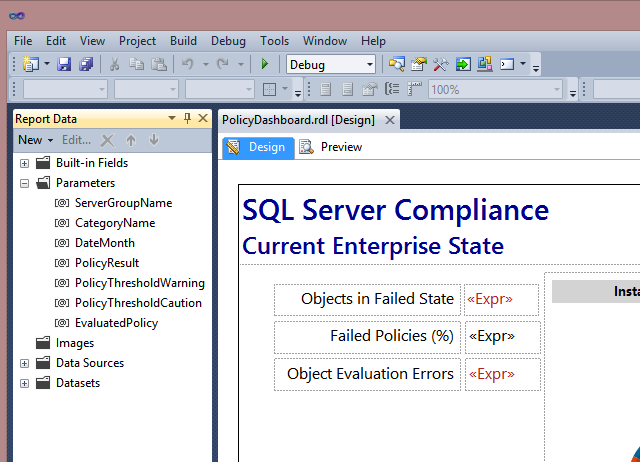


Figure 8

1. Right-click **PolicyThresholdWarning** and select *Parameter Properties*.
2. In the *Report Parameter Properties* dialog (see Figure 9), select *Default Values*.

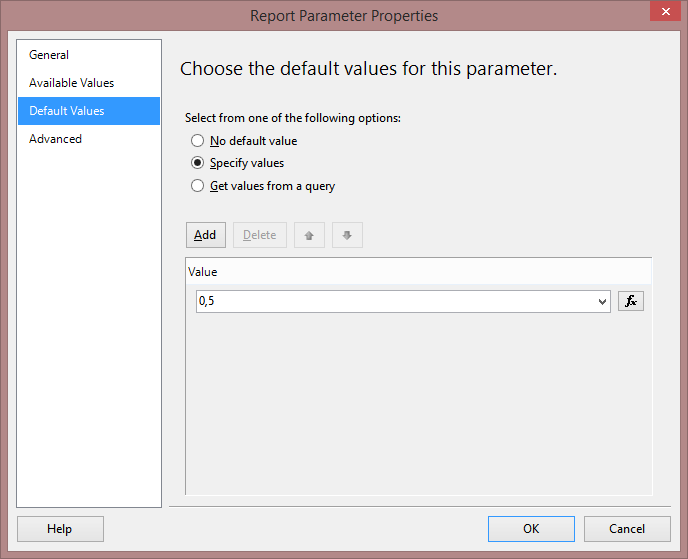


Figure 9

1. Change the value to the appropriate value for your environment. The default is **0.5** which will cause the report objects to use a Red color when the failed policy average falls below 50%. Select OK to save.
2. Right-click ***PolicyThresholdCaution*** and select *Parameter Properties*.
3. In the *Report Parameter Properties* dialog, select *Default Values*.
4. Change the value to the appropriate value for your environment. The default is **0.17** which will cause the report objects to use an orange color when the failed policy average falls between 17% and 50%. Select OK to save.   
   When the failed policy average falls between 17% a Yellow color is used.
5. In the right pane *Solution Explorer*, right-click on the *PolicyReports* project and select *Deploy*.

## **Appendix A: Additional Considerations**

### Security

When polices are evaluated through PowerShell, they will execute the policy evaluation in the context of the user issuing the evaluation. This account will require access to all instances and database objects the script will evaluate. The level of permissions will depend on what the policy is evaluating. This extends to the execution account used in a scheduling agent. In SQL Server, the SQL Server Agent job step executes in the context of a specific user. This user may be a proxy account. The account that is specified in the SQL Server Agent step must have access to all objects on all instances that the PowerShell script will be evaluating.

Prior to setting up the SQL Server Agent job, be sure to set up a Proxy Account that has the appropriate rights on the remote instances to evaluate the policies. See [Use a SQL Agent Proxy for Special Tasks](http://sqlblog.com/blogs/allen_white/archive/2008/05/06/use-a-sql-agent-proxy-for-special-tasks.aspx) for details on configuring a proxy account.

Note that the PowerShell script execution policy might need to be changed in the Central Management Server host to *RemoteSigned*, to allow script execution.

Also, by downloading the script, it might need to be unblocked, as highlighted in Figure 10:

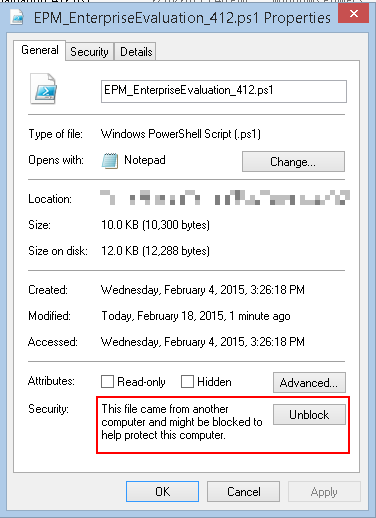


Figure 10

### Archive

The data stored in the table *policy.PolicyHistory* is not used to report through the EPM Framework reports. Environments concerned with data space used by the EPM Framework will want to set up a regular process to purge data from this table, unless it is necessary to maintain for compliance.

Below is an example of a cycle that will purge data over 90 days, which can be used inside a SQL Agent job.

WHILE (SELECT COUNT(PolicyHistoryID) FROM MDW.policy.PolicyHistory WHERE EvaluationDateTime > DATEADD(dd, DATEDIFF(dd, 0, GETDATE()), -90)) > 0

BEGIN

DELETE TOP (5000) FROM MDW.policy.PolicyHistory

WHERE EvaluationDateTime < DATEADD(dd, DATEDIFF(dd, 0, GETDATE()), -90)

END;

DELETE FROM MDW.policy.EvaluationErrorHistory

WHERE EvaluationDateTime < DATEADD(dd, DATEDIFF(dd, 0, GETDATE()), -90);

GO

### Execution Strategies

Prior to creating the SQL Server Agent job(s), determine the execution strategy for the EPM framework PowerShell script **EPM\_EnterpriseEvaluation\_412.ps1** based on scale, execution time and security. Create SQL Server Agent jobs to execute the policy evaluation according to the strategy.

Administrators managing larger environments may want to implement policy evaluation through multiple parallel jobs that guarantee evaluation completion in the desired scope of time. A large environment that is challenged with a small maintenance window to execute the evaluation may design an execution strategy where the total number of concurrent jobs equals the total number of cores available for the instance.

* + **EPM\_EnterpriseEvaluation\_412.ps1** may be configured with multiple SQL Server Agent jobs for each policy category and/or each group of servers.
  + **EPM\_EnterpriseEvaluation\_412.ps1** may be configured with a single SQL Server Agent job and multiple steps for each policy category and/or each group of servers.

The script may also be configured to evaluate all policies against all instances (groups) by passing blank strings in as the –*ConfigurationGroup* and –*PolicyCategoryFilter* parameters. This configuration is not recommended for environments with a large number of instances and a large number of policies.

The EPM Framework version 4 supports nested server groups defined in the Central Management Server. Note in Figure 10 that the group “*Production*” has three nested groups. When a parent group is used as the parameter –*ConfigurationGroup*, all instances registered to the parent group and all instances registered to child groups will be included in the evaluation.

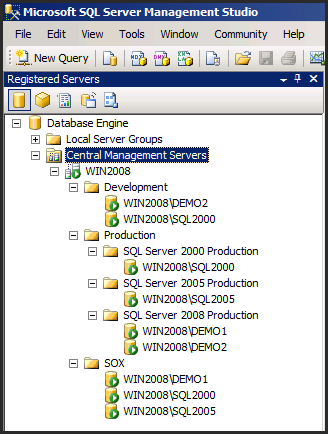


Figure 11

## Appendix B: Database Objects

### Tables

**policy.PolicyHistory**: Stores evaluation results from the PowerShell script. The XML results are shred into the table PolicyHistoryDetail. This data is not used for reporting and may be archived and/or purged as deemed necessary.

**policy.PolicyHistoryDetail**: Stores the shredded results from the EvaluationResults column in policy.PolicyHistory table. This table is the source for all policy result views.

**policy.EvaluationErrorHistory**: Stores the following types of errors:

* Server connection errors
* Evaluation process cannot write to the PolicyHistory table
* Policy cannot evaluate against an object

### Views

**policy.v\_PolicyHistory** : Returns the full historical dataset. Depends on policy.PolicyHistoryDetail.

**policy.v\_PolicyHistory\_Rank**: Ranks the policy results for each object on each server by date. Depends on policy.v\_PolicyHistory.

**policy.v\_PolicyHistory\_LastEvaluation**: Returns the last execution status for each policy evaluation against each object on each server. Depends on policy.v\_PolicyHistory\_Rank.

**policy.v\_EvaluationErrorHistory**: Returns all evaluation errors from the EvaluationErrorHistory table combined with the PolicyHIstoryDetail table.

**policy.v\_EvaluationErrorHistory\_LastEvaluation:** Returns the ranked errors from the v\_EvaluationErrorHistory view. The last error for each evaluation is returned when the query includes a filter on EvaluationOrderDesc = 1.

**policy.v\_ServerGroups:** Returns the server groups. Used in reports to drive parameters.

### Stored Procedure

**policy.epm\_Load\_PolicyHistoryDetail**: Executed during the PowerShell evaluation. Shreds the PolicyHistory XML results and stores in policy.PolicyHistoryDetail and policy.EvaluationErrorHistory.

### Function

**policy.pfm\_ServerGroupInstances**: Table-valued function to return the instances registered in a specified Central Management Server group.

## **Appendix C: Upgrade from previous version**

### Database Objects

If upgrading from the previous version of EPM, simply execute **EPM\_Upgrade\_Database\_Objects.sql** to add or change the required objects, without loss of data to your current deployment.

### PowerShell Script

Replace **EPM\_EnterpriseEvaluation\_3.0.0.ps1** or **EPM\_EnterpriseEvaluation\_4xx.ps1** with **EPM\_EnterpriseEvaluation\_412.ps1**, and change the reference in all your jobs accordingly. Refer to step 4 (Configure PowerShell Script) for further information on the script usage.

### Reports

Deploy the new reports as detailed in step 7 (Deploy Reports to SQL Server Reporting Services).