Troubleshooting Microsoft Exchange Server with PowerShell

exchangeserverpro.com

About the Author

Paul is a Microsoft Exchange Server MVP and is the publisher of Exchange Server Pro.

He is also an MCP, MCSA, MCSE, MCTS, and an MCITP for Exchange Server 2007/2010. Connect with Paul on Twitter, LinkedIn and Google+.

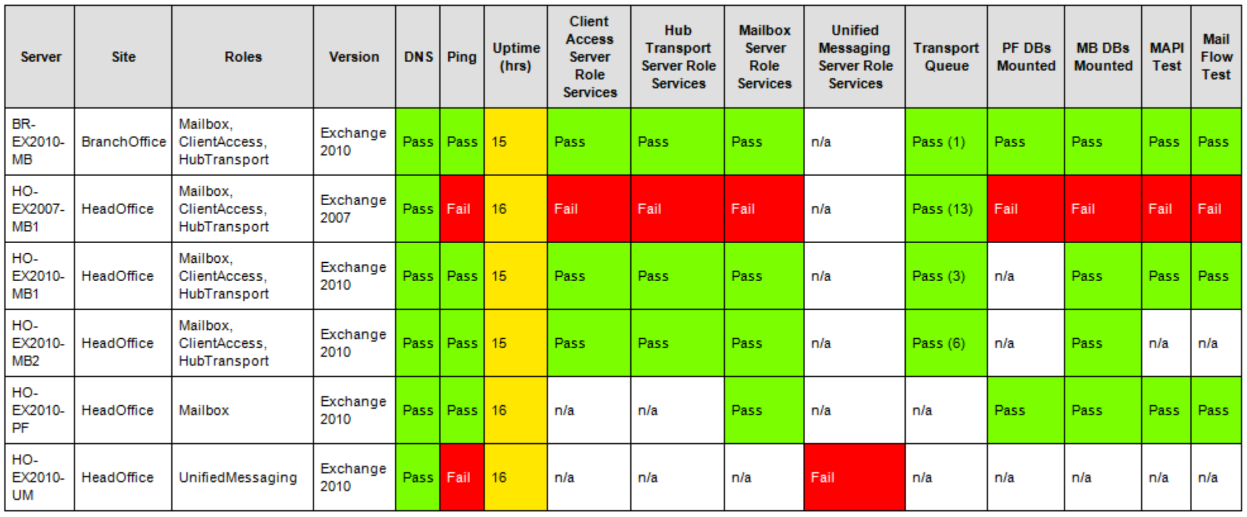
Table of Contents

Introduction

PowerShell is known to be a more efficient way to administer a Microsoft network environment. But sometimes we forget that it can also make *troubleshooting* our environments more efficient as well.

When I first published the Test-ExchangeServerHealth.ps1[1] PowerShell script one of the earliest pieces of feedback I received was for more information on how to troubleshoot the problems that the script detects.

After all, if you are running the Test-ExchangeServerHealth.ps1 script in your environment and you are presented with a health report that looks like this, where do you start troubleshooting?



Server	Site	Roles	Version	DNS	Ping	Uptime (hrs)	Client Access Server Role Services	Hub Transport Server Role Services	Mailbox Server Role Services	Unified Messaging Server Role Services	Transport Queue	PF DBs Mounted	MB DBs Mounted	MAPI Test	Mail Flow Test
BR-EX2010-MB	BranchOffice	Mailbox, ClientAccess, HubTransport	Exchange 2010	Pass	Pass	15	Pass	Pass	Pass	n/a	Pass (1)	Pass	Pass	Pass	Pass
HO-EX2007-MB1	HeadOffice	Mailbox, ClientAccess, HubTransport	Exchange 2007	Pass	Fail	16	Fail	Fail	Fail	n/a	Pass (13)	Fail	Fail	Fail	Fail
HO-EX2010-MB1	HeadOffice	Mailbox, ClientAccess, HubTransport	Exchange 2010	Pass	Pass	15	Pass	Pass	Pass	n/a	Pass (3)	n/a	Pass	Pass	Pass
HO-EX2010-MB2	HeadOffice	Mailbox, ClientAccess, HubTransport	Exchange 2010	Pass	Pass	15	Pass	Pass	Pass	n/a	Pass (6)	n/a	Pass	n/a	n/a
HO-EX2010-PF	HeadOffice	Mailbox	Exchange 2010	Pass	Pass	16	n/a	n/a	Pass	n/a	n/a	Pass	Pass	Pass	Pass
HO-EX2010-UM	HeadOffice	UnifiedMessaging	Exchange 2010	Pass	Fail	16	n/a	n/a	n/a	Fail	n/a	n/a	n/a	n/a	n/a

Fortunately Microsoft Exchange Server includes a number of built-in PowerShell test cmdlets that can be used to diagnose and troubleshoot problems.

Of course, you don't need to be running this script to benefit from learning about the test cmdlets in Exchange. The use of these cmdlets applies to any troubleshooting situation, whether it be in response to a monitoring alarm, a support ticket, or just because someone has asked you if everything is running okay.

This guide will step through the most commonly used PowerShell test cmdlets that ship with Exchange Server 2013. If you are not running Exchange Server 2013 in your environment yet don't worry, most of these cmdlets work just the same in Exchange 2007 and 2010 as well, so you will still learn some useful tricks.

[1] http://exchangeserverpro.com/powershell-script-health-check-report-exchange-2010/

Exchange Server 2013 PowerShell Test Cmdlets

Exchange Server 2013 comes with a set of PowerShell cmdlets that can be used to test the health and functionality of your servers.

The cmdlets themselves have descriptive names such as Test-Mailflow, Test-MAPIConnectivity, Test-ActiveSyncConnectivity, and so on. You can see the full list by using Get-Command.

On Exchange Server 2013:

```
[PS] C:\>Get-Command -Verb Test | Where Module -match $env:computername
```

On Exchange Server 2007 or 2010:

```
[PS] C:\>Get-Command -Verb Test
```

Tip: You could also just run "Get-Command –Verb Test". The point of filtering the output by Module is to keep some other Test-* cmdlets that aren't part of the Exchange module from appearing in the results.

Here is a brief description of the purpose of each of the test cmdlets available in Exchange 2013.

Cmdlet	Description
Test-ActiveSyncConnectivity	Tests the Microsoft Exchange ActiveSync by performing a synchronization for a specified mailbox.
Test-ArchiveConnectivity	Verifies connectivity to the archive mailbox for a specified user.
Test-AssistantHealth	Verifies that the Microsoft Exchange Mailbox Assistants service is healthy, and can also attempt to fix any problems that are

	detected.
Test-CalendarConnectivity	Verifies that anonymous calendar sharing is enabled and working properly on one or all Client Access servers.
Test-EcpConnectivity	Verifies that the Exchange Admin Center is running. This cmdlet retained the Exchange 2010 acronym “ECP” which stood for Exchange Control Panel.
Test-EdgeSynchronization	Verifies the synchronization status of the Edge Transport servers subscribed to a site.
Test-ExchangeSearch	Verifies that Exchange Search is enabled and is indexing new messages.
Test-FederationTrust	Verifies that the federation trust (a relationship between the Exchange organization and the Microsoft Federation Gateway) is configured correctly.
Test-FederationTrustCertificate	Checks the certificates on each Exchange server that are being used for federation.
Test-ImapConnectivity	Test the IMAP functionality of a Client Access server.
Test-IPAllowListProvider	Tests an IP address against a specified IP allow list provider. Only applicable when an Edge Transport server is installed in your environment.
Test-IPBlockListProvider	Tests an IP address against a specified IP block list provider. Only applicable when an Edge Transport server is installed in your environment.
Test-IRMConfiguration	Tests the Information Rights Management (IRM) configuration and functionality.
Test-Mailflow	Test email send and receive functionality for the system mailbox on a Mailbox server.
Test-MAPIConnectivity	Tests MAPI functionality by logging onto the system mailbox or a mailbox that you specify.
Test-MigrationServerAvailability	Test the availability of the target server for various cross-forest and cloud migration scenarios.
Test-MRSHealth	Tests the health of the Microsoft Exchange Mailbox Replication

	service.
Test-OAuthConnectivity	Test OAuth authentication for applications.
Test-OrganizationRelationship	Verifies the organization relationship (for federation) is configured correctly.
Test-OutlookConnectivity	Tests end-to-end Outlook client connectivity.
Test-OutlookWebServices	Verifies the Autodiscover, Availability, Outlook Anywhere, Offline address book, and Unified Messaging services for a mailbox.
Test-OwaConnectivity	This cmdlet still exists but has been deprecated. Use Get-ServerHealth instead.
Test-PopConnectivity	Test the POP functionality of a Client Access server.
Test-PowerShellConnectivity	Tests whether PowerShell remoting is functioning on a Client Access server.
Test-ReplicationHealth	Tests the replay and replication health of Mailbox servers in a database availability group.
Test-SenderId	Tests whether an IP address is a permitted sender for a domain.
Test-ServiceHealth	Tests whether all of the required services for an Exchange server role are running.
Test-SiteMailbox	Tests a site mailbox's connectivity to SharePoint, and whether users have correct permissions to use the site mailbox.
Test-SmtpConnectivity	Tests the SMTP connectivity to receive connectors on a server.
Test-UMConnectivity	Tests the Unified Messaging service functionality of a Mailbox server.
Test-WebServicesConnectivity	Tests Exchange Web Services functionality for Outlook Anywhere.

As you can see there are quite a lot of test cmdlets available for administrators to use. In reality some of the test cmdlets are more commonly used than others, and a few are primarily used by services such as System Center Operations Manager (SCOM) rather than by the administrators.

Though it is not practical to explore every test cmdlet in depth here, in the next section we'll take a closer look at some of the test cmdlets that you may find yourself using more frequently than others.

Creating the Test Mailbox User

Some of the PowerShell test cmdlets in Exchange Server 2013 rely on the administrator providing a mailbox credential for the test, or alternatively they can use a special mailbox user created specifically for use by the test cmdlets.

You can create this mailbox user on a Mailbox server by running the **new-TestCasConnectivityUser.ps1** script provided by Microsoft. Running the script on a Mailbox server will create the test user on that server.

```
[PS] C:\>cd $exscripts

[PS] C:\Program Files\Microsoft\Exchange Server\V15\scripts>.\new-
TestCasConnectivityUser.ps1

Please enter a temporary secure password for creating test users. For security
purposes, the password will be changed regularly and automatically by the system.
Enter password: ***********

Create test user on: E15MB1.exchange2013demo.com
Click CTRL+Break to quit or click Enter to continue.:

UserPrincipalName: extest_39de530f5ee44@exchange2013demo.com

You can enable the test user for Unified Messaging by running this command with
the following optional parameters : [-UMDialPlan <dialplanname> -UMExtension
<numDigitsInDialplan>] . Either None or Both must be present.
```

There are only a few ways that script can go wrong, such as not providing a password that is complex enough for your password policy, or the script being unable to determine the OU to place the user account object in. If necessary use the **–OU** parameter to specify which OU the account should be created in.

Using Test-ReplicationHealth to Test DAG Members

The Test-ReplicationHealth cmdlet checks the status of the cluster, network, log replication and log replay for Mailbox servers in a database availability group.

The Test-ReplicationHealth cmdlet can be run on a Mailbox server that is a member of a database availability group.

```
[PS] C:\>Test-ReplicationHealth

Server          Check                      Result     Error
------          -----                      ------     -----
E15MB2          ClusterService             Passed
E15MB2          ReplayService              Passed
E15MB2          ActiveManager              Passed
E15MB2          TasksRpcListener           Passed
E15MB2          TcpListener                Passed
E15MB2          ServerLocatorService       Passed
E15MB2          DagMembersUp               Passed
E15MB2          ClusterNetwork             Passed
E15MB2          QuorumGroup                Passed
E15MB2          DatabaseRedundancy         Passed
E15MB2          DatabaseAvailability       Passed
E15MB2          DBCopySuspended            Passed
E15MB2          DBCopyFailed               Passed
E15MB2          DBInitializing             Passed
E15MB2          DBDisconnected             Passed
E15MB2          DBLogCopyKeepingUp         Passed
E15MB2          DBLogReplayKeepingUp       Passed
```

You can also run the cmdlet against a remote server.

```
[PS] C:\>Test-ReplicationHealth -Identity E15MB2

Server          Check                      Result     Error
------          -----                      ------     -----
E15MB2          ClusterService             Passed
E15MB2          ReplayService              Passed
E15MB2          ActiveManager              Passed
E15MB2          TasksRpcListener           Passed
E15MB2          TcpListener                Passed
E15MB2          ServerLocatorService       Passed
```

```
E15MB2          DagMembersUp               Passed
E15MB2          ClusterNetwork             Passed
E15MB2          QuorumGroup                Passed
E15MB2          DatabaseRedundancy         Passed
E15MB2          DatabaseAvailability       Passed
```

The cmdlet also accepts pipeline input, however if you were to simply pipe Get-MailboxServer into it and you have Mailbox servers in the organization that are not DAG members then you risk seeing errors in your results.

Instead you can pipe only the members of a database availability group into Test-ReplicationHealth using the following method:

```
[PS] C:\>Get-DatabaseAvailabilityGroup | select -ExpandProperty:Servers | Test-
ReplicationHealth
```

Note: The number of tests shown in the output of Test-ReplicationHealth will vary depending on whether the DAG member has only active or passive databases on it at the time. Not all tests are relevant if the server has only active, or only passive databases on it.

Using Test-MAPIConnectivity to Test Mailbox Databases

The Test-MAPIConnectivity PowerShell cmdlet will test the availability and latency of your mailbox databases and help you to troubleshoot issues with mailbox access.

Running Test-MAPIConnectivity on a Mailbox server will test the active mailbox databases on that server.

```
[PS] C:\>Test-MAPIConnectivity

MailboxServer      Database           Result    Error
-------------      --------           ------    -----
E15MB2             Mailbox Database 2 Success
```

You can also specify a remote Mailbox server to test all of the active databases on that server. Notice what happens if a Mailbox server with no active databases is tested.

```
[PS] C:\>Test-MAPIConnectivity -Server E15MB1

MailboxServer      Database           Result    Error
-------------      --------           ------    -----
E15MB1             Mailbox Database 1 Success

[PS] C:\>Test-MAPIConnectivity -Server E15MB3
WARNING: The operation could not be performed because no mailbox database is
currently hosted on server E15MB3.
```

The test will use the system mailbox on each database, but you can also specify a mailbox and the test will run on whichever database that mailbox is hosted on.

This is useful if you have a need to verify connectivity for a specific mailbox.

```
[PS] C:\>Test-MAPIConnectivity -Identity paul.cunningham

MailboxServer      Database           Result    Error
-------------      --------           ------    -----
E15MB1             Mailbox Database 1 Success
```

Individual mailbox databases can also be tested directly.

```
[PS] C:\>Test-MAPIConnectivity -Database "Mailbox Database 1"

MailboxServer      Database           Result    Error
-------------      --------           ------    -----
E15MB1             Mailbox Database 1 Success
```

And of course you can use the pipeline to test multiple databases at the same time. You can also pipe the output to Format-List to see more details, such as the latency recorded by the test.

```
[PS] C:\>Get-MailboxDatabase | Test-MAPIConnectivity | fl


RunspaceId  : d02eb1e1-f94c-48ee-9384-b99b0654be4a
Server      : E15MB1
Database    : Mailbox Database 1
Mailbox     : SystemMailbox{a24b3b31-8f3b-4164-8e5c-bd68426fd53e}
MailboxGuid : 80d382a3-1017-4530-8c80-7ee4a40d208b
IsArchive   : False
Result      : Success
Latency     : 00:00:00.0221344
Error       :
Identity    :
IsValid     : True
ObjectState : New

RunspaceId  : d02eb1e1-f94c-48ee-9384-b99b0654be4a
Server      : E15MB2
Database    : Mailbox Database 2
```

```
Mailbox     : SystemMailbox{b53f08b3-b4cb-4f05-a1c1-3c9e71277527}
MailboxGuid : 5bf1697f-2cb2-4c32-93df-f49ac5885cd5
IsArchive   : False
Result      : Success
Latency     : 00:00:00.0135938
Error       :
Identity    :
IsValid     : True
ObjectState : New
```

If necessary you can override the default latency threshold of 90 seconds and specify your own value (in seconds).

```
[PS] C:\>Get-MailboxDatabase | Test-MAPIConnectivity -AllConnectionsTimeout 120
```

Tip: When someone asks me “Hey, is there something wrong with Exchange?”, one of the first cmdlets I will run is Test-MAPIConnectivity.

Using Test-ServiceHealth to Verify Required Services are Running

Test-ServiceHealth will check that the required services for the Exchange Server roles installed on a server are running. This includes services that Exchange itself installs, as well as those in the operating system that Exchange depends on.

Running Test-ServiceHealth on a server will check the services on the local server.

```
[PS] C:\>Test-ServiceHealth


Role                    : Mailbox Server Role
RequiredServicesRunning : True
ServicesRunning         : {IISAdmin, MSExchangeADTopology, MSExchangeDelivery,
                          MSExchangeIS, MSExchangeMailboxAssistants,
                          MSExchangeRepl, MSExchangeRPC, MSExchangeServiceHost,
                          MSExchangeSubmission, MSExchangeThrottling,
                          MSExchangeTransportLogSearch, W3Svc, WinRM}
ServicesNotRunning      : {}

Role                    : Client Access Server Role
RequiredServicesRunning : True
ServicesRunning         : {IISAdmin, MSExchangeADTopology,
                          MSExchangeMailboxReplication, MSExchangeRPC,
                          MSExchangeServiceHost, W3Svc, WinRM}
ServicesNotRunning      : {}

Role                    : Unified Messaging Server Role
RequiredServicesRunning : True
ServicesRunning         : {IISAdmin, MSExchangeADTopology, MSExchangeServiceHost,
                          MSExchangeUM, W3Svc, WinRM}
ServicesNotRunning      : {}

Role                    : Hub Transport Server Role
RequiredServicesRunning : True
ServicesRunning         : {IISAdmin, MSExchangeADTopology, MSExchangeEdgeSync,
                          MSExchangeServiceHost, MSExchangeTransport,
                          MSExchangeTransportLogSearch, W3Svc, WinRM}
ServicesNotRunning      : {}
```

You will notice that the server roles in the output above do not align with the new server role architecture of Exchange Server 2013. This may change in a future version of the cmdlet, but for now you can still use the results to identify required services that are not running.

The cmdlet can also be run against remote servers. In Exchange Server 2013 this can be a remote Mailbox server or a remote multi-role server, but unfortunately running it against a remote Client Access server will return an error[2].

```
[PS] C:\>Test-ServiceHealth -Server E15MB3
```

If a required service is not running you will see the RequiredServicesRunning result set to False, and the ServicesNotRunning result will list the names of the services.

```
[PS] C:\>Test-ServiceHealth -Server E15MB3


Role                    : Mailbox Server Role
RequiredServicesRunning : False
ServicesRunning         : {IISAdmin, MSExchangeADTopology, MSExchangeDelivery,
                          MSExchangeIS, MSExchangeMailboxAssistants,
                          MSExchangeRepl, MSExchangeRPC, MSExchangeServiceHost,
                          MSExchangeSubmission, MSExchangeThrottling,
                          MSExchangeTransportLogSearch, WinRM}
ServicesNotRunning      : {W3Svc}
```

You can quickly start a service using Invoke-Command to issue the command to the remote server.

```
[PS] C:\>Invoke-Command -ComputerName E15MB3 {Start-Service W3Svc}
```

Tip: If the WinRM service is not running then PowerShell remoting will not work, so you will need to log on directly to the server or use the Services management tool to remotely connect.

[2] http://exchangeserverpro.com/exchange-2013-test-servicehealth-error/

Using Test-Mailflow to Verify End to End Mail Delivery

The Test-Mailflow cmdlet allows you to test the delivery of email between two mailboxes. The test will be performed using system mailboxes or you can optionally specify other mailboxes to test.

Running the cmdlet on a Mailbox server will test the local server.

```
[PS] C:\>Test-Mailflow


RunspaceId         : aecc19a4-5571-43cf-affd-b893db3cfab9
TestMailflowResult : Success
MessageLatencyTime : 00:00:45.1720635
IsRemoteTest       : False
Identity           :
IsValid            : True
ObjectState        : New
```

Notice that the output includes the result (eg, “Success”) as well as the message latency.

So not only are you able to test that mail flow is working, but you can also see whether email delivery is slow, possibly indicating a server or network issue somewhere.

The cmdlet can be used to test mail flow between two Mailbox servers, as long as each server has at least one active mailbox database at the time (so that there is a system mailbox available on each server).

In Exchange Server 2013 this works as long as the local server (E15MB1 in the example below) is the server you’re connected to at the time.

If you specified a remote server then the test will fail. In previous versions of Exchange this issue doesn't exist[3].

```
[PS] C:\>Test-Mailflow E15MB1 -TargetMailboxServer E15MB2


RunspaceId         : aecc19a4-5571-43cf-affd-b893db3cfab9
TestMailflowResult : Success
MessageLatencyTime : 00:00:05.4297550
IsRemoteTest       : True
Identity           :
IsValid            : True
ObjectState        : New
```

In addition to specifying a server to test, you can also specify a database.

```
[PS] C:\>Test-Mailflow E15MB1 -TargetDatabase "Mailbox Database 2"


RunspaceId         : aecc19a4-5571-43cf-affd-b893db3cfab9
TestMailflowResult : Success
MessageLatencyTime : 00:00:55.9304181
IsRemoteTest       : True
Identity           :
IsValid            : True
ObjectState        : New
```

Or you can be even more specific by testing mail flow to an internal email address.

```
[PS] C:\>Test-Mailflow E15MB1 -TargetEmailAddress
paul.cunningham@exchange2013demo.com
```

[3] http://exchangeserverpro.com/exchange-2013-test-mailflow-error-for-remote-mailbox-servers/

Using Test-ActiveSyncConnectivity to Verify Exchange ActiveSync

The Test-ActiveSyncConnectivity cmdlet allows you to simulate an Exchange ActiveSync connection from a mobile device to a mailbox. The mailbox can either be the test user mailbox you created earlier, or a specific mailbox user.

Running the cmdlet on a Client Access server will test the local server.

```
[PS] C:\>Test-ActiveSyncConnectivity

CasServer  LocalSite     Scenario        Result  Latency(MS)
---------  ---------     --------        ------  -----------
e15mb1     Sydney        Options         Success
e15mb1     Sydney        FolderSync      Success
e15mb1     Sydney        First Sync      Success
e15mb1     Sydney        GetItemEstimate Success
e15mb1     Sydney        Sync Data       Success
e15mb1     Sydney        Ping            Success
e15mb1     Sydney        Sync Test Item  Success
```

The cmdlet can also be used to test a remote Client Access server.

```
[PS] C:\>Test-ActiveSyncConnectivity -ClientAccessServer E15MB2
```

You can also test a specific URL.

```
[PS] C:\>Test-ActiveSyncConnectivity -URL
https://mail.exchange2013demo.com/Microsoft-Server-ActiveSync
```

If you have not provisioned the correct SSL certificates you can override the SSL trust requirement and still perform the test.

```
[PS] C:\>Test-ActiveSyncConnectivity -TrustAnySSLCertificate
```

You can also use the pipeline to test multiple Client Access servers together.

```
[PS] C:\>Get-ClientAccessServer | Test-ActiveSyncConnectivity

CasServer  LocalSite     Scenario        Result  Latency(MS)
---------  ---------     --------        ------  -----------
e15mb1     Sydney        Options         Success
e15mb1     Sydney        FolderSync      Success
e15mb1     Sydney        First Sync      Success
e15mb1     Sydney        GetItemEstimate Success
e15mb1     Sydney        Sync Data       Success
e15mb1     Sydney        Ping            Success
e15mb1     Sydney        Sync Test Item  Success
e15mb2     Sydney        Options         Success
e15mb2     Sydney        FolderSync      Success
e15mb2     Sydney        First Sync      Success
e15mb2     Sydney        GetItemEstimate Success
e15mb2     Sydney        Sync Data       Success
e15mb2     Sydney        Ping            Success
e15mb2     Sydney        Sync Test Item  Success
```

Finally, if you need to test a specific mailbox you can pass the credentials for that mailbox user to the cmdlet.

```
[PS] C:\>$credential = Get-Credential -UserName e2013demo\paul.cunningham -Message
"Enter password"

[PS] C:\>Test-ActiveSyncConnectivity -MailboxCredential $credential
```

Using Test-OutlookWebServices to Verify Web Services Functionality

The Test-OutlookWebServices cmdlet allows you to test the functionality of the following services:

- Autodiscover
- Exchange Web Services
- Availability Service
- Offline Address Book

Running the cmdlet on a Client Access server will test the local server using the test mailbox user created earlier.

```
[PS] C:\>Test-OutlookWebServices

Source                                              Scenario  Result
------                                              --------  ------
E15MB1.exchange2013demo.com AutoDiscoverOutlookProvider Success
E15MB1.exchange2013demo.com         ExchangeWebServices Success
E15MB1.exchange2013demo.com         AvailabilityService Success
E15MB1.exchange2013demo.com          OfflineAddressBook Success
```

You can also perform the test for a specific mailbox by using the –Identity and –MailboxCredential parameters.

```
[PS] C:\>Get-ClientAccessServer | Test-OutlookWebServices -Identity
paul.cunningham@exchange2013demo.com -MailboxCredential (Get-Credential)
```

Tip: Testing a specific mailbox is useful if you are troubleshooting problems with one or more of the Outlook Web Services in a particular site within your organization. You can compare results between test mailboxes in different sites to help you narrow down the source of any problems you're seeing.

Using Test-MRSHealth to Verify the Mailbox Replication Service

The Test-MRSHealth cmdlet can be used to verify that the Mailbox Replication Service on Exchange Server 2013 Mailbox servers is healthy. This service is responsible for processing mailbox move requests, so a healthy MRS will be important any time you are performing migrations.

Running the cmdlet on a Mailbox server will test the local server.

```
[PS] C:\>Test-MRSHealth


RunspaceId  : e9dc1305-6b80-4e92-a8b7-9efb06e0894f
Check       : ServiceCheck
Passed      : True
Message     : The Mailbox Replication Service is running.
Identity    : E15MB1
IsValid     : True
ObjectState : New

RunspaceId  : e9dc1305-6b80-4e92-a8b7-9efb06e0894f
Check       : RPCPingCheck
Passed      : True
Message     : The Microsoft Exchange Mailbox Replication service is responding to
              a RPC ping. Server version:
              15.0.620.24 caps:3F.
Identity    : E15MB1
IsValid     : True
ObjectState : New

RunspaceId  : e9dc1305-6b80-4e92-a8b7-9efb06e0894f
Check       : QueueScanCheck
Passed      : True
Message     : The Microsoft Exchange Mailbox Replication service is scanning
              mailbox database queues for jobs. Last scan age: 00:05:35.4810000.
Identity    : E15MB1
IsValid     : True
ObjectState : New
```

The items of most interest are the Check, Passed, and possibly the Message values in the results. If a check has not passed then the message will assist you with identifying why.

With those three attributes in mind it is quite easy to test multiple Mailbox servers with a single cmdlet and output a neat report with the results.

```
[PS] C:\>Get-MailboxServer | Test-MRSHealth | Select Identity,Check,Passed,Message
| ft -auto

Identity          Check Passed Message
--------          ----- ------ -------
E15MB1     ServiceCheck   True The Mailbox Replication Service is running.
E15MB1     RPCPingCheck   True The Microsoft Exchange Mailbox Replication service
                               is responding to a RPC ping. Serve...
E15MB1   QueueScanCheck   True The Microsoft Exchange Mailbox Replication service
                               is scanning mailbox database queue...
E15MB2     ServiceCheck   True The Mailbox Replication Service is running.
E15MB2     RPCPingCheck   True The Microsoft Exchange Mailbox Replication service
                               is responding to a RPC ping. Serve...
E15MB2   QueueScanCheck   True The Microsoft Exchange Mailbox Replication service
                               is scanning mailbox database queue...
E15MB3     ServiceCheck   True The Mailbox Replication Service is running.
E15MB3     RPCPingCheck   True The Microsoft Exchange Mailbox Replication service
                               is responding to a RPC ping. Serve...
E15MB3   QueueScanCheck   True The Microsoft Exchange Mailbox Replication service
                                is scanning mailbox database queue...
```

Using Test-PowerShellConnectivity to Verify PowerShell Remoting

The Test-PowerShellConnectivity cmdlet can be used to verify that PowerShell remoting is functioning correctly.

Running the cmdlet will test the local server.

```
[PS] C:\>Test-PowerShellConnectivity

CasServer  LocalSite     Scenario        Result  Latency(MS)
---------  ---------     --------        ------  -----------
E15MB1     Sydney        Logon User      Success      312.50
```

You can also test a remote server.

```
[PS] C:\>Test-PowerShellConnectivity E15MB3

CasServer  LocalSite     Scenario        Result  Latency(MS)
---------  ---------     --------        ------  -----------
E15MB3     Sydney        Logon User      Success      140.63
```

And you can use the pipeline to test multiple servers together.

```
[PS] C:\>Get-ExchangeServer | Test-PowerShellConnectivity

CasServer  LocalSite     Scenario        Result  Latency(MS)
---------  ---------     --------        ------  -----------
E15MB1     Sydney        Logon User      Success      203.12
E15MB2     Sydney        Logon User      Success      218.79
E15MB3     Sydney        Logon User      Success      187.51
```

Although this may not be useful for ad-hoc situations it can come in handy when writing scripts. You can use Test-PowerShellConnectivity before running a series of other cmdlets that rely on remoting.

```
param(
	[Parameter( Mandatory=$true)]
	[string]$server
)


#Check Powershell Connectivity first
if ((Test-PowerShellConnectivity $server).Result.Value -eq "Success")
{
    Write-Host "PowerShell connectivity test successful"
    #Run other commands
}
else
{
    Write-Host "PowerShell connectivity test failed"
}
```

Other Useful PowerShell Cmdlets

In addition to the test cmdlets demonstrated in the previous sections of this guide there are also a number of other PowerShell cmdlets in Microsoft Exchange Server that are useful for troubleshooting situations.

Using Get-Queue to Troubleshoot Transport Queues

When you suspect a mail flow problem (such as a failed Test-MailFlow result, or a failed mail flow test in the health check script report) looking at your Transport queues is usually the next step.

The Get-Queue cmdlet will show you the current queues on a Transport server. Running the cmdlet on its own will test the local server.

```
[PS] C:\>Get-Queue

Identity                   DeliveryType           Status MessageCount
--------                   ------------           ------ ------------
HO-EX2010-MB1\13827        SmtpRelayWithinAdSite  Ready  11
HO-EX2010-MB1\13829        SmtpRelayToRemoteAdSite Retry  1
HO-EX2010-MB1\Submission   Undefined              Ready  0
HO-EX2010-MB1\Shadow\13458 ShadowRedundancy       Ready  0
```

You can also run it against a remote Transport server using the –Server switch.

```
[PS] C:\>Get-Queue -Server ho-ex2010-mb1
```

If you notice a specific queue that has a high message count you can target that queue, and pipe the output to Get-Message to look closer at the messages that are in that queue.

```
[PS] C:\>Get-Queue HO-EX2010-MB1\13827 | Get-Message

Identity                FromAddress             Status
--------                -----------             ------
HO-EX2010-MB1\13827\... Kim.Taylor@exchanges... Retry
HO-EX2010-MB1\13827\... Helen.Cail@exchanges... Retry
HO-EX2010-MB1\13827\... Carol.Okyere@exchang... Retry
HO-EX2010-MB1\13827\... Joy.Sian@exchangeser... Retry
HO-EX2010-MB1\13827\... Marcia.Barnett@excha... Retry
HO-EX2010-MB1\13827\... Suki.Murray@exchange... Retry
HO-EX2010-MB1\13827\... Lorraine.Oza@exchang... Retry
HO-EX2010-MB1\13827\... Lesley.Taggart@excha... Retry
```

Taking it one step further you can look at the last error for messages that are stuck in a queue to give you a clue as to why they are not delivering.

```
[PS] C:\>Get-Queue HO-EX2010-MB1\13827 | Get-Message | select
FromAddress,LastError

FromAddress                            LastError
-----------                            ---------
Pradip.Rasulian@exchangeserverpro.net  452 4.3.1 Insufficient system resources
Kim.Taylor@exchangeserverpro.net       452 4.3.1 Insufficient system resources
Helen.Cail@exchangeserverpro.net       452 4.3.1 Insufficient system resources
Carol.Okyere@exchangeserverpro.net     452 4.3.1 Insufficient system resources
Joy.Sian@exchangeserverpro.net         452 4.3.1 Insufficient system resources
Marcia.Barnett@exchangeserverpro.net   452 4.3.1 Insufficient system resources
Suki.Murray@exchangeserverpro.net      452 4.3.1 Insufficient system resources
Lorraine.Oza@exchangeserverpro.net     452 4.3.1 Insufficient system resources
Lesley.Taggart@exchangeserverpro.net   452 4.3.1 Insufficient system resources
Debbie.Dalgliesh@exchangeserverpro.net 452 4.3.1 Insufficient system resources
```

There are many different reasons that messages will get stuck in a queue, so the best thing to do is look at the LastError and if it does not immediately make sense to you start searching online for an explanation of what that error message means.

Using Get-MailboxDatabase to Troubleshoot Databases

Even though Test-MAPIConnectivity will tell you about the health and status of a mailbox database there are a few other things of interest that it can’t tell you. For those we can use the Get-MailboxDatabase cmdlet.

A very useful switch for Get-MailboxDatabase is the –Status switch, which returns live status information about the database rather than just configuration attributes from Active Directory.

For example, here is the Get-MailboxDatabase output showing the name and mount status of each database without using the –Status switch.

```
[PS] C:\>Get-MailboxDatabase | Select Name,Mounted

Name          Mounted
----          -------
MB-HO-01
MB-HO-02
MB-BR-01
MB-HO-Archive
MB-BR-02
MB-HO-04
MB-HO-03
```

Here is the same output, this time with the –Status switch used.

```
[PS] C:\>Get-MailboxDatabase -Status | Select Name,Mounted

Name          Mounted
----          -------
MB-HO-01         True
MB-HO-04         True
MB-HO-03         True
MB-HO-02         True
MB-BR-01         True
MB-BR-02         True
MB-HO-Archive    True
```

Another handy usage of Get-MailboxDatabase is checking the last backup timestamp for your mailbox databases, and whether a backup is currently in progress.

```
[PS] C:\>Get-MailboxDatabase -Status | Select
Name,LastFullBackup,LastIncrementalBackup,BackupInProgress

Name          LastFullBackup        LastIncrementalBackup BackupInProgress
----          --------------        --------------------- ----------------
MB-HO-01      9/11/2013 10:00:15 AM                                  False
MB-HO-04      9/11/2013 10:00:16 AM                                  False
MB-HO-03                                                             False
MB-HO-02      9/3/2013 11:59:38 PM                                   False
MB-BR-01      4/27/2013 2:31:18 AM                                   False
MB-BR-02                                                             False
MB-HO-Archive 4/27/2013 2:31:17 AM                                   False
```

Using Get-MailboxDatabaseCopy to Troubleshoot Database Copies in a DAG

Mailbox database copies and content indexes in a DAG can quietly fail and go unnoticed because the active database copy may still be online.

The Get-MailboxDatabaseCopy cmdlet will show you the health of your database copies. Running the cmdlet with a * (wildcard) will show you all databases.

```
[PS] C:\>Get-MailboxDatabaseCopyStatus *

Name                    Status CopyQueueLength ReplayQueueLength ContentIndexState
----                    ------ --------------- ----------------- -----------------
MB-HO-01\HO-EX2010-MB1 Mounted               0                 0           Healthy
MB-HO-03\HO-EX2010-MB1 Mounted               0                 0            Failed
MB-HO-02\HO-EX2010-MB1  Failed           43570                 4           Healthy
MB-HO-04\HO-EX2010-MB1 Mounted               0                 0           Healthy
MB-HO-01\HO-EX2010-MB2 Healthy               0                 0           Healthy
MB-HO-02\HO-EX2010-MB2 Mounted               0                 0           Healthy
MB-HO-04\HO-EX2010-MB2 Healthy               0                 0           Healthy
MB-BR-01\BR-EX2010-MB  Mounted               0                 0           Healthy
MB-BR-02\BR-EX2010-MB  Mounted               0                 0           Healthy
MB-HO-Archive\HO-EX2010-PF Mounted           0                 0           Healthy
```

The output of Get-MailboxDatabaseCopy lets you see at a glance which database copies are active (mounted), which are health (passive), and which are not healthy (failed, or other reasons such as suspended or seeding). You can also see the content index status, which may be healthy or failed independent of the status of the database itself.

You can also check the database copies on just one specific server.

```
[PS] C:\>Get-MailboxDatabaseCopyStatus -Server ho-ex2010-mb1

Name                   Status CopyQueueLength ReplayQueueLength ContentIndexState
----                   ------ --------------- ----------------- -----------------
MB-HO-01\HO-EX2010-MB1 Mounted              0                 0           Healthy
MB-HO-03\HO-EX2010-MB1 Mounted              0                 0            Failed
MB-HO-02\HO-EX2010-MB1  Failed          43570                 4           Healthy
MB-HO-04\HO-EX2010-MB1 Mounted              0                 0           Healthy
```

A Final Word

I hope by now you are starting to see how useful PowerShell can be for troubleshooting Exchange Server issues.

Although it is impossible to cover every possible problem scenario I hope that by reading this guide you've picked up a few tips and tricks that you can start to use on the job.

I strongly encourage you to use PowerShell as much as possible when troubleshooting Exchange servers. The more you use PowerShell the more familiar you will become with it, and the more you will find yourself rapidly diagnosing problems in your Exchange organization.

Have you enjoyed this guide and found it useful?

If so please feel free to share it with your friends and colleagues on Twitter, Facebook, LinkedIn and other social networks.