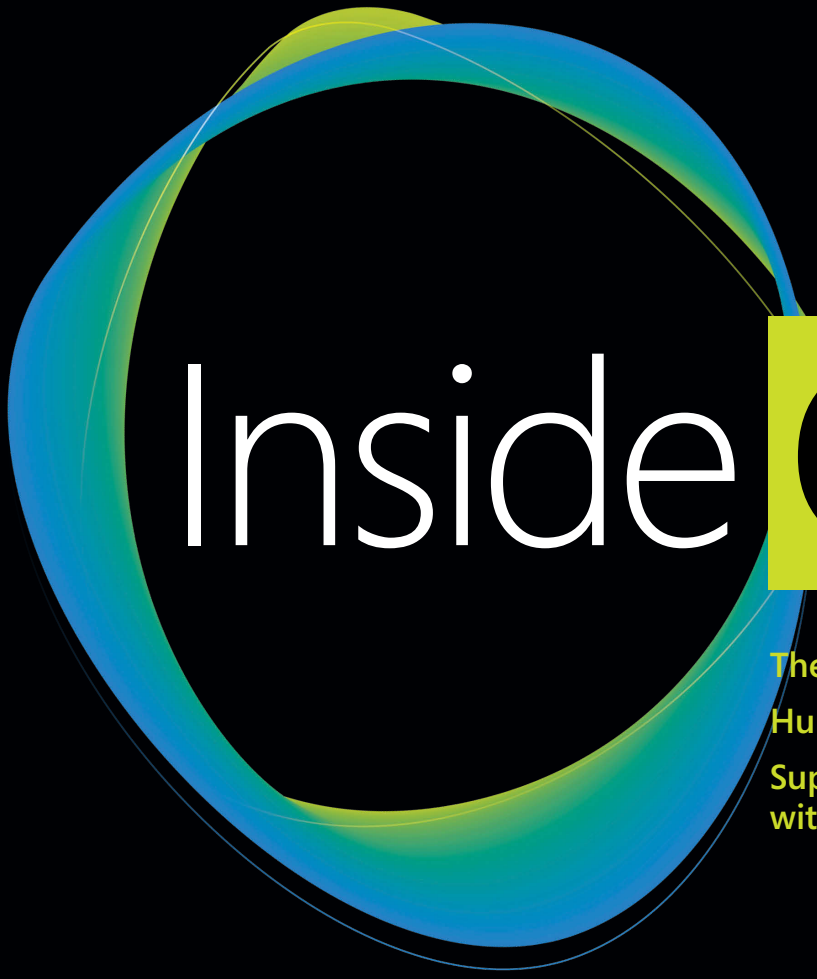
Microsoft



Inside OUT

The ultimate, in-depth reference

Hundreds of timesaving solutions

Supremely organized, packed with expert advice

Microsoft Exchange Server 2013: Connectivity, Clients, and UM

Paul Robichaux *Microsoft MVP for Exchange Server*

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2013948709
ISBN: 978-0-7356-7837-8

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at http://www.microsoft.com/learning/booksurvey.

Acquisitions Editor: Anne Hamilton
Developmental Editor: Karen Szall
Project Editor: Karen Szall
Editorial Production: nSight, Inc.
Technical Reviewer: Tony Redmond; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Copyeditor: Kerin Forsyth
Indexer: Lucie Haskins
Cover: Twist Creative • Seattle

Contents at a Glance

Table of Contents

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

Foreword for Exchange 2013 Inside Out books

Those seeking an in-depth tour of Exchange Server 2013 couldn't ask for better guides than Tony Redmond and Paul Robichaux. Tony and Paul have a relationship with the Exchange team that goes back two decades, to the days of Exchange 4.0. Few people have as much practical knowledge about Exchange, and even fewer have the teaching skills to match. You are in good hands.

Over the past few years, we have seen significant changes in the way people communicate; a growing number of devices, an explosion of information, increasingly complex compliance requirements, and a multigenerational workforce. This world of communication challenges has been accompanied by a shift toward cloud services. As we designed Exchange 2013, the Exchange team worked hard to build a product and service that address these challenges. As you read these books, you'll get an up-close look at the outcome of our efforts.

Microsoft Exchange Server 2013 Inside Out: Mailbox and High Availability covers foundational topics such as the Exchange Store, role-based access control (RBAC), our simplified approach to high availability, and the new public folder architecture. It also covers our investments in eDiscovery and in-place hold. As you read, you'll see how Exchange 2013 helps you achieve world-class reliability and provides a way to comply with internal and regulatory compliance requirements without the need for third-party products.

Microsoft Exchange Server 2013 Inside Out: Connectivity, Clients, and UM explores the technologies that give users anywhere access to their email, calendar, and contacts across multiple devices. It also explains how to protect your email environment from spam, viruses, and other threats and describes how Exchange 2013 can connect with Office 365 so you can take advantage of the power of the cloud.

From our new building-block architecture to data loss prevention, there's a lot to explore in the newest version of Exchange. I hope that as you deploy and use Exchange 2013, you'll agree that this is an exciting and innovative release.

Enjoy!

Rajesh Jha
Corporate Vice President - Exchange
Microsoft Corporation

Introduction

This book is for experienced Exchange administrators who want to gain a thorough understanding of how client access, transport, unified messaging, and Office 365 integration work in Exchange Server 2013, the latest version of the Microsoft enterprise messaging server first released in October 2012 and updated on a frequent basis since. It isn't intended to be a reference, and it isn't suitable for novices.

In 2011, when Tony Redmond and I were working together to present the Exchange 2010 Maestro workshops in cities throughout the United States, we spent a lot of time talking about the nature of an ideal Exchange book. It should be comprehensive enough to cover all the important parts of Exchange, with enough detail to be valuable to even very experienced administrators but without just parroting Microsoft documentation and guidance. As far as possible, it should draw on real-world experience with the product, which of course takes time to produce. Out of those talks came Tony's idea to write not one but two books on Exchange 2013. A single book would either be unmanageably large, both for author and reader, or would omit too much important material to be useful.

Although Tony's *Exchange 2013 Inside Out: Mailbox and High Availability* (Microsoft Press, 2013) draws on his long and broad experience with the nuances of the Exchange mailbox role and how to put it to work, this book covers all the other things Exchange does, including client access, transport, unified messaging, and the increasingly important topic of Office 365 integration. Because Exchange 2013 is an evolution of Exchange 2010, we decided to use *Microsoft Exchange Server 2010 Inside Out* (Microsoft Press, 2010) as the base for the new book. For the topics in this book, so much has changed since Exchange 2010 that only a small amount of the original material remains. The rest is new and was written to take into account the many changes and updates that Exchange 2013 has undergone since its original release.

I have had the good fortune to work with and around Exchange for nearly 20 years. During this time, I've seen the Exchange community, product team, and product evolve and grow in ways that might not have been predictable back in 1996. If you went back to, say, 2000 and told the Exchange product group, "Hey, in 2013, your product will be deployed to hundreds of millions of users worldwide, many with tiny handheld computers that are more powerful than your desktop, and a whole bunch of them running as a Microsoft-hosted service," you'd be bound to get some skeptical looks, and yet here we are.

I hope that you enjoy this book and that you'll read it alongside Tony's *Microsoft Exchange Server 2013 Inside Out: Mailbox and High Availability*. The two books really do go together. Tony and I exchanged technical editing duties for our respective books, so we share responsibility for any errors you might find.

Acknowledgments

I was incredibly fortunate to receive a great deal of help with this book from a variety of sources. A large group of Exchange experts from the Microsoft Most Valuable Professional (MVP) and Microsoft Certified Systems Master (MCSM) communities volunteered their time to read early drafts of the chapters as they were produced; their mission was to identify shortcomings or errors and to suggest, based on their own experience, ways in which the book could be improved. This book is much better thanks to their efforts, which I very much appreciate. My thanks to Kamal Abburi, Thierry Demorre, Devin Ganger, Steve Goodman, Todd Hawkins, Georg Hinterhofer, Miha Pihler, Maarten Piederiet, Simon Poirier, Brian Reid, Brian R. Ricks, Jeffrey Rosen, Mitch Roberson, Kay Sellenrode, Bhargav Shukla, Thomas Stensitzki, Richard Timmering, Steven van Houttum, Elias VarVarezis, Johan Veldhuis, and Jerrid Williams. My thanks also go to the broader MCM and MVP communities, particularly Paul Cunningham, Brian Desmond, and Pat Richard, for discussing topics or sharing scripts that informed the material I wrote.

In addition to these volunteers, I benefited greatly from the efforts of many people from the product team, including Diego Carlomagno, Bulent Egilmez, David Espinoza, Kern Hardman, Pavani Haridasyam, Tom Kaupe, Roy Kuntz, Lou Mandich, Jon Orton, Tony Smith, Greg Taylor, and Mini Varkey. Extra thanks to Rajesh Jha for taking the time to write the foreword for both books—no easy task considering how often Tony and I have hassled him about various matters.

Finally, you wouldn't have this book at all if it weren't for the stalwart efforts of Karen Szall, Valerie Woolley, and a cast of dozens at Microsoft Press. Karen never lost her temper despite the many vigorous discussions we had about my failure to meet deadlines or my obstinacy toward some of the requirements imposed by the Microsoft crack legal department. Thanks to them all for producing such a good-looking finished product.

Errata & book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site at:

http://aka.ms/EXIOv2/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

CHAPTER 1

Client access servers

The Exchange Client Access Server (CAS) role in Exchange 2013 is a critical part of delivering the features and functionality that users depend on.

In Exchange Server 4.0, Exchange Server 5.0, and Exchange Server 5.5, client access was provided by the single server role that then existed. Exchange 2000 introduced the notion of a *front-end* server—a server that didn't necessarily have any mailbox data but to which clients could connect to reach a server that *did* have mailbox data. Exchange 2007 gave us the first iteration of the CAS role, and that role was enhanced in Exchange 2010.

Since its introduction in Exchange 2007, the CAS role has been responsible for three types of traffic:

- External connections from Internet clients running any of the supported protocols offered by Exchange.
- Internal connections from intranet clients, again using any supported protocol.
- Connections that were proxied or redirected from other CAS servers. These connections might come from CAS servers running the same version of Exchange, earlier versions, or later versions.

However, the way in which the CAS role handles this traffic, the nature of the protocols supported, and the implementation behind this support have changed significantly in Exchange 2013. The Exchange 2013 CAS role now has two primary tasks: to authenticate user requests and locate the correct server to handle the user's request.

Take a look.

CAS architecture demystified

In Exchange 2013, the CAS has evolved further into what appears on the surface to be a simple proxy that handles client connections. However, a great deal is going on below the apparently simple surface, and you explore it in this chapter. How did the CAS role reach this point? As Exchange has changed over time, Microsoft has steadily worked to separate three related parts of Exchange that began life as a set of closely coupled subsystems:

- The code that handles mailbox storage, transport, and processing. The Information Store service is the best-known part of this code, but lots of other components contribute to moving messages between sender and recipient and then storing them for future use.
- The code that handles interactions with clients, including retrieving messages from the Store; formatting messages for a particular client (such as Outlook Web App); or providing client services for synchronization, message addressing, and so on.
- The business logic that Exchange uses to determine whether a request or data item is valid. For example, the Exchange business logic is supposed to catch whenever an application requests creating a corrupt item, such as a calendar item whose ending time is before its start time.

Figure 1-1 shows the results of this architectural approach in Exchange 2010. Protocol components on the server on the left communicate with both the protocol and storage layers on the right. The business logic layers on a server communicate with the protocols and storage layers on the same server *and* the same layers on other servers. This causes all sorts of actual and potential problems. For example, an older client access server might not know how to proxy specific types of traffic or protocol requests that should be sent to a newer-version CAS. This architecture also has so many dependencies among layers (both on the same server and across servers) that deploying Exchange in anything but the simplest topology required extra redundancy, such as guaranteeing that both a Hub Transport and CAS server would be in each site that had a Mailbox server.

The design goals for Exchange 2013 included a sweeping redesign of all three layers and the way they interoperate and communicate. The phrase "every server is an island" has been tossed around by various Microsoft engineers, and it neatly captures one of the main goals: eliminating linkages between disparate layers across servers so that the protocol layer on one server will only communicate with the protocol layer on other servers, never the storage or business logic layers. In this model, there should be no contact from the storage or business logic layers on one server with any layer on another server. Another goal was to eliminate the need for the CAS to maintain information about the clients with which it was communicating or the contents or *state* of their sessions.

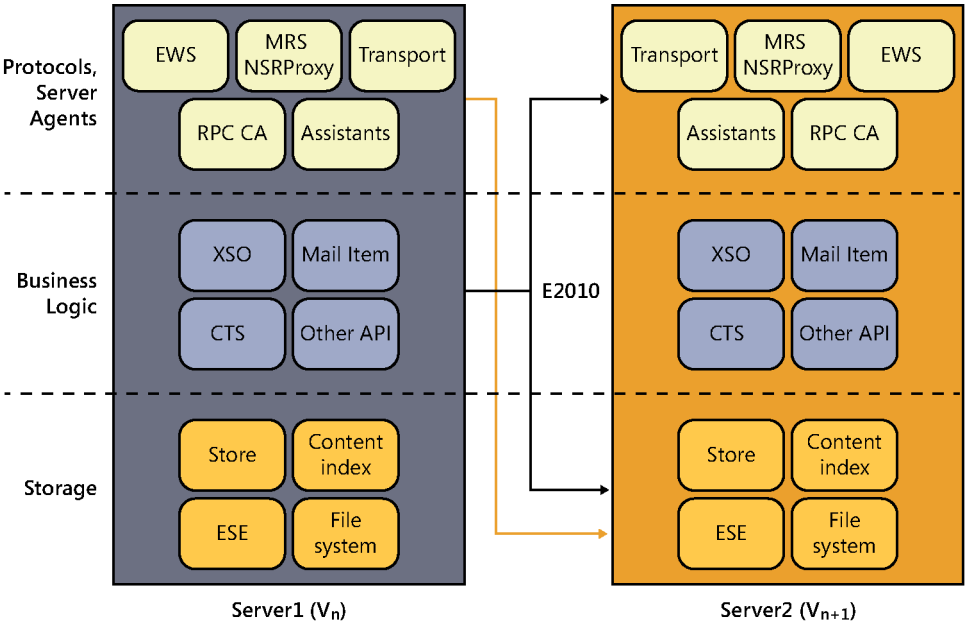


Figure 1-1 The Exchange 2010 architecture

These changes result in the architecture shown in Figure 1-2. Note that all the communications between protocol handlers now take place directly with the corresponding protocol handlers on another server. This essentially turns the CAS role into a stateless proxy that does not render or process client data (although it does publish some data of interest to clients). The CAS authenticates the user connection, determines where the correct target for the requested protocol or services is, and either redirects or proxies the client to that target. That's it. To be more precise, the CAS offers the following services:

- Client protocol access for IMAP, POP, Outlook Web App, the Exchange Administration Center (EAC), Exchange ActiveSync, and Exchange Web Services (EWS). The CAS proxies or redirects traffic for these protocols to the appropriate Mailbox server.
- Proxying requests for the Offline Address Book (OAB) to an available Mailbox server so that compatible clients can download OAB updates as they become available.
- Autodiscover, the client-oriented service that enables a compatible mobile or desktop client to find service endpoints for mailbox access, Outlook Web App, mobile device sync, and unified messaging.
- Front End Transport (FET), which accepts inbound SMTP traffic and proxies it to an Exchange 2013 Mailbox server or an Exchange 2007/2010 hub transport server. FET doesn't store or queue any messages.

- The Unified Messaging Call Router service (UMCR), which redirects incoming unified messaging requests to the appropriate Mailbox server. (For more on UMCR, see Chapter 6, "Unified messaging.")
- Proxied connections to the Availability service, which provides free/busy information for users in the organization.
- A proxy engine for the Mailbox Replication service (MRS); the MRS proxy accepts requests from outside the organization for cross-forest mailbox moves, imports, and exports and then redirects them to the appropriate Mailbox server. (For more on MRS and the role of the proxy component, see *Microsoft Exchange Server 2013 Inside Out: Mailbox and High Availability* (Microsoft Press, 2013) by Tony Redmond.)
- Initial authentication for all the services it supports; for example, the CAS would authenticate an inbound EWS request before sending it elsewhere.

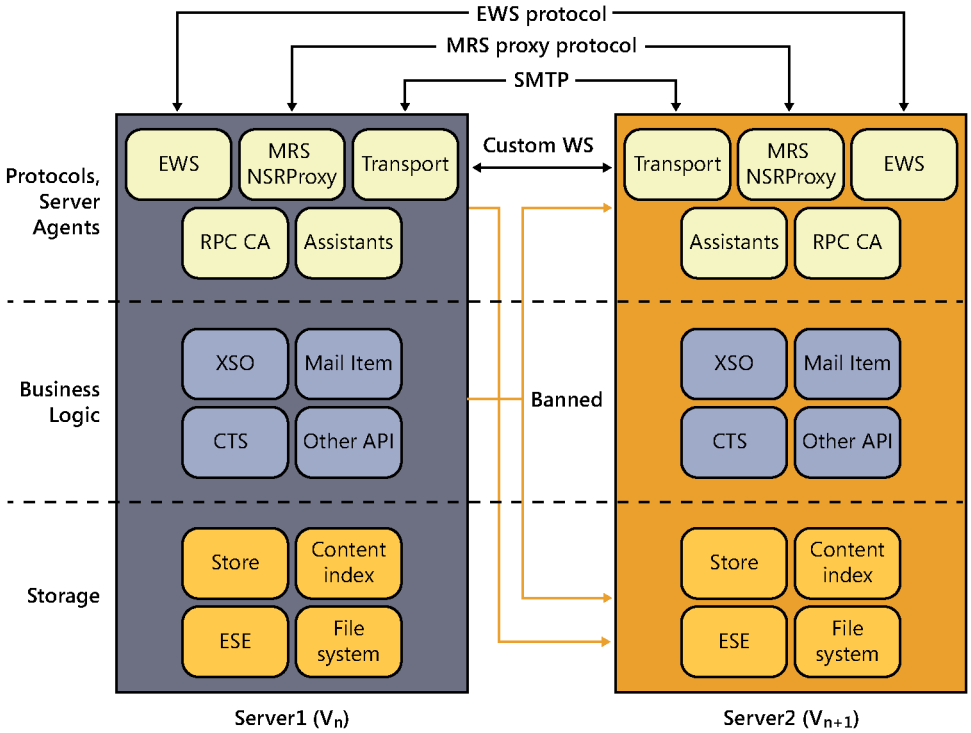


Figure 1-2 The Exchange 2013 architecture

It is also interesting to note what the Exchange 2013 CAS role does *not* do. In particular, it does not provide direct access for Messaging Application Programming Interface

(MAPI) clients using remote procedure calls (RPC) directly over TCP. This change essentially means that the RPC client access (RCA) layer is no longer present on the Exchange 2013 CAS (it now lives on the Mailbox role), so the CAS role now only has to deal with Outlook Anywhere instead of with direct connections.

Why did Microsoft make this change? It turns out that two primary factors drive the change: the Microsoft desire to improve the robustness of client connections to the Mailbox server and the ongoing need to simplify the code underlying the product. Both of these factors, in turn, are driven by the emergence of Office 365.

INSIDE OUT Oh no! Microsoft TMG is gone! What am I going to do now?

When Microsoft announced in September 2012 that it would retire its Threat Management Gateway (TMG) product, there was quite an uproar in the Exchange community. That's because TMG is the best-known reverse proxy solution that supports Exchange. With TMG out of the picture, many customers worried that they would no longer be able to secure their Exchange deployments adequately. This turns out to be a needless worry. Here's why.

First, if you currently have TMG, it will be supported until 2022 or so. At Microsoft, Greg Taylor uses the analogy of a pickup truck: if you have a truck now, it doesn't stop working and become useless because the manufacturer stops making new models. Of course, sometime before 2022, Microsoft likely will release a version of Windows Server that TMG doesn't support, but that isn't a problem you have to solve right now.

Second, Microsoft still sells the Forefront Unified Access Gateway (UAG) product, which works perfectly well with Exchange. It is harder to understand and configure, and it's more expensive than TMG, but it's still supported.

In addition, other vendors have stepped in to fill the void left by the absence of TMG. In particular, Kemp Technologies has shipped its Exchange Security Pack (ESP), which functions as a capable reverse proxy for Exchange that provides preauthentication, supports Windows PowerShell, and includes a number of other nifty features. Competitors such as F5 Networks and Cisco also have reverse proxy solutions that work well with Exchange.

A number of companies are still selling appliances that run TMG, so if you really must have TMG, this might be an option for you.

Most important, you should question whether you actually need a reverse proxy at all. When Exchange 2003 and Internet Security and Acceleration (ISA) Server 2003 shipped,

the security of both Exchange and Windows was shaky. Since then, Microsoft has made great strides in hardening both products, and it's reasonable to ask whether you need a separate reverse proxy at all. After all, when you think about what a load balancer does, it is essentially a packet filter; it only allows traffic on TCP port 443 to Exchange, and it might even do preauthentication. As the time approaches for sunsetting your existing TMG deployment, you should consider whether you need *any* reverse proxy. The Exchange team blog has an interesting post by the aforementioned Greg Taylor at *http://blogs.technet.com/b/exchange/archive/2013/07/17/life-in-a-post-tmg-world-is-it-as-scary-as-you-think.aspx* that outlines some arguments for and against a reverse proxy.

Remember that in the 2013 CAS architecture, connections asking for mailbox data will always be made only to the active copy of the mailbox database that contains the requested data. That means that 2013 CAS needs a way to identify which mailbox database it needs to talk to, not merely the server that contains (or used to contain) it. In Exchange 2007, clients connected to the RPC endpoint; in Exchange 2010, clients connected to an FQDN that represents the RPC endpoint (for instance, HSV-MBX14.contoso.com). This FQDN could point to a CAS array object or directly to an individual CAS. If the mailbox databases hosting the user's mailbox were moved due to a failover or switchover, the client had to update its local MAPI connection profile to reflect the change, and this requires the client to be restarted. In Exchange 2013, by contrast, Outlook profiles now use a globally unique identifier (GUID) representing the mailbox as the endpoint name to connect to. This GUID, which is just a property on the mailbox, remains the same no matter which server has the active mailbox database copy; the CAS can resolve the GUID to the particular server that has the active copy of the mailbox database. This approach means that the 2013 CAS can seamlessly connect to the new active copy of a database without interrupting its connection to the client, so the client will never even be aware that a different copy has become active.

Because each Exchange 2013 CAS and Mailbox server can independently determine which Mailbox server should receive traffic for a particular mailbox, the RpcClientAccessServer property on the mailbox database is no longer necessary; it's still present, but Exchange 2013 ignores it.

Another equally important side effect of this change is that the need for the RPC client access array object has vanished. You might recall that the point of this object, often just called a CAS array, was to provide a single name (and thus a single connection point) for your CAS servers (whether you had one or many) so that clients could address any CAS in the array. Now that any Exchange 2013 CAS can authenticate an incoming request and proxy it to the correct Mailbox server, it no longer matters to the client which CAS it

communicates with, so having a logical object for clients to connect to is no longer necessary. Note that CAS servers are still treated as though they're in a logical array when you put them behind a load balancer; there's no longer a need for an Exchange-specific object.

INSIDE OUT **Don't put firewalls between CAS and Mailbox servers**

Like its ancestors, Exchange 2013 does not support the deployment of firewalls between the CAS and Mailbox roles. It is not possible to deploy CAS servers in the perimeter network with a firewall protecting the Mailbox servers because Exchange uses too many open ports to make most security professionals happy. You can have Windows Firewall configured on servers because Exchange will configure it to allow communications automatically, but not on a hardware firewall.

See *http://technet.microsoft.com/en-us/library/bb331973.aspx* for a list of ports Exchange 2010 uses; Microsoft has not yet released an updated list for 2013, but there are few significant changes.

CAS authentication methods

Many Exchange administrators never tangle with the issues surrounding client authentication because the default settings that Exchange uses for a single-version installation just work. However, as the topology becomes more complex, or when you begin mixing versions, the type of authentication enabled becomes of great importance.

The authentication method matters because the Exchange 2013 CAS role will always send the requests it receives to other servers, and those servers will expect authentication information about the user who is connecting. If the user's mailbox is on an Exchange 2013 Mailbox server, the CAS can proxy directly to the HTTP proxy endpoint. If, however, the user's mailbox is on an Exchange 2007 or Exchange 2010 server, the CAS proxies to the Outlook Anywhere endpoint defined on that server. More properly, the 2013 CAS proxies Outlook Anywhere requests to a virtual directory named /rpc on the older server. The settings on this virtual directory govern which authentication methods the downlevel server will accept. This is why you have to enable Outlook Anywhere on all older CASs running on internal-facing sites.

There are actually three areas where you can specify authentication on the CAS, and they are independent of one another. You may configure authentication settings for internal clients (that is, those that connect to an internal URL, as described in the "External and internal URLs" section later in this chapter), external clients, and the RPC virtual directory itself.

The Exchange 2013 CAS supports several types of authentication, not all of which are available to on-premises customers; for example, Office 365 allows the use of Microsoft accounts (formerly known as Windows Live IDs and now more properly called Microsoft Online Services IDs) for authentication, but you can't use them in your own deployment. For the purposes of this book, the interesting types of authentication are as follows:

- Basic authentication passes user credentials in cleartext, so it must be used in combination with Secure Sockets Layer/Transport Layer Security (SSL/TLS).
- Kerberos authentication is the type Windows uses natively for client–server authentication. After a user logs on to a domain controller, he receives a Kerberos credential that can be passed to other servers and services to authenticate the user without requiring re-entry of the user's credentials. Kerberos was designed at MIT expressly to allow secure network authentication on untrusted networks, so it doesn't expose usernames or passwords in cleartext. However, it requires clients to be able to connect to a Kerberos key distribution center (KDC), which in Windows means they need to be able to reach a domain controller—something that often won't be possible for Exchange clients that are outside the firewall, not joined to an Active Directory domain, or not running Windows.
- NTLM authentication is the Windows predecessor protocol to Kerberos. Rather than depending on a centralized KDC, NTLM depends on an exchange of an encrypted challenge and response sequence. Unlike Kerberos credentials, an NTLM authentication token is only valid for the server that originally issued it.
- Integrated Windows Authentication (IWA) is what Microsoft calls the Internet Information Services (IIS) setting that enables native Kerberos and NTLM logon. This is the native method IIS uses for client and server authentication. With IWA enabled, servers request and accept Kerberos authentication, but they also accept NTLM authentication from clients that can't use Kerberos.
- Form-based authentication (FBA) is the familiar authentication method Outlook Web App uses; users see a form the Outlook Web App code generates. When they enter their credentials in the form, their browser performs an HTTP POST to an HTTPS URL on the Exchange server, so the credentials are encrypted in transit. The server then uses the credentials to authenticate the user against Active Directory; if authentication succeeds, the server generates an encrypted cookie that it returns to the client, and the client submits that cookie with each subsequent request to prove that it's previously authenticated. Some reverse proxy solutions (notably TMG) can put up their own FBA authentication page to allow preauthenticating Exchange users before they actually are allowed to connect to Exchange.

The distinctions among these authentication types can be confusing, in part because of where they can be applied. For example, to set authentication on the Outlook Anywhere virtual directory with the Set-OutlookAnywhere cmdlet, you have four choices:

- The –ExternalClientAuthenticationMethod parameter enables you to set the authentication method Exchange accepts from clients that connect to the external URL.
- The –InternalClientAuthenticationMethod parameter enables you to set the authentication method that Exchange accepts from clients that connect to the internal URL.

  Note that for both of these parameters, you only get to pick one authentication method for each client type.
- The –IISAuthenticationMethods parameter enables you to set multiple authentication methods that IIS will use. This might seem unnecessary—after all, you can set internal and external client authentication methods, so why would you need a separate way to set the methods that IIS itself will accept? The answer is that if you have a firewall between your external clients and your CAS servers, the firewall might be translating authentication methods. For example, TMG might accept basic authentication from the client and then reauthenticate to the CAS, using IWA, so IIS needs to be configured to accept IWA.
- The –DefaultAuthenticationMethod parameter applies the authentication type you specify to the ExternalClientAuthenticationMethod, InternalClientAuthenticationMethod, and IISAuthenticationMethods parameters—so you can use this switch alone to set all the authentication properties in one go.

If you configure basic authentication for Outlook Anywhere on a server, IIS only enables basic authentication on the /rpc virtual directory. To accept proxy requests from Exchange 2013, the /rpc virtual directory needs to accept Integrated Windows Authentication (IWA, previously known as NTLM) connections; otherwise, Kerberos won't work. However, if you just modify this setting directly in IIS, Exchange will overwrite it. Instead of changing the IIS settings directly, you need to use Set-OutlookAnywhere to change the settings on all your earlier CAS servers so that internal (between CAS) connections are authenticated with Kerberos while external (client) connections continue to use basic authentication. To do this, use Set-OutlookAnywhere like this:

```
Set-OutlookAnywhere –Server HSVCAS02 –ClientAuthenticationMethod Basic
–IISAuthenticationMethods Basic, NTLM
```

Not only do you have to pick the right authentication type, you must also choose wisely where you apply the authentication type you want to use! An Exchange 2013 CAS role has separate settings for external and internal authentication.

External vs. internal

When you consider what the CAS role does, it makes perfect sense to distinguish between two sources of client connections: internal clients on the organization's network and external clients that connect through a firewall by using one of the supported and enabled client protocols. Some organizations want to allow both internal and external clients to connect without restriction, whereas others limit external connections. Most organizations that deploy Exchange also want the flexibility to configure CAS behavior independently for internal and external connections.

The primary objects of interest for the external-versus-internal split are the URLs to which clients connect and the authentication methods that clients may use. To see the settings in effect on a CAS, you have to use a variety of cmdlets because each protocol has its own settings for these objects. For example, to see the external and internal configuration settings for Outlook Anywhere on a CAS named PAO-EX01, you could do the following:

```
Get-OutlookAnywhere -Server PAO-EX02 | fl -property *ternal*


 ExternalHostname                    :
 InternalHostname                    : pao-ex02.betabasement.com
 ExternalClientAuthenticationMethod : Negotiate
 InternalClientAuthenticationMethod : Ntlm
 ExternalClientsRequireSsl           : False
 InternalClientsRequireSsl           : False
```

These settings, which are unchanged from the installation defaults, show that external and internal clients use different authentication methods and that the external hostname isn't set. Compare those results to the output of Get-WebServicesVirtualDirectory, which has been edited to remove extraneous items:

```
Get-WebServicesVirtualDirectory -Server PAO-EX01 | fl -property *ternal*
```

```
InternalNLBBypassUrl           :
 InternalAuthenticationMethods : {Ntlm, WindowsIntegrated, WSSecurity, OAuth}
 ExternalAuthenticationMethods : {Ntlm, WindowsIntegrated, WSSecurity, OAuth}
 InternalUrl                   : https://pao-ex01.betabasement.com/EWS/Exchange.asmx
 ExternalUrl                    https://mail.betabasement.com/EWS/Exchange.asmx
```

Here the external and internal URLs have different values, and the virtual directory object has a set of authentication methods defined instead of a single authentication method. Why the differences?

External and internal URLs

Autodiscover publishes the external and internal URLs in the Outlook Anywhere settings, and on the Outlook Web App, EWS, EAS, and other virtual directories, to clients. It's up to the client to decide which of those two URLs to use. Of course, if one of them is blank, that makes the client's decision very easy. Presuming that both URLs are set, and their values are different, the client must decide based on its own knowledge of its network location or just by trying first the internal URL and then the external URL if the first attempt fails.

Exchange 2013 sets the internal URL for these services by default to the name of the server plus the path to the virtual directory (for instance, *https://pao-ex01.betabasement.com /EWS/Exchange.asmx* for EWS). The external URL is blank by default; you must set it yourself for each of the services you want to be externally accessible.

TROUBLESHOOTING

EWS clients report that they can't communicate with Exchange

Service Pack 2 of Outlook 2011 for Mac OS X seems to have an odd bug. Suppose that you have Outlook 2011 configured on an Apple laptop to talk to an Exchange 2013 server. You use the client while connected to the internal network, so when Autodiscover provides URLs, Outlook correctly detects and uses the internal URL. You shut down the laptop and take it to the local coffee shop, open it, and connect to that network. Outlook detects the change in network configuration, performs a new Autodiscover operation, and then ignores the external URL, so you don't reconnect to the Exchange server and don't get any new mail. This doesn't happen consistently, but it is a longstanding bug.

The problem is that this behavior is identical to what you'd see if the external URL were blank or set incorrectly: the client performs an Autodiscover, tries the internal URL, and then tries the external URL when the internal URL is unreachable. If the external URL is unreachable too, the client can't connect. The same thing can happen with the Lync desktop client for Windows or Mac OS X; it depends on EWS access to read free/busy, calendar, and contact data, so if your external URL is set incorrectly, Lync might report that it's having problems connecting to Exchange.

A default installation of Exchange 2013 leaves the ExternalUrl property of the EWS virtual directory blank. You need to set it to the correct value for external client access on all your Internet-facing CAS servers, or you'll experience these problems. The "Designing namespaces" section later in the chapter covers this topic in more depth.

External and internal authentication

Along with the URL connection points that you specify for internal and external client access, Exchange maintains separate settings for which authentication methods each client endpoint supports. In general, you should leave the default authentication settings for the CAS virtual directories alone; changing them without a good reason is a great way to break your Exchange deployment in interesting and subtle ways. One common symptom of incorrect authentication settings is a stream of repeated authentication prompts in Outlook or Outlook Web App.

> **Note**
>
> **In Exchange 2013 RTM and Exchange 2013 CU1, it is possible for users to see multiple authentication prompts when their connections are redirected from one Internet-facing CAS to another, but in Exchange 2013 CU2 and later, the authentication credentials for Outlook Web App connections are re-sent, too, so that users only have to sign in once.**

There are a few exceptions to this guideline, most of which involve coexistence with older versions of Exchange. For example, when you deploy an Exchange 2013 CAS into an existing Exchange 2007 or Exchange 2010 organization, you should configure things so that all CAS traffic goes to the Exchange 2013 CAS first because it can proxy or redirect traffic to earlier versions as necessary. For this to work, you must ensure that all your CAS servers, of any version, are configured to use basic authentication for the client and NTLM authentication on the virtual directory. Basic authentication is the lingua franca of all the HTTP-based protocols, so it will always need to be enabled for EWS, EAS, and Autodiscover.

> **Note**
>
> **Microsoft has prepared a master list of all the default permissions and URLs on virtual directories for the Exchange 2013 CAS role. This list is extremely useful as a reference in case you accidentally change something and end up with undesirable side effects. It's available at *http://technet.microsoft.com/en-us/library/gg247612(v=exchg.150).aspx*.**

Managing virtual directory settings

A default installation of a multirole Exchange 2013 server leaves you with seven virtual directories: Autodiscover, ECP, EWS, EAS, OAB, Outlook Web App, and Windows PowerShell. You can see and change the authentication and URL settings on these virtual directories in two ways. First, you can use the appropriate Set cmdlet (for example,

Set-EcpVirtualDirectory, Set-OabVirtualDirectory); second, you can use EAC. To use EAC, switch to the Servers tab and then choose the Virtual Directories tab, as shown in Figure 1-3. Opening the properties of a virtual directory by double-clicking it, or selecting it and clicking the pencil icon, produces a dialog box like the one shown in Figure 1-4.

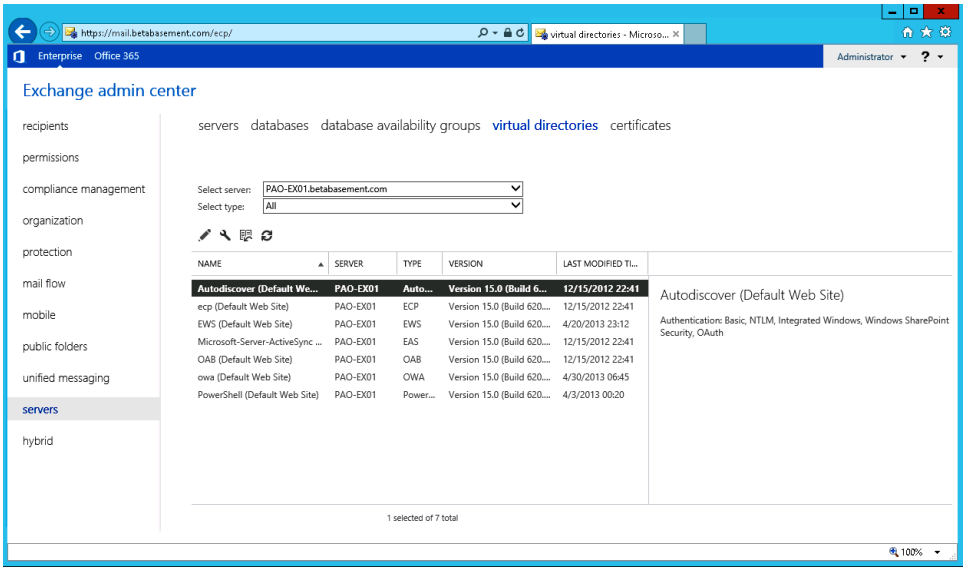


Figure 1-3 Editing virtual directory settings in EAC

Each virtual directory might have its own specific settings; for example, the EWS virtual directory has a check box for controlling whether the MRS proxy endpoint is enabled, as shown in Figure 1-4, whereas the Exchange Control Panel (ECP) virtual directory allows you to enable FBA, a setting not present on other virtual directories except /owa.

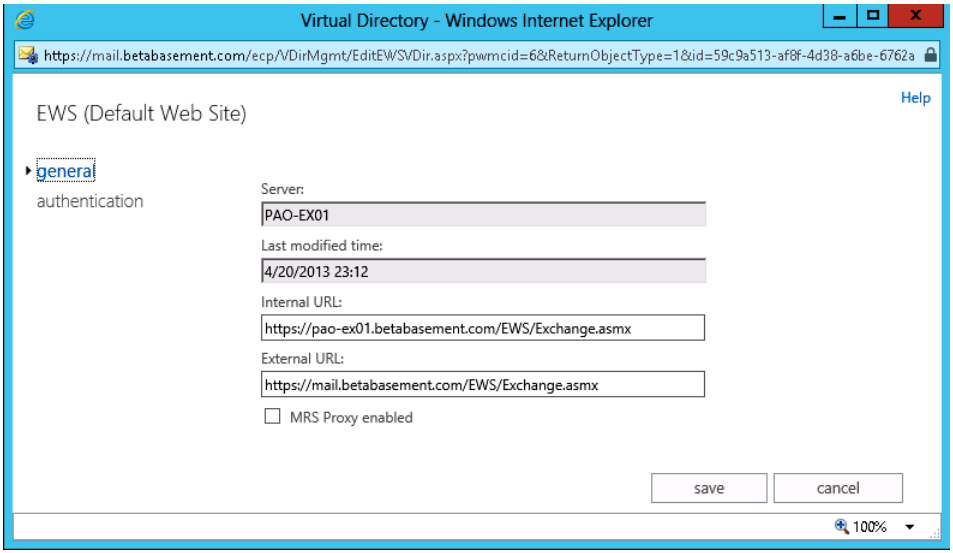


Figure 1-4 The settings for the default EWS virtual directory, which include an internal and external URL

The death of affinity

Another very important side effect of the decoupling of the CAS and Mailbox roles is a sharp reduction in the need for *client affinity*. Affinity means that a session for a given client remains with the same CAS for the duration of the connection. In other words, after a client connects to a particular CAS, the same CAS would continue to service the connection until the connection is terminated. To do this, the CAS must maintain session state so that any load balancers in the path know which CAS handles which connection. Sessions or protocols in which client affinity is maintained are said to be sticky. If a load balancer or reverse proxy solution doesn't support affinity, it will break some Exchange 2007 and Exchange 2010 Exchange services or reduce the performance of others by forcing users to reauthenticate each time the session moves to a different CAS, so this is clearly an area that you need to get right when deploying those versions of Exchange.

Exchange 2013 does away with the requirement for session affinity at the load balancer level; the load balancer doesn't have to maintain client affinity because any client can connect to any CAS and be proxied to the Mailbox server with the active copy of the mailbox database. This holds true for every protocol, so the protocols (such as Outlook Web App) that required sticky affinity in prior versions no longer have that restriction.

I mentioned that a failure of affinity would result in the client being asked to reauthenticate. Exchange 2013 does away with this circumstance by changing the way it uses cookies

to cache authentication. In Exchange 2010, whatever CAS a client connects to provides an encrypted cookie after the client is authenticated. Passing that cookie to a different CAS means that the receiving CAS can't read it, so it has to ask the client to authenticate again. In Exchange 2013, the CAS authentication cookie is encrypted with the public key of the certificate assigned to the CAS, so any server that has the corresponding private key can decrypt the cookie. If you follow the Microsoft recommendation of putting the same certificate on all your CAS servers (more on that in the "Certificate management" section later in this chapter), then any CAS can decrypt the cookie.

Load balancing made simpler

If you've ever had to pass any Microsoft certification exams involving networking, you probably remember the Open Systems Interconnect (OSI) model, with its seven layers (physical, data link, network, transport, session, presentation, and application, more easily remembered with the mnemonic, Please Do Not Throw Sausage Pizza Away). The TCP/IP internetworking model doesn't correspond exactly to the OSI model, but at least for the first four layers, it's fairly close. For discussing CAS load balancing, you're most interested in layers 4 and 7.

Layer 4 load balancing

At layer 4 (L4), the network layer, a load balancing device only has information about the Internet provider (IP) address and port to which the client is connecting. All the other potentially useful information, such as the URL it has requested or any cookies or query parameters associated with its request, is encoded in IP or User Datagram Protocol (UDP) datagrams and has been encapsulated. L4 devices don't de-encapsulate those datagrams to read the payload, so an L4 load balancer has to make decisions about where to route traffic based solely on the requested destination. L4 load balancers are therefore less complex to design, maintain, and support—they don't know or care what's happening at the application layer.

Layer 7 load balancing

Layer 7 (L7) load balancers are often labeled application-aware or smart because they can see the application-layer traffic, including the specific URLs clients request and any cookie or session information the client passes as part of its request. Consider the case when a load balancer sees a request from a mobile device for a URL such as *https://nacltl14se01.contoso.com/Microsoft-Server-ActiveSync?jQAJBBCz0DFoa3Zf/Y1CsFFhMg2bBErZMzwCV1A=HTTP/1.1*. An L4 load balancer sees only two pieces of information: the name of the requested server (naclt14se01.contoso.com) and the requested protocol (HTTPS, meaning TCP port 443). After the connection passes through the load balancer, the CAS sorts out the connection and gets it to the right place, but that's too late to have any notion of protocol

awareness. By contrast, an L7 load balancer sees the same information, but it also knows the specific URL being requested, and it can see the synchronization key. In fact, the L7 load balancer can terminate the incoming SSL connection and then inspect the packets and discover the protocol to which the connection is directed.

Exchange offers a rich set of protocols and services, including Exchange Web Services (EWS), Outlook Web App, Exchange ActiveSync (EAS), the Offline Address Book (OAB), and the Exchange Administration Center (EAC), each of whose endpoint is represented as an IIS virtual directory. The problem is that a target CAS might be inoperative—or, worse, only one of its supported protocols might have failed. The new Managed Availability capability built into Exchange 2013 attempts to resolve problems, such as a failed protocol, automatically after it detects an issue by noting that client connections are failing. In this instance, the resolution might be to recycle an application pool or even to restart a server. However, an L4 load balancer sees a CAS in the whole rather than distinguishing among the different protocols the CAS might be able to handle, some of which are healthy and some of which might not be. With L7, the load balancer would be aware that Outlook Web App is up, but EAS is down on a specific target CAS, and be able to take action to redirect traffic as individual protocols changed their status.

Given that a load balancer's job is to distribute traffic among all the servers in its array, you might wonder whether the extra knowledge an L7 load balancer has available makes a difference. The answer is an unequivocal yes. An L4 load balancer can determine whether the target IP address is up or down by pinging it—and that's about all it can do. Some L4 load balancers can perform HTTP health checks too, but the checks are limited to a single virtual IP address. A server that is turned on, plugged in to the network, and not running Exchange services at all will give the same response to a basic L4 load balancer as one that has the Exchange 2013 CAS role on it. When a service or protocol handler fails or slows down, an L4 load balancer can't tell. This leads to the possibility that the load balancer will continue to direct incoming traffic to a machine that cannot properly handle it; of course, you could argue that Managed Availability (covered in depth in *Microsoft Exchange Server 2013 Inside Out: Mailbox and High Availability*) will restore service automatically so that the load balancer won't run into this situation. However, an L7 load balancer can tell not only whether the network layer of the server is responsive but also whether the actual protocol or service the client is asking for—Outlook Web App, EAS, Outlook Anywhere, and so on—is performing properly, and it can route traffic accordingly. In addition, many L7 load balancers, such as those sold by F5 and Kemp Technologies, can perform health checks on individual services and applications so they know when an EAS endpoint, IMAP server, or whatever becomes unavailable or returns to normal service. The Managed Availability service includes a health check URL for each monitored service that any load balancer (or other monitoring tool) can use; querying the service virtual directory with /healthcheck.htm appended (for example, /owa/healthcheck.htm) will return an HTTP 200 response if the service is healthy.

DNS round robin

Strictly speaking, DNS round robin is a load balancing technology, too, although it probably doesn't seem that way at first glance. When you configure multiple IP addresses for a single Domain Name System (DNS) name, the DNS server returns all those addresses to any client that requests them, changing the order in which they appear each time. Two clients that ask a DNS server to resolve the same DNS name will get the same addresses but in a different order. Because most clients just use the first IP address the DNS server returns, this provides a simplistic means of spreading client load across multiple machines. Microsoft makes a point of saying that what it calls modern HTTP clients (including Outlook 2010 and Outlook 2013 and some compliant Exchange ActiveSync implementations) are smart enough to try each of the IP addresses they receive when they get a round robin response. It doesn't matter whether any particular client gets different CAS servers when it makes two sequential requests because the Exchange 2013 CAS is now stateless. Unfortunately, they don't tell you which clients are considered modern. For example, Safari 6.x for Mac OS X implements this behavior, whereas Safari for Windows doesn't. Although Microsoft officially supports this address resolution behavior, I don't recommend that you rely on it unless you can ensure that all your clients support it.

Even without this implementation-dependent behavior, round robin DNS has a critical flaw: it has absolutely no awareness of the state of the machines or services whose IP addresses it is handing out. If the DNS server is configured to resolve mail.contoso.com to 192.168.0.200, 192.168.0.201, and 192.168.0.202, and the machine at 192.168.0.201 is down, the DNS server will continue handing out that address as the first entry to one-third of the clients that contact it. In addition, because clients or intermediate servers might cache DNS results, changes you make manually might not be immediately visible to clients. The counterbalance for these flaws is that every modern DNS server implementation supports round robin DNS, so implementing it is easy and inexpensive. It is most often used to provide load balancing for SMTP because SMTP is stateless by nature and, thus, it doesn't matter which server two sequential clients connect to.

Windows Network Load Balancing

Microsoft began offering Windows Network Load Balancing (WNLB) in Windows 2000. Since then, it has steadily upgraded WNLB functionality, and yet relatively few Exchange sites use it, even though it's included with Windows. There are two reasons for this. First is that WNLB is unintelligent; that is, it lacks service awareness because it does not check ports and services on a server before considering it a suitable candidate for load balancing. Essentially, if a server has a pulse, NLB thinks it is good. In addition, there is no communication between Exchange and NLB, and Exchange does not attempt to balance client connections across all the CAS servers in the array.

More significantly, you cannot use WNLB with servers that are members of a Database Availability Group (DAG). Windows Failover Clustering (WFC) is incompatible with WNLB. If you separate the CAS and Mailbox roles, you could use WNLB to load balance traffic across your CAS servers, but if you want to use multirole servers, as Microsoft recommends, those servers can *either* be load balanced with WNLB *or* be DAG members. Because DAGs offer such a powerful solution for high availability, WNLB just isn't widely deployed with Exchange. Round robin DNS gives you essentially the same load balancing capability with Exchange 2013 without having to worry about separating the CAS and Mailbox roles.

Choosing a load balancing solution

Given the choice among WNLB, DNS round robin, an L4 load balancer, and an L7 load balancer, which should you choose? First, it's important to understand that no matter which solution you choose, the basic behavior will be the same:

- If you're using a load balancer or Windows NLB, clients see a single virtual IP address and FQDN for the entire set of load balanced servers. If you're using round robin DNS, each server in the array needs to have its own resolvable IP address to which clients can connect.

- The number of servers in the load balancing array is determined by the load balancer. For example, WNLB arrays are limited to 32 servers (in Windows Server 2012); other manufacturers have different limits.

- Incoming client connections arrive at the load balancer, and it determines where to send them. However, L4 and L7 load balancers make this decision using different information. L4 looks only at the destination IP and port, whereas L7 needs to look at the connection contents. This necessarily implies that an L7 load balancer must be able to terminate SSL connections so it can inspect the contents and refer connections to the appropriate server.

With those factors in mind, your next decision is the balance between functionality and cost. L4 load balancers are quite inexpensive, but they offer limited functionality compared to L7 solutions. However, one way to work around this difference in functionality is to use one namespace per protocol as described in the "Designing namespaces" section later in this chapter. By doing so, the L4 load balancer can distinguish among requests for different services because each service or protocol has a unique FQDN and virtual IP. However, L7 load balancers offer a much wider range of functionality, most significantly the ability to monitor the health and performance of individual services. Those additional capabilities demand a price premium, which many customers are willing to pay. Exchange itself is agnostic on the question because it is not integrated with or aware of either L4 or L7 load balancing.

A related question is whether to deploy physical or virtualized load balancing appliances. As software-defined networking (SDN) becomes more common, this will become a question of increasing interest; at least for now, very few organizations have deployed virtualized switches or routers, with the exception of switches integrated with hypervisors. Virtualizing your load balancers offers the same benefits, and drawbacks, as virtualizing any other part of your Exchange infrastructure. I tend to prefer physical load balancers because they don't impose the additional overhead and complexity of a virtual machine (VM) host; however, for organizations with smaller networks, or in which VM host space is abundant and well managed, virtualized load balancers offer no essential differences in functionality from their physical counterparts and might be an appropriate choice.

The role of Outlook Anywhere

It's important to understand that Outlook Anywhere is now *the* protocol by which Outlook connects to Exchange 2013. Whereas earlier versions would allow RPC connections directly to the Mailbox server, using plain TCP, Exchange 2013 uses RPC over HTTPS everywhere all the time. That means that one of the first things you should do now, before you even consider adding Exchange 2013 to your existing Exchange 2007 or Exchange 2010 deployment, is enable Outlook Anywhere on all your servers and verify that it works. This requirement holds true for *every* CAS server, even those that are not Internet-facing.

Although this requirement might seem odd, there's a good reason for it. The Exchange 2013 CAS role server includes a new proxy engine, httpproxy.dll. This replaces the role of the old rpcproxy.dll, and Exchange 2013 CAS thus cannot proxy RPC traffic directly. When it receives HTTPS encapsulated RPC traffic, it cannot de-encapsulate it directly; instead, it must proxy it to another CAS server that still has rpcproxy.dll. For this proxy operation to succeed, the Exchange 2013 mailbox server or the downlevel CAS servers must have rpcproxy.dll installed (it's installed by default on the Exchange 2013 mailbox role), and they must be enabled for Outlook Anywhere. Luckily, enabling Outlook Anywhere is simple; on an Exchange 2007 or Exchange 2010 CAS, you can use the Enable-OutlookAnywhere cmdlet, like this:

```
Enable-OutlookAnywhere -Server 'PAO-EX03' -ExternalHostname 'pao.contoso.com'
-ClientAuthenticationMethod Basic -SSLOffloading $false –IISAuthenticationMethods
Basic, NTLM
```

When you specify the –ClientAuthenticationMethod switch but not –IISAuthenticationMethods, the IIS authentication methods are set to NTLM plus basic. This is exactly what you want for a mixed organization—remember, you need every CAS in the organization to accept Outlook Anywhere traffic.

If you prefer, you can use the Exchange 2010 wizard for configuring Outlook Anywhere on your Exchange 2010 servers, the wizard will lead you through specifying external and internal URLs and authentication methods (see Figure 1-5).

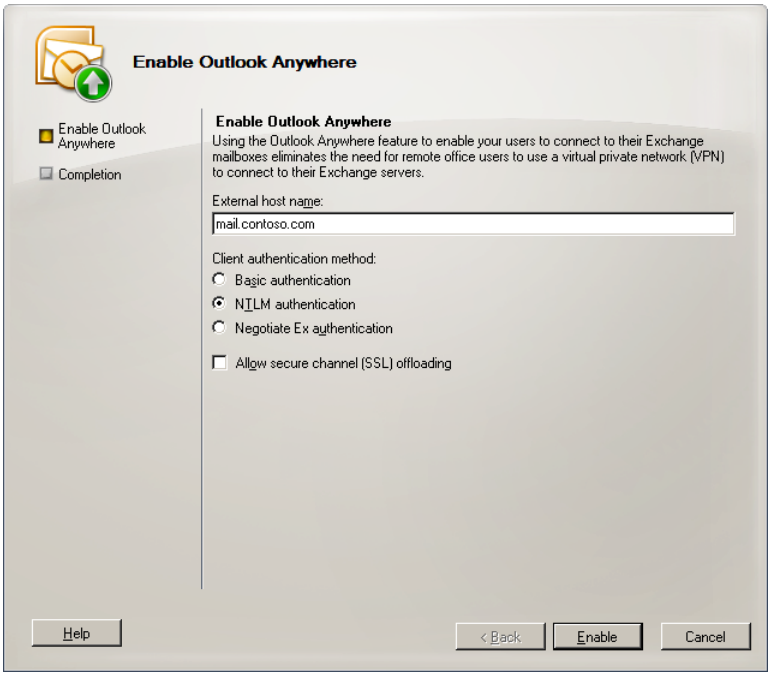


Figure 1-5 The Exchange 2010 Enable Outlook Anywhere Wizard, available from Exchange Management Console (EMC)

By default, Exchange 2013 is already configured to use Outlook Anywhere; the only task required on your part is to install a valid certificate for Outlook Anywhere to use on the CAS. The Outlook client will complain if it receives a self-signed certificate from the server unless you manually add it to the trusted certificate authority (CA) store on the client computer, which is a hassle. Instead, you should plan to install a certificate from either an internal CA in your organization or a trusted external CA on your CAS. (Certificate configuration is discussed later in the chapter, in the "Certificate management" section.) Interestingly, the Mailbox server can use a self-signed certificate for Outlook Anywhere communications because the CAS ignores those certificates. (More precisely, it knows the connection has been authenticated with Kerberos, so it only cares whether the certificate can be used for encryption.)

Designing namespaces

You will often hear the term "namespace" bandied about when discussions turn toward the CAS role. In this context, "namespace" is essentially a fancy way of saying "URL" or "FQDN." The namespace used for a particular protocol is whatever URL or FQDN clients running that protocol connect to. ("Namespace" has different and much more complex connotations when used to refer to XML documents, C# or C++ programs, and other fields.) For example, an Exchange 2010 RPC client access array might have a namespace of *mail.contoso.com*, but the namespace for Outlook Web App in the same environment might be *owa.contoso.com*. The process of designing namespaces for Exchange 2007 and Exchange 2010 is complex and error-prone, and Microsoft has greatly simplified it for Exchange 2013, but it is still not exactly what I'd call simple.

Using a single namespace

The simplest possible design is to use a single namespace (such as mail.contoso.com) for all services except Autodiscover. This might require you to use split DNS or other methods of DNS manipulation to ensure that internal and external clients get the correct address for this namespace, but it is difficult to argue against the resulting simplicity.

One name per service?

One way of achieving selective control is to publish specific connectivity points for each protocol as part of your external namespace. Therefore, instead of having the catch-all *mail.contoso.com*, you'd have a set of endpoints such as eas.contoso.com, ecp.contoso.com, owa.contoso.com, and so on. The advantage is that the L4 load balancer now sees protocol-specific inbound connections that it can handle with separate virtual IPs (VIPs). The load balancer can also monitor the health of the different services that it attaches to the VIPs and make sure that each protocol is handled as effectively as possible. The disadvantage is that you have more complexity in the namespace, particularly in terms of communication to users, and you have to either ensure that the different endpoints all feature as alternate names on the SSL certificates that are used to secure connections or buy a more expensive wildcard certificate. None of this is difficult, but it's different from before. What you gain from the transition from L7 to L4 you lose (a little) on extra work.

To accomplish this, you need to use either EAC to set the external URL for each virtual directory on each server or the appropriate Set-*VirtualDirectory cmdlet on the appropriate servers. The simplest way to do this is something like the following:

```
Get-ActiveSyncVirtualDirectory | Set-ActiveSyncVirtualDirectory –ExternalURL
activesync.contoso.com
```

Of course, this will require you to ensure that you have correctly registered your servers in DNS and that the certificates on the servers are configured with the correct subject and subjectAlternativeName fields.

Using a single internal name for Outlook Anywhere

Every Exchange server has its own unique Windows computer account name, but Exchange is perfectly happy to ignore that name in favor of a name you assign. This can be a source of both great utility and great aggravation, depending on whether you configure it properly. Note that the server name Exchange presents isn't the same as the internal or external URL. On a default Exchange installation, if you check the external hostnames defined for Outlook Anywhere, they'll be blank, like this:

```
Get-ClientAccessServer | Get-OutlookAnywhere | select identity, *hostname
```

```
Identity              ExternalHostname          InternalHostname
--------              ----------------          ----------------
PAO-EX01\Rpc          (Default Web Site)        pao-ex01.betabasement.com
PAO-EX02\Rpc          (Default Web Site)        pao-ex02.betabasement.com
```

This obviously makes it hard for external clients to use Outlook Anywhere, which is addressed in the next section. However, note that each of the two servers has its server FQDNs listed in the InternalHostname field. It would simplify both load balancing and failover operation if both those servers had the same internal hostname; in that case, the load balancer *or* a client could just resolve the internal hostname (in this case, mail.betabasement.com) and carry on with its work.

```
Get-OutlookAnywhere | Set-OutlookAnywhere -InternalHostname mail.betabasement.com
-InternalClientsRequireSsl $true
```

The –InternalClientsRequireSsl flag is required; you don't have to set it to $true, but if you do not, your internal clients won't attempt to encrypt their Outlook Anywhere traffic, including credentials, so this is pretty much a mandatory setting.

External names for Outlook Anywhere

Choosing an external name for Outlook Anywhere is slightly trickier. The name you choose has to be externally resolvable; for that reason, the consensus seems to be that you should choose a name different from the internal hostname. A common design pattern is to use "outlook" as the internal hostname and "mail" as the external hostname, so you'd see something like this when performing a Get-OutlookAnywhere:

```
Get-OutlookAnywhere | select identity, *hostname
```

```
Identity                          ExternalHostname       InternalHostname
--------                          ----------------       ----------------
PAO-EX01\Rpc (Default Web Site)   mail.betabasement.com  outlook.betabasement.com
PAO-EX02\Rpc (Default Web Site)   mail.betabasement.com  outlook.betabasement.com
```

In Exchange 2010, if the internal and external hostnames are both externally resolvable, but the internal hostname isn't actually reachable from the Internet, Outlook clients can try to connect to the internal hostname first. They'll fail, of course, but this adds an unwanted startup delay. Exchange 2013 doesn't have that problem, but many still consider it a best practice to have the internal hostname both unreachable and unresolvable from the external world. However, you cannot do this if you're using a single namespace with split DNS as described in the "Using a single namespace" section earlier in this chapter.

The Front End Transport service

The Front End Transport (FET) service is discussed in much more depth in Chapter 2, "The Exchange transport system." However, it can be explained very simply. FET is a service that accepts SMTP connections and redirects them to Mailbox servers. It doesn't queue mail for delivery, meaning that it doesn't have to keep a queue database, participate in the Exchange 2013 shadow redundancy or Safety Net features, or do anything other than authenticate recipients (if configured to do so) and pass SMTP traffic to other servers. Because FET doesn't queue anything, if it cannot immediately reach a proxy target, the FET service will return a transient SMTP error to the sender, forcing it to try again later. This is a neat trick because it shifts the burden of ensuring reliable message delivery away from the CAS and back to the sending server, whatever SMTP software it's running.

One consequence of the new FET behavior becomes evident when you have a single server on which both the Mailbox and CAS roles are installed. The transport services that belong to the Mailbox role don't receive the message directly—instead, inbound messages on that machine arrive at the FET service, which then proxies them to the Mailbox Transport service. If you look at the headers for an inbound message, you'll see more hops than you might otherwise expect. Figure 1-6 shows the output from the free Message Header Analyzer app installed on an Exchange 2013 server. (This free app and other Office apps for Outlook and Outlook Web App are described in more detail in Chapter 3, "Client management.") Note that three hops are shown for the message: it is received by the FET service on PAO-EX01, sent to the Mailbox Transport service on PAO-EX01, and then sent to the Mailbox Submission service on PAO-EX01.

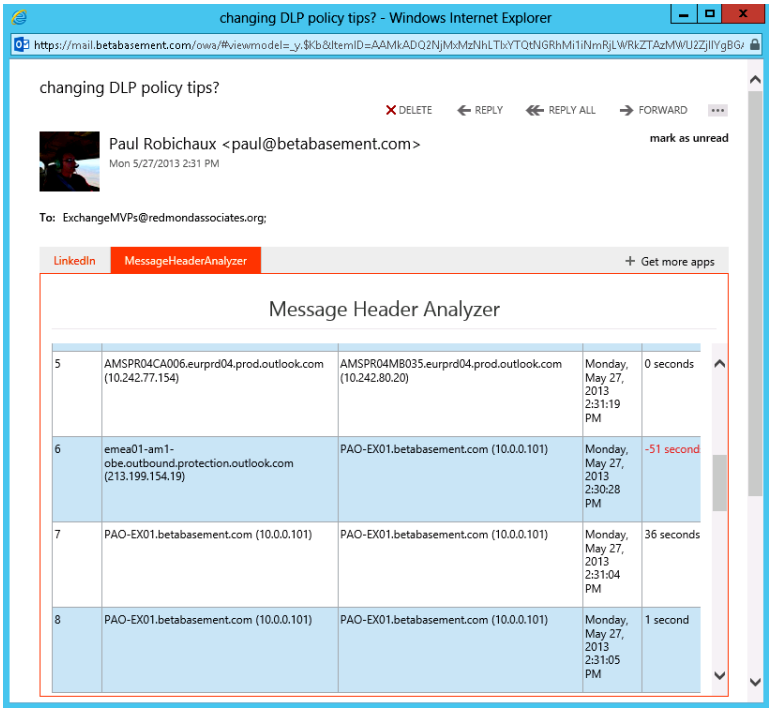


Figure 1-6 The Message Header Analyzer app running in Outlook Web App, showing the message headers for a message that a multirole Exchange 2013 server received

Autodiscover

Microsoft introduced the Autodiscover service in Exchange 2007 as a solution for the perennial problem of how to help users configure Outlook and Exchange ActiveSync with the name of their mailbox and the server that hosts the mailbox the first time they connect to Exchange. Information about the mailbox's location is subsequently held in the Outlook profile. A profile can still be configured manually, but it's a lot easier to let Autodiscover do the work for you, especially because Outlook overwrites manually configured settings that conflict with settings it gets from Autodiscover. This all works very nicely unless the mailbox for which you are attempting to configure access is hidden from Exchange address lists, in which case, Autodiscover won't be able to find the mailbox, and you'll have to configure the profile manually. Outlook can also requery Autodiscover if it loses connection to the mailbox.

Outlook 2007, Outlook 2010, Outlook 2011 for Mac OS X, and Outlook 2013 support Autodiscover, as do Outlook RT for Windows RT and the built-in mail clients for Windows Phone and Apple iOS. Each of these clients performs an Autodiscover request at start-up,

as a result of which they receive an Autodiscover manifest that contains a wealth of useful information. The Exchange ActiveSync and regular Autodiscover manifests are different; the regular XML manifest is several hundred lines long, but you can see an excerpt in Figure 1-7, which shows the output from the Outlook 2013 Test E-Mail AutoConfiguration dialog box. To access this dialog box yourself, hold down the Ctrl key and click the Outlook icon in the system tray; you'll see the Test E-mail AutoConfiguration command in the context menu. After filling in the username and password fields, a successful test shows results like those in the figure. In this case, you can see that Autodiscover has returned URLs for the Availability service, the out of office (OOF) service, the OAB download endpoint, the Unified Messaging service, and various parts of the Exchange Administration Center (EAC) that are used within Outlook (although they are still labeled Exchange Control Panel in Outlook 2013).

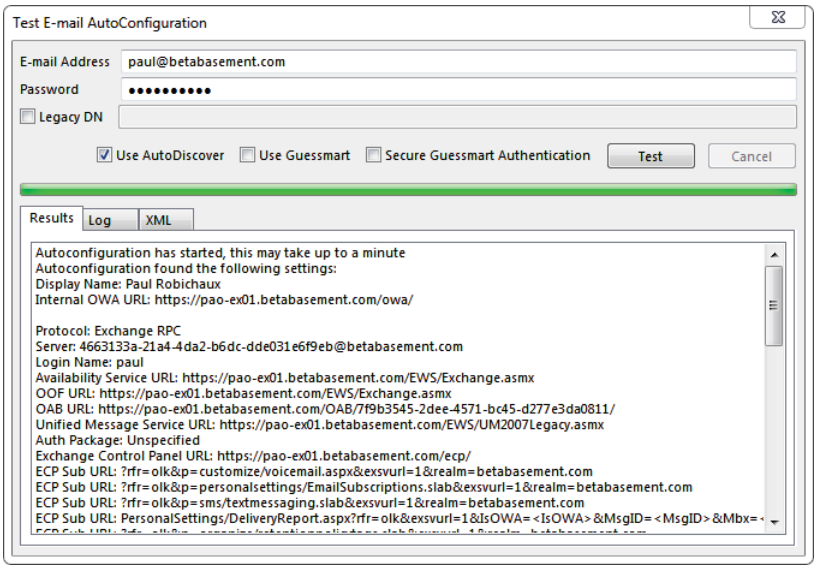


Figure 1-7 The Test E-mail AutoConfiguration dialog box in Outlook 2013, showing a portion of the data returned from a successful Autodiscover request

One of the most important pieces of information Autodiscover returns is which server contains the user's mailbox. In Figure 1-7, because Autodiscover was run against a mailbox on an Exchange 2013 server, the returned endpoint is actually a GUID; the Server line in the returned results is 4663133a-21a4-4da2-b6dc-dde031e6f9eb@betabasement.com rather than a more conventional FQDN. The mailbox might move without warning because of a failover, a switchover, or a mailbox move; thanks to the mechanism described earlier, Outlook doesn't have to start a new connection because it only communicates with the CAS. However, Outlook for Windows will repeat Autodiscover every 60 minutes or after losing connectivity to the mailbox. Other Autodiscover clients generally have similar

behavior, although the intervals might vary. Besides information about the primary mailbox, Autodiscover also returns configuration data about public folder mailboxes, shared mailboxes, and site mailboxes if they're in use.

The Autodiscover process

Autodiscover clients follow a predictable process to make requests. Understanding this process is valuable because knowing the steps in the process eliminates any mystery from discussions of why a particular client got the results it did.

The basic Autodiscover process works like this:

1. The client provides credentials, normally in the form of an SMTP email address and a password.

2. If the client is domain joined, it queries Active Directory for a list of service connection point (SCP) objects, the nature and use of which are described in the "Accessing Autodiscover through SCPs" section later in this chapter. If that query succeeds, the client will have a URL against which to try Autodiscover.

3. If the SCP query fails, the client tries to perform HTTPS POST operations against a sequence of well-known URLs. This often happens because the client either isn't domain joined or cannot reach a global catalog server to look for SCPs. Examples include mobile devices, Outlook for Windows running on home PCs, and Outlook for Mac OS X. The process for constructing these well-known URLs requires the client to treat the right side of the email address that the user provided in step 1 as a server name to which other items are added. For example, if the user enters carrie@contoso.com as her email address, her client first uses *https://contoso.com/autodiscover/autodiscover.xml* and then tries *https://autodiscover.contoso.com/autodiscover/autodiscover.xml*.

4. If the initial connection attempt to the well-known URLs fails, the client makes an unencrypted HTTP GET request against *http://autodiscover.contoso.com/autodiscover/autodiscover.xml*, expecting to receive an HTTP 302 redirect result that points to the correct Autodiscover URL. This mechanism is useful when you have a public website that uses split-brain DNS to remain separate from your internal network. Outlook will, and other clients might, generate a warning to tell the user that the server is redirecting the client to another address.

5. If the client still can't get a valid Autodiscover URL, it can query for a DNS SRV record named _autodiscover.tcp.domain. The underscore at the beginning of the record is mandatory. This record should return the FQDN of the real Autodiscover endpoint. If the query succeeds, the client can use the returned name to create a URL by appending /autodiscover/autodiscover.xml to it and attempting an HTTPS POST.

6. As a last-ditch effort, the client will look for a local XML configuration file if you've set HKEY_CURRENT_USER\Software\Microsoft\Office*X.0*\Outlook\Autodiscover to point to the path of a properly formatted Autodiscover manifest. (In this case, the *X* in the registry key refers to the Office version: 14 is Office 2010, and 15 is Office 2013.)

The Autodiscover client might receive a number of responses. In the best case, it receives an actual Autodiscover manifest. It might also receive an HTTP redirect, using the 302 result code. The server might also require the client to authenticate by returning an HTTP 401 or 403 error code, and, of course, the client might receive an HTTP error 404 if it tries to request Autodiscover data from a server that isn't an Autodiscover endpoint.

If you want to see what URLs will be returned to an Autodiscover client, you can use Get-ClientAccessServer –AutoDiscoverServiceInternalURL.

Accessing Autodiscover through SCPs

When you install a new CAS server, it registers an object known as a *service connection point* (SCP) in Active Directory. The SCP is a pointer that associates one or more service endpoints (in the form of a URL or FQDN) with a particular service and server. SCPs are used for a variety of other objects, and application developers can even register their own services by using SCPs. By querying Active Directory, software can get a list of all the end-points that offer a particular service. Figure 1-8 shows some of the properties associated with the SCP registered when the Exchange 2013 CAS named PAO-EX01 was installed. The serviceBindingInformation attribute for an Exchange SCP contains a URL based on the FQDN of the server, and the related keywords attribute on the object lists the site to which the CAS server belongs. When a domain-joined client can reach a global catlog (GC), it will bind to the GC by using the user-provided credentials and query for SCPs.

An SCP can act as a referral from one forest to another to help clients to find Autodiscover information when Exchange is deployed in a multi-forest scenario. The URLs returned from the SCPs essentially act as pointers to CASs to which clients can connect to be redirected to whatever Mailbox server currently hosts their mailbox.

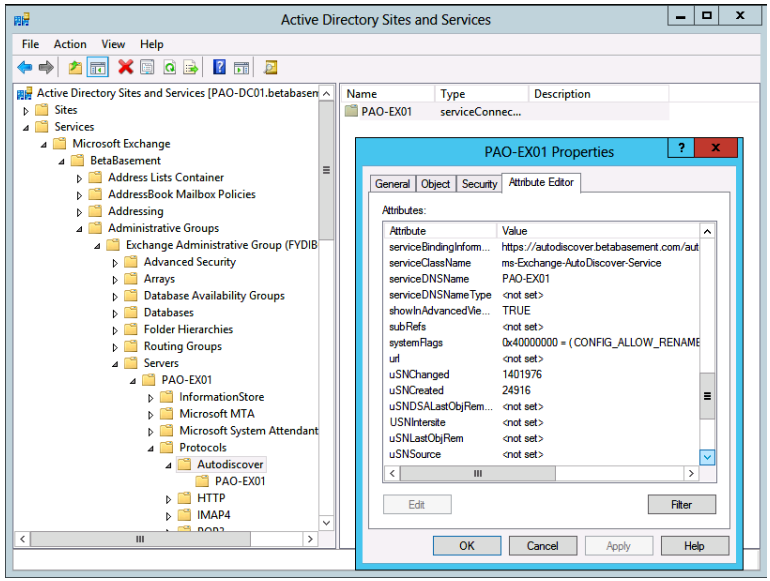


Figure 1-8 An example SCP object shown in the Active Directory Sites and Services snap-in

Accessing Autodiscover through well-known URLs

There's nothing magic about the URLs that Autodiscover uses, but the Autodiscover protocol specification calls out how they should be formed, and Exchange servers automatically create the necessary virtual directories to handle Autodiscover queries. You don't need to modify the internal and external URLs manually for the Autodiscover virtual directory; leave them blank so that Exchange correctly constructs the well-known URLs described earlier in this chapter.

The role of Exchange providers

Autodiscover manifests contain a list of Exchange *providers*, XML elements that specify the services available and the methods required to connect to them. They're sometimes called Outlook providers, too. These components of the manifest get their name from the actual providers—code that runs on the Exchange server to look up the appropriate settings and return them. Here's an example provider block (that I have shortened for print) returned by an Autodiscover query from an Exchange 2013 multirole server; it's wrapped in a <*Protocol*> XML element:

```
<Protocol>
    <Type>EXCH</Type>
    <Server>4663133a-21a4-4da2-b6dc-dde031e6f9eb@betabasement.com</Server>
          <ServerDN>/o=BetaBasement/ou=Exchange Administrative Group
```

```
(FYDIBOHF23SPDLT)/cn=Configuration/cn=Servers/cn=4663133a-21a4-4da2-b6dc
-dde031e6f9eb@betabasement.com</ServerDN>
         <ServerVersion>73C0826C</ServerVersion>
         <MdbDN>/o=BetaBasement/ou=Exchange Administrative Group
(FYDIBOHF23SPDLT)/cn=Configuration/cn=Servers/cn=4663133a-21a4-4da2-b6dc
-dde031e6f9eb@betabasement.com/cn=Microsoft Private MDB</MdbDN>
         <PublicFolderServer>mail.betabasement.com</PublicFolderServer>
         <AD>PAO-DC01.betabasement.com</AD>
         <ASUrl>https://pao-ex01.betabasement.com/EWS/Exchange.asmx</ASUrl>
         <EwsUrl>https://pao-ex01.betabasement.com/EWS/Exchange.asmx</EwsUrl>
         <EmwsUrl>https://pao-ex01.betabasement.com/EWS/Exchange.asmx</EmwsUrl>
         <EcpUrl>https://pao-ex01.betabasement.com/ecp/</EcpUrl>
         <EcpUrl-um>?rfr=olk&amp;p=customize/voicemail.aspx&amp;exsvurl
=1&amp;realm=betabasement.com</EcpUrl-um>
         <EcpUrl-ret>?rfr=olk&amp;p=organize/retentionpolicytags.slab&amp
;exsvurl=1&amp;realm=betabasement.com</EcpUrl-ret>
         <EcpUrl-sms>?rfr=olk&amp;p=sms/textmessaging.slab&amp;exsvurl
=1&amp;realm=betabasement.com</EcpUrl-sms>
         <EcpUrl-tm>?rfr=olk&amp;ftr=TeamMailbox&amp;exsvurl=1&amp;realm
=betabasement.com</EcpUrl-tm>
         <OOFUrl>https://pao-ex01.betabasement.com/EWS/Exchange.asmx</OOFUrl>
         <UMUrl>https://pao-ex01.betabasement.com/EWS/UM2007Legacy.asmx</UMUrl>
         <OABUrl>https://pao-ex01.betabasement.com/OAB/7f9b3545-2dee-4571-bc45
-d277e3da0811/</OABUrl>
         <ServerExclusiveConnect>off</ServerExclusiveConnect>
     </Protocol>
```

As you can see, this provider includes the distinguished name (DN) of the server, the DN of the mailbox database, and, most important, the GUID of the user's mailbox (in the Server element). It also contains a set of URLs for various services. If you look at a complete Autodiscover manifest, though, you'll see several of these <*Protocol*> elements. They serve different purposes:

- The EXCH provider block is used for internal connections for Exchange 2007 and Exchange 2010 services. This data includes port settings and the internal URLs for the Exchange services that you have enabled.
- The EXPR block serves data for external Outlook Anywhere connections for Exchange 2007 and Exchange 2010 services. This setting includes the external URLs for the Exchange services that you have enabled.
- The WEB setting contains the best URL for this particular user to reach Outlook Web Access for the user to use.
- The EXHTTP block is new for Exchange 2013. It's intended to supersede the EXCH and EXPR blocks for modern clients (meaning Outlook 2010 SP1 with the latest roll-ups and later versions). EXHTTP is generated by the Exchange 2013 CAS from the

provider data supplied by the internal and external providers, and the data it returns provide a roadmap for the client to know which CAS to connect to, whether it's internal or external. There will be two EXHTTP blocks; the client will try the first one it encounters, which provides the internal settings, and then fall back to the second one, which provides the external settings.

Retrieving configuration information with Autodiscover

After the client successfully connects to an Exchange 2013 Mailbox server by using Autodiscover, it receives a manifest that contains a number of useful data items. The client is supposed to cache these data items. For example, Outlook stores several of them in the user's Outlook profile; a mobile device client might store them in whatever manner makes sense. Perhaps the most important piece of this data is the location of the user's mailbox, but there are a number of other important items.

After the client connects to a Mailbox server, it attempts to retrieve the configuration information for its mailbox and the location of the various Exchange services. This step ensures that the profile is kept updated with the latest mailbox settings and that Outlook knows how to find the following:

- Out of office information
- Availability information from the calendars of other users
- Locations to download the Offline Address Book (OAB) files
- UM information (if used)
- Information about the location and availability of shared or site mailboxes
- Information about the location and availability of public folder servers

Another important update the client receives is the name of the server to which it should connect for future queries. This server becomes the endpoint that replaces the name of the original Mailbox server that is used in the profile for previous versions of Exchange. Keep in mind that if an Autodiscover client that connects from the Internet cannot connect to the Autodiscover service, client functions that depend on retrieving Autodiscover manifest information probably won't work. For example, if Autodiscover access is misconfigured, a user who takes his work laptop home and tries to connect using Outlook Anywhere might find that Outlook appears to connect normally but that he can't change his out of office message because Outlook doesn't know where the OOF service is.

INSIDE OUT **Who is the client talking to?**

The XML data returned by an Autodiscover request is generated by a CAS in the same site as the Mailbox server that holds the requested mailbox. This means that in mixed environments, you might see Exchange 2010–style Autodiscover data returned for users whose mailboxes are still on Exchange 2010 even though the initial Autodiscover request will go to an Exchange 2013 CAS. When the Exchange 2013 CAS proxies the Autodiscover request to the Exchange 2010 CAS on the same site as the Exchange 2010 Mailbox server holding the user's mailbox, that Exchange 2010 CAS generates the Autodiscover data, and the Exchange 2013 CAS returns it to the client.

INSIDE OUT **Updating Autodiscover**

Outlook—and other clients—don't just perform Autodiscover once; mailboxes can move, server assignments can change, networks might have service interruptions, and clients can physically move between sites. For these reasons, clients usually perform periodic rediscoveries to keep their knowledge of which endpoints they should be talking to up to date. Outlook requests Autodiscover data at start-up and every 60 minutes thereafter. It might also query Autodiscover if it cannot connect to the Mailbox server, in which case it will retry its request every five minutes until it succeeds or until Outlook shuts down.

Although there are rarely good reasons to do so, you can modify the interval at which Outlook performs this query by running the Set-OutlookProvider cmdlet to set a new cache lifetime in hours. For example, to set the lifetime to be three hours:

```
Set-OutlookProvider –id 'msExchAutoDiscoverConfig' –TTL 3
```

Understanding CAS proxying and redirection

The Exchange 2013 CAS role doesn't directly serve any data to clients; instead, it provides two ways for clients to connect to the correct server: *proxying* and *redirection*. In a proxied connection, the CAS accepts data from the client and forwards it to the correct server; in a redirected connection, the CAS responds to the client request with the FQDN of the server it should ideally be talking to.

In Exchange 2013, the CAS only redirects connections in a very limited number of cases. That is by design because redirections shift work from the server to the client by forcing it to reconnect, and not all clients properly handle redirections.

Proxying

CAS proxying was fairly complicated in Exchange 2007 and Exchange 2010. It is considerably simpler in Exchange 2013, but there are still several edge cases that lurk, somewhat confusingly, to cover situations when traffic must be proxied to downlevel Exchange servers or across complex Active Directory site topologies.

First, you need to stipulate that two things must take place before the CAS can proxy anything at all. The CAS that initially accepts the connection (here referred to as the original CAS) must authenticate the client to see whether it should have access, and it must determine which Mailbox server is currently hosting the active copy of the mailbox database.

After these steps have been completed, the CAS can proxy traffic. The simplest case of proxying involves IMAP and POP, which the CAS will always proxy without regard to the version of Exchange on the target Mailbox server.

Several Exchange protocols, including Outlook Anywhere, Autodiscover, Outlook Web App, Exchange ActiveSync (EAS), the Availability service, and Exchange Web Services (EWS), can be carried over HTTP or HTTPS. The good news is that the CAS role doesn't have to care about the contents of the traffic for most of these protocols, just about the destination endpoint. For example, a client with a mailbox homed on Exchange 2010 that makes an Autodiscover request will receive Autodiscover data from an Exchange 2010 CAS because the Exchange 2013 CAS role will proxy that Autodiscover request to an Exchange 2010 CAS, ideally on the same site as the Exchange 2010 Mailbox server that hosts the target mailbox.

There are a few exceptions to this general rule, all of which involve coexistence with downlevel versions of Exchange. For example, suppose an Exchange 2013 CAS named PAOCAS02 receives an Autodiscover request for a user whose mailbox is hosted on an Exchange 2007 server named AUSMBX03. The Exchange 2013 CAS will detect that the user's mailbox is on Exchange 2007 and proxy the connection to an Exchange 2013 Mailbox server, which will return an Autodiscover manifest itself instead of proxying that request to AUSMBX03. That is possible because the Exchange 2013 Mailbox role includes special code for handling Autodiscover requests on behalf of Exchange 2007 mailboxes. (Outlook Anywhere proxying is a special case because of the differences in Outlook Anywhere among Exchange 2007, Exchange 2010, and Exchange 2013, so it's discussed in the "The role of Outlook Anywhere" section earlier in the chapter.)

In topologies that have multiple Exchange 2010 CAS servers that could be proxy targets, the Exchange 2013 CAS role has to know which downlevel servers can accept traffic. This is

implemented using a simple mechanism: at start-up, the Exchange 2013 CAS queries Active Directory to get a list of all the Exchange 2007 or Exchange 2010 CAS servers. It then sends an HTTP HEAD request to each protocol virtual directory of those servers every 60 seconds. The HEAD request just asks for the page header of an endpoint, as distinguished from the more common HTTP GET request, so it's a lightweight way to see whether the protocol virtual directory is actually available. If the downlevel server responds with a 300-series or 400-series HTTP result, the Exchange 2013 CAS considers that particular virtual directory on the target server available. If the request times out or produces a 500-series HTTP result, the HEAD request is immediately retried; if the second attempt fails, that virtual directory is considered down, and no traffic will be proxied to it.

INSIDE OUT **Exempting an Exchange 2010 CAS as a proxy target**

If you want a particular Exchange 2010 CAS to be exempted from proxying traffic, you can do so provided you have at least Service Pack 3 on it. You might want to do this, for example, if you know the CAS will be down for maintenance or if you plan to decommission it. The trick is to use the –IsOutOfService parameter to Set-ClientAccessServer, like this:

```
Set-ClientAccessServer –identity HSV-CAS06 –IsOutOfService $true
```

After you run that command, HSV-CAS06 won't receive any more proxy traffic from Exchange 2013 CAS servers until you reset IsOutOfService.

Redirection

Exchange 2013 redirects CAS connections in the following cases:

- When an inbound unified messaging call arrives. The Session Initiation Protocol (SIP), discussed in more detail in Chapter 6, is built around the idea of redirects for call routing, and the Unified Messaging Call Router service on the CAS takes full advantage of this.
- When a client requests Outlook Web App and the target mailbox is on Exchange 2007. CAS 2013 redirects the client to an Exchange 2007 CAS. This is mandatory because Exchange 2007 can't accept a proxied Outlook Web App connection from Exchange 2013.
- When a client requests an Outlook Web App connection and the target mailbox is on Exchange 2013 in another Active Directory site, but only if the ExternalURL property is set on the foreign-site CAS. For example, suppose that a client connects to an

Internet-facing CAS named HSV-CAS04 on an Active Directory site named Huntsville, but the active copy of the mailbox is actually on a Mailbox server named PNS-MBX02 in the Pensacola Active Directory site. If ExternalURL is set on PNS-MBX02, then HSV-CAS04 will return that URL to the client so it can connect directly. If ExternalURL is not set on the Pensacola server, or if it is set to the same value as is set on HSV-CAS04, then HSV-CAS04 will proxy the connection instead.

There's another case when the 2013 CAS role should redirect. When a user types the FQDN of the CAS without including /owa on the end, a new redirection module provided by Exchange and loaded into IIS automatically redirects the user to Outlook Web App. This is a labor saver for users; although it was possible to do the same thing in Exchange 2007 and Exchange 2010, it required manual configuration.

CAS coexistence and migration

From a coexistence perspective, it is critical to update your Exchange organization so that all incoming CAS traffic flows first to an Exchange 2013 CAS. This is probably the biggest required change you'll face from a CAS perspective, although if you are using load balancing and want to take advantage of the end of the requirement for affinity, those changes will run a close second.

Routing inbound traffic to the 2013 CAS role

Making this change is mostly a matter of configuring systems other than Exchange. In most environments, you'll have to do some or all of the following to get traffic flowing to the right place:

- Evaluate your Active Directory site topology to decide whether there are any aspects of it you want to change.
- Obtain and install certificates that match the appropriate server names (see the "Certificate management" section later in the chapter for more on this).
- Be ready to update your reverse proxy, firewall, and/or load balancer to send inbound connections to the new CAS servers. (However, don't actually make the updates until the new servers are installed!)
- Review your internal and external DNS configuration and make any necessary updates.

After you have completed these prerequisite steps, you can install a single Exchange 2013 CAS and verify that traffic is flowing as you would expect. Where should you install that CAS? Microsoft recommends putting it on whichever Internet-facing site currently handles

Autodiscover requests, because that will allow the 2013 CAS to handle Autodiscover requests correctly for mailboxes on Exchange 2013 servers and whatever earlier versions you have around.

After installing that first CAS, you need to verify that it is reachable from the Internet, using whatever external URLs you want it to use. This will probably involve using Set-OutlookAnywhere and the Set-*VirtualDirectory cmdlets. At that point, you can add more Exchange 2013 servers, either as individual or multirole servers.

Removing ambiguous URLs

In Exchange 2010, Microsoft recommended that the CAS array objects should not be directly resolvable by external clients. One way to achieve this was to implement split-brain DNS, which is what many sites did. Microsoft also recommended assigning separate names for the CAS array object (to which ordinary MAPI RPC clients connect) and the Outlook Anywhere hostname. However, if you assigned the same name to those services, everything would still work fine. That led many administrators to take a shortcut that seemed perfectly reasonable at the time: they set the internal HTTP namespace to the same name as the RPC client access array and then depended on split-brain DNS to keep internal clients from connecting to the external IP address for the array and vice versa. Because that seemed to work well, many of them went ahead and assigned the same externally resolvable FQDN for other services such as Exchange ActiveSync or Outlook Web App.

The problem with this approach is that after you install an Exchange 2013 CAS, your MAPI clients will stop working! Why? They'll eventually try to connect to the single FQDN you have defined, which now points at an Exchange 2013 CAS. The Exchange 2013 CAS won't accept MAPI RPC connections, so those clients won't be able to connect unless they can successfully fall back to Outlook Anywhere; their ability to do so will depend on how the individual client is configured.

The recommended Microsoft way to fix this is to change the name on the CAS array object to a unique value and then update the *RpcClientAccessServer* value on your Exchange 2010 mailbox databases to point to that FQDN. That covers the case of newly created or manually updated Outlook profiles; for the other profiles, enable Outlook Anywhere for all clients and servers *before* you deploy Exchange 2013. By doing so, you ensure that your Outlook clients can successfully connect to the Exchange 2013 CAS even if they are unable to make a MAPI RPC connection. Brian Day at Microsoft has a long post on the Exchange team blog that goes into greater depth on how to resolve this situation, and I encourage you to read it before proceeding with your upgrade from Exchange 2010 to Exchange 2013: *http://blogs.technet.com/b/exchange/archive/2013/05/23/ambiguous-urls-and-their-effect-on-exchange-2010-to-exchange-2013-migrations.aspx*.

Certificate management

Although both the CAS and Mailbox roles use certificates, the CAS role depends on them much more heavily, which is why I'm talking about them so early in the book. Although Exchange 2013 (like the two preceding versions) installs self-signed certificates on each server as you install it, most organizations find that self-signed certificates don't meet their needs.

How Exchange uses certificates

Choosing the right set of certificates begins with understanding how Exchange uses certificates. The Mailbox and CAS roles each use certificates in a variety of ways:

- To give clients a way to authenticate the identity of a server. This is the most common use, and it's the one that generates the most work for Exchange administrators due to certificate name mismatches or other trust issues that cause user complaints.
- To authenticate a client or device to the server. This use, known as *client certificate authentication*, is interesting because a device that authenticates with a user's certificate doesn't have to send or even have the user's Windows credentials, so a compromised device can't be exploited to gain access to other services.
- To secure SMTP mail in transit by encrypting the SMTP connection with Transport Layer Security (TLS). TLS protection is automatically applied when Exchange servers in the same organization exchange mail; it can also be enabled for communications with external servers. This use is covered in more detail in Chapter 2.
- To give servers a way to authenticate the identity of other servers. This use, known as *mutual TLS*, is primarily of interest when integrating with Lync Server because Lync depends on mutual TLS. See Chapter 8, "Office 365: A whirlwind tour," for more on this use.

Windows, and applications that use it, can also use certificates for other purposes such as IPsec encryption or digitally signing executable code or Windows PowerShell scripts. I won't consider those purposes in the context of Exchange, although I will point out that Exchange does not, and cannot, use the IIS 8.0 centralized certificate store, a nifty feature in Windows Server 2012 that would be of great use if Exchange supported it.

If you only run Exchange for an internal network and never want to allow access from the Internet, the set of self-signed certificates installed as part of the Exchange 2013 setup program is sufficient for your purposes as long as you're willing to put up with a few shortcomings. Outlook Web App and Exchange ActiveSync users will have to install the self-signed certificates or face nagging browser warnings. For domain-joined computers