David Chappell
& Associates

PROVIDING SINGLE SIGN-ON TO AMAZON EC2 APPLICATIONS FROM AN ON-PREMISES WINDOWS DOMAIN

CONNECTING TO THE CLOUD

DAVID CHAPPELL

DECEMBER 2009

SPONSORED BY AMAZON AND MICROSOFT CORPORATION

CONTENTS

THE CHALLENGE: PROVIDING SINGLE SIGN-ON TO AMAZON EC2 APPLICATIONS

Users hate having multiple passwords. Help desks hate multiple passwords too, since users forget them. Even IT operations people hate them, because managing and synchronizing multiple passwords is expensive and problematic.

Providing *single sign-on (SSO)* lets users log in just once, then access many applications without needing to enter more passwords. It can also make organizations more secure by reducing the number of passwords that must be maintained. And for vendors of Software as a Service (SaaS), SSO can make their applications more attractive by letting users access them with less effort.

Allowing SSO across as many applications as possible makes users happier, increases security, and lowers support costs. All of these are worthy goals. Yet providing SSO often takes work. And with the emergence of cloud platforms, new SSO challenges have appeared.

For example, Amazon Web Services (AWS) provides the Amazon Elastic Compute Cloud (Amazon EC2). This technology lets a customer create Amazon Machine Images (AMIs) containing an operating system, applications, and more. The customer can then launch instances of those AMIs—virtual machines—to run applications on the Amazon cloud. Similarly, Microsoft provides Windows Azure, which lets customers run Windows applications on Microsoft's cloud. When an application running on a cloud platform needs to be accessed by a user in an on-premises Windows domain, giving that user single sign-on makes sense.

Fortunately, there are several ways to do this. For applications running on Amazon EC2, the options fall into two categories:

- Use Amazon Virtual Private Cloud (Amazon VPC), which allows Amazon EC2 instances and the Windows applications they contain to be part of an on-premises Active Directory forest.
- Use Microsoft Active Directory Federation Services (AD FS), which relies on establishing trust relationships between the Amazon EC2 environment and an on-premises Windows domain.

This paper provides an architectural overview of both approaches, describing when each one makes sense.

USING AMAZON VIRTUAL PRIVATE CLOUD

Amazon VPC is a straightforward idea: A group of one or more Amazon EC2 instances can be connected to an on-premises Windows domain using IPsec, the standard protocol for creating a virtual private network (VPN). Figure 1 illustrates this idea.

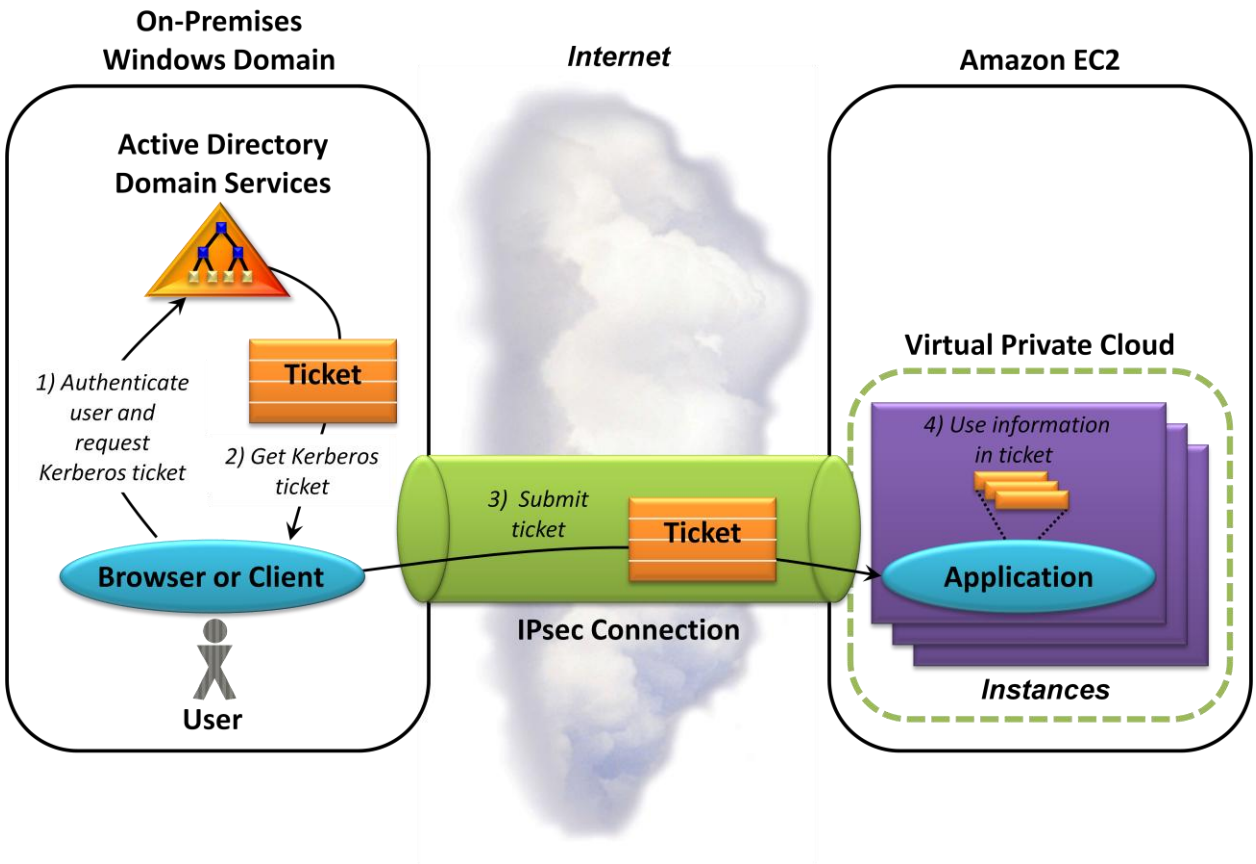

Figure 1: Amazon VPC allows applications running in Amazon EC2 instances to be part of an on-premises Active Directory forest.

The Amazon EC2 instances in the Amazon VPC are assigned IP addresses that are compatible with the network infrastructure of the on-premises Windows domain. The result is much like a branch office connected via a VPN: The instances in Amazon EC2 appear to be part of the same network as the on-premises Windows domain.

These instances belong to the same Active Directory forest as the on-premises Windows domain. There are three different approaches to combining the two worlds:

- Make the Amazon EC2 instances inside the VPC part of an existing on-premises Windows domain and site. This option doesn't require running a domain controller in the VPC, and it's the approach that Figure 1 shows.
- Make the Amazon EC2 instances inside the VPC a new site in an existing on-premises Windows domain. This option requires running Active Directory Domain Services in Amazon EC2.
- Make the Amazon EC2 instances inside the VPC a new domain in an existing on-premises Windows forest. This option also requires running Active Directory Domain Services in Amazon EC2.

Figure 1 shows how identity information flows for the first option, creating a VPC without its own domain controller. Here, when a user in the on-premises domain wishes to access an application in the VPC, she authenticates herself and requests a Kerberos ticket for the application as usual (step 1). Once this ticket

is granted (step 2), her browser or client application sends the ticket to the application in the VPC (step 3). This application then verifies the ticket and uses the information it contains (step 4). These are exactly the same steps that occur during Kerberos-based access to an application running in the user's on-premises domain—there's no difference at all.

This approach to providing SSO makes sense whenever a VPC makes sense. If the instances in Amazon EC2 are controlled by the same organization that controls the on-premises domain, for example, and if establishing a VPN connection between the two is feasible, making the VPC part of the domain can be a good solution.

USING MICROSOFT ACTIVE DIRECTORY FEDERATION SERVICES

Amazon VPC is a fine solution in some situations. There are other cases, however, in which providing SSO through a VPC isn't the best approach. Instead, using Active Directory Federation Services can make more sense.

AD FS can be used in two ways: An AD FS server can run only on-premises, or AD FS servers can run both on-premises and in a public Amazon EC2 instance. This section takes a look at both possibilities, showing how they work and describing typical scenarios for each one.

RUNNING AD FS ON PREMISES

Running AD FS on premises is simpler than running it both on premises and in an Amazon EC2 instance. Accordingly, the place to start is by seeing how this works first with AD FS 1.1, then with AD FS 2.0.

Using AD FS 1.1

Microsoft made AD FS part of Windows Server several years ago. As its name suggests, this technology can be used to provide *identity federation*—and thus SSO—across different identity scopes. Figure 2 shows one way to do federation between an on-premises Windows domain and an application running on Amazon EC2.

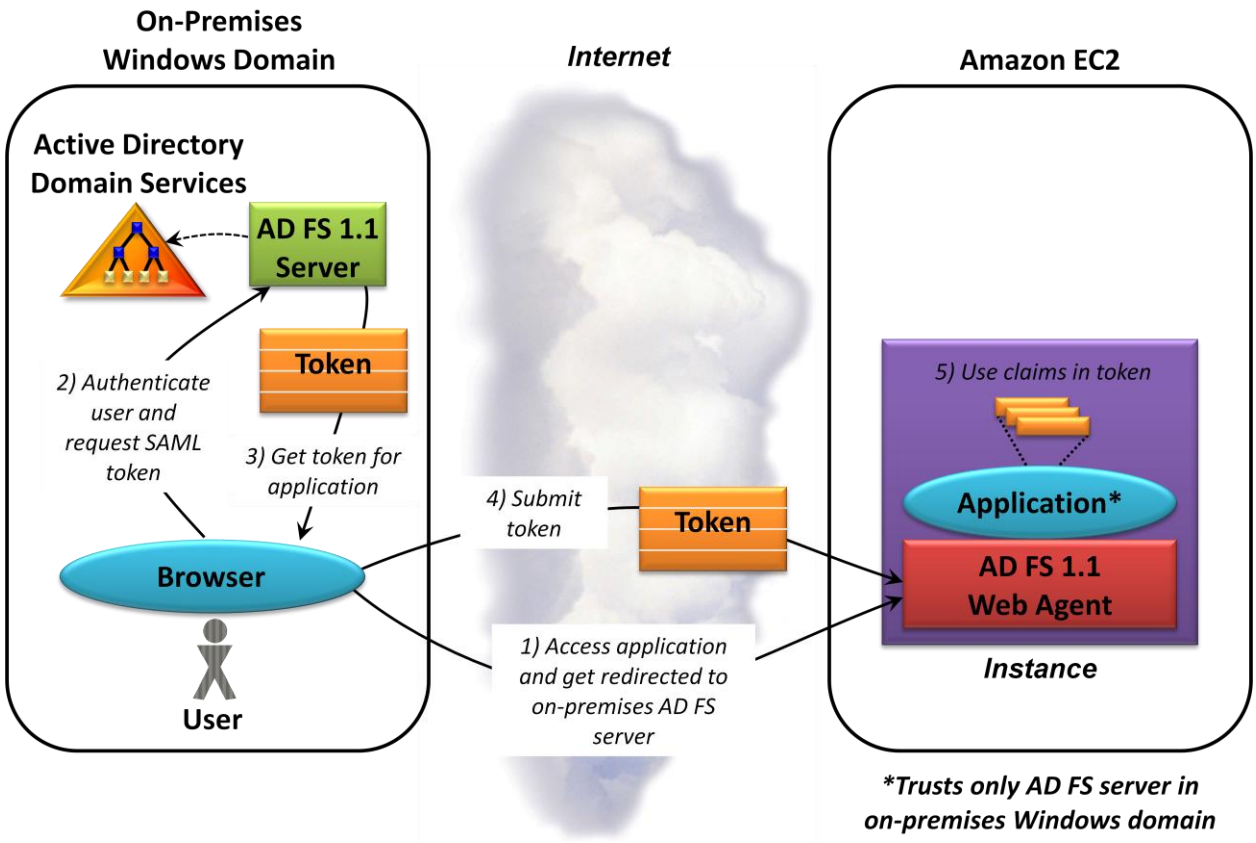

Figure 2: An application running in Amazon EC2 can allow browser access without requiring a separate login by trusting an on-premises AD FS 1.1 server.

In this example, a user in an on-premises Windows domain wishes to access a Windows application running in Amazon EC2. Access to this application must go through the AD FS 1.1 Web agent, however, which an administrator has installed in the AMI for the Amazon EC2 instance in which the application is running. This application is configured to trust only identity information supplied by the AD FS 1.1 server in the on-premises Windows domain.

When the browser attempts to access this application, the Web agent redirects the browser to the on-premises AD FS server (step 1). The browser supplies a Kerberos ticket to authenticate the user to this server, then requests a token to the Amazon EC2 application (step 2). This token is formatted using the Security Assertion Markup Language (SAML), and it contains information about this user. This information, known as *claims*, can include whatever the application needs. (The AD FS server must be configured to send the right claims for each application.) Here, for instance, the application might need to know the user's name, a list of roles she can act in, and more. Whatever claims are required, AD FS creates the requested token, digitally signs it, and returns it to the user (step 3). Once it has this SAML token, the browser sends it to the application (step 4). The Web agent then checks the token's signature to verify that it was issued by the trusted AD FS server. If the signature is correct, the claims in the token can be used by the application (step 5).

From the user's point of view, all of this is invisible. All she knows is that her attempt to access the application has succeeded. She can remain blissfully unaware of the exchanges needed to make this happen.

This scenario assumes that the Amazon EC2 instance is running Windows. This isn't required, however. The protocol exchanges shown in Figure 2 are also possible for applications running in a Linux instance if the instance's administrator has installed an agent that supports the identity federation protocol implemented by the AD FS Web agent. This protocol is called WS-Federation, and companies such as Quest and Centrify currently provide Linux agents that support it.

When does this approach make sense? Here are a couple of typical scenarios:

- Suppose an organization creates an application and runs it in Amazon EC2. Or maybe the organization buys an application and deploys it in one or more Amazon EC2 instances. In either case, installing the AD FS Web agent and configuring the application to trust the organization's on-premises AD FS server can provide users in that organization with single sign-on.

- If an organization's employees need to access an application running on Amazon EC2 from remote locations, such as a coffee shop, using AD FS as shown in Figure 2 can also make sense. Although the details get a bit more complicated, it's possible to install an *AD FS proxy* that can use Forms authentication to let a remote user enter her domain credentials (i.e., her username and password). This AD FS proxy uses these to log the user into the Windows domain and returns a SAML token for the application she's trying to access. Her browser then presents this SAML token to the application, as in step 4 in Figure 2.

AD FS 1.1 is a useful technology. Still, it has some limitations. For one thing, it supports only browsers. Smart clients, such as an application built with Windows Communication Foundation (WCF), can't use it effectively. AD FS 1.1 also lacks some of what's required to be a solid foundation for *claims-based identity*, a more general approach promoted by Microsoft, IBM, and others. As described next, AD FS 2.0 addresses these limitations.

Using AD FS 2.0

AD FS 2.0 is backward compatible with AD FS 1.1. This means that the exchanges shown in Figure 2 would also work with AD FS 2.0, since this newer version also supports WS-Federation. Yet things change a bit with AD FS 2.0, as Figure 3 shows.

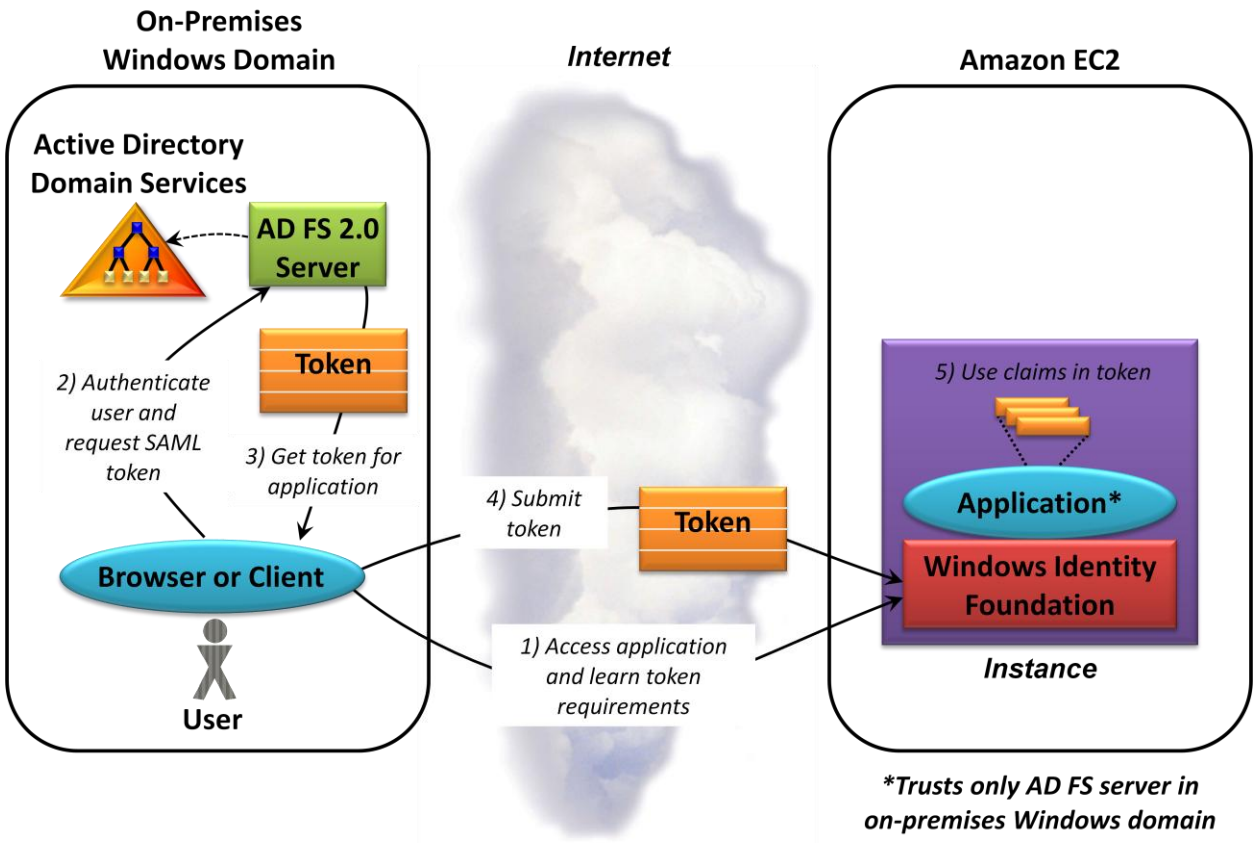

Figure 3: An application running in Amazon EC2 can allow access from browsers and other client applications without a separate login by trusting an on-premises AD FS 2.0 server.

As the figure shows, this example is much like the previous one, as are the scenarios in which using it makes sense. Still, the differences are important. Along with the upgrade to an AD FS 2.0 server, notice that the AD FS 1.1 Web agent has been replaced with Windows Identity Foundation (WIF). These changes let AD FS 2.0 work with both browsers and non-browser clients, such as a WCF application, letting a wider range of applications offer single sign-on to their users. Notice also that in step 1, WIF can explicitly tell the browser or client what its requirements are for a token. This makes it possible to use an identity selector such as Windows CardSpace, giving the user the option of choosing which identity to use (a possibility that's not shown here). And WIF provides a more general interface for working with the claims in a token than does the AD FS 1.1 Web agent, something that's useful for quite a few applications.

One more difference is that AD FS 2.0 implements something called a *Security Token Service (STS)*. Rather than relying solely on browser redirects, as does WS-Federation, an STS accepts requests made using WS-Trust, a standard SOAP-based protocol. The AD FS 2.0 server also supports the SAML 2.0 protocol, another standard option for identity federation. Accordingly, this newer version of the technology can work with a wider range of non-Microsoft products than its predecessor.

In fact, all of the protocol exchanges shown in the figure are based on industry standards. This means that as in the previous scenario, this approach to identity federation can work with an Amazon EC2 instance running Linux. As before, an administrator can install a Linux agent in the instance's AMI that implements

the correct identity federation protocol. Alternatively, the creator of the Linux application could directly modify it to work with the SAML token it receives.

Finally, while this example uses an AD FS 2.0 server to generate the SAML token, it's important to understand that IBM and other vendors also provide technologies that can be used to create these tokens. Claims-based identity is built on multi-vendor standards that are implemented by several different companies. The move to this new way of handling identity isn't a Microsoft-only effort.

RUNNING AD FS ON PREMISES AND IN AMAZON EC2

So far, the application running in Amazon EC2 has been configured to trust an AD FS server in an on-premises Windows domain. This is fine for some situations, such as those where the Amazon EC2 application and the AD FS server are owned by a single organization. But what if it's a SaaS application accessed by users in several different organizations? Configuring this cloud application to trust each customer's on-premises Windows domain directly is problematic.

A better approach is to create a Windows domain in Amazon EC2, then install an AD FS server in that domain. Once this is done, an application in this Amazon EC2 Windows domain can be configured to trust its own AD FS server rather than those in the on-premises domains of its users. This Amazon EC2 AD FS server, in turn, can then be configured to trust the AD FS servers in the on-premises domains that contain the users of these applications. AD FS servers are designed for this kind of cross-domain configuration, while individual applications are not. And once again, this can be done using either AD FS 1.1 or AD FS 2.0.

Using AD FS 1.1

Having two separate domains with two instances of AD FS makes the situation slightly more complicated than those described earlier. This complexity is hidden from the user, of course, but understanding this scenario requires knowing what's really going on. Figure 4 shows the first steps in the process when AD FS 1.1 is used.

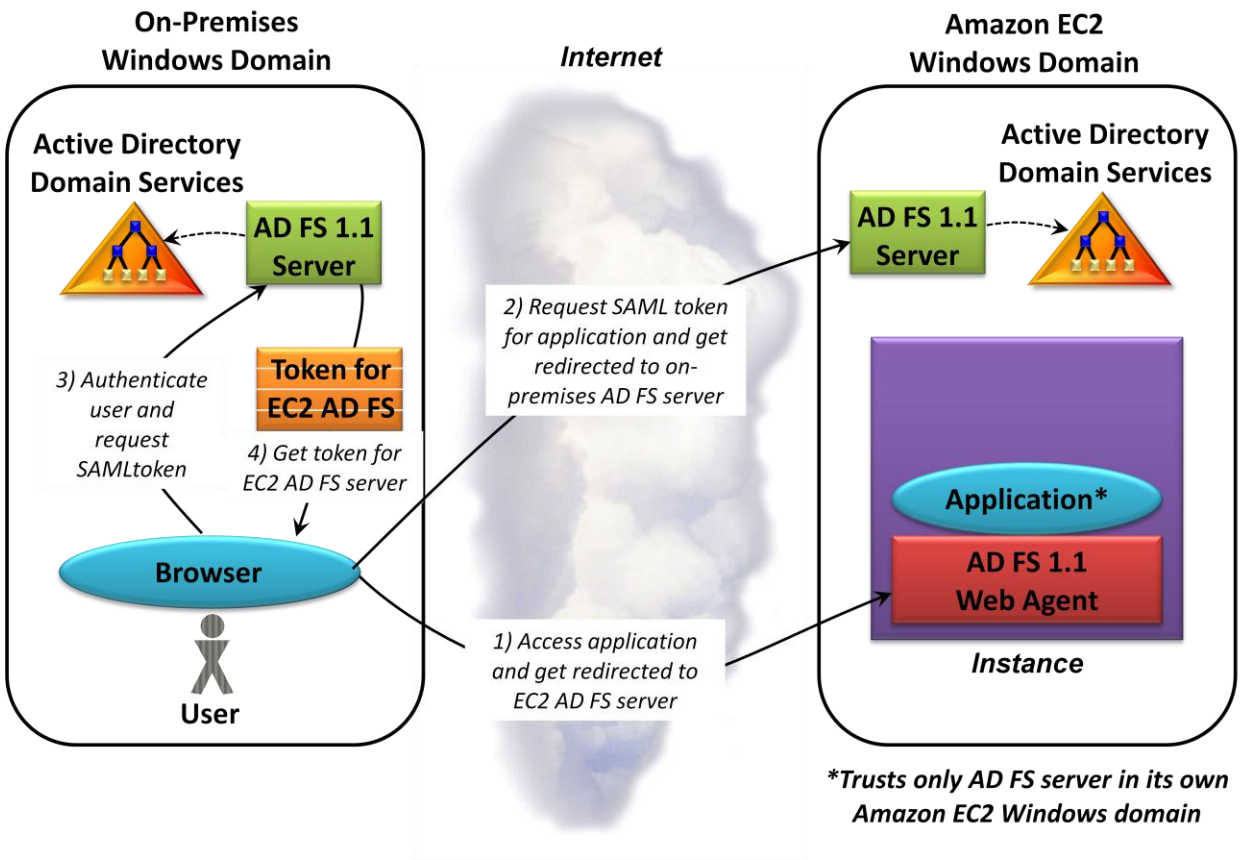

Figure 4: Running an AD FS server in both the on-premises Windows domain and an Amazon EC2 Windows domain can simplify some identity federation situations.

The process begins much as it did in the previous AD FS 1.1 scenario: The user accesses the application and is redirected to an AD FS server (step 1). This time, however, that redirect isn't to the AD FS server in the user's own Windows domain. Instead, the browser is redirected to an AD FS server in the application's domain, i.e., a Windows domain created in Amazon EC2. This is the only AD FS server that this application trusts, which means that the application will only accept SAML tokens signed by this server.

Once it receives this redirect, the browser dutifully requests a SAML token from the AD FS server in the Windows domain running in Amazon EC2. When it does this, however, that server once again redirects the browser, this time back to the AD FS server in the user's own domain (step 2). To get a token for this application, the user must first present a token from her own AD FS server. To get this token, her browser authenticates her to the AD FS server in her own Windows domain, probably using a Kerberos ticket, and asks for the token she needs (step 3). The token she's asking for isn't to the application itself, however. Instead, it's a token that will prove her identity to the AD FS server in the application's domain. Once she gets this token (step 4), she can present it to that remote AD FS server to get what she really wants: a token for the application itself. Figure 5 shows how the second half of this process looks.

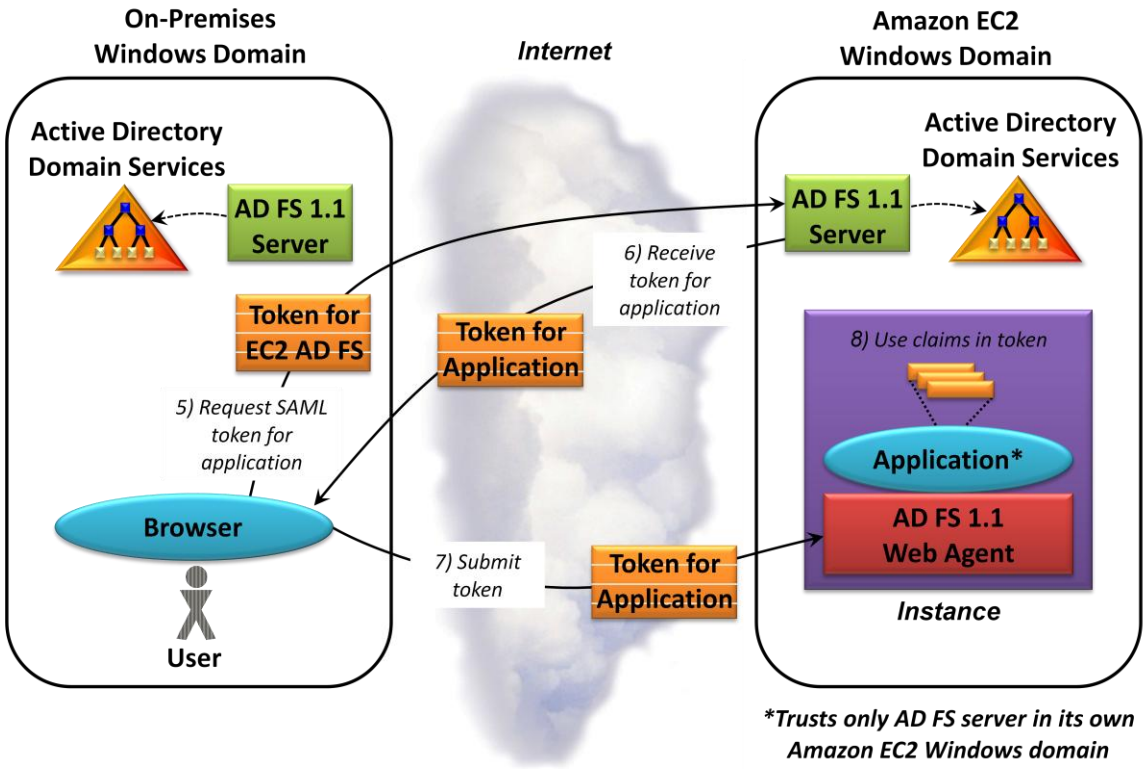

Figure 5: Because the application trusts only its own AD FS server, the browser must get a token from that server to access the application.

Now that the user's browser has a token for the AD FS server in the Amazon EC2 domain, it can present this token to that server and request a token for the application the user wishes to access (step 5). The AD FS server running in Amazon EC2 verifies that the token it receives was signed by an AD FS server in a domain that it trusts, then issues the requested token (step 6). This new token is signed by the AD FS server in the Amazon EC2 domain, which means that the application in that domain will trust it. When the browser supplies this token (step 7), the Web agent can verify its signature, then pass the claims it contains on to the application. The application can then use those claims to make an authorization decision or for some other purpose (step 8).

Notice that the AD FS server in the Amazon EC2 domain receives a SAML token in step 5, then issues another one in step 6. These two tokens might or might not contain the same claims. Suppose, for example, that this application is accessed by users in many different Windows domains, and the claims those users provide aren't exactly the same. One organization might use the string "Manager" to indicate that the user is in a managerial role, another might use "Mgr", while a third uses the numeric code "30944". To make things more consistent, the AD FS server in the Amazon EC2 domain might convert all of these to "Manager". This *claims transformation* makes life simpler for the application, since it no longer needs to deal with a diversity of ways to express the same thing.

Using AD FS 2.0

The previous scenario uses AD FS 1.1, and so only browser clients are supported. As described earlier, AD FS 2.0 can support both browsers and other clients. Using this newer technology requires an Amazon EC2

instance running Windows Server 2008—the AD FS 2.0 server can't run on earlier Windows versions. Amazon EC2 now supports Windows Server 2008, however, which makes using AD FS 2.0 possible.

The scenario is much like the one shown in Figures 4 and 5. The differences are straightforward: WIF replaces the AD FS 1.1 Web agent, and AD FS 2.0 servers replace the AD FS 1.1 servers. Doing this brings all of the advantages of claims-based security, including support for browser and non-browser clients as well as a more general way for applications to work with claims. Going forward, federation-based SSO projects using Windows Server 2008 should use AD FS 2.0 and WIF rather than their predecessors.

ANOTHER VIEW: PROVIDING SINGLE SIGN-ON TO WINDOWS AZURE APPLICATIONS

Microsoft's Windows Azure is similar in some ways to Amazon EC2. Unsurprisingly, applications running on Windows Azure can also provide single sign-on to users in an on-premises Windows domain. For example, a developer can create a Windows Azure application that uses Windows Identity Foundation, much like the scenario shown earlier in Figure 3. This application can then be configured to trust an on-premises AD FS 2.0 server. And while running an AD FS 2.0 server on Windows Azure isn't possible today, it is possible to build a custom STS for Windows Azure using WIF.

CONCLUSIONS

Single sign-on is important. Simplifying the lives of users is a good thing, as is reducing cost and complexity. Cloud platforms are also important, since they're being pressed into service by more and more organizations.

The implication is clear: Providing single sign-on from on-premises Windows domains to cloud-based applications is about to become a requirement. For applications running on Amazon EC2, there are several ways to do this. Which choice is best depends on the exact problem to be solved. Yet whatever option you choose, you can look forward to the same things: happier users and lower costs.

FOR FURTHER READING

Amazon Virtual Private Cloud: http://aws.amazon.com/vpc

Microsoft Patterns & Practices: A Guide for Claims-Based Identity and Access Control
http://claimsid.codeplex.com

ABOUT THE AUTHOR

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.