

Implementing an Identity Strategy for Amazon Web Services

Refreshed 14 June 2018, Published 24 February 2017 - ID G00292666 - 56 min read

FOUNDATIONAL This research is reviewed periodically for accuracy.

By Analysts [Mark Diodati](#)

Supporting Key Initiative is [Cloud Computing](#)

The identity environment for AWS services is comprehensive, but complex. Technical professionals will appreciate features like OpenID Connect, Cognito User Pools, Active Directory services and AWS Organizations, but will face challenges in areas such as user and access management and access control.

More on This Topic

This is part of an in-depth collection of research. See the collection:

- [Research Roundup: Building and Marketing Cloud-Based Offerings — 2Q18](#)

Overview

Key Findings

- Amazon Web Services (AWS) has added many features and services in recent years to bolster its identity and access management (IAM) capabilities. As enterprises increase their usage of AWS, they should understand the strengths — as well as the constraints — of these capabilities.
- New features include AWS Microsoft AD for providing Active Directory services for Amazon EC2; Amazon Cognito User Pools to manage mobile application users; and AWS Organizations to restrict privileges within each tenant.
- Although AWS offers great flexibility and a comprehensive set of IaaS services, it also poses some challenges for user access control, which is essential for compliance and governance purposes. Drawbacks include the lack of a single, extensible user directory; labor-intensive policy management; and the lack of modern access control methods for the AWS API.

Recommendations

Technical professionals in charge of the IAM program for AWS services should:

- Push AWS to make improvements to its IAM features. Press the company for an extensible user directory that includes all users; simpler policy management; and modern OAuth-based access control for the AWS API.
- Limit use of the AWS root administrative account. This account has unrestricted access to the tenant, and unauthorized use could result in data destruction or theft by a malicious actor. Use AWS IAM and the new AWS Organizations service to protect data and services, and use multifactor authentication for privileged IAM users.
- Use the AWS Microsoft AD service to migrate Windows-based applications to EC2, but perform thorough testing to weed out incompatibilities.
- Avoid using AWS's older, proprietary federation technology. OpenID Connect and SAML do a better job, without the need to create custom identity brokers and store IAM user credentials.

Analysis

AWS has become a prominent part of IT strategies, as organizations increasingly leverage its services to underpin critical systems and resources. This usage is reflected in AWS's explosive growth in revenue, which leapt from **\$6 billion in 2015** (<http://www.cio.com/article/2921180/cloud-computing/amazon-opens-up-about-aws-revenues.html>) (the first year that Amazon broke out AWS revenue) to an expected \$12 billion in 2016. Between 2012 and 2016, the number of AWS services expanded from 22 to more than 90 (growth of over 400%). It is no surprise that AWS is the infrastructure as a service (IaaS) leader, as reflected in Gartner's "[Magic Quadrant for Cloud Infrastructure as a Service, Worldwide.](https://www.gartner.com/document/code/278620?ref=grbody&refval=3620417)" (<https://www.gartner.com/document/code/278620?ref=grbody&refval=3620417>)

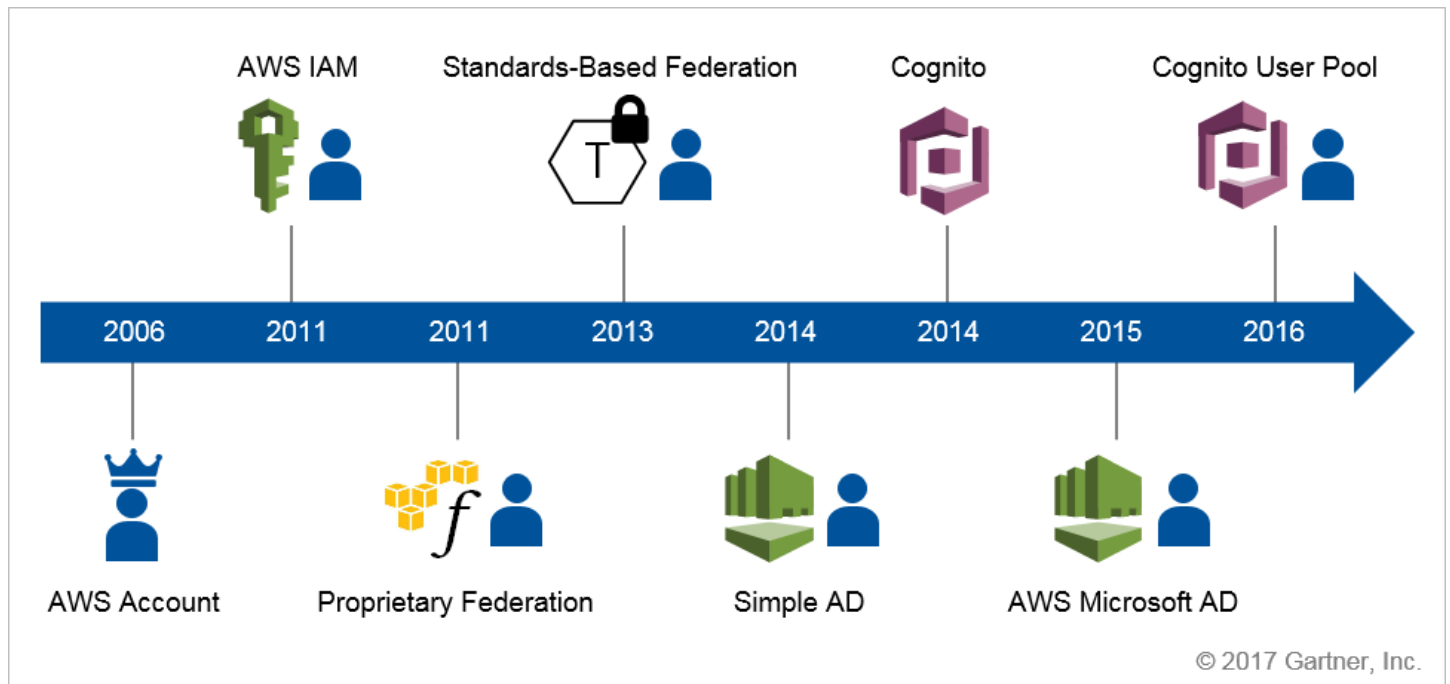
However, AWS's rapid service expansion has resulted in technical debt, particularly with regard to identity management. Instead of leveraging its core IAM for new services, it added additional user types, along with new authentication and access methods, to bring these services to market more quickly. This type of expansion is analogous to building additional rooms onto a house, and then installing individual heating and plumbing systems for each room. The proliferation of user types, access methods and authentication methods is a major challenge, because it poses too much complexity for technical professionals and developers to deal with — opening the door for mistakes, inadequate access control or security lapses.

Because of the technical debt it has accumulated over the years, AWS

identity management is approaching its complexity limit.

Figure 1 illustrates the addition of identity namespaces within AWS to accommodate newer services.

Figure 1. AWS Identity Environment



Note: This figure does not list the AD Connector user type, which is stored in an on-premises Active Directory.

Source: Gartner (February 2017)

Organizations must ensure that they thoroughly understand the issues and complexities associated with AWS identity – and that they possess the information they need to answer questions from their auditors and regulators. Organizations need to pay attention to IAM because:

- They may be storing sensitive data in AWS IaaS services, and running business-critical applications.
- IaaS services are a popular target for hackers. They contain a high percentage of privileged users, whose accounts, if compromised, could be used to breach and delete data, and to facilitate denial of service (DoS) attacks.

This report helps technical professionals evaluate key IAM features and recent enhancements offered by AWS. It assesses the strengths and challenges associated with those features, and offers recommendations for architects engaged in planning, managing or implementing IAM solutions for AWS services.

Identity

Because of the many services it added over the past six years, AWS has a fragmented identity environment. As illustrated in Figure 1, AWS has eight different user types — each with its own namespace, authentication methods and access methods. For detailed information on these user types, see the AWS User Types discussion in The Details section at the end of this report.

Some of these user types employ AWS-proprietary access methods; others employ standards-based ones; and still others employ a combination the two. These user types have varying degrees to which they can be effectively tracked and audited, and few of them are included in AWS's user directories.

For detailed descriptions of the mechanisms under which the various AWS user types authenticate to AWS and receive access rights, see the Authentication Methods and Access Methods discussions in The Details section.

AWS IAM User Directory

Except for users of the AWS account (aka the "root account"), AWS IAM users are the most privileged users within AWS. IAM users are defined in the AWS IAM user directory (a good thing). However, the IAM user directory isn't suitable as a general user directory, because it doesn't contain all the AWS user types. Moreover, it supports only four attributes. For example, the IAM directory does not support given-name, common-name or surname attributes, and it cannot be extended to support such additional attributes.

In 2016, AWS introduced another directory within the Cognito service — the User Pool — which supports schema customization (see the Amazon Cognito section, below). AWS would do well to evolve the IAM user directory to become more like the Cognito User Pool — or to converge User Pool functionality with the IAM user directory.

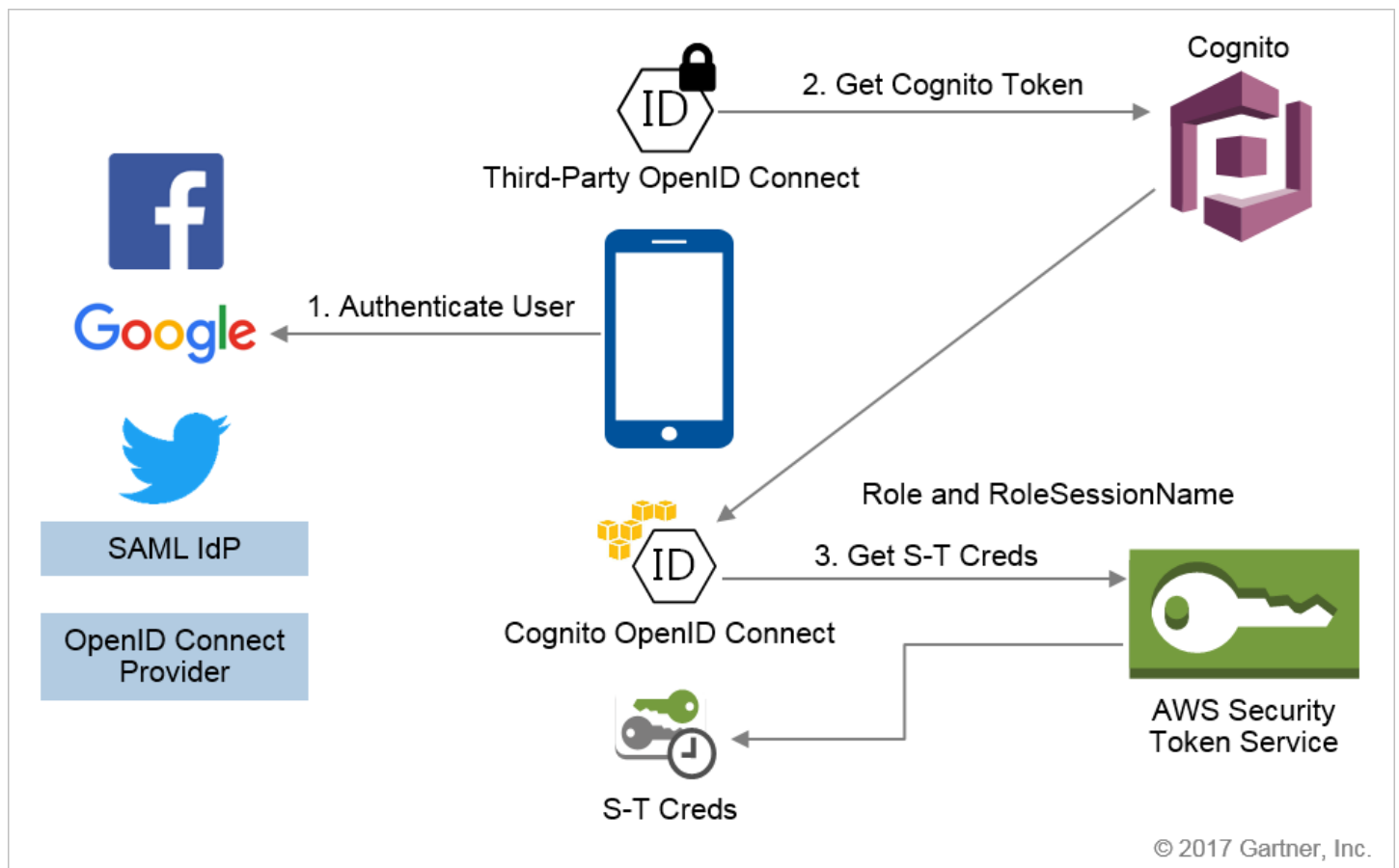
Amazon Cognito

In 2014, AWS introduced its Amazon Cognito user identity and data synchronization service. This offering added a new, OpenID Connect (OIDC)-compliant way to authenticate to AWS services (see Figure 2). Cognito is mainly marketed to mobile app developers as a way to:

- Provide social-media-based authentication to their apps
- Enable app data to be synced across different devices (such as phones and tablets)

As an access management approach for AWS, Cognito features some welcome advances in its approach to authentication and identity directories, compared with the prior approaches used in AWS IAM.

Figure 2. Third-Party Authentication to AWS Services via Cognito



IdP = identity provider; S-T creds = short-term credentials

Source: Gartner (February 2017)

However, the Cognito authentication approach is not without its drawbacks. As Figure 2 shows, the authentication process requires multiple "credential hops" to achieve access to AWS services. Moreover, it still relies on AWS's proprietary short-term credentials on the back end, requiring a complex, proprietary cryptographic hashing-and-signing process (in lieu of the simpler, standards-based approach of OAuth). The developer must use AWS-specific toolkits to be free of the cryptographic complexity.

User Pools

In 2016, AWS expanded the Cognito service by adding the User Pool, another AWS directory that can store user information and provide OIDC-based authentication. Organizations that implement the optional Cognito User Pool feature can use it to perform the user authentication portion of the process described in Figure 2. The Cognito User Pool is both an OIDC provider and a user directory, enabling organizations to employ User Pools for authentication, instead of a third-party service.

In many respects, the capabilities offered by Cognito User Pools are welcome advancements over AWS's previous IAM and federation features. For example:

- Unlike AWS's IAM user directory, whose data fields are rigidly defined and cannot be extended or customized, a Cognito User Pool has a fully extensible user schema.
- A Cognito User Pool acts as an OIDC IdP, rather than simply consuming OIDC-issued tokens and relying on third-party services to dispense them.

Active Directory Services

In the past two years, AWS has released several offerings that extend Active Directory services into Amazon Elastic Compute Cloud (EC2) virtual machine (VM) environments. These developments are significant, because migrating Windows-based applications to the cloud is a growing priority for many enterprises.

Increasingly, enterprises are migrating many of their on-premises applications to IaaS providers to gain benefits such as IT agility, cost optimization, improved alignment with subscription pricing, and elasticity (that is, the ability to grow and shrink infrastructure capacity as needed). Because many of these applications run in Windows environments, they often require Active Directory services to provide identity information, and Kerberos to enable authentication and authorization.

Providing Active Directory services to these applications once they move into EC2 may present some challenges. This won't be an issue if the applications are refactored for the cloud. In this case, IaaS-based directory and authentication services will be used. However, while such refactoring may be the long-term goal, many organizations are instead taking a more direct approach: They're rehosting Windows-based environments directly onto the cloud by running them within EC2. This approach necessitates virtual Active Directory services to provide user management, single sign-on (SSO) and authentication capabilities for the cloud-migrated Windows applications (see Figure 3). For more information on application migration, please see "[Decision Point for Choosing a Cloud Application Migration Strategy](https://www.gartner.com/document/code/299768?ref=grbody&refval=3620417)." (<https://www.gartner.com/document/code/299768?ref=grbody&refval=3620417>)

Figure 3. Application Migration From On-Premises

	On-Premises	Rehost	Refactor
File System Access	Servers	IaaS Virtual Machines	IaaS Services
Web Server			API Gateway
API Interface			
RDBMS			NoSQL or RDBMS
Kerberos (SSO)	Active Directory	Virtual Active Directory	SAML, OAuth and/or OIDC
User Directory			Cloud Directory

© 2017 Gartner, Inc.

RDBMS = relational database management system; SAML = Security Assertion Markup Language

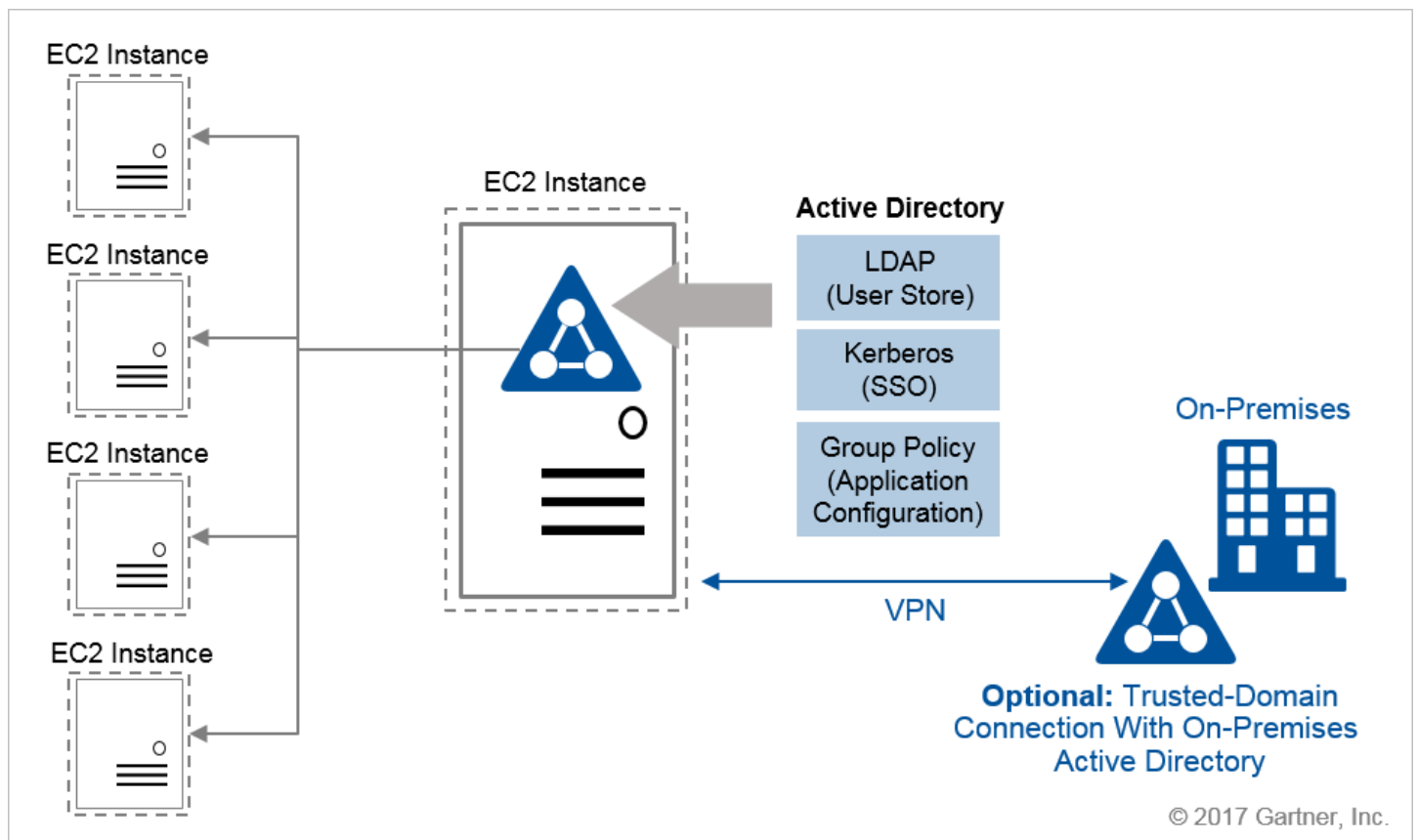
Source: Gartner (February 2017)

Past, Problematic Approaches

In the past, organizations have addressed the need to provide Active Directory services to their EC2-hosted applications by implementing an Active Directory forest running on EC2 instances (see Figure 4). This approach, however, poses the following challenges:

- The organization must manage and administer another Active Directory implementation, in addition to the on-premises one.
- The EC2 instance running the Active Directory domain controller must run constantly, creating additional computing expense.
- The AWS private networking and firewall components involved can be difficult to configure.

Figure 4. Prior Approaches: Running Active Directory on EC2 Instances



LDAP = Lightweight Directory Access Protocol

Source: Gartner (February 2017)

An additional challenge is that on-premises Active Directory services cannot be accessed from EC2 — unless a trusted VPN connection is added between the two Active Directory environments (see the right side of Figure 4). However, this option introduces a new set of network challenges related to potential VPN connectivity and latency issues. Overall, the EC2-hosted Active Directory option doesn't work well, and better approaches have emerged to replace it.

Recent AWS Offerings

In the past two years, AWS has introduced three new offerings for virtual Active Directory services. These offerings, which fall under the AWS Directory Service umbrella, address some of the issues raised above. First, in 2014, AWS introduced AD Connector, which enables on-premises Active Directory environments to provide directory service support for AWS products such as Amazon EC2 or Amazon WorkSpaces. AD Connector provided a useful step forward by extending existing Active Directory services into the cloud. However, it didn't address the need to provide cloud-native Active Directory services for Windows application environments that migrated onto EC2.

Next, AWS released two offerings that provided cloud-based Active Directory support for AWS services. The first, Simple AD, was released in 2014. Simple AD is an Active Directory-compatible directory service based on the Samba software standard. Although Simple AD accomplishes the

basic goal of providing cloud-based directory services, the offering is, as its name suggests, a relatively simple one, which lacks many of the advanced features of a native Active Directory environment.

The big step forward came with the 2015 release of AWS Directory Service for Microsoft Active Directory (Enterprise Edition) — aka AWS Microsoft AD. Unlike Simple AD, AWS Microsoft AD provides a real Active Directory solution that supports native Microsoft tools, PowerShell and forest trust. Architecturally, the AWS service is a single-forest Active Directory, and AWS delegates management to the AWS tenant.

IaaS Active Directory services are nascent, so application incompatibilities can exist. Also, these virtualized services are tailored to application migrations. Moreover, AWS Microsoft AD is not designed — at least for the foreseeable future — to support the rehosting of a typical enterprise's complete Active Directory environment.

For more information on the AWS Active Directory services mentioned above, see the AWS's Active Directory Capabilities discussion in The Details section.

Amazon Cloud Directory

In early 2017, AWS introduced the Amazon Cloud Directory service. AWS uses the technology behind Cloud Directory for both Cognito User Pools and AWS Organizations (covered in separate sections within this document).

AWS Cloud Directory is a hierarchical, general-purpose data store. While AWS provides schemas for people, organizations and devices, Cloud Directory can be used to store information about anything. Customers can create custom schemas encoded in JavaScript Object Notation (JSON), and then upload those schemas. The directory enforces object schemas via "facets," which function like LDAP's object classes.

Although Cloud Directory can store information about people, it is not plugged into any of the AWS access control mechanisms, and it lacks user authentication capabilities. However, customers could use Cloud Directory for a custom application, if they built authorization and authentication services in front of Cloud Directory.

Access Control

Access control provides the necessary tooling to enforce organizational access policies. Access control tools for AWS include IAM roles, policies and IAM groups. See The Details section for a discussion of AWS IAM groups, which are applicable only to the IAM user type.

Policy Complexity

AWS provides a rich, JSON-based syntax for granting entitlements to IAM groups and IAM roles for use with federated users. The syntax enables extreme granularity. These entitlements are contained

in a policy statement, which is attached to groups or roles to grant or deny access to specific resources. However, the rich syntax comes with a price: AWS's policies are complex and difficult to author.

The biggest problem with policy statements, per Gartner clients, is that they are often difficult and time-consuming to create. For the time being, organizations must manually create or edit the JSON-encoded syntax to get a policy statement that aligns with their security goals. Many organizations will export the policy statements to a third-party JSON editor to work on them. The AWS Management Console provides a policy generator tool that can assist. However, the GUI lacks pick list functionality — a necessary capability to simplify policy creation.

Creating policies to provide granular access to resources is nearly impossible without extensive troubleshooting and JSON editing. Gartner clients frequently spend eight hours or more creating a single policy statement and validating a single IAM role. Further, it is unclear afterward whether a user may have been given excessive access to resources. To help address these issues, AWS provides a policy simulator to facilitate policy troubleshooting. Even with this feature, however, policy creation and subsequent validation of access rights can be a time-consuming chore.

Organizations have told Gartner that they enabled "wide open" access because they could not create an AWS access policy that did not also lock out authorized users.

Managed Policies

In 2015, AWS introduced the Managed Policies feature. This feature provides a repository for access policies, including some predefined policies from AWS. The repository also delivers version control capabilities. Before Managed Policies, organizations crafted authorization policies and attached them directly to IAM groups, IAM roles or IAM users. This approach made policy reuse difficult and often resulted in access control errors.

AWS IAM Role

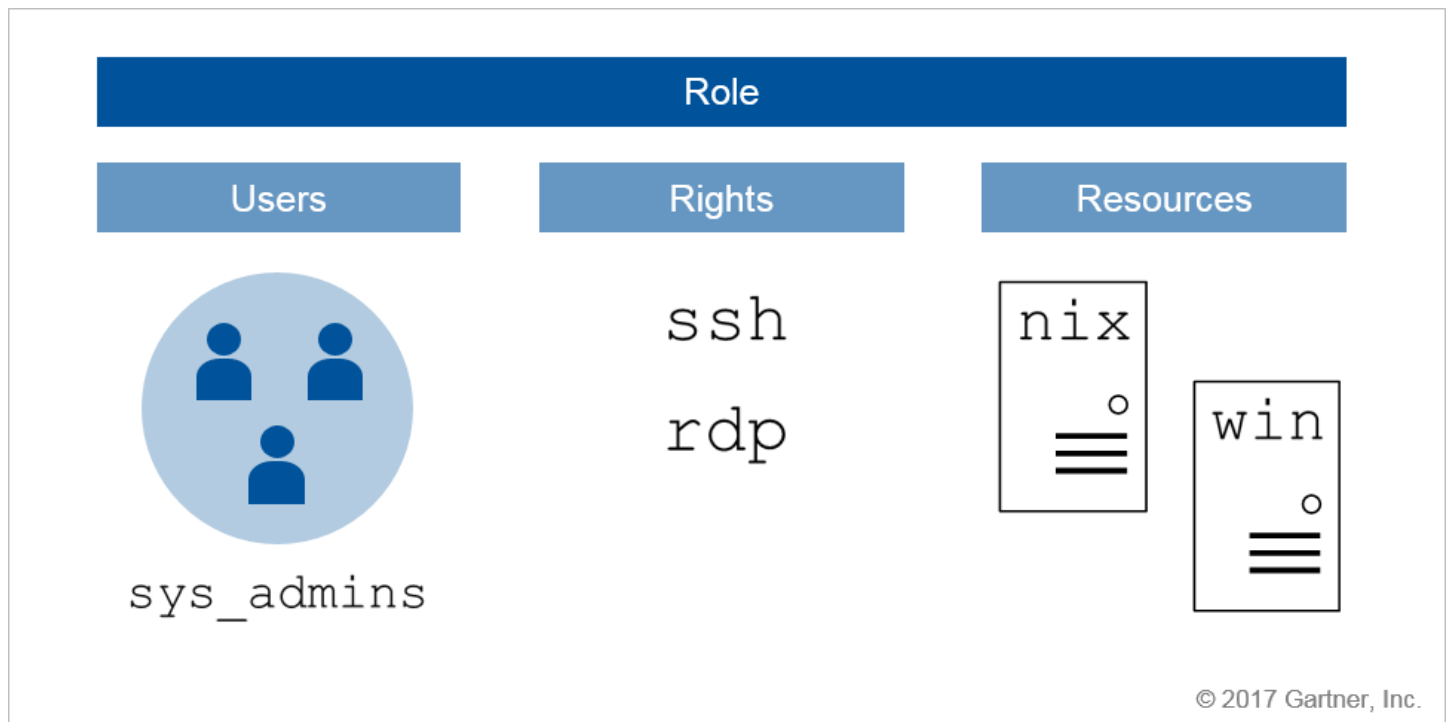
Scalable identity management requires a conduit to bind groups of users to resources. This conduit is frequently called a role. A role definition includes three components:

- **Subject:** The users who will access the specific resource. The subject definition is frequently a group.
- **Object:** The resources that will be accessed by the subject.

- **Access rights:** The specific rights that the user will have with respect to the resource or service.

Figure 5 provides an example of a typical role structure.

Figure 5. Typical Role Structure



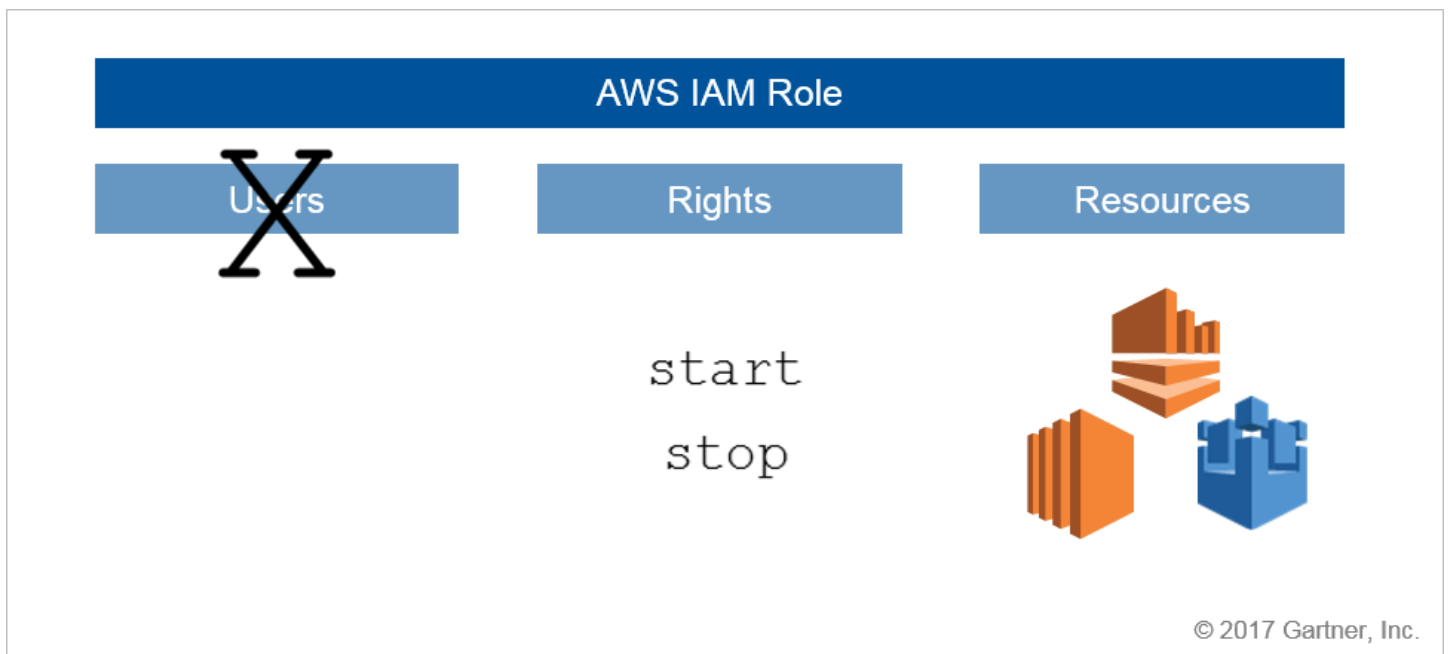
Source: Gartner (February 2017)

Roles deliver scalability across two dimensions:

- Users can be added to or deleted from the role to gain or lose access to the resources. There is no need to directly assign access rights to individual users.
- Access rights can be removed from a single role, rather than removing the rights from thousands of individual users.

AWS's IAM role structure provides challenges for authorization and governance. First, the AWS IAM role structure does not include a subject definition — that is, users or groups that have access to the resource (see Figure 6). Thus, an access certification review within AWS isn't possible. For more information on addressing these challenges, see the Identity Governance section.

Figure 6. The AWS IAM Role Lacks a User Definition



Source: Gartner (February 2017)

AWS Organizations

In late 2016, AWS introduced the Organizations feature, which was in preview status as of the publication date of this document. Many AWS customers have hundreds or even thousands of tenants. The Organizations feature establishes a new "master account" that presides over all of a customer's AWS tenants, including the AWS root account (see Figure 7). AWS Organizations enables the creation of additional AWS tenants as well as consolidated billing.

Figure 7. AWS Organizations and Privilege Delegation

Source: Gartner (February 2017)

From an access control perspective, AWS Organizations provides the critical capability of limiting privileges within each tenant — including the AWS root account. AWS Organizations uses a new policy construct — the service control policy — to restrict privileged access on the tenants. These policies are applied at the organizational unit (OU) level and impact all AWS tenants within the OU. OUs may be nested, enabling multiple levels of policy application. In nested configurations, each sub-OU inherits the SCPs from the OU above it.

Within each tenant, the SCP restricts the privileges of the AWS root account — and, in fact, of all users. The root account may delegate privileges to IAM users, groups and roles as usual — but it cannot delegate privileges that are prohibited by the SCP.

Authentication

Some AWS authentication methods are designed for access via a web browser, while other authentication methods are utilized for API access. This section provides an overview of authentication from a browser versus API perspective. For a summary of authentication methods by AWS user type, see the Authentication Methods discussion in The Details section.

Multifactor Authentication Options

AWS provides two multifactor authentication (MFA) options:

- Open Authentication (OATH)-based one-time passwords (OTPs) can be used by AWS IAM users.
- SMS OTPs can be used by both AWS IAM and Cognito User Pool users.

Neither option is modern or recommended from an identity perspective. SMS as an MFA method should be leveraged only for nonsmartphone users. There are better alternatives from both the security and the usability perspectives. Mobile push technology has eclipsed OTP as the default commercial MFA system because it is:

- More user-friendly
- Less costly
- More secure

Proprietary API Credentials

To grant access by non-IAM users, AWS introduced a proprietary federated-access method in 2011, and still offers this original method today. This proprietary method leverages an AWS IAM user's long-

term credentials as the basis for issuing short-term credentials to temporary-access users (see Figure 19 in The Details section). Enabling this capability requires developers to first create an "identity broker," which maps a given employee identifier to a specific set of AWS IAM user credentials (see the Federation User Access – Proprietary Method discussion in The Details section). To authenticate to that broker, enterprise users can employ their Active Directory credentials. The identity broker then issues a short-term credential to the federated user.

AWS now also offers newer, standards-based approaches to federated access. However, even some of these newer approaches, such as Cognito access via the OIDC standard, still employ the mechanism of issuing AWS short-term credentials on the back end. Thus, the standards-based approaches that employ this mechanism share some of the same concerns as the older federation method described above.

While the identity broker mechanism accomplishes the goal of enabling casual users to access AWS services, it poses several drawbacks. First, it is proprietary and somewhat complex. Moreover, because an IAM user's long-term credentials are used to obtain short-term ones, these long-term credentials are stored in the identity broker as part of the access process – which means they are subject to attack. When possible, SAML or OIDC should be used in lieu of this proprietary federation method (see the Use Standards-Based Federation – Not AWS's Proprietary Federation discussion in the Guidance section).

Federation

This section assesses AWS's federation capabilities, in chronological order of availability.

Proprietary

In 2011, AWS implemented a proprietary mechanism that required an IAM user to delegate access to an ephemeral user – in this case, the "proprietary-federation" user type (see Note 1). With this implementation, the burden of secure application development resided with the organization, which had to find a way to secure the IAM user's long-term credentials and authenticate users outside the AWS environment. In addition, the customized application development was hard to get right from an authorization perspective. AWS still supports this proprietary mechanism.

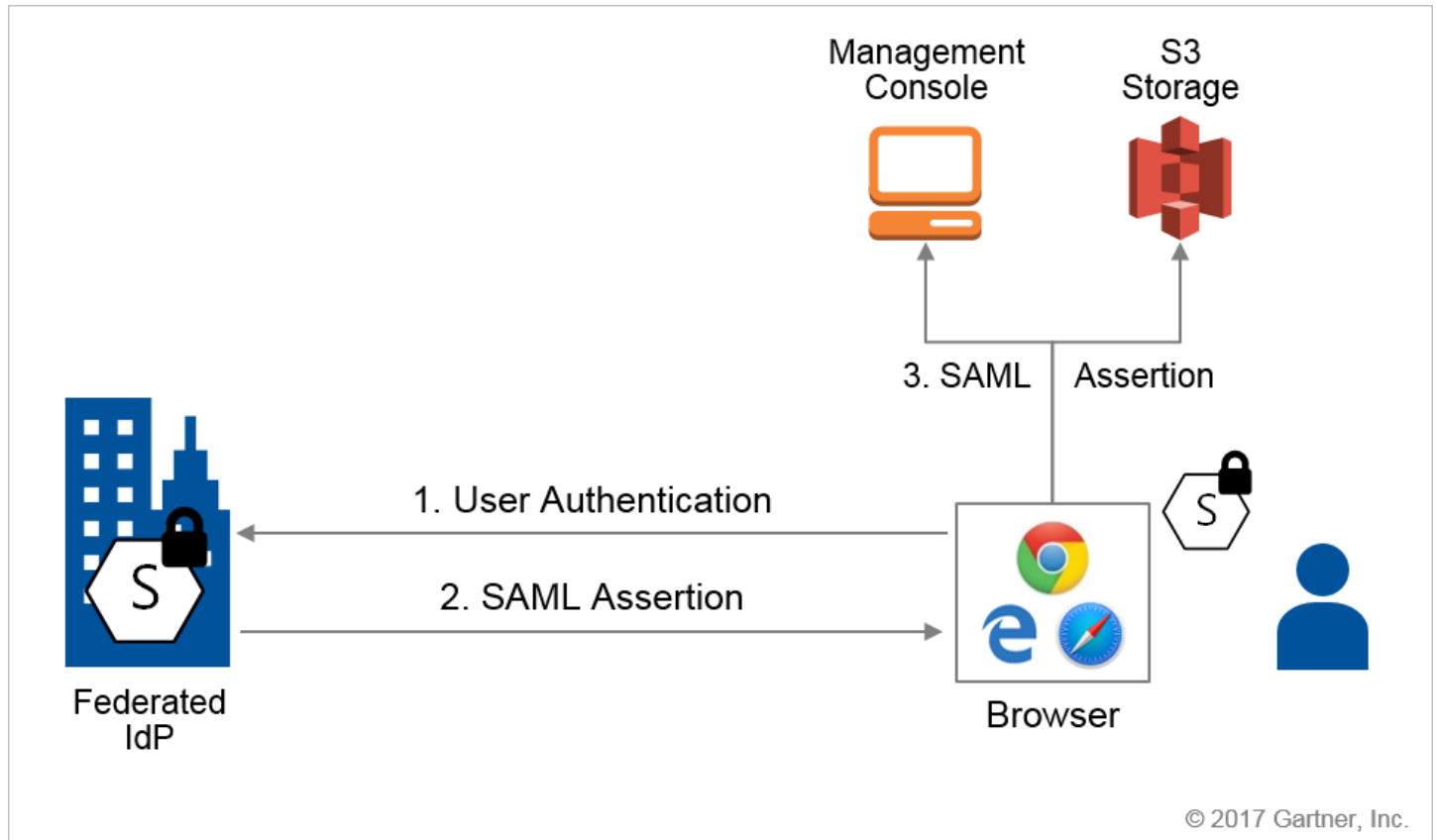
Security Assertion Markup Language

AWS supports SAML as a service provider (SP) to enable organizations to provide SSO into AWS. The federated SP capability exists within AWS Security Token Service (STS) and Amazon Cognito.

In late 2013, AWS improved its approach by introducing standards-based federation via SAML, enabling organizations to connect to AWS using commercial federation solutions like Microsoft's Active Directory Federation Services, Ping Identity's PingFederate or Okta's IDaaS product (see Figure 8). These federation protocols are employed with the "standards-based-federation" user type (see Note 1).

With SAML support, AWS provided a solution to many of the challenges associated with its proprietary federation mechanism (though the user still did not exist in any AWS user directory – see Figure 12 in the Identity Governance section). The SAML assertion can be used to access the web-based Management Console, or to access AWS resources that are reachable via a web browser (for example, Amazon S3 data storage or Amazon DynamoDB NoSQL databases). The SAML assertion includes a specified AWS IAM role.

Figure 8. SAML Authentication to AWS via STS



Source: Gartner (February 2017)

While AWS supports the SAML assertion in API-based use cases, most commercial federation IdP implementations support browser-based interactions only.

OpenID Connect

AWS supports both OIDC relying party (RP) and OIDC provider functionality. The functionality is included with AWS STS and Amazon Cognito.

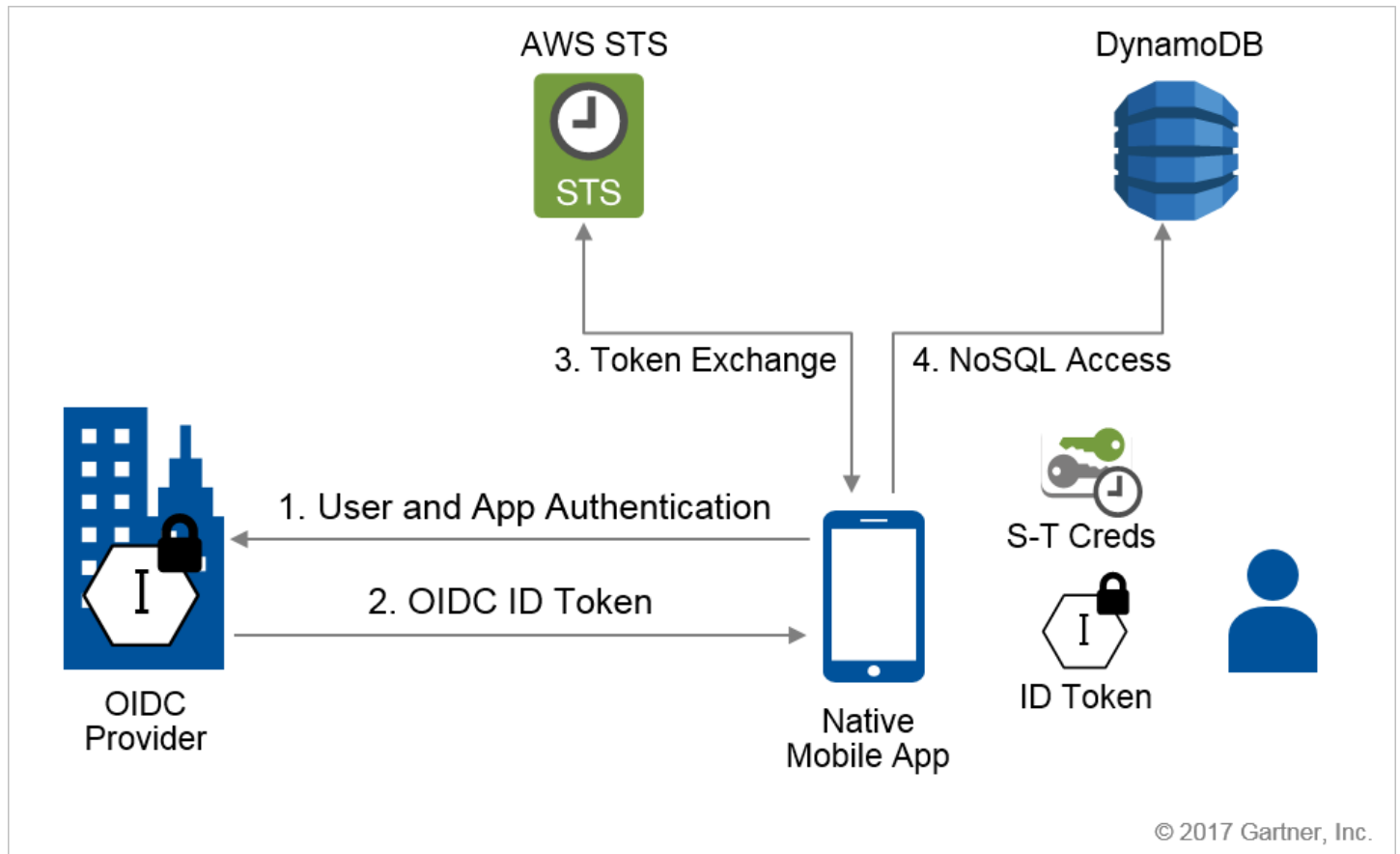
Relying Party

In 2014, AWS introduced OIDC RP support, enabling organizations to leverage third-party OIDC providers to connect users to AWS services (see Figure 9). In the flow, the user first authenticates to the OIDC provider, and then receives the OIDC ID token. The application, on behalf of the user, presents the OIDC ID token to the AWS Security Token Service. The AWS STS issues a set of

proprietary short-term credentials that the application uses to access resources via the AWS API on behalf of the user. As with the SAML assertion, the OIDC ID token contains an AWS IAM role.

Later in 2014, AWS introduced Cognito (without User Pools), which also has OIDC RP functionality.

Figure 9. OpenID Connect Integration With AWS



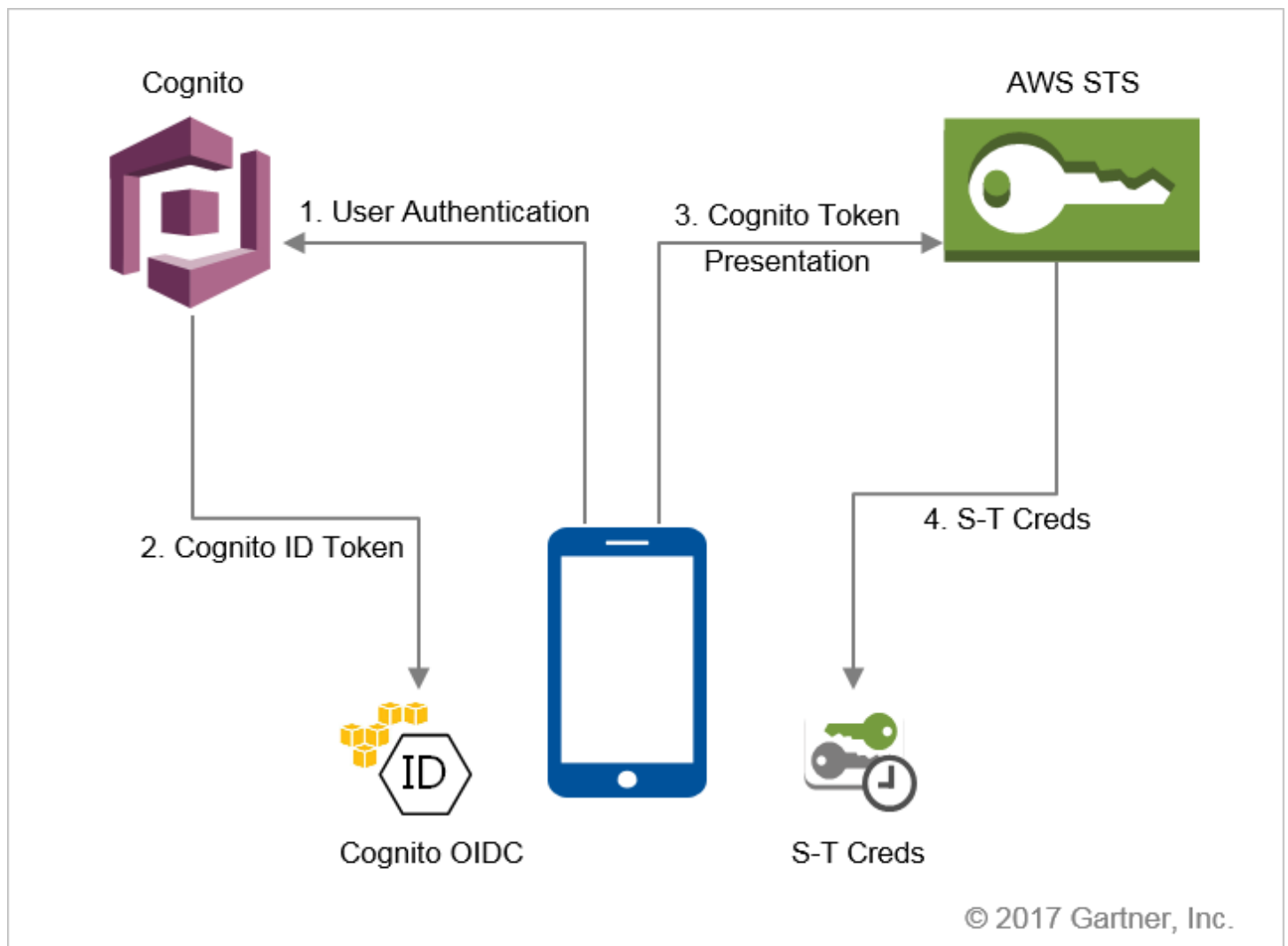
S-T creds = short-term credentials

Source: Gartner (February 2017)

Provider

When User Pools were introduced for Cognito, AWS added OIDC provider capabilities (see Figure 10). For more information on OpenID Connect, see "[Modern Identity and APIs: Mobile, OpenID Connect, OAuth, JSON and REST.](https://www.gartner.com/document/code/277246?ref=grbody&refval=3620417)" (<https://www.gartner.com/document/code/277246?ref=grbody&refval=3620417>)

Figure 10. Cognito User Pool OIDC Provider



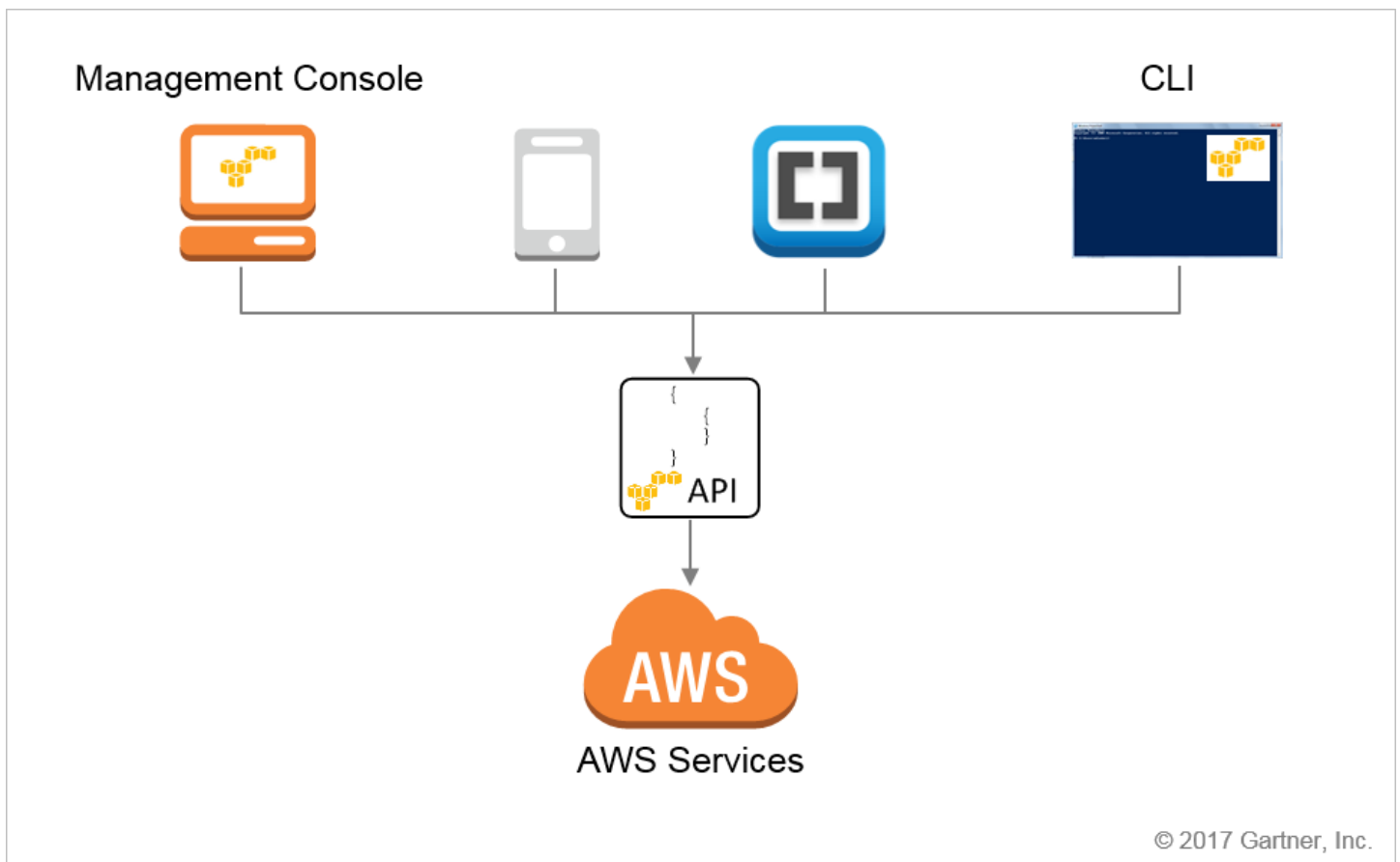
S-T creds = short-term credentials

Source: Gartner (February 2017)

Administration

AWS has always been an "API-first" organization. After developing or enhancing a stable, public API, AWS uses the API to build the command line interface (CLI) and the web-based Management Console (see Figure 11). This approach creates harmony across the three interfaces — a significant strength that supports programmatic interaction with all administrative functions.

Figure 11. AWS's API-First Philosophy



Source: Gartner (February 2017)

The interfaces serve different user constituencies. The web-based Management Console provides a simple interface for administrators to configure the initial AWS tenant and perform ongoing administration. The API can be used by developers to build scalable, repeatable operations — and to enable access to AWS services (for example, S3 and DynamoDB). The CLI allows operations personnel to script more complex operations.

API

The AWS API is strong because it provides a single interface that works with all the AWS services — something that is not possible with other IaaS providers. Even better, the API uses JSON for data representation. However, AWS's authentication mechanisms for API access can be complex and proprietary. Further, the API utilizes proprietary authorization and does not support OAuth.

One example is the mechanism used by AWS IAM users to access the API. Authentication via APIs is granted based on an IAM user's long-term credentials (represented as `AccessKeyId` and `SecretAccessKey` parameters) — codes that are the programmatic equivalent of a username and password. These long-term credentials enable an application, on behalf of the user, to access protected services through the API.

During a federated session using AWS's proprietary method, an IAM user's long-term credentials are employed to obtain short-term credentials on behalf of the federated user. These sets of credentials are stored as part of the access process – which means they are subject to attack. Fortunately, the standards-based-federation user type leverages SAML and OIDC to procure short-term credentials, eliminating the need to store an IAM user's long-term credentials.

If OAuth were used instead of AWS's proprietary methods, API requests could be handled using the OAuth token. The OAuth token would eliminate the need for an AWS software development kit (SDK) and would provide a familiar protocol for developers.

The AWS IAM role would be ideal to use as a scope in OAuth requests. The mapping of roles to scopes would provide a logical way to bridge the current AWS IAM functionality with the standards-based OAuth access control protocol.

Identity Governance

Access control and governance work together. The former provides the necessary tooling to enforce organizational access policies. The latter delivers a mechanism for reviewing access policies to ensure that they meet security policies and compliance mandates.

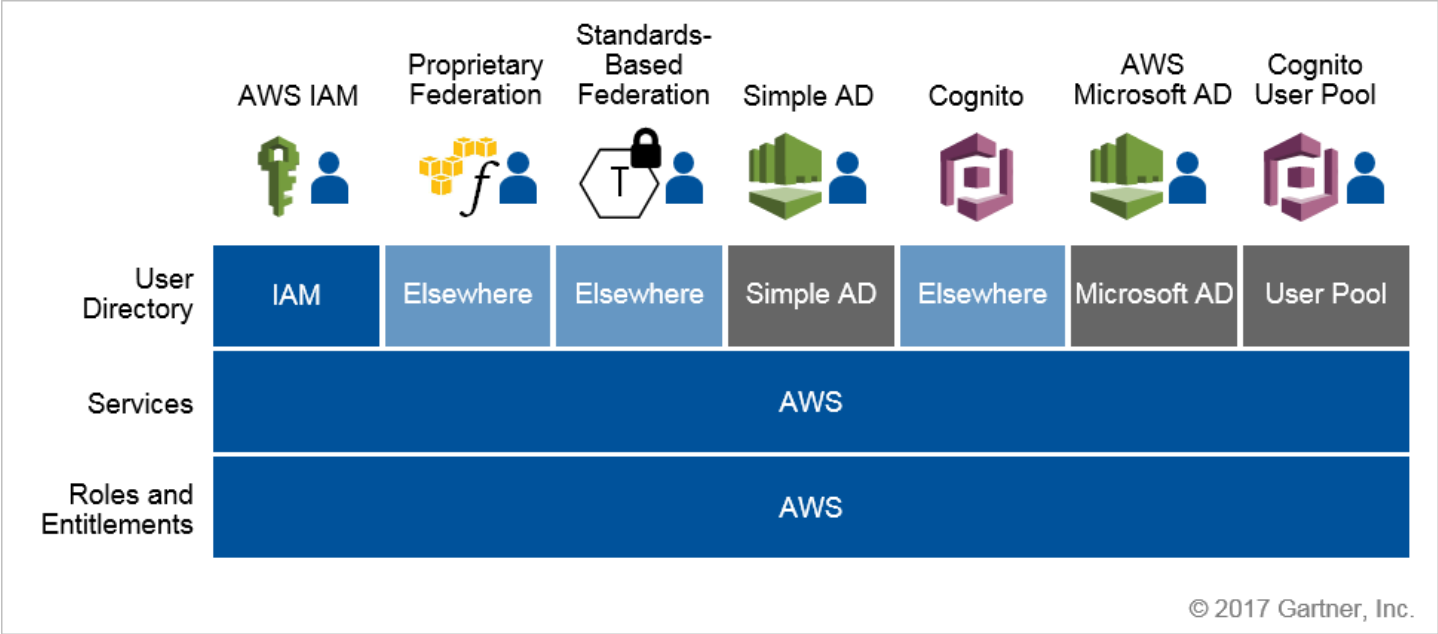
Access certification is the act of reviewing and attesting to whether appropriate access to critical resources has been given to the right people. This can involve analyzing everything from user identities to role assignments, as well as access delegation. Good governance is essential for:

- **Compliance:** Certain information may be protected under regulatory rules. In these cases, documenting information about specific users and their access rights may be required for auditing purposes.
- **Security:** Information on who can access what is needed to effectively manage the system and ensure "least-privilege" access.
- **Cost reduction:** If certain services are being accessed by users for purposes that don't have a strong business need, this can add to the costs of running AWS services.

There are two challenges for AWS identity governance. One challenge is associated with the multiplicity of user directories, some of which are not part of AWS (see Figure 12). For example, the directories associated with the proprietary-federation, standards-based-federation and Cognito user

types are not part of AWS. Therefore, access rights for these users cannot be reviewed within AWS. While the AWS IAM, Simple AD, AWS Microsoft AD and Cognito User Pool types do exist in AWS, they reside in separate user directories with separate namespaces. Another challenge is that the AWS IAM role definition does not include a subject (the "who"). See the Access Control section above for additional information.

Figure 12. AWS User Types and Associated Directories



Source: Gartner (February 2017)

Approaching the Governance Challenge

For the most part, current identity governance and administration (IGA) products don't support the AWS platform (the exception is Saviynt, which is developing a connector for AWS). Governance is a challenging task, but organizations can retrieve the data necessary for access rights analysis. There is one requirement: The organization must have access to the directory where the user information is stored. For example, if the organization enables partner access to AWS and doesn't have access to the partner's user directory, it will not be able to review these users' access rights.

The next two sections discuss two approaches to governance. For the sake of illustration, these two methods assume that the user directory is on-premises Active Directory.

Active Directory Group

In this approach, the organization's federation IdP provides the critical mapping of Active Directory groups to AWS IAM roles via the SAML assertion (see Figure 13). Access policies are bound to AWS IAM roles. Each policy contains specific access rights. To determine Active Directory user access to AWS services, perform the following tasks:

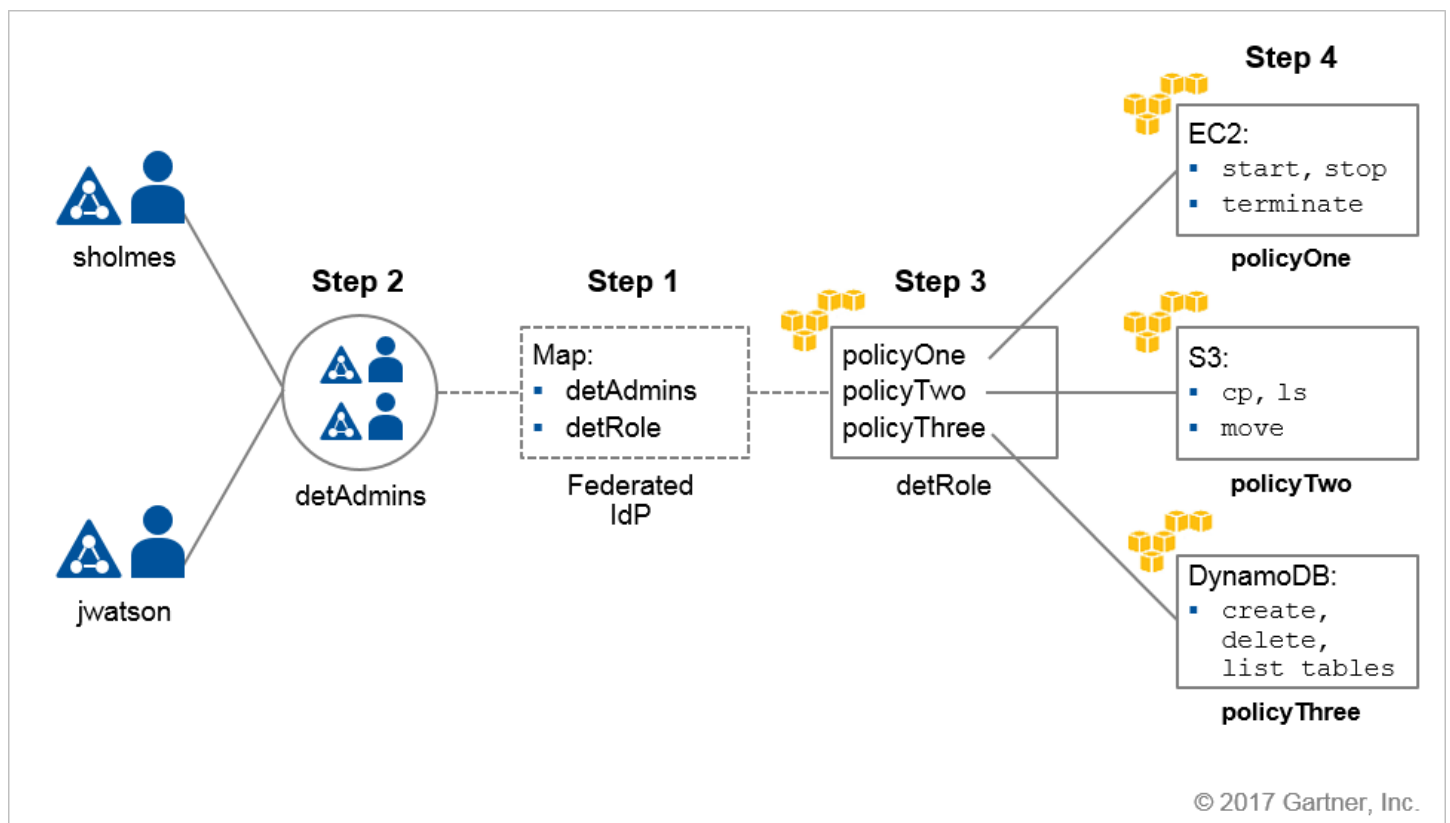
1. Review the federation IdP's authorization policies to understand the mapping of Active Directory groups to AWS IAM roles. In this example, the federated IdP creates SAML assertions for users of

the Active Directory "detAdmins" group, and then specifies the AWS IAM role "detRole" as an attribute in the SAML assertion. When the user arrives at AWS with the SAML assertion, the user will have access to services based on the AWS access policies associated with the IAM role.

2. Retrieve the list of users associated with each Active Directory group.
3. Retrieve the list of AWS access policies associated with each AWS IAM role.
4. Parse the JSON-encoded access rights in each policy that is bound to the AWS role.

Afterward, you can link Active Directory users and groups to AWS access rights, and then leverage a data analytics tool to analyze these privileges.

Figure 13. Mapping Active Directory Groups to AWS IAM Roles



Source: Gartner (February 2017)

Active Directory User Attribute

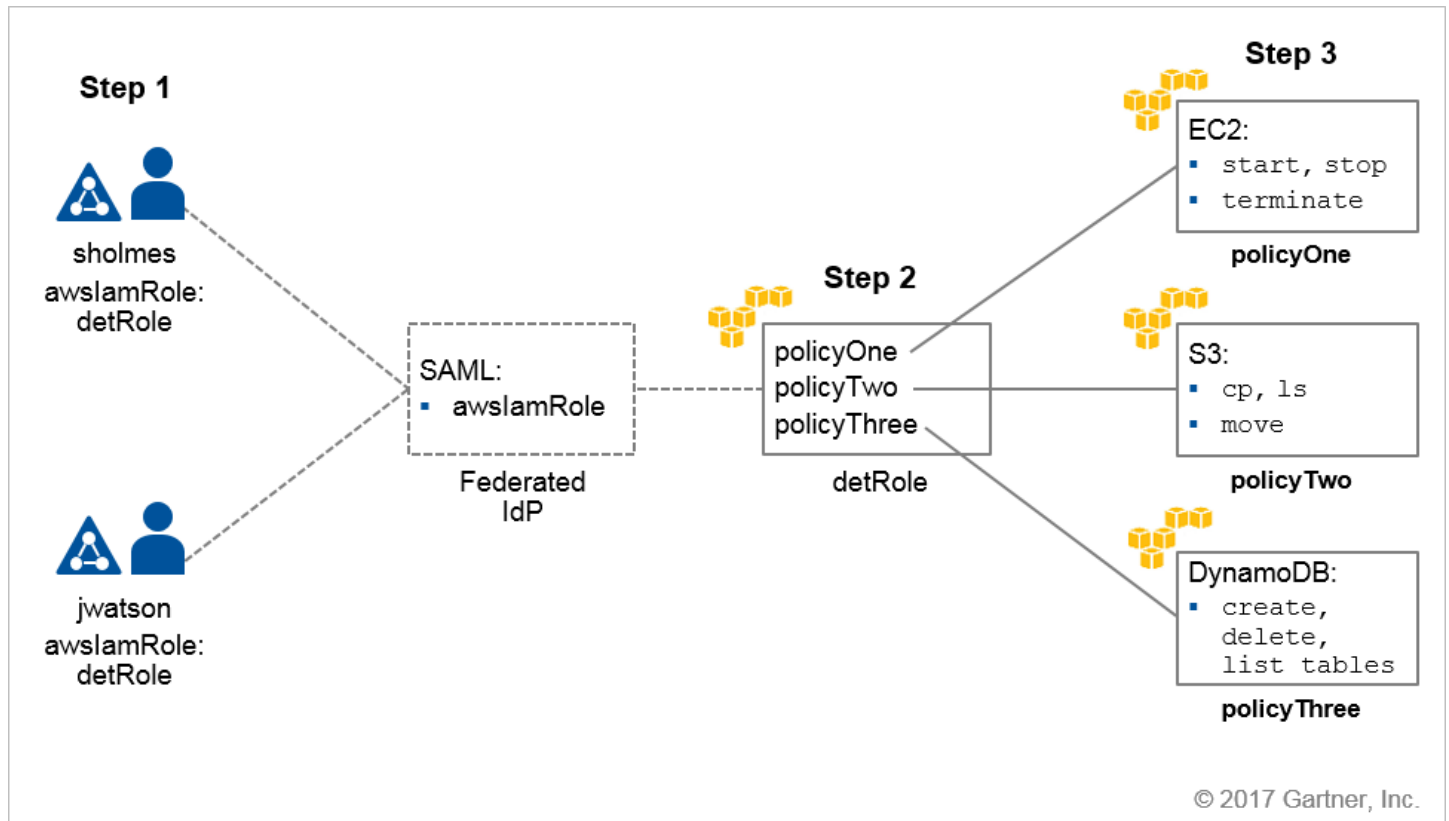
Instead of employing an Active Directory group to bind AWS access rights to users, organizations can create (or borrow) a multivalued user attribute in Active Directory to store AWS IAM role names (see Figure 14). The benefit of this approach is a simpler federated IdP configuration. The IdP copies the IAM role from the Active Directory user attribute into the SAML assertion, thus condensing the process into three steps:

1. Retrieve the list of Active Directory user objects with AWS IAM roles.

2. Retrieve the list of AWS access policies associated with each AWS IAM role.
3. Parse the JSON-encoded access rights in each policy that is bound to the AWS role.

However, with this approach, managing the Active Directory users' access to AWS services becomes more challenging, compared with adding or deleting users from an Active Directory group.

Figure 14. Mapping Active Directory User Attributes to AWS IAM Roles



Source: Gartner (February 2017)

Strengths

- **Comprehensive API:** AWS's comprehensive, unified API supports programmatic interaction with all administrative functions. This enables external services to be used in conjunction with AWS to help bridge enterprise and AWS identity environments, and to overcome AWS IAM challenges.
- **Active Directory offerings for EC2-hosted Windows environments:** The AWS Microsoft AD service provides a useful, cloud-native directory environment for EC2-hosted Windows environments.
- **Amazon Cognito:** Cognito provides a modern, standards-based approach for accessing AWS. A Cognito User Pool is extensible, and can act as an OIDC IdP and a federated SP.
- **Managed Policies:** This feature enables the sharing of access policies across users and resources. Additionally, it includes version control, which allows organizations to test new policies and revert to a prior version of a policy if necessary.

- **AWS Organizations:** This service enables customers to organize AWS tenants from an access control perspective — as well as limit the privileges within each of these tenants.

Weaknesses

- **Fragmented identity environment, with many user types and access methods:** This situation makes IAM administration complex, while increasing the risk of data breaches and DoS attacks.
- **Difficulty determining who has access to what:** Many AWS user types don't exist in any AWS directory. In addition, the roles used to assign access privileges to these users don't capture data on all the users assigned to such roles. This can make it difficult to track user access privileges for governance and audit purposes.
- **Labor-intensive policy creation and management:** AWS uses a proprietary syntax for granting entitlements through policy statements, which can be difficult and time-consuming to create. Many organizations tell Gartner they spend significant amounts of time creating policies and granting access rights.
- **Complex, proprietary API mechanisms:** AWS uses proprietary methods in its APIs. These methods can create problems due to their complexity.
- **Lack of a SAML IdP:** Although AWS can *consume* SAML assertions, it cannot act as a SAML IdP to *provide* these assertions. Instead, the assertions must be created by a third-party IdP service. An AWS federation service would be valuable, saving organizations from having to create and manage SAML IdPs for their applications.

Guidance

Gartner offers the following recommendations for technical professionals involved in planning or executing IAM for AWS services.

Pressure AWS to Improve Its IAM Capabilities

Gartner recommends that current and prospective customers of AWS use their market influence to pressure AWS into making improvements that address the current shortcomings in its IAM capabilities and features. Changes and additions that enterprises should push for include:

- **A single, extensible user directory:** Instead of the current IAM user directory, which doesn't include all user types, AWS should provide an optional unified user directory. This user directory should be extensible, so that fields can be added for additional user information. This capability will allow customers to better leverage their investment in AWS by extending the user directory database to additional purposes. Ideally, AWS should implement a System for Cross-Domain Identity Management (SCIM) 2.0 interface for standards-based user provisioning to the directory.

- **Simpler policy creation and management:** Currently, the process of defining access rights involves editing JSON, which is time-consuming and cumbersome. Customers should demand improvements that make policy management simpler. Such improvements include:
 - GUI pick list functionality
 - The ability to capture the identities of — and data on — all users who have been defined to IAM roles

The goal is to keep administrators out of a JSON editor for 90% of all policy creation.

- **Modern access control for the AWS API:** In place of the current system of proprietary API access, customers should pressure AWS to provide a more modern, standards-based approach — namely, the use of OAuth access tokens to control API-based access.
- **Modern MFA options:** Currently, AWS supports OTP and SMS authentication. Both authentication methods have lost their luster from a security and usability perspective. The OTP authentication method is being eclipsed by mobile push authentication, which delivers better security, reduced costs and greater usability. SMS authentication received scrutiny in 2016 from the National Institute of Standards and Technology (NIST). AWS should offer mobile push authentication — and evaluate the applicability of solutions based on Fast Identity Online (FIDO) 2.0.
- **A Cognito OpenID Connect provider for external use:** Cognito currently provides an OIDC provider for identities defined in User Pools. However, the OIDC capabilities can be used only for AWS access. AWS should enable Cognito's provider to support external use cases. This support would offer the following benefit: Organizations could leverage the provider to administer authentication and SSO services to third-party applications from AWS.
- **Better pricing for Active Directory services:** It is likely that AWS Microsoft AD will become the default AWS Active Directory service soon, and that AD Connector and Simple AD will become deprecated over time. However, AWS Microsoft AD is prohibitively expensive, even for small test environments. A pricing mechanism that supports reasonable pricing at small volumes would encourage more application migration to AWS.
- **Standards-based federation for AWS IAM users:** AWS currently supports SAML- and OIDC-based functionality for ephemeral user types. This functionality would be valuable for the AWS IAM user type as well. For access to the web-based Management Console, SAML support would enable an organization to authenticate with a near-infinite variety of MFA options while providing SSO for the administrator. With API access, administrators would be authenticated according to organizational policies via an OIDC ID token — removing the need to store long-term credentials inside the application. (Note that unattended services would still require the storage of long-term IAM user credentials.)

- **A subject for the AWS IAM role:** The principal conduit for AWS access control is the AWS IAM role, which is incomplete because it is missing the subject — the "who" part of a standard IAM role. Thus, organizations cannot perform identity governance tasks, such as access certification and separation-of-duties analysis, by auditing access rights within AWS. These tasks are essential for organizations under any compliance mandates — or for organizations concerned about data breaches or DoS attacks. AWS should modify its IAM role to support a subject attribute. While enhancing the AWS IAM role will not fix the problems associated with reviewing the access rights of ephemeral users, it will enable organizations to audit the access rights of users who exist in an AWS directory.

Use AWS Privilege Delegation to Protect Data and Services

Because the AWS root account has global access within AWS, it is critical to restrict the use of this account to:

- Limit the amount of damage that could be done
- Provide better tracking of who is making policy or access control changes

This account is a prime target for intruders, who could use it to breach confidential data and cause DoS attacks.

Rather than using the AWS root account for ongoing administrative tasks, use it to create IAM groups with administrative privileges. Privileges will be delegated to AWS IAM users who belong to these groups. When assigning access rights to these groups, follow a least-privilege approach to AWS entitlements: Determine upfront what users need to do within the AWS environment, and then set up policies that enable these users to perform only those specific tasks. When establishing policies for individual AWS resources, identify the users who need to access the given resource, and grant only the minimum set of privileges to those people. Understand how the AWS runtime evaluates IAM policies and determines access entitlements, so that you can better formulate those policies to operate as intended.

The AWS root account should be an emergency account only, and knowledge of its password should be strictly controlled. Controls can be implemented to force the use of MFA with the AWS root account, as well as to delete the access keys, thus preventing the programmatic use of the account. In addition, privileged access management (PAM) offerings can be used to lock down the AWS root account. For example, by employing a PAM solution, the AWS account password could be secured and made available only in times of emergency:

1. The password would be checked out under controlled circumstances.
2. Password usage would be closely monitored.

3. Then, the password could be changed upon check-in if necessary.

The AWS Organizations feature is nascent, but provides important privilege delegation functionality, including restricting the privileges of the AWS account within tenants. Enterprises with multiple AWS tenants should evaluate this feature as part of their resource protection goals.

Use MFA for Privileged Users

In cases where specific AWS IAM users have access to sensitive data or resources, provide an additional level of assurance and control by configuring MFA for these accounts. Require these users to provide a unique OTP in addition to their normal credentials to gain the necessary access.

Avoid Creating Your Own Active Directory Infrastructure Inside EC2

When attempting to migrate a Windows-based application to EC2, you may be tempted to install Active Directory inside the same EC2 instance as the application, in order to provision directory services for that application. However, you should avoid this approach because it will likely pose drawbacks, including:

- Increased effort to manage the EC2-based Active Directory implementation, including dealing with patches, upgrades and the security configuration
- Additional computing costs associated with operating the Active Directory EC2 instance, which must run continually
- Challenges associated with IaaS virtual network configurations

In some cases, there may be a compelling reason to run an Active Directory domain controller inside Amazon EC2. For example, the reason could be to provide temporary backup for an on-site Active Directory implementation (intended for use only when the on-site implementation suffers an outage or is taken down for maintenance). In most cases, however, the challenges outweigh the benefits. Instead, organizations should use the AWS Microsoft AD service to provide directory services to cloud-migrated applications.

Align Your Expectations to AWS's Active Directory Service Capabilities

The AWS Microsoft AD service is a powerful tool for organizations looking to migrate Windows-based applications to AWS. Organizations should thoroughly test applications before migrating them to AWS, as incompatibilities exist with applications that previously functioned with an on-premises Active Directory environment.

Additionally, many organizations have expressed an interest in migrating their entire Active Directory infrastructure to the cloud. The AWS Microsoft AD service — for now, anyway — is not a suitable environment for migrating most on-premises Active Directory environments.

Use Standards-Based Federation — Not AWS's Proprietary Federation

Technical professionals should look for ways to leverage standards-based identity offerings to provide identity services for AWS access, and to overcome weaknesses in AWS's proprietary implementation. Such external providers include MFA services and SAML and OIDC identity providers. The benefits that can be achieved using these third-party tools include higher scalability, better security and usability, and faster implementation times.

Improve Scalability by Leveraging AWS Managed Policies

With AWS's Managed Policies, organizations can manage a common set of access rights in one location across multiple groups and roles. This feature makes AWS's policies easier to manage, because administrators don't have to make copies for every role and group sharing the same common entitlement set. It also reduces errors by providing fewer points of control for policy management.

Avoid New Simple AD and AD Connector Deployments

As AWS has launched new regions, only the AWS Microsoft AD directory service has been made available — Simple AD and AD Connector are not present. This may be an indication that AWS views AWS Microsoft AD as its strategic direction moving forward. Gartner anticipates that Simple AD and AD Connector will become deprecated over time. Therefore, companies should avoid Simple AD and AD Connector for new deployments to avoid a potentially difficult migration in the future.

Be Consistent When Federating Users to AWS

The proprietary-federation, standards-based-federation and Cognito (without User Pools) user types access AWS via different methods. However, there is one consistency: All methods require that a RoleSessionName be specified (AWS uses the RoleSessionName to identify user activity in CloudTrail). Organizations should match the RoleSessionName to a user attribute from a local directory (for example, Active Directory) so that AWS activity can be matched with an organizational user. By matching the RoleSessionName in this manner, organizations will also be able to leverage a security information and event management (SIEM) tool, such as Splunk, to correlate user activity across multiple platforms.

The Details

This section provides additional details on:

- AWS user types
- Authentication methods
- Access methods

- Authorization constructs
- AWS's Active Directory capabilities
- Auditing

AWS User Types

As alluded to earlier in Figure 1, AWS has the following user types:

- AWS account (aka root account)
- AWS IAM
- Proprietary federation
- Standards-based federation
- Cognito
- Cognito User Pool
- AD Connector
- Simple AD
- AWS Microsoft AD

AWS Account (aka Root Account)

The AWS account was formally introduced when AWS became generally available in 2006. The AWS account is sometimes called the root account because it has global access and privileges to all AWS services within the tenant. At the time of its creation, the number of services was small, the population of users (mostly developers) was small, and few enterprise workloads existed. Access to the AWS account should be tightly protected and rarely used.

AWS IAM

In response to the need for users with different privileges, AWS established the IAM user type in 2011. To identity practitioners, the IAM user type is the most well-understood: These users are defined in the AWS IAM user directory, they have attributes, and they are assigned access rights via IAM groups and roles. Although AWS IAM users are the only ones defined in the IAM user directory, they aren't the only users who can access AWS services (see the additional sections below).

Proprietary Federation

In 2011, AWS introduced the proprietary-federation user type. The proprietary security credentials issued to these federated users have only a short life span (hence the name short-term credentials). The credentials are derived from AWS IAM users who are authorized to delegate their privileges.

Standards-Based Federation

AWS can function as a SAML SP or an OIDC RP. Standards-based-federation users authenticate to AWS via tokens issued from trusted IdPs.

Cognito and Cognito User Pool

In 2014, AWS introduced the Cognito service to assist developers in connecting users to AWS services via mobile apps. For user authentication, developers can leverage third-party identity providers (for example, Google, Salesforce and Twitter). In 2016, AWS added the User Pool feature, which provides an extensible user directory that can be leveraged to both authenticate and issue OIDC tokens.








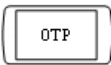

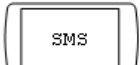



AD Connector, Simple AD and AWS Microsoft AD

The Directory Service user type is associated with AWS's AD Connector, AWS Microsoft AD and Simple AD services. Directory Service users are discussed below, and in the Analysis section of this document.

Authentication Methods

Figure 15 summarizes the authentication methods for users. Note that the graphic excludes AWS root account and Directory Service users, which are discussed elsewhere in this document.

Figure 15. AWS Authentication Options

	AWS IAM 	Proprietary Federation 	Standards-Based Federation 	Cognito 	Cognito User Pool 
L-T Creds					
S-T Creds					
Password	password				password
MFA	 				
SAML and OIDC			 		

© 2017 Gartner, Inc.

L-T creds = long-term credentials; S-T creds = short-term credentials

Source: Gartner (February 2017)

Passwords

Both AWS IAM users and Cognito User Pool users can be authenticated via password, because AWS has the respective user directories to authenticate them. Cognito (without User Pools) supports third-party identity providers, but AWS cannot authenticate these users with a password. They arrive with a credential (for example, an OIDC ID token) issued by the third-party provider.

Multifactor Authentication

In 2009, AWS introduced MFA to provide increased security within the AWS environment. AWS can authenticate IAM users via OTPs, typically generated on the user's smartphone by an MFA application. Because AWS supports the OATH OTP standard, a variety of MFA smartphone applications can be used, including Google Authenticator. AWS also provides its own OATH-based MFA application. IAM users cannot authenticate via the API using an OTP unless they also provide AWS long-term credentials.

AWS IAM and Cognito User Pool users can authenticate via SMS. SMS authentication for users arriving at the Cognito service with third-party provider credentials (for example, OIDC ID tokens) isn't possible. These users have already authenticated to the provider.

SAML and OpenID Connect

Standards-based-federation users and Cognito users (who are not defined in a Cognito User Pool) authenticate to AWS via SAML tokens or OIDC ID tokens. Generally, SAML tokens are used for web access (for example, the AWS Management Console), while OIDC ID tokens are used for AWS API access. However, with additional coding, SAML tokens can be used for API access, and OIDC ID tokens can be used for access to web-based resources.

Long-Term Credentials

AWS long-term credentials are composed of an `AccessKeyId` and a `SecretAccessKey`. AWS long-term credentials are like the "client ID" and "client secret" combination in the OAuth world, except that they are bound to an IAM user — not to an application. Only AWS IAM users can have long-term credentials, and the credentials are used when accessing AWS resources via the API. From a transaction standpoint, the use of these credentials is complex. The `SecretAccessKey` is used as part of a multiprocess sequence to create a signing key. This signing key is subsequently used for signing the data in every API access request. ¹

Short-Term Credentials

Like AWS long-term credentials, AWS short-term credentials include an `AccessKeyId` and a `SecretAccessKey`. However, they also include a `SessionToken`, which AWS uses to validate the `AccessKeyId` and `SecretAccessKey` at runtime. Further, unlike long-term credentials, short-term credentials have an expiration — by default, their validity window is one hour.

Access Methods

Before understanding the specific methods for accessing AWS services, it is important to understand the difference between AWS service access and resource access.

Service Access vs. Resource Access

In terms of managing access to services within AWS, there are differences between:

- Managing access to the administrative operations for a particular service
- Managing resources that may be available within the given service

It is not always possible for AWS to manage all the resources that may be available *within* a given service. For example, AWS IAM cannot manage resource access for Amazon EC2 and RDS, mainly because these two services are running complex environments of their own:

- In the case of Amazon EC2, the instance is running an operating system with applications running on top. Managing the internal operations of such a complex environment is out of scope for AWS.

- Similarly, Amazon RDS runs databases such as Oracle, Microsoft SQL Server and MySQL. These details are treated as a "black box" in terms of AWS management. The responsible parties manage these resources independently, with the native tools they have available.

However, in the case of other AWS offerings, such as Amazon S3 or Amazon DynamoDB, AWS IAM can control access to resources within the service. For example, AWS IAM can control user access to Amazon S3 buckets and objects.

Table 1 explores the differences between service and resource access for three popular AWS services: EC2, RDS and DynamoDB.

Table 1: Examples of AWS Intraservice Access Control

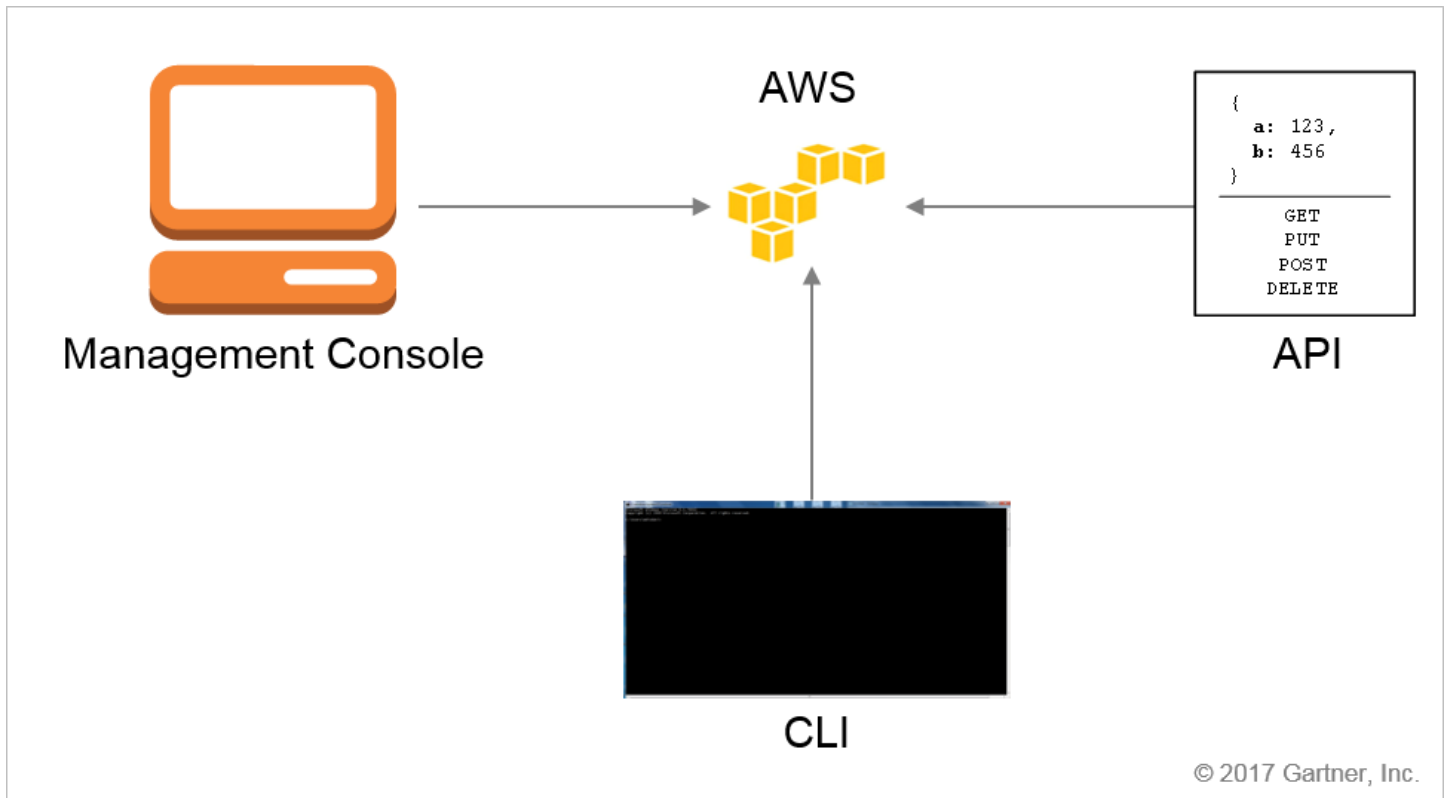
Entitlement ↓	Access Controlled by AWS? ↓
Example 1: Amazon EC2 (Virtual Machines)	
Start, stop or clone an EC2 instance	Yes
Enable access to a file on a Windows Server running inside an EC2 instance	No
Example 2: Amazon RDS (RDBMSs, Such as Oracle, MySQL and Microsoft SQL Server)	
Start, stop or clone the database	Yes
Enable table- or field-level access to data	No
Example 3: Amazon DynamoDB (NoSQL Databases)	
Start, stop or clone the database	Yes
Limit calls to access data	Yes

Source: Gartner (February 2017)

AWS Interfaces

Both the AWS Management Console and the AWS CLI leverage the AWS API as their backbone (see Figure 16). Organizations generally use the console for the basic management of users, entitlements and resources. The CLI allows developers and operations personnel to program or script more complex operations.

Figure 16. AWS Access Methods







Source: Gartner (February 2017)

The AWS Management Console is a browser-based interface for administrators and end users. In the former case, the console is used to administer AWS services. In the latter case, some resources, such as S3 storage or DynamoDB, can be accessed via the console.

Both the AWS API and the AWS CLI allow the orchestration of complex operations and repeatable processes at scale. In general, the API requires developer skills, while the command line provides a simpler interface that enables AWS administrators to easily create scripts.

Figure 17 illustrates the intersection of the AWS access methods and how different user types interact with them. See the previous section for a discussion of AWS service versus resource access.

Figure 17. AWS Access Methods and User Types

	AWS IAM 	Proprietary Federation 	Standards-Based Federation 	Cognito 
Management Console	Y	N ¹	SAML ²	N ¹
API and CLI ³	Y	Y	OIDC ²	Y

© 2017 Gartner, Inc.

N = no; Y = yes.

¹

Not possible without coding.

²

Primary access method. Other access method is possible with coding.

³

Includes PowerShell interface.

Source: Gartner (February 2017)

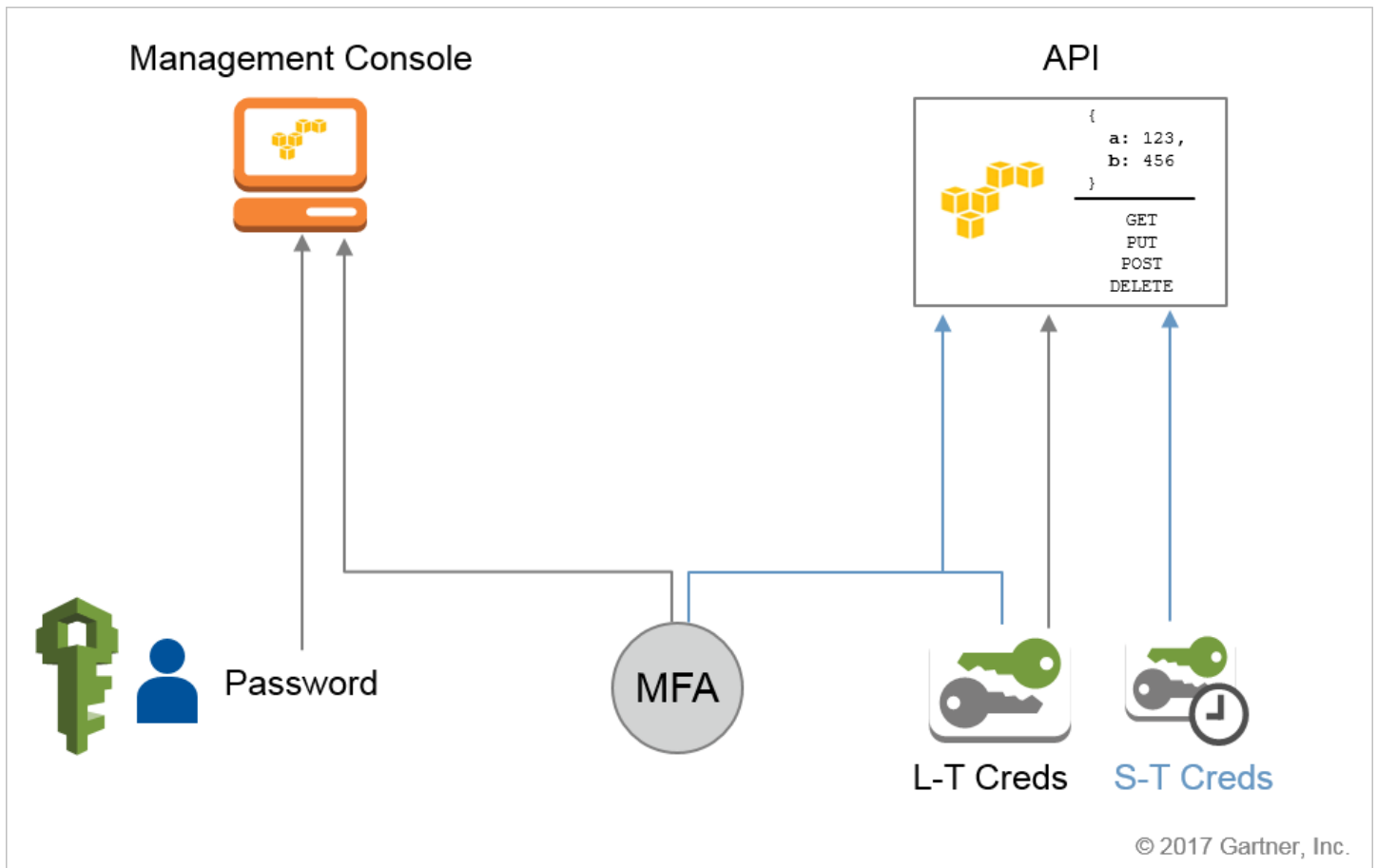
AWS IAM User Access

IAM users have access to all three AWS interfaces. Access to the Management Console functions like most websites. The console uses form-based authentication for IAM users, who use a password or MFA to gain access.

For AWS API access, the IAM user must provide long-term credentials (that is, the AccessKeyID and SecretAccessKey) for authentication. MFA is possible, but the IAM user must present long-term credentials too. MFA to the API or CLI isn't commonly done. The authentication process for the API and the CLI is the same.

Figure 18 illustrates how IAM users authenticate to the Management Console and API. (Authentication to the CLI is not shown.)

Figure 18. AWS IAM User Authentication to Interfaces



L-T creds = long-term credentials; S-T creds = short-term credentials

Source: Gartner (February 2017)

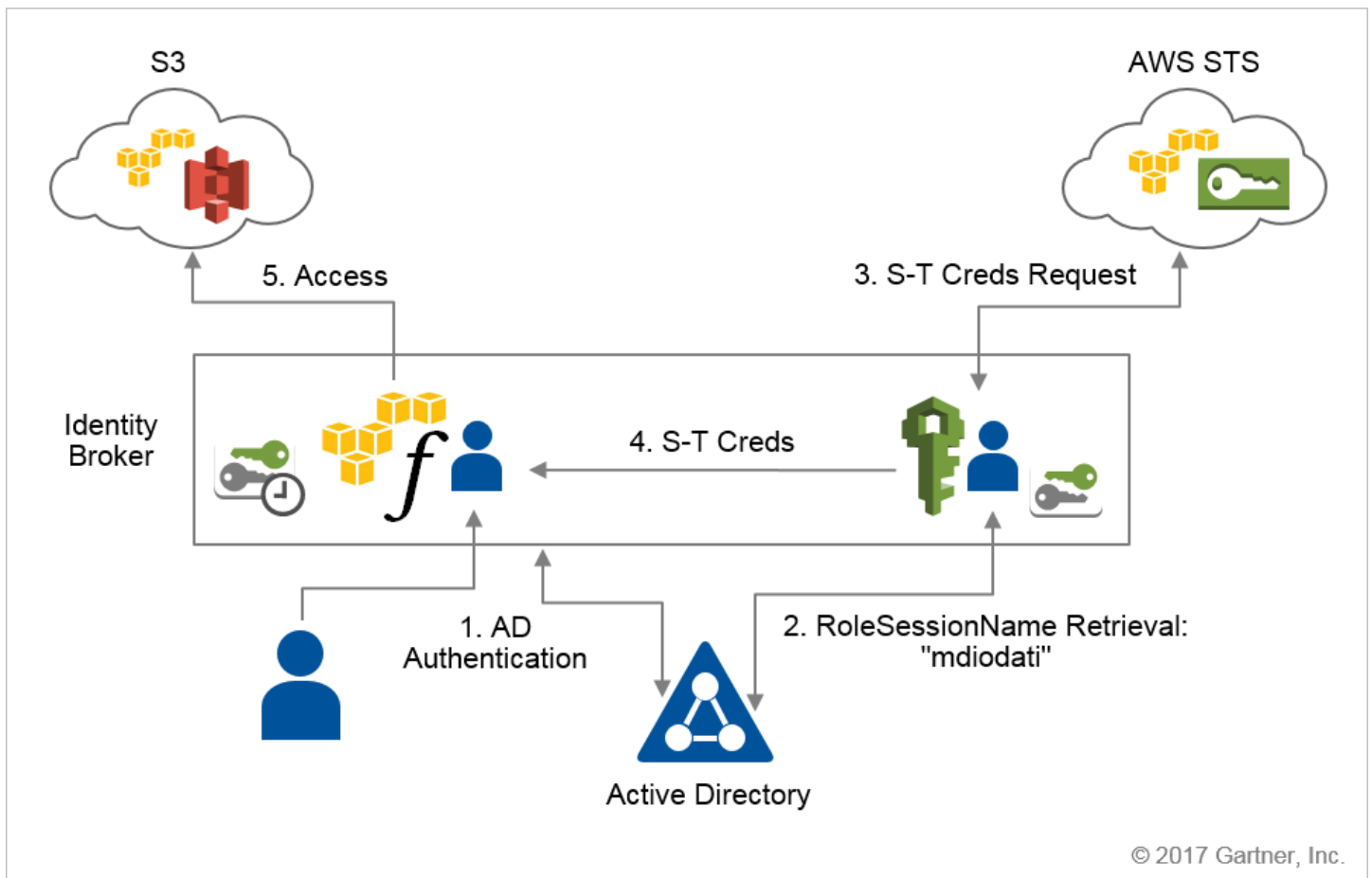
Note that IAM users can use the long-term credentials to procure short-term credentials via the AWS STS. This process enables the IAM user to access resources with a reduced set of privileges, which can improve security. For the purposes of clarity, this process is not included in the figure above.

Federation User Access – Proprietary Method

In 2011, AWS introduced the ability to allow existing enterprise identities to request access to AWS. This greatly simplified the way that more casual users of AWS offerings, such as users not in development or operations roles, could interact with the system. By allowing these users to employ their existing enterprise identities, this federated-access method reduces the administrative burden of having to create AWS IAM accounts for each "casual" user. However, enabling this capability requires the creation of a custom "identity broker" that leverages the long-term credentials of an AWS IAM user. Using these long-term credentials, the identity broker calls the AWS `GetFederationToken` API to retrieve the federated user's short-term credentials.

Figure 19 shows the process by which proprietary-federation users gain access to the API via a custom identity broker.

Figure 19. Proprietary-Federation User API Access via a Custom Identity Broker



S-T creds = short-term credentials

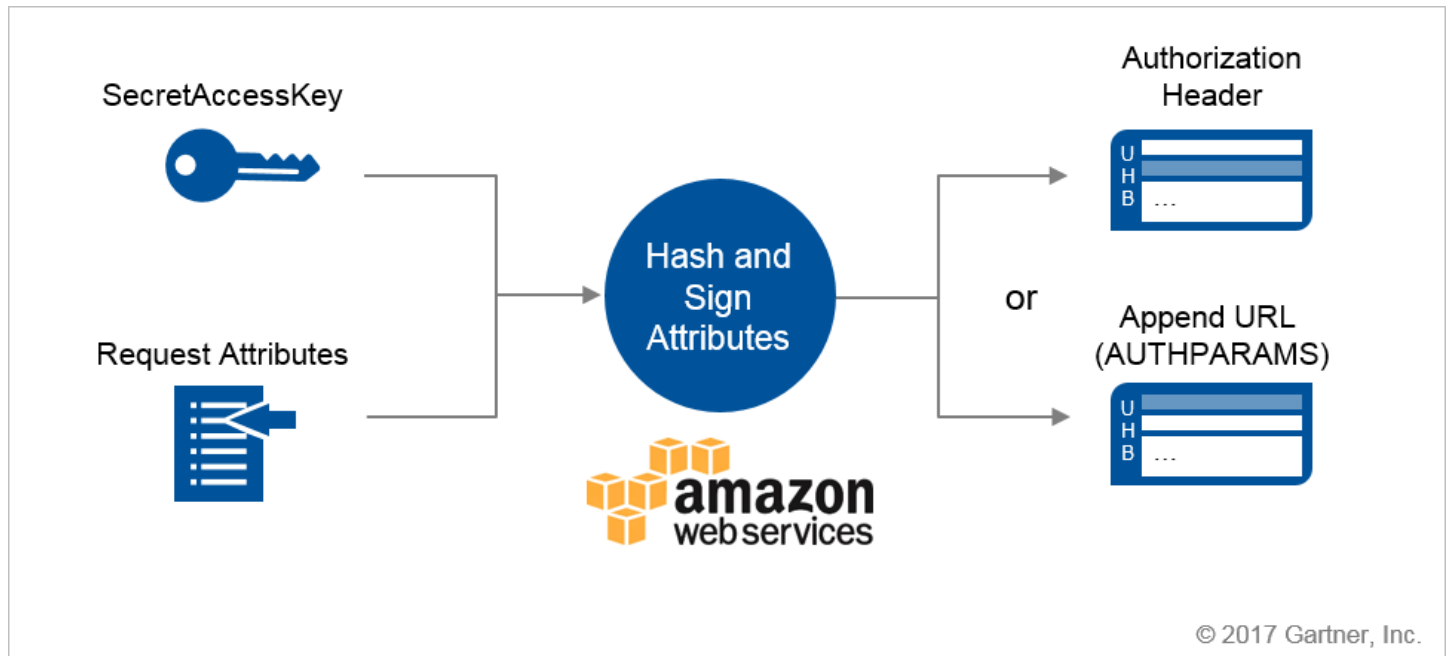
Source: Gartner (February 2017)

1. The federation user authenticates to the broker using his or her existing (non-AWS) credentials, which are contained in an enterprise identity directory such as Active Directory.
2. After the user authentication is complete, the identity broker retrieves the user's unique user identifier (for example, from Active Directory).
3. The identity broker requests short-term credentials from the AWS STS service, using the long-term credentials of an IAM user. In the request, the identity broker provides the user identifier as the AWS RoleSessionName. This identifier is not formally captured within any user directory in AWS. Rather, it is merely a label that identifies the federated user's activities in CloudTrail.
4. These short-term credentials are assigned to the temporary user.
5. The federation user queries the Amazon S3 service and authenticates via the short-term credentials.

Cryptographic Hashing

Whenever an access request is made to the API using long-term credentials, the operations within that request must include cryptographically hashed short-term credentials for authentication (see Figure 20).

Figure 20. AWS Authentication for Access



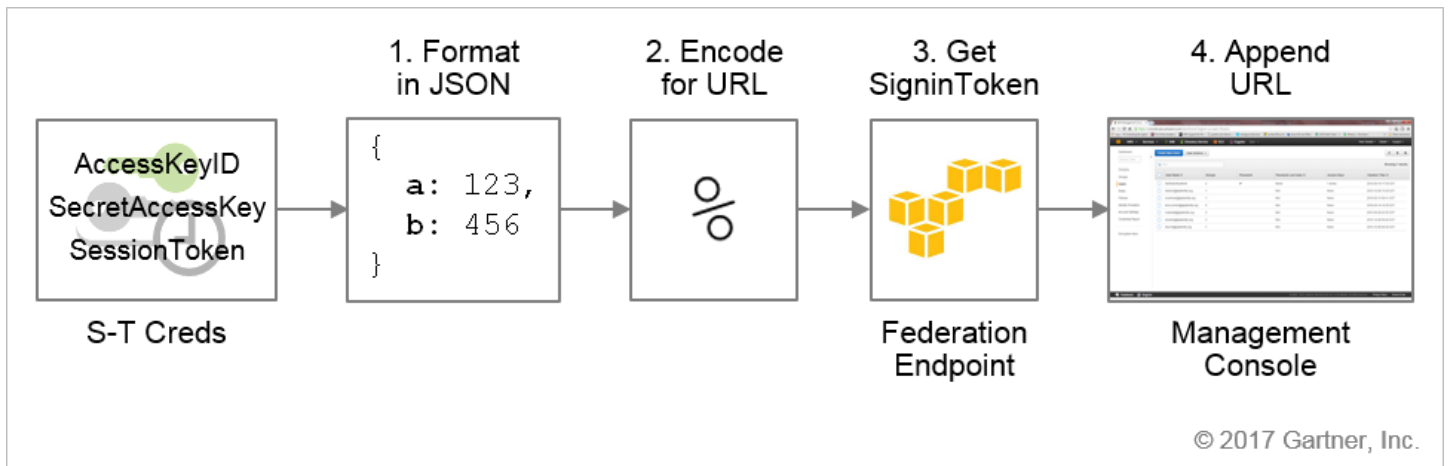
Source: Gartner (February 2017)

This proprietary method can cause challenges for developers. Because the cryptographic process that AWS uses within its API access framework is so difficult to understand and use, most developers will need an AWS SDK – whether they want to use it or not. AWS supports its SDK across many platforms (for example, Node.js, PHP, Python, Java and .NET). However, the use of unsupported platforms is impractical. If AWS used OAuth instead of its proprietary methods, no AWS SDK would be required.

AWS Management Console Access

AWS Management Console access for proprietary-federation users requires development work. Figure 21 illustrates the process for enabling proprietary-federation users to access the web-based Management Console.

Figure 21. Access to the Management Console for Proprietary-Federation Users



S-T creds = short-term credentials

Source: Gartner (February 2017)

1. The short-term credentials (that is, the AccessKeyID, SecretAccessKey and SessionToken) are JSON-formatted.
2. The JSON-formatted short-term credentials are URL-encoded to enable transport over HTTP.
3. The credentials are submitted to the AWS "federation endpoint," located at <https://signin.aws.amazon.com/federation>, via an HTTP GET request. The federation endpoint validates the short-term credentials and returns a SigninToken.
4. The SigninToken, along with other attributes, is appended to the URL for the AWS Management Console. The appended URL enables access to the console.

Federation User Access — Standards-Based Methods

Standards-based-federation users authenticate to AWS with either SAML-based or OIDC-based credentials. For SAML, AWS functions as a SAML service provider. In the case of OIDC, AWS functions as an OIDC relying party.

Note that, although both the SAML and the OIDC access methods described below employ common standards, they still use the underlying AWS IAM mechanisms of:

- Issuing short-term credentials from AWS STS
- Assigning temporary access privileges via the IAM role feature

Thus, the identities of users given role-based access don't exist in any AWS user directory.

SAML

In 2013, AWS introduced the capability to use SAML to grant access to AWS resources, using a commercial IdP that supports SAML 2.0. First, a trust relationship must be set up between the organization's IdP and AWS, by registering the organization's IdP as a SAML provider in AWS. From there, AWS IAM roles can be mapped to users or groups within the IdP to specify the access rights each federation user will be assigned in AWS.

When a federation user authenticates to the IdP to gain access to AWS services, the SAML assertion:

- Specifies the IAM role (defining what the user should have access to)
- Assigns a RoleSessionName (to identify the user in audit logs)

OpenID Connect

The OIDC standard for API-based access to AWS services is useful in situations where client-facing applications need access to AWS resources (for example, a mobile app that stores information in Amazon DynamoDB). In these cases, AWS can accept a token from a public OIDC IdP such as Facebook or Google. That way, if the user authenticates to the original application using this IdP, the user can access AWS resources using the token issued by that IdP.

Figure 9 in the Analysis section of this report illustrates how a native mobile app:

- Passes the OIDC ID token to the AWS STS via the AssumeRoleWithWebIdentity API call
- Subsequently receives the short-term security credentials it needs to access the appropriate AWS resources

Cognito User Access

Figure 2 in the Analysis section of this report illustrates how AWS access via Cognito works when a third-party provider, such as Facebook, Google or Twitter, is used for web authentication. Users are initially redirected to the third-party provider, which, upon successful authentication, issues a token back to the app. This token is exchanged for a Cognito token, which in turn is exchanged for a set of AWS STS short-term credentials that provide temporary access to an AWS service.

Authorization Constructs

The three building blocks for enabling user access to AWS services and resources are access policies, IAM groups and IAM roles. Access policies specify the access rights. The JSON-encoded policies are attached to IAM groups or IAM roles. Users inherit access privileges from an IAM group or IAM role. (Policies can also be attached to IAM users. However, Gartner does not recommend this approach, because it does not scale and will result in excessive access rights over time.)

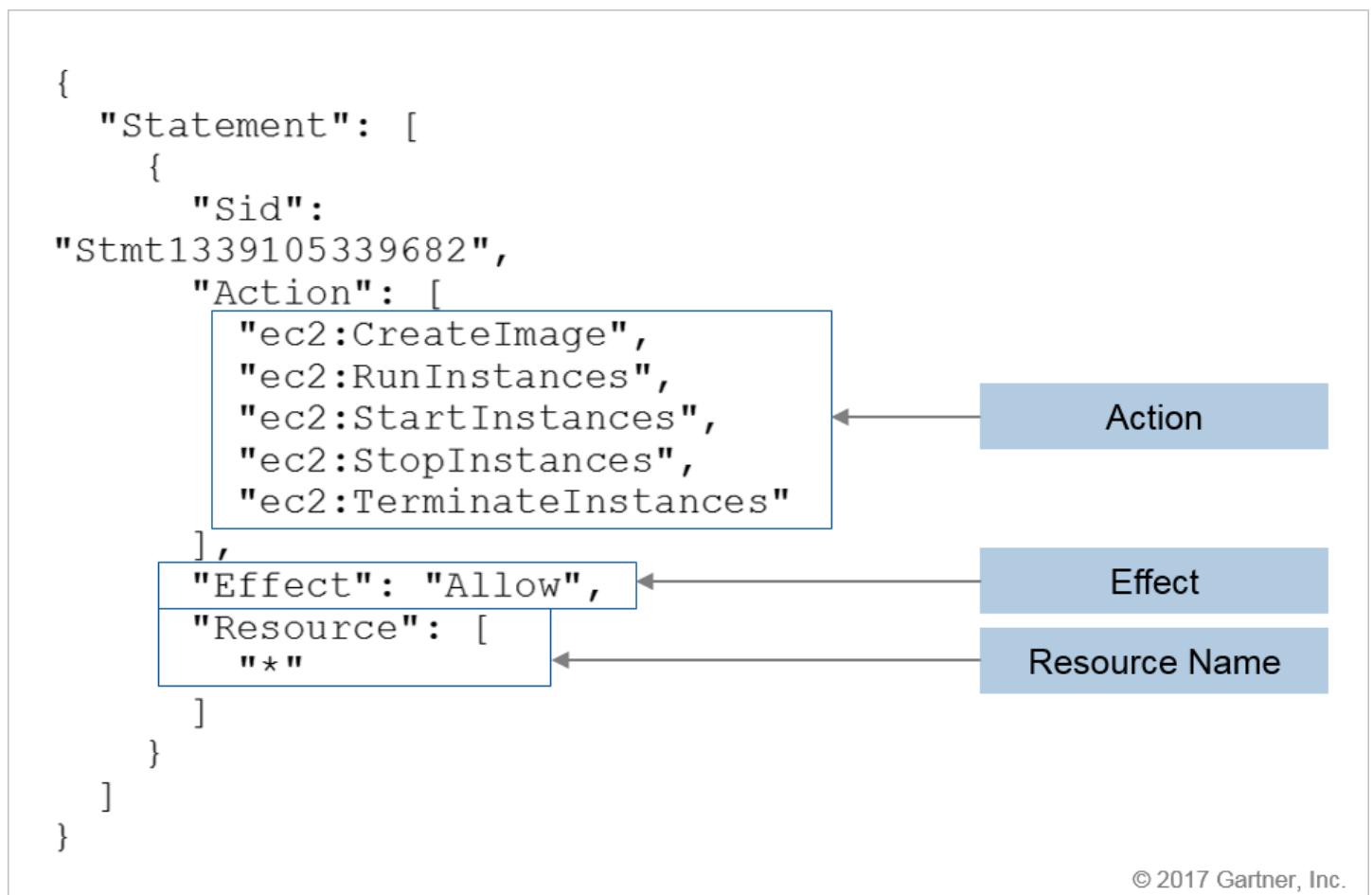
Policies

Within AWS, each policy statement contains four components that define access rights:

- **Principal:** This component is implicit when the policy is attached to IAM users, groups and roles. You must explicitly define the principal within the policy when attaching policies to resources.
- **Action:** This component contains the list of access rights.
- **Effect:** This component is the decision to allow or deny access.
- **Resource:** This component denotes the resources with which the policy is associated – as identified by an Amazon Resource Name (ARN).

To better illustrate the action, effect and resource components, Figure 22 provides a simple example of an AWS policy statement.

Figure 22. Example of an AWS Policy Statement



Source: Gartner (February 2017)

Policies also support a rich set of condition tests and, thus, are usually lengthy. For example, a relatively simple policy that enables a user to access only those EC2 instances that the user owns consumes 35 lines. Complex policies can consume over 100 lines. Many AWS customers will leverage an external JSON tool to edit and validate lengthy policies.

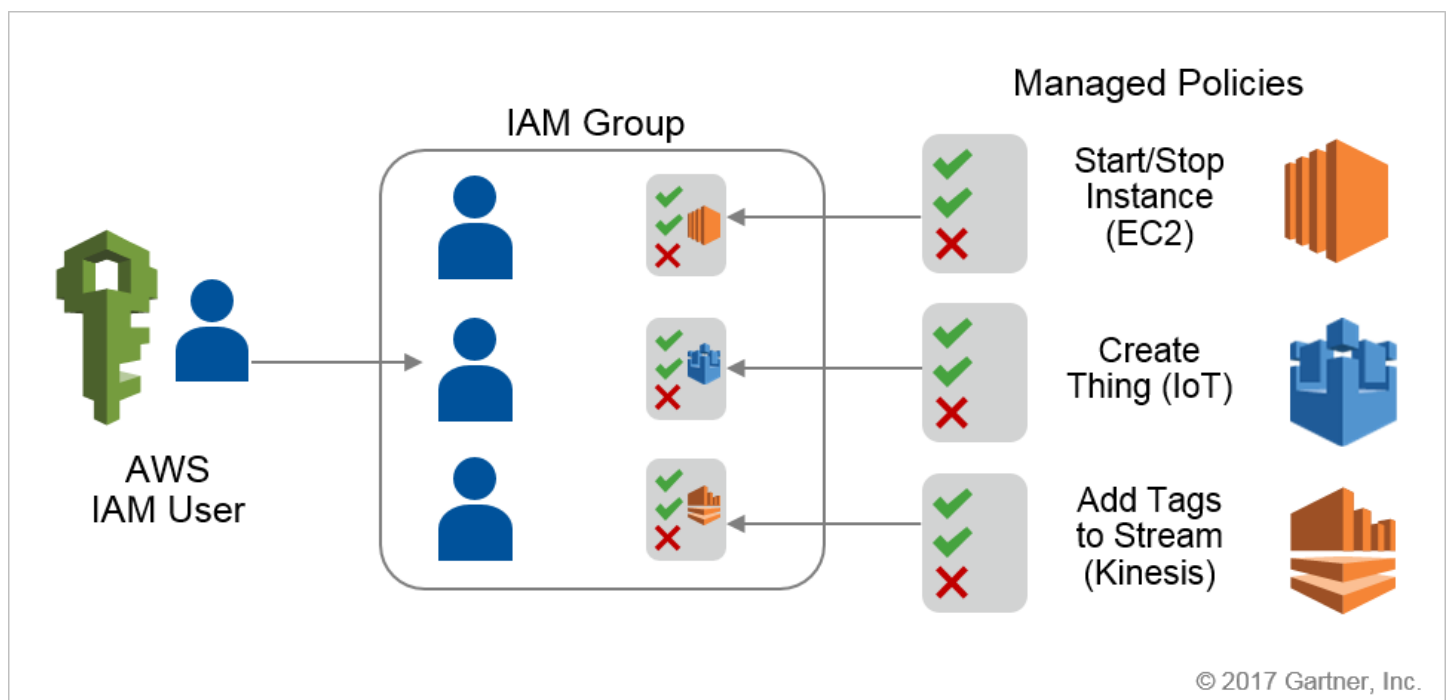
IAM Groups

In 2011, AWS introduced its Identity and Access Management service, which included a new type of user (the IAM user). The service also included a new authorization construct — the IAM group. The IAM group can be used only with the IAM user type.

Unlike the IAM role construct (see the next section), the IAM group construct enables customers to bind users to access rights. In doing so, the IAM group helps customers answer the eternal IAM question: Who has access to what?

AWS IAM groups are used to define a common set of privileges — such as the ability to start or stop an EC2 instance, or the ability to add tags to a stream in Amazon Kinesis (see Figure 23). When customers create an IAM group, they first define privileges by binding the group to new or existing AWS policies. Then, they assign specific IAM users to that group.

Figure 23. Privilege Assignment to AWS IAM Users via Groups and Managed Policies

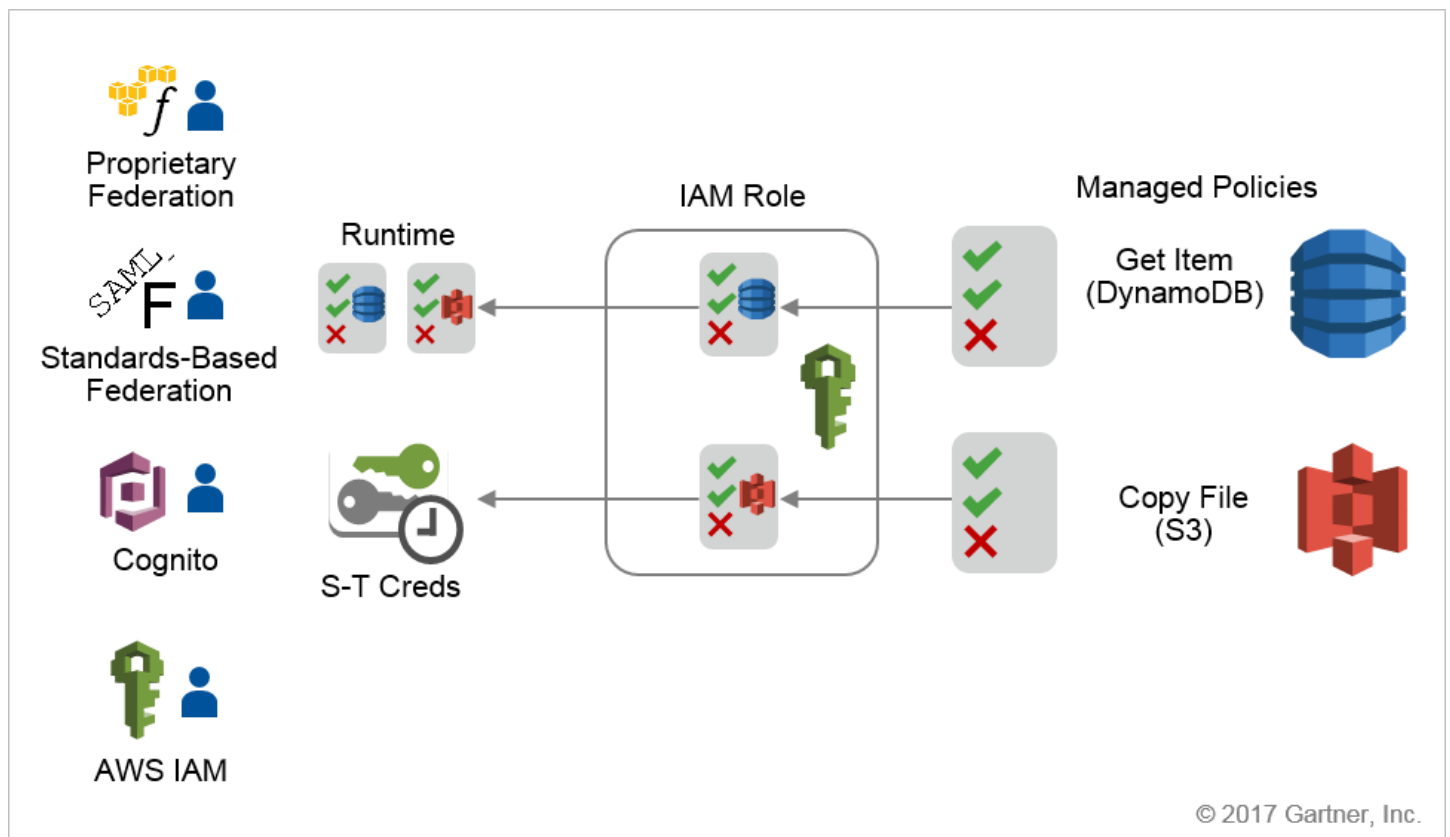


Source: Gartner (February 2017)

IAM Roles

Only AWS IAM users can be bound to IAM groups. For other user types, the IAM role must be used. At runtime, the IAM roles provide a specific set of privileges — such as the ability to retrieve specific data items from a DynamoDB database, or the ability to copy specific files from S3 (see Figure 24). IAM roles are explicitly coupled to short-term credentials. It is the runtime process that binds ephemeral users to an IAM role and the short-term credentials.

Figure 24. Access via IAM Roles



S-T creds = short-term credentials

Source: Gartner (February 2017)

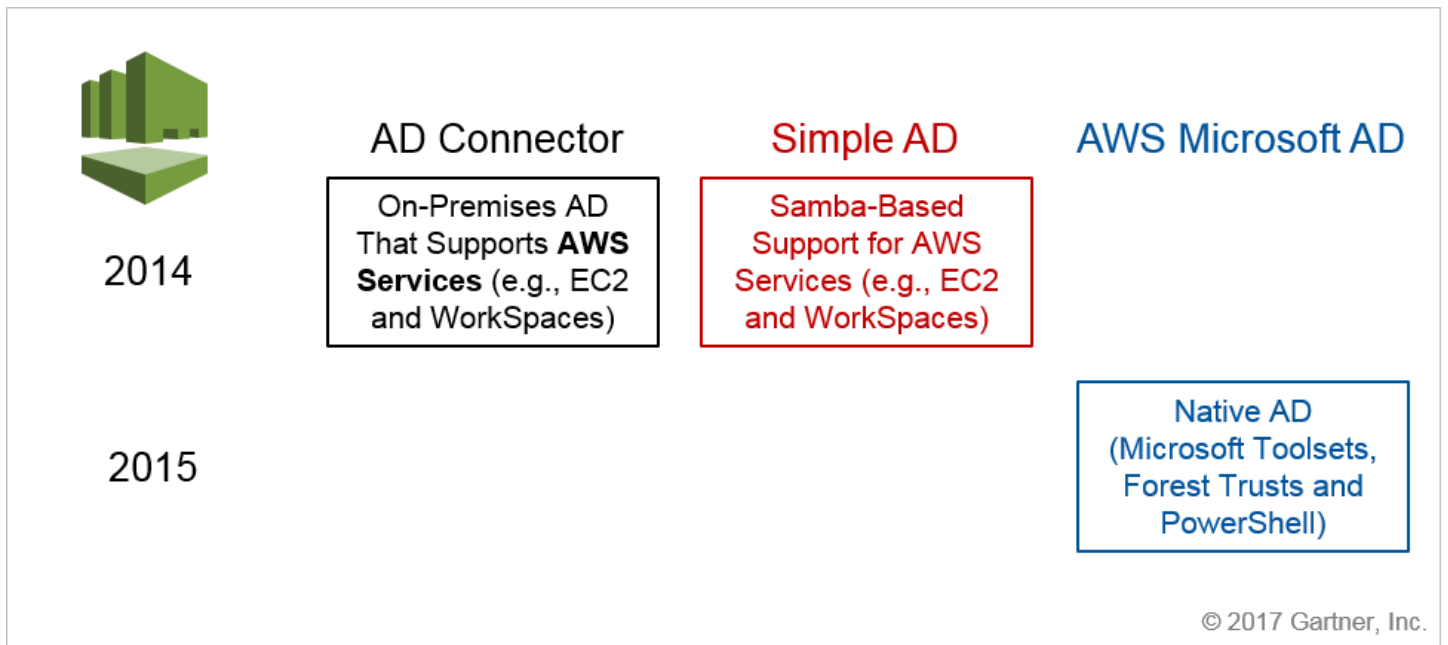
While IAM users can leverage their long-term credentials for access to AWS services, they can also receive access to resources via the combination of an IAM role and short-term credentials. Authenticating IAM users via short-term credentials may be advantageous for certain use cases (for example, to give an IAM user temporary access to AWS resources that require broader privileges than the IAM user possesses by default).

Because each role doesn't contain a list of the users who've been assigned its access rights, there is no easy way to determine which people have been granted these privileges and for how long. The only tracking or identification feature available for federation users is the RoleSessionName assigned to users when they are granted short-term access. The RoleSessionName is captured in CloudTrail when federation users access resources. However, this is only a text field for CloudTrail to record.

AWS's Active Directory Capabilities

In 2014, AWS launched its Directory Service (see Figure 25). Upon its introduction, Directory Service included two products: AD Connector and Simple AD. Each service was designed for different purposes. In 2015, AWS expanded its Directory Service capabilities by launching AWS Microsoft AD.

Figure 25. Evolution of AWS Directory Service

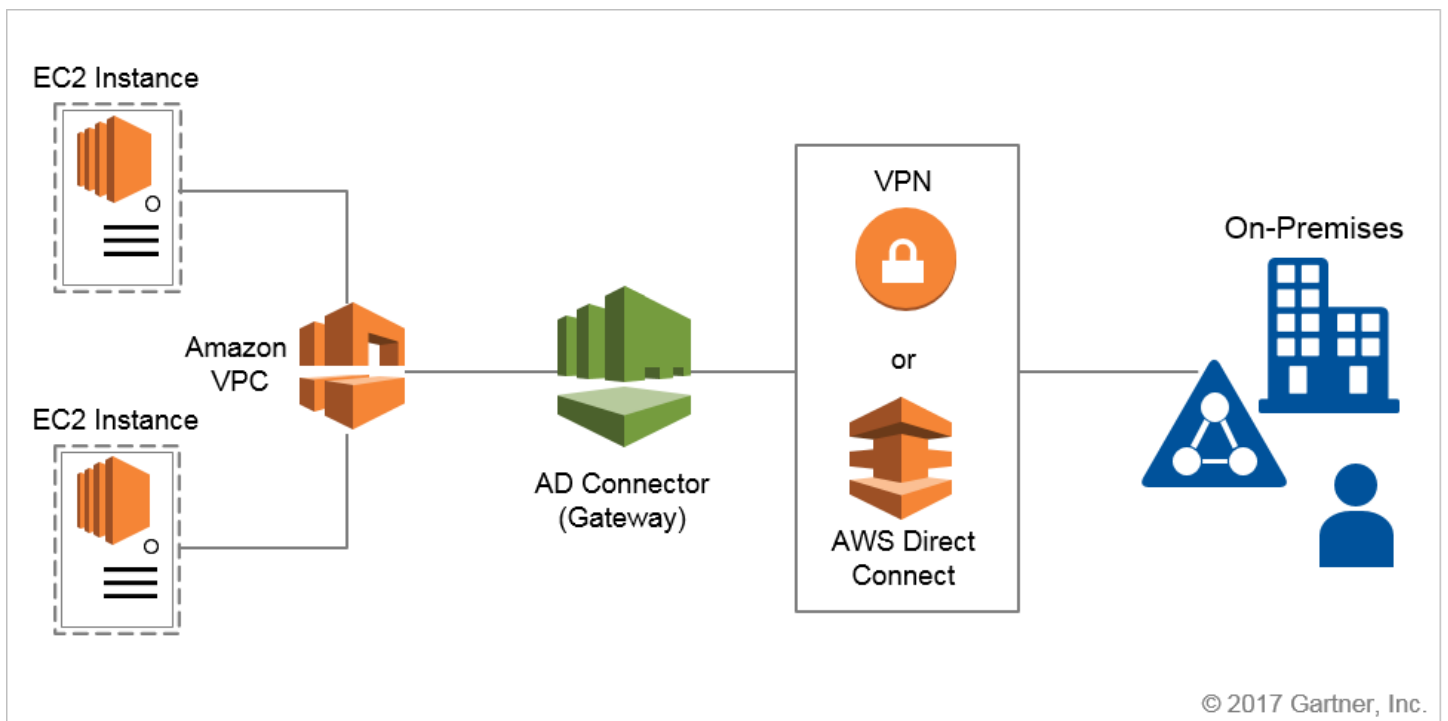


Source: Gartner (February 2017)

AD Connector

AD Connector enables organizations to authenticate to AWS services via their on-premises Active Directory infrastructure. Thus, it enables users to authenticate to both AWS services and on-premises resources through the same mechanism (see Figure 26).

Figure 26. AWS AD Connector



VPC = Virtual Private Cloud

Source: Gartner (February 2017)

Simple AD

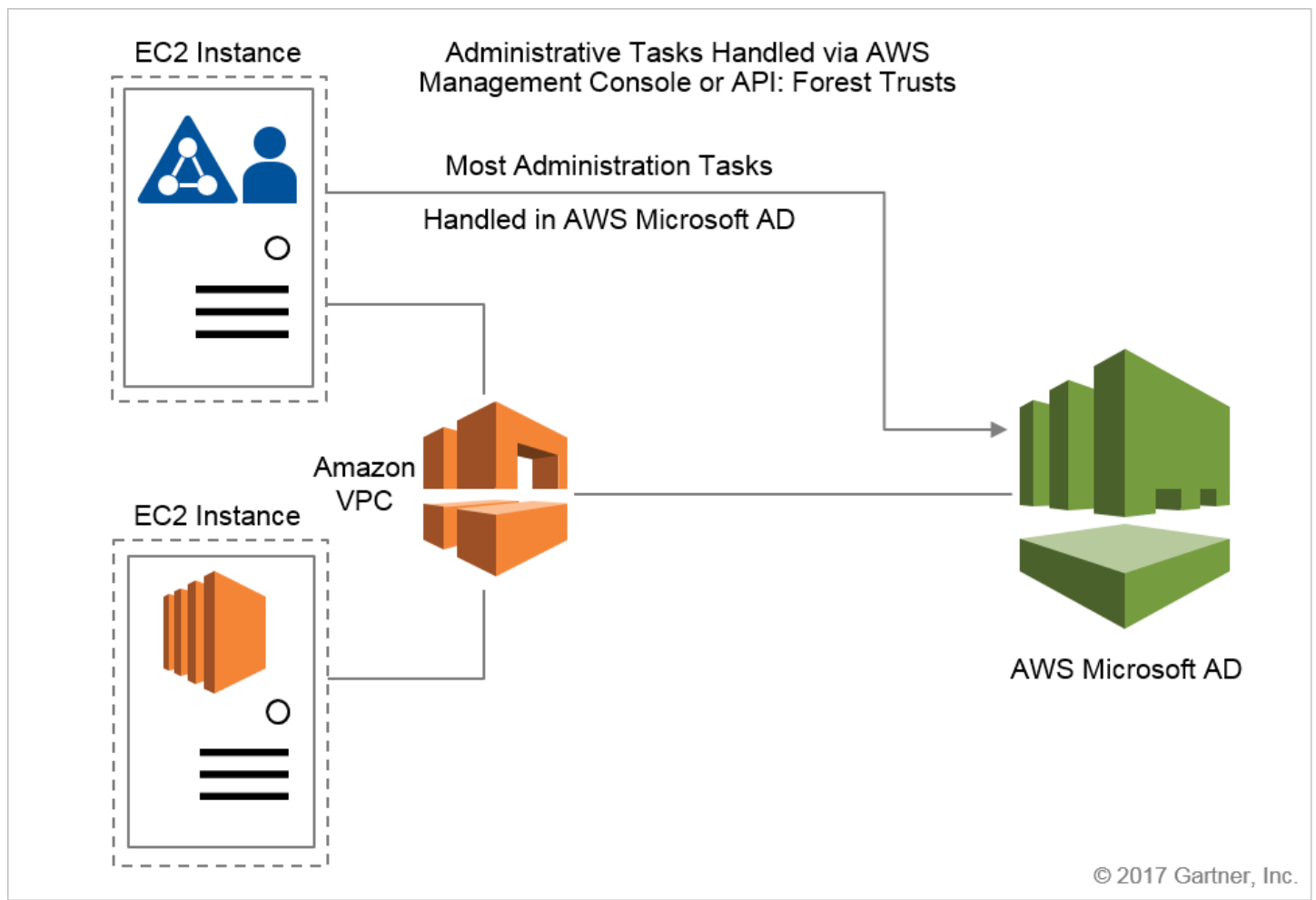
Simple AD was built using [Samba \(https://en.wikipedia.org/wiki/Samba_\(software\)\)](https://en.wikipedia.org/wiki/Samba_(software)) . Its goal was to provide Active Directory-like services, such as user authentication and management, for applications running on EC2 instances. Its purpose was to assist organizations in migrating applications to the cloud.

Simple AD worked for those organizations that needed a limited set of capabilities. However, many organizations needed additional Active Directory capabilities, including support for PowerShell, native Active Directory toolsets and forest trusts. AWS responded to these market demands by introducing AWS Directory Service for Microsoft Active Directory (Enterprise Edition) in 2015 (discussed below).

AWS Microsoft AD

Architecturally, the tenant's AWS Microsoft AD environment is an OU within AWS's single Active Directory forest. The tenant's administrative privileges are limited to its OU. Only a few high-level Active Directory administration tasks, such as forest trusts, are handled via the AWS Management Console (see Figure 27). For other tasks, administrators log in to a running EC2 instance that belongs to the OU, and then use the native Microsoft toolsets and PowerShell interfaces. Because the AWS Microsoft AD service provides a completely cloud-native way to perform native Active Directory administration for EC2-hosted Windows applications, it offers a major improvement over Simple AD toward enabling clients to more easily migrate applications to the cloud.

Figure 27. Virtual Directory Services Using AWS Microsoft AD



Source: Gartner (February 2017)

Auditing

In 2013, AWS unveiled the CloudTrail service, which provides records of all AWS API activity. With CloudTrail, organizations can collect event information for all their resources on AWS, including AWS API calls that are made from their own applications as well as from third-party applications. This capability can help with security analysis, compliance monitoring, and user and resource life cycle tracking by monitoring state changes to security group settings and policy configurations. With the integration of AWS IAM, CloudTrail can provide information on:

- What requests have been made to AWS offerings
- Who made these requests
- When the requests were made

Thus, CloudTrail opens the door for providing better context and insight into the organization's AWS environment through broader enterprise security monitoring solutions.

Evidence

¹ "Task 3: Calculate the Signature for AWS Signature Version 4,"
(<http://docs.aws.amazon.com/general/latest/gr/sigv4-calculate-signature.html>) Amazon Web Services

Note 1

Proprietary-Federation vs. Standards-Based-Federation User Types

Because AWS employs different federated-access methods, Gartner uses the terms "proprietary federation" and "standards-based federation" to distinguish among user types when necessary.

Document Revision History

Identity and Access Management Within Amazon Web Services' Public Cloud - 6 March 2014
(<https://www.gartner.com/document/code/261157?ref=ddrec>)

Identity and Access Management Within Amazon Web Services' Public Cloud - 5 February 2013
(<https://www.gartner.com/document/code/247561?ref=ddrec>)

Recommended by the Author

In-Depth Assessment of Amazon Web Services (<https://www.gartner.com/document/3371747?ref=ddrec&refval=3620417>)

Defining and Implementing Effective Cloud Security Architecture in Amazon Web Services
(<https://www.gartner.com/document/3454732?ref=ddrec&refval=3620417>)

2017 Planning Guide for Cloud Computing (<https://www.gartner.com/document/3471551?ref=ddrec&refval=3620417>)

2017 Planning Guide for Identity and Access Management
(<https://www.gartner.com/document/3477118?ref=ddrec&refval=3620417>)

Recommended For You

2019 Planning Guide for Identity and Access Management
(<https://www.gartner.com/document/3891190?ref=ddrec&refval=3620417>)

Solution Comparison for IAM Features in Three UEM Suites
(<https://www.gartner.com/document/3886676?ref=ddrec&refval=3620417>)

Active Directory: The Time to Modernize Is Now (<https://www.gartner.com/document/3890763?ref=ddrec&refval=3620417>)

Solution Comparison for OAuth-Based Access Management Capabilities of Nine Vendors
(<https://www.gartner.com/document/3895093?ref=ddrec&refval=3620417>)

IAM Is Vital for Successful Application Migration to IaaS

(<https://www.gartner.com/document/3895269?ref=ddrec&refval=3620417>)

© 2017 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates. This publication may not be reproduced or distributed in any form without Gartner's prior written permission. It consists of the opinions of Gartner's research organization, which should not be construed as statements of fact. While the information contained in this publication has been obtained from sources believed to be reliable, Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information. Although Gartner research may address legal and financial issues, Gartner does not provide legal or investment advice and its research should not be construed or used as such. Your access and use of this publication are governed by [Gartner's Usage Policy](#). Gartner prides itself on its reputation for independence and objectivity. Its research is produced independently by its research organization without input or influence from any third party. For further information, see "[Guiding Principles on Independence and Objectivity](#)."