549.1

Cloud Account Management and Identity Foundations

SANS | GIAC CERTIFICATIONS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

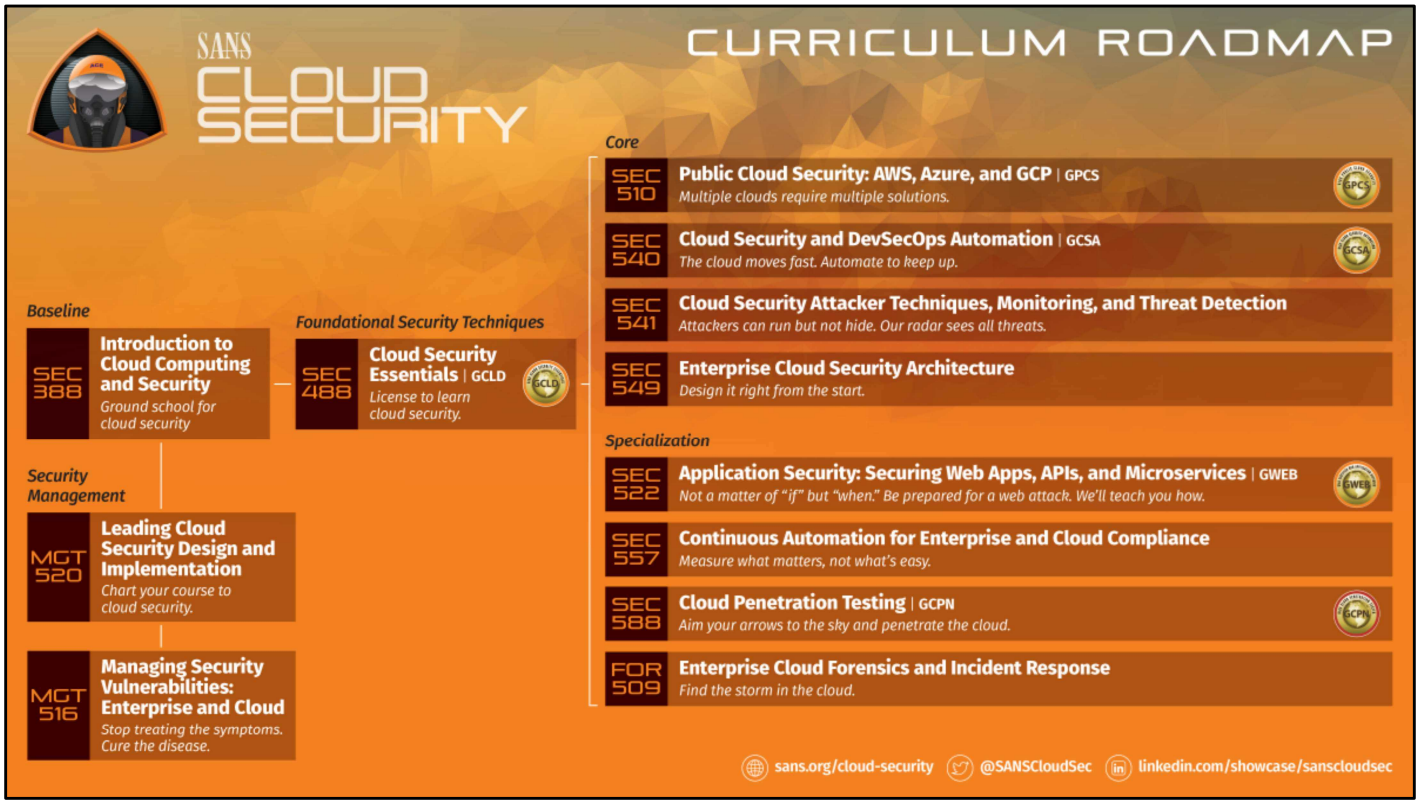
All reference links are operational in the browser-based delivery of the electronic workbook.

SEC549.1 | Enterprise Cloud Security Architecture

SANS

Cloud Account Management and Identity Foundations

This page intentionally left blank.



The SANS Institute, established in 1989 as a cooperative research and education organization, is the most trusted and by far the largest source for information security training and security certification in the world. It also develops, maintains, and makes available at no cost the largest collection of research documents about various aspects of information security, and it operates the internet's early warning system—the Internet Storm Center. Its programs now reach more than 165,000 security professionals around the world.

SANS offers a number of courses that teach developers, architects, testers, security professionals, and managers how to build more secure applications. Anyone involved in developing, securing, and defending applications can benefit from the following courses in the SANS Cloud Security Curriculum:

SEC388: Introduction to Cloud Computing and Security | 3 Sections
Advise and speak about a wide range of cloud security topics and help your organization successfully navigate both the security challenges as well as the opportunities presented by cloud services.

SEC488: Cloud Security Essentials | GCLD | 6 Sections
Advise and speak about a wide range of cloud security topics and help your organization successfully navigate both the security challenges as well as the opportunities presented by cloud services.

SEC510: Public Cloud Security: AWS, Azure, and GCP | GPCS | 5 Sections + Extended Lab Hours
Perform multicloud security assessments across AWS, Azure, and GCP clouds identifying key weaknesses and hardened configurations in core cloud services.

SEC540: Cloud Security & DevSecOps Automation | GCSA | 5 Sections + Extended Lab Hours
Provides development, operations, and security professionals with a methodology to build and deliver secure infrastructure and software using cloud services and DevSecOps workflows.

SEC541: Cloud Security, Attacker Techniques, Monitoring, and Threat Detection | 5 Sections
Leverage cloud security tools and services to monitor your environment and look for adversaries.

SEC549: Enterprise Cloud Security Architecture | 2 Sections
Ensure you have the necessary foundation, tools, and skills to utilize cloud services in architectural design and can use them in a real-world example.

SEC522: Application Security: Securing Web Apps, APIs, and Microservices | GWEB | 6 Sections
For anyone who wants to get up to speed on web application security issues and the best ways to prevent common web application vulnerabilities.

SEC557: Continuous Automation for Enterprise and Cloud Compliance | 5 Sections
Teaching professionals tasked with ensuring security and compliance how to stop being a roadblock and work at the speed of the modern enterprise.

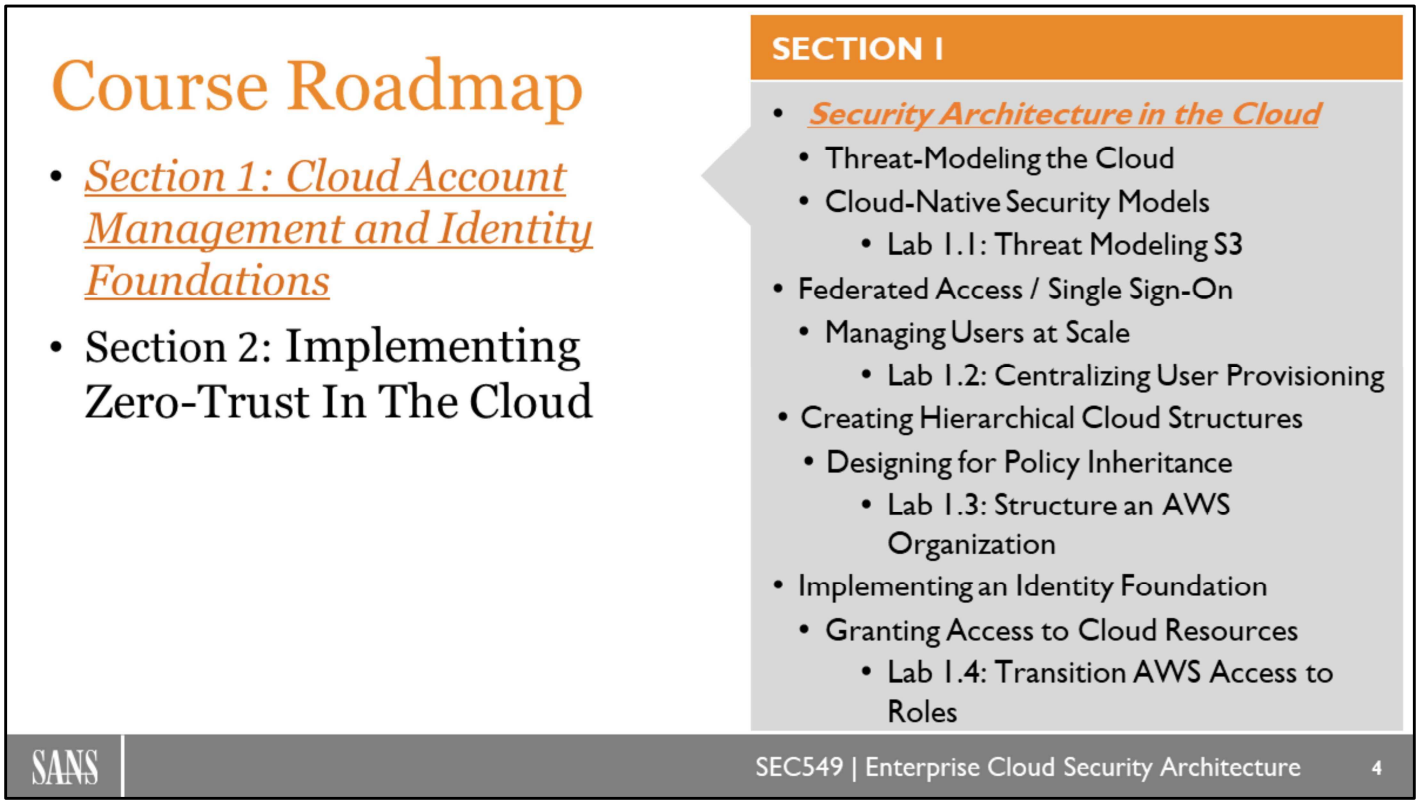
SEC588: Cloud Penetration Testing | GCPN | 6 Sections
Prepares penetration testers to assess infrastructure and applications hosted in the public using platforms such as AWS, Azure, and Kubernetes.

FOR509: Enterprise Cloud Forensics and Incident Response | 4 Sections
Designed to address today's need to bring examiners up to speed with the rapidly changing world of enterprise cloud environments.

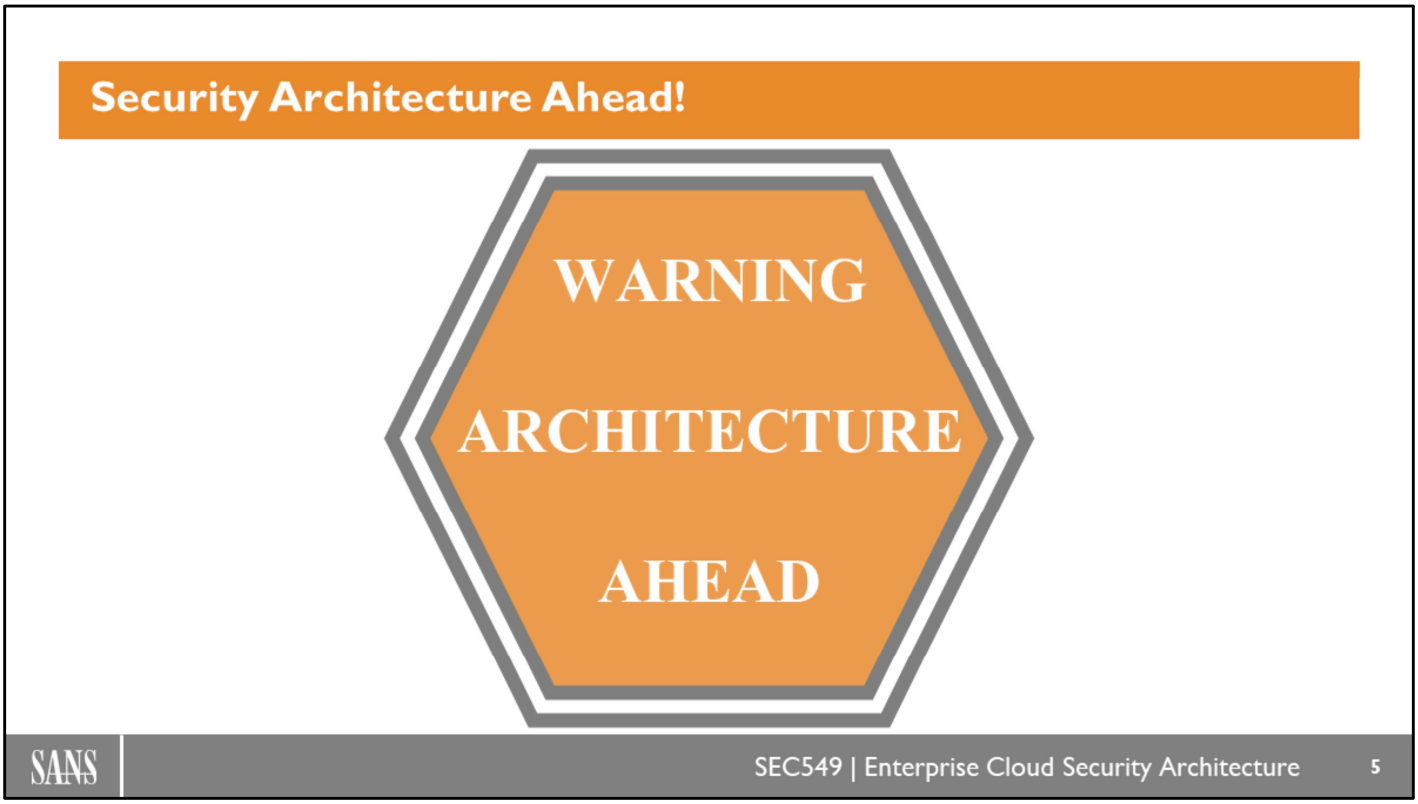
MGT520: Leading Cloud Security Design & Implementation | 3 Sections
Learn to build your cloud security program and roadmap.

MGT516: Managing Security Vulnerabilities: Enterprise and Cloud | 5 Sections
Highlights why organizations struggle with enterprise and cloud vulnerability management and shows how to solve these challenges.

Review our Job Role Flight Plan **sans.org/cloud-security**



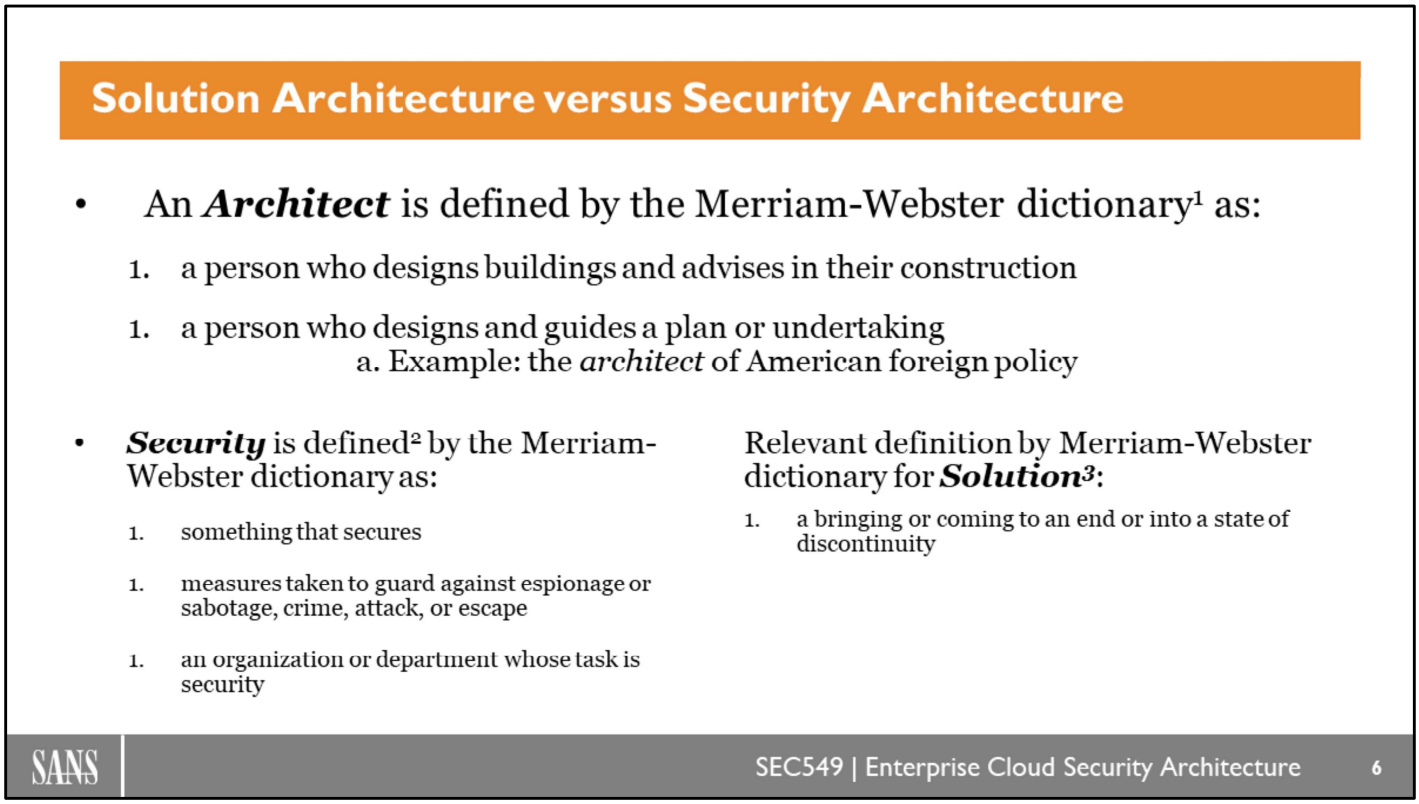
In this section, students will be introduced to security architecture as it applies to the cloud. We begin by looking at how one should ideally approach security architecture in the first place and then study how basic security architecture principles also apply to building enterprises in the cloud. The most important takeaway in this section is the understanding of the strategic nature of cloud security architecture. Through proactive strategy, security architects can build systems inherently resilient and better resistant to threats from the outset.



Warning! This course will be overflowing with architectural patterns and design philosophies in the cloud.

We won't be limited to conceptual discussions, however. In this course, you will be logging into a prebuild AWS environment of a fictional Enterprise, Delos, Inc., as it migrates from a traditional on-premises organization to the Cloud. You will need to navigate the AWS console and identify the the individual resources that are components of larger, architectural designs. Getting your hands dirty in the AWS console will give you practical experience with the nuts and bolts of cloud resources, giving you more confidence to work with them in the future.

In the upcoming module, you will be introduced to security architecture and how it translates to the Cloud. The most important takeaway in this section is the understanding of the strategic nature of cloud security architecture. Through proactive strategy, security architects can build inherently resilient systems and those that are better resistant to threats (both internal and external) from the outset.



'Solution' versus 'Security' Architecture

Let's take a moment to address the differences (and similarities) between Solution Architecture and Security Architecture and see where the concepts between the two practices diverge. Contrasting the definitions in a workman-like fashion can reveal some insights into the similarities and differences between Security Architecture and Solution Architecture.

Solution Architecture

Our starting point in this examination underscores the concept of an Architect; that is, a person who guides and plans. Therefore, a Solution Architect is a person who is creating a plan to bring an end to a state of discontinuity. This is a rather amorphous concept, but if anyone has practiced Solution Architecture, this compound definition might have some truth to it. The needs Solution Architecture must fill are very diverse. A successful Solution Architect has ingested the requirements for all stakeholders in the business and crafted a plan that takes into account usability and cost control.

Security Architecture

A Security Architect also consumes business-specific requirements, but their directive is more focused than open-ended. If we take our definitions of '***Security***' and '***Architect***' and put them together, we arrive at a definition of a person who creates plans to guard against espionage or sabotage, crime, attack, or escape. More simply, this is an architect who designs security. This is entirely on track with a Security Architect's day-to-day work, which might be consumed with designing access control patterns and network controls or prescribing sufficient logging levels across their organization.

As we all know, 'security' is a relative term. There is no absolute scale of insecure or secure, so one of the cornerstones of a Security Architect's role is to determine 'how much security' any one asset needs. This is measured by the value of the asset you are looking to secure, and the risk presented should that asset be infiltrated by a bad actor.

References:
[1]: https://sec549.com/id106
[2]: https://sec549.com/id107
[3]: https://sec549.com/id108

What Is Security Architecture?

SABSA defines security architecture as "A **rational framework** within which decisions can be made upon.[1]"

In this course we consider Security Architecture as an opportunity to consider the **strategic** dimension to technology, rather than purely tactical.

SANS | SEC549 | Enterprise Cloud Security Architecture 8

What is Security Architecture?
Ask 10 people in the security industry this question and you will get 10 different answers – especially if you ask for a 100-word synopsis of the topic. Rather than be dismayed that this is a topic that no one can define (or that it is too complex to be simply defined), focus instead on the person you hear the response from. What you will see is that each will define it clearly but will provide an answer that is true *from their own perspective*. As with most complex topics, perspective matters in security architecture.

There are a few things in each brief synopsis on security architecture that you will be the same in comparing one person's definition to another. These similarities are your take-home message to the question of what security architecture really is.

- It is a *framework* to build upon
- It has a *tangible purpose*
- It offers *measurable results*
- It is *cost-effective*
- It offers a *flexible/resilient solution* to security issues
- It *aligns itself with the vision, goals, and strategy* of the organization
- It offers a clear *return on investment*

Examples of Security Architecture Frameworks:

- ***TOGAF (The Open Group Architecture Framework)*** looks at business problems and their solutions in the area of enterprise architecture with some direction for meeting security goals. The primary focus is on the early stages of designing technical solutions, when the organization's vision, goals, and objectives are defined and the problems it wants to solve are teased out and addressed. The nuts and bolts of how to address security are not outlined in this framework as it is primarily used to to build solutions architectures.[2]

- ***SABSA (Sherwood Applied Business Security Architecture)*** is a framework that aids the security architect in answering key questions about a system (who, what, when, where, and why). The organization defines the SABSA framework as a structured systematic approach to solving complex problems in strategy, planning, and operation. The goal is to ensure that the design, delivery, and support of security services are integral to every enterprise's IT management strategy. SABSA doesn't dictate the specific technical components of security architecture (the 'how' of the process), rather it provides a framework for threat modeling a system and designing policy-driven controls.[3]

- ***OSA (Open Security Architecture)*** is a framework that looks at the technical aspects of the security architecture and is more comprehensive and specific to security needs as opposed to TOGAF. OSA looks at the key components of security architecture – the principles, threats, components, and concepts – needed to design an effective security architecture. OSA can be used once a system has already been designed by benchmarking existing systems against published patterns.[4]

- ***DODAF 2.0 (Department of Defense Architecture Framework 2.0)*** was designed for the US Department of Defense, so it isn't commonly used by businesses. It takes an integrative approach to security architecture so that their primary focus in Enterprise Architecture, for example, is to have a common language, no matter what architectural framework is being used. According to the DODAF 2.0 Architects Guide, related architectural features should have identical names, numbers, and meanings so that components of different models are more easily compared and integrated into a custom-fit design.[5]

The nuts and bolts of security architecture will look different if you are talking to someone architecting a system for the US Department of Defense (DoD) or someone doing the 'same thing' for an online retail company. No one would argue here that the security needs (and therefore the security architecture) of these two organizations differ – mostly because of vast differences in their quantified operational risk.

The heads of each organization, however, would agree they want a 'secure' system, even if it is for very different reasons. The DoD uses its own security architecture framework that, by necessity, looks very different from the framework used by those architecting a security strategy for the retail company. As you look at each framework, it is easy to be tempted to wonder, "Which is better?" This temptation is not practical, however, when discussing security architecture. Instead ask, "Why does this framework work better for one organization and not the other?" or "Which framework works best for where we are at in the process of designing a secure system for an organization, business, or other entity?"

References:
[1]: Enterprise Security Architecture: A Business-Driven Approach - By Nicholas Sherwood
[2]: https://sec549.com/id7
[3]: https://sec549.com/id9
[4]: https://sec549.com/id8
[5]:https://sec549.com/id10
John Sherwood. SALSA: A method for developing the enterprise security architecture and strategy.
Computers & Security,
Volume 15, Issue 6, 1996, Pages 501-506.
Image by FelixMittermeier from Pixabay

Good Architectural Design Helps Manage Complexity

Architectural complexity arises because of a **high number of interconnected systems.**

Modeling **complex systems** is inherently difficult.

Finding and addressing security gaps at the design stage **saves money and time.**

Good Architectural Design helps manage Complexity

Complexity is a natural outgrowth of the evolution of technology. At times, complexity is a symptom of disorder or disarray in the way systems are interconnected. Other times, complexity emerges when a systematic, orderly approach is taken to address the management of scale. In either case, complexity means an ever-expanding attack surface and increased opportunity for error.

The security architect's goal is to tease out security flaws that appear at all layers when information systems interconnect, and complexity naturally forms. The next step then is to preemptively manage them, ideally before the system is put into use. Fixing security flaws in the early stages of the development of any system is extremely time-saving and cost-saving, especially considering the high cost associated with fixing flaws after deployment.

Reference:
Image by OpenClipart-Vectors from Pixabay: https://sec549.com/id131

Security Architecture Meeting the Needs of Business

- Define the business objectives, goals and vision first. *Where is the organization going?*
- Study how to help the business achieve its goals. *Who are the users? What data needs protection?*
- Formulate a plan that embeds security into the business' technical strategy and solutions.
- Determine 'how' to get there. *How would you manifest conceptual security controls in your environment?*

Security Architecture Must Meet the Needs of Business
The security architects goal isn't to solely pursue an ever more secure system. Security organizations embedded within a larger organizations exist first to meet the needs of the business by 1) evaluating what its needs are, 2) uncovering inherent risks, and 3) designing controls to mitigate those risks.

The security team's real goal isn't to say 'no, it can't be done,' but to think 'how can we securely aid the business in its desired operations?' This begins with developing and applying the needed secure patterns to meet the business' goals, objectives, and vision. While the security architect might want to implement these patterns with a top-down approach, often their organization won't sit still long enough to achieve overly long-winded transformations. Many times, the best way to help your organization achieve a set of long-term objectives is with a phased approach where interim architectures and milestones are celebrated.

Step 1: Define the business' objectives, goals, and vision. What is the business used for? What data does it generate and therefore need to protect? Who uses it and how? What are its main objectives (security requirements, convenient UX, rapid access, remote working environment, etc.)

Step 2: Begin to think about the risks associated with the implementation of strategy that could hinder the achievement of the business's goals. Document these risks and socialize with stakeholders.

Step 3: Crystalize a plan to design and implement the needed controls to manage the risk undertaken by the business. Here the operative word is 'manage'. Not all risks can or should be completely eliminated. Some risk is inherant.

Step 4: Begin to answer the question of 'how' the design of security controls can be realized.

Reference:
Image by StockSnap from Pixabay: https://sec549.com/id132

How Does Security Architecture Apply to the Cloud?

Security design must recognize the reality of a varied attack surface: It depends on which cloud service model is in use.

- **Infrastructure as a Service (IaaS):** Threats target OS-layers and Networks. Controls for this service model often mirror those in on-premise systems.
- **Platform as a Service (PaaS):** Controls for PaaS will address Application-Layer vulnerabilities and cloud infrastructure misconfigurations.
- **Software as a Service (SaaS):** Access-Control and configuration management are the primary attack surfaces.

SANS | SEC549 | Enterprise Cloud Security Architecture 12

Security Architecture in the Cloud

The security architect designing in the Cloud must first determine what constraints are placed upon them. You need to recognize that, in certain circumstances, there is an innate loss of control of certain aspects of the tech stack. Your visibility and the approach needed depends greatly on the cloud service module being used:

- *Infrastructure as a Service (IaaS)* – these systems require security controls that consider the likelihood of threats to the storage layer, operating system, application layer, and virtual network components.
- *Platform as a Service (PaaS)* – the security architect must consider any security threats to the applications deployed to the cloud. The underlying cloud infrastructure is not within your sphere of influence, so it will not be a particular concern of the architect designing the security of a PaaS-based system. However, configurations of the service can be a frequently abused attack surface.
- *Software as a Service (SaaS)* – the architect will be concerned with how the SaaS application is integrated with the existing technology of the system. The primary focus of security in a SaaS system is the configuration of access control. The apps and infrastructure are not within your sphere of influence but, just like PaaS systems, misconfigurations can lead to an unsecure state of the deployment.

To address these needs, this course offers the Security Architect the ability to adapt and adjust their threat models to be specific to the cloud landscape. This involves the mitigation of threats through effective design strategies, allowing cloud-based businesses to expand and grow in sustainable ways in this unique environment.

Reference:
Image by jeferrb from Pixabay: https://sec549.com/id133

Security Architecture and the Identity Landscape

Designs should recognize that threats are increasingly sophisticated and **often target Identity and Access Management (IAM) systems.**

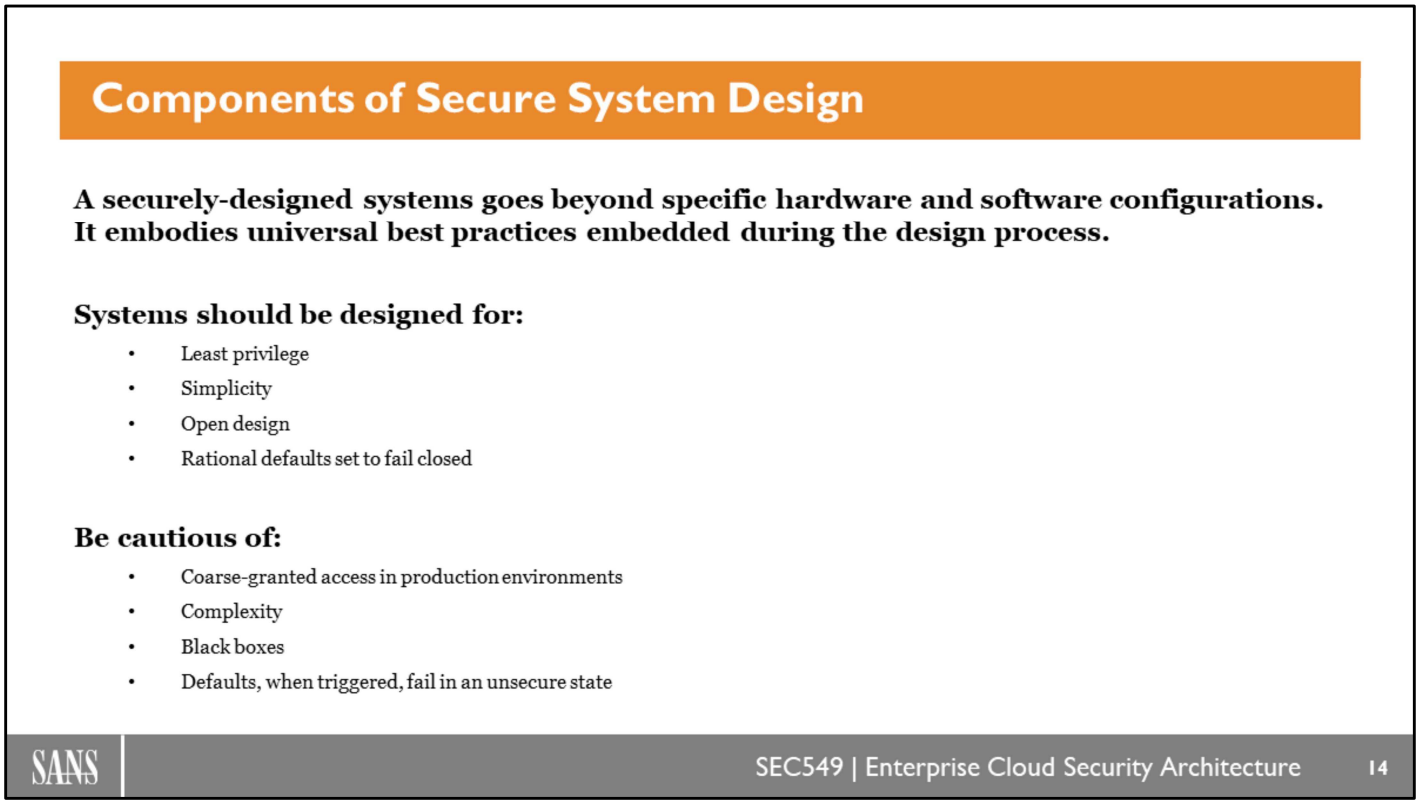
Without network limitations placed on cloud services, IAM becomes ***THE PRIMARY CONTROL*** preventing unauthorized access to the system.

SANS | SEC549 | Enterprise Cloud Security Architecture 13

In any type of cloud service model, the priority measure is to engage in a security strategy that considers threats targeting *Identity and Access Management (IAM)*. Without network limitations placed on cloud services, IAM becomes *THE MAJOR WAY* to prevent unauthorized access to the system. The challenge for the cloud security architect is that each Cloud Service Provider and SaaS vendor has developed their own IAM model. This leads to a complex IAM ecosystem and attack surfaces that are remarkably different from on-premises systems.

Success in thwarting Identity-based attacks has come a long way since the inception of Active Directory (AD). There are now cloud-based Identity-as-a-Service (IDaaS) and identity directory providers (IdPs), but these come with their own unique methods for implementing RBAC and their own unique jargon. IAM systems are now so different that separately defined security controls, security policies, and standards may be necessary.

Reference:
Image by PublicDomainPictures from Pixabay https://sec549.com/id182



A Secure Design follows these Principles:

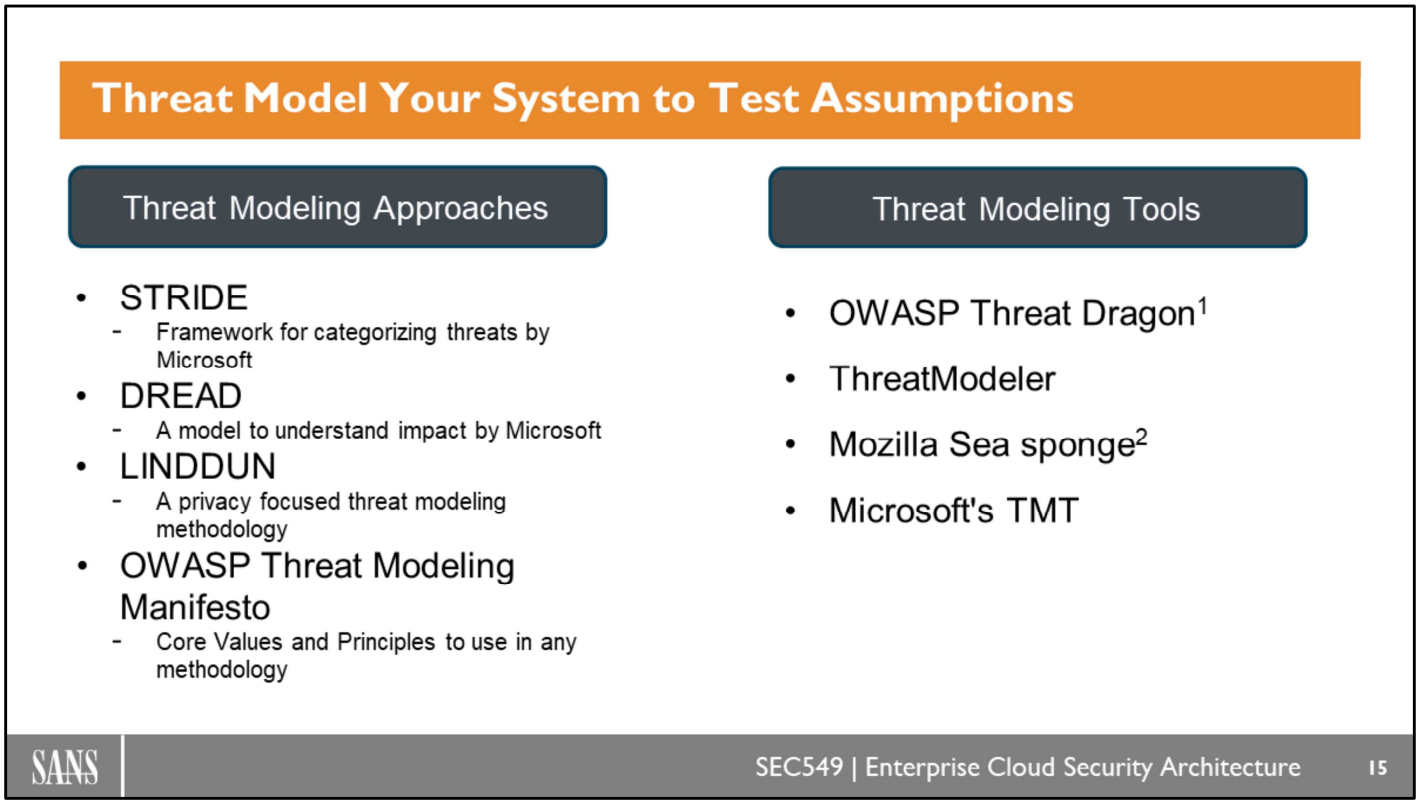
- ***Least privilege*** **–** this means you intend from the beginning to set up a system where no program or user gets privileges they do not absolutely need. By limiting privilege, you may not automatically eliminate attacks; however, you are limiting the impact or 'blast radius' when they happen.

- ***Simplicity*** **–** even if the system itself is complex, its security design should strive for simplicity. Complexity is often called the enemy of security, one reason being that complexity makes it more difficult to reason about a system.

- ***Open design*** **–** do not rely on black-box secrecy to maintain the security of a system. In short, security through obscurity is generally not a foundation of good design.

- ***Safe defaults that fail closed*** **–** ensure that configurations, whether in an application, network, or cloud control-plane, are set by default in their most secure setting. Should errors or an unexpected event occur in the system, the configured setting should default to its most secure state.

An overriding factor for any secure system is that it is easy to use. The human interfaces must be designed so that routine use will be simple enough for users to be incentivized to use systems securely.



By definition, Threat Modeling[3] is a systematic approach to undercovering and documenting threats to your systems. The process can help your organization shift from being reactive to becoming more proactive to threats. Over the years, several approaches to Threat Modeling have emerged, along with tools to help either an individual architect or a broader team collaborate in the threat modeling process.

Threat Modeling Approaches

The STRIDE[4] methodology might be the most well known of the documented processes for threat modeling. STRIDE is an acronym for the various categories of threats which can be enumerated when modeling the risk posture of a system:

- Spoofing - a category of threats related to authentication
- Tampering - threats affecting integrity
- Repudiation – threats related to non-repudiation of actions
- Information disclosure - threats affecting confidentiality
- Denial of service (DoS) - threats affecting availability
- Elevation of privilege - a category of threats related to authorization

DREAD[5] is often used in conjunction with STRIDE to help understand the risk associated from a threat identified during the STRIDE process. The following factors are considered to analyze and ultimately score the identified threat:

- Damage: How big would the damage be if the attack succeeded?
- Reproducibility: How easy is it to reproduce an attack?
- Exploitability: How much time, effort, and expertise is needed to exploit the threat?
- Affected Users: If a threat were exploited, what percentage of users would be affected?
- Discoverability: How easy is it for an attacker to discover this threat?

As with all models that attempt to quantify risk, the risk rating scoring will be subjective. One can attempt to systematize risk scoring by asking questions like "is the identified gap exploitable externally or would the attacker require a privileged position to access?" Even with a system in place to objectify risk, the ultimate rating of impact can vary greatly dependent on the background of the practitioner and the context of the broader organization the system resides in.

The LINDDUN[6] approach is an acronym representing the various threats which are cataloged in the process:

- Linkability
- Identifiability
- Non-Repudiation
- Detectability
- Disclosure of Information
- Unawareness
- Non-Compliance

OWASP Threat Modeling Manifesto[7] is not a full blow methodology as in STRIDE, DREAD or LINDDUN. Rather it provides core principals to follow when threat modeling, irrespective of the methodology you are following.

With these foundational truths in place, it is believed that more stringent methodologies can be iterated on or even adapted to fit your circumstances. The 5 values of the Threat Modeling Manifesto speak to a culture of an active, collaborative process, rather than a passive activity:

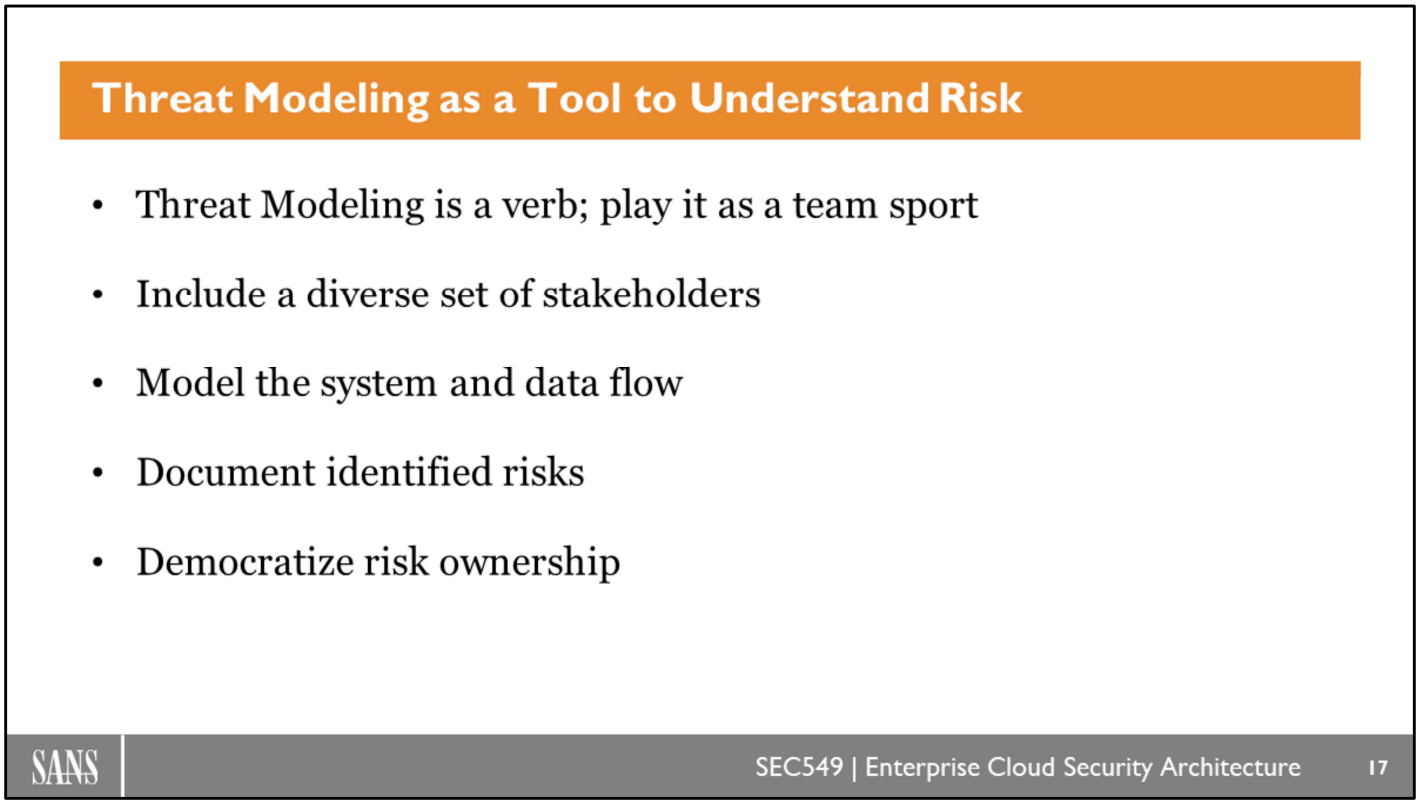
- **A culture of finding and fixing design issues** over checkbox compliance.
- **People and collaboration** over processes, methodologies, and tools.
- **A journey of understanding** over a security or privacy snapshot.
- **Doing threat modeling** over talking about it.
- **Continuous refinement** over a single delivery.

Threat Modeling Tools

Several tool exist in the marketplace to aide in the process of collaborative threat modeling. Some are open source such as OWASPs Threat Dragon and Mozilla Sea Sponge, while others are more integrated, pay-to-play platforms such as ThreatModeler, which operates on a subscription basis. Whichever tool you use should be flexible enough to allow you to adapt to the complex nature of the threats in the cloud.

References:

[1]: https://sec549.com/id183
[4]: https://sec549.com/id186
[7]: https://sec549.com/id189
[2]: https://sec549.com/id184
[3]: https://sec549.com/id185
[5]: https://sec549.com/id187
[6]: https://sec549.com/id188



Optimizing Threat Modeling
IT systems are now so highly complex and need to account for such high numbers of use cases that an unstructured approach to threat modeling enterprise workloads is simply not the best place to begin planning for adequate security threat mitigation. The time to look at threats is from the outset of any project, when the relative costs are lower (but it is never too late to start somewhere).

Threat modeling belongs in the design phase of any project. It needs to happen before any code review, code analysis, and pen-testing. When architecting a solution, your threat model helps you consider threats to the system and build controls into the design when you are in the early planning of any system. Just because a threat model will likely be updated and revised along the way, doesn't mean you don't consider threats to your system as early as possible.

Getting Started with Threat Modeling
Step 1: Assume this is a team "sport". Assemble a cross-disciplinary team of app developers, security experts, members of the Ops team, site reliability engineers, and owners – all brainstorming possible threats in a collaborative effort. Anyone who knows the ins and outs of a system is potentially a good member of this team.

Within the team, ask yourselves who plays which roles. Who on the team understands the business's objectives and which team members understand how the workloads are designed? Encourage each of the threat modelers (irrespective of their backgrounds) to play the part of the "adversary", searching for every possible design flaw in the workload, and play the "defender", devising the needed security controls.

Step 2: Create a model of the system. Consider a model that encompasses a large-enough boundary without becoming too wieldy to threat model. In your model, look at these factors:

- What are the trust boundaries surrounding the system (and within it)?
- Who are the actors that must cross the various trust boundaries?

- How does information flow inside and out of the trust boundaries?
- Identify locations in the system where information has persisted.
- What impact would a successful exploitation of the system have on the business?

Step 3: Plan how to mitigate each threat. Are there obvious fixes or large security gaps that must be addressed within the architecture? While security misconfigurations are common, they often can be mitigated if identified early. Rank the severity of each threat you have identified before planning a larger strategy.

There are many options for brainstorming threats. You can use the STRIDE model (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) or refer to an internal threat catalog to get ideas and see if any fit your current system.

Step 4: Document risks and assign individuals as "risk owners". This enhances accountability as the team identifies which operations or automated security processes can prevent further vulnerabilities down the road. For example, processes like public key authentication and role-based access control keep more of the system secure with the greatest efficiency (and least resource expenditure).

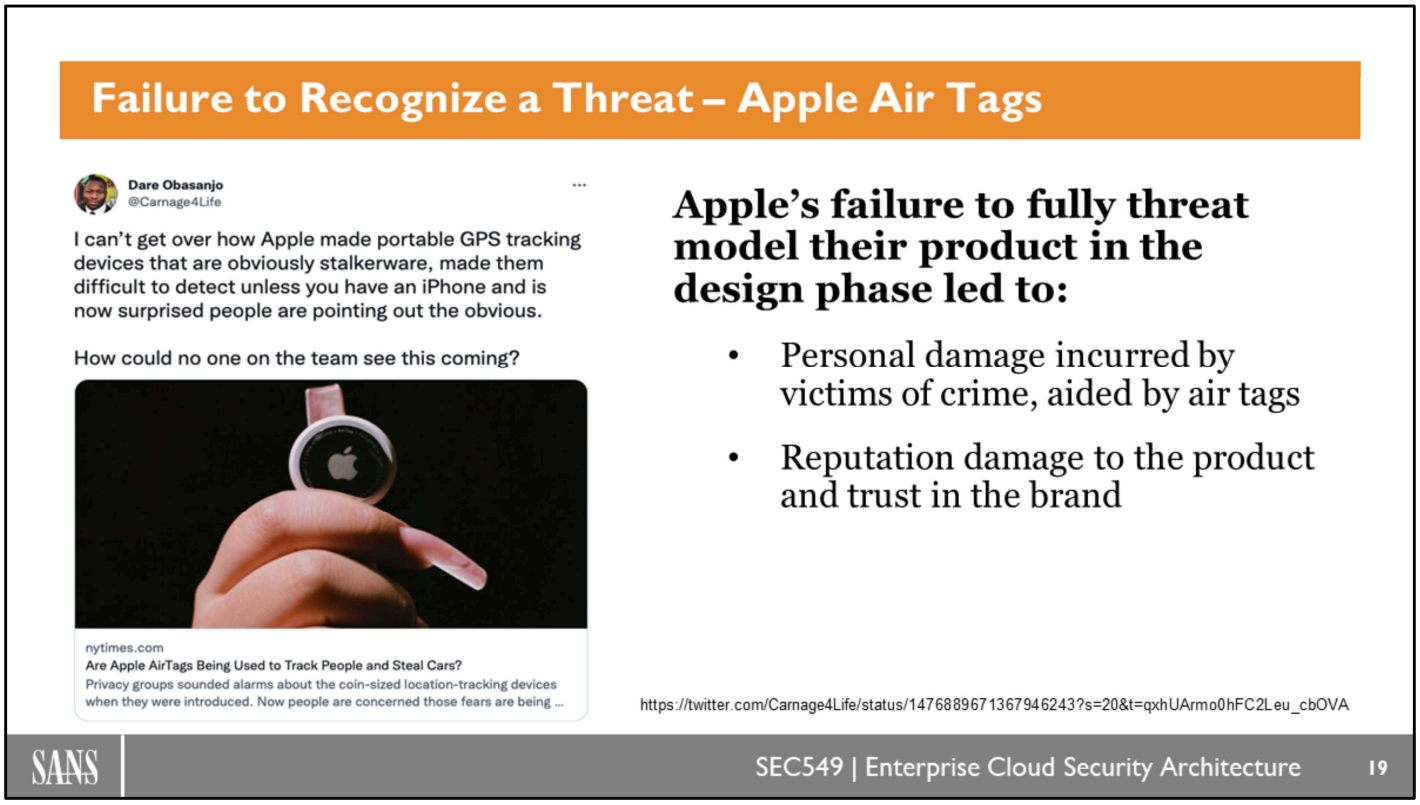
Maintaining a central "risk owner" is problematic in many ways, mostly because it creates its own innate person-centered vulnerability. By empowering every team member AND giving them ownership of a threat, bottlenecks are less likely to occur. Ownership is best given to those who understand the workload features, are already part of the design team, and become empowered to mitigate risks.

Tips for Success in Threat Modeling

1.Have a consistent and scalable approach to threat modeling. Use the same systematic approach for every threat. The first step to consistency is being uniform in how each threat is identified and ranked.

2. Make use of existing collaboration tools. Don't reinvent the wheel when developers are already using an agreed-on productivity suite. Having threat modeling documentation treated as living documents within existing tools makes it far easier to track, review, and work asynchronously. This is analogous to having an agreed-upon official language for any international team of experts.

3. Where possible, break the system into separate features that are evaluated with each threat modeling exercise. Detailed threat models on smaller workloads are more successful in identifying mitigating strategies. The process also becomes more scalable and reusable on other similar features. It is more likely that mitigations identified will be reusable across the board when the system is subdivided into its component parts.



Failure To Recognize a Threat: The Apple Air Tag Debacle

Threat assessment involves looking at the possible areas of software and hardware threats from 'every angle'. Sometimes the angle is closer to the perspective of the villain than everyone imagines. The tech giant Apple made that error recently when it debuted its Apple Air Tags in the spring of 2021.

Apple Air Tags are small coin-shaped Bluetooth beacons using broadband technology[1]. They have been marketed as anti-theft accessories. The idea is simple: attach an Air Tag to anything you might lose – your luggage, dog collar, keychain, or child's backpack. Techies everywhere quickly expanded the device's anti-theft message writing blogs, including all sorts of creative ways to use them for entertainment.

Soon after launch, the more criminal elements of our modern culture found them to be equally helpful[2]. While iPhone users could be notified that the tracker belonging to someone else was moving with them (even if they couldn't find it), Android users would be unable to do so. This left people vulnerable to stalkers who could more easily track them. Instead of protecting your car against theft, some carjackers instead made use of them for their own purposes, hiding them somewhere on the vehicle.

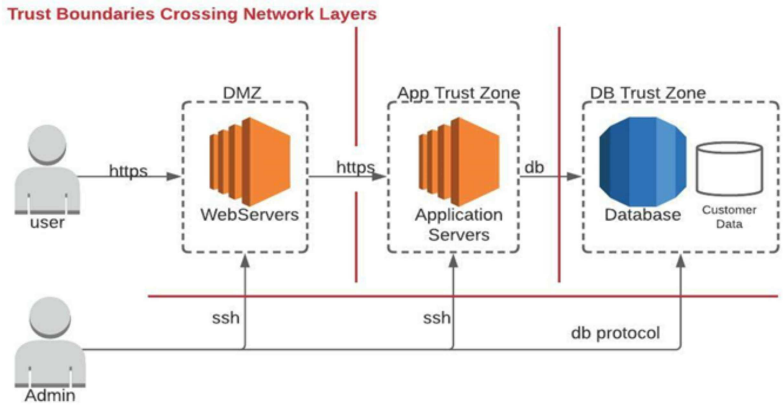
The 'intimate threat model' was completely missed by Apple threat modelers. Insufficient controls and guidance to consumers at the time of deployment led to damaging to Apple's reputation, among other things. There is no evidence to date that anyone died as a result of this failure to threat model adequately but certainly some victims have been harmed.

Apple itself incurred damage to its brand, simply because its design engineers had a failure of imagination. This debacle has since resulted in a 'patch' of sorts. Android users can now download an app called 'Tracker Detect' so that air tags can be scanned for in the individual's proximity[3]. The company also provides information on how to stay safe in the era of Apple Air Tags[4].

References:
[1]: https://sec549.com/id1
[2]: https://sec549.com/id2
[3]: https://sec549.com/id3
[4]: https://sec549.com/id4

Trust Boundaries

Trust Boundaries represent the change in trust levels between zones. This is often where the attack surface lies.



SANS | SEC549 | Enterprise Cloud Security Architecture 21

What is a Trust Boundary?

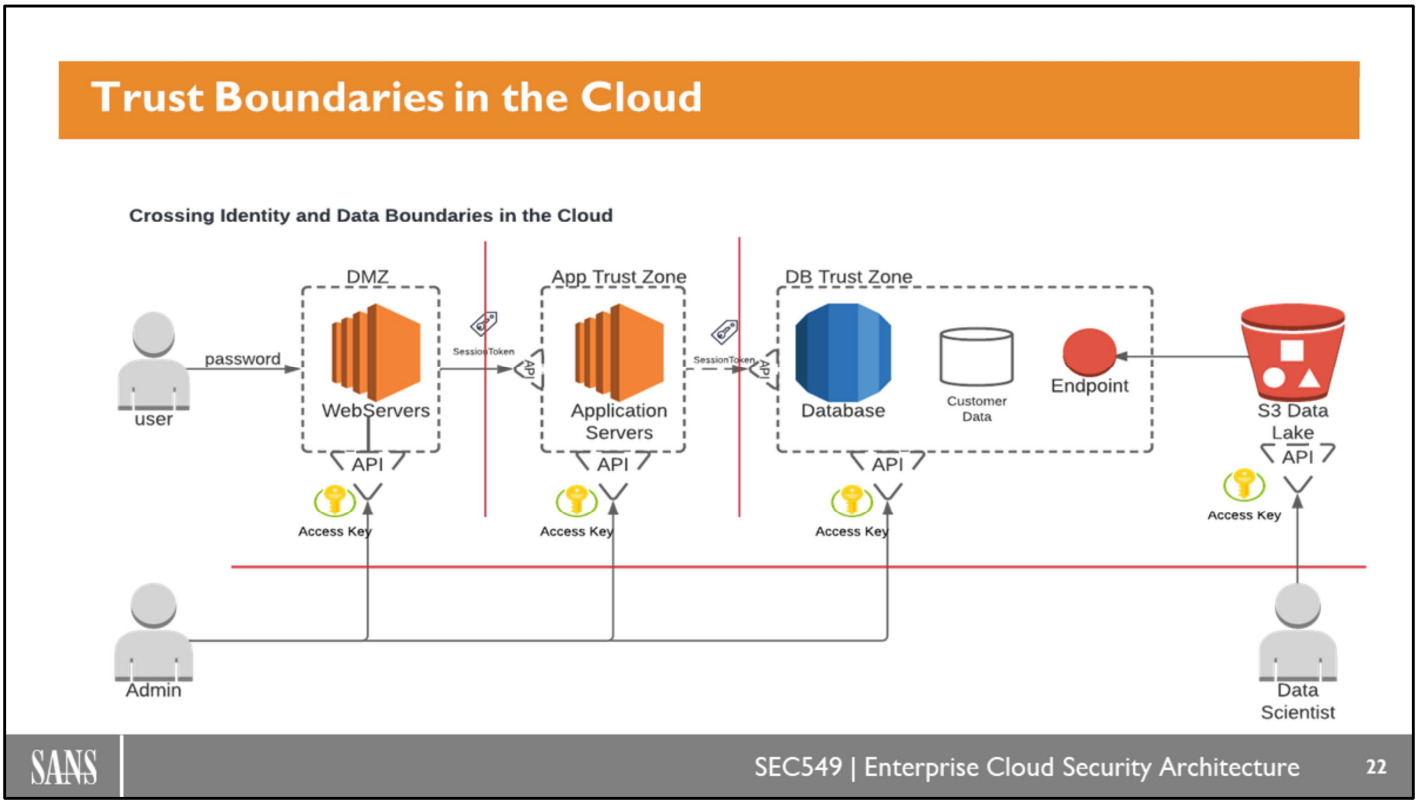
Trust Boundaries: Trust boundaries in computer systems represent a change in the level of trust as pieces of data flow between different trust zones.[1]

Trust Zones: Trust Zones are components within a bounded area where all resources in the zone are implicitly trusted and whose perimeter is policed with some form of access control.

It's at trust boundaries where attackers are likely to exploit weaknesses and vulnerabilities, such as flaws in access-control logic. This is why it is so important to identify these boundaries on any system as they indicate a likely attack surface.

There are several types of Trust Boundaries, including the physical boundary between machines and more abstract boundaries, such as privilege boundaries, integrity boundaries, and network segmentation boundaries.

Reference:
[1]: https://sec549.com/id5



Trust Boundaries in the Cloud
While classic on-premise systems might only need to be designed for access-control around their network perimeter, in the Cloud, systems are abstracted and access curated by the CSP. This results in more boundaries to be aware of, including the network layer, data access layer. and the identity layer.

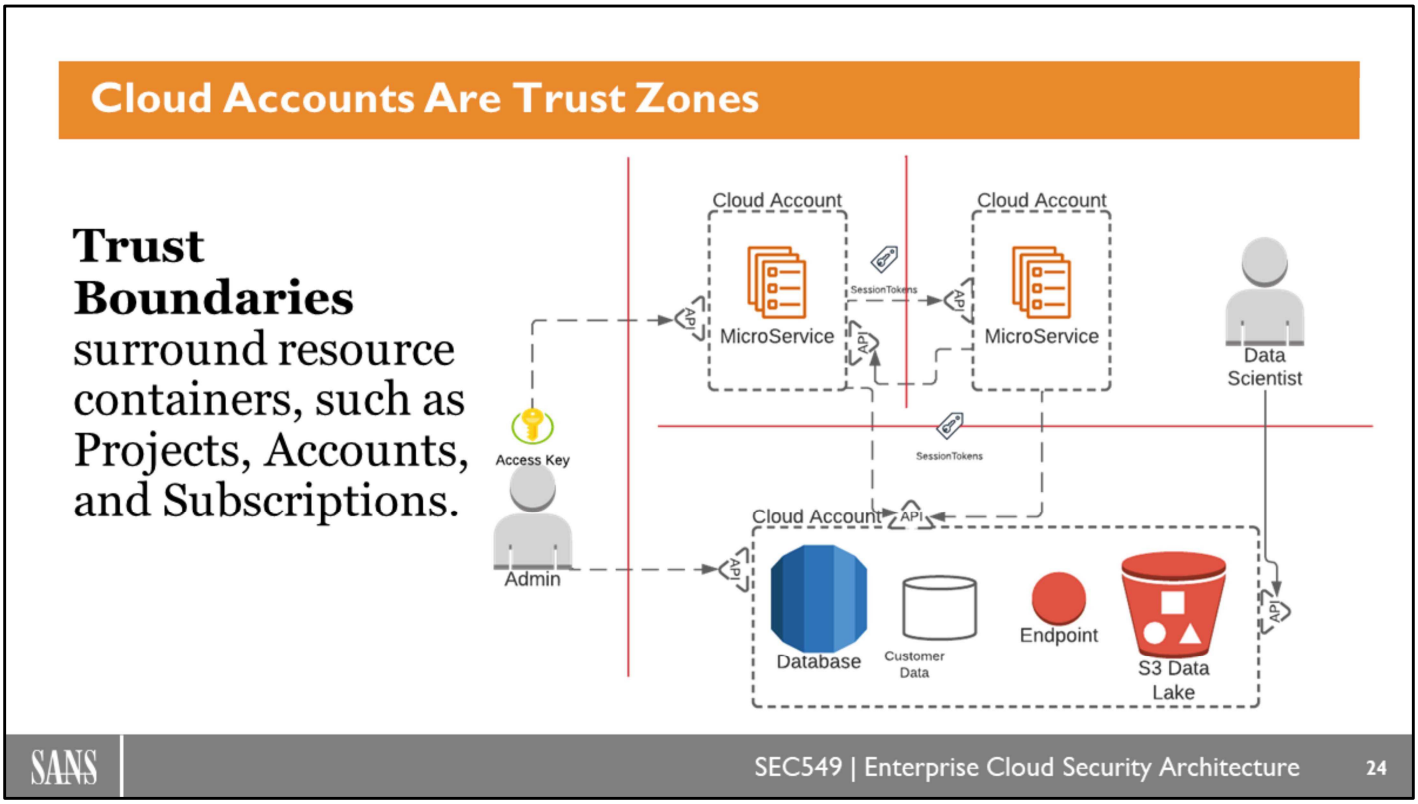
Network Layer
In all of the major clouds, private networking blocks can be created that mimic your on-premise network. Traffic within cloud networks originates from or is destined for applications running on classic Infrastructure-as-a-Service (IaaS) products such as AWS EC2, GCP Compute, or Azure Virtual Machines. Policing the network layer in the cloud will feel very similar to access controls in on-premises systems. Network Firewalls, Web Application Firewalls (WAFs) and Network Access Controls all have cloud-native cognates to apply at this trust boundary.

Data Access Layer
Data warehouses that rely on primary network boundaries for access control are being traded in. As more data is migrated to serverless, distributed cloud-hosted repositories, the need to architect for a data boundary has emerged. Data moving in and out of cloud-hosted repositories can traverse the open internet or can be architected to be accessed over your CSP 'back-bone' network, adding a layer of security to otherwise public services.

Identity Layer
CRUD operations (Create, Read, Update and Delete) are performed on your cloud-hosted resources through curated APIs exposed by your cloud provider, all of which require the end user be authorized for the action. As such, the identity layer plays an outsized role in cloud access control. In contrast with on-premise systems that might have some inherent trust when within a trust boundary, all cloud APIs operate by default in a zero-trust model, requiring authentication and authorization by all callers.

The complexity involved in policing the identity layer in the cloud has been a contributing factor to many cloud breaches, including the famous **Capital One's** data breach incident that exposed the records of almost 106 million customers[1] It's no surprise that architecting for a defensive identity layer is a core topic of this course.

Reference:
[1]: https://sec549.com/id6



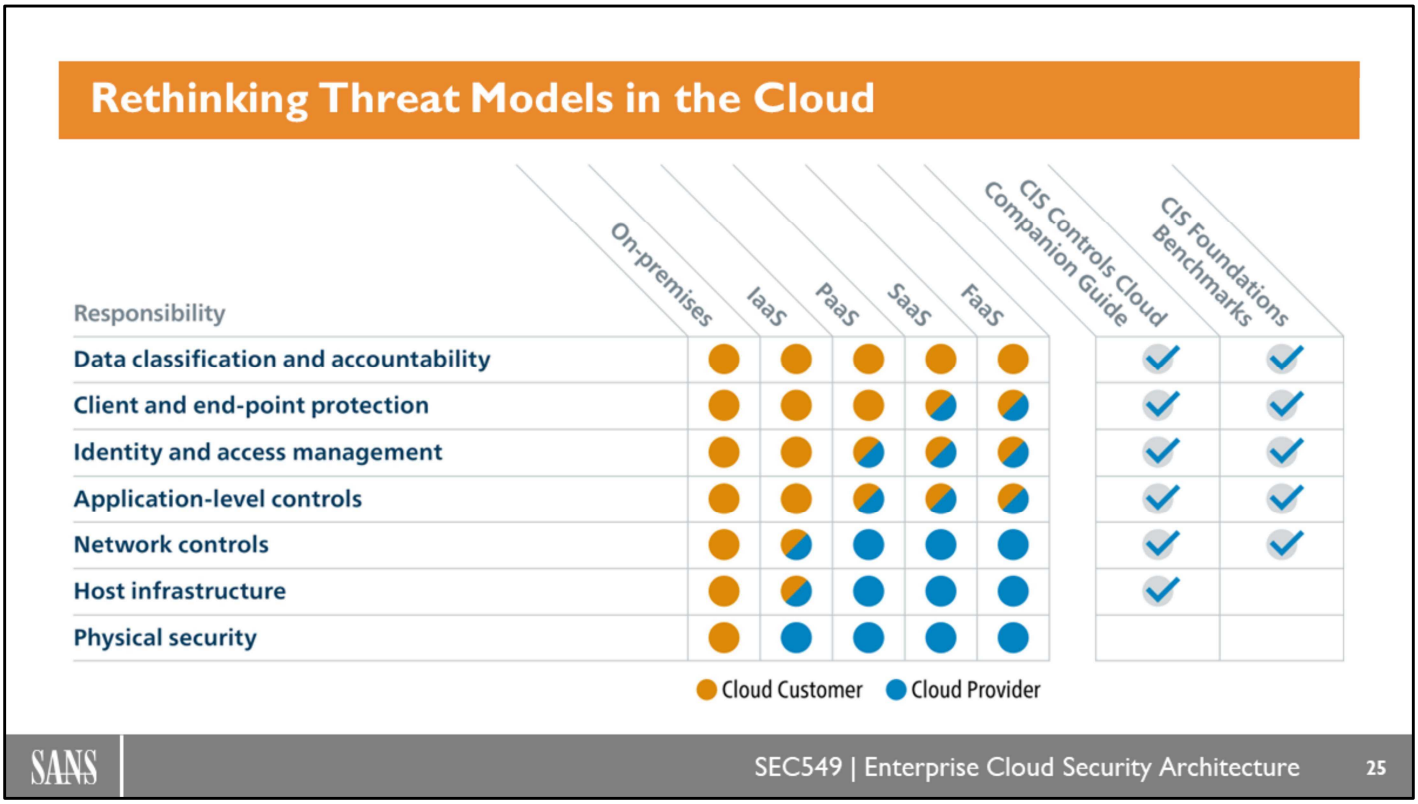
Cloud Accounts Are Trust Zones

When working with the Big-Three Cloud Providers, you automatically have built-in Trust Zones in the Identity Plane. In the slide image shown, trust boundaries can be drawn around several different larger-scale Cloud Resources (e.g., GCP Projects, AWS Cloud Accounts, and Azure Subscriptions). These larger-scale resources in the Cloud create their own natural identity boundaries.

In a GCP-based cloud system, for example, you can assume that a Project boundary is primarily an IAM boundary.

Projects in GCP, Accounts in AWS, Subscriptions in Azure - these are not simply high-level billing containers. They house resources within a context-bound trust zone, surrounded by a unique trust boundary, and separate from other larger scale resources.

Crossing the trust boundaries surrounding large scale resources is accomplished on the *cloud identity plane*. In an AWS-based system, an IAM User with access to Account A can assume a role in Account B, crossing the natural trust boundary between accounts. In Azure, a Function in Subscription X can access a Function in Subscription Y. These natural trust boundaries are easy to see and name; they are also areas where you have some control over who crosses the boundary.



Rethinking Threat Models in the Cloud

The security risks within a cloud environment are unique to the highly networked, concentrated, and shared cloud system. The responsibility for the security of any cloud-based system is inherently shared by the customers and the cloud service providers.

It is insufficient for organizations to think that migration to the Cloud requires no further thought on changing security practices. Cloud Services Providers go to great lengths to ensure that those things they are responsible for are secure. Breaches tend to occur because the customer has misconfigured some aspect of their contribution to the cloud-based system – most often related to failure to encrypt data and a lack of identity control measures or privileged access security within the company.

Ultimately, it is the customer and the architects of their cloud-based system who must know the intricacies of what they send into the cloud and how it is protected. Only through good IAM practices and data encryption can the customer responsibly utilize the services of the CSPs in ways that make their cloud experience both safe and user-friendly. In general, cloud breaches have capitalized on some error or weakness in the configuration of interfaces not protected as they were designed to be.

Reference:
Image Source: https://www.cisecurity.org/insights/blog/shared-responsibility-cloud-security-what-you-need-to-know

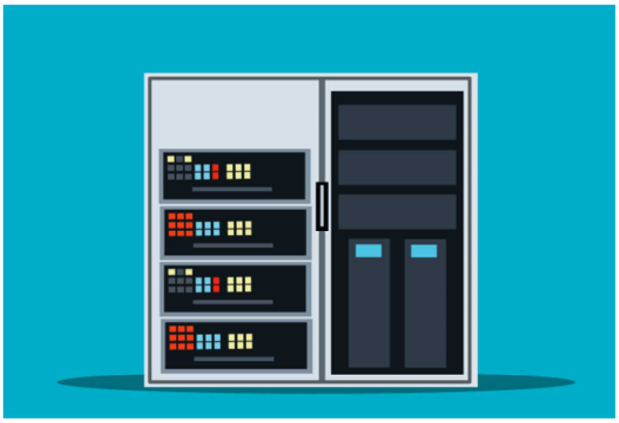
The Traditional On-premises Security Approach

The Traditional Approach Involved:

- Security controls at Layers 3 and 4
- Large network-based Trust Zones

Problems with this Model:

- Inflexible
- Inefficient
- Vendor-specific



The Traditional On-premises Security Approach

The traditional approach to network security will be familiar to any tenured security specialist. It has been relatively inflexible and narrowly focused, mainly because security was based on threats confronted by an on-premises network.

Most enterprises have now recognized the value of some migration to the cloud. Cloud migration – even in a hybrid fashion – leads to improved user experience, enhanced customer market analytics, and/or greater excellence in operations and management. The value added by cloud migration has led to an increase in at least 'dipping one's toes' into the cloud-native environment by many businesses and organizations.

Reference:

Image by mohamed_hassan from *Pixabay*: https://pixabay.com/vectors/big-data-server-rack-data-center-6100853/

Transitioning to Cloud-Native Security

Your Approach to Security Becomes:

- Policy-Centric Security
- Identity-Centric Security
- Workload-Centric Security

Advantages:

- Improved visibility
- Granularity of perimeters
- Improved handling of complexity

Transitioning to Cloud-Native Security Approaches

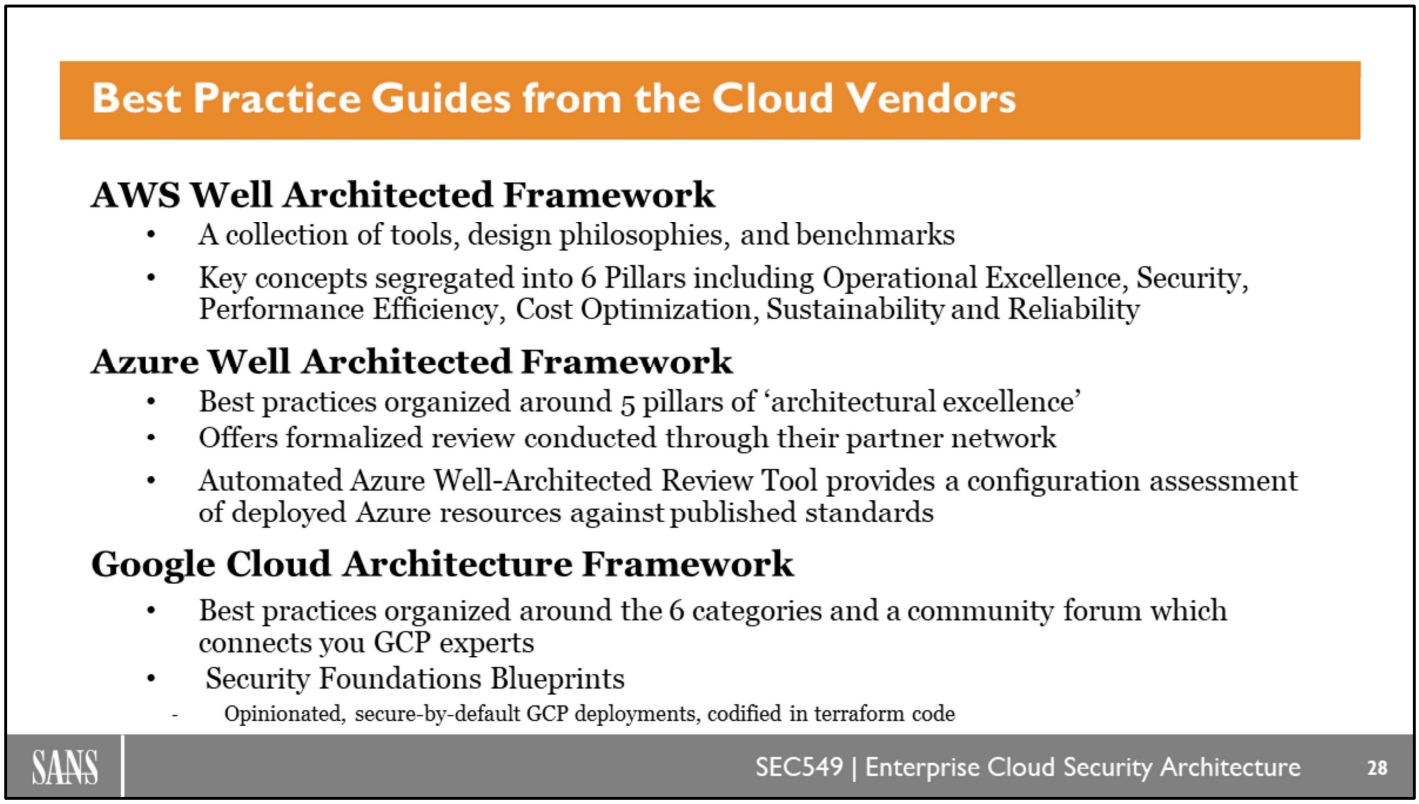
The cloud environment has led to an opening up of new attack surfaces; however, there are many more portions of the tech stack that do not require the business's attention because the responsibility for these falls on the CSP.

Increasing, because so much of the plumbing of a cloud-hosted solution is based on external API calls to the cloud control-plane, the shift is increasingly toward identity-centric and workload-centric security controls. The old 'Steady-Eddie' network controls are still relevant to the larger cloud-native picture, but they must be accompanied by identity-based controls.

This new landscape of cloud-native architecture means that the security and compliance frameworks must be network-independent. In this new world, perimeters have become fragmented, encompassing smaller and smaller workloads rather than one large perimeter around a monolithic application. The natural consequence of this is the necessity for identity-based security, where controls are far more granular, defined in policy, and enforced nearest to the workload.

In short, this is what people mean when they say that the new perimeter in the cloud environment is *identity*.

Reference:
Image by WilliamsCreativity from *Pixabay*: https://pixabay.com/vectors/big-data-server-rack-data-center-6100853/



Best Practice Guides From the Cloud Vendors
All three major cloud service providers have included a set of standards or 'pillars of architectural excellence'. The idea is to have a framework to build high-performing, secure, resilient, and efficient infrastructures for all apps deployed using their services.

The published frameworks involve various aspects of designing a cloud-native infrastructure – security, operational excellence, performance efficiency, cost optimization, sustainability, and reliability. No one would argue that these are all good things to have on your side in any venture. BUT how do these best practice guidelines help inform us on their use of cloud-native technologies?

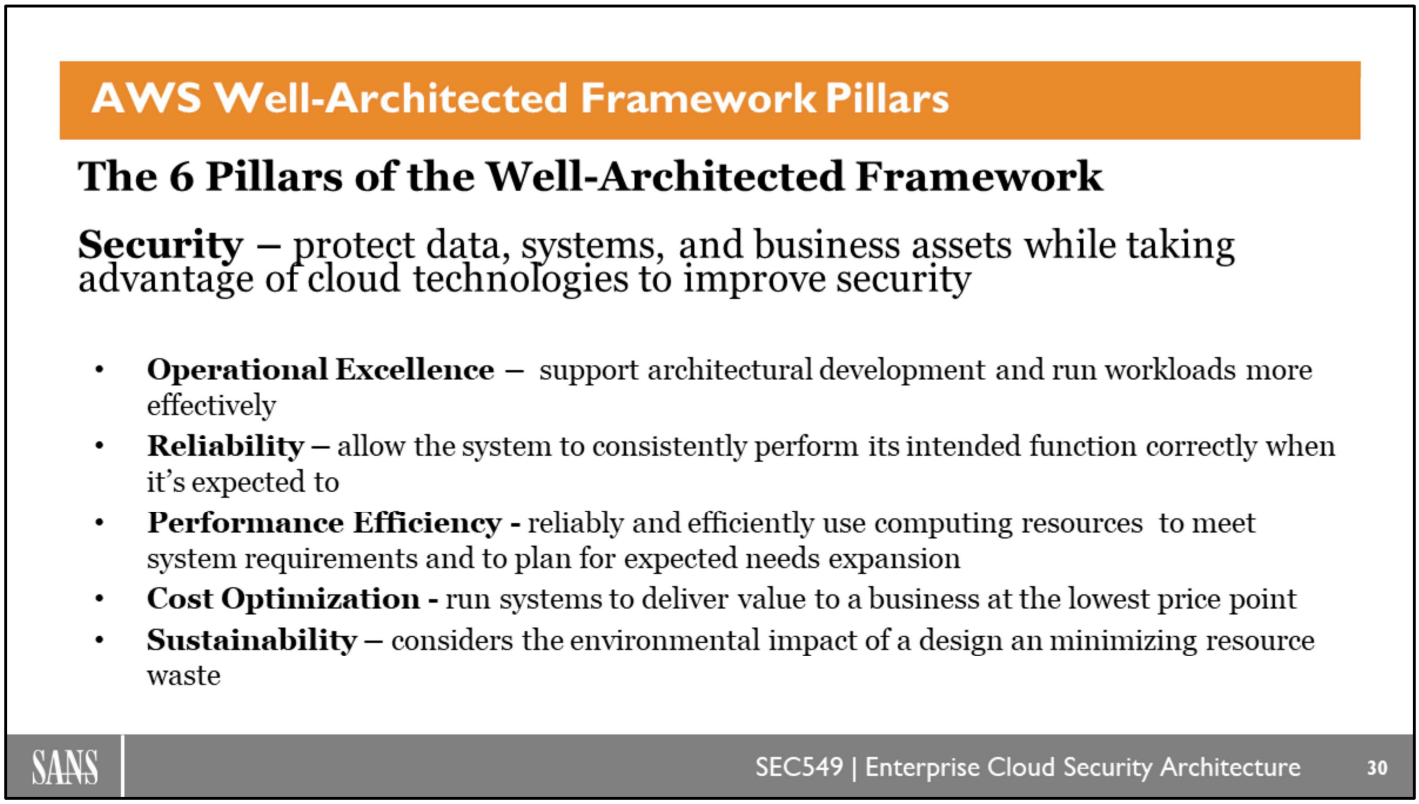
When used properly, these frameworks can help you determine how well your current cloud architecture deployment is measuring up against best practices. As with AWS, review tools are provided allow for more systematic design reviews against industry standards. This AWS Well-Architected Tool[1] is intended to be used in conjunction with an AWS technical account manager or 3rd party auditor to benchmark a cloud migration in progress.

Azure has its own tool called Azure Well-Architected Review[2]. While not providing an interactive tool, Google hosts a Cloud Architecture Community Forum with answers posted by on-staff GCP Cloud Architecture[3] experts working for Google.

Google also came out with a collection of security foundations blueprints[4] in 2021 consisting of terraform code repositories that may be leveraged to deployed, secure-by-default, highly opinionated GCP cloud deployments for those end users needing specific insight into each security decision made in the blueprint design.

The concept to understand here is that all three providers do pay attention to best practices albeit with a focus on single-cloud, greenfield deployments. They each have standards and frameworks of 'excellence'. These frameworks help developers using their cloud know where they are going in any design and offer options for sorting out the 'grayer areas' along the way.

References:
[1]: https://sec549.com/id11
[2]: https://sec549.com/id12
[3]: https://sec549.com/id13
[4]: https://sec549.com/id14



AWS Well-Architected Framework Pillars

The AWS Well-Architected Framework[1] is broken into 6 pillars providing a mechanism to categorize the key elements of AWS infrastructure design. Emphasized in this course are the security implications of design however the Security Pillar is only one aspect of the Well-Architected Framework. AWS expands on guidance within each of the pillars with an accompanied whitepaper, resources and best practices.

Security Pillar

Of interest to the Cloud Security Architect is the Security Pillar of the Well-Architected Framework where AWS provides high-level guidance on identity solutions in the cloud, ensuring traceability, implementing monitoring, automaton, compliance and separation of duties. Becoming familiar with the Security Pillar guidance and its seven design principals is a good 'first step' if you are new to the cloud or AWS design patters as a whole.

Reliability

Also of note is the Reliability pillar which AWS describes as encompassing "the ability of a workload to perform its intended function correctly and consistently when it's expected to."[2] The principals covered in this pillar are of particular interest when designing resiliency in systems and backup strategies.

Conclusion

The AWS Well-Architected Framework is a foundational guide for architectural best practices that is more description, rather than perspective. Most organizations will find the best practices outlined in the Well-Architected Framework to be aspirational goal posts but are not helpful in road-mapping a phased evolution of their current state.

References:
[1]: https://sec549.com/id15
[2]: https://sec549.com/id16

Visualization Tools for Cloud Security Architecture

Tool	Self-Hosted or SaaS	Supported CSPs	Live Load Resources	Freeform Diagraming with Icon Set ?	Free Version	Cost Estimation ?
Inviz	SaaS	AWS	Yes	Yes	Yes	No
Hava	SaaS	AWS, GCP, Azure	Yes	No	No	Yes
Draw.io	SaaS	AWS, GCP, Azure	No	Yes	Yes	No
LucidScale	SaaS	AWS, GCP, Azure	Yes	Yes	No	Yes
Cloudskew	SaaS	AWS, GCP, Azure	No	Yes	Yes	No
Cloudcraft	SaaS	AWS	Yes	Yes	Yes	Yes
Cloudokit	SaaS	AWS, GCP, Azure	Yes	Yes	No	Yes
aws-perspective	Self-hosted	AWS	Yes	Yes	Infrastructure Cost	Yes

The Cloud Architect's Visualization Toolbox

Cloud environments have challenged architects to look beyond their tried-and-true tools like Visio and look to more fully featured tools – ones specifically designed for the cloud when documenting the topology of their environment. Most are sold as a SaaS solution; however, others are self-hosted. The goal is to represent resources and their relationship in simple and clear-cut graphical formats.

Those that have live loading capabilities offer the architect a head start in the diagramming process by pre-populating certain resources and pre-drawing the blueprints. Most also allow the creation of diagrams though a designer interface or through live inventory pulled in from AWS (through a Read-Only Role).

Among the SaaS tools available for visualizing cloud architecture in the design phase:

- ***Inviz –*** a personal subscription is free and allows up to 500 renderings per month. Live resources can be loaded from an AWS account or from cloud formation templates. It does not provide a cost estimation, however.
- ***Hava*** – supports AWS, Azure, and GCP. Your resources are visible through a 'security view' so you can be aware of Ingress Points into VPCs. It has no free options, however.
- ***Draw.io –*** A free version is available and is hosted at https://app.diagrams.net/. There is paid integration with Confluence, and it works with AWS, Azure, and GCP. There is no option for live load resources.
- ***LucidScale –*** it has all the bells and whistles but is not free. You can export diagrams as terraform code and it works with AWS, Azure, and GCP. Like most of the paid services, it provides cost estimates based on the diagrams.
- ***CloudSkew*** – a free option is available for beginners, and it works with AWS, Azure, and GCP. The downside is that there are no live-loading capabilities. You need to use the icon-sets of the Big-Three to manually craft diagrams.
- ***CloudCraft*** – this option allows you to export diagrams as draw.io files and can combine blueprints that have been seeded with live resources. It only works with AWS, however.
- ***Cloudockit*** – this works with AWS, Azure, and GCP, and additionally supports O365, Kubernetes, and VMWare resources. It does not have a free version, however.

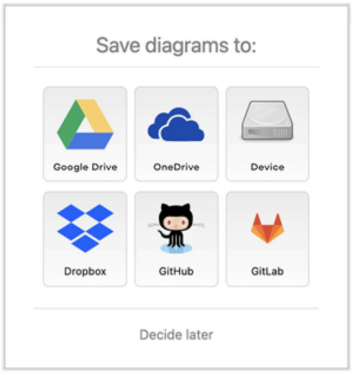
The main self-hosted visualization tool is AWS-Perspective[1]. This is AWS's version of an open-source project. There are infrastructure costs involved that add to approximately $400 per month. The tool is fully compatible with all AWS Resources that are supported with the AWS Config Service. It allows for visualizations based on live data from AWS.

The Downside of Most Options: With most SaaS tools offering live loading capabilities, only a limited set of resources is available – generally only networking resources, like IaaS, VPCs, and storage resources. You will not have reasonable access to identity-based resources. Users, Roles, Policies, and Accounts are not depicted on auto-generated diagrams.

Reference:
[1]: https://sec549.com/id17

Introduction to Draw.io

When you first navigate to draw.io, the application will ask you where you want to save diagrams. You can configure this now or decide later.



To ensure you have access to all cloud relevant shapes, click on 'More Shapes' and add in elements from the cloud providers.

Draw.io Tutorial

The open-source software, Draw.io[1] can be accessed at the URL, https://app.diagrams.net/.

1. If you've never visited draw.io before, you will be asked where you want to store your diagrams and then if you want to begin a new diagram or continue a preexisting one.
2. To create a new diagram, click 'File' -> 'New…'. Your page will pop up. Name the document.
3. Use the icons available on the left-hand side of the screen to select the icon you wish. Drag, drop, and resize it if needed. Move by clicking it and dragging it.
4. If a connector is desired, select from the connector list at the top. You can drag and drop an arrow, for example, to the edges of the two icons you want to connect. That connection will remain, even if you move one of the icons.
5. You can select a grouping by drawing a box around the items you wish to group. Right-click on the selected grouping and click 'Group'. They will move together unless you Ungroup the selection.
6. You can copy any icon you wish and can use the text section on the right. Images and boxes around them can be filled with any color.
7. All your changes are saved automatically to the storage site you chose. You can also export your diagram in other formats, including png, html, jpg, and pdf. You can turn off the grid to make a professional-looking diagram.

Reference:

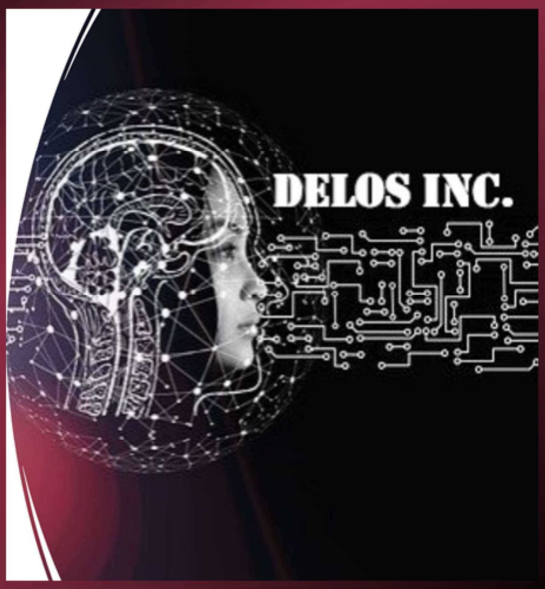
[1]: https://sec549.com/id18

Welcome to Delos International!

Scenario:

Delos is a multinational robotics company with large research and development investments. The c-suite has just announced its intention to move most of Delos' operations to the AWS cloud over the next few years.

Groups of cross-functional leaders are collaborating on the initial planning steps of the process, including the design of their future AWS Organization resources.



Delos is a multinational robotics company with large research and development investments. The C-Suite has just announced its intention to move the majority of Delos' operations to the AWS cloud over the next few years. Groups of cross-functional leaders are collaborating on the initial planning steps including the design of their future AWS Organizations resources.

Reference:
Image Source: Pixaby user geralt: https://sec549.com/id51

Delos' Business Objectives

The enterprise has two major objectives for their cloud migration:

- **The Problem:** They have a handful of small but critical internal applications still behind a VPN, but the VPN capacity limits their overall productivity. They want your help in moving these internal apps to the cloud using a Zero-Trust model for accessing them.
- **First Objective:** Their workforce has become increasingly globally-located. They want their company to support "Remote-First" operations as soon as possible. They have an Office365 subscription for remote users to access their mail and business suite applications.
- **Second Objective:** The company needs to harness the on-demand compute power the Cloud can offer. They invest heavily in R&D and their team of scientists are very eager to leverage Cloud Computing for AI and Robotics research.

This page intentionally left blank.

Lab 1.1 – Threat Modeling S3

Estimated Time: 30 minutes

The Incite team has stood up an external S3 bucket as an easy mechanism for third-parties to ingest data points into their system. You've been brought in to consult on this ingestion pattern and address any immediate risks. As a first order of business, we will stretch our threat modeling legs as we attempt to answer the question "What could go wrong?". Answering this question formally while using the STRIDE methodology will help drive the implementation of mitigations and controls.

Preparation

- View **Lab 1.1** in the course workbook
- Open the workbook in an incognito window
- https://workbook.sec549.com
  - Username: H03-student
  - Password: million-sailor-DEAR

Objectives

1. Review the description of the Attack Surface. Points where threats could occur in the systems are enumerated. Consider these trust boundaries when reviewing possible threats.
2. Review the possible threats in your Toolbox. Consider each threat for plausibility given the system design and permissions granted to external parties.
3. Only six of the twelve threats will be applicable to the system. Drag and drop the threats that are applicable the attack surface. Place them into their STRIDE category.

This page intentionally left blank.

Lab 1.1 – Summary

In this lab, you...

- Reviewed the data flow diagram of an external ingestion pattern using S3
- Evaluated the Attack Surface and the Trust Boundaries crossed when data is both written and read from S3
- Considered possible threats to the system and identified which were plausible given the design

You've successfully threat modeled your first cloud architecture!

Many traditional threats seen in on-premise architectures are not applicable, such as those relating to Denial of Service and those mitigated by encryption-in-transit. Some risks involving the access keys are mitigated given the limited (write-only) access provisioned to third-parties. However, there is still a possibility that threats could manifest in the bucket the AWS account is housed in.

An attacker (or someone negligent) could escalate their permissions and gain access to the bucket, affecting the logging of actions taken and/or the confidentiality of the objects stored in S3. We haven't been made aware of any input validation or data schema being enforced on ingestion, and as identified in the threat modeling exercise, there is the possibility for malicious or corrupted data to affect downstream, on-premise systems.

Course Roadmap

- *Section 1: Cloud Account Management and Identity Foundations*
- Section 2: Implementing Zero-Trust In The Cloud

SECTION 1

- Security Architecture in the Cloud
  - Threat-Modeling the Cloud
  - Cloud-Native Security Models
    - Lab 1.1: Threat Modeling S3
- ***Federated Access / Single Sign-On***
  - Managing Users at Scale
    - Lab 1.2: Centralizing User Provisioning
- Creating Hierarchical Cloud Structures
  - Designing for Policy Inheritance
    - Lab 1.3: Structure an AWS Organization
- Implementing an Identity Foundation
  - Granting Access to Cloud Resources
    - Lab 1.4: Transition AWS Access to Roles

This page intentionally left blank.

Takeaways

In this module, we'll begin with a level-setting primer on identity federation, single sign-on, and the protocols used in these technologies.

We will then move on to the importance of centralizing identity for both your workforce users and their group memberships and attributes, along with the risks of a fragmented IAM system.

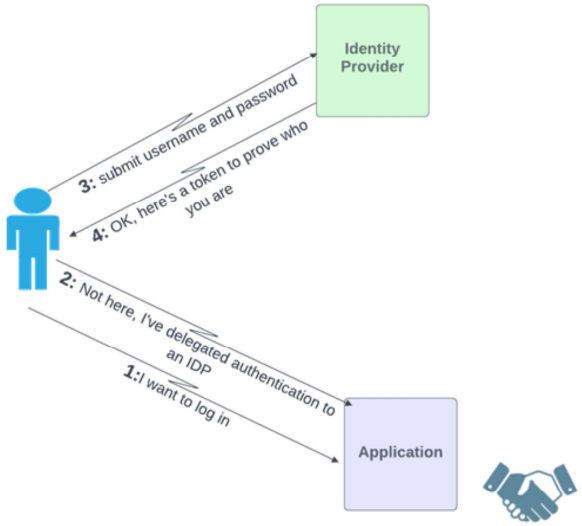
Finally, we deep dive into the service AWS Identity Center, and look at how to federate against your IdP, provision IdC users, and use Identity Center (IdC) to manage the permissions of your end users in AWS.

This page intentionally left blank.

What Is Identity Federation?

Federation

- When an application delegates the authentication of end users to a central identity provider
- Communicating an end user's authenticated state from the Identity Provider to the application is done through protocols such as SAML/2 or OIDC



Identity Federation is one of the few technologies where security and ease-of-use exist in the same pattern. Without Identity Federation, all authenticated applications would require a username and password, encouraging password reuse. When applications delegate their authentication to a centralized system, the end user experience is consistent and allows for fewer, hopefully stronger, passwords.

All Identity Federation systems consist of two parties. A typical federation system involves 1) a Service Provider that is configured to trust a 2) centralized Identity Provider (IdP).

Identity Provider (IdP) — a system that provides the primary authentication for an application. An IdP can provide a user with identifying information and serve that information to services when the user requests access.

An IdP requires a basic set of capabilities. At the highest level, these include:

- Authenticating end users
- Communicating the authenticated state of end users to replying applications, formally called Service Providers

An IdP often includes many more directory and user management features, such as:

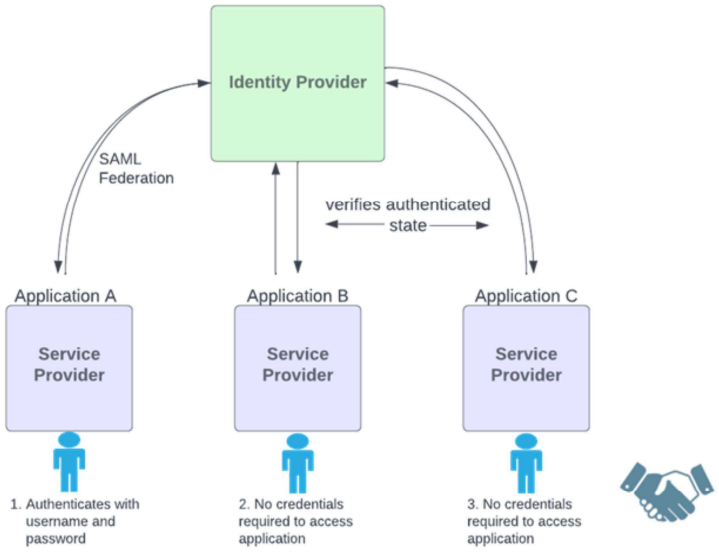
- User and group directories, including provisioning and deprovisioning
- MFA options
- Access Request and Approval workflows
- Password Management

Service Providers (SP) — applications configured to rely on an IdP for authentication. In some cases, SPs are referred to as *Relying Parties* but in this course, we'll stick to the term Service Provider. Signed authentication credentials are passed to the SPs from the Identity Provider. It is an SP's responsibility to verify the authenticity of the credentials provided by validating the signature in the provided credentials.

What Is Single Sign-On?

Single Sign-On

- Login once, Login to multiple apps
- Identity Providers can be configured to **transparently authenticate** end users to Application A if they are already authenticated to Application B
- Often accomplished with browser cookies representing the end user's authenticated state



Single Sign On (SSO) is an authentication process that transparently authenticates end users to multiple systems based on previous authentications. Single sign-on requires a centralized identity provider to maintain a view of the authenticated state of the end users and convey that authentication to service providers.

Single Sign-On in the Enterprise

An enterprise might have hundreds of internal workforce applications. Instead of having employees log in every time they need to access an application, the apps can be grouped together and configured to trust one another. When grouping applications together, be sure they have a consistent trust level.

- *Example:* If one application in an SSO group requires MFA to authenticate, all applications in the group should have the same requirement. Otherwise, lower risk applications could enable an unintentional Multi-Factor Authentication (MFA) bypass when coupled with applications with more stringent requirements.

Identity Federation versus Single Sign-On

Single Sign-On

- Shared authenticated user context between applications - brokered by a common identity provider

Federation

- Trust between an application and identity provider
- Delegation of authentication from an application to an identity provider

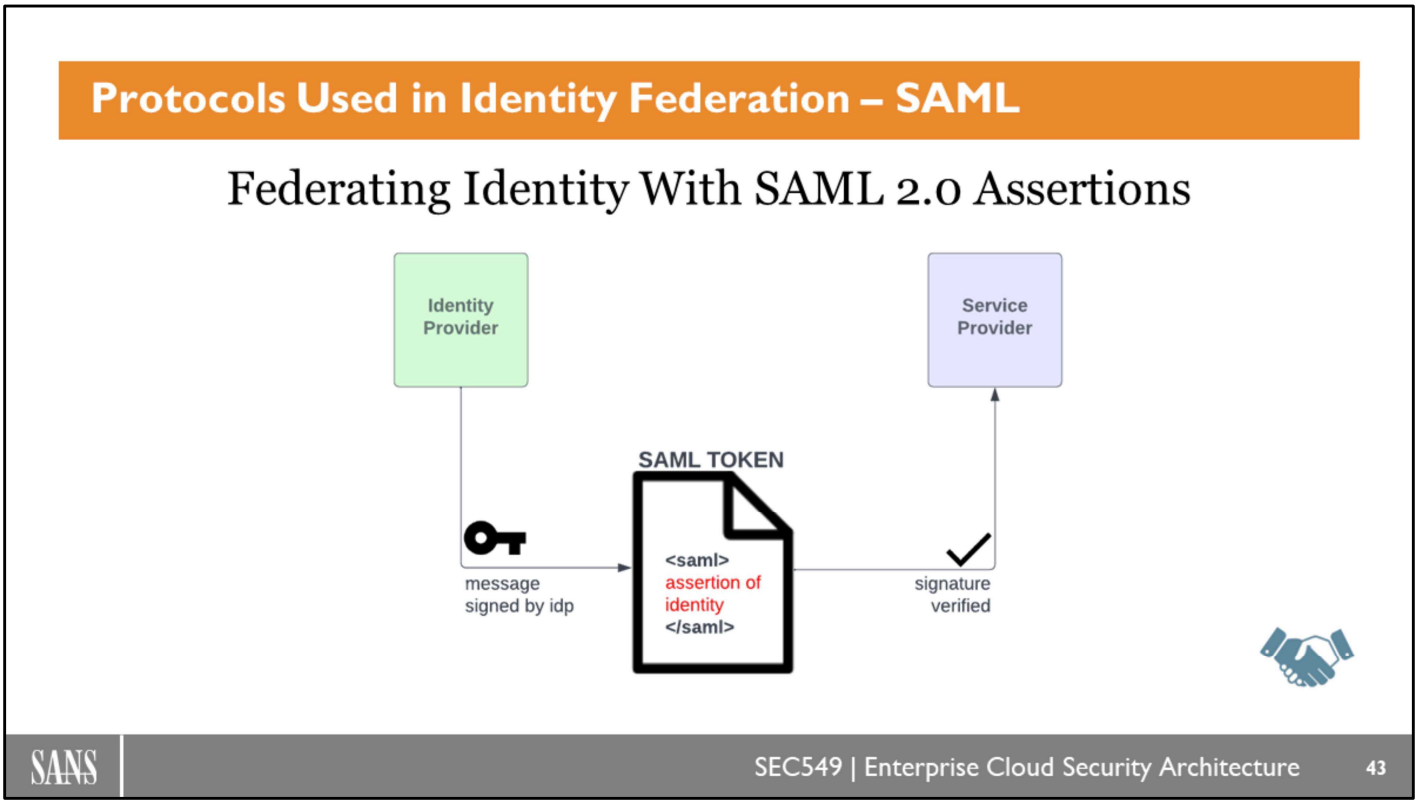
Identity Federation Versus Single Sign-On
Although Single Sign-On and Identity Federation are often used together, they do not mean the same thing.

Federation – Identity Delegation
Federation is a contract between an Identity Provider and a relying application, formally called a Service Provider. This contract establishes trust between the application and the Identity Provider, which allows authentication to be delegated. **How** the Identity Provider authenticates end users is delegated as well. End users might be required to submit a username and password, MFA could be required, or they could be transparently authenticated (SSO). Authentication options are configured in the IdP.

Single Sign-On – Transparent Application Grouping
SSO is often coupled with a Federated Identity implementation. As a result, the difference is not always clearce.

SSO is the configuration of trust between applications who share an Identity Provider. SSO grouping has a powerful shared authentication effect, where the Identity provider can transparently authenticate end users to all applications in a group when they've authenticated one end user in that group.

Protocols Used in Identity Federation – SAML

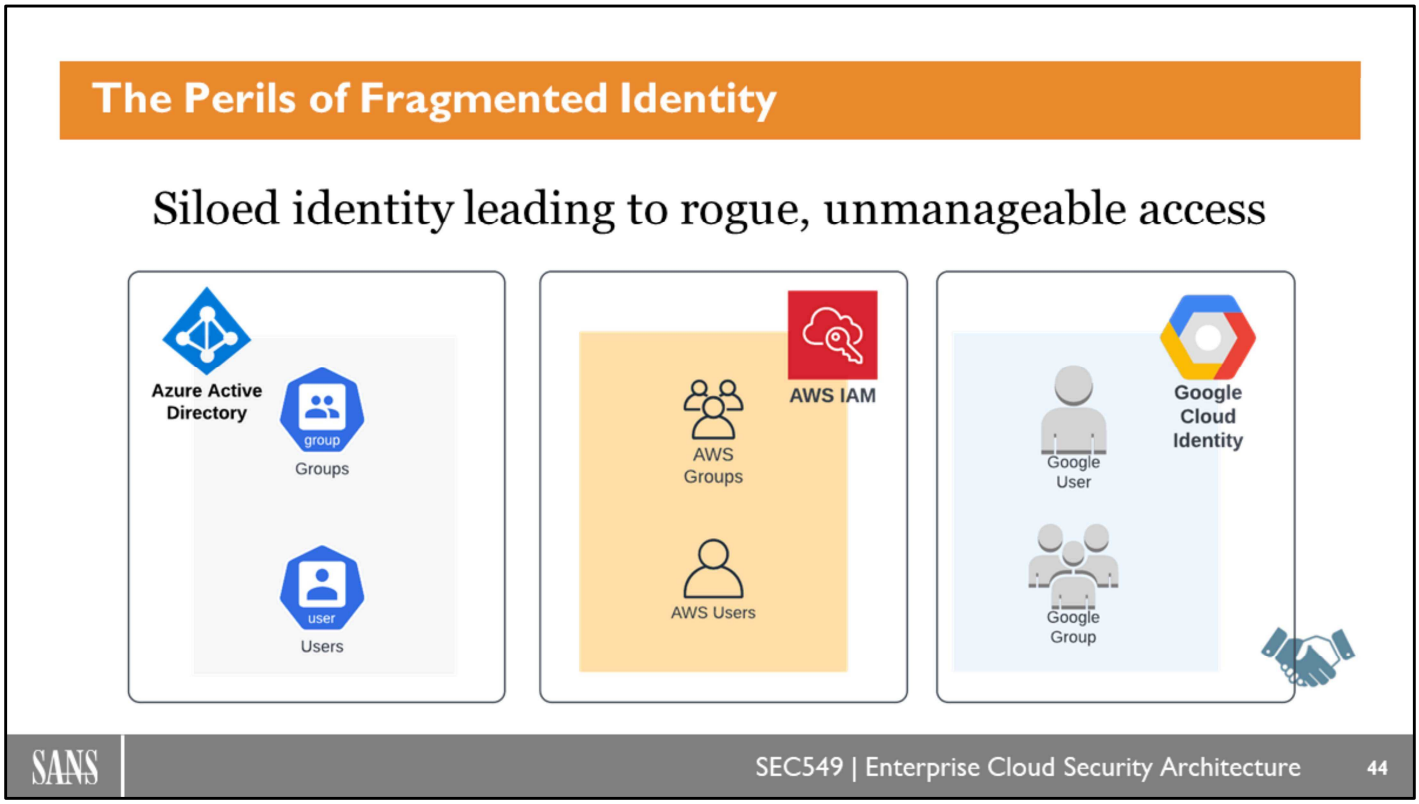


What Is SAML?
Security Assertion Markup Language (SAML) is a protocol designed to create security assertions. It provides a framework for exchanging identity information between providers. It is one of the primary protocols used to enable identity federation and authentication on the internet.

Benefits of using SAML include:
- Open standard, designed to work with any transport protocol (HTTP, SMTP or FTP)
- Provides interoperability between identity providers and service providers without these two parties needing to exchange secrets
- Allows applications (designated as service providers) to delegate authentication to a centralized identity provider

SAML Token
Signed XML document consisting of the assertion of the authenticated state of the end user along with any additional attributes of the end user. Optionally, the SAML Token may be encrypted; however, this does not add to the security posture of the document.

Establishing Trust
During configuration, the Service Provider is provided with the public certificate of the Identity Provider, which is later used to validate the signed SAML token. It is through the establishment of trust between the Identity Provider and the Service Provider that the SAML token can provide security guarantees.



The Perils of Fragmented Identity

Implementing a full IAM stack in every environment requiring authentication and authorization systems leads to siloed identity repositories. There are many problems that can arise when doing this:

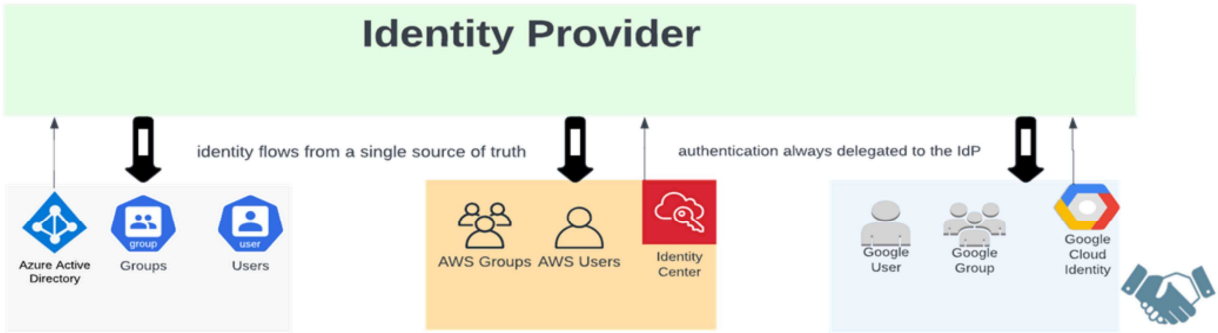
- End Users who need access to multiple target systems need to register separately in each environment, multiplying the amount of administration work needed to handle registrations and increasing the administration burden and costs.
- End Users with multiple registrations may be granted inconsistent privilege levels in the different environments. When systems remain siloed, there is no way to check for these types of inconsistencies.
- End Users who have access needing maintenance, such as during a change in job role, must have their access level updated across multiple environments, introducing the risk of error and further inconsistency.
- End Users who are terminated need their registration deleted across multiple environments, creating the risk that at least one will be missed.

To avoid each of these problems and more, a centralized approach to identity and access management is needed.

Benefits of Centralizing Identity

Account Hygiene is made possible with a central Identity Provider (IdP)

- Central provisioning and deprovisioning
- Real-time view into access-levels for all users
- Single point for account revocation and session time-out configuration



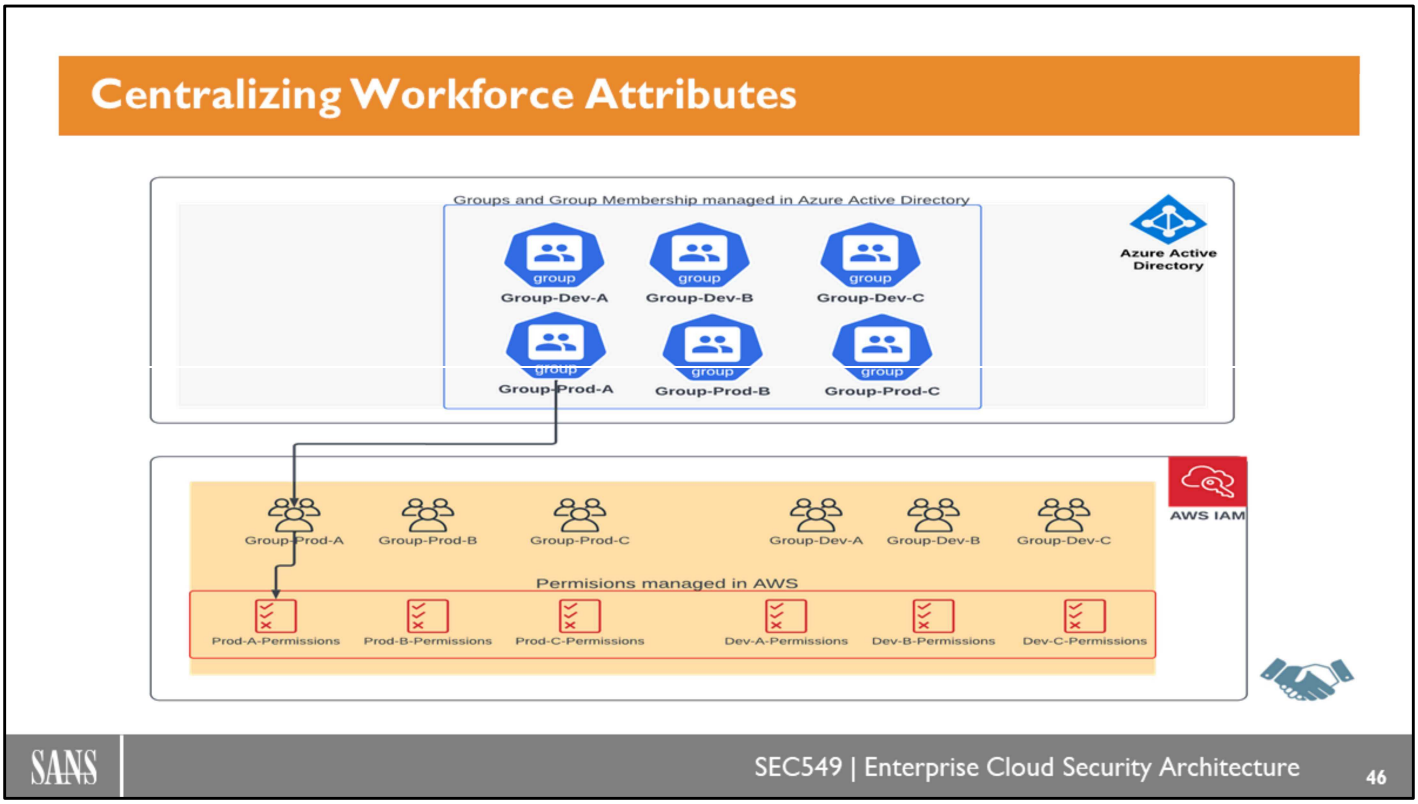
SANS | SEC549 | Enterprise Cloud Security Architecture 45

Benefits of Centralizing Identity

Centralized identity management means IAM all happens in one environment. In a workplace setting, this looks like the user signing into a single workspace to access all the applications and tools they need. Deprovisioning a user that has been centrally provisioned requires an action requiring only a single, central directory.

Decentralized identity management means IAM is spread out across multiple environments. In this instance, a user would sign into a single workspace, then continue on to sign into each individual application and tool separately. Deprovisioning this user after separation would require deactivating and removing them from multiple systems simultaneously.

With protocols like SAML, we can house users and groups in a centrally managed location and avoid the operational and security risks that identity silos create.



Centralizing Workforce Attributes and Roles
Not only should identities be housed in a central source, but so should the **roles and attributes** governing authorization policy. This is one of the key points in this course: the **centralized view of roles, groups, or attributes** translating to **access rights in the cloud**.

How can you ensure your entitlements are not maintained in siloed systems? This can be a challenge for both cloud service providers and integrated SaaS applications, particularly as the fine-grained permission management of these systems are internal constructs.

What you'll see is that these are ensured **by enforcing the same centralization principles we saw with the User to Groups and Attributes.**

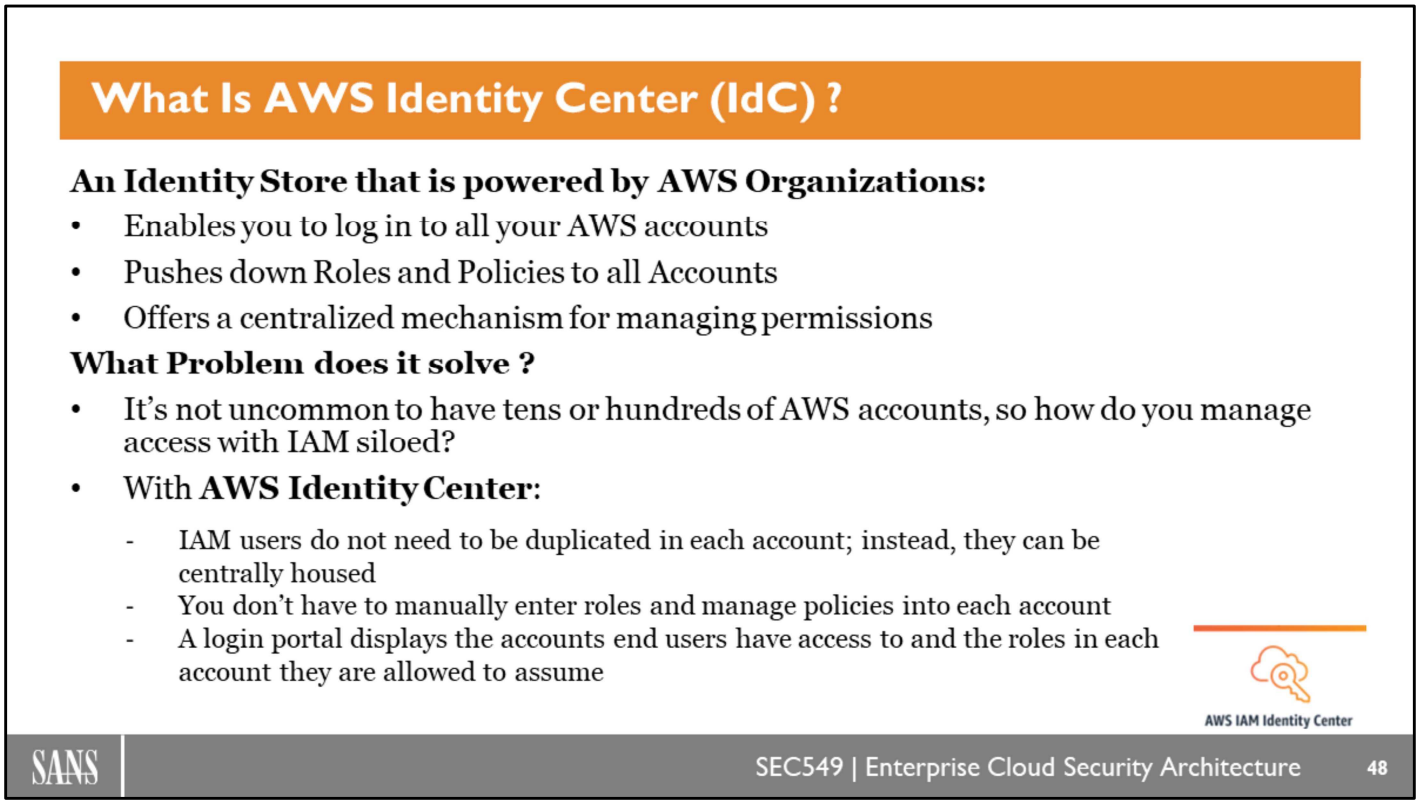
Depicted in the slide is an architectural pattern which depicts how you might centrally house user and group attributes in your Identity Provider and pass identifiers downstream. Make note of the features underpinning this pattern:

- Groups and group memberships are maintained in the Identity Provider; in this case, Azure Active Directory (AAD)
- Groups and memberships in AWS are not manually created; rather, they defer to their upstream directory for the source of truth
- Entitlements in AWS take the form of IAM permissions. Here, the assignment of groups to permissions lives as a policy assignment in AWS
- What groups are assigned to which collections of permissions must be a function of AWS and not an upstream identity provider

1. Clear Assignment Patterns
Encouraged is a one-to-one pattern where every group is assigned exactly one collection of permissions. This one-to-one mapping helps us overcome the challenges of siloed cloud access rights by making the provisioned permissions more auditable.

It will be clear if *Group-Prod-A* is assigned anything other than the collection of permissions called *'Prod-A-Permissions'*

2. Leverage Automation and Policy as Code
Policies and permissions used to grant access to your resources should be defined as code. This will allow you to take advantage of automation, track changes to permissions, and layer on governance. When making changes to permissions, delegate that task to a well-monitored, automated system. Ideally, individual teams or users should not have the ability to grant or revoke access rights. Neither should the collections of permissions be updated or modified manually.



What Is AWS Identity Center?

AWS Identity Center[1] (IdC) is intended as AWS's identity product for the enterprise workforce and was formerly called AWS Single Sign-On (AWS SSO). Why is it marketed as a solution specifically for workforce identities?

The capabilities of AWS specifically lend itself to solving problems a typical enterprise might encounter when their cloud presence grows and expands.

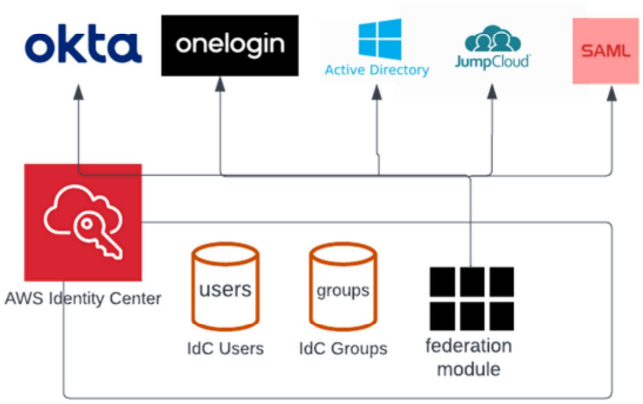
With Identity Center, you can break free of the access patterns which constrain end user identity into a single AWS Account. You can manage all attributes of a User or Group and assign permissions to each of them using Identity Center, across multiple accounts. Sticking with the theme of breaking free of the single account, by using AWS Identity Center, you can now gather collections of permissions that can now be managed outside of the context of the Account. Without leveraging Identity Center, a customer-managed policy might need to be recreated and therefore managed across dozens or hundreds of accounts. With Permission Sets in AWS Identity Center, the growing enterprise can be managed using these Permission Sets which are used to populate Roles into member accounts using a consistent version of the policy.

In this module, we will focus on the capabilities of AWS Identity Center and see how it might be used to centralize a workforce identity and provision access to AWS resources.

Reference:
[1]: https://sec549.com/id94

AWS Identity Center – Identity Storage Options

- Connect to your external identity provider or use the built-in, native user directory
- Your identity source defines where your users and groups are managed, including group membership, attributes, and entitlements
- Users managed by the Identity Center native directory (IdC Users) have unique usernames and passwords unless federated against an IdP



AWS IAM Identity Center

User Identity Storage Options in AWS Identity Center (IdC)

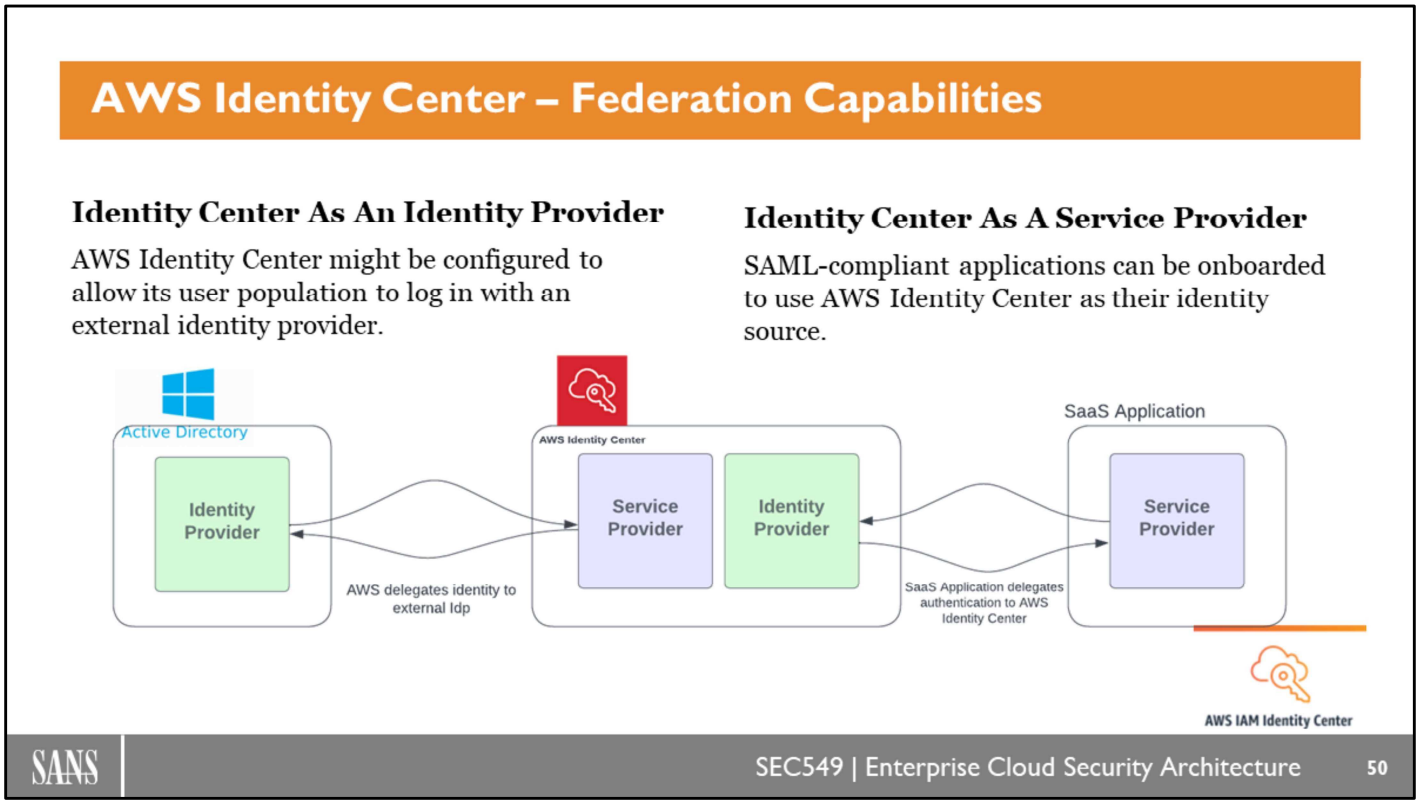
There are only a couple of prerequisites for using AWS Identity Center. First, due to its tight integration with AWS Organization, you cannot use Identity Center and its capabilities without also having enabled AWS Organization. Second, an Identity Source must be specified; this will define where Identity Center users and groups are managed and, in turn, where they will authenticate.

By default, the Identity Center built-in directory[1] is created and enabled from which you can manually create IdC Users and Groups, just as you would create a native IAM User or Group housed within an AWS Account.

Maintaining all AWS end user identities in Identity Center eliminates the problem of having to create unique users for every account; however, it still leaves AWS with its own unique bunker for identity, silo'd from other cloud identity systems or your on-premises active directory. End users who log into AWS leveraging the built-in directory do so using a unique username and password for their AWS registration.

An organization's goal when scaling operations in the Cloud should be to unify identity into a single source of truth. As such, configuring AWS Identity Center with an external identity provider should be any organizations' desired end state. Identity Center's native, built-in directory should only be used when bootstrapping the environment or for break-glass accounts.

Reference:
[1]: https://sec549.com/id190



Federation Capabilities of AWS Identity Center (IdC)
AWS Identity Center is extremely versatile and can be configured to serve as both an identity provider and service provider in the federation workflow. How would this work?

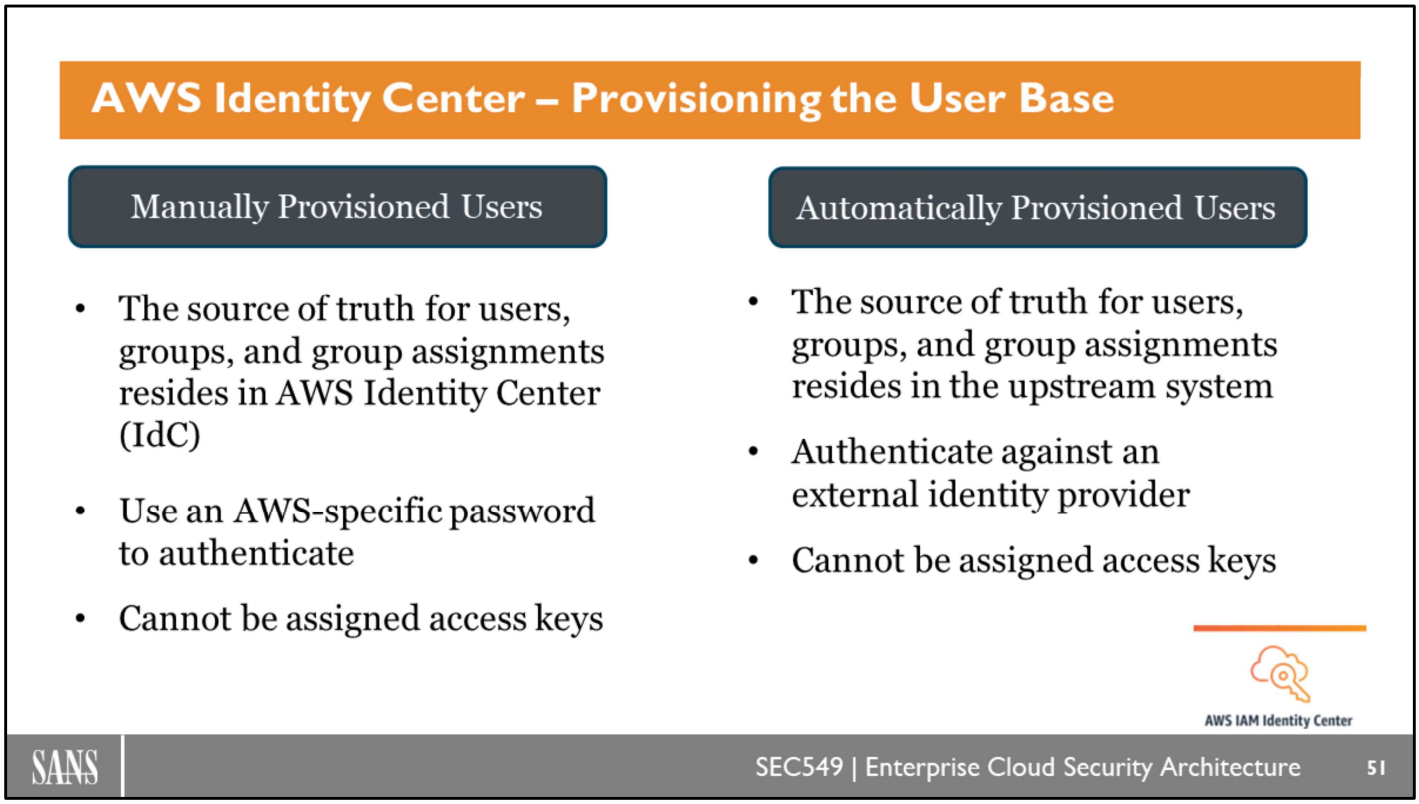
Workforce identities needing to log into your AWS environment can have their access federated against an external provider, such as Azure AD, Okta, or any other SAML-compliant IdP. In the scenario depicted here, AWS is acting as the Service Provider. It has been configured to delegate authentication to another entity and relies on the Identity Provider to assert the caller's identity.

Identity Center can also be configured to play the role of the Identity Provider. Applications can be configured to use AWS Identity Center as their external Identity Provider (IdP).

So, what happens if AWS Identity Center both federates authentication against an IdP AND serves as an IdP for applications as depicted in this diagram? While the service is completely capable of handling such a scenario, the end user experience might be less than desirable. Upon attempting to access the application, the end-user will find themselves first redirected to AWS Identity Center and then redirected to the core, external IdP as is seen in the depicted scenario. Ensuring that the authenticated user's context is accurately convened between the two federation points can be an operational headache.

Whenever possible, avoid chaining identity providers together and stick to a single IdP with hub-and-spoke patterns. Allowing AWS Identity Center to serve as an IdP may be appropriate when federating within the AWS ecosystem[1] such as to AWS Workspaces, but it rarely makes sense as an IdP for third-party applications.

Reference:
[1]: https://sec549.com/id95



Provisioning the Managed Identities User Base
AWS Identity Center allows you to provision a user and group population collectively called IdC Users and Groups. In these upcoming slides we'll cover how to provision IdC Users and Groups into AWS Identity Center and how differently provisioned user populations (manual and automatic) are incompatible.

IdC Users
Built into AWS Identity Center is a user directory, not unlike the native user directory that comes with every AWS Account. Whether users are housed in AWS Identity Center (IdC Users) or in an individual account (Native IAM Users), they are both configured with the following minimum attributes:

- Username: User uses this value to login.
- Password: The password a non-federated user uses to authenticate to AWS. One-time use, temporary passwords can be generated for users who are then forced to change them upon initial login. When you create a user with AWS Identity Center, AWS sends an email to the user by default so that they can set their own password.

The advantage of AWS IdC users over native IAM users is clear – AWS IdC users cannot be assigned Access Keys, eliminating an all-to-often source of initial compromise in the cloud.

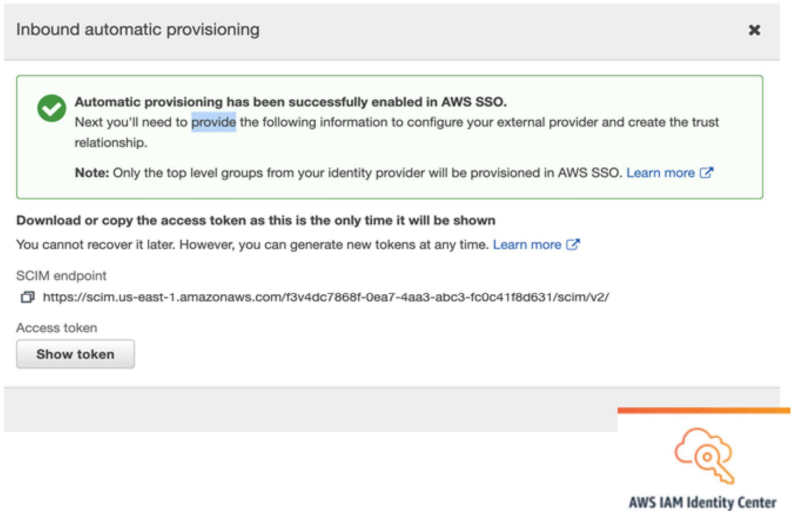
Manual Provisioning of IdC Users and Groups
Within AWS Identity Center, users and groups can be manually created. When they are, the source of truth for users, groups, and group assignments resides in AWS Identity Center. Manually provisioned AWS Identity Center users use a bespoken password for AWS for authentication and are not federated.

Automatic Provisioning of IdC Users and Groups
When users and groups are automatically provisioned, their source of truth resides in the upstream system. Automatically provisioned IdC users are also federated against an external identity provider, meaning they authenticate against another system. Automatic provisioned users ARE NOT assigned passwords.

Manually provisioned AWS Managed Identities cannot be active alongside automatically provisioned users. Once an external source for Managed Identities is configured, any previously created manual AWS Managed Identities users will be deactivated and will no longer be able to log in with their AWS-specific passwords.

SCIM for Automatic User and Group Provisioning

- SCIM: System for Cross-domain Identity Management (SCIM) v2.0
- Sync users and groups between a source and downstream directory through the Identity Center SCIM endpoint
- Trust established with a shared secret that must be manually rotated

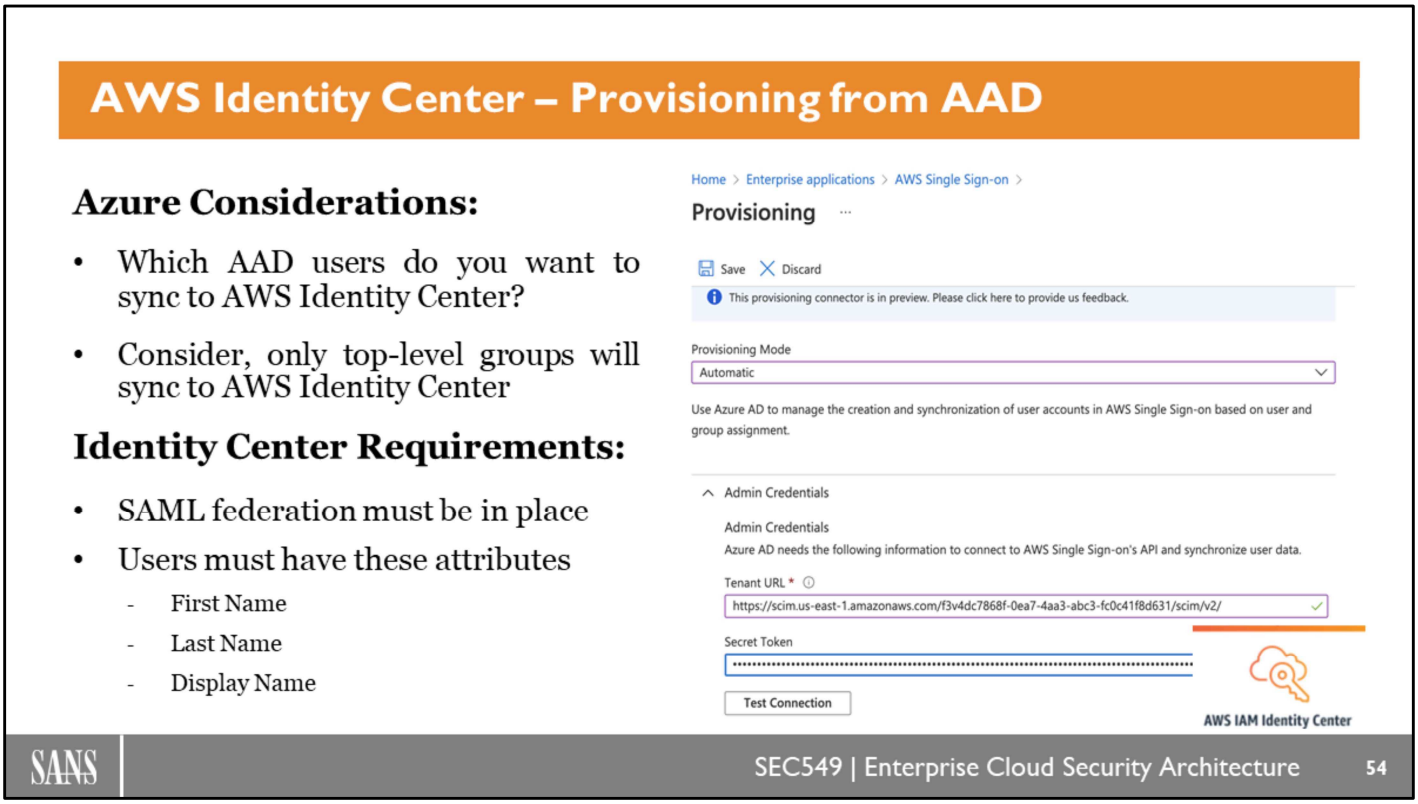


SCIM for Automatic User and Group Provisioning

Using the System for Cross-domain Identity Management (SCIM) protocol, you can sync users, their attributes, groups and group memberships to downstream directories from your primarily 'source of truth' directory. When you configure users to be provisioned via SCIM in lieu of manual creation, you create a mapping of your identity provider (IdP) user attributes to the named attributes in AWS Identity Center. This causes the expected attributes to synchronize between AWS Identity Center and your IdP.

To enabled this synchronization, trust is established between the source directory and the downstream directory through a configured shared secret. In AWS Identity Center[1], that secret is generated for you and consists of an Oauth bearer token. The bearer token is then shared with the upstream directory to authenticate requests to the AWS Identity Center published SCIM endpoint. The shared secret used in SCIM provisioning in AWS Identity Center is issued with **an expiry date of 1 year.** Set yourself a reminder to rotate this secret before it expires or provisioning via SCIM will silently break[2].

References:
[1]: https://sec549.com/id96
[2]: https://sec549.com/id97



Provisioning Users and Groups from Azure Active Directory
Mapping your user base from Azure Active Directory to AWS Identity Center is fairly straightforward, although there are a few design considerations and one or two gotchas that are important to review.

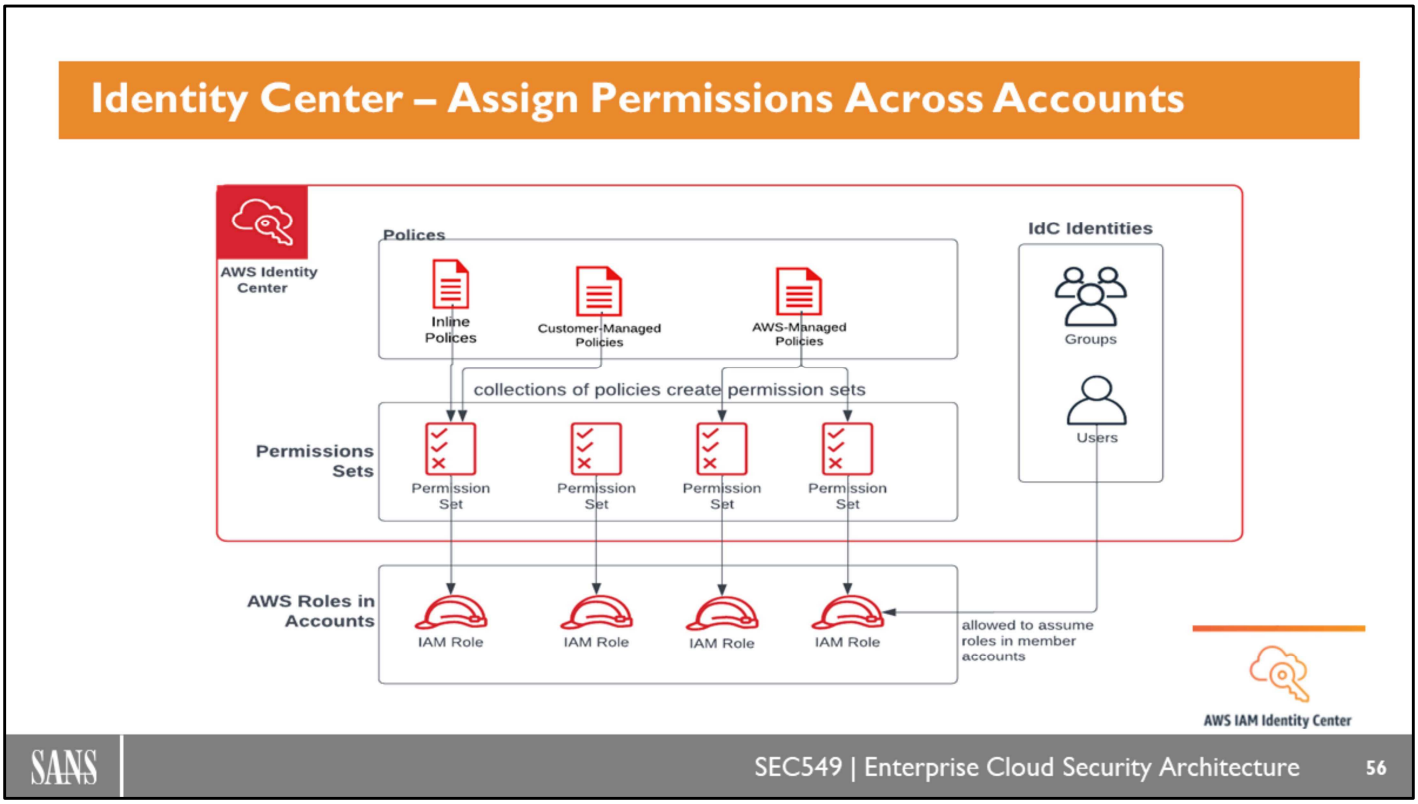
Considerations at your source directory – Azure Active Directory (AAD)
It's unlikely you will want to sync your entire AAD user population to AWS Identity Center[1]. Fortunately, you can be selective about who is provisioned into AWS. In Azure AD, provisioning can be scoped based on assignment to the enterprise application used for federation and provisioning, based on attributes of the user or by group membership.

Another consideration within AAD is group membership. Only your top level groups from Azure Active Directory are replicated to the AWS Identity Center directory. Nested groups and their memberships will not be available in AWS Identity Center.

Finally, consider your requirements for timely consistency between the two directories. Out-of-the-Box, provisioning a user base between Azure and AWS Identity is not Just-In-Time(JIT) and does suffer from a delay. The delay can be upwards of 40 minutes between a change in AAD and its populating in AWS Identity Center. There are, however, PowerShell scripts and third-party solutions which can force AAD to provision in a more timely basis if you need stronger consistency[2].

Requirements by your downstream directory – AWS Identity Center
When looking to enable automatic user and group provisioning into AWS Identity, only a handful of hard requirements are in place. First, SAML federation must be configured between AWS Identity Center and the external Identity Provider, which will allow your provisioned users to authenticate externally. Lastly, AWS Identity requires just three attributes be minimally configured for the user object for acceptance into the directory. Those are 'First Name', 'Last Name' and 'Display Name'[3].

References:
[1]: https://sec549.com/id98
[2]: https://sec549.com/id99
[3]: https://sec549.com/id100



Managing User Assignments Across all AWS Accounts
Now that users and groups have been automatically populated into AWS Identity Center, it's time to turn our sights to assigning them permissions to cloud resources.

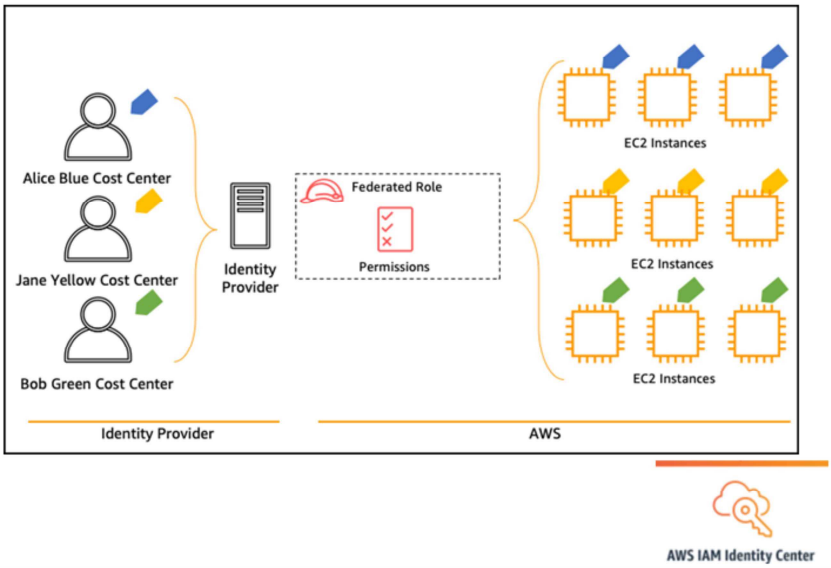
Managing User Permissions Across All AWS Accounts
Permission sets are a way to define permissions centrally in AWS Identity Center, so that they can be applied consistently across all of your AWS accounts. Permission sets are bundled collections of policies. Within AWS Identity Center, you can maintain groupings of policies and assign IdC users and groups these permission sets. When a group is assigned a permission set to a specific AWS Account, the permission set is provisioned to the targeted AWS account as an IAM role with an accompanied Identity-Based Policy. AWS Identity Center configures a trust policy on the role, allowing members of the group to assume the IAM Role into the target account.

Reference:
[1]: https://sec549.com/id101

AWS | Identity Center – Leveraging ABAC

RBAC and ABAC are complementary in AWS

- **RBAC:** IdC Users and Groups housed within AWS Identity Center are granted permission sets based on group membership
- **ABAC:** IdC Users and Groups are conditionally allowed to access certain resources based on their user attributes



In the previous slide, we examined granting access to resources using a Role-based Access Control (RBAC) paradigm. A complementary system for access control is supported in AWS – Attribute-Based Access Control (ABAC)[1].

When you use ABAC, you define permissions based on matching attributes, rather than exclusively group membership. Attributes are then used as conditional arguments in policies.

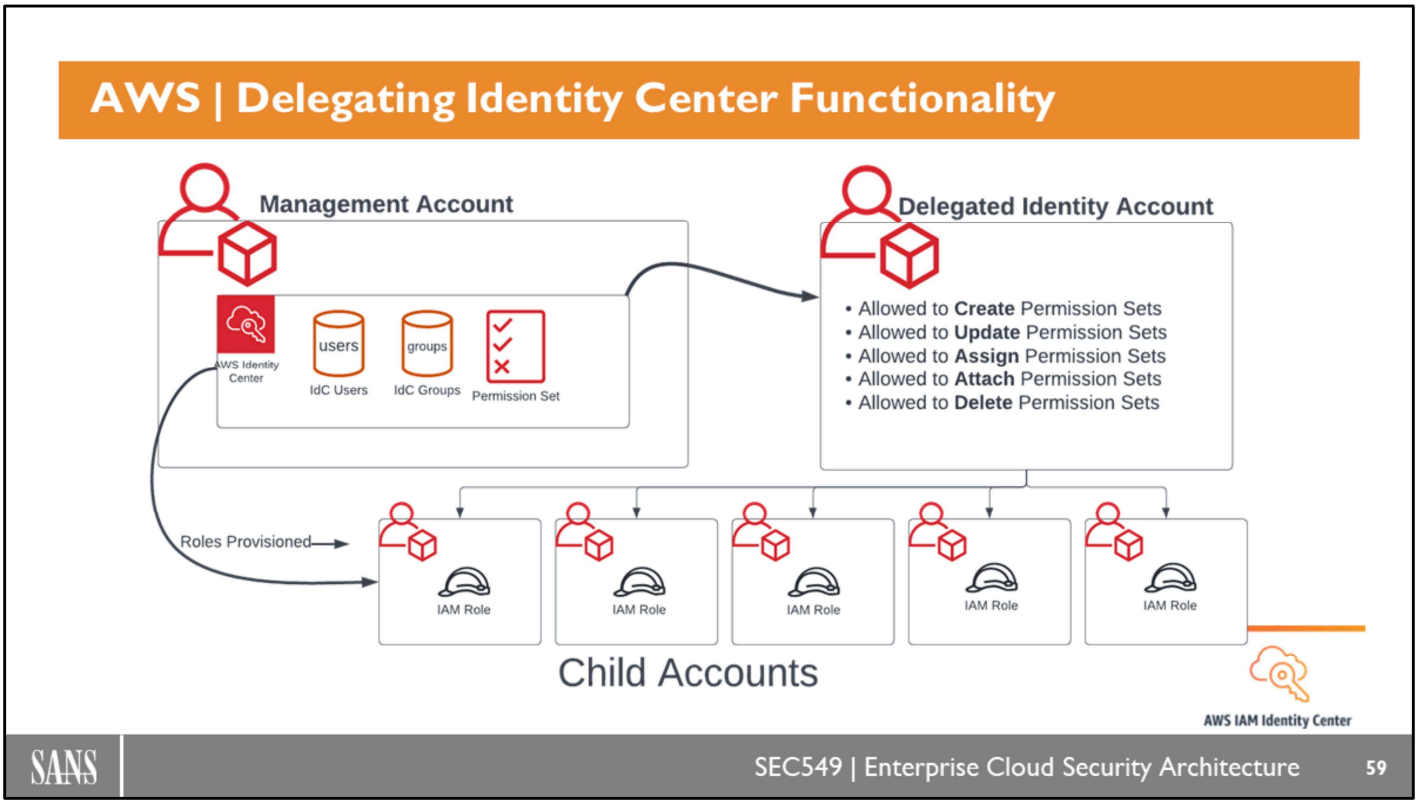
- *Example:*
  - IF a user attribute == manager THEN they can access the resources having been tagged as == confidential
  - IF a user attribute == Dept123 THEN they can access the resources having been tagged as == Dept123

Passing Attributes to AWS Identity Center (IdC)
Attributes of a user can be leveraged by AWS ABAC by either sync-ing the user attributes as a part of your SCIM provisioning process or they can be passed to AWS Identity Center during authentication as a component of the SAML identity assertion. In either case, these attributes need to be flagged in AWS Identity Center as the ones to be used in access control decisions. Once configured, these attributes are passed as session tags with the end users authenticated session and can be used to conditionally allow or deny access to particular resources or entire roles[2].

Using an ABAC strategy, defining access based on matching attributes to resource tags instead of purely group membership, helps to reduce the number of roles you must create and manage in your AWS environment.

Consideration when using AAD as the source of attributes
If an attribute is removed from a user in Azure AD, that attribute will not be removed from the corresponding IdC user in AWS Identity Center . The work-around? Instead of removing attributes from users, they should be changed to a different (non-empty) value. That change will be synchronized to AWS Identity Center appropriately.

References:
[1]: https://sec549.com/id102
[2]: https://sec549.com/id103
Image source: New for Identity Federation – Use Employee Attributes for Access Control in AWS | AWS News Blog https://sec549.com/id104



Identity Center Delegation
As the cloud footprint grows, it becomes increasingly important to integrate the management of the cloud identity lifecycle into your organizations broader Identity and Access Management (IAM) capability. Instead of granting members of the IAM team access to your all-powerful AWS Management Account, delegate portions Identity Center to a child AWS Account.

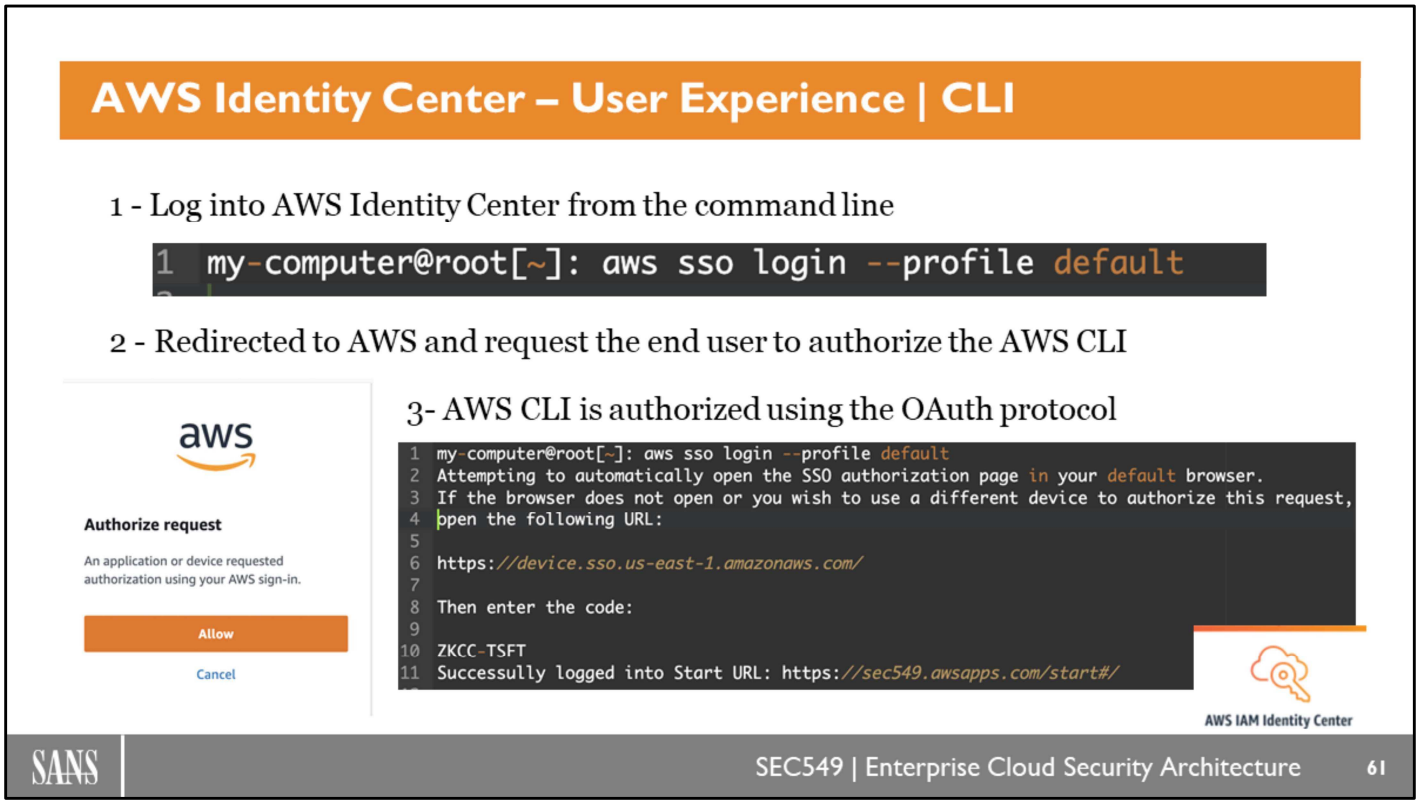
Delegate to a child account
The functionality you can delegate from Identity Center in the Management Account includes the maintaining of Permission Sets, creation and provisioning of Permission Sets and granting permissions to end users by attaching Permission Sets to Accounts. Users, Groups and SSO configurations will still reside in the Management Account, however assigning permissions will additionally be possible in the delegated account. Delegated functionality can be allowed on all child accounts or only a subset of child accounts depending on your desired delegation model.

Do Not Delegate
As a security best practice, access to the AWS Management Account should be limited, therefore its important when delegating the identity capabilities of AWS Identity Center to withhold access to the Management Account. When crafting a delegation policy, ensure the Identity Delegation Account and Management Account are excluded, only allowing permission management on child accounts. This can be accomplished by leveraging a Resource Tag applied to the Management Account[1] and Identity Delegation Account by referencing it with the *ResourceTag*[2] global condition key.

References:
[1]: https://sec549.com/id201
[2]: https://sec549.com/id202

End User Experience on the Console
Having accessed the student lab infrastructure via AWS Identity Center, you have gotten a glimpse at what an end users experience is when authenticating with AWS Identity Center .

After authenticating with the external identity provider, the end user is navigated to the Identity Center home page. Here, every account that they have been provisioned access to is presented. All AWS Accounts, and the name of every permission set is listed for the end user, allowing them to chose which account to drop into and which role to use when in that Account.



AWS Identity Center User Experience using the Command Line
AWS Identity Center is compatible both with accessing resources through the console and programmatically via the AWS command-line interface (CLI).

1. In this scenario, the end user requests to login to AWS on the command-line, indicating which AWS account they wish to access. In the AWS CLI, the account id and role an end user requests is called a 'profile'.

2. A browser is launched where the end user authenticates and is asked to authorize the AWS command-line interface (CLI). Authorizing the request generates an authorization code for use in the OAuth2 Authorization Code flow.

3. Finally, the authorization code is populated into the terminal. Using the one-time use authorization code, the AWS command-line interface (CLI) requests and stores temporary session credentials which are subsequently used when authenticating to AWS APIs.

Reference:
[1]: https://sec549.com/id105

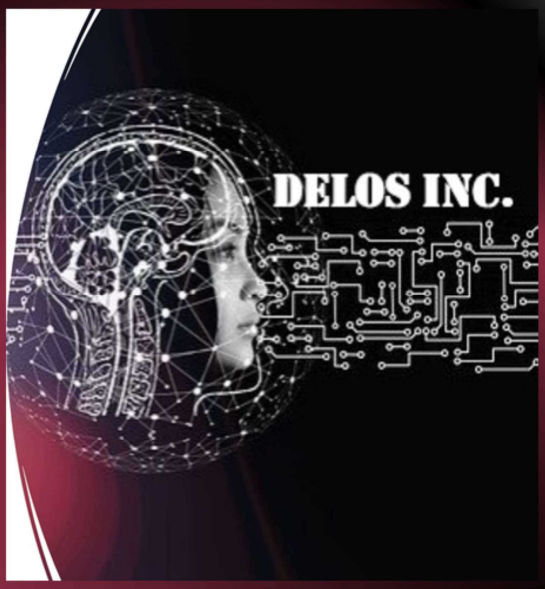
Cloud Journey Phase: Centralizing Identity Provisioning

Scenario:

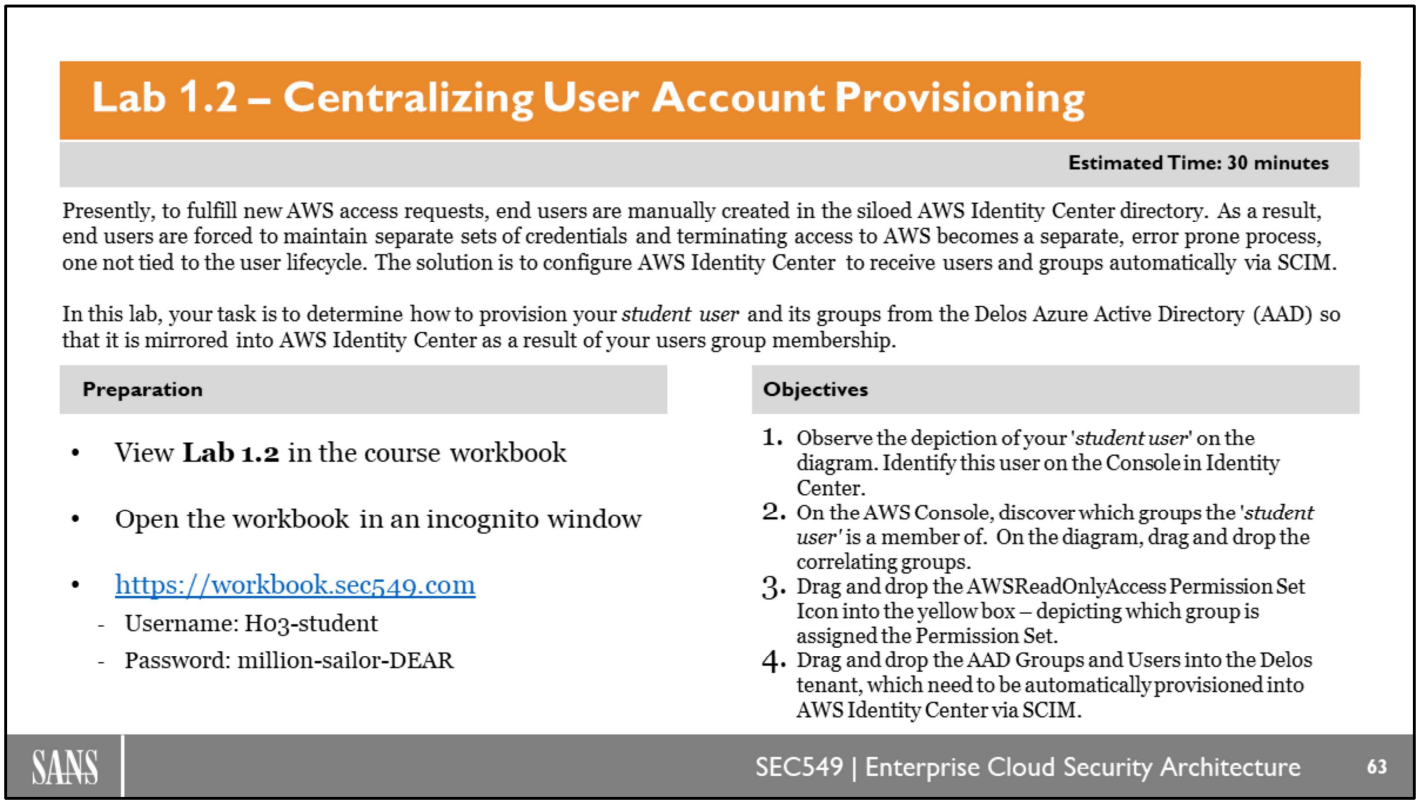
Planning and preparation for Delos Digital Transformation is in full swing. They've bought you on as a consultant to help them navigate their transformation into a cloud-first organization.

To enable third-party access into their AWS estate, they've been creating individual users in the AWS Identity Center directory. As one of the most pressing tasks, you've advised them to ensure all cloud access originates from a central identity provisioning system.

Help Delos plan for centralized account provisioning from Azure Active Directory, where the scope of users and groups is based on group membership.



Reference:
Image Source: Pixaby user geralt: https://sec549.com/id51



Lab 1.2 Instructions:

1. Observe the the depiction of your '*student user*' on the diagram. Identify this user in the AWS Identity Center console page.
2. On the AWS Identity Center console page, discover which groups the '*student user'* is a member of. On the diagram, drag and drop the correlating groups.
3. Drag and drop the AWSReadOnlyAccess Permission Set Icon into the yellow box which will depict the group being assigned the Permission Set.
4. Drag and drop the AAD Groups and Users into the Delos tenant, which need to be automatically provisioned into AWS Identity Center via SCIM.

Lab 1.2 – Summary

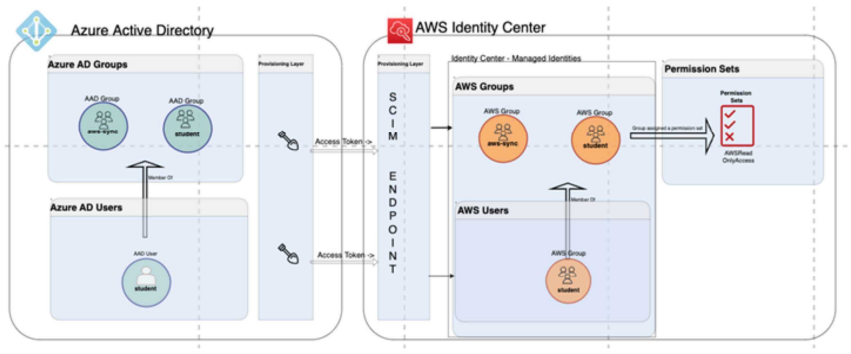
In this lab, you...

- Logged into the AWS Console to discover the group membership of the *student user*
- Depicted the creation of the *student user* and its group memberships into AWS Identity Center via Azure Active Directory (AAD) automatic SCIM provisioning
- Represented the Permission Sets in AWS Identity Center and allocated Permission Set to the *student* group
- Logged into the AWS Console to answer the ⚡ See it in Action ⚡ questions

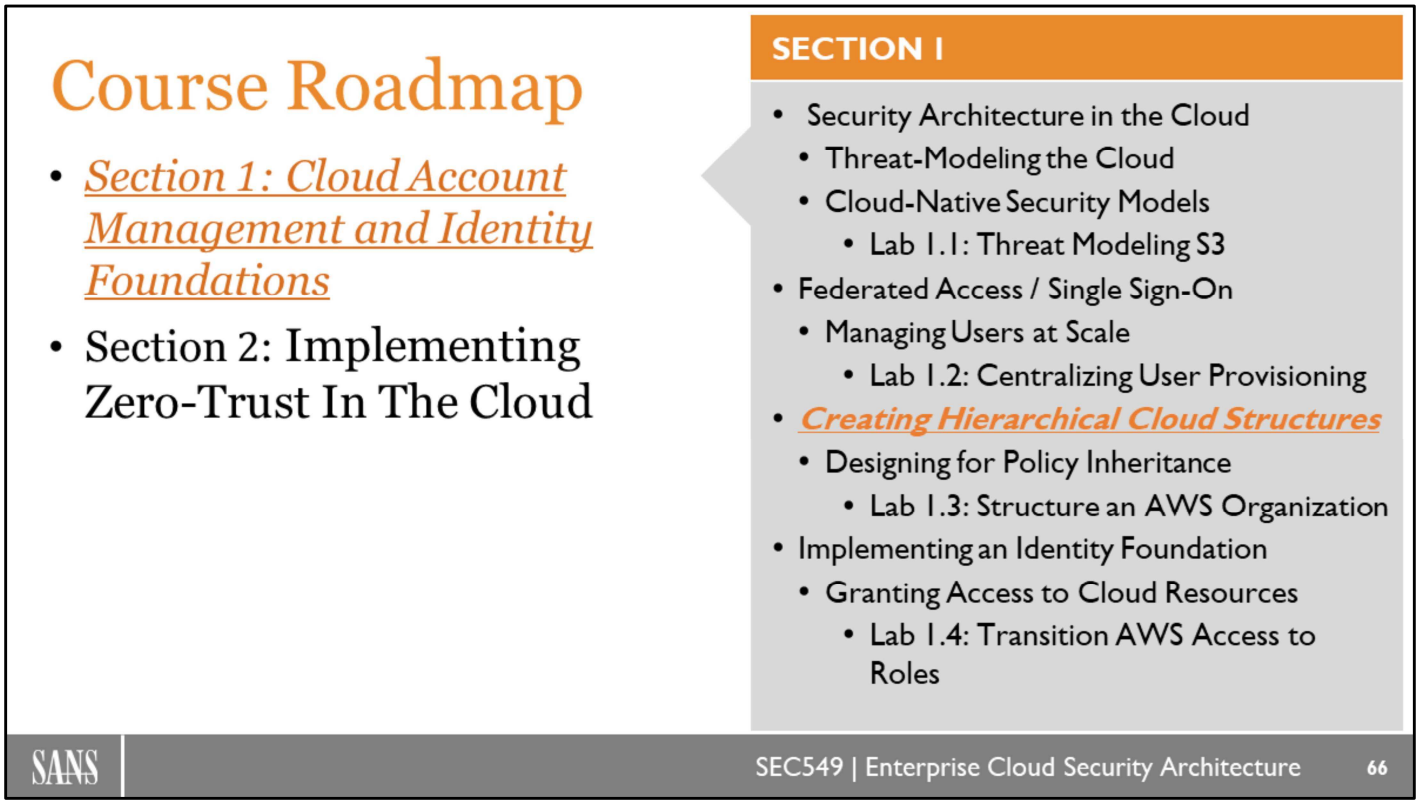
This page intentionally left blank.

Lab 1.2 – End State Architecture

- The Delos Identity Admin changed the identity source in AWS Identity Center to be 'External', enabling automatic provisioning via SCIM
- This configuration change allowed for automatic user provisioning to be utilized to provision the '*student user*' and its group memberships
- Users, groups, and group memberships which originate in Azure Active Directory are sync'd over into AWS Identity Center where they become Managed Identities
- When users are added to the '*aws-sync*' group, they are replicated into Identity Center within 40 minutes



This page intentionally left blank.



In this module, students will learn about the different resource containers available to them in AWS, Azure, and GCP and the best practices for leveraging them to segregate workloads. They will learn how to take advantage of each cloud's unique inheritance model to apply policies across swaths of resources. Using native policy enforcement, students will learn what tools they have available to enforce consistency across resources in an organization.

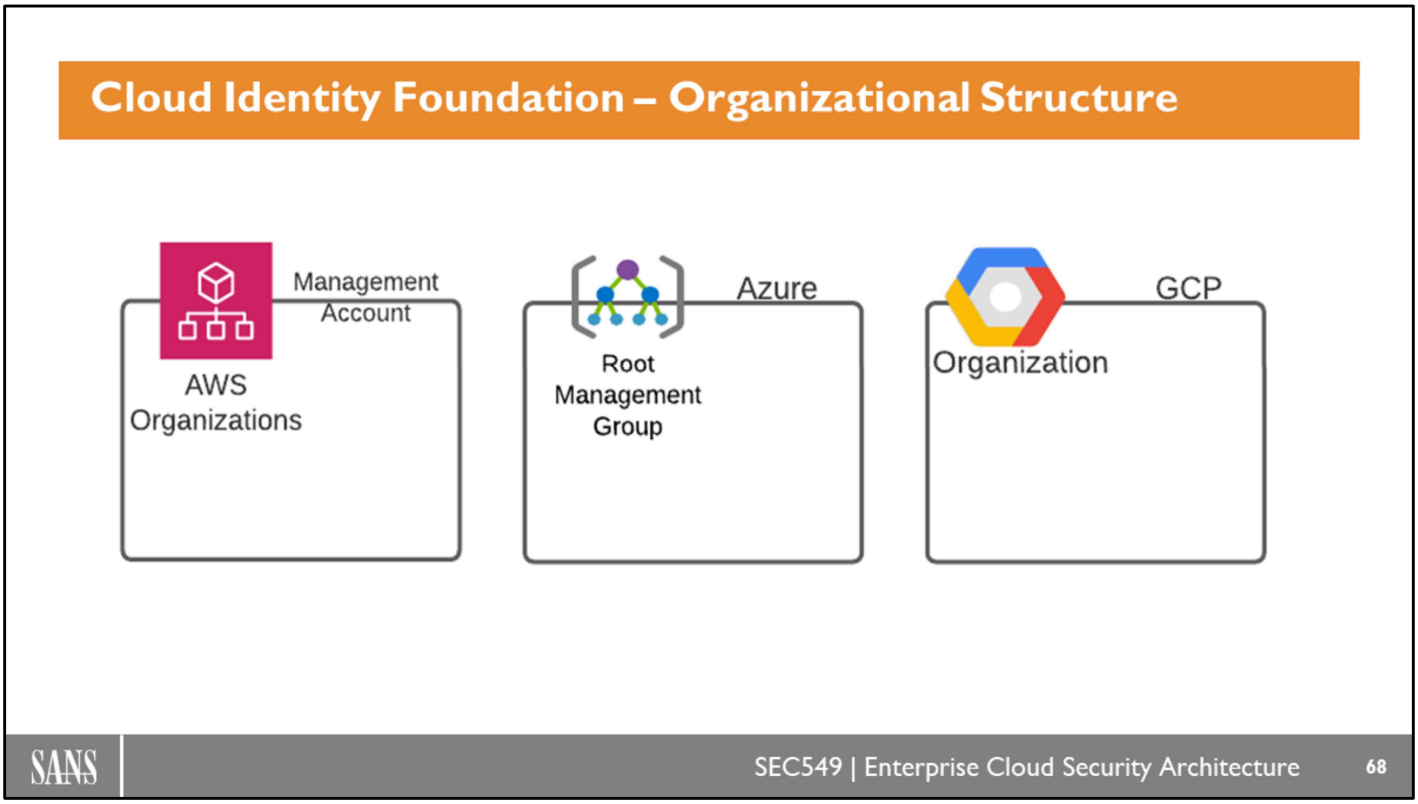
Takeaways

You will learn about the different resource containers available to you in AWS, Azure, and GCP and the best practices for leveraging them to segregate workloads.

You will learn how to take advantage of each cloud's unique inheritance model to apply policies across swaths of resources.

Using native policy enforcement, you'll discover what tools the Cloud Service Providers have available to help you enforce consistency across resources in an organization.

This page intentionally left blank.



Building an Identity Plane using the Major Cloud Providers
While called by different names, the highest-level resources from which the identity plane is built are similar in the Big Three CSPs. Breaking them down, you have these:

- ***The AWS Management Account***– despite being the oldest of the three, AWS only added this concept later as an option for customers to help manage accounts. The AWS Management Account is just like any other AWS Account except that it is the Account from which other Accounts can be members. However, the Management Account is not the top node for applying policy. In AWS, the top node which you can apply policy is called simply 'The Root' which is a construct that encompasses all member accounts.

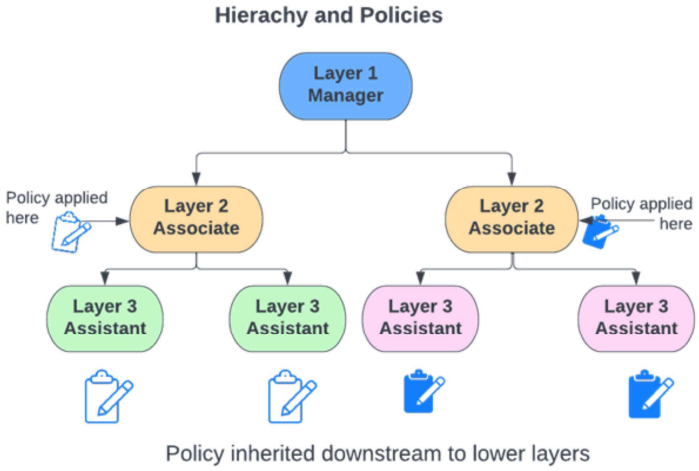
- ***Azure Root Management Group*** – this is a specifically designated Management Group which itself can contain child Management Groups. Associated to every Root Management Group is exactly one Azure Active Directory (AAD) Tenant. An Azure Tenant is an unique instance of Azure AD. A new Azure Tenant is transparency created in the background whenever a customer signs up for a new Office 365, Dynamics 365 or Azure Subscription

- ***GCP Organization*** – this top-tier node marks the initial container into which you can attach Roles. No Resources are housed in your GCP Organization; rather, from the Organization, you create smaller containers like Folders and Projects.

All IAM planning in the cloud begins with these the top-level resources. This is important because:
- The top-tier resources allow for centralizing the authority for the rest of the cloud, allowing for the creation of lower-level accounts, projects, or subscriptions.
- In the case of Azure and GCP, permissions applied at the top-tier resource convey permission in all child/subordinate resources, as they are inherited down the hierarchy.
- The AWS Management Account has unique *'super-admin'* capabilities including the ability to apply guardrail policies which can be enforced Organization-wide.

Hierarchical Cloud Account Structures

1. Direct resource ownership
   - parent resources 'own' child resources
2. Both One-To-One and One-to-Many relationships can occur
3. Resources in a cloud account structure are typically policy attachment points
4. Policy inheritance flows down the hierarchy, not up



Hierarchical Cloud Account Structure

The concept of a resource hierarchy refers to the way resources are organized inside a cloud platform account, e.g., is the structure flat or is it possible to build hierarchies. A core concept of a hierarchy is inheritance. This is where you'll see that child resources down the line can inherit the same upstream policies, whether those policies grant permissions or restrict actions.

The essential features of hierarchical cloud account structures are:

- Direct ownership of resources between the parent level to the child level (a one-way ownership)
- The possibility of one-to-one relationships or one-to-many relationships within the different levels
- Inheritance of policy down the hierarchy but never up the hierarchy
- Resources in the Cloud Account Structure become policy attachment points

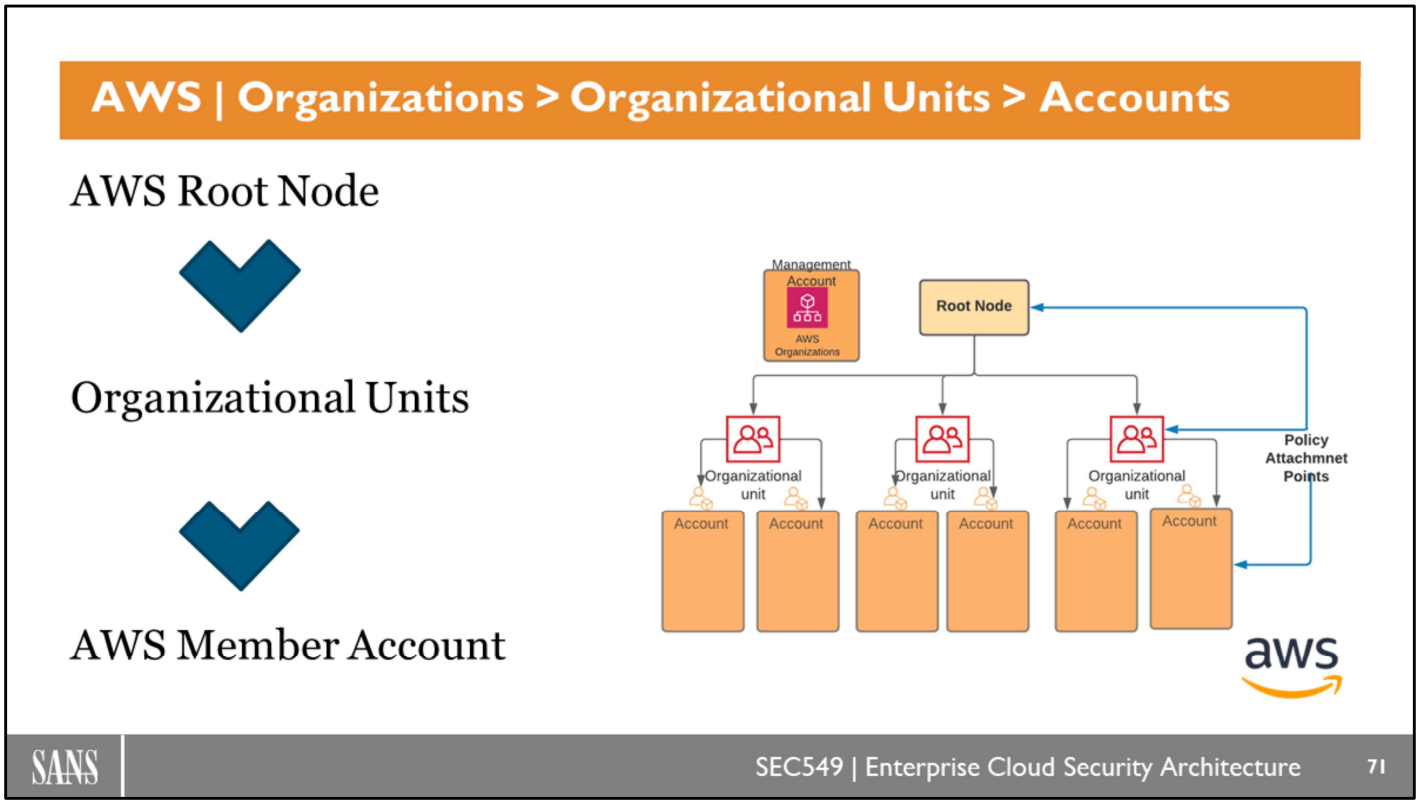
Expect differences between the Big-Three Cloud Providers:

- ***AWS*** – Organizational Units (OUs) can be created to organize accounts. You can attach a restrictive Service Control Policy to an OU so that all accounts within the OU have that inherited policy. Multiple OUs are allowed within the Organization and OUs can be nested within one another. Unlike Azure and GCP, policies **granting permissions** to Users cannot be applied to higher-level resources, such as at the Management Account, OU or Root Node. Instead, the Management Account is used to centrally organize and push down permissions that grant policies to member accounts.

In contrast to AWS which has a weak hierarchy, Azure and GCP have built their clouds to have a strong hierarchy.

- ***Azure*** – organizes itself into Management Groups, Subscriptions, and Resource Groups. Both restrictive policy (Azure Policy assignments) and policy granting permissions (role definitions) can be applied to higher-level nodes in Azure and will take advantage of the concept of inheritance. Deletion of higher-level resources deletes child resources.

- ***GCP –*** divides itself into the Organization, Folders, Projects, and resources within the Projects. Both restrictive policy (Org Policy Constraints) and policy granting permissions (Role Bindings) can be applied to higher-level nodes in GCP and have the concept of inheritance be in effect. Deletion of higher-level resources deletes child resources.



AWS Organizations → AWS Organizational Units → Member Accounts
An AWS Organization is a service that can be enabled in AWS to consolidate a collection of AWS accounts so they can be managed as a single unit. The point of an AWS Organization is to enforce consistency in policies and rules in ways that help manage the Organizational Units (OUs) and Accounts beneath it.

The AWS Account from which you initially enable Organizations is called several things, depending on what era of AWS documentation you refer to. Sometimes this Account is called the Payer Account or Root Account. More recently, it is referred to as the 'Organizational Account' or the 'Management Account'. In this course, you will see this Account referred to as the Management Account for consistency and to keep inline with the most recent AWS nomenclature. The Management Account is like any other AWS Account in that it **can** house resources like PaaS and IaaS Services. It should be noted however that this is against best practices. The Management Account should never contain infrastructure.

In all AWS Organizations, you will have a single Management Account. Everything else (member accounts, OUs, and policies) are additional/optional aspects of the Organization. The Management Account is at the 'helm' and is where the rest of the cloud estate is managed from. Even so, you would need cross-account AWS IAM roles to enable the Management Account to have access to each member account in the Organization. We will touch on this in the next slide.

What are the features of the Management Account?
- It allows for policy-based management of multiple AWS accounts at once
- It helps automate the creation and/or management of new AWS accounts
- It facilitates a consolidated billing process
- It is a free service as part of AWS

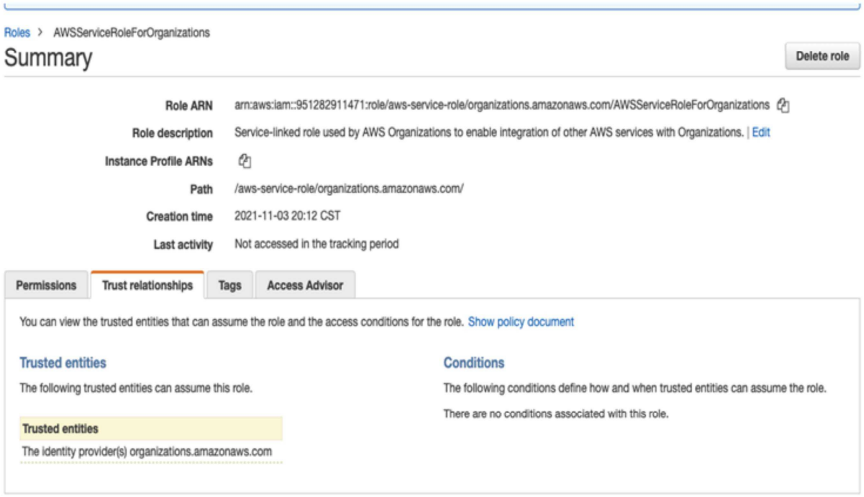
The AWS Organizations Service helps with:
- Centrally provisioning accounts and resources
- Creating shared resources between Accounts

- Leveraging integrated services to audit the environment for compliance
- Controlling access to accounts, regions, and services
- Simplifying billing and optimize costs
- Centralizing management of backups and tagging in multi-account environments

Organizational Units (OUs) can be thought of as a folder structure. They are created and managed from the Management Account. OUs themselves are strictly for grouping accounts with similar operational needs and security policies. You can organize OUs into a manageable hierarchy reflecting a company's structure or design OUs in an effort to group Accounts which are likely to have similar policy constraints.

It's best practice to use OUs to help organize member accounts according to common policies, as we will cover in an upcoming slide on Service Control Policies.

AWS | Access Accounts from the Management Account



- *OrganizationAccountAccessRole* IAM Role is created in all Member Accounts when they are created by the Organization
- This role gives the Organizational Account full access into the Member Accounts

Accessing AWS Member Accounts From the Organizational Account

The Management Account wields a lot of power over the downstream accounts in the Organization. How is this accomplished? AWS Organizations leverages the following access to manage downstream accounts:

- When an Organization is created, an IAM Role is automatically created by the AWS Organizations Service in each Member Account called '*OrganizationAccountAccessRole*'. With this access, the Organizations Service has full access to child accounts, as the *AdministrativeAccess* Managed Policy is attached to the Role.

- If you invite an existing AWS Account to join your organization, that invited Account needs to accept the invitation to join the AWS Organization. Access from the Management Account into an invited Account is not automatically configured. In this case, an *'OrganizationAccountAccessRole*' IAM Role needs to be manually created allowing the Organizational Account to access the invited member.

GCP | Organization > Folders > Projects

GCP Organization

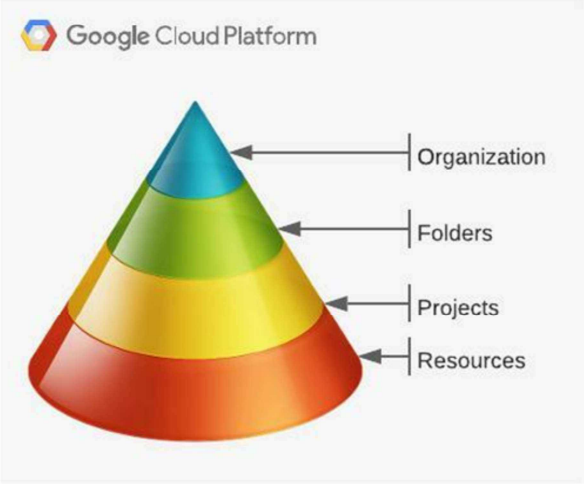
- Top-tier node
- Allows for centralized management of resources

Folders

- Optional
- A good way of sorting Projects

Projects

- Analogous to an AWS Account
- Assigned a unique project ID



GCP Organization → Folders → Projects
Resources in GCP are organized hierarchically where each Resource has exactly one parent. It is important to keep in mind that the term 'Resource' in GCP both refers to the Nodes of the organizational structure and the collection of assets in a Project, such as a Compute Instance or Cloud Bucket.[1]

The top-level Resource in GCP is the Organization[2,3]. Below the Organization, Folders can be created as generic containers. Folders are a mechanism to organize Projects and create IAM policy attachment points in order to refine access control[4]. An Organization can have zero Folders up to 300 and nested up to 10 deep.

Projects in GCP can be thought of as analogous to the Account in AWS. A Project is the container which houses all of Resources your team may utilize from GCP[4].

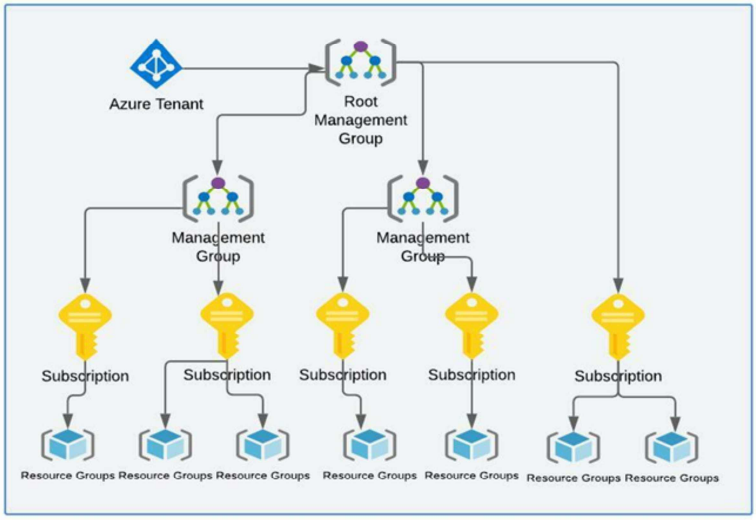
At the bottom of the hierarchy, a Project is made of a collection of Resources, fundamental components that make up all Google Cloud services. Examples of Resources include Compute Engine Virtual Machines (VMs), Pub/Sub topics, Cloud Storage buckets, App Engine instances. All of these lower-level Resources can only be owned by projects[6].

Inheritance
IAM Policy can be attached or bound to an Organization, Folder, Project, or to a specific Resource. Each of the Nodes in the Resource Hierarchy is a policy attachment point. For example, a User could have the Role Bigquery.Tables.Viewer applied at the Project Level. This would allow the User the ability to view all BigQuery Tables in the Project, present and future. Conversely, if the Role was instead bound at the Resource level, to a specific BigQuery Table, the User would only be able to view the content of that specific Table[7]. GCP, has over a 100 types of Resources to which an IAM Policy can be attached[8]. IAM Policy should be attached at the lowest resource level possible to limit the scope of permissions[9].

References:
[1] https://sec549.com/id19
[2] https://sec549.com/id20

[3] https://sec549.com/id21
[4] https://sec549.com/id22
[5] https://sec549.com/id23
[6] https://sec549.com/id24
[7] https://sec549.com/id25
[8] https://sec549.com/id26
[9] https://sec549.com/id27
Image Source: http://www.jumsoft.com/toolbox-for-keynote/

Azure | Management Groups and Subscriptions



SANS SEC549 | Enterprise Cloud Security Architecture 76

Azure Tenant → Management Groups → Subscriptions → Resource Groups
The Azure hierarchy can be even denser than GCP in that it introduces an additional node type that can be used to organize resources.

Nodes within the Azure hierarchy include the root management group, one or more child management groups, one or more subscriptions, resource groups, and the resources within them. At least one subscription must be enabled to work within Azure (along with the root management group). Subscriptions are organized into containers called management groups[1]. Management Groups, Subscriptions, and Resource Groups are all policy attachment places and IAM isolation boundaries, allowing them to be governed as a single entities.

Subscriptions[2] in Azure is the most ubiquitous resource container, analogous to AWS Accounts and GCP Projects. All new resources must be linked to a resource group which, in turn, is a member of a Subscription.

If a Subscription is moved to a different management group, the policies that it previous inherited are no longer in effect; it is now governed by any new policies applied to the new Management Group. While policies are inherited in a top-down manner, you can apply them at any level. In other words, you can apply them to a Subscription only, knowing that all resources within it are also affected by the same policy. While Azure Policy assignments used for creating guardrails in Azure cannot be overridden at a lower level, policy definitions can be created with exclusions.

References:
[1]: https://sec549.com/id28
[2]: https://sec549.com/id31
Image Source: Organize your Azure resources effectively: https://sec549.com/id29

Two Motivations Driving Your Hierarchy Design

1. **IAM Inheritance –** Consider what points in the hierarchy IAM policy will be attached giving particular attention to how permissions will be inherited by downstream resources.

2. **Clear Resource Ownership –** Consider if a cloud hierarchy structure can be leveraged to manage the lifecycle of resources. Ask if resources be organized into OUs, Folders, or Management Groups to help clarify resource ownership.

Two Motivations Driving Your Hierarchy Design

Every organization's hierarchical cloud design should be bespoken and unique to the business's stated objectives and internal structure. Laying out a cloud estate involves taking a close look at the key characteristics of the hierarchies. Once you see how these work, the answer as to the 'right' choice should become clearer.

Hierarchy Design Considerations

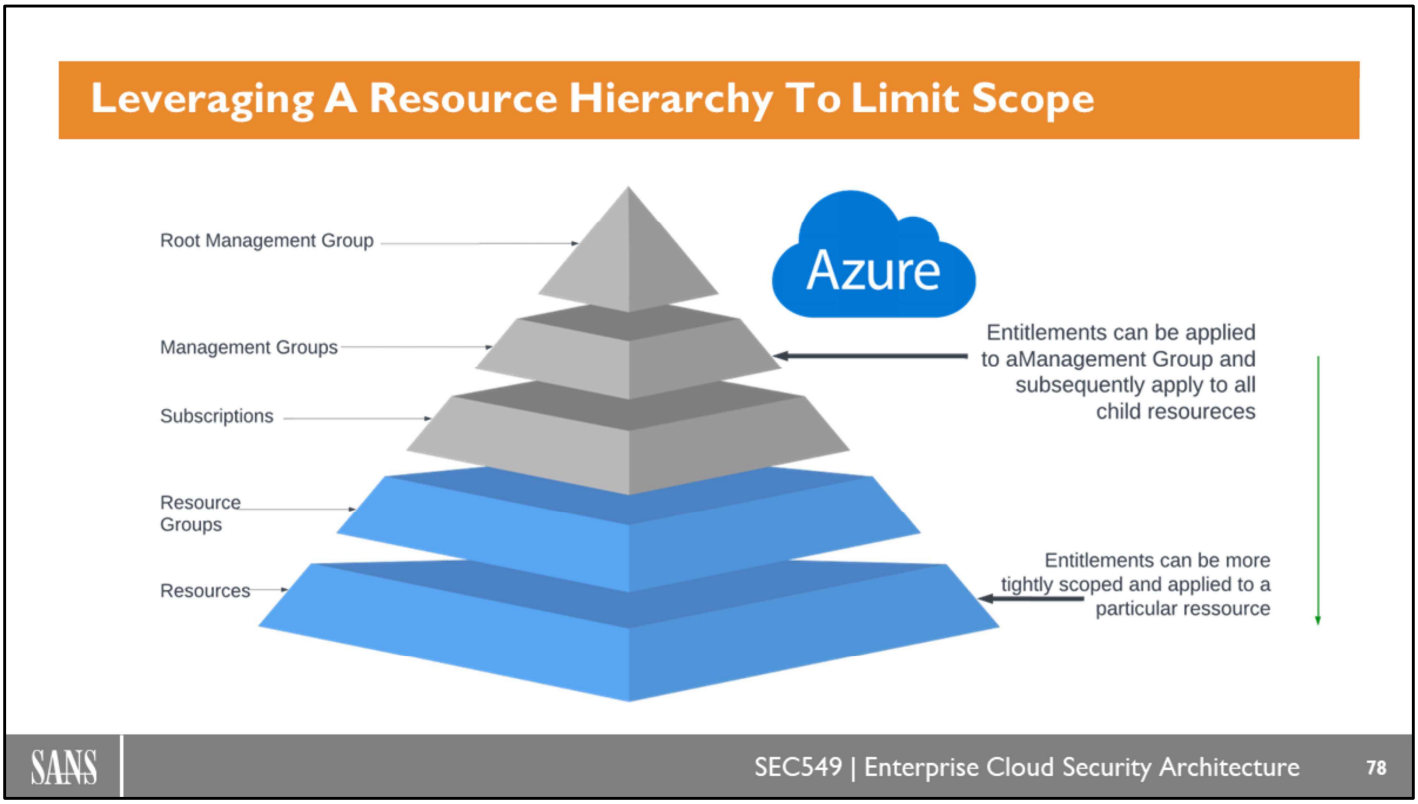
1. ***IAM Inheritance*** **–** Consider what points in the hierarchy IAM policy will be attached, giving particular attention to how permissions will be inherited by downstream resources.
2. ***Clear Resource Ownership*** **–** Consider if a cloud hierarchy structure can be leveraged to manage the lifecycle of resources. Ask if resources be organized into OUs, Folders, or Management Groups to help clarify resource ownership.

In any architectural design using the Big Three CSPs, you will want to keep their respective hierarchies in mind; take advantage of the inheritance model when applying the various policies within the structure.

Resource ownership also matters to the hierarchy design[1]. There are many practicalities to consider regarding resource ownership:

- When you delete a parent resource, the child resource(s) are automatically deleted as well.
- If you keep tightly scoped accounts (in any of the three CSPs), you will be better able to clean up resources when projects become decommissioned or transferred.
- Clear resource ownership in the Cloud helps give the business more visibility regarding costs. The more tightly scoped you keep the size of the accounts within the business, the better able is the business to attribute costs to specific initiatives and projects.

Reference:
[1]: https://sec549.com/id30



Leveraging A Resource Hierarchy

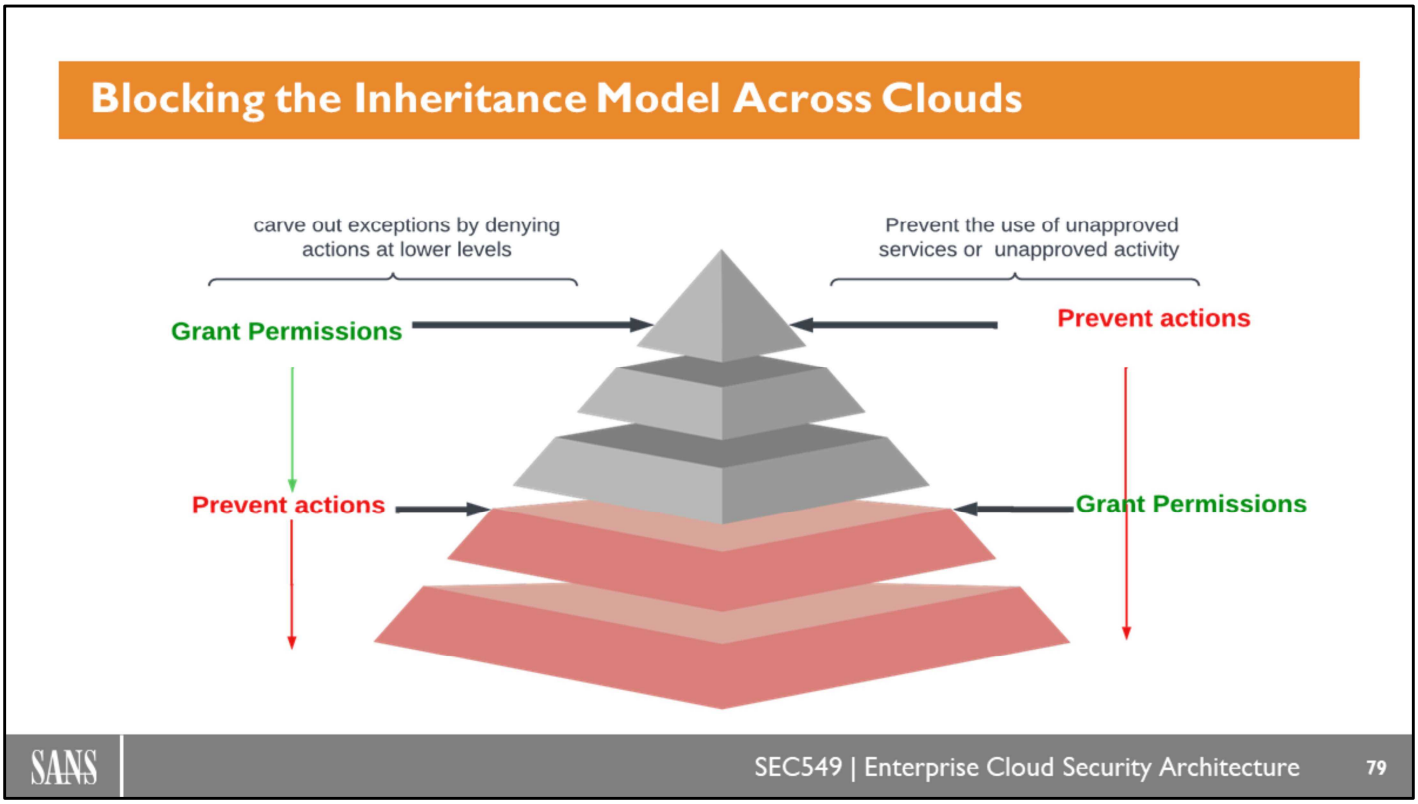
Azure and GCP built strong hierarchy models into their design from the outset. You can begin at the top and create nodes, with descendants or child resources beneath them in any necessary combination in order to leverage the inheritance of policy as it flows in a downward direction. In Azure and GCP both restrictive, guardrail policies and permissive, permission granting polices abide by the hierarchy inheritance model.

AWS didn't begin with a strong concept of hierarchy but has added some aspects of it in recent years. They introduced AWS Organizations Service, which provides the customer with the experience of central management of IAM policymaking along with the ability to attach restrictive, guardrail policies. The construction of OUs become convenient attachment points at which to apply a broad stroke of restrictive policies, similar to the way GCP Folders and Azure Management Groups are utilized.

Leverage The Resource Hierarchy To Reduce Permission Scope

In GCP and Azure Clouds, the scope which permissions and entitlements apply is dictated by which node in the resource hierarchy the role is applied. AWS does not have the concept of applying permissive policies at different levels of a resource hierarchy. Instead, to restrict the scope of granted permissions, the "Resources"[1] field in IAM Policy document is used to limit which resources the permission applies to.

Reference:
[1]: https://sec549.com/id196



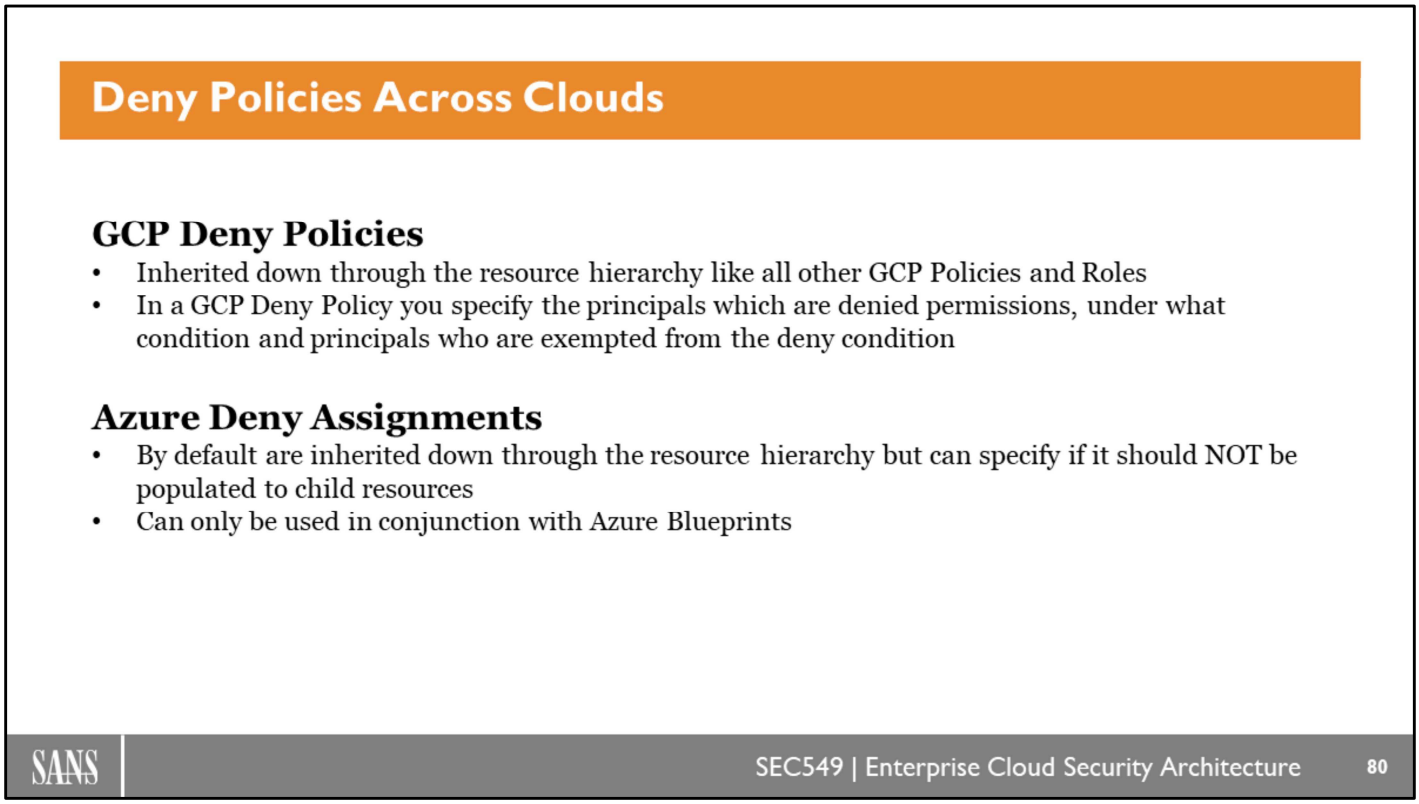
Blocking Permission Inheritance

The GCP and Azure Hierarchy models allow permissive, permission granting policies to be applied to higher level resources and cascade downwards to affect child resources. This is a powerful tool but there will be occasions when the IAM policy inheritance needs to be overridden. Let's take a look at the two approaches to blocking the inheritance model in both GCP and Azure.

Strategies for Blocking Inheritance

When mixing both permissive, permission granting policies with restrictive deny policies, two strategies are used to achieve different goals.

1. Broad, sweeping rules are often applied towards the top of a hierarchy. Examples include denying the assignment of GCP Roles to principals outside of the organization or preventing the creation of AWS IAM Users.
2. Conversely, restrictive Deny Policies can also be applied lower in the hierarchy to carve out exceptions or protect Projects which shouldn't inherit IAM policy from its parent resources.



Denying Actions
Each of these mechanisms the Big Three CSP offers are implemented differently but have one thing in common: they all are evaluated inline during the authorization of cloud control-plane APIs. Each mechanism will override any explicitly granted permissions

GCP Deny Policies[1]
Contrary to GCP Roles which grant permissions, GCP Deny Policies explicitly disallow permissions. They are evaluated by the GCP IAM service whenever a request is made to cloud control-plane API. Since Deny Policies are evaluated first, even if a Role grants permissions, Deny Policies will always take precedence.

- Examples of a Deny Policy you might create and apply at the Organization-level:
  - Prevent the creation of GCP Projects except if the caller is in the cloud administrators' group or your Project automation Service Account.
  - Prevent the deletion of GCP Projects except if the caller is in the cloud administrators' group or your Project automation Service Account under the condition that the project is not tagged as "dev".

Azure Deny Assignments[2]
Deny Assignments might seem similar to GCP Deny Policies. They are policies which can be applied to a scope to prevent a particular principal (or all principals) from performing specified actions. When a Deny Assignment is in effect, it will block a principal from performing an action in the cloud environment. The key difference between GCP Deny Policies and Azure Deny Assignments is the ability for cloud administrators to independently craft these custom, restrictive policies. At this time, Deny Assignments are not an independent feature of Azure RBAC. They are only used when resources are defined and deployed within the context of an Azure Blueprint.[3]

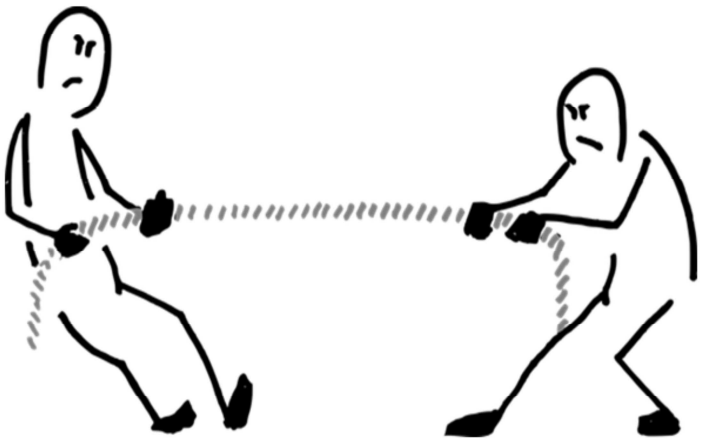
AWS
There is no 1-to-1 equivalent in AWS since there is no concept of a policy inheritance model. Instead, when granted permissions need to be overridden, IAM policies leverage the" Effect"[4] field to deny actions in the environment. In the AWS policy evaluation logic, policies which have the effect set to deny always override any allow policies which might grant permissions.

References:
[1]: https://sec549.com/id917
[2]: https://sec549.com/id198
[3]: https://sec549.com/id199
[4]: https://sec549.com/id199

Centralization versus Decentralization

Centralization of security controls is how you maintain a balance between the speed of the cloud and security guarantees.

Enforcing policy unilaterally without collaboration from impacted teams will lead to **FRICTION** and reduce the agility in the cloud.



Centralization Versus Decentralization

One of the main course 'takeaways' is that centralization of controls is highly beneficial to the architected cloud ecosystem. When you centralize policy enforcement in AWS, for example, you are exerting the only rational mechanism for maintaining security controls, especially as the business scales up in the cloud and wants to do it rapidly.

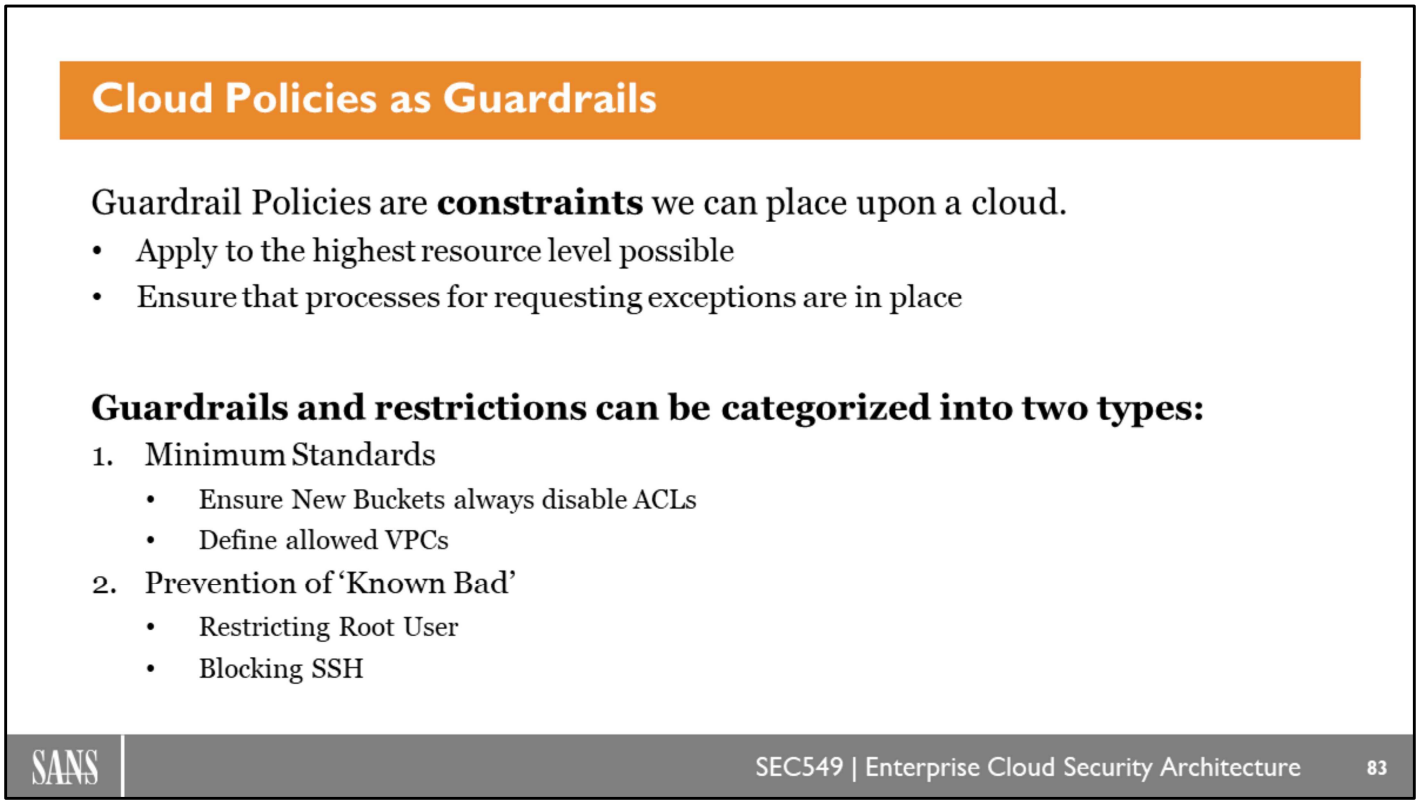
There are drawbacks, however, to centralization**.** These are summarized best in one word: **'FRICTION'.**

Migrating to the cloud is complex, no matter how well it is done. When engineers and management get together to discuss which regions to operate in, which services should be allowed, and which WAF rules to enforce, the security team may feel siloed and frustrated. Research and development can drag at a slower pace, leading to an increase in shadow accounts or the misuse of sandbox accounts by those wanting to circumvent the process.

In most compliance-focused organizations, it isn't possible to bargain out of centralized policy enforcement. Still, policy development should be collaborative. Before placing restrictive enforcement mechanisms upon swaths of cloud accounts, those responsible for the process must clearly communicate the critical impact of making these policy decisions. Published methods should be available for impacted teams who might want to request an exception.

The root node of the cloud estate may be owned by the Operations Team, Cloud Operations Team, or Security Team. Regardless of ownership, however, the decisions around centralized policy must take the needs of the internal customers into account prior to enforcing those decisions.

Reference:
Image Source: Pixaby user DeHaasbe: https://sec549.com/id32



Cloud Policies As Guardrails
Each of the Big-Three CSPs offer the opportunity to effectively use policies as high-level guardrails ensuring improved overall security of the organization's cloud presence. The key to successfully using guardrails in the Cloud is to leverage the policy inheritance principles in order to enforce restrictions across your environment. In the upcoming slides, we'll cover the mechanisms each cloud makes available for placing constraints on your cloud estate.

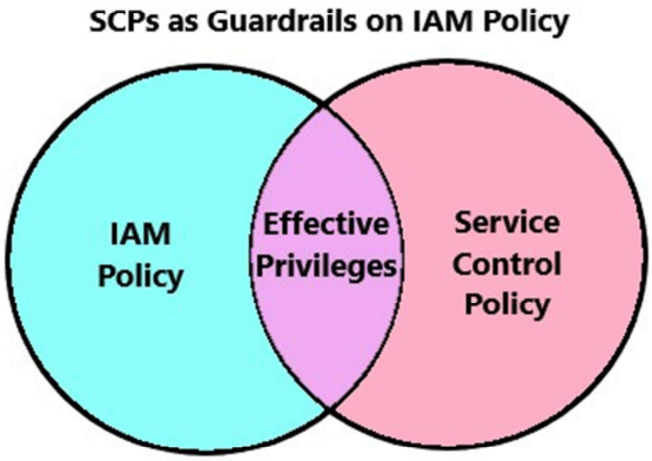
AWS
Available in AWS are a guardrail policy type called Service Control Policies (SCPs)[3]: These allow an administrator to define custom *deny* policies affecting actions taken in your Organization. When evaluating a principals authorization context, AWS always checks SCP first to determine if a particular action is denied, prior to determining if it is allowed. Using SCP requires AWS Organizations be enabled in full-feature mode.

GCP[5]
Google Cloud Platform has two mechanisms for placing restrictions on an environment. The first, GCP Organization Policy Constraints, is less comprehensive than AWS SCP. The guardrail elements of Organization Policy Constraints are not custom policies like AWS SCP. Instead, GCP defines Organization Policy Constraints centrally for all its customers. GCP is responsible for the structure of them; the impact they have on your infrastructure is only partially defined by you (the customer).
Contrary to Organization Policy Constraints, GCP Deny Policies[4] are crafted as custom policies by the customer. The format of Deny Policies allows the cloud administrator to restrict access to sensitive functionality while providing exceptions for designated teams.

Azure
Azure's guardrail policy offering is the most comprehensive of the three major clouds. Azure Policy is both a mechanism for enforcement of resource restrictions and a configuration assessment tool. Azure Policy can be used to assess resource state, deny resource creation and auto-remediate resources. Unlike GCP and AWS, Azure Policy can be used as a configuration management tool on new and existing resources. The policy engine that fuels Azure Policy will assess existing resources for non-compliance, reporting on their status in the Azure Security Center.

References:
[1]: https://sec549.com/id33
[2]: https://sec549.com/id34
[3]: https://sec549.com/id35
[4]: https://sec549.com/id195
[5]: https://sec549.com/id37

AWS | Service Control Policies (SCP)



SCP Rules:

- Does not restrict the Management Account
- Do not restrict principals outside of the organization
- Do not grant permissions (only denies actions)
- No audit mode available

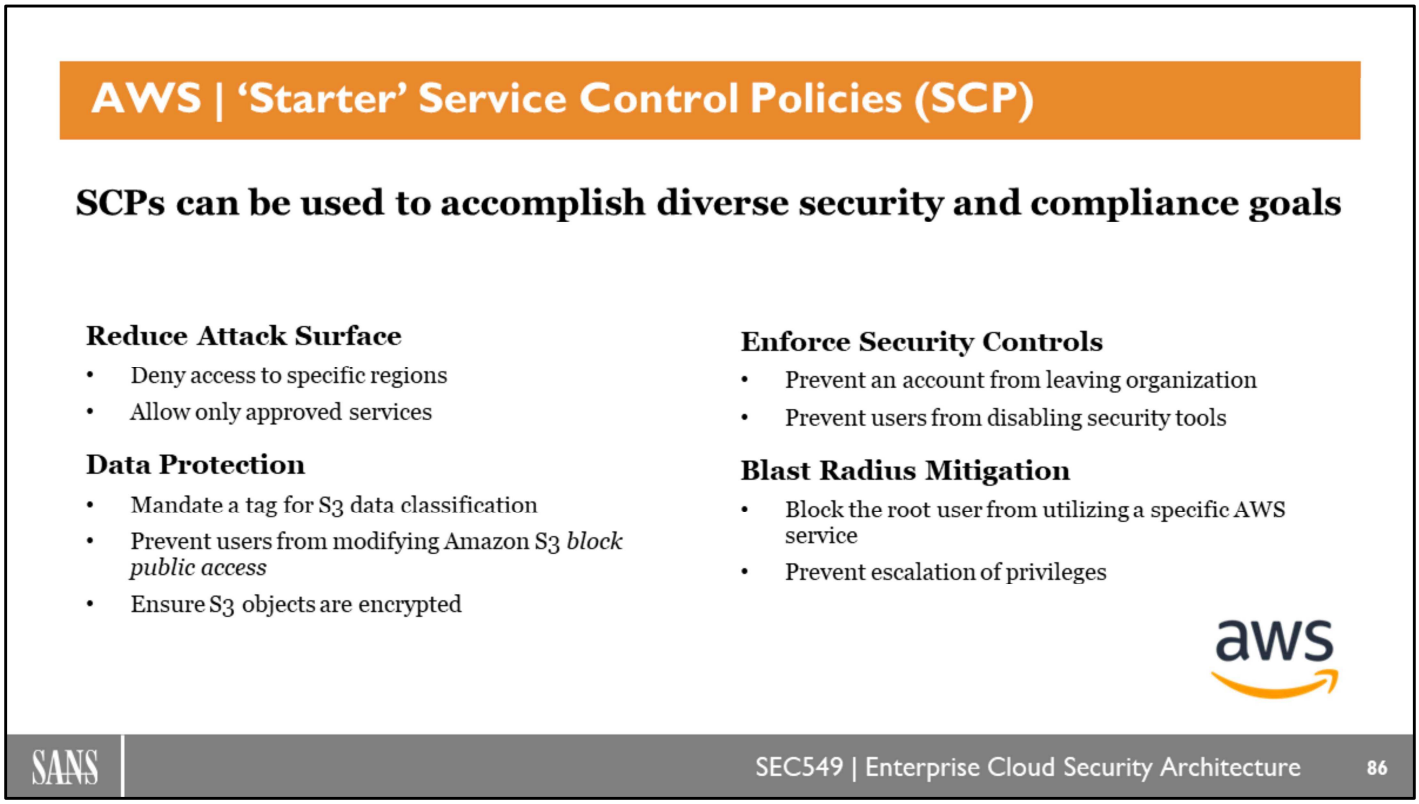
aws

AWS | Service Control Policies (SCP)
AWS SCPs are a feature of the AWS Organization administered by the Management Account. The goal of using them is to restrict the actions that can be taken on any AWS account. No User or Role in the affect Account (not even the root user) can perform an action denied by a SCP. SCPs do not grant permissions; they simply apply effective guardrails on IAM permissions granted to principals. This is accomplished by writing SCP documents defining actions which are explicitly denied under certain conditions.

IAM Policy Evaluation with SCP
What happens when an SCP prohibits an action by a user, but an IAM policy explicitly allows it? The action is denied. All actions on the platform are first passed through the SCP filter to see if they are allowed. Any deny within an SCP will result in a failed call to the AWS API. Keep in mind, SCP only apply to the principals in the same Organization. Actions taken by an external principal will not be restricted due to an SCP.

SCP Rules of the Road:
- SCPs cannot restrict the Management Account of the Organization.
- SCPs cannot restrict principals outside of the Organization.
- SCPs do not grant any permissions, they only deny or allow actions
- Unlike Azure Policy, there is no audit mode for SCP.
- Details of SCPs are invisible to principals in member accounts.

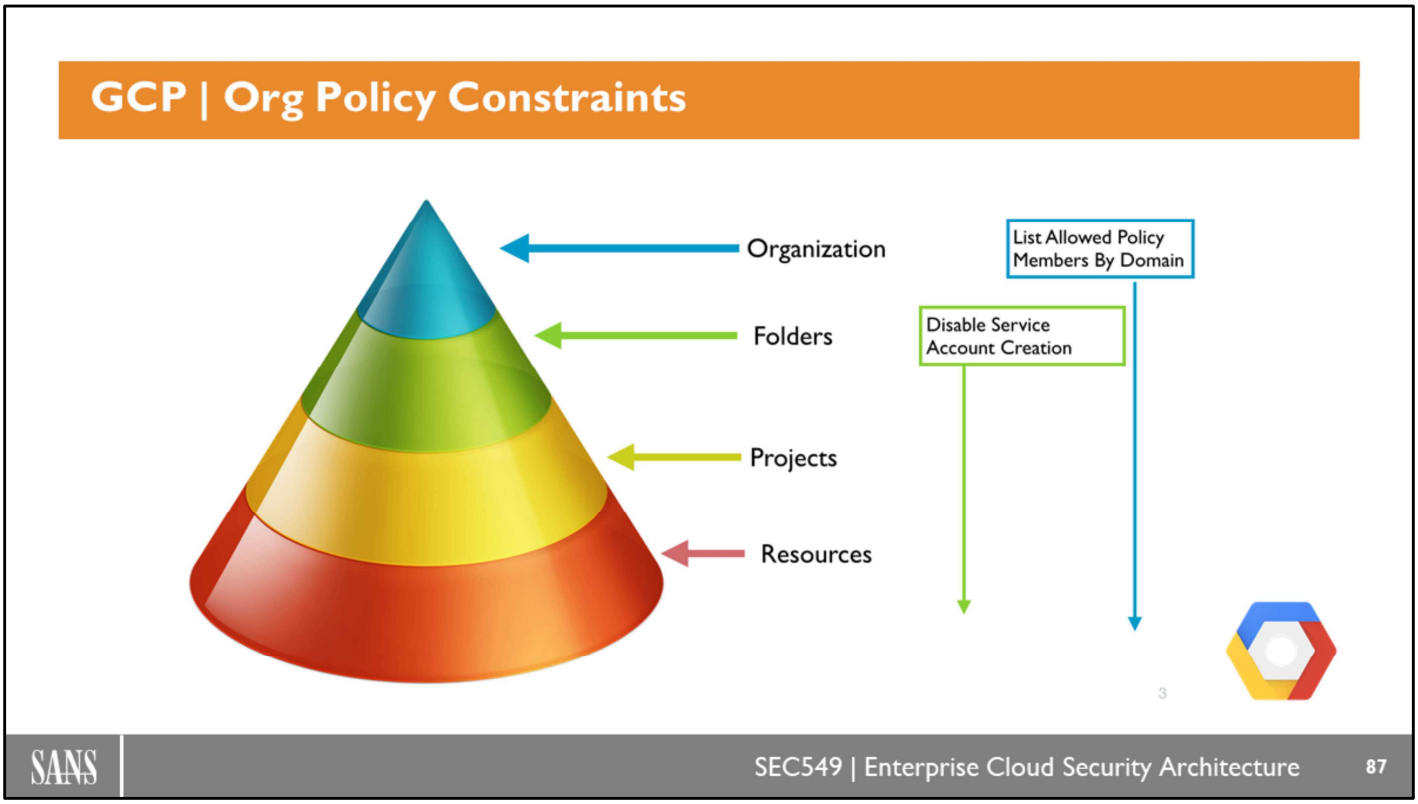
Limitations to SCPs:
- An SCP document can only be 5120 bytes in length
- Only 5 SCPs can be attached to any given node; the root, an OU, or an Account. This limitation makes it advantageous to collapse 'like' SCP restrictions into a single policy document.
- Because AWS Accounts inherit SCP from 'above', these aren't counted against the total limit applicable to an Account or OU.



AWS | 'Starter' Service Control Policies (SCPs)
AWS Organizations service does not come with automatically enabled SCPs. It is completely up to the customer to first enable Service Control Policy and then decide when and where to apply them in the organization's hierarchy[1]. To get customers going, AWS has described a number of 'starter' SCPs to utilize with a relative degree of safety. The purpose of these 'starter' SCPs[2] is to provide predictable guardrails that you can apply early on to safeguard the organization during the initial growth phases of a cloud migration.

You can select and apply an SCP crafted to deny resource creation in specific AWS region or regions, helpful to maintaining compliance or regulation agreements as you migrate into the cloud. One SCP restricts IAM users and roles from changing a specified IAM role you created. Another can keep AWS accounts from leaving the Organization. Still another SCP can be used to require MFA before any user or role can perform a specified action.

References:
[1]: https://sec549.com/id38
[2]: https://sec549.com/id39



GCP | Org Policy Constraints
Organization Policy Constraints is a type of guardrail policy which helps to enforce minimum standards across a GCP environment.
Constraints cannot be crafted by the customer. Instead, there is a suite of predefined Constraints by GCP that can be applied at the Organization, Folder, or Project Level.

Just like IAM Policy, Organization Policy Constraints are also inherited. Constraints are applied on a resource hierarchy node in order to enforce the restrictions on that node and each of its descendants.

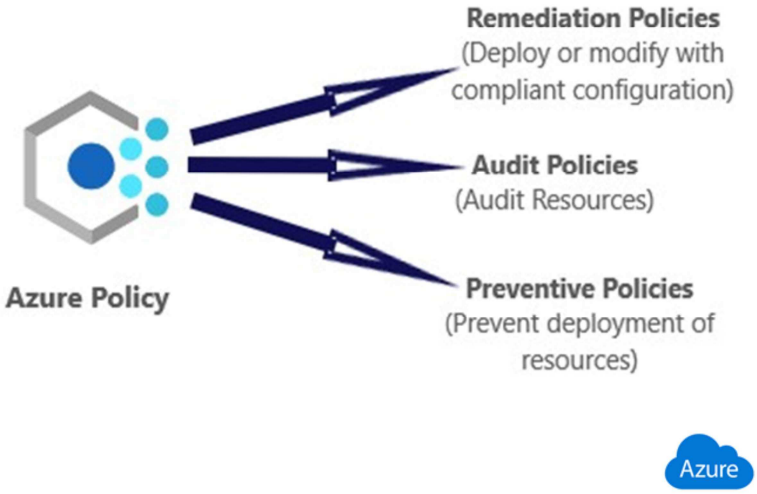
Organization Policy Constraint Types
Constraints come in two varieties: Boolean-based and list-based Constraints. An Org Policy Constraint which is list-based allows the customer to list or enumerate a set of allowed values. In the case of the Constraint "constraints/iam.allowedPolicyMemberDomains"[1], the Org Policy Admin can define the set of members by domain that can be added to Cloud IAM policies. This constraint would prevent an external User from being included in an IAM policy in your Organization. By default, all user identities can be added to Cloud IAM policies.

Organization Policy Constraint Use Cases[2]
Imagine a scenario where you would want to turn off the creation of Service Accounts except in a handful of designated Projects. You would achieve this by segregating your Projects into Folders and applying an Org Policy at the Folder in which you want to affect its descendants. With this pattern you could effectively exclude some Projects from restrictive Policy, either because they are considered Sandbox Projects with looser controls, or you have corralled a dangerous functionality like Service Account creation into a designed Project managed by your Security or Operations teams. When relying on Folder-applied Org Policy Constraints for security guarantees, the permission to move Projects between Folders becomes a potentially lucrative permission for an attacker and one to monitor for its use.

References:
[1] https://sec549.com/id42
[2] https://sec549.com/id40

Azure | Azure Policy

- **Remediate** non-compliant resources
- **Audit** resources for non-compliance
  - Deploy policies in audit mode
- **Prevent** the deployment of non-compliant resources
  - Similar capability to AWS SCP and GCP Org Policy Constraints



Azure Policy

In Azure, the ability to enforce top-down guardrails is bundled with a tool granting the administrator a larger scope of control of resources. As you'll see, the capabilities of Azure Policy is broader than GCP Org Policy Constraints and AWS Service Control Policy (SCP). In this module, we'll focus how to use Azure Policies for preventive guardrails, which involves a comparatively similar capability to GCP Org Policy Constraints and AWS SCP.

Azure – Azure Policy

Azure Policy is a configuration management tool in Azure that allows you to create policies that enforce/control the properties of a resource. It is used to evaluate any and all changes made to an Azure environment, regardless of how they are enacted. Even if end users make changes to the environment through the Azure Portal, Azure Console, PowerShell, or Azure CLI, all API calls pass through the Azure Resource Manager. It's at this control plane that Azure can consistently enforce Azure Policy.

What can Azure Policy do?

Azure policy documents support multiple effects in order to respond to a non-compliant resource.

Remediate non-compliance

- "DeployIfNotExists" deploys a companion resource defined in an ARM template when a non-compliant resource is created.
- "Modify" modifies a setting when a resource is deployed missing the required configuration.
- "Append" effect adds additional fields to a resource when it is deployed (updated or created).

Audit for non-compliance

- "Audit" mode allows a non-compliant resource to be created, but flags it as non-compliant.
- "AuditIfNotExists" mode reports on the absence of a resource of a configuration.

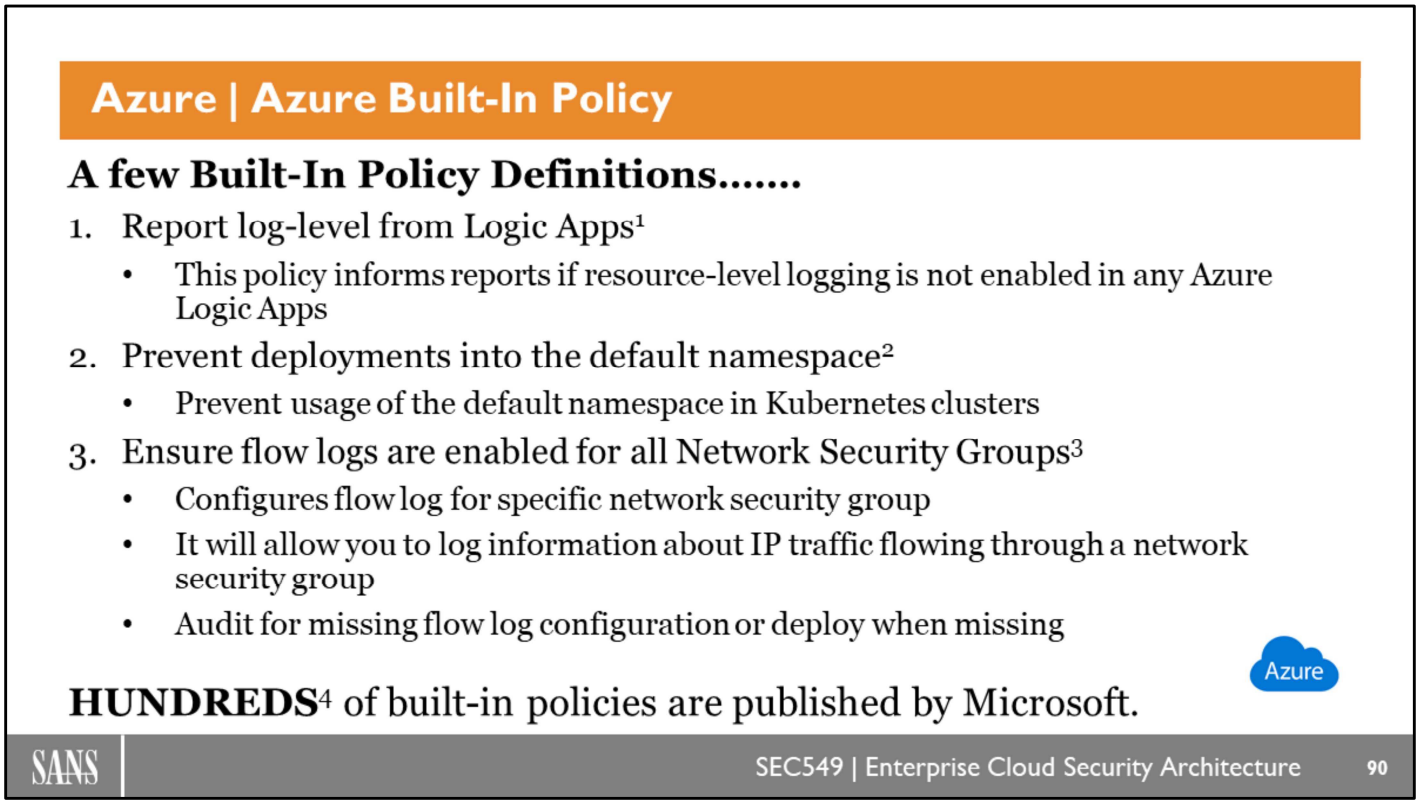
Prevent non-compliance
- "Deny" blocks the resource creation or update.

Policy Definition Templates
Azure Policy is extremely powerful from a security perspective because now we can start to enforce configurations using templates. When policies are written, the rules are written in JSON using deployable components described in Azure Resource Manager (ARM) templates. If an automation can be done with an ARM template, it can be enforced with Azure Policy.

ARM templates are powerful by themselves. With these, you can create and deploy entire Azure infrastructures in declarative form. You can deploy a VM, for example, and it will provide the means for the storage systems, network infrastructure, and any other resources needed.

Because of this power, it is strongly recommended to roll out all new policies in Audit Mode first. This way, all potentially non-compliant resources are reviewed first before any 'deny actions' impact live resources. In addition, it is crucial to protect your Policy definitions and decide who can assign Polices or edit your Policy definitions.

Reference:
[1]: https://sec549.com/id43



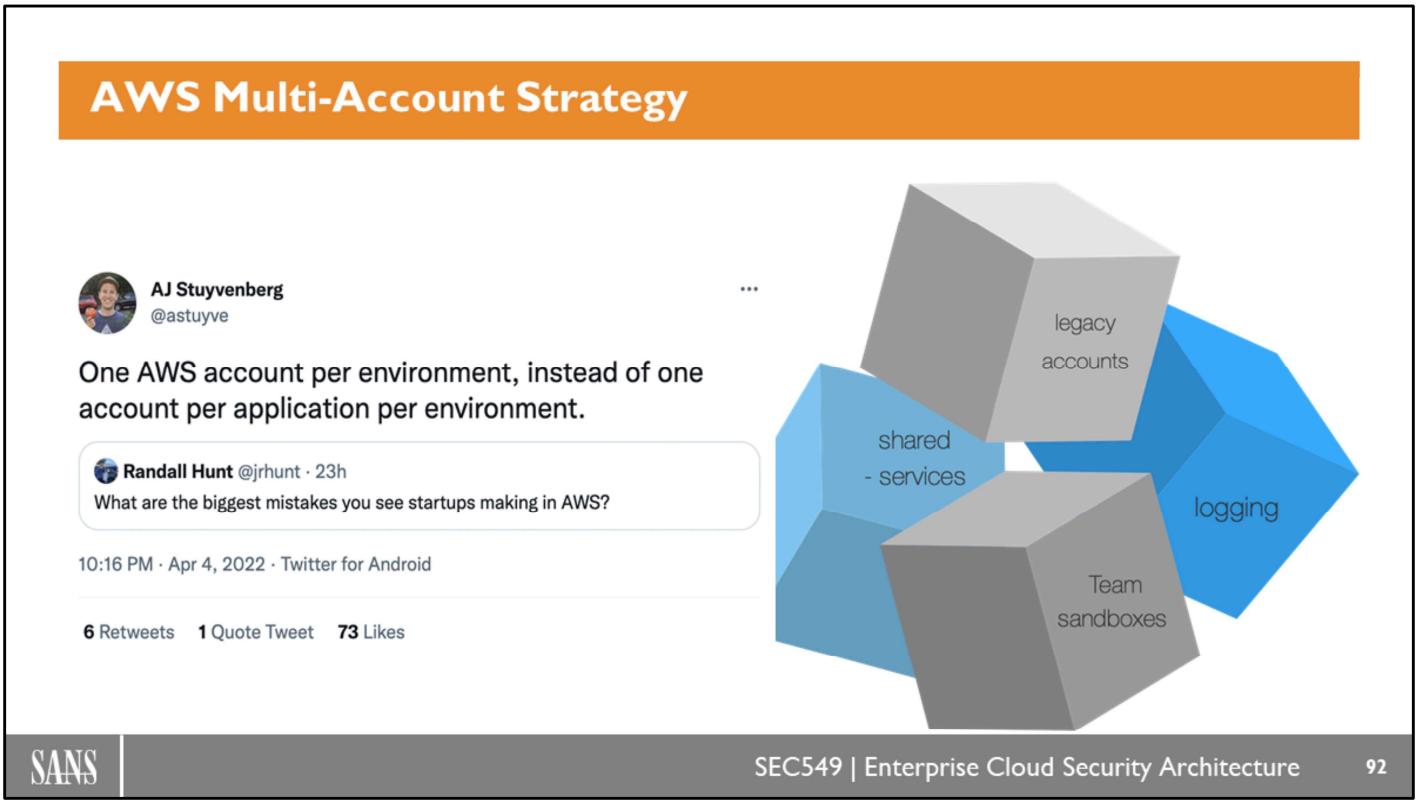
Azure Built-in Policy
One of the main differentiators for Azure is their ‘’Pre-built’ or Built-In’ Policy.

The Azure Policy is a service in Azure that permits you to define custom policies that enforce and manage the properties of a resource. But Azure customers do not need to produce the all policies from scratch. Hundreds of pre-built policy definitions are available for customers in the Azure Portal. These pre-built policies cover common scenarios for every service on the Azure Platform including guardrails for Compute, API Management, Key Vault, and Network.

References:
[1]: https://sec549.com/id44
[2]: https://sec549.com/id45
[3]: https://sec549.com/id46
[4]: https://sec549.com/id47

What would it look like if you could design an AWS Ecosystem from scratch, with no legacy dependency, taking advantage of Organizations, OUs, and tightly-scoped Accounts for each business purpose?

In this section, we will talk you through what a 'starter framework' might look like for a newly constructed AWS Organization.



AWS Multi-Account Strategy

The concept of the 'Account' in AWS is one to solidify into your thinking as a compartment for a defined function. Accounts are AWS's answer to the 'natural resource boundary' needed to segregate workloads of different types. By segregating different functions into different accounts, you gain the ability to apply different policies where appropriate.

With the advantages of Accounts and the ability to combine them in various ways, having one large account with all your workloads piled into it is neither necessary nor a good idea. One wrong move, even by someone experienced, can upset or destroy the nest with all the AWS eggs inside. Most cloud security professionals would argue against taking such an unnecessary risk.

Why have a Multi-Account Strategy?

The main reason to have a multi-account strategy from an architectural aspect is *isolation, isolation, isolation*. Having many accounts with natural resource boundaries automatically limits the blast radius in situations of unauthorized activity or unwanted access to any part of your organization.

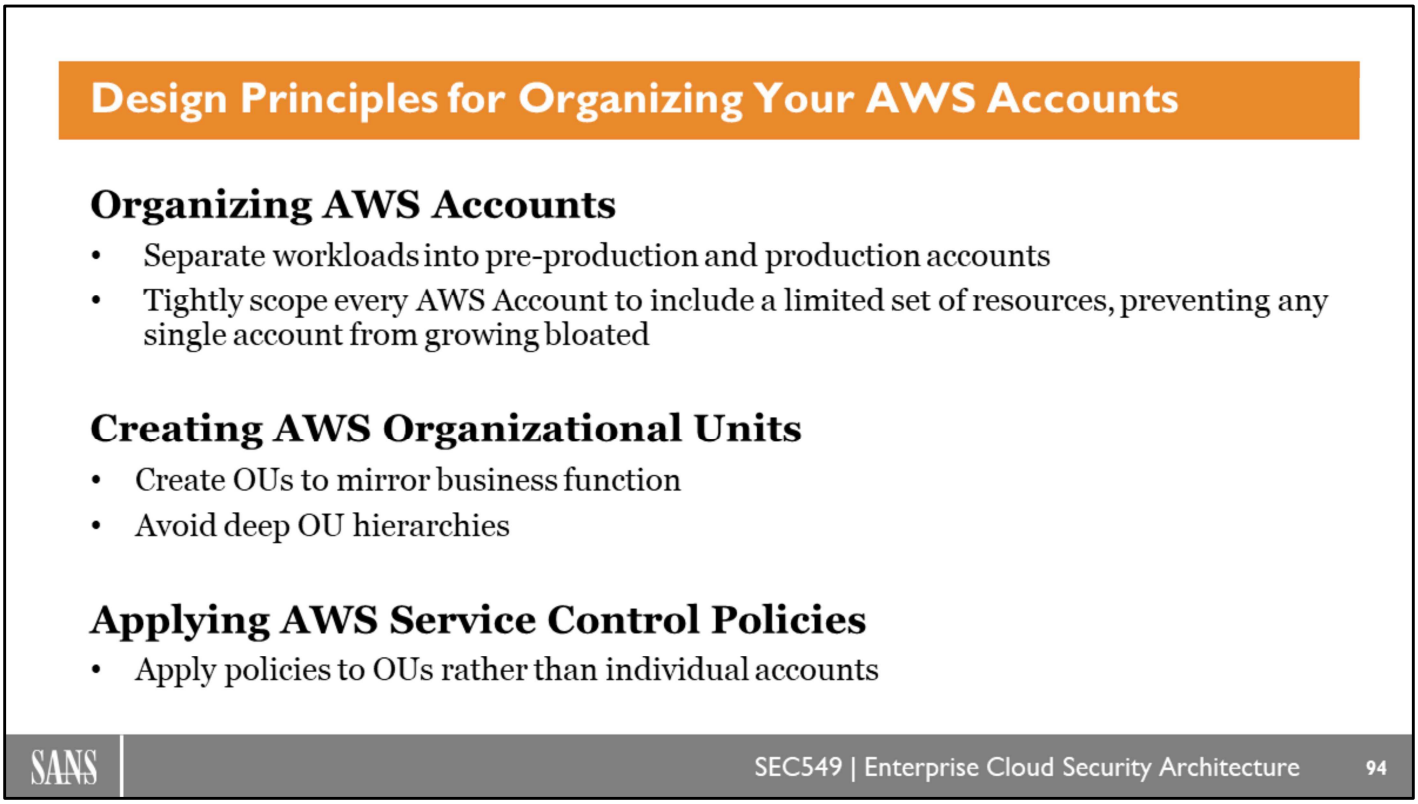
Other good reasons to have multiple accounts are these:

- Cost allocation – you can simplify your billing structure and will have more transparency into when and where charges are incurred
- Support diverse team needs – you can segregate groups of accounts in ways that allow the people who will work together remain in the same cluster. This also simplifies your ability to set policies on accounts that will need them as a unit.
- Audit and compliance – regulatory requirements for HIPAA, for example, include the need to make sure each account is in compliance. This is easier to do when you have multiple small auditable accounts.
- You also extend AWS API exhaustion when you use multiple small accounts

Are there trade-offs to consider as you scale from a single account toward what may be hundreds or thousands of accounts?
Anytime you are managing thousands of AWS accounts, the upkeep will be challenging, even when you have them well organized. For this reason, the only realistic way to do this at the speed needed by most businesses is through automation. An excellent 5-day deep dive course into automating security controls in the cloud can be found in **SEC540: Cloud Security and DevSecOps Automation.**

Management Account
Your management account is the AWS account used to begin managing your business at the hub of your landing zone. This account can manage your entire organization by itself and will innately contain those things needed to manage the Organization. Your Management Account should never contain compute resources. This top-level account has taken many names over the years. The terms "Root Account", "Payer Account", Master Account" and "Organizational Account" all refer to the same Management Account.

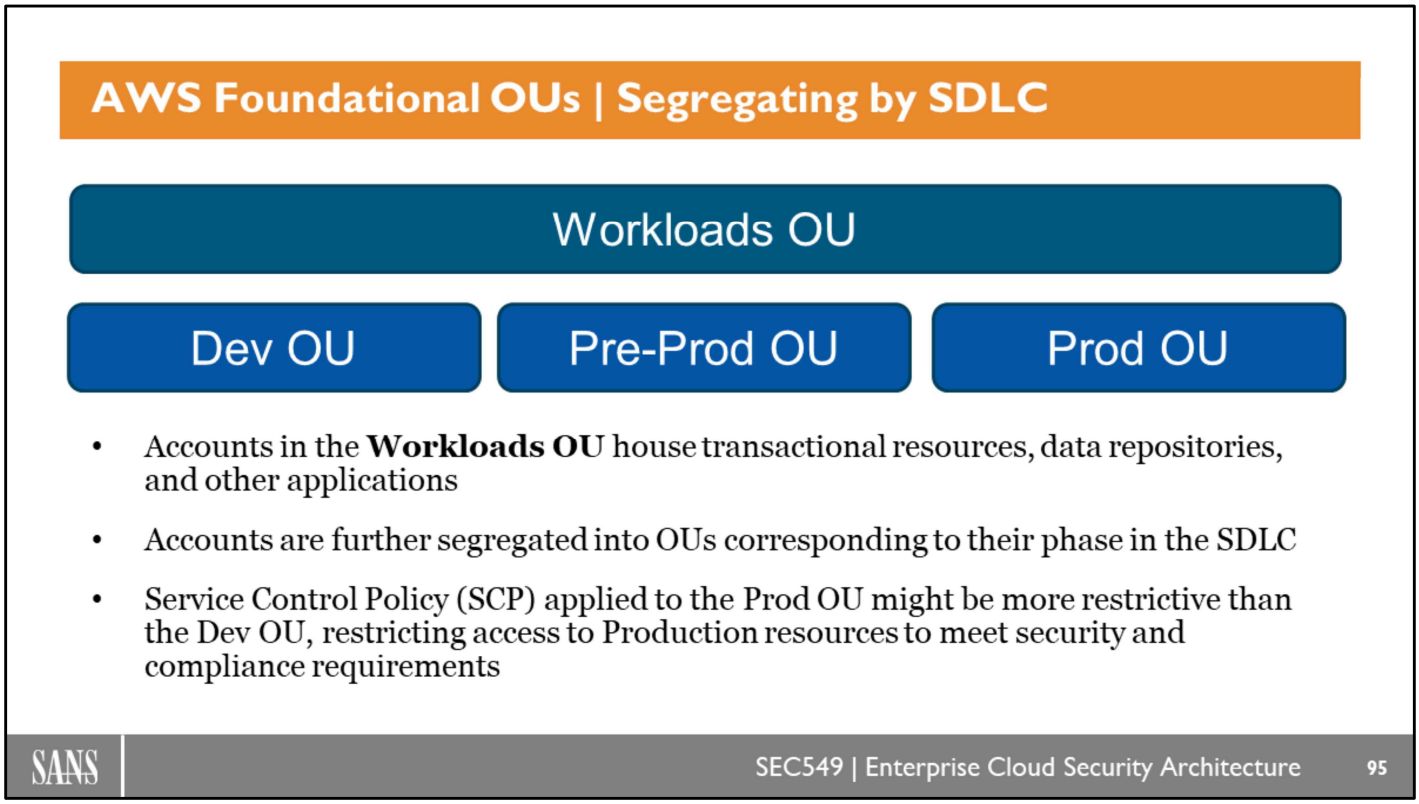
Reference:
Image Credit: Template for Keynote



Principles for organizing Accounts, creating Organizational Units (OUs), and applying Service Control Policy (SCP)

AWS has designed best practices for scoping account AWS accounts, organization those accounts into Organizational Units (OUs), and applying policy as guardrails.

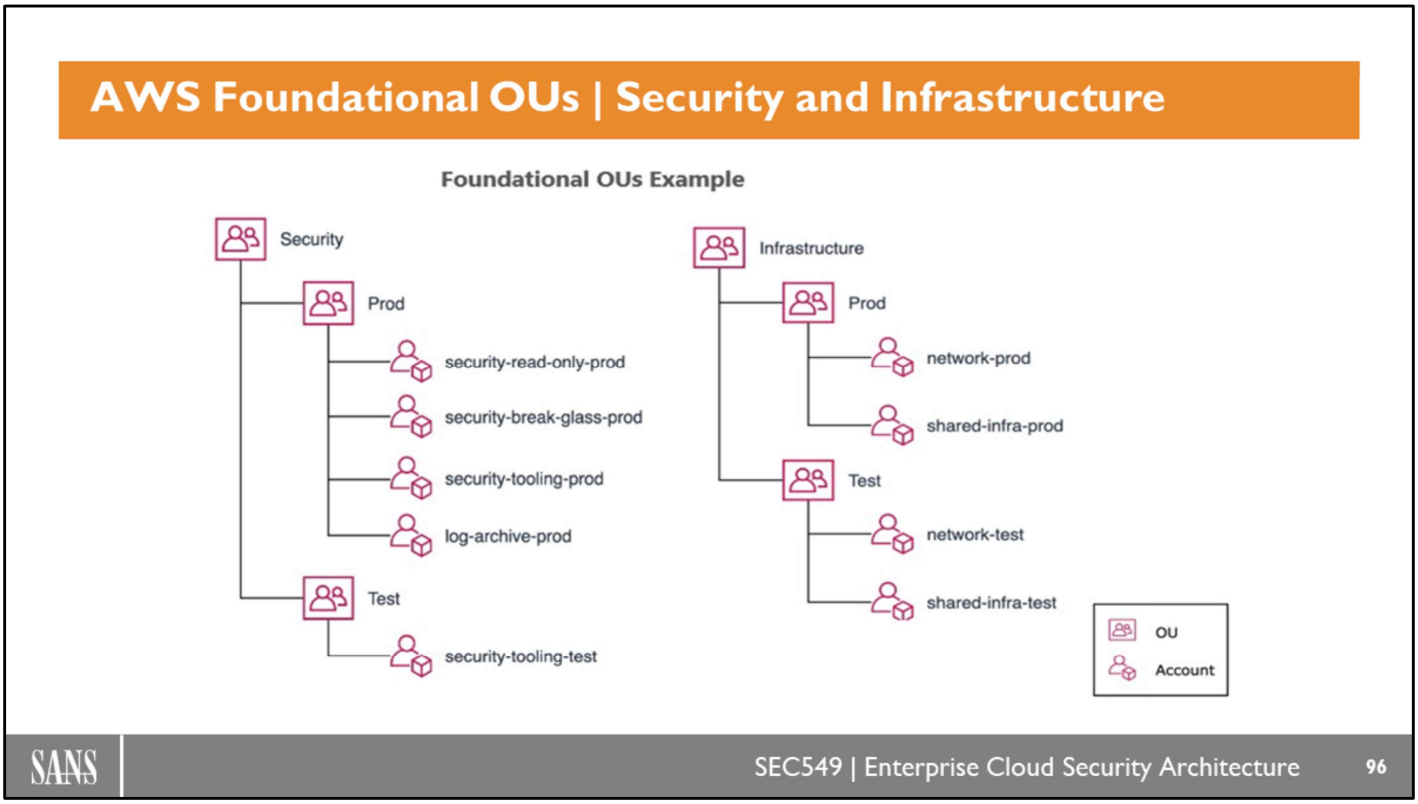
- Segregate your OUs based on function or a common set of controls rather than the organization's reporting structure.
- SCPs are best applied to OUs rather than Accounts. You can apply policies more easily when all Accounts in an entire OU can benefit from a single set of policies. This helps you efficiently manage guardrails across similar accounts.
- Avoid architectures with deep OUs. AWS Organization supports a 5-level depth structure with regard to OUs, but you should know when it is valuable or not to use all of these levels.
- Start small. There is a subset of foundational OUs that everyone should have. In the upcoming slides, we will cover foundational OUs in an AWS Organization and the common types of Accounts which might organize them.
- No workloads should ever live in the management Account!
- Non-production and production workloads should have separate Accounts and be further segmented into separate OUs.
- Tightly scope the resources which live in any single Account, assigning to them single workloads or a small, related set of workloads



AWS Foundational OUs | Segregation of Workloads by SDLC
The Workloads OU is intended to organize those Accounts that house applications, resources and data. In accordance with a well-segmented Software Development Lifecyle (SDLC), separate environments are maintained for each stage of the SDLC. Separate Accounts are maintained to to allow continuous deployment / continuous integration / continuous testing (CD/CI/CT) activities to occur without impacting production builds, allowing for speed while increasing scale.

When workloads move to the Cloud, it is important to maintain the same segregation of environments by keeping each deployment in its own Account. Since these Accounts will often have different security restrictions, its recommended to organize them into separate OUs to allow for different policies to be attached to each stage of the SDLC environments.

Maintaining Separate Environments for Non-Workload Accounts
Maintaining separate Accounts for production and non-production isn't only a necessity for the Accounts containing transactional workloads; it is also a consideration for Accounts that do not house classic application-like resources. Separate Prod/Non-Prod Accounts might be maintained for CI/CD Pipelines, Infrastructure Accounts, and Security Accounts.



AWS Foundational OUs | Security and Infrastructure
Once your organization is up and running, you will possibly have thousands of accounts and numerous OUs. Rather than create all of these possible Accounts in a flat structure, you would begin by first identifying what your foundational OUs should look like.

- What must you have?

- Operating from the initial management account you started with, where will your first foundational OUs begin to take structure?

Security OU
An Organizational Unit (OU) should be created to house activities from your Security, Operations, and Incident Response capabilities. Examples of these accounts could include:

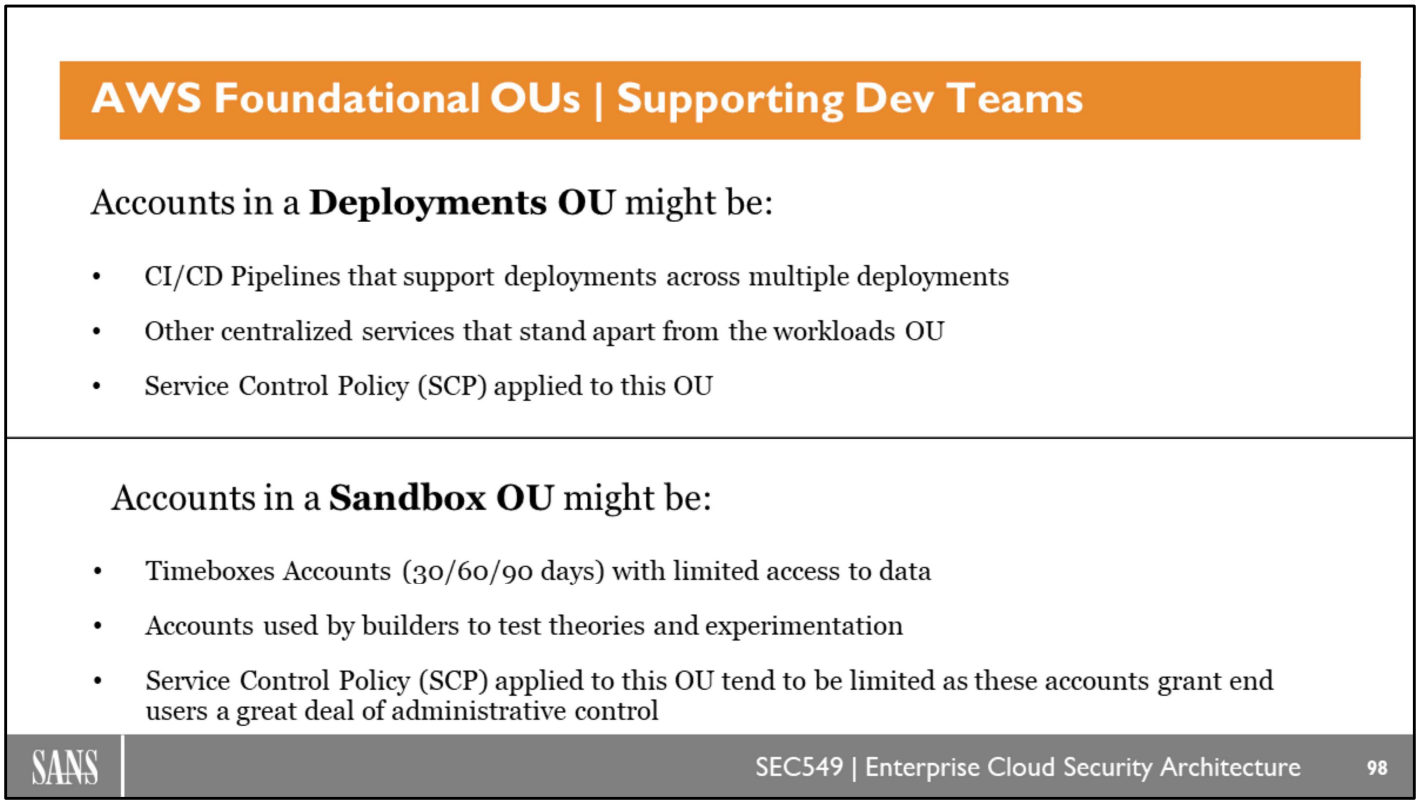
- ***Log Archive Account*** – this is where all log data is gathered from each Account in the organization.
- ***Security Tooling Account*** – this is where security services, supporting data, and other tools are housed. If your Organization utilized cloud-native tooling such as AWS Security Hub, AWS Config, and Amazon GuardDuty, they are recommended to be operationalized in a unique account, under the control of your in-house security team.
- ***Security Read-only Access Account*** – a Read-Only Access Account might be needed for a Security Organization to perform investigations if Organization-wide SSO is not enabled. A Read-Only Access Account is a hub-and-spoke pattern for access into all other AWS Accounts and follows the Identity-Bastion Account pattern, which is covered in depth in a later slide.
- ***Security Break-glass Account*** – this is a rarely-to-be-accessed account in case an incident needs urgent resolution and administrative access into an account is required. Access should be temporary, related only to a specific incident and result in an alert generated in your central SIEM.

Infrastructure OU

The Infrastructure OU is managed by your infrastructure teams, who will also manage all child OUs and their associated accounts. This is the place for centralized management of shared networking resources, such as AWS Direct Connect Integrations, DNS services, VPC endpoints, AWS Site-to-Site VPN connections, and shared VPCs. Any networking services that are centrally created and shared among consuming accounts might go into this OU.

As with security accounts, you will want to separate each segment into pre-prod and prod accounts. Your Network-Prod Account contains stable network capabilities that can be shared among relying development and deployment teams.

There will be pre-prod accounts used for testing and validating changes to network configurations. Keep in mind, networks maintained for development workload Accounts in the Network OU might still be considered a production network. A development Account in the Network OU might strictly be used by the infrastructure team. Dev network accounts are useful for maintaining stability in the production services while testing upgrades.



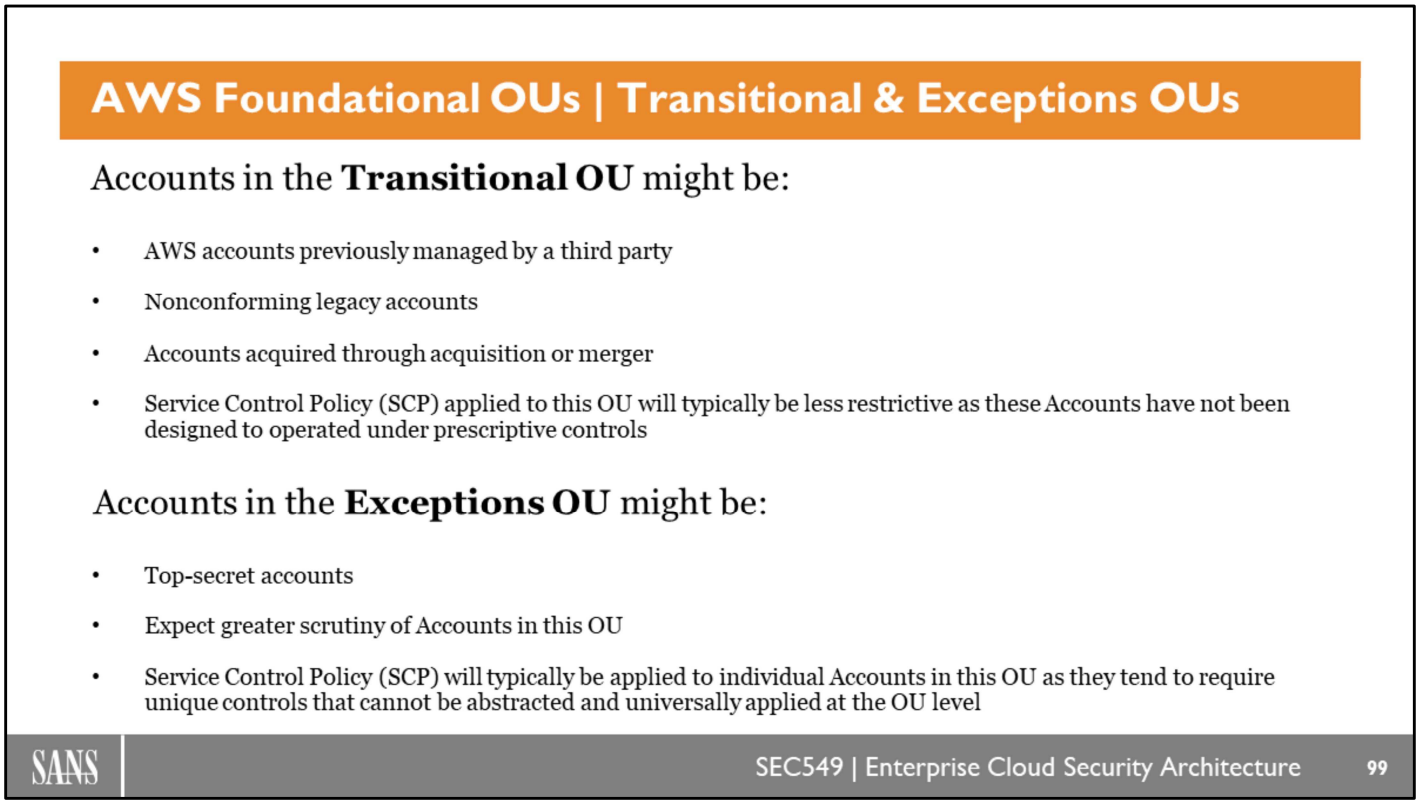
AWS Foundational OUs | Supporting Development Teams
Now that the foundational OUs have been created, your new organization needs supporting structures to enable development activities.

Deployment OU Accounts
Most development teams will find value in having access to sandbox accounts, accounts with limited network and data access, and accounts for CI/CD.

CI/CD pipeline accounts are also isolated from the rest of the organization's resources to be able to develop and test code before deploying anything for your organization. These accounts may become the location where cloud-native tools are leveraged for continuous deployment and testing such as AWS CodeDeploy or Amplify. Alternatively, a CI/CD Pipeline Account could be a landing zone for ingesting builds from external systems, such as Github or an on-premises Jenkins server.

Sandbox OU and Experimental Accounts
A Sandbox OU is the perfect place for your builders to safely experiment and test their code in an isolated environment.

Each builder or team should have their own sandbox accounts and a capped spending budget. Recycling sandbox accounts is an important hygiene practice, not only to keep costs in check but to ensure shadow infrastructure is not built into these accounts, which often operate outside of governance which more lax controls.



AWS Foundational OUs | Transitional and Exceptions OUs

Transitional OU[1]

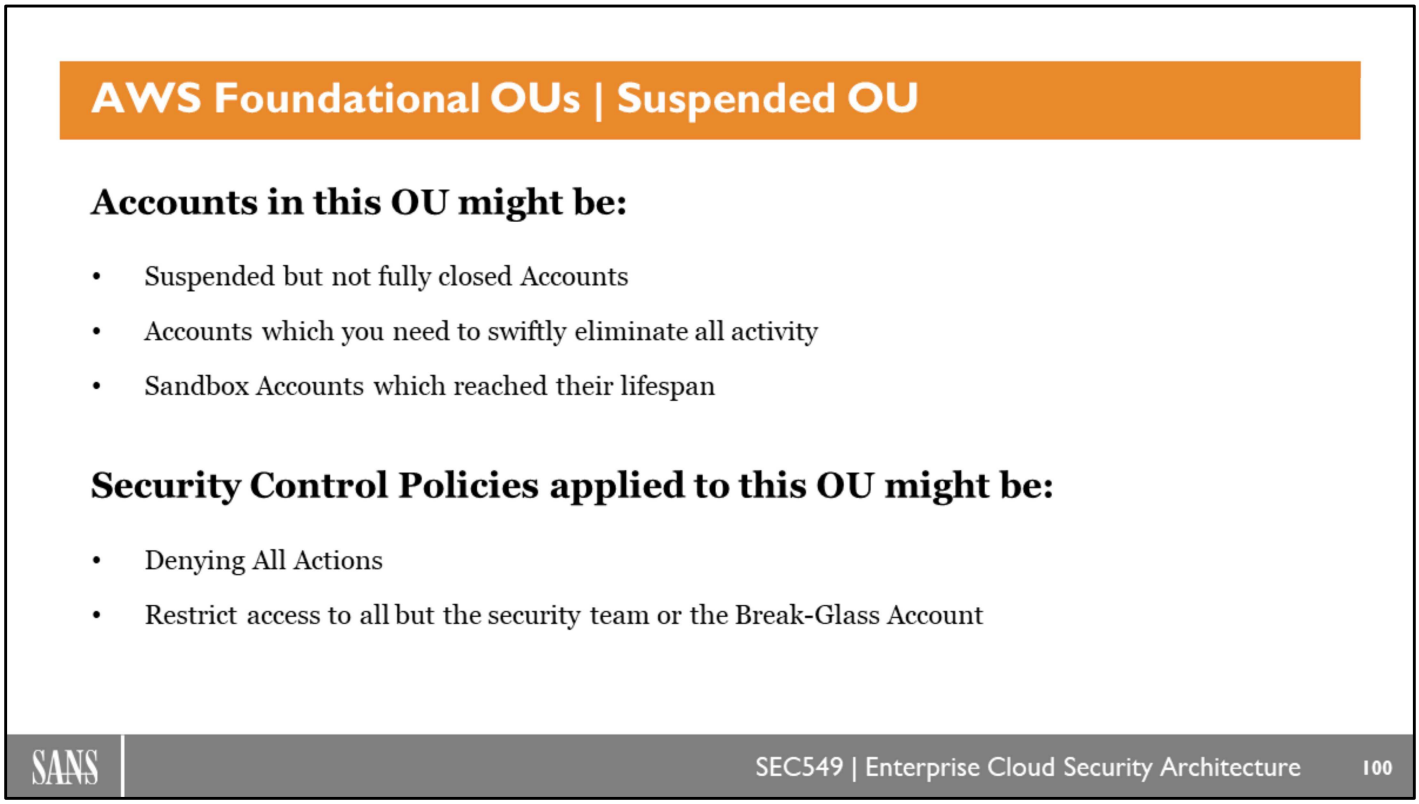
Unless you are truly starting from scratch, you may need a Transitional OU to handle existing workloads and accounts not yet fully integrated into your Organization. If you acquire accounts through an acquisition or merger, for example, some accounts can be 'parked' here until they can be integrated into AWS. Some of these Accounts might later be dissolved as functionality is rebuilt into more compliant Accounts and placed into an appropriate part of the Workloads OU.

Exceptions OU[2]

You may need an Exceptions OU to house accounts that are workloads needing an exception to the typical security policies already applied to the Workloads OU. You should have as few of these as possible and good reasons for these accounts to need an exception. SCPs are applied not at the OU but at individual accounts residing in it. You should periodically reevaluate the need for an account needing a security exception.

A good example of an Exceptions OU account is one that contains a top-secret project. The SCPs applied to it depend on what the account is used for and whether compliance issues are involved. The outcome of these accounts might be that they remain in the Exceptions OU, become part of the Workload OU, or spin off into a new initiative and Organization by themselves.

References:
[1]: https://sec549.com/id48
[2]: https://sec549.com/id49



AWS Foundational OUs | Suspended OU

Starting an AWS Organization is easy. Anyone can create an AWS Account simply by clicking a button and naming it. Closing these accounts, however, is not as simple. It involves several steps and cannot be automated. Once an account is closed, it takes 30 additional days to fully terminate it. One option is to put these accounts into the Suspended OU, restricting all inbound and outbound traffic and applying an SCP that denies all API access.

Options for accounts to include in this OU include sandbox accounts that are normally transiently but have reached the end of their lifecycle and are simply waiting to be officially terminated. If there is an active investigation on an account and it needs to be retained, the account can be frozen until its fate has been decided on.

Accounts are not generally kept here indefinitely. It is often a placeholder OU after a person has left the company or the account's resources have migrated elsewhere. You still need to officially close your unused accounts; however, they can be safely held in this OU until they finish the termination process.

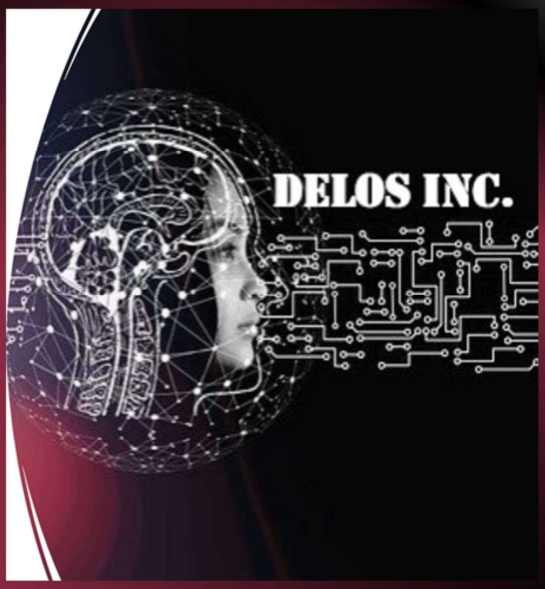
Cloud Journey Phase: Centralizing Identity Provisioning

Scenario:

Planning and preparation for Delos Digital Transformation is in full swing. They've bought you on as a consultant to help them navigate their transformation into a cloud-first organization.

To enable third-party access into their AWS estate, they're been creating individual users in the AWS Identity Center built-in directory. As a one of the most pressing tasks, you've advised them to ensure all cloud access originates from a central identity provisioning system.

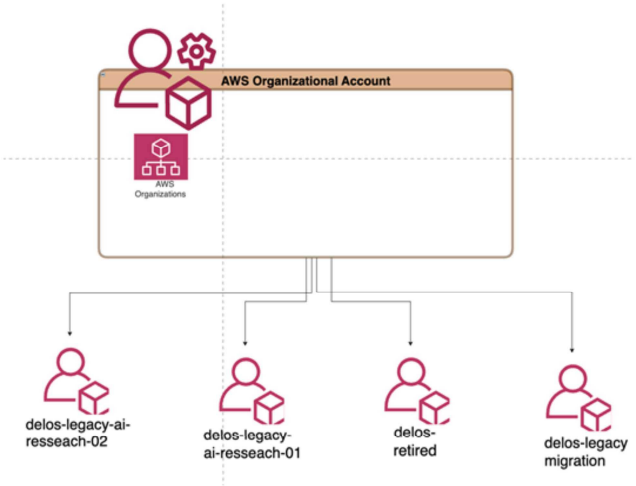
Help Delos plan for centralized account provisioning from Azure Active Directory where the scope of users and groups is based on group membership.



Reference:
Image Source: Pixaby user geralt: https://sec549.com/id51

Lab 1.3 – Current State Architecture

- AWS Organizations has been enabled, unlocking the possibility of creating 'member accounts' and OUs
- Previous 'shadow accounts' have been extended invitations to join the Organization and the account holders have accepted
- The Delos AWS Organization needs to be scaffolded out to prepare for the upcoming expansion into the cloud



Lab 1.3 – Creating Effective Hierarchies: Current State Architecture
Delos, Inc. has created their first Organization. After creating the Organization, IT invited the existing AWS Accounts and integrated them into the new Org.

With phased cloud migration pending, the new Delos AWS Organization needs to be configured to support capabilities that will be built out, such as centralized networks, customer-facing workloads, and security services.

Lab 1.3 – Creating Effective Hierarchies

Estimated Time: 45 minutes

In the labs, you will arrange three different resources to help Delos transition their operations from a data center to a hybrid cloud architecture. Organizational Units will be created to organize the initial set of AWS Accounts Delos plans to launch in their new Organization. Security Control Policies need to be applied in judicious manor to ensure their migration can proceed with confidence and speed. In this exercise, you are helping them create a solid identity foundation from which they will build their new cloud presence and design federation patterns.

Preparation

- View **Lab 1.3** in the course workbook
- Open the workbook in an incognito window
- https://workbook.sec549.com
  - Username: H03-student
  - Password: million-sailor-DEAR

Objectives

1. Arrange AWS Organization Units and AWS Accounts within a logical hierarchy.
2. Attach Service Control Policy (SCP) to logical nodes.
3. Challenge yourself to decide where on the hierarchy it would be best to apply SCP to support the accounts' wide-ranging needs.

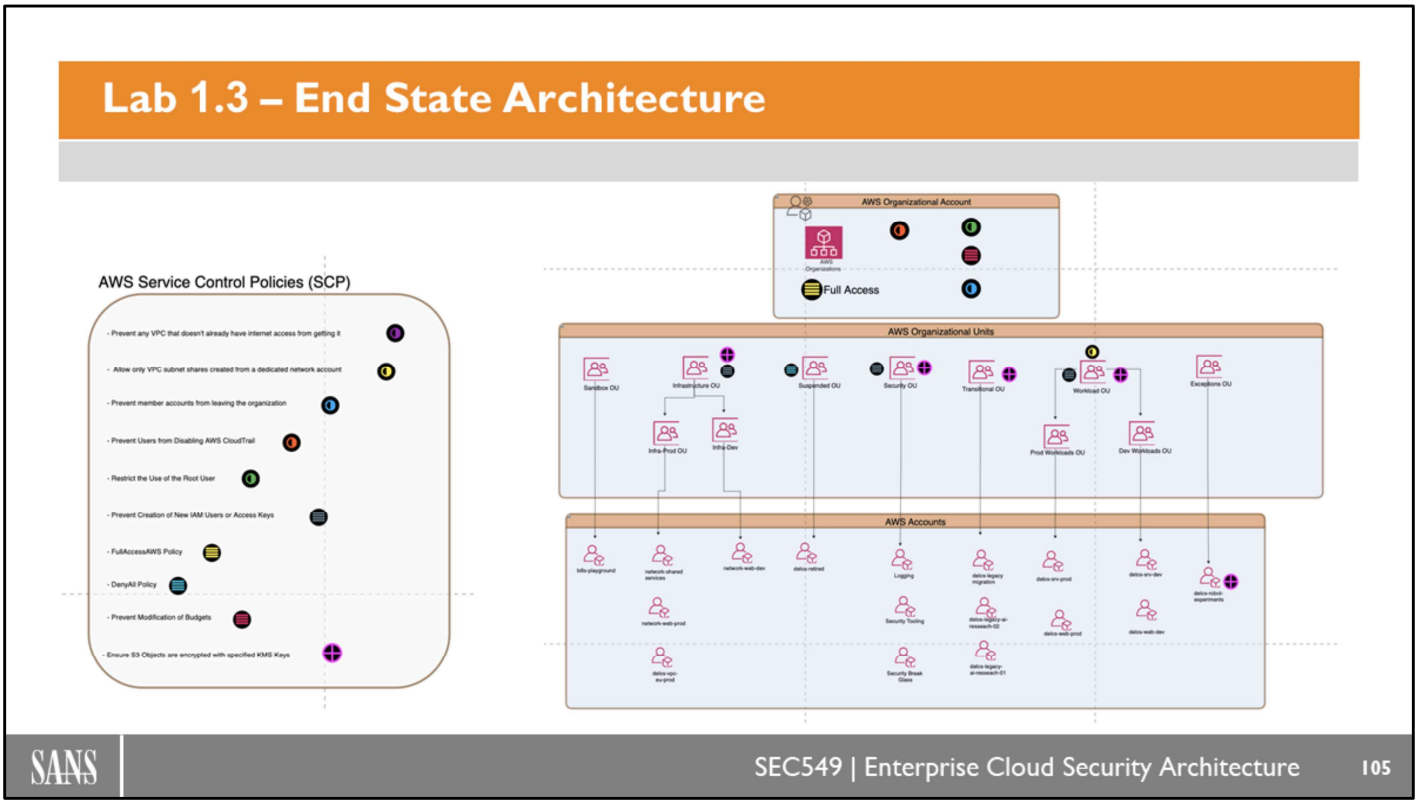
This page intentionally left blank.

Lab 1.3 – Summary

In this lab, you...

- Arranged the AWS Foundational OUs in a logical hierarchy to support various workloads, including security activities, playgrounds, and workloads, segregated according to SDLC environment
- Created a foundation that will help Delos scale up during their transition to the cloud
- Applied Service Control Policies (SCP) at various OU Levels to enforce policy
- Logged into the AWS Console to answer the ⚡ See it in Action ⚡ questions

This page intentionally left blank.



Lab 1.3 – Creating Effective Hierarchies: End State Architecture
Here's what a completed Lab 1.3 might look like. Of course, some of the placements of the SCP could be considered subjective.

There is a set of SCPs that probably will be appropriate to apply at the AWS Root. These include:

- Preventing a member account from leaving the organization
- Preventing users from disabling CloudTrail
- Restricting the use of the Root User

Remember, none of these policies will apply to the Organization Account when applied at the Root, only to the member accounts.

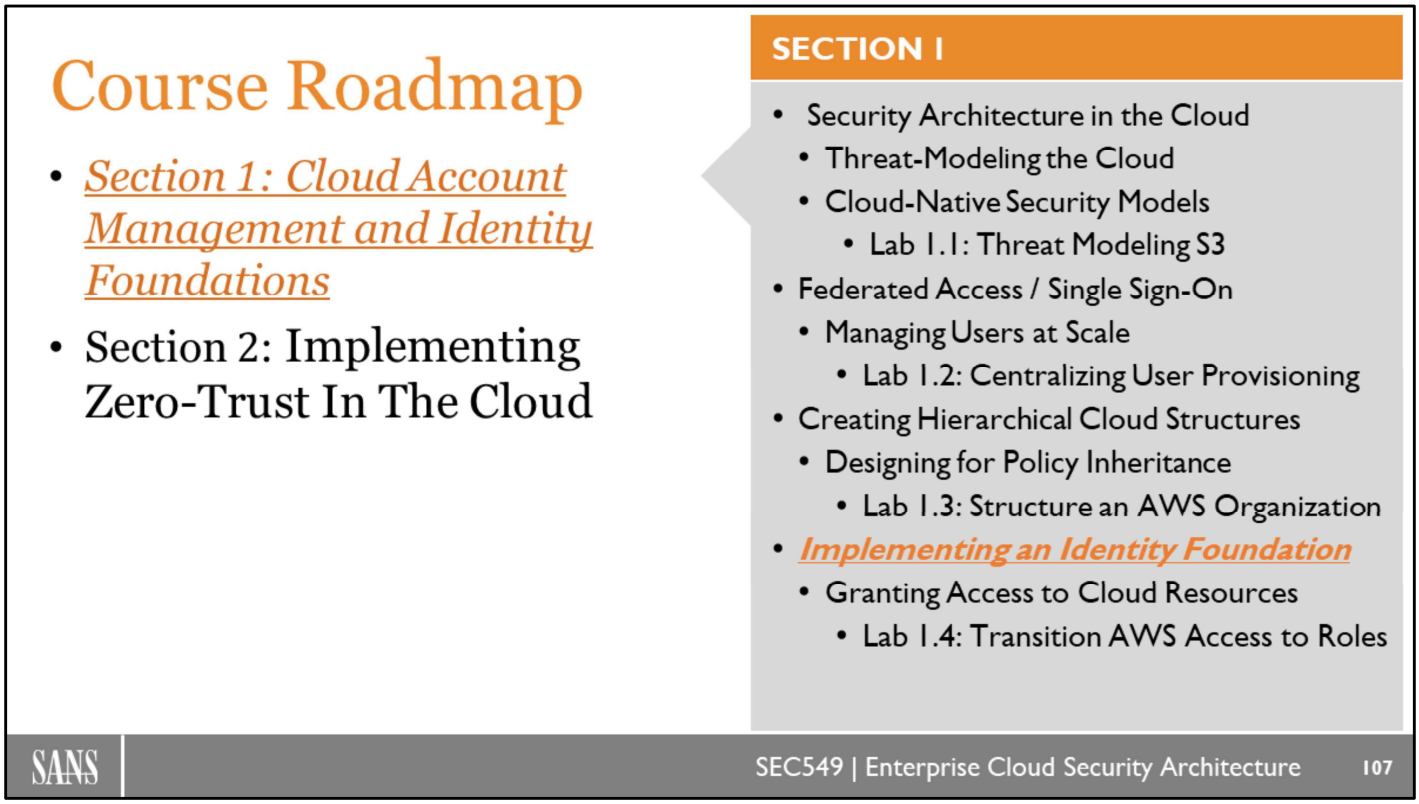
In addition to a handful of ubiquitous SCPs to apply at the Root, I've also depicted the default SCP which allows all actions.

Using an SCP, we've required all S3 objects be encrypted with specified KMS keys at the Infrastructure OU, Transitional OU, Workload OU, and Security OU. Only at the Sandbox OU have we omitted applying an SCP. Perhaps Sandbox Accounts will not require as prescriptive security controls, since we have assurances that only test-related or mock data will reside in these accounts.

We are requiring the Workloads OU via SCP to use only VPCs hosted from our Networking Accounts. Within the Infrastructure OU, Security OU, and Workloads OU, we've prevented any Access Keys or AWS IAM Users from being created. This SCP was omitted from the Sandbox OU, Transitional OU, and Exceptions OU. The Transitional OU and accounts below it are still relying on older access patterns and cannot conform to this restriction yet.

No Sandbox OU-specific SCPs are applied. This OU generally holds time-boxed accounts which only live 30-60 days and are considered playgrounds used in early phase testing.

No SCPs are applied to the Exceptions OU directly. In general, the few accounts which will be housed in this container will need bespoken SCPs, if any. You will often find yourself applying policy to the accounts directly rather than on the OU.



This is a foundational module where we cover the basic building blocks that make up cloud identity. Students will learn about the unique characteristics of cloud native users in all three major clouds, where they are housed, and how they are granted permissions. They will come away being about to articulate how Users/Roles/Groups are granted access to resources via IAM with a special emphasis on the AWS Role. We will expand upon the use of Roles as the gateway for access to resources is covered along with best practices for centrally managing Roles.

Takeaways

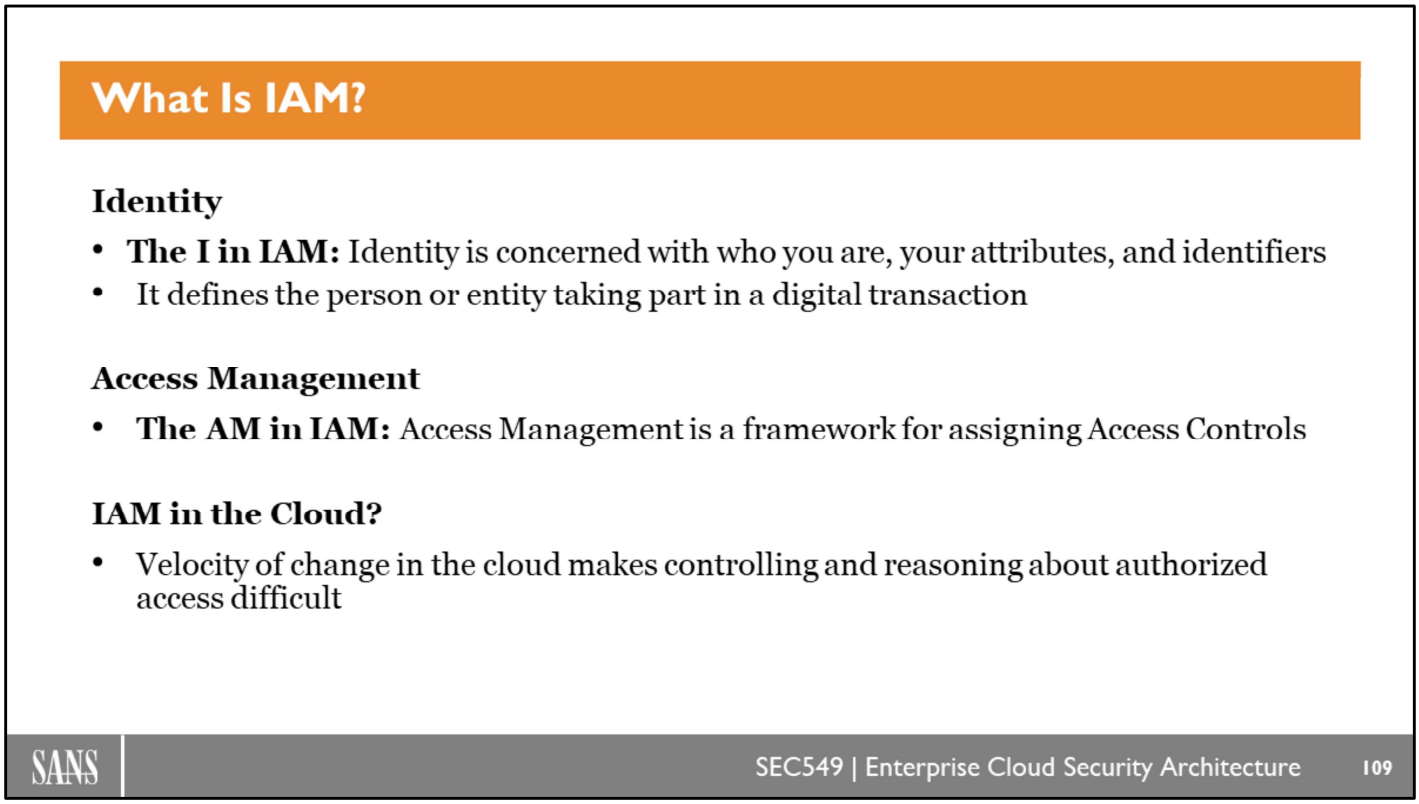
In this foundational module, you will learn the building blocks that comprise cloud identity.

You will study the unique characteristics of cloud native users in all three major clouds and where they "live".

You will be able to articulate how a User, Role, or Group is granted access to resources via IAM with a particular emphasis on AWS.

We will expand on the use of Roles as the gateway for access to resources and discuss best practices for centrally-managed Roles.

This page intentionally left blank.



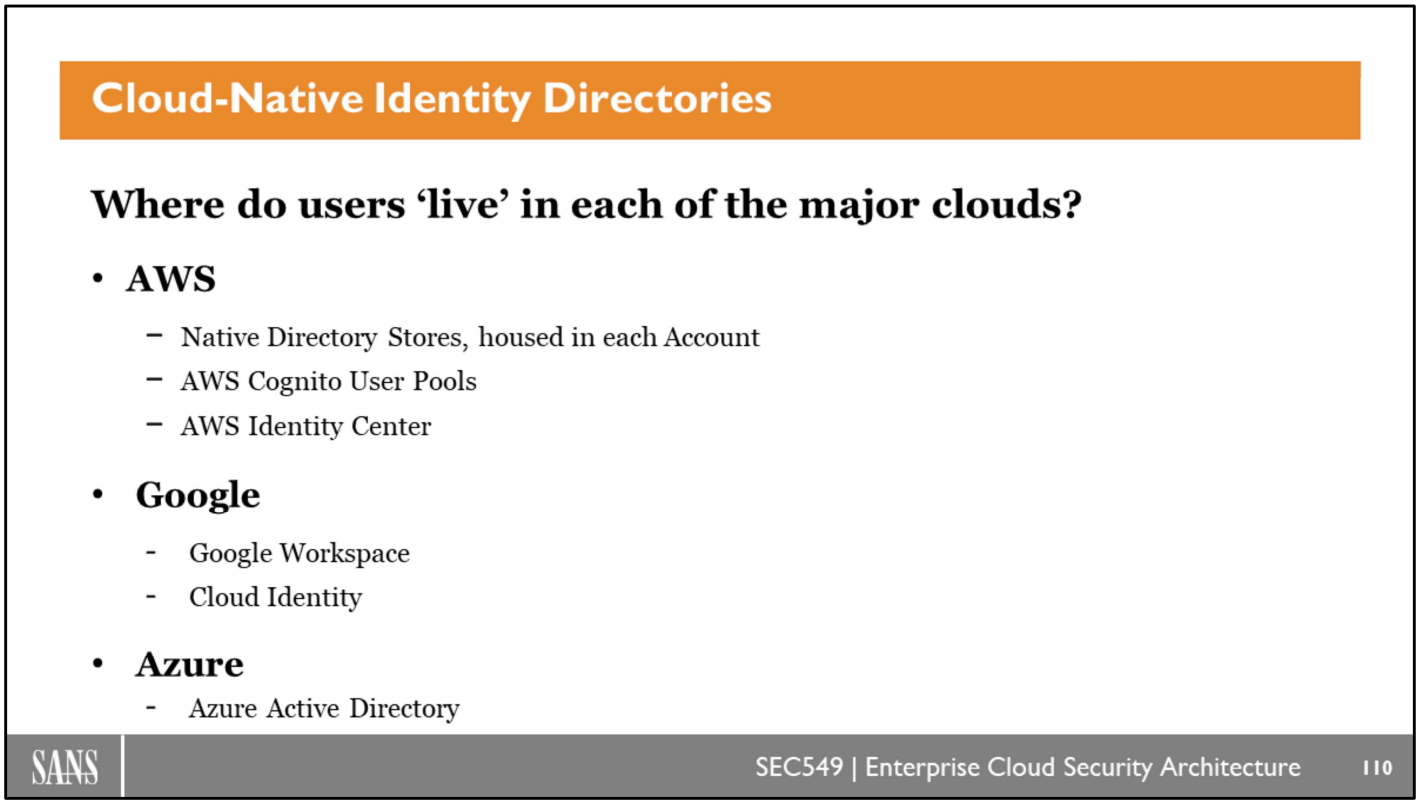
Identity and Access Management (IAM)
An Identity and Access Management system should describe WHO has access to WHAT. When access is judiciously applied to identities, the impact of a compromised identity is reduced. This is the cornerstone of the principal of least privilege, granting the minimal set of permissions required, as every person, device, application, etc., is considered a potential threat to the enterprise.

What can be an Identity? End Users, Machine Accounts, Devices, Resources. Anything that must require the assignment of permissions is an Identity. An Identity in a computer system is simply the assertion of said Identity, where that assertion is proven with credentials; passwords, tokens, or device heuristics.

An IAM program in the Enterprise is often a capability managed under the broader Security Organization or it can be the responsibility of a Technology-focused Organization alongside the management of networks and on-premises infrastructure. At its core, any identity and access management (IAM) program should be constructed to solve a business problem. Components of such programs can include:

- Self-Service Access Request Catalogs
- Group and Role Management
- Entitlement Management
- Login Portals
- MFA Options
- Password Reset Functions
- Credential Rotation
- Password Complexity Requirements

In the cloud, many of these IAM program components are provided as Cloud-Native Services by the cloud providers. When migrating workloads to the cloud, these IAM components can simply be another cloud service you consume, or you may rely on your existing IAM systems as the sources of truth, federating identity and passing attributes to your cloud systems.



Comparing Identity Directories Among the Different CSPs
While each Big-Three Cloud Service Provider has identities that can be assigned permissions, comparing where they 'live' or are housed is sometimes like comparing apples and oranges. Each has their own approach to Users and where they are managed.

GCP
GCP has no real concept of a User. Users of this platform reside in two other Identity-as-a-Service (IDaaS) Providers offered by Google: Google Workspace or Google Cloud Identity. *Google Workspace,* in addition to providing a Cloud Identity Directory, supplies users with Gmail accounts, Google Calendar, and features such as Google Drive and Meet. It offers SSO and 2-step verification features.

Google Cloud Identity is more expansive, offering access across an entire domain. Devices can be more securely managed; admin roles and privileges can be assigned and managed; security features can be fine-tuned; user provisioning can be provisioned across cloud apps.

Google Cloud Identity and Google Workspace involve an Identity management approach that is inherently not designed for on-premises Systems.

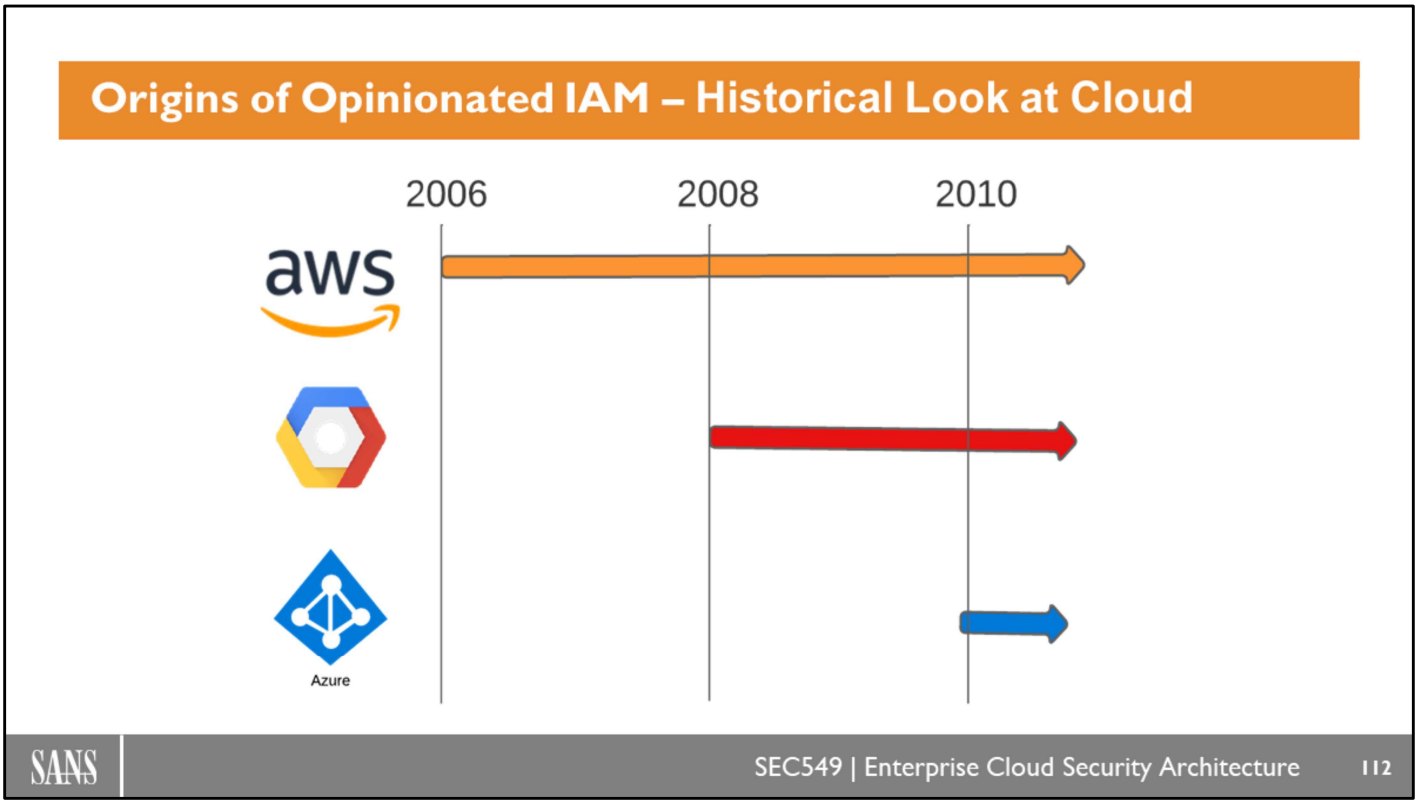
AWS
AWS has a built-in User Directory. AWS User is a resource in an AWS Account. Each account has the responsibility for maintaining that directory for their own organizational account.

Azure
Azure offers their own Identity-as-a-Service (IDaaS) Directory called *Azure Active Directory*. AAD operates similarly to GCP with a separate product that has its own attack surface to be concerned with.

Common Characteristics of the Big-Three IaaS Directories include:

- Federation capabilities via SAML and OIDC
- Ability to configure SSO for users and federated applications
- Enforcement of MFA
- User Lifecycle Management
- Programmatic Management

The choice of cloud-based identity services is less important than the concept that the choice should be the *primary centralized* and *authoritative source* used for managing user identities and their authentication.



Origins of Opinionated IAM

It can be helpful to reflect on the origin stories of each of the cloud providers to help intuit why they have different approaches to Identity.

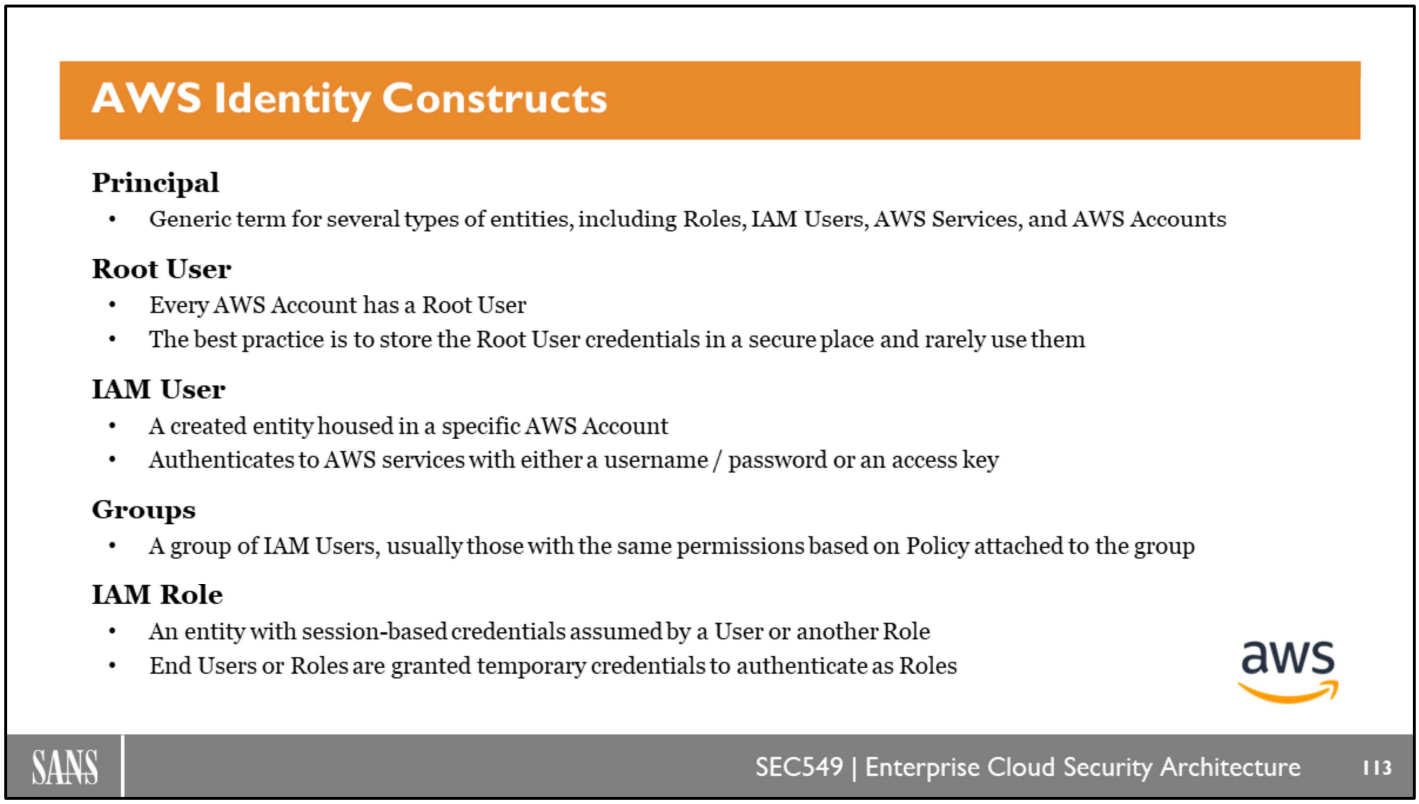
AWS began as part of Amazon and its monolithic online retail stature. The platform began to solve the company's own internal needs and eventually turned the internal tech platforms into consumer-facing products. This might be why, in AWS, the account was and still is the highest-level resource in AWS. Users in AWS are natively housed in the account and are assigned permissions to resources using identity-based policies. This means the policies are attached to the User. Answering the question, 'What does Bob have access to' is as simple as reading the policy attached to his User or Group.

Microsoft's Azure and Google's GCP are both Cloud Platforms, business units of a larger company that had a big footprint in B2B software and services prior their expansion into cloud service offerings. In the case of Microsoft, the company was already dominating enterprise software with Windows when, in 2001, it evolved its business to offer a Software-as-a-Service model for its Microsoft Office Products, called Office 365. In 2008, Microsoft unveiled its Identity-As-A-Service Product, Azure Active Directory.

Google has a longstanding image as primarily an ads business with their product, Google Ads, as the dominant player in search engines. However, Google's reach includes a suite of tools for small businesses called G Suite, launched in 2006 and recently re-branded as Google Workspace.

Azure Active Directory (AAD) and Google Workspace are both cloud-based identity directory products. As such, when Microsoft and Google launched their respective Cloud Service Offerings, neither included User Directories.

Unlike AWS, neither Azure nor GCP house identities in their cloud platform. Naturally, they are housed in their respective Identity-As-A-Service Products. This type of Identity architecture has led to a resource-based permission assignment. When resource-based policies are assigned, it's easier to describe, "Who has access to this object", rather than, "What access does Bob have."



AWS IAM Definitions

Principal
This is a generic term for several entities in AWS. A Principal can be a user, role, service principal, group or account.

Root User
The first AWS user is an entity called the "root user". It's a special kind of user in that it inherently has access to all services and resources in its AWS account. The best practice is to use the credentials for the Root User once when initially opening the account, enable MFA, and lock away the credentials, using them for the barest minimum of account-related tasks. [1]

Tasks Requiring the Root User
- Changing certain high-level AWS Account settings
- Restoring administrator IAM User permissions
- Activating IAM access to billing and cost management console
- Closing the account[2]
- Configuring MFA Delete on an S3 Bucket
- Editing or deleting an S3 Bucket Policy restricted by an invalid VPC IP
- Signing up for GovCloud

IAM User
Every other user in an AWS Account is called an IAM User. An IAM User is an entity that resides in the native directory in an AWS Account. IAM Users can sign into the ASW Console to interact with AWS or make programmatic requests (using the AWS CLI or SDK).

IAM Groups
User Groups are collections of IAM Users and can be given the same permissions. Similar to the IAM User, IAM Groups are resources that can reside in the native directory in any AWS Account.

IAM Role
An AWS Role is a transient identity often scoped around a job or task. Roles cannot be assigned usernames or passwords, instead they are assumed by another entity who needs to exercise the permissions attached to it.

References:
[1]: https://sec549.com/id52
[2]: https://sec549.com/id53

Root User Management
You won't find the Root User listed as a user resource in the IAM console nor will you be able to inspect permissions attached to it, but this User is all powerful. It inherently has full admin access to all resources in the AWS Account it created.

Every AWS Account has a Root User so how do you manage the Root Users when you have hundreds or thousands of AWS Accounts?

Root Users Tied to Manually Created AWS Accounts
During the manual account creation process, the email address used to open the account is used as the Root username and a password for this account is set. Once the email address is verified and the AWS Account is open, its strongly advisable to configure MFA for the Root User[1].

Root Users Tied to Accounts Created by AWS Organizations
Even AWS Accounts created with AWS Organizations Service have a Root User. During the process of Account creation, you are prompted for an email, which becomes the Root User for that member Account. The Organizations Service will set an initial password for the created member Account Root User that can't be retrieved. In order to login as the Root User for an AWS Account that has been created by AWS Organizations, you must follow the 'forgot my password' flow[2] from the AWS Login Console Page. A password reset link is sent to the email address associated with the Root User.

Root User Email Addresses
When an AWS Account is opened, whether manually or provisioned with AWS Organizations, you are prompted to supply an email address. This email address becomes the username for the Root User and its root of trust. Avoiding personal email addresses or unauthorized email addresses becoming Root Users is paramount. Access to the Root User email inbox allows for the retrieval of temporary passwords during the password reset process, even if MFA is enabled on the Root User.

One operational challenge is that no two AWS Accounts can be tied to the same email address. Once an email address is used as the Root user, it cannot be used again for another account, even if that account is closed.

Managing Root Users

So how should you manage potentially thousands of Root Users tied to child accounts created by your organization? The short answer is, you shouldn't. The password for these accounts is automatically set by AWS and should not be retrieved. Administrative management of this account should be performed from the Management Account that created it. The Root User password should never be retrieved or used. The only Root User which you should need to manage is the one associated with your Management Account.

Management Account - Root User MFA

Configuring MFA for your Root User is only necessary for the one associated with your Management Account.

Options for MFA Devices include a virtual device used for generating a Time-Based One-Time Passcode (TOTP), a Yubikey, or other U2F Compliant Device or a Gemalto Hardware Token. Avoiding MFA tokens residing on an employee's personal devices is a critical task. A common strategy that organizations use is to have a single hardware MFA device authorized to use for Root User credentials. This device should be stored in a safe or in an accessible yet monitored area for the operations or security teams, considering the need for both availability of the Root User credentials and the need to restrict access.

GCP Identity Constructs

Managed in Google Workspace or Cloud Identity

- **Identity –** an identity can be **Users, Groups, and Domains**
- **Group –** a collection of Users managed in Google Workspace or Cloud Identity

Managed on Google Cloud Platform (GCP)

- **Identity –** an identity can be **Service Accounts or Identity Namespaces**
- **Role –** a collection of permissions
  - primitive/basic
  - predefined
  - Custom
- **Policy –** a document stating which Principals have access to what Resources
  - Policies are bound to resources to enforce access control
- **Org Policy Constraints**
  - Boolean and List Constraints

SANS | SEC549 | Enterprise Cloud Security Architecture 117

GCP Cloud IAM
With Google Cloud IAM, you manage access control by defining who (identity) has what access (role) for which resource. Resources are organized in a hierarchical manner and permissions are inherited.[1]

Identity – a Google umbrella term for any Identity that can be a member in a policy. An Identity can be a Google Account (when the entity is an end user), a service account (when the entity is an app or virtual machine), a Google group, or a domain.

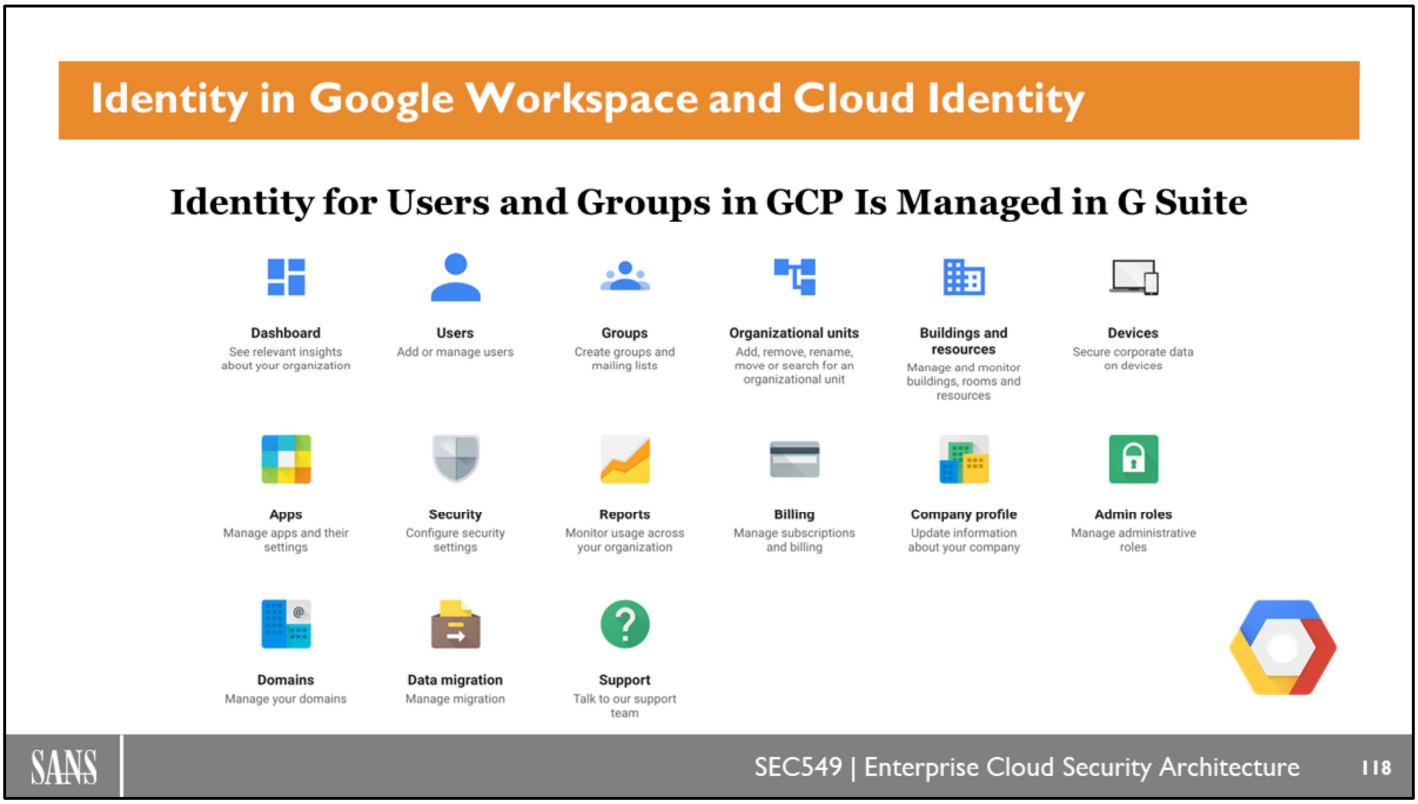
Group – a named collection of end users. It can be a convenient method of applying one or more Roles to a collection of Users. Identities added to a group inherit the policies linked to the group. Groups are not assigned credentials of their own.

Role – a collection of permissions. It defines what operations are allowed on a resource. A role can be assigned to a Principal and once done, all permissions connected to the role go with it. You can't randomly give permissions without using a role. Roles can be primitive, predefined, or custom.

✓ **Basic or primitive roles** – Owner, Editor, and Viewer
✓ **Predefined roles** – allow finer-grained access to a resource (e.g., Pub/Sub Publisher)
✓ **Custom role** – a method of tailoring permissions when predefined roles are not satisfactory.

Policy – this is a *document*. It helps you define and enforce which members have specific types of access (role) to what resources. In GCP, policies are attached to resources. A Policy binds one or more Identities to a Role. When you want to define who (identity) has what type of access (role) on a resource, you create a policy and attach it to the resource.

Reference:
[1]: https://sec549.com/id54



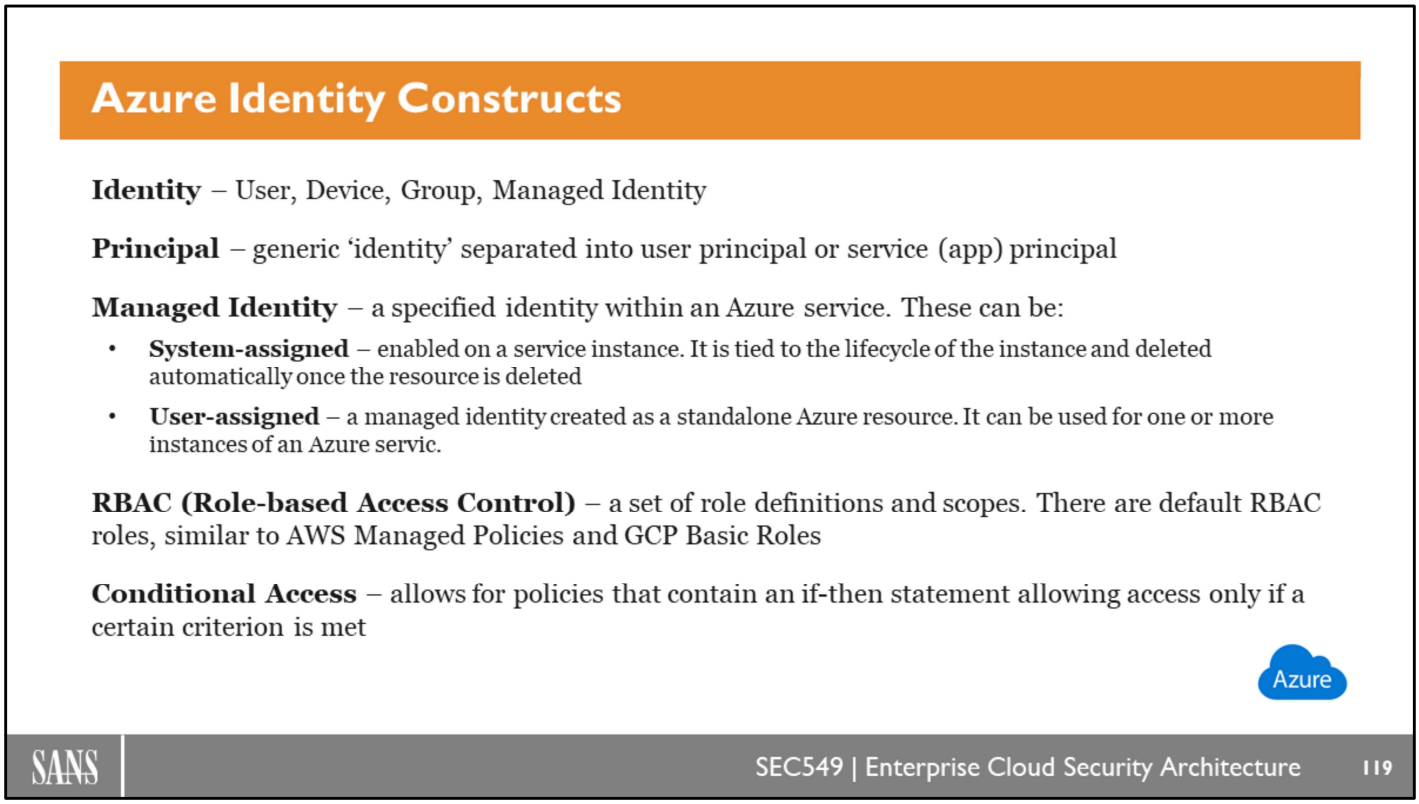
Identity in Google Workspace and Cloud Identity

Google Workspace (nee G Suite) is a collection of product offerings from Google that include their popular Gmail, Hangouts, Google Drive, and Google Docs. It also contains a Marketplace to integrate with 3rd-party apps like Zoom.[1] The Google Cloud Platform is considered a part of Google Workspace. If your Organization has a presence in GCP, its possible they also have a footprint in Workspace. All Workspace products, including GCP, use either the native Workspace directory for identity management or Cloud Identity which is available only for GCP. Workspace deals with authentication ("who you are"), while GCP deals with authorization ("what you can do").

A user who needs access to your GCP resources will need an Account in the Identity-as-a-Service Product Cloud Identity (IDaaS) or Google Workspace proper. Users managed in Google Workspace may inherit access to other Google services like Drive, Calendar, Marketplace, and Google AppScripts, depending on what type of directory your organization has enabled (classic Google Workspace or a Cloud Identity directory)[2,3,4]. Access to those services and apps can be turned off in the G Suite Admin Console using Access Groups.[5]

As one would expect with all modern SaaS tools, you can configure end user authentication to Google Workspace to federate against your Identity Provider (IdP) of choice.[6] When authentication is federated, if your IdP allows supports MFA, you can challenge your Users for a second factor at the time of authentication. Because MFA is a part of authentication and Google Workspace manages all authentication, it cannot be used as a condition or security mechanism in GCP in the same way a second factor is often used in AWS.

References:
[1]: https://sec549.com/id55
[2] https://sec549.com/id56
[3]: https://sec549.com/id57
[4]: https://sec549.com/id58
[5]: https://sec549.com/id59
[6]: https://sec549.com/id60



Azure Active Directory

Microsoft first introduced its directory service called Active Directory (AD) in 1999 for Windows 2000. It represented a way for Windows domain networks to utilize domain controllers to keep track of users and devices, manage credentials, and define a user's access rights.

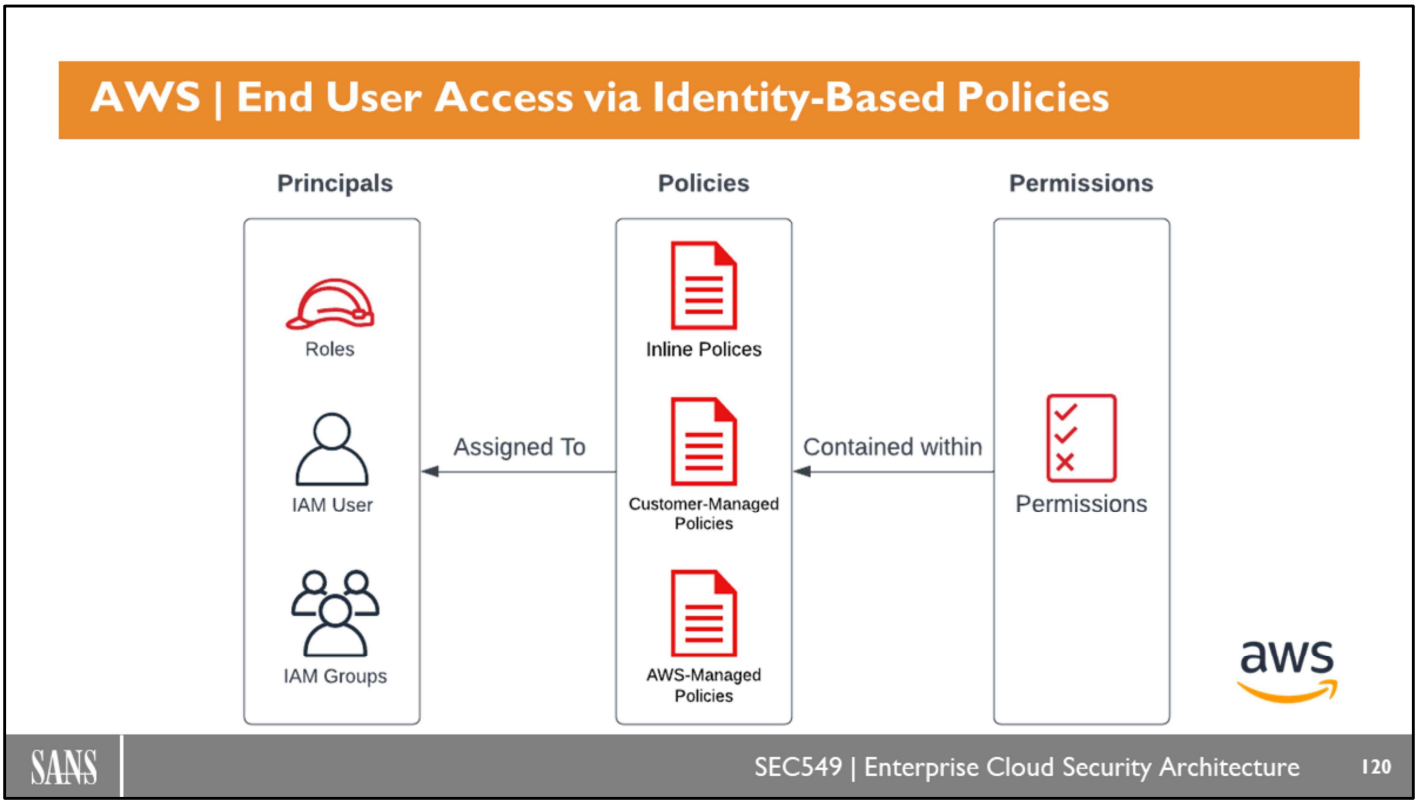
AD has undergone several different versions (and enhancements) since then; the term 'Azure Active Directory' is an unfortunate victim of the tendency to rename things with similar-sounding names that are not simply the 2.0 version of the older term. In other words, Azure Active Directory (Azure AD) is not the cloud version of AD.

There are similarities in what AD and Azure AD are designed to do:

1. Both provision users in some fashion
2. Both provision external entities in some fashion
3. Both offer varying levels of entitlement to Admin Users and Regular Users
4. Both offer credential management
5. Both work within a Windows desktop environment

Where do they differ the most? In capabilities unique to Azure AD.

1. Azure AD offers a much larger suite of service options to help tweak IAM in a cloud-based world (e.g., Azure AD Connect, Azure AD B2B, Azure AD Domain Services)
2. Azure AD offers ways to tweak entitlement enhancement for certain users or groups, using workflow and time-based criteria
3. Azure AD offers built-in roles with its Azure AD RBAC system; role management is easier to fine-tune
4. Azure AD uses intelligent password protection, MFA, and password-less technologies
5. Azure AD inherently supports SaaS apps supporting SAML OAuth2, and WS-FED
6. Azure AD natively supports mobile devices
7. Azure AD supports the ability of Linux/Unix VMs to use managed identities



Granting Access to End Users using Identity-based Policies
Assigning end users' permissions to cloud resources in AWS is characterized by the relationship of the three P's: Principals, Policies, and Permissions

Principals – The following Principals can be assigned identity-based policies: Users, Groups, Roles and Service Principals. They all can have identity-based policies attached to them and therefore, can be granted access to resources via this mechanism.

Policies – Collections of permissions. A policy document can contain as few a single permission or as many as will fit in the maximum size of the policy document, which is 2,048 characters[1] . Policies used in Identity-Based permission granting come in three forms: AWS managed, Customer Managed, and Inline.

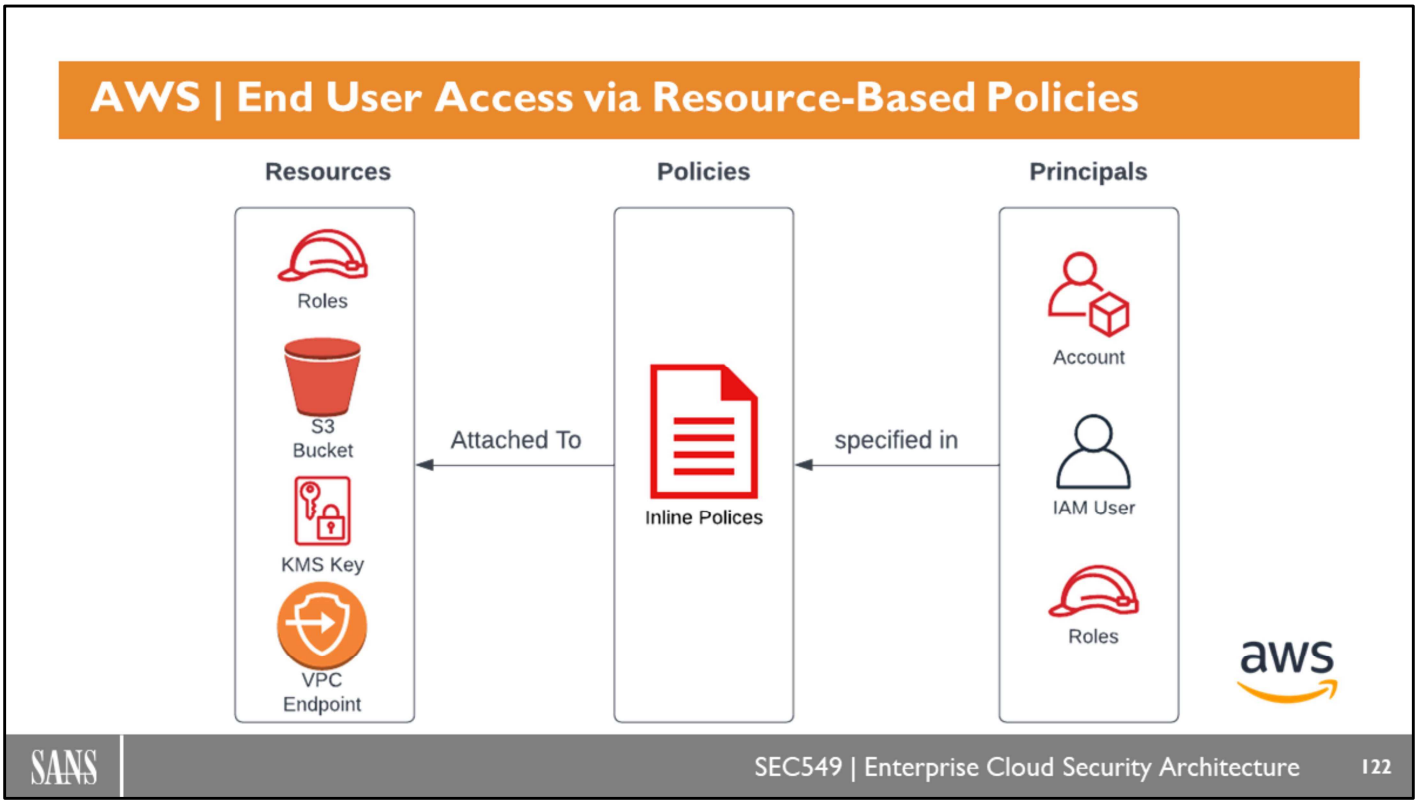
AWS Managed policies are curated by AWS and represent common collections of permissions based on job functions. These policies tend to represent coarse-grained permission sets, are best used for testing and development phases, and not recommended for use in a Production setting.[2]

Customer-Managed Policies are also collections of permissions, just as are AWS managed policies but these are created and managed by the AWS customer. Ideally, these policies are more tightly scoped to the specific collection of permissions required and are further restrained by the administrator by defining the resource field in the policy document.

Inline policies are do not have versioning and cannot be managed. They are attached directly to Principals rather than living as a manageable policy document.

Permissions – Specified as a compound string which includes both the AWS Service and the action which can be performed on the service. The example 's3:GetObject' permission would allow the grantee to get objects from the S3 service.

References:
[1]: https://sec549.com/id61
[2]: https://sec549.com/id62



Granting Access to Resources using Resource-based Policies
Resource-based policies are the converse of an identity-based policy. Instead of describing the permissions a Principal has, a resource-based policy enumerates the Principals which can act on a particular resource. Resource-based policies do not give you a complete picture of the permissions any given Principal has.

Principals – Valid Principals in resource-based policies include Users, AWS Accounts, Roles, and Service Principals. Resource-based policies must include the Principal field, specifying who has been allowed or denied permissions on any given resource. The Principal field can be leveraged to grant cross-account permissions and allow access from external entities.

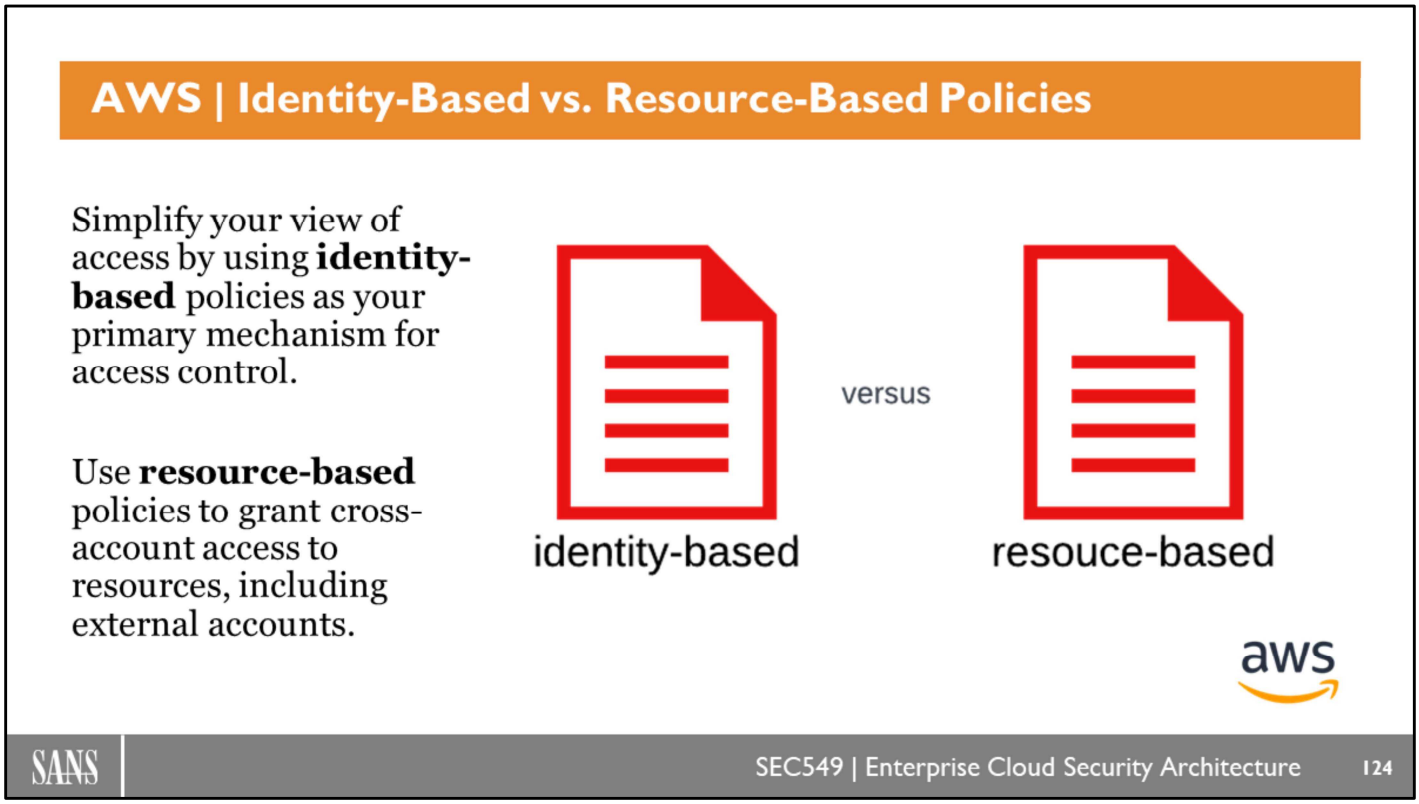
Policies – Collections of permissions. A policy document can contain as few a single permission or as many as will fit in the maximum size of the policy document, which is 2,048 characters[1] . Resource-Based policies can only be attached to resources as Inline policies and are not otherwise managed or versioned.

Resources – Any AWS resource that is compatible with resource-based policies. Policies are attached to these resources allowing the calling principal to invoke the specified actions.

Which AWS Resources support Resource-Based Policies? The list is extensive[1] but there are a handful of the most popular services:

- S3 Buckets
- KMS Keys
- VPC Endpoints
- SQS Queues
- IAM Roles
- Secrets Manager
- Lambda Functions
- Amazon Lex Bots

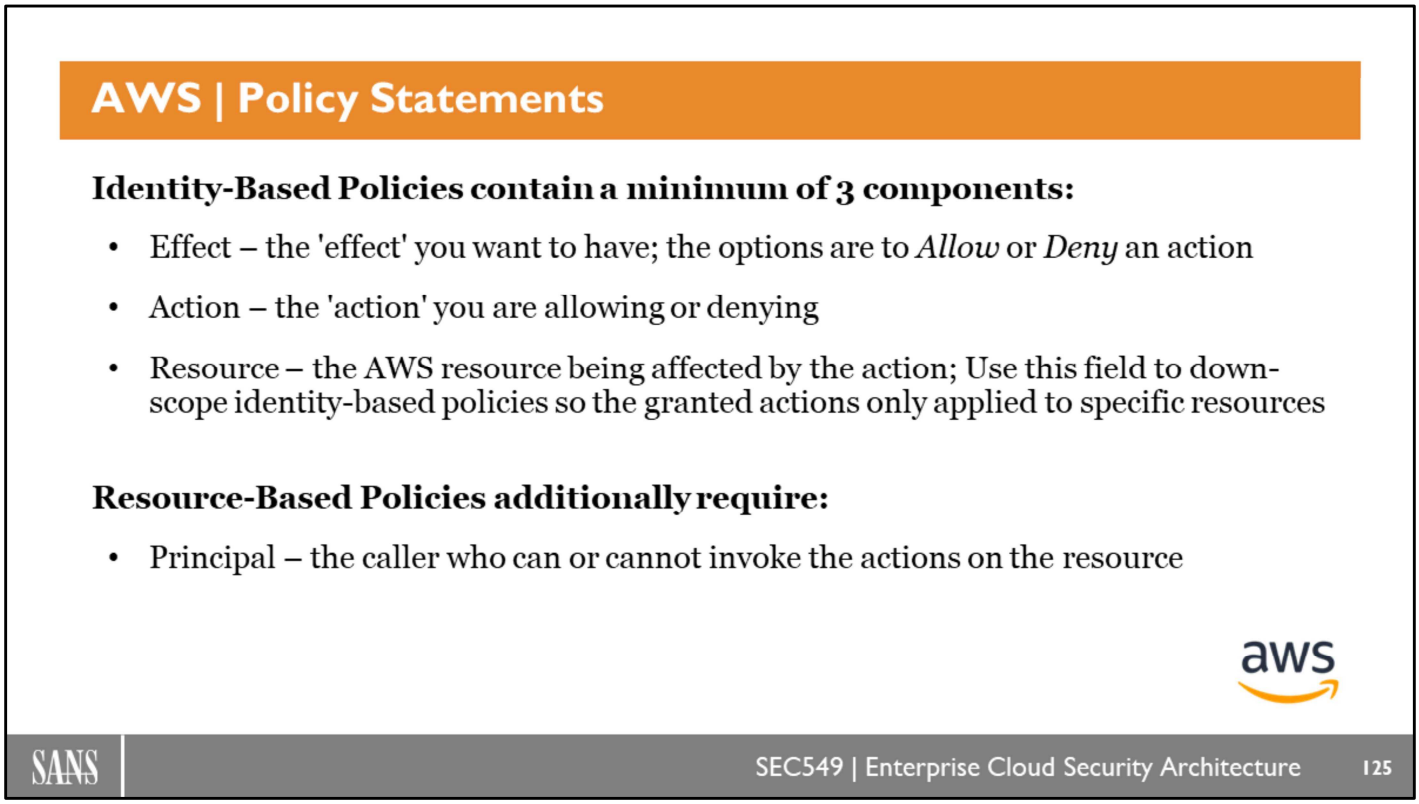
Reference:
[1]: https://sec549.com/id63



Comparing Identity-based and Resource-based Policies
Since AWS uses a combination of identity-based applied policies and resource-based applied policies, neither gives a complete picture of what actions a Principal is allowed to perform. Even so, sticking to one or the other can help simplify the lens into permissions granted to end users.

Advantages of using Identity-Based Policies
- Maintaining access from the identity perspective tends to be more familiar and easier to reason about.
- Answering the question "What does XYZ end user have access to" is simpler with identity-based policies
- Policies attached to users can be managed (either customer-managed or AWS-managed), versioned, and maintained as a policy document independent of its attachment.
- Enumerating the totality of the access needed for any given resource in a single document can often lead to a bloated policy document that challenges the 2,048-character size limit.

When might you use a resource-based policy?
When granting access to an external principal, often the most straightforward way to accomplish this is to specify the external account ID in the principal field of the resource-based policy, allowing any principal in the external account the ability to access your resource.



Anatomy of a Policy
A policy is a set of permissions provided as a 'statement'. All statements need three mandatory components:

Effect – what effect do you want to have? There are two options (Allow or Deny). The default condition for any API is to deny an action, so an explicit *allow* is needed. An explicit *deny* will override an *allow*.

- *Example:* IAM users do not automatically have permission to use any resources or API actions. All requests will be denied unless overridden.

Action – what specific action is being allowed or denied? There can be dozens or even hundreds of actions contained within a policy statement.

- *Example:* The action *"ec2:CreateVpcEndpoint"* allows the grantee to create a new VPC Endpoint.

Resource – the specified resource affected by the chosen action. The wildcard (*) indicates that the statement applies to all resources. If an Amazon Resource Name (ARN) is specified, the policy *allow* or *deny* effect is restricted in scope to this resource.

- *Example:* Permissions to access a specific EC2 instance might be accomplished by referencing the following ARN in a policy statement: *arn:aws:ec2:us-east-1:4575734578134:instance/i-054dsfg34gdsfg38*[1]

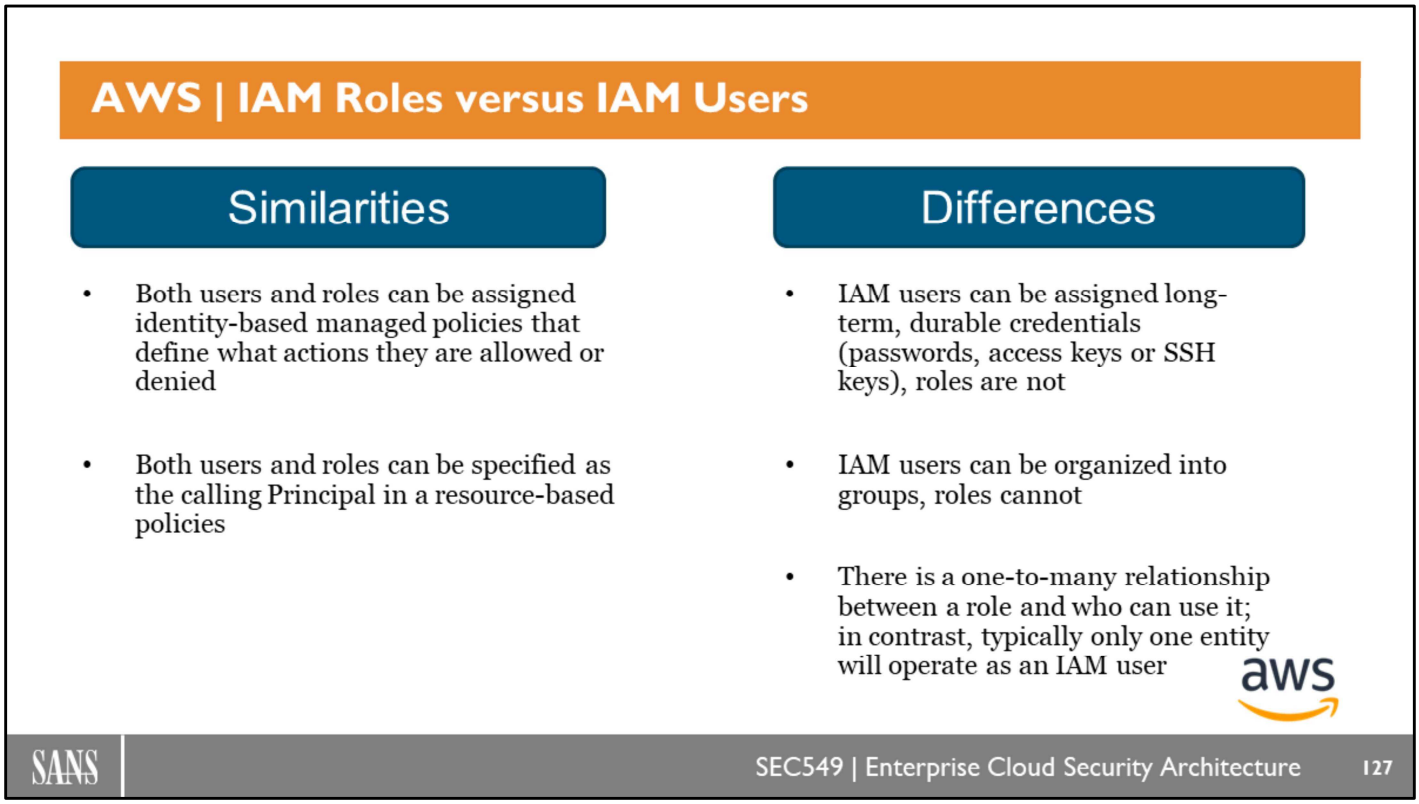
Principal – these are the identities which are granted or denied the action on the resource. The Principal field is only provided in resource-based policies. A Principal is specified as the Amazon Resource Name (ARN) of an IAM User, IAM Role, AWS Service Principal, or AWS Account.

- *Example*: A KMS Key will have a default key policy attached to it upon key creation. The following ARN might be specified to allow the originating account full access to the key: *arn:aws:iam::123456789012:root*

Condition[2] – Conditions are optional. Your policy does not need to be always in effect or under every 'condition'. By setting conditions, you can tweak a policy.

- *Example*: If you only want the policy to be valid when invoked from a particular IP address, you can add the statement: "Condition": { "IpAddressIfExists": {"aws:SourceIp" : ["xxx"]}

References:
[1]: https://sec549.com/id64
[2]: https://sec549.com/id65



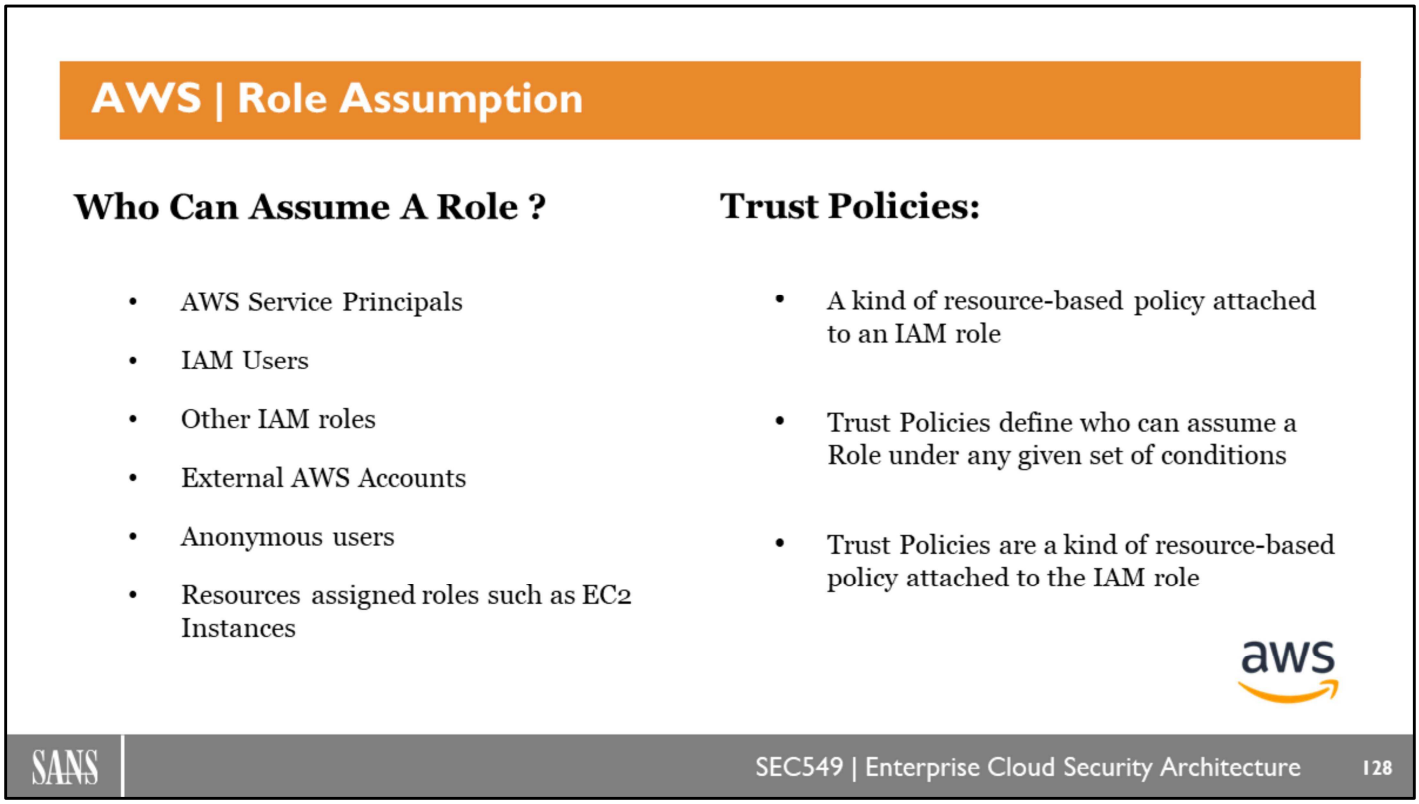
IAM Roles Versus IAM Users
Roles are used in AWS to allow other Principals to gain the ability to assume temporary permissions when needed to perform a specific task[1].

As we've seen, an AWS Role is another Principal like an IAM User. There are some distinct similarities between an IAM User and Role along with some clear differences.

Roles are often used by AWS to delegate access to users or services. Rather than assigning an end user all the permissions they might need, Roles can be created to represent different job functions or tasks an end user might perform. The permissions assigned to roles can be temporarily assumed by team members only when needed. This pattern keeps the policies directly attached to the end user relatively minimal.

Privilege Bracketing with IAM Roles & MFA
Imagine a scenario where members of cloud development teams are assigned IAM Users. On occasion, those Users may have to perform administrative tasks. Instead of assigning the IAM User administrative permissions directly, an IAM Role can be provided which is assigned the required admin policy. Team members can be allowed to use this administrative Role with the condition that MFA is provided. This pattern is referred to as *privilege bracketing*.

Reference:
[1]: https://sec549.com/id66



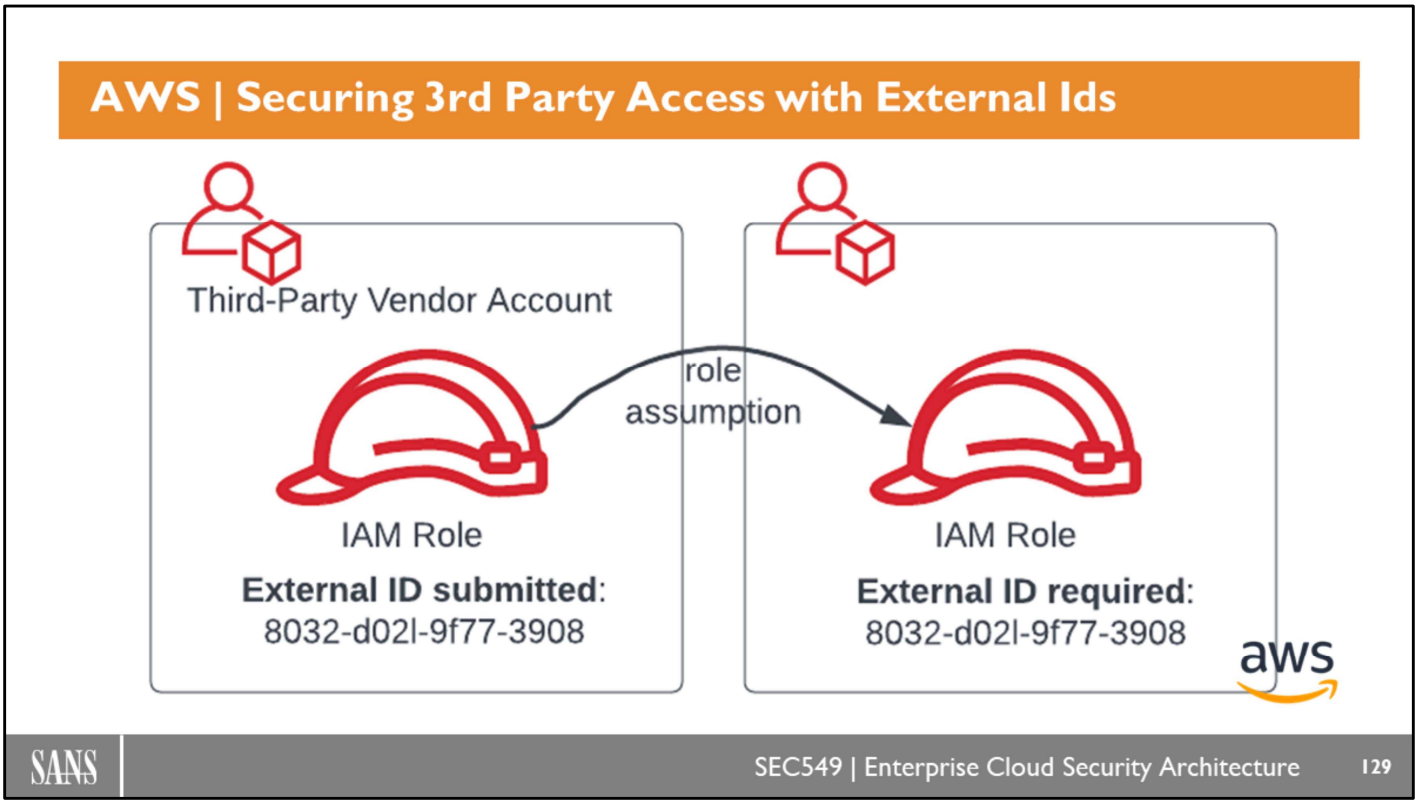
AWS Role Assumption

Roles are used in AWS to allow other Principals to assume temporary permissions when needed to perform a specific task. As a result, describing AWS Roles as 'hard hats' is a common analogy used to describe their usage patterns[1]. *Role Assumption* is the process by which an AWS Principal 'puts on' this hard hat. Who is allowed to put on the 'hard hats' is defined in the Role Trust Policy.

AWS APIs which allow the caller to Assume Roles:

- *assume-role (standard method):* An IAM User or IAM Role would use this API if their identity is already established with AWS. The assume-role API generates temporary session credentials for the targeted role.
- *assume-role-with-saml:* The method an external entity might use to authenticates to AWS with a SAML token. Upon authentication via SAML, temporary session credentials are generated for the targeted role.
- *assume-role-with-webIdentity*: The method an external entity might use to authenticates to AWS with the OIDC protocol. Upon authentication via OIDC, temporary session credentials are generated for the targeted role.

Reference:
[1]: https://sec549.com/id67



AWS | Securing 3rd Party Access With External Ids
IAM Roles are the mechanism to allow an external entity, such as a third-party vendor, access to your AWS Account(s) without needing to share long-term durable credentials (passwords or access keys). These third parties can be allowed to assume a role in your account and generate temporary session credentials just as any other internal Principal can be enabled to use a Role.

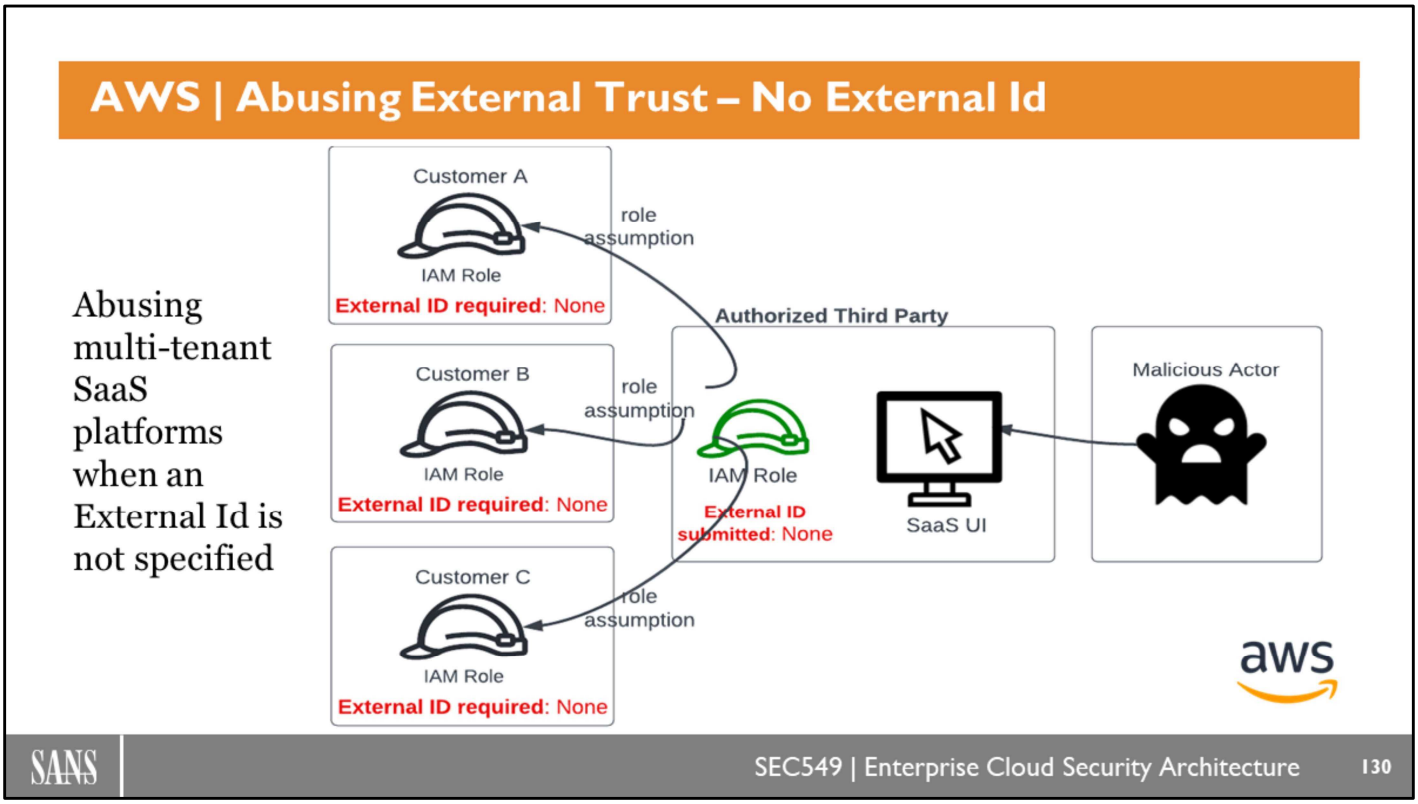
When allowing a third party to assume a Role, it is **strongly** recommended that you require the third-party caller to submit an external ID when assuming the Role.

What is an external Id?[1]
An external ID is used to mitigate the class of vulnerabilities broadly called "confused deputy" issues[2]. External IDs help to prevent a malicious actor from tricking the third party into unwittingly accessing your resources.

How to use an external Id
The external ID is specified as a condition in a trust policy, meaning you write an IAM Role Trust Policy which allows a principal to assume a Role as long as a particular string is submitted when the caller assumes the Role.

While the external ID is not stored as a secret, there are requirements around the generation of external Id to keep in mind. In the upcoming slides we will threat model the "confused deputy" vulnerability as it manifests in AWS role assumption by external parties in order to highlight how prescriptive use of the external id closes these risks.

References:
[1]: https://sec549.com/id68
[2]: https://sec549.com/id69

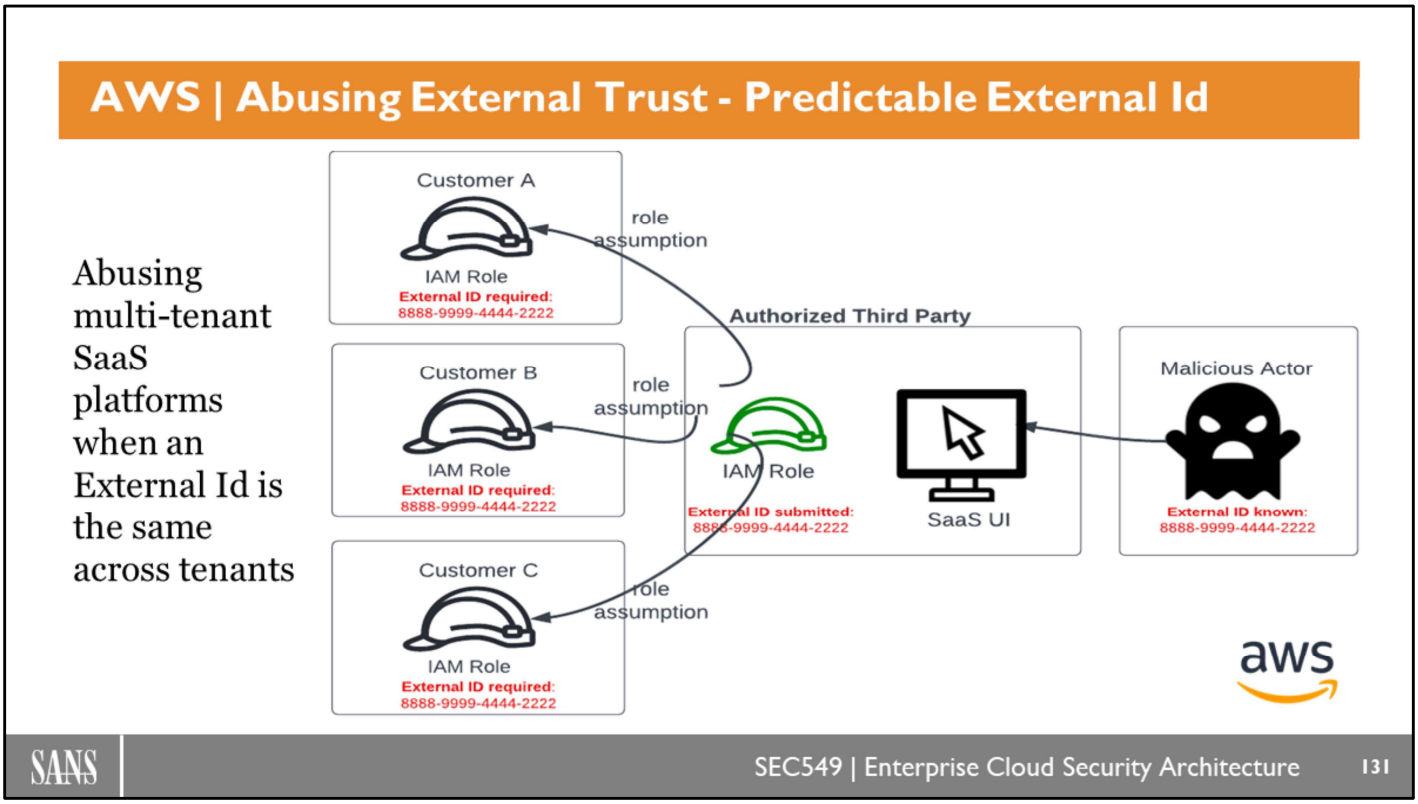


Threat Modeling 3rd Party Vendor Access
Role assumption is the best practice method for granting an external party access into an AWS account. As we saw in the previous slide, when allowing an external party to assume an AWS IAM Role, an External Id must be required to mitigate "confused deputy" vulnerabilities. In this slide, let's look at what could happen and how some malicious actors could abuse third-party vendors if an External Id is not required.

Vendor Portals as the Confused Deputy[1]
The vendor space is rife with offerings which commonly are brought on board to fill gaps in resource visibility and metrics collection, or serve as cloud infrastructure orchestrators, creating and deleting resources. As it relates to the "confused deputy", it is the vendors online web portals, leveraged to configure cross-account trust, which represent the "deputy" in this threat model discussion. In this abuse scenario, an attacker could leverage a vendor web portal to assume the preconfigured IAM Role of one of the vendors existing customers. Without requiring an External Id, anyone with knowledge of a customer's AWS account number and role name could leverage the vendor web portal to maliciously access the existing customers cloud account via the externally accessible IAM Role.

Configuring Third-Party Trust
Thanks to the work of researcher Kesten Broughton, a simple, open-source tool[2] is available which mimics the functionality of a common vendor web portal. Using this mock vendor web portal, you can put the strength of your external Id requirements to the test, vetting your own externally accessible IAM Roles for susceptibility to "confused deputy" attacks.

References:
[1]: https://sec549.com/id193
[2]: https://sec549.com/id194



Predictable External Ids

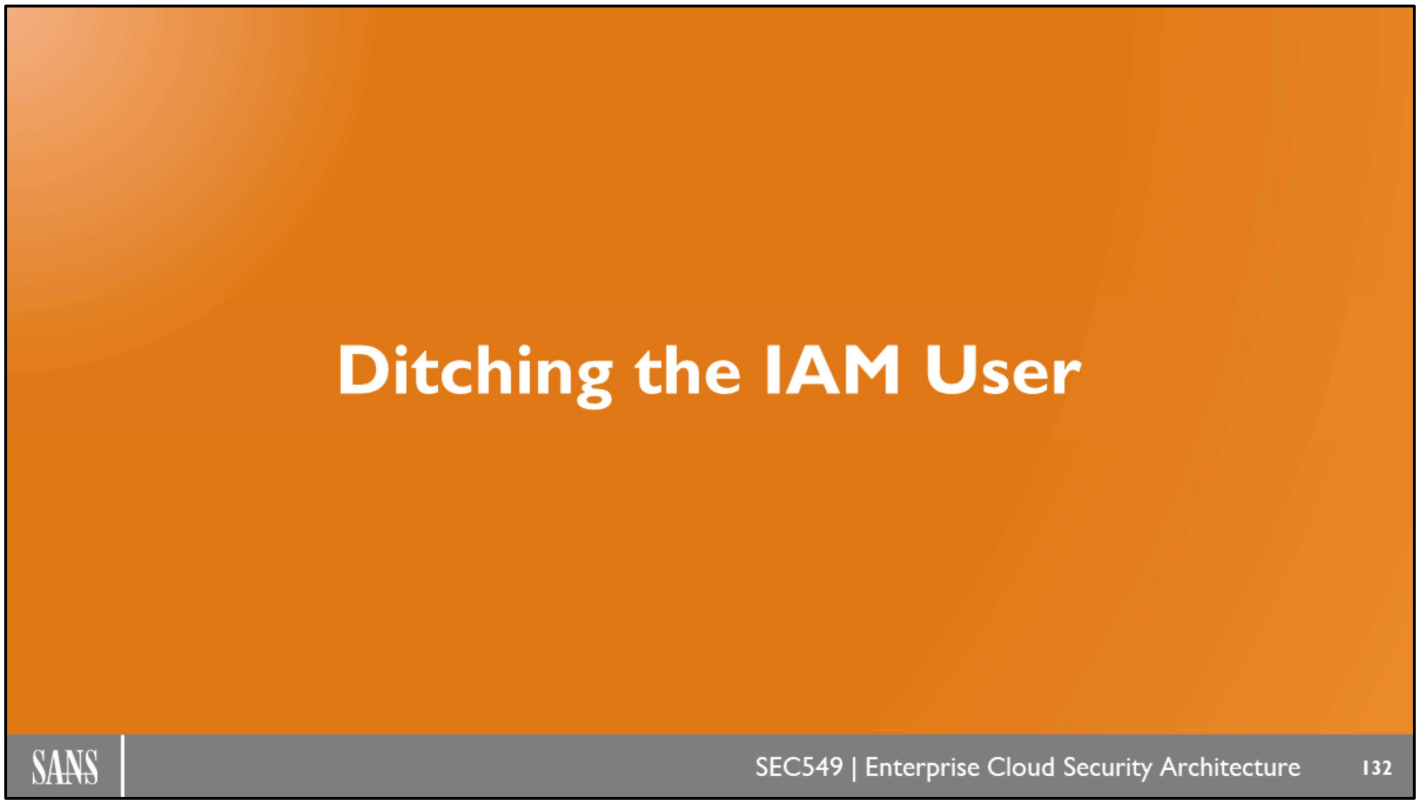
Simply requiring an external id for role assumption from external parties is not enough to fully mitigate the risk of external access from a 3rd party. If the external id is guessable or predictable, a malicious actor could leverage an available third-party web portal (the deputy) to assume a pre-configured customer IAM role, granting the attacker access to the customer account.

An external id would be considered insecure if it could be brute-forced as a result of being too short or not sufficiently random. If the external ids are predictable, perhaps they are always the name of the customer or are the same value across all customers.

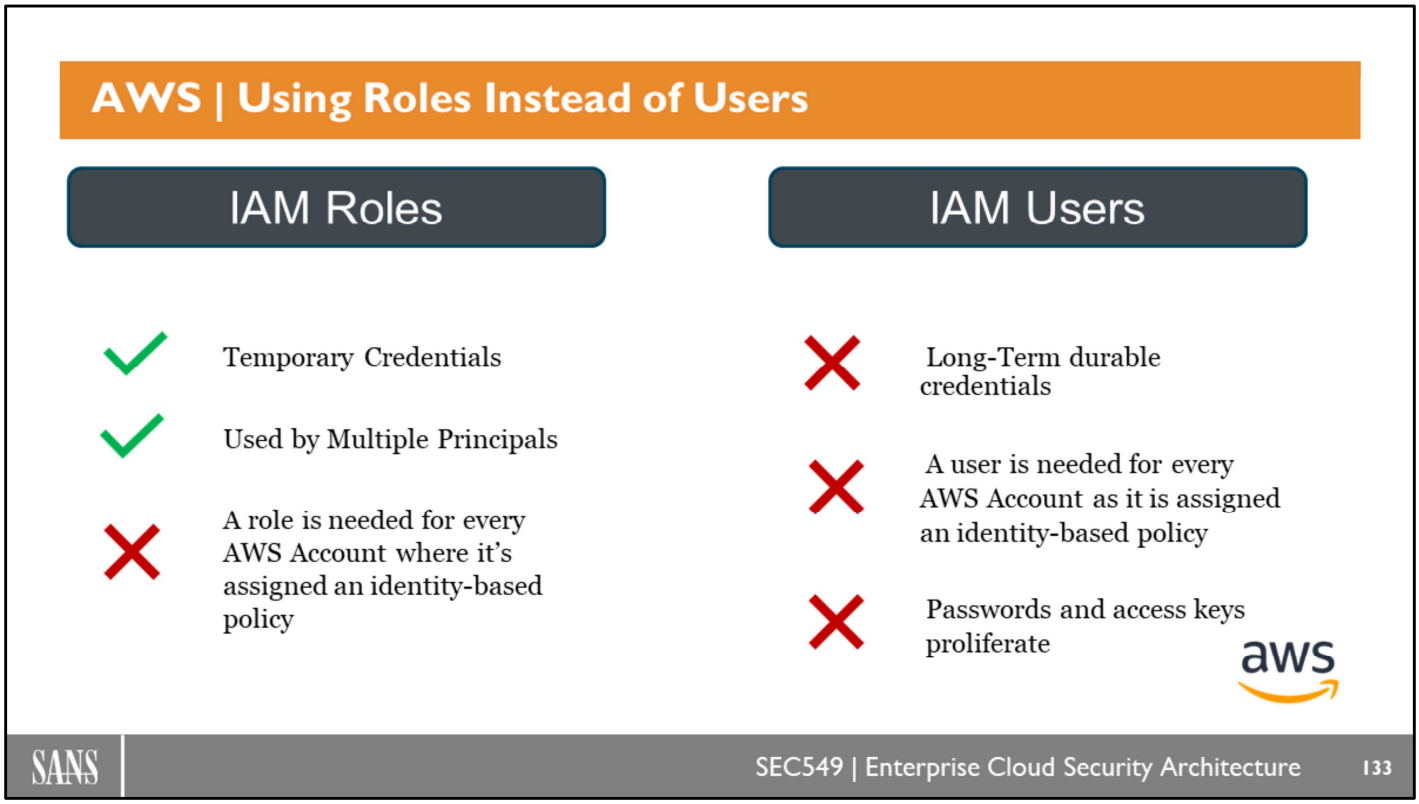
External Id Requirements

External Ids should be:

- Unique, per customer
- Random, un-guessable String
- Long in length (UUID4)
- Specified by the 3rd party, never defined by the customer.



This page intentionally left blank.

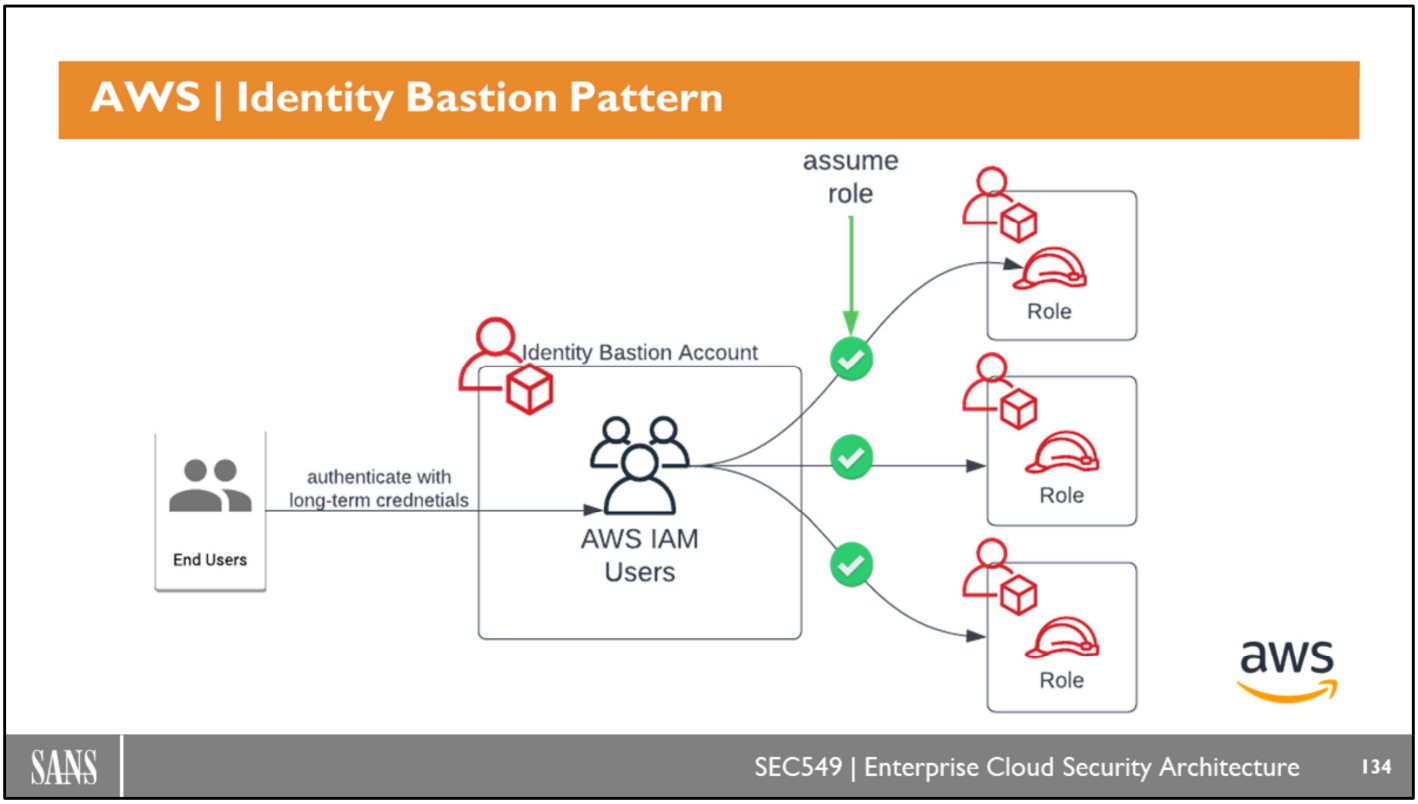


Using Roles instead of Users in AWS

If you find yourself overseeing an AWS environment involved in creating and housing IAM Users specific to every managed account, transitioning from IAM Users to IAM Roles should be your first step in simplifying and thus securing identity in AWS.

IAM Users come with a couple of disadvantages from a security and operational perspective. IAM Users are assigned long-term, durable credentials - passwords, access keys or both. They also are housed in the native user directory of every AWS Account. As an enterprise grows in their number of accounts, this often results in the same end users who must be created as multiple IAM Users with multiple passwords and access keys.

Roles, on the other hand, can be used and reused by multiple Principals. When an end user assumes a Role, they are generating temporary session credentials representing the authenticated state of the Role. There is no need to rotate the credentials used by IAM Roles.



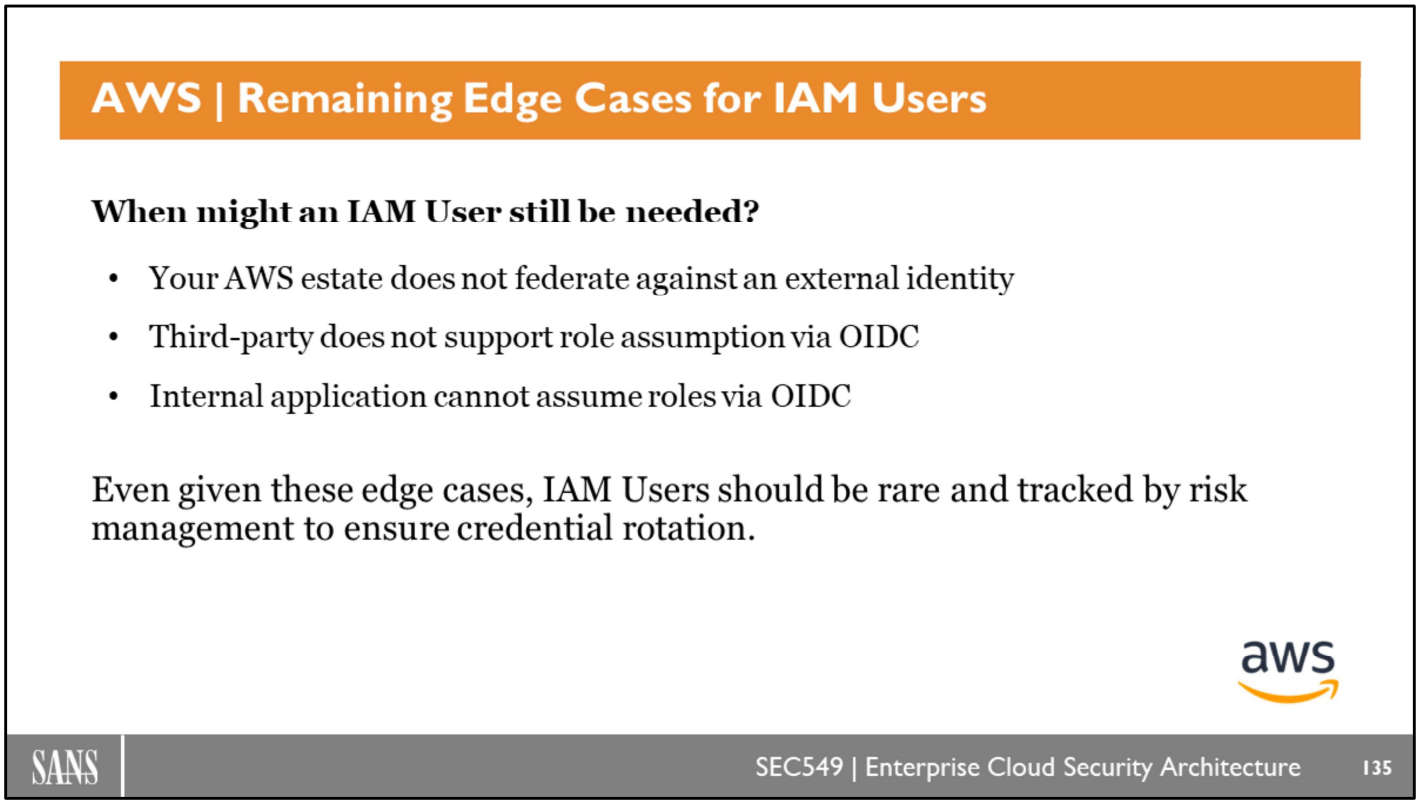
Identity Bastion Pattern in AWS
Businesses that have a myriad of AWS Accounts, each housing siloed IAM Users, might find that moving to an Identity-Bastion pattern is ideally the first step to take in centralizing their identity and access patterns in the Cloud.

In this pattern, the 'bastion' we are referring to is not a jumphost or EC2 instance, rather it is a dedicated AWS account into which all users can be centralized. It is through this Identity-Bastion Account[1] that your workforce needing access to AWS resources can be assigned a single AWS IAM User, no matter which Accounts they need to access in your Organization.

Components of an Identity Bastion Account
This Identity Bastion Account becomes the Central Identity Account in which all Users live. By creating a solitary account that initially becomes the only one allowed to house your IAM Users, you can greatly reduce the occurrence of Users and their associated durable credentials across your AWS estate.

Spoke Accounts
These are the AWS Accounts housing both IAM Roles and resources. The Roles are configured with trust policies allowing IAM Users in the Identity-Bastion Account to assume the Roles to access resources housed in 'Spoke Accounts' .

Reference:
[1]: https://sec549.com/id70



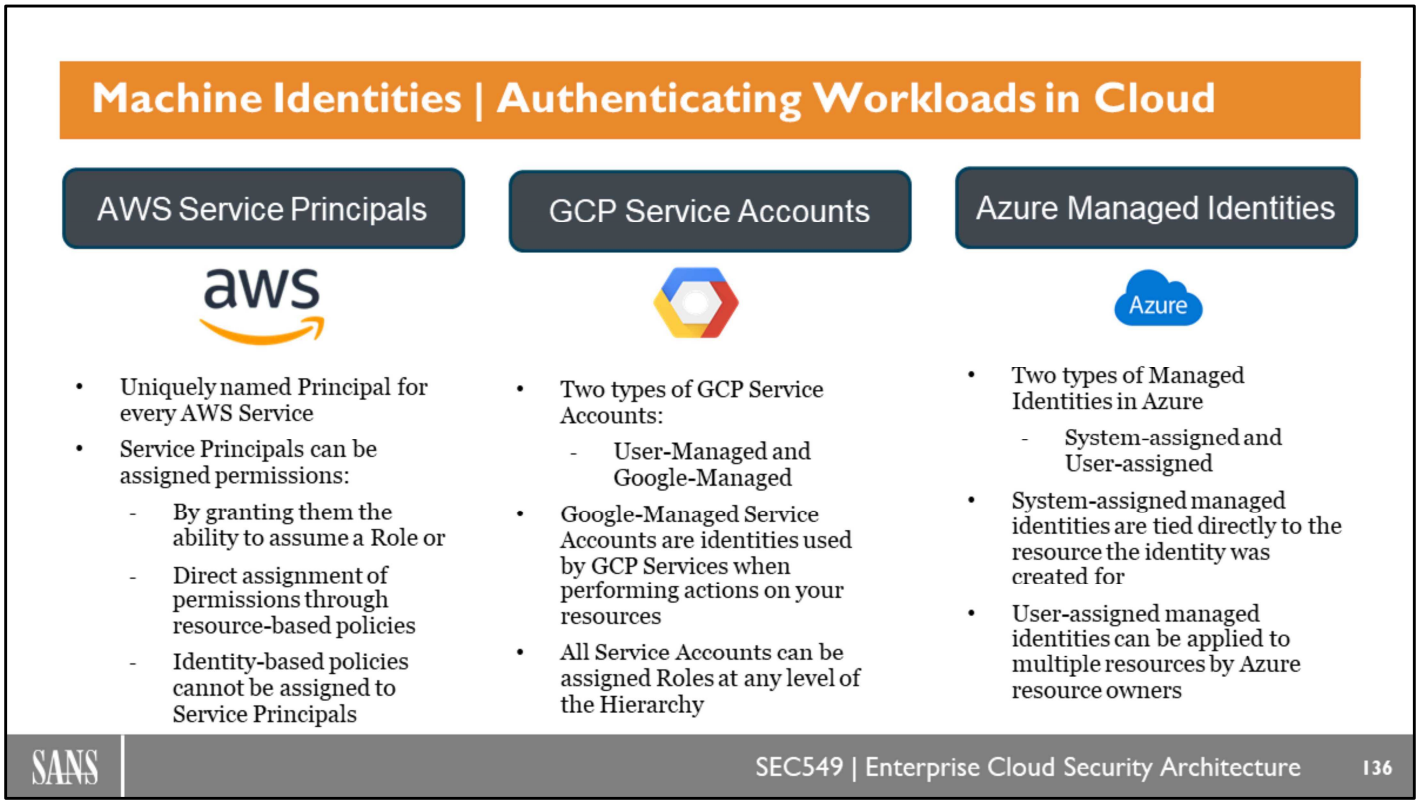
Remaining Edge Cases for IAM Users in AWS
Can you completely do away with AWS IAM Users all together? - Probably not.

The edge cases you will run into are:

- External systems that need to authenticate into your AWS estate and do not support OIDC. Instead, they need an access key to authenticate to AWS cloud control-plane APIs. This is an antiquated, less secure method of access, but in some cases, this might be your only choice. Before allowing an external party to be issued an access key, ask if they support OIDC authentication. AWS calls this mechanism, 'AssumeRoleWithWebIdentity'.

- If you need to have an AWS Lambda Function generate pre-signed S3 Links, the Lambda Function will need to have an access key assigned to an IAM User to perform this operation.[1]

Reference:
[1]: https://sec549.com/id71

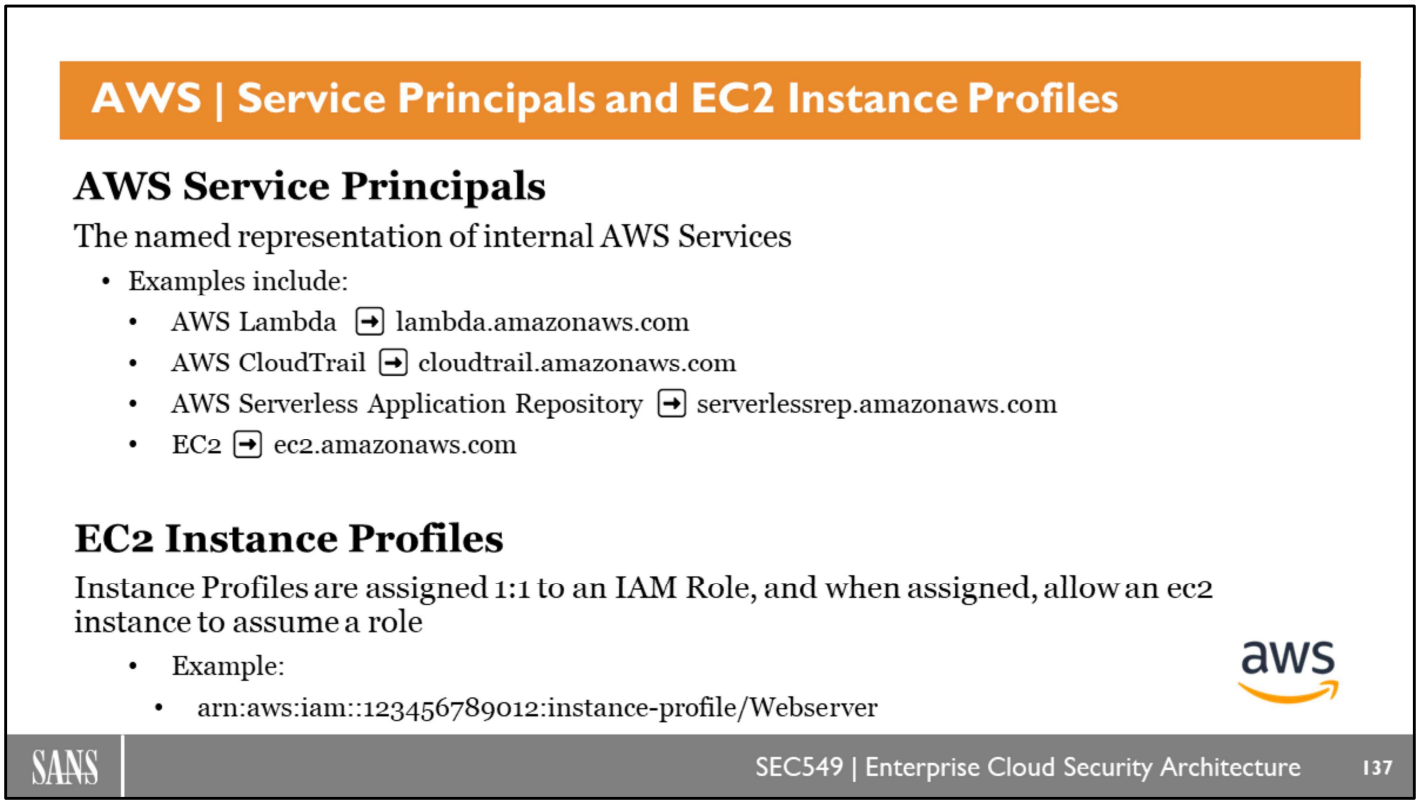


Machine Identity in The Cloud

Machine identities, those used by non-human actors, typically outnumber human entities in any sufficiently large system. Their importance in the Cloud has grown as the systems we design require resources and services to authenticate one another. As their importance grows, so too does the risk of them being abused by threat actors or misused due to human mistakes. In the upcoming slides, we'll review what machine identities look like in each cloud and the strategies you can use for securing them against abuse.

Each of the 3 major cloud providers have their own opinionated designs for machine identities. Despite many differences in architecture between them, all three clouds utilize Machine Identities as a method for assigning *entitlements, privileges, permissions to non-human entities.*

There are two types of machine identities in the cloud which the CSPs draw distinction between and treat different. First, we will address how an identity can be assigned to your own resource. Second, we'll look at how internal services are identified on the cloud service control plane.



What is a Service Principal?
A Service Principle is the named representation of AWS Services, such as EKS, CloudTrail, or Lambda. Whenever an AWS Service operates on your behalf, it uses its respective service principal to do so.

Example:

- When configuring CloudTrail, you're obliged to indicate which S3 bucket CloudTrail should deliver the logs. When CloudTrail writes logs to the specified bucket, it does so by using the identity of its Service Principal: *cloudtrail.amazonaws.com.*

Service Principals are not part of your AWS organization like IAM roles or Users, but the actions they take in an account can be affected by higher-level policy. For example, if an SCP has been applied restricting API access to only those Principals in your AWS Organization, conditional exceptions will need to be carved out to allow for the actions taken by Service Principals.

What is an EC2 Instance Profile?
EC2 Instance Profiles are identity containers assigned to EC2 Instances. They are:

- Unique identifiers attached to the compute instances
- EC2 instance profiles define "who" the instance is
- The E2 Instance Profile allows instances to assume IAM roles, which ultimately grants permissions to the instance

In AWS, internal Services use Service Principals to identity themselves, while the EC2 instance is identified with an EC2 Instance Profile. In both cases, these entities must be included in resource-based policy to acquire permissions. They cannot be assigned permissions with an identity-based policy.

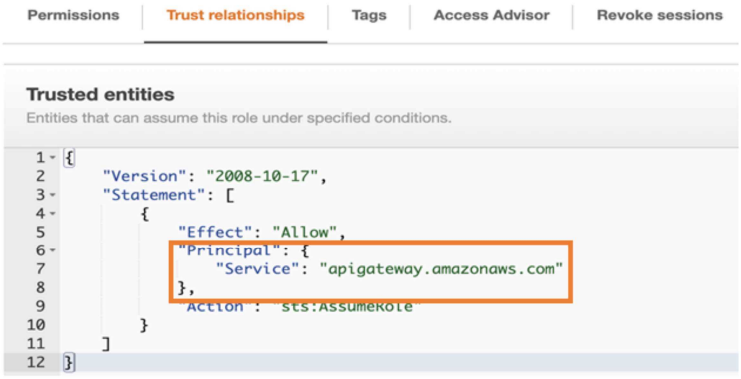
Reference:
[1]: https://sec549.com/id72

AWS | Service-Linked Roles

AWS Service-Linked Roles

A **service-linked role** is a special type of IAM Role that gives a Service Principal permissions to assume the Role and therefore access resources in your account.

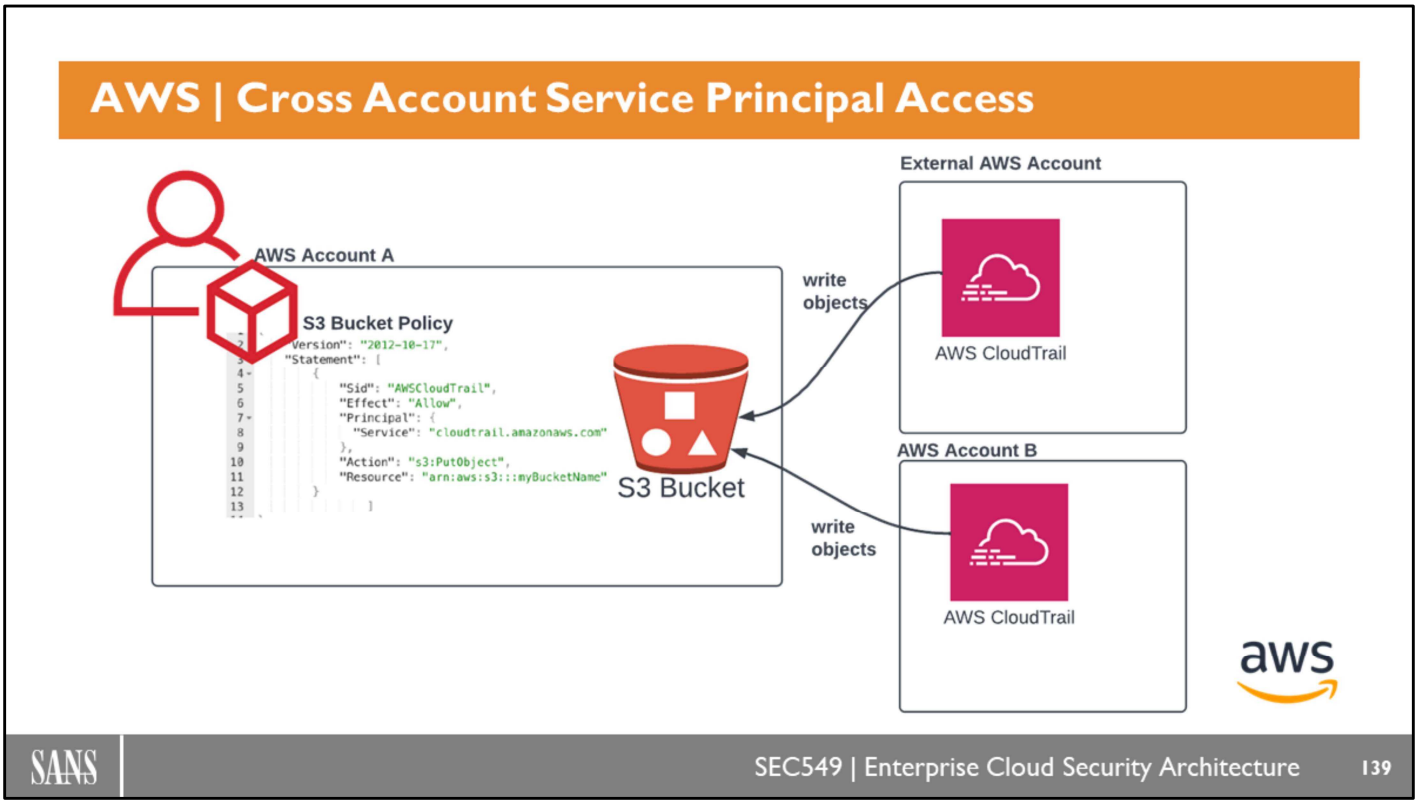
IAM Roles that can be assumed by an AWS service are called *service-linked roles*.
Service-linked roles must have an attached trust policy specifying which service which can assume the role.



Service-Linked Roles in AWS
A Service-Linked[1] role is nothing more than an IAM Role with a Trust Policy allowing an AWS Service to assume it. Instead of the Trust Policy specifying an IAM User or another IAM Role, here the Trust Policy indicates that an AWS Service Principal should be allowed to assume the Role and use any permissions attached to the Role.

The Principal type *'Service'* is the only type which does not allow wildcards.
- Example: You cannot do *"Service": "*.amazonaws.com"* if you ever wished to do so.

Reference:
[1]: https://sec549.com/id74



Cross Account Service Principal Access

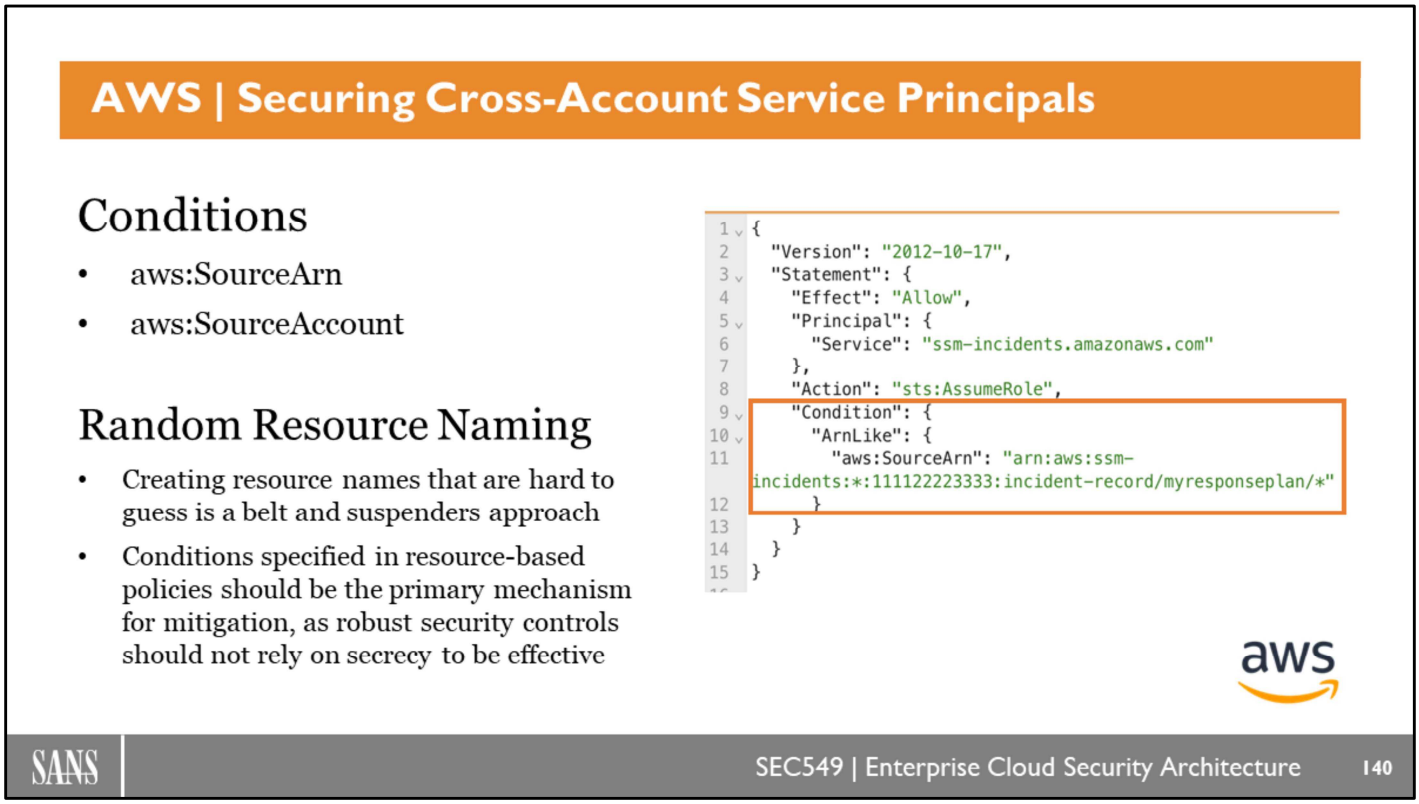
In AWS, Service Principals are not necessarily unique to each AWS account or customer; rather, they are global identities . In some cases, specifically when the service has cross-account capabilities, granting access to a Service Principal can open a security risk[1].

In the above example, a policy is applied to an S3 bucket allowing the internal AWS service, CloudTrail, to write objects to the bucket. CloudTrail across the customer's organizations will write log entries to this central bucket. This policy also authorizes any trail in any AWS Account (even one external to the Organization) to write to the bucket.

This type of cross-service interaction can be weaponized as a form of the 'confused deputy' problem[2]. The confused deputy problem is a security issue where an entity (e.g., a service in an external account) that doesn't have permission to perform an action can coerce a more-privileged entity (e.g., a service in internal account) to perform an action.

Which services can be affected by the cross-account vulnerability? Its not yet clear. Research into this space is very new; however, early indications suggest that any AWS service that can be benevolently used cross-account can also be manipulated and used in a malicious manner.

References:
[1]: https://sec549.com/id72
[2]: https://sec549.com/id73



Securing Cross-Account Service Principals
Now that we've seen the structural issues unique to AWS Service Principals, how should we design their access to ensure our Accounts are not vulnerable to the class of '*confused duty*' issues inherent to the platform?

When granting an external party access to an IAM Role, an external ID condition is used to mitigate this issue[1]. When granting access to a Service Principal, different global condition keys can be leveraged to limit exactly which Service Principal can assume the Role.

***aws:SourceArn*[2]**
Set this condition key to prevent an AWS service from being manipulated during transactions between services. Set the value of this condition key to the ARN of the resource in the request. Typically, this value will be the ARN of the resource assuming the role. It might be the ARN of the CloudTrail trail or, in the example, the ARN listed is that of the response plan resource from Incident Manager.

aws:SourceAccount[3]
Set this condition key to prevent an AWS Service from being manipulated during transactions between services. Set the value of this condition key to the account of the resource expected in the request. Typically, this value will be an ID of an AWS Account within your Organization.

A non-obvious strategy to help mitigate the confused deputy problem in AWS is randomization in naming. You might already append random strings to the end of resources to help conserve global namespace (as with S3); however, this practice has an additional security benefit. An attacker without the knowledge of your naming convention would not be able to access any resources within your Accounts.

References:
[1]: https://sec549.com/id75
[2]: https://sec549.com/id76
[3]: https://sec549.com/id77

GCP | Types of Service Accounts

User-Managed Service Accounts	Google-Managed Service Accounts
Default Service Accounts created on your behalf to serve as Identities for GCP Services OR a Manually-created Service Account	Service Account Owned by Google Created when enabling Google APIs in your Project
Will be listed as a Resource in the Project in which it was created	Will not be listed as a Service Account in your Project but its IAM Policy Binding is visible
You can generate keys, retrieve access tokens, and impersonate User-Managed Service Accounts	You cannot generate keys or impersonate Google-Managed Service Accounts You can access their access tokens via the Metadata Service

Types of Service Accounts in GCP
Google takes a wholly different approach to machine identities than does AWS. Bound within the GCP Service Accounts are two distinctive types, the User-managed Service Account and the Google-Managed Service Account. As you read the specifics of these two types of service accounts, take note of their equivalents in AWS. The Google-Managed Service Account has a clear cognate with the AWS Service Principal. The User-managed Service Account in GCP does not have a clear one-to-one relationship to an IAM construct in AWS. To achieve some of the same goals in AWS, customers might use the combination of Roles and IAM Users.

GCP Service Account 101s[0]
- Service Accounts do not have passwords and cannot log in via browsers.
- Service Accounts are associated with private/public RSA key-pairs that are used for authentication to Google. You can upload your own public key and associate it with a Service Account or have GCP generate a key pair for you.

Cloud IAM permissions can be granted to allow other users (or other service accounts) to impersonate a Service Account. Service Accounts are **not** members of your G Suite domain unlike user accounts. For example, if you share assets with all members in your G Suite domain, they will not be shared with service accounts.

Types of Service Accounts[1]

User-managed service accounts
Default Service Accounts: If in your GCP Project, you enable the Compute API, a Compute Engine Service Account is created for you by default. It is identifiable using the email: ***project-number***-compute@developer.gserviceaccount.com. This Service Account identity will be used as the default Identity for all Compute Instances created.

If your project contains a App Engine application, the default App Engine Service Account is created in your project. It is identifiable using the email:

project-id@appspot.gserviceaccount.com

These Service Accounts (SAs) among other default SA's are used as Identities for your Resources when they interact with Google APIs. When creating Resources, you can and SHOULD create a purpose-built Service Account, not utilizing the default Service Account. This is because the default Service Account is automatically assigned a Role with excessive Privilege, the Editor Role.[2]

Default Service Accounts are associated with all GCP Resources that are backed by Compute such as Cloud Functions, Cloud Run, DataFlow and GKE Nodes. By default, these service accounts automatically receive the Editor role when they are created. This behavior can be altered, and the automatic granting of the Editor role prevented with the application of a GCP Organizational Policy constraint.

Manually Created Service Accounts
Service Account can be created manually as well. With this version of a User-Managed Service Account, you will supply a name for the Service Account and will appear in this format:

service-account-name@project-id.iam.gserviceaccount.com

You can create up to 100 Service Accounts per project (including the default Compute Engine Service Account and the App Engine service account) using the IAM API, the Cloud Console, or the G Cloud command-line tool. Default Service Accounts and the Service Accounts you explicitly create are both User-Managed Service Accounts.[3]

With either version of User-Managed Service Accounts, you can optionally export the private key for the Service Account. Obviously, not exporting a private key is the more secure option but this is not feasible in all circumstances. If you have any external system like Jenkins or an application that needs to connect to Google APIs, these entities do not have a Google Identity, often exporting the private key for a service account and embedding where the external system can access is your only option for enabling multi-cloud or external functionality.[4]

The security of Service Accounts not only depends on the security of its private key, but also who is assigned the IAM Roles to manage and operate those Service Accounts. Google provides a host of functionalities that enable the use of Service Account without the need to export its private keys.[5] There is more on Service Account Impersonation in upcoming slides.

Google-managed service accounts
Google-Managed Service Accounts are created and owned by Google. These accounts represent different Google services, are created when Google APIs are created, and each account is automatically granted IAM roles to access your Google Cloud project.

An example of a Google-Managed Service Account is a Google API Service Account identifiable using the email:

project-number@cloudservices.gserviceaccount.com

This service account is designed specifically to run internal Google processes on your behalf and is not listed in the **Service Accounts** section of the Cloud Console. Google services rely on these Google-Managed Service Accounts having access to your project through IAM Policy. Google does not recommend you remove the Roles assigned to these Service Accounts, although the Roles tend to be highly privileged.[6]

Because you do not own Google-Managed Service Accounts, keys cannot be generated for them, nor can they be impersonated. Interestingly though, when utilizing GCP services, there are scenarios where the access token for the Google-Managed Service Account can be retrieved from the Metadata Service.[7]

References:
[0]: https://sec549.com/id78
[1]: https://sec549.com/id79
[2]: https://sec549.com/id80
[3]: https://sec549.com/id81
[4]: https://sec549.com/id82
[5]: https://sec549.com/id83
[6]: https://sec549.com/id84
[7]: https://sec549.com/id85

GCP | SAs as Both Identities and Resources

Granting the *serviceAccountTokenCreator* Role to serviceaccount-1:

```
{
  "bindings": [
    "members": [
      "serviceaccount-1:project-number-
          compute@developer.gserviceaccount.com"],
          "role": "roles/iam.serviceAccountTokenCreator"
  }
}
```

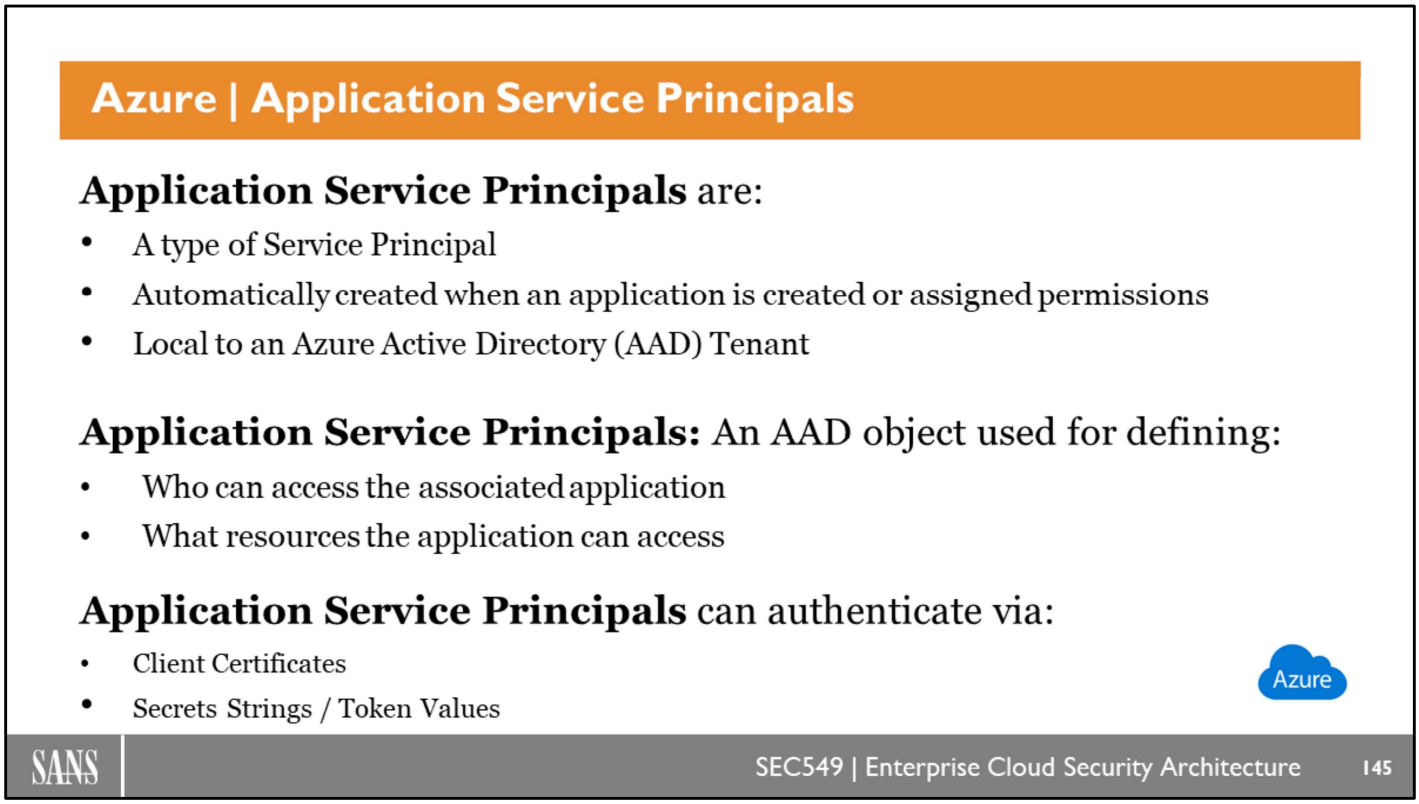
SANS | SEC549 | Enterprise Cloud Security Architecture 144

Service Accounts as both Identities and Resources

Service Accounts in GCP are both an Identity and a Resource.[1] This concept is best illustrated by talking about Service Account Impersonation. Without the need to export or distribute private keys, we can enable one Service Account (serviceaccount-1) to impersonate another Service Account (serviceaccount-2). This is accomplished with IAM Roles and the granting of permissions to generate short-lived access tokens.[2]

The IAM Binding shown above grants the Service Account Token Creator Role to the Identity, serviceaccount-1. When this Policy is bound to serviceaccount-2, it would enable serviceaccount-1 to impersonate serviceaccount-2 by generating access tokens on its behalf.[3] Other Resources that can have IAM Policy attached include Organizations, Folders, Projects, and over 100 Resource Types, Service Accounts just being one of them. If you were to grant the Service Account Token Creator Role at the Project level, it would give the User access to all Service Accounts in the Project, including Service Accounts that may be created in the future.[4]

References:
[1]: https://sec549.com/id86
[2]: https://sec549.com/id87
[3]: https://sec549.com/id88
[4]: https://sec549.com/id89



Service Principals in Azure
A Service Principal is an umbrella term for one of three types of identities in Azure: An application service principal, managed identity, or legacy. Service Principals are nothing more than an Azure AD object representing an application or resource that needs to be assigned Azure permissions. In these next two slides we will cover the usage of Application Service Principals and Managed Identities.

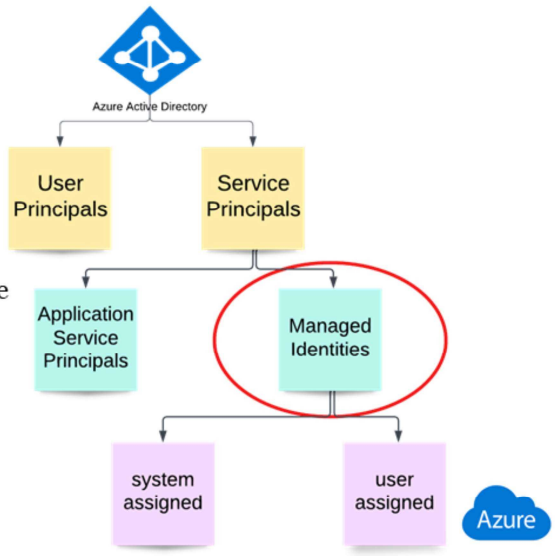
Application Service Principals
When applications are registered in Azure Active Directory (AAD), they can be granted permissions in the Azure Tenant through an Application Service Principal. Not only can Application Service Principals be granted permissions, but they can also be assigned durable credentials, like passwords and certificates to be used in authentication. Application owners inherently have permissions to add a password or certificate to the Service Principal associated with their Application.

Unlike an AWS Service Principal, an Azure Service Principal can incur login events with their credentials. Auditing Azure Service Principal log-ons is a good first step in understanding if Service Principal credentials are being mishandled.

Azure | Managed Identities

Managed Identities

- A type of Service Principal
- No long-term durable credentials issued – only temporary access tokens
- System-assigned
  - An available identity type for ~ 20 types of Azure resources including Virtual Machines and App Service
  - Automatically created and deleted when the associated resource is deleted
  - One-to-One relationship with resources
- User-assigned
  - Bound to a subscription, resource group, and region
  - Can have a one-to-many relationship with resources



Managed Identities in Azure
A Managed identity is another type of Azure Service Principal. Formerly called a 'Managed Service Identity' (MSI) by Microsoft, in this course we will stick to the calling them simply managed identities.

Azure offers two types of Managed Identities:

- **System-assigned managed identities** have their lifecycle tied to the resource that created them.
- User-assigned managed identities can be used on multiple resources and are created independently from a resource.

Common between the two types of managed identities is how they authenticate. Both types are intended to be used programmatically by applications running on Azure resources where they can be granted temporary access tokens used to access cloud-plane APIs. A clear benefit to managed identities is there are no long-term, durable credentials associated them. In this sense, they can be considered analogues to EC2 Instance Profiles in AWS.

Now let's take a look at where these two varieties of Managed Identities differ.

System-assigned Managed Identities
On select services, Azure allows a managed identity to be created along with the creation of the resource. These system-assigned identities uniquely identity that Azure resource. They are created and deleted with the creation and deletion of resources and cannot be created, updated or deleted independently.

User-Assigned Managed Identities
Managed identities may be created by the end user, independent from any resource like a Virtual Machine it might be attached to. In user-assigned managed identities the identity, its permissions and any attachments to resources are managed separate from the lifecycle of the resource.

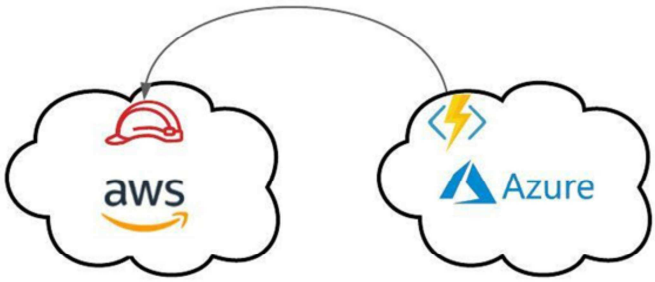
This page intentionally left blank.

Authenticating Across Clouds

Authenticating workloads across clouds is inevitable

Accomplish secure, cross-cloud access to resources:

- Prefer OIDC integrations between parties
- Generating short-term temporary credentials
- Eliminate then need for secrets rotation
- Avoiding the creation of long-term, durable credentials



Authentication Across Clouds

There are few scenarios these days that do not include the possibility of having a business existing in a multi-cloud environment. Use cases for having sprawling workloads in more than one cloud are inevitable. Some common use cases include:

- Enabling a background application or continuous integration/continuous delivery (CI/CD) pipeline that runs outside of your primary cloud. A great example of this use case is authenticating GitHub Actions[1].
- Enabling users of a web application that runs outside of your primary cloud to access cloud resources such as cloud file storage.

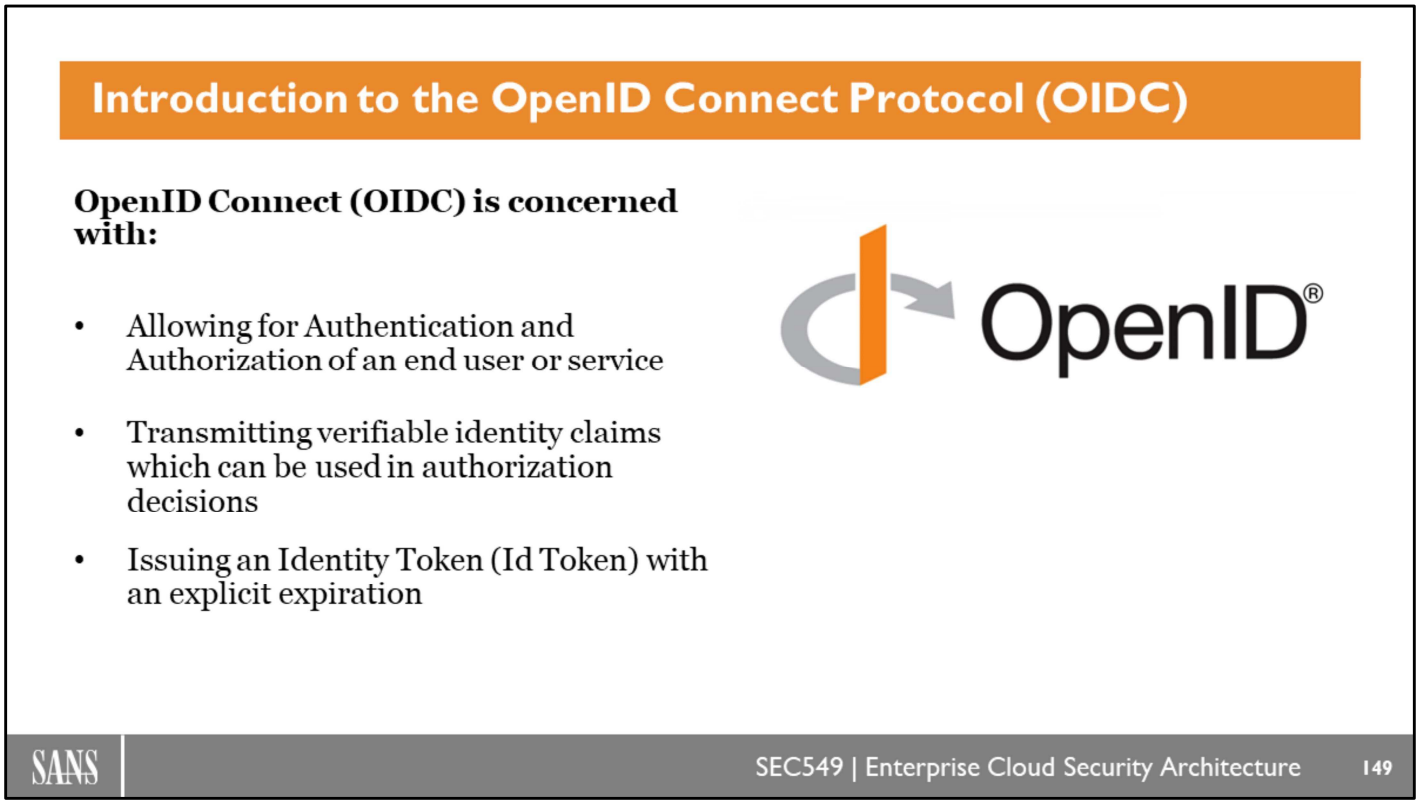
How can you enable *secure access* to cloud resources cross-cloud?

The traditional way was insecure and involved generating long-term, durable credentials for external parties to authenticate into your Cloud estate. In AWS, for example, it meant generating an IAM User for every external party and then creating an Access Key for the User. There are many problems with doing it the 'old way', particularly having to have a mature secrets rotation program for any long-term, durable credentials.

The more secure way avoids creating Access Keys for every external entity needing access and instead prefers managed integrations that generate short-term temporary credentials, limiting the blast radius if a credential Is compromised. This eliminates the need for secrets rotation.

In the next few slides, we'll learn about the Open Id Connect Protocol (OIDC), what credentials are generated during authentication and how it's used to allow external parties across cloud.

Reference:
[1]: https://sec549.com/id90



Introduction to OpenID Connect (OIDC)
OpenID Connect (OIDC) is a standardized identity layer built on top of OAuth2. It is a protocol used for user authentication, used to facilitate user federation against a centralized Identity Provider.

Let's review some basics about the protocol first before diving into how its used in the Cloud.

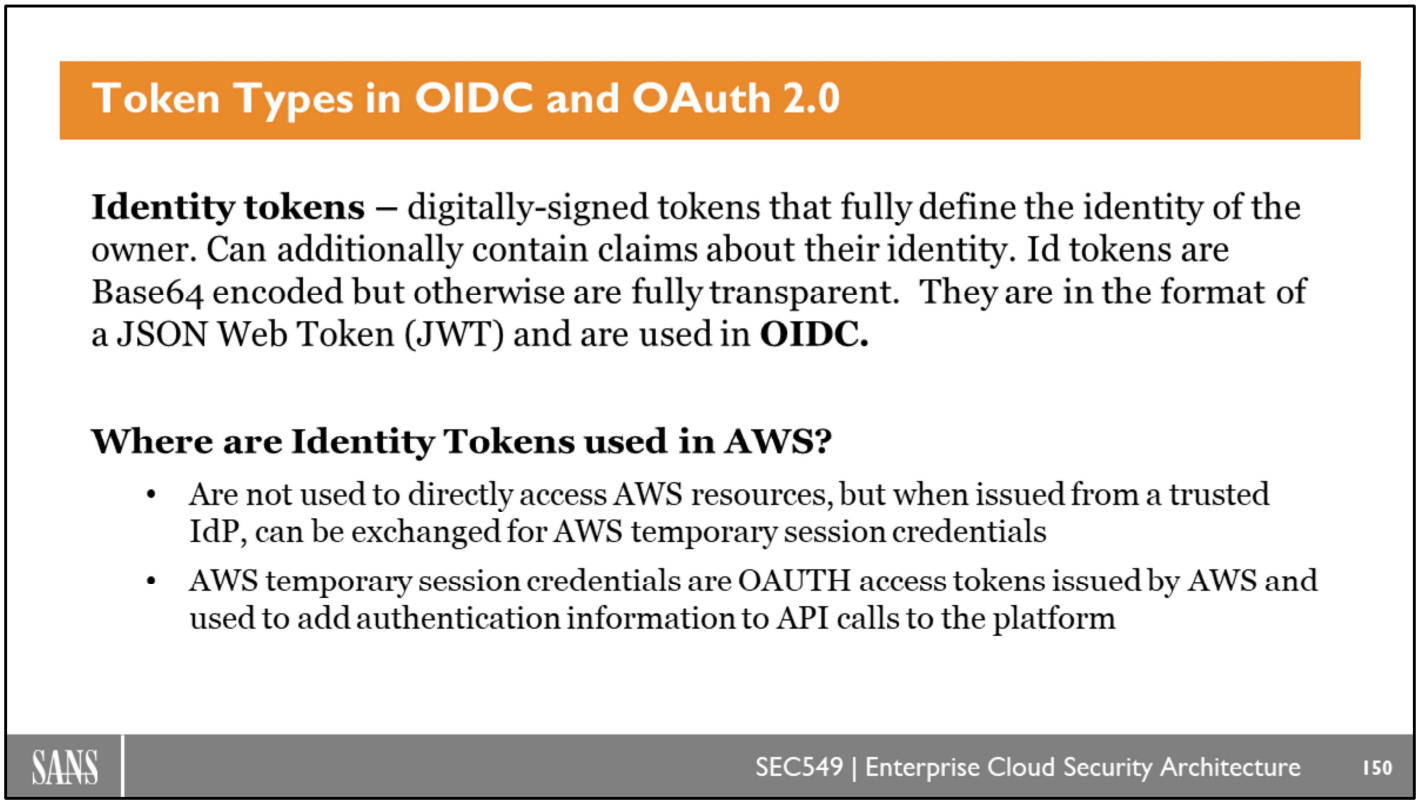
- OIDC allows a client to validate an authenticated state of an end user by validating the provided ID Token.
- OIDC allows a client to "know something" about the end user's identity, which is helpful in making authorization decisions.

OAuth 2.0 represents the framework enabling clients (apps or websites) to gain access to resources. OAuth 2.0 is a massive standard that eliminates the need for apps or websites to continually request permission to access resources. Instead, OAuth 2.0 is a mechanism for delegating resource access.

OAuth 2.0 deals strictly with the granting of authorization to third-parties. This can be seen if you've ever added an authorized application to Twitter or Google Drive. The application will ask the end user to grant permission to access the resources on their behalf. Authorizing this type of request will result in the client (Twitter or Google Drive) being issued an access token, sometimes referred to as a *bearer token*.

OIDC[1] adds value to this process by allowing the ability to authenticate an end user. This makes the OIDC protocol ideal to use in scenarios in the cloud where we need to authenticate an external entity without assigning them long-term durable credentials like access keys. OIDC is frequently used for end user authentication in conjunction with OAuth. End-user authentication flows in Google cloud use both the OIDC and OAuth protocols to both assert the identity of an authenticated user and allow them to access resources they are authorized to access.[2]

References:
[1]: https://sec549.com/id91
[2]: https://sec549.com/id92



Token Types used in OIDC and OAuth2.0

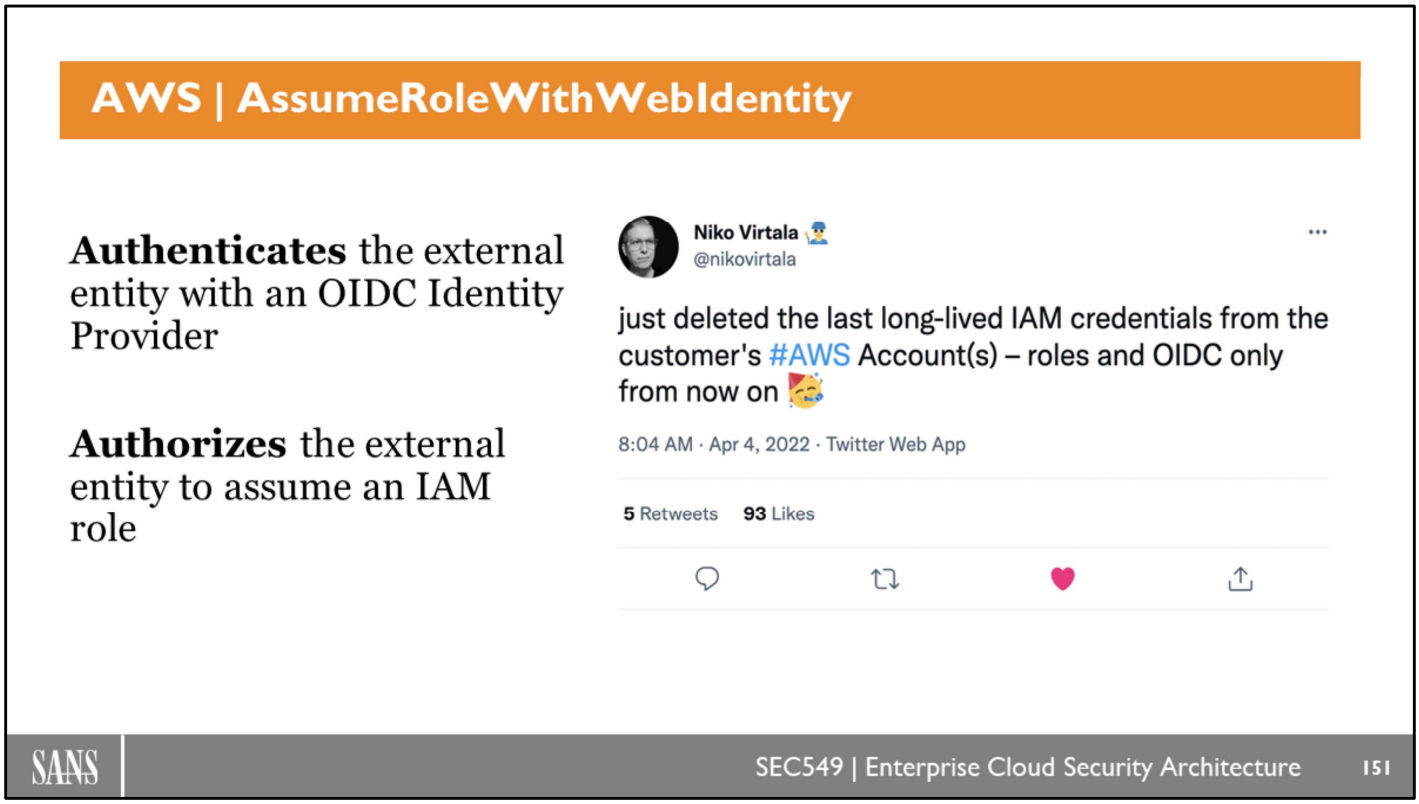
The OIDC protocol involves the generation of Id Tokens. These tokens are nothing more than JSON objects that are used for authentication and authorization of users for a resource. These four token types are important to the process:

- ***Identity tokens*** – these verify a user's identity, acting as the user's passport. They assert identity, indicate the issuing Identity Provider, and can provide information about when and how the user was authenticated. These are digitally signed and need to be verified by the client. Id Tokens are nothing more than JSON objects that are used for authentication and authorization of users when accessing a resource.

Token used in OAuth2.0

- ***Access tokens*** – these tokens determine what resources the user has been granted access to, often expressed in the scope of the token. The scope of a token determines which resources are delegated to access token bearers. This token is submitted when a user attempts to access the resource. Sometimes called bearer tokens, they will provide access to the bearer, regardless of who possesses it at the time. These are short-lived tokens, so the blast-radius of a stolen token is limited.

- ***Refresh tokens*** – access tokens can be issued with a wide range of expiration dates but are often very short lived, typically under an hour.. Without a new access token, the access to protected resources is revoked. Refresh tokens have a much longer lifespan, sometimes on the order of many days or weeks instead of an hour. Refresh tokens are used to acquire new, refreshed access tokens.



AWS Role Assumption using 'AssumeRoleWithWebIdentity'
Now that we've seen the core components of OIDC, let's see how the protocol can be used to allow an external workload to assume a role in an AWS Account.

To review, there are a few mechanisms in AWS to allow Principals to assume an IAM Role: *assume-role* (standard method), *assume-role-with-saml,* and *assume-role-with-webIdentity.*

- The *assume-role* (standard method) API is used when the end user is an AWS IAM User and has already been authenticated.
- The *assume-role-with-saml* API is used when the end-user is authenticating with SAML against a configured Identity Provider.
- Finally, the *assume-role-with-webIdentity* API is used when the end-user is authenticating with OIDC against a configured Identity Provider.

OIDC Providers[1]
Allowing an external entity to authenticate via OIDC first requires a configuration between your AWS account and the provider. This establishes trust between your AWS account and the Identity Provider such as Google, Salesforce, or GitHub, allowing your AWS Account to validate Id Tokens issued by the configured provider.

Role
As we've seen, roles are collections of permissions that can be used by authorized Principals. When enabling cross-cloud authentication via OIDC, a Role will be created with a policy attached that defines the scope of actions the Principal is allowed to perform.

Trust Policy
The trust policy needs to allow Principals who've authenticated with the configured OIDC provider to assume the Role. Additional conditional arguments should be specified to limit who from the OIDC provider can assume the Role.

Reference:
[1]: https://sec549.com/id93

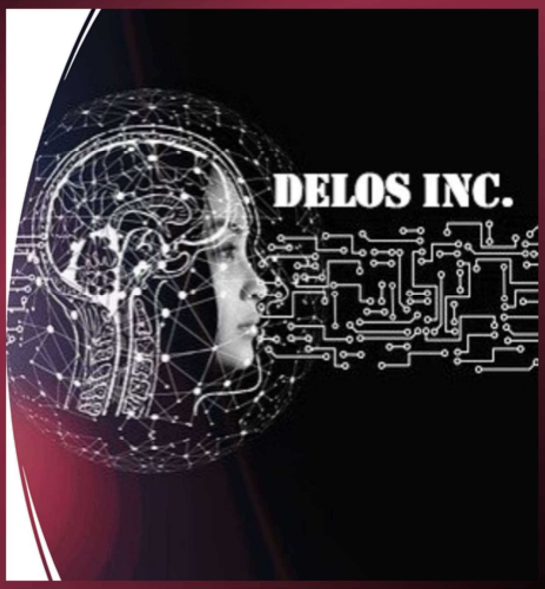
Cloud Journey Phase: Discovery!

Scenario:

As the new AWS Organization begins to take shape, stones are being overturned and previously unknown legacy environments are being uncovered.

Delos needs to continue preparing its new Greenfield environment while quickly uplifting the access patterns of legacy AWS Accounts.

Accomplishing this process will stem any immediate bleeding while preparing for a future shift to a fully federated world.



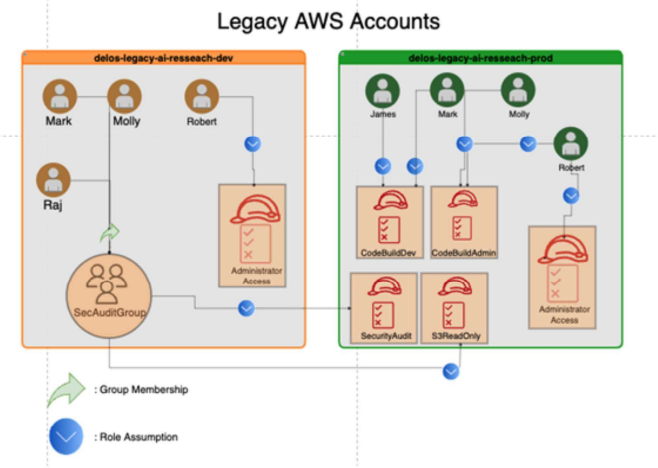
Reference:
Image Source: Pixaby user geralt: https://sec549.com/id51

Lab 1.4 – Current State Architecture

The AI Research team has been maintaining their own AWS accounts, outside of official IT processes. These two shadow accounts have come in from the cold and are now members of the new Delos AWS Organization.

While plans are being finalized to centralize identity across the Org, we need to quickly uplift these tangled access patterns to help reduce the immediate risk they present.

Untangling and streamlining access will also help prepare the users in these accounts for a future of having federated AWS access.



This page intentionally left blank.

Lab 1.4 – Transitioning Access from Users to Roles

Estimated Time: 45 minutes

You are presented with two legacy AWS Accounts with siloed identity and access patterns. These accounts have operated as 'Shadow Accounts' by the research science team. The lack of oversight has led to a tangled web of un-managed users and policies making it difficult to reason about who has access to what.

In this lab, your task is to untangle this web of local identity to prepare the end users for an eventual migration to a centralized identity pattern with AWS Identity Center (IdC).

Preparation

- View **Lab 1.4** in the course workbook
- Open the workbook in an incognito window
- https://workbook.sec549.com
  - Username: H03-student
  - Password: million-sailor-DEAR

Objectives

1. Reprovision users into an identity bastion account.
2. Recreate their access into spoke accounts using assume role policies.
3. To get you started, the new access patterns for the user 'Mark', have already been configured.

This page intentionally left blank.

Lab 1.4 – Summary

In this lab, you...

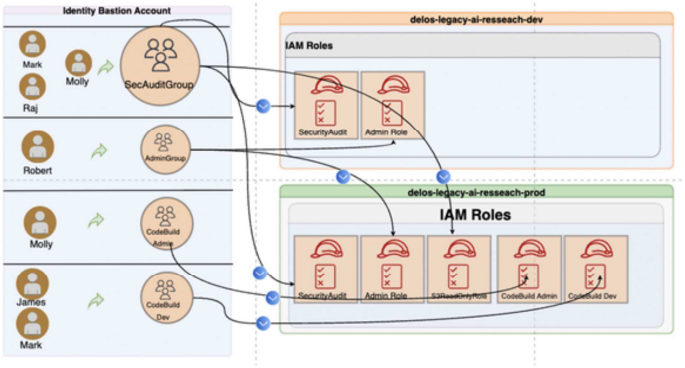
- Recreated users in the Identity Bastion Account
- Defined new IAM groups in the Identity Bastion Account to facilitate each user's access to cross-account Roles
- Indicated which users would assume which IAM roles to maintain their existing access
- Logged into the AWS Console to answer the ⚡ See it in Action ⚡ questions.

This page intentionally left blank.

Lab 1.4 – End State Architecture

Access patterns have been streamlined so that every individual on the AI Research Team only has a single IAM user to manage. From this single user housed in the Identity Bastion Account, users are arranged into AWS-native groups from which they are allowed to assume roles into their operational accounts.

Previous access into legacy accounts has been maintained but who has access to what is easier to reason about, easier to audit, and less prone to misconfigurations.



Lab 1.4 – Transitioning Access from AWS IAM Users to Roles – End State Architecture
In the end state architecture, Users' access patterns have been streamlined so that every individual on the AI Research Team only has a single IAM User to manage. From this single User housed in the Identity Bastion Account, Users are arranged into AWS-native Groups from which they are allowed to assume Roles to gain access to their operational accounts.

Previous access into legacy accounts has been maintained but who has access to what is easier to reason about, easier to audit, and less prone to misconfigurations.

This page intentionally left blank.

Section 2 – Implementing Zero Trust in the Cloud

Module 1 – Introduction to Cloud Migrations

- Cloud migrations and implementing a multi-cloud strategy

Module 2 – Implementing Zero-Trust

- History of Zero Trust and Implementation Considerations
- Using AWS Cognito as an Authentication or Authorization Platform

Module 3 – Establishing Perimeters for Application Access

- Review of the building blocks of a cloud-native network
- Building network-layer guardrails with a centralized control over network components

Module 4 – Establishing Perimeters for Data Access

- S3 Use Cases
- Access Control For Shared Data Sets

This page is left intentionally blank.