549.2

Implementing Zero-Trust in the Cloud

SANS | GIAC CERTIFICATIONS

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

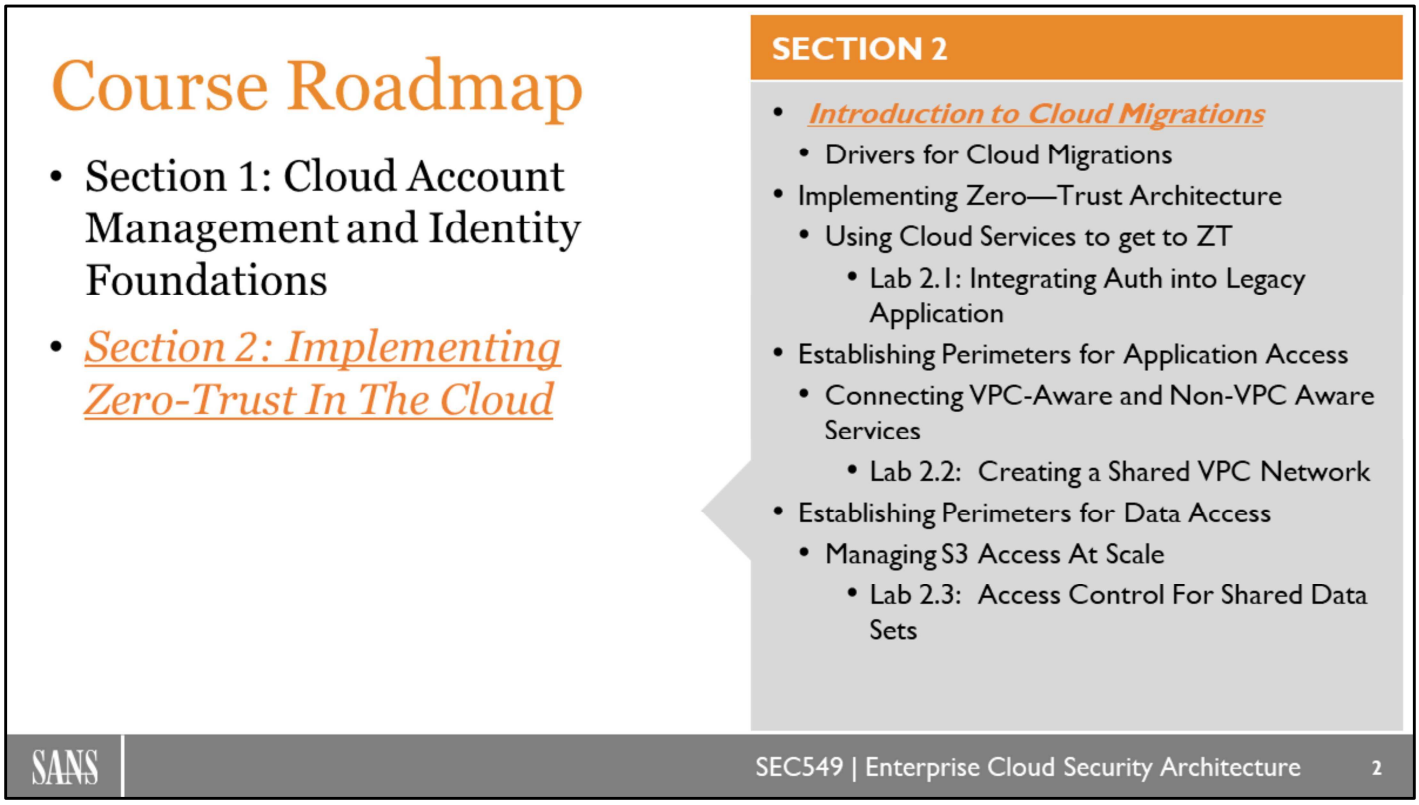
All reference links are operational in the browser-based delivery of the electronic workbook.

SEC549.2 | Enterprise Cloud Security Architecture

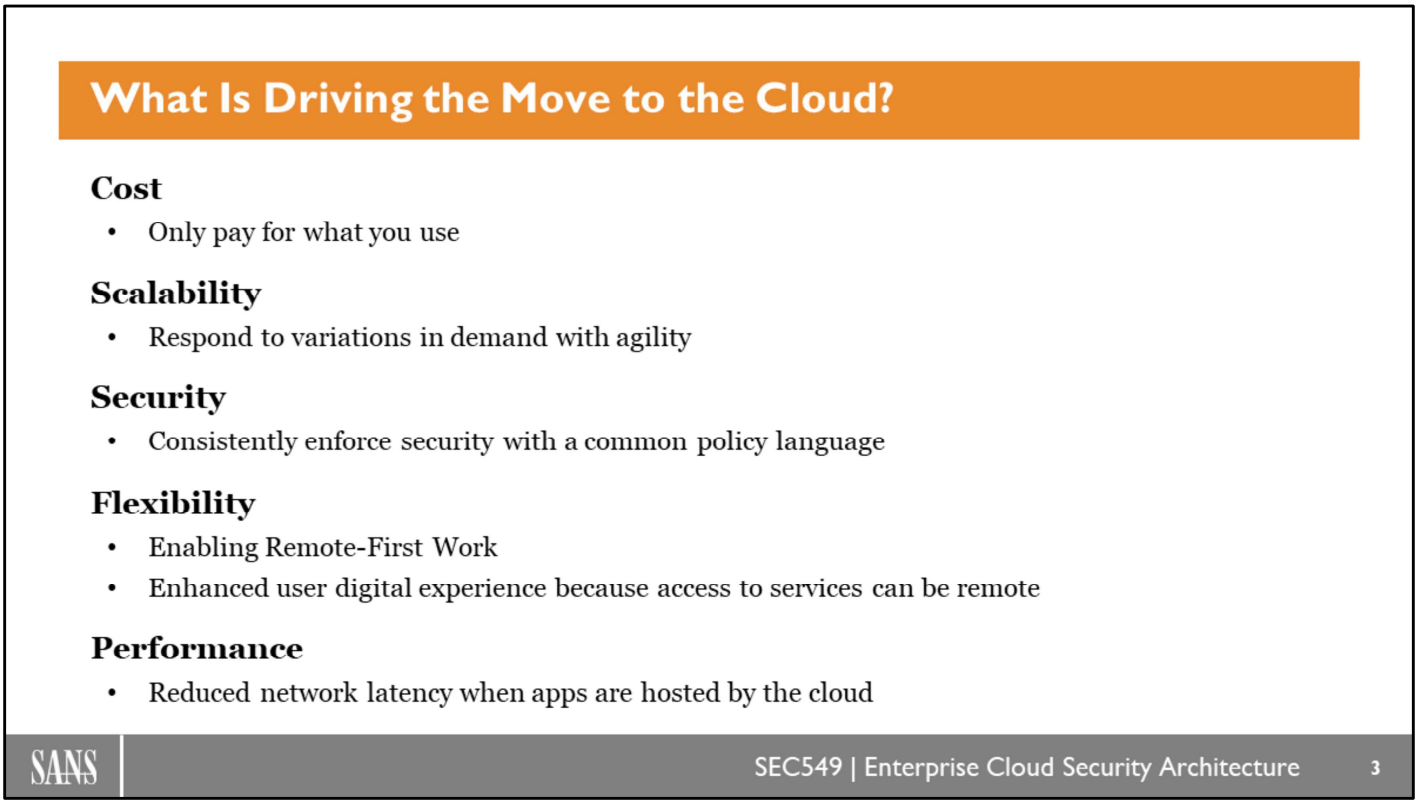
SANS

Implementing Zero-Trust in the Cloud

This page intentionally left blank.



This module begins with a discussion of cloud migration – a task many organizations are already in some stage of accomplishing. What drives migration to the cloud and how can it be undertaken? Students learn the various ways cloud migration can be approached, beginning with simply rehosting a service to completely replacing an on-premises services with a cloud-based one. The module also covers the pros and cons of migrating to the cloud, including the advantages of adopting a single cloud versus utilizing multi-cloud and/or hybrid cloud environments.



Why Do Organizations Migrate to the Cloud?

- **Cost –** maintenance and upgrades are the cost responsibility of the cloud providers, leaving businesses better positioned to focus on new products, innovation, and improvements to existing products.
- **Scalability** – cloud computing supports larger workloads and many more users than on-premises infrastructures. Scaling up and down to meet demand is simpler in the Cloud and helps controls costs
- **Security –** while security is always a shared experience, there are large swaths of infrastructure security managed by the cloud provider using a common policy language that allows for more automation and fewer person-related user errors in security management.
- **Flexibility –** cloud-based systems enable remote-first work and enhanced user digital experience. Access to services can be from anywhere, whether the user is an employee or customer.
- **Performance –** network latency is often reduced to allow for higher throughput of more users when the apps and websites are hosted in the cloud.

Challenges to Cloud Migration

Many existing organizations never planned to move to the cloud at the outset. Instead, they had 2-3 years of cloud hyper-growth, shadow accounts, and experimentation followed by a reigning in of cloud assets. The move to the cloud wasn't part of the original business strategy; rather, developers and operations teams used the cloud as a simple and rapid way to deploy their ideas, apps, and operations. This lack of strategy can make cloud migration more challenging once it becomes clear this is the route the organization wants to go in its totality.

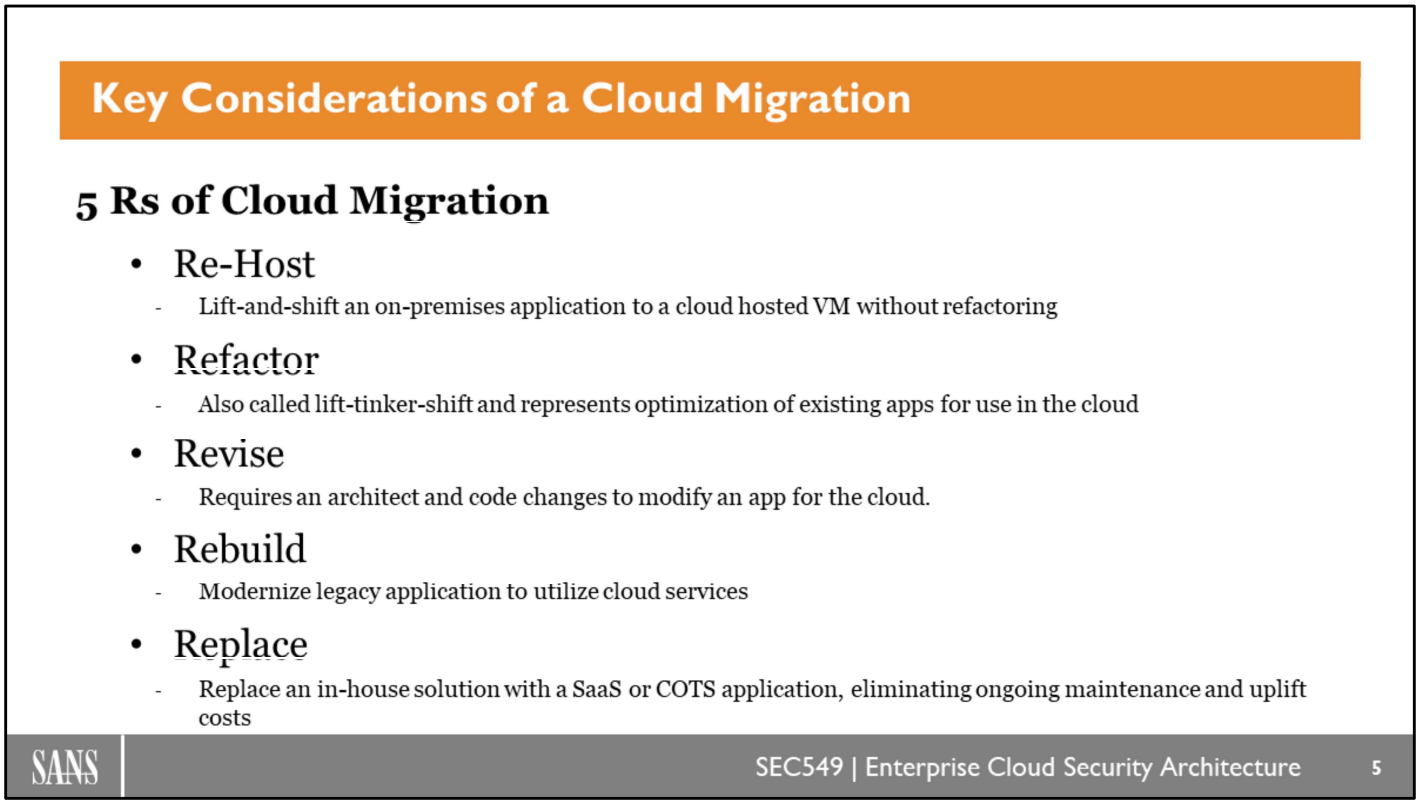
Once the decision is made, problems can ensue…

- ***The organization lacked sufficient strategy*** – without a rigorous end-to-end cloud migration strategy, too little consideration is given to the requirements each app and dataset have in the cloud environment. From a security standpoint alone, this increases the organization's vulnerability during and after the migration.

- ***The organization failed to consider cost management*** – even if cloud migration is cost effective, organizations who don't have clear KPIs cannot determine just how successful the migration may have been.

- ***Vendor lock-in issues arise*** – many organizations begin with the cloud provider they believe will provide them with what they need for the long term. When this is no longer the case, they will quickly discover that changing providers is difficult and potentially expensive to accomplish.

- ***Data security and compliance issues arise*** – cloud-based services involve shared responsibilities. The cloud service providers secure the infrastructure, while the organization/customer secures the workloads and data. Even if the cloud provider builds in robust security measures, the organization must configure those measures to maintain the entirety of the security.

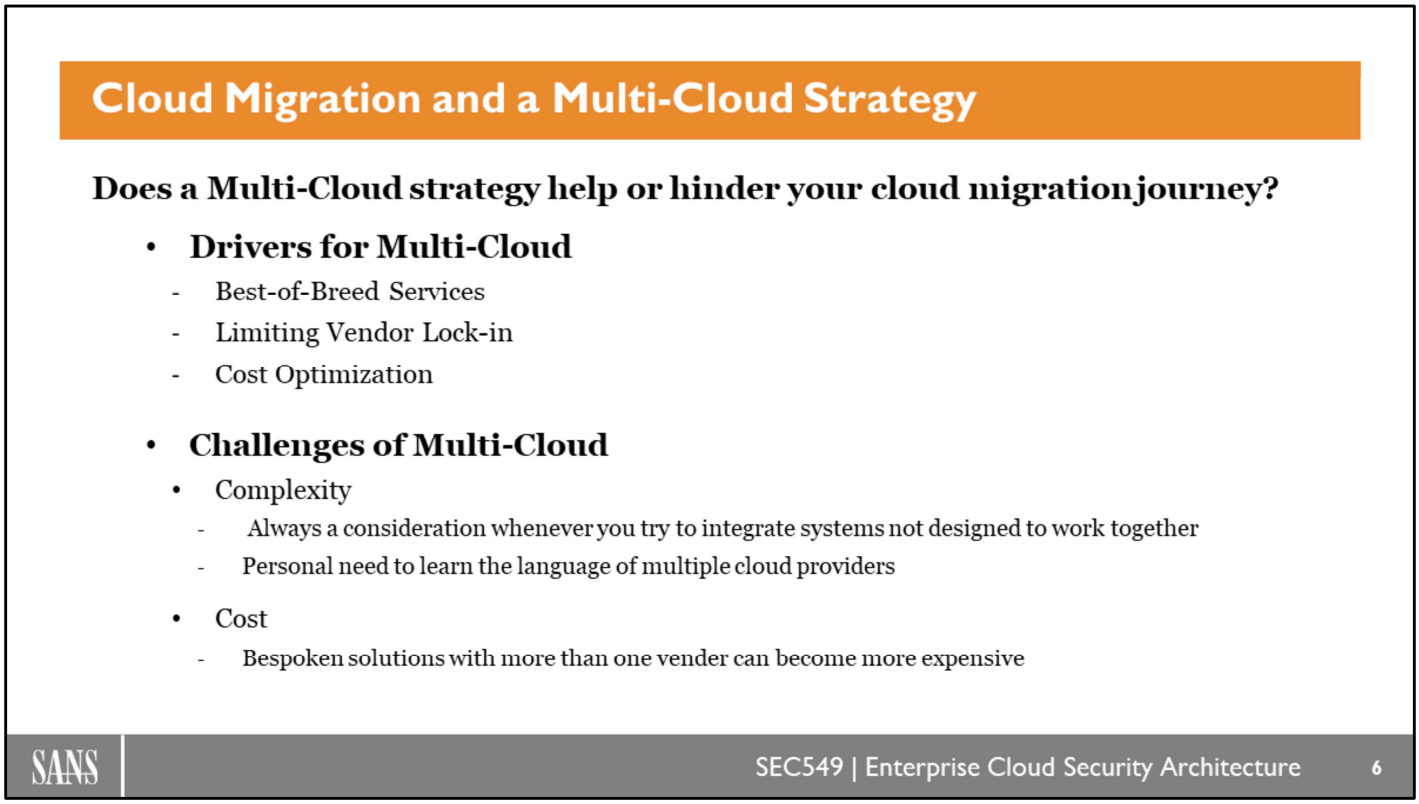


Cloud Migration Strategies
These strategies are not inclusive; neither are all strategies applicable to each type of migration. Gartner's 5 Rs involve most of the strategies now utilized:

- **Rehost –** this is also called "lift and shift". It involves IaaS and the redeployment of existing apps and data on the cloud server. It is easy enough to do by those less experienced with cloud environments and when the code is not easy to modify. Apps can be migrated relatively intact.
- **Refactor –** this is "lift and shift" with a twist. The PaaS is employed by optimizing existing apps for the cloud. The core aspects of the app's architecture are maintained, but parts of it are tweaked specifically for the Cloud.
- **Revise –** this requires more architectural and code-related changes before moving the apps to the cloud. It requires the planning and skill of a good architect who can "see" how to take a previous strategy and modify it for the cloud environment.
- **Rebuild –** this involves a more intensive approach by discarding preexisting code and replacing it. It is most commonly used by organizations who do not have viable solutions to move to the Cloud without major rewrites of what existed before the migration.
- **Replace –** this involves migrating the organization's services to a prebuilt, third-party app the vendor provides for them. The data is migrated but little else.

Other Rs have emerged as more organizations undertake cloud migrations[1]. These "less common" Rs include *Retain* (doing nothing at present for some applications), *Reuse* (creating an abstraction layer/API construct between an existing system and mobile user experience/UX, for example), *Re-architect* (modernizing a business's intellectual property/self-developed apps for cloud-based operations), and *Remediate* (a more invasive approach where the organization upgrades its operating systems, webservers, databases, and app services during the migration).

Reference:
[1]: https://sec549.com/id109



Cloud Migration and a Multi-Cloud Strategy

"Multi-cloud" simply means using two or more cloud providers and leveraging their respective advantages to suit your needs. This approach provides an alternative to relying on one cloud provider or on-premises infrastructure to handle everything. Multi-cloud refers to using more than one public cloud platform.

Does being in multiple clouds ultimately help or hurt? Or is it just more to plan and consider?

Finding Success with Multi-Cloud Migration

While multi-cloud environments can be unintentional (as when separate departments in an organization independently adopt their own shadow cloud strategies), more companies are deciding from the outset to go Multi-Cloud.

What are the potential advantages of multi-cloud migration?

- ***Improved access to specialized services*** – a multi-cloud environment means having an a la carte approach to specialized or complimentary services. Once a company sorts through what is available from various cloud providers, it has the unique opportunity to make use of what they need from more than one source.

- ***Cost is optimized*** – the savvy company can sort through competing costs to find a solution that maximizes the benefit-to-cost ratio. Companies can often get around hidden costs of egress spikes, for example, by strategizing with cloud providers having bandwidth alliances. The key here is being "savvy", as the cost might also be a hindrance, depending on the situation.

- ***Avoiding vendor lock-in*** – this is more common with the Big Three cloud providers who have large suites of services but attempt to keep them so proprietary that the temptation to use them is great. Other interoperable providers can now become more advantageous and cost-effective to use.

What about disaster recovery (DR) benefits? While it might be an attractive thought to go Multi-Cloud for its perceived DR benefits, the increased cost and complexity often do not outweigh any benefits. When looking at building fault tolerance into your cloud-hosted systems, it's best to plan for multi-region deployments for failover within the same cloud provider.

Overcoming the Challenges of Multi-Cloud Environments

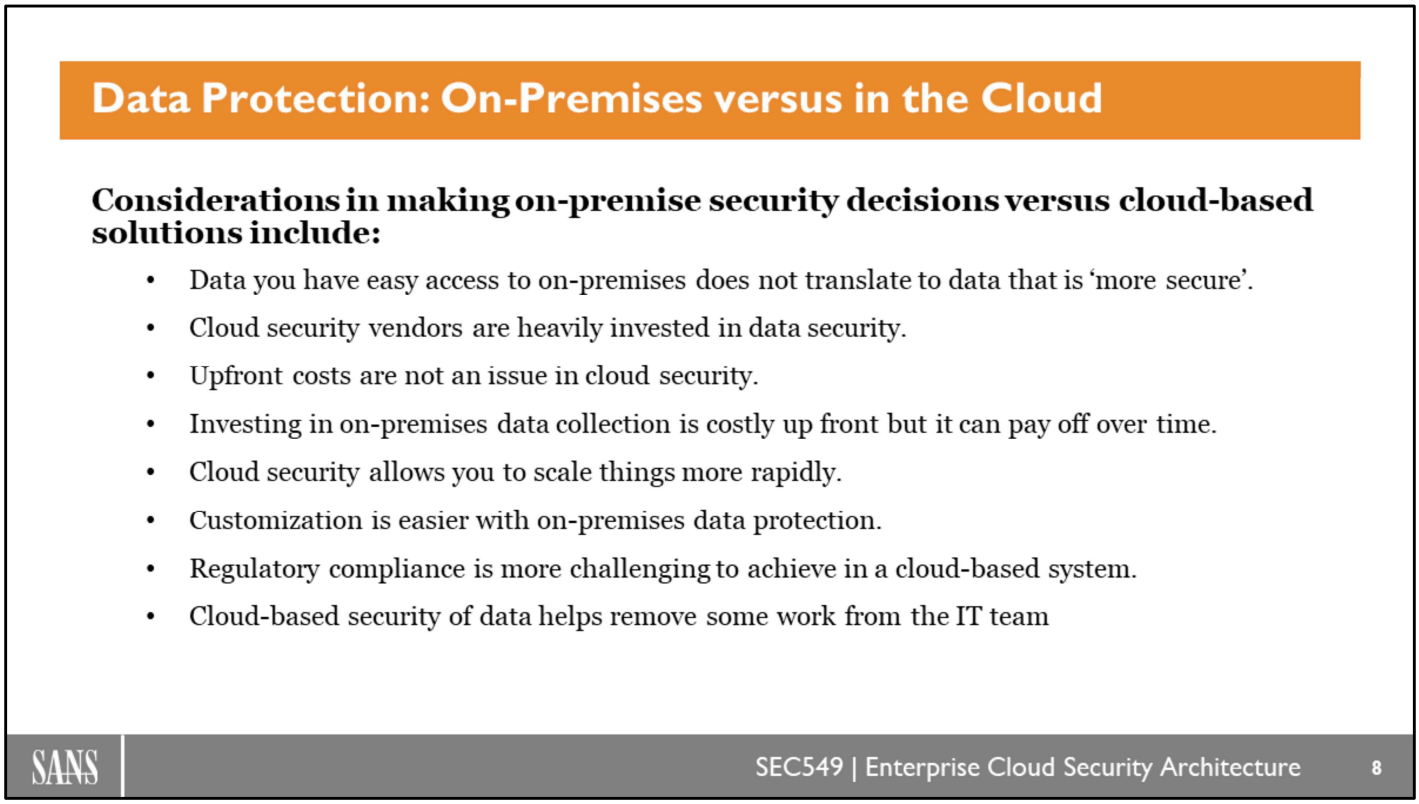
The top challenges can be overcome with multi-cloud best practices. Consider these major challenges and how you might overcome them in any organization:

- ***Deployment strategies*** – redundant deployment involves having mirrored data in more than one cloud (usually for failover or disaster recovery situations). Distributed deployment is used by teams that distribute components of their computing environment using a strategy where a mix of services are handled by different cloud providers. Each organization will need to balance the added cost of maintaining redundant cloud-hosted systems

- ***Cost management –*** cost management is an issue, even when using a single cloud. The challenge is to develop a process to attribute cloud costs to individual teams and services

- ***Data security*** – as complexity grows, so does the attack surface and security risk. This makes the centralization of identity source paramount to security in the cloud

- ***Governance –*** it is not always easy to create standards and policies for a single cloud. A multi-cloud environment might require multiple sets of policy documents bespoke to each major Cloud Service Provider



Considerations in Data Protection: On-Premises versus in the Cloud
Investments have been made for years in many organizations with on-premises data to protect. Most of these have focused on identifying network-based threats. Entire security departments exist to manage the traditional tools used to protect on-premises data.

The cloud has arrived, and organizations have seen the advantages of migrating their applications and data from an on-premises location to the Cloud. Now, organizations have a choice to make regarding data protection. Much of the choice relates to how much of the on-premises data should remain as it was or be moved via a cloud-native offering from one of the major cloud providers. Who benefits from hybrid options, for example? What are the primary considerations regarding data protection?

• Data you have easy access to on-premises does not translate to meaning the data is 'more secure'. This is only true if your IT team is heavily involved in data protection on a 24/7 basis and when common data-breach mistakes are continually addressed.

• Cloud security vendors with their own data warehouses are heavily invested in data security. Servers kept in these vendor warehouses are better protected than most on-premises warehouses as a result.

• Upfront costs are not an issue in cloud security. Security in the cloud is an operational expense involving a pay-as-you-use-it strategy. No need to invest in hardware, software, cooling equipment, or installation labor necessary with on-premises security.

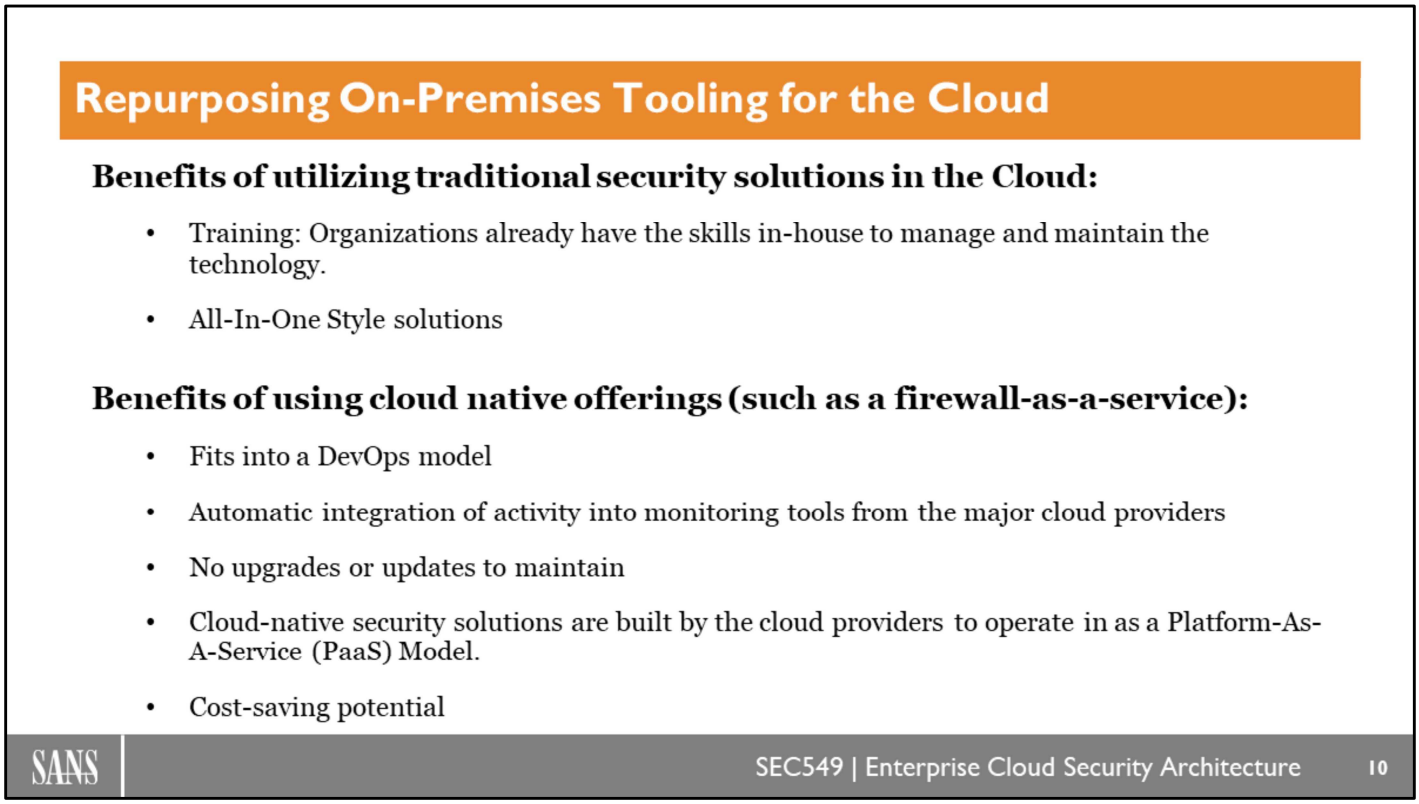
• Investing in on-premises data collection is a lot up front but it can completely pay off with the right long-term vision and strategies. The pay-as-you-go strategy is not an issue with on-premises data collection/protection, so companies wanting to avoid a subscription-like service and who have the investment cash up-front may decide to use an on-premises option.

• Cloud security allows you to scale things more rapidly and without too many unexpected high costs. Often scaling up in the cloud simply involves a subscription upgrade.

• Customization is easier with on-premises data protection. Sometimes the cloud vendor offers packages that just don't fit an organization's needs perfectly. These organizations would then benefit from a self-built system.

• Regulatory compliance is more challenging to achieve in a cloud-based system, depending on the organization type. If this is most of the data being protected, building an on-premises data protection plan is often simpler than trying the same thing using a cloud-based vendor.

• Cloud-based security of data helps remove work the IT team no longer needs to consider. This cannot be said when data protection is based on on-premises teams.



Repurposing On-Premises Tooling for the Cloud
On-premises networks have traditionally been protected using Next-Generation Firewalls, Intrusion Detection Systems (IDS), and monitoring by leveraging host-based solutions. Entire IT security departments and their skillsets are often built around operating these tools.

With the advent of cloud applications where data packets must be protected as egress or ingress traffic, can the organization count on the cloud service provider providing the necessary protection for these types of traffic patterns?

Ingress traffic can involve Internet data that can reach an organizations VPCs. Rather than having a delay involved when everything must make a hairpin turn through the data center, some can utilize AWS WAF (a web application firewall) that protects common web intrusions. ASW Shield is a Distributed Denial of Service (DDoS) service protecting web apps running on AWS proper. Deep packet inspections require a Next-Generation Firewall, such as the Palo Alto Networks VM-Series.

Egress traffic is needed for software updates or gaining access to SaaS services over the Internet. The organization can block all traffic except for trusted locations or use offerings through AWS, which provides a NAT service gateway or virtual firewall. Next-Generation firewalls offer higher levels of security. A shared security service or VPC approach may be needed, depending on the security requirements.

Both AWS and Azure have their own versions of firewall-as-a-service. Most offer the basics in intrusion prevention that may be adequate for some enterprises; however, most need to rely on a Next-Generation Firewall solution. Offerings by AWS and Azure are cost-effective, provided the fit is otherwise adequate.

Benefits of utilizing Traditional Security Solutions in the Cloud
- Most organizations already have trained personnel in-house for this type of security solution. Such individuals would manage and maintain the necessary technology.
- Traditional security offers all-In-one-style solutions

Benefits of using cloud native offerings (such as a firewall-as-a-service):
Fits into a DevOps model

- Automatic integration of activity into monitoring tools from the major cloud providers
- No upgrades or updates to maintain
- Cloud-Native security solutions are built by the cloud providers to operate in as a Platform-As-A-Service (PaaS) Model
- Cost-saving potential

Course Roadmap

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

SECTION 2

- Introduction to Cloud Migrations
  - Drivers for Cloud Migrations
- ***Implementing Zero—Trust Architecture***
  - Using Cloud Services to get to ZT
    - Lab 2.1: Integrating Auth into Legacy Application
- Establishing Perimeters for Application Access
  - Connecting VPC-Aware and Non-VPC Aware Services
    - Lab 2.2: Refactoring an Application for Hybrid Cloud
- Establishing Perimeters for Data Access
  - Managing S3 Access At Scale
    - Lab 2.3: Configuring Data Lake Access Controls

This module looks at zero-trust architecture and how cloud services can begin to operate in that environment. The history of ZT is covered, followed by a discussion of how zero-trust concepts can be approached from an architectural perspective. The lab allows students to practice integrating a legacy application into a cloud-based environment using zero-trust as one of the guiding principles.

Takeaways

In this section we will cover the history of the Zero-Trust movement and the problems this access model is trying to address.

From the perspective of cloud migration, we will discuss what things need to be considered when planning a Zero-Trust Architecture.

You will learn how modern cloud platforms like AWS are built with Zero-Trust principles in mind, as they use identity-centric controls as the basis of the security model between any two services that operate on your behalf.

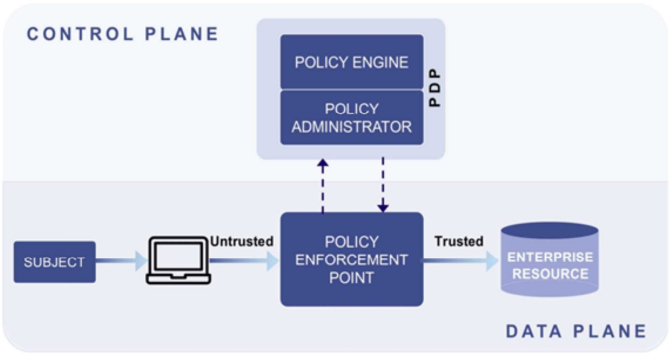
Finally, you will be introduced to AWS Cognito, which can play a role in a Zero-Trust Architecture. We will walk through the various components of Cognito, including how to use it to centralize your view of group membership and ways to leverage it to continuously authorize access to your AWS resources.

This page intentionally left blank.

What Is Zero-Trust?

Effective utilizing ZT means you –

- Prevent unauthorized access to an organization's data and services
- Create a system where access control enforcement becomes granular and sustainable
- Provide for an environment where the user's experience is not compromised
- Minimize delays in authentication processes
- Impose least privileges necessary to perform an action in any request



What Is Zero-Trust?

Zero trust (ZT) is a paradigm used in cybersecurity. It is based on the concept that trust is never implicitly granted and must always be reevaluated. ZT architecture offers an approach to enterprise security that involves authenticating the identity of users and devices, fortifying access management, credentialing, rethinking hosting environments, interconnecting infrastructures, and identifying endpoints. It begins with restricting resources only to those who need access to them and giving only the minimum privileges each user requires.

Zero trust represents a diversion from solely relying on perimeter-based security, where the risk of unauthorized lateral access is high once the perimeter is breached, and instead utilizes strategies that do not differentiate between users or devices from outside the organization and those within it. In ZT, no user or device is implicitly trusted.

ZT is the elimination of the concept of Trust within a digital system. Instead, the accretion of identity is continuously validated in order to gain confidence in the security of the system. In practical terms, Zero-Trust is a layer 7 policy that needs to be enforced at Layer 7.

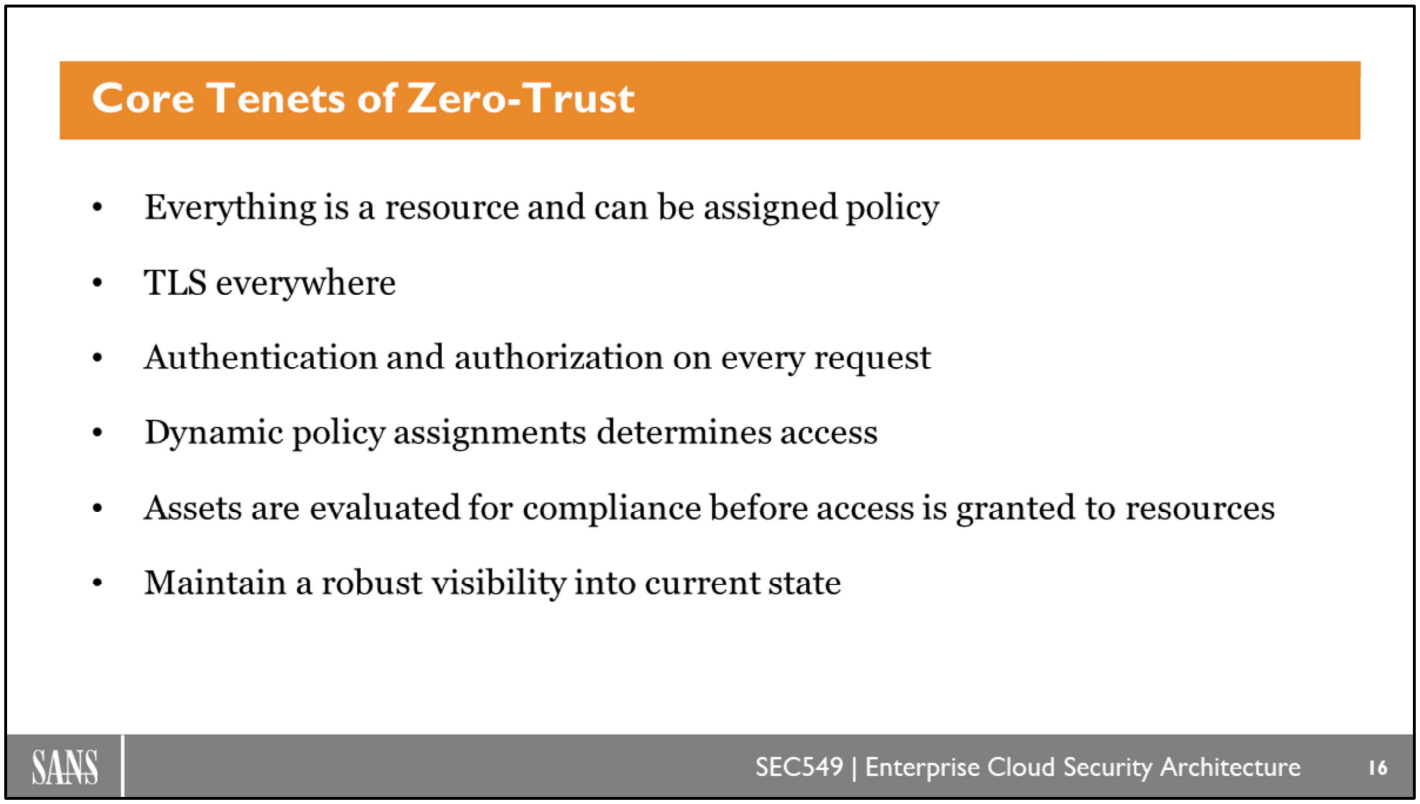
ZT must be enforced regardless of location. Because users come from everywhere and on many different devices, ZT must be visible, dynamically enforceable, and capable of working in the cloud environment. This means that it is based on a software perimeter.

When deploying ZT, the old method of "trust but verify" is abandoned, largely because *trust* was the default mechanism in this type of antiquated security system. When identity was verified, trust was assumed. Because identity credentials are easily stolen, it now becomes critical to first establish identity as an *assertion* rather than a real person. An asserted identity is inherently untrustworthy. In ZT, no user, interface, device, or packet is inherently trusted, regardless of where it comes from.

Fortunately, ZT is not nearly as expensive or complicated for organizations to utilize as many have feared. It makes use of existing technology and a few new tools along with a ZT mindset to reorient the organization's philosophy around security.

Because it represents a reorientation of the collective mindset, ZT is seen not as a destination but as a journey where one or more micro-perimeters are established around critical information and assets – one step at a time.

Image Source: NIST Special Publication 800-207 https://sec549.com/id110



Essentials of Zero-Trust

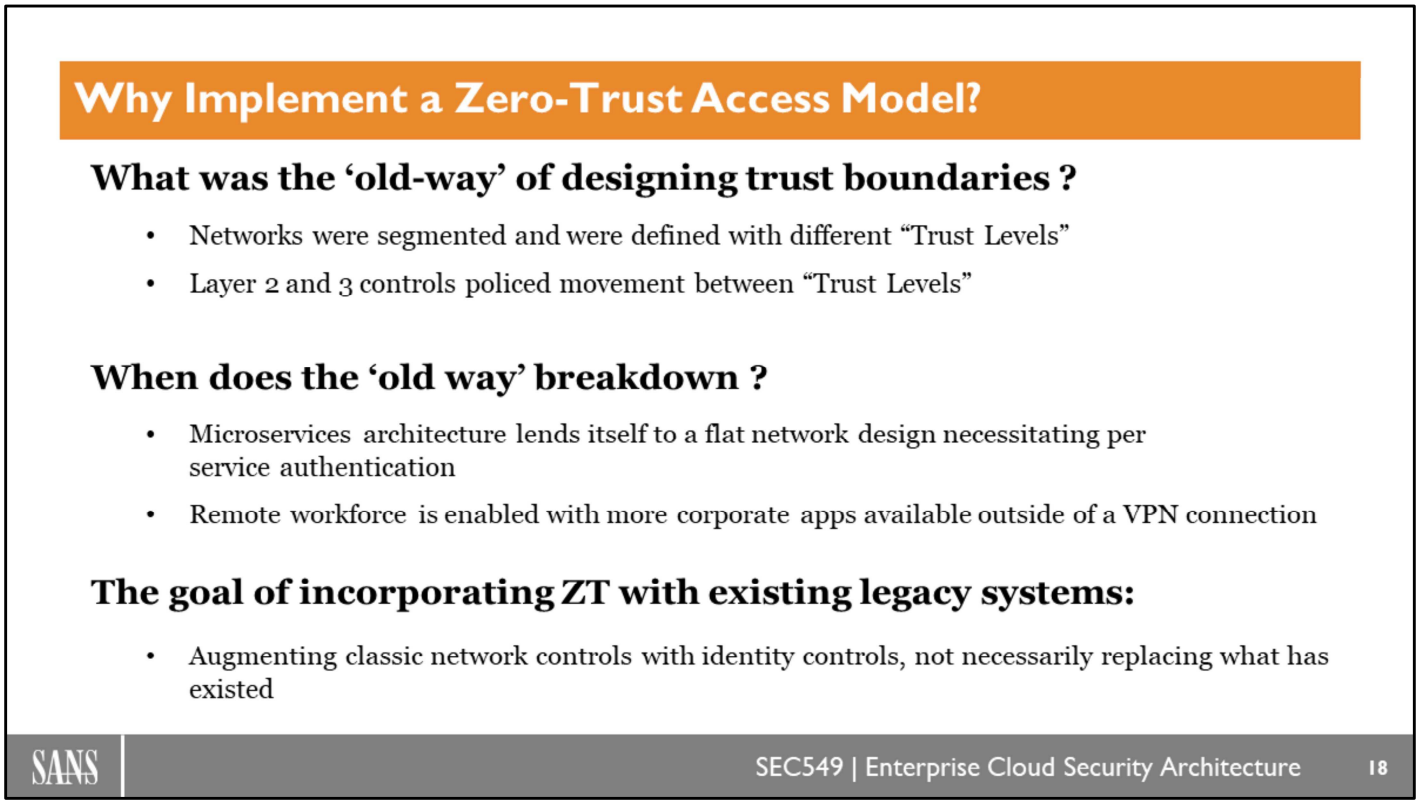
NIST[1] outlined their "tenets" of ZT but clearly indicated that these tenets were possibly an unattainable "ideal"; they recognized that some strategies could not implement all tenets. A ZT architecture is ideally designed with these tenets in mind:

1. Every data source and computing service is a resource.
   a. When everything is considered a resource, we can apply access policy to all resources
2. Encryption-In-Transit enabled at all locations of the network
   a. Every aspect of communication is secured, regardless of where data comes from in the network.
   b. Network location does not define the level of trust, which means that requests from inside a legacy network perimeter must meet the same security requests as those from the internet.
3. Per-Request, Per-Session Resource Authorization
   a. Access to individual resources is granted on a per-session basis.
   b. Trust is first evaluated before access is granted; access is granted with the least privileges needed for task completion. Authentication with allowed access to one resource does not mean access to a different resource is automatically granted.
4. Access to all resources is determined solely by dynamic policy.
   a. Aspects of policy includes the user's identity, the app or service, and the requesting asset.
   b. A policy can also depend on the behavioral and/or environmental attributes of the user.
5. The enterprise utilizes automation to continuously monitor the integrity and security posture of all assets.
   a. No asset is inherently trusted.
   b. The security posture of all assets is evaluated when evaluating a request for a resource.
6. All resource authentication and authorization processes are dynamic and are strictly enforced **before** access is allowed.
   a. IAM and asset management systems remain in place at all times with a continual cycle involving access attainment, threat scanning and assessment, adaptation, and reevaluation.
   b. This includes the use of multifactor authentication (MFA) for access to some or all enterprise resources.

7. There is maintenance of a Robust Visibility into 'Current State', **used to improve its security posture.**
    a. An enterprise should collect as much information as possible about network infrastructure, the current state of the organization's assets, and the communications within the network to manage and improve the organization's security posture.
    b. Other information to collect includes network traffic, access request patterns, and asset security to improve the creation and enforcement of policy.

NIST admitted that these tenets were ideals and that they were "technology agnostic", meaning that things like credentials could mean username and password, onetime passwords, or certificates.

Reference:
[1]: https://sec549.com/id111



What was the 'Old Way'?
The Old Trust Model had people defining "Trust Levels" of different systems and applying policy for crossing the boundary from one Trust Level to another. Those boundaries were often gated with layer 2 and 3 controls. When those control inevitably failed, the entire Trust Model collapsed.

In the beginning…

The outermost edge was protected from the "inside". A secure network within the premises was established and users were invited to connect to it.

Later, worms and viruses made it clear that the network itself must be secured internally. Firewalls were created to maintain the ability of desired traffic to travel in and out of the network, while enhancing network security. The balance was to be 'secure' and 'efficient' at the same time. Unfortunately, this desire for balance was easily exploitable.

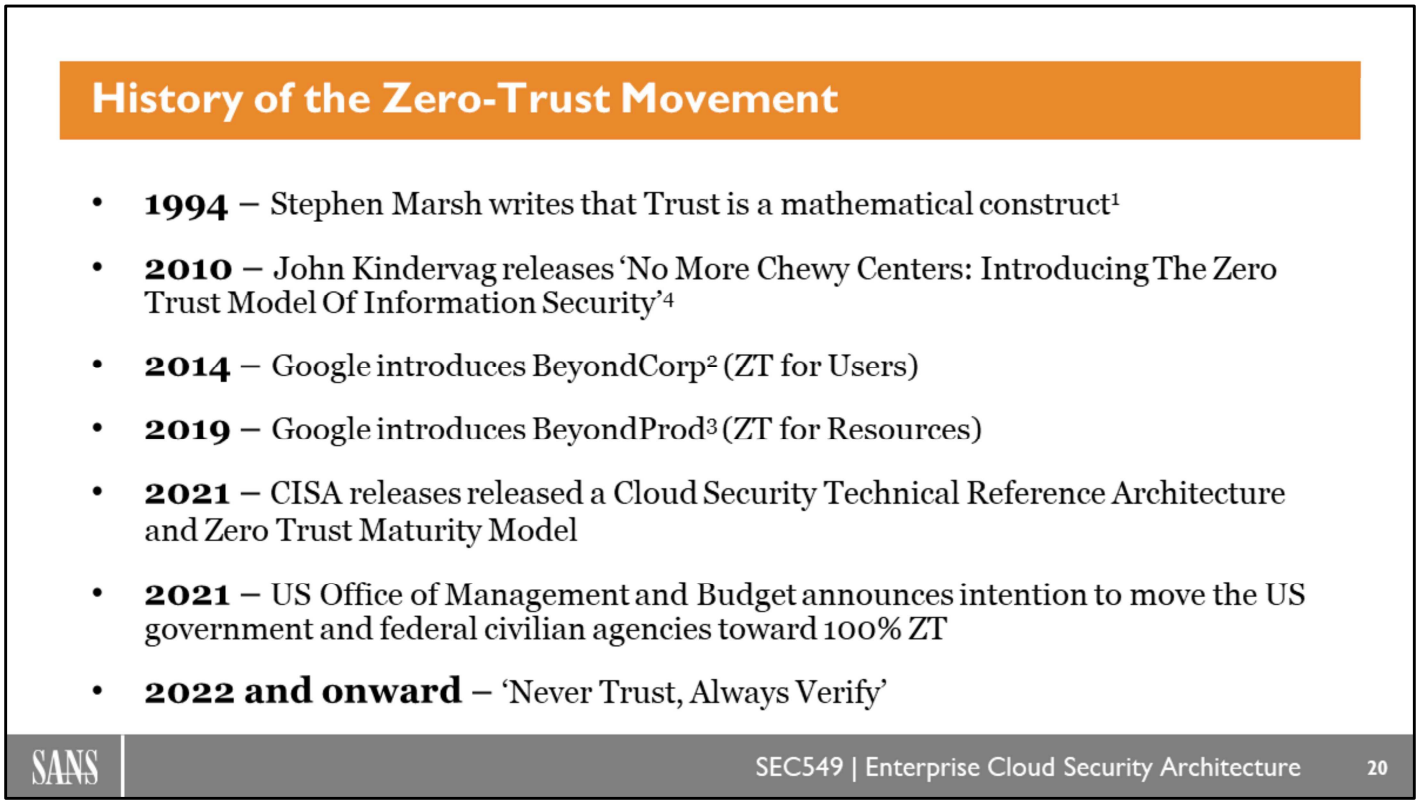
The problem with the 'Old Way'? The problems with trust have been well documented going all the way back to Ken Thompson's speech from 1984, "Reflections on Trusting Trust". It's a thesis about how the concept of Trust incentivizes bad behavior. All major security incidents boil down to an abuse of the concept of digital trust. Trust is a very human concept that should have never been applied to digital systems.

Twentieth-century security involved levels of trust – hierarchical models where (in general), users outside of the system were not 'trusted', while users were more 'trusted' in stepwise fashion. This repeatedly fell apart when the layer 2 and 3 controls were breached. After several spectacular internal attacks were uncovered in the first few years of the Twenty-first Century, it became increasingly clear that hierarchical models were not viable ways to approach security.

ZT prevents lateral threat sprawl through micro-segmentation and the development of granular perimeters of enforcement (particularly of sensitive data). All users and entities are treated the same and none are inherently trusted.

ZT designs a network from the inside out by protecting the most sensitive data and assets, and then designing around those aspects of the total digital system. This type of protection avoids large areas of exposure of any part of the organization's digital footprint.

Yet, few organizations have the luxury of scratching every legacy system they have and starting over in an entirely new ZT framework. The movement to deploy a ZT model starts with identifying which legacy systems can and cannot (yet) be deployed in a ZT environment.



History of the Zero-Trust Movement

The first individual to talk about trust in digital systems was Stephen Marsh, who wrote in his doctoral thesis in 1994 about trust as a mathematical construct and not simply a human factor related to ethics, justice, or morality. He believed that ZT principles provided a more optimal way of approaching security design and practices.

John Kindervag made the term "zero-trust" more popular in 2010, but it was not implemented in most organizations' IT standards for several years. Kindervag spoke then of the security industry's collective mistaken belief that having a security model like an M & M (with a hard outer shell and soft, chewy interior) simply rewarded would-be data thieves who only had to get through the organization's outer exterior or firewall to gain access to a large repository of potentially sensitive, unsecured information.

He indicated that remote work and cloud adoption have more recently necessitated serious reconfiguration of the security policies and practices of organizations all over the world. Some have responded to his early concerns, while others have not. Consequently, there are few companies today that can safely say that a secure perimeter is all they need to protect their assets. Those that could say that a few years ago now are scrambling to adopt ZT to their increasingly hybrid operations.

Kindervag also talked about the problems that naturally arise out of older trust models. The most obvious one is that, once an attacker breached a perimeter through a compromised identity, the impact to resources and information was often far reaching. He highlighted the "The Philip Cummings Problem", which involved an individual working for TeleData Communications in 1999-2000. Cummings access to data was excessive and broad. As a highly privileged employee he was bribed by a Nigerian crime syndicate to provide sensitive information about millions of customers. After leaving the company, the exfiltration of data continued and remained undetected for additional 2 years because Cummings had configured a laptop residing outside the organization's on-premises network to continue a "low and slow" data breach.

In addition, under older security models, valid identities were commonly over-privileged as there were often instances where that user had access to resources they did not need. It meant that someone once "trusted" in an organization (like Philip Cummings) who later became compromised (for whatever reason), their access to potentially sensitive data was unchecked and easily undetected. It became increasingly clear that initial compromises rarely occurred at the chosen target but instead began at softer, innocuous target, moving laterally in order to gain access to their intended target.

History has shown us some spectacular breaches exploiting the old trust models that now seem a lot like "gullibility models". It has become no longer viable to have a single large perimeter as the singular means to protect an organization's assets. ZT offers a paradigm different from the old access model, reducing the chances of a single point of failure.

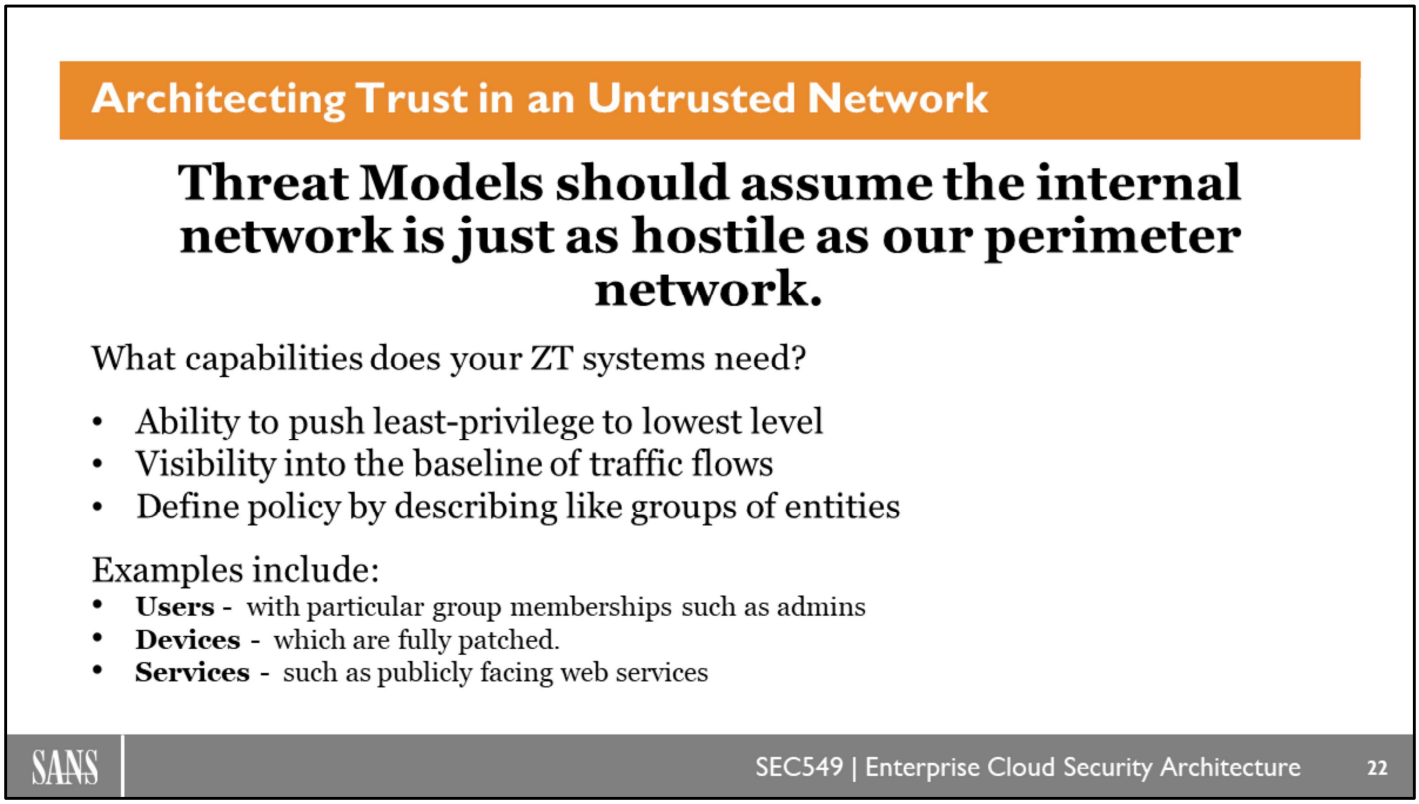
The recent September 2021 press release by the White House, while barely noted by the general public, sent ripples through the world of cybersecurity. CISA (The Cybersecurity Infrastructure Security Agency), in conjunction with the Office of Management and Budget, announced the intention to move the US Government and all federal civilian agencies toward 100% ZT Architecture over the next few years. For these agencies, this means that the option to "opt out" of ZT architecture is no longer valid. It also means that the scramble is on for all enterprise security architects to become savvy about what ZT really means for them and their agencies.

CISA also released a Cloud Security Technical Reference Architecture and Zero Trust Maturity Model to help assist agencies in implementing ZT. The major pieces of this strategy include consolidation of identity systems, utilizing MFA, encrypting data traffic, strengthening app security, and above all, assuming that all internal networks are untrusted. Fully implementing CISA's Maturity Model for all agencies is expected to take several years, but the roadmap to get there was provided, along with the admonishment, "Never trust, always verify."

Chris DeRusha, Federal Chief Information Security Officer, indicated that the roadmap was only a beginning and that outside assistance from security experts was welcomed and would contribute to what the completed plan would look like: "The federal government's approach to cybersecurity must rapidly evolve to keep pace with our adversaries and moving toward zero trust principles is the road we need to travel to get there. Today we're releasing a draft federal zero trust strategy that will help agencies put these principles into practice. While we feel the urgency to begin implementing this plan, we know that input from the broader community of experts will help ensure it is the right plan. We welcome feedback on how we can refine this strategy to best advance federal cybersecurity."

The outcome of CISA's ZT Maturity Model has yet to be defined but for many security architects, the race is on to secure their agencies using ZT principles. Some do not realize, however, that the US government is a bit late to the ZT party. More than a few enterprise cloud security architects were listening to early pioneers in ZT and have already incorporated ZT into their organization's security strategies.

References:
[1]: https://sec549.com/id112
[2]: https://sec549.com/id113
[3]: https://sec549.com/id114
[4]: https://sec549.com/id115



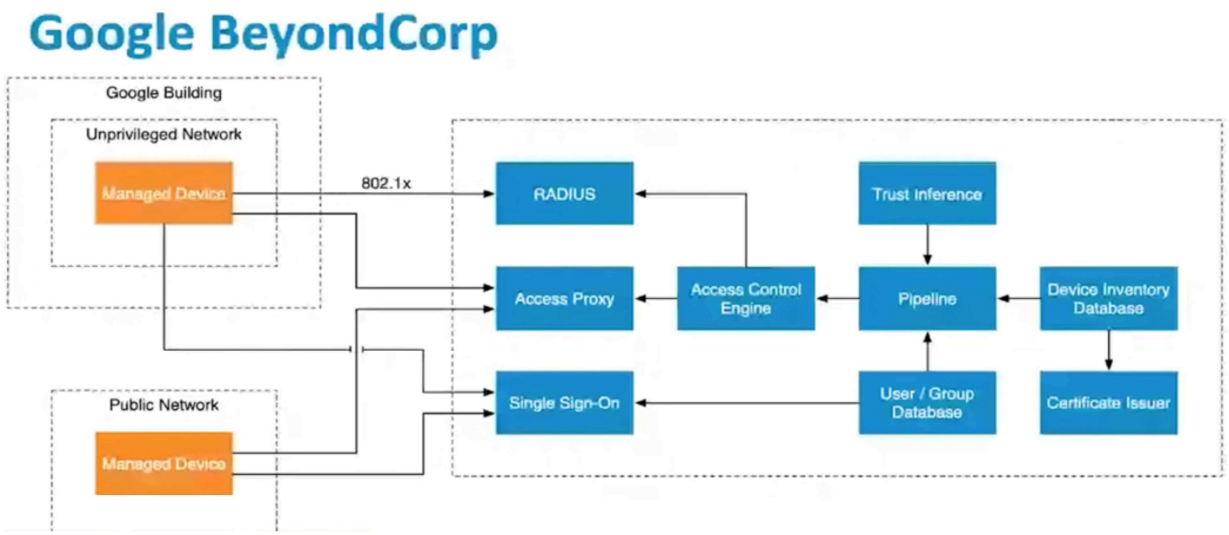
Architecting Trust in an Inherently Untrusted Network
In the subsequent slides, we will take a look at published patterns from the likes of Google and PagerDuty which show case how they designed Zero-Trust for both an end user and for service to service (east-west) traffic.

Commonality among ZT design patterns
You will see the following themes shine through in both the upcoming Google and PagerDuty case studies.

1. The ability to enforce least privilege to the lowest level possible is required – This often manifests as a policy enforcement engine which should be separate from policy definition. It is through good policy definition which policy enforcement can be calculated.
2. An accurate account of traffic flows is needed– All communication in a ZT system is expected and accounted for, as such knowing who is communicating with who is imperative in order to craft effective policy.
3. Policy needs to be attached to the workload – Policy should not be described in the form of network primitives (IP address or network address ranges), but rather should describe symbolic types of workloads. For example, you might reference 'all web servers' or 'all app servers' in policy but you wouldn't define policy against a specific IP address.

Each of these design facets can help systems adhere to the ZT principal that all internal networks are just as hostile as external or perimeter networks. The resulting ZT architectures are often considered '*perimeterless*' where trust is not established at a central choke point for large swaths of IP space, but rather is enforced on a per service or per application basis.

BeyondCorp | Zero-Trust for the User



Source: **Using SABSA to Architect Zero Trust Networks**[2]

BeyondCorp

The BeyondCorp white paper[1] was written and released by Google in 2014. In the paper, first submitted to the USENIX conference, Google described that when they migrated their corporate applications to the Cloud, their processes and guardrails were strained. In sharing some of their lessons learned, they emphasized that even those corporations having a segmented internal network needed to rethink the perception of safety in those types of networks as well. If there were internal unrecognized bad actors in the organization, the implementation of a strong exterior firewall was essentially useless.

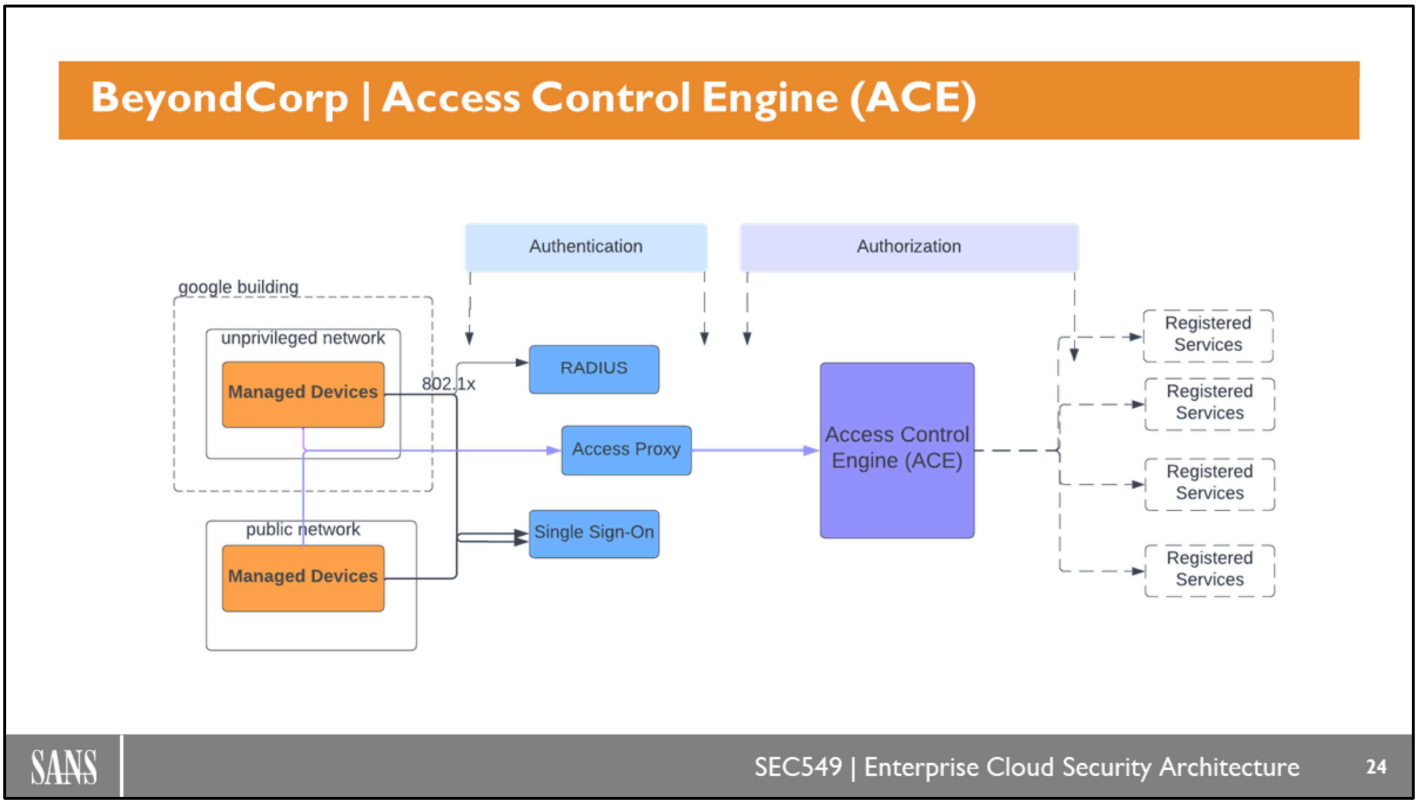
Key components built into BeyondCorp framework at Google included making use of managed devices to access corporate apps, having context-aware asset management, and using machine certificates for device identification. Such certificates were used to augment authentication of users but not as a mechanism to convey user permissions.

Authentication in BeyondCorp

How did BeyondCorp authenticate users? They enforced a particular kind of multi-factor authentication for all requests from managed corporate assets by validating both the users SSO Tokens and their embedded device certificates. By using details of the device as a second factor, the BeyondCorp Architecture achieved a type of 'Transparent MFA', gaining all the benefits of multiple authentication factors while reducing friction on their workforce.

A typical end user authentication begins when they request a backend service. Requests for all services are directed to the access proxy where their managed device submits its device certificate. If the certificate is not recognized by the proxy, the end user is redirected to the centralized identity provider (IdP) where they are prompted to provide their user credentials, including any second-factors. Upon successful authentication with the identity provider (IdP), the user is issued a SSO token and redirected back to the access proxy where they are authenticated with the IdP provided token.

References:
[1]: https://sec549.com/id116
[2]: https://sec549.com/id191



Authorization in BeyondCorp

At the heart of authorization and policy enforcement in the BeyondCorp pattern is the Access Control Engine (ACE).

BeyondCorp was designed to be deployed on an unprivileged network using Google's Access Control Engine (ACE) to guide who had access to what apps, data, and resources. The ACE was designed to be interfaced with via an internet-facing Access Proxy for both internal and external users and clients alike. Only when the access control checks are completed will the user requests be considered appropriate to the back-end application.

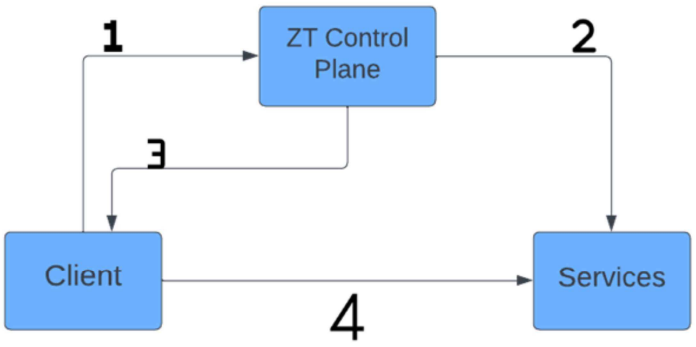
The ACE provided service-level authorization on a per-request basis. Knowing that an individual's need for access can change, they were able to use the user's dynamic level of trust to help the Access Control Engine make decisions for each request made. The ACE can enforce location-based access control, if needed, and can offer more fine-grained control over what "access" means to different users. Armed with this access knowledge, Google was able to make all their enterprise apps externally exposed and registered in public DNS.

In addition to making smart access-control decisions, BeyondCorp allowed Google to actively down-scope individual user permissions. By observing user requests to enterprise applications and looking at job functions in the company, they could segregate users of the network into finance workers, salesforce, legal, team, and engineering team members. Comparing real-time access and job roles, Google was able to dynamically reduce a user's permissions to only what they were using. They discouraged the use of the VPN, restricting access only to those with a proven need. The VPN was monitored and those who did not use it for a predefined time were removed from the access roles.

The BeyondCorp paper provides a roadmap, albeit an idealist one, for a practical implementation of Zero-Trust concepts for an enterprise user base.

The PagerDuty Approach to Zero Trust

1. Client requests a service
2. Control-plane dynamically configures service to allow access
3. Control-plane issues a one-time use token
4. Client accesses the service directly, authenticating with the control-plane provided token



Zero-Trust Control Plane

The Zero-Trust control plane component was a concept introduced in the PagerDuty Zero-Trust architecture and described at length in the book: *Zero-Trust Networks: Building Secure Systems in Untrusted Networks*[1]. In that book, the control plane component is described as "…an authoritative source, or trusted third party, is granted the ability to authenticate, authorize, and coordinate access in real time, based on a variety of inputs.".

The Zero-Trust control plane has the responsibility of:

- Communicating with the trust engine and determine if a request is authorized
- Dynamically configuring services to accept authorized traffic
- Coordinate the establishment of encrypted tunnels between the client and service

Service-Level Authentication

Let's review how an end user successfully accesses a service in the PagerDuty ZT model.

1. Client makes request to access a service – with this request comes along the user data, device data, and activity data which is fed into the trust engine to determine authorization context
2. Control plane intercepts request from client to service – determines if they are allowed to make the request given policy and dynamically configures the service to allow access
3. Control plane generates a one-time use access token – this token is passed back to the client and allows the client to access the requested service
4. Client accesses the service directly – their request is authorized with the control-plane provided access token

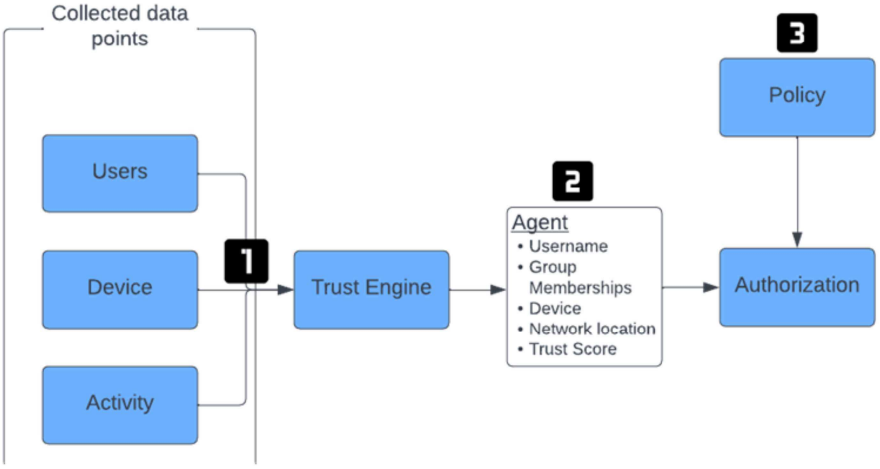
You'll notice the key difference between the ZT control plane and Googles' access proxy is where it sits in relation to the backend services. In the Google BeyondCorp pattern, the access proxy sits inline, and end users do not communicate with services directly. In contrast, the PagerDuty pattern has the concept of a control plane that negotiates access, but users ultimately access services directly.

Reference:

[1]: https://sec549.com/id192

PagerDuty | Trust Engine

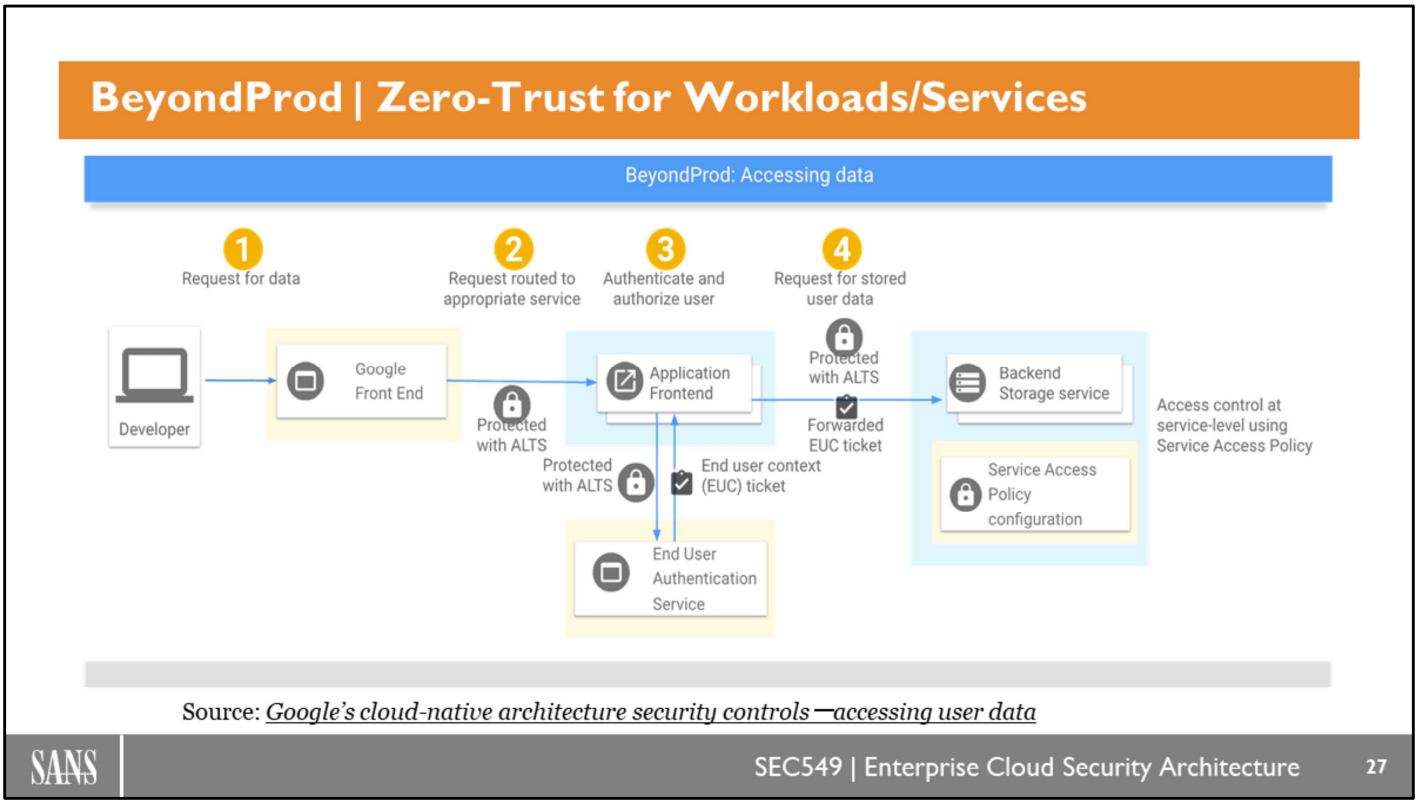
1. Details of a request are collected and processed by a Trust Engine
2. The Trust Engine creates an ephemeral 'Agent' and scores its trust level
3. Policy is applied to the Agent given its details



Trust Engine
Similar to Google's BeyondCorp Access Control Engine (ACE), the Trust Engine is core to authorization decisions in the Pager Duty ZT pattern. In the book: *Zero-Trust Networks: Building Secure Systems in Untrusted Networks*[1], it is described as a component which "…..calculates a score and forms an agent, which is then compared against policy in order to authorize a request."

The Trust Engine is the system which marries the user context and device context of the incoming request to form a Network Agent, to which policy decisions can be applied. Key to this design is the separation of policy definition from the policy enforcement process, which is the responsibility of the control plane.

Reference:
[1]: https://sec549.com/id192



BeyondProd – Zero-Trust for Workloads/Services

Five years after its technical paper 'BeyondCorp' was released, Google released a follow-up technical paper, outlining how to manage its cloud-native architecture with a cloud-native security strategy called BeyondProd[1]. Where BeyondCorp was all about defining a new access model for users, BeyondProd outlined how to rethink access to the microservice. By this time, it had become clear that the perimeter-based security models were insufficient – not just for user access but for service to service access as well. BeyondProd became a way for developers and security professionals to design cloud-native security controls for their own production networks, just as Google did when it transformed its applications and moved away from large, monolithic apps to distributed microservices.

In order to understand the need for BeyondProd, we have to discuss the Cloud-Native movement as a whole. This is represented most concisely as microservice architecture. By isolating aspects of a larger service or application, any potential damage to one piece of the whole would not necessarily affect any other piece. These were deployed as containerized applications and utilized a system of orchestration that could allow for scalability and easier manageability. The perimeter-security model that had already been established as insufficient for user access was certainly that for systems built in the style of cloud-native architecture.

Google's now-containerized infrastructure was designed with built-in redundancy so that small failures or breaches never disabled much, if any, of the whole. The containers were managed using an internal container orchestration system they called Borg, the predecessor to Kubernetes – capable of deploying billions of containers weekly.

In the BeyondProd technical paper, Google indicated that utilizing cloud-native security meant that services would necessarily need to be treated in similar ways that users were managed in BeyondCorp, particularly knowing that microservices may be run anywhere – in public clouds, within a firewalled datacenter, or from within private clouds, and will need the same level of security wherever deployed.

Microservices are now so dynamically deployed in such heterogeneous hosts that trust in a service must be dependent on characteristics, such as service identity and code provenance, rather than on the IP or hostname identity indicating a specific location on the production network. The shift means that large monolithic apps that are long-lived but difficult to update or patch instead have become collections of interrelated immutable containers that are deployed more frequently, scalable, rebuilt when needed without redesigning everything, reused, shared, and standardized, so that development is consistent and uniform between different teams independently developing pieces of a larger process.

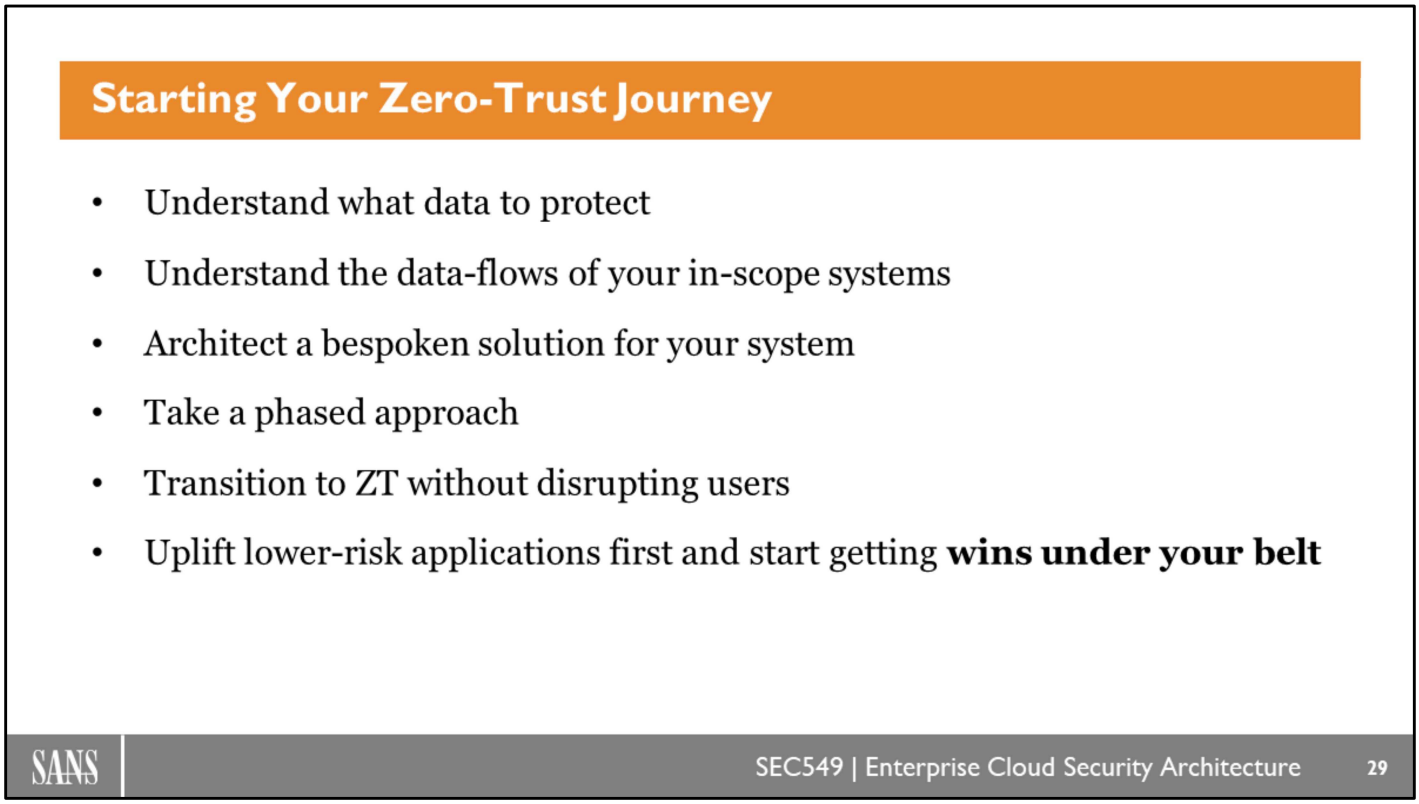
What does this cloud-native security translate to, according to the BeyondProd white paper?

- Zero-trust principles exist between services regardless of environment.
- Service-Level trust takes precedence when IPs and hardware are shared resources.
- Policies must be enforced consistently across services with shared security requirements and are pre-configured into service stacks.
- All services have the same security requirements. Security policies are centralized and centrally adhered to.

In Google's infrastructure, as little security as possible depends on human factors. While services are considered secure by default, human actions are not. Services are authenticated on the basis of the code and configuration deployed and not on the individuals who deployed the service.

ZT models mean internal traffic needs authentication and encryption, just as it would be necessary for external traffic. Microservice-level segmentation means you do not have to make the distinction between internal and external traffic; these are treated in the same way. A perimeter approach is still utilized to reduce the internet-based vulnerability of the operation's infrastructure but, without mutual trust between services, only specifically authorized callers can access any service. The distrust is mutual so, even in the event of a compromise, the blast radius remains very small.

Reference:
[1]: https://sec549.com/id117



A Brief Look at CISA's Five Pillars of ZT1 and Migrating from a 'Traditional' to an 'Optimal' Zero-Trust Model

CISA's document on ZT Migration (released in June 2021) identified five pillars of ZT and the approach organizations should have regarding each pillar when adopting ZT. They stressed that this type of adoption would not practically be 'optimal' for most organizations for several years. Still, as a guide for beginning this journey, they recommended these initial steps for adopting a ZT approach:

1. **Identity** – accept only least privileged access by first identifying all entities desiring access. An advanced ZT model eliminates usernames and passwords as the only way to validate identity. MFA is a good beginning, moving toward continuous verification of identity throughout their interactions with the organization's services and/or data. Optimal ZT translates to 'continuous validation' and 'real-time machine learning analytics' throughout a user's interaction with the organization.

2. **Device** – devices include hardware, IoT devices, laptops, servers, mobile phones, and anything else with connectivity to the cloud. Recognize that BYOD is inevitable. This means that all devices within the organization are secured and that devices inside and outside the organization need to be securely identified, authorized, and monitored throughout their interaction.

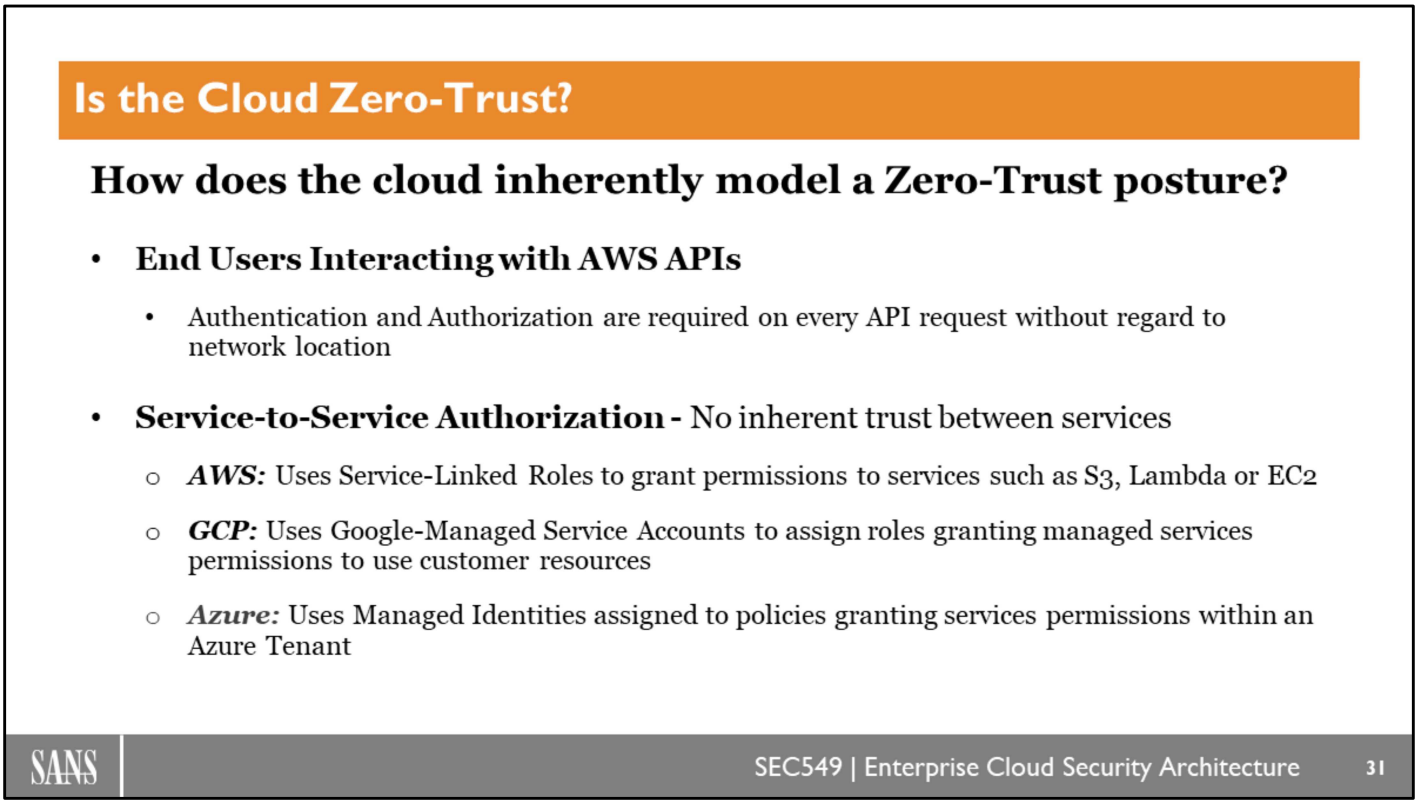
3. **Network/environment** – traditional network segmentation or 'macro-segmentation' gradually becomes 'micro-segmentation' and encryption of data flow, regardless of its origin or destination. This will allow apps and services within the organization to be capable of being directly and securely accessed from a remote location.

4. **Application workload** – newly developed and deployed apps should already utilize ZT principles with integrated security testing and verification at each level of the deployment process. This is inherent in any phased approach to cloud-native security architecture.

5. **Data** – migrate the organization toward a 'data-centric' approach by first identifying the assets and the protect surface. Once ZT 'feels' natural and practiced by the security architect, the goal then is to move toward data protection of high value assets as a priority. Ultimately, all data is encrypted in an optimal ZT environment.

John Kindervag's advice to newcomers to ZT was excellent: Practice on something you don't mind messing up on before then turning your skills toward the protection of those things that need ZT the most, using the granular application of micro-perimeters. After that, the attention can broaden out to make sure that workflow is optimized in an environment now safer and with a minimum of inconvenience for all users.

Reference:
[1]: https://sec549.com/id118



Is Your Organization Zero-Trust Simply by Migrating to the Cloud?
The simple answer is, 'no'. The cloud is built with ZT principles in mind, but organizations who use the Cloud still need to do their part to utilize what is available to them from the cloud provider and avoid setting themselves up to become the weakest link in a system where ZT is generally built into the operation of the cloud.

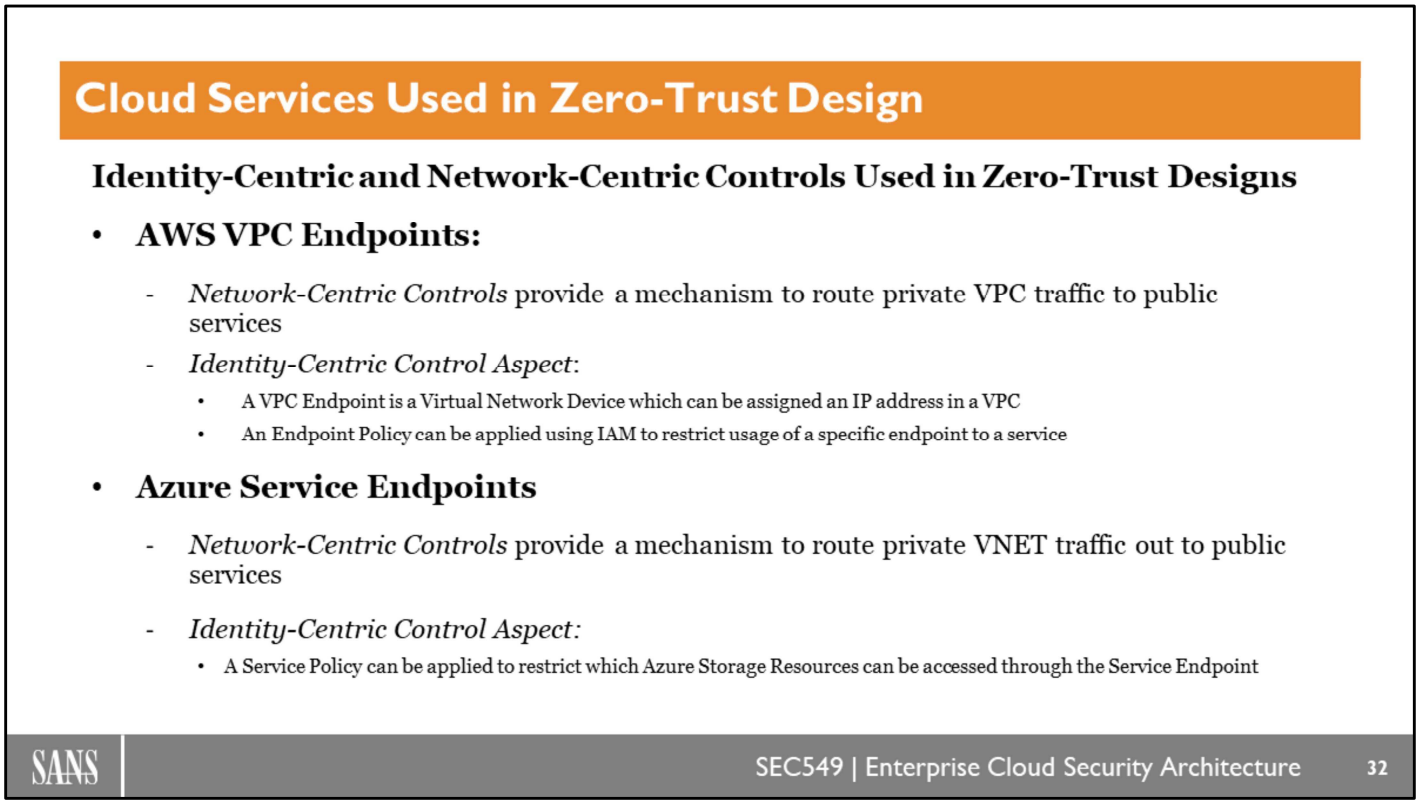
Each public cloud builds in ZT in several ways:
1. User authentication is ubiquitous. By making all entities a resource (users, policies, VMs, and storage), the cloud providers allow everything to be programmatically accessed, but only after authentication and authorization. Each and every request by end users is first authenticated and authorized, regardless of where it comes from on the network. Only then can a user receive information on a resource or use the resource in some way.

> Example: AWS requires all API requests to be signed on via the Signature Version 4 sign-on process.

2. The major cloud providers (AWS, Azure, GCP) enforce the same processes of authentication and authorization using identity-centric controls for everything. This means that mundane service-to-service interactions and back-end operations abide by the same rules as end users accessing applications. All services must be assigned permissions and be authorized first before interacting with other cloud components.

> Example: Azure services automatically receive managed identities in Azure Active Directory. When a User- or System-assigned Identity is created, the MRSP (Managed Identity Resource Provider) issues an internal certificate to the entity that becomes their 'ticket' going forward. The lifecycle of the identity depends on how it is created, with system-assigned identities being automatically deleted by Azure following the service instance.

In reality: Migrating to the cloud does not mean you are automatically 'zero-trust'. The clouds have built ZT into their public cloud operations; however, it is the customer's ongoing responsibility to use the tools offered by the cloud providers to build systems that also have these ZT principles in mind.



Cloud Services Assist You in Your ZT Journey
What tools can they provide? Both Azure and AWS, for example, allow you to utilize both identity-centric and network-centric controls to fine-tune your design, allowing you to simultaneously make use of identity and network capabilities. This ability optimizes an organization's ZT journey in the cloud. Remember that one of AWS's guiding principles of ZT is "Where possible, use identity and network capabilities together."

Two excellent examples:

- **AWS allows you to utilize AWS VPC Endpoints1.** VPC endpoints are virtual network devices you can attach to any VPC. These endpoints are used to route traffic from a private network to AWS services like S3. The identity-centric control comes into play when you attach an IAM policy to it to restrict who has access to a given resource. If you attach an endpoint policy[2] to your VPC for S3, for example, you can now restrict usage of the endpoint to specific S3 buckets.

- **Azure allows you to use Azure Service Endpoint Policy, which is identity-centric.** Service endpoints use endpoint policies that let you specify which Azure Storage accounts can receive egress traffic, restricting egress access to a specific Azure Storage account and not any other Azure Storage accounts. This allows you more granular security control over which data leaves your virtual network.

Finally, there is AWS Cognito, which is what we will cover in depth in this course. AWS Cognito is a great fit for introducing Zero-Trust because it can be easily added to individual services. This makes adding ZT incrementally easy, and also means that any code changes you make at an application level can be done with well-supported drop-in libraries.

References:
[1]: https://sec549.com/id119
[2]: https://sec549.com/id120
[3]: https://sec549.com/id121

This page is left intentionally blank.

What Is AWS Cognito?

AWS service that helps simplify end user account management for web and mobile applications

- Applications are required to integrate Amazon Amplify client libraries into their code base to take advantage of this service

Authenticating application end users

- Provides simple sign-up and sign-in capabilities
- Applications can use AWS Cognito to connect to user directory backends (User Pools)

Authorizing application end users

- Use AWS Cognito to issue AWS Temporary Credentials to users, giving them access to resources
- The authorization capabilities of Cognito are rolled into the Identity Pools feature, recently renamed 'Federated Identities'

What Is AWS Cognito?

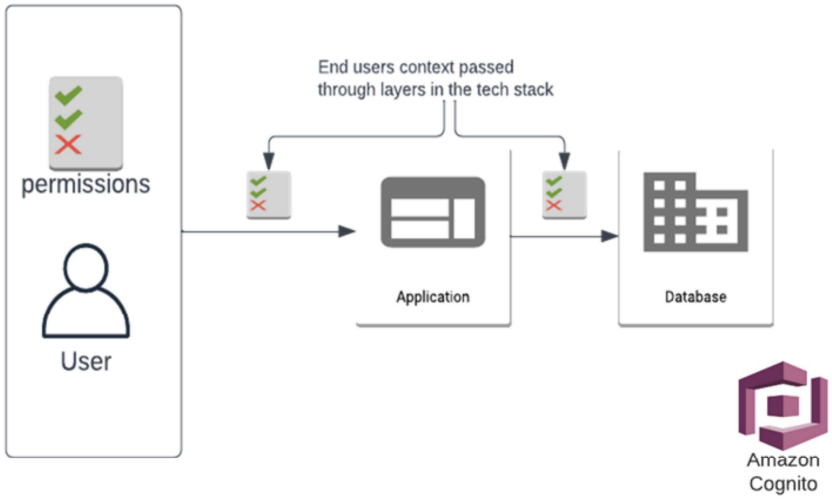
Amazon Cognito is a managed service for authentication management. It provides secure authentication, authorization, and user management for your web and mobile apps. This allows users to sign on in several ways: username and password or third-party 'social sign-ins' options like Facebook, Google, Amazon, or Apple. You can federate AWS Cognito and delegate authentication to social media authentication engines like Google, Facebook, and Amazon, or by utilizing a Security Assertion Markup Language (SAML) compliant Identity Provider like Azure Active Directory (AAD).

Amazon Cognito consists of two products, user pools and identity pools (discussed later). User pools involve user directories for sign-up/sign-in options for app users. Identity pools are designed to fulfill authorization needs by granting users access to other AWS services. User Pools and Identity Pools can be used together or separately.

Using AWS Cognito to Support Zero-Trust

AWS Cognito as a way to:

- Credential end users
- Continuously validate end user's authenticated state
- Assign privileges and permissions to end users
- Ensure the end users context is carried through



SANS | SEC549 | Enterprise Cloud Security Architecture 35

Using AWS Cognito To Support Zero-Trust

Most enterprises will have dozens if not hundreds of web applications that were built on traditional three-tier web architectures running in on-premises data centers.

Contrary to a Zero-Trust architecture, these systems often have implemented their own user login features and store user login credentials in the database, dropping the end user's context at the first layer of the tech stack.

Zero-Trust designs require authentication and authorization upon every resource request, irrespective of the end user's network location. Uplifting a fleet of legacy applications to enforce these patterns can be such a daunting task that the effort is descoped or takes years to accomplish.

Here is where cloud services like AWS Cognito can help in augmenting incremental application modernization efforts. As you'll see in the upcoming slides, AWS Cognito can be used to retrofit an application's access patterns.

AWS Cognito Authentication – User Pools

A Serverless Identity Directory for Storing:

- Users
- Groups
- Profile Information
- Custom Attributes

Application Integration for User Authentication

- Authenticates End Users with Username & Password or via an external IdP
- Handles user registration and password reset flows
- Provides a user login-hosted UI which can be customized to fit your application
- Provides simple sign-up and sign-in capabilities

Federation

- Identity Storage for Cognito can be local, via AWS IAM, or through a federated IdP such as Azure Active Directory
- Protocols for federating against external directories include:
  - SAML 2.0 and OpenID Connect (OIDC)

User Pools

User Pools in Amazon Cognito

A user pool is an Amazon Cognito User Directory defined by you, the AWS customer. As a **User Directory**, it is the place to store and manage users and their attributes, specifically for the AWS Cognito service. A User Pool allows the client's users to register / sign into their application and allows the customer to manage their user's profiles.

Sign-in functionality is available through Cognito or a third party when using a federated identity provider. All members of the user pool have a profile in the directory accessible programmatically by the client through an SDK.

By using a managed Identity Directory like Cognito User Pools, customers gain the following features:

- Built-in functionality for user signup and sign-in
- End user federation through Apple ID, Google, Facebook, or via OIDC and SAML
- Customizable UI for sign in - either under the amazoncognito.com domain or on your own custom domain
- Directory management of user profiles
- Built-in security features, such as MFA, account takeover protection, inspection for compromised credentials, and phone/email verification
- Customized workflows with user migration through AWS Lambda triggers

The advantage of choosing a managed solution from a cloud provider over custom authentication logic is obvious. You can essentially offload the logic to validate and authenticate your identities to AWS Cognito. The risky business of storing passwords, validating SAML tokens, and managing user attributes can be separated from your business logic.

AWS Cognito User Pools | Federating Access

Why Federate Cognito User Pools?

- Offload sensitive functionality:
  - Implementing custom code to act as a secure **Service Provider** is hard
  - Cognito can handle the validation of a SAML Token or the claims contained in an JWT Token
- Avoid creating a siloed identity repository in AWS Cognito
- Provide convenience to end users, letting them log in with existing credentials from 'social sign-in' IdPs like Facebook and Google
- Standard applications can use a single method for consuming identity

User Pools

SANS | SEC549 | Enterprise Cloud Security Architecture 37

Federating Access Using AWS Cognito User Pools

AWS Cognito User Pools come complete with a native user directory. One might use a simple native directory to authenticate users of a hobby project or an application in its early stages of development. However, most enterprise will look to User Pool Federation[1] when further uplifting access patterns of all of their applications.

Cognito offers several options for federating an end user's access:

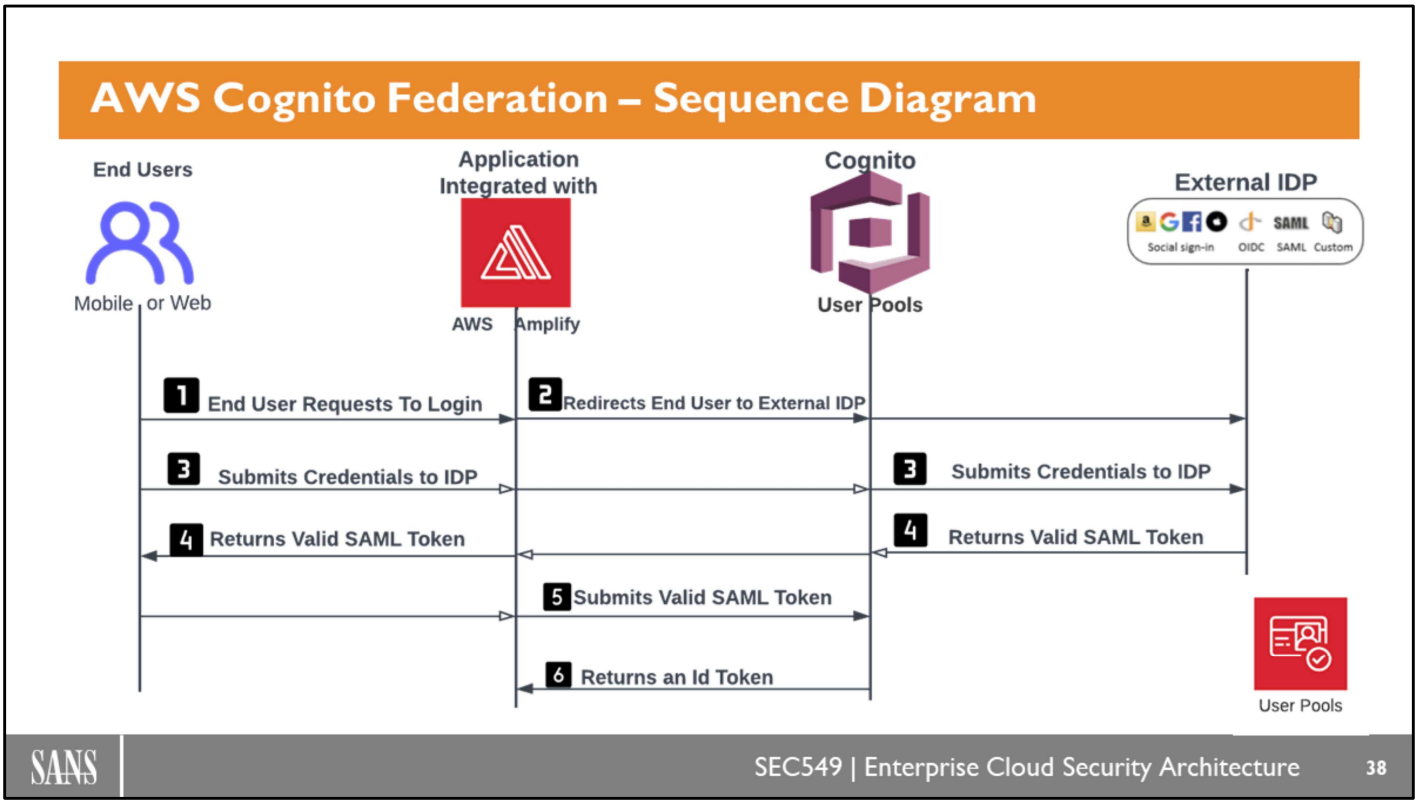
- Federating against Social Sign-in Providers (i.e., Google, Facebook) using OAuth
- Federate against any SAML-Compliant Identity Provider
- Federate against any OIDC Provider (i.e., AAD or Salesforce) using OIDC

A Cognito User Pool manages the overhead involved in handling tokens that are returned from any of these external identity provider types. Whether the integrated external identity provider is submitting a SAML Token, OIDC Id Token, or OAuth Access Token, Cognito assumes the responsibility of validating the user's authenticated state.

Cognito will provide an OpenIDConnect OIDC Token to the relying application that represents the authenticated state of your user. Applications who've delegated authentication to Cognito can standardize by consuming User Pool Id Tokens rather than supporting multiple methods for authenticating end users.

While Cognito User Pools are broadly considered to be a solution for application authentication, attributes of the end users can be carried to the relying application via the Id Token. By validating[2] and parsing the Id Token, an application can additionally make authorization decisions about the end user.

References:
[1]: https://sec549.com/id122
[2]: https://sec549.com/id123



Sequence Diagram in Federated User Pools Using AWS Cognito
Let's take an in-depth look at the authentication flow when an AWS Cognito User Pool is integrated with an External Identity Provider using SAML to authenticate end users.

1. When end users of an application wish to log in, their clients (web or mobile) are redirected to an AWS Cognito fully-managed, Hosted UI. The Cognito UI will display all login options available, whether it be a OIDC provider, social sign-in provider, or other external identity provider.
2. When a user makes their selection, AWS Cognito will redirect them to their Identity Provider of their choosing.
3. The end user is authenticated using their external Identity Provider
4. In response to a valid authentication, the external Identity Provider returns to the end user's client a SAML token, which is a signed XML document stating their authenticated state.
5. The end user's client will submit the SAML Token directly to AWS Cognito, where it will verify the integrity of the Identity Provider issued SAML Token.
6. In response to a valid SAML Token, AWS Cognito issues Id Tokens that can be used by the integrated application.*

*Note on segment 6[1]: AWS Cognito does not directly pass the Id Token to the integrated application; rather, an OAuth flow is used retrieve the User Pool Tokens. Cognito returns an Authorization Code to the end user's client, which is what the application uses to exchange for an Id Token.

The signature contained in the SAML Token submitted to the External IDP is validated by AWS Cognito. This is one of the key security benefits of using a managed service for authentication in that individual applications do not have to build logic to process cryptographic claims. The process of validating the signatures of SAML tokens is fraught with a history of problems and security failures[2]. Whenever possible, it's best to offload this responsibility to a central managed service.

References:
[1]: https://sec549.com/id124
[2]: https://sec549.com/id125

AWS Cognito User Pools – Managing Attributes

Standard Cognito Attributes

- AWS Cognito makes available standard attributes to assign to users, whether federated or local
- These values can originate from claims passed on from the Identity Provider (IdP) or can be provided during new user registration

Custom Attributes

- Custom Attributes can be assigned to users in your User Pools
- Custom Attribute values can originate from claims passed on from the Identity Provider
- Use Custom Attributes to convey Group Membership

User Pools

Managing Attributes in AWS Cognito User Pools
Whether end users authenticate against an Identity Provider using SAML or OIDC, there is an opportunity to convey centrally housed user attributes to AWS Cognito and pass those attributes to the relying application. These attributes can allow the application to make authorization decisions.

Leveraging User Pool attributes can help applications enforce one of the key components of a Zero-Trust design as outlined in Google's BeyondCorp. Attributes convey the access level of users in a real-time fashion, allowing access control decisions to be made dynamically.

What is a User Pool Attribute?
Attributes[1] are pieces of information that help you identify individual users, such as their name, email, and phone number. When creating a new user pool, a default slate of standard attributes are available, which may or may not be required with the '*username*' always being a required attribute. You can set which attributes are required and which are optional when configuring a User Pool for your application.

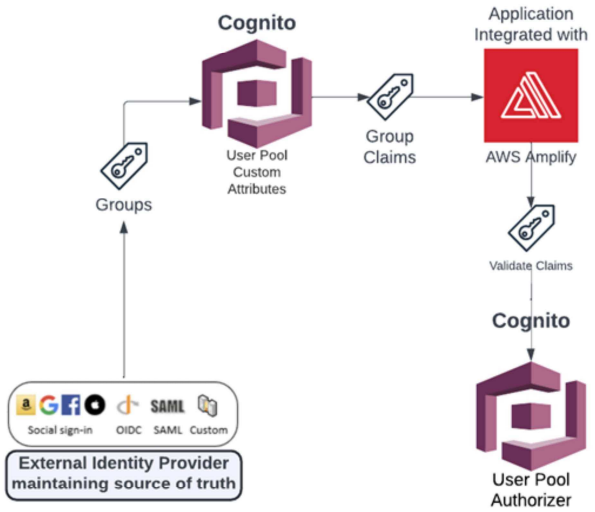
Custom Attributes[2]
Up to 50 custom attributes are allowed to be added on top of the standard, default attributes. Custom attributes are especially important when federating identity and looking to pass group membership from the external Identity provider on to an application. AWS Cognito can convert group memberships into a custom attribute and pass it on as a claim in the identity token (JWT Token).

Attribute Permissions
In an Amazon Cognito user pool, you can set *read* and *write* permissions for each default and custom attribute. The *read* permission is required to be set for all attributes which the integrated application needs to read. In this scenario, your application would be able to see the attribute value but could not modify it directly. It's considered common practice to set the *read* permission across all attributes. Given this pattern, attributes are not appropriate for storing sensitive values.

The *write* permission should be carefully set on attributes. Depending on the application configuration, this permission might allow end users to directly update the value of an attribute. This might be appropriate for an 'email' or 'phone number' field but should not be set for a field conveying group membership. Attributes used in determining access level should remain immutable, with the *write* permission remaining unset unless controls are set within the federated application itself.

References:
[1]: https://sec549.com/id126
[2]: https://sec549.com/id127

Cognito - Passing Group Memberships to Applications



Map groups from external federated identity provider to custom Cognito User Pool custom attributes in order to maintain a centralized view of group memberships.

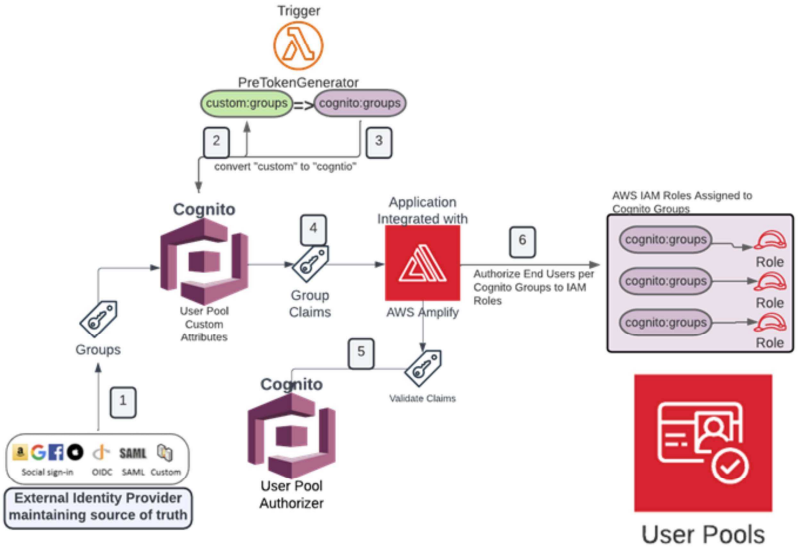
User Pools

Scenario: The customer wants to use their existing Groups and Group membership (housed in their central IDP) as the source of truth. When a user logs into an application configured with AWS Cognito, their group membership is automatically read into AWS Cognito from the 3rd party federated service. The User's group membership is a custom attribute passed to the application and can be used for application-layer decisions.

1. Pass along group memberships, stored in your external IDP, to AWS Cognito during user authentication. Cognito will map group membership as a custom attribute.
2. Then, send group membership to the application (web or mobile) in the form of a claim in the JWT token.
3. AWS Cognito exposes an API for applications to validate the JWT Token signature.
4. After this is accomplished, the application can consider that the end user is authenticated and use the claims to make application-level authorization decisions.

AWS Cognito – Leveraging 3rd Party Group Membership

1. External Identity Provider sends group membership to Cognito where they are stored as Custom Attributes
2. Prior to issuing the JWT Token, the claims are sent to the **Lambda PreToken Generator Function**
3. The Lambda PreToken Generator converts "custom:groups" to "cognito:groups" which can be recognized by IAM Policy
4. JWT Token issued to the application with claims declaring the end users "cognito:groups"
5. Application validates the signature of the JWT Token
6. Application can assume roles for end user based on their group membership declared in the token



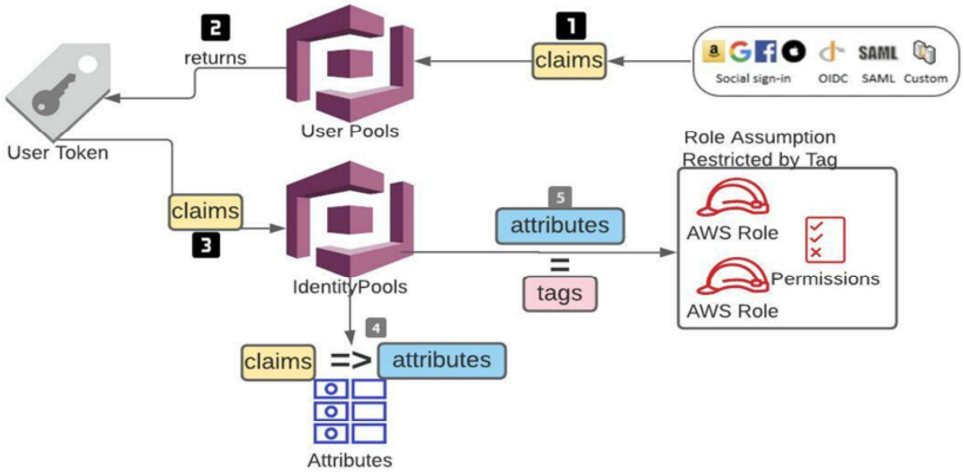
Scenario: The customer wants to use their existing Groups and Group membership (housed in their central IDP) as the source of truth. When a user logs into an application configured with AWS Cognito, their group membership is automatically read into AWS Cognito from the 3rd party federated service. Without intervention, the group membership is defined as a custom attribute, which is great for application-layer decisions but not helpful in accessing AWS IAM Roles.

An AWS Lambda called a **PreTokenGenerator** is required to:
1. Read the custom:groups attribute and convert this into the attribute named cognito:groups
2. Lambda can be configured to make any pre-authentication decisions required

As with any AWS IAM Group, Roles can be assigned to the groups. When creating a group, you can associate the group with an IAM role, meaning members of this group will automatically have that role applied. Users are then able to access AWS Resources based on the permissions assigned to those Roles.

AWS Cognito Authentication – ABAC With Tags

Attribute-Based Access Control



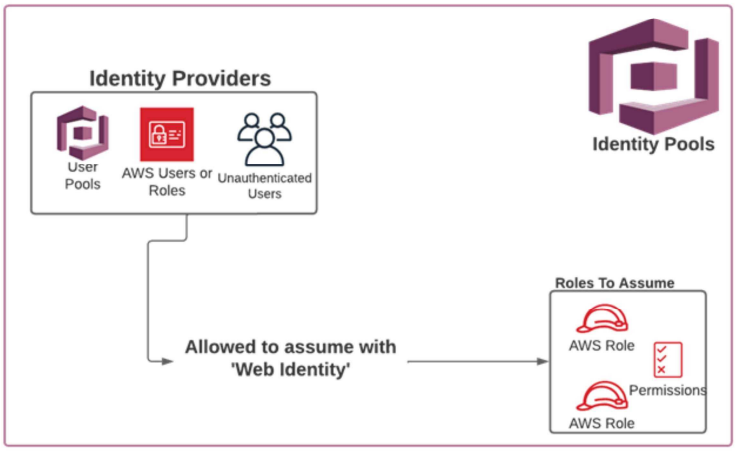
As we saw when discussing ABAC for AWS Identity Center, it is an alternative authorization strategy to RBAC that conditionally defines permissions based on attributes.

What is the main difference between ABAC and RBAC?
The main difference is in the way each method grants access. RBAC techniques allow you to grant access by **roles,** often as a result of group membership, while ABAC techniques let you determine access by **user characteristics.** Neither method is better than the other; rather, they are different techniques that can be used in your IAM Strategy depending on characteristics of your users and the outcomes you are looking for. It might be more advantageous to use an ABAC strategy when a handful of user characteristics are likely to change frequently and have many possible values. ABAC would prevent the creation of multiple roles for every possible attribute value.

Let's look at how AWS Cognito implements ABAC:

1. Attributes attached to a user are passed to the identity provider as claims, either in an identity token or as attributes in a SAML token. In this diagram, AWS Cognito is being depicted as the identity provider.
2. The claims embedded into the identity token and passed to AWS Cognito Identity Pools
3. In AWS Cognito Identity Pools, a mapping exists between claims and attributes to use in access control decisions. Claims used for ABAC will be passed through to the end users session token when accessing AWS resources.
4. As an end user attempts to assume a role in AWS, their access to role assumption is conditionally restricted based on the tag injected into their session token by AWS Cognito.

AWS Cognito Authorization – Identity Pools



- ***AWS Authorization* for end users**
- **Assigns permissions to a variety of Principals**
  - Map Principals to an AWS Role and associated permissions
  - Cognito Identity Pools will exchange User Token for Identity Tokens


Identity Pools

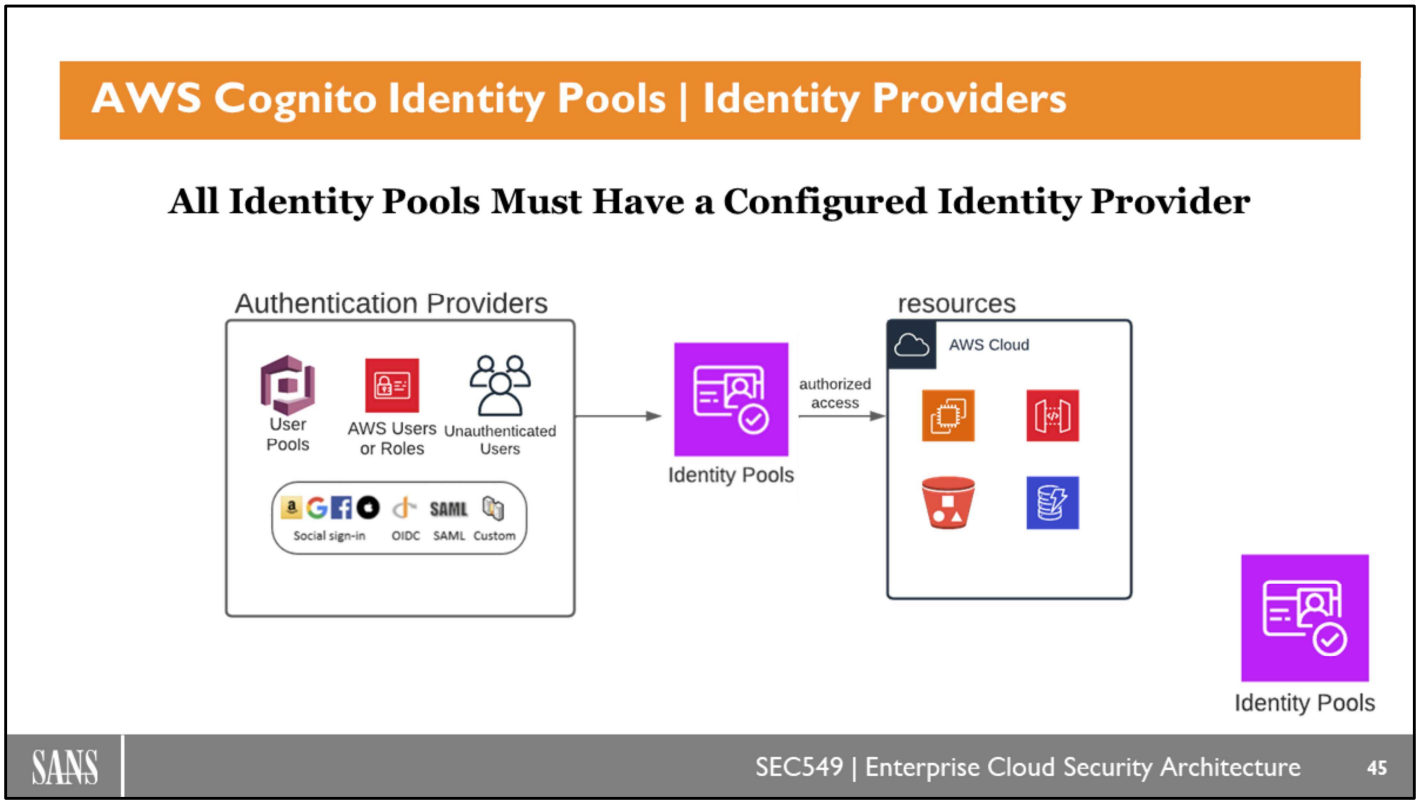
What Are Identity Pools?
Identity pools[1] in AWS are collection of Federated Identities for which you can assign AWS permissions. Identity pools enable you to grant your users access to other AWS services. You can use identity pools and user pools separately or together.

Once a user has been authenticated through AWS Cognito and verified with a specific Authentication Provider, the identities can be assigned a particular IAM Role with the permissions that role have been given. This role limits what AWS resources the authenticated Cognito end user has access to in an AWS account.

When architecting AWS Cognito into any application, you need not use both user pools and identity pools. Say if the application choses to handle authorization with custom logic and end users are not interacting with AWS resources, you may choose only to leverage User Pools.

However, if the user needs access to AWS services, mapping them to an Identity Pool is often the easiest way to assign them AWS permissions and grant them access beyond the application layer.

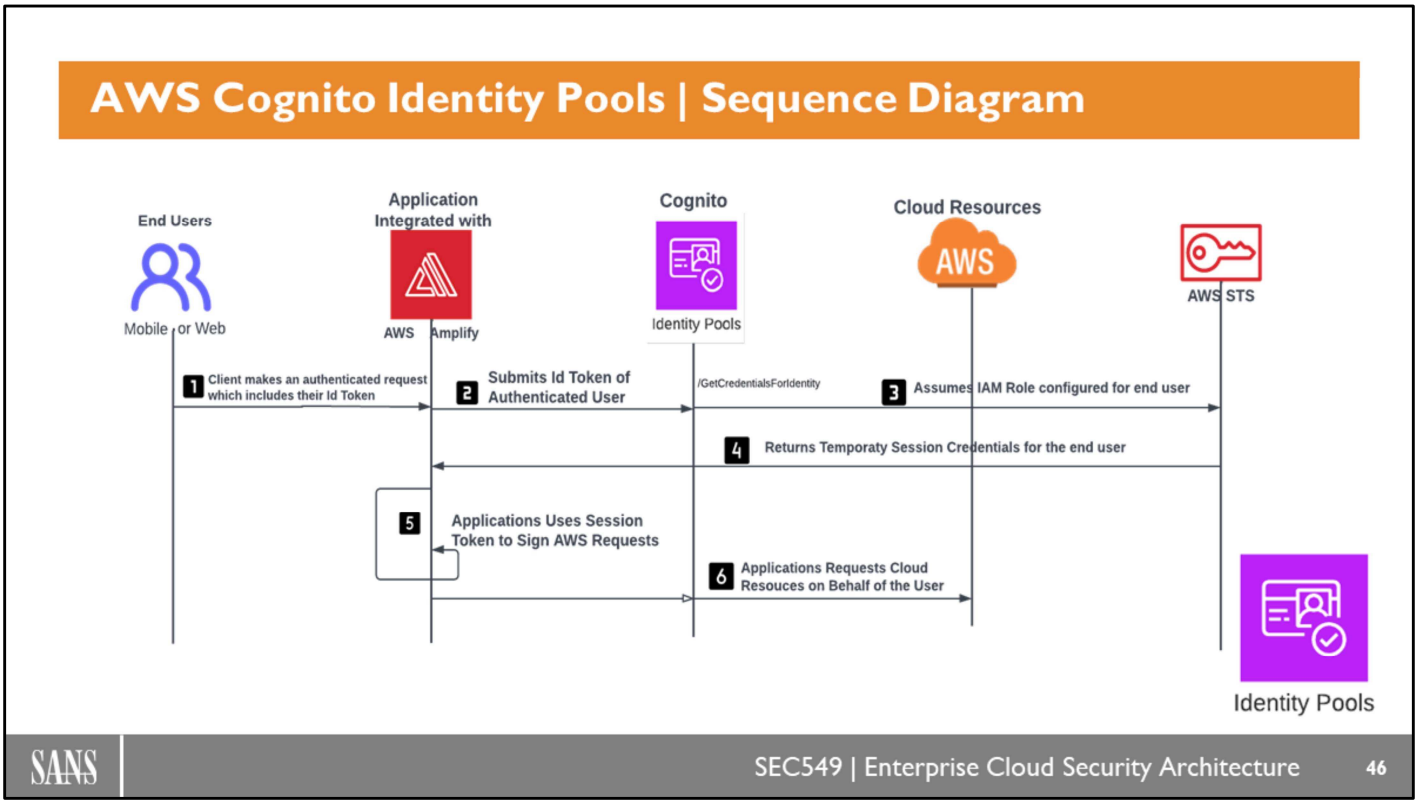
Reference:
[1]: https://sec549.com/id128



Identity Providers
Unlike User Pools, when utilizing an Identity Pool, a backend authentication provider must be configured. Identity Pools do not have a built-in user directory and do not have native authentication capabilities.

Options for external authentication providers include:

- AWS Cognito User Pools
- Social Sign-ins like Apple, Google, and Facebook
- Custom SAML integrations
- Custom OIDC integrations
- Or use your own existing authentication process, while still using Amazon Cognito to enable end users to access AWS resources.



Let's take an in-depth look at how Cognito Identity Pools allow integrated applications to authorize end users to AWS resources. In this scenario, the end user has authenticated against an Authentication Provider and was returned an Id Token. This pattern has allowed the integrated application to carry the context of the end user to backend requests.

1. An end user requests a file from the application. Their Id Token is submitted in the HTTP request to the application identifying themselves.
2. The application submits the Id Token to the AWS Cognito Identity Pool and to the '/GetCredentialsForIdentity' public endpoint[1] provided by Cognito.
3. AWS Cognito assumes the IAM role that was configured for the end user. Here, AWS Cognito is using the 'AssumeRoleWithWebIdentity' method of role assumption.
4. Upon successful role assumption, returned from the AWS Secure Token Service (STS) are temporary session credentials for the end user.
5. The application uses the temporary session token to sign a HTTP request, in a process involving the addition of authentication information to AWS API requests[2].
6. The application can now make an API request to retrieve the requested file, acting as the end user.

References:
[1]: https://sec549.com/id129
[2]: https://sec549.com/id130

Cognito Identity Pools - Cognito Specific IAM Policies

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${cognito-identity.amazonaws.com:sub}/*"]}}
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/${cognito-identity.amazonaws.com:sub}/*"]
    }
  ]
}
```

Identity Pools

Let's look at how an identity-based policy can be configured to allow your Identity Pools end users access to specific resources[1].

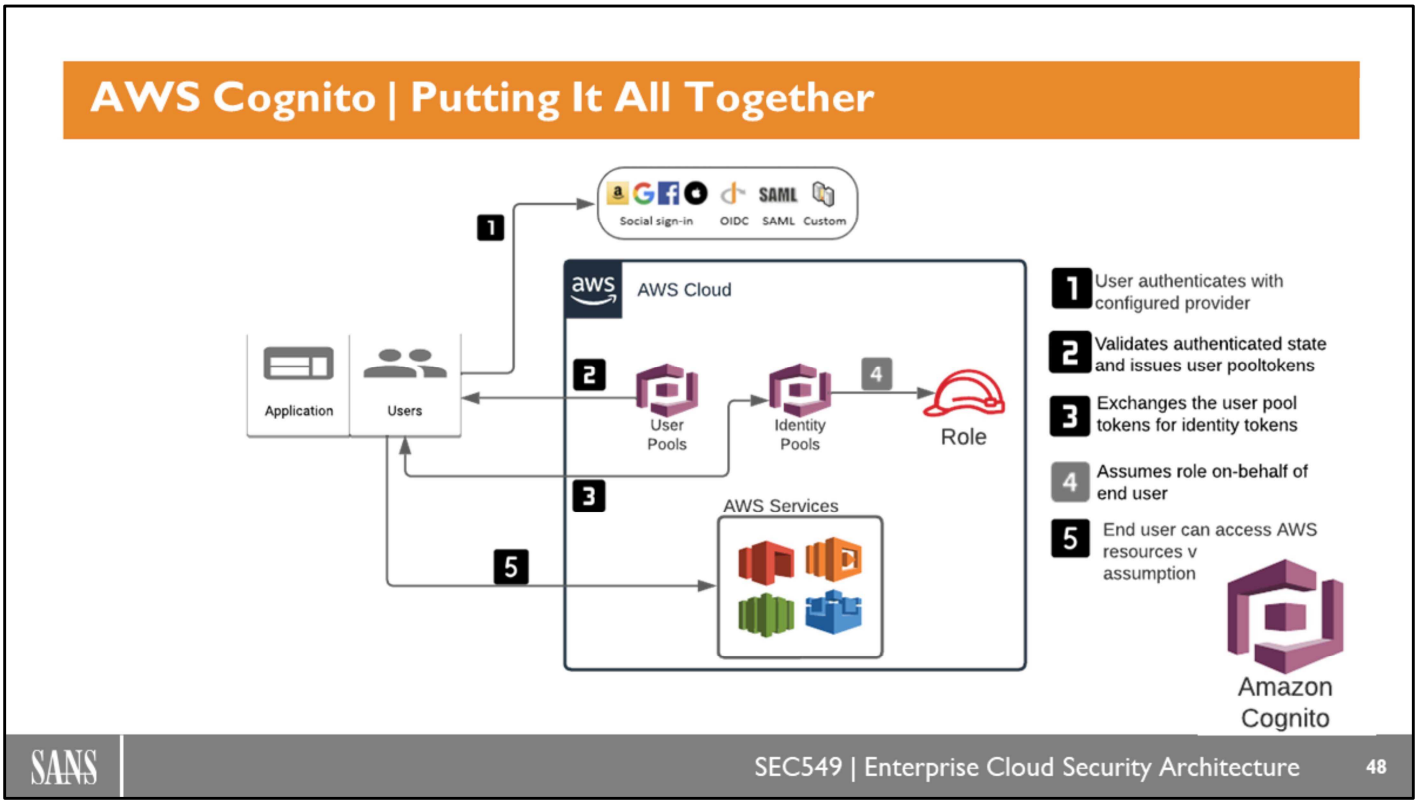
In this example, the policy will allow the ability to both read and write to a specific S3 bucket.

Read Access: All Identity Pools end users defined as those having the variable ${cognito-identity.amazonaws.com:sub} will be able to list the bucket called 'mybucket', which is a fairly innocuous and necessary global permission when working with S3 buckets

Write Access: End Users defined with the variable ${cognito-identity.amazonaws.com:sub} are granted the Get Objects and Put Objects permissions on 'myBucket' but only on a very specific prefix. Their access is scoped with the Resource field to only the S3 prefix that matches their Identity Pool User Id. Here, the end user defined with the variable ${cognito-identity.amazonaws.com:sub} would not be granted access to other objects in arn:aws:s3:::mybucket.

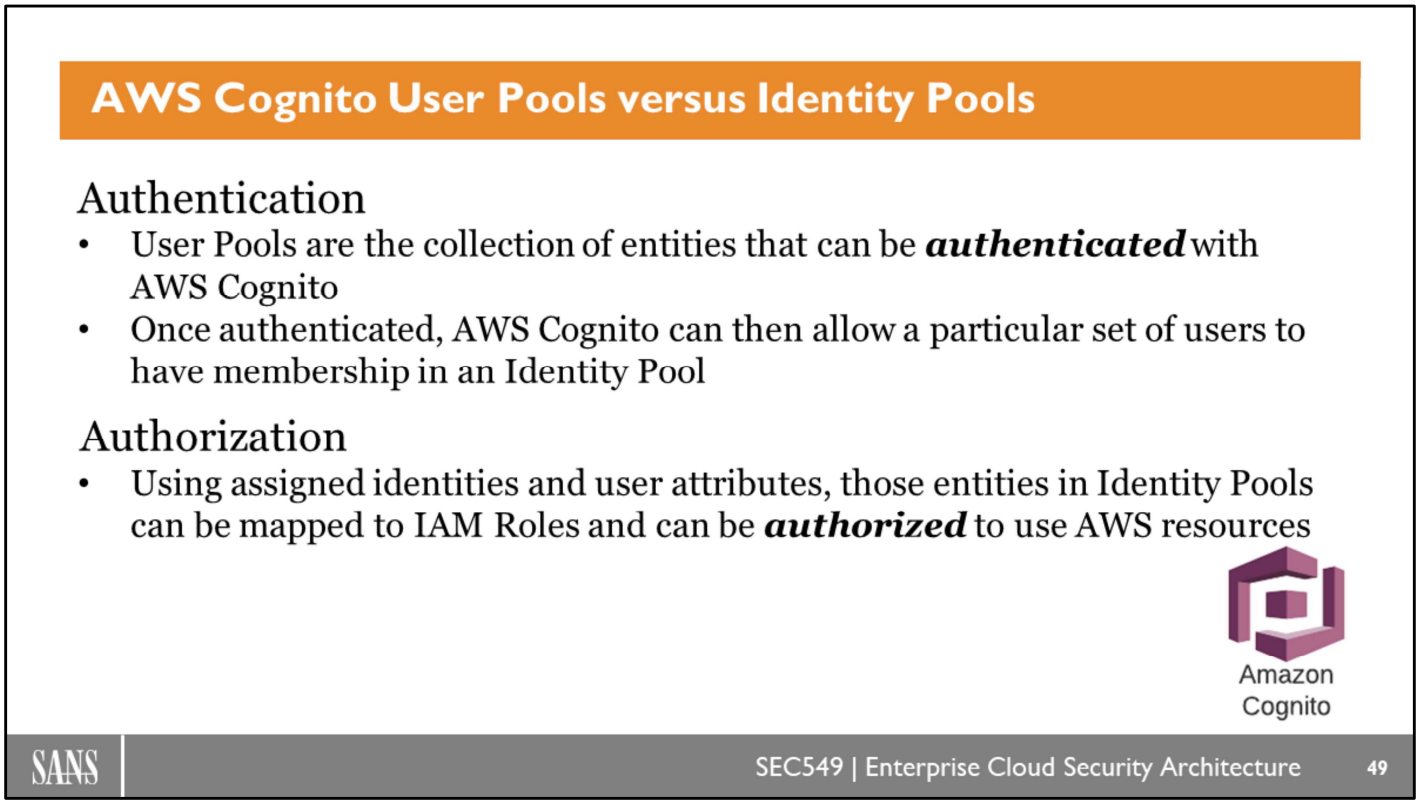
This is a common pattern seen in IAM policy if you need to enable read/write access to end users own documents stored in S3 and not other users.

Reference:
[1]: https://sec549.com/id134



Now that we've gone into the weeds looks at Cognito User Pools and Identity Pools, lets **zoom back out** and see how they fit together.

Step 1: User signs in with a configured identity provider.
Step 2: Cognito user pool validates authenticated state and issues application/end user their user pool tokens.
Step 3: The app exchanges the user pool tokens for identity tokens from the configured identity pool
Step 4: The Cognito identity pool assumes the configured role for the end user and returns temporary session credentials for the end user or application to use.
Step 5: The app user can use the AWS credentials they received to access other AWS resources.



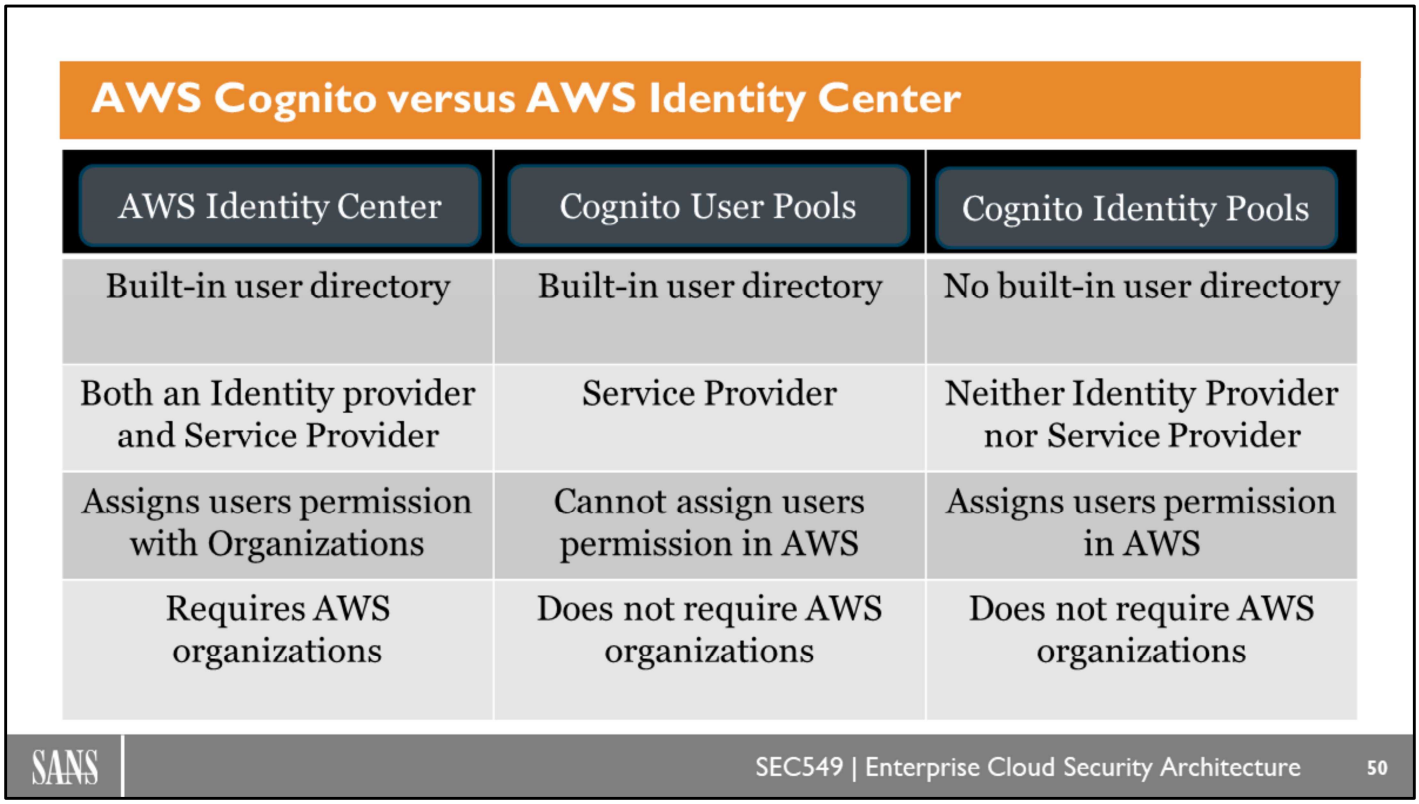
AWS Cognito User Pools versus Identity Pools
As we've covered over the last few slides, AWS Cognito is comprised of two separate, but related, services: User Pools and Identity Pools (also called Federated Identities).

Now that we've learned a little about these two services, let's make sure the distinctions between the two are clear.

User Pools provide a user directory for configured applications, which include everything you'd expect from an IAM such as sign-up and sign-in capabilities, group management, etc. User Pools also provide applications with information about the authenticated user called attributes which can be used by the integrated application for authorization decisions.

Identity Pools, in contrast, are used to assign AWS IAM roles to users who are post-authentication, having authenticated through a separate Authentication Provider. Because these users are assigned an IAM role, they can be assigned unique IAM permissions, allowing them to access AWS resources directly.

User Pools don't deal with AWS permissions. Rather, they can convey information like group membership, so applications can deal with authorization independently. Identity Pools, in contrast, grant users their permissions at the IAM level. This means that Identity Pools allow for a much more granular set of permissions with respect to AWS services. Because Identity Pools map a user from an Identity Provider to an IAM role, they allow permission management of AWS resources to stay within AWS IAM.



AWS Cognito versus AWS Identity Center

If you are wondering whether AWS Identity Center or Cognito is the right authentication solution, the question to ask is about what kind of users are needing authentication.

If your user population is a workforce employee where their identity is housed in a central identity provider, typically the right solution is to leverage AWS Identity Center. This population base typically requires some level of access to the control-plane of AWS. The configuration of their access via AWS Identity Center is fairly straightforward.

If your user population includes customers or unauthenticated users, onboarding and managing their user attributes via Cognito User Pools makes a lot of sense. With User Pools, you can provide your end users access to multiple 'social sign-on' providers, including the major public login providers Amazon, Facebook, and Google.

If either a workforce or employee user population requires access to AWS resources where access is delegated via an application, consider configuring their access via AWS Cognito Identity Pools. With Identity Pools, your end users can be housed in any external identity provider while leveraging Identity Pools to manage AWS permissions for access to resources inside an application.

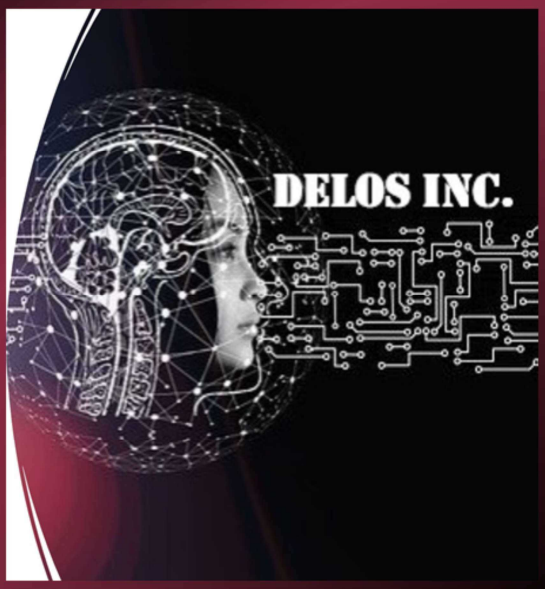
Cloud Journey Phase: Uplifting Access Patterns

Scenario:

A subsidiary of Delos Inc, Delos Destinations, was acquired several years ago but has been operating larging independently since the acquisition.

Delos Destinations runs a custom, internally developed application on AWS. Internal users have usernames and passwords for this application that are separate from their domain accounts.

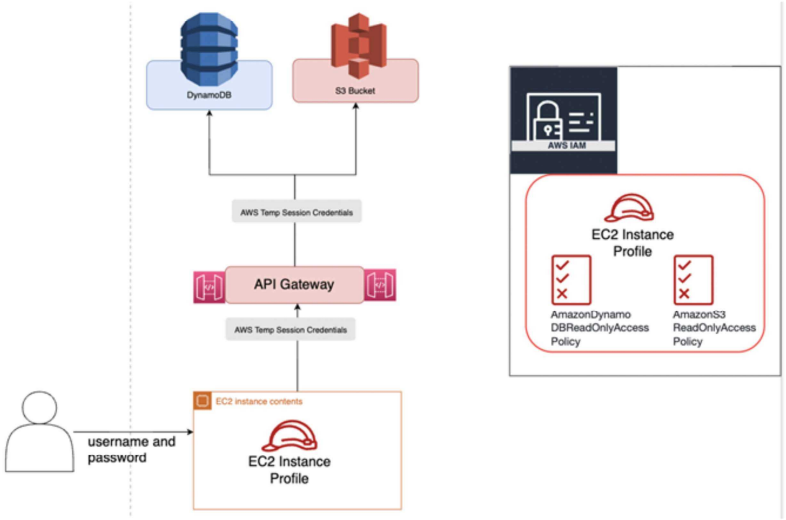
Recently, their workforce identities have been added to the core Delos Office 365 tenant/Azure AD instance. This gives us the opportunity to bring access to this application under central control and ensure that access to backend resources is scoped to the context of the end user.



Reference:
Image by User geralt from Pixaby: https://sec549.com/id51

Lab 2.1 – Current State Architecture

- Web requests made to the application hosted on EC2 are under the authenticated context of the end user
- Requests made to S3 and Dynamo are authenticated with credentials from the EC2 instances
- As a result, the EC2 instances are assigned permissions allowing them access to all S3 and DynamoDB resources



End Users are authenticating to publicly-facing web applications using username and password. Custom application logic authenticates the user and authorizes their access to backend resources, such as S3 Buckets and DynamoDB Items. The EC2 Instance is assigned an Instance Profile, a special kind of AWS Role allowing the application access to backend resources.

While web requests made to the application are under the context of the end user, requests made to retrieve S3 and DynamoDB objects are authenticated with credentials from the EC2 instances. As a result, the EC2 instances are over-privileged. They have been assigned access to all S3 and DynamoDB resources.

Lab 2.1 – Integrating Auth Into a Legacy Application

Estimated Time: 45 minutes

A bespoken web application was created in-house by the Delos Destinations development team. To uplift the access patterns, AWS Cognito User Pools have been integrated and used to delegate authentication.
Your task in this lab is to document the new authentication flow end users will experience when logging into the custom Park Tracker application.

Preparation

- View **Lab 2.1** in the course workbook
- Open the workbook in an incognito window
- https://workbook.sec549.com
  - Username: H03-student
  - Password: million-sailor-DEAR

Objectives

In the provided template sequence diagram:

1. Document how end users submit their credentials to an external identity provider
2. Document how AWS Cognito returns an identity token to the end user upon authentication

This page intentionally left blank.

Lab 2.1 – Summary

In this lab, you...

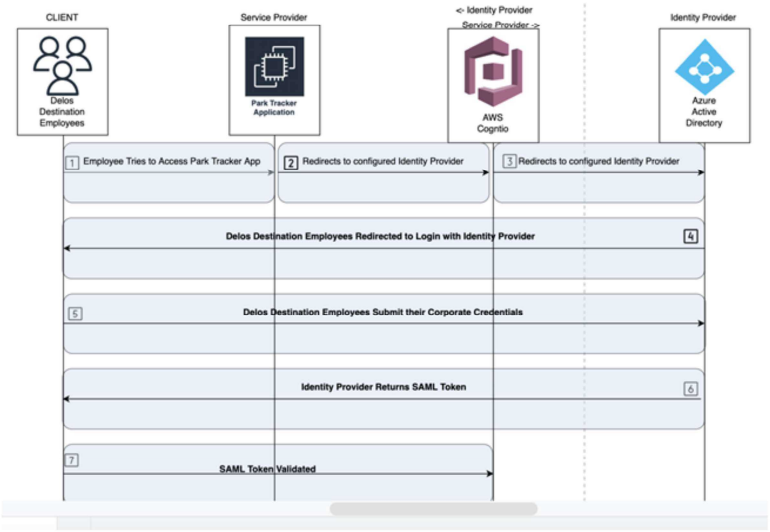
- Depicted SAML Federation between Azure Active Directory (AAD) and AWS Cognito
- Logged into the Delos Destinations Park Tracker App to observe the authentication flow
- Logged into the AWS Console to answer the ⚡ See it in Action ⚡ questions

This page intentionally left blank.

Lab 2.1 – End State Architecture (1)

Part 1

1. User requests access to an application
2. As a result of the integration, the application redirects the end user to with AWS Cognito
3. AWS Cognito User Pools redirects the user to the configured external identity provider
4. External identity provider login is presented to the Delos Destination employees
5. End users submit their credentials to the external identity provider (AAD)
6. Upon successful authentication, AAD returns a valid SAML token
7. The SAML token is submitted to AWS Cognito for validation

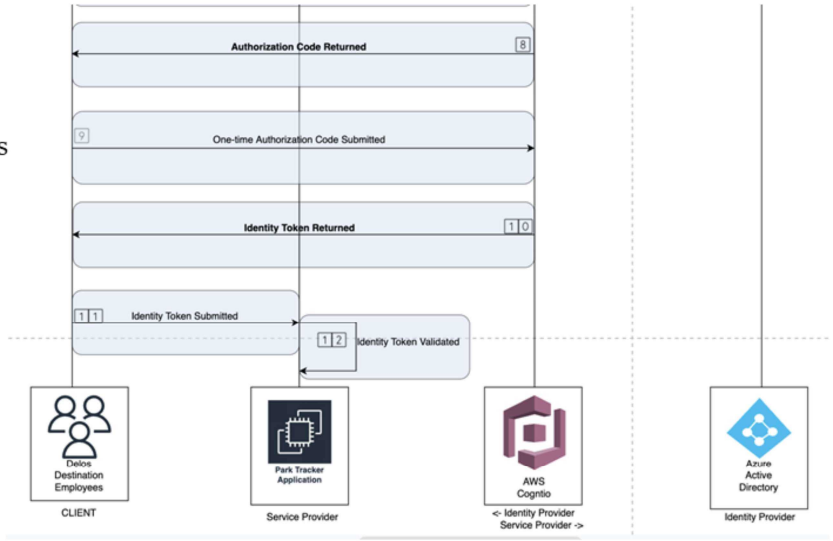


This page intentionally left blank.

Lab 2.1 – End State Architecture (2)

Part 2

8. After a SAML token is validated, returned to the end user is an authorization code
9. The one-time use authorization code is submitted to AWS Cognito
10. When the Cognito endpoint receives a valid authorization code, it is exchanged it for an identity token
11. Armed with a valid identity token, the end use submits it to the Delos Destinations Park Tracker App
12. It's the applications responsibility to validate the integrity of the identity token before use



This page intentionally left blank.

Course Roadmap

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

SECTION 2

- Introduction to Cloud Migrations
  - Drivers for Cloud Migrations
- Implementing Zero-Trust Architecture
  - Using Cloud Services to get to ZT
    - Lab 2.1: Integrating Auth into Legacy Application
- ***Establishing Perimeters for Application Access***
  - Connecting VPC-Aware and Non-VPC Aware Services
    - Lab 2.2: Creating a Shared VPC Network
- Establishing Perimeters for Data Access
  - Managing S3 Access At Scale
    - Lab 2.3: Access Control For Shared Data Sets

This page intentionally left blank.

Takeaways

In this module, you will get a refresher on basic networking constructs in the cloud and how to apply security controls at the network layer.

Diving deeper, we'll cover some common business requirements you might face when working with virtual cloud networks.

- Centrally managing network resources
- Managing network controls at-scale
- Designing for hybrid deployments
- Providing private access to public cloud services

This page intentionally left blank.

Software-Defined Networks in the Cloud

Comparison of cloud network terminology and capabilities

Cloud Provider	Segregation of Network	Regional or Global?	Bound to which resource?	Network-layer controls
AWS VPC	subnets	Regional	AWS Account	Security Groups NACLs
GCP VPC	subnets	Global	GCP Project	Firewall Rules
Azure VNet	subnets	Region	Azure Subscription	Network Security Groups

Virtual Networks in the Cloud
AWS, Azure, and GCP each provide for software-defined, virtual networks you can use as building blocks to create your own cloud-hosted network. These are the networking partitions on each cloud provider's network allowing you to configure network-layer connections between your segmented applications, providing a link between your organization's on-premises network and the public. The basic virtual network constructs in each cloud are called AWS VPC, Azure VNet, and GCP VPC. Each of the three major cloud providers let the user control much of the virtual networking environment, such as the IP address ranges, subnets, access control rules, and routing options.

AWS VPCs
The AWS VPC of your organization is a logical collection of network resources within the AWS public cloud. It uses the IPv4 addressing protocol by default; you can specify an IPv4 CIDR block when you set up your VPC and when you define your subnets.

AWS does support IPv6 as an option for you to assign to instances in your VPC subnets, but the IP range is selected by AWS. IPv4 and IPv6 resources can be configured to talk to one another but cannot do this by default. Because of the way it is set up, you need to use VPC peering to route traffic between regions. You can use AWS Console to set up your VPC IPv4 address block, apply dual stacking (if desired), and set up your subnets, region, zones, and Internet Gateway.

GCP VPCs
Google has a unique virtual networking offering to its cloud customers as a result of their global network achieved through their virtualization stack named 'Andromeda'[1].

With Andromeda, Google orchestrates its Software-Defined Networking layer to intelligently route traffic between regions and zones, serving traffic to a customer where it is most optimal to do so. This capability was extended to its cloud customers and is what powers the GCP VPC.

Virtual Networks (VPCs) on the Google platform are not scoped to a single region like with AWS or Azure. Subnets, however, are scoped to particular regions, but Google will dynamically advertise routes between subnets without the need for customers to configure gateways to connect regionally scoped networks.

One of the benefits of a utilizing a global VPC resource is that you can apply network security policies at a central point, rather than scattered across regions. With GCP firewall rules, you can create both *allow* and *deny* rules, which isn't true of AWS Security Groups. The firewall rules in GCP can be automatically applied to all compute instances in a VPC or targeted to only specific compute instances with network tags.

Azure VNets
The Azure VNet is analogous to the AWS and GCP VPCs. With this virtual network, you can deploy many different resources – Azure VMs, App services environments, and others. Subnets allow you to further segment your network, just as in AWS and GCP. This network isolation can be achieved by applying controls such as Azure Network Security Groups. Network Security Groups can be used to filter network traffic to and from compute resources in a VNet or traffic entering and leaving a subnet. Like the Security Group in AWS, Network Security Groups in Azure are stateful controls.

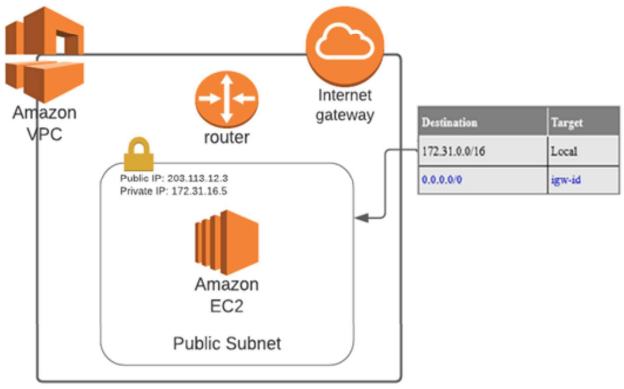
Reference:
[1]: https://sec549.com/id135

AWS | Anatomy of a Public Subnet

Required for traffic to route to the internet:

1. **Public IP** (Either on the EC2 or NAT Gateway)
2. **Connectivity** (via Internet or Transit Gateway)
3. **Route** (defined) in routing tables

AWS provides a default VPC equipped with the 'basics' to have a public-facing website, for example, without a great deal of configuring on your part.



Cloud Networks 101

Just to orient ourselves, let's talk about the basic components that are required in an AWS VPC to have public internet connectivity.

Anatomy of a Public VPC
If you want a simple website, for example, you might start with a public-facing VPC. Your website needs a domain name and your VPC must have an Internet Gateway, a public IP address (based on EC2 or NAT Gateway), and a routing table.

The most basic option is to use AWS and its default VPC. This VPC has the following provided for you:
- A size/16 IPv4 CIDR block (with up to 65,000+ private IPv4 addresses)
- A size /20 default subnet in each Availability Zone
- An Internet Gateway
- A route to the main table route. It points all traffic to the Internet Gateway
- A default Security Group
- A default NACL
- Set of DHCP options

Once you have this already configured through AWS, you can do much more to enhance its functionality for you by adding nondefault subnets, modifying the routing table, adding other routing tables, updating the rules, adding IPv4 CIDR blocks, and adding more Security Groups.

A Route 53 Resolver is included in all VPCs, including the default VPC. It maps to a DNS server that runs on a reserved IP address for the network range of your VPC (plus 2). It maps to the primary CIDR block if you have more than one CIDR block associated with your VPC. It auto-generates FQDN for all the EC2 instances in your VPC. It will resolve private IP addresses within the VPC as well as public IP addresses outside the VPC.

Creating a Private Hosted Zone

You can also integrate DNS solutions between the Route 53 Resolver and DNS resolvers on your network, creating a hybrid private hosted zone. You do this by configuring forward rules. Now, any network you can reach from your VPC (including peered VPCs and on-premises networks) can be in 'your network'. There are inbound and outbound Resolver endpoints that you create first; then you forward queries using these endpoints to route traffic within your network. Your on-premises network must connect to AWS through a NAT gateway, VPN, or AWS Direct Connect to connect to your VPC.

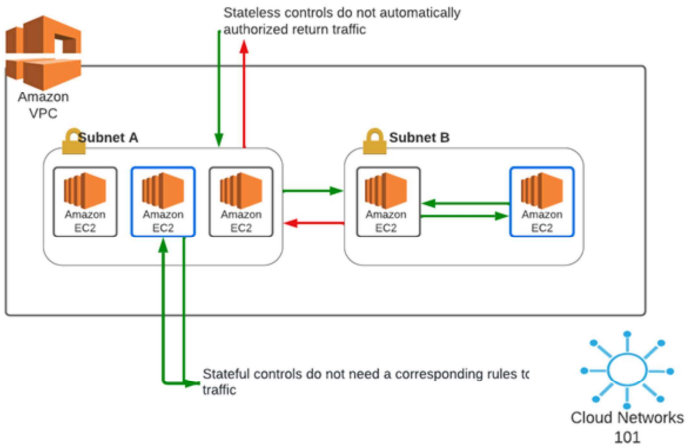
AWS | Authorizing North || South Traffic in the Cloud

Stateful Network Controls

- Response traffic is automatically allowed once a traffic is allowed
- Examples include
  - AWS Security Groups

Stateless Network Controls

- If an ingress traffic is allowed, the response is not automatically allowed unless explicitly allowed in another the rule
- Examples include
  - AWS NACLs



Authorizing North/South Traffic in the Cloud
For internet traffic to be allowed in any VPC, you need a public IP address that can be assigned to it. You can also use an Elastic IP (EIP) address. Your default VPC in AWS automatically includes an internet gateway; each launched instance into a default subnet within this VPC has a private and public IPv4 address. Any nondefault subnet or VPC has a private IPv4 address but will not have a public one unless you assign one or allow traffic through an 'allow' security rule.

Stateful Network Controls
A stateful firewall will track individual sessions and maintains a state table of all existing connections. Permitted inbound connections automatically allow responses. This is a simpler design because you need only to think about configuring your inbound traffic; the necessary outbound traffic will automatically be allowed. This is generally preferred when either stateful or stateless controls could be used because of this simplicity. AWS Network Firewall and AWS Security Groups are all based on using stateful network controls.

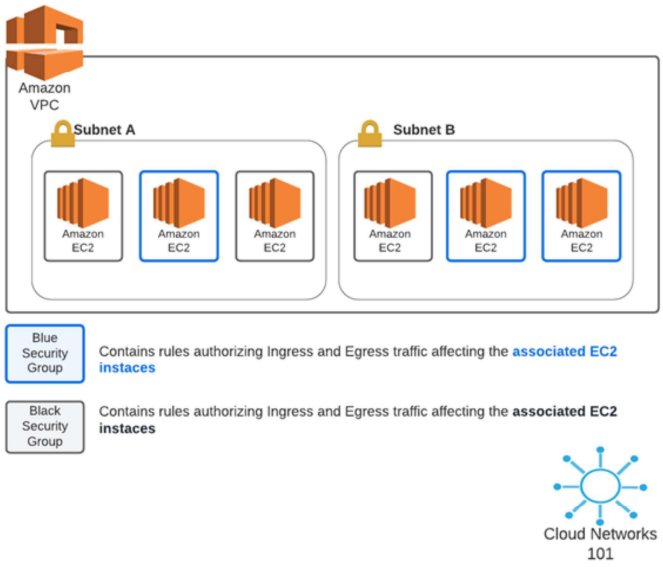
Stateless Network Controls
These are controls that do not use a state table. Stateless network controls cannot correlate inbound and outbound traffic. This means you must create individual rules for both inbound and outbound traffic. The term 'stateless' means that responses to allowed inbound traffic are subject to their own outbound traffic rules (and vice versa).

AWS recommends using a NACL (Network Access Control List) to nail down your ingress and egress subnet traffic. By default, each custom NACL denies all traffic in or out until rules are added. Each subnet you create must have a NACL or you will be left with AWS' default deny/deny situation. Each NACL can link to more than one subnet, but a subnet can only have one NACL at a time. NACLs, by definition, have separate ingress/egress rules.

AWS Internet Gateway automatically provides a stateless 1:1 Network Address Translation (NAT) between public and private IPv4 addresses. You can use a network address translation (NAT) device to allow outbound traffic but not any unsolicited inbound traffic. The cloud-hosted NAT will map multiple private IP addresses to a single public one. It can be configured with an EIP and connected to the internet gateway.

AWS | Security Groups

- **Group resources together** to define common inbound and outbound traffic rules
- Can only be applied to certain resources with with Elastic Network Interfaces (ENIs) such as:
  - EC2 Instances
  - VPC Interface Endpoints
- Security Group Ids can be used as targets to authorize ingress or egress traffic



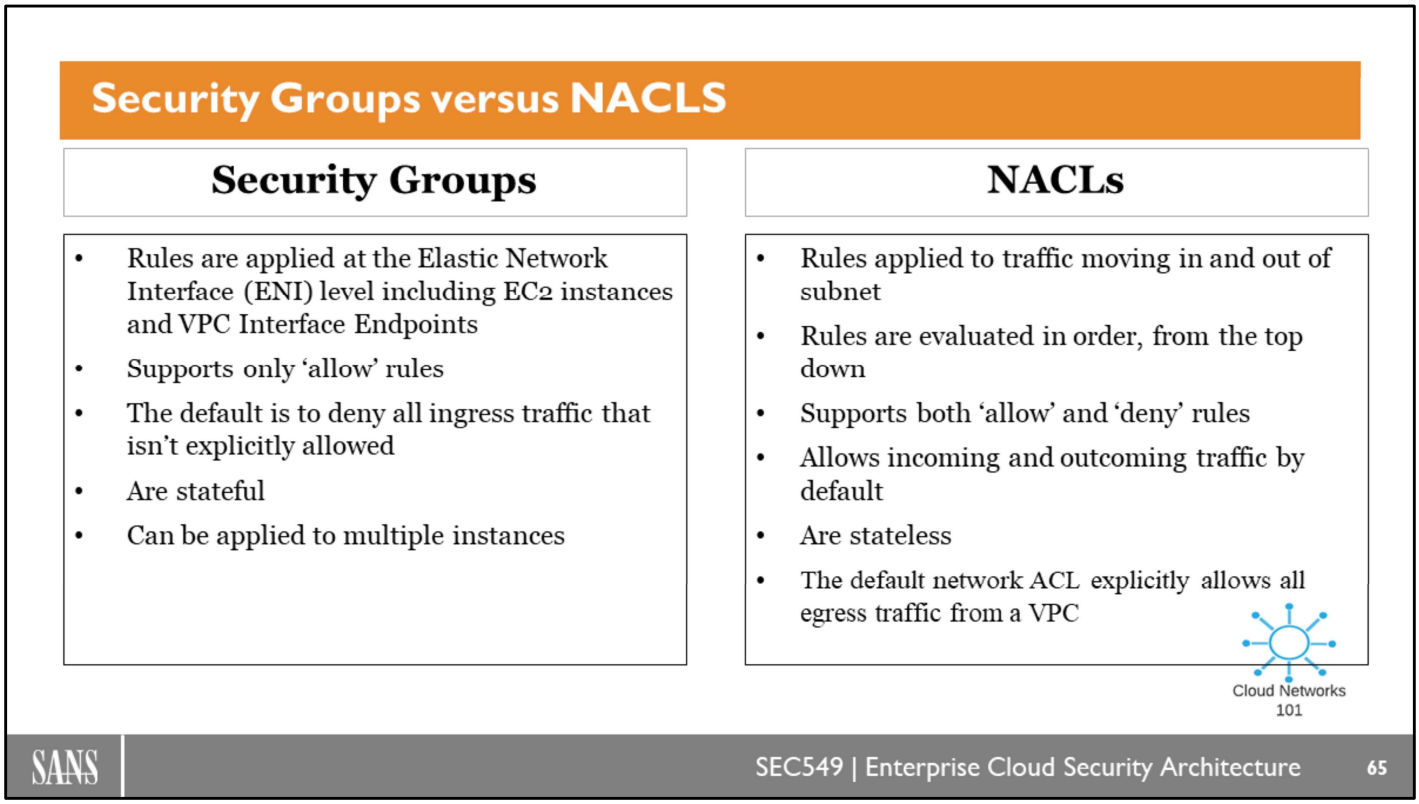
SANS | SEC549 | Enterprise Cloud Security Architecture 64

Security Groups in AWS Networking
Security groups act as virtual firewalls for your E2 instances, application load balancers, and Elastic Network Interfaces (ENIs); they control your inbound and outbound traffic by adding separate rules for the inbound and outbound traffic. Every EC2 Instance is required to be associated with a security group. If you don't specify a particular security group, the instance will be assigned to the default security group.

Security group rules use 'allow' rules only. These rules control the inbound traffic that's allowed to reach the instances associated with the security group. As the rules are stateful, they also control the outbound traffic that's allowed to leave them. If multiple security groups are added to the same instance, the rules are aggregated to a single set. For simplicity, it is best to keep the number of security groups to a minimum. There is a maximum of five security groups that can be associated to an EC2 instance.

Every security group gets assigned a unique ID. This ID can become the Source of the inbound rules of another Security Group, which is the suggested way to authorize traffic between security groups and their associated instances.

How might this work?
If you have an application that requires a set of Web Servers and Database Servers, each type of server group gets its own security group. You can set your ingress security group rules to permit traffic from the Web Servers security group to reach the Database Servers group. Because these are stateful, the responses from the Database Servers will be allowed. When you add more databases and servers, your security group applies to this scalable option without an additional firewall needed.



AWS Has Two Options for Restricting the Flow of Network Traffic: Security Groups and Network Access Control Lists (NACLs)
Both are mechanisms for controlling access to AWS networks. Both use inbound and outbound rules to control network traffic in a VPC. In many situations, they represent two separate layers that complement each other in providing network security.

In general, Network Access Controls Lists (NACLs) can be thought of as a blunt instrument, only appropriate for a handful of use cases. Use Security Groups for authorizing fine-grained access between virtual machine instances or when controlling access between VPCs. If you find yourself with an either/or situation between security groups and NACLs, you should select Security Groups.
Let's compare each:

Security Groups:
- Controls traffic to or from an ENI (Elastic Network Interface), usually attached to an EC2 instance.
- Every EC2 instance must have a security group attached.
- A default security group is automatically created by AWS for each created VPC.
- Only *allow* rules can be applied to either ingress or egress traffic
- Security groups rules are stateful. When allowing ingress traffic, the corresponding outbound traffic is automatically allowed.
- The default state of Security Groups is to deny traffic, rules are used to selectively allow

Network Access Control Lists (NACLs):
- Controls traffic to or from a subnet.
- Uses inbound and outbound rules.
- Each subnet can have just one NACL applied with up-to 20 rules defined.
- Rules are numbered and applied in order; the first rule that applies is the rule that reigns.
- Both allow and deny rules are possible.

- NACLs are stateless. When allowing ingress traffic, the corresponding outbound traffic is NOT automatically allowed.
- The default network ACL explicitly allows all egress traffic from a VPC

NACLs are a collection of stateless filtering rules that are applied at the subnet level and applies to every resource deployed to the subnet. They are stateless because, if ingress traffic is allowed, the response is not automatically allowed unless explicitly allowed in the rule for the subnet. NACLs operate at the subnet level by examining the traffic entering and exiting the subnet. NACLs can be used to set both Allow and Deny rules.

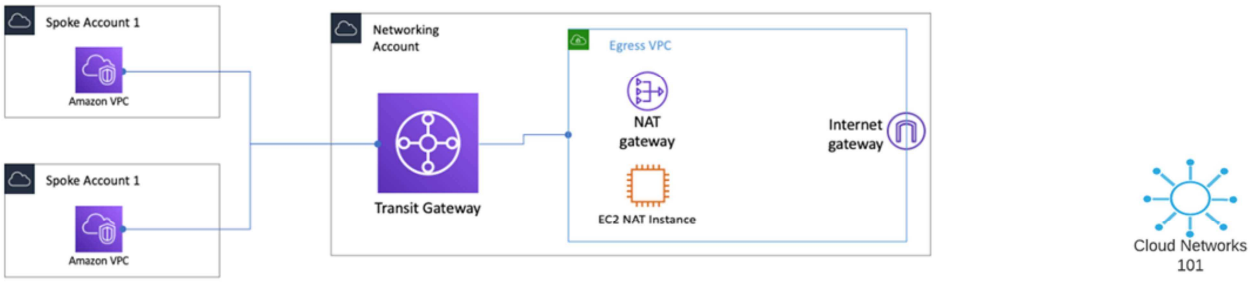
NACL Rule Evaluation:
Rules are numbered and evaluated in order, starting with the lowest numbered rule, to determine whether traffic is allowed in or out of any subnet associated with the network ACL. The last rule numbered is always an asterisk and denies traffic to the subnet. You only reach this rule only if no rules in the NACL list matches the affected traffic.

NACL Use Cases:
- If the need arises to broadly block inbound network traffic from a particular IP range.
- Shared VPC Architectures in order to restrict network traffic between subnets in different accounts.

AWS | Controlling Egress Traffic

Centralized Egress VPC or Egress Subnet

- Single NAT Gateway to manage
- Central point to enable Flow Logs to capture outbound traffic
- Pattern lends itself to an easier adoption of a vendor-provided network appliance should you need more than layer 3 controls on egress traffic



Controlling Egress Traffic in AWS
In a virtual cloud network, we can't ignore the outbound or egress rules. They remain a key component of network security even as workloads are moved to the cloud.

To provide a basic set of controls on the egress traffic, limit outbound access to include only the subnets that are required. For example, in a three-tiered web application, the app or database tiers are unlikely to require outbound access to the internet, so configure the egress security group rules to allow outbound network traffic only from those hosts needed for the proper functioning of the application.

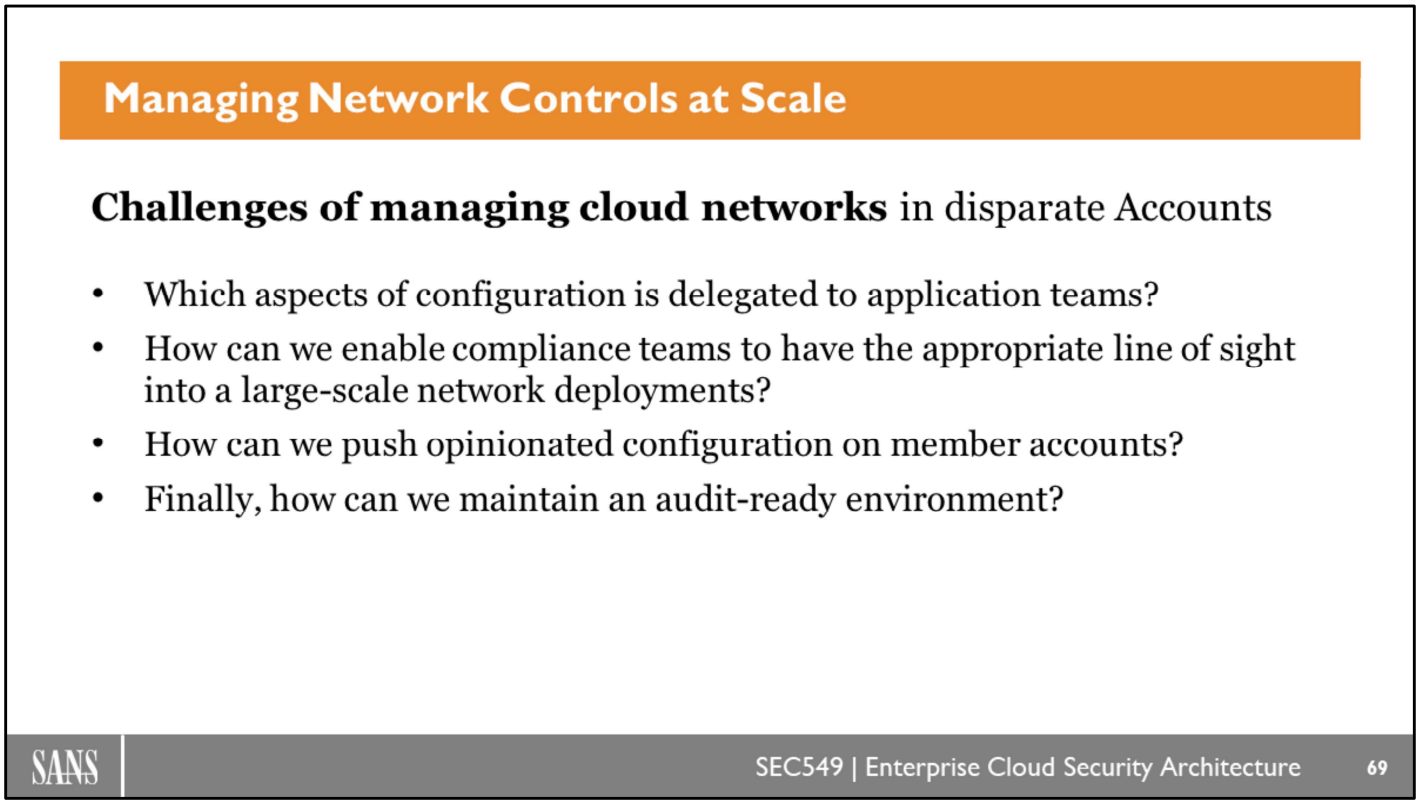
Where outbound traffic is required, AWS offers a cloud-native NAT Gateway, which is a managed network address translation resource used to ensure traffic only involves initiated outbound data and to make sure that private subnets cannot receive inbound traffic. The NAT gateway is only a layer 3 device. It can be used as an egress point for network traffic by an IP subnet, but not as an egress target originating only from certain EC2 instances in a subnet. Due to this limitation in layer 3 controls, there has been a rise in the 'egress VPC' or 'egress subnet' pattern.

Egress VPC
An egress VPC (or egress subnet in shared-VPC models) is a VPC where all outbound traffic is funneled to via route tables. In addition, the mechanism for connecting disparate VPCs together traditionally involves using Transit Gateways. You'll want to consider centralizing egress traffic for the following benefits:

- Maintaining fewer NAT Gateways in your overall architecture is more cost-effective
- Having a central point for the collection of Flow Logs configured at your NAT gateway and DNS Query Logs.
- Improving the inspection of egress traffic

Having a central egress point allows for easier adoption of vendor solutions, which allows for the inspection of egress traffic. Such a process would operate higher up the OSI Model, providing more granularity over all aspects of egress.

Reference:
Image Source: Centralized egress to internet - Building a Scalable and Secure Multi-VPC AWS Network Infrastructure https://sec549.com/id136



Managing Network Controls at Scale
A more important consideration (beyond the configuration of individual network controls) is the management of those controls at scale.

A few of the questions we will address on the upcoming slides include:

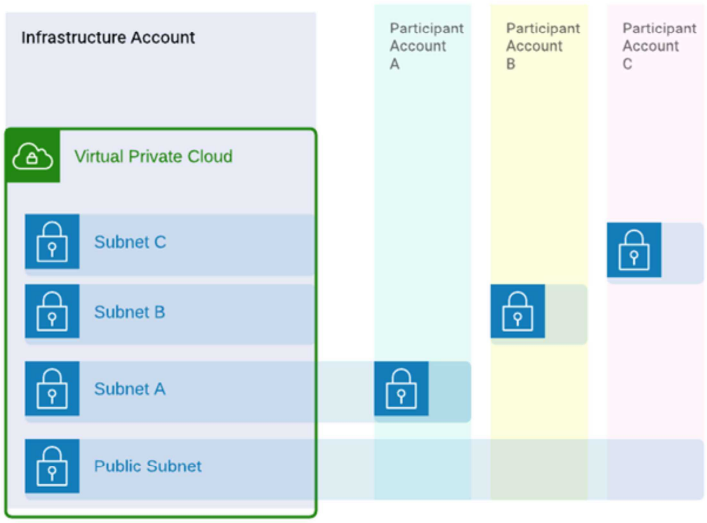
• How can we as security professionals enable compliance teams to have the appropriate line of sight into a large-scale network deployment?
• How can we push opinionated configuration on member accounts?
• Finally, how can we maintain an audit-ready environment?

It will come as no surprise to anyone who has taken other SANS cloud classes that treating your configurations as code and auditing for drift will take center stage in the upcoming slides. The necessity of having continuous audits often arises when application teams own their owner Security Group Rules. Their objectives aren't always aligned with security policy, or it becomes difficult to have central visibility into the configuration requirements. Because of these issues, designing for the central monitoring of network controls should be a priority when building cloud networks where ownership over those controls will necessarily be dispersed.

AWS | Shared VPCs

Requirements

- Designate an Account to house networks
- Enable VPC Sharing with Resource Access Manager (RAM)
- Enforce SCP to prevent overly permissive sharing



What is VPC Sharing?
VPC sharing[1] involves creating VPCs in a parent AWS account and sharing the subnets across multiple, participant accounts. The participant accounts can build resources in these shared subnets, but the parent VPC resource is not visible outside its home parent account. Any rate limiting imposed by AWS applies to the participant accounts, not to the single parent account. Since the VPC owner centrally controls routing and Network Access Control Lists (NACLs), it is possible to enforce strict security segmentation even within the same VPC.

The first step toward hosting a shared network is centrally defining the details of the network, such as its subnets, internet gateways, and routes. How this is accomplished in AWS is through leveraging the service's AWS Organizations, Resource Access Manager, and Service Control Policies to create the concept of the 'Shared VPC'.

AWS Infrastructure Account:
A central account where VPCs, subnets, Internet Gateways, and Routes are defined and managed by the organization's network team.

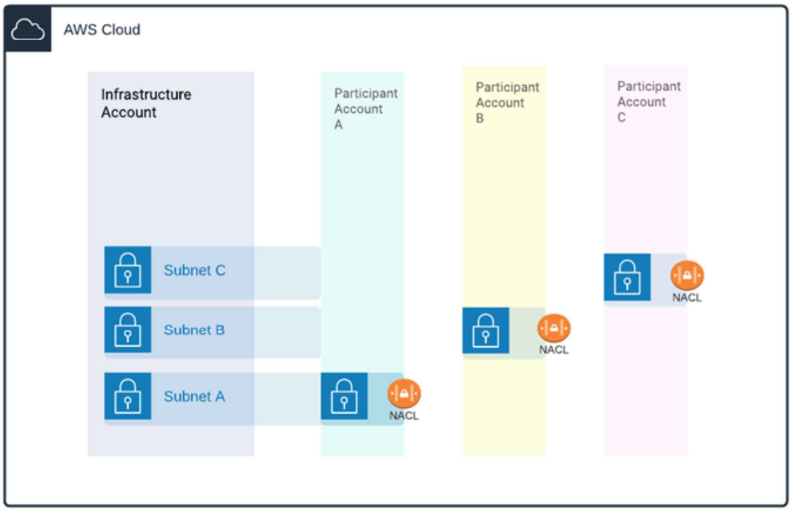
Resource Access Manager:
Resource Access Manager (RAM) is the service from AWS that lets you create a resource in one account and share it with another. It's with RAM that the VPC components can be centrally created by a 'Owner' and shared among 'Participant' accounts. Often this creates a one-to-many relationship where there is single VPC Owner with many Participant Accounts. However, this can just as easily reflect a one-to-one relationship in that a single VPC Owner shares subnets to only a single 'Participant' account.

Service Control Policies:
Overly permissive sharing with RAM needs a guardrail SCP policy.
• Allow only VPC subnet shares created from a network account
• Allow VPC subnet shares to be shared only with certain OUs or AWS accounts

Reference:
[1]: https://sec549.com/id137

AWS | Achieving Network Segmentation with NACLs

- Block inbound and outbound network traffic between shared subnets in **different accounts**
- Explicitly allow inbound and outbound network traffic between subnets shared to the **same account**
- Use **Security Groups** to enforce more granular network controls



SANS | SEC549 | Enterprise Cloud Security Architecture 72

Achieving Network Segmentation with NACLs
By default, a NACL is attached to every subnet and allows all inbound and outbound traffic. Without a clear-cut use case, it is recommended to leave the NACL rules on their default settings.

When are custom NACLs required?
The use of NACLs to restrict network traffic is a control that MUST be leveraged if using a Shared VPC pattern on AWS.

Shared VPC Architecture in AWS uses subnets as if they were VPCs (allotting the network blocks to participant accounts). NACLs will be the only mechanism to restrict network traffic between subnets and thus between AWS accounts.

Denying Traffic Between Subnets
A common use for NACLs is to deny network traffic between shared subnets that reside in different accounts[1]. The only way to accomplish subnet isolation is to update the default NACL attached to every subnet. Update each shared subnet NACL to block inbound and outbound traffic to subnets in different accounts, while allowing inbound and outbound traffic to shared subnets in the same accounts. Having a complex network configuration via NACLs is ideal for automation where the settings can be codified in code and the details can be regularly audited.

Reference:
[1]: https://sec549.com/id138

AWS | How Many VPCs Do You Need?

Technical Limits of a Single VPC

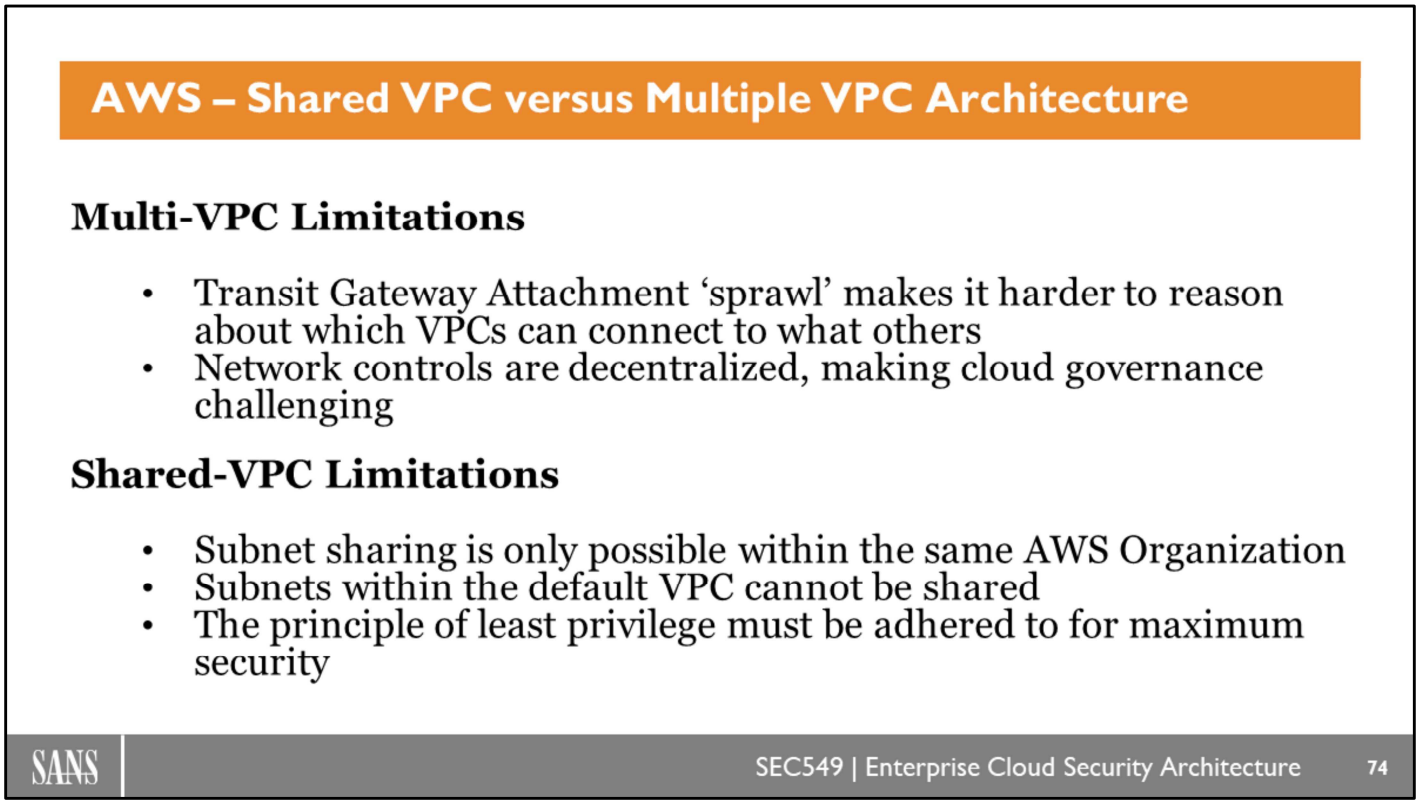
- 100 Participant Accounts per single, shared VPC (up to 1k with quota increase request)
- 200 VPC subnets per single, shared VPC (up to 1k with quota increase request)
- 5,000 ENIs per single, shared VPC (up to 70k with quota increase request)

Rational Use of Shared VPCs

- Use a single, Shared VPC for accounts of similar privilege levels
- Group accounts which need to cross communicate frequently in the same shared VPC
- Link Shared VPC architectures together with Transit Gateways

How many VPCs Do You Need?
When thinking about scaling your shared VPC environments, what hard, technical limitations or resource exhaustion do you run into? As it turns out, the quotas from AWS are very generous. So, with the ability to create a single VPC and share subnets and slices of IP space to multiple accounts, the question might be raised: can I run my organization on a single VPC?

Just because you can, doesn't mean you should
Operating a large-scale organization from a single, shared VPC increases the blast radius of any malicious incident or unintentional outage. Use the Shared VPC construct for grouping like accounts. In some cases, a VPC owner may choose to share with only a single participant account.

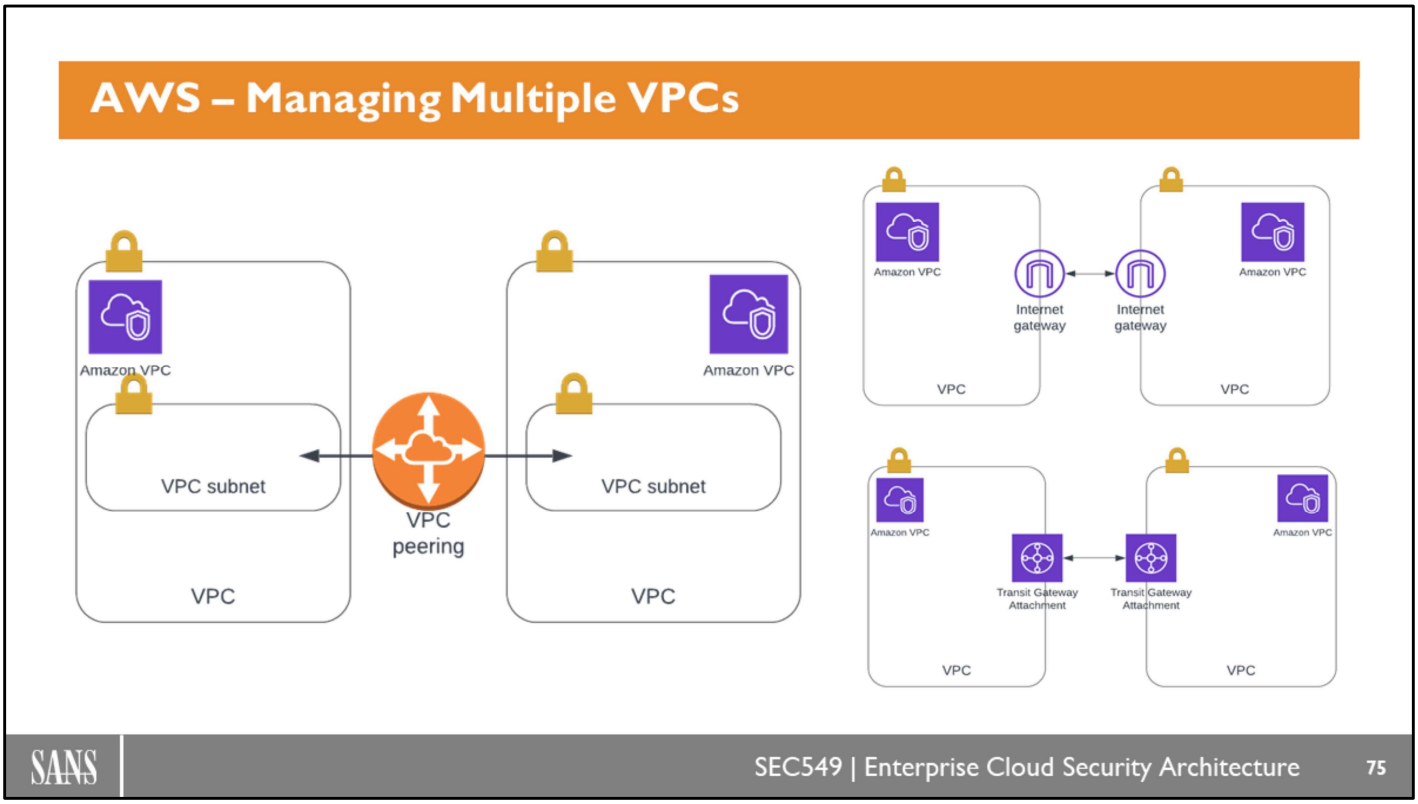


Multi-VPC Pattern
Since the VPC was first introduced in 2009, if an organization had more than one Account, the network pattern has been defined as the 'Multi-VPC Architecture'. This is a pattern that encourages every AWS Account to host their own, if not multiple VPCs, leaning into the natural network-layer isolation a VPC affords.

What has been the challenge with this pattern?
- Sprawl of Transit Gateway Attachments interconnecting various VPCs, making it difficult to reason about which VPCs can route to what others[1]
- A decentralized view of network-controls, making cloud governance difficult.

Shared VPC Limitations
It has been only more recently, with the introduction of Resource Access Manager (RAM) in 2018, that the Shared-VPC has become an option for customers. However, it is not appropriate in all situations.

Reference:
[1]: https://sec549.com/id139



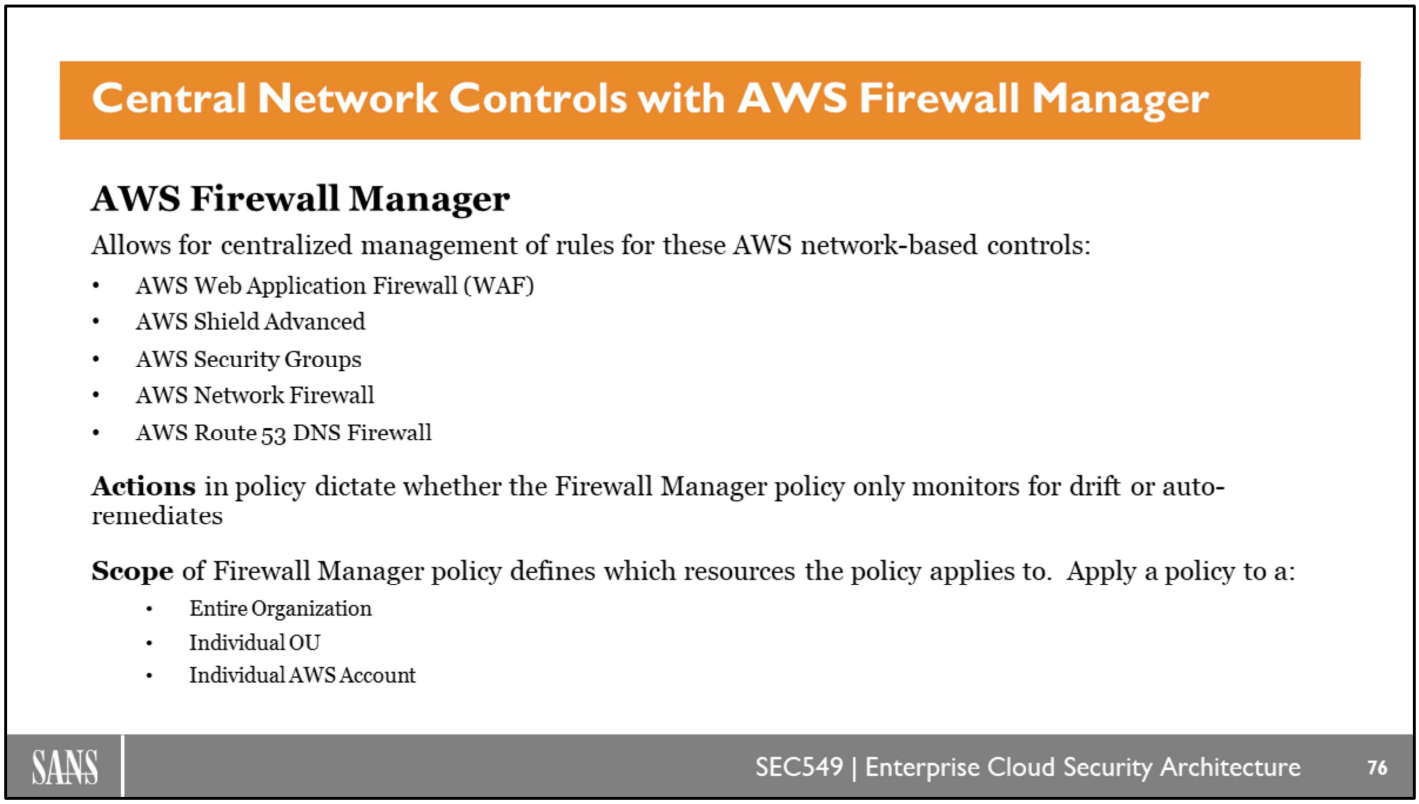
Connecting Resources in Multiple VPCs
If your organization has more than one VPC, you will need to authorize connections between them. There are several options for creating this kind of connectivity:

Option 1: Deploy Internet Gateways
This involves giving each VPC an internet gateway and then routing all traffic out through one VPC's gateway into the internet and then back into the other VPC's gateway. This is potentially unnecessary if internet traffic isn't otherwise needed but can be a cost-effective way to move data out of a VPC.

Option 2: Make use of VPC Peering
A VPC peering connection is a networking connection between two VPCs that enables you to route traffic between them using private IPv4 addresses or IPv6 addresses.

VPC Peering allows direct 1:1 communication between VPCs – even those in different regions, if needed. Because it is a 1:1 connection, you cannot have transitive routing (using one VPC to route packets between two other VPCs, for example). You need direct connections between each VPC that needs a connection with another. This can make VPC Peering pattern difficult to manage and scale if you have hundreds of VPCs you need to connect. You can have up to 125 peering connections per VPC. From a cost perspective, VPC Peering tends to be the lower cost option for connecting the IP space of VPCs rather than using a Transit Gateway.

Option 3: Make use of Transit Gateways
AWS Transit Gateway connects VPCs through a central hub acting as a cloud router. A Transit Gateway can be used as a routing target for traffic between VPCs or connected on-prem networks linked via Direct Connect. Patterns using a Transit Gateway will often use a hub and spoke design. This is the way you can connect hundreds of VPCs through a single Transit Gateway and route traffic all though a single point. Connecting VPCs through Transit Gateways might create a more elegant pattern but it tends to cost more than VPC Peering, as AWS customers are charged both for traffic volume and for each attachment point.



Cloud-Native Tools for Managing AWS Network Rules

In 2020, AWS released AWS Firewall Manager as a single interface you can use to manage the various network controls for your resources. Firewall Manager was created to solve the problem of sprawling, network controls and configuration rule sets scattered across hundreds or thousands of accounts. You can use Firewall Manager to push policies across any slice of your AWS Organization, report on the compliance state of your network controls, and enforce a predefined state.

AWS Firewall Manager[1] allows for the management of multiple network-centric tools at once, including the following:

- **AWS WAF policies –** this helps you monitor your https requests into public AWS endpoints. AWS Firewall Manager allows you to attach WAF rules to resources based on certain conditions rather than attaching them one at a time to your resources. You can determine which IP addresses you will accept and can even run CAPTCHA if you choose before accepting a request.

- **AWS Shield policies –** managed service to protect applications against DDoS attacks

- **Security Group policies** – Firewall Manager helps you determine which security groups to apply to what instances and monitors your compliance with the policies created.

- **Network Firewall policies –** provides network-level protection of your VPCs.

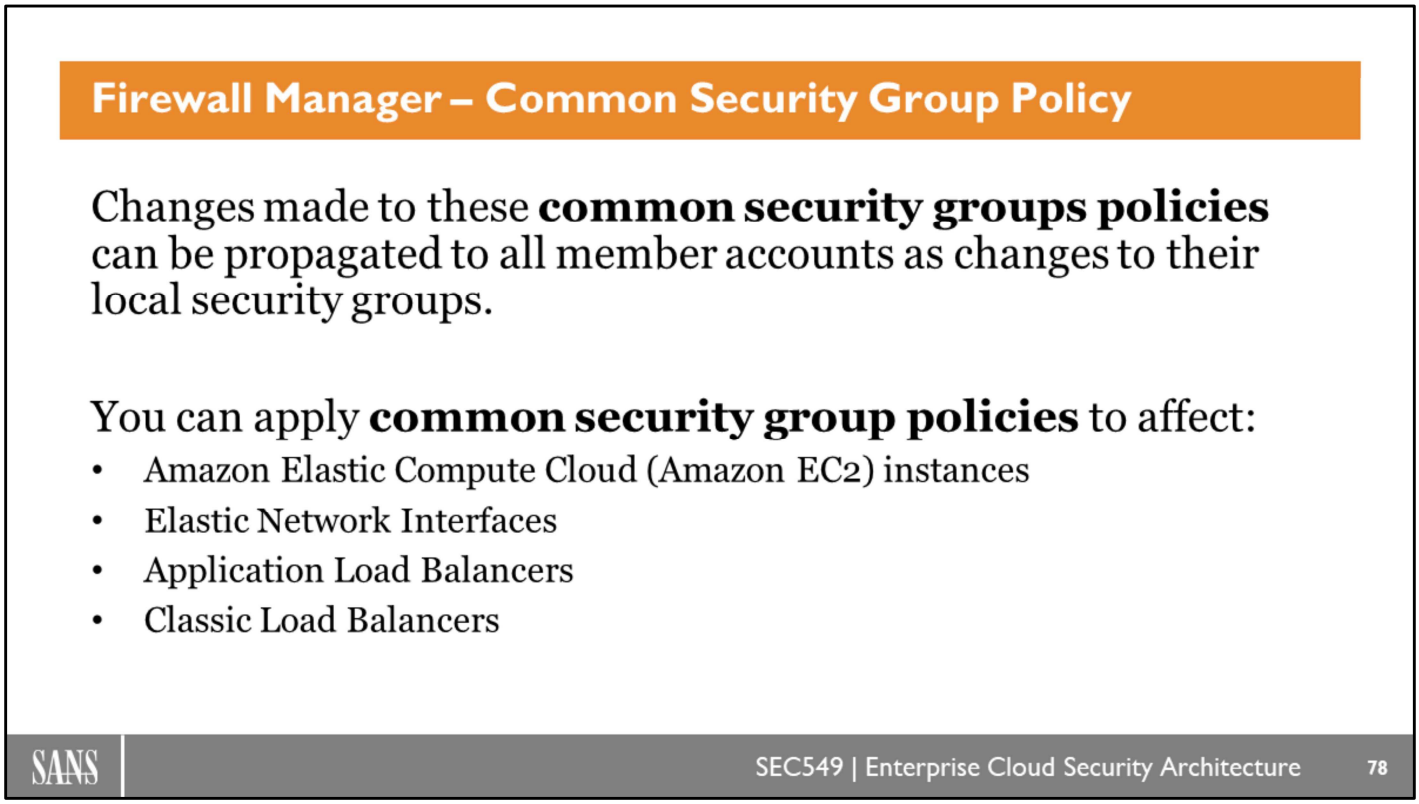
- **Amazon Route 53 Resolver DNS Firewall policies –** this is where you'll apply your DNS Firewall protections to your Organization's VPCs.

When working with any of the services integrated with Firewall Manager, you can centrally configure the policies and rules for your Organization - pushing down configurations across many accounts or resources or creating audit policies within Firewall Manager to benchmark your security posture.

Pre-requisites to Using Firewall Manager

- AWS Organizations needs to be enabled in Full-Featured Mode. You cannot use Firewall Manager without having the Organizations service enabled.
- AWS Config service needs to be enabled as under-the-hood, monitoring and enforcement is done by Config.
- A unique, separate account should be designated as the Firewall Manager administration account.

Reference:
[1]: https://sec549.com/id140



Configuring Common Security Group Rules with AWS Firewall Manager
AWS Firewall Manager integrates with several network-centric services including AWS WAF, VPC Security Groups, AWS Shield, and Route53. In this course, we'll specifically look at how to use Firewall Manager to affect VPC Security Groups across an organization.

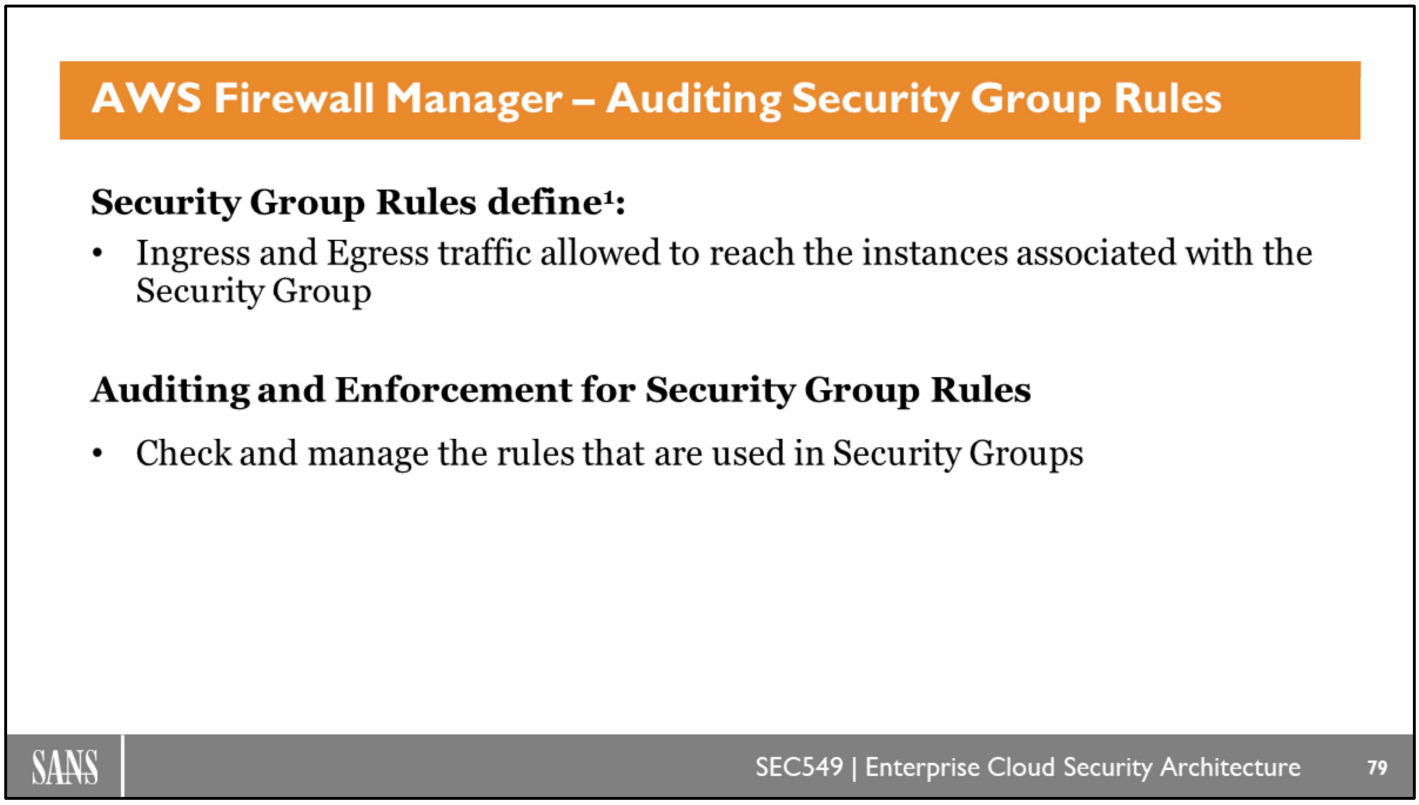
Common Security Group Policies
A common use of Firewall Manager is to leverage it to push a common set of security group rules across accounts. A network security team, for example could define a master set of Security Group Rules by using 'Common Security Group Policy'.

With 'Common Security Group Policy[1]' and Firewall Manager you can push down Security Groups Rules to member accounts. Once a policy is applied, Firewall Manager can either monitor for any changes against the policy or auto-remediate. This policy will not only define the ingress and egress rules but also the scope – the AWS Accounts, EC2 instances, or resources by tags the ingress and egress rules should apply to.

When applied to member accounts, the individual account owners are free to change these centrally-enforced rules. Here is where the 'action' portion of your 'Common Security Group Policy' comes into play. With Network Manager, you can have a Common Security Group Policy to be applied in monitor-mode, which would allow changes to be made; however, their non-compliance would be reported. You could also place the Common Security Group Policy in auto-remediation mode, which reverts changes made by individual account owners.

It is not recommended to apply policies in auto-remediation mode without first testing the affects compressively.

Reference:
[1]: https://sec549.com/id141



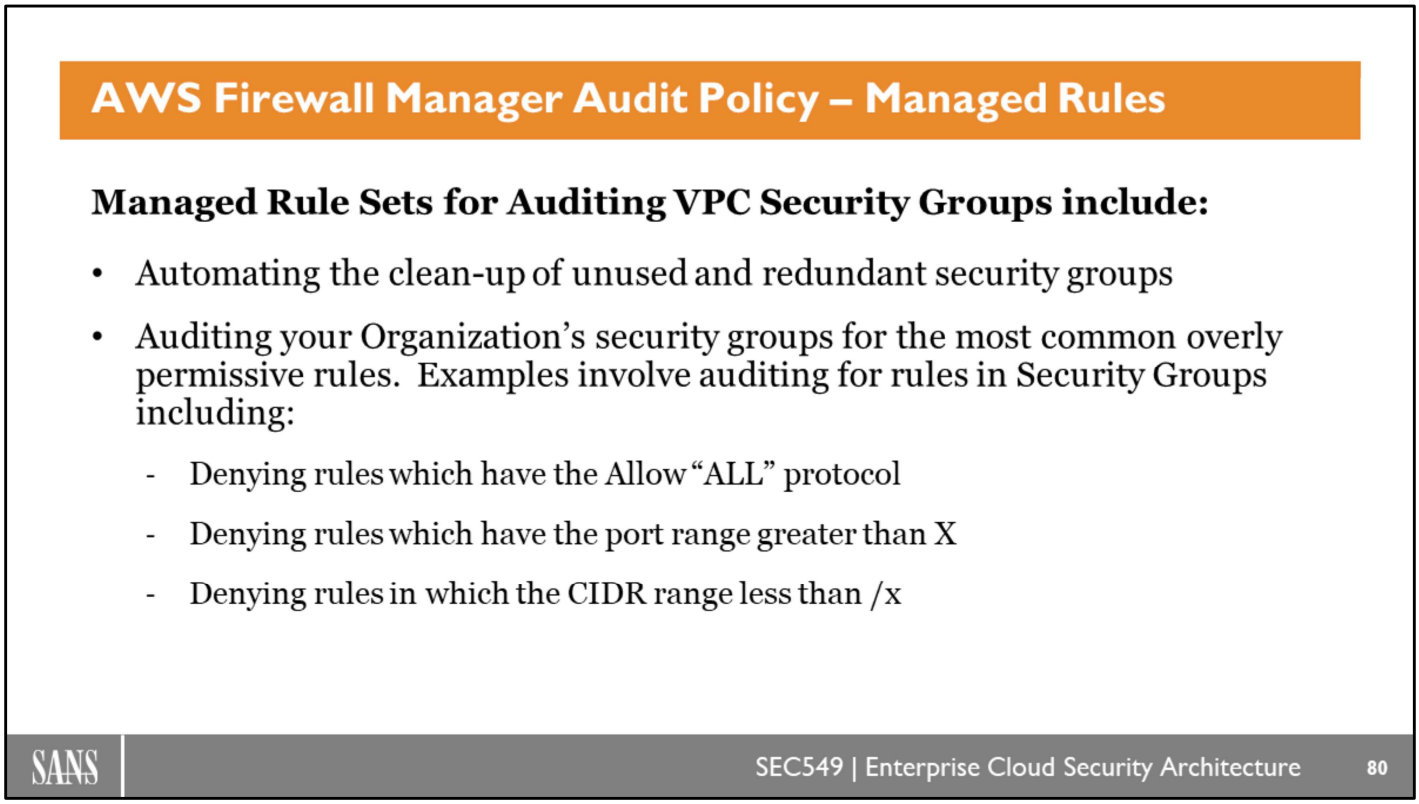
Auditing Security Group Rules
When you create Firewall Manager Audit Policies in AWS, you can define the guardrails that member accounts need to operate in while also monitoring for misconfigurations.

Components of Firewall Manager Audit Policy:

- ***Rules*** – a set of rules to audit for. Examples include rules restricting all public access or only allowing SSH from specific IP ranges.
- ***Actions to take*** - monitor or auto-remediate
- ***Scope to apply the policy to*** – The scope can include the entire Org, OU, Accounts, or only certain resources with specific Tags.

Under the hood, Firewall Manager Policy is using the service called AWS Config to monitor for deviation in rules and enforce auto-remediation.

Reference:
[1]: https://sec549.com/id142

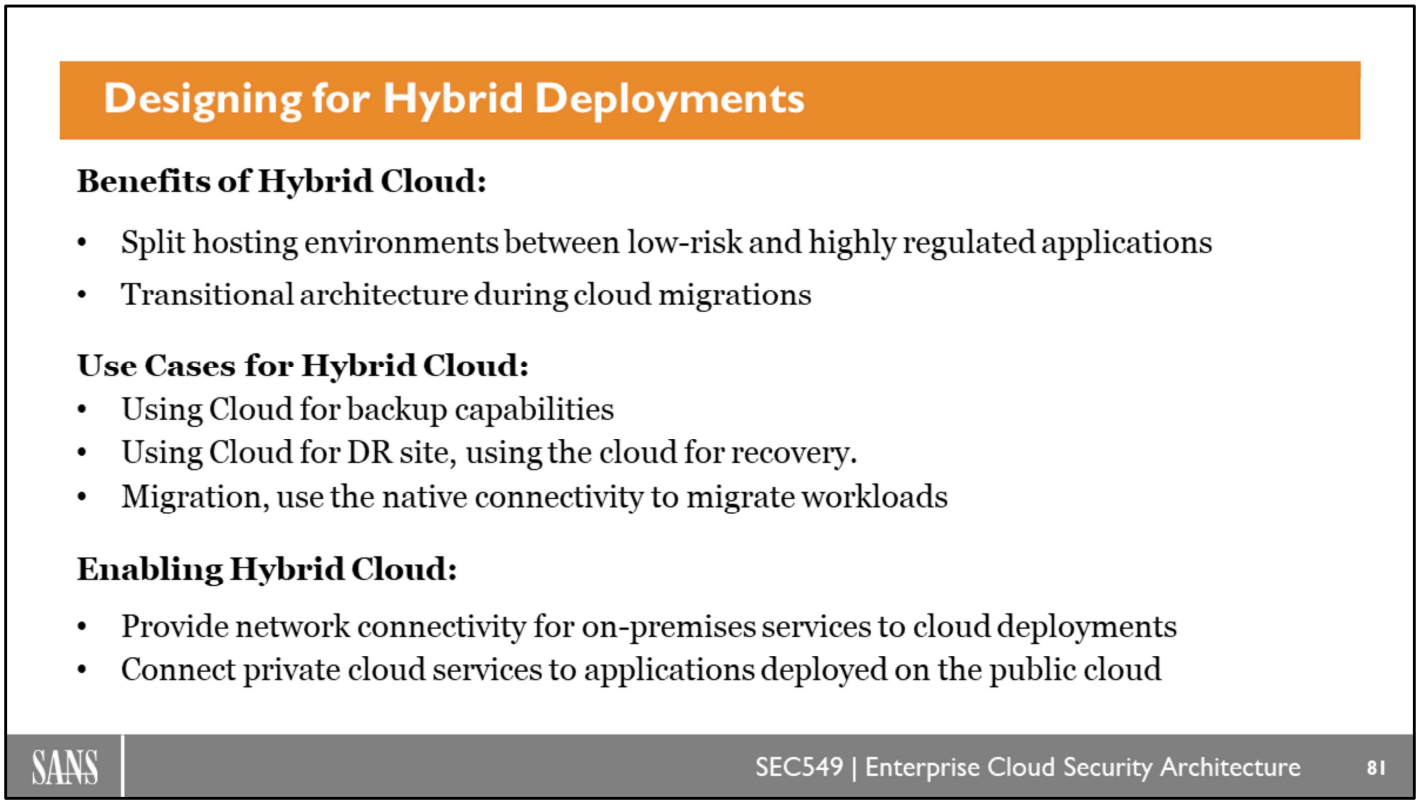


Managed Rule Sets for Auditing VPC Security Groups
There are two broad types of Firewall Manager Audit Policies. These include Custom Audit Policies and Managed Audit Policies. Managed Audit Policies are those that AWS creates and manages on your behalf.

Often, organizations will not have the spare cycles to manually audit the ingress or egress rules in individual accounts or even create Custom Rule Sets to set guardrails. To get basic coverage on your member accounts, Security Groups can catch some low hanging fruit, and you can use a tool like the built-in Managed Rule Sets for AWS Firewall Manager.

With Managed Rule Sets[1] you can quickly clean up unused and redundant Security Groups and audit member accounts for common, overly permissive rules without building out any custom rules.

Reference:
[1]: https://sec549.com/id143



Benefits of Hybrid Clouds
The ideal hybrid cloud environment offers the optimal mix of computing resources, storage buckets, and services using a 'hybrid' of (generally preexisting) on-premises infrastructure combined with private cloud services offered by a public cloud provider, such as AWS, Microsoft Azure, or GCP. The hybrid environment effectively orchestrates the various apps, databases, and services among the different platforms. Your infrastructure is referred to as 'hybrid' if it combines the various platforms within the same data center.

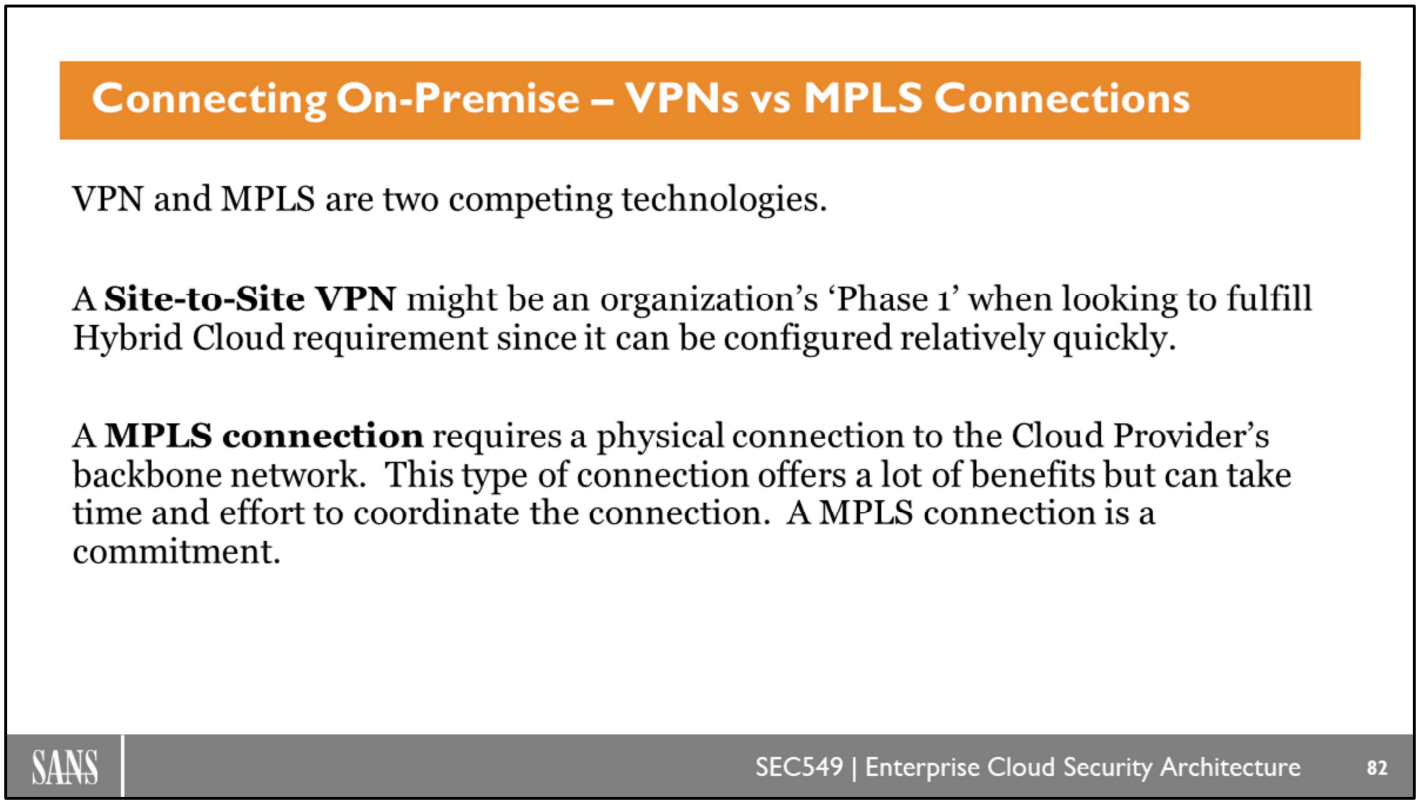
The advantages of being hybrid include:

- You can maintain different technology stacks for your disparate or varied business needs
- You can enable low-risk applications to accelerate within the cloud while maintaining tried-and-true controls over high-risk applications or those with unique compliance requirements
- You can ensure that applications stored onPrem and in the Cloud can maintain tight integration during often long transition states as an organization moves to the cloud

Use Cases for Hybrid Architecture:
- Using the Cloud for its backup capabilities
- Using the Cloud for DR Site
- Using the cloud for recovery
- During migration, using the cloud-native connectivity to migrate workloads

Enabling a Hybrid Cloud
One of the unique challenges to utilizing a hybrid cloud architectures involves the complex network designs required to support onPrem-to-Cloud or Cloud-to-onPrem network traffic.

Considerations need to be made to not only have on-premises applications with requirements to connect to cloud VPC-bound private resources, but also to have private services in the cloud needing connections to your on-premises applications.



Cloud VPNs versus MPLS Connections
Cloud Providers generally support two mechanisms to connect your virtual cloud network to an on-premises network. Let's review these two options holistically before diving into the specific offerings from the cloud providers:

You can connect your on-premises network to the Public Cloud with a **site-to-site VPN.** VPN connections between two gateways consist of the creation of a tunnel between two networks. It is deployed as an encrypted network tunnel through which data travels from one point to another across the open internet. Connectivity via VPN is likely to be slower than an MPLS connection. VPNs also have more unpredictable costs and do not scale well with increased traffic.

You can also extend your on-premises Private Cloud to the Public Cloud with a **MPLS Connection.** Multi-Protocol Label Switching (MPLS) changes the way packets travel. Conventionally, MPLS is designed more for speed than a VPN is.

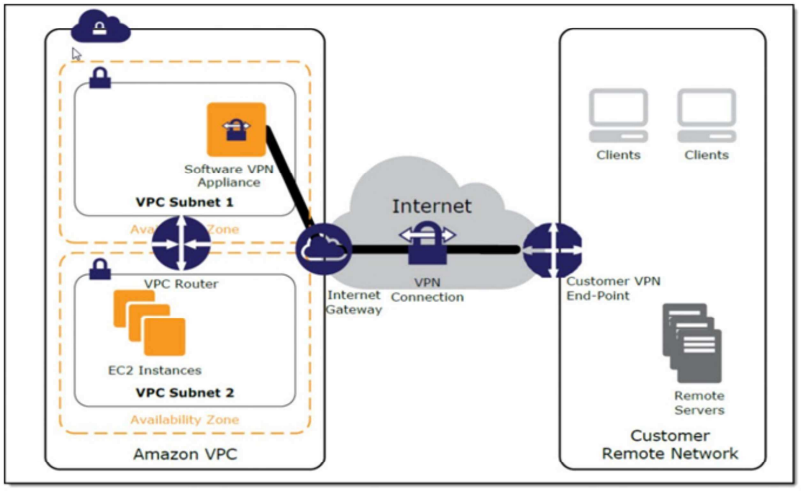
MPLS Benefits:
- Reduces your data-out rates
- Consistent network performance
- Low-latency, high through-put

All major cloud providers offer services for both VPN connections and MPLS connections to their backbone network. In the upcoming slides, we outline the specific offerings from AWS.

Connecting On-Premise – VPN Options

Two possible scenarios:

- Site-to-Site VPN – with **AWS-Managed** VPN
- Site-to-Site VPN – with **Customer-Managed** VPN



Connecting Cloud Networks to On-Premises Networks Using Virtual Private Network (VPNs)
AWS has a couple of options for doing this, one being more streamlined (AWS-Managed VPN) while the other leaves more for the customer to configure (Customer-Managed VPN).

Site-to-Site VPN – with AWS Managed VPN:
Every AWS-Managed VPN is a combination of 1 connection and 2 tunnels per VPC to provide you with high availability with your VPN tunnel.

The encrypted tunnel is an IPSec site-to-site tunnel with AES-256 encryption and supports either static or dynamic routing (BGP).
Site-to-Site VPN with AWS supports Traversal NAT (NAT-T), which is needed in situations where your VPN server on-premises sits behind a NAT device.

An AWS Managed Site-to-Site VPN should be terminated at a virtual gateway (VGW) when associating the tunnel with a single VPC. In this case, the way AWS-Managed VPN integrates with your VPC is fairly straightforward.

When you create a virtual gateway (VGW) and associate it with a VPC in AWS, you can configure which subnets will utilize the VPN connection, limiting which blocks of IP space will have direct routes back to your on-prem network.

If multiple VPCs need to share a VPN connection, you can opt to terminate the tunnel at a Transit Gateway instead.

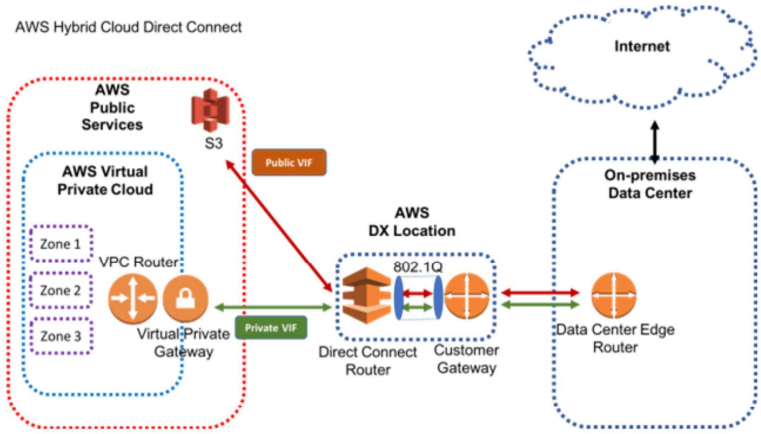
Site-to-Site VPN – with Customer-Managed VPN:
With this version, you are required to bring your own VPN software. This could be from the AWS Marketplace or any other source and deploy it on an EC2 instance.

With a Customer-Managed VPN, you are responsible for *segmenting* the VPN appliance into its own subnet, *configuring* routing to subnets you wish to participate in the connection, *maintaining* patch-levels of the EC2 Instance and VPN software, and *configuring* the system for high availability.

Connecting On-Premise – AWS Direct Connect

AWS Direct Connect

- A method of physically connecting your on-premises network to a regional AWS Direct Connect site
- Allows for multiple types of Virtual Interfaces (VIF) which you can use to leverage the direct network connection into the AWS backbone network to solve different challenges



AWS Direct Connect
AWS offers the AWS Direct Connect service to provide a physical linkage between your on-premises network and an AWS Direct Connect location. Direct Connect is a physical linkage between your on-premises environment and the AWS backbone. This can be accomplished by maintaining a network appliance in the same facility as an AWS Direct Connect endpoint (dedicated connection) or working with a Direct Connect partner who can facilitate this connection (hosted connection).

Once you've established your Direct Connect linkage with AWS, you can leverage this to create three different kinds of interfaces.

Direct Connect Virtual Interfaces
Each of these interfaces can be used to solve different networking challenges.

- Private VIF – Used to connect your on-premises network to your AWS VPCs using private IP addresses.
- Transit VIF – Used to connect your on-premises network to multiple VPCs using private IP addresses – by terminating at a Transit Gateway
- Public VIF – Used to connect any public AWS service BACK to your on-premises network using public IP addresses.

Direct Connect Global Access
AWS has connected their Direct Connect colocation facilities together, enabling Global Access. Global Access allows customers to have a single Direct Connect connection into the AWS network and have private access to all AWS public services across all regions except China.

References:
[1]: https://sec549.com/id144
[2]: https://sec549.com/id145
Image source: https://thecloudcto.com/aws-hybrid-cloud-direct-connect/ https://sec549.com/id146

Private Access to Public Cloud Services

Connecting VPC-Bound resources with Public Services

- VPC-bound Resources
  - EC2 Instances, RDS Instances
  - Reside within your VPC/subnet
  - Network-layer controls available in the form of Security Groups and Network ACLs
  - Identity-layer controls available in the form of IAM Policies
- Public AWS Services
  - Lambda Functions, S3 Buckets, SNS Topics, Kinesis Data Streams, DynamoDB Instances
  - These resources cannot be encompassed by a VPC
  - APIs to connect to these resources are by public by default
  - Only use identity-layer controls available in the form of IAM Policies, both identity-based and resource-based

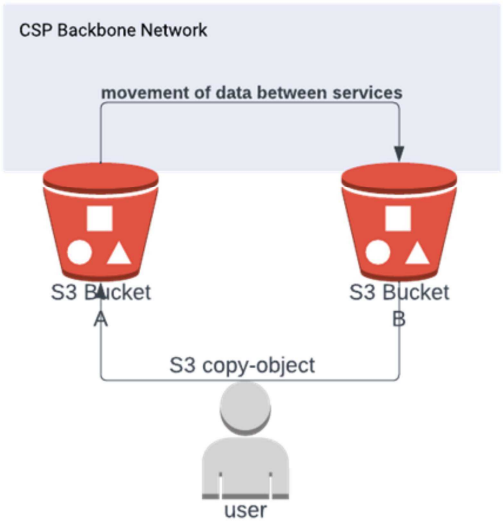
Private Access to Public Cloud Services
If you use the full breath of cloud services available, the need will arise to connect private, VPC-bound resources with public services, those not constrained by network controls.

The next group of slides is dedicated to the solutions AWS offers to allow communication between resources in private subnets to public APIs.

What Is the Backbone Network & Why Does It Matter?

Understanding your CSP's backbone network helps you:

- Maintain private connectivity between isolated networks
- Control for egress on the resource backend network



SANS | SEC549 | Enterprise Cloud Security Architecture 87

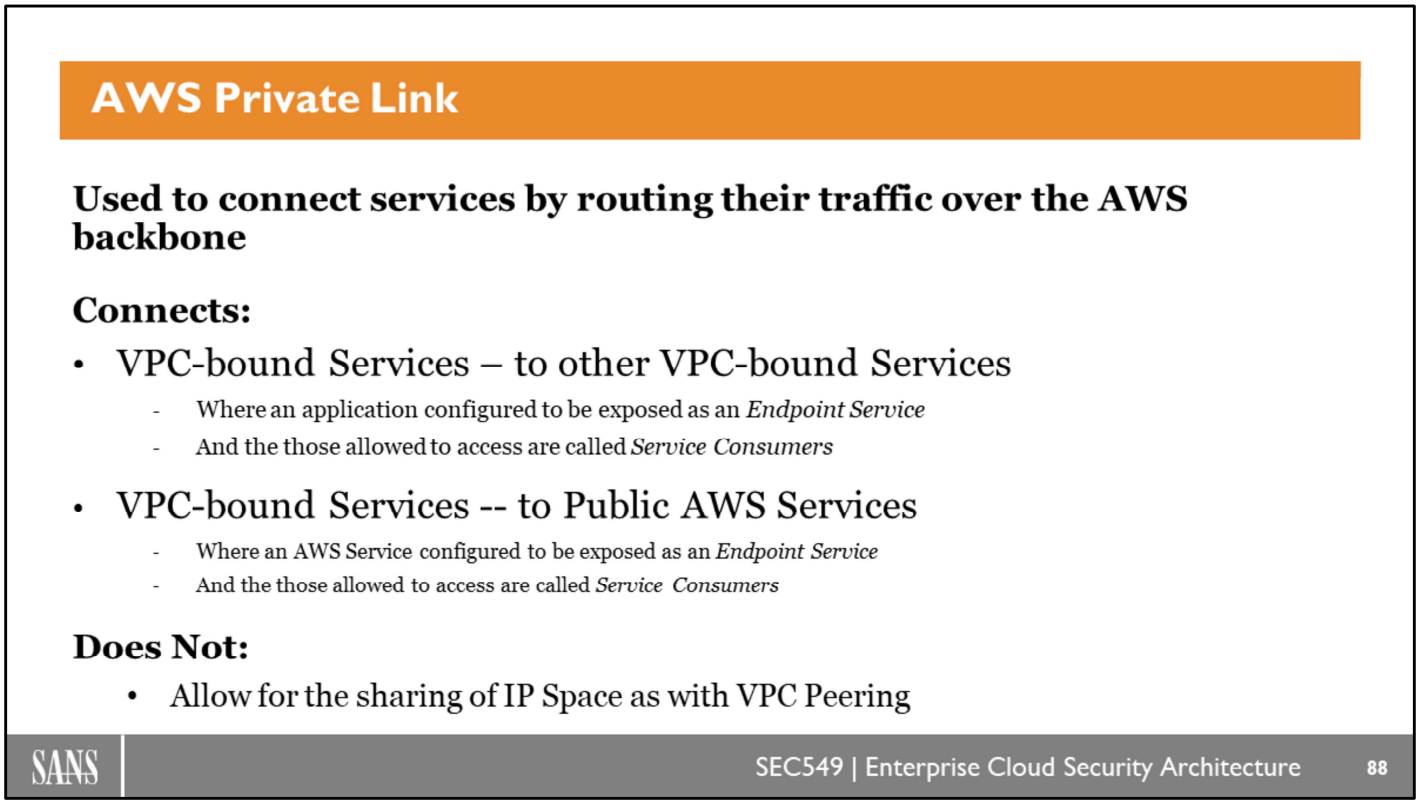
The CSPs and Their Backbone Network
The Cloud Providers backbone network is the substrate network on which their software-defined networking is built. It is a separate network from the public network, often connected directly with the ISP to help latency. This is the network where service-to-service data movement will occur. The backbone network can be leveraged using services like AWS PrivateLink to route otherwise public network traffic over the CSPs private networking stack.

Data Exfiltration on Multi-Tenant Cloud Services
Multi-Tenant Cloud Services are resources like PaaS services (BigQuery) or storage services like S3 or dynamoDB. They carry with them a different threat model than single-tenant cloud services like basic compute resources (ec2 instances, computes instances, VMs). There is broad, network-level access to multi-tenant services over the public internet and on the resource's backend network.

Movement of data between services, even between different types; occurs over your CSPs backbone network and is not subject to typical firewall policies. This scenario makes your CSPs backbone network an exfiltration path of concern.

In response to this gap, Google developed VPC Service Controls[1], allowing customers to define a network perimeter around their PaaS Services. This features restricts the movement of data and brings some of the same network-layer guarantees to multi-tenant services that their customers expect with VPC-bound resources. There is no AWS or Azure equivalent service to GCP VPC Service Controls.

Reference:
[1]: https://sec549.com/id147



AWS PrivateLink
You don't need an internet gateway, a NAT device, or a virtual private gateway to enable outbound communication with AWS PrivateLink.

AWS PrivateLink establishes a secure connection between two AWS Virtual Private Clouds (VPCs). The VPCs can belong to separate accounts, i.e., a service provider and its service consumers. AWS routes the PrivateLink traffic within the AWS data center and never exposes it to the public internet.

The advantage of using AWS PrivateLink is that you automatically have a secure and private network without needing to manage VPC CIDR blocks, VPC peering connections, or internet gateways.

PrivateLink connections are often used as an alternative to VPC Peering. Unlike VPC Peering, PrivateLink allows VPC resources to communicate with each other using private IP addresses, without requiring gateways, VPN connections, or separate network appliances.

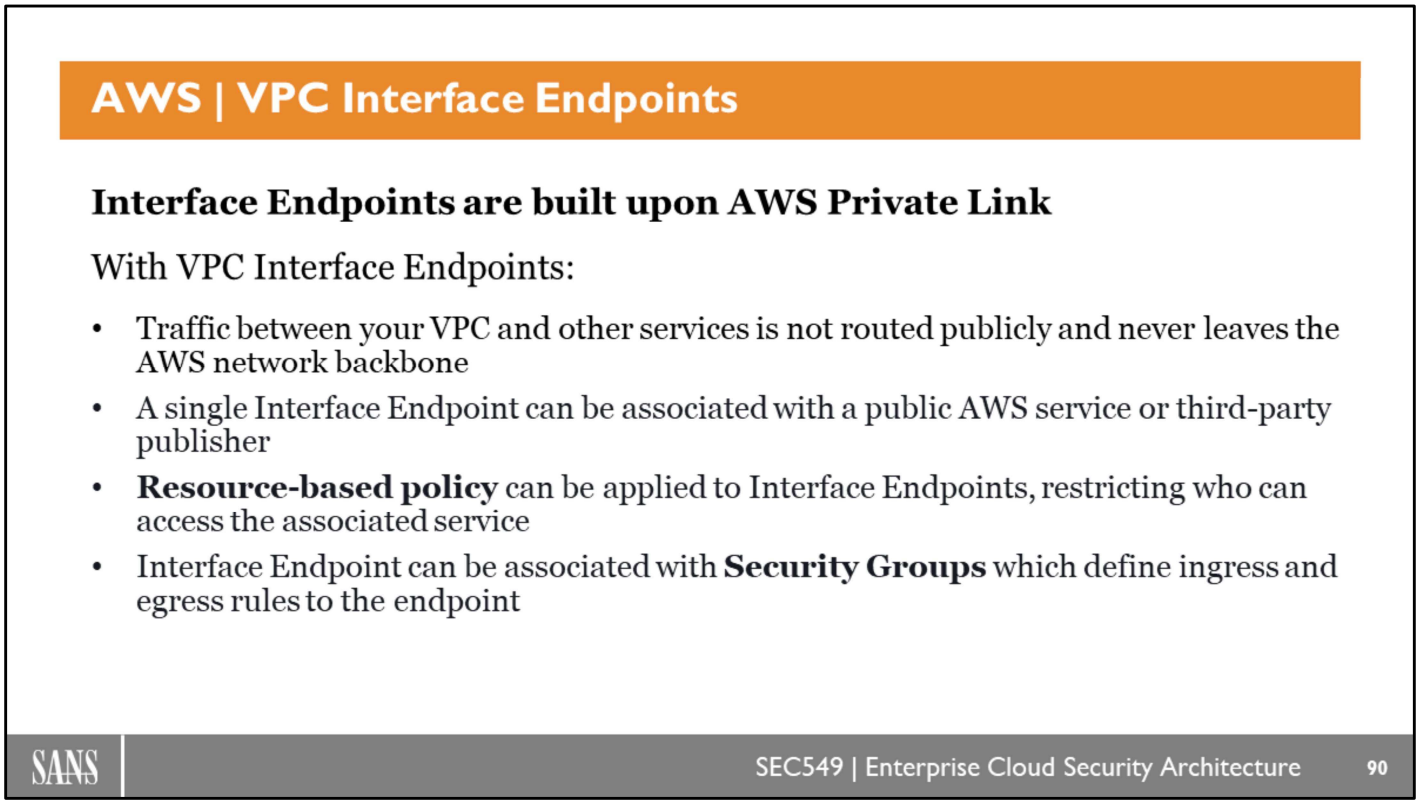
Another advantage of connecting disparate resources via PrivateLink as opposed to VPC Peering is the ability to set policy on PrivateLink VPC Endpoints. When resources in a VPC are connected by configuring a VPC Peering connection, there is no ability to set additional IAM policy regarding how the resources are allowed to interact.

AWS | Endpoints – Interface, Gateway & Load Balancer

- Mechanism for accessing public services from private, VPC-bound resources
- Depending on the type, network traffic from private subnets to public services is either routed over the AWS backbone or traverses the open internet
- Endpoints **only** allow traffic to be initiated from the service consumer side
- Three varieties of endpoints are available:
  - Interface Endpoints
  - Gateway Endpoints
  - Gateway Load Balancer Endpoints

Connecting VPC-Bound and Public Resources in AWS

For secure access to AWS Resources or third-party services, you can use *interface*, *gateway, or gateway load balancer VPC endpoints*. While each option has nuances involved in their implementation and available services, they all exist to achieve the same goal, connecting otherwise private resources with no route to the internet to public resources like S3 Buckets or SNS Queues. Prior to the introduction of VPC Endpoints, all VPC-bound resources required a connection to the internet to communicate with public services.



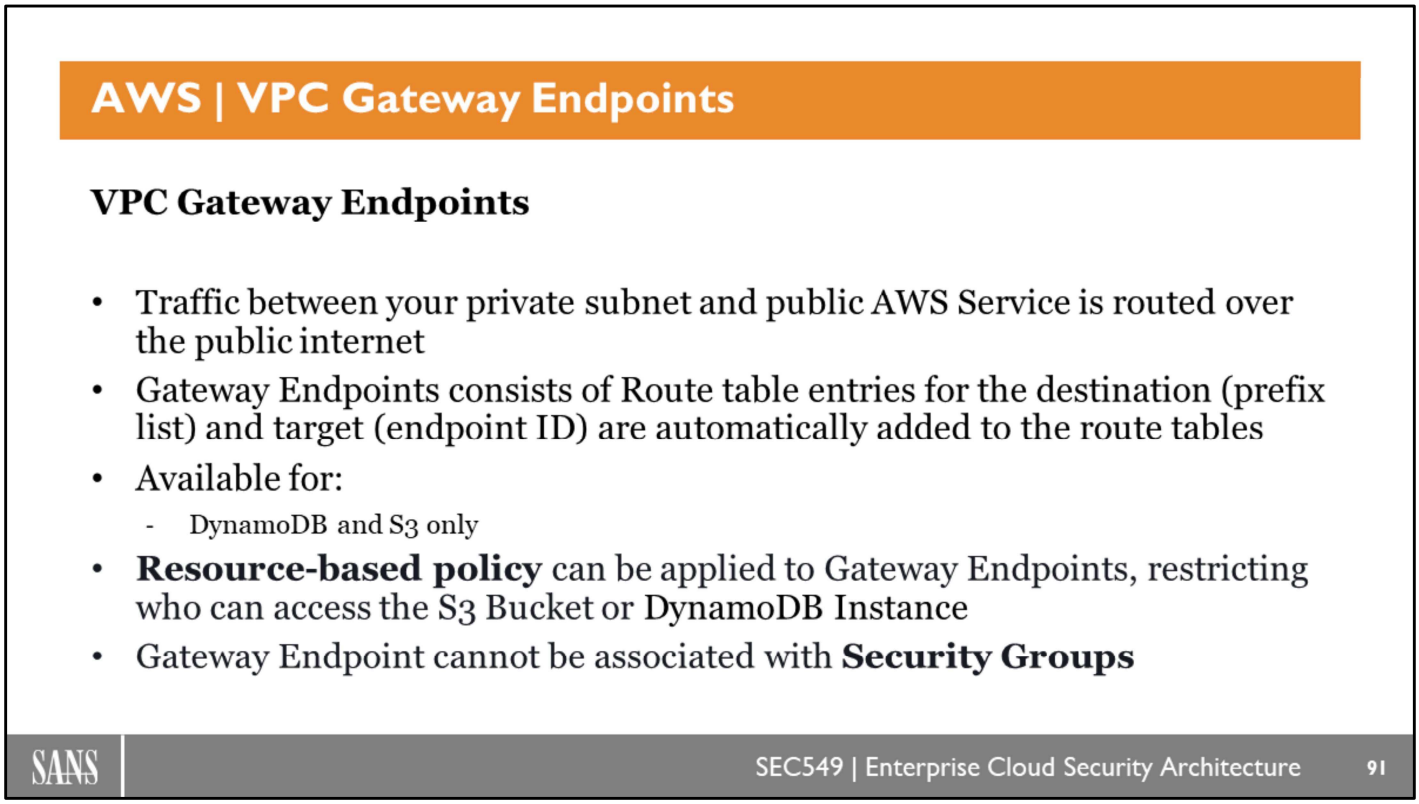
Connecting VPC-Bound and Public Resources in AWS with Interface Endpoints
The Interface Endpoint[1] is the most popular and versatile type of VPC Endpoints. When you create an Interface Endpoint, an Elastic Network Interface (ENI) is created in the subnet you specify during creation. This ENI is assigned a private IP from your subnet's pool. Interface Endpoints are commonly thought of as a way to privately consume public AWS services; however, the VPC Interface endpoint can be used to privately consume a customer-deployed service or a vendor-supplied service, say from the AWS Marketplace.

Interface Endpoint:
- Is associated with an Elastic Network Interface (ENI)
- Resides in a specific subnet in a VPC assigned during creation
- Can be attached to:
    - A single AWS Service
    - A customer created endpoint service, or
    - An AWS Marketplace Service
- VPC-bound resources interact with Interface Endpoints by calling the Endpoint URL
- Private, cross-account access can be enabled with Interface Endpoints

Network-Based Controls:
Security Groups can be attached to Interface Endpoints to define both ingress and egress traffic to and from the interface endpoint. When you create a VPC Endpoint, if no security group is specified, the default security group is associated with the interface endpoint. The default security group does not restrict ingress or egress traffic to the endpoint. As a result, any resource that is also in the default security group has unfettered network connectivity to the endpoint and the underlying service.

Identity-Based Controls:
In upcoming slides, we will cover in-depth the resource-based policies which can be applied to Interface Endpoints, regulating access to the underlying service.

Reference:
[1]: https://sec549.com/id148



AWS VPC Gateway Endpoints
VPC Gateway Endpoints solve the same problem Interface Endpoints do, in that they both provide a mechanism for private resources to have access to public services. Gateway Endpoints, however, go about solving this problem differently. A Gateway Endpoint is not assigned an ENI within your VPC; rather, it is a route in a private subnet used to access either a public S3 Bucket or DynamoDB. Rather than routing outbound traffic through a NAT gateway or providing network connectivity via an internet gateway, you can use a Gateway Endpoint to allow your private EC2 instances to access public S3 or DynamoDB. Because your egress traffic is not transiting through an intermediary appliance, Gateway Endpoints are provided to AWS customers at no cost.

Network-Based Controls:
Security Groups cannot be applied to Gateway Endpoints in the same manner as with Interface Endpoints. However, EC2 instances must allow outbound traffic to the S3 or DynamoDB service for them to be allowed to call those services via a Gateway Endpoint. The only network consideration with Gateway Endpoints is Routing. With Gateway Endpoints, route tables must be configured directing traffic destined for either S3 or DynamoDB to the Gateway Endpoint.

Identity-Based Controls:
In upcoming slides, we will cover in-depth the resource-based policies which can be applied to Gateway Endpoints, regulating access to the underlying service.

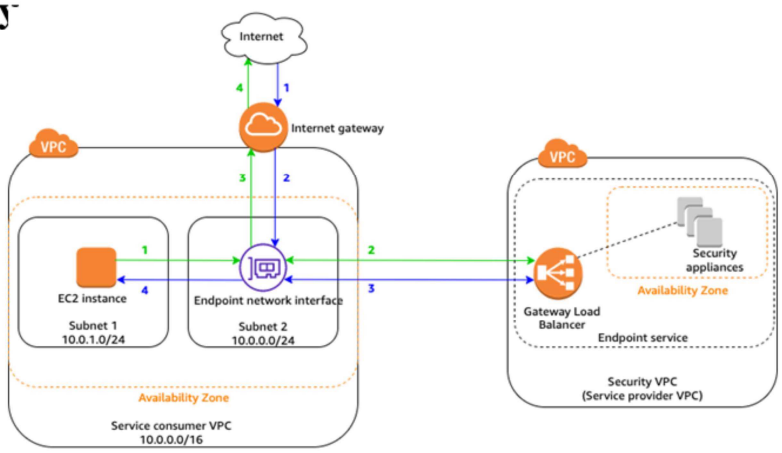
Reference:
[1]: https://sec549.com/id149

AWS | VPC Gateway Load Balancer Endpoints

VPC Endpoints For Gateway Load Balancers

Patterns of Use:

- Traffic Inspection Points
- Centralizing network traffic shaping



SANS | SEC549 | Enterprise Cloud Security Architecture 92

AWS VPC Gateway Load Balancer Endpoints

These types of endpoints are almost exclusively used to create traffic inspection points. By creating an Endpoint Service from your Gateway Load Balancer and placing it in a Security Tooling account, you can effectively create a central processing location for any North/South traffic flow in AWS.

With VPC Gateway Load Balancer Endpoints, VPC policies cannot be used to control access at the identity layer, and Security Groups cannot be used to restrict the network-layer.

References:
[1]: https://sec549.com/id150
[2]: https://sec549.com/id151
Image Source: Access virtual appliances through AWS PrivateLink - Amazon Virtual Private Cloud
https://sec549.com/id152

AWS | Access Control for VPC Endpoints

VPC Policies are:

- Resource-based policies, attached to only to Interface or Gateway Endpoints

VPC Policies define:

- Which IAM actions are allowed or denied when calling services through the Interface of Gateway Endpoint
- Which specific resources can be called in any given service
  - In the example of an Interface Endpoint exposing API Gateway, policy can define which specific API Gateway can be accessed

VPC Policy Limitations:

- As with all resource-based policies, only one policy can be attached to a resource at a time in AWS
- Policies are restricted to 20,480 characters
  - These limitations can be a factor when considering a centralized or decentralized approach to VPC Endpoints

Access Control for Endpoint Interfaces:
Identity-Based Controls: Who can call up the service via Interface or Gateway Endpoints is defined using Endpoint Policies. These are resource-based policies attached directly to each Interface Endpoint. If during endpoint creation, you do not create an Endpoint Policy, a default policy is created and attached for you. The default policy can be problematic if not modified as it doesn't restrict the resource.

Fields within a resource-based policy include:

- "Effect": Defining whether an action is allowed or denied
- "Principal": Defining who can call the underlying service, often this will be defined as an EC2 instance profile, a type of Role for an EC2 Instance.
- "Action": Defining which actions are allowed on the underlying resource
- "Resource": The most important field perhaps of the VPC Endpoint Policy, constraining which exact resources can be called.

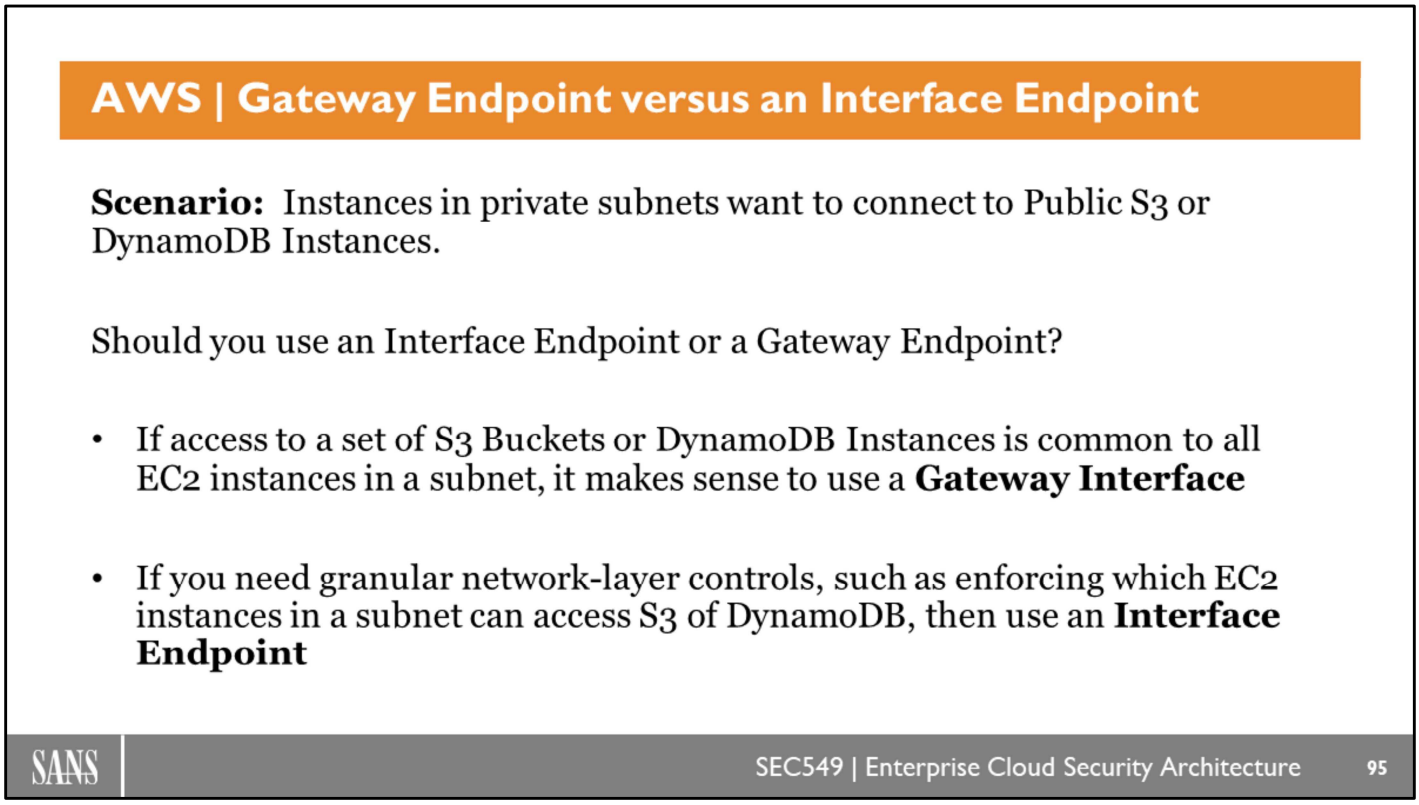
AWS | Default VPC Endpoint Policy

Default VPC Policy

- Problematic if the resource field of the default policy is not updated
- Effectively allows any resource with network connectivity to the VPC Endpoint to call **ANY** actions on the service behind the endpoint
- Can be used to exfiltrate data through an attacker controlled VPC Endpoint

```
{
    "Statement":
        [
            {
                "Action": "*",
                "Effect": "Allow",
                "Resource": "*",
                "Principal": "*"
            }
        ]
    }
```

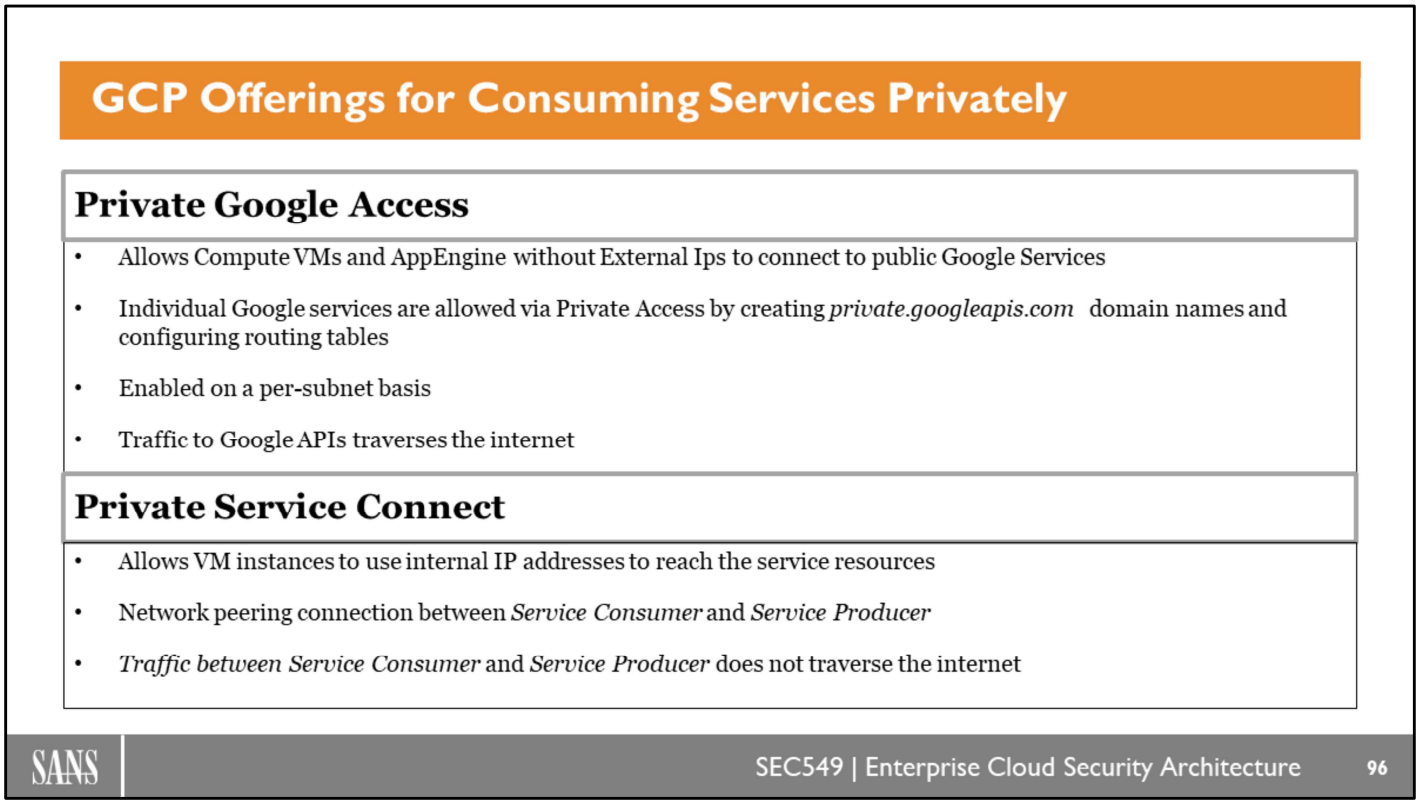

How Could the Default VPC Policy be Abused by an Attacker?
Imagine that a private subnet has VPC Endpoints configured allowing private EC2 instances to call S3. The intention is that these private EC2 instances will only need to get objects from a handful of 'in-house' AWS S3 buckets. Unfortunately, without a restrictive VPC Policy, if the private EC2 instances are compromised, a malicious actor could use the VPC Endpoints configured for S3 access to exfiltrate data and copy objects to an attacker-controlled bucket.



Gateway Endpoints versus Interface Endpoints: When to Use Which?
Using Gateway Endpoints versus Interface Endpoints
Let's compare the security configuration available for both Gateway Endpoints and Interface Endpoints.

Network-Layer Controls:
- Network controls in the form of security groups can be applied to Interface Endpoints only, controlling which EC2 instances are allowed to initiate inbound traffic.
- Security Groups cannot be applied to Gateway Endpoints. Route Table Rules are the only way to control network traffic to the Gateway Endpoint, which would enable traffic in an entire subnet to reach the gateway.

Identity-Layer Controls:
- VPC Policy can be applied to both varieties of endpoints. By leveraging the 4 policy fields of the resource-based policy, you can control WHO has access to the underlying service behind the endpoint and exactly which resource they can access.



GCP Offerings for Consuming Services Privately

Private Google Access[1]:

A compute instance in GCP is considered private if it lacks an external IP address. As such, it can only connect to other internal IP address destinations. To enable external connectivity, you can allow compute instances to connect to the set of external IP addresses used by Google APIs by enabling Private Google Access. This feature enables private access to entire services.

Private Access is most like AWS Gateway Endpoints, as this feature enables VPC-bound resources to connect to public resources. When enabled in a subnet, all compute instances can connect to enabled Google APIs and Services through custom DNS names. Network traffic from compute instances to Google APIs traverses the open internet. Unlike AWS VPC endpoints, policy cannot be layered on top of Private Access to restrict API access at the identity-layer.

GCP Private Service Connect[2]:

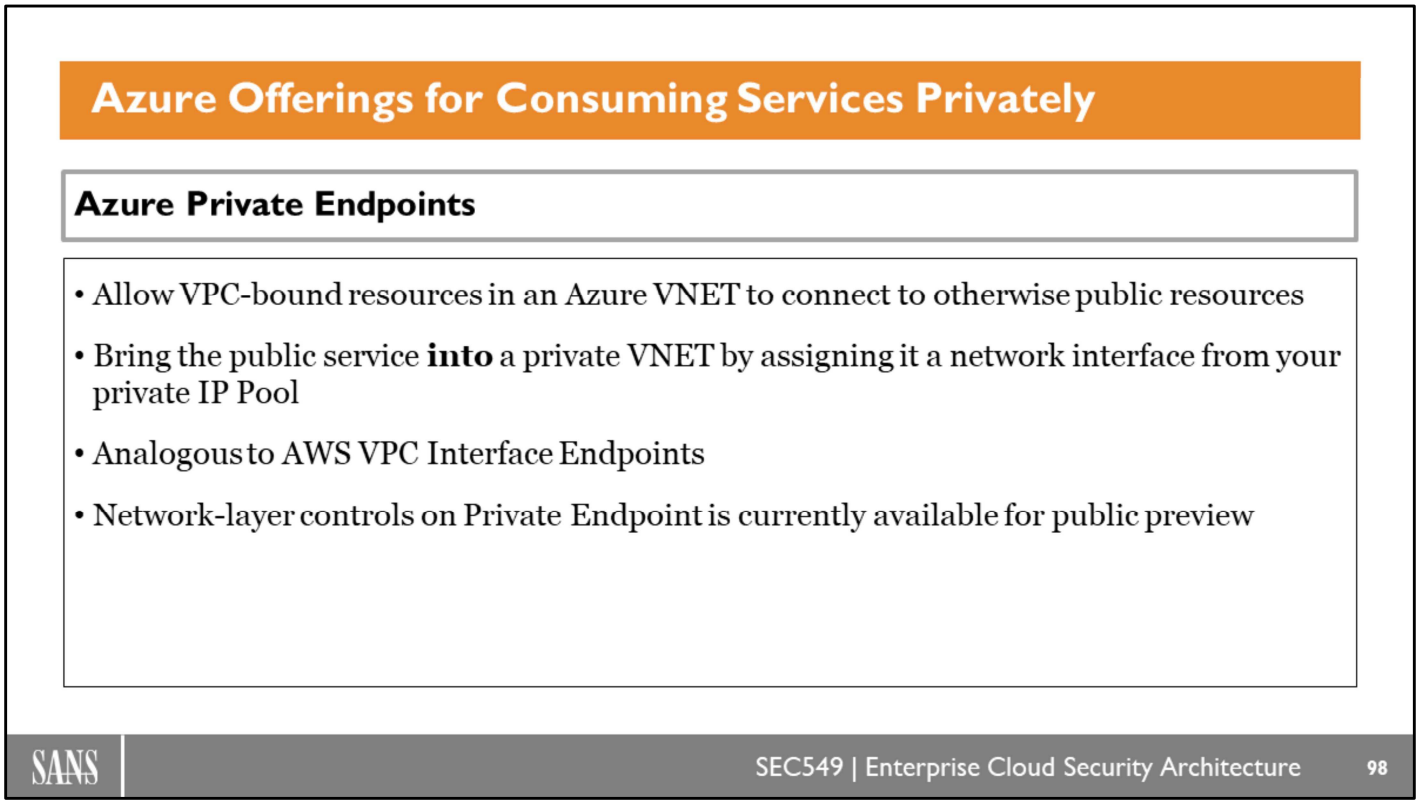
The Private Service Connect offering from Google is analogous to VPC Interface Endpoints. When configured, access to the associated service (*Service Producer*), is routed over the Google network, not over the public internet. Interfacing with a *Service Producer* is done through an endpoint. Endpoints have an internal IP address in your VPC network and are based on the forwarding rule resource.

Private Service Connect consists of an allocated set of private IP spaces in your VPC. Requests that are sent to that internal IP address block are transparently and automatically routed to a public Google-managed service over the Google backbone network. Configuring this service requires new endpoints to be configured in the reserved private IP block set aside for Private Service access.

All Service Producers can offer their services with internal IP addresses to google cloud customers. Private services access enables you to reach those internal IP addresses from inside your VPC. This is useful if you want your VM instances in your VPC network to use internal IP addresses instead of external IP addresses. The private connection links your VPC network with the service producer's VPC network.

Anyone can act as a Service Producer, publishing endpoints to be consumed privately but most likely you will interact with Private Services when Google themselves is the publisher.

References:
[1]: https://sec549.com/id153
[2]: https://sec549.com/id154



Azure Offerings for Consuming Services Privately

Azure offers its customers Private Endpoints[1] for consuming public services privately. Below are the service highlights:

- Private endpoints are powered by Azure PrivateLink.
- Communication can only occur in one way: from the clients to the service provider.
- As in AWS' VPC Endpoints and GCP's Private Service Connect, private endpoints in Azure allow for private access to an Azure service, but do not necessarily restrict public network access.
- Network-layer controls can be applied in the form of Network Service Groups (NSG); however, this capability is in public preview[2].

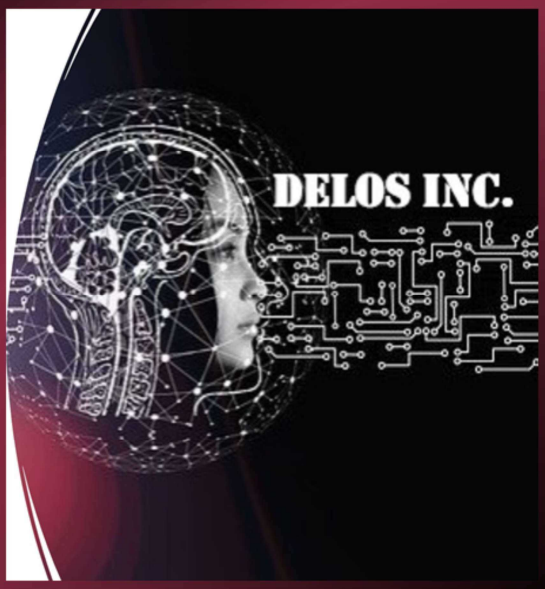
References:
[1]: https://sec549.com/id155
[2]: https://sec549.com/id156

Cloud Journey Phase: Scalable Network Design

Scenario:

Delos is planning for its first phase of publicly facing web applications and services to be rolled out onto AWS. The Network Security Team is looking for a pattern that will accomplish a couple goals. First, they want to ability to control from which subnets ingress is allowed, Second, they want to configure inspection points when traffic flows between trust zones. Finally, the Network Security Team does not have the bandwidth to manage the security groups for all application teams. Considering the herculean effort, it would be to manage all security groups in the Organization, these finer grain controls have been ceased to individual teams.

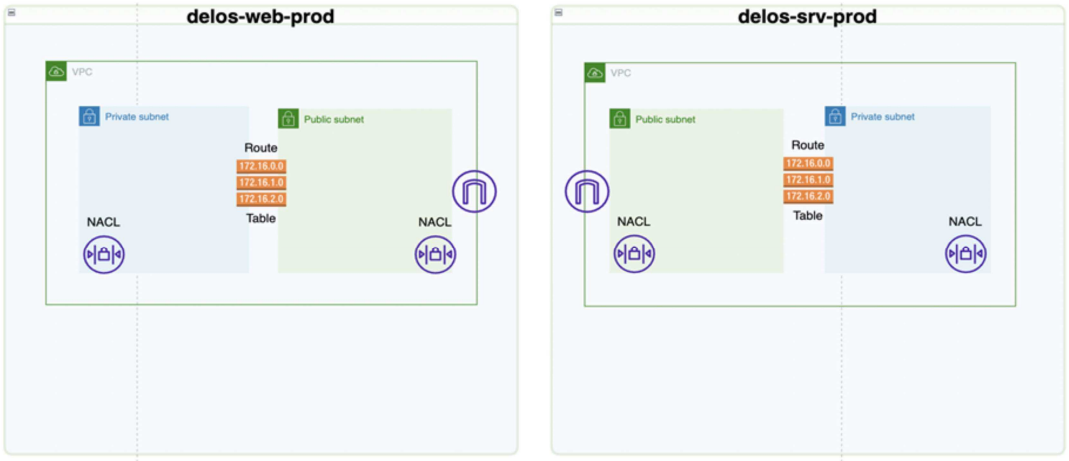
With these goals in mind, you are tasked with configuring a shared VPC architecture with the Networking Team hosting the VPCs used in the Organization.



Reference:
Image by User geralt from Pixaby: https://sec549.com/id51

Lab 2.2 – Current State Architecture

Each application team is hosting is own VPC resource. VPC-level controls are in the hands of individual teams.



An alternative to a Shared-VPC Architecture is a Multi-VPC Architecture. In this diagram, we're depicting the 'delos-web-prod' and the 'delos-srv-prod' accounts hosting their own VPCs for their EC2 and RDS Instances.

While there is nothing incorrect about this pattern it does come with downsides, including that each team that hosts the VPC has full control over the VPC-level controls such as the Internet Gateways, IP Space, NAT Gateways, VPC Peering and Network Access-Control Lists.

Lab 2.2 – Creating A Shared VPC Architecture

Estimated Time: 45 minutes

In this Lab, you are presented with the start of a shared VPC. The Networking Team is hosting a single VPC called '*prod-shared-vpc*' hosted from their *network-web-prod* account. Subnets have been shared to the *delos-web-prod* and *delos-srv-prod* AWS Accounts.

Your task is to depict the provisioning of subnets for the delos-web-prod Account and indicate between which subnets network traffic should be explicitly blocked via NACL so that traffic between the *delos-web-prod* and *delos-srv-prod* Accounts and all egress traffic is forced through a SSL inspection appliance before being forwarded on.

Preparation

- View **Lab 2.2** in the course workbook
- Open the workbook in an incognito window
- https://workbook.sec549.com
  - Username: H03-student
  - Password: million-sailor-DEAR

Objectives

In the provided template sequence diagram:

1. Depict the provisioning of shared subnets to the 'delos-web-prod' Account mimicking the deployment pattern in the 'delos-web-srv' Account.
2. Document the use of NACLs to restrict communications between the subnets in the 'delos-web-prod' Account and subnets shared to other accounts

This page intentionally left blank.

Lab 2.2 – Summary

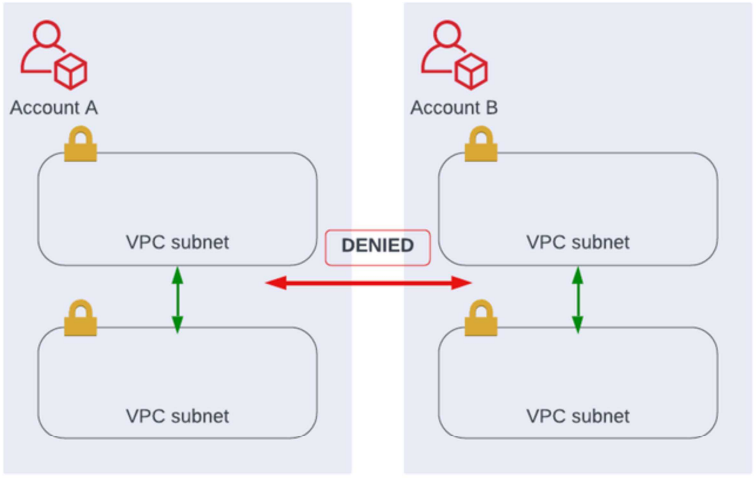
In this lab, you...

- Observed the shared public, private, and transit subnets shared from a single VPC residing in the *network-web-prod* AWS Account
- Documented how the NACLs between subnets in the same account allow all network traffic
- Documented how all network traffic between the *delos-web-prod* and the *delos-srv-prod* subnets is denied via NACLs
- Logged into the AWS Console to answer the ⚡ See it in Action ⚡ questions

This page intentionally left blank.

Lab 2.2 – End State Architecture

- A single VPC is hosted in the infrastructure account
- Subnets are shared to participant accounts, *delos-web-prod* and *delos-srv-prod*
- NACLs restrict network flow between web and services account
- Routing will be in place allowing web and srv subnets to communicate but only as routed through the inspection subnet



Several goals have been accomplished in this pattern. First and foremost, a single VPC is hosted in the infrastructure account. This allows the networking team to take ownership of the IP space, using Network ACLs defining in broad-stokes the network traffic allowed inbound and outbound between VPC, Routes, Transit Gateway, and Internet Gateways.

The Web and Services Accounts are both provisioned with three subnets, a **Public Subnet** for their publicly facing workloads, **Private Subnets** and a **Private Transit Subnet**.

- Traffic has been allowed intra-account between subnets and restricted directly between the delos-web-prod and the delos-srv-prod account.
- The only route traffic is allowed to take between the delos-web-prod and the delos-srv-prod accounts is through their **Private Transit Subnet** via the infrastructure account.
- This becomes an introspection point which can be leveraged by the Security Team to inspect traffic, either with Cloud-Native Solutions or a 3rd Party Offering from AWS Marketplace.
- NACLs are used to enforce isolation between the subnets and are controlled by the Infrastructure Team who hosts the Shared VPC. More fine-grained access-controls via Security Groups are under the control of each individual team.

Course Roadmap

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

SECTION 2

- Introduction to Cloud Migrations
  - Drivers for Cloud Migrations
- Implementing Zero-Trust Architecture
  - Using Cloud Services to get to ZT
    - Lab 2.1: Integrating Auth into Legacy Application
- Establishing Perimeters for Application Access
  - Connecting VPC-Aware and Non-VPC Aware Services
    - Lab 2.2: Creating a Shared VPC Network
- ***Establishing Perimeters for Data Access***
  - Managing S3 Access At Scale
    - Lab 2.3: Access Control For Shared Data Sets

This page intentionally left blank.

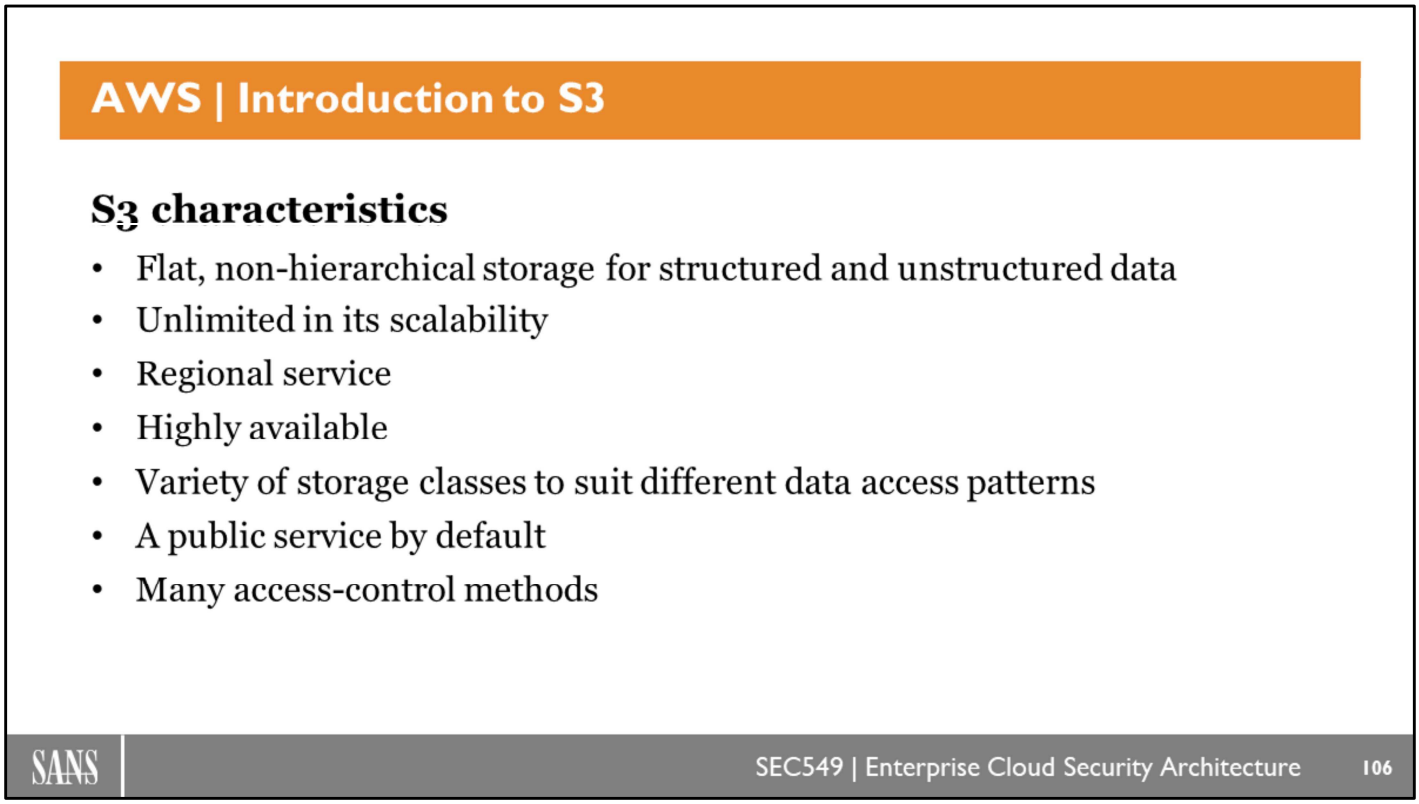
Takeaways

In this module, you will be taken through a series of use cases for AWS S3, from replica repository to data lake design. Features and configurations of S3 are framed around the variety of ways S3 is used in to meet business goals.

You will come away with a strong understanding of which features to enable for the scenario S3 is being used for. Access-controls at the bucket level will be covered in depth so you see how to granularly control access to data and prevent public bucket misconfigurations.

Finally, the configurations and features of CloudFront will be discussed in depth in conjunction with S3 as a web hosting platform.

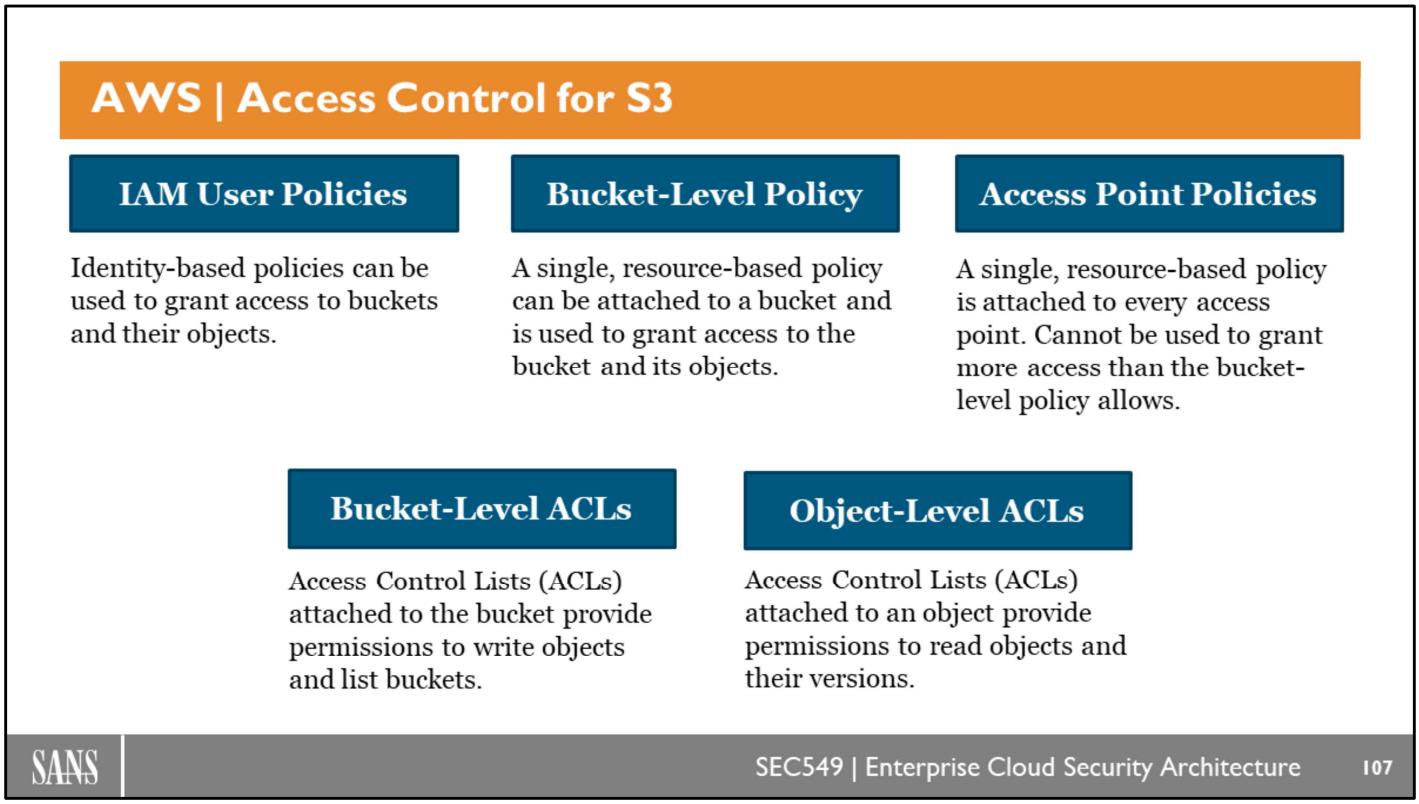
This page intentionally left blank.



Introduction to S3

A single AWS account can have hundreds of S3 buckets, and each bucket can contain many terabytes or petabytes of files. S3's ubiquity in AWS environments is due to its flexibility. S3 can be used to store application data that doesn't fit neatly into a database. This might include images, static web pages, web assets like CSS files, and user-generated data. Within serverless architectures, you could find S3 used as a source repository for a code base of serverless functions.

Over this next module, we will cover several use cases for S3, the solutions in which you might find S3 deployed, and which controls are most critical to enable in any given pattern.



Access Control for S3

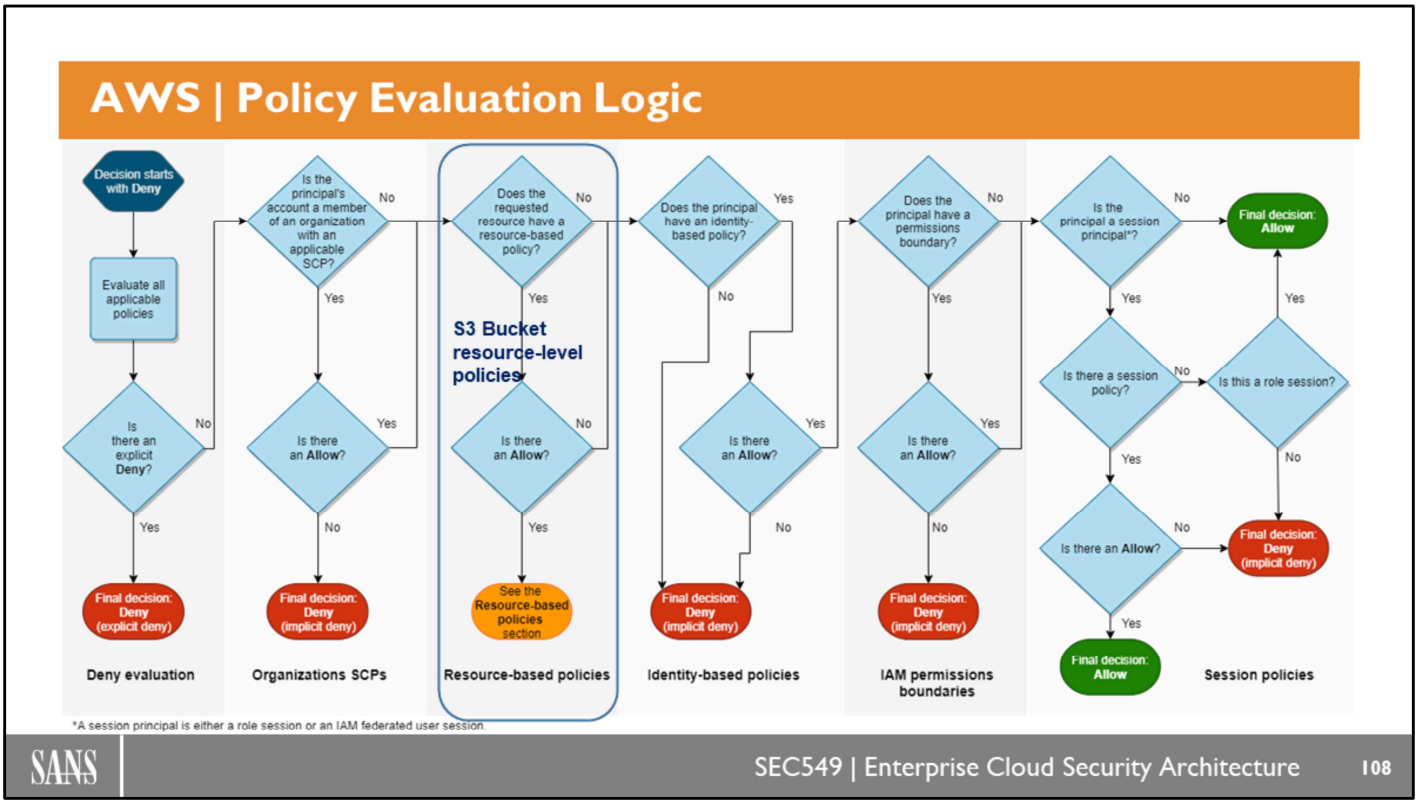
Access Control in S3 is complicated. The complexity of S3 is reflected in the page count (163 pages) of the threat model published by the security consultancy 'Trust On Cloud"[1].

In the next few slides, we'll cover the core mechanisms for granting access to objects and buckets:

- Identity-based policies and how they intersect with resource-based policies
- Resource-based policies attached to buckets
- Access Control Lists (ACLs) and how they affect buckets and objects access

Reference:

[1]: https://sec549.com/id157



S3 Policy Evaluation Logic[1] in AWS

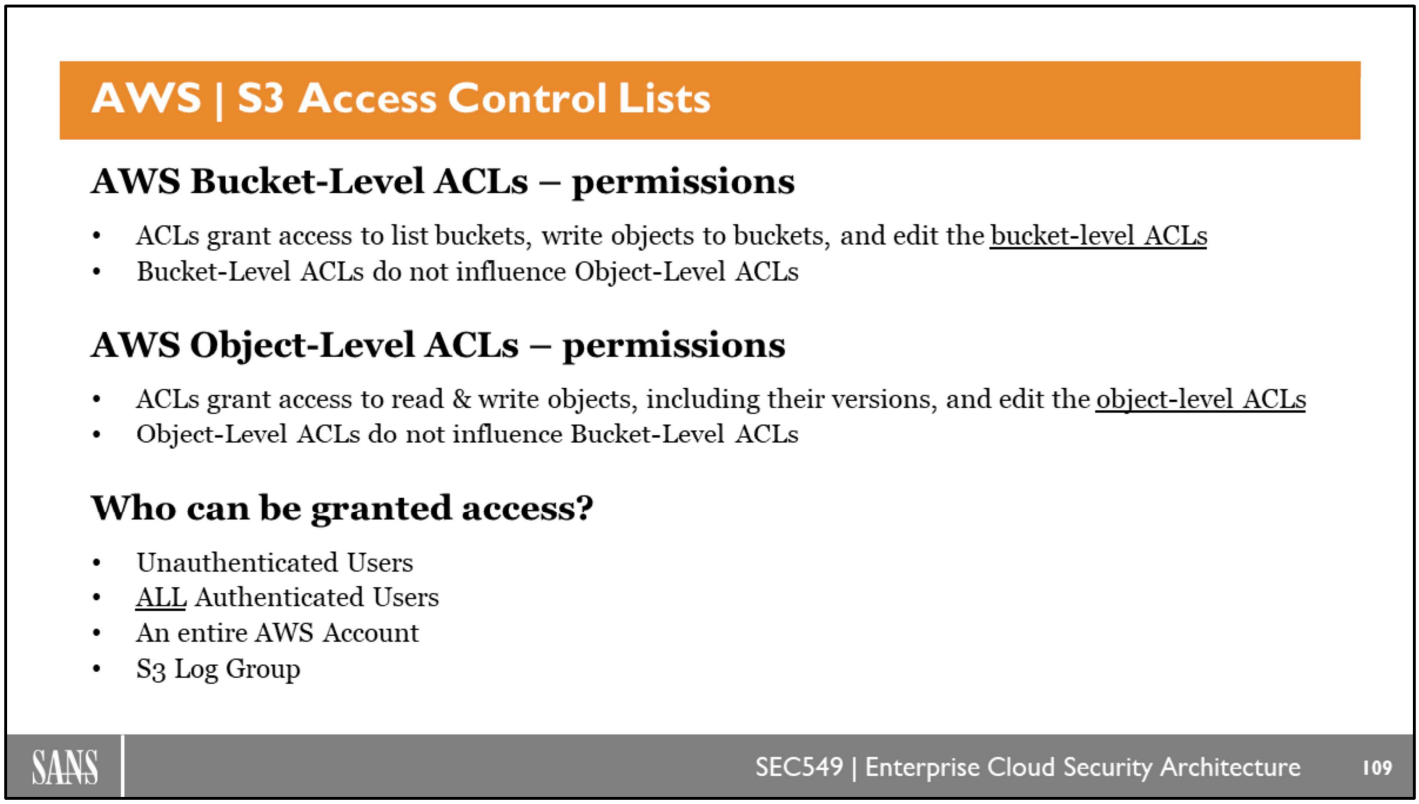
We've already seen how permissions are assigned to IAM Users, Groups, and Roles. As a refresher, a policy defines a set of permissions dictating whether a Principal is allowed or denied actions in AWS.

Policies can be attached to resources such as S3 Buckets rather than a User. With S3 resource-based policies, Principals such as IAM users, roles, or accounts can be granted access to entire buckets or to certain objects within a bucket. Any object or bucket can also have their access conditionally allowed (or denied). Using an S3 bucket policy, even external IAM Users, Roles, or Accounts can be granted access to objects.

But, what if one policy type allows a Principal to have the 'GetObject S3 action' while another policy does not grant the same permission? To answer this question, we need to consult the AWS Policy Evaluation Chart. It is particularly important to be aware of the flow of policy evaluation when working with S3 buckets or any other service which supports resource-based policies.

As you can see, when moving beyond the initial Deny evaluation and SCP evaluation, a resource policy can grant access to a Principal, even if that Principal does not have associated identity-based IAM policies granting them that access. However, allowing access via a resource-based policy or an identity-based policy will never trump a deny statement, no matter where it occurs.

Reference:
[1]: https://sec549.com/id158
Image Source: Policy evaluation logic - AWS Identity and Access Management https://sec549.com/id159

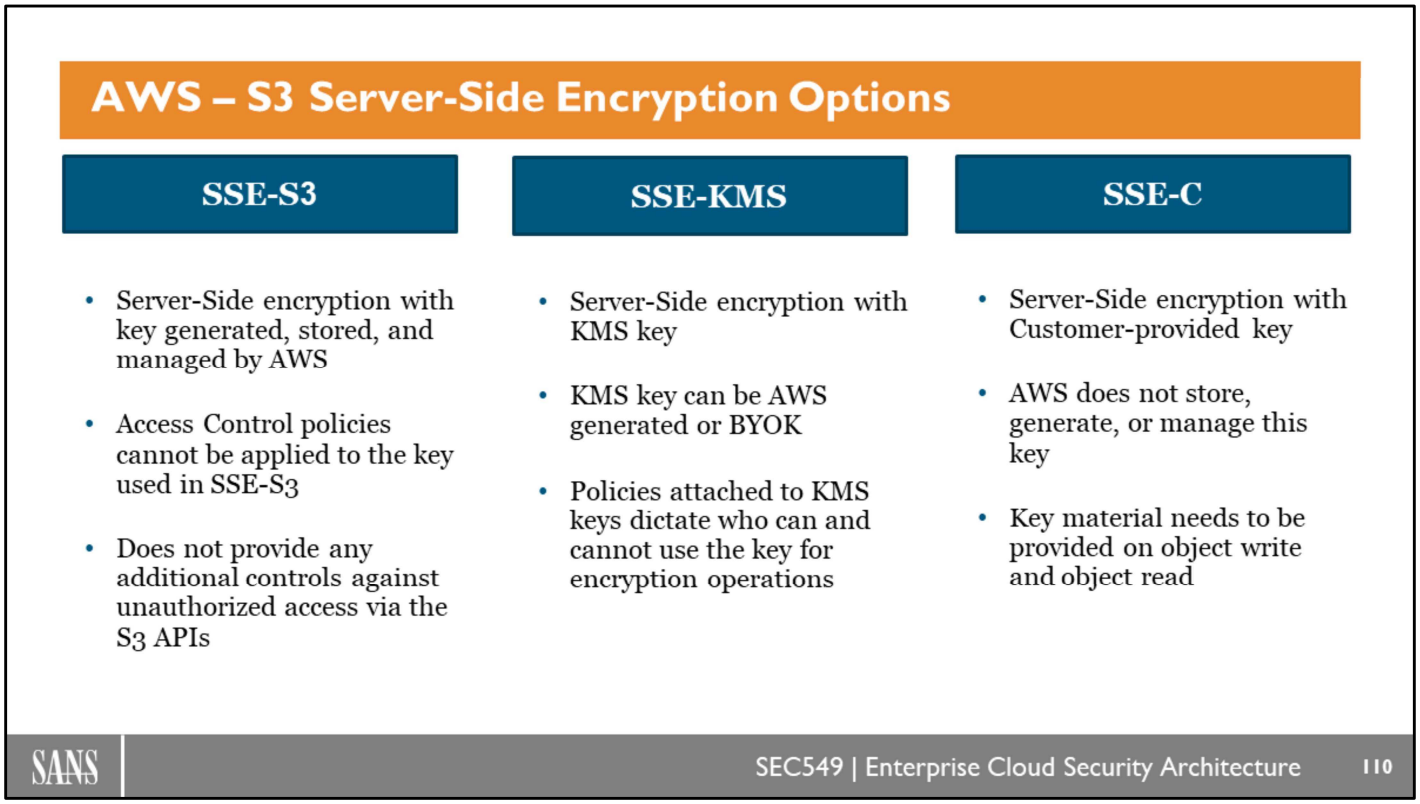


S3 Access Control Lists

S3 Access Control Lists (ACLs) are an alternative, legacy mechanism for granting access to S3 Buckets and Objects. ACLs exist separate from IAM Policies (identity-based or resources-based) for granting access. There is unlikely to be a use case for granting access via an ACL. If ACLs are used to grant access, it is generally considered an anti-pattern.

Because ACLs are a competing access model and their abuse potential is high, it is strongly recommended to disable ACLs. This configuration option was just made available to AWS customers in 2021.[1] We will cover the disabling of ACLs and how to apply the control in later slides.

Reference:
[1]: https://sec549.com/id160



S3 Server-Side Encryption Options
Three different flavors of server-side encryption exist for objects stored on S3, each with their own security benefits.

SSE-S3:
This mechanism for encryption uses an AWS-managed key to encrypt objects with a symmetric key. The key is generated, rotated, and stored by AWS and cannot be accessed by the customer. No access controls can be applied to the key used in encryption/decryption; therefore, encrypting with SSE-S3 has limited use for preventing unauthorized access to data. With SSE-S3 option enabled on your Buckets, all objects will be encrypted at rest as the data resides in an Amazon data center. Should an AWS Principal, benevolent or malicious, have the S3 permissions to access data, the SSE-S3 encryption option does not hinder their access. SSE-S3 encryption on S3 is often used to fulfill compliance requirements rather than meet security goals.

SSE-KMS:
When objects on S3 are encrypted with SSE-KMS, the key used is a KMS symmetric key. KMS stores key material for customers. Keys are either generated by AWS or (optionally) they can be used to store key material in 'Bring your own Keys' (BYOK) scenarios. These keys are regional and have attached resource-based key policies that can be leveraged to allow or deny access to encryption operations on the keys.

Leveraging the key policy for an additional layer of access control can be powerful. In scenarios when a Principal, benevolent or malicious, has the S3 permissions to access data, they additionally would need permissions to perform operations with the KMS key used to encrypt the data server-side.

SSE-C
Encrypting objects with SSE-C option requires the customer to store the symmetric key. In this scenario, AWS does not store the key. Instead, the key material is provided explicitly during *object write*. And necessarily, they are required to be provided during *object read*.

The next group of slides will walk you through the vast and various use cases for S3. Within each use case, we'll talk about the patterns and configuration settings that are most prevalent, given the use case.

AWS S3 Use Cases – Serving Static Web Content

S3 'Website Endpoints' feature:

- Serve content: html / JS Files / Images
- Subdomain or root[1] hosting
- Use S3 to redirect traffic to another domain

Why NOT to use S3 Website Endpoints?

- Does not support HTTPs
- Regional
- Bucket is publicly available to all anonymous users

Serving Static Web Content in AWS

A great use case for S3 is serving static content to a website. Whether that content is images or web pages, S3 does a great job of delivering these static files to web and mobile clients. In applications commonly referred to as 'Single-Page Apps' (SPAs), often the dynamic processing will be left to the front end or powered by serverless backend logic.

The built-in functionality to use S3 to host static website[2] content was one of the first advanced features added to S3 back in 2011. Once static web hosting is enabled, S3 will generate a URL from the name of your bucket, which can be used to access the web content.

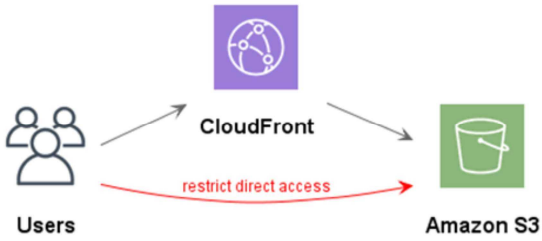
This simple pattern may suffice for hobby projects but has many drawbacks and deal-breakers, which should prevent it from being used in the enterprise setting. Among other disadvantages, buckets configured with the static web hosting feature MUST allow all anonymous users to have the ability to read the objects in the bucket. This permission is configured in the bucket's resource-based policy.

Why be concerned if objects that are intended to be public can be directly access from the bucket? The required resource-level policy configuration necessitates a lax 'block public access' setting on the bucket, making it difficult to uniformly prevent public S3 access across entire AWS Accounts. And as we will see in later slides, there are more secure configurations you can use for S3 when using it to server static web content.

References:
[1]: https://sec549.com/id161
[2]: https://sec549.com/id162

AWS | Serving S3 Content Behind CloudFront

- Configure CloudFront to use S3 as an Origin Server
- Prevent anonymous access to your S3 Bucket directly
- Maintain 'private' status of buckets serving web content
- Use https to serve files



Serving S3 Content Behind CloudFront
Instead of using the static website feature of S3, you can serve AWS S3 content in conjunction with an AWS CloudFront Distribution[1]. CloudFront will operate as a Content Delivery Network (CDN) to distribute the files on AWS Edge locations network. This will make sure your content is delivered with minimum latency anywhere in the world.

What is CloudFront?
CloudFront is Amazon's version of a managed Content Delivery Network (CDN). Using a CDN both speeds up the distribution of content to visitors and reduces the overall cost for a busy site by caching copies of commonly retrieved files.

You can use CloudFront as a CDN and configure S3 as the Origin Server to direct CloudFront to serve files from your bucket. The resource-based policy attached to the S3 bucket only allows object access from the Origin Access Identity associated with the CloudFront Distribution. This pattern allows for your content to be served over https and access restricted to only your CloudFront Distribution so the S3 bucket serving the content can remain private.

Reference:
[1]: https://sec549.com/id163
Image Source: https://davelms.medium.com/serving-static-content-from-s3-using-cloudfront-and-origin-access-identity-c8ca667b3d71 https://sec549.com/id164

AWS | S3 Object Access via CloudFront OAI

What is an Origin Access Identity (OAI)?

- A special kind of identity you manually create and associate with CloudFront Distributions
- An OAI can be assigned to a CloudFront distribution and used to identify requests to an S3 origin
- The S3 origin bucket can then use the OAI in a bucket policy to allow only requests from a CloudFront distribution with that specific OAI

Bucket Policy Example:

```
1  "Principal":
2      {
3          "AWS":      "arn:aws:iam::cloudfront:user
               /CloudFront-Origin-Access-Identity"
4      }
```

What Is Origin Access Identity (OAI)?
An OAI is a User in AWS that cannot be assigned any other roles, policies, or permissions other than for use in S3 bucket policies. The only reason an OAI exists is to allow better security of S3 origins for CloudFront distributions. IAM users cannot be assigned to CloudFront distributions; only OAI Users can.

By allowing an OAI to access S3 objects rather than anonymous users, you do not need to disable **block public access settings** for your buckets serving web content. We will talk more on the **block public access** and how to apply this guardrail setting in later slides.

AWS S3 Use Cases – Data Lake

Why maintain a central cloud-hosted repository?

- Solves disjointedness and prevents data siloes
- Scales with exploding data and user growth
- Flexible access model that grows as your data consumers grow

S3 as a repository for raw data points

- Handles diverse sets of data, including structured, unstructured, images
- Integrations with other cloud services via IAM

S3 Use Cases as a Data Lake

A successful business-driven data practice allows an organization to harvest raw facts (your data), transforms that data into information (Extract, Transform, Load capabilities), and gains insights that can fuel data-driven decisions in the business.

Why is S3 being used as a foundation for data lakes in AWS?

- S3 scales infinitely with judicious use of S3 prefixes
- Consistent and predictable behavior for storing your data points
- Flexible model for providing access controls to S3 data sets
- Access Control at the Bucket, Object, or Access Point level.

The flexible access model allows you to grow around your data foundation (S3) without having to fundamentally redesign the architecture of your data lake. In the next few slides, we'll focus on how to best organize S3 objects for scale, selectively grant access to classes of objects, and allow resources to consume data privately.

AWS | Organizing S3 Objects with Prefixes

The S3 Prefix

- A mechanism for organizing objects in S3 that resembles a file directory. It is the complete path before an object key

Prefixes for S3 scaling

- Data access limits are scoped to Prefixes.
  - 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per prefix in a bucket
- Create unlimited prefixes in your bucket to scale Read/Write operates

Access-Control for Prefixes

- Policies cannot be directly attached to prefixes
- Instead, access to prefixes is dictated from bucket-level policy, access point policy, or identity-based policies

An S3 Prefix

A prefix is a namespace you can use to subdivide a S3 bucket, helping organize the objects contained within. Prefixes are leveraged to organize data but also ensure S3 access scales vertically as data grows within a bucket. This is because request limits to S3 buckets are scoped to the Prefix rather than the bucket. Since the number of Prefixes in a bucket is infinite, the S3 service scales infinitely. Clearly, there is no question that buckets should be subdivided, but how do Prefixes work as an access control boundary?

Restricting access on the S3 Prefix Level

As the data (and Prefixes) grow in your bucket, the need will arise to grant access based on a Prefix. The urge with Prefixes is to use them as access control boundaries in addition to being a mechanism to evade S3 request limits; however, no Prefix-level policies exist.

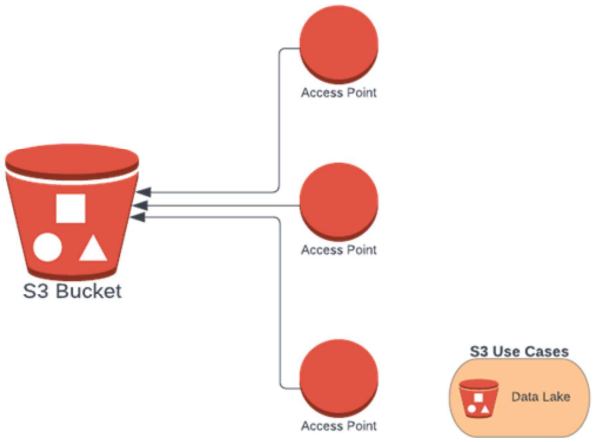
One method of accomplishing per-prefix access control is by using bucket-level policy. In order to granularly control access to every prefix, policy statements are added for every prefix to the single bucket policy. While the number of prefixes is unlimited, the size of bucket policy is not. With potentially hundreds of prefixes in a properly segmented bucket, bucket-level policy would not be large enough to accommodate the numerous access statements needed. Scoping access to S3 prefixes in a bucket policy can result in a policy document that is bloated and unmanageable.

Instead of using bucket policy to accomplish per-prefix access control, IAM identity-based policies can be used to restrict access on the per prefix level. As a third alternative, Access Points can be created for your buckets as a mechanism to overcome the policy size restrictions.

References:
[1]: https://sec549.com/id165
[2]: https://sec549.com/id166

AWS | S3 Access Points

- Access Points can be associated with exactly 1 bucket
- Access Points are by default regional but can be configured to be multi-regional[1] providing applications and end-users a global endpoint for accessing S3 data
- Access Points are **policy attachment points** allowing for granular control of S3 objects in a single policy document

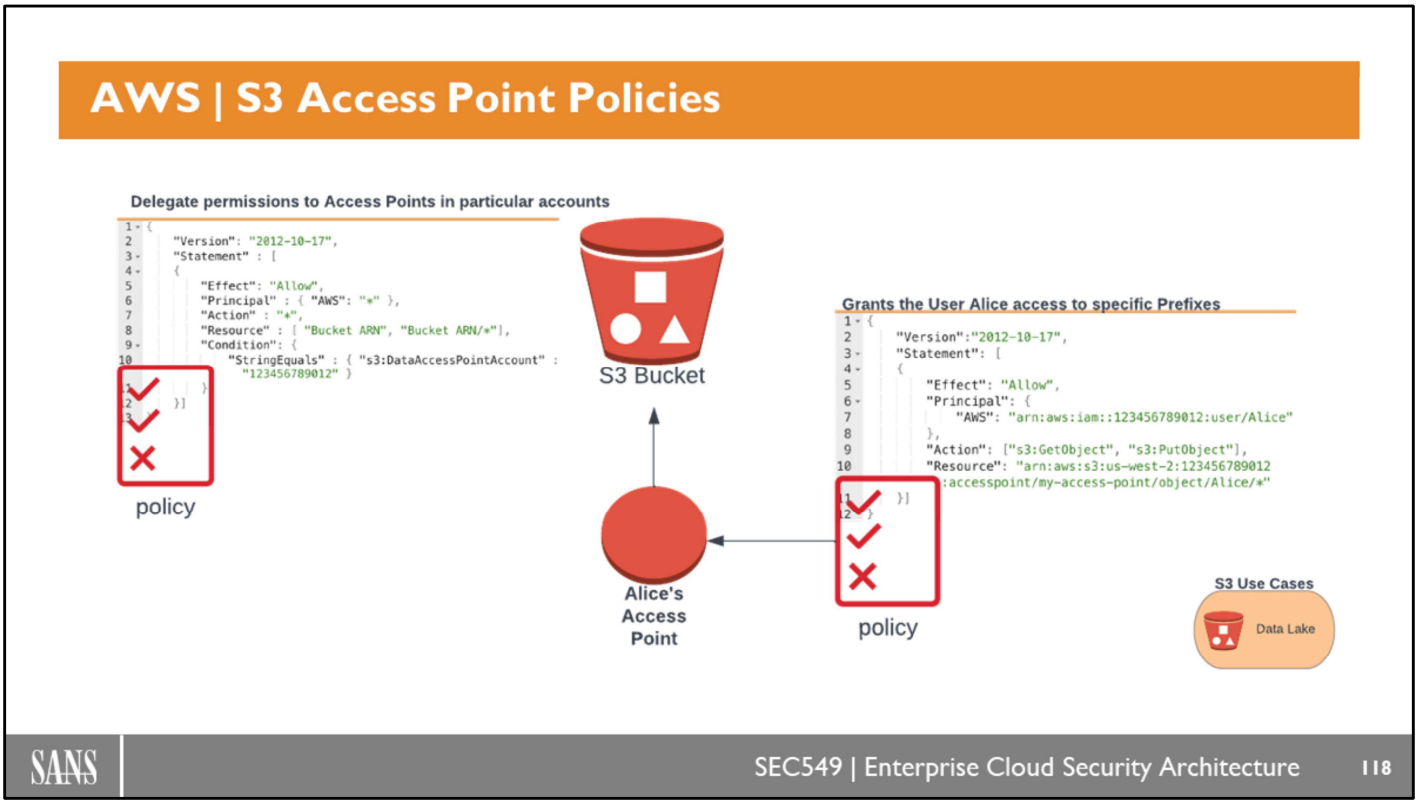


S3 Access Points
An S3 bucket will often contain mixed-use data sets, which leads to the need to segment bucket access by applications or teams. Granular access to individual objects or prefixes can be defined directly in bucket policy; however, the policy document is likely to become bloated as it defines access for dozens or even hundreds of Principals with different permission levels. S3 Access Points were created to overcome the limitations of bucket policy and make it easier to divvy up access to S3 prefixes.

S3 Access Points are simply named network endpoints, which are attached to S3 buckets. Attaching an access point to a bucket does not change anything about the underlying bucket. All existing operations against the bucket continue to work as before. S3 Access Points can be considered an abstraction on top of a bucket that can be created for every data consumer of a shared data set.

By default, the APIs used to access your Access Points are publicly available, just as with S3 Buckets. Optionally, Access Points can be combined with VPC Endpoints to provide network-layer controls. Creating an Access Point with the Network Origin ”VPC” restricts access to your named Access Point to the configured VPC.

Reference:
[1]: https://sec549.com/id167



S3 Access Point Policies
The greatest benefit of Access Points is its support of resource-based policies[1]. You can control who has access to the underlying S3 objects exactly like bucket policy but without the limitations of policy size.

Granting Access to Objects via S3 Access Point Policies
Permissions granted to a Principal via Access Point policy do not override permissions granted via bucket-level policy[2]. In order for actions to be successful against an access point, the operation must also be allowed via bucket policy. This requirement leads to an access control pattern where permissions at the bucket-level are delegated to the access point.

In the above diagram, you can see a bucket-level policy allowing actions against the bucket when the caller originates from an access point in a specific account. This has the effect of delegating granular access control decisions to the configured access points. As an example, a data user (Alice) has been given her own access point as a mechanism to manage her access to a specific prefix of S3 data. Her access is defined in resource-based policy attached to the access point unique to her and her use case.

References:
[1]: https://sec549.com/id168
[2]: https://sec549.com/id169

AWS | Allowing Access versus Denying Access in Policy

Allowing Actions **From** Access Points

- ALLOW Statement in bucket-level policy
- Allows Access Point to access objects in bucket
- Conditional statement allows actions when *"StringEquals"*

```
{
    "Version": "2012-10-17",
    "Statement" : [
    {
        "Effect": "Allow",
        "Principal" : { "AWS": "*" },
        "Action" : "*",
        "Resource" : [ "Bucket ARN", "Bucket ARN/*"],
        "Condition": {
            "StringEquals" : {
"s3:DataAccessPointAccount" : "123456789012" }
        }
    }]
}
```

Restricting Actions **To** Only Access Points

- DENY Statement in bucket-level policy
- Denies all actions in the bucket except when not originating from your Access Points
- Conditional statement denies actions except when *"StringNotEquals"*

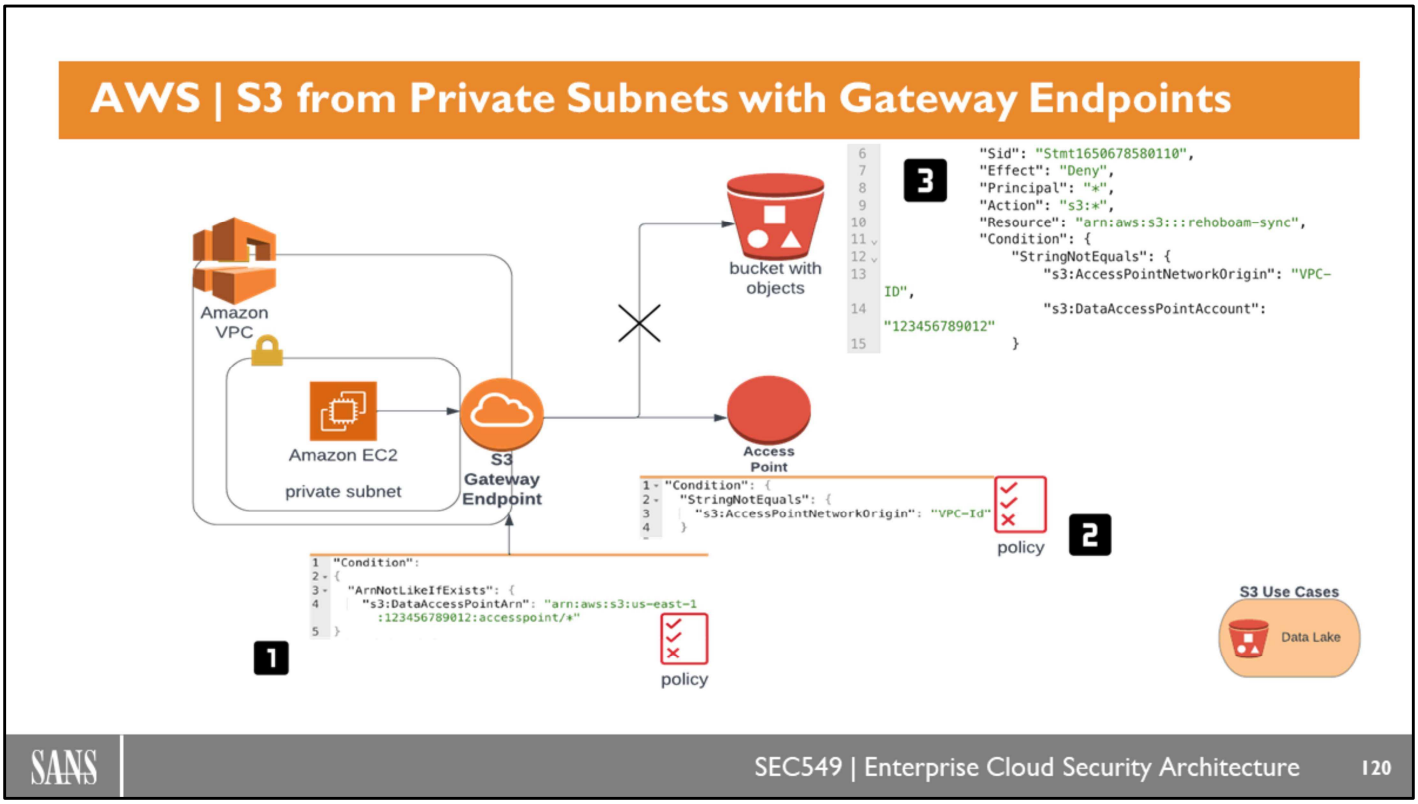
```
{
    "Version": "2012-10-17",
    "Statement" : [
    {
        "Effect": "Deny",
        "Principal" : { "AWS": "*" },
        "Action" : "*",
        "Resource" : [ "Bucket ARN", "Bucket ARN/*"],
        "Condition": {
            "StringNotEquals" : {
"s3:DataAccessPointAccount" : "123456789012" }
        }
    }]
}
```

Allowing versus Denying Access in S3 Bucket Policy

With Access Points, AWS customers can create a layer of abstraction on top of S3 Buckets, which is particularly useful for granting access to shared data sets residing in a single bucket.

As we've seen, a particular bucket-level policy is required to delegate access control policy to its access points. In this slide, we will highlight the importance differences between granting an access point permissions on objects versus restricting access to only specified access points.

Both ALLOW and DENY policy statements can and should exist in the same bucket-level policy document when the goal is to restrict access to only specific Access Points and no other callers.



Consuming Access Points From Private Subnets Using Gateway Endpoints
Access Points can be configured to accept requests only from a specified virtual private cloud (VPC). This configuration of access points leverages **VPC Gateway Endpoints** and configures a subnet route for private resources to reach public services like S3.

Network traffic between gateway endpoints and an S3 access point traverses the open internet; however, access to your S3 data can be restricted to your private VPCs if policies are configured with conditional policy statements.

In this configuration, there are 3 resource-based policies: bucket-level policies, access point policies, and VPC gateway endpoint policies. Permissions on Access Point prefixes can be defined on any of these resource-based policies. However, in order to prevent unauthorized access to data, there are condition keys in each component's resource-based policy that are needed.

1. VPC endpoint policy conditions[1]
The policy attached to VPC endpoints can optionally leverage the global condition key 's3:DataAccessPointArn' to restrict access though the gateway endpoint to all Access Points rather than Buckets in any given account. Here, instead of specifying individual buckets in the endpoint policy, an Access Point prefix can be used to specify all Access Points under an account. Use this condition to match a specific access point ARN or use wildcards to allow all Access Points in an account.

If a VPC endpoint policy does not scope access via the 's3:DataAccessPointArn' condition, then it must leverage the resource field to define a bucket or access point.

2. Access Point Policy Conditions
The policy attached to S3 Access Points should leverage the global condition key 's3:AccessPointNetworkOrigin' to restrict access only to callers originating from the VPCs you specify. This condition is used to both Deny actions and Allow permissions.
- Example: The "getObject" API call can be denied when the caller does not originate from your specified

VPC and allowed the ”getObject” API call only when the caller originates from your VPC.

3. Bucket-level Policy Conditions

The policy attached to an S3 Bucket can leverage both the global condition key ‘s3:AccessPointNetworkOrigin’ and ‘s3:DataAccessPointAccount’ to restrict access and delegate permissions to the specified access points. These conditions are used to both Deny actions and Allow permissions.

- Example: The ”getObject” API call can be denied when the caller does not originate from your specified VPC and allowed the ”getObject” API call only when the caller originates from your VPC.

Service Control Policy

You may have corners of your Organization (Production OUs) where Access Points should always be consumed privately and never created for public access. To enforce this, create an SCP dictating that all access points be created with a VPC network origin.

References:
[1]: https://sec549.com/id170
[2]: https://sec549.com/id171

AWS S3 Use Cases – Application Data Store

Allowing Application Components To Securely Access S3

Identity-Layer

- Use resource-based policy to grant application components access to S3 objects
- Grant access to:
  - AWS Service Principals
  - IAM Roles

Network-Layer

- Privately deploy access to your S3 buckets allowing VPC-bound resources like EC2 to retrieve objects

AWS Use Case as an Application Data Store

In patterns ranging from classic three-tiered web applications to event driven serverless applications, there will always be a need for AWS services to interact with S3 Buckets and Objects. Connecting all components of these solutions along both the network-layer and the identity-layer is what the next few slides cover in detail.

AWS – Allowing Services to Access S3 Objects

Granting an AWS Service the ability to access objects stored on S3 directly through **bucket policy:**

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSCloudTrailWrite20150319",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "cloudtrail.amazonaws.com"
                ]
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::MyBucketName/*",
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": "bucket-owner-full
                      -control",
                    "aws:SourceArn": "arn:aws:cloudtrail
                      :region:111111111111:trail/trailName"
                }
            }
        }
    ]
}
```

Shown here is a bucket policy allowing the AWS CloudTrail service to write objects to the bucket as long as CloudTrail is writing from the AWS Account and trail as we expect.

By specifying a *SourceArn* condition, we can prevent confused deputy vulnerabilities

Granting Service Principals Access to S3 Objects

A Service Principal is a type of AWS Principal. They can be granted permissions by direct inclusion into resource-based policies but cannot be assigned an identity-based policy.

- *Example:*
  - A Service Principal can be granted access to S3 objects if they are specified as the Principal in a bucket policy.
  - A Service Principal cannot be assigned the S3ReadOnly AWS-Managed Role.

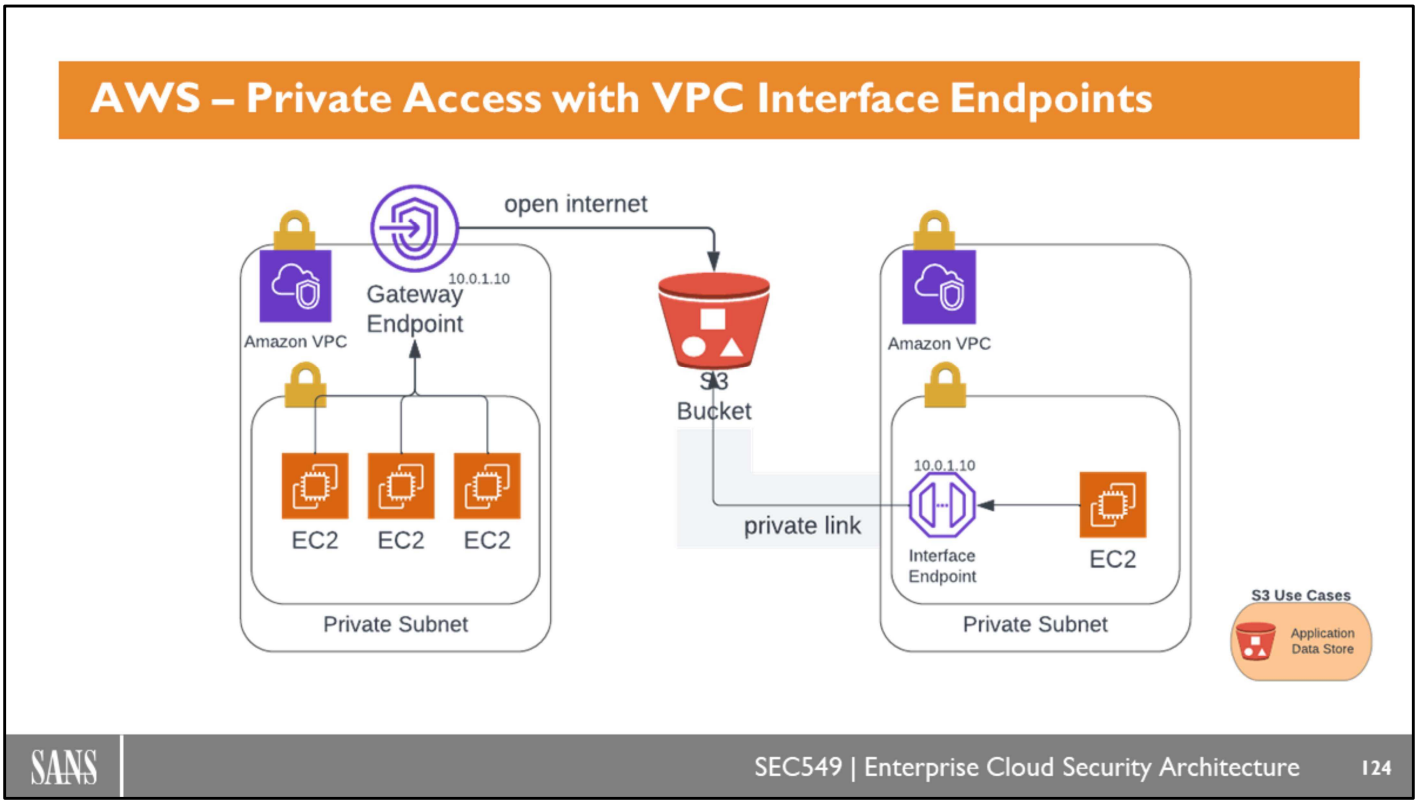
Many AWS services will need to read or write data directly from S3 as a result of configured integrations. There are two patterns used in AWS to allow a Service Principal access to resources[1]:

1. Allow the AWS service to assume a role by specifying the relevant Service Principal in trust policy
2. Grant the service principal direct access to resources through inclusion in resource-based policy.

Which pattern is used depends uniquely on the service-to-service integration. Below are instances where pattern 2 is used. The Service Principal for a given AWS service must be granted access directly in resource-based bucket policy.

- CloudTrail writing logs buckets
- CloudWatch reading and writing metrics from S3
- Amazon RDS reading and writing backups from S3

Reference:
[1]: https://sec549.com/id172



Private Access to S3 with VPC Interface Endpoints[1]
In a previous slide we looked at the consumption of Access Points from private subnets via VPC Gateway Endpoints. Another variety of Endpoints exists for accessing S3 objects - VPC Interface Endpoints.

VPC Interface Endpoints run on top of AWS Private Link. As opposed to Gateway Endpoints, which provide a route out of a private subnet to a public resource, VPC Interface Endpoints provide a connection over the AWS backbone to the resource. VPC Interface Endpoints allow you to configure a 'back door' to your resources. These two types of access are likely to coexist when designing for private S3 consumption.

Currently, two types of VPC endpoints can be used to connect to Amazon S3 – VPC Interface Endpoints and VPC Gateway Endpoints. The advantage of using Interface Endpoints is the private connection. To satisfy security or regulatory needs, you might choose Interface Endpoints, which route the traffic over the AWS service backbone, rather than the open interface. Additionally, the pattern-of-use for VPC Interface Endpoints tends to be 1-to-1. That is, for every consumer of a resource, there is a corresponding interface endpoint made available.

If a private connection is not a requirement, however, Gateway Endpoints might be a simpler (and more cost effective) deployment pattern. Gateway endpoints are nothing more than route table entries that route subnet traffic to the configured S3 service. Traffic does not flow through an intermediate device or instance, as such Gateway endpoints for S3 are offered at no cost.

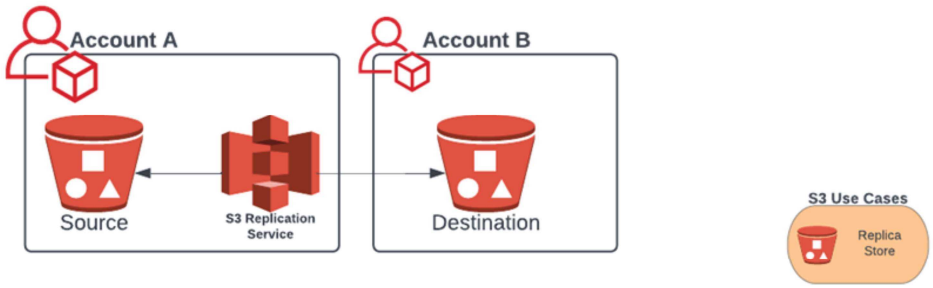
Both endpoint types support resource-based policy, so it's important whether you are working with an Interface Endpoint or a Gateway endpoint, to always specify the resource ARN in the policy. This will restrict access through the endpoint to the resource specified, whether that is a S3 Bucket, Access Point, or a DynamnoDB Instance.

Reference:
[1]: https://sec549.com/id173

AWS S3 Use Cases – As a Replica Repository

S3 is both a data store and a repository for real-time backups

- Replication across regions or between different accounts
- Replication of objects encrypted with KMS keys
  - Keys can be housed in the same account or cross-account



S3 as a Replica Repository
Protecting data in S3 starts with robustly following the principal of 'least privilege', which dictates that the least amount of authority and minimum access should be granted to accomplish a given task. Even with the most stringent preventative controls, we must plan for the day when preventative control measures fail. Designing for resiliency can help limit the damage from a material event and any inconvenience it might cause, whether the event is security related or not.

Resiliency on S3
AWS S3 provides a built-in mechanism to backup objects called S3 Replication. S3 Replication can be configured to automatically sync data between buckets across regions or even between different accounts. This feature of S3 is useful for asynchronously moving objects, whether the desired goal is fast access to your data in different regions or creating a general backup of the data within the bucket.

S3 Replication For Creating Backups
To enable the recovery of a system following a data incident, a reliable backup of the data must exist. This requirement should carry forward to your cloud environments. With the S3 Replication service, the regular backup and copying process of your data on S3 can be automated. Other critical components of a backup strategy, such as the data restoration process and recovery testing, need to be addressed with processes rather than technology in your cloud security program.

These next few slides are intended to highlight settings and features of the S3 Replication Service so it can be leverage to help mitigate an event affecting S3 availability.

AWS | S3 Replication Rules

Replication Rules Define

- Source Bucket
- Destination Bucket(s)
  - Objects can be replicated to a single or multiple buckets
- Scope of Replica
  - Which objects should be replicated
- Metrics
  - Details of replications such as latency sent to CloudWatch
- Encryption Context
  - Do the objects need to be decrypted with a KMS key and re-encrypted with KMS upon copy

Replication Rules for S3 Replication Service
Replication Rules define the 'who', 'what', 'where', and 'how' of your S3 backup automation[1]. These rules inform the S3 Replication Service which objects to copy, from which bucket to replicate the data, and where to create the duplicated data.

Out of the box, the S3 Replication Service will only replicate objects as they are uploaded to an S3 bucket. You have two options should you need to replicate existing objects. S3 objects can be re-copied into an S3 bucket that has replication configured (this is potentially costly and an onerous process) or you may engage AWS Support for assistance.

The ability for the S3 Replication Service to duplicate objects across AWS accounts is critical for ensuring your backups are isolated from any material event. This capability could also be used maliciously by a threat actor looking to migrate data outside of your AWS environment. As such, the permissions allowing for the creation and updating of Replication Rules should be treated as a highly sensitive administrative action. Custom alerting should be configured in your SIEM tool to notify responders when Replication Rules are created or updated.

Reference:
[1]: https://sec549.com/id174

Cloud-Native Ransomware Resilience – S3 Versioning

- S3 will create versions of objects during state changing requests
  - 'Deleted' objects are only marked as deleted but are still retrievable
  - Overwritten objects are not deleted but are stored as versions
- You can use versions as form of an object 'backup'
  - Permissions for accessing objects and object versions are different
  - Withhold the permission for accessing object versions to all but Account admins
- Object Lifecycle Policies
  - Ensure lifecycle policies are in place, removing old versions of objects

Using S3 Versioning for Cloud-Native Ransomware Resilience
You can add to the ransomware resiliency of your S3 buckets and their objects by enabling versioning1. This feature of S3 in not enabled by default and alters the default behavior of how S3 handles objects when they are changed. With versioning enabled, whenever there is a change in the state of the object, a new copy of the object is created. The preexisting object is not deleted but is renamed with the addition of a marker indicating it is an older version. This object and all prior versions of the same object are stored and remain retrievable.

This practice of versioning becomes an excellent protection against ransomware, because the adversary would also need to affect object versions in addition to the objects themselves. The actions used to manipulate an object and its various past versions are handled as separate permissions. If permissions to operate against object versions are handled judiciously, object versioning can be another layer of defense against attacks on S3 availability.

Reference:
[1]: https://sec549.com/id175

Cloud-Native Ransomware Resilience – S3 Object Lock

S3 Object Lock

- A method of enhancing data resiliency by locking an S3 object against any changes made, including the deletion of the object for a fixed period of time

Overriding **Object Lock** settings require specific permissions

- s3:BypassGovernanceRetention
- s3:PutBucketObjectLockConfiguration
- s3:PutObjectLegalHold
- s3:PutObjectRetention

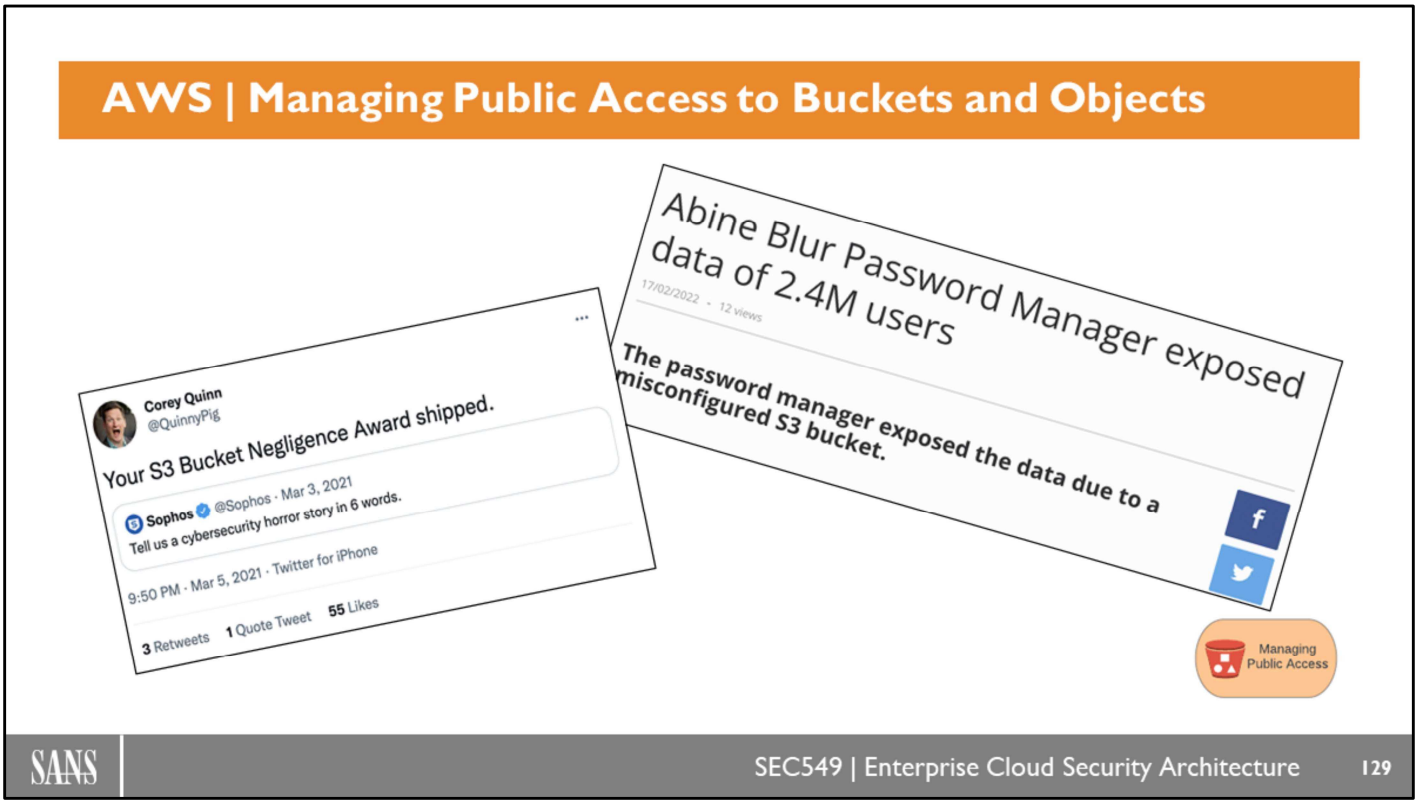
Using S3 Object Lock for Cloud-Native Ransomware Resilience

AWS offers an option called S3 Object Lock, which ensures that an S3 object remains 'alive' but locked (and unable to be deleted) until a specific date without exception1. Data becomes inherently more durable using this feature; however, it is impractical for highly transactional data that is continuously updated or modified. Still, it remains excellent method of adding to the ransomware resiliency of sensitive data and is especially applicable to backups

S3 Object Lock can be enabled on any new S3 bucket at the time of its creation using one of the advanced setting options. If a specific retention period is required based on a legal hold, that time period can be configured to apply to specific objects or all objects in the bucket. S3 Object Lock cannot be applied at the account level, only on a per bucket basis.

If an AWS Account is maintained for the explicit purpose of housing data on legal hold or S3 backups, you might consider enforcing S3 Object Lock with an SCP.

Reference:
[1]: https://sec549.com/id176
Image by jeferrb from Pixabay: https://sec549.com/id133



Managing Public Access to Buckets and Objects

Misconfigured S3 Buckets might be the most ubiquitous source of cloud data breaches. Efforts have been made to track these customer incidents, including those directly tied to publicly accessible S3 buckets and other cloud-related breaches. Below are a few projects that have undertaken the efforts in tracking known cloud breaches.

- **Upguard:** https://sec549.com/id177
- **S3-Leaks**: https://sec549.com/id178
- **aws-customer-security-incidents:** https://sec549.com/id179

What the security community has learned in examining the root causes of data incidents is that securing cloud storage buckets is hard, even with objects and buckets being private by default and despite recent tools like Access Analyzer for S3.

Over the next couple of slides, we will cover native controls you can apply in AWS to avoid having public buckets and showcase patterns you can employ when public buckets are required.

AWS | S3 Block Public Access Settings

Block Public Access Settings

BlockPublicAcls	IgnorePublicAcls	BlockPublicPolicy	RestrictPublicBucket
Reject objects if the ACL specified during upload configures an object to be public	Ignore is an ACL if it attempts to configure an object to be public	Reject bucket-level policies if they would make public any objects in the bucket	Limits access to objects to Principals in the same AWS Account, despite any allowance by bucket-level policy.

Can be set at several levels:

- AWS Account Level
- Individual Bucket Level
- Access Point Level

If any of these settings indicate that the request should be blocked, Amazon S3 rejects the request

S3 Block Public Access Settings
Block Public Access Settings are a collection of 4 restrictions that can be applied at various enforcement points. These controls are aimed at preventing public access to objects and buckets through various access control mechanisms.

Enforcement points for Block Public Access Settings:

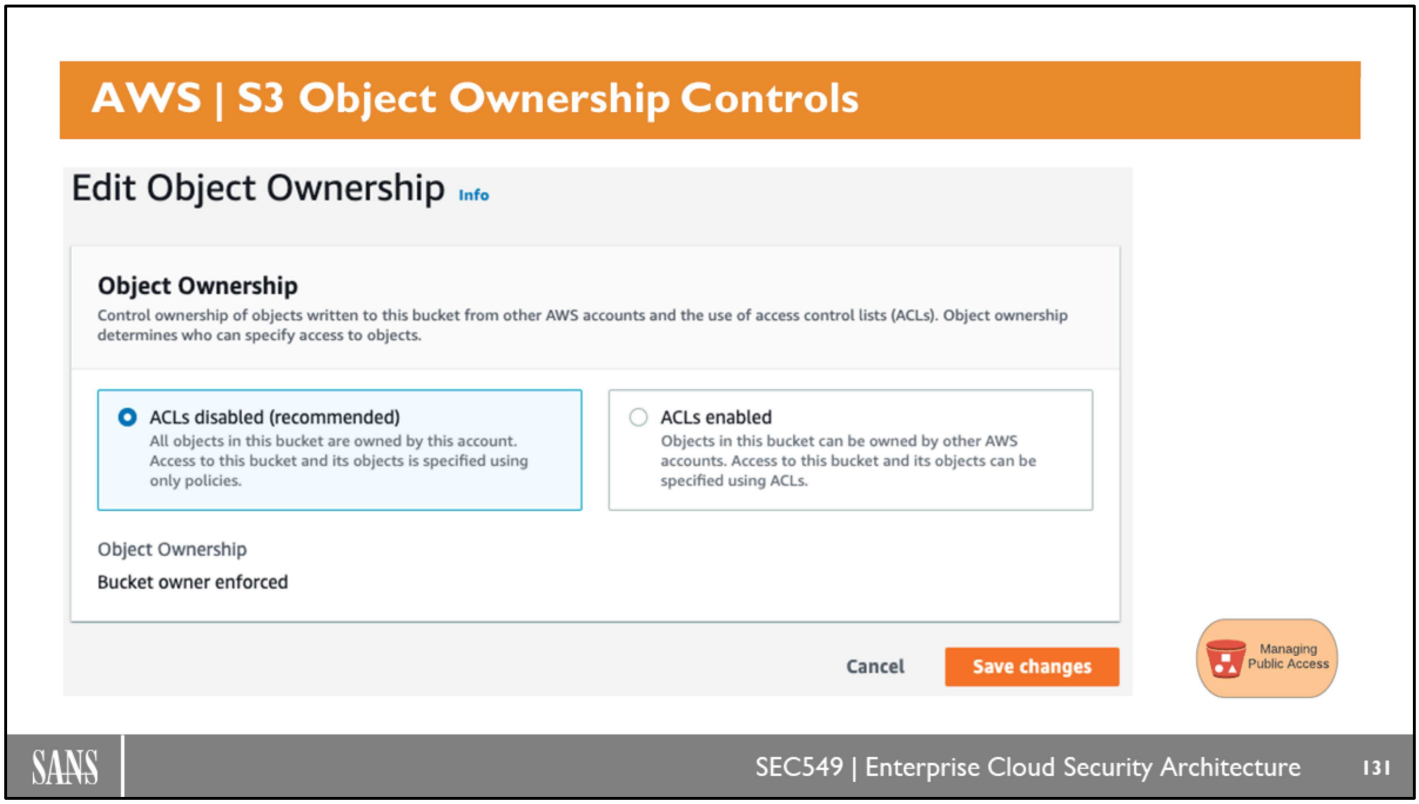
- AWS Account Level
- Bucket Level
- Access Point

AWS does not support applying any Block Public Access Settings at the object level or per prefix. If a Block Public Access Settings is applied at the Account-level, it cannot be overridden at the Bucket-level.

Architecture Considerations for Managing Public Buckets
While AWS provides the ability to apply Block Public Access Settings, more granular levels such as the bucket-level or access point level may be needed. It is preferable to configure and enforce these settings at the highest resource level possible and enforce the settings with SCP. In order to be able to be able to set all 4 Block Public Access Settings to true at the Account Level, you need a disciplined approach to public buckets.

Any sufficiently large enterprise will find a use case for a public bucket at some point which, in turn, leads to the need to revert one or all 4 Block Public Access Settings.

These authorized, intentionally public buckets should be corralled into a single account, potentially in their own OU, in order to make to the Block Public Access Settings easier to apply account-wide and enforce with OU-level SCP.



S3 Object Ownership Controls
Access Control Lists (ACLs) serve as an alternative method for granting access to S3 Buckets and Objects. Because they operate separate from IAM, the types of access ACLs granted to S3 Buckets and Objects can feel obscure and are difficult to audit.

By default, when a Principal in another AWS Account uploads an object to your S3 Bucket, that account (the object writer) owns the object as a result of an ACL that is defined upon upload. With Object Ownership Controls, you can change this default behavior.

Object Ownership Controls consists of 3 settings:
1. **Bucket Owner Enforced**: This is the default setting when creating a Bucket on the console. This setting will disable ACLs as an access granting mechanism. The Bucket owner will automatically own and have full control of every object.
2. **Bucket Owner Preferred:** ACLs are not disabled. Objects can change ownership to the Bucket owner only if specified during object write.
3. **Object Writer**: This is the default setting when creating a Bucket programmatically and the default behavior before the introduction of the Object Ownership Controls in 2021[1]. With this setting, the Principal who uploads an Object is granted full control over the object via ACLs.

Setting an Object Ownership Control can only be done on new or existing S3 Buckets. This setting cannot be selected at the AWS Account level; however, you can ensure the BucketOwnerEnforced setting is applied to all newly created Buckets with a judiciously applied SCP.

AWS | SCP Policy for Restricting S3 Object Access

Prevent users from modifying S3 'Block Public Access' setting

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutAccountPublicAccessBlock"
            ],
            "Resource": "*",
            "Effect": "Deny"
        }
    ]
}
```

Prevent users from creating buckets without the *BucketOwnerEnforced* setting

```
{
    "Version": "RequireBucketOwnerFullControl",
    "Statement": [
        {
            "Action": [
                "s3:CreateBucket"
            ],
            "Resource": "*",
            "Effect": "Deny",
            "Condition": {
                "StringNotEquals": {
                    "x-amz-object-ownership":
                      "BucketOwnerEnforced"
                }
            }
        }
    ]
}
```

SCP Policy for Restricting S3 Object Access

With controls like Block Public Access and Object Ownership Controls, an enterprise can have greater assurance that Buckets and Objects will not mistakenly be made public, resulting in a headline grabbing event.

Now that we know which controls to apply, we should consider enforcing those settings with SCP. The level at which these policies should be applied depend upon the account structure of your AWS Organization and whether or not any Accounts require exceptions to these policy enforcements.

1. **Prevent Users from Modifying S3 Block Public Access**[1]: By default, Accounts and Buckets enforce all 4 Block Public Access Settings. There is nothing you need to enable this setting; instead, an SCP is required to prevent it from being modified or deleted.
2. **Prevent Users from Creating Buckets Without the BucketOwnerEnforced Setting**[2]: While this is the default condition when creating Buckets on the CLI, it is not the default when creating buckets programmatically. This SCP will block any CreateBucket event if the BucketOwnerEnforced setting is not specified during creation.

References:
[1]: https://sec549.com/id180
[2]: https://sec549.com/id181

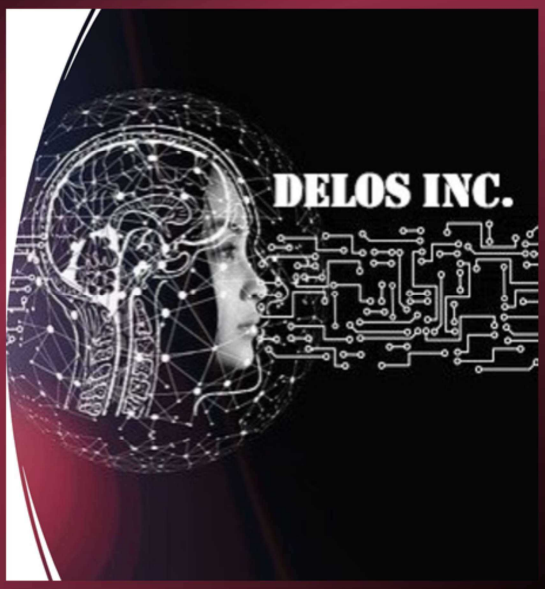
Cloud Journey Phase: Data Growth!

Scenario: Leveraging the cloud for Fine-grained access

Data sets collected across the Delos Global have been ingested into AWS and warehoused in a single S3 bucket in support of a project called 'Rehoboam'. The rest of the business has noticed this wealth of information and is taking advantage of these data points to drive business decisions. As such, there has been an explosion of requests for data access.

Access-controls on the data are entirely defined via the single bucket-level policy document. As a result, whenever a new business unit wants to access data sets, the document needs to be updated, making it a single point of failure.

Your goal is to create new access points representing the different data consumers and, where appropriate, restrict access to data subsets at the network-level.

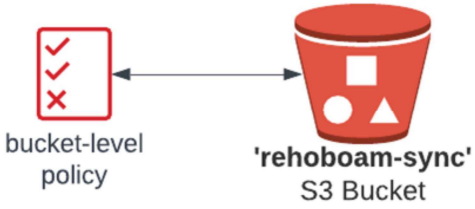


Reference:
Image by user geralt from Pixaby: https://sec549.com/id51

Lab 2.3 – Current State Architecture

Currently, all **rehoboam** data resides in a single S3 bucket. Access to the data is controlled with via statements in a single S3 bucket policy.

- Delos' monitoring account (ID: 455307868549) and the AI Admin group is granted access to the *'/error-report'* prefix through a Role in the rehoboam-data sync account (ID: 925806920140)
- The Delos monitoring account (ID: 455307868549) can also access *'/health-checks'*
- The Data Science team is provided a role in their 'robots-experiment' Account to access the *'/status'* and *'/interactions'* prefix to facilitate the training of AI prediction models
- Finally, an AI-rendering application running on EC2 is provided a Role in the rehoboam-data sync account (ID: 925806920140) to access the *'/maze-plots'* prefix



This page intentionally left blank.

Lab 2.3 – Access Control for Shared Data Sets

Estimated Time: 45 minutes

In this Lab, you'll notice that the architecture has evolved from the 'current-state' to include 4 access points on top of the *'Rehoboam-sync'* AWS S3 Bucket and a VPC Interface Endpoint in the *'delos-web-prod'* account.
Arrange each of the policy documents into the six slots provided so that consumers of data maintain their access while transitioning to consuming objects via access points.

Preparation

- View **Lab 2.3** in the course workbook
- Open the workbook in an incognito window
- https://workbook.sec549.com
  - Username: H03-student
  - Password: million-sailor-DEAR

Objectives

In the provided template sequence diagram:

1. Delegate access control from the bucket-level to the access points.
2. Configure Access Point policy to allow access to specific prefixes.
3. Configure VPC Endpoint policy allowing private S3 access by the Maze Rendering application

This page intentionally left blank.

Lab 2.3 – Summary

In this lab, you...

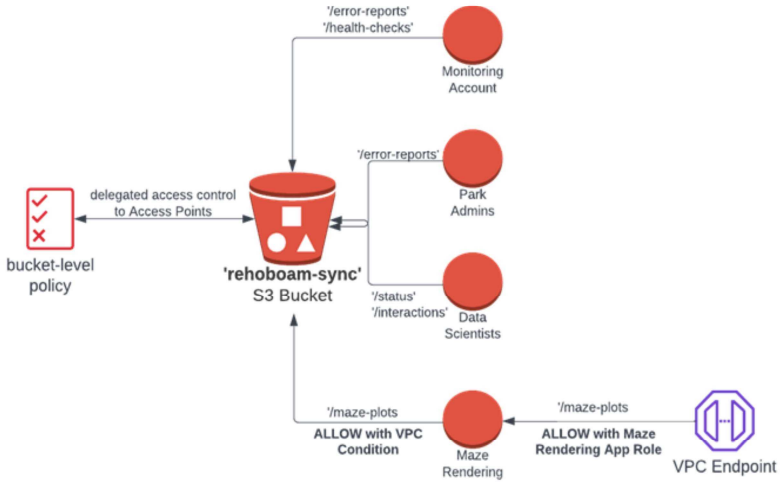
- Identified the 4 data consumers and indicated which prefixes they should be granted access to
- Configured bucket-level policy delegating access control to the access points
- Configured access point-level policy ensuring parity with older bucket-level access models
- Configured VPC endpoint policy, allowing the Maze Rendering application to access the '/maze-plot' prefix
- Logged into the AWS Console to answer the ⚡ See it in Action ⚡ questions

This page intentionally left blank.

Lab 2.3 – End State Architecture

Access Points have been created for each data consumer, the Monitoring Account, Data Science Team, Park Admins and the Maze-Rendering App.

This pattern successfully breaks up the clunky bucket-level policy into smaller, more manageable Access Point Policies and allows for the prefix requiring private access to be consumed via a VPC Interface Endpoint.



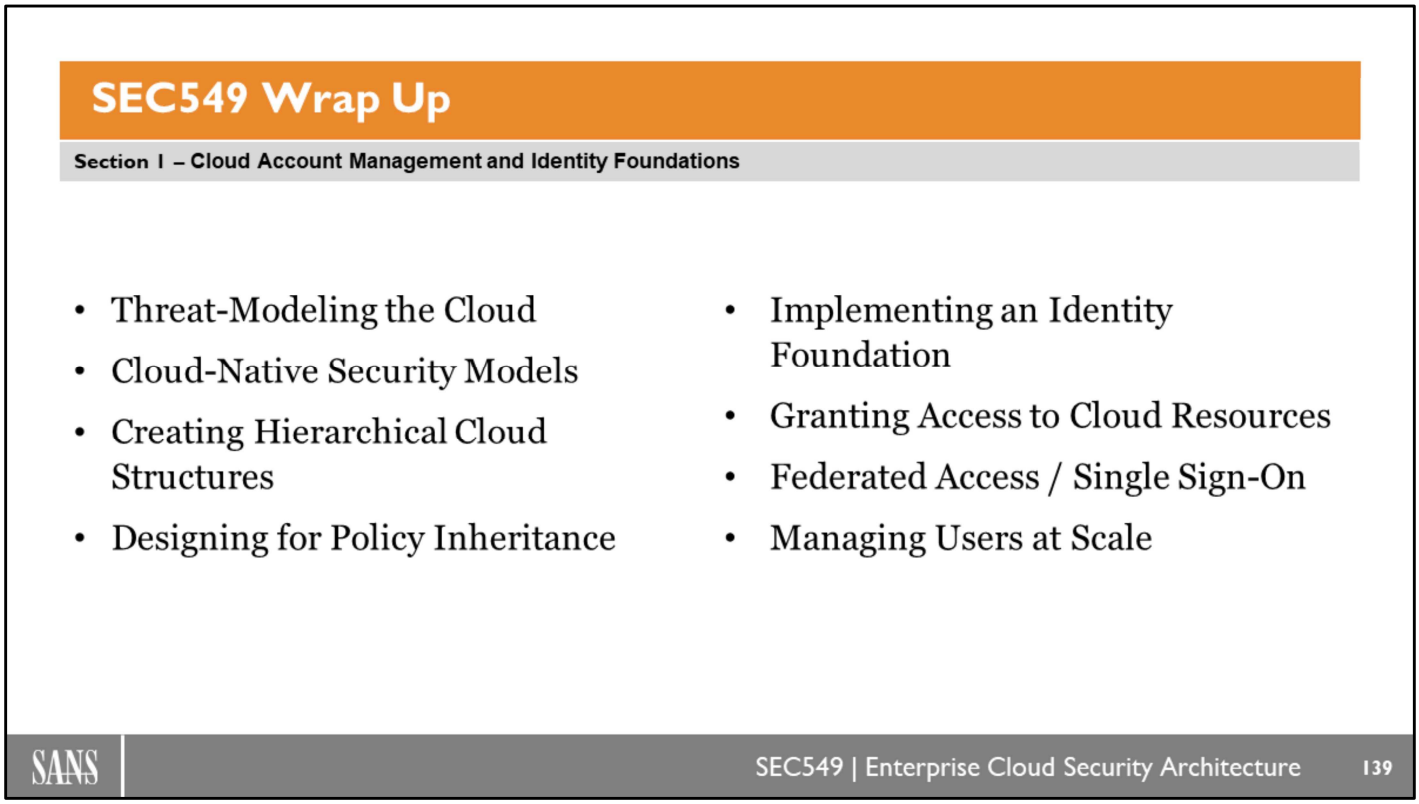
End State Architecture
Access Points have been created for each data consumer, the Monitoring Account, Data Science Team, Park Admins and the Maze-Rendering App.

Each newly created Access Point has resource-based policies attached granting access to only specific, corresponding prefixes.

The 'Maze Rendering' Access Point is consumed privately from a VPC Interface Endpoint. Its resource-based policy allows actions only when the caller originates from the delos-web-prod VPC.

VPC Endpoint in the 'delos-web-prod' Account is provisioned in its private subnet allowing backend components of the application to privately access the '/maze-plots' prefix. Policy attached to the VPC Endpoint scopes permissions to the 'Maze Rendering' Access Point with the resource field.

This page intentionally left blank.

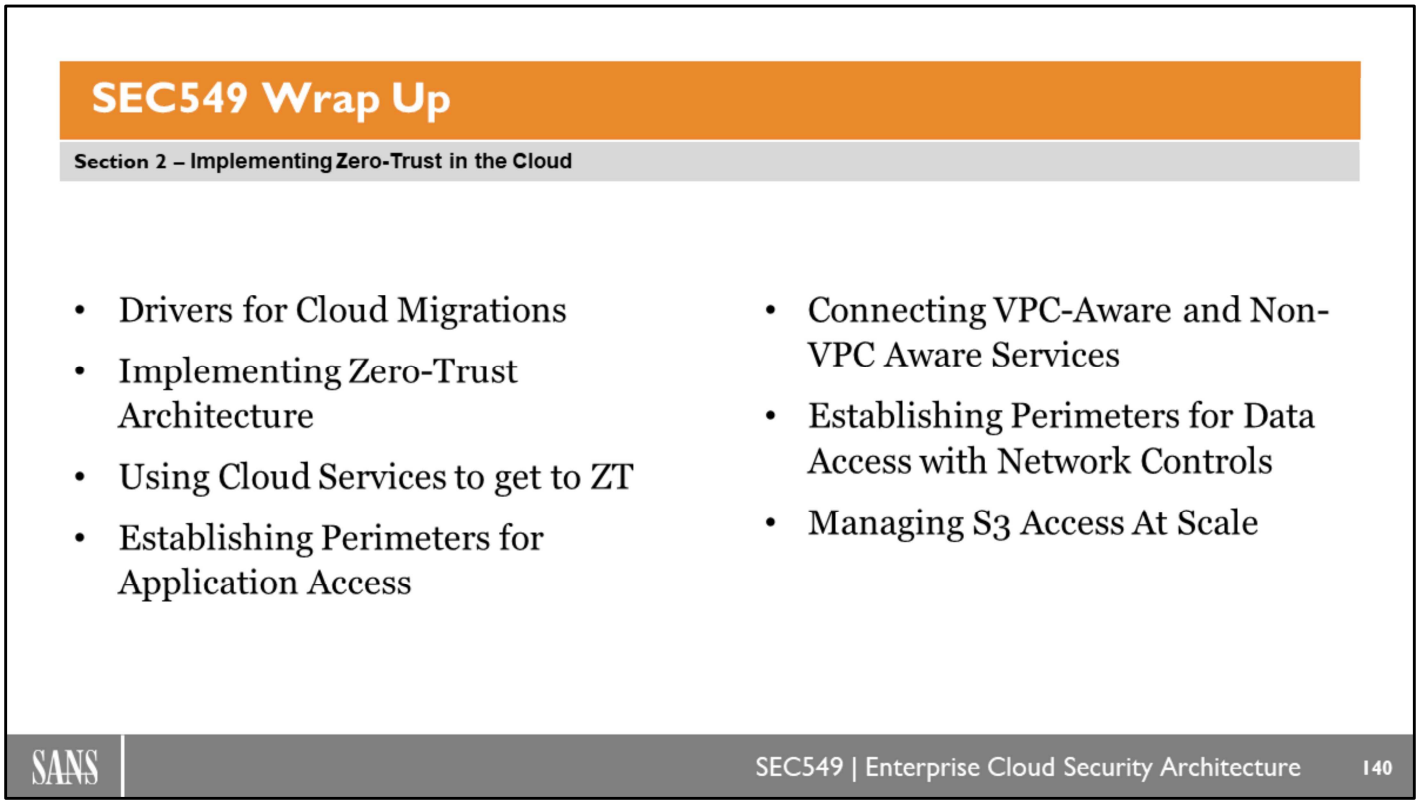


Section 1 Wrap Up

In the first section, you learned the unique aspects of security architecture in the cloud, emphasizing simplicity, least privilege, open design, and with rational defaults set to fail closed. We covered threat modeling as a tool to understand and mitigate risk. It is a team effort involving a diverse group of stakeholders who collectively model the system and data flow in order to document and identify risks. You learned that cloud security is based on securing through policy and that identity, in particular, becomes an increasing important perimeter in the cloud.

This course then covered the higher-level resources in the cloud. You learned that there is a one-way ownership structure from parent to child and the possibility of either one-to-one or one-to-many relationships between the different levels. Thinking of how resources in the cloud can be a policy attachment point, we dived into how policy inheritance goes down the hierarchy but never up the hierarchy as well as how your hierarchy design should be driven by IAM inheritance and the need for clear resource ownership.

Heading into the back half of Section 1, we covered different options for cloud-native identity directories and how to think about the identity constructs in AWS: Principle, Root User, IAM User, Group, and IAM Role. You learned the two types of policies in AWS, Identity-based Policy and Resource-based Policy, as well as the essential components of a policy. Thinking about how to scale your cloud estate as your environment grows, this course taught you federated access methods for allowing users to log into a resource with an external identity provider.



Section 2 Wrap Up

Section 2 kicked off by covering the drivers for cloud migration, including reduced costs, scalability, enhanced security, and increased flexibility, along with a review of the 5 Rs that are the strategies for cloud migrations: Rehost, Refactor, Revise, Rebuild, and Replace. Central to this section were the concepts behind the Zero-Trust movement. You learned that ZT principals can be used to enhance security by adding granular, layer 7 controls to create micro-perimeters around the most vulnerable assets. Building on our understanding of Zero-Trust in the cloud, we took a deep-dive look at Cognito and how it can be used to uplift legacy authorization schemes at the application layer.

Looking at network-layer controls, you reviewed networking and learned how the cloud service providers offer their own virtual private networks (VPNs) and the mechanisms to connect isolated virtual networks. You learned the ways the different CSPs can allow for on-premises resources to securely connect to your defined VPCs using VPNs (virtual private networks) and other options like AWS Direct Connect.

This course covered S3 bucket uses cases as a blueprint to discuss security controls available for your data perimeter. We looked at using S3 for: serving static web content, data-lakes, application backend, and replica store use cases. You learned how to apply policies for secure access to S3 buckets and the perils and some of the pitfalls involved.

Authors, Contributors, and Special Thanks

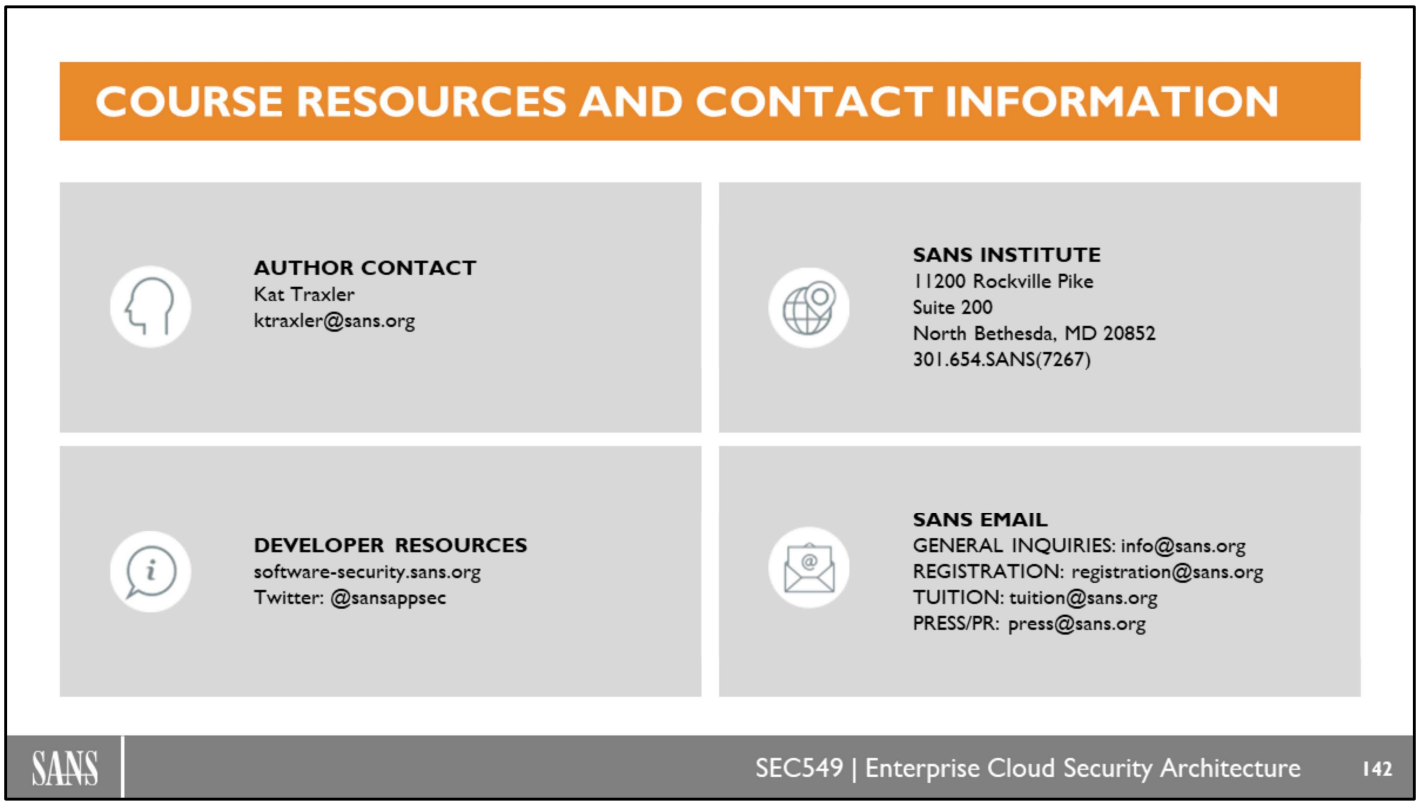
Author:

- Kat Traxler
  - ktraxler@sans.org
  - @NightmareJS

Special Thanks:

- Moises Margotto
  - @mlmargotto

This page intentionally left blank.



This page intentionally left blank.

This page intentionally left blank.