MODULE 12 · STUDY NOTES

Developing Secure Applications on AWS

Securing the connections into your application and controlling who can reach your AWS resources — TLS and ACM certificates, temporary credentials from AWS STS, and authentication with Amazon Cognito user pools and identity pools

Securing an application is not one task but two, and both fall on the developer's side of the AWS shared responsibility model: AWS secures the infrastructure *of* the cloud, and you are responsible for security *in* the cloud, at every layer. The first task is **securing the network connections** between clients and servers over the internet — SSL/TLS encryption, certificates issued by certificate authorities, and AWS Certificate Manager to buy, deploy, and renew them. The second is **controlling user access** to the AWS resources your application calls, always with temporary credentials rather than long-lived keys. That story runs from AWS STS, which issues short-lived credentials to trusted IAM and federated users, up to Amazon Cognito, which adds full authentication, authorization, and user management: user pools that sign users in and issue JWTs, and identity pools that trade those identities for temporary AWS credentials. The café running example ties it together — giving employees a login that unlocks features external users can't reach.

LEARNING OBJECTIVES

- List two aspects of securing an application
- Recall how to authenticate with the AWS Security Token Service (AWS STS)
- Describe how to build secure applications with Amazon Cognito
- Secure part of a website with Amazon Cognito

12.1 Application security and the shared responsibility model

A web-based application has layers — content delivery, load balancers, application servers, databases — and each is a place where security must be considered. Under the **AWS shared responsibility model**, AWS is responsible for the **security of the cloud**, and **you are responsible for security in the cloud**. As a developer, you carry that responsibility at *all* layers of the application. You have already met two services that are core to AWS security: **Amazon VPC**, which secures your networks and subnets and protects your resources, and **AWS IAM**, which controls access to AWS resources through policies that grant permissions to users, groups, and roles. This module adds two more application-level aspects on top of that foundation.

THE TWO ASPECTS THIS MODULE ADDS

- **Secure the network connections** between application clients and servers over the internet — encrypt traffic with SSL/TLS, establish identity with certificates from a trusted CA, and offload the work with ELB, CloudFront, and ACM.
- **Control user access** to the AWS resources the application calls — grant access with temporary (not long-lived) credentials using IAM roles, AWS STS, and Amazon Cognito; authenticate the user, then authorize what they may do.

THE CAFÉ SCENARIO

Sofía wants to modify the café website to provide a **login for café employees** that gives them access to features that aren't available to external users. She doesn't have much experience with authentication and authorization. Faythe, an AWS developer and café regular, suggests **Amazon Cognito** — it is easy to set up and integrates with the AWS services the café website already uses. By the end of the module you will secure part of the website with Cognito so that an employee (Frank) can log in and request an inventory report that external users cannot.

COMMON PITFALL

Security in the cloud is not AWS's job to do for you. AWS secures the underlying infrastructure, but configuring encryption, certificates, access control, and authentication for *your* application is your responsibility — at every layer.

12.2 Securing network connections

Transport Layer Security (TLS) and its predecessor **Secure Sockets Layer (SSL)** are open standards that use public and private **certificates** to establish the identity of websites over the internet and resources on private networks, and that **encrypt** network communications between connected resources. Amazon.com uses TLS for all traffic on its website, and AWS uses TLS to secure API calls to AWS services. An HTTPS connection might use either SSL or TLS; although the term SSL is still widely used, **TLS is typically what secures the connection** today.

WHERE TLS IS HANDLED IN AN AWS ARCHITECTURE

- **Elastic Load Balancing** simplifies building secure web apps by **terminating HTTPS and TLS traffic at the load balancer** — the load balancer encrypts and decrypts, so each EC2 instance doesn't have to do TLS termination.
- **Amazon CloudFront** can be configured to **require viewers to use HTTPS**, and to use **HTTPS with your origin**; CloudFront performs SSL/TLS negotiation between viewer and CloudFront, and between CloudFront and the origin when the response isn't already cached.
- An **origin access identity (OAI)** in CloudFront, combined with S3 bucket permissions, lets **only CloudFront** read the bucket — users can't access the content directly even if they have the URL.

KEY TERMS

Certificate authority (CA)
An entity that issues certificates to specific domains. When a domain presents a certificate that a **trusted** CA issued, your browser or application knows it is safe to make the connection.

The handshake proceeds in sequence: the **CA issues a certificate** to the server's domain; the **client** (a web browser or application) **requests identification**; the **server sends its certificate and public key**; the client checks **whether the issuing CA is trusted**; the client **sends an encrypted session key**; the server returns an **acknowledgement encrypted with that key**; and from then on **all data is encrypted with the session key**. Managing these certificates by hand, however, is difficult — which is exactly what pushes teams toward a managed service.

TABLE 12.1 Challenges of managing certificates manually

CHALLENGE	WHY IT IS HARD
Security	Certificates can be vulnerable for many reasons — name mismatch, internal names, missing or misconfigured fields, weak or outdated hashing algorithms, weak keys, weak cipher suites; finding and fixing these is challenging
Discovery	It is impractical to manually gather details about hundreds or thousands of individual certificates across a network
Rotation and renewals	Manual tracking of expirations is error-prone and can cause you to miss rotations and renewals
Authorization	You must be able to verify that someone is authorized to approve and issue a certificate
Cost	Beyond validation fees, there are support, legal (insurance, warranties), ownership/control (such as root-embedding), and infrastructure costs

AWS Certificate Manager (ACM) removes that burden. With ACM you **provision, manage, and deploy** public and private SSL/TLS certificates for AWS services and your internal connected resources, eliminating the manual process of purchasing, uploading, and renewing certificates. You can quickly request a certificate and deploy it on ACM-integrated resources — **ELB load balancers, CloudFront distributions, and APIs on Amazon API Gateway** — and ACM **manages the renewals**. You can also create private certificates for internal resources and centrally manage the certificate lifecycle.

WHAT TO REMEMBER ABOUT ACM

- **Public and private certificates provisioned through ACM for use with AWS services are free** — you pay only for the AWS resources you create to run your application.
- You can produce your own certificate with **AWS Certificate Manager Private Certificate Authority (ACM Private CA)**.
- ACM makes it easy to enable SSL/TLS, helping organizations meet regulatory and compliance requirements for **encryption of data in transit**.

COMMON PITFALL

Don't assume ACM certificates cost money or that you must renew them yourself. Certificates provisioned through ACM for use with AWS services are free, and ACM handles renewal automatically — the fee model is that you pay only for the AWS resources running your application. *Section takeaways:* encrypt with SSL/TLS, which require a CA-issued certificate; ACM simplifies their management and renewal.

12.3 Authenticating with AWS STS

The second aspect of security is **controlling application users' access** to your AWS resources. The recommended approach is to grant access with **temporary credentials**, using **IAM roles**, **AWS STS**, and **Amazon Cognito** in conjunction with IAM. This section covers STS; the next covers Cognito. First, the vocabulary of access control.

KEY TERMS

Authentication
Verifies the identity of the user — who they are.

Authorization
Verifies the user's permissions — what the user is allowed to do.

Identity provider (IdP)
Manages identity information and provides authentication services.

Identity broker
A software layer that authenticates credentials against an IdP and retrieves temporary security credentials from AWS STS.

SAML (Security Assertion Markup Language)
Open standard used to exchange authentication and authorization data between parties.

OIDC (OpenID Connect)
Open standard that third-party IdPs use so other companies or sites can authenticate users without maintaining an in-house user database.

JSON Web Token (JWT)
Open standard used to securely transmit information between two parties.

BAD PRACTICE	BEST PRACTICE
how credentials get compromised ▪ Embed access keys in unencrypted code ▪ Share access keys between users in an AWS account	*temporary credentials, always* ▪ Use **IAM roles** to retrieve temporary security credentials ▪ If IAM roles aren't an option, use **AWS STS** to retrieve credentials

The **AWS Security Token Service (AWS STS)** provides trusted users with **temporary security credentials** to access your AWS resources. Each set consists of a short-lived **access key ID, secret access key, and session token**. As the name implies, the credentials have a **limited, configurable lifetime** — from a few minutes to several hours — and after they expire AWS no longer recognizes them or allows access from requests made with them. They are **not stored with the user**; they are **generated dynamically** and provided on request, and a user can request new credentials (even before the old ones expire) as long as they still have permission to do so.

WHO STS ISSUES CREDENTIALS TO – TRUSTED USERS

- **IAM users** — for **cross-account access** (an IAM user in one account assumes a role for temporary access in another), and for applications running on **EC2 instances and other AWS compute services**.
- **Federated identities (external users)** — **enterprise identity federation** for employees authenticated through a SAML 2.0 enterprise IdP such as Active Directory or LDAP (single sign-on), and **web identity federation** for mobile/web users authenticated through a third-party IdP such as Login with Amazon, Facebook, Google, or any OIDC-compatible IdP (social sign-in).
- For federated identities you **don't create new AWS identities** — users reach AWS resources with their corporate credentials (SSO) or a third party (social sign-in).

HOW STS AUTHENTICATION WORKS FOR FEDERATED USERS

(1) A user accesses an application backed by AWS. (2) The application calls an **identity broker**, which accepts the user's identifier. (3) *First authentication* — the broker authenticates the user's identity against an **IdP** (Active Directory for enterprise, or a third-party IdP for web). (4) *Second authentication* — on success, the broker makes an API call to **AWS STS**, including an IAM policy and a duration that specify the permissions for the temporary credentials. (5) STS uses **IAM** to confirm the calling IAM user's policy permits creating new tokens. (6) STS returns **four values** to the broker: an access key, a secret access key, a session token, and a duration. (7) The broker returns the temporary credentials and token to the application. (8) The application uses them to make requests to an AWS service such as **Amazon S3**. (9) The service uses IAM to confirm the credentials allow the requested operation. *For IAM users the flow is identical, except the first authentication simply checks whether the caller is an IAM user — no external IdP is involved.*

TABLE 12.2 AWS STS API operations

OPERATION	RETURNS TEMPORARY CREDENTIALS TO...
AssumeRole	Existing IAM users for cross-account access to AWS resources
AssumeRoleWithSAML	Federated users authenticated by an organization's existing identity system (must use SAML 2.0)
AssumeRoleWithWebIdentity	Federated users authenticated through a public IdP (Login with Amazon, Facebook, Google, or any OIDC IdP); for mobile/web apps whose users have no AWS identity — though AWS recommends Amazon Cognito instead of calling this directly
GetFederationToken	Federated users; differs from AssumeRole in that the default expiration is much longer — 12 hours instead of 1 hour
GetSessionToken	Existing IAM users for enhanced security — e.g., allowing AWS requests only when MFA is enabled for the IAM user

Because federated users have no IAM identity of their own, you track their activity with **AWS CloudTrail**. CloudTrail captures actions from authenticated users, **stores its log files in Amazon S3**, and can be configured to respond with actions or notifications. To capture federated activity it records the `AssumeRoleWithSAML` and `AssumeRoleWithWebIdentity` STS API calls.

TRACKING A FEDERATED USER WITH CLOUDTRAIL

Suppose a SAML-federated employee, Bob, signs in through SSO and terminates an EC2 instance, and administrator Alice needs to find out who did it. Alice first searches the CloudTrail logs for the `eventName` **`TerminateInstances`** and, in the event's `userIdentity` section, determines the **ARN of the IAM role** the federated user assumed. She then searches for the `eventName` **`AssumeRoleWithSAML`** that includes that role ARN, and finally identifies Bob in the **`userName`** attribute of the `userIdentity` section. The ability to trace federated users this way supports auditing, compliance, and security.

COMMON PITFALL

For mobile and client-based web apps whose users have no AWS identity, don't reach for `AssumeRoleWithWebIdentity` directly. AWS recommends using **Amazon Cognito** and the Amazon Cognito credentials provider with the AWS SDKs instead — which is exactly where the next section goes. *Section takeaways:* issue temporary credentials as a best practice; AWS STS serves both IAM users and external (federated) users; use CloudTrail to track federated users' actions.

12.4 Authenticating with Amazon Cognito

Amazon Cognito provides **authentication, authorization, and user management** for your web and mobile applications. With it you can create unique identities for your users and authenticate them through **external IdPs that support SAML or OIDC**, **social IdPs** (such as Facebook, Twitter, and Amazon), and your own **custom IdPs** — and you can provide **temporary security credentials** for your application to access AWS resources and services. Cognito has two main components: user pools and identity pools.

USER POOLS	IDENTITY POOLS (FEDERATED IDENTITIES)
authentication — who the user is ▪ User directories that provide **sign-up and sign-in** for your application ▪ Users sign in directly through Cognito or federate through a third-party IdP ▪ Accept federated identities from Login with Amazon, Google, Facebook, or any SAML/OIDC IdP ▪ Every member has a **directory profile** accessible through an SDK	*authorization — reach AWS resources* ▪ Provide users with **temporary security credentials** to access other AWS services ▪ Enable creation of unique identities and federate them with various providers (for example, an OIDC or Amazon identity) ▪ You can have multiple identities for different IdPs ▪ Can be used **separately or together** with user pools

USER POOL SECURITY FEATURES AND FUNCTIONALITY

- **Sign-up and sign-in services**, including social sign-in (Facebook, Google, Login with Amazon) and SAML IdP sign-in.
- A **built-in, customizable login web UI** to sign in users, manage the directory, and access user profiles.
- **Control who can access your API** — API Gateway validates tokens from a successful user pool authentication and grants access to resources such as Lambda functions; user-pool **groups** can map to IAM roles, and group membership is carried in the ID token.
- **Compromised-credentials check** — prevents users from reusing a username/password pair exposed elsewhere; AWS is informed when a set of credentials is compromised.
- **Phone and email verification**.
- **Adaptive authentication** — block suspicious sign-ins or add second-factor authentication in response to an increased risk level.

To authenticate against a user pool, a user **answers successive challenges** until authentication either fails or the user is issued tokens; because those steps repeat, you can support any custom authentication flow. You customize the flow with **Lambda triggers** that issue and verify their own challenges — password verification, MFA, CAPTCHA, or secret questions — and you can use a Lambda trigger for **post-authentication tasks** such as logging events for custom analytics. After a successful sign-in, your app receives user pool tokens; you use them to retrieve AWS credentials or to control access to server-side resources and API Gateway, and you can create **user pool groups** to manage permissions and represent different types of users.

TABLE 12.3 User pool tokens – three JWTs

TOKEN	PURPOSE	PROTECTION
ID token	Authorizes API calls; contains claims about the identity of the user, such as name and email	Signed
Access token	Authorizes API calls based on the custom (OAuth) scopes of specified access-protected resources	Signed
Refresh token	Contains the information necessary to obtain a new ID or access token	Encrypted

When you sign in through a user pool — whether the identity lives in the pool or comes from a federated third-party IdP — Cognito gives you these **three JWTs**. **ID and access tokens are signed, not encrypted; the refresh token is encrypted.** The developer's job is to take the ID and access tokens and pass them along to authorize access to application resources; the **client saves the refresh token** to silently refresh the ID and access tokens behind the scenes. Tokens are short-lived by design: you can configure **access tokens to expire in as little as 5 minutes or as long as 24 hours**, and **refresh tokens from 1 hour to 10 years**.

USER POOL AUTHENTICATION WITH AN API GATEWAY REST API

The client first authenticates with the user pool and gets the three JWTs. It then passes the **ID and access token in the header** of its call to **API Gateway**, which **validates the token before invoking** the backend resource it integrates with — in this example, a **Lambda function**. Depending on how the API is written, API Gateway can forward the **ID token** (when you don't need to further scope access) or the **access token** (when you do — configure predefined attributes in the method request using **OAuth scopes**).

Identity pools address the other half of the problem: turning an authenticated identity into AWS access. Using **web identity federation** in a mobile app, users sign in with a supported IdP (Login with Amazon, Facebook, Google) and trade that provider's authentication token for **temporary AWS security credentials** — so you can build the app **without writing backend code** to integrate with the IdPs and **without embedding long-term AWS credentials**. With an identity pool you create unique identities and obtain **temporary, limited-privilege AWS credentials through AWS STS**. The supported IdPs are Cognito user pools, public IdPs (Login with Facebook, Google, or Amazon), SAML IdPs, OIDC IdPs, and **developer-authenticated identities** (users you register and authenticate through your own backend). In this view, a Cognito **user pool is just one more IdP**, alongside Facebook, Google, and others.

GRANTING IAM PERMISSIONS WITH AN IDENTITY POOL

A mobile app **(1) authenticates** with a Cognito user pool and **(2) gets the JWTs**. It **(3) requests AWS credentials** from the Cognito identity pool (federated identities), which **(4) validates the ID token** and **(5) provides temporary credentials**. The app **(6) signs the payload** with those credentials and calls a service API — for example **Amazon DynamoDB** — which **(7) checks the IAM policy** attached to the temporary credentials and **(8) returns the requested data**, allowing only the actions the policy permits. This is federated identities plus IAM-based authorization delivering temporary IAM credentials.

COMMON PITFALL

Keep the two pools straight. A **user pool** authenticates users and hands back JWTs (ID and access signed, refresh encrypted) — it answers *who are you?* An **identity pool** exchanges an identity for temporary AWS credentials via STS — it answers *what AWS resources may you use?* Many apps use both: sign in with a user pool, then get AWS credentials from an identity pool. *Section takeaways:* use Cognito to manage user access to AWS resources; user pools are directories of users; you associate users with credentials in identity pools; authenticate with user pools or IdP credentials.

12.5 The lab and applying it: securing the café website

Lab 12.1: Implementing Application Authentication Using Amazon Cognito puts the section into practice. Sofía uses Amazon Cognito to integrate an authentication mechanism into the café website so that **Frank can log in to confirm his identity before requesting an inventory report**, and she connects the REST API endpoint to the site so Frank can make the report request directly from the website.

LAB TASKS

- Prepare the development environment
- Configure a Cognito **user pool and app client**
- Configure the client app, and integrate the Amazon Cognito **hosted URI** into the website
- Observe the REST API endpoint details and test
- Create a **user** for the Cognito user pool
- Configure an **API Gateway authorizer**
- Test the request process from the website

HOW THE FINISHED FLOW FITS TOGETHER

Frank accesses the CloudFront-served café website and **logs in through Amazon Cognito**, which returns an **authentication token**. He chooses **Get Report**, which **passes the token with the request** to the **API Gateway** endpoint. **Step Functions** coordinates **Lambda functions** to generate the report, pulling data from the supplier database on **Amazon RDS**; the report (report.html) is placed in an **S3 bucket** and made available via a **presigned URL**; an **SNS** topic emails Frank, who then accesses the report on S3. Every earlier concept appears — TLS on the CloudFront connection, a Cognito user pool for sign-in, JWTs passed to an API Gateway authorizer, and IAM-scoped access to backend resources.

SAMPLE EXAM QUESTION, WORKED

A developer is adding sign-up and sign-in functionality to an application, which must make an API call to a custom analytics solution to log user sign-in events. Which combination of actions satisfies the requirements? (Select TWO.) The key phrases are **sign-up and sign-in**, **custom analytics solution**, and **sign-in events**. The answers are **A — use Amazon Cognito to provide sign-up and sign-in** and **E — invoke a Lambda function to make the API call that the post-authentication event initiates**. Cognito delivers user sign-up, sign-in, and access control quickly, and a **post-authentication Lambda trigger** is the built-in way to run custom code (the analytics API call) after each sign-in. **B** is wrong — IAM controls access to AWS resources for AWS principals, *not* application end-user sign-in; **C** (API Gateway) isn't how a post-authentication event fires custom code; and **D** (Secrets Manager credential rotation) addresses secret storage, not logging sign-in events.

Module summary

- Application security is the developer's responsibility *in* the cloud, at every layer, and it has two aspects: securing network connections between clients and servers, and controlling user access to the AWS resources the application calls.
- SSL/TLS encrypt traffic and use a certificate from a trusted CA to establish website identity; ELB terminates TLS at the load balancer and CloudFront negotiates HTTPS (an OAI can restrict an S3 bucket to CloudFront). AWS Certificate Manager (ACM) provisions, deploys, and auto-renews public and private certificates for ELB, CloudFront, and API Gateway — free for AWS services — and ACM Private CA issues your own private certificates.
- The best practice for access is temporary credentials — never embedded or shared long-lived keys; use IAM roles, or AWS STS when roles aren't an option. STS issues short-lived credentials (access key ID, secret access key, session token) to trusted users — IAM users (cross-account, EC2/compute) and federated identities (enterprise SSO via SAML, web social sign-in via OIDC) — through an identity broker; its operations are AssumeRole, AssumeRoleWithSAML, AssumeRoleWithWebIdentity, GetFederationToken, and GetSessionToken.
- Authentication verifies who a user is; authorization verifies what they may do; IdPs manage identities, identity brokers get STS credentials, and SAML/OIDC/JWT are the open standards. Track federated user actions with CloudTrail (records AssumeRoleWithSAML and AssumeRoleWithWebIdentity).
- Amazon Cognito provides authentication, authorization, and user management: user pools are user directories that sign users in and return three JWTs — ID and access tokens (signed) and a refresh token (encrypted); identity pools (federated identities) trade an authenticated identity for temporary, limited-privilege AWS credentials via an STS role, treating a user pool as just one supported IdP among public, SAML, OIDC, and developer-authenticated identities. The two can be used separately or together.
- The café lab wires it together: Frank signs in with Cognito, passes a token to an API Gateway authorizer, and Step Functions/Lambda generate an inventory report delivered from S3 via a presigned URL and SNS.

Review questions

1. What does the shared responsibility model make the developer responsible for, and what two application-security aspects does this module cover?

Answer: The developer is responsible for security *in* the cloud, at every layer of the application. The two aspects are securing the network connections between clients and application servers, and controlling user access to the AWS resources the application calls.

2. Walk through how TLS uses a certificate to secure a connection, and name the service that manages those certificates on AWS.

Answer: The CA issues a certificate to the domain; the client requests identification, the server sends its certificate and public key, the client verifies the issuing CA is trusted, the client sends an encrypted session key, the server acknowledges with that key, and all data is then encrypted with the session key. AWS Certificate Manager (ACM) provisions, deploys, and renews the certificates.

3. Why are temporary security credentials preferred, and what does a set of AWS STS credentials contain?

Answer: Embedding or sharing long-lived access keys is a bad practice; temporary credentials are short-lived, dynamically generated, and stop working after they expire. An STS set contains an access key ID, a secret access key, and a session token, with a configurable lifetime.

4. Distinguish authentication, authorization, an identity provider, and an identity broker.

Answer: Authentication verifies the user's identity (who); authorization verifies their permissions (what they can do); an identity provider (IdP) manages identities and provides authentication; an identity broker authenticates credentials against an IdP and retrieves temporary credentials from AWS STS.

5. Which STS API operations serve SAML-federated users, web-identity (social) federated users, and MFA-protected IAM users?

Answer: AssumeRoleWithSAML for SAML-federated users, AssumeRoleWithWebIdentity for public/OIDC (social) federated users, and GetSessionToken for existing IAM users where AWS requests are allowed only with MFA enabled.

6. Contrast an Amazon Cognito user pool with an identity pool, and describe the three tokens a user pool returns.

Answer: A user pool is a user directory that provides sign-up/sign-in (authentication) and returns three JWTs — an ID token and an access token (both signed) and a refresh token (encrypted). An identity pool (federated identities) provides temporary, limited-privilege AWS credentials via an STS role (authorization). They can be used separately or together.

7. How do you audit the actions of federated users who have no IAM identity?

Answer: Use AWS CloudTrail, which stores logs in Amazon S3 and records the AssumeRoleWithSAML and AssumeRoleWithWebIdentity STS calls; you can trace an action to the assumed role's ARN and identify the federated user in the userName attribute of the event's userIdentity section.