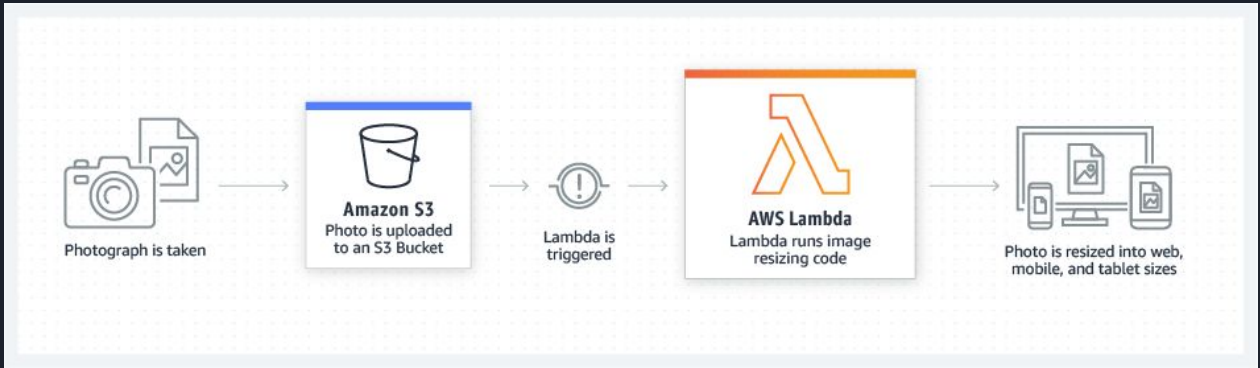
Ejemplo Básico: Build-In AWS UI

Lambdas - Definición

- AWS Lambda es un servicio informático sin servidor (serverless).
- 1 millón de solicitudes gratis con la cuenta gratuita de AWS.

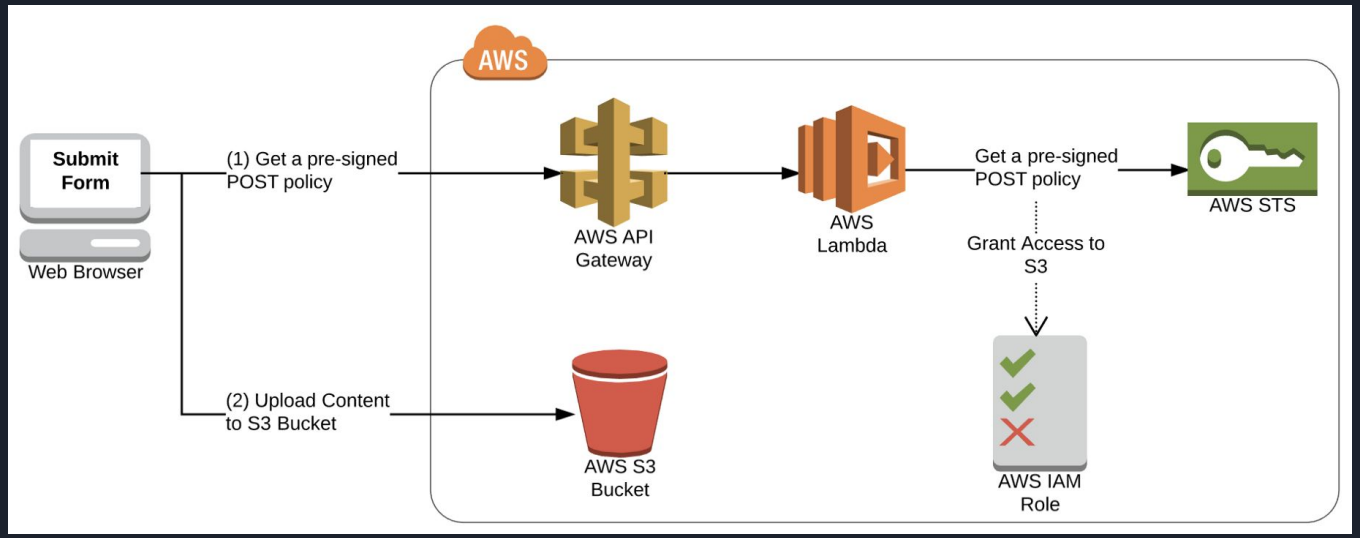


Lambdas - Caracteristicas

- No hay que administrar servidores.
- Escalado continuo.
- Coste optimizado.
- Rendimiento uniforme.

S3 - Definición

Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento.



S3 - Caracteristicas

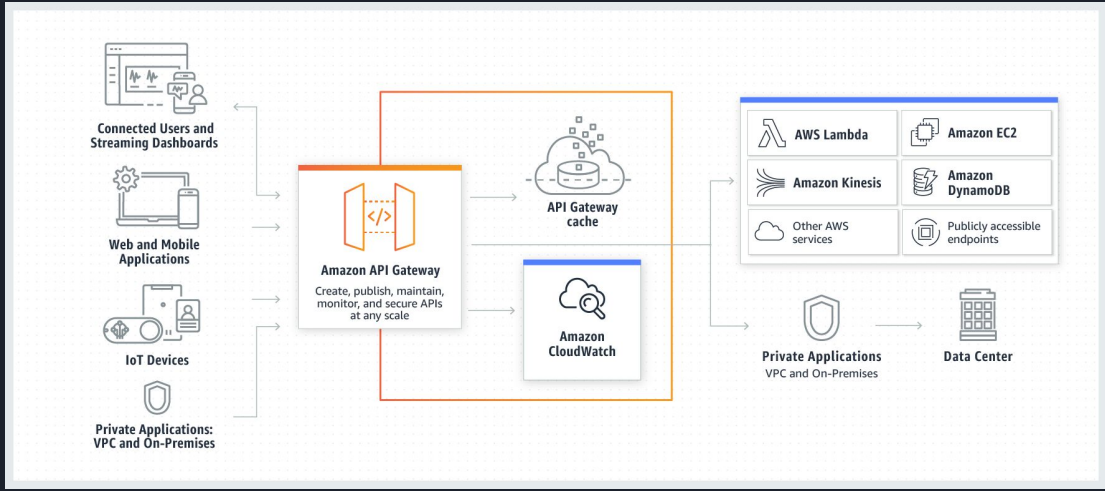
- Rendimiento.
- Escalabilidad.
- Disponibilidad.
- Durabilidad.
- Variedad de almacenamientos.
- Seguridad.

S3 - Ejemplos de utilización

- Copia de seguridad y restauración.
- Recuperación de desastres (DR).
- Archivo.
- Lagos de datos y análisis de big data.
- Almacenamiento en nube híbrida.
- Aplicaciones nativas en la nube.
- Seguridad.

API Gateway - Definición

Amazon API Gateway es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala

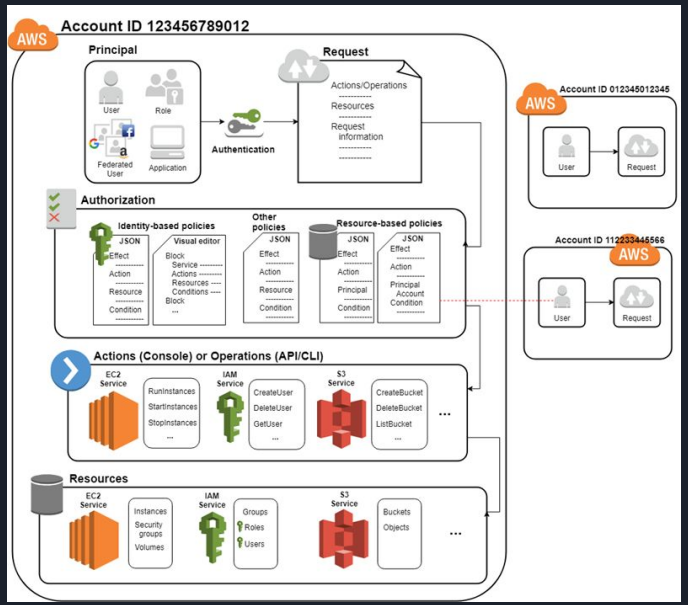


API Gateway - Caracteristicas

- Desarrollo de API eficiente.
- Rendimiento a cualquier escala.
- Ahorro de costos.
- Monitoreo fácil.
- Controles de seguridad flexibles.

IAM - Definición

Con AWS Identity and Access Management (IAM) puede administrar el acceso a los servicios y recursos de AWS de manera segura.



IAM - Ejemplos de utilización

- Control de acceso granular a los recursos de AWS.
- Autenticación multifactor para usuarios con privilegios.
- Analizar el acceso.
- Integración en su directorio corporativo.

DEMO

https://github.com/ErnestOrt/jwt-demo

Servicios Autenticación
y
Servicios Autorización

Autenticación + Autorización

Se pueden resumir ambos conceptos de la siguiente manera:

- **Autenticación**: verifica las identidades mediante diferentes métodos: algo que sabemos (contraseña), algo que tenemos (tarjeta crédito, dni electrónico), o algo que identifica quiénes somos (huellas dactilares, identificación iris, etc.).
- **Autorización**: verifica los permisos que corresponden a cada identidad, y le permite acceder a determinados recursos del sistema.

Servicios Autenticación - Definición

- Mecanismo que identifica a un usuario o un servicio.
- Consta básicamente de dos procesos, el proceso de identificación y el proceso propiamente dicho de autenticación que verifica la identidad del usuario.
- El ejemplo típico más común es la autenticación mediante usuario/contraseña.
- Para añadir más seguridad en los sistemas usuario/contraseña se pueden añadir autenticación multifactor (MFA), proceso mediante el cual se realiza una verificación en dos pasos. Por ejemplo sería primero mediante usuario/contraseña y posteriormente se añade una autenticación mediante un SMS al teléfono móvil del usuario. Se puede hacer también la verificación mediante correo electrónico.

Servicios Autenticación - Métodos

- Mediante usuario/contraseña .
- Basados en estándares, como OpenID Connect.
- Mediante redes sociales (proveedores de identidad).
- Proveedores de identidad basados en el estándar abierto SAML 2.0
- Autenticación API (mediante JWT).
- Otros (autenticación biométrica, patrón ocular, verificación de voz, etc.).

Servicios Autorización - Definición

La autorización es lo que **define a qué recursos del sistema el usuario autorizado podrá acceder**.

Normalmente definiremos a qué recursos específicos queremos conceder acceso a los usuarios.

Por ejemplo a un usuario le podríamos conceder permisos sólo para acceder a S3 (por ejemplo subir objetos), también para acceder a una instancia de EC2, o para acceder a nuestra API. En general, habrá diferentes tipos de usuarios con diferentes tipos de permisos que determinarán a qué recursos del sistema pueden acceder.

Servicios Autorización - Métodos

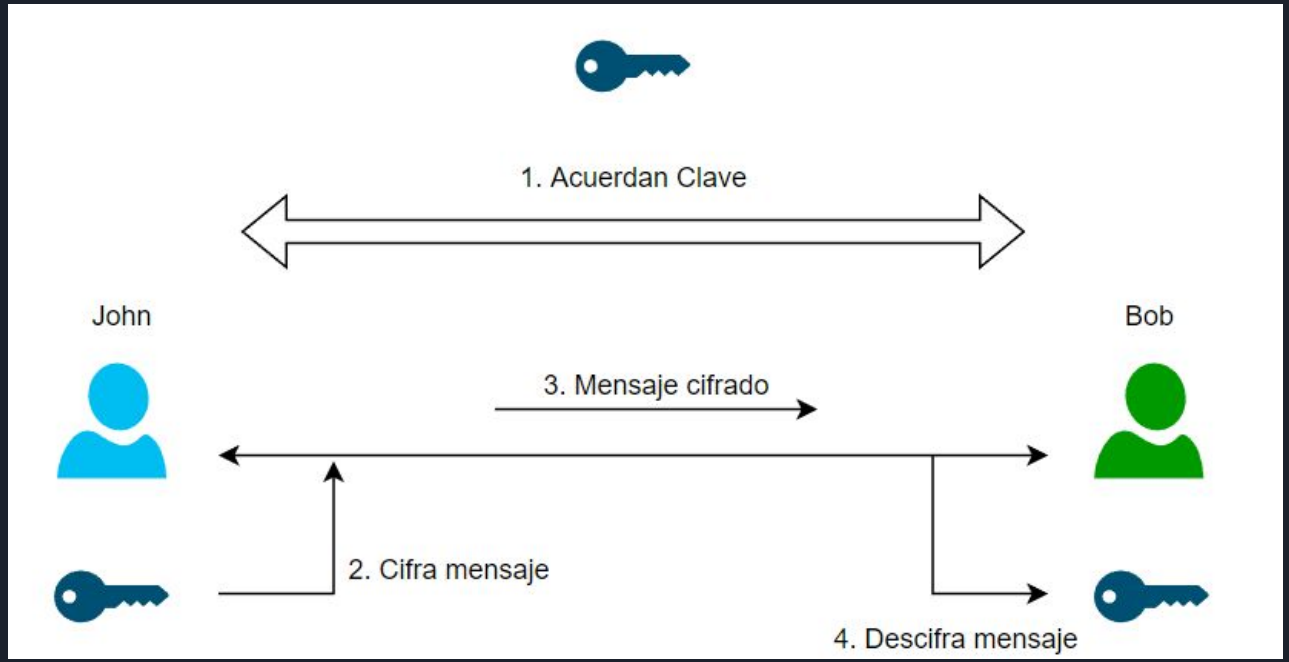
- Autorización HTTP (usuario/contraseña codificados se envían en el encabezado de una petición HTTP).
- Autorización API (se genera una clave API que actúa como un token) .
- OAuth 2.0 (framework, estándar abierto de autorización). Basado en diferentes flujos de autorización (para aplicaciones de escritorio/móviles). OpenID Connect lo usa en un nivel inferior.
- Autorización JWT (mediante Json Web Tokens).

Criptografía y encriptación

Tipos de Encriptación: Simétrica

- El cifrado simétrico se basa en una sola clave tanto para hacer el cifrado como para el descifrado.
- Garantiza confidencialidad e integridad, pero no autenticidad. Es menos segura que la criptografía asimétrica, ya que se transmite la clave por el canal de comunicación entre emisor/receptor.
- Es mucho más rápido que usar criptografía asimétrica. Se usa en sistemas donde se requiere un rendimiento más alto. Por ejemplo, para cifrar grandes cantidades de datos.

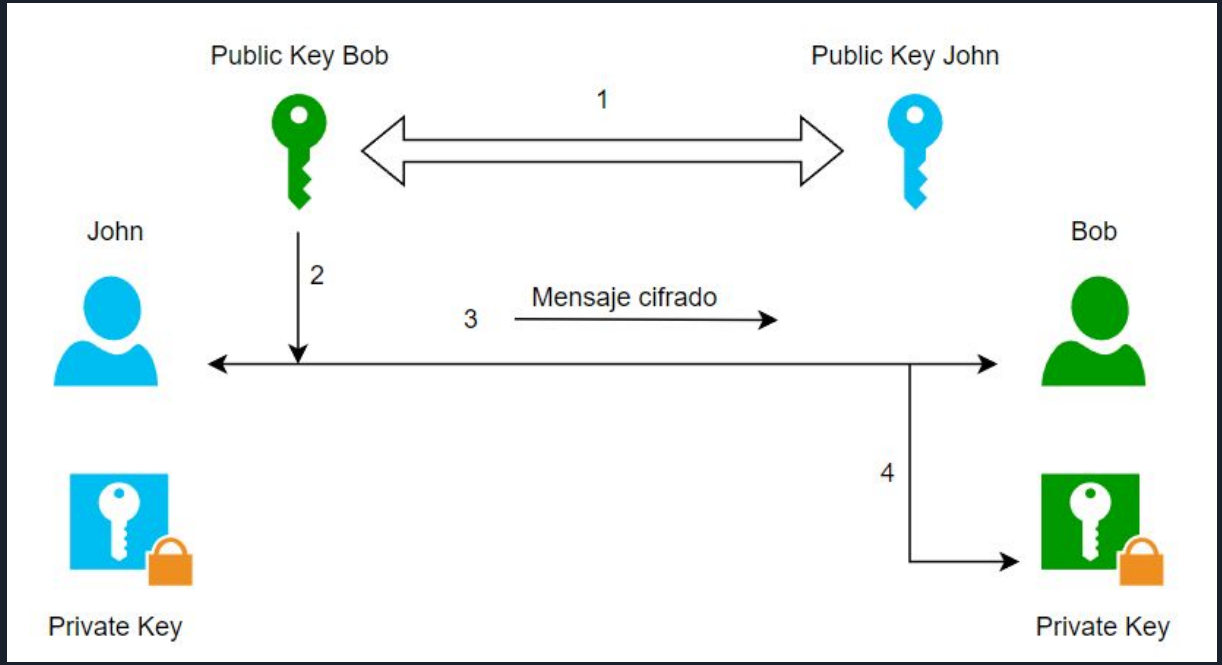
Tipos de Encriptación: Simétrica



Tipos de Encriptación: Asimétrica

- Trabaja con una codificación de información basada en dos claves: una privada y una pública.
- Proporciona confidencialidad, integridad y autenticidad (mediante los certificados y firmas digitales).
- Clave Pública: la clave pública puede encriptar mensajes que sólo se descifran con la clave privada.
- Clave Privada: usada para descifrar la información que ha sido cifrada con la clave pública. También se usa en sistemas de autenticación, ya que se usa la clave privada para firmar y la pública para validar la firma del autor del mensaje.

Tipos de Encriptación: Asimétrica



Simétrica vs Asimétrica

CARACTERÍSTICAS	SIMÉTRICA	ASIMÉTRICA
Número de claves	Irá incrementando el número de claves a medida que vayan aumentando los usuarios.	Solamente se requiere un par de claves por usuario para que cada uno pueda cifrar mensajes para todos los demás.
Velocidad	Permite cifrar gran cantidad de datos y ahorrar tiempo simultáneamente ya que es muy rápido y ágil.	Es mucho más lento y, debido a eso, no es la mejor opción si se requiere de un rendimiento óptimo .
Seguridad	Es poco seguro porque el hecho de comunicar la clave para descifrar aporta vulnerabilidad.	Cuenta con la libertad de emitir claves públicas, mientras que la privada permanece con el usuario.

Protocolos seguros SSL/TLS

- El protocolo SSL (Secure Socket Layer) es la versión anterior a TLS (Transport Layer Security). Son protocolos de comunicación segura que usan algoritmos criptográficos.
- Son la base del protocolo HTTPS (HyperText Transfer Protocol Secure) o protocolo seguro de transferencia de hipertexto utilizado en Internet.
- Utiliza criptografía asimétrica para para intercambiar una clave simétrica y poder cifrar la información que se transmite.
- Utiliza certificados digitales X.509 para la autenticación. Primero se intercambian las claves públicas entre emisor y receptor. Después se transmite una clave simétrica y se procede a enviar información cifrada con esa clave.

¿Qué son los JWT?

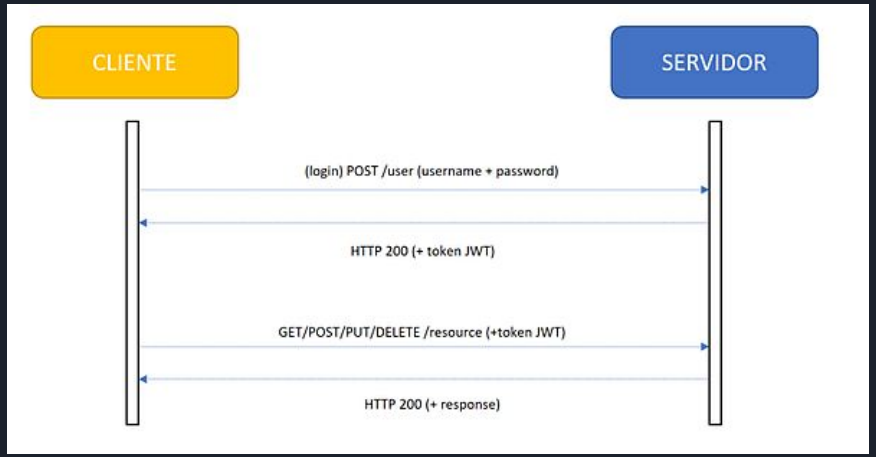
Definición de JWT

- Estándar abierto RFC 7519 basado en JSON para crear tokens.
- Permite enviar información entre dos partes de forma segura. Su objetivo principal es proporcionar autenticación y/o también autorización.
- Se suele firmar usando un algoritmo de codificación simétrica o asimétrica, por ejemplo el HS256 (cifrado simétrico) o el RS256 (cifrado asimétrico). De esta manera la información queda firmada por medio de la clave de cifrado.
- Los tokens firmados permiten verificar la integridad de la información que contienen.
- La firma también certifica que sólo la parte que posee la clave privada es la que la firmó (caso de criptografía asimétrica).

Casos de uso y precauciones de los JWT

- Para autorizar acceso a recursos, normalmente en entornos web.
- Intercambio seguro de información, vigilando siempre usar conexiones seguras https.
- Necesitamos mecanismos de identificación de usuarios o aplicaciones.
- Facilita el no tener que almacenar información sobre sesión de usuario (es *stateless*).
- Precauciones: si se guardan los tokens en el local storage del navegador hay que tener cuidado con los ataques de XSS (Cross Site Scripting). También podemos ser víctimas de un ataque CSRF (Cross Site Request Forgery) o Falsificación de Petición de Sitios Cruzados.

Diagrama secuencia ejemplo de uso de un JWT



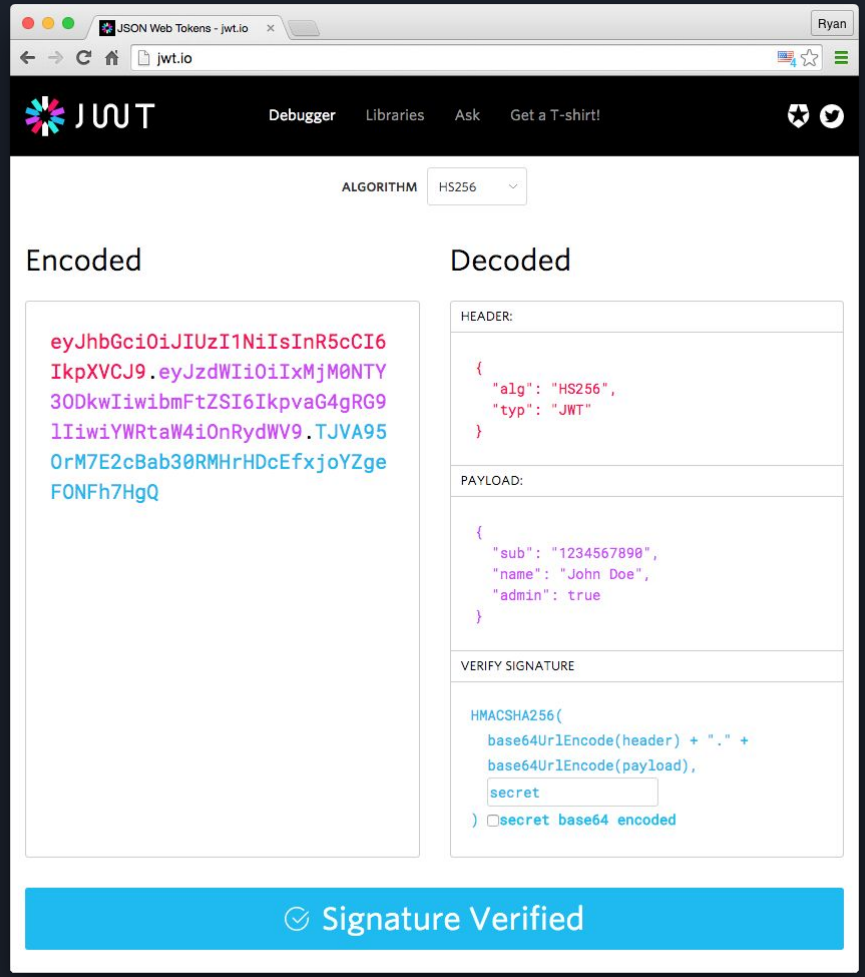
Estructura JWT

Consta de 3 partes:

- **Header**: incluye la información relacionada con el tipo de token y el algoritmo de firma.
- **Payload**: contiene propiedades o afirmaciones (claims), algunas de ellas son estándar, como “iss” (creador del JWT), “aud” (quién recibirá el JWT), “exp” (tiempo de expiración del token, “jti” (identificador único del token), etc. Podemos añadir información personalizable, pero hemos de saber que el payload no estará firmado, por lo que es recomendable no incluir información muy sensible.
- **Firma**: el header y el payload se codifican en base64Url y se cifran con la clave. El resultado de este cifrado es la firma del token, que también se codifica en base64Url.

Estructura JWT

- Header
- Payload
- Signature



Tipos de JWT

- **ID token:** este tipo de token es usado para autenticar al usuario en un sistema, como por ejemplo AWS Cognito. Es un token de autenticación.

- **Access token**: contiene toda la información que el servidor necesita para saber si el usuario o dispositivo puede acceder al recurso. Es un token de autorización.

- **Refresh token**: se utiliza para generar nuevos tokens de acceso y autorización. Se usa en casos en los que los tokens expiren su tiempo de utilización.

Estructura de un JWT de tipo ID token

Descripción campos más relevantes del Payload (llamados claims o permisos):

- sub (subject): Id único del usuario autenticado (no es el user name, lo genera Cognito).
- aud (audience): app client_id usado en la autenticación de usuario.
- cognito:groups: lista de grupos a los que el usuario pertenece.
- token_use: indica cómo se va a usar el token (id token o access token).
- auth_time: indica el tiempo en formato UTC (segundos) de cuándo se autenticó el usuario.
- iss (issuer/emisor): URL con información de zona de creación del user pool y el user pool ID.
- cognito:username : nombre de usuario en user pool.
- exp: tiempo en UTC (Unix time en segundos) en que expira el token.
- iat: tiempo en UTC (Unix time en segundos) en que fué emitido el token.
- jti: identificador único del token.

Estructura de un JWT de tipo access token

Descripción campos más relevantes del Payload (llamados claims o permisos):

- sub (subject): Id único del usuario autenticado (no es el user name, lo genera Cognito).
- cognito:groups : lista de grupos a los que el usuario pertenece.
- token_use: indica cómo se va a usar el token (id token o access token).
- scope: scopes o permisos que otorga el token (profile, email, phone).
- auth_time: indica el tiempo en formato UTC (segundos) de cuándo se autenticó el usuario.
- iss (issuer/emisor): URL con información de la zona de creación del user pool y el user pool ID.
- exp: tiempo en UTC (Unix time en segundos) en que expira el token.
- iat: tiempo en UTC (Unix time en segundos) en que fué emitido el token.
- jti: identificador único del token.
- client_id: identificador de la aplicación (app client identifier Cognito).
- username: user name del usuario.

JWK y JWKS

- **JSON Web Key (JWK)**: Un objeto JSON que representa una llave criptográfica. Las propiedades del objeto representan propiedades de la llave incluyendo su valor.
  - kid: es un identificador único de la llave.
  - alg: es el algoritmo de la llave.
  - kty: es el tipo de llave.
  - e: exponente RSA (codificado en Base64urlUInt).
  - n: módulo RSA (codificado en Base64urlUInt).
  - use: es como la llave será usada.

- **JSON Web key Set (JWKS)**: Un objeto JSON que representa un conjunto de JWKs. El objeto JSON debe tener una propiedad llamada keys que es la lista de JWKs.

JWK y JWKS

Estructura de un archivo jwks.json, es accesible por medio de un endpoint de AWS Cognito y necesitamos pasarle un par de datos que identifican al usuario.


```
{
        "keys": [{
                "kid": "1234example=",
                "alg": "RS256",
                "kty": "RSA",
                "e": "AQAB",
                "n": "1234567890",
                "use": "sig"
        }, {
                "kid": "5678example=",
                "alg": "RS256",
                "kty": "RSA",
                "e": "AQAB",
                "n": "987654321",
                "use": "sig"
        }]
}
```

OAuth2.0

Definición

- **OAuth 2.0 (Open Authorization 2.0)**: es un framework que permite crear protocolos de autorización. Este framework es el que usaremos con AWS Cognito, y también ha pasado a ser un estándar en la industria (RFC-6749).

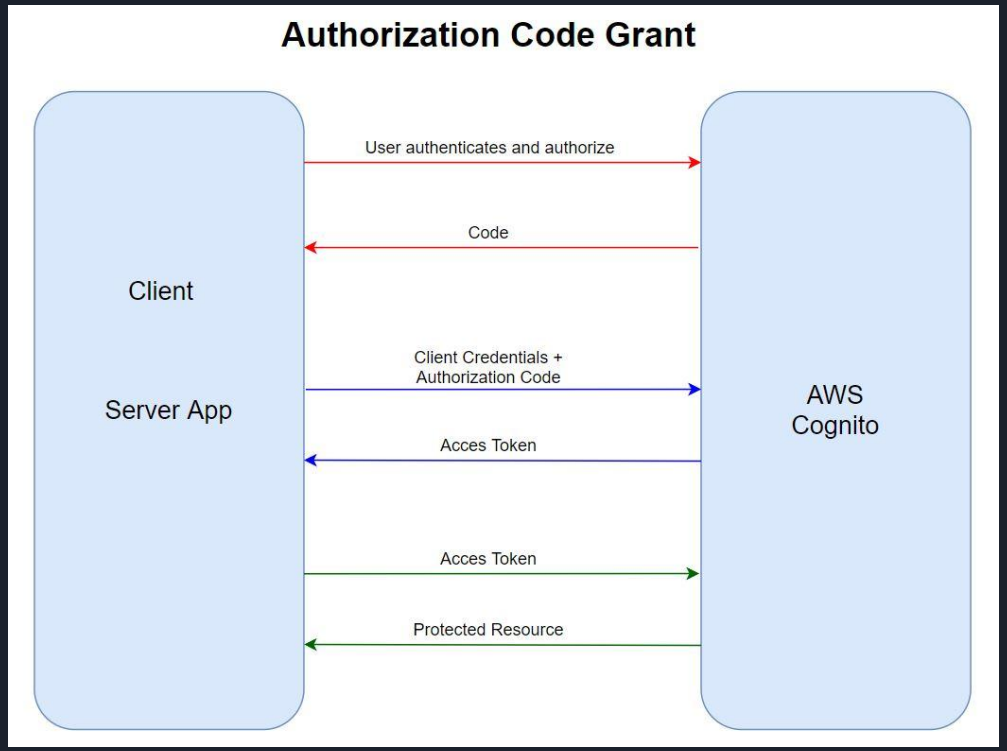
  Al ser un framework de autorización, nos permitirá acceder a ciertos recursos de AWS (o a nuestros propios recursos). Está basado en tokens, de manera que la aplicación cliente, después de autenticarse el usuario, recibirá un token de acceso con unos permisos, para poder acceder a recursos protegidos para los cuáles se le quiere conceder acceso a los usuarios o las aplicaciones cliente.

Flujos de Autorización - OAuth 2.0

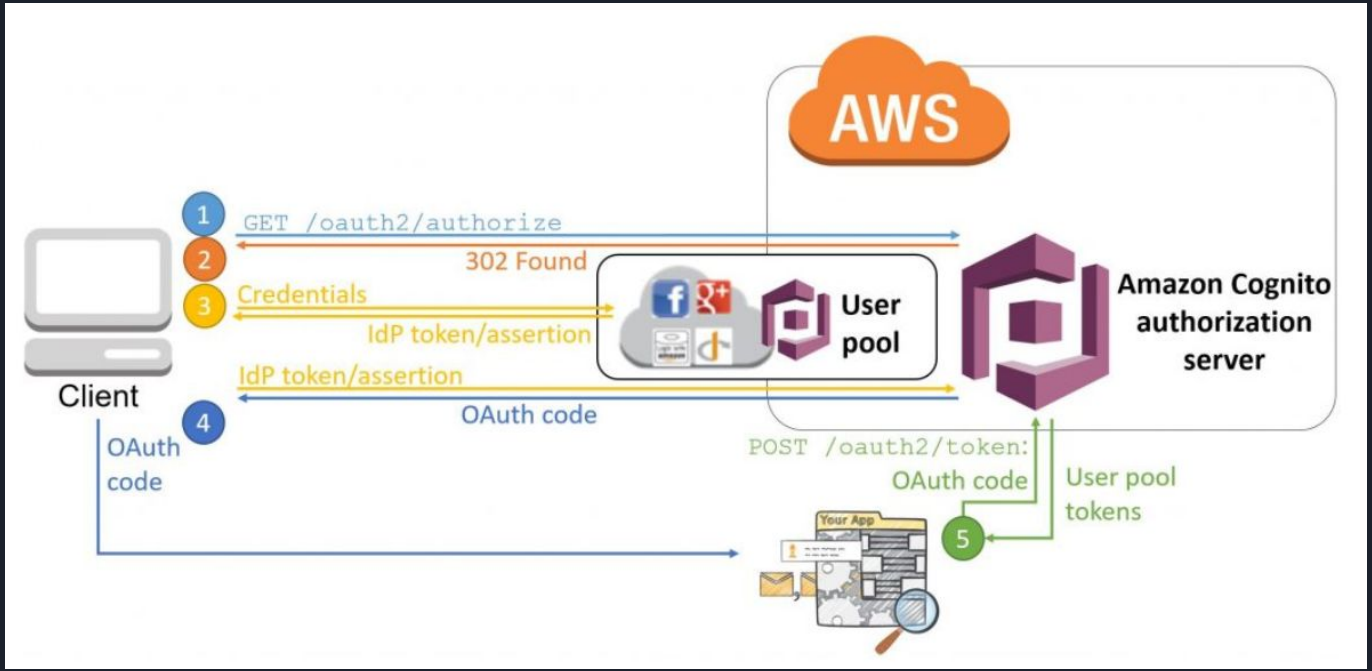
OAuth 2.0 permite 3 flujos de identificación que usaremos con AWS Cognito. Podremos elegir cuál nos interesa más, estos 3 flujos son:

- **Authorization Code Grant**: generalmente se usa cuándo la autorización la gestiona nuestra aplicación en el servidor. Es considerado uno de los flujos más seguros.
- **Implicit Grant**: se usa normalmente en las SPA (Single Page Application), ya que en este caso, el cliente es nuestra aplicación y podrá gestionar el access token que le suministrará el servicio de autorización.
- **Client Credentials Grant**: se usa en casos donde se le puede facilitar al cliente o aplicación cliente, unas credenciales de acceso (un clientId y un clientSecret). Muy usado en API's y CLI (Command Line Interface), como la CLI de AWS.

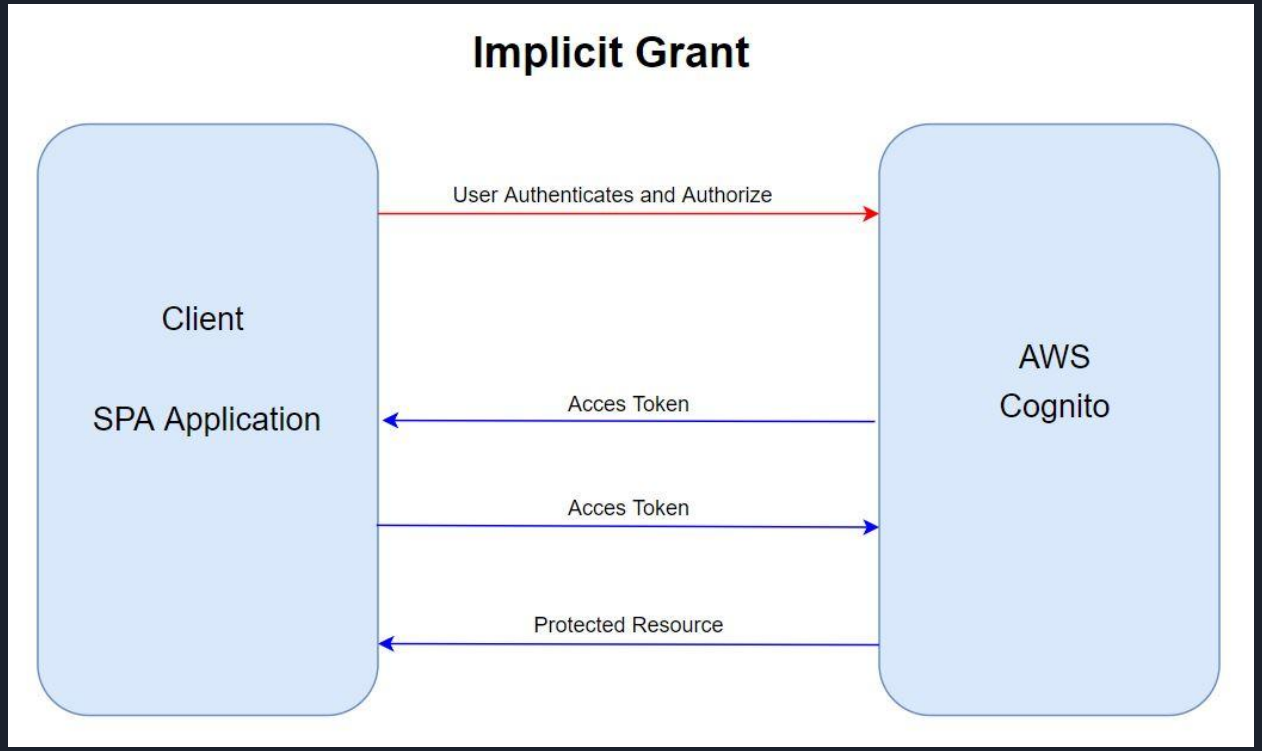
Authorization Code Grant



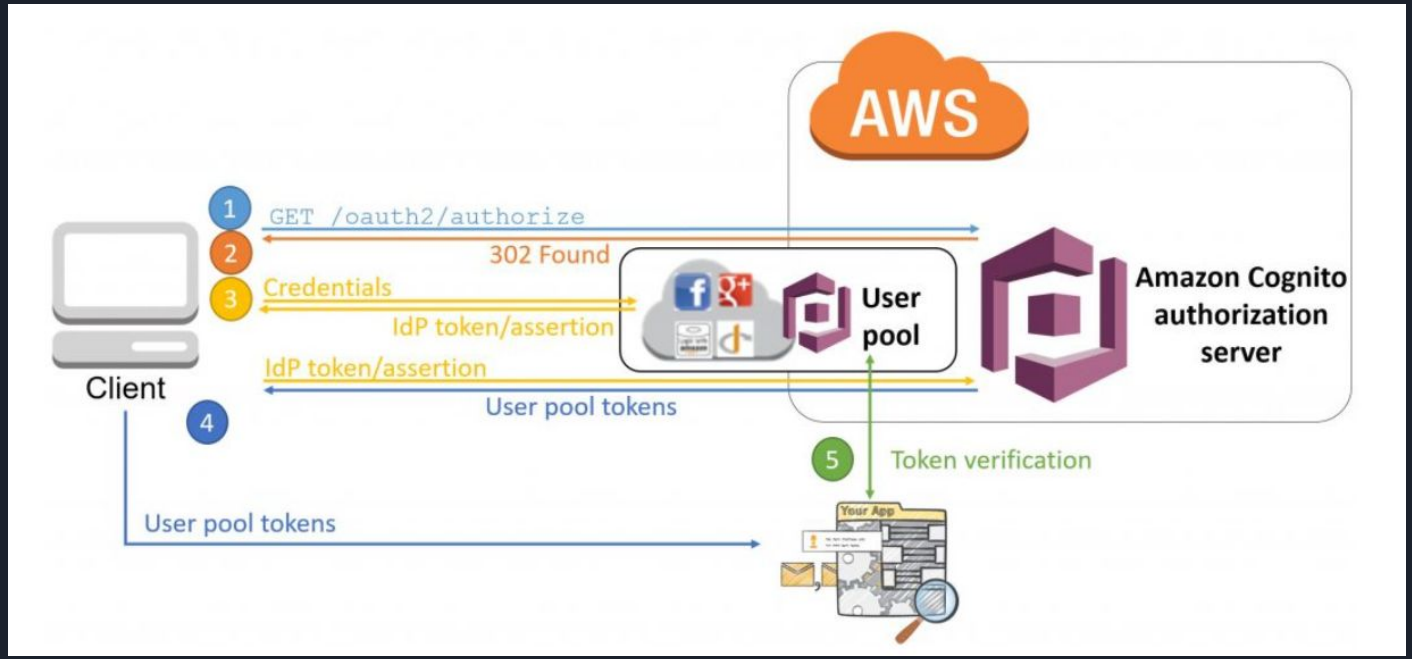
Authorization Code Grant



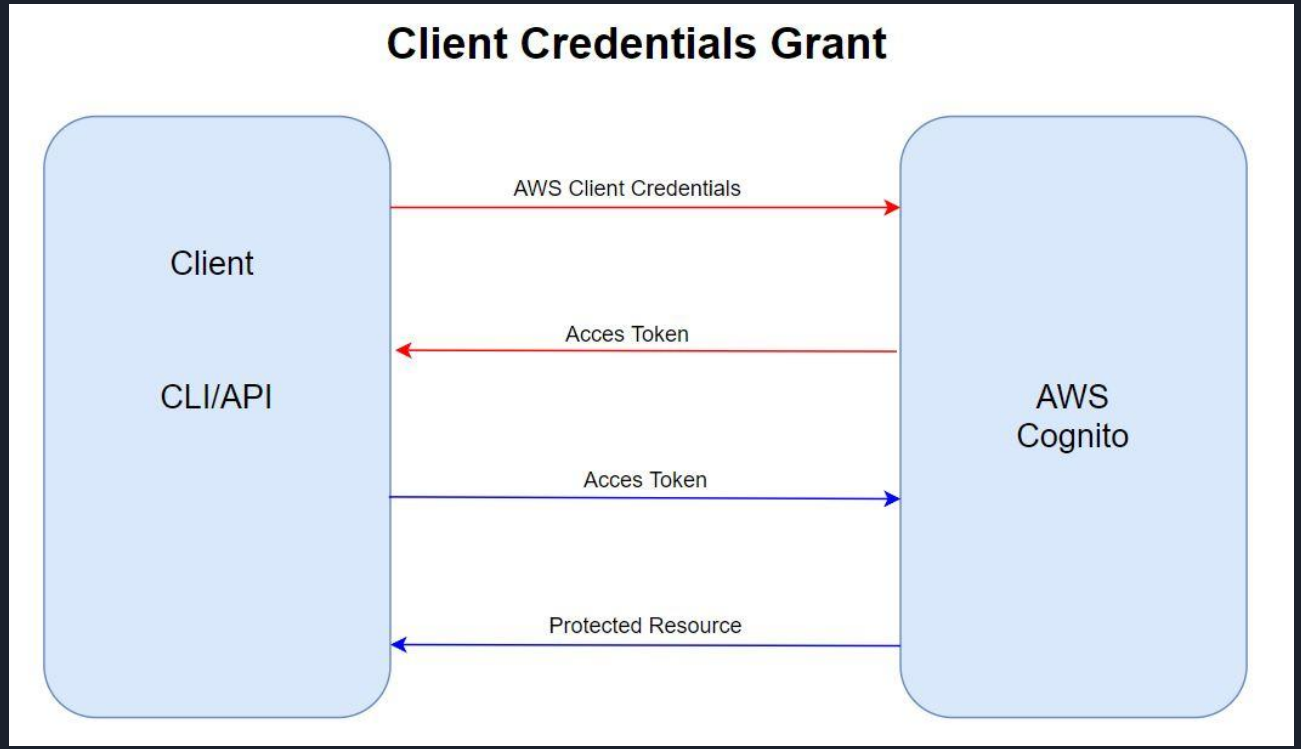
Implicit Grant



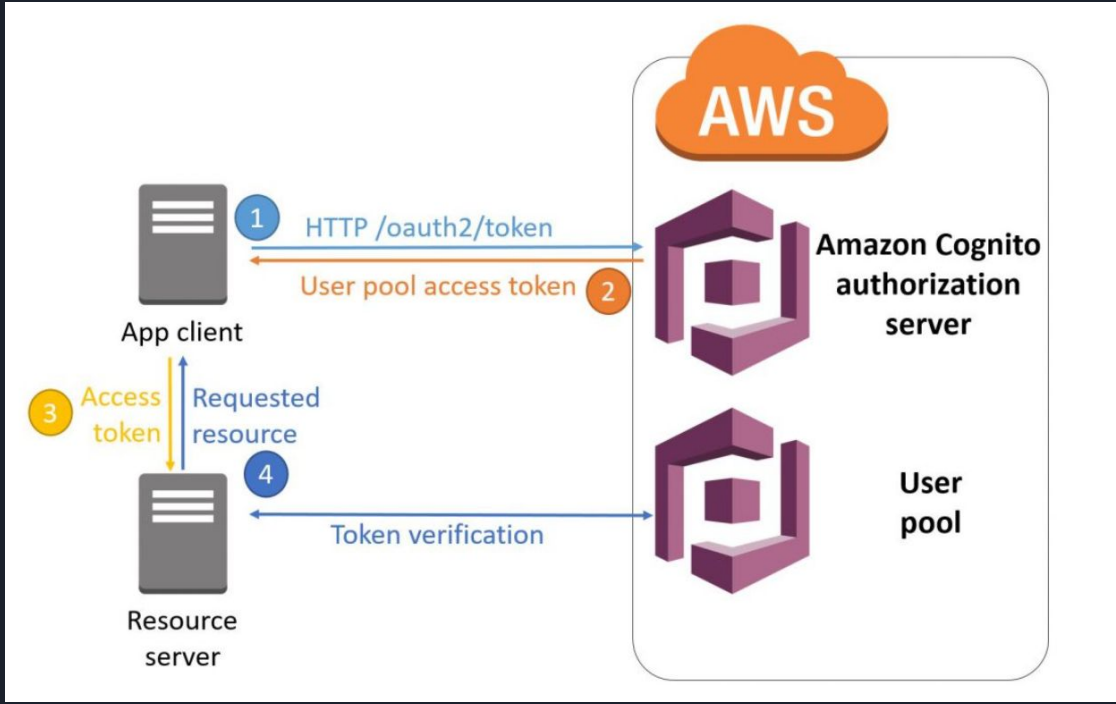
Implicit Grant



Client Credentials Grant



Client Credentials Grant



Authorization Code Grant PKCE

- Existen otros flujos con algunas características que permiten su integración con dispositivos móviles (SDK Android/iOS), web (SDK Javascript) , o Java (SDK Java).
- Es el caso del Authorization Code Grant, el cuál es vulnerable cuando se envía el authorization code. Para solucionar este problema, se usa el **Authorization Code Grant PKCE (Proof Key for Code Exchange) o clave de prueba para el intercambio de códigos** (RFC 7636), que nos facilita mecanismos para protegernos contra la intercepción del authorization code.
- El code_verifier es generado de forma aleatoria por un algoritmo criptográfico, por lo que se considera un valor aleatorio criptográficamente fuerte.
- El code_challenge se genera haciendo un hash del code_verifier.

OpenID Connect

Definición

- **OpenID Connect (OIDC):** es un protocolo y estándar de autenticación que funciona sobre OAuth2.0, de manera que obtenemos un servicio de autenticación + autorización al mismo tiempo. Es el caso de AWS Cognito, que usa este estándar.
- Permite implementar algunos de los flujos básicos de autenticación, por ejemplo: Authorization Code Grant, Authorization Code Grant con PKCE e Implicit Grant (no permite el Client Credentials Grant).
- Trabaja con tokens de acceso (access token) y de identificación (Id token). El token de identificación incluye información básica y “no sensible” de los usuarios, y el token de acceso incluye también información relativa a los permisos.

Scope y claims permitidos en OIDC

- Dentro del access token que genera Cognito, hay un parámetro llamado “scope” o alcance que permite incluir ciertos claims o permisos. Los claims permitidos en OIDC son: profile (podemos personalizar atributos dentro de profile), email, address y phone, que nos permiten definir los privilegios de acceso requeridos para los tokens de acceso (access tokens). Cognito no usa el claim address en el scope.
- Hay un enlace en el recurso adjunto dónde podemos consultar la especificación sobre los claims permitidos en OIDC.

Estructura de un JWT de tipo ID token

Descripción campos más relevantes del Payload (llamados claims o permisos):

- sub (subject): Id único del usuario autenticado (no es el user name, lo genera Cognito).
- aud (audience): app client_id usado en la autenticación de usuario.
- cognito:groups: lista de grupos a los que el usuario pertenece.
- token_use: indica cómo se va a usar el token (id token o access token).
- auth_time: indica el tiempo en formato UTC (segundos) de cuándo se autenticó el usuario.
- iss (issuer/emisor): URL con información de zona de creación del user pool y el user pool ID.
- cognito:username : nombre de usuario en user pool.
- exp: tiempo en UTC (Unix time en segundos) en que expira el token.
- iat: tiempo en UTC (Unix time en segundos) en que fué emitido el token.
- jti: identificador único del token.
- email: email que hayamos configurado.
- phone_number: teléfono configurado.

Definición

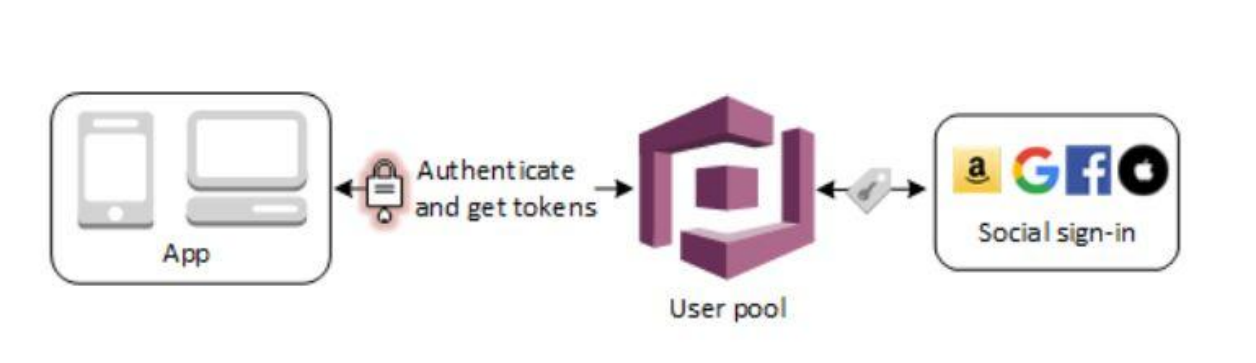
- OIDC nos proporciona una capa de autenticación por encima de OAuth 2.0 y nos permite trabajar con dos tokens, uno de identificación (Id token) y otro de acceso (Access token).

Proveedores de identidad OIDC

Proveedor de identidad OIDC	Instalación
Salesforce	Instalar un proveedor de identidad Salesforce
Ping Identity	Instalar un proveedor de identidad Ping Identity
Okta	Instalar un proveedor de identidad Okta
Microsoft Azure Active Directory (Azure AD)	Instalar un proveedor de identidad Microsoft Azure AD
Google	Instalar un proveedor de identidad Google

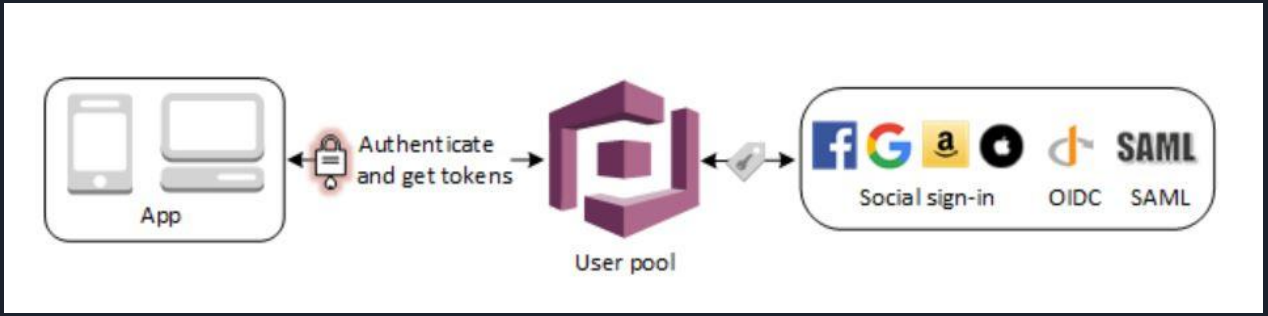
Social login con OAuth 2.0

- Imagen que muestra cómo obtener autenticación mediante User Pool en AWS Cognito con proveedores de identidad social compatibles con OAuth 2.0



Amazon Cognito

- Cognito nos ofrece una solución conjunta:



Más información sobre OIDC

- Enlaces a la web oficial de OpenID Connect:

  https://openid.net/

  https://openid.net/developers/jwt/

Amazon Cognito

Amazon Cognito. Características:

- Es un servicio de AWS que nos proporciona servicios de autenticación, autorización, administración de usuarios, e integración con nuestras aplicaciones.
- Permite la identificación a través de identidades federadas (Facebook, Amazon, Google y Apple) y otros proveedores de identidad (Microsoft Active Directory mediante SAML), así como autenticación multifactor (MFA), entre sus características más importantes.
- Cumple con los requisitos de seguridad y conformidad de las normas:
  HIPAA, PCI DSS, SOC, ISO/IEC 27001, ISO/IEC 27017, ISO/IEC 27018 e ISO 9001.

Amazon Cognito. Características:

- A través de la definición de roles podemos dar acceso a ciertos recursos de AWS a nuestros usuarios, como por ejemplo, acceso a S3, bases de datos, triggers Lambda, o acceso a instancias EC2 (por ejemplo acceso a una página web), etc.
- Admite los siguientes estándares de administración de acceso e identidades: OAuth 2.0, SAML 2.0 y OpenID Connect.
- Permite escalar a cientos de millones de usuarios.
- En la capa gratuita podemos trabajar hasta con 50.000 usuarios activos al mes.

Amazon Cognito. Características:

- Permite personalizar la interface de usuario de nuestras aplicaciones a través de herramientas como los SDK (Software Development Kit) de Android, iOS, Javascript, React, Angular, etc.
- Incorpora opciones de seguridad avanzadas para proteger el acceso de los usuarios a nuestras aplicaciones. Por ejemplo: intentos de inicio se sesión desde ubicaciones y dispositivos nuevos (autenticación adaptativa), o detección de credenciales vulnerables. De manera que podemos configurar el servicio para solicitar verificaciones adicionales (SMS, correo, etc.) o incluso bloquear el acceso.
- Permite generar informes de nuestro sistema, usuarios, accesos, roles, etc.

Amazon Cognito. Configuración:

User Pools:

- Permite autenticación de usuarios a través de nuestras aplicaciones, mediante usuario/contraseña u otros proveedores basados en OpenID Connect.
- Posibilidad de adaptar la interface de usuario e integrarla en nuestras aplicaciones. También nos provee de una UI por defecto para gestionar el acceso.
- Nos proporciona la gestión de usuarios, aplicaciones (id de aplicación) y grupos de usuarios.
- Nos permite especificar cómo queremos que se autentiquen los usuarios (email, username, phone number u otros).
- Podremos configurar los flujos de OAuth 2.0, los scopes permitidos y los proveedores de identidad social (Facebook, Google, Cognito, etc.).

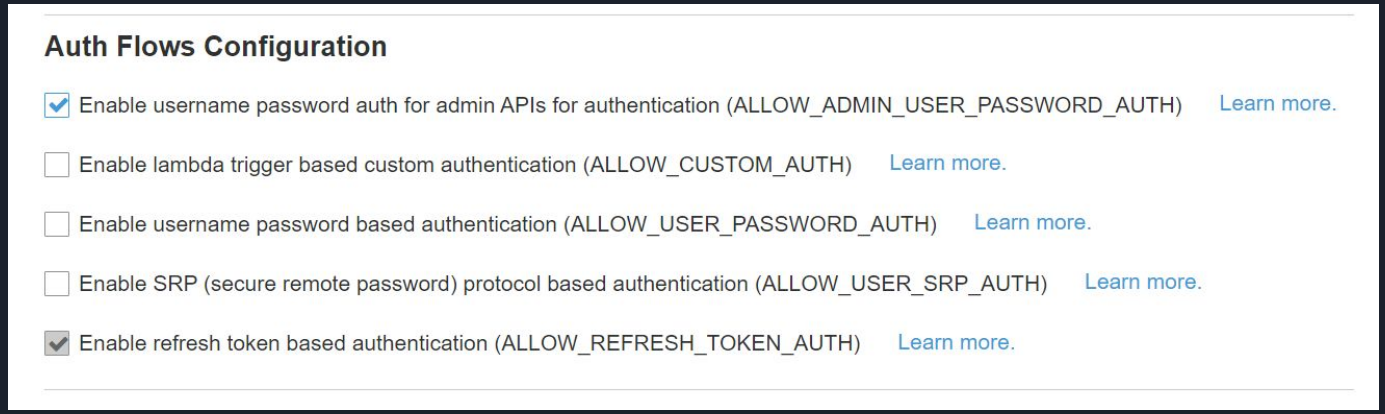
Amazon Cognito. Flujos OAuth 2.0

OAuth 2.0 permite 3 flujos de identificación que usaremos con AWS Cognito. Podremos elegir cuál nos interesa más, estos 3 flujos son:

- **Authorization Code Grant**: generalmente se usa cuándo la autorización la gestiona nuestra aplicación en el servidor. Es considerado uno de los flujos más seguros.
- **Implicit Grant**: se usa normalmente en las SPA (Single Page Application), ya que en este caso, el cliente es nuestra aplicación y podrá gestionar el access token que le suministrará el servicio de autorización.
- **Client Credentials Grant**: se usa en casos donde se le puede facilitar al cliente o aplicación cliente, unas credenciales de acceso (un clientId y un clientSecret). Muy usado en API's y CLI (Command Line Interface), como la CLI de AWS.

Amazon Cognito. Configuración User Pools:

En la configuración de User Pools, deberemos prestar especial atención a los diferentes tipos de flujos que podemos usar. Esta opción se encuentra en el apartado de “App clients” y se muestra así:



Amazon Cognito. Configuración User Pools:

A continuación explicamos cuándo debemos usar las diferentes configuraciones:

- ALLOW_ADMIN_USER_PASSWORD_AUTH: Esta opción se usa cuándo nuestra aplicación es una Server-Side App, ya que será nuestro backend el que interactúe con Cognito en la autenticación.
- ALLOW_CUSTOM_AUTH: Permite personalizar el flujo de autenticación y usar triggers lambda.
- ALLOW_USER_PASSWORD_AUTH: Se usa en aplicaciones Client-Side. El cliente se identifica con sus credenciales. Se puede usar integrando el protocolo SRP o no. En la siguiente slide explicaremos cómo funciona este protocolo.
- ALLOW_USER_SRP_AUTH: Usaremos esta opción si queremos usar el protocolo SRP (Secure Remote Password) en nuestras aplicaciones. Normalmente esta integración la proporciona el propio SDK de Cognito (de Android, IOs, Javascript, etc.). Se suele usar en aplicaciones cliente (Client-Side).

Amazon Cognito. Protocolo SRP

Características Secure Remote Password (SRP):

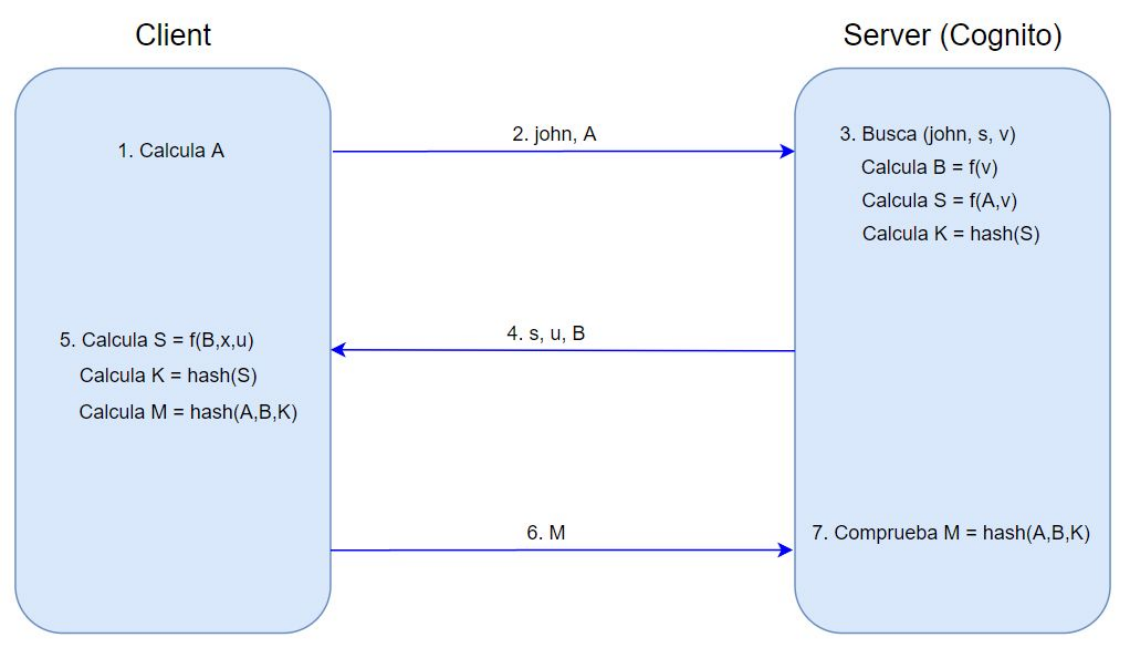
- Es un protocolo considerado muy seguro, ya que en ningún momento se transmite la contraseña del usuario en el intercambio de mensajes entre el cliente y el servidor.
- Utiliza un mecanismo similar a la criptografía asimétrica (llaves públicas/privadas).
- El servidor de autenticación (en nuestro caso Cognito), no guardará en ningún momento la contraseña del usuario, gracias a esta característica, en ningún momento la base de datos de usuarios de Cognito, quedará expuesta de forma comprometida. Por este motivo, no será vulnerable a ataques de diccionario o de fuerza bruta, ya que aunque un ciberdelincuente acceda a la base de datos, no podrá obtener nunca la contraseña.

Amazon Cognito. Protocolo SRP

Más características:

- Este protocolo utiliza un intercambio de mensajes y retos entre la aplicación cliente y el servicio de autenticación de Cognito. Esta funcionalidad la gestiona el propio SDK de Cognito en sus versiones Javascript o de aplicaciones móviles, por lo que se realiza de manera transparente para el usuario, que sólo tendrá que usar las librerías correspondientes.

Amazon Cognito. Protocolo SRP :



Amazon Cognito. Identity Pools:

Identity Pools o grupos de identidades:

- Creación de los grupos de identidades o Identity Pools.
- Nos permite obtener credenciales temporales de acceso a los servicios de AWS, tanto de usuarios autenticados, como no autenticados (opción configurable).
- Permite incluir a los usuarios creados mediante User Pools, así cómo especificar sus roles (y permisos) para acceder a los diferentes servicios de AWS, mediante la definición de roles con IAM (Identity and Access Management). Podemos configurar diferentes roles o permisos para distintos grupos de usuarios definidos mediante UserPool.

Amazon Cognito. Identity Pools, compatibilidad:

- Compatible con proveedores de identidad públicos conocidos (Amazon, Facebook, Apple, etc). Veremos ejemplos más adelante en el curso sobre cómo configurarlos.
- Compatible con Cognito User Pools.
- Proveedores basados en OpenID Connect.
- Proveedores de identidad SAML 2.0.
- Soporta también identidades autenticadas de desarrollador, así podremos utilizar nuestros propios procesos de registro y autenticación a través de Cognito.

Amazon Cognito. Identity Pools. Integración con otros servicios:

- Utilizando Amazon Cognito Streams podremos enviar datos en tiempo real (cuando estos se actualicen o sincronicen) al servicio de Kinesis. Este servicio podrá transmitir estos datos y por ejemplo guardarlos en una base de datos como DynamoDB, Redshift, o incluso en S3, etc.
- Con Amazon Cognito Events podremos ejecutar funciones Lambda cuando se sincronice un conjunto de datos o Data Set.
- Posibilidad de sincronizar datos de usuarios en diferentes dispositivos mediante el servicio de Amazon AppSync.

Amazon Cognito. Identity Pools, roles IAM:

Mediante la definición de roles en Identity Pool, podremos asociar políticas de permisos (Permissions Policies) que determinan a qué servicios de AWS puede acceder ese rol.

También podremos definir las políticas de confianza (Trust Policies), mediante las cuáles podemos también establecer condiciones adicionales a las políticas de permisos.

Para definir o modificar las políticas asociadas a los roles, deberemos usar el servicio de IAM, procurando siempre garantizar el mínimo privilegio que otorga acceso a los recursos de AWS.

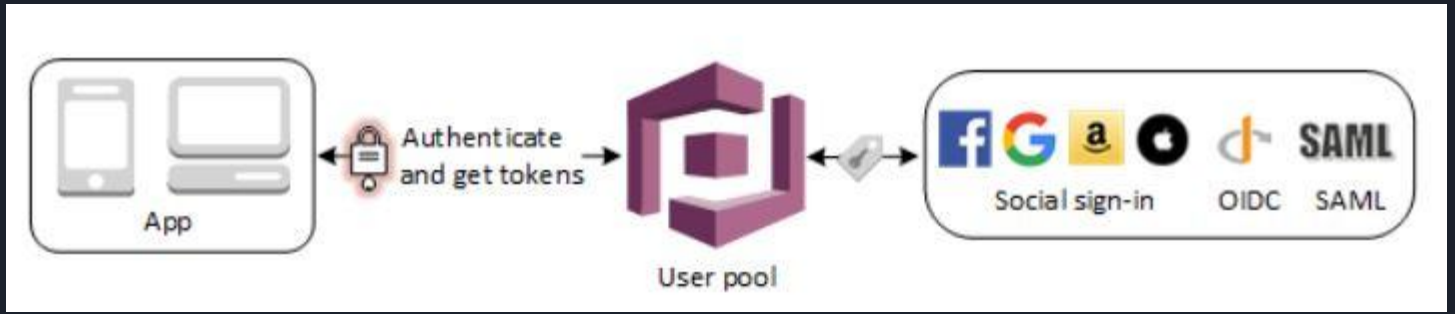
Vamos a mostrar unos ejemplos de definición de las políticas de permisos y de confianza, así como permisos basados en relaciones de confianza y permisos de rol.

Amazon Cognito. Identity Pools, usando atributos para el control de acceso:

- Podremos acceder a los recursos mediante Identity Pools basado en atributos de usuario. La gestión de permisos mediante estos atributos facilita el control de acceso.
- Las políticas de permisos sólo permitirán el acceso a los usuarios que hagan match con los atributos de usuario.
- Podremos definir políticas basadas en estos atributos para que actúen en función de sus valores. Por ejemplo podemos definir una política basada en el departamento al que pertenece el usuario, si pertenece a “marketing” sólo tendrá acceso de lectura a información guardada en S3. Por el contrario si el usuario pertenece a “administrador” podrá escribir/ leer/borrar esa información u objetos guardados en S3.

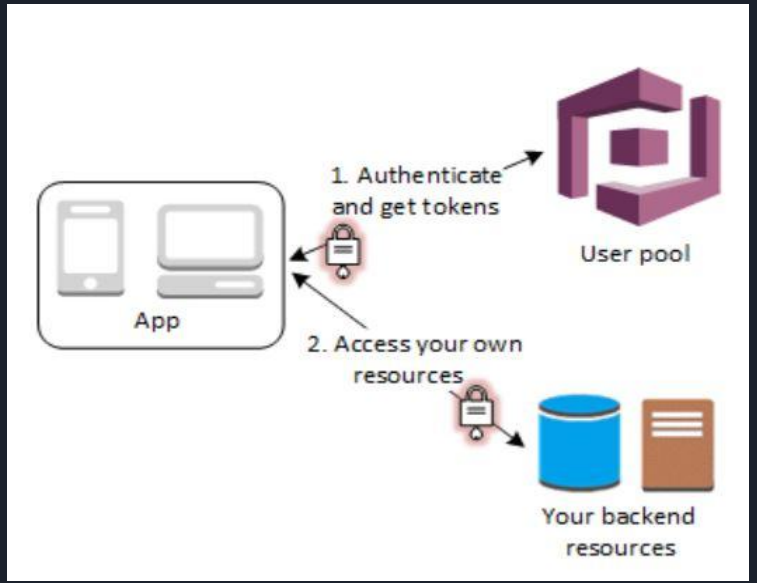
Amazon Cognito. Escenarios:

Autenticación con User Pool:



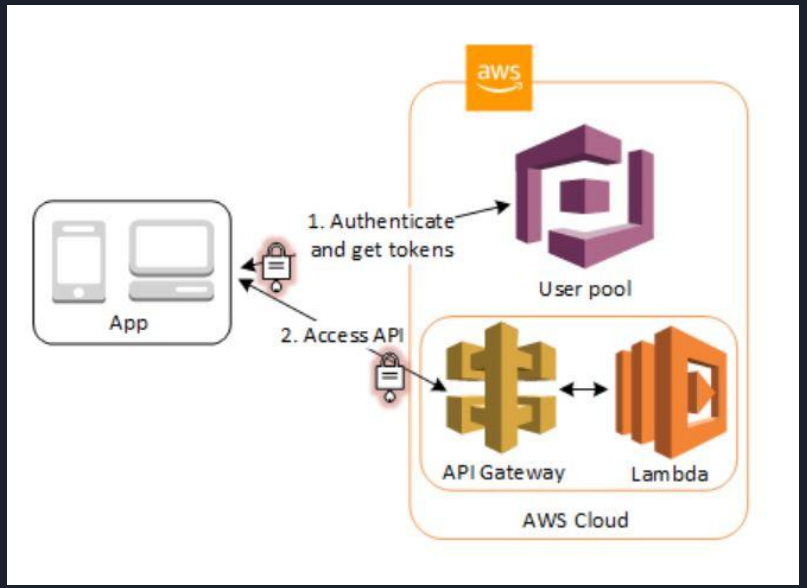
Amazon Cognito. Escenarios:

Acceso a servicios con User Pool:



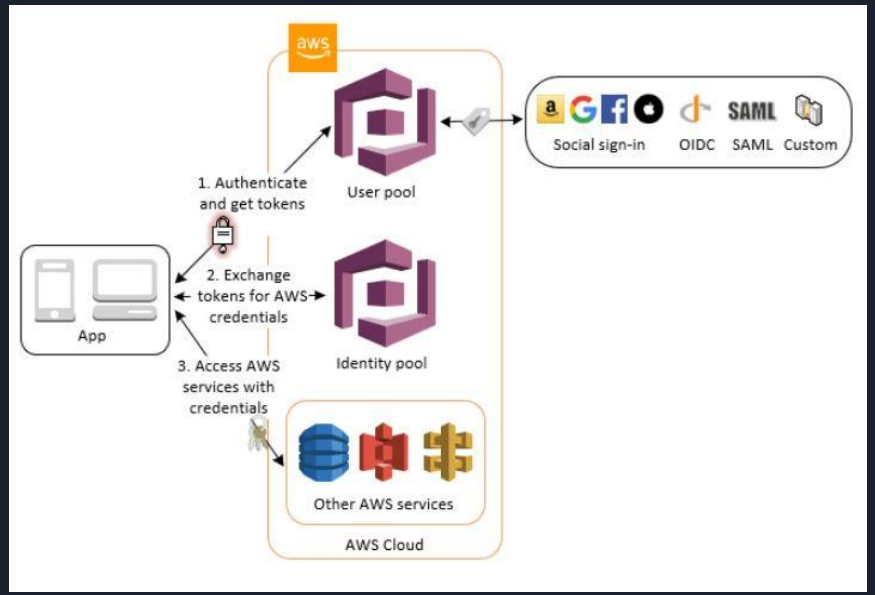
Amazon Cognito. Escenarios:

Acceso a servicios con
API Gateway y Lambda:



Amazon Cognito. Escenarios:

Acceso a servicios con User Pool e Identity Pool:



Tokens que usa Amazon Cognito

- Utiliza criptografía asimétrica, donde la clave privada se usa para firmar el token (en la parte del servidor) y la clave pública para verificar la firma (en la parte cliente).
- Usa objetos JSON para representar llaves, son los llamados JWK (JSON Web Key). También usa JWKS (JSON Web Key Set) que son conjuntos de JWK. Normalmente contiene las claves públicas que permiten verificar la firma del token.
- Más adelante veremos que estos procesos los podemos realizar de forma automatizada, ya que Cognito nos permite usar librerías en diferentes lenguajes de programación, para facilitar todas las comprobaciones necesarias.

Estructura de un JWT de tipo ID token

- Header:

```
{
"kid" : "1234example="
"alg" : "RS256",
}
```

- kid: nos indica cuál de las dos keys fué usada para la firma del JWT.
- alg: indica el algoritmo criptográfico usado para firmar el JWT.

Estructura de un JWT de tipo ID token

- Payload:


```
{
"sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
"aud": "xxxxxxxxxxxxexample",
"email_verified": true,
"token_use": "id",
"auth_time": 1500009400,
"iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_example",
"cognito:username": "janedoe",
"exp": 1500013000,
"given_name": "Jane",
"iat": 1500009400,
"email": "janedoe@example.com",
"jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
"origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee"

}
```

Estructura de un JWT de tipo ID token

Descripción campos más relevantes del Payload (llamados claims o permisos):

- sub (subject): Id único del usuario autenticado (no es el user name, lo genera Cognito).
- aud (audience): client_id usado en la autenticación de usuario.
- cognito:groups: lista de grupos a los que el usuario pertenece.
- token_use: indica cómo se va a usar el token (id token o access token).
- auth_time: indica el tiempo en formato UTC (segundos) de cuándo se autenticó el usuario.
- iss (issuer/emisor): URL con información de zona de creación del user pool y el user pool ID.
- cognito:username : nombre de usuario en user pool.
- exp: tiempo en UTC (Unix time en segundos) en que expira el token.
- iat: tiempo en UTC (Unix time en segundos) en que fué emitido el token.
- jti: identificador único del token.

Estructura de un JWT de tipo ID token

Otros campos adicionales del ID token:

- email_verified: indica si se ha verificado el email (true/false).
- phone_number_verified: indica si se ha verificado el phone number (true/false).
- cognito:roles : indica el rol de IAM asociado.
- phone_number: indica el número de teléfono del usuario.
- email: indica la dirección de correo del usuario.

Estructura de un JWT de tipo ID token

Contenido de la firma del ID token (válido también para el access token):

- Contiene la firma del token calculada a partir de la información del header y el payload, codificados en base64url y separados por un carácter de punto (.), y luego se encripta esta información con la clave privada del algoritmo asimétrico RSA.

Estructura de un JWT de tipo access token

- Header:

```
{
"kid" : "1234example="
"alg" : "RS256",
}
```

- kid: nos indica cuál de las dos keys fué usada para la firma del JWT.
- alg: indica el algoritmo criptográfico usado para firmar el JWT.

Estructura de un JWT de tipo access token

- Payload:

```
{
  "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "device_key": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "cognito:groups": [
    "admin"
  ],
  "token_use": "access",
  "scope": "aws.cognito.signin.user.admin",
  "auth_time": 1562190524,
  "iss": "https://cognito-idp.us-west-2.amazonaws.com/us-west
  "exp": 1562194124,
  "iat": 1562190524,
  "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "client_id": "57cbishk4j24pabc1234567890",
  "username": "janedoe@example.com"
}
```

Estructura de un JWT de tipo access token

Descripción campos más relevantes del Payload (llamados claims o permisos):

- sub (subject): Id único del usuario autenticado (no es el user name, lo genera Cognito).
- cognito:groups : lista de grupos a los que el usuario pertenece.
- token_use: indica cómo se va a usar el token (id token o access token).
- scope: scopes o permisos que otorga el token (profile, email, phone).
- auth_time: indica el tiempo en formato UTC (segundos) de cuándo se autenticó el usuario.
- iss (issuer/emisor): URL con información de la zona de creación del user pool y el user pool ID.
- exp: tiempo en UTC (Unix time en segundos) en que expira el token.
- iat: tiempo en UTC (Unix time en segundos) en que fué emitido el token.
- jti: identificador único del token.
- client_id: identificador de la aplicación (app client identifier Cognito).
- username: user name del usuario.

Verificación de un JWT

- Verificación de la estructura (Header.Payload.Signature = 111.222.333).
- Verificación de la firma:
  - Decodificar el token.
  - Comparar la key ID local (kid) con la key pública correcta del JWKS (con el mismo kid).
- Usar la clave pública del JWKS para verificar la firma (usaremos librerías para hacerlo).
- Verificar los claims:
  - Verificar que el token no ha expirado.
  - El campo audiencia (aud) debe coincidir con el client ID creado en Cognito User Pool.
  - El issuer (iss) debe coincidir con el User Pool ID, con el formato: https://cognito-idp.us-east-1.amazonaws.com/<userpoolID>
  - Comprobar el token_use (id token o access token).

Revocación de un Refresh Token

- Cuando creamos un User Pool la revocación de tokens está activada por defecto.
- Para revocar un refresh token podemos usar la API RevokeToken de Cognito o usar comandos de la CLI.
- También podemos revocar un refresh token utilizando un endpoint de revocación.
  Para ello, deberemos hacer una petición POST con esta estructura:

```
POST /revoke HTTP/1.1
    Host: https://auth-domain.auth.us-east-1.amazoncognito.com
    Accept: application/json
    Content-Type: application/x-www-form-urlencoded
    Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
    token=2YotnFZFEjr1zCsicMWpAA
```

Revocación de un Refresh Token

- En el campo “token” pasaremos el refresh token que queremos revocar. Automáticamente quedará inhabilitado el mismo refresh token y también el access token correspondiente.
- El campo “Authorization” incluye el client_id y client_secret si el cliente ha generado estos parámetros.

JWK y JWKS - JSON Web Key , JSON Web Key Set

- El JWK es un formato que nos permite representar una llave pública. Y el JWKS normalmente es un fichero proporcionado por el proveedor de identidad (en nuestro caso Cognito) que contiene un par de llaves, es decir contiene dos JWK.

- Estos ficheros de llaves públicas, son proporcionados por AWS Cognito mediante una URL conocida. De esta forma, cuando obtengamos estas keys, podremos comprobar que una de ellas es la que nos permite verificar o validar, que el token ha sido firmado correctamente con la correspondiente llave privada que pertenece a esa llave pública.

- Veremos cómo es su estructura en la próxima slide.

JWKS - Estructura

- Aquí podemos ver cómo son los ficheros “jwks.json” de AWS Cognito (esta imagen es de la documentación de Amazon).
- Normalmente usaremos una librería externa que hará la verificación de las keys de forma automática, por lo que no nos debemos preocuparnos por eso.
- En la siguiente slide describiremos los diferentes campos.

JWKS - Descripción campos

- kid: identificador de la clave.
- alg: contiene el algoritmo criptográfico usado para proteger el token, en nuestro caso “RS256”.
- kty: identifica la familia de algoritmos usados con la clave (“RSA” en nuestro caso).
- e: valor del exponente de la clave pública RSA.
- n: valor del módulo de la clave pública RSA.
- use: describe el uso que se dará a la clave pública, nosotros usaremos el valor “sig” ya que la pareja de la llave pública (la llave privada) se ha usado para firmar.

- **NOTA: los campos “e” y “n” son dos parámetros matemáticos que permiten construir la llave pública.**

JWKS - URL de descarga

Podremos utilizar la JWK (JSON Web Key) para nuestro user pool, como parte del JWKS (JSON Web Key Set). Podremos descargar el fichero del JWKS de la URL siguiente:

- https://cognito-idp.{region}.amazonaws.com/{userPoolid}/.well-known/jwks.json

Amazon Cognito. Escenarios:

Otros escenarios en la documentación oficial:

https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-scenarios.html

Amazon Cognito. Derechos imágenes:

Algunas capturas de pantalla utilizadas de esta presentación, han sido extraídas de la documentación pública oficial de AWS:

https://docs.aws.amazon.com/cognito/index.html