COSTRUIAMO UN SISTEMA DI AUTENTICAZIONE MACCHINA-MACCHINA CON AMAZON COGNITO

Amazon Cognito

beSharp | 18 Settembre 2020

Sempre più applicazioni, sia mobile che web, si affidano a servizi gestiti come Amazon Cognito per l'autenticazione e l'autorizzazione degli utenti. Cognito consente di sviluppare rapidamente applicazioni sicure, in linea con standard di sicurezza robusti ed affermati per l'autenticazione e l'autorizzazione degli utenti finali.

Sfruttare un servizio completamente gestito consente agli sviluppatori di smettere di preoccuparsi del flusso di autenticazione e della gestione del pool di utenti, lasciandoli liberi di concentrarsi su ciò che conta: la logica di business dei prodotti.

A volte è anche necessario implementare sistemi di autenticazione per consentire a servizi di terze parti di consumare API esposte dai servizi in sviluppo.

Sebbene Cognito sia utilizzato principalmente per i flussi di autenticazione degli utenti, può anche essere utilizzato per creare un sistema di autenticazione da macchina a macchina.

In questo articolo descriveremo come Amazon Cognito può essere utilizzato per autenticare un sistema client che necessita dell'accesso a un set di API sensibili esposte dal nostro servizio.

Prima di approfondire la descrizione della soluzione, descriviamo i servizi coinvolti.

Cos'è Amazon Cognito?

Quando è necessario implementare un sistema di autorizzazione e autenticazione su AWS, Amazon Cognito è sicuramente la scelta migliore.

Amazon Cognito consente di implementare un sistema di login – completo di verifica email e gestione della password – all'interno di applicazioni web e mobile in modo rapido e semplice. è in grado di gestire milioni di utenti e supporta l'accesso con Identity Provider social, come Facebook, Google e Amazon, così come Identity Provider aziendali tramite SAML 2.0.

Inoltre Cognito supporta anche l'autenticazione a più fattori e la crittografia dei dati at rest e in transito. Amazon Cognito è idoneo HIPAA e conforme a PCI DSS, SOC, ISO / IEC 27001, ISO / IEC 27017, ISO / IEC 27018 e ISO 9001.

Passiamo ora a descrivere i principali concetti di Cognito.

Gli utenti sono gestiti tramite due tipi di pool, il centro del funzionamento di Amazon Cognito: User Pool e Identity Pool.

User Pool

Uno User Pool (o pool di utenti) è essenzialmente una user directory che consente di archiviare in modo sicuro gli attributi del profilo degli utenti. Permette di fare off loading completo della manutenzione, della sicurezza e della disponibilità della user directory. Tra le operazioni che possono essere esternalizzate ci sono sicuramente la memorizzazione sicura dei dati degli utenti, la verifica dei numeri di telefono e /o degli indirizzi di posta elettronica, la gestione delle API del sistema di login, del flusso di registrazione, login, logout e la reimpostazione della password.

Oltre a utilizzare le API specifiche di Amazon Cognito per il flusso di autenticazione degli utenti, il servizio supporta anche il protocollo OAuth 2.0.

I pool di utenti sono una componente fondamentale di qualsiasi sistema di autenticazione basato su Amazon Cognito e ci torneranno utili per creare il nostro sistema di autenticazione machine-to-machine.

Identity Pool

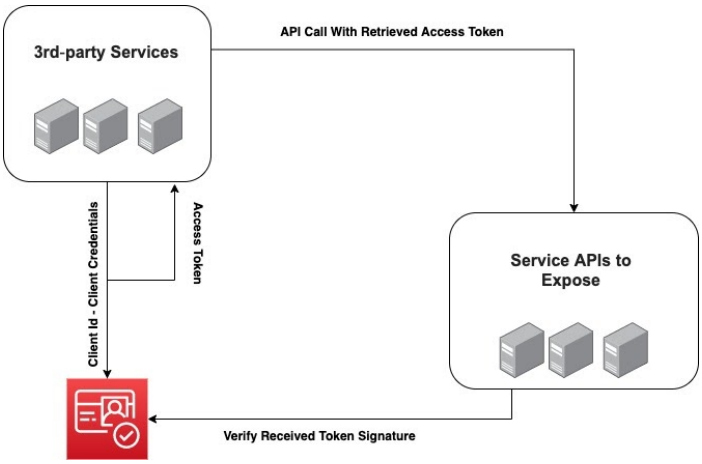
Nell'esempio che abbiamo preso in considerazione per questo articolo non sarà necessario utilizzare gli Identity Pool, ma è comunque utile descriverne brevemente il funzionamento.

Gli Identity Pool (o pool di identità) vengono utilizzati da Cognito per mantenere organizzate le identità federate. Un pool di identità associa identità federate da Identity Provider esterni, o anche da una User Pool, a un identificatore univoco. I pool di identità non memorizzano i profili utente, ma solo i loro ID univoci, che vengono generati e gestiti da Cognito. Mediante gli ID è possibile assegnare agli utenti corrispondenti un set di credenziali IAM temporanee con privilegi limitati. Gli utenti o le applicazioni client possono utilizzare queste credenziali per accedere alle risorse AWS. Le regole di autorizzazione per ogni utente sono controllate tramite ruoli e policy AWS IAM personalizzabili. È anche possibile definire regole per abbinare gli utenti al ruolo desiderato.

Ora che abbiamo definito tutti i concetti fondamentali possiamo passare alla parte centrale del nostro articolo.

Flusso di autenticazione "machine-to-machine"

Iniziamo definendo il flusso di autenticazione che configureremo nei passaggi successivi.



Come descritto nelle specifiche OAuth 2.0, possiamo autenticare un client che presenta un ID client e un client secret validi al nostro Identity Provider.

Come si può vedere nello schema, un client generico può chiamare le API di AWS Cognito con il client ID e il segreto condivisi con lo stesso. Se i due parametri sono validi, AWS Cognito restituisce un token di accesso. Da questo momento, l'applicazione di terze parti può effettuare chiamate autenticate mediante l'access token rilasciato da Cognito.

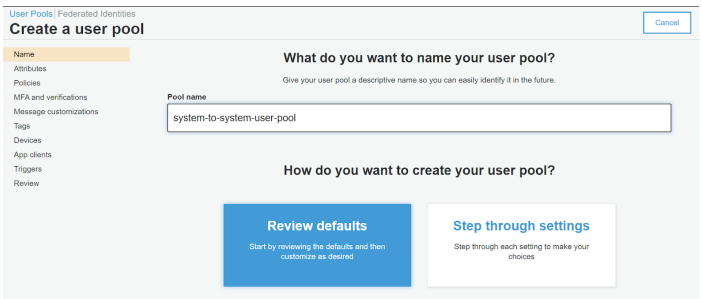
Ciò che va implementato lato servizi è la convalida del token JWT come descritto dalla specifica OAuth 2.0. Occorre.

1. Convalidare che il JWT ricevuto abbia un formato valido.
2. Convalidare la firma JWT.
3. Verifica tutti i claim.

Passiamo ora a configurare il flusso descritto all'interno del nostro account AWS.

Hands-on!

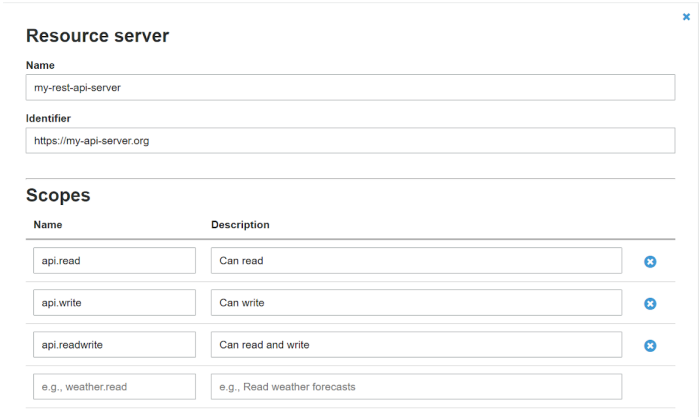
Ora che abbiamo descritto le principali funzioni di Amazon Cognito possiamo iniziare con il progetto. Prima di tutto, creiamo una User Pool dalla console AWS. Scegliamo un nome esplicativo e manteniamo tutte le impostazioni predefinite facendo clic sul pulsante "Review defaults".



A questo punto possiamo iniziare ad esplorare le numerose opzioni fornite dal nostro nuovo pool di utenti, modificando quelle necessarie per il nostro progetto.

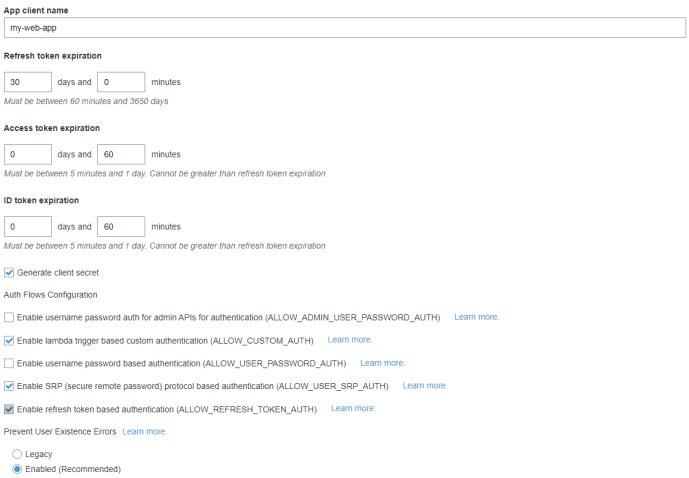
Il server o servizio che erogherà le API da autenticare può essere specificato nella pagina "resource server" nelle opzioni della User Pool.

Qui sarai in grado di determinare l'URL univoco del server che espone le API che devi proteggere. Inoltre, in questa pagina è possibile elencare una serie di scopes per discriminare, ad esempio, vari livelli di accesso al servizio.



Ora che il server di risorse è stato configurato, possiamo occuparci delle impostazioni dei client dell'app.

In questa pagina è possibile definire un nuovo client, ad esempio una nuova web-app, che deve consumare il set di API che si desidera proteggere.



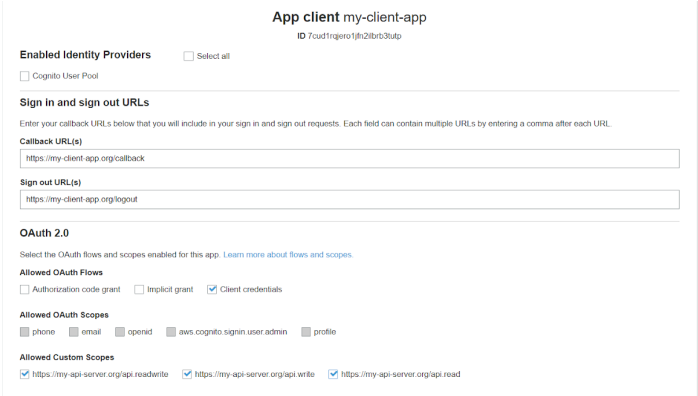
Specifichiamo il nome del nuovo client e la scadenza dei token di sicurezza utilizzati nel processo di autorizzazione. A causa delle specifiche OAuth2 non esiste un meccanismo di refresh del token, per cui è possibile solamente indicarne la scadenza. Quando il token non sarà più valido, occorrerà ripetere il flusso di autenticazione per ottenerne uno nuovo.

Salvando questi dettagli, Cognito ci chiederà un client app ID e un segreto. Sebbene l'ID sia inteso come identificativo pubblico delle app, è importante non condividere mai né l'ID né il segreto per motivi di sicurezza.

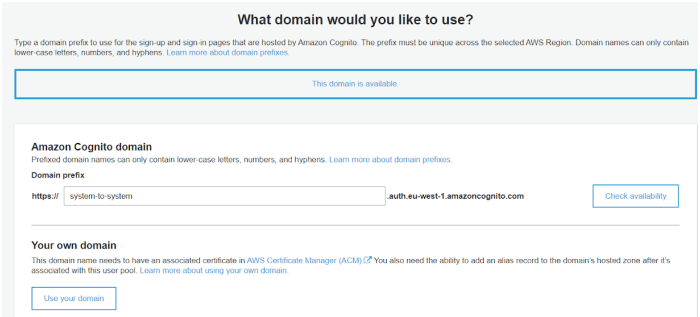
Questi due token devono essere crittografati e archiviati da ciascuna delle applicazioni registrate per richiedere i token di accesso al pool di utenti Cognito. Sono essenzialmente il nome utente e la password dell'autorizzazione da sistema a sistema.

La tua app è ora registrata correttamente ed è possibile modificare le opzioni di autorizzazione che desideri abilitare per i suoi utenti dalla **pagina di impostazioni.**

Da questa pagina, puoi anche definire quali scope possono essere utilizzati da ogni client dell'app.



Come passaggio finale sulla console Cognito, è necessario scegliere un nome di dominio. In questa pagina, infatti, sceglieremo (se disponibile) dove le applicazioni invieranno le richieste di token di accesso.



Cognito è ora correttamente configurato.

```
import functools
import json
import jwt
import urllib.request
def is_valid_token(scope) -> bool:
    def wrapped(func):
        @functools.wraps(func)
        def wrapper(access_token):
            public_keys = get_well_known_jwk(REGION, USER_POOL_ID)
            kid = jwt.get_unverified_header(access_token)['kid']
            key = public_keys[kid]
            payload = jwt.decode(access_token, key=key, algorithms=['RS256'])
            if payload['client_id'] != CLIENT_ID:
                print('Wrong client_id')
                raise Exception('Wrong client id')
            if payload['scope'] != scope:
                print('Wrong scope')
                raise Exception('Wrong scope')
            return func(access_token)
        return wrapper
    return wrapped
def get_well_known_jwk(region: str, user_pool_id: str) -> dict:
    jwk_url = f"https://cognito-idp.{region}.amazonaws.com/{user_pool_id}/.well-known
/jwks.json"
    with urllib.request.urlopen(jwk_url) as url:
        jwks = json.loads(url.read().decode())
    public_keys = {}
    for jwk in jwks['keys']:
        kid = jwk['kid']
```

```
        public_keys[kid] = jwt.algorithms.RSAAlgorithm.from_jwk(json.dumps(jwk))
    return public_keys
@is_valid_token(scope = 'https://my-api-server.org/api.write')
def api(access_token):
    resp = {
        "status_code": 200,
        "body": "This is your body."
    }
    print(json.dumps(resp))
```

Come si evince dallo snippet sopra, il recupero del token di accesso è una richiesta HTTP POST abbastanza semplice che necessita nel suo corpo dei pochi semplici parametri visti prima. L'endpoint è composto usando il dominio specificato nelle impostazioni di Cognito.

Il token di accesso che abbiamo appena ricevuto è pronto per essere spedito all'interno di una chiamata API al server che eroga le API autenticate. Quest'ultimo, quindi, deve essere in grado di decodificarlo e convalidarlo.

```
import functools
import json
import jwt
import urllib.request

def is_valid_token(func) -> bool:
    @functools.wraps(func)
    def wrapper(access_token, scope):
        public_keys = get_well_known_jwk(REGION, USER_POOL_ID)
        kid = jwt.get_unverified_header(access_token)['kid']
        key = public_keys[kid]
        payload = jwt.decode(access_token, key=key, algorithms=['RS256'])
        if payload['client_id'] != CLIENT_ID:
            raise Exception('Wrong client id')
        if payload['scope'] != scope:
            raise Exception('Wrong scope')
        return func(access_token)
    return wrapper


def get_well_known_jwk(region: str, user_pool_id: str) -> dict:
    jwk_url = f"https://cognito-idp.{region}.amazonaws.com/{user_pool_id}/.well-known
/jwks.json"
    with urllib.request.urlopen(jwk_url) as url:
        jwks = json.loads(url.read().decode())

    public_keys = {}
    for jwk in jwks['keys']:
        kid = jwk['kid']
        public_keys[kid] = jwt.algorithms.RSAAlgorithm.from_jwk(json.dumps(jwk))
    return public_keys


@is_valid_token
def api(access_token):
    resp = {
        "status_code": 200,
        "body": "This is your body."
```

```
    }
    return resp
```

È ora possibile testare la soluzione e autenticare altri servizi della propria applicazione.

È tutto per oggi! In questo articolo, abbiamo spiegato come creare un sistema di autenticazione di tipo macchina-macchina sicuro, affidabile e completamente gestito sfruttando Amazon Cognito e gli User Pool.

Contattateci o lasciateci un commento qui sotto per farci domande o semplicemente per aggiungere considerazioni sull'argomento.

Ci vediamo **tra 14 giorni** per il prossimo articolo sul blog **#Proud2beCloud**!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it