File Transfer da e verso Amazon S3 via SFTP con AWS Transfer Family

14 Maggio 2021 - 7 min. read

Amazon S3 | AWS Lambda | AWS Transfer Family | SFTP

I protocolli per il trasferimento di file verso server in remoto esistono dagli albori del networking tra computer. FTP (esattamente, File Transfer Protocol) è uno degli elementi fondamentali di Internet. Sviluppato da uno studente del MIT all'inizio degli anni '70, FTP è diventato lo standard per il trasferimento e la gestione dei file remoti per decenni.

Nel corso degli anni, FTP è stato aggiornato per fornire numerosi vantaggi. Gli esempi più importanti, SFTP e FTPS sono stati sviluppati per soppiantare lo storico protocollo data la capacità di stabilire stream di dati sicuri.

Il servizio AWS gestito, AWS Transfer Family, fornisce un set di risorse completamente gestite per supportare un modo aggiuntivo per trasferire file dentro e fuori AWS. Questo servizio consente l'esposizione di una comoda interfaccia per gestire gli oggetti su Amazon S3 ed Amazon EFS utilizzando protocolli di trasferimento file ben noti come FTP, SFTP e FTPS.

Come funziona?

Questo servizio AWS consente di evitare gli ostacoli portati dalla manutenzione di server FTP. Infatti, AWS Transfer Family si occupa di far scalare i server EC2 sottostanti garantendo il giusto apporto di risorse, mantenendo l'intero servizio altamente disponibile.

Per l'autenticazione degli utenti, AWS Transfer Family consente di scegliere tra soluzioni gestite direttamente dal servizio e personalizzate. La prima opzione, tuttavia,

pur consentendo una configurazione molto rapida del servizio utilizzando le chiavi SSH RSA generate da AWS per l'autenticazione SFTP, non supporta l'integrazione con i meccanismi di autenticazione esistenti o anche la semplice autenticazione con la vecchia password e nome utente. La seconda opzione invece, ti dà carta bianca quando è necessario integrare un identity provider esistente. Ad esempio è possibile utilizzare LDAP o Microsoft Active Directory come IdP, oppure configurare sistemi di autenticazione personalizzati grazie a funzioni lambda create ad-hoc.

Come detto in precedenza, AWS Transfer Family consente l'accesso ai file remoti archiviati su S3 o EFS utilizzando i protocolli FTP, SFTP e FTPS. È importante notare tuttavia, che l'utilizzo di FTP non è supportato per i workload internet-facing, infatti, le connessioni FTP (a differenza di quelle SFTP e FTPS) sono considerate non sicure a causa del trasferimento in chiaro delle credenziali. Infatti, è consentita l'esposizione del servizio solo in modalità VPC.

Perché AWS Transfer Family?

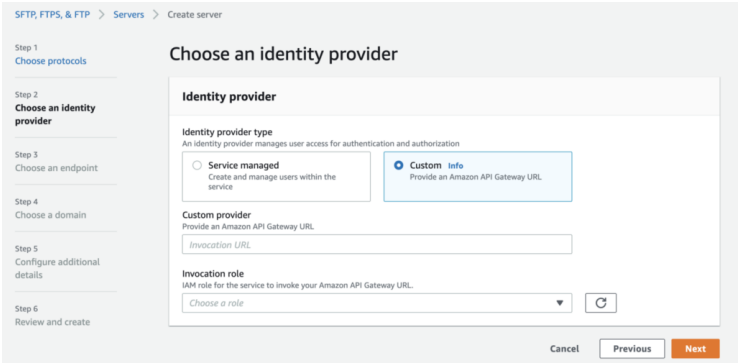
Il provisioning dell'infrastruttura necessaria per mantenere un'architettura che consenta il trasferimento di file FTP, SFTP o FTPS può essere molto onerosa in termini sia economici che di manutenzione. AWS Transfer Family consente di configurare nuovi workload o migrare quelli esistenti limitando queste preoccupazioni.

Va sottolineato che la migrazione di carichi di lavoro simili a quelli offerti da AWS Transfer Family non influisce sull'esperienza utente, consentendo loro di utilizzare i client FTP a cui sono abituati, come Cyberduck, FileZilla, WinSCP, ecc.

Al contempo, l'utilizzo di questo servizio consente di portare i tuoi dati all'interno di un servizio di archiviazione durevole e altamente disponibile. Come vedremo nella parte successiva, l'adozione di S3, ad esempio, può dare spazio a un numero significativo di casi di business interessanti.

Autenticazione degli utenti autogestita

Per configurare l'autenticazione utente autogestita in AWS Transfer Family, dovremo specificare un endpoint di API Gateway e un ruolo per chiamare le sue api.



L'API Gateway (passaggi di configurazione sono disponibili qui) deve esporre un'API a cui viene associata una AWS Lambda. Questa API verrà chiamata da AWS Transfer Family per verificare le credenziali dell'utente che ha effettuato una richiesta FTP al servizio.

La funzione Lambda conterrà la logica necessaria per autenticare l'utente. Qui è dove avviene la magia, infatti, sarai in grado di implementare qualsiasi tipo di autenticazione: dalla chiamata all'API di IdP esterni, ad una implementazione da zero utilizzando, ad esempio, altri servizi AWS.

Per semplificare la vita dello sviluppatore, AWS Transfer Family includerà le seguenti proprietà nel payload della richiesta HTTP GET:

```
{
   "username": "The user's username",
   "password": "The user's password, if empty SSH authentication is u
sed",
   "serverId": "ID of the AWS Transfer Family server",
   "protocol": "FTP | SFTP | FTPS",
   "sourceIp": "IP Address of the client"
}
```

La funzione lambda, quindi, dovrà implementare la logica per verificare le credenziali ricevute dal servizio FTP rispetto allo user store scelto. In caso di convalida riuscita, dovrebbe essere restituito un oggetto JSON.

Più specificamente, AWS Transfer Family si aspetta una risposta con il seguente schema:

```
{
   "Role": "[REQUIRED] ARN of a role allowing the base access to S3 b
```

```
uckets and KMS keys (if needed).",
   "HomeDirectory": "The S3 bucket prefix that the user will be dropp
ed into when they log in.",
   "Policy": "A scope-down policy useful to restrict access to certai
n buckets according to the user's details.",
   "PublicKeys": "If no password was provided (SSH key-based authenti
cation), then the public SSH key associated with the user is returne
d. This is a comma-separated list and can contain two public keys.",
   "HomeDirectoryDetails": "This is useful to define a logical bucket
structure (symlink). In this way it's possible to show in the FTP cli
ent all the buckets to which a user has access, or create logical lin
ks to folders."
}
```

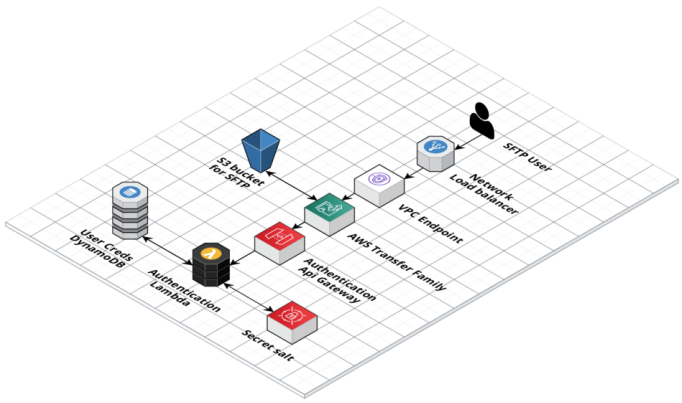

Uno use case reale

Come accennato in precedenza, AWS Transfer Family consente di sviluppare e configurare meccanismi di autenticazione personalizzati. Ad esempio, sarebbe possibile utilizzare questa funzionalità, come vedremo in questo capitolo, per creare un meccanismo completamente nuovo utilizzando i mezzi forniti da AWS.

L'obiettivo di questo articolo è presentare un esempio reale della migrazione di un carico di lavoro SFTP ospitato su EC2 ad AWS Transfer Family, utilizzando un'autenticazione personalizzata utilizzando DynamoDB come user store con latenza dell'ordine del millisecondo.

L'architettura

Per raggiungere l'obiettivo descritto sopra, dobbiamo prima delineare gli elementi costitutivi dell'infrastruttura.



Come puoi vedere dallo schema, abbiamo dovuto introdurre un paio di servizi che non avevamo previsto, come un Network Load Balancer e un VPC endpoint. Questi due componenti sono necessari per assicurarsi che gli IP esposti siano fissi e non cambieranno mai. In questo modo, infatti, è possibile allocare un elastic IP, consentendo anche di importare indirizzi IP di tua proprietà (BYOIP). Questa funzione è comunemente richiesta per problemi specifici con alcune regole di firewall.

A parte i componenti appena citati, quando una richiesta FTP (o SFTP / FTPS) viene inviata dal client dell'utente, essa viene instradata attraverso la VPC al servizio AWS Transfer Family.

A questo punto si verifica lo scenario descritto nel paragrafo Autenticazione degli utenti autogestita.

Infatti, AWS Transfer Family inoltrerà le informazioni richieste per l'autenticazione dell'utente all'API Gateway interno, che invocherà la lambda di autenticazione.

Cosa succede nella lambda di autenticazione

Come detto prima, il processo di autenticazione utilizzerà la tabella DynamoDB come user store. In questa tabella, infatti, verranno memorizzati alcuni importanti campi ed i suoi record rifletteranno la seguente struttura:

```
{
   "hashPassword": "Used as partition key, is an hash of a combinatio
n of the salt, password and username",
   "username": "The username by which a user can authenticate with a
 password",
   "bucket": "Name of the bucket to which the user can connect",
   "home": "Home directory of the selected bucket",
   "iamPolicyArn": "ARN of the scope-down policy of the user"
}
```

Prima di tutto, viene calcolato l'hash della combinazione delle credenziali ricevute più un salt segreto e fisso (archiviato in AWS Secrets Manager).

La scelta del campo *hashPassword* come partition key consente l'accesso diretto alle informazioni di un utente. Ancora più importante, una query restituirà il record specifico di un utente solo se le credenziali sono corrette, altrimenti viene ricevuta una

risposta vuota. Ciò, infatti, si traduce in una query molto veloce sulla tabella DynamoDB, garantendo un'autenticazione quasi istantanea.

Quando si verifica un'autenticazione con esito positivo, la lambda genererà un oggetto compatibile con AWS Transfer Family e lo restituirà al servizio.

```
if user:
    print("Authenticated")
    home_directory = "/" + user['bucket'] + "/" + user['home']

    policy_arn = user['iamPolicyArn']
    policy_info = iam.get_policy(PolicyArn=policy_arn)

    policy_document = json.dumps(iam.get_policy_version(
        PolicyArn=policy_arn,
        VersionId=policy_info['Policy']['DefaultVersionId']
    )['PolicyVersion']['Document'])

    role_arn = "arn:aws:iam::" + \
        os.environ['AWS_ACCOUNT_ID'] + ":role/" + \
        os.environ['USER_S3_ACCESS_ROLE']

    body = {
        'Role': role_arn,
        'Policy': policy_document,
        'HomeDirectory': home_directory
    }

    print("Body: ", body)
    return body
else:
    # Answer is empty if not succeeded
    print("Not Authenticated")
    return {}
```

Per concludere

Sebbene esistano molti protocolli per il trasferimento di file, lo storico protocollo FTP è ancora ampiamente utilizzato. La creazione o la migrazione di questi carichi di lavoro è notevolmente semplificata dall'adozione di servizi gestiti come AWS Transfer Family.

Inoltre, è importante notare che l'integrazione di servizi gestiti e serverless utilizzati, sia per l'archiviazione (S3), sia per la gestione degli utenti e la logica di autenticazione (es. ApiGateway / lambda e DynamoDB) consente al servizio di scalare per adattarsi a qualsiasi carico di lavoro e numero di utenti.

Cosa pensate delle risorse AWS Transfer Family? Le avete già utilizzate? Raccontateci dei vostri progetti nei commenti! A tra 14 giorni su **#Proud2beCloud** con un nuovo articolo dei nostri Cloud Expert!

Christian Calabrese

DevOps Engineer e Cloud Native Applications Developer @ beSharp. Appassionato del mondo dell'HiFi e macinatore incallito di videogiochi!
