

GO SERVERLESS! PARTE 2: L'INFRASTRUTTURA PER L'APPLICAZIONE DI FILE SHARING

AWS Lambda

CI/CD

Continuous Delivery

Serverless



beSharp | 25 Settembre 2018

[Leggi la prima parte](#) | [Leggi la terza parte](#)

Con la sempre maggiore diffusione dello sviluppo Serverless, è di fondamentale importanza comprendere come sfruttarne tutti i vantaggi per poter realizzare applicazioni **scalabili** e **resilienti**.

Questa è la seconda parte della serie di 3 articoli ([qui](#) trovate la prima parte) su come costruire una **piattaforma di file sharing completamente serverless**. In questo articolo andremo a realizzare l'infrastruttura, approfondendo gli aspetti fondamentali per una corretta ed efficace implementazione.

Come anticipato nell'[articolo](#) precedente, oltre all'infrastruttura dell'applicazione andremo anche a realizzare le **pipeline di Continuous Integration e Continuous Delivery**.

Cominciamo con il primo dei due punti.

Non tratteremo qui del codice applicativo e degli argomenti correlati allo sviluppo, che saranno descritti in modo approfondito nel prossimo articolo. Per testare l'infrastruttura impiegheremo delle semplici pagine statiche e una funzione lambda di test.

Iniziamo con l'analisi della soluzione presentata nella prima parte della serie.

Create bucket

1 Name and region 2 Configure options 3 Set permissions 4 Review

Name and region

Bucket name [?](#)

fileshare-example

Region

EU (Ireland)

Copy settings from an existing bucket

Select bucket (optional) 30 Buckets

Create Cancel Next

Create bucket

1 Name and region 2 Configure options 3 Set permissions 4 Review

Manage users

User ID ?	Objects ?	Object permissions ?
fileshare-Owner	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write

Access for other AWS account [+ Add account](#)

Account ?	Objects ?	Object permissions ?
---------------------------	---------------------------	--------------------------------------

Manage public permissions

Do not grant public read access to this bucket (Recommended)

Manage system permissions

Do not grant Amazon S3 Log Delivery group write access to this bucket

Previous Next

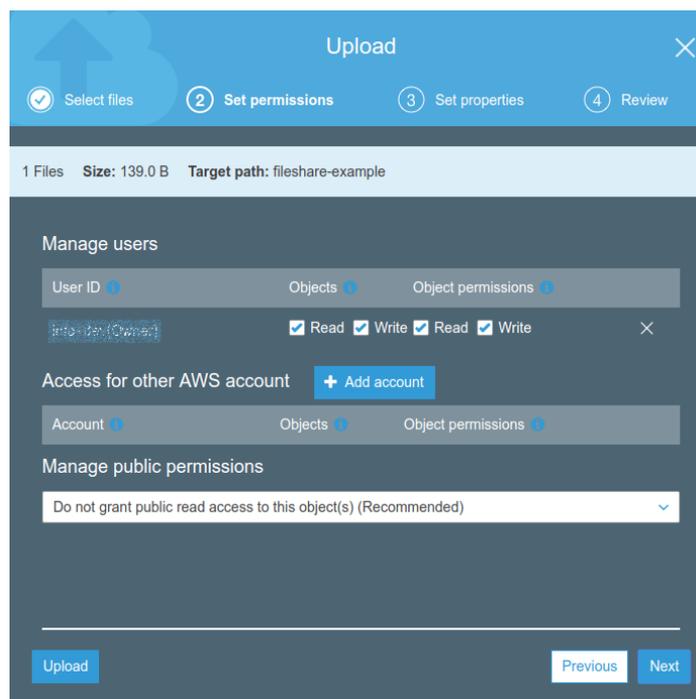
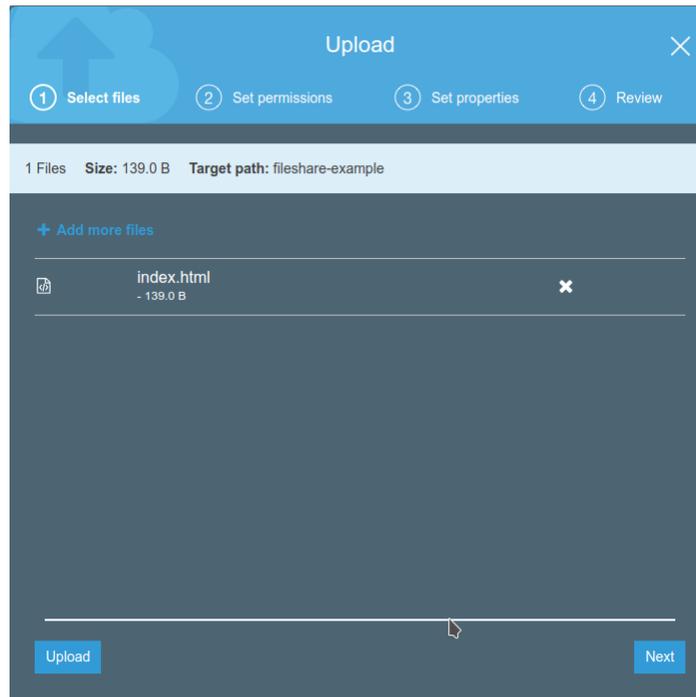
Dopo è possibile caricare un semplice file di prova, che useremo per verificare il corretto funzionamento del front-end.

```
<html>
  <head>
    <title>Esempio</title>
  </head>

  <body>
```

```
<h1>It Works</h1>
<hr/>
<p>This is a simple test page</p>
</body>
</html>
```

Salviamo questo file come “**index.html**” e carichiamolo nel bucket appena creato.



Una volta ultimate le operazioni per la creazione del bucket S3 occorre creare una **distribuzione CloudFront** che lo utilizzi come origine. Questo per distribuire il file che abbiamo caricato, che non è al momento accessibile.

Nella configurazione della distribuzione è importante abilitare l'opzione **“Restrict Bucket Access”** per mantenere il bucket S3 privato.

Al fine di consentire al servizio CloudFront di accedere al bucket in lettura bisogna provvedere ad attivare e configurare l'opzione **“Origin Access Identity”**; è possibile delegare al wizard la creazione di una Access Identity e l'aggiornamento della Bucket Policy.

Origin Settings

Origin Domain Name	<input type="text" value="fileshare-example.s3.amazonaws.com"/>	
Origin Path	<input type="text"/>	Optional. If you want CloudFront to request your content from a specific path, you can specify a path. For example, myawsbucket/production. Do not include a / at the end.
Origin ID	<input type="text" value="S3-fileshare-example"/>	
Restrict Bucket Access	<input checked="" type="radio"/> Yes <input type="radio"/> No	
Origin Access Identity	<input checked="" type="radio"/> Create a New Identity <input type="radio"/> Use an Existing Identity	
Comment	<input type="text" value="access-identity-fileshare"/>	
Grant Read Permissions on Bucket	<input checked="" type="radio"/> Yes, Update Bucket Policy <input type="radio"/> No, I Will Update Permissions	
Origin Custom Headers	Header Name <input type="text"/> Value <input type="text"/>	

Per quanto riguarda le impostazioni del comportamento di **caching** attiviamo il redirect di http su https e abilitiamo la compressione automatica dei contenuti serviti per migliorare le performance.

Default Cache Behavior Settings

Path Pattern	Default (*)	
Viewer Protocol Policy	<input type="radio"/> HTTP and HTTPS <input checked="" type="radio"/> Redirect HTTP to HTTPS <input type="radio"/> HTTPS Only	
Allowed HTTP Methods	<input type="radio"/> GET, HEAD <input checked="" type="radio"/> GET, HEAD, OPTIONS <input type="radio"/> GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE	
Field-level Encryption Config	<input type="text" value=""/>	
Cached HTTP Methods	GET, HEAD (Cached by default) <input checked="" type="checkbox"/> OPTIONS	
Cache Based on Selected Request Headers	<input type="text" value="None (Improves Caching)"/> Learn More	
Object Caching	<input checked="" type="radio"/> Use Origin Cache Headers <input type="radio"/> Customize Learn More	
Minimum TTL	<input type="text" value="0"/>	
Maximum TTL	<input type="text" value="31536000"/>	
Default TTL	<input type="text" value="86400"/>	
Forward Cookies	<input type="text" value="None (Improves Caching)"/>	
Query String Forwarding and Caching	<input type="text" value="None (Improves Caching)"/>	
Smooth Streaming	<input type="radio"/> Yes <input checked="" type="radio"/> No	
Restrict Viewer Access (Use Signed URLs or Signed Cookies)	<input type="radio"/> Yes <input checked="" type="radio"/> No	
Compress Objects Automatically	<input checked="" type="radio"/> Yes <input type="radio"/> No	

Non bisogna dimenticare di indicare quale file servire di default se non indicato, ed impostarlo ad index.html

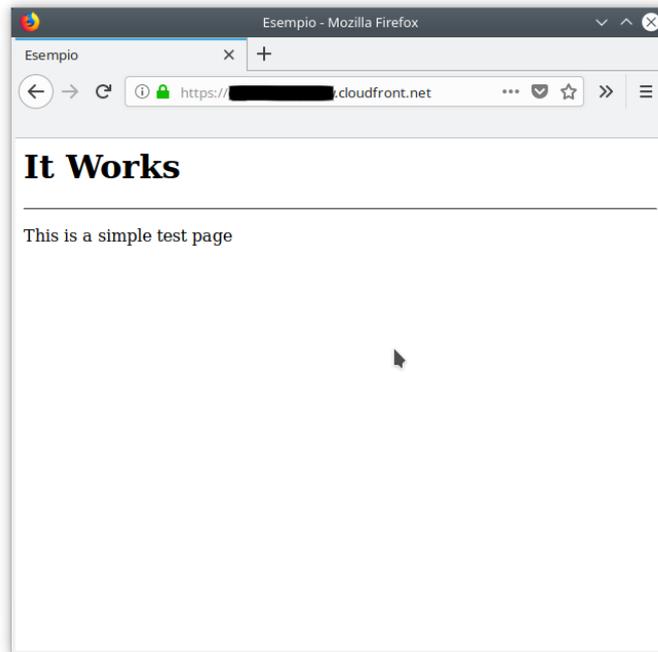
Supported HTTP Versions	<input checked="" type="radio"/> HTTP/2, HTTP/1.1, HTTP/1.0 <input type="radio"/> HTTP/1.1, HTTP/1.0	
Default Root Object	<input type="text" value="index.html"/>	

Il processo di creazione della distribuzione può richiedere diversi minuti.

In alcuni casi, anche dopo l'attivazione della distribuzione potrebbe essere visualizzato un errore di accesso, tuttavia si tratta di una situazione transitoria dovuta ai tempi di propagazione DNS, e si risolve autonomamente entro qualche minuto. [Può richiedere fino ad un ora NDR]

Quando la distribuzione sarà pronta è possibile ottenere l'url pubblico e testare il deploy del front-end navigando a quell'indirizzo.

Dovrebbe essere visualizzata la pagina di esempio che abbiamo caricato nei passaggi precedenti.



Il front-end è quindi pronto. Il contenuto del bucket viene servito in modo efficiente e cost effective attraverso la CDN; per aggiornare l'applicazione basta caricarla su S3.

Attenzione! se si modificano file già presenti occorre invalidare la cache della distribuzione per vedere gli aggiornamenti.

PREPARAZIONE DEL BACK-END

Passiamo ora alla realizzazione dell'infrastruttura del back-end del nostro sistema di file sharing.

Come prima cosa possiamo **creare una funzione lambda** che utilizzeremo ai fini di test.

Come abbiamo già detto, non ci occuperemo adesso dell'applicazione vera e propria, ma solo di costruire tutte le parti dell'infrastruttura. Questa funzione non farà altro che restituire uno stato di successo ed un messaggio.

Procediamo quindi con la creazione della funzione impostando Python come Runtime e assegnando un ruolo con permessi di base.

Crea da zero [Informazioni](#)

Nome
FSTest

Runtime
Python 3.6

Ruolo
Conferma le autorizzazioni della tua funzione. Da notare che i ruoli ruoli potrebbero non essere disponibili per alcuni minuti dopo la creazione. [Ulteriori informazioni](#) sui ruoli di esecuzione Lambda.
 Selezione un ruolo esistente

Ruolo esistente
Con questa funzione potresti usare un ruolo esistente. Da notare che il ruolo deve essere assumibile da Lambda e deve avere le autorizzazioni per CloudWatch Logs.
 lambda_basic_execution

Ecco un esempio per restituire sempre un messaggio:

```
def lambda_handler(event, context):

    response = {
        "statusCode": 200,
        "body": 'Hello from Lambda!'
    }

    return response
```

Testiamo la funzione, se tutto è stato impostato correttamente il test dovrebbe riuscire con un output simile a quello seguente:

```
{
  "statusCode": 200,
  "body": "Hello from Lambda!"
}
```

Una volta pronta la funzione possiamo effettuare il **provisioning delle altre risorse**.

All'applicazione occorrerà un ulteriore bucket S3 per gli upload dei file degli utenti. Anche in questo caso si tratterà di un bucket privato. Provvediamo quindi alla sua creazione nello stesso modo utilizzato per il front-end, e prendiamo nota del nome del bucket, ci servirà per la configurazione dell'applicazione.

Servirà anche una **tabella di DynamoDB** per contenere i metadati e le informazioni di share dei file. Procediamo quindi alla creazione di una tabella lasciando i valori di default e specificando "shareID" come chiave di partizione primaria.

Dettagli tabella

Nome tabella	fs-share
Chiave di partizione primaria	shareID (Stringa)
Chiave di ordinamento primaria	-
Recupero temporizzato	-
Crittografia	DISABILITATO
Attributo Time to Live (Periodo di vita)	-
Stato tabella	Creazione in corso
Data di creazione	19 settembre 2018 17:10:51 UTC+2
Unità di capacità in lettura con provisioning	5(Auto Scaling Disabilitato)
Unità di capacità in scrittura con provisioning	5(Auto Scaling Disabilitato)
Creazione dell'ultima riduzione	-

A questo punto dobbiamo occuparci di **Cognito** e di **API Gateway**. Il primo è necessario per la parte di autenticazione, ed il secondo permetterà al front-end di accedere alle funzionalità del back-end lambda mediante chiamate https.

Per prima cosa creiamo e configuriamo una **Cognito User Pool**.

Durante il Wizard di creazione possiamo personalizzare alcuni comportamenti; per far funzionare l'applicazione bisogna creare un Pool che permetta gli utenti di registrarsi mediante email e di verificare l'indirizzo utilizzando un codice di sicurezza gestito da Cognito. Occorre anche creare un'app client e prendere nota del suo ID.

Scegliamo un nome per la User Pool e avviamo il wizard

Create a user pool Cancel

Name

What do you want to name your user pool?
Give your user pool a descriptive name so you can easily identify it in the future.

Pool name
Required

How do you want to create your user pool?

[Review defaults](#)
Start by reviewing the defaults and then customize as desired

[Step through settings](#)
Step through each setting to make your choices

Configuriamo il meccanismo di autenticazione e i campi del profilo.

Create a user pool Cancel

Attributes

How do you want your end users to sign in?
You can choose to have users sign in with an email address, phone number, username or preferred username plus their password. [Learn more.](#)

Username - Users can use a username and optionally multiple alternatives to sign up and sign in.
 Also allow sign in with verified email address
 Also allow sign in with verified phone number
 Also allow sign in with preferred username (a username that your users can change)

Email address or phone number - Users can use an email address or phone number as their "username" to sign up and sign in.
 Allow email addresses
 Allow phone numbers
 Allow both email addresses and phone numbers (users can choose one)

Which standard attributes do you want to require?
All of the standard attributes can be used for user profiles, but the attributes you select will be required for sign up. You will not be able to change these requirements after the pool is created. If you select an attribute to be an alias, users will be able to sign-in using that value or their username. [Learn more about attributes.](#)

Required	Attribute	Required	Attribute
<input type="checkbox"/>	address	<input type="checkbox"/>	nickname
<input type="checkbox"/>	birthdate	<input type="checkbox"/>	phone number
<input checked="" type="checkbox"/>	email	<input type="checkbox"/>	picture
<input type="checkbox"/>	family name	<input type="checkbox"/>	preferred username
<input type="checkbox"/>	gender	<input type="checkbox"/>	profile
<input type="checkbox"/>	given name	<input type="checkbox"/>	zoneinfo
<input type="checkbox"/>	locale	<input type="checkbox"/>	updated at
<input type="checkbox"/>	middle name	<input type="checkbox"/>	website
<input checked="" type="checkbox"/>	name		

Infine bisogna creare un app client e prendere nota del suo ID. Non bisogna far generare alcun segreto, non servirà per l'utilizzo di Cognito mediante web app.

Create a user pool Cancel

App clients

Which app clients will have access to this user pool?
The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.

[Add an app client](#) [Return to pool details](#)

Create a user pool Cancel

App clients

Which app clients will have access to this user pool?
The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.

App client name
FileSharing

Refresh token expiration (days)
30

Generate client secret

Enable sign-in API for server-based authentication (ADMIN_NO_SRP_AUTH) [Learn more.](#)

Only allow Custom Authentication (CUSTOM_AUTH_FLOW_ONLY) [Learn more.](#)

Enable username-password (non-SRP) flow for app-based authentication (USER_PASSWORD_AUTH) [Learn more.](#)

Set attribute read and write permissions

[Cancel](#) [Create app client](#)

Con Cognito abbiamo finito per ora, queste risorse saranno utilizzate dall'applicazione e da API Gateway per il meccanismo di autenticazione.

Infine procediamo alla creazione di una API su API Gateway.

Questo componente fa da interfaccia tra le richieste dell'utente e la funzione lambda, inoltre si occuperà in modo automatico dell'autenticazione delle richieste mediante la user pool di Cognito.

Per scopo di test procediamo alla creazione di una API semplice con soltanto un metodo GET che chiama la funzione lambda.

Choose the integration point for your new method.

Integration type

- Lambda Function
- HTTP
- Mock
- AWS Service
- VPC Link

Use Lambda Proxy integration

Lambda Region eu-west-1

Lambda Function FSTest

Use Default Timeout

Una volta configurata l'integrazione è possibile testarla

```
Request: /
Status: 200
Latency: 212 ms
Response Body
{
  "statusCode": 200,
  "body": "Hello from Lambda!"
}
Response Headers
{"X-Amzn-Trace-Id":"Root=1-5ba26c3d-4f8c08b3585816ab2fe6640d;Sampled=0","Content-Type":"application/json"}
```

Nell'ultimo articolo della serie ci occuperemo della **parte applicativa** e della **pipeline di CD/CI**, e vedremo come **integrare Cognito in API Gateway** per beneficiare dell'autenticazione automatica.

Stay tuned! 😊

[Leggi la prima parte](#) | [Leggi la terza parte](#)



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

Copyright © 2011-2021 by beSharp srl - P.IVA IT02415160189