

GO SERVERLESS! PART 2: LET'S SET UP THE CLOUD INFRASTRUCTURE FOR THE FILE-SHARING APPLICATION

AWS Lambda

CI/CD

Continuous Delivery

DevOps



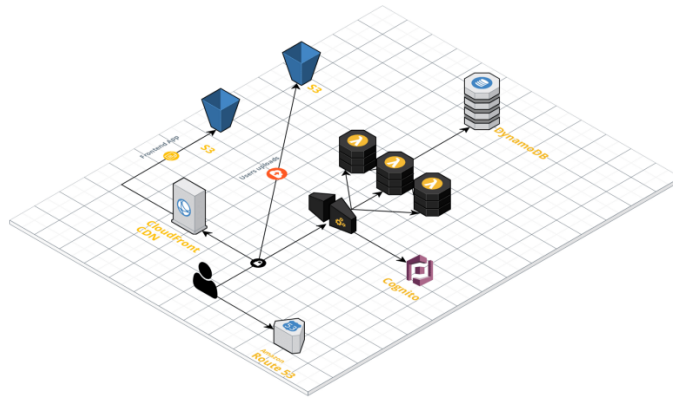
beSharp | 25 September 2018

[Go to part 1](#) | [Go to part 3](#)

The serverless approach is becoming more and more popular in the development world. But how can we actually take advantage of it to create **auto-scalable** and **resilient** applications? That's the point we are focusing on in this 3-article series explaining how to set up a **serverless File Sharing platform**.

In particular, in this article we're setting up the infrastructure for our application, focusing on the best practices to follow for a correct and effective implementation. Moreover, as revealed in our [first article](#), we are also realizing Continuous Integration and Continuous Delivery pipelines.

Let's start with the application infrastructure.



As we aren't analyzing code or other specific development subjects for now (a deep dive into them will be subject of our third article), we are going to use static pages and a lambda function to test the infrastructure we are about to create.

As shown in the graphic representation below, our future infrastructure will be composed by a **front-end**—a single page app deployed on Amazon S3 and served through **CloudFront**—and a **back-end**—a REST API built using **API Gateway** and **Lambda**.

To create the infrastructure for our serverless application, we first need to provide and set-up the following resources:

For the front-end

- A private S3 bucket
- A CloudFront distribution with permissions attached allowing it to access the private S3 bucket

For the back-end

- A lambda function to test the integration
- A Cognito User Pool
- A private S3 bucket for user's uploads
- A DynamoDB table
- An API (made with AWS API Gateway)

Front-end setup

First of all, let's create the S3 bucket we'll upload the javascript and HTML application. Note that this bucket will also act as a source for CloudFront distribution. You can set it as "private" because we'll configure the distribution so that CloudFront can access to it. We will learn how to do so later in this article.

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

✓ Name and region

✓ Configure options

3 Set permissions

4 Review

Manage users

User ID ⓘ	Objects ⓘ	Object permissions ⓘ
info@esempio.com	<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

We now need to verify that the front-end works appropriately; let's upload an example file.

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

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

Name the file “*index.html*” and upload it to the just created S3 bucket.

The image displays two sequential screenshots of the AWS S3 Upload wizard interface.

Top Screenshot (Step 1: Select files):

- Progress bar:** 1 Select files, 2 Set permissions, 3 Set properties, 4 Review.
- Summary:** 1 Files, Size: 139.0 B, Target path: fileshare-example.
- File list:** A table showing the selected file:

File icon	File name	Size	Remove
	index.html	139.0 B	X
- Buttons:** Upload, Next.

Bottom Screenshot (Step 2: Set permissions):

- Progress bar:** 1 Select files, 2 Set permissions, 3 Set properties, 4 Review.
- Summary:** 1 Files, Size: 139.0 B, Target path: fileshare-example.
- Manage users:** A table for user permissions:

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:** A button to '+ Add account'.
- Manage public permissions:** A dropdown menu with the option 'Do not grant public read access to this object(s) (Recommended)'.
- Buttons:** Upload, Previous, Next.

Once the bucket is set up, we are ready to create the **CloudFront distribution** using it as a source. In this way, we will be able to distribute the file we’ve just uploaded (not yet accessible!).

When configuring the CloudFront distribution, it is essential to enable the “**Restrict Bucket Access**” option to keep the bucket private.

Another important thing to do is to configure the “**Origin Access Identity**” option so that CloudFront can access the bucket created, even if it’s private. It is possible to delegate to the wizard both the creation of an Access Identity and the Bucket Policy update.

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, provide it here. For example, myawsbucketproduction. 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	<table border="0"> <tr> <th>Header Name</th> <th>Value</th> </tr> <tr> <td><input type="text"/></td> <td><input type="text"/></td> </tr> </table>	Header Name	Value	<input type="text"/>	<input type="text"/>	
Header Name	Value					
<input type="text"/>	<input type="text"/>					

Regarding **caching behavior**, activate HTTP redirect to **HTTPS** and enable automatic compression of the delivered content. By doing so, you will be able to improve performances.

Default Cache Behavior Settings

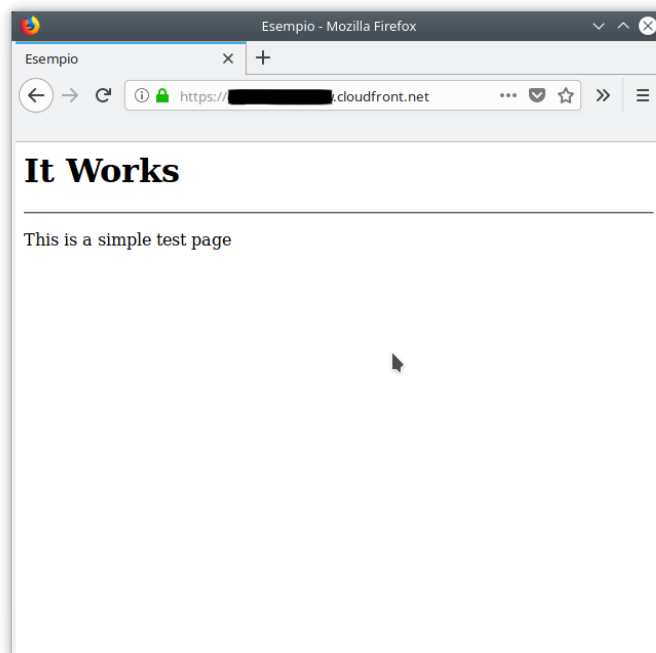
Path Pattern	<input type="text" value="Default (*)"/>	
Viewer Protocol Policy	<input type="radio"/> HTTP and HTTPS <input checked="" type="radio"/> Redirect HTTP to HTTPS <input type="radio"/> HTTPS Only	
Allowed HTTP Methods	<input type="radio"/> GET, HEAD <input checked="" type="radio"/> GET, HEAD, OPTIONS <input type="radio"/> GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE	
Field-level Encryption Config	<input type="text" value="None"/>	
Cached HTTP Methods	<input type="radio"/> GET, HEAD (Cached by default) <input checked="" type="radio"/> 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	

Don't forget to set "*index.html*" as the default file to be served.

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"/>	

The distribution creation process is usually slow, so don't panic if the process takes several minutes to complete. Note that even if the distribution has been launched, it is frequent to encounter access errors. Just don't worry about that, they usually resolve in minutes (max. 1 hour): this kind of errors are due to the fact that DNS propagation usually takes time.

Once the distribution is ready, a public URL will be available. By using it, it will be possible to check the front-end is actually working. If everything has been properly configured, something similar to the picture below will be displayed:



We are at the end of the front-end setup.

The S3 bucket is served in the most efficient and cost-effective way through a CDN. To update the application, the only action required is to upload the new version to the S3 bucket.

Please note that if updating files already saved to the bucket, you won't be able to see any change, unless you first invalidate the cache of the distribution created.

Back-end setup

Let's go deeper into the infrastructure back-end setup and creation.

The first thing we need to do is **creating a lambda function**. It will come in handy for testing by returning a "success" status and a given message.

As already said, it is not the time to focus on the real application yet, we first need to create each part of the infrastructure. So, be patient till the next article 😊

Let's go on with lambda function creation. First of all, we need to set up Python as Runtime and to define a role with basic permissions.

A screenshot of the AWS Lambda console 'Crea da zero' (Create from scratch) form. The form has the following fields: 'Nome' (Name) with the value 'FSTest', 'Runtime' with the value 'Python 3.6', 'Ruolo' (Role) with the value 'Seleziona un ruolo esistente' (Select an existing role), and 'Ruolo esistente' (Existing role) with the value 'lambda_basic_execution'. There are also links for 'Informazioni' (Information) and 'Ulteriori informazioni sui ruoli di esecuzione Lambda' (Learn more about Lambda execution roles).

Here is an example of a function returning a message:

```
def lambda_handler(event, context):  
  
    response = {  
        "statusCode": 200,  
        "body": 'Hello from Lambda!'
```

```
}
```

```
return response
```

Let's now test the function: the output coming out from a working function is similar to the example below:

The next step for us is to take care of the **provisioning of the resources** left.

Let's create another S3 bucket. This will be the place where users' files will be uploaded. Let's create it in the same way we created the bucket needed for the front-end setup. Let's keep it private, as above. Don't forget to take note of the bucket name. We will need it later while configuring the application.

Create now a **DynamoDB table**. It will contain share metadata and information about the files. Keep all the default values during the table creation and specify "shareID" as the primary partition key.

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 dall'ultima riduzione	-

It's now time to set up Cognito to prepare the authentication.

Let's start creating and configuring a **Cognito User Pool**.

The wizard allows us to customize some useful behaviors; to get a working app, we need a Pool where users can sign in through an email. Email address will be verified through the Cognito secure code.

Last but not least: creating a client app. Remember to take note of its ID!

Name first the User Pool and launch the wizard

The screenshot shows the 'Create a user pool' wizard in the AWS IAM console. On the left is a sidebar with navigation links: Name, Attributes, Policies, MFA and verifications, Message customizations, Tags, Devices, App clients, Triggers, and Review. The main area is titled 'Create a user pool' and has a 'Cancel' button in the top right. It is divided into two sections. The first section, 'What do you want to name your user pool?', includes a sub-instruction 'Give your user pool a descriptive name so you can easily identify it in the future.' and a 'Pool name' input field with a 'Required' label. The second section, 'How do you want to create your user pool?', contains two buttons: 'Review defaults' (with subtext 'Start by reviewing the defaults and then customize as desired') and 'Step through settings' (with subtext 'Step through each setting to make your choices'). A mouse cursor is pointing at the 'Step through settings' button.

Configure then the authentication process and the profile fields.

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"/>	username
<input type="checkbox"/>	locale	<input type="checkbox"/>	updated at
<input type="checkbox"/>	middle name	<input type="checkbox"/>	website
<input checked="" type="checkbox"/>	name		

It's time to create the client app taking note of its ID. Note that we don't need to generate secrets. This is not required when using Cognito through a 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](#)

FsPool

General settings

Users and groups
Attributes
Policies
MFA and verifications
Advanced security
Message customizations
Tags
Devices
App clients
Triggers
Review

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 id
f5p0h1f0c0v1u1huc0g0

[Show Details](#)

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

Cognito is now correctly configured.

The last building block we need for our infrastructure setup is an **API** built through API Gateway. It allows the front-end to access to lambda back-end functionalities thanks to HTTPS requests.

It acts as an interface between user requests and the lambda function we created. Moreover, it automatically manages requests authentication making use of Cognito User Pool.

As we only need it for testing reasons, let's create a simple API containing only a **GET method** triggering the lambda function.

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 ☒ ⓘ

The integration is completed and ready to be tested

Request: /
Status: 200
Latency: 212 ms
Response Body

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

Response Headers

```
{
  "X-Amzn-Trace-Id": "Root=1-5ba26c3d-4f9c0b03585816ab2fe6646d;Sampled=0",
  "Content-Type": "application/json"
}
```


In our next article, we are going to finally focus on the application and the CD/CI pipeline. We will learn how to **integrate Cognito with API Gateway** so that we can take advantage of the automatic authentication.

Stay tuned! 🤖

[Go to part 1](#) | [Go to part 3](#)



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