Guide de déploiement complet

Introduction

Ce guide détaille étape par étape le déploiement complet du projet **Blogify** sur AWS, en utilisant le **Serverless Framework**. Il couvre la configuration de l'authentification (AWS Cognito), le déploiement des microservices (auth-service, posts-service, media-service), la mise en place d'une documentation globale Swagger et l'utilisation de Swagger pour tester l'ensemble des endpoints (y compris les endpoints protégés par JWT).

0.1 Prérequis

— Node.js 18+ (et npm)
— Serverless Framework (installer globalement : `npm install -g serverless`)
— AWS CLI configuré (utilisateur IAM avec droits de déploiement)
— Un compte AWS actif (région : `eu-west-3` recommandée)
— Un éditeur de texte / VS code
— Swagger UI local ou service `docs-service` pour héberger la doc

0.2 1. Configuration AWS (Cognito)

0.2.1 1.1 Créer un User Pool Cognito

1. Aller sur la console AWS `Cognito` → `Manage User Pools`.
2. Cliquer sur **Create a user pool**.
3. Nommer le pool, par exemple : `blogify-users`.
4. Dans les options, choisir que l'authentification se fasse par **email**.
5. Laisser l'option "Allow sign up" active si tu veux permettre l'auto-inscription.
6. Créer le pool et noter l'**User Pool ID** (ex : `eu-west-3_gWu9ypjZR`).

0.2.2 1.2 Créer un App Client (Client ID)

1. Dans le User Pool, aller dans `App clients`.
2. Cliquer sur **Add an app client**.
3. Créer le client sans secret (pour les flows côté navigateur / Swagger).
4. Noter le **App client id** (ex : `70365...`).

0.2.3 1.3 (Optionnel) Hosted UI / Domain

— Dans `App integration` → `Domain name`, configurer un domaine Cognito (utile si tu veux tester la page de login hébergée).

0.3 2. Préparer les services (arborescence)

Structure recommandée du projet :

```
blogify/
        auth-service/
        posts-service/
        media-service/
        docs-service/      # service qui sert swagger.yaml et swagger-
    ↪ ui
```

0.4 3. Fichiers essentiels

0.4.1 3.1 Auth-service

Ton `auth-service` (lambda) doit exposer 3 routes :

— `POST /auth/signup` → appelle Cognito `SignUp`

— `POST /auth/confirm` → appelle Cognito `ConfirmSignUp`

— `POST /auth/login` → appelle Cognito `InitiateAuth` (USER_PASSWORD_AUTH) et retourne les tokens

0.4.2 3.2 Posts-service

Endpoints :

— `GET /posts` (public)

— `POST /posts` (protégé JWT)

— `GET /posts/{id}` (public)

— `PUT /posts/{id}` (protégé JWT)

— `DELETE /posts/{id}` (protégé JWT)

0.4.3 3.3 Media-service

Endpoints :

— `POST /media/upload-url` (protégé JWT) → retourne `uploadUrl` et `fileKey`

— `GET /media/download-url?key=` (public) → retourne `downloadUrl`

0.4.4 3.4 Docs (Swagger global)

Docs-service sert le fichier `swagger.yaml` et la page Swagger UI à `/docs`. Il peut être un simple petit service Express qui sert les fichiers statiques.

Exemple minimal (concept) : `docs-service/handler.js` qui renvoie HTML de Swagger UI et charge `swagger.yaml`.

0.5 4. Déploiement étape par étape

0.5.1 4.1 Configurer AWS CLI

```
aws configure
# region -> eu-west-3
# format -> json
```

0.5.2 4.2 Déployer auth-service

```
cd auth-service
serverless deploy
```

Note : après déploiement, tu auras les endpoints : `POST https://<auth-id>.execute-api.eu-west-3.a` etc.

0.5.3 4.3 Déployer posts-service

```
cd ../posts-service
serverless deploy
```

0.5.4 4.4 Déployer media-service

```
cd ../media-service
serverless deploy
```

0.5.5 4.5 Test avec Swagger

```
cd ../docs-service
npm i
npm start
```

0.6 5. Préparer Swagger (global)

1. Placer `swagger.yaml` (global) dans `docs-service/`.
2. Dans `servers` du YAML, mettre les trois URLs des services (auth, posts, media) ou utiliser un seul domaine si tu as un API Gateway central.
3. Configurer le `securityScheme` pour JWT (type : http, scheme : bearer).
4. Exposer la page Swagger UI via `/docs` (docs).

0.7 6. Utilisation de Swagger pour tester (AUTH + Endpoints protégés)

0.7.1 6.1 Récupérer un token via Swagger (auth-service)

1. Ouvrir la page Swagger : `https://port/docs`

2. Localiser la section `/auth/login`
3. Cliquer **Try it out**, renseigner `email` et `password` et exécuter.
4. Swagger renvoie `idToken` / `accessToken` / `refreshToken`.

0.7.2 6.2 S'authentifier dans Swagger (Authorize)

1. Cliquer sur le bouton **Authorize** (coin droit en haut dans Swagger UI).
2. Coller la valeur : `Bearer <idToken>` (ou `Bearer <accessToken>` selon l'authorizer).
3. Valider — maintenant les endpoints protégés acceptent les requêtes depuis Swagger.

0.7.3 6.3 Tester /posts

1. Appeler `GET /posts` (public) — tu dois voir la liste.
2. Appeler `POST /posts` (protégé) — remplir le body (JSON) → exécuter.
3. Vérifier la création dans DynamoDB (console).

0.7.4 6.4 Tester la génération d'URL d'upload (media)

1. Appeler `POST /media/upload-url` via Swagger (après Authorize).
2. Body exemple :

```
{
  "filename": "photo.jpg",
  "contentType": "image/jpeg"
}
```

3. La réponse contient `uploadUrl` et `fileKey`.

0.8 7. Upload réel du fichier (après avoir obtenu uploadUrl)

0.8.1 7.1 Via curl (terminal)

```
curl -X PUT "<uploadUrl>" \
  -H "Content-Type: image/jpeg" \
  --data-binary "@./assets/photo.jpg"
```

Important : n'ajoute pas d'en-têtes Authorization pour l'upload direct vers l'URL présignée : l'URL contient déjà la signature.

0.8.2 7.2 Vérification

— Aller dans la console S3 → bucket correspondant.
— Vérifier la présence du fichier (fileKey).
— Ou utiliser `GET /media/download-url?key=<fileKey>` via Swagger pour obtenir une URL de téléchargement si implémentée.

0.9 8. Points spécifiques sur CORS et Swagger

— Swagger UI fonctionne dans le navigateur et subit les règles CORS.
— Solution propre : **docs-service** (proxy) qui relaie les requêtes vers les services avec l'Authorization injecté, ou configurer correctement CORS sur chaque HTTP API (Serverless v4 : `httpApi.cors.allowedOrigins ...`).
— Pour tester depuis Swagger en direct (sans proxy), assure-toi que :
  — API Gateway HTTP API a bien le CORS activé pour `OPTIONS, GET, POST, PUT, DELETE`.
  — Les en-têtes `Authorization`, `Content-Type` sont autorisés.

0.10 9. Vérifications post-déploiement

1. Vérifier que les endpoints retournent 200/201 selon les cas.
2. Vérifier CloudWatch Logs si une Lambda renvoie une erreur 500.
3. Vérifier les permissions IAM (Lambda -> S3 / DynamoDB) si access denied.
4. Vérifier la création des ressources (DynamoDB / S3) dans la console.

0.11 10. Rollback / Nettoyage

```
# supprimer un service Serverless
cd posts-service
serverless remove

cd ../media-service
serverless remove

cd ../auth-service
serverless remove
```

puis supprimer manuellement :

— User Pool Cognito (si tu veux)
— Buckets S3 (vider puis supprimer)
— Tables DynamoDB

Annexe — Exemples utiles

A. Exemple de body JSON pour POST /posts

```
{
  "title": "Mon premier article",
  "content": "Voici le contenu du post",
  "imageUrl": "https://s3.../users/abcd/photo.jpg"
}
```

B. Exemple de réponse pour POST /media/upload-url

```
{
  "uploadUrl": "https://bucket.s3.eu-west-3.amazonaws.com/users/
    ↪ abcd/photo.jpg?X-Amz-...",
  "fileKey": "users/abcd/photo.jpg"
}
```

Conclusion

Ce guide décrit une procédure de test entièrement centralisée autour de Swagger UI (fournie par le service `docs-service`). Le flux d'authentification est géré correctement grâce à Cognito : on récupère d'abord un jeton via l'endpoint `/auth/login`, puis on l'injecte dans Swagger à l'aide du bouton *Authorize*. Pour les envois de fichiers binaires, on utilise ensuite l'`uploadUrl` renvoyée par l'API afin d'effectuer un `PUT` direct vers S3 (avec `curl` ou tout autre client HTTP), sans inclure d'en-tête Authorization.