

Projet Application Distribuée & Web services

Sujet : Application Spitch

TANNIER Yannis
BEN HAMDOUNE Mohamed

Table des matières

1.	Présentation	3
1.	1. Présentation générale	3
1.	2. Stack Technique.....	3
2.	2. Utilisation de l'application	4
3.	3. Base de données	6
4.	4. Manuel de démarrage	7
1.	1. Création des accès AWS et Installation aws-cli	7
2.	2. Création de l'architecture via CloudFormation.....	8
3.	3. Création de la base de données via RDS.....	9
4.	4. Déploiement des applications.....	11
1.	1. Django sur ElasticBeanstalk	11
2.	2. Spring sur Docker.....	14
5.	5. Mise en route de React Native	15
4.	4. Test et accès	17
5.	5. Conclusion.....	18
6.	6. Annexe	19
1.	1. Annexe 1.....	19
2.	2. Annexe 2.....	19
3.	3. Annexe 3.....	20
4.	4. Annexe 4.....	20
5.	5. Annexe 5.....	21

1. Présentation

1. Présentation générale

Spitch est une application de questions/réponses en vidéo permettant aux utilisateurs de découvrir, créer et partager des contenus vidéos. L'application est présente sur **Android** et sur **iPhone**.

2. Stack Technique

Voici la Stack Technique de **Spitch** :

- *Django (Python)*
- *Spring (Jee)*
- *Docker*
- *AWS Elastic Beanstalk*
- *AWS CloudFormation*
- *AWS Lambda*
- *AWS DynamoDB*
- *AWS SQS*
- *AWS Transcoder*
- *AWS StepFunction*
- *AWS S3*
- *React Native*

L'architecture de **Spitch** a été pensée pour un hébergement total dans le **Cloud AWS**, ainsi beaucoup de ressources **AWS** ont été utilisé.

Tout d'abord notre base de données est hébergée sur **AWS RDS**.

Notre Backend principale, développée en *Python* avec **Django**, est hébergée sur **Elastic Beanstalk (PaaS)** et fournit toutes les APIs Rest utilisées par le client React Native.

Nous avons développé un deuxième Backend en java avec Spring afin de respecter les conditions de ce projet, ce Backend permet d'authentifier un utilisateur et de retourner un jeton d'authentification (appelé communément **Token**). Cette application **Spring** est hébergée via **Docker** sur **AWS**.

Nos tâches asynchrones sont gérées via un **Bus SQS**. Toutes les tâches longues sont ainsi envoyées dans le bus SQS qui est écouté en permanence par un **Worker (Elastic Beanstalk)** se chargeant de récupérer et traiter toutes les tâches asynchrones.

De nombreuses fonctions lambda sont aussi utilisées, permettant ainsi de décharger une grosse partie du travail de nos Backends. Ces fonctions lambdas sont des services « **ServerLess** » proposés par **AWS**, signifiant que nous avons besoin d'aucun serveur pour utiliser ces fonctions lambdas.

Ces fonctions lambdas servent principalement aux traitements des images, des vidéos, des notifications et des envois d'email. (Quelques exemples de code lambdas sont en Annexe).

Nous utilisons également deux bases de données **NoSQL DynamoDB** :

- Pour le stockage des notifications de chaque utilisateur. Cette base de données DynamoDB est reliée à une fonction lambda (DynamoDB Streams, Lambda Trigger) afin de pouvoir envoyer une notification push si besoin à chaque utilisateur après chaque nouvel enregistrement en base de données.
- Une autre base de données **DynamoDB** est utilisée pour stocker le résultat de la reconnaissance d'images via Rekognization. Cela nous permet de faire une modération automatique afin de détecter les contenus interdit. À titre d'illustration, des contenus pornographiques et bien d'autres.

Enfin, tout notre architecture a été créé via **CloudFormation** qui fournit un langage commun pour décrire et provisionner toutes les ressources d'infrastructure dans votre environnement cloud. Cela nous permet de cloner l'intégralité de notre architecture en un instant. La template **CloudFormation** est accessible sur le lien **GitHub** plus bas.

L'intégralité des codes de l'architecture sont accessibles ici :
<https://github.com/yannistannier/descartes-projet-applidist>

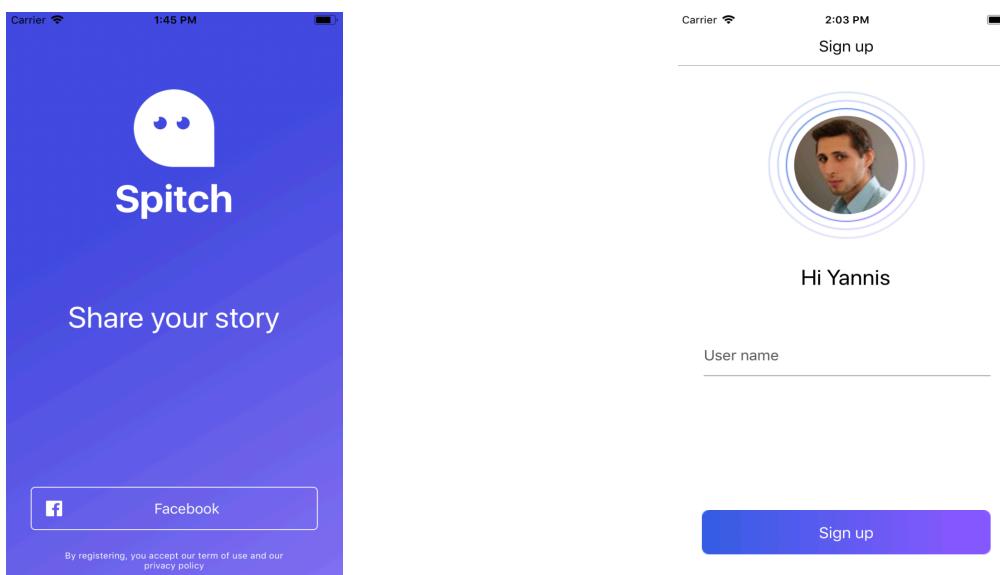
2. Utilisation de l'application

Au premier lancement de l'application, nous tombons sur la page d'accueil.

À cet instant, l'inscription et la connexion via Facebook est disponible pour faciliter l'utilisation de l'application.

Une fois que l'utilisateur s'est authentifié avec Facebook, il arrive sur la page permettant de créer son nom d'utilisateur.

Cette page est affichée uniquement à la première connexion sur l'application.



Une fois l'inscription terminée, nous tombons sur les 3 pages principales de l'application :

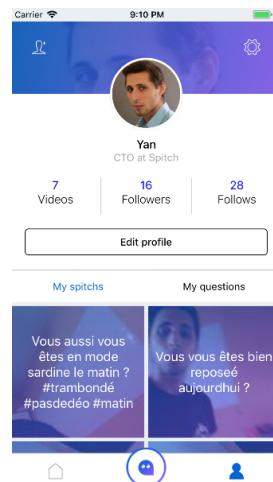
Home



Spitch



Profil



L'application s'articule autour de ces 3 pages :

1. La première correspond au flux d'actualité, sur cette page nous pouvons voir toutes les vidéos que nos amis ont effectuées.
2. La deuxième page, nommée **Spitch**, permet de voir toutes les questions que la communauté a posées pour pouvoir y répondre en vidéo si on voit une question qui nous intéresse.
3. La troisième correspond à la page profil de l'utilisateur qui répertorie toutes les vidéos et les questions de l'utilisateur.

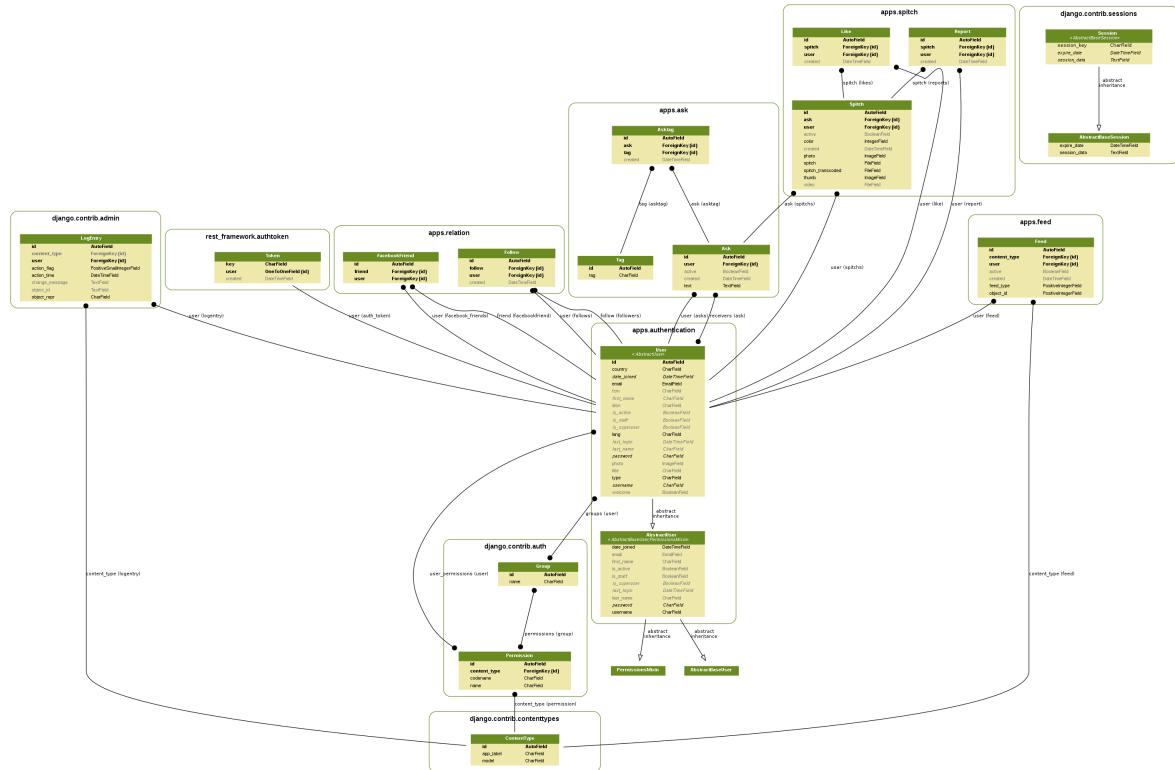


Quand nous cliquons sur une vidéo depuis le flux d'actualité ou la page profil, nous arrivons sur la page « Vidéo » qui reprend la question à laquelle l'utilisateur a répondu ainsi que le nombre de Like et l'accès au profil de la personne qui y répond.

3. Base de données

Notre base de données utilise **PostgreSQL** hébergée sur **AWS RDS**, le service de gestion de base de données relationnel d'**AWS**.

Voici le **diagramme UML**¹ de la base de données : [lien](#)



Notre base de données s'articule autour du modèle User qui regroupe toutes les données de l'utilisateur. Le modèle User étant le modèle généré par **Django**, nous avons créé le modèle AbstractUser afin d'ajouter des champs supplémentaires qui hérite d'User.

Le modèle User est relié par plusieurs relation OneToMany :

- Vers le modèle FacebookFriend et Follow, qui sert à stocker les relations d'amitié entre les utilisateurs.
- Vers le modèle Ask qui permet de stocker toutes les questions posées sur l'application.
- Vers le modèle Spitch qui conserve toutes les vidéos faites par les utilisateurs, ainsi que vers le Modèle Like et Report.
- Et enfin vers le modèle Feed qui permet de stocker le fil d'actualité de chaque utilisateur.

¹ Il est conseillé de voir l'UML à partir du lien donné afin de pouvoir zoomer sur les tables.

Le modèle Ask est relié par une relation ManyToMany avec Tag (une table intermédiaire AskTag est créée).

Les modèles Permission, Groupe, Session, ContentType et Log Entry sont des modèles générés automatiquement par **Django**.

Permissions et Groupe permettent de gérer le niveau de permission de chaque utilisateur (par exemple pour l'accès à l'administrateur).

ContentType est utilisé dans le cas de relation générique, c'est-à-dire d'un champ pouvant être relier à plusieurs modèles différent.

4. Manuel de démarrage

1. Création des accès AWS et Installation aws-cli

Avant de démarrer le manuel de démarrage, il convient de créer un compte AWS et de se créer un User IAM².

Pour cela il faut aller dans IAM sur l'interface <https://console.aws.amazon.com>, dans l'onglet « User » et cliquer sur « Add User ».

The screenshot shows the 'Set user details' step of the AWS IAM 'Add User' wizard. It includes fields for 'User name*' (set to 'projet-web'), an 'Add another user' link, and a 'Select AWS access type' section. In the 'Access type*' dropdown, 'Programmatic access' is selected (indicated by a checked checkbox). Below it, there's a note about enabling access keys and secret access keys for AWS API, CLI, SDK, and other development tools. There are also options for 'AWS Management Console access' (unchecked).

Sélectionner « Programmatic access » puis dans Next: Permissions, « Attach existing policies directly » choisir le nom de la politique :

- [AdministratorAccess](#)

Attention, cet accès représente un gros risque de sécurité car il donne l'accès total à toutes les ressource d'AWS. Pour éviter ça il est conseillé de créer des politiques de sécurité plus précise autorisant uniquement les actions sur les ressources que nous avons besoin. Mais dans le cadre du rapport et pour ne pas trop compliquer le manuel de démarrage, nous allons utiliser la politique AdministratorAccess.

² AWS Identity and Access Management (IAM) permet de contrôler de façon sécurisée l'accès aux services et ressources AWS.

Ensuite il suffit de télécharger l'Access Key ID et le Secret Access Key généré par le service pour pouvoir configurer l'interface de commande aws sur la machine.

Pour installer l'interface de commande AWS, on peut utiliser pip via la commande :

```
pip install awscli awsebcli
```

Une fois l'installation fini, tapez la commande suivante pour configuration nos identifiants :

```
aws configure
```

Et saisissez l'Access Key et le Secret Key téléchargé plus haut, mettre « eu-west-1 » dans la case « region_name ».

Ensuite pour la case « Default output format », vous pouvez appuyer sur entrer, la valeur par défaut est en JSON.

Et voilà, notre machine locale est prête pour déployer l'application.

2. Création de l'architecture via CloudFormation

Cette étape est importante car elle permet de déployer l'architecture de Spitch.

Pour ce faire, télécharger le fichier template **CloudFormation** : [lien](#)

Puis exécuter la création de la stack **CloudFormation** via la commande :

```
aws cloudformation create-stack --region eu-west-1 --template-body file://spitch.template.yaml --capabilities CAPABILITY_IAM CAPABILITY_NAMED_IAM --stack-name spitchtv
```

Cette stack **CloudFormation** permet de créer de nombreuses ressources **AWS** utilisé par l'application :

- Le bucket S3 ou les vidéos / images / fichiers statiques seront stockés.
- Les bases de données **DynamoDB** pour les notifications et la reconnaissance d'image.
- Les fonctions lambdas permettant de traiter les vidéos, envoyer les notifications, exécuter la reconnaissance d'images puis l'envoies d'email.
- Les règles CRON et les règles de flux sur les buckets **S3** et la base de données **DynamoDB**.
- Les politiques ainsi que l'IAM USER que l'application utilisera.
- Les files d'attente SQS pour les tâches asynchrones.

Une fois que la création finie, retournez sur l'interface **AWS**, dans **Cloudformation** :

The screenshot shows the AWS CloudFormation console. At the top, there is a search bar with the filter set to 'Active' and the search term 'spitchtest'. Below the search bar is a table with four columns: Stack Name, Created Time, Status, and Description. A single row is selected, showing 'spitchtest' as the Stack Name, '2018-03-03 02:52:03 UTC+0100' as the Created Time, 'CREATE_COMPLETE' as the Status, and 'Cloudformation Stack dev Sans base de donnee' as the Description. Below the table is a navigation bar with tabs: Overview, Outputs (which is selected), Resources, Events, Template, Parameters, Tags, Stack Policy, Change Sets, and Rollback Triggers. Under the 'Outputs' tab, there is another table with three columns: Key, Value, and Description. The outputs listed are: SecretKey (value: tQIZixWM53VpDC3crSWs90pwloh2rt7bSnS18PFK), DynamodbNotification (value: spitchtest-tableNotification-KIU4B5L9WMHI), BucketName (value: spitchtest-bucket-16c4q1u15xqh), AccessKey (value: AKIAIFDYHPFZWX3S2FTA), and SqsWorker (value: <https://sns.eu-west-1.amazonaws.com/930255213805/spitc> htest-sqsWorker-1X4VHAUSDEKZL).

Stack Name	Created Time	Status	Description
spitchtest	2018-03-03 02:52:03 UTC+0100	CREATE_COMPLETE	Cloudformation Stack dev Sans base de donnee

Key	Value	Description
SecretKey	tQIZixWM53VpDC3crSWs90pwloh2rt7bSnS18PFK	
DynamodbNotification	spitchtest-tableNotification-KIU4B5L9WMHI	
BucketName	spitchtest-bucket-16c4q1u15xqh	
AccessKey	AKIAIFDYHPFZWX3S2FTA	
SqsWorker	https://sns.eu-west-1.amazonaws.com/930255213805/spitc htest-sqsWorker-1X4VHAUSDEKZL	

Sélectionner le stack créé, puis « output » pour récupérer L'Access Key et le Secret Key du User IAM crée ainsi que le bucket name et l'URL **SQSWorker**³.

3. Création de la base de données via RDS

Pour créer notre base de données, nous utilisons l'interface de commande AWS.
Pour cela, tapez la commande :

```
aws rds create-db-instance --db-instance-identifier spitchinstance --allocated-storage 20 --db-instance-class db.t2.micro --engine postgres --master-username masterawsuser --master-user-password masteruserpassword --db-name spitchdatabase --no-multi-az --no-auto-minor-version-upgrade
```

³ Le paramètre affiche l'URL associée avec cette file d'attente Amazon SQS (Simple Queue Service).

Une fois la création terminée, nous devons récupérer l'endpoint de la base de données, pour cela il faut se rendre sur l'interface **AWS** dans **Relational Database Service**, l'onglet « Details » de notre instance de base de données :

Details			
Configurations	Security and network	Instance and IOPS	Maintenance details
ARN arn:aws:rds:eu-west-1:930255213805:db:spitchinstance	Availability zone eu-west-1a	Instance Class db.t2.micro	Auto minor version upgrade No
Engine PostgreSQL 9.6.6	VPC vpc-c25bfca4	Storage Type Magnetic	Maintenance window tue:22:19-tue:22:49 UTC (GMT)
License Model Postgresql License	Subnet group default	Storage 20 GB	Backup window 00:31-01:01 UTC (GMT)
Created Time Sat Mar 03 13:48:51 GMT+100 2018	Subnets subnet-0b81b950 subnet-51966837 subnet-261dd06e	Availability and durability	Pending Modifications None
DB Name spitchdatabase	Security groups default (sg-ebf67091) (active)	DB instance status available	Pending maintenance none
Username masterawsuser	Publicly accessible Yes	Multi AZ No	Encryption details
Option Group default:postgres-9-6	Endpoint spitchinstance.cegyiolqgwc.eu-west-1.rds.amazonaws.com	Automated backups Enabled (1 Day)	Encryption enabled No
Parameter group default.postgres9.6 (in-sync)	Certificate authority rds-ca-2015 (Mar 5, 2020)	Latest restore time March 3, 2018 at 1:49:09 PM UTC+1	
Copy tags to snapshots No			
Resource ID db-KSWLXXAL6QLG64WJ3F37Q7XACE			

Si l'accès au endpoint ne fonctionne pas, il faut autoriser l'accès public à l'instance de base de données, pour cela cliquez sur Security Group (dans l'exemple « default(sg-ebf67091) »), sélectionnez l'onglet « Inbound » et cliquez sur « Edit » (Type : All Traffic Custom AnyWhere).

The screenshot shows the 'Details' tab of an AWS RDS instance configuration. The 'Endpoint' field is highlighted with a red box, displaying the URL: 'spitchinstance.cegyiolqgwc.eu-west-1.rds.amazonaws.com'. This URL is the public endpoint for the database instance.

Security Group: sg-ebf67091

Inbound Rules:

- Description:** sg-ebf67091
- Type:** All traffic
- Protocol:** All
- Port Range:** All
- Source:** sg-ebf67091 (default)

« Capture d'écran de la page Web »

4. Déploiement des applications

1. Django sur ElasticBeanstalk

L'application **Django** correspond au Backend de **Spitch**, elle fournit toutes les APIs Rest utilisé par le client **React Native**⁴, elle permet aussi de fournir l'interface d'administration.

Le déploiement du Backend va aussi permettre d'initialiser plusieurs choses :

- Elle va créer les tables en base de données depuis les modèles **Django**.
- Envoyer tous les fichiers statiques dans le bucket **S3** créé par **CloudFormation**.
- Créer un utilisateur administrateur afin de pouvoir se connecter à l'administration.

Le code du Backend est accessible ici :

<https://github.com/yannistannier/descartes-projet-applidist/tree/master/backend>

Avant de déployer l'application, nous devons indiquer à notre application les configurations nécessaires.

Pour cela nous utilisons des « options settings », qui permettent d'indiquer à **ElasticBeanstalk** de créer des variables d'environnements avant d'exécuter l'application.

Pour cela, modifier le fichier « environments.config » dans le dossier «.ebextensions » (Attention le dossier peut être cacher suivant votre configuration local).

Les valeurs à remplacer sont :

- [DATABASE_HOST] par le endpoint récupéré sur RDS.
- [DATABASE_DBNAME], [DATABASE_USER], [DATABASE_PASSWORD] par les valeurs rentrées au moment de la création de l'instance de base de données RDS.
- [AWS_ACCESS_KEY_ID], [AWS_SECRET_ACCESS_KEY], [BUCKET_S3], [DYNAMODB_TABLE_NOTIF] et [SQS_WORKER] par les valeurs récupérées dans le Output de CloudFormation.

⁴ React Native est un framework utilisant Javascript et React permettant la création d'application mobile native.

Pour déployer l'application **Django** sur **AWS ElasticBeanstalk**, allez dans le dossier Backend du dépôt et taper la commande suivante :

```
eb init
```

```
+ backend git:(master) ✘ eb init
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
8) ap-southeast-2 : Asia Pacific (Sydney)
9) ap-northeast-1 : Asia Pacific (Tokyo)
10) ap-northeast-2 : Asia Pacific (Seoul)
11) sa-east-1 : South America (Sao Paulo)
12) cn-north-1 : China (Beijing)
13) us-east-2 : US East (Ohio)
14) ca-central-1 : Canada (Central)
15) eu-west-2 : EU (London)
(default is 3): 4

Enter Application Name
(default is "backend"): spitchtv
Application spitchtv has been created.

It appears you are using Python. Is this correct?
(Y/n): Y

Select a platform version.
1) Python 3.6
2) Python 3.4
3) Python
4) Python 2.7
5) Python 3.4 (Preconfigured - Docker)
(default is 1):
Cannot setup CodeCommit because there is no Source Control setup, continuing with initialization
Do you want to set up SSH for your instances?
(Y/n): n
+ backend git:(master) ✘
```

« Capture d'écran d'un terminal exécutant la commande »

Cette commande va créer notre application sur **ElasticBeanstalk**⁵.

Une fois celle-ci faite, nous devons créer notre environnement, pour cela taper la commande :

```
eb create Backend --single
```

La création de l'environnement peut prendre plusieurs minutes.

⁵ Il vous suffit de charger votre code, et Elastic Beanstalk effectue automatiquement les étapes du déploiement que sont le dimensionnement des capacités, l'équilibrage de la charge et la surveillance de l'état de l'application.

Pour récupérer l'URL créée une fois le déploiement fini, il faut se rendre sur l'interface **AWS** dans **ElasticBeanstalk**, sélectionner votre application puis votre environnement et vous verrez l'URL sur la page de Dashboard⁶ :

The screenshot shows the AWS Elastic Beanstalk dashboard for the 'backend' environment. At the top, it displays the environment ID (e-n4rp2dnhhi) and URL (backend.mwnqpza6tu.eu-west-1.elasticbeanstalk.com), which is highlighted with a red box. Below this, there's an 'Overview' section with a green health status icon (OK), a 'Running Version' (app-180303_140738), and a 'Configuration' section indicating 64bit Amazon Linux 2017.09 v2.6.5 running Python 3.6. There's also a 'Upload and Deploy' button and a 'Refresh' link. The 'Recent Events' table lists several log entries from March 3, 2018, detailing the deployment process.

Time	Type	Details
2018-03-03 14:11:50 UTC+0100	INFO	Environment health has transitioned from Info to Ok. Initialization completed 59 seconds ago and took 3 minutes.
2018-03-03 14:11:40 UTC+0100	INFO	Successfully launched environment: backend
2018-03-03 14:10:50 UTC+0100	INFO	Environment health has transitioned from Pending to Info. Initialization in progress. 1 out of 1 instance completed (running for 3 minutes).
2018-03-03 14:09:50 UTC+0100	INFO	Added instance [i-0ae640dcc40d6cfb0] to your environment.
2018-03-03 14:09:07 UTC+0100	INFO	Waiting for EC2 instances to launch. This may take a few minutes.

« Capture d'écran de l'interface web sur AWS »

Pour se connecter sur le compte de l'administrateur, il suffit juste d'ajouter /admin à la fin de l'URL.

The screenshot shows the Django administration login page. It features a dark header bar with the text 'Django administration'. Below it is a form with two fields: 'Username:' containing a placeholder 'Username' and 'Password:' containing a placeholder 'Password'. At the bottom of the form is a blue 'Log in' button.

« Capture d'écran de l'interface web pour Django »

Les accès administrateurs sont :

- Login : [administrator](#)
- Mot de passe : 123456

Il est évidemment très fortement conseillé de changer ces logins dès la première connexion

Et voilà, notre API Rest et notre Backend ont été déployé sur **AWS EBS**.

⁶ Le tableau de bord de l'état personnel vous donne une vue personnalisée sur les performances et la disponibilité des services AWS.

2. Spring sur Docker

L'application **Spring** permet d'authentifier l'utilisateur, elle ne fournit qu'une seule API Rest qui retourne ou non un token d'authentification.

Cette application utilise **JPA** pour la connexion avec la base de données **PostgreSQL**, le module « data-rest » qui permet de construire une API Rest et **Docker** pour le déploiement.

Le code de l'application Spring se trouve ici :

<https://github.com/yannistannier/descartes-projet-applidist/tree/master/spring>

Comme pour l'application **Django**, avant le déploiement nous devons rentrer les bonnes valeurs pour les variables d'environnements.

Le fichier se trouve dans .ebextensions/environments.config et les éléments à changer sont :

- [DATABASE_URL] : la valeur à saisir doit être de la forme :
dbc: postgresql://endpoint:5432/database

- [DATABASE_USERNAME], [DATABASE_PASSWORD] : par les valeurs rentrées au moment de la création de l'instance de base de données **RDS**.

Pour déployer une image docker sur **AWS EBS**, rien de plus simple, il suffit juste d'aller dans le dossier spring et de taper la commande :

```
eb init
```

```
[+ spring git:(master) ✘ eb init
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
8) ap-southeast-2 : Asia Pacific (Sydney)
9) ap-northeast-1 : Asia Pacific (Tokyo)
10) ap-northeast-2 : Asia Pacific (Seoul)
11) sa-east-1 : South America (Sao Paulo)
12) cn-north-1 : China (Beijing)
13) us-east-2 : US East (Ohio)
14) ca-central-1 : Canada (Central)
15) eu-west-2 : EU (London)
(default is 3): 4

Select an application to use
1) spitchtv
2) [ Create new Application ]
(default is 2):

Enter Application Name
(default is "spring");
Application spring has been created.

It appears you are using Docker. Is this correct?
(Y/n): Y

Select a platform version.
1) Docker 17.09.1-ce
2) Docker 17.06.2-ce
3) Docker 17.03.2-ce
4) Docker 1.12.6
5) Docker 1.11.2
6) Docker 1.9.1
7) Docker 1.7.1
8) Docker 1.6.2
9) Docker 1.5.0
(default is 1):
Cannot setup CodeCommit because there is no Source Control setup, continuing with initialization
Do you want to set up SSH for your instances?
(Y/n): n
```

« Déroulement des différentes parties de la configuration

Suivi de la commande :

```
eb create
```

AWS EBS va créer automatiquement l'image docker depuis le fichier Dockerfile et va déployer l'image sur **EBS** avec les configurations saisies dans le fichier Dockerrun.aws.json

Ces deux fichiers ⁷(Dockerfile et Dockerrun.aws.json) doivent être dans le même dossier.

Une fois le déploiement terminé, vous pouvez vous rendre sur l'interface d'**AWS** dans **ElasticBeanstalk** afin de pouvoir récupérer l'URL créée.

The screenshot shows the AWS Elastic Beanstalk console for the environment 'spring-dev'. At the top, it displays the Environment ID: e-ikymd9ddh4, URL: spring-dev.eu-west-1.elasticbeanstalk.com. A red box highlights the URL. Below this, there's an 'Actions' dropdown. The main area has tabs for 'Overview', 'Health' (which shows 'Ok'), 'Running Version' (app-180303_214842), 'Upload and Deploy', 'Configuration' (64bit Amazon Linux 2017.09 v2.8.4 running Docker 17.09.1-ce), and a 'Change' button. On the left, there's a 'Recent Events' section with a 'Show All' button. The table below lists recent events:

Time	Type	Details
2018-03-03 21:52:16 UTC+0100	INFO	Environment health has transitioned from Pending to Ok. Initialization completed 33 seconds ago and took 2 minutes.
2018-03-03 21:52:00 UTC+0100	INFO	Successfully launched environment: spring-dev
2018-03-03 21:50:55 UTC+0100	INFO	Docker container 56cf66e2fc33 is running aws(beanstalk/current-app).
2018-03-03 21:50:46 UTC+0100	INFO	Successfully built aws(beanstalk/staging-app)
2018-03-03 21:50:42 UTC+0100	INFO	Successfully pulled openjdk:8-jdk-alpine

« Capture d'écran sur l'interface AWS »

5. Mise en route de React Native

Une fois le déploiement des applications **Django** et **Spring** effectué avec succès, nous pouvons enfin mettre en route l'application mobile **React Native**.

Pour cela, télécharger le code ici :

<https://github.com/yannistannier/descartes-projet-applidist/tree/master/reactnative>

⁷ Ce sont des fichiers de configuration très importante pour l'exécution de Docker.

La première chose à faire est d'installer **Node.js 8**. Vous pouvez le faire directement via <https://nodejs.org/en/> ou sous **Linux** avec les commandes suivantes :

```
curl-sL https://deb.nodesource.com/setup_8.x | sudo-Ebash-
sudo apt install -y nodejs
```

Installation de **React Native** via npm.

```
npm install -g react-native-cli
```

Une fois l'installation de node.js et de react-native fini, nous devons installer toutes les dépendances de l'application react-native.

Pour se faire, se rendre dans le dossier reactnative du projet et tapez la commande :

```
npm install
```

Si l'installation s'est effectuée sans erreur, alors vous devriez voir la création d'un dossier node_modules dans le dossier reactnative.

En supposant que vous possédez un Mac avec **Xcode** ou alors **Android Studio**, les commandes à taper pour lancer l'application en mode développement sont :

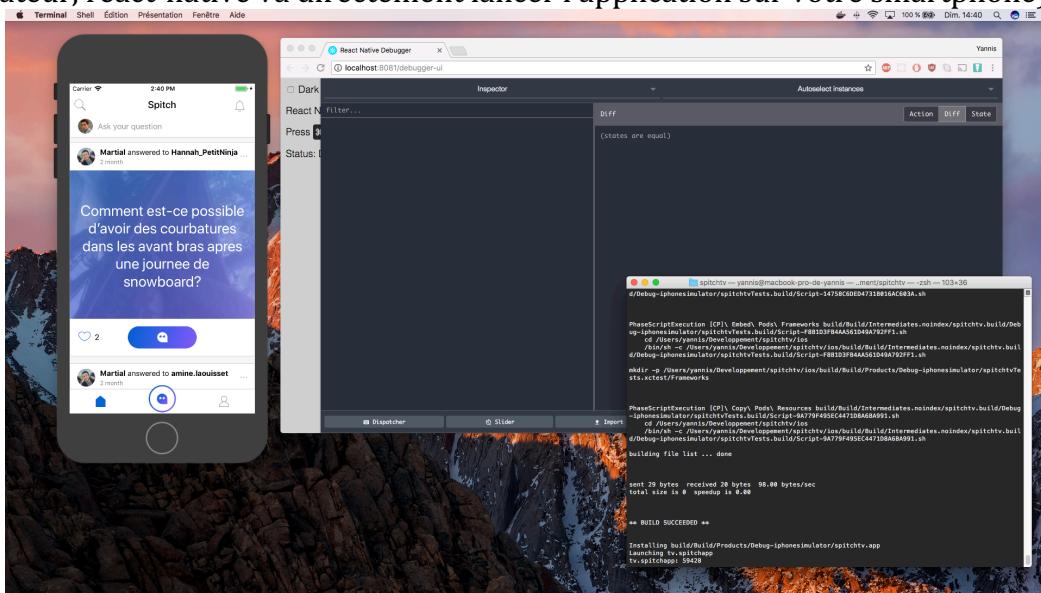
Pour exécuter l'émulateur d'IOS :

```
react-native run-ios
```

Pour exécuter l'émulateur d'Android :

```
react-native run-android
```

(Notez que si vous branchez un smartphone Android en mode développement à votre ordinateur, react-native va directement lancer l'application sur votre smartphone).

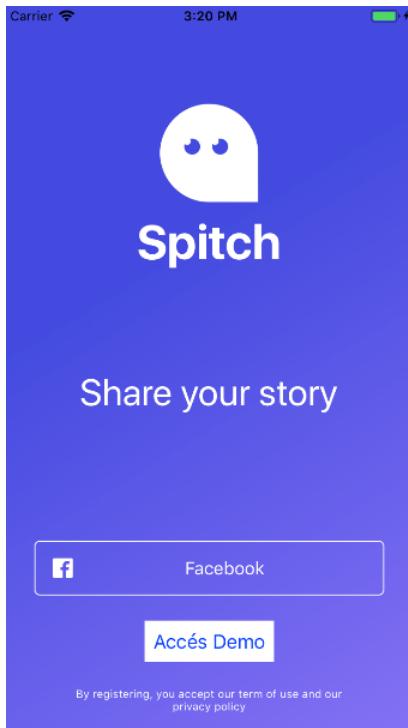


« Capture d'écran d'un émulateur sous iOS »

4. Test et accès

Pour tester l'application, vous pouvez directement télécharger la version officielle depuis les stores, elle est disponible sur **Android** et **IOS**. Vous pouvez retrouver les liens sur ce site :

<http://spitch.tv/>



Sinon vous pouvez télécharger une version de test possédant un bouton « Access Démô »⁸ permettant de pouvoir tester l'application sans devoir se connecter via Facebook et en utilisant les services créer tout au long de ce rapport.

Voici les liens pour télécharger cette version :

- Android : <https://s3-eu-west-1.amazonaws.com/spitchtv-bucket-8wwatwqktoom/app-release.apk>
- IOS : <https://s3-eu-west-1.amazonaws.com/spitchtv-bucket-8wwatwqktoom/spitchtv.ipa>

⁸ Cette option a été ajoutée afin que le correcteur puisse tester l'application sans avoir besoin de se connecter avec Facebook s'il ne souhaite pas partager ses infos.

5. Conclusion

Ce projet a été une bonne initiation au cloud et nous a permis de découvrir plusieurs ressources d'**AWS** ainsi que le concept de micro service.

En résumé, c'est un projet applicatif assez complet en terme architectural puisqu'il regroupe toutes les parties d'une application. Par la suite, il y a des parties du projet que l'on peut améliorer afin d'avoir un gain en productivité par exemple

Nous devrions améliorer le template **CloudFormation** pour automatiser la création de la base de données ainsi que le déploiement direct des applications. Et enfin la mise en place de pipeline d'intégration continue avec des services tel que CodePipeline ou CodeBuild ou bien encore CodeDeploy.

6. Annexe

spitchtv-lambdaSendNotification-1QXTA1E0DBD79

Qualifiers ▾ Actions ▾ Select a test event.. ▾ Test Save

Code entry type: Edit code inline Runtime: Python 2.7 Handler: lambda_function.lambda_handler

File Edit Find View Goto Tools Window

Environment

lambda_function

```
1 from pyfcm import FCMNotification
2 import boto3
3 import json
4 import os
5
6 def lambda_handler(event, context):
7     sqs = boto3.resource('sqs')
8     queue = sqs.get_queue_by_name(QueueName=os.environ['SQS'])
9     messages = queue.receive_messages(MaxNumberOfMessages=10, VisibilityTimeout=30)
10    notification = Notification()
11
12    for message in messages:
13        notification.send(message)
14
15    class Notification():
16        message_title = "Spitch"
17        handled = []
18
19        def __init__(self):
20            self.push_service = FCMNotification(api_key=os.environ['FCM_KEY'])
21
22        def reset(self):
23            self.body = None
24            self.message = None
25
26        def get(self, arr):
27            if 'N' in arr:
28                return int(arr['N'])
29            if 'S' in arr:
30                return arr['S']
31            if 'M' in arr:
32                return arr['M']
```

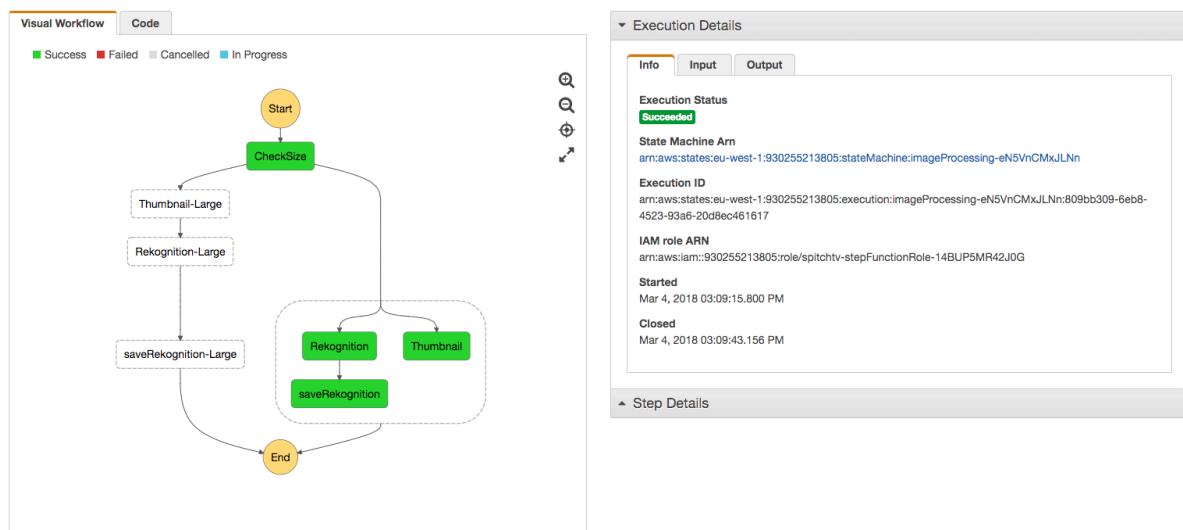
1:1 Python Tabs: 4

1. Annexe 1

« Code Python d'une entrée lambda pour les Notification »

2. Annexe 2

809bb309-6eb8-4523-93a6-20d8ec461617 ✓



« Exemple StepFunction »

3. Annexe 3

user	photo	labels
1088	media/305/s...	[{"M": {"Confidence": {"N": "53.686214447021484375"}, "Name": {"S": "File Binder"}}, {"M": {"Confidence": {"N": "53...}}
1171	media/287/s...	[{"M": {"Confidence": {"N": "99.30328369140625"}, "Name": {"S": "Human"}}, {"M": {"Confidence": {"N": "99.3032836...}}
1307	media/274/s...	[{"M": {"Confidence": {"N": "50.619792938232421875"}, "Name": {"S": "Art"}}, {"M": {"Confidence": {"N": "50.619792...}}
1307	media/274/s...	[{"M": {"Confidence": {"N": "51.156444549560546875"}, "Name": {"S": "Cushion"}}, {"M": {"Confidence": {"N": "51.15...}}
1182	media/274/s...	[{"M": {"Confidence": {"N": "99.0120086669921875"}, "Name": {"S": "Human"}}, {"M": {"Confidence": {"N": "99.01198...}}
797	media/231/s...	[{"M": {"Confidence": {"N": "83.2015308059375"}, "Name": {"S": "Electronics"}}, {"M": {"Confidence": {"N": "83.20153...}}
797	media/231/s...	[{"M": {"Confidence": {"N": "90.6044158935546875"}, "Name": {"S": "Computer"}}, {"M": {"Confidence": {"N": "90.604...}}
846	media/231/s...	[{"M": {"Confidence": {"N": "76.9631500244140625"}, "Name": {"S": "Wall"}}, {"M": {"Confidence": {"N": "55.5303459...}}
675	media/221/s...	[{"M": {"Confidence": {"N": "68.90988922119140625"}, "Name": {"S": "Screen"}}, {"M": {"Confidence": {"N": "51.0197...}}
231	media/231/p...	[{"M": {"Confidence": {"N": "89.62200164794921875"}, "Name": {"S": "Sunlight"}}, {"M": {"Confidence": {"N": "89.346...}}
231	media/231/p...	[{"M": {"Confidence": {"N": "99.2953033447265625"}, "Name": {"S": "Human"}}, {"M": {"Confidence": {"N": "99.29530...}}
231	media/231/p...	[{"M": {"Confidence": {"N": "99.31928253179828125"}, "Name": {"S": "Human"}}, {"M": {"Confidence": {"N": "99.3192...}}
231	media/231/p...	[{"M": {"Confidence": {"N": "99.21067047119140625"}, "Name": {"S": "Human"}}, {"M": {"Confidence": {"N": "99.2106...}}
231	media/231/p...	[{"M": {"Confidence": {"N": "99.0398406982421875"}, "Name": {"S": "Human"}}, {"M": {"Confidence": {"N": "99.03982...}}

« Exemple de table Reckognition avec DynamoDB »

4. Annexe 4

id	uid	fcm	lang	timestamp	type	user	vue	obj
328	1069s	fkNxRnRmAaA:APA9...	FR	1507888194	5	{"first_name": ...}	1	{"id": {"N": "152"}, "spitch": {"N": ...}}
328	1071s	fkNxRnRmAaA:APA9...	FR	1508352153	5	{"first_name": ...}	0	{"id": {"N": "153"}, "spitch": {"N": ...}}
328	1073s	fkNxRnRmAaA:APA9...	FR	1508362218	5	{"first_name": ...}	0	{"id": {"N": "147"}, "spitch": {"N": ...}}
328	1098s	fkNxRnRmAaA:APA9...	FR	1508782673	5	{"first_name": ...}	0	{"id": {"N": "181"}, "spitch": {"N": ...}}
328	1101s	fkNxRnRmAaA:APA9...	FR	1509018227	5	{"first_name": ...}	0	{"id": {"N": "189"}, "spitch": {"N": ...}}
328	1103s	fkNxRnRmAaA:APA9...	FR	1508018667	5	{"first_name": ...}	0	{"id": {"N": "190"}, "spitch": {"N": ...}}
328	1106s	fkNxRnRmAaA:APA9...	FR	1509110491	5	{"first_name": ...}	0	{"id": {"N": "197"}, "spitch": {"N": ...}}
328	1107s	fkNxRnRmAaA:APA9...	FR	1509111925	5	{"first_name": ...}	0	{"id": {"N": "196"}, "spitch": {"N": ...}}
328	1108s	fkNxRnRmAaA:APA9...	FR	1509141665	5	{"first_name": ...}	0	{"id": {"N": "198"}, "spitch": {"N": ...}}
328	1109s	fkNxRnRmAaA:APA9...	FR	1509269361	5	{"first_name": ...}	0	{"id": {"N": "204"}, "spitch": {"N": ...}}
328	1114s	fkNxRnRmAaA:APA9...	FR	1509700804	5	{"first_name": ...}	0	{"id": {"N": "221"}, "spitch": {"N": ...}}
328	1115s	fkNxRnRmAaA:APA9...	FR	1509703001	5	{"first_name": ...}	0	{"id": {"N": "220"}, "spitch": {"N": ...}}
328	1116s	fkNxRnRmAaA:APA9...	FR	1509703002	5	{"first_name": ...}	0	{"id": {"N": "220"}, "spitch": {"N": ...}}
328	1123s	fkNxRnRmAaA:APA9...	FR	1510054173	5	{"first_name": ...}	0	{"id": {"N": "230"}, "spitch": {"N": ...}}

« Exemple de table Notification avec DynamoDB »

5. Annexe 5

The screenshot shows the AWS Lambda function configuration interface for the function `spitchtv-lambdaStartImageProcessing-X4I5VUPHK4MW`. The top navigation bar includes Qualifiers, Actions, Select a test event..., Test, and Save buttons.

Designer: The left sidebar lists various trigger types: API Gateway, AWS IoT, Alexa Skills Kit, Alexa Smart Home, CloudWatch Events, CloudWatch Logs, CodeCommit, Cognito Sync Trigger, and DynamoDB. The **Add triggers** section allows selecting triggers from this list to add them to the function.

Triggers: A single trigger named `S3` is listed under the `(2)` heading. The configuration for this trigger is shown below.

Related functions: A dropdown menu titled "Select a function" is shown, listing AWS Step Functions, AWS XRay, Amazon CloudWatch Logs, and Amazon S3.

Function code: The code entry type is set to "Edit code inline". The runtime is Python 2.7, and the handler is `lambda_function.lambda_handler`.

Code Editor: The code editor window displays the Python script `lambda_function.py`:

```
1 import boto3
2 import json
3 import os
4
5 def lambda_handler(event, context):
6     sf = boto3.client('stepfunctions')
7
8     for record in event['Records']:
9         bucket = record['s3']['bucket']['name']
10        key = record['s3']['object']['key']
11        datas = key.split('/')
12
13        datas = [
14            {
15                "bucket": bucket,
16                "key": key,
17                "size": record['s3']['object']['size'],
18                "user": datas[-3]
19            }
20        ]
21
22        sf.start_execution(
23            stateMachineArn=os.environ["ARN_IMAGE_PROCESSING"],
24            input=json.dumps(datas)
25        )
```

« Exemple Lambda qui écoute le bucket S3 et récupère chaque objet »