Dossier d'Architecture Technique (DAT)

Version 1.0

Table des matières

1. Introduction

1.1. Objectif du document

Ce document est le document d'architecture de la solution logicielle Bimedoc ; il vise à donner une vision d'ensemble des problématiques structurantes et de la solution (d'un point de vue applicatif et technique), ainsi que de présenter les choix structurants de principes et de composants et les raisons de ces choix.

Il s'adresse aux personnes suivantes :

- Les architectes applicatifs des projets désirant intégrer Bimedoc
- Les architectes techniques des projets désirant intégrer Bimedoc

1.2. Structure du document

Ce document est séparé en 4 grandes parties :

- L'architecture applicative, principalement à destination des architectes applicatifs
- L'architecture technique, avec notamment :
  - En première sous-section, les principes d'architecture technique, principalement à destination des architectes d'infrastructure
  - Dans la suite, les choix d'architecture et de composants techniques, à destination des architectes d'infrastructure et des exploitants
- Les principes et règles de sécurité appliqués et applicables à la solution
- Le dossier d'exploitation de la solution

1.3. Mise à jour du document

Révision	Date	Auteur	Commentaire
1.0	25/06/2021	Thomas Emery	Première version du document

2. Rappels

2.1. Glossaire

- **AOD** : Anticoagulants Oraux Directs
- **API** : Application Programming Interface
- **AVK** : Antivitamine K
- **AWS** : Amazon Web Services
- **API REST** : API Representational State Transfer
- **AWS EC2** : Service Amazon Elastic Compute Cloud
- **AWS RDS** : Service Amazon Relational Database Service
- **AWS S3** : Amazon Simple Storage Service
- **AWS IAM** : Service Amazon Identity and Access Management
- **AWS VPC** : Virtual Private Cloud
- **BDD** : Base de Données
- **BPM** : Bilan Partagé de Médication
- **DPO** : Data Protection Officer
- **HDS** : Hébergeur de Données de Santé
- **JS** : JavaScript
- **MAJ** : Mise A Jour
- **OAuth2** : Open Authorization version 2
- **OS** : Operating System (système d'exploitation)
- **OTP** : One-Time Password (mot de passe à usage unique)
- **SAAS** : Software As A Service
- **SHA256** : Secure Hash Algorithm 256 bits
- **SSO** : Single Sign-On

3. Vue d'ensemble

3.1. Drivers du projet

3.1.1. Enjeux

Les enjeux de la solution logicielle Bimedoc se répartissent en trois grandes catégories :

- Les enjeux liés à l'hébergement de données de santé (choix d'un hébergeur certifié HDS + sécurisation de l'accès aux données)
- Les enjeux liés à la disponibilité de la solution (il ne doit pas y avoir d'interruption de service pour le professionel de santé)
- Les enjeux liés à l'interopérabilité avec d'autres éditeurs de logiciels (mise à disposition d'une API REST avec un protocole d'authentification reposant sur OAuth2)

3.1.2. Contraintes et objectifs

L'objectif métier de la solution est de permettre aux professionnels de santé d'enregistrer des données relatives aux patients pendant leurs parcours de soin. Pour faciliter la saisie des informations, le professionnel de santé peut se reposer sur plusieurs outils :

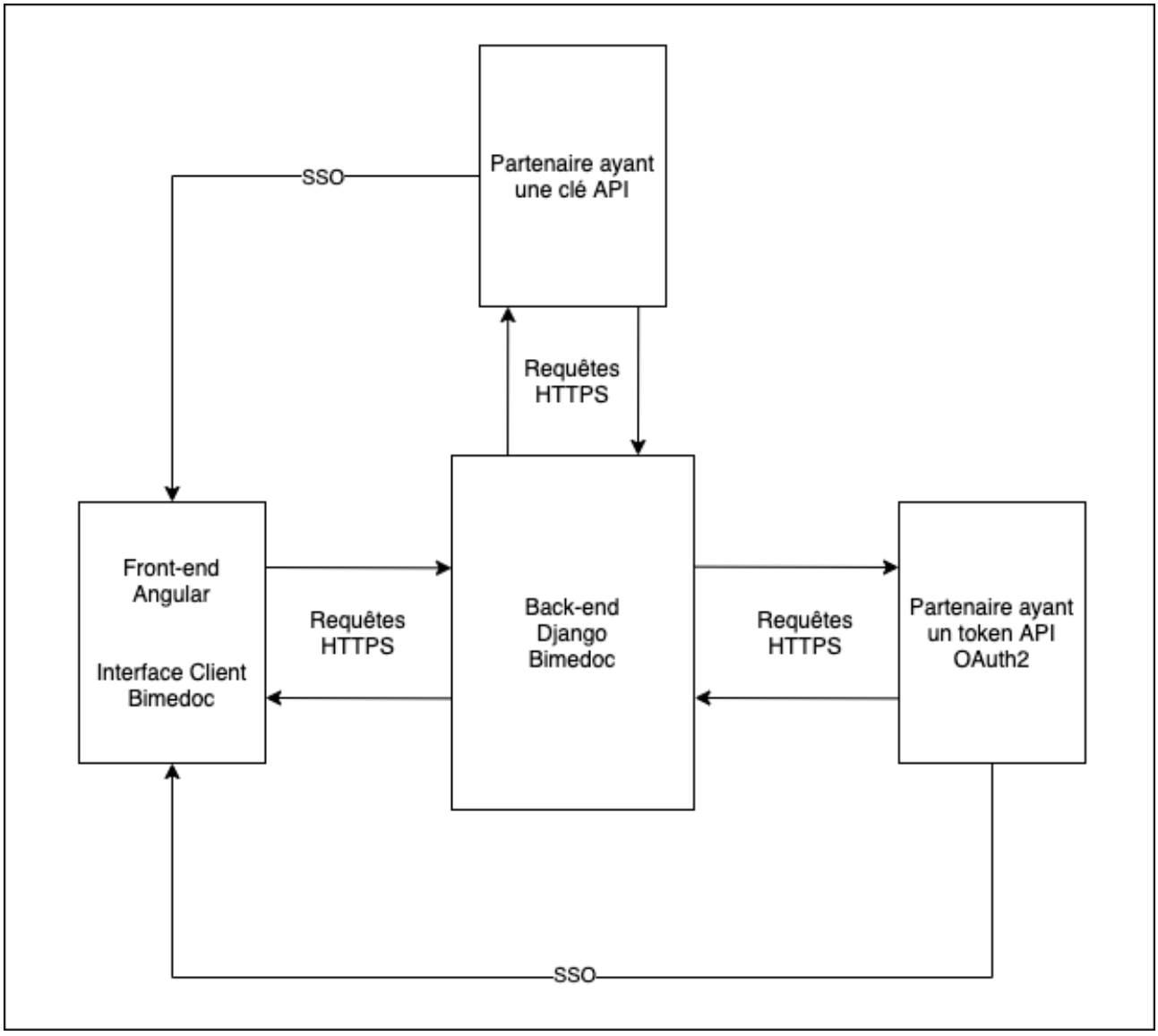
- Base de données des médicaments et des pathologies
- Assistant d'analyse médicamenteuse
- Reconnaissance optique d'ordonnance
- Messagerie sécurisée

L'hébergement et le traitement de données de santé sont réglementés en France. L'hébergement est fait sur une solution certifiée HDS et le traitement des données est encadré par notre DPO Pierre Le Guellec du cabinet Yec'hed Mat.

3.1.3. Positionnement

La solution logicielle Bimedoc est une solution SAAS permettant aux professionnels de santé (principalement les pharmaciens) de suivre les patients dans leur parcours de soin. La solution logicielle Bimedoc a pour but d'être largement réutilisable, et ce notamment en se basant sur l'usage de standards métiers :

- La fonctionnalité d'assistant d'analyse médicamenteuse a été classée en tant que dispositif médical de niveau 1. Nous travaillons sur la classification en niveau 2



Interconnexions externes

3.2. Interfaces externes du système

3.2.1. Interfaces utilisateur

La solution est accessible via le web à partir d'un navigateur et d'une version autorisés. L'objectif est de couvrir les navigateurs et les versions les plus utilisés sur le web aujourd'hui, et d'exclure les navigateurs et / ou les versions qui ne sont plus supportées ou qui ne permettent pas de fournir à l'utilisateur une expérience optimale.

Ci-dessous la configuration Browserslist :

Réglage	Description
> 0.5%	Version du navigateur qui représente plus de 0,5% de l'usage globale (tous navigateurs et versions confondus)
last 2 versions	Les deux dernières versions du navigateur
Firefox ESR	Dernière version de Firefox Extended Support Release
not dead	Exclure les versions qui n'ont pas de MAJ ou de support depuis 24 mois ou plus
not IE 9-11	Exclure Internet Explorer 9, 10 et 11

L'interface n'est pas optimisée pour une utilisation sur mobile.

3.2.2. Interfaces requises

La solution est interfacées avec quatre solutions externes en API :

- Stripe : pour gérer la facturation mensuelle
- Hubspot : mise à jour des données clients entre Hubspot (outils de gestion des clients et de la prospection) et Bimedoc (données de la pharmacie)
- Sendinblue : outils pour envoyer des emails aux clients
- Thériaque : base de données des médicaments

Bimedoc reste opérationnel même si une ou plusieurs de ces connexions sont cassées. Les fonctionnalités coeur de l'application ne dépendent pas de ces données.

3.2.3. Interfaces métier exposées

Back-office

La gestion des clients et le paramètrage de certaines fonctionnalités se font via une interface d'administration dont l'accès est autorisé à :

- Un couple email / mot de passe
- Une adresse IP autorisée

Une gestion de droit pour chaque utilisateur permet de limiter les actions.

Back-end

La gestion des bases de données et des serveurs se fait via l'interface d'AWS, qui fournit les services AWS RDS et AWS ElasticBeanstalk

Front-end

La gestion de l'application front-end est faite sur l'interface Firebase de Google qui héberge le code compilé

3.3. Orientations générales

Ces orientations générales donnent la direction vers laquelle tend la solution logicielle Bimedoc

- API REST : La communication avec le serveur est possible via une API REST, consommée par l'application front-end Bimedoc, mais aussi par les éditeurs de logiciels partenaires. Cette approche permet une accès universel quelque soit le client (Bimedoc ou autre éditeur)
- OAuth2 : L'accès à l'API REST par des éditeurs partenaires est sécurisé via le protocole OAuth2. Ce protocole a été choisi car il s'agit d'un standard largement utilisé dans l'industrie
- Authentification forte : Le premier niveau d'authentification est fait sur la base d'un coupe email / mot de passe. Le deuxième niveau est soit un code OTP envoyé par SMS, soit la vérification de l'adresse IP (elle doit être enregistrée et associée à l'utilisateur qui demande l'accès)

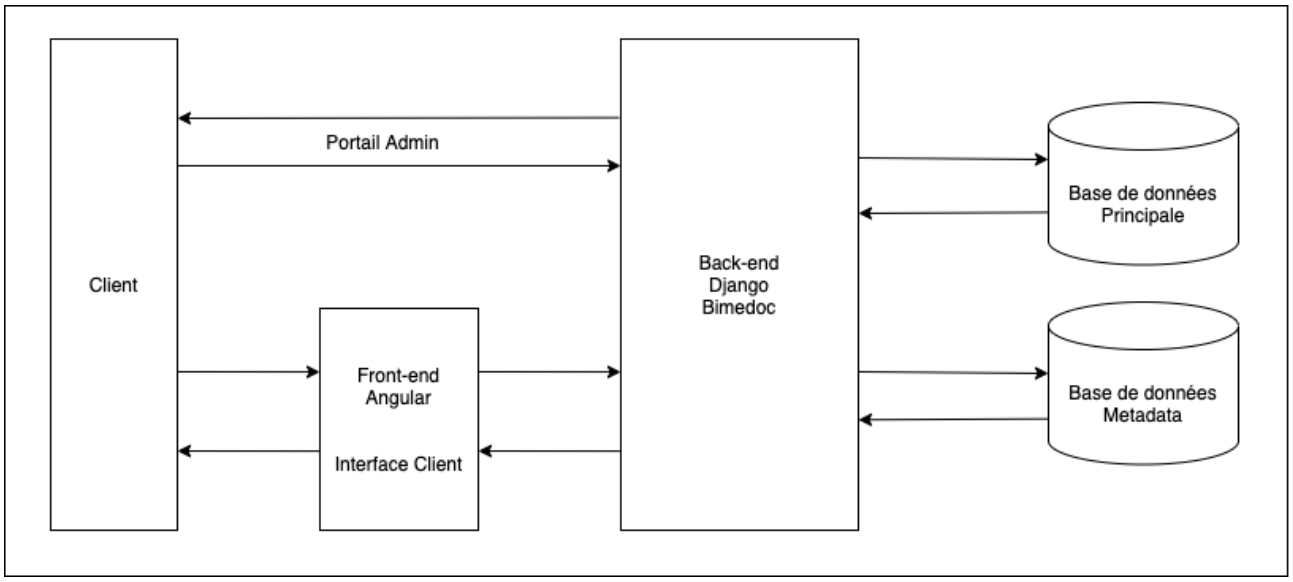
3.4. Architecture fonctionnelle

La solution est composée de deux applications front-end et d'un monolithe back-end qui communiquent par API.

Le service d'authentification par couple email / mot de passe est géré par l'application Django. Une fois l'utilisateur authentitifié par ce couple ainsi que par le code OTP envoyé par SMS ou par son adresse IP, un token JWT est genéré par le serveur. Ce token permet ensuite d'identifier chacune des requêtes HTTP envoyées au serveur.

L'interface back-office est gérée uniquement par Django. C'est le serveur qui sert lui même les fichiers HTML pour le client. La page de connexion est différente de celle pour la connexion des professionnels de santé et se fait avec un couple email / mot de passe et une adresse IP enregistrée.

Lors de chaque requête, il est vérifié que l'utilisateur est bien connecté et que son profil comporte les autorisations nécessaires.



Architecture fonctionnelle

4. Architecture applicative

Cette section décrit l'architecture applicative interne de la solution logicielle Bimedoc, i.e. les différents composants la constituant et leurs interactions.

4.1. Architecture de l'application

4.1.1. Drivers de l'architecture

Les principes d'implémentation applicative ont pour but de faciliter, voir d'assurer les enjeux auxquels la solution logicielle Bimedoc est confrontée :

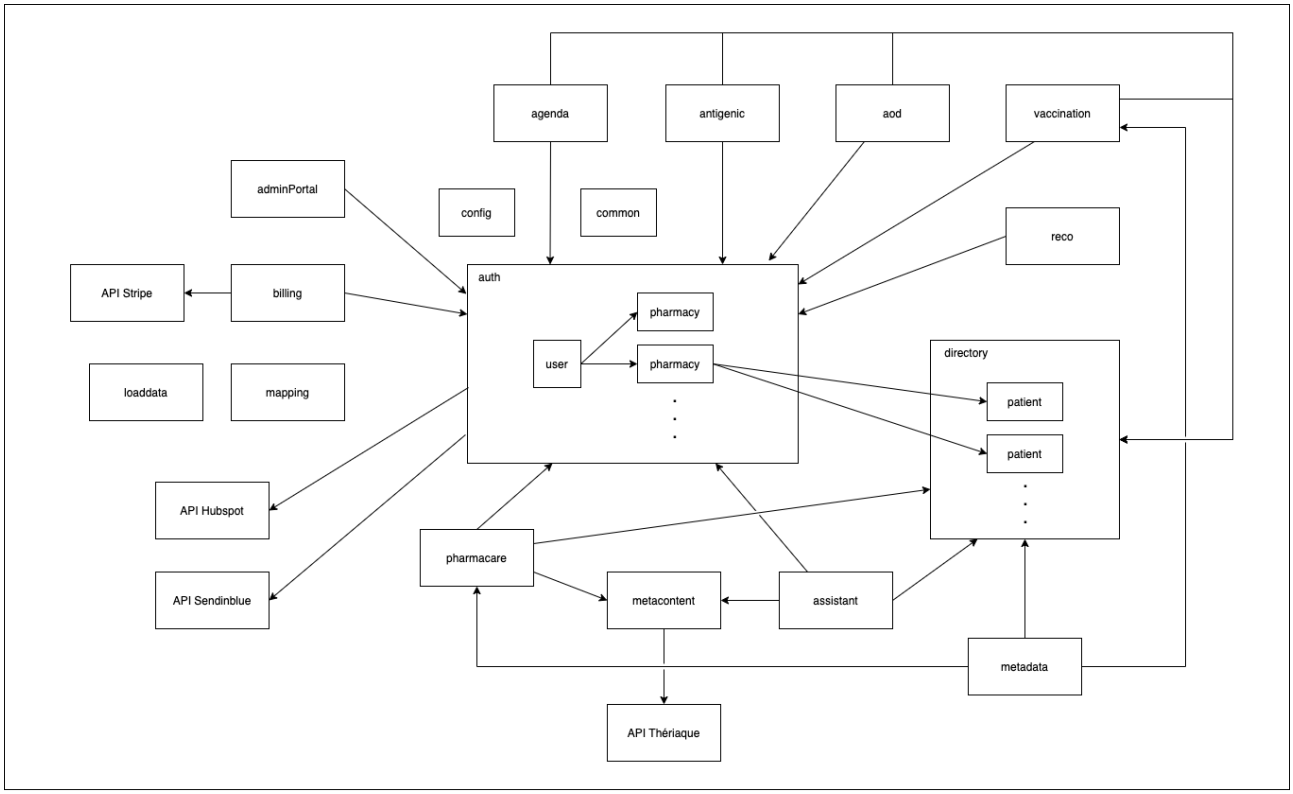
- Accès multi-devices via différents languages front-end (framework JS pour le web, language natif pour le mobile)
- Disponibilité maximum de l'application pour les professionels de santé
- Interopérabilité de la solution avec d'autres solutions logicielles dans le domaine de la santé (téléconsultation, ordonnance électronique, analyse médicamenteuse etc.)

4.1.2. Services

Le back-end Bimedoc ne comporte qu'un seul service. Il s'agit d'une application Django monolithique. Ce projet est découpé en plusieurs "applications" avec des interactions limitées entre elles. Voici la liste des applications :

- **adminPortal** : gestion de l'interface back-office
- **agenda** : gestion des endpoints liés à la fonctionnalité d'agenda
- **antigenic** : gestion des endpoints liés à la déclaration des tests antigéniques au SI-DEP
- **aod** : gestion des endpoints liés à la fonctionnalité d'entretien pharmaceutiques AOD / AVK
- **assistant** : gestion des endpoints liés à la fonctionnalité de l'assistant d'analyse pharmaceutique
- **authentication** : cette application comporte les modèles (tables de la base de données) User (authentication_user) et Pharmacy (authentication_pharmacy) qui nous permettent d'identifier un utilisateur et le scope de données auxquelles il peut accéder. Toutes les autres applications font réference à ces modèles. La gestion du login / logout est faite ici également
- **billing** : gestion de la facturation avec Stripe
- **common** : fonctions utilitaires partagées dans l'ensemble du service
- **conciliation** : gestion des endpoints liés à la concilliation médicamenteuse
- **config** : emplacement des fichiers de configuration
- **directory** : cette application contient les fonctionnalités pour la gestion des données patients. Toutes les applications y font référence. Se trouve également les fonctions de messagerie sécurisée, de bilan partagé de médication
- **loaddata** : application pour générer des données de test pour nos environnements de développement
- **mapping** : application gérant le mapping de notre base de données médicamenteuse avec l'API externe DDI. A usage interne uniquement
- **medeo_api** : gestion de la relation avec l'API de notre partenaire Medeo

- **metacontent** : gestion des données de notre base de données medicamenteuse et de pathologies. Utilisé par notre première version d'annuaire des professionnels de santé et de l'assistant d'analyse pharmaceutique
- **metadata** : gestion de la relation avec la base de données metadata qui est commune à tous nos environnements (dev, preprod, prod). Les données de cette application sont également utilisées par directory, pharmacare et vaccination
- **pharmacare** : gestion du parcours iatroprev
- **reco** : gestion de la reconnaissance des ordonnances. Utilise également les données médicamenteuses de metacontent
- **vaccination** : gestion de la vaccination



Flux d'applications Django

4.2. Architecture des données

4.2.1. Inventaire des données

Le tableau ci-dessous représente l'inventaire des données gérées par Bimedoc, avec leur localisation et le composant responsable du cycle de vie de la donnée (i.e. règles de création / modification / suppression) :

Type	Données prises en charge par le module	Type
Pharmacie	auth	Métier
Utilisateur	auth	Métier
Patient	directory	Métier

Type	Données prises en charge par le module	Type
Assistant d'analyse	assistant	Métier
BPM	directory	Métier
Pathologies	directory	Métier
Traitements	directory	Métier
Prof. de santé	directory	Métier
Entretiens AOD/AVK	aod	Métier
Agenda	agenda	Métier
Tests antigéniques	antigenic	Métier
Vaccination	vaccination	Métier
Conciliation	conciliation	Métier
Plan pharmaceutique partagé	pharmacare	Métier
Facturation	billing	technique

4.2.2. Stockage et stratégies

Données patient

Les données patients sont stockées dans une base de données PostgreSQL sur AWS RDS dans la région eu-west-3 (Paris). Les backups de ces donnnées sont stockés sur un autre compte AWS, avec AWS RDS dans la région eu-west-1 (Ireland).

Données anonymisées

Les données anonymisées utilisées pour le pilotage stratégique sont stockées sur AWS S3 dans la région eu-west-1 (Ireland).

Données de session

Les données sur l'utilisateur sont stockées en LocalStorage pour faciliter l'utilisation de la plateforme.

Champs	Description
token	token de session de l'utilisateur
id	id de l'utilisateur
email	email de l'utilisateur
first_name	prénom de l'utilisateur
last_name	nom de l'utilisateur
function	fonction de l'utilisateur
gender	genre de l'utilisateur
is_2fa_necessary	booléen pour savoir si une authentification forte est nécessaire
is_action_needed_for_payment	booléen pour savoir si un paiement est requis sur le compte
is_activated	booléen pour savoir si l'utilisateur a été activé
mobile_number	numéro de téléphone de l'utilisateur

Champs	Description
organizations	liste des organisations auquel l'utilisateur appartient (id / nom / type de compte)
rpps_number	numéro RPPS de l'utilisateur
secure_email_list	liste des adresses email sécurisées de l'utilisateur
student_code	code étudiant de l'utilisateur si fonction étudiant
university	université de l'utilisateur si fonction étudiant

L'authentification des requêtes HTML permet un filtrage préalable des données par le serveur en fonction de l'organisation de l'utilisateur. Un utilisateur ne peut pas avoir accès à des données n'ayant pas de rapport avec son organisation. Chaque donnée en fonction d'un patient est liée à une seule organisation. Par exemple, un patient A est client de deux pharmacies différentes, il sera representé deux fois en base de donnée. Certaines données peuvent êtres partagées par plusieurs organisations, comme dans le cadre du Plan Pharmaceutique Partagé.

Les documents téléversés par les utilisateurs sont sauvegardés dans un bucket S3. L'emplacement de chacun des documents est sauvegardé dans la base de données PostgreSQL.

4.2.3. Multisite

Pas de gestion multisite

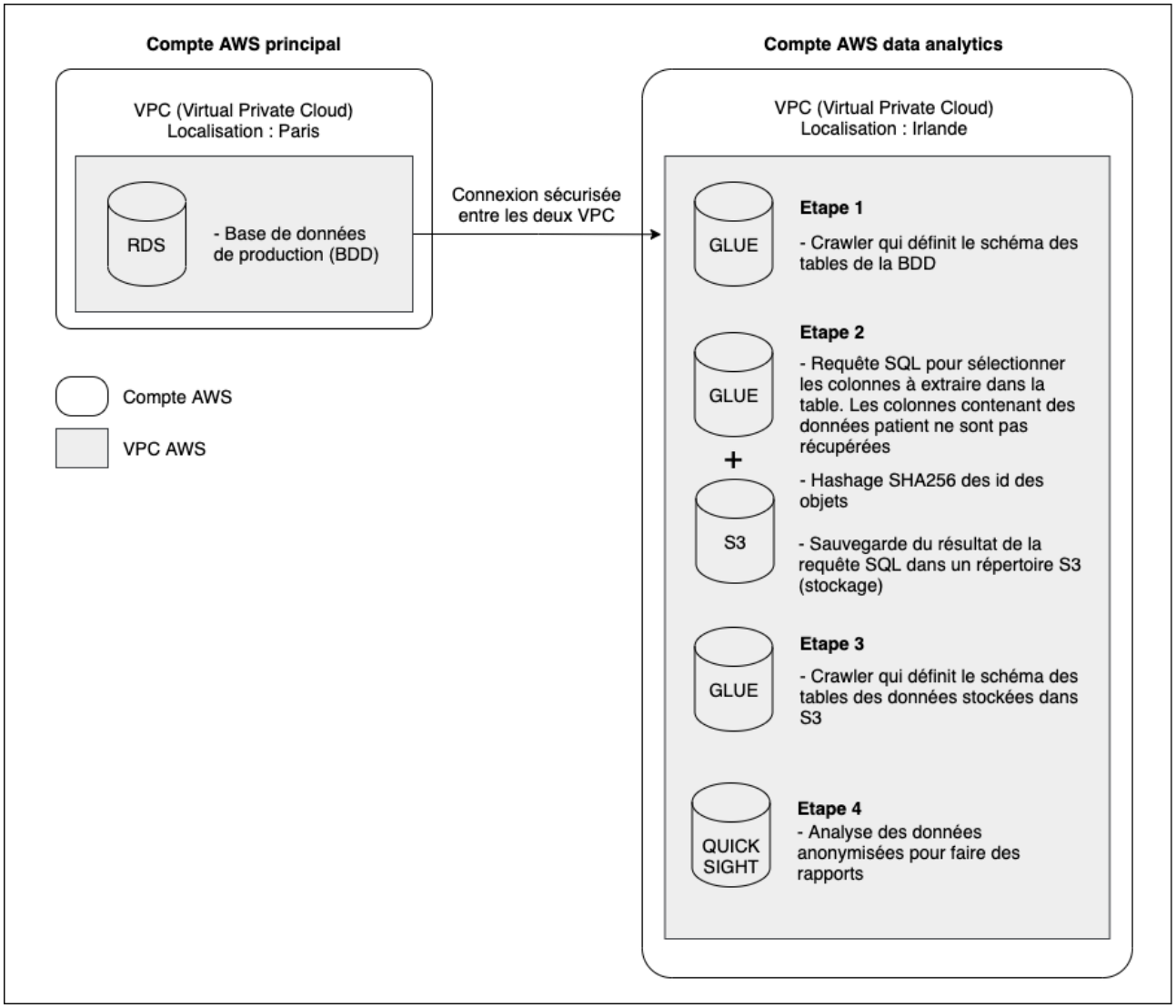
4.2.4. Traitements des données

L'analyse des données à des fins de pilotage stratégique de l'entreprise se fait sur un jeu de données anonymisé. Ci-dessous le processus d'extraction des données pour analyse :

- Lors de la requête SQL, les colonnes des tables contenant des données patient ne sont pas sélectionnées
- Les id des objets sont hashés via un algorithme SHA256

Les données extraites et sauvegardées dans AWS S3 sont remplacées à chaque nouvelle extraction, toutes les semaines. Il n'y a pas d'archivage des extractions. L'âge de la donnée correspond à la date d'enregistrement dans le dossier AWS S3.

L'accès aux données anonymisées pour la création de rapports d'analyse est restreint aujourd'hui à une personne.



Extraction des données pour analyse

5. Architecture technique

5.1. Principes d'architecture technique

Cette section vise à introduire l'environnement dans lequel s'intègrent les composants présentés à la section précédente et qui permet leur exploitation ; elle se concentre principalement sur les contraintes imposées à cet environnement et les choix d'interfaces techniques exposées et consommées avec l'écosystème logiciel d'exploitation.

5.1.1. Nommage

Type	Règle	Exemple
Endpoint	Camelcase et tiret	/agenda/informationBuffer/ -> /agenda/information-buffer
Tables de la BDD	appname_modelname (géré par Django)	agenda_informationbuffer
Fonctions	Underscore	save_data()
Variables	Underscore	patient_id
Classes	Camelcase	PatientSerializer
Identifiant	Email	thomas.emery@bimedoc.com

5.1.2. Utilisateurs, dossiers & droits

5.1.2.1. Utilisateurs et groupes d'exécution

5.1.2.1.1. Groupes

Groupe	Plateform	Description
Administrators	AWS	Accès à tous les services sans restriction
Super view	Back-office Bimedoc	Accès en lecture à toutes les tables
Developer	Back-office Bimedoc	Accès en lecture à un nombre limité de tables

5.1.2.1.2. Utilisateurs

Interface métier Bimedoc

Accès par professionnels de santé avec les identifiants suivants :

- Un couple email / mot de passe
- Un code OTP par SMS ou une adresse IP enregistrée

Interface back-office Bimedoc

Accès par les employés Bimedoc pour la gestion quotidienne des clients et le paramètrage des modules avec les identificants suivants :

- Un couple email / mot de passe
- Une adresse IP autorisée

Partenaires ayant clé API

Accès à certains services par les partenaires avec une clé API.

Partenaires ayant un token API OAuth2

Accès à certains services par les partenaires avec un token API suite à une identification OAuth2.

Services AWS

- Employés de l'équipe technique qui ont accès aux services AWS. Chaque utilisateur a un niveau d'accès différent en fonction des services qu'il administre
- Utilisateurs virtuels de l'éco-système AWS ayant des droits spécifiques pour effectuer des opérations entre services AWS. Les droits sont gérés dans AWS IAM. Les utilisateurs doivent utiliser un code valide 30 secondes pour se connecter (OTP)

5.1.2.2. Arborescence de fichiers

5.1.2.2.1. Arborescence Bimedoc

```
|  bimedoc
    |  adminPortal      -> app gérant le back-office
       |  migrations
       |  resources
       |  temp_files
       |  templates
          views
    |  application_django
       |  admin
       |  factories
       |  filtersets
       |  migrations
       |  models
       |  serializers
       |  utils
          views

...


    |  templates
```

```
      |  admin
      |  emails
      |  other
      |  registration
         widget_emails
tests
```

Contenu des packages d'une application django :

- admin : fichiers pour afficher la représentation des tables de base de données dans l'admin django
- migrations : fichiers de migrations pour la base de données
- factories : fonctions permettant de créer des modèles pour les tests
- utils : fonctions utilitaires
- filtersets : fonctions pour des filtres customisés pour les endpoints
- serializers : fichiers contenant les fonctions de sérialisation des données : representation sous format json des données de la base de données
- views : paramètrage des endpoints (méthodes http, filtres, etc…)
- models : représentation "pythonique" des tables de la base de données.

Un package contient un fichier contenant le routing des urls vers les différentes "view".

Le package `adminPortal` contient toutes les vues pour le back office.

Le package `templates` contient les différents templates html servis par le serveur, notamment pour les emails.

Le package `tests` contient tous les tests unitaires. Ils sont présents sur le repository Github mais ne sont pas transférés sur l'instance AWS EC2.

5.1.2.2.2. Intégration au système

L'applicatif n'interagit pas avec le système, autrement qu'au travers des fonctions natives de Python Django.

5.1.3. Déploiement de la solution

5.1.3.1. Principes de déploiement

Ci-dessous les étapes du déploiement de la solution :

1- Lancement d'une instance `bimedoc-prod-bdd` de BDD sur AWS RDS

2- Lancement d'un nouvel environnement serveur `bimedoc-prod-server` sur AWS ElasticBeanstalk à partir de la dernière version du code back-end de l'application disponible sur AWS CodeCommit

3- Configuration du Security Group qui gère l'accès à la base de données `bimedoc-prod-bdd`, pour autoriser les flux provenant de l'environnement serveur `bimedoc-prod-server`

4- Configuration du sous-domaine server.bimedoc.com sur AWS Route 53 pour qu'il pointe vers l'url de l'environnement serveur `bimedoc-prod-server`

5- Compilation du code de l'application front-end

6- Déploiement de l'application front-end compilée sur un environnement `bimedoc-prod-app` d'hébergement Google Firebase

7- Configuration du sous-domaine app.bimedoc.com sur AWS Route 53 pour qu'il pointe vers l'environnement `bimedoc-prod-app`

5.1.3.2. Contraintes et vue d'ensemble

Pour déployer la solution, un stockage AWS S3 est nécessaire :

- Pour héberger l'application qui tourne sur l'environnement serveur AWS ElasticBeanstalk (géré en autonomie par AWS)
- Pour stocker les fichiers qui sont chargés par les professionnels de santé sur les dossiers patients
- Pour stocker les données anonymisées nécessaires aux analyses

5.1.3.3. Installation initiale

La solution ne nécessite pas une configuration système spécifique pour être utilisée. La solution est accessible via le web à partir d'un navigateur et d'une version autorisés. L'objectif est de couvrir les navigateurs et les versions les plus utilisés sur le web aujourd'hui, et d'exclure les navigateurs et / ou les versions qui ne sont plus supportées ou qui ne permettent pas de fournir à l'utilisateur une expérience optimale.

Ci-dessous la configuration Browserslist :

Réglage	Description
> 0.5%	Version du navigateur qui représente plus de 0,5% de l'usage globale (tous navigateurs et verions confondus)
last 2 versions	Les deux dernières versions du navigateur
Firefox ESR	Dernière version de Firefox Extended Support Release
not dead	Exclure les versions qui n'ont pas de MAJ ou de support depuis 24 mois ou plus
not IE 9-11	Exclure Internet Explorer 9, 10 et 11

L'interface n'est pas optimisée pour une utilisation sur mobile.

5.1.3.4. Principes de mise à jour

La solution est mise à jour lors :

- du lancement d'une nouvelle fonctionnalité
- de la mise en ligne de correctifs sur des fonctionnalités existantes

La mise à jour est transparente pour l'utilisateur et sera effective au prochain chargement de l'application dans le navigateur.

Une mise à jour peut impliquer l'application front-end et / ou l'application back-end. Ci-dessous le processus pour chaque stack :

Back-end

1- Lancement des tests unitaires

2- Sauvegarde de la dernière version du code sur AWS CodeCommit

3- Déploiement du code depuis AWS CodeCommit sur l'environnement serveur `bimedoc-prod-server` sur AWS ELasticBeanstalk

Front-end

1- Compilation de l'application Angular

2- Déploiement de l'application compilée sur l'environnement `bimedoc-prod-app` de Google Firebase

5.1.3.5. Support de l'élasticité

Back-end

Le serveur de production est hébergé sur AWS EC2 et bénéficie d'une adaptation automatique du nombre d'instances à la charge (Load Balancing). En fonctionnement normal le serveur dispose de deux instances t3.large. En cas de charge importante, le serveur peut disposer de cinq instances t3.large. A date, nous n'avons jamais dépassé 3 instances t3.large.

Front-end

L'application Angular front-end est hébergée sur Google Firebase. Le service gère automatiquement les ressources nécessaires à la disponibilité de la solution en cas de forte demande.

5.1.3.6. Validation du déploiement

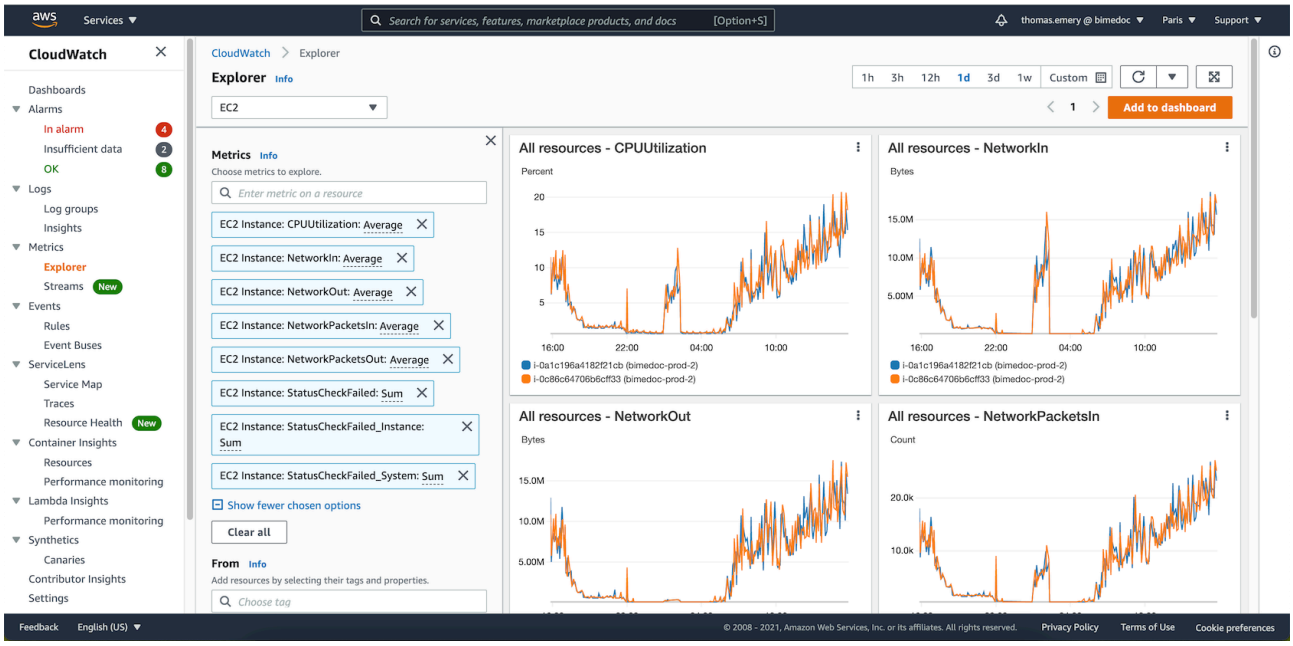
Des tests unitaires permettent de valider le fonctionnement des fonctions déployées. Les déploiements continus sont gérés avec la solution externe CircleCi qui exécute trois opérations en cascade : tests unitaires, build, déploiement. Une alerte est remontée si une erreur intervient lors de l'une de ces étapes, et le déploiement est avorté.

5.1.4. Suivi de l'état du système

5.1.4.1. Endpoint de supervision

La supervision des environnements serveurs qui tournent sur AWS ElasticBeanstalk se fait via AWS CloudWatch, un service qui récoltent toutes les données sur l'utilisation des instances serveur.

Exemple d'écran de pilotage :



Exemple tableau de bord AWS CloudWatch

5.1.4.2. Logs

Back-end

Des logs complets sont disponibles pour chaque environnement serveur sur AWS ElasticBeanstalk :

- Logs opérationnels ElasticBeanstalk
- Logs provenant du server web
- Logs yum et cron

Des logs d'erreurs fonctionnelles sont remontés sur l'interface externe Sentry (par exemple le crash d'une fonction ou une variable appelée qui est à null).

Front-end

Le service Cloud Logging de Google Firebase permet de consulter l'ensemble des reqûetes web qui sont faites à l'application

Des logs d'erreurs fonctionnelles sont remontés sur l'interface externe Sentry (par exemple le crash d'une fonction ou une variable appelée qui est à null).

5.1.4.3. Intégration à un système de monitoring tiers

La solution n'est pas interfacée avec un système de monitoring tiers en ce qui concerne l'infrastructure (si ce n'est CloudWatch et outils fournis par Firebase).

Pour ce qui est du code, toutes les erreurs (à la fois front-end et back-end) sont automatiquement enregistrées dans Sentry, et l'équipe technique reçoit une notification. Le rapport d'erreur contient de nombreuses informations (utilisateur, navigateur internet, ligne de code qui a déclenché l'erreur, etc.).

5.1.5. Administration technique

5.1.5.1. Démarrage / arrêt des services

Il n'y a pas de gestion spécifique de démarrage / arrêt des services.

- Le lancement et l'arrêt de l'application front-end sont gérés par le navigateur du client
- L'environnement serveur est géré entièrement par AWS ElasticBeanstalk

5.1.5.2. Tâches régulières

Annuaire santé

Régulièrement, Bimedoc met à jour un annuaire national des professionnels de santé et de leurs établissements, à partir d'extractions publiques fournies par l'Etat. Les données sont stockées dans la base de données `metadata` et sur AWS CloudSearch, un service d'indexation de données permettant d'obtenir une réponse très rapide lorsqu'un utilisateur fait une recherche dans l'annuaire.

Base de données des médicaments

La base de données médicamenteuse est stockée dans la table `metacontent_medication`. Cette table est mise à jour à l'aide de la base de données du service externe Thériaque. Cette base de données est gérée par l'organisime du même nom dépendant du Centre National Hospitalier d'Information sur le Médicament.

5.1.6. Gestion des données du système

5.1.6.1. Sauvegarde

Concernant la création de sauvegarde des bases de données, le Snapshot tool pour RDS automatise la création de snapshot, les copient dans un compte différent et supprime les backups après un certain nombre de jours. Le paramètrage se passe sur le compte AWS source et sur le compte cible, dédié à la sauvegarde.

Les composants créés sur le compte source, via cloudformation, sont :

- 3 fonctions Lambda :
  - TakeSnapshotsRDS
  - ShareSnapshotsRDS
  - DeleteOldSnapshotsRDS
- 3 State Machines (Amazon Step Functions) pour l'execution des 3 lambdas :
  - stateMachineTakeSnapshotRDS
  - stateMachineShareSnapshotRDS
  - stateMachineDeleteOldSnapshotsRDS
- 3 CloudWatch Event Rules pour déclencher les State Functions
- 3 CloudWatch Alarms et leurs topics SNS pour effectuer les alertes lors des échecs des State Machines
- Une stack Cloudformation contenant toutes les ressources

- Une clef KMS pour le chiffrement RDS

Les instances sont encryptées, il est donc nécessaire de fournir la clef KMS et de la partager au compte de destination.

Les composants créés sur le compte de destination, via cloudformation, sont :

- 2 fonctions Lambda :
  - CopySnapshotsDestRDS
  - DeleteOldSnapshotsDestRDS
- 2 State Machines (Amazon Step Functions) pour déclencher l'execution de chaque fonction lambda :
  - stateMachineCopySnapshotsDestRDS
  - stateMachineDeleteOldSnapshotsDestRDS
- 2 CloudWatch Event Rules pour déclencher les State Functions
- 2 CloudWatch Alarms et leurs topics SNS pour effectuer les alertes lors des échecs des State Machines
- Une stack cloudformation contenant toutes les ressources

Le fonctionnement de ce système est de prendre régulièrement des snapshots et de les copier sur le compte de destination. Ce système de sauvegarde est mis en place pour les bases de données `bimedoc-dev` et `bimedoc-prod`.

Fonctionnement du système de backup

Il y a deux sets de Lambda Step Functions qui prennent des snapshots régulièrement et les copient entre les comptes. Les snapshots prennent du temps lors de leur création et n'émettent aucun signal qui confirme la bonne réalisation. Les snapshots sont programmés pour commencer à une heure spécifique en utilisant CloudWatch Events. Ensuite, différentes Lambda Step Functions sont lancées périodiquement pour chercher de nouveaux snapshots. Quand ces nouveaux snapshots sont détectés, un partage et une copie sont réalisés.

Sur le compte source

Un Event CloudWatch est programmé pour déclencher une Lambda Step Function, via une State Machine nommée *stateMachineTakeSnapshotsRDS*. Cette State Machine invoque une lambda nommée *lambdaTakeSnapshotsRDS*. Cette fonction déclenche un snapshot et y ajoute des tags d'identification, ainsi qu'une modification de son nom, composé du nom de l'instance RDS de la date et de l'heure du snapshot. Il y a deux States Machines associées à leur lambda.

- La State Machine *statemachineShareSnapshotsRDS* vérifie l'existence de nouveaux snapshots créés par *lambdaTakeSnapshotsRDS*. Quand ils sont trouvés, ils sont partagés avec le compte de destination. Ce fonctionnement est effectué toutes les dix minutes.
- L'autre State Machine, nommée *statemachineDeleteOldSnapshotsRDS*, appelle *lambdaDeleteOldSnapshotsRDS* pour effacer les snapshots plus vieux que la date de rétention prévue. Cette State Machine est lancée toutes les heures.

Sur le compte de destination

Il y a deux States Machines associées à leur lambda.

- La State Machine *statemachineCopySnapshotsDestRDS* cherchent les nouveaux snapshots qui ont été partagés avec le compte de destination mais qui n'ont pas encore été copiés. Quand ces snapshots sont trouvés, une copie est réalisée sur le compte de destination, chiffrée avec la clef KMS qui à été précisée dans la configuration. Cette State Machine est lancée toute les 10 minutes.
- L'autre stateMachine, nommée *statemachineDeleteOldSnapshotsRDS* appelle *lambdaDeleteOldSnapshotsRDS* et, comme sur le compte source, efface les snapshots plus vieux que la date de rétention prévue. Cette state Machine est lancée toutes les heures.

5.1.6.2. Restauration

Back-end

- Base de données : Il est possible de relancer une instance de base de données sur AWS RDS en 30 minutes à partir d'un backup, soit dans la région d'origine eu-west-3 (Paris), soit dans la région de backup secondaire eu-west-1 (Ireland).
- Serveur : Il est possible de relancer un environnement serveur sur AWS ElasticBeanstalk en 30 minutes à partir d'une configuration sauvegardée (variables d'environnement, paramètrage des instances etc.) et d'une version du code stockée sur AWS Code Commit, soit dans la région d'origine eu-west-3 (Paris), soit dans une région secondaire eu-west-1 (Ireland).

Front-end

Il est possible de restaurer l'application front-end sur un autre environnement Firebase, ou sur un environnement Netlify (prestataire secondaire) en 1h. Le délai provient majoritairement de la configuration du sous-domaine qu'il faut lié à un nouvel environnement (par exemple app.bimedoc.com). Il est possible en cas d'urgence de rediriger immédiatement le sous-domaine vers l'url fournie par le prestataire (par exemple bimedoc-de2c5.web.app dans le cas de l'environnement Firebase qui héberge actuellement la version de production de l'application).

5.2. Résilience

En cas de perte de données sur une instance de base de données AWS RDS, il est possible de restaurer les données au plus tard cinq minutes avant l'incident. La procédure n'est pas automatisée, et nécessite une intervention humaine après la détection de l'incident via une alerte.

Tous les fichiers chargés sur AWS S3 sont également copiés dans un second répertoire. En cas de perte du répertoire principal, il faut manuellement router l'application vers le répertoire de copie, ce qui nécessite un changement sur les variables d'environnement du serveur. La modification et la mise à jour sont très rapides.

5.3. Haute-disponibilté

Back-end

- Environnements serveur

En cas de perte d'une instance serveur AWS EC2, elle est automatiquement remplacée.

L'environnement de production compte au minimum deux instances, ce qui permet de limiter les risques d'indisponibilité en cas de perte et de remplacement d'une instance. Si toutes les instances tombent en même temps, un délais de quelques secondes est nécessaire pour que les nouvelles instances soient opérationnelles.

- Bases de données

Il n'y a pas de mécanisme similaire en place actuellement pour les instances de base de données AWS RDS.

Front-end

Il n'existe pas de redondance pour l'application front-end. Une relance manuelle sur un autre environnement Google Firebase ou sur un environnement d'un prestataire de secours est nécessaire (Netlify). Cette opération est rapide.

L'application front-end n'héberge aucune donnée donc le remplacement en cas de problème n'engendre aucune complication, mise à part l'indisponibilité temporaire.

5.4. Règlementation hébergement de données de santé

Les données sont hébergées sur AWS RDS région eu-west-3 (Paris) et eu-west-1 (Ireland) pour les sauvegardes, qui sont certifiés HDS.

5.5. Composants logiciels utilisés

5.5.1. Fournis

Principales dépendances Python utilisées par le back-end

- boto3 : SDK AWS pour interagir avec les services AWS tels que S3 ou lambda
- django-cleanup : Gestion des suppressions des fichiers avec S3
- django-cors-headers : Gestion des CORS
- django_extensions : Helper Django
- django-filter : Helper Django pour filtrer la base de donnée
- django-import-export : Helper pour l'admin Django
- django-ipware : Django Helper pour vérifier l'adresse IP des requetes
- django-oauth-toolkit : Django Helper pour utiliser Oauth
- django-storages : Django Helper pour la gestion de fichier dans S3
- djangorestframework : Django Helper pour la structure REST de l'application
- djangorestframework-jwt : Django Helper pour générer les token JWT
- django-simple-history : Django Helper pour la sauvegarde
- dvc : Outil de versionning utilisé pour l'enregistrement des performances de notre application de reconnaissance des ordonnances
- google-api-python-client : SDK google pour comuniquer avec leurs services
- google-auth-httplib2 : SDK google pour comuniquer avec leurs services d'authentification

- M2Crypto : Bibliotheque de chiffrement afin de créer des emails S/MIME pour la déclaration des tests antigéniques au SI-DEP
- numpy : Outils de datascience pour la reconnaissance des ordonnances
- pandas : Outils de datascience pour la reconnaissance des ordonnances
- psycopg2 : Interface entre Django et PostgreSQL
- rapidfuzz : Outil de fuzzy matching utilisé pour la reconnaissance des ordonnances
- sentry-sdk : SDK de Sentry, outil de monitoring
- sklearn : Outils de datascience pour la reconnaissance des ordonnances
- stripe : Outil pour le paiement en ligne
- twilio : Outil pour l'envoi de sms
- fhir.resources : Outil pour serialiser des données au format Fhir

Le service Dependabot est utilisé pour être notifié immédiatement de nouvelles versions des dépendances ainsi que des mises à jour de sécurité (indiquées en rouge dans les rapports Dependabot).

5.5.2. Requis

5.6. Outillage de déploiement

5.6.1. Outil

- GitHub : Versionnage du code source (front-end et back-end)
- CircleCi : Solution externe utilisée pour le déploiement continu lorsqu'une modification du code est enregistrée sur GitHub
- Firebase : Déploiement de l'application Angular compilée et hébergement
- AWS CodeCommit : Versionnage du code back-end sur un service complémentaire, en vue du déploiement
- AWS ElasticBeanstalk : Déploiement du code back-end Python sur l'environnement serveur

5.6.2. Gestion des secrets

AWS Secrets permet le stockage des variables sensibles (clé RSA OAuth2, mot de passe de connexion aux bases de données). Les secrets sont utilisés de manière native dans les autres services AWS comme ElasticBeanstalk. Bimedoc ne sauvegarde jamais de valeurs sensibles dans le code.

6. Sécurité

6.1. Principes

Les requêtes à l'API REST sont exécutées via le protocole HTTPS, avec un token d'autorisation obtenu via un login utilisateur auprès du serveur (un couple email / mot de passe et un OTP par SMS ou une adresse IP enregistrée) ou une authentification OAuth2 auprès du serveur. Les tokens utilisateurs ont une validité de 24h et sont invalidés si la requête provient d'une adresse IP différente à celle du login. Les tokens OAuth2 ont une validité de 1h.

Bimedoc a plusieurs milliers de tests unitaires qui sont actionnés à chaque changement de code, aussi minime soit-il. Des membres de l'équipe produit vont ensuite faire des tests manuels de la solution en considérant de nombreux facteurs, incluant la sécurité.

6.1.1. Principes de cloisonnement

Objectif de chaque environnement

La solution est constituée de trois environnements distincts : développement, pré-production et production. Aucun lien n'existe entre les environnements.

Développement

Environnement utilisé par les développeurs pour tester les évolutions et les correctifs en cours de développement. Usage strictement interne.

Pré-production

Environnement utilisé par les product managers pour tester les évolutions et correctifs prêts à être déployés. Usage interne.

Production

Environnement utilisé uniquement par les professionnels de santé, contenant des données de santé. Usage externe.

Description des environnements

Chaque environnement est constitué de :

- Un code compilé Front-End en language Angular, hébergé sur un environnement Google Firebase indépendant
- Un code Back-End en language Python, hébergé sur un environnement serveur AWS Elastic-Beanstalk indépendant
- Un base de données indépendante hébergée sur AWS RDS

Deux éléments de configuration sont partagés par tous les environnements, assurant leur stabilité :

- Le paramètrage des environnements serveur AWS ElasticBeanstalk avec un système de Load Balancing, permettant de repartir la charge sur différentes instances, et d'en rajouter si le serveur est plus sollicité pour éviter une indisponibilité du service
- Les variables d'environnement gérées dans les paramètres des instances permettent d'avoir une configuration uniforme entre les environnements, et de faire varier la valeurs des variables si une distinction entre dev / preprod / prod existe

Par exemple : le lien vers la BDD associée est différent d'un environnement à l'autre puisque chaque environnement a sa propre BDD

6.1.2. Principes de sécurisation des accès externes

Les bases de données hébergées sur AWS RDS ne sont pas accessibles publiquement. Elles sont accessibles depuis le VPC sur lequel elles sont installées. Dans le VPC, il y a également une protection par un Security Group. Ce Security Group n'autorise aucun accès extérieur en dehors de ceux explicités dans sa configuration, et les services AWS qui ont besoin d'un accès à ces bases de données sont référencés manuellement (par exemple AWS ElasticBeanstalk).

6.1.3. Principes de sécurisation des communications internes au système

6.1.4. Principes de sécurisation des bases de données

Les bases de données sont sécurisées via un cloisonnement physique et/ou logique des différentes bases de données qui les constituent.

6.2. Certificats

Le domaine bimedoc.com est couvert par un certificat SSL géré par AWS Certificate Manager. Il permet de sécuriser les communications avec les serveurs de développement (server-dev.bimedoc.com), de pré-production (server-test.bimedoc.com) et de production (server.bimedoc.com).

7. Exploitation

Cette partie sera à terme déplacée dans un DEX.

7.1. Supervision

Différents outils sont utilisés pour superviser la bon fonctionnement de la solution :

- Sentry : solution externe permettant de capter le premier niveau d'erreurs lors de l'utilisation de la plateforme, côté front-end et côté back-end.
- Logs poussés dans Slack par le back-end : une couche de notification d'erreurs supplémentaire a été ajoutée pour certains périmètres sensibles (comme la déclaration de tests antigéniques au SI-DEP). Le serveur back-end pousse directement des notifications d'erreur dans notre solution de communication interne Slack.
- AWS EC2 / RDS : notifications d'erreur par email en cas de surcharge ou d'indisponibilité du service.

7.2. Maintien en condition opérationnelle

- Les serveurs AWS EC2 sont revus une fois par mois (mise à jour du serveur et de la version du code Python)
- Notification automatique hebdomadaire de la mise à jour des dépendances utilisées avec Dependabot (front-end et back-end)

7.3. Sécurité

- Accès aux serveurs EC2 et aux bases de données RDS : strictement réservé à deux personnes (aujourd'hui deux associés)
- Accès à l'interface back-office Bimedoc : authentification avec un couple email / mot de passe et une adresse IP enregistrée

7.4. Sauvegarde

7.4.1. Réalisation des sauvegardes

Différents mécanismes de sauvegarde sont en place pour assurer la continuité du service en cas de perte de données.

- Backup AWS RDS : un backup des bases de données est fait toutes les heures. Les backups sont supprimés après 14 jours (les données patient supprimées doivent être supprimées physiquement après 14 jours)

- Redondance backup AWS RDS : une copie des backups est envoyée toutes les heures sur un autre compte AWS dans une autre région, eu-west-1 (Ireland), pour assurer la continuité en cas de problème sur le site eu-west-3 (Paris). Les copies redondantes sont supprimées après 14 jours
- Sauvegarde de la configuration des environnements serveur EC2 : à chaque modification de la configuration de l'environnement une sauvegarde est réalisée, permettant de relancer rapidement un nouvel environnement déjà configuré
- Le code source est sauvegardé et versionné sur GitHub. Le code back-end est également sauvegardé et versionné sur AWS CodeCommit

7.4.2. Viabilité des sauvegardes

- Backups AWS RDS : possibilité de relancer une nouvelle instance de base de données en 15 minutes à partir d'une sauvegarde. Dans la région eu-west-3 (Paris) ou autre région certifiée HDS si besoin
- environnement serveur AWS ElasticBeanstalk : possibilité de relancer un nouvel environnement serveur en 30 minutes
- Front-end : possibilité de relancer un serveur pour faire tourner l'application Angular sur le service Netlify en 30 minutes, si le service Google Firebase est indisponible

7.4.3. Impacts des sauvegardes

Les backups réalisés sur les instances de base de données AWS RDS peuvent créer un ralentissement pendant quelques secondes.

7.5. Interventions support

- Equipe support disponible de 8h30 à 18h30
- Système de ticketing Jira permettant de coordonner rapidement les équipes autour de la résolution d'un problème (commercial, développeur, devops)
- Accès aux environnements serveur AWS ElasticBeanstalk et aux bases de données AWS RDS : strictement réservé à deux personnes dont un associé disponible à temps plein
- Accès à l'interface d'administration : authentification avec un couple email / mot de passe et une adresse IP enregistrée

7.6. Indicateurs de performance

7.6.1. Supervision

Un système de HealthCheck est en place sur AWS EC2 pour vérifier la disponiblité du serveur. Une alerte est remontée en cas d'erreur.

7.6.2. Performance et viabilité de l'infrastructure

Une revue des performances est effectuée une fois par semaine sur AWS CloudWatch pour vérifier la charge des serveurs et adapter les ressources en place (type d'instance serveur et nombre maximum d'instances disponibles). Une alerte est déclenchée si la charge globale des CPU des différents serveurs est supérieure à 80%. Le paramètrage Load Balancing dans AWS EC2 permet d'augmenter automatiquement le nombre d'instances en cas de surcharge, et d'assurer la continuité du service dans l'attente d'une intervention humaine.