# Table des matières

[1.](#_heading=h.gjdgxs) Table des matières 1

[2.](#_heading=h.30j0zll) Table des illustrations 3

[3.](#_heading=h.1fob9te) Validation 4

[4.](#_heading=h.2et92p0) Objet 4

[5.](#_heading=h.tyjcwt) Responsabilités 4

[6.](#_heading=h.3dy6vkm) Lexique 5

[6.1.](#_heading=h.1t3h5sf) Généralités 5

[6.2.](#_heading=h.4d34og8) Django / Python 7

[6.3.](#_heading=h.2s8eyo1) PostGreSQL / Base de données 7

[6.4.](#_heading=h.17dp8vu) Docker 7

[6.5.](#_heading=h.3rdcrjn) Git 8

[6.6.](#_heading=h.26in1rg) Kubernetes 8

[6.7.](#_heading=h.lnxbz9) React 11

[6.1.](#_heading=h.35nkun2) TerraForm 11

[7.](#_heading=h.1ksv4uv) Architecture globale 14

[7.1.](#_heading=h.44sinio) Architecture de haut niveau 14

[7.2.](#_heading=h.2jxsxqh) Interactions entre les composants 14

[7.3.](#_heading=h.3j2qqm3) Gestion des données 15

[7.3.1. Informations générales 15](#_heading=h.1y810tw)

[7.3.2. Configuration des accès 15](#_heading=h.4i7ojhp)

[7.3.3. Schéma de la base 18](#_heading=h.qsh70q)

[8.](#_heading=h.49x2ik5) Processus infrastructure 19

[8.1.](#_heading=h.2p2csry) Données d’entrée 19

[8.2.](#_heading=h.147n2zr) Données de sortie 19

[8.3.](#_heading=h.3o7alnk) Déploiement infrastructure 19

[8.4.](#_heading=h.23ckvvd) Schéma de déploiement de l’infrastructure 20

[8.5.](#_heading=h.ihv636) Configuration Kubernetes 20

[8.6.](#_heading=h.32hioqz) Description du dépôt ops 22

[8.7.](#_heading=h.32hioqz) Déploiement de Kubernetes rpi-test 30

[9.](#_heading=h.3fwokq0) Processus de déploiement de l’application 33

[9.1.](#_heading=h.1v1yuxt) FrontEnd 33

[9.2.](#_heading=h.4f1mdlm) BackEnd 33

[10.](#_heading=h.2u6wntf) Sécurité 33

[11.](#_heading=h.19c6y18) Surveillance et performance 34

# Table des illustrations

[Tableau 1 - Tableau des responsabilités 3](#_heading=h.3znysh7)

[Tableau 2 - Interactions entre les composants 8](#_heading=h.z337ya)

[Tableau 6 - Redirection de la base vers un port local 9](#_heading=h.2xcytpi)

[Tableau 7 - Configuration Connexion PostGreSQL #1 9](#_heading=h.1ci93xb)

[Tableau 8 - Configuration Connexion PostGreSQL #2 10](#_heading=h.3whwml4)

[Tableau 9 - Accès aux tables 11](#_heading=h.2bn6wsx)

[Tableau 10 - ERD Tool 12](#_heading=h.3as4poj)

[Tableau 11 - Schéma de la BDD 12](#_heading=h.1pxezwc)

[Tableau 3 - Représentation de l’architecture d'un cluster Kubernetes 13](#_heading=h.41mghml)

[Tableau 4 - Gestion d'une requête dans Kubernetes 14](#_heading=h.2grqrue)

[Tableau 5 - Carte des ressources Kubernetes 15](#_heading=h.vx1227)

[Tableau 12 - Dossier CI/CD 19](#_heading=h.1hmsyys)

# Validation

| Acteurs | Visa et signature |
| --- | --- |
| Rédacteur  Rony BOUKHRIS |  |
| Approbateur  Ilan BENICHOU |  |

*Tableau 1 - Tableau des responsabilités*

# Objet

Le document contient la description de l'infrastructure d'une application logicielle.

Il inclut des détails sur les composants logiciels, les interactions entre ces composants, les technologies utilisées, ainsi que les infrastructures matérielles et réseaux sous-jacentes. Il est essentiel pour planifier, maintenir et mettre à niveau l'infrastructure logicielle de manière efficace.

# Responsabilités

Ce document est sous la responsabilité du responsable Infrastructure de l’application UpDigital.

# Lexique

## Généralités

**Architecture**

L'**architecture** est le plan ou la conception de la structure globale d'un système informatique ou d'une application, et comprend l'organisation des différents composants logiciels et leur interaction.

Elle définit :

* **Structure logicielle** : Comment les composants logiciels (applications, services, modules) sont organisés et interconnectés.
* **Modèles d'architecture** : Choix de modèles tels que microservices, MVC (Modèle-Vue-Contrôleur), n-tiers, etc.
* **Protocoles de communication** : Définition de la manière dont les composants logiciels communiquent entre eux, par exemple via REST API, SOAP, RPC, etc.
* **Principes de conception** : Règles et principes qui guident la conception et l'implémentation, tels que la séparation des préoccupations, l'encapsulation, etc.
* **Considérations de performance et de sécurité** : Stratégies pour gérer la scalabilité, la fiabilité, la sécurité des données, etc.

La distinction clé est que l'**infrastructure** est plus axée sur les ressources physiques et logicielles nécessaires pour supporter et exécuter les systèmes, tandis que l'**architecture** est centrée sur la conception structurale et l'organisation des composants d'un système pour répondre à des exigences spécifiques. En gros, l'architecture décrit le "quoi" et le "comment" d'un système, tandis que l'infrastructure fournit le "où" et le "avec quoi".

Dans la pratique, une architecture efficace ne peut fonctionner sans une infrastructure adéquate, et une bonne infrastructure nécessite une architecture bien conçue pour être utilisée efficacement. Les deux sont donc interdépendantes mais servent des rôles différents dans le cadre de la technologie de l'information.

**Infrastructure**

L'**infrastructure** fait référence à l'ensemble des composants matériels et logiciels physiquement et fonctionnellement nécessaires pour soutenir l'exécution et la gestion des applications et des services. Cela inclut :

**Infrastructure matérielle (Hardware)**

* **Serveurs** : Machines physiques ou virtuelles qui exécutent les applications et les services.
* **Réseaux** : Composants physiques comme les routeurs, les commutateurs, et les liaisons de données nécessaires pour la connectivité.
* **Dispositifs de stockage** : Disques durs, systèmes de stockage en réseau (NAS), et autres formes de stockage physique.

**Infrastructure logicielle (Software) :**

* **Systèmes d'exploitation** : Logiciels de base qui gèrent le hardware et fournissent des services pour les applications.
* **Logiciels de gestion de bases de données** : Systèmes comme MySQL, Oracle, qui permettent de créer, de manipuler, et de stocker les données de manière organisée.
* **Middleware** : Logiciels qui agissent comme des intermédiaires entre les applications et le système d'exploitation ou entre deux applications.
* **Services cloud** : Plateformes comme AWS, Google Cloud, ou Azure qui fournissent à la fois des ressources matérielles et logicielles à la demande.

L'infrastructure est la fondation sur laquelle les applications et les systèmes sont construits et opérés. Elle peut être on-premise (sur place), hébergée à distance, ou dans le cloud.

## Django / Python

**Django** : Framework web Python de haut niveau qui encourage un développement rapide et une conception propre et pragmatique. (Documentation : <https://www.djangoproject.com/> )

**Python** : Langage de programmation interprété, interactif et orienté objet, utilisé dans de nombreux domaines allant du développement web à la science des données. (Documentation : <https://www.python.org/> )

## PostGreSQL / Base de données

**PgAdmin** : PgAdmin est l'outil de gestion graphique pour PostgreSQL, disponible sous forme d'application web. Il facilite la gestion, le développement, et l'analyse des bases de données PostgreSQL grâce à une interface utilisateur intuitive.

**PostgreSQL** : PostgreSQL est un système de gestion de base de données relationnelle et objet open source. Il est connu pour sa fiabilité, sa conformité SQL et ses fonctionnalités avancées, comme le support des transactions complexes.

**Spilo** : **Spilo** est une "recette" toute prête pour installer et gérer PostgreSQL avec Patroni et d'autres outils qui aident à la sauvegarde et à la sécurité des données. C'est comme un kit complet pour monter vos armoires, avec des outils supplémentaires pour les sécuriser et les dupliquer, tout en s'assurant qu'elles fonctionnent bien ensemble.

Patroni : **Patroni** est un outil de gestion de haute disponibilité pour PostgreSQL. Il assure qu'une copie de votre base de données est toujours prête à être utilisée, même si l'une d'elles rencontre un problème, permettant un basculement rapide et sûr pour maintenir l'accès aux données sans interruption.

## Docker

**Docker** : Plateforme open-source conçue pour faciliter la création, le déploiement et l'exécution d'applications à l'aide de conteneurs. Les conteneurs permettent de packager une application avec toutes ses dépendances dans un bloc autonome, garantissant que l'application fonctionne uniformément dans n'importe quel environnement. (Documentation : <https://www.docker.com/> )

**Dockerfile** : Un Dockerfile est un script de configuration utilisé par Docker pour automatiser le processus de création d'images de conteneurs. Il contient une série d'instructions et de commandes qui sont exécutées en séquence pour assembler une image.

## Git

**Git** est un système de contrôle de version décentralisé gratuit et open source, utilisé pour gérer le code source dans le développement logiciel.

**GitLab CI/CD** : Outil intégré à GitLab permettant l'intégration continue et le déploiement continu, facilitant l'automatisation des étapes de test et de déploiement de logiciels. (Documentation : <https://about.gitlab.com/get-started/>)

**GitHub** : Plateforme de développement collaboratif pour héberger des projets utilisant le système de contrôle de version Git. Elle propose également des fonctionnalités d'intégration continue. (Documentation : <https://github.com/>)

## Kubernetes

**Annotation** : Les annotations en Kubernetes permettent d'ajouter des métadonnées supplémentaires aux objets, telles que des descriptions ou des configurations spécifiques, qui ne sont pas utilisées directement pour identifier et manipuler ces objets.

**ConfigMaps** : Les ConfigMaps permettent de stocker des configurations non confidentielles (comme des fichiers de propriétés, des fichiers de configuration, des scripts) sous forme de paires clé-valeur, qui peuvent être utilisées par les Pods Kubernetes.

**Deployment :** Contrôleur pour déployer et gérer des applications stateless, permettant le scaling et la mise à jour automatique des pods.

**K3s** : K3s est une version allégée de Kubernetes, conçue pour des déploiements plus simples et plus rapides dans des environnements avec des ressources limitées. Il inclut les fonctionnalités essentielles de Kubernetes dans un seul binaire compact.

**Kubernetes**

Système de gestion de conteneurs open-source permettant d'automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. (Documentation : <https://kubernetes.io/fr/>)

**Helm** : Gestionnaire de paquets pour Kubernetes, permettant de définir, installer, et gérer des applications Kubernetes à l'aide de paquets appelés "charts". (Documentation : <https://helm.sh/> )

**hostPath** : Un volume hostPath est spécifique à un nœud. Il pointe vers un fichier ou un répertoire sur le système de fichiers du nœud où le pod s'exécute. Cela lie le pod à ce nœud particulier, car le volume n'est pas accessible depuis d'autres nœuds.

**Namespaces** : Les namespaces sont des partitions virtuelles au sein d'un cluster Kubernetes, permettant d'isoler et de gérer des groupes de ressources de manière indépendante. Ils facilitent l'organisation et la sécurisation des composants du cluster.

**Node** : Un node (nœud) est une machine physique ou virtuelle sur laquelle Kubernetes exécute des Pods. Chaque node est géré par le master node et contient les services nécessaires pour exécuter les Pods.

**Persistent Volume (PV)** : Un Persistent Volume est une section de stockage dans le cluster Kubernetes, indépendante du cycle de vie des Pods individuels, qui permet de stocker des données de manière durable et de les rendre accessibles aux applications. Un PV est une abstraction de stockage qui représente une partie du stockage dans un cluster Kubernetes, indépendamment des nœuds individuels.

**Persistent Volume Claim (PVC)** : Une demande de volume persistant permet à un utilisateur de réclamer un espace de stockage défini par un Persistent Volume. Le PVC spécifie la taille, et parfois le type de stockage, que l’application nécessite.

**Pod** : En Kubernetes, un pod est la plus petite unité déployable qui encapsule un ou plusieurs conteneurs, les ressources de stockage allouées, et une adresse IP unique. Les conteneurs dans un Pod partagent le réseau et d’autres ressources.

**Raspberry Pi** : Ordinateur de la taille d'une carte de crédit destiné à l'apprentissage de la programmation et aux projets électroniques. (Documentation : <https://www.raspberrypi.com/> )

**Traefik** : Proxy inverse et équilibreur de charge open-source, idéal pour les services conteneurisés car il simplifie le routage réseau et la gestion des certificats SSL. (Documentation : <https://doc.traefik.io/traefik/> )

**Secrets** : Les Secrets sont utilisés pour stocker et gérer des informations sensibles, telles que les mots de passe, les tokens OAuth, et les clés SSH, en les protégeant et en les rendant accessibles uniquement aux Pods autorisés.

**Service** : Un Service en Kubernetes est une abstraction qui définit un ensemble logique de Pods fonctionnant comme un microservice réseau. Les Services permettent aux applications de recevoir du trafic via une adresse IP fixe et éventuellement un nom DNS.

**Stateful** (avec état) : "Stateful" indique que l'application doit conserver des données d'état ou des informations sur les sessions précédentes. Ces applications suivent l'état des interactions et peuvent nécessiter des mécanismes pour stocker et récupérer cet état, comme des bases de données ou des systèmes de fichiers persistants.

**StatefulSet :** Contrôleur pour gérer des applications stateful, maintenant une identité unique et un ordre stable pour chaque pod, souvent utilisé pour des systèmes nécessitant un stockage persistant.

**Stateless :** signifie que l'application ne conserve aucune donnée d'état entre les sessions ou les requêtes des utilisateurs. Chaque requête est traitée indépendamment des précédentes, sans avoir besoin de connaître ou de se souvenir de l'état antérieur.

**Visual Studio Code (VSCode)** est un éditeur de code source gratuit et open source de Microsoft, célèbre pour sa légèreté, son extensibilité et son intégration avec Git.

**Volume** : En Kubernetes, un Volume est une unité de stockage attachée à un Pod, permettant de conserver les données au-delà de la durée de vie d’un conteneur spécifique au sein du Pod. Les Volumes peuvent être de types différents (locaux, cloud, etc.) selon les besoins de stockage.

## React

**React** : Bibliothèque JavaScript développée par Facebook pour créer des interfaces utilisateur interactives de manière efficace. (Documentation : <https://fr.react.dev/> )

## TerraForm

**Terraform** : Outil d'Infrastructure as Code (IaC) permettant de construire, modifier et versionner l'infrastructure de manière sécurisée et efficace. (Documentation : <https://www.terraform.io/> )

**Provider :** Les providers sont des plugins Terraform qui permettent de gérer les ressources de différents services cloud (comme AWS, Azure, Google Cloud, etc.), ainsi que d'autres services comme Kubernetes, SQL databases, etc. Chaque provider offre une interface pour un ensemble spécifique de services et de ressources.

**Resource** : Les ressources sont les éléments les plus fondamentaux dans les configurations Terraform. Elles représentent un composant de votre infrastructure, tel qu'une instance de machine virtuelle, un conteneur, ou une règle de sécurité réseau.

**Module** : Un module est un conteneur pour plusieurs ressources qui sont utilisées ensemble. Les modules peuvent être utilisés pour encapsuler et réutiliser des parties de l'infrastructure à travers différents projets ou environnements.

**State** : Le fichier d'état de Terraform (terraform.tfstate) garde trace de l'état actuel de toutes les ressources gérées par Terraform. Il permet à Terraform de savoir ce qui est actuellement déployé et de gérer les changements à l'infrastructure.

**Plan** : Un "plan" Terraform est une étape où Terraform calcule les changements nécessaires pour atteindre l'état désiré défini par les fichiers de configuration. C'est une prévisualisation des actions que Terraform exécutera.

**Apply** : La commande terraform apply applique les changements définis par le plan pour atteindre l'état souhaité. Elle modifie l'infrastructure pour correspondre à l'état défini par les fichiers de configuration.

**Destroy** : La commande terraform destroy est utilisée pour supprimer toutes les ressources gérées par Terraform.

**Backend :** Les backends définissent où et comment Terraform stocke l'état de l'infrastructure. Ils peuvent également gérer le verrouillage pour permettre à des équipes de collaborer sur la même infrastructure sans conflits.

**Workspace**: Les workspaces permettent de gérer différents "environnements" tels que le développement, la production, et les tests au sein de la même configuration Terraform, en utilisant des états séparés pour chaque workspace.

1**Output** : markdown

**Gestion des configurations (ConfigMaps, Secrets)**

Méthodes de gestion des configurations et des secrets dans l'environnement de production.

# Architecture globale

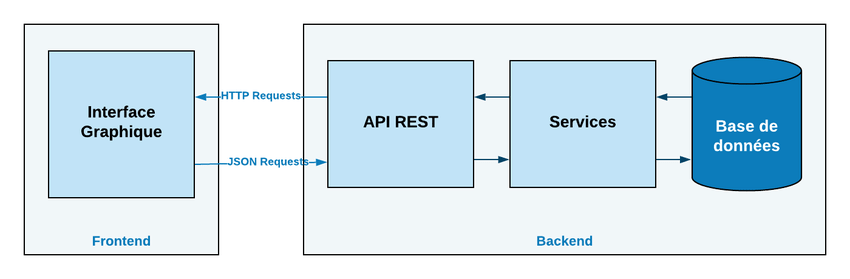
## Architecture de haut niveau

Présentation schématique des principaux composants de l'application et de leur interaction.

<https://www.dev.test.updigital.ardigitalconsulting.io/project/show/4/>

## Interactions entre les composants

Explication de la communication entre les composants, y compris les APIs et les services externes.



*Tableau 2 - Interactions entre les composants*

* FrontEnd React
* Backend Django
* Base de données PostgreSQL

## Gestion des données

### Informations générales

* Type : PostGreSQL
* Nom : Updigital
* User : Updigital / Postgres

### Configuration des accès

La structure de la base de données est accessible depuis PgAdmin depuis VSCode en passant la commande :

kubectl port-forward pods/updigital-dev-pg-0 55432:5432 -n updigital-dev

*Tableau 6 - Redirection de la base vers un port local*

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

*Tableau 7 - Configuration Connexion PostGreSQL #1*

**Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement**

*Tableau 8 - Configuration Connexion PostGreSQL #2*

**Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement**

*Tableau 9 - Accès aux tables*

### Schéma de la base

Structure de la base de données, schémas, tables et relations.

L’outil ERD Tool sur PgAdmin donne une visualisation de la base avec ses relations.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

*Tableau 10 - ERD Tool*

Une image contenant texte, capture d’écran, diagramme, Parallèle

Description générée automatiquement

*Tableau 11 - Schéma de la BDD*

# Processus infrastructure

## Données d’entrée

* Un environnement de développement servant de bac à sable pour les développeurs,
* Un environnement de staging, contenant l’ensemble des tests fonctionnels.

## Données de sortie

* Environnements Dev et staging
* Configuration du serveur de base de données
* Configuration du serveur hébergeant l’infrastructure Kubernetes
* Processus de déploiement de l’infrastructure Kubernetes pour chaque environnement

## Dossier infrastructure

Le dépôt pour la configuration de l’infrastructure est <https://github.com/Rony-44/updigital-ops>

## Déploiement de l’infrastructure



## Configuration AWS

Sur s3, Amazon Simple Storage Service, contient deux types d’objets stockés :

* bucket-database-backup-test
* bucket-terraform-state-test

**bucket-database-backup-test**

Contient les sauvegardes de la base de données (schéma, données)

**bucket-terraform-state-test**

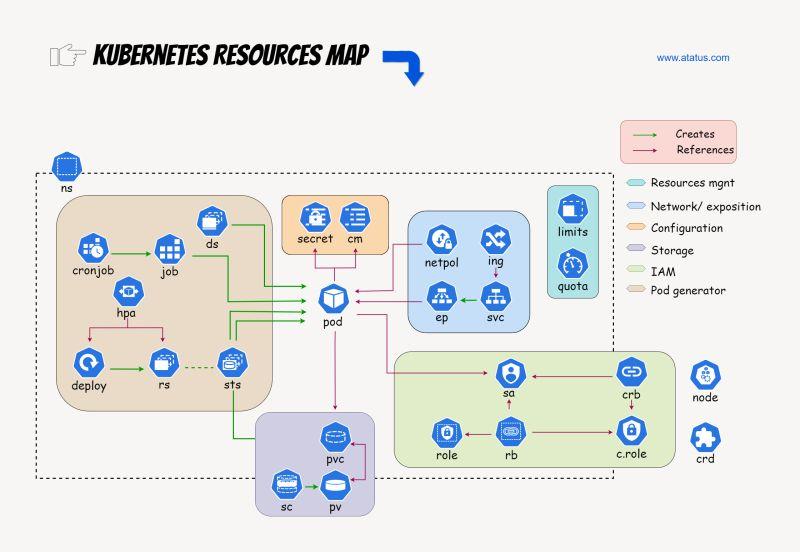
Stocke les sauvegarde du fichier terraform.tfstate.

Voici les points clés concernant le fichier terraform.tfstate :

1. **État de l'Infrastructure** : Le fichier terraform.tfstate contient l'état actuel de l'infrastructure gérée par Terraform. Cela inclut non seulement les configurations et les paramètres de chaque ressource, mais aussi des métadonnées supplémentaires nécessaires à la gestion de ces ressources.
2. **Mapping des Ressources** : Terraform utilise ce fichier pour mapper les ressources définies dans votre code Terraform aux instances réelles dans votre fournisseur de services cloud (comme AWS, Google Cloud, Azure, etc.). Cela permet à Terraform de savoir ce qui a été déployé et comment cela correspond aux configurations définies dans votre code.
3. **Synchronisation de l'État** : À chaque exécution de Terraform (terraform apply), Terraform met à jour ce fichier pour refléter l'état actuel de l'infrastructure après l'application des changements. Si des ressources sont ajoutées, modifiées ou supprimées, ces modifications sont reflétées dans le fichier d'état.
4. **Gestion des Conflits** : Le fichier d'état aide également à gérer les conflits et à prévenir les conditions de course lorsque plusieurs utilisateurs ou processus tentent de modifier l'infrastructure en même temps. Il permet de s'assurer que chaque changement est appliqué de manière cohérente et sécurisée.
5. **Stockage Sécurisé** : Il est recommandé de stocker le fichier terraform.tfstate dans un emplacement sécurisé et potentiellement partagé, comme un bucket S3 sur AWS avec le versioning activé, pour permettre une collaboration sûre et la récupération en cas de perte de données.
6. **Sensibilité des Données** : Étant donné que le fichier d'état peut contenir des informations sensibles, il est crucial de le manipuler avec soin, en utilisant des mécanismes de chiffrement et en limitant l'accès via des contrôles d'accès appropriés.

## Configuration Kubernetes

Général





### Namespaces

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

### Nodes

Une image contenant texte, capture d’écran, menu, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

### Storage

* PVC
* pvc-xxx
* PV
* Pgdata-updigital-dev-pg-0
* Pgdata-updigital-dev-pg-1
* Storage Class
* Local-path
* Longhorn

### NetWork

#### Services

* Services
* Updigital-frontend-svc
* Updigital-dev-pg
* Updigital-dev-pg-repl
* Updigital-dev-pg-config
* Updigital-backend-svc

#### Ingress

#### EndPoints

* Updigital-frontend-svc
* Updigital-dev-pg
* Updigital-dev-pg-repl
* Updigital-dev-pg-config
* Updigital-backend-svc

### Config maps

* ConfigMap
* Kube-root—ca.crt
* Updigital-frontend-cm
* Secrets
* aws-user-secret
* sh.helm.realease.v1.updigital-frontend.vxx
* updigital-backend-tls
* updigital-frontend-tls
* updigital.updigital-dev-pg.credentials.postgresql
* postgres.updigital-dev-pg.credentials.postgresql
* standby.updigital-dev-pg.credentials.postgresql
* updigital-backend-gitlab-registry-secret
* updigital-frontend-gitlab-registry-secret

### Pods



#### Pods système

#### Pods Base de données

Dans cette section, expliquer comment sont créés les pods de la base de données afin de pouvoir maintenir la configuration, la faire évoluer si nécessaire.

#### Pods applicatifs

#### Pods gitlab

**Stratégies de déploiement (Kubernetes, Docker)**

Processus et outils utilisés pour le déploiement continu et la gestion des conteneurs via via GitLab CI/CD.

Une image contenant texte, Police, capture d’écran, conception

Description générée automatiquement

*Tableau 12 - Dossier CI/CD*

* Fichier dockerfile sous cicd/docker
* .gitlab-ci.yml sous cicd/gitlab
* Dossier helm ?

## Scripts et Docker de déploiement

### Docker de déploiement

Le dépôt ops est composé de :

Du dossier permettant de construire le pod postGreOperator à partir d’un dockerfile.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

### Scripts de déploiement

Les scripts sh pour le déploiement des environnements :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* \_terraform-infrastructure-test-common.sh

Teste la présence du dossier d’infrastructure

* terraform-apply-auto-approve-infrastructure-test.sh

???

* terraform-apply-infrastructure-test.sh

Applique les changements de configuration de l’infrastructure

* terraform-destroy-infrastructure-test.sh

Détruit l’infrastructure

* terraform-init-infrastructure-test.sh

Initialise le dossier d’infrastructure

* terraform-output-infrastructure-test.sh

???

* terraform-plan-infrastructure-test.sh

Planifie l’infrastructure ?

* terraform-lint.sh

???

Dossier Common

Scripts communs à tous les serveurs

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, menu, conception

Description générée automatiquement

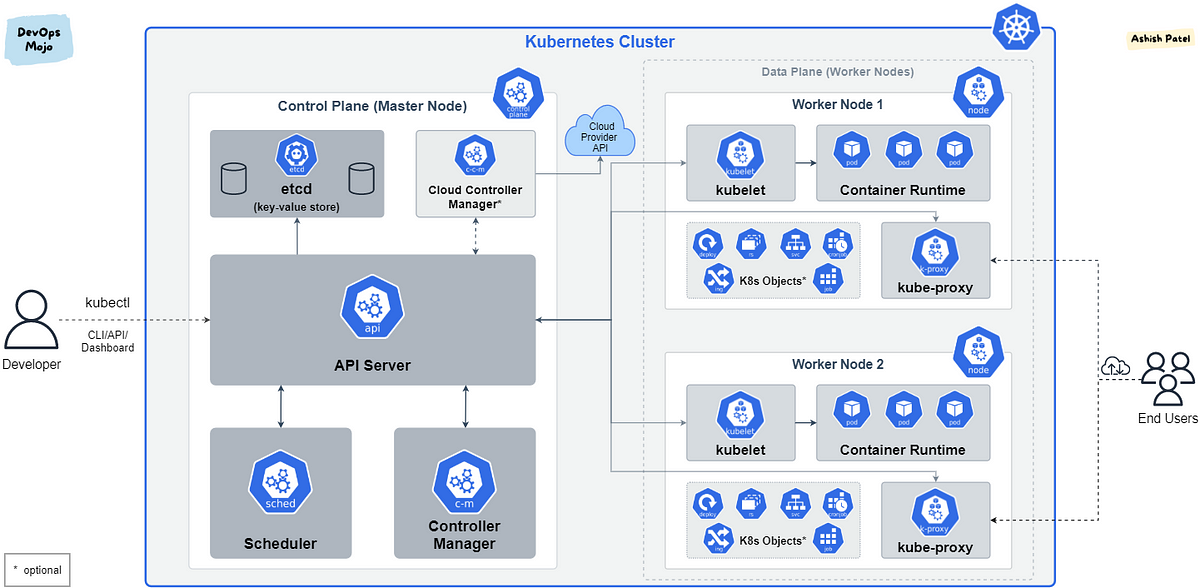
* Backend.tf
* \_locals.tf
* \_providers.tf
* \_rpi-test.tfvars
* \_variables.tf
* kubernetes-cicd.tf
* kubernetes-tools.tf
* updigital-dev.tf
* updigital-staging.tf

Modules Helm nécessaires pour faciliter la configuration des objets

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

L’application est orchestrée par un cluster Kubernetes.

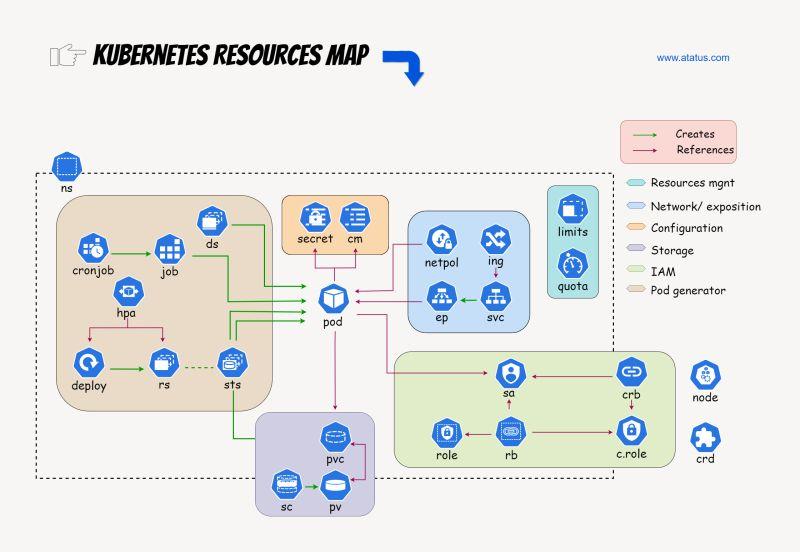


*Tableau 3 - Représentation de l’architecture d'un cluster Kubernetes*

Une image contenant texte, capture d’écran, diagramme, conception

Description générée automatiquement

*Tableau 4 - Gestion d'une requête dans Kubernetes*



*Tableau 5 - Carte des ressources Kubernetes*

**Infrastructures matérielles et réseaux**

* **Raspberry Pi5 (architecture arm64)**



* **Freebox Revolution**



**IP : 78.199.67.138**

Commande pour se connecter : ssh rony@78.199.67.138

· **Onduleur Eaton 3S**



**systemctl restart apcupsd.service**

**journalctl -u apcupsd --since "120 min ago"**

**Diagrammes d'architecture**

Diagrammes UML, séquences, déploiement, et autres visualisations pertinentes.

# Processus de déploiement de l’application

## Processus de mise à jour et de maintenance

Procédures régulières de mise à jour, de tests, et de validation de l'application via GitLab CI/CD.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

*Figure 1- étapes GiltLab CI/CD*

## FrontEnd

## BackEnd

# Sécurité

**Mécanismes de sécurité**

Techniques et outils utilisés pour sécuriser l'application et ses données.

**Gestion des accès et des autorisations**

Politiques et procédures pour la gestion des droits d'accès des utilisateurs.

# Surveillance et performance

**Outils de surveillance (Prometheus, Grafana)**

A ce jour, aucune surveillance mise en œuvre.

**Optimisation de la performance**

A ce jour, aucune stratégie et pratique d'optimisation pour garantir des performances optimales.