Documentation Technique du Déploiement

1. Introduction

Ce document décrit l'architecture de déploiement de l'application de gestion des événements, conçue pour opérer en mode Software as a Service (SaaS) avec une capacité multi-tenant. L'objectif est de fournir une vue d'ensemble complète et professionnelle des composants d'infrastructure, des services, des flux de données et des pratiques DevOps mises en œuvre pour assurer la haute disponibilité, la scalabilité et la sécurité de la plateforme. Le déploiement s'appuie sur Docker, Docker Compose pour le développement et les tests, et Docker Swarm pour la production, intégrant des techniques DevOps avancées.

2. Vue d'Ensemble de l'Architecture de Déploiement

L'architecture de déploiement est conçue pour supporter une application Laravel monolithique modulaire avec une gestion multi-tenant par base de données. Elle se compose d'un cluster Docker Swarm pour l'orchestration des conteneurs, d'un reverse proxy pour la gestion du trafic et des sous-domaines, de services applicatifs conteneurisés, de bases de données dédiées (centrale et par tenant), et de services transversaux pour le CI/CD, le monitoring, le logging, la sauvegarde et la gestion des secrets.

3. Composants de l'Architecture

3.1. Cluster Docker Swarm

Le cœur de l'infrastructure de déploiement est un cluster Docker Swarm. Il permet l'orchestration, la scalabilité et la haute disponibilité des services conteneurisés. Le cluster est composé de :

- **Nœuds Manager :** Responsables de la gestion du cluster, de la planification des tâches et de la maintenance de l'état souhaité des services. Pour la haute disponibilité, il est recommandé d'avoir un nombre impair de nœuds manager (ex: 3 ou 5).

- **Nœuds Worker :** Exécutent les conteneurs des services déployés. Ils peuvent être ajoutés ou retirés dynamiquement pour ajuster la capacité du cluster.

3.2. Load Balancer / Reverse Proxy (Traefik / Nginx)

Un composant essentiel en frontal du cluster Swarm est un Load Balancer/Reverse Proxy. Il est responsable de :

- **Routage du trafic :** Dirige les requêtes entrantes vers les services Docker appropriés.
- **Gestion des sous-domaines :** Pour l'architecture multi-tenant, il gère le routage des requêtes basées sur les sous-domaines vers les instances de l'application Laravel correspondant à chaque organisateur.
- **Terminaison SSL/TLS :** Gère les certificats SSL pour sécuriser les communications.

3.3. Services Applicatifs Conteneurisés

L'application Laravel est décomposée en plusieurs services Docker pour une meilleure gestion et scalabilité :

- **Service Web (Nginx) :** Sert les fichiers statiques et agit comme un proxy inverse pour le service PHP-FPM, transmettant les requêtes dynamiques.
- **Service PHP-FPM :** Exécute le code PHP de l'application Laravel. Il est configuré pour communiquer avec Nginx et les bases de données.
- **Service Redis :** Utilisé comme cache pour améliorer les performances de l'application et comme gestionnaire de files d'attente pour les tâches asynchrones.
- **Service Queue Worker :** Un ou plusieurs conteneurs dédiés à l'exécution des tâches en arrière-plan de Laravel (ex: synchronisation des ressources Stancl Tenancy, envoi d'e-mails, traitement d'images). Cela permet de décharger le service PHP-FPM et d'assurer la réactivité de l'interface utilisateur.

3.4. Bases de Données

L'architecture de base de données est cruciale pour le multi-tenancy :

- **Base de Données Centrale (PostgreSQL) :** Une instance PostgreSQL unique qui stocke les données globales de l'application, telles que les informations sur les SuperAdmins, Admins, Organizers et les configurations des Organizations (tenants). Elle contient également les informations nécessaires à Stancl Tenancy pour la gestion des tenants.
- **Bases de Données des Tenants (PostgreSQL) :** Chaque `Organization` (tenant) possède sa propre base de données PostgreSQL isolée. Cela garantit une isolation

stricte des données entre les tenants, une meilleure sécurité et une scalabilité horizontale facilitée. Le provisionnement de ces bases de données est automatisé lors de la création d'un nouvel organisateur.

3.5. Interfaces Utilisateur

- **Application Web :** Accessible via un navigateur web, elle regroupe les interfaces SuperAdmin, Admin et Organizers. Les requêtes sont acheminées via le Load Balancer.
- **Application Mobile Flutter :** Communique avec l'API RESTful de l'application Laravel. Les requêtes API passent également par le Load Balancer/API Gateway.

4. Flux de Données et Interactions

Le diagramme de déploiement illustre les flux de données et les interactions entre les différents composants :

- **Requêtes Web (SuperAdmin, Admin, Organizers) :** Les utilisateurs accèdent à l'application via leur navigateur web. Les requêtes sont d'abord reçues par le Load Balancer/Reverse Proxy, qui les achemine ensuite vers le service Nginx approprié au sein du cluster Docker Swarm. Nginx transmet les requêtes dynamiques au service PHP-FPM.
- **Requêtes API (Application Mobile Flutter) :** L'application mobile envoie des requêtes à l'API de l'application Laravel. Ces requêtes passent également par le Load Balancer/API Gateway, qui les dirige vers le service Nginx, puis vers PHP-FPM.
- **Communication PHP-FPM - Bases de Données :** Le service PHP-FPM interagit avec la base de données centrale pour les opérations globales (authentification, gestion des utilisateurs centraux, etc.) et avec les bases de données de tenants pour les opérations spécifiques à chaque organisation.
- **Communication PHP-FPM - Redis :** PHP-FPM utilise Redis pour le caching et pour l'envoi de tâches aux files d'attente.
- **Queue Workers :** Les services Queue Worker surveillent en permanence les files d'attente Redis et exécutent les tâches asynchrones (ex: synchronisation des données entre la base centrale et les bases de tenants, envoi de notifications).
- **Gestion des Sous-domaines :** Le Load Balancer est configuré pour router les requêtes basées sur les sous-domaines vers l'instance Laravel correspondante, permettant ainsi l'accès aux espaces dédiés de chaque organisateur.

5. Aspects DevOps et Bonnes Pratiques

Pour garantir un déploiement robuste, scalable et maintenable, plusieurs pratiques et outils DevOps sont intégrés :

5.1. Intégration et Déploiement Continus (CI/CD)

Un pipeline CI/CD est essentiel pour automatiser le processus de build, de test et de déploiement de l'application. Il assure des mises à jour rapides et fiables. Les étapes typiques incluent :

- **Build :** Création des images Docker de l'application (Nginx, PHP-FPM).
- **Tests :** Exécution des tests unitaires, fonctionnels et d'intégration.
- **Déploiement :** Mise à jour des services dans le cluster Docker Swarm via des commandes `docker stack deploy`.
- **Gestion des migrations de bases de données :** Le pipeline doit gérer l'application des migrations sur la base de données centrale et sur les bases de données de tous les tenants.

5.2. Monitoring

Le monitoring est crucial pour surveiller la santé et les performances de l'application et de l'infrastructure. Des outils comme Prometheus et Grafana peuvent être utilisés pour :

- **Collecte de métriques :** CPU, mémoire, I/O disque, trafic réseau pour les nœuds Docker Swarm et les conteneurs.
- **Performances applicatives :** Temps de réponse des requêtes, erreurs, utilisation des ressources par les services PHP-FPM et Redis.
- **Performances des bases de données :** Connexions actives, requêtes lentes, utilisation du disque.
- **Alerting :** Configuration d'alertes en cas de dépassement de seuils ou d'anomalies.

5.3. Centralisation des Logs

La centralisation des logs facilite le débogage et l'analyse des problèmes. Une stack ELK (Elasticsearch, Logstash, Kibana) ou équivalent peut être mise en place pour :

- **Collecte :** Récupération des logs de tous les conteneurs Docker (Nginx, PHP-FPM, Queue Workers, bases de données).
- **Stockage :** Persistance des logs dans un système centralisé.
- **Analyse et Visualisation :** Recherche, filtrage et visualisation des logs pour identifier les tendances et les erreurs.

5.4. Sauvegarde et Restauration

Des stratégies robustes de sauvegarde et de restauration sont indispensables pour protéger les données :

- **Bases de Données Centrales et de Tenants :** Mises en place de sauvegardes régulières (journalières, horaires) des bases de données. Utilisation de volumes persistants pour les données des bases de données.
- **Stratégies de restauration :** Définition de procédures claires pour la restauration des bases de données en cas de sinistre.

5.5. Gestion des Secrets

La gestion sécurisée des informations sensibles (mots de passe de bases de données, clés API, identifiants de services tiers) est primordiale. Docker Swarm offre un mécanisme de gestion des secrets qui permet de :

- **Stocker les secrets de manière chiffrée :** Les secrets sont chiffrés au repos et en transit.
- **Distribuer les secrets aux services :** Seuls les services autorisés ont accès aux secrets nécessaires, et ils sont montés en tant que fichiers temporaires dans les conteneurs, évitant ainsi leur exposition dans les variables d'environnement ou les images Docker.

5.6. Volumes Persistants

Les volumes Docker sont utilisés pour persister les données générées par les conteneurs, notamment :

- **Données des bases de données :** Les données des bases de données centrales et de tenants sont stockées sur des volumes persistants pour garantir leur survie au-delà du cycle de vie des conteneurs.
- **Fichiers médias :** Les fichiers téléchargés par les utilisateurs (images, documents) sont également stockés sur des volumes persistants.

5.7. Réseaux Overlay Docker

Les réseaux overlay Docker sont utilisés pour permettre la communication entre les services sur différents nœuds du cluster Swarm. Ils offrent :

- **Isolation réseau :** Chaque service peut être placé sur son propre réseau, limitant ainsi la communication non autorisée.

- **Découverte de services :** Les services peuvent se découvrir mutuellement par leur nom au sein du réseau overlay.
- **Sécurité :** Le trafic sur les réseaux overlay est chiffré par défaut.

6. Conclusion

Cette architecture de déploiement, basée sur Docker Swarm et intégrant des pratiques DevOps robustes, offre une solution complète et performante pour l'application de gestion des événements multi-tenant. Elle assure la haute disponibilité, la scalabilité, la sécurité et la maintenabilité de la plateforme, tout en facilitant le développement et les opérations. Le diagramme de déploiement associé fournit une représentation visuelle claire de cette architecture, servant de référence pour toutes les parties prenantes du projet.