Cahier des charges technique

Tom NIRRENGARTEN - Romain DESERT

15 Octobre 2025

Contexte et rappel du sujet

Depuis 20 ans, les Jardins de Cocagne utilisent un logiciel basé sur Microsoft Access. Ce dernier permet notamment la gestion des adhérents, la gestions des abonnements et des règlements, ainsi que le suivis et la livraison des paniers. Bien qu'il soit encore majoritairement utilisé, les trois quarts des Jardins de Cocagne souhaitent qu'un nouvel outil remplace l'existant pour correspondre au mieux aux nouveaux besoins.

Ainsi, nous avons pour mission la réalisation d'une nouvelle solution moderne pouvant évoluer pour toujours mieux répondre aux nouveaux besoins dans la gestion de leurs activités. L'outil prendra la forme d'un site web sécurisé, qui respectera les diverses obligations légales dans la protection des données.

Notre équipe

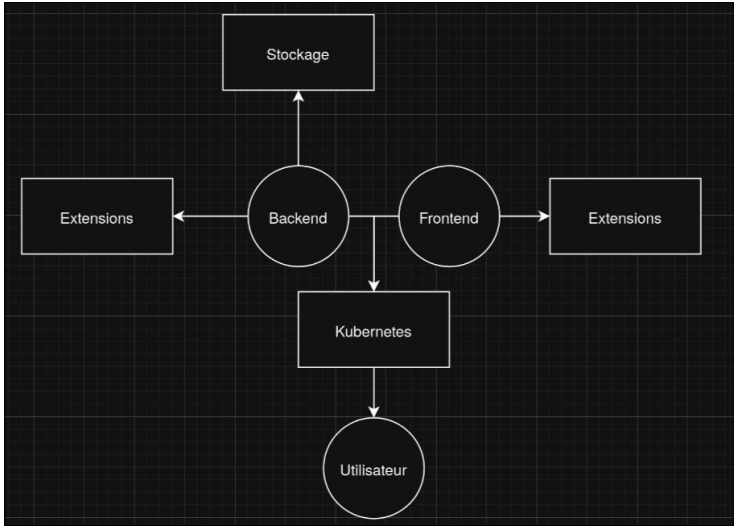
- Romain DESERT : Développeur Fullstack, connaissances sur les bases de données, la conteneurisation et le versionning
- Tom NIRRENGARTEN : Développeur Fullstack, connaissances sur les services AWS et le Cloud-Computing

Notre organisation

- Notre code sera publié sur un repository Github
- L’onglets “Projets” de Github sera utilisé pour gérer nos sprint
- Nous utiliserons un système de branches ainsi que de pull request stricte pour éviter les problèmes
- Chaque Pull Request devra être accepté à l'unanimité par l’équipe
- Nous mettrons en place des test unitaires durant le processus de développement. Les pull request ne pourront être validé par l’équipe que si les tests passent.
- L’onglet Issue sera mis à contribution pour noter tout bug découvert, et un bug devrait être résolu dans une branche dédié à ce dernier

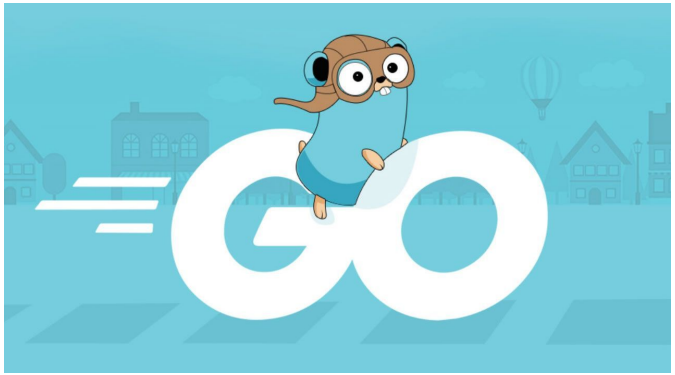
Application principale

- Nous avons un backend et un frontend qui sont reliée à un K8s
- Le serveur backend utilise des système de stockage (base de donnée et s3)
- Nous pouvons également appliquer des extensions sur le frontend et backend du serveur
- L’architecture Kubernetes a été choisie pour sa haute robustesse et sa facilité de déploiement (tout est fait en YML), celle-ci correspond mieux qu’une architecture serveur classique car elle permet d’isoler les conteneurs afin de ne pas avoir de risque associée au module utilisateurs.



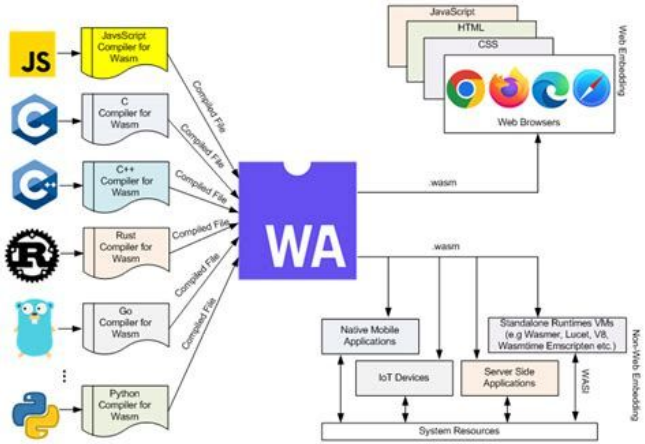
Backend

- Notre application sera écrite en Go en utilisant le framework Echo
- Go est un langage récent est puissant gagnant de plus en plus de popularité
- Il a été créé par Google pour gérer des demande forte en utilisant le moins de performance.
- Il a été choisi pour sa haute performance et ses faibles demande de ressource tout en ayant une syntaxe facile. Beaucoup d'autres langages existe mais peu arrive à un tel niveau de performance ce qui a poussé notre choix.



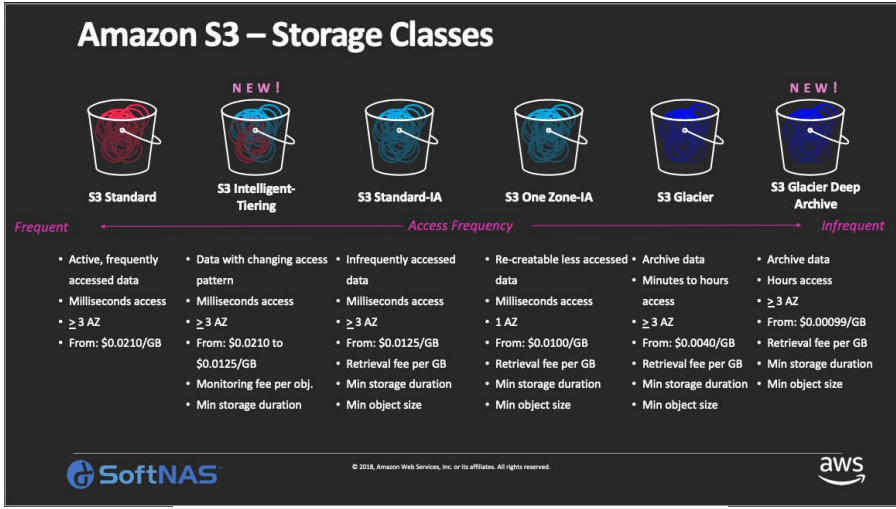
Backend

- Afin d'avoir un site modulable tout en gardant un niveau de sécurité nous avons décidé d'utiliser des extensions en WASM (Web Assembly). Celle-ci peuvent être développée en n'importe quelle langues (pour aider les développeurs) et exporter des fonctions appelée par Go (via par exemple Wazero).
- Les WASM sont fait pour être isolée comme un site web ce qui permet un haut niveau de protection pour les fonctionnalités basiques
- Les fonctionnalités avancée requièrent tout de même des modifications en Go
- Nous avons choisi cette technologie plutôt que des modifications entièrement sur le code source go pour sa modularité ainsi que son isolement (certaines fonction sont exportées mais le reste du système tourne dans une sandbox)



Stockage des données

- Notre priorité est la fiabilité de nos données, celle-ci ne doivent pas être perdu et doivent avoir des copies fréquentes.
- Nous avons décidé de partir sur l'écosystème cloud AWS car c'est celui que nous connaissons le mieux. Celui-ci permet egalement un systeme efficace d'archive ainsi que des prix bas. Un système de stockage basique ne nous aurait pas permis un tel niveau de sécurité des données et d'archivage.
- Pour le stockage de fichier brut (blob storage) nous allons utiliser un système S3 avec un archivage des données afin d'avoir des copies régulière des données.



Stockage des données

- Pour le stockage de nos données logicielle nous allons utiliser DynamoDB pour sa scalabilité horizontale et son approche de sauvegarde automatique off-site.
- Celle-ci possède également un support pour les données GPS via le plugin “Geo Library for Amazon DynamoDB”
- Celui-ci peut automatiquement augmenter en capacité tout en respectant le modèle “pay as you use” (payer uniquement pour le stockage que vous utilisez)
- Pour nous alerter de toutes anomalies dans nos requête nous allons utiliser AWS CloudWatch
- Nous avons choisi DynamoDB pour son prix très faible et son modèle “pay as you use”, modele non applicable dans des instances RDS PostgreSQL qui sont beaucoup plus cher. Nous avons décidé de stocker nos données dans le cloud pour la même problématique que S3. CloudWatch s'inscrit dans l'écosystème AWS ce qui a poussé notre choix.

Amazon DynamoDB

Amazon Cloudwatch

Notification utilisateur

- Pour communiquer avec les utilisateur (Email et SMS) nous allons utiliser AWS SES et SNS
- Ces services nous permette d'envoyer des notification/mails via REST (ou SMTP pour les mails) tout en n'ayant pas de problème de réputation avec les piscines d'IP gérée par AWS
- Nous avons utilisé ces services car les IP/Numéro de téléphones change fréquemment ce qui permet de ne pas tomber dans les SPAM des clients, d'autre service comme SendInBlue propose aussi ces services mais a des prix beaucoup plus élevée.

AWS SES

AWS SNS

Evolution AWS

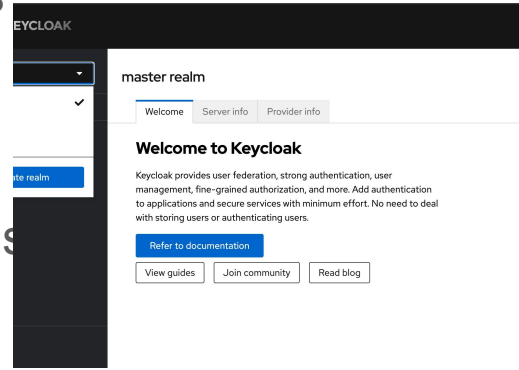
- AWS via le biai d'IAM permet de faire des groupes et des comptes utilisateurs
- Cela permet de faire des groupes et créer des utilisateurs pour les donner au responsables de l'organisation
- Cette solution a été choisi car elle est intégrée de base dans AWS et qu'il n'y a pas vraiment d'alternatives

Paiement

- Pour effectuer les paiement en ligne nous utiliserons Stripe.
- Cette solution est très commune mais prend malheureusement des commissions, ceci peut être problématique avec le client.
- Cette solution a été choisie pour son efficacité et son modèle de paiement alléchant, d'autant plus stripe propose également des terminaux de paiement électronique pour aider les distributeur de panier à collecter les payement physique s'il le faut. Square aurait également été possible cependant les prix sont plus cher

Authentification

- Pour avoir un système sécurisé supportant l'oauth2 et un système de rôle nous allons utiliser Keycloak
- Celui ci permet aux administrateurs de gérer les rôles des utilisateurs ainsi que leur groupe sur une interface graphique simple
- Cette solution OpenSource permet d'avoir un moyen simple, sécurisé et centralisé de s'occuper des connexions et des permissions utilisateur
- Nous avons choisi cette suite plutôt que Auth0 car le système de rôle est très efficace et que nous avions déjà de l'expérience avec ce logiciel.



Frontend

- Pour le frontend nous allons utiliser le framework js Vue avec l’outil de build Vite
- Vue est un framework populaire permettant de faire du frontend avec des composants. Celui-ci est hautement modulaire et autorise l'utilisateur a facilement l'éditer.
- Vite nous permet de build le site de manière statique afin de pouvoir le distribuer de manière rapide avec un serveur comme NGINX.
- Pour créer des pages simples nous utiliserons CKEditor afin de manipuler le HTML de manière graphique
- Nous avons choisi Vue pour sa communauté très large ainsi que sa simplicité d’utilisation comparée à d’autres framework comme ReactJS

Frontend

- Nous allons utiliser Vue Router pour les routes, celle-ci seront gérée entièrement côté client afin de réduire le temps de chargement du site
- Pour implémenter un système de state et de gérer les variables en temps réel nous allons utiliser Pinia, celle-ci étant la solution la plus commune pour gérer ce problème
- Nous avons choisi Vue Router et Pinia car ceux-ci sont les librairies les plus communément utiliser pour leur fonctions

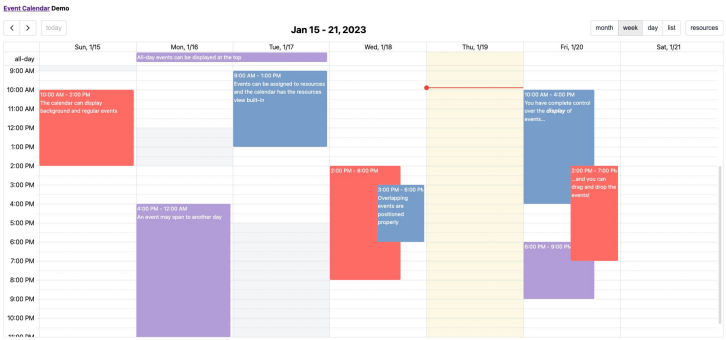
Vue Router

Cartographie

- Pour une carte simple nous allons utiliser leaflet avec son integration VueJS. Celui ci permet de poser simplement des points sur une carte Openstreetmaps
- Pour les tournées nous allons utiliser l'API Circuit pour nous faire des routes optimisée en passant par tous les points nécessaire pour ensuite les exporter vers des applis de GPS
- Leaflet a été choisi pour son caractère OpenSource et sa grande utilisation et Circuit a é choisi pour sa simplicité d'utilisation et nos expériences personnels

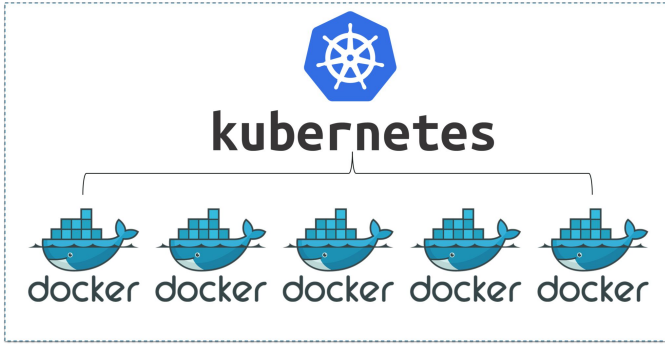
Calendrier

- Pour l’affichage des calendriers nous utiliserons la librairie EventCalendar de vkurko pour son aspect graphique simple pour l’utilisateur
- Pour l’import/export des calendriers personnel pour simplifier l’utilisation nous passeront par des fichiers ICAL. Ceux-ci peuvent souvent être chargé par URL ce qui aide à avoir un calendrier en temps réel sur l’application favorite de l’utilisateur



Hosting

- Nous avons décidé de partir sur des conteneurs afin d'isoler toutes les applications, celle-ci pouvant être personnalisé il ne suffit que d'un utilisateur pour faire planter tous les sites, cette approche permet d'isoler les organisation et de les faire marcher indépendamment
- Pour gérer tout nos conteneurs nous allons utiliser Kubernetes afin d'avoir une manière centralisée de contrôler tous nos déploiement ainsi que pour réduire le temps de panne au maximum

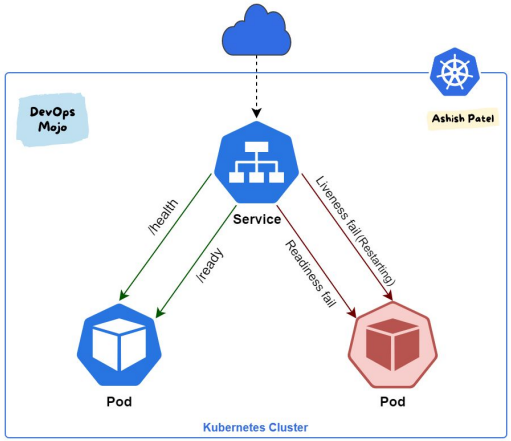


Hosting

- Pour héberger nos conteneurs nous allons utiliser EKS de AWS, il s'agit d'un runtime docker permettant automatiquement s'agrandir horizontalement sur des instances EC2 (VPS) ou en Serverless afin d'avoir toujours assez de puissance disponible.
- Pour ne pas ouvrir trop de machine engendrant un fort coup on utilisera AWS Billing and Cost management
- Lors d'un ajout d'un nouveau site par un administrateur un nouveau déploiement sera créé avec des paramètres d'environnement personnalisé et sera envoyer grâce au SDK Kubernetes (Celui ci étant écrit en Go)
- Les modules seront stockés sur le bucket S3 afin de pouvoir les gérer facilement ainsi qu'une configuration basique JSON du site afin de pouvoir récupérer tout les fichiers en cas de désastre
- EKS a été choisi car il s'agit d'une des solutions de déploiement Kubernetes les plus utilise. D'autres outils AWS qui seront abordés plus tard requièrent celui-ci pour fonctionner

Tests unitaire

- Go possède un système de test unitaire pré-intégré dans le code, ce qui rend sa réalisation simple.
- Une pipeline Github fera des tests automatique avant chaque Merge Request pour s'assurer de la fiabilité du code
- Vu que des modules utilisateur seront présent des tests seront également fait pour chaque nouveau déploiement, les résultats de celui-ci seront envoyés au contrôleur kubernetes qui va rollback ou non les modifications pour n'avoir aucun downtime

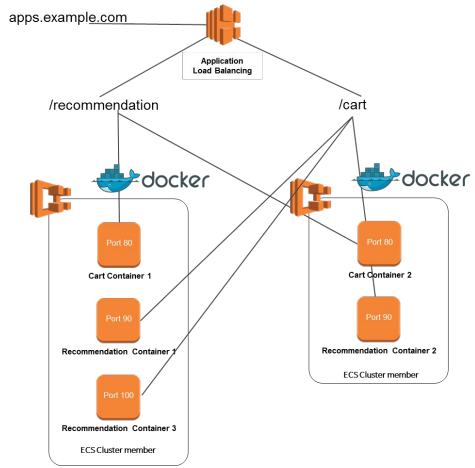


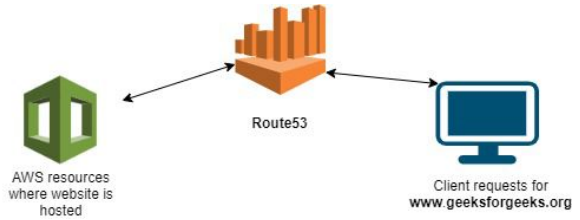
Mise à jour automatique

- Lors de chaque merge vers la branche main un build docker de l'image de base sera fait sur les actions github
- Ceux-ci seront publie sur un repository Sonatype Nexus OSS privée afin de ne pas divulguer les images clients
- On installera FluxCD afin d'automatiquement mettre à jour les images sur Kubernetes et d'avoir constamment la dernière version sans intervention manuelle.
- Kubernetes effectuant les mise a jour sur un duplicat de l'application et des essais sur les plugins étant exécutée nous n'auront aucun downtime
- FluxCD est une des solution de CI/CD les plus répandus pour pousser une image sur un repo kubernetes automatiquement, Keel aurait pu etre utilise a la place mais son approche est moins robuste que FluxCD (FluxCD contacte le repository alors que pour Keel le repository contacte Keel)
- Sonatype Nexus est un logiciel de gestion de paquets reconnu et utilisé par beaucoup (principalement pour le Java), celui-ci permet une gestion avancée des images et des permissions utilisateur pour accéder à ces images. Nous utiliserons la version OpenSource.

Accès utilisateur

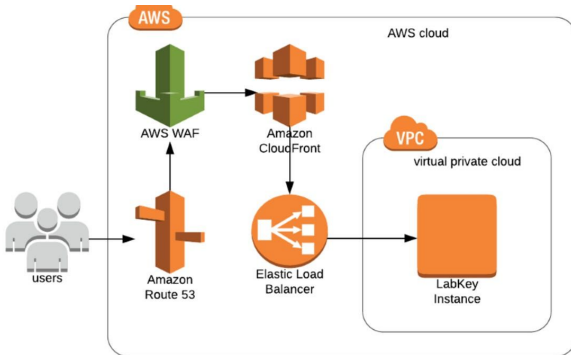
- Finalement, afin de rendre le site accessible au utilisateur on crée un Ingress sur notre cluster K8S afin de le laisser router ses connections.
- Nous allons utiliser AWS ALB afin d'avoir une bonne intégration avec nos services Amazon
- Afin de gérer facilement les domaines nous allons utiliser AWS Route 53 pour son intégration facile avec ALB. Celui-ci nous permettra de configurer rapidement nos enregistrements DNS
- Pour fluidifier l'expérience utilisateur nous allons utiliser AWS cloudfront pour mettre en cache les fichiers statique
- Ces solutions ont été utilise pour leur bonne intégration avec EKS ainsi que leur fiabilité





Sécurité

- Pour nous assurer d'éviter les attaques DDOS et les payloads malicieux nous allons utiliser AWS WAF.
- Celui-ci permet d'intercepter les requêtes et de les bloquer si elles paraissent malicieuse et/ou bloquer l'IP
- Celui-ci a été choisi pour sa fiabilité et sa facilité d'utilisation. ModSecurity aurait aussi pu être utilise avec un Ingress Nginx, cependant celui-ci ne bloque pas les DDOS



Documentation

- Pour notre documentation nous allons utiliser Read the Docs qui est une méthode simple et rapide de publier sa documentation sur un site web
- Pour la documentation des endpoints API un Swagger OpenAPI sera fait afin d’aider le debogage
- Ceux-ci ont été utilisés pour sa popularité et sa facilité de mise en place

Obligations légales

En tant que site web professionnel, nous sommes soumis à des obligation légal Française et Européen :

- Les mentions légales doivent clairement identifier l'entreprise.
- Les Conditions Générales de Ventes (CGV) doivent informer les clients sur leurs droits et obligations.
- La collecte de données doit être réalisée de manière transparente et surtout, doit être consenti par l'utilisateur. (RGPD)
- La résiliation d'un abonnement/contrat doit désormais pouvoir être réalisé par voie électronique, et non uniquement par voie postal.