Civiparoisse : présentation technique globale

(Composer, Image Docker, Chart Helm, Infrastructure Kubernetes…)

Union des Églises protestantes d'Alsace et de Lorraine

Différents livrables

- Composer : gestion des packages PHP
- Docker : encapsulation d'une release : figer une version
- Infrastructure K8S générale : ingress controller, certificats, stockage, sauvegarde…
- Chart Helm : infrastructure d'une instance

Extension CiviCRM : fr.uepalparoisse.civiparoisse

- champs spécifiques
- type de relation par rapport à une paroisse
- formulaires de création de contacts
- rapports prépackagés (SearchKit)
- widget spécifique d'affichage SearchKit (DisplayPaged)
- template de Mail (basé sur le template versafix de Mosaico)
- métriques pour la qualité des données

Autres packages logiciels spécifiques

- Plugin d’installation (pour civisetup)
- Librairie d’import de données
- Configuration des rôles avec leurs permissions associées

Stack logicielle principale

- CiviCRM
  - nécessite PHP (8.2/8/3)
  - nécessitait jusqu'à il y a peu un CMS support
  - modulaire : notion d'extensions
- CMS : Drupal (9/10)
  - Mécanismes d'imports/exports de configuration
  - Installation "scaffold" : le répertoire vendor n'est pas accessible depuis le web
  - Support de reverse proxy
- SGBD : Mysql 8(.4)
- Serveur web : Apache

Compléments tiers

- Modules Drupal :
  - role_delegation
  - drupal-l10n
  - log_stdout
- Extensions CiviCRM :
  - mosaico
  - geocoder
  - recentmenu
- Autres outils, dont :
  - drush
  - cv
  - bash

Composer

- gestionnaire de dépendances
- 2 fichiers principaux :
  - composer.json : déclaration des dépendances
  - composer.lock : verrouillage des paquets chacun dans une version définie => utile pour le SBOM, les alertes Dependabot, la reproductibilité des builds Docker (même contenu testé et releasé)
- répertoire vendor : packages récupérés + autoloader
- multiples sources de paquets :
  - sites de packages (ex : packages.drupal.org, packagist.org, ou archive privée via satis)
  - dépots de code : ex : git (si contient composer.json)
- métapackages : directement dans un composer.json, avec une référence vers le code source

Composer : plugins et exception

- Plugins utilisés :
  - composer/installers : types de packages
  - cweagans/composer-patches : patches sur les codes
  - civicrm/composer-downloads-plugin : récupération de dépendances
  - civicrm/composer-compile-plugin : compilation SCSS
  - drupal/scaffold : répartition des fichiers de code de sorte à avoir une surface exposée au web moins importante
- À l'usage, il arrive de devoir merger la configuration de plugins de composer.json de dépendances dans le composer.json principal (ex : scaffolding)
- Les plugins sont également utilisés pour le spécifique du projet
- Exception à l'utilisation de composer : récupération de cv

Gestion du code source : Github

- Github Actions : déclenchement de workflows exécutés en réponse à un évènement. Ex :
  - intégration continue : lancer des tests automatisés
  - déploiement continu : build d'image Docker, montées de versions automatisées
  - dans le futur : éventuellement mise en oeuvre d'outils de qualité de code (sonarlint/sonarqube), et d'outils d'audits (trivy)
- Alertes de sécurité Github Dependabot
- Plusieurs dépôts Git privés
- Workflow Gitflow

Introduction à Docker

- isolation d'exécution via des containers
- est comparable à un chroot survitaminé
- virtualisation légère, gérée par le kernel, notamment au niveau des interactions des processus entre eux, du système de fichiers, des accès réseaux...
- pas d'émulation de processeur : image par architecture (x86,x64,arm64,armv7...)
- habituellement, un seul programme est lancé dans un container, et chaque container a une fonctionnalité précise => architecture microservices
- les images contiennent l'ensemble des éléments nécessaires à l'exécution d'un programme
- les images sont read-only ; un overlay permet de faire une surcouche pour pouvoir modifier des fichiers
- les données persistantes sont à stocker dans des volumes séparés : la BD, /app/web/sites/default, /app/private

Construction de l’image Docker : Généralités

- Instructions de build dans un Dockerfile, qui est situé dans un répertoire qui devient une source de données lors de la compilation
- On peut partir d'une image / de rien (image scratch vide) - dans le cas de Civiparoisse : on part d'une image php en module pour apache
- On peut récupérer des éléments d'une autre image : ex : récupération de composer depuis l'image officielle
- - On peut indiquer des valeurs de variables d'environnement par défaut, qui seront personnalisées lors d'une exécution de conteneur => notion de fichier de valeurs d'une instance
- Docker, lors du build, crée une couche d'overlay par instruction ; chaque couche peut éventuellement être réutilisée lors d'un build ultérieur, pour ne pas repartir de zéro => on peut désactiver ce comportement en utilisant des labels en tant que cache-busters

Construction de l’image Docker : Opérations réalisées

- paramétrage /ajout de fichiers de configuration
- ajout des scripts d'exploitation : install, update
- installations de packages systèmes, de modules php
- lancement du composer install : récupère et dispatche les fichiers
- création d'un utilisateur civiparoisse, qui sera un membre (groupe secondaire) de www-data ; l'utilisateur civiparoisse exécutera les opérations d'installation / d'upgrade
- positionnement des droits sur les fichiers de l'image : owner des fichiers principaux : civiparoisse ; écriture possible sur les fichiers créés par www-data via le groupe d'exécution, et l'umask adéquat => on cherche surtout à limiter les possibilités de www-data à modifier le système pour améliorer la position de sécurité
- indication des volumes
- indication du point d'entrée dans l'image
- l'image est finalement taggée, et rendue disponible via Github

Particularités de l’image Civiparoisse

- image multi-usages
- si le composer.lock ne correspond pas au composer.json, le build de l'image s'arrête et échoue => lorsqu'on fait un build local après une modification du composer.json, on supprime le composer.lock, on fait le build, puis on lance un conteneur pour récupérer le composer.lock que l'on remet dans le code source
- l'image est configurée pour faire des sorties sur stdout/stderr : ces flux peuvent ensuite être intégrés dans une infrastructure centralisée de logs
- une différence entre le traitement des volumes vides entre K8S et Docker fait qu'on prépare des copies de répertoires par défaut pour populer initialement certains volumes

Kubernetes : introduction

- permet la gestion d'un cluster de serveurs à travers de déclarations de l'état (objets) souhaités
- permet d'industrialiser l'exploitation des instances civiparoisse, en mettant à disposition des mécanismes que Docker ne propose pas (ex : cron)
- existence de services centralisés, utilisés par toutes les instances, et de services par namespaces, dédiés à chaque instance
- les noeuds peuvent être des machines physiques ou (plus généralement) virtuelles
- fournisseur choisi pour les instances Civiparoisse: OVH, avec Datacenter à Strasbourg et à Gravelines : Managed Kubernetes
- les serveurs qui font parti du cluster sont des noeuds (nodes) ; les worker nodes permettent d'exécuter les applications ; il y a aussi (c'est le cas ici) des noeuds qui gèrent le cluster lui-même : les master nodes.
- une ENS nous accompagne avec brio : Atheo.

Kubernetes : quelques notions

- pod : unité d'exécution sur un noeud
- service DNS interne du cluster
- espace de nom : permet une segmentation de ressources ; dans le projet : un namespace par instance (et donc pour le moment par paroisse) ; le namespace se retrouve dans le DNS interne => permet de se baser sur la recherche en ajoutant des suffixes de moins en moins précis pour retrouver des ressources
- Physical Volume Claim : demande d'allocation d'un volume de stockage physique, avec une classe (type) de stockage, une taille souhaitée, et mode d'accès souhaité (Read Only, Read Write Once, Read Write Many)
- service : le service permet un niveau d'abstraction géré par K8S pour envoyer du trafic adhoc vers l'IP assignée à un pod
- configmap : permet de stocker des données de configuration
- secret : permet de stocker des données qui sont sensées être confidentielles

Déploiement Continu

- Flux Continous Delivery
  - Permet d'utiliser des dépôts git comme sources pour les objets présents dans K8S => permet de déployer les applications
  - Infrastructure As Code
- Kubeseal
  - Permet d'utiliser une clef publique pour chiffrer des données secrètes, qui seront déchiffrées par K8S grâce à sa clef privée : on peut ainsi stocker des secrets chiffrés dans les dépôts git.

Trafic HTTPS

- Cert-manager
  - Client pour le certificat Let's Encrypt du wildcard
  - PKI interne pour le certificat pour le serveur web de chaque instance
  - peut être dans le futur : utilisation d'une PKI pour s'interconnecter avec AWS (IAM Roles Anywhere)
- Traefik
  - contrôleur ingress pour le trafic entrant (UDP, TCP, HTTP)
  - notion de point d'entrée, de routeur, de middleware, de transportserver => fournit beaucoup de souplesse
  - pour le moment : Traefik est en coupure, avec un certificat wildcard, et forwarde les requêtes vers un service d'un namespace en fonction du champ host
  - peut être dans le futur : SSL Passthrough et orientation par SNI.

Sauvegarde : Kasten

- sauvegarde de volumes pour les fichiers (snapshot CSI)
- sauvegarde mysqldump pour les BD (blueprint Kanister)
- un bucket S3 est utilisé pour exporter la sauvegarde
- politique de sauvegarde : 1 sauvegarde/jour pour 1 semaine ; 1 sauvegarde/ semaine pour 1 mois; 1 sauvegarde/mois pour 3 mois
- produit gratuit pendant 1 an lorsqu'on utilise jusqu'à 5 noeuds => augmentation de la taille et non du nombre de noeuds dans le projet
- la période de gratuité se renouvelle à chaque fois que l'on upgrade la version
- propose des métriques liées à l'export et à la sauvegarde

Monitoring : Kubestack

- prometheus : récupération, stockage, mise à disposition des métriques, et exécution des règles d'alertes
- alertmanager : configuration de la diffusion des alertes
- grafana : affichage des valeurs
- kubestack fournit un certain nombre d'alertes et de tableaux de bord grafana
- probablement dans le futur : loki : récupération des logs

Serveur mail de test : Mailcatcher

- Point d’entrée SMTP sur un port non standard
- Fournit une interface web pour visualiser les mails reçus
- Accès protégé par un mot de passe
- Est utilisé pour la sortie des mails de certaines instances de test.

Chart Helm Civiparoisse

- déploiement = package + fichier de valeurs de déploiement
- namespace dédié
- principaux contenus:
  - un service DB, avec son stateful set associé
  - un service PHP, avec son deployment associé et containers d'init
  - un cronjob CiviCRM et un cronjob Drupal
  - des configurations via des configmaps et des secrets
  - une ingress-route et un serverstransport: CRD pour Traefik
- stockage Cinder : Read Write Once : php et cron sur le même noeud
- après le déploiement, il reste encore quelques étapes manuelles, qui n'ont pas encore été automatisées (ex : personnalisation de template mails)

Conclusion : engagez-vous dans le projet !

- l'industrialisation des couches de livrables permet la fourniture de Civiparoisse aux paroisses
- le code est amené à évoluer et s'adapter
- équipe technique trop restreinte actuellement => problème critique
- **Venez mouiller le maillot avec nous !**
- Merci pour votre attention

Union des Églises protestantes d'Alsace et de Lorraine