Kubernetes, système d'exploitation de PC-SCOL et PÉGASE

Vincent HURTEVENT
PC-SCOL / AMUE – Ops
UCBL, Bâtiment Lippmann
43 Bd du 11 novembre 1918
69622 Villeurbanne

Raymond BOURGES
PC-SCOL / Association Cocktail – Directeur technique
INSA de Rennes, Bâtiment 18
20 avenue des Buttes de Coësmes
35708 Rennes

Thomas MONTFORT
PC-SCOL / AMUE – Ops
SIMSU, Domaine universitaire
31 rue des mathématiques
38400 Saint-Martin-d'Hères

Résumé

La construction de la solution PÉGASE, futur logiciel pour la formation, la scolarité et la vie étudiante, utilise plusieurs principes novateurs dans le développement logiciel et la gestion d'infrastructure. Il s'agit tout d'abord d'une co-construction en méthode agile par plusieurs établissements (AMUE, Association Cocktail, les universités de Grenoble Alpes, Nantes et Strasbourg), et d'un point de vue architecture, d'une approche très modulaire, faiblement couplée et *API first*. Ces choix s'accompagnent de contraintes fortes auxquelles il faut répondre.

La méthode agile à l'échelle implique des cycles courts et des déploiements de plusieurs environnements, dans plusieurs versions plusieurs fois par jour.

L'approche modulaire demande un couplage faible des modules et, pour chacun d'eux, un découpage en plusieurs composants : front, back portant l'API et stockage.

À l'échelle de PÉGASE, cette approche va nécessiter un nombre important de modules et un nombre plus important encore de composants. Ces choix ont dès le début incité à considérer PÉGASE comme une application *cloud native*.

Les objectifs ambitieux de l'application ont trouvé écho dans les principes d'immutabilité, d'auto correction, d'élasticité, de haute disponibilité ou bien encore de portabilité que permettent les conteneurs et l'orchestrateur Kubernetes.

Nous verrons comment l'utilisation de Kubernetes et des bonnes pratiques s'y rattachant simplifient le développement de PÉGASE par les équipes de développement et son exploitation par l'équipe DevOps. Nous aborderons ensuite les limitations et contraintes que nous avons rencontrées. Enfin nous exposerons pourquoi tout cela nous amène à considérer Kubernetes comme le système d'exploitation de PÉGASE et par extension d'un SI.

Mots-clefs

Co-construction, application distribuée, cloud native, orchestration, Kubernetes, Helm, IaaS, SaaS

1 Introduction

Après une rapide présentation du projet PC-SCOL et de son fonctionnement, nous nous attarderons sur les objectifs de PÉGASE qui vont grandement influencer les méthodes et principes retenus pour sa construction, jusqu'à son architecture technique. Nous verrons comment Kubernetes, de par son histoire et ses mécanismes, convient particulièrement au projet.

Enfin nous conclurons sur les avantages de cette technologie et ce qu'elle pourrait apporter à la communauté dans un contexte de portabilité et de co-construction.

2 PC-SCOL

2.1 Le projet

Le projet PC-SCOL est né en 2017. Il est le fruit d'une convention signée entre deux organisations éditrices et distributrices de logiciels pour l'Enseignement Supérieur et la Recherche (ESR) : l'Agence de Mutualisation des Universités et des Établissements (AMUE) et l'Association Cocktail.

Son objectif est la refonte du système d'information (SI) de la formation, de la scolarité et de la vie étudiante (FSVE) au sein des établissements adhérents. Cette refonte est matérialisée par la construction de l'application PÉGASE. Elle est destinée à remplacer les outils existants (Apogée, SVE et Scolarix, ROF, etc). Elle couvrira le périmètre métier de chacun d'eux et pour satisfaire de nouveaux usages et besoins, la moitié du projet concerne de nouvelles fonctionnalités.

PC-SCOL s'inscrit dans le cadre de cohérence des systèmes d'information[1] qui invite à la mutualisation et à la co-construction avec les établissements de l'ESR. Les universités Université Grenoble Alpes, Université de Nantes et Université de Strasbourg ont rejoint le projet ces 2 dernières années pour la construction de PÉGASE.

2.2 Organisation & Agilité

La construction de PÉGASE au sein de PC-SCOL est organisée en 8 équipes :

- 1 équipe de direction (itinérante) ;
- 1 équipe de consultants fonctionnels (Montpellier) ;
- 4 équipes de production (Grenoble, Nantes, Strasbourg, Toulouse) ;
- 1 équipe de qualification (Montpellier) ;
- 1 équipe système (Architecte, Urbanistes, DBA, Ops, répartis en France).

La gestion de projet au sein de chacune des équipes est faite de manière agile. Ces équipes, de taille modeste selon le concept de la *pizza team*[2] travaillent avec la méthode Scrum[3]. PÉGASE est pensé comme modulaire et découpé par périmètre métier. Aussi, le développement d'un module est confié à une seule équipe, qui de par ses multi-compétences internes pourra au fur et à mesure des cycles agiles, ou itérations, livrer une nouvelle version du module concerné.

Les équipes ne sont toutefois pas isolées entre elles et de nombreuses guildes [1], par technologie ou compétence, permettent la communication et la synchronisation.

Les équipes PC-SCOL fonctionnent sur des sprints de 3 semaines qui s'achèvent par une démonstration des derniers développements réalisés. Ces démonstrations sont destinées aux équipes internes PC-SCOL mais également aux établissements pilotes qui vont prochainement utiliser PÉGASE, ainsi qu'aux établissements qui mettent à disposition du projet du temps d'expertise métier de certains de leurs agents.

1. Cadre de cohérence du SI de l'ESR, volet technique : https://esr-wikis.adc.education.fr/ca2co/index.php/Modification_de_Cadres_de_cohérence_-_consultation_et_commentaires_publics

2. Quelle est la taille idéale d'une équipe : https://blog.octo.com/2-pizza-team/

3. Scrum : https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-French.pdf

À l'échelle du projet, c'est la méthode agile SAFe[4] qui est utilisée pour synchroniser les équipes afin de garantir en fin de cycle ou incrément programme (PI)[5] une livraison d'un logiciel fonctionnel, testé et validé.

PC-SCOL fonctionne sur des PI de 12 semaines, soit 4 itérations SCRUM. Une démonstration publique de PÉGASE est faite en fin de PI auprès de la communauté de l'ESR.

À partir de la version 1.0.0 de PÉGASE, attendue début 2020, il est prévu de livrer à la communauté, hors mises à jour critiques (anomalie bloquante ou de sécurité), une nouvelle version de PÉGASE tous les 3 mois à l'issue de chaque PI.

3 PÉGASE

3.1 Les exigences & contraintes

PÉGASE sera un composant majeur du SI de l'enseignement et de la gestion de la vie étudiante. À ce titre, un certain nombre d'exigences et de contraintes sont à prendre en considération.

Tout d'abord, les utilisateurs de l'application seront divers (apprenants, enseignants, gestionnaires, etc.) et potentiellement très nombreux. À cela nous pouvons ajouter le fait qu'une telle application va être soumise à une saisonnalité et des pics de charge. Ils sont relativement identifiés (inscriptions, rentrée, résultats, etc.) et peuvent être gérés par l'exploitant qui héberge l'application. Il est toutefois envisagé de livrer PÉGASE sous une forme auto-élastique pour pallier à toutes surcharges non anticipées.

Il est souhaité que l'application PÉGASE soit accessible pour les utilisateurs au travers d'un navigateur Web et via des API Web REST pour les communications applicatives.

À l'image des applications que PÉGASE doit remplacer, la solution doit être pérenne. Elle devra pouvoir être exploitée pendant 20 ans à compter de l'achèvement de la première vague de déploiement en production. PC-SCOL prendra à sa charge la maintenance de l'application et du socle applicatif.

Compte tenu de l'importance des services que rendra PÉGASE à ses utilisateurs et de son mode d'utilisation, la disponibilité souhaitée est importante. L'objectif *zero downtime* en période d'inscription est demandé. Une disponibilité plus faible sera possible en période creuse pour permettre d'éventuelles opérations de maintenance.

L'application devra répondre aux standards d'ergonomie et de bonne expérience utilisateur en garantissant 90 % des transactions client/serveur en moins d'une seconde.

Son architecture doit faciliter la coopération avec des solutions périphériques existantes ou à venir et permettre la co-construction avec des partenaires non PC-SCOL.

Enfin, PÉGASE sera principalement proposé en mode hébergé (*SaaS*), mais possiblement diffusé pour une exploitation sur site (*on premise*).

3.2 Les méthodes & principes

Pour satisfaire ces différentes exigences et contraintes, PC-SCOL a dû adopter un certain nombre de recommandations en termes de méthode d'organisation, et d'architecture technique.

On l'a vu, PC-SCOL travaille en agilité au niveau de l'équipe, et chacune de ces équipes se voit confier le développement d'un module de PÉGASE. Ce module correspond à un macro-processus métier. Il est issu d'un travail de recueil des exigences fonctionnelles mené avec 30 établissements entre 2016 et 2017 pour cartographier tout le périmètre fonctionnel de PÉGASE.

4. SAFe Framework : https://www.scaledagileframework.com/

5. Incrément Programme : https://www.scaledagileframework.com/program-increment/

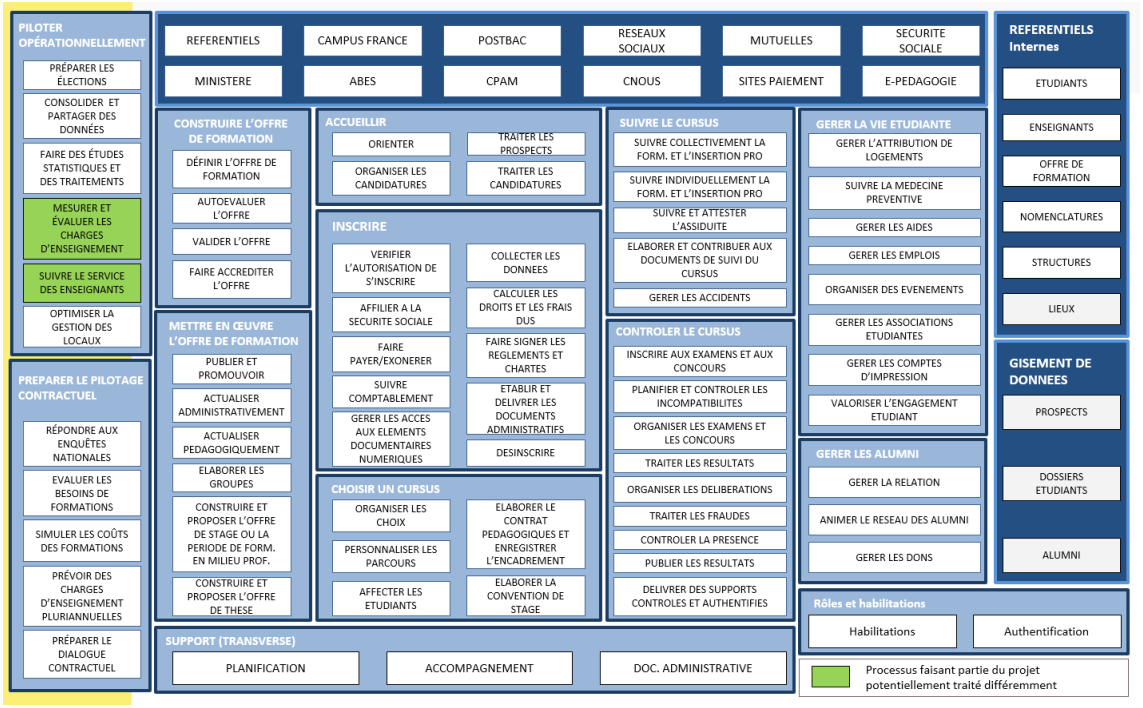


Figure 1 - Cartographie fonctionnelle de PÉGASE

L'analyse des macro-processus et la conception technique reposent sur un ensemble de bonnes pratiques aptes à satisfaire les exigences du projet.

Le manifeste Reactiv [2] recense ces bonnes pratiques afin de produire un système dit réactif. Celui-ci est composé de différents composants faiblement couplés pour répondre aux exigences de :

- disponibilité ;
- résilience ;
- élasticité.

Le découpage de l'application en de plus petits composants a plusieurs bénéfices. Tout d'abord, le développement de chacun est confié à une équipe Scrum qui pourra avancer en autonomie. Ensuite, ce découpage participe à augmenter la durée de vie du logiciel en facilitant sa maintenance sur le long terme. En réduisant le périmètre métier et la complexité de chacun des composants, l'entropie logicielle et la dette technique sont contenues.

Cependant, cela ne suffit pas, et les composants devront satisfaire un ensemble de recommandations techniques qui participeront également à la pérennité du projet. Dans ce but, les équipes PC-SCOL veillent à respecter les 12 points énumérés dans le document *12-Factor* [3].

Enfin, pour éviter de nous enfermer dans des solutions techniques spécifiques et limiter le coût de possession de PÉGASE nous utilisons des protocoles standards et des logiciels libres.

3.3 Architecture applicative

D'un point de vue architecture applicative, PÉGASE est modulaire et *API Firs*t.

Chaque module a pour objectif de répondre aux besoins métiers d'un macro-processus. Les APIs permettent non seulement la communication entre les modules en interne de PÉGASE mais aussi l'intégration de PÉGASE au sein du système d'information de l'établissement.

La pile applicative[6] d'un module de PÉGASE est composée :

- d'un (voire plusieurs) composant Web sous la forme d'une *Single Page Application* Angular, exécuté dans le navigateur de l'utilisateur (application *frontend*) ;
- d'un (voire plusieurs) composant serveur Java Spring Boot (application *backend*) ;
- d'un (voire plusieurs) composant de persistance de données (*storage*).

Les exigences fonctionnelles pilotent la description des APIs sous une forme standard (OpenAPI 3[7] pour les APIs REST). Ces APIs seront ensuite publiées pour être réutilisées par les composants qui devront les exposer ou les consommer.

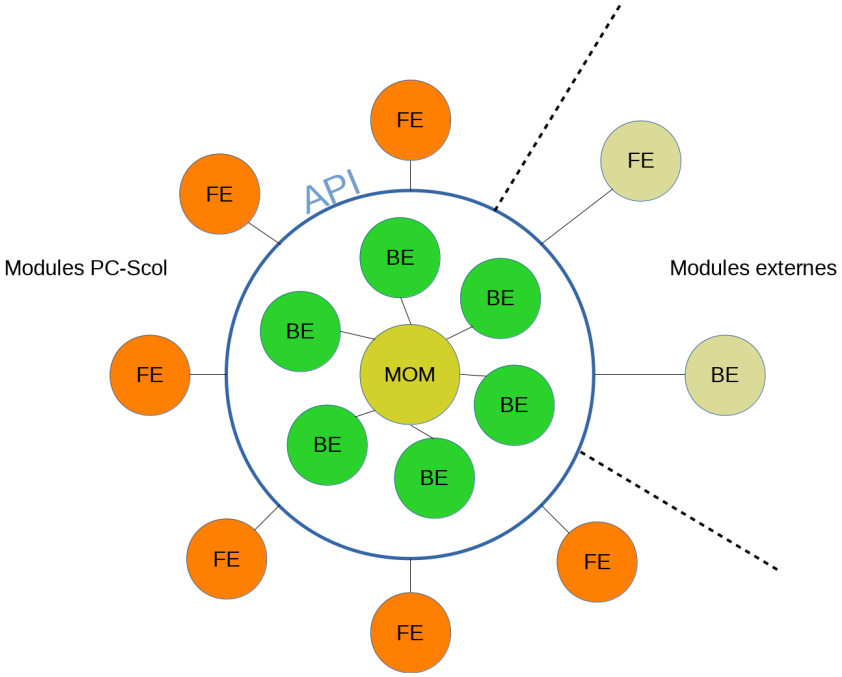


Figure 2 - Approche API first et MOM

En interne, PÉGASE utilise aussi des APIs asynchrones pour la communication entre les composants *backend*. Cette communication s'appuie sur des messages gérés par un *Message Oriented Middleware* (MOM). À ce jour, seules les API REST sont ouvertes sur l'extérieur.

PC-SCOL fait le choix d'une persistance polyglotte, c'est-à-dire que les modules de PÉGASE utiliseront un système de base de données le plus adapté à leurs exigences fonctionnelles. Les systèmes utilisés par le projet sont listés dans le tableau ci-après.

Système de base de données	Logiciel
Base de données relationnelle	PostgreSQL
Base de données de type graphe	Neo4J
Base de données de type document	MongoDB

6. La pile applicative et technologique de PC-SCOL est détaillée en annexe.

7. OpenAPI : https://openapis.org/

Les exigences de résilience et d'élasticité encouragent à concevoir l'application en limitant le plus possible les composants à état ou *stateful*. Ainsi, les composants *frontend* et *backend* sont strictement sans état ou *stateless*. La persistance des données est déléguée à des serveurs de base de données.

On notera également que l'application ne maintient pas non plus de session. On utilise OpenID Connect[8] et JWT[9], *stateless* par nature, qui participent à la disponibilité et à bonne montée en charge de l'application.

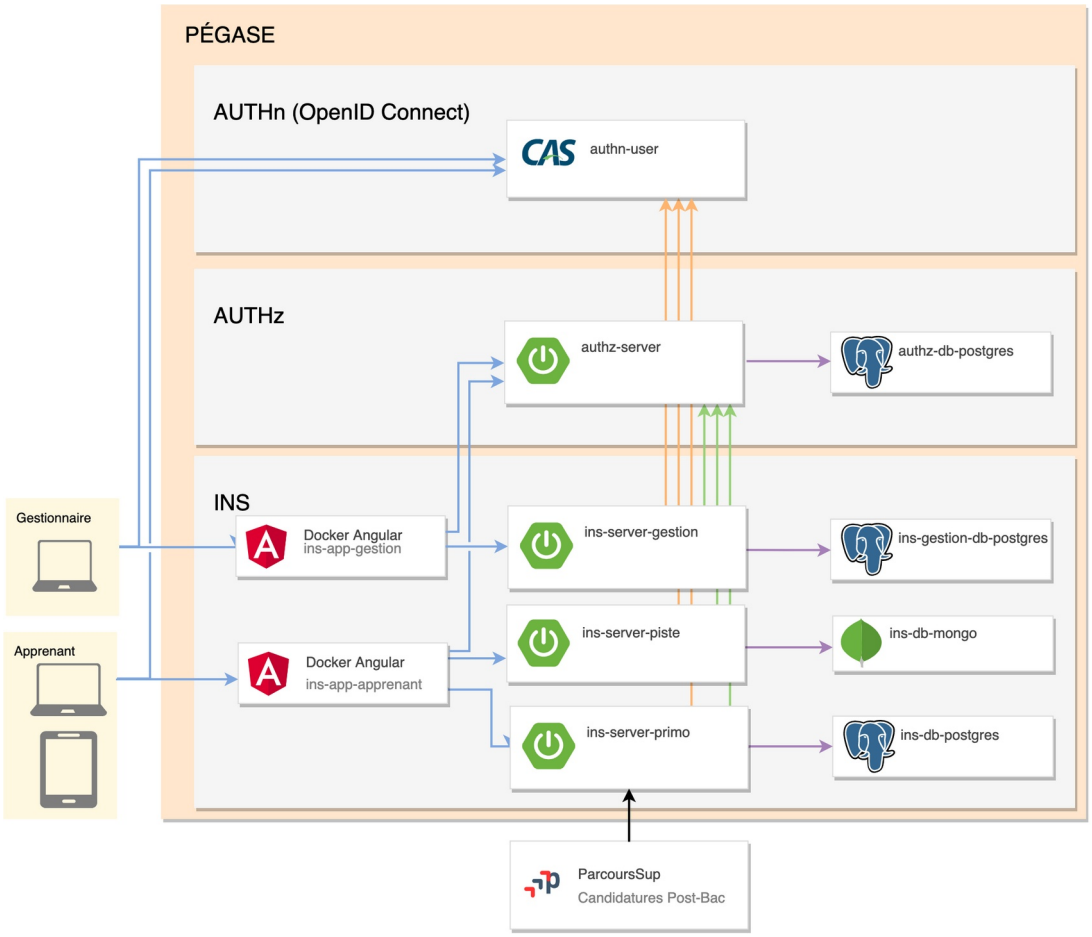


Figure 3 - Exemple du module INSCRIRE (Inscription administrative)

3.4 PÉGASE, une application qui se branche au SI

Nous avons vu que PÉGASE expose des APIs pour que des applications externes puissent interagir avec lui. Il s'interface également avec le SI afin de :

– déléguer l'authentification des utilisateurs à un système d'authentification en place, généralement de type SSO CAS[10] ;

8. OpenID Connect : https://openid.net/connect/

9. JSON Web Token : https://jwt.io/

10. Apereo CAS Server : https://apereo.github.io/cas/6.0.x/index.html

– se synchroniser avec le référentiel (SINAPS pour l'AMUE, GRHUM pour Cocktail ou un annuaire SupAnn par exemple) pour consommer des données de référence (personnes, structures, etc.) ou les compléter (apprenants).

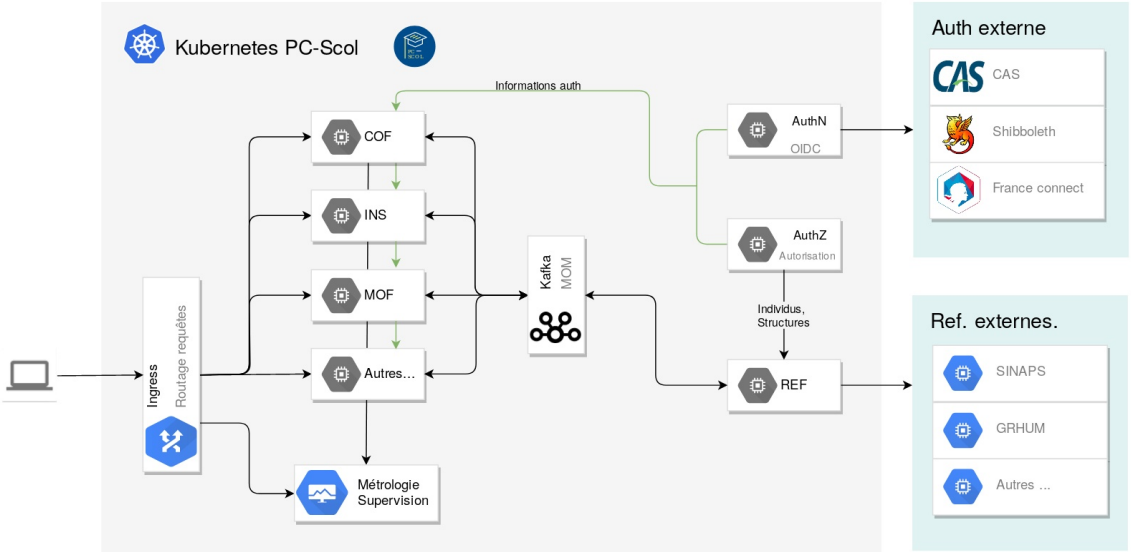


Figure 4 - Les modules de Pégase se branchant sur le SI

3.5 PÉGASE, une application cloud native

Compte tenu de l'architecture fonctionnelle et technique de PÉGASE (multiples composants, une majorité d'applications *stateless*), la conteneurisation est une technologie qui a été sélectionnée dès le début du projet. Elle apportera l'immutabilité du composant, favorisant une production plus rapide par intégration continue et permet un livrable qui sera l'unité atomique applicative de PÉGASE.

PC-SCOL fait actuellement le choix d'avoir une chaîne de construction d'images de conteneur Docker sans Docker mais avec *Bazel* et *Jib*. Le moteur actuellement utilisé est *Docker Engine CE*. À l'avenir nous pourrons nous tourner vers les standards de l'industrie : *OCI Image Specification*[11] pour le format d'image et *containerd*[12] pour le moteur.

PÉGASE, en v0.6.0 – une version de développement non diffusée – compte déjà 36 composants. On estime que la première version majeure, la v1.0.0, qui sera déployée et utilisée par 6 établissements pilotes courant 2020, comptera plus d'une cinquantaine de composants.

On peut regrouper ces composants en plusieurs catégories, comme le montre le tableau suivant.

11. Spécifications Open Container Initiative : https://github.com/opencontainers/image-spec/blob/master/spec.md
12. Containerd : https://containerd.io

Composants métiers	Forme applicative à héberger	Nombre
Frontend	Serveur Web de fichiers statiques	6
Backend	Serveur d'application Java	9
Composants d'infrastructure	Forme applicative à héberger	Nombre
Stockage	Serveur de base de données	8
Authentification	Serveur CAS	2
Intergiciels / MOM	Serveurs d'infrastructure Kafka	3
Composants d'infrastructure additionnels	Forme applicative à héberger	Nombre
Outils d'administration	Gestion de composants de stockage	4
Métrologie	Serveur de stockage et de consultation	2
Logs	Serveur d'agrégation, de stockage et de consultation	2

Il est inenvisageable de vouloir exploiter de façon traditionnelle une telle application, d'autant plus dans une perspective d'hébergement à l'échelle pour un service SaaS. Une plateforme d'orchestration est donc nécessaire. PC-SCOL a débuté le développement de PÉGASE en 2017, à cette époque une technologie gagne en popularité dans le domaine de l'orchestration de conteneur : Kubernetes[13].

4 Kubernetes

4.1 Un peu d'histoire

Google annonce travailler sur un projet nommé Kubernetes en 2014 [5]. À l'époque, Eric Brewer, vice-président Infrastructure de Google et auteur du théorème CAP [6], précise qu'il s'agit pour l'entreprise de rendre publique une version allégée de leur propre plateforme d'orchestration d'applications conteneurisées. Google n'est pas débutant dans le domaine, son intérêt pour la conteneurisation sous Linux date de 2006 avec notamment le travail de 2 de ses ingénieurs sur la fonctionnalité *Process Container*. Renommée *Control Group* ou *Cgroup* en 2007 [7], elle va permettre d'associer à un ou plusieurs processus un ensemble limité de ressources et de droits bien identifiés. *Cgroup*, et la fonctionnalité d'espace de nom (Linux namespace), apparus en 2002 dans le noyau sont les mécanismes fondamentaux de la conteneurisation sous Linux.

Cette isolation légère permise directement par le noyau Linux a pour principaux intérêts :

- un démarrage instantané : le conteneur utilise le noyau de l'hôte, il n'y a pas de VM ni de système d'exploitation à démarrer ;
- de permettre une densité plus forte : il n'y a pas de surcoût (*overhead*) en termes de ressources de calcul.

LXC pour Linux Container, sortie en 2008, est la première interface utilisable pour créer des conteneurs.

13. Kubernetes ou k8s : https://kubernetes.io/fr/

Google poursuivit de son côté en interne et rendit publique plus tard, en 2013, *Let Me Contain That For You*[14]. Il permettait à Google d'empaqueter ses propres applications pour les livrer et les exécuter sous forme de conteneurs à grande échelle sur les clusters de l'entreprise.

Un peu plus tôt dans l'année, une jeune société, dotCloud, publie en open source la première version du logiciel Docker. Il est développé en interne par la société alors présente dans le secteur de l'hébergement et du *Platform as a Service* (PaaS). L'intérêt pour la conteneurisation est évident pour ces hébergeurs : réussir à empaqueter une application et ses dépendances de manière immutable[15], les distribuer jusqu'à des clusters de machines qui en assureront leur exécution au meilleur coût dans un cadre défini et contrôlé est un objectif ultime. [8]

Docker n'invente ni le concept de la conteneurisation ni les mécanismes techniques, mais il arrive au bon moment et fournit une approche simple à utiliser qui fera sa popularité (construction d'une image à partir d'un fichier Dockerfile, gestion en couche de l'image et publication sur un dépôt central).

En interne, Google qui utilise massivement les conteneurs, a eu à développer des plateformes pour les déployer. On citera notamment Omega, présenté en 2013 [9]. L'expérience accumulée dans ce domaine et dans l'exploitation à très grande échelle aboutit à l'annonce publique de Kubernetes en 2014.

Dans le billet de blog qui annonce Kubernetes, Eric Brewer, écrit :

> *Everything at Google, from Search to Gmail, is packaged and run in a Linux container. Each week we launch more than 2 billion container instances across our global data centers, and the power of containers has enabled both more reliable services and higher, more-efficient scalability. Now we're taking another step toward making those capabilities available to developers everywhere. […] Based on our experience running Linux containers within Google, we know how important it is to be able to efficiently schedule containers at Internet scale. We use Omega within Google, but many developers have more modest needs. To that end, we're announcing Kubernetes, a lean yet powerful open-source container manager that deploys containers into a fleet of machines, provides health management and replication capabilities, and makes it easy for containers to connect to one another and the outside world. (For the curious, Kubernetes (koo-ber-nay'-tace) is Greek for "helmsman" of a ship.)*

En 2015, Google, avec d'autres industriels du secteur, crée la *Cloud Native Computer Foundation*[16] (CNCF), sous la tutelle de la *Linux Foundation*[17] et lui confie le projet Kubernetes dans sa première version de production, la v1.0[18].

Il s'agit d'un évènement fondateur ; Kubernetes qui jouit dès son annonce, de par son héritage et son architecture d'une certaine reconnaissance, est à présent soutenu et financé par une grande partie de l'industrie.

Depuis, celle-ci contribue largement à Kubernetes et son écosystème en publiant en open source et le plus souvent sous le patronage de la CNCF, des logiciels innovants et dédiés au développement ou à l'exploitation d'application cloud native.

4.2 PÉGASE et Kubernetes, des points communs

Kubernetes se définit comme une plateforme ouverte, extensible et portable pour l'exécution d'application et de services conteneurisés.

Concrètement Kubernetes se présente sous la forme d'un cluster de nœuds. Ses *masters* qui assurent le *control plane* et souvent hébergent la persistance de l'état de k8s (cluster etcd) et des *workers* qui sont des

14. Let Me Contain That For You, LMCTFY : https://github.com/google/lmctfy

15. Infrastructure immutable : on ne modifie par un système en cours d'exploitation. On déploie un nouveau système qui intègre les modifications, il remplacera l'ancien.

16. Cloud Native Computing Foundation : https://www.cncf.io/

17. Linux Foundation : https://www.linuxfoundation.org/

18. Actualité sur Techcrunch : https://techcrunch.com/2015/07/21/as-kubernetes-hits-1-0-google-donates-technology-to-newly-formed-cloud-native-computing-foundation-with-ibm-intel-twitter-and-others/

hôtes, généralement Docker, qui vont exécuter les applications suivant l'orchestration définie par le Control Plane.

L'utilisateur du cluster, peut soumettre via les API exposées par les nœuds masters (kube-apiserver), des manifestes, au format YAML, représentant l'état souhaité d'objets d'infrastructures que sait gérer k8s. En interne, les masters, via des contrôleurs (kube-controller-manager) vont, en boucle, faire en sorte de respecter ces manifestes, et via un orchestrateur (kube-scheduler) répartir la charge de travail parmi les nœuds workers.

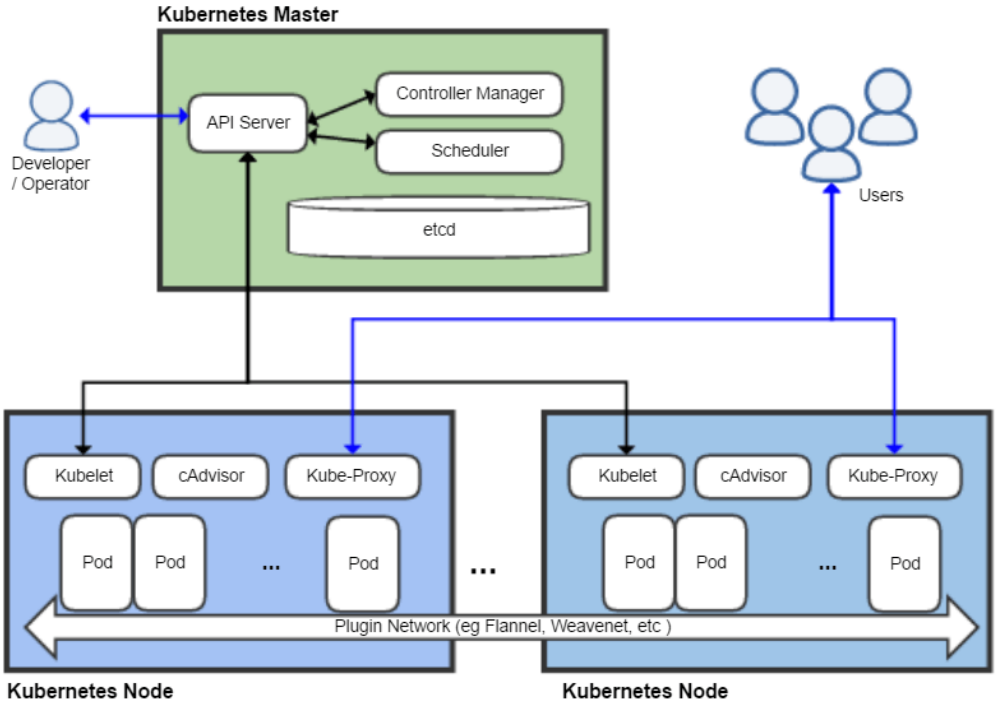


Figure 5 - Architecture de Kubernetes
Khtan66 [CC BY-SA 4.0]

Kubernetes intègre un ensemble de fonctionnalités permettant de satisfaire les exigences de PÉGASE.

– **Elasticité & Résilience** : Kubernetes est de base multi-nœuds avec une redondance des nœuds maîtres et du moteur de persistance des données (etcd). Ensuite, il propose via les mécanismes de *replicaset*, *statefulset* et de *service* de répondre aux besoins de montée en charge des applications qu'il exécute. Cette élasticité peut être définie statiquement par l'exploitant, voire traitée dynamiquement et automatiquement par des autoscalers (*Horizontal Pod Autoscalers*, *Vertical Pod Autoscaler*, et *Cluster Autoscaler*). On peut par exemple imaginer une règle qui met à l'échelle horizontalement le composant qui gère les inscriptions en fonction de métriques métiers (nombre d'étudiants connectés) et une autre qui ajoute des noeuds en fonction de métriques systèmes et de la charge du cluster. Ces systèmes sont avant tout prévus pour aligner le mieux possible les infrastructures sur la charge réelle pour limiter les coûts d'exploitation. Ils sont très intéressants pour automatiser la réponse à des pics de charge imprévus.

– **Ouverture et portabilité** : Kubernetes est un projet open source qui se structure pour garantir un cœur relativement simple et fonctionnel. L'utilisation d'interfaces standards et l'externalisation de codes additionnels tiers sont des objectifs importants des groupes de travail (Special Interest Group – SIG[19]) qui structurent le projet. Kubernetes se positionne comme une couche d'abstraction des infrastructures sous-jacentes (calcul, stockage, réseau, etc.) qui sont potentiellement portées par des constructeurs et technologies différents. Cette notion est importante car, elle permet de satisfaire une exigence importante du projet PC-SCOL, celle de la portabilité de PÉGASE et de sa compatibilité avec différents modèles d'exploitation (SaaS à grande échelle ou un déploiement sur site).

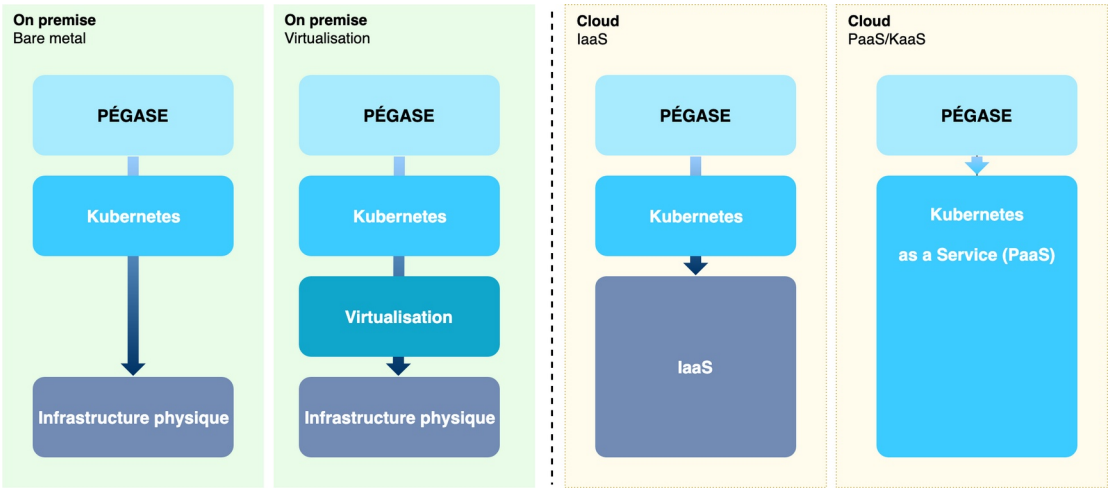


Figure 6 - Portabilité d'une application Kubernetes

– **Architecture & technologies** : PÉGASE et Kubernetes sont de même génération et ils partagent de nombreux points communs d'architecture (API First, immutabilité, *stateless*, etc).

– **Pérennité** : l'organisation du projet Kubernetes, de la CNCF et le dynamisme de l'industrie, font de Kubernetes la solution d'orchestration de conteneurs la plus populaire et la plus utilisée dans le monde. La solution est maintenant partout, des fournisseurs de clouds publics aux éditeurs. Les alternatives sont de moins en moins nombreuses et continuent d'abandonner du terrain[20].

4.3 Développer une application pour Kubernetes

Bien que Kubernetes réponde en théorie aux besoins de PC-SCOL pour l'exploitation de PÉGASE, la mise en œuvre n'est pas aisée et nécessite de se recentrer sur les bonnes pratiques de développement et de croiser le tout avec les fonctionnalités natives de k8s ou d'autres applications de son écosystème. En somme, il s'agit de développer une application pour Kubernetes.

En termes d'exploitabilité de PÉGASE, voici les objectifs que nous nous sommes fixés :

– une portabilité et compatibilité avec un hébergement à l'échelle et un hébergement léger sur site ;
– un déploiement et une mise à jour en un clic ;
– une gestion des versions sur tous les types de livrables ;
– des démarrages rapides des composants et de PÉGASE à l'échelle de la solution ;
– une automatisation des opérations de maintenance ;
– des procédures ultra-simplifiées en cas de désastre et de reprise d'activité.

19. Kubernetes SIG : https://github.com/kubernetes/community/blob/master/sig-list.md

20. Mesosphere, la société derrière Mesos/DCOS, la principale alternative à Kubernetes, change de nom et pivote vers k8s en 2019 : https://techcrunch.com/2019/08/05/mesosphere-changes-name-to-d2iq-shifts-focus-to-kubernetes-cloud-native/

Si on reprend le document *12-Factor*, on se rend compte qu'un certain nombre de points sont pertinents et implémentables dans le contexte d'intégration de notre application *cloud native*, PÉGASE, avec Kubernetes. Dans la suite de l'article nous reprenons ces points en les regroupant par thème pour décrire notre travail sur l'intégration PÉGASE pour k8s.

4.3.1 Base de code, Assemblez, publiez, exécutez & Parité dev/prod

Bien qu'il serait possible d'écrire des manifestes YAML statiques pour soumettre nos traitements à Kubernetes, nous avons préféré dès le début du projet utiliser l'outil Helm[21].

Helm est depuis devenu le gestionnaire de paquets quasi officiel pour Kubernetes.

On a vu la modularité de PÉGASE et sa décomposition en éléments applicatifs simples, nous allons maintenir cette stratégie en créant un paquet Helm, appelé chart, pour chacun des composants de PÉGASE.

Pour créer ce chart, nous écrivons du code (des fonctions *helpers*) et des modèles paramétrés et réutilisables (*templates*). Ceux-ci, au déploiement d'un chart Helm, sont compilés avec les valeurs des paramètres (*values*) pour générer des manifestes Kubernetes fonctionnels.

Pour suivre les évolutions de ces charts Helm, nous les stockons dans les mêmes dépôts Git que le code applicatif du composant.

Les développements des composants de PÉGASE s'inscrivent dans un processus intégration continue. Celui-ci exécute à chaque modification du code, un *pipeline* qui se décompose en plusieurs étapes :

1. construction du tag de version pour identifier la modification courante ;
2. la construction de l'application ;
3. les tests unitaires ;
4. les tests d'intégration ;
5. une analyse de qualité du code ;
6. une analyse statique de la sécurité du composant ;
7. la construction d'une image de conteneur et sa publication sur un dépôt Docker avec le tag de version ;
8. la construction du chart Helm du composant et sa publication sur un dépôt Helm avec le tag de version.

À l'issue de ce processus, l'application modifiée est déployée sur un environnement de développement, qui est une instance composite de PÉGASE. Elle rassemble des déploiements autonomes de chacun de ses composants.

Nous utilisons le *workflow* Helm *umbrella [10]*, qui permet de définir des paquets de paquets avec une gestion de dépendances. Aussi, le chart de PÉGASE, se présente sous la forme d'un super chart Helm dont ses composants apparaissent comme des dépendances. Chaque nouvelle modification de celles-ci provoque l'actualisation et la construction d'une nouvelle version de PÉGASE.

Également publiée sur un dépôt Helm, elle est immédiatement accessible et déployable sur les environnements des testeurs et des *products managers*.

21. Helm est créé en 2017 par la société Deis, rachetée depuis par Microsoft : https://helm.sh/

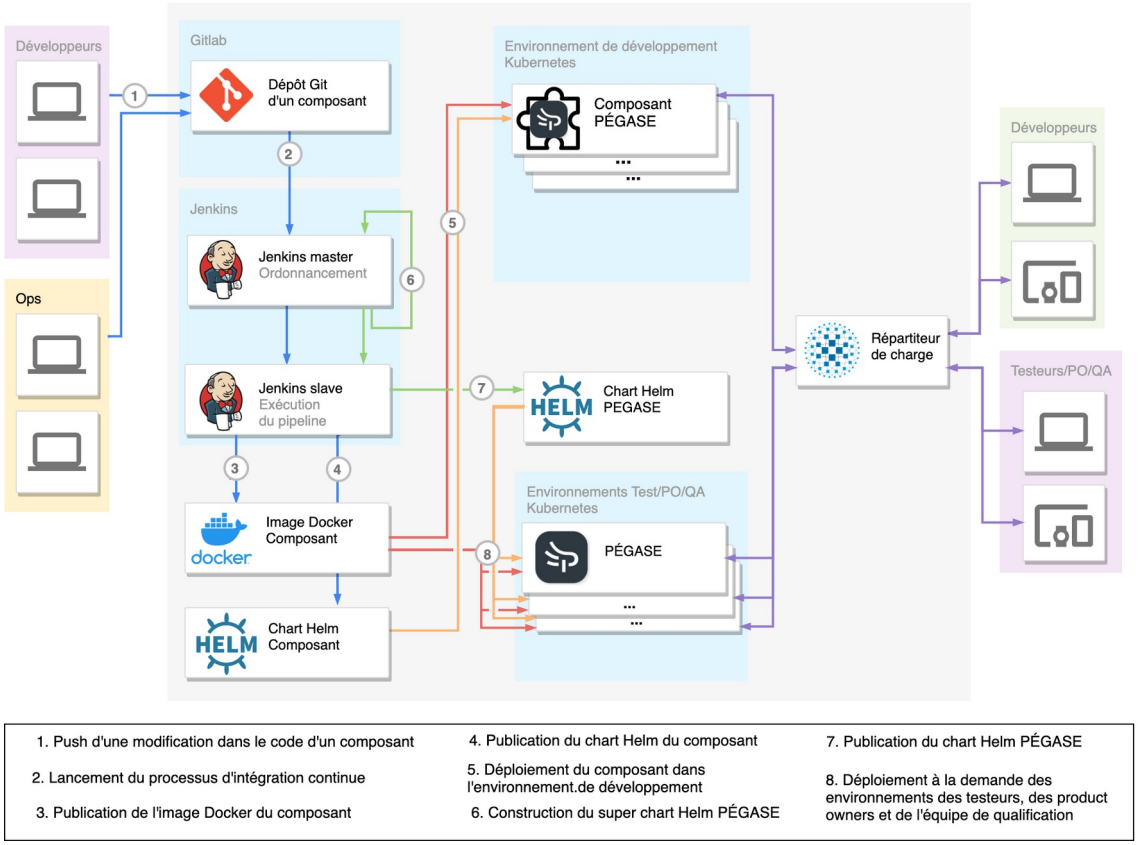


Figure 7 - Plateforme d'intégration continue de PC-SCOL

4.3.2 Dépendances & Portabilité

Kubernetes propose plusieurs types de ressources pour exécuter des applications. La plus élémentaire est le *pod*. On l'associe généralement à un conteneur.

Si un composant métier de PÉGASE peut se résumer à une application Angular servie par un serveur web simple ou une application Spring Boot, on ne réduit généralement pas son exécution ni à un conteneur seul, ni à un *pod* seul.

Il est parfois nécessaire de lui associer des conteneurs compagnons comme des *initContainers* ou des *sidecar containers*. Ces méthodes permettent d'enrichir fonctionnellement un *pod* en exécutant plusieurs conteneurs de manière séquentielle ou parallèle. Cette approche est utile pour maintenir le concept de séparation des responsabilités en dédiant un conteneur à une tâche précise avec des autorisations précises en conservant l'unité du pod. Nous utilisons ces concepts principalement sur les composants *backend*.

Outre la ressource de type *pod* qui représente l'exécution de l'application, le chart Helm d'un composant PÉGASE, associe généralement (dans le cas d'un composant *stateless*) :

- un *deployment* pour pouvoir gérer sa mise à l'échelle et ses migrations ;
- une ou plusieurs *configmap*, représentant des objets de configuration, clef/valeur ou fichiers ;
- un ou plusieurs *secrets* pour le stockage des mots de passe, des clefs, des certificats ;
- un *service* pour exposer notre composant.

C'est l'utilisation conjointe de ces différentes ressources Kubernetes qui va rendre le composant fonctionnel et utilisable. Le livrable sous forme de chart Helm est donc autoportant et se suffit à lui-même. Nous pouvons le distribuer et le déployer de manière autonome ou l'intégrer dans une application de plus haut niveau comme PÉGASE.

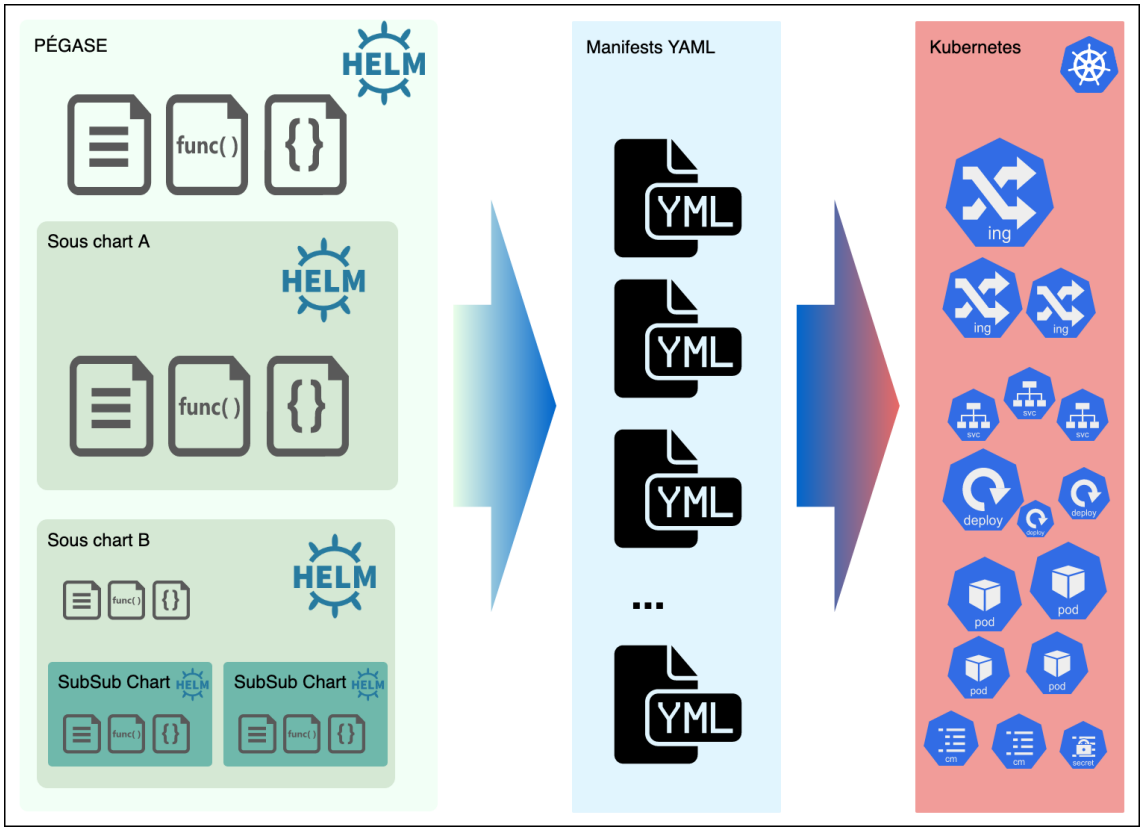


Figure 8 - Du chart Helm à l'exécution de l'application

4.3.3 Application, configuration, données, vers une application jetable

Nous sommes convaincus que la clef d'une exploitation facilitée à l'échelle repose sur une automatisation basée sur les concepts d'immutabilité et d'infrastructure définie logiciellement.

Afin de satisfaire cela pour une application complexe comme PÉGASE, il faut s'efforcer d'entretenir une séparation forte entre ce qui relève de l'application, de la configuration et des données.

L'application est portée par les images de conteneur construites au fil des développements des composants.

La configuration est assurée par un profil, une suite de valeurs de configuration que l'on pourra appliquer au déploiement de PÉGASE via Helm. Si un chart Helm peut intégrer des valeurs par défaut, celles-ci sont toutes sur-chargeables au déploiement.

Les données de PÉGASE, sont traitées via les mécanismes de *PersistentVolume* et de *StorageClass* de Kubernetes. Utilisées dans l'exécution d'application *stateful* comme les bases de données de PÉGASE, ces ressources permettent de persister des volumes de stockage sur le long terme et de les rendre accessibles à tous les nœuds du cluster. Elles favorisent la résilience en cas de perte de conteneur, de nœud, voire de *datacenter* selon l'implantation des infrastructures.

De même pour les données de sauvegarde ou d'archivage, nous favorisons un stockage objet compatible S3 distinct de l'infrastructure d'exécution. Accessible au travers du protocole HTTP et adapté à des transferts volumineux, il facilite les possibilités de reprise d'activité ou de migration vers ou depuis une infrastructure cloud.

En se concentrant sur la préservation des données :

- les images de conteneur et les charts Helm, versionnées et immutables ;
- les configurations, versionnées ;
- les données métiers, historisées via des mécanismes d'archivage temps réel et de sauvegardes.

l'application, au sens du processus en exécution, est jetable. On peut la recréer à l'identique à tout moment, et depuis n'importe quel état et rendre à nouveau le service aux utilisateurs.

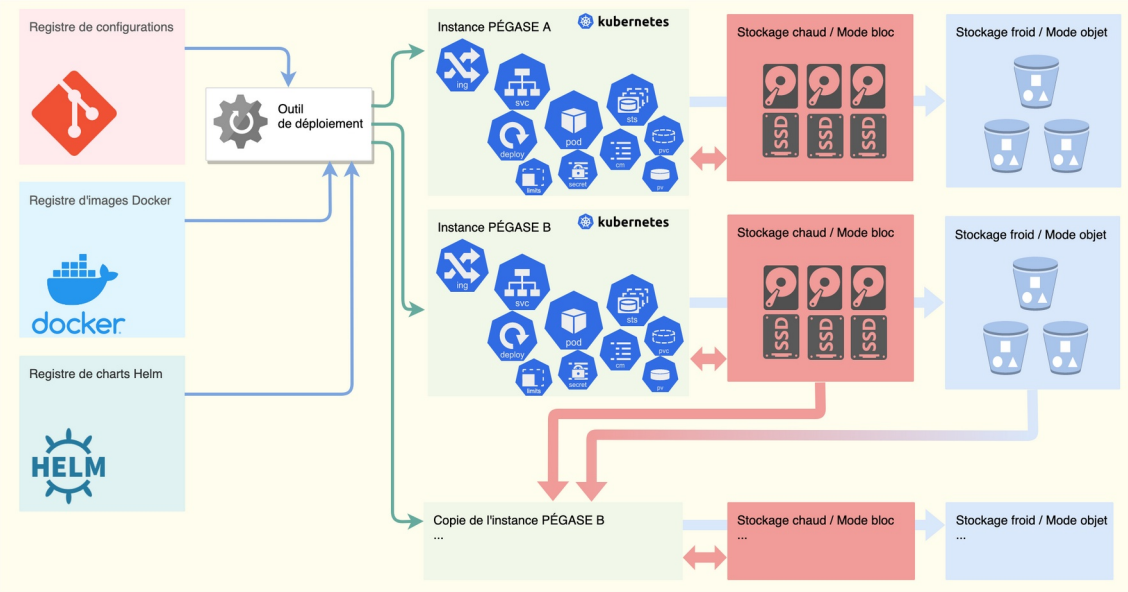


Figure 9 - Application, configuration, données

4.3.4 Processus d'administration & automatisation

Un objectif important encore, est l'automatisation des tâches généralement dévolues aux exploitants :

- sauvegarde ;
- restauration ;
- mise à l'échelle ;
- redémarrage et auto-réparation.

Dans le travail d'intégration Kubernetes, un effort particulier est fait pour automatiser le plus possible ces actions. Tout d'abord pour la sauvegarde, nous capitalisons sur les fonctionnalités de sauvegarde de chacun des composants de stockage (base de données), soit en passant par des mécanismes d'archivage en temps réel des journaux de transaction, dans le cas de PostgreSQL, soit en implémentant des traitements de sauvegarde qui vont utiliser les mécanismes de *dump* ou de *snapshot*.

Pour la restauration nous avons écrit un traitement, exécuté en option, au déploiement de PÉGASE, qui récupère les données de sauvegarde et les injecte dans les composants de stockage en cours d'initialisation.

Ces traitements prennent la forme de petites applications qui sont développées, versionnées et empaquetées avec PÉGASE. Cela permet de faire évoluer ces outils et de les adapter aux évolutions des autres composants de l'application.

Outre ces développements, Kubernetes intègre des fonctionnalités qui vont nous aider à simplifier, voire à supprimer, les tâches d'administration.

La mise à l'échelle de PÉGASE est simple pour les composants *stateless*. Ceux-ci, déployés sous forme de *deployments* sont facilement configurables pour être exécutés en plusieurs exemplaires de *pods* qui se répartiront la charge. Seulement la configuration du nombre de réplicats est un élément qui est actuellement statique et relatif à une configuration. Il ne s'adapte pas au contexte. Il est prévu que PÉGASE puisse tirer parti des fonctionnalités natives de Kubernetes comme les *Horizontal Pod Autoscalers* (HPA) qui vont automatiser la mise à l'échelle des composants *stateless* par ajout ou retrait de réplicats. Ces HPA sont entièrement configurables, et si à minima ils sont indexés sur des métriques systèmes, ils pourront quand les composants PÉGASE le permettront, prendre en charge des métriques métiers, comme le nombre d'étudiants connectés par exemple.

Pour les composants *stateful* comme les bases de données, la mise à l'échelle et l'automatisation d'opérations de maintenance sont compliquées et souvent contraires au principe d'immutabilité et d'idempotence. Des solutions commencent cependant à apparaître, les ***operators***[22]. Dans l'objectif de livrer une solution PÉGASE intégrée, nous avons commencé à utiliser ces applications qui viennent enrichir Kubernetes en apportant de nouvelles ressources et de nouveaux contrôleurs qui pourront les gérer. Les *operators* s'appuient sur les fonctionnalités natives de k8s que sont les *Custom Resources Definition* (CRD) et les *Admission Control Webhooks* pour enrichir fonctionnellement k8s en permettant l'exploitation déclarative et automatisée d'applications purement *stateful*.

En v0.6.0, PÉGASE utilise 2 *operators* :

- KubeDB pour la gestion des serveurs de base de données PostgreSQL ;
- Strimzi pour la gestion des serveurs du MOM Kafka.

Le service rendu dépend des fonctionnalités supportées par chaque operator, mais il s'apparente à celui des Database as a Services souvent proposées par les grands opérateurs de cloud public. Un contrat simple, dans une pure forme déclarative, le manifeste YAML, permet la mise à disposition quasi immédiate d'un service *stateful* complexe, par exemple : un cluster PostgreSQL multi nœuds élastique avec archivage et externalisation temps réel des logs de transactions et *dump* quotidien.

Nous n'allons pas détailler ici ces 2 *operators*, mais ils permettent à PC-SCOL d'intégrer dans PÉGASE les actions de sauvegarde, d'archivage, de mise à l'échelle, etc.

4.4 Plan de route

PÉGASE est attendu pour une utilisation en production dès 2020. Il nous reste de nombreux chantiers pour pleinement satisfaire nos objectifs.

Pour faciliter l'exploitation de la solution et améliorer la disponibilité de PÉGASE nous devons travailler sur une meilleure supervision des composants et de son intégration avec les mécanismes d'auto-réparation de Kubernetes (sondes et *autoscaler*).

Il est également nécessaire de préparer l'hébergement de PÉGASE à l'échelle. PC-SCOL devra exploiter une vingtaine d'instances du logiciel pour différents établissements pilotes. Pour garantir leur sécurité et leur isolation, nous réfléchissons à l'approche qui consisterait à dédier un cluster Kubernetes à chaque instance de PÉGASE. Cette approche radicale mais simple nous permettrait d'éviter les mécanismes de *multi-tenancy* de k8s parfois jugés faibles et la mise en œuvre de solutions complexes comme les *service meshes*. Ces clusters, construits sur une couche d'infrastructure automatisable, idéalement de l'*IaaS*, deviendraient une commodité. Faire et refaire la couche k8s et PÉGASE, ne prendrait que quelques minutes.

22. La société CoreOS, tout juste rachetée par Red Hat publie ses travaux sur son Operator Framework début 2018. Red Hat a depuis été rachetée par IBM en 2019.

5 Kubernetes de Pégase au SI de l'établissement

5.1 Retour d'expérience

La similitude qui existe entre un PÉGASE où sont juxtaposés des modules qui communiquent pour rendre un service et le Système d'Information (SI) d'un établissement où sont juxtaposés des applications qui communiquent pour rendre un service nous amène à nous interroger sur l'intérêt d'utiliser plus largement Kubernetes.

De même, PÉGASE embarque de nombreuses fonctionnalités qui ne relèvent pas strictement du métier de la scolarité comme l'authentification, l'autorisation, la gestion des logs, la supervision, l'audit, etc. Il y aurait donc un intérêt à réutiliser des services transversaux déjà en place (notés middleware dans le schéma ci-dessous) afin de mutualiser l'effort de mise en place. Mutualisation qui, de plus, permettrait d'avoir une gestion plus homogène des différents services du SI.

Nous avons touché du doigt la portabilité promise par Kubernetes et pensons qu'elle permet en effet de laisser la liberté de choisir le niveau d'hybridation que l'on souhaite en utilisant des applications en mode SaaS ou déployées sur une infrastructure locale.

Pour finir, PC-SCOL met en œuvre ses outils de construction ainsi que différents déploiements de PÉGASE sur une offre *IaaS* de l'université de Strasbourg. Nous sommes convaincus que ce modèle où on sépare clairement les problématiques d'infrastructure de la mise en œuvre du service est très intéressant.

5.2 Généralisation de l'usage de Kubernetes

En guise de conclusion, et en nous basant sur notre retour d'expérience, nous vous soumettons un modèle possible de généralisation de l'usage de Kubernetes à échelle d'un SI d'établissement.

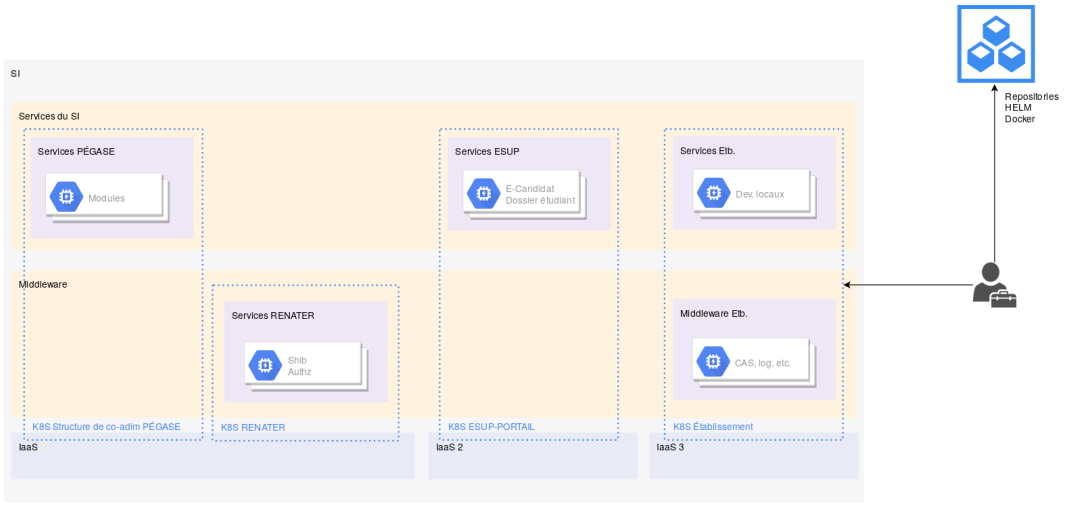


Figure 10 - Schéma possible d'utilisation généralisée de Kubernetes dans un SI d'établissement

Avertissement : Dans ce schéma très hypothétique, nous nous permettons, à titre d'exemple, d'imaginer que ESUP-PORTAIL et RENATER offrent des services en mode *SaaS* via une infrastructure Kubernetes. De même ici, RENATER et la structure de co-administration de l'offre *SaaS* de PÉGASE partagent la même infrastructure *IaaS*. À ce jour, rien de tel n'existe et nous ne présageons en rien de ce que décideront RENATER, ESUP-PORTAIL ou les établissements. La seule certitude, c'est qu'il existera bien une structure de co-administration qui offrira PÉGASE en mode *SaaS* basé sur Kubernetes.

Néanmoins dans ce schéma il y a plusieurs choses structurantes :

- Comme nous vous l'avons décrit dans cet article, nous pensons que Kubernetes est le système d'exploitation des applications et imaginons la généralisation de son usage. Ceci aussi bien pour qu'une structure puisse offrir des services en mode *SaaS* pour un ensemble d'établissements que pour un usage en interne dans l'établissement ;
- La liberté est laissée à l'établissement de rendre le service à ses utilisateurs soit en faisant appel à un prestataire qui offre un mode *SaaS*, soit en hébergeant *on premise* la fonctionnalité. Cette liberté est facilitée par les mécanismes de conteneurs et la gestion des dépendances entre ces conteneurs (Helm) décrits dans cet article ;
- Même en interne, nous imaginons une approche où les infrastructures matérielles sont offertes en mode *IaaS*. Pour une structure qui héberge une offre SaaS cela lui permet de ne pas être trop fortement couplé à un prestataire. Pour un établissement ayant les capacités d'offrir une *IaaS*, la mise en place d'une offre de service claire et d'un outillage en self service, facilite les interactions entre les pôles infrastructures et services d'une même DSI. À l'inverse, un établissement de plus petite taille peut trouver intérêt à s'appuyer sur l'offre *IaaS* d'un pôle universitaire local ;
- Nombre de services transversaux sont isolés dans une couche *Middleware*. L'objectif étant d'éviter que chaque développeur de service ne dépense de l'énergie sur des fonctionnalités non métiers.

Finalement, nous pensons qu'il faut que les développements nationaux comme locaux soient *cloud native* afin de permettre la conteneurisation et l'orchestration via un système d'exploitation applicatif comme Kubernetes et qu'il y a un travail important à faire pour définir ce que seraient les services middleware comme, par exemple, la gestion des autorisations ou la supervision. Ceci afin d'arriver à un niveau d'homogénéité intéressant, à la fois dans l'usage des services par les utilisateurs comme pour l'administration technique de ces services.

Enfin, la maîtrise au sein de l'ESR des technologies liées aux *IaaS* et *SaaS* via Kubernetes permet l'hébergement des données au sein de l'ESR tout en laissant la liberté aux établissements de choisir comment opérer le plus efficacement possible les services à offrir à leurs utilisateurs.

Annexes

PÉGASE	2019	Évolutions possibles
Frontend	Angular	
Design	Material Design	
Backend	Spring (Boot, Data)	
MOM	Apache Kafka	
Base de données relationnelles	PostgreSQL	
Base de données document	MongoDB	
Base de données graph	Neo4J	
Authentification	CAS Server	
Moteur de conteneur	Docker	Containerd
Orchestrateur de conteneur	Kubernetes	
Packaging Kubernetes	Helm	
Distribution de base de conteneur	Distroless (paquets Debian)	
Constructeur d'image de conteneur	Bazel (front) / Jib (back)	
Serveur web statique pour front	Nginx	
Operator PostgreSQL	KubeDB	
Operator Kafka	Strimzi	

PC-SCOL - Infrastructures	2019	Évolutions prévues
Infrastructure as a Service	OpenNebula	OpenStack
Code d'infrastructure	Terraform	
Stockage bloc	Ceph	
Stockage objet	Ceph Object storage	
Forge logicielle	Gitlab	
Orchestrateur CI/CD	Jenkins	
Registre d'artefacts	Nexus Repository OSS	
Registre d'images Docker	Nexus Repository OSS	Harbor
Registre de charts Helm	Chart Museum	Harbor
Kubernetes	Rancher Kubernetes Engine (RKE)	
OS hôtes	RancherOS	
Composant réseau	Canal (Calico + Flannel)	
Ingress Controller	Ingress-nginx	
Operator pour certificats x509	Cert-manager	
Équilibreur de charge	HAProxy	
Métriques	Prometheus	
Supervision	Prometheus / Grafana	
Récupération des logs	Fluentbit	
Agrégation et transformation des logs	Fluentd	
Stockage des logs	ElasticSearch	
Visualisation des logs	Kibana	
Ordonnancement et self service		Rundeck

Bibliographie

[1] Henrik Kniberg & Anders Ivarsson. Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds, 2012. https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf

[2] Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson. Reactive Manifesto, 2014. https://www.reactivemanifesto.org/

[3] Adam Wiggins. The twelve-factor app, 2011. https://12factor.net/

[4] Scott Leberknight. Polyglot Persistence, 2008. http://www.sleberknight.com/blog/sleberkn/entry/polyglot_persistence

[5] Google. An update on container support on Google Cloud Platform, 2014. https://cloudplatform.googleblog.com/2014/06/an-update-on-container-support-on-google-cloud-platform.html

[6] Eric Brewer. Towards Robust Distributed Systems, 2000. https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[7] Rohit Seth.2006. https://lwn.net/Articles/199643/

[8] Jérôme Petazzoni. Dérisquer son infrastructure avec les conteneurs, 2018. http://jpetazzo.github.io/2018/08/01/containers-cloud-immutable-infrastructure-orchestration/

[9] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, John Wilkes. Omega: flexible, scalable schedulers for large compute clusters, 2013. https://research.google.com/pubs/archive/41684.pdf

[10] Documentation de Helm. Complex charts with many dependencies. https://github.com/helm/helm/blob/master/docs/charts_tips_and_tricks.md#complex-charts-with-many-dependencies