L'unistra se dé-PaaS

Guillaume Oberlé, Alain Zamboni

L’Unistra se dé-PaaS

Guillaume Oberlé
Direction du Numérique
Université de Strasbourg
14 rue René Descartes
67 081 Strasbourg

Alain Zamboni
Direction du Numérique
Université de Strasbourg
14 rue René Descartes
67 081 Strasbourg

Résumé

Au sein de la Direction du Numérique (DNum), diverses initiatives ont été lancées pour maîtriser les technologies de conteneurs (Docker, Podman, Kubernetes) et rationaliser leur utilisation. Ces efforts visent à offrir des services PaaS robustes et évolutifs, destinés non seulement à l’Unistra, mais aussi avec l’ambition de proposer une offre de service à la communauté de l’enseignement supérieur et de la recherche française.

Le projet PaaS-Partout a ainsi démarré en mars 2022. Il s’attache à sélectionner et déployer les meilleurs outils pour une approche GitOps/DevOps commune. Les objectifs principaux sont d’étudier et choisir les outils adaptés, de mettre en place une infrastructure PaaS, de formaliser un processus de déploiement standardisé, et de définir les contours d’une offre de service.

L’étude des produits s’est officiellement achevée en mai 2023 sur le choix de plusieurs outils : OpenShift, Quay, Gitlab-CI, ArgoCD, HashiCorp Vault et Helm. Ces outils sont aujourd’hui déployés, et les premiers déploiements d’applications pilotes auront lieu en automne 2024.

Lors de notre présentation, nous partagerons notre méthodologie, les résultats des études, les choix technologiques justifiés et les réalisations techniques. Un retour d’expérience sur les outils déployés et les nouveaux processus d’adoption par les équipes sera également proposé.

Mots-clefs

PaaS, Kubernetes, OpenShift, CI, CD, intégration continue, déploiement continu, projet, Vault, ArgoCD, Gitlab-CI

1 Introduction

Au cours des dernières années, les pratiques de déploiement d'applications ont connu une évolution significative vers l'utilisation de conteneurs, orchestrés par des systèmes dédiés. Au sein des différentes équipes de la Direction du Numérique (DNum) de l'Université de Strasbourg (Unistra), plusieurs initiatives ont émergé lors de déploiements spécifiques afin de mieux appréhender ces technologies. Ces réalisations ont été effectuées avec divers outils : conteneurs simples (*Docker*, *Podman*), *docker-compose*, *Docker Swarm*, *Kubernetes*. Si ces initiatives ont permis à la DNum d'aborder ces nouvelles méthodes de déploiement d'applications, une démarche de généralisation et de rationalisation a été jugée nécessaire pour que ces technologies soient un réel bénéfice pour l'établissement.

De plus, déjà fournisseur de services IaaS, la DNum s'est engagée dans une démarche visant à héberger des applications en SaaS (*Immersup*, *Campulse*) pour des partenaires externes. Pour atteindre cet objectif et garantir une gestion opérationnelle soutenable, l'orchestration de conteneurs et le déploiement continu sont considérés comme des éléments clés du succès.

Enfin, dans le but de mutualiser les ressources entre établissements, la DNum s'efforce de construire ses services de sorte qu'ils puissent à terme être proposés à la communauté Enseignement-Recherche française, afin de fournir un environnement robuste et évolutif pour le déploiement et la gestion d'applications. Ainsi, un premier partenariat a déjà été conclu avec l'Université de Caen pour l'hébergement de l'application de gestion des mobilités étudiantes, *Smile*.

Pour ces raisons, le projet *PaaS-Partout* a été lancé en mars 2022. Ce projet fédérateur a pour objectif de sélectionner et déployer des outils permettant à l'Unistra de mettre en place une démarche commune *GitOps*/*DevOps* au sein de la DNum, et d'enrichir notre catalogue de services d'une offre PaaS à destination de la communauté de l'Enseignement Supérieur et de la Recherche (ESR).

Les principaux objectifs du projet sont les suivants :

- étudier, tester et sélectionner les outils les plus adaptés pour chaque fonction : distribution *Kubernetes*, registre d'images, intégration continue, déploiement continu, gestion des secrets ;
- déployer une infrastructure PaaS basée sur les outils sélectionnés ;
- définir et réaliser un processus standard de déploiement d'applications sur cette plateforme ;
- déployer des applications pilotes internes et externes ;
- formaliser et proposer une offre de service PaaS.

Le projet est, à l'heure où sont écrites ces lignes, toujours en cours, dans la phase de déploiement d'applications pilotes. Le présent article a pour but de vous présenter les travaux déjà réalisés. Nous présenterons d'abord la phase d'étude qui a abouti à la sélection de plusieurs éléments logiciels. Nous aborderons ensuite le déploiement de cette infrastructure PaaS à l'Unistra. Nous poursuivrons avec un premier retour sur le déploiement d'applications pilotes, bien que cette étape ne soit pas terminée. Nous conclurons sur nos premiers retours d'expérience sur nos réalisations.

2 Le projet

Au début du projet, nous avons tenté de schématiser notre cible d'une chaîne de déploiement continu.

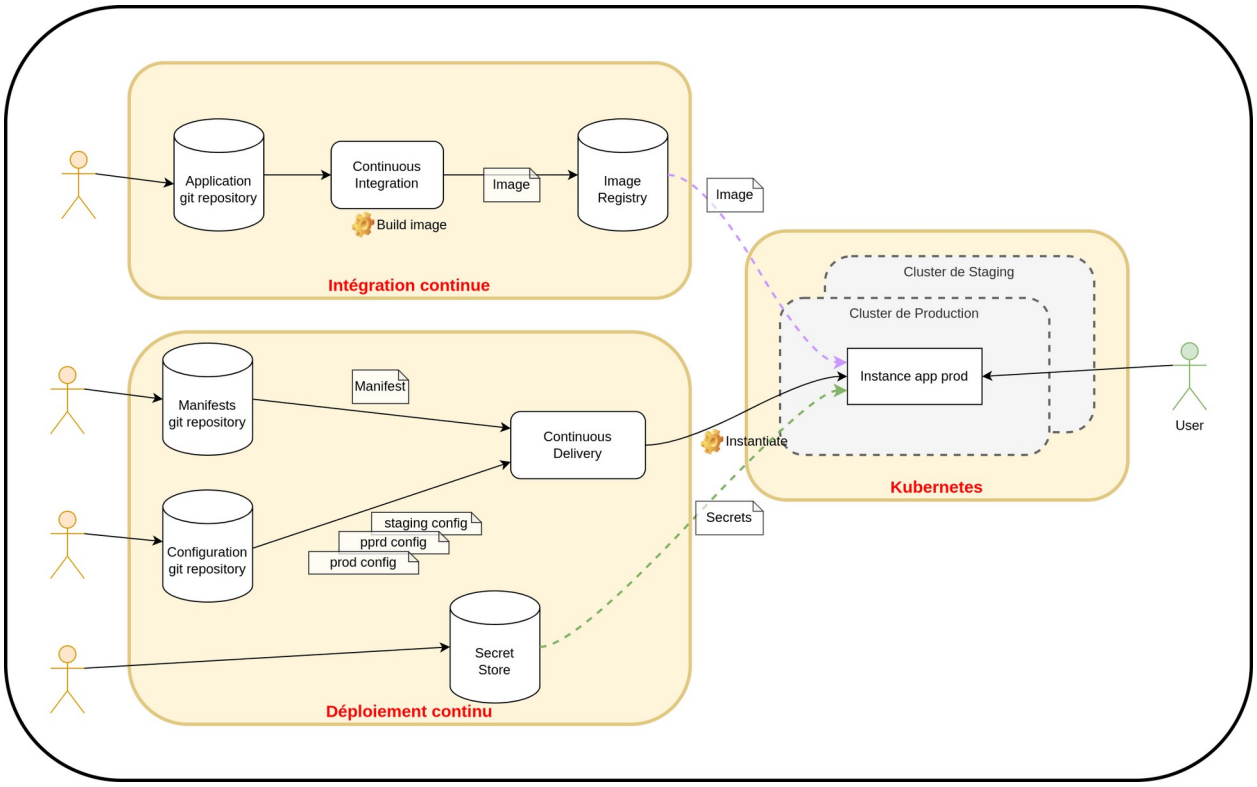

Figure 1: Cible de la chaîne CI/CD

Pour faciliter la discussion et la répartition des tâches, nous avons créé trois sous-groupes thématiques.

Le premier, sobrement nommé "*Kubernetes*", concerne les solutions d'orchestration de conteneurs et leur hébergement. Le groupe "Intégration continue" a trait à la génération d'image de conteneurs à partir du code source. Enfin le "déploiement continu" regroupe le packaging d'applications, les outils permettant leur instanciation automatique sur les clusters *Kubernetes* et aussi la gestion des secrets nécessaires à leur bon fonctionnement.

Étant donné le périmètre et les enjeux du projet, l'équipe est conséquente et transverse à l'organisation de la DNum : administrateurs systèmes, administrateurs d'applications et développeurs issus d'équipes différentes interviennent sur les diverses étapes. Au total, le nombre d'acteurs sur le projet atteint la trentaine de personnes, avec bien évidemment des degrés d'implications différents.

Le schéma suivant présente synthétiquement le planning des grandes phases du projet.

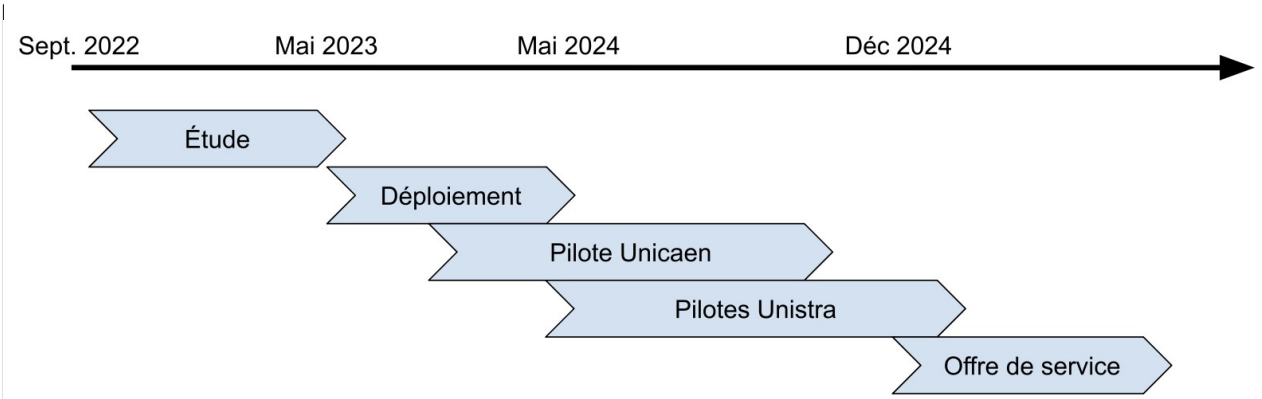

Figure 2: Macro-planning du projet PaaS-partout

On peut s'étonner de la durée des différentes phases, compte-tenu du nombre conséquent de personnes travaillant sur le projet. Ceci s'explique par le fait que les contributeurs projet ne sont pas à plein temps sur le sujet. Le temps réel passé sur le projet est aujourd'hui estimé à 200 jour-homme.

La fin du projet était initialement prévue pour décembre 2024. Il a cependant subi un retard considérable en raison de la charge d'exploitation des participants. Nous avons donc ré-estimé une nouvelle date de fin pour décembre 2025.

3 Étude et choix des outils

La première grande phase du projet fut de sélectionner les différentes applications qui constitueraient notre plateforme PaaS. L'étude a eu lieu de septembre 2022 à mai 2023. Les tests ont été effectués en interne par les équipes de l'Unistra, sans appel à des services de prestations des éditeurs. Nous souhaitions en effet pouvoir être aussi autonomes que possible sur les produits retenus. La possibilité de les prendre en main sans abus de prestation externe était donc un des critères d'évaluation de notre étude.

Les choix applicatifs de l'étude concernaient cinq fonctions différentes :

- la distribution *Kubernetes* ;
- la registry d'images de conteneurs ;
- l'outil d'intégration continue ;
- l'outil de déploiement continu ;
- l'outil de gestion des secrets.

Le tableau suivant présente les choix effectués pour chaque catégorie, les raisons principales de ce choix et les autres challengers respectifs évalués pour chaque catégorie fonctionnelle.

Tableau 1: Outils sélectionnés

Fonction	*Choix*	*Testés*	*Raisons principales*
Distribution *Kubernetes*	**Openshift Platform Plus (OPP)**	– Rancher – kOps – Ubuntu Charmed	– gestion multi-cluster (ACM) – intégration avec *OpenStack* – fonctionnalités pré-packagées (télémétrie, authentification, alerting)
Registry d'images	**Quay**	– Harbor – Nexus Repository	– fonctionnellement complet – support intégré au bundle OPP
Intégration continue	**Gitlab-CI**	– Tekton – Jenkins	– déjà utilisé – donne satisfaction – rapport bénéfices/coût de migration vers un autre outil insuffisant
Déploiement continu	**ArgoCD**	– FluxCD – Fleet – Spinnaker – Ketpn	– fonctionnellement complet – dashboard graphique – modèle centralisé – support intégré au bundle OPP
Gestion des secrets	**Hashicorp Vault**	– Sealed Secrets – CyberArk Conjur	– centralisation des secrets – polyvalence fonctionnelle

4 Déploiement et administration de la plateforme PaaS

Ce chapitre évoquera les différents éléments concernant le déploiement et l'administration de la plateforme PaaS. Nous évoquerons tout d'abord l'architecture des différents clusters *Kubernetes* et le mode de déploiement au sein d'*OpenStack*. Nous poursuivrons par la méthodologie de déploiement des différents clusters. Nous aborderons ensuite les choix effectués pour le paramétrage des différents clusters. Enfin, nous verrons comment les différents outils de la chaîne CI/CD ont été déployés.

4.1 Architecture des clusters Kubernetes

Comme nous l'avons déjà évoqué, la plateforme PaaS est composée de plusieurs clusters *Kubernetes*. Nous avons procédé à un premier choix de répartition basé sur les rôles de chacun au sein de la chaîne de déploiement continu. À n'en pas douter, l'expérience et l'évolution de nos usages nous amènera à repenser cette répartition.

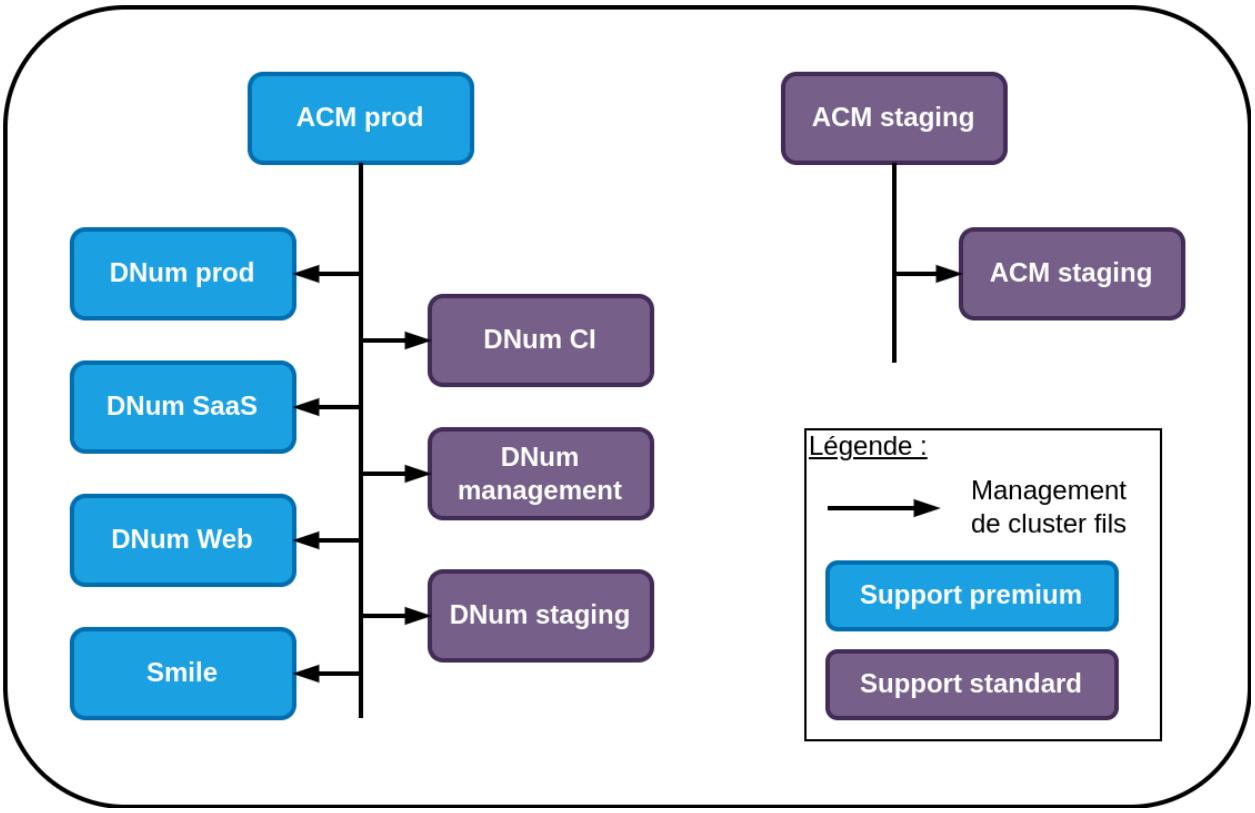

Figure 3: Liste des clusters OpenShift

La plateforme de production est la plus conséquente, et est constituée des clusters suivant :

- ACM prod : cluster maître (*Advanced Cluster Management*) ;
- DNum management : hébergement des outils CI/CD et d'administration ;
- DNum CI : hébergement des *runners Gitlab* ;
- DNum staging : hébergement des instances de développements, tests, pré-production, etc. des applications ;

- DNum prod : hébergement des instances de production des applications destinées à l'Unistra ;
- DNum SaaS : hébergement des instances de production des applications proposées en SaaS par l'Unistra ;
- DNum Web : hébergement des serveurs web Unistra ;
- Unicaen : hébergement des instances d'applications proposées en SaaS par l'Université de Caen.

En complément, nous avons également déployé une seconde plateforme PaaS de *staging* composée de deux clusters. Celle-ci à vocation à être utilisée par les administrateurs *Kubernetes* pour préparer les mises à jour de la plateforme, les modifications de configuration ou tester des nouvelles fonctionnalités. Elle est composée de deux clusters, mais peut temporairement en contenir d'autre :

- ACM staging : cluster maître hébergeant *Advanced Cluster Management* de test ;
- OPP staging : hébergement des outils CI/CD et des *workloads* de tests.

4.2 Architecture OpenShift on OpenStack

Pour des raisons d'optimisation de l'utilisation des ressources physiques des serveurs, nous avons décidé de déployer nos premiers clusters *Kubernetes* sous forme de serveurs virtuels. C'est évidemment vers notre propre plateforme IaaS *OpenStack* que le choix de l'hébergement s'est tourné.

L'alternative du déploiement *bare metal*, bien qu'avantageuse en termes de puissance et potentiellement économique à grande échelle, nécessite un niveau de charge élevé pour être justifiée, ce qui peut générer un gaspillage de ressources sur une petite infrastructure. La flexibilité offerte par la virtualisation nous permet de mieux adapter la taille des nœuds *Kubernetes* en fonction des besoins et de mutualiser les ressources entre *Kubernetes* et *OpenStack*. Une migration vers le *bare metal* pourrait être envisagée à long terme, mais seulement si l'évolution des besoins la rend pertinente.

Comme présenté dans la figure suivante, chaque cluster fait l'objet d'un projet dédié au sein d'*OpenStack*.

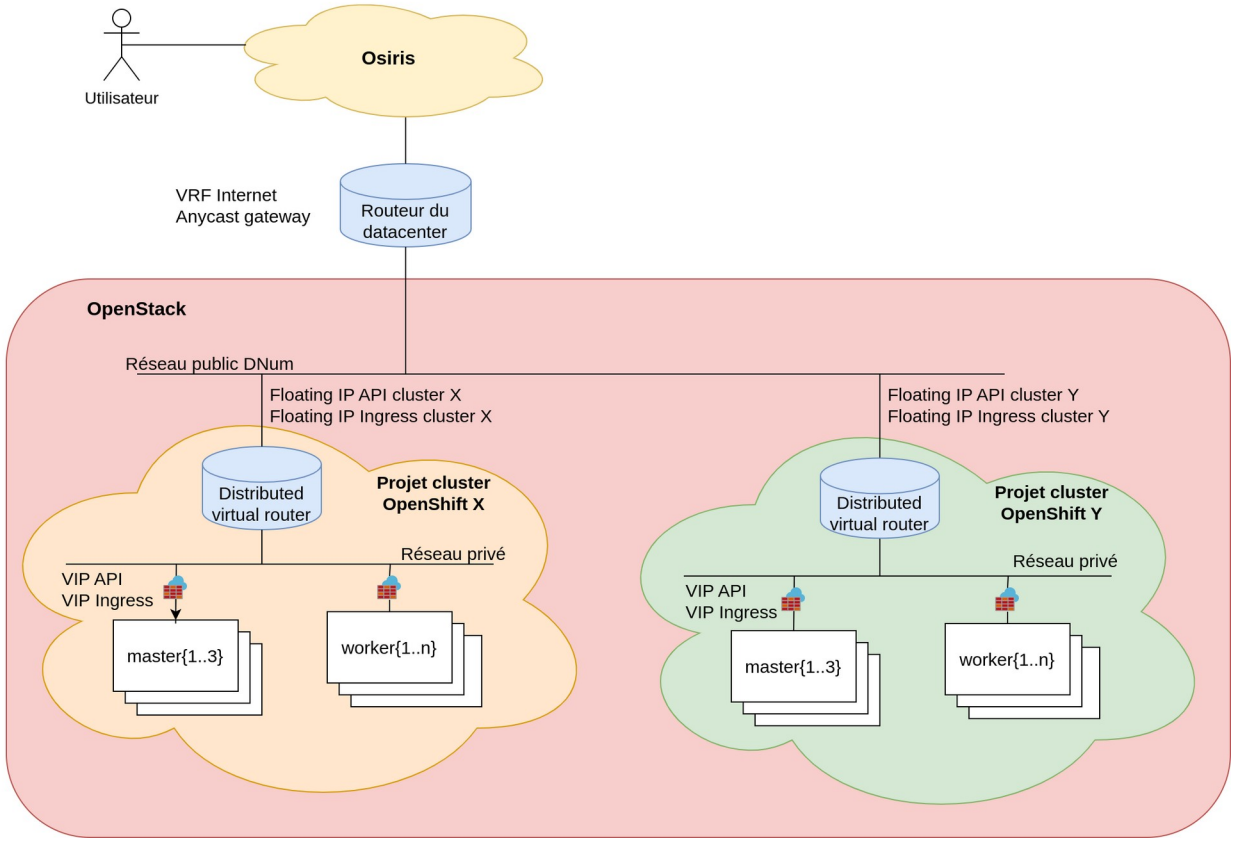

Figure 4: Intégration dans Openstack

Par défaut, chaque projet est autonome et complètement isolé des autres. Il dispose de ses propres quotas sur les ressources (vCPU, RAM, stockage, etc.). Nous avons donc la possibilité d'estimer plus facilement les ressources IaaS consommées par chaque cluster, et ainsi faciliter la facturation d'une future offre de service.

Ce découpage nous permet également de séparer les droits d'accès. Pour gérer le provisionnement des ressources sur *OpenStack*, l'installeur *OpenShift* utilise l'API de ce dernier en s'authentifiant avec un compte applicatif dédié, appelé "*application credentials*" dans le jargon *OpenStack*. Par définition, un tel compte n'a accès qu'à un seul projet. Séparer les clusters par projet nous a ainsi permis de contraindre les droits de l'*application credential* à son cluster respectif. Cette séparation présente donc d'une part un intérêt de sécurité : la compromission d'un de compte ne permettrait ainsi d'accéder qu'au projet *OpenStack* d'un seul cluster. Elle évitera également que, par une éventuelle erreur humaine ou bug d'un outil de *provisioning*, un cluster soit impacté pendant une modification sur un autre.

4.3 Déploiement des clusters

Le déploiement des clusters *OpenShift* a été effectué, selon les cas, par deux méthodes différentes.

La première concerne le déploiement des clusters *ACM* (*Advanced Cluster Management*). Ce déploiement a été réalisé manuellement à l'aide de l'outil d'installation *Red Hat* en ligne de commande, avec un paramétrage adapté à notre environnement *OpenStack*. Cette méthode ne présente que peu d'intérêt technique, se contentant de suivre pas à pas la documentation. Nous avons estimé qu'il n'était pas nécessaire d'automatiser ces étapes, car seuls les clusters *ACM*

seraient installés de cette manière. Une fois les configurations ajustées, l'installation de l'*Advanced Cluster Management* a été effectuée via l'opérateur dédié au sein d'*OpenShift*.

Le second mode de déploiement concerne les clusters enfants, qui sont déployés par le cluster *ACM* lui-même. Bien que l'outil propose un assistant graphique accessible via le tableau de bord *ACM*, nous avons choisi de ne pas l'utiliser. Tout d'abord, pour des problèmes de compatibilité avec notre installation *OpenStack*, certains paramètres clés n'étant pas modifiables via cet assistant. De plus, notre objectif étant de pouvoir versionner et rejouer les créations de clusters, l'utilisation de cet assistant ne répondait pas à nos besoins.

Pour répondre à ces exigences, nous avons opté pour l'utilisation d'objets *Kubernetes*, notamment les *Custom Resource Definitions* (CRD) spécifiques à *OpenShift*, ainsi que les secrets poussés vers le cluster *ACM*. Afin de simplifier et d'automatiser ce processus de déploiement, nous avons créé un chart *Helm* dédié. Ceci nous a permis de déployer plusieurs clusters, en ne spécifiant pour chacun que le fichier de configuration propre à chaque cluster (cf. Déploiement d'un cluster Openshift).

4.4 Configuration des clusters

Successive à leur déploiement, la configuration des clusters permet de les spécialiser dans leur rôle et de les rendre utilisables. Toujours pour rester dans l'optique de reproductibilité et d'historisation de la configuration, nous avons choisi pour ce faire d'utiliser le système de gestion de la gouvernance proposé à *ACM*. Derrière ce nom abscons se cache un système de *policies*, un framework permettant de vérifier la conformité de la configuration avec une règle décrite sur le cluster *ACM*. En cas de non-conformité de la configuration il est possible soit de simplement informer les administrateurs de l'incohérence, soit de forcer l'application de la règle. Les *policies* ne se limitent pas à une sélection restreinte de paramètres qui auraient été implémentés au sein d'*ACM*. Elles permettent de vérifier la conformité de n'importe quel objet *Kubernetes* sur un cluster.

Sans avoir la prétention de remplacer la très complète documentation *OpenShift*, nous allons essayer d'expliquer synthétiquement la constitution d'une *policy*. Pour donner une idée du résultat, nous présenterons un exemple de *policy* vérifiant l'existence d'un rôle, et le créant le cas échéant. *Kubernetes* oblige, la définition d'une *policy* et son application se présentent évidemment sous la forme de CRD. Trois différents objets (*kind*) entrent en jeu.

- *Policy* : la déclaration de la règle à vérifier elle-même, ainsi que certaines options d'évaluation. Une seule *policy* peut exiger la vérification de la conformité de la configuration de plusieurs objets *Kubernetes*.

Figure 5: Exemple de policy

Comme le nom des attributs *policy-templates* et *object-templates* l'indiquent, nous avons ici sous les yeux non pas une définition statique, mais des éléments sous forme de *template* au format Go. Il est donc possible d'utiliser des variables, dont les valeurs de substitutions pourront être récupérées dans des objets *Kubernetes*, aussi bien sur le contrôleur *ACM* que sur le cluster enfant appliquant la *policy*.

- *Placement* : un filtre permettant d'identifier un ou plusieurs clusters. Ce filtre peut utiliser les métadonnées de l'objet *ManagedCluster* géré par *ACM*. On y retrouve ainsi le nom, des *labels*, des *taints*, des *tolerations*, etc. Un *placement* peut également limiter le nombre de clusters sélectionnés.

Figure 6: Exemple de Placement

- *PlacementBinding* : cet objet associe la *policy* et son *placement*. Cette indirection permet d’utiliser un *placement* pour plusieurs *policies*, et inversement d’appliquer une même *policy* selon plusieurs filtres d’associations différentes.

Figure 7: Exemple de PlacementBinding

Les *policies* sont donc un outil très polyvalent pour la gestion des clusters. Elles peuvent permettre aussi bien de faire de la vérification, de la configuration, et même d’aller jusqu’au déploiement d’applications quand ces dernières sont disponibles via des opérateurs.

À ce jour, nous les avons donc utilisées pour configurer plusieurs éléments dont voici quelques exemples non exhaustifs :

- la classe de stockage par défaut ;
- les rôles et habilitations par défaut sur chaque cluster ;
- la configuration d’*AlertManager* ;
- l’authentification *OpenID* vers un serveur *Keycloak*, pour lui déléguer la gestion des identités et l’authentification multi-facteur.

Nous avons également utilisé les *policies* pour déployer, par l’intermédiaire d’opérateurs, les applications suivantes sur un ou plusieurs clusters :

- *Certmanager* ;

- *ArgoCD* ;
- *Quay* ;
- l'agent *Vault*.

En conclusion, le schéma suivant synthétise les étapes de déploiement d'un cluster enfant.

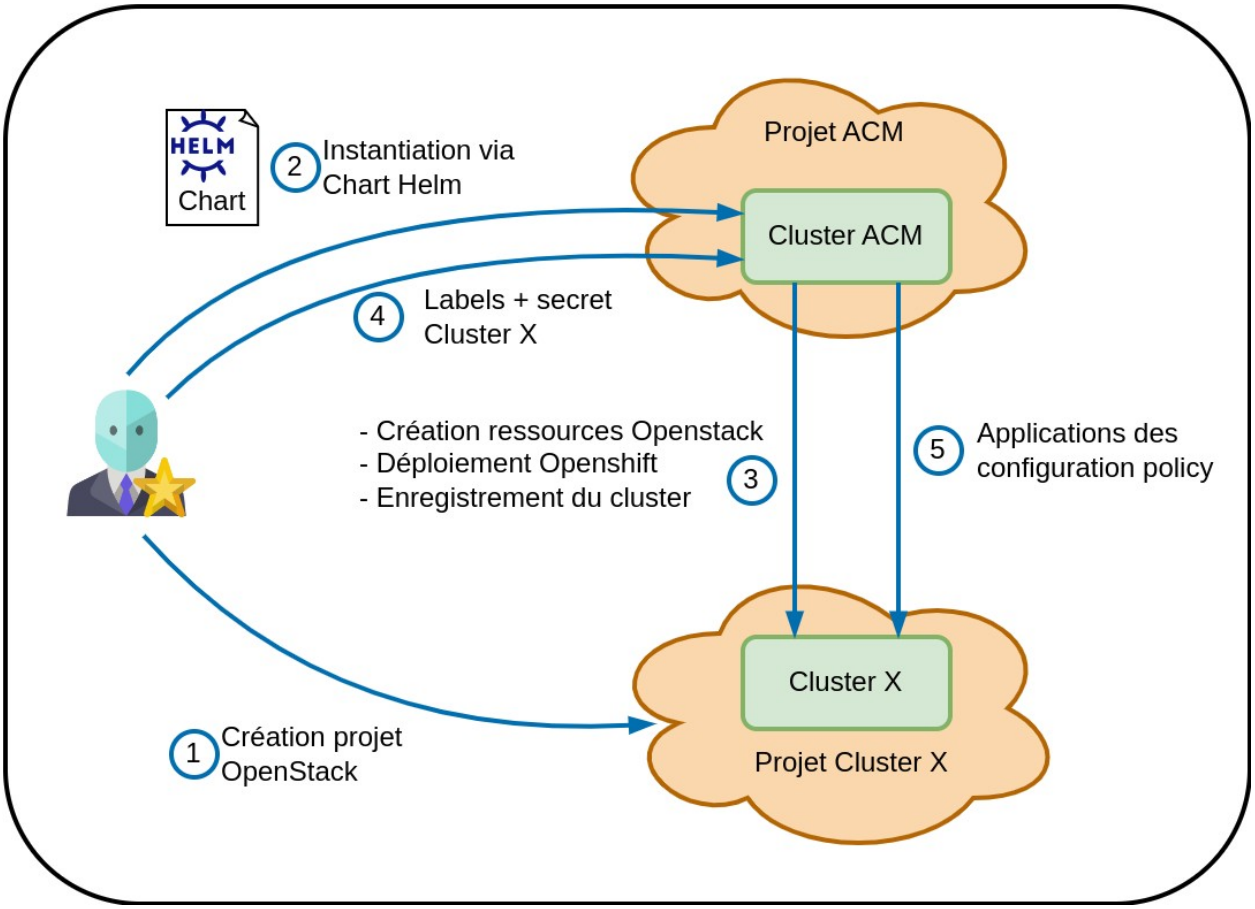

Figure 8: Déploiement d'un cluster Openshift

Sur les différentes étapes, il reste aujourd'hui une étape manuelle, l'étape 4 concernant l'application des labels et des secrets sur le cluster. Une prochaine itération sur ce processus nous amènera vraisemblablement à améliorer ces points. Nous pourrions en effet utiliser *ArgoCD* et Vault pour permettre de simplifier encore le déploiement, et entrer dans une réelle démarche *GitOps* sur l'ensemble de ces opérations.

5 Utilisation de la plateforme

Dans cette partie, nous nous concentrerons sur l'utilisation de la plateforme PaaS, en mettant l'accent sur plusieurs aspects clés de son fonctionnement. Nous aborderons tout d'abord les mesures de sécurisation que nous avons intégrées, en particulier celles liées à l'authentification et à la gestion des accès. Ensuite, nous examinerons le processus de *provisioning* des environnements applicatifs et le déploiement d'applications sur la plateforme. Enfin, nous partagerons des retours d'expérience sur les premières applications pilotes que nous avons sélectionnées pour tester et valider ces processus.

5.1 Authentification et sécurisation

Lors de la conception de notre plateforme PaaS, nous avons accordé une attention particulière à la sécurité dès le début, notamment sur l'authentification. *Keycloak*, que nous avions précédemment étudiée pour notre plateforme IaaS, nous a semblé être une solution adaptée. Ses fonctionnalités d'authentification à double facteur et de gestion centralisée des rôles et groupes contribuent à améliorer la sécurité de la plateforme, tout en permettant une gestion fine des accès.

Keycloak récupère les comptes utilisateurs directement à partir de notre annuaire *LDAP*, garantissant une intégration avec notre système d'identités existant. Nous utilisons ensuite son système de rôles pour représenter les accès aux différents projets sur *OpenShift* ainsi qu'aux outils associés. À chaque rôle est associé un groupe d'utilisateurs, également défini dans notre *LDAP*, qui regroupe les mainteneurs d'une application ou d'une équipe particulière. Cela nous permet de centraliser la gestion des accès tout en maintenant une flexibilité et un contrôle sur les permissions.

En complément de l'intégration avec notre annuaire *LDAP*, nous utilisons également la base locale de *Keycloak* pour la création des comptes administrateurs de la plateforme. Cela nous permet d'assurer une séparation et une isolation claire entre les utilisateurs standards et les administrateurs. Ce découplage renforce la sécurité en limitant l'exposition des comptes à privilèges.

5.2 Provisioning des environnements applicatif

L'automatisation du provisionnement d'environnements applicatifs sur nos plateformes PaaS a été mise en place avec trois objectifs principaux :

- la **délégation** du provisionnement aux équipes métiers avec un système de validation, garantissant le respect des processus tout en responsabilisant les équipes ;
- l'**automatisation complète** sur l'ensemble de la stack technologique, incluant *OpenShift*, *ArgoCD*, *Quay*, et *HashiCorp Vault* ;
- l'**homogénéisation** de la gestion des droits d'accès avec une granularité fine permettant de définir des permissions précises par application et par équipe.

Celle-ci repose sur un *workflow* orchestré par *AWX*, *Ansible* et *Terraform*. Cette approche garantit la création rapide et reproductible d'environnements tout en assurant leur conformité aux besoins spécifiques des projets.

Lorsqu'un utilisateur souhaite provisionner un nouvel environnement, il commence par remplir un formulaire sur l'interface *AWX*. Ce formulaire contient plusieurs paramètres, notamment :

- **plateforme de destination** : permet de spécifier si l'application est à destination de l'Unistra ou si celle-ci sera proposée en SaaS à d'autres établissements ;
- **nom de l'instance applicative** : définit le nom de l'instance qui sera déployée ;

- **environnement** : précise si l'environnement est de type production, staging, ou développement ;
- **groupes gestionnaires** : liste des équipes responsables de la gestion de l'environnement ;
- **ressources allouées** : spécifications des quotas de CPU, RAM, et stockage persistant pour l'environnement, selon les besoins.

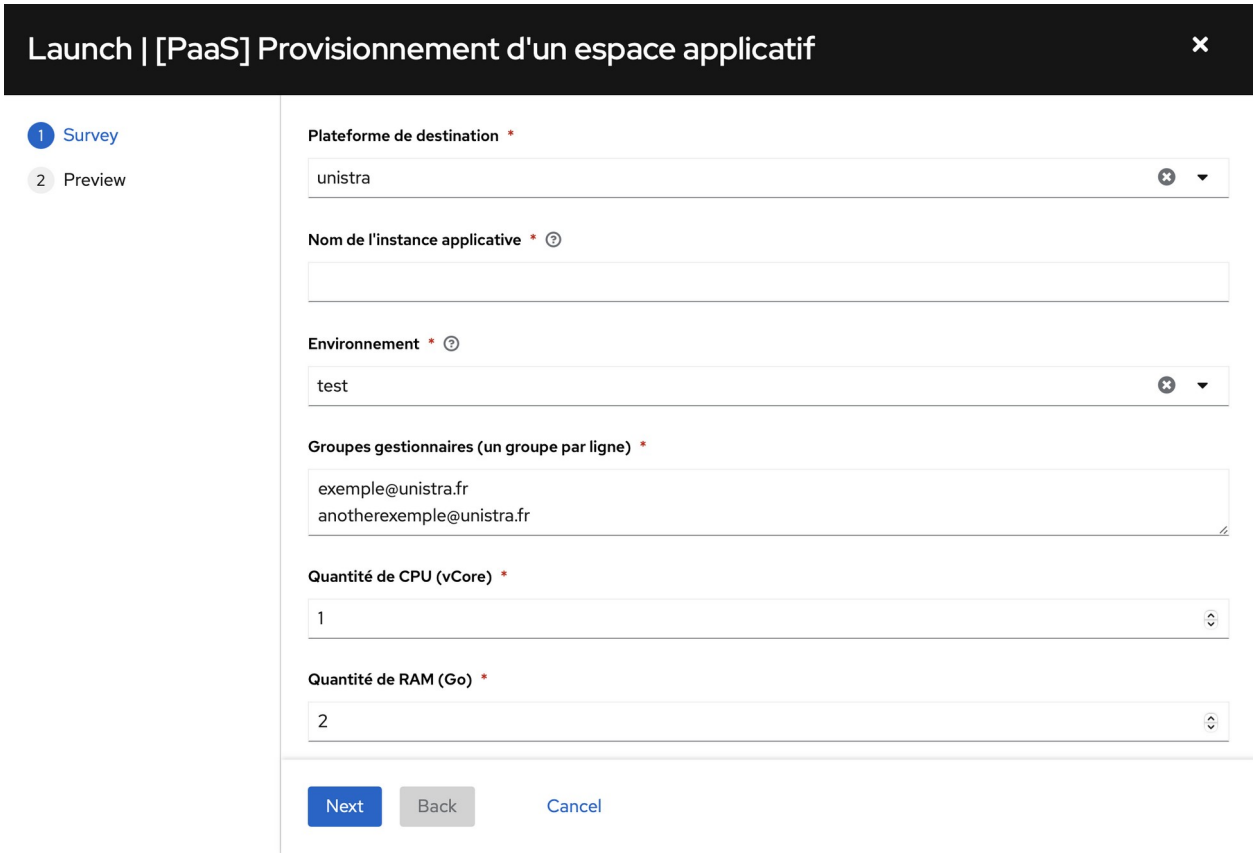

Figure 9: Formulaire de création d'un environnement applicatif

Une fois le formulaire soumis, *AWX* exécute un rôle *Ansible* qui prend en charge plusieurs actions essentielles :

- **vérifications d'usage** : un contrôle préalable est effectué afin de s'assurer des informations entrées par l'utilisateur (nom, existence préalable de l'environnement, etc) ;
- **création de la configuration** : *Ansible* génère automatiquement le fichier de configuration nécessaire pour provisionner l'environnement définitif sur la plateforme PaaS ;
- **création d'une merge request :** *Ansible* gère également la création d'une *merge request* dans le dépôt contenant l'ensemble des environnements applicatifs.

Une fois la *merge request* réceptionnée, l'équipe chargée de la plateforme PaaS s'assure de la légitimité de la demande et de l'allocation suffisante des ressources nécessaires pour le nouvel environnement. Après validation, *Terraform* est chargé de provisionner les ressources ainsi que les accès sur les différents outils :

- **Keycloak :** création d'un rôle et association du rôle aux différents groupes mainteneurs ;

- **OpenShift** : création d'un *namespace*, des limites de quotas, des roles bindings, de l'authentification vers *Quay* ainsi que vers *Vault* ;
- **ArgoCD** : création du projet ainsi que des droits associés ;
- **Quay** : création d'une équipe associés aux groupes de mainteneurs ainsi que d'un compte robot ;
- **Vault** : création d'une *policy Vault* dédiée associés aux groupes de mainteneurs permettant de limiter la visibilité au strict nécessaire.

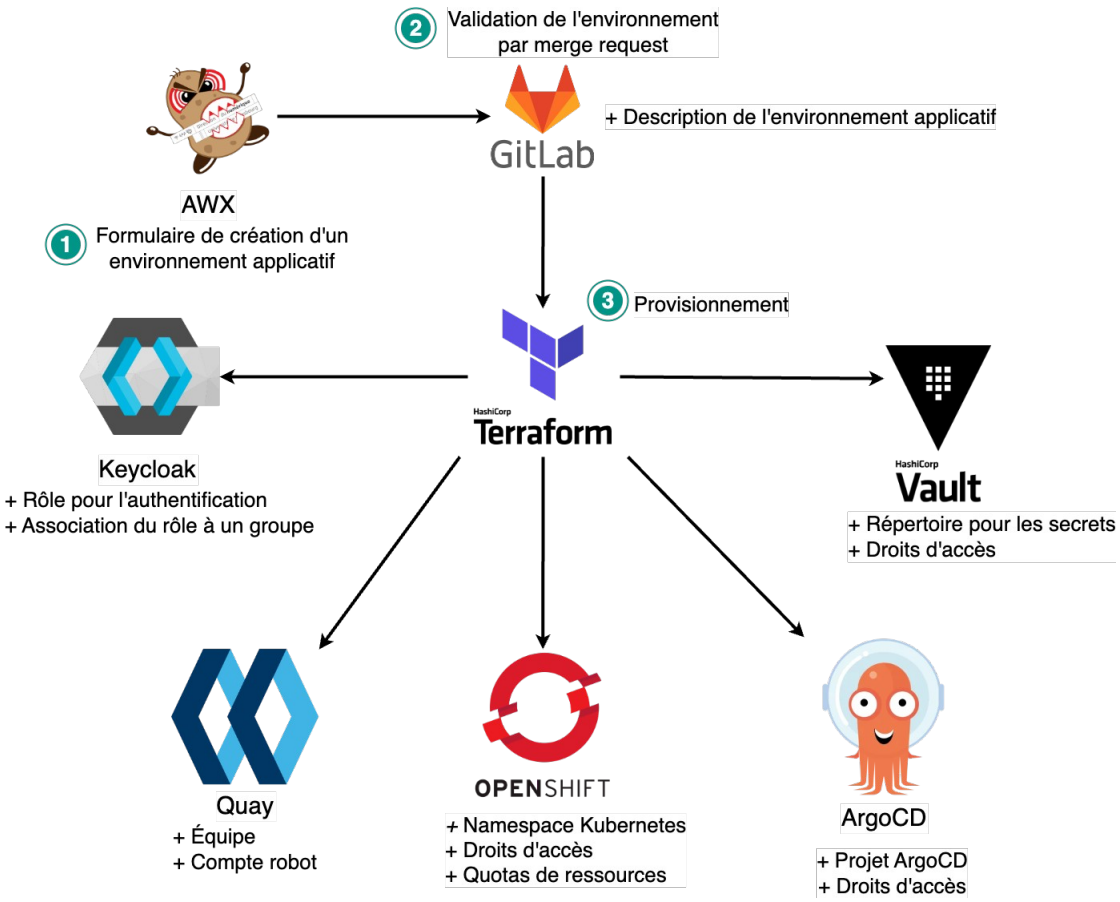

Figure 10: Processus de création d'un environnement applicatif

L'environnement étant provisionné, les groupes mainteneurs peuvent maintenant accéder à la plateforme PaaS, avec une visibilité limitée sur les ressources, pour déployer leur application. Cette automatisation permet de réduire considérablement le temps nécessaire à la mise en place d'un nouvel environnement, tout en standardisant les configurations et en limitant les interventions manuelles, souvent sources potentielles d'erreurs.

5.3 Déploiement d'une application

Dans notre démarche *GitOps*, nous souhaitons que l'intégralité du cycle de vie des applications soit gérée via des dépôts *Git*, offrant ainsi un contrôle précis et versionné de toutes les configurations.

Nous utilisons *ArgoCD* en mode *App of Apps*, permettant d'organiser proprement les déploiements. Une application racine est chargée de synchroniser les différentes applications enfants, chacune correspondant à une instance de l'application à déployer. Ce modèle nous permet de gérer les environnements applicatifs tout en garantissant la cohérence et la traçabilité des configurations.

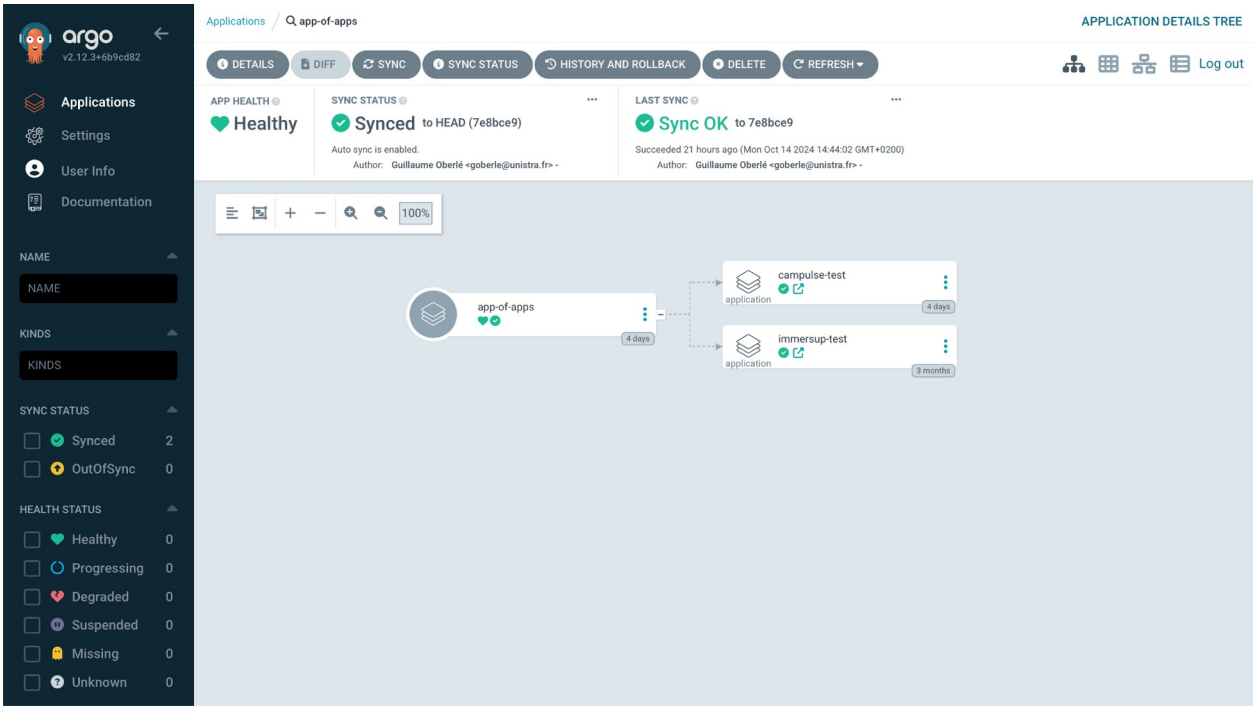

Figure 11: ArgoCD : App of Apps

Dans cet exemple, nous observons que le système *App of Apps* est chargé de déployer deux applications enfants : *Campulse* et *Immersup*. Ces deux applications sont déclarées sous forme de fichiers YAML directement dans un dépôt *Git* supervisé par le mécanisme *App of Apps* d'*ArgoCD*.

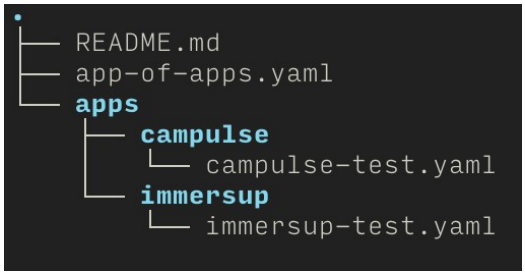

Figure 12: Arborescence d'un dépît Git d'une App of Apps

Pour chacune des applications enfants, on retrouve un fichier de définition de l'application à déployer. Celui-ci contient le projet *ArgoCD* dans lequel l'application sera déployée, le cluster de

destination et les dépôts où sont stockés les *charts Helm* ou les fichiers *Kustomize*, précisant les versions à utiliser.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: campulse-test
  namespace: openshift-gitops
spec:
  project: campulse-test
  sources:
    - repoURL: https://xxx.unistra.fr/di/helm/campulse-front.git
      path: .
      targetRevision: 1.0.1
      helm:
        valueFiles:
          - values.yaml
    - repoURL: https://xxx.unistra.fr/di/helm/campulse-back.git
      path: .
      targetRevision: 1.0.1
      helm:
        valueFiles:
          - values.yaml
    - repoURL: https://xxx.unistra.fr/di/helm/campulse-bdd.git
      path: .
      targetRevision: 1.0.1
      helm:
        valueFiles:
          - values.yaml
  destination:
    server: https://xxx.unistra.fr:6443
    namespace: campulse-test
```

Figure 13: Exemple de définition d'une application ArgoCD

En supervisant continuellement ce *repository*, le mécanisme d'*App of Apps* s'occupe de déployer automatiquement ou manuellement, selon sa configuration, les applications enfants ainsi déclarées.

Pour assurer la qualité, la cohérence et la sécurité des déploiements, ce dépôt *Git* doit faire l'objet d'une attention particulière. Ayant peu de recul pour le moment sur le sujet, nous n'avons pas totalement statué sur cet aspect. Nous envisageons toutefois d'utiliser le mécanisme des branches protégées et le système de *merge request* associé à une politique d'approbation.

5.4 Les pilotes

La mise en place et l'utilisation d'une Infrastructure PaaS sont des modifications structurantes dans la façon d'opérer l'exploitation des applications, avec beaucoup de variantes possibles et de choix à effectuer. Il nous a vite paru évident que le projet ne s'arrêterait pas au déploiement d'une plateforme livrée aux équipes applicatives, mais devait prendre en compte l'intégration d'applications pilotes pour, d'une part, adapter le PaaS à la réalité du terrain, mais également pour définir une organisation et des bonnes pratiques dans l'usage de ces outils. Nous avons donc sélectionné des applications représentatives de l'écosystème Unistra, et pour lesquelles l'hébergement sur une plateforme PaaS aura une réelle valeur ajoutée. Il s'agit donc des applications qui sont ou seront proposées sous forme d'une offre SaaS à la communauté ESR :

Immersup (immersion des lycéens dans l'enseignement supérieur) et *Campulse* (gestion des associations étudiantes)

Concernant l'offre de service ESR, nous nous sommes rapprochés de l'Université de Caen. Leur projet était de proposer à la communauté un hébergement SaaS de l'application *Smile* (gestion de la mobilité étudiante), mais ils ne disposaient pas des ressources humaines nécessaires pour déployer et maintenir une plateforme de type PaaS. Cette opportunité nous a permis d'en faire les pilotes d'une offre de service ouverte à l'extérieur de notre propre direction.

5.4.1 Immersup et Campulse

L'application *Immersup*, développée par l'Unistra, permet de gérer "l'immersion" des lycéens dans le monde de l'enseignement supérieur. *Campulse* est également développée par l'Unistra et répond au besoin de la gestion des associations étudiantes au sein de l'université. Évidemment utilisées par l'Unistra, ces deux applications sont ou seront proposées en hébergement SaaS à la communauté.

Ces deux applications ont été retenues comme pilotes pour la plateforme PaaS pour deux raisons principales. La première raison est leur adéquation technique. Ce sont deux applications web d'architecture n-tiers développées en P*ython Django* et *NodeJS* (*Immersup*). Leur modèle est donc techniquement adapté au déploiement sous forme de conteneur. Réalisées récemment et toujours en évolution, elles utilisent des technologies et éléments logiciels à jour et toujours maintenus. Les risques de difficultés à exécuter ces applications dans un environnement PaaS seront donc limités. Elles présentent toutefois des interactions avec des systèmes externes (stockage objet, authentification *SAML*, émission de courriel, etc.) qui devront potentiellement faire l'objet d'une attention particulière.

De par ces caractéristiques, elles sont représentatives de la majorité des applications déployées actuellement. Elles permettront ainsi d'établir une base de bonnes pratiques.

L'autre raison est le bénéfice à héberger ces applications sur une telle plateforme. Étant proposées en SaaS, elles exigent vraisemblablement le déploiement d'un nombre non négligeable d'instances. La plateforme PaaS présente ainsi tout son intérêt pour faciliter la reproductibilité des déploiements et la simplicité des montées de version.

Les travaux d'adaptation de ces applications à la plateforme PaaS sont en cours. À ce jour, deux ateliers, réunissant développeurs, administrateurs d'applications et administrateurs PaaS, ont été réalisés pour travailler sur le sujet. Les travaux suivants ont été réalisés :

- prise en main des différentes briques logicielles du PaaS ;
- adaptation du paramétrage des applications du PaaS ;
- adaptation des applications (back-end et front-end) pour l'exécution en conteneur ;
- création de *charts Helms* pour le déploiement de ces applications ;
- standardisation de la chaîne CI/CD : organisation G*itlab* des ressources, organisation *ArgoCD*, découpage des *namespaces* dans *Quay*, etc. ;
- maquettage d'opérateurs de base de données (CloudNative PG).

Les travaux sont à poursuivre, et nous espérons déployer nos premières instances en production pour début 2025.

5.4.2 Offre ESR : Université de Caen

Dans la théorie, nous envisageons deux versions de l'offre aux extérieurs :

- une mise à disposition de *namespace Kubernetes* au sein d'un cluster *Kubernetes* mutualisé, qu'on se permettra appeler *Namespace-as-a-Service* (NaaS), adaptée aux structures ayant un peu d'applications à héberger ;
- une mise à disposition d'un cluster *Kubernetes* complet ou *Cluster-as-a-Service* (CaaS), pour les structures ou établissements ayant des objectifs plus ambitieux et souhaitant plus d'autonomie.

Sur ces deux profils d'offres, nous avons aujourd'hui des interrogations sur le périmètre du service auxquelles nous devrons répondre avant d'établir une offre de service standard. Le premier sujet concerne le périmètre applicatif de l'offre. Les outils annexes de CI/CD (*runner* CI, *Quay*, *ArgoCD*, *Vault*) doivent-ils être proposés ? Ces services doivent-ils être "à la carte" ? Ou bien doit-on considérer que le client dispose de ses propres outils ? Sur un autre volet, dans le cas d'un service de type CaaS, la question de la limite de responsabilité est de mise. L'établissement doit-il être administrateur du cluster ? Est-il responsable des mises à jour du cluster ? C'est afin de clarifier ces questions que l'opportunité d'avoir un établissement pilote nous a séduits.

Ce partenariat avec l'Université de Caen a été réalisé sur un service de CaaS. L'objectif de Caen étant de fournir plusieurs instances de leur application, ils correspondaient au profil de charge requis pour bénéficier d'un cluster entier. Nous avons, pour cette expérimentation, choisi de répondre à Caen avec une offre de service correspondant exactement à leurs besoins. Mais nous sommes bien conscients que cette philosophie ne supportera pas le passage à l'échelle : nous devrons à terme proposer une offre standardisée, avec éventuellement quelques options. L'adaptation aux besoins spécifiques de chaque partenaire semble trop coûteuse en exploitation à long terme.

C'est ainsi que les choix suivants ont été effectués :

- l'Unistra reste administrateur du cluster, nous sommes responsables des mises à jour ;
- l'Unicaen bénéficie de l'autonomie nécessaire pour créer des *NameSpace* au sein du cluster ;
- l'Unistra fournit et opère une instance de *Quay*, *ArgoCD* et *Vault*.

Ainsi, depuis début 2024, l'Université de Caen a pu prendre en main le service PaaS et a initié l'hébergement de leur application *Smile* dessus. À ce jour, trois instances de production de *Smile* sont hébergées sur la plateforme, dont deux en production. Le passage à la phase 2, qui étend le service proposé à d'autres universités, est imminent. Une seconde application est en cours d'intégration sur la plateforme : *StageTrek*. Plus d'une dizaine d'établissements sont intéressés pour bénéficier de cet outil en SaaS. Enfin, des travaux sont encore en cours concernant deux autres applications : *Esup ORA* et *EMC2*.

6 Le retour d'expérience

Dans ce chapitre, nous allons tenter de synthétisernotre retour d'expérience sur les différents éléments qui constituent ce projet. Pour chaque rubrique, nous évoquerons les points positifs, les difficultés rencontrées et les évolutions à apporter.

6.1 La plateforme PaaS

6.1.1 Les points faibles

L'un des premiers aspects à prendre en compte est l'empreinte de base d'*OpenShift*, qui consomme une quantité importante de ressources dès son installation, même pour un usage modéré. Cette consommation peut poser problème dans des environnements où les ressources sont limitées.

De plus, certaines modifications, bien que mineures en apparence, nécessitent souvent de plonger dans la documentation. Cela est particulièrement vrai pour les opérateurs, où la recherche de solutions peut parfois s'avérer plus complexe que prévu. Nous pensons par exemple à la modification de la classe de stockage par défaut, ou encore la configuration des *Ingress*.

Sur le plan financier, *OpenShift* reste une solution onéreuse, avec un modèle de souscription qui peut rapidement peser sur les budgets. Si la DNum a fait le choix en connaissance de cause dans le cadre de ce projet, des potentiels partenaires d'une offre ESR pourraient être freinés par le coût des souscriptions. Qui plus est, nous constatons avec le temps qu'*OpenShift* promet à l'avenir de pouvoir répondre à d'autres cas d'usages intéressants (ex : intelligence artificielle, virtualisation). L'extension du périmètre sur ces domaines aurait un impact non négligeable sur le coût des souscriptions. Nous espérons donc que des options plus abordables pourront être proposées à l'avenir par *Red Hat*, par exemple une souscription libératoire à tarif raisonnable.

6.1.2 Les points forts

Malgré ses inconvénients, *OpenShift* offre un environnement extrêmement complet sur le plan fonctionnel, permettant de déployer rapidement et efficacement une plateforme PaaS sans avoir à configurer chaque détail manuellement. *OpenShift* s'intègre par ailleurs de manière fluide avec *OpenStack*, ce qui nous a permis de tirer parti de l'infrastructure existante tout en apportant une cohérence dans la gestion de l'ensemble de notre environnement cloud.

Les mises à jour sont également un atout majeur. Leur simplicité nous permet de maintenir la plateforme à jour sans interruption de service ni complexité accrue.

La documentation, très complète, est une autre force d'*OpenShift*, fournissant une aide précieuse lors des déploiements ou des configurations complexes. À ce jour, elle nous a permis de contourner toutes les spécificités de la distribution évoquées plus haut, et d'utiliser les fonctions avancées, comme les *policies ACM*.

Enfin, en termes de sécurité, en comparaison avec d'autres distributions *Kubernetes*, *OpenShift* applique des principes « by-design », avec par exemple l'utilisation de conteneurs *rootless* et d'UID d'exécution aléatoire et hors des plages système. Si ces précautions peuvent poser des difficultés de fonctionnement avec des *charts Helm* communautaires, elles renforcent la sécurité de base de l'Infrastructure.

Concernant les applicatifs de la chaîne CI/CD (*ArgoCD*, *Vault*, *Quay*), nos premiers avis sont conformes à ce que nous évoquions dans les tests de sélection. Tous montrent satisfaction sur nos usages actuels, et nous sommes loin de les avoir poussés dans leurs limites fonctionnelles. Nous

aurons vraisemblablement un retour d'expérience plus abouti sur ces outils lorsque nous aurons plus d'utilisateurs de la plateforme et d'applications déployées.

6.1.3 Les prochaines étapes

Parmi nos prochaines priorités figure la consolidation du déploiement de nos clusters. En effet, en échangeant avec d'autres administrateurs de plateforme PaaS, nous avons réalisé qu'il était pertinent d'adopter une démarche *GitOps* sur cette partie. De cette manière, nous pourrions optimiser l'automatisation des déploiements et des configurations, en garantissant le suivi des modifications. Notre base actuelle se prête à cette évolution : il nous suffirait d'utiliser une instance *ArgoCD* pour déployer notre chart *Helm* de déploiement de cluster enfant ainsi que les *policies* de configuration.

Nous souhaitons également appliquer cette approche aux *provisioning* des environnements applicatifs, en utilisant ces mêmes outils. En effet, la méthode actuelle n'adopte pas entièrement la logique *GitOps*. Nous envisageons donc dans le futur de nous séparer de *Terraform* au profit d'*ArgoCD* pour le *provisioning* des différents éléments. La seule contrainte actuellement étant *Quay*, pour lequel il n'existe aujourd'hui pas d'objets *Kubernetes* permettant d'interagir avec l'outil.

Renforcer la sécurité reste également une priorité clé, avec l'exploration de fonctionnalités dont nous disposons telles qu'*Advanced Cluster Security* ou mettre en place des *policies ACM* de sécurité. En parallèle, nous souhaitons évaluer d'autres outils tierces permettant de surveiller l'activité en temps réel, comme *Falco* ou *Neuvector*.

6.2 La démarche PaaS Unistra

Comme nous l'avons expliqué dans cet article, à l'écriture de ces lignes, nous n'avons pas encore d'applications en production sur notre plateforme PaaS. Nous pensons cependant avoir quelques éléments à partager.

6.2.1 Les difficultés rencontrées

Le plus gros point de vigilance qu'on peut identifier sur ce projet est bien évidemment son coût global, principalement en termes de ressources humaines. En effet, la transition vers une démarche SaaS/PaaS telle que nous l'avons envisagée modifie grandement les technologies et pratiques, aussi bien au niveau infrastructure qu'au niveau applicatif. La pile technologique dans son ensemble peut paraître complexe pour un débutant : conteneurs, orchestration par *Kubernetes*, packaging d'applications *Helm*, intégration continue, déploiement continu, gestion de secrets ; chacun de ces éléments est un sujet en soi à appréhender. Des cycles de formation sont évidemment à prévoir. Mais surtout, une mise en pratique rapide est nécessaire car seule l'expérience concrète permet d'acquérir une compréhension solide de ces sujets.

Autre difficulté : dans un contexte où la charge d'exploitation est constante pour tous les membres de la DNum et où projets sont simultanément en phase de réalisation, il est parfois compliqué de pouvoir réunir les bons acteurs issus des différentes équipes pour pouvoir avancer efficacement. Nous ne pouvons donc que recommander à un établissement souhaitant se lancer dans l'aventure une forte priorisation de ce genre de projet.

6.2.2 Les bénéfices espérés et constatés

Bien que cette démarche ne soit pas un long fleuve tranquille, elle nous paraît nécessaire et bénéfique à long terme.

D’une part, elle est indispensable pour garantir la modernité technique de nos infrastructures et pratiques. Les échanges avec les confrères de l’ESR, les modes de déploiement d’applications et les technologies mises en avant par les éditeurs (notamment l’Intelligence Artificielle et la virtualisation sur *Kubernetes*) nous confirment que le futur de l’hébergement d’application passera par de l’orchestration de conteneurs. Ces technologies sont également indispensables pour permettre d’assumer, à coût humain constant, l’exploitation des services applicatifs toujours plus nombreux, plus utilisés et plus critiques pour le fonctionnement de nos établissements.

Ensuite, si le panel de technologies à aborder est conséquent, notre expérience nous prouve que le défi n’est pas inaccessible. Les premiers essais de déploiement d’applications peuvent être laborieux, mais la capitalisation sur l’acquisition de ces technologies est très positive à chaque itération, leur utilisation devenant à chaque fois plus naturelle.

En réunissant les équipes *Campulse* et *Immersup* pendant 5 jours d’ateliers, nous avons pu mettre en place une instance d’application quasi fonctionnelle. Il reste évidemment des ajustements à réaliser pour aboutir à une version finale et garantir la reproductibilité. Mais les résultats sont très encourageants, même pour les collègues qui avaient peu d’expérience avant ces ateliers.

Enfin, cette démarche s’est avérée très structurante et fédératrice pour notre direction. Elle impose, de par sa transversalité, la collaboration entre équipes et nous permet de définir ensemble un mode de fonctionnement pour l’exploitation de nos applications.

6.2.3 Les prochaines étapes

Notre prochain objectif sur cet axe du projet est bien évidemment d’aboutir sur le passage en production de nos applications pilotes, avec en ligne d’arrivée l’exploitation des offres SaaS de *Campulse* et *Immersup* sur la plateforme.

Nous allons ensuite, de manière logique, étendre cet hébergement à d’autres applications, aussi bien celles développées en interne que des applications “sur-étagère”. Cette extension se fera vraisemblablement par opportunité : nous allons essentiellement privilégier les nouvelles applications, ou profiter des travaux de montée de version pour les applications existantes.

Une autre piste de réflexion concerne la possibilité d’utiliser *OpenShift Serverless*, basé sur *Knative*, dans le cadre du projet “DynamicMood”. Avec pour objectif d’automatiser la correction d’examens issus de la plateforme Moodle, des images de conteneur seraient fournies par la communauté enseignante et seraient déployées à la demande, offrant ainsi une solution extensible et flexible capable de traiter des volumes d’examens variables. Grâce à l’approche *Serverless*, les conteneurs ne seraient instanciés que lorsque nécessaire, optimisant ainsi l’utilisation des ressources.

Pour finir, un autre axe de prospection que nous n’avions pas prévu au début du projet s’est ouvert à nous au cours des derniers mois. Comme beaucoup d’établissements, l’Unistra commence ses premiers démonstrateurs sur des technologies d’intelligence artificielle. Ces usages requièrent l’utilisation de serveurs dédiés, souvent équipés de GPU, avec des profils de charge bien différent d’applications web traditionnelles. Nous allons donc, dans ce cadre, étudier les possibilités proposées par *OpenShift* pour gérer ce genre d’usages.

6.3 L’offre de service ESR

La collaboration avec l’Université de Caen s’est avérée très fructueuse. Nos confrères Caennais ont pu basculer dans le monde de l’orchestration de conteneurs sans avoir à assumer la charge coûteuse de monter une plateforme PaaS. Ils semblent satisfaits de leur expérience et du service que nous leur rendons.

De notre côté, si bien évidemment la mise à disposition de la plateforme n'a pas été à coût nul, nous avons pu capitaliser sur nos réalisations. La charge d'exploitation supplémentaire de la plateforme de Caen est rationalisée avec celle de l'Unistra, ce qui, d'un point de vue national, est bien plus intéressant que si chaque établissement disposait de sa propre plateforme.

Enfin, on ne peut évidemment que se satisfaire de cette démarche collective et souveraine au sein de l'ESR, qui vise à proposer des applications développées par des membres de l'ESR et hébergées sur des infrastructures publiques. Et, dernier point à ne pas négliger, l'expérience humaine accompagnant cette démarche, favorisant l'échange entre établissement, est une plus-value pour chacun de nous. Elle s'avère, de notre point de vue, plus efficiente et émulante qu'une relation avec un partenaire privé.

Nous ne pouvons pas aujourd'hui annoncer que le modèle utilisé avec Caen soit un standard que d'autres établissements souhaiteraient pratiquer. Qui plus est, notre idée de proposer un service *Namespace-as-a-Service* n'a pas encore été mise en pratique. Nous allons donc devoir entrer dans une démarche avec d'autres partenaires potentiels pour prendre connaissance de leurs besoins. La priorité sera sûrement placée sur les partenaires locaux (composantes Unistra, laboratoires mixtes et autres établissements régionaux), sans pour autant exclure une nouvelle opportunité de collaboration nationale.

7 Conclusion

Le projet *PaaS-Partout* de l'Université de Strasbourg est encore en pleine évolution, mais les premiers résultats sont encourageants. Grâce à l'adoption de technologies comme *OpenShift*, *ArgoCD*, et *Gitlab-CI*, nous avons posé les fondations d'une plateforme PaaS flexible et évolutive, capable de répondre à nos besoins internes tout en offrant des perspectives pour la communauté de l'enseignement supérieur et de la recherche.

Cependant, il reste beaucoup à faire. Les défis que nous avons rencontrés, qu'il s'agisse des complexités techniques, des coûts ou de la formation des équipes, nous rappellent que ce type de transformation nécessite un engagement sur le long terme. Nos premiers déploiements pilotes, comme ceux d'*Immersup* et *Campulse*, montrent qu'il est possible d'atteindre nos objectifs, mais il est clair que de nombreuses étapes sont encore à franchir avant de pouvoir prétendre à une offre PaaS pleinement opérationnelle.

Nous restons optimistes et ouverts à l'apprentissage, tout en continuant à ajuster nos pratiques au fur et à mesure des avancées du projet. Notre partenariat avec l'Université de Caen illustre cette approche collaborative, et nous espérons que d'autres établissements pourront bénéficier de cette initiative dans un avenir proche. Le chemin est encore long, mais les fondations posées nous donnent confiance pour la suite.

8 Glossaire

ACM (Advanced Cluster Management) : Outil de gestion multi-cluster fourni par *Red Hat* dans *OpenShift*, permettant de déployer et administrer plusieurs clusters *Kubernetes* depuis une interface centralisée.

App of Apps : Modèle d'organisation utilisé par *ArgoCD* pour gérer des déploiements complexes en supervisant plusieurs applications enfants à partir d'une application parent.

Application Credentials (OpenStack) : Mécanisme d'authentification d'*OpenStack* permettant à des applications de s'authentifier pour accéder à des projets ou des ressources spécifiques. Il offre un moyen sécurisé de limiter les permissions des applications dans un environnement multi-projets.

ArgoCD : Outil open source de déploiement continu basé sur *GitOps* pour *Kubernetes*, permettant de synchroniser les fichiers de configuration d'applications à partir de dépôts *Git*.

Bare Metal : En informatique, désigne l'utilisation directe des ressources matérielles d'un serveur physique sans couche de virtualisation.

Cert-manager : Outil open source pour *Kubernetes* qui automatise la gestion des certificats TLS. Il permet de générer, renouveler et gérer les certificats provenant de différentes autorités de certification (AC), facilitant ainsi la sécurisation des communications au sein des clusters *Kubernetes*.

CI/CD (Continuous Integration / Continuous Deployment) : Ensemble de pratiques qui automatisent l'intégration continue et le déploiement continu de logiciels, souvent via des *pipelines* automatisés.

Cloud-Controller : Composant dans *Kubernetes* qui permet d'intégrer un cluster avec les services d'une infrastructure cloud (par exemple *OpenStack*, AWS, Azure). Le cloud-controller gère des ressources comme les instances de calcul, le stockage et les réseaux, en communiquant avec l'API du fournisseur de cloud pour orchestrer ces services de manière native au sein du cluster *Kubernetes*.

Contrôleur Kubernetes : composant responsable de surveiller l'état du cluster et d'assurer que l'état actuel correspond à l'état souhaité défini par l'utilisateur. Les contrôleurs gèrent des objets spécifiques dans le cluster et appliquent des actions pour maintenir ou ajuster leur état en fonction des besoins.

CRD (Custom Resource Definition) : Fonctionnalité de *Kubernetes* permettant de définir des ressources personnalisées pour étendre ses fonctionnalités au-delà de celles natives.

GitOps : Méthodologie qui consiste à utiliser *Git* comme source unique de vérité pour la gestion des déploiements d'infrastructure et d'applications, en automatisant leur mise en production à partir des commits *Git*.

Helm : Gestionnaire de packages pour *Kubernetes* qui permet de déployer des applications sous forme de "*charts*", facilitant la gestion de la configuration et des dépendances des applications.

IaaS (Infrastructure as a Service) : Modèle de cloud computing qui fournit des ressources informatiques virtualisées via Internet, telles que des serveurs, du stockage et des réseaux.

Keycloak : Solution open source de gestion des identités et des accès, utilisée pour l'authentification, la gestion des rôles, et l'authentification multifactorielle (MFA).

Kubernetes : Plateforme open source d'orchestration de conteneurs permettant l'automatisation du déploiement, de la gestion et de la mise à l'échelle des applications conteneurisées.

Manifests Kubernetes : Fichiers de configuration décrits en YAML ou JSON, utilisés pour définir les ressources *Kubernetes* telles que les pods, services, déploiements, et volumes. Ces *manifests* sont appliqués à un cluster *Kubernetes* pour créer ou mettre à jour les composants d'une application.

Namespace : Méthode de partitionnement logique dans *Kubernetes* permettant de diviser un cluster en espaces isolés. Chaque *namespace* contient ses propres ressources (pods, services, volumes) et permet de gérer plus facilement les environnements de développement, de staging et de production, ou de segmenter les applications et les équipes.

OCI (Open Container Initiative) : Standard ouvert défini par l'Open Container Initiative qui spécifie une structure commune pour les images de conteneurs et les runtimes. Le format OCI assure l'interopérabilité entre différents outils et plateformes de conteneurisation en fournissant des spécifications standardisées pour la création, le stockage et l'exécution des images de conteneurs.

Opérateur (Kubernetes) : Outil dans *Kubernetes* qui automatise la gestion des applications complexes en étendant les capacités de la plateforme. Un opérateur permet de gérer le cycle de vie d'une application (installation, mise à jour, réparation) via des CRDs, tout en automatisant des tâches d'administration répétitives.

OpenShift : Distribution commerciale de *Kubernetes* développée par *Red Hat*, offrant des fonctionnalités supplémentaires comme la gestion des clusters multi-clouds, la sécurité intégrée et des outils d'administration avancés.

PaaS (Platform as a Service) : Modèle de cloud computing qui fournit des environnements de développement et de déploiement complets, hébergés sur des infrastructures gérées.

Pipeline CI/CD : Suite d'étapes automatisées qui permettent d'intégrer et de déployer du code. Ces *pipelines* exécutent des processus comme les tests, la compilation, et le déploiement des applications dans un environnement donné. Ils font partie des pratiques DevOps visant à améliorer la qualité et la rapidité des livraisons de logiciels.

Plan de contrôle (Kubernetes) : Ensemble des contrôleurs *Kubernetes*

PVC (Persistent Volume Claim) : Objet *Kubernetes* utilisé par les applications pour demander un volume de stockage persistant. Un PVC permet de définir les caractéristiques du stockage (taille, type) dont une application a besoin, et de se lier à un volume persistant (PV) pour garantir que les données restent accessibles même en cas de redémarrage des pods.

Quay : *Registry* d'images de conteneurs open source maintenue par *Red Hat*, offrant des fonctionnalités telles que la gestion des permissions, l'analyse de vulnérabilités et le cache d'images.

Registry d'images (ou registre d'images) : service qui stocke, gère et distribue des images de conteneurs.

Runner GitLab : Composant utilisé dans *GitLab* CI/CD pour exécuter des *pipelines* de CI/CD. Un runner est un programme qui exécute les tâches spécifiées dans le fichier .gitlab-ci.yml, telles que les tests, la compilation ou le déploiement d'applications. Les runners peuvent être auto-hébergés ou partagés, et ils permettent d'automatiser les workflows de développement.

Sealed Secrets : Outil de gestion décentralisée des secrets pour *Kubernetes*, qui chiffre les secrets et les stocke dans *Git*, tout en permettant aux clusters *Kubernetes* de les déchiffrer lors du déploiement.

Serverless : Modèle d'exécution dans lequel les développeurs déploient des applications sans avoir à gérer l'infrastructure sous-jacente. Les ressources sont automatiquement allouées en fonction de la demande.

Terraform : Outil d'infrastructure as code permettant de provisionner et gérer des ressources cloud de manière déclarative, automatisée et versionnée.

Vault : Outil de gestion centralisée des secrets développé par *HashiCorp*, permettant de stocker, gérer et contrôler l'accès aux secrets (mots de passe, clés d'API, etc.) de manière sécurisée.

Worker Kubernetes : machine, physique ou virtuelle, qui exécute les applications contenues dans des pods. Ces machines sont les ressources de calcul sur lesquelles Kubernetes déploie et gère les conteneurs. Peut également être appelé nœud de travail ou *worker* node.