Hermes

CANTON DU VALAIS
KANTON WALLIS

Département des finances et des institutions
Service cantonal de l'informatique

Departement für Finanzen und Institutionen
Kantonale Dienststelle für Informatik

Rapport sur l'état du projet

IN 22 Plateforme applicative au label «Energy efficient service»

Classification	interne
État	approuvé pour utilisation
Nom du programme	
N° du projet	PRJ0020548 PRJ0020979
Chef de projet	Steve Favez
Version	1.0
Date	07.12.2024
Mandant	Bertrand Zermatten, berzer@admin.vs.ch, 027 606 22 52
Auteur/Auteurs	Steve Favez
Distribution	SCI – SAN – ANS

Suivi des modifications

Version	Date	Modification	Auteur
0.1	06.12.2024	Version initiale	Steve Favez
0.2	07.12.2024	Revue et mise en page	Bertrand Zermatten
0.3	11.12.2024	Version finale	Bertrand Zermatten

Tabelle 1: Contrôle des modifications

1 Objectifs du projet

La plupart des plateformes de conteneurisation sont déployées sur des environnements virtualisés. Cette couche de virtualisation fait majoritairement doublon avec la couche de conteneurisation.

La suppression de la couche de virtualisation permettrait d'optimiser l'utilisation des ressources matérielles nécessaires au fonctionnement des conteneurs qui constituent la plateforme applicative. Dans le but d'optimiser la consommation d'énergie, la plateforme devrait également être autonome pour démarrer et stopper les prestations afin de libérer de la mémoire, du CPU et des licences.

1.1 Objectifs

- Démontrer le fonctionnement et les gains de l'utilisation d'un système de conteneurs sur demande « on demand ».
- Démontrer le fonctionnement, les gains et l'exploitabilité de l'exécution de conteneurs directement sur des serveurs physiques « on BareMetal ».

2 Aperçu de l'état du projet

2.1 Aperçu de l'état global du projet

Domaine	État	Problème	Mesures
Appréciation globale	*Jaune*	*Expérience Knative convaincante.* *BareMetal nécessite changement de hardware.*	*Discussions en cours avec Dell pour le hardware BareMetal*
Délais	*Vert*		
Coûts	*Jaune*	*Frais de consulting moins élevés qu'annoncé*	*Facturation moindre à l'ANS*
Charge de personnel	*Vert*		
Résultats	*Jaune*	*Expérience BareMetal théorique pour l'instant.* *Expérience Knative en place et ouvre des perspectives d'économies intéressantes.*	*Discussions en cours avec Dell pourle hardware BareMetal*

Tabelle 2: Aperçu

2.2 Évaluation globale / Conclusion

Les différentes analyses et POC nous ont permis de mettre en place une stratégie pour le futur de notre infrastructure OpenShift. En effet, le passage du monde de la virtualisation au monde de la conteneurisation peut être comparée à la transition, dans les années 2000, des systèmes physiques aux systèmes virtualisés.

Pour tirer pleinement parti de tels systèmes, les applications doivent être conçues de telle manière que les conteneurs à déployer soient aussi légers que possible afin d'être rapides à démarrer et surtout capables de se mettre à l'échelle rapidement.

Cette transition prendra du temps, mais, aussi bien économiquement qu'énergétiquement, l'impact sera positif.

Il faut être conscient que certains types d'applications peuvent très bien fonctionner dans le monde conteneurisé (base de données par exemple) mais ne sont pas adaptés à une vision Knative « on demand ». On ne peut pas démarrer une base de données dans la seconde car les besoins en ressources sont trop élevés. Pour ce type de cas d'utilisation, il faudrait instaurer un couplage fort avec le hardware qui permettrait de stopper une partie des CPU en fonction de la charge.

Au niveau de l'État du Valais, nous allons commencer par mettre en place une nouvelle architecture OpenShift basée sur du BareMetal en capitalisant sur l'offre ACP de Dell. Sur plusieurs années, nous allons migrer toute notre infrastructure OpenShift pour devenir totalement indépendants de la couche de virtualisation. Nous étudierons ensuite les avantages d'exécuter nos VM sur l'environnement conteneurisé OpenShift.

3 Concepts et outils

3.1 Le modèle cloud natif, application à la demande

Nous nous sommes basés sur les principes d'application cloud native dont, l'un des plus importants est la capacité de démarrer / stopper les applications à la demande.

En effet, les prestations offertes à travers nos applications ne sont pas utilisées de manière constante et intense. Certaines sont utilisées quelques fois par année, d'autres la journée de manière sporadique, et la plupart des batchs ne fonctionnent que la nuit. Aujourd'hui, nous n'arrivons pas à stopper ces prestations à la demande, nous réservons et consommons de l'espace mémoire et de la capacité énergétique inutilement et nous payons des licences pour les supporter.

Ces applications sont développées, construites et testées sur les mêmes infrastructures. Ainsi, pour chacune des applications développées, nous avons plusieurs environnements (test, intégration et qualité) qui consomment également de l'énergie et des licences.

En nous appuyant sur le principe du cloud natif, nous pourrions non seulement stopper les applications moins utilisées en production mais également les environnements de test.

3.2 Prérequis du cloud natif et impactes.

Pour qu'une application soit considérée comme « cloud native », il faut impérativement qu'elle puisse démarrer à la demande, soit, avoir un temps de démarrage très rapide (en dessous de la seconde) lors de la première invocation. À la suite d'un temps d'inactivité prédéfini, se désactiver automatiquement.

Une plateforme permettant d'orchestrer le démarrage et l'arrêt des applications à la demande est nécessaire. Une plateforme de type Kubernetes avec le support de Knative répond à ces besoins.

Références

- Knative : https://Knative.dev/docs/
- OpenShift serverless : https://www.RedHat.com/en/technologies/cloud-computing/OpenShift/serverless

3.3 Infrastructure K8s à la demande.

Un cluster Kubernetes se base sur un ensemble de nœuds sur lesquelles sont orchestrées les différentes applications. Stopper les applications diminue l'utilisation mémoire et processeur ainsi que la taille de l'infrastructure requise, cependant, les serveurs physiques sous-jacents consomment encore de l'énergie inutilement. Il serait donc très intéressant de pouvoir stopper physiquement ceux-ci et de les redémarrer dès que la charge applicative le nécessite. De cette manière il serait possible d'économiser encore plus l'énergie électrique consommée par le serveur.

Le modèle de facturation de RedHat OpenShift se faisant au nombre de core, cela rendrait possible une diminution des coûts, notamment grâce aux applications qui ne sont utilisées que périodiquement pendant l'année.

3.3.1 Prérequis pour de l'infrastructure à la demande

Les systèmes K8s / OpenShift permettent également de démarrer et stopper des nœuds physiques à la demande :
https://docs.OpenShift.com/dedicated/osd_cluster_admin/osd_nodes/osd-nodes-about-autoscaling-nodes.html
mais nécessitent d'installer OpenShift sur du BareMetal.

4 Application à l'Etat du Valais

Pour ce POC, nous avions certaines contraintes, notre volonté n'étant pas de faire tabula rasa de l'existant, mais plutôt d'assurer une transition en douceur.

- Les équipes de développement maitrisent le monde Java, de nombreuses applications sont déployées avec cette technologie. Il n'est donc pas envisageable de tout réécrire.
- Actuellement, nos clusters OpenShift sont déployés sur une infrastructure VmWare déployée elle-même sur du hardware Dell. On ne peut remplacer cette infrastructure d'un coup et la remplacer par du BareMetal. La solution doit permettre une transition en douceur.
- Les équipes systèmes bénéficient d'outils très intéressants de Dell pour maintenir l'infrastructure VmWare - permettant notamment une gestion des ESX et du hardware centralisé par une solution Dell et assurant des mises à jour maîtrisées.

4.1 Challenge Java en cloud natif

L'une des contraintes majeures est de capitaliser sur la maîtrise de java des équipes internes. Malheureusement la plupart des frameworks java du marché offrent des performances moyennes au niveau consommation mémoire, utilisation du CPU et donc temps de démarrage. Il est habituel qu'une application java écrite, par exemple, avec SpringBoot, démarre en plus de 3 secondes et exige au minimum 512 Mb de mémoire. Il n'est pas acceptable que l'utilisateur final attende autant de temps lors du premier accès à son application. Il nous fallait donc une solution pour démarrer une application écrite en java beaucoup plus rapidement, donc, en moins d'une seconde.

4.2 Java natif avec Quarkus

Nous nous sommes donc basés sur le framework Quarkus - https://quarkus.io - qui nous a totalement convaincu.

En effet, après avoir écrit de nombreuses applications et micro-services, nous ne pouvons que confirmer les affirmations des utilisateurs de la solution.

- Le démarrage de l'application en mode natif est en dessous de la seconde.
- La taille mémoire est divisée par 2 voire 3 par rapport à une application SpringBoot.
- L'utilisation du processeur se révèle bien meilleure (beaucoup moins de garbage collection notamment).
- Très simple pour un développeur SpringBoot ou JEE de s'y former. Migration de l'existant relativement simple.
- Qualité de la documentation et de la communauté excellente.

En migrant à Quarkus nous avons les bénéfices suivants :

- Économie de mémoire, donc, plus d'applications déployables par nœud, donc moins de hardware et de licences OpenShift / VmWare à acquérir.
- Applications prêtes pour du cloud natif au vu des temps de démarrage.
- Capitalisation sur nos compétences internes et externes en Java.

4.3 Knative sur OpenShift.

Un ensemble important d'applications étant « Knative ready », nous avons donc décidé d'installer l'opérateur « OpenShift serverless »
https://www.RedHat.com/en/technologies/cloud-computing/OpenShift/serverless
et en valider le fonctionnement avec une applications Quarkus.

Nos processus de déploiement étant standardisés et s'appuyant sur Helm, il nous a été relativement aisé de changer le mode de déploiement pour nous adapter aux Objets relatifs à Knative.

Cependant, nous sommes tombés sur certaines limitations qui nous ont posé problèmes et ne nous ont pas permis de passer aussi facilement que prévu en mode Knative l'ensemble des applications.

4.3.1 On demand : promesse tenue

Nous avons été surpris en bien par la facilité de déploiement d'une application en mode Knative. Le processus est vraiment bien supporté, la transition se fait vraiment simplement du mode déploiement simple au mode serverless. Le Knative « serving » montre l'application « serverless », tant qu'aucune requête ou événement n'y accède, le pod reste inactif, pas de mémoire ni de CPU consommés. Dès la première requête, un pod est lancé, répond à la demande, puis, après un temps prédéfini, configurable, sans avoir reçu de requête, se désactive. De plus, si la charge augmente, à partir de certains seuils configurables, Knative augmente automatiquement le nombre de pod à disposition pour traiter les requêtes, puis, lorsque la charge diminue, il désactive les pods inutiles.

4.3.2 Non gestion des paths (paragraphe très technique)

Knative - serverless est basé sur une gateway basée sur Kourier qui fait le lien entre les requêtes http (Ingress controller) et le service Knatif - https://docs.OpenShift.com/serverless/1.34/Knative-serving/config-applications/configuring-kourier.html. Actuellement, Kourier souffre d'une limitation qui nous a posé problème : L'Ingress ne supporte pas les "path" après le host.

Explication à travers un exemple :

Imaginons que nous ayons une application "myapps", composée d'un frontend JavaScript et d'un backend Quarkus. Nous avons donc deux applications à déployer, myapps-svc et my-apps-front.

Voici les routes (ingress) à créer en utilisant des paths :

myapps-svc	service myapps-svc	ingress host & path myapps.vs.ch/svc
myapps-front	service myapps-front	ingress host & path myapps.vs.ch

Tabelle 3: routes ingress avec path

Si nous créons une Ingress Knative, il n'est pas possible de déclarer deux services serverless avec Kourier, le path n'est pas supporté. Il faut donc modifier les urls pour n'avoir que des hosts.

myapps-svc	service myapps-svc	ingress host & path svc-myapps.vs.ch
myapps-front	service myapps-front	ingress host & path myapps.vs.ch

Tabelle 4: routes ingress url uniquement

Comme nous avons un certain nombre d'applications basés sur le principe des "path" au niveau des urls, cela peut nous demander un certain travail.

En revanche, en utilisant le ServiceMesh, l'ingress n'utilise plus Kourier, mais Envoy, et, dans ce cas, les path sont supportés.

4.3.3 Gestion de quotas

Passer en mode Knative en voulant s'assurer de ses bénéfices requiert une gestion relativement fine des différents quotas sur le cluster OpenShift. En effet, faire du Knative requiert des composants applicatifs qui sont rapide au démarrage et sont scalables facilement et n'ont donc pas besoin d'une réservation de capacité inutile. Ceci n’est pas compatible avec des applications plus lourdes, qui ont besoin d'une forte capacité minimale, et qui doivent donc réserver de la mémoire et du CPU pour s'assurer de pouvoir fonctionner.

Il est donc important de gérer les nœuds du cluster sur lesquelles les applications doivent être lancées en mode Knative et les séparer d'autres applications qui ne peuvent pas être démarrées dans ce mode.

Il faut donc totalement changer le paradigme d'une application « lourde » pour laquelle on va réserver des ressources avec un quota important pour assurer les performances (CPU / RAM), pour passer à un mode sur lequel, dès que la charge est trop élevée sur une instance d'un pod, Knative crée automatiquement une nouvelle instance et la rend dès qu'il n'en a plus besoin.

4.3.4 Résultats du passage à un mode « Knative like »

Comme nous ne pouvions migrer l'ensemble de nos applications en mode Knative à cause du changement de path, nous avons effectué l'exercice différemment pour avoir des chiffres plausibles. Nous avons mis en place un batch tournant chaque nuit pour désactiver les pods qui ne sont pas utilisés dans les environnements de test. Ceci nous a permis de simuler, pour un certain nombre d'applications, les bénéfices d'une approche Knative.

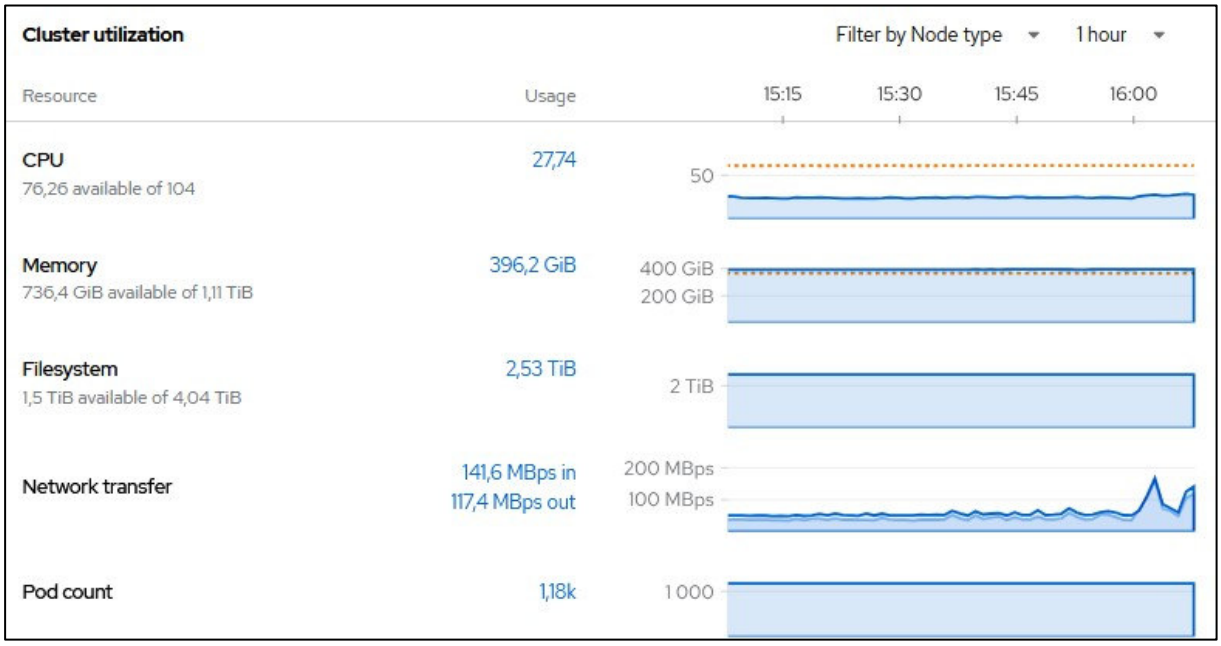

Figure 1: État du cluster avant de désactiver les différents Pods

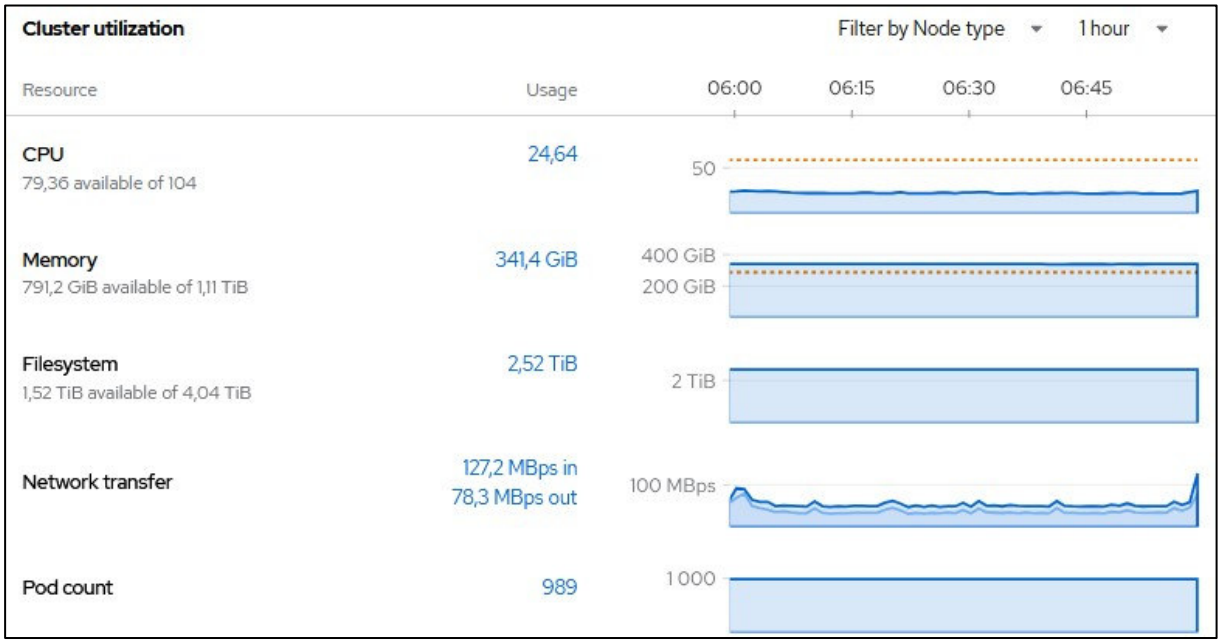

Figure 2: Etat du cluster après la désactivation des pods

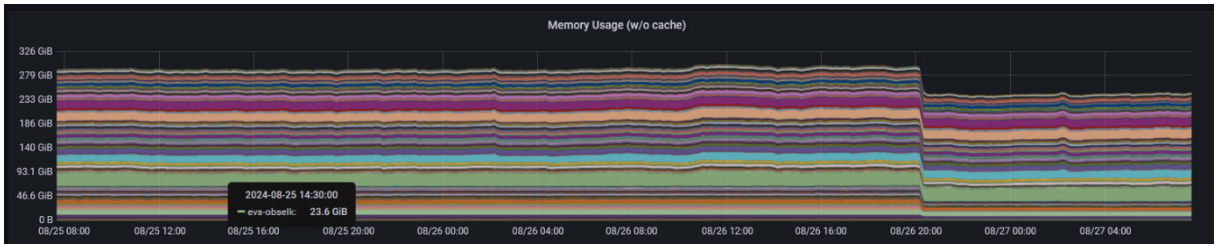

Figure 3 : Monitoring temps réel du cluster

En stoppant environ 200 Pods (16% environ), on constate un gain intéressant aussi bien au niveau de la RAM que du CPU.

	Gain brut	Utilisé avant	Utilisé après	Gain en %
Pods	191	1180	989	16%
CPU	3.1	27.74	24.64	11%
Mémoire	54.8	341.4	341.4	16%

Tabelle 5: Gains CPU et mémoire

5 OpenShift sur BareMetal

Actuellement, l'infrastructure OpenShift de l'État du Valais fonctionne sur une infrastructure VmWare.

L'avantage d'une telle approche était, à l'époque, de pouvoir facilement mettre en place une infrastructure Kubernetes sur des systèmes existants. Par ailleurs, l'installation d'OpenShift sur du BareMetal n'était pas aussi simple que des systèmes virtualisés.

En revanche, ceci a des impacts plus ou moins négatifs :

- Double licence : pour faire fonctionner une application, nous payons les licences OpenShift ainsi que les licences VmWare.
- Performances : en lieu et place de s'exécuter directement sur le BareMetal, le conteneur s'exécute sur une plateforme virtualisée, qui, malgré un grand nombre d'optimisation, consomme de la ressource pour la virtualisation de l'OS.
- Il n'est pas possible de stopper des VM à la demande en fonction de l'utilisation d'OpenShift. Donc, encore moins les serveurs physiques sous-jacents.

Le monde Kubernetes ayant beaucoup évolué ces dernières années, nous avons vu apparaître des offres sérieuses de certains fournisseurs hardware tel que Dell, permettant de gérer une plateforme OpenShift directement sur du BareMetal d'une manière aussi simple que VmWare. https://infohub.delltechnologies.com/ja-jp/l/dell-apex-cloud-platform-for-red-hat-OpenShift/what-is-the-apex-cloud-platform

En se basant sur une telle offre, nous pouvons envisager les avantages suivants :

- Une installation sur du BareMetal qui permet une gestion aussi aboutie qu'une offre ESX.
- Des applications qui utilisent les ressources systèmes au plus bas niveau, donc, des ressources systèmes utilisées uniquement pour faire tourner l'applicatif final.
- Une couche de licences en moins (plus de VMWare), donc une économie certaine.
- Une couche Kubernetes plus légère car plus proche du hardware, donc, la possibilité de faire tourner plus d'applicatifs sur la même infrastructure.
- La possibilité de mise à l'échelle les nœuds OpenShift de manière dynamique (stopper et démarrer le hardware en fonction de la charge).

5.1 Implication à l'État du Valais

Une telle transition, à notre niveau, n'est pas si simple. En effet, au vu de la taille de notre environnement OpenShift, les coûts pour acquérir le matériel ne sont pas négligeables.

5.1.1 Architecture OpenShift Pets vs Cattle.

L'architecture actuelle OpenShift de l'État du Valais a été faite avec une vision « Pets ». En effet, nous avons plusieurs clusters en fonction des environnements.

- Sandbox : permet un premier test des montées en version OpenShift.
- Test : contient l'ensemble des environnements de test.
- Production : contient l'ensemble des nœuds de production.

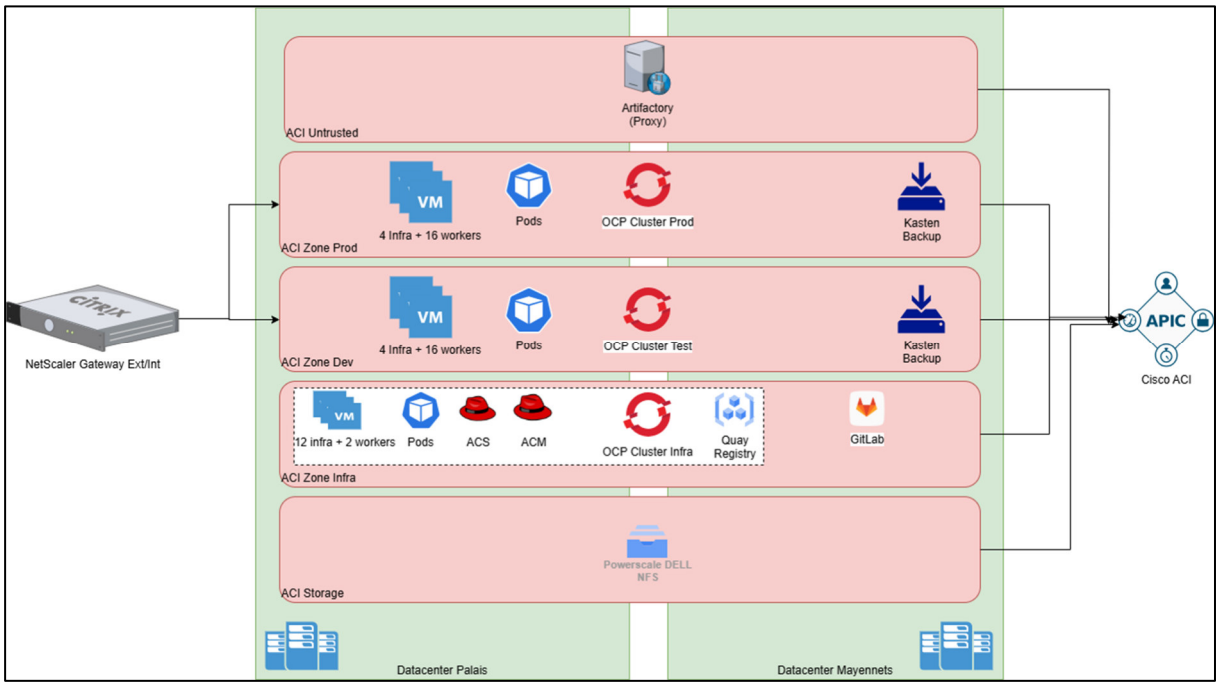

Figure 4: Architecture OpenShift actuelle "Pet"

Une telle architecture est facilement implémentable dans le monde VmWare, mais, à notre taille, difficilement implémentable en mode BareMetal. En effet, comme pour chaque cluster, il faut un minimum de 3 nœuds master et un nombre de nœuds « worker » sur lesquelles sont exécutées les applications, cela fait beaucoup de hardware, d'autant plus si les systèmes sont redondants (sur plusieurs centres de calculs). Ceci nécessiterait au minimum 7 serveurs physiques par cluster, ce qui est, économiquement, difficile à justifier.

En passant en mode BareMetal, il faut passer à une architecture avec une vision des clusters de type « Cattle ». Le cluster OpenShift peut être créé et supprimé à la demande. Les applications peuvent être déployées sur plusieurs cluster et un ServiceMesh permet de dynamiquement envoyer les requêtes au bon cluster. Le cluster devient donc un « Cattle » de nœuds, qui sont labelisés en fonction de leur utilité (prod, test) pour gérer les déploiements ainsi que les autorisations d'accès. Un tel système permet de faire une transition en douceur au niveau des équipements, car il permet de répartir des nœuds entre des worker nodes BareMetal et virtualisés.

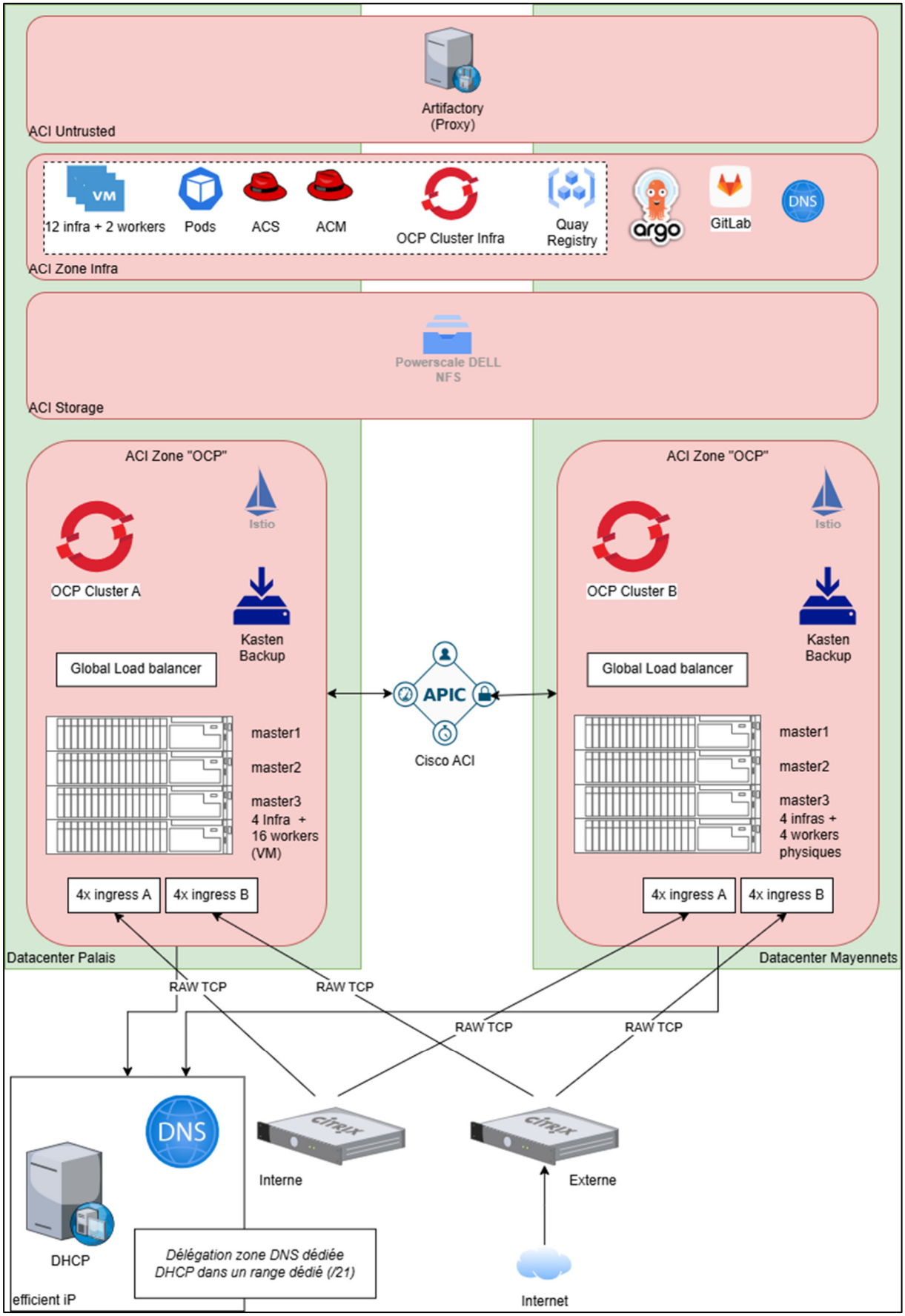

Figure 5: Architecture OpenShift visée "Cattle"

6 Charge et coûts

6.1 Coûts

Coûts (CHF)

Rôle	Montant TTC	Tâches
Delivery manager	*6'000.-*	*Suivi du projet et rédaction du rapport*
Expertise	*5'500.-*	*Workshops*
TAM RedHat	*10'000.-*	*Workshops, conseils et suivi dossier interne RedHat*

Tabelle 6: Coûts externes

6.2 Charge de travail

Jours-personnes

Phase	j/h
Interne SCI	*60*

Tabelle 7: Charge de personnel

Abréviations et glossaire

Abréviation / Terme technique	Explication
BareMetal	Serveur physique sans couche de virtualisation.
Batchs	Traitements groupés exécutés sans interaction utilisateur.
Conteneurisation	Isolation d'applications et de leurs dépendances dans des conteneurs légers.
Core	Unité de traitement d'un processeur qui exécute les instructions.
CPU	Processeur principal d'un ordinateur, responsable du calcul et de l'exécution des instructions.
Envoy	Proxy moderne pour gérer les communications entre microservices.
ESX	Hyperviseur VMware pour virtualiser les serveurs physiques.
Garbage collection	Processus automatique de gestion et libération de la mémoire inutilisée dans les programmes.
Helm	Gestionnaire de paquets pour Kubernetes.
Ingress controller	Contrôleur qui gère l'accès externe aux services Kubernetes.
K8s (Kubernetes)	Système de gestion d'orchestration de conteneurs.
Knative	Plateforme Kubernetes pour déployer des applications serverless.
Knative serverless	Fonctionnalité de Knative basée sur Kourier pour gérer les applications sans serveur.
Knative serving	Module de Knative pour exécuter et scaler les workloads serverless.
Kourier	Gateway légère pour gérer le trafic Knative serverless.
On demand	Ressources ou services fournis uniquement lorsque nécessaires.
OpenShift	Plateforme Kubernetes gérée par Red Hat avec outils supplémentaires.
OpenShift serverless	Fonctionnalité d'OpenShift pour exécuter des applications serverless.
Pets vs Cattle	Métaphore décrivant les approches IT : "Pets" pour les serveurs uniques, "Cattle" pour les génériques.
POC	*Proof of Concept*, démonstration technique pour valider une idée.
POD	Plus petite unité de déploiement dans Kubernetes, regroupant un ou plusieurs conteneurs.
Scalable	Capacité d'un système à évoluer en taille ou capacité selon les besoins.
Serverless	Architecture où les développeurs n'ont pas à gérer les serveurs sous-jacents.
ServiceMesh	Infrastructure pour gérer les communications entre microservices.
Virtualisation	Technique pour créer plusieurs machines virtuelles sur un serveur physique.
VM	Machine virtuelle, un environnement informatique isolé créé sur un serveur physique.
VmWare	Entreprise spécialisée dans la virtualisation et les logiciels connexes.
Worker nodes	Nœuds dans un cluster Kubernetes où les conteneurs sont exécutés.
BareMetal	Serveur physique sans couche de virtualisation.

Tabelle 8: Abréviations et glossaire

Table des matières

Table des tableaux

Table des figures