OpenKeel

OpenStack réécrit en Go, natif Kubernetes, iso-fonctionnel avec les composants d'origine

Plateforme Cloud et Cloud Native moderne et Souveraine, disposant d'un large écosystème



Série 2026.2 · novembre 2026

Première étape : spécifier, générer, tester

151 projets amont

Dépôts OpenStack récupérés et analysés

36 reverse-specs

Spécifications précises des composants

27 projets Go

En commençant par Placement et Keystone

Skill et tests e2e

Spécification enrichie, e2e intégrés

Principes retenus

API OpenStack

100 % compatible avec les API Python

MariaDB

100 % compatible avec le modèle de données

RabbitMQ

100 % compatible avec les messages

Tests e2e

Rejouer les jeux de tests d'OpenStack

Briques existantes

CSI, NFS-Ganesha, OVN, KubeVirt

Chronologie du projet

1 · Spécifier

1. Identifier et récupérer les projets OpenStack
2. Créer la skill reverse-spec
3. reverse-spec sur 36 projets OpenStack
4. Recherche d'un nom

2 · Générer le cœur

5. Créer la skill generate-go-from-spec
6. Créer la skill generate-openkeel-diagram
7. Go : Placement, puis Keystone
8. Go : Glance/CSI, Cinder/CSI, Neutron/OVN, Nova/KubeVirt
9. Go : Skyline API et console, un seul module

3 · Déployer et tester

10. Manifests d'installation des modules
11. Charts Helm d'installation
12. CLI okadm : lab ou production
13. Déployer un cluster OpenKeel de lab
14. Tester Skyline et les premiers modules
15. Créer la skill generate-openkeel-powerpoint

4 · Étendre

16. Go : LBaaS, FWaaS, VPNaaS, OVN-BGP-Agent
17. Go : Barbican, Manila/CSI, Zun/Kata
18. Swift au travers de Rook-Ceph
19. Ironic conservé en Python
20. Créer la skill generate-static-web-site
21. Publier les images sur un registry

5 · SecNumCloud

22. Plan de bataille SecNumCloud
23. TOTP dans Keystone
24. TLS entre les composants

6 · FinOps et GreenOps

25. reverse-spec sur CloudKitty (usages)
26. Go : CloudKitty
27. Module openkeel-pricing
28. Panneau FinOps/GreenOps dans Skyline
29. Menu « Project FinOps/GreenOps », tableau de bord

OpenKeel en une page

18

composants spécifiés puis réécrits en Go

37

reverse-specs produites sur les dépôts amont

3

skills Claude chaînées : spec, diagramme, code

Les principes

- Même API REST, même base de données, mêmes messages RabbitMQ que l'original
- Kubernetes fait le travail lourd : KubeVirt, CSI, OVN, Rook-Ceph, Medik8s
- Binaires Go statiques, images distroless sans shell
- Déploiement plug and play avec okadm, en lab (kind) ou en production
- Chaque composant validé contre son original (replay, Tempest)

Sommaire

01 Contexte : OpenStack

02 Trouver un nom

03 Une chaîne de skills Claude

04 Ne pas réécrire ce qui est complexe

05 18 composants spécifiés et générés

06 Améliorations

07 okadm : déployer en plug and play

08 Iso-fonctionnalité et tests

09 Observabilité

10 Sécurité et SecNumCloud

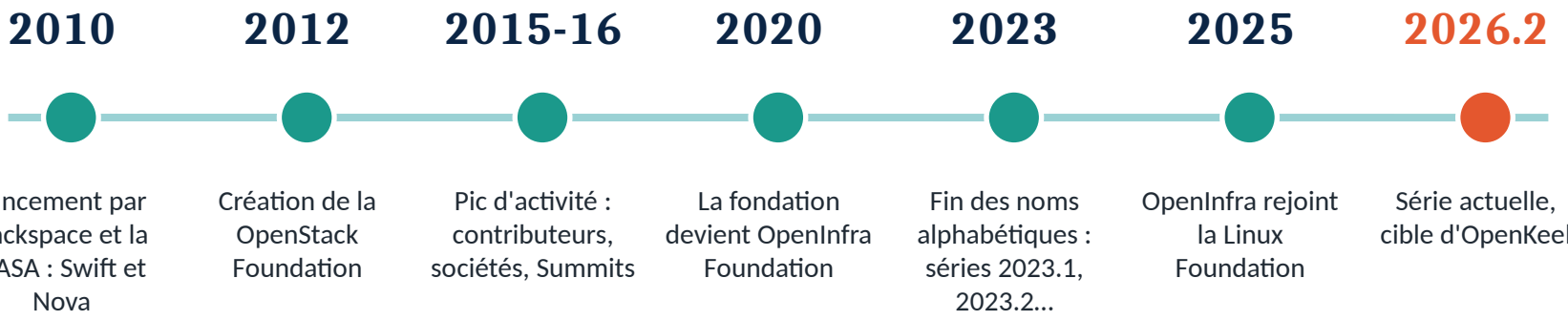
01

Contexte : OpenStack

Quinze ans d'histoire, une grande communauté, une base de code lourde



OpenStack en quelques dates



Une version tous les six mois depuis 2010 ; chaque service garde la compatibilité de son API d'une version à l'autre.

Une communauté ouverte, qui s'est resserrée



Gouvernance

Les « Four Opens » : source, design, développement et communauté ouverts. Un Technical Committee, un PTL par projet.



Outillage

OpenDev : revue Gerrit, CI Zuul, devstack et Tempest pour tester chaque changement.



Utilisateurs

Opérateurs télécoms, clouds publics et privés, recherche : de très gros parcs en production.



Évolution

Après le pic de 2015-2016, des éditeurs se retirent ; moins de contributeurs pour autant de code.

Le poids du code legacy

Python et eventlet

Green threads monkey-patchés, sortie d'eventlet encore en cours ; GIL et performances limitées

Empilement de bibliothèques

Dizaines de libs oslo, clients et drivers à maintenir ensemble

Configuration foisonnante

Des centaines d'options par service, beaucoup inutilisées

Déploiement complexe

Kolla, OpenStack-Ansible, TripleO (arrêté) : l'outillage pèse autant que le cloud

Upgrades douloureux

Migrations de schéma, RPC versionnés, ordre de mise à jour strict

Agents sur les hôtes

libvirt, agents OVS, iSCSI : un plan de données à réinventer à côté de Kubernetes

02

Trouver un nom

Libre, sans la marque OpenStack, et marin comme Kubernetes



OpenKeel : la quille du navire

Critères

- Connotation marine, dans la lignée de Kubernetes (le timonier), Helm, Harbor
- Aucune collision logicielle connue
- Organisation GitHub et domaines disponibles
- Pas de « OpenStack » dans le nom (marque OpenInfra)

Noms écartés

OpenStack-*
marque OpenInfra

ROS
Robot Operating System

Restack
git-restack, restack.io

Gopherstack, GoStack
déjà pris

Keelstone
trop proche de Keystone

OpenKeel

- Keel : la quille, colonne vertébrale du navire
- Choisi le 28 octobre 2026
- Organisation GitHub openkeel libre à cette date
- openkeel.com et .org pris ; .io et .dev sans DNS
- Dépôts de marque (INPI, EUIPO, USPTO) encore à vérifier

Un logo : le k devient le mât



0 · Actuel

Le voilier de départ



1 · K-mât

Fût = mât, jambe = chute de la voile



2 · Traversant

La jambe du K devient la quille

Retenu



3 · k minuscule

Hampe = mât, bras = bôme relevée



4 · En réserve

Un K découpé dans la voile



5 · Pastille

Le K-mât sur un rond marine



6 · Sigle

Un K en trois formes pleines



Retenu : le k minuscule

La hampe du k est le mât, son bras la bôme sous la voile citron, sa jambe un hauban ; coque orange et quille sont gardées. Le logo est repris dans les présentations, le site web et le configurator.

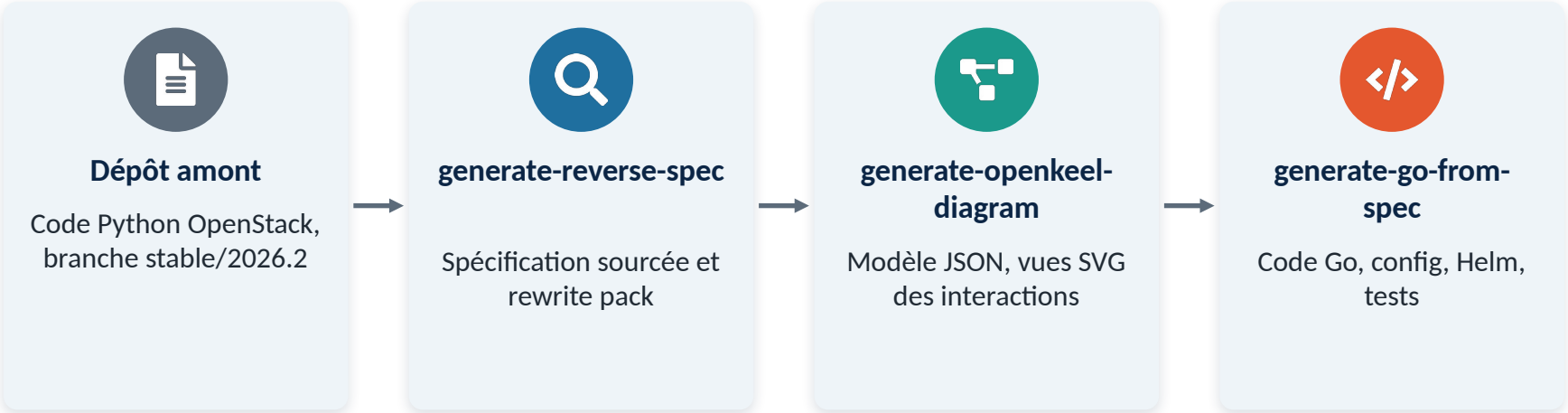
03

Une chaîne de skills Claude

De la lecture du code Python amont au code Go, aux charts et aux tests



Trois skills, une chaîne de production



Travail itératif : chaque écart trouvé en test (bug, non-conformité, piège Kubernetes) est corrigé dans le composant puis réinjecté dans la skill, au bénéfice des composants suivants.

Skill reverse-spec : rétro-spécifier l'existant

Issue de **analyse-projet-oss** (rétro-documentation en français), devenue une spécification en anglais, liée à une branche portée par chaque nom de fichier, avec un rewrite pack pour la réimplémentation.



La spécification

- Endpoints REST : description technique et fonctionnelle
- Traitements hors API : consommateurs RabbitMQ, polling SQL, événements OVSDB, tâches périodiques
- Codes d'erreur et situations qui les déclenchent
- Interactions : SQL, AMQP, Redis, REST sortant, avec trames et diagrammes de séquence
- Historique de la branche en change specs

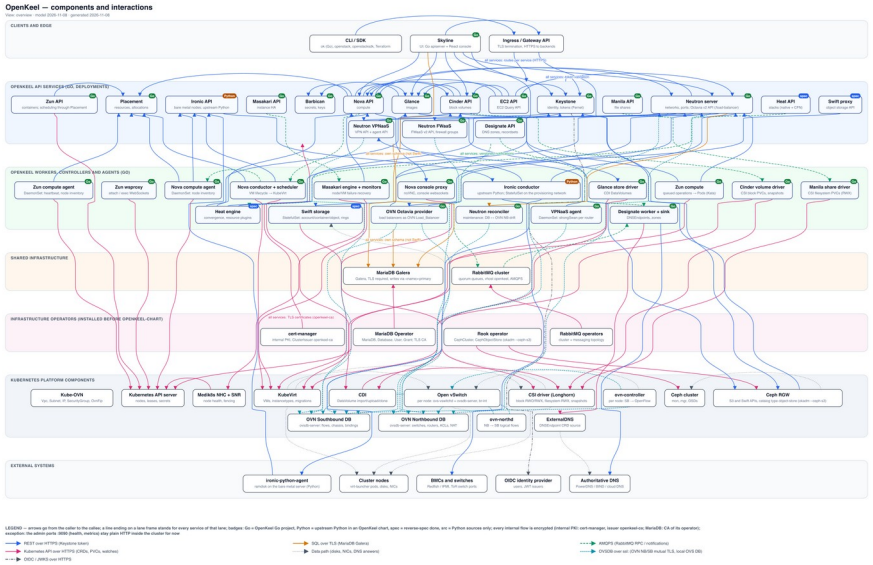


Le rewrite pack

- OpenAPI et schémas JSON
- DDL de la base de données
- Policies et options de configuration
- Données de conformité (requêtes et réponses attendues)
- Algorithmes clés et attentes des clients

Skill diagramme : un modèle, toutes les vues

- Un seul modèle JSON des services, de leurs interactions et du placement sur les nœuds
- Vues SVG : ensemble, plateforme, par service, par protocole (REST, SQL, RabbitMQ, OVSDB, OIDC)
- Placement : cluster Talos de production et lab kind
- Tableau Markdown des interactions et badges d'avancement (Go, spec, Python)
- Régénéré à chaque décision d'architecture



Vue « overview » générée par la skill

Skill go-from-spec : du rewrite pack au Go

Comportement identique

API, codes HTTP, corps d'erreur, microversions, policies

Config YAML épurée

Les options inutilisées de l'original disparaissent

MariaDB Galera

Même schéma que l'original, écritures sur le primaire

TLS partout

MariaDB vérifiée, AMQPS, HTTPS, OVSDB ssl:

RabbitMQ

RPC et notifications au format oslo.messaging

Logs structurés

JSON en anglais (slog), audit CADF

Checklist sécurité

Distroless, sans shell, UID 65532

Tests

Unitaires et e2e en replay contre l'original

Charts Helm

Un chart par service, chart parapluie

04

Ne pas réécrire ce qui est complexe

Le plan de données est délégué à des briques Kubernetes éprouvées



Chaque service s'appuie sur Kubernetes



Nova → KubeVirt

Les instances sont des VirtualMachines ; CDI importe les images



Cinder → CSI bloc

Volumes = PVC (Longhorn), snapshots CSI



Manila → NFS-Ganesha

Passerelle NFS créée à la volée dans le réseau du tenant, sur volume CSI bloc



Neutron → OVN

Kube-OVN, réseau des VMs isolé du réseau Kube (Multus)



Swift / S3 → Ceph

Rook-Ceph et sa passerelle RGW



Designate → ExternalDNS

Zones et enregistrements publiés par ExternalDNS



Masakari → Medik8s

Node Healthcheck et Self Node Remediation



Ironic reste en Python

Trop de configurations matérielles à tester

CSI Pure Storage : les composants



Sidecars CSI

provisioner, attacher, snapshotter, resizer : traduisent PVC et VolumeSnapshot en appels CSI



Contrôleur pure-csi

Volumes, hôtes, connexions, snapshots ; seul détenteur du token de la baie



FlashArray

API REST 2.x (client maison, défaut) ou REST 1.x via le SDK pugo



Plugin nœud

DaemonSet : login iSCSI sur chaque portail, multipath, contrôle du WWID, montage

openkeel-csi-pure
2026.2.0, Go 1.26

csi-provisioner
v6.4.0

csi-attacher
v4.13.0

csi-snapshotter
v8.7.0

csi-resizer
v2.3.0

node-driver-registrar
v2.14.0

Spécification CSI
v1.13.0

csi-lib-iscsi
attachement iSCSI

CSI Pure Storage : les capacités

Provisionnement

Volumes k8s-<PV>, clones et restaurations par copie instantanée, puis agrandissement à la taille demandée

Snapshots

VolumeSnapshot et VolumeGroupSnapshot : protection group, snapshot cohérent en ordre d'écriture

Extension à chaud

Volume agrandi sur la baie, rescan des chemins, multipath, puis ext4 ou xfs

Modes d'accès

RWO, RWOP, ROX en filesystem ; RWX en bloc brut pour la live migration KubeVirt

Robustesse

Idempotence par noms déterministes, WWID vérifié avant tout formatage, nettoyage complet au détachement

Sécurité

Token seulement au contrôleur, non-root ; TLS vérifié ; utilisateur dédié au rôle storage_admin

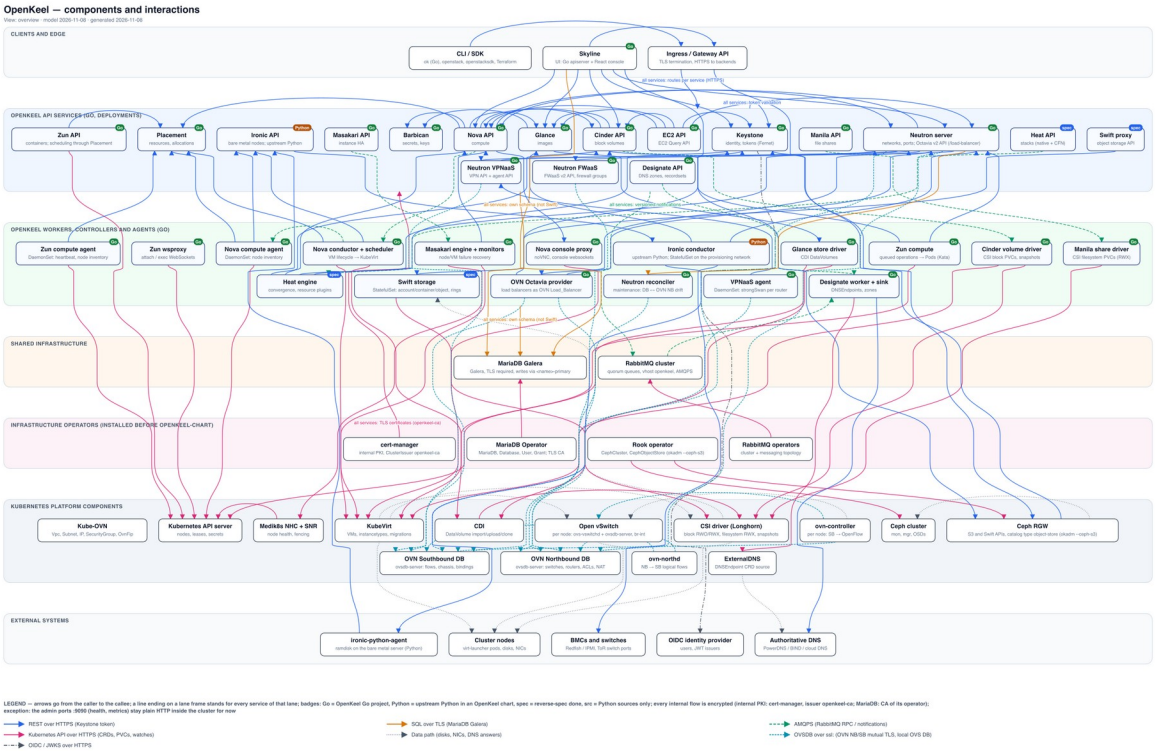
CSI Pure Storage : compatibilité

Élément	Compatibilité
Baie	Pure Storage FlashArray en iSCSI (TCP 3260), une baie par instance du driver
API	REST 2.x, Purity//FA 6.0 et plus (TLS vérifié) ; REST 1.x via pugo, dépréciée par Pure
Nœuds	Linux avec open-iscsi et multipathd (user_friendly_names no), un IQN unique par nœud
Talos	Extension iscsi-tools ; le multipath n'est pas encore validé
Kubernetes	CSI 1.13 ; CRD de snapshots et de group snapshots, snapshot-controller
OpenKeel	Backend Cinder csi-block sur la StorageClass pure-iscsi ; RWX bloc pour KubeVirt



Prototype : pas encore exécuté sur une vraie FlashArray (csi-sanity : 79 specs en mémoire). Pas de topologie ni d'ActiveCluster, portails IPv4, ni CHAP ni QoS.

Vue d'ensemble : services, données et plateforme



05

18 composants spécifiés et générés

Reverse-spec, puis code Go, pour chaque service



Calcul, stockage, identité et services



Placement

Inventaires et allocations ; premier composant réécrit



Keystone

Identité, jetons, SQL, OIDC, JWT, app credentials



Nova

Instances sur KubeVirt ; conductor, scheduler, agent



Cinder

Volumes bloc sur CSI, snapshots CSI



Cinder Backup

Sauvegarde des volumes, API et messages de cinder-backup



Manila

Partages NFS multi-tenant par passerelles Ganesha



Barbican

Secrets, conteneurs et clés



Heat

CLI seule : commandes stack de ok, pas de service Go



Designate

Zones et recordsets via ExternalDNS



Masakari

HA des instances, santé des nœuds par Medik8s

Réseau : Neutron sur OVN et ses extensions

Neutron / OVN

- neutron-api en Go : réseaux, sous-réseaux, ports, routeurs, groupes de sécurité
- Écriture dans la base OVN Northbound
- Reconciler : corrige les écarts entre la base SQL et OVN
- Réseau des VMs isolé du réseau des pods (Multus)

OVN-Octavia-Provider

API Octavia v2 servie par neutron-api ; load balancers OVN, actifs par défaut

Neutron-FWaaS

Firewall groups, policies et rules (FWaaS v2)

Neutron-VPNaaS

VPN IPsec site à site : IKE, IPsec, connexions

OVN-BGP-Agent

Annonce en BGP des IP de VMs et des floating IPs

Neutron, OVN et OVS : qui fait quoi

Neutron (Go)	API, base MariaDB, IPAM, policies ; écrit réseaux, ports, routeurs, groupes de sécurité et IP flottantes dans OVN ; un reconciler corrige les écarts
OVN Northbound	Modèle logique : switches, routeurs, ports, ACL et Port_Group, NAT (SNAT, IP flottantes), DHCP, load balancers Octavia
ovn-northd + Southbound	Compile ce modèle en flux logiques ; lie chaque port à un nœud (chassis) ; planifie les passerelles HA
ovn-controller	Sur chaque nœud : programme br-int en OpenFlow, tunnels Geneve, répond localement en DHCP, ARP et RA
Open vSwitch	ovsdb-server et ovs-vswitchd : bridge br-int, datapath noyau ou userspace, commute les paquets des VMs et conteneurs

IPv6 dans Neutron

Sous-réseaux IPv6

Modes slaac, dhcpv6-stateful et dhcpv6-stateless ; mêmes validations et erreurs que Python

Annonces de routeur

ipv6_ra_configs sur les ports de routeur OVN : adressage, MTU, route du metadata

DHCPv6 sans agent

Options DHCPv6 écrites dans OVN (serveur, DNS, mode stateless) ; adresses SLAAC calculées en EUI-64

Groupes de sécurité

ACL ip6 et icmp6, groupes distants en IPv6, règles de sortie IPv6 créées par défaut

Routage

Route par défaut ::/0 vers la passerelle externe ; SNAT et IP flottantes restent en IPv4

Isolation des VMs

Forwarding IPv6 coupé dans le pod, IPv6 désactivé sur le veth de l'hôte, link-local du bridge invisible

Pas encore vérifié : sous-réseaux IPv6 de tenant sur le vrai plan de données du lab, tests Octavia IPv6 de Tempest. La prefix delegation, dépréciée en amont, n'est pas reprise.

CaaS : Zun sur Kubernetes et Kata

Clients

API Zun v1 identique : zunclient, openstack appcontainer, Heat, Skyline

zun-api

Ordonnancement par Placement, file de tâches en base, sans RabbitMQ

zun-compute

Ports Neutron, volumes Cinder, puis un Pod par conteneur

Nœud Kata

RuntimeClass kata : chaque conteneur dans sa microVM Firecracker

Isolation forte

Un noyau par conteneur ; runc réservé aux admins, privileged refusé ; seul réseau : le port Neutron

Inventaire partagé

Même fournisseur Placement que les VMs KubeVirt de Nova sur le nœud

Écarts assumés

Pas d'agent privilégié : pause et commit en 409, stop supprime le Pod ; données dans Cinder

Skyline et la CLI : les points d'entrée

Avant

- skyline-apiserver en Python
- skyline-console servie par nginx
- Deux processus, deux images

Après : openkeel-skyline

- Un seul module Go
- Serveur web de la console et API REST
- Une image distroless, un chart



La console React reste celle de Skyline : les utilisateurs retrouvent la même interface, servie par un binaire unique.



CLI openstack → ok

Les commandes de python-openstackclient et de ses plugins, réécrites en un binaire Go statique.

Skyline : de la console à Kubernetes

Skyline Console	**skyline-console** Console React amont, inchangée, servie à la racine par le binaire Go							
Skyline API	**openkeel-skyline** API v1, session JWT, policies ; proxy vers le catalogue Keystone (port 9998)				**MariaDB Galera** revoked_token, settings			
Modules OpenKeel	**Keystone**	**Nova**	**Neutron**	**Cinder**	**Glance**	**Placement**	**Manila**	**Zun**
Kubernetes	**KubeVirt** Nova	**CDI** Glance	**CSI Longhorn** Cinder, Glance	**OVN, Multus** Neutron	**NFS-Ganesha** Manila	**Kata** Zun		

Observabilité

- /metrics sur :9090 de chaque pod, PodMonitor
- Requêtes Prometheus pour les graphes de la console
- Logs JSON collectés par Alloy vers Loki
- Audit CADF actif par défaut
- /healthz et /readyz sur :9090
- Exporters KubeVirt, Longhorn, MariaDB, RabbitMQ
- Grafana et Alertmanager

06

Améliorations

Au-delà de l'iso-fonctionnel : outillage, sécurité et confort



Go : le langage du Cloud Native

Langage du Cloud Native

Kubernetes, etcd, containerd, Helm, Prometheus, KubeVirt : le même langage que la plateforme

Écosystème intégré

client-go, SDK Helm, OVSDB, Prometheus, OIDC : des bibliothèques natives, sans pont

Typé et compilé

Typage fort, erreurs détectées à la compilation, code plus rapide que Python

Build rapide

Un binaire statique par service, compilation rapide, multi-architecture

Conteneurs minimaux

Un exécutable, rootless, sans shell ni sources, sur distroless et non sur Ubuntu

Sobre à l'exécution

Démarrage rapide et empreinte mémoire réduite : pas d'interpréteur ni d'eventlet

Moins de code, autant de garanties

993 k lignes Python des 18 modules d'origine

293 k lignes Go des modules OpenKeel

−71 % de code, tests exclus

Détail par module sur la slide suivante

Briques Core externes

OVN, KubeVirt, CSI, Medik8s pour Masakari, ExternalDNS : le plan de données n'est pas réécrit

Validé par les tests d'OpenStack

Tests unitaires et e2e amont rejoués, Tempest : API REST, schéma MariaDB, messages RabbitMQ identiques

Images légères

21 Mo pour Placement et Keystone, 32 Mo pour Neutron

Lignes de code avant et après réécriture

Module	Python	Go	Écart
Placement	10 287	10 215	−1 %
Keystone	50 584	16 321	−68 %
Nova	150 846	20 493	−86 %
Glance	31 279	14 331	−54 %
Cinder	255 934	17 359	−93 %
Neutron	127 623	25 906	−80 %
Neutron-FWaaS	8 085	9 602	+19 %
Neutron-VPNaaS	6 449	11 149	+73 %
OVN-Octavia-Provider	5 778	9 042	+56 %

Module	Python	Go	Écart
OVN-BGP-Agent	8 448	5 074	−40 %
Manila	135 162	18 244	−87 %
Barbican	18 688	12 810	−31 %
Designate	23 671	15 326	−35 %
Masakari	10 835	9 306	−14 %
Skyline	18 723	8 381	−55 %
Zun	30 938	13 906	−55 %
EC2 API	11 570	16 231	+40 %
CLI openstack → ok	88 122	59 156	−33 %



Total : 993 022 lignes Python → 292 852 lignes Go (−71 %). Les pilotes et agents remplacés par Kubernetes disparaissent ; les petits modules autrefois greffés sur Neutron ou Nova grossissent, car ils deviennent autonomes.

Configuration : d'oslo.config au YAML

	OpenStack (Python)	OpenKeel (Go)
Fichiers	INI oslo.config, plus api-paste.ini, rootwrap.conf et ses filtres, logging.conf, policy	Un config.yaml par processus, monté depuis une ConfigMap
Secrets	Mots de passe dans connection et transport_url	Champs *_file lus dans des Secrets montés, jamais dans le YAML
Processus	bind_host, workers, log_file, use_syslog, debug	Pris en charge par Kubernetes ; un seul default.log_level
Pilotes	Options de tous les pilotes livrés (libvirt, VMware, baies, ML2…)	Seules les options des briques retenues (KubeVirt, CSI, OVN)
Contrôle	Vérification à part, avec l'outil oslo-config-validator	Décodage strict : clé inconnue ou valeur hors bornes, refus de démarrer
Changement	Édition des fichiers sur chaque hôte, redémarrage	Nouveau hash de ConfigMap, rolling restart ; niveau de log par SIGHUP

Mêmes noms : groupes, options et valeurs par défaut de l'original ; OPENKEEL_<GROUPE>_<OPTION> surcharge une valeur propre à un pod.

Moins d'options : ne garder que l'utile

Module	Gardées	Retirées
Placement	7	101
Keystone	100	313
Nova	122	1 015
Glance	66	472
Cinder	49	1 411
Neutron	82	443
Manila	82	905
Barbican	72	200
Designate	84	300
Masakari	35	235
Zun	63	300
EC2 API	16	151

778

options gardées sur 6 624, dans 12 modules

88 %

des options oslo retirées



Retirées

Dépréciées, pilotes non livrés, eventlet et verrous fichiers, options du processus reprises par Kubernetes, chemins de code non implémentés



Gardées

Tout ce qui change l'API, les données ou les limites : quotas, ratios d'allocation, durée des jetons, enforce_scope

ok et okadm : deux commandes, deux métiers

ok : le client OpenStack

- Commandes de python-openstackclient 10.3.1 et de ses plugins
- Mêmes arguments, formats de sortie, messages d'erreur et codes retour
- Lit clouds.yaml, OS_* et --os-* comme l'original

```
export OS_CLOUD=openkeel
ok server list
ok loadbalancer create ...
```

okadm : le cycle de vie

- plan, install, upgrade, status, uninstall
- client-go et SDK Helm intégrés : ni kubectl ni helm requis
- Manifeste de release embarqué, upgrade composant par composant

```
okadm plan --mode lab
okadm install --mode lab \
  --create-kind --ingress
```

Sécurité et confort d'usage



Skyline-OIDC

Connexion à la console par un fournisseur d'identité OIDC : fédération, SSO, jetons validés par JWKS.



Secrets internes rotatifs

Mots de passe de base, comptes RabbitMQ et clés de service gérés comme des composants rotatifs, renouvelables sans intervention manuelle.



Credentials EC2 dans Skyline

Création et gestion des identifiants EC2 (access / secret) depuis la console, pour accéder au stockage S3.

07

okadm : déployer en plug and play

Un cluster OpenKeel de production ou de lab en une commande



okadm, Kolla Ansible, OpenStack-Helm

	Kolla Ansible	OpenStack-Helm	okadm
Outil	Python et Ansible : venv, install-deps, collection Galaxy	Charts Helm, plugin helm osh ; kubectl et helm requis	Un binaire Go : client-go et SDK Helm intégrés, charts embarqués
Cible	Hôtes Linux par SSH, inventaire all-in-one ou multinode	Cluster Kubernetes existant ; sa création est hors périmètre	Cluster Kubernetes existant, ou cluster kind créé en lab
Hôtes	bootstrap-servers : paquets, Docker, pare-feu, SELinux, AppArmor, NTP	Rôle Ansible deploy-env pour un lab, non idempotent	Rien à préparer : okadm plan vérifie et donne la commande
Plan de données	libvirt et agents OVS dans des conteneurs sur les hôtes	DaemonSets libvirt et openvswitch, Ceph à déployer	KubeVirt, OVN et CSI : des briques Kubernetes



Un seul binaire : ni venv, ni inventaire SSH, ni dépôt de charts : tout est embarqué, sauf les images.

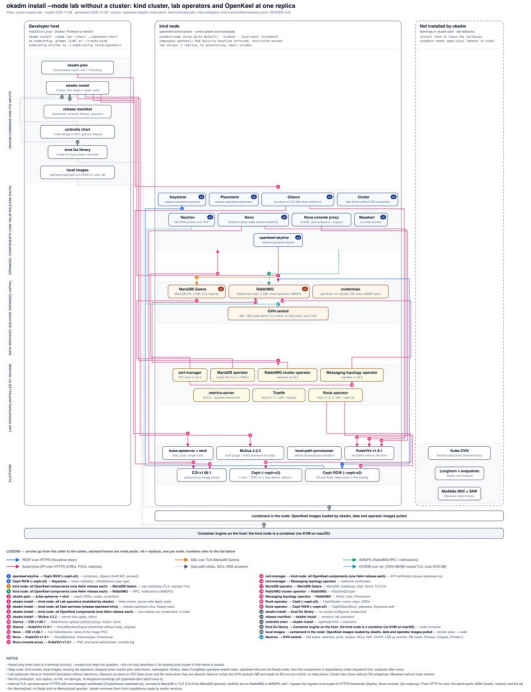
Configurer et opérer : le périmètre

	Kolla Ansible	OpenStack-Helm	okadm
Données	MariaDB, RabbitMQ, Memcached, HAProxy en conteneurs Kolla	Une release Helm par brique : mariadb, rabbitmq, memcached	Opérateurs ; instances, bases, comptes et vhosts créés par okadm
Configuration	globals.yml : 326 options ; inventaire Ansible	Values overrides par chart et par feature set	--mode lab ou production, --domain, -f values
Secrets	passwords.yml : 106 secrets générés par kolla-genpwd	Dans les values des charts	Générés dans des Secrets Kubernetes, par composant
Cycle de vie	26 sous-commandes : deploy, reconfigure, upgrade, mariadb-recovery…	helm upgrade --install, chart par chart, dans le bon ordre	plan, install, upgrade, status, access, uninstall ; idempotent

Ordre et attente intégrés : okadm suit les dépendances du manifeste de release, attend chaque composant et reprend où il s'est arrêté.

Deux modes : lab et production

	Lab	Production
Cluster	kind créé à la volée, 1 nœud	Cluster existant (Talos), 3 nœuds ou plus
Réplicas	1, sans autoscaling	3, autoscaling
Réseau	CNI du cluster et OVN central autonome	Kube-OVN et Multus
Stockage	StorageClass par défaut, 5 Gi max	Longhorn (RWO et RWX)
VM	KubeVirt en émulation (sans /dev/kvm)	KubeVirt avec KVM
Domaine	Optionnel (nip.io)	--domain obligatoire



okadm install : les étapes

1 okadm plan

Vérifie version Kubernetes, nœuds, opérateurs, StorageClass ; donne la commande de correction

2 Cluster kind

En lab sans cluster : créé via la bibliothèque kind, ports 80 et 443 avec --ingress

3 Opérateurs

Installe les prérequis manquants, attend CRD et Deployments

4 Charts Helm

Chart parapluie embarqué, découpé : openkeel-infra puis une release par composant

5 Données

MariaDB Galera en TLS, RabbitMQ et vhost ; secrets par composant ; retire Memcached et Redis

6 Composants

Releases openkeel-* dans l'ordre des dépendances, migrations de schéma

7 Flavors et image

m1.tiny à m1.xlarge ; image Alpine 3.24 cloud-init dans Glance

8 Accès

URL de la console Skyline, Keystone, compte admin, clouds.yaml exporté

Opérateurs et prérequis

Installés par okadm en lab

cert-manager
v1.21.3

MariaDB operator
26.10.2

RabbitMQ cluster operator
v2.23.2

Messaging topology operator
v1.20.5

metrics-server
v0.9.0

Multus CNI (thick)
v4.3.1

KubeVirt
v1.9.1

CDI
v1.66.1

Traefik (--ingress)
chart 41.7.1



Production : vérifiés par okadm plan

- Kube-OVN ≥ 1.16
- Longhorn (CSI bloc et RWX)
- Medik8s ≥ 0.12
- Rook-Ceph pour S3 et Swift
- KubeVirt ≥ 1.9, CDI ≥ 1.66
- Opérateurs MariaDB, RabbitMQ
- cert-manager ≥ 1.16 (PKI interne)

Production : quelle CNI pour les pods ?

Aujourd'hui : Kube-OVN

Exigé par okadm plan en production. Avec l'OVN parallèle d'OpenKeel (D2), chaque nœud porte deux piles OVS/OVN qui partagent le même module noyau : jugé fragile.

Recommandé : Cilium

Une CNI non OVN pour le réseau des pods. Chaque hyperviseur et chaque passerelle n'ont plus qu'une pile OVS/OVN, celle d'OpenKeel ; les VMs s'y attachent par Multus.

Statut : question ouverte

Cilium ou Calico, ou Kube-OVN partageant son ovs-vswitchd (à valider). okadm plan reconnaît déjà Cilium ; en lab, la CNI du cluster reste en place.

Séparation des rôles : Cilium pour les pods de la plateforme, l'OVN d'OpenKeel pour les réseaux des tenants.

Après l'installation : prêt à tester

Flavors créés par défaut

Nom	vCPU	RAM	Disque
m1.tiny	1	512 Mo	1 Go
m1.small	1	2 Go	20 Go
m1.medium	2	4 Go	40 Go
m1.large	4	8 Go	80 Go
m1.xlarge	8	16 Go	160 Go

hw_rng:allowed et resources:DISK_GB=0 (disques racine = PVC)



Image de test

Alpine 3.24.3 cloud-init enregistrée dans Glance ; l'architecture du nœud (x86_64 ou aarch64) choisit le fichier, vérifié par SHA-512.



Accéder à la console

Avec --ingress
`skyline.127.0.0.1.nip.io`
Sinon, port-forward
`svc/openkeel-skyline-api 9998`
Compte
admin, projet admin, domaine Default ; mot de passe dans le secret openkeel-keystone-admin

08

Iso-fonctionnalité et tests

Tout est testé et validé fonctionnellement contre le composant d'origine



Iso-fonctionnel : rien ne change pour l'écosystème



Même API REST

- Chemins, paramètres, codes HTTP
- Corps d'erreur et microversions
- Policies identiques



Même base de données

- Mêmes tables et colonnes
- Mêmes migrations de schéma
- Données existantes réutilisables

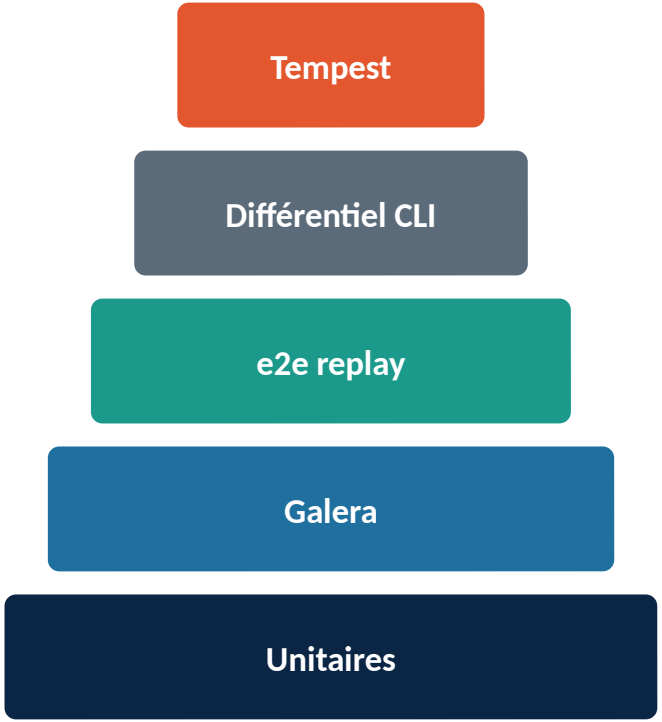


Mêmes messages RabbitMQ

- Enveloppes oslo.messaging
- Mêmes RPC et topics
- Notifications versionnées

Un composant OpenKeel remplace l'original sans que les clients, SDK, Terraform, Tempest ni les services voisins voient la différence.

Cinq niveaux de tests



Suites officielles et plugins (neutron, cinder, keystone, octavia, barbican, designate, heat, manila…) contre le lab

ok et openstack lancés côte à côte : mêmes requêtes HTTP, mêmes sorties, mêmes codes retour

Échanges enregistrés auprès du service Python, rejoués contre le Go (make e2e, report.json)

Tests d'intégration contre un vrai cluster MariaDB Galera

go vet, go test -race -cover ; faux clients Kubernetes et Helm

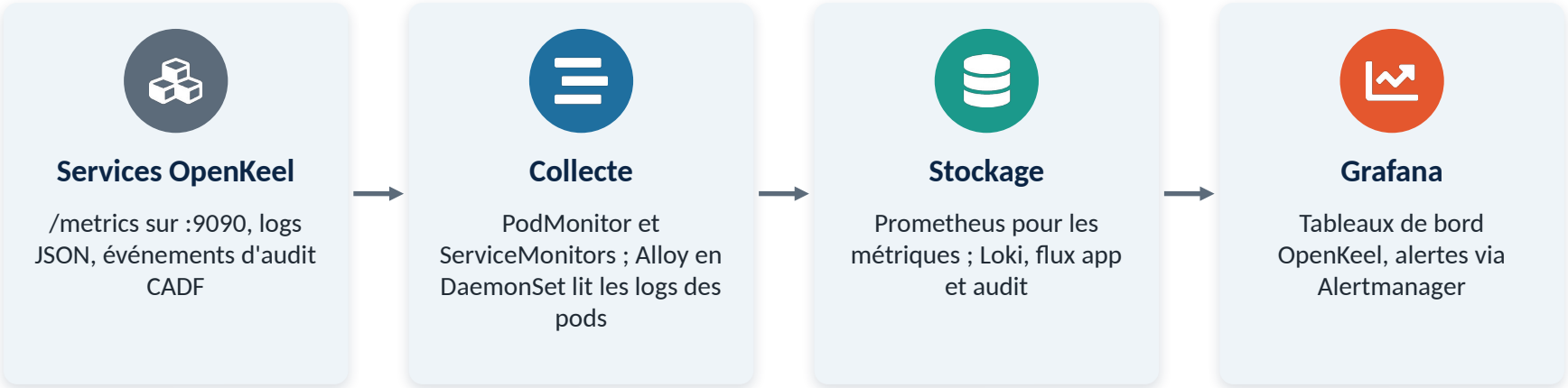
09

Observabilité

Métriques, logs et piste d'audit dans Grafana



La stack d'observabilité



kube-prometheus-stack
92.10.1

Loki
chart 7.3.0, Loki 3.6.12

Alloy
chart 1.13.1

Exporters
MariaDB, RabbitMQ, KubeVirt, Longhorn

Activation : `okadm install --grafana` (ou GRAFANA=1 make lab-up), namespace openkeel-monitoring

Piste d'audit, rétention et alertes

400 j audit en production

30 j audit en lab

31 j logs applicatifs en production

Alertes PrometheusRule

- OpenKeelAuditEventsDropped (critique)
- OpenKeelAuditEventsFailed
- OpenKeelServiceTargetDown
- OpenKeelServiceNoTarget

Loki fournit une piste d'audit consultable, pas une archive inviolable : stockage WORM ou SIEM externe relèvent de la plateforme.

10

Sécurité et SecNumCloud

Chiffrer tous les flux, tracer, cloisonner : un socle SecNumCloud-ready



Tous les flux internes chiffrés en TLS

Flux	Protocole	Confiance
Clients → Ingress → API	HTTPS, ré-chiffré vers les backends (Skyline, console Nova, wsproxy Zun)	Certificat public ; certificats internes cert-manager
API ↔ API	REST en HTTPS, URL internes https du catalogue	PKI interne cert-manager, émetteur openkeel-ca
Services → MariaDB	SQL over TLS, obligatoire côté serveur	CA de l'opérateur MariaDB, vérifiée par chaque service
Services → RabbitMQ	AMQPS (RPC et notifications)	PKI interne openkeel-ca
Neutron → OVN NB/SB	OVSDB ssl:, TLS mutuel	Certificats client et serveur OVN



Plus de Memcached ni de Redis : aucun composant ne les lisait ; okadm les retire. Seuls les ports :9090 (santé, métriques) restent en HTTP pour l'instant.

SecNumCloud-ready : où en est-on ?

Fait

- Go open source, sans télémétrie
- Audit CADF, archive WORM 12 mois
- RBAC OpenStack appliqué
- Barbican : HSM PKCS#11, OpenBao
- Zun sous Kata ; OVN en mTLS
- MFA TOTP ; MariaDB en TLS
- Memcached et Redis supprimés

En cours

- RabbitMQ en AMQPS
- Certificats cert-manager des API
- URL internes en https
- Ingress vers backends HTTPS
- Provider OVN-Octavia sur TLS

À faire

- OIDC + MFA (IdP européen, K8s)
- Clé par client, effacement crypto
- Crypto ANSSI, HSM qualifié
- Images signées, SBOM, Kyverno
- SI d'administration séparé
- Audit kube-apiserver et Talos
- Moindre privilège des comptes

SecNumCloud qualifie un service, pas un logiciel. Validation : matrice d'exigences du référentiel 3.2, pré-audit, pentest PASSI, CSPN éventuelle.

Ce qu'il faut retenir



Une méthode industrialisée

Reverse-spec, diagramme puis code Go, par des skills Claude réutilisables et améliorées à chaque composant



Kubernetes fait le travail lourd

KubeVirt, CSI, OVN, Rook-Ceph, ExternalDNS, Medik8s



Iso-fonctionnel et vérifié

Mêmes API, base et messages ; validé contre l'original par replay et Tempest



Prêt à opérer

okadm plug and play, TLS interne, audit intégré, socle SecNumCloud-ready