V1.0 – mars 2025

Cet article rassemble mes notes et remarques pour la mise en place d’un pool d’instances Oracle APEX dans un cluster Kubernetes (k8s).

Préambule

Demander une douzaine d’APEX comme on passe commande d’un plateau de fruits de mer, c’est un peu la promesse de Kubernetes, sans compter que *K8s* est un « Terminator » qui va jusqu’au bout de son contrat.

Mon objectif était de bâtir, à partir du produit *Oracle Database 23ai Free*, une solution Oracle APEX sans couts de licence, capable de supporter un très grand nombre d’utilisateurs. Pour résumer : une Database « applicative » et une armées d’ Oracle APEX en frontal.

Pourquoi un pool d’instances Oracle APEX ?

Quand on souhaite utiliser Oracle APEX à une large échelle, il faut adapter les ressources en conséquence et cela conduit à utiliser un serveur de plus en plus puissant. Cette augmentation élimine au passage tout recours à des licences Oracle Database de type *free* en raison des limitations de ces versions (2Go RAM, 2 cores, 12 Go disk …) mais … il existe des cas de figure où Oracle APEX peut être utilisé de façon partitionnée, c’est-à-dire plusieurs instances APEX indépendantes accédant à une même Database de données via des DB Links par exemple, ou via des API REST.

Rappel sur la plateforme de développement Oracle APEX

Le produit Oracle APEX est une plateforme de développement dont les prérequis sont :

- Une Database Oracle avec une licence payante ou gratuite, *on-premis*e ou bien en cloud. Une Database Oracle est obligatoire dans tous les cas.
- Le produit Oracle ORDS, dont la licence est gratuite. Il s’agit d’un serveur qui embarque *Jetty*, un serveur web et container de servlets assurant la communication entre les requêtes http et le code PL/SQL constituant Oracle APEX.
- Le code Oracle APEX qui se compose de divers objets à installer dans la Database. Sa licence est gratuite.

Une instance APEX correspond à l’installation physique du code APEX dans une Database. Une instance peut héberger des espaces logiques de travail appelés *workspaces*. Les développeurs et les utilisateurs travailleront toujours dans le contexte d’un *workspace*.

Habituellement, l’instance APEX et les données applicatives sont colocalisées dans la même Database et de ce fait, une question intéressante est de savoir comment se répartit la consommation de ressources. En effet, il y a l’activité relative à la manipulation des données applicatives et puis celle qui est spécifique au fonctionnement d’APEX (accès aux metadata et génération dynamique des pages). Si l’on sépare l’activité d’APEX de celle liée aux DATA applicatives, la mise a l’échelle pourra être réalisée de façon plus rationnelle.

Voici quelques cas d’usage pour notre pool APEX :

- Démarrer rapidement un grand nombre d’instances APEX, y déployer une ou plusieurs applications et accéder à une Database commune pour les data

applicatives partagées.
- Organiser un atelier de formation avec plusieurs centaines de participants
- Lancer une application *online* sur internet sans avoir une connaissance exacte de l’audience attendue et pouvoir réagir rapidement pour adapter les ressources. Par exemple, dans le cas d’une application que j’avais faite récemment, *Quiz Louvre*, cette architecture convient très bien car il n’y a aucune donnée partagée, l’application est en *read-only* et cela devient simple de *scaler* horizontalement.

Approche et choix techniques

Je me suis imposé la contrainte de n’utiliser que les licences gratuites disponibles chez Oracle tout en sachant les limitations d’Oracle Database *Free :*

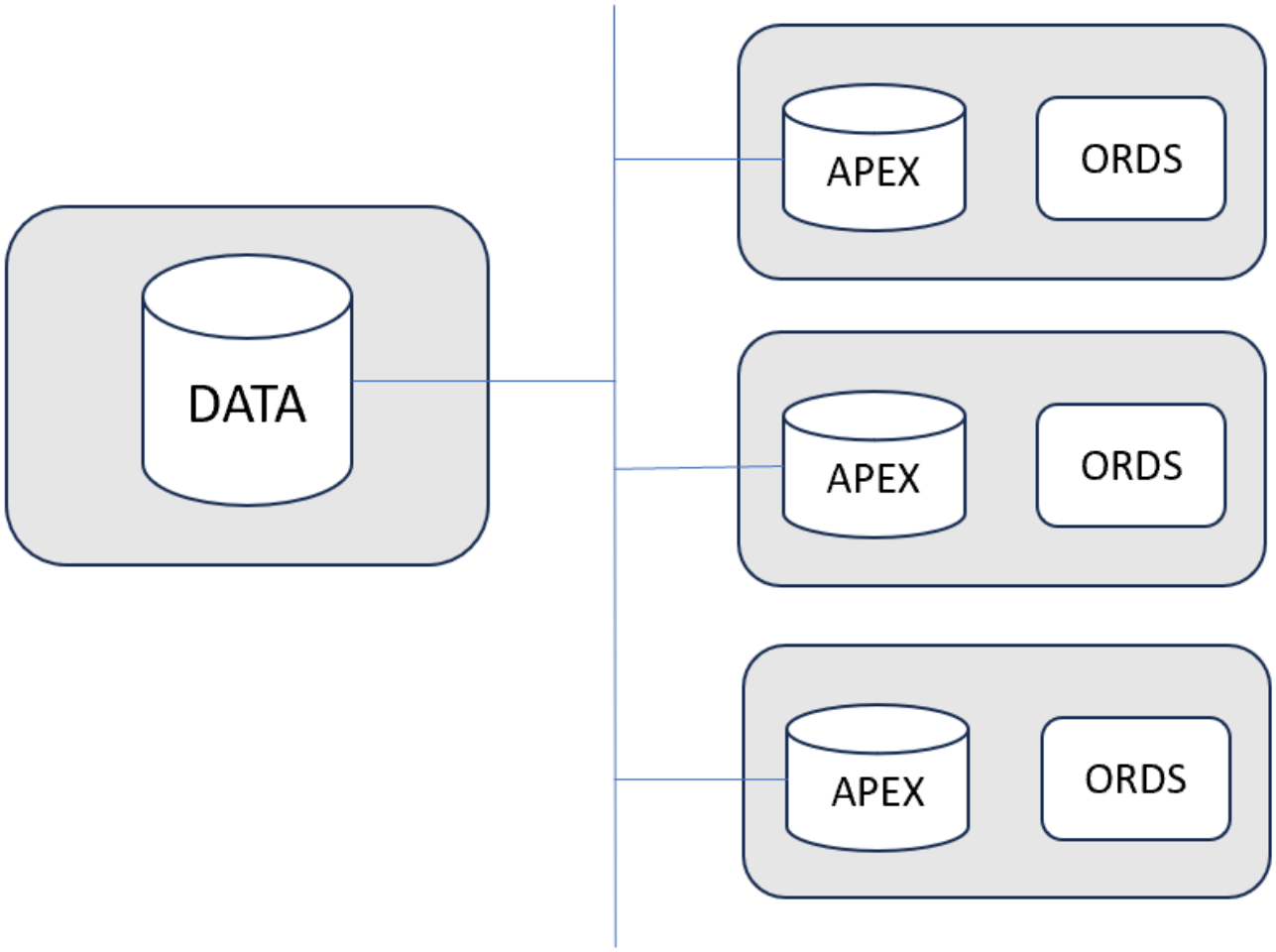
- Se limite automatiquement à 2 cores, 2 Go RAM et 12 Go disque
- Une seule installation par *node* (ou vm)

Contenu d’un *pod*

Dans l’approche initiale, j’ai considéré que le tandem « database dédiée à APEX + ORDS » constituait l’entité qui ferait l’objet de la *clusterisation*. Je n’ai donc pas fait un découpage entre la database APEX d’une part et plusieurs ORDS en frontal d’autre part. Par conséquent, le *pod* (au sens Kubernetes) relatif à l’activité APEX est constitué de deux containers :

- *free* utilisé pour la Database APEX
- *ords-developer* utilisé pour ORDS.

Schéma simplifié du cluster



Dans le schéma ci-dessus, Les *pods* sont matérialisés par une boite de couleur grise. la database DATA est de type *free*. Elle présente des limitations, comme indiqué précédemment, mais cette database DATA peut très bien être une database Oracle avec une licence de type Standard ou Enterprise, et se situer dans un environnement *on-premise* ou sur le cloud. APEX communiquera avec la database DATA via un database link.

Mise en place du cluster

Etapes pour créer un pool de pods APEX et un pod DB centrale (DATA)

J’ai testé le concept chez OVH en utilisant leur support de Kubernetes.

Les *nodes* sont de type D2-8 (4 CPU 8 GB RAM 50 GB disk)

Remarque : La configuration D2-4 (2 CPU 4 GB RAM 50 GB disk n’est pas adaptée car le *pod* sort en status « *evicted* » au bout de quelques secondes (mémoire insuffisante ?).

Cf tableau des couts pour avoir une estimation du budget

Etapes préliminaires au niveau d’OVH

- Créer un cluster Kubernetes
- Ajouter un pool de plusieurs *nodes*
- Labéliser les *nodes* : (n-1) de type *front* et un seul de type *back* (cf script en PowerShell nodes.ps1)

Choix des images pour les containers

Afin de limiter les taches de setup, je m’appuie sur les images docker suivantes deux images docker qui sont fournies par oracle :

- *free 23ai*

Je préfère prendre *free 23ai* plutôt que *express* car *free23ai*, parmi ses fonctionnalités nouvelles, élimine le besoin de gérer un wallet db pour accéder à des ressources web et c’est un très grand avantage. Parmi les variantes, il faut choisir *full*. La version *Lite* rend l’installation d’APEX impossible car il manque des packages.

- *ords-developer*
  Il s’agit de l’image qui se charge de l’installation d’APEX, à la différence de celle qui s’appelle *ords* et qui n’installe que la partie ords.

Pour l’image *free*, j’ai choisi de la faire démarrer avec sa base «pre-built».

Pour la base centrale, il s’agit d’un singleton qui pourrait être déployée sous forme de *ReplicaSet* avec un niveau réplica=1 mais dans mon cas, j’ai choisi l’option *statefulSet* (nom DNS stable même après un redémarrage). Cf db.yaml

Comme chaque *pod* dédié à APEX doit être déployé une seule fois sur chacun des trois *nodes* possédant le label « front », j’ai choisi un type *Daemonset*. Un sélecteur installe le pod sur tous les *nodes* possédant le label « front ». cf pod-apex.yaml et le script nodes.ps1.
Si on choisit un type ReplicaSet ou Deployement, le controleur k8s peut être amené à instancier deux pods sur le même node.

Ensuite, on crée un service nommé « apex » de type clusterIP qui s’appliquera à tous les pods pod-apex instanciés. Cf apex.yaml. Je ne l’ai pas créé de type LoadBalancer car cette tâche sera accomplie par un *ingress* et c’est au niveau de cet *ingress* que l’on demandera de l’affinité de session.

Ajouter un Nginx Ingress Controller au cluster k8s.

OVH fournit un chart Helm pour réaliser cette opération.

Créer un ingress cf Ingress.yaml

Ici je demande de l’affinité de session basée sur la présence d’un cookie qui est entièrement géré par Ingress et que j’ai nommé « kub-apex ».

J’avais tenté d’utiliser un service de type *LoadBalancer* mais je ne suis pas parvenu à obtenir un niveau satisfaisant d’affinité de session. Le seul mode qui fonctionnait était l’affinité d’adresse IP, ce qi ne me convenait pas entièrement puisque je voulais des *sticky sessions* basées sur la valeur d’un cookie.

Au bout d’un quart d’heure (parfois une demi-heure) le cluster est opérationnel. On peut démarrer encore plus vite comme c’est discuté plus loin.

Récupérer l’adresse IP du contrôleur Ingress :

kubectl get svc ingress-nginx-controller -n ingress-nginx -o wide

Produits tierces utiles

Le produit Lens de Mirantis pour disposer d’ un tableau de bord permettant de visualiser d’un coup d’œil l’état des ressources du cluster Kubernetes. Sinon, il faut enchainer à l’infini des commandes kubectl.

Ci-dessous la console Lens montrant l’état des dix pods lancés sur le cluster. Un pod pour DB DATA et neuf pods APEX.

Welcome | Pods - kubernetes-admin@g...

default | Search Pods... | 10 items

Name	Namespace	Cont...	CPU	Memor	Restart:	Controlled B	Node	QoS	Age	Status
dbmain-0	default		0.043	2.4GiB	0	StatefulSet	pool1-nod	BestEffort	5m44s	Running
pod-apex-65cz2	default		1.175	2.3GiB	0	DaemonSet	pool1-nod	BestEffort	5m24s	Running
pod-apex-7sjph	default		0.000	0	0	DaemonSet	pool1-nod	BestEffort	5m25s	Running
pod-apex-9ct9n	default		0.000	0	0	DaemonSet	pool1-nod	BestEffort	5m25s	Running
pod-apex-bwv6d	default		0.000	0	0	DaemonSet	pool1-nod	BestEffort	5m24s	Pending
pod-apex-h8svp	default		0.000	0	0	DaemonSet	pool1-nod	BestEffort	5m24s	Pending
pod-apex-hq6sl	default		1.250	2.3GiB	0	DaemonSet	pool1-nod	BestEffort	5m24s	Running
pod-apex-hq8h5	default		2.347	2.7GiB	0	DaemonSet	pool1-nod	BestEffort	5m24s	Running
pod-apex-mkprg	default		0.000	0	0	DaemonSet	pool1-nod	BestEffort	5m24s	Pending
pod-apex-pnbbb	default		2.048	2.6GiB	0	DaemonSet	pool1-nod	BestEffort	5m24s	Running

Résumé de l’ordre de lancement des fichiers yaml :

Tache	Commande	Ordre
Affecter label aux nodes	Powsershell label_nodes.ps1	1
Créer les secrets	Kubectl create secret ords-secret	2
Créer les CongigMaps	kubectl create configmap …	3
Créer DB DATA	Kubectl apply -f db.yaml	4
Créer les pod Apex	Kubectl apply -f pod-apex.yaml	5
Créer le service Apex	Kubectl apply -f apex-run.yaml	6
Créer le Ingress	Kubectl apply -f ingress-run.yaml	7

On suppose que le *Ingress Controler* a déjà été ajouté juste après la création du cluster K8s.

Remarque à propos des dépendances entre containers :

L’image ords-developer vérifie au startup la présence d’un fichier conn_string.txt et teste une connexion basée sur la chaine qui se trouve à l’intérieur de ce fichier. Si cela échoue, la création du container est abandonnée. Dans *Docker compose* il existe une

directive qui permet de préciser quelle est la condition de démarrage, mais pas dans Kubernetes. J’ai choisi donc de ne pas avoir recours à un container intermédiaire mais plutôt de laisser faire la nature. C’est à dire que le contrôleur K8S , dont il faut saluer l’opiniâtreté, fera autant de tentatives qu’il est nécessaire jusqu’à ce que la BD soit prête à recevoir des connexions. C’est la raison pour laquelle on observera un nombre de *restart* de l’ordre de 2 ou 3. Cela est sans conséquence.

Diminuer le temps de démarrage de la plateforme APEX

Pour un chargement plus rapide, il est intéressant de préalimenter la base APEX avec les objets spécifiques à APEX. Cette opération dure en effet une quinzaine de minutes.

Après avoir fait un *commit* du container de DB Apex dans son état « installée, j’ai obtenu un nouvelle image que j’ai poussée sur un registre privé chez OVH.

Un déploiement de la configuration pod-apex ne prend alors que moins de deux minutes au lieu de quinze. A chaque nouvelle version de l’image ords-developer, il faudra régénérer cette image pour conserver l’avantage de gain de temps.

Mode de routage vers le pool en fonction du scénario

Lorsqu’il s’agit d’un scénario de déploiement d’une application, c’est-à-dire un contexte dans lequel c’est essentiellement le runtime d’APEX qui sera sollicité, alors on privilégiera un load balancing basé sur l’affinité de session. Ici, l’objectif est de garantir l’utilisation la plus symétrique possible et ce n’est pas l’utilisateur qui choisit son pod.

S’il s’agit en revanche d’un scénario où l’on utilise les capacités de développement d’APEX, alors il est primordial de pouvoir travailler dans la durée sur une instance

(durée élevée de cookie) ou mieux, connaitre l’instance sur laquelle on travaille, de façon à pouvoir y revenir ultérieurement à partir d’un autre *device,* par exemple. Ici, on créera autant de services qu’il y a de *pods* et on fixera des règles de routage au niveau de l’Ingress, basées sur le nom de host virtuel. Par exemple, http:/host1.mydomain/ords/apex sera routée systématiquement vers le service associé au pod_1 et http:/host1.mydomain/ords/apex sera dirigé vers pod_2., etc. Cela implique donc de créer des noms de host au niveau de son DNS.
Sinon, il faudra créer des services de type NodePort, récupérer les adresses IP affectés et utiliser ces adresses distinctes dans l’url.

J’ai essayé d’utiliser un routage basé sur un path d’url mais je n’ai pas obtenu de résultat correct.

Scripts *custom* ajoutés dans le container ords-developer

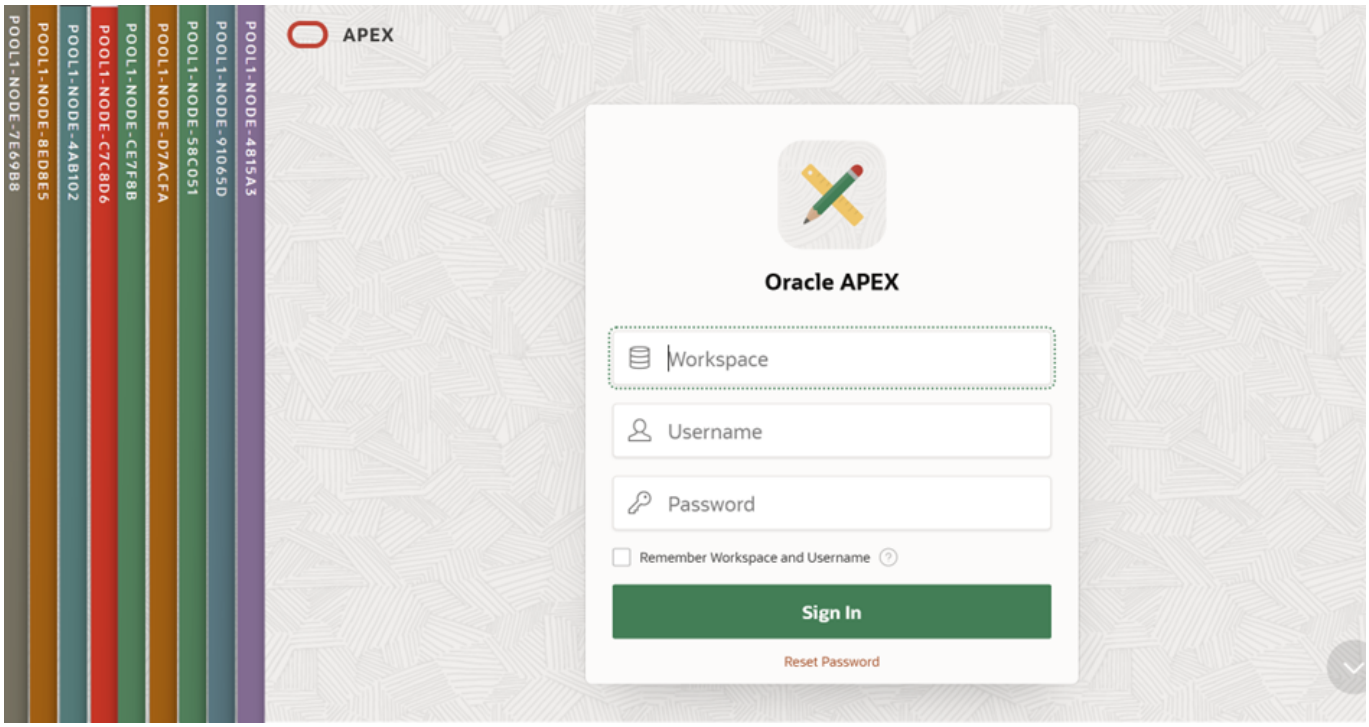
Afin de disposer d’une instance la plus opérationnelle possible, j’ai rajouté plusieurs scripts qui réalisent les taches additionnelles suivantes :

- Création d’un workspace par défaut qui s’appelle DEMO
- Ajouts des ACL pour permettre un accès à des ressources externes sur internet depuis une application APEX.
- Ajout d’une bannière avec le nom du *node* pour identifier rapidement sur quelle instance on se situe
- Modification de certaines directives ORDS dans le cas d’une utilisation d’Ingress en proxy https.
- Création d’un *database Link* vers la base DATA
- Import optionnel d’une application APEX existante

Tous les scripts sh et sql sont enregistrés dans un configmap (ords-configmap) qui est

monté comme un volume et mappé sur un répertoire. Pour rappel, la taille maximale d’un configMap est de 1 Mo.

La bannière de type « instance » est créée à partir d’un api plsql fournie avec APEX. On prend le nom du node ainsi que son numéro d’index qui servira pour fixer une couleur arbitraire entre 1et 16.



Ci-dessous, une répétition de connexions dans des *Private Windows* a abouti à neuf sessions sur chacun des neuf pods APEX disponibles.

Authentification

Chaque Workspace possède son annuaire d’utilisateurs autorisés à se connecter. Dans le cas d’un pool, il sera plus pertinent de déléguer la phase d’authentification à un module tiers, comme un annuaire LDAP, par exemple ou un OAUTH.

TLS

Je n’ai pas mis en place TLS dans mon prototype.

Les étapes sont décrites dans un document OVH.

Couts de fonctionnement

Pour trois instances APEX et une base partagée, voici les couts minimums escomptés en euros avec une projection pour une durée d’une journée de dix heures ou une durée d’un mois .

Elément	Description	Nombre	Tarif horaire	Tarif mensuel	Montant /heure	Montant /jour	Montant /mois
Nodes	D2-8 (4 CPU 8 GB RAM 50 GB disk)	4	0,0357	19,8	0,1428	1,428	79,2
Load Balancer	Size Small	1	0,0083	6,059	0,0083	0,083	6,059
Adresse IP	Adresse IP	1	0,0025	1,5	0,0025	0,025	1,5
Registry (*)	200 Go, 15 connexions	1	0,0237		0,0237	0,237	17,064
TOTAL					0,1536	1,536	103,823

* La Registry est optionnelle.

Les *nodes* qui ont été utilisés chez OVH sont référencés dans la catégorie *Discovery*, ce qui signifie que leur cout horaire est attractif mais que leurs performances peuvent être variables. Ces *shapes* sont idéales pour faire des tests.

Pour un scénario avec un nombre raisonnable d’utilisateurs (une cinquantaine ?),

l’offre Oracle nommée *Oracle APEX Service* est plutôt compétitive (122 USD/mois à la date de mars 2025).

En *compute*, une instance Oracle équivalente VM.Standard.E4.Flex (2 OCPU, 8 Go RAM, 100 Go disk) revient à 46,85 euros/mois. Comparable à la B2-15 de chez OVH qui est à 46,20 €/mois. Celle que j’ai utilisée chez OVH est une instance de catégorie Discovery, donc moins prédictible qu’une B2-15.

Fichiers de configuration

J’ai regroupé les fichiers principaux. Ces fichiers sont également accessibles depuis github.

Quelques liens utiles

Livre « Kubernetes » 2ieme édition de Kelsey Hightower, Brendan Burns ,Joe Beda et Lachlan Everson

Je remercie mon tuteur ChatGPT pour la mise au point des fichiers de configuration, le debug de toutes les misères rencontrées ainsi que pour m’avoir aidé dans la compréhension générale de Kubernetes.

Author

-

  Patrick

GPM Factory