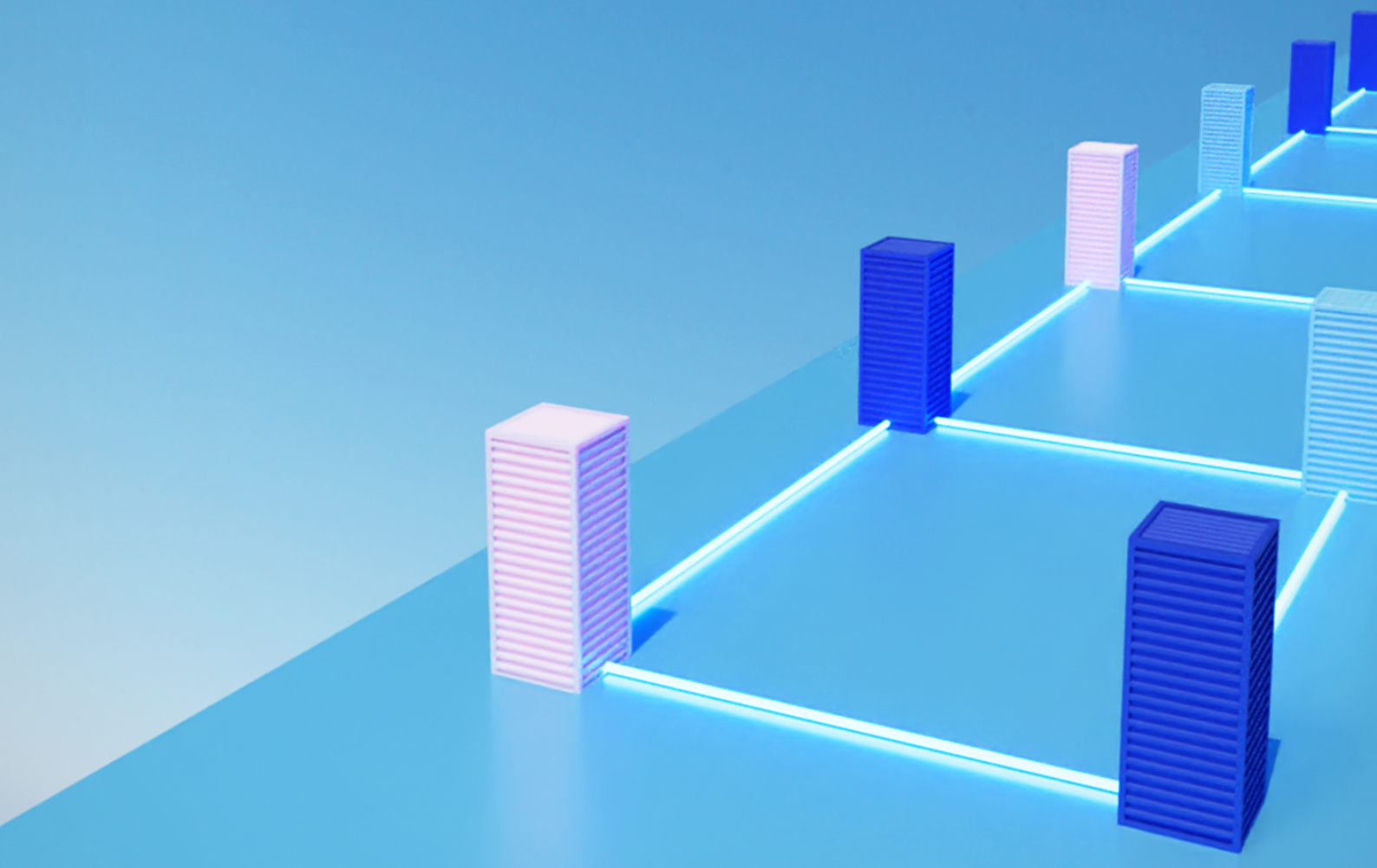
cellenza

PRÉPARER L'ADOPTION DE KUBERNETES

Orchestrez vos applications Cloud-Native



Le mois du CONTENEUR

en partenariat avec Microsoft

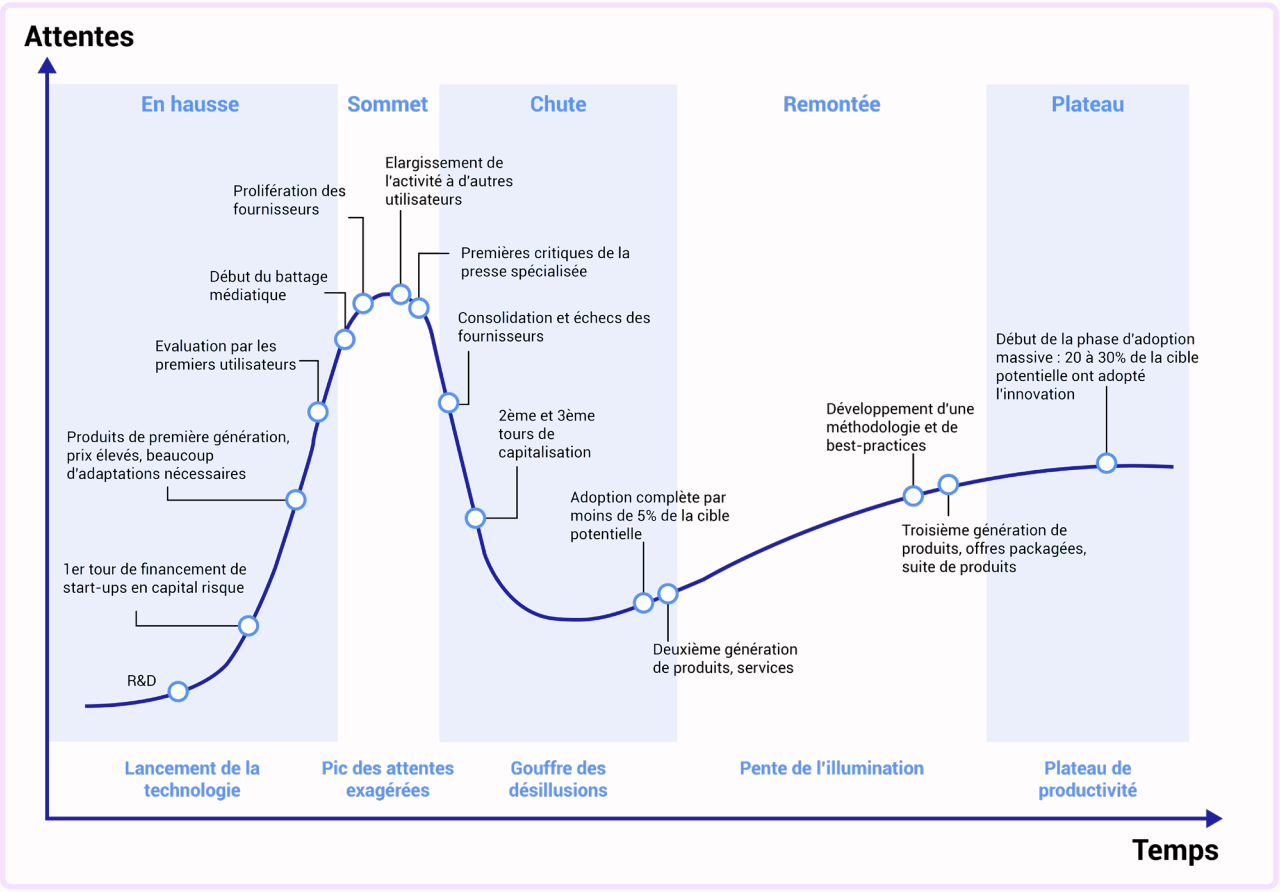
SOMMAIRE

01 INTRODUCTION

On parle de plus en plus de Kubernetes : cette solution n'est pourtant pas nouvelle puisqu'elle existe depuis plusieurs années. On la présente même comme LA solution pour héberger nos futures applications. La première version de Kubernetes est née en juin 2014, on peut donc la considérer comme mature d'un point de vue technologique. Pourtant, c'est en ce moment qu'on en parle le plus. Pas un fournisseur de solution Cloud qui n'a pas son produit « Kubernetes » dans son catalogue. Les éditeurs de logiciels sont de plus en plus nombreux à offrir à leurs clients des solutions autour de Kubernetes.

Dépasser le simple effet du cycle de battage médiatique

L'intérêt que nous portons à une technologie a été matérialisé par le Gartner dans le cycle du Hype représenté ci-dessous :



Où se situe Kubernetes sur cette courbe ? On a déjà quelques éléments de réponse :

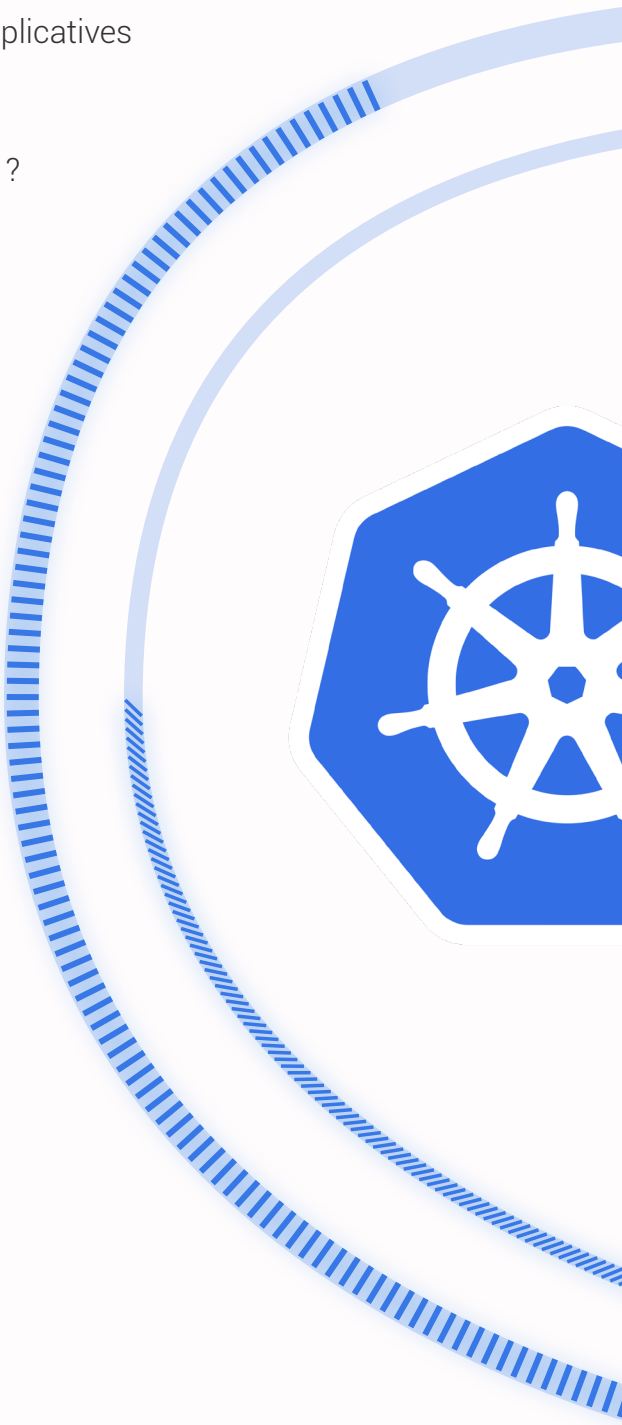
- Nous voyons de plus en plus de communications dans les médias sur la thématique de la conteneurisation ;
- De plus en plus de fournisseurs proposent des solutions en relation avec la conteneurisation.

Nous approchons donc du pic des attentes exagérées de la courbe Hype, juste avant la phase du gouffre des désillusions. Kubernetes n'échappera pas à cette courbe. D'un côté, nous voyons le potentiel de la technologie pour construire des applications dites « Cloud native ». De l'autre, nous observons l'intérêt croissant de nos clients avec beaucoup de questions. Forte de ses expériences, Cellenza s'est forgé des convictions sur le sujet, que nous partageons avec vous dans ce livre blanc.

Kubernetes est une thématique complexe que nous pourrions aborder en parlant de technologie mais le sujet est beaucoup trop vaste. Le besoin de nos clients est déjà de répondre à un certain nombre de questions :

- En quoi Kubernetes est-il différent des plateformes applicatives actuelles ?
- Suis-je prêt à l'adopter comme plateforme applicative ?
- Quels bénéfices puis-je espérer pour mes applications ?
- Si Kubernetes rend mes applications indépendantes de ma plateforme, cela réduit-il ma dépendance par rapport à mon fournisseur (Vendor-Locking) ?
- Quelles sont les étapes pour l'adopter ?
- Quels sont les challenges à relever pour adopter cette plateforme applicative ?

Ce livre blanc a donc pour objectif de répondre à ces questions et vous fournir nos convictions sur différentes thématiques en relation avec Kubernetes.



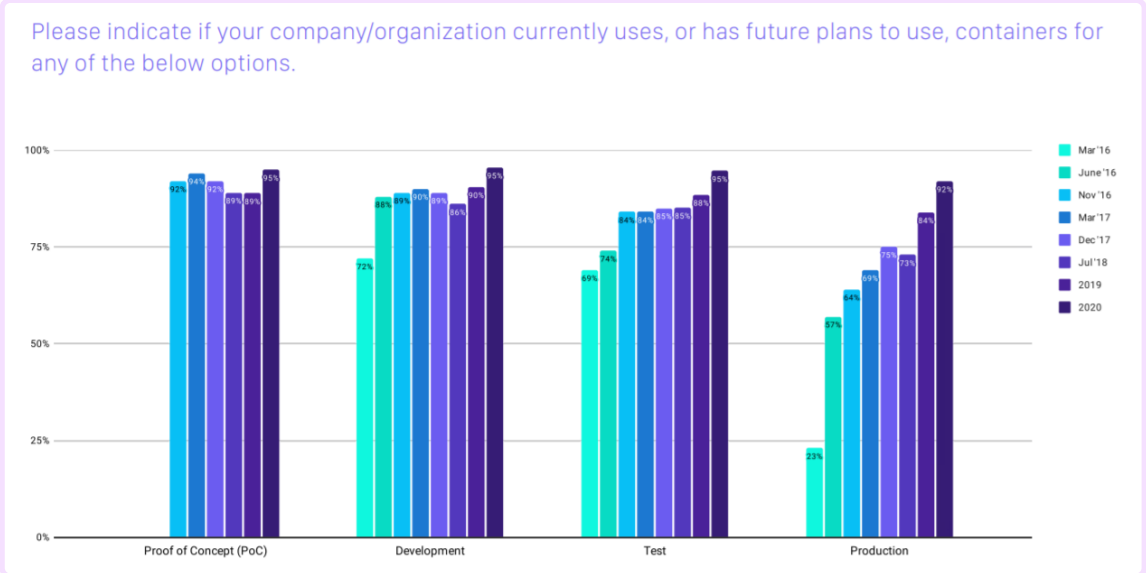
Kubernetes fait-il l'unanimité ?

On pourrait croire que Kubernetes s'est imposé par la force mais en réalité, il n'en est rien. La solution a été adoptée car elle était soutenue par toute l'industrie. Historiquement, c'est un projet interne développé par Google pour répondre aux défis de la conteneurisation des applications. Google a fait le choix de proposer sa solution en open-source au travers de la Cloud Native Computing Foundation. L'adoption a été massive. On aurait pu penser qu'un acteur comme Docker, qui a formalisé les principes des conteneurs, était le mieux placé pour répondre au challenge de la scalabilité des conteneurs. L'histoire retiendra que beaucoup ont essayé de le concurrencer mais finalement, tous se sont rangés de son côté. Kubernetes fait donc l'unanimité au point de devenir une norme.

Comme indiqué précédemment, tous les fournisseurs de solutions Cloud / éditeurs ont Kubernetes à leur catalogue :

- Microsoft Azure avec AKS ;
- Amazon Web Services avec EKS ;
- Google avec GKE ;
- Oracle avec Container Engine for Kubernetes ;
- Red Hat avec Red Hat OpenShift ;
- VMware avec VMware Tanzu.

L'illustration ci-dessous est issue de l'étude menée par la Cloud Native Computing Foundation en 2020. On peut constater que l'adoption de la plateforme est très forte et pas seulement pour tester la solution. En 2020, un grand nombre des répondants de l'étude font tourner leurs applications sur la plateforme Kubernetes en environnement de production.



Source : CNCF Survey Report 2020

Selon cette représentation, Kubernetes apparait comme une solution « mainstream » que tout le monde peut déployer en production. D'après l'étude, son usage en environnement de production a augmenté de 78% cette dernière année. Il faut toutefois relativiser ces chiffres : en effet, toujours selon cette même étude :

- Les répondants à l'étude travaillent majoritairement pour des entreprises de grande taille (supérieure à 5000 personnes) ;

- 43% des répondants annoncent opérer des clusters d'une taille relativement modeste (entre 5 et 100 nœuds physiques ou virtuels), alors que 17% des répondants opèrent des clusters de très grande taille (plus de 5000 nœuds) ;

- La majorité de ces entreprises (près de 60%) sont liées au secteur du logiciel / technologie ;

- 26% des répondants sont en capacité mettre en production une nouvelle version de leurs applications chaque semaine.

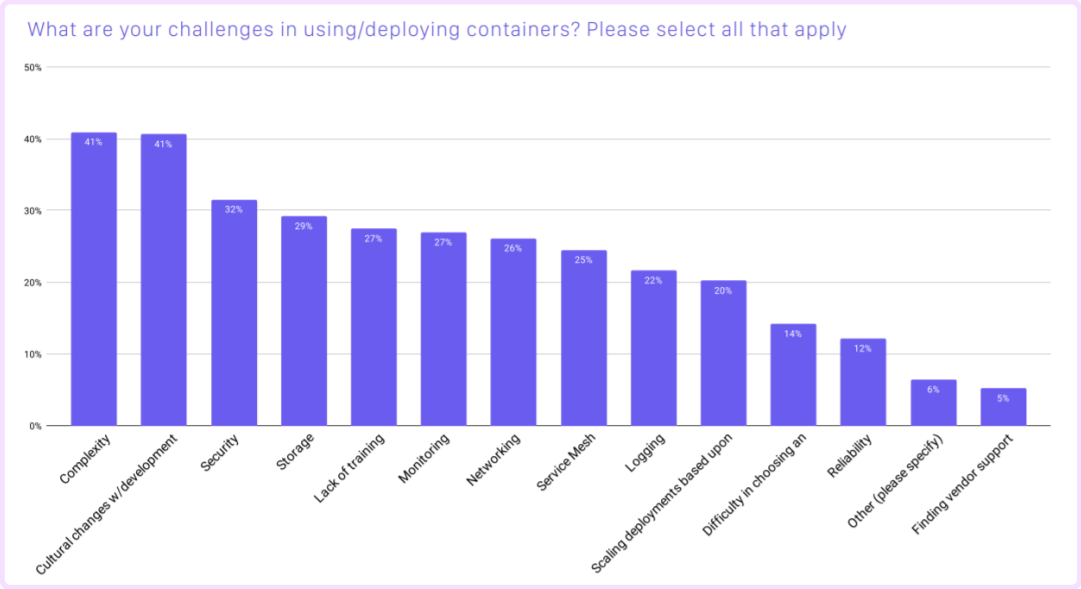
Nous reconnaissons-nous dans le profil des répondants de cette étude ? Non, c'est pour cette raison qu'il faut la relativiser. Les répondants ont déjà massivement investi dans la plateforme et en ont acquis une maîtrise, mais ce ne sont pas les consommateurs les plus représentatifs.

Au final, Kubernetes fait certes l'unanimité dans l'écosystème des fournisseurs mais **est-ce la plateforme dont vous avez besoin pour héberger vos applications ?** C'est une des questions auxquelles Cellenza s'attachera à répondre dans ce livre blanc. **Kubernetes n'est pas une solution universelle.**

Pourquoi cette thématique des conteneurs ?

Kubernetes est une tendance de fond qui a commencé avec Docker il y a quelques années en révolutionnant la manière de construire nos applications. Docker a facilité la portabilité de nos applications en les rendant indépendantes de la plateforme qui les héberge, ce qui a eu pour conséquence de rendre les équipes de développement plus autonomes dans le développement des applications. Cette nouvelle manière de construire et d'héberger des applications a nécessité de réviser la manière même de construire ces applications dites « Cloud native » qui sont à la fois portables, résilientes et adaptables à la charge de travail à la demande. Cette qualification est issue de retours d'expériences qui ont été formalisés dans la méthodologie des « Twelve Factor App ». **Adopter Kubernetes, c'est aussi adopter une nouvelle approche de conception des applications.** On ne peut adopter l'un sans l'autre. La conséquence de cette double adoption est que cela génèrera des frictions.

Toujours selon les répondants à l'étude du CNCF, il est admis que Kubernetes est une solution apparaissant comme très complexe et nécessitant beaucoup d'efforts pour l'adopter. Ces deux challenges sont même perçus comme plus importants que la sécurité. Dans le contexte du Cloud, la complexité de Kubernetes aura tendance à s'estomper car les fournisseurs Cloud nous le proposent sous la forme de services managés. La thématique du changement culturel est un des axes que Cellenza a retenus pour aborder ce livre blanc.



Source : CNCF Survey Report 2020

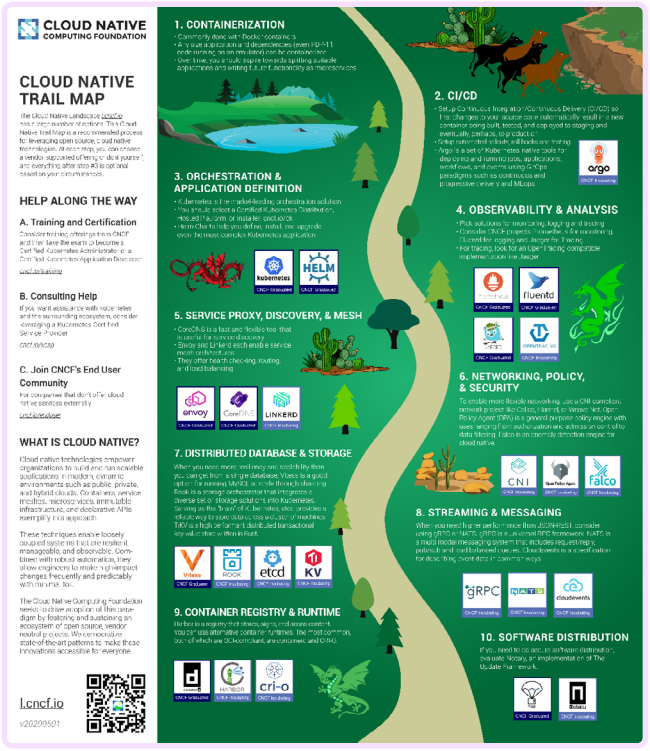
Enfin, la dernière raison pour laquelle Cellenza a choisi d'aborder la thématique des conteneurs est que cette dernière intéresse de plus en plus nos clients. Nous avons pu construire avec eux nos convictions dans différents domaines et apporter des réponses à un certain nombre de questions :

- Est-ce adapté pour héberger mes applications existantes ?
- Comment élaborer des applications « Cloud native » ?
- Quels patterns utiliser pour construire nos futures applications ?
- Comment migrer des applications existantes ?
- Que doit contenir ma plateforme pour héberger des applications en production ?
- Comment l'intégrer dans mon système d'information ?
- Mes applications seront-elles réellement portables d'un fournisseur Cloud à un autre (stratégie multicloud) ?
- Comment aborder la sécurité de la plateforme et des applications hébergées ?

L'objectif de ce livre blanc est d'apporter des réponses à ces différentes problématiques.

Trouver son chemin dans l'écosystème

D'un point de vue technique, Kubernetes est un assemblage de projets open-source portés par le CNCF. Dans l'illustration ci-contre, nous voyons les différents projets en relation. C'est ce qui fait de Kubernetes un écosystème à part entière, à côté de celui d'Azure.



Source : https://github.com/cncf/landscape#trail-map

Kubernetes est disponible dans Azure sous forme de service managé (Azure Kubernetes Services). Le service est pleinement intégré à l'écosystème Azure mais on ne peut faire l'impasse sur la partie open-source. Pour vous donner une idée de cet écosystème, nous vous invitons à aller voir le panorama disponible ici. En conséquence, c'est donc un écosystème open-source que nous devons adopter.

Par où commencer ?

Vous l'aurez compris, Kubernetes est un univers vaste offrant de multiples possibilités et alternatives pour construire la plateforme qui va héberger vos applications. Par expérience, cela implique de traiter un certain nombre de thématiques et donc de faire des choix. L'objectif de ce livre blanc sera donc de **vous aider à faire vos choix pour construire la plateforme permettant d'héberger vos applications**. Les différents chapitres de ce document vous permettront de répondre aux questions essentielles :

- Comment intégrer la plateforme dans ma stratégie applicative ?
- Pour quels usages applicatifs ?
- Comment exploiter la richesse de l'écosystème open-source pour construire ma plateforme « Production Ready » ?
- Comment l'intégrer dans ma stratégie CI/CD ?
- Comment migrer/développer des applications sur cette plateforme ?
- Comment l'intégrer dans ma stratégie de monitoring ?
- Quels sont les défis de sécurité à relever ?

C'est donc une approche pragmatique qui a pour objectif de **vous aider à intégrer Kubernetes dans votre système d'information** (pourvu que vos applications et votre culture s'y prêtent).

KUBERNETES : QUEL INTÉRÊT POUR RÉPONDRE AUX BESOINS BUSINESS ?

Le principal avantage mis en avant par la littérature autour de Kubernetes pour nos clients est d'assurer **la facilité de portabilité entre les différents Cloud providers** et ainsi réduire les effets d'un vendor locking, pouvant être source de coûts supplémentaires en cas de migration.

En dehors de cet aspect argumenté par la communauté, le but de ce chapitre est d'étudier :

- Pourquoi Kubernetes est au centre du plan de transformation des équipes Ops ?
- En quoi cet investissement peut avoir un ROI sur le long terme ?

Nous savons très bien que cette transformation des usages côté Opérations, en dehors de la mise en place purement technique, nécessite une gestion du changement et un accompagnement des équipes sur l'adaptation des processus, la manière d'architecturer, builder, déployer et sur la manière de s'organiser pour en tirer profit. Les chantiers de transformation en cours chez certains de nos clients ont pour objectif notamment de casser les silos, capitaliser sur les principes DevOps et les appliquer à leur organisation, à leurs équipes, aux besoins clients, etc., ainsi que la mise en place d'une démarche SRE (Site Reliability Engineer) soutenue par les équipes de développement et Ops.

Comme toute évolution, elle nécessite d'avoir conscience de la courbe d'apprentissage et donc d'un **besoin d'accompagnement au changement** dédié qui est propre à son entreprise, ses collaborateurs et sa maturité sur les enjeux **d'orchestration**, de **monitoring** et de **sécurité**.

A l'instar de l'étude des différentes stratégies de migration sur le Cloud, elle nécessite d'évaluer les intérêts de l'ensemble des parties prenantes afin de dégager des ROIs globaux pour la DSI. Les ROIs qui vous seront présentés dans ce chapitre ne sont pas uniquement financiers. En dehors de la rationalisation des coûts et des bénéfices que peut apporter Kubernetes aux équipes d'infrastructure dans leur quotidien, une évaluation d'**un ensemble de paramètres humains, organisationnels et temporels est à prendre en compte.**

Soutenir les développements avec un socle de delivery scalable, résilient et performant

L'enjeu principal de nos clients est de pouvoir s'assurer d'un **time to market pertinent et durable**, afin de garantir un niveau de services nécessaires à leurs clients, qu'ils soient B2B ou B2C.

Optimiser le temps et la fréquence de delivery des projets en s'appuyant sur un socle de delivery alliant scalabilité, résilience et performance, représente un **enjeu majeur** pour nos clients. Pour cela, ces derniers travaillent sur la mise en place d'organisations agiles adaptées à leur contexte et aux besoins business qu'ils doivent satisfaire. Ces organisations sont soutenues par des plans de transformation digitale généralisée, qui peuvent être cross-entités et cross-métiers.

Par le biais d'itérations et de cérémonies agiles permettant de soutenir / valider / faire évoluer le besoin d'un client, les entreprises font en sorte de maximiser l'apport de valeur, d'éviter les « effets tunnels » et de réduire la dette technique en utilisant les bonnes pratiques du marché. Pour soutenir cette démarche, **la conteneurisation permet de réduire le time to market** qui est fortement impacté par les processus de déploiement, les contraintes de production et de maintenance, et toutes actions et validations manuelles inhérentes à l'hébergement.

Kubernetes apparaît comme une solution pérenne pour répondre aux enjeux d'industrialisation de la DSI tout en restant au service de l'agilité du business. La valeur business sera donc évaluée sur la réduction du temps entre l'émergence du besoin, sa spécification et sa livraison, et sur les possibilités de tester un nouveau modèle business tout en réduisant les temps d'indisponibilité et de latence liés à l'infrastructure.

> **Kubernetes apparaît comme une solution pérenne pour répondre aux enjeux d'industrialisation de la DSI tout en restant au service de l'agilité du business.**

Livrer plus rapidement et surtout plus **fréquemment** implique la nécessité de **processus fiabilisés**, **automatisés**, sur une plateforme permettant de soutenir cette charge tout en **maîtrisant les coûts de mise en place, de formation et de maintenance.**

Etat des lieux des intérêts reconnus pour les parties prenantes

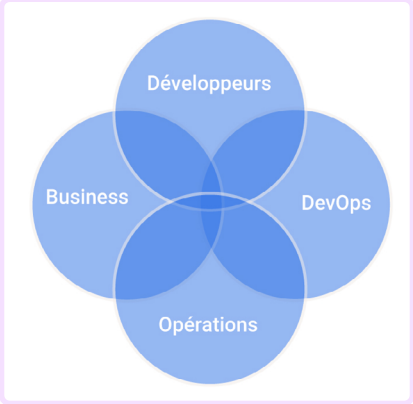
En tant que développeurs, collaborateurs des Opérations, DevOps ou représentants du métier, les avantages à la mise en place d'une stratégie de conteneurisation et d'orchestration peuvent être multiples et nécessitent pour nos clients d'établir un **plan de transformation graduel et évolutif** afin de soutenir la démarche :

Pour les développeurs :

Les développeurs représentent le premier maillon de la chaîne avec notamment les bénéfices suivants à noter :

La standardisation des processus de build et de déploiement :

- **L'artefact** est toujours le même : une **image** ;
- Sa **destination** est toujours la même : une **registry**.



Quelques **points de vigilances** à évaluer :

- Le besoin de monter en compétences sur les technologies ;
- Un accompagnement sur les bonnes pratiques à mettre en place au sein des équipes de développement (images les plus petites possible / choix des images de base / utilisation des dernières versions disponibles / sécurisation…).

Pour les DevOps :

Le principal intérêt à mentionner est lié à la **standardisation des processus de build et de déploiement** : un artefact (image) à stocker dans une registry et à déployer sur un orchestrateur.
Cet effet permet aux équipes de se concentrer sur ce qui apporte de la valeur (qualité de code, dette technique, automatisation, infra as code, etc.)

Pour le Business :

Une **amélioration du time to market** constitue l'enjeu au centre des stratégies mises en place chez nos clients.

Mais au-delà de cet avantage, Kubernetes donne aussi accès à des **environnements beaucoup plus rapides à provisionner** pour servir un besoin business nécessitant une agilité accrue (démo, POC, Sandbox…).

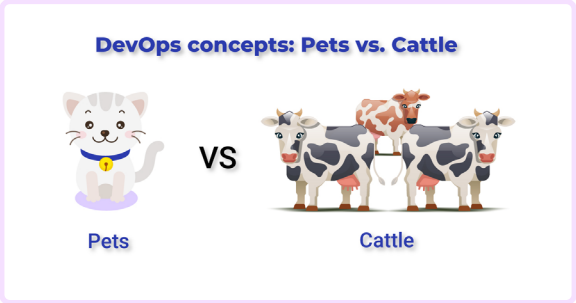
Pour les Opérations (Ops) :

La mise en place d'une stratégie de conteneurisation leur assure également plusieurs avantages :

- Une unique plateforme à gérer (au lieu d'une gestion des applications de manière individuelle avec leurs comportements en production et la gestion spécifique d'environnements) ;
- Une seule plateforme à gérer et à maintenir, indépendamment des différentes applications. Les aspects **sécurité, monitoring, scalabilité sont centralisés à un seul endroit**. La **rationalisation des coûts** est donc un axe important à prendre en compte avec la mise en place d'une démarche FinOps dès le démarrage du projet.

Soutenus par une démarche d'industrialisation (outillage, monitoring…), les Opérations pourront se concentrer sur l'apport de valeur aux équipes avec la mise en place de services adaptés selon les besoins de sécurité, de disponibilité, d'autonomie et de scalabilité des clients internes.

Introduit par Bill Baker, le concept « Pet versus Cattle » connu sein du monde DevOps s'applique aussi ici. La gestion est standardisée, automatisée avec une limitation des actions humaines, d'où l'appellation de « gestion à la chaîne ».



Vers une stratégie d'industrialisation généralisée

Les initiatives émergent au sein des entreprises par des équipes ou des individus souhaitant bénéficier des intérêts que nous explorons dans ce chapitre. Ils peuvent être liés à la conteneurisation pour les équipes de développement, au déploiement automatisé pour les DevOps et à l'orchestration pour les Ops dans le cadre d'un projet pilote ou d'un POC.

L'enjeu pour les **DSIs** est donc de pouvoir **capitaliser sur des initiatives portées par leurs équipes**, avec un niveau de maturité variable et des retours d'expériences parfois divergents.

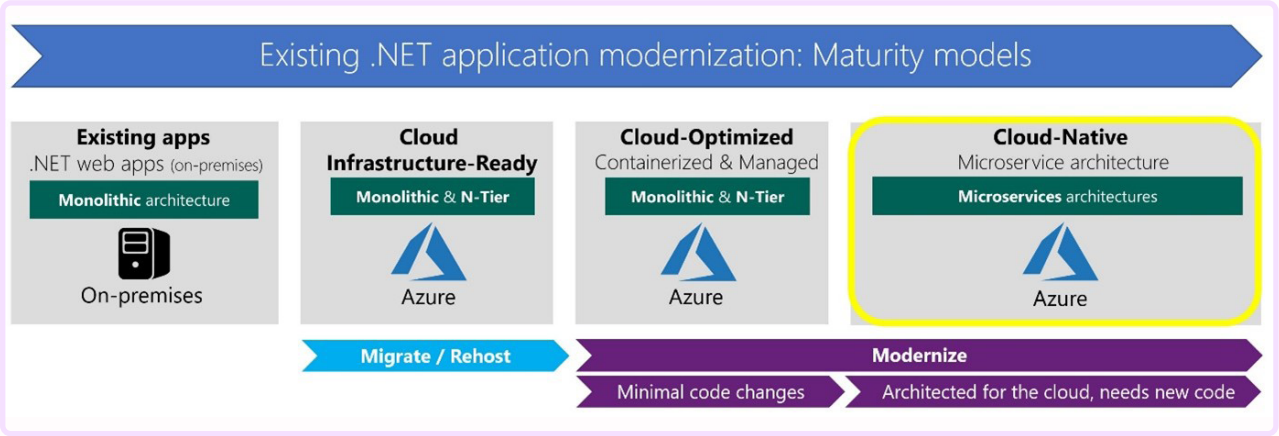
Comment des initiatives prises par des collaborateurs et liées à un contexte bien défini (une équipe, une application, un POC…) peuvent-elles venir soutenir une démarche « entreprise at scale » globale d'industrialisation ?

Parmi l'ensemble des choix possibles, il est important de **capitaliser sur les retours d'expériences et use cases du secteur** de son entreprise pour la mise en place d'une démarche fiable, avec la prise en compte des impacts au-delà de la pure évolution des technologies et des environnements. Comme énoncé plus haut, le soutien d'une transformation de l'organisation est nécessaire pour la mise en place de processus efficaces et pérennes, et ainsi bénéficier rapidement des ROIs recherchés. Les refontes et évolutions successives, les changements d'environnements technologiques, de trajectoires ou des évolutions budgétaires sont un environnement bien connu pour les entreprises.

Pour garantir le succès de ce plan de transformation, il est nécessaire de mettre en place une stratégie claire et surtout une vision éclairée des ROIs recherchés.

Accélérer le développement d'applications Cloud native et la modernisation des applications

Dans le cadre d'une stratégie de modernisation des applications, les équipes pourront capitaliser sur l'ensemble des avantages d'un socle d'orchestration pour faciliter le déploiement d'applications existantes, tout en soutenant le scaling des applications Cloud native.



Source : Documentation Microsoft

Attirer les talents

En dehors de l'évolution des stacks techniques, Kubernetes et la conteneurisation attirent les talents pouvant apporter un retour d'expérience de valeur aux entreprises. Mais cet attrait doit être contrebalancé avec le besoin de formation des équipes et l'accompagnement dans la durée par la pratique.

Bénéficier des avantages d'un Kubernetes managé

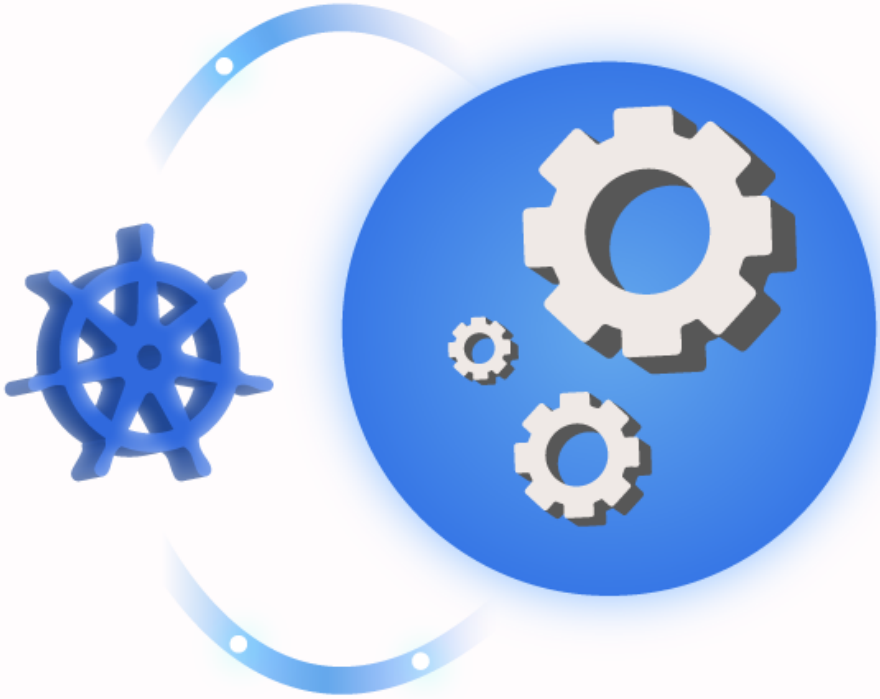
L'avantage certain d'un Kubernetes managé est lié à la délégation de la gestion des serveurs. Le Cloud provider prend à sa charge une part non-négligeable de la gestion technique du cluster Kubernetes. Le consommateur garde à sa charge la gestion des workers et représente aussi l'unique coût de ce service.

Associé à une stratégie Cloud adaptée au contexte client, il présente de nombreux avantages pour nos clients :

- Facilité d'opérations : un socle technologique ;
- Facilité de migration d'applications existantes, plutôt que vers un Kubernetes non-managé, qui peut apporter des contraintes dans la construction du socle ;
- Facilité de scaling horizontal pour des architectures micro-services qui peuvent avoir un environnement étendu et le déploiement via des policies.

En conclusion, sa mise en place permet de **favoriser la production de valeur** pour les équipes Ops, en **déléguant** certaines actions de management au service.

KUBERNETES AU SERVICE DE VOTRE STRATÉGIE APPLICATIVE



Aujourd'hui, Kubernetes est sur toutes les lèvres et dans toutes les stratégies applicatives des plus grands groupes. Cependant, même si sa notoriété et ses bénéfices sont nombreux, Kubernetes est - et doit rester - un outil technologique au service de vos applications et non l'inverse ! En effet, **le choix de Kubernetes doit être fait en connaissance de cause et non par défaut**.

Cassons donc dès à présent certains mythes :

- Micro-service ≠ conteneurisation
- Conteneur ≠ Kubernetes
- Kubernetes ≠ CaaS
- Kubernetes des Cloud providers (AKS, EKS, GKE…) ≠ PaaS

Posons ensemble une **définition d'application Cloud native**. Les applications Cloud native sont créées et optimisées pour la mise à l'échelle et les performances Cloud. Elles sont basées sur des architectures de type micro-services, utilisent des services managés et tirent parti d'une livraison continue pour gagner en fiabilité et accélérer la commercialisation.

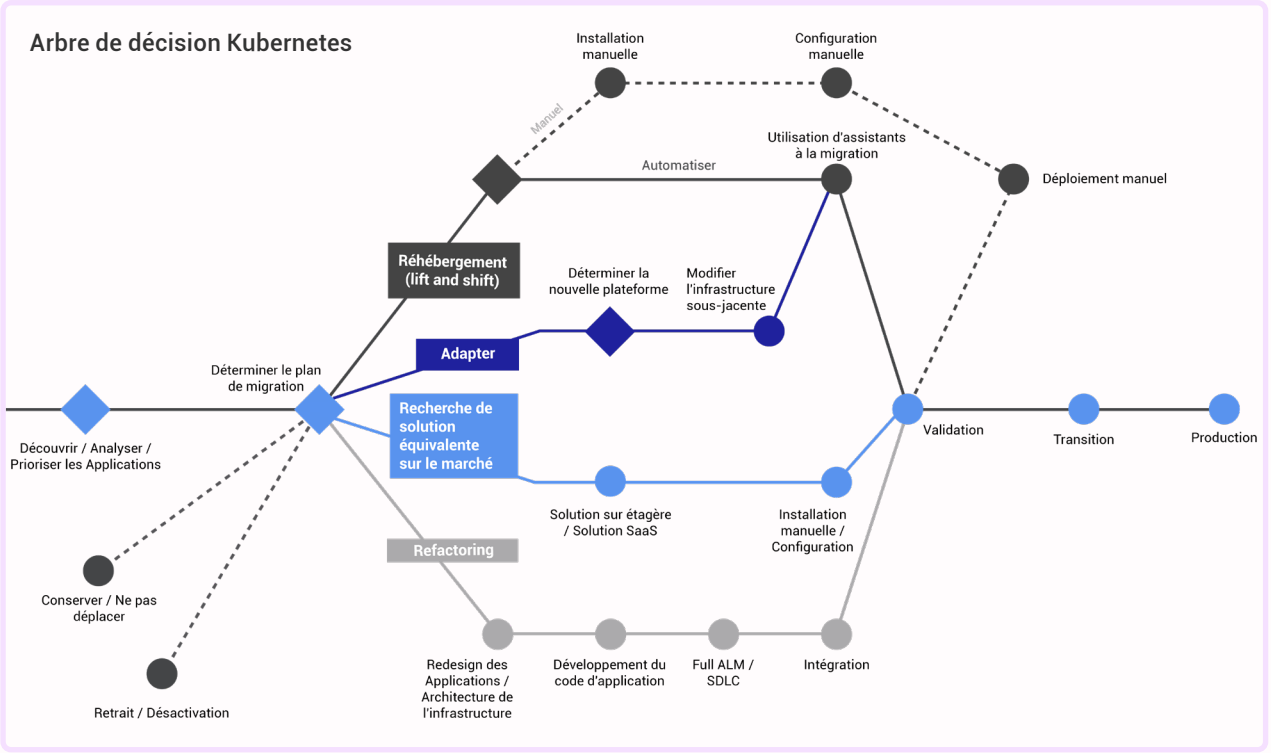
A ce titre, Kubernetes peut être envisagé mais ne répondra pas à tous les usages et se doit d'être contextualisé ! Autrement dit, **Kubernetes seul n'est rien**. Il vient avec son écosystème et doit être ancré dans une réalité : les services adjacents du Cloud provider.

Kubernetes, une plateforme parmi les autres

Kubernetes n'est qu'une plateforme applicative au service des applications. On arrive à ce choix non pas parce que c'est le sujet « Tech-Trendy » du moment mais bien parce qu'il **correspond à un besoin**.

Avant de décider si Kubernetes sera le socle de notre future plateforme applicative dans Azure, il convient de déterminer si c'est effectivement un bon candidat.

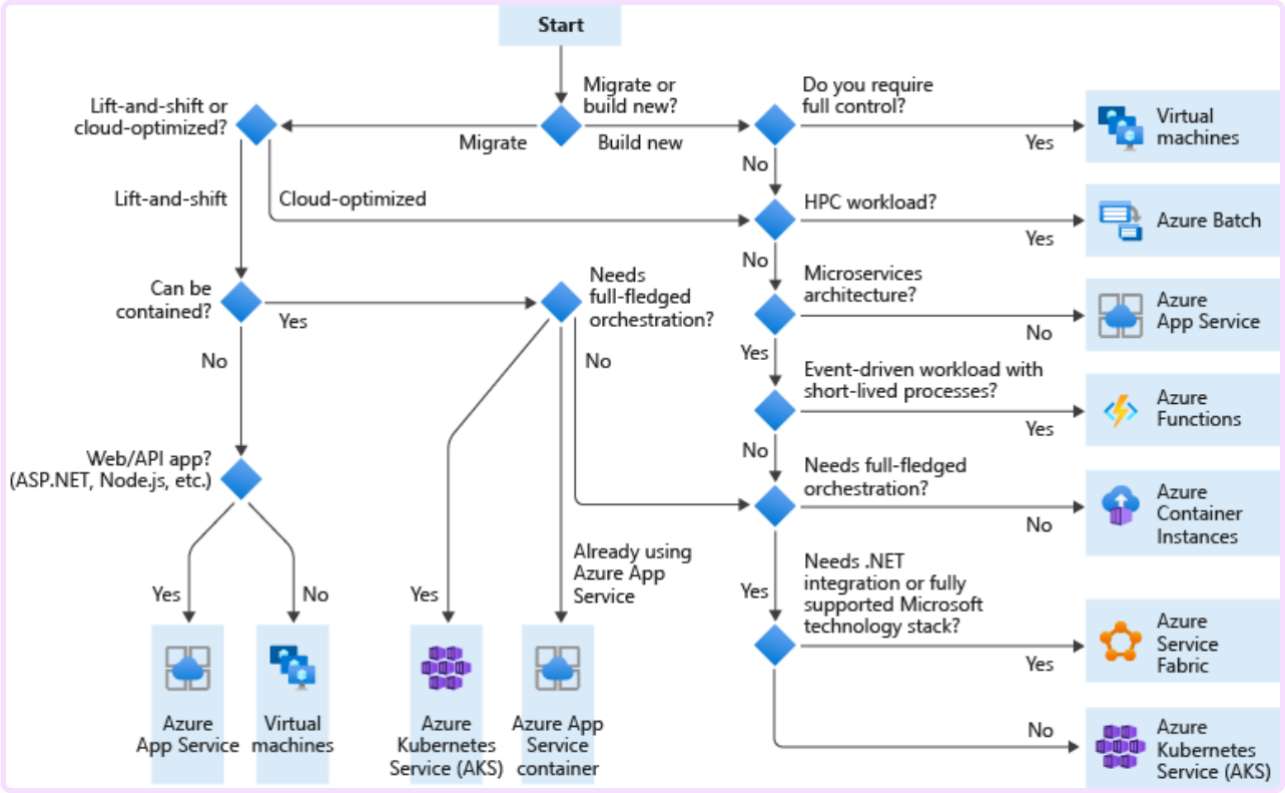
Pour évaluer cela, une étape de rationalisation doit être opérée afin de déterminer un arbre de décision concret.



La méthode des **5R** de la rationalisation Cloud est alors applicable :

- **Réhébergement (rehost) :** Également appelé migration lift-and-shift, le réhébergement déplace une application vers le Cloud en apportant des modifications minimales à l'architecture globale.
- **Refactorisation (refactor) :** Il est judicieux de refactoriser légèrement une application pour qu'elle corresponde à un modèle PaaS (remplacement de sa base de données, exposition via une API web, etc.). La refactorisation peut également s'appliquer au développement même de l'application, dans lequel du code est refactorisé pour permettre de répondre à de nouveaux besoins.
- **Réarchitecture (rearchitect) :** Certaines applications vieillissantes ne sont pas compatibles avec le Cloud en raison de décisions prises lors de leur conception. Dans ce cas, l'application doit subir une réarchitecture avant d'être transformée.
- **Reconstruction (rebuild) :** Dans certains scénarios, les efforts nécessaires à la réarchitecture d'une application peuvent être trop importants pour justifier tout investissement supplémentaire. C'est particulièrement vrai pour les applications qui répondaient autrefois aux besoins de l'entreprise, mais qui ne sont plus prises en charge ou ne sont plus alignées sur les processus métier actuels. Dans ce cas, une nouvelle application est créée pour s'aligner sur l'approche Cloud native by design.
- **Remplacement (replace) :** Parfois, les applications SaaS peuvent fournir toutes les fonctionnalités nécessaires à l'application hébergée. Dans ces scénarios, une charge de travail peut être programmée pour un remplacement futur, ce qui évite tout effort de transformation. Les services de messagerie comme Microsoft Exchange Server ont été d'excellents candidats pour ce type d'approche avec l'offre Microsoft Office 365.

Ces stratégies permettent de définir certains drivers essentiels à la construction de notre arbre de décision quant à notre future plateforme applicative. En effet, si on confronte cette rationalisation au choix d'une plateforme applicative sur Azure, nous obtenons l'arbre de décision suivant :



Source : Choose an Azure compute service for your application

En synthèse, nous avons ci-dessus un arbre de décision reprenant des critères pour nous aider à choisir la future plateforme de nos applications dans Azure. AKS (Azure Kubernetes Services) est loin d'être la seule cible. Le résultat est donc un **point de départ** à prendre en considération. Il reste ensuite à effectuer une évaluation plus détaillée du service et de l'application pour déterminer quelle plateforme est la plus adaptée.

Rappelons que Kubernetes est un **orchestrateur de conteneurs** : de ce fait, seules les applications conteneurisables sont envisageables. Ensuite, comme on peut le voir sur notre arbre de décision, un conteneur sur Azure peut s'exécuter sur différentes plateformes et ces dernières peuvent coexister sur une même architecture en fonction des besoins.

Nous pouvons donc déjà fixer quelques candidats types pour Kubernetes :

- Les applications tournant déjà sur Kubernetes – lift & shift ;
- Les nouvelles applications que nous développons selon les principes « Cloud native » ;
- Les applications conteneurisées dont les conteneurs doivent être orchestrés par un orchestrateur de conteneurs – refactor/rebuild.

Pour synthétiser, le tableau ci-dessous devrait vous aider à orienter vos choix en fonction de votre besoin.

SI VOUS VOULEZ...	... UTILISEZ
Simplifier le déploiement, la gestion et les opérations de Kubernetes	Azure Kubernetes Services (AKS)
Créer rapidement des applications Cloud performantes pour le web et les appareils mobiles	App Service
Exécuter facilement des conteneurs sur Azure sans gestion de serveurs	Container Instances
Planifier les tâches et la gestion des calculs à l'échelle du Cloud	Batch
Développer des micro-services et orchestrer des conteneurs sur Windows ou Linux	Service Fabric ou Azure Kubernetes Services (AKS)
Stocker et gérer des images de conteneur sur tous les types de déploiement Azure	Container Registry
Exécuter des clusters OpenShift complètement managés, fournis conjointement avec Red Hat	Azure Red Hat OpenShift

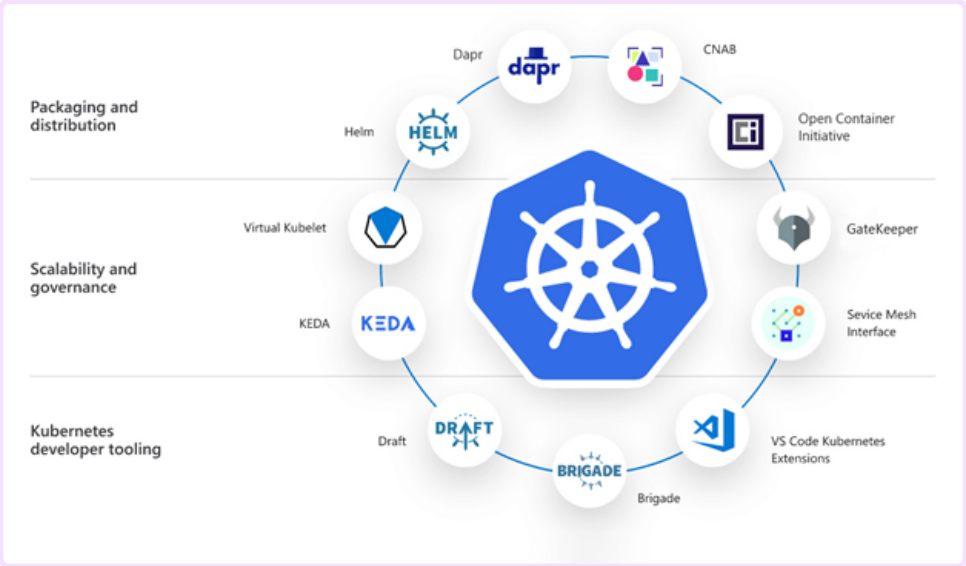
Si votre choix de Kubernetes se confirme, une question majeure va alors se poser : lors de la création d'une nouvelle application, est-ce que je veux m'investir dans l'écosystème Kubernetes ? En effet, Kubernetes (AKS, GKE, EKS…) seul ne sera pas suffisant. Il faudra s'appuyer sur un écosystème riche afin d'avoir une couverture complète de votre application avec de multiples domaines à couvrir :

- Sécurisation : Qualys, TwistLock…
- Déploiement : Helm, Flux, Argo-CD, Github Actions
- Observabilité : Elastic search, Prometeus + Graphana, Splunk…
- Gestion des secrets : Cert-Manager, Hashicorp Vault, Azure Secrets Store CSI Driver
- Etc.

Le temps de montée en compétences sur la plateforme Kubernetes peut sembler long afin de parfaitement maîtriser toute la chaîne et ainsi avoir une vraie plateforme robuste, scalable et réplicable pour héberger les applications cibles. Ainsi, pour héberger un front-web ou des APIs. Est-ce réellement la bonne stratégie ?

L'écosystème Kubernetes

Un des principaux facteurs de succès d'une plateforme réside dans la **taille de son écosystème**. C'est **une des grandes forces de Kubernetes**. Né dans le monde de l'open-source (projet hébergé par la Cloud Native Computing Foundation), cet outil a su rallier tous les acteurs autour de lui pour proposer un écosystème très riche. Microsoft a massivement investi dans cet écosystème, initiant ou participant à de nombreux projets en relation avec Kubernetes. Cet investissement de Microsoft a permis de soutenir de nombreux projets, contribuant ainsi à l'ouverture de l'écosystème de la plateforme Kubernetes.



Source : Choose an Azure compute service for your application

Il faut donc penser « Kubernetes et son écosystème » ! Ainsi, AKS ne doit pas être imaginé seul sur Azure mais bien accompagné de l'ensemble de l'écosystème afin d'avoir une chaîne complète et maîtrisée, du développement jusqu'au maintien en condition opérationnelle.

Comme vous le verrez dans le chapitre suivant, l'écosystème est vaste. Pour s'en donner une idée, il suffit de consulter le panorama proposé par la Cloud Native Computing Foundation (CNCF). Adopter AKS, c'est en fait adopter un **écosystème complet et complexe** dont le cluster Kubernetes n'est qu'une petite partie. Nous allons devoir construire notre plateforme en utilisant un certain nombre de ces composants. Les prochains chapitres de ce livre blanc vous aideront à vous focaliser sur l'essentiel.

Le rêve d'être « Cloud agnostique »

« La plateforme Kubernetes est agnostique et offre donc une solution au Vendor-Locking » : c'est une **illusion** !

Comme nous venons de le voir, Kubernetes seul n'est rien : il doit vivre avec son écosystème et la plateforme sur laquelle il tourne. En effet, dès lors qu'on parle d'une application dans un cluster Kubernetes, il faut penser à l'intégration des secrets et certificats, à un référentiel de données, à un réseau, à des mécanismes de filtrage et d'intégration propres au Cloud provider… Même si nous déportons la plupart de ces éléments vers d'autres plateformes (un vault d'Hashicorp, une base de données conteneurisée…), nous restons dépendants de ces nouveaux vendeurs que nous venons d'introduire et l'intégration même à la plateforme reste bel et bien présente.

Cela met à mal également le rêve de réversibilité : utiliser Kubernetes seul pour assurer de la réversibilité est illusoire. Il manque un outil de fédération/management faisant abstraction de la plateforme. Une solution comme Azure ARC se positionne pour unifier la gestion des clusters Kubernetes, que ceux-ci soient hébergés dans vos datacenters ou même chez d'autres Cloud providers. Finalement, ne serait-il pas plus simple d'abstraire l'infrastructure sous-jacente afin de bénéficier de l'agnostique et de la réversibilité ? Voulons-nous que nos applications soient totalement agnostiques ou uniquement portables d'un socle technique à un autre ? Ce n'est pas la même chose.

> *La plateforme Kubernetes est agnostique et offre donc une solution au Vendor-Locking* **: c'est une illusion !**

En effet, aujourd'hui, les développeurs sont à l'aise avec les architectures d'applications web + base de données (par exemple les conceptions classiques à 3 niveaux) mais pas avec les architectures de type micro-services qui sont intrinsèquement distribuées. Les développeurs veulent se concentrer sur la logique d'entreprise tout en s'appuyant sur les plateformes pour que leurs applications soient résilientes, maintenables, élastiques et tous autres attributs des architectures Cloud native. C'est là que Dapr entre en jeu.

Dapr offre des building blocks pour la construction d'applications de micro-services, indépendants, open-source, qui permettent de construire des applications portables avec le langage de programmation et le Cloud provider de votre choix. Chaque bloc est complètement indépendant et vous pouvez en utiliser un, certains ou tous dans votre application. Cela permet de s'affranchir de la cible (que ce soit l'infrastructure de compute mais également la base de données utilisée...) afin de se concentrer la construction même de l'application, conteneurisée ou non.

L'essentiel à retenir

- **Kubernetes est au service d'une stratégie applicative** : Kubernetes n'est pas LA réponse. Il permet d'assoir une certaine stratégie applicative basée sur l'**orchestration de conteneurs** dans un monde Cloud natif.

- **Kubernetes n'est pas seul** : Kubernetes doit être vu comme un écosystème complet et complexe et non uniquement comme un cluster. Il nécessite ainsi une expertise fine et possiblement complexe avant d'être mis en place à l'échelle d'une entreprise.

- **Kubernetes ne répond pas « by design » aux exigences de sécurité et de réversibilité…** : par défaut, un cluster Kubernetes est très permissif (tout est ouvert, tout est accessible, tout est en clair...), il ne permet pas d'être Cloud agnostique ni vendor-free.

Avant de se lancer dans l'aventure, il conviendra donc de **définir la cible applicative à atteindre**, que ce soit pour nos applications Legacy que nous allons porter dans le Cloud mais aussi pour les nouvelles applications.

KUBERNETES : UN ÉCOSYSTÈME À CONSTRUIRE

Gartner estime qu'en 2022, 75% des organisations exécuteront des conteneurs en production.

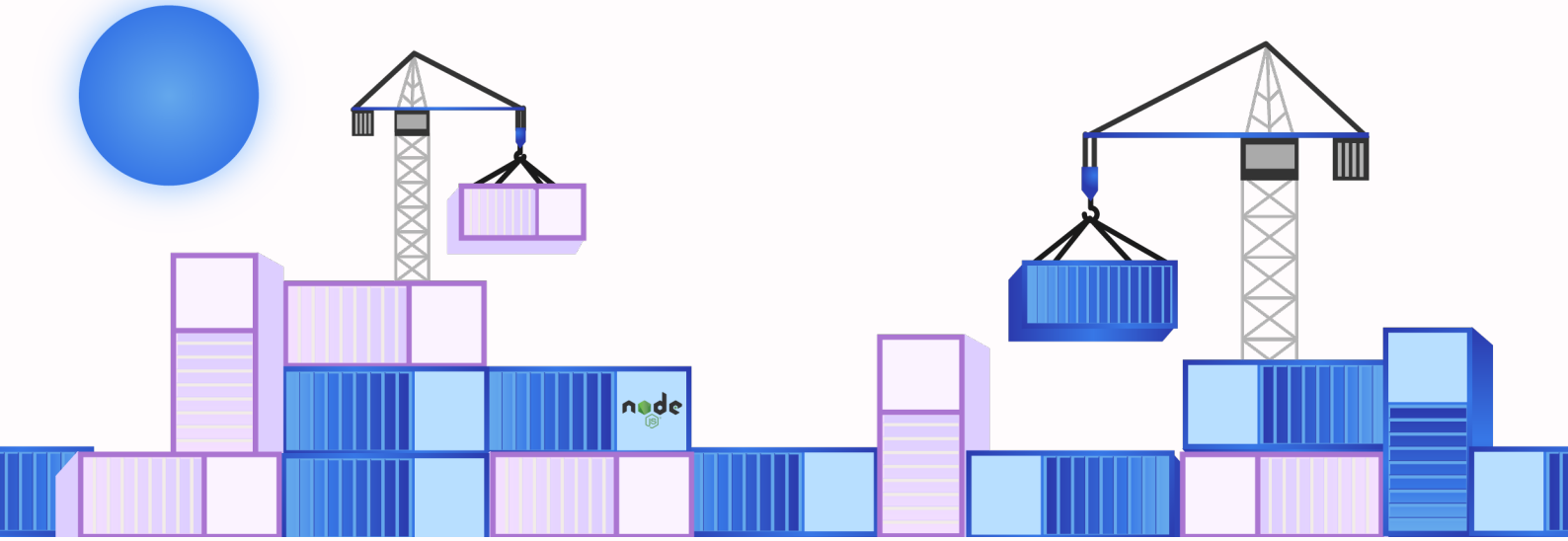
Cela fait maintenant quelques années que l'on parle de micro-services, de solutions d'orchestrations pour conteneurs et de comment ces outils peuvent changer les projets de développements informatiques. L'objectif aujourd'hui n'est pas de revenir sur certaines notions fondamentales, mais de **donner aux décideurs IT les bonnes questions à se poser et des pistes de réflexion pour intégrer au mieux la solution Kubernetes dans le SI de leur entreprise**.

En effet, d'un point de vue technique, Kubernetes est complexe mais les offres managées telles qu'AKS le sont moins. C'est pour cela que nous vous invitons à prendre le problème dans le bon sens pour réellement vous poser les bonnes questions :

- Quel projet migrer en premier ?
- Quel écosystème privilégier en termes d'outillage ?
- Sous quel prisme aborder cette implémentation ?
- Comment maximiser les chances d'aller en production rapidement ?

Nous ne répondrons pas forcément à toutes les questions, mais il est important d'avoir en tête ces différentes pistes de réflexion. Nous évoquerons donc les points suivants :

- Quelle offre choisir parmi les différentes possibilités existantes ?
- Quelle est la maturité des outils de la communauté ? Nous apporterons une première vision des solutions à mettre en place.
- Enfin, nous évoquerons plusieurs retours d'expériences sur différents sujets.



Quelle offre choisir ?

« Les organisations sous-estiment souvent l'effort pour opérer des conteneurs en production » (Gartner)

La première question à se poser est de savoir quel type de service d'orchestration vous souhaitez pour votre entreprise.
En effet, il existe **3 grandes familles de solutions** avec chacune leurs avantages et inconvénients.

Installer Kubernetes sur un environnement on-premise

La première solution est d'**installer Kubernetes (ou K8s)** sur un environnement on-premise sur des machines physiques/virtuelles ou sur le Cloud en mode IaaS. Cela permet d'avoir un contrôle complet sur la stack technique mais implique plus de responsabilités.
A noter : l'installation de K8S est longue, fastidieuse et nécessite de solides compétences (par exemple, regarder comment installer K8s from scratch : https://github.com/mmumshad/kubernetes-the-hard-way).

De plus, il faut penser à avoir une équipe pour administrer les machines et un plan de mise à niveau continue. En effet, l'écosystème évolue constamment et une équipe dédiée est nécessaire pour suivre les dernières releases, tester les outils et surtout planifier les montées de versions pour assurer le support. Dans le cas des offres managées, c'est le Cloud Solution Provider (CSP) qui vous impose son rythme.

Installer Kubernetes via un fournisseur de Cloud public

La deuxième possibilité, qui est clairement la tendance actuelle, consiste à passer par un Cloud public provider :

- Amazon avec la solution **Amazon EKS** ;
- Google avec GKE (**Google Kubernetes Engine**);
- Ou Microsoft avec **AKS**.

Le gros avantage ici est qu'**une grande partie de services sont managés pour vous** : les **nœuds** (master et slaves) ainsi que le **control plan API**, qui constitue la partie centrale de Kubernetes. Sans elle, le cluster est tout simplement inopérant.

Attention : chacun des Cloud providers possède ses propres spécificités. Par exemple, l'accès aux Secrets externalisés n'est pas identique dans Azure que pour Google Cloud Platfom (GCP). Autrement dit : l'agnosticité apportée par Kubernetes est limitée dès lors que l'on interagit avec des ressources fournies par le Cloud provider. C'est pourquoi il faut être vigilant et choisir des interfaces de type CSI, ou se baser sur des solutions open-source par exemple. De manière simpliste, on ne peut pas transposer « As-Is » un cluster AKS vers GKE de manière simple.

Utiliser RedHat OpenShift (un exemple d'une solution sur étagère)

Enfin, dernière situation : l'entreprise souhaite limiter au maximum le travail d'administration ou avoir une réelle solution déployable facilement. On achète un écosystème sur **étagère** et on réutilise au maximum les composants fournis par le service. Il faut alors regarder vers RedHat OpenShift. L'interface d'administration y est simplifiée : tout est fait pour que l'application soit rapidement et facilement déployable, afin de bénéficier d'un time to market intéressant. Cependant, il faut garder en tête qu'ici l'entreprise aura une **adhérence forte avec l'éditeur** : il ne sera pas possible de sortir simplement de cette solution pour migrer vers l'un des deux scenarios précédents.
Pour vous faire un choix en résumé, voici un rapide récapitulatif :

Solution	Full on premise	Managée avec un CSP	Solution sur étagère
Avantages	Contrôle et liberté complets Sécurité renforcée	Risques limités Coûts plus faibles Equipes IT plus concentrées sur apporter des solutions business	Peu de compétences K8S à avoir Equipes IT dirigées à 100% pour apporter de la valeur business Time to market réduit
Inconvénients	Nécessité d'administration étendue Compétences humaines nécessaires Coûts importants	Nécessité d'avoir une équipe de support dédiée et transverse Suivi des mises à jour nécessaire	Dépendance forte Moins de liberté

Mesure de la maturité des outils de la communauté

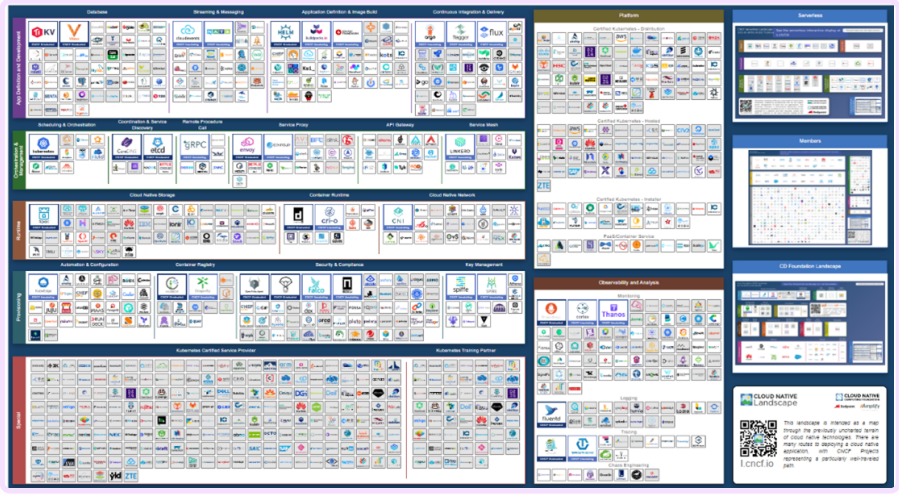
Une fois l'offre choisie, l'étape suivante dépendra du type de population :

- **Pour les décideurs/DevSecOps** : choisir les outils à intégrer tout autour de votre orchestrateur.
- **Pour les développeurs/Devops** : se transformer pour savoir développer avec l'approche « micro-services ».

- **Pour les décideurs / DevSecOps** : tout d'abord, il importe d'adopter les bons réflexes pour les décideurs & DevOps, garants de la bonne intégration dans le SI. En effet aujourd'hui, **intégrer K8s on-premise ou via des Cloud providers sans aucun outil tiers est une utopie**. Quelle solution choisir pour la partie CI/CD ? Comment appliquer la gouvernance et les policies dictées par le service sécurité ou gérer les coûts ? Commet garder un certain niveau de contrôle sur les ressources mises à disposition des équipes de développement ? Fournir un environnement (cluster) Kubernetes à une ou plusieurs équipes de développement comporte des **avantages** (liberté, autonomie) mais **il faut garder un certain niveau de contrôle** (on parle de cluster multi-tenant).

Très vite, vous allez tomber sur la principale communauté : la Cloud Native Computing Foundation (CNCF), qui est un projet de la communauté Linux. C'est ici que se trouvent l'ensemble des produits que l'on peut implémenter selon plusieurs degrés de maturité : charge à vous de les choisir, de les tester ou de les ignorer. Le monde est immense : on s'y perd et cette carte vous permet de vous repérer. On dit souvent que **Kubernetes est un Cloud dans le Cloud** : sans repère, difficile de s'en sortir !

Nous vous recommandons vivement de jeter un œil à la vision globale. Toutes les solutions y sont regroupées par thème : développement d'application, base de données, intégration logicielle, services d'infrastructure (proxy, mesh, sécurité).



Source : CNCF Landscape

Vous pouvez cliquer sur chaque solution afin d'obtenir plus d'informations telles que :

- Description succincte ;
- Langage sur lequel repose la solution ;
- Activité GitHub (pour savoir si la communauté est active) ;
- Maturité au sein de la CNCF : Graduated, incubating ou Sandbox.

Bref, avant de faire son choix, il est indispensable de se renseigner et ce **portail est la porte d'entrée**.

- Pour la communauté des **Développeurs** : ils doivent absolument s'approprier les 12 principes édictés par « The twelve-Factor App ». Développer des solutions avec une approche micro-service est différent de ce que l'on a pu avoir par rapport au mode on-premise ou IaaS : on pense plus serverless, microcomposants, scalabilité et résilience. Ce document oriente les équipes sur les thématiques telles que la gestion de la configuration, la gestion du contrôle de code source et des dépendances, les services externes, la robustesse, la gestion des environnements. Bref, on peut avoir l'environnement K8s le plus optimisé et qui réponde au besoin, **mais si on ne respecte aucun de ces facteurs, le projet est voué à l'échec**.

Voici les principes recommandés pour adopter un environnement Kubernetes dans le SI :

- Choisir un projet **peu critique** pour commencer ;
- **Identifier l'équipe** en fonction des appétences ;
- Se mettre d'accord sur un **planning ambitieux mais réaliste** ;
- Choisir les **outils** à intégrer qui correspondent au besoin : inutile de trop en mettre ;
- Procéder par **itération**.

La solution technique doit toujours être au service d'un besoin business et non l'inverse. C'est fondamental.

Retours d'expériences sur la mise en place de Kubernetes

Partageons à présent quelques retours d'expériences en termes d'approche et d'outillage.

Il faut avoir une approche par **itération**, avec des échéances de 2 à 4 semaines dans lesquelles les objectifs sont clairs et **partagés par un maximum d'acteurs de l'entreprise** : développeurs, managers, DevOps, sécurité, infra... Plus il y aura du monde, mieux ce sera. De plus, **bénéficier d'un engagement fort de la part d'un sponsor haut placé est primordial**.

DevOps

La partie relative à l'Infrastructure as Code (IaC) est importante. C'est pour cela que nous avons utilisé Terraform pour la création de ressources Azure, associé à Jenkins ou Azure Devops. Pour ce qui concerne les Registry, le choix logique lorsqu'on est dans l'écosystème Microsoft est d'utiliser Azure Container Registry (ACR), mais on peut aussi utiliser Harbor qui est indépendant du Cloud provider. Il a aussi l'avantage de stocker des charts Helm et réaliser des scans de sécurité. Enfin, pour K8s, ArgoCD a été déployé : à chaque fois qu'un commit sur un manifeste YAML a lieu, la configuration est automatiquement déployée sur le cluster (le chapitre suivant sur le CI/CD aborde cette thématique). Par exemple, créer des pipelines Jenkins pour réaliser le provisionning réseau et un autre pour la création de cluster AKS. Ce que nous avons fait chez un client est d'adopter l'approche Blue/Green d'un point de vue infrastructure pour les montées de version de Kubernetes. Tous les 3 à 5 mois, nous mettons à leur disposition un nouveau cluster dans lequel les équipes de développements doivent migrer les applicatifs. Un accompagnement est nécessaire lors de l'onboarding pour leur expliquer les bonnes pratiques et les éléments du support, leur présenter l'architecture, etc.

Sécurité

Parler de la sécurité dans K8s est un vaste sujet. En effet, la sécurité concerne l'authentification RBAC pour accéder aux API masters, le scan des images dans les registries, le principe de least privilege dans les pods, la configuration de Open Policy Agent (OPA), la bonne configuration des Ingress controlers ou encore le stockage des secrets/certificats/clés dans les services managés tels qu'Azure Key Vault... C'est pourquoi nous recommandons d'**intégrer les équipes de sécurité au plus tôt dans le processus de conception** pour implémenter les bonnes pratiques.

FinOps

La refacturation par équipe des ressources K8s est un sujet à part entière. En effet, si chaque projet possède son propre cluster, le calcul est simple. Mais quid des clusters multi-projets ? Doit-on refacturer par nœud ? Par namespace ? Et comment faire pour les ressources partagées telles que le réseau, les comptes de stockage ou les base de données ? Pour cela, nous recommandons Kubecost : cet article vous sera d'une grande aide pour alimenter votre réflexion.

Policies

Avant de mettre à disposition les clusters aux équipes projets, un certain nombre de composants techniques sont installés dans des namespaces dédiés. En effet, voici une liste (non exhaustive) d'éléments :

- Mettre des *webhooks* pour contrôler que les images proviennent de *registry* validés par l'entreprise ;

- Installer Open Policy Agent pour intégrer vos propres règles avant de créer des objets Kubernetes (sur la gestion du réseau, des droits utilisateurs, des droits OS, etc.). Cet outil est désormais parfaitement intégré à la plateforme Microsoft par exemple ;

- Kiosk est également une bonne option pour une meilleure gestion de la sécurité de vos *namespaces* ;

- Enfin bien sûr, pensez à mettre des policies propres à votre Cloud provider pour éviter par exemple que les équipes puissent modifier la taille des nodes pools ou s'octroyer plus de droits que ceux fournis par l'équipe sécurité. Sur Azure, l'installation de *blueprint* contenant des Azure policies est un bon choix.

Accompagnement des équipes

Un accompagnement des équipes de développement est primordial : si vous voulez faciliter l'adoption de K8s, il faut **impliquer ces équipes** et leur montrer que les développements seront facilités et sécurisés. Par exemple, nous conseillons d'organiser toutes les deux semaines des sessions d'onboarding au cours desquelles vous présenterez l'architecture, ce que peuvent faire les équipes lorsqu'elles auront un cluster à disposition, les éléments techniques qu'elles pourront utiliser (ingress controler, registres privés, synchronisations des secrets/certificats, accès aux bases de données…).

Ne pas oublier également de leur fournir du **support**, de la **documentation** et si besoin, organiser des réunions d'aide adaptées et personnalisables. Fournir un cluster doit donner aux équipes suffisamment de libertés pour fournir de la valeur rapidement (c'est-à-dire du code répondant à un besoin métier) : **c'est là que votre succès interviendra**.

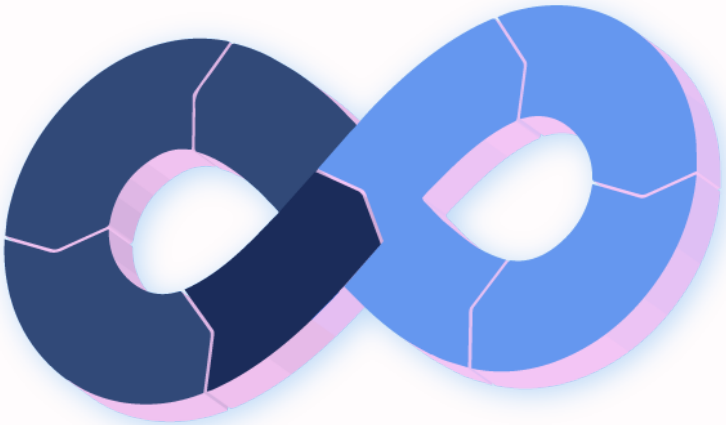
Liens utiles :

gartners 6 best-practices for containers kubernetes

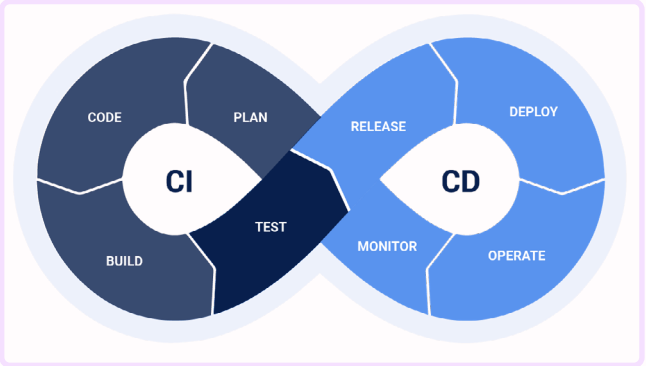
Suivre les activités sur github

05 CI/CD

STRATÉGIE DE CI/CD SUR KUBERNETES



Nous allons à présent aborder les notions d'usine logiciel au sein de l'écosystème Kubernetes. Mais avant de rentrer dans le vif du sujet, il est indispensable d'avoir quelques notions d'industrialisation sur les mises en pratique d'une bonne CI/CD pour booster votre productivité. Commençons donc par quelques définitions.



La **CI** désigne l'intégration continue, à savoir un processus d'automatisation pour les développeurs. Cette intégration continue consiste, pour les développeurs, à apporter régulièrement des modifications au code de leur application, à les tester, puis à les fusionner dans un référentiel partagé. Cette solution permet de fluidifier le processus de développement mais aussi d'éviter les régressions.

La **CD** peut désigner la « distribution continue » ou le « déploiement continu », deux concepts très proches, parfois utilisés de façon interchangeable.

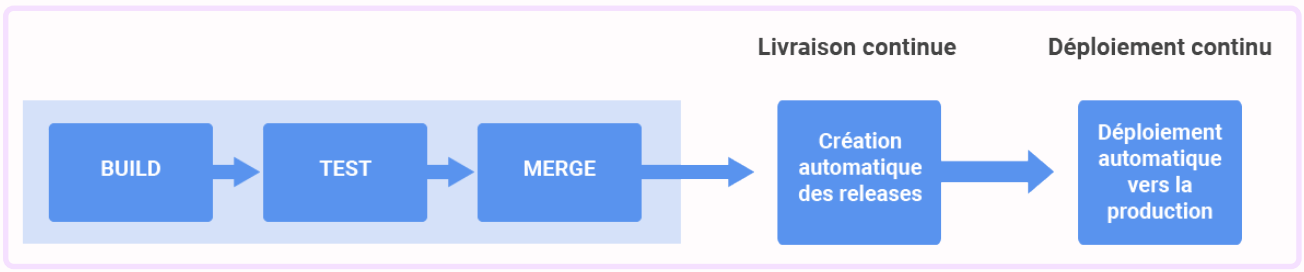
Ce qu'il faut retenir, c'est que :

- La CI permet d'automatiser les développements à des fins qualitatives, pour produire des releases ;
- La CD vous permettra de déployer automatiquement vos releases sur vos environnements de test puis de production.

C'est un bon moyen de **booster votre productivité**.

Dans ce chapitre, nous allons évoquer certains points incontournables de la CI/CD pour Kubernetes :

- Comment **créer et gérer ses clusters Kubernetes** pour héberger ses applications ?
- Comment **déployer ses applications** sur Kubernetes ?



Infrastructure as Code : création et gestion du cluster Kubernetes

L'Infrastructure **as Code (IaC)** est une méthodologie permettant de gérer l'automatisation de votre infrastructure. Tout se trouve dans votre repository Git. Certaines grandes entreprises recréent de toutes pièces leur infrastructure chaque semaine. L'automatisation devient gage de sureté. Face à un *disaster recovery*, vous pourrez plus aisément vous fier à votre IaC. Afin de créer et gérer vos clusters Kubernetes, nous vous recommandons de procéder de cette manière.

L'approche procédurale

Lorsque nous abordons notre infrastructure avec une approche procédurale, cela revient à se dire « Comment mon infrastructure devrait être changée ? ». C'est une méthode itérative dans laquelle vous allez mettre en place vos différents scripts d'automatisation afin de consolider votre infrastructure. L'inconvénient majeur de cette approche est que vous devez continuellement vérifier l'état réel de votre infrastructure. Au fil du temps, cette dernière s'étoffera et vous risquez d'en perdre le contrôle.

Les solutions pour l'approche déclarative

L'approche déclarative consiste à définir l'infrastructure souhaitée (propriétés, ressources nécessaires, dépendances...) et l'outil IaC se chargera de la configuration. Les principaux outils fournis par défaut par les Cloud providers sont :

- **ARM / Bicep** pour Azure ;
- **CloudFormation** pour AWS ;
- **Cloud Deployment Manager** pour GCP ;
- Etc.

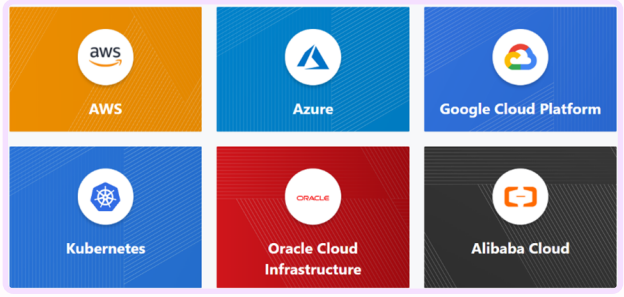
Une autre alternative permet de décrire votre infrastructure pour les Cloud providers connus du marché : il s'agit de **Terraform**. A l'aide du même langage (HCL, pour Hashicorp Configuration Language), on applique le même workflow, quelle que soit la cible :

- **Write** : décrire l'infrastructure souhaitée ;
- **Plan** : vérifier les changements avant son application ;
- **Apply** : provisionnement et mise à jour de l'infrastructure.

On peut utiliser Terraform pour les cas suivants :

- Dans le cadre du **multicloud** : clusters Kubernetes sur deux Cloud providers différents par exemple ;
- Si vous avez un doute sur la cible et qu'un changement est envisagé dans un futur proche : passage d'un Cloud provider vers un autre, par exemple.

Voici les principaux providers disponibles pour Terraform :



Quel que soit votre choix, les fichiers de configuration devront être stockés dans un système de contrôle de source tel que Git.

Le repository, reflet de notre infrastructure

Le fait de décrire l'infrastructure de manière déclarative dans votre repository permet d'avoir une seule source pour la création et la gestion des clusters Kubernetes.
En effet, on évite ainsi d'interagir directement avec l'infrastructure pour toute modification de configurations, tout en conservant un historique des modifications. Ce dernier pourrait même devenir source de documentation.

Adapter notre configuration pour les montées de version de Kubernetes

Sur Kubernetes, les montées de versions sont assez fréquentes. Comme les projets qui gravitent autour sont assez nombreux et évoluent rapidement, il faut suivre non seulement les évolutions de Kubernetes, mais aussi les projets open-source/éditeur qui peuvent être intégrés dans Kubernetes : outils de monitoring/logs, sécurité, network policy, etc.
Si nous revenons à notre mise à jour de version de Kubernetes, il existe plusieurs stratégies de mises à jour :

- La mise à jour *in place* ;
- La création d'un nouveau cluster.

Pour la suite, nous nous focaliserons uniquement sur la création d'un nouveau cluster. Le nouveau cluster sera le futur remplaçant de votre cluster actuel. Vous devez créer un autre cluster avec une version plus récente de Kubernetes. Avec le template de votre cluster Kubernetes, il ne vous reste qu'à adapter vos variables d'environnement et changer la version de Kubernetes au niveau de votre pipeline de CD.

On aura ainsi deux environnements Kubernetes et on pourra mettre en place du Blue/Green :

- L'environnement Blue est la version actuelle de Kubernetes avec le socle applicatif ;
- L'environnement Green correspond à la nouvelle version qui doit être testée et validée avant de pouvoir basculer dessus.

L'approche GitOps

Inventé récemment par Alexis Richardson de Weaveworks, GitOps est un modèle qui a beaucoup gagné en popularité. L'Infrastructure as Code nous accompagne depuis quelques années maintenant, depuis l'invention des Clouds publics où chaque ressource peut être définie comme du code. GitOps est une extension logique de cette idée aux applications exécutées dans Kubernetes. Le mécanisme de déploiement de Kubernetes nous permet de décrire complètement nos applications sous forme de manifestes que nous pouvons versionner dans Git.
La méthodologie GitOps va un peu plus loin sur l'approche déclarative. Les repositories Git deviennent la seule source de vérité. Toute modification manuelle de l'infra via l'interface utilisateur ou ligne de commande est proscrite.
Le repository Git de l'infrastructure doit être dissocié des repository applicatifs. Ils ont en effet chacun leur propre cycle de vie.

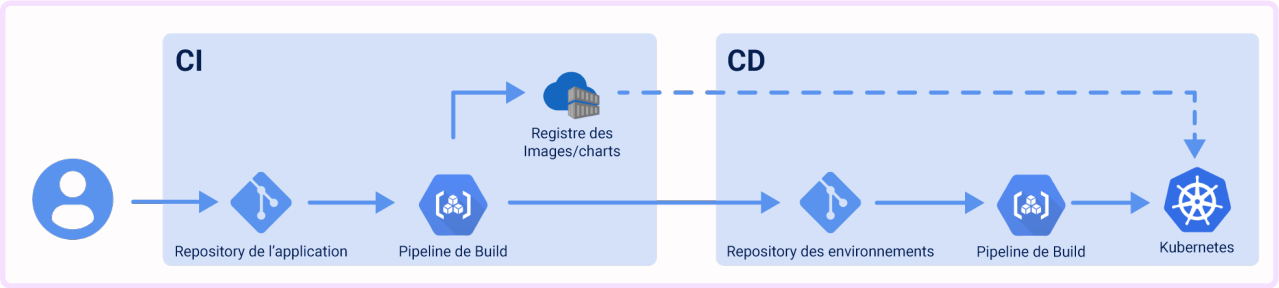
La CI va se dérouler sur le repository de l'application. La CI lancera :

- Le build applicatif ;
- Les tests unitaires ;
- Le build de l'image de l'application ;
- Et enfin, le push dans un registry.

Il existe 2 modes GitOps :

- Le mode Push
- Le mode Pull

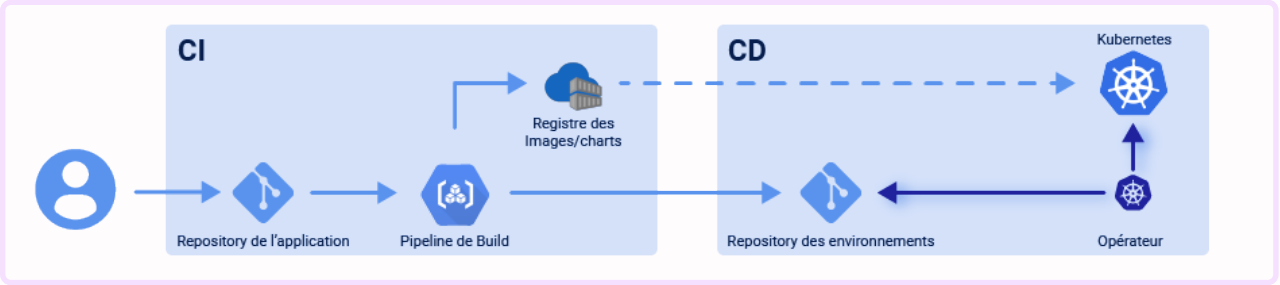
Mode Push de GitOps



La stratégie de déploiement basée sur le Push est mise en œuvre par des outils CI/CD populaires. Le code base de l'application se trouve dans le repository applicatif avec vos manifestes YAML Kubernetes nécessaires pour le déploiement. Chaque fois que le code de l'application est mis à jour, le pipeline de build est déclenché, ce qui crée vos images dans vos registries et effectue une mise à jour du repository de configuration de vos environnements avec vos nouvelles versions.

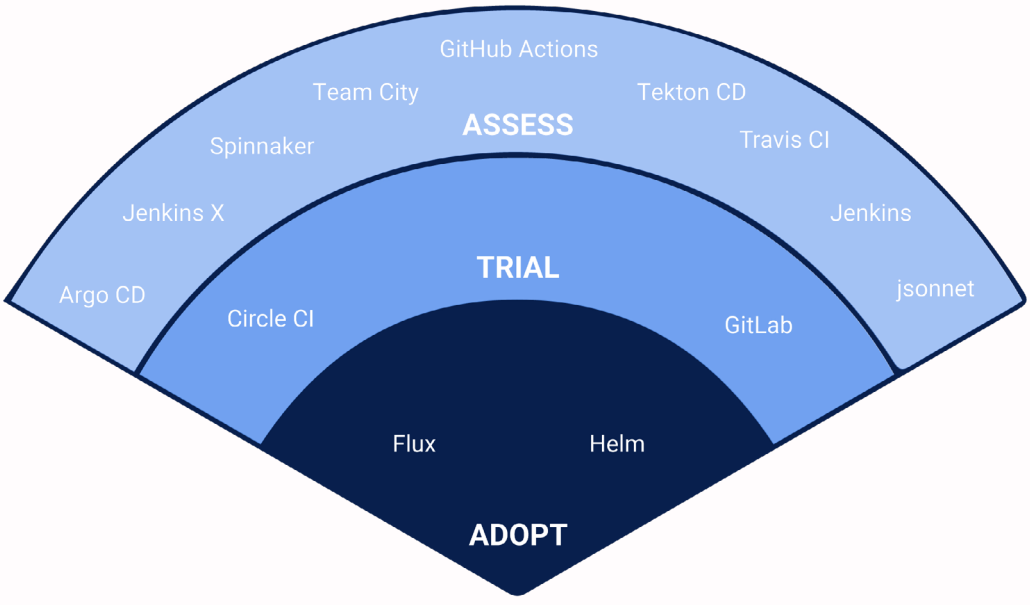
Mode Pull de GitOps

Le mode Pull est l'approche la plus intuitive dans l'écosystème Kubernetes. Nous allons donc nous y intéresser de plus près :
Les pipelines CI/CD traditionnels sont déclenchés par un événement externe, par exemple

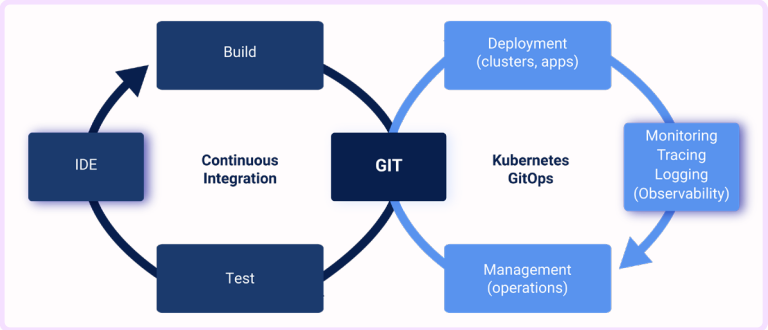


lorsqu'un nouveau code est poussé vers le repository de votre application. Avec l'approche de déploiement basée sur le Pull, une nouvelle notion est introduite : l'opérateur. Ce dernier reprend la responsabilité du pipeline en scrutant le repository de votre environnement. Il compare en permanence l'état souhaité avec l'état réel de l'infrastructure. Pour chaque différence constatée, l'opérateur met à jour l'infrastructure pour qu'elle corresponde au repository de votre environnement. L'opérateur peut aussi surveiller vos registries d'images afin de trouver de nouvelles versions d'images à déployer.

Argo CD, Flux et Jenkins X sont des outils de CD nouvelle génération adoptant la stratégie GitOps Pull.



Sur le plan architectural, GitOps nous permettra de séparer le flux d'intégration continue (CI) d'une application du processus de déploiement, car le processus de déploiement démarrera en fonction des modifications apportées à un dépôt GitOps, et non dans le cadre du processus CI.



Nous allons maintenant aborder le sujet de la CI pour rentrer dans le vif du sujet concernant nos applications à conteneuriser.

Côté applicatif

La CI

Une CI Standard

Les étapes incontournables d'une CI sont :

- Le build ;
- Les tests unitaires ;
- Une publication de votre build dans un artefact.

Dans le cadre d'une intégration continue pour vos applications qui tourneront par la suite dans le cluster AKS, nul besoin de changer vos habitudes en matière de CI. Le seul changement à apporter est de publier en fin de CI vos images Docker dans vos registries privés.

Les images de conteneur

Préparer son image ne se résume pas seulement à conteneuriser son application. Il faut prendre en considération plusieurs éléments :

- Comment versionner mon application ?
- L'image source est-elle sécurisée ?
- Mon application génère-t-elle des logs ?
- Ces logs peuvent-ils me permettre d'avoir des KPIs métiers et de performance ?
- Mon application dispose-t-elle d'un healthcheck permettant de connaître son statut ?

Ségrégation des environnements de production et de développement

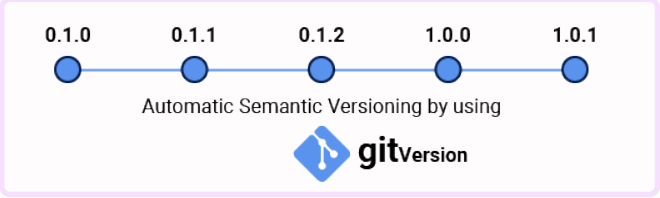
Au fil du temps, suite aux nombreuses features développées par vos équipes, vous vous retrouverez avec une multitude d'image taguées sur vos registries. Des tags très souvent inutiles et qui vous donnent l'impression de nager dans un nuage de tags.
Il devient donc utile de séparer vos travaux de développement avec ce que vous souhaitez réellement déployer en production.

Avoir un registry de développement séparé de la production permet de gagner en clarté.

Vous pouvez très bien pousser vos tags de développement depuis vos branches de feature et develop, et utiliser la branche master pour pousser vos tags de production.

Porter le versionning sur vos branches

En fonction de la taille de vos équipes, vous devez faire un choix sur l'adoption d'un flow Git : Git Flow, Github Flow, master only. Et afin d'automatiser le versionning de votre application, vous pouvez tout simplement laisser cette responsabilité à vos branches. Certains outils tels que Git Version permettent de mettre en place assez rapidement le versionning au sein de vos branches.



Il suffit alors d'utiliser la version de votre branche pour taguer vos images.
Pour le déploiement de vos applications, nous vous recommandons ne pas utiliser le tag latest. Il est préférable de toujours indiquer un tag de version, afin de garder une trace des déploiements pour pouvoir investiguer en cas de problème.

Hygiène de vos registries

Mais alors, qu'allons-nous faire de notre *registry* de développement ? Allons-nous conserver tous les tags ?

Lorsque nous travaillons sur nos branches feature sur Git, nous prenons le temps de supprimer nos branches éphémères, une fois mergées. De la même manière, tous les tags inutiles doivent être supprimés quotidiennement.

Les images utilisées sont-elles sécurisées ?

Nous avons tendance à rechercher l'image la plus petite, la moins consommatrice, au détriment de la sécurité.
Pour se prémunir de ces vulnérabilités, mieux vaut intégrer dans votre registry un outil de scan.
Très complet, le registry Harbor permet d'ajouter via un plugin des outils de scan comme Clair (open source).
Vous pouvez aussi mettre cela en place sur vos Clouds respectifs.

Ne pas oublier le monitoring

Le monitoring est la clé du succès !

Lorsque vous développez vos applications, ne soyez pas avare en log. Mettez en place des logs afin de collecter des informations de performance ainsi que des KPIs métiers pertinents.
Ces customs metrics pourront alors remonter dans vos dashboards métiers et techniques.

La CD

Aujourd'hui, l'un des plus grands défis du développement d'applications Cloud natives est d'accélérer le nombre de vos déploiements. Avec une approche de micro-services, les développeurs travaillent déjà avec et conçoivent des applications entièrement modulaires qui permettent à plusieurs équipes d'écrire et de déployer des modifications simultanément sur une application.
Des déploiements plus courts et plus fréquents offrent plusieurs avantages :

- Une réduction du time to market ;
- Les clients peuvent profiter des fonctionnalités plus rapidement ;
- Des feedbacks clients plus rapides.

Comment déployer sur Kubernetes ?

Kubectl est l'interface en ligne de commande (CLI) de Kubernetes. Il permet de déployer toutes sortes d'objets Kubernetes dans vos clusters.
Le format adopté par Kubernetes est le YAML (Yet Another Markup Language).
A travers ces commandes kubectl, vous pourrez transmettre ces manifestes afin de déployer au mieux vos applications.

Ce qu'il ne faut pas oublier pour vos containers

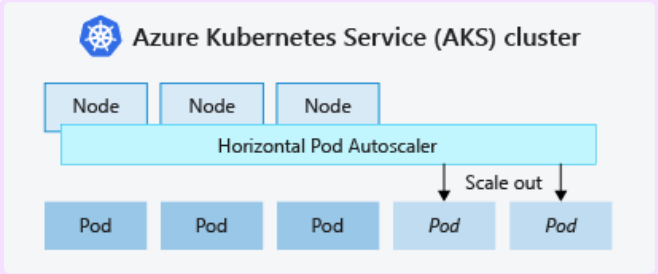
Gardez bien à l'esprit que vos applications sont déployées dans un environnement Kubernetes.
Vous devez prendre en compte un certain nombre de point clés.

Limitation CPU / Mémoire

Certaines mesures devront être prises en considération avant d'embarquer votre application dans un cluster.
Effectuez des **load tests** afin de connaître les limitations de votre application. Ils vous permettront de définir :

- Sa consommation mémoire ;
- Son utilisation CPU (Central Processeur Unit) ;
- Le nombre de requêtes par seconde qu'elle peut gérer.

Ces informations vont être capitales par la suite pour définir la haute disponibilité de vos applications.



Source : Documentation Microsoft

HealthCheck

Tout Load Balancer a besoin de scruter la santé des applications avant de router dessus. Kubernetes ne déroge pas à la règle. Prenez le temps de mettre en place un HealthCheck dans votre application afin que votre cluster puisse interroger un pod sain.

Nous avons pu voir à de nombreuses reprises des applications sans monitoring ni HealthCheck. Le problème récurrent est que votre service (ingress) redirige vers des pods en erreur (applicative) et le diagnostic devient compliqué.

Une fois le Healthcheck mis en place sur vos applications, il faut aussi prendre le temps de configurer les probes de vos containers.

Il existe 3 types de sondes :

- **Startup** probe : vérifie le démarrage initial de vos applications. Elle n'est utilisée qu'une seule fois.

- **Readiness** probe : vérifie si votre application peut répondre au trafic. Elle est utilisée tout le long de la vie de votre conteneur. En cas d'échec, Kubernetes arrêtera d'acheminer le trafic vers votre application (et essayera plus tard).

- **Liveness** probe : vérifie si votre application est en bon état de fonctionnement. Elle est utilisée tout le long de la vie de votre conteneur. En cas d'échec, Kubernetes supposera que votre application est bloquée et la redémarrera.

Montée de version

Les mises à jour de vos clusters Kubernetes ne sont pas sans danger.
Il peut arriver assez souvent que des changements de version des APIs rendent obsolètes les manifestes YAML de vos applications.

Il est toujours préférable d'anticiper la montée de version de vos clusters avec un processus de vérification :

- Mes applications sont-elles déployées correctement ?
- Mes performances sont toujours bonnes ?
- Mes tests d'intégrations se déroulent-ils correctement ?

N'effectuez pas d'upgrade sans vous assurez de ces points ci-dessus.

Deux types de scénarios sont envisagés :

- Lorsque vous effectuez une montée de version sur le même cluster (*in-place*), il est préférable de créer un nouveau cluster de test afin de vérifier la montée de version ;

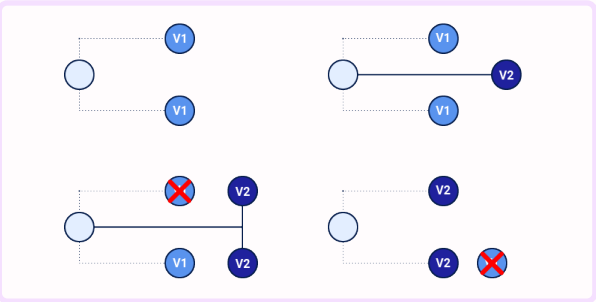
- Lorsque, pour chaque montée de version, vous effectuez la création d'un nouveau cluster (Blue-Green), vous pourrez vérifier en toute sérénité la compatibilité de vos applications avec la nouvelle version hors production.

Stratégie de déploiement

Il existe plusieurs types de stratégies de déploiement dont vous pouvez tirer parti en fonction de votre objectif.

Stratégie Rolling Update

Le Rolling Update - ou déploiement progressif - est le **déploiement standard par défaut** sur Kubernetes. Il fonctionne lentement, un par un, en remplaçant les pods de la version précédente de votre application par des pods de la nouvelle version, sans aucun temps d'arrêt du cluster, permettant aussi un rollback automatique en cas d'échec.

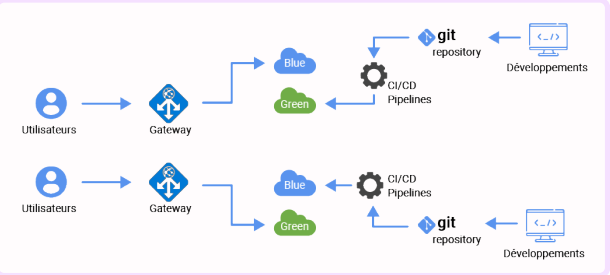


Stratégie Blue Green

Dans une stratégie de déploiement Blue-Green, deux versions de votre application coexistent :

- L'ancienne version de l'application (Blue), qui est celle vue par les utilisateurs ;
- La nouvelle version (Green), qui est en cours de développement.

Une fois la nouvelle version testée et validée pour la publication, le service passe à la version Green avec l'ancienne version Blue qui devient le slot de la future release.



Stratégie Canary Release

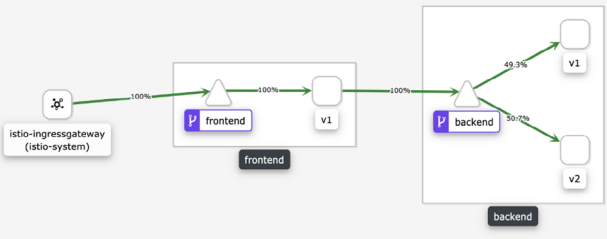
Le Canary est utile lorsque vous souhaitez tester de nouvelles fonctionnalités, généralement sur le backend de votre application, avec précaution.
Deux applications coexistent alors dans votre infrastructure :

- L'application actuelle ayant la quasi-totalité du trafic ;
- L'application disposant de la nouvelle feature que nous souhaitons éprouver avec un trafic régulé.

Vos clients et développeurs pourrons alors tester ces nouvelles fonctionnalités.

Lorsqu'aucune erreur n'est signalée sur les plans technique et fonctionnel, la nouvelle version peut progressivement se déployer sur le reste de l'infrastructure en prenant la place de l'ancienne.

Bien que cette stratégie puisse être réalisée simplement en utilisant les ressources Kubernetes et en remplaçant les anciens et les nouveaux pods, il est beaucoup plus pratique et plus facile de la mettre en œuvre avec un maillage de services comme Istio.



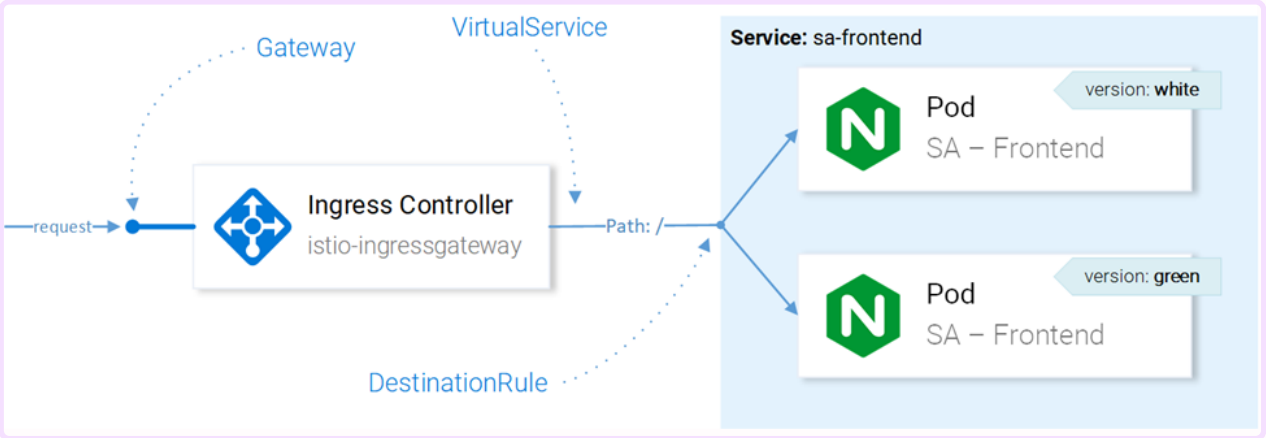
Source : documentation technique Istio

Stratégie d'A/B Testing

L'A/B Testing rejoint beaucoup le concept de Canary et est plus destiné aux fonctionnalités front-end.
Plutôt que de lancer une nouvelle fonctionnalité pour tous les utilisateurs, vous pouvez la proposer à un petit nombre d'entre eux. Les utilisateurs ne savent généralement pas qu'ils sont utilisés comme testeurs pour la nouvelle fonctionnalité.

A l'aide de métriques/KPIs, vous pourrez alors observer l'efficacité de votre nouvelle interface utilisateur. Permet-elle d'avoir une belle conversion business ? Est-elle trop déroutante pour vos utilisateurs ?

Une fois de plus, la mise en pratique est plus aisée lorsque vous disposez d'un Service Mesh dans vos clusters. Vous pourrez alors gérer le trafic via la détection de vos cookies, headers, etc.



Vue d'ensemble des stratégies de déploiement

Stratégie	Pas d'interruption de service	Coût (Compute)	Temps de rollback	Complexité de mise en œuvre
Recreate	✗	$	Lent	☐☐☐
Rolling updates	✓	$	Lent	■☐☐
Blue / Green	✓	$ $	Très rapide	■■☐
Canary	✓	$	Rapide	■■☐
A/B Testing	✓	$	Rapide	■■■

Industrialiser ses packages avec Helm

Dans Kubernetes, la configuration de vos services/applications se fait généralement via des fichiers YAML. Quand on a une seule application en ligne, cela reste assez simple mais dès qu'on a plusieurs environnements, applications et services, on se retrouve très vite submergé de fichiers plus ou moins semblables.

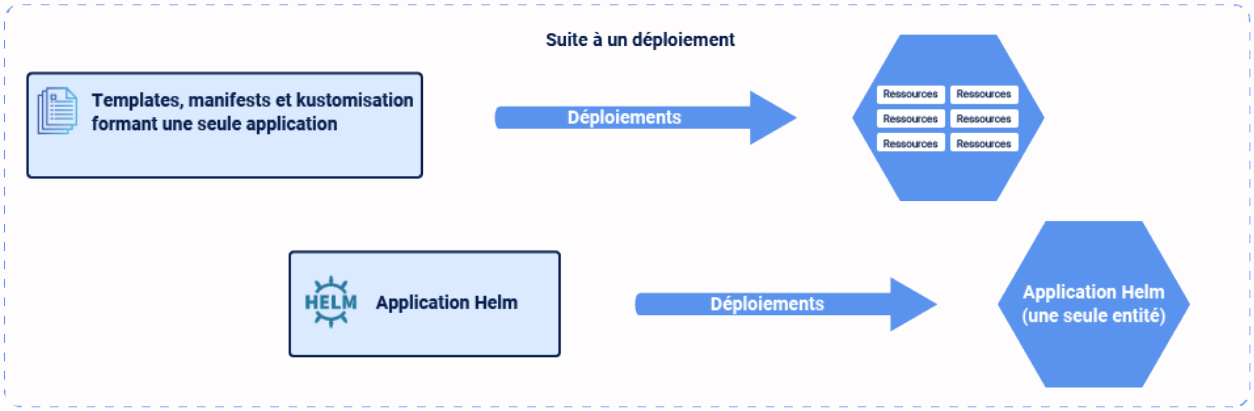
C'est là qu'intervient Helm !

Helm est le **package manager soutenu et recommandé par Kubernetes**. Il est aussi l'un des seuls sur le marché, son unique concurrent, KPM de CoreOS, n'étant plus maintenu depuis juillet 2017.

Pourquoi Helm ?

Toutes les solutions de création de modèles pour Kubernetes souffrent du même problème : Kubernetes ne connaît qu'un ensemble de manifestes et rien de plus. La notion d'application est perdue et ne peut être recréée qu'en ayant les fichiers sources d'origine à portée de main.

Helm, quant à lui, connaît l'ensemble de l'application et stocke les informations spécifiques à l'application dans le cluster lui-même. Il peut alors suivre les ressources de l'application suite au déploiement. La différence est subtile mais importante.

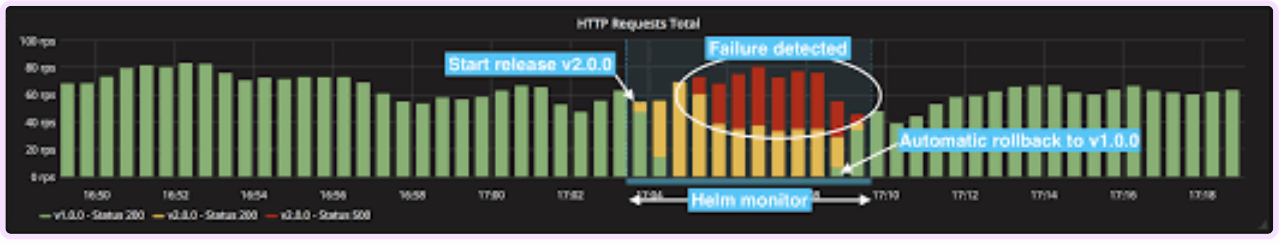


Il est inutile de comparer Helm aux outils de création de modèles tels que Kustomize / k8compt / kdeploy : Helm est bien plus qu'une solution de création de modèles.
Ce package manager présente d'autres avantages :

- Moteur de templating permettant de factoriser les manifestes YAML de toutes vos applications ;
- Organisation et unification de vos templates avec une arborescence cohérente ;
- Mise à disposition de vos charts sur des dépôts publics ou privés ;
- Update et Rollbacks des révisions de vos applications sans rupture de service ;
- Système de dépendance avec d'autres charts (encore plus de factorisation) ;
- Test via le CLI pour vérifier les connectivités.

Helm facilite le déploiement par ses notions d'upgrade et rollback. Vous pouvez déployer une nouvelle version avec un auto-incrementé et revenir sur une version antérieure dès que vous le souhaitez, sans *downtime* de vos services.

Toutes les versions de votre application sont disponibles sur votre cluster à tout moment. Il est tout à fait possible d'implémenter un rollback automatique basé sur vos métriques afin d'éviter certaines catastrophes.



Vous allez pouvoir unifier et simplifier vos centaines de manifestes YAML Kubernetes en quelques charts. Terminé les duplications de configuration !

Helm V3

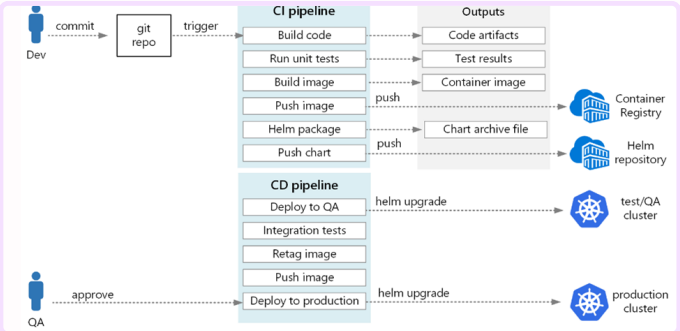
Depuis Helm V3, il n'y a plus de composant côté serveur (le tristement célèbre Tiller) : donc, si la dernière fois que vous avez évalué Helm, vous aviez des inquiétudes concernant la sécurité du cluster, vous devez jeter un nouveau regard sur Helm maintenant que Tiller ne fait plus partie de l'architecture.

À moins que vous n'ayez un flux de travail vraiment orthodoxe, ne pas utiliser Helm revient à ne pas utiliser apt-rpm pour la gestion des paquets. Si vous décidez que Helm n'est pas pour vous, cela devrait être un choix conscient (fait après mûre réflexion) et non basé sur des informations erronées sur le choix d'une « meilleure solution de modélisation que Helm ».

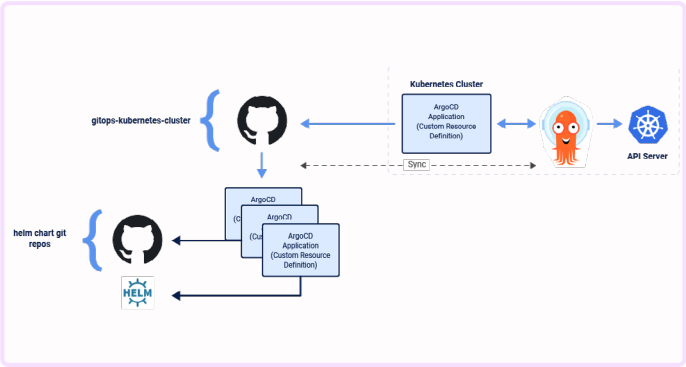
Sur la V3, certaines améliorations ont été effectuées sur la gestion des dépendances et library charts. Vous pouvez disposer de vos charts en local, et les pusher dans les registries de votre choix.

Comment Intégrer Helm dans vos pipelines CI/CD ?

L'intégration de vos packages Helm dans votre CI demande peu de modifications de votre CI. Voici le schéma d'une approche classique :



Et celui d'une approche GitOps en mode Pull :



Dans les 2 cas :

- Vous devez stocker vos charts dans vos repository Helm lors de la CI ;
- Vous pourrez alors en disposer lors de votre CD.

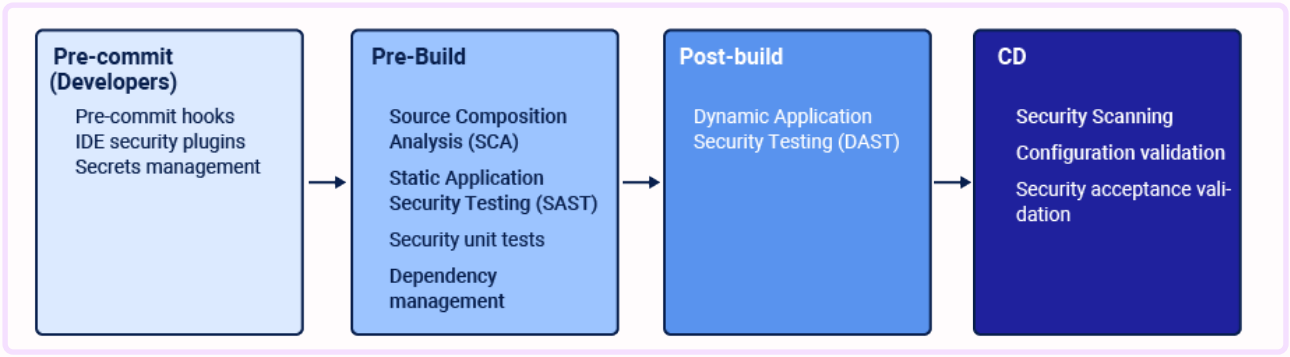
Intégrer la sécurité dans vos pipelines CI/CD

Une fois vos pipelines CI/CD mis en place et fonctionnels, vous avez la possibilité d'ajouter des tâches supplémentaires et de faire certaines vérifications avant d'exécuter chaque tâche de votre pipeline. Cela pourra créer des rapports, bloquer le pipeline, vous avertir en cas de problème...

La plupart du temps, vous allez utiliser des outils open-source ou des produits éditeurs.

Vous pouvez par exemple :

- Vérifier la présence de données sensibles (secrets, respect des bonnes pratiques de développement) ;
- Analyser le code source (applicatif, IaC) ;
- Rechercher les vulnérabilités sur les dépendances projets ;
- Lancer des tests de sécurité sur les images de conteneurs ;
- Vérifier l'intégrité de votre image de conteneur.

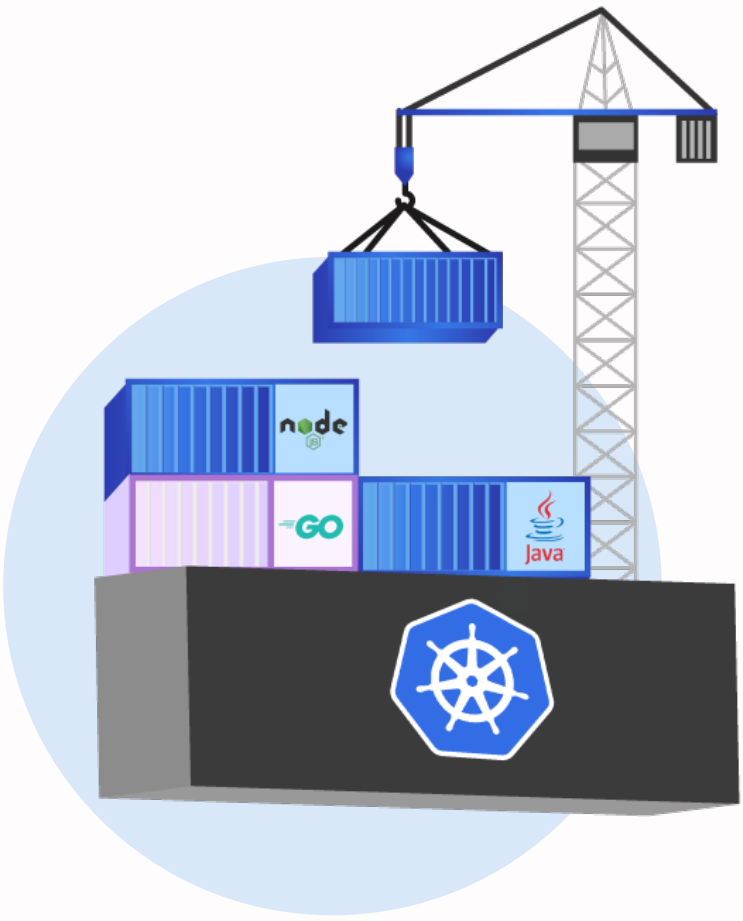


L'essentiel à retenir

Nous avons pu voir ensemble les éléments clés à ne surtout pas mettre de côté lors de la création de votre infrastructure ainsi que l'importance de la mise en place d'une bonne CI/CD.
Dans le cadre d'une modernisation de vos applications Cloud native, il est judicieux de mettre en place une **automatisation performante** afin de **réduire le time to market**.
Sur la partie infrastructure, partir d'un template est bénéfique afin de suivre l'évolution des clusters Kubernetes mais aussi de le réutiliser pour différents cas d'usage, comme le déploiement d'autres environnements et la mise à jour de Kubernetes dans le cadre du Blue/Green. **Nous recommandons fortement l'IaC**.

L'écosystème de Kubernetes est en perpétuel mouvement. Il n'est donc pas aisé de se retrouver avec cette multitude d'outils proposés par la toile. Les montées de version de vos clusters sont, elles aussi, conséquentes !
Il est donc indispensable d'être à l'écoute des technos et de vous adapter en continu.
Et aussi d'avoir une rigueur dans la mise en place de vos configurations et sans cesse optimiser vos processus : configuration des ressources, policy, sécurité, monitoring...

METTRE EN PLACE SA STRATÉGIE DE MIGRATION VERS KUBERNETES



Tout le monde veut que son application s'exécute sur Kubernetes. Mais plusieurs questions se posent : comment démarrer ce processus ? Quelles étapes doivent être effectuées ? Comment estimer l'effort ?

De nombreuses équipes de développement ne sont qu'au début de leur aventure avec K8s. Malgré la popularité de Kubernetes ces dernières années, de nombreuses organisations commencent seulement à planifier le processus de migration.

Dans ce chapitre, nous aborderons les différentes phases de migration en mettant l'accent sur les concepts de base de la conteneurisation, le passage d'un monolithe vers un micro-service et les solutions présentes sur le marché qui vont nous aider à entamer cette phase de migration vers Kubernetes et héberger nos micro-services dans des clusters Kubernetes.

Partir d'un projet existant

Pourquoi moderniser une application ?

Actuellement, la majorité des entreprises adoptent une architecture monolithique simple, légère, pour commencer leurs projets. Cette architecture représente le modèle traditionnel unifié de conception d'un programme informatique.

Mais à mesure que les applications se développent, une architecture monolithique commence à entraver et même à réduire la croissance organisationnelle.
Les nouvelles fonctionnalités prennent plus de temps à être fournies, tout comme l'intégration des nouveaux employés, ce qui impacte l'évolution des applications.

Avec l'évolution du système d'information de l'entreprise, l'architecture est passée des modèles monolithiques à une architecture orientée services et progresse rapidement vers les micro-services, une architecture qui peut résoudre une grande partie du rythme lent du développement de systèmes monolithiques.

En modernisant l'application d'une architecture monolithique vers une architecture micro-services, l'application peut être divisée en plusieurs fonctions (micro-services) qui nous permettent de travailler indépendamment sur des composants individuels, augmentant ainsi la vitesse de développement logiciel (time to market).

Définition du micro-service

L'architecture micro-service a été créée pour répondre au besoin d'agilité. Les entreprises doivent analyser leurs données, innover et lancer de nouveaux produits et services plus rapidement que leurs concurrents. Elles doivent être flexibles pour répondre aux besoins changeants de leurs clients. La migration vers l'architecture de micro-services permet d'atteindre cet objectif.

Parmi les critères d'adoption des micro-services, on peut citer :

- **Croissance de l'entreprise** : comme l'application sert plus de clients et traite plus de transactions, elle a besoin de plus de capacité et de ressources ;
- Capacité à **absorber les pics de charge** sans risquer l'interruption de service ;
- Capacité à faire évoluer/**délivrer de nouvelles fonctionnalités applicatives plus rapidement** pour répondre aux besoins ;
- **Pics de trafic** : le système devrait pouvoir évoluer automatiquement, de manière à ce que l'infrastructure ne soit pas poussée à sa capacité maximale pour prendre en charge les pics de trafic. La mise à l'échelle des applications monolithiques peut souvent être un défi ;
- **Délai de livraison plus rapide** : il y a une valeur significative pour l'entreprise lorsque l'ajout ou la modification d'une fonctionnalité prend moins de temps et ne nécessite pas de tests de régression excessifs (et souvent coûteux) ;
- **Domaine d'expertise** : les applications monolithiques évoluent très rapidement ce qui rend la prise en main de l'intégralité de l'application par le développeur un vrai défi ;
- **Agilité opérationnelle** : il est difficile d'atteindre l'agilité opérationnelle dans le déploiement répété d'artefacts d'application monolithiques ;
- **Agilité** : la facilité d'ajouter des nouvelles fonctionnalités dans un micro-service.

La transformation à l'aide du pattern Domain-Driven-Design

La migration d'un monolithe vers un micro-service nécessite un temps et un investissement importants pour éviter les pannes ou les dysfonctionnements. Commençons par comprendre l'intérêt des micro-services :

- Les services peuvent évoluer de façon indépendante en fonction des besoins des utilisateurs ;
- Les services peuvent évoluer de façon indépendante pour répondre à la demande des utilisateurs ;
- Les cycles de développements s'accélèrent pour délivrer des fonctionnalités plus rapidement ;
- La segmentation en micro-services rend l'application plus tolérante aux pannes et résiliente à la défaillance d'un de ses composants ;
- Les tests deviennent plus consistants grâce au BDD.

La stratégie de migration consiste à réaliser le *refactoring* de l'application en la découpant en services plus petits. On peut synthétiser ainsi les grands principes du *refactoring* :

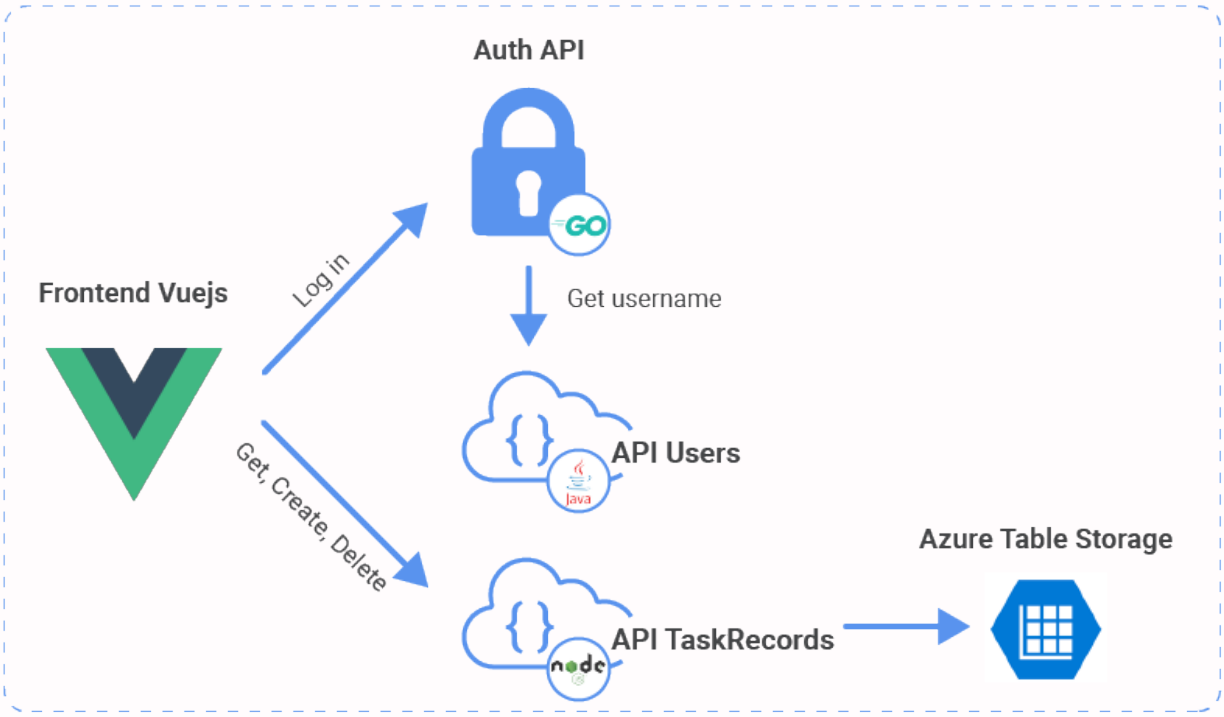
- Arrêter d'ajouter des fonctionnalités au monolithe ;
- Séparer le Frontend du Backend ;
- Décomposer et découpler le monolithe en une série de micro-services ;
- Encapsuler les opérations qui peuvent échouer pour améliorer la stabilité et la résilience de l'application.

Pour illustrer notre propos, nous allons observer une application simple construite autour de micro-services.

Notre application va authentifier dans un premier temps les utilisateurs en leur permettant de générer des tokens JWT. Par la suite, ces tokens leur serviront à exécuter un ensemble d'actions.

Dans ce qui suit, nous allons détailler les différents composants de notre application :

- La partie frontale est une application Javascript. Elle fournit une interface utilisateur et a été créée avec VueJS ;
- L'API Auth est écrite en Go et fournit des fonctionnalités d'autorisation. Elle génère des jetons JWT à utiliser avec d'autres APIs ;
- L'API Users est un projet Spring Boot écrit en Java. Cette API fournit des profils utilisateur et permet de retrouver un utilisateur ou la liste des utilisateurs.
- L'API TaskRecords est écrite en NodeJS et fournit des fonctionnalités CRUD sur les enregistrements de tâches de l'utilisateur. En outre, elle enregistre les opérations de création et de suppression dans une Azure Table Storage, afin qu'elles puissent éventuellement être traitées ultérieurement par un log message processor.



Cette architecture est le résultat d'une mise en place du pattern DDD (Domain-Driven-Design) qui facilite cette décomposition et ce découpage, en passant par une bonne compréhension métier et à l'aide des quelques Cloud-design patterns qui peuvent découler de l'application de cette approche et qui représentent un avantage pour une meilleure migration.

Ces patterns sont :

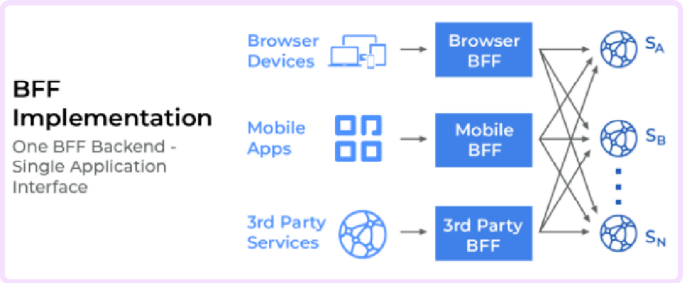
- Back-End for Front-End pattern ;
- Bulkhead pattern ;
- Retry pattern ;
- Circuit Breaker.

Back-End for Front-End pattern

Après avoir migré notre monolithe vers des micro-services, nous devons normaliser la communication entre composants front et back tout en permettant à de dernier de continuer à évoluer. Pour adresser cette problématique, on peut utiliser le pattern Backends For Frontends pour introduire une couche intermédiaire (BFF) qui va formater les données pour le compte du composant Front-End. Le composant Back-End peut donc continuer à évoluer indépendamment.

Les principaux avantages de de patterns sont :

- **SoC (Separation of Concerns)** : la séparation de responsabilités entre le Back-End et le Front-End ;
- **Meilleure gestion des erreurs dans le Front-End** : dans la plupart des cas, les erreurs du serveur n'ont pas de sens pour l'utilisateur du Front-End. Le BFF peut mapper les erreurs qui doivent être montrées à l'utilisateur, ce qui améliorera son expérience ;
- **Meilleure sécurité** : certaines informations sensibles peuvent être masquées et les données inutiles au Frontend peuvent être omises lors du renvoi d'une réponse au Front-End. L'abstraction rendra plus difficile pour les attaquants de cibler l'application.
- **Composants partagés entre les équipes** : différentes parties de l'application peuvent être gérées très facilement par différentes équipes. Les équipes frontales peuvent s'approprier à la fois leur application client et sa couche de consommation de ressources sous-jacentes (BFF), conduisant à des vitesses de développement élevées.



Bulkhead pattern

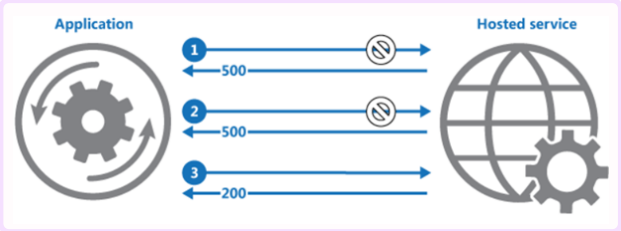
L'utilisation du Cloud pattern Domain-Driven-Design permet de rendre notre application plus résiliente en la compartimentant en sections, à la manière de la coque d'un navire. Si la coque d'un navire est compromise, seule la section endommagée se remplit d'eau, ce qui empêche le navire de couler. En appliquant ce principe à notre application, nous isolons nos micro-services dans des conteneurs dédiés.

Les principaux avantages de ce pattern sont :

- Isoler les consommateurs et les services des pannes en cascade. Un problème affectant un consommateur ou un service peut être isolé dans sa propre cloison, empêchant ainsi à l'ensemble de la solution de tomber en panne ;
- Permettre de conserver certaines fonctionnalités en cas de panne de service. Les autres services et fonctionnalités de l'application continueront à fonctionner ;
- Permettre de déployer des services qui offrent une QoS (Quality of Service) différente pour les applications consommatrices. Un pool de consommateurs à haute priorité peut être configuré pour utiliser des services à haute priorité.

Retry pattern

Certaines erreurs peuvent être transitoires. Avant de les considérer comme de véritables erreurs, il convient de réessayer en ajoutant un délai. Chaque fois que nous supposons qu'une réponse inattendue - ou aucune réponse d'ailleurs - peut être corrigée en renvoyant la demande, l'utilisation de ce pattern peut aider. Il s'agit d'un pattern dans lequel les demandes ayant échoué sont réessayées un nombre configurable de fois en cas d'échec avant que l'opération ne soit marquée comme un échec.



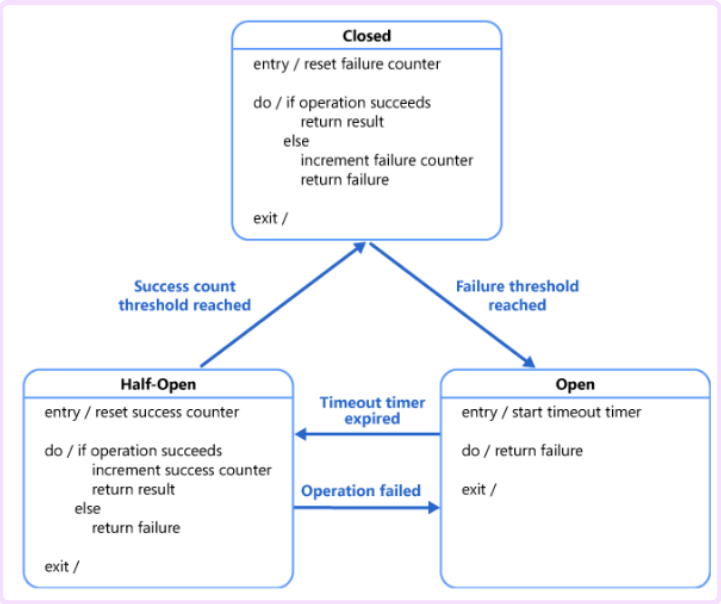
Circuit Breaker pattern

Le pattern Circuit Breaker empêche une application de tenter en permanence une opération avec de fortes chances d'échec. C'est une évolution du pattern précédent. On introduit une bascule vers une seconde chaine applicative vers laquelle on redirige les requêtes en attendant que la chaine applicative principale soit de nouveau opérationnelle.

Le Circuit Breaker présente trois états distincts :

- **Closed** : lorsque tout est normal, le Circuit Breaker reste à l'état fermé et tous les appels passent par les services. Lorsque le nombre de pannes dépasse un seuil prédéterminé, le Circuit Breaker se déclenche et passe à l'état ouvert ;
- **Open** : dans cet état, tous les appels au service échouent immédiatement, ce qui signifie qu'ils ne sont pas exécutés et renvoient ainsi la dernière exception connue à l'application ;
- **Half-Open** : après une période de temporisation, le circuit passe à un état semi-ouvert pour tester si le problème sous-jacent existe toujours. Si un seul appel réussit dans cet état, le Circuit Breaker est à nouveau déclenché et revient à l'état normal fermé. On revient donc à la chaine applicative principale.

Une application peut combiner ces deux derniers patterns en utilisant le pattern Retry pour appeler une opération via un Circuit Breaker. Cependant, la logique de Retry doit être sensible à toutes les exceptions renvoyées par le Circuit Breaker et doit abandonner les tentatives de relance si le Circuit Breaker indique que le problème n'est pas transitoire.



Migrer votre projet sur le Cloud

Avant de commencer à déployer des micro-services sur Kubernetes, nous allons tout d'abord vous parler de ce qu'est Kubernetes et des raisons pour lesquelles nous devrions l'utiliser pour déployer des micro-services.

Dans cette partie nous allons détailler les composants Kubernetes et l'architecture générale des opérations. Nous discuterons également des définitions minimales de manifeste pour le déploiement de micro-services, le déploiement des ReplicaSets, des pods et des services.

Composants Kubernetes

Nous allons détailler les composants utilisés par Kubernetes lors du déploiement d'une application. Nous expliquerons les éléments de base que nous pouvons utiliser pour déployer les micro-services sur K8s comme le déploiement, les ReplicaSets, les pods et les services.

Pods

Il s'agit des plus petites unités déployables qu'on peut gérer dans Kubernetes. Un pod est un groupe d'un ou plusieurs conteneurs, avec des ressources de stockage/réseau partagées et une spécification sur la manière d'exécuter les conteneurs.

Les pods d'un cluster Kubernetes sont utilisés de deux manières :

- **Pods qui exécutent un seul conteneur** : le modèle « un conteneur par pod » est le cas d'utilisation le plus courant de Kubernetes, dans lequel nous pouvons considérer un pod comme un *wrapper* autour d'un seul conteneur. Kubernetes gère les pods plutôt que de gérer les conteneurs directement.
- **Pods qui exécutent plusieurs conteneurs devant fonctionner ensemble** : un pod peut encapsuler une application composée de plusieurs conteneurs colocalisés qui sont étroitement couplés et doivent partager des ressources. Ces conteneurs colocalisés forment une seule unité de service cohésive. Le pod regroupe ces conteneurs, ces ressources de stockage et une identité réseau éphémère en une seule unité.

Si l'approche générale est d'avoir toujours un conteneur par pod dans un *deployment*, on pourra en revanche toujours avoir recours au modèle « pod exécutant multiples conteneurs » dans ces deux cas précis :

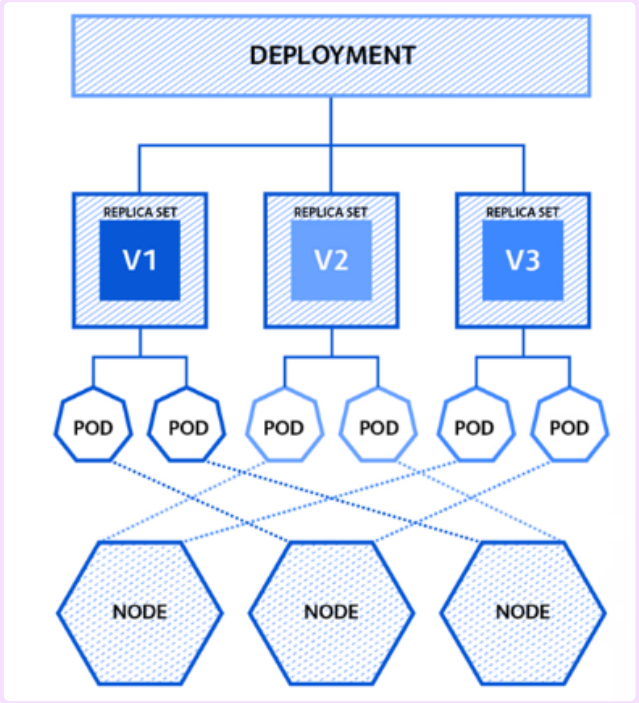
- **Sidecar containers** : ils vont aider le conteneur principal. Pour illustrer ce cas de figure, on peut penser aux observateurs de changement (trackers), aux services de log, aux adaptateurs de monitoring, etc. Un log watcher, par exemple, peut être créé par une équipe et réutilisé dans différentes applications ;
- **Proxies, bridges, adapters** : ils permettent de connecter le conteneur principal au monde extérieur. Citons à titre d'exemple un serveur HTTP Apache ou nginx qui peut servir des fichiers statiques et agir en tant que proxy inverse pour une application Web dans le conteneur principal pour logger et limiter les requêtes HTTP.

ReplicaSet

L'objectif d'un ReplicaSet est de maintenir un ensemble stable de pods de réplique en cours d'exécution à tout moment. Par exemple, il est souvent utilisé pour garantir la disponibilité d'un nombre spécifié de pods identiques.

Deployments (Déploiements)

Un déploiement fournit des mises à jour déclaratives pour les pods et les ReplicaSets.
On commence par décrire un état souhaité dans un déploiement et le contrôleur de déploiement modifie l'état réel en l'état souhaité. On pourra définir des déploiements pour créer de nouveaux ReplicaSets ou pour supprimer des déploiements existants et adopter toutes leurs ressources avec de nouveaux déploiements.



Service

Ce composant définit une manière abstraite d'exposer une application exécutée sur un ensemble de pods en tant que service réseau. Kubernetes donne aux pods leurs propres adresses IP et un seul nom DNS pour un ensemble de pods, et peut équilibrer la charge entre eux.

ConfigMaps

Un ConfigMap est un objet API utilisé pour stocker des données non confidentielles dans des paires clé-valeur. Les pods peuvent consommer des ConfigMaps en tant que variables d'environnement, arguments de ligne de commande ou fichiers de configuration dans un volume. La bonne pratique est d'externaliser ces configurations dans les ConfigMaps pour rendre le déploiement indépendant de l'environnement.

Cette idée est préconisée par la méthodologie Twelve-Factor App qui consiste à stocker la configuration dans des variables d'environnements. L'avantage de cette approche est que les variables d'environnements sont faciles à changer entre les déploiements sans changer de code, contrairement aux fichiers de configuration qui peuvent être une source de risque s'ils sont répertoriés accidentellement dans notre code source.

Secrets

Les secrets Kubernetes nous permettent de stocker et de gérer des informations sensibles, telles que les mots de passe, les tokens Oauth et les clés SSH. En effet, stocker des informations confidentielles dans un secret est plus sûr et plus flexible que de les mettre en clair dans une définition de pod ou dans une image de conteneur.

Requests et Limits resources

La gestion des **Requests** et des **Limits** est une étape fondamentale pour améliorer les performances du cluster et l'optimisation des applications.

Les **Requests** et les **Limits** sont les mécanismes utilisés par Kubernetes pour contrôler les ressources telles que le CPU (Central Processing Unit) et la mémoire. Les Requests sont ce que le conteneur a la garantie d'obtenir. Si un conteneur demande une ressource, Kubernetes ne la planifiera que sur un nœud qui peut lui donner cette ressource. En revanche, les Limits garantissent qu'un conteneur ne dépasse jamais une certaine valeur.
Il est important de préciser que **la Limit ne peut jamais être inférieure à la Request**.
Le cas échéant, Kubernetes générera une erreur et ne laissera pas exécuter le conteneur.

- **Request** : en plus de garantir une allocation de ressources appropriées pour s'assurer que l'application peut s'exécuter correctement, une demande de ressource pour un conteneur aide le planificateur Kubernetes à décider du nœud approprié sur lequel placer un pod ;
- **Limit** : les limites garantissent qu'aucun processus en cours n'utilise plus qu'une certaine part des ressources sur un nœud. Dans le cas où un conteneur commence à dépasser sa limite de CPU, le kubelet commencera à ralentir le CPU.

K8s manifest

Il s'agit d'un fichier YAML permettant de créer, modifier et supprimer des ressources Kubernetes telles que les pods, les déploiements, les services ou les Ingress. En d'autres termes, il spécifie l'état souhaité d'un objet que Kubernetes conservera lorsque nous allons appliquer le manifeste. Chaque fichier de configuration peut contenir plusieurs manifestes. Le fichier manifest comprend un ensemble de sections, dont les plus couramment utilisées sont :

- **Ingress** : un objet API qui gère l'accès externe aux services dans un cluster, généralement des appels http ;
- **HorizontalPodAutoscaler (HPA)** : une ressource API mettant automatiquement à l'échelle (scaler) le nombre de réplicas de pod en fonction de l'utilisation CPU ou de métriques personnalisées.

Exposition des micro-services dans les clusters Kubernetes

Kubernetes (K8s) facilite nativement la découverte de services via le service DNS nativement présent.
Aucune configuration supplémentaire n'est requise pour accéder aux micro-services au sein du cluster, ce qui nous aide à nous concentrer sur les définitions des services et leurs dépendances sans se soucier de la configuration réseau entre les micro-services.

Dans Kubernetes, la notion de service permet d'exprimer comment nous voulons exposer un micro-service. Plusieurs choix s'offrent à nous :

- **ClusterIP** : expose le service sur une adresse IP interne au cluster. C'est donc une exposition à usage interne/intra-applicatif uniquement ;
- **NodePort** : expose le service sur l'IP de chaque nœud composant notre cluster sur un port statique. Un service ClusterIP, vers lequel le service NodePort est acheminé, est automatiquement créé. Nous pourrons appeler le service NodePort, depuis l'extérieur du cluster, en requêtant comme suit : <NodeIP>: <NodePort> ;
- **LoadBalancer** : expose le service en externe à l'aide d'un LoadBalancer du fournisseur de Cloud. Deux types de LoadBalancer sont disponibles : Internal et External LoadBalancer.

Nos pods sont regroupés en ReplicatSets qui sont exposés via un service Kubernetes. De là, nous sommes en mesure d'exposer cette ressource K8s en interne à l'aide d'un service et en externe à l'aide d'un Ingress pour la rendre accessible de l'extérieur.

Quelques bonnes pratiques de conteneurisation d'applications

Démarrer avec un conteneur le plus léger possible

Pour conteneuriser une application, nous devons tout d'abord choisir une image de base. Nous vous conseillons de bien regarder ce qui est disponible (tags, OS, déclinaisons) et **d'utiliser des images de base les plus simples possible** car elles présentent le moins de surface d'attaque. Dans ce domaine, la distribution Linux Alpine se démarque des autres par sa taille extrêmement réduite mais aussi par le nombre de vulnérabilités qui peuvent y être détectées. En réduisant la surface d'attaque, on améliore la sécurité.

Image	Taille (en MB)	Vulnérabilités détectées (Snyk)
node:14	945.89	547
node:14-alpine	119.35	0

Pour une même version de Node.js, on remarque que la version taguée avec le suffixe « -alpine » est presque 8 fois plus légère. En partant de l'image alpine, on pourra ensuite ajouter, si besoin, les dépendances nécessaires pour faire tourner notre application.

Externaliser la configuration dans les variables d'environnement

Dans les fichiers de déploiement applicatif (chart Helm ou fichiers YAML), nous avons la possibilité de renseigner des variables d'environnement pour les conteneurs. On externalise des éléments en respect de la règle config des Twelve Factor App.

Observabilité

Afin de pouvoir monitorer une application conteneurisée et ses dépendances (base de données, services…), il est recommandé d'implémenter des APIs de *HealthCheck*. Cela se fait au niveau du code source de l'application. Pour résumer, il faudra mettre à disposition au moins un *endpoint* qui permettra de donner une réponse avec l'état de santé des composants dont on souhaite suivre l'état de santé.

Exemple avec l'état de santé d'une application et de la connexion à SQL Server :

```
{
  "status": "Healthy",
  "results": {
    "self": {
      "status": "Healthy",
      "data": {}
    },
    "sqlserver": {
      "status": "Healthy",
      "data": {}
    }
  }
}
```

Si nous souhaitons déployer notre application conteneurisée sur Kubernetes, ce point est très important (cf. liveness, readiness probes) et il est très souvent oublié. Mieux vaut s'y préparer pour éviter les mauvaises surprises.

Rootless container

Pour les conteneurs Linux, il est recommandé de ne pas faire tourner les conteneurs en mode root. Nous devons affecter un utilisateur et lui donner explicitement les permissions nécessaires. Si nous déployons nos conteneurs sur Kubernetes et que nous n'avions pas la main sur le Dockerfile, il est possible d'utiliser SecurityContext dans notre manifeste YAML.

```
securityContext:
  runAsUser: 1000
  allowPrivilegeEscalation: false
  readOnlyRootFilesystem: true
```

Se limiter au strict nécessaire

Nos images ne doivent contenir que le strict nécessaire pour faire fonctionner l'application. Tout ce qui est inutile doit être expurgé de notre image de conteneur. Le contenu statique (images, scripts, styles CSS…) peut même être externalisé dans un CDN.

La même build sur n'importe quelle machine de développement

Pour éviter de faire une build en local sur notre machine, nous pouvons la réaliser dans un conteneur grâce au **multi-stage build**. Nous pouvons partir de deux images de base :

- Une image pour compiler notre projet (build) ;
- Une image finale pour récupérer le résultat de la build.

Cela évitera notamment les imprévus avec la configuration du poste de développement et de pouvoir faire une build sur n'importe quel environnement.

Accélérateurs pour vos projets de migration vers Kubernetes

Sur Kubernetes, beaucoup de projets open-source ou produits éditeurs peuvent répondre aux problématiques autour de l'architecture micro-services. Nous allons nous focaliser sur deux d'entre eux :

- Bridge to Kubernetes ;
- Dapr – Distributed Application Runtime.

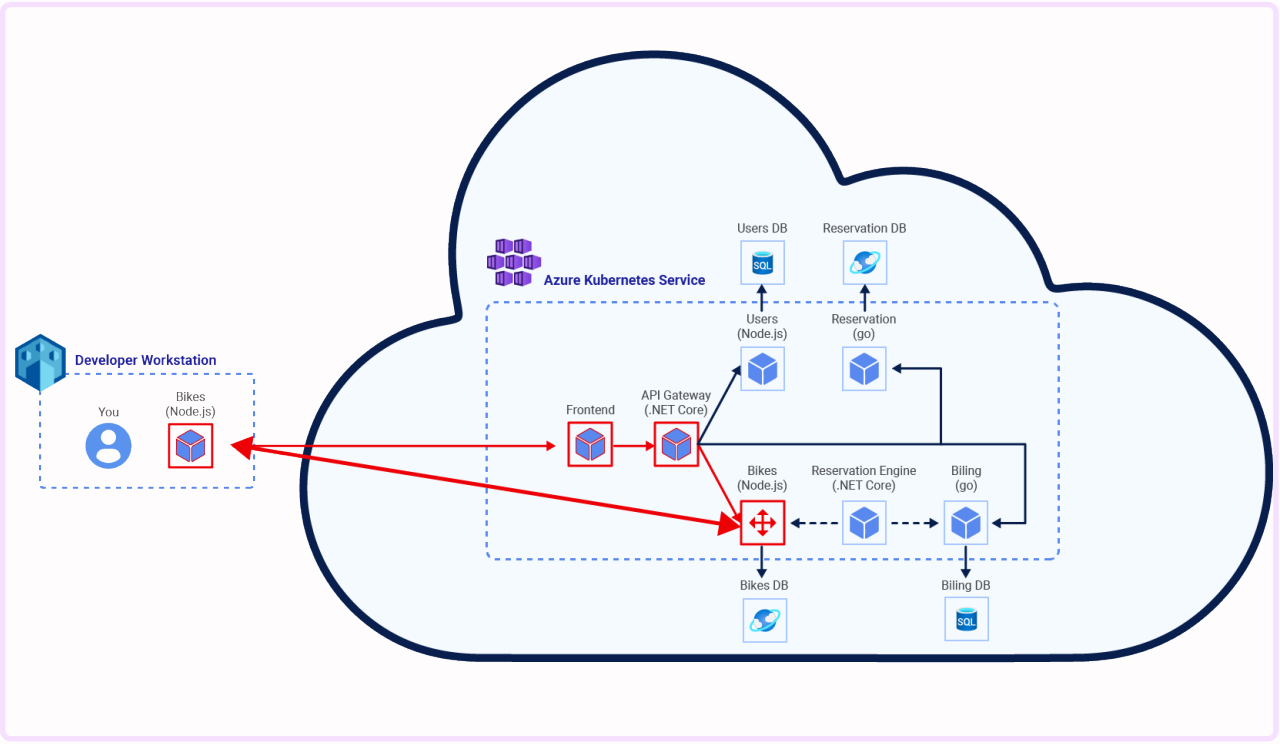
Bridge to Kubernetes

Bridge to Kubernetes vient remplacer Azure Dev Spaces. La solution est intégrée à Visual Studio et Visual Studio Code et permet de substituer un container dans Kubernetes par votre application en local afin de :

- Déboguer l'application sans la conteneuriser ;
- Tester votre projet avec des données de l'environnement cible ;
- Utiliser les services que vous ne souhaitez pas / ne pouvez pas installer en local.

La solution simplifie le développement d'applications micro-services et permet temporairement d'exposer son application locale :

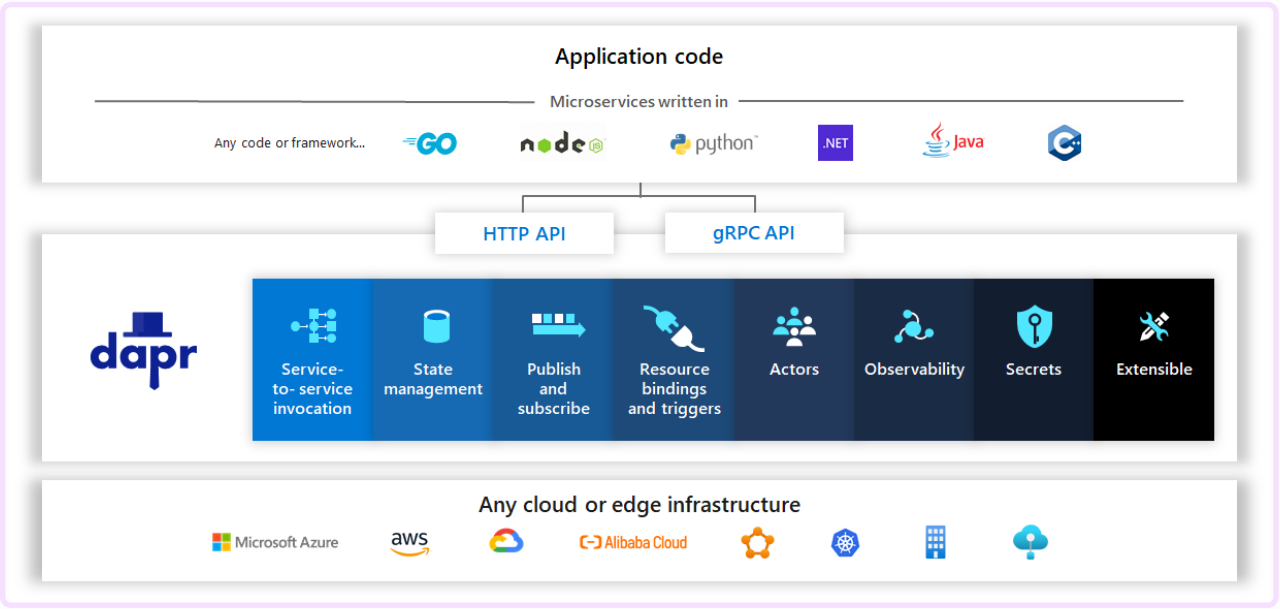
- Pour tous ceux qui accèdent à l'application : démonstration à d'autres membres de l'équipe par exemple ;
- Pour soi (isolation mode) : debug et test.



Source : Documentation Microsoft

Dapr – Distributed Application Runtime

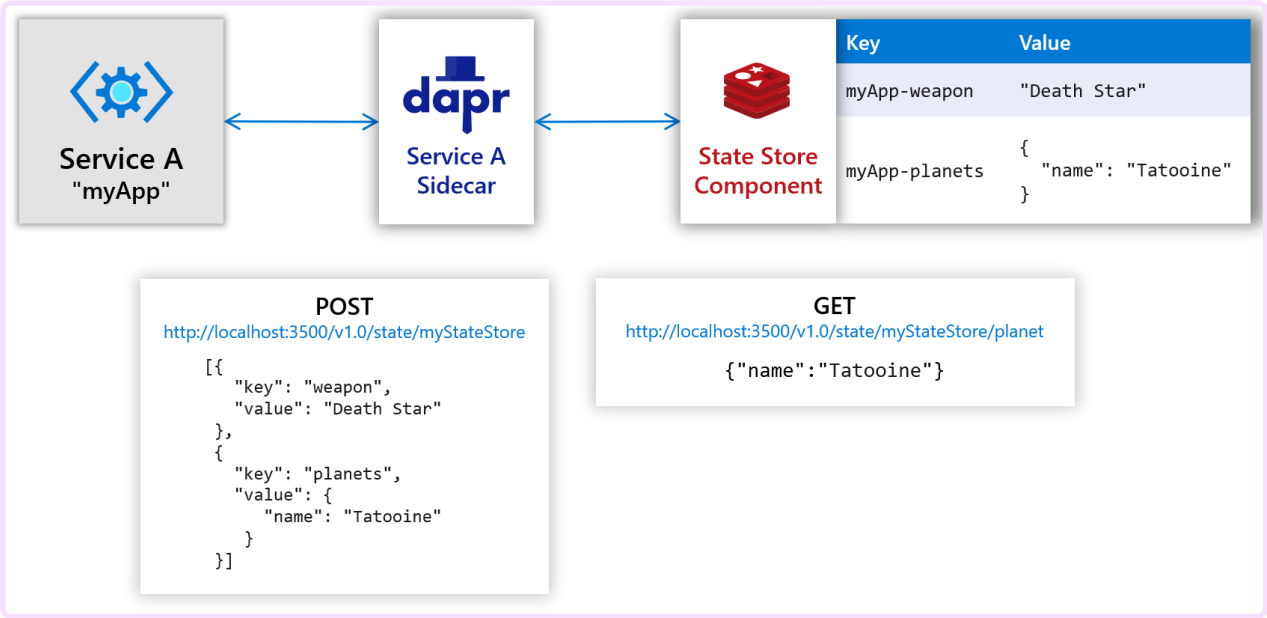
Dapr est un runtime permettant de créer une application/job conteneurisé dans le langage que l'on souhaite et déployable sur Kubernetes. Lancé fin 2019, le projet est en GA (Generally Available – disponible publiquement) depuis la mi-février 2021.



Pour le développeur, ses principales forces sont :

- Les APIs Dapr mises à disposition permettant d'implémenter des patterns récurrents pour les architectures Cloud et micro-services : utilisation de l'API Binding avec un mécanisme de retry.
- Être agnostique des composants externes :
  - Si on souhaite se connecter à un Vault pour récupérer un secret, on peut se connecter à Azure Key Vault ou à Hashicorp Vault avec le même code applicatif ;
  - Dans un scénario hybride ou multicloud, on peut basculer d'un service on-premise à un service Cloud, sans changer le code source de l'application ;
- La possibilité de développer en local (avec Docker), de déployer le même code applicatif et même type de composant Dapr sur un environnement Kubernetes.

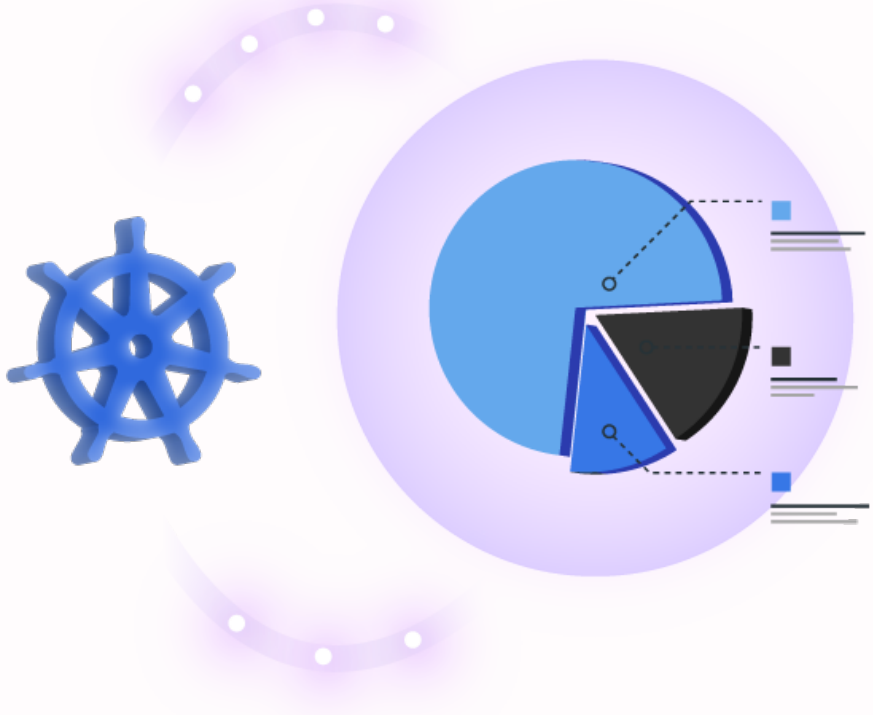
Exemple d'utilisation de Dapr avec Redis Cache (State Management) :



L'essentiel à retenir

Tout au long de ce chapitre, nous avons vu que si vous souhaitez tirer parti de la plateforme Kubernetes, vous devez transformer vos applications. La migration de chacune d'entre elles devra donc être préparée avec soin pour transformer vos monolithes applicatifs en applications « Cloud Native ». Cela constitue un investissement conséquent mais nécessaire pour moderniser vos applications. Pour vous guider, aidez-vous des Twelve-Factor App et des différents patterns architecturaux développés dans cette partie du livre blanc.

COMMENT INTÉGRER KUBERNETES DANS SA STRATÉGIE DE MONITORING ?



La stratégie de supervision de Kubernetes dépasse le cadre même de Kubernetes. On ne va pas construire une plateforme de monitoring spécifiquement pour Kubernetes mais bien **intégrer K8s dans notre stratégie globale**.

Historiquement, on parlait de supervision mais cette simple notion a beaucoup évolué avec le temps. La fonction primaire d'une solution de supervision est de nous informer sur l'état d'un système d'information (SI) : fonctionnel/dysfonctionnel, et ce pour chacune des applications le composant. C'est une information importante pour le calcul de nos SLAs mais ce dont nous avons besoin, c'est de **déterminer pourquoi un SI n'est pas fonctionnel**. De plus, nous devons distinguer l'état de notre plateforme Kubernetes (qui est un système complexe) et l'état de notre application. Devons-nous nous focaliser sur la recherche de potentielles défaillances ou sur la compréhension du fonctionnement de notre écosystème ?

Supervision versus observabilité

La supervision telle que nous la concevons se focalise sur la défaillance. Cependant, **l'absence de défaillance peut tout simplement vouloir dire que nous ne regardons pas au bon endroit**. Tous les composants peuvent être « au vert » sur le dashboard sans que nous ne comprenions si tout se passe bien en dessous.

La supervision telle que nous la connaissons se contente de répondre à deux questions simples :

- Quel est le composant dysfonctionnel ?
- Pourquoi ce composant est-il dysfonctionnel ?

Notre maîtrise de la supervision passe donc par notre **retour d'expérience** sur les situations déjà rencontrées par le passé pour **améliorer notre capacité à détecter ces situations**.

Pour aller plus loin, il faut s'intéresser au fonctionnement global du système indépendamment de son état. Kubernetes est un système complexe sur lequel nous allons déployer des applications qui sont également des systèmes complexes (micro-services). Les causes de dysfonctionnements peuvent être multiples : on doit donc comprendre leur fonctionnement pour ne plus se contenter de constater les effets (défaillances). Dans cette approche, nous devrons nous intéresser aux logs et métriques de performances mais aussi aux traces. Chaque composant de notre écosystème interagit avec un ou plusieurs autres. Ces interactions sont matérialisées par des traces que nous pouvons exploiter pour analyser notre écosystème dans le temps. L'exploitation des traces passées permet de détecter/anticiper des situations et donc d'y répondre plus vite. Par extrapolation, nous pouvons en déduire le comportement de notre écosystème et anticiper les problèmes en connaissance de cause.

Développer l'**observabilité de son écosystème** passe par la **compréhension de ses composants et de leur état (supervision)**. En conséquence développer l'observabilité passe déjà par une bonne culture du monitoring.

Quelles données collecter ?

Historiquement, les solutions de supervision sont axées sur la collecte et l'analyse de deux types de données :

- Les logs ;
- Les métriques/indicateurs de performance.

Avec les logs, nous sommes capables de déterminer l'état d'un système en distinguant différents types de messages (information, avertissement, erreur). Alors qu'avec les métriques, nous analysons le comportement de ce système en comparant un indicateur de performance avec un seuil préalablement établi. C'est le **dépassement de ces seuils** que nous utilisons pour **déclencher des alertes** et affecter des incidents à des groupes de personnes pour prise en charge.

Avec le Cloud, les choses ont un peu évolué. Les applications que nous hébergeons dans Azure sont capables de produire des logs et des métriques en quantités bien plus importantes que par le passé.

Avec Kubernetes, la volumétrie de données produites s'est même accentuée, ce qui pose plusieurs questions :

- **Devons-nous tout logger ?**

Technologiquement, c'est faisable mais il faut s'interroger : en quoi ce log/métrique me permet-il de prendre de meilleures décisions ?

- **De quelle fréquence d'échantillonnage ai-je besoin pour prendre des décisions ?**

La réponse dépendra de plusieurs critères techniques mais aussi contractuels avec le SLA (Service Level Agreement) pour lequel nous sommes engagés. Plus nous sommes exigeants dans notre engagement, plus notre temps de remise en état doit être court, plus nous avons besoin d'informations actualisées.

- **Quelle est la valeur de l'information que je collecte ?**

La réponse à cette question dépend du destinataire de l'information. La métrique de performance d'exécution d'un appel à une API intéresse clairement un développeur pour l'aider à optimiser la performance de son code. A l'opposé, la valeur marchande des transactions en attente de paiement sur un site de commerce électronique a beaucoup de valeur pour le management.

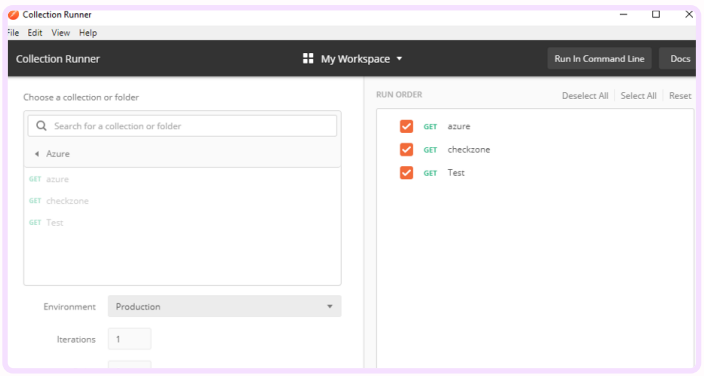
Les informations que nous allons collecter ont donc de la valeur, et cette dernière peut être différente en fonction de celui qui va l'utiliser. Il faudra être en mesure d'**identifier ce qui a de l'intérêt pour nous et sous quel format**. Bien souvent, la véritable valeur de cette donnée ne pourra être perceptible qu'après une phase de mise en corrélation et de raffinage. A l'ère du Cloud, les données constituent une grande richesse (on parle même d' « or noir ») mais il ne faut pas perdre de vue que la donnée « raffinée » a un coût et que sa valeur dépend de chaque acteur.

Ne pas négliger la mesure des usages

La plateforme Kubernetes est en mesure de fournir un grand nombre d'informations pouvant être exploitée par plusieurs équipes :

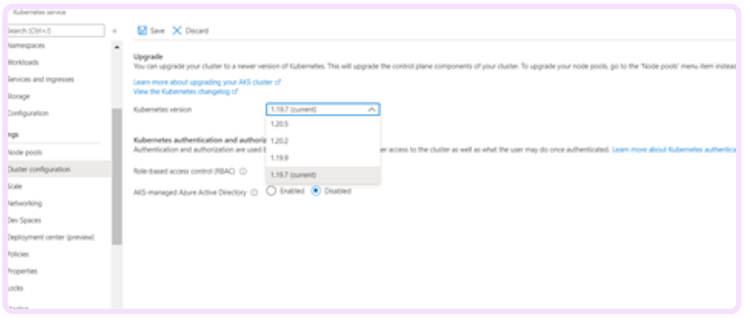
- Opérations (Ops) ;
- Applicatives ;
- FinOps.

Les équipes Ops vont s'intéresser à la mesure des usages de la plateforme Kubernetes pour **déterminer si notre cluster dispose des ressources suffisantes** (les **CPU** - Central Processing Units - et la **mémoire** sont des ressources rares) pour héberger nos applications. Les Ops utilisent ces métriques pour configurer des seuils déclencheurs pour ajuster dynamiquement le nombre de nœuds dans notre cluster (mécanisme de cluster autoscaler).



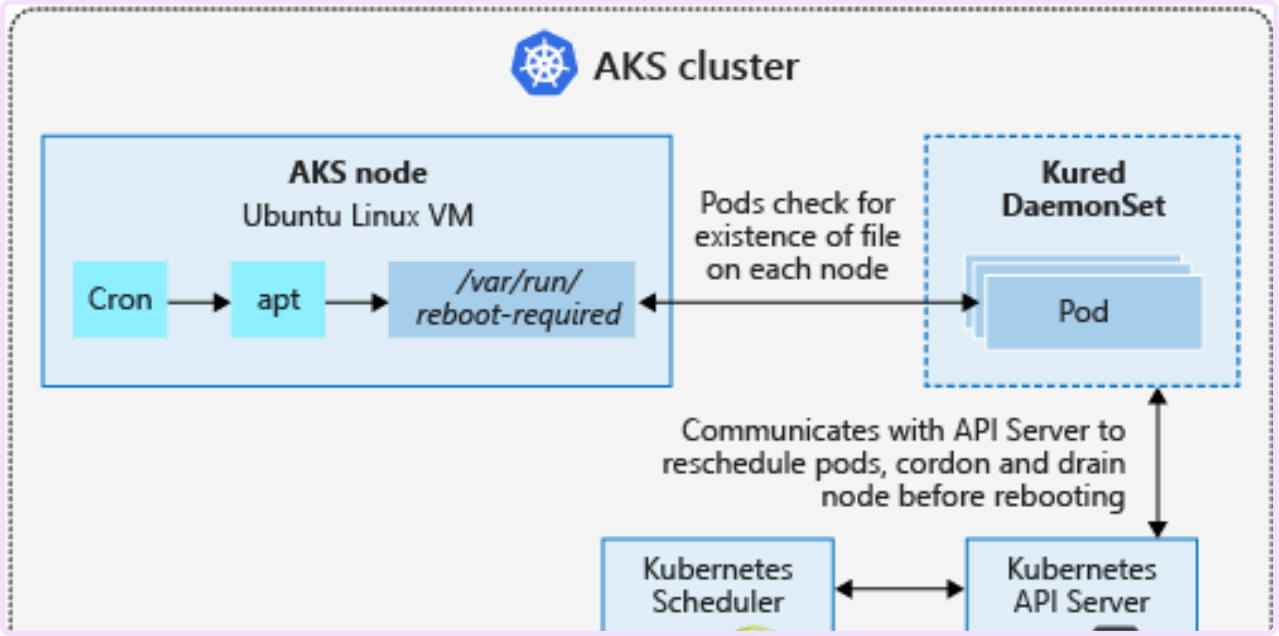
AKS Cluster Autoscaler

Au niveau applicatif, la mesure des usages nous permet de gérer la mise à l'échelle de notre application pour s'adapter dynamiquement à l'usage (mécanisme d'Horizontal Pod scaler). A ce niveau, on observe les critères techniques (CPU & mémoire) mais aussi les indicateurs de performance créés pour observer l'application (nombre d'utilisateurs connectés, nombre d'opérations métiers réalisées...). On utilise ces indicateurs pour déterminer le nombre de pods nécessaires à notre application pour toujours être en mesure de répondre aux sollicitations.



AKS Horizontal Pod Autoscaler

Au niveau **FinOps**, la mesure des usages se focalise sur le bon usage des ressources mises à disposition pour minimiser le « waste », c'est-à-dire la part de ressource Cloud que nous payons mais que nous ne consommons pas efficacement. Dans un cluster Kubernetes, il y a toujours un niveau de « waste » nécessaire et acceptable pour le bon fonctionnement. Une solution comme KubeCost collecte des métriques qui sont ensuite qualifiées en unités de mesure monétaires via les APIs de cost management de la plateforme Azure. Nous venons ainsi de « raffiner » une donnée brute en lui apportant du contexte.
Un des principes de FinOps est de responsabiliser les consommateurs sur leur usage



du Cloud. Appliqué à Kubernetes, chaque équipe en charge d'une application doit être en mesure d'avoir une projection du coût de ses ressources hébergées sur Kubernetes.

Quelles sont les solutions de supervision ?

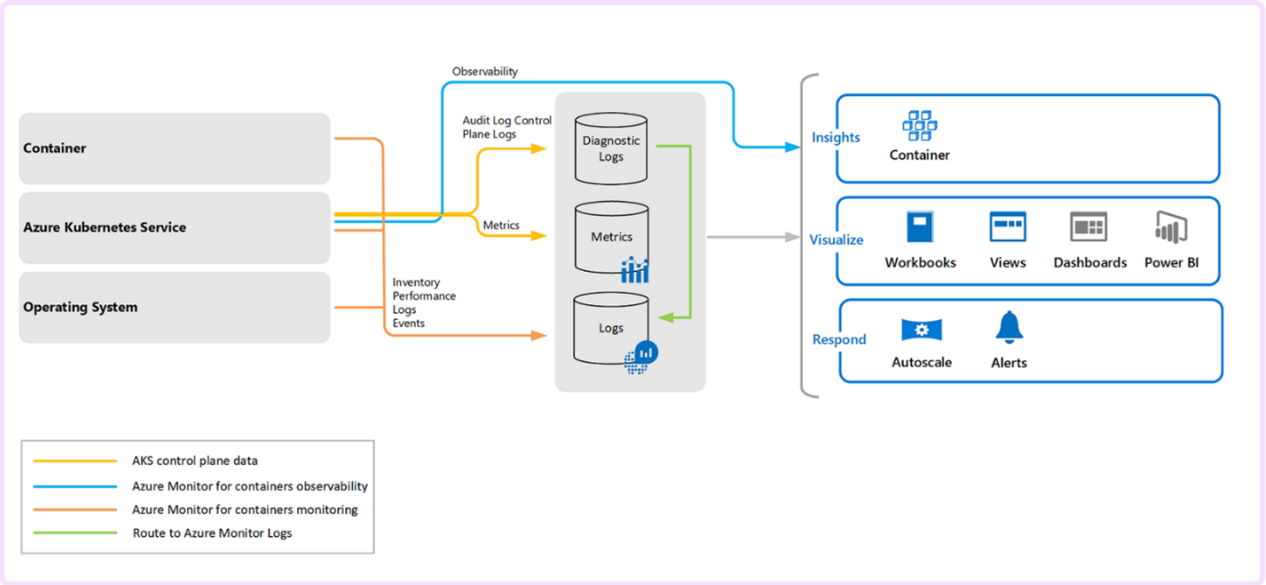
Les solutions de supervision sont nombreuses sur le marché. Sans rentrer dans un inventaire détaillé, on peut en identifier plusieurs familles :

- Solution du Cloud provider ;
- Solutions éditeurs ;
- Solutions open-source.

Pour chaque famille, nous avons retenu un candidat, le plus représentatif possible.

Solution du Cloud Provider

Dans un premier temps, il est logique de s'intéresser aux solutions mises à disposition par notre fournisseur Cloud. Dans le contexte d'Azure, c'est Azure Monitor, solution de type SaaS, nativement présente sur la plateforme Azure, rapide à mettre en œuvre et pour laquelle tout est livré sur étagère, prêt à consommer, sans aucun coût de build/run. Les clusters Kubernetes sont pris en charge par Container Insights. L'avantage de la solution est qu'elle prend en charge AKS mais aussi d'autres distributions de Kubernetes grâce à Azure ARC, que celles-ci soient déployées dans Azure, on-premise ou même chez un autre Cloud provider.



Source : Container Insights

La solution Azure Monitor présente de nombreux avantages mais peut avoir un défaut : elle est intrinsèquement liée à Azure, ce qui exclut de l'utiliser dans un contexte multicloud. Nous devons alors envisager des solutions indépendantes de notre/nos fournisseur(s) Cloud.

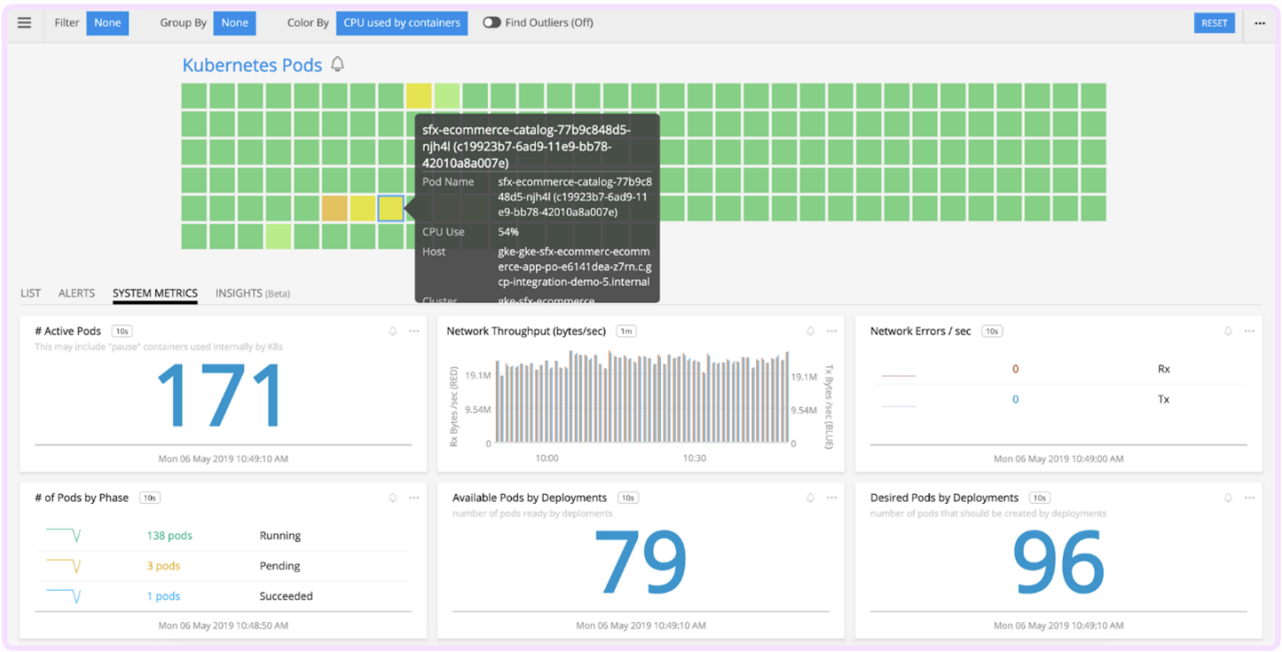
Solutions éditeurs

Dans cette deuxième catégorie de solutions, on trouve principalement des solutions de type SaaS. Cela présente de multiples avantages :

- Nous n'avons pas de coût de build/run d'une infrastructure ;
- La solution étant indépendante de notre fournisseur Cloud, nous pouvons avoir une vue complète de notre écosystème applicatif, que celui-ci soit localisé dans un datacenter ou chez n'importe quel fournisseur Cloud ;
- Les fournisseurs de ce type de solutions peuvent proposer un monitoring spécifique à certains usages comme Kubernetes ou les micro-services, ce qui peut être un accélérateur non négligeable.

Un acteur comme Splunk propose une offre SaaS couvrant tous les domaines de l'observabilité, jusqu'à la mesure de l'expérience utilisateur de nos applications. Dans le contexte de Kubernetes, la solution de Splunk se positionne à plusieurs niveaux :

- La plateforme en elle-même (Kubernetes) ;
- Les hôtes « nodes » composant notre plateforme Kubernetes ;
- Les conteneurs « pods » qui sont instanciés sur notre plateforme Kubernetes ;
- Les micro-services présents au sein de nos « pods ».



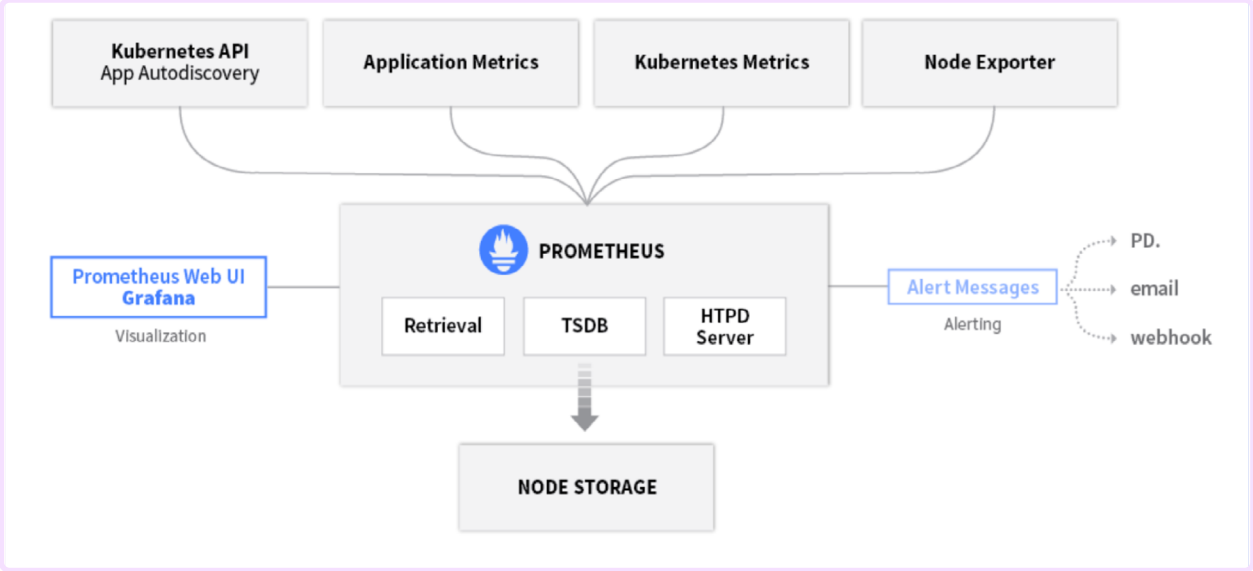
Splunk SaaS Cloud Observability suite - Kubernetes

Un autre intérêt de l'offre Splunk est qu'elle est compatible avec OpenTelemetry, un projet open-source porté par la CNCF (Cloud Computing Native Foundation) qui propose un standard unifié de collecte des données de supervision, donc un agent technique unique, indépendant de la solution de supervision que nous devons choisir. L'initiative est certes récente mais permettra à terme d'introduire une couche d'abstraction entre nos clusters Kubernetes et les solutions de supervisions.

Les solutions open-source

Reste maintenant la dernière catégorie, que l'on pourrait nommer « Do It Yourself » : des solutions open-source. La lecture du Tech-Radar de la CNCF nous permet d'avoir un panorama des solutions recommandées. On retrouve fréquemment l'association des deux projets open-source Prometheus & Grafana comme illustré ci-dessous :

D'un côté, nous avons une infrastructure Prometheus qui se charge de la collecte et du stockage des données de supervision (métriques entre autres). De l'autre, Grafana a pour fonction de proposer des dashboards. A cela, nous devons ajouter des mécanismes de notification pour informer les différents acteurs ou déclencher des réactions (WebHook). La particularité de cette infrastructure est qu'il est possible de la déployer sous forme d'applications sur notre cluster Kubernetes. Le principe peut sembler intéressant mais cela **implique déjà une certaine expérience dans le domaine de Kubernetes**. Autre point d'attention : cette architecture ne prend pas en compte les logs produits par Kubernetes ou ceux de nos applications. Ce n'est donc pas un simple assemblage de deux solutions open-source mais bien un **écosystème que nous devons construire**.



Stack Prometheus + Grafana

Ces solutions sont intéressantes mais beaucoup d'entre elles se focalisent uniquement sur Kubernetes. Or, K8s n'est qu'un composant de notre stratégie applicative. Les clusters Kubernetes hébergent nos applications et consomment beaucoup de services externes (stockage, bases de données, secrets) qui doivent eux aussi être pris en compte dans notre stratégie. Pour cette raison, **choisir une solution uniquement spécialisée sur Kubernetes n'est pas forcément pertinent**.

L'essentiel à retenir sur la stratégie de supervision

Le choix d'un produit en lui-même n'est pas le plus important. L'essentiel est de **développer SA stratégie de supervision/observabilité**. On part de données « brutes » issues d'Azure et de Kubernetes. Toutes les solutions savent les collecter. Ce qui compte c'est comment nous raffinons ces données pour produire des indicateurs qui permettent une prise de décision éclairée par différents acteurs :

- Les équipes en charge de la gestion de nos infrastructures Cloud ;
- Les équipes en charge des clusters Kubernetes ;
- Les équipes en charge des applications ;
- Les décideurs métiers.

Chacune de ces équipes a des attentes différentes. Au même titre que le DevOps, l'observabilité est une culture que l'on développe : cela prend du temps. D'un point de vue purement opérationnel, il n'est pas envisageable de mettre en production une plateforme Kubernetes sans solution de supervision/observabilité. Une stratégie peut donc impliquer de commencer avec des solutions de type SaaS qui permettent de nous focaliser sur Kubernetes et nos applications, et non sur le maintien en condition opérationnelle de la plateforme. Si en plus, cela peut être la solution que nous utilisons déjà pour Azure, nous nous économisons un temps d'apprentissage, ce qui n'est pas négligeable.

SÉCURISER SON SERVICE KUBERNETES

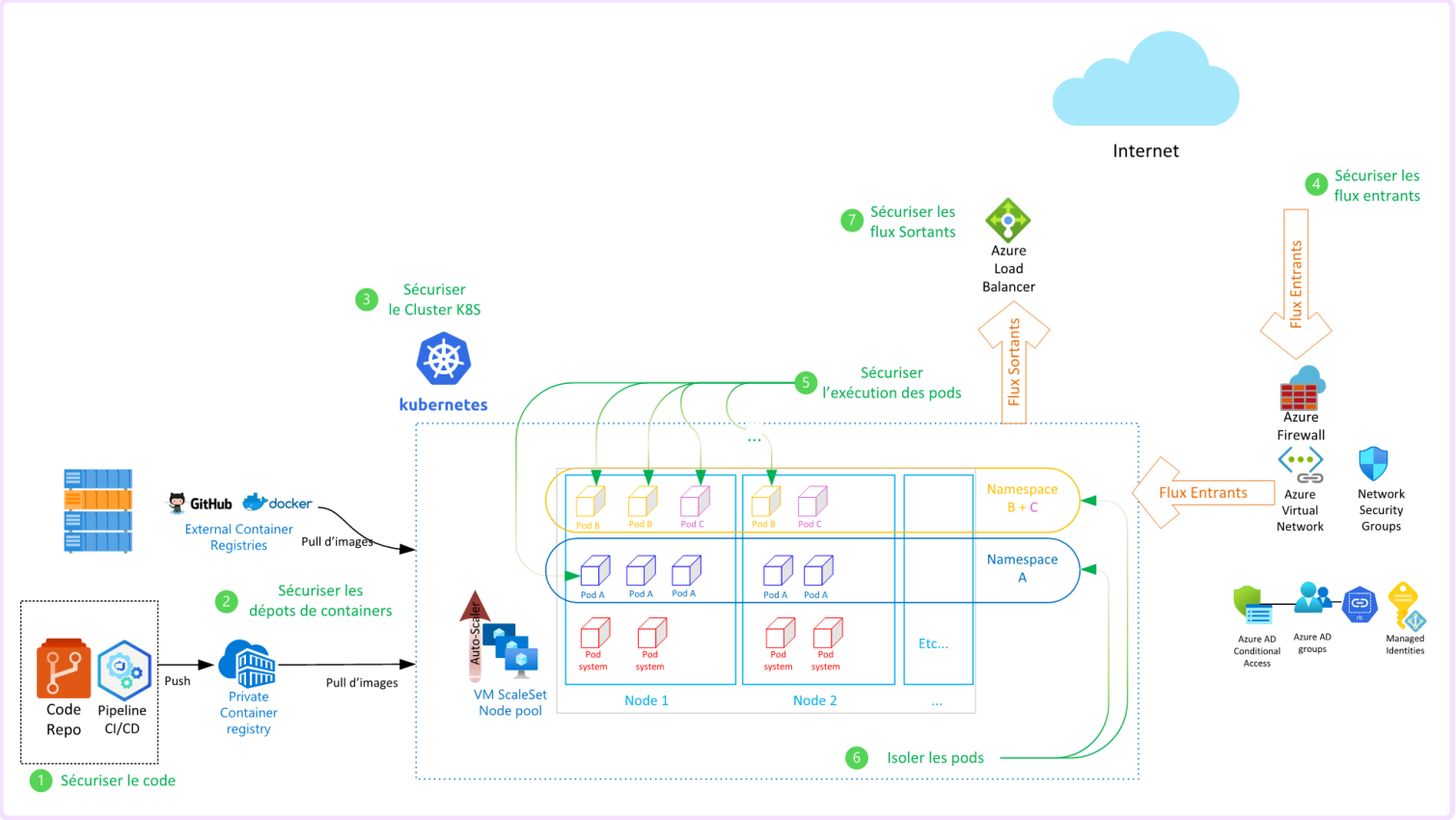
Si vous devez retenir un élément de ce chapitre, c'est le suivant : **par défaut, Kubernetes ne gère pas la sécurité**.
Cela ne signifie pas que sécuriser Kubernetes est impossible, mais il est primordial de savoir que **cette sécurité est entièrement entre les mains de l'utilisateur**.
C'est à vous qu'il appartient de prendre les mesures pour sécuriser votre cluster.
Et cette sécurisation est d'ailleurs vraie quelle que soit la distribution de Kubernetes, que celle-ci soit on-premise ou un système managé comme Azure Kubernetes Services (AKS) ou d'autres équivalents chez les autres Cloud Solution Providers (CSP).

Facteur aggravant, la popularité de Kubernetes en entreprise fait que les attaquants redoublent d'efforts pour chercher des failles et des angles d'attaques. Il serait donc vain d'essayer de lister exhaustivement toutes les failles ou les modalités des attaquants qui évoluent sans cesse.
Ce chapitre énumère les **points essentiels pour appréhender la sécurisation de vos clusters et donne des pistes de réflexions pour poursuivre les recherches**.

Nous n'oublierons pas que la sécurité ne consiste pas uniquement à se protéger des attaques. Sécuriser son service Kubernetes, c'est aussi **assurer la sécurité des données et la reprise d'activité en cas de défaillance**.

Surface d'attaque

Voici la représentation schématisée d'un cluster AKS ainsi que son environnement immédiat.
Les points à sécuriser, identifiés en vert sur le schéma, seront approfondis dans cette partie.



Point n°1 : Sécuriser le repository de code source et la chaine d'intégration

Gestionnaire de Code Source

Avant de sécuriser le cluster qui exécutera les binaires, il est primordial de **sécuriser le gestionnaire de code source** (ou repository ou gestionnaire d'artéfacts…). En effet, quiconque ayant accès à ce repository en lecture pourra glaner des informations essentielles sur la topologie de votre infrastructure. Et si un attaquant venait à prendre la main sur ce gestionnaire de code source, il pourrait pousser n'importe quelle modification pour exfiltrer les données ou utiliser votre infrastructure dans des buts malveillants (minage de cryptomonnaie, attaques DNS, etc.).

Certaines recommandations sont évidentes mais il est bon de les rappeler : **ne jamais stocker d'identifiants ou d'éléments confidentiels dans votre repository de code source**.

L'attaque du code Source SolarWinds rappelle l'importance de sécuriser son code source. Cette attaque a compromis les données des clients de SolarWinds parmi lesquels de nombreux gouvernements et grandes entreprises.

Chaine d'intégration

La majorité des personnes ayant travaillé sur une instance Kubernetes pourra en témoigner : parfois, après une mise à jour ou à la suite d'une erreur humaine ou même d'une modification anodine, le service peut dérailler. Dans cette situation, essayer d'identifier et corriger une instance de production peut se compter en heures voire bien plus.

Souvent, la solution la plus simple, la plus rapide et la plus sécurisée est de créer une nouvelle instance, s'assurer qu'elle est fonctionnelle et effectuer une bascule avant de détruire l'instance défaillante. Pour ce faire, il est primordial de constituer le plus tôt possible une chaine d'intégration **CI/CD** (CI : Continuous Integration / CD : Continuous Deployment,) et si possible **IaC** (Infrastructure as Code).

Dev + Ops : la sécurité est l'affaire de tous

L'erreur principale commise par les membres d'une équipe DevOps est de penser que le problème sera géré par le maillon suivant dans la chaîne. Pressé par le temps, un développeur se dira que son Tech Lead vérifiera sa livraison. Le Tech Lead estimera que c'est l'administrateur qui est responsable et l'administrateur pensera - à tort - que le développeur et le Tech Lead ont déjà vérifié.

> **La sécurité, c'est l'affaire de tous et non d'un seul.**

Il est ainsi primordial que les développeurs et les opérateurs (Dev + Ops) impliqués sur la plateforme Kubernetes soient tous formés à la sécurité.

Point n°2 : Sécuriser les registries de conteneurs et les images

Les « containers registry » entreposent les différentes versions des images de conteneurs. Les clusters Kubernetes pourront lancer une commande « pull » sur ces registries pour télécharger une version d'une image.
Ces registries peuvent être privés, managés par votre entreprise ou un fournisseur Cloud. Mais ils peuvent également être publics, comme Docker Hub ou GitHub.

Deux problématiques viennent s'appliquer à ces containers :

- La fiabilité des images externes ;
- La sécurisation des images de conteneurs créées par vos équipes.

Outils de recherche de vulnérabilités

Le contenu de nos images est rarement exempt de failles. Celles-ci sont dues soit aux images que nous utilisons pour construire soit à la présence de vulnérabilités introduites dans notre côté applicatif. Ces outils exploitent les rapports de publication des CVE (Common Vulnerabilities and Exposures). Cette analyse doit être industrialisée pour permettre de réagir au plus vite. Nativement, Microsoft propose cette fonctionnalité au sein de sa solution de registry (Azure Container Registry). Cette dernière propose une intégration avec des solutions comme Twistlock ou Aqua pour réaliser ces recherches.

Images de containers externes

Comme avec n'importe quel logiciel, il existe un risque qu'une personne mal intentionnée ait pu compromettre le code source ou les binaires. De plus, il peut aussi arriver que des « containers registry » publics de confiance soient eux-même compromis et diffusent des images malveillantes.
Enfin, outre les risques d'attaques, ces registries publics peuvent être indisponibles, cesser leur activité ou tout simplement ne plus proposer la version de conteneur nécessaire à vos services.

Pour prévenir ce risque, il est **préférable de ne pas utiliser directement les sources externes** mais de télécharger ces images dans un registry local à votre SI. On créera pour cela un pipeline CI/CD spécifique à la récupération des sources externes.
Ce pipeline pourra également valider des checksums (souvent disponibles sur les sites de l'éditeur), faire passer et lancer des tests de vulnérabilités (cf. les outils de scans de vulnérabilités) et des tests unitaires automatisés.
Non seulement ces pipelines pourront protéger des attaques mais ils permettront également de renforcer la qualité de service en évitant les downtimes liés à des breaking points sur des conteneurs externes.

Enfin, avoir les images de conteneurs externes sur une ressource que vous maitrisez est un point essentiel dans le cas d'un **plan de continuité d'activité**.

Images de containers Internes

A l'instar du gestionnaire de code source, les images issues de vos équipes de développement contiennent des informations essentielles de votre entreprise.
Il est bien évidemment conseillé de ne pas stocker de secrets directement dans les containers mais d'**utiliser des secrets dans des vaults**.
Contrairement à une idée reçue, les conteneurs ne sont pas des boites noires invulnérables. Quiconque peut exécuter le conteneur peut accéder à tous les fichiers de celui-ci. Quelle que soit la registry utilisée, il est vivement recommandé de s'**assurer que vous seul puissiez manipuler les images contenues**.

Point n°3 : Sécuriser/manager le cluster AKS

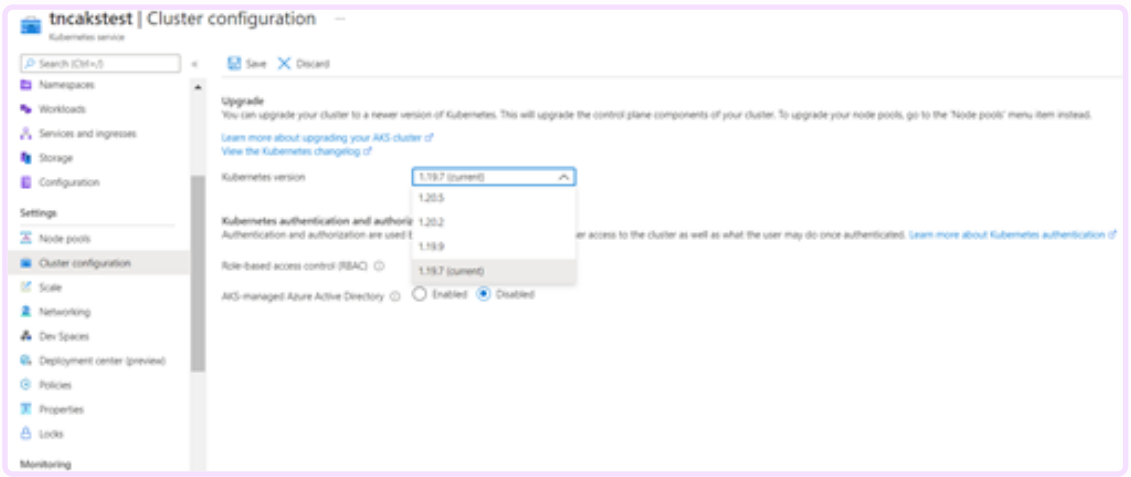
Mises à jour des versions

Version du service

Azure Kubernetes Service est un service qui bénéficie régulièrement de mises à jour par les équipes Azure. Ces mises à jour **améliorent les fonctionnalités du service** et **corrigent des failles de sécurité**.
Elles sont assurées sur les 3 dernières versions du service :

- Les deux dernières versions stables ;
- La version preview.

Les mises à jour corrigeant les failles de sécurité sont automatiquement déployées chaque nuit dès qu'elles sont disponibles. En revanche, les mises à jour de versions doivent être gérées par les administrateurs AKS via le portail ou des commandes Azure CLI.



De ce fait, toute ressource Azure Kubernetes Service doit avoir une de ces trois dernières versions afin de bénéficier de ces ajouts : en deçà de ces trois versions, la version est considérée comme dépassée. La mise à jour peut ne pas être possible. C'est pourquoi **il est primordial de maintenir ses clusters Kubernetes à jour**. Pour AKS, nous y sommes aidés car ce processus est pris en charge par la plateforme : https://docs.microsoft.com/en-us/azure/aks/upgrade-cluster.

Mises à jour des OS sur les nodes

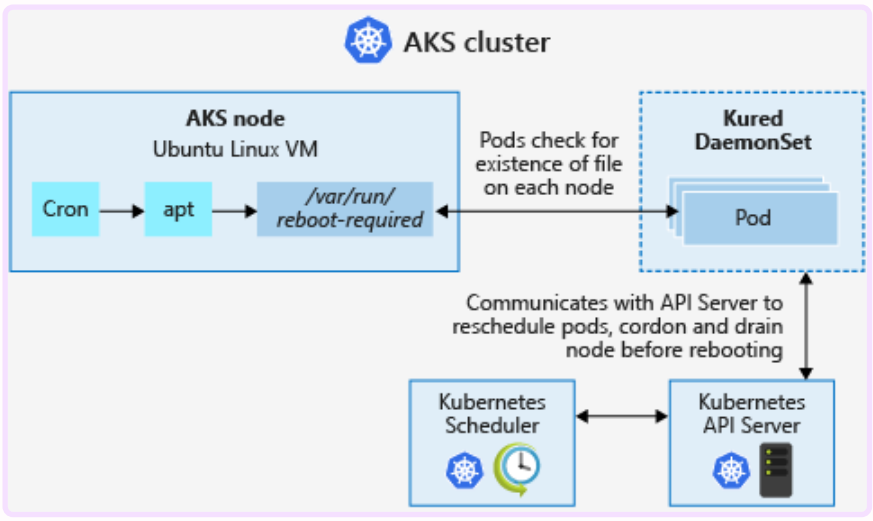
Nos cluster Kubernetes reposent sur des nodes Linux. Même si AKS prend en charge le déploiement des mises à jour, un redémarrage est souvent nécessaire pour finaliser l'opération. Celui-ci peut être automatisé en utilisant l'outil Kubernetes Reboot Daemon, plus communément appelé **Kured**.
Le processus peut être planifié en tenant compte de plusieurs critères :

- Les reboot-days : jours où les reboots sont autorisés ;
- Le time-zone : fuseau horaire concerné ;
- Le start-time : heure de début de la plage horaire ;
- Le end-time : heure de fin de la plage horaire.

Mises à jour des images sur les nodes

Les équipes AKS fournissent régulièrement de nouvelles versions des images. Il sera donc intéressant pour les équipes en charge des clusters d'effectuer régulièrement des mises à jour pour les images dans les nodes. Cela peut se faire via des commandes sur **Azure CLI** pour une mise à jour totale ou partielle sur l'ensemble de vos nodes.

Pour en savoir plus, nous vous invitons à consulter la documentation Microsoft sur la mise à jour des images sur les nodes.



Source : https://docs.microsoft.com/en-us/azure/aks/node-updates-kured

Sécuriser l'accès au cluster AKS

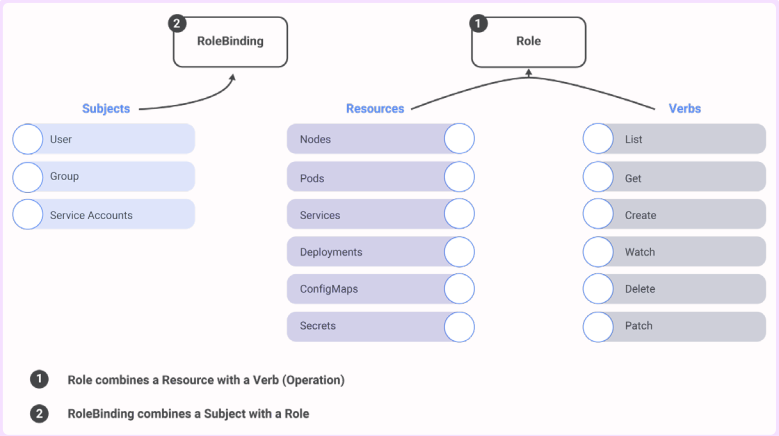
Le contrôle d'accès au cluster AKS et à ses ressources peut être géré de deux manières :

- Limiter au strict nécessaire les accès aux utilisateurs, groupes ou comptes de service selon les besoins de ces derniers avec **Kubernetes RBAC** (Role-Based Access Control) ;
- Gérer les structures de sécurité et d'autorisation avec **Azure Active Directory**.

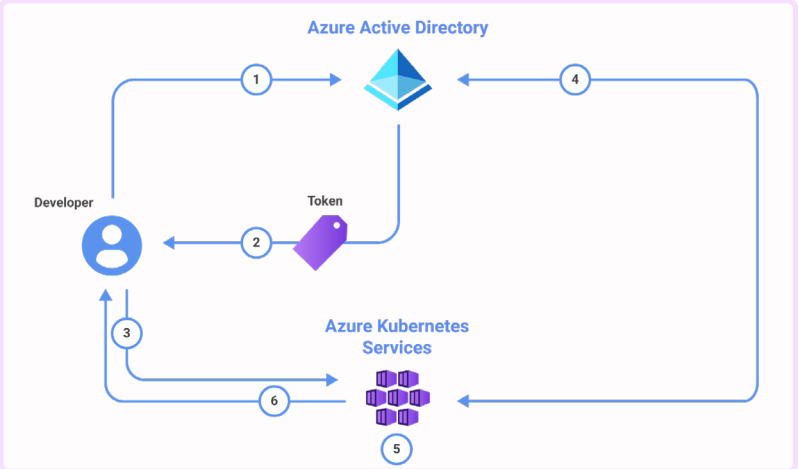
Kubernetes RBAC

Cet outil de contrôle permet :

- La création de rôles pour définir des autorisations ;
- L'affectation des utilisateurs ou groupes à ces autorisations d'opérations précises (affichage des logs, création ou modification des ressources…) ;
- La limitation plus ou moins partielle de ces autorisations sur l'ensemble du cluster.



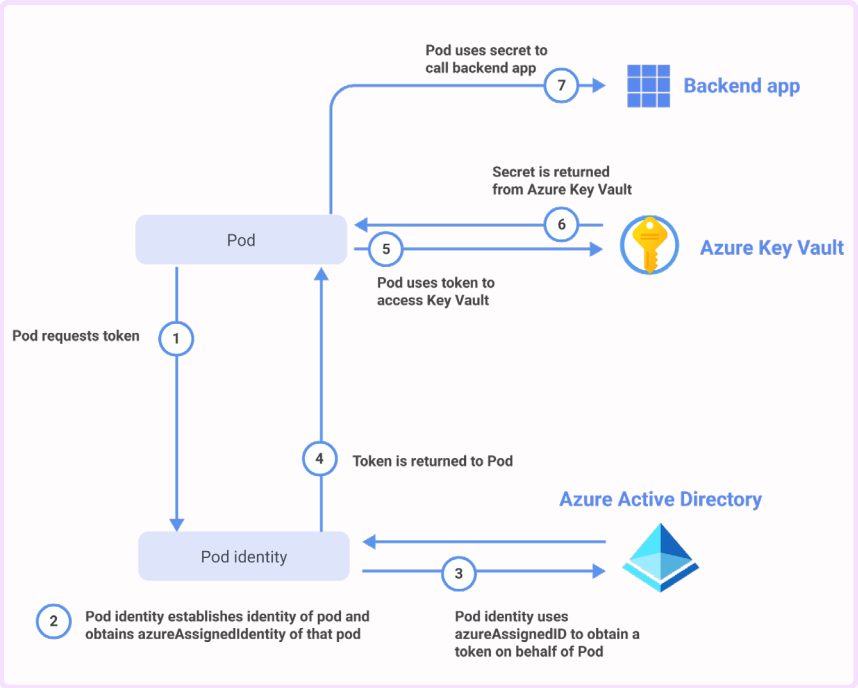
Azure Active Directory



Microsoft proposer d'intégrer Azure Active Directory comme référentiel d'authentification et d'autorisation au sein du cluster AKS. Cette intégration peut même être utilisée pour intégrer des fonctionnalités comme Multi-Factor Authentication (authentification forte) ou Conditional access.

Sécurité des pods

La sécurité s'insinue jusqu'au niveau pod. Il va de soi que le stockage d'informations sensibles (certificats, chaines de connexion à une base de données…) directement dans le pod est à proscrire. Ces éléments doivent être externalisés du code. On y fera référence au travers de la notion de secrets dans Kubernetes.



Parmi les autres bonnes pratiques à ce niveau, on peut citer :

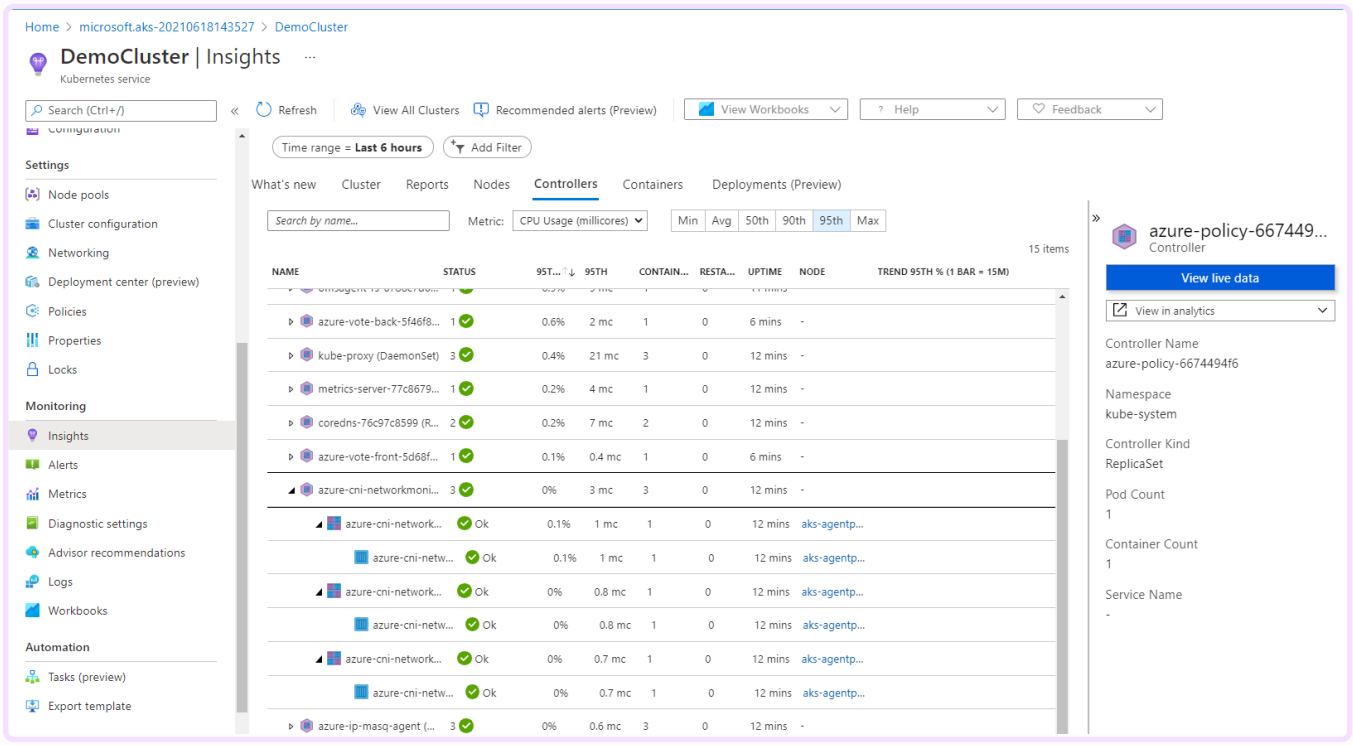
- Limiter l'accès aux processus, services et à l'élévation de privilèges en utilisant les contextes de sécurités des pods ;
- Sauvegarder et récupérer les informations d'identification dans un coffre numérique notamment un Azure Key Vault ou une solution tierce comme Hashicorp Vault.

Automatisation et Back-up

Azure Kubernetes Service est un service orchestrant plusieurs ressources simultanément afin d'obtenir un environnement d'exécution de micro-services interdépendants. **Ses multiples possibilités de configurations variées peuvent entraîner des conséquences sur la sécurité**. En conséquence, **il est recommandé d'avoir la main sur tous les paramètres, ce qui implique de déployer en Infrastructure as Code (IaC)**. L'industrialisation du déploiement va nous permettre d'améliorer la sécurité de notre infrastructure. Ce même processus de déploiement sera utilisé pour réaliser les montées de versions. Il est en effet plus sûr et plus simple de construire une nouvelle infrastructure que de procéder à sa mise à niveau.

Monitoring

Afin d'assurer la supervision de vos applications hébergées sur AKS, Azure dispose de l'outil Azure Monitor. Celui-ci permet de surveiller les performances de votre cluster, ainsi que les ressources consommées par les pods composant vos applications.



Point n°4 : Sécuriser les données entrantes

Les protections par défaut

Par défaut, aucune restriction réseau n'est appliquée, que ce soit pour notre cluster Kubernetes ou même les pods composant nos applications. Azure Kubernetes Services prend en charge deux architectures réseau principales :

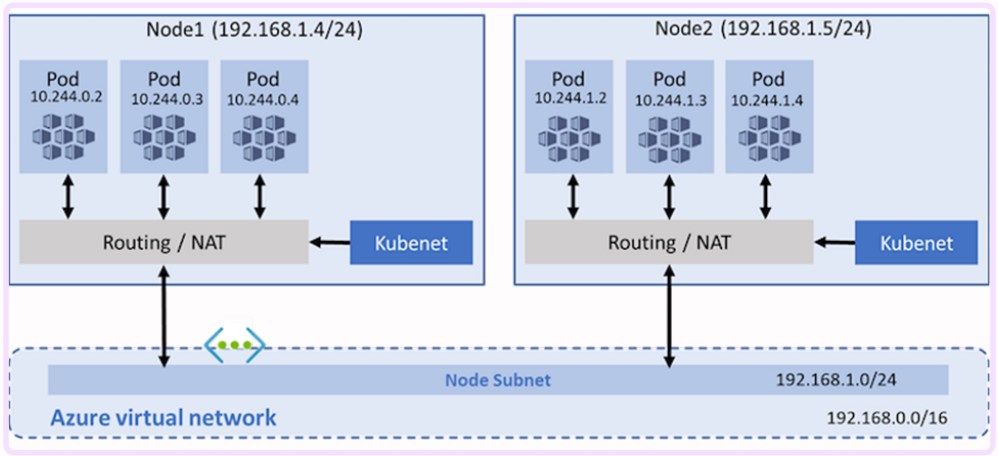
- **Kubenet** : par défaut. Chaque nœud et chaque service ont une adresse IP dans le même VNet Azure. Les pods reçoivent une adresse IP dans un sous-réseau mais ne sont pas accessibles directement.
Les requêtes doivent passer par le pod Kubenet du node.
L'avantage de Kubenet c'est que seuls les nodes et les services Kubenet sont exposés. La solution présente cependant quelques inconvénients :
  - Maximum de 110 pods par node ;
  - Maximum de 400 nodes par cluster ;
  - Seuls des nodes Linux sont autorisés.

- **Azure CNI**

Chaque nœud et chaque pod reçoivent une adresse IP qui devra être unique dans le VNet Azure. Azure CNI permet d'utiliser des machines Windows et jusqu'à 250 pods par node. Mais trois inconvénients surviennent :

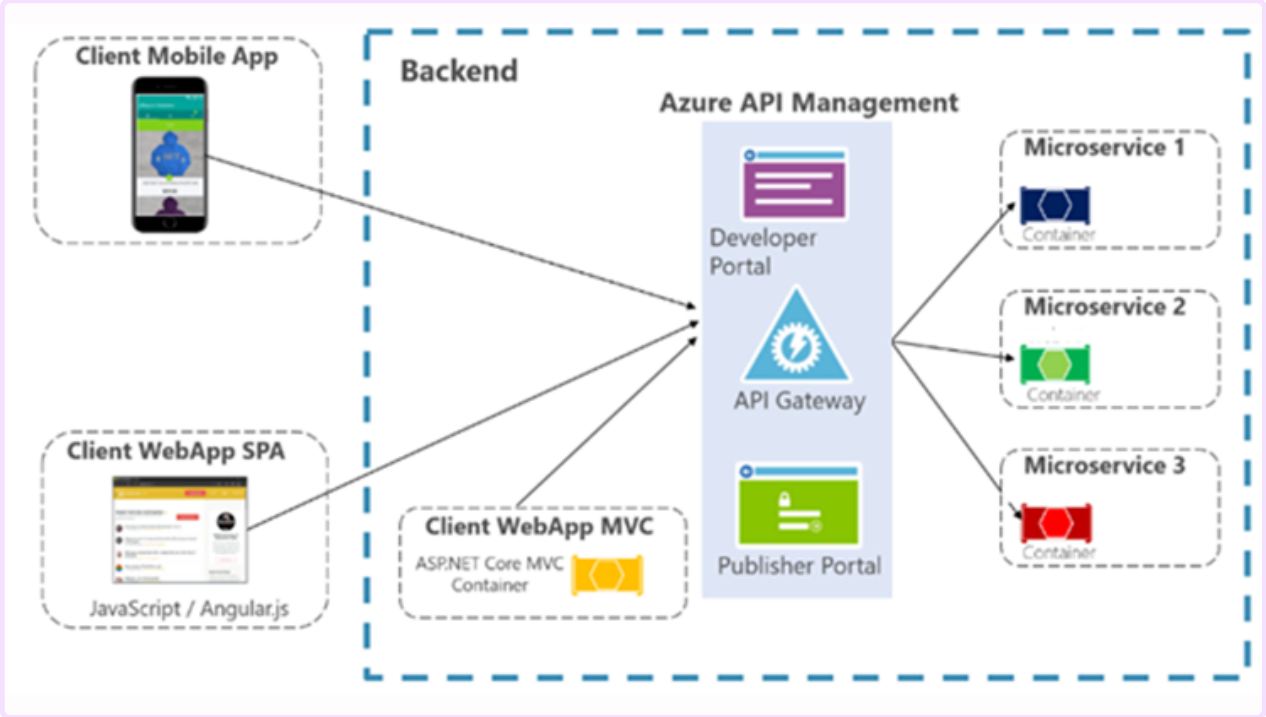
Dans Azure Kubernetes, la mise en œuvre des mécanismes de filtrage des flux réseaux est à notre charge. On peut travailler à plusieurs niveaux :

- Maîtriser l'évasion Internet des nodes composant notre cluster AKS en appliquant un filtrage réseau avec Azure Firewall ;
- Maîtriser les flux entrants avec un Ingress comme NGINX ou même utiliser la solution AGIC de Microsoft qui permet de déporter ce filtrage sur le service Azure Application Gateway ;
- Maîtriser les flux réseaux au niveau podavec des network policies ou des solutions tierces plus évoluées (Istio, Linkerd, Consul…).



Source : documentation Microsoft

Enfin, on pourra aussi travailler l'exposition des APIs de nos applications avec une solution comme **Azure API Management** qui peut être déployée comme service PaaS ou même comme pod au sein de notre cluster AKS. APIM va nous permettre de gérer l'exposition des APIs de nos applications, pour des usages internes mais aussi externes. Le service joue le rôle de centralisateur d'accès et permet d'authentifier et autoriser les usages.



Monitoring et logs

Comme tout service Azure, Azure Kubernetes Service produit des métriques et des logs avec différents niveaux de détails. Le monitoring de notre cluster AKS peut être réalisé avec la solution Azure Monitor, des solutions open-source ou même des solutions de type SaaS. Ces points ont été abordés dans les précédents chapitres de ce livre blanc. Dans le contexte de la sécurité, ce qui nous intéressera ici, c'est l'intégration avec vos outils de sécurité. La mise en œuvre d'**Azure Defender for Kubernetes** est vivement recommandée, tout comme l'**ingestion des données dans votre solution SIEM** (Azure Sentinel ou autre) pour surveiller l'activité de vos clusters.

Point n°5 : Sécuriser l'exécution des pods

Encore un mythe qui a la vie dure : la containerisation ne protège pas le contenu du conteneur (binaires, configuration et données). Vos applications doivent être conçues selon le **principe du moindre privilège** et suivre des bonnes pratiques de développement sécurisé.

Voici une liste de bonnes pratiques à prendre en compte lors du développement d'une application conteneurisée :

- **Security Context** : on peut définir l'option **runAsUser** et **fsGroup** pour assumer les autorisations appropriées. Cela va définir et contrôler les droits que l'application va revendiquer en s'exécutant, et ainsi éviter l'accès à des services et processus internes K8s ou l'accès aux nœuds sous-jacents ;

- **Privilege Escalation : ne pas donner des privilèges administrateur (root) en les désactivant** dans votre application. Toujours développer votre application en utilisant le principe des moindres privilèges afin de réduire au maximum la surface d'attaque ;

- Limiter l'exposition des informations d'identification avec un **coffre-fort numérique** ;

- Utiliser des **comptes de service**. Eviter l'usage de clés SAS, de comptes de base de données, etc. Tous les Cloud providers proposent des solutions de comptes de service permettant de déléguer la gestion des droits vers le service Cloud et un système de gestion d'identité. Un compte de service (dans Azure : Managed Identity) permet à un pod de s'authentifier lui-même auprès de services compatibles. Par exemple : un conteneur peut se connecter à un coffre en utilisant sa propre identité et un jeton qu'il va demander à un système de gestion d'identité. Cette approche permet d'éviter la gestion des secrets pour se connecter à d'autres services.

Point n°6 : Segmenter les applications

Par défaut, le réseau d'une plateforme Kubernetes est ouvert à tout le trafic. Tous les pods communiquent entre eux.

Pour maitriser et sécuriser une plateforme Kubernetes, **il est impératif d'isoler les pods en n'autorisant que le strict minimum**. On garantit ainsi que si un service est attaqué ou un pod compromis, le reste de la plateforme n'est pas mis en danger.

La segmentation des applications peut être réalisée à plusieurs niveaux. La mise en œuvre de namespaces est un premier niveau d'isolation basé sur les RBAC de Kubernetes. On pourra envisager un second niveau d'isolation réseau en mettant en place des network *policies* (Calico & Weave par exemple). Enfin, nous pouvons travailler finement au niveau pod en intégrant une solution de type Service Mesh (Open Service Mesh, Istio…) qui va permettre de gérer la découverte des pods entre eux ainsi que régenter la possibilité de communiquer entre eux.

L'essentiel à retenir sur la sécurité de Kubernetes

Comme nous venons de le voir, Kubernetes est un domaine très vaste. Comme indiqué au début de ce chapitre, **la sécurité de Kubernetes est entre vos mains**. On doit adresser le sujet aussi bien au niveau de la plateforme (notre cluster Kubernetes) que des applications que nous hébergeons dessus. Certains de nos choix auront un impact sur nos applications. De ce fait, **on ne peut dissocier la sécurité de la plateforme de la dimension applicative**.
Il faudra procéder par itérations successives pour atteindre un niveau de sécurité acceptable pour opérer notre plateforme en production. Le passage en production ne marque pas une fin en soi, mais plutôt le début d'un nouveau cycle d'itérations visant à améliorer le niveau de sécurité de notre plateforme Kubernetes et des applications que nous y hébergeons.
Enfin, on retiendra que **la sécurité est l'affaire de tous** et qu'il faut sans cesse se tenir informé de l'actualité dans ce domaine en suivant les CVE (Common Vulnerabilities and Exposures) sur le sujet, ce que ne manquera pas de faire tout responsable de la sécurité informatique de votre organisation. D'ailleurs, il y a de grandes chances que celui-ci s'intéresse aussi à Kubernetes mais en utilisant une autre grille d'analyse : la matrice des menaces autour de Kubernetes. A vous de définir ensemble quels risques doivent être adressés, quels moyens à mettre en œuvre pour les réduire ou au mieux les atténuer.

09 CONCLUSION

Comme nous avons pu le voir Kubernetes est un écosystème complexe. En un peu plus de six ans d'existence, la plateforme s'est positionnée comme LE socle pour héberger nos futures applications Cloud-Native. Sommes-nous au pic du battage médiatique ? Chez Cellenza, nous pensons que non car les Cloud Solution Providers que sont Azure AWS et GCP investissent massivement dessus et l'écosystème qui gravite autour ne cesse de grandir. Nous pensons que la plateforme va s'inscrire dans le temps pour construire une plateforme applicative avec laquelle les dépendances tendront à disparaître car la plateforme deviendra une commodité comme tout autre service Cloud mis à disposition. C'est à nous qu'il revient de construire « NOTRE » plateforme applicative que nous devrons sécuriser. Même avec AKS, nous ne sommes pas exclus de toute responsabilité dans le modèle de responsabilité partagée.

Pour nos applications, même s'il est techniquement possible de les porter à moindres frais (lift and shift) sur notre plateforme, nous ne pourrons exploiter pleinement ses capacités que si nous adoptons aussi les usages et pratiques des applications Cloud-Native. C'est en portant de plus en plus d'applications sur notre plateforme Kubernetes que nous pourrons obtenir un retour sur investissement. Plus ces applications partageront les mêmes fondations, plus nous pourrons bénéficier du retour d'expériences des premières applications et donc accélérer.

Les auteurs

Wael Amri

Consultant Azure
cellenza

Adnan El Akkaoui

Consultant Azure
cellenza

Didier Esteves

Azure & AKS Architect
cellenza

Mustapha Ferhaoui

Consultant Azure
cellenza

Coralie Martinez

Business Manager
cellenza

Sylvain Martinez

Technical Officer
cellenza

Ricardo Piteira

Architecte Cloud & Sécurité
cellenza

Benoit Sautière

Senior Technical Officer
cellenza

Hendrick Johann Tankeu

Consultant Azure
cellenza

Jérôme Thin

Consultant Azure
cellenza

Sébastien Thomas

Architecte SI
cellenza

Maxime Villeger

Cloud Solution Architect
Microsoft

Qui sommes-nous ?

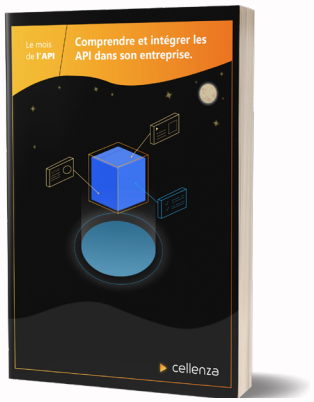
cellenza

Cellenza est un cabinet de conseil d'expertise technique et de réalisations. Experts des technologies Microsoft et des bonnes pratiques agiles, nous intervenons sur toute la chaine de valeurs IT : conseil, développement d'applications, coaching, transfert de savoir-faire et formation.

Pour en savoir plus : **www.cellenza.com**

Suivre Cellenza sur **Linkedin**

NOS DERNIERES PUBLICATIONS



Retrouvez aussi l'ensemble des articles techniques
rédigés par nos Cellenzans :

blog.cellenza.com

cellenza

156 Boulevard Haussmann - 75008 PARIS
Tél : +33 (0)1 45 63 14 29 - www.cellenza.com

Rejoignez-nous sur

Crédits photos et illustrations :
vecteezy.com / Freepik.com / Microsoft / CNCF