



ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE ET
D'ANALYSE DES SYSTÈMES - RABAT

Conception et mise en oeuvre d'un éditeur Graphique pour Kubernetes

Réalisé par :

EL KARMY OTHMANE
EL JAZOULY ANASS
Filière :GL

Encadré par :

Mr.HAMLAOUI MAHMOUD

Membre du jury :

Mr.HAMLAOUI MAHMOUD
Mr.BAINA KARIM

Année académique 2021/2022



Remerciements :

Au terme de ce travail, je saisis cette occasion pour exprimer mes vifs remerciements à toute personne ayant contribué, de près ou de loin, à la réalisation de ce projet.

L'expression de notre haute reconnaissance à Mr.Hamlaoui Mahmoud qui nous a encadré avec patience tout au long du projet et n'a épargné aucun effort pour mettre à notre disposition les explications et la documentation nécessaires. Son assistance prestigieuse et ses conseils nous ont été d'un précieux apport.

Nous exprimons également notre gratitude aux membres du jury, qui nous ont honorés en acceptant de juger ce travail. Enfin nous tenons à remercier l'ensemble du corps enseignant de l'école Nationale Supérieure d'Informatique et d'Analyse des Systèmes.

Résumé :

Kubernetes, également connu sous le nom de K8, est un système open source pour la gestion des applications conteneurisées sur plusieurs hôtes. Il fournit des mécanismes de base pour le déploiement, la maintenance et l'échelle des applications.

Notre projet consiste à élaborer un éditeur graphique pour kubernetes . Ce document a pour but de décrire le déroulement de notre projet. Nous vous présenterons donc tout au long du rapport, les étapes de la réalisation de ce projet, ainsi que les outils que nous avons utilisés.

Au premier lieu, nous allons présenter le contexte général du projet, ensuite nous exposerons l'avancement de cette technique à travers l'état de l'art, on mettra en évidence les phases d'analyse et de conception et finalement la réalisation.

Abstract :

Kubernetes, also known as K8s, is an open source system for managing containerized applications across multiple hosts. It provides the basic mechanism for deploying, maintaining, and scaling applications.

Our project is to develop a graphical editor for Kubernetes. This document is intended to describe the process of our project. Throughout the report, we will describe the steps to implement the project and the tools we used.

First we present the general background of the project, then we present the progress of this technology through the state-of-the-art, we emphasize the analysis and design phases, and finally the implementation.

Table des matières

Introduction	1
1 Contexte général du projet	2
1.1 Présentation du projet	3
1.2 Problematique	3
1.3 Objectif du projet	3
2 État de l'art	4
2.1 Introduction	5
2.2 Workflow Agile	5
2.3 Pratiques d'agiles	6
2.4 CI/CD	6
2.4.1 Continuous integration	6
2.4.2 Continuous delivery	7
2.4.3 Continuous deployment	7
2.5 DevOps	8
2.5.1 Definition	8
2.5.2 Relation entre DevOps et CI/CD	8
2.5.3 Secteur d'automatisation	9
2.5.4 Effet du DevOps à grand echelle	9
2.6 L'emergence de l'architecture en microservice	11
2.6.1 Architecture Monolitique	11
2.6.2 Architecture Microservice	11
2.7 Docker	13
2.7.1 Definition	13
2.7.2 Docker Containers et machines virtuelles	13
2.7.3 les avantages de Docker	13
2.7.4 Les inconvénients de Docker	14
2.7.5 Architecture de Docker	15
2.7.6 Le rôle de Docker dans le DevOps	16
2.8 kubernetes	18
2.8.1 Kubernetes comme étant un orchestrateur	18
2.9 kubernetes et Docker	19
2.9.1 kubernetes architecture	20
2.9.2 Pods	20
2.9.3 Déploiements	21
3 Analyse et conception	24
3.1 Analyse	25

3.1.1	Cahier des charges	25
3.1.2	Analyse de l'existant : kubectl-neat	25
3.1.2.1	Présentation de la solution	25
3.1.2.2	Problématique résolue	26
3.1.2.3	Point en Commun	26
3.2	Conception	26
3.2.1	Diagramme de classe	26
4	Réalisation de l'interface graphique	28
4.1	Outils	29
4.1.1	Eclipse Modeling Framework (EMF)	29
4.1.1.1	Ecore	29
4.1.1.2	.genmodel	30
4.1.2	Eclipse Xtext	30
4.1.3	LaTeX	30
4.1.4	Visual Studio	31
4.2	Developpement	31
4.2.1	Ecore	31
4.2.2	.genmodel	32
4.2.3	La grammaire	33
4.3	Tokens	39
	Conclusion et perspectives	42

Liste des tableaux

2.1	Fréquence de déploiement des logiciels dans diverses entreprises[1]	10
-----	---	-----------	----

Table des figures

1.1	Exemple d'erreur Kubernetes	3
2.1	Workflow Agile	5
2.2	Continuous Integration flow	7
2.3	Workflow CI/CD	8
2.4	Etape de developement en DevOps	9
2.5	Difference entre un architecture Monolitique et en Microservice	12
2.6	Logo de Docker	13
2.7	Les Conteneurs Docker en Comparaison avec les machine virtuelle	14
2.8	Architecture de Docker	15
2.9	Les principaux technologies de DevOps	17
2.10	Logo de Kubernetes	18
2.11	Kubernetes Cluster	20
2.12	Page d'Accueil	21
2.13	Pod Kubernetes	21
2.14	Exemple de Deployment	22
3.1	Résultat du Kubectl-neat	25
3.2	Diagramme de classe	27
4.1	Logo EMF	29
4.2	Logo du logiciel Eclipse	30
4.3	Logo du LaTeX	31
4.4	Logo de Vs code	31
4.5	Ecore	32
4.6	Le genmodel	33
4.7	Etapes de la génération de la grammaire par Xtext	34
4.8	Exemple de l'exécution du DSL	42
4.9	Exemple de l'exécution du DSL	42

Introduction

Kubernetes est désormais la base du développement de systèmes portables et fiables couvrant les principaux clouds publics, les clouds privés et les environnements du type "bare-metal"¹. Cependant, même si Kubernetes est devenu omniprésent au point qu'on peut faire tourner un cluster dans le Cloud rapidement, il est encore beaucoup moins évident de déterminer où aller une fois qu'on a créé ce cluster. [2]

Kubernetes est désormais la base sur laquelle plusieurs applications sont construites, et il fournit une grande bibliothèque d'API et d'outils pour construire ces applications, mais il ne fournit guère à l'architecte ou au développeur d'applications d'indications ou de conseils sur la façon dont ces différentes pièces peuvent être combinées en un système complet et fiable qui répond aux besoins de l'entreprise.

Dans ce projet, nous allons présenter, dans un premier temps, le contexte général du projet. Ensuite, nous nous intéressons à son état de l'art. Dans le troisième chapitre, nous introduisons la relation entre Docker et Kubernetes et leur but dans le DevOps. Ensuite, on présentera la phase d'analyse et de conception. Enfin nous allons présenter la phase de Réalisation.

1. un système informatique matériel utilisé sans système d'exploitation complexe mais directement par un programme, généralement à l'aide d'une bibliothèque spécialisée.

1

Contexte général du projet

Ce chapitre est dédié à la description du contexte général du projet et ses objectifs.

1.1 Présentation du projet

Dans le cadre des nombreux projets proposés par l'Ensias, afin de permettre aux étudiants de développer leurs connaissances. Le projet de fin d'année est l'occasion idéale de mettre en pratique tout ce qui a été assimilé pendant l'année ou de découvrir de nouvelles notions, de valider les compétences acquises dans le cursus d'ingénierie et aussi acquérir de nouvelles, afin d'étaler nos compétences en conception et en développement. Les étudiants travaillent en binôme et bénéficient des conseils d'un professeur encadrant.

1.2 Problematique

Lors de l'utilisation de Kubernetes, on exprime dans des fichiers Yaml l'architecture qu'on souhaite adopter pour notre projet, Si on rencontre un problème il s'avère difficile ou plutôt impossible de détecter rapidement la source du problème sans oublier qu'il faut "Build" la configuration pour obtenir des logs incompréhensible.

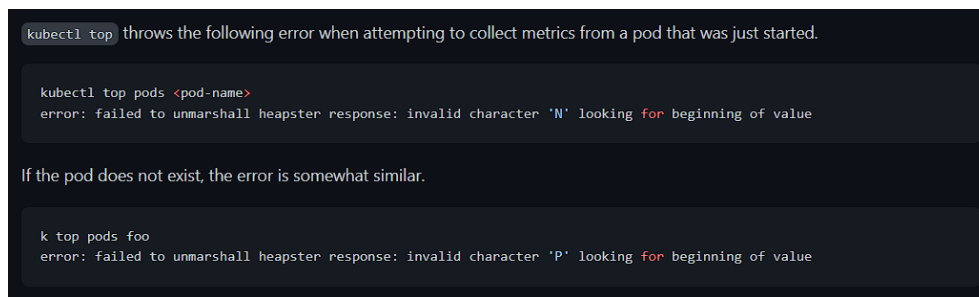


FIGURE 1.1 – Exemple d'erreur Kubernetes

«Améliorer l'expérience d'utilisateur» est l'une des principales priorités de ce projet. L'une des exigences pour se rapprocher du programmeur est de s'assurer d'une productivité accrue dans notre travail.

1.3 Objectif du projet

Notre projet traite la création d'un éditeur graphique pour Kubernetes. L'objectif est d'optimiser l'utilisation des fichiers Manifest lors de la configuration de Kubernetes, et en facilitant la création ainsi que la vérification ces fichiers, ainsi que d'apprendre les principaux bons pratiques du DevOps.

2

État de l'art

Ce chapitre a été dédié pour présenter le contexte global dans lequel le déploiement est situé, les différents aspects, et l'historique dont Kubernetes s'était inspiré, ainsi que sa relation avec les modèles et avancées théoriques auparavant réalisés dans le principe Agile et DevOps que l'homme a conçu .

2.1 Introduction

Par le passé, de nombreux domaines de l'ingénierie ont connu une sorte d'évolution notable, élevant continuellement la compréhension de leur propre travail. Bien qu'il existe des programmes d'études universitaires et des organisations professionnelles de l'ingénierie.

Pensez à la conception d'un véhicule à haute performance. Où se termine le travail de l'ingénieur mécanicien et où commence celui de l'ingénieur électricien ? une personne ayant une connaissance approfondie de l'aérodynamique doit-il collaborer avec un expert en ergonomie des passagers ? [3]

Pour qu'un domaine ou une discipline progresse et mûrisse, il doit atteindre un point où il peut réfléchir de manière réfléchie à ses origines, rechercher sur ces réflexions un ensemble de perspectives diverses et placer cette synthèse dans un contexte utile pour la manière dont la communauté se représente l'avenir [4], et donc il était sûrement nécessaire de se munir d'outils pour faciliter et améliorer cette production, et c'est ainsi que les pratiques agiles sont nées en soulignant la collaboration entre les équipes auto-organisées, multidisciplinaires et leurs clients.

2.2 Workflow Agile

Le développement logiciel agile est une approche utilisée pour concevoir un processus de gestion de logiciel discipliné qui permet également des modifications fréquentes dans le projet de développement. Il s'agit d'un type de méthodologies de développement logiciel qui constitue un cadre conceptuel pour entreprendre divers projets de génie logiciel. Elle est utilisée pour minimiser les risques en développant des logiciels dans des intervalles de temps courts, appelés itérations, qui durent généralement entre une semaine et un mois. Plusieurs idées qui se sont concrétisées avec la méthodologie Agile ont été adoptées par la suite par DevOps, comme nous l'expliquerons plus en détail dans les sections suivantes.

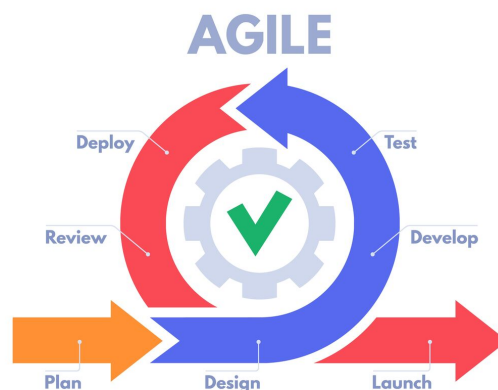


FIGURE 2.1 – Workflow Agile

2.3 Pratiques d'agiles

Il existe plusieurs domaines qui pourraient être importants pour la réussite du projet. Nous allons nous concentrer sur les pratiques agiles.

Un ensemble de chercheur [3] ont pu montrer qu'en combinant un cadre agile (Scrum) et des pratiques supplémentaires, les équipes ont pu améliorer la qualité, la productivité et la précision des estimations, et ainsi 25 pratiques agiles ont été identifiées :

Standup	Coding standards
Sprints/iterations	Refactoring
Continuous integration	Planning games
Sprint planning	Incremental design
Retrospective	Automated testing
Pair programming	Scrum master
Sprint review	User stores
Test-driven development	Architecture focus
Scrum of scrums	Burn down chart
Onsite customer	Sit together
Backlog	Enough documentation
Integration testing	Collective code ownership

Les différentes méthodes ont des pratiques différentes. Lors du choix d'une méthode pour un projet donné, ces pratiques doivent être prises en compte afin de s'assurer qu'elles correspondent au projet. Toutes les méthodes ne décrivent pas un grand nombre de pratiques, mais se concentrent plutôt sur d'autres aspects. Scrum décrit les pratiques sous forme d'événements, de rôles et d'artifacts. Extreme programming décrit 24 pratiques sous la forme de pratiques d'ingénierie. Kanban décrit cinq principes qui peuvent également être considérés comme des pratiques. Une méthode peut être considérée comme un ensemble de bonnes pratiques, de valeurs ou de principes, dont l'efficacité a été prouvée pour certains types de projets.

2.4 CI/CD

Dans cette partie, on va détailler plus sur les principes qui ont mené à la naissance du DevOps, mais tout d'abord il est nécessaire de définir un aspect primordial qui est le CI/CD.

2.4.1 Continuous integration

L'intégration continue ou bien "Continuous integration" est une philosophie de codage et un ensemble de pratiques qui incitent les équipes de développement à mettre en œuvre

fréquemment de petites modifications du code et à les enregistrer dans un référentiel de contrôle de version. La plupart des applications modernes nécessitent de développer du code à l'aide d'une variété de plateformes et d'outils. Les équipes ont donc besoin d'un mécanisme cohérent pour intégrer et valider les changements. L'intégration continue établit un moyen automatisé de construire, d'empaqueter et de tester leurs applications. L'existence d'un processus d'intégration cohérent encourage les développeurs à apporter des modifications au code plus fréquemment, ce qui améliore la collaboration et la qualité du code.[5]

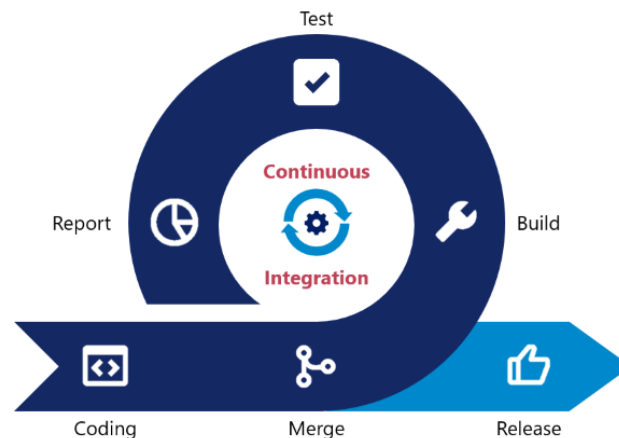


FIGURE 2.2 – Continuous Integration flow

2.4.2 Continuous delivery

La livraison continue aussi appelée "Continuous delivery" reprend là où l'intégration continue s'arrête, et automatise la livraison des applications dans des environnements sélectionnés, notamment les environnements de production, de développement et de test. La livraison continue est un moyen automatisé de pousser les changements de code vers ces environnements.

2.4.3 Continuous deployment

Le déploiement continu va un peu plus loin que la livraison continue. Avec cette pratique, chaque modification qui passe toutes les étapes de notre pipeline de production est diffusée à nos clients. Il n'y a aucune intervention humaine, et seul l'échec d'un test empêche le déploiement d'une nouvelle modification en production.

Le déploiement continu est un excellent moyen d'accélérer la boucle de rétroaction avec nos clients et de soulager l'équipe, car il n'y a plus de "jour de sortie". Les développeurs peuvent se concentrer sur la création de logiciels, et ils voient leur travail mis en ligne quelques minutes après avoir fini de travailler dessus.[5]

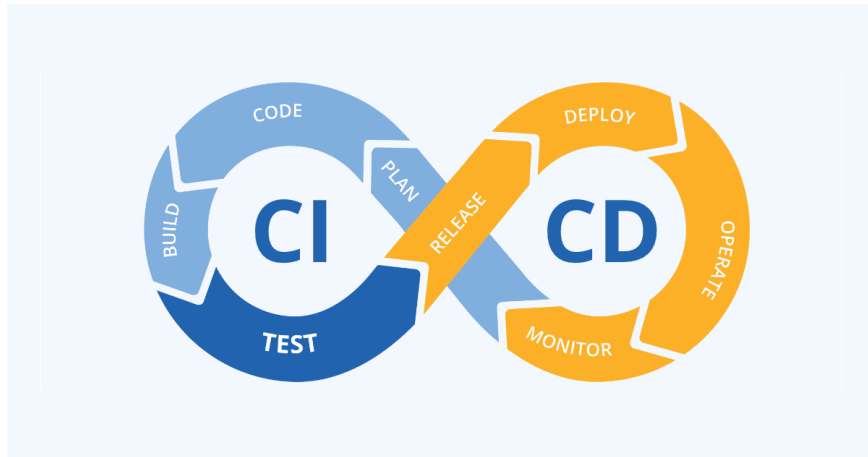


FIGURE 2.3 – Workflow CI/CD

2.5 DevOps

2.5.1 Definition

DevOps s'explique par le fait que les gens travaillent ensemble pour concevoir, construire et livrer des logiciels sécurisés à la vitesse maximale. Les pratiques DevOps permettent aux développeurs de logiciels (devs) et aux équipes d'exploitation (ops) d'accélérer la livraison grâce à l'automatisation, la collaboration, le retour d'information rapide et l'amélioration itérative.[6]

Issu d'une approche Agile du développement logiciel, un processus de livraison DevOps développe l'approche transversale de la création et de l'expédition d'applications de manière plus rapide et plus itérative. En adoptant un processus de développement DevOps, on prend la décision d'améliorer le flux et la valeur de notre application en encourageant un environnement plus collaboratif à toutes les étapes du cycle de développement.

DevOps représente un changement d'état d'esprit pour la culture informatique. En s'appuyant sur Agile, les pratiques allégées et la théorie des systèmes, DevOps se concentre sur le développement incrémental et la livraison rapide de logiciels. Le succès repose sur la capacité à créer une culture de la responsabilité, de la collaboration améliorée, de l'empathie et de la responsabilité conjointe pour les résultats commerciaux.[6]

2.5.2 Relation entre DevOps et CI/CD

Le CI/CD est un élément essentiel de DevOps et de toute pratique moderne de développement de logiciels. Une plateforme CI/CD spécialement conçue peut optimiser le temps de développement en améliorant la productivité d'une organisation, en augmentant l'efficacité et en rationalisant les flux de travail grâce à l'automatisation, aux tests et à la collaboration intégrés.

À mesure que les applications se développent, les fonctionnalités de CI/CD peuvent contribuer à réduire la complexité du développement. L'adoption d'autres pratiques DevOps aide à briser les silos de développement, à évoluer en toute sécurité et à tirer la meilleure partie du CI/CD.

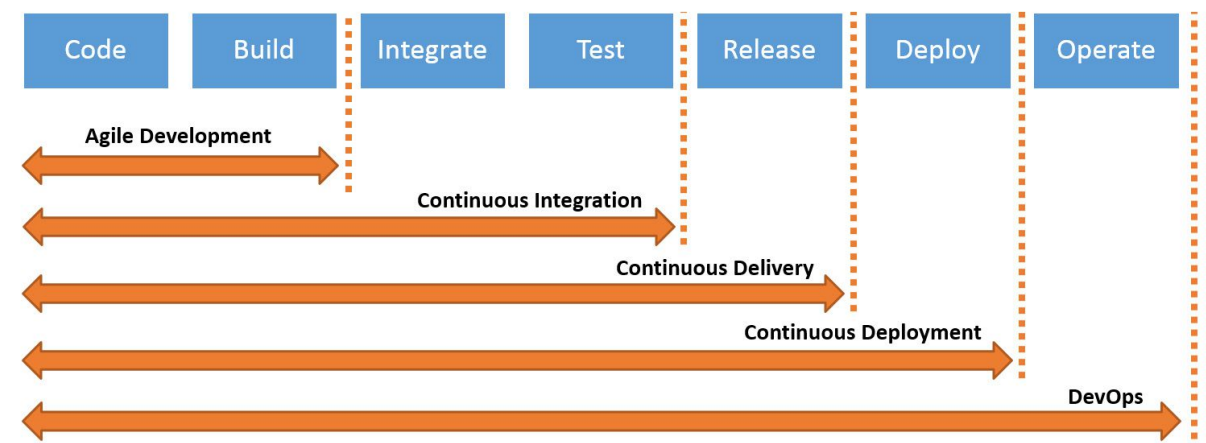


FIGURE 2.4 – Etape de developement en DevOps

2.5.3 Secteur d'automatisation

Dans le domaine des logiciels, nous pouvons distinguer différents types d'automatisation [1] :

- Installation de serveurs : consiste à installer et à configurer des serveurs par le biais de l'automatisation.
- Automatisation de l'infrastructure : il s'agit essentiellement de ce que l'on appelle IaC, un paradigme informatique orienté vers le provisionnement et le déploiement de l'infrastructure avec code, tout comme le reste de nos logiciels. Des outils tels qu'Ansible ont cet objectif.
- Automatisation des tests et des constructions : principalement, les outils d'automatisation des tests et des constructions permettent de continuer à développer sans avoir notre ordinateur portable gelé à cause d'un manque de RAM ou de CPU lors de la compilation d'un grand projet, ce type d'outils permet de connecter des nœuds esclaves dans lesquels les constructions auront lieu afin que les développeurs n'aient pas à exécuter les processus de construction localement sur leurs ordinateurs. C'est aussi une bonne pratique pour centraliser toutes les compilations d'un projet ainsi que les artefacts binaires qui en résultent, et cela ouvre la porte au monde DevOps.

2.5.4 Effet du DevOps à grand echelle

La durée de tous ces processus peut varier en fonction de la taille du projet, il est si courant qu'ils puissent être effectués plus de 5 fois par jour par développeur (voir dans le tableau suivant les fréquences de déploiement typiques) que les étapes suivies doivent être efficaces et bien structurées. L'ensemble de ces processus est conçu comme un pipeline ou pipeline DevOps, un pipeline consiste en un ensemble d'outils, de flux et de processus automatisés, permettant aux équipes de construire et de déployer des logiciels efficacement, et comprend l'intégration continue et le déploiement continu.[1]

Company	Deployment Frequency
Amazon	23,000 per day
Google	5,500 per day
Netflix	500 per day
Facebook	1 per day
Twitter	3 per week
Typical Company	1 every 9 months

TABLE 2.1 – Fréquence de déploiement des logiciels dans diverses entreprises[1]

AI/ML applique en DevOps

L'IA et l'apprentissage automatique (ML) sont encore en train de mûrir dans leurs applications pour DevOps, mais les organisations peuvent en tirer profit dès aujourd'hui, notamment en utilisant la technologie pour donner du sens aux données de test.

L'IA et l'apprentissage automatique peuvent trouver des modèles, déterminer les problèmes de codage à l'origine des bugs et alerter les équipes DevOps afin qu'elles puissent creuser davantage.

De même, les équipes DevOps peuvent utiliser l'IA et le ML pour passer au crible les données de sécurité provenant des journaux et d'autres outils afin de détecter les brèches, les attaques et autres. Une fois ces problèmes trouvés, l'IA et le ML peuvent réagir avec des techniques d'atténuation et des alertes automatisées.

L'IA et le ML peuvent faire gagner du temps aux développeurs et aux professionnels des opérations en apprenant comment ils travaillent le mieux, en faisant des suggestions au sein des flux de travail et en provisionnant automatiquement les configurations d'infrastructure préférées.[7]

DevOps et Le "Cloud-Native Mouvement"

Le transfert du développement logiciel vers le cloud présente tellement d'avantages que de plus en plus d'entreprises adoptent Cloud-Native Computing. La création, le test et le déploiement d'applications à partir du cloud permettent de réaliser des économies car les entreprises peuvent faire évoluer les ressources plus facilement, prendre en charge une expédition plus rapide des logiciels, s'aligner sur les objectifs commerciaux et libérer les équipes DevOps pour qu'elles innovent plutôt que de maintenir l'infrastructure.

Le développement d'applications natives dans le nuage permet aux développeurs et aux équipes d'exploitation de travailler de manière plus collaborative, ce qui se traduit par de meilleurs logiciels livrés plus rapidement. Des plateformes comme AWS, Azure ou GCP propose nativement des approches pour intégrer le DevOps tout au long du flu de travail lors du developement.[8]

2.6 L'émergence de l'architecture en microservice

2.6.1 Architecture Monolithique

Les logiciels monolithiques sont conçus pour être autonomes, c'est-à-dire que les composants ou fonctions du programme sont étroitement couplés plutôt que faiblement couplés, comme dans les logiciels modulaires. Dans une architecture monolithique, chaque composant et ses composants associés doivent tous être présents pour que le code soit exécuté ou compilé et que le logiciel fonctionne.[9]

Les applications monolithiques sont à un seul niveau, ce qui signifie que plusieurs composants sont combinés en une seule grande application. Par conséquent, elles ont tendance à avoir des bases de code importantes, ce qui peut être difficile à gérer au fil du temps.

En outre, si un composant du programme doit être mis à jour, d'autres éléments peuvent également nécessiter une réécriture, et l'ensemble de l'application doit être recompilé et testé. Ce processus peut prendre beaucoup de temps et limiter l'agilité et la rapidité des équipes de développement logiciel. Malgré ces problèmes, l'approche est toujours utilisée car elle offre certains avantages. En outre, de nombreuses applications anciennes ont été développées en tant que logiciels monolithiques, de sorte que l'approche ne peut être complètement ignorée lorsque ces applications sont encore utilisées et nécessitent des mises à jour.[9]

Les architectures monolithiques présentent des avantages, c'est pourquoi de nombreuses applications sont encore créées à l'aide de ce paradigme de développement. Par exemple, les programmes monolithiques peuvent avoir un meilleur débit que les applications modulaires. Ils peuvent également être plus faciles à tester et à déboguer car, avec moins d'éléments, il y a moins de variables et de scénarios de test qui entrent en jeu.

Au début du cycle de vie du développement logiciel, il est généralement plus facile d'opter pour une architecture monolithique, car le développement peut être plus simple pendant les premières étapes. Une base de code unique simplifie également la journalisation, la gestion de la configuration, le contrôle des performances de l'application et d'autres problèmes de développement. Le déploiement peut également être plus facile en copiant l'application packagée sur un serveur. Enfin, plusieurs copies de l'application peuvent être placées derrière un équilibreur de charge pour la faire évoluer horizontalement.[9]

Cela dit, l'approche monolithique est généralement préférable pour les applications simples et légères. Pour des applications plus complexes, avec des modifications de code fréquentes ou des exigences d'évolutivité en constante évolution, cette approche n'est pas adaptée.

2.6.2 Architecture Microservice

L'architecture microservices (souvent abrégée en microservices) désigne un style architectural pour le développement d'applications. Les microservices permettent de séparer une grande application en plusieurs parties indépendantes plus petites, chaque partie

ayant son propre domaine de responsabilité. Pour répondre à une demande unique d'un utilisateur, une application basée sur les microservices peut faire appel à de nombreux microservices internes pour composer sa réponse.[10]

Les conteneurs sont un exemple d'architecture microservices bien adapté, car ils nous permettent de nous concentrer sur le développement des services sans qu'on se soucie des dépendances. Les applications cloud-natives modernes sont généralement construites en tant que microservices à l'aide de conteneurs.

Les microservices, peuvent être modifiés indépendamment sans affecter les autres parties du programme et sans créer de changements imprévus dans d'autres éléments.[10]

Les programmes modulaires s'adaptent également mieux aux processus de développement itératifs et aux pratiques Agile que les programmes monolithiques. Ils sont également plus évolutifs et peuvent être testés individuellement en raison du couplage lâche entre les différents composants. Les modules communiquent également entre eux, disposent de leurs propres bases de données et augmentent la vitesse de démarrage des applications.

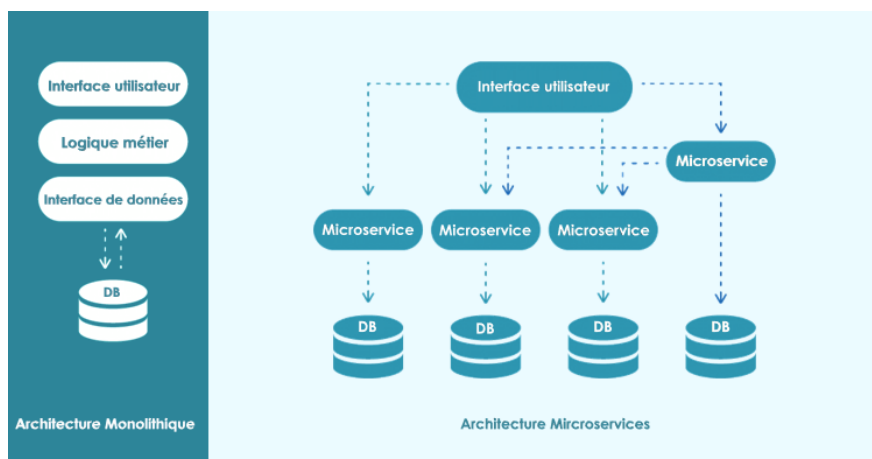


FIGURE 2.5 – Difference entre un architecture Monolithique et en Microservice

2.7 Docker

2.7.1 Definition

Docker est une plateforme comportant différents composants qui éliminent certaines des tâches ardues que les utilisateurs auraient à supporter autrement lorsqu'ils ont affaire à une virtualisation de bas niveau du système d'exploitation. En termes officiels, Docker s'efforce de conteneuriser un logiciel dans un fichier complet. Un logiciel dans un système de fichiers complet qui contient tout ce dont il a besoin pour fonctionner : code, runtime, outils système, bibliothèques système, afin que son environnement est similaire quel que soit l'endroit où il est exécuté.[11]

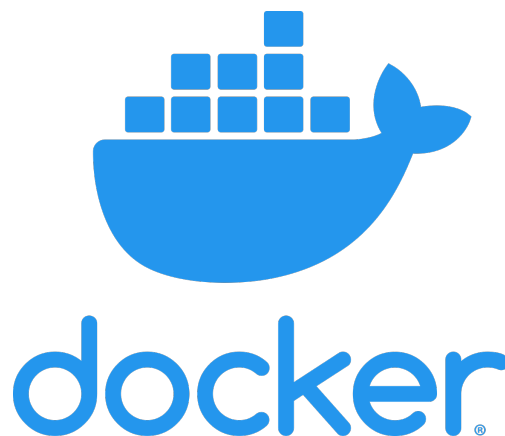


FIGURE 2.6 – Logo de Docker

2.7.2 Docker Containers et machines virtuelles

Les conteneurs sont légers (en termes de ressources) par défaut. Un seul Kernel peut exécuter plusieurs conteneurs en même temps et, en général, plus de conteneurs que de VMs. Ce site encourage l'utilisateur à séparer chaque composant de l'application dans différents conteneurs différents et à les relier entre eux à l'aide d'outils fournis par la plate-forme Docker (tels que Docker Compose) au lieu d'essayer de placer chaque composant dans une seule VM afin que des ressources matérielles pourraient être conomisées.[12]

2.7.3 les avantages de Docker

Du point de vue architectural, les VMs exigent que chaque composant applicatif ait son propre OS invité installé alors que Docker partage certaines des ressources de l'OS hôte et offre donc les avantages suivants : [13]

Petit : le conteneur nécessite les données de l'application elle-même et l'OS, mais le noyau est partagé avec l'OS hôte .

Rapide : comme il n'est pas nécessaire de charger le noyau, le processus de démarrage du conteneur prend normalement de quelques millisecondes à quelques secondes. Dans la

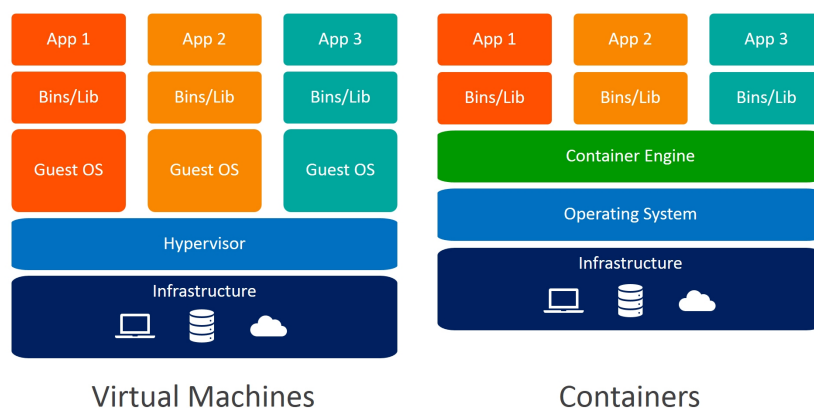


FIGURE 2.7 – Les Conteneurs Docker en Comparaison avec les machine virtuelle

pratique, la virtualisation n'est qu'un processus supplémentaire en cours d'exécution qui est isolé du reste, et la création de nouveaux processus dans le système d'exploitation hôte déjà en cours d'exécution est peu coûteuse.

Flexibilité de l'utilisation des ressources : alors que les machines virtuelles nécessitent des limites strictes concernant l'utilisation du matériel, Docker permet également de définir des limites souples, ce qui permet aux conteneurs en cours d'exécution d'utiliser les ressources sous-utilisées qui ont été allouées aux autres conteneurs.

Réutilisabilité accrue : il est simple de créer des conteneurs basés sur d'autres conteneurs, de sorte que les étapes ne sont pas répétées. De plus, différents conteneurs partageant la même image de base peuvent fonctionner simultanément sans qu'il soit nécessaire de répliquer l'image sur le disque.

2.7.4 Les inconvénients de Docker

Docker présente également des inconvénients comme [13] :

Isolation plus faible : il existe une différence significative par rapport à l'isolation que peut offrir un hyperviseur. Les conteneurs logiciels sont des processus sandboxés utilisant certaines des fonctionnalités du noyau Linux (namespaces, cgroups, AppArmor, etc.). En raison de cette nature, cela ne peut pas être radicalement modifié.

Une stabilité réduite : le rythme rapide des changements de la plate-forme est un effet secondaire courant des nouvelles technologies, et Docker ne fait pas exception. Les utilisateurs qui recherchent un logiciel mature et stable pour faire fonctionner une infrastructure critique et qui offre des fonctionnalités similaires devraient utiliser la virtualisation traditionnelle.

Pénalités de performance : alors que les conteneurs Docker atteignent des performances proches de celles d'une machine nue en général, l'utilisation de Docker NAT ajoute une surcharge CPU et la vitesse d'E/S du stockage du conteneur est plus lente que celle de la

machine nue ou des volumes de données attachés au conteneur.

2.7.5 Architecture de Docker

Docker utilise une architecture client-serveur. Le client Docker communique avec le démon Docker, qui se charge de construire, d'exécuter et de distribuer nos conteneurs Docker. Le client et le démon Docker peuvent fonctionner sur le même système, ou on peut connecter un client Docker à un démon Docker distant. Le client et le démon Docker communiquent à l'aide d'une API REST, via des sockets UNIX ou une interface réseau. Un autre client Docker est Docker Compose, qui nous permet de travailler avec des applications composées d'un ensemble de conteneurs.[14]

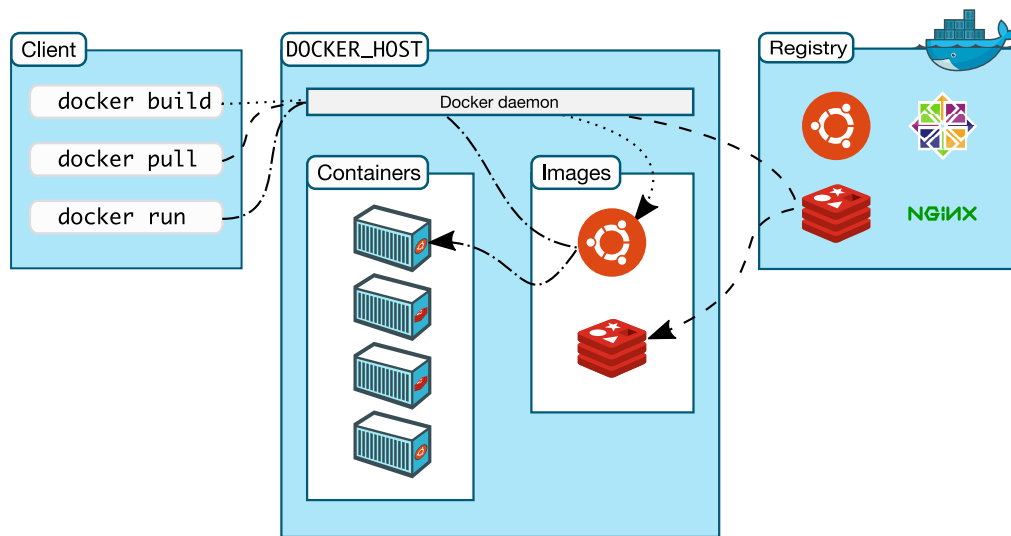


FIGURE 2.8 – Architecture de Docker

Le Docker daemon

The Docker daemon (dockerd) écoute les demandes de l'API Docker et gère les objets Docker tels que les images, les conteneurs, les réseaux et les volumes. Un démon peut également communiquer avec d'autres démons pour gérer les services Docker.

Le Docker client

Le Docker client (docker) est le principal moyen par lequel de nombreux utilisateurs de Docker interagissent avec Docker. Lorsqu'on utilise des commandes telles que docker run, le client envoie ces commandes à dockerd, qui les exécute. La commande docker utilise l'API de Docker. Le client Docker peut communiquer avec plus d'un démon.

Docker registries

Un registre Docker stocke les images Docker. Docker Hub est un registre public que tout le monde peut utiliser, et Docker est configuré pour rechercher les images sur Docker Hub par défaut. on peut même exécuter notre propre registre privé.

Lorsque on utilise les commandes docker pull ou docker run, les images requises sont extraites de notre registre configuré. Lorsque on utilise la commande docker push, notre image est poussée vers notre registre configuré.

Docker objects

Lorsque on utilise Docker, on crée et utilise des images, des conteneurs, des réseaux, des volumes, des plugins et d'autres objets. Cette section est un bref aperçu de certains de ces objets.

Images

Une image est un modèle en lecture seule contenant des instructions pour créer un conteneur Docker. Souvent, une image est basée sur une autre image, avec quelques personnalisations supplémentaires. Par exemple, on peut construire une image basée sur l'image ubuntu, mais qui installe le serveur web Apache et notre application, ainsi que les détails de configuration nécessaires au fonctionnement de notre application.

on peut créer nos propres images ou utiliser uniquement celles créées par d'autres et publiées dans un registre. Pour créer notre propre image, on crée un Dockerfile avec une syntaxe simple pour définir les étapes nécessaires à la création de l'image et à son exécution. Chaque instruction d'un Dockerfile crée une couche dans l'image. Lorsque on modifie le Dockerfile et reconstruisez l'image, seules les couches qui ont été modifiées sont reconstruites. C'est en partie ce qui rend les images si légères, petites et rapides, par rapport aux autres technologies de virtualisation.

Containers

Un conteneur est une instance exécutable d'une image. on peut créer, démarrer, arrêter, déplacer ou supprimer un conteneur à l'aide de l'API ou du CLI de Docker. on peut connecter un conteneur à un ou plusieurs réseaux, lui attacher du stockage, ou même créer une nouvelle image basée sur son état actuel.

Par défaut, un conteneur est relativement bien isolé des autres conteneurs et de sa machine hôte. on peut contrôler le degré d'isolement du réseau, du stockage ou des autres sous-systèmes sous-jacents d'un conteneur par rapport aux autres conteneurs ou à la machine hôte.

Un conteneur est défini par son image ainsi que par les options de configuration qu'on lui fournisse lorsqu'on le crée ou le démarre. Lorsqu'un conteneur est retiré, toutes les modifications apportées à son état qui ne sont pas stockées dans le stockage persistant disparaissent.

2.7.6 Le rôle de Docker dans le DevOps

Docker est un outils qui s'avère parfaitement adapté à l'écosystème DevOps. Les avantages que Docker offre à l'environnement DevOps en ont fait un outil irremplaçable dans la chaîne d'outils.

La raison pour laquelle Docker est si bon pour DevOps est que ses avantages et ses cas d'utilisation de conteneurisation des applications qui soutiennent le développement et la libération rapide. DevOps est principalement utilisé pour surmonter les problèmes de "Dev" et d'"Ops", et Docker semble résoudre la plupart d'entre eux, le principal étant qu'il

peut fonctionner sur n'importe quelle machine. Ainsi, il permet à toutes les équipes de collaborer et de travailler de manière efficace et efficiente.[14]

Docker permet de créer des environnements inévitables de développement, de production et de mise en scène, en offrant ainsi un contrôle transparent de l'ensemble des changements. En effet Tous les environnements se ressemblent davantage. Docker garantit que si une fonctionnalité fonctionne dans l'environnement de développement, elle fonctionnera également dans les environnements de production et de test.

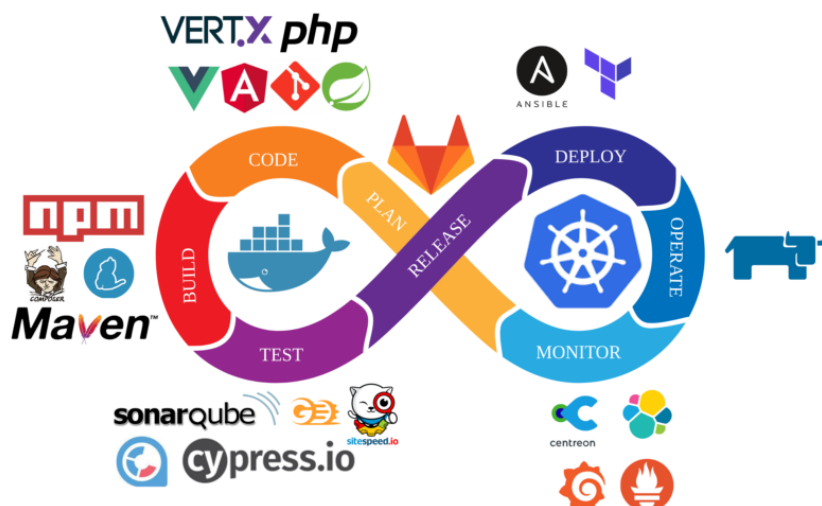


FIGURE 2.9 – Les principaux technologies de DevOps

L'un des principaux avantages de l'utilisation de Docker dans le cadre de DevOps est que les développeurs, les testeurs et les administrateurs système l'utilisent tous. Par exemple, les développeurs peuvent utiliser Dockerfiles pour créer des images Docker sur des ordinateurs locaux et les exécuter. Les administrateurs système peuvent utiliser les mêmes images Docker pour effectuer des mises à jour et mettre en scène les environnements de production.[14]

Ainsi, on peut créer des environnements stables pour le développement, la production et la mise en scène. Cette approche présente également plusieurs avantages, comme nous le verrons plus loin.

Docker et le problème d'orchestration

Suite au succès de Docker, on a remarqué que pour contrôler de la manière dont les conteneurs agissent entre eux on avait à écrire des scripts bash ou autres, ce qui était très difficile à maintenir ainsi que Debugger, [15] les conteneurs garantissent effectivement que ces applications s'exécutent de la même manière partout, ce qui nous permet de profiter rapidement et facilement de tous ces environnements. En outre, lorsque nous faisons évoluer nos applications, nous avons besoin d'outils pour automatiser la maintenance de ces applications, permettre le remplacement automatique des conteneurs défectueux et gérer le déploiement des mises à jour et des reconfigurations de ces conteneurs pendant leur cycle de vie, ceci marquera l'inspiration pour développer l'outil dont on va parler au chapitre suivant qui est Kubernetes.

2.8 kubernetes

Kubernetes est une plate-forme open source évolutive et portable pour la gestion des charges de travail et des services conteneurisés. Il facilite l'écriture de configuration déclarative et l'automatisation. Il s'agit d'un vaste écosystème en pleine expansion. Les services, le support et les outils Kubernetes sont largement disponibles.

Google a rendu open-source le projet Kubernetes en 2014.[16] Kubernetes est désormais le deuxième plus grand projet open source au monde, juste derrière Linux.



FIGURE 2.10 – Logo de Kubernetes

2.8.1 Kubernetes comme étant un orchestrateur

Un orchestrateur est un système qui déploie et gère des applications. Il peut déployer les applications et répondre dynamiquement aux changements. Par exemple, Kubernetes peut [16] :

- Déployer Une application.
- la faire évoluer dynamiquement vers le haut ou le bas en fonction de la demande.
- L'auto-réparer en cas de panne.
- Effectuer des mises à jour et des retours en arrière sans temps d'arrêt.

Kubernetes nous fournit un outils qui prend en charge la mise à l'échelle et le basculement d'une application, fournit des modèles de déploiement, etc.

Kubernetes fournit aussi[17] :

La découverte de services et l'équilibrage de charge (load Balancer) : Kubernetes peut exposer un conteneur en utilisant le nom DNS ou en utilisant sa propre adresse IP. Si le trafic vers un conteneur est élevé, Kubernetes est capable d'équilibrer la charge et de distribuer le trafic réseau afin que le déploiement soit stable.

Orchestration du stockage : Kubernetes permet de monter automatiquement un système de stockage de notre choix, tel qu'un stockage local, des fournisseurs de cloud public, etc. Déploiements et retours en arrière automatisés : pouvez décrire l'état souhaité pour nos conteneurs déployés à l'aide de Kubernetes, et celui-ci peut faire évoluer l'état réel vers l'état souhaité à un rythme contrôlé. Par exemple, on peut automatiser Kubernetes pour créer de nouveaux conteneurs pour notre déploiement, supprimer les conteneurs existants et adopter toutes leurs ressources vers le nouveau conteneur.

Emballage automatique des conteneurs : on fournit à Kubernetes un cluster de nœuds qu'il peut utiliser pour exécuter des tâches conteneurisées. on indique à Kubernetes la quantité de CPU et de mémoire (RAM) dont chaque conteneur a besoin. Kubernetes peut placer les conteneurs sur nos nœuds afin d'utiliser au mieux nos ressources.

Auto-réparation : Kubernetes redémarre les conteneurs qui échouent, remplace les conteneurs, tue les conteneurs qui ne répondent pas à notre contrôle de santé défini par l'utilisateur et ne les annonce pas aux clients avant qu'ils ne soient prêts à servir.

Gestion des secrets et des configurations : Kubernetes permet de stocker et de gérer des informations sensibles, telles que des mots de passe, des jetons OAuth et des clés SSH. on peut déployer et mettre à jour les secrets et la configuration des applications sans reconstruire nos images de conteneurs, et sans exposer les secrets dans la configuration de notre pile.

2.9 kubernetes et Docker

Kubernetes et Docker sont des technologies complémentaires.

Docker dispose d'outils permettant de construire et de conditionner des applications sous forme d'images de conteneurs. Il peut également exécuter des conteneurs. Kubernetes ne peut faire ni l'un ni l'autre. En revanche, Kubernetes opère à un niveau supérieur en fournissant des services d'orchestration tels que l'auto-réparation, la mise à l'échelle et les mises à jour.

Il est courant d'utiliser Docker pour les tâches de construction, telles que le conditionnement des applications en conteneurs, puis d'utiliser une combinaison de Kubernetes et de Docker pour les exécuter. Dans ce modèle, Kubernetes exécute les tâches d'orchestration de haut niveau telles que l'auto-réparation, la mise à l'échelle et les mises à jour, mais il a besoin d'un outil comme Docker pour exécuter les tâches de bas niveau telles que le démarrage et l'arrêt des conteneurs.

Supposons que on a un cluster Kubernetes avec 10 nœuds pour exécuter nos applications de production. Dans les coulisses, chaque nœud du cluster exécute Docker en tant que moteur de conteneur. Cela signifie que Docker est la technologie de bas niveau qui démarre et arrête les applications conteneurisées. Kubernetes est la technologie de plus haut niveau qui s'occupe de la l'image globale, comme le choix des nœuds sur lesquels exécuter les conteneurs, la décision de monter ou de descendre en charge et l'exécution

des mises à jour. l'exécution des mises à jour.

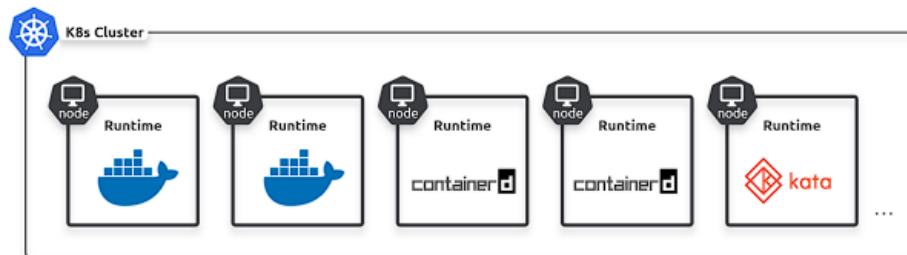


FIGURE 2.11 – Kubernetes Cluster

Docker n'est pas le seul runtime de conteneur pris en charge par Kubernetes. En fait, Kubernetes dispose de quelques fonctionnalités qui permettent d'abstraire le runtime de conteneur et de le rendre interchangeable.

2.9.1 kubernetes architecture

Les fichiers manifestes sont écrits en YAML simple et indiquent à Kubernetes à quoi devrait ressembler une application. C'est ce qu'on appelle l'état souhaité. Il comprend des éléments tels que l'image à utiliser, le nombre de répliques à exécuter, les ports réseau à écouter et la manière d'effectuer les mises à jour.[17]

Une fois que on a créé le manifeste, on le poste sur le serveur d'API. La façon la plus simple de le faire est d'utiliser l'utilitaire de ligne de commande kubectl. Il envoie le manifeste au plan de contrôle sous forme de HTTP POST.

Une fois la requête authentifiée et autorisée, Kubernetes inspecte le manifeste, identifie le contrôleur auquel l'envoyer (par exemple, le contrôleur Deployments) et enregistre la configuration dans le cluster store en tant que partie de l'état global souhaité. Une fois cette étape franchie, toutes les tâches requises sont programmées sur les nœuds du cluster, où le kubelet coordonne le travail difficile consistant à extraire les images, à démarrer les conteneurs, à se connecter aux réseaux et à lancer les processus d'application.

2.9.2 Pods

Les pods sont les objets déployables les plus petits et les plus basiques de Kubernetes. Un pod représente une instance unique d'un processus en cours d'exécution dans notre cluster.

Les pods contiennent un ou plusieurs conteneurs, tels que des conteneurs Docker. Lorsqu'un Pod exécute plusieurs conteneurs, ces derniers sont gérés comme une seule entité et partagent les ressources du Pod. En général, l'exécution de plusieurs conteneurs dans

un seul Pod est un cas d'utilisation avancé.

Les pods contiennent également des ressources de réseau et de stockage partagées pour leurs conteneurs :

Réseau : Des adresses IP uniques sont automatiquement attribuées aux Pods. Les conteneurs d'un Pod partagent le même espace de noms de réseau, y compris l'adresse IP et les ports de réseau. Les conteneurs d'un Pod communiquent entre eux à l'intérieur du Pod sur localhost.

Stockage : Les Pods peuvent spécifier un ensemble de volumes de stockage partagé qui peuvent être partagés entre les conteneurs. on peut considérer un pod comme un "hôte logique" autonome et isolé qui contient les besoins systémiques de l'application qu'il sert.

FIGURE 2.12 – Page d'Accueil

Au plus haut niveau, un pod est un environnement délimité pour exécuter des conteneurs. Les pods eux-mêmes n'exécutent pas réellement applications. les applications sont toujours exécutées dans des conteneurs, le pod n'est qu'une unite pour exécuter un ou plusieurs conteneurs. Pour rester à un niveau élevé, les pods délimitent une zone du système d'exploitation hôte, construisent un réseau, créent un certain nombre d'espaces de noms du noyau et exécutent un ou plusieurs conteneurs[17].

Si on exécute plusieurs conteneurs dans un Pod, ils partagent tous le même environnement Pod. Cela inclut le network stack, les volumes, l'espace de noms IPC, la mémoire partagée, etc. À titre d'exemple, cela signifie que tous les conteneurs du le même Pod partagent la même adresse IP (l'IP du Pod).

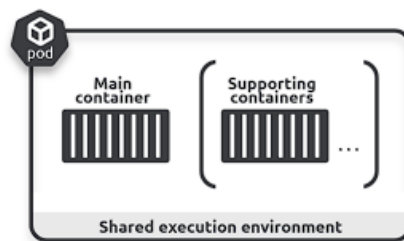


FIGURE 2.13 – Pod Kubernetes

2.9.3 Déploiements

La plupart du temps, nous déployons les pods indirectement via des contrôleurs de niveau supérieur. Les Deployments, DaemonSets et StatefulSets sont des exemples de contrôleurs de niveau supérieur. Par exemple, un Deployment est un objet Kubernetes de niveau supérieur qui entoure un Pod et ajoute des fonctionnalités telles que l'auto-réparation, la mise à l'échelle, les déploiements sans temps d'arrêt et les retours en arrière par version. En coulisses, les Deployments, DaemonSets et StatefulSets sont implémentés

comme des contrôleurs qui fonctionnent comme des boucles de surveillance observant constamment le cluster pour s'assurer que l'état observé correspond à l'état souhaité.[2]

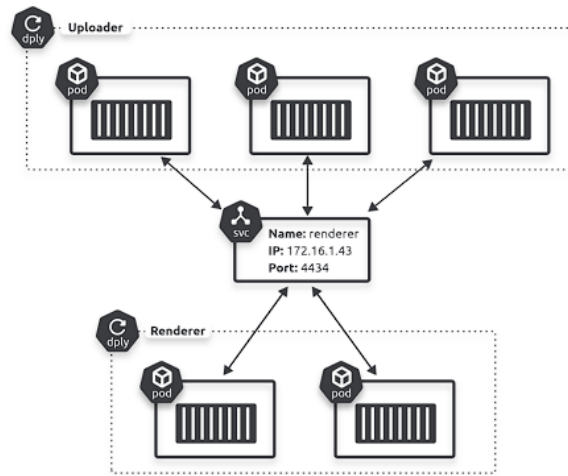


FIGURE 2.14 – Exemple de Deployment

ConfigMap

Les ConfigMaps lient des artefacts de configuration non sensibles, tels que des fichiers de configuration, des arguments de ligne de commande et des variables d'environnement, à nos conteneurs Pod et composants système au moment de l'exécution. Une ConfigMap sépare nos configurations de notre Pod et de nos composants, ce qui contribue à la portabilité de nos charges de travail. Cela rend leurs configurations plus faciles à modifier et à gérer, et évite de coder en dur les données de configuration dans les spécifications des Pods.

3

Analyse et conception

Il est à rappeler que l'objectif du projet consiste à créer un éditeur graphique pour l'outil Kubernetes. Ce chapitre permet de faire une analyse théorique. En effet, cette conception est cruciale afin de comprendre la totalité des principes et les coder en se basant sur le cahier des charges fournit.

3.1 Analyse

3.1.1 Cahier des charges

Le sujet s'intéresse à la création d'un éditeur graphique permettant la détection des erreurs syntaxiques ainsi que la saisie semi-automatique du code au niveau de Kubernetes.

Charges fonctionnelle :

- L'éditeur doit être capable de détecter les erreurs syntaxiques au sein du code.
- L'éditeur doit être capable de faire une saisie semi-automatique des mots clés du Kubernetes.
- Une interface graphique simple et conviviale.

3.1.2 Analyse de l'existant : kubectl-neat

3.1.2.1 Présentation de la solution

Kubectl-neat a pour objectif supprimer le désordre des manifestes Kubernetes pour les rendre plus lisibles. Voici le résultat d'un Pod simple après l'usage du Kubectl-neat. Les lignes marquées en rouge sont considérées comme redondantes et seront retirées de la sortie par kubectl-neat en utilisant la commande :

```
KUBECTL GET POD -O YAML
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2019-04-24T19:55:27Z"
  labels:
    name: myapp
  name: myapp
  namespace: default
  resourceVersion: "274103"
  selfLink: /api/v1/namespaces/default/pods/myapp
  uid: e8330f3c-66ca-11e9-b6fa-0800271788ca
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: myapp
    ports:
    - containerPort: 1234
      protocol: TCP
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-nmshj
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: minikube
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
```

FIGURE 3.1 – Résultat du Kubectl-neat

3.1.2.2 Problématique résolue

Lors la création d’une ressource Kubernetes, disons un Pod, Kubernetes ajoute toute une série d’informations système internes au yaml ou json initialement écrit. Cela comprend :

- Metadata comme la création du timestamp, ou certains IDs internes.
- Remplir les attributs manquants avec les valeurs par défaut.
- Des attributs supplémentaires du système créés par les contrôleurs d’admission, tels que le jeton de compte de service.
- Renseignements sur l’état.

Si Nous essayons de visualiser les ressources créées, elles ne ressembleront plus à ce que nous avons écrit à l’origine, et seront inutilement verbeuses. Kubectl-neat nettoie cette information redondante.

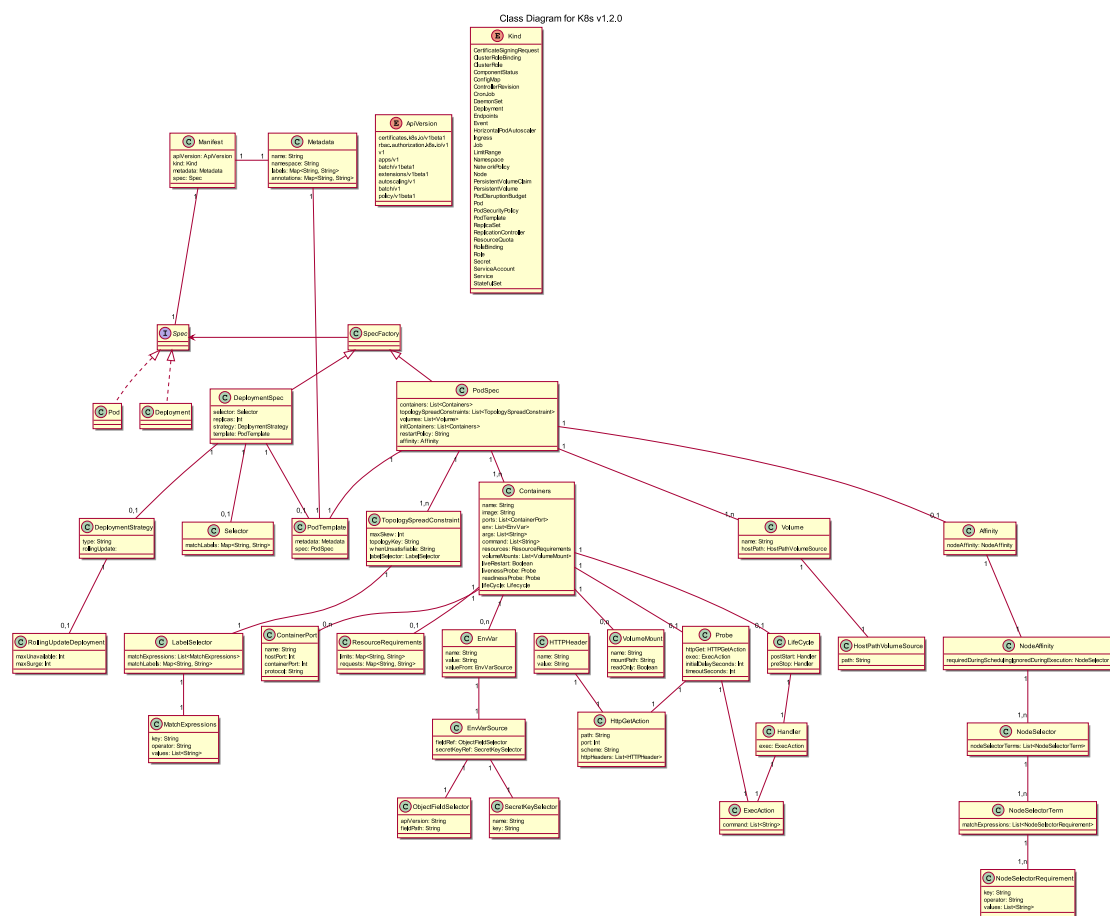
3.1.2.3 Point en Commun

Kubectl-neat et notre projet ont un point commun qui est la detection des differents composants du code (classes) dans un code kubernetes, par contre Kubectl-neat modifie le code en supprimant les informations redondantes , notre projet se base sur la saisie semi-automatique et la detection des erreurs au niveau de la sémantique ainsi que la syntaxique.

3.2 Conception

3.2.1 Diagramme de classe

Le diagramme de classe fait partie de la partie statique d’UML car il fait abstraction des aspects temporels et dynamiques. Une classe décrit les responsabilités, le comportement et le type d’un ensemble d’objets. Les éléments de cet ensemble sont les instances de la classe.



4

Réalisation de l'interface graphique

Ce chapitre est consacré à la présentation de l'environnement logiciel utilisé pour le développement de la solution proposée, avec une explication éventuelle aux choix techniques relatifs aux langages de programmation et des outils utilisés.

4.1 Outils

4.1.1 Eclipse Modeling Framework (EMF)

Eclipse Modeling Framework (EMF) est un framework de modélisation basé sur Eclipse et une fonction de génération de code pour la création d'outils et d'autres applications basés sur un modèle de données structurées.

À partir d'une spécification de modèle décrite dans XML Metadata Interchange (XMI), EMF fournit des outils et une prise en charge du runtime pour produire un ensemble de classes Java pour le modèle, un ensemble de classes d'adaptateur qui permettent l'affichage et la modification basée sur des commandes du modèle, et un éditeur de base. Les modèles peuvent être spécifiés à l'aide de documents Java, UML, XML annotés ou d'outils de modélisation, puis importés dans EMF. Plus important encore, EMF fournit la base de l'interopérabilité avec d'autres outils et applications basés sur EMF.[18]

EMF model peut être généré par deux méthodes : depuis une Rational Rose model ou bien par des sets de Java interfaces et des classes . Les étapes de la génération est comme suit :

- Importer le Model depuis Rose ou bien Définir le Model en utilisant l'annotation Java.
- Générer le EMF Model Code.
- Générer un Editeur depuis le Model.
- Lancer l'Editeur généré.

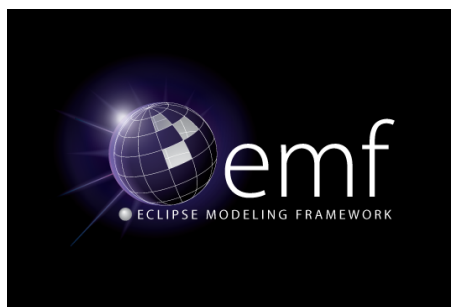


FIGURE 4.1 – Logo EMF

4.1.1.1 Ecore

Ecore est le (méta-)modèle de base au cœur des CEM. Il permet d'exprimer d'autres modèles en tirant parti de ses constructions. Ecore est aussi son propre métamodèle (c'est-à-dire que Ecore est défini en termes de lui-même).

Selon Ed Merks, responsable du projet EMF, « Ecore est l'implémentation de référence de facto de l'EMOF » (Essential Meta-Object Facility) d'OMG. Toujours selon Merks,

EMOF a en fait été défini par OMG comme une version simplifiée du « C'MOF » plus complet en s'appuyant sur l'expérience de la simplification réussie de la mise en œuvre originale d'Ecore.

L'utilisation d'Ecore comme méta-modèle fondamental permet à un modélisateur de tirer parti de l'ensemble de l'écosystème et des outils EMF - dans la mesure où il est alors raisonnablement facile de mapper les modèles au niveau de l'application à Ecore. Cela ne veut pas dire qu'il est recommandé pour les applications de tirer directement parti d'Ecore comme métamodèle ; ils pourraient plutôt envisager de définir leurs propres métamodèles basés sur Ecore.

4.1.1.2 .genmodel

La génération de code est spécifiée dans le fichier "esdl.genmodel". Dans Eclipse, Nous pouvons ouvrir ce fichier dans son propre éditeur. Dans les propriétés, il existe de nombreuses options que nous pouvons configurer pour modifier le comportement de la génération de code.

En cliquant le bouton droit de la souris sur le package esdl, nous pouvons sélectionner Générer... pour générer un ou plusieurs des artefacts, ou appuyez sur Ctrl + Maj + G pour afficher la boîte de dialogue Générateur.

Le modèle "esdl.gen actuel" est configuré pour produire également un schéma XML, qui peut être utilisé par d'autres logiciels non basés sur Java.

4.1.2 Eclipse Xtext

Eclipse Xtext est un framework qui facilite le développement de DSL. Il facilite l'évolution agile et itérative ; et la sortie n'est pas seulement l'infrastructure pour analyser les fichiers de votre langage nouvellement créé, mais un IDE sophistiqué avec mise en surbrillance du code, complétion de code et vérification des erreurs pour le nouveau DSL. Xtext est un cadre mature utilisé dans la recherche et l'industrie par des entreprises telles qu'Oracle, Google, BMW et bien d'autres.[19]



FIGURE 4.2 – Logo du logiciel Eclipse

4.1.3 LaTeX

Les documents professionnels nécessitent un logiciel de traitement, Latex est l'outil parfait pour cette tâche. Il était donc inévitable de travailler avec ce dernier.



FIGURE 4.3 – Logo du LaTeX

4.1.4 Visual Studio

Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation du code et Git intégré. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions qui ajoutent des fonctionnalités supplémentaires.



FIGURE 4.4 – Logo de Vs code

4.2 Développement

Dans cette partie , nous allons suivre les étapes mentionnées avant afin de réaliser notre éditeur graphique.

4.2.1 Ecore

Après la conception d'un diagramme de classes, on procède à la génération d'un modèle encore tout en se basant sur le diagramme de classes. L'Emf Model Code a deux présentations une sous-forme de code (xml) et une autre sous forme graphique. Afin de faciliter la création nous avons suivi l'approche graphique, au-dessous l'écore créée.

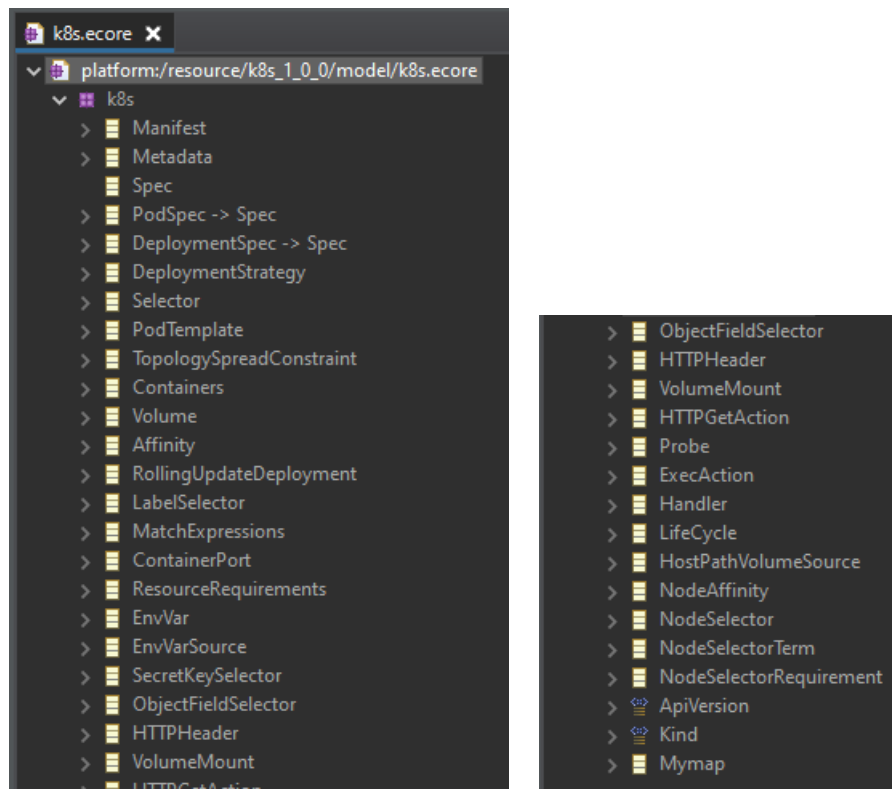


FIGURE 4.5 – Ecore

4.2.2 .genmodel

La création du .genmodel se base principalement sur l'Ecore.ci-dessous les étapes suivies à fin de créer le .genmodel

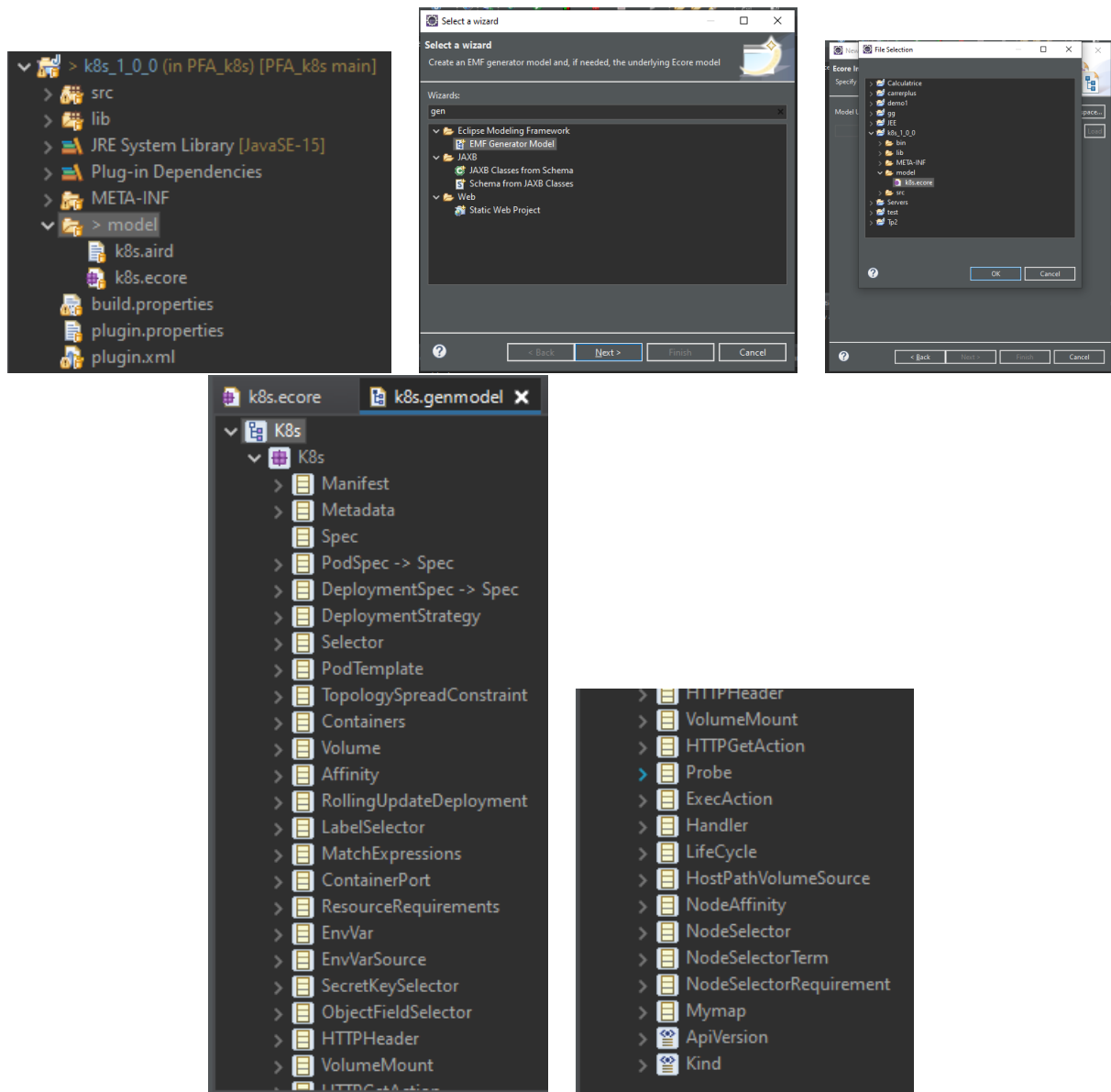


FIGURE 4.6 – Le genmodel

4.2.3 La grammaire

Après la génération du `.genmodel`, nous sommes prêts à générer notre grammaire, cette dernière doit être LL(2). une grammaire LL est une grammaire sans contexte qui peut être analysé par un analyseur LL, qui analyse l'entrée de gauche à droite, et construit une dérivation Leftmost de la phrase. Les analyseurs LL sont des analyseurs basés sur des tables, similaires aux analyseurs LR. Les grammaires LL peuvent également être caractérisé comme précisément celles qui peuvent être analysées par un analyseur prédictif – un analyseur de descente récursive sans retour en arrière. L'outil Xtext vient pour faciliter la génération de la grammaire.

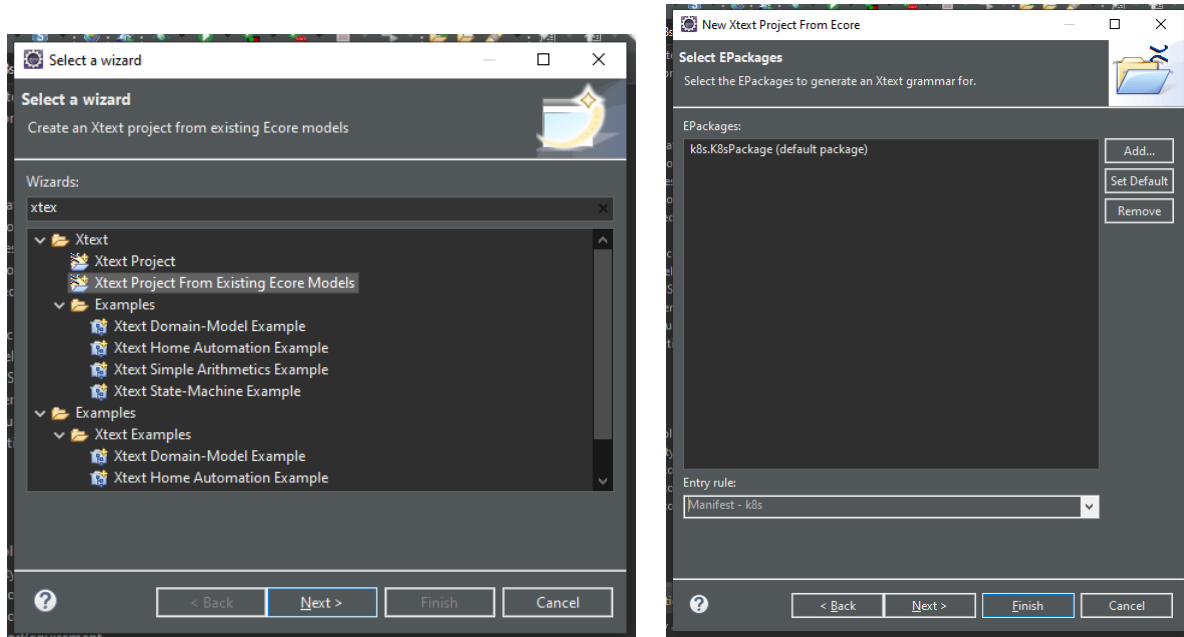


FIGURE 4.7 – Etapes de la génération de la grammaire par Xtext

La grammaire générée après une adaptation est comme suit :

```

1 // automatically generated by Xtext
2 grammar org.xtext.example.k8s.K8s with org.eclipse.xtext.common.Terminals
3
4 import "http://k8s.com"
5 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
6
7
8 Manifest returns Manifest:
9
10     'apiVersion:' apiVersion=ApiVersion
11     'kind:' kind=Kind
12     ('metadata:' metadata=Metadata)?
13     ('spec:' spec=Spec)?
14
15 ;
16
17 Spec returns Spec:
18     PodSpec | DeploymentSpec;
19
20
21 enum ApiVersion returns ApiVersion:
22     v1 = 'v1' | certificates_k8s_io_v1beta1 = '
23         certificates_k8s_io_v1beta1' | rbac_authorization_k8s_io_v1 = '
24         rbac_authorization_k8s_io_v1' | apps_v1 = 'apps_v1' | batch_v1beta1 = '
25         batch_v1beta1' | extensions_v1beta1 = 'extensions_v1beta1' |
26         autoscaling_v1 = 'autoscaling_v1' | batch_v1 = 'batch_v1' |
27         policy_v1beta1 = 'policy_v1beta1';
28
29 enum Kind returns Kind:
30     CertificateSigningRequest = 'CertificateSigningRequest' |
31     ClusterRoleBinding = 'ClusterRoleBinding' | ClusterRole = 'ClusterRole'
32     | ComponentStatus = 'ComponentStatus' | ConfigMap = 'ConfigMap' |
33     ControllerRevision = 'ControllerRevision' | CronJob = 'CronJob' |
34     DaemonSet = 'DaemonSet' | Deployment = 'Deployment' | Endpoints = '

```

```

Endpoints' | Event = 'Event' | HorizontalPodAutoscaler = '
HorizontalPodAutoscaler' | Ingress = 'Ingress' | Job = 'Job' |
LimitRange = 'LimitRange' | Namespace = 'Namespace' | NetworkPolicy = '
NetworkPolicy' | Node = 'Node' | PersistentVolumeClaim = '
PersistentVolumeClaim' | PersistentVolume = 'PersistentVolume' |
PodDisruptionBudget = 'PodDisruptionBudget' | Pod = 'Pod' |
PodSecurityPolicy = 'PodSecurityPolicy' | PodTemplate = 'PodTemplate' |
ReplicaSet = 'ReplicaSet' | ReplicationController = '
ReplicationController' | ResourceQuota = 'ResourceQuota' | RoleBinding =
'RoleBinding' | Role = 'Role' | Secret = 'Secret' | ServiceAccount = '
ServiceAccount' | Service = 'Service' | StatefulSet = 'StatefulSet';
26
27 Metadata returns Metadata:
28   {Metadata}
29
30   'name:' name=EString
31   ('namespace:' namespace=EString)?
32   ('labels:' '-' labels+=Mymap ( "-" labels+=Mymap)* )?
33   ('annotations:' '-' annotations+=Mymap ( "-" annotations+=Mymap)* )?
34   ;
35
36 EString returns ecore::EString:
37   STRING | ID;
38
39 Mymap returns Mymap:
40   {Mymap}
41
42   ( key=EString)?
43   ':'
44   ( value=EString)?
45   ;
46
47 PodSpec returns PodSpec:
48   {PodSpec}
49
50   ('restartPolicy:' restartPolicy=EString)?
51   ('affinity:' affinity=Affinity)?
52   ('containers:' '-' containers+=Containers ( "-" containers+=Containers)
53   * )?
54   ('volumes:' '-' volumes+=Volume ( "-" volumes+=Volume)* )?
55   ('topologySpreadConstraints:' '-' topologySpreadConstraints+=
56   TopologySpreadConstraint ( "-" topologySpreadConstraints+=
57   TopologySpreadConstraint)* )?
58   ('initContainers:' '-' initContainers+=Containers ( "-" initContainers
59   +=Containers)* )?
60   ;
61
62 DeploymentSpec returns DeploymentSpec:
63   {DeploymentSpec}
64
65   ('selector:' selector=Selector)?
66   ('strategy:' strategy=DeploymentStrategy)?
67   ('template:' template=PodTemplate)?
68   ;
69
70 Affinity returns Affinity:
71   {Affinity}
72
73

```

```

69     ('nodeAffinity:' nodeAffinity=NodeAffinity)?
70 ;
71
72 Containers returns Containers:
73     (liveRestart?='liveRestart')?
74
75     'name:' name=EString
76     'image:' image=EString
77     ('command:' '-' command+=EString ( "-" command+=EString)* )?
78     ('args:' '-' args+=EString ( "-" args+=EString)* )?
79     ('resources:' resources=ResourceRequirements)?
80     ('livenessProbe:' livenessProbe=Probe)?
81     ('readinessProbe:' readinessProbe=Probe)?
82     ('lifeCycle:' lifeCycle=LifeCycle)?
83     ('volumeMounts:' '-' volumeMounts+=VolumeMount ( "-" volumeMounts+=
VolumeMount)* )?
84     ('env:' '-' env+=EnvVar ( "-" env+=EnvVar)* )?
85     ('ports:' '-' ports+=ContainerPort ( "-" ports+=ContainerPort)* )?
86 ;
87
88 Volume returns Volume:
89
90     'name:' name=EString
91     'hostPath:' hostPath=HostPathVolumeSource
92 ;
93
94 TopologySpreadConstraint returns TopologySpreadConstraint:
95     {TopologySpreadConstraint}
96
97     ('topologyKey:' topologyKey=EString)?
98     ('whenUnsatisfiable:' whenUnsatisfiable=EString)?
99     ('maxSkew:' maxSkew=EInt)?
100     ('labelSelector:' labelSelector=LabelSelector)?
101 ;
102
103 NodeAffinity returns NodeAffinity:
104
105     'requiredDuringSchedulingIgnoredDuringExecution:' '-'
requiredDuringSchedulingIgnoredDuringExecution+=NodeSelector ( "-"
requiredDuringSchedulingIgnoredDuringExecution+=NodeSelector)*
106 ;
107
108 NodeSelector returns NodeSelector:
109
110     'nodeSelectorTerms:' '-' nodeSelectorTerms+=NodeSelectorTerm ( "-"
nodeSelectorTerms+=NodeSelectorTerm)*
111 ;
112
113 NodeSelectorTerm returns NodeSelectorTerm:
114
115     'matchExpressions:' '-' matchExpressions+=NodeSelectorRequirement ( "-"
matchExpressions+=NodeSelectorRequirement)*
116 ;
117
118 NodeSelectorRequirement returns NodeSelectorRequirement:
119
120     'values:' '-' values+=EString ( "-" values+=EString)*
121     'key:' key=EString

```

```

122     'operator:' operator=EString
123     ;
124
125 EBoolean returns ecore::EBoolean:
126     'true' | 'false';
127
128 ResourceRequirements returns ResourceRequirements:
129     {ResourceRequirements}
130
131     ('limits:' limits=Mymap)?
132     ('requests:' requests=Mymap)?
133     ;
134
135 Probe returns Probe:
136
137
138     ('initialDelaySeconds:' initialDelaySeconds=EInt)?
139     ('timeoutSeconds:' timeoutSeconds=EInt)?
140     'httpGet:' httpGet=HTTPGetAction
141     'exec:' exec=ExecAction
142     ;
143
144 LifeCycle returns LifeCycle:
145     {LifeCycle}
146
147     ('postStart:' postStart=Handler)?
148     ('preStop:' preStop=Handler)?
149     ;
150
151 VolumeMount returns VolumeMount:
152     (readOnly?='readOnly')?
153
154     'name:' name=EString
155     'mountPath:' mountPath=EString
156     ;
157
158 EnvVar returns EnvVar:
159     {EnvVar}
160
161     'name:' name=EString
162     ('value:' value=EInt)?
163     ('valueFrom:' valueFrom=EnvVarSource)?
164     ;
165
166 ContainerPort returns ContainerPort:
167
168     'name:' name=EString
169     'protocol:' protocol=EString
170     'hostPort:' hostPort=EInt
171     'containerPort:' containerPort=EInt
172     ;
173
174 HTTPGetAction returns HTTPGetAction:
175
176     'path:' path=EString
177     'port:' port=EInt
178     ('scheme:' scheme=EString)?
179     ('httpHeaders:' '-' httpHeaders+=EString ( "-" httpHeaders+=EString)*

```

```

179     )?
180 ;
181
182 ExecAction returns ExecAction:
183
184     'command:' '─' command+=EString ( "─" command+=EString)*
185 ;
186
187 EInt returns ecore::EInt:
188     '─'? INT;
189
190 Handler returns Handler:
191
192     'exec:' exec=ExecAction
193 ;
194
195 EnvVarSource returns EnvVarSource:
196
197     'fieldRef:' fieldRef=SecretKeySelector
198     ('secretKeyRef:' secretKeyRef=ObjectFieldSelector)?
199 ;
200
201 SecretKeySelector returns SecretKeySelector:
202
203     'name:' name=EString
204     'key:' key=EString
205 ;
206
207 ObjectFieldSelector returns ObjectFieldSelector:
208
209     'apiVersion:' apiVersion=ApiVersion
210     'fieldPath:' fieldPath=EString
211 ;
212
213 HostPathVolumeSource returns HostPathVolumeSource:
214
215     'path:' path=EString
216 ;
217
218 LabelSelector returns LabelSelector:
219     {LabelSelector}
220
221     ('matchExpressions:' '─' matchExpressions+=MatchExpressions ( "─"
222     matchExpressions+=MatchExpressions)* )?
223     ('matchLabels:' matchLabels=Mymap)?
224 ;
225 MatchExpressions returns MatchExpressions:
226
227     'key:' key=EString
228     ('values:' '─' values+=EString ( "─" values+=EString)* )?
229     ('operator:' operator=EString)?
230 ;
231
232 Selector returns Selector:
233     {Selector}
234
235     ('matchExpressions:' '─' matchExpressions+=MatchExpressions ( "─"

```



```

236     matchExpressions+=MatchExpressions)*    )?
237     ( 'matchLabels:' '-' matchLabels+=Mymap ( "-" matchLabels+=Mymap)*    )?
238     ;
239 DeploymentStrategy returns DeploymentStrategy:
240     {DeploymentStrategy}
241
242     ( 'type:' type=EString)?
243     ( 'rollingUpdate:' rollingUpdate=RollingUpdateDeployment)?
244     ;
245
246 PodTemplate returns PodTemplate:
247     {PodTemplate}
248
249     ( 'metadata:' metadata=Metadata)?
250     ( 'spec:' spec=PodSpec)?
251     ;
252
253 RollingUpdateDeployment returns RollingUpdateDeployment:
254
255     'maxUnavailable' maxUnavailable=EInt
256     'maxSurge:' maxSurge=EInt
257     ;

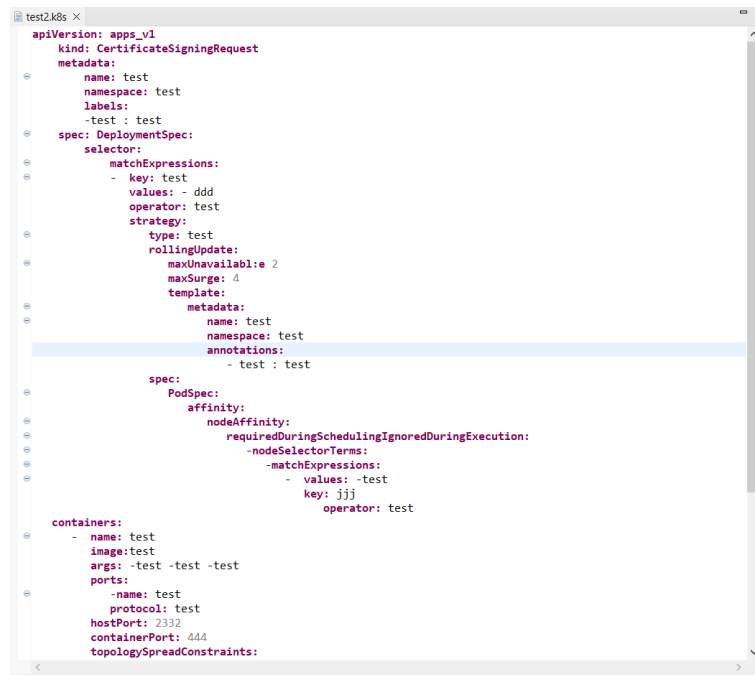
```

4.3 Tokens

Token	Token
affinity	podAffinityTerm
annotations	podAntiAffinity
apiVersion	ports
conditionType	progressDeadlineSeconds
configMap	readinessGates
containerPort	replicas
containers	requiredDuringSchedulingIgnoredDuringExecution
dnsConfig	revisionHistoryLimit
dnsPolicy	role
effect	rollingUpdate
fsGroup	runAsGroup
hostNetwork	runAsNonRoot
image	searches
key	securityContext
kind	selector
labels	seLinuxOptions
labelSelector	serviceAccountName
level	spec
matchExpressions	strategy
matchLabels	template
maxSurge	terminationGracePeriodSeconds
maxUnavailable	tolerations
metadata	topologyKey
name	type
nameservers	user
nodeAffinity	value
nodeSelectorTerms	values
octopusexport	volumes
operator	web
options	weight
podAffinity	

Kind	apiVersion
CertificateSigningRequest	certificates.k8s.io/v1beta1
ClusterRoleBinding	rbac.authorization.k8s.io/v1
ClusterRole	rbac.authorization.k8s.io/v1
ComponentStatus	v1
ConfigMap	v1
ControllerRevision	apps/v1
CronJob	batch/v1beta1
DaemonSet	extensions/v1beta1
Deployment	extensions/v1beta1
Endpoints	v1
Event	v1
HorizontalPodAutoscaler	autoscaling/v1
Ingress	extensions/v1beta1
Job	batch/v1
LimitRange	v1
Namespace	v1
NetworkPolicy	extensions/v1beta1
Node	v1
PersistentVolumeClaim	v1
PersistentVolume	v1
PodDisruptionBudget	policy/v1beta1
Pod	v1
PodSecurityPolicy	extensions/v1beta1
PodTemplate	v1
ReplicaSet	extensions/v1beta1
ReplicationController	v1
ResourceQuota	v1
RoleBinding	rbac.authorization.k8s.io/v1
Role	rbac.authorization.k8s.io/v1
Secret	v1
ServiceAccount	v1
Service	v1
StatefulSet	apps/v1

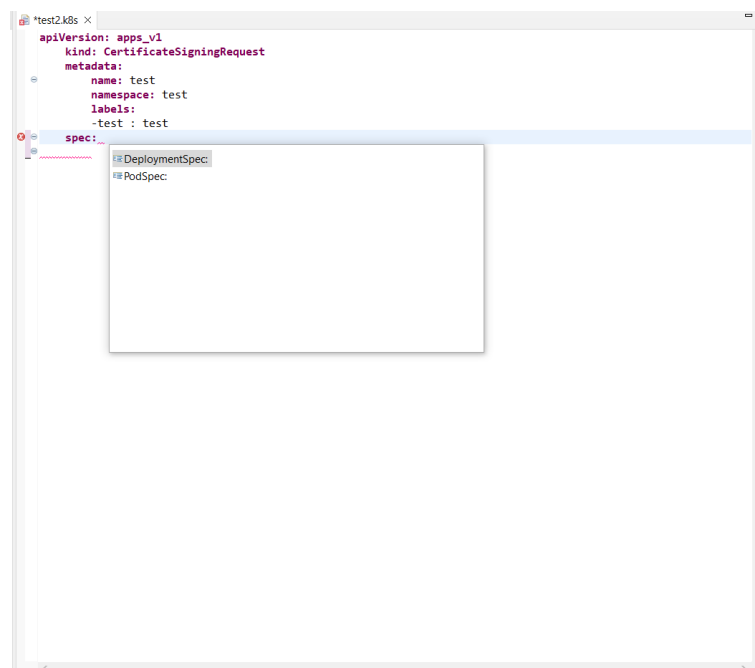
Ceci avec l'addition d'autres modules vont générer le DSL ayant l'interface suivante avec une syntaxe correcte :



```
apiVersion: apps_v1
kind: CertificateSigningRequest
metadata:
  name: test
  namespace: test
  labels:
    -test : test
spec: DeploymentSpec:
  selector:
    matchExpressions:
      - key: test
        values: - ddd
        operator: test
    strategy:
      type: test
      rollingUpdate:
        maxUnavailable: 2
        maxSurge: 4
      template:
        metadata:
          name: test
          namespace: test
          annotations:
            - test : test
        spec:
          PodSpec:
            affinity:
              nodeAffinity:
                requiredDuringSchedulingIgnoredDuringExecution:
                  -nodeSelectorTerms:
                      -matchExpressions:
                          - values: -test
                            key: jjj
                            operator: test
            containers:
              - name: test
                image: test
                args: -test -test -test
                ports:
                  -name: test
                    protocol: test
                hostPort: 2332
                containerPort: 444
                topologySpreadConstraints:
```

FIGURE 4.8 – Exemple de l'execution du DSL

Il peut aussi suggérer une saisie en respectant notre langage, ainsi qu'indiquer les erreurs détecté.



```
apiVersion: apps_v1
kind: CertificateSigningRequest
metadata:
  name: test
  namespace: test
  labels:
    -test : test
spec:
```

A dropdown menu is open below the 'spec:' line, showing two options: 'DeploymentSpec' and 'PodSpec'.

FIGURE 4.9 – Exemple de l'execution du DSL

Conclusion et perspectives

Ce projet réalisé dans le cadre de la fin de notre deuxième année nous a été très bénéfique à plusieurs niveaux.

D’abord, il était une occasion, pour se documenter sur les différents aspects du DevOps et notamment sur Kubernetes, qui est aujourd’hui en pleine expansion, sans oublier que ce projet nous a aussi permis d’appliquer nos connaissances acquises lors de notre cursus universitaire.

En matière de perspective, à cause de la contrainte du temps, nous n’avons pas pu implémenter quelques fonctionnalités tels que augmenter le support pour tout les objets de Kubernetes , afficher des exemples de fichiers Yaml si on place notre souris sur un token ou bien une interface purement graphique avec des glissements de component.

En guise de conclusion, on estime que ce projet aura du potentiel d’application vu que c’est un sujet d’actualité.

Bibliographie

- [1] https://upcommons.upc.edu/bitstream/handle/2117/329278/thesis_report.pdf. [Online].
- [2] KubernetesPatternsReusableElementsforDesigningCloud-NativeApplications. [Online].
- [3] <https://munin.uit.no/bitstream/handle/10037/9198/thesis.pdf?sequence=1>. [Online].
- [4] <https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and.html>. [Online].
- [5] <https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and.html>. [Online].
- [6] <https://ttu-ir.tdl.org/bitstream/handle/2346/85394/COGO-DISSERTATION-2019.pdf?sequence=1>. [Online].
- [7] <https://about.gitlab.com/topics/devops/>. [Online].
- [8] <https://www.cloudkubed.com/the-cloud-native-movement/>. [Online].
- [9] <https://fenix.tecnico.ulisboa.pt/downloadFile/1970719973968132/77940-Dissertation.pdf>. [Online].
- [10] <https://www.techtarget.com/whatis/definition/monolithic-architecture/>. [Online].
- [11] <https://www.janbasktraining.com/blog/what-is-docker/>. [Online].
- [12] <https://www.theseus.fi/bitstream/handle/10024/123246/DockerThesis.pdf;jsessionid=6B6D7CAE6349713F63A31E719BA3B018?sequence=1>. [Online].
- [13] <https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-application-lifecycle/containers-foundation-for-devops-collaboration>. [Online].
- [14] <https://docs.docker.com/>. [Online].
- [15] <https://dev.to/kodekloud/the-role-of-docker-in-devops-1con>. [Online].
- [16] <https://kubernetes.io/fr/>. [Online].
- [17] The Kubernetes Book (Nigel Poulton) . [Online].
- [18] https://en.wikipedia.org/wiki/Eclipse_Modeling_Framework. [Online].
- [19] https://www.eclipse.org/community/eclipse_newsletter/2014/august/article1.php. [Online].
- [20] <https://about.gitlab.com/topics/devops/>. [Online].