République Algérienne Démocratique et Populaire
الجمهوريـــة الجـزائريـة الديمـقراطيـة الشـعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التـعليـم العـالى و البـحث العلـمى

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
Ecole nationale Supérieure d'Informatique

2ème année cycle supérieure(2CS)

Option: Systèmes Informatiques (SQ)

Thème:

Etude de l'architecture logicielle du système de gestion des TPs via Kubernetes

Equipe N° 10:
- BELKESSA Linda (CE)
- DJABELKHIR Sarah
- CHELLAT Hatem
- BENHAMADI Yasmine
- BOUZOUAD Meriem
- LAMDANI Wilem

Encadré par:
- Mr CHEBIEB Abdelkarim

Promotion : 2021-2022

Résumé

Pour développer et construire une infrastructure cloud moderne ou une implémentation DevOps, Docker et Kubernetes ont révolutionné l'ère du développement et des opérations logicielles. Bien que les deux soient différents, ils unifient le processus de développement et d'intégration, il est désormais possible de construire n'importe quelle architecture en utilisant ces technologies. Docker est utilisé pour créer, expédier et exécuter n'importe quelle application n'importe où tout en permettant d'utiliser les mêmes ressources disponibles. Ces conteneurs peuvent être utilisés pour accélérer les déploiements et réduire l'espace consommé, sont fiables et sont très rapides. Kubernetes quant à lui est une plateforme automatisée de gestion, de déploiement et de mise à l'échelle des conteneurs.

Ce projet a pour but de faciliter l'exploitation des différents logiciels utilisés dans le cadre académique durant les séances de TP. Cette première partie du travail élabore le résultat d'une étude approfondie des solutions existantes, ressources, besoins des utilisateurs (enseignants et étudiants) ainsi qu'une étude théorique de l'architecture et fonctionnement des outils que nous allons utiliser, une étude comparative accompagne cette partie pour justifier nos choix de technologies. A la fin de notre travail, nous avons opté pour la combinaison docker pour la conteneurisation et kubernetes pour l'orchestration. De ce fait, une réponse aux besoins et exigences des clients a été formalisée sous forme de solution que nous traiterons durant la prochaine phase du projet.

Mots Clés : Software architecture, Client-server, Microservices, Kubernetes

Table des matières

Table des figures

Liste des tableaux

1. Introduction

En tant qu'établissement d'enseignement supérieur dans le domaine de l'informatique et des technologies, la formation pratique est une composante essentielle du cursus de l'École nationale supérieure d'informatique.

L'une des responsabilités des enseignants de cours pratiques est la mise en place et la gestion des machines des salles TPs et d'assurer la bonne installation des outils nécessaires au bon déroulement des séances de travaux et examens pratiques.

Le problème posé par les enseignants est le processus fastidieux et répétitif d'installation et de configuration des applications sur les machines des salles TPs.

Dans le cadre de notre projet 2CS, nous sommes amenés à concevoir et à implémenter une infrastructure exploitant les solutions Kubernetes et Docker, afin de faciliter la gestion des travaux et examens pratiques au sein de notre école. Le but du système est d'héberger des applications régulièrement utilisées lors des activités pratiques : telles que les IDEs, les simulateurs, ...

Sa fonction principale est de faciliter et d'optimiser la gestion, configuration, et partage de ces applications par les enseignants de L'ESI avec leurs étudiants.

La première étape de notre conception est l'établissement d'une architecture représentant l'organisation de notre système en termes de ses composants, les relations entre eux, et ses interactions avec son environnement. C'est une conception haut niveau qui a comme but de réaliser un système qui répond au mieux aux spécifications fonctionnelles tout en prenant en considération les exigences des clients et les contraintes d'implémentation.

Dans ce rapport, nous commencerons par décrire l'architecture adoptée, tout en spécifiant les besoins fonctionnelles, les exigences et les contraintes imposées par le client et la considération de système actuel, les parties prenantes, acteurs et intervenants, qui influencent et seront influencés par cette approche, et les styles architecturaux appliqués, la motivation derrière leur adoption et leur représentation sur les différents vues.

2. Les intervenants et acteurs du système

Intervenant	**Role**
Les utilisateurs	Exploitation de la solution
La direction des études	Décideur
Équipe de projet	Réalisation du projet
Chef de projet	diriger le projet
administrateurs du cluster	Maintenance,support

TABLE 2.1 – Liste des intervenants

Commanditaire (décideur) : Instance de décisions générales concernant le projet. C'est entre autres le client du projet (l'ESI -précisément la direction des études-).

Utilisateurs : Personnes auxquelles est destiné le produit. Ce sont en l'occurrence les étudiants, les enseignants et les responsables des salles TP.

Équipe projet : Ensemble de personnes contribuant à la réalisation du projet.

Chef de projet : Personne chargée de diriger le projet : soit de constituer une équipe, organiser le travail, communiquer, gérer le budget et tenir les délais.

Administrateurs du cluster : Ensemble des personnes responsable de l'exploitation informatique, assurant les missions de support aux utilisateurs, de gestion des réseaux et des systèmes, ainsi que la gestion des applications.

acteurs	utilisation
Les binomes	faire les TPs
les enseignants	demandes de déploiements ,création des images
l'administrateur du cluster	déployer les images,assurer le bon fonctionnement

TABLE 2.2 – Liste des acteurs

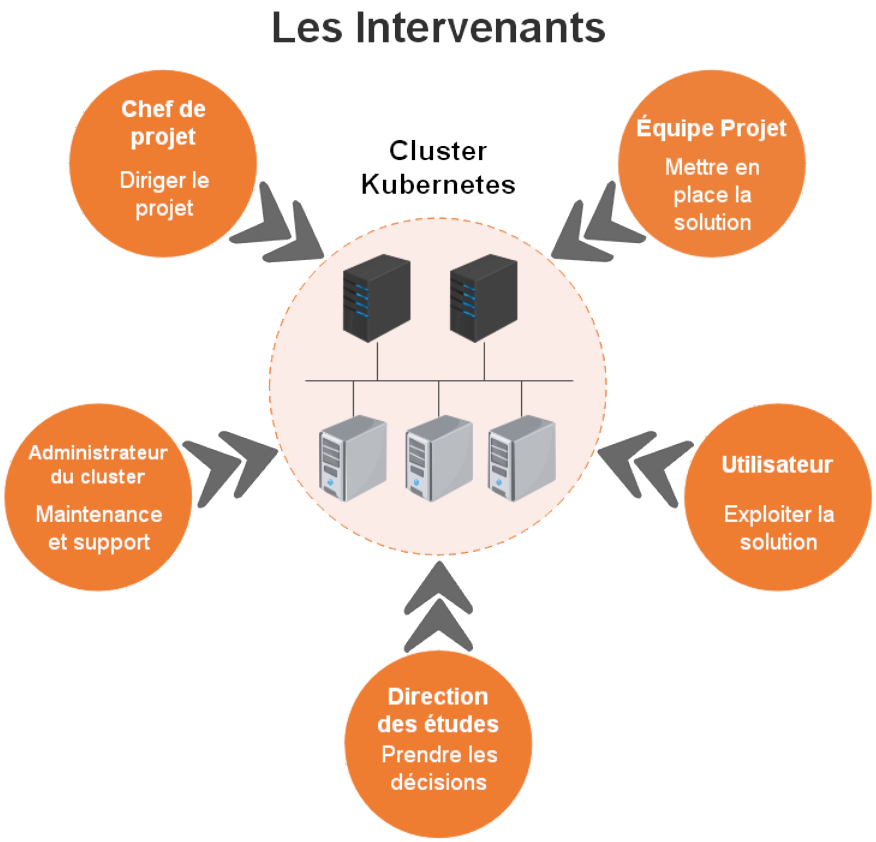


FIGURE 2.1 – Les intervenants

3. Spécification des besoins

Les besoins sont divisés en deux catégories à savoir les besoins fonctionnels et les besoins non fonctionnels : Besoins fonctionnels : Ce sont les actions et les réactions que le système doit faire suite à une demande d'un acteur. Besoins non fonctionnels : Il s'agit des besoins qui caractérisent le système, divisés entre les exigences de qualité et les contraintes techniques

Besoins fonctionnels	**Besoins non Fonctionnels (contraintes techniques + exigences de qualité)**
- Le système doit permettre aux étudiants d'accéder aux applications conteneurisées via une plateforme web. - Le système doit offrir des noms de domaine et les numéros de ports aux étudiants pour donner l'accès. - Le système doit permettre aux administrateurs de déployer les images docker des applications concernées. - Le système doit permettre aux responsables du data center d'interagir avec le cluster. - Le système doit être relativement simple d'utilisation pour les acteurs. - Le système doit offrir toutes les fonctionnalités aux administrateurs. - Le système doit permettre aux étudiants de s'authentifier pour accéder uniquement aux zones qui leurs sont réservées. - Le système doit être capable de conserver l'état des instances des applications lancées par les binômes. - Le système doit permettre aux administrateurs de joindre des worker nodes facilement. - Le système doit permettre aux administrateurs de créer les fichiers de déploiement des services/pods.	- **Exigences de qualité :** - Sécurité : Le système doit assurer un niveau de sécurité acceptable pour les utilisateurs - Performance : Le système doit avoir un temps de réponse minimal (chargement de l'application, ouverture de l'application, délais de rafraîchissements...) - Capacité : Le système doit être capable de traiter plusieurs requêtes par unité de temps, et de stocker des données. - Disponibilité : Le système doit être disponible quelque soit la période (week-ends, vacances, périodes de maintenance...) - Fiabilité : Le système doit avoir un temps d'arrêt minime (un bon temps de rétablissement) - Intégrité : Le système doit conserver l'intégrité des données des utilisateurs - Compatibilité : Le système doit être capable de fonctionner avec les différentes applications sur différents systèmes d'exploitation - Scalabilité : Le système doit être capable de s'adapter à une montée en charge ou à une multiplication des données à traiter. - Maintenabilité : Le système doit être rétabli dans les conditions de fonctionnement spécifiées, dans les limites de temps désirées. - **Contraintes techniques :** - Configuration minimale : Le système doit disposer d'une configuration prérequise minimale. - Technologies utilisées : Ubuntu/Debian comme distribution Linux et Vmware Workstation/Virtual Box comme plateforme de virtualisation.

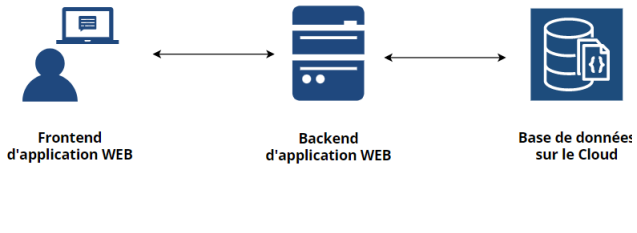
TABLE 3.1 – Recensement des besoins fonctionnels et non-fonctionnels

4. Choix su style architectural et justification

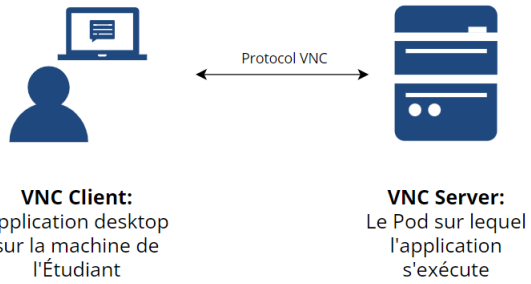
4.1 Style global choisi

Dans le cadre de ce projet et en tenant compte des besoins du client et les exigences de qualité mentionnées au début du rapport nous avons opté pour le style de client serveur comme style global Un client a besoin de communiquer avec le cluster Kubernetes qui est considéré comme serveur pour notre cas :le binôme a besoin d'accéder à l'application qui sera un service à l'intérieur du cluster (serveur).

- Tout d'abord le client accède à l'application web qui est hébergée à l'intérieur du cluster (mais reste indépendante du reste des applications cibles tournant sur le même cluster) et sera accessible par le réseau LAN de l'école
- Pour l'authentification le binôme fait entrer ses identifiants au niveau de l'interface de l'application web et le serveur de cette dernière confirme l'authentification en accédant à la base de donnée où sont stockées les informations de l'utilisateur et allume son pod en changeant le nombre de réplicats à 1 lorsqu'on est l'utilisateur est connecté , 0 si l'utilisateur est déconnecté
- Le serveur web envoie une réponse au client pour exécuter l'application du vnc viewer et accéder au service de l'application cible exemple packet tracer, via un serveur vnc.



(a) L'architecture 3 tiers



(b) L'architecture Client-Serveur VNC

4.1.1 Justification

l'architecture du client serveur se base principalement sur un client qui envoie des requêtes et le serveur qui les exécute, il est adapté pour les applications qui communiquent à travers le réseau .

les avantages du style client serveur sont :

- **une meilleure sécurité :** les points d'accès au cluster sont via une adresse ip unique juste en changeant l'intervalle de port pour chaque application au moment de déploiement selon la nécessité du module
- **une administration au niveau serveur :** le cluster kubernetes a besoin d'être administrer le déploiement et la gestion d'accès au poids et au cluster .
- **un réseau qui peut être étendu :** on peut supprimer ou ajouter autant qu'on veut de binômes sans beaucoup de changement au niveau de performance le diagramme ci-dessous montre comment ce style est vu d'une manière globale :

4.2 Style architectural de la partie serveur

Les applications dans le cluster de serveurs sont vues comme des microservices . Chaque application associée à un module est déployée comme étant un service à part entière . pour chaque service on crée des pods tels qu'un pod est associé à un seul binôme (ou monôme)

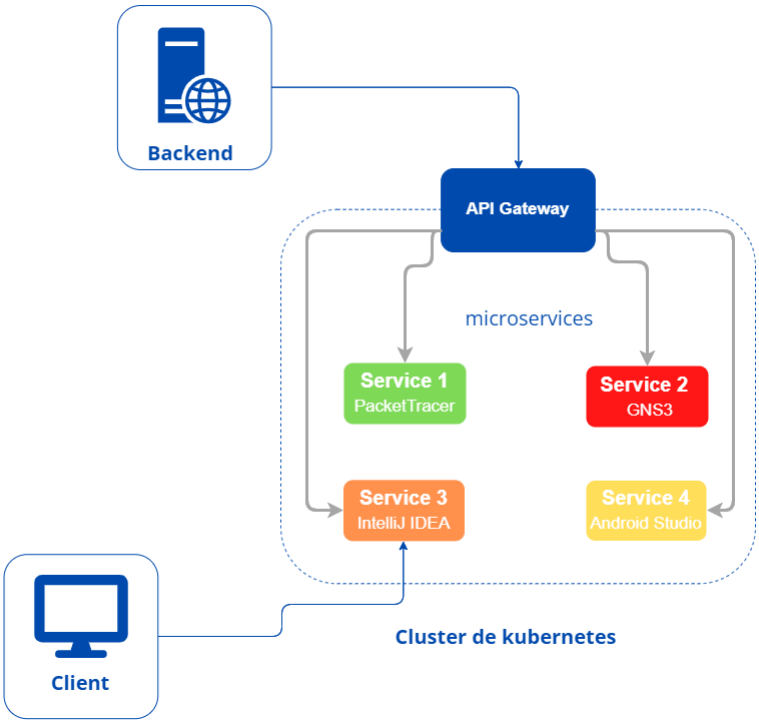


FIGURE 4.2 – Microservices

4.2.1 Justification

- chaque instance de l'application est considérée comme un service executable d'une manière autonome et indépendante des autres application
- Les services ont des codes indépendants des autres services chaque service est créé à partir d'une image docker spécifique
- Les services sont responsables de gérer leurs propres données, dans ce cas là chaque service accède à son volume pour persistance de données comme par exemple déposer le fichier de tp dans son espace.
- Bien que les services pour ce cas de projets ne communiquent pas entre eux mais il n'y a une possibilité de communication si on considère que la communication entre packet tracer et wireshark se fait à travers des pods différents chacun déployé d'une manière indépendante du reste des services
- Et un autre point important pour considérer l'architecture des microservice est que chaque service utilise des technologies et dépendances différentes
- De plus, pour plus d'adaptation aux micro services, kubernetes par défaut un orchestrateur est utilisé pour la gestion et l'équilibrage de charge au niveau des nœuds du cluster.

5. Description de l'architecture

Pour décrire notre architecture en va la présenter en utilisant deux vues :

— La vue en exécution qui sert à représenter les différents modules exécutables sur notre système, et l'organisation de ces derniers ainsi que les communications entre eux.

— La vue en déploiement qui sert à représenter les ressources utilisées pour supporter les modules de notre système, et l'allocation de ses ressources.

5.1 Vue en exécution :

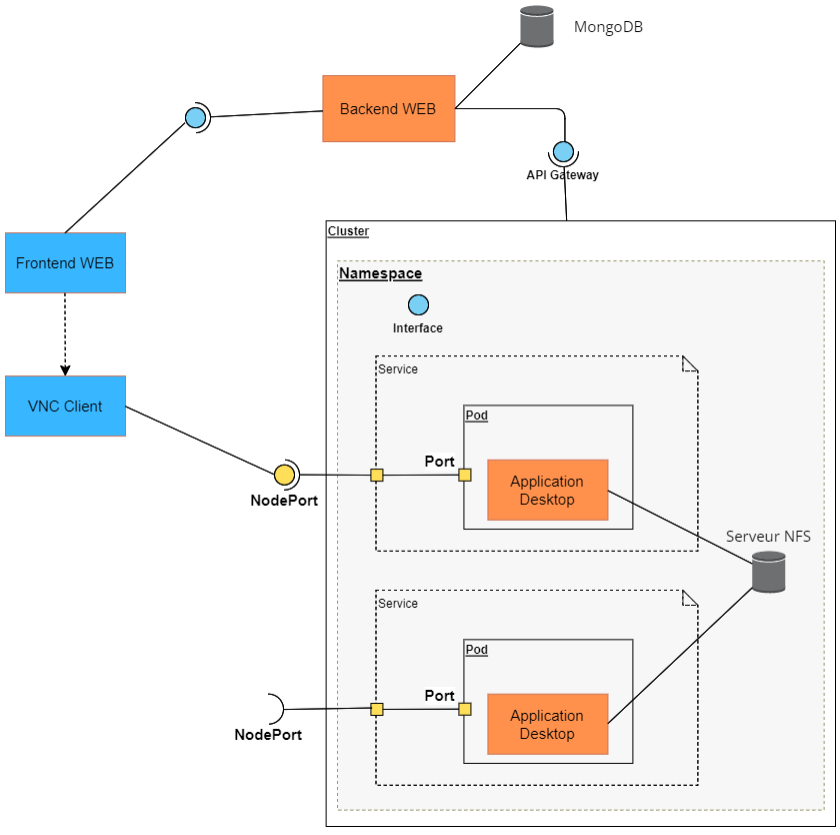


FIGURE 5.1 – Vue en exécution du système

Selon le diagramme ci-dessus, on peut distinguer 3 catégories de modules exécutables dans notre système :

1. **Les modules clients :** se sont des modules utilisés par l'acteur Étudiant pour interagir avec notre système :
   - le Frontend d'une application Web utilisée comme une interface homme-machine, qui facilite l'authentification et l'accès au cluster.
   - Le VNC client qui est une application desktop de partage d'écran graphique, elle est utilisée par l'acteur Étudiant de notre systemes pour accéder aux applications hébergés dans le Cluster.
2. **Le module Backend d'application WEB :** c'est le bloc de code qui gère la logique et les traitements de l'application Web, et qui relie les modules client avec le cluster Kubernetes. Il a 3 fonctions principales :
   - La communication avec le Cluster à travers un package appelé **@kubernetes/client-node** et le **API Gateway** pour obtenir les informations sur les Pods qui permettent au modules clients de se connecter à ces derniers.
   - La communication des informations obtenue avec le module Frontend pour présenter des informations obtenues sur les Pods.
   - L'authentification des utilisateurs et la gestion des données.
3. **Les modules sur les Cluster kubernetes :**
   - Pods : dans notre système un Pod est l'unité de déploiement la plus petite, il est considéré comme un processus sur kubernetes, qui contient une application en cours d'exécution.
   - Les services : sur kubernetes, un service représente un ensemble de Pods qui exécutent le même service(en termes de fonctionnalités), dans notre cas on a utilisé un service NodePort pour exposer ses pods, et en utilisant les numéros de ports pour y accéder.
   - Namespaces(espaces de nommage) : ils peuvent être considérés comme des clusters virtuels, qui servent à séparer les services en des domaines selon leurs utilisateurs, et l'équipe concernée par la gestion et maintenance de ses services.
   - Le API Gateway de kubernetes (appelé aussi Ingress) : c'est un module appartenant au distribution kubernetes qui assure différentes fonctionnalités. Dans cette vue en particulier, il sert au communication avec le backend de l'application Web.
4. En plus, et pour le stockage de données, notre système utilise une base de données MongoDB dans le cloud, et un serveur NFS pour l'accès aux fichiers.

5.2 Vue en déploiement :

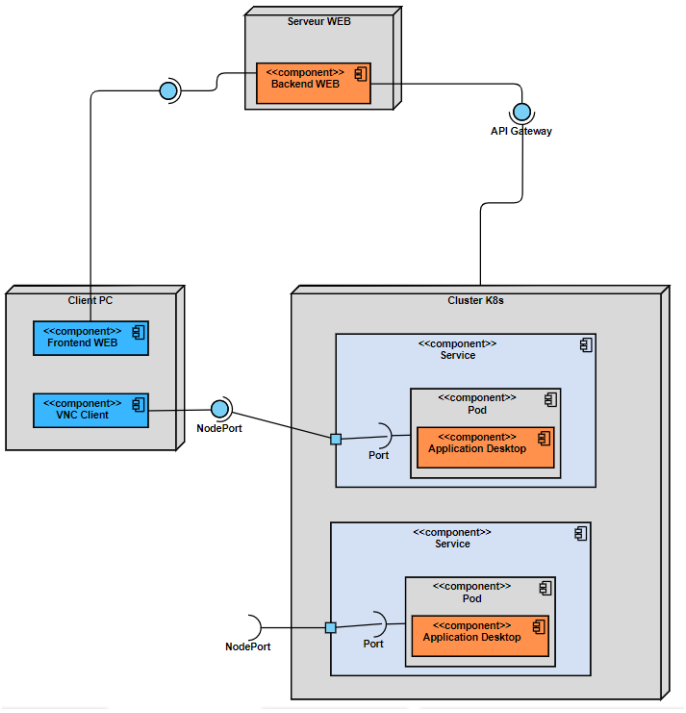


FIGURE 5.2 – Vue en déploiement du système

Le diagramme ci-dessus représente les ressource utilisée pour supporter l'exécution des différents modules de note systems :

- Les modules Clients sont exécutés sur une seule machine physique accessible par l'acteur Étudiant.
- La base de données est hébergée sur le cloud.
- Le backend de notre application est hebergé sur un serveur Web
- Et les modules kubernetes, sont hébergés sur l'infrastructure kubernetes (worker nodes)

5.3 Éléments architecturaux :

Notre architecture peut être séparée en 3 groupes des éléments architecturaux :

1. Éléments web : architecture 3 tier
   - Composants : Frontend, le backend, et la base de données.
   - Interface : codée en React, elle permet de recevoir les requêtes d'authentification et accès effectuées par l'utilisateur, et rediriger vers les routes adéquates du backend

- Serveur backend : codé en nodeJS, il communique avec la base de données hébergée sur MangoDB Atlas, il assure principalement la vérification des informations de login fournies pour ensuite lancer automatiquement le client vncviewer accompagné de l'adresse IP du node + numéro de port stocké dans la bdd.

2. Éléments de connexion client-serveur (VNC) :
   - Composants :
     — Le client : l'application VNC client.
     — le serveur : le Pod sur lequel l'application s'exécute.
   - Interfaces : Le NodePort qui représente une adresse IP + un numéro de Port.
3. Éléments de cluster : architecture microservices.
   - Composants : les services.
   - Interfaces : les ports et les NodePorts.

6. Conclusion

Pour conclure, ce travail détaille les décisions architecturales importantes qui sont prises pour assurer la livraison d'un produit final conforme aux attentes du client, en particulier lorsqu'il s'agit d'assurer les attributs et les exigences de qualité. Cette architecture client-serveur avec seevice étant des microservices, nous permet de séparer la présentation de la logique comme premier gain majeur, rendant également le serveur beaucoup plus robuste avec le déploiement des services sur Kubernetes, qui assure l'orchestration, le loadbalancing et la haute disponibilité. Notre équipe bénéficiera également de cette décision de manière à ce que le développement des deux parties du système reste faiblement couplé, puis dans le domaine des microservices, cela reste valable étant donné que le processus de déploiement est indépendant du processus de création d'images, une application majeure de ce est la capacité de donner à l'enseignant suffisamment de contrôle sur ce qu'il veut être livré aux étudiants, sans avoir à se soucier du processus de déploiement de lf exécuté par l'agent de la salle des machines.