Figure 4 :

Cette figure illustre trois commandes clés utilisées dans le processus de mise en place de l'environnement et de préparation pour le déploiement automatisé des templates NiFi. Ces commandes revêtent une importance significative pour assurer la cohérence et la préparation de l'environnement avant le déploiement.

La première commande, `minikube start --cpus 4 --memory 9000`, lance Minikube avec des paramètres spécifiques, allouant 4 cœurs CPU et 9 000 Mo de mémoire. Cette commande est cruciale pour garantir que Minikube dispose des ressources nécessaires pour simuler un cluster Kubernetes robuste, permettant ainsi l'exécution efficace des déploiements.

La deuxième commande, `kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.7.2/cert-manager.yaml`, déploie Cert-Manager, un outil utilisé pour gérer les certificats SSL/TLS dans un cluster Kubernetes. Cette étape est essentielle pour sécuriser les communications et garantir la fiabilité des déploiements.

Enfin, la troisième commande, `kubectl create namespace nifi`, crée un namespace nommé "nifi". L'utilisation de namespaces permet d'isoler les ressources et les déploiements, ce qui est crucial pour maintenir un environnement propre et organisé, tout en évitant les conflits entre différentes applications.

Figure 5 :

Cette figure nous montre que *Minikube* lui-même est un container docker. Ce container est celui qui nous simule un environnement Kubernetes sur lequel on peut travailler sur notre machine locale.

Figure 6 :

La figure 6 nous montre les commandes utilisées pour l’installation des chartes *helm* pour les deux application *Nifikop* et *Zookeeper*. Ces deux applications vont nous aider à déployer des ressources NiFi sur Minikube, notamment des *NiFiCluster* et des instances *NiFi* sur les pods du cluster. On peut également allouer des ressources de mémoire vive et de CPU avec également les limites pour ces ressources. On fait le tout sur le namespace de *nifi* qu’on a créé dans une étape précédente.

Figure 7 :

Après avoir fait l’installation des pods de NiFi, on check les podes sur ce namespace avant de créer notre *NiFiCluster* grâce au fichier yaml *persistentnificluster.yaml* qui permet de créer un cluster NiFi mono-nœud avec des volumes de persistance de Kubernetes afin de garder en mémoire tous les changements faits aux dataflows créés sur l’instance de NiFi.

Figure 8 :

On peut voir sur cette figure le fichier de configuration pour le NiFiCluster qu’on a créé.

Figure 9 :

Sur cette figure, on voir le dashboard de notre cluster Minikube sur notre machine locale. Ces figures nous montrent l’utilisation du CPU et de la mémoire vive tout en affichant les déploiements, les Pods, les Replica Sets et les Stateful Sets qui sont sur Minikube.

Figure 10 :

On peut voir le seul déploiement