Docker

Docker : Logiciel libre. Permet « d'empaqueter » une application avec toutes ses dépendances dans un bloc, un conteneur (container) standardisé, càd utilisable sur toutes sortes de machines et/ou serveurs, pour le développement de logiciels.

Les containers vont avoir un logiciel dans un système de fichiers qui va contenir tout le nécessaire (code, outils, librairies…). Permet un fonctionnement identique au logiciel choisi, avec une compatibilité améliorée car il pourra être lancé sur n'importe quel système.

Les containers sont basés sur des standards ouverts(?), leur permettant de tourner sur toutes les grosses distributions de LINUX et les OS de Microsoft avec des supports pour toutes les infrastructures.

Les containers qui tournent sur une même machine partagent le même noyau de système d'exploitation (gère les ressources d'un ordinateur et permet la communication inter-logiciels et matériels) àLancement instantané et usage plus efficace de RAM.

Les containers isolent chaque application entre elles et l'infrastructure sous-jacente(?) tout en apportant une couche de protection pour l'application.

Les containers sont similaires aux machines virtuelles au niveau de l'isolation des ressources et les avantages de l'allocation de données, mais ont une architecture différente : Tandis que les VM installe une application avec lib, code… et un OS invité ; les containers n'ayant pas d'infrastructure spécifique et qu'ils n'ont pas besoin d'OS, ils peuvent tourner sur n'importe quel ordinateur, infrastructure et cloud.

Docker permet de faire des copies de l'environnement (de travail) en temps réel et le faire tourner sur toute « machine » (??) qui tourne sous Docker.

Les containers tournant sur le même noyau tout en étant isolés entre eux, les conflits d'utilisations de plusieurs langages/outils n'existent plus. Les développeurs peuvent utiliser le langage/outil qu'ils estiment plus adapté pour leur application sans se soucier de ce problème.

L'empaquetage du logiciel nécessaire dans les containers font que l'application sera toujours lancée comme programmé localement, même s'il est lancé dans une autre machine, en test ou en prod.

Docker Hub : Pour contenir les Docker images, automatiquement partagé avec le groupe.

Mesos

Mesos est construit en utilisant les mêmes principes que le noyau Linux, mais à un niveau d'abstraction. Le noyau Mesos fonctionne sur chaque machine et fournit des applications (par exemple, Hadoop , Spark , Kafka , Elastic Search) avec l'API de gestion des ressources et la planification à travers les datacenter et de cloud computing.

Mesos se compose d'un démon maître (master daemon) qui gère démons esclaves (slave daemon) fonctionnant sur chaque nœud du cluster, et des applications de mesos (également appelés Framework) qui exécutent des tâches sur ces esclaves.

Le maître (master) permet le partage fine des ressources (CPU, RAM, ...) à travers des applications en leur faisant des offres de ressources. Chaque offre de ressources contient une liste de <ID esclave, Resource1: montant1, ressource2, amount2, ...>. Le maître décide

combien de ressources pour offrir à chaque framework selon une politique d'organisation donnée, telles que le partage équitable, ou de priorité stricte. Le maître utilise une architecture modulaire qui le rend facile d'ajouter de nouveaux modules de répartition via un mécanisme de plugin.

Un framework qui tourne sur mesos sont composé de 2 éléments : scheduler(ordonnanceur) qui enregistre avec « the master » à offrir toute les ressources et un processus d'exécution qui execute les slaves nodes qui tourne sur les frameworks.

The master (ME) alloue sufisament de ressource aux framework schedulers.

Marathon

Marathon est un framework, il fonctionne est un complément de Mesos Apache. Ces deux tournes avec le framework Chronos. Marathon lance deux instances de l'ordonnanceur Chronos comme une tâche Marathon. Si l'une des deux tâches Chronos meurt - dus aux accidents d'esclaves sous-jacentes, la perte de puissance dans le cluster, etc. - Marathon sera redémarré, et Chronos sera sur un autre esclave. Cette approche garantit que deux processus Chronos fonctionnent toujours.
Marathon gère également les autres applications qui composent notre site Web, tels que les serveurs JBoss, un service de Jetty, Sinatra, rails, et ainsi de suite.
Marathon c'est du python.

Chronos est en cours d'exécution de deux tâches. Un est chargé de la base de données MySQL de production à S3, tandis qu'un autre envoie un bulletin électronique à tous les clients par l'intermédiaire de Rake.

Sensu : Ecrit en Ruby.

Sensu est un framework libre de « suivi » flexible qui permet aux entreprises de composer eux-mêmes des solutions qui répondent à leurs besoins sans se soucier du « comment » et se concentrer sur « quoi » surveiller.

Il peut surveiller l'état d'application et de logiciel type Gmail (service systems) et revoie diverses sorties selon le type d'outil analysé, d'une appli web au logiciel type HAproxy.
Sensu peut envoyer automatiquement des notif et/ou des alertes à l'équipe avant les clients en utilisant des services de messagerie type Email, Slack…

Sensu récupère des mesures/données/indicateurs (metrics) provenant de « distributed systems »(?) de façon simplifiée. Sensu utilise des «sensu checks» qui permettent de surveiller toutes sortes de services, apps, ressources de serveurs lancé sur Sensu, et ces checks sont composés d'une sortie de statut de code et de l'info nécessaire (payload). Cela permet de prendre n'inporte quelle type de données provenant de n'importe quelle « distributed system » dans une seule plateforme.

L'API de Sensu permet d'accéder aux données des clients et des évènements, la possibilité de faire des « checks » et des résoudre des évènements. L'API donne aussi une base de données façon clé/valeur qui peut être utilisé pour diverses fonctions.

Sensu dispose d'un agent de contrôle (sensu-client) qui donne un socket (connecteur réseau ?) de type TCP ou UDP qui peut prendre des données externes JSON. Les applications peuvent se servir de ces données pour envoyer des erreurs directement vers Sensu ou passer/transférer des applications spécifiques de données de mesures.
Sensu peut utiliser les « transports » qui offre des SSL (Secure Socket Layer = protocole de sécurisation des échanges de données sur Internet) de cryptage, d'authentification et des liste de contrôle d'accès (ACL pour Access Control List : liste d'adresse et ports autorisé ou interdits par un pare-feu en réseau) granulé (plus petit niveau de détail géré (?))

Sensu contient une centaine de plugins pour pouvoir suivre efficacement votre infrastructure qui couvre divers secteurs comme les bases de données, les serveurs web et leur proxy…

Jenkins

Jenkins, de son nom original Hudson, est un outil d'intégration continue («I.C») permettant d'accélérer la livraison des logiciels en réduisant le temps d'intégration.

L'intégration consiste à partager les sources d'un projet via un serveur de Gestion de Contrôle des Sources (ou " SCM " pour *Source Control Management*), tel que Subversion, pour cela il faut que les développeurs committent régulièrement.

Des outils vont se charger de régulièrement vérifier les modifications sur le SCM pour assurer la non régression de l'application, en compilant l'ensemble du projet et en en exécutant les tests unitaires.

Les principaux intérêts de l'IC sont les suivants :

- Vérification fréquente du code, et de sa bonne compilation.
- Réalisation des tests unitaire et / ou fonctionnels, voire tests d'intégration.
- Mise à disposition éventuelle d'une version testable comportant les dernières modifications du code.
- Possibilité de créer des rapports périodiques exprimant la qualité du code, la couverture des tests. Si le projet est configuré pour Maven, alors on pourra créer le site du projet qui contiendra l'ensemble de ces rapports.

Les deux premiers avantages de cette liste montrent que l'IC permet de détecter quasi-immédiatement la présence d'un problème dans le code et donc d'y apporter une solution instantanément.

Concernant sa configuration :

Contrairement à certains autres outils d'IC, Hudson ne nécessite l'écriture ou la modification d'aucun fichier XML. Toute la configuration (qu'elle soit générale ou spécifique à un projet) se fait directement depuis l'interface en ligne d'Hudson. Sur la page principale.

L'intérêt principal d'Hudson est sa simplicité :

- simplicité d'installation

- simplicité de configuration
- simplicité d'utilisation.

Ceci fait d'Hudson un outil idéal pour disposer d'un processus d'Intégration Continue rapidement exploitable, y compris sur des projets de petite taille.