



PARIS
OPEN
SOURCE
SUMMIT

ÉDITION 2016 | 16&17 NOVEMBRE

#OSSPARIS16

UN ÉVÉNEMENT



Architecture « Big Data » Open Source SMACK





Julien Anguenot

VP Software Engineering, iland cloud

Datastax MVP for Apache Cassandra



DRaaS, Cloud Backup, Secure Cloud

 **iland**TM
www.iland.com



Les challenges du Big Data

- Sources **multiples** de données (internes / externes)
- Taille des données **augmente**
- **Scalabilité et disponibilité**
- Les systèmes doivent être **distribués** (multi-datacenters)
- **Hybride** (Cloud et sur site)
- Traitement **dynamique** des données (donnée à un temps « t »)
- Chaîne de traitement de données (**pipeline**)
- Gestion de systèmes complexes (clusters) distribués
- **Coût** (technologique et humain)

S.M.A.C.K

- Spark (moteur)
- Mesos (conteneur)
- Akka (modèle)
- Cassandra (stockage)
- Kafka (agent de message)





<https://www.apache.org/>



Kafka

- <https://kafka.apache.org>
- Système de messaging **distribué**
- Publication – Abonnement (« **pub-sub** »)
- Rapide, durable, évolutif, performant, faible latence
- Développé par LinkedIn
- Apache Foundation

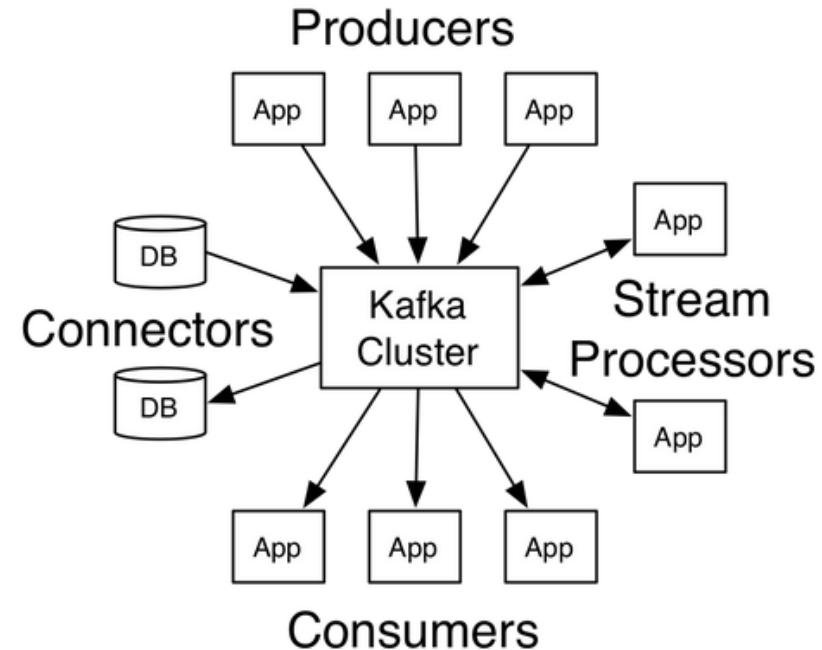


Kafka : cas d'utilisation

- Agrégation de logs
- Files d'attente
- Monitoring
- **Traitement « temps réel »**

Kafka : APIs core (1/2)

- API « **producer** »
- API « **consumer** »
- API « **stream** »
- API « **connector** »



<https://kafka.apache.org/intro>

Kafka : APIs core (2/2)

- API « **producer** » : publication de données vers un ou plusieurs brokers (topics)
- API « **consumer** » : abonnement et récupération de données depuis un ou plusieurs brokers
- API « **stream** » application consomme des données depuis 1 ou plusieurs topics et publient en sortie sur un ou plusieurs topics. (Processing et transformation)
- API « **connector** » composants extensibles réutilisables (Akka Stream par exemple)

Kafka : concepts de base (1/2)

- Les **producteurs** publient des données vers les **brokers**
- Les **consommateurs** s'abonnent et récupèrent les données depuis les brokers

Kafka : concepts de base (2/2)

- Kafka tourne en **mode cluster**
- Le cluster stocke des « streams » de données dans des **topics**
- Les topics sont divisés en **partitions** et **répliqués** fonction du facteur de réPLICATION. (configurable)
- Chaque enregistrement contient : **clef, valeur et timestamp**

Kafka : les garanties

- Les messages sont ordonnés dans l'ordre dans lequel ils sont **envoyés** par le **producteur**
- Les consommateurs reçoivent les messages dans l'ordre dans lequel ils sont **insérés** par le **producteur** dans le log
- Les messages sont délivrés **au moins 1x**
- **Si RF = N, N-1 pannes serveurs sans perde de messages commîtes**

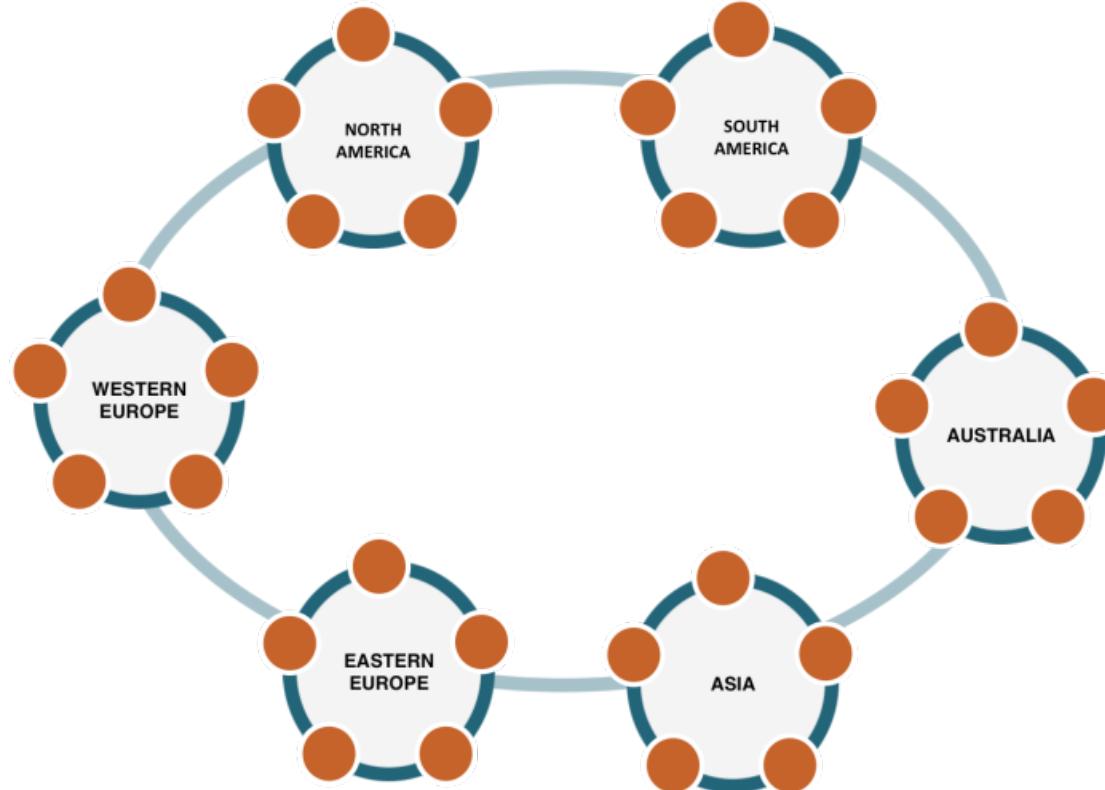
Cassandra

- Base de données **distribuée « orientée colonnes »**
- Google **Big Table** (modèle de données)
- Amazon's **Dynamo** (architecture distribuées)
- **RéPLICATION multi data-center native** (Cloud et sur site)
- Fini le sharding
- **Faible latence** en **écriture**
- **Large volume de données**
- **Élasticité** : scalabilité linéaire
- **Haute dispo** : Pas de master / slave == pas de SPOF
- <http://cassandra.apache.org/>



Pourquoi utiliser Cassandra ?

- Déploiements **massifs**
- Beaucoup **d'écritures**
- Application **distribuée géographiquement**
- Application a besoin **d'évoluer**



<http://www.datastax.com/>

Comment ça marche ?

- Écriture **séquentielle** dans un **commit log** (fichier)
- Indexation et écriture en mémoire dans des **memtables**
- **Sérialisation** des données sur disque dans des **SSTables**
- Les données sont alors **partitionnées** et **répliquer automatiquement** dans le cluster
- **SSTables** sont consolidées (fonction de la stratégie) par des **compactions** et nettoie les **tombstones**
- Les **repairs** garantissent la consistance du cluster

Cassandra : l'abstraction

- Cluster
- Data-center
- Racks
- Serveurs (l'instance Cassandra)
- Vnodes (Abstraction de stockage)

Cassandra : modèle de données

- Keyspace
- Table
- Primary Key
- Index

Cassandra : CQL

- Cassandra Query Language (CQL)
- Remplace **Thrift**
- CQL **ressemble** à du SQL
- DDL (CREATE, ALTER, DROP)
- DML (INSERT, UPDATE, DELETE, TRUNCATE)
- Query (SELECT)
- Types assez **riches** et similaires à SQL
- **Drivers et cqlsh**

lectures

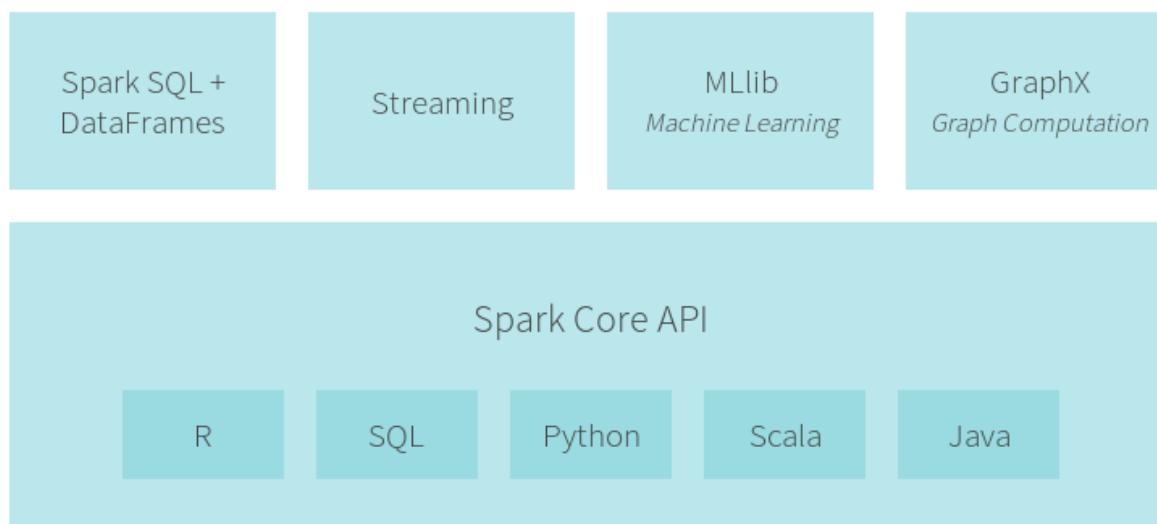
- **Dynamo: Amazon's Highly Distributed Key-Value Store:)**
http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html
- **Brewer's CAP theorem :**
<https://fenix.tecnico.ulisboa.pt/downloadFile/1126518382178117/10.e-CAP-3.pdf>

Spark

- Moteur **générique performant** pour **larges sources** de données **distribuées**
- Framework **complet** et **unifié** (type de données et sources de données)
- MapReduce ++
- Utilisation mémoire et CPU vs disque et réseau (Hadoop)
- Java, Scala, Python, Closure et R
- <https://spark.apache.org/>



Apache Spark Ecosystem



<https://databricks.com/spark/about>

Spark : l'écosystème

- **Spark Streaming** : traitement temps-réel des données en flux (Dstream, **RDD** (Resilient Distributed Dataset))
- **Spark SQL** : expose des données Spark par API JDBC - SQL, BI etc. - données extraite en JSON, Parquet, etc.
- **Spark Mlib** : bibliothèque de machine learning - classification, regression, clustering, reduction etc.
- **Spark GraphX** : API pour traitement graphs - extensions des RDD

En particulier

- **Spark Cassandra Connector :**
<https://github.com/datastax/spark-cassandra-connector>
- **Spark Streaming + Kafka :**
<http://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html>

Spark : les jobs

- Standalone
- **Mesos**
- YARN

Spark & Hadoop

- Spark ne remplace pas Hadoop
- Spark et Hadoop fonctionne très bien ensemble
- Hadoop peut être utiliser comme stockage de données distribuées (vs Cassandra)
- Spark beaucoup plus rapide que Hadoop pour les traitement en mémoire
- « fast-data » vs « slow-data »

Akka

- <http://akka.io/>
- **Toolkit et runtime pour applications concurrentes, distribuées, résilientes et « event-driven » sur JVM**
- Java & Scala : bibliothèque (JAR) ou natif SBT
- Reactive manifesto : <http://www.reactivemanifesto.org/>
- « Actor model »
- <http://akka.io/>



« Actor model »

- **Distribué**
- **Parallèle**
- **Async**
- « **Non-blocking** »
- **Tolérant à la panne** : superviseur et « let it crash » sémantique
- Le système peut tourner en **multi-JVM**
- Changements d'état du système **peuvent être sauvegardés**
- Peu de place en HEP pour des millions d'acteurs

Akka et Kafka

- **Akka Streams** : Kafka « Connector »
<http://doc.akka.io/docs/akka-stream-and-http-experimental/current/>
- **Reactive Kafka** :
<https://github.com/softwaremill/reactive-kafka>

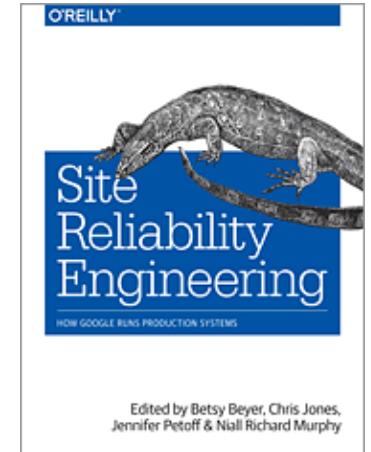
Mesos

- **Gestion de cluster** pour environnements **distribués**
- **Abstraction** (Physique, VM, Conteneur Docker) **et unification de l'ensemble des ressources** : CPU, mémoire, disque
- Allocation **dynamique** de ressources
- Gestion de l'exécution **des tâches**
- **Isolation** des ressources
- **Tolérant à la panne**
- **Optimisation des ressources** et des **coûts**
- <http://mesos.apache.org/>



Google « Borg »

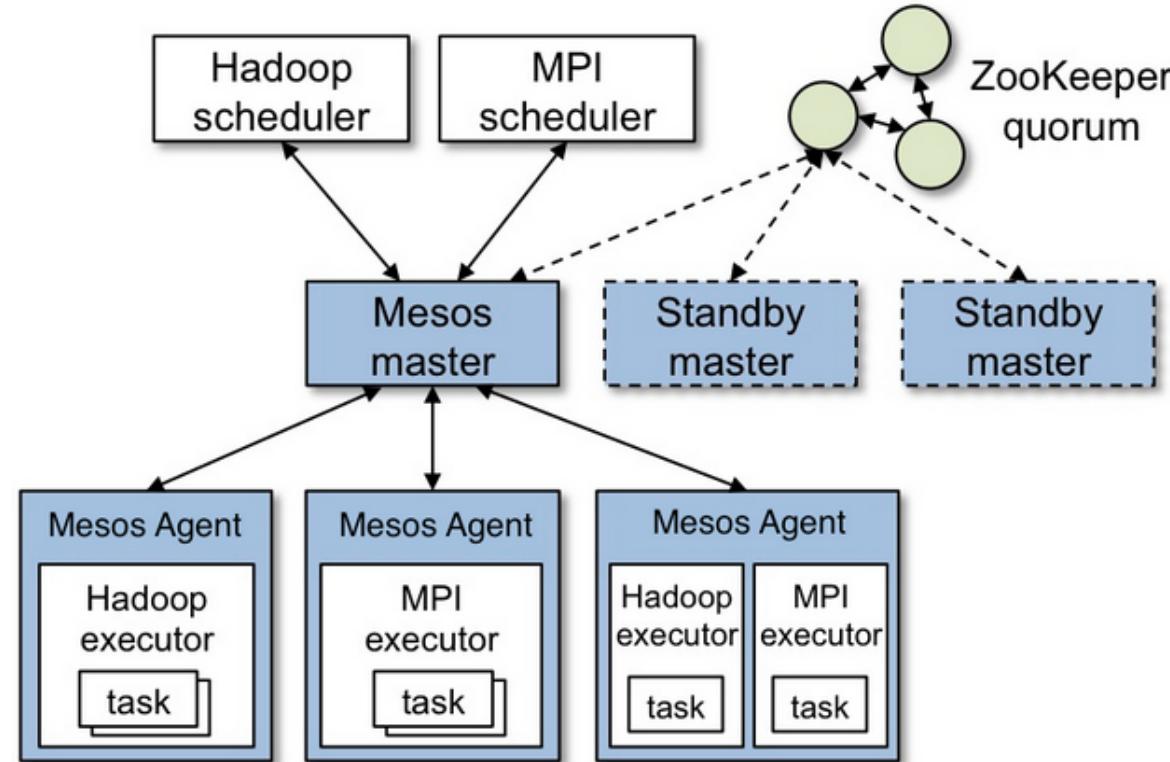
- Google « **Borg** » white paper :
<http://research.google.com/pubs/pub43438.html>
- **Site Reliability Engineering** - How Google Runs Production Systems :
<http://shop.oreilly.com/product/0636920041528.do>



Mesos : concepts de base (1/2)

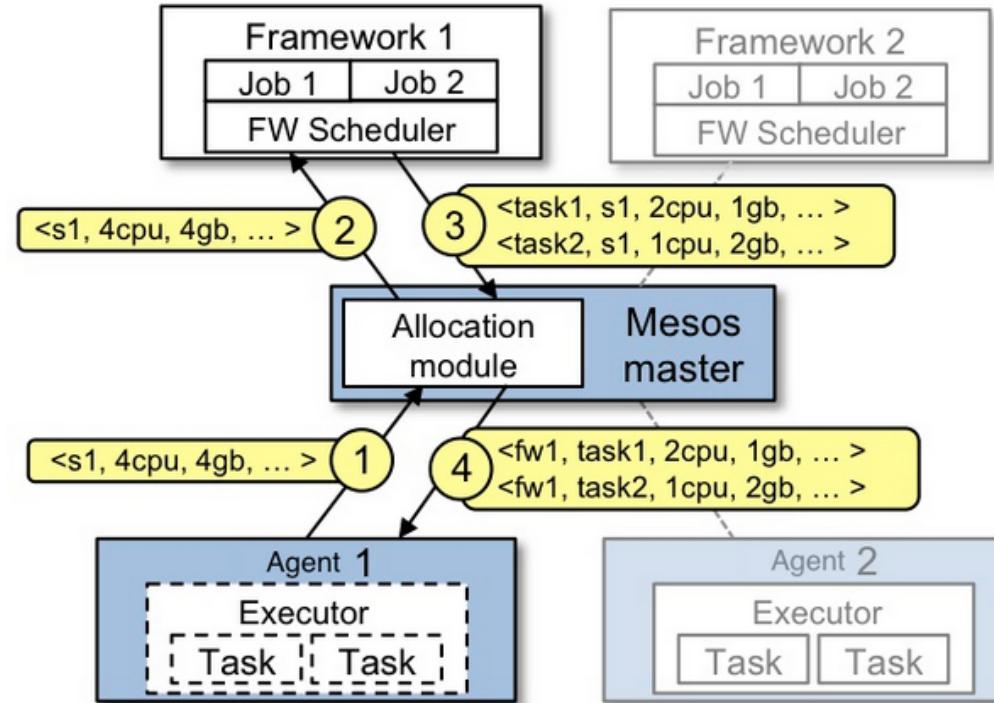
- Le **master** daemon gère des **agents** (slaves) qui tournent sur chaque nœuds du cluster
- Le master gère le partage des ressources (CPU, RAM, Disque) entre les frameworks et leur fait des « resource offer » (CPU : X, RAM : Y, DISQUE : Y)
- Le **framework** a 2 composants : **scheduler** et **executor**
- **Scheduler** : s'inscrit sur le **master** pour recevoir des offres de ressources
- **Executor** : exécuté sur **l'agent** par le **master** et lance les tâches du **framework**

Mesos : concepts de base (2/2)



<http://mesos.apache.org/documentation/latest/architecture/>

Mesos : resource offer



<http://mesos.apache.org/documentation/latest/architecture/>

Services (vs tâches) ?

- Apache Aurora :
- <http://aurora.apache.org/>

Mesos & conteneurs ?

- **Marathon** : orchestration de conteneurs
<https://github.com/mesosphere/marathon>
- cgroups ou Docker
- **DC/OS** :
<https://dcos.io/get-started/#marathon>
- Google Kurnet :
• <https://github.com/mesosphere/kubernetes-mesos>

En résumé, SMACK c'est ...

- Un ensemble de logiciels **stables** et **Open Source**
- Déploiement **Cloud** ou **sur site**
- Une **plateforme unique** pour adresser une multitudes de cas d'utilisation pour des **applications** orientées vers le traitement « temps réel » de **large volume de données**
- **Scalabilité** et **réPLICATION** des données avec une **faible latence**
- **Gestion unifiée** du cluster
- Prototypage **rapide** et à **moindre coût**

Questions

<http://slideshare.net/anguenot/>

@anguenot

