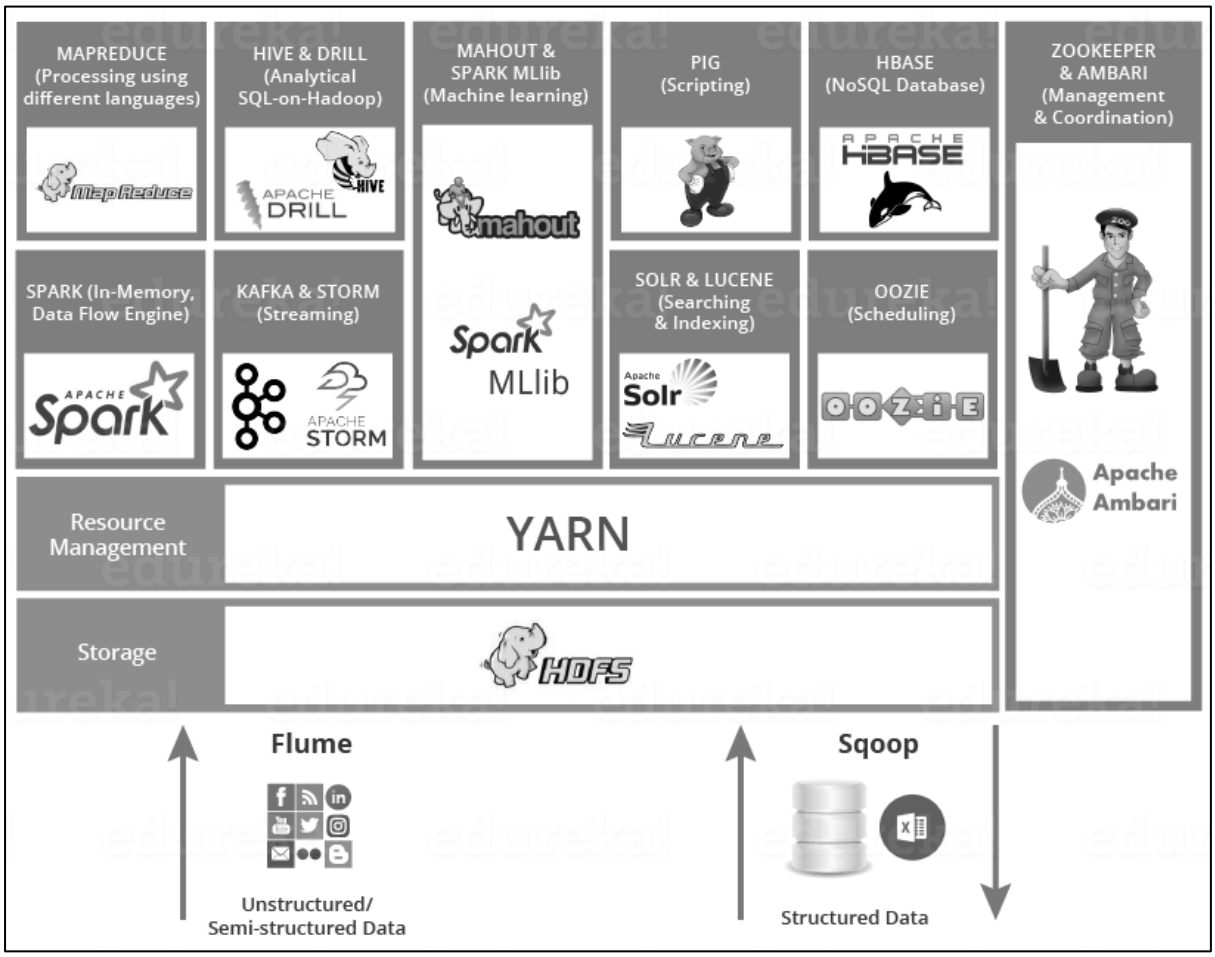
Internet of Everything (IoE)

Data Analytics for IoE

Apache Hadoop:

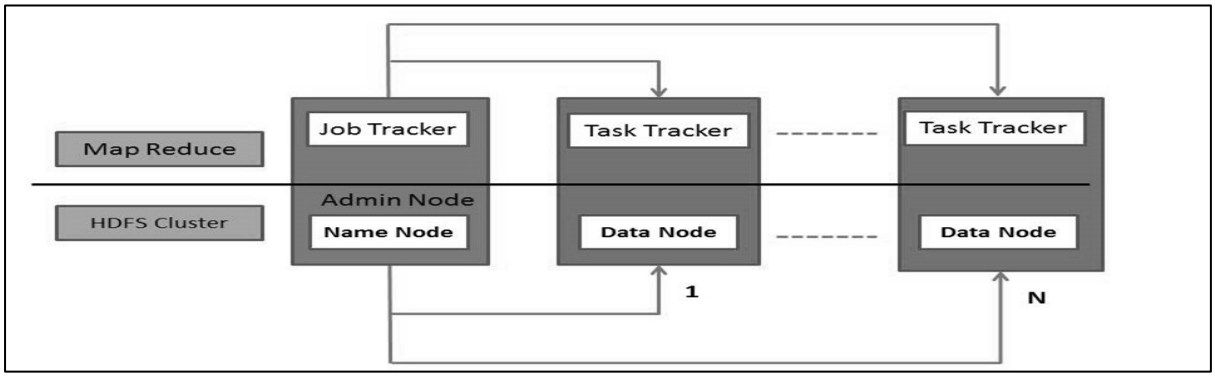
Refer Assignment No.3. for theory.



Apache Hadoop Ecosystem

Hadoop Core Components:

- HDFS – Hadoop Distributed File System (Storage)
- Map Reduce (Processing)



Nodes:

- Name Node: It is the master of the system. It maintains and manages the blocks which are present on the Data Nodes.
- Data Nodes: It is Slaves which are deployed on each machine and provide the actual storage. Responsible for serving read and write requests for the clients.
- Job Tracker: Takes care of all the job scheduling and assign tasks to Task Trackers.
- Task Tracker: A node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a job tracker.

HDFS (Hadoop Distributed File System):
Hadoop Distributed File System (HDFS) is designed to reliably store very large files across machines in a large cluster. It is inspired by the Google File System. It distributes large data files into blocks. The blocks are managed by different nodes in the clusters. Each block is replicated on multiple nodes. Name node stores metadata information about files and blocks.

Map Reduce:
The Mapper: Each block is processed in isolation by a map task called mapper. Map task runs on the node where the block is stored.
The Reducer: It consolidate result from different mappers and produces final output.

HBase:
HBase is an open source, non-relational, distributed database modeled after Google's BigTable. It is a type of NoSQL database. It is strongly consistent read and write. It runs on top of Hadoop and HDFS, providing BigTable-like capabilities for Hadoop. It provides automatic sharing, automatic region server failover, Hadoop/HDFS integration. It supports massively parallelized processing via MapReduce for using HBase as both source and sink. It supports an easy to use Java API for programmatic access. It also supports Thrift and REST for non-Java front-ends.
Uses:

- When there is real big data: millions or billions of rows, in other way data can't store in a single node.
- When random read/write access to big data
- When require to do thousands of operations on big data
- When there is no need of extra features of RDMS like typed columns, secondary indexes, transactions, advanced query languages, etc.
- When there is enough hardware.

HDFS	**HBase**
Good for storing large file	Built on top of HDFS. Good for hosting very large tables like billions of rows X millions of columns
Write once. Append to files in some of recent versions but not commonly used	Read/write many
No random read/write	Random read/write
No individual record lookup rather read all data	Fast records lookup(update)

Hive:
It is open source project, developed by Facebook. It is SQL like interface to Hadoop. It provides data warehouse infrastructure built on top of Hadoop. It also provides data summarization, query and analysis. It provides query execution via MapReduce. Hive interpreter convert the query to MapReduce format. Also used by Netflix, Cnet, Digg, eHarmony etc.

```
Ex.   SELECT customerId, max(total_cost)
      from hive_purchases
      GROUP BY customerId
      HAVING count(*) > 3;
```

Pig:
It is open source project developed by Yahoo. It's a scripting platform for processing and analyzing large data sets. Apache Pig allows to write complex MapReduce programs using a simple scripting language. It uses high level language "Pig Latin" which is a data flow language. Pig translate Pig Latin script into MapReduce to execute within Hadoop.

```
Ex.   A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int, gpa:float);
      X = FOREACH A GENERATE name,$2;
      DUMP X;
```

Pig and Hive:
Both are fault tolerant and good for batch processing and ETL jobs. Both requires compiler to generate Map reduce jobs. Hence high latency queries when used for real time responses to ad-hoc queries.

Impala:
It is open source project developed by Cloudera. Impala is a query engine that runs on Apache Hadoop. It is similar to HiveQL. It does not use MapReduce. It is optimized for low latency queries. It is much faster than Hive or pig.

Description of Feature	**Pig**	**Hive**	**Impala**
SQL based query language	No	yes	yes
Schema	optional	required	required
Process data with external scripts	yes	yes	no
Extensible file format support	yes	yes	no
Query speed	slow	slow	fast
Accessible via ODBC/JDBC	no	yes	yes

Sqoop:
It is command-line interface for transforming data between relational database and Hadoop. It supports incremental imports. Imports use to populate tables in Hadoop. Exports use to put data from Hadoop into relational database such as SQL server.

The dataset being transferred is broken into small blocks. Map's only job is to launch. Individual mapper is responsible for transferring a block of the dataset.

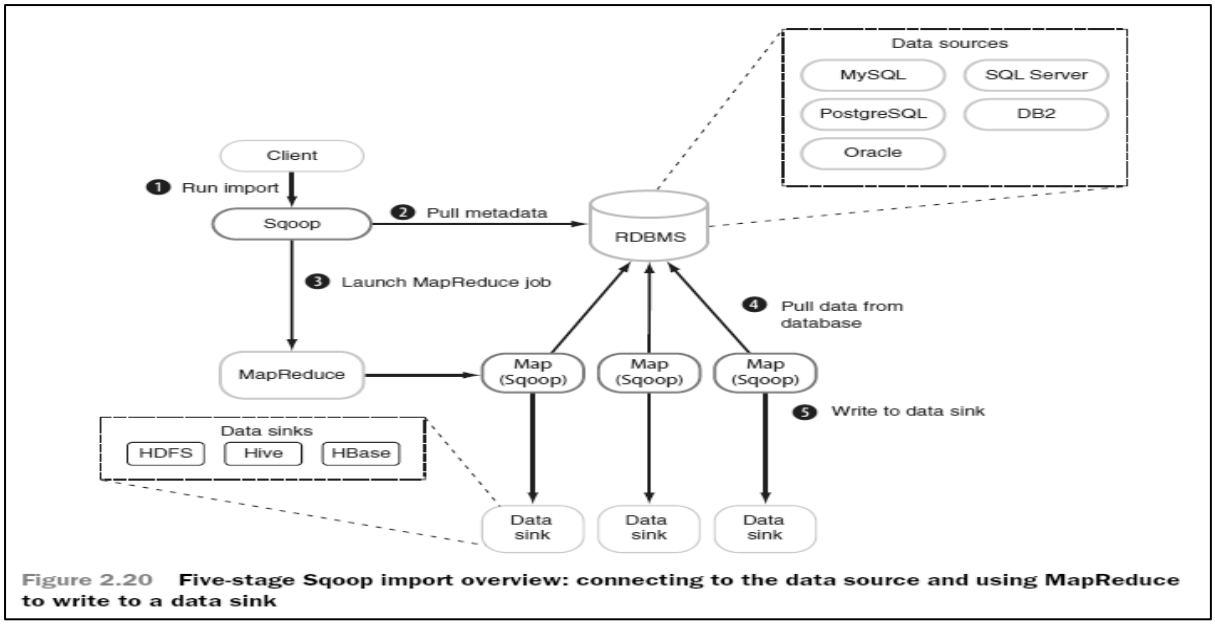


Figure 2.20 **Five-stage Sqoop import overview: connecting to the data source and using MapReduce to write to a data sink**

Flume:

Apache Flume is a distributed, reliable and available service for efficiently collecting, aggregating and moving large amounts of streaming data into the Hadoop Distributed File System (HDFS).

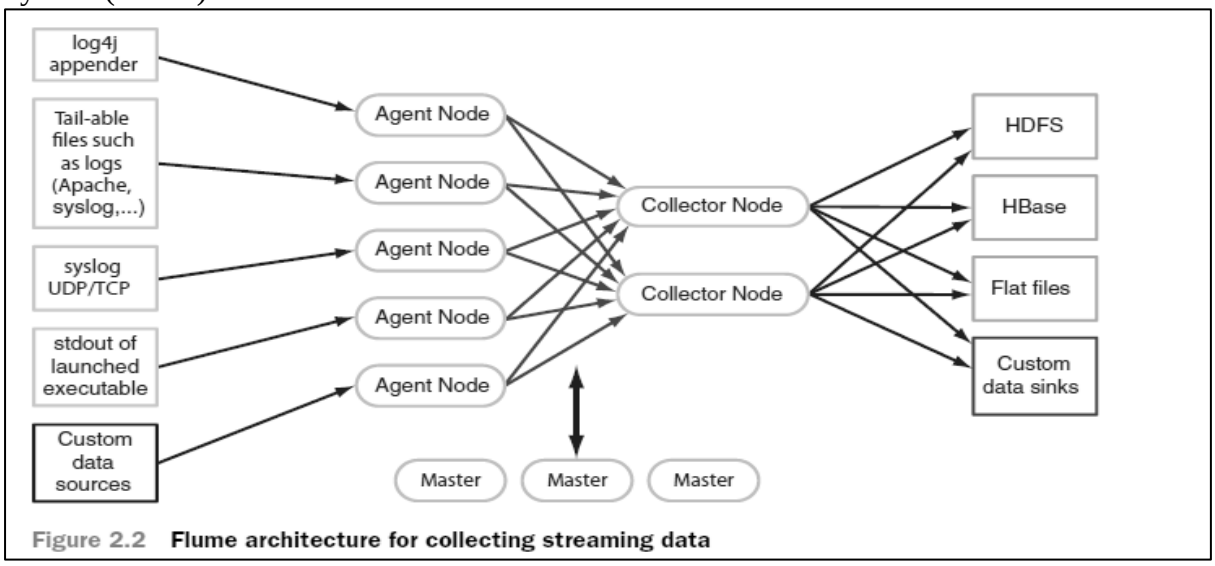


Figure 2.2 **Flume architecture for collecting streaming data**

Data flows like: Agent tier -> Collector tier -> Storage tier

Agent nodes are typically installed on the machines that generate the logs and are data's initial point of contact with Flume. They forward data to the next tier of collector nodes, which aggregate the separate data flows and forward them to the final storage tier.

Hue:

It is open source web interface. It is graphical front end to the cluster which makes Hadoop platform (HDFS, Map reduce, oozie, Hive, etc.) easy to use.

ZooKeeper:

Since coordinating distributed systems is a Zoo. ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

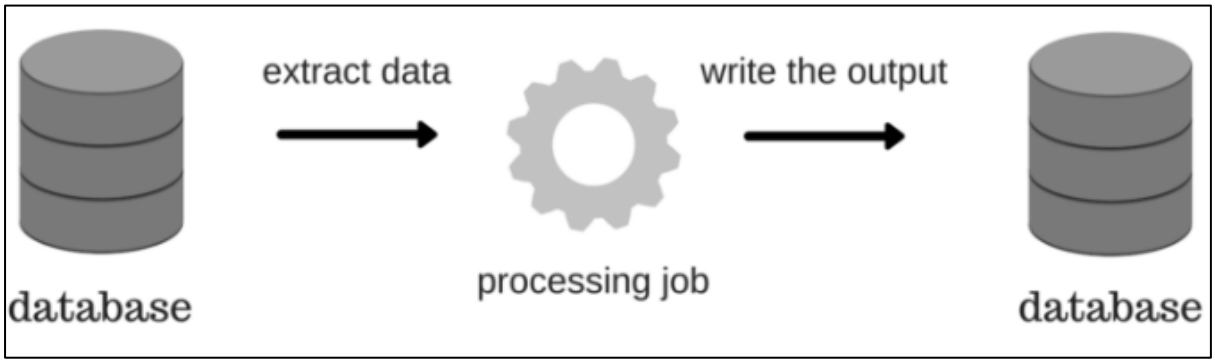
Apache Hadoop Limitations:

1. Security Concerns
2. Vulnerable by nature
3. Not fit for small data
4. Potential stability issue
5. General Limitations

Using MapReduce for Batch Data Analysis:

Batch processing is an automated job that does some computation, usually done as a periodical job. It runs the processing code on a set of inputs, called a batch. Usually, the job will read the batch data from a database and store the result in the same or different database.

An example of a batch processing job could be reading all the sale logs from an online shop for a single day and aggregating it into statistics for that day (number of users per country, the average spent amount, etc.). Doing this as a daily job could give insights into customer trends.

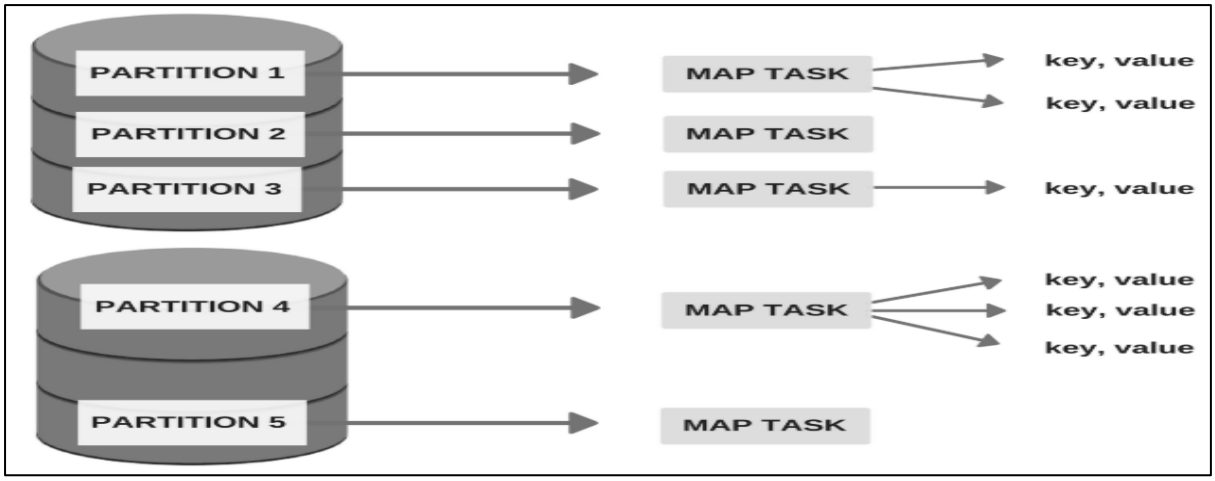


MapReduce:

MapReduce is a programming model that was introduced in a white paper by Google in 2004. Today, it is implemented in various data processing and storing systems (Hadoop, Spark, MongoDB, …) and it is a foundational building block of most big data batch processing systems. For MapReduce to be able to do computation on large amounts of data, it has to be a distributed model that executes its code on multiple nodes. This allows the computation to handle larger amounts of data by adding more machines – horizontal scaling. This is different from vertical scaling, which implies increasing the performance of a single machine.

Map:

Hadoop, along with its many other features, had the first open-source implementation of MapReduce. It also has its own distributed file storage called HDFS. In Hadoop, the typical input into a MapReduce job is a directory in HDFS. In order to increase parallelization, each directory is made up of smaller units called partitions and each partition can be processed separately by a map task (the process that executes the map function). This is hidden from the user, but it is important to be aware of it because the number of partitions can affect the speed of execution.



The map task (mapper) is called once for every input partition and its job is to extract key-value pairs from the input partition. The mapper can generate any number of key-value pairs from a single input (including zero, see the figure above). The user only needs to define the code inside the mapper. Below, we see an example of a simple mapper that takes the input partition and outputs each word as a key with value 1.

Source Code:

```
def mapper(key, value):
# Split the text into words and yield word,1 as a pair
for word in value.split():
normalized_word = world.lower()
yield normalized_word, 1
```

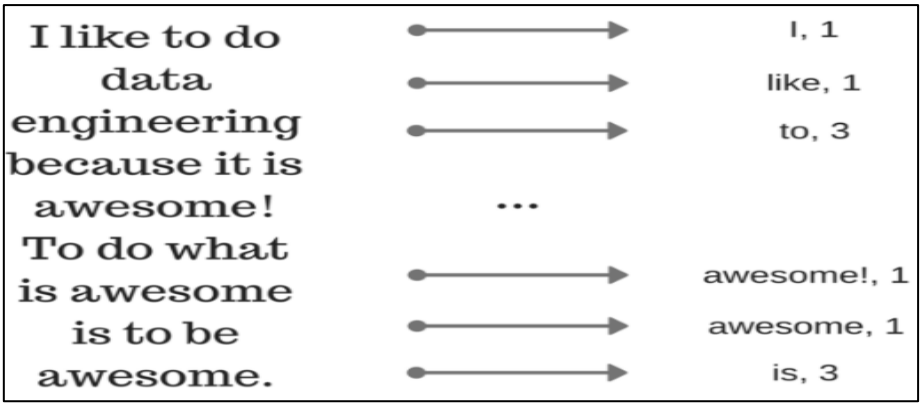
Reduce:

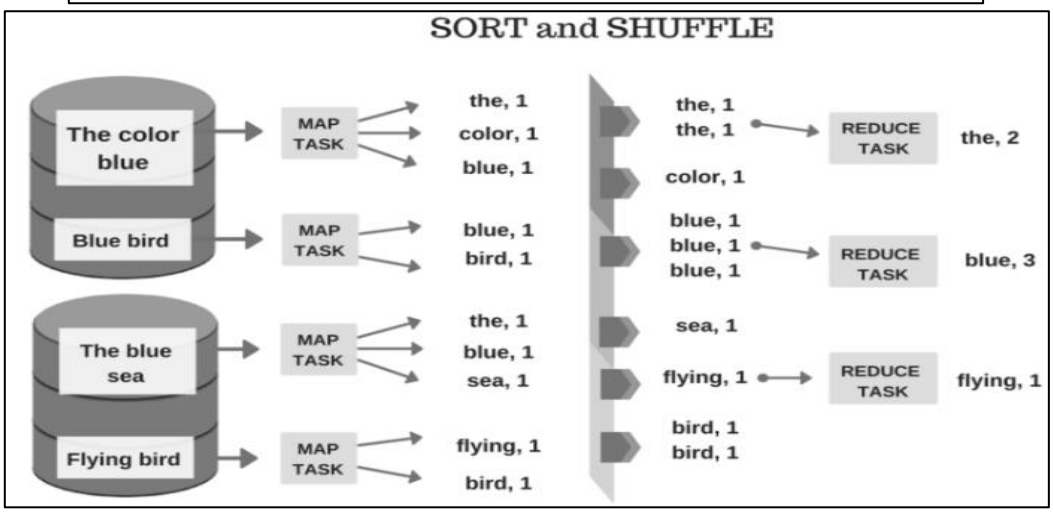
The MapReduce framework collects all the key-value pairs produced by the mappers, arranges them into groups with the same key and applies the reduce function. All the grouped values entering the reducers are sorted by the framework. The reducer can produce output files which can serve as input into another MapReduce job, thus enabling multiple MapReduce jobs to chain into a more complex data processing pipeline.

Source Code:

```
def reducer(key, values):
# Sum all the values with the same key
result = sum(values)
return result
```

The mapper yielded key-value pairs with the word as the key and the number 1 as the value. The reducer can be called on all the values with the same key (word), to create a distributed word counting pipeline. In the image below, we see that not every sorted group has a reduce task. This happens because the user needs to define the number of reducers, which is 3 in our case. After a reducer is done with its task, it takes another group if there is one that was not processed.





Apache Oozie:

It is a server-based workflow scheduling system to manage Hadoop jobs. Workflows in Oozie are defined as a collection of control flow and action nodes in a directed acyclic graph. Control flow nodes define the beginning and the end of a workflow (start, end, and failure nodes) as well as a mechanism to control the workflow execution path (decision, fork, and join nodes). Action nodes are the mechanism by which a workflow triggers the execution of a computation/processing task. Oozie provides support for different types of actions including Hadoop MapReduce, Hadoop distributed file system operations, Pig, SSH, and email. Oozie can also be extended to support additional types of actions.

Oozie workflows can be parameterized using variables such as ${inputDir} within the workflow definition. When submitting a workflow job, values for the parameters must be provided. If properly parameterized (using different output directories), several identical workflow jobs can run concurrently. Oozie is implemented as a Java web application that runs in a Java servlet container and is distributed under the Apache License 2.0.

Apache Spark:

Apache Spark has as its architectural foundation the Resilient Distributed Dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. The Dataframe API was released as an abstraction on top of the RDD, followed by the Dataset API. In Spark 1.x, the RDD was the primary application programming interface (API), but as of Spark 2.x use of the Dataset API is encouraged even though the RDD API is not deprecated. The RDD technology still underlies the Dataset API. Apache Spark requires a cluster manager and a distributed storage system. For cluster management, Spark

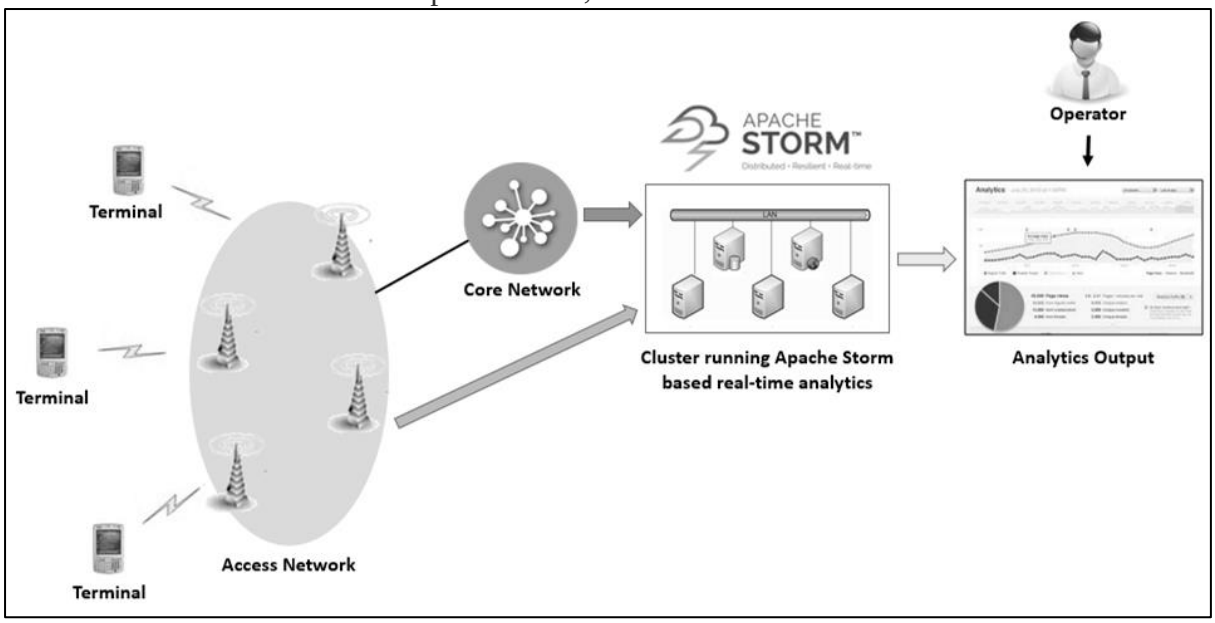
supports standalone (native Spark cluster, where you can launch a cluster either manually or use the launch scripts provided by the install package. It is also possible to run these daemons on a single machine for testing), Hadoop YARN, Apache Mesos or Kubernetes. For distributed storage, Spark can interface with a wide variety, including Alluxio, Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, Lustre file system, or a custom solution can be implemented. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in such a scenario, Spark is run on a single machine with one executor per CPU core.

Apache Storm: *Refer Assignment No 2,3.*

Using Apache Storm for real-time data analysis:

Apache Storm based real-time analytics solution, using an example of a telecom service provider. In the network of a telecom service provider, there can be different sources of incoming data, like:

- Stream of data generated due to use of services by subscribers
- Performance data of access network, as reported by network probes
- Data related with new subscription orders, activation and terminate orders



A single storm topology running inside a cluster can listen to all different kinds of incoming data. The storm topology may have different spouts for each source of input. Spouts are followed by Bolts which take care of processing the incoming data and calculate statistics which captures information across services, regarding the quality of service provided to the subscribers, performance of network elements and new received orders.

The processed statistics are stored in a NoSQL database for future reference and are also used to plot several services-related real-time charts. The analytics can be used to device a solution that can automatically detect under-performing network elements and reconfigure the network such that it is more balanced with respect to its utilization.

Similarly, network coverage related issues can be tracked and input can be provided to network planning, in real-time. Also, the analytics charts can be monitored by proactive call center operators, who can reach out to subscribers (who are facing service-related issues) with attractive offers to replace their service plan or terminal which suits better to their usage pattern.

Such seamless handling of network-related issues and proactive customer support can be instrumental in improving user experience manifold.
It can help a telecom service provider in building a competitive advantage in the market, by maintaining a real-time focus on customer experience.
Apache Storm provides a stable and robust framework for a real-time analytics solution. The framework provides base classes for spouts and bolts. Spout class inherits class BaseRichSpout and bolt class inherits BaseRichBolt. One is required to just implement nextTuple() method in spout class such that it reads data from an incoming data stream and emits it inside the storm topology. Similarly, one has to write the implementation of execute() method in bolt class to provide business logic to process the data passed on by the connected spout.
Multiple spouts can be defined for different sources of data. For example, one spout for tapping into charging data, second to tap performance data from the access network and third spout for accessing data from incoming order requests.

Benefits of using Storm:

- It allows real-time stream processing.
- Scalability – where throughput rates of even one million 100 byte messages per second per node can be achieved.
- Low latency – Storm performs data refresh and end-to-end delivery response in seconds or minutes depends upon the problem.
- Reliable – Storm guarantees that each unit of data (tuple) will be processed at least once or exactly once. Messages are only replayed when there are failures.
- Easy to operate – standard configurations are suitable for production on day one. Once deployed, Storm is easy to operate.
- Fault-tolerant: The ability of fault-tolerant is extremely important for storm as it processes massive data all time and should not be interrupted by a minimal failure, such as hardware fail in nodes of the storm cluster. Storm can redeploy tasks when it is necessary.
- Data guarantee: No data loss is one of the essential requirements for a data processing
- system. The risk of losing data would not be accepted in the use of most fields, especially for those ask for accurate results. Storm makes sure that all the data would be processed as they are designed during their processing in the topology.
- Ease of use in deploying and operating the system.
- Support for multiple programming languages.
- Fraud can be detected the moment it happens and proper measures can be taken to limit the damage.

SHM: *Refer Assignment No 2,3 as well case study.*

Tool for IoT:
Puppet:
Puppet is a Configuration Management tool that is used for deploying, configuring and managing servers. It performs the following functions:

- Defining distinct configurations for each and every host, and continuously checking and confirming whether the required configuration is in place and is not altered (if altered Puppet will revert back to the required configuration) on the host.
- Dynamic scaling-up and scaling-down of machines.
- Providing control over all your configured machines, so a centralized (master-server or repo-based) change gets propagated to all, automatically.

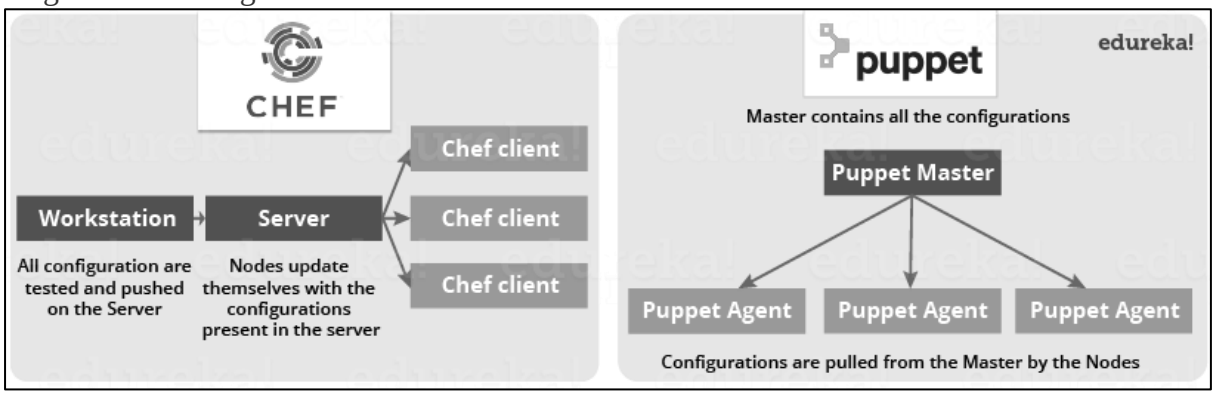
Puppet uses a Master Slave architecture in which the Master and Slave communicate through a secure encrypted channel with the help of SSL.

Chef: Chef is a tool used for Configuration Management and is closely competing with *Puppet*. Chef is an automation tool that provides a way to define infrastructure as code. Infrastructure as Code (IAC) simply means that managing infrastructure by writing code (Automating infrastructure) rather than using manual processes. It can also be termed as programmable infrastructure. Chef uses a pure-Ruby, domain-specific language (DSL) for writing system configurations.

Chef supports multiple platforms like AIX, RHEL/CentOS, FreeBSD, OS X, Solaris, Microsoft Windows and Ubuntu. Additional client platforms include Arch Linux, Debian and Fedora. It can be integrated with cloud-based platforms such as Internap, Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, Microsoft Azure and Rackspace to automatically provision and configure new machines. Chef has an active, smart and fast-growing community support. Because of Chef's maturity and flexibility, it is being used by giants like Mozilla, Expedia, Facebook, HP Public Cloud, Prezi, Xero, Ancestry.com, Rackspace, Get Satisfaction, IGN, Marshall University, Socrata, University of Minnesota, Wharton School of the University of Pennsylvania, Bonobos, Splunk, Citi, DueDil, Disney, and Cheezburger. Below are the types of automation done by Chef, irrespective of the size of infrastructure:

- Infrastructure configuration
- Application deployment
- Configurations are managed across your network

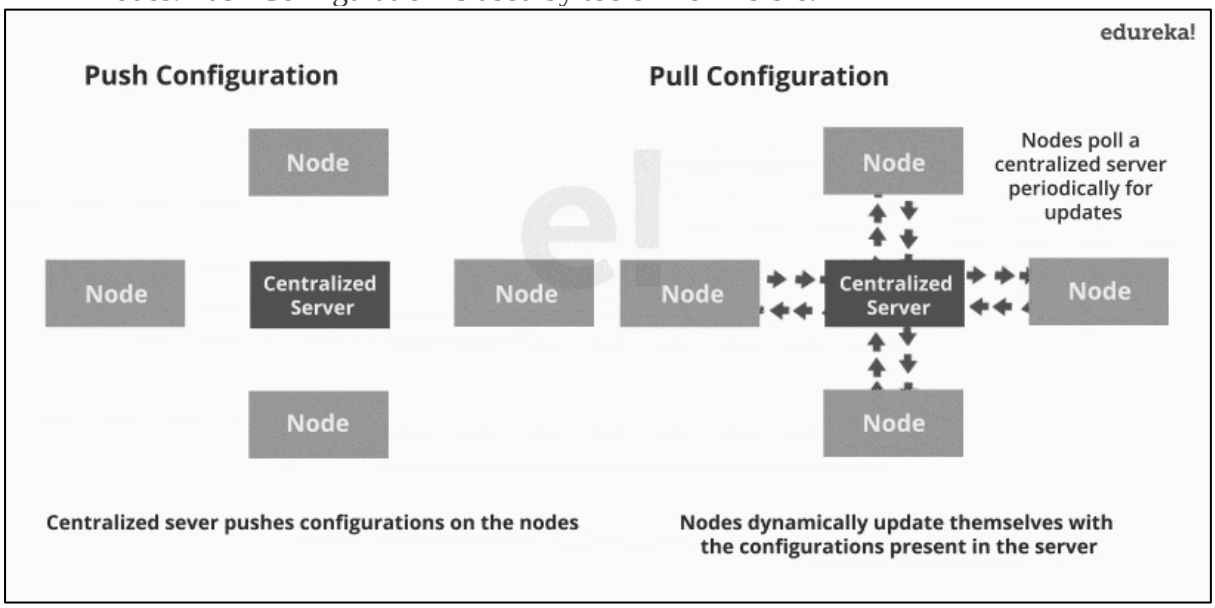
Like ***Puppet*** which has a Master-Slave architecture even Chef has a Client-Server architecture. But Chef has an extra component called Workstation. I will talk about workstation in my next blog. Refer the diagram below:



In Chef, Nodes are dynamically updated with the configurations in the Server. This is called **Pull Configuration** which means that we don't need to execute even a single command on the Chef server to push the configuration on the nodes, nodes will automatically update themselves with the configurations present in the Server
There are broadly two ways to manage your configurations namely Push and Pull configurations:

- **Pull Configuration:** In this type of Configuration Management, the nodes poll a centralized server periodically for updates. These nodes are dynamically configured so basically, they are pulling configurations from the centralized server. Pull configuration is used by tools like Chef, Puppet etc.

- **Push Configuration:** In this type of Configuration Management, the centralized Server pushes the configurations to the nodes. Unlike Pull Configuration, there are certain commands that have to be executed in the centralized server in order to configure the nodes. Push Configuration is used by tools like Ansible.



Chef Use Case:

Gannett is a publicly traded American media holding company. It is the largest U.S. newspaper publisher as measured by total daily circulation. It is the largest U.S. newspaper publisher as measured by total daily circulation. Gannett's traditional deployment workflow was characterized by multiple handoffs and manual tests.

Problems faced by Gannett with this process

- Maintaining accurate, repeatable builds was difficult.
- There were many build failures and tests were often running in the wrong environments.
- Deployment and provisioning times could range from a few days to several weeks.
- Operations team didn't have access to the cloud or development environments.
- Every group used its own tool-set, and there was no accountability to finance or security. No one knew how much an application actually cost. Security had no way to audit the software stacks.

Resolutions

Gannett was ready for the change. Developers wanted to deploy their applications quickly. Operations wanted a stable infrastructure where they could build and deploy in a repeatable way. Finance wanted to know the true cost of an application. Security wanted to view and audit all stacks and to be able to track changes.

Gannett saw that cloud as a service offered many advantages. Developers had access to standardized resources. It was easier to handle peaky traffic because of cloud's compute-on-demand model, and handoffs were minimized.

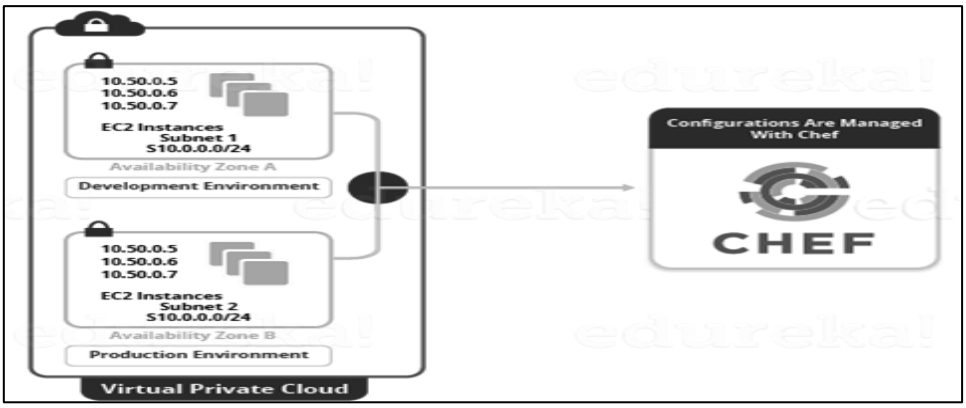
Roleplay by Chef

Chef allows you to dynamically provision and de-provision your infrastructure on demand to keep up with peaks in usage and traffic. It enables new services and features to be deployed and updated more frequently, with little risk of downtime. With Chef, you can take advantage of all the flexibility and cost savings that cloud offers.

Let us see what were the functions performed by Chef at Gannett:

- Gannett started building VPC (Virtual Private Cloud) for development environment that would mimic the production. None of the tools that they were already using were appropriate. But they found that Chef worked well with the cloud and both Linux and Windows environment. They used Chef to build a development environment that perfectly matched production environment.
- For an application to move into the VPC, it had to be provisioned and deployed with Chef.
- Security would be involved early on and would manage the mandatory controls for access to Chef and for maintaining system security standards.

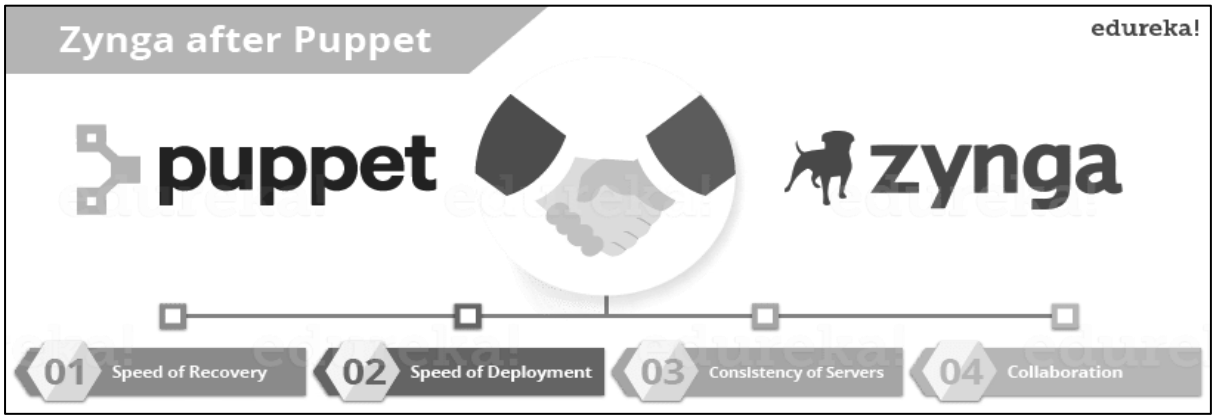
VPC Deployment using Chef



Results

- Gannett's deployment became quicker and more reliable. Application provisioning and deployment, which once took weeks, after using Chef it took minutes.
- All new applications were deployed on the cloud with Chef. These applications were deployed to all environments in the same way that they were deployed to production. Also, testing occurred in each environment, so that the deployments were reliable.
- All infrastructure was treated as code, which greatly increases visibility into any changes that occurred. Development, Operations, Security and Finance all were benefited from this.

Puppet Case Study:



If you are a poker enthusiast or if you have ever played online games, then you must have heard about Zynga. It is the world's largest social game developer. Zynga's infrastructure uses tens of thousands of servers in both public cloud and private data centers. Early on they were using a manual process, including kickstarters and post installs to get hundreds of servers online. The company was smart enough to quickly realize the need for an automated process even before they hit rapid scaling, that's when Puppet came into the picture. Let us understand how Puppet contributes to their organization.

Problems with Zynga:

- **Scalability & Consistency**: Zynga was experiencing phenomenal growth and its infrastructure needed to keep pace with the industry. Script-based solutions and manual approaches were not sufficient for their needs.
- **Portable Infrastructure**: Zynga needed a way to leverage a consistent configuration management approach in both their public cloud infrastructure and their own data centers.
- **Flexibility**: Given the diversity of the various Zynga gaming properties, it was important for the team to be able to quickly match the right configuration for the right machine.
- **Infrastructure Insights**: As the organization matured, it became more important to have an automated method of visualizing the properties of each machine.

NETCONF-YANG: *Refer Assignment No 2,3 as well as case study.*

Multi-tier deployment:

Multitier deployment provides sites the flexibility of installing packages on workstations and servers from more than one deployment location and more than one deployment server. These additional deployment locations and servers are called deployment tiers. Specifically, instead of installing multiple workstations across a wide area network (WAN) circuit, multitier deployment enables you to transfer a compressed package from the centralized location to the remote workgroup server, which acts as a second deployment tier. Multitier deployment means deploying from more than one deployment tier.

For example, you might have one deployment server at the main location and a second deployment server for a remote location. Because the server at the remote location is responsible for deploying to workstations and servers at that location, you do not need to deploy packages from the main deployment server across a WAN, as you would in a single-tier deployment configuration.

IoT Code Generator:

The figure shows how the heterogeneous code generator backend fits into the Orcc architecture. The heterogeneous backend knows, which parts of the dataflow application are assigned to which homogeneous backend, partitions the graph and hands parts of the graph to their associated homogeneous code generators.