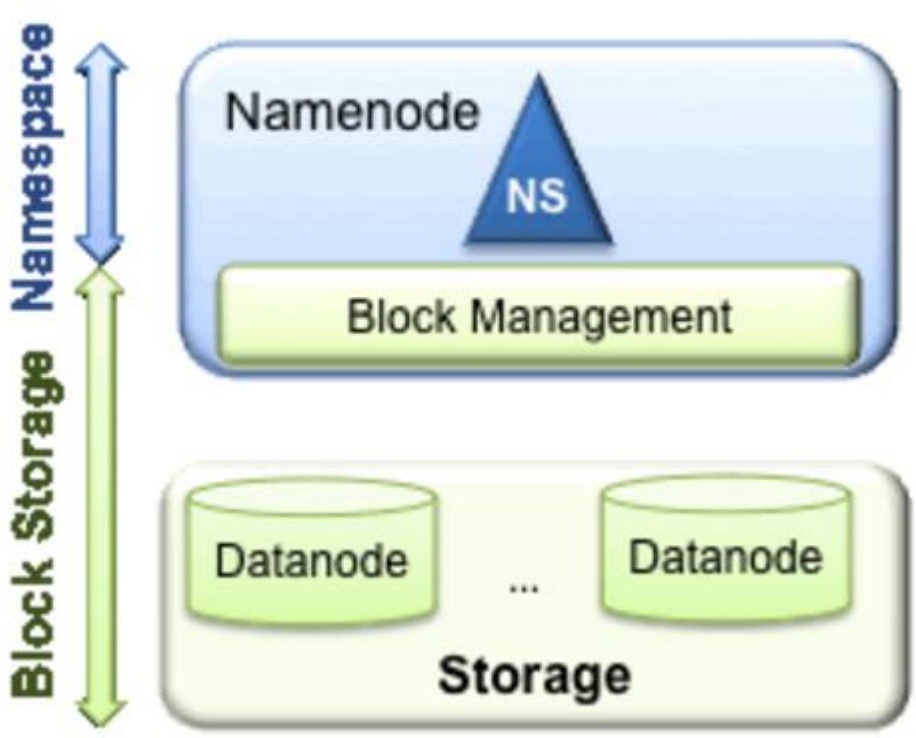


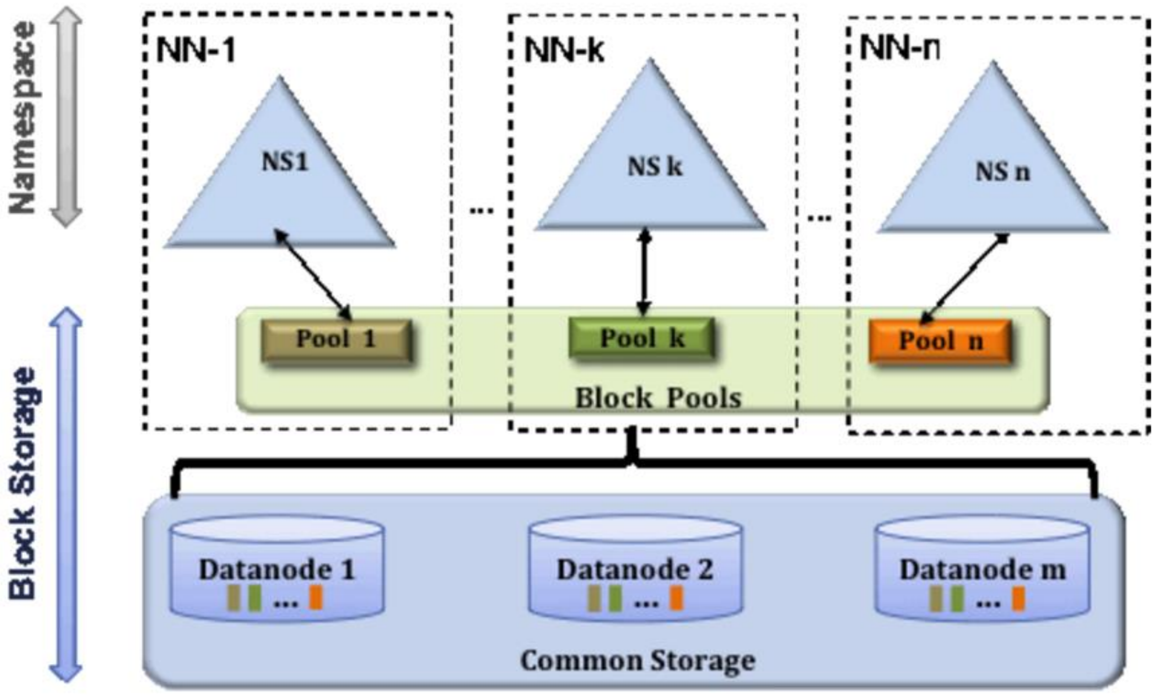
HDFS has two main layers:

- **Namespace**
  - Consists of directories, files and blocks.
  - It supports all the namespace related file system operations such as create, delete, modify and list files and directories.
- **Block Storage Service**, which has two parts:
  - Block Management (performed in the Namenode)
    - Provides Datanode cluster membership by handling registrations, and periodic heart beats.
    - Processes block reports and maintains location of blocks.
    - Supports block related operations such as create, delete, modify and get block location.
    - Manages replica placement, block replication for under replicated blocks, and deletes blocks that are over replicated.
  - Storage - is provided by Datanodes by storing blocks on the local file system and allowing read/write access.

The prior HDFS architecture allows only a single namespace for the entire cluster. In that configuration, a single Namenode manages the namespace. HDFS Federation addresses this limitation by adding support for multiple Namenodes/namespaces to HDFS.

Multiple Namenodes/Namespaces

In order to scale the name service horizontally, federation uses multiple independent Namenodes/namespaces. The Namenodes are federated; the Namenodes are independent and do not require coordination with each other. The Datanodes are used as common storage for blocks by all the Namenodes. Each Datanode registers with all the Namenodes in the cluster. Datanodes send periodic heartbeats and block reports. They also handle commands from the Namenodes.



Block Pool

A Block Pool is a set of blocks that belong to a single namespace. Datanodes store blocks for all the block pools in the cluster. Each Block Pool is managed independently. This allows a namespace to generate Block IDs for new blocks without the need for coordination with the other namespaces. A Namenode failure does not prevent the Datanode from serving other Namenodes in the cluster.

A Namespace and its block pool together are called Namespace Volume. It is a self-contained unit of management. When a Namenode/namespace is deleted, the corresponding block pool at the Datanodes is deleted. Each namespace volume is upgraded as a unit, during cluster upgrade.

ClusterID

A **ClusterID** identifier is used to identify all the nodes in the cluster. When a Namenode is formatted, this identifier is either provided or auto generated. This ID should be used for formatting the other Namenodes into the cluster.

Key Benefits

- Namespace Scalability - Federation adds namespace horizontal scaling. Large deployments or deployments using lot of small files benefit from namespace scaling by allowing more Namenodes to be added to the cluster.
- Performance - File system throughput is not limited by a single Namenode. Adding more Namenodes to the cluster scales the file system read/write throughput.
- Isolation - A single Namenode offers no isolation in a multi user environment. For example, an experimental application can overload the Namenode and slow down

production critical applications. By using multiple Namenodes, different categories of applications and users can be isolated to different namespaces.

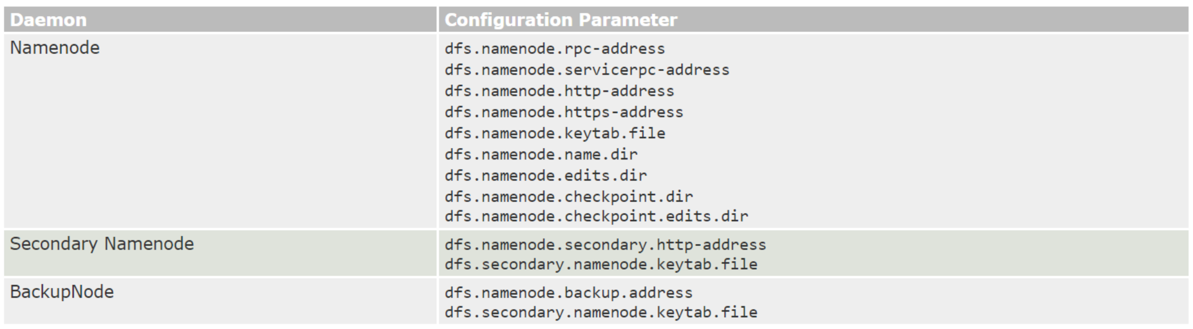
Federation Configuration

Federation configuration is **backward compatible** and allows existing single Namenode configurations to work without any change. The new configuration is designed such that all the nodes in the cluster have the same configuration without the need for deploying different configurations based on the type of the node in the cluster.

Federation adds a new `NameServiceID` abstraction. A Namenode and its corresponding secondary/backup/checkpointer nodes all belong to a NameServiceId. In order to support a single configuration file, the Namenode and secondary/backup/checkpointer configuration parameters are suffixed with the `NameServiceID`.

Step 1: Add the `dfs.nameservices` parameter to your configuration and configure it with a list of comma separated NameServiceIDs. This will be used by the Datanodes to determine the Namenodes in the cluster.

Step 2: For each Namenode and Secondary Namenode/BackupNode/Checkpointer add the following configuration parameters suffixed with the corresponding `NameServiceID` into the common configuration file:



Daemon	Configuration Parameter
Namenode	dfs.namenode.rpc-address dfs.namenode.servicerpc-address dfs.namenode.http-address dfs.namenode.https-address dfs.namenode.keytab.file dfs.namenode.name.dir dfs.namenode.edits.dir dfs.namenode.checkpoint.dir dfs.namenode.checkpoint.edits.dir
Secondary Namenode	dfs.namenode.secondary.http-address dfs.secondary.namenode.keytab.file
BackupNode	dfs.namenode.backup.address dfs.secondary.namenode.keytab.file

Here is an example configuration with two Namenodes:

```
<configuration>
  <property>
    <name>dfs.nameservices</name>
    <value>ns1,ns2</value>
  </property>
  <property>
    <name>dfs.namenode.rpc-address.ns1</name>
    <value>nn-host1:rpc-port</value>
  </property>
  <property>
    <name>dfs.namenode.http-address.ns1</name>
    <value>nn-host1:http-port</value>
  </property>
  <property>
```

```
    <name>dfs.namenode.secondaryhttp-address.ns1</name>
    <value>snn-host1:http-port</value>
  </property>
  <property>
    <name>dfs.namenode.rpc-address.ns2</name>
    <value>nn-host2:rpc-port</value>
  </property>
  <property>
    <name>dfs.namenode.http-address.ns2</name>
    <value>nn-host2:http-port</value>
  </property>
  <property>
    <name>dfs.namenode.secondaryhttp-address.ns2</name>
    <value>snn-host2:http-port</value>
  </property>

  .... Other common configuration ...
</configuration>
```