Setting up a Hadoop Distributed File System (HDFS) cluster using virtual machines (VMs) is a vital for development, testing, and sometimes even production environments where resource constraints or ease of management dictate the use of virtualization technologies. Here's a high-level overview of how we are doing the architect design such a system:

1. Understanding HDFS:
   1. HDFS: A Hadoop cluster, running HDFS, consists of one name-node and several data-nodes. The data-nodes are used for storing data but are also used for processing the stored data. To provide reliability several copies of the same data is stored on different data-nodes, in the case of one data-node going down the data will still be available. The name-node together with its data-nodes are called the Hadoop distributed file system (HDFS). When data is stored on the Hadoop file system it is divided into blocks of smaller previously given size. The smaller blocks are then distributed over the data-nodes and information about their position is given to the name-node. After the data has been stored some processing can be performed. This is done by MapReduce.
   2. MapReduce: MapReduce consist of two major parts, the job-tracker and the task-trackers. The job-trackers role is to format the data into tuples, a key and value pair, but is also supposed to work as a daemon service to keep track of the task-trackers. The task-trackers role is to process and reduce the data.

A diagram of a software process

Description automatically generated with medium confidence

1. VM Selection: We can choose different as a hypervisor such as VMware vSphere, Microsoft Hyper-V, or open-source solutions like VirtualBox or KVM. We chose here instance as a name with a specific available zone, sufficient resources (VCPUs(ssc.medium) and OS[Ubuntu 20.04 - 2023.12.07], RAM (4 GB), disk space (20 GB), and network bandwidth [as default], Key Pair(\*.pem), security group[assign a security group for protected services for authenticate IPs]) to support the number of VMs you plan to deploy.
2. VM Configuration:
   1. NameNode: Deploy one or more VMs to act as NameNodes. NameNode stores metadata about the files in the cluster.
   2. DataNode: Deploy multiple VMs to act as DataNodes. DataNodes store actual data blocks and communicate with the NameNode.
3. Network Configuration: Here we chose the default network configuration but here is a possible option to choose static, Associate floating IP and DHCP IPs.
4. Storage Configuration: Allocate storage to each VM to serve as the underlying storage for HDFS data. Here we allocate 20 GB VM size. Configure Hadoop to use this storage for data storage and replication.
5. Hadoop Installation:
   1. Install Hadoop on each VM. Follow the standard installation procedure, ensuring that you configure each node correctly according to its role (NameNode, DataNode, Secondary NameNode, etc.).
   2. Configure Hadoop's XML configuration files (core-site.xml, hdfs-site.xml, etc.) to reflect the cluster's architecture and network setup.
6. Backup and Disaster Recovery: We keep this as a default setup.
7. Security: Implement security measures such as Key \*.pem authentication, encryption, and firewall rules (setup rule for trust local machine public IPs, secure port for Associate floating IP and port trust for HDFS and Spark authentication) to protect the cluster from unauthorized access and data breaches.
8. Install Applications for analysis service: Java, Python, Jupyter Notebook, Spark, etc.
9. Testing and Optimization: Test the cluster's performance under different workloads and optimize configurations as necessary.

Spark Setup with high-level overview of Architecture:

Apache Spark is a distributed computing framework designed for fast and flexible large-scale data processing.

**Cluster Setup:**

1. Provision nodes for Spark cluster.
2. Install Java Development Kit (JDK) on all nodes.
3. Install and configure Spark on each node.
4. Configure Spark's environment variables (conf/spark-env.sh).

HDFS Setup:

1. Install and configure Hadoop HDFS.
2. Ensure HDFS accessibility from all Spark nodes.
3. Configure Spark to use HDFS (spark-defaults.conf).

Git Setup:

1. Install Git on all nodes.
2. Set up a Git repository for Spark code.
3. Clone Git repository to each node.

Develop Spark Application:

Develop Spark application code.

1. Save code within the Git repository.
2. Ensure application interacts with HDFS as needed.

Submit Spark Job:

Use Spark-submit command to submit application.

Specify path to application code and necessary arguments.

Ensure job accesses HDFS data and saves results as required.

Integration:

Integrate Git into development workflow.

1. Commit changes to Git repository and pull changes on each node.
2. Consider CI/CD tools for automation.

Monitoring and Management:

1. Monitor cluster and HDFS using appropriate tools.
2. Manage resources and configurations for optimal performance.
3. These actions collectively establish a Spark cluster connected to a Git repository, ensuring seamless development, version control, and execution of Spark applications integrated with HDFS.