Lab1: programmation avec l'API HDFS

L'objectif de ce TP est de :

- s'initier à la programmation avec l'API HDFS
- Installer l'environnement de développement VScode/git/github
- Lire/Ecrire un fichier sur HDFS

I. Démarrer le Cluster Hadoop

1. Démarrage des containers

- Démarrer le cluster hadoop créé précédemment

```
docker start hadoop-master hasdoop-slave1 hadoop-slave2
```

2. Accéder au master

Entrer dans le conteneur master pour commencer à l'utiliser

```
docker exec -it hadoop-master bash
```

3. Démarrer hadoop et yarn

lancer hadoop et yarn en utilisant un script fourni appelé start-hadoop.sh.

```
./start-hadoop.sh
```

- A la fin du démarrage, vérifier si hadoop et yarn ont démarré correctement. Pour ce faire :
  - Ressource manager web UI `http://localhost:8088`
  - HDFS web UI: `http://localhost:9870`
- utiliser la commande shell `jps` pour vérifier si les processus en relation sont en cours d'exécution

II. Programmation avec l'api HDFS

Dans cette partie, nous allons développer quelques jar pour la manipulation des fichiers avec l'api HDFS.

1. Installation de l'environnement de développement

- Outils et environnement dont on aura besoin :
  - Visual Studio Code (ou tout autre IDE de votre choix)
  - Java Version 1.8
  - Unix-like ou Unix-based Systems
- Créer un projet Maven (no archtype) dans VSCode (ajouter les extensions nécessiares ***Maven for Java*** et ***Extension Pack for Java)***
  - choisir **no archetype / groupId :** *edu.supmti.hadoop* / **artifactId** *hadoop_lab*
  - Créer un répertoire ***BigdataLabs*** où vous allez mettre votre projet **hadoop_lab**
  - ajouter les dépendances au fichier **pom.xml**

```
<properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
```

```
</properties>
<dependencies>
	<dependency>
		<groupId>org.apache.hadoop</groupId>
		<artifactId>hadoop-hdfs</artifactId>
		<version>3.2.0</version>
	</dependency>
	<dependency>
		<groupId>org.apache.hadoop</groupId>
		<artifactId>hadoop-common</artifactId>
		<version>3.2.0</version>
	</dependency>
	<dependency>
		<groupId>org.apache.hadoop</groupId>
		<artifactId>hadoop-mapreduce-client-core</artifactId>
		<version>3.2.0</version>
		</dependency>
	</dependencies>
<build>
	<plugins>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-compiler-plugin</artifactId>
			<version>3.8.1</version>
		</plugin>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-jar-plugin</artifactId>
			<version>3.2.2</version>
			<configuration>
				<archive>
					<manifest>
						<mainClass>edu.supmti.hadoop.Main</mainClass>
					</manifest>
				</archive>
				<finalName>hadoop-app</finalName>
			</configuration>
		</plugin>
	</plugins>
</build>
</project>
```

- Créer un package hdfslab sous le répertoire src/main/java/edu/*supmti*/hadoop/

2. Premier exemple

Créer une première classe `HadoopFileStatus qui permet de retourner le nom et la taille d'un fichier sur hdfs et de changer son nom`

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
```

```
public class HadoopFileStatus {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Configuration conf = new Configuration();
		FileSystem fs;
		try {
			fs = FileSystem.get(conf);
		Path filepath = new Path("/user/root/input", "purchases.txt") ;
		FileStatus infos = fs.getFileStatus(filepath);
		if(!fs.exists(filepath)){
			System.out.println("File does not exists");
			System.exist(1);
		}
		System.out.println(Long.toString(infos.getLen())+" bytes");
		System.out.println("File Name: "+filepath.getName());
		System.out.println("File Size: "+status.getLen())
		System.out.println("File owner: "+status.getOwner())
		System.out.println("File permission: "+status.getPermission())
		System.out.println("File Replication: "+status.getReplication());
		System.out.println("File Block Size: "+status.getBlockSize());
		BlockLocation[] blockLocations = fs.getFileBlockLocations(status, 0,
		status.getLen());
		for(BlockLocation blockLocation : blockLocations) {
		    String[] hosts = blockLocation.getHosts();
		    System.out.println("Block offset: " + blockLocation.getOffset());
		    System.out.println("Block length: " + blockLocation.getLength());
		    System.out.print("Block hosts: ");
		    for (String host : hosts) {
		        System.out.print(host + " ");
		    }
		    System.out.println();
		}

		fs.rename(filepath, new Path("/user/root/input", "achats.txt") ;

		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```

- à travers le fichier pom.xml
  - nommer le jar à créer par HadoopFileStatus.jar
  - N'oublier pas de spécifier la classe du main (HadoopFileStatus)
- Copier le jar créé vers le dossier de partage /hadoop_project
- sur l'invité de commande shell de votre container lancer la commande

hadoop jar /shared_volume/HadoopFileStatus.jar

- modifier la classe HadoopFileStatus (le jar évidemment) pour qu'elle puisse lire les paramètres *chemin_fichier nom_fichier nouveau_nom_fichier* lors de l'exécution

Exemple

hadoop jar /shared_volume/HadoopFileStatus.jar /user/root/input purchases.txt achats.txt

3. Lire un fichier sur HDFS

la deuxième classe à créer une classe java (ReadHDFS) qui permet de lire un fichier sur HDFS

Créer une classe qui permet de lire les informations se trouvant dans un fichier sur HDFS

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;

public class ReadHDFS {
		public static void main(String[] args) throws IOException{

		Configuration conf = new Configuration();
		FileSystem fs = FileSystem.get(conf);

		Path nomcomplet = new Path("/user/root/purchases.txt");
		FSDataInputStream inStream = fs.open(nomcomplet);
		InputStreamReader isr = new InputStreamReader(inStream);
		BufferedReader br = new BufferedReader(isr);
		String line= null;
		while((line = br.readLine())!=null) {
			System.out.println(line);
		}
		System.out.println(line);
		inStream.close();
		fs.close();
	}

}
```

- Créer un fichier jar que vous allez nommer ***ReadHDFS.jar***
- Copier le jar créé vers le dossier de partage /hadoop_project
- sur l'invité de commande shell de votre container lancer la commande

hadoop jar /shared_volume/ReadHDFS.jar

- Modifier la classe pour qu'elle puisse lire tout le fichier
- Modifier la classe pour qu'on puisse passer le nom du fichier à lire en paramètre

Exemple

hadoop jar /shared_volume/ReadHDFS.jar ./purchases.txt

4. Ecrire un fichier sur HDFS

la troisième classe à créer est une classe java (WriteHDFS) qui permet de créer un fichier sur HDFS

```
import java.io.IOException;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;

public class HDFSWrite {
	public static void main(String[] args) throws IOException {

		Configuration conf = new Configuration();
```

```
            FileSystem fs = FileSystem.get(conf);

            Path nomcomplet = new Path(args[0]);
            if (! fs.exists(nomcomplet)) {
      FSDataOutputStream outStream = fs.create(nomcomplet);
                  outStream.writeUTF("Bonjour tout le monde !");
                  outStream.writeUTF(args[1]);
                  outStream.close();
            }
            fs.close();
      }
}
```

- Créer un fichier jar que vous allez nommer ***WriteHDFS.jar***
- Copier le jar créé vers le dossier de partage /hadoop_project
- sur l'invité de commande shell de votre container lancer la commande

```
hadoop jar /shared_volume/WriteHDFS.jar ./input/bonjour.txt
```

III. Initialiser Git et Github

- Sur la ligne de commande, accéder à votre répertoire ***BigdataLabs.***
- **Vérifier l'installation de Git**

  ```
  git version
  ```
- **Configurer votre compte github**

  ```
  git config --global user.name "votre_user_name"
  git config --global user.email "votre_user_email"
  ```
- vérifier la configuration

  ```
  git config --list
  ```
- **Initialiser le dépot git de votre projet**

  ```
  git init
  ```
- **ajouter le dépôt distant github**

```
git remote add origin https://github.com/username/BIGDATA_ENGINEERING_LABS.git
```

- Premier commit et push

  ```
  git add .
  git commit -m "Initialisation du dépôt Big Data Labs"
  git branch -M main
  git push -u origin main
  ```

Vérifier sur github

```
A chaque nouveau Lab :
git add lab3
git commit -m "Ajout du lab3 : …"
```

git push

- Sortir de bash de hadoop-master **`exit`**
- Arrêter les trois conteneurs

```
docker stop hadoop-master hadoop-slave1 hadoop-slave2
```