LIGADATA

Fatafat 1.0 Setup and Running Guide

Overview

This guide will help you get the LigaDATA Fatafat engine (Online Learning Engine Platform) and Metadata API installed and ready to use.

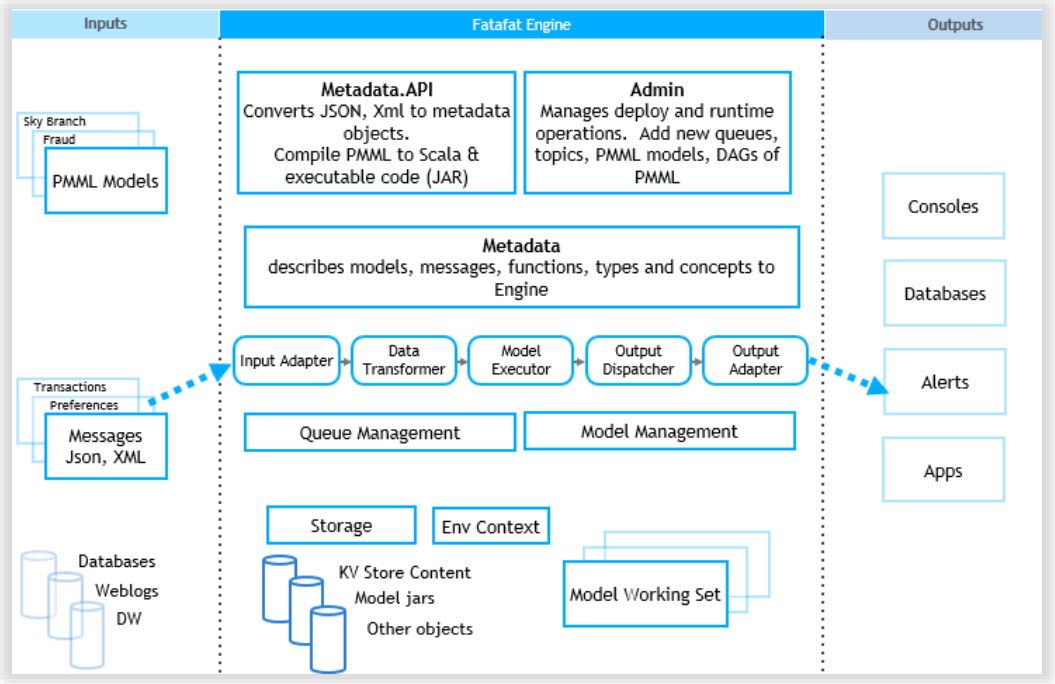
High Level Process Steps in this Guide:

Step 1. Installing Fatafat
1. Install Fatafat Package from Source OR
2. Install Fatafat Package from Binaries

Step 2. Deploying and Running Fatafat
1. Load Core Fatafat Metadata
2. Load Application Metadata
3. Load the input data and the Rule set
4. Create Queues and Push data to Kafka Queues
5. Run the engine
6. Push Sample Data
7. View Results

Once you have started the Engine, it will process the input data against the rules defined in the Metadata and produce "alerts" (output that matches your application's rules).

The Fatafat engine processes incoming streaming data, transforms it into messages or containers, and processes it according to the ruleset you have supplied in your models. It then produces decision or alert data that can be acted upon.



The metadata objects include Types, Functions, Concepts, Message Definitions, and Model Definitions. The Metadata API defines create and read operations for metadata objects supported by this system. All functions take String values as input in XML or JSON Format and return a JSON string of ApiResult object.

Prerequisites:

- CentOS/RedHat/OS X (If using windows, set up a virtual machine running a Linux distribution)
- Approximately 400 MB for installation (3 GB if building from source)
    - Access to: https://github.com/ligaDATA/Fatafat OR the Fatafat install package
- Install the JDK 1.7.1 or greater (which can be downloaded here.)
- Install Scala v2.10.4: http://www.scala-lang.org/download/2.10.4.html
- Install sbt: http://www.scala-sbt.org/download.html
- Download and install Zookeeper: http://zookeeper.apache.org/releases.html#download
- Download and install Kafka 2.10-0.8.1.1: http://kafka.apache.org/
- The definition of the data models and rule sets used in your business that need to be created as models.

Assumptions:

At least one instance of each of the following is running:

- Zookeeper Service
- Kafka
- Cassandra OR HBase

NOTE: Cassandra and HBase are optional - you will need some input/output to the engine.

Linux users: It is suggested that Linux users run the services/scripts under screens (http://www.howtoforge.com/linux_screen), so that if the connection is lost, they will still be running. This will likely become a service in a later release.)

Step 1: Installing Fatafat

Note: Install from source (option 1.) or binary (option 2.)

Option 1: Install Fatafat Package from Source:

1. Download the project source.
    a. From GitHub, clone: https://github.com/ligaDATA/Fatafat.git

2. Run install script:

    a. cd <localCloneDirectory>/Fatafat/trunk/SampleApplication/EasyInstall

    b. bash easyInstallFatafat.sh <InstallDirectoryPath> <trunk sources Path> <ivyPath for dependency jars> <KafkaLocation>

    Example: bash easyInstallFatafat.sh /tmp/InstallDirectory ~/repos/Fatafat/trunk ~/.ivy2 ~/Kafka/kafka_2.10-0.8.1.1/kafka_2.10-0.8.1.1

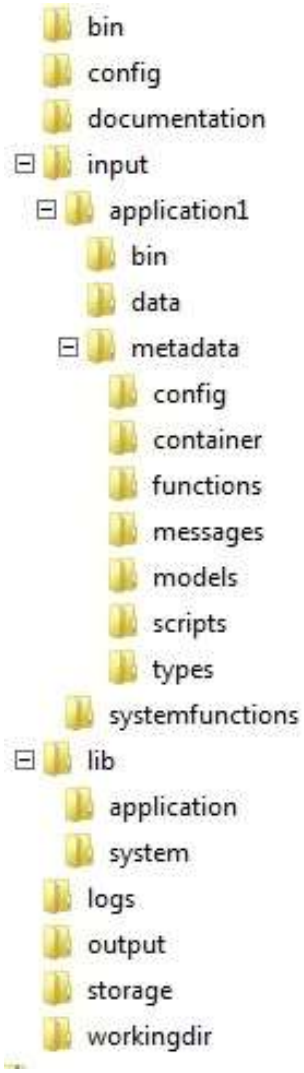
Option 2: Install Fatafat Package from Binaries:

1. For this example, we will assume the binaries are on a thumb drive. Copy the Fatafat directory from the thumb drive to a local directory (Referred to hereafter as <InstallDirectoryPath>).

2. Run the command below to set the needed paths to generate the scripts from the current directory:

    a. bash <InstallDirectoryPath>/bin/SetPaths.sh

After installation, the following directory structure will be created:



Step 2: Deploying and Running Fatafat

Note: You will need to ensure that Zookeeper and Kafka are running before running Fatafat.

#1 - Load Core Fatafat Metadata

1. Execute the following script to load cluster configuration into metadata:
    a. bash <InstallDirectoryPath>/bin/ClusterMetadata.sh
    b. Add Cluster Configuration (Option 33)

#2 - Load Application Metadata

1. Execute the following script to load the application definitions into metadata:
    a. bash <InstallDirectoryPath>/input/application1/bin/ApplicationMetadata.sh
    b. Add Containers, Messages, Models

    NOTE: Each application will have an individual script for loading metadata

#3 - Load the input data and the Rule set

1. Execute the following command to load the patient data (containers) and the rule set:
    a. bash <InstallDirectoryPath>/input/application1/bin/InitKvStores.sh

#4 - Create Queues and Push data to Kafka Queues

1. Execute the following command to create queues:
    a. bash <InstallDirectoryPath>/bin/CreateQueues.sh

#5 - Run the engine:

1. Execute the following command to run the Engine:
    a. bash <InstallDirectoryPath>/bin/StartEngine.sh
    b. Typing “quit” or pressing ctrl-c will stop the engine

#6 - Push Sample Data:

1. Execute the following command to push input data to queues:
    a. bash <InstallDirectoryPath>/input/application1/bin/PushSampleDatazToKafka.sh
    b. The input file being used can be found at
    <InstallDirectoryPath>/input/application1/data/copd_demo.csv.gz

#7 - View Results

1. Execute the following script to see status:
    a. bash <InstallDirectoryPath>/bin/WatchStatusQueue.sh
2. Execute the following script to see the output results:
    a. bash <InstallDirectoryPath>/bin/WatchOutputQueue.sh

THE REMAINING CONTENT STILL NEEDS REVIEW.

Multi-Node Account Setup Instructions

Each Fatafat cluster installed on a network is ideally run by a distinct user account. These instructions provide the basics for setting up such a user so that it can easily manage the cluster administration.

A user name and group should be selected to be the owner and user that runs the cluster. It can be any useful and meaningful name not already in use. In the examples below, the name *fatafat* is used for both the account and group name. Two nodes named *ls19.dc.npario.com* and *ls20.dc.npario.com* are used to illustrate the setup on two machines.

It is assumed that you can get to all machines from another account and have sudoer privileges.

Admin setup items – verify or do the following:

1. If the *fatafat* group does not exist on the prospective cluster nodes, add it to each node with the following command:

   sudo groupadd fatafat

2. Establish a user account on each prospective cluster node making the new user use the fatafat group as its default login group. Be sure to establish a home directory for the user. Example:

   sudo useradd --gid fatafat --create-home fatafat

3. Set the password for the fatafat cluster login to something using sudo

   sudo passwd fatafat

4. If the fatafat release is staged from source on one of the machines in the new cluster or cluster's network, make sure that the staging user (or the developer) account is also a member of the *fatafat* group. This can be done with this command:

   sudo usermod -G fatafat releaseAdmin

where the user account is releaseAdmin. This really is only needed on this one machine, not every node in the cluster, although THAT would not necessarily be a bad idea. This is security policy dependent – ensure that you have those policies correct setup.

5. Passphrase-less access to/from all participating nodes in the *fatafat* cluster required for proper operation. Each node could have its own public and private key if your security policy requires it. However, it is far easier to login to one of the nodes as the fatafat user and set up the .ssh keys once, then propagate the entire directory to the other nodes in the cluster. Assuming the accounts are new, there is really no harm in replacing the entire .ssh directory. If you have existing keys there, however, don't replace. Instead append.

To append to the .ssh keys

1. generate a public private key

   ssh-keygen

It will prompt for a key type. Take the default (RSA 2048) by hitting return. Then it will ask for a passphrase; hit return for "passphraseless" behavior. It will prompt again to type the same passphrase; once again hit return. This will generate a public and private key for the current user account.

2. Install the public key on the current machine.

To install the new public key in the current directory, you can cat the public key into a file called .ssh/authorized_keys or perhaps a little easier use this approach, as it will properly set the access permissions for the file it will create. If the current machine was named ls19.dc.npario.com and the fatafat account being setup is fatafat, do the following:

   ssh-copy-id fatafat@ls19.dc.npario.com

You will have to type the password this one time to establish the authorized key file, but after that you should be able to access without giving a passphrase.

3. Verify that the $HOME/.ssh permissions is strict. There should be no permitted access to the .ssh directory except by the owner.

4. As the fatafat account, Zip/Bzip/Gzip the .ssh folder in its entirety and leave it in $HOME. For example:

      tar cvjf fatafatNodeSSH.bz2 .ssh

This creates an archive that will be copied around to the other nodes that are to participate in the cluster.

5. As the fatafat account, copy the archive to every node in the cluster and unzip it. For example, if ls20.dc.npario.com were another node in the cluster to be setup, do the following:

   scp fatafatNodeSSH.bz2 ls20.dc.npario.com:

This will leave the tarball in the fatafat user's ls20 home folder. Do this same action for all the nodes to particpate or might participate in the cluster being set up.

6. ssh to each node as the fatafat user and decompress the file copied in the previous step. For example,

   tar xvjf fatafatNodeSSH.bz2

This will update the .ssh folder with the authorized_keys as well as the public and private keys that were generated in part a) above. To verify that it worked, try to ssh to the original machine where the keys were generated. For example,

```
ssh ls19.dc.npario.com
```

If successful, then you should login immediately (not requiring a passphrase from the console).

7. Check further. A good check would be to login to one of the cluster nodes to particpate in the fatafat cluster being formed AS the fatafat account and try to log in to every other machine in the cluster. These logins should succeed without a passphrase being required.

Should there be problems, there are many resources on the web that describe the setup of a passphrase-less account. In particular, the access permissions for the account and its .ssh need to be strict (no group or global permissions should be allowed). Ssh will complain if these requirements are not met.

8. Finally, remove the tarball that has been passed around to all of the nodes from each fatafat user account on each node.

More information. The instructions above are indicative, not authoritative, of what must be done to establish password-less access. If you experience problems, consult the documentation for your Linux enlistment for special requirements that your environment may require.

Installing Fatafat to a cluster from your Git Repository

Fatafat is typically installed from the trunk directory of an "install" or "release" machine, building the Fatafat installation directory on the local system, creating a tarball of that directory, sending it to the nodes that are part of the Fatafat cluster, untar and decompress the tarball there and move the resulting directory into place.

Several scripts are part of the repository that are used to accomplish this. These scripts must be moved onto your PATH and given "execute" permissions. They are:

1. trunk/SampleApplication/Medical/bin/clusterInstallFatafat.sh
2. trunk/SampleApplication/Medical/bin/installFatafat_Medical.sh
3. trunk/SampleApplication/Medical/bin/nodeInfoExtract.sh
4. trunk/Pmml/Scripts/sbtProjDependencies/src/main/scala/sbtProjDependencies.scala

The ***clusterInstallFatafat.sh*** *script* controls the installation, using the cluster configuration information either found in the supplied NodeConfigPath argument or from the metadata store described in the MetadataAPIConfig file. In the first case, the metadata store may have no configuration and the supplied NodeConfigPath will provide it. In the second case, where no NodeConfigPath is supplied, the metadata store is expected to have one. Should a valid configuration not be found, the script will issue a RuntimeException with a useful message describing the particular issue. Here are examples of script invocation

```
clusterInstallFatafat.sh
--MetadataAPIConfig SampleApplication/Medical/Configs/MetadataAPIConfig.properties
```

clusterInstallFatafat.sh
--NodeConfigPath SampleApplication/Medical/Configs/Engine2BoxConfigV1.json

The ***installFatafat_Medical.sh*** will build the Fatafat install directory. The *nodeInfoExtract*.sh is used to determine which cluster nodes are to receive the build. The s*btProjDependencies* is a useful tool used by a number of scripts that are part of Fatafat. In this case, it is used to build the classpath for the *nodeInfoExtract* `application.

Invoke the installer

Use the following command by invoking it from the trunk directory of the local Git repository which is to be installed. Run it from the account that is to be used to run/manage the Fatafat cluster and refer to directories that are write-able by that account.

```
Cluster InstallFatafat.sh
  --MetadataAPIConfig  SampleApplication/Medical/Configs/MetadataAPIConfig.properties
  --NodeConfigPath SampleApplication/Medical/Configs/Engine2BoxConfigV1.json
```

Currently two configuration files are required, one for the MetadataAPI and the other that describes at least the cluster for the FatafatManager. The cluster description is the key information as to what sort of distribution and what machines and nodes are involved in the installation. Note that multiple nodes can if desired be installed on the same physical computer.

For more information about what these configuration files contain, see <some reference to other documentation describing the configuration>.

Start the Cluster

The Fatafat cluster start script - **startFatafatCluster.sh**, like the install script, can take one or two arguments. If the NodeConfigPath is present, its cluster config will be added to the metadata store specified with the MetadataAPIConfig argument. If not present, the metadata store is expected to have a configuration. Should a valid configuration not be found, the script will issue a RuntimeException with a useful message describing the particular issue.

startFatafatCluster.sh
--MetadataAPIConfig <metadataAPICfgPath> [--NodeConfigPath <fatafatCfgPath>]

Appendix

Please use this example of the EngineConf.Json file as a guide to the values you need to change in the system you are using to run the Fatafat engine. Gray shading indicates those values.


```
{
  "Clusters": [
    {
      "ClusterId": "ligadata1",
      "Config": {
        "DataStore": "{\"StoreType\": \"cassandra\",\"SchemaName\": \"testdata\",\"Location\":
\"localhost\"}",
        "StatusInfo": "{\"StoreType\": \"cassandra\",\"SchemaName\":
\"statusinfo\",\"Location\": \"localhost\"}",
        "ZooKeeperInfo": "{\"ZooKeeperNodeBasePath\":
\"/ligadata\",\"ZooKeeperConnectString\": \"localhost:2181\",\"ZooKeeperSessionTimeoutMs\":
\"250\",\"ZooKeeperConnectionTimeoutMs\": \"30000\"}",
        "EnvironmentContext":
"{\"classname\":\"com.ligadata.SimpleEnvContextImpl.SimpleEnvContextImpl$\",\"jarname\":\"simp
le
envcontextimpl_2.10-1.0.jar\",\"dependencyjars\":[\"log4j-1.2.17.jar\",\"onlepbase_2.10-
1.0.jar\",\"metadata_2.10-1.0.jar\",\"serialize_2.10-1.0.jar\",\"storage_2.10-
0.0.0.2.jar\",\"asm-3.1.jar\",\"metrics-core-3.0.2.jar\",\"cassandra-driver-core-
2.0.2.jar\",\"kryo-2.21.jar\",\"minlog-1.2.jar\",\"reflectasm-1.07-shaded.jar\",\"jackson-
annotations-2.3.0.jar\",\"jackson-core-2.3.1.jar\",\"jackson-databind-2.3.1.jar\",\"findbugs-
annotations-1.3.9-1.jar\",\"jsr305-1.3.9.jar\",\"google-collections-1.0.jar\",\"guava-
16.0.1.jar\",\"protobuf-java-2.5.0.jar\",\"protobuf-java-2.6.0.jar\",\"java-xmlbuilder-
0.4.jar\",\"jsch-0.1.42.jar\",\"compress-lzf-0.9.1.jar\",\"junit-interface-0.11-
RC1.jar\",\"je-4.0.92.jar\",\"jersey-core-1.9.jar\",\"jersey-json-1.9.jar\",\"jersey-server-
1.9.jar\",\"jaxb-impl-2.2.3-1.jar\",\"paranamer-2.3.jar\",\"paranamer-2.6.jar\",\"chill-java-
0.3.6.jar\",\"chill_2.10-0.3.6.jar\",\"commons-beanutils-core-1.8.0.jar\",\"commons-beanutils-
1.7.0.jar\",\"commons-cli-1.2.jar\",\"commons-codec-1.9.jar\",\"commons-collections-
3.2.1.jar\",\"commons-configuration-1.6.jar\",\"commons-dbcp-1.2.2.jar\",\"commons-digester-
1.8.jar\",\"commons-el-1.0.jar\",\"commons-httpclient-3.1.jar\",\"commons-io-
2.4.jar\",\"commons-lang-2.6.jar\",\"commons-logging-1.1.3.jar\",\"commons-net-
3.1.jar\",\"commons-pool-1.5.2.jar\",\"netty-3.9.0.Final.jar\",\"activation-1.1.jar\",\"jsp-
api-2.1.jar\",\"servlet-api-2.5.jar\",\"jaxb-api-2.2.2.jar\",\"stax-api-1.0-2.jar\",\"jline-
0.9.94.jar\",\"joda-time-2.3.jar\",\"junit-4.11.jar\",\"log4j-1.2.17.jar\",\"jets3t-
0.9.0.jar\",\"jna-3.2.7.jar\",\"avro-1.7.4.jar\",\"commons-compress-1.4.1.jar\",\"commons-
math3-3.1.1.jar\",\"hadoop-annotations-2.4.1.jar\",\"hadoop-auth-2.4.1.jar\",\"hadoop-common-
2.4.1.jar\",\"hbase-client-0.98.4-hadoop2.jar\",\"hbase-common-0.98.4-hadoop2.jar\",\"hbase-
protocol-0.98.4-hadoop2.jar\",\"httpclient-4.2.5.jar\",\"httpcore-4.2.4.jar\",\"zookeeper-
3.4.6.jar\",\"htrace-core-2.04.jar\",\"jackson-core-asl-1.8.8.jar\",\"jackson-jaxrs-
1.8.3.jar\",\"jackson-mapper-asl-1.8.8.jar\",\"jackson-xc-1.8.3.jar\",\"jettison-
1.1.jar\",\"hamcrest-core-1.3.jar\",\"jdom-1.1.jar\",\"joda-convert-1.6.jar\",\"json4s-
ast_2.10-3.2.9.jar\",\"json4s-core_2.10-3.2.9.jar\",\"json4s-jackson_2.10-
3.2.9.jar\",\"json4s-native_2.10-3.2.9.jar\",\"mapdb-1.0.6.jar\",\"jetty-util-
6.1.26.jar\",\"jetty-6.1.26.jar\",\"objenesis-1.2.jar\",\"asm-commons-4.0.jar\",\"asm-tree-
4.0.jar\",\"asm-4.0.jar\",\"scalap-2.10.0.jar\",\"test-interface-
1.0.jar\",\"quasiquotes_2.10.4-2.0.0-M6.jar\",\"scalatest_2.10-2.2.0.jar\",\"slf4j-api-
1.7.7.jar\",\"slf4j-log4j12-1.7.5.jar\",\"xz-1.0.jar\",\"snappy-java-1.0.4.1.jar\",\"jasper-
compiler-5.5.23.jar\",\"jasper-runtime-5.5.23.jar\",\"voldemort-0.96.jar\",\"xmlenc-
0.52.jar\"]}"
      },
      "Nodes": [
        {
          "NodeId": "1",
          "NodePort": 6541,
          "NodeIpAddr": "localhost",
          "JarPaths": [
            "/tmp/OnLEPInstall"
          ],
```

```
          "Scala_home": "/home/xxxxxx/scala-2.10.4",
          "Java_home": "/home/xxxxxx/jdk1.8.0_05",
          "Classpath": ".:/tmp/OnLEPInstall/metadata_2.10-
1.0.jar:/tmp/OnLEPInstall/basefunctions_2.10-0.1.0.jar:/tmp/OnLEPInstall/messagedef_2.10-
1.0.jar:/tmp/OnLEPInstall/methodextractor_2.10-1.0.jar:/tmp/OnLEPInstall/pmmlcompiler_2.10-
1.0.jar:/tmp/OnLEPInstall/onlepbase_2.10-1.0.jar:/tmp/OnLEPInstall/bootstrap_2.10-
1.0.jar:/tmp/OnLEPInstall/joda-time-2.3.jar:/tmp/OnLEPInstall/joda-convert-
1.6.jar:/tmp/OnLEPInstall/basetypes_2.10-0.1.0.jar:/tmp/OnLEPInstall/pmmludfs_2.10-
1.0.jar:/tmp/OnLEPInstall/pmmlruntime_2.10-1.0.jar:/tmp/OnLEPInstall/json4s-native_2.10-
3.2.9.jar:/tmp/OnLEPInstall/json4s-core_2.10-3.2.9.jar:/tmp/OnLEPInstall/json4s-ast_2.10-
3.2.9.jar:/tmp/OnLEPInstall/jackson-databind-2.3.1.jar:/tmp/OnLEPInstall/jackson-annotations-
2.3.0.jar:/tmp/OnLEPInstall/json4s-jackson_2.10-3.2.9.jar:/tmp/OnLEPInstall/jackson-core-
2.3.1.jar:/tmp/OnLEPInstall/log4j-1.2.17.jar"
        },
        {
          "NodeId": "2",
          "NodePort": 6542,
          "NodeIpAddr": "localhost",
          "JarPaths": [
            "/tmp/OnLEPInstall"
          ],
          "Scala_home": "/home/xxxxxx/scala-2.10.4",
          "Java_home": "/home/xxxxxx/jdk1.8.0_05",
          "Classpath": ".:/tmp/OnLEPInstall/metadata_2.10-
1.0.jar:/tmp/OnLEPInstall/basefunctions_2.10-0.1.0.jar:/tmp/OnLEPInstall/messagedef_2.10-
1.0.jar:/tmp/OnLEPInstall/methodextractor_2.10-1.0.jar:/tmp/OnLEPInstall/pmmlcompiler_2.10-
1.0.jar:/tmp/OnLEPInstall/onlepbase_2.10-1.0.jar:/tmp/OnLEPInstall/bootstrap_2.10-
1.0.jar:/tmp/OnLEPInstall/joda-time-2.3.jar:/tmp/OnLEPInstall/joda-convert-
1.6.jar:/tmp/OnLEPInstall/basetypes_2.10-0.1.0.jar:/tmp/OnLEPInstall/pmmludfs_2.10-
1.0.jar:/tmp/OnLEPInstall/pmmlruntime_2.10-1.0.jar:/tmp/OnLEPInstall/json4s-native_2.10-
3.2.9.jar:/tmp/OnLEPInstall/json4s-core_2.10-3.2.9.jar:/tmp/OnLEPInstall/json4s-ast_2.10-
3.2.9.jar:/tmp/OnLEPInstall/jackson-databind-2.3.1.jar:/tmp/OnLEPInstall/jackson-annotations-
2.3.0.jar:/tmp/OnLEPInstall/json4s-jackson_2.10-3.2.9.jar:/tmp/OnLEPInstall/jackson-core-
2.3.1.jar:/tmp/OnLEPInstall/log4j-1.2.17.jar"
        },
        {
          "NodeId": "4",
          "NodePort": 6544,
          "NodeIpAddr": "localhost",
          "JarPaths": [
            "/tmp/OnLEPInstall"
          ],
          "Scala_home": "/home/xxxxxx/scala-2.10.4",
          "Java_home": "/home/xxxxxx/jdk1.8.0_05",
          "Classpath": ".:/tmp/OnLEPInstall/metadata_2.10-
1.0.jar:/tmp/OnLEPInstall/basefunctions_2.10-0.1.0.jar:/tmp/OnLEPInstall/messagedef_2.10-
1.0.jar:/tmp/OnLEPInstall/methodextractor_2.10-1.0.jar:/tmp/OnLEPInstall/pmmlcompiler_2.10-
1.0.jar:/tmp/OnLEPInstall/onlepbase_2.10-1.0.jar:/tmp/OnLEPInstall/bootstrap_2.10-
1.0.jar:/tmp/OnLEPInstall/joda-time-2.3.jar:/tmp/OnLEPInstall/joda-convert-
1.6.jar:/tmp/OnLEPInstall/basetypes_2.10-0.1.0.jar:/tmp/OnLEPInstall/pmmludfs_2.10-
1.0.jar:/tmp/OnLEPInstall/pmmlruntime_2.10-1.0.jar:/tmp/OnLEPInstall/json4s-native_2.10-
3.2.9.jar:/tmp/OnLEPInstall/json4s-core_2.10-3.2.9.jar:/tmp/OnLEPInstall/json4s-ast_2.10-
3.2.9.jar:/tmp/OnLEPInstall/jackson-databind-2.3.1.jar:/tmp/OnLEPInstall/jackson-annotations-
2.3.0.jar:/tmp/OnLEPInstall/json4s-jackson_2.10-3.2.9.jar:/tmp/OnLEPInstall/jackson-core-
2.3.1.jar:/tmp/OnLEPInstall/log4j-1.2.17.jar"
        }
      ]
    }
  ],
  "Adapters": [
    {
      "Name": "TestIn_1",
      "TypeString": "Input",
      "DataFormat": "CSV",
```

```
    "ClassName": "com.ligadata.InputAdapters.KafkaSimpleConsumer$",
    "JarName": "kafkasimpleinputoutputadapters_2.10-1.0.jar",
    "DependencyJars": [
      "jopt-simple-3.2.jar",
      "kafka_2.10-0.8.1.1.jar",
      "log4j-1.2.15.jar",
      "metrics-core-2.2.0.jar",
      "slf4j-api-1.7.2.jar",
      "snappy-java-1.0.5.jar",
      "zkclient-0.3.jar",
      "zookeeper-3.3.4.jar",
      "onlepbase_2.10-1.0.jar"
    ],
    "AdapterSpecificCfg": "{\"HostList\": \"localhost:9092\",\"TopicName\": \"testin_1\" }"
  },
  {
    "Name": "TestOut_In_1",
    "TypeString": "Validate",
    "DataFormat": "JSON",
    "ClassName": "com.ligadata.InputAdapters.KafkaSimpleConsumer$",
    "JarName": "kafkasimpleinputoutputadapters_2.10-1.0.jar",
    "DependencyJars": [
      "jopt-simple-3.2.jar",
      "kafka_2.10-0.8.1.1.jar",
      "log4j-1.2.15.jar",
      "metrics-core-2.2.0.jar",
      "slf4j-api-1.7.2.jar",
      "snappy-java-1.0.5.jar",
      "zkclient-0.3.jar",
      "zookeeper-3.3.4.jar",
      "onlepbase_2.10-1.0.jar"
    ],
    "AdapterSpecificCfg": "{\"HostList\": \"localhost:9092\",\"TopicName\": \"testout_1\" }"
  },
  {
    "Name": "TestOut_1",
    "TypeString": "Output",
    "InputAdapterToVerify": "TestOut_In_1",
    "ClassName": "com.ligadata.OutputAdapters.KafkaProducer$",
    "JarName": "kafkasimpleinputoutputadapters_2.10-1.0.jar",
    "DependencyJars": [
      "jopt-simple-3.2.jar",
      "kafka_2.10-0.8.1.1.jar",
      "log4j-1.2.15.jar",
      "metrics-core-2.2.0.jar",
      "slf4j-api-1.7.2.jar",
      "snappy-java-1.0.5.jar",
      "zkclient-0.3.jar",
      "zookeeper-3.3.4.jar",
      "onlepbase_2.10-1.0.jar"
    ],
    "AdapterSpecificCfg": "{\"HostList\": \"localhost:9092\",\"TopicName\": \"testout_1\" }"
  },
  {
    "Name": "TestStatus_1",
    "TypeString": "Status",
    "ClassName": "com.ligadata.OutputAdapters.KafkaProducer$",
    "JarName": "kafkasimpleinputoutputadapters_2.10-1.0.jar",
    "DependencyJars": [
      "jopt-simple-3.2.jar",
      "kafka_2.10-0.8.1.1.jar",
      "log4j-1.2.15.jar",
      "metrics-core-2.2.0.jar",
```

```
        "slf4j-api-1.7.2.jar",
        "snappy-java-1.0.5.jar",
        "zkclient-0.3.jar",
        "zookeeper-3.3.4.jar",
        "onlepbase_2.10-1.0.jar"
      ],
      "AdapterSpecificCfg": "{\"HostList\": \"localhost:9092\",\"TopicName\": \"teststatus_1\"
}"
    }
  ]
}
```