DataIngestion Handover

WL

Topics

- System Architecture
- Operation
- Program Architecture

Data ingestion defined

- Data ingestion is a process by which data is moved from one or more sources to a destination where it can be stored and further analyzed. The data might be in different formats and come from various sources, including RDBMS, other types of databases, S3 buckets, CSVs, or from streams. Since the data comes from different places, it needs to be cleansed and transformed in a way that allows you to analyze it together with data from other sources. Otherwise, your data is like a bunch of puzzle pieces that don't fit together.

Data ingestion defined

- You can ingest data in real time, in batches, or in a combination of the two (this is called lambda architecture). When you ingest data in batches, data is imported at regularly scheduled intervals. This can be very useful when you have processes that run on a schedule, such as reports that run daily at a specific time. Real-time ingestion is useful when the information gleaned is very time-sensitive, such as data from a power grid that must be monitored moment-to-moment. Of course, you can also ingest data using a lambda architecture. This approach attempts to balance the benefits of batch and real-time modes by using batch processing to provide comprehensive views of batch data, while also using real-time processing to provide views of time-sensitive data.

In this project, we try to achieve …

- We try to import data from Oracle ODS database into HDFS filesystem on Hadoop cluster. The loading method of ODS tables are including truncate-insert, delete-insert, upsert, and append. The data ingest modules will use a corresponding processing method, that is Truncate-Insert and Delete-Insert, according each table's loading method, data size, network bandwidth and so on.
- After importing, there will be a interface for external users to access the imported data. The data ingest module will create responding hive/impala tables as well for this purpose.

Glossary

ATHEMASTER

- Data Ingestion Project (DI)
- Module
  - importer
  - init-tools
  - interface
  - udf (user defined function)
- Processing Method
  - truncate-insert
  - delete-insert

Glossary

- Configuration
  - config db (configuration tables in RDB)
  - config files

ATHEMASTER

System Architecture

Before we start, this is the modules we use …

- java library
  - spring-core: 5.1.5.RELEASE
  - spring-boot-starter: 2.1.3.RELEASE
  - spring-boot-starter-data-jpa: 2.1.3.RELEASE
- spark: 2.3.0.cloudera4
- sqoop: sqoop-1.4.6
- hive: hive-1.1.0
- bash

Also, there are some important things …

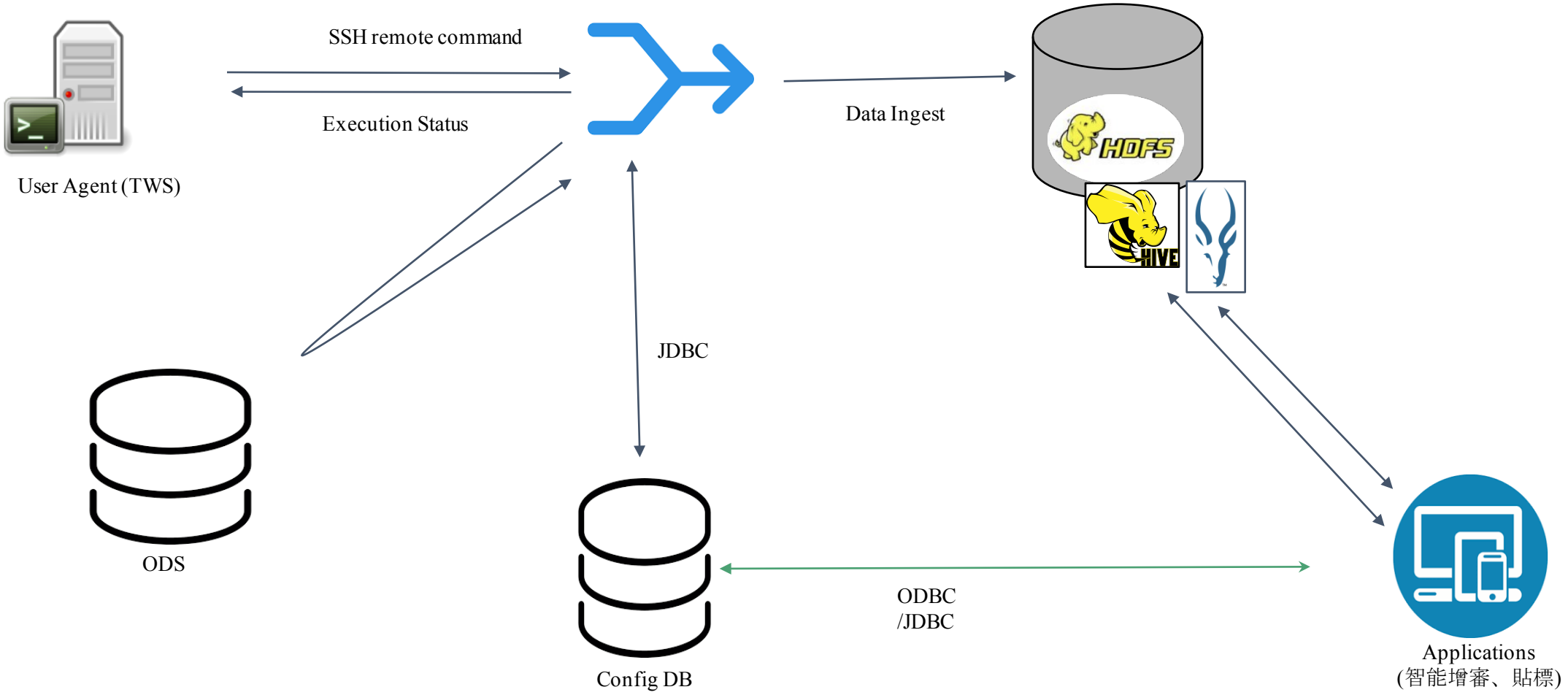
- Database accounts
  - lnk_bdp_sqoop: for importing data by using sqoop from ods to HDFS
  - bdp_etl_pipe: for recording execution status and initialing ODS tables configuration
- Configuration tables
  - IMPORTER_CONFIG — table configuration, such as processing method, data fetching/dropping range definition …
  - CAST_SETTINGS — column definition, such as type casting and masking rule
  - MASK_OPERATION — masking module definition
  - BDP_ETL_LOG — execution status log, a record for each execution
  - EXEC_STATUS — execution status log, a record for each table

Some important things (2)

- Configuration
  - application.yml — login password and environment settings …
  - delete-insert.config — delete insert (loading method) settings
  - masking-rule.csv — data masking settings

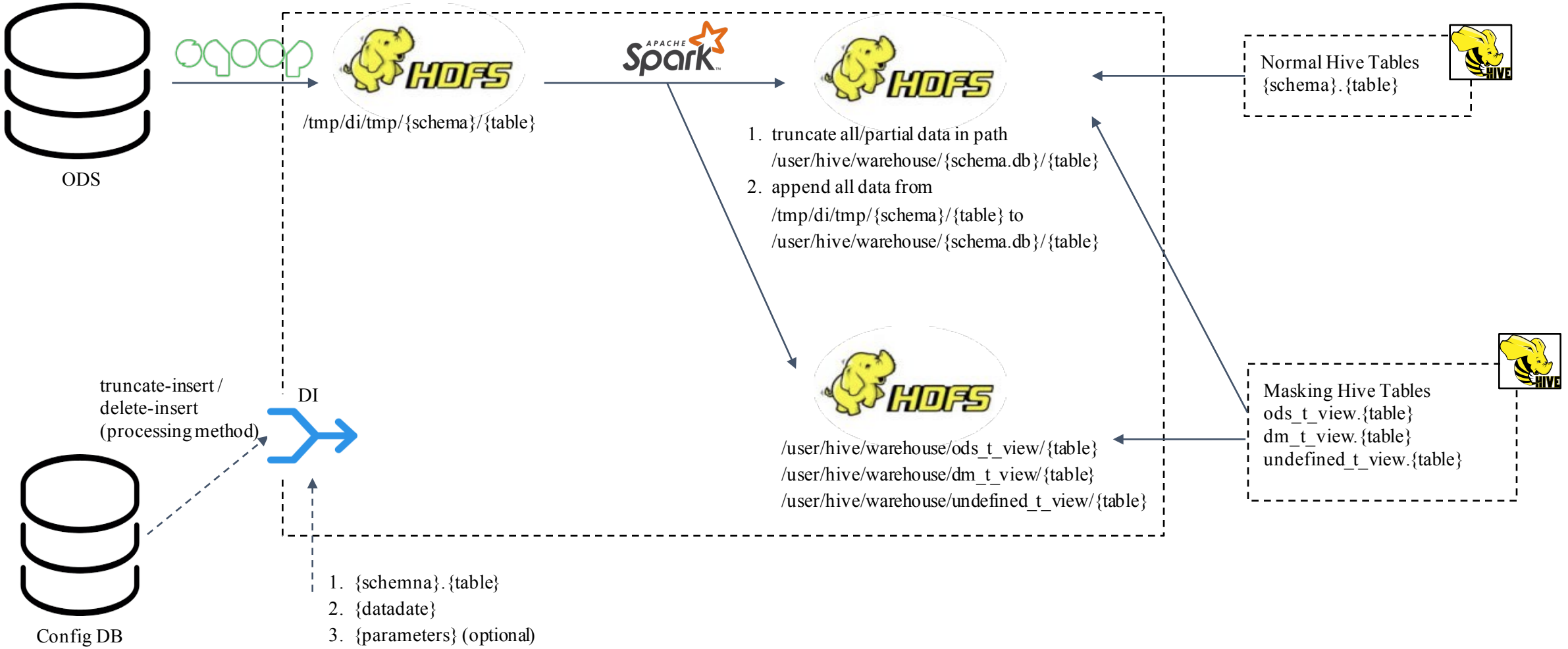
Data Flow Diagram

ATHEMASTER



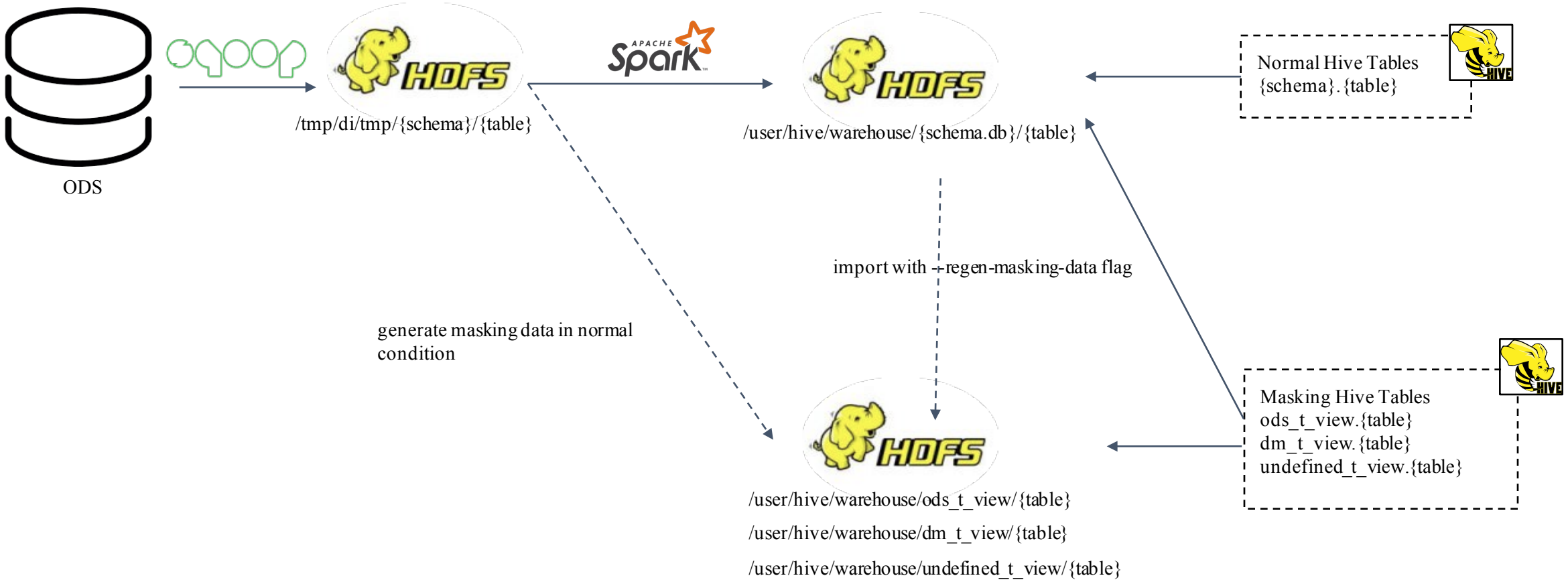
Truncate/Delete-Insert Data Flow Diagram

ATHEMASTER



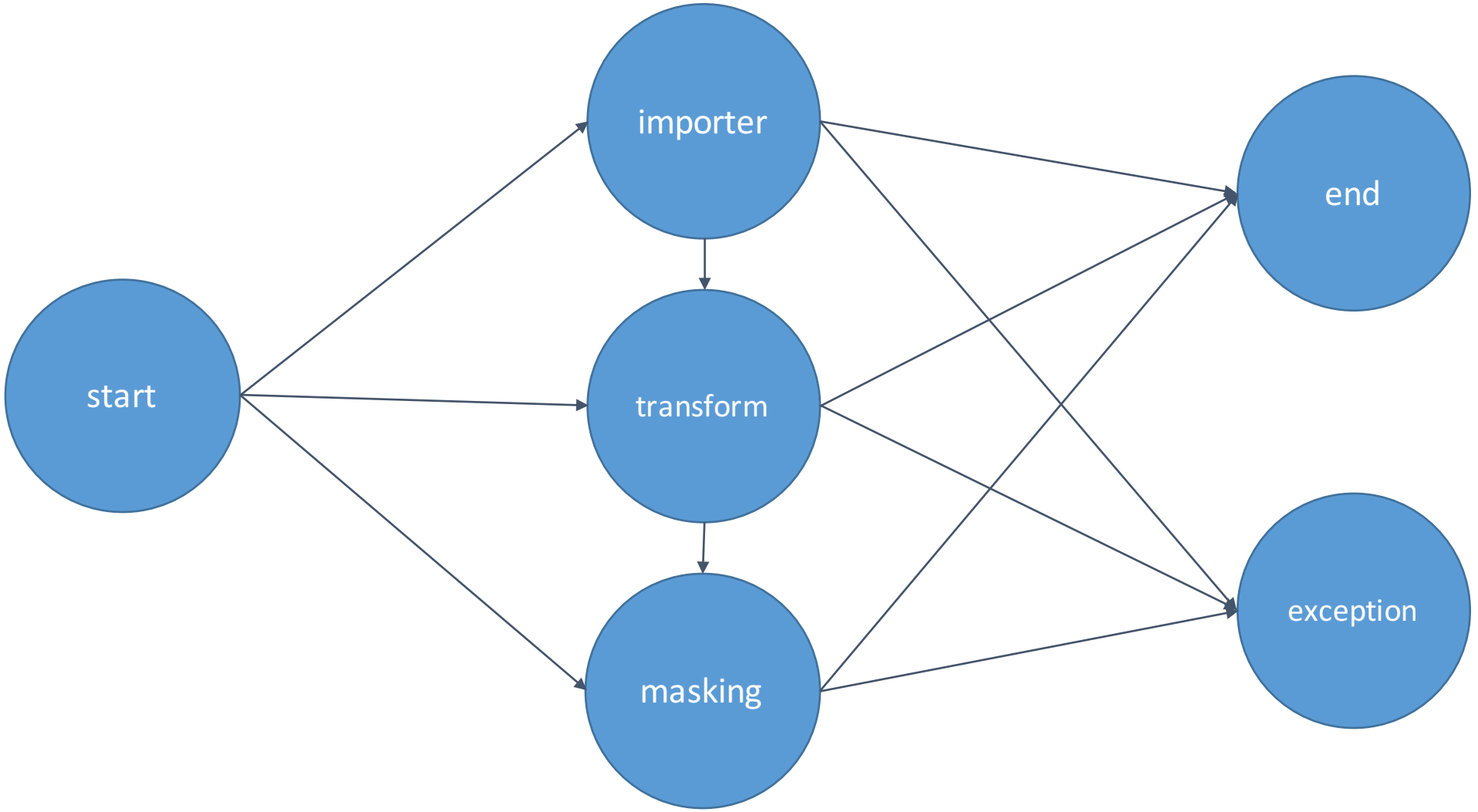
Masking Data Flow Diagram

ATHEMASTER

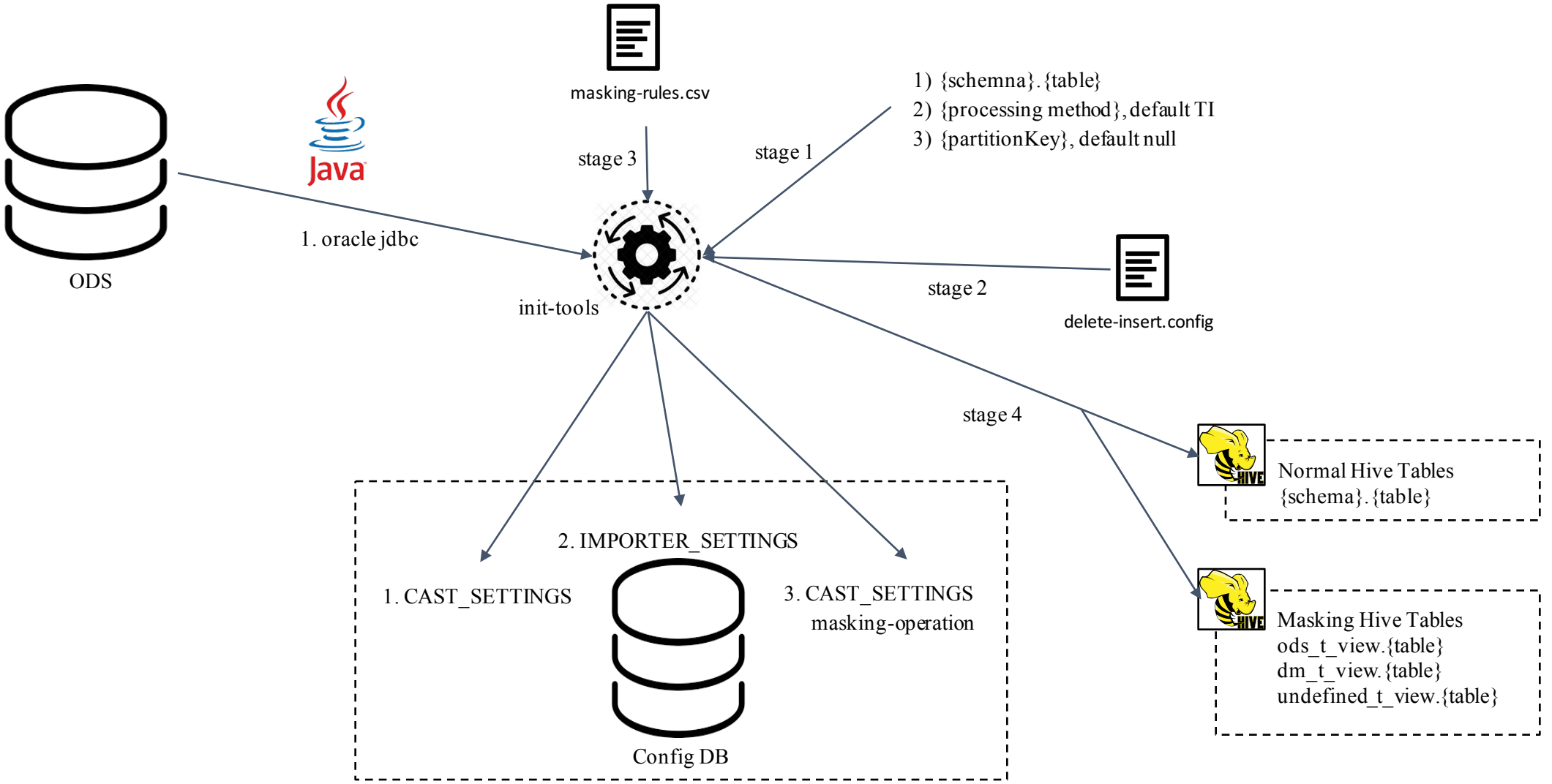


Execution stage and status diagram

ATHEMASTER



Configuration Initial Flow Diagram



BDP_ETL_LOG & EXEC_STATUS

- BDP_ETL_LOG
  - stage: i: importing, t: transforming, m: masking
  - status: r: running, s: succeed, f: failed

CHEDULE_DATE	OWNER	TABLE_NAME	START_D...	END_D...	STAGE	STATUS
UN-19	DM_CC	F_CC_APPL_MA...	20-JUN-19...	20-JUN-...	T	R

- EXEC_STATUS
  - stage: i: importing, t: transforming, m: masking
  - status: 1: running, 0: succeed, -1: failed

	SCHEM...	TABLE_NAME	CREATED_TIME	LAST_...	SNAP_...	STAGE	STATUS
1	DM_CC	F_CC_APPL_MASTER	20-JUN-19 01.42.48....	20-JUN-...	05-JUN-19	M	0

IMPORTER_CONFIG

Column	Description	Example
SCHEMA_NAME	ods schema name	ODS_BANK
TABLE_NAME	ods table name	LMD1V_DAY
LOADING_METHOD	table loading method	DELETE_INSERT
HIVE_PARTITION_KEY	hive partition key	snap_date_str
FETCH_CONDITION	the where clause which is used to fetch data from ODS	snap_date = to_date ('%s','YYYYMMDD')
FETCH_CONDITION_PARAMS	parameters used in fetch_condition	snapdate.value
DROP_CONDITION	the where clause which is used to drop data on HDFS	snap_date_str = %s
DROP_CONDITION_PARAMS	parameters used in drop_condition	snapdate.value
CREATED_TIME	record created time	(omitted)
UPDATED_TIME	record last updated time	(omitted)
ENABLED	is enabled	1

CAST_SETTINGS

ATHEMASTER

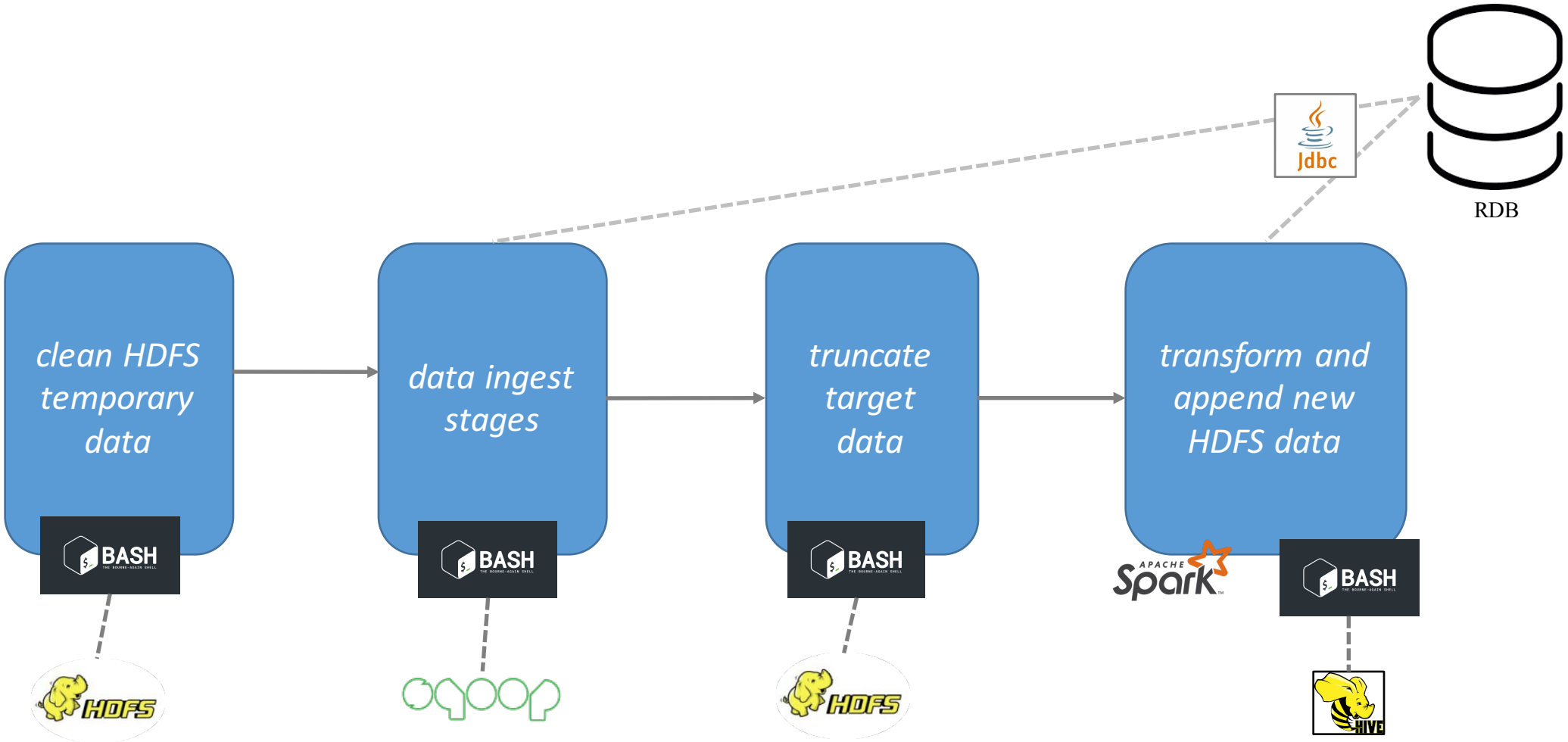
Column	Description	Example
HIVE_DATABASE	hive database name	ods_jcic
HIVE_TABLE_NAME	hive table name	bam061_his
HIVE_COLUMN_NAME	hive column name	cname
HIVE_COLUMN_TYPE	hive column type	string
RDB_TABLE_NAME	related rdb table name	ODS_JCIC
RDB_SCHEMA_NAME	related rdb column name	BAM061_HIS
RDB_COLUMN_NAME	related rdb column name	CNAME
CAST_OPERATION	column type casting operation from rdb to hive	new Column(”cname")
MASK_OPERATION	column masking operation	NAME
SN	serial number	2
ENABLED	is enabled	1

MASK_OPERATION

Column	Description	Example	Example2	Example3
MASK_OPERATION_ID	masking operation id	NAME	BIR	EDU
UDF	associated udf class	MaskNameUdf	MaskBirUdf	MaskNullUdf

the workflow of DI's modules

ATHEMASTER



ATHEMASTER

Operation

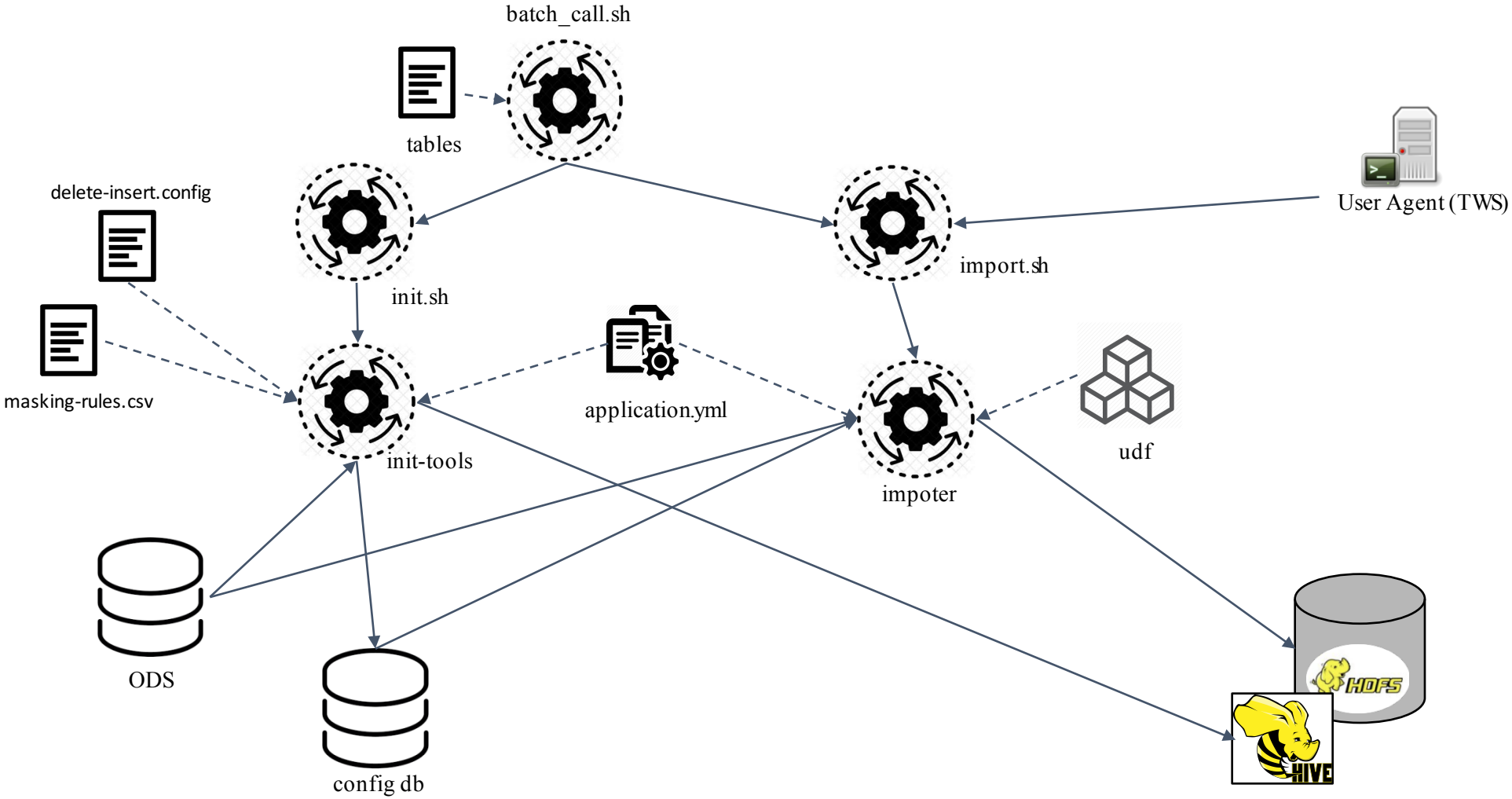
This is DataIngestion module file diagram

- interface/ — the interface used by other application
  - batch/
    - log/ — the execution log
    - batch_import.sh — the batch process of initialing and importing ODS table
    - tables — the table list used for batch_import.sh
  - importer.sh — data importing module interface
  - init.sh — table initial module interface
  - …

Data Ingest module file diagram

- conf/
  - applicationy.ml — login password and environment settings …
  - delete-insert.config — delete insert settings (drop/fetch condition)
  - masking-rule.csv — data masking settings
- importer-1.0-SNAPSHOT.jar — import ods data to hdfs core module
- udf-1.0-SNAPSHOT.jar — masking user defined functions module
- init-tools-1.0-SNAPSHOT.jar — initial table configuration and create hive table core module

Projects/Configurations relation diagram



How to set our configuration

- before we start to learn how to operate the DataIngestion module, we must know how to set these tools' configuration first
  - Sqoop
  - Java spring modules
  - spark-submit command

Protecting password from preying eyes

- Hadoop 2.6.0 provides an API to separate password storage from applications. This API is called the credential provided API and there is a new credential command line tool to manage passwords and their aliases. The passwords are stored with their aliases in a keystore that is password protected. The keystore password can be the provided to a password prompt on the command line, via an environment variable or defaulted to a software defined constant.

Protecting password from preying eyes

- Once the password is stored using the Credential Provider facility and the Hadoop configuration has been suitably updated, that is hadoop.security.credential.provider.path, all applications can optionally use the alias in place of the actual password and at runtime resolve the alias for the password to use.
- Checking Sqoop command usage instruction, you will find Generic Hadoop command-line arguments defined as follow:
  - Generic options supported are

    -D <property=value> use value for given property

    (omitted)

Different types of keystore in Java - JKS

- JKS is Java Keystore, a proprietary keystore type designed for Java. It can be used to store private keys and certificates used for SSL communication, it cannot store secret keys however. The keytool shipped with JDKs cannot extract private keys stored on JKS. Different type of keystore in Java usually are PrivateKey、SecretKey 、JKS、PKCS12、**JCEKS**、DKS and etc …

Protecting password from preying eyes

- First, we generate our keystore by using haddop command -- credential.
- hadoop [--config confdir] credential
  - interact with credential providers Hadoop jar and the required libraries
- Usage: hadoop credential [generic options]
  - [-help]
  - [create <alias> [-value alias-value] [-provider provider-path] [-strict]]
  - [delete <alias> [-f] [-provider provider-path] [-strict]]
  - [list [-provider provider-path] [-strict]]

Protecting password from preying eyes

- Second, create the keystore file with ODS login password for Sqoop to access ODS database.
  - hadoop credential create oracle.pwd.alias -provider jceks://hdfs/user/password/oracle.pwd.jceks

```
[root@quickstart ~]# hadoop credential create oracle.pwd.alias -provider jceks://hdfs/user/password/oracle.pwd.jceks
WARNING: You have accepted the use of the default provider password
by not configuring a password in one of the two following locations:
    * In the environment variable HADOOP_CREDSTORE_PASSWORD
    * In a file referred to by the configuration entry
      hadoop.security.credstore.java-keystore-provider.password-file.
Please review the documentation regarding provider passwords in
the keystore passwords section of the Credential Provider API
Continuing with the default provider password.

Enter alias password:
Enter alias password again:
oracle.pwd.alias has been successfully created.
Provider jceks://hdfs/user/password/oracle.pwd.jceks has been updated.
```

Protecting password from preying eyes

- Third, you can check the keystore by using ‘list’ option
  - hadoop credential list -provider jceks://hdfs/user/password/oracle.pwd.jceks

```
[root@quickstart ~]# hadoop credential list -provider jceks://hdfs/user/password/oracle.pwd.jceks
Listing aliases for CredentialProvider: jceks://hdfs/user/password/oracle.pwd.jceks
oracle.pwd.alias
```

Securely managing passwords in Sqoop

- Sqoop has been enhanced to allow usage of this functionality if it is available in the underlying Hadoop version being used. One new option has been introduced to provide the alias on the command line instead of the actual password (--password-alias). The argument value this option is the alias on the storage associated with the actual password. Example usage is as follows:
  - sqoop import -Dhadoop.security.credential.provider.path=jceks://hdfs/user/password/oracle.pwd.jceks --connect jdbc:oracle:thin:@//172.17.241.246:5211/TSID --username lnk_bdp_sqoop --password-alias oracle.password.alias …

Externalized Configuration

- Spring Boot lets you externalize your configuration so that you can work with the same application code in different environments. You can use properties files, YAML files, environment variables, and command-line arguments to externalize configuration. Property values can be injected directly into your beans by using the @Value annotation, accessed through Spring's Environment abstraction, or be bound to structured objects through @ConfigurationProperties.

Application Property Files

- Spring Application loads properties from application.properties (or application.yml) files in the following locations and adds them to the Spring Environment:
  - A /config subdirectory of the current directory
  - The current directory
  - A classpath /config package
  - The classpath root
- The list is ordered by precedence (properties defined in locations higher in the list override those defined in lower locations). When custom config locations are configured by using spring.config.location, they replace the default locations.

application.yml


```yaml
---

spring:
  profiles: dev_fubon
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    url: jdbc:oracle:thin:@//172.17.241.246:5211/TSID
    username: POOL_AM_WINTHER
    password: 1qaz@WSX
```

application.yml (2)

ATHEMASTER

```yaml
dataingest:
  datasource:
    connection: jdbc:oracle:thin:@//172.17.241.246:5211/TSID
    username: lnk_bdp_sqoop
    password:
  security:
    credential.path: jceks://hdfs/user/password/oracle.pwd.jceks
    password.alias: oracle.pwd.alias

  importer:
    dir:
      tmp: hdfs://nameservice/tmp/di/tmp
      target: hdfs://nameservice/user/hive/warehouse
    file:
      masking-rule: /home/bdpadmin/exec/conf/masking-rule.csv
      delete-insert-config: /home/bdpadmin/exec/conf/delete-insert.config
```

application.yml (3)

spark-submit


```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

spark-submit

- Usage: spark-submit [options] <app jar | python file | R file> [app arguments]
- Options:
  - --master MASTER_URL — spark://host:port, mesos://host:port, yarn, k8s://https://host:port, or local (Default: local[*]).
  - --deploy-mode DEPLOY_MODE — Whether to launch the driver program locally ("client") or on one of the worker machines inside the cluster ("cluster") (Default: client).
  - --name NAME — A name of your application.
  - --jars JARS — Comma-separated list of jars to include on the driver and executor classpaths.
  - --driver-memory MEM — Memory for driver (e.g. 1000M, 2G) (Default: 1024M).
  - --executor-memory MEM — Memory per executor (e.g. 1000M, 2G) (Default: 1G).
  - --num-executors NUM — Number of executors to launch (Default: 2). If dynamic allocation is enabled, the initial number of executors will be at least NUM.

Master URLs

- local — Run Spark locally with one worker thread (i.e. no parallelism at all).
- local[*] — Run Spark locally with as many worker threads as logical cores on your machine.
- yarn — Connect to a YARN cluster in client or cluster mode depending on the value of --deploy-mode. The cluster location will be found based on the HADOOP_CONF_DIR or YARN_CONF_DIR variable.

Difference between yarn client and cluster

- There are two deploy modes that can be used to launch Spark applications on YARN. In cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application. In client mode, the driver runs in the client process, and the application master is only used for requesting resources from YARN.
- Unlike other cluster managers supported by Spark in which the master's address is specified in the --master parameter, in YARN mode the ResourceManager's address is picked up from the Hadoop configuration. Thus, the --master parameter is yarn.

importer help

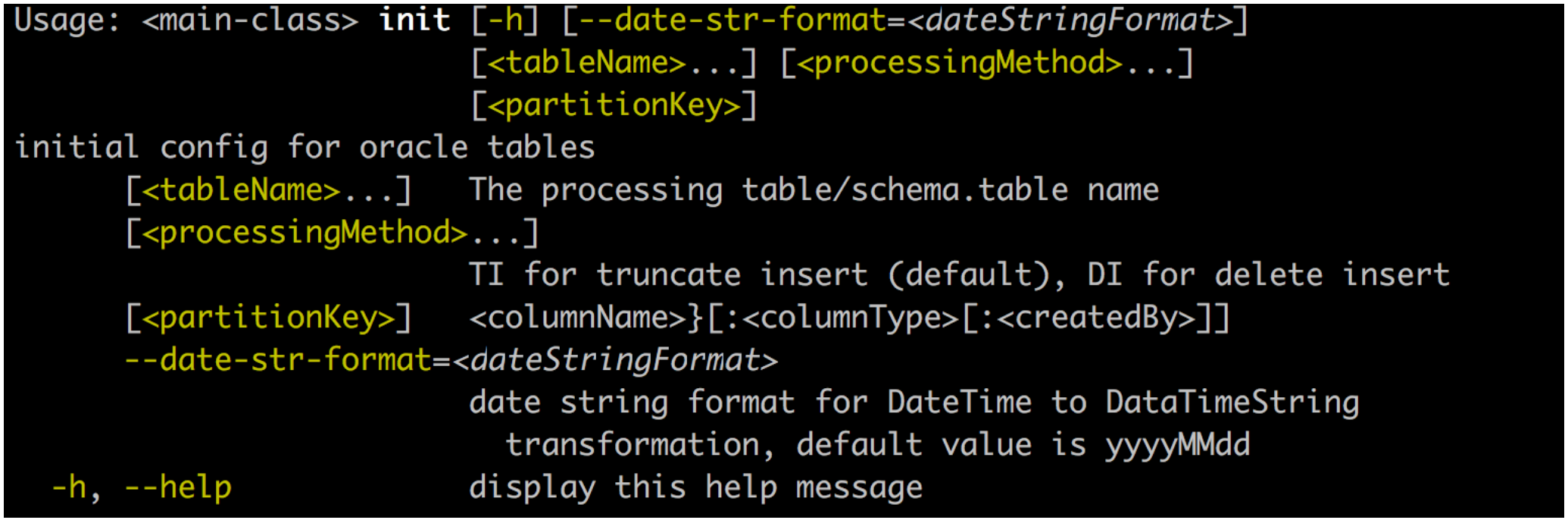

```
Usage: <main class> [-h] [--import-only] [--init] [--regen-masking-data]
                    [--skip-download] [--fetch-size=<fetch_size>]
                    [-m=<num_mapper>] [-P=<String=String>]... [table_name]
                    [sanp_date]
      [table_name]           the [owner.]table to be processed
      [sanp_date]            data event date
      --fetch-size=<fetch_size>
                             set number 'n' of rows to fetch from the database when
                               more rows are needed
      --import-only
      --init                 first import of data, especially for delete insert
                               loading method
      --regen-masking-data   regenerate masking data from target folder
      --skip-download        data ingest without importing data from rdb
  -h, --help                 display this help message
  -m, --num-mappers=<num_mapper>
                             use 'n' map tasks to import in parallel
  -P, --parameters=<String=String>
                             user-defined parameters
```

init-tools and init-masking help


```
Usage: <main class> [-h] [COMMAND]
  -h, --help   display this help message
Commands:
  init          initial config for oracle tables
  init-masking  initial masking table config for oracle tables
```

```
Usage: <main-class> init-masking [-h] [<tableNames>...]
initial masking table config for oracle tables
      [<tableNames>...]   masking table name
  -h, --help              display this help message
```

init-tools init help



hands on practices

- import a defined truncated-insert (loading method) table from ODS
- import a defined deleted-insert table from ODS (partial data)
- import a defined deleted-insert table from ODS (first time/all data)
- import a undefined truncated-insert table from ODS
- import a undefined deleted-insert table from ODS
- insert/update/remove masking rules
- …

Program Architecture

Data Ingestion Projects

- importer
- init-tools
- udf
- interface

What is Maven?

- At first glance Maven can appear to be many things, but in a nutshell Maven is an attempt *to apply patterns to a project's build infrastructure in order to promote comprehension and productivity by providing a clear path in the use of best practices*. Maven is essentially a project management and comprehension tool and as such provides a way to help with managing:
  - Builds
  - Documentation
  - Reporting
  - Dependencies
  - SCMs
  - Releases
  - Distribution

Maven's Objectives

- Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal, there are several areas of concern that Maven attempts to deal with:
  - Making the build process easy
  - Providing a uniform build system
  - Providing quality project information
  - Providing guidelines for best practices development
  - Allowing transparent migration to new features

How do I create a JAR

- Making a JAR file is straight forward enough and can be accomplished by executing the following command:
  - mvn package
- If you take a look at the POM for your project you will notice the packaging element is set to jar. This is how Maven knows to produce a JAR file from the above command. You can now take a look in the ${basedir}/target directory and you will see the generated JAR file.

Integrated development environment

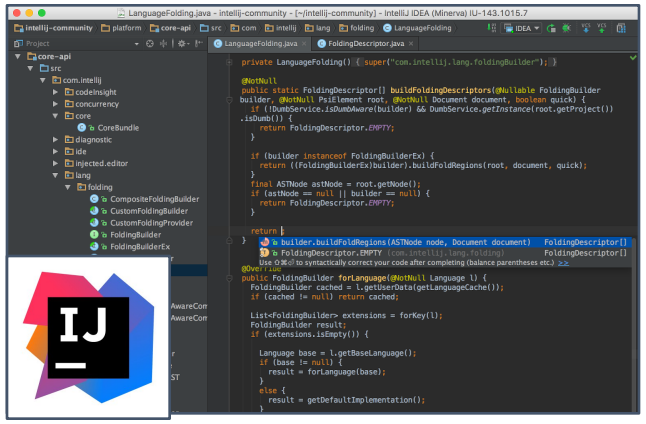
- An **integrated development environment** (**IDE**) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools, and a debugger. Some IDEs, such as NetBeans and Eclipse, contain the necessary compiler, interpreter, or both; others, such as SharpDevelop and Lazarus, do not.
- The boundary between an IDE and other parts of the broader software development environment is not well-defined; sometimes a version control system or various tools to simplify the construction of a graphical user interface (GUI) are integrated. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram for use in object-oriented software development.

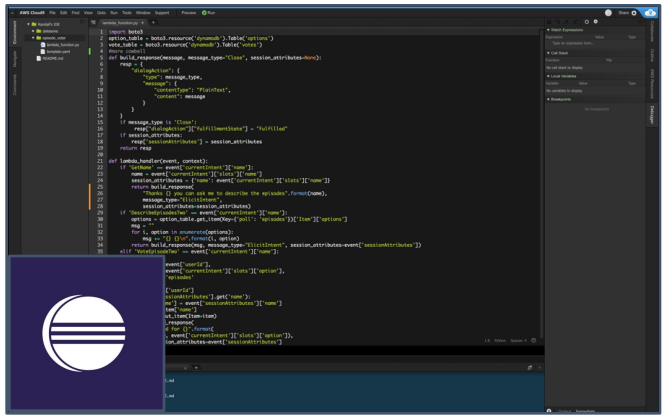
the Best Java IDEs in 2018

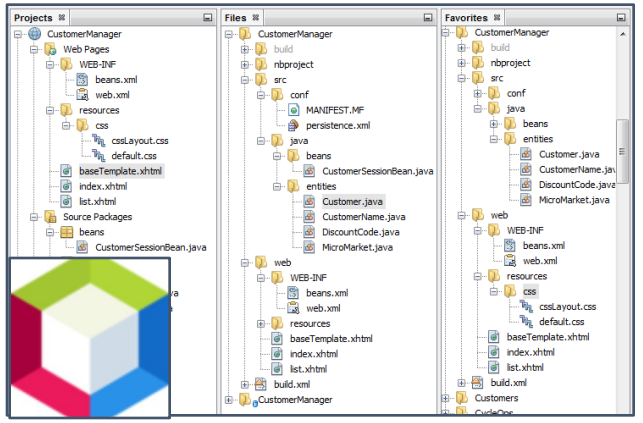
- IntelliJ IDEA
- Vendor: JetBrains

- **Eclipse**
- Vendor: Eclipse Foundation

- **NetBeans**
- Vendor: NetBeans







Import a Maven project into IDE

- On the main menu, select **File | Open**.
- In the dialog that opens, select the pom.xml of the project you want to import. Click **OK**.
- On the first page of the **Import Project** wizard, in the **Import Project from External model** select **Maven** and click **Next**.(This page is not displayed if you selected the pom.xml.)

Import a Maven project into IDE

- IntelliJ IDEA displays the found projects and you can select the ones you need to import.
  Click **Next**.
- Specify the project's SDK and click **Next**.
- Specify a name and the location of your project.
  Click **Finish**.

importer Project file tree diagram

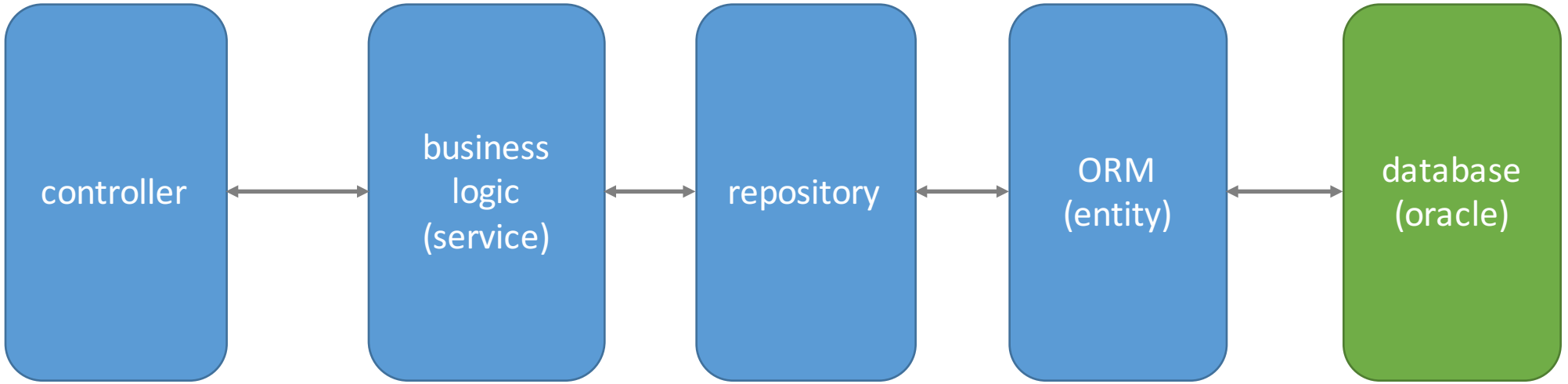
- src/main/resource/
  - application.yml
- src/main/java/
  - bean/
  - entity/
  - enums/
  - …

importer Project file tree diagram

- src/main/java/
  - controller/
  - service/
  - repository/
  - utility/
  - aspect/
  - ImporterApplication.java
- pom.xml

Server side flow diagram

ATHEMASTER



init-tools Project file tree diagram

- src/main/resource/
  - application.yml
  - delete-insert.config
  - masking-rules.csv
  - init.sql
- src/main/java/
  - entity/
  - bean/
  - enum/

init-tools Project file tree diagram

- src/main/java/
  - controller/
  - service/
  - repository/
  - utility/
  - command/
  - InitApplication.java
- pom.xml

UDF Project file tree diagram

- src/main/java/
  - MaskAddrUdf.java
  - MaskBirUdf.java
  - MaskIpUdf.java
  - MaskMailUdf.java
  - MaskNameUdf.java
  - MaskNullUdf.java
  - MaskTellUdf.java

Interface Project file tree diagram

- batch/
  - batch_call.sh
  - tables
- importer.sh
- init.sh

ATHEMASTER

Q & A

ATHEMASTER

THANK YOU

- THE PATHFINDER TO OPENSOURCE -

www.athemaster.com
info@athemaster.com
FB:athemaster