# Branching Policy

[Edit](https://github.com/Prifender/anvizent/wiki/Branching-Policy/_edit)[New Page](https://github.com/Prifender/anvizent/wiki/_new)

Konstantin Komissarchik edited this page on Mar 28 · [4 revisions](https://github.com/Prifender/anvizent/wiki/Branching-Policy/_history)

This describes the branching policy for the shared Anvizent/Prifender Github repository.

# **Master Branch**

Master branch is used for delivery of the current milestone.

# **Feature Branches**

1. For every new feature, such as a new connector, create a new branch from master. Name the branch using the corresponding Jira task name, such as AN-123.
2. Commit to the feature branch as many time as necessary until the feature is done, tested by Anvizent and reviewed by Prifender.
3. Merge the feature branch to master and delete the feature branch

# **Milestone Branches**

1. When a milestone is nearing completion, a milestone branch is created from master using M## naming convention.
2. A bug that's filed against a previously-delivered milestone should be fixed in the milestone branch, then the branch changes should be merged into master. A fix that's needed in both a milestone and master should always be made in the milestone branch first.
3. The milestone branches are never deleted.

# Data Extraction Service Architecture

[Edit](https://github.com/Prifender/anvizent/wiki/Data-Extraction-Service-Architecture/_edit)[New Page](https://github.com/Prifender/anvizent/wiki/_new)

Konstantin Komissarchik edited this page on Feb 13 · [9 revisions](https://github.com/Prifender/anvizent/wiki/Data-Extraction-Service-Architecture/_history)

# **Overview**

The Data Extraction Service (DES) is a component of the Prifender product that handles extracting data from a wide variety of data sources and delivering it for analysis to other Prifender components. The actual data extraction is handled by Talend, an open source ETL framework. The DES is composed of a controller and a set of data extraction nodes.

# **Controller**

This process implements [DES REST Service API](https://github.com/Prifender/anvizent/blob/master/des.api/swagger.yaml) and manages the data extraction nodes. Only one controller is active at a time.

* Must be implemented by extending from the [provided service stub](https://github.com/Prifender/anvizent/tree/master/des.stub)
* Must be a [Spring Boot](https://projects.spring.io/spring-boot/) application
* Must continuously store its state on disk in a JSON file
* Must read a configuration file to get addresses of data extraction nodes

# **Job Compilation**

When the DES controller receives a data extraction job, it will need to compile one or more Talend job that is specific to the target data source. It is expected that when faced with extracting large volume of data, the DES will intelligently break up a single extraction job into multiple Talend jobs to optimize the extraction time.

Adding support for a new data source type will require extending the compiler, but the rest of the DES should not have to change.

[Prioritized List of Data Source Types](https://github.com/Prifender/anvizent/wiki/Data-Source-Types)

# **Encryption of Credentials**

The credentials that DES stores for connection profiles must be encrypted with a key provided to DES. The mechanism for how the key is to be provided is TBD.

# **Data Extraction Node**

The data extraction node receives Talend job requests from the Controller and executes them. It delivers the extracted data to the messaging service. One or more data extraction nodes would be active at any given time, depending on customer data volume and available hardware.

# **Transformation**

After a data row is extracted, one or more transformations to attribute values may need to be applied, as specified in the job definition. Out of the box, DES will come with a set of transformation functions and custom functions can be added to a running system through DES API.

A transformation function is [implemented in Java](https://github.com/Prifender/anvizent/blob/master/des.transform.api/src/main/java/com/prifender/des/transform/Transformation.java). Standard functions are located in [the provided module](https://github.com/Prifender/anvizent/tree/master/des.transform.standard) and are loaded using Java ServiceLoader. See unit tests for loading and usage examples.

# **Messaging Service**

The data read by the data extraction jobs should be delivered to a messaging service, directly from the Talend node doing the extraction. The provided messaging.api module should be used for this purpose, rather than assuming a particular messaging service implementation. Each job should deliver to a separate queue, named after the job id. Each extracted data row gets it's own message, encoded in JSON.

For instance, if a job is scanning a table and extracting data from columns FirstName and LastName, each message written to the corresponding queue should be of the following format:

{

"FirstName": "Joe",

"LastName": "Somebody"

}

# **Docker**

Prifender is deployed using Docker Swarm. DES must be packaged as two docker images. One for controller and one for the data extraction nodes.

# **Resiliency and High Availability**

In case of a VM failure, Docker Swarm will restart the lost containers on another VM. DES must be written such that it can resume operation after the lost containers are recovered.

If the controller is lost, it must be able to resume operation without loosing track of the previously-active jobs or other state.

If a data extraction node is lost, it is acceptable if extraction resumes from the beginning of a chunk or a collection. That is, duplicate data rows are acceptable in case of a VM failure and recovery if there is no efficient method for avoiding duplicates.

# **Future Direction**

None of the following is in scope for M1.

* DES controller should be able to automatically scale up and scale down the number of data extraction nodes (docker containers) that are active based on instruction provided by Prifender central controller.
* We may want to switch from JSON to Apache Avro for message format to optimize perf and on disk footprint. Avro would also make it easier to handle binary data, such as when we scan a file system full of image files.
* DES will need to store configuration and state in a resilient repository (not a JSON file). What repository will be used is yet to be decided. Could be Apache ZooKeeper. Could be something else.
* The Data Extraction Service (DES) implemented the below structured adapters(simplified,relational,hierarchical)
* Struc
* SQL Server
* MySQL
* MongoDB
* Couchbase
* SAP HR (ECC 6.0)
* Hive
* Teradata
* HBase
* HortonWorks2.2
* Azure Data Warehouse
* Sales Force
* DB2
* PostgreSQL
* Cassandra
* SharePoint
* RedShift
* Kafka
* Active Directory
* File System
* Sybase
* Cloudera
* Oracle Enterprise Suite
* Elastic Search 5.X
* Sap - Other Modules
* Sap Hana
* Workday
* CosmosDB
* Google BigQuery
* Net Suite
* DynamoDB
* Informix
* Responsys CRM
* People X
* FieldGlass
* Azure DataLake
* The Data Extraction Service (DES) implemented the below unstructured unstructured.
* 1).box
* 2).google drive.
* 3).unbox.
* 4)confluence wiki.
* 5).imap.
* 6).sharepoint.

# **Infrastructure Requirements**

## **Work Queueing**

At times, DES may get flooded all at once by extraction job requests. It needs to implement a first-in/first-out queue for keeping track of work to be done. When a data extraction job is compiled, the list of Talend jobs that must be executed is added to the queue. When capacity is available, a Talend job is removed from the queue and executed.

In the initial single-process implementation of DES, a configuration parameter should be used to set the max number of threads. When Remote Execution of Extraction is implemented, this will be dynamically controlled by adding and removing data extraction nodes/containers.

The provided messaging service API should be used to implement the work queue. This will ensure that the work queue survives a failure of the DES controller process.

## **Chunking**

When given an extraction job request, DES should evaluate whether to execute it as a single chunk or multiple chunk. This evaluation should be based on metrics such as the number of records that need to be extracted. DES should not expect the administrator to specify chunking configuration. Each chunk is a separate Talend job. All Talend jobs that comprise a data extraction job are added to the scheduling queue when the job is compiled.

## **Remote Execution of Extraction**

DES must be scalable in throughput beyond what a single machine can provide. This is to be accomplished by performing data extraction in a pool of extraction nodes that get work from the DES controller.

An extraction node should be a Spring Boot application that takes the URL of the DES controller as a configuration parameter. Upon startup, the extraction node connects to the controller and asks the controller for a Talend job to run. The controller then marks the job as given to a particular extraction node.

An extraction node only runs one job at a time (single threaded). Running multiple extraction threads on a single VM will be accomplished through Docker Swarm (multiple extraction node containers on a single VM).

The extraction node is expected to contact the controller at set intervals with progress. If the controller has not heard from an extraction node in a while, it assumes that the extraction node has failed and returns the failed Talend job to the end of the work queue.

A Talend job can fail either because the extraction node goes offline or because of error in database connectivity. A failed Talend job is given a set number of retries (each time going back to the end of the work queue). If all retries are exhausted, the entire data extraction job is marked as failed and all of the associated Talend jobs are removed from the work queue. Note that it is acceptable to produce duplicate rows onto the message bus due to failure retries.

The REST API that the extraction nodes use to communicate with the controller should be separate (separate port?) from the main DES API. There should be a separate swagger file, etc.

## **Applying transforms per data extraction spec**

## **Supporting addition of custom transform functions via DES API**

# **Prifender**

1. Nimrod Luria (CEO)
2. Christopher Glover (CTO)
3. Konstantin Komissarchik (Director of Engineering)
4. Michael Jonas (Core)
5. Eugene Kondratenko (Core)
6. Ramesh Natarajan (Core)
7. Sergey Zinkevych (Core)
8. Oleg Andreyev (UI Lead)
9. Eugene Dementev (UI)
10. Oleksandr Vasiliev (DevOps)
11. Artur Mkrtychian (QA Lead)
12. Andrey Ivantsiv (QA)
13. Natallia Linda (QA)
14. Vladimir Vishnevsky (QA)

# **Anvizent**

Raj Koneru (CEO) Raju N (Director)

Sudheer Yekkaladevi ( Onsite Lead) Edukondalu Battula ( Technical Manager)

Satish N (IT Team Lead) Vamsikrishna Boinpally ( IT Team )

Rajesh Anthari ( Core Lead) Mahender Alaveni ( Core) Sri Ravali Samudrala (Core) Bharath HG (Core) Usha Rani (Core)

Archana Laddagiri (ETL Lead) Prince Kumar ( ETL Team) Arun Kumar Sappa ( ETL Team)

Narayanrao M ( QA Lead) Sundeep Gangadhara ( QA ) Krishna Chirukuri (QA) Navya Ranga (QA)

# Test Schema : Hierarchical

[Edit](https://github.com/Prifender/anvizent/wiki/Test-Schema-%3A-Hierarchical/_edit)[New Page](https://github.com/Prifender/anvizent/wiki/_new)

Konstantin Komissarchik edited this page on Mar 29 · [5 revisions](https://github.com/Prifender/anvizent/wiki/Test-Schema-:-Hierarchical/_history)

The hierarchical test schema is to be used for data sources that are capable of storing nested content, such as JSON, XML, Avro, MongoDB, Couchbase, etc.

{

"id" : 1234,

"first\_name" : "Joe",

"last\_name" : "Somebody",

"ssn" : "555-55-4444",

"email" : "joe@somebody.net",

"hire\_date" : "2017-01-17",

"dob" : "1988-11-12",

"nationality" : "USA",

"salary" : 54200,

"commission\_pct" : 15,

"manager\_id" : 7357,

"addresses" :

[

{

"street" : "23423 153rd Ave NE",

"city" : "Redmond",

"region" : "WA",

"postal\_code" : "97334",

"country" : "USA"

},

{

"street" : "11444 2nd Ave NE",

"city" : "Woodinville",

"region" : "WA",

"postal\_code" : "98115",

"country" : "USA"

}

],

"phone\_numbers" : [ "206-555-3322", "425-777-8388", "720-883-2299" ],

"pay\_checks" :

[

{

"payment\_date" : "2016-01-01",

"amount" : 253.12

},

{

"payment\_date" : "2016-02-01",

"amount" : 253.13

},

{

"payment\_date" : "2016-03-01",

"amount" : 253.14

}

]

}

# Test Schema : Relational

[Edit](https://github.com/Prifender/anvizent/wiki/Test-Schema-%3A-Relational/_edit)[New Page](https://github.com/Prifender/anvizent/wiki/_new)

Konstantin Komissarchik edited this page on Mar 29 · [4 revisions](https://github.com/Prifender/anvizent/wiki/Test-Schema-:-Relational/_history)

The relational test schema is to be used for data sources that are capable of storing multiple collections of flat records with foreign key relations, such as Oracle, MySql and SQL Server.

Employees

{

id : int (pkey)

first\_name : string

last\_name : string

ssn : string

email : string

hire\_date : date

dob : date

nationality : string

salary : decimal

commission\_pct : decimal

manager\_id : int

}

Addresses

{

id : int (pkey)

street : string

city : string

region : string

postal\_code : string

country : string

}

EmployeeToAddress

{

employee\_id : int (pkey)

address\_id : int (pkey)

}

PhoneNumbers

{

number : string (pkey)

employee\_id : int (pkey)

}

PayChecks

{

id : int (pkey)

employee\_id : int

payment\_date : date

amount : decimal

}

# Test Schema : Simplified

[Edit](https://github.com/Prifender/anvizent/wiki/Test-Schema-%3A-Simplified/_edit)[New Page](https://github.com/Prifender/anvizent/wiki/_new)

Konstantin Komissarchik edited this page on Mar 29 · [5 revisions](https://github.com/Prifender/anvizent/wiki/Test-Schema-:-Simplified/_history)

The simplified test schema is to be used for data sources that are a single collection of flat records, such as a CSV file.

Employees

{

id : int

first\_name : string

last\_name : string

ssn : string

phone\_mobile : string

phone\_home : string

phone\_office : string

email : string

addr\_street : string

addr\_city : string

addr\_region : string

addr\_postal\_code : string

addr\_country : string

hire\_date : date

dob : date

nationality : string

salary : decimal

commission\_pct : decimal

manager\_id : int

}