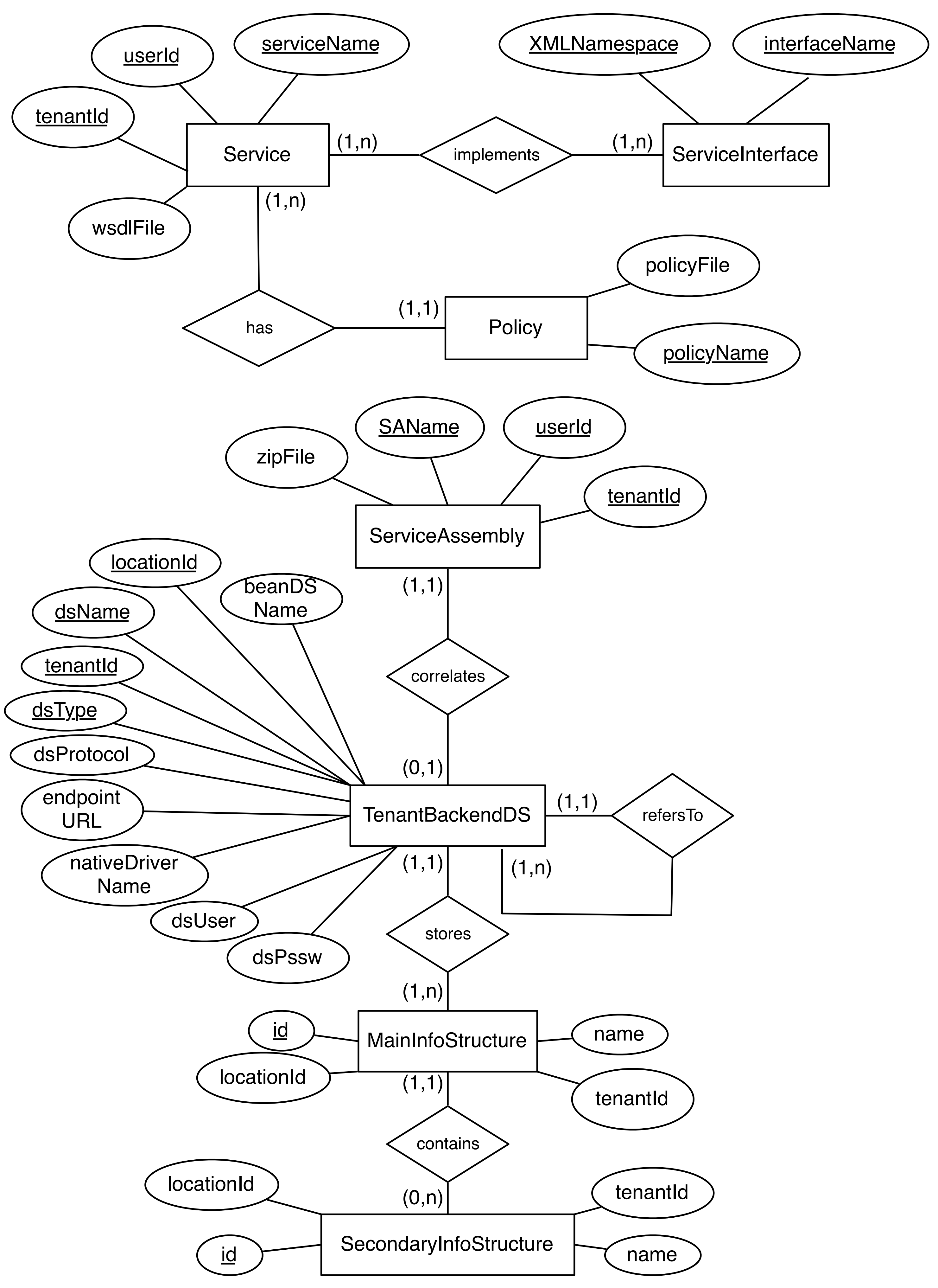

Each database (sql or nosql) contains one main information structure and one secondary information structure. Let's say:

- SQL: in the sql case, we have in one target endpoint, one main info structure which is the database and one or more secondary info structures which are the tables. In this case, we don't need a secondary info structure entity because we have one main info structure per endpoint. Not like in the nosql which we describe after. The database name is stored in the dsName attribute of the TenantBackendDS, and the table in the mainSrcId and mainTargetId in of the MainInfoStructure entity.
In this case we dont need a relationship to a secondary info structure. We have the data we need.

- NoSql: here we have some variations in terms of the names depending on the different providers, but at the end it is the same NoSql concept, in terms of nosql family classification. In this part we have different main info structure names for different providers, e.g. tableId for Amazon DynamoDB, bucketId for for Google Cloud Storage, domainId for the Amazon SimpleDB, collection for Apache Couch DB or Mongo DB, or column for Cassandra. These three main info structures describe at the end the same concept, they store a secondary information structure. In this case one tenant may have one or more main information structures in the same endpoint, e.g. 10 buckets in google cloud storage endpoint google_tenant_1.com, and inside those buckets cero to n objects. So we can set an arbitrary name (but it must be an unique id) for dsName in the TenantBackendDS because we dont need to use this, but we must concentrate in the main and secondary info structure. To differentiate which kind of datasource we are dealing with, we have the field dsType. This could have the format, e.g. family-mainInfoStructure-subInfoStructure-version, e.g. sql-database-table-x.x.x for the sql case. For the nosql case it could be, e.g. nosqlKeyValue-table-item-x.x.x for amazon dynamodb, nosqlKeyValue-bucket-object-x.x.x for google cloud storage, nosqlDocument-collection-document-x.x.x for Apache Couch DB or Mongo DB, and nosqlColumn-column-subcolumn-x.x.x for Cassandra. I've put some sql and nosql dbs as example, but this is extensible because we will have always the same concept (We have associated one db in one family): we have one main info structure and one secondary info structure in a database, and the main info structure stores the secondary one.

Both dsName and dsType have to be filled or calculated and filled with the data received from the tenant in the cloud migration tool, as well as the protocol attribute (which can be: mysql | nosql, or whatever you need for transforming) and nativeDriverName (string with the driver name. This is really important for the dynamic creation of connections in the sql approach).

the other properties in the tenantBackendDS entity are fixed and retrieved from the tenant, and have the values which the tenant provides during the migration process in the cloud migration tool, e.g. tenantId, endpointURL, dsUser, dsPsswd. The beanDSName will be filled by the ESB, as it creates dynamically the datasource connections and saves them in the camel registry.

The MainInformationStructure stores the tenant id proprietary of the structure, and the main src and target info structure identification: table name for sql database, or bucket, table, collection, column for nosql databases.
The SecondaryInformationStructure stores the secondary info structure information. This is used only for the nosql databases, because as said before this is not needed for the sql databases. It stores the tenant id proprietary of the secondary info structure, the id of the pair (auto generated id), and the source and target secondary info structure ids: for item, object, document or subcolumn.

Due to the storage of both front and backend datasources, and main and secondary info structures in the same table, we need to identify which one is the frontend and which one is the backend, at all the levels of storage (which one is the source ds/main info structure/sec info structure and which one is the target). For this requirement we add the locationId field, which can take the values of source | target. An additional id is required in the main an secondary info structure to compose the composite key. This id is an auto increment int.

The field name in the main and secondary information structure cant be used as a key because we are storing in the table information about all the families we support, and one table can have the same name as one entity, but they reference one unique datasource. When storing the data, the reference to the correct unique datasource has to be ensured.

Attribute types and values that can store:

Leyend: Y : needs to be filled by the cloud migration tool and / or JBIMulti2 SA deployment
N : no need to be filled during migration and / or JBIMulti2 SA deployment. Can be filled during routing

1) TenantBackendDS:
- beanDSName = string, can be null. (N)
- locationID = "source" | "target". (Y)
- dsName = string. (Y)
- tenantId = string. (Y)
- dsType = string with the format explained before. (Y)
- dsProtocol = string. Note: we may need to discuss what you need for the transformation between different nosql providers (ones accept rest, others xml in post, others soap). (Y)
- endpointURL: string. (Y). Note: ip/hostName + port
- nativeDriverName = string. (Y) Note: e.g. "com.mysql.jdbc.Driver", "org.postgresql.Driver",...
- dsUser = string. (Y)
- dsPssw = string. (Y)

2) MainInfoStructure
- id: int. (N) . Autoincrement id
- locationID = "source" | "target". (Y)
- name = string. (Y)
- tenantId = string. (Y)

3) SecondaryInfoStructure
- id: int. (N). Autoincrement id
- locationID = "source" | "target". (Y)
- name = string. (Y)
- tenantId = string. (Y)