Механизм выгрузки объектов из внешних источников ETL

ETL

В мире многообразия систем, выполняющих различные задачи, часто возникает задача на основании данных одной или нескольких систем создать объекты (таблицы или сущности) в другой системе. В общем случае за это отвечает механизм `ETL`.

ETL (от англ. `Extract`, `Transform`, `Load` — дословно «извлечение, преобразование, загрузка»). Это системы корпоративного класса, которые применяются, чтобы привести к одним справочникам и загрузить одну систему данные из нескольких разных учетных систем. Т.е. решает задачу **однонаправленного** обмена данными между исходной и целевой системой.

Подробнее можно почитать:

- Wiki
- Введение в ETL
- Habr. Основные функции ETL-систем

NOC ETL

В NOCe реализован базовый функционал ETL - возможность извлекать данные из внешней системы (**`Remote System`**) и на них основе получать объеты в НОКе. На данный момент возможна вгрузка следующих сущностей:

- Зоны ответсвенности (`Administrative Domain`)
- Объекты управления (`Managed Object`)
- Профили объектов (`Manged Object Profile`)
- Сегменты (`Segments`)
- Точки присутствия (`PoP`)
- Профили аутентификации (`Auth profile`)
- Сервисы (`Services`)
- Абоненты (`Subscribers`)
- Линки (`Links`)
- Ресурсные группы (`Resource Group`)

Кратко механизм выглядит так:

1. **Реализуется адаптера выгрузки** (`extractor`). Его задача - получить данные из внешней системы и отдать в виде списка полей, определённых в . Подробнее см главу
2. В интерфейсе настраивается и выбираются реализованные
3. После настройки даётся команда *`./noc etl extract <remote_system_name>`*. Происходит извлечение информации из внешней системе (при помощи адаптера, написанного на шаге 1). Всё складывается в файлы **`import.csv.gz`** в директории */var/lib/noc /import/<remote_system_name>/<loader_name>/import.csv.gz*
4. Командой *`./noc etl check <remote_system_name>`* проверяем целостность выгрузки
5. Командой *`./noc etl diff <remote_system_name>`* смотрим изменения относительно предыдущего файла выгрузки. В первым раз все объекты будут показаны как новые.
6. Командой *`./noc etl load <remote_system_name>`* заливаем данные в НОК (при этом создаются объекты соотв. загрузчику).

После окончания файл **`import.csv.gz`** перемещается в папку */var/lib/noc/import/<remote_system_name>/<loader_name>/archive /import_date.csv.gz* и файл **`mappings.csv`** дополняется связкой: **`ID`** <-> **`ID`** .

> Путь `/var/lib/noc/import` задаётся настройкой `path` -> `etl_import`

Настройка внешней системы

Настройка начинается в пункте меню **`Main`** -> **`Setup`** -> **`Remote Systems`**. После нажатия на кнопку **`Add`** открывается форма создания внешней ситемы с пунктами:

- **`Name`** - имя внешней системы. Будет использоваться при работе с командой `ETL`. Желательно выбирать краткое и без пробелов.
- **`Description`** - описание (какой-нибудь текст)
- **`Handler`** - ссылка на в виде строчки импорта питона.

> *Н-р: **`noc.custom.etl.extractors.zabbix.ZBRemoteSystem`** рассчитывает, что файлик лежит в кастоме по пути* `<custom_folder>/etl/extractor/zabbix.py`

- **`Extractors/Loaders`** - список доступных для моделей для загрузки. **Требует реализацию в адаптере**.
- **`Environment`** - настройки адаптера загрузки (передаются в него при работе)

Также, в **объектах**, поддерживающих создание из механизма ETL присутствуют поля:

- **`Remote System`** - это указание из какой внешней системы приехал объект
- **`Remote ID`** - текстовое поле, ID объекта во внешней системе

Поля **`Remote System`**, **`Remote ID`** заполняются автоматически. Вносить изменения вручную не рекомендуется.

После настройки внешней системы дальнейшая работа идёт с командой *`./noc etl`*. Для лучшего понимания мы начнём рассмотрение с **последнего этапа** - загрузки в НОК.

Загрузка

Загрузка заливка извлечённых данных в НОК выполняется командой ***`./noc etl load REMOTE_SYSTEM_NAME <loadername>`***. Происходит применение изменений по следующем правилам:

- При создании объекта связка **`ID`** : **`ID`** NOC записывается в файл **`mappings.csv`**, расположенным в папке загрузчика.
- при создании **`PoP`** (Point of Presence)по пути создаются объекты типа
- изменения считаются относительно данных загрузки (!не данных в НОКе)
- удалённые объекты управления (**`Managed Object`**) переводятся в состояние **`unmanaged`**
- при удалении сегмента с находящиеся в нём объекты перемещаются в сегмент ALL

> ⚠ Важно понимать, что изменения вычисляются относительно предыдущей загрузки (предыдущего состояния) из внешней системы. По этой причине, если внести изменения по полю в НОКе - загрузка эти изменения не откатит.
> Также, если потерять архивные файлы по последней выгузке, то все объекты будут пересозданы.

Адаптеры для загрузки (загрузчики), ответственные за создание расположены в директории **`core/etl/loader`**. Разберём на примере *`managedobject(core/etl/loader/managedobject.py`*):

ManagedObject Loader

```
class ManagedObjectLoader(BaseLoader):
    """
    Managed Object loader
    """
    name = "managedobject"  #  (Loader Name)
    model = ManagedObject   # NOC object Model (,  )
    fields = [        #  ,
                      #
        "id",
        "name",
        "is_managed",
        "container",
        "administrative_domain",
        "pool",
        "segment",
        "profile",
        "object_profile",
        "static_client_groups",
        "static_service_groups",
        "scheme",
        "address",
        "port",
        "user",
        "password",
        "super_password",
        "snmp_ro",
        "description",
        "auth_profile",
        "tags",
        "tt_system",
        "tt_queue",
        "tt_system_id"
    ]

    mapped_fields = {  #       (loader)
        "administrative_domain": "administrativedomain",
        "object_profile": "managedobjectprofile",
        "segment": "networksegment",
        "container": "container",
        "auth_profile": "authprofile",
        "tt_system": "ttsystem",
        "static_client_groups": "resourcegroup",
        "static_service_groups": "resourcegroup"
    }
```

Структура состоит из аттрибутов:

- **name**. .
- **model** - Указания на **модель**, загрузку которой он реализует
- **fields** - , .
- **mapped_fields** - .

> *Остановимся на карте связей подробнее. Для системы нормально, когда одни сущности связываются с другими. Это позволяет* не мешать всё в одну кучу, *по этой причине и существуют (`mappings map`). В примере указано что поле* ***`object_profile`*** *нео входидо связать с вгрузчиком* ***`managedobjectprofile`***. *Сама привязка идёт по полям ID (всегда первые в списке), а сам вгрузчик ищется по имени:*

Рассмотрим пример загрузчика для ManagedObject Profile

ManagedObjectProfile Loader

```
class ManagedObjectProfileLoader(BaseLoader):
    """
    Managed Object Profile loader
    """
    name = "managedobjectprofile"
    model = ManagedObjectProfile
    fields = [
        "id",
        "name",
        "level"
    ]
```

Как видно, для **managedobjectprofile** достаточно 3 полей: id, , . При этом, (mapped_fields) отсутствует.

При работе с картами связей, необходимо помнить - что не все поля являются обязательными. Н-р для модели **managedobject** обязательными являются: **administrative_domain**, **object_profile**, **segment**

следовательно, для реализаци объектов управления (**ManagedObject**), необходимо будет реализовать выгрузку для **administrativedomain**, **networksegment** и **managedobjectprofile**. Иначе при выполнении команды *./noc etl check* будет множество ошибок вида:

```
[noc.core.etl.loader.base] [RS|managedobject] ERROR: Field #4(administrative_domain) == 'administrativedomain'
refers to non-existent record: 10106,mos-pma-pta-pta1-sw01#10106,True,,administrativedomain,default,!new,
Generic.Host,zb.std.sw,,,2,192.168.3.2,,,,,,,ZB.AUTO,,
[noc.core.etl.loader.base] [RS|managedobject] ERROR: Field #4(administrative_domain) == 'administrativedomain'
refers to non-existent record: 10107,mos-pma-lta-lta1-sw01#10107,True,10107,administrativedomain,default,!new,
Generic.Host,zb.std.sw,,,2,192.168.3.4,,,,,,,ZB.AUTO,,
```

Определение и проверка изменений

Проверка выгруженных данныч выполняется командой *./noc etl check REMOTE_SYSTEM_NAME*. Происходит проверка файла **import.csv** на правильность структуры и связей.
Возможные ошибки:

- Отсутствует объект по ссылке

```
[noc.core.etl.loader.base] [RS|managedobject] ERROR: Field #4(administrative_domain) == 'administrativedomain'
refers to non-existent record: 10106,mos-pma-pta-pta1-sw01#10106,True,,administrativedomain,default,!new,
Generic.Host,zb.std.sw,,,2,192.168.3.2,,,,,,,ZB.AUTO,,
[noc.core.etl.loader.base] [RS|managedobject] ERROR: Field #4(administrative_domain) == 'administrativedomain'
refers to non-existent record: 10107,mos-pma-lta-lta1-sw01#10107,True,,administrativedomain,default,!new,
Generic.Host,zb.std.sw,,,2,192.168.3.4,,,,,,,ZB.AUTO,,
```

Расшифровывается, что поле **administrative_domain** ссылается на несуществующую запись в выгрузке с **administrativedomain** (на это указывает поле из **mapped_fields**) с ID **administrativedomain**

- Неправильный формат загрузки

Команда ***./noc etl diff REMOTE_SYSTEM_NAME <ExtractorNAME>*** позволяет увидеть разницу между последней успешной и текущей загрузками. В построчно формате с указателями:

- **/** - изменение
- **+** - новый объект
- **-** - удаление объекта

```
--- RS.admdiv
--- RS.networksegmentprofile
+ zb.default,zb.default
--- RS.networksegment
+ !new,,,,zb.default
+ !rej,,,,zb.default
+ !tgfake,,tgfake,,zb.default
--- RS.container
+ 10107,ZabbixHost,PoP | Access,,0,60.646729,56.852081,, .  4
--- RS.resourcegroup
--- RS.managedobjectprofile
+ zb.core.sw,zb.core.sw,35
+ zb.std.sw,zb.std.sw,25
--- RS.administrativedomain
+ zb.root,,
--- RS.authprofile
+ ZB.AUTO,ZB.AUTO,,S,,,,,
+ snmp.default,snmp.default,,G,,,,public,
--- RS.ttsystem
--- RS.managedobject
+ 10106,mos-pma-pta-pta1-sw01#10106,True,,zb.root,default,!new,Generic.Host,zb.std.sw,,,2,192.168.3.2,,,,,,,ZB.
AUTO,
+ 10107,mos-pma-lta-lta1-sw01#10107,True,10107,zb.root,default,!new,Generic.Host,zb.std.sw,,,
2,192.168.3.4,,,,,,,ZB.AUTO,
--- RS.link
--- RS.subscriber
--- RS.serviceprofile
--- RS.service
```

Есть дополнительный ключ - `summary` позволяет посмотреть суммарное число изменений:
`./noc etl diff --summary REMOTE_SYSTEM_NAME <ExtractorNAME>`

```
             Loader |      New |  Updated |  Deleted
             admdiv |        0 |        0 |        0
networksegmentprofile |        1 |        0 |        0
      networksegment |        3 |        0 |        0
           container |        1 |        0 |        0
       resourcegroup |        0 |        0 |        0
managedobjectprofile |        2 |        0 |        0
```

Выгрузка (Извлечение)

Выполняется командой *`./noc etl extract REMOTE_SYSTEM_NAME <EXTRACTOR_NAME>`*, где:

- **`REMOTE_SYSTEM_NAME`** - имя внешней системы, указанное на предыдущем шаге
- **`<EXTRACTOR_NAME>`** - опциональное имя модели для загрузки

При этой команде произойдёт подключение к внешней системе, забор информации с неё и формирование файлов **`import.csv`** по пути: **`<etl_path>/remote_system_name/loader_name/`**

Термины

- **Внешняя система** (`Remote System`) - Система, являющаяся источником данных для работы ETL
- **`Extractor`** - Адаптер выгрузки, отвечающий за извлечение информации из , преобразование её к необходимому для работы формату. Итогом работы является файл **`import.csv`**
- **`Loader`** - адаптер загрузки (заливки). На основании файла с выгрузкой (**`import.csv`**) создаёт объекты в НОКе
- **`mappings.csv`** файл привязки между ID (ID НОКа <> ID внешней системы)

Реализация адаптера на примере Zabbix

Для примера, реализуем загрузчик данных из **`Zabbix`**. Перед началом реализации необходимо посмотреть - какую информацию может выдать внешняя система. В этом плане заббиксе позволяет:

- Выгрузить с IP адресами и . На их основе есть возможность создать “Объекты управления” (ManagedObject) и контейнеры с геоинформацией (в Инвентори есть поле **`location`**)
- Выгрузить . На основе групп можно будет проводить фильтрацию объекты по правильным профилям.

Также `Zabbix` поддерживает 2 варианта доступа к данным:

1. API (при помощи WEB API)
2. Прямой доступ к базе (есть возможность выдать прямой доступ к БД `Zabbix`)

Воспользуемся вариантов взаимодействия через `API.  :  Zabbix.`

Для настройки адаптера сохраним файл в одном из 2 мест:

1. **contrib/share/zabbix.py**. В этом случае строчка Handlers будет выглядеть как: noc.contrib.share.zabbix.ZBRemoteSystem.
2. **<custom_path>/etl/extractors/zabbix.py** - в этом случае строчка Handlers в настройках будет выглядеть: noc.custom.etl.extractors.zabbix.ZBRemoteSystem

> ⚠ В каждой папке пути необходимо поместить файл **__init__.py**:

Адаптер требует установленного пакета **py-zabbix**. Для этого в паке NOCa выполняем: **./bin/pip install py-zabbix**

1. В меню **Main** → **Setup** → **Remote Systems** WEB интерфейса добавляем внешнюю систему н-р **ZABBIX**. Отмечаем галочки на реализованных загрузчиках и прописываем в настройках URL, пользователя и пароль:
2. В папке НОКа выдаём команду *`./noc etl extract ZABBIX`*. Ожидаем окончания выгрузки.
3. Командой *`./noc etl check ZABBIX`*. Проверяем отсутствие ошибок.
4. Командой *`./noc etl diff`* `ZABBIX` смотрим выгруженные объекты.
5. *`./noc etl load ZABBIX`* запускает заливку объектов.