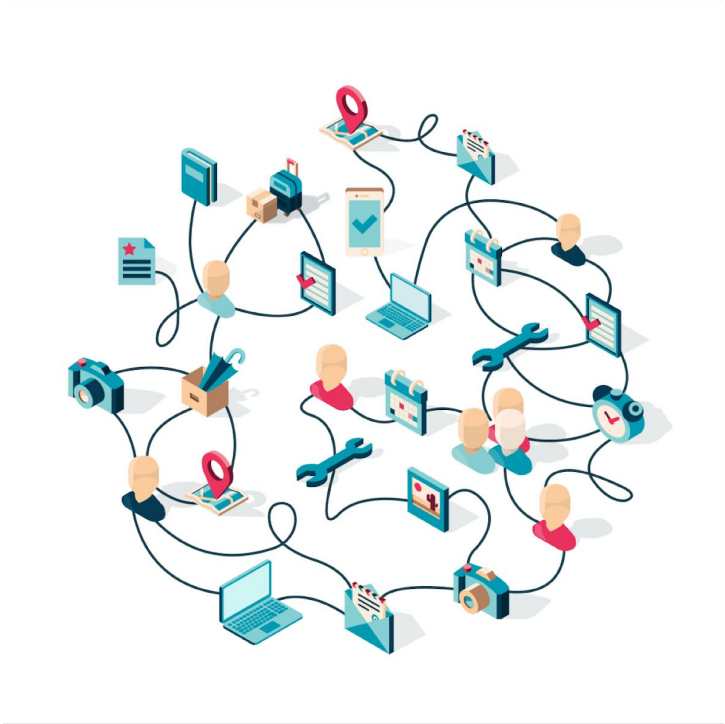
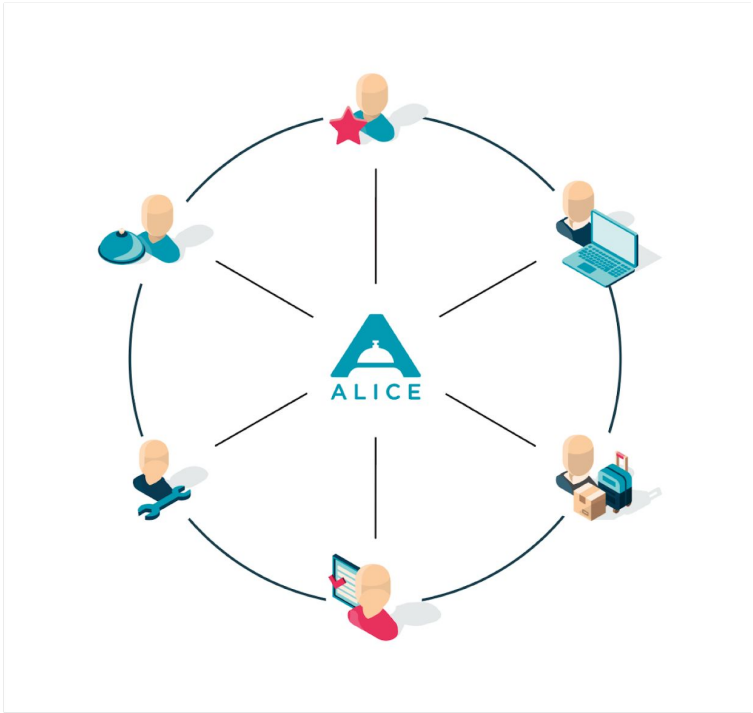
AWS Glue: The True Story of One Transfiguration

Кто такая Alice?



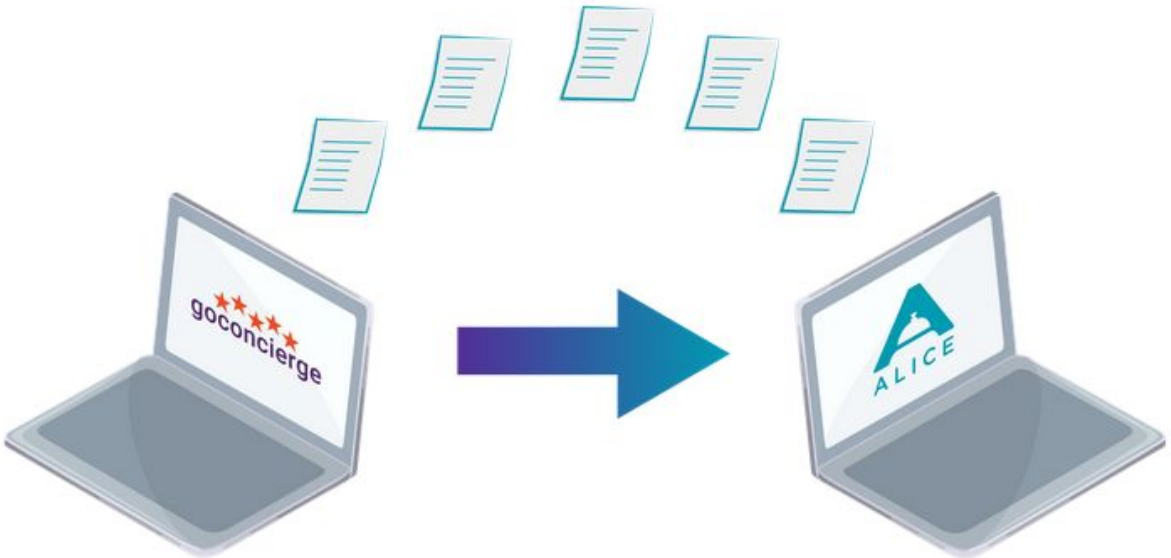
Before ALICE



After ALICE

GoConcierge

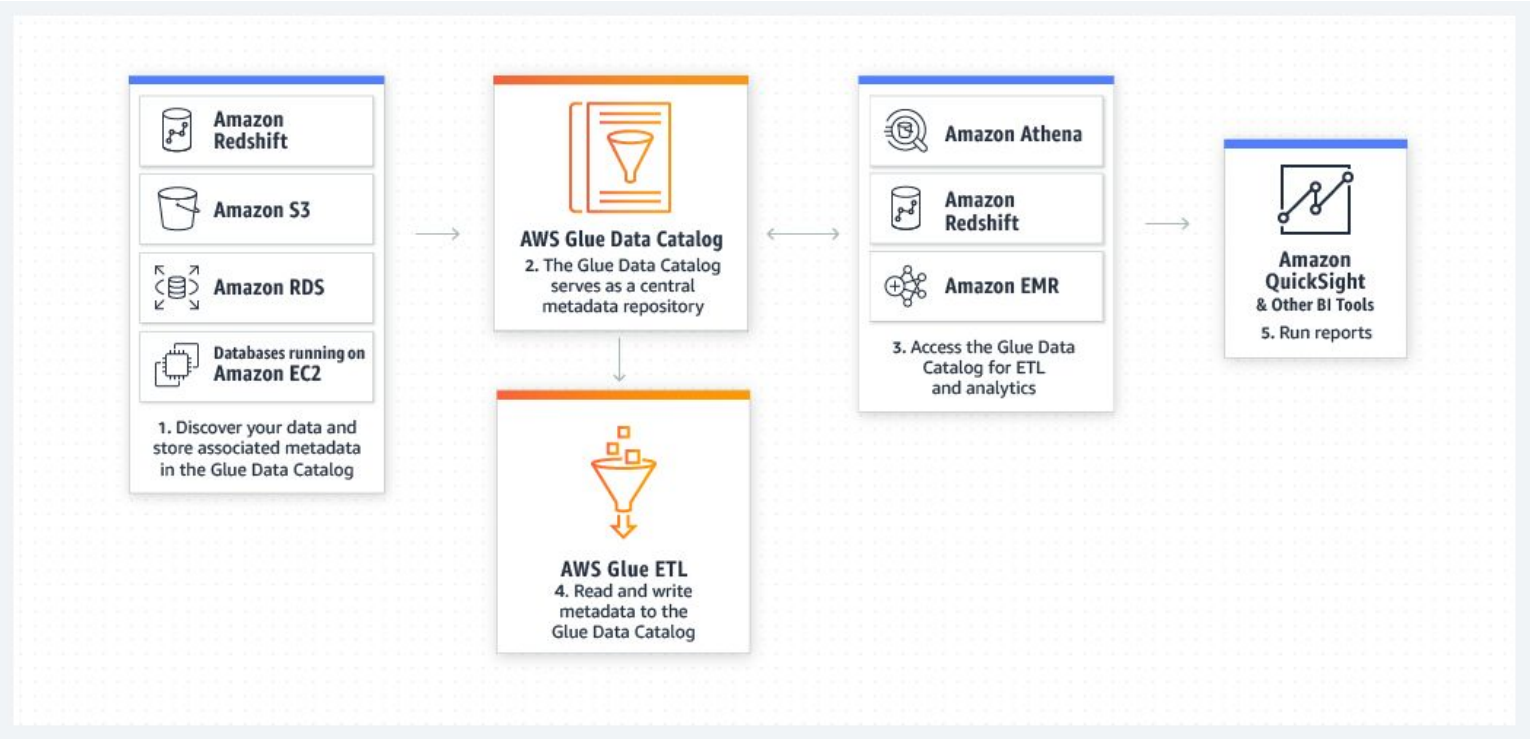
Sit back and relax! Upgrading to ALICE Concierge is a seamless installation. ALICE will import your GoConcierge users, tasks, task detail fields, vendors, and quick links.



Выбираем ETL

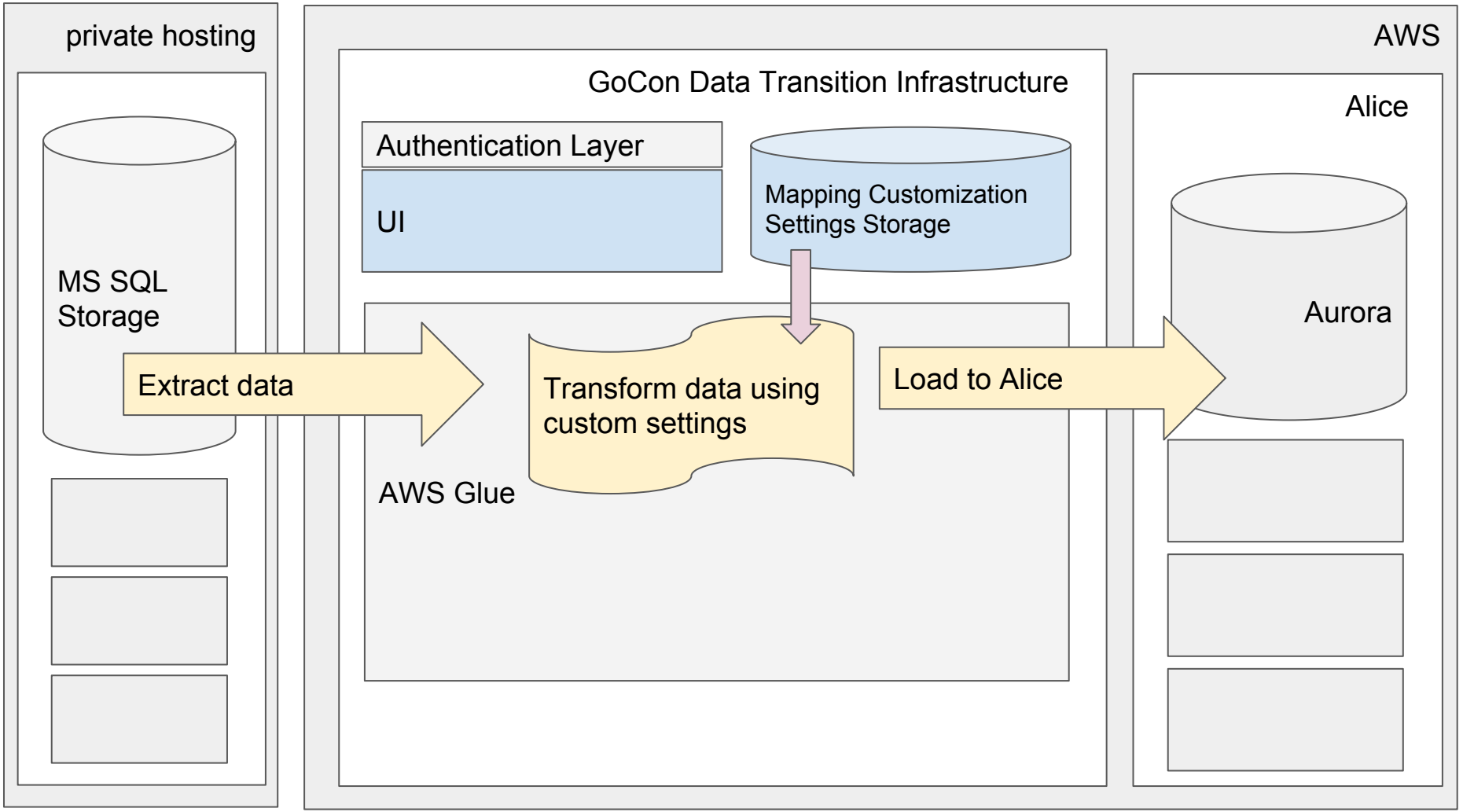
AWS Glue

AWS Glue


Amazon Redshift
Amazon S3
Amazon RDS
Databases running on Amazon EC2
1. Discover your data and store associated metadata in the Glue Data Catalog
AWS Glue Data Catalog
2. The Glue Data Catalog serves as a central metadata repository
AWS Glue ETL
4. Read and write metadata to the Glue Data Catalog
Amazon Athena
Amazon Redshift
Amazon EMR
3. Access the Glue Data Catalog for ETL and analytics
Amazon QuickSight
& Other BI Tools
5. Run reports

Components


Data Catalog
Connections
Tables
Crawlers
Auto-generated jobs
ETL
Jobs
Triggers
Dev endpoints

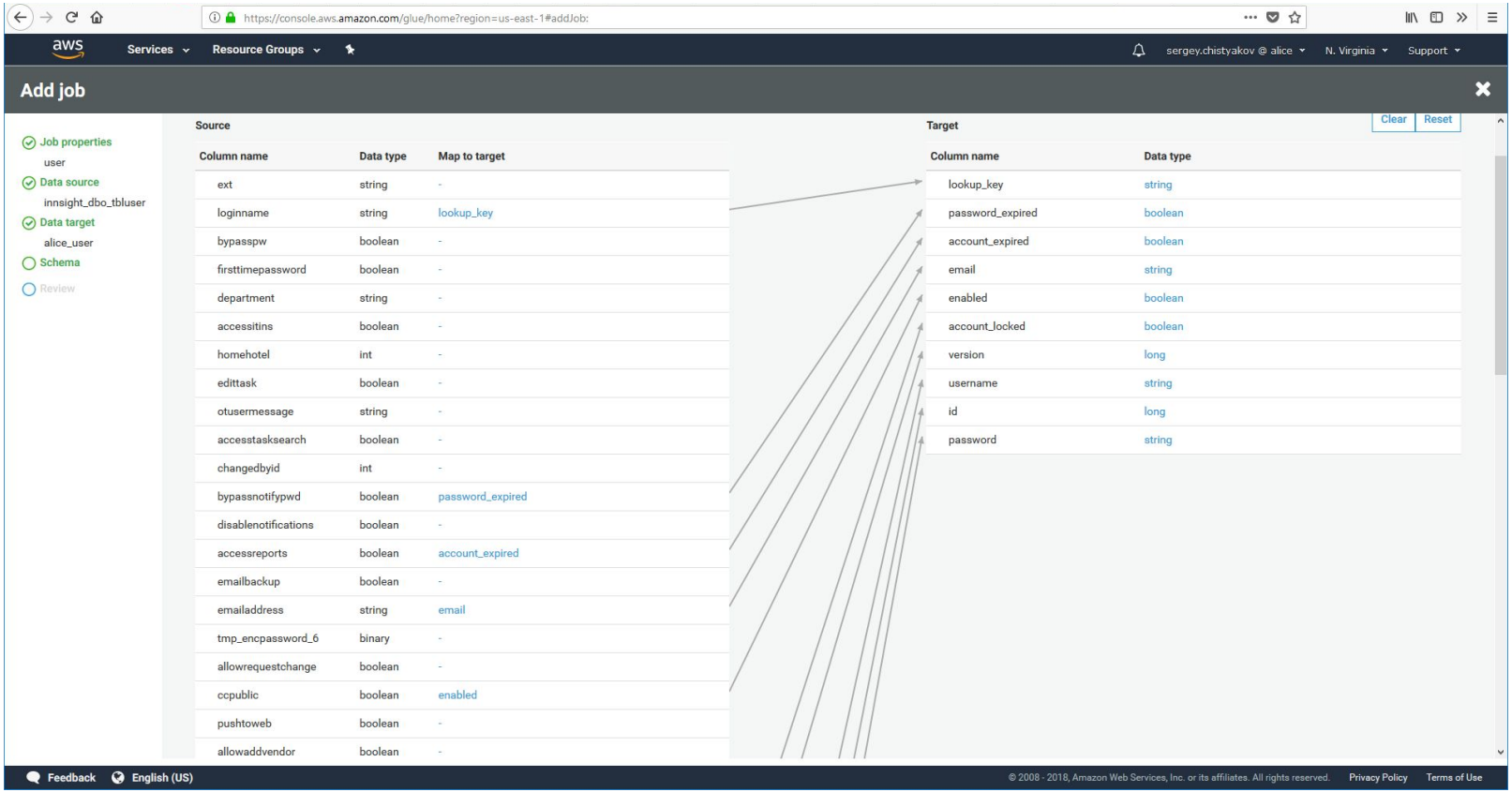

private hosting
AWS
GoCon Data Transition Infrastructure
Alice
Authentication Layer
UI
Mapping Customization Settings Storage
MS SQL Storage
Aurora
Extract data
Transform data using custom settings
Load to Alice
AWS Glue

ROUND 1: Проще не придумаешь

Одна сущность - один Job

Сгенерируем скрипт, а чего не хватает - скопипастим со Stackoverflow

Auto-generated jobs



Auto-generated jobs

```python
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [database = "innsight_replica_glue", table_name = "innsight_dbo_tbluser", transformation_ctx = "datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "innsight_replica_glue", table_name = "innsight_dbo_tbluser", transformation_ctx = "datasource0")
## @type: ApplyMapping
## @args: [mapping = [("loginname", "string", "lookup_key", "string"), ("bypassnotifypwd", "boolean", "password_expired", "boolean"), ("accessreports", "boolean", "account_expired", "boolean"), ("emailaddress", "st
## @return: applymapping1
## @inputs: [frame = datasource0]
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("loginname", "string", "lookup_key", "string"), ("bypassnotifypwd", "boolean", "password_expired", "boolean"), ("accessreports", "boolean", "acco
## @type: SelectFields
## @args: [paths = ["password", "account_expired", "account_locked", "password_expired", "lookup_key", "id", "version", "email", "enabled", "username"], transformation_ctx = "selectfields2"]
## @return: selectfields2
## @inputs: [frame = applymapping1]
selectfields2 = SelectFields.apply(frame = applymapping1, paths = ["password", "account_expired", "account_locked", "password_expired", "lookup_key", "id", "version", "email", "enabled", "username"], transformation
## @type: ResolveChoice
## @args: [choice = "MATCH_CATALOG", database = "alice_glue", table_name = "alice_user", transformation_ctx = "resolvechoice3"]
## @return: resolvechoice3
## @inputs: [frame = selectfields2]
resolvechoice3 = ResolveChoice.apply(frame = selectfields2, choice = "MATCH_CATALOG", database = "alice_glue", table_name = "alice_user", transformation_ctx = "resolvechoice3")
## @type: ResolveChoice
## @args: [choice = "make_cols", transformation_ctx = "resolvechoice4"]
## @return: resolvechoice4
## @inputs: [frame = resolvechoice3]
resolvechoice4 = ResolveChoice.apply(frame = resolvechoice3, choice = "make_cols", transformation_ctx = "resolvechoice4")
## @type: DataSink
## @args: [database = "alice_glue", table_name = "alice_user", transformation_ctx = "datasink5"]
## @return: datasink5
## @inputs: [frame = resolvechoice4]
datasink5 = glueContext.write_dynamic_frame.from_catalog(frame = resolvechoice4, database = "alice_glue", table_name = "alice_user", transformation_ctx = "datasink5")
job.commit()
```

```
alice_workflow_dyf = glueContext.create_dynamic_frame.from_catalog(database = 'alice_glue', table_name = 'alice_workflow')

alice_workflow_df = alice_workflow_dyf.toDF().select(['id', 'name']).withColumnRenamed('id', 'workflow_id')

alice_hotel_dyf = glueContext.create_dynamic_frame.from_catalog(database = 'alice_glue', table_name = 'alice_hotel')

alice_hotel_df = alice_hotel_dyf.toDF().select(['id', 'name']).withColumnRenamed('id', 'hotel_id')

alice_workflow_mapping_df = alice_workflow_df.join(alice_hotel_df, alice_workflow_df.name == alice_hotel_df.name)\

   .select(alice_workflow_df.workflow_id, alice_hotel_df.hotel_id)\

   .withColumn('version', lit(0))\

   .withColumn('deleted', lit(0))

alice_workflow_mapping_dyf = DynamicFrame.fromDF(alice_workflow_mapping_df, glueContext,"nested")

datasink = glueContext.write_dynamic_frame.from_catalog(frame = alice_workflow_mapping_dyf,

                                        database = "alice_glue",

                                        table_name = "alice_workflow_mapping",

                                        transformation_ctx = "datasink5")

job.commit()
```

Warning!

1. Долгий разогрев
2. Проблема связи между сущностями:

   Mapping tables?

   Нет однозначного соответствия между таблицами

   - Одной GC таблице соответствует несколько в Alice и наоборот
   - Joins мультиплицируют данные
   - Сложно не запутаться

   Lookup keys

   EVENT_ID-APPOINTMENT_ID-WHAT_EVER_YOU_WANT_ID

```
# ------------ hotel section ---------------

def add_hotel_lookup_key(self, dxf):

    return self.add_generic_lookup_key(dxf, ["CompanyID"])



def add_alice_hotel_reference(self, dxf):

    return self.add_generic_intermediate_reference(dxf, "alice_hotel", "hotel_id", ["CompanyID"])
```

ROUND 2: Сложнее, чем казалось

Python libraries

One job - many runs

Warning!

1. Очень долгие транформации
2. Другие проблемы
    a. Long warm up
    b. Lazy filtering
    c. Credentials
    d. Mapping functions

ROUND 3: препроцессинг

JDBC + SQL

Оркестрация

CI/CD

Validation tests

```
join_df = spark.read \
    .format("jdbc") \
    .option("url", jdbc_url) \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .option("user", db_user) \
    .option("password", db_password) \
    .option("dbtable", query_string) \
    .load()
```

?